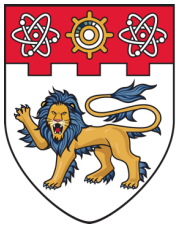
ADAPTIVE NEURAL NETWORKS FOR EDGE INTELLIGENCE



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

HAO KONG

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

30 November 2023

Date

HAO KONG

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accordance with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

30 November 2023

Date

Liu Weichen

Assoc Prof. Liu Weichen

Authorship Attribution Statement

This thesis contains material from 5 paper(s) published in the following peer-reviewed journal(s) / from paper(s) accepted at conferences in which I am listed as an author.

Chapter 2.4 is published as Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing AI to Edge: From deep learning's perspective. Neurocomputing, 485, 297-320 (2022). DOI: 10.1016/j.neucom.2021.04.141.

The contributions of the co-authors are as follows:

- Assoc/Prof Weichen Liu indicated the initial survey direction and edited the manuscript drafts.
- Dr. Di Liu prepared the manuscript drafts.
- I reviewed the literature on model pruning and contributed to the corresponding section for model pruning.
- Xiangzhong Luo contributed to the section for neural architecture search.
- Dr. Ravi Subramaniam proofread the final manuscript.

Chapter 3 is published as Hao Kong, Shuo Huai, Di Liu, Lei Zhang, Hui Chen, Shien Zhu, Shiqing Li, Weichen Liu, Manu Rastogi, Ravi Subramaniam, Madhu Athreya, and Anthony Lewis. EDLAB: A benchmark for edge deep learning accelerators. IEEE Design & Test, 39, 8-17 (2022). DOI: 10.1109/MDAT.2021.3095215

The contributions of the co-authors are as follows:

- Assoc/Prof Weichen Liu, Madhu Athreya, and Anthony Lewis provided the initial project direction.
- I collected experimental data and prepared the manuscript drafts.
- Dr. Di Liu, Shuo Huai, Lei Zhang, Hui Chen, and I designed the hardware benchmarking methodology.
- Shuo Huai, Hui Chen, Shien Zhu, Shiqing Li, and I conducted evaluation experiments on edge deep learning hardware accelerators.
- Manu Rastogi and Ravi Subramaniam proofread the final manuscript and provided feedback.

Chapter 4 is published as Hao Kong, Di Liu, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. HACScale: Hardware-aware compound scaling for resource-efficient DNNs. 27th Asia and South Pacific Design Automation Conference (ASP-DAC), 708-713 (2022). DOI: 10.1109/ASP-DAC52403.2022.9712593.

The contributions of the co-authors are as follows:

- Assoc/Prof Weichen Liu and Dr. Di Liu provided the initial project direction and edited the manuscript drafts.
- I designed the framework, implemented the algorithms, conducted experiments, and prepared the manuscript drafts.
- Xiangzhong Luo prepared some benchmarks for experiments and collected some experimental results.
- Ravi Subramaniam proofread the final manuscript and provided feedback.

Chapter 5 is published as Hao Kong, Xiangzhong Luo, Shuo Huai, Di Liu, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu. EMNAPE: Efficient multi-dimensional neural architecture pruning for edgeAI. Design, Automation, and Test in Europe Conference and Exhibition (DATE), 1-2 (2022). DOI: 10.23919/DATE56975.2023.10137122

The contributions of the co-authors are as follows:

- Assoc/Prof Weichen Liu and Dr. Di Liu provided the initial project direction and edited the manuscript drafts.
- I designed the framework, implemented the algorithms, conducted experiments, and prepared the manuscript drafts.
- Xiangzhong Luo and Shuo Huai prepared the datasets for experiments and edited the manuscript drafts.
- Ravi Subramaniam, Christian Makaya, and Qian Lin proofread the final manuscript and provided feedback.

Chapter 6 is published as Hao Kong, Di Liu, Shuo Huai, Xiangzhong Luo, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu. EdgeCompress: Coupling multi-dimensional model compression and dynamic inference for edgeAI. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), early access (2023). DOI: 10.1109/TCAD.2023.3276938.

The contributions of the co-authors are as follows:

- Assoc/Prof Weichen Liu provided the initial project direction and edited the manuscript drafts.
- I designed the framework, implemented the algorithms, conducted experiments, and prepared the manuscript drafts.
- Dr. Di Liu edited the manuscript drafts.
- Xiangzhong Luo and Shuo Huai prepared the datasets for experiments and edited the manuscript drafts.
- Ravi Subramaniam, Christian Makaya, and Qian Lin proofread the final manuscript and provided feedback.

30 November 2023

. . . .

Date

. . . .

HAO KONG

Acknowledgements

When the day does come, I finally get a chance to have all the important names in my life engraved and preserved forever.

Prof. Weichen Liu is the first person I would like to express my sincere gratitude to. He is my supervisor and the most critical leader in my academic career. Without him, I would not have been able to embark on my academic journey and get to where I am now. His rigorous attitude towards academic research and meticulous care for students deeply infected me and inspired me to be a person who is beneficial to society and others in the future. It is a lifelong honor for me to spend four years studying under his supervision. In addition, I would also show my sincere thanks to Dr. Di Liu, who can always give me advice at critical moments to help me solve difficulties in my researches. Meanwhile, we are also friends in life and comrades-in-arms on the sports field. Prof. Wei Jiang and Prof. Jinyu Zhan, my undergraduate mentors, made me interested in academic research and gave me a lot of valuable guidance at the early stage.

I would extend my thanks to the Thesis Advisory Committee (TAC) members: Prof. Rui Tan and Dr. Zhehui Wang for their support and encouragement. Meanwhile, I sincerely thank the thesis examiners for their generous and selfless efforts in reviewing this dissertation.

During the long journey towards the Ph.D. degree, I have met a bunch of talented, warm, and generous people: Dr. Hui Chen, Dr. Shien Zhu, Xiangzhong Luo, Shuo Huai, Shiqing Li, Guochu Xiong, Dr. Jun Zhou, Dr. Luan H.K. Duong, Dr. Peng Chen, Dr. Mengquan Li, Lei Zhang, Wenyang Liu, Chunyun Chen, Loy Yong Yi Wendy, Vikash Sathiamoorthy. Their dedication to research, optimistic attitude towards life, and considerate care for me are constantly encouraging me to move forward in the future.

For Ph.D. students, financial issues are as stressful as research difficulties. But thankfully, Prof. Tan Ming Jen, Dr. Qian Lin, Dr. Ravi Subramaniam, Dr.

Christian Makaya, Dr. Manu Rastogi, and Mr. Madhu Athreya provided both generous financial support and valuable project suggestions under the HP-NTU Digital Manufacturing Corporate Lab.

Family and friends are always the most solid and reliable backing for me on the way to my dreams. I hereby express my deepest love to my father (Deren Kong), my mother (Suifen Peng), my two elder sisters (Xiujuan Kong and Yaling Kong), my brother-in-law (Chao Meng), and my best friends (Haowei Shi and Xiangjin Zhong).

Last but not least, my better half, Xiaoqian Huang, has been waiting for me more than 3,000 kilometers away for four years. May I live up to her love and unconditional support throughout my life.

COVID-19 restricts my freedom, but it cannot restrain my thoughts of family, the better half, and friends. The longer the restriction, the stronger the yearning. The pandemic will eventually dissipate, but love will never fade.

Abstract

Deep neural networks (DNNs) have achieved remarkable results and have become the mainstay of many applications including autonomous driving and emerging AI-enabled chatbots. However, the superior performance of advanced DNN models comes at the cost of enormous computation and memory footprint. For instance, ChatGPT enabled by the GPT-3.5 model breaks the records of multiple benchmarks with 175 Billion parameters, which puts a significant strain on hardware capabilities. To satisfy the resource consumption of DNNs and efficiently execute DNNs to process input data, the traditional paradigm of AI hosts DNN models on powerful cloud servers, where data of users will be uploaded to the cloud for processing and the results will be returned to users. Inevitably, this mode aggravates users' concern about the leakage of data privacy. To address this concern, a new paradigm has emerged that proposes deploying DNNs to edge devices near users to process private data securely. However, edge devices are usually sensitive to resource consumption, and different edge devices are distinct from each other in terms of available resources and computing capability. Therefore, how to adapt DNNs to efficiently utilize the resources of various edge hardware for better performance becomes urgent for edge intelligence. To answer this question, we have been focusing on the design and efficient adaptation (e.g., compression and scaling) of DNN models, so that the execution overhead of models can be matched to the given hardware resources to achieve the best trade-off between execution efficiency and prediction accuracy. First, to comprehensively understand the hardware, we introduce EDLAB, an end-to-end benchmark, to evaluate edge deep learning accelerators. EDLAB consists of state-of-the-art deep learning models, a unified workload preprocessing and deployment framework, as well as a collection of comprehensive metrics. In EDLAB, we also propose parameterized models to model the hardware performance bound, so that EDLAB can identify the hardware potentials and the hardware utilization of different deep learning applications. After evaluating hardware devices, we will adapt DNNs accordingly to efficiently utilize available hardware resources for better performance. Specifically, for powerful

edge devices that have adequate resources, we propose HACScale and AdaptScale to efficiently scale DNN models for better accuracy without sacrificing execution efficiency. HACScale is a hardware-aware scaling framework, which jointly scales different dimensions of a model according to their impact on resource utilization and accuracy. When applying HACScale to different models, we observe that the optimal scaling strategy of different models is very distinct, and sharing the same scaling strategy across different models may not achieve the best accuracy and resource utilization. Therefore, we further propose AdaptScale, a model-aware adaptive scaling framework to efficiently customize the scaling strategy for different models for the best model performance.

Meanwhile, for less capable edge devices, we also introduce two novel model compression frameworks: TECO and TICO, to reduce the costs of DNN models so that they can be efficiently deployed onto resource-constrained edge devices. TECO is a multi-dimensional model pruning framework. Compared to existing pruning frameworks that only prune a single dimension of DNN models, TECO collaboratively prunes multiple dimensions (i.e., depth, width, and resolution) to more comprehensively reduce redundant parameters and computation for higher execution efficiency. In TECO, we first introduce a two-stage importance evaluation framework, which efficiently and comprehensively evaluates each pruning unit according to both the local importance inside each dimension and the global importance across different dimensions. Based on the evaluation framework, we present a heuristic pruning algorithm to progressively prune the three dimensions of CNNs towards the optimal trade-off between accuracy and efficiency. In addition, we find that existing compression approaches mainly focus on reducing the inference overhead of models while ignoring the training overhead, which loses the opportunity to update the deployed model with private data on edge devices due to the huge training cost. To address this issue, we propose TICO, a co-optimization framework to optimize both the training and inference performance of deep learning models. In TICO, we first introduce a novel multi-objective pruning approach, where we take both training and inference performance as optimization objectives, and then formulate the pruning of a model as a multi-objective optimization problem. Subsequently, we design an evolutionary algorithm to efficiently search for the optimal pruning decision. Moreover, to further compress the training cost, we also propose a resolution-adaptive training strategy, which trains models with a small image size at early training epochs and progressively increases the size of training images.

Compared to the traditional training paradigm which trains a model with the same large image size throughout the whole training process, our approach significantly reduces the training cost and improves the training performance of models on edge devices.

In addition to optimizing the efficiency of DNN models at design time, we also propose EdgeCompress, a dynamic inference framework to avoid unnecessary computation of DNN models at inference time. In EdgeCompress, we first introduce dynamic image cropping, where we design a lightweight foreground predictor to accurately crop the most informative foreground object of input images for inference, which avoids redundant computation on background regions. Subsequently, we present compound shrinking to collaboratively compress the three dimensions (depth, width, and resolution) of CNNs. Dynamic image cropping and compound shrinking together constitute a multi-dimensional CNN compression framework, which is able to comprehensively reduce the computational redundancy in both input images and neural network architectures, thereby improving the inference efficiency of CNNs. Further, we present a dynamic inference framework to efficiently process input images with different recognition difficulties, where we cascade multiple models with different complexities from our compression framework and dynamically adopt different models for different input images, which further compresses the computational redundancy and improves the inference efficiency of CNNs, facilitating the deployment of advanced CNNs onto embedded hardware.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Over the past decade, since the advent of AlexNet [3], deep neural networks (DNNs) have been increasingly demonstrating magnificent potential in solving various tasks, such as image classification [1], object detection [2], natural language processing (NLP) [4], and etc. Due to their excellent performance, many DNNs have already been deployed to cloud servers to efficiently handle massive amounts of data and user requests. For example, the Google search engine enables the function of searching by images, which utilizes vision models to extract the features of images, and then finds related images [5]. Meanwhile, many online language translators exploit NLP models like Bert [4] to improve the translation quality. More recently, ChatGPT's [6] performance on multiple NLP tasks, such as relation classification, textual event detection [7, 8], and entity typing, has refreshed people's expectations for deep learning (DL) models. By scaling up the model to an extremely large size, it obtains an emergent ability that has not appeared in all previous small models. which even exceeds the best human performance in many tests [9].

1.1.1 Edge Intelligence

In the above-mentioned paradigm, deep learning models are usually deployed in powerful datacenters, where users need to upload their data to the cloud for processing. However, this will inevitably lead to some problems. First, uploading data

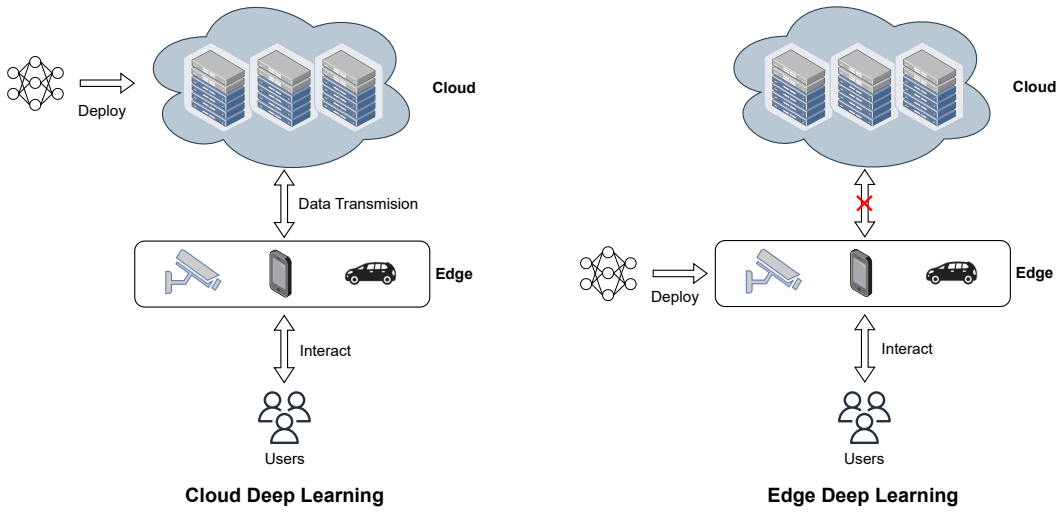


FIGURE 1.1: Cloud deep learning v.s. edge deep learning.

through the network is bound to bring a certain amount of network latency. The network latency may fluctuate based on the network environment and the cloud server load, which can be fatal for some safety-critical or latency-critical applications, such as autonomous driving [10] or unmanned aerial vehicle (UAV) [11]. In addition, uploading private data to the cloud for processing may result in data leakage, which does not meet people's increasing demands for data privacy. For example, uploading pictures or videos captured by home cameras may violates the privacy of users and may cause security issues. To solve the network and data privacy issues, as demonstrated in Figure 1.1, there emerges a new trend to deploy DL models onto various edge devices, which are near to users and thus can process data locally instead of uploading them onto the cloud. This paradigm of deep learning is called edge intelligence [12, 13], which has been widely used in latency-critical applications. For example, many electric vehicles are equipped with edge deep learning accelerators, such as NVIDIA Jetson AGX Orin [14], to process images collected by cameras and execute deep learning-based self-driving algorithms. Without uploading private data to cloud datacenters, the network latency and data privacy issues are mitigated significantly. Consequently, the safety of self-driving cars can be guaranteed.

However, every coin has two sides. Edge intelligence also brings new challenges. On the one hand, it is well known that DL models are usually resource-hungry, which contain a large amount of computation and parameters. For example, ResNet-50 [15] takes about 4.1 billion floating-point operations (FLOPs) to process an input

TABLE 1.1: Different edge deep learning hardware platforms. Note that AGX Orin supports the acceleration for sparse DNNs, so we display its performance for both sparse and dense computing.

Device	Power	Memory	CPU Freq.	GPU Freq.	Performance
AGX Orin	60 W	64GB LPDDR5	2.2 GHz	1.3 GHz	275 TOPS (Sparse) / 138 TOPS (Dense)
AGX Xavier	30 W	32GB LPDDR4x	2.2 GHz	1377 MHz	32 TOPS (Dense)
Jetson TX2	15 W	8GB LPDDR4	2.0 GHz	1.3 GHz	1.33 TFLOPS (Dense)
Jetson Nano	10 W	4GB LPDDR4	1.43 GHz	921 MHz	472 GFLOPS (Dense)
Edge TPU	5 W	4GB LPDDR4	1.5 GHz	N/A	4 TOPS (Dense)
Intel NCS2	5 W	4GB LPDDR4	700 MHz	N/A	4 TOPS (Dense)

image at the size of 224×224. SENet [16], a more advanced DL model, uses 146 million parameters and 42 billion FLOPs to achieve 82.7% top-1 validation accuracy on ImageNet [1]. More recently, the backbone of ChatGPT, GPT-3 [6], even scales up the model parameters to 1,700 billion for better performance. The massive parameters and computation pose serious challenges to hardware capabilities like memory capacity and computing capability. On the other hand, considering the restricted energy supply in edge environments [12], conventional edge hardware platforms are usually designed to be resource-economic, which means only very limited hardware resources, such as memory, computing units, etc, will be available for the execution of deployed software. As a consequence, the execution of DL models on edge hardware devices will be extremely slow, which may lead to catastrophic consequences for some applications. Moreover, due to the memory mismatch between DL models and edge devices, some large models may not even be able to be deployed onto the hardware devices.

To enable the deployment and efficient execution of DL models on edge hardware platforms, efforts have been made on both the hardware side and the software side. First, on the hardware side, many emerging edge DL hardware accelerators are proposed to accommodate the resource requirements of DNNs and speed up their execution [14, 17, 18]. These edge DL accelerators are usually equipped with more computing units and larger memory capacity, so that larger models can be successfully loaded into the memory and more computation can be executed in parallel, thereby improving the execution efficiency of DNNs on edge devices. Meanwhile, the emerging edge accelerators also strive to minimize energy consumption, so that they can be deployed in more edge environments with strict energy constraints and achieve higher power efficiency. Specifically, NVIDIA launches the Jetson family, a series of embedded GPU platforms, which includes Jetson AGX Orin [14], Jetson AGX Xavier [17], Jetson TX2 [19], and Jetson Nano [20]. From the most powerful platform, Jetson AGX Orin [14], to the most resource-economic platforms,

Jetson Nano [20], The Jetson family offers different trade-offs between hardware capabilities and power consumption to meet resource constraints and achieve the best performance in various edge environments. For example, as demonstrated in Table 1.1, Jetson AGX Orin [14] is equipped with 64 GB 256-bit LPDDR5 memory with a bandwidth of 204.8 GB/s, which is able to deliver up to 275 Tera operations per second (TOPS) AI performance for sparse DNNs with power consumption up to 60 W. As a comparison, Jetson Nano [20] only has 4 GB 64-bit LPDDR4 memory and the memory bandwidth is 25.6 GB/s. The peak computing performance Nano can achieve is about 472 Giga floating-point operations per second (GFLOPS). Correspondingly, the power consumption of Nano only ranges from 5 W to 10 W. In addition, Google also proposes Edge Tensor Processing Unit (TPU) [18] to facilitate the development of edge intelligence. Edge TPU is an Application-Specific Integrated Circuit (ASIC) based hardware specially designed to run inference at the edge, which provides 4 GB LPDDR4 memory and can deliver 4 TOPS inference performance and 2 TOPS/W power efficiency. Different from the above standalone edge platforms, Intel Neural Compute Stick (NCS) [21] is designed as a plugin accelerator, which needs to work with other edge devices that have an operating system, such as Raspberry Pi [22]. The host edge device of NCS is responsible for preprocessing input images, controlling the execution of DL models on NCS, and collecting the execution results. To facilitate the deployment of DL models onto edge devices, different hardware vendors also provide specific software development kits (SDKs) for their hardware. For example, NVIDIA uses TensorRT [23] to optimize models for better performance, while Google employs TensorFlow Lite [24] as the deployment tool. Intel also develops OpenVINO [25] for NCS to efficiently deploy DL models. The boom of heterogeneous DL accelerators facilitates the deployment of AI solutions in different edge environments, but it also brings new problems that need to be solved. Specifically, different hardware vendors only provide the specifications of their DL accelerators, like the peak computing performance and memory capacity. However, the actual system performance depends not only on hardware specifications but also on the deep learning model being deployed. To get the actual execution performance of DL models on a specified hardware accelerator, we have to deploy the model onto the target hardware. However, as we discussed before, the heterogeneous hardware architecture and various SDKs make it difficult to deploy a model onto different accelerators to evaluate their actual performance. In addition, in hardware-aware DL model

design processes, there may be dozens of candidate models to be evaluated. The complex deployment and evaluation procedure of edge accelerators will also reduce the design efficiency.

On the software side, researchers focus on designing lightweight neural networks to reduce resource consumption and execution time, so that those neural networks can be deployed onto more resource-constrained edge devices to provide service. There are mainly two ways to design lightweight neural networks: 1) manual design and 2) neural architecture search (NAS). The first category designs neural networks via human experts [26–31]. Specifically, SqueezeNet [27] introduces the Fire module, a novel basic block to construct lightweight convolutional neural networks (CNNs). The Fire module utilizes 1×1 convolution filters to replace the original 3×3 filters, which significantly reduces the parameters and computation of CNNs, achieving higher execution speed and memory efficiency. MobileNet [28] proposes depthwise separable convolution. By combining the depthwise convolution with 3×3 filters and the pointwise convolution with 1×1 convolution, it remarkably reduces the number of 3×3 filters, thereby reducing model parameters and computation for higher efficiency and less resource consumption. Further, MobileNetV2 [29] introduces an inverted residual bottleneck block, which exploits depthwise separable convolution in residual bottleneck blocks and increases the number of 3×3 filters for better feature representation. Thanks to the depthwise separable convolution, adding 3×3 filters will not lead to significant increases in computation and parameters, optimizing the trade-off between efficiency and accuracy. ShuffleNet [26] proposes the Shuffle unit, which consists of pointwise group convolution, channel shuffle, and depthwise convolution. The novel pointwise group convolution divides 1×1 filters into multiple groups and only performs the convolution operation inside each group, further reducing the computation. Thereafter, the channel shuffle will fuse the features obtained from different groups to enable information communication across different groups, thereby preserving prediction accuracy. More recently, Radosavovic *et al.* [31] propose a design paradigm for good models via exploring the design space of DL models. Based on the residual bottleneck block [15], the researchers gradually shrink the design space via empirical experiments. Finally, they identify a relatively small design space, and neural networks designed within this design space are expected to achieve good accuracy and execution performance.

As shown in Figure 1.2, different from manually designed neural networks, NAS

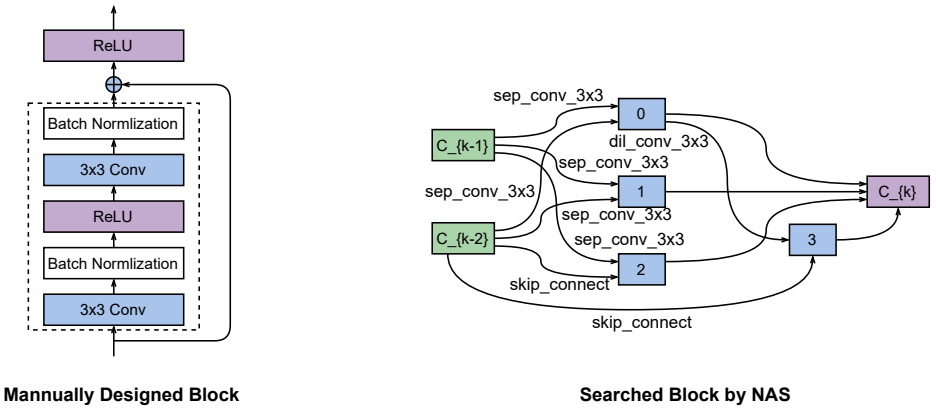


FIGURE 1.2: Two different blocks obtained through manual design and NAS.

directly searches for the optimal network architecture within a pre-defined search space [32–38], which effectively reduces the design cost and the requirement for human labor. According to whether the search process is differentiable, current NAS algorithms can be divided into two categories: 1) non-differentiable NAS and 2) differentiable NAS. Non-differentiable NAS approaches [32, 34, 35, 37] operate by discretizing the search space and using discrete optimization techniques, such as evolutionary algorithms or reinforcement learning, to explore and optimize the architectures. For example, MnasNet [32] takes the model complexity and on-device inference latency as the optimization objectives, and then utilizes reinforcement learning to search for the optimal lightweight architecture for the given hardware platform. AmoebaNet [37] employs an evolutionary algorithm as the optimization tool to find the desired network architecture. However, since the search space is not discrete, non-differentiable NAS approaches cannot continuously optimize the network architectures. Instead, they have to evaluate all candidates generated during the search process to find the optimal architecture, which leads to prohibitive evaluation costs and makes the search process extremely slow. In contrast, by constructing a continuous search space, differentiable NAS approaches [33, 36, 38] are capable of continuously exploring the design space without interrupting the search algorithm to evaluate candidates. By this means, the design cost can be reduced and the search process can be accelerated significantly. Specifically, DARTS [33] makes the search space continuous through a relaxation strategy, and then leverages gradient descent to optimize the architecture, reducing the search complexity significantly. PC-DARTS [38] further introduces partial channel connection to reduce the time and memory overhead of DARTS [33]. Meanwhile, ProxylessNAS [36] formulates the neural network search as a differentiable path pruning process,

updates the scores of different paths through gradient descent, and finally obtains the optimal architecture by removing paths with lower scores.

As new edge DL accelerators with different capabilities and new DL models with different complexities are constantly proposed [39], how to match the model complexity and the given hardware capability becomes an urgent problem. Generally, two types of resource mismatch can happen when we deploy a given model onto a given hardware platform. The first type of mismatch is that the model complexity exceeds the hardware capability. In this situation, the execution of the model on the hardware will be extremely slow, or the model can not even be loaded into the memory. The other type of resource mismatch, on the contrary, is that the hardware capability far exceeds the model overhead, which can happen when a small model is deployed onto a powerful edge device. In this situation, the hardware resources are not fully utilized. As edge DL accelerators are usually resource-efficient, failing to fully utilize the hardware resources can lose the opportunity to achieve higher model accuracy. Both cases of resource mismatch will prevent us from achieving the optimal trade-off between execution efficiency and prediction accuracy. However, as new edge DL hardware keeps coming up, specially designing a DL model for each hardware device will cause a lot of duplicated efforts and waste existing models. Instead, efficiently adapting the complexity of existing models is a more feasible way. Currently, there are two frequently utilized approaches to flexibly adjust the overhead of a given model: 1) model scaling and 2) model pruning.

1.1.2 Deep Learning Model Scaling

As we all know, the golden rule of thumb in model design is that "the larger the model, the better the accuracy" [31, 40–42]. Model scaling is proposed to improve the prediction accuracy by increasing the computation and parameters of DL models [43–46]. For small models that cannot fully utilize the underlying hardware, model scaling can efficiently scale up the model capacity to improve prediction accuracy and hardware utilization. Thanks to the higher hardware utilization, the actual on-device execution efficiency will not be affected significantly even though the scaled models contain more computation and parameters [41]. Taking CNNs as an example, model scaling can be mainly conducted in three dimensions: 1) depth. 2) width, and 3) resolution. As shown in Figure 1.3, the depth of a CNN

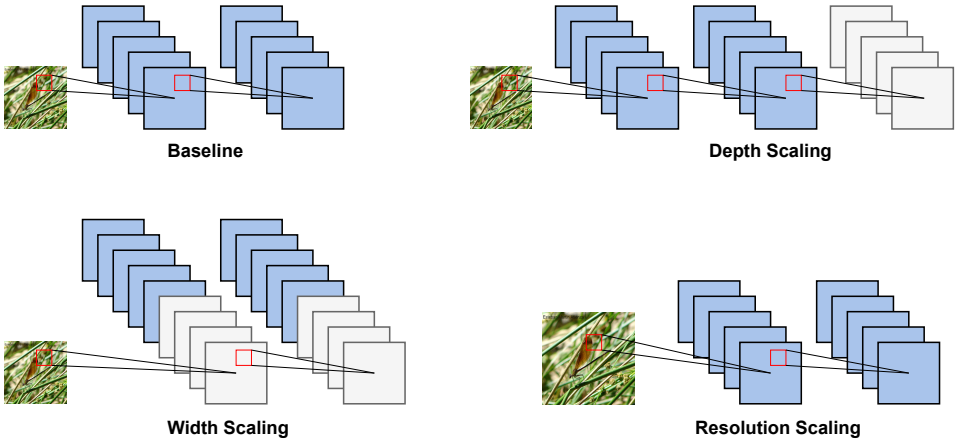


FIGURE 1.3: Demonstration of different scaling approaches.

denotes the number of layers contained by the network, the width represents the number of convolutional filters in each layer, and the resolution refers to the size of the input images. VGG [42] designs a plain CNN architecture, and scales up the depth of the CNN to achieve different trade-offs between model complexity and accuracy. After that, many followers also try to increase the number of layers for higher accuracy [15, 47, 48]. Wide ResNet [41], instead, argues that scaling up the width dimension can also achieve considerable accuracy improvement. Moreover, since the increased computation and parameters on the width dimension can be processed in parallel, the actual training performance and inference performance on hardware are not greatly affected. MorphNet [44] and NeuralScale [45] further improves the efficiency of width scaling by introducing non-uniform width scaling to scale up the width in a layer-wise manner. Instead of scaling up the network architecture, InceptionV2 [49] simply scales up the resolution of input images from 224×224 to 299×299, improving model accuracy without modifying the network architecture. Although scaling the three dimensions alone (i.e., single-dimensional model scaling) can improve the accuracy of DL models, the accuracy will soon be saturated. Continuing to scale up the dimension will not bring about a significant improvement in accuracy, but will lead to an increase in model complexity and deterioration in efficiency [43]. To address this issue, compound model scaling is proposed to jointly scale the three dimensions towards better trade-offs between accuracy and efficiency [43, 46, 50, 51]. EfficientNet [43] first points out that collaboratively scaling multiple dimensions can drive the model to a better balance, achieving higher accuracy than single-dimensional scaling. In compound model scaling, the core optimization problem is to efficiently allocate the resources for

scaling among the three dimensions, so that the scaled model can achieve the highest accuracy under the given model complexity constraint. EfficientNet [40] solves the optimization problem by exhaustively searching the design space for the optimal solution. However, the exhaustive search process will cause a huge cost. Meanwhile, it only takes model parameters and FLOPs as the constraint without considering the wall-clock inference latency, and thus the scaled model may only achieve a sub-optimal on-device efficiency. EfficientNet-X [50] addresses this issue by integrating latency as one of the optimization objectives into the search process. On the other hand, after analyzing the relationship among model FLOPs, parameters, activations, and the wall-clock execution time, Dollár *et al.* [46] find that the actual execution time, including training time and inference time, is more correlated to the number of activations, and scaling the width dimension will lead to the minimum increase in activations among the three dimensions. Therefore, they introduce a width-dominant compound scaling strategy, where most resources will be allocated to the width dimension to obtain a hardware-friendly scaled model.

Although compound model scaling achieves higher accuracy than single-dimensional scaling, it still has some open issues. First, current compound scaling algorithms conduct scaling in a coarse granularity [40, 46, 50, 51]. More specifically, they only identify a single scaling coefficient for each dimension, and thus every single dimension will be scaled in a uniform manner. For example, once the width scaling coefficient is determined, all layers will be uniformly scaled based on this coefficient. However, since it has been widely proven that different layers of a network are of distinct importance to model accuracy and execution efficiency [44, 52, 53], evenly scaling all layers may lead to sub-optimal accuracy and efficiency. In addition, the search process for the optimal compound scaling for a given model will explore a huge design space, which can lead to a considerable design overhead. Moreover, due to the heterogeneity of different DNNs, the searched scaling strategy can hardly be generalized to other architectures. Therefore, how to reduce the design cost of compound scaling and efficiently design the optimal scaling strategy for different models is a critical and urgent problem.

1.1.3 Deep Learning Model Pruning

As the opposite of model scaling, model pruning [13, 54] is intended to decrease the model complexity by removing the redundant architectures within the network, thereby reducing the resource requirement of the model and fitting the model into resource-constrained edge devices. A deep neural network contains millions of neurons, and Han *et al.* [55] points out that not every neuron contributes equally to the model accuracy. Therefore, by deleting those less sensitive units from the network architecture, the computational complexity and memory consumption of the model will be significantly reduced. According to the pruning granularity, model pruning can be divided into two categories: 1) unstructured pruning and 2) structured pruning.

Unstructured pruning operates at a finer granularity, which mainly considers removing every single redundant parameter for higher computational complexity while minimizing the accuracy loss [55–59]. Specifically, different unstructured pruning approaches introduce different metrics or algorithms, such as L1 normalization, to identify those redundant parameters and set the parameter weight to zero. For example, Han *et al.* [56] argue that those parameters with a smaller L1 normalization value have a less significant impact on model accuracy, and thus they can be safely removed to reduce model overhead without affecting accuracy. Further, in [55], model quantization [54] and Huffman Coding are utilized to further compress the pruned architecture. Generally, to accurately identify redundant weights, a model needs to be fully trained to convergence before the pruning starts. The obtained weights can be considered as the optimum for the unpruned network. After pruning is completed, since the architecture of the pruned model has changed significantly, the original network weights may not be optimal for the current architecture and the pruned network may not be convergent anymore, which can cause remarkable accuracy loss. Therefore, the pruned architecture usually needs to be fine-tuned to retrieve the accuracy [56]. NeST [58], instead of pruning all redundant neurons and connections at once, introduces a grow-and-prune paradigm, which combines gradient-based network growing and magnitude-based network pruning to iteratively search the design space for the optimal tiny structure. Since unstructured pruning conducts pruning in a fine-grained manner, it is able to comprehensively discover and reduce the redundancy in DNNs, achieving a high compression ratio and maintaining accuracy. However, unstructured pruning

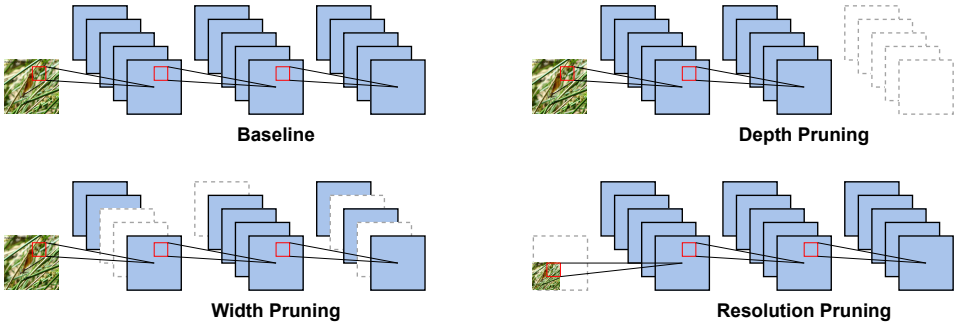


FIGURE 1.4: Demonstration of different structured pruning approaches.

generates sparse network architecture, which can be hardly accelerated by existing hardware devices. Therefore, we cannot enjoy the actual speedup brought by unstructured pruning on existing widely deployed hardware [60].

Unlike unstructured pruning, structured pruning performs pruning on a coarser granularity, e.g., filter, and thus it generates dense architectures that can be well accelerated by exiting hardware platforms [52, 53]. According to the pruning dimension, structured pruning can be further divided into 1) width pruning, 2) depth pruning, and 3) resolution pruning. As demonstrated in Figure 1.4, the width of a DNN refers to the number of filters or channels in each layer, and thus width pruning mainly focuses on eliminating less important filters [44, 52, 53, 61–63]. One of the biggest differences between different width pruning approaches is the pruning metric. HRank [53] proposes using the rank of feature maps as the metric to determine the importance of each channel/filter, and consequently, those channels with low ranks are considered redundant and will be discarded. However, this metric cannot be compared across different layers, thus the researchers have to decide the pruning ratio of each layer manually. Instead, Molchanov *et al.* [52] propose a method to estimate the importance of channels globally, where the importance of each filter is approximated as its gradient through Taylor expansions. This metric can be compared across different layers, and thus the pruning ratio of each layer can be automatically determined during pruning. Depth pruning focuses on constructing shallower DNNs by removing less important layers [64]. As different layers of a DNN are executed sequentially on devices, depth pruning can usually bring about remarkable reductions in execution latency. On the other hand, depth pruning may result in significant accuracy drops since it will remove all filters of pruned layers. Resolution pruning, instead of modifying the network architecture, decreases the computational complexity of DNNs by reducing the size of input

images [29, 32, 65, 66]. MobileNetV2 [29] and MnasNet [32] uniformly shrink the size of all input images for higher efficiency. In addition, DR-ResNet [65] further optimizes the efficiency and accuracy by dynamically adjusting the size of different input images. However, simply shrinking images may cause the loss of important foreground information and lead to a considerable drop in accuracy. GFNet [66] solves this issue by selective cropping, which utilizes RNNs to dynamically crop different patches of the input images for inference, improving model efficiency without sacrificing accuracy.

Despite that the above single-dimensional pruning techniques have achieved considerable improvements in model efficiency, there are still some issues that limit the further potential of model pruning to improve the execution efficiency of DNNs. First, single-dimensional pruning can only achieve very limited compression ratios. Because as the compression ratio increases, there may be some important units of the dimension being removed, affecting the prediction accuracy significantly. In addition, pruning different dimensions is capable of bringing about different benefits. For example, pruning layers can shorten the execution path of DNNs, reducing the inference latency, while resolution pruning is able to greatly optimize the training speed and memory consumption due to the reduction in intermediate activations. Therefore, only pruning a single dimension may fail to enjoy the benefits of pruning other dimensions. Instead, collaboratively pruning multiple dimensions can achieve higher efficiency and accuracy, facilitating the development of edge intelligence.

1.1.4 Dynamic Neural Networks

Both model scaling and model pruning optimize DL models at design time. Once the optimization is completed, the model will be deployed onto hardware devices to execute in a static manner. Considering that the input data at runtime may change with time or the deployment environment, resulting in different recognition complexities between different input samples, such a static inference pattern may fail to deal with different samples efficiently and accurately. To address this limitation, a new DL model deployment paradigm, dynamic inference [67], emerges to efficiently optimize the model efficiency and accuracy at runtime. Specifically, for different input images, dynamic inference will adaptatively change the network architecture or the preprocessing of images to achieve better efficiency and prediction

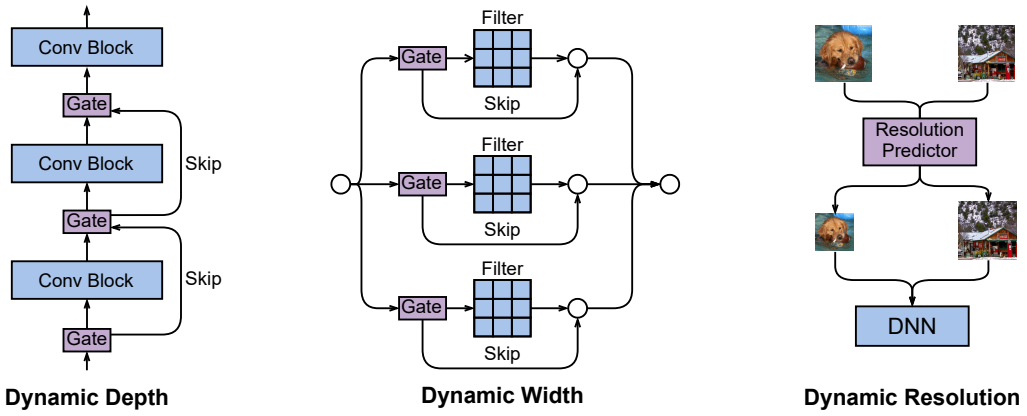


FIGURE 1.5: Demonstration of different dynamic neural networks.

accuracy for each image. For example, for 'easy' input images that have a simple pattern, maybe only a part of the whole network will be dynamically activated to obtain the final result, which can effectively reduce the inference overhead without affecting the correctness of the result. Similar to model scaling and model pruning, such dynamic adaptation of DNNs can be conducted on different dimensions (e.g., depth, width, and resolution) [65, 66, 68–73]. As shown in Figure 1.5, for different images, dynamic image preprocessing can flexibly change the image size [65] or selectively crop different patches [66] for inference, which efficiently removes the spatial redundancy in input images and optimizes the run-time computational and memory efficiency of DNNs. Inference with dynamic depth, however, tends to optimize efficiency and accuracy by dynamically controlling the execution of layers for different input images [68, 73]. More specifically, dynamic depth can be achieved by the early-exit scheme and layer skipping. Early-exit DNNs [68, 69, 74–76] employ multi-branch architectures or intermediate classifiers to implement dynamic inference, where simple images may exit at a smaller branch or early intermediate classifier without executing remaining computation. As a comparison, layer skipping [73, 77–80] achieves greater flexibility by skipping arbitrary intermediate layers. By skipping intermediate layers or omitting deeper layers during inference, dynamic depth actually modifies the computing graph and shortens the execution path of DNNs at runtime, optimizing inference efficiency. Dynamic width [81] can operate at multiple granularities: 1) neuron skipping, 2) channel skipping, and 3) branch skipping. Neuron skipping [81–83] implements dynamic inference by selectively activating different neurons in fully-connected layers. For CNNs, the

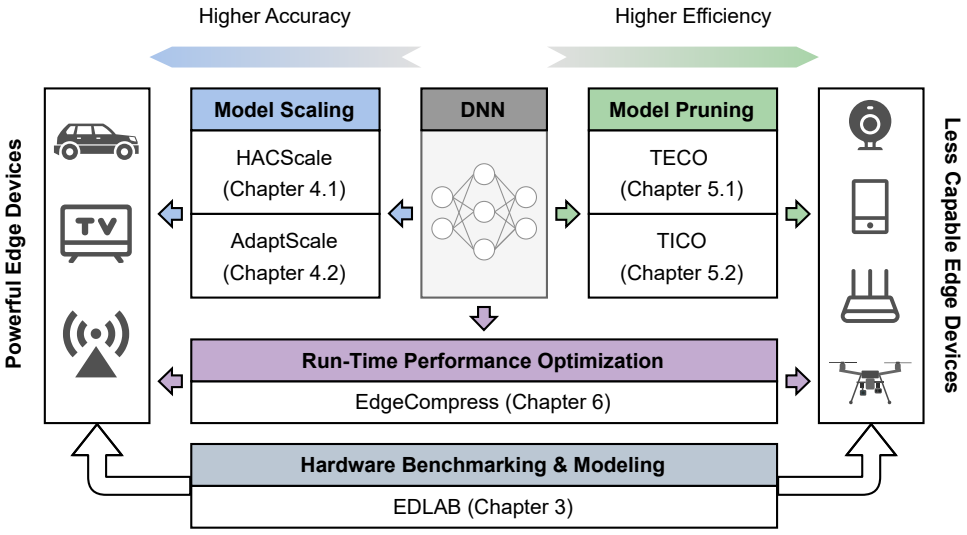


FIGURE 1.6: Overview of the thesis.

majority of computation comes from convolutional layers. Therefore, channel skipping [84, 85] proposes to selectively skip the computation of some unimportant channels for the current input sample for higher efficiency. While branch skipping [71, 72] is mainly proposed for multi-branch DNNs to improve their run-time efficiency. Different dynamic inference approaches effectively skip the sample-wise redundant computation and parameters of DNNs, improving the efficiency of the backbone network. However, the identification of redundant units (e.g., channels, layers) for different input samples inevitably incurs additional costs, and sometimes the identification costs can even far exceed the saved costs on the backbone. As a result, the actual performance of the whole system may deteriorate obviously. Therefore, how to reduce the identification costs of dynamic inference remains an open issue.

1.2 Major Contributions

To address the above problems, in this thesis, we propose multiple approaches to efficiently and flexibly adapt the overhead of DNNs to fit them into various edge devices. The overview of this thesis is demonstrated in Figure 1.6, where our main contributions can be stated as follows:

- *Hardware Evaluation*: We first propose a novel benchmark methodology accompanying an out-of-the-box benchmark suit, dubbed EDLAB, to evaluate Edge Deep Learning Accelerators (EDLAs) in a more comprehensive way. EDLAB integrates data preprocessing, model preprocessing, model deployment, model execution, and performance analysis into an automated benchmarking process, which enables users to efficiently and conveniently evaluate various EDLAs and quickly prototype their products. Meanwhile, we develop a new DL workload, the parameterized model, to exhaust EDLAs to detect their performance and resource constraints, which helps users understand the potential of different EDLAs and choose the appropriate DL workload for better results. EDLAB also plays a critical role as a hardware evaluation and modeling tool in our subsequent works on software and hardware co-design.

- *Model Scaling*: According to the hardware evaluation results, for powerful edge hardware devices with redundant resources, we propose two model scaling frameworks: 1)HACScale and 2)AdaptScale, to efficiently scale up tiny DL models for better accuracy while not sacrificing execution efficiency. HACScale investigates the impact of scaling different dimensions of DL models on accuracy and execution efficiency, and then collaboratively scale multiple dimensions for the best trade-off between accuracy and efficiency. Further, AdaptScale proposes an adaptive scaling framework that can efficiently customize scaling strategies for different models to achieve higher efficiency and accuracy, which greatly reduces the cost of designing the optimal scaling strategy for different models.

- *Model Pruning*: For resource-constrained edge devices, we propose two model compression frameworks: 1)TECO and 2)TICO, to comprehensively reduce redundant computation and parameters for higher efficiency and less resource consumption. TECO comprehensively investigates the redundancy in the three dimensions (depth, width, and resolution) of models, and then collaboratively compresses multiple dimensions for higher efficiency while maintaining model accuracy. Different from TICO, TECO further considers the impact of pruning different dimensions on both training and inference efficiency, and then proposes to jointly prune different dimensions for both higher training efficiency and inference efficiency. Meanwhile, TICO also introduces

a resolution-adaptive training strategy to training models with different image sizes at different epochs, which further improves the training efficiency of DNN models.

- *Run-time Model Optimization*: Besides optimizing the model efficiency and accuracy at design time, we also propose a dynamic inference framework dubbed EdgeCompress, to improve the model efficiency and accuracy at run-time. EdgeCompress first utilizes the aforementioned multi-dimension model compression to comprehensively compress a given model to different complexities. Then, EdgeCompress introduces a novel dynamic image cropping strategy to selectively crop the most discriminative foreground area and discard redundant background pixels, which greatly reduces redundant computation in images. Finally, different cropped images will be sent to different models according to their recognition difficulty for efficiency inference. EdgeCompress not only removes redundant parameters and computation in models but also dynamically chooses the most appropriate model for each input image, improving the run-time efficiency of DL inference significantly.

1.3 Outline of the Thesis

Chapter 1 introduces the background and current research progress in resource-efficient DNN design for emerging edge DL hardware. We then summarize our solutions and novel contributions to this research field.

Chapter 2 reviews the most cutting-edge research progress in resource-efficient DNN design, including model compression, model scaling, dynamic neural networks, etc.

Chapter 3 details the design of our edge DL hardware evaluation benchmark EDLAB. Besides, we also disclose the evaluation results of multiple EDLAs obtained by EDLAB.

Chapter 4 presents our multi-dimensional model scaling frameworks in detail, including the motivations, methodologies, and experiments on multiple datasets.

Chapter 5 introduces our multi-dimensional model pruning frameworks in detail, including the motivations, methodologies, and experiments on popular benchmarks.

Chapter 6 introduces EdgeCompress, a dynamic inference framework, which consists of a novel dynamic image cropping algorithm, a compound model compression strategy, and a dynamic inference mechanism.

Chapter 7 summarizes the methodologies of this thesis and discusses some future directions.

Chapter 2

Literature Review

In this chapter[1], we mainly review the models and techniques proposed for computer vision applications and there are two reasons for this. First, computer vision models are currently the major application for edge intelligence systems. Second, although some recent studies start to investigate how to design lightweight NLP models for edge devices like, [86–88], they are far from mature unlike models and techniques for computer vision tasks. Hence, in this chapter, we use DNN models for computer vision as the major example for our presentation. It is also worth noting that although the emerging 3D DNNs are demonstrating their huge potential in some future edge intelligence systems, like self-driving cars, and virtual reality/augmented reality systems, the majority of edge intelligence research still focuses on the 2D DNN models. Therefore, we use 2D CNNs as an example to present the preliminaries. We refer, readers who are interested in 3D CNNs, to [89]. Even though we only focus on computer vision tasks in this section, many of the techniques and models we investigated are transferable to other tasks, such as natural language processing. For example, network pruning techniques, including both structured pruning and unstructured pruning, can also be applied to Transformer models, because there are also many weight matrices in the Transformer architecture, and different weights have significantly different contributions to the final performance. Therefore, similar to compressing computer vision models, we can also compress the models for other applications by removing unimportant parameters and computation [90].

