Neural Networks Model Compression
The Static, the Dynamic and the Shallow

by

Sara Elkerdawy

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

Doctor of Philosophy

Department of Computing Science
University of Alberta

Abstract

Deep neural networks (DNN) have emerged as the state-of-the-art method in several research areas. DNN is yet to fully permeate resource-constrained computing platforms, such as mobile phones. Accurate DNN models being deeper and wider take considerable memory and time to execute on small devices posing challenges to many significant real-time applications, e.g., robotics and augmented reality applications. Considerations for memory and power consumption are as important for low-end devices as they are for cloud-based with multiple graphical processing units (GPUs). In cloud-based solutions, factors such as performance-per-watt, performance-per-dollar, and throughput are important. Recently, different techniques were proposed to tackle the computational and memory issues inherent in DNN. We focus on neural network model pruning and distillation for inference and training acceleration respectively.

First, early work in model pruning often relied on performing sensitivity analysis before pruning to set the pruning ratio per layer. This process is computationally expensive and hinders scalability for deeper, larger, and more connectivity complex models. We propose to train a binary mask for each convolutional filter that acts as a learnable pruning gate. In training, we encourage smaller models by inducing sparsity by minimizing the ℓ_1-norm of the masks. The task and pruning loss are trained jointly to allow for end-to-end fine-tuning and pruning.

Second, we present a layer pruning framework for hardware-friendly pruned models optimized for latency reduction. Our layer pruning framework is a twofold contribution. One, we present a one-shot accuracy approximation by imprinting for layer ranking. We rank layers based on the difference between their approximated accuracy

and that of the previous layer. Second, we adopt statistical criteria from filter pruning literature for layer ranking and examined both iterative filter pruning and layer pruning training paradigms under similar importance criteria in terms of accuracy and latency reduction.

Third, we propose a dynamic filter pruning inference method to tackle diminishing accuracy gain from adding more neurons. Motivated by the popular saying in neuroscience: "neurons that fire together wire together", we propose to equip each convolution layer with a binary mask predictor that selects a handful of filters to process in the next layer given the input feature maps. We pose the problem as a supervised binary classification problem. Each mask predictor module is trained to estimate the log-likelihood for each filter in the next layer to belong to the top-k activated filters.

Finally, we propose a distillation pipeline to accelerate the training of vision transformers. We adopt 1) self-distillation loss, and 2) query efficient teacher-study distillation loss. In self-distillation training, early layers mimic the output of the final layer within the same model. This achieves 2.8x speedup in comparison to teacher-student distillation with matched accuracy in many cases. We also propose a simple yet effective query-efficient distillation in case a trained teacher is available to further boost the accuracy. We query the teacher model (CNN) only when the student (transformer) fails to predict the correct output. This simple criterion not only saves computational resources but also achieves higher accuracy than a full query teacher-student.

Preface

This thesis is an original work by Sara Elkerdawy. The research project, of which this thesis is a part, was funded by Huawei's Toronto Heterogenous Compiler Lab.

Acknowledgments

"Whoever has not thanked people, has not thanked Allah."— The Prophet Muhammad, peace and blessings be upon him.

I would like to thank my supervisor committee Nilanjan Ray, Hong Zhang, and Martha White for their support throughout this tough journey. I would also like to thank my examining committee for their constructive feedback and insightful discussion. I would not have been able to finish the thesis if not for Mostafa Elhoushi whose feedback and broad knowledge guided me in establishing my work. He was a great mentor and collaborator that I truly appreciate.

The kindred people who received my constant rants and temper with tenderness and great love are my Mom, aunt, sister, brother, and their little rascals (a.k.a nieces and nephews) who were able to make me laugh which is a tough task. I also want to thank my comfort zone friends Omnia ElSaadany, Sara Nada, and Amany Hisham.

Finally, to my special person who would have loved seeing me moving forward and instilled loving science in me, to my Dad, may he rest in peace.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Convolutional Neural Networks (CNN) have become the state-of-the-art in various computer vision tasks, e.g., image classification [1], object detection [2], depth estimation [3]. These CNN models are designed with deeper [4] and wider [5] convolutional layers with a large number of parameters and convolutional operations. These architectures hinder deployment on low-power devices, e.g, phones, robots, wearable devices as well as real-time critical applications, such as autonomous driving. As a result, computationally efficient models are becoming increasingly important and multiple paradigms have been proposed to minimize the complexity of CNNs.

One paradigm is to manually design networks with a small footprint from the start such as [6–10]. This direction does not only require expert knowledge and multiple trials (*e.g.,* up to 1000 neural architectures explored manually [11]), but also does not benefit from available, pre-trained large models. Quantization [12, 13] and distillation [14, 15] are two other techniques, which utilize the pre-trained models to obtain good quality light-weight models. Quantization reduces the bit-width of parameters and feature maps, and thus decreases memory footprint, but requires specialized hardware instructions to achieve latency reduction. While distillation trains a pre-defined smaller model (student) with guidance from a larger pre-trained model (teacher) [14]. Finally, model pruning aims to automatically remove the least important filters (or weights) to reduce the number of parameters or FLOPs (*i.e.,*

indirect measures). We present a thorough investigation of the challenges, myths, and solutions to model compression. Throughout the thesis, we focus mainly on model pruning and explore deeply its challenges and solutions. Then we touch on distillation and quantization for further field coverage.

1.1 Motivation

Electronics and computing allowed for unprecedented complex services in this age of the Internet of Things (IoT). Unfortunately, these advanced services run with computing-intensive machine learning methods. This comes of significant cost to the environment to the point where training an AI model can emit as much carbon as five cars in their lifetimes [16].

Convolutional Neural Networks pruning have achieved outstanding performance on different tasks such as image classification [17–19] and object detection [20]. A commonly reported metric in literature to assess the pruned model quality is the floating-point operations (FLOPs). However, prior work in model pruning [18, 21, 22] showed that neither number of pruned parameters nor FLOPs reduction directly correlate with latency (*i.e.,* a direct measure). Latency reduction, in that case, depends on various aspects, such as the number of filters per layer (signature) and the deployment device. Most GPU programming tools require careful compute kernels[1] tuning for different matrices shapes (*e.g.,* convolution weights) [23, 24]. These aspects introduce non-linearity in modeling latency with respect to the number of filters per layer. Recognizing the limitations in terms of latency or energy by simply pruning away filters, recent works [18, 19, 21] proposed optimizing directly over these direct measures. These methods require per hardware and NN architecture latency measurements collection to create lookup tables or latency prediction models which can be time-intensive. In addition, the filter pruned methods are bounded by the model's

[1] A compute kernel refers to a function such as convolution operation that runs on a high throughput accelerator such as GPU

depth and can only reach a limited goal for latency consumption. On the other hand, filter pruning has finer search space in terms of memory consumption than layer pruning. This fine gradual decrease in memory consumption allows for a more articulated setup in terms of memory. Memory consumption is directly related to FLOPs and the number of parameters metrics, thus by reducing these indirect measures, we are gaining benefits in terms of memory.

Besides targeted resources and metrics, a recent direction in pruning tackles the problem from the data perspective. Large models are needed for a small portion of the data (hard samples) while we can most of the time perform fairly well using a lightweight model. For example, MobileNetv1 [7] requires almost double the computations than MobileNetv1_75 with only 2.2% accuracy gain. This is known as the diminishing returns *i.e.,*high computational needs to eke out small additional gains. This motivates the concept of dynamic pruning, where sub-networks from the large model are processed depending on the input sample [25–27]. The current dynamic pruning methods learn the routing in an unsupervised way and lack direct relation between computation saving and hyperparameter tuning.

1.2 Contributions

Motivated by these points, we propose multiple approaches to tackle each limitation and leverage strengths from a different perspective. The contributions along with the thesis organization are presented as follows:

- **Chapter 2** presents background and related work.
- **Chapter 3** proposes a **joint end-to-end filter pruning** for FLOPs and parameters reduction. The method tackles the issue with early-work literature on pre-defining a pruning ratio per layer which hindered scalability to very deep networks. We present an end-to-end joint training pruning by learning pruning

binary masks per layer. Our method imposes a sparsity regularization on the learnable binary masks along with the task loss. We showed how pruning benefits training small models compared to training from scratch, especially with limited data.

- **Chapter 4** focuses on latency reduction by noting that filter pruning methods achieve limited speedup gain. We explore higher pruning granularity through layer pruning resulting in depth shrinking. We introduce a novel **one-shot layer-wise accuracy approximation criterion for layer pruning**. Motivated by imprinting in few-shot literature, we equip each layer with a proxy classifier to rank layers by their approximate gain in accuracy. Our criterion is one-shot, unlike the iterative criterion evaluation adopted in filter pruning. We show that our layer pruned models achieve much better latency reduction than the state-of-the-art filter pruning methods.

- **Chapter 5** further extends our evaluation to go beyond accuracy approximation by imprinting as a layer ranking method. We present **LayerPrune** framework in which we compare filter and layer pruning paradigms under different statistical criteria, batch sizes, and hardware platforms. We demonstrate the effectiveness of layer pruned models as hardware-agnostic models. That means reaching computational budgets such as latency without tailoring to different inference setups. Budget constrained filter pruning requires re-training with a hardware-aware optimization jointly with the task to reach a budget.

- **Chapter 6** highlights the fact that small models perform fairly well in most cases as presented in our previous works. We tackle in this chapter dynamic pruning in which different sub-networks are routed in inference time per input sample. We present a **novel formulation for dynamic model pruning that poses the pruning decision as a supervised binary classification problem**. Similar to other dynamic pruning methods, we equip a cheap decision

head to the original convolutional layer. However, we propose to train the decision heads in a self-supervised paradigm instead of adding a regularization term. Decision heads predict the most likely to be highly activated filters given the layer's input activation. The masks are trained using a binary cross-entropy loss decoupled from the task loss to remove loss interference.

- Chapter 7, unlike previous chapters, the goal is to reduce the model's computation at inference time, this chapter focuses on training acceleration. In specifics, we focus on training acceleration for knowledge distillation training paradigm on vision transformers as a case study. We propose a **self-distillation training pipeline for efficient vision transformer**. We also show simple query-efficient distillation in the case of pre-trained teacher availability. Auxiliary lightweight classifiers are inserted at different depths in the transformer backbone. These classifiers are trained using a weighted sum between task loss and distillation loss. The distillation loss enforces these classifiers to mimic the output of the final classifier in a self-supervision way. We show that self-distillation achieved 2.8x speedup over teacher-student distillation while achieving on-bar accuracy in most cases.

1.3 Published Papers

Some extracts from this thesis appear in the following authored publications and preprints.

- Elkerdawy, Sara, et al. "Fire Together Wire Together: A Dynamic Pruning Approach with Self-Supervised Mask Prediction." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (**CVPR 2022**).

- Liu, Hongyang, et al. "Layer Importance Estimation with Imprinting for Neural Network Quantization." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (**CVPRW 2021**).

- Elkerdawy, Sara, et al. "To filter prune, or to layer prune, that is the question." Proceedings of the Asian Conference on Computer Vision (**ACCV 2020**).

- Elkerdawy, Sara, et al. "One-shot layer-wise accuracy approximation for layer pruning." 2020 IEEE International Conference on Image Processing (**ICIP 2020**).

- Elkerdawy, Sara, Hong Zhang, and Nilanjan Ray. "Lightweight monocular depth estimation model by joint end-to-end filter pruning." 2019 IEEE International Conference on Image Processing (**ICIP 2019**).

Chapter 2

Background and Related Work

2.1 Background

There is a variety of methodologies to tackle the efficiency of processing Deep Neural Networks (DNN). We first draw the big picture of the field for completeness and then dive more into details for model pruning in specific as the main area of interest. We also touch on knowledge distillation training acceleration. Figure 2.1 shows the broad categorization to address CNN efficiency.

2.1.1 Neural Network Architecture Design

New architectures have gone a long way from simple sequential AlexNet and VGG to more sophisticated structures with smaller components and more complex connections such as ResNet, ShuffleNet, MobileNet, and Squeeze-and-Excitation Networks. This direction does not only require expert knowledge and multiple trials, (*e.g.,* up to 1000 neural architectures explored manually [11]), but also does not benefit from available, pre-trained large models. Another point that is worth mentioning is the lack of efficiency gain metric in most of these works. The reported metric is mainly in FLOPs which is not necessarily an indication for efficient processing. There is an increasing effort to benchmark architectures across different hardware targets such as DeepBench [29], MLPerf [30] and DAWNBench [31]. The key point here is that different inference setups such as hardware targets, batch size, and backend libraries

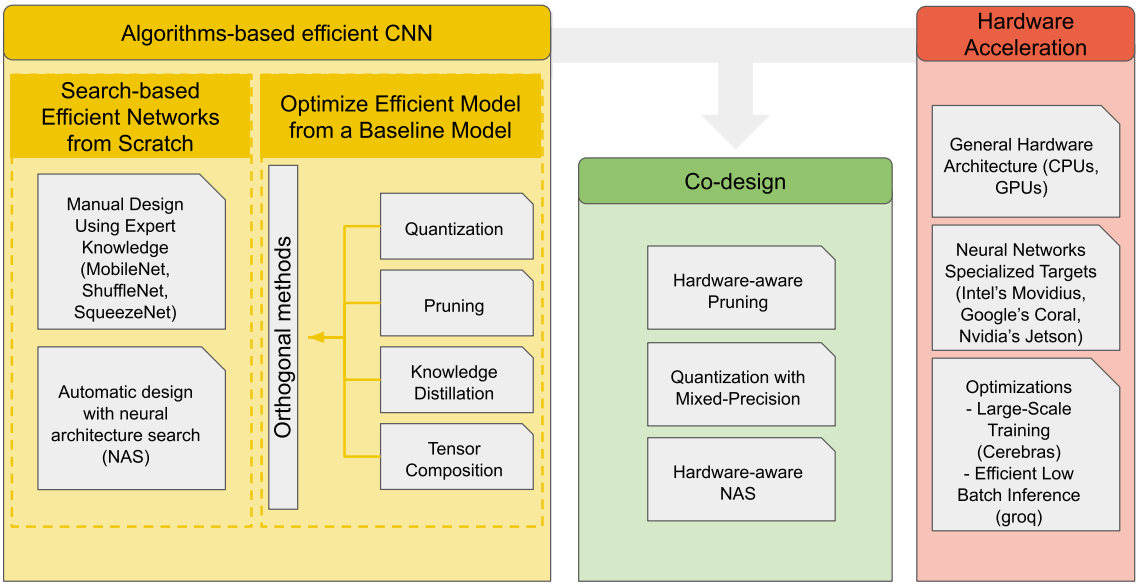


Figure 2.1: Overview of efficient CNN categories. Inspired from [28]

require different architecture. For example, a matrix multiplication based on the size of the matrices being multiplied and the kernel implementation may be compute-bound based on processing units available in hardware, memory-bound based on cache levels and memory available, or streaming occupancy-bound based on level of parallelism. Each architecture uses these operations with different parameters and thus the optimization space for hardware and software targeting deep learning is large and underspecified. To tackle some of these issues, Neural Architecture Search (NAS) for automatic design is a growing research area. Such works [32–35] involve defining search space for different components and connections inspired from manual designs so far. The cost function can be modeled for metrics of interest such as FLOPs, latency, memory, energy, etc. This direction certainly reduced human intervention and added flexibility on modeling cost functions to accommodate different needs. However, the computing power needed in training is tremendous due to the large search space.

2.1.2 Quantization

Model quantization focuses on reducing the bit-width of the operations, hence, reducing memory consumption and latency of operations. Researchers showed that using half-precision is sufficient in training deep neural networks and is resilient to the errors in the back-propagation phase [36–39]. There are mainly two paradigms for model quantization: 1) Post-training quantization, and 2) Training-aware quantization. The former quantizes a pre-trained 32-bit floating-point model, thus enabling available off-the-shelf model exploitation. The latter performs quantization within training for gradual loss optimization. A limitation of quantization is how tightly dependable it is on hardware vendors and back-end libraries. The hardware target needs to support processing low-bit precision through instruction sets and the ability to implement back-end kernels using these instructions. Many works in the literature show only the theoretical gain on mixed-precision or lower than 8-bit quantization. Advances in quantization and hardware design go hand in hand which can be limiting deployment with currently available hardware platforms.

2.1.3 Design of Hardware and Accelerators

As briefly mentioned in the quantization section, the design of hardware plays a big role in exploiting and enabling performance gain of theoretical literature work. Features that target DNN processing are being developed in new hardware platforms. For instance, Intel's Knights Mill [40] processor introduced a special number of instructions such as fused multiply-accumulate operations heavily used in DNN. The Nvidia PASCAL GP100 GPU [41] features a 16-bit floating-point (FP16) arithmetic support to perform two FP16 operations on a single-precision core for faster computation. In addition to special instructions, new accelerators such as Eyeriss [42] and simulators such as [43–45] exploit efficient processing of sparsity and dynamic precision requirement. We also note the constant supply of specialized neural compute sticks such as Intel's Movidius, Google's Coral, and Nvidia's Jetson line of boards to

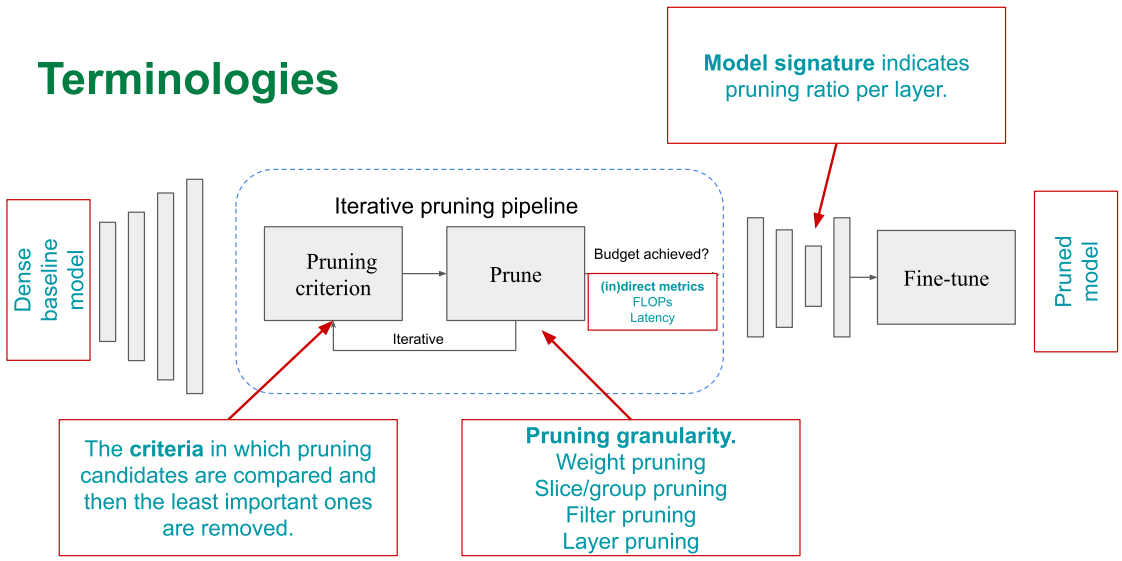


Figure 2.2: Neural network pruning process.

bring accelerated AI on the edge. Systems for large-scale efficient training are also increasingly built such as Cerebras [46].

2.1.4 Knowledge Distillation

Knowledge distillation is a training pipeline that poses a loss in training a small model (student) from the knowledge of a pre-trained large model (teacher). Different approaches to knowledge transfer are proposed such as softmax probability approximation, feature maps mimicking, and statistical properties transfer. Distillation showed effectiveness in transfer between similarly designed architecture (e.g ResNet101 as teacher and ResNet18 as a student). While distillation is a beneficial training pipeline, it increases training time in comparison to training students alone. The benefit of distillation is the usage of the available trained model zoo and no additional cost in inference as there is no special operations or change in architecture.

2.1.5 Model Pruning

Recent research showed that model over-parameterization usually makes training easier [47, 48]; thus many of the weights in DNN models exhibit high redundancy and

can be removed in inference without reducing accuracy. There are different levels of sparsity (*granularity*) such as weight sparsity, filter pruning, and layer pruning. Model pruning algorithms tend to follow the same process which is shown in Figure 2.2. First, *weight initialization* stage which can start from scratch (*i.e.,* random weight or trained for a few epochs) or initialize from a pre-trained dense model. Then, based on pruning granularity, pruning candidates are sorted based on an *importance criteria* to evaluate their significance to the task. Next, the pruning ratio per layer (referred to as *model signature*) is determined to remove the least important weights/filters. Signatures in early work of model pruning were based on a pre-processing step called sensitivity analysis. It involves evaluating the sensitivity of each layer independently to decide the pruning ratio per layer. This analysis is computationally expensive to conduct and becomes even less feasible for deeper models. Later work proposes global importance criteria that can evaluate weights and filters across different layers. However, results using such approaches can be sensitive to different CNN architecture and statistical properties per layer. Finally, a fine-tuning step is performed on the newly pruned network to recover accuracy from pruning. A recent direction branched from model pruning is called dynamic inference. The previously explained pruning pipeline produces a static pruned model. Static models treat all input samples the same in terms of computations. Yet, different input samples might require different computational needs. For instance, a small faraway object requires more complex feature extraction (e.g deeper network evaluation) than a large clear near object. Also, samples that belong to vehicles activate a different group of filters than that of animals.

2.1.6 Cloud versus Embedded

The various stages in DNN processing (training versus inference) or applications (*e.g.,*robotics, surveillance, or recommendation systems) have different computational needs. Training is typically done in the cloud where offline processing of large available

datasets is performed. Inference, on the other hand, can happen in a wider spectrum of platforms ranging from cloud to low-power edge devices. In many applications, it is desirable to perform locally on the edge to reduce communication costs from-to the cloud. For instance, measuring wait time in stores or granting access to a gated parking lot. In other applications, such as robotics and autonomous driving, local processing is desirable as the latency and security risks of relying on cloud processing are too high. Products such as Apple Siri and Amazon Alexa voice services are done on the cloud, yet it is desirable to perform locally on the device to increase security. Some work divides the processing of a neural network between cloud and edge such as Neurosurgeon [49] or split the problem formulation into a Two-Phase Predictions format such as adopted in Google Home [50], which are activated by a wake word and can then respond to commands. This problem can be split into an initial cheap model that listens for a wake word, and a more complex model for the more complex speech commands.

2.1.7 Key Metrics

In this section, we discuss the evaluation metrics usually reported in papers and operational metrics that are of interest at deployment time and the gap between these two sets of metrics.

Accuracy Accuracy is used to indicate the quality of the output for a given task, thus it differs based on the task. For instance, in image classification, papers report top-1 and top-5 accuracy which indicate the number of samples that are correctly classified within the first or five top predictions, respectively. In object detection, mAP (mean average precision) is reported to assess the quality of the bounding box prediction.

FLOPs and Parameters The quality of the computationally efficient model is measured by the number of parameters and number of floating-point operations

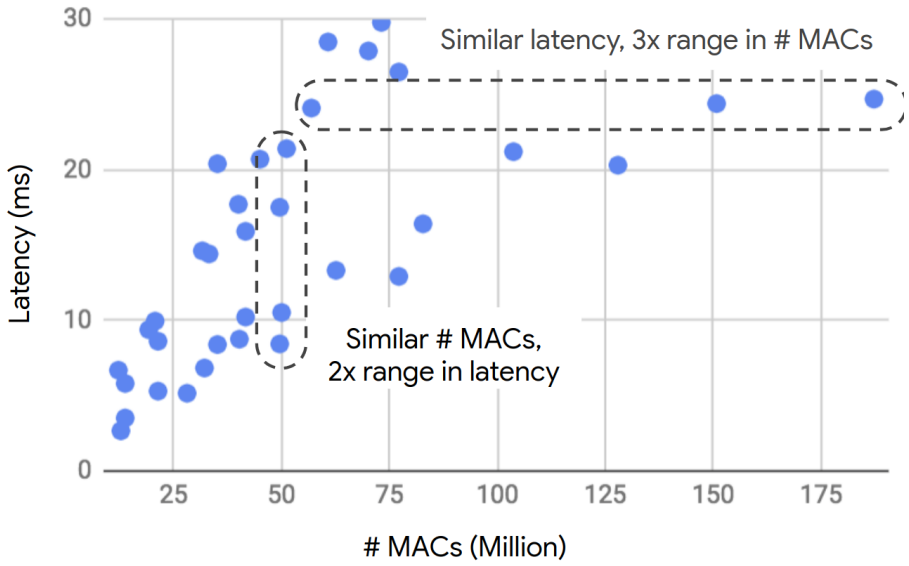


Figure 2.3: The number of MAC operations in various DNN models versus latency measured on Pixel phone. (Figure from [51])

FLOPs or Multiply-and-Accumulate (MACs) per image. These metrics are referred to as indirect metrics. The number of FLOPs and parameters do not encapsulate many computational concerns such as memory transfers, number of processing units, caching levels, and back-end variations. Figure 2.3 shows that the number of MACs is a poor proxy for latency. On the other hand, FLOPs are still considered a good estimation of memory consumption as the metric is parameterized in both the weights and size of feature maps.

Throughput and Latency Throughput and latency are often assumed to be directly derivable from each other. However, they are indicating different deployment needs and measures. Throughput is used to indicate the amount of data or number of clients that can be processed/served in a given time frame. Latency, however, indicates the time needed to process input from time of arrival to results generation. An example to show the distinct difference is for instance when batching input (*i.e.*,multiple data points are batched and processed together) to better utilize resources. At 30 frames per second input streaming and batch size of 128 results in a

latency delay for some input points to 4.2 seconds. This trade-off between low latency and high throughput with maximally utilized resources can be challenging. In some of our work, we focus on latency as a direct metric for online streaming setup rather than throughput which can be more beneficial in cloud-based deployments. Yet, we wanted to display the differences so give a more thorough big picture of different deployment metrics.

Energy Efficiency Energy efficiency is a direct measurement indicating the number of data points that can be processed in a given energy unit. It is usually reported as the number of operations per joule. Unlike latency and throughput, energy measurement requires power consumption readings through voltage monitor tools. This restricts accessibility and widens the gap between deployment and metrics reporting.

