ABSTRACT

Title of Dissertation: IMPROVING MODEL AND DATA EFFICIENCY FOR DEEP LEARNING

Renkun Ni
Doctor of Philosophy, 2023

Dissertation Directed by: Professor Tom Goldstein
Department of Computer Science

Deep learning has achieved or even surpassed human-level performance in a wide range of challenging tasks encompassing computer vision, natural language processing, and speech recognition. Nevertheless, such achievements are predominantly derived from training huge models (i.e., billions of parameters) on numerous labeled examples, which requires considerable computation resources and expensive data collection costs. Various studies have strived to enhance efficiency in these domains. In terms of model efficiency, remarkable advancements have been made to accelerate the training and inference by methods such as quantization and pruning. Regarding data efficiency, few-shot learning, semi-supervised learning, and self-supervised learning have gathered more attention due to their abilities to learn feature representations with few labeled examples or even without human supervision. This dissertation introduces several improvements and provides an in-depth analysis of these methodologies, aiming to address the computational challenges and augment the efficiency of deep learning models, especially in computer vision.

In addressing model efficiency, we explore the potential for improvement in both the training and inference phases of deep learning processes. For model inference acceleration, we investigate the challenges of using extremely low-resolution arithmetic in quantization methods, where integer overflows frequently happen and the models are sensitive to these overflows. To address this issue, we introduce a novel module, designed to emulate the "wrap-around" property of integer overflow, which maintains comparable performance with 8-bit low-resolution accumulators. In addition, to scale inferences of Vision Transformers on mobile devices, we propose an efficient and flexible local self-attention mechanism optimized directly on mobile devices that achieves comparable performance to global attention while significantly reducing the on-device latency, especially for high-resolution tasks. Besides the computational costs, training deep neural networks consumes a large amount of memory which is another bottleneck to applying model training on edge devices. To improve the memory efficiency of training deep networks on resource-limited devices, we propose a quantization aware training framework for federated learning where only the quantized model is distributed and trained on the client devices.

In the realm of label efficiency, we first develop a better understanding of the models trained by meta-learning, which has a unique training pipeline, for few-shot classification tasks. In addition, a comprehensive analysis has been conducted to integrate data augmentation strategies into the meta-learning pipeline, leading to Meta-MaxUp, a novel data augmentation technique for meta-learning, demonstrating enhanced few-shot performance across various benchmarks. Beyond few-shot learning, the research explores the application of meta-learning methods in the context of self-supervised learning. We discuss the close relationship between meta-learning and contrastive learning, a method that achieves excellent results in self-supervised learning, under a certain task distribution.

IMPROVING MODEL AND DATA EFFICIENCY FOR DEEP LEARNING

by

Renkun Ni

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Professor Tom Goldstein, Chair/Advisor
Professor Uzi Vishkin, Dean's Representative
Professor Furong Huang
Professor Shuvra S. Bhattacharyya
Professor Tianyi Zhou

Acknowledgments

I owe my gratitude to all the people who have supported me throughout my Ph.D. journey. Without their assistance, this dissertation would not have come to fruition.

First and foremost, I would like to thank my advisor, Tom Goldstein, who supported and guided me throughout my research. His expertise and insights have been invaluable in profoundly shaping my research ideas and methodologies. His enthusiasm for work and research set a perfect example for me, illuminating a pathway on how to navigate toward a successful career. Prof. Tom Goldstein is such an open-minded and nice person, from him, I learned how to lead projects, how to cooperate and discuss with other people, and how to appreciate and respect other people's work. It was a great pleasure to work with him, and I had a wonderful time during my Ph.D. journey.

I would like to thank the members of my Ph.D. committee, Prof. Uzi Vishkin, Prof. Furong Huang, Prof. Tianyi Zhou, and Prof. Shuvra S. Bhattacharyya. Their valuable time, feedback, and suggestions have helped me a lot to improve my research, dissertation, and presentation. I would also like to express my appreciation to all the professors and colleagues I collaborated with. I would like to thank Pingyeh Chiang who worked with me on most of the projects and discussed and shared research ideas. I am also honored to collaborate with the postdocs from our lab, Micah Goldblum and Jonas Geiping. Without their help and organization, my research wouldn't be that smooth. I would like to thank all my wonderful colleagues at UMD, including

Manli Shu, Kezhi Kong, Hongmin Chu, Amr Sharaf, Hossein Souri, Chen Zhu, Amin Ghiasi, Ali Shafahi, Ahmed Abdelkader, Valeriia Cherepanova, and Liam Fowl, with whom we worked on interesting and challenging problems. I also appreciate the support from people in ETH, including Oscar Castañeda and Christoph Studer who extended my knowledge and enhanced the comprehensiveness of my research.

I was fortunate to have the opportunity to collaborate with researchers from the industry through internships. Thanks to Xue Feng, I had a great time working on computer vision applications for medical images. I appreciate the support from Adobe, including Curtis Wigington, Jiuxiang Gu, Priyanka Kulkarni, and Laurie Byrum, on the project for improving the latency of vision transformers on mobile devices. I had an unforgettable experience interning with Yonghui Xiao, Oleg Rybakov, Phoenix Meadowlark, Ananda Theertha Suresh, Ignacio Lopez Moreno, Mingqing Chen, Rajiv Mathews at Google, which introduced me the real-world problems and how to solve and conduct research on these problems. Special thanks to Jianguo Li, who gave me the first internship opportunity at Intel Labs working on quantization methods in deep learning.

Finally, I owe my gratitude to my family and friends, who gave me care and support. Thanks to my parents for supporting me over the years, and encouraging me to follow my dreams. Thanks to my brother who takes care of my family when I am away from them during my study. Thanks to my friends who gave me advice and I had lots of fun with them. Thanks to my roommates for their care which gave me a home away from home. I extend my sincerest apologies to anyone unintentionally omitted, as every contribution to my journey has been invaluable, profoundly impacting both my life and work.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ASR	Automatic Speech Recognition
BWN	Binary Weight Networks
BN	Batch Normalization
CNN	Convolutional Neural Network
DNN	Deep Neural Network
FL	Federated Learning
FSL	Few-Shot Learning
LLM	Large Language Models
LSA	Local Self-Attention
LN	Layer Normalization
MAC	Multiply-Accumulate
MHSA	Multi-Head Self-Attention
QAT	Quantization Aware Training
SSL	Self-Supervised Learning
SGD	Stochastic Gradient Descent
STE	Straight Through Estimator
TWN	Ternary Weight Networks
ViT	Vision Transformer

Chapter 1: Introduction

1.1 Background

Deep Neural Networks (DNNs) have achieved substantial success across numerous tasks, such as image recognition [6], automatic speech recognition [7], and language processing [8]. In recent decades, to achieve such promising performance, deep models have largerly expanded in both memory and computational costs. For example, vision backbones grow from 11-layers AlexNet [9] to hundreds of layers architectures like ResNet [10], DenseNet [11] and ViTs [12]. The size of the model grows from 3 MB to 210 MB, or even larger (632MB for ViT [12]). Large language models (LLM) such as Llama, GPT, and PaLM variants have billions of parameters. As a result, it is hard to apply and train these giant models on edge devices such as smartphones, drones, self-driving cars, or satellites, due to the long inference time and large energy costs. Meanwhile, more labeled training samples are required to learn these models. ImageNet [6] has around 1.2 million labeled images, COCO [13] has 328K images with extra labels such as bounding boxes used for object detection and segmentation. Collecting such a large dataset itself is difficult and expensive in the real world. In this dissertation, we summarize our recent works that improve both the model- and data-efficiency of deep learning.

1.2 Model Efficient Deep Learning

Model size and computational cost reduction can be realized through various techniques such as pruning, quantization, low-rank factorization, and efficient architecture design. Within the extensive pieces of literature, this study mainly focuses on uniform quantization for DNNs and efficient attention mechanisms used in ViTs.

1.2.1 Quantization

Quantization mainly aims at accelerating DNNs inference, where the precision of weights and activations are quantized to low bit-width, i.e., 8-bit. For uniform quantization, the tensors are first scaled to the target bit-width based on their ranges and then the float values are rounded to their closest integers [14, 15, 16]. Based on the distribution of the tensors, we mainly have two kinds of quantization, namely symmetric quantization and asymmetric quantization. For asymmetric quantization, an offset (a mapping from an integer to the real zero value) is calculated and applied to the scaled tensors, which is usually used to quantize the network's activations. Meanwhile, symmetric quantization is typically used for weight quantization, where the offset is set as zero, and the maximum of the absolute value of the tensor is used to scale it to the target bit-width.

With such quantization methods, integer-only matrix multiplication, when quantizing both weights and activations, or even simple bit-wise operations can be applied during the inference, which makes DNNs more hardware-friendly regarding inference speed and memory requirement. Unfortunately, even with these methods, we still require high-resolution arithmetic during the inference since we need an accumulator with high bit-width to accumulate hundreds or even

thousands of products after every linear operation. Otherwise, integer overflow will happen and destroy the performance of the model. How to deal with the overflows during the inference and thus allow the model to use ultra-low-resolution arithmetic still remains challenging. In this dissertation, we investigate the issue caused by overflow during both inference and training and introduce a novel module and training framework to support low-bit accumulators.

In addition to the computational cost reduction in the inference, memory usage saving in the training stage has become more and more important recently, since on-device training is widely used to improve data privacy, such as federated learning [17]. In federated learning, decentralized training is used where a model will be distributed to the client devices and this model will be trained on the client's data locally. Due to the constrained resources for client devices, such as mobile phones, it's hard to train the large and powerful models that consume a great amount of memory. As a result, it's natural to ask whether we can enjoy the same benefits from quantization when we apply it to the training process. However, to reduce the gap between the full-precision model and the quantized model, quantization-aware-training (QAT) [14] is usually used which incorporates the quantization process during the training. The full-precision weights and activations will be quantized to low-bit first and then applied in the forward pass. Since the quantization functions are not differentiable (i.e., the round function), a straight-through estimator (STE) [18] is used to calculate the gradient of the full-precision weights during the backward pass. Although memory costs can be compressed during the inference of quantized networks, for training such a model, we need to keep both the full-precision weights and the low-bit weights, which will not achieve any memory savings. This dissertation discusses how we can extend quantization to the training in federated learning thus improving memory efficiency.

1.2.2 Efficient Vision Transformer

Besides compression methods such as quantization, efficient model design is an alternative strategy that improves inference speed. Although efficient convolutional-based networks [19, 20] demonstrate great on-device latency, Vision Transformers (ViT) [12] have rapidly attracted significant attention in the recent, achieving promising results across various vision tasks. Nevertheless, ViTs are slower on-device compared to CNNs mainly due to frequent reshaping operations and expensive attention mechanisms (quadratic to the image resolution) [21]. Consequently, numerous works focused on accelerating ViTs by revisiting network architectures [21, 22, 23, 24, 25, 26, 27, 28, 29] and proposing efficient attention mechanisms [30, 31, 32, 33, 34, 35]. While these approaches enable ViTs to match the on-device latency of CNNs, the trade-off between the latency and model quality still falls short in comparison to original transformers, especially with higher input resolution for tasks such as object detection and segmentation. In this work, we revisit the local attention and optimize the performance of ViTs on edge devices directly, which achieves a better trade-off between the latency and model quality.

1.3 Data-Efficient Learning

On the other hand, constructing such ideal datasets for real-world applications, such as gathering segmentation maps for medical imaging or sourcing global street-view images for autonomous vehicles, is challenging and expensive. Recently, there has been a surge of interest in the ability of learned representations to generalize to novel tasks, particularly when trained with only limited or no labeled data. For instance, few-shot learning [3, 36, 37, 38] aims to adapt a network rapidly to new tasks with limited labeled data and generalize to unseen examples; Do-

main adaption aims to learn a network with supervision from a source data domain and generalize to the samples in a novel target domain; Self-supervised learning [39, 40, 41, 42] aims to learn feature representations without any manual supervision, which can be used to solve downstream tasks such as object detection and transfer learning. These lines of research attempt to efficiently exploit both the labeled and unlabeled data, thus making the networks generalize to unseen or novel tasks. In this section, we first introduce the standard supervised learning in machine learning, and then we mainly focus on the following two problems of data-efficient learning, namely, few-shot learning (FSL) and self-supervised learning (SSL).