[1]The work in this chapter has been published in [13].

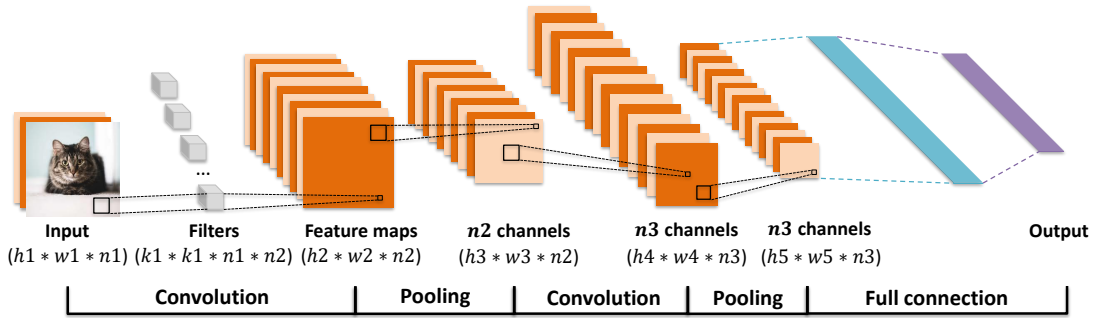


FIGURE 2.1: Overall architecture of a DNN model

2.1 Preliminaries

Before reviewing related works, we first introduce some fundamental knowledge of DNNs and edge systems to better understand the techniques and approaches discussed in subsequent sections.

2.1.1 Deep Neural Networks

Figure 2.1 shows a simple CNN with two convolutional layers and one fully connected layer. The convolutional layer is the core ingredient in CNNs, which extracts the patterns/features from input data at different granularity. Convolutional layers account for the major resource and time cost of a CNN model. To better illustrate how the convolutional layer works, we give some terminologies about convolutional layers as follows:

- **Kernel** – A kernel is a 2D square matrix with size like 1×1[2], 3×3, 5×5, 7×7 and a convolutional operand.
- **Filter** – A filter is a collection of kernels with size of $k \times k \times c$, where k is the kernel size and c is equal to the number of input channels. A filter is convolved with all input channels to derive a new channel/feature map/activation map. For a single-channel layer, the filter is the same as the kernel. For multi-channel layers, the filter is a collection of kernels. Kernels inside the filter are applied to corresponding channels of the input to generate an output feature map.

[2] 1×1 is usually called point-wise convolutional kernel

- **Feature map/activation** – Feature map and activation have the same meaning in CNNs, which denote the output tensor generated by a convolutional block. Now, many hand-crafted CNN models are designed by stacking the same convolutional block, where a block contains a couple of operations. Usually, the output activation of a convolutional block is a 3D matrix, which is the output of the previous block, and at the same time, they are also the input of the next block.

As demonstrated in Figure 2.1, the input of the first convolutional layer is an image with n_1 channels, and each channel is a 2D array with height h_1 and width w_1. Correspondingly, there are n_1 kernels in each filter, and the kernel size is $k_1 \times k_1$. There are n_2 filters, and thus n_2 feature maps will be generated after the first convolution operation and the size of the first feature maps is $(h_2 \times w_2 \times n_2)$, where h_2 and w_2 are the height and weight of the first feature maps, respectively. If the padding is used during convolution, then $h_1 = h_2$ and $w_1 = w_2$. Otherwise, $h_1 > h_2$ and $w_1 > w_2$. For more details, interesting readers are referred to [43].

Usually, a convolutional layer is followed by a pooling layer and an activation layer. In Figure 2.1, we omit activation layers. Activation functions are used to introduce non-linearity into a neural network, and the common activation functions are rectified linear units (ReLU)[91], sigmoid, etc. The pooling layer down-samples the feature map to reduce the spatial size of the feature map and increase its receptive field. The max pooling [92] and the average pooling are the two common pooling methods. A DNN model usually ends up with a couple of fully connected layers, which are used to fuse the feature information from the last convolutional layer and predict the classification of the input image. To improve the performance and training speed, modern DNN models also have other operation layers, like batch normalization layer [93], squeeze-and-excitation layer [16], etc.

2.1.2 Edge Intelligent Systems

As widely known, traditional computing paradigms like cloud computing are highly centralized where the requested tasks are first gathered from the outermost edge and then transmitted to the remote computing center for processing. This has been dominating the last several decades since most of the devices located at

the network edge are computation-limited, which are not capable of processing those computation-intensive tasks on their own. As a result, the latency overheads of transmitting tasks become non-trivial, which significantly hurts processing efficiency. Although some standard data compression techniques, such as image compression and video compression, can reduce the amount of data transmitted, they fail to address the most critical network issue. When the network condition is not good, even a small amount of data will cause an extremely high latency, which will bring catastrophic consequences to latency-critical applications such as autonomous driving. With the miniaturization of advanced processing and storage technologies, the computational gap between resource-limited edge devices and computation-intensive tasks has been alleviated, thereby making the edge computing paradigm more and more mainstream, e.g., edge systems in robot edge systems, unmanned aerial vehicles (UAVs), mobile edge systems, etc. In practice, the edge computing paradigm brings two main benefits. On the one hand, since we do not need to distribute the tasks to the remote cloud, the runtime latency greatly decreases, thereby significantly improving the quality of service (QoS) and quality of experience (QoE) [94]. It is worth noting that runtime latency is of great essence in real-world scenarios like autonomous driving, which have rigid latency requirements. On the other hand, the security and reliability of processed tasks are significantly improved. In practice, when you feed all of your data to the remote cloud center for analysis, the crucial data visibility will suffer from potential attacks, thereby being highly vulnerable, i.e., when you distribute your data to be analyzed, you distribute the risks as well. Previous security algorithms, like encryption, can protect data privacy from being attacked during the transmission process, but the risk of data privacy leakage also exists during the inference stage at cloud servers. When data reach cloud servers, they usually need to be decrypted before they can be processed normally, which may also lead to privacy leakage. The emerging edge computing paradigm can easily address the above potential attacks since all the tasks are processed locally at the network edge, i.e., all the related data is only visible to local edge devices.

Furthermore, in the past few years, deep learning techniques especially DNNs have demonstrated great success in many real-world applications like image classification [15, 41, 95], object detection [96, 97], natural language processing [87, 98], etc. However, such success comes with extremely high computation resources. To tackle this, several deep learning-driven intelligent edge devices like Nvidia Jetson Series

[19, 20] and Intel Neural Computing Stick [21] have been well-designed. Nonetheless, due to the limited resource budgets like power consumption, deploying those state-of-the-art DNNs to the edge devices is still of great challenge.

In traditional cloud computing, there are usually sufficient hardware resources, thus the resource consumption of applications is less considered and improving the QoS and QoE become the main optimization goals. However, in edge intelligent systems, the hardware resources are very limited, so resource consumption also needs to be considered. Specifically, in edge AI systems, the inference latency (i.e., response time), the throughput (i.e., frame rate), the energy consumption, the memory consumption, and the prediction accuracy are the most commonly considered optimization metrics. The inference latency reflects the performance of an edge system to sequentially process data, which can be evaluated by setting the inference batch size to 1. Different from inference latency, the throughput can indicate the capability of an edge system to process data in parallel, which can be evaluated by batch inference. Meanwhile, energy consumption and memory consumption are to measure the resource consumption of edge systems. Finally, prediction accuracy focuses on evaluating the software performance, such as the classification accuracy for classification models and detection accuracy for detection models. All these metrics are critical for comprehensively evaluating an edge AI system.

2.2 Deep Learning Hardware Benchmarking

Nowadays, DNNs have been widely deployed in various applications. To efficiently handle the huge computation of DNNs and improve the application performance, many deep learning accelerators are designed to speed up the execution of DNNs. As there are more and more DL accelerators, efficiently evaluating different DL accelerators and selecting the appropriate accelerators for different applications are getting more difficult. To address these issues, many DL benchmarks are introduced to efficiently and comprehensively evaluate the performance of various emerging DL hardware devices [99–104].

MLPerf training benchmark [100] is proposed to evaluate the training performance of different kinds of DL accelerators. Instead of directly evaluating the hardware,

it only specifies the DL workloads (i.e., models), datasets, and benchmarking rules, and then asks the hardware vendor to implement evaluation and report the evaluation codes and results to it. Finally, the benchmark will check the evaluation results to ensure their correctness. The main novelty of this benchmark is that it splits the whole benchmark into two divisions: 1) the closed division and 2) the open division. In the closed division, the main purpose is to directly reflect the performance of various hardware platforms or software frameworks in common scenarios and to compare their performance strictly and fairly. Thus, no performance optimization (e.g., quantization) is allowed in this division. While the open division is intended to foster innovation and fully explore the hardware potential for higher performance, and thus it allows the vendor to use different optimizations to improve the hardware performance.

MLPerf inference benchmark [99] is introduced to evaluate the inference performance of various DL hardware. Similar to MLPerf training benchmark [100], this inference benchmark also has both open division and closed division. To be more comprehensive, MLPerf inference benchmark [99] contains multiple DL tasks (e.g., image classification, object detection, medical imaging, speech-to-text, natural language processing, and recommendation), multiple models (e.g., ResNet, RetinaNet, and BERT), multiple metrics (e.g., peak hardware performance, power consumption, and memory consumption), and multiple inference scenarios (e.g., server inference, offline inference). Moreover, to ensure the correctness of submitted evaluation results and fairly compare different hardware, it also designs a load generator to check whether the submitted code is modified.

Even though the MLPerf training benchmark [100] and MLPerf inference benchmark [99] can comprehensively evaluate the performance of different hardware, they highly rely on the vendors to implement the evaluation process and reports results. It is still difficult for users to use these benchmarks as tools to evaluate different hardware on their own. To provide easy tools for users to evaluate their hardware, many out-of-the-box benchmarks are introduced [101–104]. These benchmarks provide both the benchmarking methodology and codes to users so that they can efficiently evaluate their own hardware devices. Specifically, Ignatov *et al.* [102] propose AI Benchmark, which includes multiple AI tasks, such as image classification, face recognition, image enhancement, etc. Users can download and execute the benchmark tool on their own devices and get the evaluation score.

Currently, AI Benchmark mainly supports various smart mobile phones and less considers other types of DL hardware. By contrast, AI Matrix [103] and DeepBench [104] consider various heterogeneous DL accelerators, such as CPU and GPU. AI Matrix [103] and DeepBench [104] are similar in terms of the target hardware platforms, DL applications, and the benchmarking methodology. To be more specific, both benchmarks split the benchmark into the macro benchmark and the micro benchmark. In the macro benchmark, the DL workloads are some commonly utilized DL models, which aim to measure the hardware performance while running real DL applications. On the contrary, the micro benchmark is composed of several basic operations of DL models, such as the convolutional layer, the fully-connected layer, the recurrent layers, and the activation layer, etc. The micro benchmark is intended to analyze the hardware capabilities for different DL models at a finer granularity. According to the results of the micro benchmark, researchers are able to accurately locate the performance bottleneck of different DL models on the given hardware platforms, and optimize the model architecture in a targeted manner for higher execution performance on the hardware.

Different from all mentioned benchmarks above that only focus on evaluating the hardware performance, DAWNBench [101] also quantifies training and inference costs in dollars, which gives users an intuitive demonstration of how much they would spend if they want to deploy a DL model on a commercial cloud server.

2.3 Deep Learning Model Scaling

DNNs have been widely utilized in many edge applications to improve service quality [13]. To better accommodate the resource consumption and improve the on-device execution efficiency of various DNNs, edge DL hardware devices have been evolving for higher capabilities. These emerging edge DL hardware are usually equipped with more hardware resources like memory and computing units, which can process more operations in parallel and thus execute larger models in a shorter time and with lower power consumption. For example, NVIDIA Jetson AGX Xavier [17], an embedded GPU platform, can provide 32 TOPS peak performance with only 30W power consumption. As such, more advanced DL models can be deployed in edge environments to improve the service of applications. However, this also results in the fact that previous lightweight DL models designed for less

capable edge devices, such as SqueezeNet [27] and ShuffleNet [26], cannot fully utilize the hardware resources. It is universally recognized that the model accuracy is highly correlated to the model size: the larger the model size, the higher the accuracy. Underutilizing hardware resources may lose the opportunity to further improve model accuracy.

To address this problem and enable previous lightweight models to utilize available hardware resources efficiently, many methods are proposed to increase the parallelism of DNN inference for better hardware utilization, thereby improving inference accuracy. Specifically, some works [105] propose executing multiple models in parallel for better accuracy, but this paradigm has relatively high requirements on hardware resources (e.g., multiple GPUs), thus it is usually used in cloud applications. Besides, it lacks enough flexibility to cope with edge devices with different resources. On the other hand, some works focus on improving the data-level parallelism, such as batch inference. However, they can only optimize some performance metrics like throughput, while the inference accuracy cannot benefit from this. To bridge this gap, model scaling is proposed to flexibly scale up existing models for higher accuracy while not sacrificing the execution efficiency on edge devices. Model scaling can mainly be performed on three dimensions of DNNs: 1) depth, 2) width, and 3) resolution. The depth of a DNN denotes the number of layers in the network architecture, the width of a DNN represents the number of kernels or channels in each layer, and the resolution dimension means the size of input images.

Depth scaling [15, 42, 47, 106] focuses on scaling up the depth of DNNs by stacking more layers in the network architecture to construct 'deeper' networks. As discussed in [42], deeper neural networks are more capable of extracting high-level semantics and recognizing difficult images correctly, improving model accuracy. However, deeper neural networks are also more difficult to train due to gradient descent and gradient explosion. After this problem is addressed by [15] with residual connections, training super-deep neural networks becomes feasible, and thus more researchers even try to increase the number of layers to 1,000 [47, 106].

Width scaling, on the contrary, tries to construct shallow but wide neural networks to achieve higher accuracy. As pointed out by [41], training super-deep neural networks will cause a huge memory consumption and a very slow training speed due to a large number of intermediate results and gradients generated by each layer.

Moreover, the increased computation results from width scaling can be processed in parallel, which can fully utilize the underlying hardware resources, achieving higher accuracy while not compromising the actual execution efficiency on devices. Therefore, to achieve higher accuracy while maintaining the on-device execution efficiency, many methods are proposed to increase the width of DNNs for higher accuracy and better hardware utilization [29, 32, 41, 44, 45]. Specifically, [29, 32, 41] scales the width of all layers in a uniform manner by applying the same scaling factor to all layers. This paradigm is simple and can minimize the design cost of scaling. However, uniformly scaling all layers is inefficient because the importance of each layer to accuracy and efficiency is distinct from each other, and uniformly scaling them can lead to a sub-optimal result for accuracy and execution efficiency. To address this issue, non-uniform width scaling is proposed to scale different layers in a more fine-grained manner. Specifically, MorphNet [44] combines non-uniform layer pruning and uniform scaling to achieve non-uniform scaling, which iteratively executes pruning and scaling to attain the desired network architecture. Further, NeuralScale [45] points out that the uniform scaling algorithm used in MorphNet [44] is inefficient and can not achieve the optimal result. Instead, NeuraScale [45] proposes a novel non-uniform scaling framework, which introduces a non-uniform channel expansion algorithm and iteratively executes the channel expansion algorithm and the pruning algorithm introduced in [52] to scale DNNs.

Resolution scaling, instead of scaling the network architecture, increases the size of input images to improve model accuracy because images at a higher resolution may provide more semantic information and more fine-grained patterns to help DNNs to recognize the image correctly. Specifically, resolution scaling can be efficiently conducted with up-sampling algorithms, such as bilinear interpolation and bicubic interpolation. For example, previous DNNs usually use 224×224 as the input size for images in ImageNet [1], while InceptionV2 [49] up-samples the image size to 299×299 with bilinear interpolation for higher accuracy. Similarly, Zoph *et al.* [34] and GPipe [107] further increase the input image size to 321×331 and 480×480 to achieve the state-of-the-art classification accuracy on ImageNet [1]. In addition to image classification tasks, higher resolutions (e.g., 600×600) are also widely utilized in object detection [108, 109] and semantic segmentation tasks [110, 111].

In addition to scaling a single dimension, some works also argue that collaboratively scaling the three dimensions (depth, width, and resolution) can achieve a better

trade-off between model complexity and accuracy. Particularly, EfficientNet [40] believes that the compound scaling of multiple dimensions helps to obtain a better balance among different dimensions, thereby achieving higher accuracy under the same computational constraint. To combine the scaling of the three dimensions, EfficientNet [40] directly searches for the optimal scaling factor for each dimension under the given model computation constraint. However, the exhaustive search for scaling factors introduces a huge design cost. Meanwhile, the searched scaling strategy only considers model computation while the on-device execution efficiency is ignored. To reduce the design cost and improve the on-device efficiency of the scaled models, Dollár *et al.* [46] introduces a fast and accurate compound model scaling framework, which mainly scales width, supplemented by depth scaling and resolution scaling to improve both accuracy and on-device execution efficiency. Moreover, taking the inference efficiency on powerful cloud devices as the optimization objective, Li *et al.* [50] searches for the scaling factor of each dimension that can maximize the trade-off between model accuracy and the execution efficiency on powerful cloud devices. In summary, compared to single-dimensional model scaling, multidimensional model scaling avoids the early saturation of accuracy, achieving better trade-offs between model accuracy and model efficiency (e.g., model computation or on-device efficiency).

2.4 Deep Learning Model Pruning

Designing a novel architecture is really challenging due to its large design space and complicated parameter tuning. Unfortunately, the majority of DNN models are designed to pursue better accuracy without consideration of resource constraints of edge systems, where complex DNN models have a few hundred layers and several billions of parameters to achieve competitive accuracy. As indicated in [112], DNN models usually have significant redundancy in terms of weights and parameters. Then, an interesting question is raised: *Can we reduce the complexity of DNN models by removing these redundancies without greatly compromising their predictive performance?*

Network compression tackles this problem by removing the redundancy of over-parameterized networks. Generally, network compression techniques fall into three categories: 1) network pruning which removes the redundant weights and channels

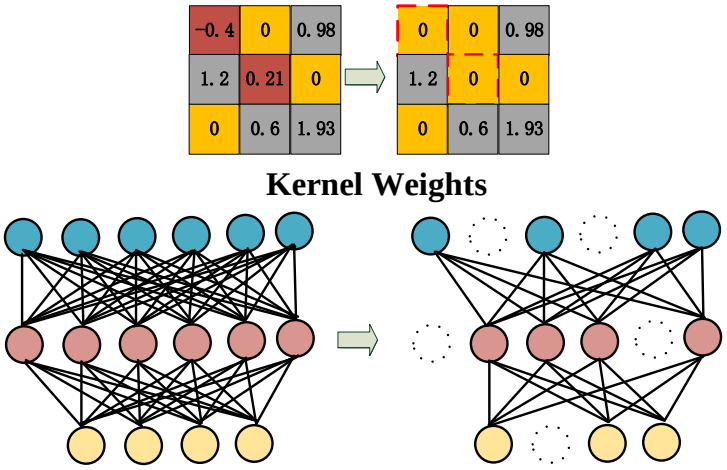


FIGURE 2.2: Unstructured pruning

of over-parameterized DNNs, 2) quantization which uses fewer bits to store DNN weights and intermediate results (e.g., from float point 32, FP32 to integer 8, INT8), and 3) knowledge distill which learns a small and compact (student) model from a large and over-parameterized (teacher) model. It is worth noting that the three approaches are not mutually exclusive to each other and in many cases, they are combined to maximally compress the redundant models. In this section, we mainly discuss network pruning as it is the most relevant to this thesis.

The main motivation behind network pruning is that DNN models are usually over-parameterized in terms of weights and channels [112], and eliminating these redundancies within the model can hugely reduce the computational complexity and storage requirement. Many network pruning methods are proposed in the past 5 years, and we like to classify them into two major branches based on the pruned structure: 1) unstructured pruning and 2) structured pruning.

2.4.1 Unstructured Pruning

Unstructured pruning, also known as weight pruning, conducts a fine-grained operation by removing less sensitive weights in over-parameterized DNN models. As shown in Figure 2.2, it removes individual weights within a kernel or individual neurons within a fully connected layer. Unstructured pruning can significantly reduce the number of parameters and memory footprint.

Unstructured neural network pruning can be traced back to the 1990s [113, 114]. Deep Compression framework [55, 56] is one of the pioneers in applying unstructured pruning to DNN models. Deep Compression uses three steps, pruning, quantization, and Huffman Encoding, to compress over-parameterized DNN models, such as VGG [42] and AlexNet [3]. The pruning step removes the weights with magnitudes lower than a threshold and corresponding connections. Then, quantization reduces the bit-width of weights to reduce the model size (More details about quantization in the subsequent section). Finally, Huffman encoding further compresses the weight storage. Experimental results show deep compression can significantly reduce the model size with no or negligible accuracy loss.

Inspired by the success of deep compression, many new methods are proposed to further improve the efficacy of unstructured pruning. Molchanov *et al.* [57] extend variational drop rate to each weight of a DNN model, and weights with high drop-rate are deemed irrelevant and thus can be removed for model compression. Different from the aforementioned approaches which conduct unstructured pruning directly on pre-trained networks, NeST [58], which is inspired by the development of human brains, adopts a grow-and-prune scheme, where NeST first makes a sparse seed DNN model bigger and more complex and then prunes some irrelevant weights from the grown model to generate the final compact model. Zhang *et al.* [59] formulate the weight pruning problem as a non-convex problem which can be solved by alternating direction method of multipliers (ADMM) method [115].

All methods discussed above take the model size as the main objective, i.e., they aim to or in literature intend to reduce memory size by removing small magnitude weights and accordingly unused neurons which have no or small impact on predictive accuracy. However, for edge devices, power, and energy are also important metrics to consider. Yang *et al.* in [116] propose an energy-aware pruning method, in which they strive to reduce the energy consumption of DNN models by unstructured pruning. The core idea behind their approach is to order layers according to their energy consumption and then it prunes weights according to that order.

Although unstructured pruning can significantly reduce memory footprint and multiply-accumulate (MACs) of DNNs, such reduction is unable to directly translate to latency improvement. This is because unstructured pruning generates sparse structures which lead to an irregular memory access pattern. The irregular pattern of sparsified DNN models needs special formats, e.g., compressed sparse row

TABLE 2.1: Comparison of unstructured pruning methods. We only show the accuracy of the most complex models used in the paper, and in "$(xx \rightarrow yy)$" "xx" and "yy" denote the original accuracy and the accuracy after compression, respectively. In addition, "I" and "C10" denote the ImageNet dataset and the CIFAR-10 dataset, respectively. "CR" denotes the compression ratio.

Methods	Year	Pruning Method	Metrics	Top-1 Accuracy (%)	CR
Han *et al.* [55]	2015	Threshold	Model size	VGG16(68.5 → 68.8), I	49×
Molchanov *et al.* [57]	2017	Drop Rate	Model size	VGG16(92.5 → 92.7), C10	48×
Yang *et al.* [116]	2017	Heuristic	Energy	AlexNet(80.4 → 79.6)*, I	11×
Zhang *et al.* [59]	2017	ADMM	Model size	AlexNet(80.2 → 80.2)*, I	21×
NeST [58]	2019	Grow-and-Prune	Model size	VGG16(71.6 → 69.3), I	30×

and compressed sparse column, to store sparse matrices. The off-the-shelf hardware and software cannot conduct efficient calculations on compressed formats, so specialized hardware and software libraries are required to execute sparsified DNN models [60, 117].

2.4.2 Structured Pruning

Structured pruning, on the other hand, prunes networks by maintaining a regular computation pattern. To keep regularity, Structured pruning completely removes some unimportant channels and filters that have less impact on the model's prediction accuracy as shown in Figure 2.3. Since structured pruning does not lead to irregular patterns, the compressed network pruned by structured pruning can directly accelerate its inference on off-the-shelf hardware platforms without specialized software library support. Therefore, it has been receiving growing attention in recent years.

The common process of structured pruning is (1) defining a pruning criterion; (2) selecting pruned channels according to the criterion and goal, such as compression ratio and the number of MACs or FLOPs; and (3) fine-tuning the pruned model, i.e., retraining the pruned network to retain accuracy. Works in structured pruning propose different criteria and methods to select and prune channels while minimizing accuracy loss. In terms of pruning methods, we can classify them into two categories: 1) training-based and 2) inference-based.

- **Training-based**: The pruning method is conducted during the training procedure, where a sparsity constraint is exposed and a compact network is directly learned from a big and over-parameterized network.

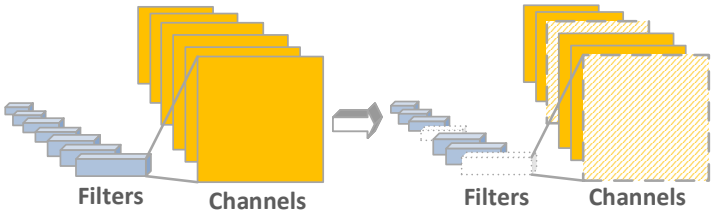


FIGURE 2.3: A simple visualization of structured filter pruning

- **Inference-based**: The pruning method is conducted on a pre-trained model, where a defined pruning rule is applied.

In terms of pruning scope, we have local pruning vs global pruning.

- **Local pruning**: The pruning process is conducted layer by layer and each layer may have a different pruning constraint or target, e.g., compression ratio.
- **Global pruning**: The pruning process is applied to the whole network. All channels/filters from different layers are sorted together in terms of a global ranking method and channels are removed according to a defined target.

The pruning process is either rule-based or learning-based.

- **Rule-based**: The pruning is conducted according to some defined rules, like heuristic algorithms.
- **Learning-based**: The pruning is conducted by a learning algorithm, such as reinforcement learning [118], evolutionary algorithms [119], and gradient-based optimization.

Li *et al.* [120] adopt a layer pruning method where filters within each layer are sorted in terms of absolute weight sum and then filters with low magnitude are pruned according to a pruning ratio and related channels are all removed. He *et al.* [63] present a layer pruning method using LASSO regression and reconstruction error to select pruning channels. Hu *et al.* [62] observe that a fraction of activation weights in DNNs are zero and these zero weights imply the corresponding filters are likely to be redundant and can be pruned, and they thus propose using Average

TABLE 2.2: The summary of structure pruning methods. We only show the accuracy of the most complex models used in the paper, and in "$(xx \rightarrow yy)$" "xx" and "yy" denote the original accuracy and the accuracy after compression, respectively. In addition, "I", "M", and "C10" denote the ImageNet dataset, MNIST dataset, and CIFAR-10 dataset, respectively. For some methods, they do not show the compress rate, but provide the acceleration times of the compressed FLOPs. "CR" denotes the compression ratio.

Method	Year	Type	Scope	Process	Top-1 Accuracy (%)	CR
Li *et al.* [120]	2016	Inference	Layer	Rule	VGG16(93.3 → 93.4),C10	2.78×
Hu *et al.* [62]	2016	Inference	Global	Rule	VGG16(69.3 → 70.4),I	2.59×
SSL [121]	2016	Training	Layer	Rule	LeNet(99.1 → 99.2),M	11.9× FLOPs
He *et al.* [63]	2017	Inference	Layer	Rule	VGG16(89.9 → 89.6)*,I	5× Acceleration
ThiNet [122]	2017	Inference	Layer	Rule	VGG16(89.9 → 87.2)*,I	5× Acceleration
DeepIoT [123]	2017	Training	Layer	Learing	VGG16(90.6 → 90.6),C10	40.98×
DCP [124]	2021	Inference	Layer	Rule	VGG16(94.0 → 94.6),C10	15.58×
SFP [125]	2018	Training	Layer	Rule	ResNet101(77.4 → 77.5),I	1.73×
Huang *et al.* [126]	2018	Training	Global	Rule	VGG16(93.9 → 93.9),C10	1.5×
AMC [127]	2018	Inference	Layer	Learning	VGG16(70.5 → 69.1),I	5× FLOPs
NetAdapt [128]	2018	Inference	Layer	Learning	MobileNetV1(68.8 → 69.1),I	1.15× FLOPs
You *et al.* [129]	2019	Inference	Global	Rule	ResNet56(93.1 → 93.1),C10	3×
LFPC [130]	2020	Inference	Layer	Learning	ResNet56(93.6 → 93.2),C10	2.13×
HRank [53]	2020	Inference	Layer	Rule	VGG16(94.0 → 93.4),C10	5.85×

Percentage of Zeros (APoZ) of a filter as the criteria to select pruned channels. Instead of using the information from the currently pruned layer for selecting pruned channels, ThiNet [122] proposes to exploit the information from the output of the next layer to determine pruned filters. Discriminate-aware channel pruning (DCP) in [124] relies on the discrimination of each filter to select the pruned channels. The main concept is that the discriminated channels provide more relevant information or features to retain accuracy and then the channels which are inadequately discriminated can be pruned for complexity reduction. You *et al.* [129] propose a gate decorator module to replace the batch normalization module in DNNs to select pruned filters globally. Most structured pruning methods adopt a uniform pruning criterion for all layers, but different layers have different functions, thereby likely benefiting from employing different pruning criteria at different layers. Recently, He *et al.* [130] propose LFPC to learn an optimal pruning criterion for each layer by using a gradient-based method. HRank [53] empirically finds that filters with a low rank are less informative than those with a high rank, and thus uses this observation to prune the unimportant channels.

Wen *et al.* [121] propose SSL to have a training-based pruning method, learning a compact and sparse model from a pre-trained model. Liu *et al.* [61] propose an approach called network slimming, which takes wide and large networks as input models, but during training, insignificant channels are automatically identified and

pruned afterward. Soft filter pruning (SFP) [125] deploys a training-based pruning method to prune a complex network. Huang *et al.* [126] propose the concept of sparsity scaling factor for each filter which is learned during training, and then filters with scaling factor 0 are removed. Yao *et al.* [123] train a recurrent neural network (RNN) to determine the dropout probability of each filter and prune the network layer by layer according to drop probability.

Most of the above-mentioned structured pruning methods are rule-based, i.e., some heuristic algorithms devised according to their own criterion. On the contrary, some works employ learning algorithms to automatically prune network models. AMC [127] proposes to use reinforcement learning to automatically prune channels of each layer. Anwar *et al.* [131] prune a DNN model at the feature level, kernel level, and intra-kernel level (i.e., weight), where they deploy evolutionary algorithm to find the best combination of different pruning granularities. However, searching in a discrete space using reinforcement learning (RL) and evolutionary algorithms (EA) is really costly, so the gradient-based method is recently proposed to find an optimal pruning criterion for each layer [130].

To determine how many channels should be pruned, the above-mentioned works use indirect metrics like FLOPs or compression ratio to prune networks. Nevertheless, the reduced FLOPs and compression ratio cannot directly translate to performance improvement. In addition, a diverse of hardware accelerators have emerged for boosting the execution of DNNs, but various systems demonstrate different capabilities to handle network complications. Hence, some pruning studies directly target the direct metric upon specific hardware, e.g., latency. NetAdapt [128] proposes an automated framework to prune filters in different layers such that the pruned model can be adapted to a target platform. To optimize the latency on a target platform, NetAdapt builds up a look-up table (LUT) for different operations and layers, so instead of measuring latency on the real platform, it can quickly estimate the latency based on the model architecture and LUT. Yu *et al.* [132] introduce a SIMD-aware pruning framework that employs different pruning strategies for different underlying hardware, like weight pruning for low-parallelism CPU and filter pruning for high-parallelism.

2.4.3 Other Model Compression Techniques

Quantization: Network pruning can reduce the complexity of DNN models by removing redundant parameters. However, the state-of-the-art models have more than billions of parameters. During inference, a model produces a large portion of intermediate results (activation/feature maps) which usually occupy a large memory space. As a result, the huge memory requirement prohibits DNN models from being implemented on memory-limited edge devices [17, 19, 20]. For example, ResNet-50 [15] has 26 million parameter weights, generates 16 million activations in one inference, requires around 168 MB of memory space, and needs at least 3GB/s memory bandwidth. It is not difficult to see that it is unlikely to deploy these state-of-the-art models to edge devices that have limited storage and computational resources. In this case, *quantization* [55, 133–137] becomes a promising approach to address the aforementioned issue. Quantization indicates the processing of mapping values from a large range into values within a small range. By using quantization, values or numbers need less memory to store. The weights and activations of DNNs are all in floating point format and usually need 32-bit to represent one number, i.e., FP32. However, we can use other formats, e.g., FP16, INT8, and binary, to represent the weights and activations. Some early work has shown that using FP16 to train DNN models can reduce the computational cost while retaining accuracy [138]. Quantization significantly benefits DNN models on resource-limited devices, and it is capable of fitting the whole model into the on-chip memory of edge devices such that the high overhead caused by off-chip memory access can be mitigated. In addition, since operations with low-bit representation usually consume less energy and execute faster, quantization reduces energy consumption and latency as well on some hardware platforms [139].

Knowledge Distillation: Knowledge distillation is another technique to conduct model compression, where a more compact student model can learn the knowledge from a complicated and powerful teacher model. Bucila *et al.* [140] first propose the concept of knowledge distillation, and Hinton *et al.* [141] generalize knowledge distillation and apply it to DNNs. After [141], many efforts are made towards improving the performance of knowledge distillation. The work in [142, 143] extends the number of teacher models from one to multiple. However, there exist performance differences among those teacher models. To tackle this issue, they

propose to assign different weights for each teacher model, and then weighted-average probability distributions from different teachers are applied to supervise the student model. Combined with quantization, Polino *et al.* [144] introduce the *quantized distillation*, which leverages distillation during the training process by incorporating knowledge distillation loss. Ravi [145] introduces a neural projection approach to design and train efficient on-device neural networks. Before the prediction, input instances are transformed into binary representations, which significantly reduces memory consumption. Then, the prediction weights are learned by knowledge distillation to achieve higher generalization capability. In [146], Li *et al.* propose to use knowledge distillation to efficiently compress models, where the uncompressed model and compressed model are considered as a teacher-student pair. This new method can avoid the time-consuming fine-tuning after pruning and achieve data efficiency. The above works only use the knowledge from the outputs of the last layer in the teacher model. *Can the intermediate knowledge help to obtain a better model?* Remero *et al.* [147] adopt knowledge distillation to train a compact model, namely FitNets. The main idea in FitNets is to train a deeper and thinner student with the knowledge transferred from the shallower and wider teacher model. Different from the previous works, the knowledge in FitNets is not only from the final outputs but also from intermediate feature representations of the teacher model. By doing so, the student model in FitNets mimics or imitates the teacher model from different granularity levels. Similarly, Zagoruyko *et al.* [148] introduce the attention transfer strategy to mimic the attention maps of a powerful teacher network, which proves to improve the performance of the student network.

2.5 Dynamic Neural Networks

Traditional DNNs are usually executed in a static manner [3, 15, 29, 32, 40, 47]. In other words, the network architecture, model parameters, and the input image size of a model are determined and fixed before it is deployed onto the target platform. During inference, all images will be resized to the same size, and go through the same computational graph to obtain the inference results. However, Han *et al.* [67] points out that, because different images have different patterns and are with different recognition difficulties, processing all images with the same

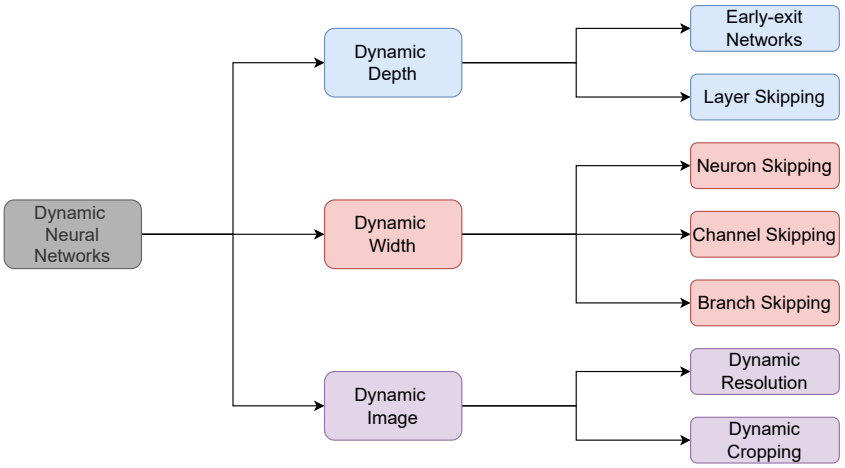


FIGURE 2.4: Classification of different dynamic neural networks.

computational graph and parameters may limit the execution efficiency and representation power of DNNs. To address this issue, dynamic neural networks are proposed to optimize the execution efficiency and representation power of DNNs [65, 66, 68, 69, 77, 84, 149]. According to the input sample, dynamic neural networks can flexibly adjust different dimensions of DNNs, such as layers, channels, or image resolution, to balance the trade-off between model execution overhead and prediction accuracy. As shown in Figure 2.4, based on the dimension, dynamic neural networks can be divided into three categories: 1) dynamic depth, 2) dynamic width, and 3) dynamic resolution. Related works under each category are introduced in detail as follows.

2.5.1 Dynamic Depth

The main characteristic of neural networks with dynamic depth is that they can flexibly execute different layers according to different input images [67]. There are mainly two ways to achieve dynamic depth: 1) early exiting and 2) layer skipping. layer skipping achieves dynamic depth by selectively executing important layers while skipping unimportant layers for each input image [73, 77, 78]. As a comparison, early-exit neural networks will go through layers sequentially without skipping any layer, but they may terminate the inference immediately at intermediate layers once a confident prediction is obtained [68, 69, 150]. In other words, simple images will exit at early layers without executing deeper layers. By doing so, the computational complexity of simple images can be reduced significantly.

Early-exit networks [68, 69, 150] are proposed based on the fact that different images correspond to distinct recognition complexity. Some "simple" images may have a clear and large foreground, which is easy to be correctly classified even by a small neural network. Therefore, executing the whole network for such images is unnecessary and inefficient. To address this problem, Park *et al.* [75] cascades two models with different numbers of layers for inference. An input image will first be fed into the shallower network to obtain its *softmax* output, and then a decision on whether the inference should exit at the shallower model will be made according to the prediction confidence score. Once the confidence score exceeds the preset threshold, the inference will be terminated immediately and the deeper network will not be executed. Further, Bolukbasi *et al.* [150] and Wang *et al.* [76] cascade multiple neural networks with different depths, and a classifier is placed at the end of each network. Once the prediction of a neural network is considered confident, the inference will be terminated without executing the following larger networks, thereby avoiding unnecessary computational overhead. Instead of cascading multiple neural networks, [68, 151–153] insert multiple intermediate classifiers into a single backbone network for dynamic inference, where the inference may exit at early classifiers without executing subsequent layers, thereby optimizing the trade-off between model overhead and accuracy performance.

Layer skipping [73, 77, 78] is of more flexibility than early exiting. Early-exit networks actually skip all subsequent layers, while layer skipping is capable of selectively skipping arbitrary intermediate layers and exiting at the final classifier. In this way, no additional intermediate classifier is needed, and thus the overhead is further reduced. To efficiently determine whether an intermediate layer should be skipped or not, Graves *et al.* [77] introduce halting score in RNNs for natural language processing applications, which uses an accumulated scalar to determine if the next block should be executed or skipped. Besides RNNs, Figurnov *et al.* [79] generalize the halting strategy to CNNs (e.g., ResNet [15]) for vision tasks by viewing residual blocks within a ResNet stage as linear layers within a step of RNNs. Furthermore, Dehghani *et al.* [80] exploit the halting strategy in Transformers [154] to implement dynamic inference for higher efficiency.

In addition to the halting scheme, there is another paradigm of layer skipping utilizing a gating function to control the execution of each intermediate layer [73, 78, 155]. Since it can be performed in a plug-and-play manner, the gating function

is widely utilized in different kinds of neural networks. For example, SkipNet [73] and Conv-AIG [78] introduce the gating function to CNNs, where both approaches exploit lightweight computation to generate a binary scalar to decide whether to execute or skip the corresponding convolutional layer. Specifically, SkipNet [73] utilizes RNN as the gating function to generate the skipping decision for each layer, while Conv-AIG [78] inserts fully-connected layers behind each block to attain the decision. Instead of skipping convolutional layers in CNNs, Guo *et al.* [155] exploits the gating function in recursive neural networks. In [155], one block in a stage may be executed repeatedly with parameter sharing, and the gating function will be utilized to dynamically determine the recursive depth of the network for higher run-time efficiency.

2.5.2 Dynamic Width

Different from the dynamic depth that directly ignores all computation of the skipped layer, dynamic width implements dynamic inference in a more fine-grained manner, where all layers will be executed sequentially, and only part of the computation of each layer will be skipped. According to the skipped unit, dynamic width can be mainly divided into three categories: 1) neuron skipping, 2) channel skipping, and 3) branch skipping.

Fully-connected layers account for a considerable portion of the computational cost of a network [55]. The computation of a fully-connected layer mainly depends on the number of input neurons and output neurons, and we all know that different neurons are usually responsible for different patterns. Since different images have distinct patterns, it is unnecessary and inefficient to activate all neurons in fully-connected layers for every input image. To avoid activating redundant neurons in fully-connected layers, [81–83] propose using lightweight auxiliary branches to efficiently identify the neurons to activate for each input image. Meanwhile, Davis *et al.* [156] achieves the same goal with low-rank approximation.

Similar to the neurons in fully-connected layers, different channels in convolutional layers also represent different features, and directly activating all channels for every input image can bring significant computational redundancy. To enable dynamic inference in the channel dimension, CGNet [84] first performs computation on

some of the channels in each layer, and then it will selectively activate the remaining channels based on the input sample. S2DNAS [85] searches for multiple sub-architectures with diverse widths (i.e., the number of channels), and once a sub-architecture obtained a confident result, the inference will be terminated immediately without executing the complete network. Instead of determining the dynamic execution of channels according to the last output, another paradigm uses gating functions to control the execution of different channels in each layer [70, 149, 157, 158]. Specifically, RPN [149] utilizes the Markov decision process to dynamically prune channels at runtime. More recently, Li *et al.* [70] introduce a gate module to flexibly identify the optimal width for each residual block.

Branch skipping is usually exploited in multi-branch neural networks [159] to reduce the inference overhead. In conventional multi-branch networks [159, 160], multiple branches are cascaded and executed in parallel. The features obtained from different branches are scaled and ensembled via data-dependent weights. However, in this way, all branches will be executed, and this will bring a significant increase in computation. To improve the efficiency of multi-branch networks at test time, hard gates are introduced to dynamically select a portion of the branches to execute while skipping the remaining [71, 72, 161]. Specifically, HydraNet [161] discards the conventional convolutional block and uses multiple branches in the same place. While executing inference, only part of the multiple branches will be dynamically selected for execution according to the input sample. DRNet [72] conduct the data-dependent selection operation in a cell structure that is frequently used in differentiable neural architecture search [33]. The aforementioned hard-gated neural networks usually exploit a binary value to determine the skipping of a branch. Instead, another solution tends to assign a real-valued weight to each branch based on the input image. The branches are then ranked according to the corresponding weight, and only the top-K branches can be executed at test time [162, 163].

2.5.3 Dynamic Images

Traditional inference patterns usually preprocess (e.g., cropping and scaling) input images in a static manner [15, 164, 165]. In other words, all images will be cropped and scaled to the same size for inference. However, as discussed in [65, 74], different images are with distinct recognition difficulties, and statically processing all images

will lead to much redundant computation and memory consumption. To reduce such redundancy, many approaches are proposed to perform dynamic inference in the spatial dimension [65, 66, 74]. The dynamic inference in the spatial dimension can be mainly categorized as 1) dynamic resolution and 2) dynamic cropping.

Dynamic resolution neural networks [29, 65, 74, 166, 167] are proposed based on the fact that different images have distinct patterns and recognition complexities. Some 'easy' images may have a large and clear foreground, which can be predicted correctly even at a very small resolution. While for some 'complex' images with confusing patterns, a larger resolution is necessary to provide enough spatial information for an accurate prediction. MobileNet [29] coarsely shrinks or scales the size of all images to adapt to different hardware resource constraints. DR-ResNet [65] introduces a lightweight resolution predictor to efficiently estimate the optimal inference resolution for each input image, which achieves sample-wise dynamic inference in the resolution level. RANet [74] utilized dynamic resolution in multi-scale neural networks, where each sub-network corresponds to a resolution and different input images will first be resized to the smallest resolution and fed into the smallest sub-network, and then they will exit at different scales according to their complexity. In addition to classification tasks, Hao *et al.* [166] dynamically zoom different images for face detection to ensure the faces in different images are at the appropriate range for inference.

Dynamic image cropping implements dynamic inference at the region level [66, 168–171]. In classification tasks, Mnih *et al.* [169] formulate the inference of an image into a sequential decision problem, where the inference will only be performed on a patch of the original image for each iteration, and then the prediction output will be sent to RNNs to determine the next patch for inference. Li *et al.* [170] further improve the inference flexibility by introducing early stopping to control the number of patches. Furthermore, GFNet [66] proposes a general dynamic cropping framework, which first shrinks the image to a smaller resolution for inference, and then selectively crops different patches for inference via RNNs. GFNet [66] can be applied to multiple backbones for efficient inference. Other than RNNs, some works [171, 172] also utilize CAM [173] to generate the saliency map of input images, and then efficiently localize and crop the most discriminative patch of each image for inference. By doing so, the huge redundant computation on the background pixels can be removed and the inference efficiency can be improved significantly.

Chapter 3

Edge Deep Learning Hardware Benchmarking

As more and more deep learning algorithms are deployed to edge environments to mitigate privacy and latency issues of cloud computing. Various edge deep learning accelerators are devised to speed up deep learning algorithms on edge devices. Different edge deep learning accelerators feature different characteristics in terms of power and performance, which makes it a very challenging task to efficiently and uniformly compare different accelerators. In this chapter, we introduce EDLAB[1], an end-to-end benchmark, to comprehensively evaluate the performance of edge deep learning accelerators. EDLAB consists of state-of-the-art deep learning models, a unified workload preprocessing and deployment framework, as well as a collection of comprehensive metrics. In addition, we introduce parameterized models to evaluate the hardware performance bound so that EDLAB can identify the hardware potentials and the hardware utilization of different deep learning applications. Finally, we employ EDLAB to benchmark three edge deep learning accelerators and analyze the benchmarking results. From the analysis, we obtain some insightful observations that can guide the design of efficient deep learning applications.

[1]The work in this chapter has been published in [174]

3.1 Introduction

Deep neural networks (DNNs) have achieved considerable success in recent years, demonstrating outstanding performance in image classification [15, 49], machine translation [4], etc.

To pursue higher accuracy, DL models are growing more complex. Therefore, DL models are usually trained and deployed on powerful servers. However, executing DL models on cloud or remote servers suffers from two critical issues: 1) latency issue – some safety-critical systems, like self-driving cars and autonomous unmanned aerial vehicles, are subject to rigorous latency requirements, but inferring results from remote servers is difficult to guarantee latency requirements due to network bandwidth and availability; 2) privacy issue – some data are confidential and the data owner is unwilling to upload these data to remote cloud and servers. This drives the birth of edge computing [12], a computing service close to users acting like a middle-ware between cloud and end-users.

Edge systems are usually constrained by power, memory, cost, and physical limitations [12]. Therefore, power-hungry hardware, like high-end GPUs with more than 100W power consumption, may not be deployed in some cases with limited power supply. However, computation-intensive DL models require powerful computing units to provide sufficient computation for on-time/real-time operation. Thus, it results in a computational gap. To close the computational gap, edge deep learning accelerators (EDLAs) draw increasing attention from research and industry. Various low-power and efficient EDLAs are designed to speed up the inference in different edge environments.

Emerging EDLAs have different underlying hardware characteristics and employ various software tools to enable and adopt deep learning at the edge. The hardware and toolchain diversity of EDLAs makes it difficult to directly and effectively compare the performance and cost of different EDLAs, thereby hindering DL practitioners from effectively and efficiently deploying their newly designed DL models on suitable EDLAs. Moreover, the state-of-the-art benchmark methods, like [99, 101, 102], focus on conventional performance metrics' competition like highest accuracy, or lowest latency/highest throughput, which are inadequate for making a thorough evaluation for EDLAs and lack an evaluation from the hardware perspective.

Model	Scaling Factor	Top-1 Accuracy (%)	Latency (ms)
Wide-ResNet	1	88.09	5.50
	2	92.71	5.58
	4	94.19	5.58

TABLE 3.1: Accuracy and latency of Wide-ResNet model with different width scaling factors on NVIDIA Jetson Xavier.