2.2 Related Work

We dive more into model pruning literature as the main focus of our work and touch on training acceleration for knowledge distillation.

We divide existing pruning methods into four categories: weight pruning, hardware-agnostic filter pruning, hardware-aware filter pruning and layer pruning.

2.2.1 Weight pruning

An early major category in pruning is individual weight pruning (unstructured pruning). Weight pruning methods leverage the fact that some weights have minimal effect on the task accuracy and thus can be zeroed out. In [53], weights with small magnitude are removed, and in [54], quantization is further applied to achieve more model compression. Another data-free pruning is [55] where neurons are removed iteratively from fully connected layers. L_0-regularization based method [56] is proposed to encourage network sparsity in training. Finally, in the lottery ticket hypothesis

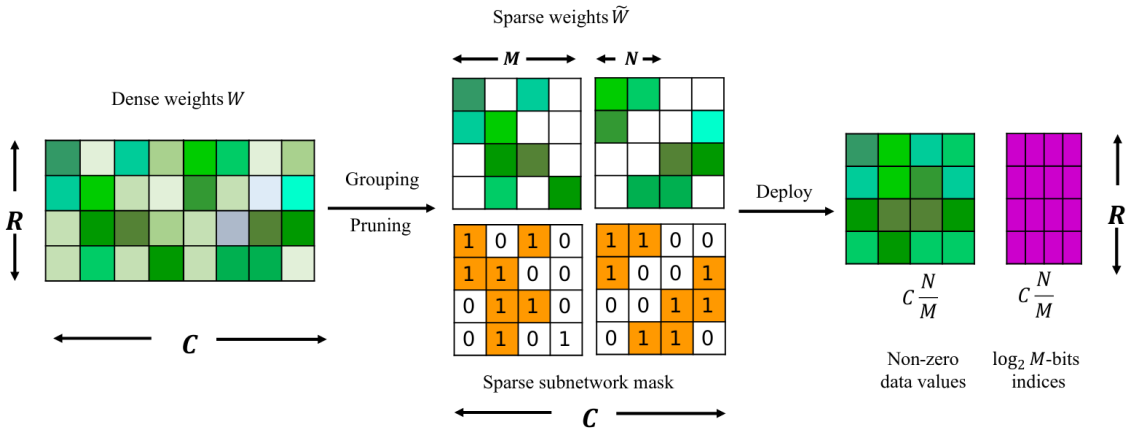


Figure 2.4: Illustration of achieving N :M structure sparsity (Left) In a weight matrix of 2:4 sparse neural network, whose shape is R x C (*e.g.*,R = output channels and C = input channels in a linear layer), at least two entries would be zero in each group of 4 consecutive weights. (Middle & Right) The process that the original matrix is compressed, which enables processing of the matrix to be further accelerated by designated processing units (*e.g.*,Nvidia A100). Figure from [52].

[57], the authors propose a method of finding winning tickets which are sub-networks from random initialization that achieve higher accuracy than the dense model. The limitation of the unstructured weight pruning is that dedicated hardware and libraries [58] are needed to achieve speedup from the compression. Even with dedicated sparse compute, unstructured sparsity fails to utilize vector processing architectures. This reduces latency with random memory access. To add more structure to such fine-grained sparsity, a recent Nvidia Ampere architecture [52] found in A100 GPUs are equipped with Sparse Tensor Cores to accelerate 2:4 (Figure 2.4 structured sparsity.

2.2.2 Hardware-agnostic filter pruning

Methods in this category (also known as structured pruning) aim to reduce the footprint of a model by pruning filters without any knowledge of the inference resource consumption. Examples of these are [60–64], which focus on removing the least important filters and obtaining a slimmer model. Earlier filter-pruning methods [62, 64] required layer-wise sensitivity analysis to generate the signature (*i.e.*, number

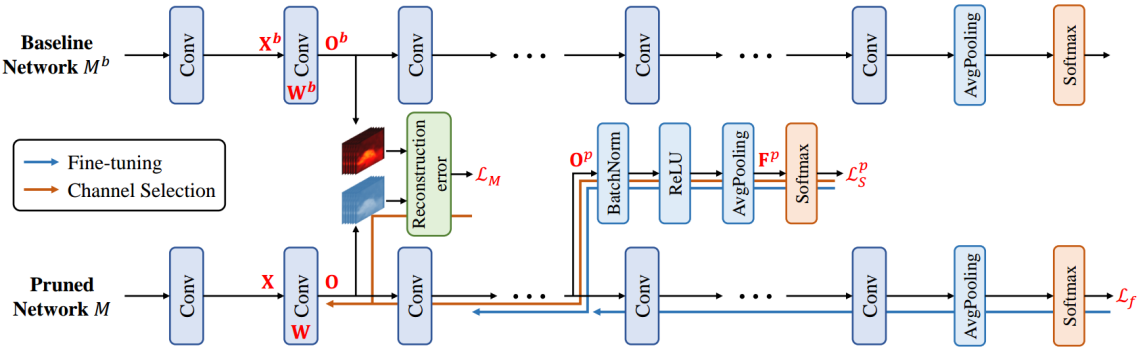


Figure 2.5: Illustration of discrimination-aware channel pruning. Here, L_S^p denotes the discrimination-aware loss (*e.g.*, cross-entropy loss) in the L_p-th layer, L_M denotes the reconstruction loss, and L_f denotes the final loss. Figure from [59].

of filters per layer) as a prior and remove filters based on a filter criterion. The sensitivity analysis is computationally expensive to conduct and becomes even less feasible for deeper models. Recent methods [60, 61, 63] learn a global importance measure removing the need for sensitivity analysis. Molchanov et al. [60] propose a Taylor approximation of the network's weights where the filter's gradients and norm are used to approximate its global importance score. Liu et al. [61] and Wen et al. [63] propose sparsity loss for training along with the classification's cross-entropy loss. Filters with importance less than a threshold are removed and the pruned model is finally fine-tuned. Zhao et al. [65] introduce channel saliency that is parameterized as Gaussian distribution and optimized in the training process. After training, channels with a small mean and variance are pruned. DCP [66] utilizes multiple losses such as discrimination loss, feature reconstruction loss, and task loss as shown in Figure 2.5. In general, methods with sparsity loss lack a simple approach to respect a resource consumption target and require hyperparameter tuning to balance different losses.

2.2.3 Hardware-aware filter pruning

To respect a resource consumption budget, recent works [18, 19, 67, 68] have been proposed to take into consideration a resource target within the optimization pro-

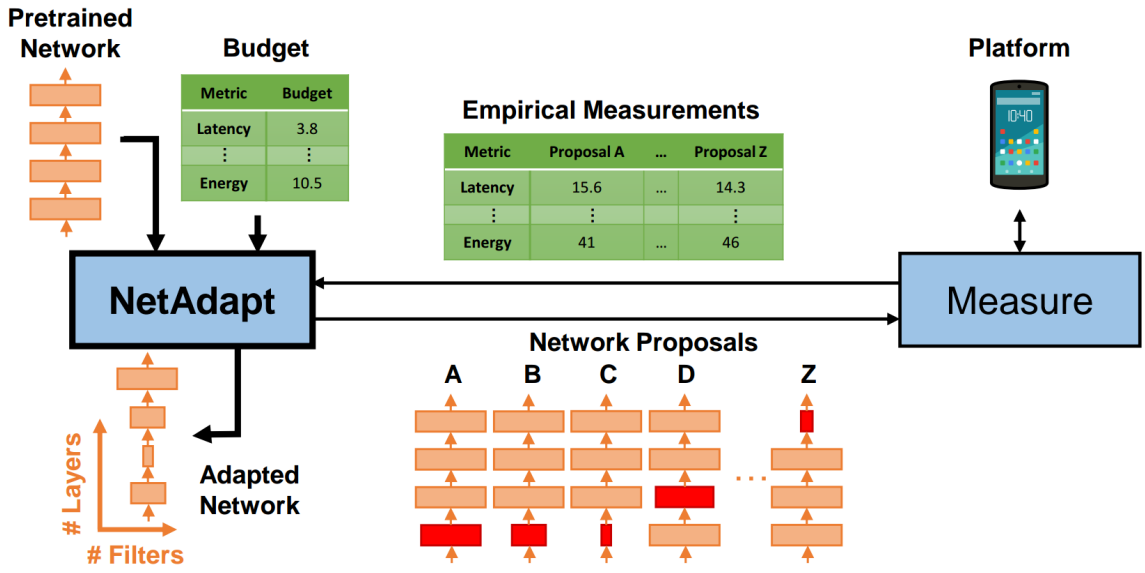


Figure 2.6: NetAdapt automatically adapts a pretrained network to a mobile platform given a resource budget. Figure from [18].

cess. NetAdapt [18] prunes a model to meet a target budget using heuristic greedy search. A lookup table is built for latency prediction and then multiple candidates are generated at each pruning iteration by pruning a *ratio* of filters from each layer independently as shown in Figure 2.6. The candidate with the highest accuracy is then selected and the process continues to the next pruning iteration with a progressively increasing *ratio*. On the other hand, AMC [68] and ECC [19] propose an end-to-end constrained pruning. AMC utilizes reinforcement learning to select a model's signature by trial and error. ECC simplifies the latency reduction model as a bilinear per-layer model. The training utilizes the alternating direction method of multiplier (ADMM) to perform constrained optimization. ADMM alternates between network weight optimization and learnable dual variables that control layer-wise pruning ratio. Although these methods incorporate resource consumption as a constraint in the training process, the range of attainable budgets is limited by the depth of the model. In addition, generating data measurements to model resource consumption per hardware and architecture can be expensive, especially on low-end hardware platforms.

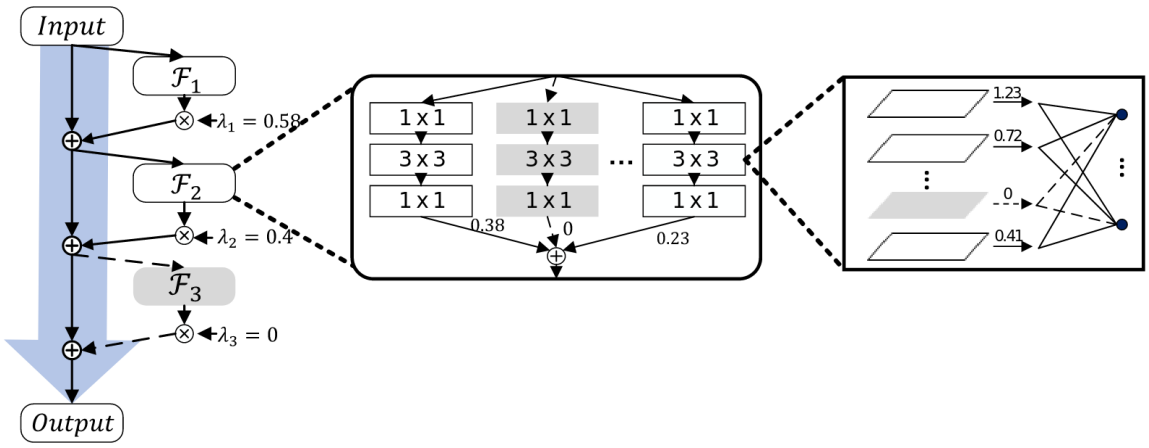


Figure 2.7: SSS architecture with different pruning granularity. Gray block, group and neuron mean they are inactive and can be pruned since their corresponding scaling factors are 0. Figure from [69].

2.2.4 Layer pruning

Unlike filter pruning, little attention is paid to shallower CNNs in the pruning literature. In SSS [69], the authors propose to train a scaling factor for structure selection such as neurons, blocks, and groups. A limitation of their method is the joint pruning and fine-tuning which limits the types of architectures to produce shallower models. Shallower models are only possible with architectures with residual connections as shown in Figure 2.7. That is to allow data flow in the optimization process with joint pruning and fine-tuning. Closest to our work for a general (*i.e.*,not constrained by architecture type) layer pruning approach is the work done by Chen et al. [70]. In their method, linear classifier probes are utilized and trained independently per layer for layer ranking. After the layer ranking learning stage, they prune the least important layers and fine-tune the shallower model. In our work, we propose an efficient accuracy approximation for layer pruning in one-shot (Sec 4). We also give a thorough comparison between filter and layer pruned models in terms of latency reduction under different inference setups (Sec 5).

2.2.5 Dynamic inference

Previous methods produce a static pruned model in which all input samples are processed with the same operations. In dynamic or conditional inference, samples are processed using different routes (i.e dynamic pruning) or using variant depth of the network (i.e early-exit). In Runtime Neural Pruning (RNP) [26], a decision unit is modeled as a global recurrent layer, which generates discrete actions corresponding to four preset channel selection groups. The group selection is trained with reinforcement learning. Similarly in BlockDrop [71], a policy network is trained to skip residual blocks in residual networks instead of only channels. D^2NN [72] defines a variant set of conditional branches in DNN, and uses Q-learning to train the branching policies. These methods train their policy functions by reinforcement learning which can be a non-trivial optimization task along with the CNN backbone. Feature Boosting and Suppression (FBS) [73] method generates continuous channel saliency. Wang et al. [74] proposed to obtain a discrete action from N learned group of channels sampled from Gumbel distribution. They adopt annealing temperature to stabilize training and introduce diversity in the learned routes. These dynamic inference methods require additional careful tuning to stabilize training either from policy gradients or learning diverse routes in an unsupervised way. Hence, we propose to formulate the channel selection as a supervised binary classification in which routes are learned and simply trained with SGD (Sec 6).

2.2.6 Knowledge Distillation Training Acceleration

Previous sections focus on generating a lightweight model for inference acceleration. Knowledge distillation (KD) is an approximation loss used in training a small model to improve its accuracy. However, a classical knowledge distillation requires a pre-trained teacher as a prior. We focus on accelerating the training part for the knowledge

distillation training paradigm. Recent works explore training acceleration by either 1) removing the dependence on a trained teacher, or 2) reducing query calls to the teacher while training the student. For example, [75] introduces an active mixup augmentation strategy that selects hard samples to query the teacher. This query efficient teacher-student distillation reduces the number of calls to the teacher during student training. In [76], the knowledge distillation operates in a few-shot training setup where the available training data is limited. While there is still an accuracy gap to reach the distillation accuracy with the full data, the speedup to using only a few images (*e.g.,*up to 10 images only) is promising. Another direction explores self-distillation [77] to reduce the overhead of training a teacher model and teacher's labels query in training the student. Auxiliary classifiers are inserted at different depths in the backbone to mimic the behavior of the final classifier. Knowledge distillation is also beneficial to vision transformers especially in training small and medium-sized datasets. DeiT [78] applies knowledge distillation [79] by adding a KD token distillation that matches the output of a CNN pre-trained teacher. DeiT queries each batch to both the CNN teacher and the transformer student to match their outputs; This can be memory-demanding and time-consuming. We explore the training acceleration, particularly for vision transformer as a use case.

Chapter 3

Joint End-to-End Filter Pruning

3.1 Motivation

Early work in model pruning often relied on performing sensitivity analysis before pruning to set the pruning ratio per layer. Sensitivity analysis involves iterating over all layers independently and tuning the pruning ratio for each layer such that we are able to rank layers by their sensitivity to pruning. This process is computationally expensive and hinders scalability for deeper, larger, and connectivity complex models. In this work, we propose to automatically generate a slimmer model from a pre-trained dense model in an end-to-end training pipeline. Our end-to-end training automatically selects the pruning ratio per layer which scales better to deep models such as in depth estimation vision tasks. We propose to train a binary mask for each convolutional filter that performs gating that allows filter contribution or not. In training, we encourage smaller models by inducing sparsity by minimizing the ℓ_1-norm of the masks. To take into account the task loss as well, the masks are trained with both ℓ_1 loss and the depth estimation loss. Closest to our proposed pruning method is [17] in the gating aspect, however, we must emphasize that our masks are learned jointly on the whole network. Learnable masks allow for removal to preset compression rate per layer or computing layerwise sensitivity analysis as in [80]. In [17], the authors set the compression rate for each layer and adopt layer by layer pruning where each prune is followed by a fine-tune. Both of these points obstruct

scalability for large datasets and models such as encoding-decoding (i.e hourglass) architectures with images as inputs and outputs. These hourglass models are twice the size of classification models. All of these issues motivated us to train an end-to-end joint pruning method that can be adopted in large-scale models and datasets suitable for depth estimation.

Key points: We propose an end-to-end solution to learn the pruning ratio and mask per layer jointly with the task training. This joint training mitigates the need for a pre-defined pruning ratio per layer. Hence, it removes the manual tuning by an expert and simplifies the scalability to other deep neural architectures.

3.2 Proposed Method

3.2.1 Preliminary

In this work, we showcase the pruning method for the vision task monocular depth estimation. Our baseline deep models are trained based on LRC [81] casting the problem as image reconstruction from stereo input pairs in an unsupervised setup. Figure 3.1 summarizes the detailed monocular depth estimation. Stereo input images I^l and I^r are utilized to encourage the reconstructed images $\hat{I}^l$ and $\hat{I}^r$ to appear similar to the corresponding training input. Dense disparity images d^l and d^r define the displacements between corresponding points. Multiple training losses are utilized in the depth estimation such as reconstruction error, left-right consistency loss, and smoothness loss. In our proposed pruning, we refer to all these losses inherent in the task as task loss. Detailed definitions of these losses are defined in the next section.

3.2.2 Joint Training Losses

In this section, we describe our end-to-end joint pruning method for monocular depth estimation. We base our solution on the unsupervised image reconstruction proposition by Godard et al. [81].

The pipeline contains two main losses: 1) Task loss, and 2) sparsity loss. Task loss

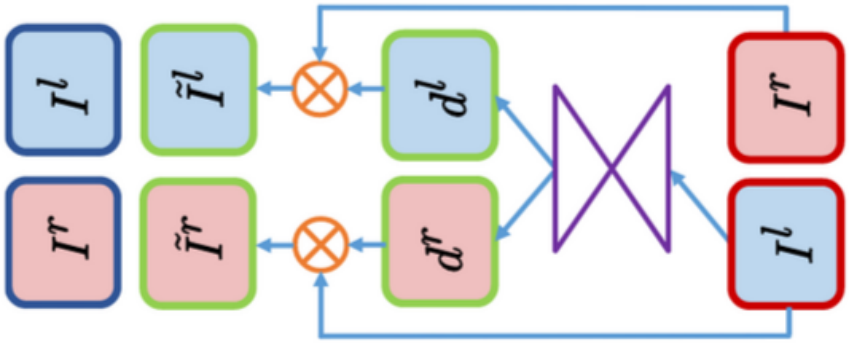


Figure 3.1: MonoDepth training. Image from [81].

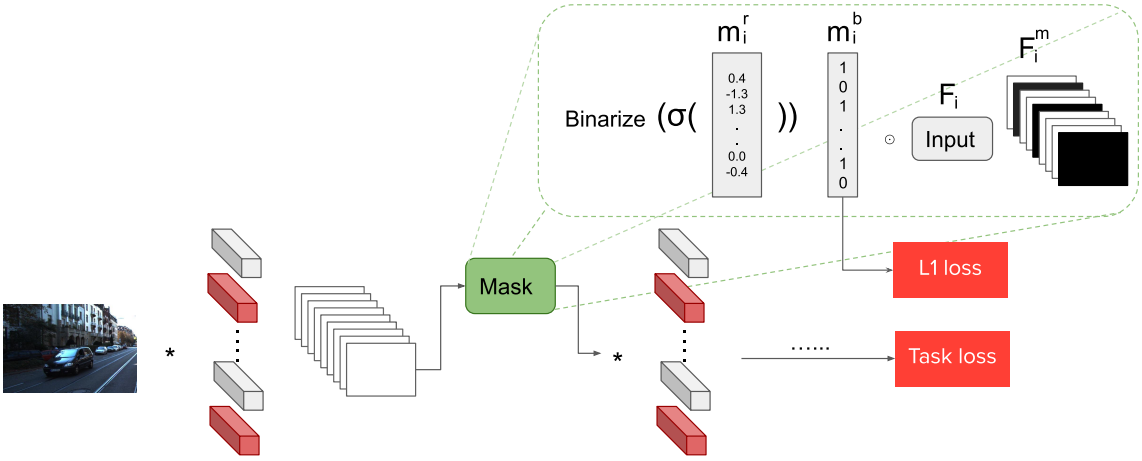


Figure 3.2: Proposed joint end-to-end pruning. Red and grey filters indicate pruned or kept respectively. A real-valued mask m_i^r is learned through STE [82] from its corresponding binary m_i^b estimation. The binary mask is multiplied by the input feature maps F_i to drop the corresponding filter contribution. The new masked feature maps F_i^m (e.g black features zeroed out) are the new input for the next layer. We apply sigmoid function σ on $m_{i,j}^r$ to limit the range of the real-values and simplify threshold selection in binarize function. ℓ_1 loss on all masks and task loss are jointly optimized.

includes all losses with image reconstruction, disparity smoothness, and lr-consistency. The sparsity loss is applied on the masks with ℓ_1-norm to encourage our model with fewer features.

Task loss . We train all the models with three weighted losses contributing to the final task loss as formulated in [81].

$$L_{task} = \alpha_{ap}(L^l_{ap}+L^r_{ap}) + \alpha_{ds}(L^l_{ds}+L^r_{ds}) + \alpha_{lr}(L^l_{lr}+L^r_{lr}) \tag{3.1}$$

Each loss term is calculated for both left and right images in the stereo input pair. The first term L_{ap} calculates the reconstruction loss between the original image and the warped image using SSIM [83] and $L1$ difference. The second term L_{ds} encourages disparity discontinuities only at the gradient δI. Finally, the left-right consistency term L_{lr} enforces coherence between predicted left disparity d^l and predicted right disparity d^r.

Mask sparsity loss . This loss term controls the model size. Before diving into the sparsity loss, we explain the masking formulation. First, we initialize real-valued mask $m^r_i \in \mathbb{R}^{n_i}$ for each layer i with n_i filters. A binary function is then applied on the real-valued masks to get m^b_i based on a threshold t (e.g $m^r_{i,j} \geqslant t$ outputs 1 and 0 otherwise). Finally, ℓ_1 is applied on all the masks to form a sparsity loss term:

$$L_{mask} = \frac{\sum_i^N \parallel m^b_i \parallel_1}{\sum_i^N n_i} \tag{3.2}$$

where N is the total number of layers in the network. Looking carefully at (3.2), as m^b_i vectors are binary, the loss term is calculating the ratio between the total number of filters in the new model and the original large model. Minimizing this loss is equivalent to maximizing the compression rate. Our total loss is then given by

$$L_{total} = L_{task} + L_{mask} \tag{3.3}$$

3.2.3 Forward and Backward Passes

Forward pass. Let $F_{i,j}$ be the j-th feature map of the i-th layer, the new feature map $F^m_{i,j}$ and binary mask $m^b_{i,j}$ are thus given by:

$$\begin{aligned} m^b_{i,j} &= Binarize(Sigmoid(m^r_{i,j}), 0.5) \\ F^m_{i,j,h,w} &= F_{i,j,h,w} \odot m^b_{i,j} \end{aligned} \tag{3.4}$$

We apply a sigmoid function on our real-valued masks to transfer the input into [0,1] range before passing through binarization. This simplifies the selection of threshold t as a sensible choice would be 0.5. Finally, $F^m_{i,j}$ is passed as the new $(i+1)$-th layer's input which corresponds to either $F_{i,j}$ or 0. Zeroing out a feature map $F_{i,j}$ simulates dropping the corresponding filter $f_{i,j}$. Figure 3.2 shows the masking block embedded within the network summarizing the forward pass.

Backward pass. In the backward pass, we update the convolutional kernels and $m^r_{i,j}$ for each layer. As $Binarize$ is a conditional non-differentiable function, we backpropagate the gradients to $m^r_{i,j}$ utilizing the straight-through-estimator (STE) proposed in [82]. They showed that we can approximate the gradient of a real-valued weight with the gradient of its discretization. Even though gradients calculated through such a function (e.g $Binarize$) are noisy, they serve as regularizers and are acceptable approximations of the true gradients to the real-valued masks. Using STE and from (3.4), we have the gradients as:

$$\begin{aligned} \delta m^b_{i,j} \triangleq \frac{\partial L_{total}}{\partial m^b_{i,j}} &= \sum_h^H \sum_w^W \delta F^m_{i,j,h,w} \cdot F_{i,j,h,w} \\ \delta m^r_{i,j} &= \delta m^b_{i,j} \cdot Sigmoid(m^r_{i,j}) \cdot (1 - Sigmoid(m^r_{i,j})) \end{aligned} \tag{3.5}$$

The double sum stems from the fact that $m^b_{i,j}$ is shared among all spatial locations in $F_{i,j}$.

3.3 Experiments and Analysis

Results are compared using different depth metrics commonly used for monocular depth estimation and adopted from [84] on KITTI dataset [85].

Eigen split. Table 3.1 shows evaluation on Eigen split with the other methods reporting on Eigen. Our smaller models achieve better accuracy than the supervised methods [84, 86] and unsupervised method [87]. Interestingly, the gap in accuracy (e.g 5th column) between our pruned model and PyD-Net differs based on the training data. As small models require a large amount of data (e.g CS+K) to achieve good results, our method on the other hand benefits from the pre-trained large model even when trained with KITTI dataset only. This shows the benefit of pruning rather than training from scratch, especially with limited training data.

