TOWARDS EFFICIENT INFERENCE AND IMPROVED TRAINING EFFICIENCY OF DEEP NEURAL NETWORKS

by

Ravi Raju

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2022

Date of Final Oral Examination: 08/17/2022

The dissertation is approved by the following members of the Final Oral Committee:
Mikko Lipasti, Professor, Electrical and Computer Engineering
Yu Hen Hu, Professor, Electrical and Computer Engineering
Kassem Fawaz, Assistant Professor, Electrical and Computer Engineering
Earlence Fernandes, Assistant Professor, Computer Science

Dedication

To Amma, Appa, and Vidhya Akka, for their love, support, wisdom, and compassion.

Declaration

I declare that for the conditional execution work, I collaborated with Dibakar Gope, Urmish Thakkar, and Jesse Beu. My contribution of the work was: 1) writing the code for integrating the FBS work with CondConv as well as writing the paper. Dibakar suggested the initial intuition of expert filters allowing pruning methods to be more aggressive. Urmish suggested the static channel pruning baseline. Jesse suggested the experiment exposing the filter under-utilization of conditional filters.

I declare that for the dynamic data pruning work, I collaborated with Kyle Daruwalla. My contributions for this work was: 1) performing the literature review for methods accelerating training networks, 2) creating the dynamic data pruning framework and applying techniques from active learning as selection criteria, 3) creating the categorization of different samples in the dataset and 4) coding the implementation of the method. Kyle's contributions are as follows: 1) suggesting experiments to explore the efficacy of random data pruning, 2) integrating reinforcement learning techniques to vary the sample selection process, and 3) proposing the experiments on imbalanced and downsampled CIFAR10 datasets.

Acknowledgements

Upon the completion of this document and journey, it feels in some sense surreal that I have actually come this far and am going to complete this degree. I remember a while back when I was still just a master's student, I watched a Youtube video about a lecture on tensor decomposition and I thought it would be intellectually fulfilling to pursue higher learning and research. Funnily enough, I didn't do anything related to that topic throughout my graduate years but I have learned so much about technical topics and perhaps more importantly, professional and personal relationships. Below, I want to enumerate a long list of individuals to thank for my development through the PhD process.

First off, I want to thank my parents whose support and guidance has been an integral part of me being to complete this PhD. I remember the days when my mother and father would frantically worry if I would become successful and made me work hard (harder than I wanted to at the time). Their efforts distilled in a work ethic that geared me for success and the challenges that come along with this degree. I hope that this PhD can make their hearts swell with pride. I also want to give thanks to my sister, Vidhya, for keeping in touch despite the busy life she leads working and maintaining a family. Our conversations about various subjects allowed me to destress and feel light-hearted, which sustained my graduate journey. Thank you for being there from the start and believing that I could do it.

Next, I want to express my heartfelt gratitude to my advisor, Professor Mikko Lipasti. I am very grateful to have been mentored by Mikko. He has an incredibly methodological approach to research, both from a macroscopic and a microscopic perspective. In fact, it

was Mikko's suggestion of looking at active learning methods to facilitate sample selection is what lead to developing dynamic data pruning. His attention to detail and ability to ask the right questions and challenge me pushed me to think more deeply about the problems I was working on. Mikko is an extremely likeable person and an absolute pleasure to work with. I hope to carry the mindset Mikko instilled in me into my future work and projects.

I also want to thank my other PhD Committee members for their guidance through the PhD process. I want to thank Prof. Kassem Fawaz who discussed with me what topics I should pursue at the aftermath of my preliminary exam. This is one of many conversation that pushed my research towards improving computational efficiency of deep learning models rather than on adversarial robustness. I also want to thank Prof. Earlence Fernandes for meeting with me thorough my exploration of deep learning topics. His work on RP2 attacks on stop sign images inspired the first project, BlurNet, I worked on. I also want to thank Prof. Yu Hen Hu for serving on the committee. Prof. Hu's ECE552 class was the first class I took at UW-Madison and I am grateful that he taught the class. I remember meeting Prof. Hu as a confused grad student looking for research and he encouraged me to focus on doing well in coursework and worrying about research later. This advice helped me knock out requirements early on so that I could focus on research in the years to come.

Next, I want to thank all of the members of PHARM Research group. It was really such a pleasure to be around these individuals. I want to thank all my seniors in the group - Rohit Shukla, Gokul Subramanian Ravi, Micheal Mishkin and David Schlais. They were always willing to entertain my stupid questions and guide me throughout their time in graduate school. They also encouraged me to connect with other members of the computer/machine learning community; in fact, it was Gokul who suggested I talked to Urmish Thakkar, which lead to my first internship at ARM. Next, I want to thank my fellow peers, Kyle Daruwalla, Heng Zhuo, Carly Schulz, Chein-Fu Chen, Soroosh Khoram, and Ranganathan (Bujji) Selgamsetty. Whether it was going to lunch, debating the finer aspects of technical subjects or going to the gym, these individuals created a space which allowed me to express my ideas

and I have learned so much from them; I will cherish these memories as I move forward in my life.

I want to give a brief thanks to those people that I meet during my two summers I spent at ARM. Not only did I learn about machine learning, but I also learned techniques that made me an effective communicator and drastically improved my technical presentations. I want to thank Urmish Thakkar, who I met at NeurIPS 2019, and referred me for an intern position at the ARM ML research team. It was also Urmish who referred me to SambaNova Systems for an applied research full-time position. I am extremely grateful to him as he has been instrumental in helping me with career opportunities. I want to thank former PHARM member Dibakar Gope who worked closely with me during both internship summers. I will never forget the zoom calls where we go through the FBS-CondConv code with a fine-grained brush to uncover the bug we were trying to resolve. I also want to thank Shounak Dutta who partially mentored me on a project concerning vision transformers (ViTs) - his guidance was one of the motivating factors in exploring patch pruning to accelerate ViTs. I also want to thank fellow intern and friend, Xuequin (Annie) Huang, for all the thoughtful discussions and fun times playing mobile League games. I want to give a special thanks to my manager and mentor, Jesse Beu. Jesse is one of the best managers I've ever had. His attitude towards research about being detail-oriented and taking babysteps/failing fast are behaviors I will carry forward with me. I remember during an exit interview Jesse gave me some feedback about being more detail oriented in research and he demonstrated this through an exercise: you needed to guess the number in the real domain and you could ask as many questions but you were allowed only one guess. The point of the exercise was to eliminate possible guesses to narrow in on the right answer. I will always remember what a learning experience that was and will apply that mindset in my future work.

I also want to thank the ECE GSA community for hosting events to bring the ECE department together. I will keep the experiences of Friendsgiving, going to the movies, hosting happy hours and other events close to my heart. There are a number of people

and friends who have supported me during my PhD endeavours and I am thankful for their support. I also want to thank my undergraduate advisor, Jeorg Mossbrucker, who suggested to me that I try my hand at a PhD. I also want to thank the UW ECE Department's IT support, espeically David Hanke, for answering my requests when the machines weren't working. I also want to thank the UW Madison Center for High Throughput Computing (CHTC), which provided GPU servers that allowed me to run the dynamic data pruning work. UW's infrastructure has undoubtedly accelerated my research progress and allowed me to run experiments at scale which otherwise would not be possible.

Finally, the last person I want to thank is my girlfriend, Aparna-Vaishnavi Narayan. I met her near the end of my time in graduate school but her presence has greatly offset my stress and has brought much joy to my life. She has allowed me a space to not talk about work and has made me realize that there's much more to life than worrying about the outcome of some experiments or what some reviewer's opinion is. Our daily WhatsApp calls and jokes are one of the contributing factors which has pushed me over the finish-line for this PhD.

Table of Contents

List of Figures

List of Tables

Abstract

In recent years, deep neural networks have surpassed human performance on image classification tasks and and speech recognition. While current models can reach state of the art performance on stand-alone benchmarks, deploying them on embedded systems that have real-time latency deadlines either cause them to fail these requirements or severely get degraded in performance to meet the stated specifications. This requires intelligent design of the network architecture in order to minimize the accuracy degradation while deployed on the edge. Similarly, deep learning often has a long turn-around time due to the volume of the experiments on different hyperparameters and consumes time and resources. This motivates a need for developing training strategies that allow researchers who do not have access to large computational resources to train large models without waiting for exorbitant training cycles to be completed.

This dissertation addresses these concerns through data dependent pruning of deep learning computation. First, regarding inference, we propose an integration of two different conditional execution strategies we call **FBS-pruned CondConv** by noticing that if we use input-specific filters instead of standard convolutional filters, we can aggressively prune at higher rates and mitigate accuracy degradation for significant computation savings. Then, regarding long training times, we introduce our **dynamic data pruning** framework which takes ideas from active learning and reinforcement learning to dynamically select subsets of data to train the model. Finally, as opposed to pruning data and in the same spirit of reducing training time, we investigate the vision transformer and introduce a unique training method called **PatchDrop** (originally designed for robustness to occlusions on transformers [1]), which uses the self-supervised DINO [2] model to identify the salient patches in an image and train on the salient subsets of an image. These strategies/training methods take a step in a direction to make models more accessible to deploy on edge devices in an efficient inference context and reduces the barrier for the independent researcher to train deep learning models which would require immense computational resources, pushing towards the democratization of machine learning.

Chapter 1

Introduction

Over the recent years, deep learning (DL) has permeated a large variety of application domains such as image classification, object detection, natural language processing, question answering, etc. [3–5]. A feature of these types of machine learning (ML) models is as more parameters are added into the model via adding more layers, their generalization, that is the gap between loss incurred on the training data and a hold out validation set, improves dramatically. This particular trend has caused deep learning to explode in parameter count and memory size both in computer vision but especially in natural language processing. Advances in computer architecture such as specialized accelerators like the tensor processing unit (TPU) and parallelization schemes like data parallelism, model parallelism, and tensor parallelism [6] has allowed large language models (LLMs) to scale up to billions or even trillions of parameters across thousands of compute devices with high throughput [7]. The emergence of these LLMs has allowed new paradigms of DL to be explored such as few-shot learning and zero-shot learning which aligns more closely with how a human might approach a new task.

However, this line of research only addresses one subfield of ML that operates in an environment where constraints like latency or real-time deadlines are not a concern. With the Internet of Things (IoT) devices becoming more prevalent, there is a push to move

computer-vision based models from the cloud onto the device itself to meet user specifications. This subfield of DL is sometimes called efficient inference by attempting to meet device requirements such as memory storage, latency, and power consumption while minimizing the penalty to validation accuracy of the DL. Current approaches in efficient inference include: weight magnitude pruning, quantization, tensor decomposition, and neural architecture search. As our world becomes more integrated with devices communicating with each other, it is paramount that current DL models can keep up with consumer/business demands for more accurate, faster, and efficient devices.

As mentioned before, this view of deep learning of scaling up via model parameters and data is valuable as it demonstrates that models can learn to do tasks they weren't explicitly trained on. However, this type of research can only be conducted by large industry labs or academic institutions that have access to the resources to train these models. In comparison with the prior problem with deploying large models to embedded devices, this issue has to do with how to efficiently perform training a DL model. The training costs to obtain a trained DL model are often prohibitive; it is common, standard practice to run hyperparameter parameter search, the process of doing multiple experimental runs with different input parameters to maximize the generalization of the model. One can even argue in the case of LLMs this is still a problem even given a massive amount of compute resources: on one of the tasks GPT3 was evaluated against, some of the validation data was leaked to the training data, which lead to an inflated zero-shot performance, but the researchers could not retrain the model because it would have exceed their cost budget. For the average DL researcher or practitioner, reducing the turn-around time for testing new ideas on their model and data would dramatically reduce their costs in terms of time spent waiting for experiments to complete as well as dollars if they are using a service such as Google Cloud and Amazon Web Services. If democratization of DL to be a goal that the community has in mind, developing methods to empower newcomers to train DL models with fewer compute resources is essential.

This dissertation takes a step in addressing these problems through data dependent pruning of deep learning computation. We delineate our approach into two distinct categories: efficient inference and efficient training. This chapter delineates our approach to each one of these problems by first addressing efficient inference through conditional execution (Section 1.1). We then describe dynamic data pruning to accelerate DL training by periodically pruning data to alleviate waiting for long training cycles (Section 1.2). We further expand on pruning framework by applying it to the vision transformer (ViT) [8] architecture to accelerate training through a different axis than pruning samples (Section 1.3). We conclude the chapter by presenting the contributions (Section 1.4) and structure of this dissertation (Section 1.5).

1.1 Conditional Execution

Recently, deep neural networks have been tailored to be deployable on real-time embedded systems in order to meet application specific latency constraints. This on-device model strategy is spurred by the communication costs/latency incurred by sending a request to a data center to get the output from the model in the cloud and return it to the device. These networks are overparameterized to attain SOTA accuracies on benchmarks and require techniques like weight magnitude pruning [9–11] or quantization [12–18] to fit to device memory constraints. However, there exist a class of applications, such as self-driving cars and server side real-time video processing, that are not limited by parameter count but have strict latency requirements during inference; in this case, network pruning/quantization will not help in terms of directly improving on-device latency. Conditional execution is new model of efficient inference that can address these concerns that more traditional compression methods cannot.

Conditional execution is a paradigm that addresses the computational costs of these models while still preserving their performance. As opposed to more traditional pruning

methods which leverage weight sparsity to reduce computation, conditional execution methods preserve the full capacity of a model and accuracy while cutting down computation by selectively performing operations based on an importance criterion. While conditional execution methods do not reduce static model size, they can significantly reduce the number of memory accesses for a single image inference; prior work shows that the peak memory usage is lower than models without conditional execution, which improves cache utilization [19]. By retaining the weight parameters, the tradeoff between accuracy and latency is potentially much smoother than static methods.

While there are exists many variants of conditional execution methods [19–25], an underlying theme is to have an auxiliary connection called a gating network which determines during runtime which computation for a particular input should be skipped, whether that be individual pixels, feature maps (channels) or even layers themselves. Feature Boosting and Suppression (FBS) is one such work that falls under this category: FBS is a state-of-the-art conditional execution mechanism which adopts this viewpoint by dynamically pruning channels on a layer-by-layer basis during inference [19]. FBS introduces auxiliary connections to the convolutional layers which use features from the previous layers to make a decision on which channels should be zeroed out. By training this predictor, FBS adaptively learns which filters to activate or suppress for specific input features. At low to moderate pruning rates, FBS dynamically prunes convolutional filters with little loss in accuracy.

Dynamically pruning pixels, channels, or layers is one method of accelerating inference but an alternative view of conditional execution is dynamic filter generation [23] which borrows concepts from mixture of experts [26]. Recent work in Conditionally Parameterized Convolutions (CondConv) takes this view in that conventional convolutional weights are replaced with specialized, input-specific filters dynamically created through a linear combination of n experts $(\alpha_1 W_1 + \cdots + \alpha_n W_n)$, where α's are a function of the input [23]. It employs a routing function similar to FBS to encode the important features into the α parameters. CondConv allows for the model capacity to be increased while maintaining a

relatively static compute cost since the experts are combined prior to multiplication against the input features when compared to increasing the depth or width of the network.

The issue with methods like FBS is that they are constrained by how aggressive the pruning rate is set; at low to moderate pruning rates, FBS is an effective pruning strategy which will confer computational savings with little to no cost to accuracy. However, similar to traditional weight pruning, FBS suffers from substantial accuracy loss with high pruning rates, since the number of surviving filters are not sufficient in order to correctly classify the input image. However, we notice an opportunity in FBS's shortcomings. We hypothesize that replacing the conventional convolutional filters in FBS with the high-capacity input-specialized filters of CondConv will directly address the accuracy loss that FBS suffers because of the additional network capacity multiple experts provides. This combined with the ability to create highly efficient composite filters from the combination of experts (which learn distinct features from different classes) counteracts the dynamically lower active filter count and inference-time network narrowing effects of aggressive dynamic pruning. FBS applied to a CondConv model in the experimental results corroborates this hypothesis.

Therefore, our proposal is to exploit one prominent conditional execution technique, which replaces existing convolutional kernels with high-capacity input-specific kernels [23] for enabling high pruning ability of another state-of-the-art dynamic execution mechanism [19] without compromising model accuracy. Our intuition is that with higher quality expert filters generated with CondConv, FBS can be more aggressive in pruning since fewer expert filters can accomplish the same task as numerous generic filters. We test our FBS-pruned CondConv model on CIFAR-10 with a custom 9-layer CNN and demonstrate that we can achieve up to 47.2% savings in computational costs at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over the state-of-art FBS technique. We introduce our FBS-pruned CondConv strategy in Chapter 3.

1.2 Dynamic Data Pruning

The prior section 1.1 was dedicated to one particular way of accelerating inference and reducing computational cost to improve latency [19]. In comparison, accelerating training and reducing training time is much less studied area, partially relying on advances due to technology scaling, better infrastructure, and different parallelization strategies like tensor parallelism, model parallelism and data parallelism [6]. In particular, the modern trend has seemed to embrace this strategy of scaling as evidenced by release of LLMs such as GPT3 [3, 4]. It is accepted that the average practitioner needs access to seemingly countless numbers of GPUs and special accelerators to train models on nontrivial datasets and that the time-scale of these models are on the order of days or weeks. This trend deters independent researchers from applying state-of-the-art techniques to novel datasets and applications, and even large research organizations accept this approach at significant costs.

Currently accepted methods, namely such as mixed precision training[27], target the per-iteration penalty of evaluating the model during training; however, not as much effort has been on spent reducing the total number of training iterations. Since even simple datasets [28] require hundreds of epochs over tens of thousands of samples, eliminating a non-essential subset of data presents a promising opportunity for efficiency.

Work from other DL domains suggest that only a subset of the data influences the decision boundary and contributes a majority of the loss incurred by the model [29, 30]. Furthermore, curriculum learning [31] asserts that samples can be ranked which might allow us to prune redundant “easy” samples. This observation leads itself to the methodology of data pruning, the method of selecting a subset of the dataset that when trained on with a model can achieve the same performance as the full dataset.

Prior work on data pruning [30, 32] take advantage of this property to eliminate a majority of the dataset without incurring significant performance loss. Unfortunately, these methods run their scoring algorithm prior to training and require one or more passes over the dataset. When we include the cost of scoring the samples, the total run time exceeds the time it takes

to do a single conventional training run. This prevents researchers from utilizing the prior work on new, non-standard datasets.

Unlike prior works, our work is based on dynamically selecting a subset of the data at fixed checkpoints throughout training. We notice that by counting every time each sample is selected across all scoring checkpoints, we find that a dataset can be qualitatively split into three groups — always samples, never samples, and sometimes samples. Always samples are selected at nearly every scoring checkpoint. Similarly, never samples are rarely selected. Sometimes samples are selected at only some checkpoints, and their inclusion is highly variable across different training runs. Static pruning methods, like the prior work, can identify always or never samples but fail to effectively target sometimes samples. In fact, we find that randomly selecting a subset of the data at each checkpoint is more effective than the static baselines.

Given this grouping of the dataset, we design a dynamic scoring mechanism based on per-sample loss. Despite scoring more frequently, our mechanism reduces the total run time including the cost of scoring, while the prior work typically increases it. Moreover, at aggressive pruning rates, we obtain higher final test accuracies on CIFAR-10, CIFAR-100, and a synthetically imbalanced variant of CIFAR-10. Since the sometimes samples vary in importance across training runs, we note that the optimal dynamic scoring selection is tightly coupled to the model trajectory. So, we re-frame the data pruning problem as a decision making process. Under this lens, we propose two variants of our scoring mechanism that borrow from ϵ-greedy and upper confidence bound (UCB) algorithms in reinforcement learning. With these additional improvements, we obtain even higher performance at aggressive pruning rates even when the dataset is imbalanced. We discuss our dynamic data pruning framework in Chapter 4.

1.3 PatchDrop for Efficient Training

Section 1.2 discussed data pruning to accelerate training neural networks. Putting data pruning aside, vision transformers (ViT) [8, 33, 34] have seemingly replaced convolution networks for image classification, reinforcing the necessity for accruing massive amounts of computational resources. This continued demand achieves new SOTAs on benchmarks but restricts the number of researcher who are able to use such models. The timescale of these models can be on the order of days to weeks to months depending on the scale.

There have been prior works in accelerating training such as mixed precision training, which targets the precision of the model and quantizes it in a manner where no accuracy loss is sustained but actual speed-ups in training time are realized [27]. Progressive resizing of the input images during training is another strategy that is employed to accelerate network training [35]. In a similar vein, data pruning is a method of finding a subset of the full dataset which allows the model to achieve the same validation accuracy as if the model was trained with the full dataset [30, 32, 36].

Briefly leaving training methods aside, there has been a wealth of literature on accelerating inference at the edge. Referring back to section 1.1, the paradigm of conditional execution is based on the notion that computation of filters should only be done on input features that would generate large responses [19, 20, 22, 37]. These ideas have been extended to pruning input patches to vision transformers to accelerate inference [38–42]. At the heart of these methods to accelerate inference at the edge is the assumption that there are certain patches in the image that are important compared to others and skipping computation on the irrelevant patches will not have a large detrimental effect on accuracy.

In this work, we want to take the idea of data pruning and pruning patches to accelerate inference to create a novel training strategy of accelerating network training by pruning patches which have the least salient features. Because of the vision transformer's unique architecture of separating the image into tokens, we are able to realize actual speedups during training runs which would not be possible on convolutional and MLP architectures

without sparse matrix-matrix multiplication CUDA kernels. We adopt the methodology of [1] who introduced the idea of PatchDrop to stress-test the vision transformer's robustness to occlusion by using the self-supervised ViT model, DINO, [2] to identify the most salient patches in the image. We use their taxonomy, NonSalient PatchDrop, of accelerating network training by identifying heatmaps of the most salient patches and training on them as opposed to the all of the patches in the image. If data pruning should be looked at accelerating network training in a coarse-grained manner, PatchDrop for efficient training is its fine-grained counterpart. We outline our patch pruning method in Chapter 5.

1.4 Contributions

Now that we have outlined the basic premises of the works, the key underlying thread is each of these methods is based on a pruning of irrelevant computation. Specifically, we distinguish that this document aims to target improving models at the edge (efficient inference) and accelerating network training/reducing training time/costs (training efficiency). With these goals in mind, we make the following list of contributions:

- We propose **FBS-Pruned CondConv** as an improved conditional execution method, which results from the combination of two other prior methods and noticing their synergy together. We test our FBS-pruned CondConv model on CIFAR-10 with a custom 9-layer CNN and demonstrate that we can achieve up to 47.2% savings in computational costs at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over the state-of-art FBS technique. Our FBS-pruned CondConv model is a step to straddle the accuracy-computational cost in a gentle way which is a step to more computationally efficient models at inference.

- We propose accelerating network training through **dynamic data pruning**, which dynamically trains on subsets of the dataset. Our algorithm is based on rescoring the training subset based on a scoring criterion, taking inspiration from active learning

and reinforcement learning, on a specified interval. Through empirical observations of scoring, we notice that three distinct categories of samples emerge: always, never, and sometimes samples. Sometimes samples are those which matter intermittently during the training process and is tied to the model's training trajectory. We reframe the data pruning problem as a decision making process. Under this lens, we propose two variants of our scoring mechanism that borrow from ϵ-greedy and upper confidence bound (UCB) algorithms in reinforcement learning. With these additional improvements, we obtain even higher performance at aggressive pruning rates even when the dataset is imbalanced.

- To accelerate ViTs, we propose **PatchDrop for Efficient Training** by leveraging the self-supervised vision transformer DINO [2] and a scheme for identifying the most/least salient patches (NonSalient PatchDrop) in an image [1]. Due to the ViT's unique input representation, we can drop the patches after the positional encoding has been injected into the embeddings to realize actual speedups without the need to have specialized kernels to skip computation in the input. We test our method on the Visual Wake Words dataset [43] against a Random PatchDrop baseline and evaluate each of their respective F_1 score to see NonSalient PatchDrop incurring less accuracy degradation. We also note that when we adjust the image quality to account for image artifacts when upsampling the original dataset, our PatchDrop method suffers less accuracy penalty which suggests that our method will be applicable on higher-resolution/larger scale datasets.

1.5 Structure

The rest of the document's structure is as follows: Chapter 2 discusses the background material required to understand the following chapters. In particular, the basics of conditional execution are introduced as well as more detailed descriptions of FBS and CondConv. Active

learning and reinforcement learning is discussed to understand the methods that are used in the dynamic data pruning work. The structure of the vision transformer is elaborated as well as a brief background on self-supervision to explain how each of these components can be exploited for PatchDrop. Chapter 3 addresses work on making models on the edge more efficient and discusses how FBS-pruned CondConv achieves higher accuracies than each individual method like FBS and CondConv, by exploiting and pruning dynamically generated input-specific filters. Chapter 4 discusses reducing training time by dynamically training on subsets of data. Chapter 5 explores a different access of pruning to reduce training time on ViTs by pruning patches via PatchDrop. Chapter 6 concludes the document by listing the thesis contributions, followed by some reflections.

Chapter 2

Background

This chapter provides background information for the subsequent chapters in this document. Specifically, section 2.1 gives a brief introduction into conditional execution which is purely based on accelerating inference. Further subsections detail the finer details of the conditional execution methods the next chapter is based on. The next section 2.2 on active learning (AL) gives a primer on active learning strategies such as uncertainty sampling, density-based sampling, and hybrid methods to select unlabeled samples to be labeled for training. The following section 2.3 goes over the basic reinforcement learning (RL) framework of agent and environments as well as general methods to yield higher rewards. The uncertainty sampling methods from active learning and ϵ-greedy/UCB approaches form the basis for our dynamic data pruning work. The final section 2.4 is on the vision transformer (ViT) architecture and the details of the multi-headed self-attention (MHSA) module. The last part of the section discusses the DINO model, which is a self supervised vision transformer, that performs image segmentations. This forms the basis for the final chapter on pruning image patches on ViTs to accelerate training.

2.1 Conditional Execution

2.1.1 Overview

Conditional execution is a paradigm that addresses the computational costs of these models while still preserving their performance. As opposed to more traditional pruning methods which leverage weight sparsity to reduce computation, conditional execution methods preserve the full capacity of a model and accuracy while cutting down computation by selectively performing operations based on an importance criterion. There are many flavors of conditional execution but one prevailing viewpoint states that there is unnecessary computation performed for every example; some specific subset of filters will be more activated for a particular class label than another.

In general, conditional execution follows a particular framework [10] and the decision system decides what to prune:

- The type of the decision components: additional connections attached to network during inference or a policy network which learns what parts of the network should be active for the current images [19, 24, 25].
- The pruning granularity: either at the channel level[19, 25], block level [24] or layers [44] themselves.
- Computing a decision score or threshold with some l_p-norm-based criterion to prune channels or layers. [44] .
- Stopping criteria: a) in the case of the layer-wise and network-wise pruning, some pruning algorithms skip the pruned layer/network b) dynamically choose the data path and c) ending computation early and outputting the predicted results [19, 24, 25, 45].

The next set of subsections will discuss Feature Boosting and Suppression (FBS) [19]

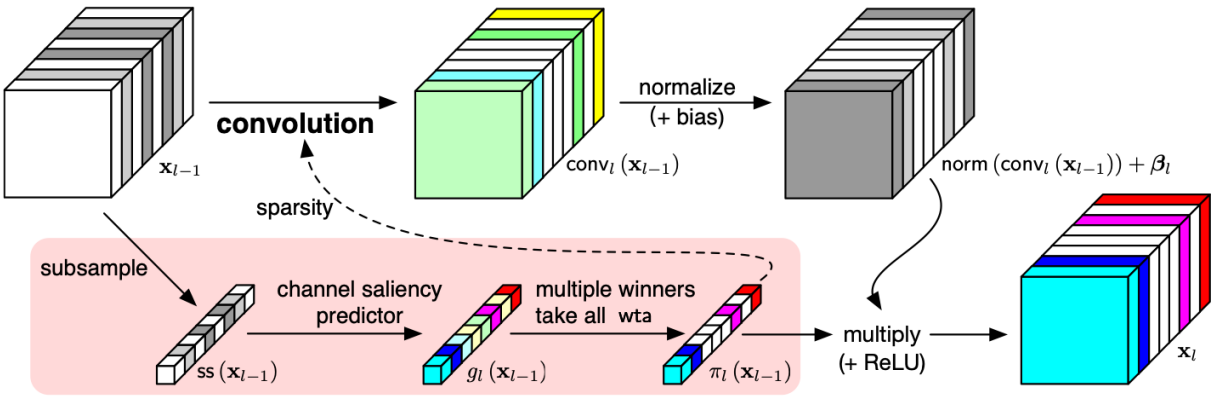
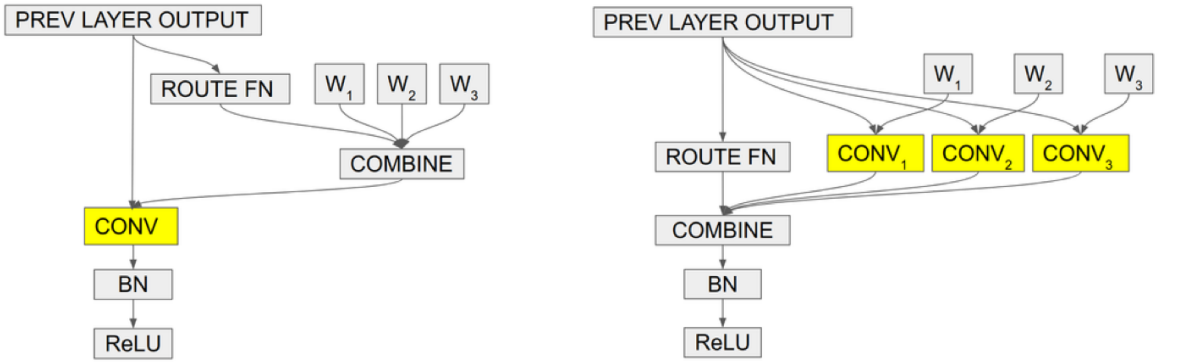


Figure 2.1: FBS overview. Shown in red, an auxiliary connection is added to each layer in the CNN. The input feature is subsampled into a 1D representation to save computation and passed through a feedforward layer called channel saliency predictor to evaluate which channels are important. A winner take-all activation function thresholds the output and drops all the irrelevant input features.

and Conditionally Parameterized Convolutions (CondConv) [23], on which our work [37] is based on.

2.1.2 Feature Boosting and Suppression (FBS)

FBS is a state-of-the-art dynamic execution mechanism that dynamically prunes intermediate channels based on input features [19], shown in Figure 2.1. It introduces an auxiliary connection, called the channel saliency predictor, to evaluate which channels have the important information content. Based on the output of this predictor, a hyperparameter known as the gate density, d, selects which channels should be pruned in each layer via a top-k winner take all activation function. Once the output of the predictor is obtained, only the convolutional weights corresponding to the surviving channels are used for computation. FBS is preferable to previous static pruning methods because it reduces the dynamic model footprint and minimizes the impact on accuracy while still preserving all neurons in the model.



(a) CondConv: $(\alpha_1 W_1 + \ldots + \alpha_n W_n) * x$ (b) Mixture of Experts: $\alpha_1 (W_1 * x) + \ldots + \alpha_n (W_n * x)$

Figure 2.2: Comparing CondConv to traditional Mixture of Experts (MoE) approach. CondConv passes the input feature maps through a routing function which will produce input-dependent scalars to scale multiple filters to create an expert filter. On the other hand, the MoE approach creates intermediate feature maps with separate filters and then is linearly combined with the routing function.

2.1.3 Conditionally Parameterized Convolution (CondConv)

CondConv is a conditional execution technique that replaces conventional convolutional layer with specialized convolutional kernels for each example [23]. As shown in Figure 2.2, the kernels are generated as a linear combination of experts, which are input-dependent scalars. These experts are created in a similar fashion as the auxiliary path in FBS, with the exception there is no pruning of channels. The intuition is rather than increasing the size of the convolution filter, increasing the number of experts in a CondConv layer is more computationally efficient while increasing the model capacity and improving accuracy. Compared with a traditional Mixture of Experts (MoE) approach with n experts, CondConv only incurs one convolution operation with a linear combination of filters whereas MoE incurs n convolutions with linear combination of feature maps. Both of the aforementioned methods will be the foundation for our FBS-pruned CondConv method we introduce in Chapter 3.