EDLAs have a wide spectrum from high-performance computing units installed within communication stations to low-power, less-capable wearables. Simply knowing the best accuracy or latency for a DNN model on diverse EDLAs cannot provide insightful information for system designers, especially when developing an in-house edge intelligence system. It is unable to answer the practical question: *Can we further improve a developed model on an EDLA for better accuracy without compromising performance*?

Table 3.1 shows an experimental result of Wide-ResNet [41] with different width configurations on a representative edge device, NVIDIA Jetson Xavier, where the width scaling factor means the number of channels is scaled up by that value. We see that the model benefits from the increasing width with higher accuracy, while not influencing the latency. For EDLAs with low or mid computational capability, since they are likely to be resource-limited, it is essential to know the performance upper bound of EDLAs and DL models' efficiency, analogous to the Roofline model [175] for applications on multicore systems, such that designers can exploit the full potential of EDLAs to achieve the best trade-off between accuracy and performance.

However, evaluating the hardware performance bound of EDLAs exposes a great challenge. Different from CPUs which usually have several performance counters to provide system information to evaluate applications' efficiency, EDLAs are like a 'black-box' system, because they lack performance counters to monitor their operational status. Therefore, it needs a systematic method to approximately evaluate EDLAs' performance bound.

In this chapter, we propose a new benchmark method to evaluate EDLAs in a more comprehensive way, dubbed Edge Deep Learning Accelerator Benchmark (EDLAB). EDLAB has two unique features: 1) it provides a tool to integrate different software toolchains into a unified framework such that it can conveniently deploy DNN models onto various EDLAs; 2) we develop a parameterized model (PM) to evaluate the performance bound of black-box-like EDLAs. By means of

PM, EDLAB is capable of efficiently evaluating the execution efficiency of different DNN models on EDLAs. The proposed EDLAB is an easy-to-use tool for machine learning practitioners who wish to quickly prototype their newly proposed DNN algorithms on edge systems in the coming edge era. Although some EDLAs are developed for DL training, most EDLAs are devised for DNN inference. The current version of EDLAB tends to only benchmark the inference EDLAs. We may extend EDLAB for training as our future work.

The rest of this chapter is organized as follows: Section 3.2 discusses some existing EDLAs to demonstrate the challenges of benchmarking EDLAs. Section 3.3 presents the details of EDLAB. Section 3.4 demonstrates and analyzes the benchmarking results by using EDLAB on three off-the-shelf EDLAs and Section 3.5 summarizes this chapter.

3.2 Edge Deep Learning Accelerators

In this section, we use three widely-used EDLAs as examples to present the challenges of benchmarking EDLAs, such as the heterogeneity of different EDLAs. Then, we demonstrate how these difficulties are addressed in EDLAB.

- **Edge TPU:** Google develops Edge TPU to complement Cloud TPU to provide fast inference at the network edge, where TPU is built based on the core component of systolic array processors. To deliver high performance at a low resource cost, Edge TPU employs INT8 quantization to compress DNN models to reduce the memory footprint. For the deployment software, it relies on TensorFlow Lite to quantize DNN models and compile models into executable workloads.

- **Neural Compute Stick:** Intel Neural Compute Stick (NCS) is designed as a plug-in gadget to be integrated into some legacy systems which do not have AI boosting capability, where the core component of NCS is an array of VLIW vector processors. NCS uses OpenVINO, a vendor-provided DL library, to convert the models trained in other frameworks like TensorFlow to the format supported by NCS, where the model is quantized into FP16. It is worth noting that NCS is only applicable to AI vision tasks.

- **Jetson Xavier:** Jetson Xavier is a powerful edge GPU platform developed by NVIDIA. Since its GPU architecture is compatible with NVIDIA's Desktop GPU, the most of existing DL frameworks can be directly employed. However, in order to achieve a better performance, NVIDIA provides TensorRT framework to optimize the DNN inference by quantizing the model and fusing tensors.

From the above three EDLAs, it is not difficult to see that EDLAs differ from each other in terms of the underlying hardware architecture and the compiler used for deployment. The underlying architecture of EDLAs is decisive for the performance of EDLAs, which is fixed and cannot be modified during the deployment. Besides the architecture, the compiler or model optimizer provided by vendors is also of great significance for improving the actual performance of EDLAs. These compilers are configurable, and different configurations can lead to diverse performance improvements, which makes it challenging to fairly evaluate and compare different EDLAs. For example, TensorRT provides multiple quantization precision, including INT8, FP16, and FP32, which bring different latency-accuracy trade-offs. To solve this problem, on one hand, EDLAB unifies the deployment configuration to preserve the consistency of the workload when comparing different EDLAs. On the other hand, EDLAB integrates different optimization methods provided by the toolchain, which enables exploring the best performance of EDLAs under different deployment options.

3.3 Benchmarking Methodology

This section introduces the design philosophy of EDLAB, including the benchmarking models (Section 3.3.1), benchmark workload processing (Section 3.3.2), and selected metrics (Section 3.3.3). Fig. 3.1 presents the overview of EDLAB. First, deep learning applications and parameterized models are designed and trained in general deep learning frameworks. Then these trained models are sent to EDLAB, which determines the deployment hyperparameters (e.g. inference batch size, quantization precision, etc.) according to the target platform and the metric to evaluate and convert the models into various intermediate representations for different EDLAs. The converted deployable workload is deployed to the corresponding EDLA

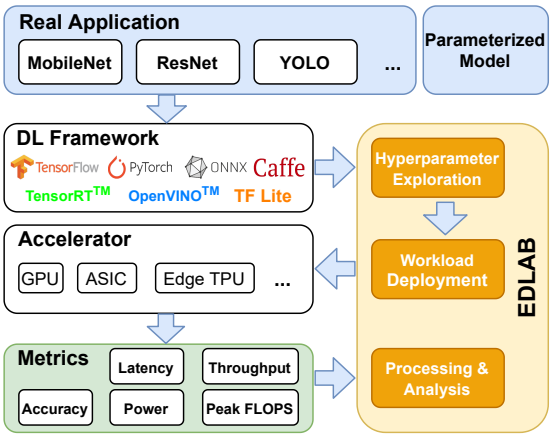


FIGURE 3.1: Overall architecture of EDLAB

TABLE 3.2: Some state-of-the-art models used for benchmarking. Classification models are trained on ImageNet [1] and the detection model is trained on COCO [2].

Model	MACs (G)	Params (M)	Top-1/mAP (%)	Description
InceptionV3	5.73	27.16	78.0	Classification
MobileNetV2	0.30	3.47	71.8	Classification
MnasNetB1	0.32	3.90	74.5	Classification
SSD-MobileNetV2	0.30	3.47	24.0	Detection

to perform inference and measure the proposed metrics. Finally, the obtained performance data of multiple metrics will be post-processed and analyzed by EDLAB in a fine-grained way.

3.3.1 Benchmarking Models

In EDLAB, we prepare two categories of DNN models for comprehensively benchmarking EDLAs with regard to different metrics, which will be discussed in detail in Section 3.3.3.

The state-of-the-art DNN models: Since in many deep learning applications, DNN models are developed based on these state-of-the-art (SOTA) models by using transfer learning, selecting some representative DNN models as the partial benchmark suite is a must. For the current EDLAB, we mainly target EDLAs with resource and power constraints, so we carefully selected several distinct DNN models as shown in TABLE 3.2, which can be well supported by the EDLAs we

have evaluated[2]. NLP models are not involved here because they are too large for some resource-constraint EDLAs. Moreover, some operators of NLP models are not well supported by some accelerators, such as Intel NCS, which is only applicable for vision tasks. In addition, DNN design is gradually shifting from manual design to automatic design, thus we take MnasNet [32] designed by NAS to follow the development trend. However, it is worth noting that there is no limitation in selecting the SOTA models for evaluation in EDLAB. Users are free to select the desired models. We have prepared API to seamlessly add new models into EDLAB.

Dicussion: The selection of benchmarking models is critical to fairly evaluate and compare different EDLAs. We must take into consideration the extensiveness, representativeness, and practicality of different models. In addition, the supportiveness of different EDLAs is also an important factor for model selection. Since most ELDAs are heterogeneous and do not support direct deployment via PyTorch or TensorFlow, different hardware vendors have developed their own model deployment tools for their own hardware. These model deployment tools are currently immature and only support a few models. For other unsupported models or unsupported operators, the model conversion process will go wrong. Sometimes this error manifests as a model conversion failure. Sometimes, although the model conversion can be completed, the accuracy of the converted model drops drastically to an unacceptable level. We believe that, as the development toolchains of EDLAs continue to evolve, the model conversion issues will finally be addressed and more advanced models will be supported. As a consequence, more models can be seamlessly integrated into our benchmark and the applicability and extensiveness of our benchmark can be well improved.

3.3.2 Workload Pre-processing Framework

The successful adoption of EDLAs relies on general DL development frameworks, such as TensorFlow, PyTorch, MXNet, as well as vendor-provided software toolsets.

[2]More models were evaluated, but we encountered some model conversion issues for some EDLAs. In order to have a fair comparison, we only keep these four for the current EDLAB and plan to investigate more in the next version.

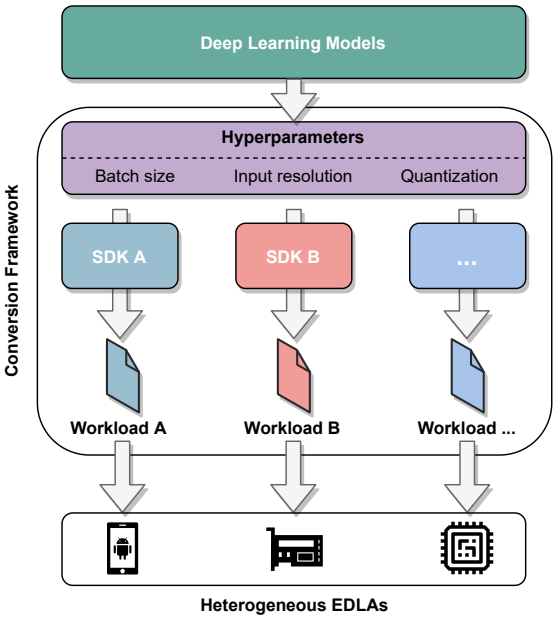


FIGURE 3.2: The unified model conversion framework, which converts a deep learning model into platform-specific deployable workload.

The general DL frameworks are employed to design and train models while vendor-provided tools optimize the trained models and convert the models into platform-specific intermediate representation.

Different platform-specific tools are not compatible with each other, which may lead to the inconsistency of converted workload for different platforms, and consequently affect the fairness of the benchmark results. In the proposed model conversion framework, we integrate different model conversion tools and uniformly control the model conversion parameters in the upper layer, which ensures that the workload generated for different platforms remains consistent. In the current version, the framework takes as input a trained model from TensorFlow, determines the conversion hyperparameters according to the target platform and the metric to evaluate, and calls the corresponding vendor-provide tool to perform the conversion. For now, there are three adjustable hyperparameters: 1) inference batch size; 2) input resolution; 3) quantization precision. The batch size depends on the metrics to evaluate, which will be discussed in detail in Section 3.3.3. The input resolution is determined by the benchmark dataset and the model, and the quantization precision is related to the hardware platform. With the information, EDLAB is capable of converting the model from TensorFlow into platform-specific deployable files.

Discussion: Currently, new edge accelerators are constantly emerging, and it is crucial to flexibly expand our benchmark to support more accelerators, thereby improving the reusability of our benchmark for different accelerators. However, as aforementioned, since different edge accelerators are usually driven by specific development software designed by the vendors themselves, we are unable to use the same code to evaluate all accelerators. To address this problem, we design two evaluating solutions for different EDLAs. First, for those EDLAs that can be directly driven by general deep learning frameworks, such as TensorFlow, PyTorch, and ONNX, etc., we develop codes to implement the whole benchmarking process in an end-to-end manner. Hence, users can directly use our benchmark tool to evaluate this kind of EDLAs without making any further development. While for those EDLAs that only support the vendor-specific development software, our benchmark will be responsible for implementing the shared functions, such as data preparation, data preprocessing, hyperparameter exploration, model preprocessing, etc. Then, our benchmark will export and freeze the trained models from general deep learning frameworks, and then we will generate a model conversion configuration file to guide the model conversion and optimization that need to be completed by the vendor-provided software. Subsequently, users need to complete the model conversion and deployment with the corresponding software tool automatically or manually. Finally, our benchmark will collect the benchmarking results and generate the final benchmarking report.

3.3.3 Benchmarking Metrics

The design goal of EDLAB is to provide a comprehensive evaluation for EDLAs which can cover various design concerns of edge systems. Therefore, in EDLAB, we evaluate and report several metrics. To accurately measure each metric, EDLAB separately customizes the deployment and benchmark settings for each metric. Take the inference batch size as an example, we will dynamically change the inference batch size when measuring different performance metrics. For instance, when we evaluate latency, we want to test the fastest speed at which the hardware can process a single image. To achieve this goal, the hardware cannot handle other tasks or other images in parallel, which will increase the latency, so we need to set the batch size to 1. While testing throughput, we want to test how many images the hardware can process at most within a certain period of time, so we need to

increase the batch size to make full use of hardware resources and thus obtain the highest throughput.

Accuracy (ACC): Usually, the accuracy can be reproduced once the model structure is determined and weights are well-trained and fixed. However, EDLAs employ different tools to convert pre-trained models for efficient execution and such conversion may change the model structures, data representation, and precision, hence it may cause an accuracy drop. It is important to report this accuracy drop for accuracy-sensitive applications. EDLAB calculates the accuracy based on the prediction results of all images in the test dataset. Each image will be predicted only once to mitigate the benchmarking cost. For simplicity, the batch size of inference is set to 1.

Latency (L): Latency indicates the time elapsed between one input data and its corresponding prediction. For some systems with rigorously temporal requirements, e.g., autonomous driving, latency is essentially critical. In EDLAB, we only measure the latency of the given model processing one image, thus the batch size has to be 1 for the latency evaluation. In order to avoid the variance caused by different system statuses, EDLAB runs a model on one image by 50 times and calculates the average value as the model's latency.

Throughput (T): The throughput of an EDLA is the largest number of samples processed within one second. The throughput is considered as an important metric for some applications, such as video analytics, which relies on the high throughput performance of hardware to guarantee high Frame-Per-Second (FPS) requirements. For both image classification and object detection, the throughput unit is images/second. To measure the highest throughput of EDLAs, We make the batch size as large as possible until the memory runs out. Similar to the latency measurement, each input batch will be repeated 50 times.

FLOPS (F): We also measure the number of Floating-point Operations executed Per Second (FLOPS) to evaluate the execution efficiency of EDLAs. FLOPS can help to evaluate the execution efficiency of different EDLAs under various DNN models. FLOPS is derived from the throughput and the number of operations as shown in Eq. 3.1:

$$F = T \times \text{FLOPs} \tag{3.1}$$

where T is the throughput of a DNN model on an EDLA and FLOPs denote the number of floating-point operations the DNN model has in total. Note that FLOPS indicates the hardware performance while FLOPs represent the DNN model's complexity.

Power (P): Some edge intelligent systems, especially those supplied by batteries, are subject to a limited power budget and prefer to achieve an expected performance under a limited power budget. However, power measurement relies on the power sensor to accurately obtain it. For those EDLAs which have internal onboard sensors, like the NVIDIA Jetson series, we can directly obtain power consumption from power sensors. For those without on-board sensors, power has to be measured by an external power meter. It is worth noting that the power consumption of an EDLA varies under different running configurations, and EDLAB only reports the highest power consumption, which can be obtained under the benchmark configuration of the throughput measurement.

Efficiency (E): The efficiency of EDLAs is a combined and more comprehensive metric, which is calculated as Equation 3.2:

$$E = \frac{T}{P} \tag{3.2}$$

where T is the throughput and P is the power consumption of the accelerator when achieving that throughput. The unit of efficiency is images per second per watt (imgs/s/W). This metric unifies performance and power into one combined metric, which is instrumental when comparing various EDLAs with different computational capabilities and power consumption levels.

3.4 Evaluation

In this section, we use EDLAB to benchmark three off-the-shelf EDLAs, analyze the benchmarking results and provide some insightful observations.

Experimental Settings: The test data of classification models and the object detection model are from the ImageNet2012 validation set and COCO2017 validation set, respectively. Before evaluating each metric, EDLAB first executes the model for 10 iterations for hardware warm-up.

TABLE 3.3: Some hardware platforms used in the evaluation experiments.

Platforms	Precision	Software
NCS2+Raspberry Pi4	FP16	OpenVINO
Edge TPU	INT8	TFlite
Jetson Xavier	INT8,FP16,FP32	TensorRT

TABLE 3.4: Experimental results of basic metrics. We do not measure memory utilization because the deployment toolchains of some EDLAs do not support monitoring memory usage. We will continue to update our benchmark to support evaluating more metrics.

Model	Accelerator	Latency (ms)	Accuracy (%)	Throughput (imgs/s)	Power (W)	Conversion (ms)
InceptionV3	NCS2 (FP16)	83.13	75.24	21.44	6.10	17990
	Edge TPU (INT8)	52.77	77.62	18.95	**4.69**	**17455**
	Xavier (INT8)	**3.55**	77.70	**641.73**	8.01	1579809
	Xavier (FP16)	4.90	77.79	428.84	15.12	589188
	Xavier (FP32)	14.51	77.82	111.06	21.84	42808
	Xavier (woRT)	27.21	**77.83**	101.70	20.67	0
MobileNetV2	NCS2 (FP16)	25.08	70.72	86.60	5.61	**6960**
	Edge TPU (INT8)	2.56	70.18	384.62	**5.08**	N.A*
	Xavier (INT8)	1.45	70.68	**1670.76**	6.49	325319
	Xavier (FP16)	**1.43**	71.16	1657.69	7.04	237770
	Xavier (FP32)	2.57	71.15	810.73	10.47	24038
	Xavier (woRT)	6.91	**71.17**	309.84	10.89	0
MnasNetB1	NCS2 (FP16)	33.56	73.13	63.35	5.49	9690
	Edge TPU (INT8)	3.50	70.94	285.71	**4.89**	**6051**
	Xavier (INT8)	**1.73**	71.26	**1194.59**	7.82	275228
	Xavier (FP16)	1.75	73.53	1193.49	8.56	222774
	Xavier (FP32)	3.17	73.52	614.26	12.02	24841
	Xavier (woRT)	8.14	**73.54**	265.40	15.13	0
SSD-MobileNetV2	NCS2 (FP16)	70.13	23.60	27.86	5.96	44830
	Edge TPU (INT8)	**15.45**	21.70	**64.72**	4.71	N.A.*
	Xavier (woRT)	47.26	**24.00**	34.02	**4.02**	0

*MobileNetV2 and SSD-MobileNetV2 for Edge TPU are directly taken from Google.

Evaluation Devices: The three EDLAs are listed in Table 3.3. Since quantization is a promising technique to reduce the model size and boost model inference, some EDLAs support different quantization precision configurations, from INT8 to FP32. For the EDLAs which support different quantization precision configurations, we evaluate all quantization configurations supported. NCS2 is a plug-in gadget that needs to be combined with one host machine. In our setting, we combine it with Raspberry Pi4 to set up an intact edge system. Raspberry Pi4 is responsible for processing data while NCS2 is responsible for conducting DNN inference.

All the benchmarking results are reported in Table 3.4, including accuracy, latency, throughput, power and conversion time which denotes the time to convert a general model to the format that EDLAs support. In addition, we visualize throughput, FLOPS, and efficiency in Fig. 3.3. Since Jetson Xavier can directly support general frameworks like TensorFlow, etc, we show the benchmarking results of Xavier

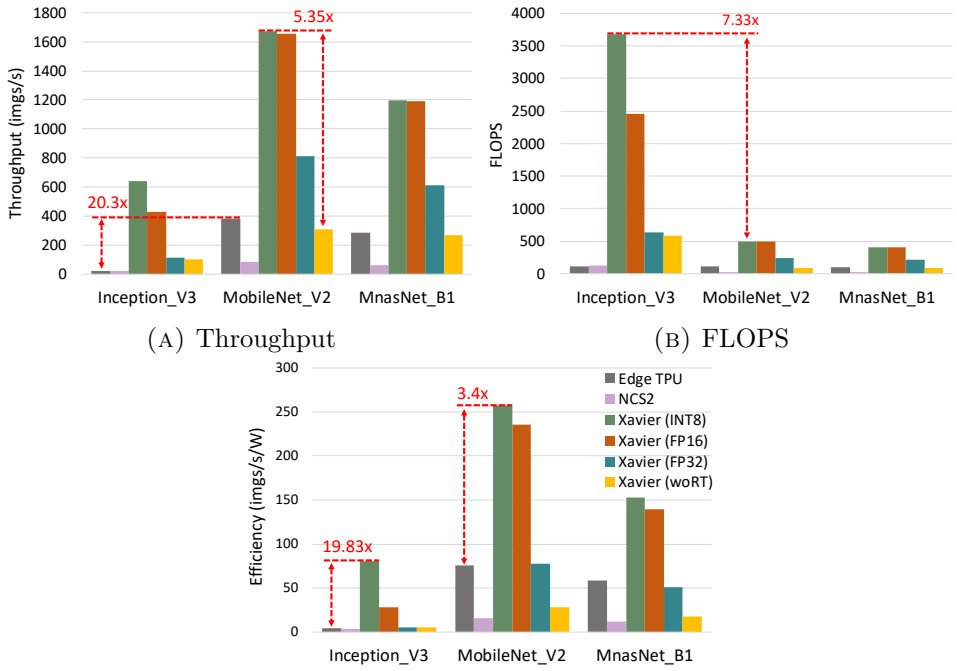
without using the optimization tool, TensorRT (denoted as woRT).

Accuracy: For all data precision, Xavier performs better in prediction accuracy than other accelerators that use the same data precision. For instance, Xavier (INT8) outperforms Edge TPU by 0.08% - 0.5% accuracy advantage. Also, for Xavier (FP16) and NCS2, there is an accuracy gap ranging from 0.4% to 2.55%. This reveals that different quantization strategies and internal data representation affect the prediction quality. For accuracy-sensitive applications, it should be taken into account.

Latency: From the results of Xavier, the quantized models can achieve up to 4× latency reduction, but interesting to see that lower precision quantization does not necessarily lead to lower latency. For example, Xavier (INT8) has higher latency than Xavier (FP16) for MobileNetV2. The latency reduction of Edge TPU from InceptionV3 to MobileNetV2 and MnasNetB1 is much more significant than NCS2. This may be because these two models are developed by Google and they develop these models with consideration of underlying hardware architecture. This somehow indicates that the hardware-software co-design method can better exploit the limited resources of EDLAs.

Throughput: Fig. 3.3a shows the throughput results of EDLAs. All EDLAs achieve higher throughput for MobileNetV2 and MnasNetB1 than for InceptionV3, since MobileNetV2 and MnasNetB1 are designed for mobile devices with fewer operations than InceptionV3, which seeks higher accuracy without consideration of hardware limitations. Edge TPU attains 20.3× throughput improvement for MobileNetV2 compared to InceptionV3, where the corresponding throughput improvement of NCS2 and Xavier (FP32) are 4.04× and 7.30×, respectively. This again confirms the importance of hardware-software co-design. In addition, model optimizations adopted by EDLAs also bring throughput improvement. For instance, Xavier (INT8) has 5.35× higher throughput than Xavier (woRT). Even using the same data precision, Xavier (FP32) achieves higher throughput than Xavier (woRT).

Power: The high computing capability of Xavier comes at the cost of higher power consumption. An interesting finding about power consumption is that quantization contributes to reducing power consumption. Xavier (INT8) achieves up to 2.73× power reduction for InceptionV3 compared to Xavier (FP32). In addition, the large model consumes more power than the two mobile setting models, and we think it is



(A) Throughput (B) FLOPS

(C) Efficiency

FIGURE 3.3: Evaluation results of derived metrics

because large models need more resources to exploit the model's parallelism. With the power and inference latency, we can easily derive the total energy consumption, which is critical for some battery-based edge devices.

FLOPS: Fig. 3.3b is the results of FLOPS of EDLAs for the three DNN models. In contrast to the throughput results, all EDLAs achieve the highest FLOPS performance for InceptionV3, even though MobileNetV2 and MnasNetB1 have higher throughput. The gap of FLOPS for different models on the same platform is up to 7.35× because large DNN models have more parallel operations, which can employ more computing cores to work concurrently, and thus process more operations at the same time.

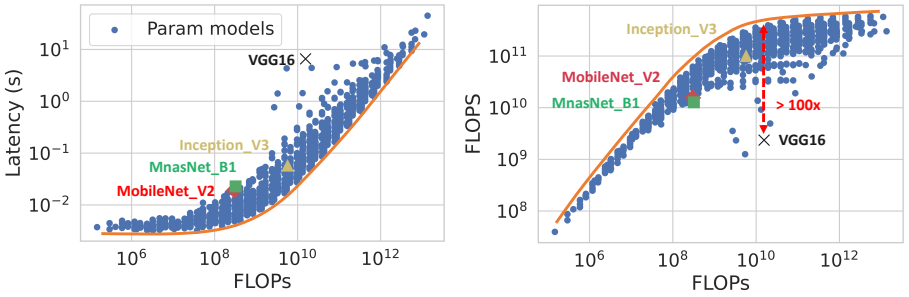
Efficiency: From Fig. 3.3c, since Edge TPU consumes less power, the efficiency difference between Xavier (INT8) and Edge TPU is reduced. For example in MnasNetB1, if we only look at the throughput between Xavier(FP32) and Edge TPU, Xavier(FP32) has better throughput. However, in terms of efficiency metric, Edge TPU is better and this indicates that Edge TPU utilizes power more efficiently.

Model Conversion Cost: EDLAB strives to provide an end-to-end test for all tested EDLAs. Table 3.4 shows the conversion cost for the three EDLAs. The majority

of this time cost is from the conversion tools provided by vendors. We can see that the conversion cost is not negligible. Since automated DNN design which explores many hyper-parameter configurations to find the best design is becoming mainstream, testing different designs on the target EDLAs would be a significantly time-consuming procedure. This suggests modeling the hardware at the system level, such that the design cost can be reduced. We envision that parameterized models may be used for this purpose.

Parameterized Model Benchmarking: In this section, we show the benchmarking results of using parameterized models of EDLAB. Since the evaluation with the parameterized model is quite time-consuming, we currently only conduct the evaluation under two performance modes of Jetson Xavier (15W and 30W), which is equivalent to two different devices. according to the results, we find the distributions of the two experiments are similar. Therefore, we only show the experimental results under the 15W mode for analysis. The experimental results are shown in Fig. 3.4, where the blue dots indicate the parameterized models. It is clear to see that by means of parameterized models EDLAB can identify the hardware upper bound of Xavier under different complexity in terms of latency (Fig 3.4a) and FLOPS (Fig 3.4b). From Fig 3.4a, We observe that inference latency does not increase with model FLOPs until model FLOPs reach 10^8. This is because the hardware is not fully utilized at this time, so increasing model size and computation can significantly improve hardware utilization and thus the inference latency does not increase significantly. After model FLOPs reach 10^8, we reach the full utilization of hardware, which is reflected in the fact that the FLOPS of the hardware no longer increases with model FLOPs (as shown in Fig 3.4b). Therefore, further increasing model FLOPs will cause a significant increase in inference latency. We place four existing models in the performance and latency spectrum. Except for the three used in our previous evaluation, we add VGG16 to highlight the efficiency issue.

We can see that no model achieves the best performance of the tested EDLA. Especially, the old VGG16 has the largest number of FLOPs but the lowest efficiency (100× difference between the performance bound and the model performance), thereby leading to the longest latency. The execution efficiency has been improved for the recently developed models, but still, they have not achieved the best performance on the hardware. As we know for DNN models, the deeper and wider, the



(A) Latency (B) Performance

FIGURE 3.4: Evaluation results of the parameterized model.

better. This implies that we should bear in mind the underlying hardware when designing edge intelligence systems.

3.5 Summary

To address the challenges of benchmarking EDLAs, we present a unified benchmark methodology and a benchmark tool EDLAB. EDLAB integrates multiple SDKs and unifies the workload processing and deployment for different EDLAs. One novelty of EDLAB is to integrate parameterized models which enable us to model the performance bound of EDLAs and quickly evaluate the performance of diverse DNN models on different EDLAs. We use EDLAB to benchmark three off-the-shelf EDLAs and the experimental results show the applicability of EDLAB and provide some interesting observations which may help the design of efficient DNN models. In the future, we will keep expanding EDLAB to integrate more types of deep learning tasks and support more emerging EDLAs. In addition, EDLAB can be combined with neural architecture search, model compression, and model scaling to design effective and efficient edge intelligence systems.

Chapter 4

Efficient Model Scaling for Resource Utilization

In this chapter [1], we present our efforts in hardware-efficient model scaling, where we tune model architectures and input images to improve hardware utilization to achieve higher accuracy without sacrificing execution efficiency on edge devices. Our two research works: 1) hardware-aware compound model scaling, and 2) model-aware adaptive model scaling, are presented in detail in Section 4.1 and Section 4.2, respectively.

4.1 Hardware-Aware Compound Model Scaling

Model scaling is an effective way to improve the accuracy of DNNs by increasing the model capacity. However, existing approaches seldom consider the underlying hardware, causing inefficient utilization of hardware resources and consequently high inference latency. In this section, we propose HACScale, a hardware-aware model scaling strategy to fully exploit hardware resources for higher accuracy. In HACScale, different dimensions of DNNs are jointly scaled with consideration of their contributions to hardware utilization and accuracy. To improve the efficiency of width scaling, we introduce importance-aware width scaling in HACScale, which

[1] The work in this chapter has been published in [176]

computes the importance of each layer to the accuracy and scales each layer accordingly to optimize the trade-off between accuracy and model parameters. Experiments show that HACScale improves the hardware utilization by 1.92× on ImageNet, as a result, it achieves 2.41% accuracy improvement with a negligible latency increase of 0.6%. On CIFAR-10, HACScale improves the accuracy by 2.23% with only 6.5% latency growth.

4.1.1 Introduction

As we discussed in Section 1.1.2, model scaling is one of the main techniques to improve the model capacity for higher prediction accuracy, which achieves higher accuracy at the cost of more computation (i.e., FLOPs) and parameters. Model scaling can be mainly conducted on the depth, width, and resolution dimensions of CNNs, and scaling different dimensions has distinct impacts on model costs (i.e., FLOPs, parameters) and accuracy. Many model scaling approaches have been proposed to strive to achieve the best trade-off between model costs and accuracy. Some of them only scale a single dimension [29, 32, 44, 45], while others strive to jointly scale multiple dimensions for further accuracy improvement [40, 46, 177]. However, these approaches suffer from two problems: 1) single-dimensional scaling only brings limited accuracy improvements and the accuracy quickly saturates. 2) They do not consider the impact of scaling different dimensions on hardware utilization. As a result, the scaled models still cannot utilize hardware resources efficiently, resulting in a significant increase in latency.

To address the above issues, we propose a hardware-aware compound scaling framework, dubbed HACScale, to fully exploit hardware resources to improve the accuracy without increasing the latency. We first investigate how scaling different dimensions affects the hardware utilization and the latency, and find that width scaling and resolution scaling can considerably improve the hardware utilization and maintain the original latency. Therefore, we design a scaling strategy to mainly scale the width and resolution of DNNs for better accuracy. In addition, as part of HACScale, we introduce importance-aware width scaling to separately scale the width of each layer according to their importance to the accuracy. By this means, the accuracy of a DNN can be further improved within the same memory constraint compared with uniformly scaling all layers [29, 41]. As shown in Figure

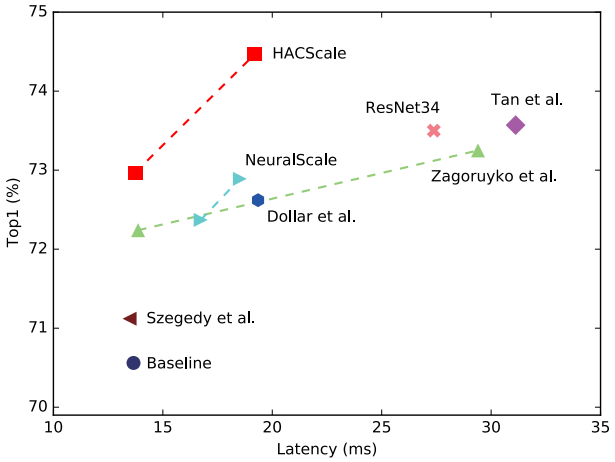


FIGURE 4.1: Experimental results on the trade-off between accuracy and latency of different scaling methods on ImageNet. The baseline network is ResNet18 and the latency is measured on NVIDIA Jetson Xavier.

4.1, for systems that have a tight latency constraint, HACScale can significantly improve the accuracy without affecting the latency. Besides, for a relaxed latency constraint, HACScale still achieves a better trade-off between accuracy and latency compared with other methods. Our main contributions are summarized as follows:

- We present HACScale, a hardware-aware compound scaling framework, to improve the hardware utilization of existing DNNs. By collaboratively scaling different dimensions based on their contributions to hardware utilization and prediction accuracy, HACScale empowers different DNNs to fully exploit the underlying hardware to improve their accuracy.
- We propose a novel importance-aware width scaling method. During the scaling process, we quickly estimate the importance of each layer through an importance predictor and scale each layer individually, which enables us to control the width of each layer in a fine-grained manner and improves the efficiency of width scaling.
- Experiments show that HACScale achieves 2.41% and 2.23% accuracy gains on ImageNet and CIFAR-10 with increasing the latency by only 0.6% and 6.5%, respectively.

4.1.2 Problem Formulation

In this section, we give the formal formulation of our problem. Given a baseline convolutional neural network $\mathcal{N}$ with n layers, each convolutional layer can be defined as a function: $Y_i = \mathcal{F}_i(X_{<H_i,W_i,C_i>})$, where $X_{<H_i,W_i,C_i>}$ is the input tensor with a spatial dimension (H_i, W_i) and a channel dimension C_i. $\mathcal{F}_i$ and Y_i are the convolutional operator and the output tensor, respectively. A convolutional neural network is formed by cascading n convolutional layers. Consequently, the baseline neural network $\mathcal{N}$ can be formulated as:

$$\mathcal{N} = \bigodot_{i=1...n} \mathcal{F}_i(X_{<H_i,W_i,C_i>}) \tag{4.1}$$

Given a latency constraint $\mathcal{L}$, and a memory constraint $\mathcal{M}$, we strive to find a collection of scaling coefficients (d, r, Θ) to scale all dimensions of the neural network $\mathcal{N}$, where d is the depth scaling coefficient, r is the resolution scaling coefficient, and $\Theta = \{w_1, w_2, ..., w_n\}$ is the collection of each layer's width scaling coefficient. By this means, the scaled neural network can maximize its accuracy while still satisfying the latency constraint $\mathcal{L}$ and the memory constraint $\mathcal{M}$. The optimization objective can be formulated as:

$$\begin{aligned} \max_{d,r,\Theta} \quad & Accuracy(\mathcal{N}(d, r, \Theta)) \\ s.t. \quad & \mathcal{N} = \bigodot_{i=1...n \cdot d} \mathcal{F}_i(X_{<r \cdot H_i, r \cdot W_i, w_i \cdot C_i>}) \\ & Latency(\mathcal{N}(d, r, \Theta)) \leq \mathcal{L} \\ & Memory(\mathcal{N}(d, r, \Theta)) \leq \mathcal{M} \end{aligned} \tag{4.2}$$

However, exhaustively searching for the optimal combination of (d, r, Θ) suffers from a prohibitively expensive search cost, in the next section, we will present how HACScale can efficiently achieve the optimization goal.

4.1.3 Hardware-Aware Compound Scaling Algorithm

Compound scaling, instead of scaling an individual dimension, jointly scales multiple dimensions of a network to improve accuracy. Compared with single-dimensional scaling, compound scaling is capable of achieving higher accuracy.

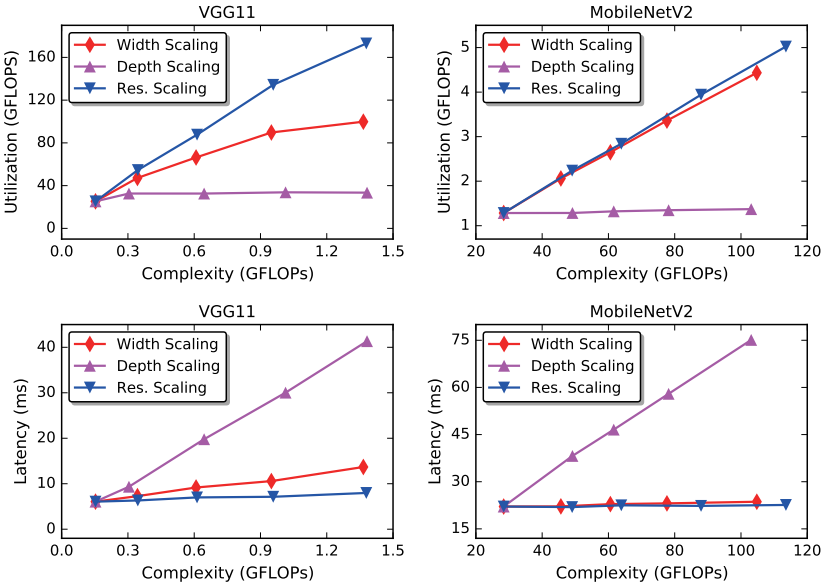


FIGURE 4.2: (Upper): The improvement of hardware utilization by different scaling methods. The horizontal axis denotes the computational complexity of scaled models and the vertical axis is the hardware utilization while running those scaled models. (Lower): The impact of different scaling methods on the inference latency. The vertical axis represents the inference latency (ms) measured on NVIDIA Jetson Xavier. Those figures show that depth scaling can hardly improve the hardware utilization and brings a steeper increase in the latency than width scaling and resolution scaling.

To scale a network for higher accuracy without affecting the inference latency noticeably, we need to better exploit the hardware parallelism. Therefore, we selectively scale those dimensions which can improve the hardware utilization. To identify the impact of scaling different dimensions on the hardware utilization, we conduct a series of experiments, where different baseline models are scaled up to different computational complexity using three scaling methods: 1) width scaling, 2) depth scaling, and 3) resolution scaling. As shown in Figure 4.2, for resolution scaling and width scaling, we find that the hardware utilization (GLOPS[2]), which is represented by the number of operations processed by the hardware per second, rises with the model complexity (GFLOPs[3]), while depth scaling does not change the hardware utilization obviously regardless of the model complexity. Consequently, scaling the depth of networks results in a sharp increase in latency.

For many latency-critical systems, they expect to improve the accuracy without changing the latency. Then, depth scaling is inapplicable in this case since it

[2]GFLOPS: Giga Floating-Point Operations Per Second
[3]GFLOPs: Giga Floating-Point Operations

will notably increase the latency. Considering our target (i.e., increasing accuracy without latency increase), we focus on the width and resolution scaling instead of jointly scaling all three dimensions. Our compound scaling strategy is formulated as:

$$d = 1, \quad r = \sqrt{s}^{\alpha}, \quad w = \sqrt{s}^{1-\alpha} \tag{4.3}$$

where s is a predefined scaling factor of the model's computational complexity, and $\alpha \in (0, 1)$ is a hyperparameter used to allocate the resources between resolution scaling and width scaling. To find the optimal value of α, we formulate the optimization problem as:

$$\begin{aligned} \max_{\alpha} \quad & Accuracy(\mathcal{N}(1, \sqrt{s}^{\alpha}, \sqrt{s}^{1-\alpha})) \\ s.t. \quad & \mathcal{N} = \bigodot_{i=1...n \cdot 1} \mathcal{F}_i(X_{<\sqrt{s}^{\alpha} \cdot H_i, \sqrt{s}^{\alpha} \cdot W_i, \sqrt{s}^{1-\alpha} \cdot C_i>}) \\ & \alpha \in (0, 1) \end{aligned} \tag{4.4}$$

Different from the previous work directly searching for all three scaling factors [40], we only need to optimize a single hyperparameter α, which narrows the search space and greatly cut down the search cost. After obtaining the optimal value of α, the scaling coefficients for all dimensions can be computed with Equation 4.3. The predefined depth scaling coefficient and the searched resolution scaling coefficient will be applied directly, while the width scaling coefficient for each layer will be further optimized by the proposed importance-aware width scaling algorithm.

4.1.4 Fine-Grained Width Scaling Algorithm

Width scaling has been widely used to improve the representational power of DNNs by adding additional weight kernels to each layer. In practice, previous methods scale the width of networks in a uniform manner, where all layers will be scaled by a uniform scaling coefficient [29, 40, 41]. However, different layers in a network contribute differently to the accuracy [52, 53, 128], thus scaling all layers uniformly

is inefficient in terms of improving the accuracy. To address this problem, we propose to scale each layer individually according to their importance to the accuracy instead of using the searched uniform width scaling coefficient.

Inspired by [52], we define the importance of a layer as the loss increment of the network when removing a kernel from this layer. The loss increment can be approximated by adopting the first-order Taylor expansion on the removed weight kernel:

$$\begin{aligned} \mathcal{I}_l &= (\mathbb{E}(K) - \mathbb{E}(K|k_l = 0))^2 \\ &\approx \sum_{m \in k_l} (\frac{\partial \mathcal{L}}{\partial m} \cdot m)^2 = \sum_{m \in k_l} (g_m \cdot m)^2 \end{aligned} \tag{4.5}$$

where $\mathbb{E}$ denotes the loss function, K represents the weight kernels of the network, and k_l is a weight kernel in the l-th layer. m is a single parameter in k_l and $g_m = \frac{\partial \mathcal{L}}{\partial m}$ is the gradient of m, which can be computed through backpropagation on the network.

As a weight kernel is added to or removed from a layer, the importance of the layer will change and the previously computed importance score will no longer apply. However, it is computationally prohibitive to update the importance of each layer by training the network from scratch whenever a new weight kernel is added to the network. Therefore, we build an importance predictor for each layer to predict its importance. We first sample several networks with different layer width configurations, and then we train these networks to obtain the importance of each layer under different width configurations. Specifically, given a baseline model with n layers, we denote the width configuration of the baseline network as $\{C_1, C_2, ..., C_n\}$, where C_l means the number of channels in the l-th layer. Subsequently, we will sample around 5 scaling coefficients from 1 to 2 and use these scaling coefficients to generate the sampled models. Let k_i be the i-th scaling coefficient, the corresponding width configuration of the scaled model will be $\{k_i \cdot C_1, k_i \cdot C_2, ..., k_i \cdot C_n\}$. These sampled models are then initialized and trained to get the importance data. To reduce the training overhead, we only train each model for 10 epochs. Meanwhile, it is worth noting that the importance predictor is model-aware, which means that we will build an importance predictor for each given baseline network. Finally, the obtained importance data will be utilized to train the importance predictor.

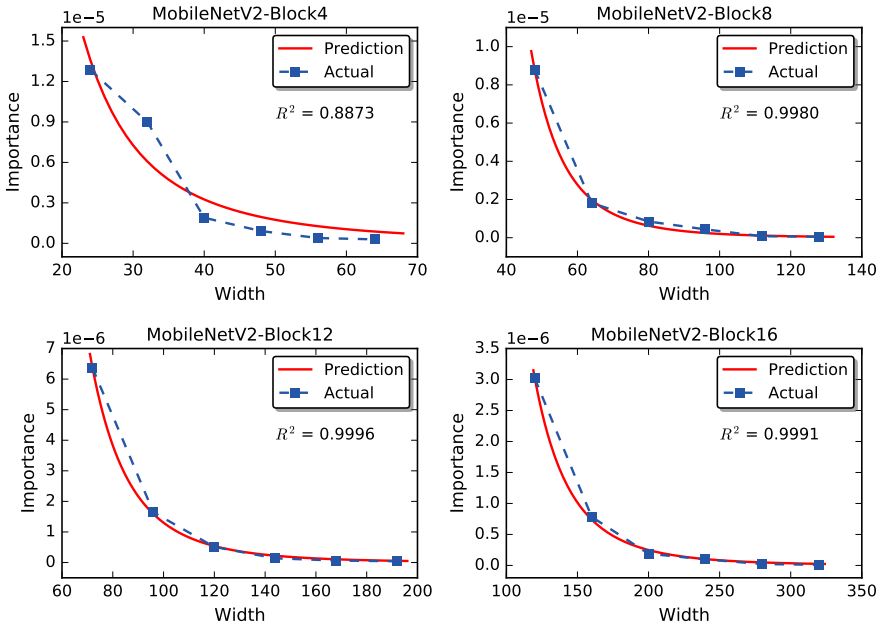


FIGURE 4.3: The actual and predicted importance of different layers on CIFAR-10. $R^2 \in [0, 1]$ denotes the coefficient of the determination between the actual and predicted importance. A higher value of R^2 means a better prediction model.

As demonstrated in Figure 4.3, the importance of a layer is highly related to the layer width. We model the relationship between the importance and the layer width as a power function:

$$\mathcal{I}_l = a_l \cdot C_l^{b_l} \tag{4.6}$$

where C_l is the number of channels of the l-th layer. a_l and b_l are the parameters of the l-th layer's prediction model, which are fitted by non-linear least squares. With these importance prediction models, we are able to quickly estimate the importance of each layer without conducting the time-consuming training process, thereby accelerating the scaling process.

The importance-aware width scaling algorithm is summarized in Algorithm 1. First, we initialize a baseline model and generate the importance score of each layer with its importance prediction model. Subsequently, we iteratively add weight kernels to the layer with the highest importance score. In each iteration, we will do four things: 1) Sorting all layers according to their importance and identify the most important layer (denoted as i). 2) Updating the width scaling coefficient of layer i with a scaling stride $s = 10$ (i.e., the number of weight kernels to add). 3) Updating the width of layer i by adding s weight kernels to this layer. 4) Updating the importance score of layer i using its prediction model according to the

Algorithm 1: Importance-Aware Width Scaling

Require: The importance prediction model of each layer $\{\mathcal{I}_l = a_l \cdot C_l^{b_l}\}_{l=1}^n$, the resolution scaling coefficient r, the baseline width configuration $\{C_l\}_{l=1}^n$, the latency constraint $\mathcal{L}$, and the memory constraint $\mathcal{M}$.
Ensure : The width scaling coefficient of each layer $\Theta = \{w_l\}_{l=1}^n$
Initialize the network $\mathcal{N}$ with r and $\{C_l\}_{l=1}^n$, the scaling stride $s = 10$, and the width scaling coefficients $\{w_l = 1\}_{l=1}^n$;
for *Each layer l in the network $\mathcal{N}$* **do**
  Compute importance $\mathcal{I}_l = a_l \cdot C_l^{b_l}$;
end
while $Latency(\mathcal{N}) \leq \mathcal{L}$ *and* $Memory(\mathcal{N}) \leq \mathcal{M}$ **do**
  Find the most important layer $i = \underset{l}{\text{argmax}} \ \mathcal{I}_l$;
  Update the scaling coefficient $w_i = \frac{C_i + s}{C_i} \cdot w_i$;
  Update the layer width $C_i = C_i + s$;
  Update the importance $\mathcal{I}_i = a_i \cdot C_i^{b_i}$;
end
return $\Theta = \{w_l\}_{l=1}^n$

updated width. At the end of each iteration, we will check the latency and memory consumption of the scaled model to make sure the latency constraint and memory constraint are not violated. To eliminate the expensive deployment cost, the latency is estimated by a simple but effective latency predictor, and the memory consumption is quantified by the network parameters. After the whole scaling process, we will obtain the scaling coefficient of each layer and we can directly scale the model as Equation 4.2.

4.1.5 Experiments

The proposed HACScale is implemented with PyTorch and evaluated on CIFAR-10 and ILSVRC-2012 (i.e., ImageNet). CIFAR-10 contains 50,000 training images and 10,000 test images with a size of 32×32. ImageNet contains 1.28 million training images and 50,000 validation images with a size of 224×224. We scale networks with HACScale and other state-of-the-art approaches under two scenarios: 1) We set a tight latency constraint to compare the potential of different methods for improving the network accuracy under current latency. 2) We loosen the latency constraint to explore the efficacy of different methods in terms of the trade-off between accuracy and latency.

TABLE 4.1: Experimental results of different model scaling approaches on ImageNet and CIFAR-10. The latency on ImageNet is measured on NVIDIA Jetson Xavier, and the latency on CIFAR-10 is measured on NVIDIA Jetson TX2.