	Ours		Lower is better			Higher is better			
Method	Supervised	Dataset	Abs Rel	RMS	RMS_{log}	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	Params
Eigen et al. [84]	Yes	K	0.203	6.307	0.282	0.702	0.890	0.958	54.2M
Liu et al. [86]	Yes	K	0.201	6.471	0.273	0.680	0.898	0.967	40.0M
Zhou et al. [87]	No	K	0.208	6.856	0.283	0.678	0.885	0.957	34.2M
LRC + VGG [81]	No	K	0.148	5.927	0.247	0.803	0.922	0.964	31.6M
VGG + L_{total}	No	K	0.1356	5.891	0.236	0.827	0.927	0.965	5.7M ↓ 81.8%
PyD-Net	No	K	0.163	6.253	0.262	0.759	0.911	0.961	1.9M
DORN ResNet101 [88]	Yes	ILSVRC+K	0.072	2.727	0.120	0.932	0.984	0.994	NA
Zhou et al. [87]	No	CS+K	0.198	6.565	0.275	0.718	0.901	0.960	34.2M
LRC + VGG [81]	No	CS+K	0.124	5.311	0.219	0.847	0.942	0.973	31.6
VGG + L_{task}	No	CS+K	0.124	5.280 ↓ 0.03	0.219	0.848	0.942	0.973	30.8M ↓ 2%
VGG + L_{total}	No	CS+K	0.1452 ↑ 0.02	5.835 ↑ 0.524	0.239	0.815	0.927	0.967	5.9M ↓ 81.1%
LRC + ResNet50 pp* [81]	No	CS+K	0.114	4.935	0.206	0.861	0.949	0.976	58.4M
PyD-Net [89]	No	CS+K	0.148	5.929	0.244	0.800	0.925	0.967	1.9M

Table 3.1: **Comparison on Eigen split**. In dataset, K indicates training on Kitti [85] and CS indicates Cityscapes [90]. Our models compress more than 70% the original model with small drop in accuracy. *pp post-processing done by [81] but requires two forward passes.

Qualitative results Figure 3.3 shows some qualitative comparison to LRC [81] and PyDNet [89]. Although our pruned model is 5x smaller than LRC, they still produce similar good quality smooth output. Our model benefits from the pre-trained VGG model to produce smooth output and not as noisy as the case with a similar small-sized model PyDNet. It is worth noting that small models (ours and PyDNet) better regularize scenes with fewer data in the training unlike LRC as shown in the third column. However, the pruned model shows an accuracy drop with small objects

Figure 3.3: Depth predictions on KITTI Eigen compared with LRC [81] 31.6M, ours VGG+L_{total} 5.9M, PyD-Net 1.9M [89] from top to bottom. Our pruned model produces good quality smooth output compared to PyD-Net but still with small accuracy drop (e.g pole in first column). Small models better regularize scenes with fewer data in the training (e.g a turn in third column)

(e.g poles).

3.4 Conclusion

We proposed a lightweight model for monocular depth estimation motivated by pruning literature. Our joint end-to-end pruning is scalable for deep models adopted in depth estimation. We learn binary masks within the network to drop filters jointly without pre-defined layer-wise compression rates. We showed how pruning benefits small model training compared to training from scratch, especially with limited data.

Chapter 4

Accuracy Approximation by Imprinting for Layer Pruning

4.1 Motivation

Pruned model quality is assessed based on the drop in accuracy under different memory budgets (as shown in Sec 3) and FLOPs reduction. However, these metrics are indirect metrics and ignore many aspects that affect performance. In this work, we show the limitations of filter pruning methods in terms of latency reduction. Aspects such as memory transfer, cache capacity, model parallelism, and hardware architecture affect latency but are not reflected in FLOPs metric. The speedup gain in filter pruning is also dependent on the pruned model signature (*i.e.,*pruning ratio per layer). To remedy these issues, we explore a more hardware-friendly pruning through layer pruning. Fig. 4.1 shows the range of attainable latency reduction on randomly generated models. Each box bar summarizes the latency reduction of 100 random models with filter and layer pruning on different network architectures and hardware platforms. For each filter pruned model i, a random pruning ratio $p_{i,j}$ per layer j such that $0 \leq p_{i,j} \leq 0.9$ is generated; thus models differ in width. For each layer pruned model, M layers out of total L layers (dependent on the network) are randomly selected for retention such that $1 \leq M \leq L$ thus models differ in depth. As to be expected, layer pruning has a higher upper bound in latency reduction compared to filter pruning, especially on modern complex architectures with residual blocks.

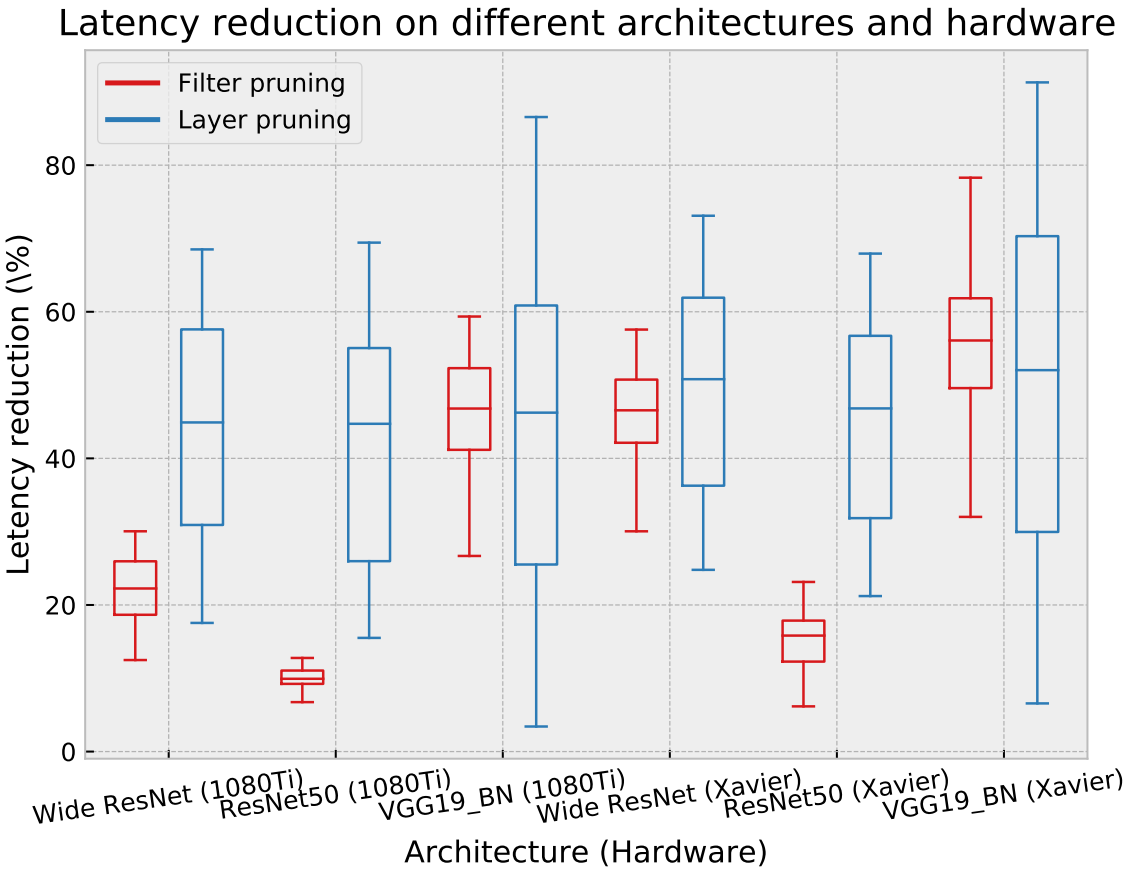


Figure 4.1: Example of 100 randomly pruned models per boxplot generated from different architectures. The plot shows layer pruned models have a wider range of attainable latency reduction consistently across architectures and different hardware platforms (1080Ti and Xavier). Latency is estimated using 224x224 input image and batch size=1.

However, we want to highlight quantitatively in the plot the discrepancy of attainable latency reduction using both methods. Filter pruning is not only constrained by the depth of the model but also by the connection dependency in the architecture. An example of such connection dependency is the element-wise sum operation in the residual block between identity connection and residual connection. Filter pruning methods commonly prune in-between convolution layers in a residual to respect the number of channels and spatial dimensions. BAR [91] proposed an atypical residual block that allows mixed connectivity between blocks to tackle the issue. However, this requires special implementations to leverage the speedup gain and results in sub-

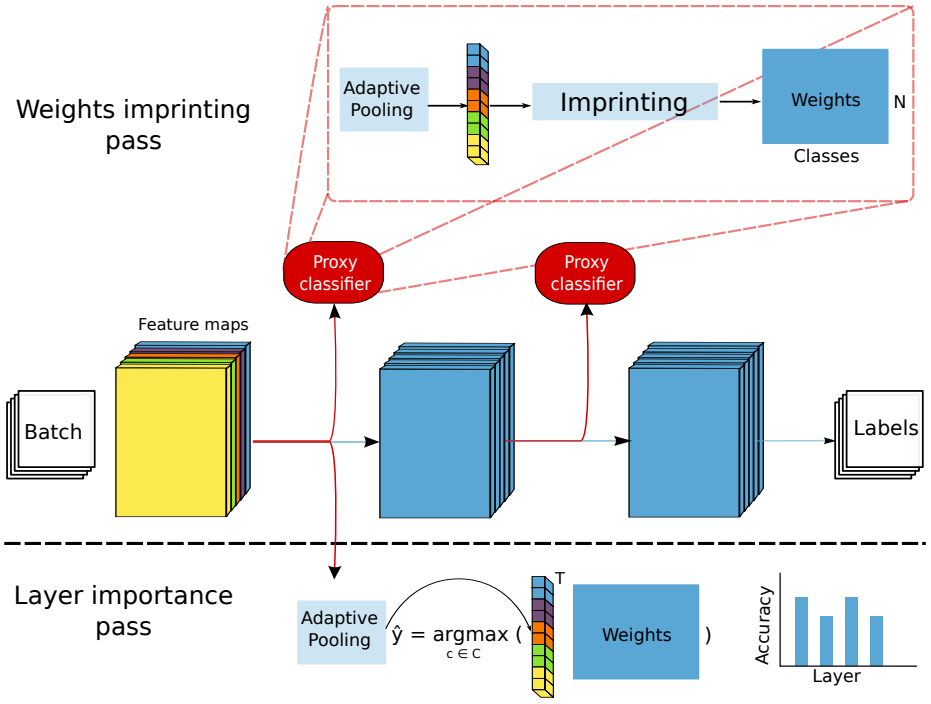


Figure 4.2: Proposed layer-wise accuracy prediction by imprinting. Feature maps are flattened to the same embedding length N in all layers using adaptive average pooling. First phase implements weights imprinting using training data, where a proxy classifier is estimated after each candidate layer for pruning. Each column (i.e class) in the weights matrix is imprinted as the average embedding for all samples belonging to that class. Second phase uses the imprinted weights to estimate layer-wise class predictions ($\hat{y}$) using a validation set. Finally, layers are ranked based on their accuracy difference to be pruned.

optimal memory access. Another limitation in filter pruning is the iterative process. Filter pruning adopts iteratively there steps: 1) filter ranking, 2) prune, and 3) short fine-tune. That is to allow for filter ranking re-evaluation after incremental pruning. This process is constrained to keep a minimum number of filters per layer during optimization to allow for data passing. In some cases with a high pruning ratio, layers collapse to one filter depending on the ranking criteria.

Motivated by these points, what remains to ask is **how well layer pruned models perform in terms of accuracy compared to filter pruned methods**. Another question is **how to evaluate layer importance in an efficient way**.

Key points: We study pruning granularity and its effect on latency and FLOPs

reduction. In specifics, we show that layer pruned models have higher latency reduction and are more hardware-friendly than filter pruning commonly proposed in the literature. We propose a novel layer ranking criterion based on accuracy approximation using imprinting.

4.2 Proposed Method

The brute-force way to check if a layer contributes to the model's representative power would be to check accuracy with and without the layer. However, this approach would be combinatorial and expensive to evaluate. We propose to apply a proxy classifier after each pruning candidate layer (e.g convolutional layers, fully connected layers, or residual blocks) in a one-shot pass by imprinting. Imprinting is used in the few-shot learning [92, 93] to approximate a classifier's weights when only a few training samples are available. Although we have adequate training samples, we are inspired by the efficiency of imprinting and utilize it to approximate the accuracy up to each layer. Fig. 4.2 shows the pipeline of our proposed method consisting of 1) weights imprinting pass and 2) layer importance pass.

4.2.1 Weights imprinting.

In this phase, training data is used to imprint the classifier weight matrix for each pruning candidate layer. Since each layer has a different output feature shape, we apply adaptive average pooling to simplify our method and unify the embedding length so that each layer produces roughly an output of the same size. Specifically, the pooling is done as follows:

$$\begin{aligned} d &= round(\sqrt{\frac{N}{f_i}}) \\ E_i &= AdaptiveAvgPool(F_i, d), \end{aligned} \tag{4.1}$$

where N is the embedding length, f_i is layer i's number of filters, F_i is layer i's output feature map, and AdaptiveAvgPool [94] reduces F_i to embedding $E_i \in \mathbb{R}^{d \times d \times f_i}$. Finally, embedding per layer is flattened to be used in imprinting. Imprinting calculates the weights matrix W_i used for classification as follows:

$$W_i[:, c] = \frac{1}{N_c} \sum_{j=1}^{N} \mathbb{I}_{[c_j == c]} E_j \tag{4.2}$$

where c is the class id, N_c is the number of samples in class c, and N is the total number of samples.

4.2.2 Layer importance.

In this phase, validation data is used to calculate the accuracy per layer given the imprinted weight matrix. The prediction for each sample j is calculated for each layer i as:

$$\hat{y}_j = \underset{c \in \{1,\ldots,C\}}{\operatorname{argmax}} W_i[:, c]^T E_j, \tag{4.3}$$

where E_j is calculated as shown in Eq.4.1. This is equivalent to finding the nearest class from the imprinted weights in the embedding space. After acquiring accuracy for each layer, we rank the layers based on their accuracy difference from the preceding pruning candidate. We prune layers with the least till a budget is achieved.

4.3 Experiments and Analysis

We compare our method with several state-of-the-art methods: Taylor [60], ECC [19], masking [3], slimming [61] and ThiNet [62]. We set learning rate to 0.1 with SGD optimizer for all methods except for ECC, where we use their default setup with Adam optimizer. We add our proxy classifier after transformations (e.g BatchNorm) and non-linearity (e.g ReLU) that follow convolutional or fully connected layers. In all experiments, we set N in Eq 4.1 to the length of the last layer of the architecture.

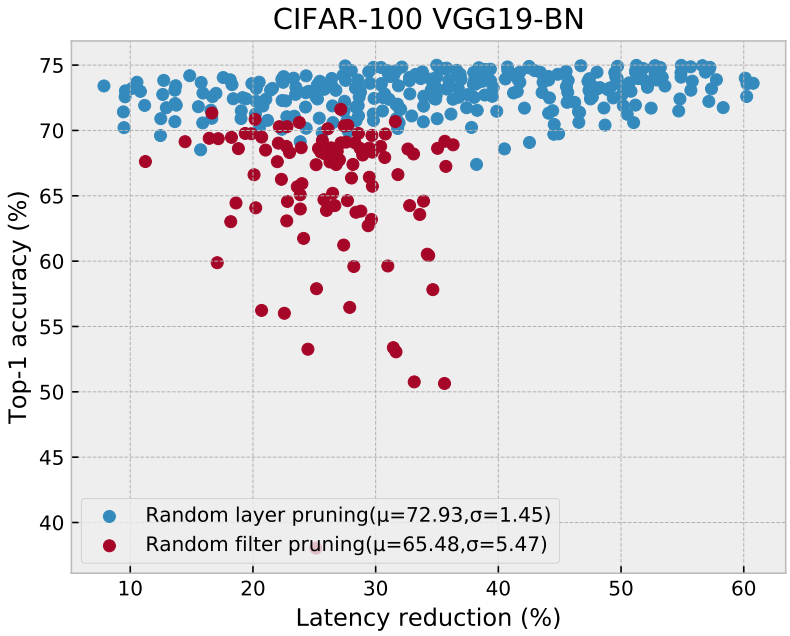


Figure 4.3: Example of 100 random filter pruned and layer pruned models generated from VGG19-BN (Top-1=73.11%). Accuracy mean and standard deviation is shown in parentheses. Latency is calculated on 1080Ti with batch size 8.

4.3.1 Random filters vs. Random layers

Initial hypothesis verification is to generate random filter and random layer pruned models, then train them to compare their accuracy and latency reduction. Random models generation follows the same setup as explained in Section (4.1). Each model is trained with SGD optimization for 164 epochs with a learning rate 0.1 that decays by 0.1 at epochs 81, 121, and 151. Figure 4.3 shows the latency-accuracy plot for both random pruning methods. Layer pruned models outperform filter pruned ones in accuracy by 7.09% on average and can achieve up to 60% latency reduction. In addition, within the same latency budget, filter pruning shows a higher variance in accuracy than layer pruning. This suggests that latency-constrained optimization with filter pruning is complex and requires careful per-layer pruning ratio selection. On

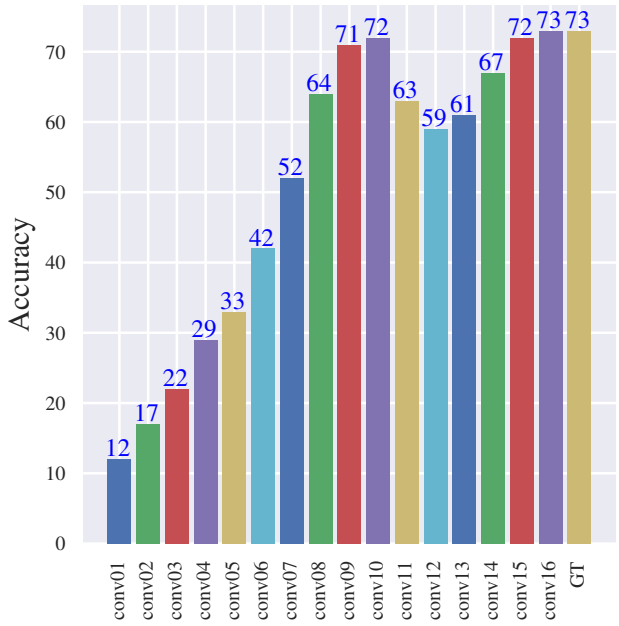


Figure 4.4: Layer-wise accuracy, rounded for better visualization, using proposed proxy classifier for VGG19 on CIFAR100. GT shows the actual accuracy of the full model.

the other hand, layer pruning has small accuracy variation, in general within a budget.

4.3.2 CIFAR100

Fig. 4.4 shows layer-wise accuracy using our proposed proxy classifier for VGG19-BN on CIFAR100. It is worth noting that both the proxy classifier from the last layer, conv16, and the actual model classifier, GT, have the same accuracy, showing how the proxy classifier is a plausible approximation to the converged classifier. We see a drop in accuracy followed by an increasing trend from conv10 to conv14. This is likely because the number of features is the same from conv10 to conv12. We start to observe an accuracy increase only at conv13 that follows a max-pooling layer and has twice as many features. This shows the importance of pooling and the increase in the number of features at this point of the model. We train the pruned model after removing layers conv11-14. Table 4.1 compares the competing filter pruning methods

in terms of accuracy and latency reduction. Our method improves on the previously reported accuracy by 1.18% while achieving a 43.70% latency reduction over VGG19 and a 16.95% speed up over [3]. We also compare with random layer pruning to evaluate the quality of our selection of unimportant layers. We outperform the average of 10 randomly layer-pruned models of similar latency reduction as ours ($\approx 40\%$) by 5.43% in accuracy.

Method	Accuracy	N layers	Params ($1e^6$)	Latency reduction (%)
VGG19 baseline	73.11	16	20.09	0
Random layer pruning	68.95	-	-	40.00
Layer-wise proxy (ours)	**74.38**	**12**	9.28	**43.70**
Slimming [61]	72.32	16	5.00	25.26
Masking [3]	**73.2**	16	**4.20**	**26.75**
Taylor [60]	72.61	16	**4.79**	23.24
ECC [19]	72.71	16	7.86	25.17

Table 4.1: **Pruning results on CIFAR100** showing **best** and **second best** in each criterion. Latency reduction is measured on 1080Ti GPU across 1000 runs.

4.3.3 ImageNet

We also evaluate our method on the challenging ImageNet (ILSVRC2012) dataset on ResNet-50 architecture. For all experiments in this section, we start from pre-trained models available in PyTorch [95] and follow the same training setup as [60]. Similar to the previous setup, we insert a proxy classifier after each pruning candidate. We revise ResNet-50 architecture in Figure 4.5. As can be seen, ResNet-50 consists of multiple blocks where each branch into a residual part and identity part. This creates a tensor shape dependency between the two branches, we experiment with two variants of our method. First, we only treat each residual block as a pruning candidate, so that we prune by removing a block instead of a layer. Second, we treat residual layers as normal layers and remove identity connections when layers within the residual block are pruned.

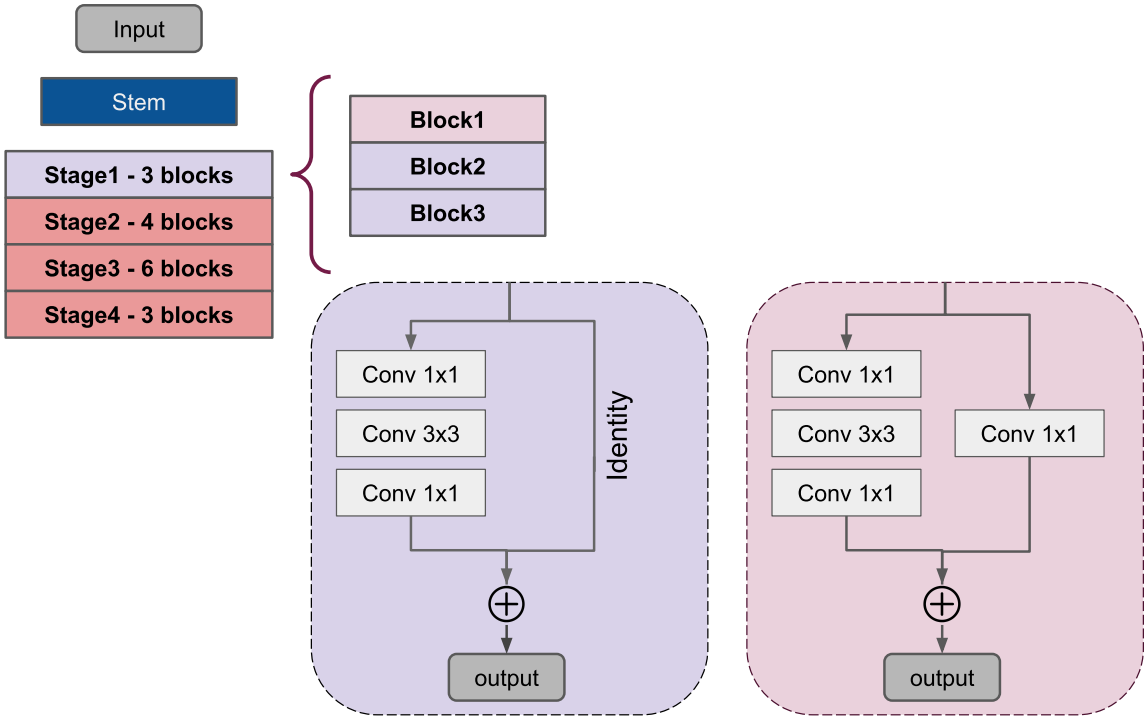


Figure 4.5: ResNet50 architecture.

Figure 4.6 shows layer-wise (bars) and block-wise (color-coded) accuracy with our method. Analyzing the block-wise accuracy will lead us to remove block 3 (no accuracy improvement from block 2). Block 3 is the last block that operates on 56x56 input images. Similar to our previous experiment on CIFAR100, accuracy tends to saturate till an input size downscaling by pooling or stride is applied. Numerical results are presented in Table 4.2. Pruning block 3 results in 16% latency reduction with slight accuracy improvement. On the other hand, we also observe a repeated drop in accuracy in the feature upscaling layers (second to last layers in blocks). Pruning 3 of these layers caused a 1.14% drop in total accuracy. This can be explained by the fact that these residual features are added to the input feature maps. Thus, their accuracy cannot be compared to the previous block's output on their own but only as residuals to input (block output). From these results, we proceeded with pruning blocks as a whole instead of layers.

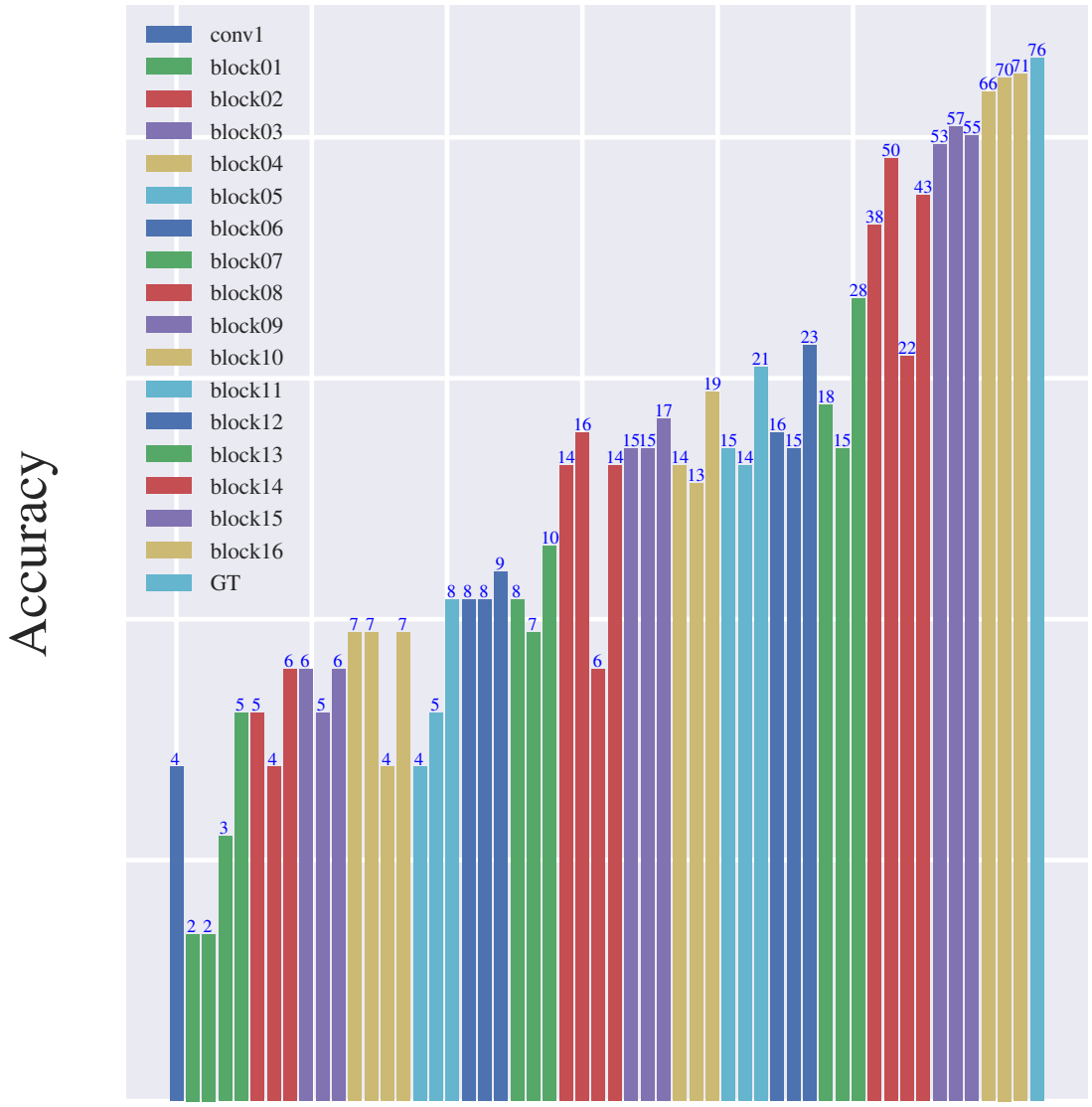


Figure 4.6: Layer-wise accuracy using proposed proxy classifier for ResNet-50 on ImageNet. GT shows the actual accuracy of the full model. Log scaling is used on y-axis for better visualization.