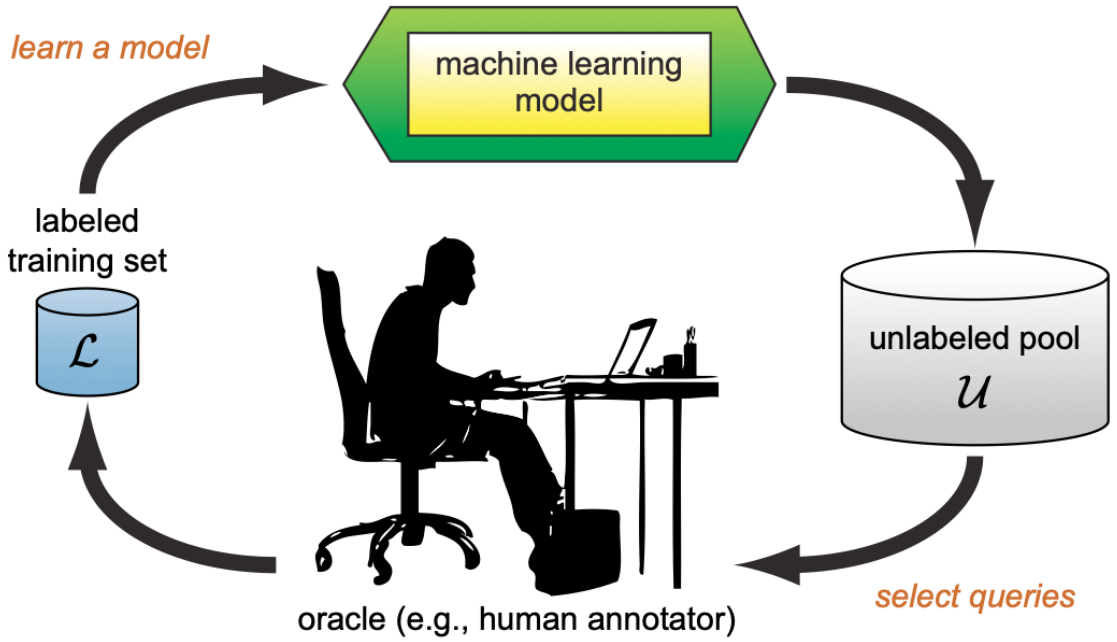


Figure 2.3: Active Learning pipeline. Figure from [49].

2.2 Active Learning

2.2.1 Overview

Current deep learning models require large amounts of labeled data to achieve superhuman accuracy on image classification task [46] and there is a large amount of unlabeled image data to sample from to further improve these models. The main barrier to harnessing this vast data is that the manual effort required by a human oracle to label the images for supervised learning is infeasible. Active learning addresses this problem by selecting a smaller set of examples for an oracle to label [47]. By efficiently selecting which samples are necessary for labeling, active learning cuts down on the oracle effort, while achieving the same performance as if the full pool of examples had been labeled [48].

The active learning pipeline follows several sequential steps as referenced in Figure 2.3. A set of labeled data is obtained and the model is trained on this subset from the distribution. Once the model has reached an acceptable threshold for performance, the unlabeled pool

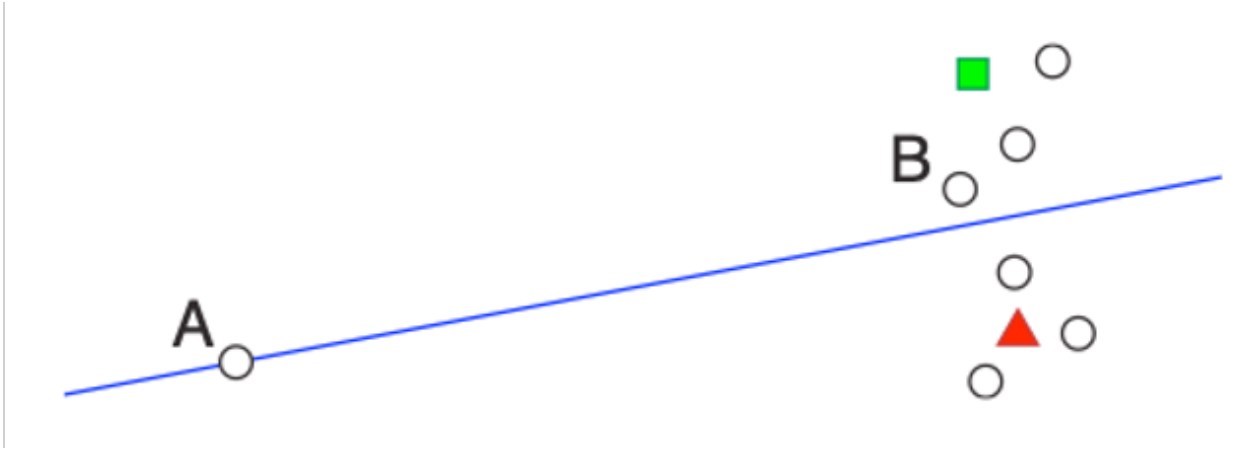


Figure 2.4: An example of uncertainty sampling. The red triangle and green square represent labeled instances from different classes and the white circles represent unlabeled examples. Figure from [49].

is queried for new samples, which are filtered according to a selection process and subject to a labeling budget, the amount of images allowed from the unlabeled pool. The model is retrained with this new data. As one can imagine, retraining is a time-consuming process and is a drain on computation resource, especially when those models are DNNs. As a result, in order for active learning to be scaled to DNNs, the selection criterion needs to be quick and compatible with high dimensional data.

2.2.2 Uncertainty Sampling

One of the challenges in active learning is choosing the selection criterion to query the unlabeled samples [49]. The most ubiquitous strategies is uncertainty sampling, which the model preferentially selects examples for which its current prediction is least confident. Uncertainty sampling is quickly computeable and generally selects points close to the estimated decision boundary, which is a measure of informativeness of a sample. This includes measures such as least confident sampling, which selects unlabeled samples such that the model is most uncertain measured via the softmax output or entropy based sampling, which measures how the amount of information is in a particular sample to encode a distribution. A particular type is uncertainty sampling is margin sampling [50], where margin M is defined as the difference

between the largest and the 2nd largest softmax output of a sample $M = P(y_1|x) - P(y_2|x)$, where y_1 and y_2 are the first and second most probable label for sample x. In this case, the active learning algorithm will select samples with the lowest margin M as it represents greater uncertainty in the sample x.

The limitations of uncertainty sampling are that it is sensitive to label noise and prone to outliers in the data. Moreover, uncertainty sampling leads to insufficient diversity of batch query samples which would lead to unsatisfactory DNN training performance [47]. Figure 2.4 encapsulates how uncertainty sampling works and its pitfalls. The labeled instances in the figure are denoted by a red triangle and green square whereas the white circles are unlabeled instances, with A and B being different unlabeled instances. An uncertainty sampling based approach would select A to be labeled because of its closeness to the the decision boundary but arguably choosing B would result in a better global view of the data distribution.

2.2.3 Density-based Sampling

Another type of selection criterion is know as density-based or diversity based approaches. These methods aim to select samples from the unlabeled pool that are representative of the input distribution, also called core set, rather than the instances which would cause maximal change in the model. The construction of the coreset is to represent the distribution of the feature space of the entire original dataset [47, 51], which rely on distance-based metrics. Farthest First Active Learning is based on the idea of undergoing a traversal in the space of neural activation over a representation layer [52]. In order to improve sample diversity, active learning can use random sampling in the early states to enhance selection criteria [51]. An approach that depends on the data distribution by considering active learning as a binary classification task [53]; by assuming the distribution of the unlabeled data and labeled data are the same, the authors consider whether the samples are similar to the labeled data. The samples which are different from the current distribution are selected to promote diversity. However, the downside to the coreset approach is that it requires a large distance matrix to

be built based on the label set, which is further exacerbated when scaled to larger, complex problems. The density-based approaches often rely on greedy algorithms to select subsets, which inhibit their scalabilty.

2.2.4 Hybrid methods

The final category of selection criterion is a hybrid query strategy between uncertainty sampling and diversity sampling. A hybrid approach would take into both the information volume and diversity of samples implicitly or explicitly. [49] describe a hybrid scheme which they call the information density framework, which is as follows:

$$x^*_{ID} = \text{argmax}_x \phi_A(x) \times (\frac{1}{U} \sum_{u=1}^{U} \text{sim}(x, x^u))^\beta. \tag{2.1}$$

In the above equation, $\phi_A(x)$ represents some query strategy based on uncertainty sampling. The second term weights the informativeness of x its average similarity to all other instances in the input distribution, subject to a parameter β that controls the relative importance of the density term. This approach is used in recent approaches like Exploration-P, which combines ideas like informativeness and diversity [54]. Rather than relying on a feature vector to calculate similarity amongst instances, they propose a deep learning approach where a pairwise deep network project instances into a similar space where they can be measured more precisely. In this manner, an active learning algorithm can select set of instances that maximally uncertain, which resembles exploitation, whereas the most diverse from labeled instances, which resembles exploration. These methods seem to include the best elements from both worlds but the main challenge is figuring out the appropriate tradeoffs between uncertainty and diversity as well as impact of other common deep learning hyperparameters such as the batch size [55]. Chapter 4 discusses how we integrate active learning techniques into our dynamic data pruning framework.

2.3 Reinforcement Learning

2.3.1 Overview

Reinforcement learning is a different paradigm of machine learning than supervised and unsupervised learning; compared to supervised learning, reinforcement learning uses the training data to evaluate actions taken rather than instructing what the correct actions are [56]. It is predicated on how "intelligent" agents ought to take actions in an environment in order to maximize a defend reward function. Reinforcement learning has been a principled mathematical framework for experience-driven autonomous learning but has lacked scalability and was thus limited to low-dimension problems.

The reinforcement learning problem is split two main components: agents/actors and the environment. The agent observes state s_t from the environment at timestep t and performs action a_tm which consequently changed state s_{t+1}. The goal of the agent is to learn a *policy*, which is a mapping from perceived states of the environment to actions to be to be taken when in those states. The environment sends the agent back a *reward signal* which defines what is a good action in an immediate sense. Subsequently, a *value function* specifies what actions are good in the long term; the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state [56, 57]. Figure 2.5 outlines this process. An optional component of RL problems is construction of a model of environment: this allows for a prediction of the next state of the environment, based on the current state of the environment and action of the agent. This seperates RL methods into 2 classes: model based methods and model-free methods.

A simplified version of the RL learning problem is to consider a setting called the n-armed bandit problem. Essentially, the agent is faced with n different choices and each action will yield some expected reward from an underlying true reward distribution, which the agent does not have access to. Again, the goal in these types of problem is yield the maximum reward so the actions selected would be the ones that give the highest immediate reward.

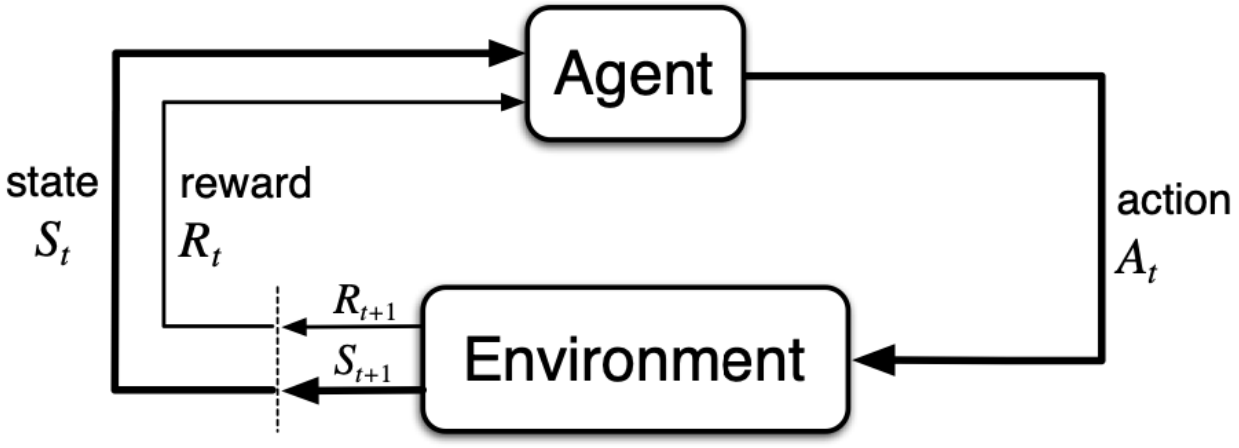


Figure 2.5: The reinforcement learning problem setup with the agent interacting with the environment. Figure from [56].

This setting allows us to understand the exploration-exploitation tradeoff.

An additional feature from RL problems is the inherent exploration-exploitation tradeoff: this refers to what actions an agent should taken in order to maximize its value function. An exploitation strategy can be an agent taking the action which yields the maximum rewards in the short term. On the other hand, an exploration strategy might select suboptimal actions to order to get more information about the environment for better rewards in the long term. There are many different strategies to address this tradeoff but two (relatively) simple approaches are: ϵ-greedy and Upper Bound Confidence (UCB) methods.

2.3.2 Greedy and epsilon-greedy strategy

Assuming that the true value of an action a is $q(a)$ (action-value), $Q_t(a)$ is defined as the estimated value on the tth time step. More explicitly, it is the averaging the rewards over t time steps

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)}, \tag{2.2}$$

where $R_1, R_2, ..., R_{N_t(a)}$ are the rewards at each time step and $N_t(a)$ is the number of steps. As $N_t(a) \to \infty$, $Q_t(a)$ converges to the true value of $q(a)$ by the law of large numbers.

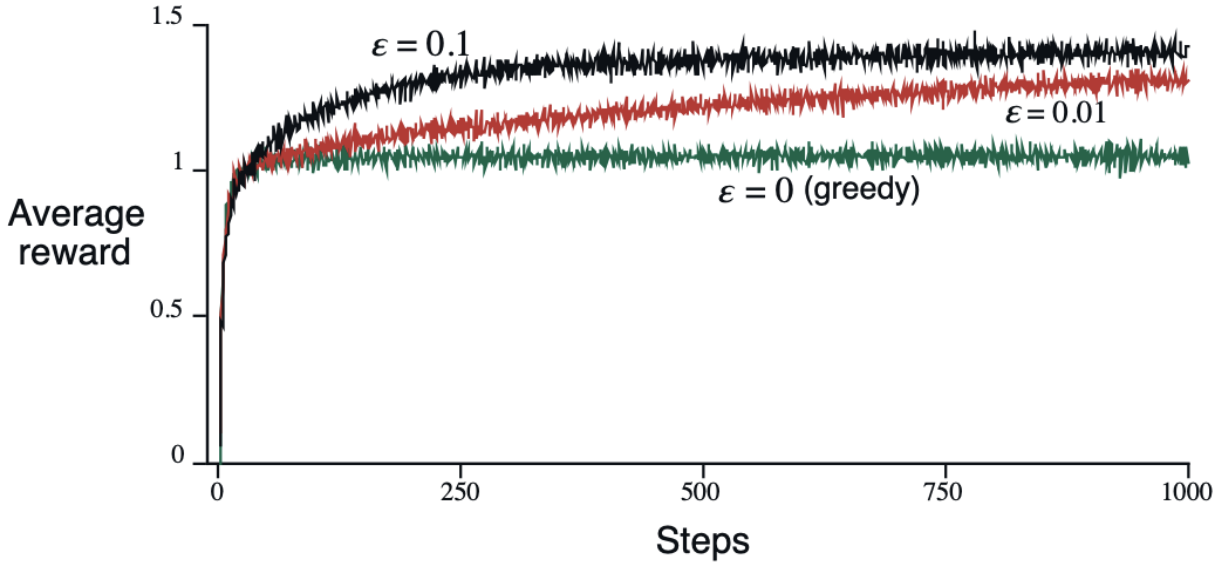


Figure 2.6: Average performance of greedy vs ϵ-greedy strategy. All methods used sample averages as their action-value estimates. With larger ϵ values, the small amount of randomness provides a substantial increase in average reward with more time steps. Figure from [56].

As mentioned earlier in the n-armed bandit problem, the agent will choose the actions that will maximize the rewards. This is known as the greedy action and is purely based on exploitation only strategy. It can be written as

$$A_t = \text{argmax}_a Q_t(a). \tag{2.3}$$

There is no time spent seeing other inferior actions to reap larger rewards in the long term. A simple alternative to the greedy strategy is perform the greedy action expect for a few suboptimal actions that will be taken with a small probability ϵ: randomly selecting an action irrespective of the reward function. This is known as an *epsilon*-greedy strategy. To compare the difference between these two methods with $\epsilon = 0.01$ and $\epsilon = 0.1$, [56] devise an experiment where a set of 2000 of randomly generated of 10-arm bandits and the action values, $q(a)$, $a = 1, ..., 10$ were selected to Gaussian distribution with mean 0 and variance 1 as shown in Figure 2.6. On each time step, the reward R_t was $q(A_t)$ plus a noise term drawn from the same distribution. As ϵ increased, the average reward is much higher than

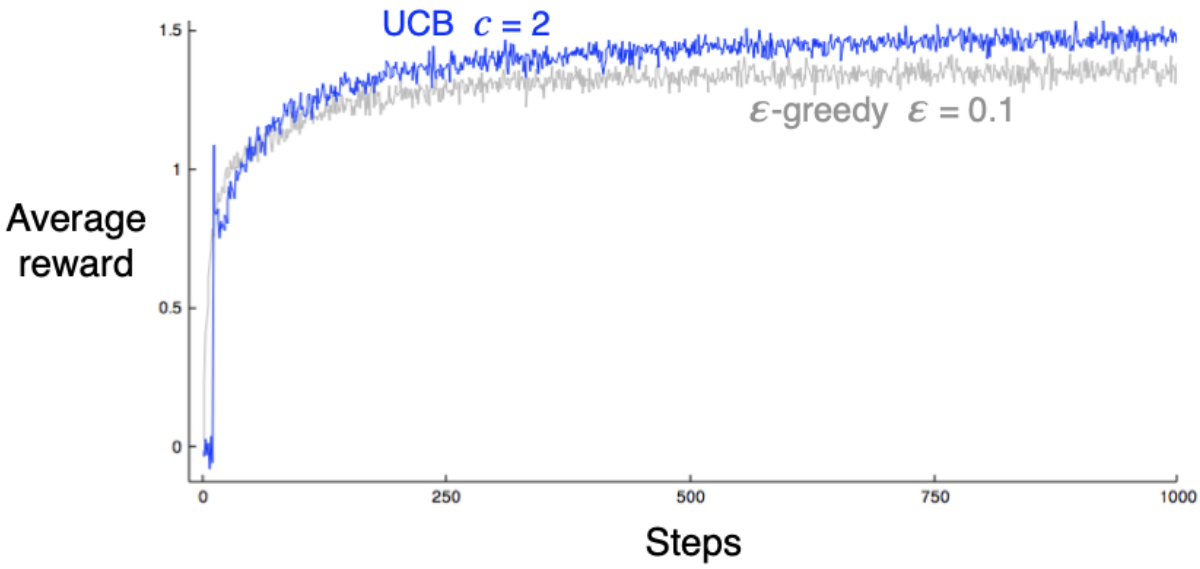


Figure 2.7: UCB, with confidence parameter $c = 2$, vs ϵ-greedy, with $\epsilon = 0.1$, on a 10-arms bandit problem. UCB is able to retain higher average rewards due its ability to select action-value estimate's variance. Figure from [56].

the greedy method; this is because the greedy method does not have enough information to choose the most optimal actions in the long term. There are variants of ϵ-greedy where *epsilon* is set to a large value at the onset of training but is slowly annealed to a purely exploitation-based approach [58–61].

2.3.3 Upper Bound Confidence (UCB)

The ϵ-greedy action selection method facilitates exploration but it does so in an indiscriminate manner by randomly choosing actions regardless of their action-value estimates. The upper bound confidence (UCB) action-selection selects action based on maximal rewards from an action and the uncertainties in the action-value estimates. The UCB action selection is defined as

$$A_t = \text{argmax}_a[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}}], \tag{2.4}$$

where t is a constant that is incremented any time an action other than action a is selected and c is confidence parameter to determine how much exploration the agent should do.

UCB takes into account the variance of an action-value estimate that ϵ-greedy is not able to. Figure 2.7 shows the results of the previous 10-arm bandits, which shows that UCB is able to achieve higher average rewards than the ϵ-greedy baseline. UCB has been used in works that implement a concept called intrinsic motivation, which decreases uncertainty in learning about the environment [57, 62] by minimizing model prediction error or maximizing information gain [63, 64]. Chapter 4 shows how we are able to apply ϵ-greedy and UCB reinforcement learning methods into our selection criteria for our dynamic data pruning work.

2.4 Vision Transformer

2.4.1 Overview

Transformer architectures have been at the forefront of success for domains like natural language processing and general reasoning tasks like GPT3 [3, 7, 65]. This success of the transformer in these domains provided inspiration for the computer vision community to adopt it for image recognition. [8] was the first of a long line of work [33, 34, 66, 67] to transmute the transformer encoder-decoder architecture to computer vision problems. The main component that makes the transformer different from its recurrent predecessor (recurrent neural network) is the multi-headed self attention (MHSA) module, which enables the model to capture long term dependencies that the RNN was unable to encode with the backpropagation through time (BPTT) algorithm. Figure 2.8 outlines at a high level how an input is consumed by the transformer.

2.4.2 Processing the input

Specifically, only the encoder portion of the transformer is used for the vision task. The transformer can only process sequences of tokens so the representation of the input image needs to be converted into a sequence. The input image can be transformed into a sequence

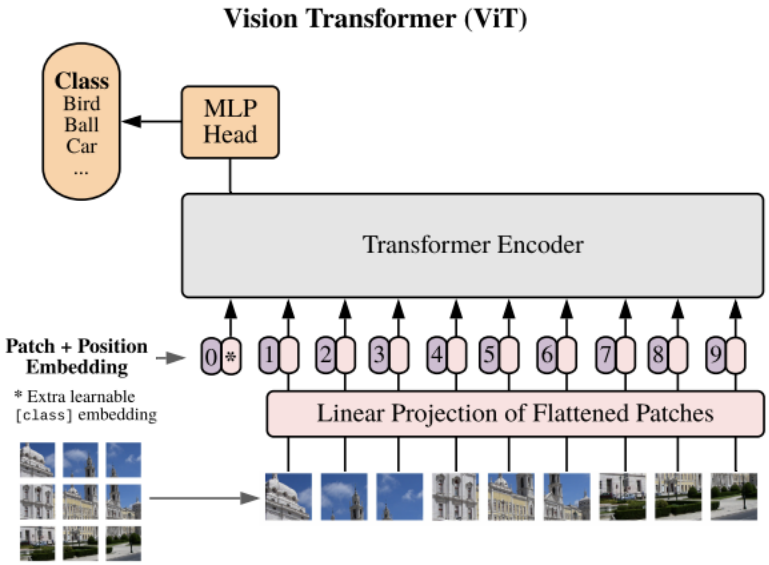


Figure 2.8: Vision Transformer Architecture. The input image is broken up into smaller patches and fed through a linear projection layer. A learnable class token is appended to capture the effect of mixing the other input tokens. To encode the position of the patches and preserve ordering, a positional embedding is injected into all the tokens and then is passed into the transformer. After the final MHSA layer, the class token is extracted and passed through a feedforward layer to produce the final classification output. Figure from [8].

by segmenting the image into smaller patches and unrolling them; the resolution of the smaller patches is a hyperparameter. Typically, a smaller patch size is beneficial as this can isolate different features at a more fine-grained level but comes at the price of higher computation.

Given an image $x \in \mathbb{R}^{H \times W \times C}$, the transformer flattens the input and reshapes it into $x \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the number of patches. The flattened are fed through a linear layer in order to match the embedding space D of the transformer encoder as shown in Equation 2.5. As seen from Equation 2.5, a class token, a learnable embedding, is added to the input in order to capture the global information when the patches mix in the MHSA layer and is passed to the classification head to determine the output.

$$z_0 = [x_{class}; x_p^1 E; x_p^2 E; \ldots; x_p^N E] + E_{pos} \quad E \in \mathbb{R}^{P^2 \cdot C \times D} \quad E_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{2.5}$$

Since the architecture is viewing the inputs all at once rather than a sequential fashion like RNN, a positional encoding scheme must be introduced to encode ordering, denoted by E_{pos} in Equation 2.5. This can be accomplished either through a learnable embedding table or through a set of predetermined sinusoidal functions for even and odd positions in the sequence, as shown in [7, 8].

2.4.3 Multi-headed Self Attention (MHSA) block

The Multi-headed Self Attention (MHSA) module is the main component of the transformer architecture and is the feature that allows the transformer to have a global view of the image compared to its CNN counterpart. It is primarily based on 2 subcomponents: the self-attention (SA) module and a feedforward or linear layer.

Self-Attention (SA)

For an input sequence $x \in \mathbb{R}^{N \times D}$, the SA operation is defined as follows:

$$[Q, K, V] = xP \quad P \in \mathbb{R}^{D \times 3D} \tag{2.6}$$

$$\text{SA}(Q, K, V) = V * \text{softmax}(\frac{QK^T}{\sqrt{D}}) \tag{2.7}$$

where N is the number of input patches and the D is the embedding dimension. The input sequence is projected by a learnable matrix P into a higher-dimensional space where it is split into 3 separate embeddings: queries (Q), keys (K), and values (V) as seen in Equation 2.6. The SA module then takes the Q and K embeddings to generate the attention matrix in order to evaluate the relationship of features encoded in one embedding and other features in the remaining embeddings to see how similar. This is the feature which allows the transformer to learn long range dependencies. After applying a scaling factor $\frac{1}{\sqrt{D}}$ to the product, the output is passed through a softmax to generate attention weights to scale the remaining value embeddings. A way to think about the self attention is to think of the attention matrix as a kernel function which computes a similarity metric between two embeddings for all the different pairs of embedding and then mixing the pairwise similarity with the values embedding.

The SA operation can be extended by executing several different SA operations in parallel, where each SA module is referred to as a head. Stacking these heads, H, in parallel is referred to as the multi-headed self attention (MHSA) module, described in Equation 2.8:

$$\text{MHSA}(x) = W_O \text{ConCat}[SA(x)_1 \dots SA(x)_H]_{h=1}^{H}. \tag{2.8}$$

Specifically, the *ConCat* operation takes all the outputs of each head and concatenates them into one large embedding, which is fed through another linear layer $W_O \in \mathbb{R}^{N \times NH}$ in

order to mix the information from the various heads into the final output embedding.

FeedForward (FF)

The FeedFoward (FF) Layer takes in the output of the MHSA block and passes it through a linear layer with the GELU non-linearity activation function [68]. These operations are generally preceded with Layer Normalization (LN) [69] and residual connections.

$$x_i = \text{LN}(\text{MHSA}(x_{i-1})) + x_{i-1} \tag{2.9}$$

$$x_i = \text{LN}(FF(x_i)) + x_i \tag{2.10}$$

Equations 2.9 and 2.10 describe the operations of the transformer encoder. After a series of stacked layers of encoders, the class token x_{class} is extracted from the final encoder layer and passed through a FF head which will map the output to the label space.

2.4.4 ViT variations

This is the basic structure of the baseline ViT architecture. There is a plethora of works that have emerged based on ViTs success on image tasks. The current ViT model require orders of magnitude more data compared to CNN counterparts [8, 67]. To surpass ImageNet benchmarks, the ViT requires pretraining on large-scale datasets like JFT-300M and ImageNet-21k. However, other ViTs have introduced changes to rival CNNs in terms of data efficiency. DeiT incorporates knowledge distillation [70] and a distillation token, which is a learnable token, to match the output distribution of a pretrained CNN teacher model [33, 34]. This allows the transformer to be trained to SOTA from random parameters. Tokens-to-token (T2T) ViTs aggregate adjacent tokens to structurize them into one token such that local structure represented by surrounding tokens can be modeled and token length can be reduced [71]. LeViT applies a four-layered CNN block (with 3×3 convolutions) at the

beginning with progressively increasing channels (3,32,64,128,256). For a 3×224×224 input image, the resulting 256×14×14 output from the CNN block becomes input to a hierarchical ViT [72]. These are some of the more recent changes to the ViT architecture but a more detailed description of various changes can be found in [73].

2.4.5 ViTs and Self-Supervised Learning

Vision transformers can also be applied in self-supervised settings [2]. Self-supervised learning is based on a paradigm of learning where the model learns features from unlabeled data via different proxy tasks. A particular strand of this approach is called instance classification in which each example in the dataset is treated as its unique class and trains the model via data augmentation [74–76]. Unfortunately, these type of approaches do not scale with the number of images in the dataset. In particular [76] uses a noise contrastive estimator to compare instances rather than classifying them. An orthogonal approach to self-supervised learning is described in Bootstrap Your Own Latent in which two neural networks are used, one being an "online" and the other "target" network [77]. The idea is that the online network will try to predict the target network's representation from the same image given different augmentations. The target network then gets slowly updated via an exponential moving average of the online network. This approach resembles knowledge distillation [70] which is the process of guiding a smaller compact student model to match the output distribution of a typically larger teacher model.

With these preliminaries, we can discuss one particular self-supervised vision transformers which relies on a form of self-**di**stillation with **no** labels, called DINO [2]. The general flow of DINO is presented in Figure 2.9: a student model g_{θ_s}, is trained to match the output of the teacher model with the cross entropy loss function given as such:

$$\min \quad H(P_t(x), P_s(x)) \tag{2.11}$$

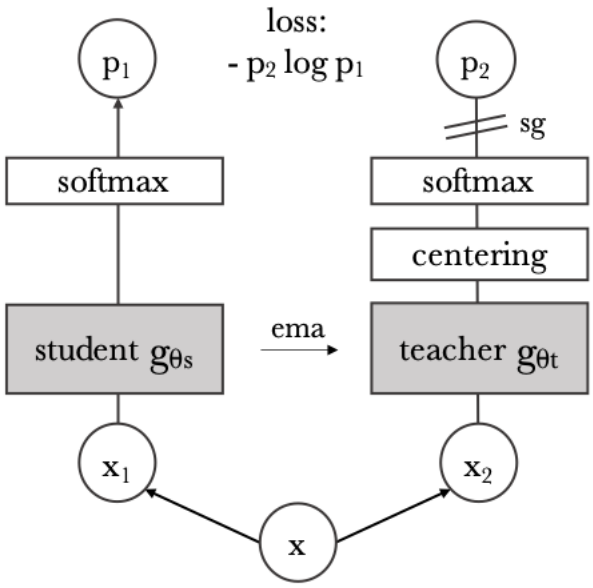


Figure 2.9: An input, x, is split into two different views of the original image via random transformations into inputs, $x1$ and $x2$. This is passed through two networks presented as the student and teacher networks to obtain a probability vector. The loss is computed via the cross entropy loss between the output distributions of the student and teacher models. The student model is updated via gradient descent whereas the teacher model is updated via an exponential moving average. Figure from [2].

Figure 2.10: Comparing the output of the DINO model from the self-attention maps compared to a supervised baseline on the PASCAL VOC12 dataset. It is clear that the segmentation obtained from the DINO model is much more focused on the foreground object. Figure from [2].

where $H(a, b) = -a \log(b)$ and P(s) and P(t) are the output distributions of the student and teacher models. The probability P, of course, is obtained via the softmax function with a temperature parameter, τ, to control the sharpness of the output distribution:

$$P_s(x) = \frac{exp(g_\theta/\tau)}{\sum_{k=1}^{K} exp(g_\theta/\tau)}. \tag{2.12}$$

In order to learn the features of objects with the target labels, DINO uses a multi-cropping strategy [78] where a set of different views is generated and split into 2 distinct sets: local views and global views. The student receives all the various types of crops whereas the teacher only obtains the global crops. This type of separation of the inputs from the student and teacher models facilities the student to learn "local-to-global" features. The student model is updated via gradient descent and the teacher model is updated via an exponential moving average. The last feature of the DINO model is a centering and sharpening operation at the output of the teacher model. This centering operation is to avoid model collapse which is the phenomenon where the model learns a trivial representation of the data and is not able to generalize. And the sharpening operation accomplishes the goal where small values in the probability vector get suppressed and larger values get amplified and makes the output of the teacher similar to a one-hot label. DINO can be used for several different

applications such as $k-NN$ classification, image retrieval, transfer learning, etc. but for our work, we are interested in DINO's ability to produce good heatmaps for segmentation on foreground objects in images. Figure 2.10 shows the superiority of DINO over its supervised counterparts. DINO is an integral part of of the patching pruning procedure we introduce in Chapter 5.

Chapter 3

Understanding the Impact of Dynamic Channel Pruning on Conditionally Parameterized Convolutions

This chapter discusses the integration of Dynamic Channel Pruning on Conditionally Parameterized Convolutions. Section 3.1 introduces the concept of conditional execution and the combination of two different paradigms of conditional execution. The next section 3.2 talks about about related work with compact architectures, sparsity, and other conditional execution works. Section 3.3 describes the intricate details on how we augment FBS to Conditionally Parameterized Convolutions. The subsequent section 3.4 shows the results of FBS-pruned CondConv over the two other conditional execution methods, FBS and CondConv, which shows that less accuracy degradation is suffered. Section 3.5 summarizes the results and suggests avenues for future work.

3.1 Introduction

In recent years, deep neural networks have surpassed human performance on image classification tasks and and speech recognition. The accuracy performance gains are attributed to increasing the model size by adding more layers, which comes at increased computation cost. While these deep models are highly accurate at classification, deploying these models in latency-critical applications, such as self-driving cars and server side real-time video processing, is impractical [23]. These particular applications are unique in that they are not constrained by parameter count, but have strict latency requirements at inference, which motivates the need for a new class of models.