Method	Params (M)	FLOPs (M)	Latency (ms)	Power (W)	Utilization (GFLOPS)	Power Eff. (GFLOPs/J)	Top-1 (%)
ResNet18 on ImageNet							
Baseline [15]	11.68	1735.22	13.66	14.87	127.03	8.54	70.56
Zagoruyko et al. [41]	16.33	2365.95	13.87	22.34	170.58	7.64	72.24
Szegedy et al. [164]	11.68	3253.31	13.48	25.60	241.34	9.43	71.12
NeuralScale [45]	12.74	2988.33	16.73	20.68	178.62	8.64	72.37
HACScale (our)	12.96	3359.73	13.74	24.73	**244.52**	**9.89**	**72.97**
Zagoruyko et al. [41]	19.54	2933.99	29.40	29.87	99.80	3.34	73.25
ResNet34 [15]	21.79	3587.41	27.38	18.54	131.02	7.07	73.50
Tan et al. [40]	18.74	2836.77	31.12	23.90	91.16	3.81	73.57
Dollar et al. [46]	16.21	2788.23	19.35	21.27	144.09	6.77	72.62
NeuralScale [45]	17.96	3983.63	18.52	29.30	215.10	7.34	72.89
HACScale (our)	17.14	4812.85	19.21	33.21	**250.54**	**7.54**	**74.47**
VGG11 on CIFAR-10							
Baseline [42]	9.20	150.00	11.92	7.62	12.58	1.65	92.06
Zagoruyko et al. [41]	20.76	343.08	12.88	12.23	26.63	2.18	92.72
VGG16 [42]	14.73	314.03	18.45	9.14	17.02	1.86	93.83
Tan et al. [40]	16.28	493.48	15.37	11.90	32.11	2.69	92.88
Dollar et al. [46]	17.07	328.83	13.25	11.74	24.82	2.11	92.69
NeuralScale [45]	14.44	604.81	12.56	11.94	48.15	4.03	93.26
HACScale (our)	12.24	1442.37	12.70	12.92	**113.57**	**8.79**	**94.29**
Zagoruyko et al. [41]	36.89	608.44	28.63	12.98	21.25	1.63	92.98
VGG19 [42]	20.40	399.00	23.43	10.61	17.03	1.64	93.71
Tan et al. [40]	21.20	893.15	24.98	12.38	35.75	2.89	94.27
Dollar et al. [46]	29.74	575.07	24.58	13.64	23.40	1.70	92.78
NeuralScale [45]	36.90	1418.14	29.19	13.98	48.58	3.48	93.31
HACScale (our)	21.10	2547.08	25.50	14.15	**99.89**	**7.06**	**94.38**

Training Settings: All networks are trained using stochastic gradient descent (SGD) with a momentum of 0.9. On CIFAR-10, networks are trained for 300 epochs with a batch size of 128. The initial learning rate $\lambda = 0.1$ and divided by 10 at epoch 100, 200, and 250. On ImageNet, we train all networks for 100 epochs with a batch size of 1024 and 5 epochs of gradual warmup. We use exponential learning rate scheduling with the initial learning rate $\lambda = 2.0$ and the decay factor $\beta = 0.02$.[4] Besides, we also use label smoothing with $\epsilon = 0.1$, mixup with $\alpha = 0.2$, AutoAugment, stochastic weight averaging, and mixed precision training. For a fair comparison, the results of all methods are obtained under the same training settings.

Deployment: We deploy models for CIFAR-10 onto NVIDIA Jetson TX2, and models for ImageNet onto NVIDIA Jetson Xavier to evaluate the inference latency and the power consumption. The final latency is computed by averaging the latency

[4] $\lambda_t = \lambda\beta^{\frac{t}{T}}$, where λ_t is the learning rate of the current epoch, t is the current epoch, and T is the number of total epochs.

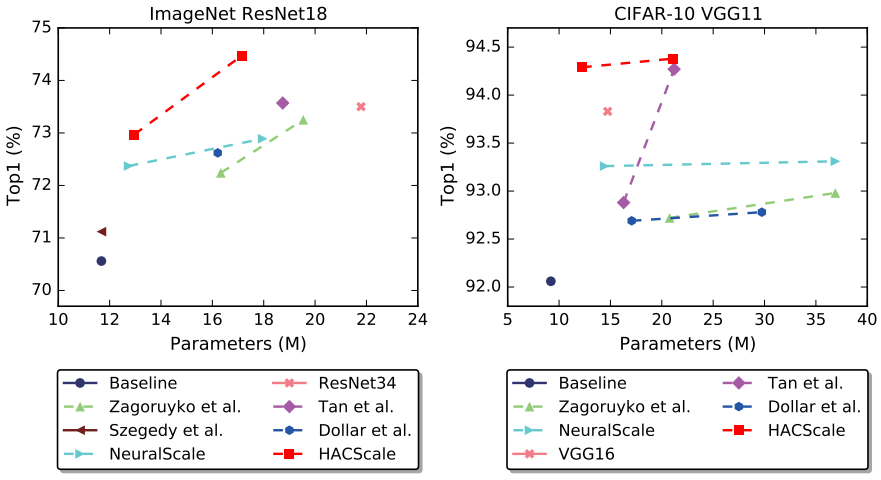


FIGURE 4.4: The parameter efficiency of different scaling methods. The baseline models for ImageNet and CIFAR-10 are ResNet18 and VGG11, respectively.

of 50 inferences with a batch size of 1. The final power consumption is also the average power consumption measured during inference.

Results on ImageNet: We use ResNet18 [15] as the baseline model for different scaling methods on ImageNet. As demonstrated in Table 4.1, under both the tight latency constraint and the loose constraint, HACScale achieves the highest accuracy among all approaches. Compared with the baseline model, HACScale improves the accuracy by 2.41% almost without increasing the latency, while obtaining 3.91% accuracy improvement by consuming an additional 40% latency budget. By contrast, Tan et al. [40] attains an accuracy gain of 3.01% with an increase of 128% in latency. Meanwhile, as shown in Figure 4.4, benefiting from the importance-aware width scaling strategy, HACScale achieves the best parameter efficiency, which improves the accuracy by 2.41% with an increase of only 11% in parameters, while NeuralScale increases the parameters by 53.77% for an accuracy improvement of 2.33%. As a consequence, the scaled model by HACScale can be deployed to more memory-constrained devices. In addition, HACScale improves the hardware utilization by a large margin. It achieves 1.92× and 1.97× improvements in the hardware utilization under the tight and loose latency constraint, respectively. Consequently, even though the scaled models by HACScale contain more FLOPs, they can still achieve lower latency due to more efficient utilization of hardware. The power efficiency is represented as the number of GFLOPs completed per Joule. As shown in Figure 4.5, HACScale also achieves better power efficiency compared with other methods.

Results on CIFAR-10: As demonstrated in Table 4.1, under the tight latency constraint, HACScale improves the hardware utilization by 9.03×, which surpasses other approaches by a large margin. Consequently, HACScale obtains an accuracy improvement of 2.23% with a latency increase of only 6.5%. As a comparison, NeuralScale [45] only improves the accuracy by 1.2% with similar latency. Besides, compared with the baseline model, HACScale achieves 5.33× higher power efficiency, which is also the highest among all approaches for comparison. Due to the importance-aware width scaling, HACScale only improves the model parameters by 33.1% compared with the baseline model, while NeuralScale [45] has 56.96% more parameters than the baseline model.

On the other hand, under the loose latency constraint, we improve the hardware utilization by 7.94×. Consequently, HACScale gains a 2.32% accuracy improvement with a latency of 25.5ms on NVIDIA Jetson TX2, while NeuralScale [45] takes 29.19ms but only improves the accuracy by 1.25%. Similarly, Zagoruyko et al. [41] spends 28.63ms on achieving a 0.92% accuracy improvement. In addition, the scaled model by HACScale only contains 21.1M parameters, which is similar to Tan et al. [40] and VGG19 but better than the rest. Meanwhile, we note that on both CIFAR-10 and ImageNet, the power consumption of the scaled models increases significantly. The increase in power consumption actually originates from the increased parallel FLOPs. More parallel FLOPs mean that we need to utilize more computing units in parallel and generate more memory access operations, so the power consumption will increase accordingly. Even though our method incurs higher power consumption, our inference latency is lower than other methods, thus our overall inference energy is better than theirs.

4.1.6 Summary

In this section, we present a compound scaling framework to efficiently utilize the hardware to improve the accuracy of DNNs without affecting the latency. After analyzing the hardware utilization of different scaling strategies, we find that width scaling and resolution scaling can significantly improve hardware utilization. By collaboratively scaling the width and the resolution of a DNN based on their contributions to the accuracy, HACScale yields networks that achieve notably higher accuracy without compromising the latency. By separately scaling each layer of

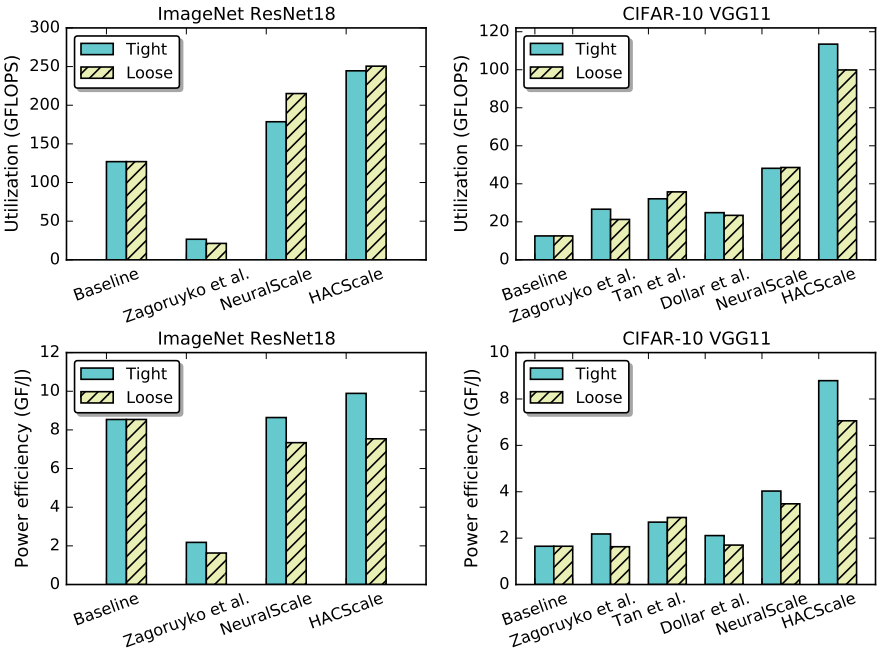


FIGURE 4.5: Comparison of hardware utilization and power efficiency among different approaches under the tight latency constraint and the loose latency constraint. The hardware platforms for ImageNet and CIFAR-10 are NVIDIA Jetson Xavier and NVIDIA Jetson TX2, respectively.

a DNN according to their importance to the accuracy, we further improve the accuracy with fewer parameters. Experiments on different datasets and hardware platforms demonstrate that, compared with other approaches, HACScale is capable of achieving higher accuracy with fewer parameters and lower latency.

4.2 Model-Aware Adaptive Model Scaling

Model scaling has achieved impressive accuracy by extending the three dimensions (depth, width, and resolution) of CNNs. However, designing an ideal scaling strategy for a given model is extremely onerous due to the large design space formed by the three dimensions. Moreover, the obtained scaling strategy can hardly be shared across models due to the heterogeneity of different models. To address these problems, we propose an efficient model-aware scaling framework to adaptively scale different models for higher accuracy. Our approach utilizes the model balance among the three dimensions as our optimization objective and devotes to scaling models towards better balance, thereby optimizing model accuracy. To achieve the goal, we first model the balance among the three dimensions as *model-data balance*

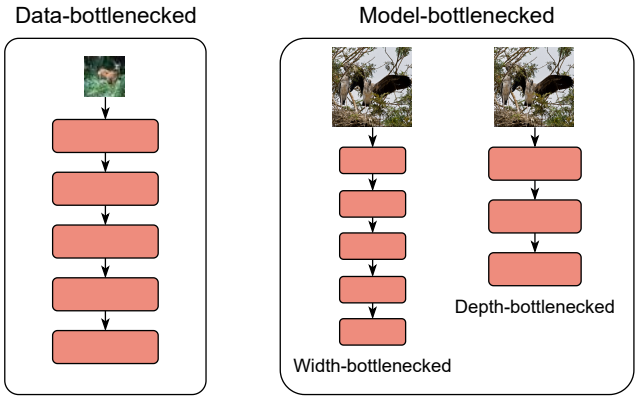


FIGURE 4.6: The heterogeneity of models. As the accuracy of different models is limited by different dimensions, the scaling strategy should be adaptive to achieve better accuracy.

and *structure balance*. Subsequently, we employ an efficient evolutionary algorithm to identify the scaling strategy that can better optimize the two balance metrics. Compared to directly optimizing model accuracy, our approach eliminates the tedious training of models, which greatly reduces the design cost of scaling and thus enables efficient model-aware scaling. Extensive experiments on CIFAR-10, Tiny-ImageNet, and COCO demonstrate the advantages of our method over existing state-of-the-art approaches.

4.2.1 Introduction

In Section 4.1, we introduce a compound model scaling framework to jointly scale multiple dimensions of CNNs. The experimental results reveal that compound scaling is able to achieve higher accuracy at the same computational cost and parameters than single-dimensional scaling. Multi-dimensional scaling has a much larger design space compared to single-dimensional scaling. On the one hand, the large design space enables us to explore more effective scaling strategies for higher accuracy. On the other hand, it also brings larger design costs. How to efficiently couple the scaling of the three dimensions at a given computational budget for better accuracy, however, has always been challenging due to the enormous design space. To find an ideal compound scaling strategy, existing approaches [40, 50] treat the scaling model as a black box and exhaustively search the large design space for the highest accuracy, which inevitably results in a huge search cost. Moreover,

the searched scaling solution for a specific model may only achieve suboptimal performance for other models due to the heterogeneity of models.

In this section, we investigate a central question of compound model scaling for CNNs: *how to efficiently design the scaling strategy for different CNN models to achieve better accuracy and efficiency*? In practise, we empirically observe that models with good balance among the three dimensions are more likely to obtain higher accuracy. To achieve such balance, the compound scaling strategy should be adaptive for different models because of the heterogeneity of models. For instance, as shown in Figure 4.6, for baseline models whose accuracy is limited by input data, we should scale the resolution dimension more than the other dimensions to achieve better balance. To this end, we propose an efficient model-aware compound scaling framework to adaptively scale different models towards better balance, thereby achieving higher accuracy and efficiency. First, we model the balance of a CNN model among the three dimensions into two novel metrics: 1) *model-data balance* and 2) *structure balance*, where the *model-data balance* is to quantify the balance between input data and the network, and the *structure balance* is to quantify the balance of the network architecture between depth and width. Subsequently, taking the compound scaling strategy as the variable and the two balance metrics as the optimization objectives, we formulate the design of the scaling strategy into a typical multi-objective optimization problem. Finally, we employ an evolutionary algorithm to efficiently solve this problem and obtain the scaling strategy that can best optimize the balance of the given model for higher accuracy. By optimizing model balance instead of model accuracy, our framework eliminates the tedious model training during optimization and greatly improves the design efficiency of compound scaling for different models. The contributions of this section are three-fold:

- We propose a model-aware compound scaling framework, where we utilize the balance among the the three dimensions instead of model accuracy as our optimization objective and adaptively scale different models towards better balance for higher accuracy, which significantly reduces the design cost of compound scaling.
- We model the balance among the three dimensions as *model-data balance* and *structure balance*, and then introduce a balance optimization framework to

simultaneously optimize the two balance metrics, where we utilize an evolutionary algorithm to efficiently find the model-aware scaling solution that can better optimize the balance of the baseline model for higher accuracy.

- Extensive experiments on multiple representative datasets and models demonstrate the superiority of our approach over existing compound scaling frameworks.

4.2.2 Problem Formulation

Given a CNN model $\mathcal{N}$ with t layers. The dataflow inside the model can be represented as follows:

$$\mathcal{N} = \bigodot_{i=1,2,...,t} \mathcal{F}_i(X_{<H_i,W_i,C_i>}) \tag{4.7}$$

where $\mathcal{F}_i$ denotes the i-th layer, $X_{<H_i,W_i,C_i>}$ denotes the input feature map of the i-th layer, which has C_i channels and each channel has a spatial size of (H_i, W_i). Let s_d, s_w, s_r be the scaling coefficients of depth, width, and resolution, respectively, and the scaled model can be formulated as follows:

$$\mathcal{N}_{<s_d,s_w,s_r>} = \bigodot_{i=1,2,...,t \cdot s_d} \mathcal{F}_i(X_{<H_i \cdot s_r, W_i \cdot s_r, C_i \cdot s_w>}) \tag{4.8}$$

Given a computational budget, i.e., a MACs budget m, the optimization problem of compound scaling can be defined as follows:

$$\begin{aligned} &\max_{s_d,s_w,s_r} \ Accuracy(\mathcal{N}_{<s_d,s_w,s_r>}) \\ &s.t. \quad MACs(\mathcal{N}_{<s_d,s_w,s_r>}) \leq m \end{aligned} \tag{4.9}$$

where the main optimization objective is to find the optimal scaling coefficients (i.e., s_d, s_w, s_r) for the three dimensions to maximize the accuracy of the scaled model under the given MACs constraint m. By tuning the value of m, we can flexibly generate models with different trade-offs between model MACs and accuracy, thereby accommodating different design considerations.

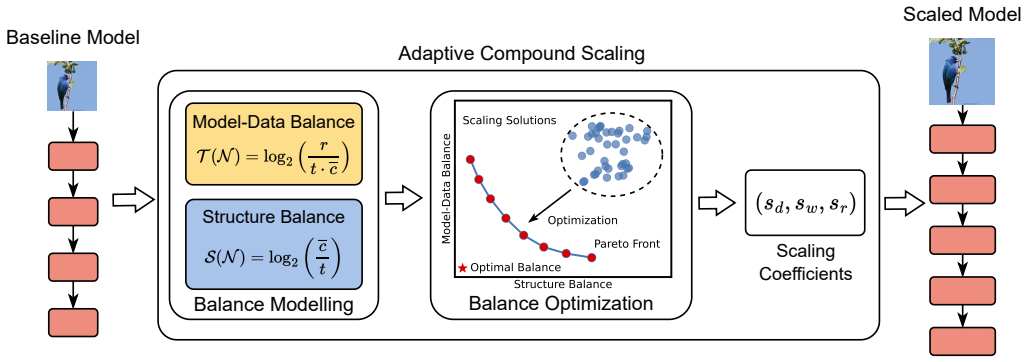


FIGURE 4.7: Overview of our adaptive scaling framework, which can efficiently customize the compound scaling strategy for each given baseline model to generate a more balanced model for higher accuracy.

4.2.3 Framework Overview

In this section, we introduce our adaptive scaling framework (i.e., AdaptScale) in detail. As shown in Figure 4.7, for each input baseline model, our framework first efficiently identifies the current balance, including the *model-data balance* and *structure balance* of the baseline model according to model specifications (i.e., depth, width, and resolution). Subsequently, our framework employs multi-objective optimization to efficiently find the compound scaling strategy for the baseline model that can better optimize the model balance towards the optimal balance, thereby maximizing model accuracy.

4.2.4 Balance Modeling Among Different Dimensions

To solve the problem defined in Equation 4.9 for a given model, the most intuitive approach is to take the accuracy of the scaled model as the optimization objective and directly search for the optimal scaling strategy that can maximize accuracy [40]. However, this approach will be computationally prohibitive as it needs to train hundreds of models and evaluate their accuracy during search. Moreover, such a huge search cost makes it impractical to repeat the search process to find the optimal scaling solution for different network architectures and input data. As discussed before, balancing the three dimensions is critical to achieving higher accuracy. To this end, we propose to utilize the balance of the scaled model as our optimization objective and search for the scaling strategy that can better optimize the balance, thereby improving the accuracy. Without evaluating the accuracy,

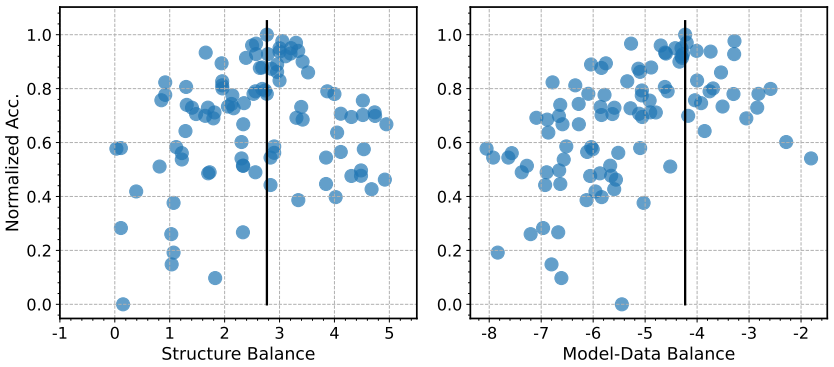


FIGURE 4.8: Accuracy distribution along *structure balance* and *model-data balance*. The black vertical lines are the most likely best *structure balance* and *model-data balance*.

we eliminate the time-consuming training of models at design time, which greatly improves the design efficiency of compound scaling.

To efficiently optimize the balance of CNNs, we first model the balance of CNNs among the three dimensions with two novel metrics: 1) *model-data balance* and 2) *structure balance*. The *model-data balance* is designed to evaluate the balance between input images and the model architecture, which is formulated as follows:

$$\mathcal{T}(\mathcal{N}) = \log_2\left(\frac{r}{t \cdot \overline{c}}\right) \tag{4.10}$$

where r denotes the resolution of input images, t is the number of layers, and $\overline{c}$ is the average number of channels of all layers. Increasing r means the input data can provide more detailed information while increasing t and $\overline{c}$ means we can improve the model capacity to more accurately extract the information from the data. Finally, the balance between data and model capacity can be reflected in the value of $\mathcal{T}$. According to Equation 4.10, the larger the value of $\mathcal{T}$, the more biased the balance towards the resolution dimension. In addition, the *structure balance* is introduced to model the balance of the model architecture between network depth and width, which is formulated as:

$$\mathcal{S}(\mathcal{N}) = \log_2\left(\frac{\overline{c}}{t}\right) \tag{4.11}$$

where the larger the value of $\mathcal{S}$, the more biased the balance towards the width dimension. With the two balance metrics, we are able to efficiently evaluate the balance status of a given model according to the model specifications (i.e., t, $\overline{c}$, r).

After modelling the balance of CNN models, we further investigate how the model balance affects model accuracy. To answer this question, we randomly sample models with diverse *model-data balance* and *structure balance* and evaluate their accuracy. Specifically, we sample 100 models with around 1 billion MACs, which are then trained on ImageNet-100 for 100 epochs. ImageNet-100 is a subset of ImageNet, which can serve as an approximation of ImageNet to reduce the training overhead of sampled models. Specifically, the overhead of training 100 models with 1 billion MACs on ImageNet-100 is equivalent to training 1 model with 10 billion MACs (e.g., EfficientNet-B5) on ImageNet. We utilize RegNet [31] as the baseline architecture for sampling. The residual bottleneck block used in RegNet also serves as the basic block in many SOTA models, and thus the observations on RegNet can be easily generalized to other models. Finally, we summarize the sampling results in Figure 4.8. According to the accuracy distribution of sampled models, we observe that models distributed around specific *model-data balance* (denoted as $\mathcal{T}^*$) and *structure balance* (denoted as $\mathcal{N}^*$) are more likely to achieve higher accuracy. Therefore, we identify ($\mathcal{T}^*$, $\mathcal{N}^*$) as the optimal balance.

4.2.5 Adaptive Model Scaling via Balance Optimization

As discussed in Section 4.2.4, models that are closer to the optimal balance are more likely to achieve satisfactory accuracy. Based on this observation, our adaptive scaling framework employs the *model-data balance* and *structure balance* as the optimization objectives, and searches for a more balanced scaled model, thereby maximizing the accuracy of the scaled model. To achieve this goal, we first quantify the gap between the scaled model and the optimal balance. Specifically, the gap between the scaled model and the optimal *model-data balance* is represented as:

$$\Delta\mathcal{T} = |\mathcal{T}(\mathcal{N}_{<s_d,s_w,s_r>}) - \mathcal{T}^*| = \left|\log_2\left(\frac{s_r \cdot r}{s_d \cdot t \cdot s_w \cdot \overline{c}}\right) - \mathcal{T}^*\right| \tag{4.12}$$

Similarly, the gap between the scaled model and the optimal *structure balance* is formulated as:

$$\Delta\mathcal{S} = |\mathcal{S}(\mathcal{N}_{<s_d,s_w,s_r>}) - \mathcal{S}^*| = \left|\log_2\left(\frac{s_w \cdot \overline{c}}{s_d \cdot t}\right) - \mathcal{S}^*\right| \tag{4.13}$$

TABLE 4.2: Specifications of the evolutionary algorithm used for balance optimization.

Name	Value	Explaination
g	20	*The number of evolutionary generations*
n	20	*The population size*
o	10	*The number of offsprings for each generation*
p_c	0.9	*The crossover probability*
p_m	0.3	*The mutation probability*

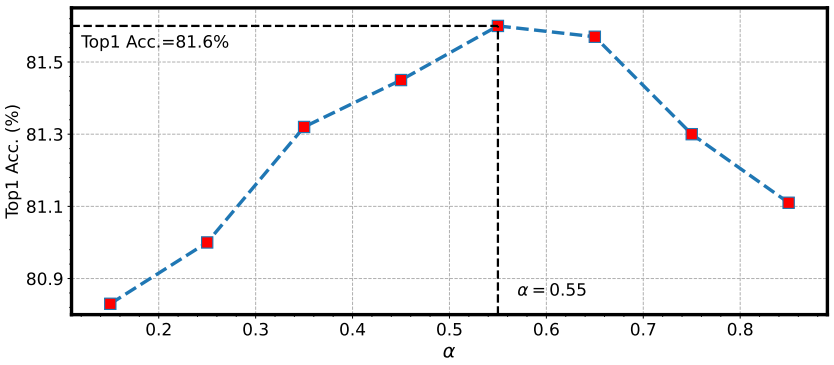


FIGURE 4.9: The impact of α on accuracy. The experiment is performed on ImageNet, and the baseline model is RegNetZ.

where we can see that both $\Delta\mathcal{T}$ and $\Delta\mathcal{S}$ can be directly derived from the specifications of the baseline model and the scaling coefficients of the three dimensions, thus the evaluation cost of $\Delta\mathcal{T}$ and $\Delta\mathcal{S}$ is negligible compared to directly evaluating model accuracy. Subsequently, taking $\Delta\mathcal{T}$ and $\Delta\mathcal{S}$ as the optimization objectives, we reformulate the scaling of CNN models into the following optimization problem:

$$\begin{aligned} \min_{s_d, s_w, s_r} \quad & \Delta\mathcal{T},\ \Delta\mathcal{S} \\ s.t. \quad & MACs(\mathcal{N}_{<s_d, s_w, s_r>}) \leq m \end{aligned} \tag{4.14}$$

To efficiently solve this optimization problem, we exploit an efficient multi-objective evolutionary algorithm, NSGA-II [178], to find the optimal scaling coefficients for the three dimensions. The specifications of the evolutionary algorithm are summarized in Table 4.2. At the end of the evolutionary algorithm, we will obtain a collection of non-dominated solutions. To select a single optimal scaling strategy from those non-dominated solutions, we utilize the linear weighted sum of the two optimization objectives for the final decision making, which is represented as follows:

$$\mathcal{W} = \alpha \cdot \Delta\mathcal{T} + (1 - \alpha) \cdot \Delta\mathcal{S} \tag{4.15}$$

where α is a hyperparameter that determines the importance of $\Delta\mathcal{T}$ and $\Delta\mathcal{S}$. To find the optimal value of α, we conduct empirical experiments and summarize the experimental results in Figure 4.9, where we observe the highest accuracy when $\alpha = 0.55$. Therefore, we let $\alpha = 0.55$ for subsequent experiments. It is worth noting that a more fine-grained search for α may further improve the accuracy, but it also introduces higher exploration costs. Through our balance optimization framework, we can adaptively scale different models towards the optimal balance to achieve higher accuracy.

4.2.6 Experiments

Our framework is implemented with PyTorch and evaluated on three widely utilized datasets: 1) CIFAR-10, 2) Tiny-ImageNet, 3) ImageNet, and 4) COCO. To validate the efficacy of our adaptive scaling framework for different CNN architectures, we select VGG11-BN, MobileNetV2, EfficientNet-B0, and RegNetZ as the baseline models for scaling.

Optimization Settings: To enable a fair comparison, all the scaled models are trained with the same configuration. We use SGD with a momentum of 0.9 as the optimizer. On CIFAR-10, we use a batch size of 128 to train models for 300 epochs. The initial learning rate is 0.1 and decayed by 10 at epoch 100, 200, and 250. On Tiny-ImageNet, models are trained for 200 epochs with a batch size of 1024. The initial learning rate is 2.0 and scheduled by exponential learning rate policy with a decay factor of 0.02. On ImageNet, we use the same learning rate schedule as Tiny-ImageNet to train models for 100 epochs, and the first 5 epochs are for warmup. Also, we use AutoAugment, mixup with a mixup factor of 0.2, label smoothing with $\epsilon = 0.1$, and stochastic weight averaging.

Results on CIFAR-10: We conduct scaling experiments on CIFAR-10 with two baseline models: MobileNetV2 and RegNetZ. We scale the baseline models to different complexities using four compound scaling methods. Among them uniform scaling equally scales all dimensions. For EfficientNet scaling [40] and Fast scaling [46], the scaling strategies are directly borrowed from the corresponding papers. As shown in Figure 4.10, our method outperforms the competitors by a large margin in terms of the trade-off between model MACs and accuracy. For instance, compared

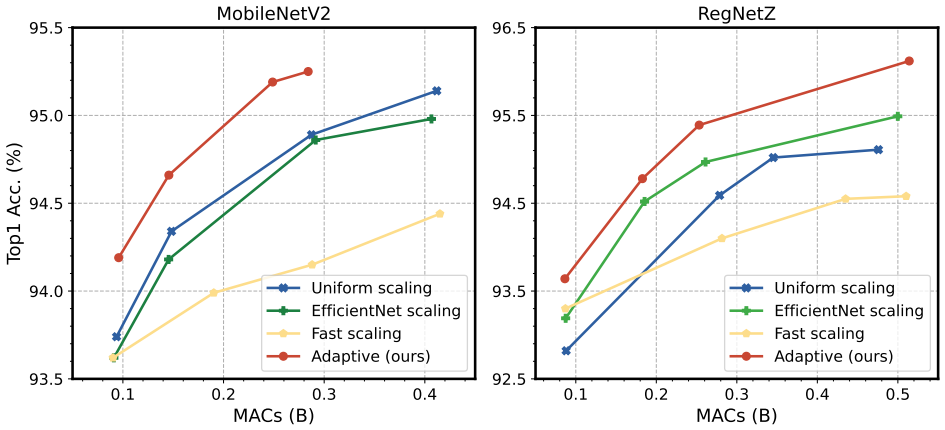


FIGURE 4.10: Comparison with other scaling frameworks in terms of model accuracy and MACs on CIFAR-10.

TABLE 4.3: Experimental results of scaling VGG11-BN with different approaches on Tiny-ImageNet.

Method	Params (M)	MACs (B)	Top1 Acc. (%)
Baseline [42]	9.3	0.6	52.5
Uniform scaling	22.3	2.5	57.9
EfficientNet scaling [40]	20.7	2.5	58.3
Fast scaling [46]	26.2	2.4	56.1
AdaptScale (ours)	**13.7**	**2.4**	**58.4**
Uniform scaling	27.7	4.8	59.3
EfficientNet scaling [40]	25.8	4.8	60.1
Fast scaling [46]	50.8	4.9	58.0
AdaptScale (ours)	**20.7**	**4.8**	**61.1**

to Fast scaling, our method reduces the MACs by 40% (249M v.s. 415M) for MobileNetV2 while achieving 0.75% (95.19% v.s. 94.44%) higher accuracy.

Results on Tiny-ImageNet: For experiments on Tiny-ImageNet, we use another baseline model, VGG11-BN, to validate the efficacy of our approach for different baseline networks. We summarize the performance of different methods in Table 4.3. Again, our approach significantly surpasses the others in terms of model parameters. MACs, and accuracy. Specifically, our scaled model achieves 3.1% higher accuracy with similar model MACs compared to Fast scaling.

Results on ImageNet: On ImageNet, we utilize two advanced CNN models: EfficientNet-B0 and RegNetZ-500MF to compare the performance of different scaling methods. In addition, to further demonstrate the efficacy of our scaling framework, we also compare it with other SOTA large model design methodologies, such as NAS and vision transformer. The results of EfficientNet-B0 are demonstrated in Table 4.4, where our approach observes the highest accuracy under the same

TABLE 4.4: Experimental results of scaling EfficientNet-B0 with different approaches on ImageNet.

Method	MACs (B)	Top1 Acc. (%)	Top5 Acc. (%)
Baseline [40]	0.4	76.2	93.2
EfficientNet scaling [40]	1.5	79.9	95.0
Fast scaling [46]	1.5	80.0	95.2
AdaptScale (ours)	**1.5**	**80.3**	**95.3**

TABLE 4.5: Comparison with different scaling methods and large model design methodologies, such as NAS and vision transformer, over model accuracy and MACs on ImageNet. '-' means no result is reported in related papers.

Method	MACs (B)	Top1 Acc. (%)	Top5 Acc. (%)
Baseline [46]	0.5	77.1%	93.6%
RegNetY-1.6GF [31]	**1.6**	78.0	-
DeiT-S [179]	4.6	79.8	-
EfficientNet-X-B2 [50]	1.9	80.0	-
EfficientNet scaling [40]	2.0	81.3	95.6
Fast scaling [46]	2.0	81.4	95.5
AdaptScale (ours)	2.0	**81.6**	**95.7**
RegNetY-8.0GF [31]	8.0	79.9	-
EfficientNet-X-B4 [50]	10.4	83.0	-
EfficientNet scaling [40]	**7.9**	83.0	96.4
Fast scaling [46]	7.9	82.9	96.3
Swin Transformer [180]	8.7	83.0	-
ConvNeXt-S [51]	8.7	83.1	-
AdaptScale (ours)	8.0	**83.1**	**96.4**
RegNetY-32GF [31]	32.3	81.0	-
DeiT-B [179]	17.6	81.8	-
SENet [16]	42.0	82.7	96.2
AmoebaNet-A [37]	23.0	82.8	96.1
EfficientNet scaling [40]	**16.1**	83.1	96.3
Fast scaling [46]	16.3	83.1	96.5
AdaptScale (ours)	16.2	**83.4**	**96.5**

MACs constraint. For RegNetZ, we scale the baseline model to different computation regimes and summarize their performance in Table 4.5. Also, our approach achieves the highest accuracy in all MACs regimes. For instance, under 2 billion MACs constraint, our scaled model delivers 1.6% higher top-1 accuracy than EfficientNet-X-B2 [50], a new variant of EfficientNet. Compared to the SOTA transformer model, DeiT-S [179], our approach also achieves 1.8% higher accuracy with 56.5% fewer MACs. Moreover, compared to AmoebaNet-A [37], an advanced large model obtained by NAS, we reduce the MACs by 30% while achieving 0.6% higher top-1 accuracy.

Results on COCO: To validate the generalization performance of our method on other vision tasks, we conduct object detection experiments on the COCO2017 dataset. we adopt the scaled models as the backbone of SSD300 [181] and evaluate

TABLE 4.6: Object detection results of SSD300 with different backbones on COCO. The baseline network is RegNetZ.

Backbone	MACs (B)	mAP@[.5, .95] (%)
Baseline [46]	0.9	21.4
ResNet-50 [15]	4.1	24.8
EfficientNet scaling [40]	**2.7**	23.6
AdaptScale (ours)	2.8	**24.9**

TABLE 4.7: The results of ablation experiments on ImageNet.

Model	Structure	Model-Data	MACs (B)	Top1 Acc. (%)
RegNetZ	✓		2.1	81.4
		✓	2.1	81.5
	✓	✓	**2.0**	**81.6**
EfficientNet	✓		1.5	79.8
		✓	1.5	80.0
	✓	✓	**1.5**	**80.3**

their performance on COCO. All backbone networks are fully trained on ImageNet. The detector is trained on COCO for 65 epochs. As shown in Table 4.6, our approach achieves 1.3% higher mAP than EfficientNet scaling with a similar computational cost (i.e., MACs).

Visualization: To better demonstrate the superiority of adaptive scaling, we visualize the class activation map [173] from different scaling frameworks. For each image, we scale the baseline model to the given MACs with different methods. As shown in Figure 4.11, our method can locate the important region more accurately than the competitors, and consequently pay more attention to the important region to generate an accurate prediction.

Ablation Study: Our scaling framework obtains the optimal scaling strategy by optimizing both the *model-data balance* and *structure balance*. In this section, we perform ablation experiments to validate the efficacy of each component. Specifically, we separately remove the *model-data balance* optimization and the *structure balance* optimization from our framework and compare their performance with the completed framework. The experimental results are shown in Table 4.7. For both RegNetZ and EfficientNet, our completed framework achieves the best trade-off between model MACs and accuracy, while removing any component will result in obvious accuracy degradation. The ablation experiments demonstrate that both components of our framework contribute to the final performance.

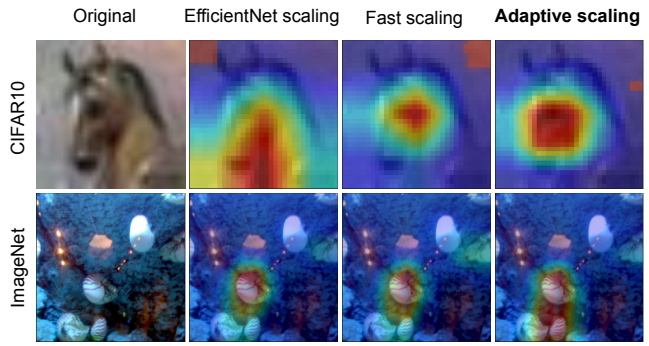


FIGURE 4.11: Class activation maps for different methods. The images are randomly selected from CIFAR-10 and ImageNet.

4.2.7 Summary

We propose AdaptScale, a model-aware compound scaling framework for CNNs to achieve better accuracy and efficiency. To optimally scale different models while minimizing the design cost, we model the balance among the depth, width, and resolution of CNNs and investigate the correlation between model balance and accuracy. By this means, we transform the accuracy optimization problem into a balance optimization problem, which eliminates tedious model training and accuracy evaluation, thereby enabling efficient compound scaling for different CNNs. Powered by AdaptScale, CNN models can be efficiently and flexibly scaled to adapt to edge devices with diverse resources and capabilities. Compared with other methods to improve hardware utilization, such as inference with multiple models in parallel, our method has better flexibility, parallelism, and execution efficiency. Experiments demonstrate the efficacy of the proposed approach.

Chapter 5

Efficient Model Pruning for Execution Efficiency

In this chapter [1], we introduce our efforts in compressing DL models to improve the execution efficiency on resource-constrained embedded hardware platforms. Specifically, we will present two novel research works: 1) inference optimization via fine-grained multi-dimensional pruning and 2) efficient training and inference co-optimization in Section 5.1 and Section 5.2, respectively.

5.1 Inference Optimization via Fine-Grained Multi-Dimensional Pruning

In this section, we propose TECO, a multi-dimensional pruning framework to collaboratively prune the three dimensions (depth, width, and resolution) of CNNs for better execution efficiency on embedded hardware. In TECO, we first introduce a two-stage importance evaluation framework, which efficiently and comprehensively evaluates each pruning unit according to both the local importance inside each dimension and the global importance across different dimensions. Based on the evaluation framework, we present a heuristic pruning algorithm to progressively

[1] The work in this chapter has been published in [182]

prune the three dimensions of CNNs towards the optimal trade-off between accuracy and efficiency. Experiments on multiple benchmarks validate the advantages of TECO over existing SOTA approaches.

5.1.1 Introduction

Over the past decade, the evolution of deep CNNs has been benefiting many deep learning applications [15]. Recently, there is a growing demand to deploy advanced CNNs at the edge to address the concerns of network latency and data privacy [12, 13]. However, modern CNNs are usually equipped with billions of operations. For example, the most popular CNN model, ResNet50 [15], has 4.1 billion MACs, which are computationally prohibitive for embedded hardware [174].

To enable more edge deep learning applications, such as autopilot and smart cameras, to be benefited from the advances of CNNs, efforts have been made to compress CNNs for a better trade-off between execution efficiency and accuracy. Neural network pruning [13, 52, 53, 126, 183, 184], as one of the promising model compression techniques, reduces the complexity of CNNs by removing redundant parameters and computation from the three dimensions (depth, width, resolution) of CNNs. In this way, model pruning effectively reduces the computing time and memory consumption, so that the models can be deployed onto more memory-constrained edge devices for efficient execution. Specifically, width pruning [52, 53, 184–188] devotes to compressing models by removing less important channels, while depth pruning [64, 121, 126, 184] conducts pruning at a coarser granularity (i.e., layer). More recently, resolution pruning [65, 74, 189] has been proposed to compress the spatial redundancy in input images, which also effectively reduces the complexity of CNNs. However, the aforementioned approaches mainly focus on pruning a single dimension while ignoring the redundancy in the other dimensions, which can only achieve a sub-optimal trade-off between model accuracy and execution efficiency.

In this section, we propose a multi-dimensional pruning framework, TECO, to coordinately prune the three dimensions of CNNs. To accurately identify redundant units in the three dimensions, we first propose an inter-dimensional evaluation strategy (ITES) to comprehensively evaluate the importance of pruning units

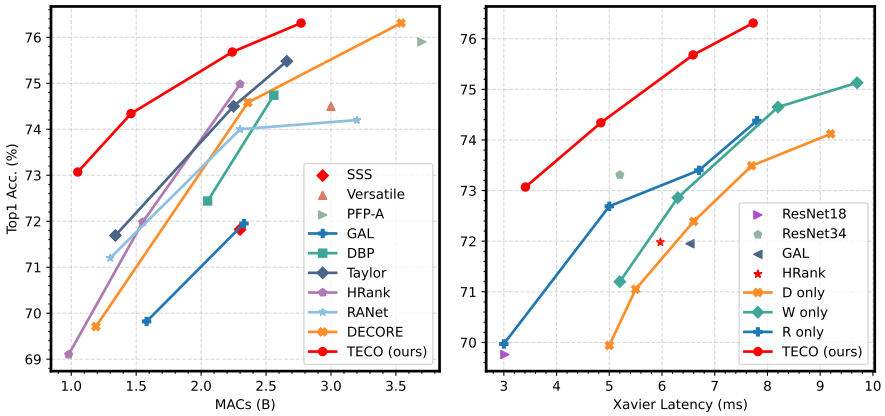


FIGURE 5.1: Experimental results of different methods in terms of model MACs, accuracy, and inference latency. The baseline network is ResNet50, which is trained on ImageNet. The latency is measured on Jetson Xavier with a power budget of 30W.

across different dimensions. In ITES, we integrate the contribution of each pruning unit to model complexity, accuracy, and inference latency into a unified metric, global importance, according to which the unit with the lowest global importance is considered redundant and can be safely removed. However, as ITES needs to collect multiple metrics for comprehensive evaluation, directly applying ITES to traverse all units of the three dimensions will be extremely time-consuming. To this end, we also introduce an inner-dimensional evaluation strategy (INES) to first quickly evaluate units within each dimension and identify the most redundant unit of each dimension. By this means, ITES only needs to be performed on the most redundant unit of each dimension for the final pruning decision. INES reduces the pruning candidates for each dimension from multiple to one, which reduces the evaluation cost of ITES and enhances the pruning efficiency significantly. On top of the two-step evaluation framework composed by INES and ITES, we design a heuristic pruning algorithm, which utilizes INES and ITES to progressively identify and prune redundant units. In this way, we can efficiently search for the optimal tiny architecture for resource-constrained embedded devices in the huge design space formed by the three dimensions.

Our main contributions are three-fold:

- We introduce an inter-dimensional importance evaluation strategy (ITES) to evaluate the importance of units across different dimensions. We integrate the contribution of each unit to model complexity, accuracy and latency into

a comprehensive metric, global importance, which enables us to accurately identify redundant units in the three dimensions.

- We also propose an inner-dimensional importance evaluation strategy (INES) to quickly evaluate the local importance of units within each dimension, which reduces the pruning choices for each dimension from multiple to one, alleviating the evaluation overhead of ITES and improving the pruning efficiency of our framework.
- Based on ITES and INES, we design a heuristic pruning algorithm to progressively prune the three dimensions of CNNs. By iteratively identifying and removing redundant units with INES and ITES, our pruning algorithm can efficiently find the optimal tiny model for edge devices in the huge design space of multi-dimensional pruning.

As shown in Figure 5.1, our TECO obtains 3.97% higher top-1 accuracy on ImageNet than HRank [53] with similar MACs. For on-device acceleration, TECO is 1.91× faster than GAL [184] while still achieving 1.12% higher accuracy.

5.1.2 Framework Overview

In this section, we first outline the design of TECO, and then introduce each subcomponent in detail. As demonstrated in Figure 5.2, Given a CNN model $\mathcal{N}$, we first quickly evaluate the local importance of each unit inside each dimension with INES. Subsequently, the unit with the lowest local importance in each dimension is selected to perform ITES, where the three units of different granularities are fairly compared according to their global importance, and then the unit with the lowest global importance score is pruned safely. Afterwards, the model is fine-tuned to restore accuracy for the next pruning iteration. Finally, INES and ITES are executed iteratively to progressively prune the three dimensions of the given CNN model $\mathcal{N}$ for a better trade-off between model accuracy and execution efficiency.

5.1.3 Inter-Dimensional Importance Evaluation

Compared to single-dimensional pruning [53, 64, 74], a major challenge faced by multi-dimensional pruning is to effectively evaluate and compare the importance

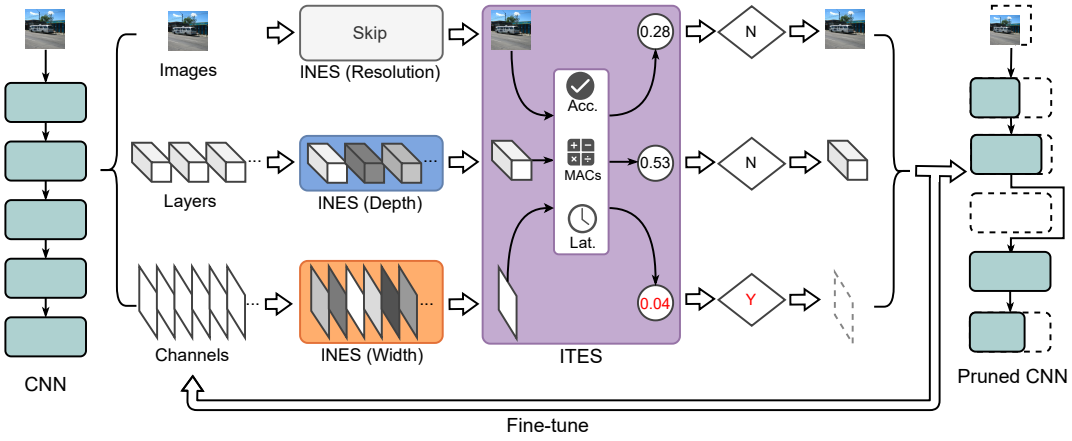


FIGURE 5.2: Overview of TECO, where INES evaluates the local importance of units inside each dimension and ITES evaluates the global importance of units across different dimensions. The inner-dimensional evaluation is skipped for the resolution dimension (See Section 5.1.4 for the detailed reason).

of the pruning units of different dimensions. Intuitively, the pruning units of different dimensions (layer for depth, channel for width, pixel for resolution) are at different granularities, and pruning them can lead to diverse model complexity and accuracy. Moreover, for edge computing, pruning different dimensions also results in different run-time latency on embedded devices. However, single-dimensional pruning [52, 53, 64] mainly considers model accuracy as the only metric to evaluate the importance of pruning units, which is inapplicable for evaluating units of different dimensions. To this end, we propose an inter-dimensional evaluation strategy (ITES) to comprehensively evaluate the importance of units across different dimensions according to their impacts on accuracy, model complexity, and on-device inference latency.

Accuracy: We quantify the contribution of a unit to accuracy as the increase in the cross entropy loss of the model prediction when removing this unit from the model. The cross entropy loss of image classification tasks can be defined as:

$$L_{CE}\left(\mathcal{N}\right) = -\sum_{i=1}^{n} t_i \log(p_i) \tag{5.1}$$

where t_i is the ground truth probability for class i, p_i is the predicted probability of model $\mathcal{N}$, and n is the number of classes. Let u be an arbitrary pruning unit of

the three dimensions, the impact of u on accuracy is formulated as:

$$\begin{aligned}\mathcal{A}(u) &= L_{CE}\left(\mathcal{N}'\right) - L_{CE}\left(\mathcal{N}\right) \\ &= \sum_{i=1}^{n} t_i \log(p'_i) - \sum_{i=1}^{n} t_i \log(p_i) = \sum_{i=1}^{n} t_i \log(\frac{p'_i}{p_i})\end{aligned} \tag{5.2}$$

where $\mathcal{N}'$ is the pruned model by removing u from $\mathcal{N}$.

