# INSIGHTS FROM DEEP REPRESENTATIONS FOR MACHINE LEARNING SYSTEMS AND HUMAN COLLABORATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Maithra Raghu

August 2020

INSIGHTS FROM DEEP REPRESENTATIONS FOR MACHINE LEARNING SYSTEMS AND HUMAN COLLABORATIONS

Maithra Raghu, Ph.D.

Cornell University 2020

Over the past several years, we have witnessed fundamental breakthroughs in machine learning, largely driven by rapid advancements of the underlying deep neural network models and algorithms. This has consequently spurred the development of new, powerful machine learning systems, with emerging applications in specialized, high-stakes domains such as medicine. However, the increased capabilities of these novel systems come at a cost of much greater complexity, with the design of machine learning systems becoming ever more laborious, computationally expensive and opaque. This can result in catastrophic failures and significantly hinders effective collaboration with human experts, often central to successful deployment. In this thesis, we present research results that take steps to addressing these challenges. Having first overviewed some of the key deep learning models, algorithms and use cases, we begin by introducing quantitative techniques that can give insights into neural network hidden representations, which provide both fundamental understanding on central aspects of machine learning, and also inform algorithms for efficiently learning and training these complex systems. Finally, we study how these fully trained AI systems can be adapted to work effectively with human experts, resulting in better outcomes than either humans or AI alone. We conclude with a discussion on the many rich further directions and open questions for future study.

## BIOGRAPHICAL SKETCH

Maithra Raghu is a Computer Science PhD Candidate at Cornell University. Her research centers on developing quantitative tools to gain insights into deep neural network representations, and using these insights to inform better design and training of machine learning systems, as well as investigations on how these systems can effectively collaborate with human experts. Her work has been published in conferences such as NeurIPS, ICML, ICLR, WWW and has also been covered by many press outlets including The Washington Post, WIRED and Quanta Magazine. She has been named one of the Forbes 30 Under 30 in Science and a Rising Star in EECS. Prior to her PhD, Maithra received her BA and Masters in Mathematics with First Class Honors at the University of Cambridge.

## ACKNOWLEDGEMENTS

This PhD journey and its rich experiences, from exploring central scientific questions, to participating/organizing conferences, to the numerous friendships made, would not have been possible without the guidance and support of many people.

Firstly, I would like to thank my close collaborators Chiyuan Zhang, Katy Blumer, Ben Poole, Justin Gilmer, Alex Irpan, Ari Morcos and Jacob Andreas. Very grateful to all of you for both our exciting joint projects and many fruitful research discussions! I would also like to thank my early collaborators Tamas Sarlos, Ravi Kumar and Andrew Tomkins who introduced me to aspects of data science and machine learning, and to Surya Ganguli and Jascha Sohl-Dickstein who cemented my interests in deep learning.

The process of learning how to effectively explore open directions, assimilate key ideas from prior work, ask critical questions and formulate new problems is a highly rewarding one but can also be a tumultuous experience. In the multiple years it takes to become adept at all of these skills, there is at least one point in time where many things may be failing all at once! During times like those, as well as other critical points during the PhD, I've been very fortunate to have the support of many mentor figures who provided advice and encouragement. Thank you to Jason Yosinski, David Sontag, Aleksander Madry, Peter Norvig, Alex Smola, Fernanda Viegas, Martin Wattenberg, Diane Greene, Quoc Le, Oriol Vinyals, Johan Ugander, Edith Cohen, Chris Manning, Jure Leskovec, Jitendra Malik, Jack Po, Zak Stone, Nati Srebro, Sanjeev Arora, Bobby Kleinberg, Kilian Weinberger and of course Jon Kleinberg for your time and inspiration!

In many of my projects, I've been fortunate to collaborate with researchers at

Google, which added a huge amount of depth (no deep learning puns intended) to my perspectives on identifying and defining problems. I'm extremely grateful to Quoc Le, Samy Bengio, Jeff Dean and Deanna Chen for all the support on the Google side, and Becky Stewart, Bobby Kleinberg, Kilian Weinberger and Jon Kleinberg for advice and support on the Cornell side. These projects wouldn't have been possible without you!

Saving most important till last, I want to extend my deepest gratitude to my closest mentors and collaborators in this journey. Thank you Sendhil Mullainathan for advice, encouragement, and vision on the creative ways machine learning can be used across many domains. Thank you Dave Donoho for all of your guidance — I am always in admiration of your incredible insights on *both* theoretical and empirical approaches to understanding deep learning. The NAS colloquium we co-organized will remain one of my most exciting and memorable graduate school experiences! Thank you Eric Schmidt, for the encouragement, inspiration and enriching perspectives on the advances in this field and how they apply in much broader contexts. Formulating and writing our survey paper has been a wonderfully rewarding experience! Thank you to Geoff Hinton, for always listening to new ideas with such an open mind, and the encouraging, pragmatic feedback — some of these discussions were pivotal in informing my research directions! Your creativity and integrity never ceases to inspire!

A huge thank you to Samy Bengio, who has been a second advisor to me. Our numerous research discussions and the resulting projects were central to my graduate school experience, especially in informing my knowledge on the breadth of techniques in the field, and how they might interface with the questions at hand. And even beyond research collaborations, I am especially grateful for your advice and friendship through the past several years! Hoping to keep

up with all of this going forwards.

And finally, it is hard to express how grateful I am to Jon Kleinberg. From guidance and on-point feedback when exploring new research directions, to encouragement when things weren't working, to articulate insights on research and the scientific pursuit as a whole. Our conversations and many years of collaboration has been foundational not only to my research but my core beliefs as a scientist. I am also always in awe of your perceptiveness on so many topics, be they technical, societal or any hybrid of the two. Thank you so much for your time, camaraderie, and wisdom during this journey, and I look forward to working together more in this next stage!

On the professional *and* personal side, I am very fortunate to have many wonderful women in STEM peers. Meera Ramaswamy, Anna Goldie, Azalia Mirhoseni, Arathi Mani, Pooja Sethi, Sarah Tan, Chelsea Finn, Katy Blumer, Kate Niehaus and Ramya Ramakrishnan — you have been my inspirations, friends and mentors through this journey. Thank you for everything!

Finally, a huge thank you to my parents for all of their love and support, my brother Aniruddh Raghu for being both an inspirational sibling and a co-author (hoping we write more papers together), and my better half Arun Chaganty for all of your love, advice and encouragement — wouldn't have made it to the end without you!

TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# Part I

# Introduction and Background

# Overview

CHAPTER 1

# INTRODUCTION

The past several years have witnessed incredible breakthroughs in many core problems in machine learning, ranging from speech recognition [98, 233], to complex tasks in computer vision [168, 116, 335], to central questions in natural language, such as machine translation, question answering and dialogue [256, 239, 325, 270]. These breakthroughs have been largely driven by advances in deep learning, with the underlying deep neural network models seeing significant development in *competition benchmarks* [58, 256, 331].

Instigated by this progress, over just the past *few* years, there has been substantial interest in applying advances in areas such as computer vision and natural language to specialized, often high-stakes domains such as medicine and healthcare. Importantly, these applications consist of both research studies as well as full *deployment* [316, 101, 255, 359, 10, 174].

However, to gain the full potential of these machine learning systems, and utilise them in a safe, reliable and sustainable fashion, it is important to have well designed models and algorithms, which are tailored to the requirements of these specialized, high-stakes domains. Unfortunately, with the increase in capabilities of these systems and the tasks they can address, also comes an increase in the *complexity* of designing the system. Indeed, designing a new machine learning system requires consists of making numerous specific choices on the type of neural network model [168, 325, 117, 116], ways to preprocess, curate and augment the data [372, 54, 363, 61], the type of optimization method and regularization to apply [137, 154, 87] and the learning algorithm and related

hyperparameters [130, 367, 368].

The opacity in knowing which set of choices will lead to a robust, reliable model poses significant challenges for potential (deployed) use-cases. For example, a recent study [346] on a device using a deep learning model for hip fracture detection found that the presence of surgical skin markings on the patient (evidence of prior treatment for skin cancer), had a catastrophic effect on the device, with an enormous false positive rate. Another recent paper [15] performed a validation study on a deep neural network trained to perform hip fracture detection. When presented with a new validation dataset (from a different hospital), the model failed to generalize. While the cause of this was at first unknown, a careful analysis of the internal representations of the system revealed that the choice of model and training process was leading it to overfit to confounding attributes in the training data, causing a failure in generalization. Needless to say, such failures cannot be afforded in these critical applications and it is vital to develop techniques that enable their preemptive identification and resolution.

These failures also have direct ramifications for another important consideration in deploying and using these systems — the ability to work effectively with human experts. In specialized applications such as medicine, vital functions such as navigating patient preferences, complex surgical interventions and palliative care are performed by human experts. Such functions are not automatable, and hence any deployed AI system must effectively support the human experts in these roles.

These considerations lead us to the main research results presented in this thesis. Having surveyed many of the key neural network models, tasks, al-

gorithms and end-to-end design process in Chapter 2, we begin in Part II by developing quantitative techniques that can give us insights into the internals of these complex machine learning systems, and reveal important properties in central components of the design process. These results have been published in [253, 220], with many followup results from the community [95, 278, 164, 169, 25]. In Part III we then use these techniques and insights to inform the design of new algorithms for *efficiently* learning and training these systems, with the corresponding papers being [254, 250]. Finally, in Part IV, we examine how fully trained AI systems can effectively collaborate with human experts [252, 251].

More specifically, we described earlier the numerous intricate choices in designing a robust machine learning system, and the challenges presented by the opacity of the process. In practice, to identify good design choices for the end to end system, numerous such design choices are tried out [51, 308, 382], resulting in a *population* of deep learning systems, which are monitored through accuracy and loss metrics, typically all the way through the training process. However, these accuracy and loss metrics primarily give us insights on only the output (top layer) of the underlying neural network model, not the hidden representations, which contain the bulk of the parameters and learning. In Part II, we address this challenge by proposing an algorithm that can quantitatively analyze these hidden representations. We use this algorithm to study many central components of the design process, from understanding how hidden representations converge through training, to model compression to properties of systems that generalize well, and differences across varying types of neural network architectures.

Using these algorithms and informed by these insights, in Part III, we turn to studying methods to train machine learning systems in a data efficient fashion.

In many specialized applications, access to enough training data can be a significant bottleneck, as it requires human experts such as doctors to help with data curation and labelling. We study two directions which can significantly help data efficiency — few-shot learning and transfer learning. In few-shot learning, the goal is to be able to learn many tasks, but with very limited data for each task. Using the techniques introduced in Part II, we propose a new algorithm that unifies and significantly simplifies existing approaches, and can be much more easily scaled for use with e.g. large scale medical data. We then turn to studying transfer learning applied to medical imaging, investigating the effectiveness of standard neural network architectures for this new data type, as well as localizing feature reuse and even identifying feature independent effects of transfer learning. These results inform new, hybrid approaches to transfer learning that better enable design to suit needs of the domain.

Finally, in Part IV, we take fully trained AI systems in these specialized domains (particularly medicine) and look into how these systems can work effectively with human experts. One concrete question to study is how to combine human expert effort in (say) diagnosing conditions with automated outputs from machine learning systems. A successful combination relies on a new prediction problem — training the machine learning system to successfully detect human error, or in the medical setting, doctor disagreements. Having designed a system that can do this, we then use these predictions to guide the combination of human expert knowledge and AI system predictions, with the hybrid outperforming either of the entities used in isolation. This raises important points on novel ways to evaluate our AI system when designing for deployment.

We conclude the thesis by summarizing the challenges, the presented results,

and discussing many rich open directions for future exploration in Part V.

CHAPTER 2

# BACKGROUND ON DEEP LEARNING

We begin by first surveying key deep learning models, algorithms and tasks to put both central challenges and the research results aimed at mitigating them into broader context. In particular, this survey chapter is structured as follows:

- We overview highly diverse set of deep learning concepts, from deep neural network models for varied data modalities (CNNs for visual data, graph neural networks, RNNs and Transformers for sequential data) to the many different key tasks (image segmentation, super-resolution, sequence to sequence mappings and many others) to the multiple ways of training deep learning systems.
- But the explanation of these techniques is relatively high level and concise, to ensure the core ideas are accessible to a broad audience, and so that the entire survey can be read end to end easily.
- Keeping in mind the goal of specialized applications of these techniques in domains such as medicine, we highlight advances in (i) techniques for interpretability and representation analysis, for going beyond predictive accuracy and gaining insight into the design of the system and (ii) *efficient learning* methods, focusing on using deep learning with less data (self-supervision, semi-supervised learning, and others) .These are two exciting and rapidly developing research areas.
- The survey also focuses on helping quickly ramp up implementation, and in addition to overviews of the entire deep learning design process and a section on implementation tips (Section 2.9), the survey has a plethora of

open-sourced code, research summaries and tutorial references developed by the community throughout the text, including a full section (Section 2.3) dedicated to this.

## 2.1 Chapter Outline

The subsequent sections of this chapter are structured as follows

- Section 2.2 starts with some high level considerations for using deep learning. Specifically, we first discuss some template ways in which deep learning might be applied in scientific domains, followed by a general overview of the entire deep learning design process, and conclude with a brief discussion of other central machine learning techniques that may be better suited to some problems. The first part may be of particular interest to those considering scientific applications, while the latter two parts may be of general interest.
- Section 2.3 provides references to tutorials, open-sourced code model/algorithm implementations, and websites with research paper summaries, all developed by the deep learning community. This section should be very helpful for many readers and we encourage skimming through the links provided.
- Section 2.4 then overviews many of the standard tasks and models in deep learning, covering convolutional networks and their many uses, graph neural networks, sequence models (RNNs, Transformers) and the many associated sequence tasks.

- Section 2.5 looks at some key variants of the supervised learning training process, such as transfer learning, domain adaptation and multitask learning. These are central to many successful applications of deep learning.
- Section 2.6 overviews advances in interpretability and representational analysis, a set of techniques focused on gaining insights into the internals of the end-to-end system: identifying important features in the data, understanding its effect on model outputs and discovering properties of model hidden representations. These are very important for many scientific problems which emphasise understanding over predictive accuracy, and may be of broader interest for e.g. aiding model debugging and preemptively identifying failure modes.
- Section 2.7 considers ways to improve the data efficiency for developing deep neural network models, which has been a rapidly evolving area of research, and a core consideration for many applications, including scientific domains. It covers the many variants of self-supervision and semi-supervised learning, as well as data augmentation and data denoising.
- Section 2.8 provides a brief overview of more advanced deep learning methods, specifically generative modelling and reinforcement learning.
- Section 2.9 concludes with some key implementation tips when putting together an end-to-end deep learning system, which we encourage a quick read through!

## 2.2 High Level Considerations for Deep Learning

In this section we first discuss some high level considerations for deep learning techniques. We start with overviews of template ways in which deep learning might be applied in scientific settings, followed by a discussion of the end-to-end design process and some brief highlights of alternate machine learning methods which may be more suited to some problems.

### 2.2.1 Templates for Deep Learning in Scientific Settings

What are the general ways in which we might apply deep learning techniques in scientific settings? At a very high level, we can offer a few *templates* of ways in which deep learning might be used in such problems:

(1) *Prediction Problems* Arguably the most straightforward way to apply deep learning is to use it to tackle important *prediction problems*: mapping inputs to predicted outputs. This predictive use case of deep learning is typically how it is also used in core problems in computing and machine learning. For example, the input might be a biopsy image, and the model must output a prediction of whether the imaged tissue shows signs of cancer. We can also think of this predictive use case as getting the model to *learn a target function*, in our example, mapping from input visual features to the cancer/no cancer output. Using deep learning in this way also encapsulates settings where the target function is very complex, with no mathematical closed form or logical set of rules that describe how to go from input to

output. For instance, we might use a deep learning model to (black-box) *simulate* a complex process (e.g. climate modelling), that is very challenging to explicitly model [145].

(2) *From Predictions to Understanding* One fundamental difference between scientific questions and core machine learning problems is the emphasis in the former on *understanding* the underlying mechanisms. Oftentimes, outputting an accurate prediction alone is not enough. Instead, we want to gain interpretable insights into what properties of the data or the data generative process led to the observed prediction or outcome. To gain these kinds of insights, we can turn to interpretability and representation analysis methods in deep learning, which focus on determining how the neural network model makes a specific prediction. There has been significant work on both tools to understand what features of the input are most critical to the output prediction, as well as techniques to directly analyze the *hidden representations* of the neural network models, which can reveal important properties of the underlying data.

(3) *Complex Transformations of Input Data* In many scientific domains, the amount of generated data, particularly visual data (e.g. fluorescence microscopy, spatial sequencing, specimen videos [244, 141]) has grown dramatically, and there is an urgent need for efficient analysis and automated processing. Deep learning techniques, which are capable of many complex transformations of data, can be highly effective for such settings, for example, using a deep neural network based segmentation model to automatically identify the nuclei in images of cells, or a pose estimation system to rapidly label behaviors seen in videos of mice for neuroscience analysis.

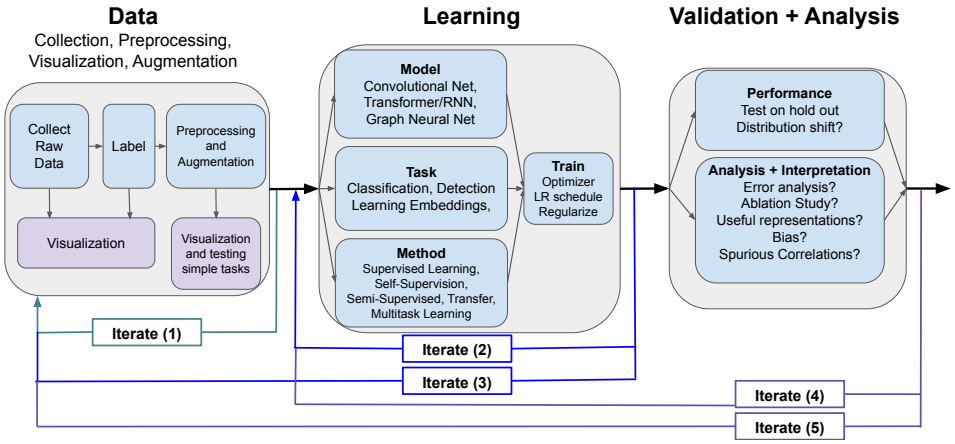



Figure 2.1: **Schematic of a typical deep learning workflow.** A typical development process for deep learning applications can be viewed as consisting of three sequential stages (i) data related steps (ii) the learning component (iii) validation and analysis. Each one of these stages has several substeps and techniques associated with it, also depicted in the figure. In the survey we will overview most techniques in the learning component, as well as some techniques in the data and validation stages. Note that while a natural sequence is to first complete steps in the data stage, followed by learning and then validation, standard development will likely result in multiple different iterations where the techniques used or choices made in one stage are revisited based off of results of a later stage.

### 2.2.2 Deep Learning Workflow

With these examples of templates for deep learning applications in science, we next look at the end to end workflow for designing a deep learning system. Figure 2.1 illustrates what a typical workflow might look like.

Having selected the overarching (predictive) problem of interest, we can broadly think of having three stages for designing and using the deep learning system: (i) *data* related steps, such as collection, labelling, preprocessing, visualization, etc (ii) *learning* focused steps, such as choice of deep neural network model, the task and method used to train the model (iii) *validation and analysis* steps, where performance evaluations are conducted on held out data, as well as analysis and interpretation of hidden representations and ablation studies of the

overall methods.

These three stages are naturally sequential. However, almost all of the time, the first attempt at building an end-to-end deep learning system will result in some kind of failure mode. To address these, it is important to keep in mind the *iterative* nature of the design process, with results from the different stages informing the redesign and rerunning of other stages.

Figure 2.1 provides some examples of common iterations with the backward connecting arrows: (i) the *Iterate (1)* arrow, corresponding to iterations on the data collection process, e.g. having performed some data visualization, the labelling process for the raw instances might require adjusting — the first labelling mechanism might be too noisy, or not capture the objective of interest (ii) the *Iterate (2)* arrow, corresponding to iterations on the learning setup, due to e.g. deciding that a different task or method might be more appropriate, or decomposing the learning process into multiple steps — first performing self-supervision followed by supervised learning (iii) the *Iterate (3)* arrow, changing the data related steps based off of the results of the learning step (iv) the *Iterate (4)* arrow, redesigning the learning process informed by the validation results e.g. finding out the model has overfit on the training data at validation and hence reducing training time or using a simpler model (v) the *Iterate (5)* arrow, adapting the data steps based off the validation/analysis results, e.g. finding that the model is relying on spurious attributes of the data, and improving data collection/curation to mitigate this.

**Focus of Survey and Nomenclature** In this survey, we provide a comprehensive overview of many of the techniques in the *learning stage*, along with some techniques (e.g. data augmentation, interpretability and representation analysis,

Section 2.6) in the data and validation stages.

For the learning stage, we look at popular *models*, *tasks* and *methods*. By *models* (also sometimes referred to as *architecture*), we mean the actual structure of the deep neural network — how many layers, of what type, and how many neurons, etc. By *tasks*, we mean the kind of prediction problem, specifically, the type of input and output. For example, in an *image classification* task, the input consists of images and the output a probability distribution over a (discrete) set of different categories (called classes). By *methods*, we refer to the type of learning process used to train the system. For example, *supervised learning* is a very general learning process, consisting of the neural network being given data instances with corresponding *labels*, with the labels providing supervision.

Unlike different models and tasks, methods can be subsets of other methods. For example, *self-supervision*, a method where the neural network is trained on data instances and labels, but the labels automatically created from the data instance, can also be considered a type of *supervised learning*. This can be a little confusing! But it suffices to keep in mind the general notions of *models*, *tasks* and *methods*.

### 2.2.3 Deep Learning or Not?

As a final note before diving into the different deep learning techniques, when formulating a problem, it is important to consider whether deep learning provides the right set of tools to solve it. The powerful underlying neural network models offer many sophisticated functionalities, such learned complex image

transforms. However, in many settings, deep learning may not be the best technique to start with or best suited to the problem. Below we very briefly overview some of the most ubiquitous machine learning methods, particularly in scientific contexts.

**Dimensionality Reduction and Clustering** In scientific settings, the ultimate goal of data analysis is often *understanding* — identifying the underlying mechanisms that give rise to patterns in the data. When this is the goal, *dimensionality reduction*, and/or *clustering* are simple (unsupervised) but highly effective methods to reveal hidden properties in the data. They are often very useful in the important first step of exploring and visualizing the data (even if more complex methods are applied later.)

*Dimensionality Reduction:* Dimensionality reduction methods are either *linear*, relying on a linear transformation to reduce data dimensionality, or *non-linear*, reducing dimensionality while approximately preserving the non-linear (manifold) structure of the data. Popular linear dimensionality reduction methods include *PCA* and *non-negative matrix factorization*, with some popular non-linear methods including *t-SNE* [199] and *UMAP* [209]. Most dimensionality reduction methods have high-quality implementations in packages like `scikit-learn` or on github, e.g. `https://github.com/oreillymedia/t-SNE-tutorial` or `https://github.com/lmcinnes/umap`.

*Clustering:* Often used in combination with dimensionality reduction, clustering methods provide a powerful, unsupervised way to identify similarities and differences across the data population. Commonly used clustering methods include *k-means* (particularly the *k-means++* variant), *Gaussian Mixture Models*

(GMMs), *hierarchical clustering* and *spectral clustering*. Like dimensionality reduction techniques, these clustering methods have robust implementations in packages like `scikit-learn`.

In Section 2.6.2, we discuss how dimensionality reduction and clustering can be used on the hidden representations of neural networks.

**Linear Regression, Logistic Regression (and variants!)** Arguably the most fundamental techniques for *supervised* problems like classification and regression, linear and logistic regression, and their variants (e.g. Lasso, Ridge Regression) may be particularly useful when there is limited data, and a clear set of (possibly preprocessed) features (such as in tabular data.) These methods also often provide a good way to sanity check the overarching problem formulation, and may be a good starting point to test out a very simple version of the full problem. Due to their simplicity, linear and logistic regression are highly interpretable, and provide straightforward ways to perform *feature attribution*.

**Decision Trees, Random Forests and Gradient Boosting** Another popular class of methods are decision trees, random forests and gradient boosting. These methods can also work with regression/classification tasks, and are well suited to model non-linear relations between the input features and output predictions. Random forests, which ensemble decision trees, can often be preferred to deep learning methods in settings where the data has a low signal-to-noise ratio. These methods can typically be less interpretable than linear/logistic regression, but recent work [226] has looked at developing software libraries `https://github.com/interpretml/interpret` to address this challenge.

**Other Methods and Resources:** Both the aforementioned techniques and many other popular methods such as graphical models, Gaussian processes, Bayesian optimization are overviewed in detail in excellent course notes such as University of Toronto's Machine Learning Course or Stanford's CS229, detailed articles at https://towardsdatascience.com/ and even interactive textbooks such as https://d2l.ai/index.html (called Dive into Deep Learning [369]) and https://github.com/rasbt/python-machine-learning-book-2nd-edition.

## 2.3 Deep Learning Libraries and Resources

A remarkable aspect of advances in deep learning so far is the enormous number of resources developed and shared by the community. These range from tutorials, to overviews of research papers, to open sourced code. Throughout this survey, we will reference some of these materials in the topic specific sections, but we first list here a few general very useful frameworks and resources.

**Software Libraries for Deep Learning:** Arguably the two most popular code libraries for deep learning are PyTorch (with a high level API called Lightning) and TensorFlow (which also offers Keras as a high level API.) Developing and training deep neural network models critically relies on fast, parallelized matrix and tensor operations (sped up through the use of Graphical Processing Units) and performing automatic differentiation for computing gradients and optimization (known as *autodiff*.) Both PyTorch and TensorFlow offer these core utilities, as well as many other functions. Other frameworks include Chainer,

ONNX, MXNET and JAX. Choosing the best framework has been the source of significant debate. For ramping up quickly, programming experiences closest to native Python, and being able to use many existing code repositories, PyTorch (or TensorFlow with the Keras API) may be two of the best choices.

**Tutorials:** (i) https://course.fast.ai/ fast.ai provides a free, coding-first course on the most important deep learning techniques as well as an intuitive and easy to use code library, https://github.com/fastai/fastai, for model design and development. (ii) https://towardsdatascience.com/ contains some fantastic tutorials on almost every deep learning topic imaginable, crowd sourced from many contributors. (iii) Many graduate deep learning courses have excellent videos and lecture notes available online, such as http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/ for Deep Learning and Neural Networks, or the more topic specific Stanford's CS224N NLP with Deep Learning. A nice collection of some of these topic specific lectures is provided at https://github.com/Machine-Learning-Tokyo/AI_Curriculum. There are also some basic interactive deep learning courses online, such as https://github.com/leriomaggio/deep-learning-keras-tensorflow.

**Research Overviews, Code, Discussion:** (i) https://paperswithcode.com/ This excellent site keeps track of new research papers and their corresponding opensourced code, trending directions and displays state of the art results (https://paperswithcode.com/sota) across many standard benchmarks. (ii) Discussion of deep learning research is very active on Twitter. http://www.arxiv-sanity.com/top keeps track of some of the top most discussed pa-

pers and comments. (iii) https://www.reddit.com/r/MachineLearning/ is also a good forum for research and general project discussion. (iv) https://www.paperdigest.org/conference-paper-digest/ contains snippets of all the papers in many different top machine learning conferences. (v) IPAM (Institute for Pure and Applied Mathematics) has a few programs e.g. https://www.ipam.ucla.edu/programs/workshops/new-deep-learning-techniques/?tab=schedule and https://www.ipam.ucla.edu/programs/workshops/deep-learning-and-medical-applications/?tab=schedule with videos overviewing deep learning applications in science.

**Models, Training Code and Pretrained Models:** As we discuss later in the survey, publicly available models, training code and *pretrained* models are very useful for techniques such as transfer learning. There are many good sources of these, here are a few that are especially comprehensive and/or accessible:

(i) Pytorch and TensorFlow have a collection of pretrained models, found at https://github.com/tensorflow/models and https://pytorch.org/docs/stable/torchvision/models.html.

(ii) https://github.com/huggingface Hugging Face (yes, that really is the name), offers a huge collection of both pretrained neural networks and the code used to train them. Particularly impressive is their library of *Transformer* models, a one-stop-shop for sequential or language applications.

(iii) https://github.com/rasbt/deeplearning-models offers many standard neural network architectures, including multilayer perceptrons, convolutional neural networks, GANs and Recurrent Neural Networks.

(iv) https://github.com/hysts/pytorch_image_classification does a deep dive into image classification architectures, with training code, highly popular data augmentation techniques such as *cutout*, and careful speed and accuracy benchmarking. See their page for some object detection architectures also.

(v) https://github.com/openai/baselines provides implementations of many popular RL algorithms.

(vi) https://modelzoo.co/ is a little like paperswithcode, but for models, linking to implementations of neural network architectures for many different standard problems.

(vii) https://github.com/rusty1s/pytorch_geometric. Implementations and paper links for many graph neural network architectures.

**Data Collection, Curation and Labelling Resources:** A crucial step in applying deep learning to a problem is collecting, curating and labelling data. This is a very important, time-intensive and often highly intricate task (e.g. labelling object boundaries in an image for segmentation.) Luckily, there are some resources and libraries to help with this, for example https://github.com/tzutalin/labelImg, https://github.com/wkentaro/labelme, https://rectlabel.com/ for images and https://github.com/doccano/doccano for text/sequential data.

**Visualization, Analysis and Compute Resources:** When training deep neural network models, it is critical to visualize important metrics such as loss and accuracy *while* the model is training. Tensorboard https://

www.tensorflow.org/tensorboard (which works with Pytorch and TensorFlow) is a very popular framework for doing this. Related is the *colab* effort https://colab.research.google.com/notebooks/welcome.ipynb, which, aside from providing a user-friendly, interactive way for model development and analysis (very similar to jupyter notebooks) also provides some (free!) compute resources.

## 2.4 Standard Neural Network Models and Tasks

In this section, we overview the standard *neural network models* and the kinds of *tasks* they can be used for, from convolutional networks for image predictions and transformations to transformer models for sequential data to graph neural networks for chemistry applications.

### 2.4.1 Supervised Learning

Before diving into the details of the different deep neural network models, it is useful to briefly discuss *supervised learning*, the most standard method to *train* these models. In the supervised learning framework, we are given data instances and an associated label for each data instance, i.e. (data instance, label) pairs. For example, the data instances might comprise of chest x-ray images, and the labels (one for each chest x-ray image) a binary yes/no to whether it shows the symptoms of pneumonia. *Training* the neural network model then consists of finding values for its parameters so that when it is fed in a data instance (chest x-

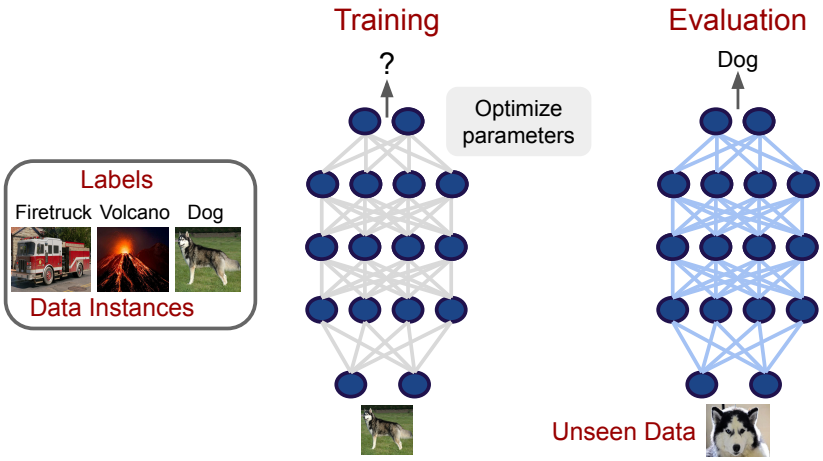



Figure 2.2: **The Supervised Learning process for training neural networks.** The figure illustrates the supervised learning process for neural networks. Data instances (in this case images) and corresponding labels are collected. During the training step, the parameters of the neural network are optimized so that when input a data instance, the neural network outputs the corresponding label. During evaluation, the neural network is given unseen data instances as input, and if trained successfully, will output a meaningful label (prediction).

ray) as input, it correctly outputs the corresponding label (yes/no on whether the chest x-ray has pneumonia.) To find these parameter values, we perform iterative *optimization* to guide the neural network parameters to appropriate values, using the given labels to provide supervision. Figure 2.2 shows a schematic of the supervised learning setup for deep learning.

Supervised learning is the most basic yet most critical method for training deep neural networks. As will be seen through the subsequent sections, there can be significant diversity in the kinds of (data, label) pairs used. Even in settings where clear (data, label) pairs are not possible to collect (Sections 2.7, 2.7.2), the training problem is often reformulated and recast into a supervised learning framework.

### 2.4.2 Multilayer Perceptrons

The first and most basic kind of deep neural network is the *multilayer perceptron.* These models consist of a stack of fully connected layers (matrix multiplications) interleaved with a nonlinear transform.

Despite their simplicity, they are useful for problems where the data might consist of a set of distinct, (possibly categorical) features, for example, tabular data. These models have more expressive power than logistic/linear regression, though those methods would be a good first step to try. One way to apply these models might be to first preprocess the data to compute the distinct set of features likely to be important, and use this as input. `https://github.com/rasbt/deeplearning-models` provides some implementations of some example multilayer perceptron architectures.

**Scientific Examples** One recent scientific example is given by the use of simple MLPs for pharamaceutical formulation [357], developing variants of a drug that is stable and safe for patient use.

### 2.4.3 Convolutional Neural Networks

These are arguably the most well known family of neural networks, and are very useful in working with any kind of image data. They are characterized by having convolutional layers, which allow the neural network to reuse parameters across different spatial locations of an image. This is a highly useful inductive

bias for image data, and helping with efficiently learning good features, some, like Gabor filters, which correspond to traditional computer vision techniques. Convolutional neural networks (CNNs) have so many possible uses that we overview some of the most ubiquitous tasks separately below.

### Image Classification

This is arguably the simplest and most well known application of convolutional neural networks. The model is given an input image, and wants to output a class — one of a (typically) mutually exclusive set of labels for that image. The earlier example, of mapping a chest x-ray image to a binary disease label, is precisely image classification.

Convolutional neural networks for image classification is an extremely common application of deep learning. There many different types of CNN models for classification: *VGG* — a simple stack of convolutional layers followed by a fully connected layer [292], *ResNets* — which are a family of convolutional networks of different sizes and depths and *skip connections* [117], *DenseNets* — another family of models where unlike standard neural networks, every layer in a "block" is connected to every other layer [135]. More recent, complex models include *ResNeXt* [354] and recently *EfficientNets*, which have separate scaling factors for network depth, width and the spatial resolution of the input image [308]. Tutorials, implementations and pretrained versions of many of these models can be found in the references given in Section 2.3.

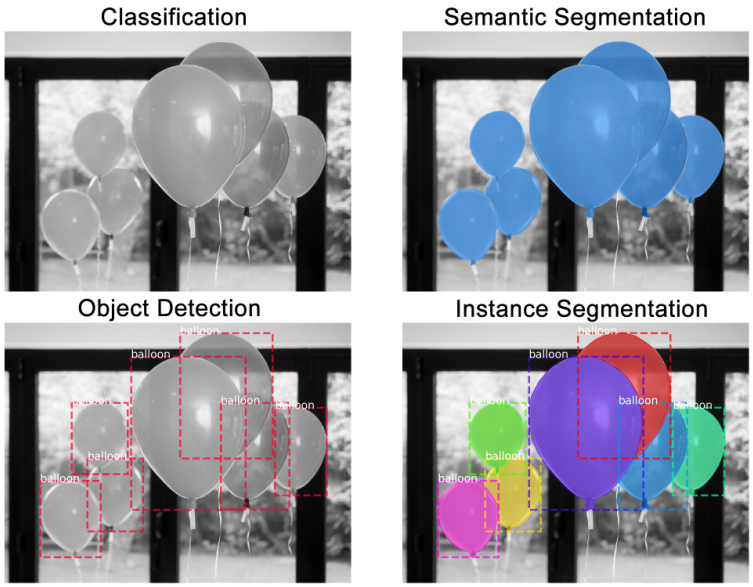



Figure 2.3: **Differences between Image Classification, Object Detection, Semantic Segmentation and Instance Segmentation tasks. Image source [2]** The figure illustrates the differences between classification, object detection, semantic segmentation and instance segmentation. In classification, the whole image gets a single label (balloons), while in object detection, each balloon is also localized with a bounding box. In semantic segmentation, all the pixels corresponding to balloon are identified, while in instance segmentation, each individual balloon is identified separately.

**Scientific Examples:** Image classification has found many varied scientific applications, such as in analyzing cryoEM data [312] (with associated code https://github.com/cramerlab/boxnet). An especially large body of work has looked at *medical imaging* uses of image classification, specifically, using CNNs to predict disease labels. Examples range from ophthalmology [101], radiology (2D x-rays and 3D CT scans) [359, 10, 255], pathology [193, 73], analyzing brain scans (PET, fMRI) [279, 62]. An excellent survey of the numerous papers in this area is given by [316].

### Object Detection

Image classification can be thought of as a global summary of the image. Object detection dives into some of the lower level details of the image, and looks at identifying and *localizing* different objects in the image. For example, given an input image of an outdoor scene having a dog, a person and a tree, object detection would look at both identifying the presence of the dog, person and tree and 'circle their location' in the image — specifically, put a *bounding box* around each of them. The supervised learning task is thus to take an input image and output the coordinates of these bounding boxes, as well as categorizing the kind of object they contain.

Like image classification, there are many high performing and well established convolutional architectures for object detection. Because of the intricacy of the output task, these models tend to be more complex with a *backbone* component (using an image classification model) and a *region proposal* component for bounding box proposals. But there are still many pretrained models available to download. One of the most successful early models was *Faster R-CNN* [264], which significantly sped up the slow bounding box proposal component. Since then there have been many improved models, including *YOLOv3* [262], and most recently *EfficientDets* [309]. Arguably the most popular recent architecture however has been *Mask R-CNN* and its variants [116, 348]. Mask R-CNN performs some segmentation as well as object detection (see below). Besides some of the resources mentioned in Section 2.3, a good source of code and models is `https://github.com/rbgirshick`, one of the key authors in a long line of these object detection models. (Note though that there are many other popular implementations, such as `https://github.com/matterport/Mask_RCNN`.)

This in depth article towardsdatascience object detection Faster R-CNN offers a detailed tutorial on downloading, setting up and training an object detection model, including helpful pointers to data collection and annotation (the latter using https://rectlabel.com/.) Most recently the Detectron2 system https://github.com/facebookresearch/detectron2 [348] builds on Mask R-CNN and offers many varied image task functionalities.

**Scientific Examples:** Object detection has also gained significant attention across different scientific applications. It has been used in many medical settings to localize features of interest, for example, tumor cells across different imaging modalities [181, 373] or fractures in radiology [274, 314].

### Semantic Segmentation and Instance Segmentation

Segmentation dives into the lowest possible level of detail — categorizing every single image *pixel*. In semantic segmentation, we want to categorize pixels according to the high level group they belong to. For example, suppose we are given an image of a street, with a road, different vehicles, pedestrians, etc. We would like to determine if a pixel is part of any pedestrian, part of any vehicle or part of the road — i.e. label the image pixels as either pedestrian, vehicle or road. Instance segmentation is even more intricate, where not only do we want to categorize each pixel in this way, but do so separately for each instance (and provide instance specific bounding boxes like in object detection). The differences are illustrated in Figure 2.3 (sourced from [2].) Returning to the example of the image of the street, suppose the image has three pedestrians. In

semantic segmentation, all of the pixels making up these three pedestrians would fall under the same category – pedestrian. In instance segmentation, these pixels would be further subdivided into those belonging to pedestrian one, pedestrian two or pedestrian three.

Because segmentation models must categorize every pixel, their output is not just a single class label, or a bounding box, but a full image. As a result, the neural network architectures for segmentation have a slightly different structure that helps them better preserve spatial information about the image. A highly popular and successful architecture, particularly for scientific applications, has been the *U-net* [269], which also has a 3d volumetric variant [46]. Other architectures include FCNs (Fully Convolutional Networks) [194], SegNet [16] and the more recent Object Contextual Representations [362]. A couple of nice surveys on semantic segmentation methods are given by towardsdatascience Semantic Segementation with Deep Learning and https://sergioskar.github.io/Semantic_Segmentation/.

For instance segmentation, Mask R-CNN [116] and its variants [348] have been extremely popular. This tutorial Mask R-CNN tutorial with code provides a step by step example application. The recent Detectron2 package [348] (https://github.com/facebookresearch/detectron2) also offers this functionality.

**Scientific Examples:** Out of all of the different types of imaging prediction problems, segmentation methods have been especially useful for (bio)medical applications. Examples include segmenting brain MR images [218, 330], identifying key regions of cells in different tissues [355, 297] and even studying bone

structure [187].

### Super-Resolution

Super resolution is a technique for transforming low resolution images to high resolution images. This problem has been tackled both using convolutional neural networks and supervised learning, as well as generative models.

Super resolution formally defined is an underdetermined problem, as there may be many possible high resolution mappings for a low resolution image. Traditional techniques imposed constraints such as sparsity to find a solution. One of the first CNNs for super resolution, *SRCNN* [67] outlines the correspondences between sparse coding approaches and convolutional neural networks. More recently, *Residual Dense Networks* [374] have been a popular approach for super-resolution on standard benchmarks (with code available https://github.com/yulunzhang/RDN), as well as Predictive Filter Flow [163], (code: https://github.com/aimerykong/predictive-filter-flow) which has also looked at image denoising and deblurring. In some of the scientific applications below, *U-nets* have also been successful for super resolution.

**Scientific Examples:** Super resolution is arguably even more useful for scientific settings than standard natural image benchmarks. Two recent papers look at U-nets for super-resolution of fluorescence microscopy [342] (code: https://csbdeep.bioimagecomputing.com/) and electron microscopy [74]. Other examples include super resolution of chest CT scans [320] and Brain MRIs [44].

### Image Registration

Image registration considers the problem of *aligning* two input images to each other. Particularly relevant to scientific applications, the two input images might be from different imaging modalities (e.g. a 3D scan and a 2D image), or mapping a moving image to a canonical template image (such as in MRIs.) The alignment enables better identification and analysis of features of interest.

The potential of image registration is primarily demonstrated through different scientific applications. At the heart of the technique is a convolutional neural network, often with an encoder-decoder structure (similar to the U-net [269]) to guide the alignment of two images. Note that while this underlying model is trained through supervised learning, many registration methods *do not require explicit labels*, using similarity functions and smoothness constraints to provide supervision. For example, [19] develop an unsupervised method to perform alignment for Brain MRIs. The code for this and several followup papers [20, 52] provides a helpful example for building off of and applying these methods `https://github.com/voxelmorph/voxelmorph`. Other useful resources include `https://github.com/ankurhanda/gvnn` (with corresponding paper [110]) a library for learning common parametric image transformations.

### Pose Estimation

Pose estimation, and most popularly human pose estimation, studies the problem of predicting the pose of a human in a given image. In particular, a deep neural network model is trained to identify the location of the main joints, the *keypoints*

Figure 2.4: **Pose Estimation. Image source [300]** The task of pose estimation, specifically multi-person 2D (human) pose-estimation is depicted in the figure. The neural network model predicts the positions of the main joints (keypoints), which are combined with a body model to get the stick-figure like approximations of pose overlaid on the multiple humans in the image. Variants of these techniques have been used to study animal behaviors in scientific settings.

(e.g. knees, elbows, head) of the person in the image. These predictions are combined with existing *body models* to get the full stick-figure-esque output summarizing the pose. (See Figure 2.4, sourced from [300], for an illustration.)

(2D) Human pose estimation is a core problem in computer vision with multiple benchmark datasets, and has seen numerous convolutional architectures developed to tackle it. Some of the earlier models include a multi-stage neural network introduced by [341], and a stacked hourglass model [223] that alternatingly combines high and low resolutions of the intermediate representations. More recently, HRNet [300], which keeps a high resolution representation throughout the model is a top performing architecture (code at `https://github.com/leoxiaobin/deep-high-resolution-net.pytorch`). Also of interest might be [35] provides an end-to-end system for multiperson pose

detection in the corresponding code repository https://github.com/CMU-Perceptual-Computing-Lab/openpose.

**Scientific Examples:** Pose estimation has gained significant interest in neuroscience settings, where videos of animals are recorded, and automatically predicting poses in the image can help identify important behaviors. An example is given by [206, 207], with associated code http://www.mousemotorlab.org/deeplabcut.

#### Other Tasks with Convolutional Neural Networks

In the preceding sections, we have overviewed some of the most common tasks for which convolutional neural networks are used. However, there are many additional use cases of these models that we have not covered, including *video prediction* [77], *action recognition* [69] and *style transfer* [88]. We hope that the provided references and resources enable future investigation into some of these methods also.

### 2.4.4 Graph Neural Networks

Many datasets, such as (social) network data and chemical molecules have a *graph* structure to them, consisting of vertices connected by edges. An active area of research, graph neural networks, has looked at developing deep learning methods to work well with this kind of data. The input graph consists of nodes $v$

having some associated feature vector $h_v$, and sometimes edges $e_{uv}$ also having associated features $z_{e_{uv}}$. For example, nodes $v$ might correspond to different atoms, and the edges $e_{uv}$ to the different kinds of chemical bonds between atoms. At a high level, most graph neural networks compute useful information from the data by (i) using the feature vectors of the *neighbors* of each vertex $v$ to compute information on the input graph instance (ii) using this information to update the feature vector of $v$. This process, which respects the connectivity of the graph, is often applied iteratively, with the final output either at the vertex level (Are meaningful vertex feature vectors computed?) or at the level of the full input graph (Is some global property of the entire graph correctly identified?)

**Application Characteristics** Problems where the data has an inherent graph structure, and the goal is to learn some function on this graph structure — either at the per vertex level or a global property of the entire graph. There are also spatio-temporal graph neural networks — performing predictions on graph structures evolving over time.

**Technical References** Although most graph neural networks follow the high level structure of aggregating information from vertex neighbors and using this information to update feature vectors, there are many many different architectural variants, with connections to other neural network models such as convolutional nets and recurrent models. Recent work has also looked at spatio-temporal graph networks for problems like action recognition in video [179]. A nice unification of many of the first popular methods, such as [70, 24, 185], is given by [92]. A more recent survey paper [350], provides an *extremely comprehensive* overview of the different kinds of architectures, problems,

benchmark datasets and open source resources. Some useful code repositories include https://github.com/rusty1s/pytorch_geometric, https://github.com/deepmind/graph_nets and https://github.com/dmlc/dgl, which together cover most of the popular deep learning frameworks.

**Scientific Examples** Graph neural networks have been very popular for several chemistry tasks, such as predicting molecular properties [70, 134, 92, 147], determining protein interfaces [82, 317] and even generating candidate molecules [56, 31]. A useful library for many of these chemistry tasks is https://github.com/deepchem, which also has an associated benchmark task [349]. A detailed tutorial of different graph neural networks and their use in molecule generation can be seen at https://www.youtube.com/watch?v=VXNjCAmb6Zw.

### 2.4.5 Neural Networks for Sequence Data

A very common attribute for data is to have a *sequential* structure. This might be frames in a video, amino acid sequences for a protein or words in a sentence. Developing neural network models to work with sequence data has been one of the most extensive areas of research in the past few years. A large fraction of this has been driven by progress on tasks in *natural language processing*, which focuses on getting computers to work with the language used by people to communicate. Two popular tasks in this area, which have seen significant advances, have been *machine translation* — developing deep learning models to translate from one language to another and *question answering* — taking as input a (short) piece of text and answering a question about it. In the following sections, we

first overview some of the main NLP tasks that have driven forward sequence modelling and then the neural network models designed to solve these tasks.

**Language Modelling (Next Token Prediction)**

Language modelling is a training method where the deep learning model takes as input the tokens of the sequence up to time/position $t$, and then uses these to predict token $t + 1$. This is in fact a *self-supervised* training method (see Section 2.7), where the data provides a natural set of labels without additional labelling needed. In the NLP context, the neural network is fed in a sequence of words, corresponding to a sentence or passage of text, and it tries to predict the next word. For example, given a sentence, "The cat sat on the roof", the network would first be given as input "The" and asked to predict "cat", then be fed in "The cat" and asked to predict "sat", and so on. (There are some additional details in implementation, but this is the high level idea.) Because of the easy availability of data/labels, and the ability to use language modelling at different levels — for words and even for characters, it has been a popular benchmark in natural language, and also for capturing sequence dependencies in scientific applications, such as protein function prediction [112, 118], and using the hidden representations as part of a larger pipeline for protein structure prediction in AlphaFold [282] (with opensourced code https://github.com/deepmind/deepmind-research/tree/master/alphafold_casp13.)

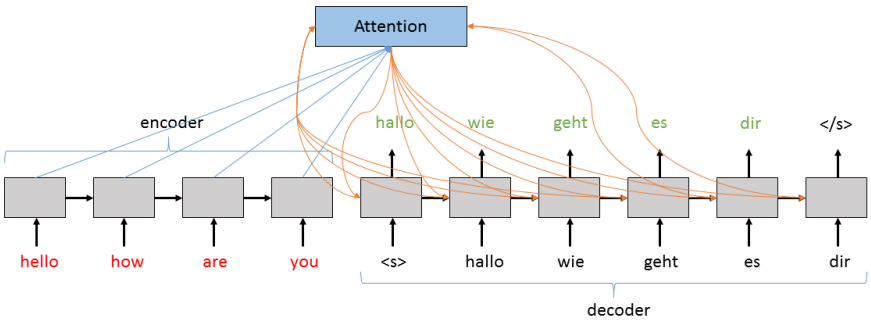



Figure 2.5: **Illustration of the Sequence to Sequence prediction task. Image source [369]** The figure shows an illustration of a Sequence to Sequence task, translating an input sentence (sequence of tokens) in English to an output sentence in German. Note the encoder-decoder structure of the underlying neural network, with the encoder taking in the input, and the decoder generating the output, informed by the encoder representations and the previously generated output tokens. In this figure, the input tokens are fed in one by one, and the output is also generated one at a time, which is the paradigm when using *Recurrent Neural Networks* as the underlying model. With *Transformer* models, which are now extremely popular for sequence to sequence tasks, the sequence is input all at once, significantly speeding up use.

### Sequence to Sequence

Another very popular task for sequence data is *sequence to sequence* — transforming one sequence to another. This is precisely the setup for machine translation, where the model gets an input sentence (sequence) in say English, and must translate it to German, which forms the output sentence (sequence). Some of the first papers framing this task and tackling it in this way are [17, 304, 328]. Sequence to sequence tasks typically rely on neural network models that have an *encoder-decoder* structure, with the encoder neural network taking in the input sequence and learning to extract the important features, which is then used by the decoder neural network to produce the target output. Figure 2.5(sourced from [369]) shows an example of this. This paradigm has also found some scientific applications as varied as biology [34] and energy forcasting [205]. Sequence to sequence models critically rely on a technique called *attention*, which we overview

below. For more details on this task, we recommend looking at some of the tutorials and course notes highlighted in Section 2.3.

### Question Answering

One other popular benchmark for sequence data has been question answering. Here, a neural network model is given a paragraph of text (as context) and a specific question to answer on this context as input. It must then output the part of the paragraph that answers the question. Some of the standard benchmarks for this task are [121, 256], with http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture10-QA.pdf providing an excellent overview of the tasks and common methodologies. Question answering critically relies on the neural network model understanding the relevance and similarity of different sets of sequences (e.g. how relevant is this part of the context to the question of interest?). This general capability (with appropriate reformulation) has the potential to be broadly useful, both for determining similarity and relevance on other datasets, and for question answering in specialized domains [84].

### Recurrent Neural Networks

Having seen some of the core tasks in deep learning for sequence data, these next few sections look at some of the key neural network models.

Recurrent neural networks (RNNs) were the first kind of deep learning model successfully used on many of the aforementioned tasks. Their distinguishing feature, compared to CNNs or MLPs (which are *feedforward* neural networks,

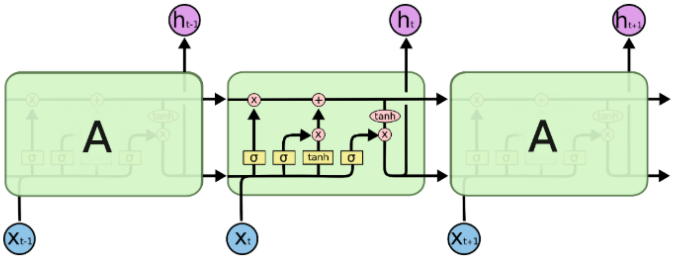



Figure 2.6: **Diagram of a Recurrent Neural Network model, specifically a LSTM (Long-Short Term Network). Image source [229]** The figure illustrates an LSTM network, a type of Recurrent Neural Network. We see that the input $x_t$ at each timestep also inform the internal network state in the next timestep (hence a recurrent neural network) through a *gating mechanism*. This gating mechanism is called an LSTM, and consists of sigmoid and tanh functions, which transform and recombine the input for an updated internal state, and also emit an output. The mechanics of this gating process are shown in the middle cell of the figure.

mapping input straight to output), is that there are *feedback connections*, enabling e.g. the output at each timestep to become the input for the next timestep, and the preservation and modification of an internal state across timesteps. When RNNs are used for sequential data tasks, sequences are input token by token, with each token causing an update of the internal *cell state* of the RNN, and also making the RNN emit a token output. Note that this enables these models to work with *variable length* data — often a defining characteristic of sequence data. How the input is processed, cell state updated and output emitted are controlled by *gating functions* — see the technical references!

**Application Characteristics:** Problems where the data has a sequential nature (with different sequences of varying length), and prediction problems such as determining the next sequence token, transforming one sequence to another, or determining sequence similarities are important tasks.

**Technical References:** Research on sequence models and RNNs has evolved dramatically in just the past couple of years. The most successful and popular kind of RNN is a *bi-LSTM with Attention*, where LSTM (Long-Short Term Memory) [126] refers to the kind of gating function that controls updates in the network, bi refers to bidirectional (the neural network is run forwards *and* backwards on the sequence) and Attention is a very important technique that we overview separately below. (Some example papers [210, 211] and code resources https://github.com/salesforce/awd-lstm-lm.) This excellent post https://colah.github.io/posts/2015-08-Understanding-LSTMs/ provides a great overview of RNNs and LSTMs in detail. (Figure 2.6 shows a diagram from the post revealing the details of the gating mechanisms in LSTMs.) The post also describes a small variant of LSTMs, Gated Recurrent Units (GRUs) which are also popular in practice [185]. While RNNs (really bi-LSTMs) have been very successful, they are often tricky to develop and train, due to their recursiveness presenting challenges with optimization (the vanishing/exploding gradients problem [125, 237, 111]), with performing fast model training (due to generating targets token by token), and challenges learning long term sequential dependencies. A new type of feedforward neural network architecture, the *Transformer* (overviewed below), was proposed to alleviate the first two of these challenges.

**Scientific Examples:** RNNs have found several scientific applications for data with sequential structure, such as in genomics and proteomics [242, 190, 160].

### Attention

A significant problem in using RNNs and working with sequential data is the difficulty in capturing *long range dependencies*. Long range dependencies are when tokens in the sequence that are very far apart from each other must be processed together to inform the correct output. RNNs process sequences in order, token by token, which means they must remember all of the important information from the earlier tokens until much later in the sequence — very challenging as the memory of these architectures is far from perfect. *Attention* [45, 18] is a very important technique that introduces *shortcut connections* to earlier tokens, which alleviates the necessity to remember important features for the duration of the entire sequence. Instead it provides a direct way to model long term dependencies — the neural network has the ability to *look back* and *attend* to what it deems relevant information (through learning) earlier in the input. A very nice overview of attention is provided by https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html. A variant of attention, *self-attention*, which can be used to help predictions on a single input sequence, is the core building block of Transformer models.

### Transformers

While attention helped with challenges in long range dependencies, RNNs still remained slow to train and tricky to design (due to optimization challenges with vanishing/exploding gradients.) These challenges were inherent to their recurrent, token-by-token nature, prompting the proposal of a new *feedforward* neural network to work with sequential data, the Transformer [325], which

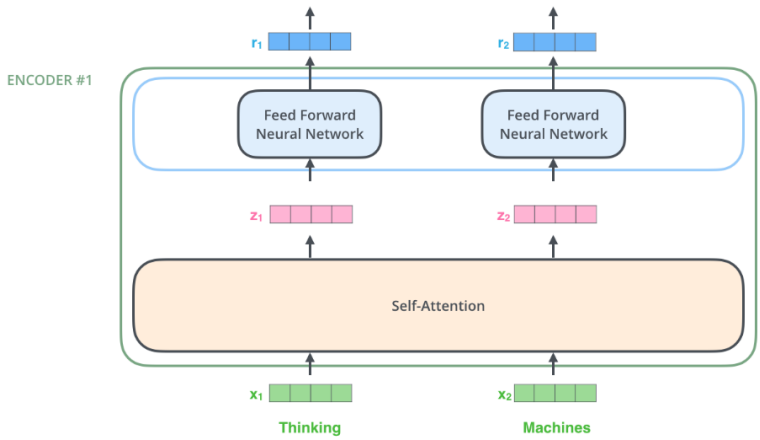



Figure 2.7: **Image of a couple of layers from a Transformer network. Image source [8]** The figure depicts the core sequence of layers that are fundamental to Transformer neural networks, a *self-attention* layer (sometimes called a self-attention head) followed by fully connected layers. Note that when working with sequence data, transformers take the entire input sequence all at once, along with positional information (in this case the input sequence being "Thinking Machines".)

critically relies on attentional mechanisms (the paper is in fact titled *Attention is All you Need*.) During training transformers take in the *entire* sequence as input all at once, but have *positional embeddings* that respects the sequential nature of the data. Transformers have been exceptionally popular, becoming the dominant approach to many natural language tasks and sequential tasks.

**Application Characteristics:** Problems where the data has a sequential nature and long range dependencies that need to be modelled. Given the large number of pretrained transformer models, they can also be very useful in settings where pretrained models on standard benchmarks can be quickly adapted to the target problem.

**Technical References:** The original transformer paper [325] provides a nice overview of the motivations and the neural network architecture. The model was

designed with machine translation tasks in mind, and so consists of an *encoder* neural network and a *decoder* neural network. With transformers being adopted for tasks very different to machine translation, the encoder and decoder are often used in stand-alone fashions for different tasks — for example, the encoder alone is used for question answering, while the decoder is important for text generation. Two very accessible step by step tutorials on the transformer are The Annotated Transformer and The Illustrated Transformer. A nice example of some of the language modelling capabilities of this models is given by [247].

Since the development of the transformer, there has been considerable research looking at improving the training of these models, adjusting the self-attention mechanism and other variants. A very important result using the transformer has been BERT (Pretraining of deep Bi-directional Transformers for Language understanding) [60]. This paper demonstrates that performing *transfer learning* (see Section 2.5.1) using a transformer neural network can be extremely successful for many natural language tasks. (Some of the first papers showing the potential of transfer learning in this area were [132, 247], and since BERT, there have been followups which extend the model capabilities [358].) From a practical perspective, the development of transformers, BERT and transfer learning mean that there are many resources available online for getting hold of code and pretrained models. We refer to some of these in Section 2.3, but of particular note is `https://github.com/huggingface/transformers` which has an excellent library for transformer models. A good overview of BERT and transfer learning in NLP is given in `http://jalammar.github.io/illustrated-bert/`.

**Scientific Examples:** There have been several interesting examples of transformers used in scientific settings, such as training on protein sequences to find representations encoding meaningful biological properties [267], protein generation via language modelling [200], bioBERT [174] for text mining in biomedical data (with pretrained model and training code), embeddings of scientific text [27] (with code `https://github.com/allenai/scibert`) and medical question answering [332].

#### Other Tasks with Sequence Data

In the previous sections, we've given an overview of some of the important benchmark tasks for sequential data, and the types of deep learning models available to tackle them. As with convolutional networks, this is not a comprehensive overview, but hopefully thorough enough to help with generating ideas on possible applications and offering pointers to other useful related areas. A few other sequential data tasks that might be of interest are *structured prediction*, where the predicted output has some kind of structure, from tree structures (in e.g. parsing) [39, 343] to short, executable computer program structure [375] and *summarization*, where passages of text are summarized by a neural network [188, 378]. We'll also discuss *word embeddings* later in the survey.

### 2.4.6 Section Summary

In this section, we have overviewed supervised learning, some of the core neural network models and the kinds of important tasks they can be used for. As

previously discussed, these topics span an extremely large area of research, so there are some areas, e.g. deep neural networks for set structured data [364, 162], modelling different invariances — invariances to specified Lie groups for application to molecular property prediction [80], spherical invariances [48, 49] not covered. But we hope the material and references presented help inspire novel contributions to these very exciting and rapidly evolving research directions.

## 2.5 Key (Supervised Learning) Methods

In the previous section we saw different kinds of neural network models, and the many different types of tasks they could be used for. To train the models for these tasks, we typically rely on the supervised learning methodology — optimize model parameters to correctly output given labels (the supervision) on a set of training data examples.

In more detail, the standard supervised learning method for deep neural networks consists of (i) collecting data instances (e.g. images) (ii) collecting *labels* for the data instances (e.g. is the image a cat or a dog) (iii) splitting the set of collected (data instance, label) into a training set, validation set and test set (iv) *randomly initializing* neural network parameters (iv) optimizing parameters so the network outputs the correct corresponding label given an input data instance on the training set (v) further tuning and validating on the validation and test sets.

In this section we overview methods that use variants of this process, for example initializing the neural network parameters differently or dealing with

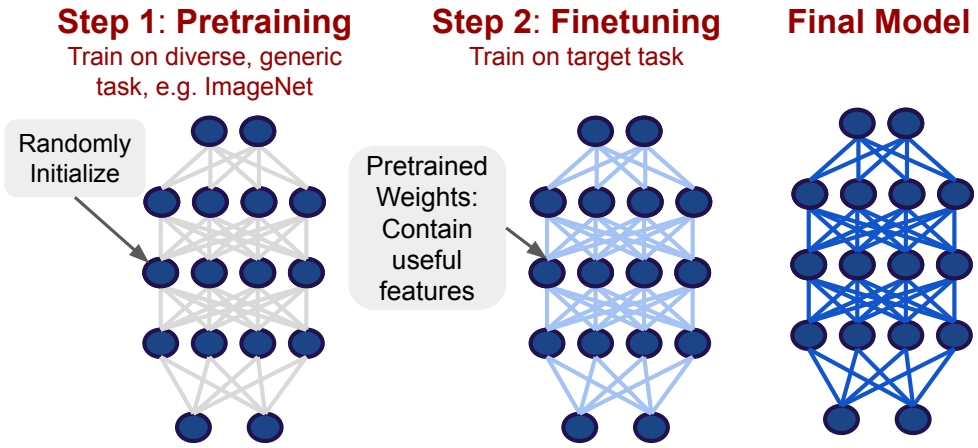



Figure 2.8: **The Transfer Learning process for deep neural networks.** Transfer learning is a two step process for training a deep neural network. Instead of intializing parameters randomly and directly training on the target task, we first perform a *pretraining* step, on some diverse, generic task. This results in the neural network parameters converging to a set of values, known as the *pretrained weights*. If the pretraining task is diverse enough, these pretrained weights will contain useful features that can be leveraged to learn the target task more efficiently. Starting from the pretrained weights, we then train the network on the target task, known as *finetuning*, giving us the final model.

shifts between the training data and the test sets. In Section 2.7, we look at variants that reduce the dependence on collecting labels.

### 2.5.1 Transfer Learning

Through the preceding sections, we've made references to using *pretrained* models. This is in fact referring to a very important method for training deep neural networks, known as *transfer learning*. Transfer learning is a two step process for training a deep neural network model, a *pretraining* step, followed by a *finetuning* step, where the model in trained on the target task. More specifically, we take a neural network with parameters randomly initialized, and first train it on a

standard, generic task — the pretraining step. For example, in image based tasks, a common pretraining task is ImageNet [58], which is an *image classification* task on a large dataset of natural images. With an appropriate pretraining task that is generic and complex enough, the pretraining step allows the neural network to learn useful features, stored in its parameters, which can then be reused for the second step, *finetuning*. In finetuning, the pretrained neural network is further trained (with maybe some minor modifications to its output layer) on the *true target task* of interest. This process is illustrated in Figure 2.8. But being able to use the features it learned during pretraining often leads to boosts in performance and convergence speed of the target task, as well as needing less labelled data.

Because of these considerable benefits, transfer learning has been extraordinarily useful in many settings, particularly in computer vision [136], which had many early successful applications. As overviewed in Section 2.4.5, the recent development of models like ULMFiT [132] and especially BERT [60] has also made transfer learning extremely successful in natural language and sequential data settings, with recent work making the transfer learning process even more efficient [130, 276]. Most importantly, the ready availability of standard neural network architectures pretrained on standard benchmarks through many open sourced code repositories on GitHub (examples given in Section 2.3) has meant that downloading and finetuning a standard pretrained model has become the *de-facto standard* for most new deep learning applications.

Typically, performing transfer learning is an *excellent* way to start work on a new problem of interest. There is the benefit of using a well-tested, standard neural network architecture, aside from the knowledge reuse, stability and convergence boosts offered by pretrained weights. Note however that the precise

effects of transfer learning are not yet fully understood, and an active research area [166, 254, 368, 224, 202, 248, 329] looks at investigating its exact properties. For transfer learning in vision [166, 368, 161] may be of particular interest for their large scale studies and pretraining recommendations.

### 2.5.2 Domain Adaptation

Related to transfer learning is the task of *domain adaptation*. In (unsupervised) domain adaptation, we have training data and labels in a *source* domain, but want to develop a deep learning model that will also work on a *target* domain, where the data instances may look different to those in the source domain, but the high level task is the same. For instance, our source domain many consist of images of handwritten digits (zero to nine) which we wish to classify as the correct number. But the target domain many have photographs of house numbers (from zero to nine), that we also wish to classify as the correct number. Domain adaptation techniques help build a model on the source domain that can also work (reasonably) well out-of-the-box on the shifted target domain.