We challenged our method by pruning more blocks and compare with manually designed ResNet variants with similar latency reduction. Our method outperforms ResNet-41 by 0.9% and ResNet-34 by 1.44%. It is also worth noting that the minimal model that can be achieved by filter pruning methods such as ECC achieves 11.56% latency reduction. The minimal model is one with the same depth as the dense

model but with one filter per each prunable layer. This model will likely result in GPU under-utilization and very low accuracy. In comparison, we can achieve up to 39% latency reduction by pruning layers with a 60.1% accuracy gain.

Method	Accuracy	N layers	Params ($1e^6$)	Latency reduction (%)
ResNet-50 baseline	76.14	53	25.5	0
Layer-wise proxy - 1 block (ours)	**76.72**	**50**	25.4	**16.06**
Layer-wise proxy - 1 block + 3 layers (ours)	75.0	**44**	24.1	**24.02**
ThinNet [62]	72.04	53	**16.94**	10.52
Taylor [60]	**76.43**	53	**22.6**	2.73
ECC [19]	74.88	53	23.5	1.93
ECC minimal model	16.3	53	6.14	11.56
Layer-wise proxy - 4 blocks (ours)	**76.40**	**41**	**24.8**	25
ResNet-41 [69]	75.50	44	25.3	25
Layer-wise proxy - 6 blocks (ours)	**74.74**	**35**	23.4	39
ResNet-34 [4]	73.30	37	**21.7**	39

Table 4.2: Pruning results on ImageNet showing **best** and **second best** in each criterion. Latency reduction is measured on 1080Ti GPU across 1000 runs with batch size=1.

4.4 Conclusion

We have proposed a novel method to apply a one-shot proxy classifier by weight imprinting to evaluate the classification accuracy per layer. We have demonstrated that our layer pruning method achieves much better latency reduction than the state-of-the-art filter pruning methods. A thorough evaluation on different inference setups such as deployment hardware platforms and batch sizes are explored. In addition to adopting filter criteria proposed in the literature in a layer-pruning pipeline.

Chapter 5

Generalized Layer Pruning with LayerPrune Framework

5.1 Motivation

In this work, we extend the idea of layer pruning presented in Sec 4 and propose a LayerPrune framework. LayerPrune presents a set of layer pruning methods based on different criteria that are adopted from filter pruning literature. The goal of this work is to compare layer pruning and filter pruning under different inference setups such as hardware platform, batch size, and importance criteria.

Fig. 5.1 shows accuracy and images per second between our LayerPrune and several state-of-the-art pruning methods, as well as, several handcrafted architectures. In general, pruning methods tend to find better quality models than handcrafted architectures. It is worth noting that filter pruning methods such as ThiNet [62] and Taylor [60] show a small speedup gain as more filters are pruned compared to depth pruning such as our LayerPrune and SSS [69]. This shows the limitation of filter pruning methods on latency reduction.

Key points: We extend the work from chapter 4 to generalize a layer pruning framework. We present a thorough study on filter vs layer pruning under different ranking criteria, batch sizes, and hardware. In addition to efficient inference, our layer pruned models accelerate fine-tuning as well. In filter pruning iterative training, binary masks simulate dropping filters in terms of task optimization but will not have

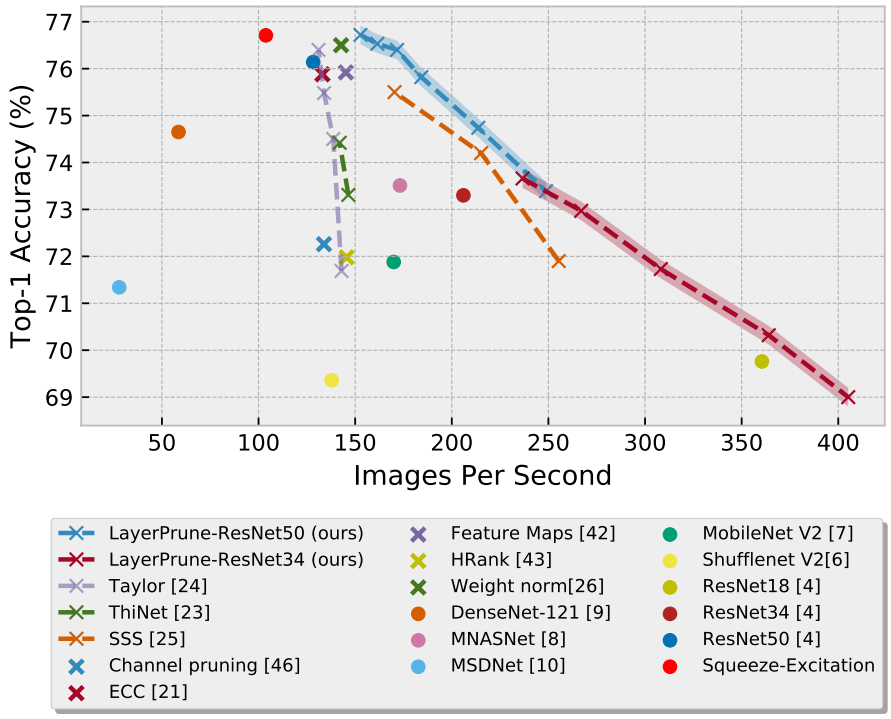


Figure 5.1: Evaluation on ImageNet between our LayerPrune framework, handcrafted architectures (dots) and pruning methods on ResNet50 (crosses). Inference time is measured on 1080Ti GPU.

actual performance gain in training. However, our layer pruned model prunes the layers in one-shot by replacing them with identity blocks. Fine-tuning for the layer pruned models has accelerated thanks to identity replacement and one-shot pruning.

5.2 Proposed Method

In this section, we describe in detail LayerPrune for layer pruning using existing filter criteria along with a novel layer-wise accuracy approximation. A typical filter pruning method follows a three-stage pipeline as illustrated in Figure 5.2. Filter importance is iteratively re-evaluated after each pruning step based on a pruning meta-parameter such as pruning N filters or pruning those with importance $\leq$ threshold. In LayerPrune, we remove the need for the iterative pruning step and show that using the

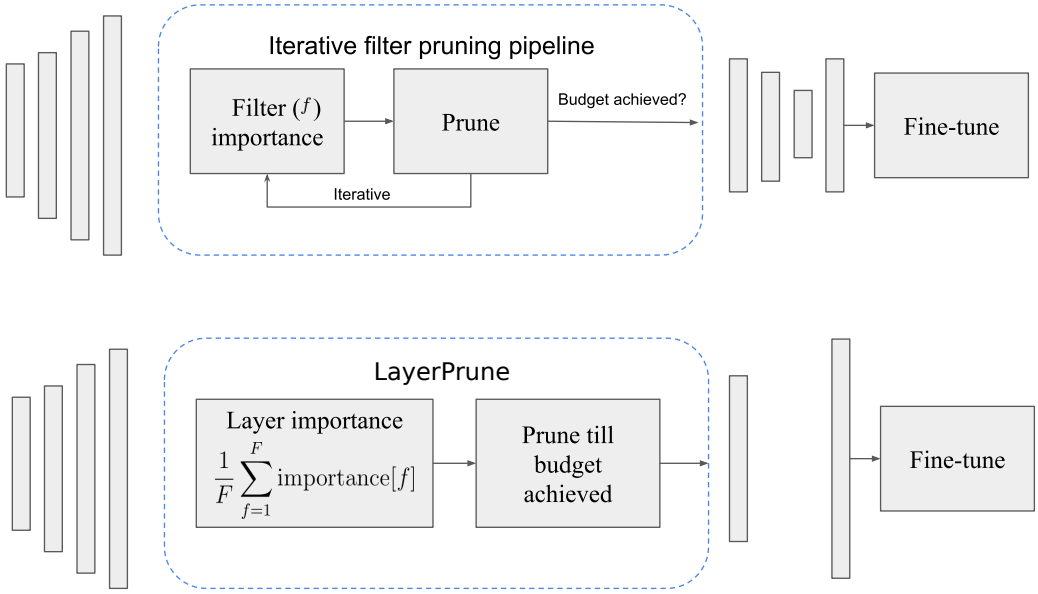


Figure 5.2: Illustrates the difference between typical iterative filter pruning and the proposed LayerPrune framework. Filter pruning (top) produces thinner architecture in an iterative process while LayerPrune (bottom) prunes whole layers in one-shot. In LayerPrune, layer's importance is calculated as the average importance of each filter f in all filters F at that layer.

same filter criterion, we can remove layers in one-shot to respect a budget. This simplifies the pruning step to a hyper-parameter-free process and is computationally efficient. Layer importance is calculated as the average of filter importance in this layer.

Although existing filter pruning methods are different in algorithms and optimization used, they focus more on finding the optimal per-layer number of filters and share common filter criteria. We divide the methods based on the filter criterion used and propose their layer importance counterpart used in LayerPrune.

5.2.1 Pruning Criteria

Preliminary notion. Consider a network with L layers, each layer l has weight matrix $W^{(l)} \in \mathbb{R}^{N_l \times F_l \times K_l \times K_l}$ with N_l input channels, F_l number of filters and K_l is the size of the filters at this channel. Evaluated criteria and methods are:

Weight statistics. [19, 53, 64] differ in the optimization algorithm but share weight statistics as a filter ranking. Layer pruning for this criteria is calculated as:

$$\text{weights-layer-importance}[l] = \frac{1}{F_l} \sum_{i=1}^{F_l} \left\| W^{(l)}[:, i, :, :] \right\|_2 \tag{5.1}$$

Taylor weights. Taylor method [60] is slightly different from previous criterion in that the gradients are included in the ranking as well. Filter f ranking is based on $\sum_s (g_s w_s)^2$ where s iterates over all individual weights in f, g is the gradient, w is the weight value. Similarly, layer ranking can be expressed as:

$$\text{taylor-layer-importance}[l] = \frac{1}{F_l} \sum_{i=1}^{F_l} \left\| G^{(l)}[:, i, :, :] \odot W^{(l)}[:, i, :, :] \right\|_2 \tag{5.2}$$

where $\odot$ is element-wise product and $G^{(l)} \in \mathbb{R}^{N_l \times F_l \times K_l \times K_l}$ is the gradient of loss with respect to weights $W^{(l)}$.

Feature map based heuristics. [62, 96, 97] rank filters based on statistics from output of layer. In [62], ranking is based on the effect on the next layer while [96], similar to Taylor weights, utilizes gradients and norm but on feature maps.

Channel saliency. In this criterion, a scalar is multiplied by the feature maps and optimized within a typical training cycle with task loss and sparsity regularization loss to encourage sparsity. Slimming [61] utilizes Batch Normalization scale γ as the channel saliency. Similarly, we use Batch Normalization scale parameter to calculate layer importance for this criteria, specifically:

$$\text{BN-layer-importance}[l] = \frac{1}{F_l} \sum_{i=1}^{F_l} (\gamma_i^{(l)})^2 \tag{5.3}$$

Ensemble. We also consider diverse ensemble of layer ranks where the ensemble rank of each layer is the sum of its rank per method, more specifically:

$$\text{ensemble-rank}[l] = \sum_{m \in \{1 \ldots M\}} (\text{LayerRank}(m, l)) \tag{5.4}$$

where l is the layer's index, M is the number of all criteria and LayerRank indicates the order of layer l in the sorted list for criterion m.

5.3 Experiments and Analysis

5.3.1 Training Setup

We follow standard hyperparameters used for fine-tuning [20, 60, 69]: 30 epoch learning rate $1e^{-3}$ on SGD optimizer. An exception is the comparison with Chen et al. [70] as the authors train the pruned model with the standard hyperparameters used for training from scratch: 160 epochs with an initial learning rate of 0.1 and decays on epoch [81, 122] by 0.1.

Layer pruning: For layer pruning, we calculate layer importance as explained in Section 5.2.1 using a one-shot pass over the training set.

Filter pruning: For filter pruning, we prune total 500 and 100 filters in VGG19 and ResNet56 respectively for global-based filter importance criteria such as weight norm, Taylor approximation, and feature maps. We follow the same iterative pruning hyperparameter setup as Taylor [60]. We prune 100 filters each 10 mini-batches. For other pruning methods, we report results using their published code with default setup setting such as slimming [61] and ECC[19].

5.3.2 CIFAR

We evaluate CIFAR-10 and CIFAR-100 on ResNet-56 [4].

ResNet56

We compare on the the complex architecture ResNet-56 on CIFAR-10 and CIFAR-100 in Table 5.1. On a similar latency reduction, LayerPrune outperforms [70] by 0.54% and 1.23% on CIFAR-10 and CIFAR-100 respectively. On the other hand, within each filter criterion, LayerPrune outperforms filter pruning and is on par with the baseline in accuracy. In addition, filter pruning can result in a latency increase (i.e negative LR) with specific hardware targets and batch sizes [98] as shown with batch size 8. However, LayerPrune consistently shows latency reduction under different environmental setups. We also compare with larger batch sizes to further encourage filter pruned models to better utilize the resources. Still, we found LayerPrune achieves overall better latency reduction with a large batch size. Latency reduction variance, LR var, between different batch sizes within the same hardware platform is shown as well. Consistent with previous results on VGG, LayerPrune is less sensitive to changes in criterion, batch size, and hardware than filter pruning. We also show results up to 2.5x latency reduction with less than 2% accuracy drop.

Latency vs Number of Filters

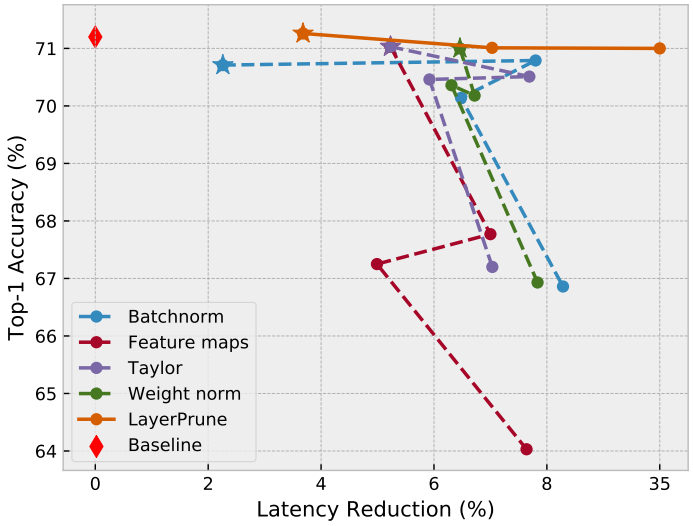
We show accuracy degradation on aggressive filter pruning and the achieved latency reduction compared to LayerPrune. Fig. 5.3 shows filter pruning under different number of filters pruned (i.e 100:400) and latency reduction on GPU 1080Ti on batch size=64. Dots are connected based on ascending order of the number of filters pruned. It is apparent that pruning more filters doesn't necessarily decrease latency and the relationship between pruned filters and latency reduction is non-linear. In CIFAR-100, a latency reduction of $\approx 8\%$ results in a large drop in accuracy from 71.2% to 67%. It is worth noting that LayerPrune is able to achieve up to 35% latency reduction with accuracy 71%. Similarly on CIFAR-10, pruning 50% of the filters can only achieve around 5% latency reduction.

Method	Shallower?	Top1-accuracy (%)	LR (%) 1080Ti bs=8	LR (%) 1080Ti bs=64	LR (%) Xavier bs=8	LR (%) Xavier bs = 64
CIFAR-10 ResNet-56 baseline (93.55%)						
Chen et al. [70]	✓	93.09	**26.60**	**26.31**	26.96	25.66
LayerPrune$_8$-Imprint	✓	**93.63**	**26.41**	**26.32**	**27.30**	**29.11**
Taylor weight [60]	✗	93.15	0.31	5.28	-0.11	2.67
LayerPrune$_1$	✓	**93.49**	2.864	**3.80**	5.97	5.82
LayerPrune$_2$	✓	93.35	**6.46**	**8.12**	**9.33**	**11.38**
Weight norm [53]	✗	92.95	-0.90	5.22	1.49	3.87
L1 norm [64]	✗	93.30	-1.09	-0.48	2.31	1.64
LayerPrune$_1$	✓	**93.50**	2.72	3.88	7.08	5.67
LayerPrune$_2$	✓	93.39	**5.84**	**7.94**	**10.63**	**11.45**
Feature maps [96]	✗	92.7	-0.79	6.17	1.09	**8.38**
LayerPrune$_1$	✓	**92.61**	3.29	2.40	7.77	2.76
LayerPrune$_2$	✓	92.28	**6.68**	**5.63**	**11.11**	5.05
Batch Normalization [61]	✗	93.00	0.6	3.85	2.26	1.42
LayerPrune$_1$	✓	**93.49**	2.86	3.88	7.08	5.67
LayerPrune$_2$	✓	93.35	**6.46**	**7.94**	**10.63**	**11.31**
LayerPrune$_{18}$-Imprint	✓	92.49	57.31	55.14	57.57	63.27
CIFAR-100 ResNet-56 baseline (71.2%)						
Chen et al. [70]	✓	69.77	**38.30**	34.31	38.53	39.38
LayerPrune$_{11}$-Imprint	✓	**71.00**	**38.68**	**35.83**	**39.52**	**54.29**
Taylor weight [60]	✗	71.03	2.13	5.23	-1.1	3.75
LayerPrune$_1$	✓	**71.15**	3.07	3.74	3.66	5.50
LayerPrune$_2$	✓	70.82	**6.44**	**7.18**	**7.30**	**11.00**
Weight norm [53]	✗	71.00	2.52	6.46	-0.3	3.86
L1 norm [64]	✗	70.65	-1.04	4.06	0.58	1.34
LayerPrune$_1$	✓	**71.26**	3.10	3.68	4.22	5.47
LayerPrune$_2$	✓	71.01	**6.59**	**7.03**	**8.00**	**10.94**
Feature maps [96]	✗	70.00	1.22	9.49	-1.27	**7.94**
LayerPrune$_1$	✓	**71.10**	2.81	3.24	4.46	5.56
LayerPrune$_2$	✓	70.36	**6.06**	**6.70**	**7.72**	**7.85**
Batch Normalization [61]	✗	70.71	0.37	2.26	-1.02	2.89
LayerPrune$_1$	✓	**71.26**	3.10	3.68	4.22	5.47
LayerPrune$_2$	✓	70.97	**6.36**	**6.78**	**7.59**	**10.94**
LayerPrune$_{18}$-Imprint	✓	68.45	60.69	57.15	61.32	71.65

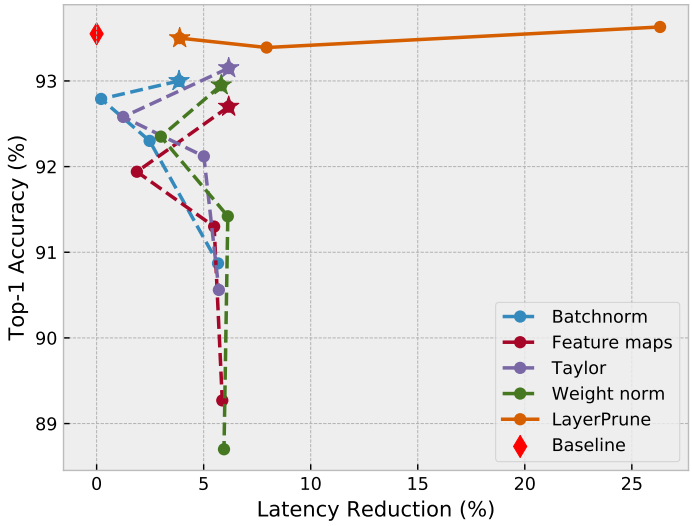
Table 5.1: **Comparison of different pruning methods on ResNet56 CIFAR-10/100.** The accuracy for baseline model is shown in parentheses. LR and bs stands for latency reduction and batch size respectively. x in LayerPrune$_x$ indicates number of blocks removed.

As for VGG19, the maximally achieved pruning latency reduction is 20% to maintain the accuracy from baseline. On the other hand, LayerPrune finds better models

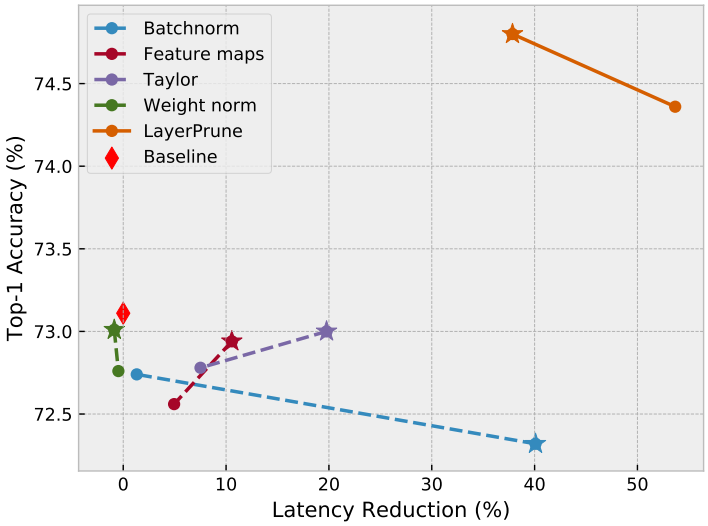
than baseline and filter pruned methods. On comparison with the random experiment shown in Section 4.3.1, filter pruning methods hover around baseline accuracy and fail to discover other regularized models compared to layer pruning.



(a) CIFAR-100/ResNet56



(b) CIFAR-10/ResNet56



(c) CIFAR-100/VGG19-BN

Figure 5.3: Latency reduction of different filter pruning methods under different pruning ratios. Star in each method indicates the lowest pruning ratio (starting point). Dots are connected based on ascending order of number of filters pruned.

5.3.3 ImageNet

We evaluate the methods on the challenging ImageNet dataset for classification. For all experiments in this section, PyTorch pre-trained models are used as starting point for network pruning. We follow the same setup as in [60] where we prune 100 filters for each 30 mini-batches for 10 pruning iterations. The pruned model is then fine-tuned with a learning rate $1e^{-3}$ using SGD optimizer and 256 batch size. Results on ResNet50 are presented in Table 5.2.

In general, LayerPrune methods improve accuracy over the baseline and their counterpart filter pruning methods. Although feature maps criterion [96] achieves better accuracy by 0.92% over LayerPrune$_1$, LayerPrune has a higher latency reduction that exceeds by 5.7%. It is worth mentioning that the latency aware optimization ECC has an upper bound latency reduction of 11.56%, on 1080Ti, with an accuracy of 16.3%. This stems from the fact that iterative filter pruning is bounded by the network's depth and structure dependency within the network, thus not all layers are considered for pruning such as the gates at residual blocks. In addition, we compare with ECC which is a hardware-aware pruning method. ECC builds a layer-wise bilinear model to approximate the latency of a model given the number of input channels and output filters per layer. This simplifies the non-linear relationship between the number of filters per layer and latency. We show latency reduction on Xavier for an ECC pruned model optimized for 1080Ti, and this pruned model results in latency increase on batch size 1 and the lowest latency reduction on batch size 64. This suggests that a hardware-aware filter pruned model for one hardware architecture might perform worse on another hardware than even a hardware-agnostic filter pruning method.

It is worth noting that the filter pruning HRank [97] with 2.6x FLOPs reduction shows large accuracy degradation compared to LayerPrune (71.98 vs 74.31). Even with aggressive filter pruning, speed up is noticeable with large batch size but shows small speed gain with small batch size. Within shallower models, LayerPrune out-

performs SSS on the same latency budget even when SSS supports block pruning for ResNet50, which shows the effectiveness of accuracy approximation as layer importance.

In the Xavier edge device case, we observe a lower latency reduction on batch size 64 in all methods than on batch size 1. The reason for that, as the batch size increases, the feature map dimensions increase as well, this results in extra overload for data transfer between different storage memory. Nonetheless, the manually predefined architecture used in ThiNet [62] suffer less from this data transfer overhead with a large batch size due to the uniform pruning ratio. Yet, accuracy suffers from that uniform pruning and results in a sub-optimal pruned model. This opens the door for future interesting research questions such as, how can we utilize uniform filter pruning to serve as an initial model for layer pruning? A joint uniform filter pruning and layer pruning would result in a hardware-friendly pruning technique. Or how to achieve a certain memory and latency budget jointly utilizing both pruning paradigms within a simple one-shot pass?