Model Complexity: We use model MACs to quantify the complexity of CNNs as all three dimensions can affect model MACs while parameters are only related to depth and width. Therefore, the impact of u on model complexity can be efficiently measured by calculating the MACs reduction achieved by removing u, which is represented as:

$$\mathcal{M}(u) = |\text{MACs}(\mathcal{N}') - \text{MACs}(\mathcal{N})| \tag{5.3}$$

On-Device Latency: The most intuitive way to evaluate the impact of u on latency is to measure the latency reduction. However, directly measuring the latency reduction on embedded devices during pruning will incur a huge time cost and reduce the pruning efficiency. To this end, we introduce a dimension-wise latency model to efficiently estimate the latency reduction according to the MACs reduction of each dimension. To build the dimension-wise latency model, we vary the three dimensions of ResNet50 to sample models with different MACs and measure their latency on the target device. The sampling results are summarized in Figure 5.3, which demonstrates a linear relationship between the latency and MACs. Therefore, we formulate the latency model for each dimension as follows:

$$l_s = a_s \cdot m + c_s \tag{5.4}$$

where l_s is the predicted latency for dimension $s \in \{d, w, r\}$ and m is the model MACs. a_s and c_s are dimension-wise hyperparameters, which are fitted by Least Squares Method with the sampled data. Since the residual network architecture of ResNet50 is widely utilized in many advanced CNNs, the established latency model can be well generalized to other advanced CNNs. Let s be the dimension of unit u, the impact of u on latency (i.e., the latency reduction) is derived as:

$$\mathcal{T}(u) = |l'_s - l_s| = a_s(|m' - m|) = a_s\mathcal{M}(u) \tag{5.5}$$

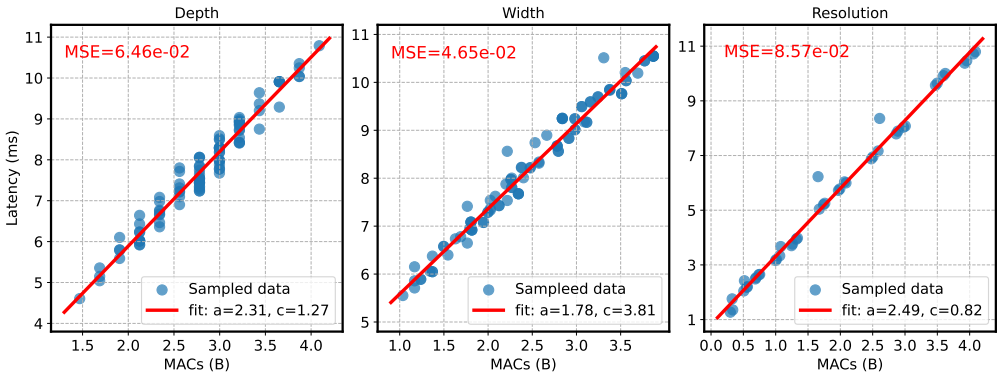


FIGURE 5.3: Latency distributions obtained by separately pruning the three dimensions. The low mean squared errors (MSE) reveal that the proposed latency model well fits the sampled data.

where m' and l'_s are the MACs and estimated latency of the pruned model $\mathcal{N}'$. Through Equation 5.5, we can quickly evaluate the impact of u on latency using the MACs reduction and the dimension-wise latency hyperparameter a_s. As shown in Figure 5.3, the value of the latency hyperparameter a_s varies across dimensions, which reveals that, for different dimensions, the same reduction on MACs can lead to diverse latency reduction.

Global Importance: Finally, we combine the impact of u on accuracy, model complexity, and latency as a unified metric coined global importance, which is formulated as:

$$\mathcal{I}(u) = \frac{\mathcal{A}(u)}{\alpha\mathcal{M}(u) + (1-\alpha)\mathcal{T}(u)} \tag{5.6}$$

where $\alpha \in [0, 1]$ is a hyperparameter to control the contribution of $\mathcal{M}(u)$ and $\mathcal{T}(u)$, which provides ITES with enough flexibility to accommodate different design considerations. Specifically, by increasing the value of α, ITES will focus more on the reduction of model complexity (i.e., MACs). Otherwise, the on-device latency will be the main consideration of ITES. In our experiments, we empirically set $\alpha = 0.5$ to equalize their contribution to the global importance. According to Equation 5.6, unit u is considered less important if pruning it can bring more significant reduction in model MACs and latency with less increase in prediction loss.

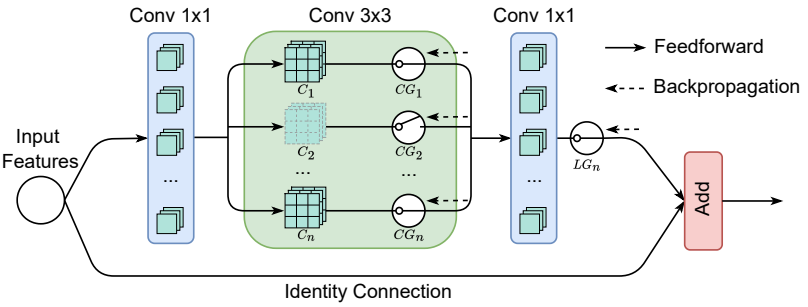


FIGURE 5.4: Architecture of the fully gated residual bottleneck block with channel gates and a layer gate.

5.1.4 Inner-Dimensional Importance Evaluation

ITES effectively evaluates units across different dimensions using global importance. However, obtaining the global importance of a unit is relatively time-consuming due to the calculation of $\mathcal{A}(u)$, $\mathcal{M}(u)$, and $\mathcal{T}(u)$, and thus directly using ITES to traverse the three dimensions will lead to an expensive evaluation cost, thereby degrading the efficiency of our pruning framework. Therefore, we further propose an inner-dimensional evaluation strategy (INES) to cooperate with ITES to efficiently evaluate all units. INES first quickly evaluates the local importance of units inside each dimension, and then only the unit with the lowest local importance in each dimension will be selected to perform ITES for its global importance. In this way, ITES will be performed only on three units and thus the evaluation overhead is reduced significantly. Finally, the unit with the lowest global importance will be considered redundant and removed. Through such a two-step evaluation mechanism, we are capable of accurately and efficiently identifying the redundant units in the three dimensions.

To efficiently evaluate the local importance of units inside each dimension, we design a fully gated residual bottleneck block. The block architecture is demonstrated in Figure 5.4, where each channel is followed by a channel gate (CG). Meanwhile, there is a layer gate (LG) at the end of the block. These gates are introduced for two reasons: (1) to control the pruning of each channel or the whole layer by setting the gate's weight to 0 (pruned) or 1 (preserved); (2) to quickly calculate the local importance for inner-dimensional evaluation.

Local Importance: Inspired by the channel pruning approach proposed in [52],

we approximate the local importance of a channel with the gradient of the corresponding channel gate, which can be formulated as follows:

$$\mathcal{I}_i^w = \left(\frac{\partial L_{CE}}{\partial CG_i} \right)^2 \tag{5.7}$$

where $\mathcal{I}_i^w$ is the local importance of the i-th channel and CG_i is the gate of the i-th channel. Further, we extend this idea to the depth dimension, where we add a layer gate at the end of each layer to collect the layer-level gradients and utilize the layer-level gradients to quantify the importance of each layer:

$$\mathcal{I}_i^d = \left(\frac{\partial L_{CE}}{\partial LG_i} \right)^2 \tag{5.8}$$

where $\mathcal{I}_i^d$ and LG_i represent the local importance and the gate of the i-th layer, respectively. Thanks to our fully gated block architecture, we are able to simultaneously obtain the gradient of all channel gates and layer gates by only performing backpropagation once, and thus the evaluation cost of INES is reduced significantly. In practice, we randomly select multiple images from the training set to perform backpropagation and average their gradients to obtain a more consistent and accurate estimation of the local importance of each unit. In our test, we empirically observe that 5,000 images are adequate to produce an accurate estimation. Using more images brings only negligible accuracy improvement at the expense of larger time overhead, degrading the efficiency of our approach. The backpropagation of 5,000 images only takes about 2.94 seconds on a RTX3090 GPU, which validates the efficiency of INES.

For the resolution dimension, selectively pruning the millions of images in large-scale datasets (e.g., ImageNet) is unpractical for our framework due to the enormous overhead [65]. Instead, we implement resolution pruning by uniformly shrinking all images, which eliminates the evaluation cost of INES for the resolution dimension, enhancing the efficiency of our method.

5.1.5 Heuristic Architecture Descent

To efficiently find the optimal architecture for a given resource budget, we further propose a heuristic pruning algorithm, which progressively executes INES and ITES

Algorithm 2: Heuristic Architecture Descent

Data: overparameterized CNN $\mathcal{N}$, training dataset D_t, evaluation dataset D_v, pruning iterations n
Result: pruned network $\mathcal{N}_p$
$iter \leftarrow 0$;
while $iter < n$ **do**
  `/* Inner-dimensional evaluation */`
  $(d^*, w^*, r^*) \leftarrow \text{INES}(\mathcal{N}, D_v)$
  `/* Inter-dimensional evaluation */`
  $dim \leftarrow \text{ITES}(\mathcal{N}, D_v, d^*, w^*, r^*)$
  if *dim is depth* **then**
    Prune$(\mathcal{N}, d^*)$ `/* Prune the layer */`
  else if *dim is width* **then**
    Prune$(\mathcal{N}, w^*)$ `/* Prune the channel */`
  else if *dim is resolution* **then**
    Prune$(\mathcal{N}, r^*)$ `/* Prune the image */`
  Fine-tune$(\mathcal{N}, D_t)$
  $iter \leftarrow iter + 1$
$\mathcal{N}_p \leftarrow \mathcal{N}$

to prune redundant units from the three dimensions. Inspired by gradient descent, we coin this algorithm heuristic architecture descent as the architecture is gradually descending along the direction that achieves the best efficiency-accuracy trade-off.

The proposed heuristic architecture descent is defined in Algorithm 2. Given a baseline network $\mathcal{N}$, a training dataset D_t, and an evaluation dataset D_v that consists of 5,000 randomly selected images from D_t, we perform the pruning operation for n iterations. For each iteration, we first conduct INES for $\mathcal{N}$ on D_v to obtain the local importance of each unit within each dimension. Based on the local importance, we select the least important unit of each dimension (d^* for depth, w^* for width, and r^* for resolution) to perform ITES for their global importance, according to which the unit with the lowest global importance score will be pruned. Subsequently, the model is fine-tuned on D_t for 1 epoch to retain its accuracy for the next pruning iteration, where the optimizer for fine-tuning is SGD with a learning rate of 1e-4. The completely pruned model $\mathcal{N}_p$ will be generated after n iterations. The value of n depends on the resource budget. The smaller the budget, the larger the value of n. In practise, 10 iterations are adequate to obtain a compact model, which indicates that the time cost of pruning is much smaller than the main training of CNNs. Finally, the completely pruned model will be trained from scratch for final accuracy. Compared to global search algorithms [190] that

TABLE 5.1: Comparison with SOTA pruning approaches on ImageNet. The baseline network is ResNet50. {d, w, r} indicate the pruned dimensions in different methods. TECO-S, TECO-M, and TECO-L are obtained from our pruning framework by performing different numbers of pruning iterations.

Method	d	w	r	MACs (B)	Top-1 Acc. (%)	Top-5 Acc. (%)
ResNet50 [15]				4.10	76.80	93.38
DECORE-4 [185]		✓		1.19	69.71	89.37
ResNet18 [15]	✓			1.80	69.76	89.08
GAL-1 [184]	✓	✓		1.58	69.82	89.75
Bilinear [29]			✓	1.10	69.97	89.19
HAP [186]		✓		1.34	71.18	-
Taylor [52]		✓		1.34	71.69	-
HRank [53]		✓		1.55	71.98	91.09
DBP-0.5 [64]	✓			2.05	72.44	-
TECO-S (ours)	**✓**	**✓**	**✓**	**1.05**	**73.07**	**91.18**
SSS-26 [126]	✓	✓		2.33	71.82	90.79
GAL-0.5 [184]	✓	✓		2.33	71.95	90.94
ResNet34 [15]	✓			3.70	73.31	91.42
Bilinear [29]			✓	2.53	73.40	91.30
RANet [74]			✓	2.30	74.00	-
Taylor [52]		✓		2.25	74.50	-
DBP-0.4 [64]	✓			2.56	74.74	-
HRank [53]		✓		2.30	74.98	92.33
DR-ResNet50 [65]			✓	2.30	75.30	92.20
TECO-M (ours)	**✓**	**✓**	**✓**	**2.24**	**75.68**	**92.79**
SSS-32 [126]	✓	✓		2.82	74.18	91.91
Bilinear [29]			✓	3.00	74.30	91.90
HAP [186]		✓		2.71	75.12	-
C-SGD70 [191]		✓		**2.60**	75.30	92.50
Taylor [52]		✓		2.66	75.48	-
PFP-A [192]		✓		3.70	75.90	92.80
DECORE-8 [185]		✓		3.54	76.31	93.02
TECO-L (ours)	**✓**	**✓**	**✓**	2.87	**76.34**	**93.20**

directly search the huge design space for the optimal solution, our heuristic pruning algorithm greatly reduces the exploration overhead.

5.1.6 Experiments

In this section, we conduct extensive experiments to validate the superiority of TECO over other SOTA approaches in terms of accuracy, model complexity, and run-time latency. In our experiments, we select three widely utilized embedded platforms: 1) Jetson Nano, 2) Jetson TX2, and 3) Jetson Xavier, to deploy pruned models obtained from different approaches and measure their actual inference latency. Also, we perform ablation experiments to validate the efficacy of each component.

Experiments on ImageNet: On the most representative large-scale dataset, ImageNet, we train all models from scratch for 120 epochs using SGD optimizer with a momentum of 0.9. The batch size for training is 1024. Correspondingly, the initial learning rate is set to 1.6 and decayed by cosine annealing scheduling [193].

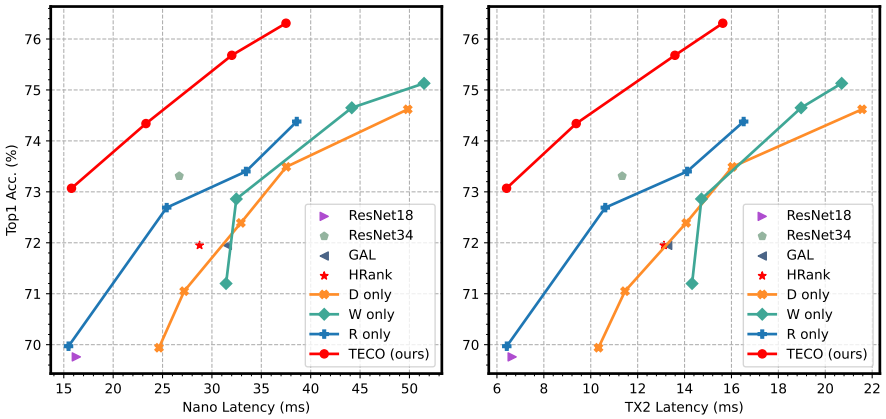


FIGURE 5.5: Comparison of inference latency on Jetson Nano and Jetson TX2. For the consistency of results, all models are executed for 30 iterations to get the average inference latency.

In addition, the learning rate for fine-tuning is 1e-3. To prevent overfitting, we also use label smoothing with $\epsilon = 0.1$. The results are summarized in Table 5.1, which shows that our approach achieves the highest accuracy across a wide spectrum of model MACs. Specifically, in the low compute regime, our TECO-S achieves 3.36% higher top-1 accuracy with about 12% fewer MACs compared to DECORE-4 [185]. In comparison with GAL-1[184], TECO-S also improves the top-1 accuracy by 3.25% while reducing the MACs by 33.5%. In the middle MACs regime, TECO-M achieves 0.7% higher top-1 accuracy with similar MACs compared to HRank[53]. In the highest MACs regime, TECO-L outperforms SSS-32 [126] with 2.16% higher top-1 accuracy.

Comparison of On-Device Acceleration: In this experiment, we evaluate the run-time latency of models pruned by different frameworks on three widely used edge platforms: Jetson Xaiver, Jetson TX2, and Jetson Nano. The corresponding results are shown in Figure 5.1 and Figure 5.5, respectively. We observe that our approach surpasses all competitors on all devices. Specifically, our method achieves 3.7% higher accuracy (75.68% v.s. 71.98%) than HRank [53] with a similar latency (13.59 ms v.s. 13.12 ms) on Jetson TX2. On Jetson Nano, a more resource-economic edge device, our TECO-S observes 1.12% higher accuracy (73.07% v.s. 71.95%) than GAL [184] with only 50% latency budget (15.78 ms v.s. 31.56 ms). Similar results are also observed on Xavier. The experiment validates the efficacy of our method in optimizing the execution efficiency of CNNs on various devices.

Experiments on CIFAR-10: To validate the efficacy of TECO in extremely

resource-constrained environments (e.g., TinyML), we conduct experiments to further compress small models for edge devices. We use ResNet110 [15] as the baseline network and use CIFAR-10 as the dataset. ResNet110 is a lightweight CNN specially designed for tiny images, which only contains 1.7M parameters and 252M MACs. All models are trained for 200 epochs using SGD optimizer. The batch size is 128 and the initial learning rate is 0.1, which is decayed by cosine annealing [193]. The latency of all models is measured on Jetson Nano. The results in Table 5.2 show that our method achieves the best latency-accuracy trade-off. For instance, compared to HRank-2 [53], we achieve 1.29% higher accuracy with only 84.2% inference latency. Interestingly, our method reduces the MACs of the baseline ResNet110 by 57% while still achieving 0.44% higher accuracy, which is because CNNs usually overfit on CIFAR-10 [185], and our approach greatly mitigates the overfitting by comprehensively reducing the redundancy in the three dimensions, thereby improving the accuracy. Besides MACs, the reduction in model parameters also plays a significant role in optimizing the memory consumption and execution efficiency of DNNs. Similar to other model pruning techniques, our method can also optimize the memory occupation of CNNs. With fewer parameters, the compressed models can be deployed onto more memory-constrained edge devices for execution. Table 5.2 shows that our method significantly reduces the model parameters and optimizes the memory efficiency of CNNs. It is worth noting that, our approach still achieves lower latency than HRank even if our approach uses more MACs and parameters. This is because the actual latency is not only correlated to MACs and parameters but also highly depends on the model architecture. For example, as network layers are usually executed sequentially, narrow but deep networks may have higher latency than wide but shallow networks even if the deep networks have fewer MACs and parameters. Since our method jointly prunes multiple dimensions including depth, we comprehensively reduce the redundancy of the depth dimension and obtain a shallower network, so we can also get lower latency even if we use more MACs and more parameters.

Ablation Study: We introduce two novel evaluation strategies, INES and ITES in TECO to enable efficient and accurate multi-dimensional pruning. To validate the efficacy and efficiency of each component, we perform comprehensive ablation experiments on ImageNet. First, we perform multi-dimensional pruning with both INES and ITES removed from TECO. The results in Table 5.3 show that, without INES and ITES, the multi-dimensional pruning only achieves comparable accuracy

TABLE 5.2: Experiments on CIFAR-10. "-" means no source code is provided for reproducing the experiment.

Method	MACs (M)	Params (M)	Latency (ms)	Top-1 Acc. (%)
ResNet110 [15]	252.89	1.72	4.98	93.50
GAL-0.5 [184]	130.20	0.95	2.96	92.55
HRank-2 [53]	**79.30**	**0.53**	3.49	92.65
HRank-1 [53]	105.70	0.70	3.92	93.36
DECORE-300 [185]	96.66	0.61	-	93.50
DECORE-500 [185]	163.30	1.11	-	93.88
TECO-Tiny (ours)	108.60	0.98	**2.94**	**93.94**

TABLE 5.3: Experimental results of ablation experiments. The baseline network is ResNet50 and the latency is measured on Xavier. We quantify the pruning cost of different methods as the time consumed on a single RTX3090 GPU for pruning.

Method	MACs (B)	Latency (ms)	Top-1 Acc. (%)	Cost (hour)
ResNet50	4.10	10.60	76.80	-
Depth only	2.30	6.60	72.39	7.52
Width only	2.30	8.20	74.65	26.44
Resolution only	2.50	6.70	73.40	0.00
w/o INES + w/o ITES	2.17	7.34	74.67	20.77
INES + w/o ITES	2.30	6.84	75.28	21.26
w/o INES + ITES	**2.12**	**6.42**	**75.71**	179.42
Ours (INES + ITES)	2.24	6.51	75.68	30.12

to single-dimensional pruning. It is worth noting that, for resolution pruning, we directly shrink the resolution of images and do not evaluate the pruning units of the network, thus the pruning cost is negligible. Then, we separately retrieve INES and ITES, and both of which observe a remarkable accuracy gain. When only using ITES to evaluate all units, the pruned model achieves the best performance in terms of MACs, inference latency, and accuracy, which proves that ITES is able to accurately identify the redundancy in the three dimensions. However, this strategy also results in an unbearable time cost. In contrast, by collaboratively using INES and ITES, we achieve competitive model performance while reducing the pruning cost by 83.21% compared to using ITES alone, which greatly increases the efficiency of our framework. The ablation study reveals that both INES and ITES contribute to our pruning framework.

Interpretability Analysis: To gain insight into the advantages of our approach, we visualize the class activation map [173] for TECO and single-dimensional pruning. The results are demonstrated in Figure 5.6, where we observe that single-dimensional pruning approaches only focus on part of the foreground object of input images, which may overlook important features and consequently lead to wrong predictions. In contrast, our TECO utilizes the whole foreground object

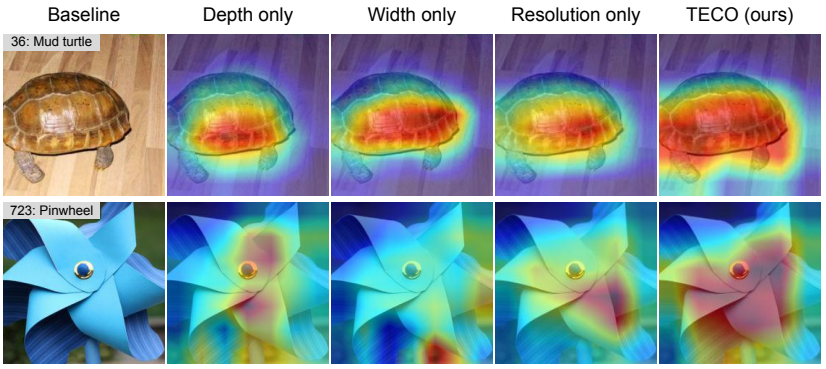


FIGURE 5.6: The class activation map (CAM) for different pruning methods. The region in red is the most contributing part of the image. The images are randomly selected from ImageNet.

for prediction, which effectively addresses the aforementioned problem of single-dimensional pruning, significantly improving model accuracy.

5.1.7 Summary

In this section, we present a multi-dimensional pruning framework, TECO, to jointly prune the three dimensions of CNNs for embedded devices. First, we introduce an inter-dimensional evaluation strategy (ITES), which enables comprehensive evaluation of units across different dimensions with a novel metric named global importance, thereby accurately identifying the redundancy in the three dimensions. Meanwhile, we also propose an inner-dimensional evaluation strategy (INES) to efficiently evaluate units inside each dimension with local importance. By collaboratively using ITES and INES, we accurately and efficiently identify the redundancy in the three dimensions. Based on INES and ITES, we further propose a heuristic pruning algorithm, which utilizes INES and ITES to progressively identify and prune redundant units in the three dimensions. By this means, our pruning framework efficiently explores the huge design space formed by the three dimensions and finds the optimal tiny model for embedded devices. Extensive experiments validate the efficacy and efficiency of our approach.

5.2 Efficient Training & Inference Co-Optimization

EdgeAI deploys deep learning models at the edge, mitigating network latency and protecting data privacy. As edge hardware devices are usually resource-constrained, model compression has been proposed to reduce the overhead of models and facilitate their deployment onto edge devices. However, existing compression approaches mainly focus on reducing the inference overhead of models while ignoring the training overhead, which loses the opportunity to update the deployed model with private data on edge devices due to the huge training cost. To address this issue, in this section, we propose TICO, a co-optimization framework to optimize both the training and inference performance of deep learning models. In TICO, we first introduce a novel multi-objective pruning approach, where we take both training and inference performance as optimization objectives, and then formulate the pruning of a model as a multi-objective optimization problem. Subsequently, we design an evolutionary algorithm to efficiently search for the optimal pruning decision. Moreover, to further compress the training cost, we also propose a resolution-adaptive training strategy, which trains models with a small image size at early training epochs and progressively increases the size of training images. Compared to the traditional training paradigm which trains a model with the same large image size throughout the whole training process, our approach significantly reduces the training cost and improves the training performance of models on edge devices. Extensive experiments on CIFAR-10 and ImageNet validate the superiority of TICO over other SOTA approaches.

5.2.1 Introduction

In Section 5.1, we introduce our multi-dimensional pruning framework for optimizing the inference performance of CNNs on edge devices. As the application scenarios of CNNs get more diverse and complex, the conventional paradigm of EdgeAI that handles the training and inference of CNNs separately encounters new challenges. Specifically, as shown in Figure 5.7, a CNN model is firstly trained on cloud servers, and then the fully trained model will be deployed onto edge devices for inference. Due to the distribution discrepancy between the training data and the inference data in the deployment environments, the model accuracy may degrade significantly when the model is deployed to different environments. To

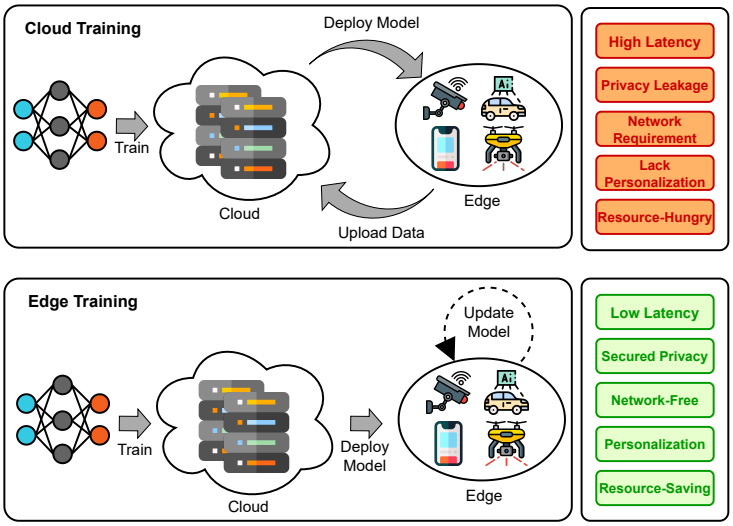


FIGURE 5.7: Differences between cloud training and on-device training. Compared to cloud training, on-device training has significant advantages in model performance and data privacy.

minimize the distribution discrepancy between training data and real-world inference data for higher model accuracy, an intuitive approach is to periodically upload the collected data to the server to update the model and download the updated model to the device. However, this will also sacrifice data privacy and deployment efficiency.

More recently, a new paradigm named on-device deep learning is proposed to customize the model with real-world data for higher accuracy without compromising data privacy [194]. As shown in Figure 5.7, in this paradigm, only the initial training is performed on cloud servers, while both the inference and model updating are completed on edge devices, which enables the personalization of the model while protecting data privacy. However, the training costs for updating the model, such as memory occupation and time cost, are much more heavier than inference, bringing great challenges to edge hardware devices.

To facilitate the deployment of such on-device DL systems, in this section, we propose to co-optimize the training and inference overhead of CNN models and improve the efficiency of the system on resource-constrained edge devices. As aforementioned, existing pruning approaches are mainly proposed to optimize the inference efficiency of CNN models, which seldom consider the training efficiency and thus can hardly benefit the training of the on-device DL system. To overcome

this problem, we propose a multi-objective model pruning framework, which integrates both training performance and inference performance as the optimization objectives and efficiently searches for the pruning strategy that can maximize both the training and inference efficiency. Different from prior works that only prune a single dimension, we emphasize that pruning different dimensions can cause diverse impacts on our multiple optimization objectives. Therefore, in our pruning framework, we include all three dimensions of CNN models in our design space and collaboratively prune the three dimensions towards optimal training and inference efficiency. The larger design space formed by the three dimensions enables us to find better pruning solutions for the co-optimization of training and inference. However, it also brings higher search costs for the pruning decision. To reduce the exploration costs in the large design space, we introduce a novel performance estimator to quickly evaluate different pruning solutions, with which we improve the design efficiency of our framework significantly. In addition to compressing the model itself, we observe that redundancy also exists in the training process. Specifically, the conventional training paradigm uses the same image resolution (i.e. image size) for training and inference, while FastScaling [46] points out that the size of images is of great significance for training efficiency and a large size can lead to very slow training and huge memory occupation. To this end, many works [177, 195, 196] propose to train CNNs with smaller sizes to save computation and memory consumption, but they often cause a drop in accuracy. To compress the training overhead while not compromising accuracy, we introduce a novel resolution-adaptive training (RAT) strategy. Different from previous works that split the training process into several stages and tune the resolution of each stage manually, our RAT automatically grows the resolution of each epoch with a cosine-like scheduling strategy. At early epochs, the model will be trained with a very small resolution, which can speed up the training and reduce memory consumption significantly. As the training goes by, the resolution will also grow automatically to restore accuracy. By means of RAT, we can remarkably optimize the resource consumption and time cost of the whole training process, facilitating the deployment of on-device DL systems onto embedded devices.

Our main contributions can be summarized as follows:

- We propose a multi-objective pruning (MOP) approach to compress CNN models for better training and inference performance. By integrating the

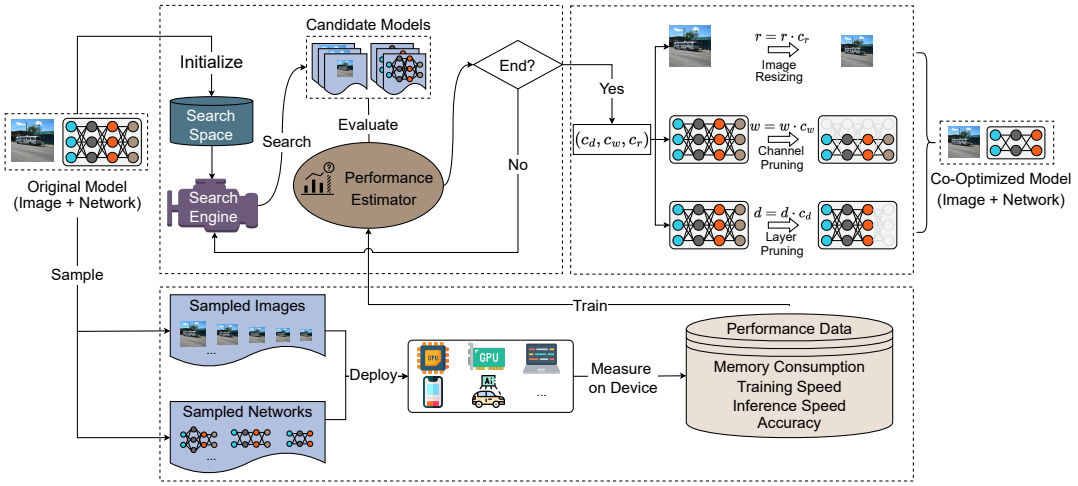


FIGURE 5.8: Overview of the proposed co-optimization framework for on-device training and inference.

pruning of CNN architectures and input images searching for the optimal compound pruning scheme, MOP effectively addresses the training and inference bottlenecks of CNN models.

- We design a learning-based performance predictor to quickly and accurately evaluate the training and inference performance of searched candidates of MOP, which alleviates the tedious on-device measurement and significantly accelerates the searching of MOP.

- We also propose resolution-adaptive training (RAT) to further compress the training overhead of CNNs, which automatically schedules the image size at different training epochs, mitigating the memory occupation and accelerating the training significantly without affecting model accuracy.

- We seamlessly integrate MOP and RAT as a comprehensive training and inference co-optimization framework namely TICO, which facilitates the deployment of emerging on-device DL systems on resource-constrained embedded devices.

- We conduct extensive experiments on different datasets with various CNN models. Our TICO delivers 4.14× training acceleration, 2.32× inference acceleration, and 4.02× memory reduction than prior art.

5.2.2 Framework Overview

As shown in Figure 5.8, the proposed training and inference co-optimization framework consists of two components: 1) multi-objective model pruning (MOP) and 2) resolution-adaptive training (RAT). For a given model and dataset, our co-optimization framework will first search for the optimal model pruning strategy that can best optimize both the training efficiency and inference efficiency. To accelerate the search process, we build a set of performance estimators to quickly evaluate the candidate pruning strategies, which prevents a large number of pruned candidate models from being deployed onto the target device for evaluation, significantly mitigating the evaluation cost. Once the optimal pruning strategy is obtained, different dimensions of the given model will be pruned accordingly. Finally, our resolution-adaptive strategy will be applied to the pruned model to further optimize the training efficiency. In the following sections, we will detail the design of each component and the integration of both components.

5.2.3 Multi-Objective Model Pruning

Prior studies [52, 53, 64, 65, 189] of model pruning for CNNs mainly focus on optimizing the inference efficiency by pruning a single dimension, which poses two challenges to our goal, the co-optimization of training and inference. On the one hand, they utilize inference performance as the only objective to guide the selection and pruning of redundant computation and parameters. On the other hand, only pruning a single dimension restricts the exploration space and thus fails to comprehensively remove the redundancy for better training and inference performance. To address these issues, we propose multi-objective pruning (MOP), where we utilize multiple performance metrics relating to both training and inference as our optimization objectives and collaboratively prune all three dimensions to achieve both better training and inference performance, thereby boosting the deployment of on-device DL systems.

Optimization Metrics: To achieve the co-optimization of training and inference for embedded devices, we need to first identify the most important on-device performance metrics relating to training and inference. For training, a complete training iteration contains three computation-intensive operations: 1) forward propagation,

2)backward propagation, and 3) parameter update, which can lead to a huge time cost on embedded devices [100, 197]. Therefore, we select the training time as one of our optimization goals. In addition, the backward propagation will generate numerous intermediate activations and gradients, which can result in extremely large memory occupation during training [198]. As memory is a critical resource for various edge devices, the memory consumption of training is also identified as our optimization objective. For inference, inference latency and accuracy are two critical metrics reflecting the inference performance, thus we also include inference latency and accuracy in our optimization objectives. In summary, we carefully select four performance indicators relating to training and inference as our optimization objectives: 1) *training time* (O_{train}), 2) *memory consumption* (O_{mem}) 3) *inference latency* (O_{lat}), and 4) *prediction accuracy* (O_{acc}).

Multi-Objective Evolutionary Search: After determining the optimization metrics, we propose to collaboratively prune the three dimensions (depth, width, and resolution) to simultaneously optimize all metrics. Previous studies [176, 190, 199] have pointed out that, even with the same computational complexity (i.e., FLOPs), different network architectures can have diverse on-device inference latency. Similarly, in our context, pruning different dimensions to the same computational complexity can also lead to very distinct training and inference performance. For example, shrinking the image size (resolution) will remarkably reduce the activations, which can effectively alleviate the pressure on memory and accelerate the training of CNNs [66, 189]. While pruning layers (depth) can shorten the forward path, optimizing the inference latency [64]. To achieve the best trade-off between training and inference under the given complexity constraint, we need to carefully allocate the pruning ratio of each dimension. As such, the core of our multi-objective pruning can be interpreted as the search for the optimal pruning ratio of each dimension under a given complexity constraint to maximize the performance indicators. Let $\{c_d, c_w, c_r\}$ be the pruning ratio of the depth, width, and resolution dimension, respectively, and the pruned model can be represented as follows:

$$N' = N(c_d, c_w, c_r) = \bigodot_{i=1}^{d \cdot c_d} F_i(X_{<H_i \cdot c_r, W_i \cdot c_r, C_i \cdot c_w>}) \tag{5.9}$$

Algorithm 3: Evolutionary Search for Pruning Ratios

Data: overparameterized CNN N, FLOPs constraint f, training dataset D_t, validation dataset D_v, population size P, number of iterations Q
Result: optimal pruning ratios $\{c_d^*,\ c_w^*,\ c_r^*\}$

```
P_0 ← Initialize_Population(f, P);
p_m ← 0.3; /* Set the mutation probability */
p_c ← 0.9; /* Set the crossover probability */
for i ← 1 to Q do
    /* Crossover with probability p_c */
    P_i ← Crossover(P_{i-1}, p_c);
    /* Mutate with probability p_m */
    P_i ← Mutate(P_i, p_m);
    /* Evaluate the optimization objectives */
    R ← Evaluate(P_i, D_t, D_v);
    /* Select with elitist preservation */
    P_i ← Select_Best(P_{i-1} ∪ P_i, R, P)
/* Multi-criteria decision making */
c_d^*, c_w^*, c_r^* ← Decision_Making(P_Q);
return {c_d^*, c_w^*, c_r^*};
```

where N' represents the pruned models, which is obtained by pruning the original model N with the pruning ratios $\{c_d, c_w, c_r\}$. The optimization objective of our MOP is then formulated as follows:

$$\begin{aligned} \min_{c_d, c_w, c_r} \quad & O_{mem}(N'),\ O_{train}(N'),\ O_{lat}(N'),\ \Delta O_{acc}(N') \\ s.t. \quad & \Delta O_{acc}(N') = O_{acc}(N) - O_{acc}(N') \\ & FLOPs(N') \leq f \\ & c_d,\ c_w,\ c_r \in (0, 1] \end{aligned} \tag{5.10}$$

where ΔACC is the accuracy drop of the pruned model compared to the original model, and f is the complexity constraint of the compressed model. Here we use model FLOPs instead of parameters to quantify the complexity of CNNs as pruning the image size will not change model parameters, and thus it cannot accurately reflect the change of model complexity caused by pruning the three dimensions.

As described in Equation 5.10, given a model complexity constraint, we strive to search for the optimal pruning ratios of the three dimensions to minimize the memory occupation, training time, inference latency, and accuracy drop of the pruned

model. As such, we successfully formulate the allocation of pruning ratios as a multi-objective optimization problem, where the variables are the pruning ratios of the three dimensions, the optimization objectives are the four performance indicators relating to training and inference, and the main constraint is the FLOPs of the pruned model. Subsequently, we introduce an evolutionary algorithm (EA) to efficiently solve this optimization problem. The EA is demonstrated in Algorithm 3, where we set the population size to 20 and the number of iterations to 20 for a good balance between the cost of the algorithm and the performance of obtained solutions. During each search iteration, we first apply crossover and mutation with preset probabilities to the population from the previous iteration (i.e., the parent population) to generate 20 offsprings (i.e., the offspring population) that satisfy the FLOPs constraint. All offsprings are then evaluated with the four performance metrics (i.e., memory occupation, training time, inference latency, and accuracy). According to the evaluation results, we select the best 20 individuals with elitist preservation mechanism [178]. More specifically, the parent population and the offspring population will be combined as a larger pool, where the top 20 individuals are selected to form the new population of the current iteration. After all iterations are executed, we will obtain a set of non-dominated solutions. To determine a single optimal solution, we exploit a simple yet effective linear weighted sum method to transform the multiple objectives into an overall performance metric:

$$\begin{aligned} O_{overall} = k_{mem} \cdot O_{mem} + k_{train} \cdot O_{train} + k_{lat} \cdot O_{lat} \\ + k_{acc} \cdot O_{acc} \end{aligned} \tag{5.11}$$

where the weight of each optimization objective can be adjusted flexibly to accommodate different design considerations. In this section, we simply consider all objectives equally important and set all weights to the same value. It is worth noting that the four metrics (i.e., $O_{mem}, O_{train}, O_{lat}, O_{acc}$) have different units and ranges. Therefore, we normalize all of these metrics to the same range so that the assigned weight for each metric can effectively control the importance of each metric.

Learning-Based Performance Estimator: In Algorithm 3, each individual in the population represents a searched pruning configuration for the three dimensions, which corresponds to a pruned model. Accurately evaluating all pruned

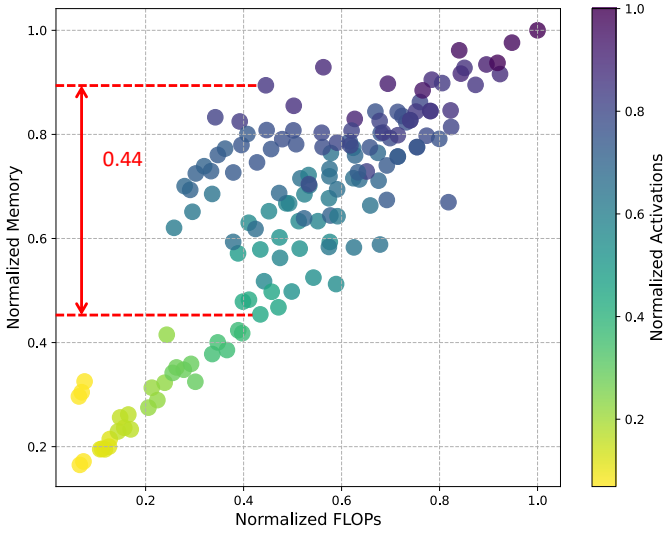


FIGURE 5.9: Distribution of memory consumption along model FLOPs and activations. Two models with similar model FLOPs can have a 0.44 difference in normalized memory occupation. In addition, memory consumption is also correlated to activations. More activations mean larger memory consumption.

models in each iteration is of great significance to the performance of the final solution. However, there will be more than 400 models generated during the entire search process, and it will incur a huge cost to actually deploy all models onto the target hardware for measurement. Particularly, the evaluation of accuracy requires a lot of tedious training work, which degrades the efficiency of the proposed EA significantly. To mitigate the evaluation cost of models, prior studies, on the one hand, train models with fewer epochs to reduce the training overhead [31]. However, this approach only optimizes the evaluation cost of accuracy, which still has to deploy all models onto the hardware to record other performance metrics like memory consumption. In addition, it needs to train models during the search process, which can affect the search efficiency significantly. On the other hand, some approaches exploit DL-based predictors (e.g., Multilayer Perceptron (MLP)) [36, 190, 200] to quickly estimate the performance of models, which eliminates the prohibitive training and on-device measurement costs during search, improving the search efficiency significantly. However, as pointed out by [201], a large number of training samples are needed to avoid the possible overfitting of these DL-based predictors, which inevitably increases the training cost of predictors.

To overcome the shortages of existing DL-based predictors, we propose a novel learning-based predictor to predict the multiple performance metrics of models. Instead of using a DL-based architecture like MLP, we utilize a simple yet effective

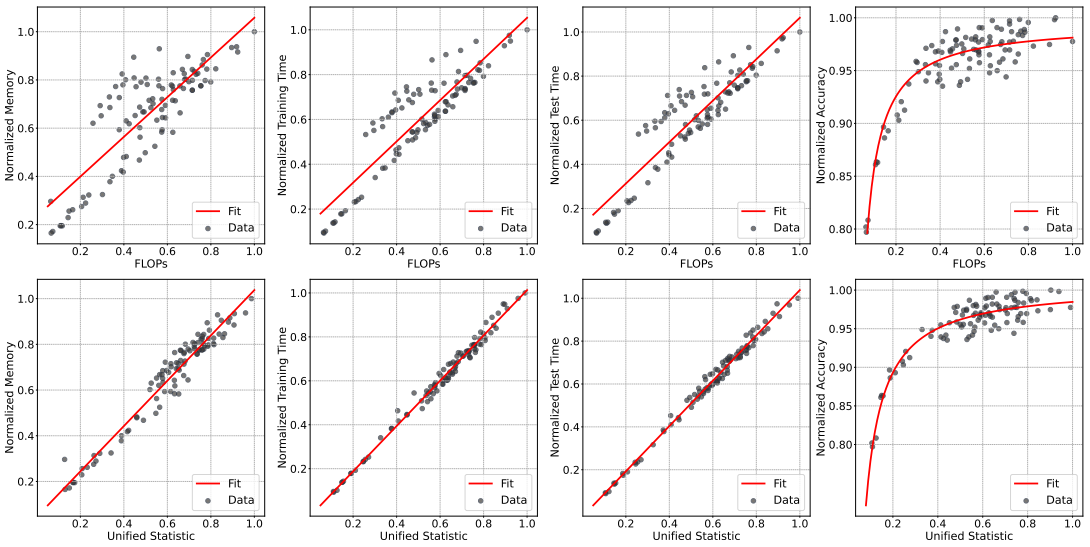


FIGURE 5.10: Distribution of the optimization metrics along FLOPs and the unified statistic, respectively. The models are from the training set.

polynomial model as the base architecture of our performance predictors, which is formulated as follows:

$$G(N) = \sum_{i=0}^{n} \theta_i \cdot S(N)^i, \quad s.t.\ \theta_i \in \Theta \tag{5.12}$$

where n is the degree of the polynomial predictor, which can be changed flexibly to better fit the data for different performance metrics. Θ is the coefficient vector of the predictor. $S(N)$ represents the model statistics of N that can be directly derived from the model architecture. Previous works have tried different model statistics to estimate the performance. For instance, [189] uses FLOPs to predict model accuracy, while [46] argues that the inference latency is more correlated to the number of intermediate activations. However, our empirical experiments in Figure 5.9 emphasize that a performance metric can be correlated to multiple statistics and using a single statistic may result in an inaccurate estimation. In addition, manually selecting the most appropriate statistic for each performance metric requires considerable human labor in our context. To address these concerns, we propose to integrate multiple model statistics into a unified statistic to predict different performance metrics. The unified statistic is represented as follows:

$$S(N) = \lambda_f \cdot Z_f(N) + \lambda_p \cdot Z_p(N) + \lambda_a \cdot Z_a(N) \tag{5.13}$$

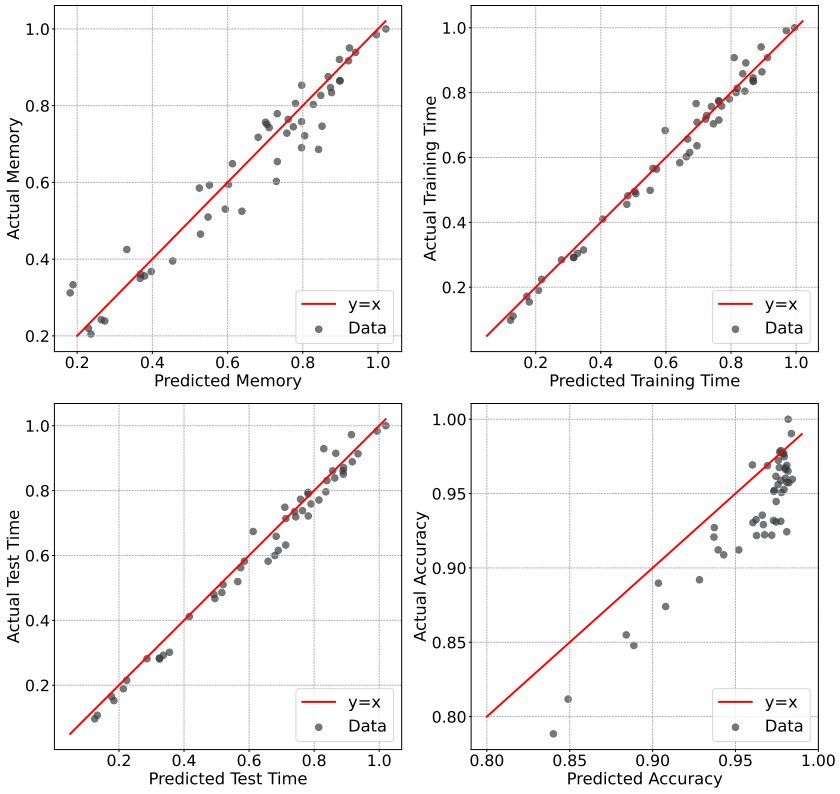


FIGURE 5.11: The prediction accuracy of our our performance predictors on the sampled validation data. We observe that the predicted results are very close to the actual performance.

where Z_f, Z_p, and Z_a are the normalized model FLOPs, parameters, and activations respectively. λ_f, λ_p, and λ_a are their coefficients, which are are learned from the sampled data. The learning equation of the coefficients is formulated as:

$$\begin{aligned} \min_{\lambda_f,\lambda_p,\lambda_a} \quad & \sum_{N_i \in \Omega} (G(N_i) - A(N_i))^2 \\ s.t. \quad & \lambda_f + \lambda_p + \lambda_a = 1 \\ & \lambda_f, \lambda_p, \lambda_a \in [0, 1] \end{aligned} \tag{5.14}$$

where $G(N_i)$ and $A(N_i)$ are the predicted performance and the actually measured performance of model N_i, respectively. Ω denotes the collection of sampled models for training. It is worth noting that the predictor G and the unified statistic S are different across performance metrics. We determine the optimal λ_f, λ_p, and λ_a for each performance metric via random search. After that, we use the least-square method to find the optimal coefficient vector (i.e. Θ) and fit the corresponding

predictor for each performance metric. Specifically, we sampled a total of 150 models in the design space, among which 100 models are used for training and the remaining 50 models are for validation. We then deploy all models onto the target hardware to measure their actual performance. The performance of training models along FLOPs and the unified statistic are summarized in Figure 5.10, which reveals that the performance of models along the unified statistic is much more predictive than along a single statistic (e.g., FLOPs). Finally, we validate the performance of our predictors on the validation data. The results are demonstrated in Figure 5.11, where we observe that our performance predictors can efficiently and accurately estimate the multiple performance metrics of models.