The most dominant approach to domain adaptation in deep learning is to build a model that can (i) perform well on the source domain task, and (ii) learns features that are as invariant to the domain shift as possible. This is achieved through *jointly optimizing* for both of these goals. Returning to our example on handwritten digits and house number photographs, (i) corresponds to the standard supervised learning classification problem of doing well on the (source) task of identifying handwritten digits correctly while (ii) is more subtle, and typically involves explicitly optimizing for the *hidden layer representations* of

handwritten digits and house number photographs to look the same as each other — domain invariance. Some popular ways to implement this include *gradient reversal* [85], minimizing a distance function on the hidden representations [195], and even *adversarial training* [86, 289]. More recently, [301] look at using *self-supervision* (see Section 2.7) to jointly train on the source and target domains, enabling better adaptation.

Other approaches to domain adaptation include translating data instances from the source to the target domain, and bootstrapping/co-training approaches (see Section 2.7.2). Some of these methods are overviewed in tutorials such as Deep Domain Adaptation in Computer Vision.

### 2.5.3 Multitask Learning

In many supervised learning applications, ranging from machine translation [5] to scientific settings [257, 243], neural networks are trained in a *multitask* way – predicting several different outputs for a single input. For example, in image classification, given an input medical image, we might train the network not only to predict a disease of interest, but patient age, history of other related disease, etc. This often has beneficial effects even if there is only one prediction of interest, as it provides the neural network with useful additional feedback that can guide it in learning the most important data features. (This can be so useful that sometimes auxiliary prediction targets are defined solely for this purpose.) Additionally, the prediction of multiple targets can mean that more data is available to train the model (only a subset of the data has the target labels of interest, but many more data instances have other auxiliary labels.) The

most extreme version of this is to *simultaneously train* on two entirely different datasets. For example, instead of performing a pretraining/finetuing step, the model could be trained on both ImageNet and a medical imaging dataset at the same time.

Multitask learning is usually implemented in practice by giving the neural network multiple *heads*. The head of a neural network refers to its output layer, and a neural network with multiple heads has one head for each predictive task (e.g. one head for predicting age, one for predicting the disease of interest) but *shares* all of the other features and parameters, across these different predictive tasks. This is where the benefit of multitask learning comes from — the shared features, which comprise of most of the network, get many different sources of feedback. Implementing multitask learning often also requires careful choice of the way to weight the training objectives for these different tasks. A nice survey of some popular methods for multitask learning is given by https://ruder.io/multi-task/index.html#fn4, and a tutorial on some of the important considerations in http://hazyresearch.stanford.edu/multi-task-learning. One package for implementing multitask learning is found in https://github.com/SenWu/emmental and step-by-step example with code excerpts in towardsdatascience Multitask Learning: teach your AI more to make it better.

### 2.5.4 Weak Supervision (Distant Supervision)

Suppose it is very difficult to collect high quality labels for the target task of interest, and neither is there an existing, standard, related dataset and corresponding

pretrained model to perform transfer learning from. How might one provide the deep learning model with enough supervision during the training process? While high quality labels might be hard to obtain, *noisy labels* might be relatively easy to collect. *Weak supervision* refers to the method of training a model on a dataset with these noisy labels (typically for future finetuning), where the noisy labels are often generated in an *automatic process*.

In computer vision (image based) tasks, some examples are: taking an image level label (for classification) and automatically inferring pixel level labels for segmentation [238], clustering hidden representations computed by a pretrained network as pseudo-labels [356], or taking Instagram tags as labels [202] for pretraining. In language tasks, examples are given by [214, 127, 366], which provide noisy supervision by assuming all sentences mentioning two entities of interest express a particular relation (also known as *distant supervision*). A nice overview of weak supervision and its connection to other areas is given in https://hazyresearch.github.io/snorkel/blog/ws_blog_post.html, with a related post looking specifically at medical and scientific applications http://hazyresearch.stanford.edu/ws4science.

### 2.5.5 Section Summary

In this section, we have overviewed some of the central supervised learning based methodologies for developing deep learning models. This is just a sampling of the broad collection of existing methods, and again, we hope that the descriptions and references will help facilitate further exploration of other approaches. One method not covered that might be of particular interest is *multimodal learning*,

where neural networks are simultaneously trained on data from different modalities, such as images and text [197, 333, 146]. Multimodal learning also provides a good example of the fact that it is often difficult to precisely categorize deep learning techniques as only being useful for a specific task or training regime. For example, we looked at language modelling for sequence tasks in this supervised learning section, but language modelling is also an example of self-supervision (Section 2.7) and generative models (Section 2.8.1). There are many rich combinations of the outlined methods in both this section and subsequent sections, which can prove very useful in the development of an end to end system.

## 2.6 Interpretability, Model Inspection and Representation Analysis

Many standard applications of deep learning (and machine learning more broadly) focus on *prediction* — learning to output specific target values given an input. Scientific applications, on the other hand, are often focused on *understanding* — identifying underlying mechanisms giving rise to observed patterns in the data. When applying deep learning in scientific settings, we can use these observed phenomena as prediction targets, but the ultimate goal remains to understand what attributes give rise to these observations. For example, the core scientific question might be on how certain amino acid sequences (encoding a protein) give rise to particular kinds of protein function. While we might frame this as a prediction problem, training a deep neural network to take as input an amino acid sequence and output the predicted properties of the protein, we would ideally like to understand how that amino acid sequence resulted in the

observed protein function.

To answer these kinds of questions, we can turn to *interpretability* techniques. Interpretability methods are sometimes equated to a fully understandable, step-by-step explanation of the model's decision process. Such detailed insights can often be intractable, especially for complex deep neural network models. Instead, research in interpretability focuses on a much broader suite of techniques that can provide insights ranging from (rough) *feature attributions* — determining what input features matter the most, to *model inspection* — determining what causes certain neurons in the network to fire. In fact, these two examples also provide a rough split in the type of interpretability method.

One large set of methods (which we refer to as Feature Attribution and Per Example Interpretability) concentrates on taking a specific input along with a trained deep neural network, and determining what features of the input are most important. The other broad class of techniques looks at taking a trained model, and a *set* of inputs, to determine what different parts of the network have learned (referred to as Model Inspection and Representational Analysis). This latter set of methods can be very useful in revealing important, hidden patterns in the data that the model has implicitly learned through being trained on the predictive task. For example, in [169], which looks at machine translation, representation analysis techniques are used to illustrate latent linguistic structure learned by the model. We overview both sets of methods below.

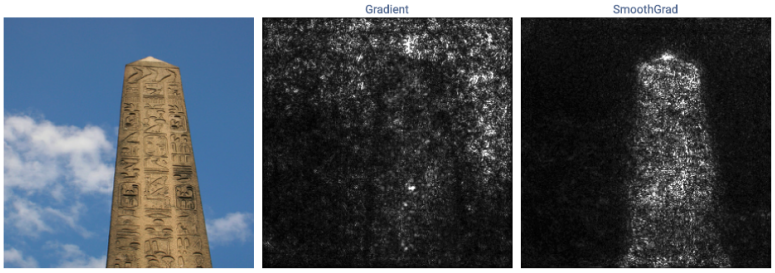



Figure 2.9: **The output of SmoothGrad, a type of saliency map. Image source [294]** The figure shows the original input image (left), raw gradients (middle), which are often too noisy for reliable feature attributions, and SmoothGrad (right), a type of saliency map that averages over perturbations to produce a more coherent feature attribution visualization the input. In particular, we can clearly see that the monument in the picture is important for the model output.

### 2.6.1 Feature Attribution and Per Example Interpretability

We start off by overviewing some of the popular techniques used to provide *feature attribution* at a per example level, answering questions such as which parts of an input image are most important for a particular model prediction. These techniques can be further subcategorized as follows:

#### Saliency Maps and Input Masks

At a high level, saliency maps take the gradient of the *output prediction* with respect to the *input*. This gives a *mask* over the input, highlighting which regions have large gradients (most important for the prediction) and which have smaller gradients. First introduced by [291], there are many variants of saliency maps, such as Grad-CAM [281], SmoothGrad [294], IntGrad [302], which make the resulting feature attributions more robust. These and other methods are implemented in `https://github.com/PAIR-code/saliency`. Note that while these methods can be extremely useful, their predictions are not perfect [151],

and must be validated further.

Closely related to these saliency methods is [232], which provides the ability to inspect the kinds of features causing neurons across different hidden layers to fire. The full, interactive paper can be read at https://distill.pub/2018/building-blocks/ with code and tutorials available at https://github.com/tensorflow/lucid.

Many other techniques look at computing some kind of input mask, several of them using *deconvolutional* layers, first proposed by [365] and built on by [152] and [30]. Other work looks at directly optimizing to find a sparse mask that will highlight the most important input features [81] (with associated code https://github.com/jacobgil/pytorch-explain-black-box) or finding such a mask through an iterative algorithm [36].

#### Feature Ablations and Perturbations

Related to some of masking approaches above, but with enough differences to categorize separately are several methods that isolate the crucial features of the input either by performing feature *ablations* or computing perturbations of the input and using these perturbations along with the original input to inform the importance of different features.

Arguably the most well known of the ablation based approaches is the notion of a *Shapely value*, first introduced in [284]. This estimates the importance of a particular feature $x_0$ in the input by computing the predictive power of a subset of input features containing $x_0$ and averaging over all possible such subsets. While

Shapely values may be expensive to compute naively for deep learning, follow on work [198] has proposed more efficient (and expressive) variants, with highly popular opensourced implementation: https://github.com/slundberg/shap.

The *shap* opensourced implementation above also unifies some related approaches that use *perturbations* to estimate feature values. One such approach is LIME [266], which uses multiple local perturbations to enable learning an interpretable local model. Another is DeepLIFT, which uses a reference input to compare activation differences [288], and yet another approach, Layer-wise Relevance Propagation [13] looks at computing relevance scores in a layerwise manner.

Other work performing ablations to estimate feature importance includes [380] (with code https://github.com/lmzintgraf/DeepVis-PredDiff), while [81], described in Section 2.6.1 has elements of using input perturbations.

### 2.6.2 Model Inspection and Representational Analysis

In this second class of interpretability methods, the focus is on gaining insights not at a *single* input example level, but using a set of examples (sometimes implicitly through the trained network) to understand the salient properties of the data. We overview some different approaches below.

Figure 2.10: **Visualization of the kinds of features hidden neurons have learned to detect. Image source [231]** This figure, from [231], illustrates the result of optimizing inputs to show what features hidden neurons have learned to recognize. In this example, the hidden neuron has learned to detect (especially) soccer balls, tennis balls, baseballs, and even the legs of soccer players.

### Probing and Activating Hidden Neurons

A large class of interpretability methods looks at either (i) *probing* hidden neurons in the neural network — understanding what kinds of inputs it activates for (ii) directly optimizing the *input* to activate a hidden neuron. Both of these techniques can provide useful insights into what the neural network has chosen to pay attention to, which in turn corresponds to important properties of the data.

Several papers falls into the probing category [361, 376], with an especially thorough study given by *Network Dissection* [26]. Here here hidden neurons are categorized by the kinds of features they respond to. The paper website `http://netdissect.csail.mit.edu/` contains method details as well as links to the code and data.

The other broad category of methods take a neural network, fix its parameters, and optimize the *input* to find the kinds of features that makes some hidden neuron activate. There are several papers using this approach, but of particular note is *Feature Visualization* [231], with an interactive article and code at: `https://distill.pub/2017/feature-visualization/`. Fol-

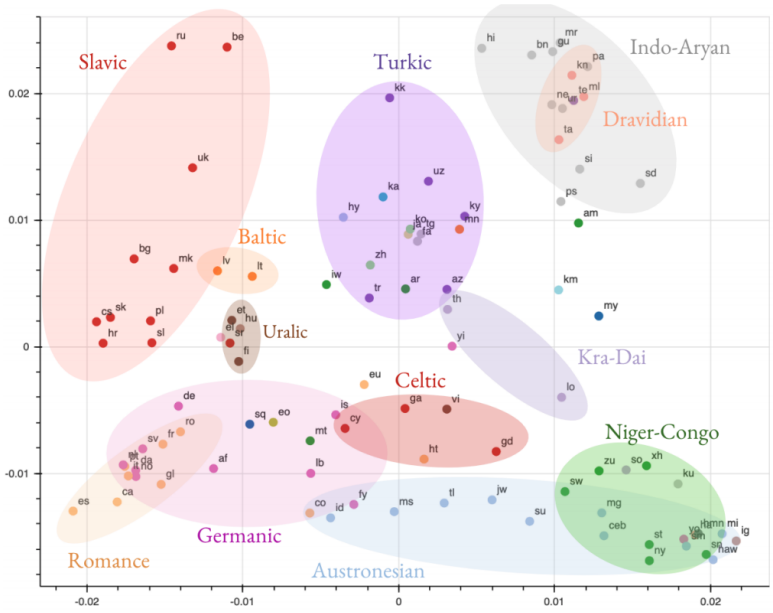



Figure 2.11: **Clustering neural network hidden representations to reveal linguistic structures. Image source [169]** In work on analyzing multilingual translation systems [169], representational analysis techniques are used to compute similarity of neural network (Transformer) hidden representations across different languages. Performing clustering on the result reveals grouping of different language representations (each language a point on the plot) according to language families, which affect linguistic structure. Importantly, this analysis uses the neural network to identify key properties of the underlying data, a mode of investigation that might be very useful in scientific domains.

lowup work, *Activation Atlases* [37] (with page `https://distill.pub/2019/activation-atlas/`), does this across many different concepts, providing a full mapping of the features learned by the neural network. More recently [230] has used this as a building block to further understand how certain computations are performed in a neural network. Also related is [150], which looks at finding linear combinations of hidden neurons that correspond to interpretable concepts.

### Dimensionality Reduction on Neural Network Hidden Representations

In many standard scientific settings, e.g. analyzing single cell data, dimensionality reduction methods such as PCA, t-SNE [199], UMAP [209] are very useful in revealing important factors of variation and critical differences in the data subpopulations e.g. tumor cells vs healthy cells. Such methods can also be used on the *hidden activations* (over some input dataset) of a neural network. Through the process of being trained on some predictive task, the neural network may implicitly learn these important data attributes in its hidden representations, which can then be extracted through dimensionality reduction methods.

### Representational Comparisons and Similarity

Related to more standard approaches of dimensionality reduction and clustering, a line of work has studied *comparing* hidden representations across different neural network models. Early work applied matching algorithms [184] with follow on approaches using *canonical correlation analysis* [253, 220] (with associated code `https://github.com/google/svcca`.) This latter approach has been used to identify and understand many representational properties in natural language applications [169, 25, 329] and even in modelling the mouse visual cortex as an artificial neural network [286]. Another recent technique uses a kernel based approach to perform similarity comparisons [164].

### 2.6.3 Technical References

The preceding sections contain many useful pointers to techniques and associated open sourced code references. One additional reference of general interest may be https://christophm.github.io/interpretable-ml-book/ a fully open sourced book on interpretable machine learning. This focuses slightly more on more traditional interpretability methods, but has useful overlap with some of the techniques presented above and may suggest promising open directions.

## 2.7 Doing More with Less Data

Supervised learning methods, and specific variants such as transfer learning and multitask learning have been highly successful in training deep neural network models. However, a significant limitation to their use, and thus the use of deep learning, is the dependence on large amounts of *labelled* data. In many specialized domains, such as medicine, collecting a large number of high quality, reliable labels can be prohibitively expensive.

Luckily, in just the past few years, we've seen remarkable advances in methods that reduce this dependence, particularly *self-supervision* and *semi-supervised learning*. These approaches still follow the paradigm of training a neural network to map raw data instances to a specified label, but critically, these labels are not collected separately, but *automatically defined* via a pretext task. For example, we might take a dataset of images, rotate some of them, and then define the label


Figure 2.12: **Training neural networks with Self-Supervision.** The figure illustrates one example of a self-supervision setup. In self-supervision, we typically have a collection of unlabelled data instances, in this case images. We define a *pretext task*, that will automatically generate labels for the data instances. In this case, the pretext task is rotation — we randomly rotate the images by some amount and label them by the degree of rotation. During training, the neural network is given this rotated image and must predict the degree of rotation. Doing so also requires the neural network learn useful hidden representations of the image data in general, so after training with self-supervision, this neural network can then be successfully and efficiently finetuned on a downstream task.

as the degree of rotation, which is the prediction target for the neural network. This enables the use of *unlabelled data* in training the deep neural network. In this section, we cover both self-supervision and semi-supervised learning as well as other methods such as data augmentation and denoising, all of which enable us to do more with less data.

### 2.7.1 Self-Supervised Learning

In self-supervision, a *pretext task* is defined such that labels can be *automatically* calculated directly from the raw data instances. For example, on images, we

could rotate the image by some amount, label it by how much it was rotated, and train a neural network to predict the degree of rotation [91] — this setup is illustrated in Figure 2.12. This pretext task is defined without needing any labelling effort, but can be used to teach the network good representations. These representations can then be used as is or maybe with a little additional data for downstream problems. Arguably the biggest success of self-supervision has been *language modelling* for sequential data and specifically natural language problems, which we overviewed in Section 2.4.5. Below we outline some of the most popular and successful self-supervision examples for both image and sequential data. (A comprehensive list of self-supervision methods can also be found on this page https://github.com/jason718/awesome-self-supervised-learning.)

#### Self-Supervised Learning for Images

A recent, popular and simple self-supervised task for images is to predict image rotations [91]. Each image instance is transformed with one of four possible rotations and the deep learning model must classify the rotation correctly. Despite its simplicity, multiple studies have shown its success in learning good representations [368, 367, 161]. Another popular method examined in those studies is *exemplar* [68], which proposes a self-supervision task relying on invariance to *image transformations*. For example, we might take a source image of a cat, and perform a sequence of transformations, such as rotation, adjusting contrast, flipping the image horizontally, etc. We get multiple images of the cat by choosing many such sequences, and train the neural network to recognize these all as the same image.

Other methods look at using image patches as *context* to learn about the global image structure and important features. For example, [65] defines a pretext task where the relative locations of pairs of image patches must be determined, while [227] teaches a neural network to solve jigsaw puzzles. This latter task has been shown to be effective at large scales [96], with nice implementations and benchmarking provided by https://github.com/facebookresearch/fair_self_supervision_benchmark. A recent line of work has looked at using *mutual information* inspired metrics as a way to provide supervision on the relatedness of different image patches [122, 235, 14, 215], but these may be more intricate to implement. Many of these mutual information based metrics also rely on *contrastive losses* [42], which, at a high level, provides supervision to the network by making representations of a pair of similar inputs more similar than representations of a pair of different inputs. Very recently, a new self-supervision method, SimCLR [41], uses this to achieve high performance (one implementation at https://github.com/sthalles/SimCLR.)

Note that some of the *image registration* examples given in Section 2.4.3 are also examples of self-supervised learning, where some kind of domain specific similarity function can be automatically computed to assess the quality of the output. Such approaches may be relevant to other domains, and are useful to explore. A great set of open-sourced implementations of many of self-supervision methods is provided by https://github.com/google/revisiting-self-supervised.

**Self-Supervised Learning for Sequential (Natural Language) Data**

While research on self-supervision techniques for images has been extremely active, the strongest successes of this framework have arguably been with sequential data, particularly text and natural language. The sequential structure immediately gives rise to effective self-supervision pretext tasks. Two dominant classes of pretext tasks operate by either (i) using neighboring tokens of the sequence as input *context* for predicting a target token (ii) taking in all tokens up to a particular position and predicting the next token. The latter of these is *language modelling*, which was overviewed in Section 2.4.5. The former is the principle behind *word embeddings*.

Word embeddings have been critical to solving many natural language problems. Before the recent successes of full fledged transfer learning in language (Section 2.5.1) this simple self-supervised paradigm was where knowledge reuse was concentrated, and formed a highly important component of any deep learning system for natural language (sequential) data. From a scientific perspective, learning word embeddings for sequential data has the potential to identify previously unknown similarities in the data instances. It has already found interesting uses in aiding with the automatic analysis of scientific texts, such as drug name recognition systems [189], biomedical named entity recognition [105], identifying important concepts in materials science [319] and even detecting chemical-protein interactions [50].

The key fundamental ideas of word embeddings are captured in the *word2vec* framework [213, 212], the original framework relying on either a *Continuous-Bag-of-Words* (CBOW) neural network or a *Skip-Gram* neural network. Actually,

both of these models are less neural networks and more two simple matrix multiplications, with the first matrix acting as a *projection*, and giving the desired embedding. In CBOW, the context — defined as the neighborhood words — are input, and the model must correctly identify the target output word. In Skip-Gram, this is reversed, with the center word being input, and the context being predicted. For example, given a sentence "There is a cat on the roof", with the target word being cat, CBOW would take in the vector representations of (There, is, a, on, the, roof) and output "cat", while Skip-Gram would roughly swap the inputs and outputs. The simplicity of these methods may make them more suitable for many tasks compared to language modelling. Two nice overviews of the these methods are given by Introduction to Word Embeddings and word2vec, and `https://ruder.io/word-embeddings-1/`. Other embedding methods include [240, 178].

**Self-Supervision Summary**

In this section we have outlined many of the interesting developments in self-supervised learning, a very successful way to make use of unlabelled data to learn meaningful representations, either for analysis or other downstream tasks. Self-supervision can be effectively used along with other techniques. For example, in the language modelling application, we saw it used for transfer learning (Section 2.5.1), where a deep learning model is first pretrained using the language modelling self supervision objective, and then finetuned on the target task of interest. In the following section, we will other ways of combining self-supervision with labelled data.

### 2.7.2 Semi-Supervised Learning

While collecting *large* labelled datasets can be prohibitively expensive, it is often possible to collect a smaller amount of labelled data. When assembling a brand new dataset, a typical situation is having a small amount of labelled data and a (sometimes significantly) larger number of data instances with no labels. *Semi-supervised learning* looks at precisely this setting, proposing techniques that enable effective learning on labelled and unlabelled data. Below we overview some of the popular methods for semi-supervised learning.

#### Self-Supervision with Semi-Supervised Learning

Following on from the previous section, one natural way to make use of the unlabelled data is to use a self-supervised pretext task. To combine this with the labelled data, we can design a neural network that has *two different outputs heads* (exactly as in multitask learning, see Section 2.5.3), with one output head being used for the labelled data, and the other for the self-supervised objective on the unlabelled data. Importantly, this means that the features learned by the neural network are *shared* between the labelled *and* unlabelled data, leading to better representations. This simple approach has been shown to be very effective [368, 367].

### Self-Training (Bootstrapping)

Self-training, sometimes also referred to as *bootstrapping* or *pseudo-labels*, is an iterative method where a deep neural network is first developed in a supervised fashion on the labelled data. This neural network is then used to provide (pseudo) labels to the unlabelled data, which can then be used in conjunction with the labelled data to train a new, more accurate neural network. This approach often works well and can even be repeated to get further improvements. There are a couple of common details in implementation — often when adding the neural network pseudo-labelled data, we only keep the most *confidently* pseudo-labelled examples. These pseudo-labelled examples may also be used for training with a different objective function compared to the labelled data. One of the early papers proposing this method was [172], with a more recent paper [353] demonstrating significant successes at large scale. Other variants, including *mean teacher* [311], *temporal ensembling* [170] and the recent *MixMatch* [28] also primarily use the self-training approach, but incorporate elements of *consistency* (see below). There are nice open sourced implementations of these methods, such as `https://github.com/CuriousAI/mean-teacher` for mean teacher and `https://github.com/google-research/mixmatch` and `https://github.com/YU1ut/MixMatch-pytorch` for MixMatch.

### Enforcing Consistency (Smoothness)

An important theme in many semi-supervised methods has been to provide supervision on the unlabelled data through enforcing *consistency*. If a human was given two images A and B, where B was a slightly perturbed version of A

(maybe blurred, maybe some pixels obscured or blacked out), they would give these images the same label — consistency. We can also apply this principle to provide feedback to our neural network on the unlabelled data, combining it with the labelled data predictions as in multitask learning (Section 2.5.3) to form a semi-supervised learning algorithm. A popular method on enforcing consistency is *virtual adversarial training* [216], which enforces consistency across carefully chosen image perturbations. Another paper, *unsupervised data augmentation* [352], uses standard data augmentation techniques such as cutout [61] for images and back translation for text [283] to perturb images and enforces consistency across them. [367] uses consistency constraints along with other semi-supervised and self-supervised techniques in its full algorithm.

### Co-training

Another way to provide feedback on unlabelled data is to train two (many) neural network models, each on a different *view* of the raw data. For example, with text data, each model might see a different part of the input sentence. These models can then be given feedback to be maximally consistent with each other, or with a different model which sees all of the data, or even used for self-training, with each different model providing pseudo labels on the instances it is most confident on. This post `https://ruder.io/semi-supervised/` gives a nice overview of different co-training schemes, and [47, 246, 107] are some recent papers implementing this in text and images.

#### Semi-Supervised Learning Summary

Semi-supervised learning is a powerful way to reduce the need for labelled data and can significantly boost the efficacy of deep learning models. Semi-supervised learning can be applied in any situation where a meaningful task can be created on the unlabelled data. In this section we have overviewed some natural ways to define such tasks, but there may be many creative alternatives depending on the domain of interest. We hope the references will provide a helpful starting point for implementation and further exploration!

### 2.7.3 Data Augmentation

As depicted in Figure 2.1, *data augmentation* is an important part of the deep learning workflow. Data augmentation refers to the process of artificially increasing the size and diversity of the training data by applying a variety of transformations to the raw data instances. For example, if the raw instances were to consist of images, we might artificially *pad* out the image borders and then perform an off center (random) *crop* to give us the final augmented image instance. Aside from increasing the size and diversity of the data, data augmentation offers the additional benefit of encouraging the neural network to be robust to certain kinds of common transformations of data instances. In this section, we overview some of the most popular data augmentation techniques for image and sequential data. These techniques will typically already be part of many open sourced deep learning pipelines, or easy to invoke in any mainstream deep learning software package. There are also some specific libraries written for augmen-

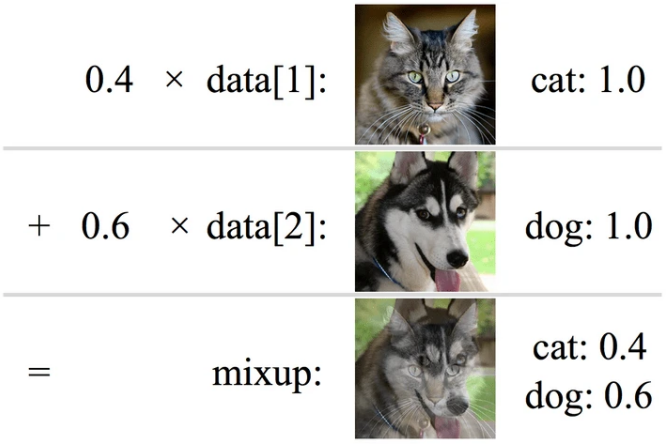



Figure 2.13: **An illustration of the Mixup data augmentation technique. Image source [54]** The figure provides an example of the Mixup data augmentation method — an image of a cat and an image of a dog are linearly combined, with 0.4 weight on the cat and 0.6 weight on the dog, to give a new input image shown in the bottom with a smoothed label of 0.4 weight on cat and 0.6 weight on dog. Mixup has been a very popular and successful data augmentation method for image tasks.

tations, for example *imgaug* https://github.com/aleju/imgaug, *nlpaug* https://github.com/makcedward/nlpaug and *albumentations* https://github.com/albumentations-team/albumentations.

### Data Augmentation for Image Data

Simple augmentations for image data consider transformations such as *horizontal flips* or *random crops* (padding the image borders and taking an off center crop.) Inspired by these simple methods are two very successful image augmentation strategies, *cutout* [61], which removes a patch from the input image, and RICAP [307], which combines patches from four different input image to create a new image (with new label a combination of the original labels.) This somewhat surprising latter technique of combining images has in fact shown to be very successful in *mixup* [372], another data augmentation strategy where linear

combinations of images (instead of patches) are used. (This strategy has also been combined with cutout in the recently proposed *cutmix* augmentation strategy [363], with code https://github.com/clovaai/CutMix-PyTorch.)

Other useful augmentation strategies include *TANDA* [258] which learns a model to compose data augmentations, the related *randaugment* [51], choosing a random subset of different possible augmentations, population based augmentation [123] which randomly searches over different augmentation policies, [131] applying color distortions to the image and the recently proposed *augmix* [120] (code https://github.com/google-research/augmix.)

#### Data Augmentation for Sequence Data

Data augmentation for sequential data typically falls into either (i) directly modifying the input sequence, or (ii) in the case of *sequence to sequence* tasks (Section 2.4.5), increasing the number of input-output sequences through noisy translation with the neural network. When directly modifying the input sequence, common perturbations include randomly deleting a sequence token (comparable to the masking approach used in [60]), swapping sets of sequence tokens, and replacing a token with its *synonym*. This latter strategy is usually guided by word embeddings [334] or contextualized word embeddings [157]. Examples of combining these transformations are given by [340, 142], with code repositories such as https://github.com/makcedward/nlpaug providing some simple implementations.

The other dominant approach to data augmentation of sequences is using

sequence-to-sequence models to generate new data instances, known as *back-translation* [283, 71]. Concretely, suppose we have a model to translate from English sequences to German sequences. We can take the output German sequence and use existing tools/noisy heuristics to translate it back to English. This gives us an additional English-German sequence pair.

### 2.7.4 Data (Image) Denoising

When measuring and collecting high dimensional data, noise can easily be introduced to the raw instances, be they images or single-cell data. As a result there has been significant interest and development of deep learning techniques to denoise the data. Many of these recent methods work even without paired noisy and clean data samples, and many be applicable in a broad range of settings. For instance, *Noise2Noise* [177] uses a *U-net* neural network architecture to denoise images given multiple noisy copies. The recent *Noise2Self* [23] (with code: `https://github.com/czbiohub/noise2self`) frames denoising as a self-supervision problem, using different subsets of features (with assumed independent noise properties) to perform denoising, applying it to both images as well as other high dimensional data.

## 2.8 Advanced Deep Learning Methods

The methods and tasks overviewed in the survey so far — supervised learning, fundamental neural network architectures (and their many different tasks), dif-

ferent paradigms like transfer learning as well as ways to reduce labelled data dependence such as self-supervision and semi-supervised learning — are an excellent set of first approaches for any problem amenable to deep learning. In most such problems, these approaches will also suffice in finding a good solution.

Occasionally however, it might be useful to turn to more advanced methods in deep learning, specifically *generative models* and *reinforcement learning*. We term these methods advanced as they are often more intricate to implement, and may require specific properties of the problem to be useful, for example an excellent environment model/simulator for reinforcement learning. We provide a brief overview of these methods below.

### 2.8.1 Generative Models

At a high level, generative modelling has two fundamental goals. Firstly, it seeks to *model* and enable *sampling* from high dimensional data distributions, such as natural images. Secondly, it looks to learn low(er) dimensional latent encodings of the data that capture key properties of interest.

To achieve the first goal, generative models take samples of the high dimensional distribution as input, for example, images of human faces, and learn some task directly on these data instances (e.g. encoding and then decoding the instance or learning to generate synthetic instances indistinguishable from the given data samples or generating values per-pixel using neighboring pixels as context). If generative modelling achieved *perfect* success at this first goal, it would make it possible to continuously sample 'free' data instances! Such perfect

Figure 2.14: **Human faces generated from scratch by StyleGAN2. Image source [144]** The figure shows multiple human face samples from StyleGAN2 [144]. While perfectly modelling and capture full diversity of complex data distributions like human faces remains challenging, the quality and fidelity of samples from recent generative models is very high.

success is extremely challenging, but the past few years has seen enormous progress in the diversity and fidelity of samples from the data distribution.

For the second goal, learning latent encodings of the data with different encoding dimensions correspond to meaningful factors of variation, having an explicit encoder-decoder structure in the model can be helpful in encouraging learning such representations. This is the default structure for certain kinds of generative models such as *variational autoencoders* [156] but has also been adopted into other models, such as BigBiGAN [66], a type of *generative adversarial network*. In the following sections we overview some of these main types of generative models.

### Generative Adversarial Networks

Arguably the most well known of all different types of generative models, Generative Adversarial Networks, commonly known as GANs, consist of two neural networks, a *generator* and a *discriminator*, which are pitted in a game against each other. The generator takes as input a *random noise vector* and tries to output samples that look like the data distribution (e.g. synthesize images of peoples faces),

while the discriminator tries to distinguish between true samples of the data, and those synthesized by the generator. First proposed in [93], GANs have been an exceptionally popular research area, with the most recent variations, such as BigGAN [32] (code: https://github.com/ajbrock/BigGAN-PyTorch), BigBiGAN [66] and StyleGAN(2) [144] (code: https://github.com/NVlabs/stylegan2) able to generate incredibly realistic images.

**Unconditional GANs vs Conditional GANs** The examples given above are all *unconditional* GANs, where the data is generated with only a random noise vector as input. A popular and highly useful variant are *conditional* GANs, where generation is conditioned on additional information, such as a label, or a 'source' image, which might be *translated* to a different style. Examples include *pix2pix* [139] (code: https://phillipi.github.io/pix2pix/), cycleGAN [379], and applications of these to videos [38].

GANs have found many scientific applications, from performing data augmentation in medical image settings [90] to protein generation [265]. The 'adversarial' loss objective of GANs can make them somewhat tricky to train, and useful implementation advice is given in https://www.fast.ai/2019/05/03/decrappify/, and (for conditional GANs) is included in https://github.com/jantic/DeOldify.

### Variational Autoencoders

Another type of generative model is given by the *variational autoencoder*, first proposed by [156]. VAEs have an encoder decoder structure, and thus an explicit

latent encoding, which can capture useful properties of the data distribution. They also enable estimation of the *likelihood* of a sampled datapoint — the probability of its occurrence in the data distribution. VAEs have also been extremely popular, with many variations and extensions proposed [296, 143, 153, 99]. Because of the explicit latent encoding and the ability to estimate likelihoods, they have also found use cases in various scientific settings, such as for modelling gene expression in single-cell RNA sequencing [196].

### Autoregressive Models

Yet another type of generative model is *autoregressive models*, which take in inputs sequentially and use those to generate an appropriate output. For instance, such models may take in a sequence of pixel values (some of them generated at a previous timestep) and use these to generate a new pixel value for a specific spatial location. Autoregressive models such as PixelRNN [234], PixelCNN (and variants) [322, 275] and the recently proposed VQ-VAE(2) [260] (code: `https://github.com/rosinality/vq-vae-2-pytorch`) offer very high generation quality.

### Flow Models

A relatively new class of generative models, *flow models*, looks at performing generation using a sequence of invertible transformations, which enables the computation of *exact likelihoods*. First proposed in [63, 64], performing an expressive but tractable sequence of invertible transformations is an ac-

tive area of research [155, 124]. A nice introduction to normalizing flows is given in this short video tutorial https://www.youtube.com/watch?v=i7LjDvsLWCg&feature=youtu.be.

### 2.8.2 Reinforcement Learning

Reinforcement learning has quite a different framing to the techniques and methods introduced so far, aiming to solve the *sequential decision making* problem. It is typically introduced with the notions of an *environment* and an *agent*. The agent can take a *sequence of actions* in the environment, each of which affect the environment *state* in some way, and also result in possible *rewards* (feedback) — 'positive' for good sequences of actions resulting in a 'good' state and 'negative' for bad sequences of actions leading to a 'bad' state. For example, in a game like chess, the state is the current position of all pieces in play (the game state), an action the moving of a piece, with a good sequence of actions resulting in a win, a bad sequence of actions in a loss and the reward might be one or zero depending on having a win or loss respectively.

With this being the setup, the goal of reinforcement learning is to learn, through interaction with the environment, good sequences of actions (typically referred to as a *policy*). Unlike supervised learning, feedback (the reward) is typically given only after performing the *entire sequence* of actions. Specifically, feedback is *sparse* and *time delayed*. There are a variety of different reinforcement learning use cases depending on the specifics of the problem.

**RL with an Environment Model/Simulator**

Some of the most striking results with RL, such as AlphaGoZero [290], critically use an environment model/simulator. In such a setting, a variety of learning algorithms [338, 280, 186] (some code: `https://github.com/openai/baselines`) can help the agent learn a good sequence of actions, often through simultaneously learning a *value function* — a function that determines whether a particular *environment state* is beneficial or not. Because the benefit of an environment state may depend on the entire sequence of actions (some still in the future), RL is very important in properly assessing the value of the environment state, through implicitly accounting for possible future actions. Combining value functions with traditional search algorithms has been a very powerful way to use RL, and may be broadly applicable to many domains.

Specifically, if developing a solution to the problem is multistep in nature, with even a noisy validation possible in simulation, using RL to learn a good value function and combining that with search algorithms may lead to discovering new and more effective parts of the search space. Approaches like these have gained traction in considering RL applications to fundamental problems in both computer systems, with [204] providing a survey and a new benchmark, and machine learning systems [241], in designing task-specific neural network models. The latter has recently also resulted in scientific use cases — designing neural networks to emulate complex processes across astronomy, chemistry, physics, climate modelling and others [145].

**RL without Simulators**

In other settings, we don't have access to an environment model/simulator, and may simply have records of sequences of actions (and the ensuing states and rewards). This is the *offline* setting. In this case, we may still try to teach an agent a good policy, using the observed sequences of actions/states/rewards in conjunction with *off-policy* methods [287, 249, 192], but thorough validation and evaluation can be challenging. Evaluation in off-policy settings often uses a statistical technique known as *off-policy policy evaluation* (example algorithms include [245, 191]). In robotics, reinforcement learning literature has looked at performing transfer learning between policies learned in simulation and policies learned on real data [273]. A thorough overview of deep reinforcement learning is given in http://rail.eecs.berkeley.edu/deeprlcourse/.

## 2.9 Implementation Tips

In this section, we highlight some useful tips for implementing these models.

**Explore Your Data** Before starting with steps in the learning phase (see Figure 2.1), make sure to perform a thorough exploration of your data. What are the results of simple dimensionality reduction methods or clustering? Are the labels reliable? Is there *imbalance* amongst different classes? Are different subpopulations appropriately represented?

**Try Simple Methods** When starting off with a completely new problem, it is useful to try the simplest version possible. (It might even be worthwhile starting with no learning at all — how does the naive *majority baseline* perform? For datasets with large imbalances, it may be quite strong!) If the dataset is very large, is there some smaller subsampled/downscaled version that can be used for faster preliminary testing? What is the simplest model that might work well? How does a majority baseline perform? (This ties in settings where the data has class imbalance.) Does the model (as expected) overfit to very small subsets of the data?

**Where possible, start with well tested models/tasks/methods** With the plethora of standard models (many of them pretrained), data augmentation, and optimization methods readily available (Section 2.3), most new problems will be amenable to some standard set of these choices. *Start with this!* Debugging the dataset and objective function associated with a new problem at the same time as debugging the neural network model, task choice, optimization algorithm, etc is very challenging.

Additionally, many of the standard model/task/method choices are very well benchmarked, and exploring performance in these settings is an excellent first step in understanding the inherent challenges of the new problem. Wherever possible, the easiest way to get starting with the learning phase is to clone an appropriate github repository that has the models and training code needed, and make the minimal edits needed to work with the new dataset and objective function.

**First Steps in Debugging Poor Performance** Having put together an end-to-end system, you observe that it is not performing well on the validation data. What is the reason? Before getting into more subtle design questions on hyperparameter choice (below), some first things to look at might be (i) Is the model overfitting? If so, more regularization, data augmentation, early stopping, smaller model may help. (ii) Is there a *distribution shift* between the training and validation data? (iii) Is the model underfitting? If so, check the optimization process by seeing if the model *overfits when trained on a smaller subset of the training data*. Test out a simpler task. Check for noise in the labels or data instances and for distribution shift. (iv) Look at the instances on which the model makes errors. Is there some pattern? For imbalanced datasets, loss function reweighting or more augmentation on the rarer classes can help. (v) How stable is the model performance across multiple random reruns? (vi) What are gradient and intermediate representation norms through the training process?

**Which hyperparameters matter most?** A big challenge in improving deep learning performance is the multitude of hyperparameters it is possible to change. In practice, some of the simplest hyperparameters often affect performance the most, such as learning rate and learning rate schedule. Picking an optimizer with subtleties such as weight decay correctly implemented can also be very important, see this excellent article on a very popular optimizer, AdamW https://www.fast.ai/2018/07/02/adam-weight-decay/. It might also be very useful to visualize the contributions to total loss from the main objective function vs different regularizers such as weight decay.

Other hyperparameters that can be explored include batch size and data

preprocessing, though if standard setups are used for these, varying learning rate related hyperparameters is likely to be the first most useful aspect to explore. To test different hyperparameter settings, it can be very useful to *cross-validate*: hold out a portion of the training data, train different hyperparameter settings on the remaining data, pick whichever hyperparameter setting does best when evaluated on the held out data, and then finally retrain that hyperparameter setting on the full training dataset.

**Validate your model thoroughly!** Deep learning models are notorious for relying on *spurious correlations* in the data to perform their predictions [15, 228, 346]. By spurious correlation, we mean features in the data instances that happen to co-occur with a specific label, but will not result in a robust, generalizable model. For example, suppose we have data from different chest x-ray machines (corresponding to different hospitals) that we put together to train a deep learning model. It might be the case that one of these machines, so happens to scan many sick patients. The deep learning model might then implicitly learn about the chest x-ray machine instead of the features of the illness. One of the best tests for ensuring the model is learning in a generalizable way is to evaluate the model on data collected *separately* from the training data, which will introduce some natural *distribution shift* and provide a more robust estimate of its accuracy. Some recent interesting papers exploring these questions include [119, 261].

Relatedly, deep neural networks will also pick up on any biases in the data, for example, learning to pay attention to gender (a sensitive attribute) when made to predict age due to class imbalances leading to spurious correlations. This can pose significant challenges for generalizable conclusions in scientific

settings where data may be collected from one population, but the predictions must be accurate across all populations. It is therefore important to perform postprocessing analysis on the model representations to identify the presence of such biases. A line of active research studies how to debias these representations [9, 337].

**Implementation References** Some of the general design considerations when coming to implementation (along with factors affecting larger scale deployment, not explored in this survey) are discussed in this overview https://github.com/chiphuyen/machine-learning-systems-design/blob/master/build/build1/consolidated.pdf.

For specific points on training and debugging deep learning systems, two excellent guides are given by http://josh-tobin.com/assets/pdf/troubleshooting-deep-neural-networks-01-19.pdf and http://karpathy.github.io/2019/04/25/recipe/.

## 2.10 Conclusion

In this survey chapter, we have overviewed many of the highly successful deep learning models, tasks and methodologies, with references to the remarkably comprehensive open-sourced resources developed by the community. We hope it serves to set the research results of the subsequent chapters in context and helps inspire exploration of new directions and applications of these versatile techniques.

# Part II

# Insights on Neural Network Hidden Representations

CHAPTER 3

# QUANTITATIVE TECHNIQUES FOR INSIGHTS ON DEEP REPRESENTATIONS

In the following two chapters we introduce quantitative techniques to study the *hidden representations* of deep neural networks. While neural network hidden representations contain the bulk of the parameters, computation and learning, their complexity and lack of any interpretable mapping makes them challenging to study. One basic question is being able to meaningfully *compare* hidden representations across neural networks, which, despite being fundamental, has subtlety on how a hidden representation is even defined, and addressing the lack of any meaningful interpretable (or comparable) mapping of representations across neural networks

## 3.1 Overview of SVCCA and Analysis Insights

We begin by introducing a new approach based on an analysis of each neuron's *activation vector* – the scalar outputs it emits on input datapoints. With this interpretation of neurons as vectors (and layers as subspaces, spanned by neurons), we introduce SVCCA, Singular Vector Canonical Correlation Analysis, an amalgamation of Singular Value Decomposition and Canonical Correlation Analysis (CCA) [129], as a powerful method for analyzing deep representations.

The main contributions resulting from the introduction of SVCCA are the following:

1. We ask: is the dimensionality of a layer's learned representation the same as the number of neurons in the layer? *Answer: No.* We show that trained networks perform equally well with a number of directions just a fraction of the number of neurons with no additional training, provided they are carefully chosen with SVCCA (Section 3.2.1). We explore the consequences for model compression (Section 3.4.4).
2. We ask: what do deep representation learning dynamics look like? *Answer: Networks broadly converge bottom up.* Using SVCCA, we compare layers across time and find they solidify from the bottom up. This suggests a simple, computationally more efficient method of training networks, *Freeze Training*, where lower layers are sequentially frozen after a certain number of timesteps (Sections 3.4.1, 3.4.2).
3. We develop a method based on the discrete Fourier transform which greatly speeds up the application of SVCCA to convolutional neural networks (Section 3.3).
4. We also explore an interpretability question, of when an architecture becomes sensitive to different classes. We find that SVCCA captures the semantics of different classes, with similar classes having similar sensitivities, and vice versa. (Section 3.4.3).

**Experimental Details** Most of our experiments are performed on CIFAR-10 (augmented with random translations). The main architectures we use are a convolutional network and a residual network. To produce a few figures, we also use a toy regression task: training a four hidden layer fully connected network with 1D input and 4D output, to regress on four different simple functions.

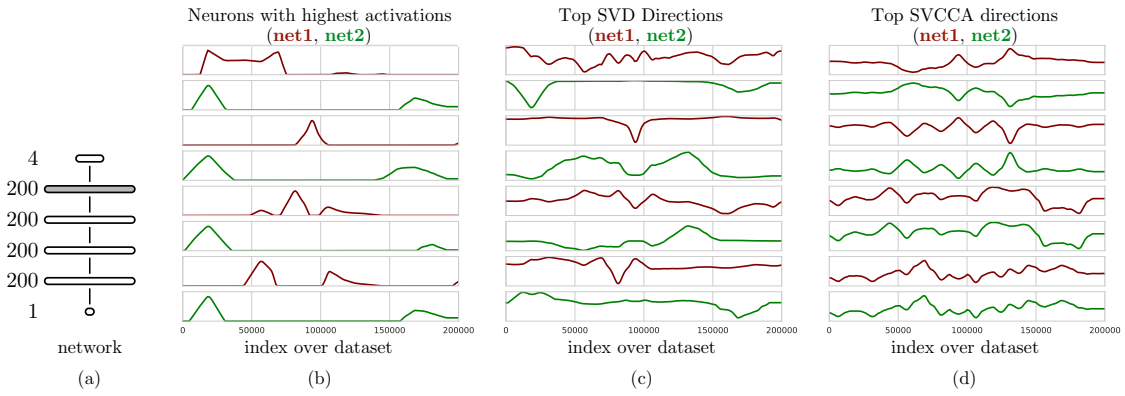



Figure 3.1: To demonstrate SVCCA, we consider a toy regression task (regression target as in Figure 3.3). **(a)** We train two networks with four fully connected hidden layers starting from different random initializations, and examine the representation learned by the penultimate (shaded) layer in each network. **(b)** The neurons with the highest activations in net 1 (maroon) and in net 2 (green). The x-axis indexes over the dataset: in our formulation, the *representation* of a neuron is simply its value over a dataset. **(c)** The SVD directions — i.e. the directions of maximal variance — for each network. **(d)** The top SVCCA directions. We see that each pair of maroon/green lines (starting from the top) are almost visually identical (up to a sign). Thus, although looking at just neurons (b) seems to indicate that the networks learn very different representations, looking at the SVCCA subspace (d) shows that the information in the representations are (up to a sign) nearly identical.

## 3.2 Measuring Representations in Neural Networks

Our goal in this paper is to analyze and interpret the representations learned by neural networks. The critical question from which our investigation departs is: how should we define the representation of a neuron? Consider that a neuron at a particular layer in a network computes a real-valued function over the network's input domain. In other words, if we had a lookup table of all possible input $\rightarrow$ output mappings for a neuron, it would be a complete portrayal of that neuron's functional form.

However, such infinite tables are not only practically infeasible, but are also problematic to process into a set of conclusions. Our primary interest is not in

the neuron's response to random data, but rather in how it represents features of a specific dataset (e.g. natural images). Therefore, in this study we take *a neuron's representation to be its set of responses over a finite set of inputs* — those drawn from some training or validation set.

More concretely, for a given dataset $X = \{x_1, \cdots x_m\}$ and a neuron $i$ on layer $l$, $\boldsymbol{z}_i^l$, we *define* $\boldsymbol{z}_i^l$ to be the *vector* of outputs on $X$, i.e.

$$\boldsymbol{z}_i^l = (\boldsymbol{z}_i^l(x_1), \cdots, \boldsymbol{z}_i^l(x_m))$$

Note that this is a different vector from the often-considered vector of the "representation at a layer of a single input." Here $\boldsymbol{z}_i^l$ is a *single* neuron's response over the entire dataset, not an entire layer's response for a single input. In this view, a neuron's representation can be thought of as a single vector in a high-dimensional space. Broadening our view from a single neuron to the collection of neurons in a layer, the layer can be thought of as the set of neuron vectors contained within that layer. This set of vectors will span some subspace. To summarize:

*Considered over a dataset X with m examples, a neuron is a vector in* $\mathbb{R}^m$*.*
*A layer is the subspace of* $\mathbb{R}^m$ *spanned by its neurons' vectors.*

Within this formalism, we introduce *Singular Vector Canonical Correlation Analysis* (SVCCA) as a method for analysing representations. SVCCA proceeds as follows:

- **Input:** SVCCA takes as input two (not necessarily different) sets of neurons (typically layers of a network) $l_1 = \{\boldsymbol{z}_1^{l_1}, ..., \boldsymbol{z}_{m_1}^{l_1}\}$ and $l_2 = \{\boldsymbol{z}_1^{l_2}, ..., \boldsymbol{z}_{m_2}^{l_2}\}$

- **Step 1** First SVCCA performs a singular value decomposition of each subspace to get sub-subspaces $l'_1 \subset l_1, l'_2 \subset l_2$ which comprise of the most important directions of the original subspaces $l_1, l_2$. In general we take enough directions to explain 99% of variance in the subspace. This is especially important in neural network representations, where as we will show many low variance directions (neurons) are primarily noise.
- **Step 2** Second, compute the Canonical Correlation similarity ([129]) of $l'_1, l'_2$: linearly transform $l'_1, l'_2$ to be as aligned as possible and compute correlation coefficients. In particular, given the output of step 1, $l'_1 = \{\boldsymbol{z'}^{l_1}_1, ..., \boldsymbol{z'}^{l_1}_{m'_1}\}, l'_2 = \{\boldsymbol{z'}^{l_2}_1, ..., \boldsymbol{z'}^{l_2}_{m'_2}\}$, CCA linearly transforms these subspaces $\tilde{l}_1 = W_X l'_1$, $\tilde{l}_2 = W_Y l'_2$ such as to maximize the correlations $corrs = \{\rho_1, \dots \rho_{\min(m'_1, m'_2)}\}$ between the transformed subspaces.
- **Output:** With these steps, SVCCA outputs pairs of aligned directions, $(\tilde{\boldsymbol{z}}^{l_1}_i, \tilde{\boldsymbol{z}}^{l_2}_i)$ and how well they correlate, $\rho_i$. Step 1 also produces intermediate output in the form of the top singular values and directions.

For a more detailed description of each step, see the Appendix. SVCCA can be used to analyse any two sets of neurons. In our experiments, we utilize this flexibility to compare representations across different random initializations, architectures, timesteps during training, and specific classes and layers.

Figure 3.1 shows a simple, intuitive demonstration of SVCCA. We train a small network on a toy regression task and show each step of SVCCA, along with the resulting very similar representations. SVCCA is able to find hidden similarities in the representations.

### 3.2.1 Distributed Representations

An important property of SVCCA is that it is truly a *subspace* method: both SVD and CCA work with $\text{span}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m)$ instead of being axis aligned to the $\boldsymbol{z}_i$ directions. SVD finds singular vectors $\boldsymbol{z}'_i = \sum_{j=1}^{m} s_{ij} \boldsymbol{z}_j$, and the subsequent CCA finds a linear transform $W$, giving orthogonal canonically correlated directions $\{\tilde{\boldsymbol{z}}_1, \ldots, \tilde{\boldsymbol{z}}_m\} = \{\sum_{j=1}^{m} w_{1j} \boldsymbol{z}'_j, \ldots, \sum_{j=1}^{m} w_{mj} \boldsymbol{z}'_j\}$. In other words, SVCCA has no preference for representations that are neuron (axes) aligned.

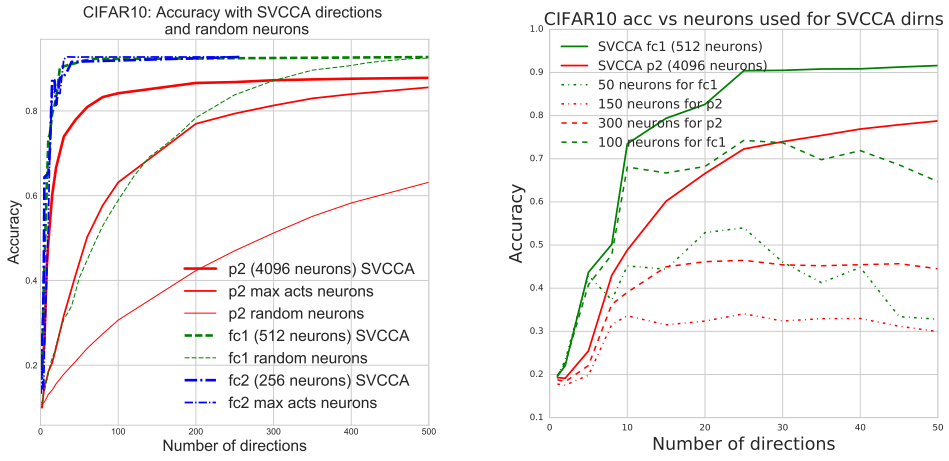
If representations are distributed across many dimensions, then this is a desirable property of a representation analysis method. Previous studies have reported that representations may be more complex than either fully distributed or axis-aligned [306, 183] but this question remains open.

We use SVCCA as a tool to probe the nature of representations via two experiments:

(a) We find that the subspace directions found by SVCCA are disproportionately important to the representation learned by a layer, relative to neuron-aligned directions.

(b) We show that at least some of these directions are distributed across many neurons.

Experiments for (a), (b) are shown in Figure 3.2 as (a), (b) respectively. For both experiments, we first acquire two different representations, $l_1, l_2$, for a layer $l$ by training two different random initializations of a convolutional network on CIFAR-10. We then apply SVCCA to $l_1$ and $l_2$ to get directions $\{\tilde{\boldsymbol{z}}_1^{l_1}, ..., \tilde{\boldsymbol{z}}_m^{l_1}\}$ and




(a) (b)

Figure 3.2: Demonstration of *(a)* disproportionate importance of SVCCA directions, and *(b)* distributed nature of some of these directions. For both panes, we first find the top $k$ SVCCA directions by training two conv nets on CIFAR-10 and comparing corresponding layers. *(a)* We project the output of the top three layers, pool1, fc1, fc2, onto this top-$k$ subspace. We see accuracy rises rapidly with increasing $k$, with even $k \ll$ num neurons giving reasonable performance, with *no* retraining. Baselines of random $k$ neuron subspaces and max activation neurons require larger $k$ to perform as well. *(b)*: after projecting onto top $k$ subspace (like left), dotted lines then project again onto $m$ neurons, chosen to correspond highly to the top $k$-SVCCA subspace. Many more neurons are needed than $k$ for better performance, suggesting distributedness of SVCCA directions.

$\{\tilde{\mathbf{z}}_1^{l_2}, ..., \tilde{\mathbf{z}}_m^{l_2}\}$, ordered according to importance by SVCCA, with each $\tilde{\mathbf{z}}_j^{l_i}$ being a linear combination of the original neurons, i.e. $\tilde{\mathbf{z}}_j^{l_i} = \sum_{r=1}^{m} \alpha_{jr}^{(l_i)} \mathbf{z}_r^{l_i}$.

For different values of $k < m$, we can then restrict layer $l_i$'s output to lie in the subspace of span$(\tilde{\mathbf{z}}_1^{l_i}, \ldots, \tilde{\mathbf{z}}_k^{l_i})$, the most useful $k$-dimensional subspace as found by SVCCA, done by projecting each neuron into this $k$ dimensional space.

We find — somewhat surprisingly — that very few SVCCA directions are required for the network to perform the task well. As shown in Figure 3.2(a), for a network trained on CIFAR-10, the first 25 dimensions provide nearly the same accuracy as using all 512 dimensions of a fully connected layer with 512 neurons. The accuracy curve rises rapidly with the first few SVCCA directions,

and plateaus quickly afterwards, for $k \ll m$. This suggests that the useful information contained in $m$ neurons is well summarized by the subspace formed by the top $k$ SVCCA directions. Two baselines for comparison are picking random and maximum activation neuron aligned subspaces and projecting outputs onto these. Both of these baselines require far more directions (in this case: neurons) before matching the accuracy achieved by the SVCCA directions. These results also suggest approaches to model compression, which are explored in more detail in Section 3.4.4.

Figure 3.2(b) next demonstrates that these useful SVCCA directions are at least somewhat distributed over neurons rather than axis-aligned. First, the top $k$ SVCCA directions are picked and the representation is projected onto this subspace. Next, the representation is further projected onto $m$ neurons, where the $m$ are chosen as those most important to the SVCCA directions. The resulting accuracy is plotted for different choices of $k$ (given by x-axis) and different choices of $m$ (different lines). That, for example, keeping even 100 fc1 neurons (dashed green line) cannot maintain the accuracy of the first 20 SVCCA directions (solid green line at x-axis 20) suggests that those 20 SVCCA directions are distributed across 5 or more neurons each, on average. Figure 3.3 shows a further demonstration of the effect on the output of projecting onto top SVCCA directions, here for the toy regression case.

**Why the two step SV + CCA method is needed.** Both SVD and CCA have important properties for analysing network representations and SVCCA consequently benefits greatly from being a two step method. CCA is **invariant** to affine transformations, enabling comparisons without natural alignment (e.g. different

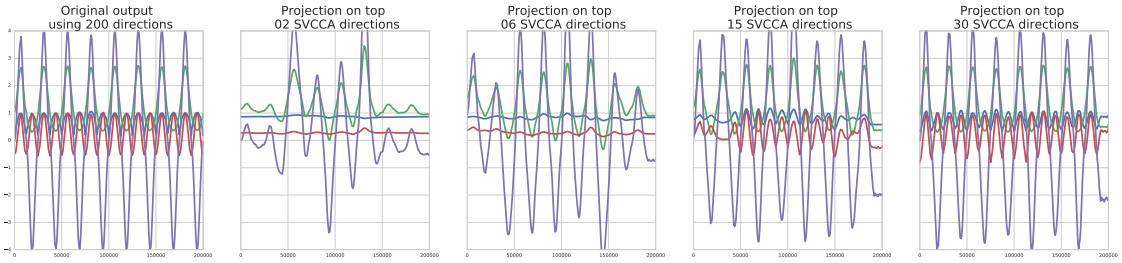



Figure 3.3: The effect on the output of a latent representation being projected onto top SVCCA directions in the toy regression task. Representations of the penultimate layer are projected onto $2, 6, 15, 30$ top SVCCA directions (from second pane). By 30, the output looks very similar to the full 200 neuron output (left).

architectures, Section 3.4.4). See Appendix A.2 for proofs and a demonstrative figure. While CCA is a powerful method, it also suffers from certain shortcomings, particularly in determining how many directions were important to the original space $X$, which is the strength of SVD. See Appendix for an example where naive CCA performs badly. Both the SVD and CCA steps are critical to the analysis of learning dynamics in Section 3.4.1.

## 3.3 Scaling SVCCA for Convolutional Layers

Applying SVCCA to convolutional layers can be done in two natural ways:

(1) *Same layer comparisons:* If $X, Y$ are the same layer (at different timesteps or across random initializations) receiving the same input we can concatenate along the pixel (height $h$, width $w$) coordinates to form a vector: a conv layer $h \times w \times c$ maps to $c$ vectors, each of dimension $hwd$, where $d$ is the number of datapoints. This is a natural choice because neurons at different pixel coordinates see *different* image data patches to each other. When $X, Y$ are two versions of the same layer, these $c$ different views correspond perfectly.

(2) *Different layer comparisons:* When $X, Y$ are not the same layer, the image patches seen by different neurons have no natural correspondence. But we can flatten an $h \times w \times c$ conv into $hwc$ neurons, each of dimension $d$. This approach is valid for convs in different networks or at different depths.

### 3.3.1 Scaling SVCCA with Discrete Fourier Transforms

Applying SVCCA to convolutions introduces a computational challenge: the number of neurons ($h \times w \times c$) in convolutional layers, especially early ones, is very large, making SVCCA prohibitively expensive due to the large matrices involved. Luckily the problem of approximate dimensionality reduction of large matrices is well studied, and efficient algorithms exist, e.g. [106].

For convolutional layers however, we can avoid dimensionality reduction and perform *exact* SVCCA, even for large networks. This is achieved by preprocessing each channel with a Discrete Fourier Transform (which preserves CCA due to invariances, see Appendix), causing all (covariance) matrices to be block-diagonal. This allows all matrix operations to be performed block by block, and only over the diagonal blocks, vastly reducing computation. We show:

**Theorem 1.** *Suppose we have a translation invariant (image) dataset $X$ and convolutional layers $l_1, l_2$. Letting $DFT(l_i)$ denote the discrete fourier transform applied to each channel of $l_i$, the covariance $cov(DFT(l_1), DFT(l_2))$ is block diagonal, with blocks of size $c \times c$.*

We make only two assumptions: 1) all layers below $l_1, l_2$ are either conv or pooling layers with circular boundary conditions (translation equivariance) 2)

The dataset $X$ has all translations of the images $X_i$. This is necessary in the proof for certain symmetries in neuron activations, but these symmetries typically exist in natural images even without translation invariance, as shown in Figure App.2 in the Appendix. Below are key statements, with proofs in Appendix.

Say a single channel image dataset $X$ of images is *translation invariant* if for any (wlog $n \times n$) image $X_i \in X$, with pixel values $\{z_{11}, \ldots z_{nn}\}$, $X_i^{(a,b)} = \{z_{\sigma_a(1)\sigma_b(1)}, \ldots z_{\sigma_a(n)\sigma_b(n)}\}$ is also in $X$, for all $0 \leq a, b \leq n-1$, where $\sigma_a(i) = a + i \mod n$ (and similarly for $b$).

For a multiple channel image $X_i$, an $(a, b)$ translation is an $(a, b)$ height/width shift on every channel separately. $X$ is then translation invariant as above.

To prove Theorem 1, we first show another theorem:

**Theorem 2.** *Given a translation invariant dataset $X$, and a convolutional layer $l$ with channels $\{c_1, \ldots c_k\}$ applied to $X$*

*(a) the DFT of $c_i$, $FcF^T$ has diagonal covariance matrix (with itself).*
*(b) the DFT of $c_i, c_j$, $Fc_iF^T$, $Fc_jF^T$ have diagonal covariance with each other.*

Finally, both of these theorems rely on properties of *circulant matrices* and their DFTs:

**Theorem 3.** *The covariance matrix of $c_i$ applied to translation invariant $X$ is circulant and block circulant.*

**Theorem 4.** *The DFT of a circulant matrix is diagonal.*

## 3.4 Applications of SVCCA

### 3.4.1 Learning Dynamics with SVCCA

We can use SVCCA as a window into learning dynamics by comparing the representation at a layer at different points during training to its final representation. Furthermore, as the SVCCA computations are relatively cheap to compute compared to methods that require training an auxiliary network for each comparison [7, 183], we can compare all layers during training at all timesteps to all layers at the final time step, producing a rich view into the learning process.

The outputs of SVCCA are the aligned directions $(\tilde{x}_i, \tilde{y}_i)$, how well they align, $\rho_i$, as well as intermediate output from the first step, of singular values and directions, $\lambda_X^{(i)}, x'^{(i)}, \lambda_Y^{(j)}, y'^{(j)}$. We condense these outputs into a single value, the *SVCCA similarity* $\bar{\rho}$, that encapsulates how well the representations of two layers are aligned with each other,

$$\bar{\rho} = \frac{1}{\min(m_1, m_2)} \sum_i \rho_i, \tag{3.1}$$

where $\min(m_1, m_2)$ is the size of the smaller of the two layers being compared. The SVCCA similarity $\bar{\rho}$ is the average correlation across aligned directions, and is a direct multidimensional analogue of Pearson correlation.

The SVCCA similarity for all pairs of layers, and all time steps, is shown in Figure 3.4 for a convnet and a resnet architecture trained on CIFAR10.

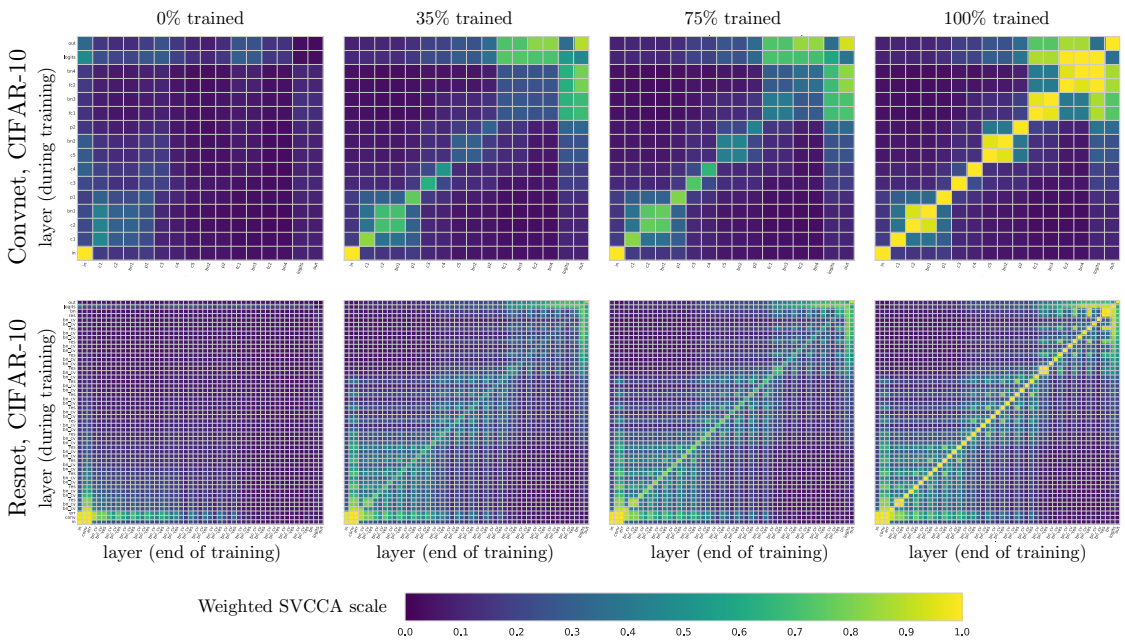



Figure 3.4: Learning dynamics plots for conv (top) and res (bottom) nets trained on CIFAR-10. Each pane is a matrix of size layers × layers, with each entry showing the SVCCA similarity $\bar{\rho}$ between the two layers. Note that learning broadly happens 'bottom up' – layers closer to the input seem to solidify into their final representations with the exception of the very top layers. Per layer plots are included in the Appendix. Other patterns are also visible – batch norm layers maintain nearly perfect similarity to the layer preceding them due to scaling invariance (with a slight reduction since batch norm changes the SVD directions which capture 99% of the variance). In the resnet plot, we see a stripe like pattern due to skip connections inducing high similarities to previous layers.