As described in Chapter 1, conditional execution is a paradigm that addresses the computational costs of these models while still preserving their performance. Feature Boosting and Suppression (FBS) is a state-of-the-art conditional execution mechanism which adopts this viewpoint by dynamically pruning channels on a layer-by-layer basis during inference [19]. As referenced in Chapter 2, at low to moderate pruning rates, FBS dynamically prunes convolutional filters with little loss in accuracy. However, with a more aggressive pruning rate FBS suffers from heavy accuracy loss in a similar way to aggressive static pruning, due to a low number of active filters and correspondingly narrower effective network per inference, making the classification task more challenging. The other work we described in Chapter 2 is Conditional Parameterized Convolutions (CondConv) where conventional convolutional weights are replaced with specialized, input-specific filters dynamically created through a linear combination of n experts [23].

While analyzing these two methods, we observed a potential opportunity to address the accuracy loss of the state-of-the-art FBS mechanism at high pruning rates. We hypothesize that replacing the conventional convolutional filters in FBS with the high-capacity input-specialized filters of CondConv will directly address the accuracy loss that FBS suffers because of the additional network capacity multiple experts provides. This combined with the ability to create highly efficient composite filters from the combination of experts (which

learn distinct features from different classes) counteracts the dynamically lower active filter count and inference-time network narrowing effects of aggressive dynamic pruning. FBS applied to a CondConv model in the experimental results corroborates this hypothesis.

In this chapter, our key contribution is that we discover the opportunity to exploit one prominent conditional execution technique, which replaces existing convolutional kernels with high-capacity input-specific kernels [23] for enabling high pruning ability of another state-of-the-art dynamic execution mechanism [19] without compromising model accuracy. Our intuition is that with higher quality expert filters generated with CondConv, FBS can be more aggressive in pruning since fewer expert filters can accomplish the same task as numerous generic filters. We test our FBS-pruned CondConv model on CIFAR-10 with a custom 9-layer CNN and demonstrate that we can achieve up to 47.2% savings in computational costs at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over the state-of-art FBS technique.

3.2 Related Work

Designing efficient CNNs has been an active, ongoing area of research. In recent years, numerous research efforts have been devoted to compressing neural networks through the use of model pruning [79], quantization [16–18], low-rank tensor decomposition [80–83], compact network architecture design [84], etc.

3.2.1 Compact Network Architectures

One such approach is creating new custom network architectures. SqueezeNet reduces the number of parameters by replacing them with 1×1 convolutions in the fire module [85]. The MobileNet series introduce multiple changes, such as decomposition of 3×3 convolutions into depthwise convolution and pointwise convolution, adding inverted residuals and linear bottlenecks, and squeeze-excitation layers [86–89]. Shufflenet introduces a pointwise group

convolution and channel shuffle operation to reduce operations while preserving accuracy [90]. These architectural optimizations are complementary when compared to our proposal in this work. They can be used in conjunction with our proposal to further reduce model size and computational complexity.

3.2.2 Sparsity

Several other works deal with the computational efficiency issue by inducing sparsity in the network. Traditional methods involve weight pruning, which zeroes out individual weight parameters which contribute little to the output [91]. Tensor factorization is another model compression approach that decomposes a single layer into multiple small layers [92, 93]. Channel pruning involves trying to prune entire neurons from the network [21, 94]. The key drawback with these works is that the model capacity gets permanently reduced, potentially incurring significant loss in accuracy.

3.2.3 Conditional Execution

Conditional execution is another approach to efficient deep learning inference. These methods rely on skipping part of an existing model based on the input features [19, 20, 22, 23, 95–97].

Feature boosting and suppression (FBS) is a state-of-the-art dynamic execution mechanism that dynamically prunes intermediate channels based on input features [19]. It introduces an auxiliary connection, called the channel saliency predictor, to evaluate which channels have the important information content. Based on the output of this predictor, a hyperparameter known as the gate density, d, selects which channels should be pruned in each layer via a top-k winner take all activation function. Once the output of the predictor is obtained, only the convolutional weights corresponding to the surviving channels are used for computation. FBS is preferable to previous static pruning methods because it reduces the dynamic model footprint and minimizes the impact on accuracy while still preserving all

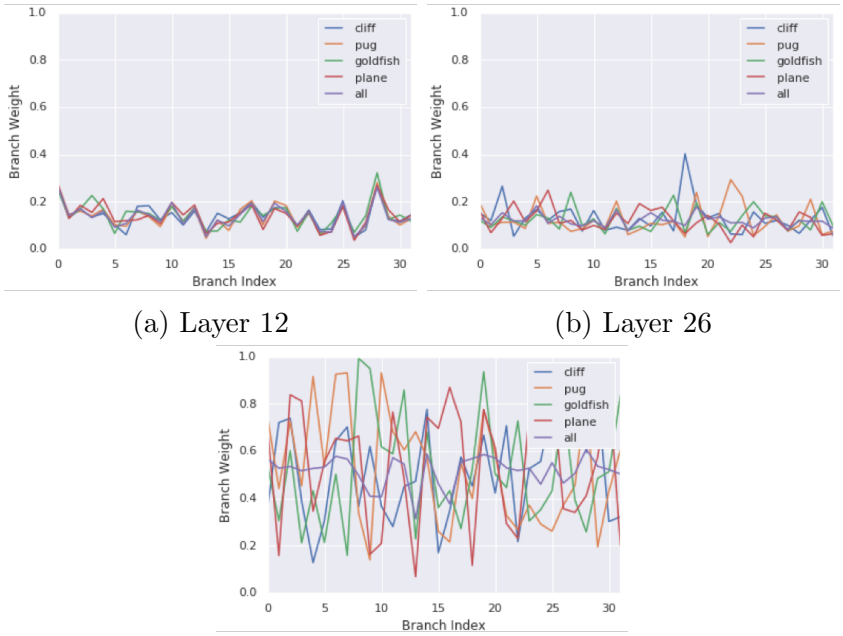
neurons in the model.

Conditionally Parameterized Convolutions (CondConv) is another prominent conditional execution technique that replaces conventional convolutional layer with specialized convolutional kernels for each example [23]. In CondConv, the kernels are generated as a linear combination of experts, which are input-dependent scalars. These experts are created in a similar fashion as the auxiliary path in FBS, with the exception there is no pruning of channels. The intuition is rather than increasing the size of the convolution filter, increasing the number of experts in a CondConv layer is more computationally efficient while increasing the model capacity and improving accuracy.

Besides the above two, channel gating neural networks identify regions in the features that contribute less to the output and skips the computation for those specific regions [96]. Precision Gating is a quantization technique that computes most features using low precision and only a small subset of important features with higher precision to preserve accuracy [20]. Batch-shaped channel gated networks use the concept called batch-shaping, matching the marginal aggregate posterior of features in a network to a pre-specified prior distribution, to force the gating mechanism to be more conditional on data [22]. Dynamic Convolution augments the convolution kernels by dynamically scaling them based on their attention, making them input-dependent [95].

3.3 Feature Boosting and Suppression of Conditionally Parameterized Convolutions

While FBS is able to prune filters effectively, one limiting factor of FBS is the requirement of certain minimum number of active filters in all layers to ensure no or minimal degradation in accuracy. This minimum amount of filters places a restriction on how aggressive the pruning rate can be set. However, our observations indicate that we can address this limitation by replacing conventional convolutional kernels with CondConv kernels so that we can prune



(a) Layer 12

(b) Layer 26

(c) Full Connected (FC)

Figure 3.1: Mean CondConv routing weights for four classes averaged across the ImageNet validation set at three different depths (a) layer 12, (b) layer 26 and (c) the FC layer of MobileNetV1. CondConv routing weights become more class-specific at deep layers. Figure is from [23].

more heavily with no loss in accuracy.

3.3.1 Motivation

At moderate pruning rates where a majority of filters are retained, FBS can perform well while reaping substantial gains in computational savings. However, this is not necessarily the case at higher pruning rates since FBS may not retain enough input-specific specialized filters to classify an image correctly. Because FBS does not have as much representational ability at dynamically-narrow aggressive pruning rates, it becomes more challenging to differentiate between classes. In spite of this fact, pruning aggressively is still attractive since we want to capture the benefits of a quadratic savings in compute.

CondConv can resolve some of the issues FBS suffers through the generation of input-specific filters since CondConv may select and combine from a varied set of experts for distinct classes. By separating which experts get activated for a particular class, we may more easily

discriminate between which filters are candidates to be pruned since the probability of two different classes sharing input-specific filters is low. In other words, CondConv may enable FBS to aggressively prune a majority of the other input-specific filters since a smaller number of these filters may be sufficient to classify the input when compared to the baseline FBS, which needs to prune generic (non-input specific) filters prone to destructive interference.

To illustrate the points presented above, consider an example layer in a FBS network layer with 32 channels for a toy problem with 10 classes. If the pruning rate (gate density in this case) is set to 0.5, half of the filters are eliminated per-inference, but classification accuracy may be sustained since the remaining 16 filters are still sufficient to classify the input. If the pruning rate is set too aggressively though, e.g. to 0.3, it is plausible that using only 10 filters does not have enough representative power to distinguish between classes in all cases and would cause a drop in accuracy. However, if the generic filters in this FBS layer are replaced with ConvConv filters with 4 experts, two important observations arise: (1) the model capacity is inflated by 4×, granting much more representational ability to the layer, and (2) since the CondConv filters are created in a fine-grained, input-specific manner, good classification may not require all the experts. These reasons are an explanation for why using CondConv filters may allow FBS to use a high pruning rate with little to no loss in accuracy.

To reiterate, CondConv experts learn distinct features from certain classes and not others. Figure 3.1, taken from [23], plots the mean routing weights for MobileNetV1 with 32 experts on 4 classes of the ImageNet validation set [46]. For earlier layers in the network, as in 3.1a, experts are not learning specific filters which differentiate between classes, implying generally the dominance of useful low-level features. This figure shows that replacing the convolution layer with a CondConv layer is akin to increasing the depth of the layer. From Figure 3.1b we see deeper layer experts become more input specific and certain combinations of experts become active for distinct classes. At the penultimate layer in Figure 3.1c, only one or two experts are active for each class example indicating that in the deepest layers of

the network CondConv kernels resemble a mixture of experts. Although not as pronounced, specialization of filters is present at layer 26 as shown in Figure 3.1b. However, filters become more individual and specialized deeper in the network as demonstrated in Figure 3.1c. The authors of [23] further corroborates this observation that CondConv experts specialize, going so far as to graphically demonstrate a specific instance of a filter whose experts specializes in wheeled vehicles, rectangles, cylinders and dogs. Figure 3.2, also from [23], corroborates this observation further showing that experts in deep layers are triggered in an exclusive fashion. The key takeaway is that which experts even get activated by the auxiliary paths or routing function preemptively reveals which class the input is likely to belong to and thus reduces the scope of the classification task, which allows for more pruning opportunity.

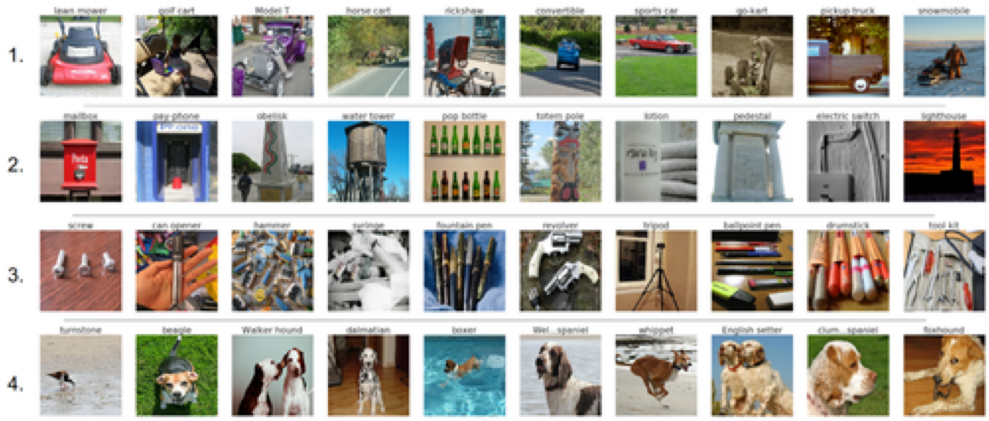


Figure 3.2: The examples that maximally activated each expert, by routing weight, for the last layer of MobileNetV1 on the ImageNet validation set. The first expert is activated by wheeled vechicles. The second by rectangular objects. The third by household cylindrical objects. The last by black and brown dog breeds. Figure is from [23].

This key insight motivates applying dynamic channel pruning, similar to FBS, to a model made of input-specific expert filters, as in CondConv, in order to reap more computational savings while ensuring little to no loss in model accuracy. Experts becoming highly specialized imply that fewer filters are needed for proper prediction. With higher quality filters, FBS can take advantage of the other irrelevant, specialized filters and prune them at an aggressive rate with no accuracy penalty. In the following subsections, we provide detailed descriptions of expert filter generation in CondConv and then show how FBS can be applied to enable

abundant pruning of expert filters, resulting in significant reductions in computational cost.

3.3.2 Designing a Conditionally Parameterized Convolutional Layer

We consider a CNN, $F(\cdot)$, with L layers with the following definition, $y = F(x_{in}) = f_L(\cdots f_2(f_1(x_0))\cdots)$, in which the n^{th} layer $f_n : \mathbb{R}^{C_{n-1}\times H_{n-1}\times W_{n-1}} \rightarrow \mathbb{R}^{C_n\times H_n\times W_n}$ and x_{in}, y are the input and output respectively. Each layer in the network will produce an output with C_n channels of features with height H_n and width W_n. Each f is described by a convolution operation, followed by an activation function. The n^{th} layer can be expressed as

$$f_n(x_{n-1}) = \sigma_n(\gamma_n \cdot norm(conv_n(x_{n-1}, W_n)) + \beta_n). \tag{3.1}$$

In the above equation, $conv_n$ is parameterized by weight tensors, $W_n \in \mathbb{R}^{C^l\times C^{l-1}\times k\times k}$ (k being the kernel size), and input from the previous layer $x_{n-1} \in \mathbb{R}^{C^{l-1}\times H^{l-1}\times W^{l-1}}$. $norm$ is the batch normalization operation [98] and σ_n denotes the activation function at the n^{th} layer. γ_n and β_n are introduced as trainable parameters for every convolutional layer which will be introduced in section 3.3.3.

Adding CondConv into the derivation above is simple in that only the $conv_n$ operation gets replaced. The input-specific CondConv filters are generated with a routing function. The concrete definition of CondConv at layer n takes the following form:

$$conv_{cond}(x, W) = (\alpha_1 W_1 + \cdots + \alpha_n W_n) * x$$

$$f_n(x_{n-1}) = \sigma_n(\gamma_n \cdot norm(conv_{cond}(x_{n-1}, W_n)) + \beta_n), \tag{3.2}$$

where α values represent the expert scalar weights, n is the number of experts, and σ is the activation function. Note that each weight W in the CondConv operation has the same dimension as the weight in the original layer. While the capacity of each layer is increased, CondConv still incurs computational costs on par with conventional convolution

operation since it performs only one convolution after linearly combining the experts. The small overhead to pay is the generation of routing weights and the linear combination of n experts using them before applying the convolution.

The routing function is an integral component for CondConv to generate input-specific filters while still being computationally efficient. It is comprised of steps, such as Global Average Pooling (GAP) operation, fully-connected (FC) layer operation, and sigmoid activation function. The GAP step reduces the dimension of the previous layer input into a 1-d vector to reduce computation. Assuming each filter has n experts, the FC layer, initialized with weight $R \in \mathbb{R}^{C_{in}} \rightarrow \mathbb{R}^n$, takes in the vector to generate n scalar values, subsequently passed through a sigmoid activation function to produce routing weights. The sigmoid activation function is chosen for this case, since empirically, it has better performance than other choices such as softmax, etc. The routing function design is detailed below with x being the previous layer input.

$$\alpha = Sigmoid(GAP(x)R). \tag{3.3}$$

It is possible to use more complex routing functions like multi-layer perceptrons (MLPs) to potentially obtain better expert weights at the cost of increased computation. However, these complex functions tend to be prone to overfit so we stick to using a single FC layer. With this in-depth understanding of creating CondConv filters, in the next subsection, we explore the finer details of FBS and how expert filters allow for a higher pruning rate.

3.3.3 Feature Boosting and Suppression with Channel Saliencies

Recall from Equation 3.2 that γ_n and β_n are trainable parameters. FBS essentially exploits these trainable parameters to dynamically boost and suppress channels. γ_n can be thought of a batch normalization scale factor and is the mechanism that decides which channels will

be zeroed out. It is calculated for layer n as follows:

$$\gamma_n(x_{n-1}) = wta_{\lceil dC_n \rceil}(g_n(x_{n-1})). \tag{3.4}$$

Here, $wta_k(\cdot)$ is a k-winner take all activation function which selects the top k entries in the input and zeros out the rest. The number of channels that are zeroed out are determined by a hyperparameter known as the gate density ratio, d, which controls the level of pruning. In other words, $wta_{\lceil dC_n \rceil}$ takes the top dC_n channels and only performs the convolution with respect to the surviving channels. It important to note that none of the expert scalar weights, αs, are pruned but rather the input-specific CondConv filters.

FBS has a similar routing mechanism to CondConv called the channel saliency predictor, represented as $g_n(x_{n-1})$ in Equation 3.2. It is also implemented as a low-cost FC layer appended to the convolutional layers. The predictor takes the previous layer input and calculates the channel importance so the wta activation function can rank the channels accordingly. The predictor maps the previous layer input to a 1-d vector by subsampling to reduce computation cost, $ss : \mathbb{R}^{C_{n-1} \times H \times W} \rightarrow \mathbb{R}^{C_n}$:

$$ss(x_{n-1}) = \frac{1}{HW}[s(x^1_{n-1})s(x^2_{n-1}) \cdots s(x^C_{n-1})]. \tag{3.5}$$

$s(x^c_{n-1})$ reduces the c^{th} channel of the input to a scalar through the ℓ_1 norm, $||x_{n-1}||$, for instance which is effectively a GAP operation. Various choices of norms for the subsampling function, such as $||\ell_2||$, ℓ_∞, etc., can be considered but we select the ℓ_1 norm for simplicity. After the subsampling operation, the channel saliency predictor is constructed with a weight tensor, $\phi_n \in \mathbb{R}^{C^n \times C^{n-1}}$ and bias, $\rho_n \in \mathbb{R}^{C^n}$:

$$g_n(x_{n-1}) = \sigma_n(ss(x_{n-1})\phi_n + \rho_n). \tag{3.6}$$

Just as in CondConv, the routing function can be more complex than a single FC layer but

again is prone to overfitting; thus, we choose one single FC layer for the predictor. Since both routing functions use the same input, CondConv will generate filters that will more readily capture the most relevant features and FBS will aid in this process by eliminating the irrelevant channels. This synergy between these methods is the enabling factor for FBS to use a higher pruning rate and win on computational savings with negligible loss in accuracy.

3.3.4 Training Methodology

From the onset of training, FBS is applied to prune a network with CondConv layers while CondConv layers are simultaneously being trained as opposed to training the CondConv layers in absence of FBS first and fine tuning the resultant network with turning on FBS later. Since the *wta* activation function is a piecewise and differentiable, standard gradient descent (GD) training methods can be used to train the network. For the loss function, we choose the standard cross entropy loss with L_2 weight decay, $\lambda_{wd} = 10^{-4}$. We also impose a Lasso constraint on the the output of the channel saliency predictor $g_l(x_{l-1}) : \lambda \sum_{l=1}^{n} ||g(x_{l-1})||_1$, where we set $\lambda = 10^{-8}$.

3.4 Experiments and Results

3.4.1 Architecture, Datasets and Experimental Setup

We primarily conduct experiments on the CIFAR-10 dataset, which is a standard vision benchmark in deep learning. The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [99].

For the baseline model, we use M-CifarNet, a custom 9-layer CNN. This M-CifarNet model is adopted from the state-of-the-art FBS work [19]. The baseline M-CifarNet model achieves an accuracy of 94.84%. We train all our models in the TensorFlow framework. We replace the convolutional filters in the M-CifarNet model with CondConv filters, adding

the appropriate routing functional. In all our experiments, we fix the number of CondConv experts in each layer (n) to 4.

Gate Density (d)	MACs (MOps)	MAC savings
1.0	174.3	1x
0.9	145.5	1.20x
0.7	88.3	1.97x
0.5	45.3	3.85x
0.3	16.5	10.58x
0.25	11.5	15.1x

Table 3.1: Number of MAC operations and savings at different gate densities (equivalent to (1 - d) pruning rates under SCP)

We train and apply FBS to the baseline model and the model with CondConv filters for 480 epochs. We use standard data augmentation techniques such as random flip, random crop (with a value of 4), data whitening, in addition to cutout regularization, setting the cutout parameter to 18 [100]. We use RMSProp [101] as the optimizer with the learning rate initially set to 0.01 and a cosine annealing decay learning rate scheduler [102]. We set our batch size to 256 for all the networks we trained. We initialize all network parameters with He initialization [103]. We use the swish activation function [104] for all the layers in the model with the exception of the CondConv routing function, which exclusively uses sigmoid. We also applied gradient clipping to ensure smoother convergence during the training process of our FBS-pruned CondConv models. We measure performance as top-1 accuracy on the test set and the computational cost in multiple-accumulate (MACs) operations. In Table 3.1, we show the MACs operations for FBS-pruned CondConv at various gate densities as well as its associated MAC savings. We omit reporting the MACs and savings for FBS since FBS will incur a similar computation cost to that of the FBS-pruned CondConv since the additional overhead of the routing function computation and the combination of experts is marginal (2.94% at 0.5 gate density over FBS) in comparison to the cost of the convolution operation at different gate densities.

Gate Density (d)	SCP Accuracy (%)	SCP CondConv Accuracy (%)	FBS-pruned CondConv Accuracy (%)
0.9	94.91	94.85	95.13
0.7	94.17	94.03	94.41
0.5	93.12	93.43	94.06
0.3	90.32	90.88	92.05
0.25	88.56	89.83	91.19

Table 3.2: Comparison of FBS-pruned CondConv with SCP-pruned CondConv Accuracy and SCP at different MAC budgets.

3.4.2 CIFAR-10 Classification

We demonstrate the accuracy of FBS-pruned CondConv at different gate densities and compare this against the baseline FBS in Figure 3.3. The y-axis reports the accuracy, while the x-axis reports the number of operations (measured in MACs) at different gate densities. At every gate density, FBS-pruned CondConv clearly outperforms FBS in its ability to retain accuracy at the expense of reduced computation cost. For example, FBS-pruned CondConv at gate density $d = 0.5$ is able to offer similar accuracy to that of baseline FBS at $d = 0.7$ while incurring about 43M fewer MAC operations. Similarly, FBS-pruned CondConv at $d = 0.25$ can match the accuracy of baseline FBS at $d = 0.3$ while incurring about 5M fewer MAC operations. The trend in Figure 3.3 illustrates that as the pruning rate becomes more aggressive, the performance gap between FBS-pruned CondConv and FBS becomes wider. This result affirms our hypothesis that the input-specific filters from CondConv enable FBS to prune aggressively with negligible accuracy degradation and supports the assertion that a smaller set of expert filters can accomplish the same task as a larger number of generic filters.

The other baseline is static channel pruning (SCP) where we reduce the capacity of the model prior to training to match the inference budget of FBS with respect to a selected gate density ratio. In order to validate the potential of FBS-pruned CondConv over statically channel pruned CondConv network (SCP CondConv), we trained a CondConv model with statically pruning away channels proportional to different gate density ratios of FBS. The

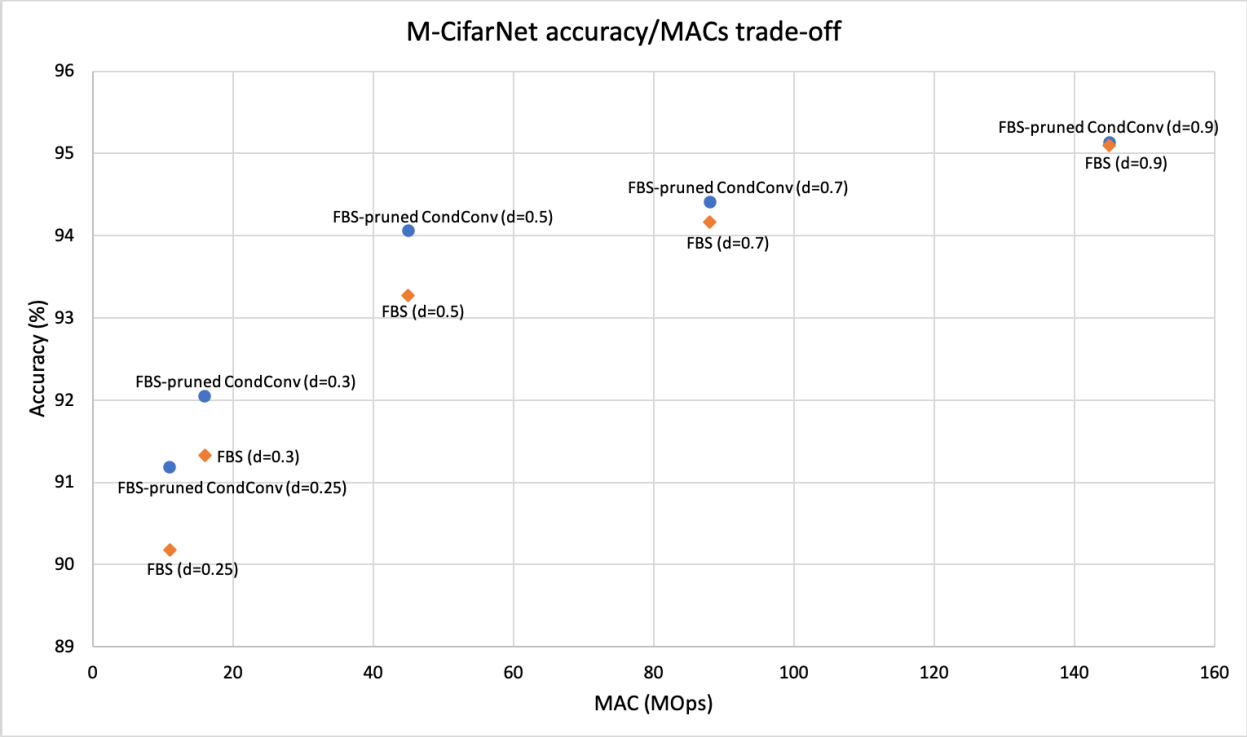


Figure 3.3: Accuracy/MACs trade-off for baseline FBS and the FBS-pruned CondConv model. For lower MAC operation points, FBS-pruned CondConv's accuracy is significantly higher than that of the FBS baseline. d is the gate density of FBS mechanism.

accuracy results are shown in Table 3.2. FBS-pruned CondConv significantly outperforms SCP-pruned CondConv with accuracy margins of 1.17% and 1.36% at the highest reported pruning rates of $d = 0.3$ and $d = 0.25$, respectively. This analysis highlights the advantage that conditional execution techniques such as FBS provides over static pruning methods by maintaining model capacity and minimizing the accuracy penalty.

3.4.3 Under-utilization of Conditional Filters

When analyzing FBS at various gate densities, we observed that FBS has a side effect of not utilizing all filters in every layer. We compare FBS against a network which has the same computation budget by statically pruning away channels proportional to the gate density ratio. The expectation is that FBS should outperform SCP at all gate densities because FBS has a larger parameter budget, which if it is using correctly should translate into computational savings with little to no accuracy loss [19, 22]. However, we perform this comparison and report the results in Table 3.2.

Looking closer at Table 3.2, an interesting observation is that FBS outperforms SCP by 0.96% and 0.56%, respectively for gate densities 0.3 and 0.25, but not for the less constrained MAC settings (e.g. higher gate densities). This result motivated us to question how differently is FBS at these various MAC budgets. Figure 3.4 is a sorted count of the filter activations at each layer on the CIFAR10 testset. We see that for most of the intermediate layers the parameter budget used by FBS is similar to SCP, leaving large amounts of opportunity on the table. FBS wins over SCP at lower gate densities in the first and last layers where all the filters are activated for all examples.

This phenomenon has also been observed in [22] who reported certain gates in networks are always off. Their method, batch-shaping, resolves this issue by matching the marginal aggregate posteriors features captured in the network to a pre-specified prior distribution. This tool is complementary to both FBS and ConvCond+FBS models and will be a focal point for future work. Other potential solutions to consider for this problem is designing

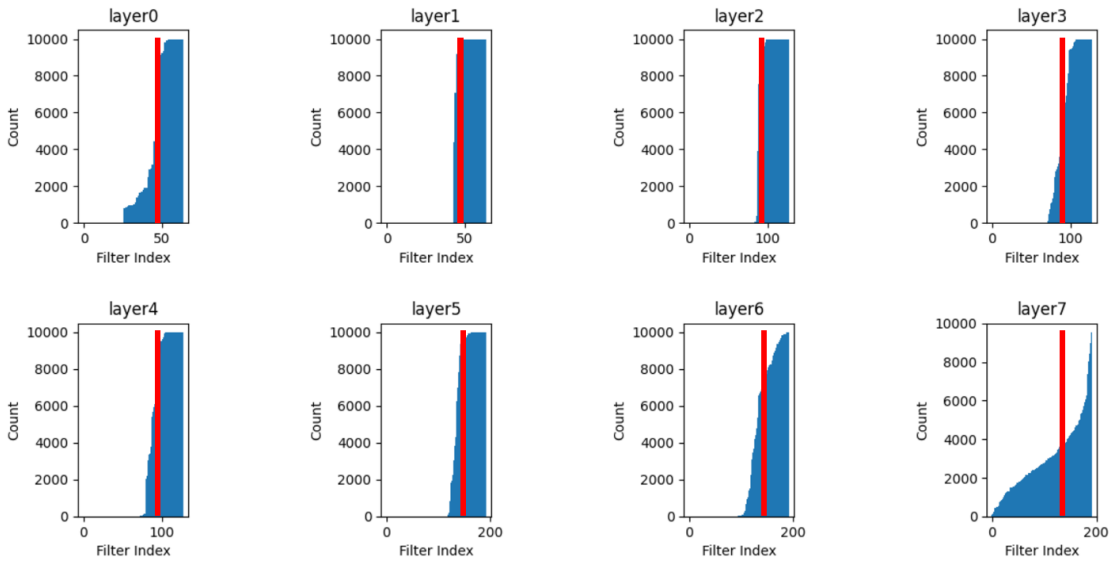


Figure 3.4: Sorted count of filter activations a FBS model across all layers at gate density $d = 0.3$ on the CIFAR10 testset. The red line, from the start to the end of each layer subplot, represents the parameter budget of a static channel pruning baseline. In both plots, we see that in intermediate layers, a sizeable portion of filters are not activated for any test example. Furthermore, the number filters that are active is comparable with the SCP baseline.

a regularizer which will promote filter diversity so filters that were pruned still see some activity, which is left for future work.

3.5 Conclusion/Future Work

In this work, we demonstrate the synergy between two state-of-the-art conditional execution techniques to achieve aggressive pruning ability of convolutional networks. This in turn results in large computation savings over existing dynamic execution mechanisms while ensuring little to no loss in model accuracy. The input-specific nature of CondConv filters as well as their unique property of separating classes by experts addresses the destructive interference issue that FBS experiences in high pruning regimes. We conducted CIFAR-10 experiments showing FBS-pruned CondConv can achieve up to 47.2% reduction in computational cost at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over FBS. Note that this work was targeted towards achieving high accuracy while minimiz-

ing computational costs albeit at the cost of increasing model size from the use of CondConv kernels.

There are several avenues for future work:

1. This particular work is targeted at applications where memory is not a constraint and the network can be large. However, more times than not, embedded devices have low on-chip memory so if we want FBS-pruned CondConv to apply to these use cases, we need to integrate it with other static inference schemes such as weight-magnitude pruning, quantization, and tensor factorization.

2. FBS-pruned CondConv combines two conditional approaches together to exploit the best of both methods. There are techniques in the conditional execution literature such as channel gating/precision gating that recognize that certain spatial position across the input channels encode more information than others, which is absent in the current mixture. Therefore, we want to incorporate these methods into the FBS-pruned CondConv model to push the boundary of the accuracy-MACs tradeoff further.

3. The insight from Figure 3.4 reveals that a surprising number of filters in the intermediate layers in the network are not used in the classifications of the test images. A solution to this phenomenon is introduce a filter diversity regularization term as proposed above to encourage a uniform participation of filters; the other fix is to vary the pruning rates and number of experts across different layers, potentially making them learnable. This will be an essential step when we want to apply our approach to larger models and ImageNet-like challenging datasets.