1.3.1 Standard Supervised Learning

For a standard supervised learning problem, we aim to learn a model that generalized well to samples from a distribution given a set of training samples, typically from the same distribution. Formally, given training samples $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ drawn i.i.d. from distribution $\mathcal{D}$, our goal is to learn a family of models $h \in \mathcal{H}$ parameterized by $\boldsymbol{w} \in \mathcal{W}$ that has low testing loss $\mathcal{L}_{\mathcal{D}}(h) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}[l(h; \boldsymbol{x}, \boldsymbol{y})]$. The parameters are learned by minimizing the empirical loss $\mathcal{L}_{\mathcal{S}}(h) = 1/n \sum_{i=1}^{n} l(h; \boldsymbol{x}_i, \boldsymbol{y}_i)$. For example, in image classification problems, each pair of samples $(\boldsymbol{x}, \boldsymbol{y})$ is an image, and its category, h is a deep neural network, a cross-entropy loss is typically used as the loss function l and the problem is optimized by methods such as stochastic gradient descent (SGD).

1.3.2 Few-shot Learning and Meta-Learning

In modern DNNs, models usually have many parameters and can easily overfit the training samples, especially when the size of the training dataset is small. Therefore, Few-shot learning (FSL) targets to generalize the model to unseen tasks given limited training data. Different from standard supervised learning, where the model is evaluated by the testing data, FSL evaluates the model by testing tasks. Take a classification problem as an example, each learning task $\mathcal{T}_i$ itself is a N-way classification problem. In each task, it contains a support data, $\mathcal{T}_i^s = \{(\boldsymbol{x}_j^s, \boldsymbol{y}_j^s)\}_{j=1}^{K*N}$ and a query data, $\mathcal{T}_i^q = \{(\boldsymbol{x}_j^q, \boldsymbol{y}_j^q)\}_{j=1}^{M}$. Here, K denotes the number of training samples in each class, and K is usually a small number, for example, $K = 1$ or $K = 5$. M denotes the number of testing data in each task. Such learning tasks are named as K-shot-N-way classification problems. During the evaluation, the pre-trained model will first be adapted to the support data, and then the adapted model will be used to predict the label of the query data. To be noticed, in order to evaluate the generalization ability to unseen tasks, the testing tasks are typically sampled from novel classes different from the training tasks. For example, in CIFAR-FS benchmark, the training data are images sampled from 64 classes (from CIFAR-100) and the testing tasks are constructed by the images from the remaining 36 classes. The data used in testing tasks have never been seen during the training stage.

Meta-learning is usually known as learning to learn, which aims to learn a model over multiple training episodes and can quickly adapt to new tasks [43]. As a result, meta-learning methods align well with FSL and achieve promising results [44, 45, 46]. When applied to FSL, the training episodes of meta-learning are few-shot classification problems containing support and query sets which is the same as the evaluation process of FSL. Models are learned to gener-

alize to new tasks by training to solve problems with randomly sampled and shuffled categories from training data. While the literature is rich with meta-learning methods, the underlying mechanics of why meta-learned feature extractors perform so well, as well as their distinctions and similarities with those learned from conventional model training, remain inadequately explored and understood. In addition, as the training process of meta-learning is very different from the conventional model training, the proposition of proper data augmentation techniques becomes crucial for enhancing performance.

1.3.3 Self-Supervised Learning

Self-supervised learning (SSL) aims to learn meaningful feature representations for downstream tasks without any manual supervision. Given training samples $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i)\}$, a feature extractor f parameterized by $\boldsymbol{w} \in \mathcal{W}$ is learned with a specifically designed loss $\mathcal{L}_{\mathcal{S}}(f) = 1/n \sum_{i=1}^{n} l(f; \boldsymbol{x}_i)$. Thanks to no manual supervision, here n can be much larger but with little additional cost. In this dissertation, we mainly focused SSL methods for computer vision. Although different kinds of loss functions can be used, most SSL methods share the same training pipeline, where different data augmentations are applied to the same base image, and distances of the augmentations from the same base images are measured during training. Contrastive learning approaches, including SimCLR [40], MoCo variants [39, 47] apply a contrastive loss that not only minimizes the distances between augmented images from the same base images (positive pairs) but also maximizes the distances between augmented images derived from disparate bases (negative pairs). Later on, other techniques [41, 48, 49] find that SSL models can be trained successfully without negative pairs by different distance metrics, such as l_1 loss [48] or entropy

loss [41]. Moreover, methods such as MAE [42] train a vision model in a generative way where they use an autoencoder to reproduce the masked input images. The performance of SSL algorithms is evaluated by downstream tasks such as linear evaluation (the feature extractor is fixed and only learns a linear classification head), semi-supervised learning (fine-tuning the whole model end-to-end but with only a small proportion of labeled data), or setting the learned feature extractor as the backbone for other tasks such as object detection and segmentation.

1.4 Outline of Thesis

Chapter 2 to Chapter 4 of this dissertation will introduce our work on improving model efficiency with quantization and efficient vision transformer designs. In Chapter 2, we propose a network [50] that can accelerate the model inference by leveraging ultra-low-precision arithmetic. In addition to using low-bit weights and activations, our method facilitates the model using low-bit accumulators as well. Besides model inference, Chapter 3 presents a quantized model training framework for federated learning which reduces the memory costs during training thus enabling training deep models on resource-limited devices such as mobile phones. Chapter 4 explores the opportunity for reducing the computational cost of attention mechanisms in vision transformers. Our proposed local self-attention can achieve a better trade-off for on-device latency and accuracy, especially when the input resolution is very high.

In the last three chapters, we present our efforts to understand and enhance label efficiency for deep neural networks. Chapter 5 proposes a better understanding of why meta-learning works better than conventional training in few-shot learning [51]. Moreover, Chapter 6 introduces a novel data augmentation pipeline [52] for meta-learning based on its unique episode training

which improves the few-shot performance by a large margin. Chapter 7 revisits the training framework for contrastive-based SSL methods and shows the similarity to the meta-learning training pipeline [53]. We consider the SSL problem as a 1-shot few-shot classification and learn the feature representation un-supervisedly with meta-learning algorithms. The meta-learned feature extractors present a strong ability to transfer to datasets with other domains and datasets with limited training examples.

Chapter 2: Accelerate Network Inference with Ultra-Low-Precision Arithmetic

2.1 Introduction

Significant progress has been made in quantizing (or even binarizing) neural networks, and numerous methods have been proposed that reduce the precision of weights, activations, and even gradients while retaining high accuracy [18, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67]. Such quantization strategies make neural networks more hardware-friendly by leveraging fast, integer-only arithmetic, replacing multiplications with simple bit-wise operations, and reducing memory requirements and bandwidth.

Unfortunately, the gains from quantization are limited because quantized networks still require high-precision arithmetic. Even if weights and activations are represented with just one bit, deep feature computation requires the summation of hundreds or even thousands of products. Performing these summations with low-precision registers results in integer overflow, contaminating downstream computations and destroying accuracy. Moreover, as multiplication costs are slashed by quantization, high-precision accumulation starts to dominate the arithmetic cost. Indeed, our own hardware implementations show that an 8-bit $\times$ 8-bit multiplier consumes comparable power and silicon area to a 32-bit accumulator. When reducing the precision to a 3-bit $\times$ 1-bit multiplier, a 32-bit accumulator consumes more than $10\times$ higher power and area; see Section 2.4.5. Evidently, low-precision accumulators are the key to further accelerating quantized

nets.

In custom hardware, low-bit accumulators reduce area and power requirements while boosting throughput. On general-purpose processors, where registers have fixed size, low-precision accumulators are exploited through *bit-packing*, i.e., by representing multiple low-precision integers side-by-side within a single high-precision register [57, 68, 69]. Then, a single vector instruction is used to perform the same operation across all of the packed numbers. For example, a 64-bit register can be used to execute eight parallel 8-bit additions, thus increasing the throughput of software implementations. Hence, the use of low-precision accumulators is advantageous for both hardware and software implementations, provided that integer overflow does not contaminate results.

We propose WrapNet [50], a network architecture with extremely low-precision accumulators. WrapNet exploits the fact that integer computer arithmetic is cyclic, i.e., numbers are accumulated until they reach the maximum representable integer and then "wrap around" to the smallest representable integer. To deal with such integer overflows, we place a differentiable cyclic (periodic) activation function immediately after the convolution (or linear) operation, with period equal to the difference between the maximum and minimum representable integer. This strategy makes neural networks resilient to overflow as the activations of neurons are unaffected by overflows during convolution.

We explore several directions with WrapNet. On the software side, we consider the use of bit-packing for processors with or without dedicated vector instructions. In the absence of vector instructions, overflows in one packed integer may produce a carry bit that contaminates its neighboring value. We propose training regularizers that minimize the effects of such contamination artifacts, resulting in networks that leverage bit-packed computation with very little

impact on final accuracy. For processors with vector instructions, we modify the Gemmlowp library [70] to operate with 8-bit accumulators. Our implementation achieves up to $2.4\times$ speed-up compared to a 32-bit accumulator implementation, even when lacking specialized instructions for 8-bit multiply-accumulate. We also demonstrate the efficacy of WrapNet in terms of cycle time, area, and energy efficiency when considering custom hardware designs in a commercial 28 nm CMOS technology.

2.2 Related Work and Background

2.2.1 Network Quantization

Network quantization aims at accelerating inference by using low-precision arithmetic. In its most extreme form, weights and activations are both quantized using binary or ternary quantizers. The binary quantizer Q_b corresponds to the sign function, whereas the ternary quantizer Q_t maps some values to zero. Multiplications in binarized or ternarized networks [54, 56, 57, 58, 71] can be implemented using bit-wise logic, leading to impressive acceleration. However, training such networks is challenging since fewer than 2 bits are used to represent activations and weights, resulting in a dramatic impact on accuracy compared to full-precision models.

Binary and ternary networks are generalized to higher precision via uniform quantization, which has been shown to result in efficient hardware [14]. The multi-bit uniform quantizer Q_u is given by: $Q_u(x) = \text{round}(x/s_x)s_x$, where s_x denotes the quantization step-size. The output of the quantizer is a floating-point number x that can be expressed as $x = s_x x_q$, where x_q is the fixed-point representation of x. The fixed-point number x_q has a "precision" or "bitwidth," which is the number of bits used to represent it. Note that the range of floating-point numbers

representable by the uniform quantizer Q_u depends on both the quantization step-size s_x and the quantization precision. Nonetheless, the number of different values that can be represented by the same quantizer depends only on the precision.

Applying uniform quantization to both weights $w = s_w w_q$ and activations $x = s_x x_q$ simplifies computations, as an inner-product simply becomes

$$z = \sum_i w_i x_i = \sum_i (s_w (w_q)_i)(s_x (x_q)_i) = (s_w s_x) \sum_i (w_q)_i (x_q)_i = s_z z_q. \tag{2.1}$$

The key advantage of uniform quantization is that the core computation $\sum_i (w_q)_i (x_q)_i$ can be carried out using fixed-point (i.e., integer) arithmetic only. Results in recent works [64, 65, 66, 67, 72, 73] have shown that high classification accuracy is attainable with low-bitwidth uniform quantization, such as 2 or 3 bits. Although $(w_q)_i$, $(x_q)_i$, and their product may have extremely low precision, the accumulated result z_q of many of these products has a very high dynamic range. As a result, high-precision accumulators are typically required to avoid overflows, which is the bottleneck for further arithmetic speedups.

2.2.2 Low-precision Accumulation

Several approaches have been proposed that use accumulators with fewer bits to obtain speed-ups. For example, reference [74] splits the weights into two separate matrices, one with small- and another with large-magnitude entries. If the latter matrix is sparse, acceleration is attained as most computations rely on fast, low-precision operations. However, to significantly reduce the accumulator's precision, one would need to severely decrease the magnitude of the entries of the first matrix, which would, in turn, prevent the second matrix from being sufficiently

sparse to achieve acceleration. Recently, de Bruin et al. [75] proposed using layer-dependent quantization parameters to avoid overflowing accumulators with fixed precision. Fine-tuning is then used to improve performance. However, if the accumulator precision is too low (e.g., 8 bits or less), the optimized precision of activations and weights is too coarse to attain satisfactory performance. Another line of work [76, 77, 78] uses 16-bit floating-point accumulators for training and inference—such approaches typically require higher complexity than methods based on fixed-point arithmetic.

2.2.3 The Impact of Integer Overflow

Overflow is a major problem, especially in highly quantized networks. Table 2.1 demonstrates that overflows occur in around 11% of the neurons in a network with 3-bit activations (A) and binary weights (W) that uses 8-bit accumulators for inference after being trained on CIFAR-10 with standard precision. Clearly, overflow has a significant negative impact on accuracy. Table 2.1 shows that if we use an 8-bit (instead of a 32-bit) accumulator, then the accuracy of a binary-weight network with 2-bit activations drops by more than 40%, even when only 1.72% neurons overflow. If we repeat the experiment with 3-bit activations and binary weights, the accuracy is only marginally better than a random guess. Therefore, existing methods try to *avoid* integer overflow by using accumulators with relatively high precision, and pay a correspondingly high price when doing arithmetic.