5.2.4 Resolution-Adaptive Training

Guided by the multiple performance metrics, MOP effectively prunes the redundancy of CNNs and obtains better training and inference performance. However, we observe that the training costs, such as memory occupation and the latency for a training iteration, are still dominant and unaffordable for some resource-constrained embedded devices. One of the main reasons for the vast training costs is the large size of images used for training. More specifically, the current practice uses the same image size for training and inference, for instance, most CNNs use a size of 224×224 for both training and inference images in ImageNet [1], which will generate a large number of activations and gradients during the backward propagation, leading to huge memory consumption and slow training. To optimize the training overhead, a simple practice is to reduce the image size for training. For example, [195] uses a fixed smaller image size during training time for higher training efficiency. However, [196] argues that different training epoch has distinct sensitivity to image size, and using a fixed image size throughout the training may fail to achieve the optimal trade-off between training efficiency and accuracy. Generally, early epochs are less sensitive to image size than late epochs, and thus a smaller image size can be used for training at early epochs. To this end, [177, 196] propose to split the training process into multiple stages and dynamically change the image size to further compress the training overhead. However, the manual tune of some important hyperparameters, such as the number of stages and the image size for each stage, requires considerable human labor. In addition, the training strategies

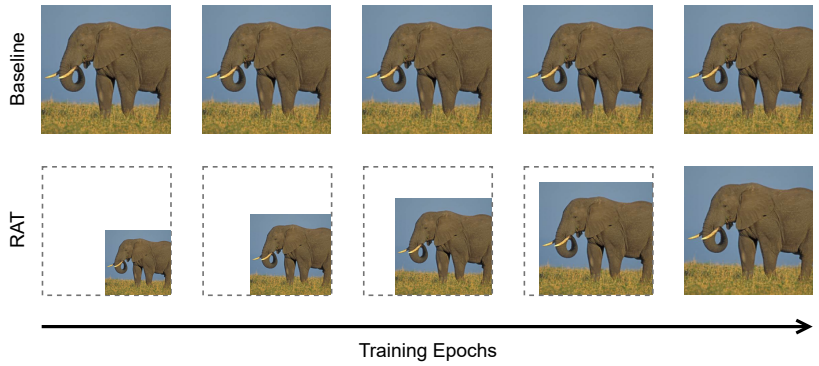


FIGURE 5.12: Comparison between RAT and the traditional training strategy which training models with the same image size (i.e., 224×224 for ImageNet) throughout the entire training process.

in [177, 196] are mainly proposed for large models and images, which may cause a significant accuracy drop in our context.

To address the above concerns, we propose resolution-adaptive training (RAT) to further optimize the training overhead of pruned models from MOP. The overview of RAT is demonstrated in Figure 5.12, where we start with a very small image size, and gradually increase the image size as training epochs. Instead of splitting the training process into multiple stages and determining the image size of each stage manually, we design a cosine-like scheduling strategy to automatically grow the image size for each training epoch. The scheduling of image size is formulated as follows:

$$r_i = \left\lceil \frac{1}{2}\left(\cos\left(\left(\frac{i}{n}+1\right)\cdot\pi\right)+1\right)\cdot(r_n - r_0) + r_0 \right\rceil \tag{5.15}$$

where $i \in [1, n-1]$ represents the index of the current training epoch, r_i represents the image size for the i-th epoch, and n is the total number of epochs. Besides, r_0 and r_n are the image sizes for the first and the last training epoch respectively. In our scenario, r_n is equal to the inference image size, which is determined once the model is pruned by our MOP, while r_0 is the only tunable hyperparameter. Once r_0 is determined, the image size for each training epoch can be directly derived by Equation 5.15, which greatly mitigates the human labor compared to manually tuning the image size of each epoch [177]. The optimal r_0 is highly correlated to the inference image size [195]. Therefore, the most intuitive approach to determine r_0 is to directly search for the optimal r_0 for the given inference image size. However, for different resource constraints, MOP will generate models with different inference image sizes and exhaustively searching for the optimal r_0 for each inference image

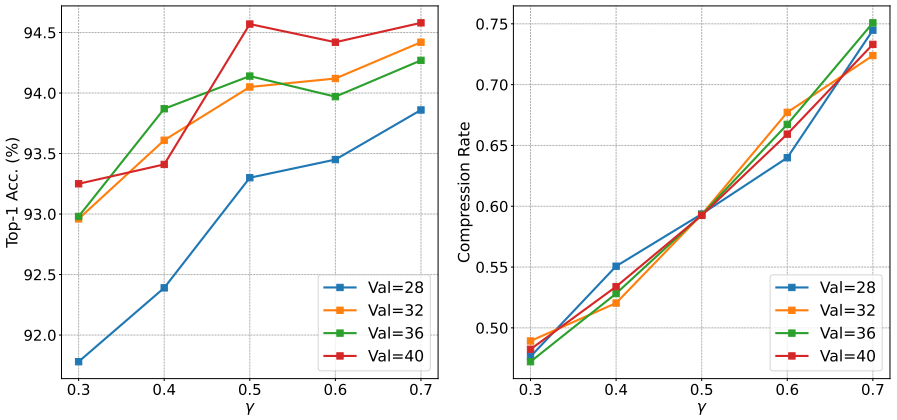


FIGURE 5.13: Comparisons between different beginning resolutions.

size will be computationally prohibitive. Instead, we propose to connect r_0 and the inference image size using a coefficient, and then search for the optimal coefficient to strike a balance between accuracy and training efficiency while eliminating the repetitive searching work. Specifically, the connection between r_0 and the inference image size is represented as follows:

$$r_0 = \gamma \cdot r \tag{5.16}$$

where r is the inference image size and γ is the coefficient. To find the optimal value for γ, we conduct empirical experiments with multiple inference image sizes. The experimental results are demonstrated in Figure 5.13

Integration of MOP and RAT: Given a baseline CNN model, MOP first compresses the three dimensions of the model to co-optimize its training and inference efficiency. Afterward, according to the inference resolution obtained by MOP, RAT efficiently calculates the optimal training resolution of each training stage. With the seamless combination of MOP and RAT, the training efficiency and inference efficiency of the model are significantly improved.

5.2.5 Experiments

In this section, we conduct extensive experiments on different hardware platforms to validate the advantages of TICO over SOTA approaches in terms of training and inference efficiency. In addition, we perform comprehensive ablation studies

TABLE 5.4: Specifications of hardware platforms with different capabilities used in our experiments. Jetson Nano and Jetson Xavier are representative standalone edge GPU platforms, and RTX3090 is used with an AMD 3990X CPU.

Device	Power	CPU Freq.	GPU Freq.	CUDA Cores	Memory
Jetson Nano	10 W	1479 MHz	922 MHz	128	4 GB
Jetson TX2	15 W	2048 GHz	1300 MHz	256	8 GB
Jetson Xavier	30 W	1780 MHz	900 MHz	512	16 GB
RTX3090	350 W	2200 MHz	1433 MHz	10496	24 GB

to better demonstrate the contribution of each component in our framework. The framework is implemented in PyTorch.

Hardware Platforms: To validate the actual training and inference efficiency on resource-constrained devices, we employ multiple embedded hardware platforms to deploy our framework. We select three embedded GPU platforms with different capabilities: 1) Jetson Nano, 2) Jetson TX2, and 3) Jetson Xavier. Moreover, we also employ CPUs and desktop GPUs to demonstrate the efficacy of our approach. The details of utilized hardware platforms are listed in Table 5.4.

Neural Networks: To demonstrate the efficacy of our approach for different CNNs, we select multiple popular CNN models as the baseline model for experiments. Specifically, both block-based and block-free CNN models are involved. Block-based neural networks include MobileNetV2 [29], ResNet50 [15], ResNet110 [15], RegNet-X [31], and GoogLeNet [202]. Block-free neural networks include AlexNet [3] and VGG11-BN [42]

Settings: On CIFAR-10, we use SGD with a momentum of 0.9 as the optimizer to train all models for 200 epochs. The weight decay is set to 5e-4 and the batch size of training is 128. The initial learning rate is 0.1 and decayed by cosine annealing policy. The training resolution of each epoch is automatically scheduled with RAT. On ImageNet, we also use an SGD optimizer with a momentum of 0.9 to train models for 120 epochs. The first 5 epochs are for warmup. The training batch size is 1024 and the initial learning rate is 1.6, which is decayed by cosine annealing. We also use RAT to automatically schedule the training resolution for each epoch. To prevent overfitting, we also use label smoothing with $\epsilon = 0.1$.

Results on CIFAR-10: In this subsection, we demonstrate the experimental results of our approach on the CIFAR-10 dataset and compare it with other methods.

Comparison with Different Baselines: We first apply our approach to various popular CNNs to demonstrate the efficacy of our approach in optimizing the training

TABLE 5.5: Compression results of our approach for various popular architectures. The dataset is CIFAR-10.

Model	Method	FLOPs		Params	Top-1%
		train	test		
AlexNet	Baseline	500.76M	166.92M	2.25M	89.50
	TICO (ours)	175.47M	70.50M	0.95M	89.11
VGG11	Baseline	459.09M	153.03M	9.49M	92.69
	TICO (ours)	115.52M	65.19M	4.01M	91.49
VGG16	Baseline	940.41M	313.47M	14.99M	94.02
	TICO (ours)	266.03M	150.12M	6.78M	93.25
MobileNetV1	Baseline	466.08M	155.36M	3.22M	92.35
	TICO (ours)	116.71M	65.86M	1.44M	91.47
MobileNetV2	Baseline	229.77M	76.59M	2.24M	92.28
	TICO (ours)	63.87M	36.04M	1.09M	91.17
ResNet110	Baseline	758.67M	252.89M	1.73M	94.01
	TICO (ours)	159.08M	89.77M	0.59M	93.99
DenseNet40	Baseline	848.76M	282.92M	1.04M	94.65
	TICO (ours)	249.65M	140.88M	0.51M	93.53

and inference computation. As shown in Table 5.5, both the training and inference computation of different CNNs are significantly reduced with only a mere accuracy drop. Specifically, we observe 79.0% reduction in training FLOPs and 64.5% reduction in inference FLOPs for ResNet110, while the corresponding accuracy drop is only 0.02%. It is worth noting that the compression ratio of training FLOPs is higher than that of inference FLOPs, which is because we first utilize our MOP to simultaneously compress training and inference FLOPs, then we exploit RAT to further reduce training FLOPs. By this means, the computational gap between training and inference is remarkably narrowed and the overhead of the whole deep learning system is significantly optimized.

Comparison with Prior Art: In addition to the comparison with baseline models, we also compare our approach with many SOTA CNN compression algorithms. The experiments are conducted on CIFAR-10 with three widely utilized CNN architectures: 1) ResNet110, 2) VGG16 with batch normalization layers, and 3) DenseNet40. It is worth noting that we do not compare the actual memory consumption and inference latency in some experiments (e.g., Table 5.6), which is because some of the compared approaches do not provide their codes and models for such measurement. Therefore, to fairly compare as many methods as possible, we only list the results reported in the corresponding papers for comparison. For those methods that provided their codes, we have made the deployment to measure their actual memory consumption and inference latency.

The compression results of different methods on ResNet110 are compared in Table

TABLE 5.6: Compression results of different approaches on ResNet110. The dataset is CIFAR-10. "CR" denotes the compression ratio.

Method	FLOPs (CR)		Top-1%
	train	test	
ResNet110	758.67M (0.0%)	252.89M (0.0%)	94.01
L1 [203]	465.00M (38.7%)	155.00M (38.7%)	93.30
GAL-0.5 [184]	390.60M (48.5%)	130.20M (48.5%)	92.55
Slimable [61]	306.96M (59.5%)	102.32M (59.5%)	92.56
DECORE-500 [185]	489.90M (35.4%)	163.30M (35.4%)	93.88
HRank [53]	317.10M (58.2%)	105.70M (58.2%)	93.36
DECORE-300 [185]	289.98M (61.8%)	96.66M (61.8%)	93.50
TICO (ours)	**159.08M (79.0%)**	**89.77M (64.5%)**	**93.99**

TABLE 5.7: Compression results of different approaches on VGG16. The dataset is CIFAR-10. "CR" denotes the compression ratio.

Method	FLOPs (CR)		Top-1%
	train	test	
VGG16	940.41M (0.0%)	313.47M (0.0%)	94.02
SSS [126]	549.39M (41.6%)	183.13M (41.6%)	93.02
Zhao et al [204]	570.00M (39.4%)	190.00M (39.4%)	93.18
Slimable [61]	451.44M (52.0%)	150.48M (52.0%)	93.36
GAL-0.05 [184]	568.47M (39.6%)	189.49M (39.6%)	92.03
HRank [53]	436.83M (53.5%)	**145.61M (53.5%)**	**93.43**
GAL-0.1 [184]	515.67M (45.2%)	171.89M (45.2%)	90.73
TICO (ours)	**266.03M (71.7%)**	150.12M (52.1%)	93.25

5.6, where we observe that our TICO outperforms all competitors in terms of training FLOPs, inference FLOPs, and accuracy. Specifically, compared to DECORE-500 [185], our method achieves 43.6% higher compression ratio in training FLOPs and 29.1% higher compression ratio in inference FLOPs. Meanwhile, the accuracy of our approach is even 0.11% higher than DECORE-500. Compared to HRank [53], we achieve a similar compression ratio (64.5% v.s. 58.2%) in inference FLOPs, but our TICO observes remarkable improvements in training FLOPs (79.0% v.s. 58.2%) and accuracy (93.99% v.s. 93.36%).

Table 5.7 demonstrates the compression results of different methods on VGG16, from which we observe that, except HRank [53], our TICO surpasses all other methods in both training FLOPs and inference FLOPs. Specifically, TICO reduces the training FLOPs and inference FLOPs of the baseline VGG16 by 71.7% and 52.1%, respectively. For HRank [53], it achieves slightly higher accuracy (93.43% v.s. 93.25%) than our TICO with similar inference FLOPs (145.61M v.s. 150.12M). However, the training FLOPs of HRank are significantly higher (436.83M v.s. 266.03M) than that of our approach.

The results of DenseNet40 are summarized in Table 5.8, which indicates the advantages of TICO over other existing approaches. Compared to the baseline model,

TABLE 5.8: Compression results of different approaches on DenseNet40. The dataset is CIFAR-10. "CR" denotes the compression ratio.

Method	**FLOPs (CR)**		**Top-1%**
	train	**test**	
DenseNet40	848.76M (0.0%)	282.92M (0.0%)	94.65
Zhao et al. [204]	468.00M (44.9%)	156.00M (44.9%)	93.16
GAL-0.05 [184]	384.33M (54.7%)	128.11M (54.7%)	93.53
HRank [53]	377.67M (56.0%)	**125.89M (56.0%)**	93.29
TICO (ours)	**249.65M (70.6%)**	140.88M (50.2%)	**93.53**

TICO compresses training FLOPs by 70.6% and inference FLOPs by 50.2%, while Zhao et al. [204] only achieves 44.9% compression ratio in training FLOPs, while is significantly lower than our approach. Meanwhile, the accuracy of TICO is higher (93.53% v.s. 93.16%) than Zhao et al. [204]. It is worth noting that both HRank [53] and GAL-0.05 [184] achieve slightly higher compression ratios in inference FLOPs than TICO, but their accuracy and training FLOPs compression ratios are much lower than TICO.

The comparison experiments show that TICO enables better training and inference efficiency than previous approaches. On the one hand, such improvements benefit from the proposed MOP, where multiple dimensions of CNNs, including the network architecture and input data, are collaboratively pruned and thus we can achieve a higher compression ratio with a smaller accuracy drop. In contrast, previous methods mainly focus on pruning a single dimension and ignore the redundancy in other dimensions, which restricts the compression ratio and causes remarkable accuracy degradation. In addition, it is worth noting that previous approaches usually achieve the same compression ratio for training FLOPs and inference FLOPs, while TICO obtains a higher compression ratio for training FLOPs than for inference FLOPs. This phenomenon results from that previous methods only compress the model itself, which will equally reduce training FLOPs and inference FLOPs. As a comparison, in addition to compressing the model itself, TICO also introduces RAT to compress the training process, which further reduces the training cost of the compressed model, thereby minimizing the gap between training cost and inference cost, and facilitating the deployment of deep learning systems in edge environments.

On-device Performance Evaluation: To validate the efficacy of our approach in optimizing the on-device performance of CNNs, we utilize different approaches to compress multiple popular CNN models. Then, we deploy the compressed models from

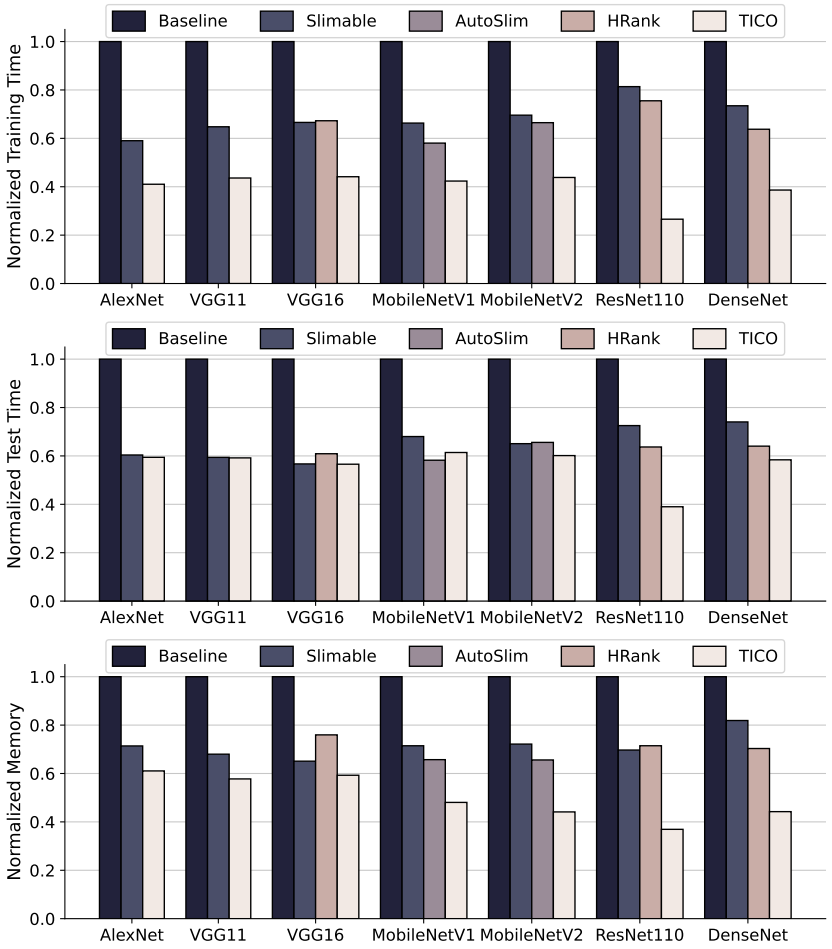


FIGURE 5.14: On-device performance of models obtained from different compression approaches. The dataset is CIFAR-10 and the performance is measured on a single RTX3090 GPU with a batch size of 1024.

different approaches onto hardware and compare their performance. The experimental results are summarized in Figure 5.14, where TICO observes the best performance in training efficiency, memory occupation, and inference efficiency among all competitors. Particularly, TICO achieves the greatest advantages over other approaches on ResNet110. TICO accelerates the training of ResNet110 by 3.76× (from 267.45ms per batch to 71.11ms per batch), which is remarkably higher than Slimable (217.64ms per batch, 1.23×) and HRank (201.99ms per batch, 1.32×). Meanwhile, TICO accelerates the inference of ResNet110 by 2.56× (from 66.27ms per batch to 25.84ms per batch) and reduces the memory occupation by 63.1% (from 10.13GB to 3.74GB). In addition, TICO also outperforms those competitors on other architectures. For example, on MobileNetV2, TICO improves the training speed by 1.59× (81.87ms v.s. 129.97ms) compared to AutoSlim [205]. At the same time, the inference speed is also improved by 1.09× (30.01ms v.s. 32.73ms),

and the memory occupation is reduced by 32.74% (7.13GB v.s. 10.6GB). This experiment shows that our approach is able to effectively optimize the training and inference overhead of CNNs on actual hardware devices, and it can achieve better performance than previous methods.

Experimental Results on ImageNet: To better demonstrate the efficacy of TICO on large-scale datasets, we perform experiments on the most widely used large-scale dataset, ImageNet [1] (as known as ILSVRC-2012 or ImageNet-2012).

Comparison with Prior Art: We first use ResNet50 [15] as the baseline network, and compare the performance of TICO with current SOTA model compression techniques. To extensively compare our method with other approaches at different FLOPs regimes, we select three different FLOPs constraints and consequently obtain three models with different FLOPs. As shown in Table 5.9, in different FLOPs regimes, our TICO achieves the best trade-off among training, inference, and accuracy. Specifically, in the highest compute regime, TICO reduces training FLOPs by 59.1% and inference FLOPs by 30.7% with a negligible drop of 0.03% in top-5 accuracy. In contrast, other approaches usually lead to significant accuracy degradation. For example, DECORE-8 [185] only reduces training FLOPs and inference FLOPs by 13.7%, while the top-1 accuracy drop is up to 0.49%. In the middle compute regime, TICO also achieves the highest compression ratio in training FLOPs (68.6%), which is 34.7% higher than HAP (33.9%). Meanwhile, the top-1 accuracy of TICO (75.77%) is 0.65% higher than HAP (75.12%). In the lowest compute regime, TICO also achieves the lowest training FLOPs and the highest accuracy. It is worth noting that TICO's inference FLOPs (1.03B) are slightly higher than HRank (0.98B), but its training FLOPs and top-1 accuracy are much better.

On-Device Performance Evaluation: To further demonstrate the advantages of TICO in optimizing the on-device efficiency of CNNs, we also conduct experiments on multiple distinct hardware devices and compare the performance of different methods. The baseline network architecture exploited in experiments is ResNet50 and the dataset is ImageNet. The results of different approaches are compared in Figure 5.15, where we observe that, on all devices, TICO surpasses those competitors by a large margin. Specifically, TICO achieves 4.55× speedup (from 480ms per batch to 106.38ms per batch) in training time on RTX3090 compared to the baseline, while HRank only accelerates the training by 1.61× (from 480ms per batch

TABLE 5.9: Compression results of different approaches on ResNet50. The dataset is ImageNet. "CR" denotes the compression ratio.

Method	FLOPs (CR)		Top-1%	Top-5%
	train	test		
ResNet50	12.30B (0.0%)	4.10B (0.0%)	76.80	93.38
SSS [126]	8.46B (31.2%)	**2.82B (31.2%)**	74.18	91.91
AutoPruner [206]	11.28B (8.3%)	3.76B (8.3%)	74.76	92.15
Taylor [52]	7.98B (35.1%)	2.66B (35.1%)	75.50	-
DECORE-8 [185]	10.62 (13.7%)	3.54B (13.7%)	76.31	93.02
TICO (ours)	**5.03 (59.1%)**	2.84B (30.7%)	**76.62**	**93.35**
GAL-0.5 [184]	6.99B (43.2%)	2.33B (43.2%)	71.95	90.94
AutoPruner [206]	7.92B (35.6%)	2.64B (35.6%)	73.05	91.25
Taylor [52]	6.75B (45.1%)	2.25B (45.1%)	74.50	-
DECORE-6 [185]	7.08B (42.4%)	2.36B (42.4%)	74.58	92.18
HRank [53]	6.90B (43.9%)	2.30B (43.9%)	74.98	92.33
SRR-GR [207]	6.78B (44.9%)	2.26B (44.9%)	75.11	92.35
HAP [186]	8.13B (33.9%)	2.71B (33.9%)	75.12	-
Adapt-DCP [124]	5.85B (52.4%)	**1.95B (52.4%)**	75.15	92.30
TICO (ours)	**3.86B (68.6%)**	2.18B (46.8%)	**75.77**	**92.85**
HRank [53]	2.94B (76.1%)	**0.98B (76.1%)**	69.10	89.58
DECORE-4 [185]	3.57B (71.0%)	1.19B (71.0%)	69.71	89.37
GAL-1 [184]	4.74B (61.5%)	1.58B (61.5%)	69.82	89.75
HAP [186]	4.05B (67.1%)	1.35B (67.1%)	71.18	-
Tylor [52]	4.02B (67.3%)	1.34B (67.3%)	71.69	-
HRank [53]	4.65B (62.2%)	1.55B (62.2%)	71.98	91.01
DECORE-5 [185]	4.80B (61.0%)	1.60B (61.0%)	72.06	90.82
TICO (ours)	**1.83B (85.1%)**	1.03B (74.9%)	**73.01**	**91.20**

to 313.51ms per batch). Meanwhile, the memory reduction on RTX3090 achieved by TICO is 71.6% (from 23.07GB to 6.55GB), which is also much higher than the 12.1% memory reduction achieved by HRank (from 23.07GB to 20.29GB). On other devices, TICO also achieves considerable advantages. For example, TICO reduces the inference latency (i.e., test time in the figure) on TX2 to 227.85ms per batch, which is 70.3% lower than the baseline (766.81ms) and 50.5% lower than GAL-0.5 (459.86ms). The experiments validate the efficiency of TICO on various hardware devices.

Ablation Study: The proposed TICO framework mainly consists of two core components: 1) MOP and 2) RAT. In this subsection, we perform comprehensive ablation experiments to separately evaluate the efficacy of each component.

Validation of MOP: To better demonstrate the advantages of MOP, we first conduct experiments on multiple distinct hardware platforms to compare the on-device efficiency of MOP with single-dimensional pruning. The experimental results are visualized in Figure 5.16, where we observe that, on all devices, MOP surpasses single-dimensional pruning approaches by a wide margin in both training and inference performance. Meanwhile, we also apply MOP and other advanced model pruning approaches to multiple representative CNN architectures and compare

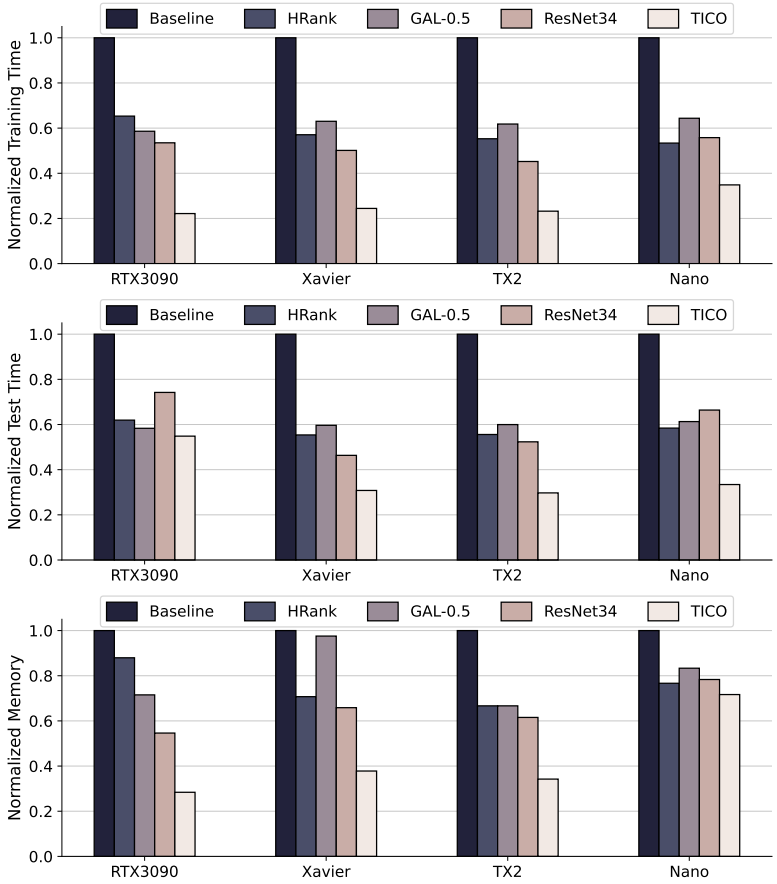


FIGURE 5.15: On device performance of different compression approaches on multiple hardware platforms. The dataset is ImageNet and the baseline model is ResNet50.

TABLE 5.10: Comparison between MOP and other advanced model pruning methods. The dataset is CIFAR-10.

Model	Method	FLOPs		Top-1%
		train	test	
MobileNetV1	Baseline	466.08M	155.36M	92.35
	Slimable [61]	202.44M	67.48M	92.26
	MOP (ours)	**197.58M**	**65.86M**	**92.28**
ResNet110	Baseline	758.67M	252.89M	94.01
	Slimable [61]	306.96M	102.32M	92.56
	HRank [53]	**265.32M**	**88.44M**	93.14
	DECORE-500 [185]	489.90M	163.30M	93.88
	MOP (ours)	269.31M	89.77M	**93.91**
DenseNet40	Baseline	848.76M	282.92M	94.65
	Zhao et al. [204]	468.00M	156.00M	93.16
	HRank [53]	**377.67M**	**125.89M**	93.29
	MOP (ours)	422.64M	140.88M	**93.52**

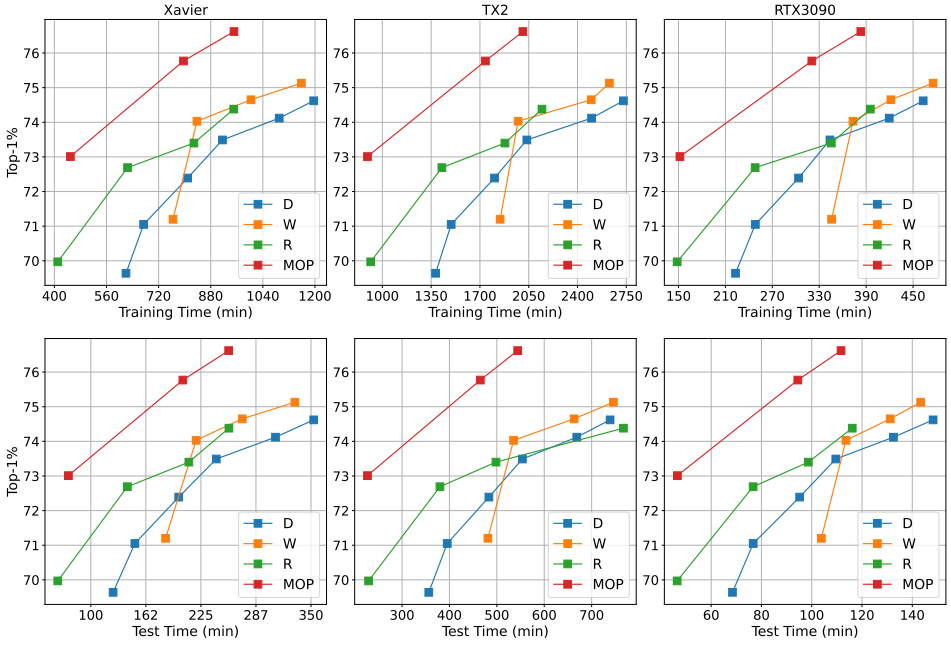


FIGURE 5.16: Comparison of training and inference efficiency on different devices between MOP and single-dimensional pruning approaches. D, W, and R represent depth pruning, width pruning, and resolution pruning, respectively. The baseline model is ResNet50 and the experimental dataset is ImageNet.

their results in Table 5.10. From the results, we can see that MOP is able to achieve better accuracy and higher compression ratios in training and inference computation than other approaches. For instance, compared to HRank on ResNet110, MOP achieves 0.77% higher top-1 accuracy with similar training FLOPs and inference FLOPs. The experiments validate that MOP can effectively compress the multiple dimensions of CNN models for better accuracy and efficiency than existing approaches.

Validation of RAT: To validate the proposed RAT module, we first compare the on-device training efficiency of RAT with various progressive training strategies. We select AlexNet, ResNet110, and DenseNet40 as the experimental models and train them for 200 epochs with different approaches. The training curves of different approaches are visualized in Figure 5.17. We find that, compared to the competitors, our RAT module improves the final accuracy significantly without sacrificing training efficiency. Moreover, we also summarize the training FLOPs, training time, and memory usage of different approaches in Table 5.11. From the results, we observe that RAT achieves the highest top-1 accuracy for all CNN architectures among all approaches, while it does not compromise the training overhead obviously. Specifically, for AlexNet, RAT reduces the training FLOPs by 40.7%, the

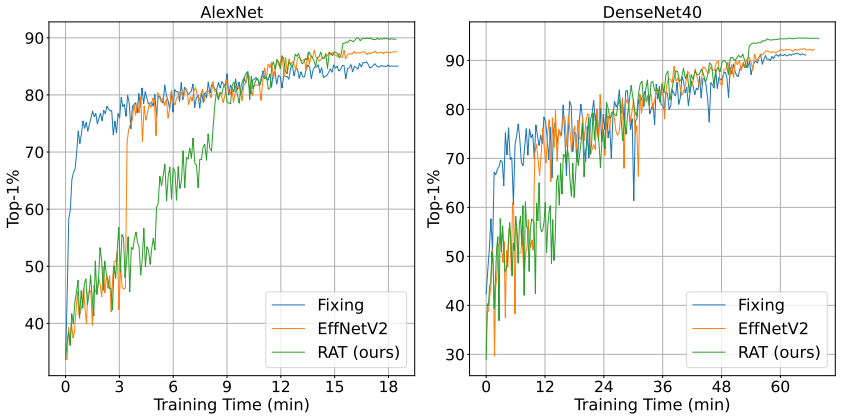


FIGURE 5.17: Training curves of different progressive training strategies on a RTX3090 GPU. The dataset is CIFAR-10.

TABLE 5.11: Comparison of training efficiency and memory usage between RAT and other progressive training strategies. The experimental dataset is CIFAR-10, and the training time is measured on a RTX3090 GPU.

Model	Method	Training		Memory	Top-1%
		FLOPs	Time		
AlexNet	Baseline	500.76M	36.36ms	6.78G	89.50
	Fixing [195]	**281.68M**	27.48ms	**5.24G**	85.47
	EffNetV2 [177]	291.07M	25.90ms	5.25G	88.16
	RAT (ours)	297.05M	**24.69ms**	5.25G	**89.74**
ResNet110	Baseline	758.67M	267.45ms	10.13G	94.01
	Fixing [195]	**426.75M**	**178.23ms**	**6.54G**	89.54
	EffNetV2 [177]	440.98M	178.43ms	6.66G	92.52
	RAT (ours)	448.15M	182.79ms	6.73G	**93.92**
DenseNet40	Baseline	848.76M	410.64ms	23.42G	94.65
	Fixing [195]	**477.43M**	**256.79ms**	16.10G	90.72
	EffNetV2 [177]	493.34M	260.80ms	16.09G	92.38
	RAT (ours)	501.36M	264.37ms	**15.08G**	**94.42**

training time by 32.1%, and the memory usage by 22.6% while improving the top-1 accuracy by 0.24% compared to the baseline training strategy. These experiments indicate that the proposed RAT strategy further improves the training efficiency and reduces the resource consumption of CNN models while not sacrificing model accuracy.

Integration of MOP and RAT: To further demonstrate the efficacy of TICO, in this subsection, we compare the training performance of TICO with its sub-components (i.e., MOP and RAT). The training curves of MOP, RAT, and TICO are visualized in Figure 5.18, where we observe that both sub-components achieve considerable improvements in training efficiency compared to the baseline, while the highest improvement is achieved by TICO. Meanwhile, for both AlexNet and ResNet110, the improvement in training efficiency does not sacrifice the final accuracy. Specifically,

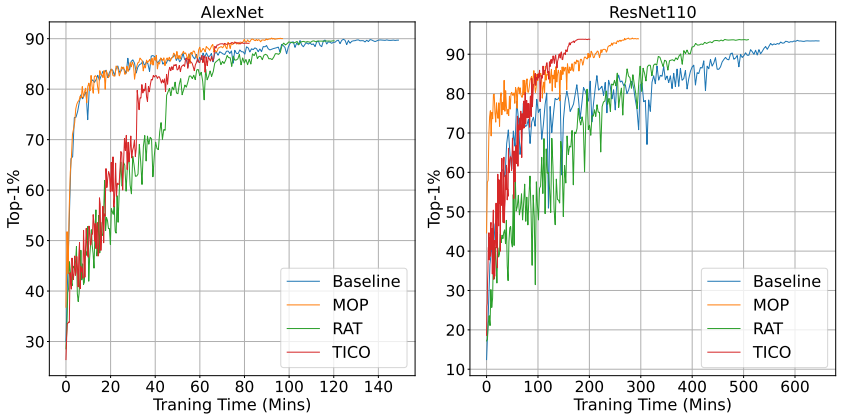


FIGURE 5.18: Training curves of MOP, RAT, and TICO. The hardware is Jetson Xavier and the dataset is CIFAR-10. All models are trained for 200 epochs.

for ResNet110, the baseline obtains 93.41% top-1 accuracy with 646.8 minutes on Xavier. As a comparison, TICO only takes 200.7 minutes to achieve 93.8% accuracy, which is 3.2× faster in training speed and 0.39% higher in accuracy than the baseline.

Analytical Experiments: In this subsection, we conduct experiments to analyze the impact of some important hyperparameters on the performance of our framework.

Complexity of Evolutionary Algorithm: As mentioned in Section 5.2.3, we propose to search for the optimal pruning decision for the three dimensions of CNNs with an evolutionary algorithm, where the population size and the number of generations are two critical hyperparameters that can affect the performance and complexity of our framework. To determine the optimal value for the two hyperparameters, we conduct experiments with different population sizes and generations. The results are shown in Figure 5.19, where we can find that, for all population sizes, the performance fluctuates significantly in early generations. When the evolutionary algorithm runs beyond 35 generations, the performance metrics begin to converge, and keep increasing the number of generations does not improve the performance obviously. Therefore, in our framework, we set the number of generations to 35 for the evolutionary algorithm. As for the population size, we can see from Figure 5.13 that, with the number of generations fixed to 35, the best memory usage, training time, and inference time are achieved when the population size equals 40. Hence, we set the population size to 40 in our experiments.

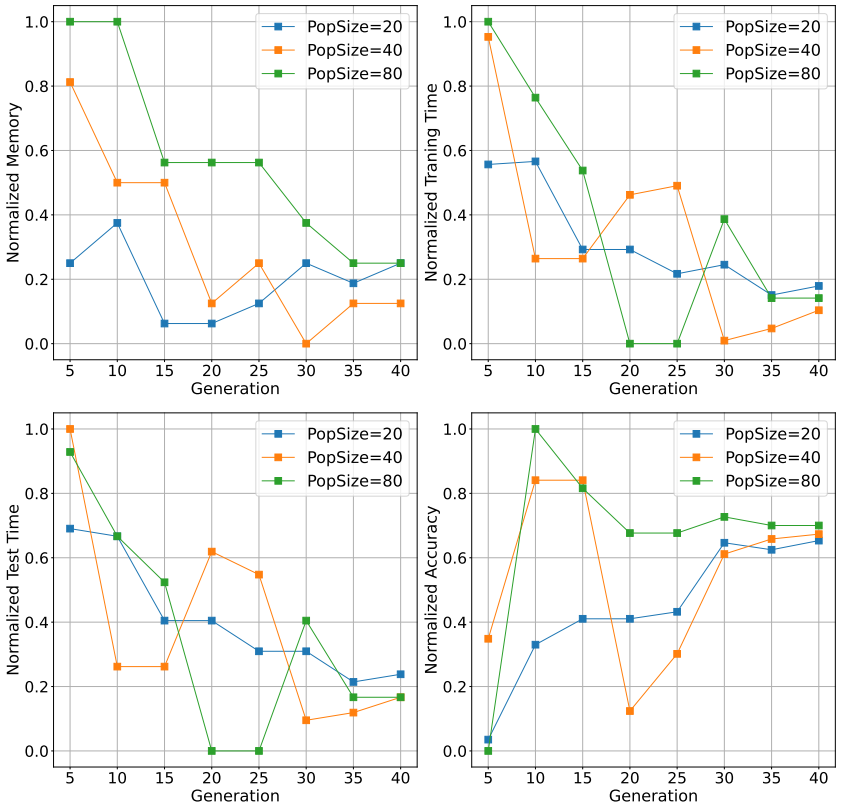


FIGURE 5.19: Impacts of population size and generations on the optimization performance of our framework. For memory usage, training time, and test time, the lower the better. For accuracy, the higher the better.

Visualization of Compressed Models: To better demonstrate the advantages of TICO over existing model compression approaches, we visualize the compressed models from different approaches in Figure 5.20. We observe that previous methods usually focus on compressing a single dimension, which may cause a significant drop in accuracy when pursuing a high compression ratio. In contrast, TICO is committed to comprehensively compressing the redundancy of multiple dimensions to achieve a good balance between these dimensions, thereby improving the compression ratio without affecting accuracy. In addition, it is worth noting that both GAL and HRank do not compress input images, while we find that there are also considerable redundant pixels in input images. Therefore, we also compress the size of input images, which further improves the efficiency of our models.

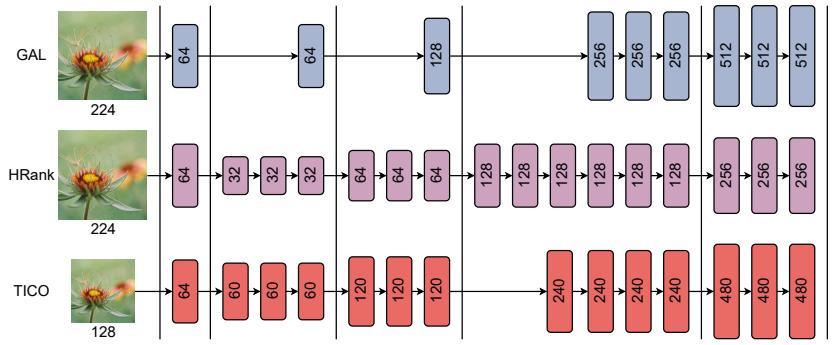


FIGURE 5.20: Visualization of compressed models from different approaches. The baseline architecture is ResNet50 and the original image size is 224×224.

5.2.6 Summary

In this section, we present TICO, a co-optimization framework to simultaneously optimize the training efficiency and inference efficiency of CNNs for embedded devices. First, we propose multi-objective model compression (MOP) to jointly compress the three dimensions of CNNs for better training and inference efficiency. We formulate joint compression as a multi-objective optimization problem and efficiently find the optimal compression strategy that can achieve the best trade-off between training efficiency and inference efficiency. Moreover, we also propose resolution-adaptive training (RAT), which splits the training into multiple stages, and adaptively schedules the training resolution for different stages. In this way, RAT further improves the training efficiency of the compressed model without compromising accuracy. Finally, MOP and RAT are integrated to work coordinately for the best trade-off between training efficiency and inference efficiency. Extensive experiments validate the efficacy and efficiency of our approach.

Chapter 6

Run-Time Model Optimization via Dynamic Inference

CNNs have demonstrated encouraging results in image classification tasks. However, the prohibitive computational cost of CNNs hinders the deployment of CNNs onto resource-constrained embedded devices. To address this issue, we propose EdgeCompress[1], a comprehensive compression framework to reduce the computational overhead of CNNs. In EdgeCompress, we first introduce dynamic image cropping, where we design a lightweight foreground predictor to accurately crop the most informative foreground object of input images for inference, which avoids redundant computation on background regions. Subsequently, we present compound shrinking to collaboratively compress the three dimensions (depth, width, and resolution) of CNNs according to their contribution to accuracy and model computation. Dynamic image cropping and compound shrinking together constitute a multi-dimensional CNN compression framework, which is able to comprehensively reduce the computational redundancy in both input images and neural network architectures, thereby improving the inference efficiency of CNNs. Further, we present a dynamic inference framework to efficiently process input images with different recognition difficulties, where we cascade multiple models with different complexities from our compression framework and dynamically adopt different models for different input images, which further compresses the computational redundancy and improves the inference efficiency of CNNs, facilitating the deployment of advanced CNNs onto embedded hardware. Experiments on ImageNet-1K

[1]The work in this chapter has been published in [208]

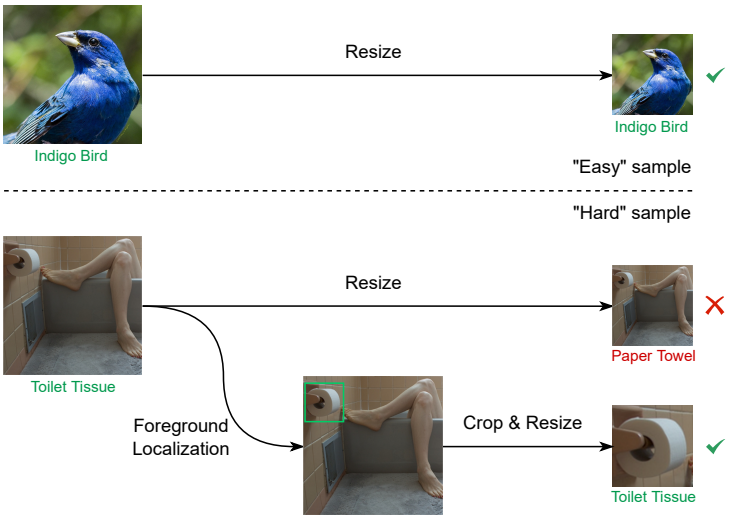


FIGURE 6.1: Predictions from ResNet50. For easy samples, the network can still generate correct predictions at a smaller resolution (e.g., 112 × 112 for ImageNet-1K). For hard samples, simply resizing images to a smaller resolution can lead to misclassification, while using dynamic cropping can correctly classify hard samples at a smaller resolution.

demonstrate that EdgeCompress reduces the computation of ResNet50 by 48.8% while improving the top-1 accuracy by 0.8%. Meanwhile, we improve the accuracy by 4.1% with similar computation compared to HRank. the SOTA compression framework.

6.1 Introduction

We introduce several methods to optimize DNN models at design time. Like many other model optimization approaches, given a resource budget, they usually yield a fixed compressed neural network and resolution for all images. However, as discussed in [65, 66], different images are of distinct recognition difficulties, using a static model and resolution to process all images can lead to inefficient utilization of computation, achieving only sub-optimal efficiency and accuracy. Practically, for images with simple features, a small CNN model is adequate to generate correct results. For complex images, a larger model with higher capability should be used to extract high-level features for a correct prediction. This inspires our last motivation: *Can we dynamically adjust the model and resolution for different images during inference to further optimize inference efficiency and accuracy*?

To address the above questions for more efficient image classification with CNNs, we, in this chapter, propose a novel inference framework, EdgeCompress, to comprehensively reduce the inference overhead of CNNs, thereby optimizing the classification efficiency of CNNs and facilitating the deployment of advanced CNNs onto edge devices. In EdgeCompress, we first propose a two-stage multi-dimensional model compression framework to coordinately compress all three dimensions of CNNs. In the first stage, we introduce a novel dynamic image cropping (DIC) strategy to accurately remove the spatial redundancy in input images, in which we design a lightweight foreground predictor to efficiently localize the most discriminative foreground of input images, then only the detected foreground will be preserved for classification and the redundant background will be discarded. As shown in Figure 6.1, through the dynamic image cropping strategy, we are capable of generating fine-cropped images with less spatial redundancy, thereby achieving satisfactory classification accuracy even at a smaller resolution. In the second stage, we present a compound shrinking (CS) strategy to jointly compress the three dimensions of CNNs, thereby further reducing the redundancy in input images and network architectures. We first quantify the impact of shrinking different dimensions on model complexity and accuracy, according to which we automatically calculate a shrinking coefficient for each dimension to coordinate the shrinking of different dimensions to achieve a higher compression rate while still maintaining accuracy. By means of the two-stage multi-dimensional compression framework, given a computation budget, we are able to comprehensively reduce redundant computation to meet the budget without sacrificing accuracy obviously. Based on the compression framework, we further propose a novel dynamic inference framework to adaptively process different input images with different models and resolutions at runtime. First, we utilize the compound shrinking strategy to compress the given baseline network and generate multiple sub-networks with diverse model sizes and accuracy, which are then cascaded in ascending order of the model size and then each input image will be processed by those models sequentially. At the end of the inference of each model, we propose a novel metric to evaluate the confidence of the prediction result. Once a confident prediction is obtained, the dynamic inference will be terminated without executing subsequent models. In practice, most input images can be confidently recognized by early models with small computational overhead, while large models will be activated only for a few hard samples. Consequently, compared to static inference with a single model,

the overall computational complexity of our dynamic inference is reduced significantly without compromising accuracy. Our main contributions are summarized as follows:

- We propose dynamic image cropping to reduce the spatial redundancy in images, where we design a lightweight detector to efficiently localize the foreground area of an image and conduct instance-aware dynamic cropping. Those finely cropped images can be correctly recognized even at a smaller resolution, which greatly reduces the computational cost of CNNs.
- We also propose compound shrinking to jointly compress the three dimensions of a CNN. We first quantify the impact of each dimension on accuracy and model complexity, and then generate the optimal joint compression strategy accordingly. By this means, we greatly reduce the redundancy in both input images and network architectures for a higher compression rate.
- We further introduce a dynamic inference framework to efficiently process input images with different recognition difficulties. We cascade multiple models from our compression framework and adaptively utilize different models and resolutions for different images. In this way, we effectively adjust the computational cost for different input images, reducing the overall computational cost without compromising the final accuracy.
- We seamlessly integrate the dynamic image cropping, compound shrinking, and dynamic inference into a deep compression framework (i.e., EdgeCompress) for efficient deep learning inference, which can optimally adapt the model cost to meet different resource constraints of embedded hardware while maximizing model accuracy.