Chapter 4

Accelerating Deep Learning with Dynamic Data Pruning

This chapter introduces Dynamic Data Pruning. Specifically, section 4.1 introduces the notion of accelerating network training and the classification of samples in the dataset between easy samples and hard samples. The following section 4.2 discusses related work from active learning and static data pruning which is the paradigm of data pruning we wish to compare against. Section 5.3 discusses the framework for dynamic data pruning and the various methods borrowed from active learning and reinforcement learning used for sample selection. The next section 4.4 disseminates results about the aforementioned methods and reveals that samples should be categorized into 3 distinct sections: always, sometimes and never samples. This observation is able to help us in understanding why dynamic pruning methods are better than the static counterparts. The final section 4.5 summarizes the findings from dynamic data pruning and lays groundwork for societal impact, limitations, and future work.

4.1 Introduction

A growing body of literature recognizes the immense scale of modern deep learning (DL) [105, 106], both in model complexity and dataset size. The DL training paradigm utilizes clusters of GPUs and special accelerators for days or weeks at a time. This trend deters independent researchers from applying state-of-the-art techniques to novel datasets and applications, and even large research organizations accept this approach at significant costs.

Currently accepted methods [27] target the per-iteration penalty of evaluating the model during training; however, not as much effort has been spent on reducing the total number of training iterations. Since even simple datasets [28] require hundreds of epochs over tens of thousands of samples, eliminating a non-essential subset of data presents a promising opportunity for efficiency.

Work from other DL domains suggest that only subset of the data influences the decision boundary and contributes a majority of the loss incurred by the model [29, 30]. Furthermore, curriculum learning [31] asserts that samples can be ranked which might allow us to prune redundant "easy" samples. Prior work on data pruning [30, 32] take advantage of this property to eliminate a majority of the dataset without incurring significant performance loss. Unfortunately, these methods run their scoring algorithm prior to training and require one or more passes over the dataset. When we include the cost of scoring the samples, the total run time exceeds the time it takes to do a single conventional training run. This prevents researchers from utilizing the prior work on new, non-standard datasets.

In our work, we dynamically select a subset of the data at fixed checkpoints throughout training. We make the following new observations and contributions:

1. By counting every time each sample is selected across all scoring checkpoints, we find that a dataset can be qualitatively split into three groups — always samples, never samples, and sometimes samples (see 4.1). Always samples are selected at nearly every scoring checkpoint. Similarly, never samples are rarely selected. Sometimes

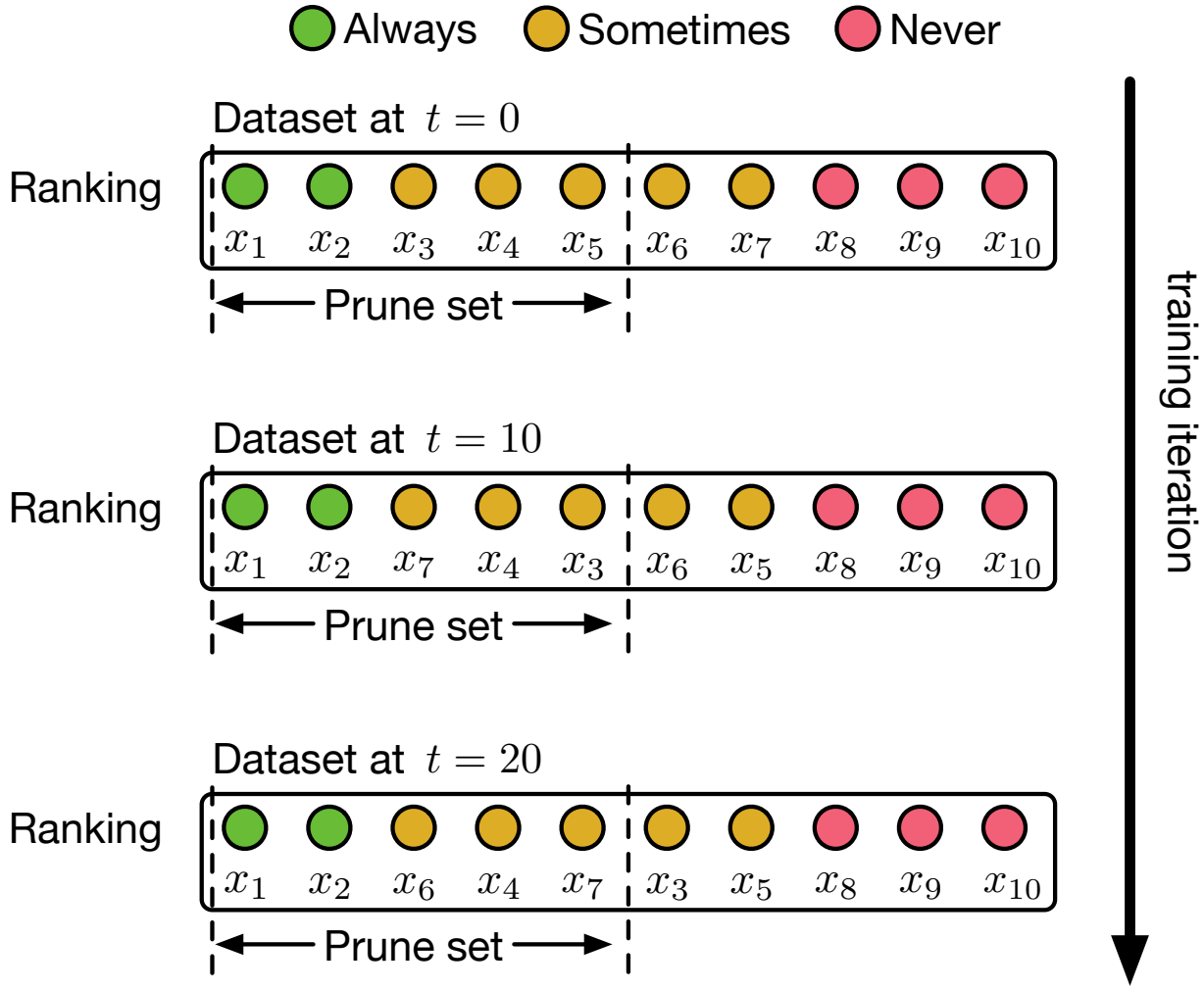


Figure 4.1: Dynamic data pruning presents multiple opportunities for a sample to be selected for training. This illustration shows our finding that datasets are separated into three groups — samples that are always selected, samples that are never selected, and samples that are selected only some of the time. Static methods fail to effectively target the last group, since their ranking varies during training.

samples are selected at only some checkpoints, and their inclusion is highly variable across different training runs. Static pruning methods, like the prior work, can identify always or never samples but fail to effectively target sometimes samples. In fact, we find that randomly selecting a subset of the data at each checkpoint is more effective than the static baselines.

2. Given this grouping of the dataset, we design a dynamic scoring mechanism based on per-sample loss. Despite scoring more frequently, our mechanism reduces the total run time including the cost of scoring, while the prior work typically increases it. Moreover, at aggressive pruning rates, we obtain higher final test accuracies on CIFAR-10, CIFAR-100, CINIC-10, and a synthetically imbalanced variant of CIFAR-10.

3. Since the sometimes samples vary in importance across training runs, we note that the optimal dynamic scoring selection is tightly coupled to the model training trajectory. So, we re-frame the data pruning problem as a decision making process. Under this lens, we propose two variants of our scoring mechanism that borrow from ϵ-greedy and upper confidence bound (UCB) algorithms in reinforcement learning. With these additional improvements, we obtain even higher performance at aggressive pruning rates even when the dataset is imbalanced.

4.2 Related Work

4.2.1 Curriculum learning and active learning

The idea that certain samples are more influential than others has been observed for some time in curriculum learning and active learning. Curriculum learning exploits this to present samples to the model under a favorable schedule during training to obtain better generalization [31, 107]. Our work has a similar objective, but we are concerned with ranking samples in order to eliminate them instead of schedule them.

Active learning attempts to rank new, unlabeled samples based on redundancy instead of difficulty. One of the computationally cheap techniques in this setting is uncertainty sampling [49], which we use in our work as well. Unlike active learning, in data pruning we have access to all the samples and their labels.

4.2.2 Static data pruning

Several works have studied how to statically prune a dataset prior to training. Toneva *et al.*[30] proposed the concept of forget scores, which track the number of times an example transitions from classified correctly to misclassified throughout training. They classified the data into two sets — forgettable examples (frequently misclassified) and unforgettable examples (rarely misclassified). By pruning samples based on the forget scores, they eliminate up to 30% of the data with no loss in performance on CIFAR-10. An important limitation of their method is the need to perform a full training run to obtain the forget scores. Coleman *et al.*[108] build on forget scores in their work called selection via proxy (SVP). SVP uses a smaller proxy model to rank examples in order to choose a subset that can be used to train a larger model. By re-using the forget scores algorithm and training a smaller network for fewer epochs, they obtain a subset of samples to train a larger model with no loss in performance even when the subset is 50% of the original dataset size. Unlike the forget score-based methods, Paul *et al.*[32] use the gradient norm-based (GraNd) scores of individual samples averaged across 10 different random models to prune data at initialization. They also prune up to 50% of CIFAR-10 with no loss in accuracy. Even though the SVP and GraNd methods require less computation than the original forget score work to obtain a ranking, the results are too noisy to statically prune the data. Thus, they must rely on more iterations or averaging across trials to obtain the final subset of samples. As a result, all three methods incur a high overhead for pruning that increases the total run time to train the final model. This prevents these methods from being applied to new datasets for the first time. Since our method is dynamic, it can be applied directly to new data without *a*

priori information.

4.3 Methods

In this section, we describe our approach to data pruning. First, we re-frame data pruning as a dynamic decision making process. Next, we present our scoring mechanism based on filtered uncertainty sampling. Finally, we discuss how the ϵ-greedy and upper confidence bound algorithms used in reinforcement learning can be used on top of our scoring mechanism to further improve the final test accuracy.

4.3.1 Dynamic data pruning

Static data pruning assumes that the optimal subset of samples depends primarily on the data distribution. Model training training is a means to an end, and the trajectory of a specific model does not factor into the final pruned dataset. In our work, instead of approaching data pruning as a one-time step, we view it as a dynamic process that is coupled to a given model's training run. The total number of training epochs is divided by fixed pruning checkpoints. At each checkpoint, an algorithm may observe the current performance of the model on the full dataset. Given this observation and information retained from previous checkpoints, the algorithm makes a decision on which points to use for training until the next checkpoint. This process is described in Algorithm 1.

Algorithm 1 Dynamic data pruning process

Require: k = # of samples to keep, s = pruning selection criterion, T_p = pruning period, T = total # of epochs, dataset X (w/ N samples), classifier f_θ
  for T/T_p passes **do**
    Select subset, $X_p = s(f_\theta, X, k)$
    for T_p epochs **do**
      Train f_θ on X_p
    end for
  end for

4.3.2 Pruning selection criterion

In the next subsections, we describe the different selection criteria, s, that are proposed in our work.

Uncertainty sampling with EMA

Uncertainty sampling preferentially selects samples for which the model's current prediction is least confident. In general, it is quick to compute and identifies points close to the estimated decision boundary which is a measure of the informativeness of a sample. Specifically, we use the per-sample cross entropy loss for assigning a score to individual sample x with label y over C classes:

$$s_{\text{uncertainty}}(x) = \sum_{i=1}^{C} -y_i \log \left(\frac{\exp(f_\theta(x_i))}{\sum_{j=1}^{C} \exp(f_\theta(x_j))} \right) \tag{4.1}$$

This scoring criterion is similar to the EL2N score from Paul *et al.*[32]. In that work, the final EL2N score is obtained by averaging the uncertainty over several trials to reduce the noise given in a single trial. We observe the same noisy behavior across pruning checkpoints in our work. A network's decision boundary can vary dramatically, especially with respect to points which are included in one checkpoint and excluded in another. Since we are pruning dynamically, instead of a static average, we use an exponential moving average (EMA) filter:

$$s_{\text{ema}}(x) = \alpha s_{\text{uncertainty}}(x) + (1 - \alpha) s_{\text{previous ema}}(x) \tag{4.2}$$

where $s_{\text{previous ema}}(x)$ is the value of $s_{\text{ema}}(x)$ from the previous checkpoint and α is the constant linearly interpolating between two quantities.

ϵ-greedy approach

So far, our decision making criterion has been fairly simple. The points with the highest average loss are selected at each checkpoint. This approach can be viewed as a greedy

solution to a multi-armed bandit problem from reinforcement learning [56]. Each sample is an arm associated with some unknown value (i.e. score). By selecting points, we make k pulls of various arms, and when we measure the uncertainty on the next checkpoint, we make a noisy observation of the value of every sample in the dataset.

Selecting the samples with the highest uncertainty at every checkpoint corresponds to a greedy decision. Since we maximally exploit our current information, we might under-select some points and obtain a suboptimal solution. A simple fix is to encourage more exploration by randomly selecting points which do not currently have the highest score. This is referred to as an ϵ-greedy approach in the bandit setting [56].

Concretely, at every frequency checkpoint, $(1-\epsilon)k$ points are selected using 4.2, and ϵk points are selected from the remaining data uniformly at random. Our results show that this additional exploration can help boost performance even further.

Upper-confidence bound approach

While the ϵ-greedy approach does introduce more exploration, it does so in a random manner. A more directed approach would be to prioritize samples for which the running variance of 4.1 is high even when the mean is low. In this way, when we do explore, we choose to focus on the samples who's scores are poorly estimated.

The upper-confidence bound (UCB) [56] algorithm is another RL method that selects arms based on the mean value and the variance. Assuming the initial mean scores are aggregated from the random model and the starting variance is zero, we compute the variance based on Welford's algorithm [109]:

$$\begin{aligned} \text{var}(x) = (1-\alpha)\text{var}_{\text{previous}}(x) \\ + \alpha(s_{\text{uncertainty}}(x) - s_{\text{previous ema}}(x))^2 \end{aligned} \tag{4.3}$$

With the mean and variance computed, the final score of the sample is computed by adding

the two quantities together:

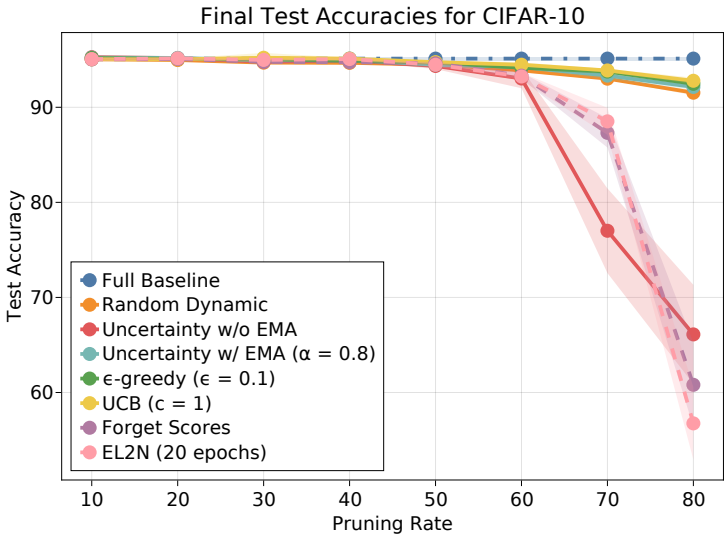
$$s_{\text{ucb}}(x) = s_{\text{ema}}(x) + c\text{var}(x) \tag{4.4}$$

where c is a hyper-parameter which dictates the variance's influence on the total score.

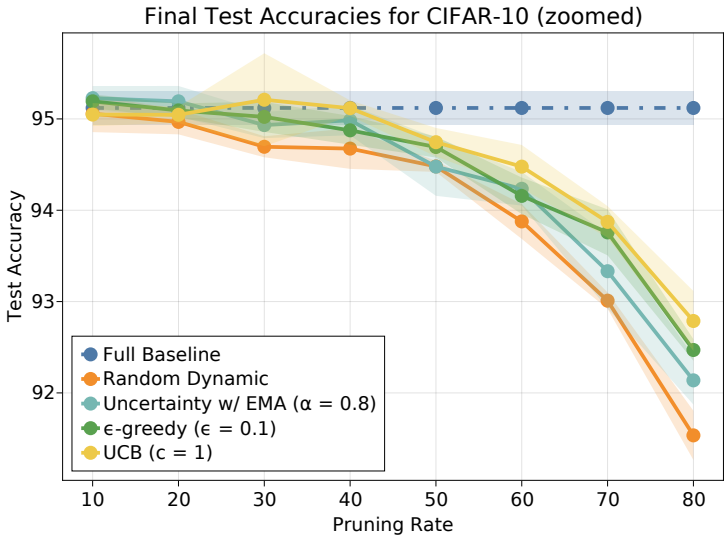
4.4 Results and discussion

We experimentally evaluate our methods on the CIFAR-10 and CIFAR-100 datasets [28] (following the setup in Paul *et al.*[32]). Each image is augmented with standard transformations such as normalizing the images by per channel mean and standard deviation, padding by 4 pixels, random cropping to 32 by 32 pixels, and horizontal flipping with probability 0.5. We use ResNet-18 [110] with the initial 7x7 convolutional layer and 2x2 pooling layer swapped out for a 3x3 convolution layer and a 1x1 pooling layer to account for change in image resolution. We train our model for 200 epochs with the stochastic gradient descent (SGD) optimizer with an initial learning rate = 0.1 with Nesterov momentum = 0.9, weight decay = 0.0005, and batch size = 128. The learning rate was decayed by a factor of 5 at 60, 120, and 160 epochs. Our implementation is based on PyTorch [111] and all experiments were conducted on a 11GB NVIDIA GeForce RTX 2080 Ti GPUs. For CIFAR100, we use the same experimental setup but swap out ResNet-18 for ResNet-34.

For our baselines, we compare against conventional training with the full dataset, forget scores, and EL2N scores. For the forget scores, we train the model for 200 epochs and compute the forget scores as per the prior work. For the EL2N scores, we also follow the scoring algorithm laid out in the prior work and average over 10 independent models trained for 20 epochs each. After obtaining the forget and EL2N scores, we statically prune the data set by selecting the top scoring samples based on the designated pruning rate. For all our experiments, we run each method over 4 independent random seeds. We also present results on the distribution of selected samples to corroborate our findings on sometimes



(a) CIFAR10 accuracy



(b) CIFAR10 accuracy (zoomed)

Figure 4.2: Dynamic data pruning applied to CIFAR-10 with ResNet-18. 4.2a shows the final test accuracies for each method, 4.2b shows the same data without the under-performing methods

samples. For these results, we use the Turing package [112] in the Julia language [113] to fit a mixture model with Bayesian MCMC methods to delineate between sample groups. For all distributions, we use the Metropolis-Hastings algorithm [114] with 1000 samples and 3 chains (averaged).

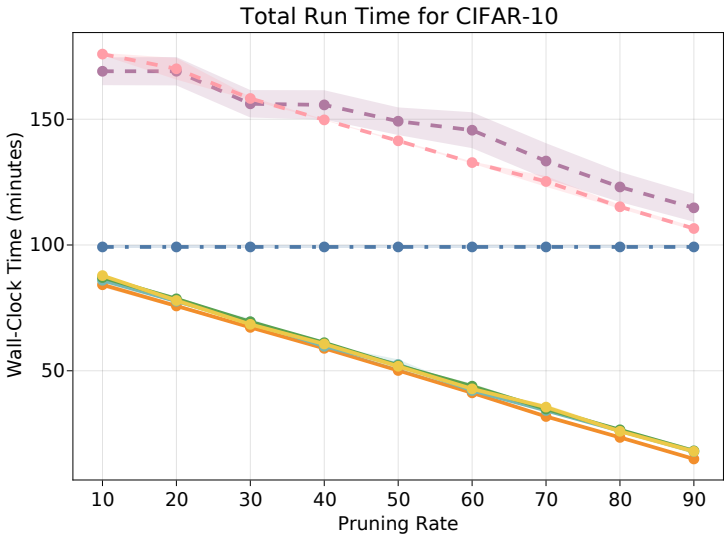


Figure 4.3: CIFAR10 run time comparing dynamic channel pruning with the baseline static methods.

4.4.1 CIFAR-10 results

Figure 4.2 shows the result of dynamic data pruning with the various selection methods introduced in the prior section. We sweep the pruning rate with the pruning period (T_p) set to every 10 epochs. For all methods that use an EMA, we set $\alpha = 0.8$.

Below 50% of the data pruned, all methods appear to perform equally well. At more aggressive rates, the static baselines and the uncertainty without the EMA drop dramatically in performance. Our proposed methods continue to maintain reasonable performance even when 80% of the dataset is pruned. As expected, the ϵ-greedy and UCB improve the performance slightly over the uncertainty with the EMA.

In fig. 4.3, we plot the measured wall-clock time for each method. Since the static methods perform pruning prior to training with an overhead equivalent to training on the full data, their wall-clock time is nearly double the full baseline (when omitting the static overhead, the wall-clock time is on par with random pruning). In contrast, all the dynamic methods reduce the wall-clock time even at modest pruning rates.

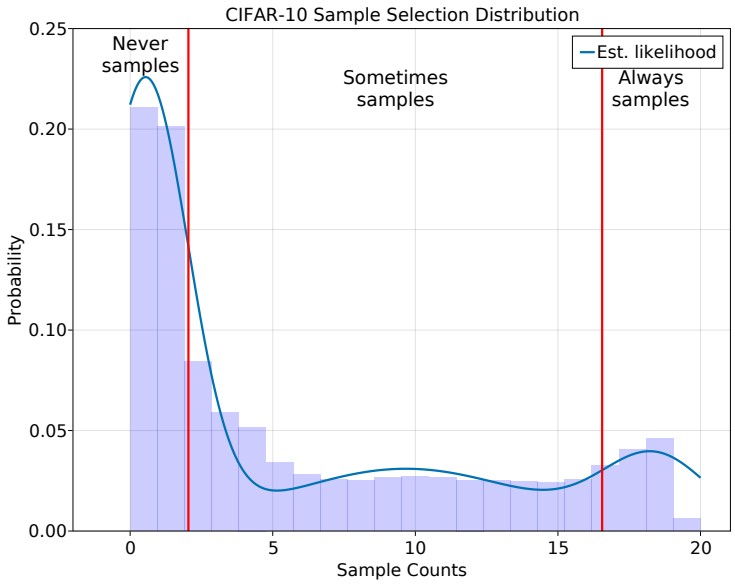


Figure 4.4: The density function (of a mixture model) of sample counts from 4 independent CIFAR-10 runs during our uncertainty with EMA method at 70% pruning. The horizontal axis represents the number of times a sample has been counted by the selection process. The data is separated into three groups—samples that are always selected, samples that are selected only some of the time, and samples that are never selected. The red lines are denote one standard deviation from the mean of the left and right mixture components. Static methods fail to effectively target the middle group.

4.4.2 Unreasonable effectiveness of random pruning

The most surprising result in fig. 4.2a is the performance of the random dynamic pruning algorithm. This method randomly samples the training data at each checkpoint without any consideration to the uncertainty or previous checkpoints. Still, the approach is able to outperform the static baselines, and it suffers only a 3.5% drop in accuracy at a pruning rate of 80%. While this is unexpected in the context of this paper, it does match intuition from first principles. The very basis of SGD training relies on the ability to train on randomly sampled batches. If there was a strong bias in the relative importance of samples, then most DL training would be quite inefficient.

This is corroborated by our findings on the existence of "sometimes" points. When performing each of the trials to obtain fig. 4.2, we count the number of times each sample is selected across all checkpoints and trials. In Fig. 4.4, we plot the normalized histogram

of these counts. We fit a mixture of three truncated normal distributions (between zero and the total number of checkpoints) using the Turing probabilistic programming language as mentioned at the start of this section. The resulting density function (likelihood) is plotted on top of the histogram in Fig. 4.4.

We observe that the curve can be separated into three distinct regions—always, sometimes, and never samples. Always samples are selected at nearly every checkpoint; moreover, the standard deviation in this region is low based on the range of occupied sample counts and variance of the right Gaussian component after the mixture model fit was completed. Similarly, the never samples are rarely selected, based on the left Gaussian component occupying a small number of sample counts and a tight variance. Like always samples, these points are consistently never selected.

Contrary to always and never samples, sometimes samples are not consistently selected or omitted. The middle mixture component fits a large section of the support. This wide range of sample counts indicates the high variance in this region and shows that the ranking of sometimes samples is not consistent. More importantly, a significant portion of the mass is contained in this component, suggesting that a large subset of the dataset cannot be classified as always or never selected.

Static pruning methods largely target the elimination of never samples. By selecting the top k points in a single pass, they successfully eliminate never samples and keep always samples; however, the small mass within the always region means that static methods must select some arbitrary subset of sometimes samples. In contrast, dynamic methods can rotate their selection of sometimes samples based on which ones are most beneficial to a specific model's training trajectory. Most importantly, since such a large fraction of the dataset does not have a consistent ranking, even randomly sampling these points (as the random dynamic method does) is better than statically committing to a fixed subset of them.

We test this hypothesis by training a model on CIFAR-10 using the uncertainty with EMA method at a pruning rate of 70%, and we save the scores across checkpoints. Then, we train

Table 4.1: The effect of randomly selecting sometimes samples when re-training a model using previous scores.

Method	Accuracy
Original dynamic training	93.33%
Always set + static sometimes set	89.52%
Always set + random sometimes set	92.50%

a new initialization with dynamic pruning using two different subsets of the original dataset. First, we choose the top 30% of samples according to the scores from the final checkpoint. This creates a static policy that uses the same subset at every checkpoint. Second, we choose only the always samples, which make up about 15% of the data, and we choose the remaining samples in the pruning budget uniformly at random for each checkpoint. The difference is shown in table 4.1. Statically choosing the sometimes samples results nearly a 4% performance drop. Dynamically sampling from the remaining samples only results in a 1% performance degradation.

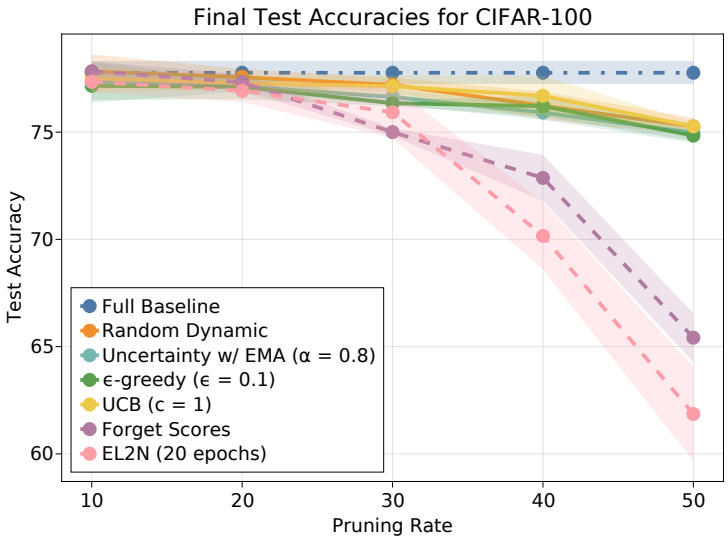
4.4.3 CIFAR-100 results

fig. 4.5 shows the result of dynamic data pruning on CIFAR-100. We retained the same hyper-parameters as in the CIFAR-10 result but omit the uncertainty method without the EMA as its performance was unsatisfactory. We can see that EL2N and forget scores are able to nearly retain the performance up to a pruning rate of 20% but deteriorate rapidly beyond that point.

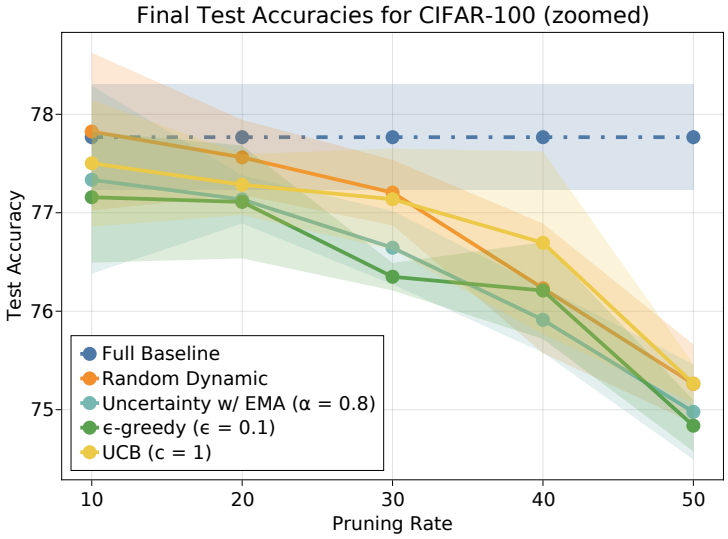
All the dynamic methods outperform the static baselines. While the UCB does appear to have slightly higher accuracy than the random dynamic method at more aggressive pruning rates, most of our methods underperform compared to the random dynamic method. This is not surprising when we consider the sample selection distribution in fig. 4.7.

Unlike CIFAR-10, CIFAR-100 has almost no never points and is far more uniform in its sample selection distribution. This indicates that there is very little opportunity to select

samples more intelligently than a random dynamic method. One plausible explanation for the lack of never samples is the low number of samples per class in CIFAR-100 (500 vs. 5000 for CIFAR-10). Thus, each sample has more influence on the decision boundary, and the intrinsic redundancy in the dataset is much lower.



(a) CIFAR100 accuracy



(b) CIFAR100 accuracy (zoomed)

Figure 4.5: Dynamic data pruning CIFAR-100 with ResNet-34. fig. 4.5a shows the final test accuracies for each method, fig. 4.5b shows the same data without the static methods.

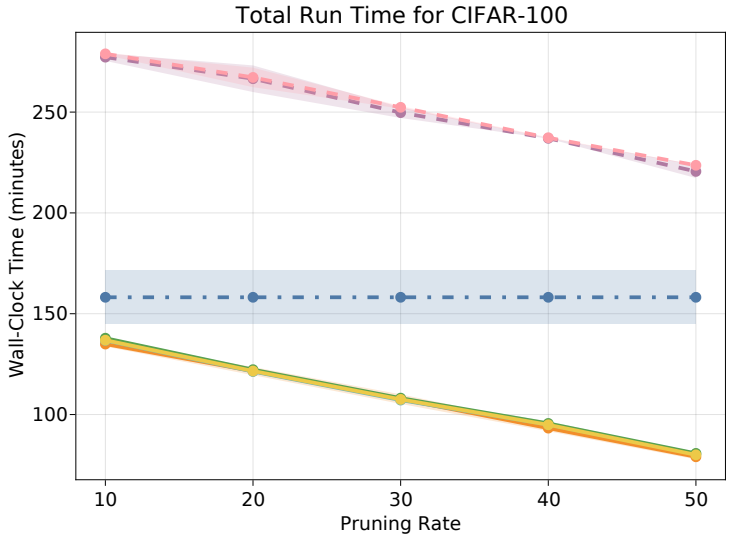


Figure 4.6: CIFAR100 run time

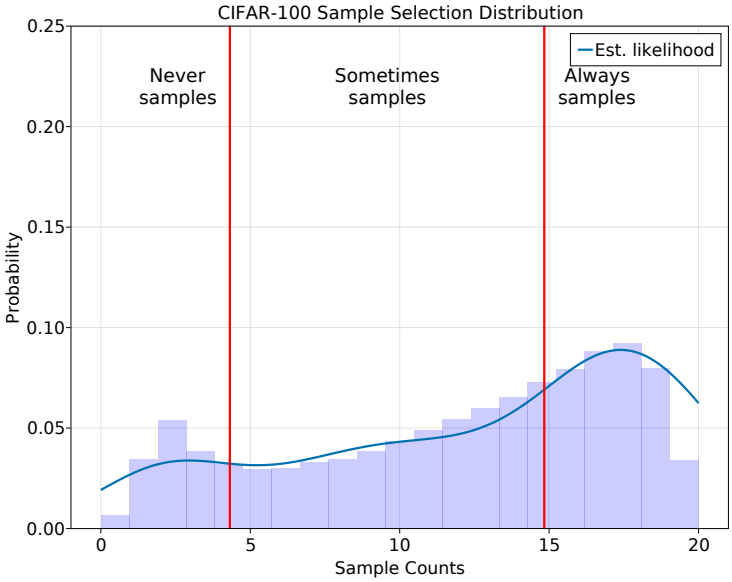


Figure 4.7: Sample selection distribution on CIFAR-100 at 40% pruning with UCB. Unlike CIFAR-10, CIFAR- 100 has very few never samples, which means that the always and sometimes samples takes a large portion of the dataset. Thus, a random dynamic method can prune close to optimally on this dataset.

Table 4.2: The subsampling rates by class for the imbalanced CIFAR-10 dataset.

Class Indices	Subsample Rate
0, 1	25%
2, 3, 4	50%
5, 6	75%
7, 8, 9	100%

4.4.4 Imbalanced CIFAR-10 results

So far, we have studied standard datasets which are well balanced. Our CIFAR-100 results suggest that the number of samples per class can have a large influence on the pruning opportunity. To further explore this phenomenon, we generate a synthetically imbalanced variant of CIFAR-10 by sub-sampling each class according to table 4.2.