### 3.4.2 Freeze Training

Observing in Figure 3.4 that networks broadly converge from the bottom up, we propose a training method where we successively *freeze* lower layers during training, only updating higher and higher layers, saving *all* computation needed for deriving gradients and updating in lower layers.

We apply this method to convolutional and residual networks trained on CIFAR-10, Figure 3.5, using a linear freezing regime: in the convolutional network, each layer is frozen at a fraction (layer number/total layers) of total training

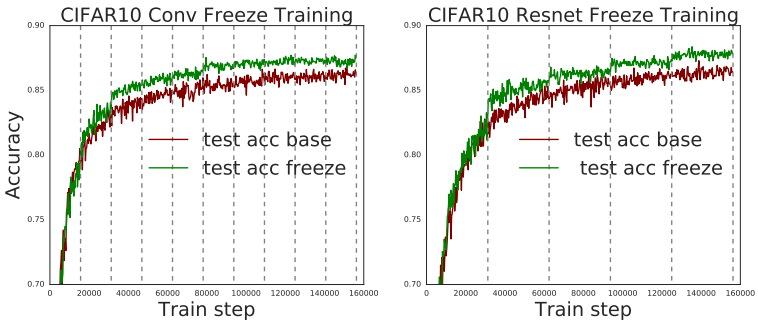



Figure 3.5: Freeze Training reduces training cost and improves generalization. We apply Freeze Training to a convolutional network on CIFAR-10 and a residual network on CIFAR-10. As shown by the grey dotted lines (which indicate the timestep at which another layer is frozen), both networks have a 'linear' freezing regime: for the convolutional network, we freeze individual layers at evenly spaced timesteps throughout training. For the residual network, we freeze entire residual blocks at each freeze step. The curves were averaged over ten runs.

time, while for resnets, each residual block is frozen at a fraction (block number/total blocks). The vertical grey dotted lines show which steps have another set of layers frozen. Aside from saving computation, Freeze Training appears to actively *help* generalization accuracy, like early stopping but with different layers requiring different stopping points.

### 3.4.3 Interpreting Representations: when are classes learned?

We also can use SVCCA to compare how correlated representations in each layer are with the logits of each class in order to measure how knowledge about the target evolves throughout the network. In Figure 3.6 we apply the DFT CCA technique on the Imagenet Resnet [117]. We take five different classes and for different layers in the network, compute the DFT CCA similarity between the logit of that class and the network layer. The results successfully reflect semantic aspects of the classes: the firetruck class sensitivity line is clearly distinct from

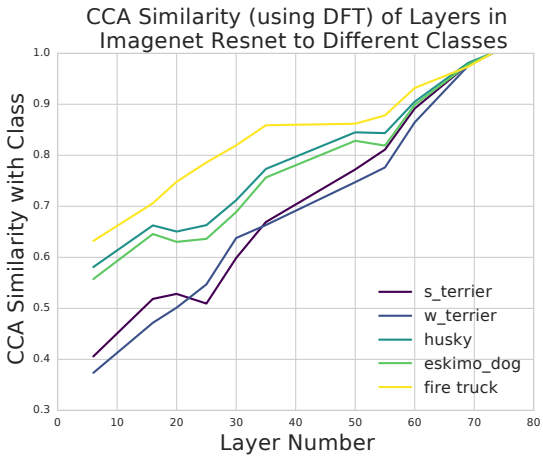



Figure 3.6: We plot the CCA similarity using the Discrete Fourier Transform between the logits of five classes and layers in the Imagenet Resnet. The classes are firetruck and two pairs of dog breeds (terriers and husky like dogs: husky and eskimo dog) that are chosen to be similar to each other. These semantic properties are captured in CCA similarity, where we see that the line corresponding to firetruck is clearly distinct from the two pairs of dog breeds, and the two lines in each pair are both very close to each other, reflecting the fact that each pair consists of visually similar looking images. Firetruck also appears to be *easier* for the network to learn, with greater sensitivity displayed much sooner.

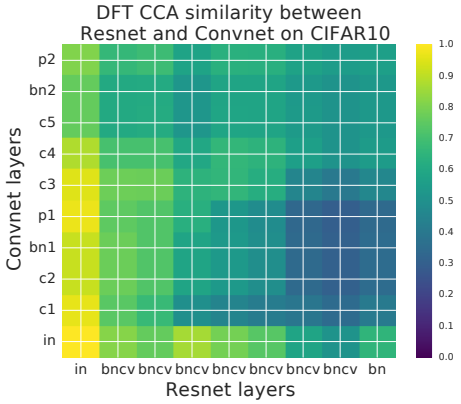



Figure 3.7: We plot the CCA similarity using the Discrete Fourier Transform between convolutional layers of a Resnet and Convnet trained on CIFAR-10. We find that the lower layers of both models are noticeably similar to each other, and get progressively less similar as we compare higher layers. Note that the highest layers of the resnet are least similar to the lower layers of the convnet.

the two pairs of dog breeds, and network develops greater sensitivity to firetruck earlier on. The two pairs of dog breeds, purposefully chosen so that each pair is similar to the other in appearance, have cca similarity lines that are very close to each other through the network, indicating these classes are similar to each other.

### 3.4.4 Other Applications: Cross Model Comparison and compression

SVCCA similarity can also be used to compare the similarity of representations across different random initializations, and even different architectures. We compare convolutional networks on CIFAR-10 across random initializations (Appendix) and also a convolutional network to a residual network in Figure 3.7, using the DFT method described in 3.3.

In Figure 3.3, we saw that projecting onto the subspace of the top few SVCCA directions resulted in comparable accuracy. This observations motivates an approach to model compression. In particular, letting the output vector of layer $l$ be $\boldsymbol{x}^{(l)} \in \mathbb{R}^{n \times 1}$, and the weights $W^{(l)}$, we replace the usual $W^{(l)}\boldsymbol{x}^{(l)}$ with $(W^{(l)}P_x^T)(P_x\boldsymbol{x}^{(l)})$ where $P_x$ is a $k \times n$ projection matrix, projecting $\boldsymbol{x}$ onto the top SVCCA directions. This bottleneck reduces both parameter count and inference computational cost for the layer by a factor $\sim \frac{k}{n}$. In Figure App.5 in the Appendix, we show that we can *consecutively* compress top layers with SVCCA by a significant amount (in one case reducing each layer to 0.35 original size) and hardly affect performance.

## 3.5 Chapter Summary

In this chapter we introduced SVCCA, an algorithm which enables the meaningful comparison of hidden representations between different neural network layers and architectures. Using SVCCA we obtain novel insights into the learning dynamics and learned representations of common neural network architectures.

These insights motivated a new Freeze Training technique which can reduce the number of flops required to train networks and potentially even increase generalization performance. We observe that CCA similarity can be a helpful tool for interpretability, with sensitivity to different classes reflecting their semantic properties. This technique also motivates a new algorithm for model compression.

CHAPTER 4

# IMPROVING ROBUSTNESS OF REPRESENTATION ANALYSIS AND APPLICATIONS TO GENERALIZATION

Building on the results of Chapter 3, we continue the investigation into the hidden (distributed) representations of neural networks. Specifically, we analyse the technique introduced in [253], and identify a key challenge: the method does not effectively distinguish between the signal and the noise in the representation. We address this via a better aggregation technique (Section 4.1.2). Building off of [221], we demonstrate that groups of networks which generalize converge to more similar solutions than those which memorize (Section 4.2.1), that wider networks converge to more similar solutions than narrower networks (Section 4.2.2), and that networks with identical topology but distinct learning rates converge to a small set of diverse solutions (Section 4.2.3). Using CCA to analyze RNN representations over training, we find that, as with CNNs [253], RNNs exhibit bottom-up convergence (Section 4.3.1). Across sequence timesteps, however, we find that RNN representations vary significantly (Section B.0.3).

## 4.1 Canonical Correlation Analysis on Neural Network Representations

Canonical Correlation Analysis [128], is a statistical technique for relating two sets of observations arising from an underlying process. It identifies the 'best' (maximizing correlation) linear relationships (under mutual orthogonality and norm constraints) between two sets of multidimensional variates.

Concretely, in our setting, the underlying process is a neural network being trained on some task. The multidimensional variates are *neuron activation vectors* over some dataset $X$. As in [253], a neuron activation vector denotes the outputs a single neuron $z$ has on $X$. If $X = \{x_1, ..., x_m\}$, then the neuron $z$ outputs scalars $z(x_1), ..., z(x_m)$, which can be stacked to form a vector.[1]

A single neuron activation vector is one multidimensional variate, and a layer of neurons gives us a *set* of multidimensional variates. In particular, we can consider two layers, $L_1$, $L_2$ of a neural network as two sets of observations, to which we can then apply CCA, to determine the similarity between two layers. Crucially, this similarity measure is invariant to (invertible) affine transforms of either layer, which makes it especially apt for neural networks, where the representation at each layer typically goes through an affine transform before later use. Most importantly, it also enables comparisons between *different* neural networks,[2] which is not naively possible due to a lack of any kind of neuron to neuron alignment.

### 4.1.1 Mathematical Details of Canonical Correlation

Here we overview the formal mathematical interpretation of CCA, as well as the optimization problem to compute it. Let $L_1, L_2$ be $a \times n$ and $b \times n$ dimensional matrices respectively, with $L_1$ representing $a$ multidimensional variates, and $L_2$ representing $b$ multidimensional variates. We wish to find vectors $w, s$ in $\mathbb{R}^a, \mathbb{R}^b$

[1] This is *different* than the vector of all neuron outputs on a *single* input: $z_1(x_1), ..., z_N(x_1)$, which is also sometimes referred to as an activation vector.

[2] Including those with different topologies such that $L_1$ and $L_2$ have different sizes.

respectively, such that the dot product

$$\rho = \frac{\langle w^T L_1, s^T L_2 \rangle}{\|w^T L_1\| \cdot \|s^T L_2\|}$$

is maximized. Assuming the variates in $L_1, L_2$ are centered, and letting $\Sigma_{L_1,L_1}$ denote the $a$ by $a$ covariance of $L_1$, $\Sigma_{L_2,L_2}$ denote the $b$ by $b$ covariance of $L_2$, and $\Sigma_{L_1,L_2}$ the cross covariance:

$$\frac{\langle w^T L_1, s^T L_2 \rangle}{\|w^T L_1\| \cdot \|s^T L_2\|} = \frac{w^T \Sigma_{L_1,L_2} s}{\sqrt{w^T \Sigma_{L_1,L_1} w} \sqrt{s^T \Sigma_{L_2,L_2} s}}$$

We can change basis, to $w = \Sigma_{L_1,L_1}^{-1/2} u$ and $s = \Sigma_{L_2,L_2}^{-1/2} v$ to get

$$\frac{w^T \Sigma_{L_1,L_2} s}{\sqrt{w^T \Sigma_{L_1,L_1} w} \sqrt{s^T \Sigma_{L_2,L_2} s}} = \frac{u^T \Sigma_{L_1,L_1}^{-1/2} \Sigma_{L_1,L_2} \Sigma_{L_2,L_2}^{-1/2} v}{\sqrt{u^T u} \sqrt{v^T v}} \tag{*}$$

which can be solved with a singular value decomposition:

$$\Sigma_{L_1,L_1}^{-1/2} \Sigma_{L_1,L_2} \Sigma_{L_2,L_2}^{-1/2} = U \Lambda V$$

with $u, v$ in (*) being the first left and right singular vectors, and the top singular value of $\Lambda$ corresponding to the canonical correlation coefficient $\rho \in [0, 1]$, which tells us how well correlated the vectors $w^T L_1 = u^T \Sigma_{L_1,L_1}^{-1/2} L_1$ and $s^T L_2 = v^T \Sigma_{L_2,L_2}^{-1/2} L_2$ (both vectors in $\mathbb{R}^n$) are.

In fact, $u, v, \rho$ are really the first in a series, and can be denoted $u^{(1)}, v^{(1)}, \rho^{(1)}$. Next in the series are $u^{(2)}, v^{(2)}$, the second left and right singular vectors, and $\rho^{(2)}$ the corresponding second highest singular value of $\Lambda$. $\rho^{(2)}$ denotes the correlation between $(u^{(2)})^T \Sigma_{L_1,L_1}^{-1/2} L_1$ and $(v^{(2)})^T \Sigma_{L_2,L_2}^{-1/2} L_2$, which is the next highest possible correlation under the constraint that $\langle u^{(1)}, u^{(2)} \rangle = 0$ and $\langle v^{(1)}, v^{(2)} \rangle = 0$.

The output of CCA is a series of singular vectors $u^{(i)}, v^{(i)}$ which are pairwise orthogonal, their corresponding vectors in $\mathbb{R}^n$: $(u^{(i)})^T \Sigma_{L_1,L_1}^{-1/2} L_1$ and $(v^{(i)})^T \Sigma_{L_2,L_2}^{-1/2} L_2$,

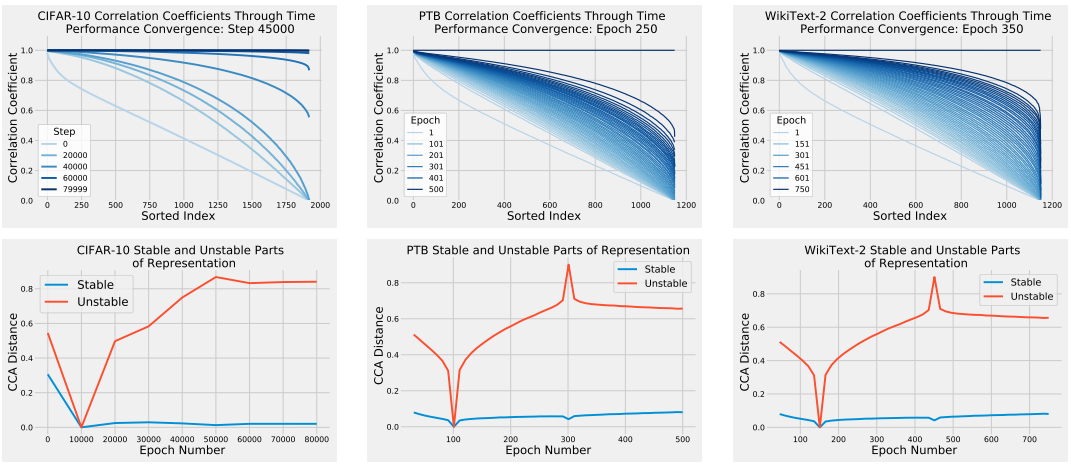



Figure 4.1: **CCA distinguishes between stable and unstable parts of the representation over the course of training.** Sorted CCA coefficients ($\rho_t^{(i)}$) comparing representations between layer $L$ at times $t$ through training with its representation at the final timestep $T$ for CNNs trained on CIFAR-10 (**a**), and RNNs trained on PTB (**b**) and WikiText-2 (**c**). For all of these networks, at time $t_0 < T$ (indicated in title), the performance converges to match final performance (see Figure App.1). However, many $\rho_t^{(i)}$ are unconverged, corresponding to unnecessary parts of the representation (noise). To distinguish between the signal and noise portions of the representation, we apply CCA between $L$ at timestep $t_{early}$ early in training, and $L$ at timestep $T/2$ to get $\rho_{T/2}$. We take the 100 top converged vectors (according to $\rho_{T/2}$) to form $S$, and the 100 least converged vectors to form $B$. We then compute CCA similarity between $S$ and $L$ at time $t > t_{early}$, and similarly for $B$. $S$ remains stable through training (signal), while $B$ rapidly becomes uncorrelated (**d-f**). Note that the sudden spike at $T/2$ in the unstable representation is because it is chosen to be the least correlated with step $T/2$.

and finally their correlation coefficient $\rho^{(i)} \in [0, 1]$, with $\rho^{(i)} \leq \rho^{(j)}, i > j$. Letting $c = \min(a, b)$, we end up with $c$ non-zero $\rho^{(i)}$.

Note that the orthogonality of $u^{(i)}, u^{(j)}$ also results in the orthogonality of $(u^{(i)})^T \Sigma_{L_1,L_1}^{-1/2} L_1, (u^{(j)})^T \Sigma_{L_1,L_1}^{-1/2} L_1$, as

$$\langle (u^{(i)})^T \Sigma_{L_1,L_1}^{-1/2} L_1, (u^{(j)})^T \Sigma_{L_1,L_1}^{-1/2} L_1 \rangle = (u^{(i)})^T \Sigma_{L_1,L_1}^{-1/2} L_1 L_1^T \Sigma_{L_1,L_1}^{-1/2} (u^{(j)}) = (u^{(i)})^T (u^{(j)}) = 0 \qquad (**)$$

and so our CCA directions are also orthogonal.

### 4.1.2 Beyond Mean CCA Similarity

To determine the representational similarity between two layers $L_1, L_2$, [253] prunes neurons with a preprocessing SVD step, and then applies CCA to $L_1, L_2$. They then represent the similarity of $L_1, L_2$ by the *mean* correlation coefficient. Adapting this to make a distance measure, $d_{SVCCA}(L_1, L_2)$:

$$d_{SVCCA}(L_1, L_2) = 1 - \frac{1}{c} \sum_{i=1}^{c} \rho^{(i)}$$

One drawback with this measure is that it implicitly assumes that all $c$ CCA vectors are equally important to the representations at layer $L_1$. However, there has been ample evidence that DNNs do not rely on the full dimensionality of a layer to represent high performance solutions [171, 75, 12, 219, 182, 221, 180]. As a result, the mean correlation coefficient will typically underestimate the degree of similarity.

To investigate this further, we first asked whether, over the course of training, all CCA vectors converge to their final representations before the network's performance converges. To test this, we computed the CCA similarity between layer $L$ at times $t$ throughout training with layer $L$ at the final timestep $T$. Viewing the sorted CCA coefficients $\rho$, we can see that many of the coefficients continue to change well after the network's performance has converged (Figure 4.1a-c, Figure App.1). This result suggests that the unconverged coefficients and their corresponding vectors may represent "noise" which is unnecessary for high network performance.

We next asked whether the CCA vectors which stabilize early in training

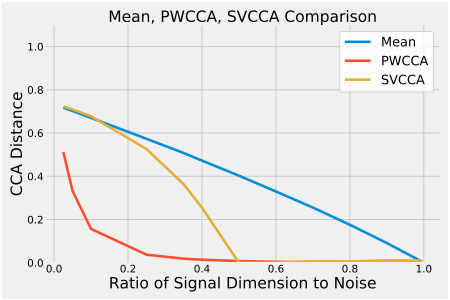



Figure 4.2: **Projection weighted (PWCCA) vs. SVCCA vs. unweighted mean** Unweighted mean (blue) and projection weighted mean (red) were used to compare synthetic ground truth signal and uncommon (noise) structure, each of fixed dimensionality. As the signal to noise ratio decreases, the unweighted mean underestimates the shared structure, while the projection weighted mean remains largely robust. SVCCA performs better than the unweighted mean but less well than the projection weighting.

remain stable. To test this, we computed the CCA vectors between layer $L$ at timestep $t_{early}$ in training and timestep $T/2$. We then computed the similarity between the top 100 vectors (those which stabilized early) and the bottom 100 vectors (those which had not stabilized) with the representation at all other training times. Consistent with our intuition, we found that those vectors which stabilized early remained stable, while the unstable vectors continued to vary, and therefore likely represent noise.

These results suggest that task-critical representations are learned by midway through training, while the noise only approaches its final value towards the end. We therefore suggest a simple and easy to compute variation that takes this into account. We also discuss an alternate approach in Section B.0.2.

**Projection Weighting** One way to address this issue is to replace the mean by a weighted mean, in which canonical correlations which are more important to the underlying representation have higher weight. We propose a simple method, *projection weighting*, to determine these weights. We base our proposition on the

hypothesis that CCA vectors that account for (loosely speaking) a larger proportion of the original outputs are likely to be more important to the underlying representation.

More formally, let layer $L_1$, have neuron activation vectors $[z_1, ..., z_a]$, and CCA vectors $h_i = (u^{(i)})^T \Sigma_{L_1,L_1}^{-1/2} L_1$. We know from (**) that $h_i, h_j$ are orthogonal. Because computing CCA can result in the accrual of small numerical errors [321], we first explicitly orthonormalize $h_1, ..., h_c$ via Gram-Schmidt. We then identify how much of the original output is accounted for by each $h_i$:

$$\tilde{\alpha}_i = \sum_j \left| \langle h_i, z_j \rangle \right|$$

Normalizing this to get weights $\alpha_i$, with $\sum_i \alpha_i = 1$, we can compute the projection weighted CCA distance[3]:

$$d(L_1, L_2) = 1 - \sum_{i=1}^{c} \alpha_i \rho^{(i)}$$

As a simple test of the benefits of projection weighting, we constructed a toy case in which we used CCA to compare the representations of two networks with common (signal) and uncommon (noise) structure, each of a fixed dimensionality. We then used the naive mean and projected weighted mean to measure the CCA distance between these two networks as a function of the ratio of signal dimensions to noise dimensions. As expected we found that while the naive mean was extremely sensitive to this ratio, the projection weighted mean was largely robust (Figure 4.2).

[3]We note that this is technically a pseudo-distance rather than a distance as it is non-symmetric.

## 4.2 Using CCA to measure similarity of converged solutions

Because CCA measures the distance between two representations independent of linear transforms, it enables formerly difficult comparisons between the representations of different networks. Here, we use this property of CCA to evaluate whether groups of networks trained on CIFAR-10 with different random initializations converge to similar solutions under the following conditions:

- When trained on identically randomized labels (as in [370]) or on the true labels (Section 4.2.1)
- As network width is varied (Section 4.2.2)
- In a large sweep of 200 networks (Section 4.2.3)

### 4.2.1 Generalizing networks converge to more similar solutions than memorizing networks

It has recently been observed that DNNs are capable of solving image classification tasks even when the labels have been randomly permuted [370]. Such networks must, by definition, memorize the training data, and therefore cannot generalize beyond the training set. However, the representational properties which distinguish networks which memorize from those which generalize remain unclear.

In particular, we hypothesize that the representational similarity in a group of generalizing networks (networks trained on the true labels) should differ from

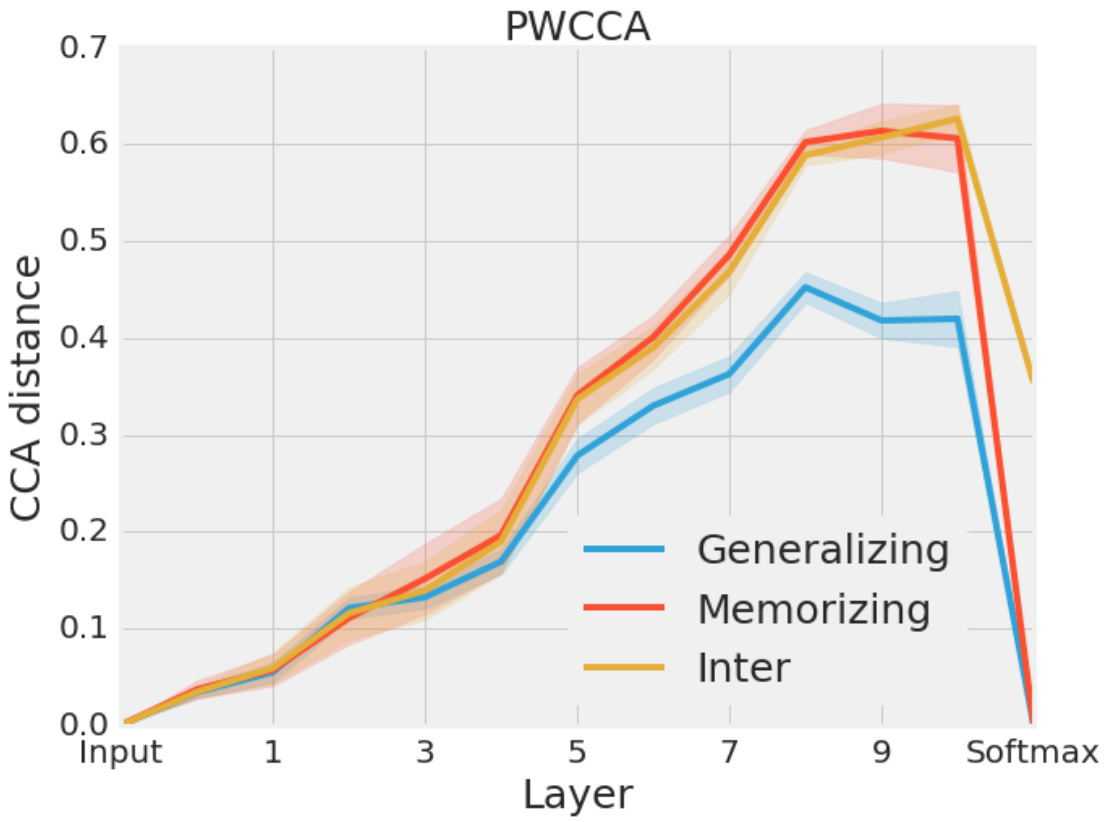



Figure 4.3: **Generalizing networks converge to more similar solutions than memorizing networks.** Groups of 5 networks were trained on CIFAR-10 with either true labels (generalizing) or a fixed random permutation of the labels (memorizing). The pairwise CCA distance was then compared within each group and between generalizing and memorizing networks (inter) for each layer, based on the training data, and the projection weighted CCA coefficient (with thresholding to remove low variance noise.) While both categories converged to similar solutions in early layers, likely reflecting convergent edge detectors, etc., generalizing networks converge to significantly more similar solutions in later layers. At the softmax, sets of both generalizing and memorizing networks converged to nearly identical solutions, as all networks achieved near-zero training loss. Error bars represent mean ± std weighted mean CCA distance across pairwise comparisons.

the representational similarity of memorizing networks (networks trained on random labels.)

To test this hypothesis, we trained groups of five networks with identical topology on either unmodified CIFAR-10 or CIFAR-10 with random labels (the same set of random labels was used for all networks), all of which were trained

to near-zero training loss[4]. Critically, the randomization of CIFAR-10 labels was consistent for all networks. To evaluate the similarity of converged solutions, we then measured the pairwise projection weighted CCA distance for each layer among networks trained on unmodified CIFAR-10 ("Generalizing"), among networks trained on randomized label CIFAR-10 ("Memorizing") and between each pair of networks trained on unmodified and random label CIFAR-10 ("Inter"). For all analyses, the representation in a given layer was obtained by averaging across all spatial locations within each filter.

Remarkably, we found that not only do generalizing networks converge to more similar solutions than memorizing networks (to be expected, since generalizing networks are more constrained), but memorizing networks are as similar to each other as they are to a generalizing network. This result suggests that the solutions found by memorizing networks were as diverse as those found across entirely different dataset labellings.

We also found that at early layers, all networks converged to equally similar solutions, regardless of whether they generalize or memorize (Figure 4.3). Intuitively, this makes sense as the feature detectors found in early layers of CNNs are likely required regardless of the dataset labelling. In contrast, however, at later layers, groups of generalizing networks converged to substantially more similar solutions than groups of memorizing networks (Figure 4.3). Even among networks which generalize, the CCA distance between solutions found in later layers was well above zero, suggesting that the solutions found were quite diverse. At the softmax layer, sets of both generalizing and memorizing networks converged to highly similar solutions when CCA distance was computed based

[4]Details of the architectures and training procedures for this and following experiments can be found in Appendix B.0.4.

on training data; when test data was used, however, only generalizing networks converged to similar softmax outputs (Figure App.10), again reflecting that each memorizing network memorizes the training data using a different strategy.

Importantly, because each network learned a different linear transform of a similar solution, traditional distance metrics, such as cosine or Euclidean distance, were insufficient to reveal this difference (Figure App.5). Additionally, while unweighted CCA revealed the same broad pattern, it does not reveal that generalizing networks get more similar in the final two layers (Figure App.9).

### 4.2.2 Wider networks converge to more similar solutions

In the model compression literature, it has been repeatedly noted that while networks are robust to the removal of a large fraction of their parameters (in some cases, as many as 90%), networks initialized and trained from the start with fewer parameters converge to poorer solutions than those derived from pruning a large networks [108, 109, 75, 12, 219, 182]. Recently, [83] proposed the "lottery ticket hypothesis," which hypothesizes that larger networks are more likely to converge to good solutions because they are more likely to contain a sub-network with a "lucky" initialization. If this were true, we might expect that groups of larger networks are more likely to contain the same "lottery ticket" sub-network and are therefore more likely to converge to similar solutions than smaller networks.

To test this intuition, we trained groups of convolutional networks with increasing numbers of filters at each layer. We then used projection weighted CCA

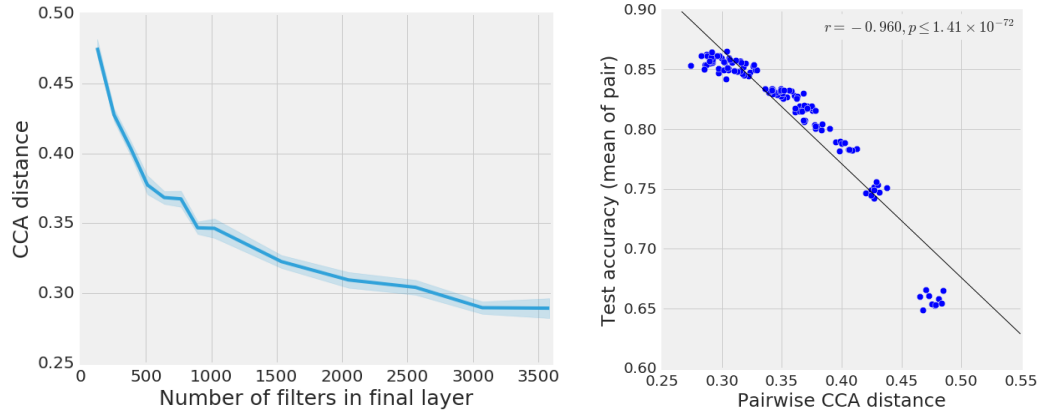



Figure 4.4: **Larger networks converge to more similar solutions.** Groups of 5 networks with different random initializations were trained on CIFAR-10. Pairwise CCA distance was computed for members of each group. Groups of larger networks converged to more similar solutions than groups of smaller networks (**a**). Test accuracy was highly correlated with degree of convergent similarity, as measured by CCA distance (**b**).

to measure the pairwise similarity between each group of networks of the same size. Consistent with our intuition, we found that larger networks converged to much more similar solutions than smaller networks (Figure 4.4).[5] This is also consistent with the equivalence of deep networks to Gaussian processes (GPs) in the limit of infinite width [173, 208]. If each unit in a layer corresponds to a draw from a GP, then as the number of units increases the CCA distance will go to zero.

Interestingly, we also found that networks which converged to more similar solutions also achieved noticeably higher test accuracy. In fact, we found that across pairs of networks, the correlation between test accuracy and the pairwise CCA distance was -0.96 (Figure 4.4), suggesting that the CCA distance between groups of identical networks with different random initializations (computed using the *train* data) may serve as *a strong predictor of test accuracy*. It may therefore

[5]To control for variability in CCA distance due to comparisons across representations of different sizes, a random subset of 128 filters from the final layer were used for all network comparisons. This bias should, if anything, lead to an overestimate of the distance between groups of larger networks, as they are more heavily subsampled.

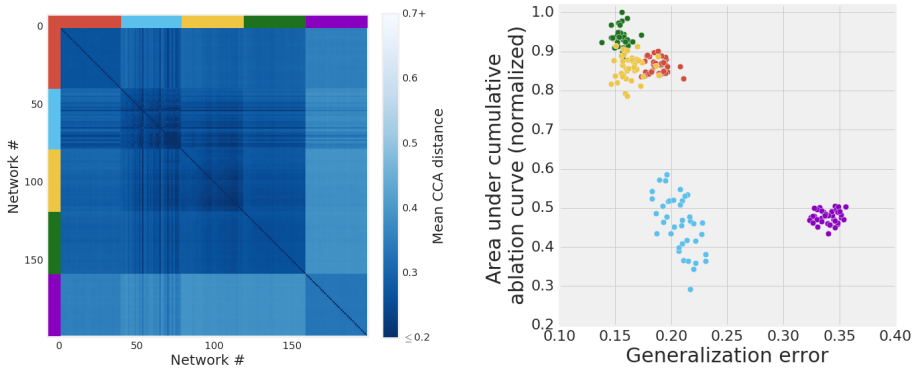



Figure 4.5: **CCA reveals clusters of converged solutions across networks with different random initializations and learning rates.** 200 networks with identical topology and varying learning rates were trained on CIFAR-10. CCA distance between the eighth layer of each pair of networks was computed, revealing five distinct subgroups of networks (**a**). These five subgroups align almost perfectly with the subgroups discovered in [221] (**b**; colors correspond to bars in **a**), despite the fact that the clusters in [221] were generated using robustness to cumulative ablation, an entirely separate metric.

enable accurate prediction of test performance without requiring the use of a validation set.

### 4.2.3 Across many initializations and learning rates, networks converge to discriminable clusters of solutions

Here, we ask whether networks trained on the same data with different initializations and learning rates converge to the same solutions. To test this, we measured the pairwise CCA distance between networks trained on unmodified CIFAR-10. Interestingly, when we plotted the pairwise distance matrix (Figure 4.5), we observed a block diagonal structure consistent with five clusters of converged network solutions, with one cluster highly dissimilar to the other four clusters. Despite the fact that these networks all achieved similar train loss (and many reached similar test accuracy as well), these clusters corresponded with

the learning rate used to train each network. This result suggests that there exist multiple minima in the optimization landscape to which networks may converge which are largely specified by the optimization parameters.

In [221], the authors also observed clusters of network solutions using the relationship between networks' robustness to cumulative deletion or "ablation" of filters and generalization error. To test whether the same clusters are found via these distinct approaches, we assigned a color to each cluster found using CCA (see bars on left and top in Figure 4.5), and used these colors to identify the same networks in a plot of ablation robustness vs. generalization error (Figure 4.5). Surprisingly, the clusters found using CCA aligned nearly perfectly with those observed using ablation robustness.

This result suggests not only that networks with different learning rates converge to distinct clusters of solutions, but also that these clusters can be uncovered independently using multiple methods, each of which measures a different property of the learned solution. Moreover, analyzing these networks using traditional metrics, such as generalization error, would obscure the differences between many of these networks.

## 4.3 CCA on Recurrent Neural Networks

So far, CCA has been used to study *feedforward* networks. We now use CCA to investigate RNNs. Our RNNs are LSTMs used for the Penn Treebank (PTB) and WikiText-2 (WT2) language modelling tasks, following the implementation in [210, 211].

One specific question we explore is whether the learning dynamics of RNNs mirror the "bottom up" convergence observed in the feedforward case in [253], as well as investigating whether CCA produces qualitatively better outputs than other metrics. However, in the case of RNNs, there are two possible notions of "time". There is the training timestep, which affects the values of the weights, but also a 'sequence timestep' – the number of tokens of the sequence that have been fed into the recurrent net. This latter notion of time does not explicitly change the weights, but results in updated values of the cell state and hidden state of the network, which of course affect the representations of the network.

In this work, we primarily focus on the training notion of time; however, we perform a preliminary investigation of the sequence notion of time as well, demonstrating that CCA is capable of finding similarity across sequence timesteps which are missed by traditional metrics (Figures App.2, App.4), but also that even CCA often fails to find similarity in the hidden state across sequence timesteps, suggesting that representations over sequence timesteps are often not linearly similar (Figure App.3).

### 4.3.1 Learning Dynamics Through Training Time

To measure the convergence of representations through training time, we computed the projection weighted mean CCA value for each layer's representation throughout training to its final representation. We observed bottom-up convergence in both Penn Treebank and WikiText-2 (Figure 4.6a-b). We repeated these experiments with cosine and Euclidean distance (Figure App.8), finding that while these other metrics also reveal a bottom up convergence, the results with

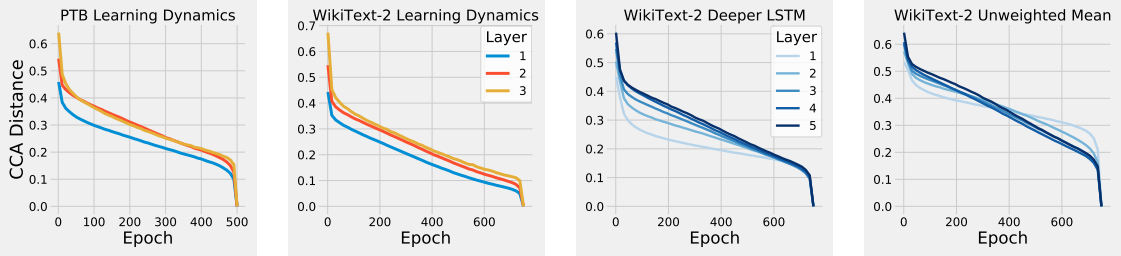



Figure 4.6: **RNNs exhibit bottom-up learning dynamics.** To test whether layers converge to their final representation over the course of training with a particular structure, we compared each layer's representation over the course of training to its final representation using CCA. In shallow RNNs trained on PTB (**a**), and WikiText-2 (**b**), we observed a clear bottom-up convergence pattern, in which early layers converge to their final representation before later layers. In deeper RNNs trained on WikiText-2, we observed a similar pattern (**c**). Importantly, the weighted mean reveals this effect much more accurately than the unweighted mean, which is also supported by control experiments (Figure App.8) (**d**), revealing the importance of appropriate weighting of CCA coefficients.

CCA highlight this phenomena much more clearly.

We also observed bottom-up convergence in a deeper LSTM trained on WikiText-2 (the larger dataset) (Figure 4.6). Interestingly, we found that this result changes noticeably if we use the unweighted mean CCA instead, demonstrating the importance of the weighting scheme (Figure 4.6).

## 4.4 Chapter Discussion and Future Directions

In this study, we developed CCA as a tool to gain insights on many representational properties of deep neural networks. We found that the representations in hidden layers of a neural network contain both "signal" components, which are stable over training and correspond to performance curves, and an unstable "noise" component. Using this insight, we proposed projection weighted CCA,

adapting [253]. Leveraging the ability of CCA to compare across different networks, we investigated the properties of converged solutions of convolutional neural networks (Section 4.2), finding that networks which generalize converge to more similar solutions than those which memorize (Section 4.2.1), that wider networks converge to more similar solutions than narrow networks (Section 4.2.2), and that across otherwise identical networks with different random initializations and learning rates, networks converge to diverse clusters of solutions (Section 4.2.3). We also used projection weighted CCA to study the dynamics (both across training time and sequence steps) of RNNs, (Section 4.3), finding that RNNs exhibit bottom-up convergence over the course of training (Section 4.3.1), and that across sequence timesteps, RNN representations vary nonlinearly (Section B.0.3).

One interesting direction for future work is to examine what is unique about directions which are preserved across networks trained with different initialization. Previous work has demonstrated that these directions are sufficient for the network computation [253], but the properties that make these directions special remain unknown. Furthermore, the attributes which specifically distinguish the diverse solutions found in Figure 4.5 remain unclear. We also observed that networks which converge to similar solutions exhibit higher generalization performance (Figure 4.4). In future work, it would be interesting to explore whether this insight could be used as a regularizer to improve network performance. Additionally, it would be useful to explore whether this result is consistent in RNNs as well as CNNs. Another interesting direction would be to investigate which aspects of the representation present in RNNs is stable over time and which aspects vary. Additionally, in previous work [253], it was observed that fixing layers in CNNs over the course of training led to better test performance

("freeze training"). An interesting open question would be to investigate whether a similar training protocol could be adapted for RNNs.

# Part III

# Informing Algorithms for Efficient Learning

CHAPTER 5

# RAPID LEARNING OR FEATURE REUSE? INVESTIGATING FEW-SHOT LEARNING VIA META-LEARNING

In the following two chapters, we study algorithms for training machine learning systems more *efficiently*. Specifically, a core challenge in the application of machine learning systems to (especially) specialized domains such as medicine is getting access to enough training data. We study two learning methods that can be used to mitigate these issues, few-shot learning and (in the next chapter) transfer learning.

In few-shot learning, new tasks must be learned with a very limited number of labelled datapoints. A significant body of work has looked at tackling this challenge using meta-learning approaches [158, 327, 295, 76, 277, 259, 225]. Broadly speaking, these approaches define a family of tasks, some of which are used for training and others solely for evaluation. A proposed meta-learning algorithm then looks at learning properties that generalize across the different training tasks, and result in fast and efficient learning of the evaluation tasks.

One highly successful meta-learning algorithm has been *Model Agnostic Meta-Learning (MAML)* [76]. At a high level, the MAML algorithm is comprised of two optimization loops. The outer loop (in the spirit of meta-learning) aims to find an effective *meta-initialization*, from which the inner loop can perform efficient *adaptation* – optimize parameters to solve new tasks with very few labelled examples. This algorithm, with deep neural networks as the underlying model, has been highly influential, with significant follow on work, such as first order variants [225], probabilistic extensions [79], augmentation with generative

modelling [272], and many others [133, 78, 97, 318].

Despite the popularity of MAML, and the numerous followups and extensions, there remains a fundamental open question on the basic algorithm. Does the meta-initialization learned by the outer loop result in *rapid learning* on unseen test tasks (efficient but significant changes in the representations) or is the success primarily due to *feature reuse* (with the meta-initialization already providing high quality representations)? In this chapter, we explore this question and its many surprising consequences. Our main contributions are:

- We perform layer freezing experiments and latent representational analysis of MAML, finding that feature reuse is the predominant reason for efficient learning.
- Based on these results, we propose the *ANIL (Almost No Inner Loop)* algorithm, a significant simplification to MAML that removes the inner loop updates for all but the head (final layer) of a neural network during training *and* inference. ANIL performs identically to MAML on standard benchmark few-shot classification and RL tasks and offers computational benefits over MAML.
- We study the effect of the head of the network, finding that once training is complete, the head can be removed, and the representations can be used without adaptation to perform unseen tasks, which we call the *No Inner Loop (NIL)* algorithm.
- We study different training regimes, e.g. multiclass classification, multitask learning, etc, and find that the task specificity of MAML/ANIL at training facilitate the learning of better features. We also find that multitask training,

a popular baseline with no task specificity, performs worse than random features.

- We discuss rapid learning and feature reuse in the context of other meta-learning approaches.

## 5.1 Related Work

MAML [76] is a highly popular meta-learning algorithm for few-shot learning, achieving competitive performance on several benchmark few-shot learning problems [158, 327, 295, 277, 259, 225]. It is part of the family of optimization-based meta-learning algorithms, with other members of this family presenting variations around how to learn the weights of the task-specific classifier. For example [176, 94, 29, 175, 377] first learn functions to embed the support set and target examples of a few-shot learning task, before using the test support set to learn task specific weights to use on the embedded target examples. [114] also proceeds similarly, using a Bayesian approach. The method of [21] explores a related approach, focusing on applications in text classification.

Of these optimization-based meta-learning algorithms, MAML has been especially influential, inspiring numerous direct extensions in recent literature [11, 79, 97, 272]. Most of these extensions critically rely on the core structure of the MAML algorithm, incorporating an outer loop (for meta-training), and an inner loop (for task-specific adaptation), and there is little prior work analyzing why this central part of the MAML algorithm is practically successful. In this work, we focus on this foundational question, examining how and why MAML

leads to effective few-shot learning. To do this, we utilize analytical tools such as Canonical Correlation Analysis (CCA) [253, 220] and Centered Kernel Alignment (CKA) [164] to study the neural network representations learned with the MAML algorithm, which also demonstrates MAML's ability to learn effective features for few-shot learning.

Insights from this analysis lead to a simplified algorithm, ANIL, which almost completely removes the inner optimization loop with no reduction in performance. Prior works [381, 140] have proposed algorithms where some parameters are only updated in the outer loop and others only in the inner loop. However, these works are motivated by different questions, such as improving MAML's performance or learning better representations, rather than analysing rapid learning vs feature reuse in MAML.

## 5.2 MAML, Rapid Learning, and Feature Reuse

Our goal is to understand whether the MAML algorithm efficiently solves new tasks due to *rapid learning* or *feature reuse*. In rapid learning, large representational and parameter changes occur during adaptation to each new task as a result of favorable weight conditioning from the meta-initialization. In feature reuse, the meta-initialization already contains highly useful features that can mostly be reused as is for new tasks, so little task-specific adaptation occurs. Figure 5.1 shows a schematic of these two hypotheses.

We start off by overviewing the details of the MAML algorithm, and then we study the rapid learning vs feature reuse question via layer freezing experiments

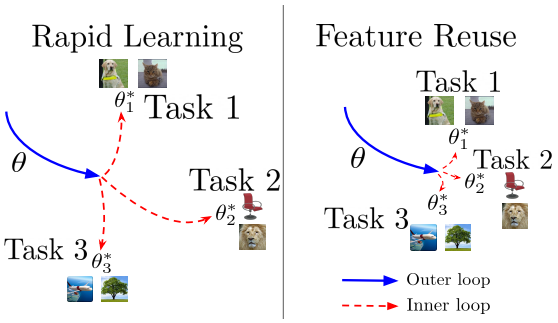



Figure 5.1: **Rapid learning and feature reuse paradigms.** In Rapid Learning, outer loop training leads to a parameter setting that is well-conditioned for fast learning, and inner loop updates result in significant task specialization. In Feature Reuse, the outer loop leads to parameter values corresponding to reusable features, from which the parameters do not move significantly in the inner loop.

and analyzing latent representations of models trained with MAML. The results strongly support feature reuse as the predominant factor behind MAML's success. In Section 5.3, we explore the consequences of this, providing a significant simplification of MAML, the ANIL algorithm, and in Section 5.5, we outline the connections to meta-learning more broadly.

### 5.2.1 Overview of MAML

The MAML algorithm finds an *initialization* for a neural network so that new tasks can be learnt with very few examples ($k$ examples from each class for $k$-shot learning) via two optimization loops:

- **Outer Loop:** Updates the initialization of the neural network parameters (often called the *meta-initialization*) to a setting that enables fast adaptation to new tasks.
- **Inner Loop:** Performs *adaptation*: takes the outer loop initialization, and,

*separately for each task*, performs a few gradient updates over the $k$ labelled examples (the *support set*) provided for adaptation.

More formally, we first define our base model to be a neural network with meta-initialization parameters $\theta$; let this be represented by $f_\theta$. We have have a distribution $\mathcal{D}$ over tasks, and draw a batch $\{T_1, ..., T_B\}$ of $B$ tasks from $\mathcal{D}$. For each task $T_b$, we have a *support set* of examples $\mathcal{S}_{T_b}$, which are used for inner loop updates, and a *target set* of examples $\mathcal{Z}_{T_b}$, which are used for outer loop updates. Let $\theta_i^{(b)}$ signify $\theta$ after $i$ gradient updates for task $T_b$, and let $\theta_0^{(b)} = \theta$. In the inner loop, during each update, we compute

$$\theta_m^{(b)} = \theta_{m-1}^{(b)} - \alpha \nabla_{\theta_{m-1}^{(b)}} \mathcal{L}_{S_{T_b}}(f_{\theta_{m-1}^{(b)}(\theta)}) \tag{1}$$

for $m$ fixed across all tasks, where $\mathcal{L}_{S_{T_b}}(f_{\theta_{m-1}^{(b)}(\theta)})$ is the loss on the support set of $T_b$ after $m-1$ inner loop updates.

We then define the meta-loss as

$$\mathcal{L}_{meta}(\theta) = \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Z}_{T_b}}(f_{\theta_m^{(b)}(\theta)})$$

where $\mathcal{L}_{\mathcal{Z}_{T_b}}(f_{\theta_m^{(b)}(\theta)})$ is the loss on the target set of $T_b$ after $m$ inner loop updates, making clear the dependence of $f_{\theta_m^{(b)}}$ on $\theta$. The outer optimization loop then updates $\theta$ as

$$\theta = \theta - \eta \nabla_\theta \mathcal{L}_{meta}(\theta)$$

At test time, we draw unseen tasks $\{T_1^{(test)}, ..., T_n^{(test)}\}$ from the task distribution, and evaluate the loss and accuracy on $\mathcal{Z}_{T_i^{(test)}}$ after inner loop adaptation using $\mathcal{S}_{T_i^{(test)}}$ (e.g. loss is $\mathcal{L}_{\mathcal{Z}_{T_i^{(test)}}}\left(f_{\theta_m^{(i)}(\theta)}\right)$).

### 5.2.2 Rapid Learning or Feature Reuse?

We now turn our attention to the key question: *Is MAML's efficacy predominantly due to rapid learning or feature reuse?* In investigating this question, there is an important distinction between the *head* (final layer) of the network and the earlier layers (the *body* of the network). In each few-shot learning task, there is a different alignment between the output neurons and classes. For instance, in task $\mathcal{T}_1$, the (wlog) five output neurons might correspond, in order, to the classes (dog, cat, frog, cupcake, phone), while for a different task, $\mathcal{T}_2$, they might correspond, in order, to (airplane, frog, boat, car, pumpkin). This means that the head must necessarily change for each task to learn the new alignment, and for the rapid learning vs feature reuse question, we are primarily interested in the behavior of the body of the network. We return to this in more detail in Section 5.4, where we present an algorithm (NIL) that does not use a head at test time.

To study rapid learning vs feature reuse in the network body, we perform two sets of experiments: (1) We evaluate few-shot learning performance when freezing parameters after MAML training, without test time inner loop adaptation; (2) We use representational similarity tools to directly analyze how much the network features and representations change through the inner loop. We use the MiniImageNet dataset, a popular standard benchmark for few-shot learning, and with the standard convolutional architecture in [76]. Results are averaged over three random seeds. Full implementation details are in Appendix C.2.

| Freeze layers | MiniImageNet-5way-1shot | MiniImageNet-5way-5shot |
|---|---|---|
| None | 46.9 ± 0.2 | 63.1 ± 0.4 |
| 1 | 46.5 ± 0.3 | 63.0 ± 0.6 |
| 1,2 | 46.4 ± 0.4 | 62.6 ± 0.6 |
| 1,2,3 | 46.3 ± 0.4 | 61.2 ± 0.5 |
| 1,2,3,4 | 46.3 ± 0.4 | 61.0 ± 0.6 |

Table 5.1: **Freezing successive layers (preventing inner loop adaptation) does not affect accuracy, supporting feature reuse.** To test the amount of feature reuse happening in the inner loop adaptation, we test the accuracy of the model when we freeze (prevent inner loop adaptation) a contiguous block of layers at test time. We find that freezing even all four convolutional layers of the network (all layers except the network head) hardly affects accuracy. This strongly supports the feature reuse hypothesis: layers don't have to change rapidly at adaptation time; they already contain good features from the meta-initialization.

### Freezing Layer Representations

To study the impact of the inner loop adaptation, we freeze a contiguous subset of layers of the network, during the inner loop at test time (after using the standard MAML algorithm, incorporating both optimization loops, for training). In particular, the frozen layers are not updated at all to the test time task, and must reuse the features learned by the meta-initialization that the outer loop converges to. We compare the few-shot learning accuracy when freezing to the accuracy when allowing inner loop adaptation.

Results are shown in Table 5.1. We observe that even when freezing all layers in the network body, performance hardly changes. This suggests that the meta-initialization has already learned good enough features that can be reused as is, without needing to perform any rapid learning for each test time task.

**Representational Similarity Experiments**

We next study how much the latent representations (the latent functions) learned by the neural network change during the inner loop adaptation phase. Following several recent works [253, 278, 220, 203, 254, 95, 25] we measure this by applying Canonical Correlation Analysis (CCA) to the latent representations of the network. CCA provides a way to the compare representations of two (latent) layers $L_1, L_2$ of a neural network, outputting a similarity score between 0 (not similar at all) and 1 (identical). For full details, see [253, 220]. In our analysis, we take $L_1$ to be a layer before the inner loop adaptation steps, and $L_2$ after the inner loop adaptation steps. We compute CCA similarity between $L_1$, $L_2$, averaging the similarity score across different random seeds of the model and different test time tasks. Full details are in Appendix C.2.2

The result is shown in Figure 5.2, left pane. Representations in the body of the network (the convolutional layers) are highly similar, with CCA similarity scores of $> 0.9$, indicating that the inner loop induces little to no functional change. By contrast, the head of the network, which does change significantly in the inner loop, has a CCA similarity of less than 0.5. To further validate this, we also compute CKA (Centered Kernel Alignment) [164] (Figure 5.2 right), another similarity metric for neural network representations, which illustrates the same pattern. These representational analysis results strongly support the feature reuse hypothesis, with further results in the Appendix, Sections C.2.3 and C.2.4 providing yet more evidence.

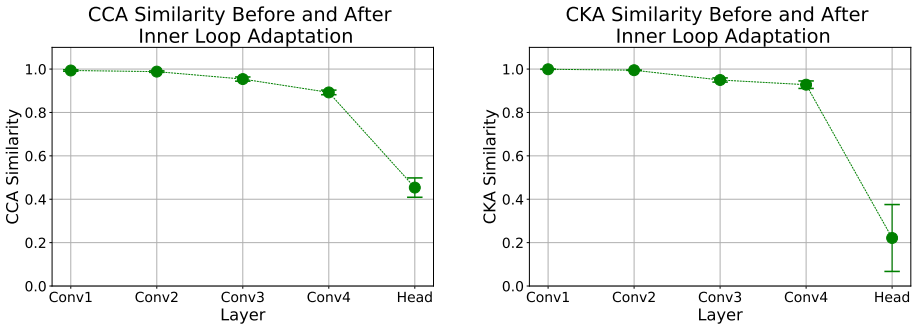



Figure 5.2: **High CCA/CKA similarity between representations before and after adaptation for all layers except the head.** We compute CCA/CKA similarity between the representation of a layer before the inner loop adaptation and after adaptation. We observe that for all layers except the head, the CCA/CKA similarity is almost 1, indicating perfect similarity. This suggests that these layers do not change much during adaptation, but mostly perform feature reuse. Note that there is a slight dip in similarity in the higher conv layers (e.g. conv3, conv4); this is likely because the slight representational differences in conv1, conv2 have a compounding effect on the representations of conv3, conv4. The head of the network must change significantly during adaptation, and this is reflected in the much lower CCA/CKA similarity.

### Feature Reuse Happens Early in Learning

Having observed that the inner loop does not significantly affect the learned representations with a fully trained model, we extend our analysis to see whether the inner loop affects representations and features earlier on in training. We take MAML models at 10000, 20000, and 30000 iterations into training, perform freezing experiments (as in Section 5.2.2) and representational similarity experiments (as in Section 5.2.2).

Results in Figure 5.3 show the same patterns from early in training, with CCA similarity between activations pre and post inner loop update on MiniImageNet-5way-5shot being very high for the body (just like Figure 5.2), and similar to Table 5.1, test accuracy remaining approximately the same when freezing contiguous subsets of layers, even when freezing all layers of the network body. This shows

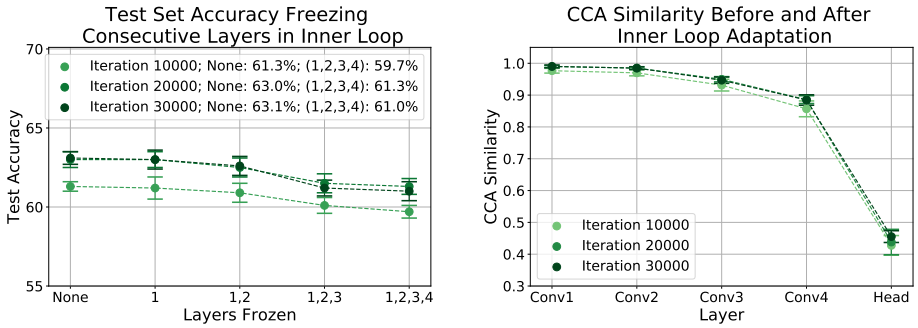



Figure 5.3: **Inner loop updates have little effect on learned representations from early on in learning.** Left pane: we freeze contiguous blocks of layers (no adaptation at test time), on MiniImageNet-5way-5shot and see almost identical performance. Right pane: representations of all layers except the head are highly similar pre/post adaptation – i.e. features are being reused. This is true from early (iteration 10000) in training.

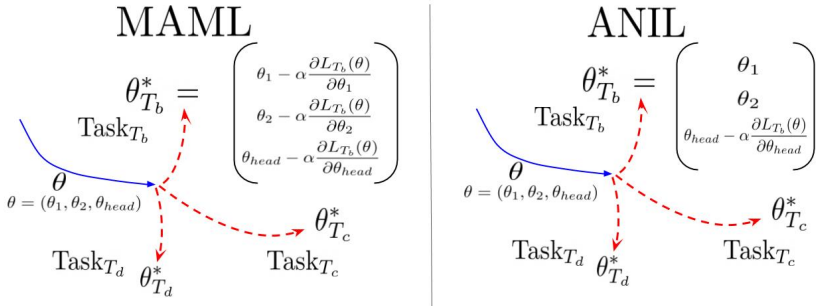



Figure 5.4: **Schematic of MAML and ANIL algorithms.** The difference between the MAML and ANIL algorithms: in MAML (left), the inner loop (task-specific) gradient updates are applied to all parameters $\theta$, which are initialized with the meta-initialization from the outer loop. In ANIL (right), only the parameters corresponding to the network head $\theta_{head}$ are updated by the inner loop, during training **and** testing.

that even early on in training, significant feature reuse is taking place, with the inner loop having minimal effect on learned representations and features. Results for 1shot MiniImageNet are in Appendix C.2.5, and show very similar trends.

## 5.3 The ANIL (Almost No Inner Loop) Algorithm

In the previous section we saw that for all layers except the head of the neural network, the meta-initialization learned by the outer loop of MAML results in

| Method | Omniglot-20way-1shot | Omniglot-20way-5shot |
|---|---|---|
| MAML | 93.7 ± 0.7 | 96.4 ± 0.1 |
| ANIL | 96.2 ± 0.5 | 98.0 ± 0.3 |

| Method | MiniImageNet-5way-1shot | MiniImageNet-5way-5shot |
|---|---|---|
| MAML | 46.9 ± 0.2 | 63.1 ± 0.4 |
| ANIL | 46.7 ± 0.4 | 61.5 ± 0.5 |

| Method | HalfCheetah-Direction | HalfCheetah-Velocity | 2D-Navigation |
|---|---|---|---|
| MAML | 170.4 ± 21.0 | -139.0 ± 18.9 | -20.3 ± 3.2 |
| ANIL | 363.2 ± 14.8 | -120.9 ± 6.3 | -20.1 ± 2.3 |

Table 5.2: **ANIL matches the performance of MAML on few-shot image classification and RL.** On benchmark few-shot classification tasks MAML and ANIL have comparable accuracy, and also comparable average return (the higher the better) on standard RL tasks [76].

very good features that can be reused as is on new tasks. Inner loop adaptation does not significantly change the representations of these layers, even from early on in training. This suggests a natural simplification of the MAML algorithm: the *ANIL (Almost No Inner Loop) algorithm*.

In ANIL, during training **and** testing, we *remove* the inner loop updates for the network body, and apply inner loop adaptation *only to the head*. The head requires the inner loop to allow it to align to the different classes in each task. In Section 5.4.1 we consider another variant, the *NIL (No Inner Loop) algorithm*, that removes the head entirely at test time, and uses learned features and cosine similarity to perform effective classification, thus avoiding inner loop updates altogether.

For the ANIL algorithm, mathematically, let $\theta = (\theta_1, ..., \theta_l)$ be the (meta-initialization) parameters for the $l$ layers of the network. Following the notation

of Section 5.2.1, let $\theta_m^{(b)}$ be the parameters after $m$ inner gradient updates for task $\mathcal{T}_b$. In ANIL, we have that:

$$\theta_m^{(b)} = \left(\theta_1, \ldots, (\theta_l)_{m-1}^{(b)} - \alpha \nabla_{(\theta_l)_{m-1}^{(b)}} \mathcal{L}_{S_b}(f_{\theta_{m-1}^{(b)}})\right)$$

i.e. only the final layer gets the inner loop updates. As before, we then define the meta-loss, and compute the outer loop gradient update. The intuition for ANIL arises from Figure 5.3, where we observe that inner loop updates have little effect on the network body even early in training, suggesting the possibility of removing them entirely. Note that this is distinct to the freezing experiments, where we only removed the inner loop at inference time. Figure 5.4 presents the difference between MAML and ANIL, and Appendix C.3.1 considers a simple example of the gradient update in ANIL, showing how the ANIL update differs from MAML.

**Computational benefit of ANIL:** As ANIL almost has no inner loop, it significantly speeds up both training and inference. We found an average speedup of 1.7x per training iteration over MAML and an average speedup of 4.1x per inference iteration. In Appendix C.3.5 we provide the full results.

**Results of ANIL on Standard Benchmarks:** We evaluate ANIL on few-shot image classification and RL benchmarks, using the same model architectures as the original MAML authors, for both supervised learning and RL. Further implementation details are in Appendix C.3.4. The results in Table 5.2 (mean and standard deviation of performance over three random initializations) show that ANIL matches the performance of MAML on both few-shot classification (accuracy) and RL (average return, the higher the better), demonstrating that the

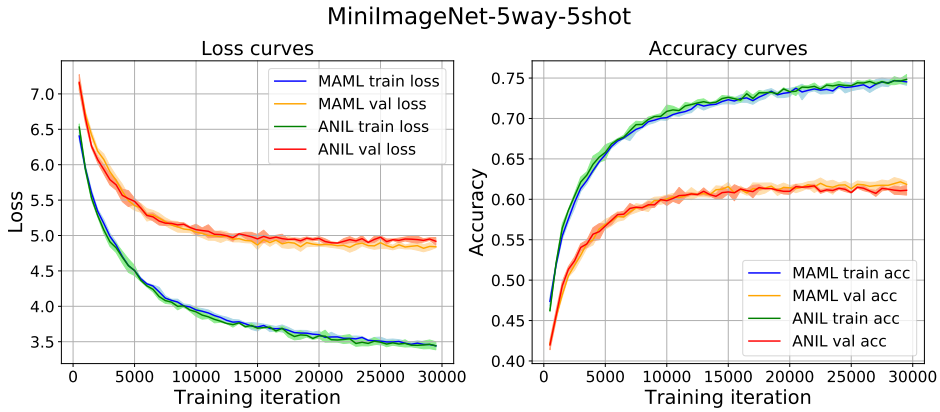



Figure 5.5: **MAML and ANIL learn very similarly**. Loss and accuracy curves for MAML and ANIL on MiniImageNet-5way-5shot, illustrating how MAML and ANIL behave similarly through the training process.

inner loop adaptation of the body is unnecessary for learning good features.

**MAML and ANIL Models Show Similar Behavior:** MAML and ANIL perform equally well on few-shot learning benchmarks, illustrating that removing the inner loop during training does not hinder performance. To study the behavior of MAML and ANIL models further, we plot learning curves for both algorithms on MiniImageNet-5way-5shot, Figure 5.5. We see that loss and accuracy for both algorithms look very similar throughout training. We also look at CCA and CKA scores of the representations learned by both algorithms, Table 5.3. We observe that MAML-ANIL representations have the same average similarity scores as MAML-MAML and ANIL-ANIL representations, suggesting both algorithms learn comparable features (removing the inner loop doesn't change the kinds of features learned.) Further learning curves and representational similarity results are presented in Appendices C.3.2 and C.3.3.

| Model Pair | CCA Similarity | CKA Similarity |
|---|---|---|
| MAML-MAML | 0.51 | 0.83 |
| ANIL-ANIL | 0.51 | 0.86 |
| ANIL-MAML | 0.50 | 0.83 |

Table 5.3: **MAML and ANIL models learn comparable representations.** Comparing CCA/CKA similarity scores of the of MAML-ANIL representations (averaged over network body), and MAML-MAML and ANIL-ANIL similarity scores (across different random seeds) shows algorithmic differences between MAML/ANIL does not result in vastly different types of features learned.

## 5.4 Contributions of the Network Head and Body

So far, we have seen that MAML predominantly relies on feature reuse, with the network body (all layers except the last layer) already containing good features at meta-initialization. We also observe that such features can be learned even without inner loop adaptation during training (ANIL algorithm). The head, however, requires inner loop adaptation to enable task specificity.

In this section, we explore the contributions of the network head and body. We first ask: *How important is the head at test time, when good features have already been learned?* Motivating this question is that the features in the body of the network needed no adaptation at inference time, so perhaps they are themselves sufficient to perform classification, with no head. In Section 5.4.1, we find that test time performance is entirely determined by the quality of these representations, and we can use similarity of the frozen meta-initialization representations to perform unseen tasks, *removing the head entirely*. We call this the NIL (No Inner Loop) algorithm.

Given this result, we next study how useful the head is at training (in ensuring the network body learns good features). We look at multiple different

training regimes (some without the head) for the network body, and evaluate the quality of the representations. We find that MAML/ANIL result in the best representations, demonstrating the importance of the head during training for feature learning.

### 5.4.1 The Head at Test Time and the NIL (No Inner Loop) Algorithm

We study how important the head and task specific alignment are when good features have *already* been learned (through training) by the meta-initialization. At test time, we find that the representations can be used directly, with *no* adaptation, which leads to the *No Inner Loop (NIL) algorithm*:

1. Train a few-shot learning model with ANIL/MAML algorithm as standard. We use ANIL training.
2. At test time, remove the head of the trained model. For each task, first pass the $k$ labelled examples (support set) through the body of the network, to get their penultimate layer representations. Then, for a test example, compute cosine similarities between its penultimate layer representation and those of the support set, using these similarities to weight the support set labels, as in [327].