Table 2.1: Average overflow rate (in 8 bits) of each layer for a low-precision network and corresponding test accuracy using either 32-bit or 8-bit accumulators during inference on CIFAR10.

Bit (A/W)	Overflow rate (8-bit)	Accuracy (32-bit)	Accuracy (8-bit)
full precision	–	92.45%	–
3/1	10.84%	91.08%	10.06%
2/1	1.72%	88.46%	44.04%

2.3 Proposed Method

2.3.1 WrapNet: Dealing with Integer Overflows

We now introduce WrapNet, which includes a cyclic activation function and an overflow penalty, enabling neural networks to use low-precision accumulators. We also present a modified quantization step-size selection strategy for activations, which retains high classification accuracy. Finally, we show how further speed-ups can be achieved on processors with or without specialized vector instructions.

We propose training a network with layers that emulate integer overflows on the fixed-point pre-activations z_q to maintain high accuracy. However, directly training a quantized network with an overflowing accumulator diverges (see Table 2.2) due to the discontinuity of the modulo operation. To facilitate training, we insert a cyclic "smooth modulo" activation immediately after every linear/convolutional layer, which not only captures the wrap-around behavior of overflows, but also ensures that the activation is continuous everywhere. The proposed smooth modulo activation c is a composite function of a modulo function m and a basis function f that ensures continuity. Specifically, given a b-bit accumulator, our smooth-modulo c for fixed-point inputs is

as follows:

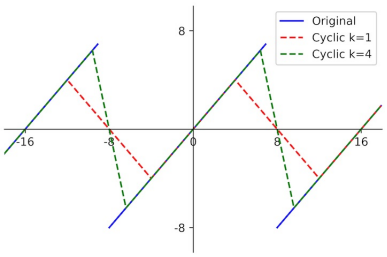
$$f(m) = \begin{cases} m, & \text{for } -\frac{k}{k+1}2^{b-1} \leq m \leq \frac{k}{k+1}2^{b-1} \\ -k2^{b-1} - km, & \text{for } m < -\frac{k}{k+1}2^{b-1} \\ k2^{b-1} - km, & \text{for } m > \frac{k}{k+1}2^{b-1} \end{cases}$$

$$c(z_q) = f(\text{mod}(z_q + 2^{b-1}, 2^b) - 2^{b-1}),$$

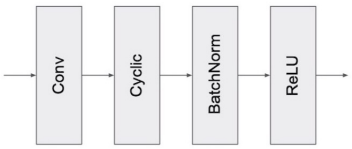
where k is a hyper-parameter that controls the slope of the transition. Note that we apply constant shifts to keep the input of f in $[-2^{b-1}, 2^{b-1})$. Figure 2.1a illustrates the smooth modulo function with two different slopes $k = 1, 4$. As k increases, the cyclic activation becomes more similar to the modulo operator and has a greater range, but the transition becomes more abrupt. Since our cyclic activation is continuous and differentiable almost everywhere, standard gradient-based learning can be applied easily. A convolutional block with cyclic activation layer is shown in Figure 2.1b. After the convolution result goes into the cyclic activation, the result is multiplied by s_z to compute a floating-point number, which is then processed through BatchNorm and ReLU. A fixed per-layer quantization step-size is then used to convert the floating-point output of the ReLU into a fixed-point input for the next layer. We detail the procedure to find this step-size in Section 2.3.3.

2.3.2 Overflow Penalty

An alternative way to adapt quantized networks to low-precision accumulators is to directly reduce the amount of overflows. To achieve this, we propose a regularizer that penalizes outputs



(a)



(b)

Figure 2.1: (a) Example of the proposed cyclic activation with different slopes k and the original modulo operator for a 4-bit accumulator. (b) Convolutional block with proposed cyclic activation.

that exceed the bitwidth of the accumulation register. Concretely, for a b-bit accumulator, we define an overflow penalty for the l-th layer of the network as follows:

$$R_l^o = (1/N) \sum_i \max\{|z_q^i| - 2^{b-1}, 0\}.$$

Here, z_q^i is the fixed-point result in (2.1) for the i-th neuron of the l-th layer, and N is the total number of neurons in the l-th layer. The overflow penalty is imposed after every quantized linear layer and before the cyclic activation. All these penalties are combined into one regularizer $R^o = \sum_l R_l^o$.

2.3.3 Selection of Activation Quantization Step-Size

To keep multiplication simple, the floating-point output of ReLU must be quantized before it is fed into the following layer. However, as shown in Table 2.1, a significant number of overflows occur even with 3-bit activations. From our experiments (see Table 2.3), we have observed that if overflow occurs too frequently (i.e., on more than 10% of the neurons), then WrapNet starts to suffer significant accuracy degradation. However, if we reduce the activation precision so that no overflows happen at all, several layers will have 1-bit activations (see Table- 2.3), thereby in-

creasing quantization errors and degrading accuracy. To balance accumulation and quantization errors, we adjust the quantization step-size s_x of each layer based on the overflow rate, i.e., the percentage $p\%$ of neurons that overflow in the network. If the overflow rate $p\%$ is too large, then we increase s_x to reduce the overflow rate $p\%$. The selected quantization step-size is then fixed for further fine-tuning.

2.3.4 Adapting to Bit-Packing

Most modern processors provide vector instructions that enable parallel operation on multiple 8-bit numbers. For instance, the AVX2 (NEON) instruction set on x86 (ARM) processors provides parallel processing with 32 (16) 8-bit numbers. Vector instructions provide a clean implementation of bit-packing, which WrapNet can leverage to attain significant speed-ups. While some embedded processors and legacy chips do not provide vector instructions, bit-packing can still be applied. Without vector instructions for multiplication, binary/ternary weights must be used to replace multiplication with bit-wise logic [68, 69]. Furthermore, bit-packing of additions is more delicate: Each integer overflow not only results in wrap-around behavior, but also generates a carry bit that contaminates the adjacent number—specialized vector instructions avoid such contamination. We propose the following strategies to minimize the impact of carry propagation.

Reducing variance in the number of carries. The number of carries generated during a convolution operation can be large. Nevertheless, if we can keep the number of carries approximately the same for all the neurons among a batch of images, the estimated number of carries can be subtracted from the result to correct the outputs of a bit-packed convolution operation. To achieve this, during training, we calculate the number of carries for each neuron and impose a

regularizer, R^c, to keep the variance of the number of carries small. The detailed formulation of R^c can be found in Appendix 2.6.1.1. **Using a buffer bit.** Alternatively, since each addition can generate at most one carry bit, we can place a buffer bit between every low-bit number in the bit-packing. For example, instead of packing eight 8-bit representations into a 64-bit number, we pack eight 7-bit numbers with one buffer bit between each of them. These buffer bits absorb the carry bits, and are cleared using bit-wise logic after each addition. Buffering makes representations 1-bit smaller, which potentially degrades accuracy. **A hybrid approach.** To get the benefits from both strategies, we use a variance penalty on layers that have small standard deviations to begin with and equip the remaining layers with a buffer bit.

2.4 Experiments

We compare the accuracy and efficiency of WrapNet to networks with full-precision accumulators using the CIFAR-10 and ImageNet datasets. Most experiments use binary or ternary weights for WrapNet as AVX2 lacks 8-bit multiplication instructions, but supports 8-bit additions and logic operations needed for binary/ternary convolutions.

2.4.1 Training Pipeline

We first pre-train a network with quantized weights and no cyclic layers, while keeping full-precision activations. Then, we select the quantization step-sizes of the activations (see Section 2.3.3) such that each layer has an overflow rate of around $p\%$ (a hyper-parameter) with respect to the desired accumulator bitwidth. Given the selected quantization step-size for each layer and the pre-trained network, we insert our proposed cyclic activation layer. We then warm-

up our WrapNet by fine-tuning with full-precision activation for several epochs. Finally, we further fine-tune the network with both activations and weights quantized. Both overflow and carry variance regularizers are only applied in the final fine-tuning step, except when training ResNet for ImageNet, where the regularizers are also included during warm-up.

2.4.2 Adapting to Low-precision Accumulators

We conduct ablation studies on the following factors: the type of cyclic function, the initial overflow rate for quantization step-size and precision selection, and the coefficient of the overflow penalty regularizer. These experiments are conducted on VGG-7 [55], which is commonly used in the quantization literature for CIFAR-10. We binarize the weights as in Rastegari et al. [57], and we train WrapNet to adapt to an 8-bit accumulator. As our default setting, we use $k = 2$ as the transition slope, $p = 5\%$ as the initial overflow rate, and 0 as the coefficient of the regularizer.

Cyclic activation function. We compare the performance of various transition slopes k of our cyclic function c in Table 2.2, and we achieve the best performance when $k = 2$. If k is too small, then the accuracy decreases due to a narrower effective bitwidth (only half of the bitwidth is used when $k = 1$). Meanwhile, the abrupt transition for large k hurts the performance as well. In the extreme case where the cyclic function degenerates to modulo ($k \to \infty$), WrapNet diverges to random guessing, which highlights the importance of training with a "smooth" cyclic non-linearity to assimilate integer overflow. We also find that placing a ReLU after batch norm yields the best performance, even though the cyclic function is already nonlinear. More experimental results can be found in Appendix 2.6.2.1.

Quantization step-size. As described in Section 2.3.3, the quantization step-sizes are se-

Table 2.2: Results for different transition slopes for cyclic function; $*$ denotes divergence.

k	1	2	4	10	∞
Accuracy	90.24%	90.52%	90.25%	89.16%	$*$

lected to balance the rounding error of the activations and accumulation errors due to overflow. We compare the classification performance when we choose different step-sizes to control the overflow rate as in Table 2.3. If the initial overflow rate is large, then the quantization step-size will be finer, but training is less stable. We obtain the best performance when the initial overflow rate is around 5%. The median bitwidths of the activations across layers are also reported in Table 2.3. Note that if we want to suppress all overflows, we can only use 1-bit activations. We also observe that WrapNet can attain reasonable accuracy (85%) even with a large overflow rate (around 30%), which demonstrates that our proposed cyclic activations provide resilience against integer overflows.

Overflow penalty. The overflow penalty regularizer improves the stability of step-size selection. More specifically, in Table 2.4, the difference in accuracy between two step-size selections decreases from 2.27% to 0.76% after adding the regularizer. The overflow penalty also complements our cyclic activation, as we achieve the best performance when using both of them together during the fine-tuning stage. Moreover, in Appendix 2.6.2.2, we compare our results to fine-tuning the pre-trained network using the overflow regularizer only. In the absence of a cyclic layer, neural networks still suffer from low accuracy (as in Section 2.2.3) unless a very strong penalty is imposed.

Table 2.3: Results for different step-sizes based on overflow rate $p(\%)$. $*$ denotes divergence.

p	Bits	Accuracy	p	Bits	Accuracy
0	1	90.07%	20	4	88.25%
2	3	90.51%	30	5	85.30%
5	3	90.52%	40	5	36.11%
10	4	89.92%	50	5	$*$

Table 2.4: Results for fine-tuning with the overflow penalty (R^o).

R^o	$p\%$	Accuracy	Difference
0	20	88.25%	–
0	5	90.52%	2.27%
0.01	20	90.05%	–
0.01	5	90.81%	0.76%

2.4.3 Adapting to Bit-Packing

We now show the efficacy of WrapNet for bit-packing without vector operations. We use the same experiments setting as in Section 2.4.2. The training details can be found in Appendix 2.6.1.2. We consider CIFAR-10, and we compare with the best result of WrapNet from the previous section as a baseline. Without specific vector instructions, accuracy degenerates to a random guess because of undesired carry contamination during inference.

Surprisingly, with the carry variance regularizer, WrapNet works well even with abundant carry contamination during inference (for each neuron, 384 on average over all the dataset). The regularizer drops the standard deviation of the per-neuron carry contamination by 90%. When we use the hybrid approach, the accuracy is further improved (89.43%) and close to the best result (90.81%) we can achieve with vector instructions that do not propagate carries across different numbers (see Table 2.5).

Table 2.5: Results for adaptation to bit-packing with 8-bit accumulator. (v) denotes no carry contamination as in a vector instruction; (c) denotes carry propagation between different numbers.