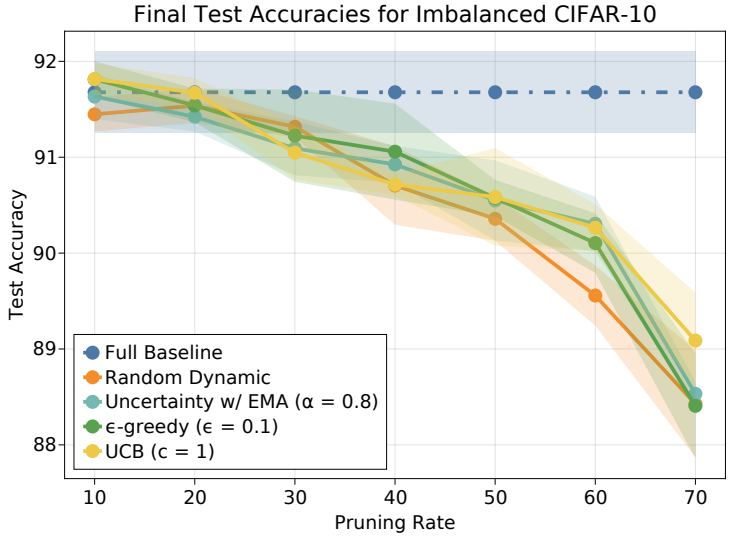


Figure 4.8: Dynamic data pruning a synthetically imbalanced CIFAR-10 with ResNet-18. The horizontal axis refers to the amount of data removed from the training set, and the vertical axis shows the final test accuracies for each method.

The results are shown in fig. 4.8. Our methods outperform the random dynamic method at aggressive pruning rates, but all methods lose performance at modest pruning rates. This is expected based on fig. 4.9. The lack of always/never samples makes this dataset difficult to prune relative to a dataset like CIFAR-10. Unlike CIFAR-100, there does seem to be a

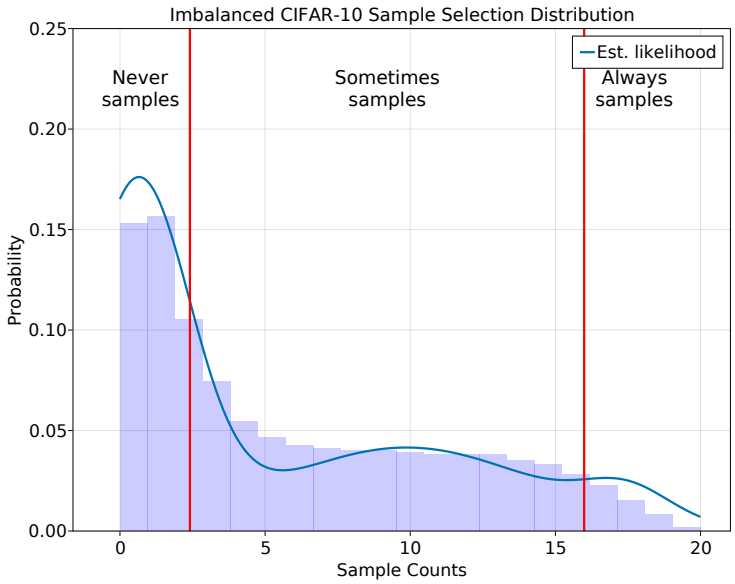


Figure 4.9: Sample selection distribution on the imbalanced CIFAR-10 dataset at 40% pruning with UCB. The lack of always samples contributes to lower accuracies across all methods, but UCB is still able to obtain a slight competitive edge by delineating samples into the three regions more accurately.

larger portion of never samples and consequently more opportunity for pruning. The UCB method is able to target this opportunity once the pruning rate is $> 50\%$.

4.4.5 Downsampled CIFAR-10 results

Similar to the imbalanced CIFAR-10 dataset, we also study a downsampled variant of CIFAR-10. We decrease the number of samples per class from 5000 to 1000 for every class. This mimics CIFAR-100 where the number of samples per class is 500.

The results are shown in fig. 4.10. Similar to the CIFAR-100 results, the uncertainty with EMA and ϵ-greedy methods are worse than the random dynamic method. Only the UCB method matches the random dynamic method in performance. These results are corroborated by fig. 4.11 which shows that the sample selection distribution follows the same trend as CIFAR-100.

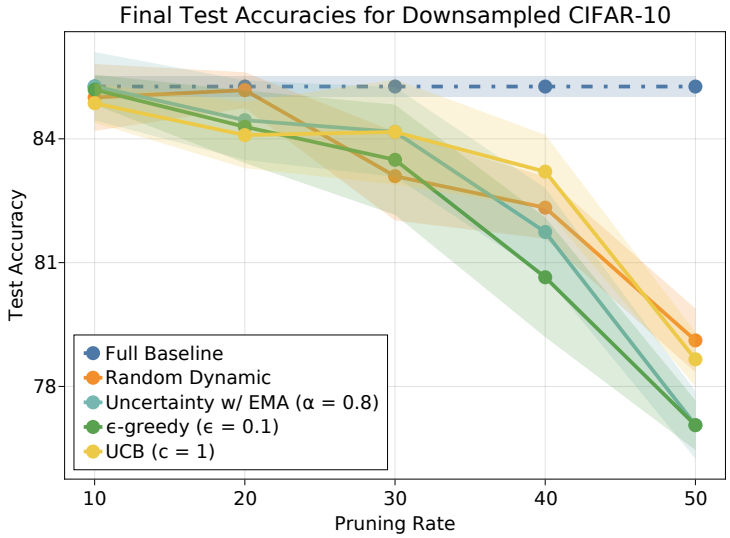


Figure 4.10: Dynamic data pruning a synthetically downsampled CIFAR-10 with ResNet-18. The horizontal axis refers to the amount of data removed from the training set, and the vertical axis shows the final test accuracies for each method.

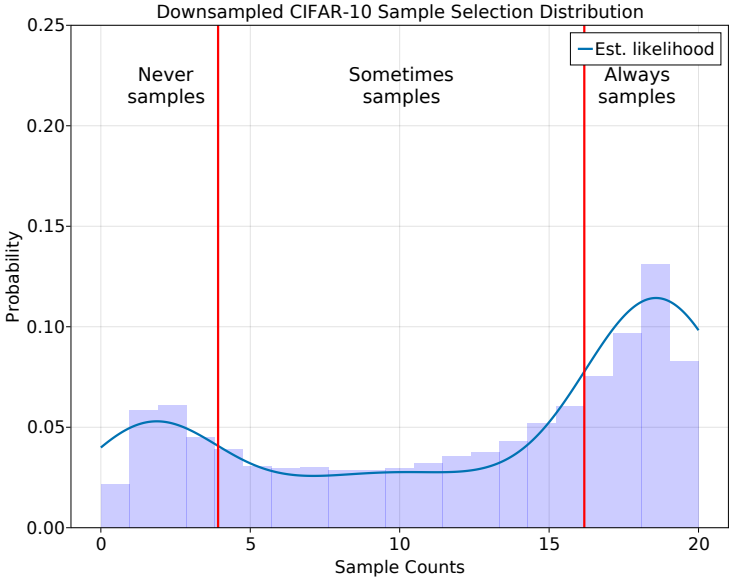
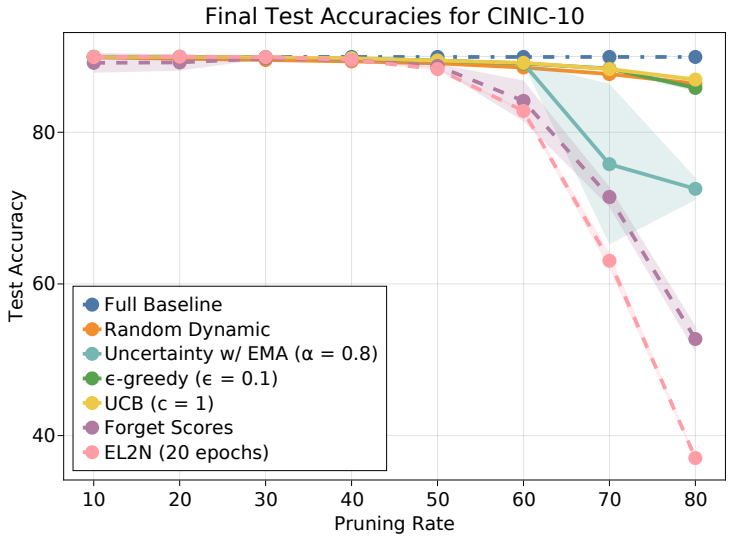
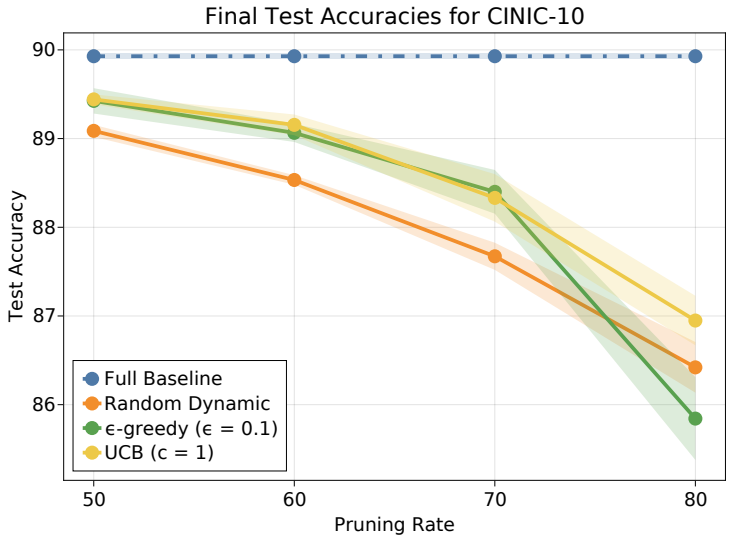


Figure 4.11: Sample selection distribution of samples on the downsampled CIFAR-10 dataset at 40% pruning with UCB. The density function follows a similar shape to CIFAR100 where there are very few never samples.



(a) CINIC-10 accuracy



(b) CINIC-10 accuracy (zoomed)

Figure 4.12: Dynamic data pruning applied to CINIC-10 with ResNet-18. Figure 4.12a shows the final test accuracies for each method, Figure 4.12b shows a zoomed version.

4.4.6 CINIC10

So far, the datasets, CIFAR10 and CIFAR100, we have looked at have been fairly smaller scale. To test our methods on a larger dataset, we use CINIC10 as a testbed. CIFAR10 and CIFAR100 are on opposite ends of difficulty: CIFAR10 has 6000 examples per class whereas CIFAR100 has 600 examples; CINIC10 is in between both of these datasets in

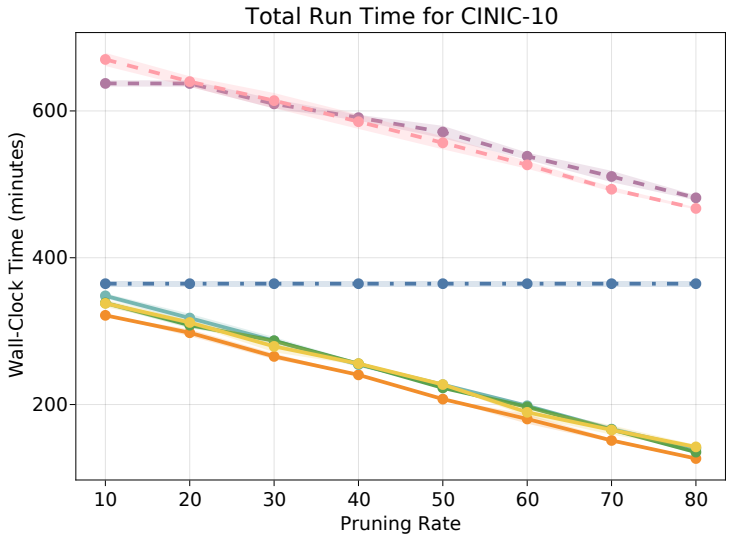


Figure 4.13: CINIC-10 run time including the run time to execute the approach (including scoring cost).

terms of difficulty. It has 270,000 images (4.5 ×) and we combine the training and validation set. It has images sourced from both ImageNet as well as CIFAR10. We apply the same methodology and training framework as we used for all the other datasets.

As shown in fig. 4.12, our methods reduce the training cost by nearly 5× at 80% pruning while remaining close to the baseline accuracy. Similar to all other datasets, UCB method is superior compared to the other dynamic pruning strategies. In comparison, the static methods begin to drop down in accuracy at a pruning rate of 60% and then rapidly drops off as the pruning rate is increased. fig. 4.14 shows the sample selection distribution of UCB at a pruning rate at 70% which follows the trends we see in other datasets: we can delineate the sample by looking at the variance in the scores. The main point is that our methods are able to outperform static approaches by effectively targeting the sometimes samples during training.

4.4.7 Static approaches

In our dynamic methods, we maintain running means and variances of the scores. In contrast, the static EL2N method takes an offline mean across many model initializations. A natural

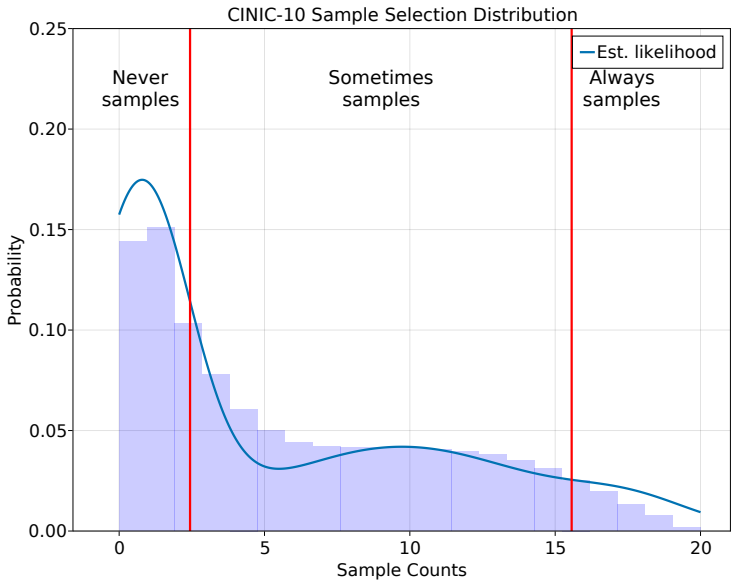


Figure 4.14: Sample selection distribution on the CINIC10 dataset at 70% pruning with UCB. Similar to the imbalanced CIFAR10 dataset, UCB is able to target the never samples and cycle through the sometimes samples to maintain high accuracy even at an aggressive pruning rate.

question is whether our methods can be applied in a static fashion for the same accuracy gains. The results in fig. 4.15 show that this is not the case.

When we statically apply the UCB algorithm to various model initialization trials in the EL2N method, we see no improvement in performance. This makes sense, because the variance in the UCB method is used to dynamically sample under-observed points. In a static variation, this feature of the algorithm is unused.

If we apply an ϵ-greedy approach to the static EL2N policy (i.e. every checkpoint we select $1 - \epsilon$ fraction of the samples using the pre-computed EL2N scores, and we select ϵ fraction randomly), we see an increase in accuracy. Therefore, we see that even a small amount of dynamism improves static scores. The boost in accuracy still under-performs compared to the random dynamic method.

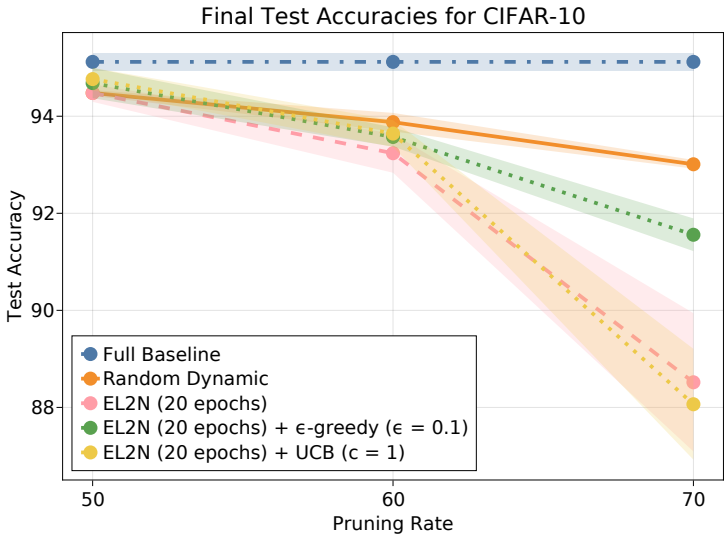


Figure 4.15: Apply our RL methods on top the static EL2N scores. "EL2N + ϵ-greedy" uses the static EL2N scores at each checkpoint but also selects ϵ fraction of the samples randomly. "EL2N + UCB" applies the UCB algorithm statically to the EL2N scores from each model initialization trial.

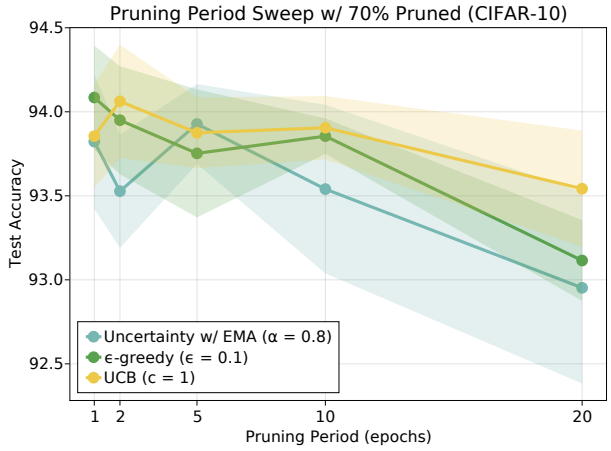
4.4.8 Hyperparameter search

Fig. 4.16 shows the results of sweeping the pruning period (T_p), the exploration rate (ϵ; ϵ-greedy only) and confidence (c; UCB only). The pruning period of 10 epochs provides the best performance vs. wall-clock time tradeoff for all methods. Varying the exploration rate does not impact the accuracy, suggesting that only a small amount of randomness is required. Varying the confidence suggests that incorporating variance into the scoring mechanism is sufficient to increasing the performance, agnostic of c's value.

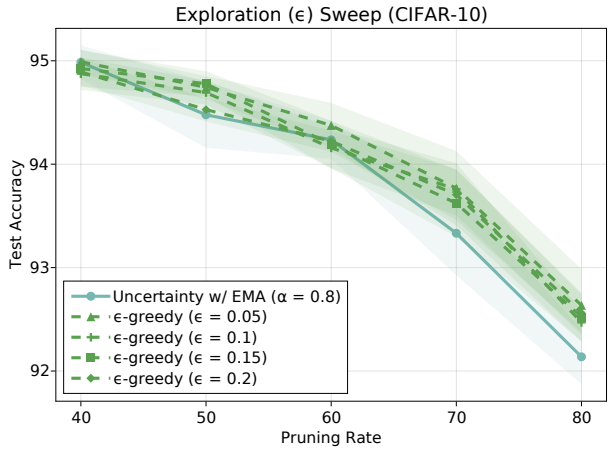
4.4.9 Training with fewer epochs

[!htb]

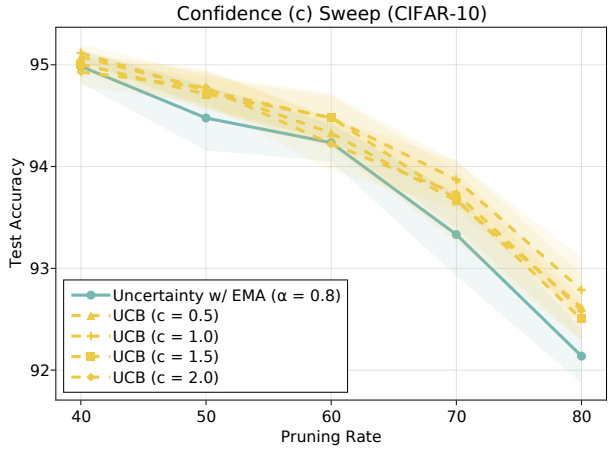
We have shown in Section 4.4.1 that dynamic data pruning is effective on CIFAR10. One may potentially wonder that the random dynamic method's performance is high simply due to the fact that the baseline was trained for 200 epochs. That is, CIFAR10 is a comparatively simple dataset for a model like ResNet18 and if the number of epochs was increased/inflated,



(a) Sweeping the frequency for all methods.

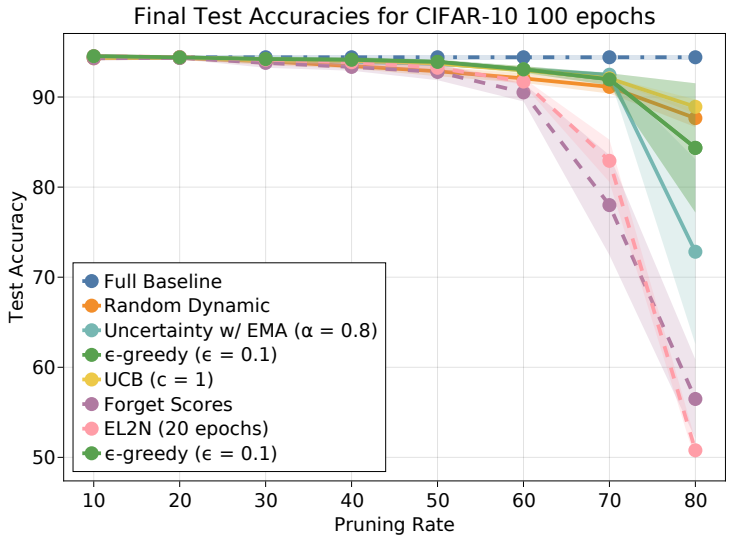


(b) Sweeping the exploration rate (ϵ) for the ϵ-greedy method.

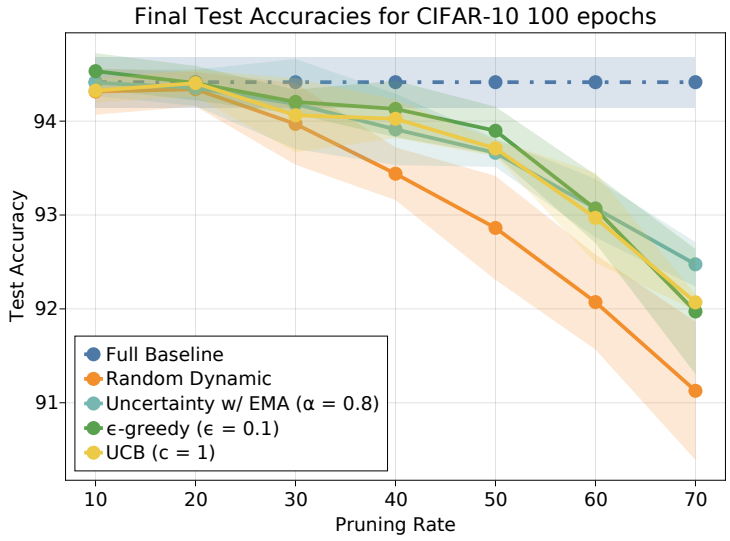


(c) Sweeping the confidence (c) for the UCB method.

Figure 4.16: The effect of sweeping various hyper-parameters while training on CIFAR-10.



(a) CIFAR10 accuracy 100 epochs



(b) CIFAR10 accuracy (zoomed) 100 epochs

Figure 4.17: Dynamic data pruning CIFAR-10 with ResNet-18 with 100 epochs. fig. 4.17a shows the final test accuracies for each method, fig. 4.17b shows the same data without the static methods.

the random dynamic pruning theoretically approach the performance of the baseline trained on the full dataset. To verify if 200 epochs has an impact on the performance of the random dynamic method, we repeat the experiments in Section 4.4.1 except for 100 epochs. We also scale the learning rate scheduler to match the change in epochs.

Figure 4.17 shows the results of this experiment. By comparing the accuracy of the full baseline with 100 to 200 epochs, we observe a difference of 0.72% across 4 different random seeds, which shows that the additional epochs given to the model are nontrivial. When we analyze the static methods, we see that they behave similarly as in the 200 epoch case as they drop off rapidly in performance as we see in Figure 4.17a. The random dynamic method at a pruning rate of 70% maintains a margin of roughly 2.5% from the full baseline compared to a margin of 2% for CIFAR10 training with 200 epochs. The other dynamic methods including the uncertainty with EMA, ϵ-greedy and UCB are able to maintain the same trend as with 200 epochs until 80% pruning before uncertainty with EMA and ϵ-greedy drop in performance. UCB, however, still remains higher than the random dynamic method. From the result of the experiment, we can say that training for more epochs is certainly beneficial for dynamic data pruning in terms of accuracy but our overall conclusions do not change that dynamic data pruning enables more aggressive pruning of the dataset than static pruning.

4.5 Conclusion

This work presents a dynamic approach to data pruning. We reframe the problem as a decision making process, and we present three methods (uncertainty with EMA, ϵ-greedy, UCB) for pruning datasets online while training. This allows our methods to be applied to novel datasets that are being trained on for the first time. Unlike the prior work, our approaches do not incur significant overhead, and we are able to substantially reduce the total run time.

Moreover, we introduce the notion of a sample selection distribution which separates a

dataset into three groups—always samples, sometimes samples, and never samples. We show that sometimes samples are difficult to rank, and as a result, static pruning methods cannot effectively select a subset of them. Surprisingly, we find that some datasets, like CIFAR-100, have little opportunity for sophisticated pruning. This is because the dataset has very few always/never samples. In these cases, the best option is to prune the dataset using a random dynamic method, since all samples have equal importance. We corroborate these conclusions by testing these hypotheses on synthetically modified CIFAR-10 datasets with severe class imbalance or low samples-per-class.

We hope our work emphasizes the need to understand data pruning as a function of the dataset, model, and training trajectory. By viewing the pruning problem as an online decision-making process, we expect future work to borrow from active learning and reinforcement learning to more effectively target sometimes samples. In lieu of these improvements, our methods bring practical, efficient data pruning to DL researchers.

4.5.1 Societal impacts

This work has financial and environmental societal implications. Current trends in deep learning have made training state-of-the-art models prohibitively expensive. As a result, only large research organizations have access to large, overparameterized models. By reducing the total training time, our methods allow independent researchers to train more sophisticated models. Moreover, unlike the prior work, our methods can be applied directly to novel datasets.

More importantly, large-scale deep learning requires weeks of compute time on an energy-hungry GPU cluster. Many of these GPUs are used to parallelize iterating over the dataset. By pruning the data, we allow models to be trained with fewer GPUs or in less time. Both outcomes translate to lower energy consumption and carbon footprints for DL training.

On the other hand, by helping to democratize complex DL models to new domains, our work may negatively impact areas of society affected by those domains. Namely, since

bias and robustness of DL is poorly understood, an increase in its applicability may cause unexpected harm.

4.5.2 Limitations and future work

Our extensive experiments show that dynamic data pruning is an effective approach for accelerating deep learning training; however, there are limitations to our techniques. As the proposed methods rely upon the per-sample loss, label noise would hurt the accuracy of the classifier significantly more than the full baseline as each individual sample has more influence on the decision boundary.

Additionally, our RL-based methods fail to outperform a random dynamic baseline for datasets with many sometimes samples, like CIFAR-100. In these contexts, a more thorough understanding of sometimes samples could lead to a more sophisticated approach that improves over random. In particular, we separate the dataset into the three regions qualitatively. A more principled, formal approach to labeling each samples as "always," "sometimes," or "never" could help tailor pruning methods to the specific dataset.

Moreover, our methods only make a decision about which samples to keep at each checkpoint, but future methods could include various hyperparameters as part of the decision making process. For example, an algorithm could decide the samples to keep at the current checkpoint, as well as the period until the next checkpoint or the pruning rate at the current checkpoint. Such methods could iteratively prune the dataset leading to a better final test accuracy.

Chapter 5

PatchDrop for Efficient Training

This chapter discusses PatchDrop for Efficient Training. Specifically, section 5.1 gives a brief introduction into methods discussing accelerating training such as mixed precision training and data pruning as well as patch pruning strategies for inference and combines them into a new method, PatchDrop for Efficient Training. The following section 5.2 discusses related work to redesigning the attention layer in transformer architecture, pruning patches for efficient inference and accelerating training. Section 5.3 outlines the approach to generate the heatmaps of the training data as well as the method used to drop patches during training to realize actual speed-ups. The subsequent section 5.4 talks about experimental details and the implications that PatchDrop has on both accuracy and runtime. The final section 5.5 summarizes the findings from PatchDrop and discusses directions for future work.

5.1 Introduction

Deep learning has encapsulated many tasks like computer vision, natural language processing, reinforcement learning, and many others through sheer scaling of both dataset size as well as model complexity [3, 4]. The adoption of these large scale models requires significant amount of hardware accelerators like GPUs and TPUs to train them, in addition to various parallelization strategies like model-level parallelism and tensor-level parallelism [6]. The ex-

plosion of vision transformer [8, 34] architectures, which has seemingly replaced convolution networks for image classification, reinforces the necessity for accruing massive amounts of computational resources. This demand continues to drive state of the art performance on benchmarks but restricts the number of researcher who are able to use such models. The timescale of training these models can be on the order of days to weeks to months depending on the scale. If democratizing ML is an effort that is of interest to the community, efficient training strategies must be investigated.

There have been prior works in accelerating training such as mixed precision training, which targets the precision of the model and quantizes it in a manner where no accuracy loss is sustained but actual speed-ups in training time are realized [27]. Progressive resizing of the input images during training is another strategy that is employed to accelerate network training [35]. In a similar vein, data pruning is a method of finding a subset of the full dataset which allows the model to achieve the same validation accuracy as if the model was trained with the full dataset. These can be broken down further into static and dynamic methods: static methods require a pretraining phase to explore which samples would be an approximation for the full data and then using that subset to train a model from scratch [30, 32] whereas dynamic approaches rely on choosing samples during training, potentially periodically swapping which samples should be selected based on the current model's training trajectory [36]. The idea of data pruning can thought of as a coarse-grained approach to accelerate network training.

Briefly leaving training methods aside, there has been a wealth of literature on accelerating inference at the edge. The paradigm of conditional execution is based on the notion that computation of filters should only be done on input features that would generate large responses [19, 20, 22, 37]. These ideas have been extended to pruning input patches to vision transformers to accelerate inference [38–42]. At the heart of these methods to accelerate inference at the edge is the assumption that there are certain patches in the image that are important compared to others and skipping computation on the irrelevant patches will not

have a large detrimental effect on accuracy.

In this work, we want to take the idea of data pruning and pruning patches to accelerate inference to create a novel training strategy of accelerating network training by pruning patches which have the least salient features. Because of the vision transformer's unique architecture of separating the image into tokens, we are able to realize actual speedups during training runs which would not be possible on convolutional and MLP architectures without sparse matrix-matrix multiplication CUDA kernels. We adopt the methodology of [1] who introduced the idea of PatchDrop to stress-test the vision transformer's robustness to occlusion by using the self-supervised ViT model, DINO, [2] to identify the most salient patches in the image. We use their taxonomy, NonSalient PatchDrop, of accelerating network training by identifying heatmaps of the most salient patches and training on them as opposed to the all of the patches in the image. If data pruning should be looked at accelerating network training in a coarse-grained manner, PatchDrop for efficient training is its fine-grained counterpart. We make the following contributions:

1. We describe a novel training strategy to accelerate ViT [8] training through NonSalient PatchDrop [1] with the DINO model [2] to compute saliency maps of the input image. We test our method on a varient of the ViT-small architecture on the Visual Wake Words (VWW) dataset [43] and find that we can prune upto 50% of input patches to accelerate training by 1.66× with a penalty of 1% accuracy.

2. We compare our method with a Random PatchDrop baseline and found that, surprisingly, it achieves similar performance to NonSalient PatchDrop. To understand this phenomenon, we trained the same ViT with Salient PatchDrop(training on "irrelevant features") to occlude the most important features and found that Random and Salient PatchDrop incur more false positive errors than NonSalient PatchDrop. When we analyze the models' performance, we see that NonSalient PatchDrop is indeed the superior training method.

3. We noticed that the original VWW dataset was at a lower resolution (96×96) than our target resolution (224×224) which could potentially smoothen out features. We regenerated the VWW dataset from the COCO dataset using the original pipeline to get a higher image quality. When we trained our models with this new version of the dataset, we not only saw better performance in terms of accuracy but when we performed PatchDrop training, the performance degradation in terms of accuracy was lower than using the poor resolution dataset.

5.2 Related Work

The following section discusses related work associated with patch pruning. The first subsection talks about reducing the complexity of the self-attention module to below $O(n^2)$ complexity. The following subsection introduces patch pruning strategies to accelerate inference. The last subsection talks about different methods to accelerate neural network training.