The results for the NIL algorithm, following ANIL training, on few-shot classification benchmarks are given in Table 5.4. Despite having no network head and no task specific adaptation, NIL performs comparably to MAML and

| Method | Omniglot-20way-1shot | Omniglot-20way-5shot |
|---|---|---|
| MAML | 93.7 ± 0.7 | 96.4 ± 0.1 |
| ANIL | 96.2 ± 0.5 | 98.0 ± 0.3 |
| NIL | 96.7 ± 0.3 | 98.0 ± 0.04 |
| Method | MiniImageNet-5way-1shot | MiniImageNet-5way-5shot |
| MAML | 46.9 ± 0.2 | 63.1 ± 0.4 |
| ANIL | 46.7 ± 0.4 | 61.5 ± 0.5 |
| NIL | 48.0 ± 0.7 | 62.2 ± 0.5 |

Table 5.4: **NIL algorithm performs as well as MAML and ANIL on few-shot image classification.** Performance of MAML, ANIL, and NIL on few-shot image classification benchmarks. We see that with no test-time inner loop, and just learned features, NIL performs comparably to MAML and ANIL, indicating the strength of the learned features, and the relative lack of importance of the head at test time.

ANIL. This demonstrates that the features learned by the network body when training with MAML/ANIL (and reused at test time) are the critical component in tackling these benchmarks.

### 5.4.2 Training Regimes for the Network Body

The NIL algorithm and results of Section 5.4.1 lead to the question of how important task alignment and the head are during training to ensure good features. Here, we study this question by examining the quality of features arising from different training regimes for the body. We look at (i) MAML and ANIL training; (ii) *multiclass classification*, where all of the training data and classes (from which training tasks are drawn) are used to perform standard classification; (iii) *multitask training*, a standard baseline, where no inner loop or task specific head is used, but the network is trained on all the tasks at the same time; (iv) *random features*, where the network is not trained at all, and features

are frozen after random initialization; (v) NIL at training time, where there is no head and cosine distance on the representations is used to get the label.

After training, we apply the NIL algorithm to evaluate test performance, and quality of features learned at training. The results are shown in Table 5.5. MAML and ANIL training performs best. Multitask training, which has no task specific head, performs the worst, even worse than random features (adding evidence for the need for task specificity at training to facilitate feature learning.) Using NIL during training performs worse than MAML/ANIL. These results suggest that the head is important at training to learn good features in the network body.

In Appendix C.4.1, we study test time performance variations from using a MAML/ANIL head instead of NIL, finding (as suggested by Section 5.4.1) very little performance difference. Additional results on similarity between the representations of different training regimes is given in Appendix C.4.2.

## 5.5 Feature Reuse in Other Meta-Learning Algorithms

Up till now, we have closely examined the MAML algorithm, and have demonstrated empirically that the algorithm's success is primarily due to feature reuse, rather than rapid learning. We now discuss rapid learning vs feature reuse more broadly in meta-learning. By combining our results with an analysis of evidence reported in prior work, we find support for many meta-learning algorithms succeeding via feature reuse, identifying a common theme characterizing the operating regime of much of current meta-learning.

| Method | MiniImageNet-5way-1shot |
|---|---|
| MAML training-NIL head | 48.4 ± 0.3 |
| ANIL training-NIL head | 48.0 ± 0.7 |
| Multiclass training-NIL head | 39.7 ± 0.3 |
| Multitask training-NIL head | 26.5 ± 1.1 |
| Random features-NIL head | 32.9 ± 0.6 |
| NIL training-NIL head | 38.3 ± 0.6 |
| Method | MiniImageNet-5way-5shot |
| MAML training-NIL head | 61.5 ± 0.8 |
| ANIL training-NIL head | 62.2 ± 0.5 |
| Multiclass training-NIL head | 54.4 ± 0.5 |
| Multitask training-NIL head | 34.2 ± 3.5 |
| Random features-NIL head | 43.2 ± 0.5 |
| NIL training-NIL head | 43.0 ± 0.2 |

Table 5.5: **MAML/ANIL training leads to superior features learned, supporting importance of head at training.** Training with MAML/ANIL leads to superior performance over other methods which do not have task specific heads, supporting the importance of the head at training.

### 5.5.1 Optimization and Model Based meta-learning

MAML falls within the broader class of *optimization based* meta-learning algorithms, which at inference time, directly optimize model parameters for a new task using the support set. MAML has inspired many other optimization-based algorithms, which utilize the same two-loop structure [176, 272, 79]. Our analysis so far has thus yielded insights into the feature reuse vs rapid learning question for this class of algorithms. Another broad class of meta-learning consists of *model based* algorithms, which also have notions of rapid learning and feature reuse.

In the model-based setting, the meta-learning model's parameters are *not*

directly optimized for the specific task on the support set. Instead, the model typically conditions its output on some representation of the task definition. One way to achieve this conditioning is to *jointly encode* the entire support set in the model's latent representation [327, 303], enabling it to adapt to the characteristics of each task. This constitutes rapid learning for model based meta-learning algorithms.

An alternative to joint encoding would be to encode each member of the support set independently, and apply a cosine similarity rule (as in [327]) to classify an unlabelled example. This mode of operation is purely feature reuse – we do not use information defining the task to directly influence the decision function.

If joint encoding gave significant test-time improvement over non-joint encoding, then this would suggest that rapid learning of the test-time task is taking place, as task specific information is being utilized to influence the model's decision function. However, on analyzing results in prior literature, this improvement appears to be minimal. Indeed, in e.g. Matching Networks [327], using joint encoding one reaches 44.2% accuracy on MiniImageNet-5way-1shot, whereas with independent encoding one obtains 41.2%: a small difference. More refined models suggest the gap is even smaller. For instance, in [43], many methods for one shot learning were re-implemented and studied, and baselines without joint encoding achieved 48.24% accuracy in MiniImageNet-5way-1shot, whilst other models using joint encoding such as Relation Net [303] achieve very similar accuracy of 49.31% (they also report MAML, at 46.47%). As a result, we believe that the dominant mode of "feature reuse" rather than "rapid learning" is what has currently dominated both MAML-styled optimization based meta-learning

*and* model based meta-learning.

## 5.6 Chapter Summary

In this chapter, we studied a fundamental question on whether the highly successful MAML algorithm relies on rapid learning or feature reuse. Through a series of experiments, we found that *feature reuse* is the dominant component in MAML's efficacy on benchmark datasets. This insight led to the ANIL (Almost No Inner Loop) algorithm, a simplification of MAML that has identical performance on standard image classification and reinforcement learning benchmarks, and provides computational benefits. We further study the importance of the head (final layer) of a neural network trained with MAML, discovering that the body (lower layers) of a network is sufficient for few-shot classification at test time, allowing us to remove the network head for testing (NIL algorithm) and still match performance. We connected our results to the broader literature in meta-learning, identifying feature reuse to be a common mode of operation for other meta-learning algorithms also. Based off of our conclusions, future work could look at developing and analyzing new meta-learning algorithms that perform more rapid learning, which may expand the datasets and problems amenable to these techniques. We note that our study mainly considered benchmark datasets, such as Omniglot and MiniImageNet. It is an interesting future direction to consider rapid learning and feature reuse in MAML on other few-shot learning datasets, such as those from [318].

CHAPTER 6

# UNDERSTANDING TRANSFER LEARNING WITH APPLICATIONS TO MEDICAL IMAGING

Following the results of Chapter 5 on few-shot learning as one method for (data) *efficient* training of machine learning systems, in this chapter we study *transfer learning*, a highly popular method for developing deep learning models, especially in medical applications.

Indeed for many applications of deep learning to medical imaging imaging, the present standard is to take an existing architecture designed for natural image datasets such as ImageNet, together with corresponding pretrained weights (e.g. ResNet [117], Inception [305]), and then fine-tune the model on the medical imaging data.

This basic formula has seen almost universal adoption across many different medical specialties. Two prominent lines of research have used this methodology for applications in radiology, training architectures like ResNet, DenseNet on chest x-rays [336, 255] and ophthalmology, training Inception-v3, ResNet on retinal fundus images [3, 102, 252, 57]. The research on ophthalmology has also culminated in FDA approval [316], and full clinical deployment [323]. Other applications include performing early detection of Alzheimer's Disease [62], identifying skin cancer from dermatologist level photographs [73], and even determining human embryo quality for IVF procedures [149].

Despite the immense popularity of transfer learning in medical imaging, there has been little work studying its precise effects, even as recent work on transfer

learning in the *natural image* setting [115, 165, 224, 136, 89] has challenged many commonly held beliefs. For example in [115], it is shown that transfer (even between similar tasks) does not necessarily result in performance improvements, while [165] illustrates that pretrained features may be less general than previously thought.

In the medical imaging setting, many such open questions remain. As described above, transfer learning is typically performed by taking a standard ImageNet architecture along with its pretrained weights, and then fine-tuning on the target task. However, ImageNet classification and medical image diagnosis have considerable differences.

First, many medical imaging tasks start with a large image of a bodily region of interest and use variations in local textures to identify pathologies. For example, in retinal fundus images, small red 'dots' are an indication of microaneurysms and diabetic retinopathy [1], and in chest x-rays local white opaque patches are signs of consolidation and pneumonia. This is in contrast to natural image datasets like ImageNet, where there is often a clear global subject of the image (Fig. 6.1). There is thus an open question of how much ImageNet feature reuse is helpful for medical images.

Additionally, most datasets have larger images (to facilitate the search for local variations), but with many fewer images than ImageNet, which has roughly one million images. By contrast medical datasets range from several thousand images [149] to a couple hundred thousand [102, 255].

Finally, medical tasks often have significantly fewer classes (5 classes for Diabetic Retinopathy diagnosis [102], 5 – 14 chest pathologies from x-rays [255])

than the standard ImageNet classification setup of 1000 classes. As standard ImageNet architectures have a large number of parameters concentrated at the higher layers for precisely this reason, the design of these models is likely to be suboptimal for the medical setting.

In this chapter, we present results from performing a fine-grained study on transfer learning for medical images. Our main contributions are:

**[1]** We evaluate the performance of standard architectures for natural images such as ImageNet, as well as a family of non-standard but smaller and simpler models, on two large scale medical imaging tasks, for which transfer learning is currently the norm. We find that (i) in all of these cases, transfer does not significantly help performance (ii) smaller, simpler convolutional architectures perform comparably to standard ImageNet models (iii) ImageNet performance is not predictive of medical performance. These conclusions also hold in the very small data regime.

**[2]** Given the comparable performance, we investigate whether using pre-trained weights leads to different learned representations, by using (SV)CCA [253] to directly analyze the hidden representations. We find that pretraining does affect the hidden representations, but there is a confounding issue of model size, where the large, standard ImageNet models do not change significantly through the fine-tuning process, as evidenced through surprising correlations between representational similarity at initialization and after convergence.

**[3]** Using further analysis and weight transfusion experiments, where we partially reuse pretrained weights, we isolate locations where meaningful feature reuse does occur, and explore hybrid approaches to transfer learning where

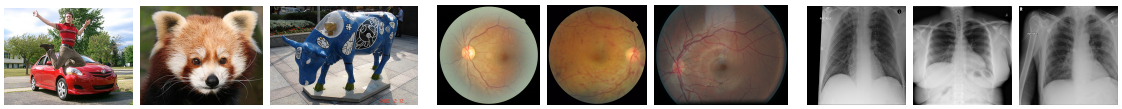


Figure 6.1: Example images from the *ImageNet*, the *retinal fundus photographs*, and the CheXpert datasets, respectively. The fundus photographs and chest x-rays have much higher resolution than the ImageNet images, and are classified by looking for small local variations in tissue.

a subset of pretrained weights are used, and other parts of the network are redesigned and made more lightweight.

**[4]** We show there are also *feature-independent* benefits to pretraining — reusing only the *scaling* of the pretrained weights but not the features can itself lead to large gains in convergence speed.

## 6.1 Datasets

Our primary dataset, the Retina data, consists of retinal *fundus photographs* [102], large $587 \times 587$ images of the back of the eye. These images are used to diagnose a variety of eye diseases including Diabetic Retinopathy (DR) [6]. DR is graded on a five-class scale of increasing severity [1]. Grades 3 and up are *referable DR* (requiring immediate specialist attention), while grades 1 and 2 correspond to *non-referable DR*. As in prior work [102, 3] we evaluate via AUC-ROC on identifying referable DR.

We also study a second medical imaging dataset, CheXpert [138], which consists of chest x-ray images (resized to 224×224), which can be used to diagnose 5 different thoracic pathologies: atelectasis, cardiomegaly, consolidation, edema and pleural effusion. We evaluate our models on the AUC of diagnosing each of

these pathologies. Figure 6.1 shows some example images from both datasets and ImageNet, demonstrating drastic differences in visual features among those datasets.

## 6.2 Models and Performance Evaluation of Transfer Learning

To lay the groundwork for our study, we select multiple neural network architectures and evaluate their performance when (1) training from random initialization and (2) doing transfer learning from ImageNet. We train both standard, high performing ImageNet architectures that have been popular for transfer learning, as well as a family of significantly smaller convolutional neural networks, which achieve comparable performance on the medical tasks.

As far as we are aware, there has been little work studying the effects of transfer learning from ImageNet on smaller, non-standard ImageNet architectures. (For example, [236] studies a different model, but does not evaluate the effect of transfer learning.) This line of investigation is especially important in the medical setting, where large, computationally expensive models might significantly impede mobile and on-device applications. Furthermore, in standard ImageNet models, most of the parameters are concentrated at the top, to perform the 1000-class classification. However, medical diagnosis often has considerably fewer classes – both the retinal fundus images and chest x-rays have just 5 classes – likely meaning that ImageNet models are highly overparametrized.

We find that across both datasets and all models, transfer learning does not significantly affect performance. Additionally, the family of smaller lightweight

convolutional networks performs comparably to standard ImageNet models, despite having significantly worse accuracy on ImageNet – the ImageNet task is not necessarily a good indication of success on medical datasets. Finally, we observe that these conclusions also hold in the setting of very limited data.

### 6.2.1 Description of Models

For the standard ImageNet architectures, we evaluate ResNet50 [115] and Inception-v3 [305], which have both been used extensively in medical transfer learning applications [3, 102, 336]. We also design a family of simple, smaller convolutional architectures. The basic building block for this family is the popular sequence of a (2d) convolution, followed by batch normalization [137] and a relu activation. Each architecture has four to five repetitions of this basic layer. We call this model family CBR. Depending on the choice of the convolutional filter size (fixed for the entire architecture), the number of channels and layers, we get a family of architectures with size ranging from a third of the standard ImageNet model size (CBR-LargeT, CBR-LargeW) to one twentieth the size (CBR-Tiny). Full architecture details are in the Appendix.

### 6.2.2 Results

We evaluate three repetitions of the different models and initializations (random initialization vs pretrained weights) on the two medical tasks, with the result shown in Tables 6.1, 6.2. There are two possibilities for repetitions of transfer

| Dataset | Model Architecture | Random Init | Transfer |
|---|---|---|---|
| Retina | Resnet-50 | 96.4% ± 0.05 | 96.7% ± 0.04 |
| Retina | Inception-v3 | 96.6% ± 0.13 | 96.7% ± 0.05 |
| Retina | CBR-LargeT | 96.2% ± 0.04 | 96.2% ± 0.04 |
| Retina | CBR-LargeW | 95.8% ± 0.04 | 95.8% ± 0.05 |
| Retina | CBR-Small | 95.7% ± 0.04 | 95.8% ± 0.01 |
| Retina | CBR-Tiny | 95.8% ± 0.03 | 95.8% ± 0.01 |

| Dataset | Model Architecture | Parameters | ImageNet Top5 |
|---|---|---|---|
| Retina | Resnet-50 | 23570408 | 92.% ± 0.06 |
| Retina | Inception-v3 | 22881424 | 93.9% |
| Retina | CBR-LargeT | 8532480 | 77.5% ± 0.03 |
| Retina | CBR-LargeW | 8432128 | 75.1% ± 0.3 |
| Retina | CBR-Small | 2108672 | 67.6% ± 0.3 |
| Retina | CBR-Tiny | 1076480 | 73.5% ± 0.05 |

Table 6.1: **Transfer learning and random initialization perform comparably across both standard ImageNet architectures and simple, lightweight CNNs for AUCs from diagnosing moderate DR. Both sets of models also have similar AUCs, despite significant differences in size and complexity.** Model performance on DR diagnosis is also not closely correlated with ImageNet performance, with the small models performing poorly on ImageNet but very comparably on the medical task.

learning: we can have a fixed set of pretrained weights and multiple training runs from that initialization, or for each repetition, first train from scratch on ImageNet and then fine-tune on the medical task. We opt for evaluating the former, as that is the standard method used in practice. For all models except for Inceptionv3, we first train on ImageNet to get the pretrained weights. For Inceptionv3, we used the pretrained weights provided by [293].

Table 6.1 shows the model performances on the Retina data (AUC of identifying moderate Diabetic Retinopathy (DR), described in Section 6.1), along with ImageNet top 5 accuracy. Firstly, we see that transfer learning has minimal effect on performance, not helping the smaller CBR architectures at all, and only providing a fraction of a percent gain for Resnet and Inception. Next, we see that despite the significantly lower performance of the CBR architectures on

| Model Architecture | Atelectasis | Cardiomegaly | Consolidation | Edema | Pleural Effusion |
|---|---|---|---|---|---|
| Resnet-50 | 79.52±0.31 | 75.23±0.35 | 85.49±1.32 | 88.34±1.17 | 88.70±0.13 |
| Resnet-50 (trans) | 79.76±0.47 | 74.93±1.41 | 84.42±0.65 | 88.89±1.66 | 88.07±1.23 |
| CBR-LargeT | 81.52±0.25 | 74.83±1.66 | 88.12±0.25 | 87.97±1.40 | 88.37±0.01 |
| CBR-LargeT (trans) | 80.89±1.68 | 76.84±0.87 | 86.15±0.71 | 89.03±0.74 | 88.44±0.84 |
| CBR-LargeW | 79.79±0.79 | 74.63±0.69 | 86.71±1.45 | 84.80±0.77 | 86.53±0.54 |
| CBR-LargeW (trans) | 80.70±0.31 | 77.23±0.84 | 86.87±0.33 | 89.57±0.34 | 87.29±0.69 |
| CBR-Small | 80.43±0.72 | 74.36±1.06 | 88.07±0.60 | 86.20±1.35 | 86.14±1.78 |
| CBR-Small (trans) | 80.18±0.85 | 75.24±1.43 | 86.48±1.13 | 89.09±1.04 | 87.88±1.01 |
| CBR-Tiny | 80.81±0.55 | 75.17±0.73 | 85.31±0.82 | 84.87±1.13 | 85.56±0.89 |
| CBR-Tiny (trans) | 80.02±1.06 | 75.74±0.71 | 84.28±0.82 | 89.81±1.08 | 87.69±0.75 |

Table 6.2: **Transfer learning provides mixed performance gains on chest x-rays.** Performances (AUC%) of diagnosing different pathologies on the CheXpert dataset. Again we see that transfer learning does not help significantly, and much smaller models performing comparably.

ImageNet, they perform very comparably to Resnet and Inception on the Retina task. These same conclusions are seen on the chest x-ray results, Table 6.2. Here we show the performance AUC for the five different pathologies (Section 6.1). We again observe mixed gains from transfer learning. For Atelectasis, Cardiomegaly and Consolidation, transfer learning performs slightly worse, but helps with Edema and Pleural Effusion.

### 6.2.3 The Very Small Data Regime

We conducted additional experiments to study the effect of transfer learning in the very small data regime. Most medical datasets are significantly smaller than ImageNet, which is also the case for our two datasets. However, our datasets still have around two hundred thousand examples, and other settings many only have a few thousand. To study the effects in this very small data regime, we trained models on only 5000 datapoints on the Retina dataset, and examined the effect of transfer learning. The results, in Table 6.3, suggest that

| Model | Rand Init | Pretrained |
|---|---|---|
| Resnet50 | 92.2% | 94.6% |
| CBR-LargeT | 93.6% | 93.9% |
| CBR-LargeW | 93.6% | 93.7% |

Table 6.3: **Benefits of transfer learning in the small data regime are largely due to architecture size.** AUCs when training on the Retina task with only 5000 datapoints. We see a bigger gap between random initialization and transfer learning for Resnet (a large model), but not for the smaller CBR models.

while transfer learning has a bigger effect with very small amounts of data, there is a confounding effect of model size – transfer primarily helps the large models (which are designed to be trained with a million examples) and smaller models again show little difference between transfer and random initialization.

## 6.3 Representational Analysis of the Effects of Transfer

In Section 6.2 we saw that transfer learning and training from random initialization result in very similar performance across different neural architectures and tasks. This gives rise to some natural questions about the effect of transfer

learning on the kinds of *representations* learned by the neural networks. Most fundamentally, does transfer learning in fact result in any representational differences compared to training from random initialization? Or are the effects of the initialization lost? Does feature reuse take place, and if so, where exactly? In this section, we provide some answers to these basic questions. Our approach directly analyzes and compares the hidden representations learned by different populations of neural networks, using (SV)CCA [253, 220], revealing an important dependence on *model size*, and differences in behavior between lower and higher layers. These insights, combined with results Section 6.4 suggest new, hybrid approaches to transfer learning.

**Quantitatively Studying Hidden Representations with (SV)CCA** To understand how pretraining affects the features and representations learned by the models, we would like to (quantitatively) study the learned intermediate functions (latent layers). Analyzing latent representations is challenging due to their complexity and the lack of any simple mapping to inputs, outputs or other layers. A recent tool that effectively overcomes these challenges is (Singular Vector) Canonical Correlation Analysis, (SV)CCA [253, 220], which has been used to study latent representations through training, across different models, alternate training objectives, and other properties [253, 220, 278, 201, 95, 169, 329]. Rather than working directly with the model parameters or neurons, CCA works with *neuron activation vectors* — the ordered collection of outputs of the neuron on a sequence of inputs. Given the activation vectors for two sets of neurons (say, corresponding to distinct layers), CCA seeks linear combinations of each that are as correlated as possible. We adapt existing CCA methods to prevent the size of the activation sets from overwhelming the computation in large models (details in Appendix D.3), and apply them to compare the latent representations

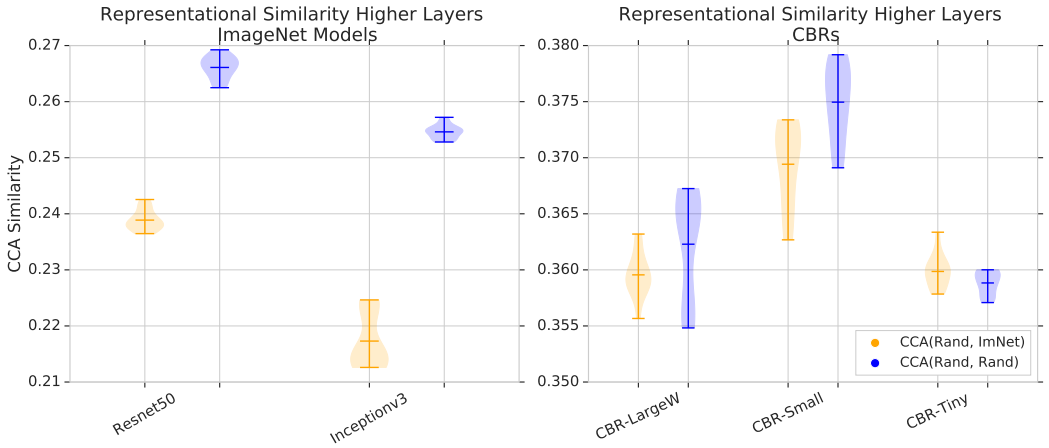



Figure 6.2: **Pretrained weights give rise to different hidden representations than training from random initialization for large models.** We compute CCA similarity scores between representations learned using pretrained weights and those from random initialization. We do this for the top two layers (or stages for Resnet, Inception) and average the scores, plotting the results in orange. In blue is a baseline similarity score, for representations trained from different random initializations. We see that representations learned from random initialization are more similar to each other than those learned from pretrained weights for larger models, with less of a distinction for smaller models.

of corresponding hidden layers of different pairs of neural networks, giving a CCA similarity score of the learned intermediate functions.

**Transfer Learning and Random Initialization Learn Different Representations** Our first experiment uses CCA to compare the similarity of the hidden representations learned when training from pretrained weights to those learned when training from random initialization. We use the representations learned at the top two layers (for CBRs) or stages (for Resnet, Inception) before the output layer, averaging their similarity scores. As a baseline to compare to, we also look at CCA similarity scores for the same representations when training from random initialization with two different seeds (different initializations and gradient updates). The results are shown in Figure 6.2. For larger models (Resnet, Inception), there is a clear difference between representations, with the similarity of representations between training from random initialization and pretrained

weights (orange) noticeably lower than representations learned independently from different random initializations (blue). However for smaller models (CBRs), the functions learned are more similar.

**Larger Models Change Less Through Training** The reasons underlying this difference between larger and smaller models becomes apparent as we further study the hidden representations of all the layers. We find that *larger models change much less during training*, especially in the lowest layers. This is true *even when they are randomly initialized*, ruling out feature reuse as the sole cause, and implying their overparametrization for the task. This is in line with other recent findings [371].

In Figure 6.3, we look at per-layer representational similarity before/after finetuning, which shows that the lowest layer in Resnet (a large model), is significantly more similar to its initialization than in the smaller models. This plot also suggests that any serious feature reuse is restricted to the lowest couple of layers, which is where similarity before/after training is clearly higher for pretrained weights vs random initialization. In Figure 6.4, we plot the CCA similarity scores between representations using pretrained weights and random initialization *at initialization* vs after training, for the lowest layer (conv1) as well as higher layers, for Resnet and CBR-Small. Large models changing less through training is evidenced by a surprising correlation between the CCA similarities for Resnet conv1, which is not true for higher layers or the smaller CBR-Small model.

**Filter Visualizations and the Absence of Gabors** As a final study of how pretraining affects the model representations, we visualize some of the filters

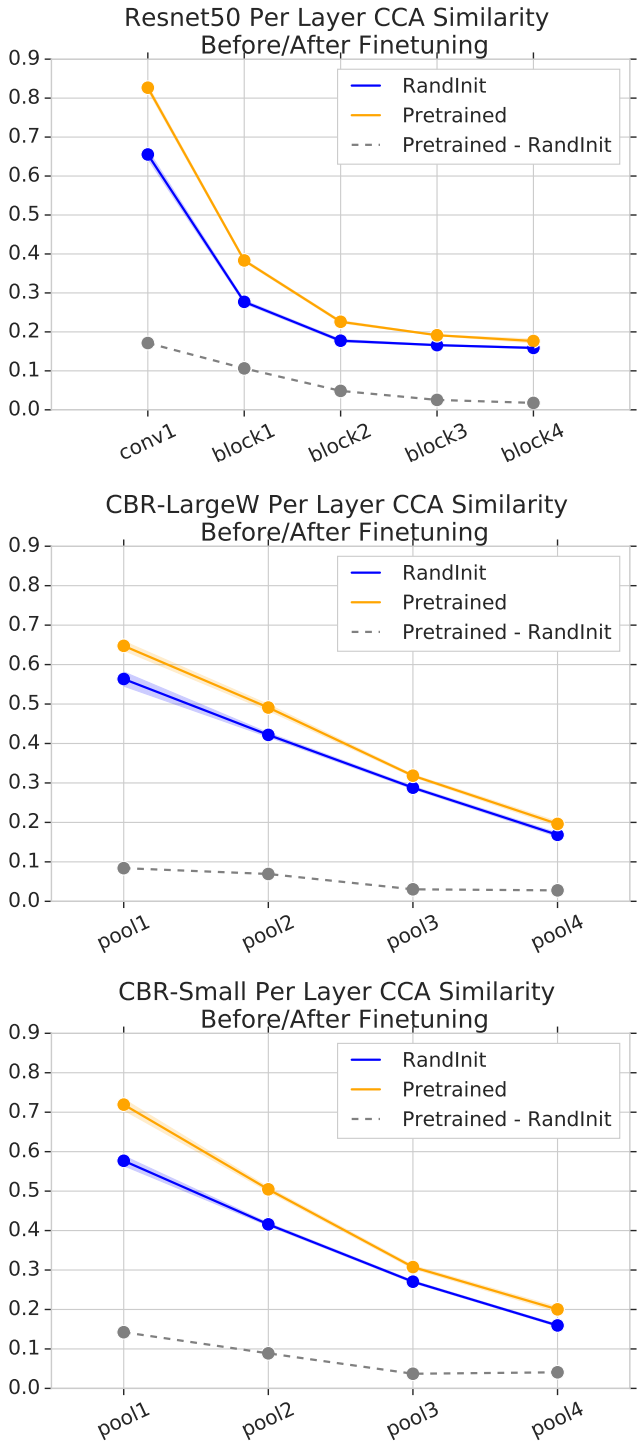



Figure 6.3: **Per-layer CCA similarities before and after training on medical task.** For all models, we see that the lowest layers are most similar to their initializations, and this is especially evident for Resnet50 (a large model). We also see that feature reuse is mostly restricted to the bottom two layers (stages for Resnet) — the only place where similarity with initialization is significantly higher for pretrained weights (grey dotted lines shows the difference in similarity scores between pretrained and random initialization).

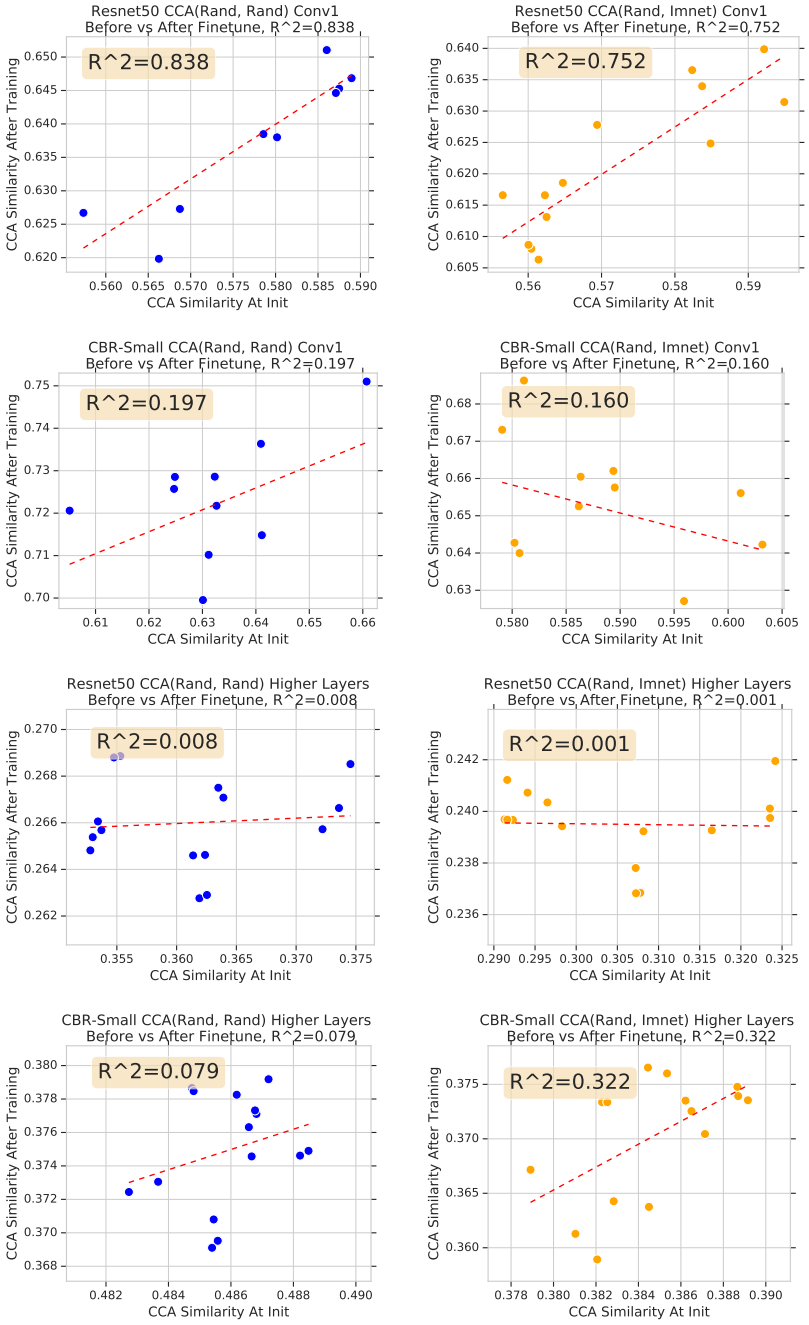
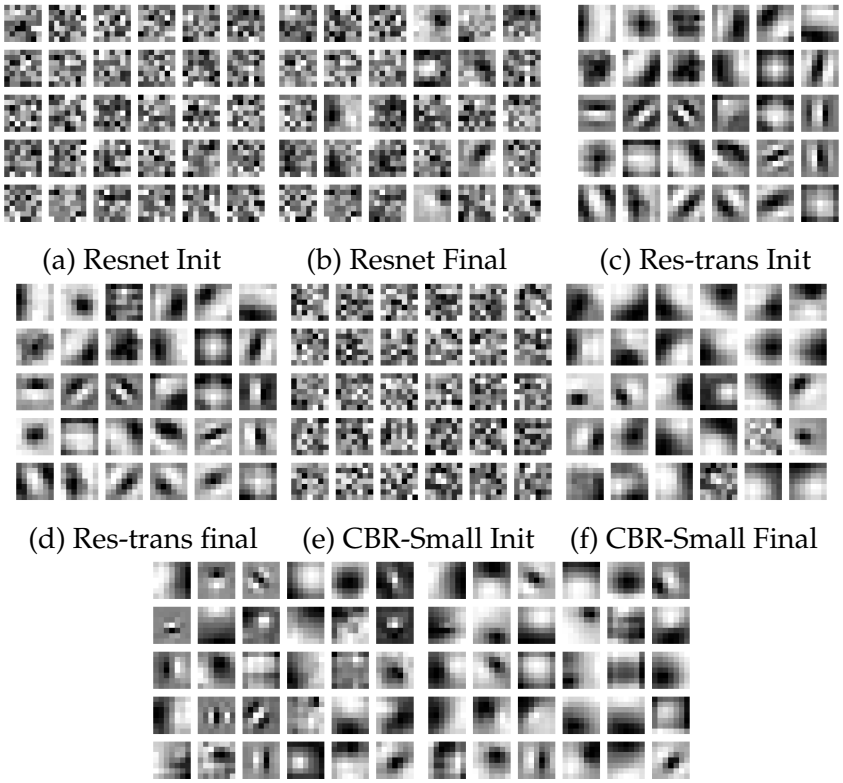



Figure 6.4: **Large models move less through training at lower layers: similarity at initialization is highly correlated with similarity at convergence for large models.** We plot CCA similarity of Resnet (conv1) initialized randomly and with pretrained weights at (i) initialization, against (ii) CCA similarity of the converged representations (top row second from left.) We also do this for two different random initializations (top row, left). In *both* cases (even for random initialization), we see a surprising, strong correlation between similarity at initialization and similarity after convergence ($R^2 = 0.75, 0.84$). This is not the case for the smaller CBR-Small model, illustrating the overparametrization of Resnet for the task. Higher must likely change much more for good task performance.

of conv1 for Resnet and CBR-Small (both 7x7 kernels), before and after training on the Retina task. The filters are shown in Figure 6.5, with visualizations for chest x-rays in the Appendix. These add evidence to the aformentioned observation: the Resnet filters change much less than those of CBR-Small. In contrast, CBR-Small moves more from its initialization, and has more similar learned filters in random and pretrained initialization. Interestingly, CBR-Small does not appear to learn Gabor filters when trained from scratch (bottom row second column). Comparing the third and fourth columns of the bottom row, we see that CBR-Small even erases some of the Gabor filters that it is initialized with in the pretrained weights.

## 6.4 Convergence: Feature Independent Benefits and Weight Transfusion

In this section, we investigate the effects of transfer learning on convergence speed, finding that: (i) surprisingly, transfer offers *feature independent* benefits to convergence simply through better weight scaling (ii) using pretrained weights from the lowest two layers/stages has the biggest effect on convergence — further supporting the findings in the previous section that any meaningful feature reuse is concentrated in these lowest two layers (Figure 6.3.) These results suggest some hybrid approaches to transfer learning, where only a subset of the pretrained weights (lowest layers) are used, with a lightweight redesign to the top of the network, and even using entirely *synthetic* features, such as synthetic Gabor filters (Appendix D.6.3). We show these hybrid approaches capture most of the benefits of transfer and enable greater flexibility in its application.



(a) Resnet Init (b) Resnet Final (c) Res-trans Init

(d) Res-trans final (e) CBR-Small Init (f) CBR-Small Final

(g) CBR Trans (h) CBR Trans Final

Figure 6.5: **Visualization of conv1 filters shows the remains of initialization after training in Resnet, and the lack of and erasing of Gabor filters in CBR-Small.** We visualize the filters before and after training from random initialization and pretrained weights for Resnet (top row) and CBR-Small (bottom row). Comparing the similarity of (e) to (f) and (g) to (h) shows the limited movement of Resnet through training, while CBR-Small changes much more. We see that CBR does not learn Gabor filters when trained from scratch (f), and also erases some of the pretrained Gabors (compare (g) to (h).)

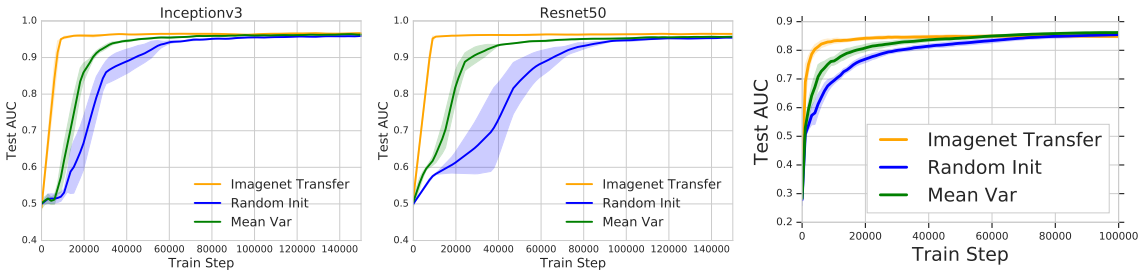



Figure 6.6: **Using only the scaling of the pretrained weights (Mean Var Init) helps with convergence speed**. The figures compare the standard transfer learning and the *Mean Var* initialization scheme to training from scratch. On both the Retina data (a-b) and the CheXpert data (c) (with Resnet50 on the *Consolidation* disease), we see convergence speedups.

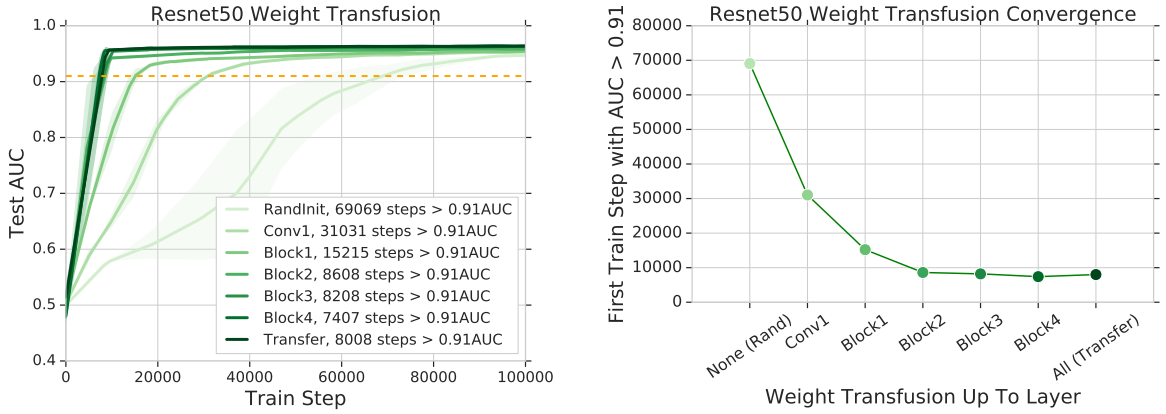



Figure 6.7: **Reusing a subset of the pretrained weights (weight transfusion), further supports only the lowest couple of layers performing meaningful feature reuse.** We initialize a Resnet with a contiguous subset of the layers using pretrained weights (weight transfusion), and the rest randomly, and train on the Retina task. On the left, we show the convergene plots when transfusing up to conv1 (just one layer), up to block 1 (conv1 and all the layers in block1), etc up to full transfer. On the right, we plot the number of train steps taken to reach 91% AUC for different numbers of transfused weights. Consistent with findings in Section 6.3, we observe that reusing the lowest layers leads to the greatest gain in convergence speed. Perhaps surprisingly, just reusing conv1 gives the greatest marginal convergence speedup, even though transfusing weights for a block means several new layers are using pretrained weights.

**Feature Independent Benefits of Transfer: Weight Scalings** We consistently observe that using pretrained weights results in faster convergence. One explanation for this speedup is that there is significant feature reuse. However, the results of Section 6.3 illustrate that there are many confounding factors, such as model size, and feature reuse is likely limited to the lowest layers. We thus tested to see whether there were *feature independent* benefits of the pretrained weights, such as *better scaling*. In particular, we initialized a *iid weights* from $\mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2)$, where $\tilde{\mu}$ and $\tilde{\sigma}^2$ are the mean and variance of $\tilde{W}$, the pretrained weights. Doing this for each layer separately inherits the scaling of the pretrained weights, but destroys all of the features. We called this the *Mean Var* init, and found that it significantly helps speed up convergence (Figure 6.6.) Several additional experiments studying batch normalization, weight sampling, etc are in the Appendix.

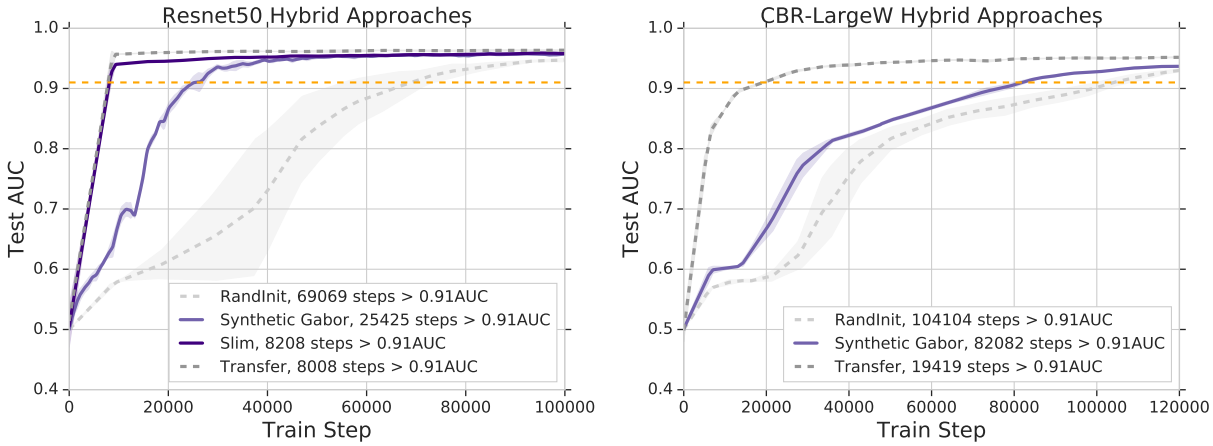



Figure 6.8: **Hybrid approaches to transfer learning: reusing a subset of the weights and slimming the remainder of the network, and using synthetic Gabors for conv1.** For Resnet, we look at the effect of reusing pretrained weights up to Block2, and slimming the remainder of the network (halving the number of channels), randomly initializing those layers, and training end to end. This matches performance and convergence of full transfer learning. We also look at initializing conv1 with *synthetic Gabor filters* (so *no* use of pretrained weights), and the rest of the network randomly, which performs equivalently to reusing conv1 pretrained weights. This result generalizes to different architectures, e.g. CBR-LargeW on the right.

**Weight Transfusions and Feature Reuse** We next study whether the results suggested by Section 6.3, that meaningful feature reuse is restricted to the lowest two layers/stages of the network is supported by the effect on convergence speed. We do this via a *weight transfusion* experiment, transfering a contiguous set of some of the pretrained weights, randomly initializing the rest of the network, and training on the medical task. Plotting the training curves and steps taken to reach a threshold AUC in Figure 6.7 indeed shows that using pretrained weights for lowest few layers has the biggest training speedup. Interestingly, just using pretrained weights for conv1 for Resnet results in the largest gain, despite transfusion for a Resnet block meaning multiple layers are now reusing pretrained weights.

**Takeaways: Hybrid Approaches to Transfer Learning** The transfusion re-

sults suggest some hybrid, more flexible approaches to transfer learning. Firstly, for larger models such as Resnet, we could consider reusing pretrained weights up to e.g. Block2, redesiging the top of the network (which has the bulk of the parameters) to be more lightweight, initializing these layers randomly, and training this new Slim model end to end. Seeing the disproportionate importance of conv1, we might also look at the effect of initializing conv1 with *synthetic Gabor filters* (see Appendix D.6.3 for details) and the rest of the network randomly. In Figure 6.8 we illustrate these hybrid approaches. Slimming the top of the network in this way offers the same convergence and performance as transfer learning, and using synthetic Gabors for conv1 has *the same effect* as pretrained weights for conv1. These variants highlight many new, rich and flexible ways to use transfer learning.

## 6.5 Chapter Summary and Discussion

In this chapter, we have present results on many central questions on transfer learning for medical imaging applications. Having benchmarked both standard ImageNet architectures and non-standard lightweight models (itself an underexplored question) on two large scale medical tasks, we find that transfer learning offers limited performance gains and much smaller architectures can perform comparably to the standard ImageNet models. Our exploration of representational similarity and feature reuse reveals surprising correlations between similarities at initialization and after training for standard ImageNet models, providing evidence of their overparametrization for the task. We also find that meaningful feature reuse is concentrated at the lowest layers and explore more

flexible, hybrid approaches to transfer suggested by these results, finding that such approaches maintain all the benefits of transfer and open up rich new possibilities. We also demonstrate feature-independent benefits of transfer learning for better weight scaling and convergence speedups.

# Part IV

# Human-AI Collaboration

CHAPTER 7

# DIRECT UNCERTAINTY PREDICTION FOR MEDICAL SECOND OPINIONS

In the following two chapters, we take a fully trained AI system and examine considerations for being able to meaningfully combine it with human experts, a central factor in successful deployment in applications such as medicine. In this chapter, we begin by studying methods that can enable these AI systems to effectively predict *human expert error*. This predictive task is both of independent interest (e.g. in flagging patients who might benefit from a medical second opinion), and crucially, in Chapter 8, it informs the effective combination of human experts and the AI system. This provides better performance than either entity alone, and also raises broader points on the evaluation of machine learning systems for deployment.

## 7.1 The Doctor Disagreement Problem and Overview of Results

In both the practice of machine learning and the practice of medicine, a serious challenge is presented by disagreements amongst human labels. Machine learning classification models are typically developed on large datasets consisting of $(x_i, y_i)$ (data instance, label) pairs. These are collected [271, 344] by assigning each raw instance $x_i$ to multiple human evaluators, yielding several labels $y_i^{(1)}, y_i^{(2)}, ...y_i^{(n_i)}$. Unsurprisingly, these labels often have disagreements amongst them and must be carefully aggregated to give a single target value.

This label disagreement issue becomes a full-fledged clinical problem in the healthcare domain. Despite the human labellers now being highly trained medical experts (doctors), disagreements (on the diagnosis) persist [324, 4, 1, 102, 255]. One example is [324], where agreement between referral and final diagnoses in a cohort of two hundred and eighty patients is studied. *Exact* agreement is only found in 12% of cases, but more concerningly, 21% of cases have significant disagreements. This latter group also turns out to be the most costly to treat. Other examples are given by [53], a study of tuberculosis diagnosis, showing that radiologists disagree with colleagues 25% of the time, and with *themselves* 20% of the time, and [72], studying disagreement on cancer diagnosis from breast biopsies.

These disagreements arise not solely from random noise [268], but from expert judgment and bias. In particular, some patient cases $x_i$ intrinsically contain features that result in greater expert uncertainty (e.g. Figure 7.2.) This motivates applying machine learning to *predict* which patients are likely to give rise to the most doctor disagreement. We call this the *medical second opinion* problem. Such a model could be deployed to automatically identify patients that might need a second doctor's opinion.

Mathematically, given a patient instance $x_i$, we are interested in assigning a scalar uncertainty score to $x_i$, $h(x_i)$ that reflects the amount of expert disagreement on $x_i$. For each $x_i$, we have multiple labels $y_i^{(1)}, y_i^{(2)}, ...y_i^{(n_i)}$, each corresponding to a different individual doctor's grade.

One natural approach is to first train a classifier mapping $x_i$ to the $y_i^{(j)}$, e.g. via the empirical distribution of labels $\hat{\mathbf{p}}_i$. For ungraded examples, a measure of

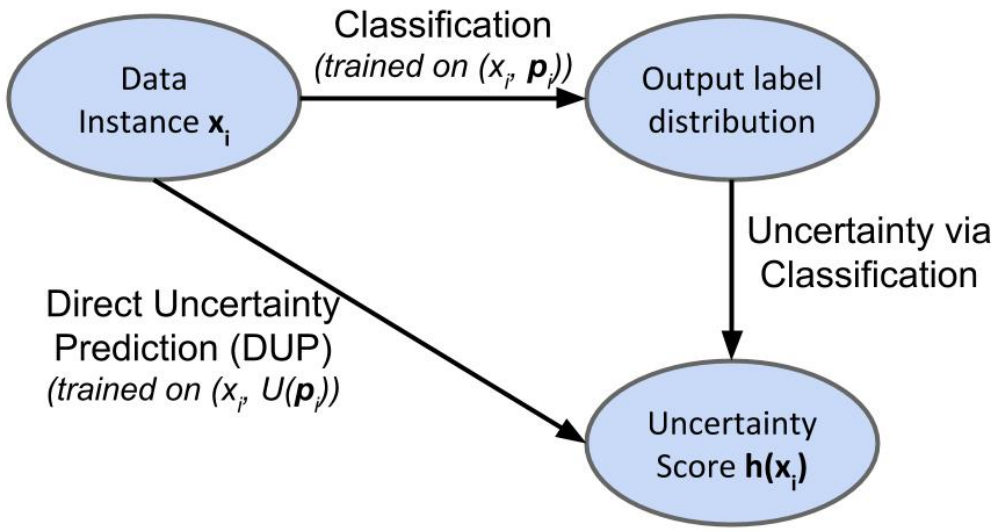



Figure 7.1: **Different ways of computing an uncertainty scores.** An uncertainty score $h(x_i)$ for $x_i$ can be computed by the two step process of Uncertainty via Classification: training a classifier on pairs (data instance, empirical grade distribution from $y_i^{(j)}$) $(x_i, \mathbf{p}_i)$, and then post processing the classifier output distribution to get an uncertainty score. $h(x_i)$ can also be learned directly on $x_i$, i.e. Direct Uncertainty Prediction. DUP models are trained on pairs (data instance, target uncertainty function on empirical grade distribution), $(x_i, U(\mathbf{p}_i))$. Theoretical and empirical results support the greater effectiveness of Direct Uncertainty Prediction.

spread of the output distribution of the classifier (e.g. variance) could be used to give a score. We call this *Uncertainty via Classification (UVC)*.

An alternate approach, *Direct Uncertainty Prediction (DUP)*, is to learn a function $h_{dup}$ directly mapping $x_i$ to a scalar uncertainty score. The basic contrast with Uncertainty via Classification is illustrated in Figure 7.1. Our central methodological finding is that Direct Uncertainty Prediction (provably) works better than the two step process of Uncertainty via Classification.

In particular, our three main contributions are the following:

1. We define simple methods of performing Direct Uncertainty Prediction on

data instances $x_i$ with multiple noisy labels. We prove that under a natural model for the data, DUP gives an *unbiased* estimate of the true uncertainty scores $U(x_i)$, while Uncertainty via Classification has a bias term. We then demonstrate this in a synthetic setting of mixtures of Gaussians, and on an image blurring detection task on the standard SVHN and CIFAR-10 benchmarks.

2. We train UVC and DUP models on a large-scale medical imaging task. As predicted by the theory, we find that DUP models perform better at identifying patient cases that will result in large disagreements amongst doctors.

3. On a small gold standard *adjudicated* test set, we study how well our existing DUP and UVC models can identify patient cases where the individual doctor grade disagrees with a consensus *adjudicated* diagnosis. This adjudicated grade is a proxy for the best possible doctor diagnosis. All DUP models perform better than all UVC models on all evaluations on this task, in both an uncertainty score setting and a ranking application.

## 7.2 Direct Uncertainty Prediction

Our core prediction problem, motivated by identifying patients who need a *medical second opinion*, centers around learning a scalar *uncertainty scoring function* $h(x)$ on patient instances $x$, which signifies the amount of expert disagreement arising from $x$.

To do so, we must first define a *target* uncertainty scoring function $U(\cdot)$. Our

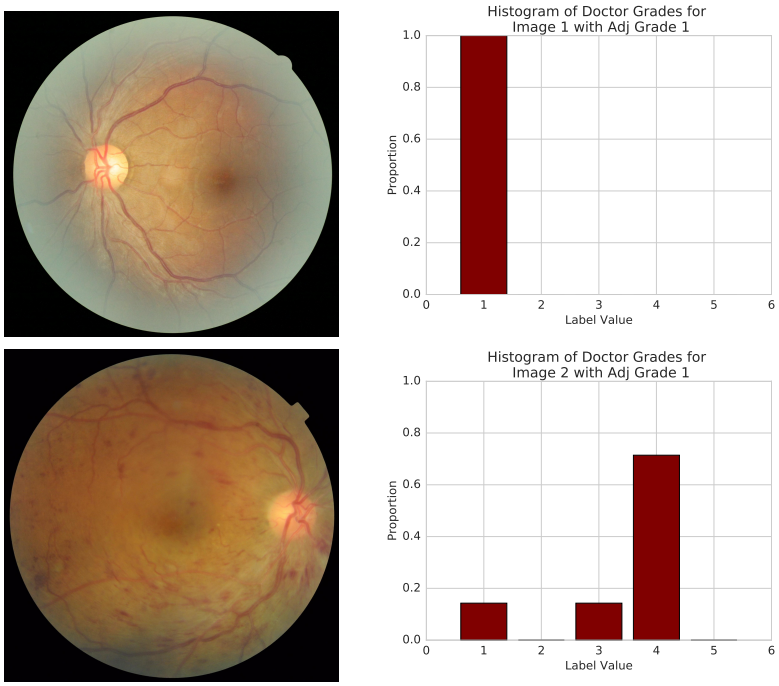



Figure 7.2: **Patient cases have features resulting in higher doctor disagreement.** The two rows give example datapoints in our dataset. The patient images $x_i, x_j$ are in the left column, and on the right we have the empirical probability distribution (histogram) of the multiple individual doctor DR grades. For the top image, all doctors agreed that the grade should be 1, while there was a significant spread for the bottom image. When later performing an *adjudication* process (Section 7.5), where doctors discuss their initial diagnoses with each other and come to a consensus, both patient cases were given an *adjudicated* DR grade of 1.

data consists of pairs of the form (patient features, multiple individual doctor labels), $(x_i; y_i^{(1)}, y_i^{(2)}, ...y_i^{(n_i)})$ (Figure 7.2). Letting $c_1, ..., c_k$ be the different possible doctor grades, we can define the empirical grade distribution – the empirical *histogram*: $\hat{\mathbf{p}}_\mathbf{i} = [\hat{p}_i^{(1)}, ..., \hat{p}_i^{(k)}]$, with

$$\hat{p}_i^{(l)} = \frac{\sum_j \mathbb{1}_{y_i^{(j)}=c_l}}{n_i}$$

Our target uncertainty scoring function $U(\cdot)$ then computes an uncertainty score for $x_i$ using the empirical histogram $\hat{\mathbf{p}}_\mathbf{i}$. One such function, which computes the probability that two draws from the empirical histogram will disagree is

$$U_{disagree}(x_i) = U_{disagree}(\hat{\mathbf{p}}_\mathbf{i}) = 1 - \sum_{l=1}^{k} (\hat{p}_i^{(l)})^2 \tag{1}$$

Another uncertainty score, which penalizes larger disagreements more, is the variance:

$$U_{var}(x_i) = U_{var}(\hat{\mathbf{p}}_\mathbf{i}) = \sum_{l=1}^{k} c_l \cdot (\hat{p}_i^{(l)})^2 - \left(\sum c_l \cdot \hat{p}_i^{(l)}\right)^2 \quad (2)$$

For a large family of these uncertainty scoring functions (including entropy, variance, etc) we can show that Direct Uncertainty Prediction gives an unbiased estimate of $U(\hat{\mathbf{p}}_i)$, whereas uncertainty via classification has a bias term.

The key observation is that while we want our model to predict doctor disagreement, it does not see all the patient information the doctors do. In particular, the model must predict doctor disagreement based off of only $x_i$ (in our setting, images). In contrast, human doctors see not only $x_i$ but other extensive information, such as patient medical history, family medical history, patient characteristics (age, symptom descriptions, etc) [57].

Letting $o$ denote all patient features seen by the doctors, we can think of $x_i$ as being the image of $o$ under a (many to one) mapping $g$, which hides the additional patient information, i.e. $x_i = g(o)$. Suppose there are $k$ possible doctor grades, $c_1, .., c_k$. Let $f$ denote the joint distribution over patient features and doctor grades. In particular, let $O$ be a random variable representing patient features, and $Y$ the doctor label for $O$. Then our density function assigns a probability to (patient features, doctor grade) pairs $(o, y)$.

This can also be defined with a vectorized version of the grades: let $Y_l = \mathbf{1}_{Y=c_l}$, the event that $O$ is diagnosed as $c_l$. Then we define the vector $\mathbf{Y} = [Y_1, ..., Y_k]$. $f$ is therefore also a density over the points $f(O = o, \mathbf{Y} = \mathbf{y})$. Let the marginal probability of the patient features be $f_O$, with $f_O = \int_{\mathbf{y}} f(O, \mathbf{y})$.

Given an uncertainty scoring function $U(\cdot)$, we would like to predict the disagreement in labels amongst doctors who have seen the patient features $O$. But as the patient features $O$ and doctor grades $\mathbf{Y}$ are jointly distributed according to $f$, this is just the uncertainty of the expected value of $\mathbf{Y}$ under the posterior of $\mathbf{Y}$ given $o$. In particular, we are interested in predicting:

$$U\left(\int_{\mathbf{y}} \mathbf{y} \cdot f(\mathbf{Y} = \mathbf{y}|O)\right) = U(\mathbb{E}[\mathbf{Y}|O])$$

This is a function taking as input a patient's features. For a particular patient's features $o$, we get a scalar uncertainty score given by

$$U(\mathbb{E}[\mathbf{Y}|O = o])$$

However, our model doesn't see $o$, but only $x = g(O)$. We make the mild assumptions that $Y$ is conditionally independent of $g(O)$ given $O$, and that $g(\cdot)$ truly hides information, loosely that $O|g(O) = x$ is not a point mass (see Appendix for details.) In this setting, direct uncertainty prediction, $h_{dup}$ computes the expectation of the uncertainty scores of all the possible posteriors, i.e.

$$\begin{aligned} h_{dup}(x) &= \mathbb{E}\left[U(\mathbb{E}[\mathbf{Y}|O])|g(O) = x\right] \\ &= \int_o U(\mathbb{E}[\mathbf{Y}|O = o]) f_O(o|g(O) = x) \end{aligned}$$

Uncertainty via classification $h_{uvc}$ does this in reverse order, first computing the expected posterior, and assigning an uncertainty score to that:

$$\begin{aligned} h_{uvc}(x) &= U(\mathbb{E}[\mathbf{Y}|g(O) = x]) \\ &= U\left(\int_o \mathbb{E}[\mathbf{Y}|O = o] f_O(o|g(O) = x)\right) \end{aligned}$$

In this setting we can show

**Theorem 5.** *Using the above notation*

*(i)* $h_{dup}$ *is an unbiased estimate of the true uncertainty*

*(ii) For any concave uncertainty scoring function* $U(\cdot)$ *(which includes* $U_{disagree}, U_{var}$*), uncertainty via classification,* $h_{uvc}$ *has a bias term.*

The full proof is the Appendix. A sketch is as follows: the unbiased result arises from the tower rule (law of total expectations). The bias of $h_{uvc}$ follows by the concavity of $U()$, Jensen's inequality, and the fact that $g(\cdot)$ truly hides some patient features. For $U_{disagree}$ and $U_{var}$, we can compute this bias term exactly (full computation in Appendix):

**Corollary 1.** *For* $U_{disagree}, U_{var}$ *the bias term is:*

*(i) Bias of* $h_{uvc}$ *with* $U_{disagree}$*:*

$$\mathbb{E}_{g(O)}\left[\sum_l Var_{O|g(O)}\Big(\mathbb{E}[Y_l|O]\Big|g(O)\Big)\right]$$

*(ii) Bias of* $h_{uvc}$ *with* $U_{var}$*:*

$$\mathbb{E}_{g(O)}\left[Var_{O|g(O)}\left(\sum_l l \cdot \mathbb{E}[Y_l|O]\middle| g(O)\right)\right]$$

In Sections 7.4, 7.5 we train both Direct Uncertainty Prediction (DUP) models and Uncertainty Via Classification (UVC) models on a large scale medical imaging task. However, to gain intuition for the theoretical results, we first study a toy case on a mixture of Gaussians.

### 7.2.1 Toy Example on Mixture of Gaussians

To illustrate the formalism in a simplified setting, we consider the following pedagogical toy example. Suppose our data is generated by a mixture of $k$

| **Model Type** | $(3d, 5G)$ | $(5d, 4G)$ | $(10d, 4G)$ |
|---|---|---|---|
| UVC | 69.1% | 62.0% | 56.0% |
| DUP | **74.6**% | **71.2**% | **63.4**% |

Table 7.1: **DUP and UVC trained to predict disagreement on mixtures of Gaussians.** We train DUP and UVC models on different mixtures of Gaussians, with$(nd, mG)$ denoting a mixture of $m$ Gaussians in $m$ dimensions. Results are in percentage AUC over 3 repeats. The means of the Gaussians are drawn iid from a multivariate normal distribution (full setup in Appendix.) We see that the DUP model performs much better than the UVC model at identifying datapoints with high disagreement in the labels.

Gaussians. Let $f_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, and $q_i$ be mixture probabilities. Then $f(o, y = i) = q_i f_i(o)$ and the marginal $f_O(o) = \sum_{i=1}^{k} q_i f_i(o)$. Additionally, the probability of a particular class $l$ given $o$, $f(y = l|o)$ is simply $\frac{q_l f_l(o)}{\sum_{i=1}^{k} q_i f_i(o)}$.

**Two 1-D Gaussians:** As the first, most simple case, suppose we have two one dimensional Gaussians, the first, $f_1 = \mathcal{N}(-1, 1)$ and the second, $f_2 = \mathcal{N}(1, 1)$. Assume that the mixture probabilities $q_1, q_2$ are equal to 0.5. Given $o$ drawn from this mixture $q_1 f_1 + q_2 f_2$, we'd like to estimate $U(f(y|o))$. Suppose the model sees $x = g(o) = |o|$, the absolute value of $o$. Then, DUP can estimate the uncertainty exactly:

$$\begin{aligned}
\mathbb{E}\left[U(\mathbb{E}[\mathbf{Y}|O])|x = |o|\right] =& 0.5 \cdot U(\mathbb{E}[\mathbf{Y}|O = o]) \\
&+ 0.5 \cdot U(\mathbb{E}[\mathbf{Y}|O = -o]) \\
&= U(\mathbb{E}[\mathbf{Y}|O = o]) \\
&= U\Big([f(1|o), 1 - f(1|o)]\Big)
\end{aligned}$$

where the third line follows by the symmetry of the two distributions, with

$$f(1|o) = \frac{0.5 f_1(o)}{0.5 f_1(o) + 0.5 f_2(o)}$$

On the other hand, the expected posterior over labels in UVC, $\mathbb{E}[\mathbf{Y}|x = |o|]$, is just $[0.5, 0.5]$, as by symmetry, given $x = g(o) = |o|$, $o$ is equally likely to come from

$f_1$ or $f_2$. So UVC outputs a constant uncertainty score $U([0.5, 0.5])$ for all $x = |o|$, despite the true varying uncertainty scores.

**Training DUPs and UVCs on Mixture of Gaussians:** In Table 7.1 we train DUPs and UVCs on a few different mixture of Gaussian settings. We generate data $o$ from a Gaussian mixture with iid centers, and labels for the data using the posterior over the different centers given $o$. We use these labels to score $o$ on its uncertainty (using $U_{disagree}$). We then train a model on $x = g(o) = |o|$ to predict whether $x$ is low or high uncertainty. (Full details in Appendix.) We see that DUP performs consistently better than UVC.

### 7.2.2 Example on SVHN and CIFAR-10

Another empirical demonstration is given by training DUP and UVC to predict label agreement in an image blurring setting. For a source image in SVHN or CIFAR-10, we first apply a Gaussian blur, with a variance chosen for that source image. Then, we draw three noisy labels for the source image, where the noise distribution over labels corresponds to the severity of the image blur. For example, for an image that has a Gaussian blur of variance 0 (i.e. no blurring), the distribution over labels is a point mass on the true label. For an image that has been blurred severely, there is significant mass on incorrect labels. (Exact distributional details are given in the Appendix.) We train DUP and UVC models on this dataset and evaluate their ability to predict label disagreement. We again find that DUP models outperfom UVC models. This is despite the setting not directly mapping onto the statement of Theorem 5 – there is no obscuring function $g$. This suggests the benefits of DUP are more general than the precise

| Model | **SVHN** (disagree) | **CIFAR-10** (disagree) |
|---|---|---|
| UVC | 75.8% | 79.1% |
| DUP | **88.0**% | **85.3**% |

Table 7.2: **DUP and UVC trained to predict label disagreement corresponding to image blurring on SVHN and CIFAR-10.** DUP outperforms UVC on predicting label disagreement on SVHN and CIFAR-10, where the labels are drawn from a noisy distribution that varies depending on how much blurring the source image has been subjected to. Full details in Appendix.

theoretical setting. We also observe that the DUP and UVC models learn different features (see Appendix.)