Method	Accuracy (v)	Accuracy (c)	Carry	Carry Std
Baseline	90.81%	10.03%	254.91	159.55
Buffer Bit	–	88.22%	–	–
R^c	–	87.86%	384.42	17.91
Hybrid	–	89.43%	482.4	16.18

2.4.4 Benchmark Results

In this section, we compare our WrapNet when there is no carry contamination, with the following 32-bit accumulator baselines: a full-precision network (FP), a network trained with binary/ternary weights but with full-precision activations (BWN/TWN), and a network where both weights and activations are quantized to the same precision as our WrapNet (BWN/TWN-QA). We benchmark our results on both CIFAR-10 and ImageNet. We use VGG7 and ResNet20 for our CIFAR-10 experiments, and we use AlexNet [9, 79], ResNet18 and ResNet50 [10] for our ImageNet experiments. Details of training can be found in Appendix 2.6.2.3.

For CIFAR-10, even with an 8-bit accumulator, our results are comparable to both BWN and TWN. When adapting to a 12-bit accumulator, we further achieve performance on-par with TWN and better than BWN (see Table 2.6). For ImageNet, our WrapNet can achieve accuracy as good as BWN when adapting to a 12-bit accumulator where we can use binary weights and roughly 7-bit quantized activations. However, in the extremely low-precision case (8-bit), the accuracy of our binary WrapNet drops around 8% due to the limited bitwidth we can use for activations. As reported in Table 2.6, the median activation bitwidth is roughly 3-bit, and for some layers in AlexNet, we can only use 1-bit activations. Despite the gap from BWN, we observe that our model can achieve comparable performance as BWN-QA where the same precision is used for activations. When using ternary weights and an 8-bit accumulator, our WrapNet only drops by 3% and 2% from TWN for ResNet18 and ResNet50, respectively. In addition, in the case of adapting to a 12-bit accumulator, our ternary WrapNet with roughly 7-bit activations is even slightly better than TWN for ResNet50. Note that, without cyclic activation function, all the results for networks using 8-bit accumulator are as poor as random guessing which is consistent

with Table 2.1.

Table 2.6: Top-1 test accuracy for both CIFAR-10 and ImageNet with different architectures. Here, "Acc" represents accumulator, and "QA" represents quantized activation.

	Bits			CIFAR-10		ImageNet		
	Activation	Weight	Acc	VGG7	ResNet20	AlexNet	ResNet18	ResNet50
FP	32	32	32	92.45%	91.78%	60.61%	69.59%	76.15%
BWN	32	1	32	91.55%	90.03%	56.56%	63.55%	72.88%
BWN-QA	~ 3	1	32	91.30%	89.86%	46.30%	57.54%	66.85%
WrapNet	~ 3	1	8	90.81%	89.78%	44.88%	55.60%	64.30%
WrapNet	~ 7	1	12	91.59%	90.17%	56.62%	63.11%	72.37%
TWN	32	2	32	91.56%	90.36%	57.57%	65.70%	73.31%
TWN-QA	~ 4	2	32	91.49%	90.12%	55.84%	63.67%	72.50%
WrapNet	~ 4	2	8	91.14%	89.56%	52.24%	62.13%	71.62%
WrapNet	~ 7	2	12	91.53%	90.88%	57.60%	63.84%	73.93%

2.4.5 Efficiency Analysis

We conduct an efficiency analysis of parallelization by bit-packing, both with and without vector operations, on an Intel i7-7700HQ CPU operating at 2.80 GHz. We also conduct a detailed study of improvements that can be obtained using custom hardware.

AVX2 instruction efficiency analysis. We study the empirical efficiency of WrapNet when vector operations are available. We extended Gemmlowp [70] to implement matrix multiplications using 8-bit accumulators with AVX2 instructions. To demonstrate the efficiency of low-precision accumulators, we compare our implementation with the AVX2 version of Gemmlowp, which uses 32-bit accumulators. We report the execution speed of both on various convolution kernels of ResNet18 in Table 2.7. From Table 2.7 we observe significant speed-ups ranging from $2\times$ to $2.4\times$ among different blocks. Besides, we compare the entire inference time (ms) of ResNet18 for WrapNet (234.74) with a 32b-accumulator quantized network (312.42), which gains 33% speed-up. The result provides solid evidence for the efficiency advantage of using

Table 2.7: Time cost (ms) for 3×3 convolution kernels in ResNet using various accumulator bitwidths.

Input size	Output	8-bit	32-bit
64x56x56	64	**3.467**	8.339
128x28x28	128	**2.956**	6.785
256x14x14	256	**2.499**	5.498
512x7x7	512	**2.710**	5.520

Table 2.8: Time cost (ms) for 3×3 convolution kernels in ResNet with no vector instructions using bit packing.

Input size	Output	bit packing	naïve
64x56x56	64	**29.80**	83.705
128x28x28	128	**23.86**	80.557
256x14x14	256	**21.71**	86.753
512x7x7	512	**20.41**	87.671

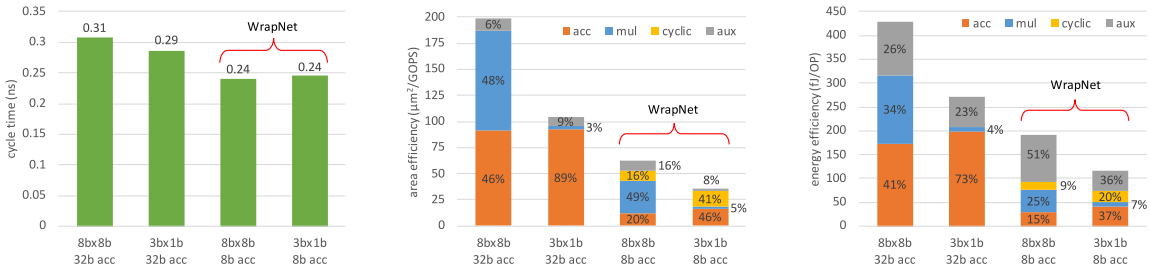
low-precision accumulators. We remark that in average, the time cost for cyclic activation is only around 10% of the time cost for the GEMM kernel. We also remark that AVX2 lacks a single instruction that performs both multiplication and accumulation for 8-bit data, but it does have such instruction for 32-bit data. Thus, further acceleration can be achieved on systems like ARM where such combined instructions for 8-bit data are available.

Bit-packing results without vector operations. We implement a naïve for-loop based matrix multiplication, which uses buffer bit and logical operations introduced in Section 2.3.4 to form the baseline. We then pack four 8-bit integers into 32 bits, and report the execution speed of both implementations on various convolution kernels of ResNet18 in Table 2.8. The results show significant speed-ups ranging from $2.8\times$ to $4.3\times$. Such observations demonstrate our proposed approach to handle extra carry bits makes bit-packing viable and efficient, even when vector instructions are not available.

Hardware analysis. To illustrate the potential benefits of WrapNet for custom hardware accelerators, we have implemented a multiply-accumulate (MAC) unit in a commercial 28nm

CMOS technology. The MAC unit consists of (i) a multiplier with an output register, (ii) an accumulator with its corresponding register, and (iii) auxiliary circuitry. Please refer to Appendix 2.6.3 for the details. We have considered 8-bit $\times$ 8-bit and 3-bit $\times$ 1-bit multipliers, as well as 32-bit and 8-bit accumulators, where the latter option is enabled by our WrapNet approach and its cyclic activation function. We consider a slope $k = 2$ for the cyclic activation. Figure 2.2 shows our post-layout results.

Figure 2.2a shows that reducing the multiplier bitwidth decreases the cycle time by 7%; reducing the accumulator precision from 32-bit to 8-bit further the cycle time by 16%. Figures 2.2b and 2.2c highlight the importance of reducing the accumulator's precision. When using an 8-bit $\times$ 8-bit multiplier, the 32-bit accumulator already constitutes more than 40% of the area and energy of a MAC unit. Once the multiplier's precision reduces, the accumulator dominates area- and energy-efficiency. Thanks to WrapNet, we can reduce the accumulator precision from 32-bit to 8-bit, thus reducing the accumulator's area- and energy-efficiency by more than $5\times$ and $4\times$, respectively. WrapNet requires the implementation of the cyclic activation, which has an area- and energy-efficiency comparable (although lower) to that of the accumulator. In spite of this overhead, WrapNet is still able to reduce the total MAC unit's area- and energy-efficiency by up



(a) (b) (c)

Figure 2.2: (a) Cycle time, (b) area and (c) energy efficiency for different MAC units implemented in 28nm CMOS. We consider 8-bit $\times$ 8-bit or 3-bit $\times$ 1-bit multipliers with 32-bit or 8-bit accumulators.

to $3\times$ and $2\times$, respectively. While our hardware implementation only uses one adder per inner-product, we note that WrapNet can also be applied to spatial architectures, such as systolic arrays, which use several adders per inner-product. For such spatial architectures, WrapNet avoids an increase in the adders' bitwidth, normalizing all adders to the same low bitwidth. Moreover, the use of several adders per inner-product amortizes the overhead from the cyclic activation, of which only one is needed per inner-product. Finally, we note that this analysis only considers the computation part of a hardware accelerator as this is where WrapNet has a significant impact—the memory sub-system will remain virtually the same, as existing methods already quantize the output activations to low-bit before storing them in memory.

2.5 Conclusion

We have proposed WrapNet, a novel method to render neural networks resilient to integer overflow, which enables the use of low-precision accumulators. We have demonstrated the effectiveness of our adaptation on both CIFAR-10 and ImageNet. In addition, our custom GEMM kernel achieves $2.4\times$ acceleration over its standard library version, and our hardware exploration shows significant improvements in area- and energy-efficiency. Our hope is that hardware-aware architectures will enable deep learning applications on a wide range of platforms and mobile devices. Furthermore, with future innovations in GPU and data center technologies, we hope that WrapNet can provide further speed-ups by enabling inference using quarter-precision—a step forward in terms of performance from the currently available half-precision standard available on emerging GPUs.

2.6 Appendix: Additional Details and Analysis

2.6.1 Details of Carry Variance Reduction Regularizer

2.6.1.1 Carry Variance Calculation

With two's complement representations for signed integers, a carry bit is generated in the following three cases: (i) addition of two negative numbers, (ii) addition of two positive numbers whose result exceeds the representation range, thus provoking integer overflow, and (iii) addition of a positive and a negative number whose result is a positive number. Dealing with these cases individually is complicated, but the calculation can be simplified by first reinterpreting the two's complement representation as an unsigned integer. Carry bits resulting from accumulation of unsigned integers are easier to calculate as they can only happen in case (ii) as described above.

Since we only consider binary/ternary weights for bit-packing, carry bits can only be generated during accumulation, and not by multiplication. To produce a single output from a convolution, we must perform the accumulation $\sum_{i=1}^{L} \mathbf{v}_i$ of all entries of the vector $\mathbf{v} \in \mathbb{R}^L$. This is done by batching computations inside a b-bit register as follows. First, we bit-pack groups of numbers $\mathbf{v}_i$ into several high-resolution registers. For example, let us consider the use of 32-bit registers to pack four $b = 8$-bit numbers; then, we need to use $\lceil L/4 \rceil$ 32-bit registers to represent all L entries of $\mathbf{v}$. In the absence of vector instructions, the addition of these high-resolution registers will generate carry bits that will contaminate the adjacent bit-packed numbers. After all $\lceil L/4 \rceil$ additions take place, we add the 4 bit-packed numbers together to get a final result.

When one output feature is calculated by bit-backing as described above, the effect of carry bits is easy to simulate; accumulations can be done without accounting for carry bits during

Algorithm 1 Carry Amount Calculation

Initialize: $\mathbf{v}, b$
$u = \sum_i \left((\text{sign}(\mathbf{v}_i) + 1)/2\right) \mathbf{v}_i + \left((-\text{sign}(\mathbf{v}_i) + 1)/2\right) \left(\mathbf{v}_i + 2^b\right)$
$c_i = u, r_i = 0, c = 0$
while $c_i \neq 0$ **do**
  $c_{i+1} = \left\lfloor (c_i + r_i)/2^b \right\rfloor$
  $r_{i+1} = (c_i + r_i) \bmod 2^b$
  $c = c + c_{i+1}$
  $c_i = c_{i+1}, r_i = r_{i+1}$
Return: c

convolution, and then the carry bits can be added into the the final result after convolution takes place. If the total number of carries is large, this final correction can in turn produce new carry bits. Hence, we use Algorithm 1 to compute the total number of carry bits that are generated in an accumulation. The first equation simply reinterprets the signed representation to its unsigned counterpart u. Then, we compute the amount of carry bits c_i, as well as the result r_i remaining within the b-bit accumulator. Due to carry contamination, the carry bits c_i will be added to the result r_i, which may generate new carry bits c_{i+1}. We keep on adding the new carry bits to the accumulator until no new carry bits are generated. Note that, in real hardware at inference time, the most significant carry bit produced inside a register will be thrown away. For simplicity, our simulations during training accumulate all carry bits, including the most significant. We find that dropping the most significant carry during inference does not significantly impact testing.