Extensive experiments demonstrate the advantages of the proposed EdgeCompress over other SOTA model compression approaches. Specifically, EdgeCompress reduces the MACs of ResNet50 by 48.8% while improving the top-1 accuracy by 0.8% on ImageNet-1K. Moreover, compared to the SOTA compression framework, HRank [53], EdgeCompress also achieves 4.1% higher accuracy with similar model MACs.

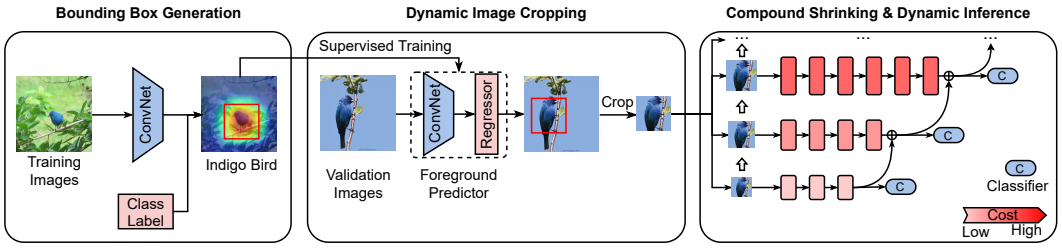


FIGURE 6.2: Overview of the proposed EdgeCompress framework, which mainly consists of four components: bounding box generation, dynamic image cropping, compound shrinking, and dynamic inference.

6.2 Framework Overview

In this section, we first outline the design of EdgeCompress and then describe each component in detail. As demonstrated in Figure 6.2, before inference, we first utilize Grad-CAM [209] to generate the salience map of all training images in the classification dataset $\mathcal{D}$, and then we generate a bounding box for each image according to the salience map and form a pseudo bounding box label set $\mathcal{B}$. Thereafter, we exploit the image-box pairs $\{\mathcal{D}_i, \mathcal{B}_i\}$ to train a lightweight predictor. Meanwhile, we use compound shrinking to jointly compress the three dimensions of a CNN and generate multiple CNNs with different computational complexities, which are then cascaded for dynamic inference. In inference, the input image will be first fed into the trained predictor to efficiently localize the foreground object. Thereafter, the foreground object will be cropped and sequentially sent to the CNN models generated by compound shrinking for dynamic inference. Once a confident prediction is obtained, the inference will be terminated immediately without executing subsequent models.

6.3 Dynamic Image Cropping

Bounding Box Generation: As aforementioned, dynamically cropping the foreground for inference is promising in reducing the computation and improving classification accuracy. However, for classification datasets like ImageNet-1K, there is no out-of-the-box position annotation for the foreground object. Moreover, the position of the foreground object varies in different images, which makes it difficult to efficiently localize the foreground object.

To address this limitation, we first use Grad-CAM [209] to automatically generate the position annotations. Specifically, let the class label of the given image be c. We first perform forward inference with a well-trained CNN (e.g. ResNet50) to obtain the prediction score p^c for class c, and then conduct backpropagation to compute the gradient of the score p^c with respect to each activation of the last convolutional layer. Thereafter, the gradients are aggregated within each channel via global average pooling. The obtained scalar for each channel can be seen as the weight of the channel, which can be calculated as follows:

$$a_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{pooling} \overbrace{\frac{\partial p^c}{A_{ij}^k}}^{gradients} \tag{6.1}$$

where a_k^c is the weight of channel k for class c, and A_{ij}^k is a single activation indexed by i and j in the 2-D feature map of channel k. With the weights of all channels determined, the salience map for class c can be obtained by computing the weighted sum of all feature maps over the channel dimension, which is formulated as:

$$L_{Grad\_CAM}^c = ReLU \underbrace{\left(\sum_k a_k^c A^k\right)}_{linear\ combination} \tag{6.2}$$

where A^k is the 2-D feature map of channel k, and ReLU is used to eliminate the impact of negative activations. Finally, the obtained salience map is upsampled to the same size as the input image via bilinear interpolation.

With the salience map generated, we then introduce a simple yet effective strategy to determine the bounding box of the foreground object. Initially, we set the box as the boundary of the image. Subsequently, we shrink the four sides of the box simultaneously, and once a side reaches our preset salience threshold t, the side is frozen. The bounding box is determined after all sides are frozen. Note that it is crucial for the final result to appropriately select the value of t. As demonstrated in Figure 6.3, a too-small threshold will result in residual background redundancy, while a too-large threshold will lose some important features. Therefore, we conduct empirical experiments to determine the optimal threshold value. As shown in Table 6.1, we achieve the highest accuracy when the threshold t is set to 0.5. Therefore, we set $t = 0.5$ in our experiments. Note that more fine-grained searching for t may further improve the accuracy, but it also increases the search cost.

FIGURE 6.3: By applying different salience threshold t, we can obtain different cropped images. The larger the threshold value, the more radical the cropping.

TABLE 6.1: Impact of using different salience thresholds on prediction accuracy. The model is trained and evaluated on ImageNet-1K. $t = 0$ means using the original images without Grad-CAM cropping.

Model	Params (M)	MACs (B)	t	Top-1 Acc. (%)
ResNet50	25.6	4.1	0.00	76.02
			0.25	76.45
			0.50	**76.88**
			0.75	76.32

Finally, the generated box annotations are saved in the form of $[X_{min}, Y_{min}, X_{max}, Y_{max}]$, which denotes the boundary of the foreground in the image.

Figure 6.4 shows that we are capable of accurately localizing the foreground of images with Grad-CAM. However, Grad-CAM cannot be directly applied to edge applications because of the time-consuming backpropagation process. Moreover, Grad-CAM requires the class label as weak supervision, which is unavailable for validation images. To address these issues, we design a foreground predictor to efficiently localize the foreground of input images.

Predictor Architecture: Existing detection models, such as Faster R-CNN [108], are mainly proposed for object detection tasks (e.g., MS COCO [2]), which usually contain a large number of parameters and computation to accurately localize and identify the multiple objects in each input image. However, we focus on classification tasks, where each input image contains only one object and thus the localization difficulty is much lower than in detection tasks. Moreover, to achieve dynamic cropping, we only need to output the position of the foreground without predicting its label. Consequently, existing detection models become redundant and inefficient in our context. To this end, we design a novel lightweight foreground predictor to efficiently localize the unique foreground object of each input image. The details of the proposed foreground predictor are summarized in Table 6.2, which consists of several residual bottleneck blocks [15] and a fully connected layer. A residual bottleneck contains two convolutional layers with 1×1 kernels and

FIGURE 6.4: Bounding boxes generated with the salience threshold $t = 0.5$, which accurately localize the key object in each image.

TABLE 6.2: Architecture of the proposed box predictor. C denotes the number of channels and L denotes the number of layers.

Stage	Block	Resolution	C	L
1	Conv 3×3	224 × 224	16	1
2	Residual Bottleneck	112 × 112	16	2
3	Residual Bottleneck	56 × 56	32	2
4	Residual Bottleneck	28 × 28	32	2
5	Residual Bottleneck	14 × 14	64	2
6	Pooling & Linear	7 × 7	4	1
Params: 0.27M MACs: 0.09B				

one convolutional layer with 3×3 kernels in the middle. The computational cost mainly results from the 3×3 convolutional layer. Therefore, to reduce the cost and accelerate the predictor, we only stack two residual bottleneck blocks in each stage and each block is only equipped with a small number of channels. Consequently, the proposed predictor only contains 0.27M parameters and 0.09B MACs, which is negligible compared to popular object detectors (e.g., Faster R-CNN with 134.7M (499×) parameters and 15.1B (167.8×) MACs [210]).

Training of Foreground Predictor: We train the predictor in a supervised manner. First, we generate a bounding box label set $\mathcal{B}$ for all training images, then the labels are utilized to train the predictor. We use the mean square error (MSE) as the loss function. Let $\mathcal{P}_i = [X^p_{min}, Y^p_{min}, X^p_{max}, Y^p_{max}]$ be the output of the predictor, and $\mathcal{G}_i = [X^g_{min}, Y^g_{min}, X^g_{max}, Y^g_{max}]$ be the generated box label, the

loss function is formulated as:

$$\begin{aligned}\mathcal{L}_{box} &= MSELoss(\mathcal{P}_i,\ \mathcal{G}_i) \\ &= \frac{1}{4}((X^g_{min} - X^p_{min})^2 + (Y^g_{min} - Y^p_{min})^2 \\ &\quad + (X^g_{max} - X^p_{max})^2 + (Y^g_{max} - Y^p_{max})^2)\end{aligned} \tag{6.3}$$

To balance the training overhead and prediction accuracy, we train the predictor with Adam [211] optimizer for 40 epochs. The initial learning rate is set to 1e-3, and the learning rate is scheduled using exponential decay [212]. The training of the box predictor is decoupled with backbone networks. Once the predictor is trained, it can be directly applied to different classification backbones without any training overhead. During inference, the trained predictor will quickly localize the foreground object of the input image and generate a finely cropped image, which significantly reduces the redundancy in the input image.

6.4 Compound Model Shrinking

The proposed DIC significantly reduces the redundancy in images, improving computational efficiency. We observe that redundancy also exists in network architectures (e.g., redundant parameters), and only removing the redundancy in images loses the opportunity to further compress the model for embedded hardware. Besides, [40] demonstrates that jointly adjusting different dimensions promises higher accuracy. To this end, we propose a compound shrinking (CS) strategy to jointly compress the three dimensions (depth, width, resolution) of CNNs to further reduce the redundancy in images as well as networks while maintaining accuracy.

Intuitively, shrinking different dimensions has different impacts on accuracy and model overhead. The core of our compound shrinking strategy is to calculate a shrinking coefficient for each dimension according to their trade-off between accuracy and model overhead. A larger coefficient denotes more radical shrinking. More specifically, the dimension with a steep accuracy drop during shrinking will be assigned a small shrinking coefficient to prevent severe accuracy degradation. To calculate the shrinking coefficients, we first quantify the trade-off of each dimension between accuracy and model overhead. Here we use MACs as the metric to

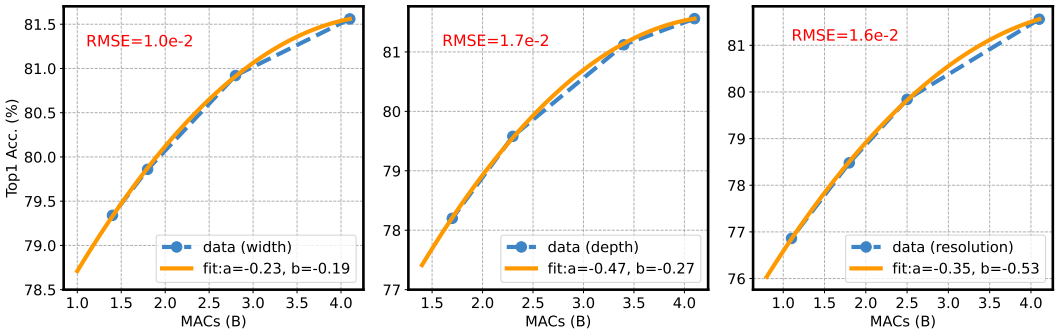


FIGURE 6.5: The actual accuracy (blue dotted line) and the estimated accuracy (yellow line) over MACs by separately shrinking the three dimensions. The low root mean square error (RMSE) indicates that the accuracy estimator can well fit the sampled data.

measure the cost of models because all three dimensions are related to the MACs of a model while only the depth and width can affect the model parameters. Given a MACs budget $\mathcal{M}$, we first obtain the accuracy drops resulting from separately shrinking different dimensions, which can be represented as:

$$\Delta A_s(\mathcal{M}) = A_0 - A_s(\mathcal{M}) \tag{6.4}$$

where $s \in \{d, w, r\}$ represents the shrunk dimension, $A_s(\mathcal{M})$ denotes the accuracy of the shrunk model, and A_0 is the accuracy of the original model. To comply with the rule that the steeper the drop in accuracy, the smaller the coefficient of the corresponding dimension, we design the following equation to determine the shrinking coefficient for each dimension:

$$\mathcal{C}_s(\mathcal{M}) = \frac{\sqrt[3]{\Delta A_d(\mathcal{M}) \cdot \Delta A_w(\mathcal{M}) \cdot \Delta A_r(\mathcal{M})}}{\Delta A_s(\mathcal{M})} \tag{6.5}$$

where $\mathcal{C}_s(\mathcal{M})$ denotes the shrinking coefficient of the dimension s ($s \in \{d, w, r\}$). Through Equation 6.4 and Equation 6.5, we are able to efficiently calculate the coefficients once we obtain the accuracy degradation of the three dimensions in the given MACs regime.

However, the training cost of the compressed models to calculate the accuracy drop is still non-negligible. To mitigate the training overhead, we propose a dimension-wise accuracy estimator to quickly estimate the accuracy of the compressed models and calculate the accuracy degradation resulting from shrinking different dimensions in the given MACs regime. First, we sample a couple of models with different

MACs by separately shrinking the three dimensions. As demonstrated in Figure 6.5, the accuracy distribution of the three dimensions along MACs can be well-fitted by a quadratic polynomial. Therefore, we design a simple yet effective polynomial estimator to predict the accuracy with respect to the target MACs $\mathcal{M}$. The estimator is formulated as follows:

$$A_s(\mathcal{M}) = a_s(\mathcal{M} - \mathcal{M}_0)^2 + b_s(\mathcal{M} - \mathcal{M}_0) + A_0 \tag{6.6}$$

where $\mathcal{M}_0$ is the MACs of the original model. a_s and b_s are the hyperparameters to fit for dimension s ($s \in \{d, w, r\}$). Subsequently, we train the dimension-wise estimator using least square regression with the aforementioned sampled data. Figure 6.5 shows that the proposed estimator can well fit existing data. Due to the simple and intuitive design of the estimator, we only need to sample and train very few models to train the estimator, and this cost is a one-time cost. With the accuracy estimator established, we are able to quickly estimate the accuracy drop and then calculate the optimal shrinking coefficients for the three dimensions under any given resource constraint. According to the coefficients, we will jointly compress the three dimensions of the baseline network and generate a compact model with optimized efficiency. As the compressed model can be viewed as a subset of the baseline network, we call the compressed model a sub-network.

6.5 Dynamic Inference

Through dynamic image cropping and compound shrinking, we can optimally compress a CNN model to different complexities to satisfy various resource constraints in edge environments. Given an embedded device, an intuitive deployment strategy is to select a single model that best fits the hardware capabilities (e.g., memory capacity, computing power) to balance the trade-off between accuracy and execution efficiency. However, as different images correspond to distinct recognition difficulties [65, 74], using a single model for all images may over-process simple images and waste resources, while for complex images, the model may under-process them and generate wrong predictions, leading to a sub-optimal trade-off between accuracy and efficiency.

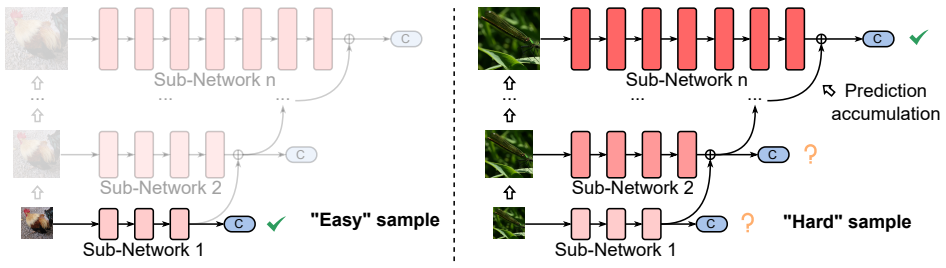


FIGURE 6.6: The proposed dynamic inference framework, which utilizes multiple sub-networks to achieve instance-aware inference. These sub-networks are obtained by compressing the baseline network using compound shrinking.

TABLE 6.3: Specifications of the sub-networks generated by the compound shrinking strategy. The baseline network is ResNet50. The accuracy is measured on ImageNet-1K and the latency is measured on Jetson Nano.

Sub-network Id	Params (M)	MACs (B)	Latency (ms)	Top1 Acc. (%)
1	11.65	1.21	27.15	73.86
2	11.79	1.30	28.05	74.21
3	13.53	1.84	41.62	75.56
4	15.40	2.40	49.41	76.32
5	20.30	3.22	56.01	76.81
6	25.90	4.20	57.09	77.20

To address this problem, we propose a dynamic inference strategy to further optimize the run-time efficiency of CNNs on embedded devices without sacrificing accuracy. As demonstrated in Figure 6.6, we first apply different MACs constraints to the compound shrinking strategy to generate multiple sub-networks with different accuracy and overhead. The specifications of all sub-networks are summarized in Table 6.3, where we observe that the obtained sub-networks are different from each other in terms of model size (i.e., the number of parameters) and model MACs. This reveals that both the model architecture and the input data are compressed. Consequently, we achieve significant reductions in the on-device inference latency with minimal accuracy drop. Thereafter, we deploy the generated sub-networks onto the target hardware before inference and dynamically activate different sub-networks at runtime for better accuracy and efficiency. For easy samples with a distinct foreground, maybe only the smallest sub-network will be activated to efficiently generate the correct prediction, while for hard samples with which small models are unable to produce a confident prediction, larger sub-networks will be gradually activated until a confident prediction is obtained. By doing so, we can avoid unnecessary computation and resource consumption for simple images, improving inference efficiency.

Termination Condition: Modern large-scale datasets for image classification

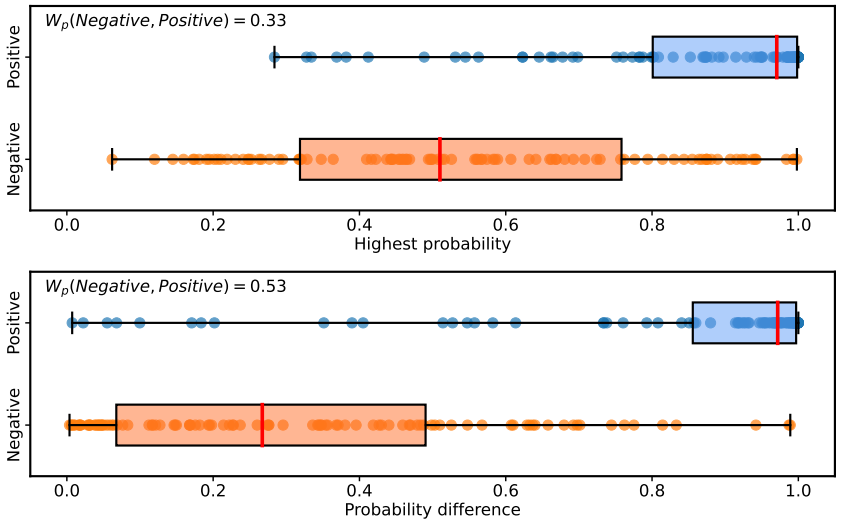


FIGURE 6.7: Distributions of negative results and positive results along different confidence metrics. W_p denotes the Wasserstein distance between negative results and positive results. The larger the value of W_p, the more distinct the two distributions, such that our dynamic inference framework can more accurately determine whether the sample is correctly classified.

usually contain millions of images. For example, ImageNet-1K has about 1.3 million images. It is non-trivial to determine when to terminate the inference for each image. Current dynamic inference approaches, such as multi-scale inference [69] and early-exit networks [213, 214], exploit the highest prediction probability among all classes as the prediction confidence to control the termination of dynamic inference. Given a CNN $\mathcal{N}$ and an image x, the prediction confidence of existing methods can be represented as:

$$\begin{aligned} \mathcal{I}_{\mathcal{N}} &= \max\left(Softmax(\mathcal{N}(x))\right) \\ &= \max\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right) \quad for\ i = 1, 2, \ldots, K \end{aligned} \tag{6.7}$$

where z_i denotes the i-th logit (i.e., the i-th output of the fully connected layer) of $\mathcal{N}$, which is transformed into the prediction probability for the i-th class with the Softmax function. With Equation 6.7, the prediction confidence can be efficiently calculated using the output of the network. Generally, if a high prediction confidence score is obtained from the current model, then the image is considered correctly classified and the inference will be terminated immediately.

In this chapter, we rethink the efficacy of the confidence metric. First, we randomly sample 100 negative prediction results and 100 positive prediction results

from a well-trained model. Subsequently, we summarize the distribution of the sampled data along the highest prediction probability in Figure 6.7 and exploit the Wasserstein distance to quantify the similarity between the distributions of positive samples and negative samples. The smaller the Wasserstein distance between two distributions, the more similar the two distributions are. As shown in the upper figure of Figure 6.7, the negative samples and positive samples are distributed close along the highest probability with a small Wasserstein distance, which reveals that this confidence metric fails to effectively separate the positive predictions and negative predictions of a model, degrading the efficacy of dynamic inference. To address the issue, we introduce a novel metric, the probability difference, to control the termination of dynamic inference. The probability difference is defined as the difference between the highest prediction probability and the second highest probability, which is formulated as:

$$\mathcal{D}_{\mathcal{N}} = \mathcal{I}_{\mathcal{N}} - \mathcal{I}'_{\mathcal{N}} \tag{6.8}$$

where $\mathcal{I}'_{\mathcal{N}}$ represents the second highest prediction probability. Equation 6.8 reveals that, unlike the existing confidence metric which only focuses on the highest prediction probability, the proposed confidence metric considers both the highest prediction itself and its advantages over other competitors. The distributions of the negative samples and positive samples along the probability difference are demonstrated in the lower figure of Figure 6.7, where we observe that the two distributions are more distinct and the Wasserstein distance between them is much larger compared to the existing confidence metric (i.e., the highest probability), which indicates that the proposed confidence metric is able to estimate the correctness of a prediction more accurately, enabling effective control over the dynamic inference.

After evaluating the confidence of a prediction with the proposed metric, we will compare the evaluation result with a preset threshold value $\mathcal{D}_0$. If the evaluation result is larger than the preset threshold value, the prediction is considered confident and the inference will be terminated. Otherwise, the image will be sent to a larger model for more accurate prediction. By changing the threshold value $\mathcal{D}_0$, we are able to flexibly adjust the trade-off of the dynamic inference between inference overhead and accuracy. Specifically, a higher threshold value will force more images to flow to large models, and thus the accuracy will be improved at the cost

TABLE 6.4: Comparison of different confidence metrics in terms of the trade-off between model complexity and accuracy. The accuracy is measured on ImageNet-1K.

Architecture	Metric	MACs (B)	Top-1 Acc. (%)
ResNet50	Highest probability	2.07	76.44
	Probability difference	2.07	**76.68**
	Highest probability	2.39	76.91
	Probability difference	2.33	**77.07**

of higher inference costs. On the contrary, reducing the threshold value will allow more images to exit at small models, thereby saving the inference overhead. To validate the proposed confidence metric, we perform experiments on ImageNet-1K and present the results in Table 6.4, where we observe that the proposed metric remarkably improves accuracy without sacrificing the computational cost.

Prediction Accumulation: During dynamic inference, hard samples may flow through multiple models. Some approaches directly adopt the output of the last model as the final result [68], which wastes the information from the previously executed models and consequently losses the opportunity to further improve accuracy. Instead, some other methods propose to utilize the information of previous models by merging the feature maps from previous models into the current model for higher accuracy [74]. However, the fusion of feature maps of different models introduces additional computational overhead, reducing the efficiency of dynamic inference.

To address the above concerns, we propose prediction accumulation to effectively utilize the information from different models for higher accuracy. Different from the fusion of feature maps [74] which requires a large amount of additional computation, we efficiently integrate the information from different models without compromising the computational overhead by accumulating the output of the last fully connected layer in each model (i.e., the logits), which is formulated as:

$$\mathcal{Z}'_i = \alpha \mathcal{Z}_i + \mathcal{Z}'_{i-1} \tag{6.9}$$

where $\mathcal{Z}_i$ denotes the logits of the current model, and $\mathcal{Z}'_i$ represents the accumulated logits of the current model, which will be used to calculate the prediction of the current model. $\mathcal{Z}'_{i-1}$ is the accumulated logits of the previous model and α is a hyperparameter to control the contribution of the prediction of the current

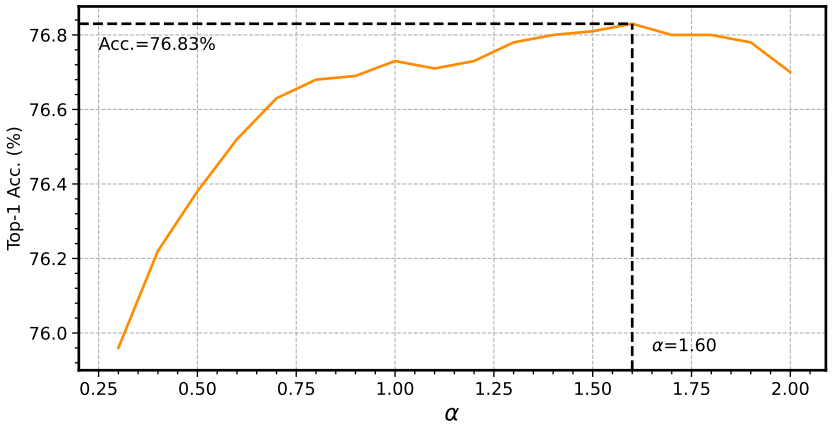


FIGURE 6.8: Impact of the value of α on the final accuracy of dynamic inference. We observe the highest accuracy at $\alpha = 1.60$, and thus we fix $\alpha = 1.60$ for subsequent experiments. The target dataset is ImageNet-1K.

TABLE 6.5: Impact of our prediction accumulation strategy on model computation, inference latency, and accuracy. The inference latency is measured on Jetson Xavier, and the accuracy is measured on the ImageNet-1K dataset.

Architecture	Accumulation	MACs (B)	Latency (ms)	Top-1 Acc. (%)
ResNet50	✗	2.07	7.84	76.68
	✓	2.07	7.91	**76.83**
	✗	2.33	8.63	77.07
	✓	2.33	8.60	**77.35**

model (i.e., $\mathcal{Z}_i$). The value of α can affect the final accuracy of dynamic inference obviously. To identify the optimal value of α, we sample multiple values of α and summarize the relationship between accuracy and α in Figure 6.8, where we observe the highest accuracy when $\alpha = 1.60$. Therefore, we fix $\alpha = 1.60$ in our experiments.

The logits of a model can directly determine the prediction results, and thus accumulating logits can effectively utilize the information from multiple models for higher accuracy. The overhead of accumulating logits is determined by the number of logits in each model and the number of models to accumulate. Specifically, the computational cost of logits accumulation can be calculated as follows:

$$Q_{acc} = n_l \cdot (n_m - 1) \tag{6.10}$$

where n_l is the number of logits and n_m is the number of models. For example, for two models with 1,000 logits, the computational cost of logits accumulation will be $1,000 \times (2 - 1) = 1,000$ FLOPs, which can be neglected compared to the

Algorithm 4: Dynamic Inference Algorithm

Data: CNN models $\{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_n\}$, input image x, termination threshold D_0
Result: Prediction result o

$i \leftarrow 0$;
while $i < n$ **do**
  $\mathcal{Z}_i \leftarrow \mathcal{N}_i(x)$ `/* Inference with the current model */`
  if $i = 0$ **then**
    $\mathcal{Z}'_i \leftarrow \mathcal{Z}_i$
  else
    $\mathcal{Z}'_i \leftarrow AccumulateLogits(\mathcal{Z}_i,\ \mathcal{Z}'_{i-1})$ `/* See Eq. 6.9 */`
  `/* Calculate the termination metric, see Eq. 6.8 */`
  $\mathcal{D}_{\mathcal{N}_i} \leftarrow CalculateMetric(\mathcal{N}_i,\ \mathcal{Z}'_i)$
  if $\mathcal{D}_{\mathcal{N}_i} > \mathcal{D}_0$ **then**
    Break `/* Terminate dynamic inference */`
  $i \leftarrow i + 1$ `/* Activate the next model */`
$o \leftarrow Softmax(\mathcal{Z}'_i)$ `/* Calculate the final result */`

inference overhead of CNN backbones (e.g., ResNet50 with 4.1 Billion FLOPs). As shown in the experimental results in Table 6.5, the accumulation strategy improves the accuracy remarkably while not increasing the computational cost and inference latency of our framework.

Dynamic Inference Algorithm: We demonstrate our dynamic inference algorithm in detail in Algorithm 4. Given a series of CNN models ordered from low to high computational complexity and an input image, we initiate the dynamic inference with the smallest model and gradually activate models with higher computational complexity. For each model, we first perform inference with the model to obtain its prediction logits, then, we accumulate the logits of this model with all previously executed models as Equation 6.9. Subsequently, the proposed termination metric of the current model is calculated using the accumulated logits according to Equation 6.8, which is then compared with the preset threshold $\mathcal{D}_0$. If the calculated metric is larger than the threshold, the predicted result is considered confident and dynamic inference will be terminated immediately. Otherwise, a larger model will be activated for inference. In practice, we observe that, for most images, dynamic inference is able to produce a confident prediction and be terminated at the smallest model. In this case, the overall latency of dynamic inference is equal to the inference latency of the smallest model. Consequently, we avoid using large models for most images, saving computation and reducing

TABLE 6.6: Hardware specifications of three platforms. The column "Cores" denotes the number of CUDA cores and CPU cores for GPU platforms (i.e., Jetson Xavier and Jetson Nano) and the CPU platform (I7-9750H), respectively.

Device	Power	Memory	Cores	Core Freq.	Performance
Jetson Xavier	15 W	32 GB	512	900 MHz	11.0 TOPS
Jetson Nano	5 W	4 GB	128	992 MHz	0.5 TOPS
i7-9750H	45 W	16 GB	6	2600 MHz	0.4 TOPS

latency significantly compared to static inference. Since the inference latency may vary with input images, i.e., hard samples have higher latency while easy samples have lower latency, we report the average latency of processing one image among the whole dataset as the latency of our dynamic inference framework. The results are presented in the Experimental Results section.

6.6 Experiments

In this section, we perform extensive experiments on different benchmarks to validate the efficacy of EdgeCompress and demonstrate its advantages over existing SOTA approaches in terms of accuracy, computational complexity (i.e., MACs), and run-time efficiency. Further, we conduct experiments study to show the contribution of each component in our framework.

Hardware Devices: To validate the run-time efficiency of EdgeCompress, including inference latency and throughput, we select two representative embedded GPU platforms, NVIDIA Jetson Xavier and Jeton Nano, and Intel i7-9750H@2.6GHz CPU to deploy different methods and compare their performance. The specifications of selected devices are shown in Table 6.6.

Datasets: We validate the proposed EdgeCompress on four representative datasets: 1) CIFAR-10, 2) CIFAR-100, 3) ImageNet-100, and 4) ImageNet-1K [1]. ImageNet-1K (also known as ImageNet or ILSVRC-2012) is one of the most popular large-scale datasets for image classification, which includes 1,000 classes. ImageNet-100 is a subset of ImageNet-1K, which consists of 100 classes randomly selected from ImageNet-1K. The details of ImageNet-100 can be found in the code repository. All images are preprocessed following a simple configuration as [31].

Networks: We apply our EdgeCompress framework to three widely utilized CNN backbones, VGG16_BN [42], ResNet50 [15], and RegNet-X [31]. For each model,

we employ EdgeCompress to remove the spatial redundancy in input images and the architecture redundancy in networks, thereby reducing the computational cost and improving the inference efficiency. As a comparison, we also report the results of other methods.

Optimization Settings: All models in our experiments are trained using SGD optimizer with a momentum of 0.9. We first train models for 100 epochs without using dynamic image cropping, where the first 5 epochs are for warmup. For experiments on ImageNet-100 and ImageNet-1K, the learning rate is set to 2.0, which will be decayed by exponential learning rate policy with a decay factor of 0.02. The training batch size is set to 1024. Subsequently, the proposed dynamic image cropping is utilized to fine-tune the pretrained models for 20 epochs. The learning rate for fine-tuning is 5e-4. In addition, we also use label smoothing with the smoothing factor $\epsilon = 0.1$ [49] to prevent overfitting. For experiments on CIFAR-10 and CIFAR-100, the initial learning rate is 0.1 and the training batch size is 128.

Evaluation Methodology: In this chapter, we propose three novel approaches to comprehensively reduce the computational cost of CNNs. Thanks to the flexible design of these approaches, they can be used separately or coupled for a higher compression ratio. To better demonstrate the flexibility and efficacy of our design, we evaluate different combinations of the three approaches. Specifically, EC-DIC represents that only the dynamic image cropping component is exploited, while EC-Static denotes both the dynamic image cropping and compound shrinking are adopted. Finally, EC-Dynamic denotes the completed framework that contains all three components, which further integrates the dynamic inference approach based on EC-Static.

Results on ImageNet-100: We conduct experiments on ImageNet-100 with ResNet50 and RegNet-X, where we use different methods to compress models to different complexities. As a comparison, we use the most popular image cropping method, ResizedCenterCrop (RCC) to crop and resize images to different sizes. Finally, all models are deployed to hardware to evaluate latency and throughput.

ResNet50: As shown in Table 6.7, all of our approaches outperform the competitor (i.e., RCC) in terms of model complexity, on-device execution efficiency, and

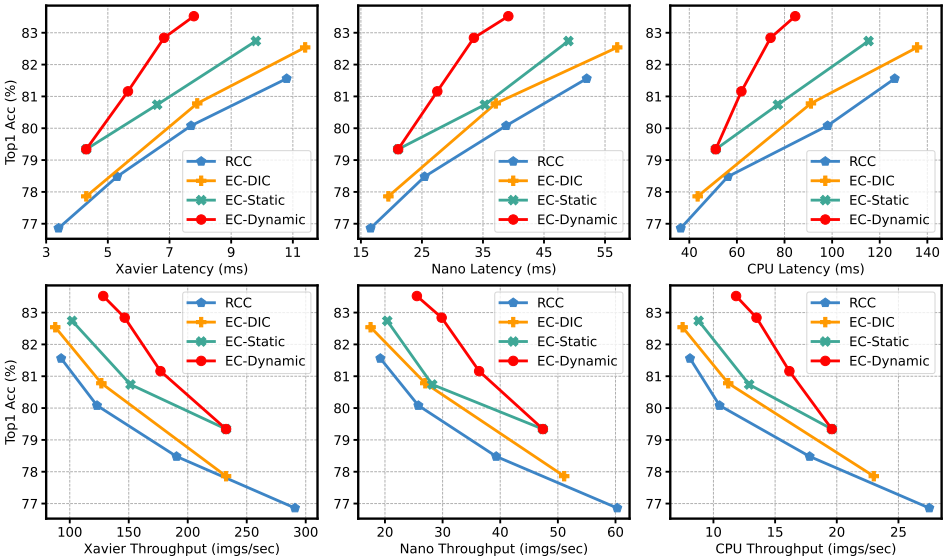


FIGURE 6.9: Actual performance of ResNet50 compressed by different methods on three distinct hardware devices. Accuracy is measured on ImageNet-100.

TABLE 6.7: Results of ResNet50 on ImageNet-100. RCC-Baseline represents the baseline ResNet50 model, where we crop and resize all images to the size 224×224 with RCC. "CR" denotes the compression ratio.

Method	Params (M)	MACs (B)	MACs CR (%)	Top1 Acc. (%)
RCC-Baseline	23.7	4.1	0.0	81.6
EC-DIC	24.0	4.2	-2.4	82.5
EC-Static	17.3	3.0	26.8	82.7
EC-Dynamic (ours)	**13.6**	**2.0**	**51.2**	**83.5**
RCC	23.7	3.0	26.8	80.1
EC-DIC	24.0	2.6	36.6	80.8
EC-Static	14.5	2.4	41.5	81.5
EC-Dynamic (ours)	**11.8**	**1.7**	**58.5**	**82.8**
RCC	23.7	1.1	73.2	76.9
EC-DIC	24.0	1.2	70.7	77.9
EC-Static	**7.8**	**1.0**	**75.6**	79.3
EC-Dynamic (ours)	8.9	1.2	70.7	**80.2**

accuracy. Specifically, compared to the baseline ResNet50 (RCC-Baseline), EC-DIC improves the accuracy by 0.9% with a negligible increase in model parameters (1.2%) and MACs (2.4%), while EC-Static further pushes up the accuracy improvement to 1.1% with a parameter reduction of 27.0% and a MACs reduction of 26.8%. Finally, EC-Dynamic achieves the best performance, which compresses the MACs by 51.2% while still improving the accuracy by 1.9% compared to RCC-Baseline. In the low complexity regime, EC-Dynamic achieves 3.3% higher accuracy than RCC with only 37.6% model parameters (8.9M v.s. 23.7M) and similar MACs. Unlike EC-DIC which only compresses the input data, both EC-Static and EC-Dynamic use the compound shrinking algorithm to jointly compress the model architecture and input data, thereby achieving significant reductions in model parameters. In

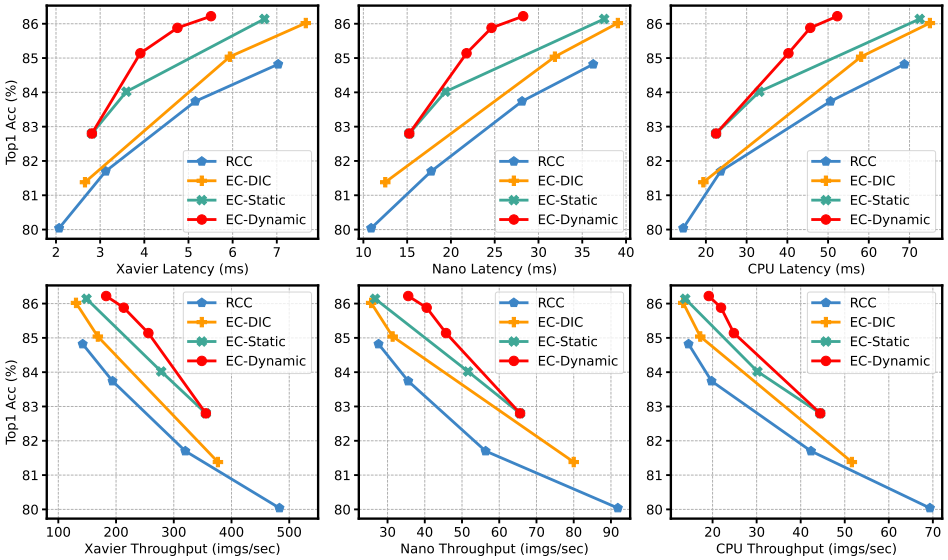


FIGURE 6.10: Actual performance of RegNet-X compressed by different methods on three distinct hardware devices. Accuracy is measured on ImageNet-100.

TABLE 6.8: Results of RegNet-X on ImageNet-100. RCC-Baseline represents the baseline RegNet-X model with all input images cropped and resized to 224×224 with RCC. "CR" denotes the compression ratio.

Method	Params (M)	MACs (B)	MACs CR (%)	Top1 Acc. (%)
RCC-Baseline	8.4	1.6	0.0	84.8
EC-DIC	8.7	1.3	18.8	85.0
EC-Static	6.3	1.3	18.8	86.1
EC-Dynamic (ours)	**4.4**	**0.8**	**50.0**	**86.2**
RCC	8.4	0.7	56.3	82.3
EC-DIC	8.7	0.8	50.0	83.8
EC-Static	3.6	0.6	62.5	84.0
EC-Dynamic (ours)	**3.3**	**0.6**	**62.5**	**85.1**
RCC	8.4	0.4	75.0	80.0
EC-DIC	8.7	0.5	68.8	81.4
EC-Static	**2.5**	**0.4**	**75.0**	82.8
EC-Dynamic (ours)	2.8	0.5	68.8	**83.7**

other words, the difference in parameters between EC-DIC and EC-Static originates from our compound shrinking algorithm. Meanwhile, Figure 6.9 indicates that all of our methods outperform RCC by a large margin across a wide spectrum of inference latency and throughput on different resource-constrained embedded devices. Particularly, EC-Dynamic achieves 83.5% top-1 accuracy with a latency of 7.8ms on Xavier, which is 1.9% higher in accuracy and 27.7% lower in latency compared to RCC (81.6% top1 accuracy, 10.8ms). At the same time, the throughput of EC-Dynamic on Xavier is 128.4 *imgs/sec*, which is 38.5% higher than RCC (92.7 *imgs/sec*). On Nano and Intel i7-9750H CPU, EC-Dynamic also improves the throughput by 33.0% and 46.2%, and reduces the latency by 24.7% and 33.1%, respectively.

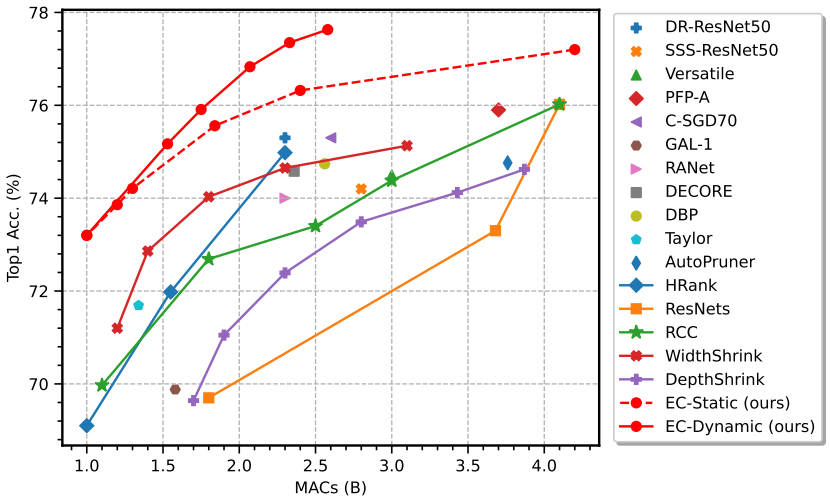


FIGURE 6.11: Comparison of our EdgeCompress with other model compression methods. The baseline model is ResNet50 and the dataset is ImageNet-1K.

RegNet-X: The experimental results of RegNet-X are summarized in Table 6.8 and Figure 6.10, where we also observe a significant improvement of our method. EC-Dynamic outperforms the baseline RegNet-X (RCC-Baseline) with an improvement of 1.4% in accuracy and a reduction of 50.0% in MACs. Meanwhile, EC-Dynamic reduces the model parameters by 47.6% (4.4M v.s. 8.4M). In the low MACs regime, EC-Dynamic observes a remarkable 3.7% improvement in accuracy with only 33.3% parameters (2.8M v.s. 8.4M) compared to RCC. As for the real performance on hardware, EC-Dynamic obtains an accuracy of 86.2% with 5.5ms latency on Xavier, which is 1.4% higher in accuracy and 21.4% lower in latency than RCC (84.8% top-1 accuracy, 7.0ms). Similarly, the latency reductions of EC-Dynamic on Nano and Intel CPU are 22.0% and 24.0%, respectively. Besides, EC-Static also observes a 29.0% throughput improvement (35.6 *imgs/sec* v.s. 27.6 *imgs/sec*) on Nano and a 31.7% throughput improvement (19.1 *imgs/sec* v.s. 14.5 *imgs/sec*) on CPU compared to RCC.

Results on ImageNet-1K: In this subsection, we evaluate our approach on ImageNet-1K, and compare the evaluation results with many SOTA CNN compression frameworks. To enable a comprehensive comparison with more SOTA frameworks, we employ ResNet50 as the baseline network. In addition, we also compare our compressed models with many popular backbone architectures in different computation regimes.

Comparison with SOTA Compression Methods: Starting with the baseline ResNet50

TABLE 6.9: Comparison with other popular backbone networks. "CR" denotes the compression ratio.

Model	Params (M)	MACs (B)	MACs CR (%)	Top1 Acc. (%)
ResNet50[15]	25.6	4.1	0.0	76.0
ResNet101[15]	44.6	7.9	-92.7	77.4
DenseNet161[47]	28.7	7.9	-92.7	77.1
InceptionV3[49]	27.2	5.8	-41.5	77.3
EC-DIC	25.9	4.2	-2.4	77.2
EC-Dynamic (ours)	**20.4**	**2.6**	**36.6**	**77.6**
ResNet34[15]	21.8	3.7	9.8	73.3
DenseNet169[47]	**14.2**	3.4	17.1	75.6
EC-DIC	25.9	3.1	24.4	76.3
EC-Static	15.4	2.4	41.5	76.3
EC-Dynamic (ours)	17.1	**2.1**	**48.8**	**76.8**
ResNet18[15]	11.7	1.8	56.1	69.8
DenseNet121[47]	**8.0**	2.9	29.3	74.6
BN-Inception[215]	11.2	2.1	48.8	73.5
EC-DIC	25.9	1.9	53.7	74.9
EC-Static	13.5	1.8	56.1	75.6
EC-Dynamic (ours)	15.1	**1.8**	**56.1**	**75.9**

model, we implement different model shrinking methods, including resolution shrinking (RCC), width shrinking (WidthShrink) [29, 41], depth shrinking (DepthShrink) [15], EC-DIC, EC-Static, and EC-Dynamic to compress the three dimensions of the model to different MACs regimes and compare their performance. In addition, we also report the performance of multiple SOTA model compression techniques from the related papers, including DR-ResNet50 [65], SSS-ResNet50 [126], Versatile [183], PFP-A [192], C-SGD70 [191], GAL-1 [184], HRank [53], AutoPruner [206], and RANet [74]. The comparison results are summarized in Figure 6.11, which shows that our method achieves the highest accuracy across a wide range of MACs. Particularly, compared to the baseline ResNet50, our EC-Static achieves 1.2% accuracy improvement (76.0% to 77.2%) with a negligible increase in MACs (4.1B to 4.2B). Moreover, EC-Dynamic further improves the accuracy to 77.6% with only 2.6B MACs, which is 1.6% higher in accuracy and 36.6% lower in MACs compared to the baseline ResNet50. As we continue to reduce the MACs budget, EC-Dynamic reduces the MACs by 48.8% (4.1B to 2.1B) while still achieving 0.8% higher accuracy (76.0% to 76.8%). In the lowest MACs regime, EC-Dynamic and EC-Static achieve similar trade-offs between model MACs and accuracy, both of which remarkably improve the accuracy by 4.2% (70.0% to 74.2%) compared to RCC with similar MACs. In comparison with other SOTA compression methods, our method also achieves the best trade-off between MACs and accuracy. For example, EC-Dynamic achieves 5.3% higher accuracy (75.2% v.s. 69.9%) than GAL-1 [184] with less MACs (1.5B v.s. 1.6B).

Comparison with Popular Backbones: In this experiment, we compare our results on ResNet50 with other models from the ResNet family, such as ResNet101 and ResNet34, etc. In addition, we also conduct extensive comparisons with other popular backbones like DenseNets [47] and the Inception family [49, 215]. As demonstrated in Table 6.9, in the highest MACs regime, EC-Dynamic uses 67.1% less MACs (2.6B v.s. 7.9B) to achieve 0.5% higher accuracy than DenseNet161, while in the lowest MACs regime, EC-Dynamic also obtains the highest top-1 accuracy (75.9%), which is 6.1% and 2.4% higher than ResNet18 (69.8%) and BN-Inception (73.5%), respectively. The comparison results with other backbones reveal that our method can achieve promising results without redesigning the network architecture, which avoids the time-consuming exploration of the design space.