## 7.3 Related Work

The challenges posed by expert disagreement is an important one, and prior work has put forward several approaches to address some of these issues. Under the assumption that the noise distribution is conditionally independent of the data instance given the true label, [222, 299, 263, 285] provide theoretical analysis along with algorithms to denoise the labels as training progresses, or efficiently collect new labels. However, the conditional independence assumption does not hold in our setting (Figure 7.2.) Other work relaxes this assumption by defining a domain specific generative model for how noise arises [217, 351, 326] with some methods using additional clean data to pretrain models to form a good prior for learning. Related techniques have also been explored in semantic segmentation [104, 159]. Modeling uncertainty in the context of noisy data has also been looked at through Bayesian techniques [148, 310], and (for different models) in the context of crowdsourcing by [345, 339]. A related line of work [55, 344] has looked at studying the per labeler error rates, which also requires

| Task | | Model Type | AUC |
|---|---|---|---|
| Variance Prediction | UVC | Histogram-E2E | 70.6% |
| Variance Prediction | UVC | Histogram-PC | 70.6% |
| Variance Prediction | DUP | Variance-E2E | 72.9% |
| Variance Prediction | DUP | Variance-P | 74.4% |
| Variance Prediction | DUP | Variance-PR | 74.6% |
| Variance Prediction | DUP | Variance-PRC | **74.8%** |
| Disagreement Prediction | UVC | Histogram-E2E | 73.4% |
| Disagreement Prediction | UVC | Histogram-PC | 76.6% |
| Disagreement Prediction | DUP | Disagree-P | **78.1%** |
| Disagreement Prediction | DUP | Disagree-PC | **78.1%** |
| Variance Prediction | DUP | Disagree-PC | 73.3% |
| Disagreement Prediction | DUP | Variance-PRC | 77.3% |

Table 7.3: **Performance (percentage AUC) averaged over three runs for UVC and DUPs on Variance Prediction and Disagreement Prediction tasks**. The UVC baselines, which first train a classifier on the empirical grade histogram, are denoted Histogram-. DUPs are trained on either $T_{train}^{(disagree)}$ or $T_{train}^{(var)}$, and denoted Disagree-, Variance- respectively. The top two sets of rows shows the performance of the baseline (and a strengthened baseline Histogram-PC using Prelogit embeddings and Calibration) compared to Variance and Disagree DUPs on the (1) Variance Prediction task (evaluation on $T_{test}^{(var)}$) and (2) Disagreement Prediction task (evaluation on $T_{test}^{(disagree)}$). We see that in both of these settings, the DUPs perform better than the baselines. Additionally, the third set of rows shows tests a Variance DUP on the disagreement task, and vice versa for the Disagreement DUP. We see that both of these also perform better than the baselines.

the additional information of labeler ids, an assumption we relax. Most related is [100], where a multiheaded neural network is used to model different labelers. Surprisingly however, the best model is independent of image features, which is the source of signal in our experiments.

## 7.4 Doctor Disagreements in DR

Our main application studies the effectiveness of Direct Uncertainty Predictors (DUPs) and Uncertainty via Classification (UVC) in identifying patient cases with

high disagreements amongst doctors in a large-scale medical imaging setting. These patients stand to benefit most from a medical second opinion.

The task contains patient data in the form of retinal fundus photographs [102], large (587 x 587) images of the back of the eye. These photographs can be used to diagnose the patient with different kinds of eye diseases. One such eye disease is Diabetic Retinopathy (DR), which, despite being treatable if caught early enough, remains a leading cause of blindness [6].

DR is graded on a 5-class scale: a grade of 1 corresponds to *no* DR, 2 to *mild* DR, 3 to *moderate* DR, 4 to *severe* and 5 to *proliferative* DR [1]. There is an important clinical threshold at grade 3, with grades 3 and above corresponding to *referable* DR (needing immediate specialist attention), and 1, 2 being non-referable. Clinically, the most costly mistake is not referring referable patients, which poses a high risk of blindness.

Our main dataset $T$ has many features typical of medical imaging datasets. $T$ has larger but much fewer images than in natural image datasets such as ImageNet. Each image $x_i$ has a few (typically one to three) individual doctor grades $y_i^{(1)}, ..., y_i^{(n_i)}$. These grades are also very noisy, with more than 20% of the images having large (referable/non-referable) disagreement amongst the grades.

### 7.4.1 Task Setup

In this section we describe the setup for training variants of DUPs and UVCs using a train test split on $T$. We outline the resulting model performances in Table

7.3, which measure how successful the models are in identifying cases where doctors most disagree with each other and consequently where a medical second opinion might be most useful. In Section 7.5, we perform a different evaluation (disagreement with consensus) of the best performing DUPs and UVCs on a special, gold standard *adjudicated* test set. In both evaluation settings, we find that DUPs noticeably outperform UVCs.

The DUP and UVC models are trained and evaluated using a train/test split on $T$, $T_{train}, T_{test}$. This split is constructed using the patient ids of the $x_i \in T$, with 20% of patient ids being set aside to form $T_{test}$ and 80% to form $T_{train}$ (of which 10% is sometimes used as a validation set.) Splitting by patient ids is important to ensure that multiple images $x_i, x_j \in T$ corresponding to a single patient are correctly split [102].

We apply $U_{disagree}(\cdot)$ to the $x_i$ in $T_{train}, T_{test}$ with more than one doctor label to form a new train/test split $T_{train}^{(disagree)}, T_{test}^{(disagree)}$. We repeat this with $U_{var}(\cdot)$ to also form a train/test split $T_{train}^{(var)}, T_{test}^{(var)}$. These two datasets capture the two different medical interpretations of DR grades:

*Categorical Grade Interpretation*: The DR grades can be interpreted as categorical classes, as each grade has specific features associated with it. A grade of 2 always means microaneurysms, while a grade of 5 can refer to lesions or laser scars [1]. The $T_{train}^{(disagree)}, T_{test}^{(disagree)}$ data measures disagreement in this categorical setting.

*Continuous Grade Interpretation*: While there are specific features associated with each DR grade, patient conditions tend to progress sequentially through the different DR grades. The $T_{train}^{(var)}, T_{test}^{(var)}$ data thus accounts for the magnitude of

differences in doctor grades.

Having formed $T^{(disagree)}_{train}, T^{(disagree)}_{test}$ and $T^{(var)}_{train}, T^{(var)}_{test}$, which consist of pairs $(x_i, U_{disagree}(\hat{\mathbf{p}}_\mathbf{i}))$ and $(x_i, U_{var}(\hat{\mathbf{p}}_\mathbf{i}))$ respectively, we binarize the uncertainty scores $U_{disagree}(\hat{\mathbf{p}}_\mathbf{i}), U_{var}(\hat{\mathbf{p}}_\mathbf{i})$ into 0 (low uncertainty) or 1 (high uncertainty) to form our final prediction targets. We denote these $U^B_{disagree}(\hat{\mathbf{p}}_\mathbf{i}), U^B_{var}(\hat{\mathbf{p}}_\mathbf{i})$. More details on this can be found in Appendix Section E.4.

### 7.4.2 Models and First Experimental Results

We train both UVCs and DUPs on this data. All models rely on an Inception-v3 base that, following prior work [102], is initialized with pretrained weights on ImageNet. The UVC is performed by first training a classifier $h_c$ on $(x_i, \hat{\mathbf{p}}_\mathbf{i})$ pairs in $T_{train}$. The output probability distribution of the classifier, $\tilde{\mathbb{p}}_i = h_c(x_i)$ is then used as input to the uncertainty scoring function $U(\cdot)$, i.e. $h_{uvc}(x_i) = U \circ h_c(x_i)$ In contrast, the DUPs are trained directly on the pairs $(x_i, U^B_{disagree}(\hat{\mathbf{p}}_\mathbf{i}))$, $(x_i, U^B_{var}(\hat{\mathbf{p}}_\mathbf{i}))$, i.e. $h_{dup}(x_i)$ directly tries to learn the value of $U^B(\hat{\mathbf{p}}_\mathbf{i})$

The results of evaluating these models (on $T^{(disagree)}_{test}$ and $T^{(disagree)}_{var}$) are given in Table 7.3. The *Variance Prediction* task corresponds to evaluation on $T^{(disagree)}_{var}$, and the *Disagreement Prediction* task to evaluation on $T^{(disagree)}_{test}$. Both tasks correspond to identifying patients where there is high disagreement amongst doctors. As is typical in medical applications due to class imbalances, performance is given via area under the ROC curve (AUC) [102, 255].

From the first two sets of rows, we see that DUP models perform better than

their UVC counterparts on both tasks. The third set of rows shows the effect of using a variance DUP (Variance-PRC) on the disagreement task and a disagree DUP (Disagree-PC) on the variance task. While these don't perform as well as the best DUP models on their respective tasks, they still beat both the baseline and the strengthened baseline. Below we describe some of the different UVC and DUP variants, with more details in Appendix Section E.4.

**UVC Models** The UVC models are trained on (image, empirical grade histogram) $(x_i, \hat{\mathbf{p}}_i)$ pairs, and denoted *Histogram-* in Table 7.3. The simplest UVC is *Histogram-E2E*, the same model used in [102]. We improved this baseline by instead taking the prelogit embeddings of Histogram-E2E, and training a small neural network (fully connected, two hidden layers width 300) with temperature scaling (as in [103]) only on $x_i$ with multiple labels. This gave the strengthened baseline *Histogram-PC*.

**DUP Variance Models** The simplest Variance DUP is *Variance-E2E*, which is analogous to Histogram-E2E, except trained on $T^{(var)}_{train}$. This performed better than Histogram-E2E, but as $T^{(var)}_{train}$ is small for an Inception-v3, we trained a small neural network (fully connected, two hidden layers width 300) on the prelogit embeddings, called *Variance-P*. Small variants of *Variance-P* (details in Appendix Section E.4) give *Variance-PR*, and *Variance-PRC*.

**DUP Disagreement Models** Informed by the variance models, the *Disagree-P* model was designed exactly like the *Variance-P* model (a small fully connected network on prelogit embeddings), but trained on $T^{(disagree)}_{train}$. A small variant of this with calibration gave *Disagree-PC*.

In the Appendix, we demonstrate similar results using entropy as the uncer-

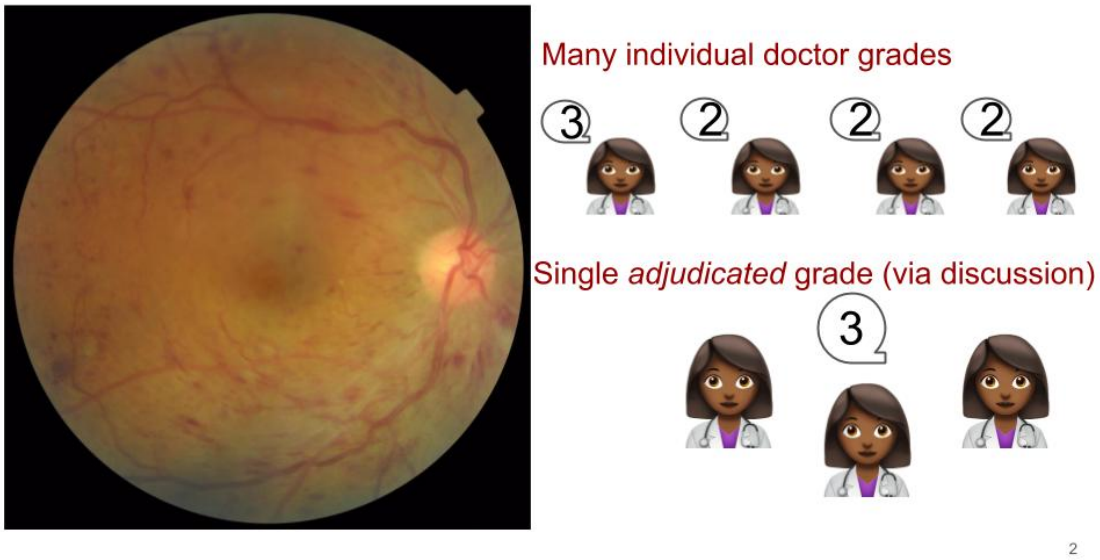



Figure 7.3: **Labels for the adjudicated dataset $A$.** The small, gold standard adjudicated dataset $A$ has a very different label structure to the main dataset $T$. Each image has many individual doctor grades (typically more than 10 grades). These doctors also tend to be specialists, with higher rates of agreement. Additionally, each image has a single *adjudicated* grade, where three doctors first grade the image individually, and then come together to discuss the diagnosis and finally give a single, *consensus diagnosis*.

tainty function, as well as experiments studying convergence speed and finite sample behaviour of DUP and UVC. We find that the performance gap between DUP and UVC is robust to train data size, and manifests early in training.

## 7.5 Predicting Disagreement with Consensus: Adjudicated Evaluation

Section 7.4 trained DUPs and UVCs on $T_{train}$, and evaluated them on their ability to identify patient cases where individual doctors were most likely to disagree with each other. Here, we take these trained DUPs/UVCs, and perform an

| | Model Type | Majority | Median | Majority = 1 |
|---|---|---|---|---|
| UVC | Histogram-E2E-Var | 78.1% | 78.2% | 81.3% |
| UVC | Histogram-E2E-Disagree | 78.5% | 78.5% | 80.5% |
| UVC | Histogram-PC-Var | 77.9% | 78.0% | 80.2% |
| UVC | Histogram-PC-Disagree | 79.0% | 78.9% | 80.8% |
| DUP | Variance-PR | 80.0% | 79.9% | 83.1% |
| DUP | Variance-PRC | 79.8% | 79.7% | 82.7% |
| DUP | Disagree-P | **81.0%** | 80.8% | **84.6%** |
| DUP | Disagree-PC | 80.9% | **80.9%** | 84.5% |

| | Model Type | Median = 1 | Referable |
|---|---|---|---|
| UVC | Histogram-E2E-Var | 78.1% | 85.5% |
| UVC | Histogram-E2E-Disagree | 77.0% | 84.2% |
| UVC | Histogram-PC-Var | 77.7% | 85.0% |
| UVC | Histogram-PC-Disagree | 79.2% | 84.8% |
| DUP | Variance-PR | 80.5% | 85.9% |
| DUP | Variance-PRC | 80.2% | 85.9% |
| DUP | Disagree-P | **81.9%** | **86.2%** |
| DUP | Disagree-PC | 81.8% | **86.2%** |

Table 7.4: **Evaluating models (percentage AUC) on predicting disagreement between an average individual doctor grade and the adjudicated grade.** We evaluate our models's performance using multiple different aggregation metrics (majority, median, binarized non-referable/referable median) as well as special cases of interest (no DR according to majority, no DR according to median). We observe that **all** direct uncertainty models (Variance-, Disagree-) outperform *all* classifier-based models (Histogram-) on *all* tasks.

*adjudicated* evaluation, to satisfy two additional goals.

Firstly, and most importantly, the clinical question of interest is not only in identifying patients where individual doctors disagree with each other, but cases where a more thorough diagnosis – the *best possible* doctor grade – would disagree significantly with the *individual* doctor grade. Evaluation on a gold-standard *adjudicated* dataset $A$ enables us to test for this: each image $x_i \in A$ not only has many individual doctor grades (by specialists in the disease) but also a single *adjudicated* grade. This grade is determined by a group of doctors seeking to reach a *consensus* on the diagnosis through discussion [167]. Figure 7.3 illustrates

the setup.

We can thus evaluate on this question by seeing if high model uncertainty scores correspond to disagreements between the (average) individual doctor grade and the adjudicated grade. More specifically, we compute different aggregations of the individual doctor grades for $x_i \in A$, and give a binary label for whether this aggregation agrees with the adjudicated grade (0 for agreement, 1 for disagreement). We then see if our model uncertainty scores is predictive of the binary label.

Secondly, our evaluation on $A$ also provides a more accurate reflection of our models's performance, with less confounding noise. The labels in $A$ (both individual and adjudicated) are much cleaner, with greater consistency amongst doctors. As $A$ is used *solely* for evaluation (all evaluated models are trained on $T_{train}$, Section 7.4), this introduces a distribution shift, but the predicted uncertainty scores transfer well. The results are shown in Table 7.4. We evaluate on several different aggregations of individual doctor grades. Like [102], we compare agreement between the majority vote of the individual grades and the adjudicated grades. To compensate for a bias of individual doctors giving lower DR grades [167], we also look at agreement between the median individual grade and adjudicated grade. Additionally, we look at referable/non-referable DR grade agreement. We binarize both the individual doctor grades and the adjudicated grade into 0/1 non-referable/referable, and check agreement between the median binarized grades and adjudicated grade. Finally, we also look at the special case where the average doctor grade is 1 (no DR), and compare agreement with the adjudicated grade.

We evaluate both baseline models (Histogram-E2E, Histogram-PC) as well as the best performing DUPs, (Variance-PR, Variance-PRC, Disagree-P, Disagree-PC.) The additional -Var, -Disagree suffixes on the baseline models indicate which uncertainty function ($U_{var}$ or $U_{disagree}$) was used to postprocess the classifier output distribution $\tilde{\mathbb{p}}$ to get an uncertainty score. We find that *all* DUPs outperform *all* the baselines on *all* evaluations.

### 7.5.1 Ranking Evaluation

A frequent practical challenge in healthcare is to *rank* cases in order of hardest (needing most attention) to easiest (needing least attention), [113]. Therefore, we evaluate how well our models can rank cases from greatest disagreement between the adjudicated and individual grades to least disagreement between the adjudicated and individual grades. To do this however, we need a *continuous* ground truth value reflecting this disagreement, instead of the binary 0/1 agree/disagree used above. One natural way to do this is to compute the *Wasserstein* distance between the empirical histogram (individual grade distribution) and the adjudicated grade.

At a high level, the Wasserstein distance computes the minimal cost required to move a probability distribution $\mathbf{p}^{(1)}$ to a probability distribution $\mathbf{p}^{(2)}$ with respect to a given metric $d(\cdot)$. In our setting, $\mathbf{p}^{(1)}$ is the empirical histogram $\hat{\mathbf{p}}_i$ of $x_i$, and $\mathbf{p}^{(2)}$ is the point mass at the adjudicated grade $a_i$. When one distribution is a point mass, the Wasserstein distance has a simple interpretation:

**Theorem 6.** *Let $\mathbf{p}^{(1)}$ and $\mathbf{p}^{(2)}$ be two probability distributions, with $\mathbf{p}^{(2)}$ a point mass with non-zero value $a$. Let $d(\cdot)$ be a given metric. The Wasserstein distance between $\mathbf{p}^{(1)}$*

*and* $\mathbf{p}^{(2)}$, $\|\mathbf{p}^{(1)} - \mathbf{p}^{(2)}\|_w$ *with respect to* $d(\cdot)$ *can be written as*

$$\|\mathbf{p}^{(1)} - \mathbf{p}^{(2)}\|_w = \mathbb{E}_{C\sim\mathbf{p}^{(1)}}[d(C, a)]$$

The proof is in Appendix E.6. In our setting, the theorem says that the (continuous) disagreement score for $x_i \in A$ is just the expected distance between a grade drawn from the empirical histogram and the adjudicated grade. We consider three different distance functions $d(\cdot)$: (a) the absolute value of the grade difference, (b) the 2 – *Wasserstein* distance, a metrization of the squared distance penalizing large grade differences more (details in Appendix E.6) and (c) a 0/1 binary agree/disagree metric, in line with the categorical and continous interpretations of DR grades, Section 7.4.

We compare the ranking induced by this continuous disagreement score on $A$ with the ranking induced by the model's predicted uncertainty scores. To evaluate the similarity of these rankings, we use Spearman's rank correlation [298], which takes a value between $[-1, 1]$. A $-1$ indicates perfect negative rank correlation, 1 a perfect positive rank correlation and 0 no correlation. The results are shown in Table 7.5. Similar to Table 7.4, we observe strong performance with DUPs: all DUPs beat all the baselines on all the different distances.

This task also enables a natural comparison between the models and doctors. In particular, we can compute a third ranking over $A$, by sampling $n$ individual doctor grades, and computing the Wasserstein distance between this subsampled empirical histogram and the adjudicated grade. This experiment tells us how many doctor grades are needed to give a ranking as accurate as the models. For DUPs, we need on average 5 doctors, while for the UVC baseline, we need on average 4 doctors.

## 7.6 Chapter Discussion

In this chapter, we show that machine learning models can successfully be used to predict data instances that give rise to high expert disagreement. The main motivation for this prediction problem is the medical domain, where some patient cases result in significant differences in doctor diagnoses, and may benefit greatly from a medical second opinion. We show, both with a formal result and through extensive experiments, that Direct Uncertainty Prediction, which learns an uncertainty score directly from the raw patient features, performs significantly better than Uncertainty via Classification. Future work might look at transferring these techniques to different data modalities, and extending the applications to machine learning data denoising. In the next chapter, we will use these results to inform the effective combination of human experts and machine learning predictions.

| | Prediction Type | Absolute Val | 2-Wasserstein |
|---|---|---|---|
| UVC | Histogram-E2E-Var | 0.650 | 0.644 |
| UVC | Histogram-E2E-Disagree | 0.645 | 0.633 |
| UVC | Histogram-PC-Var | 0.638 | 0.639 |
| UVC | Histogram-PC-Disagree | 0.660 | 0.655 |
| DUP | Variance-PR | 0.671 | 0.664 |
| DUP | Variance-PRC | 0.665 | 0.658 |
| DUP | Disagree-P | **0.682** | **0.670** |
| DUP | Disagree-PC | 0.680 | 0.669 |
| | 2 Doctors | 0.460 | 0.448 |
| | 3 Doctors | 0.585 | 0.576 |
| | 4 Doctors | 0.641 | 0.634 |
| | 5 Doctors | 0.676 | 0.670 |
| | 6 Doctors | 0.728 | 0.712 |

| | Prediction Type | Binary Disagree |
|---|---|---|
| UVC | Histogram-E2E-Var | 0.643 |
| UVC | Histogram-E2E-Disagree | 0.643 |
| UVC | Histogram-PC-Var | 0.619 |
| UVC | Histogram-PC-Disagree | 0.649 |
| DUP | Variance-PR | 0.660 |
| DUP | Variance-PRC | 0.656 |
| DUP | Disagree-P | **0.676** |
| DUP | Disagree-PC | 0.675 |
| | 2 Doctors | 0.455 |
| | 3 Doctors | 0.580 |
| | 4 Doctors | 0.644 |
| | 5 Doctors | 0.675 |
| | 6 Doctors | 0.718 |

Table 7.5: **Ranking evaluation of models uncertainty scores using Spearman's rank correlation.** In the top set of rows, we compare the ranking induced by the model uncertainty scores to the (ground truth) ranking induced by the Wasserstein distance between the empirical grade histogram and the adjudicated grade. We use three different metrics for evaluating Wasserstein distance: absolute value distance, 2-Wasserstein and Binary agree/disagree (more details in Appendix E.6.) Again, we see that *all* DUPs outperform *all* baselines on *all* metrics. The second set of rows provides another way to interpret these results. We subsample $n$ doctors to create a new subsampled empirical grade histogram, and compare the ranking induced by the Wasserstein distance between this and the adjudicated grade to the ground truth ranking. We can thus say that the average DUP ranking corresponds to having 5 doctor grades, and the average UVC ranking corresponds to 4 doctor grades.

CHAPTER 8

# THE ALGORITHMIC AUTOMATION PROBLEM: TRIAGE, PREDICTION AND HUMAN EFFORT

On a variety of high-stakes tasks, machine learning algorithms are on the threshold of doing what human experts do with such high fidelity that we are contemplating using their predictions as a substitute for human output. For example, convolutional neural networks are close to diagnosing pneumonia from chest X-rays better than radiologists can [255, 315]; examples like these underpin much of the widespread discussion of algorithmic automation in these tasks.

In assessing the potential for algorithms, however, the community has implicitly equated the specific task of prediction with the general task of automation. We argue here that this implicit correspondence misses key aspects of the automation problem; a broader conceptualization of automation can lead directly to concrete benefits in some of the key application areas where this process is unfolding.

We start from the premise that automation is more than just the replacement of human effort on a task; it is also the meta-decision of which instances of the task to automate. And it is here that algorithms distinguish themselves from earlier technology used for automation, because they can actively take part in this decision of what to automate. But as currently constructed, they are not set up to help with this second part of the problem. The automation problem, then, should involve an algorithm that on any given instance both (i) produces a prediction output; and (ii) additionally also produces a triage judgment of its effectiveness relative to the human effort it would replace on that instance.

Viewed in this light, machine learning algorithms as currently constructed only solve the first problem; they do not pose or solve the second problem. In effect, currently when we contemplate automation using these algorithms, we are implicitly assuming that we will automate all instances or none. In this chapter, we argue that when algorithms are built to solve both problems – prediction and triage – overall performance is significantly higher. In fact, even on tasks where the algorithm significantly outperforms humans on average per instance, the optimal solution is to automate only a fraction of the instances and to optimally divide up the available human effort on the remaining ones. And correspondingly, even on tasks where an algorithm does not beat human experts, the optimal solution may still be to automate a subset of instances.

Now is the right time to ask these questions because the AI community is on the verge of translating some of its most successful algorithms into clinical practice. Notably, an influential line of work showed how a well-constructed convolutional net trained on gold-standard consensus labels for diagnosing diabetic retinopathy (DR) outperforms ophthalmologists in aggregate, and these results have led to considerable optimism about the role of algorithms in this setting [315]. But the community's discussion around these prospects has focused on the algorithms' per-instance prediction performance without considering the problem of recognizing which instances to automate.

Using largely the same data, we build this additional, crucial component and find that, even in this context where an algorithm outperforms human experts in the aggregate, the optimal level of triage is not full automation. Instead significantly more accuracy can be had by triaging a fraction of the instances to the algorithm and leaving the remaining fraction to the human experts. Specifically,

full automation reduces the error rate from roughly 5.5% by human doctors to 4% with an algorithm solving every instance; automation with optimal triage, though, reduces the error further to roughly 3.5% – effectively adding a significant further fraction to the gains that were realized by algorithmically automating the task in the first place.

This gain occurs for two reasons: first the algorithm's high average performance hides significant heterogeneity in performance. For example, on roughly 40% of the instances the algorithm has *zero* errors. By the same token, on a small set of instances, the algorithm makes far more errors than average and these instances can be assigned to humans. Second, when the algorithm automates a fraction of the cases, that frees up human effort; reallocating that effort to the remaining cases can achieve further gains. In principle these gains could come from a single doctor spending additional time on the instance, or from multiple doctors looking at it; in our case, the available data allows us to explicitly quantify the gains arising from the second of these effects, due to the fact that we have multiple doctor judgments on each instance.

These results empirically demonstrate the importance of the triage component for the automation problem. We show that the gains we demonstrate are unlikely to have fully tapped the potential gains to be had through algorithmic triage: this neglected component deserves the kind of sustained effort from the machine learning community that the prediction component has received to date. In fact, given the disparity in efforts on these two problems, it is possible that the highest return to improving automation performance is through solving triage rather than further improving prediction.

## 8.1 General Framework

In a typical application where we consider using algorithmic effort in place of human effort, the goal is to *label* a set of instances of the problem with a positive or a negative label. For example, in a medical diagnosis setting, we may have a set of medical images, and the goal is to label each with a binary yes/no diagnosis. Let $x$ be an instance of the labeling problem, and let $a(x)$ be its *ground truth* value. For our purposes (as in the example of a binary diagnosis), we will think of this ground truth value as taking a value of either 0 or 1, with $a(x) = 0$ corresponding to a negative label and $a(x) = 1$ corresponding to a positive label.

How do we approach this problem algorithmically? Given a set $U$ of instances, we could train an algorithm to produce a numerical estimate $m(x) \in [0, 1]$ with the goal of minimizing a loss function $\sum_{x \in U} L(a(x), m(x))$, where $L(\cdot, \cdot)$ increases with the distance between its two inputs. For notational convenience, we will write $g(x)$ for $L(a(x), m(x))$, the algorithmic error on instance $x$. The $m(x)$ values are then converted into (binary) predictions, and we can evaluate the resulting error relative to ground truth. As a concrete example, one option is to threshold the $m(x)$ to produce a 0 or 1 value, and evaluate agreement with $a(x)$.

When a social planner considers the prospect of introducing algorithms into an existing task, we often imagine the question to be the following. The planner currently has human effort being devoted to instances of the task; for an instance $x \in U$, we can imagine that there is a human output $h(x)$, resulting in a loss $f(x) = L(a(x), h(x))$. The question of whether to automate the task could then be viewed as a comparison between $\sum_{x \in U} g(x)$ and $\sum_{x \in U} f(x)$ — the loss from algorithmic effort relative to the loss from human effort.

**Allocating Human Effort** In order to think about the activity of automation in a richer sense, it is useful to start from the realization that even in the absence of algorithms, the social planner is implicitly working with a larger space of choices than the simple picture above suggests. In particular, they have some available budget of total human effort, and they do not need to allocate it uniformly across instances: for an instance $x$, the planner can allocate $k$ units of human effort for different possible values of $k$. There are multiple possible interpretations for the meaning of $k$; for example, in the case of diagnosis we could think of $k$ as corresponding to the number of distinct doctors who look at the instance, or alternately to the total amount of time spent collectively by doctors on the instance. Thus, our functions $h$ and $f$ should more properly be written as two-variable functions that take an instance $x$ and a level of effort $k$: we say that $h(x, k)$ is the label provided as a result of $k$ units of human effort on instance $x$, and $f(x, k) = L(a(x), h(x, k))$ is the resulting loss that we would like to minimize.

Note that as functions of effort $k$, it may be that $f(x, k)$ and $f(x', k)$ are quite different for different instances $x$ and $x'$. For example, instance $x$ may be much harder than instance $x'$, and hence $f(x, k)$ will be much larger than $f(x', k)$; similarly, instance $x$ might not exhibit as much marginal benefit from additional effort as instance $x'$, and hence the growth of $f(x, k)$ over increasing values of $k$ might be much flatter than the growth of $f(x', k)$. The social planner might well not have precise quantitative estimates for values like $f(x, k)$, but implicitly they are seeking to allocate human effort across the set of instances $U$ so as to minimize the total loss incurred. And indeed, a number of basic protocols — such as asking for second opinions — can be viewed as increasing the amount of effort spent on instances where there might be benefits for error reduction.

### 8.1.1 Automation involving Algorithms and Humans

When algorithms are introduced, the social planner has several new considerations to take into account. First, the full automation problem should be viewed more broadly than just a binary comparison of human and algorithmic performance; it can involve decisions about the allocation of both human and algorithmic effort. The introduction of the algorithm need not be all-or-nothing: we can choose to apply it to some instances in a way that replaces human effort, thereby potentially freeing up this effort to be used on other instances. The average overall comparison might even hide instances where the algorithm much more significantly under- (or out-) performs the human. Second, decisions about the allocation of human effort depend on the function $f(x, k)$, which can be challenging to reason about. Algorithms can potentially provide assistance in estimating these quantities $f(x, k)$ to help even in the allocation of human effort.

The general problem can therefore be viewed as follows. We would like to select a subset $S$ of instances on which no human effort will be used (only the algorithm), and we will then optimally allocate human effort on the remaining instances $T = U - S$. Suppose that we have a budget $B$ on the total number of units of human effort that we can allocate, and we decide to allocate $k_x$ units of effort to each instance $x \in T$. On such an instance $x$, we incur a loss of $f(x, k_x)$, using our notation above; and on the instances $x \in S$ we incur a loss of $g(x)$ from the algorithmic prediction.

We thus have the following optimization problem.

$$\text{Min} \quad \sum_{x \in S} g(x) + \sum_{x \in T} f(x, k_x) \tag{8.1}$$

$$\text{subject to} \quad \sum_{x \in T} k_x \leq B \tag{8.2}$$

$$S \cup T = U; \;\; S \cap T = \phi \tag{8.3}$$

Our earlier discussions about algorithms and humans in isolation are special cases of this optimization problem: *full automation*, when the algorithm substitutes completely for human effort, is the case in which $S = U$; and the social planner's problem in the absence of an algorithm — which still involves decisions about the effort variables $k_x$ — is the case in which $T = U$. Intermediate solutions can be viewed as performing a kind of *triage*, a term we use here in a general sense for a process in which some instances go purely to an algorithm and others receive human attention.

By deliberately adopting a very general formulation, we can also get a clearer picture of the kinds of information we would need in order to perform automation more effectively. Specifically,

(i) In addition to making algorithmic predictions $m(x)$, the automation problem benefits from more accurate estimates of the algorithm's instance-specific error rate $g(x)$.

(ii) The allocation of human effort benefits from better models of human error rate, including error as a function of effort spent $f(x, k)$. As noted above, we can use an algorithm to help in estimating this human error rate.

(iii) Given estimates for the functions $f$ and $g$, we can obtain further performance improvements purely through better allocations of human effort in

the optimization problem (8.1).

We note that the notion of human error involves an additional set of complex design choices, which is how humans decide to make use of algorithmic assistance on the instances (in the set $T$) where they spend effort. In particular, if we imagine that algorithmic predictions are available on the instances in $T$, then the humans involved in the decision on $x \in T$ may have the ability to incorporate the algorithmic prediction $m(x)$ into their overall output $h(x, k_x)$, and this will have an effect on the error rate $f(x, k_x)$. In general, of course, it will be difficult to model a priori how this synthesis will work, although it is a very interesting question; we show that our results for the automation problem do not require assumptions about this aspect of the process, but we explore this question later in the chapter.

### 8.1.2 Heuristics for Automation

If we think of the social planner as the entity tasked with solving the automation problem in (8.1), they are now faced with a set of considerations: not simply the binary question of whether to use human or algorithmic effort, but instead how to divide the instances between those (in $S$) that will be fully automated and those (in $T$) that will involve human effort, and how to estimate the error rates $g(x)$ and $f(x, k)$ so as to solve the allocation problem effectively.

We will show that significant performance gains can be achieved over both algorithmic and human effort even if we use only very simple heuristics for the different components of the allocation problem. Moreover, through a stronger

benchmark based on ground truth, we will also show that much stronger gains are in principle achievable with improved approaches to the components.

We can describe the simplest level of heuristics in terms of sub-problems (i), (ii), and (iii) from earlier in this section. The simplest heuristic for (i) is to use the functional form of the variance, $m(x)(1 - m(x))$ as a measure of the algorithm's uncertainty in its prediction on $x$. A comparably natural predictor does not exist for (ii). We therefore design new algorithmic predictors to estimate the values of both (i) and (ii), and use these to guide the allocation of algorithmic and human effort. We show that using separate predictors in this way also strengthens the performance gains relative to the simpler heuristic based on $m(x)(1 - m(x))$, although even this basic heuristic yields improvements over full automation.

Given these predictors for (i) and (ii), what does this suggest about simple strategies for approximating (iii), the allocation of human effort? First, we could restrict attention to solutions in which each instance in $T$ receives the same amount of effort. Thus, if there are $N$ total instances in $U$, we could choose a real number $\alpha \in [0, 1]$, perform full automation on a set $S$ of $\alpha N$ instances, and divide the $B$ units of human effort evenly across the remaining $\beta = 1 - \alpha$ fraction of the instances. This means that each instance in the set receiving human effort gets $B/\beta N$ units of effort. For simplicity, let us write $B = cN$, so that the human effort per instance in this remaining set is $c/\beta$. With this allocation of effort, the resulting loss is $\sum_{x \in S} g(x) + \sum_{x \in T} f(x, c/\beta)$.

This restriction on the set of possible solutions suggests the following heuristic. Consider any partition of the instances into $S$ and $T$, and suppose we use the algorithm on all the instances. Then we can write the resulting loss in the

following convenient way: $\sum_{x \in S} g(x) + \sum_{x \in T} g(x)$. Subtracting from the loss that results when we assign $c/\beta$ units of human effort to each instance in $T$, we see that the difference is $\sum_{x \in T} [f(x, c/\beta) - g(x)]$.

Thus, for a given value of $\alpha$ (specifying the fraction of instances that we wish to assign to the algorithm), it is sufficient to rank all instances $x \in U$ by $\tau_\alpha(x) = f(x, c/\beta) - g(x)$, and then choose the $\alpha N$ instances with the largest values of $\tau_\alpha$ to put in the set $S$ that we give to the algorithm. We can thus think of $\tau_\alpha(x)$ as the *triage score* of instance $x$, since it tells us the effect of algorithmic triage relative to human effort on the expected error.

### 8.1.3 Overview of Results

We put these ideas together in the context of a widely studied medical application, concerned with the diagnosis of diabetic retinopathy, detailed in the next section. We rank instances by their triage score, using simple forms for the algorithmic loss $g(x)$ and human loss $f(x, c/\beta)$, and we then search over possible values of $\alpha$, evaluating the performance at each. We find that there is range of values of $\alpha$, and a way of choosing $\alpha N$ instances to give to the algorithm, so that the resulting performance exceeds either of the binary choices of fully assigning the instances to the algorithm or to human effort.

As a scoping exercise, to see how strong the possible gains from our automation approach might be, we consider what would happen if we ranked instances by a triage score derived from a ground-truth estimate of the individual human error on each instance. Such a benchmark indicates the power of the optimization

framework if we are able to get better approximations to the key quantities of interest — the functions $f$ and $g$. We find large performance gains from this benchmark, and we also explore some stronger methods to work on closing the gap between our simple heuristics and this ideal.

**Different Costs for Error.** In many settings, a social planner may associate higher costs to errors committed by automated methods relative to errors committed by humans — for example, there may be concern about the difficulty in identifying and correcting errors through automation, or the end users of the results may have a preference for human output. It is natural, therefore, to consider a version of the optimization problem in which the objective (8.1) has an additional parameter $\lambda$ specifying the relative cost of error between algorithms and humans. This new objective function is

$$\lambda \sum_{x \in S} g(x) + \sum_{x \in T} f(x, k_x). \tag{8.4}$$

One might suppose that as $\lambda$ grows large, the social planner would tend to favor purely human effort, given the relative cost of errors from automation. And indeed, the basic comparison that is typically made between $\sum_{x \in U} g(x)$ (for full automation) and $\sum_{x \in U} f(x, k_x)$ (for purely human effort) would suggest that this should be the case, since eventually $\lambda$ will exceed the ratio between these two quantities. But our more detailed framework makes clear that these aggregate measures of performance can obscure large levels of variability in difficulty across instances. And what we find in our application is that it is possible for the algorithm to identify a large set of instances $S$ on which it makes *zero errors*. Thus, even with strong preferences for human effort over algorithmic effort, it may still be possible to find sizeable subsets of the problem that should nevertheless be

Figure 8.1: **Example fundus photographs**. Fundus photographs are images of the back of the eye, which can be used by an opthalmologist to diagnose patients with different kinds of eye diseases. One common such eye disease is *Diabetic Retinopathy*, where high blood sugar levels cause damage to blood vessels in the eye.

automated — a fact that is hidden by comparisons based purely on aggregate measures of performance.

## 8.2 Medical Preliminaries, Data and Experimental Setup

We first outline the details of the medical prediction problems, and describe the data and experimental setup used to design the automated decision making algorithm. As our primary goal is to study the interaction of this algorithm with human experts, we treat many of the underlying algorithmic components (e.g. a deep neural network model trained for predictions) as fixed, and focus on the different modes of interactions.

The main setting for our study is the use of *fundus photographs*, large images of the back of the eye, to automatically detect *Diabetic Retinopathy* (DR). Diabetic Retinopathy is an eye disease caused by high blood sugar levels damaging blood vessels in the retina. It can develop in anyone with diabetes (type 1 or type 2), and despite being treatable if caught early enough, it remains a leading cause of blindness [6].

A patient's fundus photograph is graded on a five class scale to indicate the presence and severity of DR. Grade 1 corresponds to no DR, 2 to mild (nonproliferative) DR, 3 to moderate (nonproliferative) DR, 4 to severe (nonproliferative) DR and 5 to proliferative DR. An important clinical threshold is at grade 3, with grades 3 and above called *referable* DR, requiring immediate specialist attention, [1]. Figure 8.1 shows some example fundus photos.

### 8.2.1 Data

The data used for designing the algorithm consists of these fundus photographs, with each photograph having multiple DR grades. These grades are assigned by individual doctors independently looking at the fundus photograph and deciding what DR classification the image should get. There are important distinctions between the data used for training the algorithm, and the data used for evaluation. The training dataset is much larger in size (as a key component is a large deep neural network) and hence each image is more sparsely labelled – typically with one to three DR grades. The evaluation dataset is much smaller and more extensively annotated. It is described in detail below in Section 8.2.3.

In the mechanics of training our classifier, it will be useful to view DR diagnosis as a 5-class classification, using the 5-point grading scheme. However, when we consider the problem of triage and automation at a higher level, we will treat the task as a binary classification problem into images that are referable or non-referable.

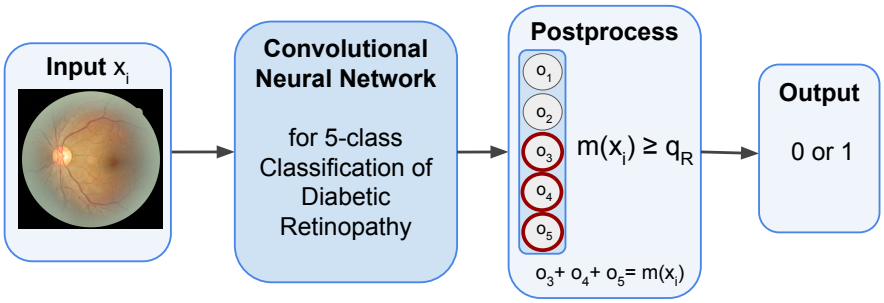



Figure 8.2: **Diagram of Algorithm for Diagnosing Diabetic Retinopathy (DR).** The algorithm takes as input a fundus photograph, which, with doctor grades as targets, is used to train a convolutional neural network to perform 5 class classification of DR. For evaluation on an image $i$ the output values of the convolutional neural network on grades $\geq 3$, $o_3, o_4, o_5$, are summed to give $m(x_i)$, the total output mass on a referable diagnosis. $m(x_i)$ is then thresholded with $q_R$ – the threshold for a referable diagnosis. This binary decision is output by the algorithm.

### 8.2.2 A Decision Making Algorithm for Diabetic Retinopathy

Similar to prior work [102], we first use the training dataset to train a convolutional neural network to classify each image. Specifically, the CNN outputs a distribution over the 5 different DR grades for each fundus photograph, with the empirical distribution of the individual doctor grades for that image as the target.

The question of whether the patient has referable DR (with a grade of at least 3), and hence needs specialist attention, is one of the most important clinical decisions. The outputs of the trained convolutional neural network form the basis of an algorithm to make this decision. First, we compute the predicted probability of referable DR by summing the model's output mass on DR grades $\geq 3$. For each image $x_i$, this gives a predicted referable DR probability of $m(x_i)$. Next, we rank the images according to the $m(x_i)$ values, and pick a threshold $q_R$.

Images $x_i$ with $m(x_i) \geq q_R$ are labelled as referable DR by the algorithm, and the others as non-referable.

The choice of the threshold $q_R$ is made so that the total number of cases marked as referable by the algorithm matches the total number of cases marked as referable when aggregating the human doctor grades. This ensures that the effort, resources, and expense needed to act upon the algorithmic decisions match the current (feasible) effort resulting from the human decision making process. This is discussed in further detail in Section 8.2.4

The result of this process is an algorithm taking as input a patient's fundus photograph, and outputting a binary 0/1 decision on whether the patient has non-referable/referable Diabetic Retinopathy. We illustrate the components of the DR algorithm in Figure 8.2. The full details of our algorithm development setup can be found in Appendix Section F.1.

### 8.2.3 Evaluation

We evaluate our decision-making algorithm on a special, gold-standard *adjudicated* dataset [167]. This dataset is much smaller than our training data, but is meticulously labelled. For every fundus photograph in the dataset, there are many individual doctor grades, and also a single *adjudicated* grade, given after multiple doctors discuss the appropriate diagnosis for the image. This adjudicated grade acts as a proxy for the ground truth condition, and we use it to evaluate both the individual human doctors and the decision making algorithm. In Appendix Section F.5 we carry out an additional evaluation of the methods on

a different dataset, which exhibits the same results.

### 8.2.4 Aggregation and Thresholding

During evaluation and the triage process, we often have multiple (binary) grades per image. These grades might correspond to multiple different human doctors individually diagnosing the image, or the algorithm's binary decision along with human doctor grades. In all of these cases, for evaluation, we must typically aggregate these multiple grades into a concrete decision – a single summary binary grade. To do so, we compute the mean grade and threshold by a value $R$. If the mean is greater than $R$, this corresponds to a decision of 1 (referable); otherwise the decision is 0 (non-referable).

The choice of the threshold $R$ also affects the choice of $q_R$ which is used for the algorithm's decision. To compute $q_R$, we first aggregate the multiple doctor grades per image into a single grade by computing their mean and thresholding with $R$. This gives us the total number of patients marked as referable by the human doctors, and we pick $q_R$ so that the algorithm matches this number.

In the main text, we give results for $R = 0.5$, which corresponds to the *majority vote* of the multiple grades for an image. In the Appendix, we include results for $R = 0.3, 0.4$, which support the same conclusions.

## 8.3 The Triage Problem and Human Effort Reallocation

The performance of human experts and algorithmic decisions are typically summarized and compared via a single number, such as average error, F1 score, or AUC. Seeing the algorithm outperform human experts according to these metrics might suggest the hypothesis that the algorithm uniformly outperforms human experts on any given instance.

What we find instead, however, is significant diversity across instances in the performance of humans and algorithms: for natural definitions of human and algorithmic error probability (formalized below), there are instances in which human effort has lower error probability than the algorithm, and instances in which the algorithm has lower error probability than human effort. Moreover, this diversity is partially *predictable*: we can identify with non-trivial accuracy those instances on which one entity or the other will do better. This diversity and its predictability is an important component of the automation framework, since it makes it possible to divide instances between algorithms and humans so that each party is working on those instances that are best suited to it.

We first study this performance diversity, and then move on to the problem of allocating effort between humans and algorithms across instances.

### 8.3.1 Per Instance Error Diversity of Humans and Algorithms

In order to look at the differences in performance between humans and algorithms on an instance-by-instance level, we want to define, for each instance $x_i$

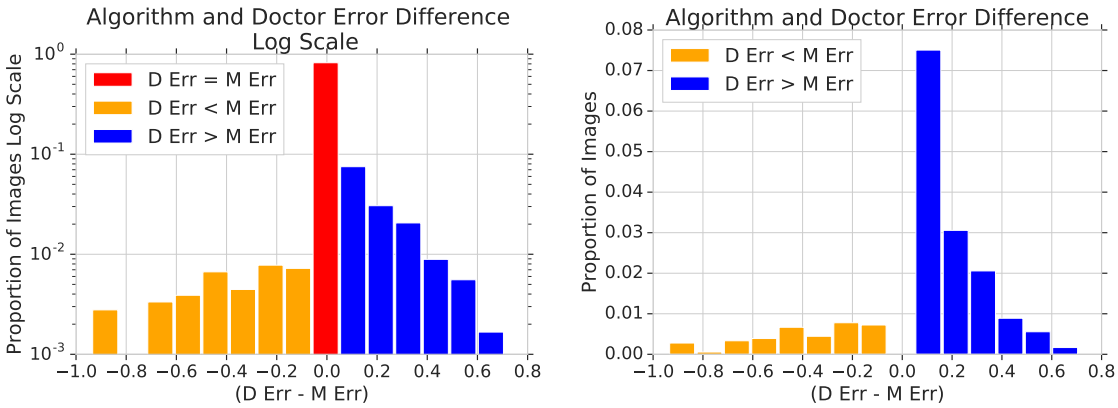



Figure 8.3: **Histogram plot of** $\Pr[H_i] - \Pr[M_i]$ **for instances** $i$ **in the adjudicated evaluation dataset.** We show a histogram of probability of human doctor error minus probability of model error over the examples in the adjudicated dataset. The orange bars correspond to examples where the human expert has a lower probability of error than the algorithm, the red where the probability of error is approximately equal, and the blue where the algorithm's probability of error is lower than the human expert's. The left pane is a log plot, and the right is standard scaling (pictured without the red bar.) While the algorithm clearly has lower error probability than the human in more cases, there is a nontrivial mass ( 5%) where the human experts have lower error probability than the model.

in the adjudicated dataset, an error probability $\Pr[H_i]$ for the doctors (human experts) and an error probability $\Pr[M_i]$ for the algorithm.

The quantity $\Pr[H_i]$ is straightforward to compute on the adjudicated dataset: for an instance $x_i$, suppose that $n_i$ doctors evaluate it, assigning it binary non-referable/referable grades $h^{(1)}(x_i), ..., h^{(n_i)}(x_i)$. Let $a(x_i)$ be the binary adjudicated grade for $x_i$. Then we can define

$$\Pr[H_i] = \frac{\sum_{j=1}^{n_i} |h^{(j)}(x_i) - a(x_i)|}{n_i}.$$

That is, $\Pr[H_i]$ is the average disagreement of doctors with the adjudicated grade.

Computing $\Pr[M_i]$ is a little more complicated. Recall that for an instance $i$, the convolutional neural network model in the algorithm outputs a value $m(x_i)$ between $[0, 1]$ that is then thresholded to give a binary decision. A naive estimate

of the error probability could therefore be $m(x_i)$ if the instance is *not* referable, and $1 - m(x_i)$ if the instance *is* referable. Unfortunately, deep neural networks are well-known to be poorly calibrated [103], and this naive approximation is both poorly calibrated and at a different scaling to the human doctors. This is not a concern for the algorithm's binary decision, since only the rank-ordering of the $m(x_i)$ values matter for this, but it poses a challenge for producing a probability that can serve as $\Pr[M_i]$.

**Determining Algorithm Error Probabilities**

To overcome this issue, we develop a simple method to calibrate the convolutional neural network's output. Recall that the neural network outputs a value $m(x_i) \in [0, 1]$ for each image $x_i$ – i.e. it induces a ranking over the images $x_i$, which is used to determine the algorithmic decision. We evaluate this induced ranking directly by asking:

*Suppose we produced a (random) number $R$ of referable instances by sampling a random doctor for each instance, what is the probability that $x_i$ is among the top $R$ instances in the induced ranking?*

We define $\tilde{p}(x_i)$ as the probability that the prediction algorithm declares $x_i$ to be referable. We can then define the error probability, $\Pr[M_i]$, as $\tilde{p}(x_i)$ if the adjudicated grade $a(x_i)$ is referable, and $1 - \tilde{p}(x_i)$ if $a(x_i)$ is non-referable. In Appendix Section F.2, we provide specific details of the implementation.

**Results on Performance Diversity**

We can now use the estimate of $\Pr[M_i]$ and $\Pr[H_i]$ to study the variation in human expert and algorithmic error across different instances. Specifically, we plot a histogram of values of $\Pr[H_i] - \Pr[M_i]$ across all the adjudicated image instances.

The result is shown in Figure 8.3. We see that while there are more images where $\Pr[M_i] < \Pr[H_i]$, there is a non-trivial fraction of images with $\Pr[M_i] > \Pr[H_i]$. In the subsequent sections, we analyze different ways of *predicting* these differences as a way to perform triage, and demonstrate the resulting gains.

### 8.3.2 Performing Triage and Reallocating Human Effort

In formulating the basic problem of automation, we considered two baselines for performance. The first is *full automation*, in which the overall loss is $\sum_{x_i \in U} g(x_i)$. The second is equal coverage of all instances by human effort: if we have a budget of $B = cN$ units of effort for $N$ instances, then we allocate $c$ units of human effort to each, resulting in a loss of $\sum_{x_i \in U} f(x_i, c)$. Our goal here is to show that by allocating human and algorithmic effort more effectively according to optimization problem (8.1) from Section 8.1, we can improve on both of these baselines.

Recall the basic heuristic from Section 8.1: for an arbitrary $\alpha \in [0, 1]$, we compute a *triage score* $\tau_\alpha(x_i)$ for each instance $x_i$; we assign the first $\alpha N$ to the set $S$ to be handled by the algorithm, and we allocate equal amounts of human effort to the remaining set $T$ of $(1 - \alpha)N$ instances. Note that $\alpha = 1$ corresponds

to the full automation baseline, while $\alpha = 0$ corresponds to equal coverage of all instances by human effort. We will see, however, that stronger performance can be achieved for intermediate values of $\alpha$.

We begin with two ways of computing the triage score. The first follows the basic strategy from Section 8.1, where we train two algorithmic predictors to estimate (i) the algorithm's error probability, $\Pr[M_i]$ and (ii) the human error probability $\Pr[H_i]$. Specifically, we train two auxillary neural networks, one to predict $\Pr[H_i]$ and one to predict $\Pr[M_i]$. To predict $\Pr[H_i]$, we build off of the work of [**?**] on direct prediction of doctor disagreement: we label each example with a 0 if there is agreement amongst the doctor grades, and 1 otherwise, and train a small neural network to predict these agreement labels from the image embedding. A similar setup is employed for predicting $\Pr[M_i]$, where the binary label now corresponds to whether the output of the diagnostic 5-class convolutional neural network agrees with the doctor grades – i.e. does the model make an error on that image. The full details of this process are described in Appendix F.2.

The second method of computing a triage score establishes an "ideal" benchmark on the potential power of the optimization problem (8.1) using aspects of ground truth, sorting the instances by the true value of $\Pr[H_i] - \Pr[M_i]$, since this divides the instances between humans and algorithms based on the relative strength of each party on the respective instances.

In both cases, we determine the performance of the human effort using the average of a corresponding number of randomly sampled doctor grades from the data. This allows us to demonstrate improvements without any assumptions

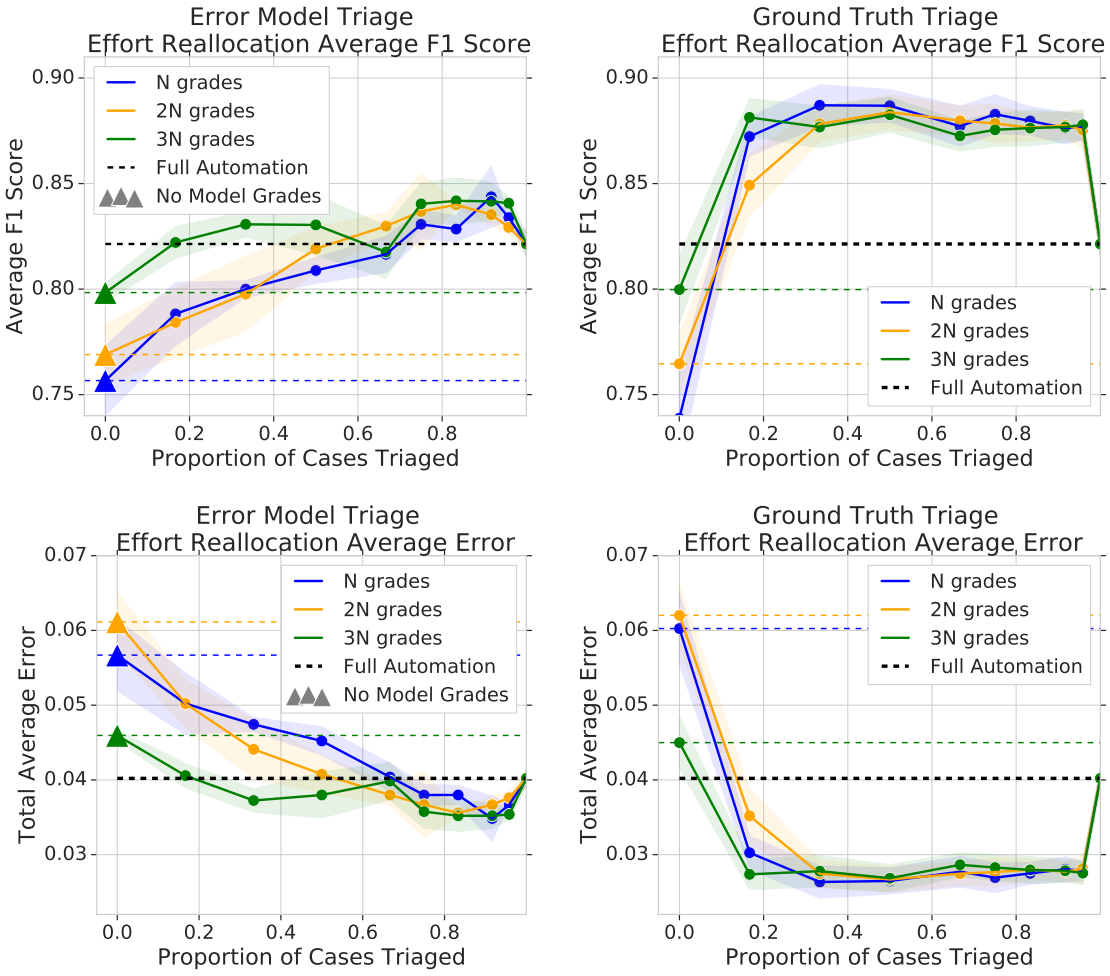



Figure 8.4: **Combing algorithmic and human effort by triaging outperforms full automation and the equal coverage human baseline. Left column: triage by the difference between the predicted values of** $\Pr[H_i]$ **and** $\Pr[M_i]$**. Right column: triage by ground truth** $\Pr[H_i] - \Pr[M_i]$ We order the images by their triage scores (predicted $\Pr[H_i] - \Pr[M_i]$ for the left column and ground truth $\Pr[H_i] - \Pr[M_i]$ on the right), and automate an $\alpha$ fraction of them. The remaining $(1-\alpha)N$ images have the human doctor budget ($N, 2N, 3N$ grades) allocated amongst them, according to the equal coverage protocol. This is described in further detail in Appendix Section F.3. The black dotted line is the performance of full automation, and the coloured dotted lines the performance of equal coverage for the different total number of doctor grades available. We see that triaging and combining algorithmic and human effort performs better than all of these baselines. Triaging by ground truth (right column) gives significant gains, and suggests that better triage prediction is a crucial problem that merits further study. In Appendix Section F.4, we also include results when the remaining $(1-\alpha)N$ cases have the algorithm grade available, along with the reallocated human effort. The qualitative conclusions are identical.

on how the doctors might use information from the algorithmic predictions on these instances. It is also reasonable, however, to imagine a scenario in which the algorithmic predictions are still freely available even on the instances that we assign to the human doctors, and to consider simple models for how the doctor grades might be combined with these algorithmic predictions. We consider this case in the Appendix, which supports the same conclusions.

**Triage Results**

The results for these two triage scores, as we vary $\alpha$, are shown in Figure 8.4. The figure depicts both the average error (bottom row), as well as the F1 score (top row), which accounts for imbalances between the number of referable and non-referable instances. The left column corresponds to using the difference between the predicted values of $\Pr[H_i]$ and $\Pr[M_i]$ as a triage score, while the right column corresponds to using the true value $\Pr[H_i] - \Pr[M_i]$ to perform triage. In both triage schemes, we observe that the best performance comes for $0 < \alpha < 1$, beating both the full automation protocol (dotted black line) and equal coverage of all instances by human effort (coloured dotted lines).

While combining algorithmic and human effort in both of these ways leads to performance gains, we see that the ground truth triage ordering performs significantly better than triaging by the predicted error probability. This suggests that learning better triage predictors might have an even greater impact on overall deployed performance than continuous slight improvements to diagnostic accuracy.

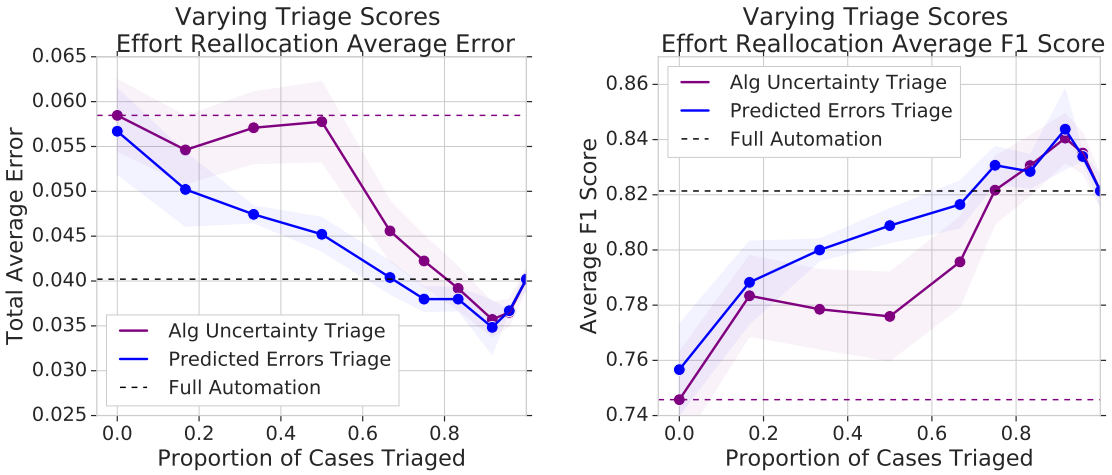



Figure 8.5: **Even triaging by algorithm uncertainty leads to gains over pure algorithmic and pure human performance.** Instead of the separate error prediction algorithms, we triage by the simple algorithm uncertainty: $m(x)(1 - m(x))$, which acts as a proxy for algorithm error probability (and no explicit modelling of human error probability.) The same qualitative conclusions hold with this simple triage score also (purple line), although larger gains are achieved with the separate error prediction algorithms (blue line). These results are for $N$ doctor grades, the same conclusions hold for $2N, 3N$ grades.

### The Simplest Heuristic: Algorithmic Uncertainty

In the previous section, we saw the results of training two separate algorithmic predictors to estimate the values of $\Pr[H_i]$ and $\Pr[M_i]$, and using the difference between these predicted values as a triage score. An even simpler triage score is given by only using the algorithm's uncertainty, $m(x)(1 - m(x))$. In Figure 8.5, we show that even this triage score, available 'for free' from the algorithmic predictor, improves upon pure automation and pure human effort, although larger gains are available through using the two algorithmic error predictors. These results reiterate the rich possibilities for gains from algorithmic triage.

### 8.3.3 Differential Costs and Zero-Error Subsets

Finally, we recall a further consideration from the framework in Section 8.1: suppose the social planner views errors made by algorithms as more costly than errors made by humans, resulting in an objective function of the form in (8.4), $\lambda \sum_{x \in S} g(x) + \sum_{x \in T} f(x, k_x)$. As $\lambda$ becomes large, what does this imply about the use of algorithmic predictions?

We find in our application that it is possible to identify large subsets of the data on which the algorithm achieves *zero error*. Such a fact can easily be hidden by considering only aggregate measures of algorithmic performance, and it implies that even when $\lambda$ is large, there may still be an important role for algorithms in automation.

To quantify this effect, we order the instances by a triage score as in our earlier analyses. We then look at the average error of the algorithmic predictions on the first $\alpha$ fraction of images: for $\alpha$ varying between 0 and 1, we plot

$$\frac{M_{err}(\alpha N)}{N}$$

where $M_{err}(\alpha N)$ is the number of errors made by the model on the first $\alpha N$ instances.

#### Results

Figure 8.6 left pane shows the results of plotting this quantity. We triage the cases both by our prediction of $\Pr[H_i] - \Pr[M_i]$ from the two error prediction algorithms as well as the simple algorithm uncertainty term, $m(x)(1 - m(x))$. We evaluate the

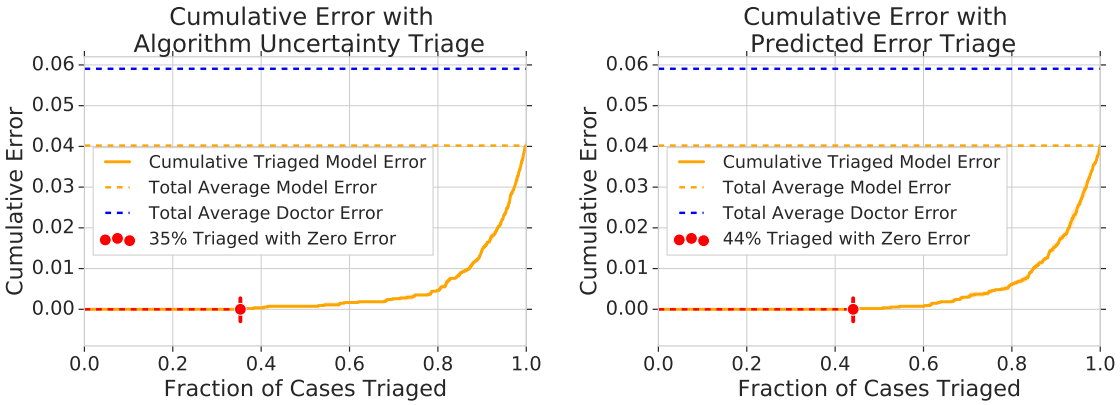



Figure 8.6: **Triage identifies large subsets of the data with zero error.** We plot the average cumulative error $\frac{M_{err}(\alpha N)}{N}$, where $M_{err}(\alpha N)$ is the number of errors made by the algorithm on the first $\alpha$ fraction of the $N$ images when triaged. We observe that triaging even by the simple uncertainty measure, $m(x_i)(1 - m(x_i))$ (left plot), can identify a 35% fraction of data where the algorithm makes zero errors. Using the separate error prediction model from Section 8.3.2, we can improve on this, identifying 44% of the data where the algorithm has zero errors. The plot is averaged over three repetitions (so each repeat identified at least 35%, 44% of the data respectively.)

average error of the algorithmic predictions on the first $\alpha$ fraction of images, over three repetitions of training the diagnostic neural network component of the algorithm. We see that even using the simple $m9x)(1 - m(x))$ as a triage score, we can identify a zero-error subset that is 35% the size of the entire dataset. Similar to Section 8.3.2, further improvements are shown by predicting the value of $\Pr[H_i] - \Pr[M_i]$. The right pane of Figure 8.6 shows this result, where we can identify a zero-error subset of size 44%, again averaged over three repetitions.

## 8.4 Related Work

With the successes of machine learning and particularly deep learning methodologies in modalities such as imaging, there have been numerous works comparing algorithmic performance to human performance in medical tasks, albeit in

frameworks that implicitly interpret automation as success in prediction. In this prediction setting, the general comparison is between the case in which only the algorithm is used and the case in which only human effort is used; such comparisons have been done for chest x-rays [255], for Alzheimer's detection from PET scans [62], and for the setting we consider here based on diabetic retinopathy diagnosis from fundus photographs (and OCT scans) [57, 102]. The recent survey paper by Topol [315] references several additional studies of this kind. A few papers have begun to look at fixed modes of interaction with humans, including processes in which algorithmic outputs are reviewed by physicians [33, 59, 193], as well as fixed combinations of physician and algorithmic judgments, as seen in Chapter 7.

## 8.5 Discussion

This chapter has presented a framework for analyzing automation by algorithms. Rather than treating the introduction of algorithms in an all-or-nothing fashion, we show that stronger performance can be obtained if algorithms are used both (i) for prediction on instances of the problem, and (ii) for providing triage judgments about which instances should be handled algorithmically and which should be handled by human effort. This broader formulation of the automation question highlights the importance of accurately estimating the propensity of both humans and algorithms to make errors on a per-instance basis, and the use of these estimates in an optimization framework for allocating effort efficiently. Analysis of an application in diabetic retinopathy diagnosis shows that this framework can lead to performance gains even for well-studied problems in AI

applications in medicine.