Given the number of carry bits calculated during the inner product, the variance of the carry among a batch (b_s) of images is calculated as follows:

$$\text{m}_{b_s}(n^{i,l}) = \frac{1}{b_s} \left(\sum_{k=1}^{b_s} n_k^{i,l} \right), \tag{2.2}$$

$$\text{var}_{b_s}(n^{i,l}) = \frac{1}{b_s} \left(\sum_{k=1}^{b_s} \left(n_k^{i,l} - \text{m}_{b_s}(n^{i,l}) \right)^2 \right), \tag{2.3}$$

where $n^{i,l}$ is the carry bit for the i-th neuron in l-th layer (assuming all the feature maps are vectorized). The estimated mean among all the images is learned by a moving average based on the mean of batches equation 2.2. However, the sign and rounding function may have zero gradient almost everywhere. To make all the operations differentiable, we replace the sign function with a tanh function and we use a straight through estimator for rounding during the backward pass (gradient is identity). Then finally, our regularizer R^c will be the mean variance among all the neurons.

2.6.1.2 Training with Carry Variance Reduction Regularizer

Due to the large amount and high variance of carry-bit occurrences, it is hard to fine-tune our WrapNet even when using the carry variance reduction regularizer. The generated carry bits will be accumulated, which increases the overflow rate dramatically. In addition, the accumulation error will contaminate downstream computations and destroy accuracy. As a result, we fine-tune WrapNet with simulated carry bits layer by layer, starting from the layer that has the least carry variance. For the hybrid approach, we stop simulating the carry bit when we notice a significant accuracy drop; the remaining layers are trained using a buffer bit instead.

2.6.2 Experimental Details

2.6.2.1 More Cyclic Functions

We compare two more "smooth" cyclic functions with our proposed cyclic activation function in Section 2.3. Specifically, we consider a cyclic absolute value function, and a ReLU-like function with transition slope k as alternative cyclic activations. Figure 2.3 illustrates the com-

pared functions. We compare the results with and without a ReLU activation after batch normalization as well. Table 2.9 shows that retaining the ReLU activation after the batch normalization layer always achieves a better result, and that our proposed cyclic activation outperforms the other two choices.

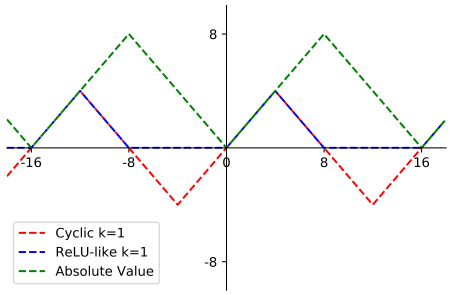


Figure 2.3: Example of the compared cyclic functions for a 4-bit accumulator.

Table 2.9: Results for different types of cyclic activation

Cyclic Function	ReLU	slope k	Accuracy(%)
Proposed	√	2	**90.52**
Proposed		2	89.28
ReLU-like	√	1	90.25
ReLU-like	√	2	90.31
ReLU-like	√	3	90.15
ReLU-like		1	88.62
ReLU-like		2	89.01
ReLU-like		3	88.53
Absolute	√	–	90.17
Absolute		–	89.19

2.6.2.2 Full Overflow Penalty Results

Table 2.10 shows the results for fine-tuning our WrapNet with different coefficients for the overflow penalty. When applying the overflow penalty, the overflow rate decreases and we can achieve a higher accuracy. In addition, when we apply the regularizer to a network with low-resolution accumulators that does not use our cyclic activation, the network still suffers from

performance degradation unless a large coefficient is used. However, a strong penalty kills almost all of the overflow, which may limit the performance of a deep neural network.

Table 2.10: Comparison for fine-tuning network without cyclic activation and our WrapNet, with overflow penalty R^o.

Cyclic	R^o	Overflow rate (%)	Accuracy(%)
√	0	6.29	90.52
√	0.001	1.88	90.33
√	0.01	1.24	**90.81**
√	0.1	1.04	89.52
	0.01	5.91	64.69
	0.1	0.35	88.94
	1	0.06	90.26
	2	0.03	90.20

2.6.2.3 Training Details for Benchmark Results

For fair comparison, all our baselines (BWN/TWN, BWN-/TWN-QA) are fine-tuned from a pre-trained full-precision network. We leave the first and last layer at full-precision as in Rastegari et al. [57], Zhou et al. [62]. To obtain the benchmark results of our WrapNet, we follow a training pipeline, where we first warm-up our WrapNet with full-precision activations, and then we fine-tune the network for quantized activations. We set the transition slope $k = 2$, and the initial overflow rate $p = 5\%$. The overflow penalty coefficients for CIFAR-10 and ImageNet are 0.01 and 0.001, respectively.

For the CIFAR-10 results, we use ADAM as our optimizer with an initial learning rate of 0.001. For both warm-up and fine-tuning stages, we run 200 epochs, and the learning rate is divided by 10 every 60 epochs. For all the ImageNet results, we use SGD with momentum 0.9, weight decay 1×10^{-4} as our optimizer. We run 60 epochs for both warm-up and fine-tuning stages, where the initial learning rate is 0.01, which is divided by 10 at (20, 40, 50) epochs. We

note that, due to the depth of ResNet, we select a fixed quantization step-size for all the layers, where the average initial overflow rate is around 5%. As a result, the overflow penalty is also imposed during the warm-up stage for ResNet experiments.

2.6.3 Hardware Analysis

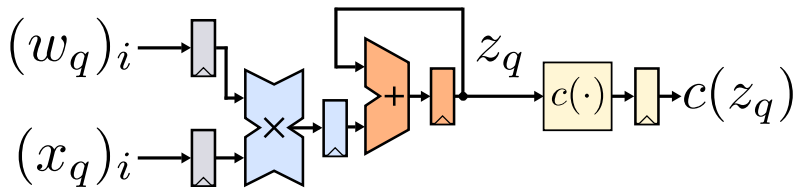


Figure 2.4: Multiply-accumulate (MAC) unit, together with cyclic activation function $c(\cdot)$, implemented for hardware analysis.

Figure 2.4 shows the multiply-accumulate (MAC) unit supplied in TSMC 28nm CMOS. The MAC unit multiplies two scalars and accumulates these products using an adder. To perform this functionality, the MAC unit is composed of multiplication, accumulation, and auxiliary circuitry, colored in Figure 2.4 with blue, orange, and gray, respectively. Clock distribution circuitry is not shown, but is included in our results as part of the auxiliary circuitry. Furthermore, we have implemented the cyclic activation function in hardware, colored in Figure 2.4 with yellow, which is only used together with low-bit accumulators. To achieve lower cycle times (i.e., faster operation frequencies), as well as to separate the multiplier's and accumulator's critical paths, we introduced a pipeline register between the multiplier and accumulator. For our implementation results, we consider this pipeline register as part of the multiplication circuitry.

We implemented the circuit in Figure 2.4 using different bitwidths for the multiplier (8-bit$\times$ 8-bit or 3-bit$\times$ 1-bit) and the accumulator (32-bit or 8-bit). When using the 8-bit$\times$8-bit multiplier with the 32-bit accumulator, we use 16 bits for the multiplier's output register to represent all

possible products. When using the 8-bit $\times$ 8-bit multiplier with the 8-bit accumulator, we use 8 bits for the multiplier's output register, since the accumulator does not support larger bitwidths. When using the 3-bit $\times$ 1-bit multiplier, we use 4 bits for the multiplier's output register, regardless of the accumulator's bitwidth. The cyclic activation is only implemented when using the 8-bit accumulator, for both multiplier's bitwidth. We implemented the cyclic activation for slopes of $k = 2$ and $k = 4$.

The four different MAC units were synthesized using Synopsys Design Compiler (DC), and automatically placed-and-routed using Cadence Innovus. Power analysis was done using Cadence Innovus with stimuli-based post-layout simulations at 0.9V and 25°C in the typical-typical corner. For the stimuli, we used weights and activations extracted from a layer of the ResNet-18 network. Tables 2.11, 2.12, and 2.13 show the implementation results from Figure 2.2 in tabular form. Note that throughput is computed as $2/$cycle time, as the MAC unit completes two operations (multiplication and accumulation) in a single clock cycle. However, in Figure 2.2, we decided to report cycle time so that, for all metrics presented (cycle time, area- and energy-efficiency), a lower value corresponds to a better performance. Note that circuits with a higher throughput (which corresponds, in this case, to a lower cycle time) often result in higher area and power consumption. As a matter of fact, dynamic power consumption is directly proportional to operation frequency (i.e., 1/cycle time). Thus, to perform a fair comparison, we have normalized the area and power reported in Table 2.11 by the throughput achieved, resulting in the area- and energy-efficiencies reported in Tables 2.12 and 2.13, respectively.

Table 2.11: Hardware implementation results for one multiply-accumulate (MAC) unit in 28nm CMOS

Bits			Cyclic act.	Cycle time	Throughput	Cell area	Power
Act.	Weight	Acc.	slope k	(ns)	(Gops)	(μm^2)	(mW)
8	8	32	–	0.31	6.5	1 298	2.78
3	1	32	–	0.29	7.0	732	1.90
8	8	8	2	0.24	8.3	521	1.60
8	8	8	4	0.25	8.1	523	1.64
3	1	8	2	0.24	8.3	290	0.93
3	1	8	4	0.24	8.3	285	0.87

Table 2.12: Area breakdown of one multiply-accumulate (MAC) unit in 28nm CMOS

Bits			Cyclic act.	Cell area efficiency (μm^2/Gops)				
Act.	Weight	Acc.	slope k	Multiplier	Accumulator	Cyclic act.	Auxiliary	Total
8	8	32	–	96 (48%)	91 (46%)	–	12 (6%)	199
3	1	32	–	3 (2%)	93 (89%)	–	9 (9%)	105
8	8	8	2	31 (49%)	12 (19%)	10 (16%)	10 (16%)	63
8	8	8	4	33 (50%)	12 (19%)	8 (13%)	12 (18%)	65
3	1	8	2	2 (5%)	17 (46%)	15 (41%)	3 (8%)	37
3	1	8	4	2 (5%)	18 (52%)	12 (35%)	3 (8%)	35

Table 2.13: Energy breakdown of one multiply-accumulate (MAC) unit in 28nm CMOS

Bits			Cyclic act.	Energy efficiency (fJ/op)				
Act.	Weight	Acc.	slope k	Multiplier	Accumulator	Cyclic act.	Auxiliary	Total
8	8	32	–	144 (34%)	173 (40%)	–	111 (26%)	428
3	1	32	–	10 (4%)	197 (73%)	–	64 (23%)	271
8	8	8	2	48 (25%)	29 (15%)	17 (9%)	98 (51%)	192
8	8	8	4	53 (26%)	28 (14%)	17 (8%)	105 (52%)	203
3	1	8	2	8 (7%)	42 (37%)	23 (20%)	42 (36%)	115
3	1	8	4	6 (6%)	42 (40%)	24 (23%)	33 (31%)	105

2.6.4 Using more weight bits

Since ARM provides arithmetic operations that handle multiplication between various 8-bit numbers in parallel, we further conduct experiments in which more bits are used for weight quantization. Table 2.14 displays the classification accuracy, as well as the overflow rate of the final models. Surprisingly, in some cases, we may have a lower overflow rate even when using more bits for the weight quantization. We also collect the accuracy degradation from the full

precision network. Our results show that the best performance is achieved when we use 4-bit weights, which is close to the full-precision result (around 0.7% degradation).

Table 2.14: Results for WrapNet with more bits for weight quantization, where we use ternary weights for 2-bit.

Bits	Overflow Rate	Accuracy	Degradation
1	1.24%	90.81%	1.64%
2	0.12%	91.14%	1.31%
3	0.02%	91.55%	0.90%
4	0.04%	**91.73**%	0.72%
5	0.4%	91.20%	1.25%

Chapter 3: Memory Efficient Training with Quantized Model

3.1 Introduction

End-to-end deep models are widely used in modern automatic speech recognition (ASR) applications [80, 81, 82]. These models achieve excellent recognition performance, but their performance depends heavily on the size of the model and the amount of training data. To increase the amount of training data while protecting user privacy, federated learning (FL) [83] has become a popular learning framework by training models on users' devices without sharing their data with a central server.