5.3.4 Ablation Study

One-shot vs Iterative

We conducted experiments on one-shot vs iterative filter pruning to be comparable with our one-shot LayerPrune pruning step. Our reported results on iterative filter pruning follow the same setup used in literature [60], that is prune 10% each pruning iteration. In one-shot filter pruning, the hyperparameter total number of filters is pruned at once. Table 5.3 shows results of iterative vs one-shot. Consistent with [20, 60], iterative pruning (i.e re-evaluating criterion of filter after each prune) gives a slightly better accuracy. That shows that it is mandatory for filter pruning to be iterative.

We also analyzed the sensitivity of ranking by imprinting on layer pruned models in one-shot and iterative ranking. We calculated Spearman's rank-order correlation

ResNet50 baseline (76.14)						
Method	Shallower?	Top1-accuracy (%)	LR(%) 1080Ti bs=1	LR (%) 1080Ti bs=64	LR (%) Xavier bs=1	LR (%) Xavier bs = 64
Weight norm [53]	✗	76.50	6.79	3.46	6.57	8.06
ECC [19]	✗	75.88	13.52	1.59	-4.91**	3.09**
LayerPrune$_1$	✓	**76.70**	15.95	4.81	21.38	6.01
LayerPrune$_2$	✓	76.52	**20.32**	**13.23**	**26.14**	**13.20**
Batch Normalization	✗	75.23	2.49	1.61	-2.79	4.13
LayerPrune$_1$	✓	**76.70**	15.95	4.81	21.38	6.01
LayerPrune$_2$	✓	76.52	**20.41**	**8.36**	**25.11**	**9.96**
Taylor [60]	✗	76.4	2.73	3.6	-1.97	6.60
LayerPrune$_1$	✓	**76.48**	15.79	3.01	21.52	4.85
LayerPrune$_2$	✓	75.61	**21.35**	**6.18**	**27.33**	**8.42**
Feature maps [96]	✗	**75.92**	10.86	3.86	20.25	8.74
Channel pruning* [17]	✗	72.26	3.54	6.13	2.70	7.42
ThiNet* [62]	✗	72.05	10.76	**10.96**	15.52	**17.06**
LayerPrune$_1$	✓	75.00	16.56	2.54	23.82	4.49
LayerPrune$_2$	✓	71.90	**22.15**	5.73	**29.66**	8.03
SSS-ResNet41 [69]	✓	75.50	25.58	24.17	31.39	21.76
LayerPrune$_3$-Imprint	✓	**76.40**	22.63	25.73	30.44	20.38
LayerPrune$_4$-Imprint	✓	75.82	**30.75**	**27.64**	**33.93**	**25.43**
SSS-ResNet32 [69]	✓	74.20	**41.16**	29.69	**42.05**	29.59
LayerPrune$_6$-Imprint	✓	**74.74**	40.02	**36.59**	41.22	34.50
HRank-2.6x-FLOPs* [97]	✗	71.98	11.89	36.09	20.63	**40.09**
LayerPrune$_7$-Imprint	✓	**74.31**	**44.26**	**41.01**	**41.01**	38.39

Table 5.2: **Comparison of different pruning methods on ResNet50 ImageNet.** * manual pre-defined signatures. ** same pruned model optimized for 1080Ti latency consumption model in ECC optimization

Dataset/Model	Pruning ratio	Criterion	Iterative	One-shot
CIFAR100/ResNet56 (71.20%)	20%	Weight Norm	70.18	70.00
		Feature Maps	67.77	67.7
		Taylor	70.51	70.01
		Batchnorm	70.79	70.36
		Median	**70.34**	70.00
	30%	Weight Norm	70.36	69.1
		Feature Maps	67.25	66.34
		Taylor	70.46	68.27
		Batchnorm	70.14	69.4
		Median	**70.25**	68.68
CIFAR10/ResNet56 (93.55%)	20%	Weight Norm	92.35	92.31
		Feature Maps	91.94	91.9
		Taylor	92.88	92.8
		Batchnorm	92.79	92.76
		Median	**92.57**	92.53

Table 5.3: Evaluation of iterative and one-shot filter pruning. Baseline accuracy indicates in parentheses.

	CIFAR-10 ResNet-56 (93.55%)			CIFAR-100 ResNet-56 (71.2%)		
N pruned	One-shot (%)	Iterative (%)	Spearman	One-shot (%)	Iterative (%)	Spearman
1	93.32	93.32	0.99	71.10	71.10	0.96
2	93.28	93.31	0.97	70.93	70.94	0.97
3	93.17	93.15	0.96	70.88	70.86	0.94
4	93.10	93.03	0.96	70.64	70.71	0.91

Table 5.4: Spearman rank correlation between one-shot and iterative ranking with imprinting.

between layer ranking by one-shot (as explained in Section 4.2) and layer ranking by re-calculating ranks iteratively after each pruning step. Table 5.4 shows the accuracy of one-shot and iterative layer pruning and their ranking correlation. The Spearman column indicates a high positive relationship between both ranking methods demonstrating the robustness of ranking by imprinting. We observed the difference in ranking is between similarly important layers and this explains why accuracy is not significantly affected even as correlation decreases, and it shows the sufficiency of one-shot rank estimation with imprinting.

Discussion on Architectures

VGG19-BN Pruned architectures for results on VGG19-BN are presented in Table 5.5. All layer pruning methods mostly agree on removing same layers. While in filter pruning methods, as minimum number of filters are required per layer, the early layers are pruned as well and thus hurting accuracy.

ResNet56 has 3 groups of 9 basic blocks where each basic block has two 3x3 convolution layers. We show block importance based on each criterion for CIFAR-10 in Fig. 5.4 and CIFAR-100 in Fig. 5.5. Weight magnitude, Batch Normalization, and Taylor magnitude criteria have similar block ordering that focuses more on the early layers. On the other hand, the feature maps criterion is more biased toward pruning the deeper layers. This stems from the fact that as we go deeper, feature

Method	Accuracy	Architecture
VGG19 (baseline)	73.11	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M']
Weight norm [53]	73.01	[47, 64, 'M', 127, 128, 'M', 256, 256, 256, 256, 'M', 512, 508, 494, 472, 'M', 502, 512, 499, 509, 'M']
ECC [19]	72.71	[50, 23, 'M', 128, 128, 'M', 254, 254, 254, 254, 'M', 508, 311, 164, 131, 'M', 158, 319, 509, 64, 'M']
Layer pruning$_2$	73.60	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', **0, 0,** 512, 512,'M']
Layer pruning$_5$	74.80	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, **0, 0,** 'M', **0, 0, 0,** 512, 'M']
Slimming [61]	72.32	[42, 64, 'M', 125, 128, 'M', 255, 256, 255, 256, 'M', 433, 291, 82, 46, 'M', 45, 44, 62, 367, 'M']
Layer pruning$_2$	73.60	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', **0, 0,** 512, 512,'M']
Layer pruning$_5$	74.80	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, **0, 0,** 'M', **0, 0, 0,** 512, 'M']
Taylor [60]	72.61	[61, 64, 'M', 127, 128,'M', 256, 256, 256, 256,'M', 512, 505, 383, 205,'M', 109, 118, 422,482, 'M']
Layer pruning$_2$	73.60	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, **0,** 'M', **0,** 512, 512, 512,'M']
Layer pruning$_5$	74.80	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, **0, 0**, 'M', **0, 0, 0,** 512, 'M']

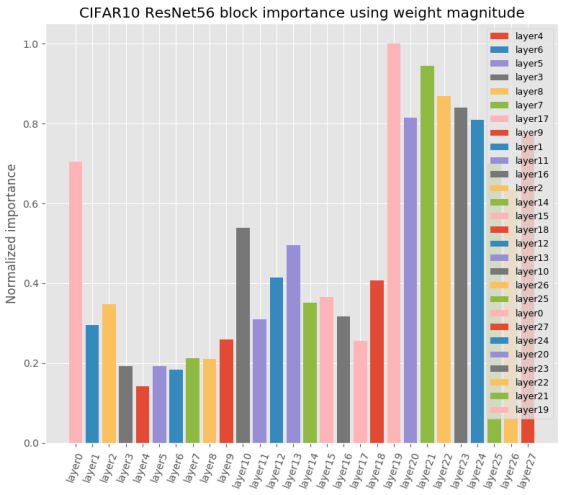
Table 5.5: Architectures of different pruning methods on VGG19-BN CIFAR-100. x in Layer pruning$_x$ indicates number of layers removed. Number of filters per layer is shown where 0 indicates removed layers and 'M' indicates max pooling operation.

maps tend to be sparser and so their importance calculated using Taylor on feature maps [96] will lead to a bias and failure in deeper models. Ensemble selects layers that are constantly voted as not important (e.g CIFAR-10 blocks 6,4,5), however, it is sensitive to individual errors. For example in CIFAR-10, the ensemble prioritizes pruning block17 over block7 even when the latter has lower ranks in most of the criteria but the large ranking gap in one criterion, that is feature maps criterion, resulted in block17 having a lower rank.

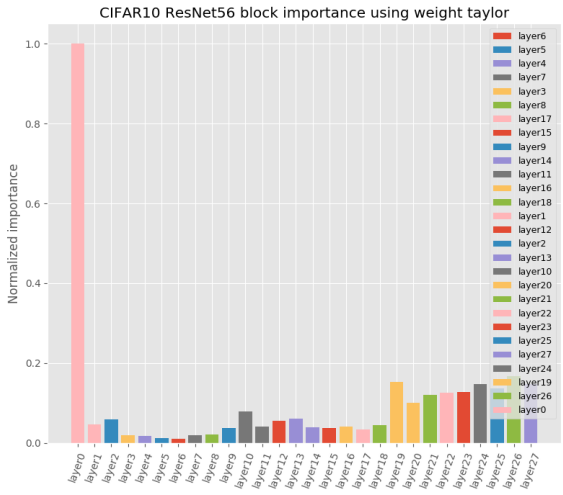
This diversity in criteria calls for a deeper study of a wide range of networks. Different ranking criterion could be optimal for one architecture but fails in another.

Training from Scratch

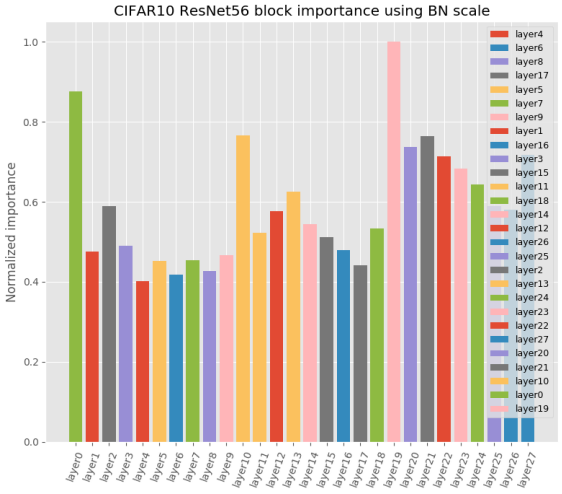
Training from scratch for ImageNet is done for 90 epochs with 0.1 initial lr, 0.1 lr decay each 30 epochs. Fine-tuning is done for 30 epochs with 1e$-$3 initial lr, 0.1 lr decay each 10 epoch. In Table 5.6, we compare our LayerPrune models trained from scratch and fine-tuned. Fine-tuned models consistently outperform training from scratch of the same pruned architecture.



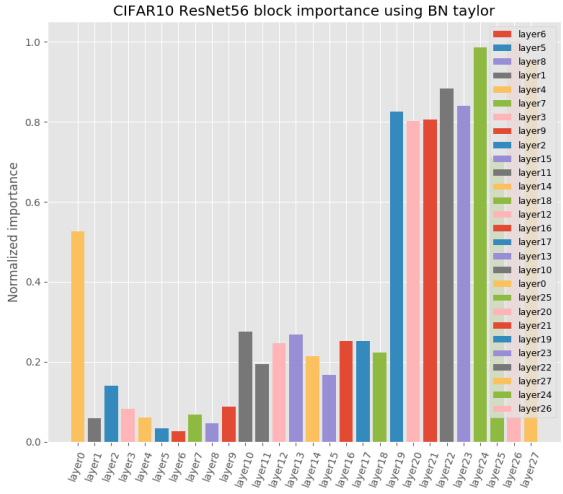
(a) Weight norm



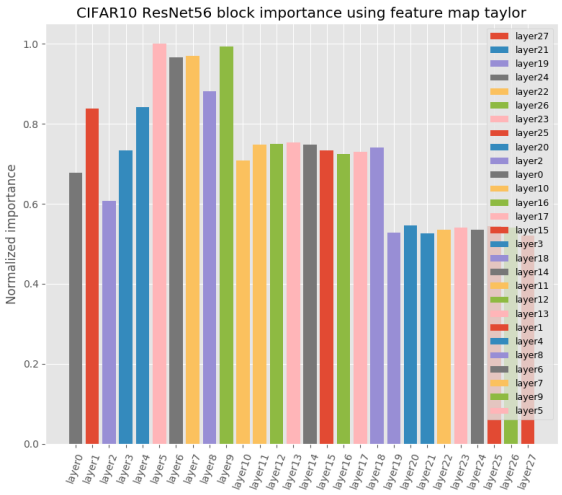
(b) Weight Taylor



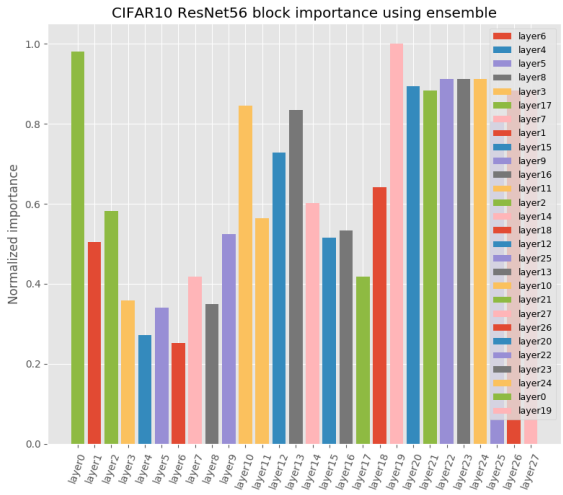
(c) Batch Normalization



(d) Batch Normalization Taylor



(e) Feature maps



(f) Ensemble

Figure 5.4: Plots of block importance using different layer criterion on CIFAR-10 ResNet56. Legend on each sub-plot shows sorted blocks in ascending order based on importance.

(a) Weight norm

(b) Weight Taylor

(c) Batch Normalization

(d) Batch Normalization Taylor

(e) Feature maps

(f) Ensemble

Figure 5.5: Plots of block importance using different layer criterion on CIFAR-100 ResNet56. Legend on each sub-plot shows sorted blocks in ascending order based on importance.

N Blocks pruned	Fine-tuned	Scratch
1	**76.72**	75.70
2	**76.53**	75.96
3	**76.40**	75.80
4	**75.82**	75.0

Table 5.6: Accuracy of our ResNet50 pruned models trained from scratch and fine-tuned.

Training Speed

End-to-end optimization filter pruning methods such as slimming require training from scratch with sparsity-inducing terms in the training. This requires 90 epochs in ImageNet. All methods, including ours, were fine-tuned for 30 epochs. Hence, our layer-pruning is 4 times faster than these methods.

For iterative filter pruning methods, we observed an average 1.9x speedup in the fine-tuning phase in layer-pruned models compared to fine-tuning phase in filter pruning. The training is conducted on 4 x V100 GPUs.

5.4 Conclusion

We presented a LayerPrune framework that includes a set of layer pruning methods. We show the benefits of LayerPrune on latency reduction compared to filter pruning. The key findings of this paper are the following:

- For a filter criterion, training a LayerPrune model based on this criterion achieves the same, if not better, accuracy as the filter pruned model obtained by using the same criterion.

- Filter pruning compresses the number of convolution operations per layer and thus latency reduction depends on hardware architecture, while LayerPrune removes the whole layer. As a result, filter pruned models might produce non-optimal matrix shapes for the compute kernels that can lead even to latency

increase on some hardware targets and batch sizes.

- Filter pruned models within a latency budget have a larger variance in accuracy than LayerPrune. This stems from the fact that the relation between latency and the number of filters is non-linear and optimization constrained by a resource budget requires complex per-layer pruning ratios selection.

- We also showed the importance of incorporating accuracy approximation in layer ranking by imprinting.

Chapter 6

Fire Together Wire Together: A Dynamic Pruning Approach

6.1 Motivation

Static pruning advanced the neural network compression area, however, as we have seen in Figure 5.1, we could fairly perform well using a light-weight model. The full model is needed to correctly predict the complex hard samples. In addition, considering the manually designed architectures in Table 6.1, we usually demand double the FLOPs for approximately 2% gain in accuracy.

Ideally, we would like to be able to distinguish the hard and easy samples at run-time and apply different computation budgets accordingly. This recent perspective allowed for what is known as conditional inference or dynamic pruning. The idea is that instead of a lightweight static network for all input samples, sub-networks are processed conditioned on the input instead. This allows for a higher degree of freedom and the ability to preserve the full ability of the original network. However, training multiple routes or sub-networks within the same network is challenging. In dynamic or conditional inference, samples are processed using different routes (i.e dynamic pruning) or using variant depth of the network (i.e early-exit).

Key points: A typical dynamic filter pruning method equips a decision gate to select a handful of filters to compute per each sample. This decision gate usually is trained through a regularization with continuous channel saliency learning or pre-

Model	Accuracy	FLOPs	Accuracy gain	FLOPs increase
ResNet18	69.76%	1.8B	–	–
ResNet34	73.31%	3.6B	3.55	2.1x
MobileNetv1	70.6%	569M	–	–
MobileNetv1_75	68.4%	325M	2.2	1.75x
Inception v3	78.00	7B	–	–
Inception v4	80.2	16B	2.2	2.2x

Table 6.1: Diminishing returns to adding more FLOPs. Double the computation is needed for $\approx 2\%$ gains

defined N group of filters. These training techniques result in a training instability as balancing the multi-loss training (*i.e.,*decision gates regularization and task loss becomes challenging. In our method, we decouple the two losses to eliminate gradient interference from one loss to another. We achieve this by self-supervised training for the decision gates while updating the model's weights with the task loss.

6.2 Proposed Method

6.2.1 Preliminary

Current dynamic filter pruning approaches typically introduce a regularization term to induce sparsity over a continuous parameter for channel gating/masking [73, 74, 99]. Others adopt policy gradient introduced in reinforcement learning [100] to learn different routes. These methods require careful tuning in training to tackle issues such as training stability with schedule annealing [74], biased training handling [101], or predefined pruning ratio per layer [26, 73].

Also, as noted in [99], additional sparsity loss degrades task loss as it is difficult to balance the task loss and the pruning loss. That is especially evident under a high pruning ratio as shown in Figure 6.1. Moreover, the FLOPs reduction of these dynamic methods is dependent on the selected sparsity weight hyperparameter. This

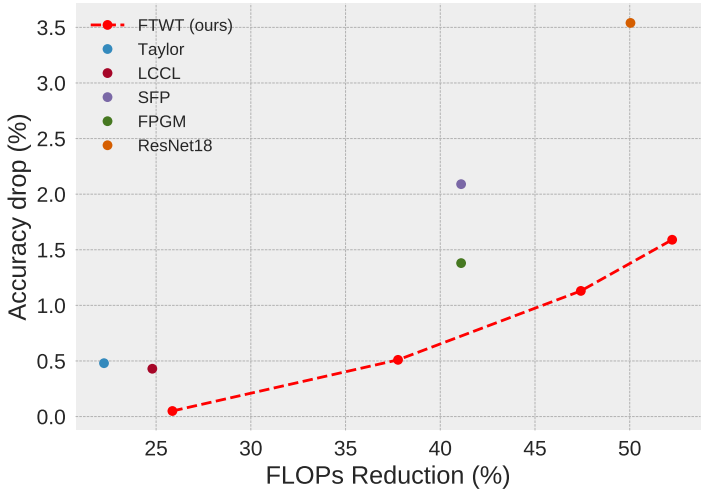
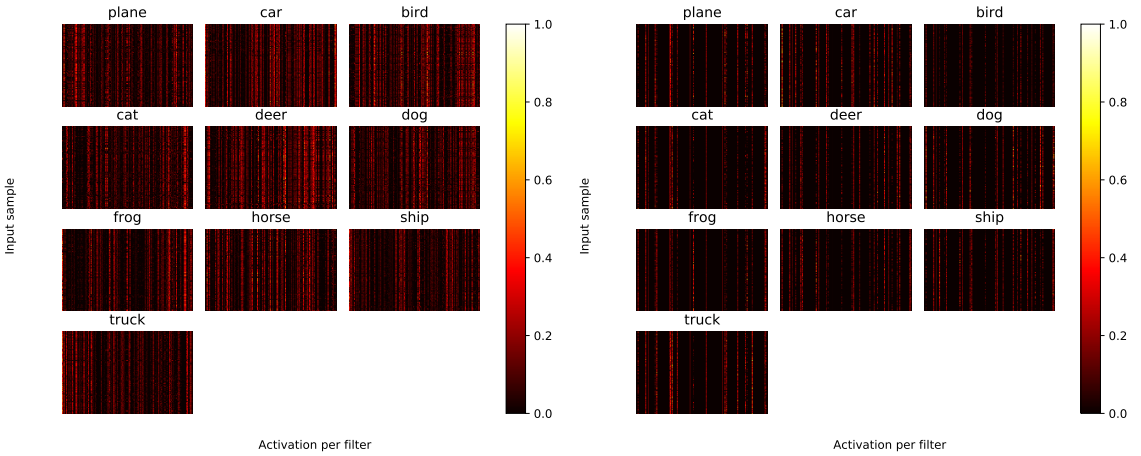


Figure 6.1: FLOPs reduction vs accuracy drop from baselines for various dynamic and static models on ResNet34 ImageNet.

hyperparameter selection lacks transparent relation between sparsity weight and the reached FLOPs; thus, hinders practical efficient training with many iterations of trial and error to achieve a target FLOPs reduction.

In our method, we tackle these issues by formulating the problem as a self-supervised binary classification task. We generate the binary mask of the current layer (wiring) based on the activation (firing) of the previous layer. We draw inspiration from the Hebbian theory [102] in Neuroscience with a twist that we enforce this wiring-firing relation instead of a study of causation as in the theory. Figure 6.2 displays the maximum response for each filter (x-axis) of the last convolutional layer and a middle layer of MobileNet-V1 for random input samples (y-axis) grouped by their class. The plot shows that samples that belong to the same class tend to activate the same combination of filters and thus we only need to process a handful of the filters. Our method relies on the predictability assumption that given the response of the layer's input, we can predict the top-k activation of the output. It is worth noting that the number of clusters varies per layer. Similar to other dynamic pruning methods, we learn a decision head for channel gating. However, we learn the gating using binary



(a) Last convolutional layer (b) 8th depthwise convolutional layer

Figure 6.2: Maximum activations in all features at the last convolutional layer and a middle layer in mobilenetv1 CIFAR-10. Each row in a subplot represents an input sample. Samples that belong to the same class activate the same group of filters. Better visualized in color.

cross-entropy loss per channel. Each layer predicts the filters which are most likely to be highly activated given the layer's input activations. We generate ground truth binary masks per layer based on the mass of the heatmap per sample. This formulation provides advantages in two aspects. First, the channel gating loss implicitly complies and adapts to the backbone's status which stabilizes training in comparison to the case with sparsity regularization or RL-based training. Second, reduction in FLOPs can be estimated before training, as the target mask is controlled by the generated ground truth mask which gives an estimate on the reduction. This simplifies the hyperparameter selection that controls the pruning ratio. The main contributions are summarized as follows:

- A novel loss formulation with self-supervised ground truth mask generation that is stochastic gradient descent (SGD) friendly with no gradient weighting tricks.
- We propose a novel dynamic signature based on the heatmap mass without a pre-defined pruning ratio per layer.
- Simple hyperparameter selection that enables FLOPs reduction estimation be-

fore training. This simplifies realizing a prior budget target with bounded hyperparameter search space.

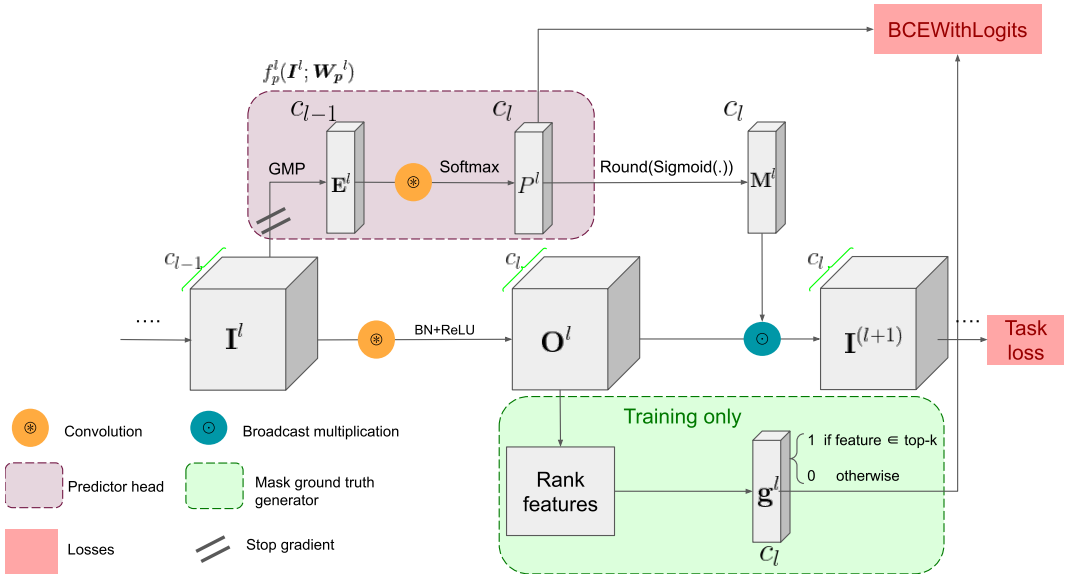


Figure 6.3: Proposed pipeline for training dynamic routing for one layer. For a layer l, prediction head $f_p^l(\boldsymbol{I}^l; \boldsymbol{W_p}^l)$ takes an input $\boldsymbol{I}^l$, applies global max pooling (GMP), normalizes with Softmax, then feeds to 1x1 convolution to generate logits $\boldsymbol{P^l}$ for the binary mask $\boldsymbol{M}^l$. Binary Cross Entropy (BCEWithLogits) loss penalizes the mask prediction based on the top-k obtained from the unpruned feature maps $\boldsymbol{O}^l$.