Through the analysis of benchmarks for stronger performance, we also highlight how stronger predictions of per-instance error has the potential to yield still better performance. Our findings thus demonstrate how further study of algorithmic triage and its role in allocating human and computational effort has the potential to yield substantial benefits for the task of automation.

# Part V

# Conclusion and Future Directions

CHAPTER 9

# CONCLUSION AND FUTURE DIRECTIONS

With the dramatic increase in the size and complexity of data across many specialized domains, and the continuing breakthroughs in central machine learning problems, there are many exciting opportunities for applications of machine learning in high-stakes domains such as medicine. However, these opportunities also present new challenges, such as gaining insights into the system beyond performance metrics, designing efficient learning algorithms and enabling effective collaboration between AI systems and human experts (with better evaluation of these capabilities).

In this thesis we have presented some of the research results taking steps to address these challenges. We started by developing quantitative techniques for giving insights into the hidden representations of neural network models, which shed light on many fundamental design components such as the training process and generalization. These insights went on to inform simpler and more flexible methods for efficient learning, across both few-shot learning and transfer learning. Finally, we looked at approaches to combine trained AI systems with human experts, formulating and tackling new prediction problems along the way and demonstrating that effective combinations could outperform either entity alone.

While these are important steps in surmounting the aforementioned challenges, there remain many rich open directions for future exploration. Considering the results presented in Part II, there is significant scope to build more techniques to give different kinds of insights into deep learning systems. As a

concrete example, we might consider using Partial Least Squares [347] or Generalized Canonical Correlation [313] as a foundation for representation analysis. There are also interesting open questions in combining representational analysis methods with interpretability techniques [231] to gain further understanding of the distributed nature of representations.

Similarly, building the results of both Part II and Part III we might consider understanding and unifying the diverse set of new *self-supervision* and *semi-supervised* learning methods developed by the community, and overviewed in Chapter 2. Such techniques could also dramatically help reduce the dependence on the need for costly labels in specialized applications. In Part III, we focused on transfer learning and few-shot learning as two approaches to reduce label dependence. An interesting regime to further explore is *medium-shot* learning, where there are more data instances than in few-shot learning, but perhaps not enough for full-fledged finetuning like in transfer learning. Understanding which algorithms might be most effective in this setting is an underexplored direction.

Finally, there are many rich directions to explore in enabling better collaboration between human experts and AI systems. Part IV presented a simple predictive problem and a resulting (effective) way to do a single-step combination of AI systems and human experts. There are many followup questions from even this first set of results. How faithfully can AI systems model human experts (and their potential errors)? Can we use techniques like few-shot learning to model individual human behaviours with less data? What are the evaluation metrics for AI systems that might best indicate their ability to work well with human experts on predictive tasks? Are there *multistep* interactions or techniques

for feedback incorporation (by either entity) that can be modelled and deployed?

We hope that the work presented in this thesis provides a foundation of both results and questions that will help further progress on these central challenges and enable the full potential of the design and deployment of machine learning systems.

APPENDIX A

# CHAPTER 3 APPENDIX

## A.1 Mathematical details of CCA and SVCCA

**Canonical Correlation of $X, Y$** Finding maximal correlations between $X, Y$ can be expressed as finding $a, b$ to maximise:

$$\frac{a^T \Sigma_{XY} b}{\sqrt{a^T \Sigma_{XX} a} \sqrt{b^T \Sigma_{YY} b}}$$

where $\Sigma_{XX}, \Sigma_{XY}, \Sigma_{YX}, \Sigma_{YY}$ are the covariance and cross-covariance terms. By performing the change of basis $\tilde{\boldsymbol{x}}_1 = \Sigma_{xx}^{1/2} a$ and $\tilde{\boldsymbol{y}}_1 = \Sigma_{YY}^{1/2} b$ and using Cauchy-Schwarz we recover an eigenvalue problem:

$$\tilde{\boldsymbol{x}}_1 = \text{argmax} \left[ \frac{x^T \Sigma_{XX}^{-1/2} \Sigma_{XY} \Sigma_{YY}^{-1} \Sigma_{YX} \Sigma_{XX}^{-1/2} x}{||x||} \right] \quad (*)$$

**SVCCA** Given two subspaces $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_{m_1}\}, Y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_{m_2}\}$, SVCCA first performs a singular value decomposition on $X, Y$. This results in singular vectors $\{\boldsymbol{x'}_1, ..., \boldsymbol{x'}_{m_1}\}$ with associated singular values $\{\lambda_1, ..., \lambda_{m_1}\}$ (for $X$, and similarly for $Y$). Of these $m_1$ singular vectors, we keep the top $m_1'$ where $m_1'$ is the smallest value that $\sum_{i=1}^{m_1'} |\lambda_i| (\geq 0.99 \sum_{i=1}^{m_1} |\lambda_i|)$. That is, 99% of the variation of $X$ is explainable by the top $m_1'$ vectors. This helps remove directions/neurons that are constant zero, or noise with small magnitude.

Then, we apply Canonical Correlation Analysis (CCA) to the sets $\{\boldsymbol{x'}_1, ..., \boldsymbol{x'}_{m_1'}\}, \{\boldsymbol{y'}_1, ..., \boldsymbol{y'}_{m_2'}\}$ of top singular vectors.

CCA is a well established statistical method for understanding the similarity of two different sets of random variables – given our two sets of vectors $\{\boldsymbol{x'}_1, ..., \boldsymbol{x'}_{m'_1}\}, \{\boldsymbol{y'}_1, ..., \boldsymbol{y'}_{m'_2}\}$, we wish to find linear transformations, $W_X, W_Y$ that maximally correlate the subspaces. This can be reduced to an eigenvalue problem. Solving this results in linearly transformed subspaces $\tilde{X}, \tilde{Y}$ with directions $\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i$ that are maximally correlated with each other, and orthogonal to $\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{y}}_j, j < i$. We let $\rho_i = corr(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i)$. In summary, we have:

**SVCCA Summary**

1. Input: $X, Y$
2. Perform: SVD(X), SVD(Y). Output: $X' = UX, Y' = VY$
3. Perform CCA($X'$, $Y'$). Output: $\tilde{X} = W_X X'$, $\tilde{Y} = W_Y Y'$ and $corrs = \{\rho_1, \ldots \rho_{\min(m_1, m_2)}\}$

## A.2 Additional Proofs and Figures from Section 3.2.1

Proof of Orthonormal and Scaling Invariance of CCA:

We can see this using equation (*) as follows: suppose $U, V$ are orthonormal transforms applied to the sets $X, Y$. Then it follows that $\Sigma^a_{XX}$ becomes $U\Sigma^a_{XX}U^T$, for $a = \{1, -1, 1/2, -1/2\}$, and similarly for $Y$ and $V$. Also note $\Sigma_{XY}$ becomes $U\Sigma_{XY}V^T$. Equation (*) then becomes

$$\tilde{x}_1 = \text{argmax}\left[\frac{x^T U\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}\Sigma_{XX}^{-1/2}U^T x}{||x||}\right]$$

So if $\tilde{u}$ is a solution to equation (*), then $U\tilde{u}$ is a solution to the equation above, which results in the same correlation coefficients.

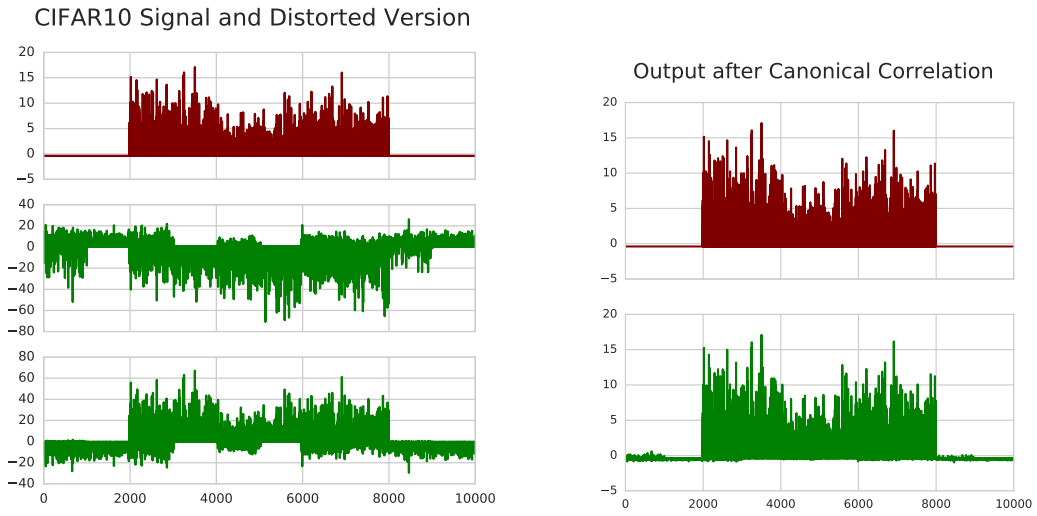



Figure App.1: This figure shows the ability of CCA to deal with orthogonal and scaling transforms. In the first pane, the maroon plot shows one of the highest activation neurons in the penultimate layer of a network trained on CIFAR-10, with the x-axis being (ordered) image ids and the y-axis being activation on that image. The green plots show two resulting distorted directions after this and two of the other top activation neurons are permuted, rotated and scaled. Pane two shows the result of applying CCA to the distorted directions and the original signal, which succeeds in recovering the original signal.

**The importance of SVD: how many directions matter?**

While CCA is excellent at identifying useful learned directions that correlate, independent of certain common transforms, it doesn't capture the full picture entirely. Consider the following setting: suppose we have subspaces $A, B, C$, with $A$ being 50 dimensions, $B$ being 200 dimensions, 50 of which are perfectly aligned with $A$ and the other 150 being noise, and C being 200 dimensions, 50 of which are aligned with $A$ (and $B$) and the other 150 being useful, but different directions.

Then looking at the canonical correlation coefficients of $(A, B)$ and $(A, C)$ will give the *same* result, both being 1 for 50 values and 0 for everything else. But these are two very different cases – the subspace $B$ is indeed well represented by the 50 directions that are aligned with $A$. But the subspace $C$ has 150 more useful directions.

This distinction becomes particularly important when aggregating canonical correlation coefficients as a measure of similarity, as used in analysing network learning dynamics. However, by first applying SVD to determine the number of directions needed to explain 99% of the observed variance, we can distinguish between pathological cases like the one above.

## A.3 Proof of Theorem 1

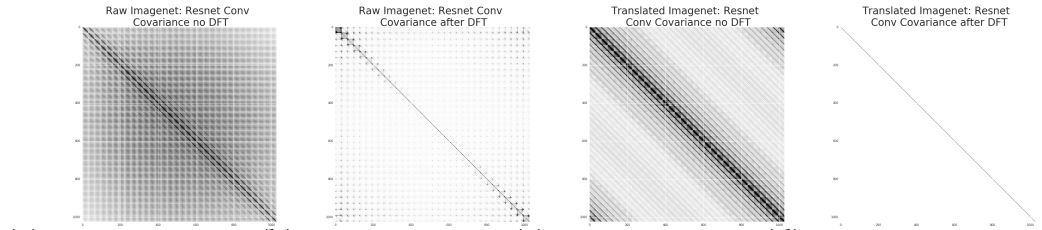



Figure App.2: This figure visualizes the covariance matrix of one of the channels of a resnet trained on Imagenet. Black correspond to large values and white to small values. *(a)* we compute the covariance without a translation invariant dataset and without first preprocessing the images by DFT. We see that the covariance matrix is dense. *(b)* We compute the covariance after applying DFT, but without augmenting the dataset with translations. Even without enforcing translation invariance, we see that the covariance in the DFT basis is approximately diagonal. *(c)* Same as (a), but the dataset is augmented to be fully translation invariant. The covariance in the pixel basis is still dense. *(d)* Same as (c), but with dataset augmented to be translation invariant. The covariance matrix is exactly diagonal for a translation invariant dataset in a DFT basis.

Here we provide the proofs for theorem 3, theorem 4, Theorem 2 and finally Theorem 1.

A preliminary note before we begin:

When we consider a (wlog) $n$ by $n$ channel $c$ of a convolutional layer, we assume it has shape

$$\begin{bmatrix} z_{0,0} & z_{1,2} & \dots & z_{0,n-1} \\ z_{1,0} & z_{2,2} & \dots & z_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n-1,0} & z_{n-1,1} & \dots & z_{n-1,n-1} \end{bmatrix}$$

When computing the covariance matrix however, we vectorize $c$ by stacking the columns under each other, and call the result $vec(c)$:

$$vec(c) = \begin{bmatrix} z_{0,0} \\ z_{1,0} \\ \vdots \\ z_{n-1,0} \\ z_{0,1} \\ \vdots \\ z_{n-1,n-1} \end{bmatrix} := \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \\ z_n \\ \vdots \\ z_{n^2-1} \end{bmatrix}$$

One useful identity when switching between these two notations is

$$vec(AcB) = (B^T \otimes A)vec(c)$$

where $A, B$ are matrices and $\otimes$ is the Kronecker product. A useful observation arising from this is:

**Theorem 7.** *The CCA vectors of $DFT(c_i), DFT(c_j)$ are the same (up to a rotation by F) as the CCA of $c_i, c_j$.*

*Proof:* From Section A.2 we know that unitary transforms only rotate the *CCA* directions. But while DFT pre and postmultiplies by $F, F^T$ – unitary matrices, we cannot directly apply this as the result is for unitary transforms on $vec(c_i)$. But, using the identity above, we see that $vec(DFT(c_i)) = vec(Fc_iF^T) = (F \otimes F)vec(c_i)$, which is unitary as $F$ is unitary. Applying the same identity to $c_j$, we can thus conclude that the DFT preserves CCA (up to rotations).

As Theorem 1 preprocesses the neurons with DFT, it is important to note that by the theorem above, we do not change the CCA vectors (except by a rotation).

### A.3.1 Proof of theorem 3

*Proof. Translation invariance is preserved* We show inductively that any translation invariant input to a convolutional channel results in a translation invariant output: Suppose the input to channel $c$, ($n$ by $n$) is translation invariant. It is sufficient to show that for inputs $X_i, X_j$ and $0 \leq a, b, \leq n-1$, $c(X_i) + (a, b) \mod n = c(X_j)$. But an $(a, b)$ shift in neuron coordinates in $c$ corresponds to a (height stride $\cdot\, a$, width stride $\cdot\, b$) shift in the input. And as $X$ is translation invariant, there is some $X_j = X_i +$ (height stride $\cdot\, a$, width stride $\cdot\, b$).

*$cov(c)$ is circulant:*

Let $X$ be (by proof above) a translation invariant input to a channel $c$ in some

convolution or pooling layer. The empirical covariance, $cov(c)$ is the $n^2$ by $n^2$ matrix computed by (assuming $c$ is centered)

$$\frac{1}{|X|}\sum_{X_i \in X} vec(c(X_i)) \cdot vec(c(X_i))^T$$

So, $cov(c)_{ij} = \frac{1}{|X|} z_i^T z_j = \frac{1}{|X|}\sum_{X_l \in X} z_i^T(X_l) z_j(X_l)$, i.e. the inner products of the neurons $i$ and $j$.

The indexes $i$ and $j$ refer to the neurons in their vectorized order in $vec(c)$. But in the matrix ordering of neurons in $c$, $i$ and $j$ correspond to some $(a_1, b_1)$ and $(a_2, b_2)$. If we applied a translation $(a, b)$, to both, we would get new neuron coordinates $(a_1 + a, b_1 + b), (a_2 + a, b_2 + b)$ (all coordinates $\mod n$) which would correspond to $i + an + b \mod n^2$ and $j + an + b \mod n^2$, by our stacking of columns and reindexing.

Let $\tau_{a,b}$ be the translation in inputs corresponding to an $(a, b)$ translation in $c$, i.e. $\tau_{a,b} = (\text{height stride} \cdot a, \text{width stride} \cdot b)$. Then clearly $z_{(a_1,b_1)}(X_i) = z_{(a_1+a,b_1+b)}(\tau_{(a,b)}(X_i))$, and similarly for $z_{(a_2,b_2)}$

It follows that $\frac{1}{|X|} z_{(a_1,b_1)}^T z_{(a_2,b_2)} = \frac{1}{|X|} z_{(a_1+b,b_1+b)}^T z_{(a_2+a,b_2+b)}$, or, with $vec(c)$ indexing

$$\frac{1}{|X|} z_i^T z_j = \frac{1}{|X|} z_{(i+an+b \mod n^2)}^T z_{(j+an+b \mod n^2)}$$

This gives us the circulant structure of $cov(c)$.

*$cov(c)$ is block circulant:* Let $z^{(i)}$ be the ith column of $c$, and $z^{(j)}$ the jth. In $vec(c)$, these correspond to $z_{(i-1)n}, \ldots z_{in-1}$ and $z_{(j-1)n}, \ldots z_{jn-1}$, and the $n$ by $n$ submatrix at those row and column indexes of $cov(vec(c))$ corresponds to the covariance of column $i$, $j$. But then we see that the covariance of columns $i + k$, $j + k$, corresponding

to the covariance of neurons $z_{(i-1)n+k\cdot n}, \ldots z_{in-1+k\cdot n}$, and $z_{(j-1)n+k\cdot n}, \ldots z_{jn-1+k\cdot n}$, which corresponds to the 2-d shift $(1,0)$, applied to every neuron. So by an identical argument to above, we see that for all $0 \leq k \leq n-1$

$$cov(z^{(i)}, z^{(j)}) = cov(z^{(i+k)}, z^{(j+k)})$$

In particular, $cov(vec(c))$ is block circulant. □

An example $cov(vec(c))$ with $c$ being 3 by 3 look like below:

$$\begin{bmatrix} A_0 & A_1 & A_2 \\ A_2 & A_0 & A_1 \\ A_1 & A_2 & A_0 \end{bmatrix}$$

where each $A_i$ is itself a circulant matrix.

### A.3.2 Proof of theorem 4

*Proof.* This is a standard result, following from expressing a circulant matrix $A$ in terms of its diagonal form , i.e. $A = V\Sigma V^T$ with the columns of $V$ being its eigenvectors. Noting that $V = F$, the DFT matrix, and that vectors of powers of $\omega_k = \exp(\frac{2\pi ik}{n})$, $\omega_j = \exp(\frac{2\pi ik}{n})$ are orthogonal gives the result. □

### A.3.3 Proof of Theorem 2

*Proof.* Starting with (a), we need to show that $cov(vec(DFT(c_i)), vec(DFT(c_i))$ is diagonal. But by the identity above, this becomes:

$$cov(vec(DFT(c_i)), vec(DFT(c_i)) = (F \otimes F)vec(c_i)vec(c_i)^T(F \otimes F)^*$$

By theorem 3, we see that

$$cov(vec(c_i)) = vec(c_i)vec(c_i)^T = \begin{bmatrix} A_0 & A_1 & \dots & A_{n-1} \\ A_{n-1} & A_0 & \dots & A_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ A_1 & A_2 & \dots & A_0 \end{bmatrix}$$

with each $A_i$ circulant.

And so $cov(vec(DFT(c_i)), vec(DFT(c_i))$ becomes

$$\begin{bmatrix} f_{00}F & f_{01}F & \dots & f_{0,n-1}F \\ f_{10}F & f_{11}F & \dots & f_{1,n-1}F \\ \vdots & \vdots & \ddots & \vdots \\ f_{n-1,0}F & f_{n-1,1}F & \dots & f_{n-1,n-1}F \end{bmatrix} \begin{bmatrix} A_0 & A_1 & \dots & A_{n-1} \\ A_{n-1} & A_0 & \dots & A_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ A_1 & A_2 & \dots & A_0 \end{bmatrix} \begin{bmatrix} f_{00}^*F^* & f_{10}^*F^* & \dots & f_{n-1,0}^*F^* \\ f_{01}^*F^* & f_{11}^*F^* & \dots & f_{n-1,1}^*F^* \\ \vdots & \vdots & \ddots & \vdots \\ f_{0,n-1}^*F^* & f_{1,n-1}^*F^* & \dots & f_{n-1,n-1}^*F^* \end{bmatrix}$$

From this, we see that the $sj$th entry has the form

$$\sum_{l=0}^{n-1}\left(\sum_{k=0}^{n-1} f_{sk}FA_{l-k}\right)f_{lj}^*F^* = \sum_{k,l} f_{sk}f_{lj}^*FA_{l-k}F^*$$

Letting $[FA_rF^*]$ denote the coefficient of the term $FA_rF^*$, we see that (addition being $\mod n$)

$$[FA_rF^*] = \sum_{k=0}^{n-1} f_{sk}f_{(k+r)j}^* = \sum_k e^{\frac{2\pi isk}{n}} \cdot e^{\frac{-2\pi ij(k+r)}{n}} = e^{\frac{-2\pi ijr}{n}} \sum_{k=0}^{n-1} e^{\frac{2\pi ik(s-j)}{n}} = e^{\frac{-2\pi ijr}{n}} \cdot \delta_{sj}$$

with the last step following by the fact that the sum of powers of non trivial roots of unity are 0.

In particular, we see that only the diagonal entries (of the $n$ by $n$ matrix of matrices) are non zero. The diagonal elements are linear combinations of terms of form $FA_rF^*$, and by theorem 4 these are diagonal. So the covariance of the DFT is diagonal as desired.

Part (b) follows almost identically to part (a), but by first noting that exactly by the proof of theorem 3, $cov(c_i, c_j)$ is also a circulant and block circulant matrix.

□

### A.3.4 Proof of Theorem 1

*Proof.* This Theorem now follows easily from the previous. Suppose we have a layer $l$, with channels $c_1, ..., c_k$. And let $vec(DFT(c_i))$ have directions $\tilde{\boldsymbol{z}}_0^{(i)}, \cdots \tilde{\boldsymbol{z}}_{n^2-1}^{(i)}$. By the previous theorem, we know that the covariance of all of these neurons only has non-zero terms $cov(\tilde{\boldsymbol{z}}_k^{(i)}, \tilde{\boldsymbol{z}}_k^{(j)}$.

So arranging the full covariance matrix to have row and column indexes being $\tilde{\boldsymbol{z}}_0^{(1)}, \tilde{\boldsymbol{z}}_0^{(1)}, \ldots \tilde{\boldsymbol{z}}_0^{(k)}, \tilde{\boldsymbol{z}}_1^{(1)} \ldots \tilde{\boldsymbol{z}}_{n^2}^{(k)}$ the nonzero terms all live in the $n^2$ $k$ by $k$ blocks down the diagonal of the matrix, proving the theorem. □

### A.3.5 Computational Gains

As the covariance matrix is block diagonal, our more efficient algorithm for computation is as follows: take the DFT of every channel ($n \log n$ due to FFT) and then compute covariances according to blocks: partition the $k^n$ directions into the $n^2$ $k$ by $k$ matrices that are non-zero, and compute the covariance, inverses and square roots along these.

A rough computational budget for the covariance is therefore $kn \log n + n^2 k^{2.5}$, while the naive computation would be of order $(kn^2)^{2.5}$, a polynomial difference. Furthermore, the DFT method also makes for easy parallelization as each of the $n^2$ blocks does not interact with any of the others.

## A.4 Per Layer Learning Dynamics Plots from Section 3.4.1

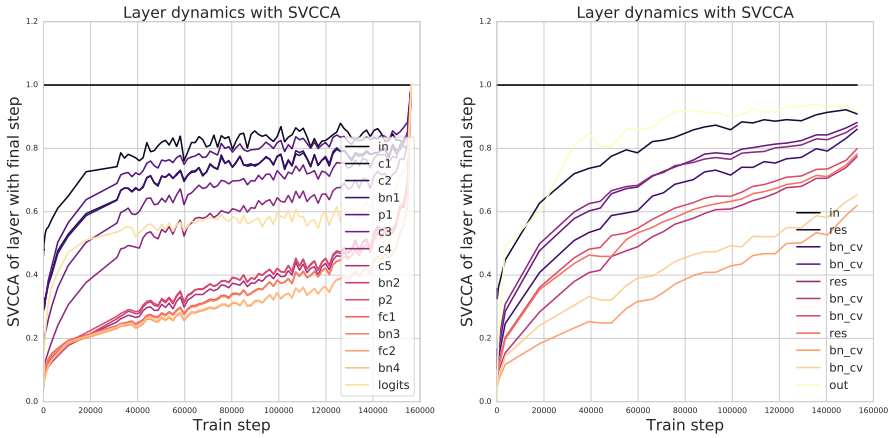



Figure App.3: Learning dynamics per layer plots for conv (left pane) and res (right pane) nets trained on CIFAR-10. Each line plots the SVCCA similarity of each layer with its final representation, as a function of training step, for both the conv (left pane) and res (right pane) nets. Note the bottom up convergence of different layers

## A.5 Additional Figure from Section 3.4.4

Figure App.4 compares the converged representations of two different initializations of the same convolutional network on CIFAR-10.

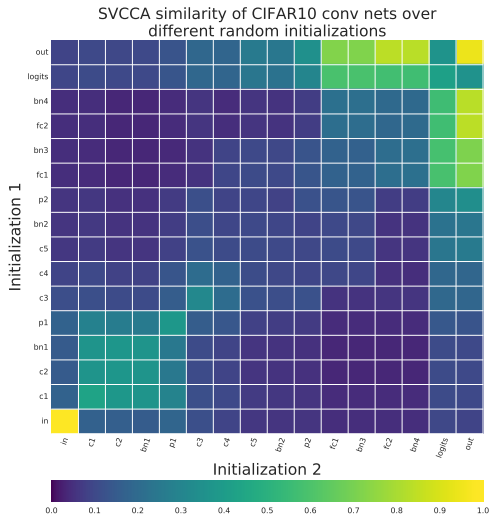



Figure App.4: Comparing the converged representations of two different initializations of the same convolutional architecture. The results support findings in [183], where initial and final layers are found to be similar, with middle layers differing in representation similarity.

## A.6 Experiment from Section 3.4.4

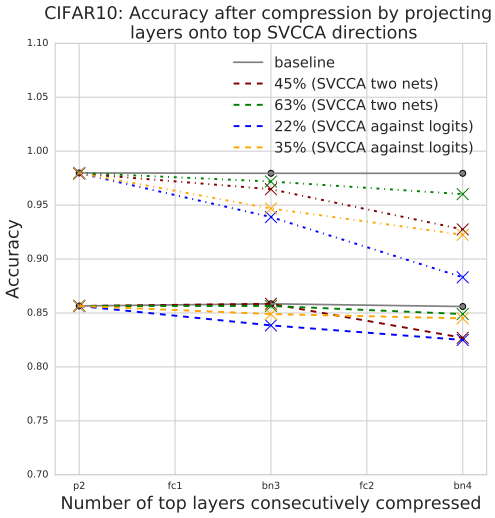



Figure App.5: Using SVCCA to perform model compression on the fully connected layers in a CIFAR-10 convnet. The two gray lines indicate the original train (top) and test (bottom) accuracy. The two sets of representations for SVCCA are obtained through 1) two different initialization and training of convnets on CIFAR-10 2) the layer activations and the activations of the logits. The latter provides better results, with the final five layers: pool1, fc1, bn3, fc2 and bn4 all being compressed to 0.35 of their original size.

APPENDIX B

# APPENDIX TO PWCCA AND GENERALIZATION

## B.0.1 Performance Plots for Models

We include the train/test curves for models trained in Figure 4.1. Comparing the curves to Figure 4.1, we can see that for all the models, there is a train time $t_0$ where performance is almost equivalent to final performance, but most CCA coefficients $\rho^{(i)}$ still haven't converged. This suggests that the vectors associated with these $\rho^{(i)}$ are noise in the representation, which is not necessary for doing well at the task.

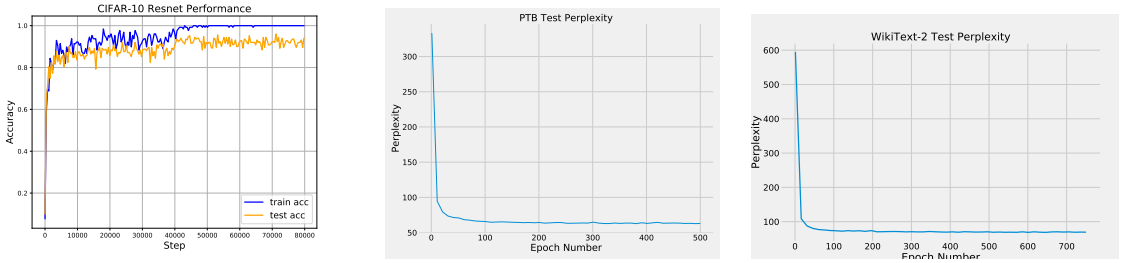



Figure App.1: **Performance convergence for CIFAR-10 CNNs, and PTB and WikiText-2 RNNs**.

## B.0.2 Additional reduction methods for CCA

**Bartlett's Test** Another potential method to reduce across CCA vectors of varying importnace is to estimate the number of important CCA vectors $k$, and perform an average over this. A statistical hypothesis test, proposed by Bartlett [22], and known as Bartlett's test, attempts to identify the number of statistically significant canonical correlations. Key to the test is the computation of Bartlett's

statistic:

$$T_k = -\left(n - k - \frac{1}{2}(a + b + 1) + \sum_{i=1}^{k} \frac{1}{(\rho^{(i)})^2}\right) \log\left(\prod_{i=k+1}^{c} (1 - (\rho^{(i)})^2\right)$$

where, in the same notation as previously, $n$ is the number of datapoints, and $a, b$ are the number of neurons in $L_1, L_2$, with $c = \min(a, b)$. The null hypothesis $H_0$ is that there are $k$ statistically significant canonical correlations with the remaining $\rho^{(i)}$ are generated randomly via a normal distribution [22]. Under the null, the distribution of $T_k$ becomes chi-squared with $(a - k)(b - k)$ degrees of freedom. We can then compute the value of $T_k$ and determine if $H_0$ satisfactorily explains the data.

However, the iterative nature of this metric makes it expensive to compute. We therefore focus on projection weighting in this work, and leave further exploration of Bartlett's test for a future study.

### B.0.3 Representation Dynamics in RNNs Through Sequence (Time) Steps

Here, we investigate the utility of CCA for analyzing representations of RNNs unrolled across sequence time steps. As a toy example of CCA's benefit in this case, we first initialize a linear vanilla RNN with a unitary recurrent matrix (such that it simply rotates the hidden representation on each timestep). We then use cosine distance, Euclidean distance, and CCA to compare the hidden representation at each timestep to the representation at the final timestep (Figure App.2a-c). While both cosine and Euclidean distance fail to realize the similarity between timesteps, CCA, because of its invariance to linear transforms, immediately

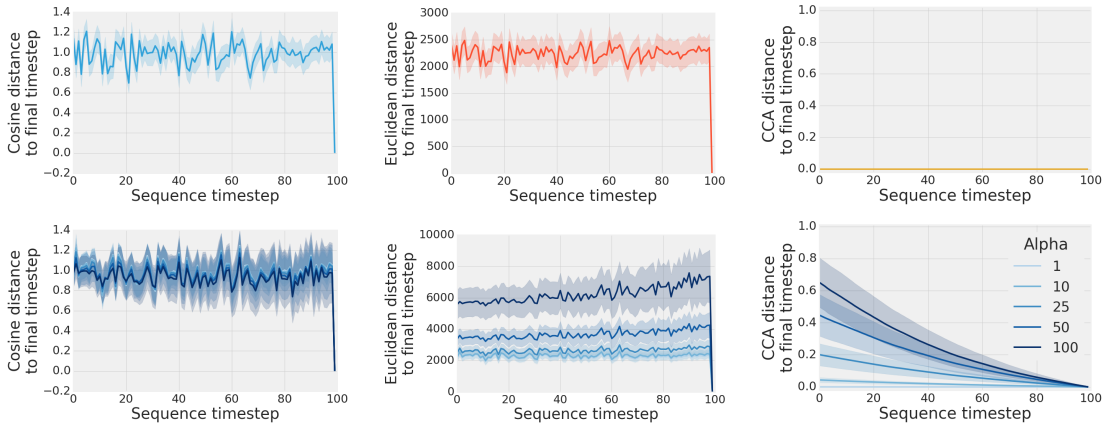



Figure App.2: **Toy RNN examples demonstrating that CCA is comparatively rotation invariant.** In a toy example, vanilla RNNs were initialized with a random rotation matrix and run 1000 times with a random starting hidden state and no inputs. Hidden states at each timepoint were compared to the final hidden state using cosine distance (**a**, **d**), Euclidean distance (**b**, **e**), and CCA (**c**, **f**). Due to its rotation invariance, CCA recognized all states as similar in both linear RNNs (**a-c**), and a blended linear/non-linear case (**d-f**; $h_{t+1} = W_{rot}h_t + \alpha \cdot \sigma(W_{rand}h_t) + b$, where $W_rot$ is a random rotation matrix, $W_{rand} \sim \mathcal{N}(0, I)$), while both cosine and Euclidean distance largely fail. Error bars represent mean ± std.

recognizes that the representations at all timesteps are linearly equivalent.

However, as linear networks are limited in their representational capabilities, we next examine a toy case of a network involving both a linear and non-linear component. We again initialize a simple RNN with the following update rule:

$$h_{t+1} = W_{rot}h_t + \alpha \cdot \sigma(W_{rand}h_t) + b$$

where $h_t$ is the hidden state at time $t$, $\sigma$ represents the sigmoid nonlinearity, $W_{rot}$ is a random rotation matrix, $W_{rand} \sim \mathcal{N}(0, I)$), and $\alpha$ is a scale factor between the linear and non-linear components. For values of $\alpha$ as high as 100 (suggesting that the nonlinear component has 100 times the magnitude of the linear component), we again find that, in contrast to CCA, cosine and Euclidean distance fail

to recognize the similarity between timesteps (Figure App.2d-f).

However, both of the above cases are toy examples. We next analyze the application of CCA to the more realistic situation of LSTM networks trained on PTB and WikiText-2. To do this, we unroll the RNN for 20 sequence steps, and collect the activations of each neuron in the hidden state over the appropriate sequence tokens for each of the 20 timesteps. More precisely, we can represent our output by a matrix $O$ with dimensions $(N, m)$ where $N$ is the number of neurons and $m$ is the total sequence length. Our per sequence step matrices would then be $O_0, ..., O_{19}$, with $O_j$ consisting of all the outputs corresponding to sequence tokens with index equal to $j$ modulo 20, and our matrix would have dimensions $(N, m/20)$. We can then compare $O_j$ to $O_{19}$ analogous with the comparison to the final timestep. We then apply CCA, Cosine and Euclidean distance as above. To our surprise, the hidden state varies significantly from sequence timestep to sequence timestep, Figure App.4.

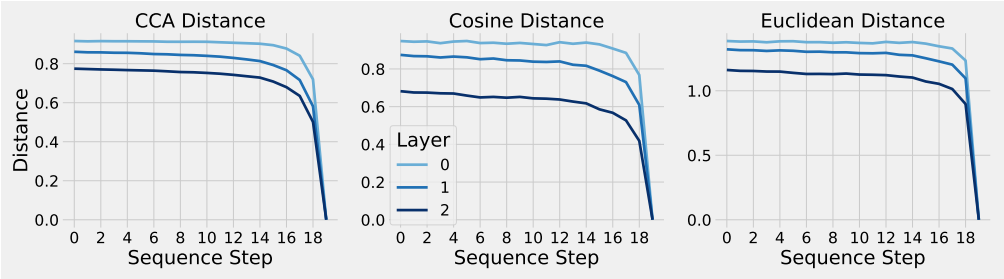



Figure App.3: **Hidden states are nonlinearly variable over sequence timesteps.** Using CCA (left), cosine distance (middle), and Euclidean distance (right), we measured the distance between representations at sequence timestep $t$ and the final sequence timestep $T$. Interestingly, even CCA failed to find similarity until late in the sequence, suggesting that the hidden state varies nonlinearly in the presence of unique inputs.

The above result demonstrates that the hidden state varies nonlinearly in the presence of unique inputs. However, this nonlinearity could be caused by the recurrent dynamics or novel inputs. To disambiguate these two cases, we asked how the hidden state changes when the same input is repeated. We therefore

repeat the same input for 20 timesteps, beginning the repetition after some percentage of previous steps containing unique inputs (e.g., 1%, 10%, ... through the $m$ input sequence tokens). When the repeating inputs were presented early in the sequence, CCA recognized that the hidden state was highly similar, while cosine and Euclidean distance remained insensitive to this similarity (Figure App.4, light blue lines). This result appears to suggest that the recurrent dynamics are approximately linear in nature.

However, when the same set of repeating inputs was presented late in the sequence (Figure App.4, dark blue lines), we found that the CCA distance increased markedly, suggesting that the nonlinearity of the recurrent dynamics depends not only on the (fixed) recurrent matrix, but also on the sequence history of the network.

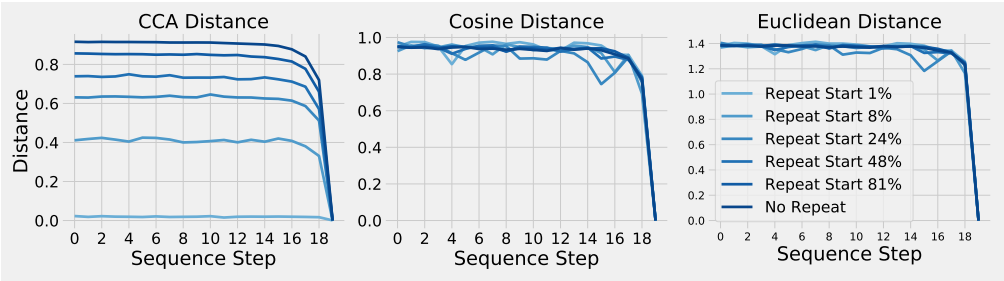



Figure App.4: **Hidden states vary linearly in the presence of repeated inputs.** To test whether the nonlinearity in the hidden state over sequence timesteps was due to input variability or recurrent dynamics, we measured the CCA distance (left), cosine distance (middle), and Euclidean distance (right) between sequence timestep $t$ and the final sequence timestep $T$ in the presence of repeating inputs. Interestingly, we found that when the repetition started after only a small set of unique inputs have been presented (light blue lines), CCA was able to recognize that the hidden states at each sequence timestep were highly similar. However, after many unique inputs had been delivered, the CCA distance markedly increased, suggesting that the nonlinearity of the recurrent dynamics is dependent on the network's history.

### B.0.4 Experimental details

**CIFAR-10 ConvNet Architecture:** The convolutional networks trained on CIFAR-10 were identical to those used in [221]. All CIFAR-10 networks were trained for 100 epochs using the Adam optimizer with default parameters, unless otherwise specified (learning rate: 0.001, beta1: 0.9, beta2: 0.999). Default layer sizes were: 64, 64, 128, 128, 128, 256, 256, 256, 512, 512, 512, with strides of 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, respectively. All kernels were 3x3 and a batch size of 32. Batch normalization layers were present after each convolutional layer. For the experiments in Section 3.2, all layers were scaled equally by a constant factor $\in$ 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0.

**RNN Experiments:** RNN experiments on PTB and WikiText2 followed the experimental setup in [210] and [211]. In particular, we used the open sourced model code[1] for training the word level Penn TreeBank and WikiText-2 LSTM models, (without finetuning or continuous cache pointer augmentation). All hyperparameters were left unmodified, so experiments can be reproduced by training LSTM models using the command to run *main.py*, and then applying CCA to the hidden states, via the open source implementation[2].

**Toy Experiments:** Generate $k$ vectors in $\mathbb{R}^{2000}$ of 'signal' (iid standard normal), for $k \in 20, 50, 70, 80, 100, 120, 140, 160, 180, 199$ and concatenate this $\mathbb{R}^{k \times 2000}$ matrix with a noise matrix: $\mathbb{R}^{(200-k)\times 2000} \sim \mathcal{N}(0, 0.1)$ to. (Note that the noise being lower magnitude than the signal is something that we see in typical neural networks –

[1] https://github.com/salesforce/awd-lstm-lm
[2] https://github.com/google/svcca/

work on network compression has showing that pruning low magnitude weights is an effective compression strategy.) Putting together gives matrix $X$, 200 (neurons) by 2000 (datapoints). Apply a randomly sampled orthonormal transform to the $k$ by 2000 subset of $X$ to get a new $k$ by 2000 matrix, and again add iid noise of dimensions $(200 - k)$ by 2000 to get matrix $Y$. Apply CCA based methods to detect similarity between $X, Y$. Of particular interest are cases $k << 200$ (low dim. signal in noise).

### B.0.5 Additional control experiments

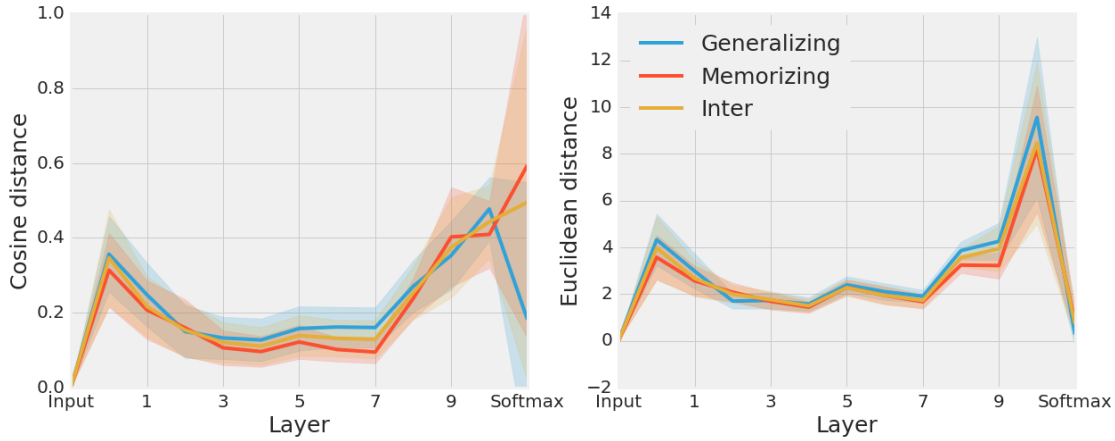



Figure App.5: **Cosine and Euclidean distance do not reveal the difference in converged solutions between groups of generalizing and memorizing networks.** Groups of 5 networks were trained on CIFAR-10 with either true labels (generalizing) or random labels (memorizing). The pairwise cosine (left) and eucldean (right) distance was then compared among generalizing networks, memorizing networks, and between generalizing and memorizing networks (inter) for each layer. While its invariance to linear transforms enabled CCA distance to reveal a difference between groups generalizing and memorizing networks in later layers (Figure 4.3), cosine and Euclidean distance fail to detect this difference. Error bars represent mean ± std distance across pairwise comparisons.

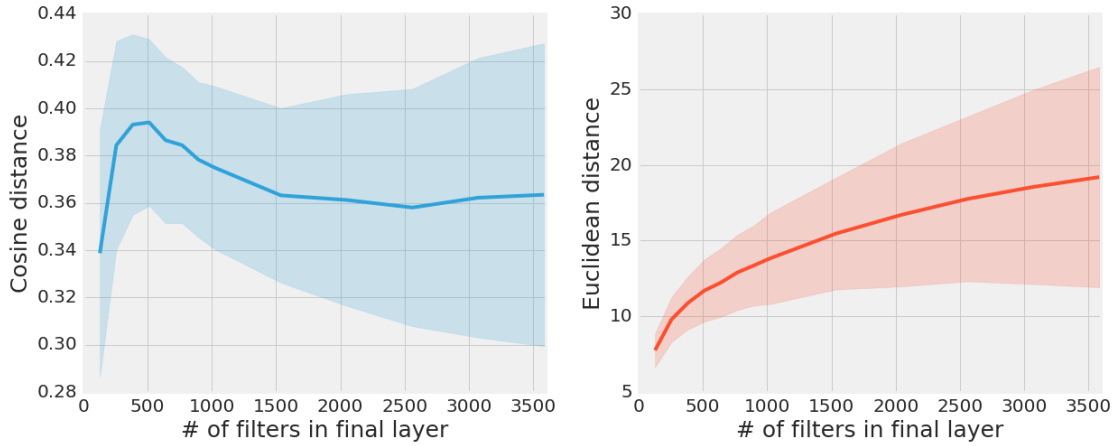



Figure App.6: **Cosine and Euclidean distance do not reveal the relationship between network size and similarity of converged solutions.** Groups of 5 networks with different random initializations were trained on CIFAR-10. Each group contained filter sizes of $\lambda[64, 64, 128, 128, 128, 256, 256, 256, 512, 512, 512]$ with $\lambda \in \{0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0\}$. Pairwise cosine (left) and Euclidean (right) distance was computed for each group of networks. While CCA distance revealed that larger networks converge to more similar solutions (Figure 4.4), cosine and Euclidean distance fail to find this relationship. Error bars represent mean ± std distance across pairwise comparisons.

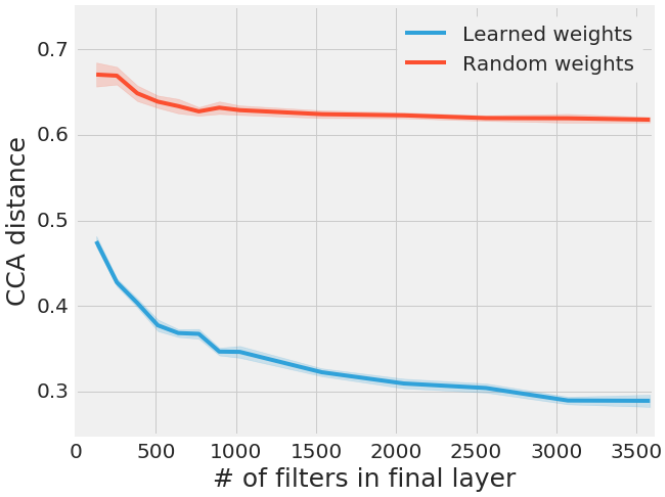



Figure App.7: **Relationship between network size and similarity of converged solutions is not present at initialization.** Activations at initialization (random weights) and after training (learned weights) were extracted from groups of 5 networks with different random initializations from CIFAR-10 data. Each group contained filter sizes of $\lambda[64, 64, 128, 128, 128, 256, 256, 256, 512, 512, 512]$ with $\lambda \in \{0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0\}$. While CCA distance decreases substantially for trained networks (from approximately 0.47 to 0.28), CCA distance only decreased moderately (from approximately 0.67 to 0.63) and plateaued past approximately 1000 filters. Error bars represent mean ± std distance across pairwise comparisons.

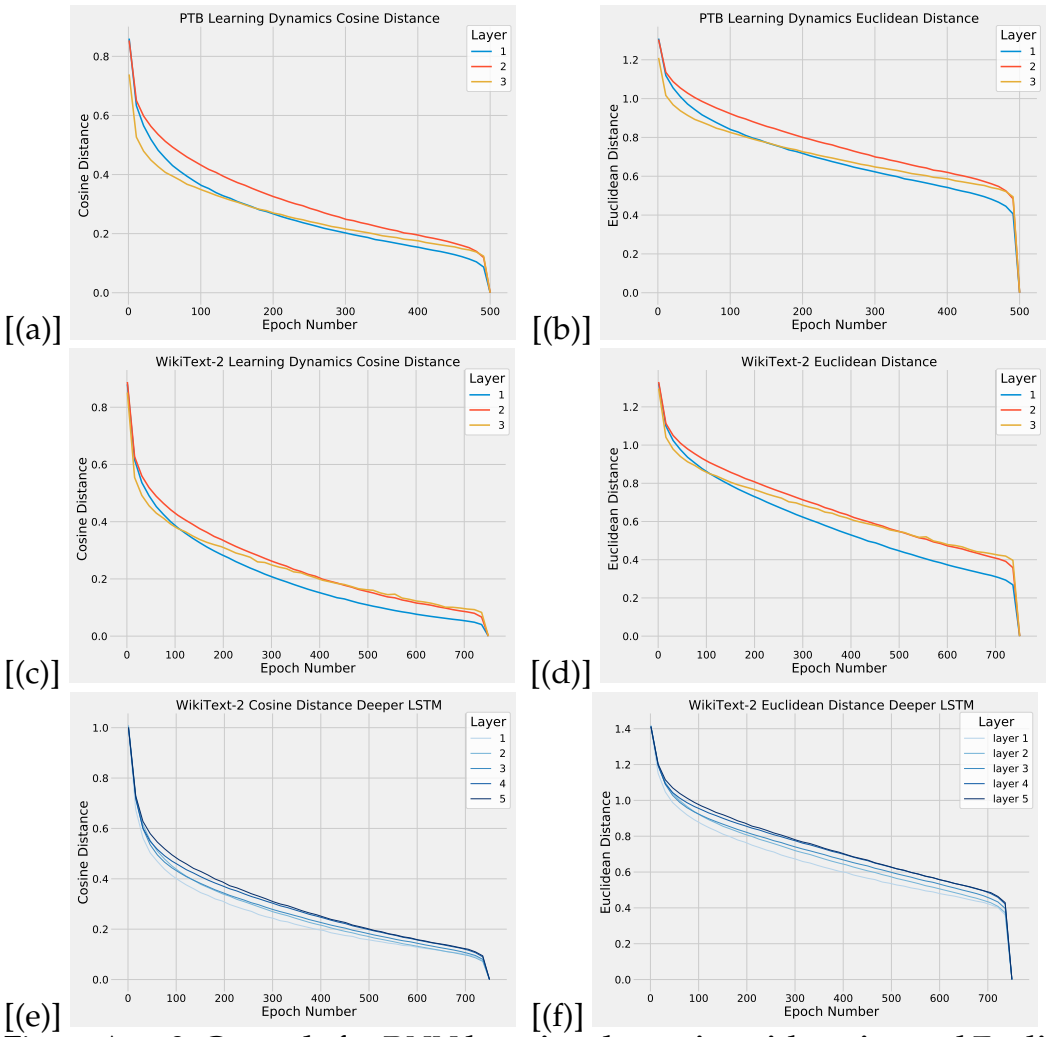



Figure App.8: **Controls for RNN learning dynamics with cosine and Euclidean distance** To test whether layers converge to their final representation over the course of training with a particular structure, we compared each layer's representation over the course of training to its final representation using cosine (**a**, **c**, **e**) and Euclidean distance (**b**, **d**, **f**). In shallow RNNs trained on PTB (**a-b**), and WikiText-2 (**c-d**), both cosine and Euclidean distance display properties of bottom-up convergence, albeit with substantially more noise than CCA (4.6). In deeper RNNs trained on WikiText-2, we observed a similar pattern (**e-f**).

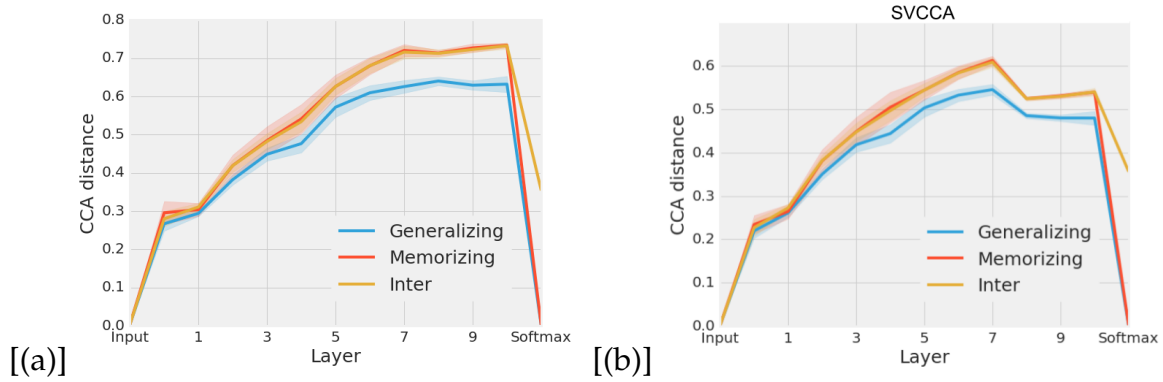



Figure App.9: **Unweighted CCA and SVCCA also finds that generalizing networks converge to more similar solutions than memorizing networks, but misses several key features.** While weighted CCA (Figure 4.3), unweighted CCA (**a**), and SVCCA (**b**) reveal the same broad pattern across generalizing and memorizing networks, unweighted CCA and SVCCA miss several key features. First, unweighted CCA misses the fact that generalizing networks become more similar to one another in the final two layers. Second, both unweighted CCA and SVCCA overestimate the distance between networks in early layers. Error bars represent mean ± std unweighted mean CCA and unweighted mean SVCCA distance across pairwise comparisons.

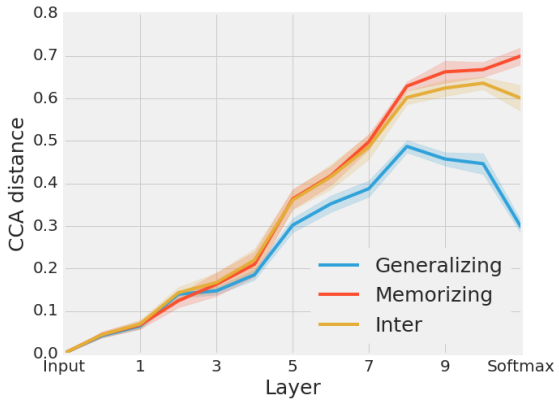



Figure App.10: **On test data, generalizing networks converge to similar solutions at the softmax, but memorizing networks do not.** Groups of 5 networks were trained on CIFAR-10 with either true labels (generalizing) or random labels (memorizing). The pairwise CCA distance was then compared within each group and between generalizing and memorizing networks (inter) for each layer, based on the test data. At the softmax, sets of generalizing networks converged to similar (though not identical) solutions, but memorizing networks did not, reflecting the diverse strategies used by memorizing networks to memorize the training data. Error bars represent mean ± std weighted mean CCA distance across pairwise comparisons.

APPENDIX C

# CHAPTER 5 APPENDIX

## C.1 Few-Shot Image Classification Datasets and Experimental Setups

We consider the few-shot learning paradigm for image classification to evaluate MAML and ANIL. We evaluate using two datasets often used for few-shot multiclass classification – the Omniglot dataset and the MiniImageNet dataset.

**Omniglot:** The Omniglot dataset consists of over 1600 different handwritten character classes from 23 alphabets. The dataset is split on a character-level, so that certain characters are in the training set, and others in the validation set. We consider the 20-way 1-shot and 20-way 5-shot tasks on this dataset, where at test time, we wish our classifier to discriminate between 20 randomly chosen character classes from the held-out set, given only 1 or 5 labelled example(s) from each class from this set of 20 testing classes respectively. The model architecture used is identical to that in the original MAML paper, namely: 4 modules with a 3 x 3 convolutions and 64 filters with a stride of 2, followed by batch normalization, and a ReLU nonlinearity. The Omniglot images are downsampled to 28 x 28, so the dimensionality of the last hidden layer is 64. The last layer is fed into a 20-way softmax. Our models are trained using a batch size of 16, 5 inner loop updates, and an inner learning rate of 0.1.

**MiniImageNet:** The MiniImagenet dataset was proposed by ravi2016optimization, and consists of 64 training classes, 12 validation classes, and 24 test classes. We consider the 5-way 1-shot and 5-way 5-shot tasks on this dataset, where the test-time task is to classify among 5 different randomly chosen validation classes, given only 1 and 5 labelled examples respectively. The model architecture is again identical to that in the original paper: 4 modules with a 3 x 3 convolutions and 32 filters, followed by batch normalization, ReLU nonlinearity, and 2 x 2 max pooling. Our models are trained using a batch size of 4. 5 inner loop update steps, and an inner learning rate of 0.01 are used. 10 inner gradient steps are used for evaluation at test time.

## C.2 Additional Details and Results: Freezing and Representational Similarity

In this section, we provide further experimental details and results from freezing and representational similarity experiments.

### C.2.1 Experimental Details

We concentrate on MiniImageNet for our experiments in Section 5.2.2, as it is more complex than Omniglot.

The model architecture used for our experiments is identical to that in the original paper: 4 modules with a $3 \times 3$ convolutions and 32 filters, followed by

batch normalization, ReLU nonlinearity, and $2 \times 2$ max pooling. Our models are trained using a batch size of 4, 5 inner loop update steps, and an inner learning rate of 0.01. 10 inner gradient steps are used for evaluation at test time. We train models 3 times with different random seeds. Models were trained for 30000 iterations.

### C.2.2 Details of Representational Similarity

CCA takes in as inputs $L_1 = \{z_1^{(1)}, z_2^{(1)}, ..., z_m^{(1)}\}$ and $L_2 = \{z_1^{(2)}, z_2^{(1)}, ..., z_n^{(2)}\}$, where $L_1, L_2$ are layers, and $z_i^{(j)}$ is a neuron activation vector: the vector of outputs of neuron $i$ (of layer $L_j$) over a set of inputs $X$. It then finds linear combinations of the neurons in $L_1$ and neurons in $L_2$ so that the resulting activation vectors are maximally correlated, which is summarized in the canonical correlation coefficient. Iteratively repeating this process gives a similarity score (in $[0, 1]$ with 1 identical and 0 completely different) between the representations of $L_1$ and $L_2$.

We apply this to compare corresponding layers of two networks, net1 and net2, where net1 and net2 might differ due to training step, training method (ANIL vs MAML) or the random seed. When comparing convolutional layers, as described in svccaurl, we perform the comparison over channels, flattening out over all of the spatial dimensions, and then taking the mean CCA coefficient. We average over three random repeats.

### C.2.3 Similarity Before and After Inner Loop with Euclidean Distance

In addition to assessing representational similarity with CCA/CKA, we also consider the simpler measure of Euclidean distance, capturing how much weights of the network change during the inner loop update (task-specific finetuning). We note that this experiment does not assess functional changes on inner loop updates as well as the CCA experiments do; however, they serve to provide useful intuition.

We plot the per-layer average Euclidean distance between the initialization $\theta$ and the finetuned weights $\theta_m^{(b)}$ across different tasks $T_b$, i.e.

$$\frac{1}{N}\sum_{b=1}^{N} \|(\theta_l) - (\theta_l)_m^{(b)}\|$$

across different layers $l$, for MiniImageNet in Figure App.1. We observe that very quickly after the start of training, all layers except for the last layer have small Euclidean distance difference before and after finetuning, suggesting significant feature reuse. (Note that this is despite the fact that these layers have more parameters than the final layer.)

### C.2.4 CCA Similarity Across Random Seeds

The experiment in Section 5.2.2 compared representational similarity of $L_1$ and $L_2$ at different points in training (before/after inner loop adaptation) but corresponding to the same random seed. To complete the picture, it is useful to study whether representational similarity across *different* random seeds is also mostly

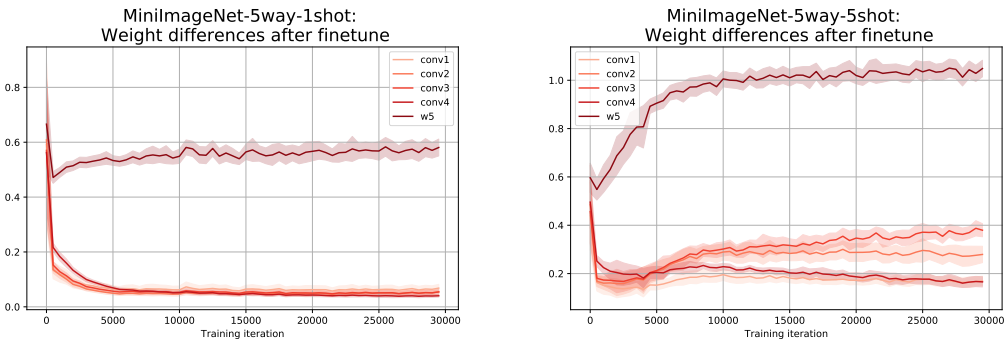



Figure App.1: **Euclidean distance before and after finetuning for MiniImageNet**. We compute the average (across tasks) Euclidean distance between the weights before and after inner loop adaptation, separately for different layers. We observe that all layers except for the final layer show very little difference before and after inner loop adaptation, suggesting significant feature reuse.

unaffected by the inner loop adaptation. This motivates four natural comparisons: assume layer $L_1$ is from the first seed, and layer $L_2$ is from the second seed. Then we can compute the representational similarity between ($L_1$ pre, $L_2$ pre), ($L_1$ pre, $L_2$ post), ($L_1$ post, $L_2$ pre) and ($L_1$ post, $L_2$ post), where pre/post signify whether we take the representation before or after adaptation.

Prior work has shown that neural network representations may vary across different random seeds raghu2017svcca, morcos2018insights, li2015convergent, Wang2018ToWE, organically resulting in CCA similarity scores much less than 1. So to identify the effect of the inner loop on the representation, we plot the CCA similarities of (i) ($L_1$ pre, $L_2$ pre) against ($L_1$ pre, $L_2$ post) and (ii) ($L_1$ pre, $L_2$ pre) against ($L_1$ post, $L_2$ pre) and (iii) ($L_1$ pre, $L_2$ pre) against ($L_1$ post, $L_2$ post) separately across the different random seeds and different layers. We then compute the line of best fit for each plot. If the line of best fit fits the data and is close to $y = x$, this suggests that the inner loop adaptation doesn't affect the features much – the similarity before adaptation is very close to the similarity after adaptation.

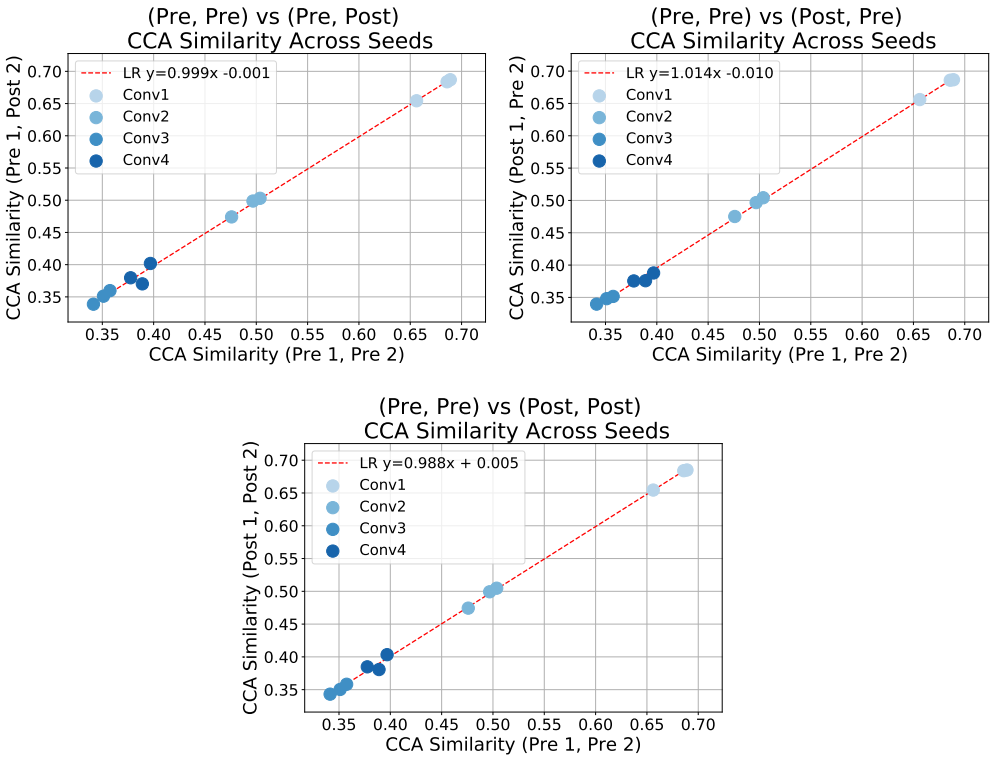



Figure App.2: **Computing CCA similarity pre/post adaptation across different random seeds further demonstrates that the inner loop doesn't change representations significantly.** We compute CCA similarity of $L_1$ from seed 1 and $L_2$ from seed 2, varying whether we take the representation *pre* (before) adaptation or *post* (after) adaptation. To isolate the effect of adaptation from inherent variation in the network representation across seeds, we plot CCA similarity of of the representations before adaptation against representations after adaptation in three different combinations: (i) ($L_1$ pre, $L_2$ pre) against ($L_1$ pre, $L_1$ post), (ii) ($L_1$ pre, $L_2$ pre) against ($L_1$ pre, $L_1$ post) (iii) ($L_1$ pre, $L_2$ pre) against ($L_1$ post, $L_2$ post). We do this separately across different random seeds and different layers. Then, we compute a line of best fit, finding that in all three plots, it is almost identical to $y = x$, demonstrating that the representation does not change significantly pre/post adaptation. Furthermore a computation of the coefficient of determination $R^2$ gives $R^2 \approx 1$, illustrating that the data is well explained by this relation. In Figure App.3, we perform this comparison with CKA, observing the same high level conclusions.

The results are shown in Figure App.2. In all of the plots, we see that the line of best fit is almost exactly $y = x$ (even for the pre/pre vs post/post plot, which could conceivably be more different as both seeds change) and a computation of the coefficient of determination $R^2$ gives $R^2 \approx 1$ for all three plots. Putting this together with Figure 5.2, we can conclude that the inner loop adaptation step

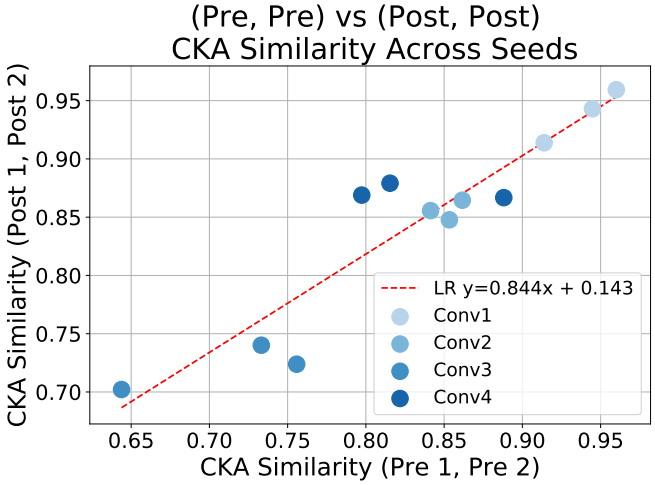



Figure App.3: We perform the same comparison as in Figure App.2, but with CKA instead. There is more variation in the similarity scores, but we still see a strong correlation between (Pre, Pre) and (Post, Post) comparisons, showing that representations do not change significantly over the inner loop.

doesn't affect the representation learned by any layer except the head, and that the learned representations and features are mostly reused as is for the different tasks.