However, deploying huge models, such as the Conformer model [84] with approximately 130 million parameters, on edge devices with constrained memory capacities is challenging. Recently numerous studies have concentrated on reducing latency and model size while preserving model quality including pruning techniques [85, 86, 87], online model compression [88] and federated dropout [85]. One line of research investigates quantization aware training [89, 90] of ASR models on a centralized server. These methods have quantized the model to low-precision such as Int-8 and Int-4 to reduce the model size and improve the inference latency by allowing low-bit matrix operations. Specifically, in the forward propagation, the quantized variables are derived from the original full-precision variables to mimic the inference computation; and in the backward propagation, gradients are calculated with respect to the full-precision model for the

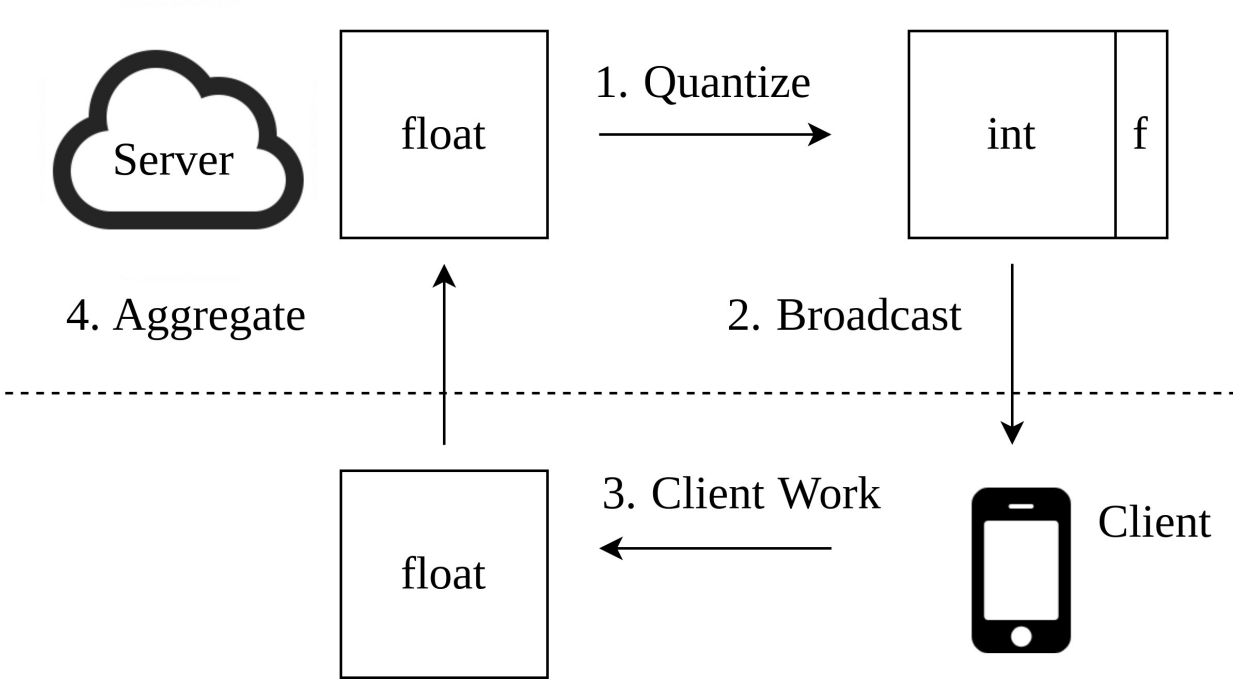


Figure 3.1: A federated round of FedAQT. The server quantizes the global model and broadcasts the quantized model to clients. Clients use the quantized model to compute the model deltas which are sent back to the server for aggregation and update of the global model.

model update in the learning process. Because the full-precision variables have to be loaded into memory in the quantization aware training, directly applying it does not reduce the memory consumption on the client's devices of FL.

In this work, we propose FedAQT, an accurate quantized training framework with FL that is designed to be memory-friendly to edge devices. Figure 3.1 illustrates the learning process of our method. Similar to the existing quantization method, we keep a full-precision global model on the server. However, during the broadcast stage, we quantize the model and only distribute the low-bit weights and their float quantization scales to the clients' devices. This dramatically reduces the training memory usage on clients' devices as well as the transportation load between server and clients. On the client computation, for the forward propagation, low-bit matrix multiplication is applied on the quantized weights and activation variables. For the backward propagation, due to

the discontinuity of the integer values for gradient computation, we de-quantize the weights and calculate the gradients in full-precision under gradient checkpointing [91] method to bound the memory usage. Finally, we aggregate all the float gradients and update the global model on the server. We show that FedAQT can achieve comparable accuracy against full-precision training while significantly reducing memory usage on devices. Our main contributions are illustrated as below:

- Our work is the first to propose a quantization aware training framework that trains low-bit networks directly (without a full-precision model) on edge devices under FL setting and hence opens up the possibility of training larger and more powerful ASR models under FL.

- To align our method with quantization aware training, we introduce a gradient relay between FL clients and server. We show that for FedSGD [92] the model updates calculated by gradient relay are the same as quantization aware training on a centralized server.
- We analyze the trade-off between model quality and quantization bit-width. We observe that conformer-like ASR models have no (minor) accuracy degradation with int8 (int4) weights on Librispeech data under FL setting.

3.2 Related work and Preliminaries

3.2.1 Quantization Aware Training

Quantization methods aim to decrease the model size and accelerate the inference speed of neural networks by using low-precision parameters and arithmetic operations. Binary, ternary, and uniform quantization are the most common choices and have been hardware-friendly [14,

54, 65]. To avoid accuracy drop, quantization aware training is usually applied for end-to-end training, where quantization operations are inserted into linear layers such as convolution layers and full-connected layers. Formally, for a uniform quantizer Q_b with target bit-width b, we quantize the float tensors $\mathbf{w}$ by $Q_b(\mathbf{w}) = round(\mathbf{w}/\mathbf{s}_w)\mathbf{s}_w$. Here, we assume the distribution of input tensor values is symmetrical and set the zero-point as 0 for convenience. We denote the integer representation of $\mathbf{w}$ as $\mathbf{q}_w$, and the quantization representation $\mathbf{w}_q$ can be expressed as $\mathbf{w}_q = Q_b(\mathbf{w}) = \mathbf{s}_w\mathbf{q}_w$ where $\mathbf{s}_w$ is the quantization scale, which controls the trade-off between range and the precision of the quantization representations. In practice, the scale is usually achieved dynamically during training by $\mathbf{s}_w = \max(abs(\mathbf{w}))/(2^{b-1}-1)$, where b is the bit-with.

During end-to-end training, since the rounding function has zero gradients almost everywhere, we follow Hubara et al. [54], Ding et al. [89] to use a straight-through-estimator (STE) to set the gradients of the round function as identity and apply backpropagation. Formally, given a loss function $\mathcal{L}$ and the learnable weight $\mathbf{w}$, the derivative respect to $\mathbf{w}$ is

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}_q} \cdot \frac{\partial \mathbf{w}_q}{\partial \mathbf{w}}. \tag{3.1}$$

When quantization scales are treated as constants [14, 89], the derivative of quantized representations $\mathbf{w}_q$ respect to the float parameter $\mathbf{w}$ is identity as well, by using the chain rule and STE. As a result, the gradient of the full-precision and quantized variables are the same in such cases.

In addition, recent quantization methods [66, 93] consider the quantization scale as a function of the float variable $\mathbf{w}$ thus backpropagating the gradient through the scale. Again, following

the chain rule, the gradient of the full-precision weight will be

$$\nabla \mathcal{L}(\mathbf{w}) = \nabla \mathcal{L}(\mathbf{s}_w) \odot ([\mathbf{w}]_{\max} \odot \frac{\text{sign}(\mathbf{w})}{b'}) + \nabla \mathcal{L}(\mathbf{w}_q) \tag{3.2}$$

$$\nabla \mathcal{L}(\mathbf{s}_w) = \sum \nabla \mathcal{L}(\mathbf{w}_q) \odot (\mathbf{q}_w - \mathbf{w}/\mathbf{s}_w) \tag{3.3}$$

Here $b' = 2^{b-1} - 1$, $\odot$ denotes elementwise production of two matrix, $[\mathbf{w}]_{\max}$ denotes for a matrix that has value 1 for the positions of maximum and 0 otherwise. Intuitively, by passing through the scales, we penalize the outliers of the float variables and adjust the scales accordingly, which improves the performance especially when we have extreme-low-bit such as 2-bit [66, 93]. In this work, we extend Ding et al. [89] to deploy the quantization method on both FL server and clients.

3.2.2 Quantization in Federated Learning

In the realm of FL, many works have explored the opportunity to compress the communication by quantization methods [94, 95]. The online model compression [88] sends quantized models to clients in FL but still uses full-precision variables in computation. Besides the compression during the communication, recent work [96] proposes a quantization method that is robust to aggregate local updates with various bit-widths. In addition, fixed-point inference is explored in Macha et al. [97] to reduce execution time. However, these methods still follow the typical quantization aware training approach, where a full-precision model is kept during on-device training. On the contrary, our training framework enables training native quantized variables directly and retain the full-precision model on the server only, thus leading to memory reduction.

3.3 Method

3.3.1 Federated Accurate Quantized Training

Figure 3.1 illustrates one round of FL using our framework. We explain the four federated training steps: Broadcast, Client Computation, Aggregation, and Model update.

Broadcast. As illustrated in Figure 3.1, we always keep a float global model on the server. For each FL round, we first quantize the *float* model into the *low-bit* one and their *float* quantization scales. Similar to Ding et al. [89], we only quantize the encoder of the ASR model and leave the decoder part full-precision to avoid an accuracy drop. Then only the *low-bit* and the *float* quantization scales will be distributed to the clients.

Client Computation. At the client training stage, each device trains the quantized model only with its own training data. Compared to standard FL and quantization aware training, our method only loads the *low-bit* variables into the memory, and calculates their respective *float* gradients on the fly, which dramatically reduces the memory usage. In addition, our framework allows executing different training algorithms such as FedSGD and FedAVG [92] at this stage.

Aggregation. After local training, we aggregate the *float* updates from the clients, which is exactly the same as the typical FL. Consequently, most aggregation algorithms that are applicable to the conventional FL framework can also be implemented in our proposed method, such as secure model aggregation [98].

Model Update. Finally, the server aggregates the model updates from the clients in the current round and updates the float model. The server then moves to the next round and applies quantization on the updated model.

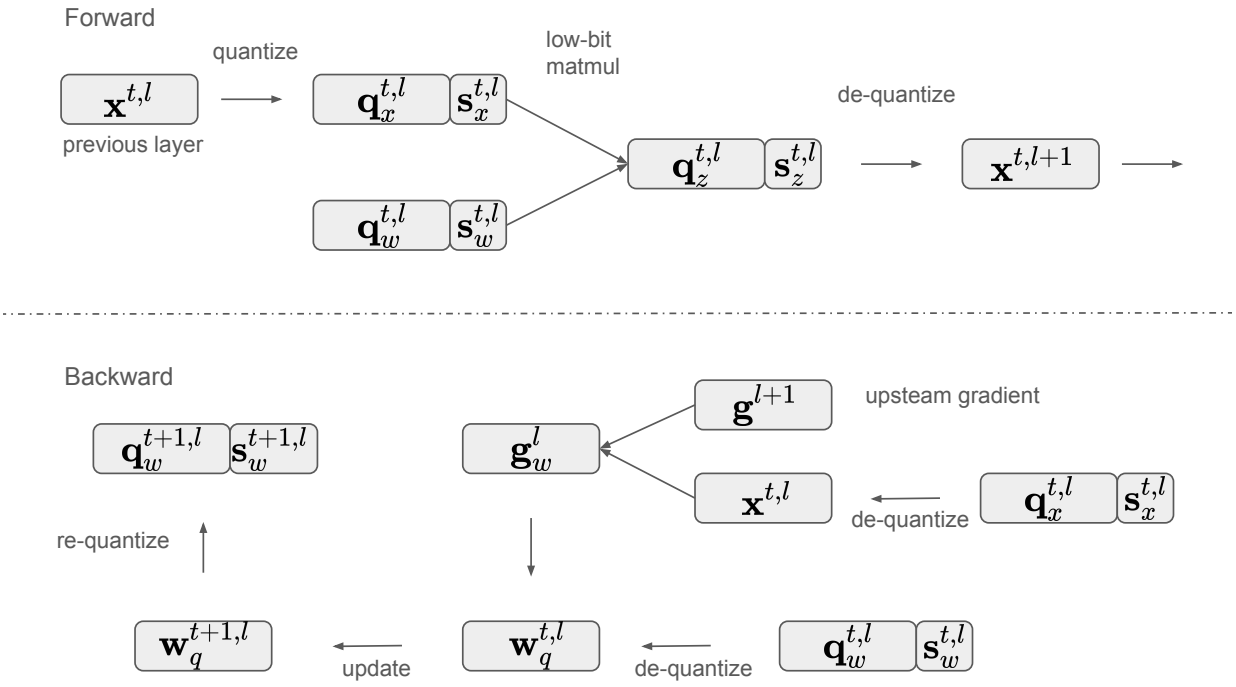


Figure 3.2: Training with quantized variables.