5.2.1 Reducing Attention Complexity

Transformers have established themselves as SOTA in natural language processing and now vision tasks [7] but unfortunately are still slow to train and use in practice [65]. While the self-attention module captures long-term dependencies by examining the input tokens in a global context, it incurs a $O(n^2)$ time and space complexity cost by generating the attention matrix, where n is the sequence length. Several proposals attempt to address the cost of generating this matrix by manipulating the architecture of the self-attention module. Sparse Transformer [115] reduces the quadratic complexity to $O(n\sqrt{n})$ by restructuring the residual blocks and storing a set of sparse attention kernels to compute subsets of the attention matrix. ReFormer only incurs $O(n \log n)$ by swapping out the dot-product attention module for one that uses locality-sensitive hashing and reversible residual layers [66]. Finally, Linformer

further reduces the complexity to $O(n)$ by showing the attention matrix can be approximated with one that is low-rank [116]. The key differentiating factor between these works and the one presented in this paper is that rather than altering the attention mechanism, we focus on eliminating the irrelevant patch embeddings by dropping them at the input. As such, our methods are complementary to approaches described above.

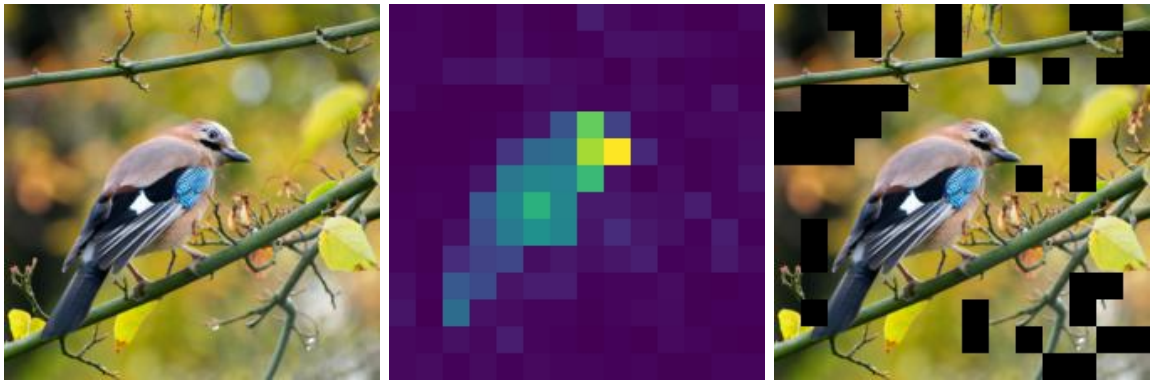
5.2.2 Patch Pruning for Efficient Inference

Pruning patches has been explored since the inception of ViTs [8] in order to reduce computational complexity for efficient inference [38–40, 42, 117–119]. Tang *et al.*[117] apply a recursive algorithm to prune patches from the last layer to the first layer and each patch is assigned an importance score such that lower score patches will be pruned. Rao *et al.*[38] follows this line of patch pruning by augmenting a pretrained ViT with an auxiliary prediction model at different MHSA layers to predict which patch should be dropped; in order to realize actual speedups during inference, they propose an attention masking strategy to differentiably prune a token by blocking its interactions with other tokens. Xu *et al.*[39] augments the ViT by a custom module which exploits global class attention to categorize tokens into informative ones and placeholder ones. The informative ones are passed into the attention and feedforward module whereas the others are fed into a cheaper path of feedforward networks. Lin *et al.*[41] enables the ViT to achieve adaptive accuracy-speed trade-offs based on hardware requirements based on training with multiple patch sizes and token keeping rates. Wang *et al.*[118] notice that not all images require all the tokens to be present and that some are easier than others to classify. They introduce an early exit scheme by cascading multiple transformers with an increasing number of tokens and terminate the computation when a confident prediction is achieved. These schemes all are focused on making modifications to the transformer architecture and are concerned with accelerating transformers on the edge through token pruning. Our work is similar in regards to token pruning but we want to apply token pruning to the training phase in order to train ViT models faster and unlike many

of the listed works here, we only make modifications at the input layer after the positional embeddings are injected to obtain the maximum possible speedups.

5.2.3 Accelerating Training

Accelerating training is a much less focused part of efficient deep learning when compared to efficient inference. Most of the innovation comes for free from hardware accelerators like GPUs and TPUs [120]. In addition, schemes like data parallelism, model parallelism and tensor parallelism helped scale large language models (LLMs) and speed up computations [6, 121]. Mixed precision training quantizes the weights of the network to 16-bit precision and keep a copy of full-precision weights [27]. This leads to a significant savings in both computation as well as memory consumed. Tan *et al.*trains an EfficientNetV2 model through progressively increasing the image size during training, but it often causes a drop in accuracy [35]. They incorporate changing the data augmentation recipe based on the image size. Another avenue to accelerate training is designing network architectures which run more efficiently than traditional backbones [90, 122, 123]. A final orthogonal method is finding a subset of data which is able to stand in as a substitute for the full dataset; the process of selecting this subset to save on compute is know as data pruning [30, 32, 36]. In data pruning, many works have focused on summarizing the dataset in a static step during a pretraining phase portion and then using that subset to train a new model to match the prior model's performance. Toneva *et al.*[30] trains a model and counts the number of times an example is misclassified during training; once completed, the examples will be ranked based on the number of times a model "forgot" (misclassified) an example, called forget scores, and a subset will be chosen based on a user-defined threshold. Paul *et al.*[32] builds on this work by training an ensemble of models for a much smaller number of epochs to capture the forget score ranking earlier in training and even sometimes at initialization. They find that their GraNd/EL2N scores can match the performance of forgets scores and show that they can prune upto 50% of the dataset on CIFAR10 with no loss in performance. Noticing that



(a) Original (b) Attention Map (c) Occluded

Figure 5.1: A bird image occluded with the Non-Salient PatchDrop method with λ=0.2. The attention maps are obtained from the final MHSA block from DINO[2] and subsequently averaged across all heads. The dropped patches in the image are background pixels which do not contribute to the object of interest.

these approaches to data pruning require an expensive pretraining phase, Raju *et al.*[36] take ideas from active learning and reinforcement learning [49, 56] to dynamically prune the data during training and let the model choose which data points it should train on according to a set number of epochs before reselecting the samples. They find that they are able to prune more aggressively than other static baselines while incurring less accuracy degradation.

5.3 PatchDrop Training

In this section, we want to explore the possibilities of trying to accelerate ViT training by pruning patches. We begin by outlining the computational complexity of the multi-headed self-attention (MHSA) module and how much computation we can cut down by dropping patches. We explore dropping patches in a static fashion prior to training. In particular, a subset of image patches will contribute more to the model's generalization than other subsets and we want spend the computation budget on subsets that contain the object of interest. We use the insights and methods of a recent work on the ViT's robustness to occlusions [1] to identify non-salient patch candidates to prune. Finally, we examine the accuracy-wall-clock time tradeoff to determine what is the optimal number of patches to use during training.

5.3.1 Computational Complexity

ViTs are composed of myriads of components so we want to be able to get an estimate of much computational savings we can reap by dropping patches during training. Assuming an input $x \in \mathbb{R}^{N \times D}$, we need to project x into the embedding space to obtain our queries, keys and values, where N is the number of tokens and D is the hidden dimension. Assuming a hidden dimension that matches the input, this will take 3 linear operations, incurring $O(3ND^2)$ operations. To generate the attention matrix, we transpose the key embeddings and multiply them with the query embeddings, which is another $O(N^2D)$ operations. This then needs to scale the value embeddings, adding $O(N^2D)$ cost. Since we perform the SA operation over H heads for MHSA, we have to multiply the total complexity by H and then we concatenate the outputs of all heads together to transform the outputs from all the heads back to the original embedding space, which incurs $O(NHD^2)$. Adding all the costs together and factoring out H, we can see that the final cost is $O(2N^2D + 4ND^2)$. The additional 2 set of MLPs succeeding the MHSA block, assuming hidden dimension d', will add $O(Ndd')$ operations. The space complexity of the MHSA will be dominated by the generation of the attention matrix, which will also impose $O(n^2)$ memory cost.

The complexity with respect to the number of tokens in the sequence length and the hidden dimension is quadratic. When we introduce the patch drop term λ, our goal is to target the number of tokens in the input sequence to alleviate both the quadratic cost in terms of memory and space; if we assume a constant D and drop patches according to λ, we can realize a theoretical speedup and memory reduction of $\frac{1}{\lambda^2}$ for the MHSA block and speedup of $\frac{1}{\lambda}$ for the subsequent linear layers. This quadratic scaling can provide massive speedups if λ can be increased and therefore, for accelerating ViTs, dropping patches seems promising.

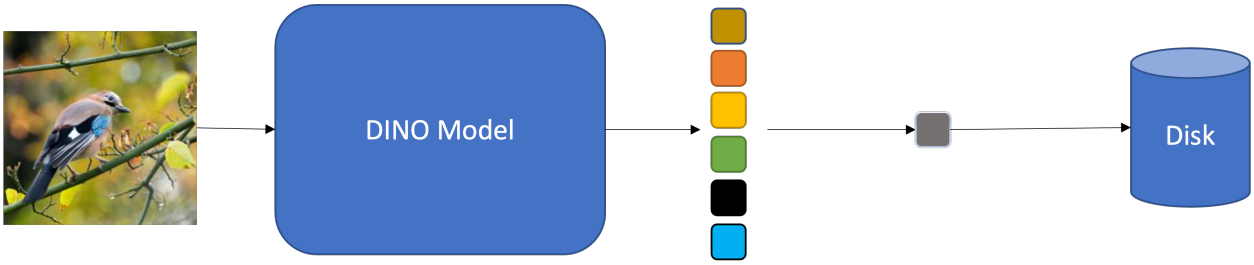


Figure 5.2: As a pretraining step, we generate the averaged heatmaps using the DINO model from the self-attention heads in the last layer and save them to the disk.

5.3.2 Obtaining Patch Segmentations

ViTs' success rests on the multi-headed self attention (MHSA) layer to model interactions between different image patches [7, 8]. More recently, Naseer *et al.*[1] studied ViTs' robustness to various kinds of occlusions and perturbations on the ImageNet validation set [46] and found that they are highly resistant to them due to ViT's dynamic receptive field. Specifically for occlusion, they develop a masking strategy, PatchDrop, which zeros out patches based on patch selection criteria and a patch drop rate hyperparameter, λ, which determines the percentage of patches that are occluded.

NonSalient PatchDrop is one such policy that zeros out patches if their saliency is below a threshold set by λ. Concretely, Naseer *et al.*[1] use the attention maps from the penultimate layer of a self-supervised ViT model DINO [2] to segment the objects in the image and create a binary mask to occlude the image. Figure 5.1 shows the effectiveness of the DINO model on a random image from the ImageNet validation set at a patch drop rate of $\lambda = 0.2$. By averaging the attention maps of the last MHSA block, the foreground object has the largest magnitude and DINO can generate fine-grained segmentation masks of our object of interest.

5.3.3 Static PatchDrop Training

NonSalient PatchDrop was designed to test ViT's robustness to occlusions but we are interested in leveraging this method for accelerating the training time of ViTs. We perform one pass of the training set with DINO to obtain the averaged attention maps and save them

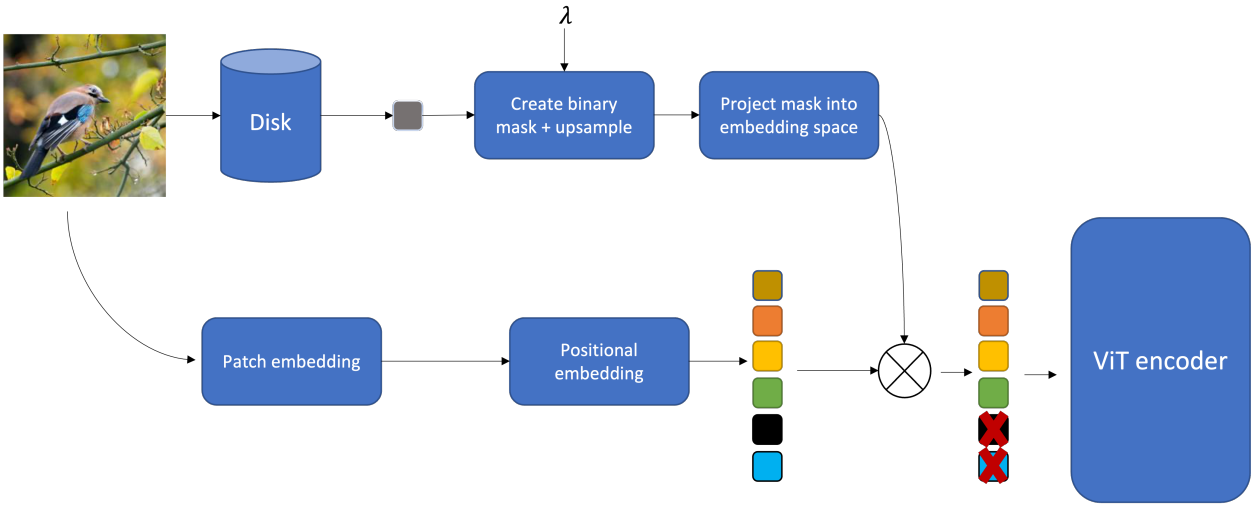


Figure 5.3: We describe the pruning pipeline to accelerate ViTs in a static fashion. The averaged heatmaps are loaded from the disk and a binary mask is created based on the input patch pruning rate, λ. This mask is upsampled and projected into the same embedding space as the input after the patch and positional embeddings and been applied. The "zeroed-out" patches in the tensor are dropped and passed through the ViT.

to disk as detailed in Figure 5.2. We follow the same methodology as in detailed by Naseer *et al.*[1] by thresholding the patches and setting them to zero with a binary mask. Since we are interested in realizing wall-clock time reduction, we cannot just set the dropped patches to zero, but rather, we need to filter them out to save on computation. We perform this thresholding once the positional encoding has been injected into the patch embeddings by transforming the binary mask to the same shape as patch embeddings as seen in Figure 5.3. In this manner, we load the heatmaps from disk and drop them to prevent computation. As a control to the NonSalient method, we design a Random PatchDrop method which randomly drops patches in the image to evaluate if the DINO model is providing optimal segmentation of the foreground objects.

5.4 Experiments

In this section, we discuss the datasets we apply our method as well as the model/training setup. We further elaborate on the results on sweeping the patchdrop parameter λ and its

impact on validation accuracy and runtime. We find surprisingly that NonSalient PatchDrop achieves similar performance to Random PatchDrop, which motivated us to perform Salient PatchDrop, which is analogous to NonSalient except that we occlude the most salient patches in the image.

5.4.1 Dataset

We use the Visual Wake Words dataset as a case study for accelerating ViTs by dropping patches [43]. The Visual Wake Words (VWW) dataset is a benchmark that is used for tiny vision models and distinguishes between a whether a person is present within a frame or absent, making it a binary classification task. The dataset was generated from the COCO dataset taking 115k images and split it into a training set and a validation set, consisting of 78,889 images and 36101 images respectively. According to the authors who released the dataset, the classes are relatively balanced 47% of the images in the training set are labeled as in the 'person' class and similarly, 47% of the images in the validation set are also in the 'person' category. The image resolution of the Visual Wake Word dataset is 96x96, which is higher than the typical small dataset resolution of CIFAR10/CIFAR100 of 32x32. While the dataset is simple in the sense of being a binary classification problem, the higher resolution of the images makes it more amenable to use as a case study for vision transformers rather than a standard dataset like CIFAR10/CIFAR100 and simpler to reason about than a larger-scale task like ImageNet.

5.4.2 Training setup

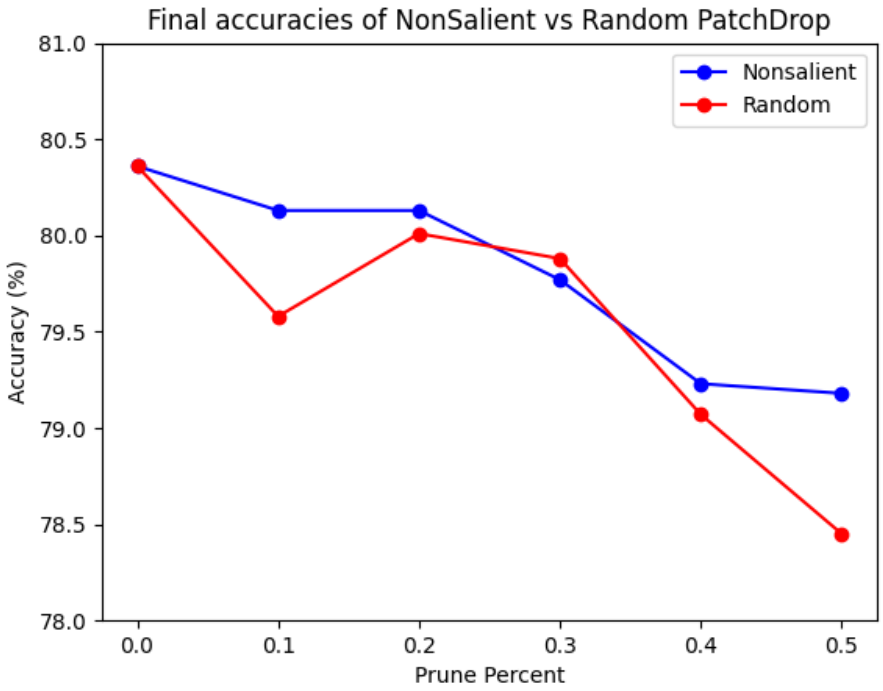
We train the small ViT [8] as the target model with some modifications. We use 8 MHSA layers with each layer having 12 heads, set the embedding dimension to 384, and a patch size of 16. We do not use any kind of dropout or stochastic depth for dropping paths in the network. For the DINO model, we use the ViT-Base model trained on ImageNet to generate the heatmaps for the VWW images; rather than another model that is trained on VWW,

using DINO is sufficient as the person category is included in the ImageNet training set and provides good segmentations. To process the VWW images into heatmaps, we upscale the image resolution from 96x96 to 224x224. Similarly, for the input pipeline, we rescale the image to 224x224 resolution, RandomCrop with padding=4, and the standard normalization scheme. In addition, we also use the cutmix augmentation with $\beta = 1$ and the probability of using cutmix on the batch set to 0.5 for mitigating overfitting [124]. We use the Adam [125] optimizer with the following setup: a learning rate of 10^{-3} with $\beta_1 = 0.9, \beta_2 = 0.99$, weight decay=$5 \cdot 10^{-4}$, Nesterov momentum=0.9 and cosine decay scheduler with 5 warmup epochs over 100 epochs with a batch size=128. We use the cross entropy loss with label smoothing $\epsilon = 0.1$. In order to accelerate our experiments, we use mixed precision training on an NVIDIA GTX2070 GPU [27]. All our experiments are run in the Pytorch framework [111].

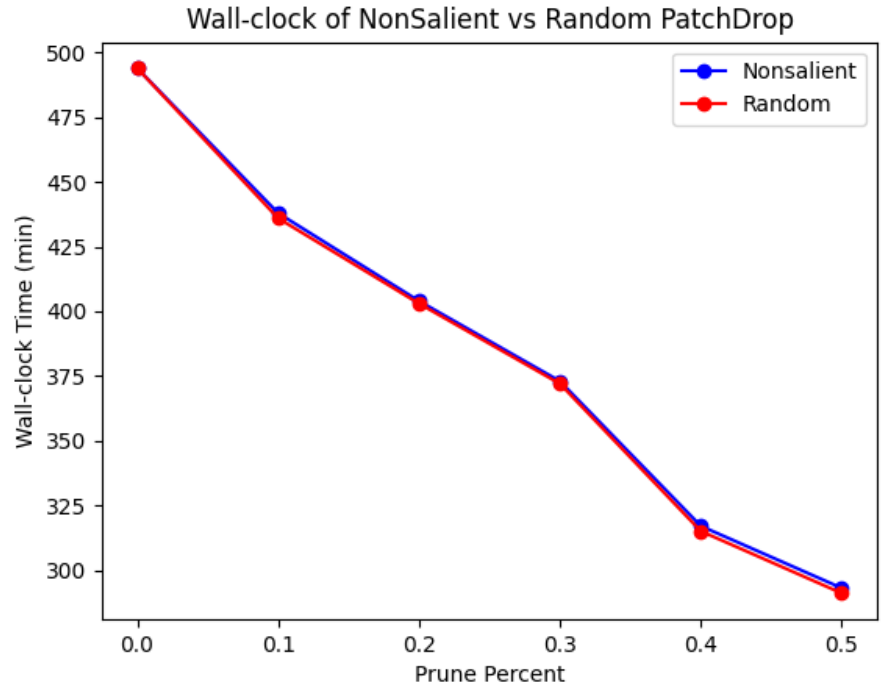
5.4.3 Results

VWW accuracy and runtime

We first compare the accuracy and runtime of NonSalient vs Random Patchdrop in Figure 5.4. We sweep the λ patchdrop parameter from 0.1 to 0.5. The difference between the Random PatchDrop and NonSalient PatchDrop, as shown in Figure 5.4a, is surprisingly not very large, although there is some deviation at λ=0.5. The runtime is aligned for both methods and achieves about 1.66× speedup. If we reference the computational complexity analysis in section 5.3.1, the actual speedup does not compare to the theoretical speedup. One of the reasons for this potential delta is that an overhead is incurred on indexing the original tensor to obtain the patchdropped tensor that is larger than the time of propagating the input through the block of self-attention and feedforward layers. This phenomenon is well-documented [126] and this overhead is exposed due to the optimized nature of kernels for dense matrix multiplications.



(a) VWW accuracy



(b) VWW runtime

Figure 5.4: PatchDrop applied to VWW with on a variant of ViT. 5.4a shows the final test accuracies for each method, 5.4b shows the runtime.

NonSalient vs. Random vs. Salient PatchDrop

This result seems to imply that there is no difference between information content in each patch, which seems to defy intuition. In order to understand this lack of difference, we introduce another baseline, Salient PatchDrop, which in contrast to NonSalient PatchDrop, occludes the most salient patches in the image. The expectation is that the performance should be quite subpar since the most important features in the person class are dropped. We perform this evaluation without the cutmix augmentation [124] so that we can isolate the effect of incorporating PatchDrop as an augmentation method. Figure 5.5 shows the difference between NonSalient and Salient PatchDrop validation accuracy with patchdrop parameter $\lambda = 0.5$ as well as the baseline's accuracy to show the performance drop. The validation accuracy of Salient PatchDrop, NonSalient PatchDrop and the full baseline is 71.95%, 77.64%, and 79.95% respectively. The validation accuracy shows the difference between Salient and NonSalient PatchDrop at roughly 5-6% percentage, demonstrating that there is a definite ordering to which patches are more relevant than others.

While Salient PatchDrop certainly has worse performance, its validation accuracy is still quite decent only being 5% lower. It would be expected that the model would perform poorly since most of the person features have been occluded. We plot the confusion matrix for Salient, NonSalient and Random PatchDrop, in addition to the baseline, to understand what kind of errors each model is making on the validation set.

Figure 5.6 shows this analysis; the predicted label is the output from the model with "0" being no person detected and "1" being a person detected in the frame. The confusion matrix for the baseline indicates that a larger percentage of errors come from false negatives (FNs) compared to all the other methods. Interestingly, the Salient PatchDrop method classifies correctly most of the person images but a large percentage of the non-person images were misclassified. This result indicates that Salient PatchDrop does not classify a person image based on the features of a person since only the background is present; this can be seen in the false positive rate (FPR) and the true negative rate (TNR) of Salient PatchDrop which

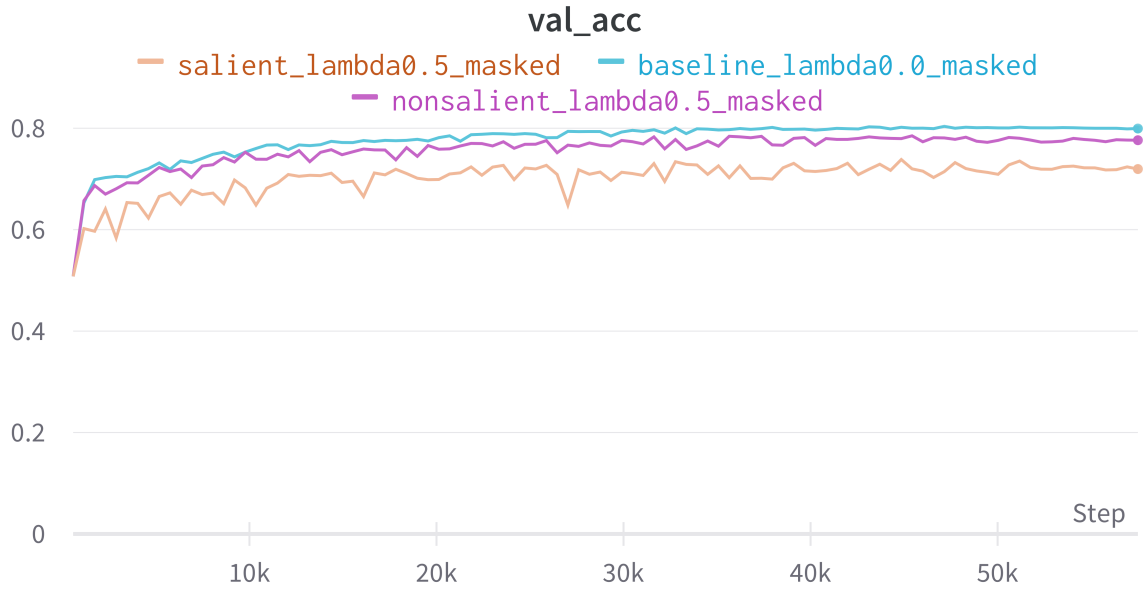
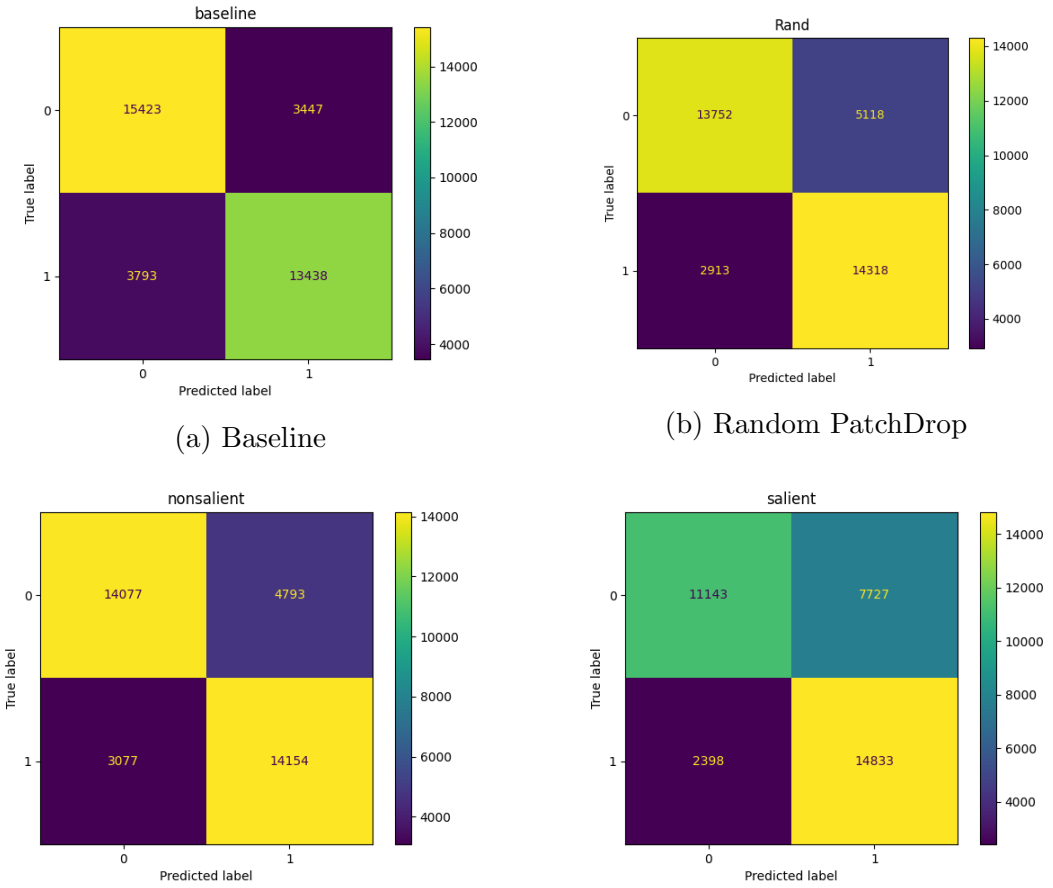


Figure 5.5: Comparing the difference in validation accuracy between NonSalient (purple) and Salient (tan) PatchDrop at a patchdrop parameter set to $\lambda = 0.5$ with a baseline with no PatchDrop (blue). We trained these models with no cutmix [124] to isolate the impacts of PatchDrop.

is significantly higher than the FPR of the baseline and lower TNR of the baseline. Based on this explanation, all PatchDrop methods have a bias towards classifying images as person images; the caveat being that NonSalient PatchDrop is less biased than Random and Salient PatchDrop in this case. We can see see that NonSalient PatchDrop is superior through comparing all the methods' F_1 scores.

Typically, we evaluate the validation set on accuracy but there are other measures like precision, recall, and F_1-score. Precision is the correct positive predictions relative to the total positive predictions (precision $= \frac{TPR}{TPR+FPR}$) whereas recall is the correct positive predictions relative to the total actual positives (recall $= \frac{TPR}{TPR+FNR}$). The F_1-score is defined as the harmonic mean of the precision and recall. This score is slightly better than accuracy when it comes to dealing with class imbalances and such it may provide better assessment of model performance than a raw measure like accuracy. The highest possible value of the F_1 score is 1.0 whereas the lowest is 0.0, similar to accuracy. The F_1 scores for the baseline,



(a) Baseline

(b) Random PatchDrop

(c) NonSalient PatchDrop

(d) Salient PatchDrop

Figure 5.6: Confusion matrix for the a) baseline, b) Random PatchDrop, c) NonSalient PatchDrop and d) Salient PatchDrop at $\lambda = 0.5$ trained with no cutmix [124]. The confusion matrix from the Salient and Random PatchDrop indicate that a larger percentage of errors are from false positives the model misidentifying that a person is in the image when the true label has no person present.

NonSalient, Random, and Salient PatchDrop are 0.787, 0.79, 0.78, and 0.744, respectively. The F_1 scores for the Salient and Random PatchDrop method are lower due to the disproportionate amount of false positives compared to NonSalient PatchDrop and the baseline. While NonSalient and Random PatchDrop achieve similar accuracies, NonSalient PatchDrop has a higher F_1 score and has better discrimination as evidenced by the higher TNR and lower FPR. Another contributing factor could be the scope of the problem being a binary classification task: even if the classifier errs on classifying all samples from one particular class like Salient PatchDrop, the performance is still relatively high. If there were more

(a) 96×96 upsampled to 224×224 (b) 640×480 downsampled to 224×224

Figure 5.7: Comparing the image quality of downsampling from initial resolution (96×96) before upsampling to target resolution(224×224) with downsampling 640×480 to target resolution(224×224). Upsampling the resolution in Figure 5.7a shows that the image has been significantly smoothened and many features are blurred compared to Figure 5.7b where many of sharp features are preserved.

classes present, the possibility of making an incorrect classification would be larger and we would likely see more of a change comparing NonSalient, Random, and Salient PatchDrop.

5.4.4 Changing image quality of VWW dataset

The VWW dataset we used for training our models was sourced from the MLCommon's tinyML github repository. The data was preprocessed to an image size of 96×96 from the original COCO dataset which has an image size of 640×480. We need to upscale the resolution of the image to 224×224 to have the DINO model to process the image. This procedure of upscaling from a lower resolution could potentially introduce noise and unwanted artifacts into the image which would affect performance of the heatmaps and subsequently of the classifier. In order to address this issue, we followed the procedure outline in [43] to regenerate the dataset and downsample directly to an image resolution of 224×224. Figure 5.7 shows the difference in image quality from different procedures and directly downsampling from the higher resolution 650×480 COCO image saves many of the

Table 5.1: Comparing the accuracy of ViT with different image qualities.

Method	**Accuracy**	**Δ**
Baseline ($96 \times 96 \rightarrow 224 \times 224$)	80.36%	-
Baseline ($650 \times 480 \rightarrow 224 \times 224$)	81.91%	-
PatchDrop ($\lambda = 0.5$) ($96 \times 96 \rightarrow 224 \times 224$)	79.18%	-1.18%
PatchDrop ($\lambda = 0.5$) ($650 \times 480 \rightarrow 224 \times 224$)	81.02%	-0.89%

finer features in the dataset which is going to improve the model we want to train.