### C.2.5 MiniImageNet-5way-1shot Freezing and CCA Over Training

Figure App.4 shows that from early on in training, on MiniImageNet-5way-1shot, that the CCA similarity between activations pre and post inner loop update is very high for all layers but the head. We further see that the validation set accuracy suffers almost no decrease if we remove the inner loop updates and freeze all layers but the head. This shows that even early on in training, the inner loop appears to have minimal effect on learned representations and features. This supplements the results seen in Figure 5.3 on MiniImageNet-5way-5shot.

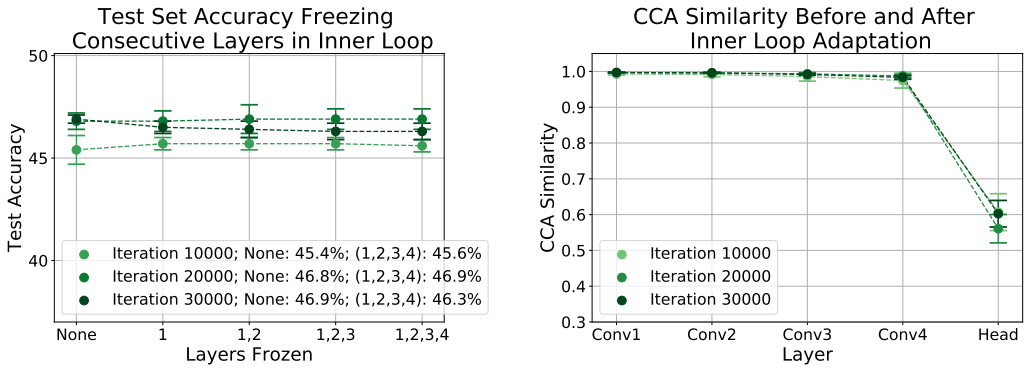



Figure App.4: **Inner loop updates have little effect on learned representations from early on in learning.** We consider freezing and representational similarity experiments for MiniImageNet-5way-1shot. We see that early on in training (from as few as 10k iterations in), the inner loop updates have little effect on the learned representations and features, and that removing the inner loop updates for all layers but the head have little-to-no impact on the validation set accuracy.

## C.3 ANIL Algorithm: More Details

In this section, we provide more details about the ANIL algorithm, including an example of the ANIL update, implementation details, and further experimental results.

### C.3.1 An Example of the ANIL Update

Consider a simple, two layer linear network with a single hidden unit in each layer: $\hat{y}(x;\boldsymbol{\theta}) = \theta_2(\theta_1 x)$. In this example, $\theta_2$ is the *head*. Consider the 1-shot regression problem, where we have access to examples $\left\{(x_1^{(t)}, y_1^{(t)}), (x_2^{(t)}, y_2^{(t)})\right\}$ for tasks $t = 1, \ldots, T$. Note that $(x_1^{(t)}, y_1^{(t)})$ is the (example, label) pair in the meta-training set (used for inner loop adaptation – support set), and $(x_2^{(t)}, y_2^{(t)})$ is the pair in the meta-validation set (used for the outer loop update – target set).

In the few-shot learning setting, we firstly draw a set of $N$ tasks and labelled examples from our meta-training set: $\left\{(x_1^{(1)}, y_1^{(1)}), \ldots, (x_1^{(N)}, y_1^{(N)})\right\}$. Assume for simplicity that we only apply one gradient step in the inner loop. The inner loop updates for each task are thus defined as follows:

$$\theta_1^{(t)} \leftarrow \theta_1 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_1} \tag{C.1}$$

$$\theta_2^{(t)} \leftarrow \theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2} \tag{C.2}$$

where $L(\cdot, \cdot)$ is the loss function, (e.g. mean squared error) and $\theta_i^{(t)}$ refers to a parameter after inner loop update for task $t$.

The task-adapted parameters for MAML and ANIL are as follows. Note how only the head parameters change per-task in ANIL:

$$\boldsymbol{\theta}_{\text{MAML}}^{(t)} = \left[\theta_1^{(t)}, \theta_2^{(t)}\right] \tag{C.3}$$

$$\boldsymbol{\theta}_{\text{ANIL}}^{(t)} = \left[\theta_1, \theta_2^{(t)}\right] \tag{C.4}$$

In the outer loop update, we then perform the following operations using the data from the meta-validation set:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}^{(t)}), y_2^{(t)})}{\partial \theta_1} \tag{C.5}$$

$$\theta_2 \leftarrow \theta_2 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}^{(t)}), y_2^{(t)})}{\partial \theta_2} \tag{C.6}$$

Considering the update for $\theta_1$ in more detail for our simple, two layer, linear network (the case for $\theta_2$ is analogous), we have the following update for MAML:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{MAML}}^{(t)}), y_2^{(t)})}{\partial \theta_1} \tag{C.7}$$

$$\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{MAML}}^{(t)}) = \left( \left[ \theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2} \right] \cdot \left[ \theta_1 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_1} \right] \cdot x_2 \right) \tag{C.8}$$

For ANIL, on the other hand, the update will be:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{ANIL}}^{(t)}), y_2^{(t)})}{\partial \theta_1} \tag{C.9}$$

$$\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{ANIL}}^{(t)}) = \left( \left[ \theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2} \right] \cdot \theta_1 \cdot x_2 \right) \tag{C.10}$$

Note the lack of inner loop update for $\theta_1$, and how we do not remove second order terms in ANIL (unlike in first-order MAML); second order terms still persist through the derivative of the inner loop update for the head parameters.

### C.3.2 ANIL Learns Almost Identically to MAML

We implement ANIL on MiniImageNet and Omniglot, and generate learning curves for both algorithms in Figure App.5. We find that learning proceeds almost identically for ANIL and MAML, showing that removing the inner loop has little effect on the learning dynamics.

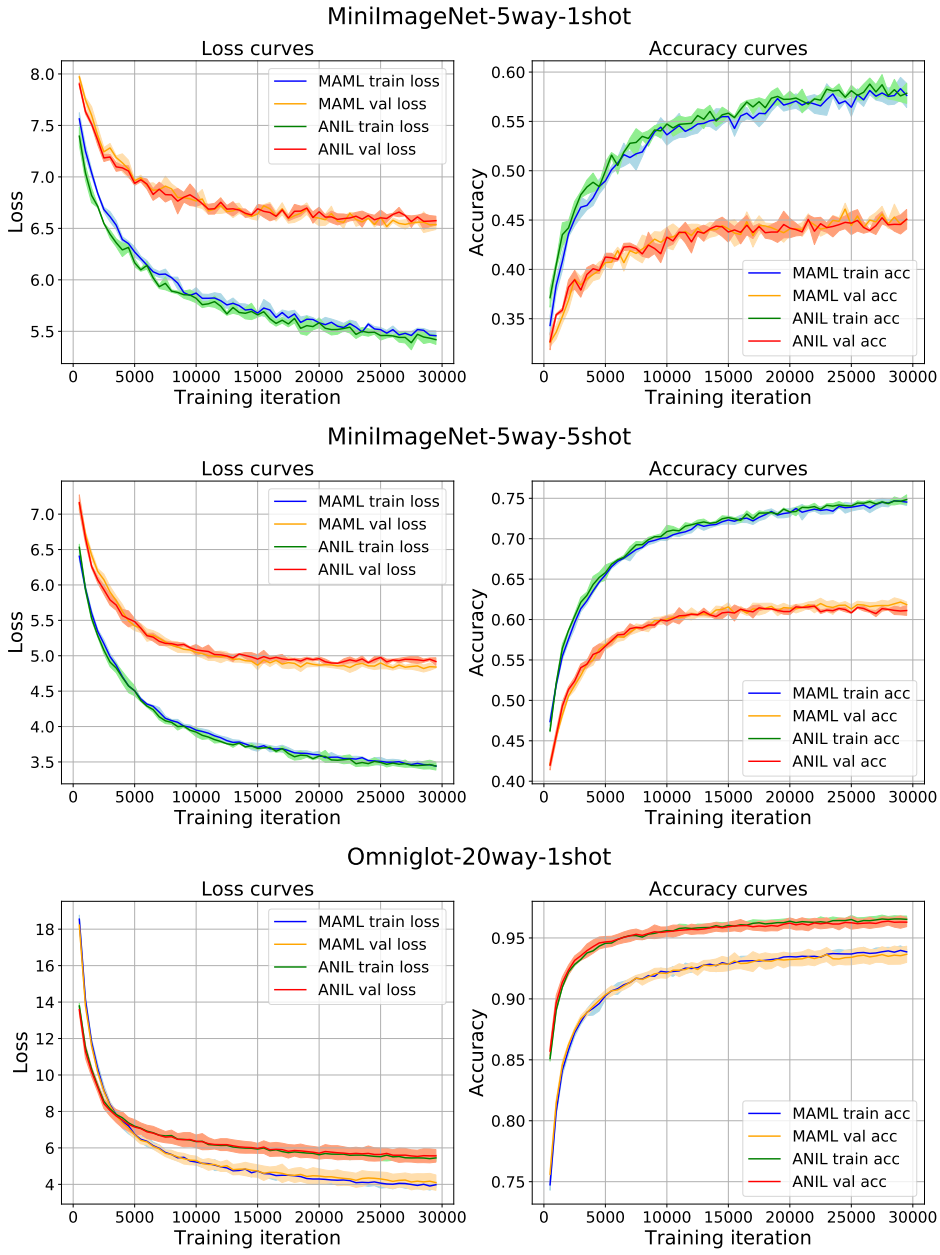



Figure App.5: **ANIL and MAML on MiniImageNet and Omniglot**. Loss and accuracy curves for ANIL and MAML on (i) MiniImageNet-5way-1shot (ii) MiniImageNet-5way-5shot (iii) Omniglot-20way-1shot. These illustrate how both algorithms learn very similarly over training.

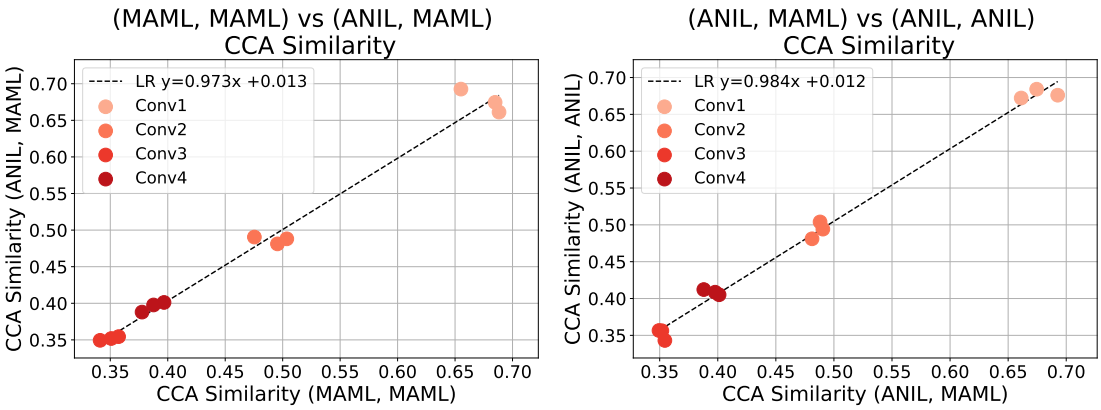



Figure App.6: **Computing CCA similarity across different seeds of MAML and ANIL networks suggests these representations are similar.** We plot the CCA similarity between an ANIL seed and a MAML seed, plotted against (i) the MAML seed compared to a different MAML seed (ii) the ANIL seed compared to a different ANIL seed. We observe a strong correlation of similarity scores in both (i) and (ii). This tells us that (i) two MAML representations vary about as much as MAML and ANIL representations (ii) two ANIL representations vary about as much as MAML and ANIL representations. In particular, this suggests that MAML and ANIL learn similar features, despite having significant algorithmic differences.

### C.3.3 ANIL and MAML Learn Similar Representations

We compute CCA similarities across representations in a MAML seed and an ANIL seed, and then plot these against the same MAML seed representation compared to a different MAML seed (and similarly for ANIL). We find a strong correlation between these similarities (Figure App.6), which suggests that MAML and ANIL are learning similar representations, despite their algorithmic differences. (ANIL and MAML are about as similar to each other as two ANILs are to each other, or two MAMLs are to each other.)

### C.3.4 ANIL Implementation Details

**Supervised Learning Implementation:** We used the TensorFlow MAML implementation open-sourced by the original authors finn2017model. We used the same model architectures as in the original MAML paper for our experiments, and train models 3 times with different random seeds. All models were trained for 30000 iterations, with a batch size of 4, 5 inner loop update steps, and an inner learning rate of 0.01. 10 inner gradient steps were used for evaluation at test time.

**Reinforcement Learning Implementation:** We used the open source PyTorch implementation of MAML for RL [1], due to challenges encountered when running the open-sourced TensorFlow implementation from the original authors. We note that the results for MAML in these RL domains do not exactly match those in the original paper; this may be due to large variance in results, depending on the random initialization. We used the same model architecture as the original paper (two layer MLP with 100 hidden units in each layer), a batch size of 40, 1 inner loop update step with an inner learning rate of 0.1 and 20 trajectories for inner loop adaptation. We trained three MAML and ANIL models with different random initialization, and quote the mean and standard deviation of the results. As in the original MAML paper, for RL experiments, we select the best performing model over 500 iterations of training and evaluate this model at test time on a new set of tasks.

[1] https://github.com/tristandeleu/pytorch-maml-rl

### C.3.5 ANIL is Computationally Simpler Than MAML

Table App.1 shows results from a comparison of the computation time for MAML, First Order MAML, and ANIL, during training and inference, with the TensorFlow implementation described previously, on both MiniImageNet domains. These results are average time for executing forward and backward passes during training (above) and a forward pass during inference (bottom), for a task batch size of 1, and a target set size of 1. Results are averaged over 2000 such batches. Speedup is calculated relative to MAML's execution time. Each batches' images were loaded into memory before running the TensorFlow computation graph, to ensure that data loading time was not captured in the timing. Experiments were run on a single NVIDIA Titan-Xp GPU.

During training, we see that ANIL is as fast as First Order MAML (which does not compute second order terms during training), and about 1.7x as fast as MAML. This leads to a significant overall training speedup, especially when coupled with the fact that the rate of learning for these ANIL and MAML is very similar; see learning curves in Appendix C.3.2. Note that unlike First Order MAML, ANIL also performs very comparably to MAML on benchmark tasks (on some tasks, First Order MAML performs worse finn2017model). During inference, ANIL achieves over a 4x speedup over MAML (and thus also 4x over First Order MAML, which is identical to MAML at inference time). Both training and inference speedups illustrate the significant computational benefit of ANIL over MAML.

| | Training: 5way-1shot | | |
|---|---|---|---|
| | Mean (s) | Median (s) | Speedup |
| MAML | 0.15 | 0.13 | 1 |
| First Order MAML | 0.089 | 0.083 | 1.69 |
| ANIL | 0.084 | 0.072 | 1.79 |

| | Training: 5way-5shot | | |
|---|---|---|---|
| | Mean (s) | Median (s) | Speedup |
| MAML | 0.68 | 0.67 | 1 |
| First Order MAML | 0.40 | 0.39 | 1.7 |
| ANIL | 0.37 | 0.36 | 1.84. |

| | Inference: 5way-1shot | | |
|---|---|---|---|
| | Mean (s) | Median (s) | Speedup |
| MAML | 0.083 | 0.078 | 1 |
| ANIL | 0.020 | 0.017 | 4.15 |

| | Inference: 5way-5shot | | |
|---|---|---|---|
| | Mean (s) | Median (s) | Speedup |
| MAML | 0.37 | 0.36 | 1 |
| ANIL | 0.076 | 0.071 | 4.87 |

Table App.1: **ANIL offers significant computational speedup over MAML, during both training and inference.** Table comparing execution times and speedups of MAML, First Order MAML, and ANIL during training (above) and inference (below) on Mini-ImageNet domains. Speedup is calculated relative to MAML's execution time. We see that ANIL offers noticeable speedup over MAML, as a result of removing the inner loop almost completely. This permits faster training and inference.

## C.4 Further Results on the Network Head and Body

### C.4.1 Training Regimes for the Network Body

We add to the results of Section 5.4.2 in the main text by seeing if training a head and applying that to the representations at test time (instead of the NIL

algorithm) gives in any change in the results. As might be predicted by Section 5.4.1, we find no change the results.

More specifically, we do the following:

- We train MAML/ANIL networks as standard, and do standard test time adaptation.
- For multiclass training, we first (pre)train with multiclass classification, then throw away the head and freeze the body. We initialize a new e.g. 5-class head, and train that (on top of the frozen multiclass pretrained features) with MAML. At test time we perform standard adaptation.
- The same process is applied to multitask training.
- A similar process is applied to random features, except the network is initialized and then frozen.

The results of this, along with the results from Table 5.5 in the main text is shown in Table App.2. We observe very little performance difference between using a MAML/ANIL head and a NIL head for each training regime. Specifically, task performance is purely determined by the quality of the features and representations learned during training, with task-specific alignment at test time being (i) unnecessary (ii) unable to influence the final performance of the model (e.g. multitask training performance is equally with a MAML head as it is with a NIL-head.)

| Method | MiniImageNet-5way-1shot |
|---|---|
| MAML training-MAML head | 46.9 ± 0.2 |
| MAML training-NIL head | 48.4 ± 0.3 |
| ANIL training-ANIL head | 46.7 ± 0.4 |
| ANIL training-NIL head | 48.0 ± 0.7 |
| Multiclass pretrain-MAML head | 38.4 ± 0.8 |
| Multiclass pretrain-NIL head | 39.7 ± 0.3 |
| Multitask pretrain-MAML head | 26.5 ± 0.8 |
| Multitask pretrain-NIL head | 26.5 ± 1.1 |
| Random features-MAML head | 32.1 ± 0.5 |
| Random features-NIL head | 32.9 ± 0.6 |

| Method | MiniImageNet-5way-5shot |
|---|---|
| MAML training-MAML head | 63.1 ± 0.4 |
| MAML training-NIL head | 61.5 ± 0.8 |
| ANIL training-ANIL head | 61.5 ± 0.5 |
| ANIL training-NIL head | 62.2 ± 0.5 |
| Multiclass pretrain-MAML head | 54.6 ± 0.4 |
| Multiclass pretrain-NIL head | 54.4 ± 0.5 |
| Multitask pretrain-MAML head | 32.8 ± 0.6 |
| Multitask pretrain-NIL head | 34.2 ± 3.5 |
| Random features-MAML head | 43.1 ± 0.3 |
| Random features-NIL head | 43.2 ± 0.5 |

Table App.2: **Test time performance is dominated by features learned, with no difference between NIL/MAML heads.** We see identical performances of MAML/NIL heads at test time, indicating that MAML/ANIL training leads to better learned features.

### C.4.2 Representational Analysis of Different Training Regimes

In Table App.3 we include results on using CCA and CKA on the representations learned by the different training methods. Specifically, we studied how similar representations of different training methods were to MAML training, finding a direct correlation with performance – training schemes learning representations most similar to MAML also performed the best. We computed similarity scores

| Feature pair | CCA Similarity | CKA Similarity |
|---|---|---|
| (MAML, MAML) | 0.51 | 0.83 |
| (Multiclass pretrain, MAML) | 0.48 | 0.79 |
| (Random features, MAML) | 0.40 | 0.72 |
| (Multitask pretrain, MAML) | 0.28 | 0.65 |

Table App.3: **MAML training most closely resembles multiclass pretraining, as illustrated by CCA and CKA similarities**. On analyzing the CCA and CKA similarities between different baseline models and MAML (comparing across different tasks and seeds), we see that multiclass pretraining results in features most similar to MAML training. Multitask pretraining differs quite significantly from MAML-learned features, potentially due to the alignment problem.

by averaging the scores over the first three conv layers in the body of the network.

APPENDIX D

# CHAPTER 6 APPENDIX

## D.1 Details on Datasets, Models and Hyperparameters

The Retina dataset consisted of around 250k training images, and 70k test images. The train test split was done by patient id (as is standard for medical datasets) to ensure no accidental similarity between the train/test dataset. The chest x-ray dataset is open sourced and available from [138], which has all of the details. Briefly, they have 223k training images and binary indicator for multiple diseases assiciated with each image extracted automatically from the meta data. The standard ImageNet (ILSVRC 2012) dataset was also used to pretrain models.

For dataset preprocessing we used mild random cropping, as well as standard normalization by the mean and standard deviation for ImageNet. We augmented the data with hue and contrast augmentations. For the Retina data, we used random horizontal and vertical flips, and for the chest x-ray data, we did not do random flip. We did not do model specific hyperparameter tuning on each target data, and used fixed standard hyperparameters.

For experiments on the Retina data, we trained the standard ImageNet models, Resnet50 and Inception-v3, by replacing the final 1000 class ImageNet classification head with a five class head for DR diagnosis, or five classes for the five different chest x-ray diseases. We use the sigmoid activation at the top instead of the multiclass softmax activation, and the train the models in the multi-label binary classification framework.

The CBR family of small convolutional neural networks consists of multiple conv2d-batchnorm-relu layers followed by a maxpool. Each maxpool has spatial window (3x3) and stride (2x2). For each CBR architecture, there is one filter size for all the convolutions (which all have stride 1). Below, conv-n denotes a 2d convolutionl with n output channels.

- **CBR-LargeT**(all) has 7x7 conv filters: (conv32-bn-relu) maxpool (conv64-bn-relu) maxpool (conv128-bn-relu) maxpool (conv256-bn-relu) maxpool (conv512-bn-relu) global avgpool, classification
- **CBR-LargeW**(ide) has 7x7 conv filters: (conv64-bn-relu) maxpool (conv128-bn-relu) maxpool (conv256-bn-relu) maxpool (conv512-bn-relu) maxpool, global avgpool, classification.
- **CBR-Small** has 7x7 conv filters: (conv32-bn-relu) maxpool (conv64-bn-relu) maxpool (conv128-bn-relu) maxpool (conv256-bn-relu) maxpool global avgpool, classification
- **CBR-Tiny** has 5x5 conv filters: (conv64-bn-relu) maxpool (conv128-bn-relu) maxpool (conv256-bn-relu) maxpool (conv512-bn-relu) maxpool, global avgpool, classification.

The models on Retina are trained on $587 \times 587$ images, with learning rate 0.001 and a batch size of 8 (for memory considerations.) The Adam optimizer is used. The models on the chest x-ray are trained on $224 \times 224$ images, with a batch size of 32, and vanilla SGD with momentum (coefficient 0.9). The learning rate scheduling is inherited from the ImageNet training pipeline, which warms up from 0 to $0.1 \times \frac{32}{256}$ in 5 epochs, and then decay with a factor of 10 on epoch 30, 60, and 90, respectively.

| Model | Init Method | 5k | 10k | 50k | 100k |
|---|---|---|---|---|---|
| Resnet50 | ImageNet Pretrained | 94.6% | 94.8% | 95.7% | 96.0 |
| Resnet50 | Random Init | 92.2% | 93.3% | 95.3% | 95.9% |
| CBR-LargeT | Random Init | 93.6% | - | - | - |
| CBR-LargeT | Pretrained | 93.9% | - | - | - |
| CBR-LargeW | Random Init | 93.6% | - | - | - |
| CBR-LargeW | Pretrained | 93.7% | - | - | - |
| Resnet50 | Conv1 Pretrained | 92.9% | - | - | - |
| Resnet50 | Mean Var Init | - | 94.4% | 95.5% | 95.8% |

Table App.1: **Additional performance results when varying initialization and the dataset size on the Retina task.** For Resnet50, we show performances when training on very small amounts of data. We see that even finetuning (with early stopping) on 5k datapoints beats the results from performing fixed feature extraction, Figure App.4, suggesting finetuning should always be preferred. For 5k, 10k datapoints, we see a larger gap between transfer learning and random init (closed by 50k datapoints) but this is likely due to the enormous size of the model (typically trained on 1 million datapoints) compared to the dataset size. This is supported by evaluating the effect of transfer on CBR-LargeT and CBR-LargeW, where transfer again does not help much. (These are one third the size of Resnet50, and we expect the gains of transfer to be even more minimal for CBR-Small and CBR-Tiny.) We also show results for using the MeanVar init, and see some gains in performance for the very small data setting. We also see a small gain on 5k datapoints when just reusing the conv1 weights for Resnet50.

## D.2 Additional Dataset Size Results

Complementing the data varying experiments in the main text, we additional experiments on varying the amount of training data, fidning that for around 50k datapoints, we return to only seeing a fractional improvement of transfer learning. Future work could study how hybrid approaches perform when less data is available.

## D.3 CCA Details

For full details on the implementation of CCA, we reference prior work [253, 220], as well as the open sourced code (the source of our implementation): https://github.com/google/svcca

One challenge we face when implementing CCA is the large size of the convolutional activations. These activations have shape $(n, h, w, c)$, where $n$ is the number of datapoints, $c$ the number of channels, and $h, w$ the spatial dimensions. These values all vary significantly across the network, e.g. conv1 has shape $(n, 294, 294, 64)$, while activations at the end of block 3 have shape $(n, 19, 19, 1024)$. Because CCA is sensitive to both the number of datapoints $n$ (actually $hwn$ for convolutional layers) and the number of neurons – $c$ for large convolutional layers – there is large variations in scaling across different layers in the model. To address this, we do the following: let $L$ and $L'$ be the layers we want to compare, with shape (height, width, channels), $(h_L, w_L, c_L)$. We apply CCA as follows:

- Pick $p$, the total number of image patches to compute activation vectors and CCA over, and $d$, the maximum number of neuron activation vectors to correlate with
- Pick the number of datapoints $n$ so that $nh_Lw_L = p$.
- Sample $d$ of the $c_L$ channels, and apply CCA to the resulting $d$ x $nh_Lw_L$ activation matrices.
- Repeat over samples of $d$ and $n$.

This works much better than prior approaches of averaging over all of the spatial

dimensions [220], or flattening across all of the neurons [253] (too computationally expensive in this setting.)

## D.4 Additional Results from Representation Analysis

| Description | Conv1 | Block1 | Block2 | Block3 | Block4 |
|---|---|---|---|---|---|
| Resnet50 CCA(ImNet1, ImNet2) | 0.865 | 0.559 | 0.421 | 0.343 | 0.313 |
| Resnet50 CCA(Rand1, Rand2) | 0.647 | 0.369 | 0.277 | 0.256 | 0.276 |
| Resnet50 Diff | **0.218** | **0.191** | **0.144** | **0.086** | **0.037** |

| Description | Pool1 | Pool2 | Pool3 | Pool4 |
|---|---|---|---|---|
| CBR-Small CCA(ImNet1, ImNet2) | 0.825 | 0.709 | 0.477 | 0.395 |
| CBR-Small CCA(Rand1, Rand2) | 0.723 | 0.541 | 0.401 | 0.349 |
| CBR-Small Diff | **0.102** | **0.168** | **0.076** | **0.046** |

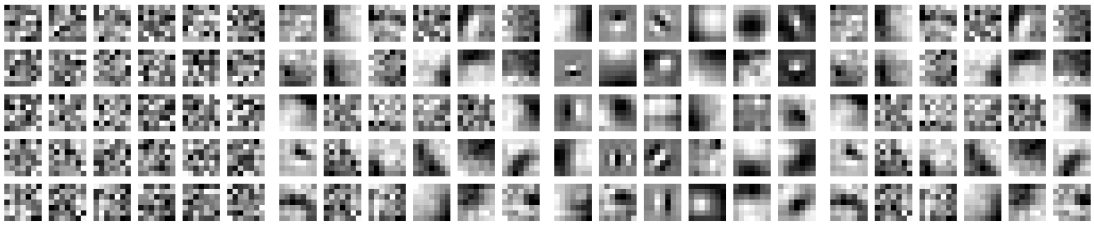
Table App.2: **Representational comparisons between trained ImageNet models with different seeds highlight the variation of behavior in higher and lower layers, and differences between larger and smaller models.** We compute CCA similarity between representations at different layers when training from different random seeds with (i) (the same) pretrained weights (ii) different random inits, for Resnet and CBR-Small. The results support the conclusions of the main text. For Resnet50, in the lowest layers such as Conv1 and Block1, we see that representations learned when using (the same) pretrained weights are much more similar to each other (diff 0.2 in CCA score) than representations learned from different random initializations. This $\sim 0.2$ difference is also much higher than (somewhat) corresponding differences in CBR-Small, for Pool1, Pool2. Actually, as Resnet50 is much deeper, the large difference in Block1 is very striking. (Block 1 alone contains much more layers than all of CBR-Small.) By Block3 and Block4 however, the CCA similarity difference between pretrained representations and those from random initialization is much smaller, and slightly lower than the differences for Pool3, Pool4 in CBR-Small, suggesting that pretrained weights are not having much of a difference on the kinds of functions learned. For CBR-Small, we also see that pretrained weights result in larger differences between the representations in the lower layers, but these become much smaller in the higher layers. We also observe that representations in CBR-Small trained from random initialization (especially in the lower layers e.g. Pool1) are more similar to each other than in Resnet50, suggesting things move more.

Here, we include some additional results studying the representations of these models. We perform more representational similarity comparisons be-

tween networks trained from (the same) pretrained weights (as is standard), but different random seeds. We do this for Resnet50 (a large model) and CBR-Small (a small model), and Table App.2 includes these results as well as similarity comparisons for networks trained with different random seeds and *different* random initializations as a baseline. The comparisons across layers and models is slightly involved, but as we detail below, the evidence further supports the conclusions in the main text:

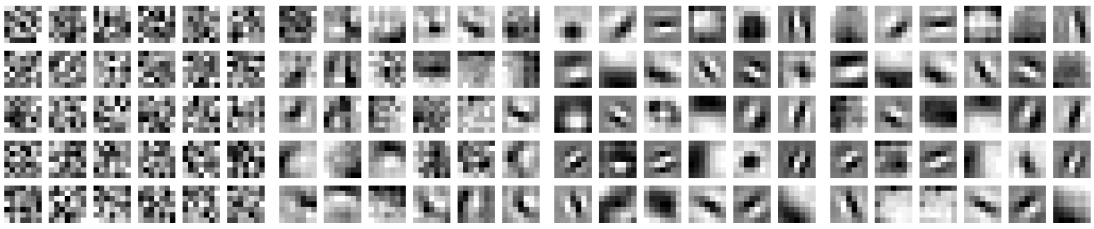
- *Larger models change less through training.* Comparing CCA similarity scores across models is a little challenging, due to different scalings, so we look at the difference in CCA similarity between two models trained with pretrained weights, and two models trained from random initialization, for Resnet50 and CBR-Small. Comparing this value for Conv1 (in Resnet50) to Pool1 (in CBR-Small), we see that pretraining results in much more similar representations compared to random initialization in the large model over the small model.
- *The effect of pretraining is mostly limited to the lowest layers* For higher layers, the CCA similarities between representations using pretrained weights and those trained from random initializations are closer, and the difference between CBR-Small and Resnet-50 is non-existent, suggesting that the effects of pretraining mostly affect the lowest layers across models, with finetuning changing representations at the top.

Figure App.1 and Figure App.2 compare the first layer filters between transfer learning and training from random initialization on the CheXpert data for the CBR-Small and Resnet-50 architectures, respectively. Those results complement Figure 6.5 in the main text.



(a) rand init (b) final (rand init) (c) transfer init (d) final (transfer)

Figure App.1: **First layer filters of CBR-Small on the CheXpert data.** (a) and (c) show the randomly initialized filters and filters initialized from a model (the same architecture) pre-trained on ImageNet. (b) and (d) shows the final converged filters from the two different initializations, respectively.



(a) rand init (b) final (rand init) (c) transfer init (d) final (transfer)

Figure App.2: **First layer filters of Resnet-50 on the CheXpert data.** (a) and (c) show the randomly initialized filters and filters initialized from a model (the same architecture) pre-trained on ImageNet. (b) and (d) shows the final converged filters from the two different initializations, respectively.

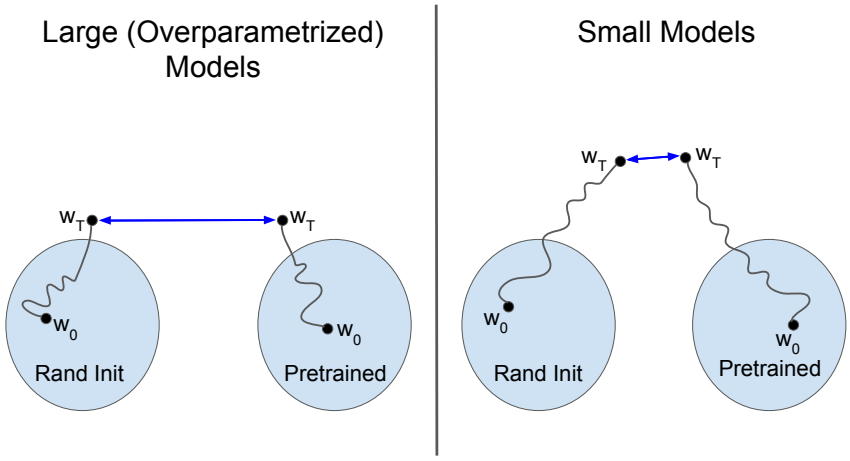



Figure App.3: **Larger models move less through training than smaller networks.** A schematic diagram of our intuition for optimization for larger and smaller models.

## D.5 The Fixed Feature Extraction Setting

To complete the picture, we also study the fixed feature extractor setting. While the most popular methodology for transfer learning is to initialize from pretrained weights and fine-tune (train) the entire network, an alternative is to initialize all layers up to layer $L$ with pretrained weights. These are then treated as a fixed feature extractor, with only layers $L + 1$ onwards, being trained. There are two variants of this fixed feature extractor experiment: **[1]** Initialize all layers with pretrained weights and only train layer $L + 1$ onwards. **[2]** Initialize only up to layer $L$ with pretrained weights, and layer $L + 1$ onwards randomly; then train only layers $L + 1$ onwards.

We implement both of these versions across different models trained on the Retina task in Figure App.4, and CheXpert in Figure App.5, including a baseline of using random features – initializing the network randomly, freezing up to layer $L$, and training layer $L + 1$ onwards. For the Retina task, we see that the pretrained ImageNet features perform significantly better than the random features baseline, but this gap is significantly closer on the chest x-rays.

More surprisingly however, there is little difference in performance between initializing all layers with pretrained weights and only up to layer $L$ with pretrained weights. This latter experiment has also been studied in [360], where they found that re-initializing caused drops in performance due to *co-adaptation*, where neurons in different layers have evolved together in a way that is not easily discoverable through retraining. This analysis was done for highly similar tasks (different subsets of ImageNet), and we hypothesise that in our setting, the significant changes of the higher layers (Figures 6.3, 6.4) means that the correct

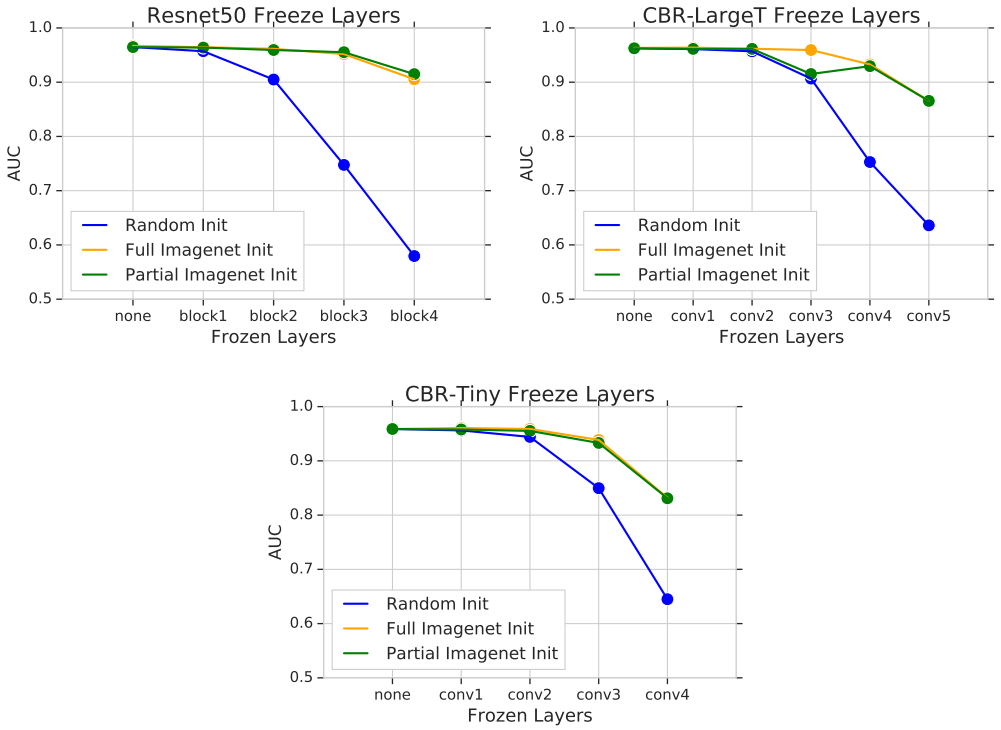



Figure App.4: **ImageNet features perform well as fixed feature extractors on the Retina task, and are robust to coadaptation performance drops.** We initialize (i) the full architecture with ImageNet weights (yellow) (ii) up to layer $L$ with ImageNet weights, and the rest randomly. In both, we keep up to layer $L$ fixed, and only train layers $L + 1$ onwards. We compare to a random features baseline, initializing randomly and training layer $L + 1$ onwards (blue). ImageNet features perform much better as fixed feature extractors than the random baseline (though this gap is much closer for the CheXpert dataset, Appendix Figure App.5.) Interestingly, there is no performance drop due to the *coadaptation* issue [360], with partial ImageNet initialization performing equally to initialzing with all of the ImageNet weights.

adaptation is naturally learned through training.

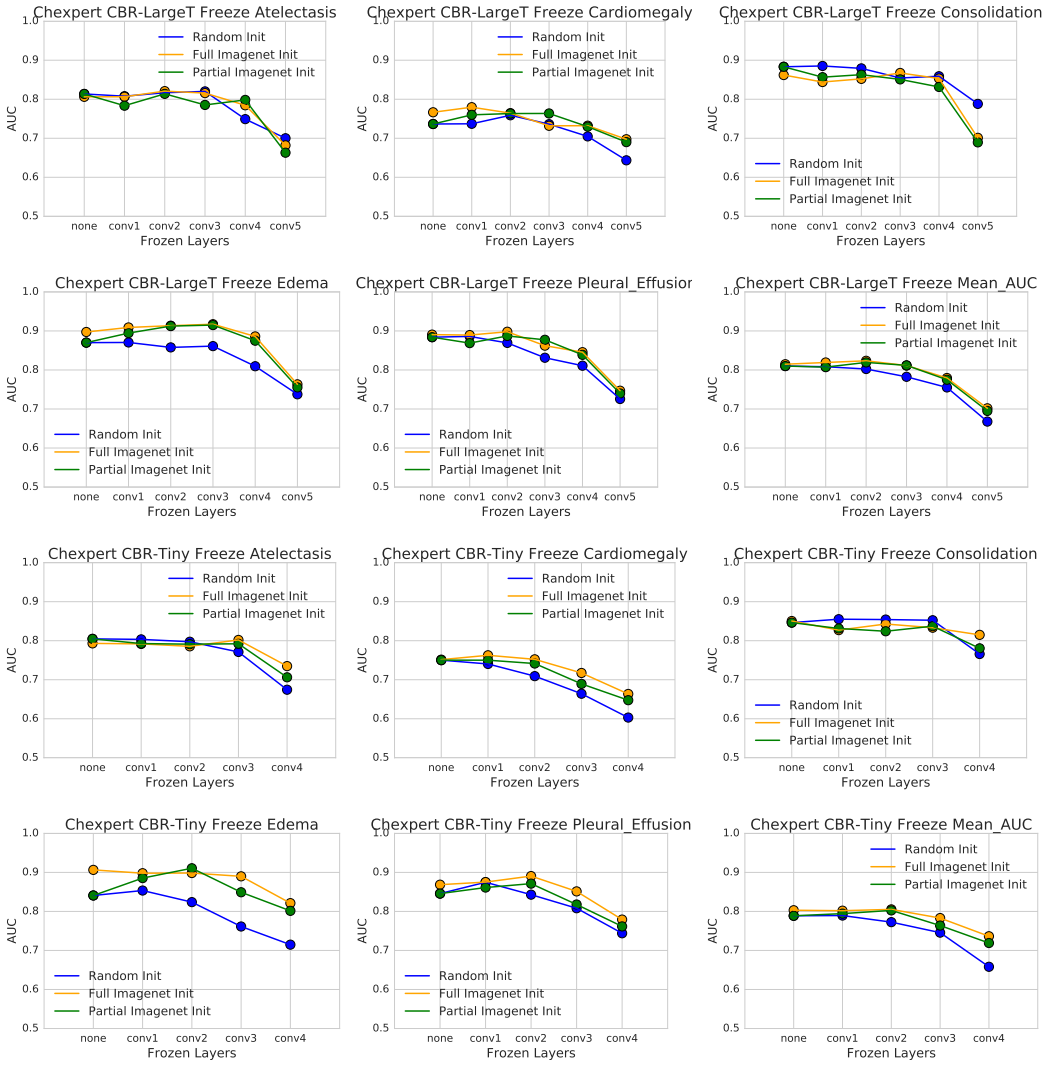



Figure App.5: **Experiments on freezing lower layers of CBR-LargeT and a CBR-Tiny model on the CheXpert data.** After random or transfer initialization, we keep up to layer $L$ fixed, and only train layers $L + 1$ onwards. ImageNet features perform better as fixed feature extractors than the random baseline for most diseases, but the gap is much closer than for the Retina data, Figure App.4. We again see that there is no significant performance drop due to coadaptation challenges.

## D.6 Additional Results on Feature Independent Benefits and Weight Transfusions

Figure App.6 visualizes the first layer filters from various initialization schemes. As shown in the main text, the *Mean Var* initialization could converge much faster than the baseline random initialization due to better parameter scaling transferred from the pre-trained weights. Figure App.7 shows more results on Retina with various architectures. We find that on smaller models, the effectiveness of the *Mean Var* initialization is less very pronounced, likely due to them being much shallower.

Figure App.8 shows all the five diseases on the CheXpert data for Resnet-50. Except for Cardiomegaly, we see benefits of the *Mean Var* initialization scheme on convergence speed in all other diseases.

### D.6.1 Batch Normalization Layers

Batch normalization layers [137] are an essential building block for most modern network architectures with visual inputs. However, these layers have a slightly different structure that requires more careful consideration when performing the Mean Var init. Letting $x$ be a batch of activations, batch norm computes

$$\gamma\left(\frac{(x-\mu_B)}{\sigma_B+\epsilon}\right)+\beta$$

Here, $\gamma, \beta$ are learnable scale, shift parameters, and $\mu_B, \sigma_B$ are an accumulated running mean and variance over the train dataset. Thus, in transfer learning,

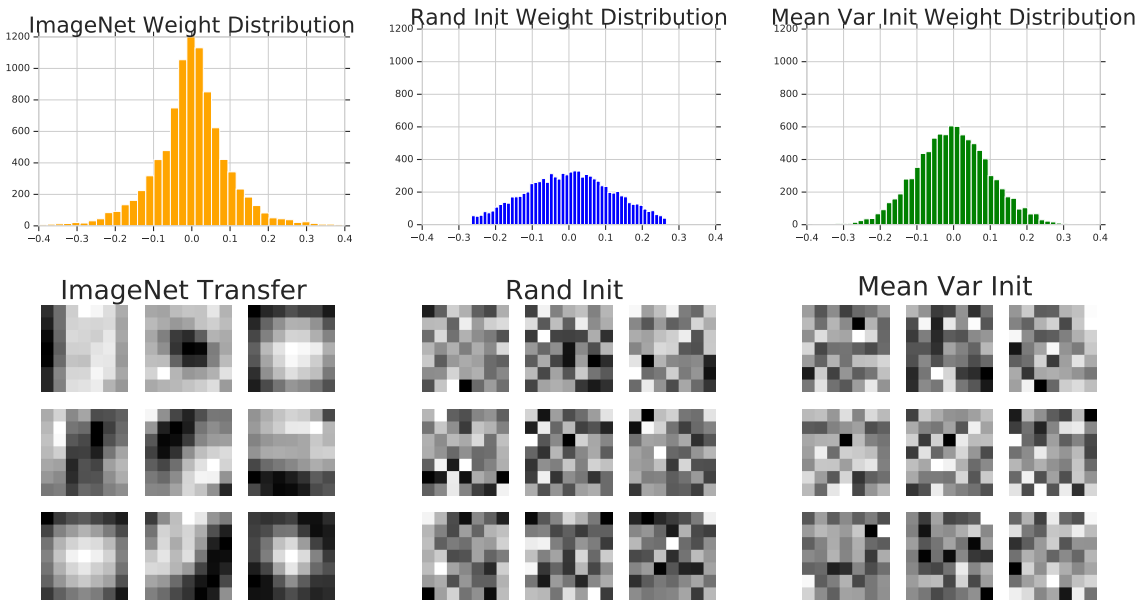



Figure App.6: **Distribution and filter visualization of weights initialized according to pretrained ImageNet weights, Random Init, and Mean Var Init**. The top row is a histogram of the weight values of the the first layer of the network (Conv 1) when initialized with these three different schemes. The bottom row shows some of the filters corresponding to the different initializations. Only the ImageNet Init filters have pretrained (Gabor-like) structure, as Rand Init and Mean Var weights are iid.

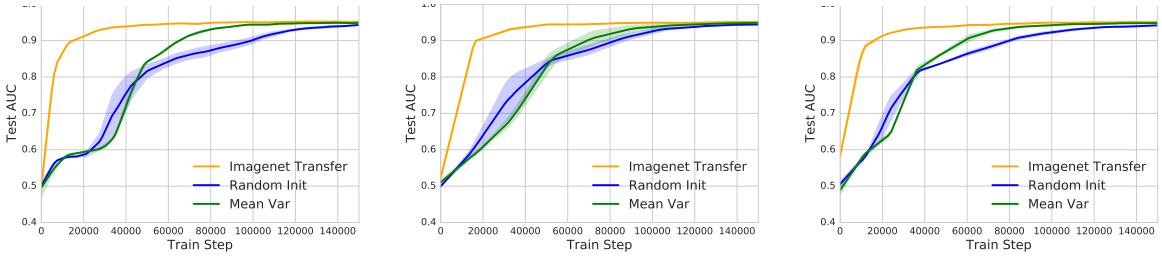
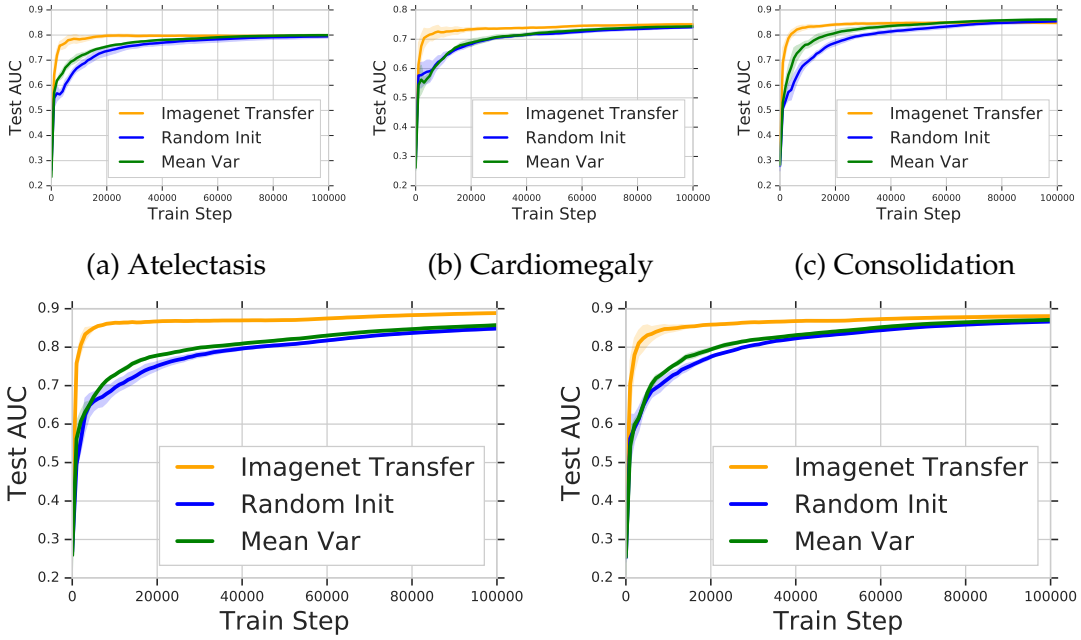



Figure App.7: **Comparison of convergence speed for different initialization schemes on Retina with various model architectures.** The three plots present the results for CBR-LargeW, CBR-Small and CBR-Tiny, respectively.




(a) Atelectasis (b) Cardiomegaly (c) Consolidation

(d) Edema (e) Pleural Effusion

Figure App.8: **Comparison of convergence speed for different initialization schemes on the CheXpert data with Resnet-50.**

$\mu_B, \sigma_B$ start off as the mean/variance of the ImageNet data activations, unlikely to match the medical image statistics. Therefore, for the Mean Var Init, we initialized all of the batch norm parameters to the identity: $\gamma, \sigma_B = 1, \beta, \mu_B = 0$. We call this the *BN Identity Init*. Two alternatives are *BN ImageNet Mean Var*, resampling the values of all batch norm parameters according to the ImageNet means and variances, and *BN ImageNet Transfer*, copying over the batch norm parameters from ImageNet. We compare these three methods in Figure App.9, with non-batchnorm layers initialized according to the Mean Var Init. Broadly, they perform similarly, with *BN Identity Init* (used by default in other Mean Var related experiments) performing slightly better. We observe that *BN ImageNet Transfer*, where the ImageNet batchnorm parameters are transferred directly to the medical images, performs the worst.

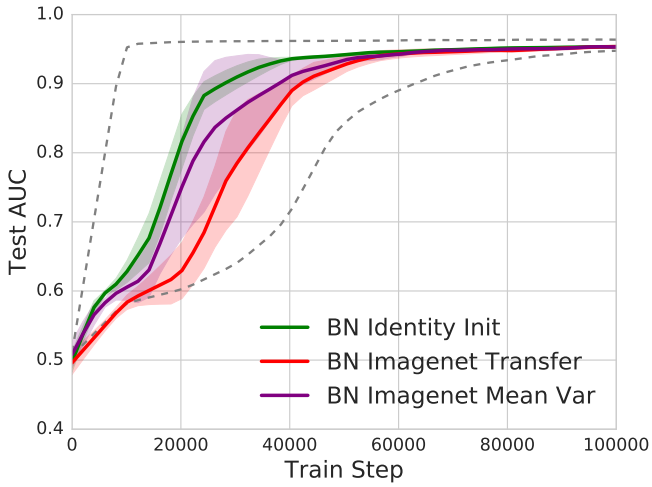
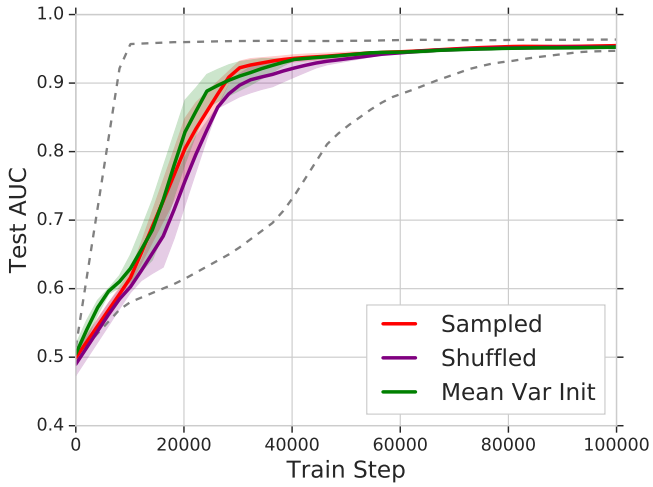



Figure App.9: **Comparing different ways of importing the weights and statistics for batch normalization layers.** The rest of the layers are initialized according to the Mean Var scheme. The two dashed lines show the convergence of the ImageNet init and the Random init for references. The lines are averaged over 5 runs.

### D.6.2 Mean Var Init vs Using Knowledge of the Full Empirical ImageNet Weight Distribution

In Figure App.6, we see that while the Mean Var Init might have the same mean and variance as the ImageNet weight distribution, the two distributions themselves are quite different from each other. We examined the convergence speed of initializing with the Mean Var Init vs initializing using knowledge of the entire empirical distribution of the ImageNet weights.




In particular, we looked at (1) *Sampling Init:* each weight is drawn iid from the full empirical distribution of ImageNet weights (2) *Shuffled Init:* random shuffle of the pretrained ImageNet weights to form a new initialization. (Note this is exactly sampling from the empirical distribution without replacement.) The results are illustrated in Figure D.6.2. Interestingly, Mean Var is very similar in convergence speed to both of these alternatives. This would suggest that further improvements in convergence speed might have to come from also modelling correlations between weights.

### D.6.3 Synthetic Gabor Filters

We test mathematically synthetic Gabor filters in place of learned Gabor filters on ImageNet for its benefits in speeding up the convergence when used as initialization in the first layer of neural networks. The Gabor filters are generated with the `skimage` package, using a code snippet, which is given in full in the corresponding paper, [254].

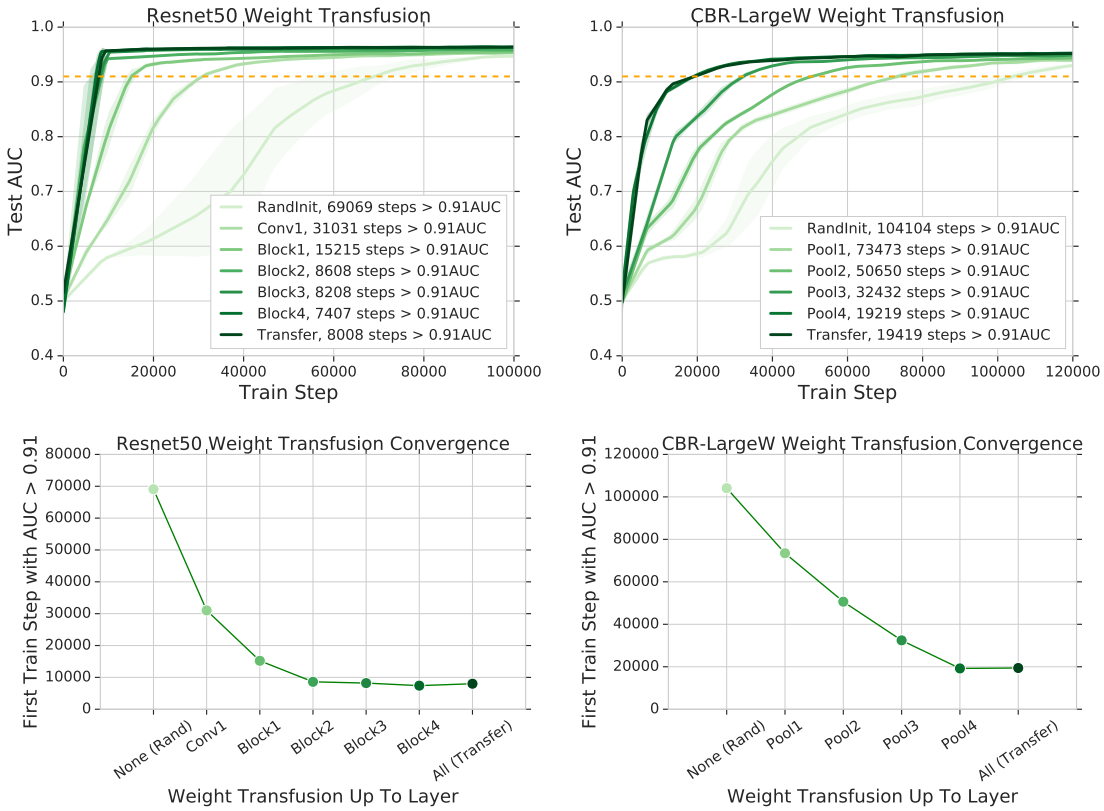



Figure App.10: **Weight transfusion results on Resnet50 (from main text) and CBR-LargeW**. These broadly show the same results — reusing pretrained weights for lowest layers give significantly larger speedups. Because CBR-LargeW is a much smaller model, there is slightly more change when reusing pretrained weights in high layers, but we still see the same diminishing returns pattern.

Figure App.12 visualize the synthetic Gabor filters. To ensure fair comparison, the synthesized Gabor filters are scaled (globally across all the filters with a single scale parameter) to match the numerical magnitudes of the learned Gabor filters.

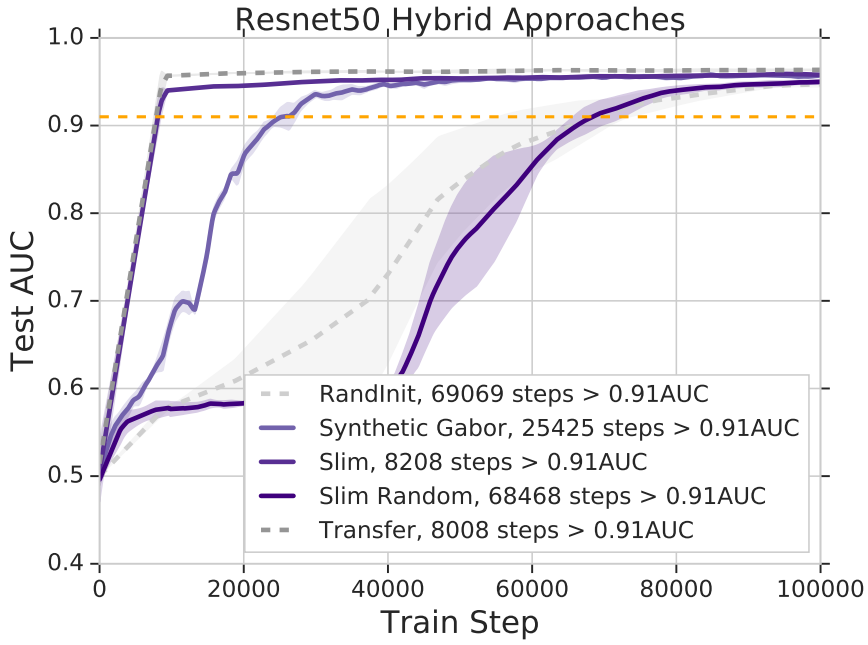



Figure App.11: **Convergence of Slim Resnet50 from random initialization**. We include the convergence of the slim Resnet50 — where layers in Block3, Block4 have half the number of channels, and when we don't use any pretrained weights. We see that it is significantly slower than the hybrid approach in the main text.

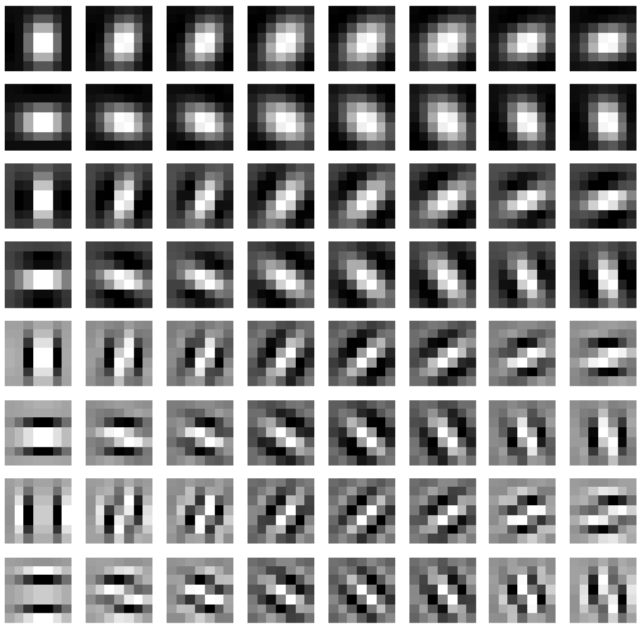


Figure App.12: Synthetic Gabor filters used to initialize the first layer if neural networks in some of the experiments in this study. The Gabor filters are generated as grayscale images and repeated across the RGB channels.

APPENDIX E

# CHAPTER 7 APPENDIX

## E.1 Proofs of Direct Uncertainty Prediction Results

We first prove Theorem 5.

*Proof.* To show unbiasedness of $h_{dup}$, we need to show that $\mathbb{E}[h_{dup}] = \mathbb{E}[U(\mathbf{Y})]$. But from the tower law (law of total expectation):

$$\mathbb{E}[h_{dup}] = \mathbb{E}\Big[\mathbb{E}[U(\mathbb{E}[\mathbf{Y}|O])\Big|g(O)]\Big] = \mathbb{E}[U(\mathbb{E}[\mathbf{Y}|O])]$$

To prove the biasedness of $h_{uvc}$, first note that

$$h_{uvc} = U(\mathbb{E}[\mathbf{Y}|g(O)]) = U(\mathbb{E}[\mathbb{E}[\mathbf{Y}|O]|g(O)])$$

by the conditional independence of $Y, g(O)$ given $O$. Next, by the fact that $U(\cdot)$ is concave and Jensen's inequality,

$$h_{uvc} = U(\mathbb{E}[\mathbb{E}[\mathbf{Y}|O]|g(O)]) \geq \mathbb{E}[U(\mathbb{E}[\mathbf{Y}|O]|g(O)] = h_{dup}$$

This is a strict inequality whenever the distribution of posteriors induced by conditioning on $g(O)$ is not a point-mass. Therefore we have that $h_{uvc}$ overestimates the the true uncertainty $U(\mathbf{Y})$. □

For specific $U(\cdot)$, we can compute the bias term by first computing $h_{uvc} - h_{dup}$ and then taking an expectation. For $U_{disagree}$, we have

$$h_{uvc} = U(\mathbb{E}[\mathbf{Y}|g(O)]) = 1 - \sum_l \mathbb{E}[\mathbb{E}[Y_l|O]|g(O)]^2$$

and

$$h_{dup} = \mathbb{E}[U(\mathbb{E}[\mathbf{Y}|O])|g(O)] = 1 - \sum_l \mathbb{E}[\mathbb{E}[Y_l|O]^2|g(O)]$$

And so,

$$h_{uvc} - h_{dup} = \sum_l \mathbb{E}[\mathbb{E}[Y_l|O]^2|g(O)] - \sum_l \mathbb{E}[\mathbb{E}[Y_l|O]|g(O)]^2$$

But this is just

$$Var\left(\sum_l \mathbb{E}[\mathbb{E}[Y_l|O]|g(O)]\right)$$

Taking expectations over values of $g(O)$ gives the bias, i.e.

$$\mathbb{E}\left[Var\left(\sum_l \mathbb{E}[\mathbb{E}[Y_l|O]|g(O)]\right)\right]$$

For $U_{var}$, we have

$$h_{uvc} = \sum_l l^2\mathbb{E}[\mathbb{E}[Y_l|O]|g(O)] - \left(\sum_l l\mathbb{E}[\mathbb{E}[Y_l|O]|g(O)]\right)^2$$

and

$$h_{dup} = \mathbb{E}\left[\sum_l l^2\mathbb{E}[Y_l|O] - \left(\sum_l l\mathbb{E}[Y_l|O]\right)^2 \middle| g(O)\right]$$

And so $h_{uvc} - h_{dup}$ becomes

$$\mathbb{E}\left[\left(\sum_l l\mathbb{E}[Y_l|O]\right)^2 \middle| g(O)\right] - \left(\sum_l l\mathbb{E}[\mathbb{E}[Y_l|O]|g(O)]\right)^2$$

Which is just

$$Var\left(\sum_l l \cdot \mathbb{E}[Y_l|O] \middle| g(O)\right)$$

Taking expectations over values of $g(O)$ like before gives the result.

## E.2 Mixture of Gaussians Setting

We train a DUP and UVC (Figure 7.1) on a synthetic task where data is generated from a mixture of Gaussians. All of our settings have uniform mixtures

of Gaussians, with the Gaussian mean vectors being drawn from $\mathcal{N}(0, 1/d)$ ($d$ corresponding to the dimension) so that in expectation, each mean vector has norm 1. The variance is set to be the identity. Like before, we set $x = g(o) = |o|$. We draw five labels for each $x$ from the posterior distribution over Gaussian centers given $x$, and apply $U_{disagree}()$ to the empirical histogram. As with the medical imaging application, we threshold these uncertainty scores (with threshold 0.5) to give a binary low uncertainty/high uncertainty label, which we use to train our DUPs and UVCs. Results are given in percentage AUC to account for some settings having unbalanced classes.

We train fully connected networks with two hidden layers of width 300 on this task, using the SGD with momentum optimizer and an initial learning rate of 0.01.

## E.3 SVHN and CIFAR-10 Setting

In Section 7.2.2, we train DUP and UVC models to predict label disagreement on a synthetic task on SVHN and CIFAR-10. The task setup is as follows: for each image in SVHN/CIFAR-10, we decide on a variance $(0, 1, 2, 3)$ for a Gaussian filter that is applied to the image. Three labels are then drawn for the image from a noisy distribution over labels, with the label noise distribution depending on the variance of the Gaussian filter. Specifically, for a Gaussian filter with variance 0, the noise distribution is just a point mass on the true label. For a Gaussian filter with variance 1, the three labels are drawn from a distribution with 0.02 mass on four incorrect labels, and the remaining 0.92 mass on the correct label.

For variance equal to 2, the labels are drawn from a distribution with 0.08 mass on four different labels, and remaining mass on the true label. For variance 3, this mass is now 0.12 on the incorrect labels.

A simple conv network, with 3x3 kernels and channels $64 - -128 - -256$, followed by fully connected layers of width 1000 and 200 (each with batch normalization) is trained on this dataset, with the UVC model trained on the empirical histogram, and the DUP model trained on a binary agree/disagree target. (Disagreement threshold is if at least one label disagrees.) We find that DUP outperforms UVC on both SVHN and CIFAR-10.

**Learned Features** Interestingly, we also observe that the features learned by the DUP and UVC models are different to each other. We apply saliency maps, specifically SmoothGrad [294] and IntGrad [302] to study the features that DUP and UVC pay attention to in the input image.

## E.4 Details of DUP in the Medical Domain

As described in Section 7.4, to train DUP models, we threshold the scores given by applying $U_{disagree}$, $U_{var}$ to the data $(x_i, \hat{\mathbf{p}}_i)$. Preliminary experiments in trying to directly regress onto the raw scores using mean-squared error performed poorly.