3.3.2 Training with quantized variables

We explain the client computation with the native quantized variables in this section. Figure 3.2 shows our training method. Note that our method is applicable on both FedSGD where clients compute only one batch of data without applying the gradients and FedAVG where the gradients are applied on local variables during multiple iterations.

In forward propagation of iteration t at layer l, the activation variable $\mathbf{x}^{t,l}$ is first quantized to $\mathbf{q}_x^{t,l}$ with $\mathbf{s}_x^{t,l}$. Then the quantized weights $\mathbf{q}_w^{t,l}$ and activations $\mathbf{q}_x^{t,l}$ are computed by the low-bit native matmul operation to generate the quantized outcome $\mathbf{q}_z^{t,l}$. In this way, we reduce the memory consumption by avoiding the full-precision operations. The $\mathbf{q}_z^{t,l}$ and the scale $\mathbf{s}_z^{t,l}$ are then de-quantized for the activation computations.

In the backward propagation, once the activation variable $\mathbf{x}^{t,l}$ is de-quantized from $\mathbf{q}_x^{t,l}$ and

$\mathbf{s}_x^{t,l}$, the gradient of weight $\mathbf{w}^{t,l}$ can be derived from the upstream $\mathbf{g}^{l+1}$ and $\mathbf{x}^{t,l}$ with the STE method [54, 89]. At this step, the gradient $\mathbf{g}_w$ can be sent back to the server as the model updates if we only consider the quantization scale non-trainable. For FedAVG algorithm, we need to apply the gradients on the local variables that are quantized. Hence we first de-quantize the weights $\mathbf{w}_q^t$ from $\mathbf{q}_w^t$ and $\mathbf{s}_w^t$, and then apply the gradients $\mathbf{g}_w$ on it to generate $\mathbf{w}_q^{t+1}$ for the next iteration $t+1$. Then the variable is re-quantized to $\mathbf{q}_x^{t+1}$ with $\mathbf{s}_w^{t+1}$ for the computation of the next iteration. Note that by applying the gradients on local variables FedAVG consumes more memory than FedSGD. Therefore we recommend using FedSGD when memory consumption is concerned. In standard quantization aware training, the gradients are calculated with respect to the latent full-precision variables. However, in our method, clients only have the quantized variables to apply the gradients. To achieve the same gradients as in centralized quantization aware training, we introduce gradient relay in FedSGD in the next section.

3.3.3 Gradient Relay

We explain the gradient relay to compute the gradients of clients if considering the quantization scale as a function [66, 93]. The idea is that clients and server each compute a part of the overall gradients in FedSGD.

The gradient $\mathbf{g}_w$ in Figure 3.2 is sent from clients to server as model deltas. Based on Eq equation 3.1, when we consider the quantization scales non-trainable, the model updates are exactly the same as those for centralized quantization, where $\nabla\mathcal{L}(\mathbf{w}) = \mathbf{I} \cdot \nabla\mathcal{L}(\mathbf{w}_q) = \nabla\mathcal{L}(\mathbf{w}_q)$. However, according to Eq equation 3.2, when backpropagation is performed through the quantization scales, the gradients with respect to the full-precision weights $\nabla\mathcal{L}(\mathbf{w})$ differ

from those with respect to the quantized weights $\nabla\mathcal{L}(\mathbf{w}_q)$, where the former relies on the original floating-point weights to penalize the outlier. Although in FedAQT, we do not have the full-precision model on client devices, we retain these weights on the server side. Moreover, as we show in Eq equation 3.2 and Eq equation 3.3, the difference between the two gradients is

$$\nabla\mathcal{L}(\mathbf{s}_w) \odot ([\mathbf{w}]_{\max} \odot \frac{\text{sign}(\mathbf{w})}{b'}) \tag{3.4}$$

which can be obtained with $\nabla\mathcal{L}(\mathbf{w}_q)$ and $\mathbf{w}$ only. Therefore, after the aggregation stage, we can easily adjust the global model updates on the server by Eq equation 3.4, thus leading to the exact same results as centralized quantization.

3.4 Results

3.4.1 Experiment Setup

We conducted experiments on LibriSpeech corpus [99], which contains 960 hours of speech in the training set. The development (dev) and testing datasets also contain "clean" and "noisy" subsets. We simulate the real low-bit quantized training in the following manners. For FedSGD, as we discussed in Section 3.3.3 that training on quantized variables can achieve the same model updates as training on the latent float variables, we simulate FedAQT by distributing the latent float variables to the client and apply quantization directly during the client computation. We conduct experiments on a conformer-based ASR model with roughly 133M parameters in total. Instead of training from scratch, we pre-train our model on industry-scale data collected from different domains, which achieved a Word Error Rate (WER) of 6.0 on the dev-clean set. We then

finetune the model with our FedAQT framework for 6000 steps. For FedSGD, we follow the IID training regime, and for each round, we randomly sample 128 clients from the client pool, which consists of 28124 clients and 10 examples per client. We used the Stochastic Gradient Descent (SGD) optimizer for both client and server updates with a small batch size of 2 to simulate real-world applications. Since for each client, we only take one batch, we are roughly using one-fifth of the training data (2/10). For FedAVG simulation, since it has been shown that FL achieves similar results as centralized training in ASR tasks [85, 100], we emulate the IID FL process with centralized training on Conformer-L models following Gulati et al. [84], Ding et al. [89]. To verify the effectiveness of our method, we re-quantize the updated weight after every iteration to ensure that we will only see the quantized variables for the next iteration. We evaluate our method with Word Error Rate (WER) metric for model quality and peak memory usage for model efficiency.

3.4.2 FedAQT Results

FedSGD. Table 3.1 shows the WER for our FedAQT with various bit-width. On both subsets, our method is able to train a low-bit model (for both int4 and int8) with similar performance as a full-precision model. To demonstrate the memory efficiency of our method, we implement the real training process with fake gradients. On the broadcast stage, we distribute quantized variables, namely int8 variables and float scales, to the clients. During the client computation, instead of calculating the gradient for real int8 variables, we generate empty gradients as local updates and aggregate them to update the server model. We implement this training process for different precisions of our on-device model, including float32, float16, and int8 for fair comparisons.

Table 3.1: FedSGD simulation with FedAQT of various bit-width. We can achieve a similar WER to float model on Librispeech even with 4-bit weights.

Bit-width	test-clean	test-other	dev-clean	dev-other
Float32	4.8	10.2	4.7	10.3
int8	4.8	10.2	4.7	10.3
int4	4.9	10.3	4.7	10.3

Table 3.2 shows that using float16 variables saves roughly 160MB of memory. Additionally, our proposed framework can further reduce memory usage by 100MB when using int8 variables.

Table 3.2: Memory consumption for different variable precision for one-round FL on-device training.

Variable Dtype	Memory (MB)	Saving (MB)
Float32	734188	-
Float16	568012	166176
Int8	468312	265876

FedAVG. Table 3.3 displays the WER on Librispeech or both the development (dev) and test subsets on a Conformer-L model. Our proposed FedAQT can achieve the same WER as the full-precision model and the normal quantization aware training [89] methods with 8-bit weights on both the dev and test subsets

Table 3.3: Comparison of our FedAVG simulation train on 8-bit conformer only with other quantization methods.

Method	test-clean	test-other	dev-clean	dev-other
Float32	2.0	4.4	1.9	4.3
QAT [89]	2.0	4.5	1.9	4.3
FedAQT	2.0	4.4	1.9	4.3

3.5 Conclusions

In this work, we propose a novel FedAQT quantized training framework with FL to train low-bit ASR models directly on client devices without storing a full-precision model. We show that our framework can train low-bit models with comparable performance to full-precision models. In addition, the 8-bit model under our framework only uses 60% of the memory, which verifies the effectiveness of our proposed method.

Chapter 4: Improve On-Device Latency for Mobile Vision Transformers

4.1 Introduction

Transformer-based models [12] have been widely used and achieve remarkable performance for many computer vision tasks, such as image classification, object detection, and semantic segmentation. However, the computational complexity of these methods is significant due to the computational cost of Multi-Head Self-Attention (MHSA), the foundational mechanism in transformers, which scales quadratically to token length. This cost is a barrier to deploying vision transformers in resource-limited mobile devices.

Latency improvements for transformers have been widely explored by many [21, 24, 27, 32, 34, 101, 102, 103, 104, 105, 106]. One line of research employs a CNN-Transformer hybrid architecture to reduce the computational complexity. These hybrid methods apply convolutional blocks, or other cheap operations, at the shallow stages of a network. Then, attention layers are used in later stages where the resolution of feature maps is lower, resulting in shorter token sequences. Although these hybrid architectures significantly improve on inference speed, their accuracy is degraded since only a few transformer blocks are used. In addition, the computational cost of their attention layers remains quadratic, making them impractical for use on the high-resolution inputs commonly needed for accurate object detection and semantic segmentation. The second line of research seeks to reduce the cost of the attention mechanism directly [31,

32, 33, 34, 35, 107, 108]. Among these methods, transformers using local self-attention (LSA) achieve promising performance in vision tasks. Instead of employing global attention for all the tokens, these local methods apply self-attention within individual token partitions, such as within window partitions [32] or horizontal and vertical stripes [34]. As long as the local size is fixed, the computational cost will be linear with respect to token length.

However, some recent works [21, 106, 108] point out that these local self-attention (LSA) mechanisms are not well-equipped to handle mobile devices, such as iPhones, which use the Apple Neural Engine (ANE). Existing LSA implementations with ANE have high latency, if a model is supported at all. In this work, we first dissect LSAs and find that inefficient in-memory partition and shifting operations are the main bottlenecks for on-device latency. Because some approaches are not out-of-the-box compatible with ANE, we adapt these models using new reshaping tricks and remove non-supported shifting operations to allow for accurate benchmarking and comparison of LSA. However, even with these improvements via reshaping tricks, LSA is still slow on-device [21] and the removed shifting operations cause a degradation in accuracy [32].

To alleviate the aforementioned problems, we propose a simple but effective local self-attention method that does not rely on problematic partitioning methods that are incompatible with or slow on embedded frameworks. In brief, we flatten feature maps in row-column or column-row order and partition the sequence with a predetermined local length. We alternate between row-column and column-row order in consecutive transformer blocks to enhance the connection between these token partitions. This simple partition method can be computed very efficiently on-device.

We apply our proposed LSA in state-of-the-art mobile-friendly architectures [101, 106] to build an efficient and general-purpose backbone for multiple vision tasks. To demonstrate the

effectiveness of our proposed model, we benchmark on various vision tasks, including image classification, object detection, and semantic segmentation. In addition, we measure the latency of our method under popular mobile frameworks, CoreML and TensorRT, on real-world mobile devices, such as iPhone ANEs. We show that our method can achieve comparable performance on low-resolution tasks, such as image classification, while running $1.5 \times faster$ than the SOTA methods in object detection with similar precision.

We summarize our main contributions as follows:

- We first indicate that local self-attention itself is mobile-friendly and efficient. The bottleneck of on-device deployment mainly focuses on ill-supported operations used for token partition and communication among these partitions.

- With simple but critical modification, our proposed LSA is entirely mobile-friendly, and we demonstrate the efficacy of our method on multiple vision tasks. Our method can achieve comparable performance to global MHSA while dramatically reducing computational costs and on-device latency.

- When used on SOTA mobile architectures, our efficient transformer can achieve the best trade-off between accuracy and latency, especially for tasks requiring high-resolution inputs.

- Multiple ablation studies are conducted to show and understand the effectiveness of our proposed mobile-friendly LSA mechanism.