6.2.2 Channel Gating

Let $\boldsymbol{I^l}$, $\boldsymbol{W^l}$ be the input features, and weights of a convolution layer l, respectively, where $\boldsymbol{I^l} \in \mathbb{R}^{c_{l-1} \times w_l \times h_l}$, $\boldsymbol{W^l} \in \mathbb{R}^{c_l \times c_{l-1} \times k_l \times k_l}$, and c_l is the number of filters in layer l. A typical CNN block consists of a convolution operation ($*$), batch normalization (BN), and an activation function (f) such as the commonly used ReLU. Without loss of generality, we ignore the bias term because of BN inclusion, thus, the output feature map $\boldsymbol{O^l}$ can be written as $\boldsymbol{O^l} = f(\text{BN}(\boldsymbol{I^l} * \boldsymbol{W^l}))$. We predict a binary mask $\boldsymbol{M^l} \in \mathbb{R}^{c_l}$ denoting the highly activated output feature maps $\boldsymbol{O^l}$ from the input activation map $\boldsymbol{I^l}$ by applying a decision head f_p^l with learnable parameters $\boldsymbol{W_p^l}$. Masked output $\boldsymbol{I^{l+1}}$ is then represented as $\boldsymbol{I^{l+1}} = \boldsymbol{O^l} \odot Binarize(f_p^l(\boldsymbol{I^l}; \boldsymbol{W_p}))$. $Binarize(.)$ function

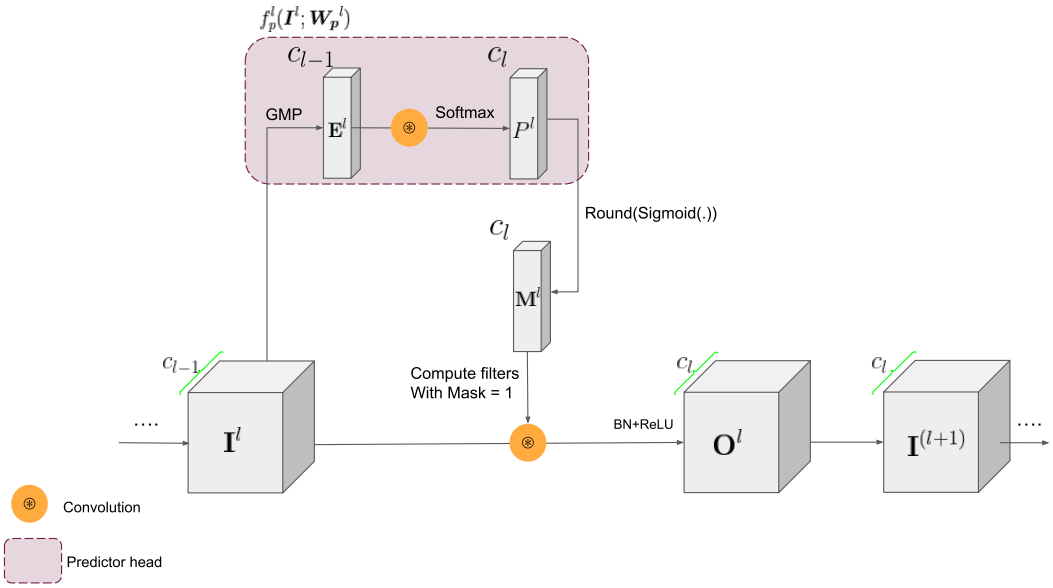


Figure 6.4: Proposed pipeline in testing time. For each layer, only filters with mask prediction=1 are selected and computed while the rest is pruned.

is $round(Sigmoid(.))$ to convert logits to a binary mask. The prediction of the highly activated output feature maps allows for processing filters f where $M_f^l = 1$ in the inference time and skipping the rest.

6.2.3 Self-Supervised Binary Gating

Our proposed method as shown in Figure 6.4 learns this dynamic routing in a self-supervised way by inserting a predictor head after each convolutional block to predict the highly k activated filters of the next layer. The k value is automatically calculated per input based on the mass of heatmap.

Loss function The ground truth binary mask of the highly activated features is attainable by sorting the norm of the features. The overall training objective is:

$$\begin{aligned} \min_{\{\boldsymbol{W}, \boldsymbol{W_p}\}} L_{total} =& L_{ent}(f_n(\boldsymbol{x}; \boldsymbol{W}), \boldsymbol{y}_k) \\ &+ L_{pred}(\{f_p^l(\boldsymbol{I}^l; \boldsymbol{W_p}^l), \boldsymbol{g}^l\}_L) \end{aligned} \tag{6.1}$$

where f_n is the backbone of the baseline model, L_{ent} is the cross-entropy task loss,

L_{pred} is the total predictor loss for all layers $l \in 1...L$. In details, we define L_{pred} as follows:

$$L_{pred}(\{\boldsymbol{P}^l, \boldsymbol{g}^l\}_L) = \sum_{l}^{L} \sum_{f}^{F_l} BCEWithLogits(\boldsymbol{P}^l_f, \boldsymbol{g}^l_f) \tag{6.2}$$

where $\boldsymbol{P}^l$ is the output of the decision head $f^l_p(\boldsymbol{I}^l; \boldsymbol{W_p}^l)$, $\boldsymbol{g^l}$ is the generated ground truth mask based on the top-k highly activated output $\boldsymbol{O^l}$, $BCEWithLogits$ is a $Sigmoid(\sigma)$ followed by the binary cross entropy loss:

$$BCEWithLogits(p, g) = -\left[g \cdot \log \sigma(p) + (1 - g) \cdot \log(1 - \sigma(p))\right] \tag{6.3}$$

Algorithm 1 Binary mask ground truth generation

Input: $I^1 \dots I^L$, r
Output: g binary ground truth with 0 as to prune

```
gt ← ones(L, c_l)
for l ← 1 to L do
    acts ← GMP((|O^l|))
    normalized ← acts / Σ acts
    sorted, idx ← SORT(normalized, "descend")
    cumulative ← CUMSUM(sorted)
    prune_idx ← WHERE(cumulative > r)
    gt[l][prune_idx] ← 0
end for
```

Number of activated k We automatically calculate k by keeping a constant percentage r of the mass of heatmap. For each channel $i = 1, ..., c_l$, we keep the maximum response by applying a global maximum pooling (GMP) (GMP($\left|O^l_i\right|$). For each input example, k is the number of filters kept such that the cumulative mass of the sorted heatmaps reaches $r\%$. The ground truth generation algorithm is shown in Algorithm 1. We use the same r for all layers, however, each sample will have a different pruning ratio per layer based on its activation. As the target binary ground-truth is generated from the activations of the unmasked filters, FLOPs reduction can be loosely

estimated prior to training to adjust r accordingly. This advantage adds to the practicality of our method which does not rely on indirect hyperparameter tuning to reach a budget target in FLOPs reduction. It is also worth mentioning that $r = 1$ is a special case that indicates the decision head will predict the completely deactivated features (e.g maximum response is zero). This enables maintaining the accuracy of baseline with ideally trained decision heads for highly sparse backbones. A visual representation for binary mask generation is shown in Figure 6.5

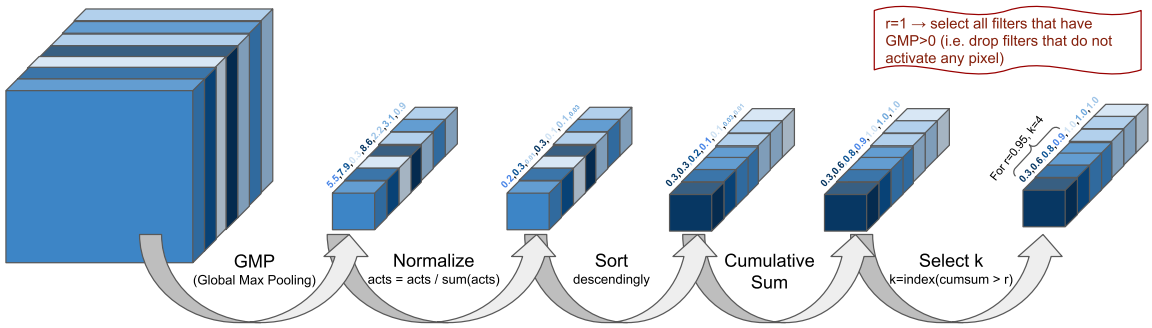


Figure 6.5: Binary mask ground truth generation.

6.2.4 Prediction Head Design

Prediction head design should be modeled in a simple way to reduce overhead over the baseline network. In forward pass, we apply GMP that reduces feature map $\mathbf{I^l}$ per layer to $E^l \in \mathbb{R}^{c^{l-1}\times 1\times 1}$. Next, we apply 1x1 convolution on the flattened embedding E^l to produce the mask's logits. We experiment with two training modes: 1) decoupled, and 2) joint. In both modes, we train backbone weights and those of decision heads in parallel. The distinction is whether we do a fully differentiable training (joint) or stop gradients from heads to backpropagate to the backbone and vice versa (decoupled). In joint training, the decision head is fully differentiable except at the binarization part. Similar to previous works [3, 103, 104], we utilize straight-through estimator (STE) to bypass the non-differentiable function. An issue to consider is loss interference with multiple losses at different depths in the network

as pointed out in [10]. Losses interference indicates that layers can be biased towards achieving high accuracy to local tasks more than the overall architecture. Unlike other methods that rely on careful training tuning to manage gradients from different losses, we train the heads along with the backbone in parallel yet collaboratively as the masks are generated from the current status of the model. The ground-truth binary masks are explicitly adjusted by the updated backbone weights, thus, implicitly complying with the backbone learning speed.

6.3 Experiments and Analysis

We evaluate our method on CIFAR [105] and ImageNet [106] datasets on a variety of architectures such as VGG [107], ResNet [108] and MobileNet [7]. In all architectures, ground truth masks are generated after each conv-BN-ReLU block. For training the baseline dense models, we follow the same setup in [108]. For CIFAR models, we train for 200 epochs using a batch size of 128 with SGD optimizer. The initial learning rate (lr) 0.1 is divided by 10 at epochs 80, 120, and 150. We use a momentum of 0.9 with a weight decay of 5^{-4}. For ImageNet, we use the pre-trained models in PyTorch [109] as baselines. Weights of decision heads are updated with an initial learning rate of 0.1 and the same learning rate schedule as the backbone. We use a 4 V100-GPU machine in our experiments.

6.3.1 Experiments on CIFAR

We follow similar training settings used in baseline for dynamic training, but we train all models with initial learning rate of $1e^{-2}$. We report the average accuracy over three repeated experiments and FLOPs reduction on CIFAR-10 on multiple architectures in Table 6.2. Our method (FTWT) achieves higher FLOPs reduction on similar top1-accuracy than static and dynamic pruning methods. We achieve up to 66% FLOPs reduction on VGG-16 and ResNet-56, that is higher than dynamic filter pruning methods RNP [26], FBS[73], LCS [74] by up to 15%. Joint training performs

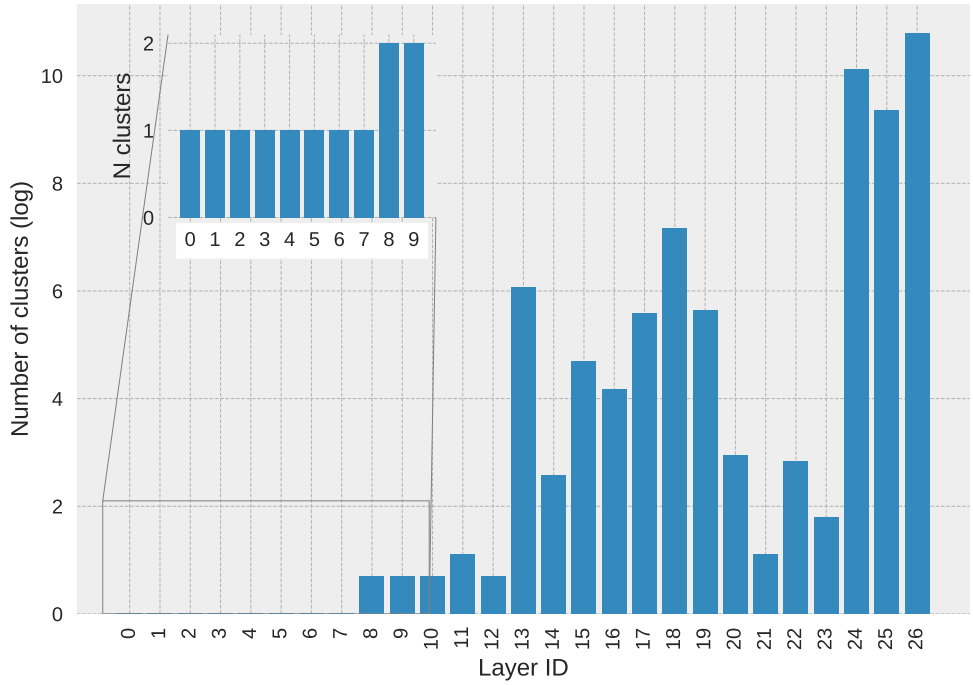
	Model	Dynamic?	Top-1 Acc. (%)	FLOPs red. (%)
VGG16-BN	Baseline	–	93.82	—
	L1-norm [80]	N	93.00	34
	ThiNet [62]	N	93.36	50
	CP [17]	N	93.18	50
	Taylor-50 [60]	N	92.00	51
	RNP [26]	Y	92.65	50
	FBS [73]	Y	93.03	50
	LCS [74]	Y	93.45	50
	FTWT$_{\mathbf{J}}$ ($r = 0.92$)	Y	93.55	**65**
	FTWT$_{\mathbf{D}}$ ($r = 0.92$)	Y	**93.73**	56
	Taylor-59 [60]	N	91.50	59
	FTWT$_{\mathbf{D}}$ ($r = 0.85$)	Y	**93.19**	73
	FTWT$_{\mathbf{J}}$ ($r = 0.88$)	Y	92.65	**74**
ResNet56	Baseline	–	93.66	–
	Uniform from [74]	N	74.39	50
	ThiNet [62]	N	91.98	50
	SFP [110]	N	92.56	48
	LCS [74]	Y	92.57	52
	FTWT$_{\mathbf{D}}$ ($r = 0.80$)	Y	**92.63**	**66**
	FTWT$_{\mathbf{J}}$ ($r = 0.88$)	Y	92.28	54
MobileNetV1	Baseline	–	90.89	–
	MobileNet_75 [7]	N	89.79	42
	MobileNet_50 [7]	N	87.58	73
	FTWT$_{\mathbf{D}}$ ($r = 1.0$)	Y	91.06	**78**
	FTWT$_{\mathbf{J}}$ ($r = 1.0$)	Y	**91.21**	**78**

Table 6.2: Results on CIFAR-10. FLOPs red. indicates reduction in FLOPs in percentage. r in our method states the hyperparameter ratio in Algorithm 1. $\mathbf{x}$ in FTWT$_{\mathbf{x}}$ indicates joint (J) or decoupled (D) training.

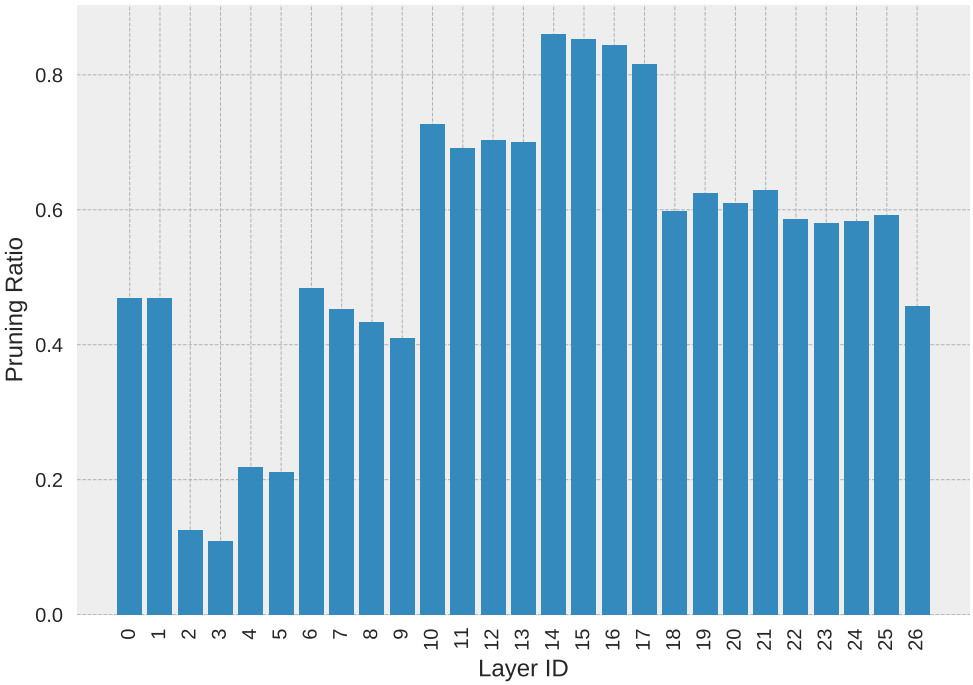
equally well as decoupled training on high r thresholds. However, the accuracy drops in comparison to decoupled training on lower thresholds. That is due to conflict

increase between losses as can be seen on $\approx$ 73% FLOPs reduction on VGG. We further achieve a 73% FLOPs reduction on VGG with only a 0.63% accuracy drop. Moreover, FTWT outperforms smaller variants of MobileNet in accuracy by 3.42% with higher FLOPs reduction.

Visualization We visualize the number of the unique combinations of filters (clusters) that are activated over the whole dataset D and the pruning ratio per layer in Figure 6.6. Meaning that, each sample i that produces a binary mask $m_{i,j}$ per layer j, the unique clusters per layer is $set(m_{0,j}, ..., m_{i,j}, ...m_{|D|,j})$. In LCS and RNP, a fixed number of clusters is preset as a hyperparameter for all layers, we show in Figure 6.6a that layers differ in the number of diverse clusters. Our method adjusts the different number of clusters per layer automatically due to the self-supervised mask generation mechanism. For easier visualization, the y-axis is shown on a log scale. Early layers have small diversity in the group of filters activated, thus, acting similar to static pruning. This is sensible as early layers detect low-level features and have less dependency on the input. On the other hand, the number of clusters increases as we go deeper into the network. It is worth mentioning that these different clusters are fine-grained, which means clusters can differ in one filter only. We also calculated the percentage of core filters that are shared among all clusters per layer. We found that the range of the percentage of core filters with respect to total filters varies from 40% to 100%. This gives insight into why static pruning methods result in a large drop in accuracy with large pruning. As the attainable pruning ratio is limited by the number of core filters and further pruning will limit the model's capacity. An interesting future research question would be if we can determine the compressibility of a model based on the core filters ratio notion. Finally, Figure 6.6b shows the pruning ratio per layer, as to be expected, the later layers are more heavily pruned than early layers as layers get wider and more compressible. We notice heavy pruning reaching 85% in the middle layers with a sequence of layers with 512 filters.



(a) Number of unique group of filters (clusters) per layer.



(b) Pruning ratio per layer.

Figure 6.6: MobileNetV1 CIFAR10 distributions

We visualize the heatmap of a highly pruned layer in comparison to the baseline model. Figure 6.7 shows the comparison between the heatmap from the baseline with all filters activated and the heatmap of dynamically selected filters. As can be seen, dynamic pruning approximates the baseline with high attention to foreground objects. This shows that even with a 70% pruning ratio in that layer, we are able to approximate the behavior of the original model.

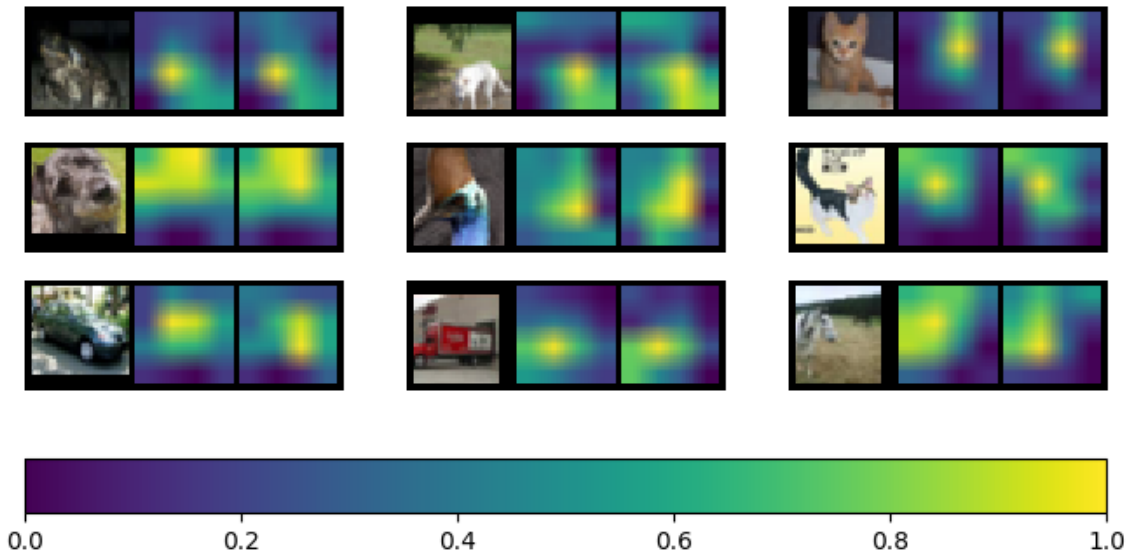


Figure 6.7: Heatmap visualization of random input samples from CIFAR for the 10th layer in MobileNetV1. Each triplet represents input image, baseline heatmap, pruned heatmap. FLOPs reduction in the layer is $\approx$ 70%, yet the pruned heatmap highly approximate the heatmap with fully activated filters.

Different pruning ratio We compare our method on MobileNetv1/v2 under different pruning ratio with other pruning methods such as EagleEye [111], SCOP [112] and DCP [59]. Note that EagleEye reports the best out of two candidate models different in signature, thus double the training time. We outperform SOTA by 37% higher FLOPs reduction on similar accuracy as shown in Figure 6.8.

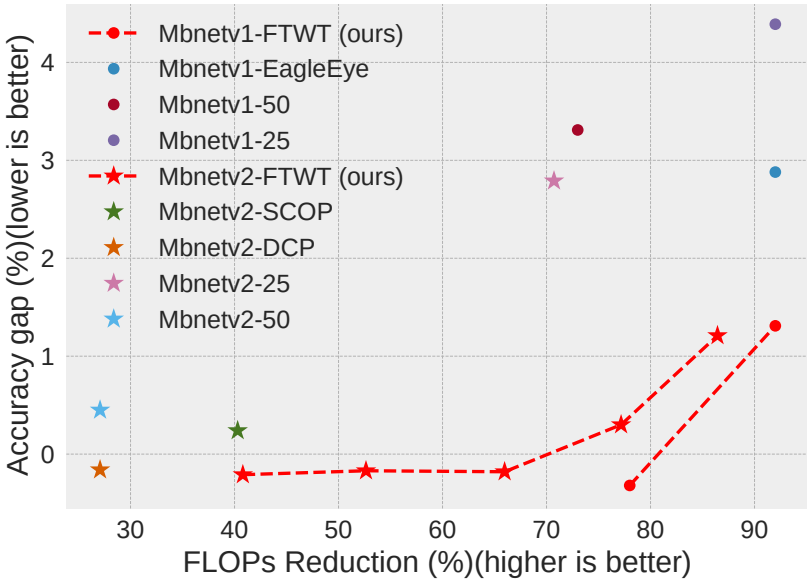


Figure 6.8: MobileNetV1/V2 on CIFAR10.

6.3.2 Experiments on ImageNet

For ImageNet, we train for 90 epochs with an initial learning rate of 10^{-2} that decays every 30 epochs by 0.1. Experiments on ImageNet are done with the decoupled training mode. Table 6.3 shows a drop in accuracy from baseline for each method to account for training differences due to augmentations. Results show that our method achieves a smaller drop in accuracy with higher FLOPs reduction in comparison to other SOTA methods. We achieve a similar accuracy reduction as LCCL with 13% higher FLOPs reduction on ResNet34. On the other hand, on similar FLOPs reduction ($\approx$ 25), we have a minimal drop in accuracy ($\approx$ 0.05%). We achieve a higher compression rate on ResNet18 with similar accuracy as FBS which uses a predefined number of filters per layer. This shows the effectiveness of our dynamic signature using the ratio of heatmap mass. We also compare with architecture's smaller variants such as ResNet18 and MobileNet-75. We outperform ResNet18 and MobileNet-75 by $\approx$ 2% in accuracy on a similar computation budget.