We threshold the scores as follows. For $U_{var}$ we thresholded at approximately 2/9, the variance when three doctors have more than an 'off by one' disagreement: more than a single disagreement, or a single grade disagreement.

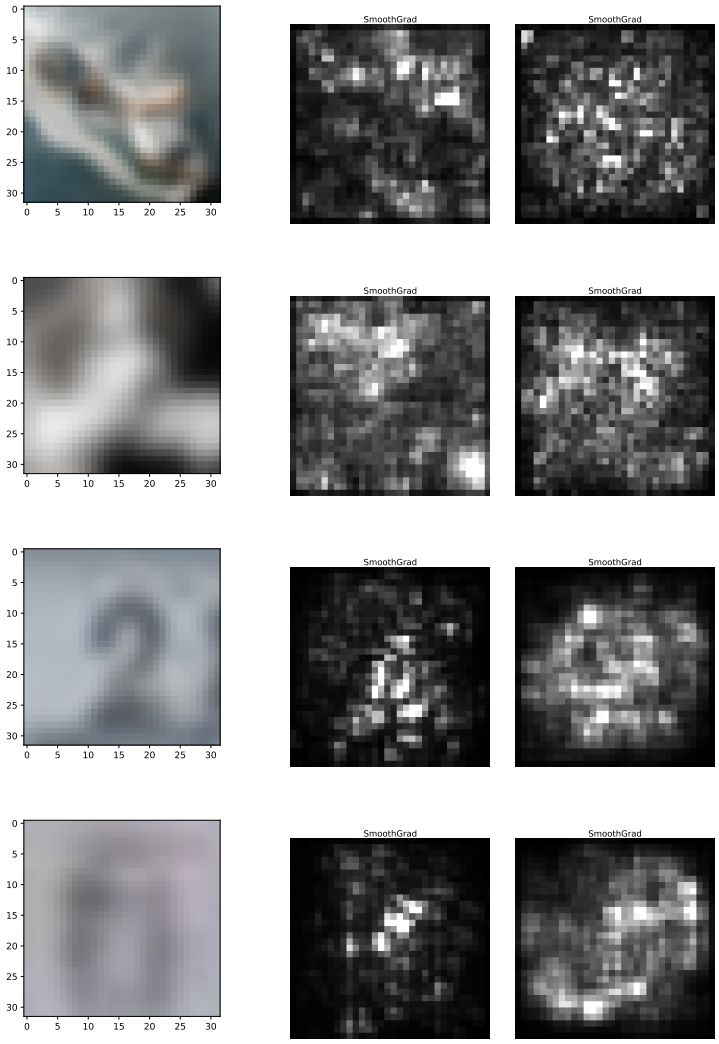



Figure App.1: **Saliency maps for DUP and UVC models on the SVHN/CIFAR-10 disagreement task.** The plot shows two images from the blurred CIFAR-10 dataset and two images from the blurred SVHN dataset. The second column is SmoothGrad applied to the UVC model, and the third SmoothGrad applied to the DUP model. We observe that the DUP and UVC models appear to be paying attention to different features of the dataset.

| Model Type | $T_{test}$ AUC | Majority | Median | Majority= 1 |
|---|---|---|---|---|
| Disagree Soft Targets | 76.3% | 79.0% | 78.7% | 81.6% |
| Disagree-P | **78.1%** | **81.0%** | 80.8% | **84.6%** |
| Disagree-PC | **78.1%** | 80.9% | **80.9%** | 84.5% |

| Model Type | $T_{test}$ AUC | Median= 1 | Referable |
|---|---|---|---|
| Disagree Soft Targets | 76.3% | 79.0% | 84.7% |
| Disagree-P | **78.1%** | **81.9%** | **86.2%** |
| Disagree-PC | **78.1%** | 81.8% | **86.2%** |

Table App.1: Using soft targets for disagreement prediction does not help in performance (AUC). Holdout AUC column corresponds to Disagreement Prediction Performance in Table 7.3, other columns refer to Table 7.4 in main text.

For $U_{disagree}$, where only the number of disagreements counts, we thresholded at 0.3, to prioritize being sensitive enough to disagreement cases and having more than 20% of the data marked as high disagreement. We also experimented with using soft targets for disagreement classification, but the results (Table App.1) showed that this was less effective than than having the binary 0/1 scores, likely because this makes the classification problem more like a regression.

Our model consists of an Inception-v3 base, with the ImageNet head removed and a small (2 hidden layer, 300 hidden units) fully connected neural network using Inception-v3 PreLogits to perform DUP. The full Inception-v3 network is trained with a batch size of 8 and learning rate 0.001 with the Adam optimizer. For training only the small neural network, we use the SGD with momentum optimizer, a batch size of 32 and learning rate of 0.01.

**Prelogits, Calibration and Regularization** Our training data for DUP models, $T_{train}^{(var)}, T_{train}^{(disagree)}$, only consists of $x_i$ with more than one label, and is too small to effectively train an Inception sized model end to end. Therefore, we use the prelogit embeddings of $x_i$ from a pretrained DR classification model (Histogram-

| Task | Model Type | Performance (AUC) |
|---|---|---|
| Variance Prediction | Variance-E2E-2H | 72.7% |
| Variance Prediction | Variance-LR | 72.4% |
| Disagreement Prediction | Disagree-LR | 75.9% |

Table App.2: Additional results from table 7.3.

E2E), and training smaller models on top of these embeddings. We do this both for the baseline, getting the Histogram-PC model, as well as the DUP models, Variance-PRC and Disagree-PC.

The $C$ suffix of all of these models corresponds to calibration on the logits. Following the findings of [103], we apply temperature scaling on the logits: we set the predictions of the model to be $f(F/T)$ where $f$ is the softmax function, applied pointwise, and $F$ are the logits. We initialize $T$ to 1, and then split e.g. $T_{train}^{(disagree)}$ into a $T_{train}^{'(disagree)}$ and a $T_{valid}^{'(disagree)}$, with 10% of the data in the validation set. We train as normal on $T_{train}^{'(disagree)}$, with $T$ fixed at 1, and then train on $T_{valid}^{'(disagree)}$, by *only* varying the temperature $T$, and holding all other parameters fixed.

The use of Prelogit embeddings and Calibration gives the strongest performing baseline UVC and DUPs: Histogram-PC, Variance-PRC and Disagree-PC. For the Variance DUP, an additional regularization term is added to the loss by having a separate regressing on the raw variance value.

**Additional Model: Variance-E2E** We tried a variant of Variance-E2E, Variance-E2E-2H, which has one head for predicting variance and the other for classifying, to enable usage of all the data. We then evaluate the variance head on $T_{test}$, but in fact noticed a small drop in performance, Table App.2.

**Do we need the Prelogit embeddings?** We tried seeing if we could match performance by training on pretrained classifier logits instead of the prelogit embeddings. Despite controlling for parameter difference by experimenting with more hidden layers, we found we were unable to match performance from the prelogit layer, Table App.2, compare to Table 7.3. This demonstrates that some information is lost between the prelogit and logit layers.

## E.5 Additional Results: Entropy, Finite Sample Behavior and Convergence Analysis

We performed additional experiments to further understand the properties of DUP and UVC models. For these experiments, we compare a representative DUP model, *Disagree-P*, to a representative UVC model, *Histogram-PC*.

Theorem 5 states that DUP offers benefits over UVC for *concave* target uncertainty functions. This is a natural property for measures of spread, simply stating that the measure of spread increases with averaging (probability distributions). In the main text, we concentrate on two such specific uncertainty functions, $U_{var}$ and $U_{disagree}$, which are particularly suited to the domain. However, other standard uncertainty functions, such as entropy, are also concave. We test the performance of DUP (*Disagree-P*) and UVC (*Histogram-PC*) with $U_{entropy}$ as the target function.

The results are shown in Table App.3, where we again see that DUP outperforms UVC.

| Task | | Model Type | Performance (AUC) |
|---|---|---|---|
| Entropy Prediction | UVC | Histogram-PC | 75.5% |
| Entropy Prediction | DUP | Disagree-P | **77.2%** |

Table App.3: DUP and UVC models trained with entropy as a target function. Again, we see that the DUP model outperforms the UVC model.

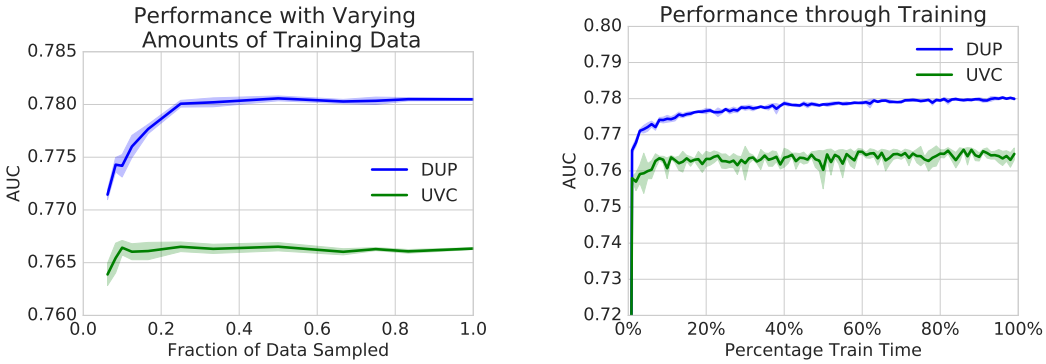



Figure App.2: **DUP and UVC performance during training and when varying train data size.** We study DUP (Disagree-P) and UVC (Histogram-PC) performance for varying amounts of training data. We find that the gap in performance is robust to variations in dataset size. For more than 30% of the data, performance of DUP and UVC remains relatively constant, supporting the applicability of Theorem 5 in the finite data setting. The right plot looks at performance through training, with the gap appearing rapidly early in training, and slowly widening.

We also study how model performance is impacted by different training set sizes (similar to the analysis in [40]). We subsample different amounts of the original training set $T_{train}^{(disagree)}$, and train DUP and UVC models on this subset. The results over 5 repeats of different subsamples and optimization runs are shown in Figure App.2.

We see that the performance gap between DUP and UVC is robust to train data size differences. Additionally, when $\geq 30\%$ of the training data is used, DUP and UVC performance remains relatively constant. This supports carrying over the results of Theorem 5) and the full joint distribution $f(o, \mathbf{y})$ to the finite data setting.

We also study convergence of DUP and UVC models. We find that the performance gap between DUP and UVC manifests very early in training (Figure App.2, right plot), and continues to gradually widen through training.

## E.6 Background on the Wasserstein Distance

Given two probability distributions $f, g$, and letting $\Pi(f, g)$ be all product probability distributions with marginals $f$, $g$, the Wasserstein distance between $p, q$ is

$$\|f - g\|_w = \inf_{\pi \in \Pi(f,g)} \mathbb{E}_{(r,t)\sim\pi} [d(r, t)]$$

where $d(,)$ is some metric. This distance has connections to optimal transport, and corresponds to the minimum cost (with respect to $d(,)$) of moving the mass in distribution $f$ so that it is matches the mass in distribution $g$. We can represent the amount of mass to move from $r$ to $t$ with $\pi(r, t)$. To be consistent with the mass at the start, $f(r)$, and the mass at the end $g(t)$ we must have that $\int_{t'} \pi(r, t') = f(r)$ and $\int_{r'} \pi(r', t) = g(t)$.

The result in the main text follows from the following theorem:

**Theorem 8.** *If $f, g$ are (discrete) probability distributions and $g$ is a point mass distribution at $t_0$, then $\pi \in \Pi(f, g)$ is uniquely defined as:*

$$\pi(r, t) = \begin{cases} 0 & \text{if } t \neq t_0 \\ f(r) & \text{if } t = t_0 \end{cases}$$

*Proof.* The proof is direct: for $t \neq t_0$, we must have $\int_{r'} \pi(r', t) = g(t) = 0$, and so $\int_{t'} \pi(r, t') = \pi(r, t_0) = f(r)$. □

We consider three different distances $d(,)$:

1 *Absolute Value* $d(r,t) = |r-t|$. This follows an interpretation in which the grades are equally spaced, so that all successive grade differences have the same weight.

2 *2-Wasserstein Distance* $d(r,t) = (r-t)^2$, and, to make into a metric

$$||f-g||_w = (\mathbb{E}_{(r,t)\sim\pi}[d(r,t)])^{1/2}$$

This adds a higher penalty for larger grade differences.

3 *Binary Disagreement* We set $d(r,t) = 0$ if $r = t$ and 1 otherwise.

APPENDIX F

# CHAPTER 8 APPENDIX

## F.1 Training Data and Models Details

Our training dataset consists of fundus photographs with labels corresponding to individual doctor grades. There are 5 possible DR grades and hence 5 possible class labels. A subset of this data has fundus photographs with more than one doctor grade, corresponding to multiple doctors individually and independently deciding on the grade for the image. The label for these images is not a one-hot class label but the empirical distribution of grades. For example, if an image $i$ has grades $\{2, 3, 3\}$, then its label would be $[0, 1./3, 2./3, 0, 0]$.

On this data, we train a convolutional neural network, an Inception-v3 model with weights pretrained from ImageNet and a new five class classification head. We train with the Adam optimizer [154] and an initial learning rate of 0.005. To better calibrate the model, we retrain the very top of the network (from the PreLogits layer) on just the data with two or more doctor grades.

**Training Error Probability Prediction Models** For Figures 8.4, 8.5 and 8.6, we use separate error probability prediction algorithms to predict the values of $\Pr[H_i]$ and $\Pr[M_i]$. The setup for predicting $\Pr[M_i]$ is as follows: after training the main convolutional neural network on the train dataset, we train a small fully connected deep neural network to take the prelogit embeddings of a train image $x_i$, and predict whether or not the main convolutional neural network was correct on that image. The label for the image is binary: agree/disagree on

whether the mass $m(x_i)$ put on referable by the convolutional neural network thresholded at 0.5 equals the mass on referable by the human doctor grades, again thresholded at 0.5.

The setup for training $\Pr[H_i]$ builds off of [**?**]. First, we only select cases for which we have at least two doctor grades. For these, we take the image embedding from the Prelogit layer of the large diagnostic convolutional neural network as input, and the label as a binary target. This label is defined as follows: we split the available doctor grades into two evenly sized sets $A$ and $B$. We aggregate all the grades in $A$ into a single referable/non-referable grade by averaging and thresholding at 0.5, and do the same for the grades in $B$. If these two aggregated grades agree, we label the image with 0 (agreement, low doctor error probability), if not, we label with 1 (disagreement, high doctor error probability.)

## F.2 Computing $\Pr[M_i]$

In Section 8.3.1, we overviewed the method used to define a well calibrated error probability for the output of the convolutional neural network. In Algorithm , we give a step-by-step overview of the implementation of this method. In our experiments, we set $C = 2000$.

**Algorithm 1** Model Error Probability Calibration

$Err_i = 0$ for $i$ in instances.
**for** $(r = 0; r < C; r{+}{+})$ **do** ▷ $C$ is a sufficiently large constant.
  $R = 0$
  **for** $i$ `in instances` **do** ▷ Sample a doctor grade and count number of referables.
    `Sample doctor grade` $h^{(i)}$
    **if** $(h^{(i)} \geq 3)$ **then**
      $R \leftarrow R + 1$
    **end if**
  **end for**
  `Rank instances` $i$ `from highest to lowest` $m(x_i)$
  $A_R = \{i : rank(i) \leq R\}$
  **for** $i$ `in instances` **do** ▷ Assign binary grades to instances. Top $R$ referable.
    **if** $(i \in A_R)$ **then**
      $m_i \leftarrow 1$
    **else**
      $m_i \leftarrow 0$
    **end if**
    $Err_i \leftarrow Err_i + |m_i - a_i|$ ▷ $a_i$ is the adjudicated grade
  **end for**
**end for**
**return** $\Pr[M_i] = Err_i/C$ ▷ Error probability by averaging over number of repetitions.

## F.3 Triage and Allocation Algorithm

When using triage to reallocate human effort, we first order the instances by their triage scores, and then fully automate the first $\alpha N$ of them. On the remainder $(1 - \alpha)N$ images, we allocate the budget of $cN$ human doctor grades we have available. To allocate this set of $cN$ grades, we use the equal coverage protocol: each of the remaining $(1 - \alpha)N$ cases gets $cN/((1 - \alpha)N)$ grades. If this is a non-integer amount, with $r$ spare grades, the $r$ cases identified as the hardest (according to the triage scores) get an additional grade. We then compute the

final binary decision by taking the mean grade (for each case) and thresholding by 0.5 (the majority vote.)

### F.3.1 Results on other Thresholds

As described in Section 8.2.4, at evaluation, for an instance with multiple grades we aggregate all the scores by taking the mean and thresholding. In the main text, we pick this threshold to be 0.5, corresponding to the majority vote of all the grades. In Figure App.1, we show the results corresponding to Figure 8.4 in the main text, but for when we take the thresholds to be 0.3 (top row) and 0.4 (bottom row). We see that the qualitative conclusions remain the same – combining human and algorithmic effort beats the full allocation and equal coverage protocols for both triage by the error prediction models and triage by the ground truth. We also see the same significant gap between ground truth and triage by error predictions.

Note that this choice of threshold affects the choice of $q_R$, which is chosen so that the number of cases marked as referable by the model matches the number of cases marked as referable by the aggregated and thresholded grade of the human doctors, and could potentially affect the results of Figure 8.6. However, as shown in Figure App.2, the choice of aggregation threshold does not affect the identification of zero error subsets.

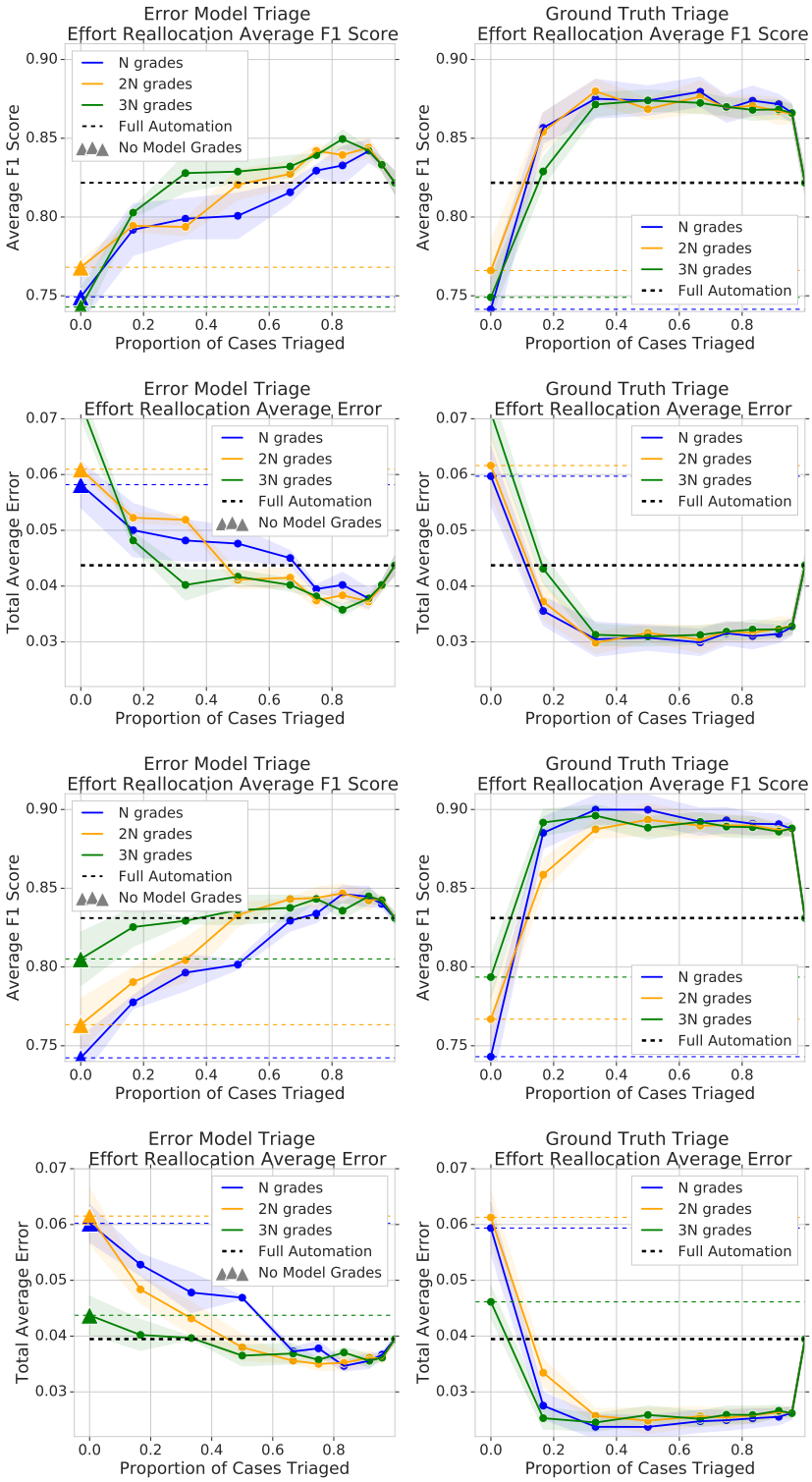



Figure App.1: **Triage and human expert effort reallocation for different thresholds:** 0.3 **top row,** 0.4 **bottom row.** We plot the same results as in Figure 8.4 in the main text, but for different thresholds — 0.3, 0.4 — for aggregation.

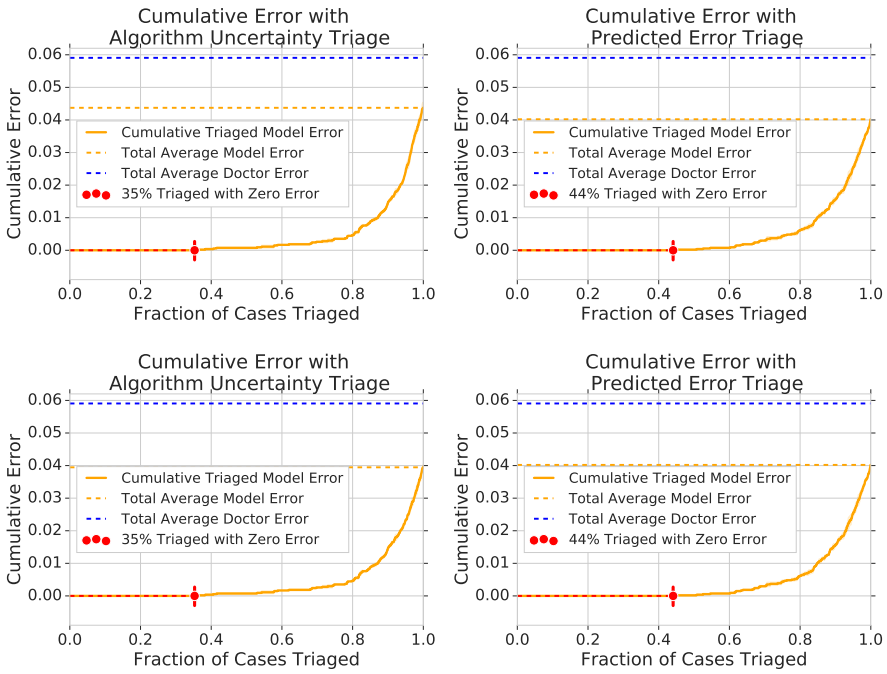



Figure App.2: **Triage for Zero Error subsets with thresholds** 0.3 **(top) and** 0.4 **(bottom) for aggregation.** The size of the zero error subsets remain the same, showing that the choice of threshold does not affect the identification of these subsets.

## F.4 Triage and Human Effort Reallocation with Model Grades

The triage process for effort reallocation – Figures 8.4, App.5 – assumes that the algorithm decision is not available for the $(1-\alpha)N$ cases that are not automated. This may be the situation if computing an algorithm decision is expensive (less likely) or (more likely) the algorithm decision is purposefully not shown in cases where it is unsure, so as not to bias the human doctors. However, another equally likely scenario is that the algorithm decision is also available 'for free' for the $(1-\alpha)N$ cases that are not fully automated. In Figure App.3, we show the effort reallocation results from triaging if the model grades were available for all the cases (compare to Figure 8.4 in the main text). We observe that all of the main conclusions – the optimal performance is through a combination of automation

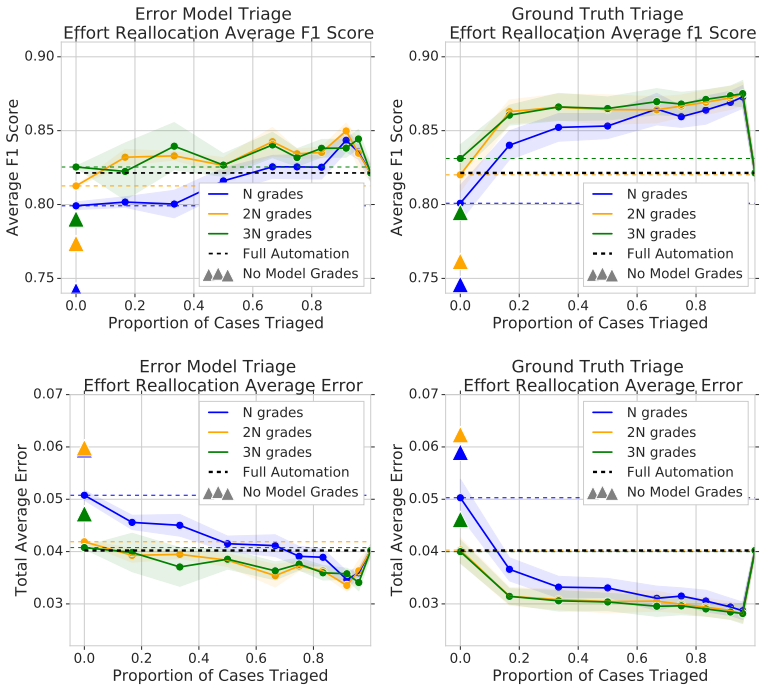



Figure App.3: **Effort reallocation results with the algorithm's grade being available for all cases.** Here we assume that the algorithm's grade is available for all the patient cases. We see that the same qualitative conclusions hold – a mix of automation and human effort outperforming the pure algorithm/human expert.

and human effort, which beats both full automation and the different equal coverage baselines.

## F.5 Results on Additional Holdout Dataset

The results in the main text are on the adjudicated evaluation dataset, which, aside from multiple independent grades by individual doctors, also have a consensus score, the adjudicated grade, which is used as a proxy for ground truth. To further validate our result, we use an additional holdout set which doesn't have an adjudicated grade, but does have many individual doctor grades. For each instance $i$, we use half of its grades to compute a proxy ground truth

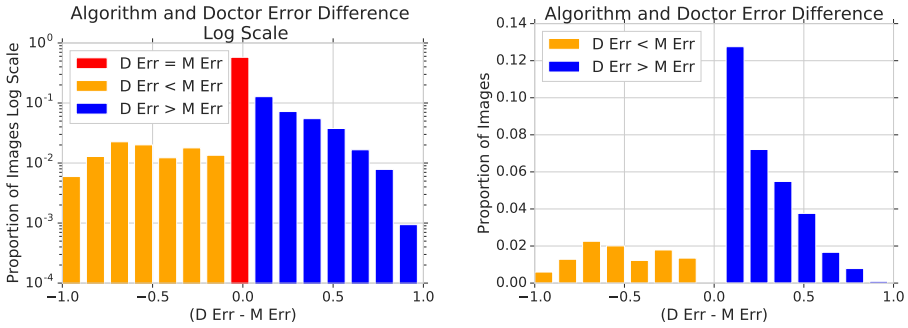



Figure App.4: **Histogram plot of $\Pr[H_i] - \Pr[M_i]$ for instances $i$ on the additional holdout evaluation dataset.** Compare to Figure 8.3 in the main text. We see a diversity of values across different instances.

grade, by aggregating and then thresholding the doctor grades. The other half of the grades are used in effort reallocation and evaluating the equal coverage baseline. The individual doctor grades in this dataset are slightly noiser (higher disagreement rates) than in the adjudicated evaluation dataset. Nevertheless this additional evaluation also supports all of the main findings.

The results on this dataset are qualitatively identical to those with the adjudicated data. Again, we see that there is a diverse spread of $\Pr[H_i] - \Pr[M_i]$ across instances, with around 10% of the instances having the human experts perform better (Figure App.4).

This diversity continues to be predictable, and we observe that triaging (by the error prediction models and by the ground truth) to combine human expert effort and and the algorithm's decisions, Figure App.5, also demonstrates that this combination works better than both full automation and equal coverage – the same conclusions seen in Figure 8.4 in the main text. Like the main text, we see a gap between triaging by the error prediction models and the ground truth score.

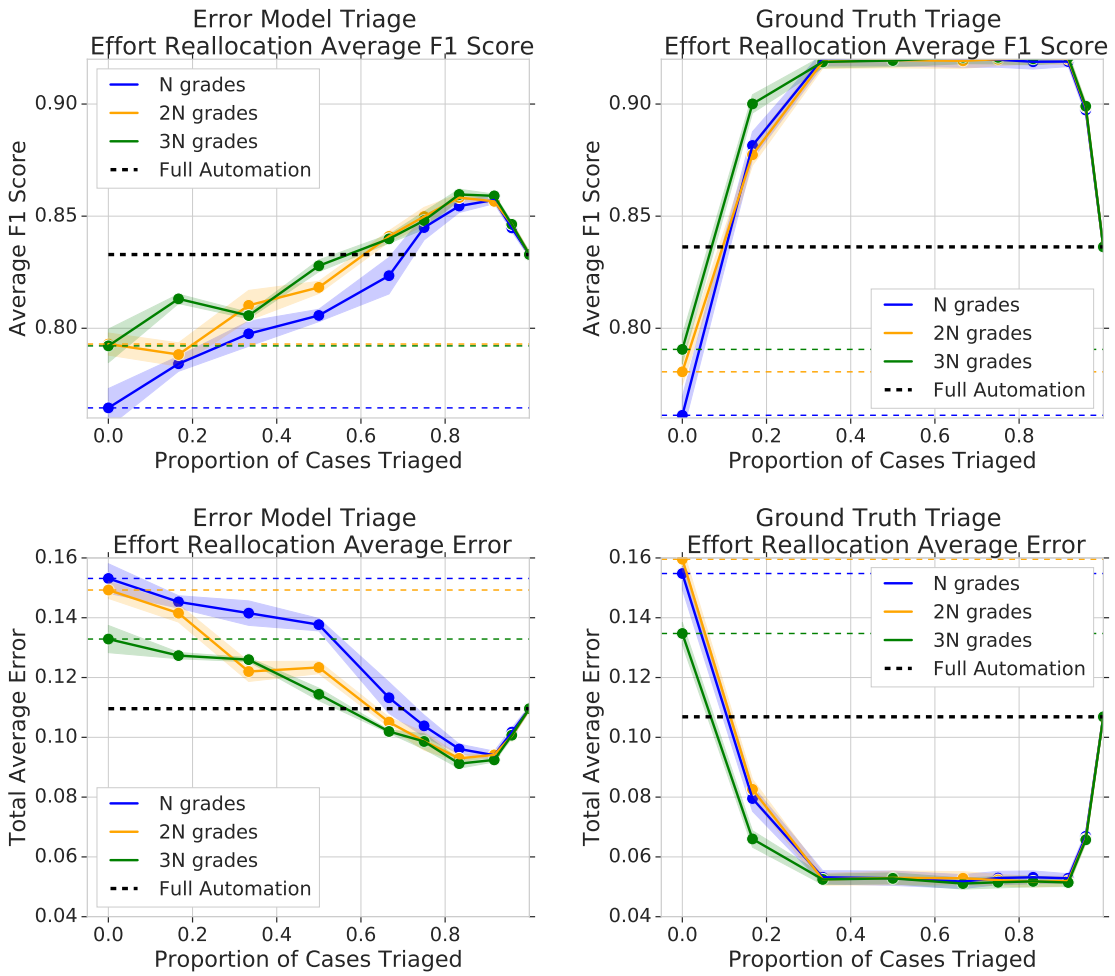



Figure App.5: **Triaging to combine human effort and algorithm decisions outperforms full automation and the equal coverage protocols, compare to Figure 8.4 in the main text.** We perform the same experiments as in Figure 8.4 in the main text, using the aggregated doctor grades as the ground truth instead of the adjudicated grade.

Finally, we also test to see if triaging can help find sets of zero error, like in Section F.5 and Figure 8.6. We find that this is indeed the case, though the fractions are slightly smaller with this holdout dataset, likely because the labels are noisier than on the adjudicated evaluation dataset.

We also see that the fraction of zero error examples triaged is slightly lower with the separate error prediction model (Figure App.6 right) than triaging by model uncertainty (Figure App.6 left). The reason for this becomes apparent after further inspection: the results of Figure App.6 are averaged over three inde-

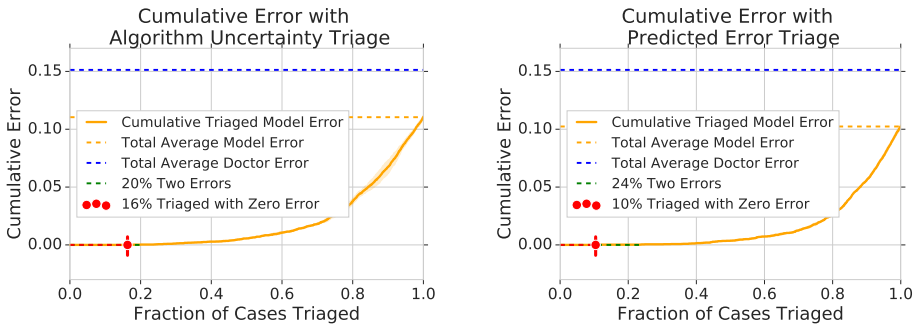



Figure App.6: **Proportion of data with zero errors when triaging.** Compare to Figure 8.6 in the main text. The proportion of the dataset found with zero errors is slightly lower, likely because the labels are noisier, and total error is much higher (10% compared to 4% in the main text.) Unlike the main text, we see that triaging by algorithmic uncertainty, left pane, seems to perform better than triaging with a separate model. Upon closer inspection, we find that this is because one repetition of the separate error model makes two errors earlier on, and accounting for this (green dotted lines) shows that the separate error model performs comparably/slightly better.

pendent repetitions of training a main diagnostic model, and a corresponding separate error model. We find that one of the three repetitions of the separate error model makes two errors – at 10% of the way through the data, it triages two examples that are errors.This causes the percentage with zero error to drop from 24% to 10%. If we account for these two errors, we see that in fact triaging by the error prediction model is doing comparably to triaging by algorithm uncertainty, where allowing two errors gets to 20% of the data.

## BIBLIOGRAPHY


[1] AAO. *International Clinical Diabetic Retinopathy Disease Severity Scale Detailed Table*. American Academy of Ophthalmology, 2002.

[2] Waleed Abdulla. Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow, 2018. https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46.

[3] Michael David Abràmoff, Yiyue Lou, Ali Erginay, Warren Clarida, Ryan Amelon, James C Folk, and Meindert Niemeijer. Improved automated detection of diabetic retinopathy on a publicly available dataset through integration of deep learning. *Investigative ophthalmology & visual science*, 57(13):5200–5206, 2016.

[4] Lisa S. Abrams, Ingrid U Scott, George L. Spaeth, Harry A. Quigley, and Rohit Varma. Agreement among optometrists, ophthalmologists, and residents in evaluating the optic disc for glaucoma. *Ophthalmology*, 101(10):1662–1667, 1994.

[5] Roee Aharoni, Melvin Johnson, and Orhan Firat. Massively multilingual neural machine translation. *arXiv preprint arXiv:1903.00089*, 2019.

[6] Hasseb Ahsan. Diabetic retinopathy – biomolecules and multiple pathophysiology. *Diabetes and Metabolic Syndrome: Clincal Research and Review*, pages 51–54, 2015.

[7] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[8] Jay Alammar. The Illustrated Transformer, 2018. http://jalammar.github.io/illustrated-transformer/.

[9] Mohsan Alvi, Andrew Zisserman, and Christoffer Nellåker. Turning a blind eye: Explicit removal of biases and variation from deep neural network embeddings. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[10] Marios Anthimopoulos, Stergios Christodoulidis, Lukas Ebner, Andreas Christe, and Stavroula Mougiakakou. Lung pattern classification for in-

terstitial lung diseases using a deep convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1207–1216, 2016.

[11] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. *arXiv preprint arXiv:1810.09502*, 2018.

[12] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 13(3):32:1–32:18, February 2017.

[13] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[14] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *arXiv preprint arXiv:1906.00910*, 2019.

[15] Marcus A Badgeley, John R Zech, Luke Oakden-Rayner, Benjamin S Glicksberg, Manway Liu, William Gale, Michael V McConnell, Bethany Percha, Thomas M Snyder, and Joel T Dudley. Deep learning predicts hip fracture using confounding patient and healthcare variables. *npj Digital Medicine*, 2(1):31, 2019.

[16] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[17] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[18] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949. IEEE, 2016.

[19] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. An unsupervised learning model for deformable medical image registration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9252–9260, 2018.

[20] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. Voxelmorph: a learning framework for deformable medical image registration. *IEEE transactions on medical imaging*, 2019.

[21] Yujia Bao, Menghua Wu, Shiyu Chang, and Regina Barzilay. Few-shot text classification with distributional signatures. *arXiv preprint arXiv:1908.06039*, 2019.

[22] Maurice S. Bartlett. The statistical significance of canonical correlations. In *Biometrika*, volume 32, pages 29 – 37, 1941.

[23] Joshua Batson and Loic Royer. Noise2self: Blind denoising by self-supervision. *arXiv preprint arXiv:1901.11365*, 2019.

[24] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[25] Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

[26] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6541–6549, 2017.

[27] Iz Beltagy, Arman Cohan, and Kyle Lo. Scibert: Pretrained contextualized embeddings for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[28] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019.

[29] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[30] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. Visualbackprop: visualizing cnns for autonomous driving. *arXiv preprint arXiv:1611.05418*, 2, 2016.

[31] Xavier Bresson and Thomas Laurent. A two-step graph convolutional decoder for molecule generation. *arXiv preprint arXiv:1906.03412*, 2019.

[32] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[33] Carrie J Cai, Emily Reif, Narayan Hegde, Jason Hipp, Been Kim, Daniel Smilkov, Martin Wattenberg, Fernanda Viegas, Greg S Corrado, Martin C Stumpe, et al. Human-centered tools for coping with imperfect algorithms during medical decision-making. *arXiv preprint arXiv:1902.02960*, 2019.

[34] Renzhi Cao, Colton Freitas, Leong Chan, Miao Sun, Haiqing Jiang, and Zhangxin Chen. Prolango: protein function prediction using neural machine translation based on a recurrent neural network. *Molecules*, 22(10):1732, 2017.

[35] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.

[36] Brandon Carter, Jonas Mueller, Siddhartha Jain, and David Gifford. What made you do this? understanding black-box decisions with sufficient input subsets. *arXiv preprint arXiv:1810.03805*, 2018.

[37] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. https://distill.pub/2019/activation-atlas.

[38] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5933–5942, 2019.

[39] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[40] Irene Chen, Fredrik D Johansson, and David Sontag. Why is my classifier discriminatory? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3543–3554. Curran Associates, Inc., 2018.

[41] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[42] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. On sampling strategies for neural network-based collaborative filtering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 767–776, 2017.

[43] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

[44] Yuhua Chen, Yibin Xie, Zhengwei Zhou, Feng Shi, Anthony G Christodoulou, and Debiao Li. Brain mri super resolution using 3d deep densely connected neural networks. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 739–742. IEEE, 2018.

[45] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

[46] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[47] Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370*, 2018.

[48] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

[49] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018.

[50] Peter Corbett and John Boyle. Improving the learning of chemical-protein interactions from literature using transfer learning and specialized word embeddings. *Database*, 2018, 2018.

[51] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaug-

ment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2019.

[52] Adrian Dalca, Marianne Rakic, John Guttag, and Mert Sabuncu. Learning conditional deformable templates with convolutional networks. In *Advances in neural information processing systems*, pages 804–816, 2019.

[53] Thomas M Daniel. *Toman's tuberculosis. Case detection, treatment and monitoring: questions and answers*. ASTMH, 2004.

[54] Yann Dauphin. mixup: Beyond Empirical Risk Minimization Image, 2017. https://www.dauphin.io/.

[55] P. Dawid, A. M. Skene, A. P. Dawidt, and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, pages 20–28, 1979.

[56] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[57] Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, et al. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature medicine*, 24(9):1342, 2018.

[58] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[59] Cem M Deniz, Siyuan Xiang, R Spencer Hallyburton, Arakua Welbeck, James S Babb, Stephen Honig, Kyunghyun Cho, and Gregory Chang. Segmentation of the proximal femur from mr images using deep convolutional neural networks. *Scientific reports*, 8(1):16485, 2018.

[60] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[61] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[62] Yiming Ding, Jae Ho Sohn, Michael G Kawczynski, Hari Trivedi, Roy Harnish, Nathaniel W Jenkins, Dmytro Lituiev, Timothy P Copeland, Mariam S Aboian, Carina Mari Aparici, et al. A deep learning model to predict a diagnosis of alzheimer disease by using 18f-fdg pet of the brain. *Radiology*, 290(2):456–464, 2018.

[63] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[64] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[65] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[66] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In *Advances in Neural Information Processing Systems*, pages 10541–10551, 2019.

[67] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[68] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in neural information processing systems*, pages 766–774, 2014.

[69] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1110–1118, 2015.

[70] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[71] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018.

[72] Joann G Elmore, Gary M Longton, Patricia A Carney, Berta M Geller, Tracy

Onega, Anna NA Tosteson, Heidi D Nelson, Margaret S Pepe, Kimberly H Allison, Stuart J Schnitt, et al. Diagnostic concordance among pathologists interpreting breast biopsy specimens. *Jama*, 313(11):1122–1132, 2015.

[73] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[74] Linjing Fang, Fred Monroe, Sammy Weiser Novak, Lyndsey Kirk, Cara R Schiavon, B Yu Seungyoon, Tong Zhang, Melissa Wu, Kyle Kastner, Yoshiyuki Kubota, et al. Deep learning-based point-scanning super-resolution imaging. *bioRxiv*, page 740548, 2019.

[75] Mikhail Figurnov, Aizhan Ibraimova, Dmitry P Vetrov, and Pushmeet Kohli. PerforatedCNNs: Acceleration through elimination of redundant convolutions. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 947–955. Curran Associates, Inc., 2016.

[76] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[77] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[78] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv preprint arXiv:1710.11622*, 2017.

[79] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

[80] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *arXiv preprint arXiv:2002.12880*, 2020.

[81] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.

[82] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[83] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

[84] Ferenc Galkó and Carsten Eickhoff. Biomedical question answering via weighted neural network passage retrieval. In *European Conference on Information Retrieval*, pages 523–528. Springer, 2018.

[85] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[86] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[87] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[88] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[89] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *ICLR*, 2019.

[90] Amirata Ghorbani, Vivek Natarajan, David Coz, and Yuan Liu. Dermgan: Synthetic generation of clinical skin images with pathology. *arXiv preprint arXiv:1911.08716*, 2019.

[91] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[92] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Pro-*

*ceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[93] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[94] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E Turner. Meta-Learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*, 2018.

[95] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

[96] Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235*, 2019.

[97] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

[98] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[99] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*, 2018.

[100] Melody Y. Guan, Varun Gulshan, Andrew M. Dai, and Geoffrey E. Hinton. Who said what: Modeling individual labelers improves classification, 2018.

[101] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[102] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu,

Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip Q Nelson, Jessica Mega, and Dale Webster. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316(22):2402–2410, 2016.

[103] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. abs/1706.04599, 2017.

[104] Danna Gurari, Kun He, Bo Xiong, Jianming Zhang, Mehrnoosh Sameki, Suyog Dutt Jain, Stan Sclaroff, Margrit Betke, and Kristen Grauman. Predicting foreground object ambiguity and efficiently crowdsourcing the segmentation (s). *International Journal of Computer Vision*, 126(7):714–730, 2018.

[105] Maryam Habibi, Leon Weber, Mariana Neves, David Luis Wiegandt, and Ulf Leser. Deep learning with word embeddings improves biomedical named entity recognition. *Bioinformatics*, 33(14):i37–i48, 2017.

[106] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[107] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*, pages 8527–8537, 2018.

[108] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the 4th International Conference on Learning Representations (ICLR'16)*, October 2015.

[109] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[110] Ankur Handa, Michael Bloesch, Viorica Pătrăucean, Simon Stent, John McCormac, and Andrew Davison. gvnn: Neural network library for geometric computer vision. In *European Conference on Computer Vision*, pages 67–82. Springer, 2016.

[111] Boris Hanin. Which neural net architectures give rise to exploding and

vanishing gradients? In *Advances in Neural Information Processing Systems*, pages 582–591, 2018.

[112] Jack Hanson, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Improving protein disorder prediction by deep bidirectional long short-term memory recurrent neural networks. *Bioinformatics*, 33(5):685–692, 2016.

[113] Frank E Harrell Jr, Kerry L Lee, Robert M Califf, David B Pryor, and Robert A Rosati. Regression modelling strategies for improved prognostic prediction. *Statistics in medicine*, 3(2):143–152, 1984.

[114] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. *arXiv preprint arXiv:1807.08912*, 2018.

[115] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*, 2018.

[116] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[117] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[118] Rhys Heffernan, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017.

[119] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[120] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019.

[121] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espe-

holt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

[122] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[123] Daniel Ho, Eric Liang, Ion Stoica, Pieter Abbeel, and Xi Chen. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393*, 2019.

[124] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

[125] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[126] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[127] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics, 2011.

[128] Harold Hotelling. Relations between two sets of variates. In *Biometrika*, volume 28, pages 321–337, 1936.

[129] Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics*, pages 162–190. Springer, 1992.

[130] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. *arXiv preprint arXiv:1902.00751*, 2019.

[131] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

[132] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[133] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

[134] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[135] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[136] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[137] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[138] Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn Ball, Katie Shpanskaya, et al. CheXpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[139] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[140] Khurram Javed and Martha White. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588*, 2019.

[141] Na Ji. Adaptive optical fluorescence microscopy. *Nature methods*, 14(4):374, 2017.

[142] Robin Jia and Percy Liang. Data recombination for neural semantic parsing. *arXiv preprint arXiv:1606.03622*, 2016.

[143] Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

[144] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[145] MF Kasim, D Watson-Parris, L Deaconu, S Oliver, P Hatfield, DH Froula, G Gregori, M Jarvis, S Khatiwala, J Korenaga, et al. Up to two billion times acceleration of scientific simulations with deep neural architecture search. *arXiv preprint arXiv:2001.08055*, 2020.

[146] Jeremy Kawahara, Sara Daneshvar, Giuseppe Argenziano, and Ghassan Hamarneh. Seven-point checklist and skin lesion classification using multitask multimodal neural nets. *IEEE journal of biomedical and health informatics*, 23(2):538–546, 2018.

[147] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[148] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, volume 30, pages 5580–5590, 2017.

[149] Pegah Khosravi, Ehsan Kazemi, Qiansheng Zhan, Marco Toschi, Jonas E Malmsten, Cristina Hickman, Marcos Meseguer, Zev Rosenwaks, Olivier Elemento, Nikica Zaninovic, et al. Robust Automated Assessment of Human Blastocyst Quality using Deep Learning. *bioRxiv*, page 394882, 2018.

[150] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

[151] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 267–280. Springer, 2019.

[152] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

[153] D Kingma, Tim Salimans, R Josefowicz, Xi Chen, Ilya Sutskever, Max Welling, et al. Improving variational autoencoders with inverse autoregressive flow. 2017.

[154] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[155] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[156] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[157] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201*, 2018.

[158] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[159] Simon AA Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R Ledsam, Klaus H Maier-Hein, SM Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images. *arXiv preprint arXiv:1806.05034*, 2018.

[160] Kaname Kojima, Shu Tadaka, Fumiki Katsuoka, Gen Tamiya, Masayuki Yamamoto, and Kengo Kinoshita. A recurrent neural network based method for genotype imputation on phased genotype data. *bioRxiv*, page 821504, 2019.

[161] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. *arXiv preprint arXiv:1901.09005*, 2019.

[162] Patrick T Komiske, Eric M Metodiev, and Jesse Thaler. Energy flow networks: deep sets for particle jets. *Journal of High Energy Physics*, 2019(1):121, 2019.

[163] Shu Kong and Charless Fowlkes. Image reconstruction with predictive filter flow. *arXiv preprint arXiv:1811.11482*, 2018.

[164] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[165] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974*, 2018.

[166] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.

[167] Jonathan Krause, Varun Gulshan, Ehsan Rahimy, Peter Karth, Kasumi Widner, Gregory S. Corrado, Lily Peng, and Dale R. Webster. Grader variability and the importance of reference standards for evaluating machine learning models for diabetic retinopathy. *Ophthalmology*, 125 8:1264–1272, 2018.

[168] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[169] Sneha Reddy Kudugunta, Ankur Bapna, Isaac Caswell, Naveen Arivazhagan, and Orhan Firat. Investigating multilingual nmt representations at scale. *arXiv preprint arXiv:1909.02197*, 2019.

[170] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[171] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In D S Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[172] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013.

[173] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations (ICLR'17)*, 2018.

[174] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: pre-trained biomedical language representation model for biomedical text mining. *arXiv preprint arXiv:1901.08746*, 2019.

[175] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[176] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. *arXiv preprint arXiv:1801.05558*, 2018.

[177] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.

[178] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[179] Chaolong Li, Zhen Cui, Wenming Zheng, Chunyan Xu, and Jian Yang. Spatio-temporal graph convolution for skeleton based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[180] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, April 2018.

[181] Hailiang Li, Jian Weng, Yujian Shi, Wanrong Gu, Yijun Mao, Yonghua Wang, Weiwei Liu, and Jiajie Zhang. An improved deep learning approach for detection of thyroid papillary cancer in ultrasound images. *Scientific reports*, 8(1):6600, 2018.

[182] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *International Conference on Learning Representations (ICLR'17)*, pages 1–10, 2017.

[183] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *FE@ NIPS*, pages 196–212, 2015.

[184] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *Iclr*, 2016.

[185] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[186] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[187] Fang Liu, Zhaoye Zhou, Hyungseok Jang, Alexey Samsonov, Gengyan Zhao, and Richard Kijowski. Deep convolutional neural network and 3d deformable approach for tissue segmentation in musculoskeletal magnetic resonance imaging. *Magnetic resonance in medicine*, 79(4):2379–2391, 2018.

[188] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[189] Shengyu Liu, Buzhou Tang, Qingcai Chen, and Xiaolong Wang. Effects of semantic features on machine learning-based drug name recognition systems: word embeddings vs. manually constructed dictionaries. *Information*, 6(4):848–865, 2015.

[190] Xueliang Liu. Deep recurrent neural network for protein function prediction from sequence. *arXiv preprint arXiv:1701.08318*, 2017.

[191] Yao Liu, Omer Gottesman, Aniruddh Raghu, Matthieu Komorowski, Aldo A Faisal, Finale Doshi-Velez, and Emma Brunskill. Representation balancing mdps for off-policy policy evaluation. In *Advances in Neural Information Processing Systems*, pages 2644–2653, 2018.

[192] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *arXiv preprint arXiv:1904.08473*, 2019.

[193] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*, 2017.

[194] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[195] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2208–2217. JMLR. org, 2017.

[196] Romain Lopez, Jeffrey Regier, Michael Cole, Michael Jordan, and Nir Yosef. A deep generative model for gene expression profiles from single-cell rna sequencing. *arXiv preprint arXiv:1709.02082*, 2017.

[197] Donghuan Lu, Karteek Popuri, Gavin Weiguang Ding, Rakesh Balachandar, and Mirza Faisal Beg. Multimodal and multiscale deep neural networks for the early diagnosis of alzheimer's disease using structural mr and fdg-pet images. *Scientific reports*, 8(1):5697, 2018.

[198] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[199] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[200] Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R Eguchi, Possu Huang, and Richard Socher. Progen: Language modeling for protein generation. *bioRxiv*, 2020.

[201] Martin Magill, Faisal Qureshi, and Hendrick de Haan. Neural networks trained to solve differential equations learn general representations. In *Advances in Neural Information Processing Systems*, pages 4075–4085, 2018.

[202] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 181–196, 2018.

[203] Niru Maheswaranathan, Alex H. Willams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Universality and individuality in neural dynamics across large populations of recurrent networks. *arXiv preprint arXiv:1907.08549*, 2019.

[204] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani, Songtao He, et al. Park: An open platform for learning augmented computer systems. 2019.

[205] Daniel L Marino, Kasun Amarasinghe, and Milos Manic. Building energy load forecasting using deep neural networks. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 7046–7051. IEEE, 2016.

[206] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281, 2018.

[207] Mackenzie Weygandt Mathis and Alexander Mathis. Deep learning tools for the measurement of animal behavior in neuroscience. *Current Opinion in Neurobiology*, 60:1–11, 2020.

[208] AGDG Matthews, J Hron, M Rowland, RE Turner, and Z Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations (ICLR'18)*, 2018.

[209] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[210] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182*, 2017.

[211] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*, 2018.

[212] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[213] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[214] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[215] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations, 2019.

[216] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[217] Volodymyr Mnih and Geoffrey Hinton. Learning to label aerial images from noisy data. *International Conference on Machine Learning*, 2012.

[218] Pim Moeskops, Max A Viergever, Adriënne M Mendrik, Linda S de Vries, Manon JNL Benders, and Ivana Išgum. Automatic segmentation of mr brain images with a convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1252–1261, 2016.

[219] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR'17)*, November 2016.

[220] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, pages 5727–5736, 2018.

[221] Ari S. Morcos, David G.T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *International Conference on Learning Representations (ICLR'18)*, 2018.

[222] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems 26*, pages 1196–1204. 2013.

[223] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[224] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V Le, and Ruoming Pang. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, 2018.

[225] Alex Nichol and John Schulman. Reptile: A scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

[226] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

[227] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[228] Luke Oakden-Rayner, Jared Dunnmon, Gustavo Carneiro, and Christopher Ré. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. *arXiv preprint arXiv:1909.12475*, 2019.

[229] Chris Olah. Understanding LSTM Networks, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[230] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[231] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[232] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[233] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[234] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[235] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[236] F Pasa, V Golkov, F Pfeiffer, D Cremers, and D Pfeiffer. Efficient Deep Network Architectures for Fast Chest X-Ray Tuberculosis Screening and Visualization. *Scientific reports*, 9(1):6268, 2019.

[237] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2, 2012.

[238] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1796–1804, 2015.

[239] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[240] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[241] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[242] Gianluca Pollastri, Darisz Przybylski, Burkhard Rost, and Pierre Baldi. Improving the prediction of protein secondary structure in three and eight

classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.

[243] Ryan Poplin, Avinash V Varadarajan, Katy Blumer, Yun Liu, Michael V McConnell, Greg S Corrado, Lily Peng, and Dale R Webster. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering*, 2(3):158, 2018.

[244] Rory M Power and Jan Huisken. A guide to light-sheet fluorescence microscopy for multiscale imaging. *Nature methods*, 14(4):360, 2017.

[245] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[246] Siyuan Qiao, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan Yuille. Deep co-training for semi-supervised image recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 135–152, 2018.

[247] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[248] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[249] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. Continuous state-space models for optimal sepsis treatment-a deep reinforcement learning approach. *arXiv preprint arXiv:1705.08422*, 2017.

[250] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019.

[251] Maithra Raghu, Katy Blumer, Greg Corrado, Jon Kleinberg, Ziad Obermeyer, and Sendhil Mullainathan. The algorithmic automation problem: Prediction, triage, and human effort. *arXiv preprint arXiv:1903.12220*, 2019.

[252] Maithra Raghu, Katy Blumer, Rory Sayres, Ziad Obermeyer, Sendhil Mul-

lainathan, and Jon Kleinberg. Direct uncertainty prediction with applications to healthcare. *arXiv preprint arXiv:1807.01771*, 2018.

[253] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pages 6076–6085, 2017.

[254] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In *Advances in Neural Information Processing Systems*, pages 3342–3352, 2019.

[255] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[256] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[257] Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*, 2015.

[258] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

[259] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[260] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

[261] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019.

[262] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[263] Scott E. Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. abs/1412.6596, 2014.

[264] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[265] Donatas Repecka, Vykintas Jauniskis, Laurynas Karpus, Elzbieta Rembeza, Jan Zrimec, Simona Poviloniene, Irmantas Rokaitis, Audrius Laurynenas, Wissam Abuajwa, Otto Savolainen, et al. Expanding functional protein sequence space using generative adversarial networks. *bioRxiv*, page 789719, 2019.

[266] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[267] Alexander Rives, Siddharth Goyal, Joshua Meier, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, page 622803, 2019.

[268] David Rolnick, Andreas Veit, Serge J. Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *CoRR*, abs/1705.10694, 2017.

[269] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[270] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[271] Olga Russakovsky and Li Fei-Fei. Attribute learning in large-scale datasets. In *European Conference of Computer Vision (ECCV), International Workshop on Parts and Attributes*, 2010.

[272] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[273] Andrei A Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[274] Ruhan Sa, William Owens, Raymond Wiegand, Mark Studin, Donald Capoferri, Kenneth Barooha, Alexander Greaux, Robert Rattray, Adam Hutton, John Cintineo, et al. Intervertebral disc detection in x-ray images using faster r-cnn. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 564–567. IEEE, 2017.

[275] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[276] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[277] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, pages 1842–1850, 2016.

[278] Naomi Saphra and Adam Lopez. Understanding learning dynamics of language models with SVCCA. *arXiv preprint arXiv:1811.00225*, 2018.

[279] Saman Sarraf, Ghassem Tofighi, et al. Deepad: Alzheimer disease classification via deep convolutional neural networks using mri and fmri. *BioRxiv*, page 070441, 2016.

[280] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[281] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[282] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson,

Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, pages 1–5, 2020.

[283] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[284] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[285] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 614–622. ACM, 2008.

[286] Jianghong Shi, Eric Shea-Brown, and Michael Buice. Comparison against task driven artificial neural networks reveals functional properties in mouse visual cortex. In *Advances in Neural Information Processing Systems*, pages 5765–5775, 2019.

[287] Susan M Shortreed, Eric Laber, Daniel J Lizotte, T Scott Stroup, Joelle Pineau, and Susan A Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine learning*, 84(1-2):109–136, 2011.

[288] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[289] Rui Shu, Hung H Bui, Hirokazu Narui, and Stefano Ermon. A dirt-t approach to unsupervised domain adaptation. *arXiv preprint arXiv:1802.08735*, 2018.

[290] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[291] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[292] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[293] Tensorflow Slim. Tensorflow slim inception-v3. https://github.com/tensorflow/models/tree/master/research/slim, 2017.

[294] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[295] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[296] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[297] Youyi Song, Ling Zhang, Siping Chen, Dong Ni, Baopu Li, Yongjing Zhou, Baiying Lei, and Tianfu Wang. A deep learning based framework for accurate segmentation of cervical cytoplasm and nuclei. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2903–2906. IEEE, 2014.

[298] Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, pages 72–101, 1904.

[299] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. Training convolutional networks with noisy labels. *CoRR*, abs/1406.2080, 2014.

[300] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5693–5703, 2019.

[301] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*, 2019.

[302] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution

for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[303] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[304] I Sutskever, O Vinyals, and QV Le. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

[305] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[306] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[307] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[308] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[309] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070*, 2019.

[310] Ryutaro Tanno, Daniel Worrall, Aurobrata Ghosh, Enrico Kaden, Stamatios N. Sotiropoulos, Antonio Criminisi, and Daniel C. Alexander. Bayesian image quality transfer with cnns: Exploring uncertainty in dmri super-resolution. *Medical Image Computing and Computer Assisted Intervention*, pages 611–619, 2017.

[311] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*, pages 1195–1204, 2017.

[312] Dimitry Tegunov and Patrick Cramer. Real-time cryo-em data pre-processing with warp. *BioRxiv*, page 338558, 2018.

[313] Arthur Tenenhaus and Michel Tenenhaus. Regularized generalized canonical correlation analysis. *Psychometrika*, 76(2):257, 2011.

[314] Yee Liang Thian, Yiting Li, Pooja Jagmohan, David Sia, Vincent Ern Yao Chan, and Robby T Tan. Convolutional neural networks for automated fracture detection and localization on wrist radiographs. *Radiology: Artificial Intelligence*, 1(1):e180001, 2019.

[315] Eric Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25:44–56, 2019.

[316] Eric J Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44–56, 2019.

[317] Raphael Townshend, Rishi Bedi, Patricia Suriana, and Ron Dror. End-to-end learning on 3d protein structure for interface prediction. In *Advances in Neural Information Processing Systems*, pages 15616–15625, 2019.

[318] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-Dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[319] Vahe Tshitoyan, John Dagdelen, Leigh Weston, Alexander Dunn, Ziqin Rong, Olga Kononova, Kristin A Persson, Gerbrand Ceder, and Anubhav Jain. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, 571(7763):95–98, 2019.

[320] Kensuke Umehara, Junko Ota, and Takayuki Ishida. Application of super-resolution convolutional neural network for enhancing image resolution in chest ct. *Journal of digital imaging*, 31(4):441–450, 2018.

[321] Viivi Uurtio, João M. Monteiro, Jaz Kandola, John Shawe-Taylor, Delmiro Fernandez-Reyes, and Juho Rousu. A tutorial on canonical correlation methods. *ACM Comput. Surv.*, 50(6):95:1–95:33, November 2017.

[322] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[323] Amber A Van Der Heijden, Michael D Abramoff, Frank Verbraak, Manon V van Hecke, Albert Liem, and Giel Nijpels. Validation of automated screening for referable diabetic retinopathy with the idx-dr device in the hoorn diabetes care system. *Acta ophthalmologica*, 96(1):63–68, 2018.

[324] Monica Van Such, Robert Lohr, Thomas Beckman, and James M Naessens. Extent of diagnostic agreement among medical referrals. *Journal of evaluation in clinical practice*, 23(4):870–874, 2017.

[325] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[326] Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge J. Belongie. Learning from noisy large-scale datasets with minimal supervision. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6575–6583, 2017.

[327] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching Networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[328] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[329] Elena Voita, Rico Sennrich, and Ivan Titov. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. *arXiv preprint arXiv:1909.01380*, 2019.

[330] Christian Wachinger, Martin Reuter, and Tassilo Klein. Deepnat: Deep convolutional neural network for segmenting neuroanatomy. *NeuroImage*, 170:434–445, 2018.

[331] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3266–3280, 2019.

[332] Kun Wang, Bite Yang, Guohai Xu, and Xiaofeng He. Medical question retrieval based on siamese neural network and transfer learning method. In *International Conference on Database Systems for Advanced Applications*, pages 49–64. Springer, 2019.

[333] Nancy XR Wang, Ali Farhadi, Rajesh PN Rao, and Bingni W Brunton. Ajile movement prediction: Multimodal deep learning for natural human neural recordings and video. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[334] William Yang Wang and Diyi Yang. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using# petpeeve tweets. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2557–2563, 2015.

[335] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[336] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3462–3471. IEEE, 2017.

[337] Zeyu Wang, Klint Qinami, Yannis Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. *arXiv preprint arXiv:1911.11834*, 2019.

[338] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[339] Fabian L Wauthier and Michael I. Jordan. Bayesian bias mitigation for crowdsourcing. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1800–1808. 2011.

[340] Jason W Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[341] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[342] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature methods*, 15(12):1090, 2018.

[343] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*, 2015.

[344] Peter Welinder and Pietro Perona. Online crowdsourcing: Rating annotators and obtaining cost-effective labels. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2010, San Francisco, CA, USA, 13-18 June, 2010*, pages 25–32, 2010.

[345] Keenon Werling, Arun Tejasvi Chaganty, Percy S Liang, and Christopher D Manning. On-the-job learning with bayesian decision theory. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3465–3473. 2015.

[346] Julia K Winkler, Christine Fink, Ferdinand Toberer, Alexander Enk, Teresa Deinlein, Rainer Hofmann-Wellenhof, Luc Thomas, Aimilios Lallas, Andreas Blum, Wilhelm Stolz, et al. Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition. *JAMA dermatology*, 155(10):1135–1141, 2019.

[347] Svante Wold, Michael Sjöström, and Lennart Eriksson. Pls-regression: a basic tool of chemometrics. *Chemometrics and intelligent laboratory systems*, 58(2):109–130, 2001.

[348] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[349] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[350] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[351] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2691–2699, 2015.

[352] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

[353] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

[354] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[355] Jun Xu, Xiaofei Luo, Guanhao Wang, Hannah Gilmore, and Anant Madabhushi. A deep convolutional neural network for segmenting and classifying epithelial and stromal regions in histopathological images. *Neurocomputing*, 191:214–223, 2016.

[356] Xueting Yan, Ishan Misra, Abhinav Gupta, Deepti Ghadiyaram, and Dhruv Mahajan. Clusterfit: Improving generalization of visual representations. *arXiv preprint arXiv:1912.03330*, 2019.

[357] Yilong Yang, Zhuyifan Ye, Yan Su, Qianqian Zhao, Xiaoshan Li, and Defang Ouyang. Deep learning for in vitro prediction of pharmaceutical formulations. *Acta pharmaceutica sinica B*, 9(1):177–185, 2019.

[358] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[359] Koichiro Yasaka, Hiroyuki Akai, Osamu Abe, and Shigeru Kiryu. Deep learning with convolutional neural network for differentiation of liver masses at dynamic contrast-enhanced ct: a preliminary study. *Radiology*, 286(3):887–896, 2017.

[360] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[361] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[362] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065*, 2019.

[363] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6023–6032, 2019.

[364] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[365] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[366] Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753–1762, 2015.

[367] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. *arXiv preprint arXiv:1905.03670*, 2019.

[368] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. The visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[369] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *Unpublished draft. Retrieved*, 3:319, 2019.

[370] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR'16)*, abs/1611.03530, 2016.

[371] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

[372] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[373] Junkang Zhang, Haigen Hu, Shengyong Chen, Yujiao Huang, and Qiu Guan. Cancer cells detection in phase-contrast microscopy images based on faster r-cnn. In *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 363–367. IEEE, 2016.

[374] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2472–2481, 2018.

[375] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[376] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[377] Fengwei Zhou, Bin Wu, and Zhenguo Li. Deep meta-learning: learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*, 2018.

[378] Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. Neural document summarization by jointly learning to score and select sentences. *arXiv preprint arXiv:1807.02305*, 2018.

[379] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[380] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.

[381] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and

Shimon Whiteson. Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.

[382] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.