To follow up, we are interested in the impact of the image quality on accuracy when we perform patch pruning. Table 5.1 indicates that training with a higher quality image gives 1.55% higher percentage points in validation accuracy. When we apply PatchDrop with $\lambda = 0.5$, the drop in performance is slightly lower with higher image quality at -0.89% compared with -1.18% with the lower image quality. This result suggests that training PatchDrop on higher complex datasets will be promising.

5.5 Conclusion

This work presents a new strategy of training ViT architectures by pruning patches with NonSalient PatchDrop. We demonstrated on the VWW dataset that we are able to train small ViTs to comparable accuracies to the full baseline while realizing actual speedups. ViT process inputs in a different manner than MLPs and convolutional networks that make these speedups possible.

We compared our approach to a Random PatchDrop baseline as well as Salient PatchDrop baseline and found that NonSalient PatchDrop does a better job in terms of minimizing false positives and contributes to a higher F_1 score, which we show through the confusion matrix on the validation set. Furthermore, we show that if we clean the data by downsampling to the target resolution we would like to train on, we get higher performance which in turn leads to lower accuracy drop when pruning patches. This suggests that scaling our method

to datasets which contain higher resolution images is a promising direction to explore.

For additional future work, we would like to pursue:

1. We have seen that our PatchDrop approach works with mixed precision training [27]. We want to combine our approach with methods in data pruning [30, 32, 36] to further accelerate network training and explore this design space.

2. When we perform PatchDrop training, the memory consumed by training process is much lower so we can select a model which previously would have run out of memory. This leads to an open question of whether it is better to prune patches in order to fit a larger model on device or stick with a small model with all the patches.

3. The current iteration of PatchDrop depends on the practitioner to select the appropriate PatchDrop parameter λ to train the model as well as passing the dataset through the DINO model [2], which adds more complexity to the training recipe. For future work, we want to design a controller that can dynamically set what the pruning rate should be and determine the saliency maps during training.

Chapter 6

Conclusion

In this chapter, we review our findings on our methodologies on efficient inference and training efficiency in Section 6.1 and outline future directions in Section 6.2. Finally, I reflect on some of work that did not fit into our paradigm of efficient inference/training as well as some other topics in deep learning.

6.1 Summary and Conclusions

We have outlined our approach of pruning channels, samples, and patches which ties together the theme of data dependent pruning of deep learning computation. We first have shown the synergy between Feature Boosting and Suppression (FBS) [19] and Conditionally Parameterized Convolutions (CondConv) [23] to combine these methods into **FBS-pruned CondConv** [37] to achieve higher pruning rates with reduced accuracy degradation by using dynamic filter generation; in particular, our goal is to show that we can use non-traditional ways of pruning to accelerate edge computation which is not necessarily the goal of a method like weight-magnitude pruning.

On the theme of pruning, we next described our framework of **dynamic data pruning** [36] and its advantage(s) over static pruning approaches. We borrow concepts from active learning like uncertainty sampling for sample selection. By allowing samples to be selected

on the fly, we were able to apply basic reinforcement learning strategies like ϵ-greedy and upper-bound confidence (UCB) algorithms to improve the sample selection process during training. Setting up this framework, we noticed this category of samples which would be important during some points during training but then would not be relevant later, which was in contrast to prior models that separated the samples into important ones and irrelevant ones. The existence of sometimes samples are the key insight that the dynamic data pruning work exposes as it explains why some datasets are more amenable to pruning than others and that by cycling the samples we can stave off the accuracy deficit that the static methods suffer with wall-clock savings being realized.

As we alluded in the Introduction chapter, if data pruning should be looked at a coarse grained method to accelerate training, (NonSalient) **PatchDrop** should be viewed as its fine grained cousin. The motivation for the work is straightforward: there is clearly a hierarchy among what patches contribute most to a model's output prediction. This is evident on the fact that so many conditional execution works [22, 23, 25, 37] rely on this assumption as well as patch pruning works to accelerate inference. We exploit this fact when we want to train on a subset of patches on ViT [8] that are relevant to save on training time. We used the self-supervised DINO model [2] to produce saliency heatmaps to segment important parts of the input image. We adopted the PatchDrop method from [1] which originally was designed to test ViT's robustness to occlusions to speed up training time. We tested our idea on the Visual Wake Words (VWW) dataset and found that when we compared our method to a Random PatchDrop baseline, at about 50% patch pruning rate, we observed an accuracy difference of only 0.73%. To verify our assumption of some patches having more importance than others, we trained a model with Salient PatchDrop, which showed that the performance on training on background features is worse. Furthermore, when we looked at each method's F_1 score, we can see that NonSalient PatchDrop has the superior discriminatory ability over the other baselines. Finally, we see that when we recreated the dataset to support higher image quality our PatchDrop method suffers less accuracy degradation, which seems

promising for scaling up to larger-scale tasks.

In short, it is imperative that we move towards methods that improve the computational efficiency of models during inference and especially during training. The current trend in deep learning exploding the number of parameters in a model is negatively impacting the independent researchers who cannot operate at such a large scale. Dynamic data pruning as well as PatchDrop are small steps towards reducing the computational burden on individuals who do not have numerous resources to apply SOTA methods to their specific use-cases.

6.2 Future work and directions

In this section, we outline some interesting directions that each one of these projects can take and that we intend to explore for future work, which were not discussed in full in this document.

6.2.1 Conditional execution applied to Differentiable Neural Architecture Search (DNAS)

Throughout this document, the main selling point of conditional execution methods was to perform selective computation in order to reduce latency. For future work, we are interested in applying some of these methods to problems like Differentiable Neural Architecture Search (DNAS) [127] in order to reduce the memory requirements imposed by the supernet [128–130]. We will describe Neural Architecture Search (NAS) very briefly to understand the context of this proposal.

NAS is the method of automatically designing network architectures as opposed to handcrafting them which was the old way of determining network backbones [131, 132]. This be done through reinforcement learning by sampling different network architectures from a controller and collecting the validation accuracy of the target network as a reward or defining a larger supernet [127–130], containing numerous subnetworks that are jointly trained

in a differentiable manner (DNAS) with the parameters in each block and is finally sampled after a "search" phase. It is yet to be determined which search method is definitely superior (even Random Search performs quite well) [131] in terms of producing networks with high accuracy but gradient descent as a search strategy generally can cut down the number of GPU-days it takes to get a decent model [127, 128]. A limiting factor to the gradient descent strategy though is that the models are limited by the memory consumed [129] and thus a small supernet needs to selected. This has a negative impact because it reduces the space of possible models that are generated, perhaps eliminating optimal architectures.

Proxyless NAS [129] makes note of this observation by seeing that the SOTA NAS methods at the time relied on generating architectures on proxy tasks (CIFAR10 for example) before transferring to a larger scale task like ImageNet since the algorithms are constrained by memory issues. The mismatch on searching on proxy tasks produces suboptimal networks so as an alternative they train a model in a supernet-style manner but use BinaryConnect [133] to zero out certain candidate paths on the forward pass. By only forcing one path to be active at run-time, they are able to train a model on the target task with no transfer.

While conditional execution methods are primarily designed for reducing latency constraints, it can also be used as a substrate to eliminate memory costs. As there are a wealth of different methods in the conditional execution literature, cost saving methods such as FBS [19, 37] can drop intermediate feature maps which are the costly components when it comes to memory issues in deep learning. By applying conditional execution to NAS during the search phase, we can expand the size of the supernets and therefore increase the search space of the models and promote diversity of operations that are in the supernet.

6.2.2 Dynamic Data Pruning

In the main body of the document, we described dynamic data pruning as a sample selection process during training. We discussed this solely on image classification workloads like CIFAR10 and CIFAR100 but we would like to expand our training methods to other domains

like natural language processing (NLP) and audio processing.

Dynamic data pruning in the context of image classification is removing certain images from the training subset; for NLP, pruning would be removing sentences from the target document. This would be an interesting direction to explore as the prior works related to data pruning are restricted to image classification tasks [30, 32, 108]. However, a major roadblock to applying these methods directly to language modeling is that the current paradigm seems to be transferring large language models (LLMs) to finetune on the target task [134]. These language modeling tasks only have a few thousand examples in the training set and our results from our CIFAR100 experiments indicate if there is a low example to class ratio, the opportunity for pruning is substantially reduced. However, for neural machine translation (NMT) tasks, the English-German training dataset contains roughly 4.5M sentences [135] for the model to train on. This task is attractive in the sense that there is a lot of data and potential for considerable amount of redundancy in the dataset.

Within the context of pretraining, our data pruning framework could be applicable to unsupervised training methods used by LLMs like BERT and GPT3 [3, 4, 65]. In this case, the Masked Language-Modeling (MLM) objective used in BERT can easily be adapted to an uncertainty sampling method. Again, when it comes to training large models, it is imperative to target potential redundancy in the dataset to achieve faster speedups. The main challenge with directly applying these ideas to these models such as BERT is that a massive upfront computational cost is necessary even to pretrain these models so it is difficult to truly say that our approach can produce a similar model since most of the time we do not have access to the resources to replicate the pretraining flows. Nevertheless, data pruning will, at the very least, reduce the number of resources required to execute the pretraining flows.

In addition to the application to NLP, data pruning is applicable to iterative procedures found in NAS [131, 132]. Described in the previous section, NAS is an expensive search procedure to find optimal architectures on the tasks we are interested in as practitioners. Typically, it is split into two separate phases: search phase and finetuning phase. The

finetuning phase is quite inexpensive in terms of training time and can be thought of as a small fraction of total training cost; the search phase, whether it be RL or differentiable search strategies, far outweighs the finetuning phase and is an ideal target for our data pruning framework. In particular, depending on the task, we can employ an extremely aggressive pruning strategy since we primarily are interested in the decision(s) the NAS algorithm makes and the accuracy of the model is secondary as we can use the full dataset during the finetuning phase to recover the accuracy. These arguments are a justification to applying data pruning to NAS but similar reasoning can be used to apply data pruning to training flows such as the lottery ticket hypothesis [136–138] and robust adversarial training [139–144].

The lottery ticket hypothesis (LTH) [138], which states that within a dense network there exists subnetworks that can achieve the same performance as the dense counterpart, seems to imply that there is a minimum working structure of weight connections that is sufficient to perform the learning task. In the context of data pruning, a question that arises: "Does there exist a minimum coreset within a dataset similar to subnetworks existing in a dense network?" The answer to this question is most likely yes but it ignores the nuances of what models select the samples; while the LTH admits a sparse structure, it does not claim the subnetwork obtained is unique. For data pruning, a certain set of samples will undoubtedly be selected over all models but each network's decision boundary will be different causing different subsets to be selected. One interesting experiment would be to extend NAS to select the samples subject to a sample budget constraint in addition to creating new architectures. This way the subset will perform quite well across all types of target architectures, although the initial pretraining phase to generate this subset will be highly expensive.

From a more theoretical perspective, we do not have a good grasp on how data pruning is sensitive to other hyperparameters in the training recipe. By assuming that a model is going to be trained with the same flow similar to the full dataset, we may be inducing a suboptimal training strategy; for example, it is conceivable that dynamically pruning sample

leads to large jumps in parameter space and it may be necessary to scale the learning rate back to account for the change in the input distribution so as to not cause instability in the training procedure. This begs the question of how is dynamically pruning data changing the loss landscape since the input distribution is constantly changing in comparison to using the full dataset or static data pruning.

As a brief aside, one may wonder how data pruning performs on other types of compute platforms as we have only demonstrated its effectiveness on a single GPU. However, due to the abstraction provided by the computing stack, data pruning will work regardless of whatever the underlying substrate or device whether it be a CPU, embedded device, TPU, or another accelerator, provided that it is capable of data parallelism.

6.2.3 PatchDrop for Efficient Training

In Chapter 5, we discuss how we can use the unique input representation of the vision transformer [8] in order to speed up training by pruning the least relevant patches in an image. One limitation of our method is that we rely on the self-supervised DINO [2] model in order to generate the saliency heatmaps. This limits the application of our training method: in an online environment where each data point is trained sequentially, it would need to consumed by the DINO model before being passed onto the ViT. Ideally, we would like to eliminate the use of the DINO model and solely rely on the ViT being able to drop input patches by itself through a differentiable controller. In addition, we note that unlike dynamic data pruning, our training is contingent on the target model as a ViT; it would be interesting to develop kernels which exploit sparsity in the input as most SpMM kernels rely on sparsity in the weight matrices [145, 146]. We can then apply our patch pruning approaches to traditional convolutional neural networks and more exotic ones such as the MLP-Mixer [147] or ConvMixer models [148].

We have made the coarse-grained vs fine-grained analogy between data pruning as well as pruning patches but have discussed each one separately throughout the course of this

dissertation. However, a natural extension to both works could be to integrate both methods together and explore this new design space. This leads to questions such as whether it is better to prune patches more than the entire sample itself as it preserves the integrity of the data distribution or if pruning data is better as even the background features in a single image provide more valuable information than only having the most salient regions from each image.

6.3 Reflections

In this section, I discuss some work that I completed during the course of my PhD that did not fit into the neat paradigm of efficient inference or training. This is most in relationship to adversarial machine learning (AML) and sparsity. Specifically, I look at the interaction of adversarial training [144] and the lottery ticket hypothesis [138]; I give a brief background on AML and the lottery ticket hypothesis. Then, I talk about some further opinions I have about AML and some interesting ideas worth revisiting at a future time.

6.3.1 Prerequisites

Adversarial Robustness

Broadly speaking, an common approach in adversarial machine learning is properly defining a threat model. The threat model states the information known to a malicious actor/adversary and allowable actions that the actor/adversary can to sway the model's prediction. Trivially, the attacker can change drastically the input if unconstrained; the threat model defines a maximum allowable distance, denoted by ϵ, and the adversary can change the input up to this noise parameter. Typically, the adversary operates in a white-box setting, in which they have access to the all the parameters of models, what kind of loss function it was trained, and perturb the input in such a way that causes the model to fail. While there are claims that this threat model is unrealistic, it can used as a valuable tool to analyze the true robustness

of neural network classifiers and training methods.

Pixel-based adversarial attacks

Adversarial examples are inputs that are imperceptibly perturbed such that they cause classifiers to completely fail [149–152]. They are generated through gradient-based algorithms such as the fast gradient sign method (FGSM) or the iterative version, the projected gradient descent(PGD) attack. The PGD formulation is given below as:

$$x^{t+1} = \Pi_{x+\epsilon}(x^t + \alpha sgn(\nabla_x L(\theta, x, y))). \tag{6.1}$$

The input image, x, is altered by the signed gradient of the loss with respect to the input, scaled by α, a step size for gradient descent, similar to the learning rate used in conventional training. If any value in x exceeds the range of allowable distance of the threat model $(x - \epsilon, x + \epsilon)$, it is clipped to the valid range. The algorithm is then iterated on a certain number of steps until it reaches convergence.

Adversarial training is known to be the state of the art defense method against adversarial attacks [151, 152]. It is formulated based on minimizing the expected adversarial loss on a worst case sample via min-max optimization:

$$\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\sim D} [\max_{\delta \in \Omega} L(\theta, x + \delta, y)]. \tag{6.2}$$

Here, θ refers to the model parameters; δ refers to the perturbation added to the sample, which is bounded by Ω to ensure that the adversarial example is in close proximity to the original sample, while the inputs, x, y, are drawn from the given data distribution, D. Since the inner maximization loop of the adversarial loss is an intractable optimization, an approximation is computed using methods like PGD, which are sufficient to obtain decent performance [151].

Lottery Ticket Hypothesis

The lottery ticket hypothesis is a recent proposal that provides a new perspective on network pruning [153]. It states that within a neural network, there exists smaller subnetworks that are able to achieve comparable or better generalization than the full capacity model. The important factor to obtaining these subnetworks is maintaining a specific set of weights at initialization; these subnetworks are called winning tickets. According to Frankle *et al.*[153], winning tickets can be 10-20% the size of the original model and can generalize faster than full capacity models.

The process of finding winning tickets is expensive. The parameters of the model are first saved at initialization and then trained to convergence. Each layer is assigned a binary mask to note which parameters are pruned or not; then $p\%$ of the lowest magnitude parameters are pruned and the corresponding positions of the pruned parameters in the mask are set to 0. The network is then reset with the saved initialization parameters with the binary mask applied and retrained. This iterative pruning process is repeated n times until a sufficiently pruned, high accuracy model is obtained. As iterated above, the specific initialized weights are necessary to obtained winning tickets; Frankle *et al.*[153] demonstrated this by performing the same pruning scheme but randomly reinitialized the unpruned parameters and observed the test performance drop as the model became more sparse.

For larger models, it has proven to be challenging to find winning tickets. Pruning of the weight parameters can be done through two methods: layerwise and globalwise pruning. Layerwise pruning involves through each layer and deleting $p\%$ of the lowest magnitude weights whereas global pruning looks at the network weight magnitudes in totality and removes the lowest $p\%$ weight magnitude. For deeper networks, Morcos *et al.*found [137, 153] that global pruning yielded better winning tickets than layerwise pruning; more specifically, they reported that globally pruned networks showed sparser connections in latter layers. Potentially, pruning parameters at earlier layers where smaller connections are present is too damaging for the network to recover. Another important parameter to that influences

winning tickets is the learning rate. Conventionally, a nominal learning rate for training ResNet18 on CIFAR10 to convergence is 0.1. However, even with warmup, this learning rate is too high to yield any winning tickets. In follow-up work, Frankle *et al.*[136] developed an approach called "weight rewinding" which states: rather than resetting the unpruned parameters back to their initial values, reset them to the values after the first few epochs of training. Empirically, this eliminates the need to use warmup learning rate schedules to reduce the size of deep networks. We perform layerwise pruning for shallower networks and global pruning for deeper ones.

6.3.2 Is LTH true with adversarial training?

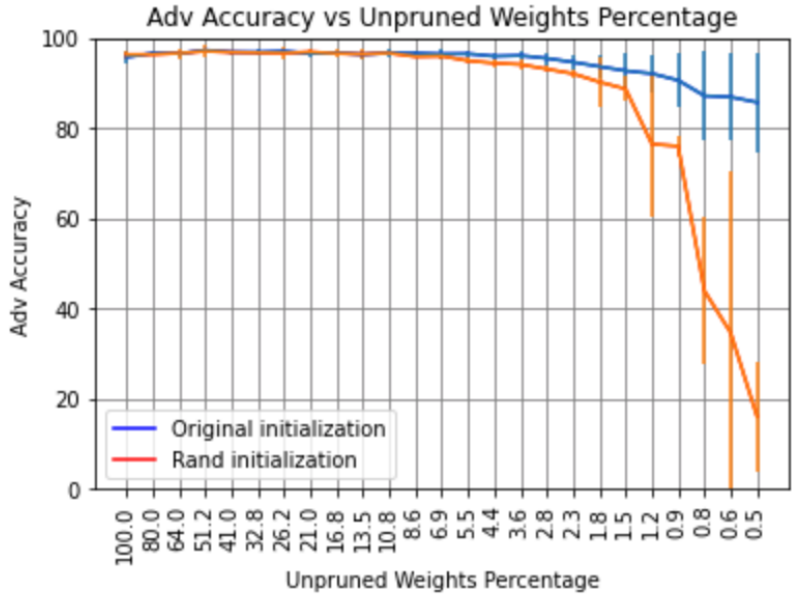
Deep learning has been successful in terms of achieving high accuracy as networks have been getting deeper. The extreme overparameterization of these networks seem to violate classical statistics bias-variance tradeoff curves such that they are still able to generalize well. When considering adversarial robustness, the overparameterization of networks seems to be even more necessary for models to have decent adversarially robust generalization. Furthermore, it has been posited in multiple different works that there is a tradeoff between accuracy and robustness [141, 154, 155]. One explanation to why such a tradeoff exists is that current classifiers are not complex enough to represent decision boundaries that are sufficiently far enough from the data samples [140, 156].

The Lottery Ticket Hypothesis (LTH) seems to be counter-intuitive to the idea of over-parameterizing neural networks. The existence of winning tickets demonstrates that overparameterization is not necessary if initialization schemes can be improved. The LTH's success with uncovering sparse, accurate models begs the question, "Is the LTH true for the adversarial setting?" That is, do winning tickets exist such that they have the same or better adversarial accuracy compared to their dense counterpart? If true, this hypothesis would suggest, contrary to conventional wisdom, there exists sparse, robust subnetworks which are able to match the adversarial accuracy of its dense counterpart.

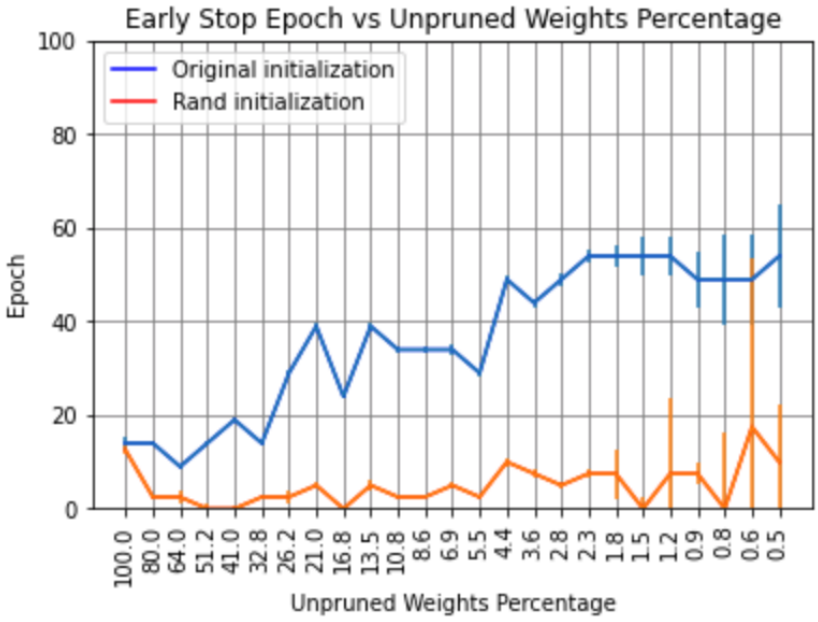
LTH: Initial Experiments and Results

Specially, we study the effect of the LTH in the adversarial context by substituting standard training with adversarial training. Given a neural network $f(\theta, x_{in}, y_{target})$ parameterized by θ, the LTH procedure uses iterative magnitude pruning(IMP) to produce the binary masks, M_x to obtain a potential winning ticket. IMP is executed through global pruning, where a percentage of the lowest magnitude weights are set to zero. Recall if the adversarial accuracy of the resulting pruned network matches the adversarial accuracy of its dense counterpart, it is deemed a winning ticket and would suggest that networks do not need to be overparameterized for robust learning. In addition, we analyze the impact of weight rewinding/late resetting for larger models. In prior work, weight rewinding was shown to be effective in enabling IMP to find tickets on larger models and datasets [136].

We perform experiments on the MNIST and CIFAR10[99] datasets. On MNIST, we use a LeNet5 architecture and train it for 85 epochs. We select Adam as the optimizer with learning rate of 0.01, along with the weight decay parameter, $\lambda_{wd} = 5e-4$ and incorporate a piecewise linear decay learning scheduler, setting the batch size to 128. For adversarial training, we set $\epsilon = 0.1$ and trained each mini-batch for 7 PGD steps. During training, we took 5000 examples from the training set to use as a holdset to early stop training in the event of overfitting. The final adversarial accuracy reported was against a 50-step PGD adversary with 10 random restarts with step size $\alpha = 0.02$. Once the first training round was completed, we performed layer-wise pruning on the convolutional and fully-connected layers with a pruning rate of 20%. We omit pruning the final layer of the network as 1) there are not a large number of parameters in final layer of the network and 2) by eliminating parameters in the penultimate layer could potentially have a drastic impact on classification accuracy. We then reset the parameters back to their initial values with the updated pruned masks and repeat the training-pruning process 25 times. To prove that the initialization parameters are the driving force in achieving the sparse network, we include a random initialization control baseline, which we randomly reinitialize the model parameters after pruning. Figure 6.1a



(a) MNIST LeNet5 adversarial accuracy



(b) Early Stop Epoch

Figure 6.1: Figure 6.1a reports the adversarial accuracy of LeNet5 on MNIST when pruned iteratively reported over 3 random seeds. At 3.6% of the original model size, winning tickets do exist and are able to outperform the random reinitialization baseline. Figure 6.1b refers to the epoch which the winning ticket reaches the best performance on a holdout validation set reported over 3 random seeds.

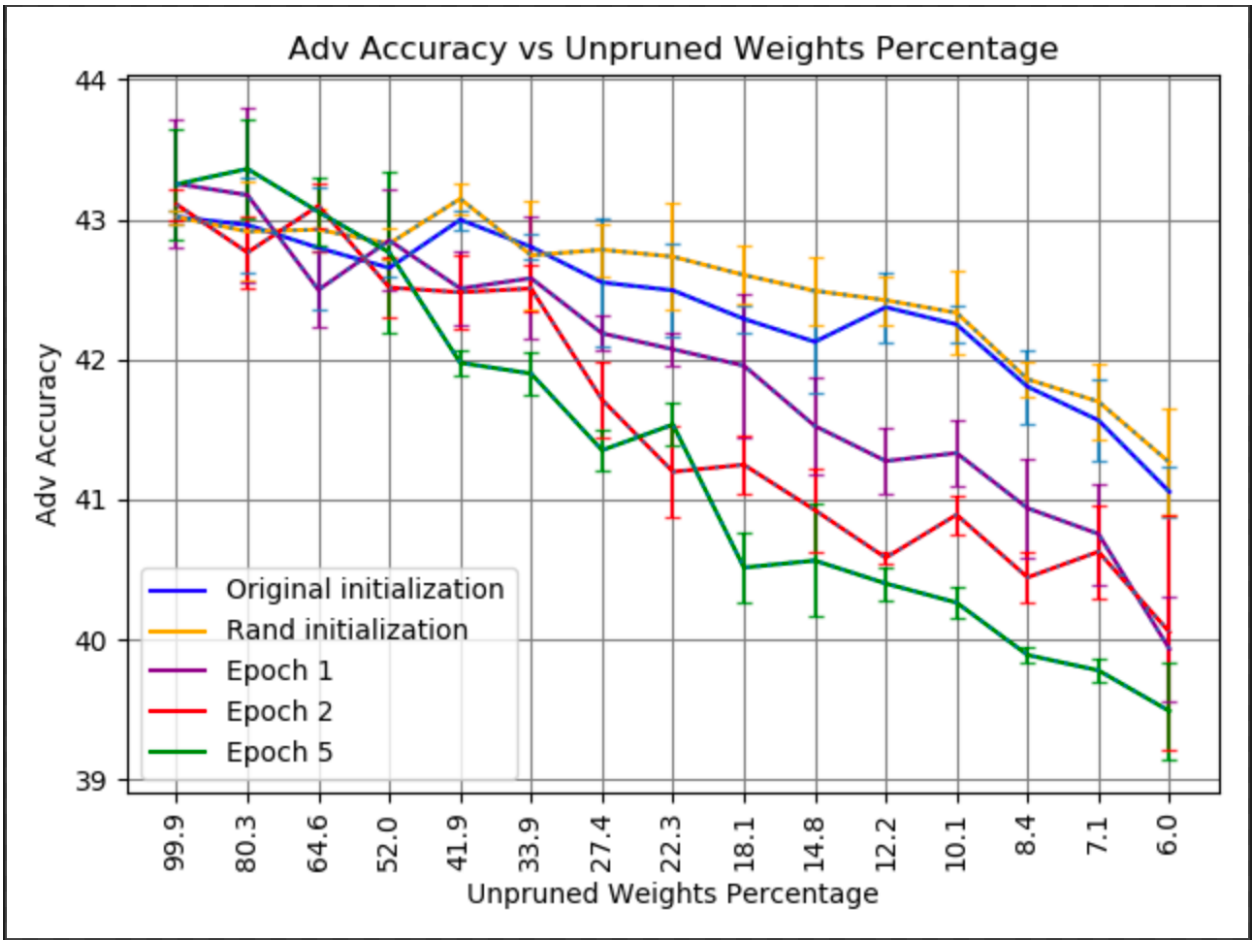
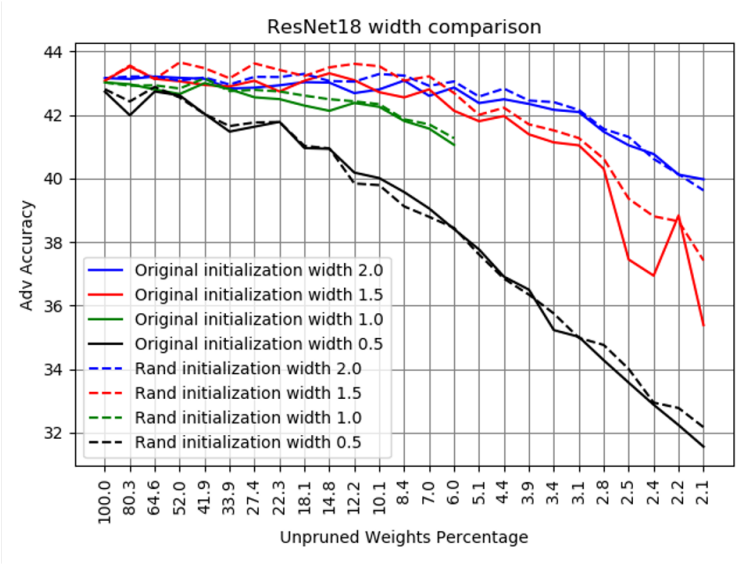


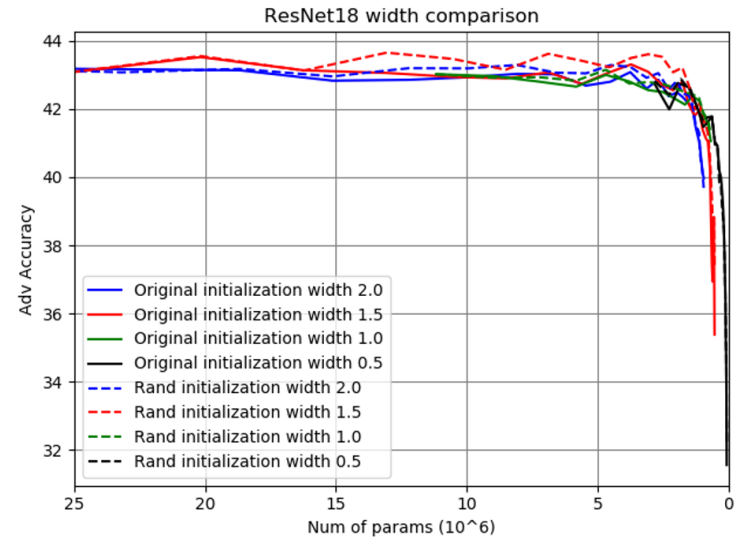
Figure 6.2: Adversarial accuracy of PreResNet18 on CIFAR10 when pruned iteratively reported over 3 random seeds. Random reinitialization either outperforms or has same performance the original initialization. The epoch lines refer to rewinding to earlier phases in training. Contrary to the phenomenon that occurs in the natural setting, weight rewinding seems to hurt adversarial accuracy.

reports the adversarial accuracy of the LeNet5 model as more parameters are being pruned as well as the early stop epoch before training is halted over 3 different random initial seeds. It is clear to see that winning tickets exist for MNIST, evidenced by the LeNet5 model at 3.6% of the original model capacity achieving the same performance compared with the model retaining all parameters. Contrary to prior observations [136, 153], however, the early stop epoch for the original initialization network happens much later in training. The results in Figure 6.1a state that the lottery ticket hypothesis with adversarial training holds true under simplistic models and datasets like LeNet5 and MNIST. We next describe our training setup for CIFAR10.

For CIFAR10, we perform adversarial training with a PreResNet18 and train it for 50 epochs. We select SGD as our optimizer with momentum=0.9, set the weight decay parameter, $\lambda_{wd} = 5e-4$, and the batch size to 128. We use a cyclic learning rate scheduler[102] setting our maximum learning rate as 0.2 and the minimum learning rate as 0 over all the epochs. Due to standard adversarial training being both time-consuming and computationally expensive, we train with a one-step FGSM adversary [143] with $\epsilon = 8/255.0$ with random initialization. The adversary we evaluate against is a 50-step PGD adversary with $\epsilon = 8/255.0$ and step size $\alpha = 2/255.0$ with 10 random restarts. Once training completes, we globally prune 20% of the lowest magnitude weights, omitting the linear layers as well as the weights on the skip connections, as they comprise a small number of weights compared to the convolutional layers. To study the effect of weight rewinding, we replace the initialization parameters with those at Epoch 1, 2, and 5. We perform 15 rounds of training-pruning cycles. We also include the random reinitialization baseline to test the effect of original lottery ticket initialization. Figure 6.2 shows the relationship of the adversarial accuracy of PreResNet18 initial initialization as it becomes pruned. We observe that at trivial sparsities as $80 - 41.9\%$ both initial and random initializations are able to maintain the adversarial accuracy but begins to drop as more parameters get pruned. This result seems to suggest that sparsity is at odds with robustness and implies that overparmeterized networks seem



(a) PreResNet18 CIFAR10 width comparison



(b) PreResNet18 CIFAR10 parameter count comparison

Figure 6.3: Figure 6.3a shows the impact of altering the width of the PreResNet18 model by 0.5, 1.5, and 2.0 by 0.5, 1.5, 2.0 denoted by the black, red, and blue lines. Contrary to Figure 6.2, winning tickets do emerge for the overparameterized models at 12.2% for width of 1.5 and 8.4% for width of 2.0. Figure 6.3b compares the different width configurations by comparing them with respect to parameter count. The trend suggests the performance of lightly pruned smaller models matches those of winning tickets when we normalize for parameter count between different configurations. This leads to the conclusion that the LTH will find winning tickets but the resulting network will continue to be overparameterized.

necessary for more complex image classification tasks. Interestingly, it indicates that, contrary to the standard training setting, weight rewinding hurts the adversarial accuracy of the model and it worsens as the reset point is set later.