Comparison with SOTA Dynamic Inference Frameworks: We can observe from the above experiments that our compression framework with dynamic inference (i.e., EC-Dynamic) surpasses the one without dynamic inference (i.e., EC-Static) in model efficiency and accuracy. To further validate the advantages of our dynamic inference framework, we compare it with multiple SOTA dynamic inference frameworks. The comparison results are shown in Table 6.10, from which we observe that our EC-Dynamic framework achieves significant improvements in accuracy without sacrificing model complexity compared to other approaches. It is worth noting that the most of existing dynamic inference approaches only adjust a single dimension during inference, while our approach enables the joint adaptation of the three dimensions based on our multi-dimensional compression framework, achieving higher accuracy and efficiency. This also reveals that all components of our framework can be seamlessly coupled for a better result.

On-Device Efficiency of Dynamic Inference: In this experiment, we demonstrate the running efficiency of our approach on various edge devices and analyze how the preset threshold affects the on-device latency and accuracy. As shown in Table 6.11, we first apply a small threshold (i.e., 0.1) to our dynamic algorithm, which achieves higher accuracy and lower latency than static inference. As we increase the threshold, there are more images whose prediction confidence is smaller than the threshold, and thus more images are sent to the larger model for further inference. Consequently, both the classification accuracy and average inference latency of processing one image increase.

TABLE 6.10: Comparison with other dynamic inference frameworks. {d, w, r} denote the dimensions involved for dynamic inference.

Method	d	w	r	MACs (B)	Top-1 Acc. (%)
ResNet50 [15]				4.1	76.0
SkipNet [73]	✓			3.6	76.2
ConvNet-AIG [78]	✓			3.1	76.2
Channel Selection [157]		✓		**2.5**	76.2
DR-ResNet [65]			✓	3.2	77.0
EC-Dynamic (ours)	✓	✓	✓	2.6	**77.6**
RANet [74]	✓		✓	**2.0**	75.2
ConvNet-AIG [78]	✓			2.6	75.3
DR-ResNet [65]			✓	2.3	75.3
MSDNet [69]	✓			2.1	75.7
Channel Selection [157]		✓		2.3	76.1
EC-Dynamic (ours)	✓	✓	✓	2.1	**76.8**
RANet [74]	✓		✓	1.5	74.6
MSDNet [69]	✓			1.6	74.8
EC-Dynamic (ours)	✓	✓	✓	**1.5**	**75.2**
MSDNet [69]	✓			**1.2**	72.4
RANet [74]	✓		✓	1.3	73.7
EC-Dynamic (ours)	✓	✓	✓	1.3	**74.2**

TABLE 6.11: Static inference v.s. Dynamic inference in terms of runtime latency and accuracy. The latency is represented by the average inference latency. To fairly compare static inference and dynamic inference in different computing regimes, we use different models for static inference in different computing regimes. The models we have used for static inference have been shown in Table 6.3.

Method	Threshold	Xavier (ms)	Nano (ms)	CPU (ms)	Top1 Acc. (%)
EC-Static	N.A.	8.3	41.6	81.8	75.6
EC-Dynamic	0.1	**7.0**	**34.0**	**74.8**	**75.9**
EC-Static	N.A.	9.4	49.4	96.9	76.3
EC-Dynamic	0.2	**7.9**	**38.2**	**84.9**	**76.8**
EC-Static	N.A.	10.8	56.0	117.6	76.8
EC-Dynamic	0.3	**8.6**	**41.6**	**93.0**	**77.4**
EC-Static	N.A.	11.8	57.1	134.5	77.2
EC-Dynamic	0.4	**9.5**	**44.7**	**100.4**	**77.6**

Results on CIFAR-10 and CIFAR-100: To better demonstrate the efficacy of our approach on small-scale datasets, we conduct extensive experiments on both CIFAR-10 and CIFAR-100 datasets. The experimental results are shown in Table 6.12, which indicate that our approach has significant advantages over other existing methods on both datasets. For instance, our approach observes 2.09% higher top-1 accuracy with 41.13% less computation on CIFAR-10.

Robustness Analysis: To evaluate the robustness of the proposed framework, we perform experiments in two long-tail settings: 1) exponential and 2) step, and compare the results with the normal setting. The experiments are conducted on CIFAR-10 with an unbalancing factor of 0.5. The experimental results are shown

TABLE 6.12: Experimental results on CIFAR-10 and CIFAR-100. The baseline network is VGG16_BN. "CR" denotes the compression ratio.

Dataset	Method	MACs (M)	MACs CR (%)	Top1 Acc. (%)
CIFAR-10	VGG16_BN [42]	313.74	0.00	93.96
	GAL-0.1 [184]	171.89	45.21	90.73
	Hrank [53]	108.61	65.38	92.34
	EdgeCompress (ours)	**101.18**	**67.75**	**92.82**
	SSS [126]	183.13	41.63	93.02
	Zhao et al [204]	190.00	39.44	93.18
	Hrank [53]	145.61	53.59	93.43
	EdgeCompress (ours)	**120.55**	**61.58**	**93.64**
CIFAR-100	SSS [126]	223.13	28.89	71.08
	Zhao et al [204]	256.00	18.42	73.33
	EdgeCompress (ours)	**206.34**	**34.24**	**73.35**

TABLE 6.13: Results in different long-tail settings, where "Normal" denotes the results in the normal setting. The network utilized is VGG16_BN and the dataset is CIFAR-10.

Setting	Params (M)	MACs (M)	Top1 Acc. (%)
VGG16_BN [42]	14.98	313.74	93.96
Normal	5.39	101.18	92.82
Exponential	5.92	112.12	91.69
Step	5.79	109.33	91.86
Normal	6.59	126.14	93.54
Exponential	6.90	132.54	92.42
Step	6.71	128.53	92.43

in Table 6.13, where the minor accuracy degradation in long-tail settings validates the robustness of our framework.

Analytical Experiments: In this subsection, we show the impact of some important hyperparameters on the final performance of our framework.

The Architecture of The Foreground Predictor: The design of the foreground predictor can significantly affect the efficiency and accuracy of our framework. To validate the proposed lightweight foreground predictor, we conduct comparison experiments by integrating different advanced CNN into our framework as the foreground predictor. The experimental results are summarized in Table 6.14, where we observe that our design achieves the best trade-off between accuracy and efficiency. Even though some modern architectures can achieve slightly higher accuracy, they result in magnitudes higher model complexity and latency. For instance, EfficientNet-B1 only achieves a mere 0.05% accuracy improvement with 24.6× parameters and 6.3× MACs compared to our predictor, which significantly reduces the efficiency of the whole framework.

TABLE 6.14: Comparison with different foreground predictors in terms of classification accuracy and model efficiency.

Model	Params (M)	MACs (B)	Latency (ms)	Top1 Acc. (%)
ResNet18 [15]	11.23	1.81	3.01	75.59
ResNet34 [15]	21.33	3.66	5.19	75.50
RegNet-X_800MF [31]	6.65	0.80	4.50	75.50
RegNet-X_1.6GF [31]	8.37	1.60	7.26	75.61
EfficientNet-B0 [40]	4.13	0.38	4.31	75.57
EfficientNet-B1 [40]	6.64	0.57	6.21	**75.62**
Ours	**0.27**	**0.09**	**1.04**	75.57

The Number of Models: The number of models cascaded for dynamic inference is also crucial to the final performance of our framework. We perform comprehensive experiments to identify the optimal number of models from the perspective of accuracy, computational complexity, and actual inference latency. The experimental results in Figure 6.12 uncover an interesting insight that using too many models can worsen the trade-off between model efficiency and accuracy. Specifically, we observe that using two models for dynamic inference achieves the optimal trade-off between model efficiency and accuracy among all configurations. Meanwhile, the two-model configuration avoids loading too many models onto the device, optimizing the memory occupation of dynamic inference.

Ablation Study: Our framework contains three novel components: 1) Dynamic Image Cropping (DIC), 2) Compound Shrinking (CS), and 3) Dynamic Inference (DI). To validate the efficacy and efficiency of each component separately, we conduct ablation experiments on the ImageNet-1K dataset. The experimental results are shown in Table 6.15, where we observe that our DIC framework (i.e., EC-DIC) outperforms the ResizedCenterCrop strategy by a remarkable 1.5% accuracy improvement. Afterwards, we gradually integrate the CS module and DI strategy with DIC to build EC-Static and EC-Dynamic. The experimental results demonstrate that both EC-Static and EC-Dynamic further improve the accuracy, and the complete framework EC-Dynamic achieves the best accuracy. Thanks to the novel design of our framework, a significant improvement in accuracy is achieved at a slightly higher latency cost, optimizing the trade-off between accuracy and execution efficiency. The ablation experiments reveal that all DIC, CS, and DI components contribute to the final performance.

Foreground Prediction Visualization: We visualize the bounding boxes generated from both Grad-CAM and our predictor in Figure 6.13. We can see that the foreground of most images only occupies part of the whole image, thus performing

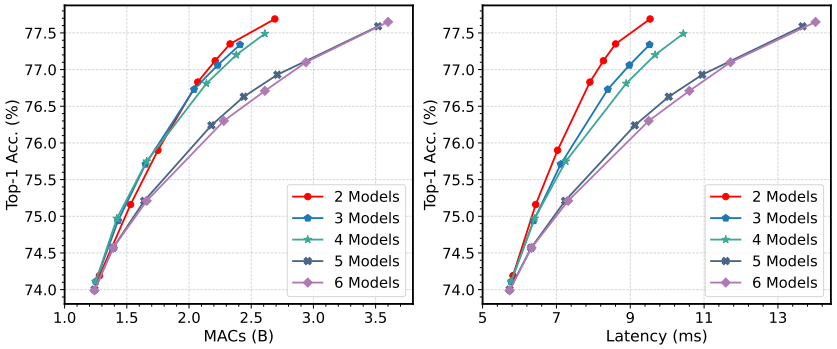


FIGURE 6.12: Impact of the number of models used for dynamic inference on the computational complexity and on-device latency. The latency is quantified as the average latency of all images on Jetson Xavier.

TABLE 6.15: Results of ablation experiments. The baseline network is ResNet50 and the target dataset is ImageNet-1K.

Method	MACs (B)	Latency (ms)	Top1 Acc. (%)
ResNet50	4.1	10.6	76.0
ResizedCenterCrop (RCC)	1.8	**6.3**	73.4
EC-DIC (DIC only)	1.9	7.3	74.9
EC-Static (DIC + CS)	1.8	8.3	75.6
EC-Dynamic (DIC + CS + DI)	**1.8**	7.0	**75.9**

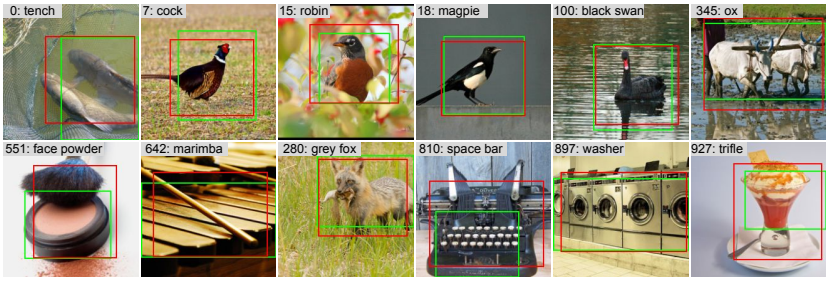


FIGURE 6.13: Visualization of the predicted bounding boxes (red) and the ground truth bounding boxes generated from Grad-CAM (green). Our predictor achieves a high localization accuracy of 62.1% mAP on ImageNet-1K validation set. The images above are randomly selected from ImageNet-1K.

inference on the whole image is unnecessary and inefficient, which coincides with our motivation. Moreover, Figure 6.13 validates that, even though the lightweight foreground predictor only has very limited computation and parameters, it can still accurately and efficiently localize the main object. Due to the design of the efficient foreground predictor, we can remove the spatial redundancy in images, accelerating CNNs on edge devices.

Visualization of Easy Samples & Hard Samples: We visualize some easy samples and hard samples from ImageNet-1K to more intuitively demonstrate the

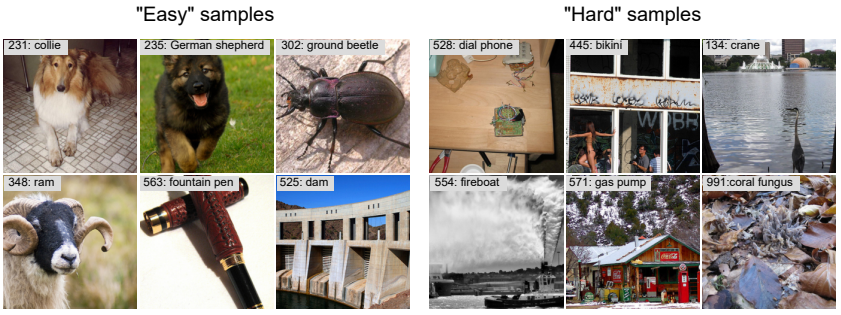


FIGURE 6.14: Visualization of some hard samples and easy samples. Hard samples are considered as the images that cannot be confidently classified by the first model, and easy samples refer to those images that can be confidently classified by the first model. All images are from ImageNet-1K.

difference between them. The visualization results in 6.14 indicate that the most of easy images have a simple and clear foreground, and thus they can be correctly recognized by a small model. For hard samples, the images are more confusing because of their unintuitive foreground, and larger models are needed to mine high-level semantics in images for the correct prediction. Through the proposed dynamic inference framework, images with different difficulties can be processed by the appropriate model, achieving higher run-time efficiency and accuracy.

6.7 Comparison with Our Previous Methods

Chapter 5 and Chapter 6 focus on optimizing the inference efficiency of DNN models by different compression techniques. Specifically, Chapter 5 utilizes fine-grained multi-dimensional model pruning to optimize the network architecture and input data, while Chapter 6 further introduces multi-scale dynamic inference to adaptively process different input data for better trade-offs between efficiency and accuracy. We have conducted comprehensive experiments to compare our methods with other state-of-the-art approaches separately. However, different methods our ours have not been compared with each other yet. Actually, since model pruning and dynamic inference share the same target, our methods can be compared with each other as long as the baseline network, dataset, and deployment hardware are aligned. Therefore, in this section, we perform an inter-chapter evaluation to demonstrate the advantages of different methods. The experimental results are shown in Table 6.16, where we observe that TECO and TICO achieve slightly

TABLE 6.16: Comparison with our previous methods. The target dataset is ImageNet and the latency is measured on Jetson Xavier.

Method	MACs (B)	Latency (ms)	Top1 Acc. (%)	Top5 Acc. (%)
ResNet50	4.1	10.6	76.8	93.4
TECO-S	1.1	3.4	73.1	91.2
TICO-S	1.0	3.5	73.0	91.2
EC-Dynamic	1.3	5.8	74.2	91.8
TECO-M	2.2	6.6	75.7	92.8
TICO-M	2.2	6.5	75.8	92.9
EC-Dynamic	1.8	7.1	75.9	92.5
TECO-L	2.9	7.7	76.3	93.2
TICO-L	2.8	7.9	76.6	93.4
EC-Dynamic	2.3	8.6	77.4	93.2

better latency than EdgeCompress. This is because TICO and TECO are one-stage optimization frameworks, in which only the classification models are executed. In contrast, EdgeCompress is a two-stage optimization framework, where both the foreground prediction network and the main classification model will be executed. Therefore, the longer pipeline of EdgeCompress brings additional time overhead. But EdgeCompress also improves the accuracy significantly and achieves a better trade-off between MACs and accuracy.

6.8 Summary

In this chapter, we propose EdgeCompress, a comprehensive CNN compression framework to reduce the computational redundancy in both input images and network architectures, facilitating the deployment of advanced CNNs onto embedded devices. In EdgeCompress, we first introduce dynamic image cropping, which effectively and efficiently removes the redundancy in input images. Subsequently, we present compound shrinking to collaboratively compress the three dimensions of CNNs, reducing the computational redundancy in both input images and network architectures. Finally, we design a dynamic inference strategy, which adaptively execute different models for different input images, further improving the inference efficiency of CNNs. Extensive experiments validate the advantages of EdgeCompress over existing SOTA approaches.

Chapter 7

Conclusion

7.1 Conclusion

In the era of artificial intelligence, edge intelligence effectively mitigates the network and privacy issues of conventional cloud intelligence, facilitating the deployment of deep learning models in some safety-critical and latency-critical applications like autonomous driving. However, in order to adapt to various edge environments, there are a large number of edge deep learning hardware devices with distinct resources being proposed. Since edge deep learning devices are resource-sensitive, failing to effectively utilize the hardware resources may result in unsatisfactory execution latency and accuracy. To address this issue, in this thesis, we propose multiple approaches to efficiently and adaptatively adjust the overhead of DNNs according to the available hardware resources, thereby achieving a better trade-off between execution efficiency and prediction accuracy.

In Chapter 3, we propose an end-to-end hardware performance benchmark (EDLAB) to efficiently evaluate various edge deep learning accelerators. In EDLAB, we design two types of deep learning workloads: 1) real DNN models and 2) parameterized models. Real models can efficiently evaluate the actual performance and resource consumption of edge deep learning accelerators when hosting existing DNN models, while parameterized models are utilized to evaluate the peak performance of different devices and facilitate the hardware-efficient design of DNNs. To efficiently deploy the designed deep learning workloads onto various heterogeneous edge accelerators, we also introduce a workload pre-processing framework, where

the development toolchains of different devices are integrated for easy deployment, and the hyperparameters are controlled uniformly to ensure the fairness of benchmarking. Finally, we conduct evaluation experiments with EDLAB on multiple edge deep learning accelerators and analyze their performance and resource consumption. In our following research works, EDLAB serves as a hardware evaluation tool to facilitate the hardware-aware DNN design.

In Chapter 4, we propose two model scaling frameworks: 1) HACScale (Chapter 4.1) and 2) AdaptScale (Chapter 4.2), to efficiently enlarge the model size for higher accuracy on powerful devices with redundant resources. In HACScale, we investigate the impact of scaling different dimensions on the execution efficiency and resource consumption of DNNs. Subsequently, we propose a hardware-aware model scaling algorithm to efficiently couple the scaling of different dimensions for higher accuracy while not compromising execution efficiency. Extensive experiments on multiple benchmarks demonstrate the efficiency and efficacy of our approach. In AdaptScale, we further find that the optimal scaling strategy of different models is distinct from each other, and sharing the same scaling strategy among different models may lead to a significant accuracy drop. Therefore, we propose an adaptive model scaling framework to efficiently customize the best scaling strategy for different models. We model the balance among different dimensions of DNNs and efficiently search for the optimal scaling strategy that can achieve the best model balance with multi-objective evolutionary search. Finally, we perform experiments on multiple models, tasks, and datasets, and the experimental results validate the advantages of our approach.

In Chapter 5, we propose two model pruning frameworks: 1) TECO (Chapter 5.1) and 2) TICO (Chapter 5.2), to efficiently reduce the computational complexity and resource consumption of DNNs for higher execution efficiency on resource-constrained edge devices. In TECO, we propose a fine-grained multi-dimensional pruning framework, where we first introduce inner-dimensional importance evaluation and inter-dimensional importance evaluation to efficiently and comprehensively identify redundant units in DNNs. In addition, we also introduce a heuristic pruning algorithm to gradually search for the optimal tiny architecture. Experiments show the advantages of our approach over existing methods. In TICO, we further consider both the training efficiency and inference efficiency of DNNs, and propose a training & inference co-optimization framework. The co-optimization

framework contains a multi-objective pruning algorithm and a resolution-adaptive training strategy. The multi-objective pruning algorithm utilizes multi-objective evolutionary optimization to find the optimal pruning decision, effectively optimizing both the training and inference efficiency. Further, the resolution-adaptive training strategy utilizes different image sizes at different training stages, which further reduces the training overhead while not sacrificing accuracy. Extensive experiments show the efficiency and efficacy of our method.

In Chapter 6, we propose EdgeCompress, a dynamic inference framework of DNNs to optimize the trade-off between execution efficiency and accuracy at runtime. In EdgeCompress, we first introduce a dynamic image cropping algorithm, where different images are cropped dynamically to remove the redundant background pixels and preserve the most informative foreground objective. In addition, we also introduce a compound shrinking algorithm to efficiently compress the three dimensions of DNNs and generate multiple models with different complexities. Finally, we design a dynamic inference algorithm, where different input images with distinct recognition difficulties will be processed by different models to achieve the best trade-off between accuracy and efficiency. Experimental results indicate that our approach outperforms existing related methods.

On the one hand, HACScale and AdaptScale focus on model scaling, which enables us to efficiently improve the capacity of DNN models for powerful devices, such that we can more efficiently utilize the underlying hardware resources to achieve higher accuracy while not sacrificing execution efficiency. On the other hand, TECO and TICO focus on model compression, which provides solutions to compress DNN models for resource-constrained edge devices with minimum accuracy loss. Both model scaling and model compression can be considered part of model adaptation, and they can work complementaryly to generate models with different trade-offs between accuracy and efficiency for a variety of emerging AI devices and applications. Besides optimizing the model itself, the proposed EdgeCompress also strives to optimize the inference mechanism to further improve the efficiency of DNNs, which can collaborate with our model adaptation approaches. For example, the model adaptation methods generate models with different costs and accuracy, and then EdgeCompress efficiently schedules the generated models for efficient and adaptive inference. In addition, the EDLAB framework enables us to efficiently evaluate our models on different hardware platforms and obtain feedback, which

effectively reduces the design cost of our model optimizing methods and accelerates the design process. In summary, our works establish a large adaptation framework for DNN models, which enhances the flexibility, adaptability, and applicability of DNN models for different hardware platforms in edge computing.

7.2 Future Directions

In this section, we briefly discuss several directions for our future research.

7.2.1 Hardware-Efficient Large Model Design

Large language models (LLM) like GPT-3 [6] and LLaMa [216] are becoming increasingly popular in the research community of deep learning, which has made significant breakthroughs in multiple tasks. However, such impressive performance comes at the cost of higher execution overhead, which hinders the deployment of large models. To more efficiently build large language models with Transformer, openAI proposes the scaling law [217], which explores the relationship between model performance, model size, data size, and training costs, and then summarizes some rules for scaling up Transformer to large language models. When large language models are deployed onto cloud datacenters for service, they will be facing billions of inference requests every day, which can put a super huge pressure on hardware. However, currently, the inference efficiency of large language models is not as valued as training costs. To mitigate this issue, one big direction of our future research will be optimizing the inference efficiency of large language models. Model scaling, as one of the main approaches to increase the capability of DNNs, has been widely utilized in large model designs. However, existing model scaling approaches mainly consider the scores on various benchmarks while ignoring the execution efficiency on hardware. A potential improvement is to take into consideration the underlying hardware and integrate the execution efficiency on the target platform into the optimization objectives of scaling. In this way, the obtained model may achieve a better trade-off between efficiency and accuracy. Meanwhile, after the model is designed and trained, we can also perform post-optimization to further optimize the efficiency. For example, we can design hardware-efficient

pruning or quantization techniques to remove redundant parameters, and by doing so, we can alleviate the computing and memory pressure of large language models.

7.2.2 Robust Adaptive Neural Networks

Our works have demonstrated that model scaling and model pruning are able to flexibly adapt the overhead of DNNs for various edge devices with distinct capabilities, thereby constructing efficient edge AI systems. Specifically, for different DNN models, we inspect the importance of different dimensions and selectively add or remove units accordingly. However, in our previous works, we mainly consider the execution performance on hardware and models' prediction accuracy on different datasets. One key aspect we fail to consider is the robustness of DNNs. Actually, In edge AI systems, application security is as important as latency and accuracy. In some security-critical edge applications, such as home smart cameras and autonomous driving, if the model is attacked, it will result in catastrophic consequences. In addition, after a model is designed and optimized, it may be deployed in different environments and faced with different input data. These data may not be independent and identically distributed (IID), such as long-tailed distributed data, which may cause significant performance degradation. Therefore, another possible direction of our future research may be designing robust model adaptation algorithms, so that we can flexibly adapt DNN models for different hardware devices while improving the robustness of our adapted models.

7.2.3 Efficiency of Model Pruning and Scaling

In our model pruning and model scaling frameworks, we have exploited both search-based and heuristic approaches to optimize the model. Both approaches have relatively high optimization costs. Specifically, the search-based approach faces a large search space and has to bear enormous search costs, such as the evaluation cost of searched candidates during the search process. For the heuristic approach, we have to perform the optimization iteratively. the more optimization iterations we perform, the larger the overall optimization costs. In addition, the evaluation cost of each optimization iteration is also not negligible. Moreover, our model optimization solutions are usually hardware-aware and model-aware, which means the

solution designed for one model or one hardware device may only achieve an unsatisfactory performance when applied to other models and devices. Therefore, in our future works, we may focus on improving the designing efficiency of model pruning and model scaling, such that we can efficiently customize the optimal strategy for different models and hardware platforms for better efficiency and accuracy.

List of Author's Awards, Patents, and Publications[1]

Awards

- **DAC Young Fellow Award**, *The 60th Design Automation Conference, San Francisco, 2023.*

Technical Disclosures

- Weichen Liu, **Hao Kong**, Shuo Huai, and Ravi Subramaniam, "A Multi-Dimensional Pruning Framework Based on Double Evaluation Mechanism", *NTU-Ref 2022-410*, 2022.
- Weichen Liu, **Hao Kong**, Shuo Huai, and Ravi Subramaniam, "A Collaborative Optimization Framework for Edge Training and Inference Based on Evolutionary Algorithms", *NTU Ref 2022-409*, 2022.
- Weichen Liu, **Hao Kong**, Shuo Huai, and Ravi Subramaniam, "Smart Scissor: A Deep Compression Framework For Jointly Reducing the Redundancy In Images And Neural Networks", *NTU-Ref 2022-288*, 2022.
- Weichen Liu, Di Liu, **Hao Kong**, Lei Zhang, Shuo Huai, Shiqing Li, Hui Chen, and Shien Zhu, "EDLAB: A Benchmark Tool for Edge Deep Learning Accelerators", *NTU-Ref 2020-264*, 2020.

[1]The superscript * indicates joint first authors

Journal Articles

- **Hao Kong**, Di Liu, Shuo Huai, Xiangzhong Luo, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu, "EdgeCompress: Coupling Multi-Dimensional Model Compression and Dynamic Inference for EdgeAI", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (**TCAD**), 2023.

- **Hao Kong**, Shuo Huai, Di Liu, Lei Zhang, Hui Chen, Shien Zhu, Shiqing Li, Weichen Liu, Manu Rastogi, Ravi Subramaniam, Madhu Athreya, and M. Anthony Lewis, "EDLAB: A Benchmark for Edge Deep Learning Accelerators", *IEEE Design & Test* (**D&T**), 2021.

- Di Liu*, **Hao Kong***, Xiangzhong Luo*, Weichen Liu, and Ravi Subramaniam, "Bringing AI To Edge: From Deep Learning's Perspective", **Neurocomputing**, 2021.

- Xiangzhong Luo, Di Liu, **Hao Kong**, Shuo Huai, Hui Chen, and Weichen Liu, "LightNAS: On Lightweight and Scalable Neural Architecture Search for Embedded Platforms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (**TCAD**), 2023.

- Xiangzhong Luo, Di Liu, **Hao Kong**, Shuo Huai, Hui Chen, and Weichen Liu, "SurgeNAS: A Comprehensive Surgery on Hardware-Aware Differentiable Neural Architecture Search", *IEEE Transactions on Computers* (**TC**), 2023.

- Shuo Huai, Di Liu, **Hao Kong**, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin, "Latency-Constrained DNN Architecture Learning for Edge Systems Using Zerorized Batch Normalization", *Future Generation Computer Systems* (**FGCS**), 2023.

- Xiangzhong Luo, Di Liu, Shuo Huai, **Hao Kong**, Hui Chen, and Weichen Liu, "Designing Efficient DNNs via Hardware-Aware Neural Architecture Search and Beyond", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (**TCAD**), 2021.

Conference Proceedings

- **Hao Kong**, Di Liu, Xiangzhong Luo, Shuo Huai, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu, "Towards Efficient Convolutional Neural Network for Embedded Hardware via Multi-Dimensional Pruning", *ACM/ IEEE Design Automation Conference* (**DAC**), 2023.

- **Hao Kong**, Xiangzhong Luo, Shuo Huai, Di Liu, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu, "EMNAPE: Efficient Multi-Dimensional Neural Architecture Pruning for EdgeAI", *ACM/IEEE Design, Automation and Test in Europe* (**DATE**), 2023.

- **Hao Kong**, Di Liu, Shuo Huai, Xiangzhong Luo, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin, "Smart Scissor: Coupling Spatial Redundancy Reduction and CNN Compression for Embedded Hardware", *ACM/IEEE International Conference on Computer-Aided Design* (**ICCAD**), 2022.

- **Hao Kong**, Di Liu, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam, "HACScale: Hardware-Aware Compound Scaling for Resource-Efficient DNNs", *ACM/IEEE Asia and South Pacific Design Automation Conference* (**ASP-DAC**), 2022.

- Xiangzhong Luo, Di Liu, **Hao Kong**, Shuo Huai, Hui Chen, and Weichen Liu, "You Only Search Once: on Lightweight Differentiable Architecture Search for Resource-Constrained Embedded Platforms", *ACM/IEEE Design Automation Conference* (**DAC**), 2022.

- Shuo Huai, Di Liu, **Hao Kong**, Xiangzhong Luo, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin, "Collate: Collaborative Neural Network Learning for Latency-Critical Edge Systems", *IEEE 40th International Conference on Computer Design* (**ICCD**), 2022.

- Xiangzhong Luo, Di Liu, **Hao Kong**, and Weichen Liu, "EdgeNAS: Discovering Efficient Neural Architectures for Edge Systems", *IEEE 38th International Conference on Computer Design* (**ICCD**), 2020.

Bibliography

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. xxvii, 1, 3, 27, 48, 111, 119, 144

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, pages 740–755, 2014. xxvii, 1, 48, 133

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, 2012. 1, 30, 36, 114

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1, 44

[5] Google. Google image. `https://images.google.com/`, 2023. 1

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020. 1, 3, 160

[7] Jinzhi Liao, Xiang Zhao, Xinyi Li, Lingling Zhang, and Jiuyang Tang. Learning discriminative neural representations for event detection. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 644–653, 2021. 1

[8] Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. Cost-sensitive regularization for label confusion-aware event detection. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5278–5283, 2019. 1

[9] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. A multitask,

multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023. 1

[10] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020. 2

[11] Huang Yao, Rongjun Qin, and Xiaoyu Chen. Unmanned aerial vehicle for remote sensing applications—a review. *Remote Sensing*, 11(12):1443, 2019. 2

[12] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016. 2, 3, 44, 86

[13] Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing ai to edge: From deep learning's perspective. *Neurocomputing*, 485: 297–320, 2022. 2, 10, 19, 25, 86

[14] NVIDIA. Nvidia jetson agx orin. `https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/jetson-orin/`, 2023. 2, 3, 4

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 2, 5, 8, 22, 26, 35, 36, 38, 40, 44, 68, 69, 82, 86, 95, 97, 98, 114, 119, 133, 144, 149, 151, 153

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141, 2018. 3, 21, 81

[17] NVIDIA. Jetson agx xavier. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/`, 2023. 3, 25, 35

[18] Google. Edge tensor processing unit (tpu). `https://cloud.google.com/edge-tpu`, 2023. 3, 4

[19] NVIDIA. Nvidia jetson tx2. `https://developer.nvidia.com/embedded/jetson-tx2`, 2023. 3, 23, 35

[20] NVIDIA. Nvidia jetson nano. `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`, 2023. 3, 4, 23, 35

[21] Intel. Interl neural compute stick. `https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html`, 2023. 4, 23

[22] Raspberry Foundation. Raspberry pi. `https://www.raspberrypi.org/`, 2023. 4

[23] NVIDIA. Tensorrt sdk. `https://developer.nvidia.com/tensorrt`, 2023. 4

[24] Google. Tensorflow lite. `https://www.tensorflow.org/lite`, 2023. 4

[25] Intel. Openvino. `https://docs.openvino.ai/2023.0/home.html`, 2023. 4

[26] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. 5, 26

[27] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 5, 26

[28] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 5, 12, 27, 36, 41, 60, 64, 95, 114, 149

[30] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2752–2761, 2018.

[31] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10428–10436, 2020. 5, 7, 77, 81, 108, 114, 144, 153

[32] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2820–2828, 2019. 6, 12, 27, 36, 49, 60

[33] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 6, 40

[34] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017. 6, 27

[35] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018. 6

[36] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019. 6, 108

[37] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 4780–4789, 2019. 6, 81

[38] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. 6

[39] Sathwika Bavikadi, Abhijitt Dhavlle, Amlan Ganguly, Anand Haridass, Hagar Hendy, Cory Merkel, Vijay Janapa Reddi, Purab Ranjan Sutradhar, Arun Joseph, and Sai Manoj Pudukotai Dinakarrao. A survey on machine learning accelerators and evolutionary hardware platforms. *IEEE Design & Test*, 39 (3):91–116, 2022. 7

[40] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, pages 6105–6114. PMLR, 2019. 7, 9, 28, 36, 60, 64, 68, 69, 70, 72, 75, 79, 80, 81, 82, 135, 153

[41] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016. 7, 8, 22, 26, 27, 45, 60, 64, 68, 70, 149

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 7, 8, 26, 30, 68, 80, 114, 144, 152

[43] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. 7, 8, 21

[44] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure

learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1586–1595, 2018. 8, 9, 11, 27, 60

[45] Eugene Lee and Chen-Yi Lee. Neuralscale: Efficient scaling of neurons for resource-constrained deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1478–1487, 2020. 8, 27, 60, 68, 70

[46] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 924–932, 2021. 7, 8, 9, 28, 60, 68, 79, 80, 81, 82, 102, 109

[47] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 8, 26, 36, 149, 150

[48] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2736–2746, 2022. 8

[49] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. 8, 27, 44, 145, 149, 150

[50] Sheng Li, Mingxing Tan, Ruoming Pang, Andrew Li, Liqun Cheng, Quoc V Le, and Norman P Jouppi. Searching for fast model families on datacenter accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8085–8095, 2021. 8, 9, 28, 72, 81

[51] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11976–11986, 2022. 8, 9, 81

[52] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11264–11272, 2019. 9, 11, 27, 64, 65, 86, 89, 92, 95, 104, 120

[53] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision*

and Pattern Recognition (CVPR), pages 1529–1538, 2020. 9, 11, 33, 64, 86, 88, 89, 95, 96, 97, 98, 104, 116, 117, 120, 121, 130, 149, 152

[54] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020. 10

[55] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. 10, 30, 31, 35, 39

[56] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1135–1143, 2015. 10, 30

[57] Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 2498–2507. PMLR, 2017. 30, 31

[58] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers (TC)*, 68(10):1487–1497, 2019. 10, 30, 31

[59] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *The European Conference on Computer Vision (ECCV)*, September 2018. 10, 30, 31

[60] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016. 11, 31

[61] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019. 11, 33, 116, 121

[62] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 32, 33

[63] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406. IEEE Computer Society, 2017. 11, 32, 33

[64] Wenxiao Wang, Shuai Zhao, Minghao Chen, Jinming Hu, Deng Cai, and Haifeng Liu. Dbp: Discrimination based block-level pruning for deep model acceleration. *arXiv preprint arXiv:1912.10178*, 2019. 11, 86, 88, 89, 95, 104, 105

[65] Mingjian Zhu, Kai Han, Enhua Wu, Qiulin Zhang, Ying Nie, Zhenzhong Lan, and Yunhe Wang. Dynamic resolution network. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:27319–27330, 2021. 12, 13, 37, 40, 41, 86, 93, 95, 104, 128, 137, 149, 151

[66] Yulin Wang, Kangchen Lv, Rui Huang, Shiji Song, Le Yang, and Gao Huang. Glance and focus: a dynamic approach to reducing spatial redundancy in image classification. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:2432–2444, 2020. 12, 13, 37, 41, 105, 128

[67] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(11):7436–7456, 2021. 12, 36, 37

[68] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016. 13, 37, 38, 141

[69] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018. 13, 37, 38, 139, 151

[70] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8607–8617, 2021. 40

[71] Xin Wang, Fisher Yu, Lisa Dunlap, Yi-An Ma, Ruth Wang, Azalia Mirhoseini, Trevor Darrell, and Joseph E Gonzalez. Deep mixture of experts via shallow embedding. In *Uncertainty in Artificial Intelligence*, pages 552–562. PMLR, 2020. 14, 40

[72] Shaofeng Cai, Yao Shu, Wei Wang, and Beng Chin Ooi. Dynamic routing networks, 2020. 14, 40

[73] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. 13, 37, 38, 39, 151

[74] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2369–2378, 2020. 13, 40, 41, 86, 88, 95, 137, 141, 149, 151

[75] Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. Big/little deep neural network for ultra low power inference. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 124–132. IEEE, 2015. 38

[76] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. *arXiv preprint arXiv:1706.00885*, 2017. 13, 38

[77] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. 13, 37, 38

[78] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018. 37, 38, 39, 151

[79] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1039–1048, 2017. 38

[80] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019. 13, 38

[81] Kyunghyun Cho and Yoshua Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*, 2014. 13, 39

[82] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[83] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *International Conference on Learning Representations (ICLR) Workshop*, 2015. 13, 39

[84] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. 14, 37, 39

[85] Zhihang Yuan, Bingzhe Wu, Guangyu Sun, Zheng Liang, Shiwan Zhao, and Weichen Bi. S2dnas: Transforming static cnn model for dynamic inference via neural architecture search. In *European Conference on Computer Vision (ECCV)*, pages 175–192, 2020. 14, 40

[86] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2020. 19

[87] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Proceedings of the 2020 Empirical Methods in Natural Language Processing (EMNLP)*, pages 4163–4174, 2020. 22

[88] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2158–2170, 2020. 19

[89] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(12):4338–4364, 2020. 19

[90] Krishna Teja Chitty-Venkata, Sparsh Mittal, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. A survey of techniques for optimizing transformer inference, 2023. 19

[91] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2011. 21

[92] Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 342–347. IEEE, 2011. 21

[93] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. 21

[94] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019. 22

[95] Xiangzhong Luo, HK Luan Duong, and Weichen Liu. Person re-identification via pose-aware multi-semantic learning. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020. 22

[96] Alexander Womg, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. In *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 95–101. IEEE, 2018. 22

[97] Jun Wang, Tanner A. Bohn, and Charles X. Ling. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1967–1976, 2018. 22

[98] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2020. 22

[99] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020. 23, 24, 44

[100] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *Proceedings of Machine Learning and Systems (MLSys)*, 2:336–349, 2020. 23, 24, 105

[101] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. In *31th Conference on Neural Information Processing Systems (NeurIPS) Workshop*, 2017. 24, 25, 44

[102] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018. 24, 44

[103] Wei Zhang, Wei Wei, Lingjie Xu, Lingling Jin, and Cheng Li. Ai matrix: A deep learning benchmark for alibaba data centers, 2019. 25

[104] Baidu Inc. Deepbench: Benchmarking deep learning operations on different hardware. `https://github.com/baidu-research/DeepBench`, 2023. 23, 24, 25

[105] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC)*, pages 199–216, 2022. 26

[106] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500, 2017. 26

[107] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. 27

[108] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015. 27, 133

[109] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017. 27

[110] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–420, 2018. 27

[111] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1925–1934, 2017. 27

[112] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2148–2156, 2013. 28, 29

[113] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 598–605, 1990. 30

[114] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks (ICNN)*, pages 293–299. IEEE, 1993. 30

[115] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011. 30

[116] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5687–5695, 2017. 30, 31

[117] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016. 31

[118] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 32

[119] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001. 32

[120] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017. 32, 33

[121] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016. 33, 86

[122] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5068–5076. IEEE Computer Society, 2017. 33

[123] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2017. 33, 34

[124] Jing Liu, Bohan Zhuang, Zhuangwei Zhuang, Yong Guo, Junzhou Huang, Jinhui Zhu, and Mingkui Tan. Discrimination-aware network pruning for deep model compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(8):4035–4051, 2021. 33, 120

[125] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2234–2240. ijcai.org, 2018. 33, 34

[126] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Vomputer Vision (ECCV)*, pages 304–320, 2018. 33, 34, 86, 95, 96, 116, 120, 149, 152

[127] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *The European Conference on Computer Vision (ECCV)*, September 2018. 33, 34

[128] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018. 33, 34, 64

[129] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2130–2141, 2019. 33

[130] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2006–2015. IEEE, 2020. 33, 34

[131] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017. 34

[132] Jiecao Yu, Andrew Lukefahr, David J. Palframan, Ganesh S. Dasika, Reetuparna Das, and Scott A. Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 548–560, 2017. 34

[133] Sambhav R Jain, Albert Gural, Michael Wu, and Chris Dick. Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware. *arXiv preprint arXiv:1903.08066*, 6(6):3, 2019. 35

[134] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[135] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018.

[136] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[137] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015. 35

[138] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning, (ICML)*, volume 37, pages 1737–1746, 2015. 35

[139] William J Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. *Communications of the ACM*, 63(7):48–57, 2020. 35

[140] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 535–541, 2006. 35

[141] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 35

[142] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1195–1204, 2017. 35

[143] Yuang Liu, Wei Zhang, and Jun Wang. Adaptive multi-teacher multi-level knowledge distillation. *Neurocomputing*, 415:106–113, 2020. 35

[144] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018. 36

[145] Sujith Ravi. Efficient on-device models using neural projections. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pages 5370–5379, 2019. 36

[146] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. Few sample knowledge distillation for efficient network compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14627–14635, 2020. doi: 10.1109/CVPR42600.2020.01465. 36

[147] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015. 36

[148] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017. 36

[149] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017. 37, 40

[150] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning (ICML)*, pages 527–536. PMLR, 2017. 37, 38

[151] Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. The cascading neural network: building the internet of smart things. *Knowledge and Information Systems*, 52:791–814, 2017. 38

[152] Jiaqi Guan, Yang Liu, Qiang Liu, and Jian Peng. Energy-efficient amortized inference with cascaded deep classifiers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2184–2190, 2018.

[153] Xin Dai, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit with flexible multi-branch network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 235–244, 2020. 38

[154] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017. 38

[155] Qiushan Guo, Zhipeng Yu, Yichao Wu, Ding Liang, Haoyu Qin, and Junjie Yan. Dynamic recursive neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5147–5156, 2019. 38, 39

[156] Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013. 39

[157] Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. Channel selection using gumbel softmax. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 241–257. Springer, 2020. 40, 151

[158] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. In *International Conference on Learning Representations (ICLR)*, 2020. 40

[159] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013. 40

[160] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1930–1939, 2018. 40

[161] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8080–8089, 2018. 40

[162] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. 40

[163] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022. 40

[164] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, 2017. 40, 68

[165] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2019. 40

[166] Zekun Hao, Yu Liu, Hongwei Qin, Junjie Yan, Xiu Li, and Xiaolin Hu. Scale-aware face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6186–6195, 2017. 41

[167] Huiyu Wang, Aniruddha Kembhavi, Ali Farhadi, Alan L Yuille, and Mohammad Rastegari. Elastic: Improving cnns with dynamic scaling policies. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2258–2267, 2019. 41

[168] Adria Recasens, Petr Kellnhofer, Simon Stent, Wojciech Matusik, and Antonio Torralba. Learning to zoom: a saliency-based sampling layer for neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 51–66, 2018. 41

[169] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014. 41

[170] Zhichao Li, Yi Yang, Xiao Liu, Feng Zhou, Shilei Wen, and Wei Xu. Dynamic computational time for visual attention. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, pages 1199–1209, 2017. 41

[171] Amir Rosenfeld and Shimon Ullman. Visual concept recognition and localization via iterative introspection. In *13th Asian Conference on Computer Vision (ACCV)*, pages 264–279. Springer, 2017. 41

[172] Jianlong Fu, Heliang Zheng, and Tao Mei. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4438–4446, 2017. 41

[173] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016. 41, 82, 98

[174] Hao Kong, Shuo Huai, Di Liu, Lei Zhang, Hui Chen, Shien Zhu, Shiqing Li, Weichen Liu, Manu Rastogi, Ravi Subramaniam, et al. Edlab: A benchmark for edge deep learning accelerators. *IEEE Design and Test*, 2021. 43, 86

[175] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. 45

[176] Hao Kong, Di Liu, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Hacscale: Hardware-aware compound scaling for resource-efficient dnns. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 708–713. IEEE, 2022. 59, 105

[177] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021. 60, 102, 111, 112, 123

[178] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. 78, 107

[179] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, pages 10347–10357, 2021. 81

[180] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021. 81

[181] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *The 14th European Conference on Computer Vision (ECCV)*, pages 21–37, 2016. 81

[182] Hao Kong, Xiangzhong Luo, Shuo Huai, Di Liu, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu. Emnape: Efficient multi-dimensional neural architecture pruning for edgeai. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023. 85

[183] Yunhe Wang, Chang Xu, Chunjing Xu, Chao Xu, and Dacheng Tao. Learning versatile filters for efficient convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018. 86, 149

[184] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2799, 2019. 86, 88, 95, 96, 98, 116, 117, 120, 149, 152

[185] Manoj Alwani, Yang Wang, and Vashisht Madhavan. Decore: Deep compression with reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12349–12359, 2022. 95, 96, 97, 98, 116, 119, 120, 121

[186] Shixing Yu, Zhewei Yao, Amir Gholami, Zhen Dong, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. Hessian-aware pruning and optimal neural implant. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3880–3891, 2022. 95, 120

[187] Souvik Kundu, Mahdi Nazemi, Peter A Beerel, and Massoud Pedram. Dnr: A tunable robust pruning framework through dynamic network rewiring of dnns. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 344–350, 2021.

[188] Kai Huang, Siang Chen, Bowen Li, Luc Claesen, Hao Yao, Junjian Chen, Xiaowen Jiang, Zhili Liu, and Dongliang Xiong. Acceleration-aware fine-grained channel pruning for deep neural networks via residual gating. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 41(6):1902–1915, 2021. 86

[189] Hao Kong, Di Liu, Shuo Huai, Xiangzhong Luo, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin. Smart scissor: Coupling spatial redundancy reduction and cnn compression for embedded hardware. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9, 2022. 86, 104, 105, 109

[190] Shuo Huai, Lei Zhang, Di Liu, Weichen Liu, and Ravi Subramaniam. Zerobn: Learning compact neural networks for latency-critical edge systems. In *Proceedings of 58th ACM/IEEE Design Automation Conference (DAC)*, pages 151–156, 2021. 94, 105, 108

[191] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4943–4953, 2019. 95, 149

[192] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations (ICLR)*, 2020. 95, 149

[193] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017. 95, 97

[194] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Transactions on Internet of Things*, 2(3):1–49, 2021. 101

[195] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. 102, 111, 112, 123

[196] Elad Hoffer, Berry Weinstein, Itay Hubara, Tal Ben-Nun, Torsten Hoefler, and Daniel Soudry. Mix & match: training convnets with mixed image sizes for improved accuracy, speed and scale resiliency. *arXiv preprint arXiv:1908.08986*, 2019. 102, 111, 112

[197] Jinsu Lee and Hoi-Jun Yoo. An overview of energy-efficient hardware accelerators for on-device deep-neural-network training. *IEEE Open Journal of the Solid-State Circuits Society*, 1:115–128, 2021. 105

[198] David Ojika, Bhavesh Patel, G Anthony Reina, Trent Boyer, Chad Martin, and Prashant Shah. Addressing the memory bottleneck in ai model training. *arXiv preprint arXiv:2003.08732*, 2020. 105

[199] Shuo Huai, Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, Ravi Subramaniam, Christian Makaya, and Qian Lin. Collate: Collaborative neural network learning for latency-critical edge systems. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 627–634. IEEE, 2022. 105

[200] Xiangzhong Luo, Di Liu, Shuo Huai, Hao Kong, Hui Chen, and Weichen Liu. Designing efficient dnns via hardware-aware neural architecture search and beyond. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 41:1799–1812, 2021. 108

[201] Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Haifeng Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In *International Conference on Machine Learning (ICML)*, pages 10717–10726. PMLR, 2021. 108

[202] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 114

[203] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017. 116

[204] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2780–2789, 2019. 116, 117, 121, 152

[205] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 118

[206] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 107: 107461, 2020. 120, 149

[207] Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14913–14922, 2021. 120

[208] Hao Kong, Di Liu, Shuo Huai, Xiangzhong Luo, Ravi Subramaniam, Christian Makaya, Qian Lin, and Weichen Liu. Edgecompress: Coupling multi-dimensional model compression and dynamic inference for edgeai. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023. doi: 10.1109/TCAD.2023.3276938. 127

[209] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017. 131, 132

[210] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-dsod: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*, 2018. 134

[211] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 135

[212] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations (ICLR)*, 2020. 135

[213] Stefanos Laskaridis, Stylianos I Venieris, Hyeji Kim, and Nicholas D Lane. Hapi: Hardware-aware progressive inference. In *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, pages 1–9, 2020. 139

[214] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning (ICML)*, pages 3301–3310. PMLR, 2019. 139

[215] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456. pmlr, 2015. 149, 150

[216] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. 160

[217] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. 160