	Method	Dynamic?	Top-1 Acc. (%)			FLOPs red. (%)
			Baseline	Pruned	Delta	
ResNet34	Taylor [60]	N	73.31	72.83	0.48	22.25
	LCCL [99]	Y	73.42	72.99	0.43	24.80
	FTWT ($r = 0.97$)	Y	73.30	73.25	**0.05**	25.86
	FTWT ($r = 0.95$)	Y	73.30	72.79	0.51	37.77
	SFP [110]	N	73.92	71.83	2.09	41.10
	FPGM [113]	N	73.92	72.54	1.38	41.10
	FTWT ($r = 0.93$)	Y	73.30	72.17	**1.13**	47.42
	ResNet18 [108]	N	73.30	69.76	3.54	50.04
	FTWT ($r = 0.92$)	Y	73.30	71.71	**1.59**	52.24
ResNet18	PFP-B [114]	N	69.74	65.65	4.09	43.12
	SFP [110]	N	70.28	67.10	3.18	41.80
	LCCL [99]	Y	69.98	66.33	3.65	34.60
	FBS [73]	Y	70.70	68.20	2.50	49.49
	FTWT ($r = 0.91$)	Y	69.76	67.49	**2.27**	51.56
MobileNetV1	MobileNetV1-75 [7]	N	69.76	67.00	2.76	42.85
	FTWT ($r = 1$)	Y	69.57	69.66	**-0.09**	41.07

Table 6.3: Results on ImageNet. Baseline accuracy for each method is reported along with the pruned model's accuracy and accuracy change from baseline. FLOPs red. represents reduction in FLOPs in percentage. Negative delta indicates increase in accuracy from the baseline. r in our method states the hyperparameter ratio in Algorithm 1

6.3.3 Ablation Study

Uncertainty under Dataset Shift

In this section, we measure the sensitivity of our routing to dataset shift. Metrics under dataset shift are rarely inspected in model pruning literature. We believe in its importance as inference complexity increases and thus would like to initiate reporting such comparisons. We conduct experiments for VGG16-bn CIFAR-10 with a high pruning ratio of 73% for all pruned models. Inspired by [115], we report

Brier score [116] under different type of noise such as Gaussian blur Table 6.4a and additive noise Table 6.4b with baseline dense model as a reference. As can be seen, our method is more resilient than static Taylor pruning with lower brier scores. We also compare with static uniform pruning, we achieve a similar (sometimes slightly lower) Brier score. This shows the resilience of our model to data shift even when compared with static pruning decision that is not data-dependent. Finally, as to be expected, the dense model is the most resilient to noise. However, our method still shows a fair quality matching overall. We attribute this distribution stability to the softmax in the head. The softmax acts as a normalizer which reduces sensitivity to distribution shift. We compare our method with and without softmax normalization in the decision head to verify this hypothesis. Table 6.4c shows Brier scores with additive and blurring noise for this comparison. As can be seen, indeed, the normalizer stabilizes the decision masks output, especially in the case of blurring. We also tried

σ	Dense model	FTWT (ours)	Taylor Pruning	Uniform Pruning
0.5	0.11	**0.12**	0.20	**0.12**
0.7	0.16	**0.18**	0.39	0.19
0.9	0.38	**0.39**	0.57	0.42
1.09	0.69	**0.58**	0.66	0.61
1.27	0.74	**0.68**	0.69	0.71
1.45	0.76	**0.73**	0.74	0.75

(a) Gaussian blur noise.

σ	Dense model	FTWT (ours)	Taylor Pruning	Uniform Pruning
0.00	0.11	**0.12**	0.16	**0.12**
0.02	0.11	**0.12**	0.16	**0.12**
0.05	0.11	**0.12**	0.17	0.13
0.11	0.13	**0.14**	0.20	0.15
0.14	0.19	**0.21**	0.30	0.22
0.20	0.39	0.43	0.51	**0.42**

(b) Additive Gaussian noise.

Gaussian Blur		Additive Noise	
FTWT with normalization	FTWT without normalization	FTWT with normalization	FTWT without normalization
0.12	0.19	**0.12**	0.12
0.18	0.48	**0.12**	0.13
0.39	0.73	**0.14**	0.15
0.58	0.85	**0.21**	0.24
0.68	0.90	**0.43**	0.47

(c) Our method with and without softmax normalization in decision heads.

Table 6.4: Dataset shift experiments: Numbers represent Brier score on CIFAR-10 VGG16

to compare with other publicly available methods such as BN scalars norm [61] and weights norm [53] but the pruned network collapsed (i.e many layers collapsed to 1 filter) when trying to achieve a high pruning ratio.

Dynamic Signature and Dynamic Routing

We investigate decoupling the effect from the dynamic signature (i.e. pruning ratio per layer) per sample from the dynamic routing (i.e group of filters to be activated). We explore the effectiveness of dynamic routing with a pre-defined signature for all inputs. In these experiments, the signature is pre-defined using Taylor criteria proposed in [60] as a case study. As in the previous setup, we select the highly activated k features where k is defined by the signature while samples differ in which k filters are selected. Table 6.5 shows results of dynamic routing under different pruning ratios. As can be seen, dynamic routing performs better than static inference especially on high pruning ratio by up to 4%. Training setup is the same for CIFAR as explained in the paper, however, we train ImageNet models for 30 epochs using finetune setup instead of training setup with 90 epochs as differentiated in [20] to accelerate the experiment.

Dataset	Model	FLOPs (%)	Top-1 acc. (%) Static	Dynamic
CIFAR-10	VGG16-BN	50	92.00	**93.80**
		85	91.12	**92.75**
	ResNet56	70	91.61	**92.09**
CIFAR-100	VGG16-BN	30	72.65	**73.67**
		65	68.17	**72.18**
		93	58.74	**60.05**
ImageNet	ResNet18	45	64.89	**65.11**

Table 6.5: Accuracy comparison of dynamic routing with a pre-defined signature and its counterpart with static inference.

Model	Est. FLOPs (%)	Final FLOPs (%)
$\text{MobileNet}_{(1.0)}$	42.3	41.07
$\text{Resnet34}_{(0.97)}$	23.32	25.86
$\text{Resnet34}_{(0.95)}$	31.77	37.77

Table 6.6: Estimated FLOPs *before* training under different thresholds (indicated in parentheses) vs final FLOPs achieved after training.

Hyperparameter r selection

The hyperparameter r (i.e mass ratio) is selected based on a simple evaluation before training. We freeze the dense model and obtain the expected FLOPs reduction using the generated ground truth on the training data (one-shot pass). Subsequently, we can get an estimate of the FLOPs reduction under different r values before the training is initiated. This simplifies hyperparameter selection to achieve a target FLOPs reduction. On the other hand, a sparse regularization hyperparameter is usually fine-tuned with a cross-validation process and requires a trial and error of multiple full training to achieve a target budget. There is no direct relation between the regularization weight and the final achieved FLOPs reduction knowingly before training. Our method simplifies the selection and makes it a more practical option when a target budget is given as a prior. Table 6.6 shows the estimated FLOPs before training using different thresholds, r, and the actual reached FLOPs reduction after training. The difference in reduction is due to the inaccuracy of the decision heads. Nonetheless, the estimated FLOPs give a good approximation to the final reached FLOPs and thus reduce the hyperparameter search.

6.3.4 Theoretical vs Practical Speedup

For all compression methods, including static and dynamic pruning, there is often a wide gap between FLOPs reduction and realistic speedup due to other factors such

Model	FLOPs reduction (%)	Latency reduction (%)
ResNet34	52.18	27.17
	37.77	19.78
	25.86	11.08

Table 6.7: Realistic speedup vs theoretical speedup on ImageNet on AMD Ryzen Threadripper 2970WX CPU with batch size of 1 on a single thread.

as I/O delays and BLAS libraries. Speedup is hardware and backend dependent as shown in prior works [21, 22, 117]. We test the realistic speed on PyTorch [109] using MKL backend on AMD CPU using a single thread. The results are shown in Table 6.7.

Limitations. Our realistic speedup is less than FLOPs reduction and that is attributed to two factors: 1) Data transfer overhead from slicing the dense weight matrix based on the mask prediction, which can be mitigated by backends with efficient in-place sparse inference. 2) Speed up is dependent on the model's signature and hardware's specs. Pruning from later layers that process smaller input resolution might not achieve as much speedup as pruning from early layers. Constraint aware optimization using Alternating Direction Method of Multipliers (ADMM) [118] such as proposed in [119] can be further integrated with our method to optimizing over latency instead of FLOPs.

6.4 Conclusion

In this paper, we propose a novel formulation for dynamic model pruning. Similar to other dynamic pruning methods, we equip a cheap decision head to the original convolutional layer. However, we propose to train the decision heads in a self-supervised paradigm. This head predicts the most likely to be highly activated filters given the layer's input activation. The masks are trained using a binary cross-entropy loss de-

coupled from the task loss to remove loss interference. We generate the mask ground truth based on a novel criterion using the heatmap mass per input sample. In our experiments, we showed results on various architectures on CIFAR and ImageNet datasets, and our approach outperforms other dynamic and static pruning methods under similar FLOPs reduction.

Broader Impact

Neural Network pruning has the potential to increase deployment efficiency in terms of energy and response time. However, obtaining these pruned models is yet to be optimized for a better overall computational consumption and more environment-friendly. Moreover, pruning requires careful understanding of deployment scenarios such as questioning out-of-distribution generalization [120] or altering the behavior of networks in unfair ways [121]. We showed some results on the out-of-distribution shift to tackle the first part. We did not investigate the fairness of the model's prediction as both datasets (i.e CIFAR and ImageNet) are balanced. Although, we prune based on the mass of the heatmap equally for all samples. We hypothesize this trait can give our method an advantage over the fixed pruning ratio which might hurt some input samples over some others.

Chapter 7

Query Efficient and Self Knowledge Distillation

7.1 Motivation

In previous chapters, we tackled model pruning to generate a lightweight model for inference time. In this chapter, we wanted to explore training acceleration for the distillation training paradigm. We adopt vision transformers as a use case to accelerate its training. Transformer models have been widely used in natural language processing (NLP) tasks. Recently, transformers are gaining momentum to be adopted in computer vision tasks such as image classification [78, 122, 123] and object detection [78]. Convolutional neural networks (CNN) have been the fundamental off-the-shelf models for vision tasks. As noted in [122], vision transformers relax the inductive bias attached to each layer in convolution networks such as locality, 2D neighborhood, and translation equivalence. These inductive biases are useful for smaller datasets but can limit accuracy gain when large datasets are available. This reliance on large (i.e 90+M) datasets for training visual transformers limits its application on limited and medium-sized datasets.

DeiT [78] proposes a teacher-student distillation loss to benefit from pre-trained CNN models as teachers for data-efficient training. Knowledge distillation aims to approximate the student's output to that of the teacher. This requires training a CNN teacher model on the dataset of interest and querying the teacher in each iteration

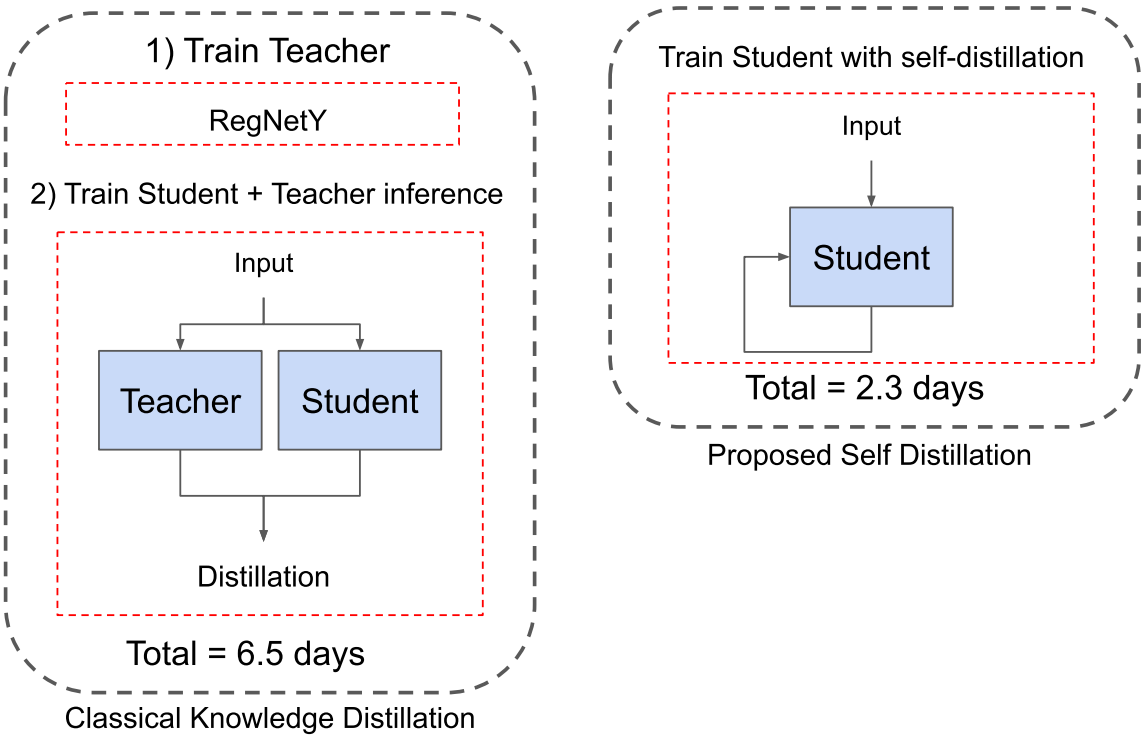


Figure 7.1: Comparison between teacher-student knowledge distillation and proposed self-distillation for transformers. Training time is reduced by 2.8x factor. Baseline without any distillation takes 2.1 days.

in the student's training phase to match their outputs. Thus, this strategy is time-consuming and memory intensive. In addition, we observed that teacher-student distillation achieves up to 2% improvement over baselines. So, only a relatively small amount of data benefits from the teacher-student query. This observation motivates us to explore query efficient training where we approximate the teacher's output for a handful of inputs only. We address these issues by a two-fold contribution with 1) self-distillation loss and 2) optional query efficient teacher-study distillation loss. First, we propose a self-distillation training in which early layers mimic the output of the final layer within the same model. This achieves 2.8x speedup in comparison to teacher-student distillation with matched accuracy as shown in Figure 7.1. We also propose a simple yet effective query-efficient distillation in case a trained teacher is available for further accuracy boost. We query the teacher model (CNN) only when the student (transformer) fails to predict the correct output. This simple criterion

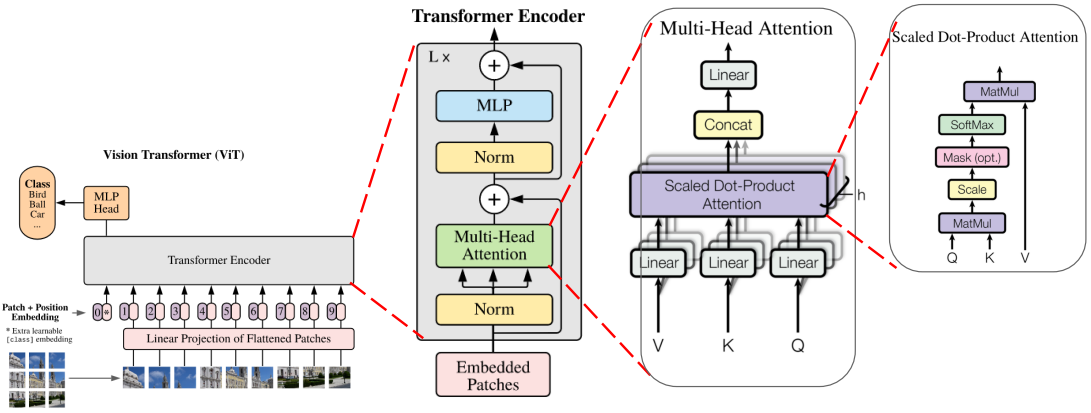


Figure 7.2: ViT architecture [122].

not only saves computational resources but also achieves higher accuracy than a full query teacher-student. As training progresses, the student's accuracy improves and fewer inference calls to the teacher are made.

Key points: Unlike previous chapters, we aim to accelerate the training instead of a lightweight model for efficient inference. We take vision transformers as a use case given that it is more data hungry than CNN. We propose to adopt self-distillation to increase the discrimination ability of early layers by mimicking the model's prediction by the final layer. This illuminates the need for CNN teacher inference as used in DeiT for data-efficient training. In case of a trained teacher availability, we also explore input sampling to reduce the number of query calls to the teacher model. A typical student-teacher distillation feeds all input samples to both student and teacher which increase the memory consumption and training time.

7.2 Proposed Method

7.2.1 Preliminaries

Vision Transformers. ViT [122] is the first application of transformers on large-scale vision tasks. Although ViT shows promise of full-transformer architecture for vision tasks, its performance is still inferior on small and medium-sized datasets such as ImageNet [106]. Many variants followed with different architecture building blocks

such as [78, 123, 124]. In T2T [123], the authors borrow from the CNN architecture and propose a mixed CNN-transformer architecture. Instead of a linear projection for tokenization, CNN is adopted in the early stage for tokenization. Swin [124] proposes a hierarchical transformer for multi-scale processing. On the other hand, DeiT [78] introduces several training strategies to improve [122] for data-efficient training. DeiT [78] applies knowledge distillation (KD) [79] by adding a KD token distillation that matches the output of a CNN pre-trained teacher. However, this method requires us to pre-train a teacher model on the task that might not be readily available in many practical cases. Besides, during training, KD feeds each batch to the teacher and the student to match their output. This can be memory-demanding and time-consuming. Without losing generality, in any vision transformer, the input images are divided into a sequence of patches P that are fed into a token extraction module. This tokenization can be as simple as a linear projection with a fully-connected layer such as in [78, 122] or a CNN architecture such as in [123]. A transformer network is then applied for relationship modeling and feature aggregation. A transformer block usually consists of a multi-head self-attention layer (MSA) and an MLP block. Layernorm (LN) is applied before each layer and residual connections in both the self-attention layer and MLP block. Figure 7.2 lays out the components of ViT [122] as an example. Our training treats transformer block as a black box and can be utilized with different backbone designs.

Knowledge Distillation. Model distillation is one of the most popular techniques to boost lightweight models' accuracy for model compression. One of the earliest work [79] proposes to match the prediction distribution between a pre-trained model and the student. A plethora of distillation loss is further studied in CNN [125–127]. A recent direction is to adopt a self-distillation training on CNN to improve the discriminative ability of early layers [77, 128]. Motivated by this direction, we explore self-distillation on vision transformers.

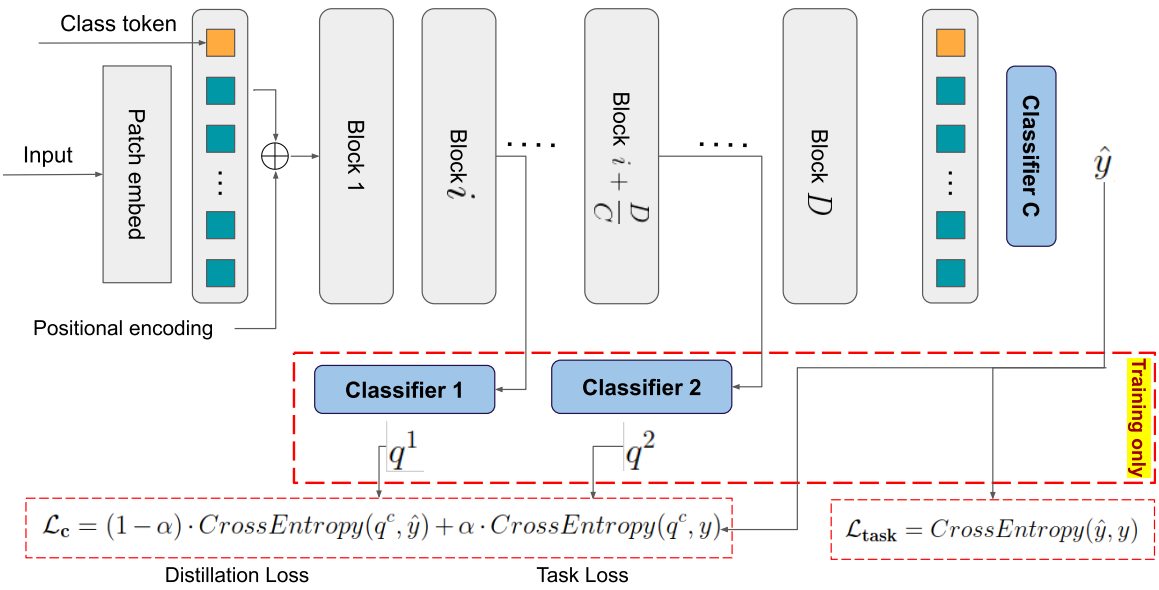


Figure 7.3: Proposed self-distillation loss. Given a transformer model with depth D, C classifiers are inserted each $\frac{D}{C}$ blocks. Each auxiliary classifier outputs a probability distribution q^c. The auxiliary classifiers are trained using a weighted sum of distillation loss and task loss with ground truth y. Final classifier is trained using the task loss only.

7.2.2 Self-distillation

Given a model with depth D, we split the model into C parts. We insert a lightweight classifier Θ_c after each split. Each classifier is defined by its own parameters, so that, given N examples $X = \{x_i\}_{i=1}^N$, the logits output for a classifier Θ_c is $z_i^c = \Theta_c(x_i^c)$. A softmax layer is set after each classifier to produce $\mathbb{M}$ class probability $q_i^c \in \mathbb{R}^{\mathbb{M}}$:

$$q_i^c = \frac{exp(z_i^c/T)}{\sum_j^c exp(z_i^c/T)} \tag{7.1}$$

where T is a temperature that is normally set to 1.

We train each classifier Θ_c with a weighted sum of self-distillation and cross-entropy loss with true labels.

$$\mathcal{L}_c = (1-\alpha) \cdot CrossEntropy(q^c, \hat{y}) + \alpha \cdot CrossEntropy(q^c, y) \tag{7.2}$$

where $\hat{y}$ is the transformer's final prediction, y is the true labels and α is a weight factor that starts with 1 and gradually decreases to 0. The motivation for the grad-

ual weight decrease is that when training with only the task loss, the layers tend to greedily optimize for their local classifiers. This results in gradient interference from different classifier heads and overall suboptimal performance. By starting with a higher weight for the task loss, we benefit from faster convergence. As we gradually increase the weight towards distillation loss, we promote collaborative training between classifier heads, because the distillation loss follows the classifier output by the final layer. Figure 7.3 shows the full pipeline.

7.2.3 Query-efficient Distillation

In addition to self-distillation, we also propose an optional query-efficient distillation loss. One of the issues of teacher-student distillation is pretraining the teacher on the same task as a prerequisite. In some cases, pre-trained models are already available in the public domain. However, feeding all input samples to the teacher alongside the student can be computationally expensive and memory-intensive. We apply a selective teacher-student distillation, in addition to the self-distillation, to further boost the accuracy without compromising the training efficiency. We propose a simple yet effective selection criterion based on the prediction output of the transformer. We query the teacher only if the transformer fails to correctly predict the input. As training progresses, fewer samples are fed to teachers saving up to 70% of inference calls.

We adopt a distillation token $\hat{d}$ similar to DeiT [78], but in contrast, we only update using teacher-student for the misclassified samples $\hat{y} \neq y$. Given a teacher Γ, we calculate the query-efficient distillation loss as:

$$\mathcal{L}_{qe} = CrossEntropy(\hat{d}_{[\hat{y}\neq y]}, \Gamma(x_{[\hat{y}\neq y]})) \tag{7.3}$$

To sum up, the total loss function consists of three parts:

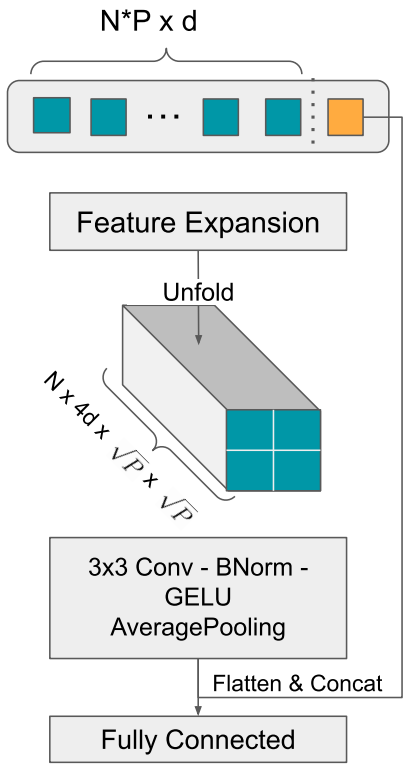


Figure 7.4: Auxiliary classifier design.

$$\mathcal{L} = \sum_{c=1}^{C-1} \mathcal{L}_c + \mathcal{L}_{qe} + CrossEntropy(\hat{y}, y) \tag{7.4}$$

Note that the last classifier is only trained with the task prediction loss.

7.2.4 Auxiliary Classifier Design

We adopt a lightweight classifier design to not comprise training speed. Figure 7.4 shows the classifier design. Given the embedding at the l_{th} block $x_i^l \in \mathbb{R}^{N*P\times d}$, we first expand the features to $4*d$ with linear projection to increase representation. Where N is the number of samples, P is the number of patches and d is the embedding length. Feature expansion is followed by an unfold operation to order the patches into CNN friendly format $x_i^c \in \mathbb{R}^{N\times 4*d\times\sqrt{P}\times\sqrt{P}}$. Finally, a common CNN block with convolution, batch norm, and GELU block is adopted and followed by a classification linear layer. As the input is of small spatial size (e.g $d = 192$ $\sqrt{P} = 16$), the computation overhead

is negligible.

7.3 Experiments

We evaluated our method on various transformer architectures on CIFAR [129] and ImageNet [106] datasets. We train our models with the same training setup for each architecture for a fair comparison. We set the number of classifiers $C = 4$. We use 4 V100 GPUs and train for 300 epochs using AdamW [130] as the optimizer and cosine learning rate decay [131].

	Classifier	Classifier 4/4	Classifier 3/4	Classifier 2/4	Classifier 1/4	Total training time in hours (speedup)
CIFAR-100	Deit-tiny (independent training)	85.46	85.31	**83.33**	75.83	18
	Deit-tiny (Ours)	**86.55**	**85.78**	83.26	**76.72**	**7 (2.5x)**
ImageNet	Deit-tiny (independent training)	72.20	70.43	65.08	55.30	154
	Deit-tiny (Ours)	**73.60**	**71.10**	**67.43**	**57.59**	**55 (2.8x)**

Table 7.1: Independently training classifiers from scratch vs our auxiliary classifiers from one training.

7.3.1 CIFAR

We evaluate on CIFAR on DeiT [78] and T2T [123] and follow their training setup. We finetune the pre-trained ImageNet model on CIFAR and resize the input image to 224x224 to match the ImageNet input size as adopted in [78, 122]. Table 7.2 shows results on CIFAR-10/100. As can be seen, self-distillation consistently improves the baseline accuracy by up to 1%. We also compare the accuracy of the auxiliary classifiers in comparison to training independently custom models. Each model varies in depth with the same number of blocks equivalent to the corresponding auxiliary classifier. Table 7.1 shows that our self-distilled classifiers achieve higher accuracy than if trained independently. Not only do these classifiers have higher accuracy, but also save 2.5x of training time. Our training is able to provide multiple small models in one training. This eases deployment under different inference computational budgets.