One potential reason for iterative magnitude pruning not being to identify winning tickets is that the current models in use may not be sufficiently overparameterized. We explore this hypothesis by varying the width of by inflating/deflating the number of filters in each layer of the PreResNet18 as shown in Figure 6.3. The widths explored are: 0.5x, 1.5x, and 2.0x of the baseline PreResNet18 model. In Figure 6.3a, we observe that winning tickets do emerge for the width-augmented 1.5x PreResNet18 at 12.2% and width-augmented 2.0x PreResNet18 at 8.4%. If we compare the performance of all models with respect to the number of parameters, a lightly pruned dense model achieves similar robust accuracy compared to winning tickets of additionally overparameterized models.

For future work, we would like to understand why weight rewinding causes the performance of the model to suffer and test lottery tickets can be still be found on more complex tasks like ImageNet. Another potential direction to see if we can improve the adversarial accuracy of the lottery tickets: given a winning ticket initialization, is it possible to reintroduce or revive connections to boost the adversarial accuracy of the original model. Work on the lottery ticket hypothesis has been solely focused on destroying low weights in the network but there has been a plethora of pruning criteria that been introduced that may be better aligned with preserving adversarial accuracy than traditional weight pruning; we wish explore these criteria for future work.

6.3.3 Opinions on AML

After studying adversarial robustness for what seems to be 2 years, it seems like the prevailing model to obtaining a robust model is that the neural network should be treated as a black-box function and the adversarial input is obtained as a function of the network function and the natural input. This view is expressed by the adversarial training methods [139, 140, 143,

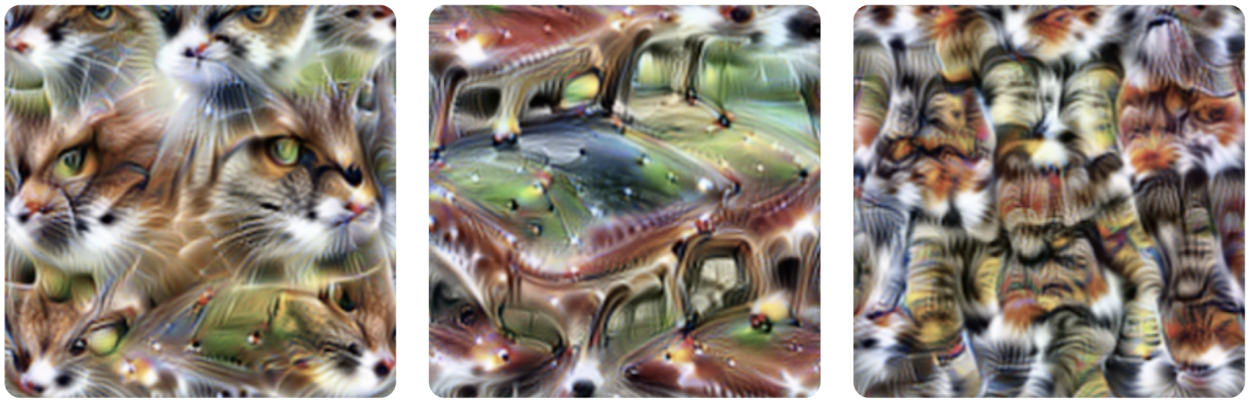


Figure 6.4: Intermediate feature representation of InceptionV1 of a polysemantic neuron that responds to cat faces, fronts of cars, and cat legs. Figure from [157].

144, 151] as well as other approaches such as randomized smoothing. I have some reservations about looking at AML through this lens. Firstly, AML has been often framed as a security threat problem where by if an adversary gets access to a classifier. If an adversary has white-box access to the model, there were not enough measures in the first place to thwart an attack, which is a purely security problem rather than an AML problem; moreover, many of the defenses proposed in literature are in response to pixel-based attacks which really cannot be replicated in the physical world. Despite these criticisms, AML offers value in terms of understanding what the neural network learns and the incongruity between how humans and machines recognize objects. However, this perspective cannot be appreciated by solely analyzing the network as a black box but should be examined at a neuron-neuron level to understand how adversarial noise is propagated through each layer and affects the classification output.

Olah *et al.*[157] advocates for analyzing neural networks as circuit models by zooming in on individual weights and features and building an understanding of network interpretation from the ground up. This model is built on three claims: 1) Features are the fundamental unit of neural networks, 2) Features are connected by weights, forming circuits which can be rigorously studied and 3) analogous features and circuits form across models and tasks. Features can be discovered through many different avenues: feature visualization which is

the optimizing the input to cause curve detectors to fire reliably produces curves, dataset examples and synthetic examples that maximally excite the feature that particular neuron has learned. Based on the hierarchy of layers, this can lead to filter representations which are complementary with each other and even combine together to form higher, more complex representations. Olah *et al.*[157] then go on to mention the interesting phenomenon that are known as "polysemantic neurons".

Shown in Figure 6.4, polysemantic neurons are those that are activated by unrelated inputs. The rationale for these neurons existing is that that there are too many features in the data for the network to learn so the network uses neurons for different classes. For example, Figure 6.4 shows that this particular filter is activated both cat and car features, which to humans seem totally disparate objects. Most likely, these polysemantic neurons are a potential cause of what is causing networks to fail under adversarial attacks since neurons are being shared by seemingly unrelated classes. I think by examining the difference how natural and adversarial inputs are propagated we can have a better empirical understanding of where the failure case is happening. In this sense, I think the circuit model of viewing neural networks as conglomerations of feature detectors is potentially more useful than analyzing it as a black box.

It would be worthwhile to see if polysemantic neurons indeed have this parasitic effect on adversarial robustness on deep learning models. A first stab at ameliorating this problem is a solution from the 90's called predictability minimization [158, 159]. The basic idea behind these works is that each neuron would be penalized if it fired when another neuron also fired; this means that neurons will forced to learn unique features and the phenomenon of polysemantic neurons would be mitigated to some degree. Consequently, this could have implications on network pruning and sparsity [10, 91]. A reason given why network pruning works is that networks carry so much redundancy resulting from their overparameterized nature. Potentially by reducing the number of polysemantic neurons in a network allows for more compact representation of models and limits the sparsity we can induce in the

weights. Regardless, its clear that the circuits models poses lots of interesting thoughts and experiment for future work.

Bibliography

[1] Muzammal Naseer et al. *Intriguing Properties of Vision Transformers*. Tech. rep. URL: https://git.io/Js15X..

[2] Mathilde Caron et al. "Emerging Properties in Self-Supervised Vision Transformers". In: (Apr. 2021). DOI: 10.48550/arxiv.2104.14294. URL: http://arxiv.org/abs/2104.14294.

[3] Tom B. Brown et al. "Language models are few-shot learners". In: *Advances in Neural Information Processing Systems*. 2020.

[4] Aakanksha Chowdhery et al. "PaLM: Scaling Language Modeling with Pathways". In: (Apr. 2022). DOI: 10.48550/arxiv.2204.02311. URL: http://arxiv.org/abs/2204.02311.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* (2017). ISSN: 15577317. DOI: 10.1145/3065386.

[6] Noam Shazeer et al. "Mesh-tensorflow: Deep learning for supercomputers". In: *Advances in Neural Information Processing Systems*. 2018.

[7] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017.

[8] Alexey Dosovitskiy et al. *AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE*. Tech. rep. URL: https://github.com/.

[9] Vikash Sehwag et al. *On Pruning Adversarially Robust Neural Networks*. 2020.

[10] Tailin Liang et al. "Pruning and Quantization for Deep Neural Network Acceleration: A Survey". In: (Jan. 2021). DOI: 10.48550/arxiv.2101.09671. URL: http://arxiv.org/abs/2101.09671.

[11] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: (Oct. 2015). DOI: 10.48550/arxiv.1510.00149. URL: http://arxiv.org/abs/1510.00149.

[12] Yash Bhalgat et al. "LSQ+: Improving low-bit quantization through learnable offsets and better initialization". In: (Apr. 2020). DOI: 10.48550/arxiv.2004.09576. URL: http://arxiv.org/abs/2004.09576.

[13] Steven K. Esser et al. "Learned Step Size Quantization". In: (Feb. 2019). DOI: 10.48550/arxiv.1902.08153. URL: http://arxiv.org/abs/1902.08153.

[14] Sangil Jung et al. "Learning to Quantize Deep Networks by Optimizing Quantization Intervals with Task Loss". In: (Aug. 2018). DOI: 10.48550/arxiv.1808.05779. URL: https://arxiv.org/abs/1808.05779.

[15] Markus Nagel et al. "Data-free quantization through weight equalization and bias correction". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00141.

[16] Dibakar Gope et al. "Ternary MobileNets via Per-Layer Hybrid Filter Banks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2020.

[17] Dibakar Gope, Ganesh Dasika, and Matthew Mattina. "Ternary Hybrid Neural-Tree Networks for Highly Constrained IoT Applications". In: *Proceedings of Machine Learning and Systems 2019*. 2019, pp. 190–200.

[18] Dibakar Gope et al. "Aggressive Compression of MobileNets Using Hybrid Ternary Layers". In: *tinyML Summit* (2020). URL: https://www.tinyml.org/summit/abstracts/Gope_Dibakar_poster_abstract.pdf.

[19] Xitong Gao et al. *Dynamic Channel Pruning: Feature Boosting and Suppression*. 2018.

[20] Yichi Zhang et al. *Precision Gating: Improving Neural Network Efficiency with Dynamic Dual-Precision Activations*. 2020.

[21] Yihui He, Xiangyu Zhang, and Jian Sun. *Channel Pruning for Accelerating Very Deep Neural Networks*. 2017.

[22] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. *Batch-Shaping for Learning Conditional Channel Gated Networks*. 2019.

[23] Brandon Yang et al. "CondConv: Conditionally Parameterized Convolutions for Efficient Inference". In: *Advances in Neural Information Processing Systems 32*. Ed. by H Wallach et al. Curran Associates, Inc., 2019, pp. 1307–1318. URL: http://papers.nips.cc/paper/8412-condconv-conditionally-parameterized-convolutions-for-efficient-inference.pdf.

[24] Zuxuan Wu et al. "BlockDrop: Dynamic Inference Paths in Residual Networks". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018. DOI: 10.1109/CVPR.2018.00919.

[25] Ji Lin et al. "Runtime neural pruning". In: *Advances in Neural Information Processing Systems*. Vol. 2017-Decem. 2017.

[26] Noam Shazeer et al. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. 2017.

[27] Sharan Narang et al. “Mixed {Precision} {Training}”. In: *International {Conference} on {Learning} {Representations} 2018*. 2018, p. 12. URL: https://openreview.net/forum?id=r1gs9JgRZ.

[28] Alex Krizhevsky. “Learning {Multiple} {Layers} of {Features} from {Tiny} {Images}”. In: (2009), p. 60.

[29] Chirag Agarwal, Daniel D’souza, and Sara Hooker. “Estimating {Example} {Difficulty} {Using} {Variance} of {Gradients}”. In: *arXiv:2008.11600 [cs]* (Sept. 2021). URL: http://arxiv.org/abs/2008.11600.

[30] Mariya Toneva et al. “An {Empirical} {Study} of {Example} {Forgetting} {During} {Deep} {Neural} {Network} {Learning}”. In: *International {Conference} on {Learning} {Representations} 2019*. 2019, p. 18. URL: https://openreview.net/forum?id=BJlxm30cKm.

[31] Yoshua Bengio et al. “Curriculum learning”. In: *Proceedings of the 26th {Annual} {International} {Conference} on {Machine} {Learning} - {ICML} ’09*. Montreal, Quebec, Canada: ACM Press, 2009, pp. 1–8. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380. URL: http://portal.acm.org/citation.cfm?doid=1553374.1553380.

[32] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. “Deep {Learning} on a {Data} {Diet}: {Finding} {Important} {Examples} {Early} in {Training}”. In: *35th {Conference} on {Neural} {Information} {Processing} {Systems}*. Vol. 34. Curran Associates, Inc., 2021, p. 12. URL: https://proceedings.neurips.cc/paper/2021/hash/ac56f8fe9eea3e4a365f29f0f1957c55-Abstract.html.

[33] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: (Dec. 2020). DOI: 10.48550/arxiv.2012.12877. URL: http://arxiv.org/abs/2012.12877.

[34] Hugo Touvron, Matthieu Cord, and Hervé Jégou. "DeiT III: Revenge of the ViT". In: (Apr. 2022). DOI: 10.48550/arxiv.2204.07118. URL: http://arxiv.org/abs/2204.07118.

[35] Mingxing Tan and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training". In: (Apr. 2021). DOI: 10.48550/arxiv.2104.00298. URL: https://arxiv.org/abs/2104.00298.

[36] Ravi S Raju, Kyle Daruwalla, and Mikko Lipasti. "Accelerating Deep Learning with Dynamic Data Pruning". In: (Nov. 2021). DOI: 10.48550/arxiv.2111.12621. URL: https://arxiv.org/abs/2111.12621.

[37] Ravi Raju et al. "Understanding the Impact of Dynamic Channel Pruning on Conditionally Parameterized Convolutions". In: *AIChallengeIoT 2020 - Proceedings of the 2020 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*. 2020. ISBN: 9781450381345. DOI: 10.1145/3417313.3429381.

[38] Yongming Rao et al. *DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification*. Tech. rep. URL: https://github.com/.

[39] Yifan Xu et al. "Evo-ViT: Slow-Fast Token Evolution for Dynamic Vision Transformer". In: (Aug. 2021). DOI: 10.48550/arxiv.2108.01390. URL: http://arxiv.org/abs/2108.01390.

[40] Lingchen Meng et al. *AdaViT: Adaptive Vision Transformers for Efficient Image Recognition*. Tech. rep.

[41] Mingbao Lin et al. "Super Vision Transformer". In: (May 2022). DOI: 10.48550/arxiv.2205.11397. URL: http://arxiv.org/abs/2205.11397.

[42] Zhenglun Kong et al. "SPViT: Enabling Faster Vision Transformers via Soft Token Pruning". In: (Dec. 2021). DOI: 10.48550/arxiv.2112.13890. URL: http://arxiv.org/abs/2112.13890.

[43] Aakanksha Chowdhery et al. “Visual Wake Words Dataset”. In: (June 2019). DOI: 10.48550/arxiv.1906.05721. URL: https://arxiv.org/abs/1906.05721.

[44] Sam Leroux et al. “The cascading neural network: building the Internet of Smart Things”. In: *Knowledge and Information Systems* 52.3 (2017). ISSN: 02193116. DOI: 10.1007/s10115-017-1029-1.

[45] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. “BranchyNet: Fast inference via early exiting from deep neural networks”. In: *Proceedings - International Conference on Pattern Recognition*. Vol. 0. 2016. DOI: 10.1109/ICPR.2016.7900006.

[46] J Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[47] Pengzhen Ren et al. *A Survey of Deep Active Learning*. 2020.

[48] Yanyao Shen et al. *Deep Active Learning for Named Entity Recognition*. 2018.

[49] Burr Settles. *Active {Learning} {Literature} {Survey}*. Tech. rep. 2009, p. 67. URL: http://www.burrsettles.com/pub/settles.activelearning.pdf.

[50] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. “Active hidden markov models for information extraction”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 2189. 2001. DOI: 10.1007/3-540-44816-0{_}31.

[51] Ozan Sener and Silvio Savarese. *Active Learning for Convolutional Neural Networks: A Core-Set Approach*. 2018.

[52] Yonatan Geifman and Ran El-Yaniv. “Deep Active Learning over the Long Tail”. In: (Nov. 2017). DOI: 10.48550/arxiv.1711.00941. URL: https://arxiv.org/abs/1711.00941.

[53] Xuefeng Du, Dexing Zhong, and Huikai Shao. "Building an Active Palmprint Recognition System". In: *Proceedings - International Conference on Image Processing, ICIP*. Vol. 2019-September. 2019. DOI: 10.1109/ICIP.2019.8803135.

[54] Changchang Yin et al. "Deep Similarity-Based Batch Mode Active Learning with Exploration-Exploitation". In: (2017). DOI: 10.1109/ICDM.2017.67.

[55] Jordan T Ash et al. *Deep Batch Active Learning by Diverse, Uncertain Gradient Lower Bounds*. 2020.

[56] Richard S Sutton and Andrew G Barto. *Reinforcment {Learning}: {An} {Introduction}*. 2nd ed. Adaptive {Computation} and {Machine} {Learning}. MIT Press, 2018. ISBN: 978-0-262-03924-6. URL: http://incompleteideas.net/book/RLbook2020.pdf.

[57] Kai Arulkumaran et al. "A Brief Survey of Deep Reinforcement Learning". In: (Aug. 2017). DOI: 10.1109/msp.2017.2743240. URL: https://arxiv.org/abs/1708.05866.

[58] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: (2015). DOI: 10.1038/nature14236.

[59] Yoon Sang Lee and Riyaz Sikora. "Application of adaptive strategy for supply chain agent". In: *Information Systems and e-Business Management* 17.1 (2019). ISSN: 16179854. DOI: 10.1007/s10257-018-0378-y.

[60] Jen Jen Chung, Nicholas R.J. Lawrance, and Salah Sukkarieh. "Learning to soar: Resource-constrained exploration in reinforcement learning". In: *International Journal of Robotics Research* 34.2 (2015). ISSN: 17413176. DOI: 10.1177/0278364914553683.

[61] Michel Tokic. "Adaptive ϵ-greedy exploration in reinforcement learning based on value differences". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6359 LNAI. 2010. DOI: 10.1007/978-3-642-16111-7{_}23.

[62] Jürgen Schmidhuber. "Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010)". In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010). DOI: 10.1109/TAMD.2010.2056368.

[63] Shakir Mohamed and Danilo Jimenez Rezende. "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning". In: (Sept. 2015). DOI: 10.48550/arxiv.1509.08731. URL: http://arxiv.org/abs/1509.08731.

[64] Rein Houthooft et al. "VIME: Variational Information Maximizing Exploration". In: (May 2016). DOI: 10.48550/arxiv.1605.09674. URL: http://arxiv.org/abs/1605.09674.

[65] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (Oct. 2018). DOI: 10.48550/arxiv.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[66] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: (Jan. 2020). DOI: 10.48550/arxiv.2001.04451. URL: https://arxiv.org/abs/2001.04451.

[67] Alexander Kolesnikov et al. "Big Transfer (BiT): General Visual Representation Learning". In: ().

[68] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)". In: (June 2016). DOI: 10.48550/arxiv.1606.08415. URL: https://arxiv.org/abs/1606.08415.

[69] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: (July 2016). DOI: 10.48550/arxiv.1607.06450. URL: http://arxiv.org/abs/1607.06450.

[70] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: (Mar. 2015). DOI: 10.48550/arxiv.1503.02531. URL: http://arxiv.org/abs/1503.02531.

[71] Li Yuan et al. “Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet”. In: (Jan. 2021). DOI: 10.48550/arxiv.2101.11986. URL: https://arxiv.org/abs/2101.11986.

[72] Ben Graham et al. “LeViT: a Vision Transformer in ConvNet’s Clothing for Faster Inference”. In: (Apr. 2021). DOI: 10.48550/arxiv.2104.01136. URL: http://arxiv.org/abs/2104.01136.

[73] Salman Khan et al. “Transformers in Vision: A Survey”. In: (Jan. 2021). DOI: 10.1145/3505244. URL: https://arxiv.org/abs/2101.01169.

[74] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. “Self-labelling via simultaneous clustering and representation learning”. In: (Nov. 2019). DOI: 10.48550/arxiv.1911.05371. URL: http://arxiv.org/abs/1911.05371.

[75] Kaiming He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: (Nov. 2019). DOI: 10.48550/arxiv.1911.05722. URL: https://arxiv.org/abs/1911.05722.

[76] Zhirong Wu et al. “Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination”. In: (May 2018). DOI: 10.48550/arxiv.1805.01978. URL: https://arxiv.org/abs/1805.01978.

[77] Jean-Bastien Grill et al. “Bootstrap your own latent: A new approach to self-supervised Learning”. In: (June 2020). DOI: 10.48550/arxiv.2006.07733. URL: https://arxiv.org/abs/2006.07733.

[78] Mathilde Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: (June 2020). DOI: 10.48550/arxiv.2006.09882. URL: https://arxiv.org/abs/2006.09882.

[79] Song Han, Huizi Mao, and William J Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *CoRR* abs/1510.0 (2015).

[80] Urmish Thakker et al. “Pushing the limits of RNN Compression”. In: *CoRR* abs/1910.0 (2019).

[81] Urmish Thakker et al. “Run-Time Efficient {RNN} Compression for Inference on Edge Devices”. In: *CoRR* abs/1906.0 (2019).

[82] Urmish Thakker et al. “Compressing RNNs for IoT devices by 15-38x using Kronecker Products”. In: *CoRR* abs/1906.0 (2019).

[83] Urmish Thakker et al. *Rank and run-time aware compression of NLP Applications*. 2020.

[84] Urmish Thakker et al. *Compressing Language Models using Doped Kronecker Products*. 2020.

[85] Forrest N Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size*. 2016.

[86] Andrew G Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017.

[87] Andrew Howard et al. *Searching for MobileNetV3*. 2019.

[88] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2018.

[89] Jie Hu et al. *Squeeze-and-Excitation Networks*. 2017.

[90] Xiangyu Zhang et al. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. 2017.

[91] Song Han, Huizi Mao, and William J Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2015.

[92] Xiangyu Zhang et al. *Accelerating Very Deep Convolutional Networks for Classification and Detection*. 2015.

[93] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. *Speeding up Convolutional Neural Networks with Low Rank Expansions*. 2014.

[94] Pavlo Molchanov et al. *Pruning Convolutional Neural Networks for Resource Efficient Inference*. 2016.

[95] Yinpeng Chen et al. *Dynamic Convolution: Attention over Convolution Kernels*. 2019.

[96] Weizhe Hua et al. *Channel Gating Neural Networks*. 2018.

[97] Jin Tao et al. "Skipping RNN State Updates without Retraining the Original Model". In: *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems*. SenSys-ML 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 31–36. ISBN: 9781450370110. DOI: 10.1145/3362743.3362965. URL: https://doi.org/10.1145/3362743.3362965.

[98] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[99] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[100] Terrance DeVries and Graham W Taylor. *Improved Regularization of Convolutional Neural Networks with Cutout*. 2017.

[101] T Tieleman and G Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[102] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2016.

[103] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015.

[104] Prajit Ramachandran, Barret Zoph, and Quoc V Le. *Searching for Activation Functions*. 2017.

[105] Rishi Bommasani et al. “On the {Opportunities} and {Risks} of {Foundation} {Models}”. In: *arXiv:2108.07258 [cs]* (Aug. 2021). URL: http://arxiv.org/abs/2108.07258.

[106] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and {Policy} {Considerations} for {Deep} {Learning} in {NLP}”. In: *Proceedings of the 57th {Annual} {Meeting} of the {Association} for {Computational} {Linguistics}*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355. URL: https://www.aclweb.org/anthology/P19-1355.

[107] Qi-Zhi Cai, Chang Liu, and Dawn Song. “Curriculum {Adversarial} {Training}”. In: *Proceedings of the {Twenty}-{Seventh} {International} {Joint} {Conference} on {Artificial} {Intelligence}*. Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 3740–3747. ISBN: 978-0-9992411-2-7. DOI: 10.24963/ijcai.2018/520. URL: https://www.ijcai.org/proceedings/2018/520.

[108] Cody Coleman et al. “Selection via {Proxy}: {Efficient} {Data} {Selection} for {Deep} {Learning}”. In: *International {Conference} on {Learning} {Representations} 2020*. 2020, p. 25. URL: https://openreview.net/pdf?id=HJg2b0VYDr.

[109] B P Welford. “Note on a {Method} for {Calculating} {Corrected} {Sums} of {Squares} and {Products}”. In: *Technometrics* 4.3 (Aug. 1962), pp. 419–420. ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.1962.10490022. URL: http://www.tandfonline.com/doi/abs/10.1080/00401706.1962.10490022.

[110] Kaiming He et al. “Deep {Residual} {Learning} for {Image} {Recognition}”. In: *2016 {IEEE} {Conference} on {Computer} {Vision} and {Pattern} {Recognition} ({CVPR})*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/.

[111] Adam Paszke et al. “Automatic differentiation in {PyTorch}”. In: *31st {Conference} on {Neural} {Information} {Processing} {Systems}*. Vol. 30. 2017, p. 4. URL: https://openreview.net/pdf?id=BJJsrmfCZ.

[112] Hong Ge, Kai Xu, and Zoubin Ghahramani. “Turing: A Language for Flexible Probabilistic Inference”. In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1682–1690. URL: https://proceedings.mlr.press/v84/ge18b.html.

[113] Jeff Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: (Nov. 2014). DOI: 10.48550/arxiv.1411.1607. URL: https://arxiv.org/abs/1411.1607.

[114] Christian P. Robert. “The Metropolis-Hastings algorithm”. In: (Apr. 2015). DOI: 10.48550/arxiv.1504.01896. URL: https://arxiv.org/abs/1504.01896.

[115] Rewon Child et al. “Generating Long Sequences with Sparse Transformers”. In: (Apr. 2019). DOI: 10.48550/arxiv.1904.10509. URL: http://arxiv.org/abs/1904.10509.

[116] Sinong Wang et al. “Linformer: Self-Attention with Linear Complexity”. In: (June 2020). DOI: 10.48550/arxiv.2006.04768. URL: https://arxiv.org/abs/2006.04768.

[117] Yehui Tang et al. *Patch Slimming for Efficient Vision Transformers*. Tech. rep.

[118] Yulin Wang et al. *Not All Images are Worth 16x16 Words: Dynamic Transformers for Efficient Image Recognition*. Tech. rep. URL: https://github.com/.

[119] Zhuofan Zong et al. “Self-slimmed Vision Transformer”. In: (Nov. 2021). DOI: 10.48550/arxiv.2111.12624. URL: http://arxiv.org/abs/2111.12624.

[120] Norman P. Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit”. In: (Apr. 2017). DOI: 10.48550/arxiv.1704.04760. URL: http://arxiv.org/abs/1704.04760.

[121] Mohammad Shoeybi et al. “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism”. In: (Sept. 2019). DOI: 10.48550/arxiv.1909.08053. URL: https://arxiv.org/abs/1909.08053.

[122] Wenhai Wang et al. “Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions”. In: (Feb. 2021). DOI: 10.48550/arxiv.2102.12122. URL: https://arxiv.org/abs/2102.12122.

[123] Byeongho Heo et al. “Rethinking Spatial Dimensions of Vision Transformers”. In: (Mar. 2021). DOI: 10.48550/arxiv.2103.16302. URL: http://arxiv.org/abs/2103.16302.

[124] Sangdoo Yun et al. “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features”. In: (May 2019). DOI: 10.48550/arxiv.1905.04899. URL: http://arxiv.org/abs/1905.04899.

[125] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (Dec. 2014). DOI: 10.48550/arxiv.1412.6980. URL: http://arxiv.org/abs/1412.6980.

[126] Pytorch. “Indexing into tensor order of magnitude slower than numpy”. In: (2019). URL: https://github.com/pytorch/pytorch/issues/29973.

[127] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: Differentiable Architecture Search”. In: (June 2018). DOI: 10.48550/arxiv.1806.09055. URL: http://arxiv.org/abs/1806.09055.

[128] Bichen Wu et al. “FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search”. In: (Dec. 2018). DOI: 10.48550/arxiv.1812.03443. URL: https://arxiv.org/abs/1812.03443.

[129] Han Cai, Ligeng Zhu, and Song Han. "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware". In: (Dec. 2018). DOI: 10.48550/arxiv.1812.00332. URL: http://arxiv.org/abs/1812.00332.

[130] Han Cai et al. "Once-for-All: Train One Network and Specialize it for Efficient Deployment". In: (Aug. 2019). DOI: 10.48550/arxiv.1908.09791. URL: https://arxiv.org/abs/1908.09791.

[131] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey". In: (Aug. 2018). DOI: 10.48550/arxiv.1808.05377. URL: http://arxiv.org/abs/1808.05377.

[132] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning". In: (Nov. 2016). DOI: 10.48550/arxiv.1611.01578. URL: http://arxiv.org/abs/1611.01578.

[133] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: (Nov. 2015). DOI: 10.48550/arxiv.1511.00363. URL: http://arxiv.org/abs/1511.00363.

[134] Md Mohiuddin Abdul Qudar and Vijay Mago. "A Survey on Language Models". In: *Association for Computing Machinery* 1.September (2020). ISSN: 16130073.

[135] Minh-thang Luong and Christopher D. Manning. "Stanford Neural Machine Translation Systems for Spoken Language Domains". In: *Iwslt-2015* (2015).

[136] Jonathan Frankle et al. *Stabilizing the Lottery Ticket Hypothesis*. 2019.

[137] Ari S Morcos et al. *One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers*. 2019.

[138] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *7th International Conference on Learning Representations, ICLR 2019*. 2019.

[139] Shaokai Ye et al. *Adversarial Robustness vs Model Compression, or Both?* 2019.

[140] Preetum Nakkiran. *Adversarial Robustness May Be at Odds With Simplicity*. 2019.

[141] Ludwig Schmidt et al. "Adversarially Robust Generalization Requires More Data". In: (2018).

[142] Florian Tramer et al. *On Adaptive Attacks to Adversarial Example Defenses*. 2020.

[143] Eric Wong, Leslie Rice, and J Zico Kolter. *Fast is better than free: Revisiting adversarial training*. 2020.

[144] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: (June 2017). DOI: 10.48550/arxiv.1706.06083. URL: https://arxiv.org/abs/1706.06083.

[145] Scott Gray, Alec Radford, and Diederik P Kingma. "GPU Kernels for Block-Sparse Weights". In: *OpenAI '17* 1 (2017). ISSN: 0270-6474.

[146] Trevor Gale et al. "Sparse gpu kernels for deep learning". In: *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*. Vol. 2020-Novem. 2020. DOI: 10.1109/SC41405.2020.00021.

[147] Ilya Tolstikhin et al. "MLP-Mixer: An all-MLP Architecture for Vision". In: (May 2021). DOI: 10.48550/arxiv.2105.01601. URL: http://arxiv.org/abs/2105.01601.

[148] Asher Trockman and J. Zico Kolter. "Patches Are All You Need?" In: (Jan. 2022). DOI: 10.48550/arxiv.2201.09792. URL: http://arxiv.org/abs/2201.09792.

[149] Christian Szegedy et al. "Intriguing properties of neural networks". In: *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199.

[150] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*. 2015. URL: http://arxiv.org/abs/1412.6572.

[151] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *CoRR* abs/1706.0 (2017). URL: http://arxiv.org/abs/1706.06083.

[152] Nicholas Carlini and David A Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *CoRR* abs/1608.0 (2016). URL: http://arxiv.org/abs/1608.04644.

[153] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Training Pruned Neural Networks". In: *CoRR* abs/1803.0 (2018). URL: http://arxiv.org/abs/1803.03635.

[154] Ravi S. Raju and Mikko Lipasti. "BlurNet: Defense by Filtering the Feature Maps". In: *Proceedings - 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2020*. 2020. ISBN: 9781728172637. DOI: 10.1109/DSN-W50199.2020.00016.

[155] Dimitris Tsipras et al. *Robustness May Be at Odds with Accuracy*. 2018.

[156] Marc Khoury and Dylan Hadfield-Menell. *On the Geometry of Adversarial Examples*. 2018.

[157] Chris Olah et al. "Zoom In: An Introduction to Circuits". In: *Distill* 5.3 (2020). DOI: 10.23915/distill.00024.001.

[158] Jürgen Schmidhuber. "Learning Factorial Codes by Predictability Minimization". In: *Neural Computation* 4.6 (1992). ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.6.863.

[159] Jürgen Schmidhuber. "Unsupervised Minimax: Adversarial Curiosity, Generative Adversarial Networks, and Predictability Minimization". In: *Neural Networks* 127 (2020). ISSN: 18792782.