Model	Teacher	Accuracy (%)
CIFAR-100		
Deit-tiny [78]	None	85.46
	Self (Ours)	**86.55**
T2T-ViT-7 [123]	None	85.37
	Self (Ours)	**86.12**
CIFAR-10		
Deit-tiny [78]	None	98.10
	Self (Ours)	**98.30**
T2T-ViT-7 [123]	None	97.81
	Self (Ours)	**98.01**

Table 7.2: Evaluation on CIFAR.

7.3.2 ImageNet

We also evaluate our distillation method on the challenging ImageNet task across variants of architectures as presented in Table 7.3. Self-distillation achieves up to 3% gain in accuracy and closes the gap with teacher-student distillation. As self-distillation training does not require teacher pretraining or teacher queries, it saves 2.8x training time. DeiT with knowledge distillation takes 6.5 days, while our self-distillation training reduces the time to 2.3 days. It is worth mentioning that baseline without any distillation takes 2.1 days. This shows that the auxiliary classifiers add negligible overhead. Also, as can be seen, the performance gain from teacher-student training decreases as model size increases. There is only a 0.23% difference between self-distillation and DeiT with a teacher in the case of Deit-small. This shows that self-distillation in some cases is sufficient and computationally efficient. To further boost the accuracy, we show results when query-efficient loss $\mathcal{L}_{qe}$ is added as well. Interestingly, we saw an improvement in accuracy in comparison to a full query distillation. We hypothesize the query selection based on the false prediction emphasizes

Model	Teacher	Accuracy (%)
Deit-tiny [78]	None	72.20
	RegNetY-160	74.50
	Self (ours)	73.60
	Self + QE (Ours)	**74.95**
Deit-small [78]	None	79.90
	RegNetY-160	81.20
	Self (Ours)	80.97
	Self + QE (Ours)	**81.83**
T2T-ViT-7 [123]	None	71.70
	Self (Ours)	**74.60**
T2T-ViT-14 [123]	None	81.85
	Self (Ours)	**81.85**

Table 7.3: Evaluation on ImageNet. Self indicates using proposed $\mathcal{L}_c$. Self + QE indicates using $\mathcal{L}_c + \mathcal{L}_{qe}$.

more distillation weight on the hard to learn samples. Full query distillation consists of many small errors for the correctly classified samples and less weight is given to the incorrect prediction. This can relate to the idea of focal loss [132] proposed to treat imbalance between majority confidently classified samples and minority hard to classify samples. Selection in query-efficient acts as a form of regularization.

7.4 Conclusion

We propose a self-distillation loss for data-efficient transformer training. We insert auxiliary lightweight classifiers at different depths in the backbone. Classifiers are trained using a weighted sum between task loss and distillation loss that matches the prediction of the model's final output. In addition, we propose a query efficient distillation that inference the teacher in a selective manner. We show that our training outperforms the SOTA while being computationally efficient.

Chapter 8

Conclusion and Future Work

In this chapter, we summarize our contribution and the future directions that can be tackled moving forward with our conclusions. Throughout this thesis, we have thoroughly investigated model pruning literature as a neural network model compression technique. We have further touched on knowledge distillation on vision transformers as a use case for training acceleration.

8.1 Summary of Contributions

As a summary of the thesis contributions:

- **Joint End-to-End Pruning:** We have proposed a joint regularization-based training to tackle the issue of pre-defining pruning ratio per layer. Our joint end-to-end pruning is scalable for deep models such as adopted in depth estimation. We learn binary masks per layer to drop filters jointly with task loss. We showed how pruning benefits small model training compared to training from scratch, especially with limited data.

- **Layer Pruning:** We questioned the speedup gain from filter pruning granularity in comparison to filter pruning. In this line of work, our contribution is two-fold. First, we proposed a novel accuracy criterion by weight imprinting to evaluate each layer's importance. We have demonstrated that our layer pruning method achieves much better latency reduction than the state-of-the-art filter

pruning methods. Secondly, we further propose LayerPrune framework for a thorough evaluation of both granularities under different inference setups. We explore high-end and edge devices as a deployment hardware platform and different batch sizes. We conclude that for low batch size, filter pruning achieves minimal gain in speedup for GPUs due to underutilization. While higher batch size allows for better utilization, speedup gain is highly affected by filter pruning ratio per layer. This complicates hardware-aware filter pruning optimization. On the other hand, layer pruning allows for more hardware-agnostic pruning.

- **Dynamic Pruning:** A recent interest in deployment is conditional inference. Lightweight models perform fairly well in most input cases. Dynamic pruning allows for inference of different sub-networks which can be seen as online pruning. We proposed a novel formulation for dynamic model pruning. Similar to other dynamic pruning methods, we equip a cheap decision head to the original convolutional layer. However, we proposed to train the decision heads in a self-supervised paradigm instead of adding a regularization term. Decision heads predict the most likely to be highly activated filters given the layer's input activation. The masks are trained using a binary cross-entropy loss decoupled from the task loss to remove loss interference. We generated the mask ground truth based on a novel criterion using the heatmap mass per input sample. In our experiments, reached similar accuracy to SOTA methods with 15% and 24% higher FLOPs reduction on CIFAR. Similarly in ImageNet, we achieve a lower drop in accuracy with up to 13% improvement in FLOPs reduction.

- **Self-distillation:** The previous contribution focused on model pruning with the goal to increase the model's efficiency at inference time. In this contribution, we tackled training efficiency, especially that of vision transformers. One way to fill the accuracy gap between vision transformers and convolution networks for small datasets, in which the former underperforms the latter, is knowledge

distillation. However, model distillation requires 1) a pre-trained teacher, and 2) teacher inference while training the student. We proposed a self-distillation loss for data-efficient transformer training. Auxiliary lightweight classifiers are inserted at different depths in the transformer backbone. These classifiers are trained using a weighted sum between task loss and distillation loss. The distillation loss enforces these classifiers to mimic the output of the final classifier in a self-supervision way. In addition, we also proposed a query efficient distillation that inference the teacher in a selective manner. We showed that self-distillation achieved 2.8x speedup over teacher-student distillation while achieving on-bar accuracy in most cases. We also showed that in the case of pre-trained teacher availability, selective occasional calls to the teacher are sufficient to reach the full potential of distillation.

8.2 Future Work

For our future work we leave multiple research questions that we think are of significance and have not been thoroughly tackled in the literature yet:

- **Rethinking Model Compression Metrics:** As we have seen, the commonly reported metrics are indirect metrics such as FLOPs and the number of parameters. These metrics do not fill the gap between practical gain and may contribute to misleading evaluations. A community drive hardware sharing or closer to deployment workshops is growing but not yet fully present in papers due to technical difficulty. Papers such as [133] provide a practical recipe for INT8 quantization in specific. Such practical questions or recipes are a direction we need to bring more in academic publications, especially model compression.

- **Dynamic Inference:** Conditional inference is another direction that is gaining more attention. Some with ready-to-deploy pipelines such as [134, 135], but the majority still do not reach their full potential due to hardware and

framework limitations. We think that conditional inference can benefit greatly from advances in curriculum learning literature. In curriculum learning, methods define a difficulty measurer module to gradually include samples of harder aspects. These measures can help motivate the way the network condition its decisions. On the practical front, dynamic filter inference requires more backend framework development to allow for efficient dynamic slicing of different filters. A thorough evaluation for different dynamic granularity such as filter vs early exit, a cascade of independent lightweight models, or a hybrid edge-cloud deployment is needed to layout promising tracks and raise research questions.

- **Retraining for Different Budgets:** Model pruning methods require retraining to reach different compute budgets, making the training process very expensive. There are methods such as slimmable [136] that train multiple networks with different widths in the same training. However, training from scratch of the individual networks still outperform the joint training due to the complexity to balance different objectives.

8.3 Closing Remarks

There has been a lot of development in the model compression literature. Unlike other research problems, model compression includes many intersectionalities across different areas including software and hardware. This results in a cloudy big-picture in terms of practical gain and the expectation from different sub-problem. We close this thesis with an attempt for such a practical recipe shown in Figures **??** and 8.2 for pruning and model compression respectively. We include only the most beneficial tracks from personal experience under different inference setups in terms of speedup gain. It is worth mentioning that distillation can be considered in all scenarios.

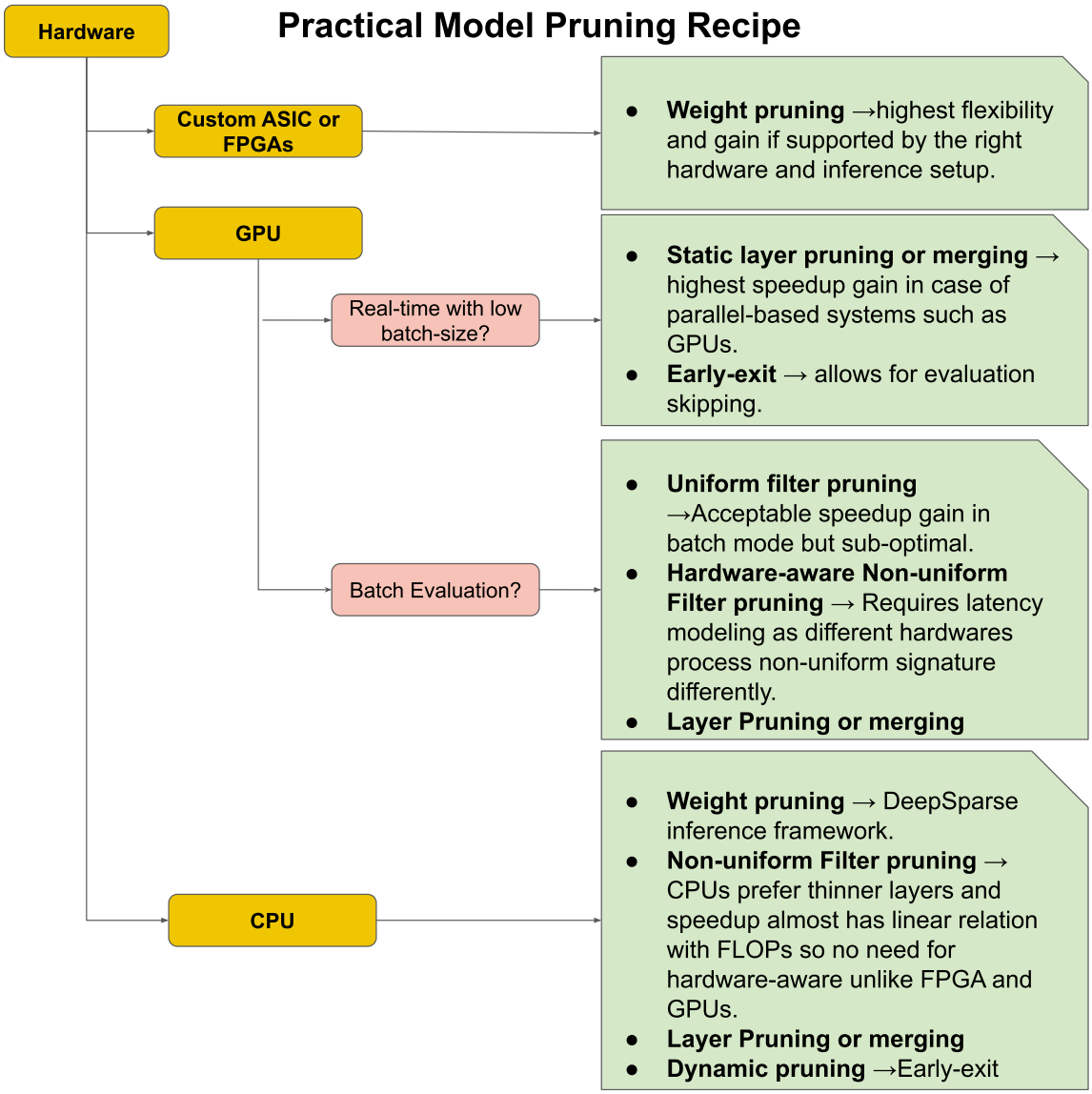


Figure 8.1: Practical guidelines to apply pruning on different deployment setups.

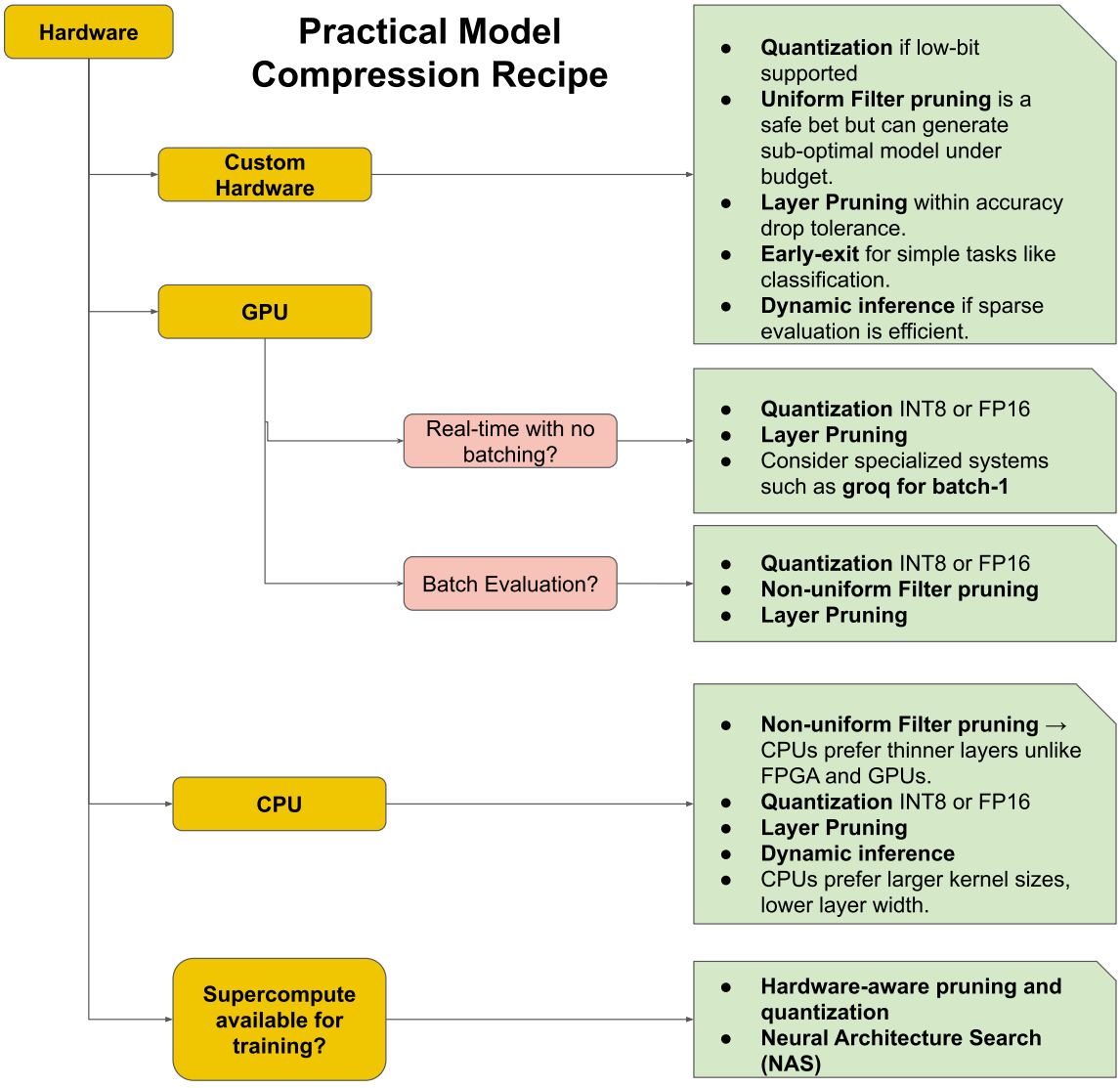


Figure 8.2: An attempt to draw a big-picture with a practical guideline under different inference scenarios.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] S. Elkerdawy, H. Zhang, and N. Ray, "Lightweight monocular depth estimation model by joint end-to-end filter pruning," in *2019 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2019, pp. 4290–4294.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE CVPR*, 2016, pp. 770–778.

[5] Z. Wu, C. Shen, and A. Van Den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," *Pattern Recognition*, pp. 119–133, 2019.

[6] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.

[7] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[8] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," in *Advances in Neural Information Processing Systems*, 2018, pp. 1963–1972.

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[10] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," *ICLR*, 2018.

[11] e. a. Kurt Keutzer, "Abandoning the dark arts: Scientific approaches to efficient deep learning," *The 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing , Conference on Neural Information Processing Systems*, 2019.

[12] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE CVPR*, 2019, pp. 8612–8620.

[13] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[14] C. Yang, L. Xie, C. Su, and A. L. Yuille, "Snapshot distillation: Teacher-student optimization in one generation," in *Proceedings of the IEEE CVPR*, 2019, pp. 2859–2868.

[15] X. Jin *et al.*, "Knowledge distillation via route constrained optimization," in *Proceedings of the IEEE ICCV*, 2019, pp. 1345–1354.

[16] K. Hao, *Training a single ai model can emit as much carbon as five cars in their lifetimes*, 2020.

[17] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE ICCV*, 2017, pp. 1389–1397.

[18] T.-J. Yang *et al.*, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[19] H. Yang, Y. Zhu, and J. Liu, "Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model," in *Proceedings of the IEEE CVPR*, 2019, pp. 11 206–11 215.

[20] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *ICLR*, 2019.

[21] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[22] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, pp. 64 270–64 277, 2018.

[23] B. van Werkhoven, "Kernel tuner: A search-optimizing gpu code auto-tuner," *Future Generation Computer Systems*, pp. 347–358, 2019.

[24] C. Nugteren and V. Codreanu, "Cltune: A generic auto-tuner for opencl kernels," in *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, IEEE, 2015, pp. 195–202.

[25] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 409–424.

[26] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 2178–2188.

[27] W. Hua, Y. Zhou, C. M. De Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019.

[28] D. Ghimire, D. Kil, and S.-h. Kim, "A survey on efficient convolutional neural networks and hardware acceleration," *Electronics*, vol. 11, no. 6, p. 945, 2022.

[29] Baidu, *Deepbench: Benchmarking deep learningoperations on different hardware*, https://github.com/baidu-research/DeepBench.

[30] *Mlperf*, https://mlperf.org/.

[31] C. Coleman *et al.*, "Dawnbench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, p. 102, 2017.

[32] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *ICLR*, 2017.

[33] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[34] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[35] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.

[36] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*, PMLR, 2015, pp. 1737–1746.

[37] P. Micikevicius *et al.*, "Mixed precision training," *ICLR*, 2017.

[38] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," *NeurIPS*, 2017.

[39] D. Das *et al.*, "Mixed precision training of convolutional neural networks using integer operations," *ICLR*, 2018.

[40] *Knights mill - microarchitectures - intel*.

[41] N. DGX, *With tesla v100 system architecture*, 1.

[42] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[43] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, IEEE, 2018, pp. 1–6.

[44] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[45] A. D. Lascorz *et al.*, "Shapeshifter: Enabling fine-grain data width adaptation in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 28–41.

[46] K. Rocki *et al.*, "Fast stencil-code computation on a wafer-scale processor," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2020, pp. 1–14.

[47] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, 2012.

[48] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *NeurIPS*, 2014.

[49] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[50] V. Lakshmanan, S. Robinson, and M. Munn, *Machine learning design patterns.* " O'Reilly Media, Inc.", 2020.

[51] B. Chen and J. M. Gilbert, *Introducing the cvpr 2018 on-device visual intelligencechallenge, google ai blog, 2018*, https://ai.googleblog.com/2018/04/introducing-cvpr-2018-on-device-visual.html.

[52] A. Zhou *et al.*, "Learning n: M fine-grained structured sparse neural networks from scratch," *ICLR*, 2021.

[53] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[54] S Han, H Mao, and W. Dally, "Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR 2017*, 2015.

[55] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *BMVC*, 2015.

[56] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," *ICLR*, 2018.

[57] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *ICLR*, 2019.

[58] S. Sharify *et al.*, "Laconic deep learning inference acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 304–317.

[59] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[60] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE CVPR*, 2019, pp. 11 264–11 272.

[61] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE ICCV*, 2017, pp. 2736–2744.

[62] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE ICCV*, 2017, pp. 5058–5066.

[63] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[64] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *ICLR*, 2017.

[65] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2780–2789.

[66] J. Liu *et al.*, "Discrimination-aware network pruning for deep model compression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[67] T.-W. Chin, C. Zhang, and D. Marculescu, "Layer-compensated pruning for resource-constrained convolutional neural networks," *NeurIPS*, 2018.

[68] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800.

[69] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 304–320.

[70] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE transactions on pattern analysis and machine intelligence*, no. 12, pp. 3048–3056, 2018.

[71] Z. Wu *et al.*, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8817–8826.

[72] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[73] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[74] Y. Wang, X. Zhang, X. Hu, B. Zhang, and H. Su, "Dynamic network pruning with interpretable layerwise channel selection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 6299–6306.

[75] D. Wang, Y. Li, L. Wang, and B. Gong, "Neural networks are more productive teachers than human raters: Active mixup for data-efficient knowledge distillation from a blackbox model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1498–1507.

[76] C. Shen, X. Wang, Y. Yin, J. Song, S. Luo, and M. Song, "Progressive network grafting for few-shot knowledge distillation," *AAAI*, 2020.

[77] L. Zhang, C. Bao, and K. Ma, "Self-distillation: Towards efficient and compact neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[78] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, PMLR, 2021, pp. 10 347–10 357.

[79] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[80] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *ICLR*, 2016.

[81] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 270–279.

[82] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, 2013.

[83] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[84] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2650–2658.

[85] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, IEEE, 2012, pp. 3354–3361.

[86] F. Liu, C. Shen, G. Lin, and I. D. Reid, "Learning depth from single monocular images using deep convolutional neural fields.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 2024–2039, 2016.

[87] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *CVPR*, vol. 2, 2017, p. 7.

[88] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2002–2011.

[89] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "Towards real-time unsupervised monocular depth estimation on cpu," *IROS*, 2018.

[90] M. Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[91] C. Lemaire, A. Achkar, and P.-M. Jodoin, "Structured pruning of neural networks with budget-aware regularization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9108–9116.

[92] H. Qi, M. Brown, and D. G. Lowe, "Low-shot learning with imprinted weights," in *Proceedings of the IEEE CVPR*, 2018, pp. 5822–5830.

[93] M. Siam and B. Oreshkin, "Adaptive masked weight imprinting for few-shot segmentation," 2019.

[94] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer International Publishing, 2014, pp. 346–361.

[95] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *The Journal of Machine Learning Research*, no. 1, pp. 5595–5637, 2017.

[96] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *ICLR*, 2017.

[97] M. Lin *et al.*, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1529–1538.

[98] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[99] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: A more complicated network with less inference complexity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5840–5848.

[100] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[101] W. Hua, Y. Zhou, C. De Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," *NeurIPS*, 2018.

[102] S. Lowel and W. Singer, "Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity," *Science*, vol. 255, no. 5041, pp. 209–212, 1992.

[103] X. Ye *et al.*, "Accelerating cnn training by pruning activation gradients," in *European Conference on Computer Vision*, Springer, 2020, pp. 322–338.

[104] A. Mallya and S. Lazebnik, "Piggyback: Adding multiple tasks to a single, fixed network by learning to mask," *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[105] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[106] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[107] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[108] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[109] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NeurIPS*, 2019.

[110] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *IJCAI*, 2018.

[111] B. Li, B. Wu, J. Su, and G. Wang, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," in *European conference on computer vision*, Springer, 2020, pp. 639–654.

[112] Y. Tang *et al.*, "Scop: Scientific control for reliable neural network pruning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 936–10 947, 2020.

[113] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

[114] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus, "Provable filter pruning for efficient neural networks," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[115] Y. Ovadia *et al.*, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *NeurIPS*, 2019.

[116] G. W. Brier *et al.*, "Verification of forecasts expressed in terms of probability," *Monthly weather review*, vol. 78, no. 1, pp. 1–3, 1950.

[117] S. Elkerdawy, M. Elhoushi, A. Singh, H. Zhang, and N. Ray, "To filter prune, or to layer prune, that is the question," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[118] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[119] H. Yang, Y. Zhu, and J. Liu, "Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 206–11 215.

[120] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, and D. Rus, "Lost in pruning: The effects of pruning neural networks beyond test accuracy," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[121] M. Paganini, "Prune responsibly," *arXiv preprint arXiv:2009.09936*, 2020.

[122] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[123] L. Yuan *et al.*, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 558–567.

[124] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[125] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, and Z. Dai, "Variational information distillation for knowledge transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9163–9171.

[126] L. Yu, V. O. Yazici, X. Liu, J. v. d. Weijer, Y. Cheng, and A. Ramisa, "Learning metrics from teachers: Compact networks for image embedding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2907–2916.

[127] L. Sun, J. Gou, B. Yu, L. Du, and D. Tao, "Collaborative teacher-student learning via multiple knowledge transfer," *CoRR*, 2021.

[128] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3713–3722.

[129] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[130] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *ICLR*, 2019.

[131] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *ICLR*, 2016.

[132] X. Li, W. Wang, X. Hu, J. Li, J. Tang, and J. Yang, "Generalized focal loss v2: Learning reliable localization quality estimation for dense object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 632–11 641.

[133] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *arXiv preprint arXiv:2004.09602*, 2020.

[134] Y. Long, I. Chakraborty, and K. Roy, "Conditionally deep hybrid neural networks across edge and cloud," *arXiv preprint arXiv:2005.10851*, 2020.

[135] R. G. Pacheco, K. Bochie, M. S. Gilbert, R. S. Couto, and M. E. M. Campista, "Towards edge computing using early-exit convolutional neural networks," *Information*, vol. 12, no. 10, p. 431, 2021.

[136] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *ICLR*, 2018.