4.2 Related Work

Transformer Variants. Transformer networks [109] were first proposed for the natural language processing (NLP) domain. Vision transformers (ViTs) [12] successfully adapted the self-attention mechanism to vision tasks where input images are partitioned into patches, and each patch is treated as a token. Although ViTs can perform well on many vision tasks, they usually require large-scale datasets and massive computational resources. As a result, a lot of work has been devoted to improving the training and model efficiency of ViTs. DeiT [22] introduces a novel distillation method to train ViTs efficiently even without extra datasets. Methods like T2T and TNT [23, 24] propose better tokenizers to encode input images into patch tokens efficiently.

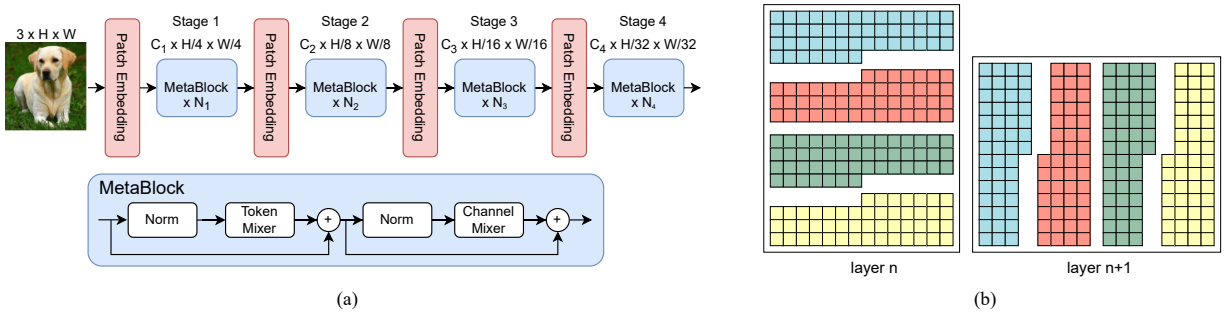
To further enhance the performance and efficiency of transformers, strategies from Convolutional Neural Networks (CNN) are widely adopted [21, 103, 106, 110, 111, 112, 113, 114]. Convolutional units gradually reduce the number of tokens by down-sampling or pooling [31, 32, 115]. Additionally, convolutional blocks or other cheaper operations have been introduced into transformers that achieve lower on-device latency. In MobileViT [103], self-attention blocks are adopted into MobileNet to achieve a lightweight transformer for general purposes. Mobile-Former [27] benefits from the local processing of MobileNet and the global interaction of transformers to achieve a hybrid model with extremely low FLOPs while maintaining high performance. To optimize the on-device latency, TRT-ViT [114] and EfficientFormer [21] stack only a few attention layers at the final stage, where the feature resolution is low. Next-ViT [106] refines the hybrid strategy where several efficient spatial reduction attention layers are inserted into early stages as well.

Efficient Self-Attention Mechanism. PVT [31] introduces a spatial reduction self-attention,

where local tokens are grouped and self-attention is applied to these reduced tokens. The Swin transformer [32] partitions feature maps into non-overlapping windows and then self-attention is conducted within each window. As long as the window size is fixed, the computational cost of local self-attention in a Swin transformer is linear in token length. To improve the connectivity between windows, Swin further shifts windows in each consecutive layer. Although the Swin transformer greatly reduces FLOPs and maintains high performance, the shifting operation is memory-consuming, especially on mobile devices. In addition, although the window partition is efficient for data center GPUs, this operation is also not well-supported on mobile devices such as iPhones using the iOS ANE, where a combination of expensive reshaping operations are needed [21].

Following Swin [32], CSWin [34] computes local self-attention in parallel within the horizontal and vertical strips of feature maps, which reduces computational costs to sub-quadratic but does not achieve linear scaling. In addition, this parallelization mechanism requires more reshaping operations. Recently, Twins [35] combines local and group self-attention to achieve a better spatial reduction self-attention for LSA. MaxVit [33] swaps the axis of the window partitions to achieve grid partitions, which removes the shift operation and can be implemented efficiently on TPU devices.

None of these architectures are well-equipped to handle resource-limited devices, which makes it difficult to run them in terms of both implementation complexity and latency. In this work, we revisit these methods and propose an efficient and effective method based on operations optimized directly for current mobile frameworks.



(a) (b)

Figure 4.1: (a) Overview of our architecture. We adopt hierarchical architecture with 4 stages of gradually reduced feature resolutions. Within each stage, several MetaBlocks are used where in general we use cheap operations, i.e., depth-wise convolution, as token mixers in the first two stages and attention mechanisms for the latter two stages. (b) Examples of our proposed mobile-friendly local self-attention on a 14×14 feature map, with local size 49. We horizontally and vertically split the tokens into non-overlapping partitions for layer n and layer $n+1$, respectively.

4.3 Method

We first describe the overall architecture used in our experiments (Sec. 4.3.1). We then address the on-device latency bottlenecks for existing local self-attention mechanisms and introduce our proposed method to address these problems (Sec. 4.3.2). We also include details of our models for a range of model sizes (Sec. 4.3.3).

4.3.1 Base Architectures

The overall architecture of our model (shown in Figure 4.1) follows MetaFormer [101], with minor modifications as suggested by Efficientformer [21] for faster on-device inference. Given an input image $I \in \mathbb{R}^{H \times W \times 3}$, a stem with two successive convolution layers of size 3×3 is applied to compute patches, also referred to as tokens, with dimension $\frac{H}{4} \times \frac{W}{4} \times C_1$, where $\frac{H}{4} \times \frac{W}{4}$ denotes the token length and C_1 is the dimension of each token embedding. Following the hierarchical structure from recent transformers [31, 32, 101], our model has four stages,

and the token length is gradually downsampled by a convolutional layer at the end of each stage. Respectively, each stage has $\frac{H}{4} \times \frac{W}{4}$, $\frac{H}{8} \times \frac{W}{8}$, $\frac{H}{16} \times \frac{W}{16}$, and $\frac{H}{32} \times \frac{W}{32}$ tokens with embedding dimension C_1, C_2, C_3, C_4.

Within each stage, several MetaBlocks are used, where each block consists of a token mixer, exchanging information among tokens, and a channel mixer, exchanging information among channels. Token mixers can be either cheap operations like pooling or depth-wise convolution, or expensive and computation heavy units like multi-head self-attention (MHSA). A channel mixer is usually a 2-layer MLP with non-linear activations. Similar to other hybrid methods [21, 101, 113, 114], we use cheap operations such as pooling and depth-wise convolution as token mixers in the first two stages, and use self-attention, including the proposed mobile-friendly local self-attention, in the latter two stages.

4.3.2 Mobile-Friendly Local Self-Attention

Standard vision transformers [12, 22] use global self-attention to enlarge the receptive field and capture long-range context. However, the computational complexity of global self-attention is quadratic in the sequence length which is prohibitive for mobile deployment, especially for tasks requiring higher input resolution. To reduce complexity, existing methods [32, 33, 34] conduct local self-attention for the tokens within a specific partition. For example, Swin [32] performs self-attention within non-overlapping windows and the window partition is shifted in successive blocks to enlarge the receptive field. CSWin [34] performs self-attention in horizontal and vertical strips in parallel within each transformer block with a dynamic stripe width based on the input resolution for each stage. Instead of using shifted windows, MaxViT [33] applies

Token mixer	Original	Ours iOS15.2/16.0	Reshape Numbers
Attention	3.8	3.8/3.8	32
Window	NS	2.8/2.8	62
CS Window	NS	14.6/3.2	170
Mobile-Friendly (Ours)	2.7	2.7/2.7	32

Table 4.1: Latency (ms) and reshape operation required for transformers with various LSA mechanisms. These methods are not supported (marked as NS), on iPhone ANE with the original implementation. Latency increases significantly when more reshape operations are introduced, especially on previous iOS versions.

local self-attention within grids, namely dilated windows, after each window attention. Although these methods can dramatically reduce computation complexity, which is linear $O(L)$ [32, 33] or $O(L^{1.5})$ [34] to the token length L, deploying these methods on mobile devices is still challenging. Many related works [21, 106, 108] point out that such methods are not well supported on real devices, such as on iPhones using the ANE, which is a major testbed for our study. We revisit these methods and find the following observations.

Bottlenecks for On-device Latency. There are mainly two reasons that impede the deployment of these LSA networks on mobile devices. Firstly, some operations, such as shifting the window, are memory-consuming and not supported on resource-constrained devices. Although we can easily fix this by removing such an operation, the network may suffer drops in accuracy [32]. Secondly, all of these methods rely heavily on window partitioning which requires complicated reshape operations that are not natively supported as atomic operations on devices such as iPhone ANE, as they significantly reconfigure the memory layout of these tensors. We re-implement these partitions with a reshape trick that makes these models runnable on-device. However, this implementation trick actually introduces a large number of simple reshape operations, which have been shown to be slow on-device [21]. Table 4.1 shows the latency for networks with different

attention mechanisms on an *iPhone 13 pro* with different iOS versions. We can see that window partitioning doubles the number of reshape operations required for the transformer blocks using global self-attention, where reshaping is used to convert re-stride 4D input to 3D. In addition, the cross-shape strip partition requires additional movement, since horizontal and vertical strips are calculated in parallel within a block. Although these blocks can be deployed faster with the latest iOS version, the latency is significantly slower for previous versions. Therefore, to achieve low on-device latency, we propose a mobile-friendly and effective local self-attention mechanism, which is well-supported on-device and without any additional reshape operations.

Mobile-Friendly LSA. To address the aforementioned issues, we propose a local self-attention that is fully based on operations friendly to mobile devices and requires no additional reshapes (See in Table 4.1). We split adjacent tokens into non-overlapping partitions and then self-attention is only conducted individually within each local partition. Formally, let $\mathbf{X}^n \in \mathbb{R}^{L\times C}$ be the encoded features at the n-th transformer layer, where L denotes the total token length and C is the dimension of token embeddings. To obtain the features at the $(n+1)$-th transformer layer, we first partition $\mathbf{X}^n$ into l sequences,

$$\mathbf{X}^n = [X_1^n, X_2^n, \cdots, X_l^n], \tag{4.1}$$

where each $X_i \in \mathbb{R}^{M\times C}$ and $M = L/l$ where M is the number of tokens in each partition. Then, MHSA is applied on these token partitions, and the results are concatenated to return the final output of the local self-attention,

$$Y_i^n = \mathrm{MHSA}(X_i^n),\ X_i^n \in \mathbf{X}^n \tag{4.2}$$

$$\mathbf{Y}^n = [Y_1^n, Y_2^n, \cdots, Y_l^n], \tag{4.3}$$

where $\mathbf{Y}^n$ will be passed to a feed-forward network and outputs the encoded features $\mathbf{X}^{n+1}$ for $(n+1)$-th layer.

A Pytorch-like pseudo code is presented in Appendix 4.6.1 (Alg. 2). Theoretically, given input tokens with resolution L and local length M, the computation complexity of the proposed attention mechanism is

$$\Omega(LSA) = 4LC^2 + 2MLC. \tag{4.4}$$

When M is fixed and relatively small, the complexity is linear in token length. In general, M can be any factor of the token length L, otherwise, additional padding tokens are required which would imply additional costs. By default, M is set to 49, which is the same as the Swin Transformer [32]. With the same local size, the theoretical complexity of our method is the same as Swin, namely $O(L)$, which is faster than spatial reduction attention (SRA) with complexity $O(L^2)$ and cross-shape window attention with complexity $O(L^{1.5})$. Notably, our method can be applied to other architectures as well, such as Swin [32] or PVT [31]. Here, we focus our experiments on MetaFormer [101] mainly due to its simplicity and efficiency in experiments.

Dimension Swap and Relative Position Bias. Swin Transformer [32] uses shifted window partitions in successive blocks to capture context from a longer range. Similarly, to introduce connections between token partitions, we swap the width and height dimensions of the feature map for successive blocks. Shown in Figure 4.1, tokens are horizontally split in layer n and then vertically split in the next layer $n+1$. Unlike the shifted window operation, dimension swapping can be efficiently achieved *with no additional costs* (See Alg. 2 in Appendix 4.6.1). Additionally, we adopt a relative position bias similar to Swin [32] in the local self-attention.

4.3.3 Architecture Variants

For fair comparisons, we provide three variants with the same channel numbers as the small-sized MetaFormer, and each network has 12, 18, and 24 layers in total, respectively. We also provide a model with the exact same channel numbers and block layers for each stage as Swin-S to show the effectiveness of our method. Depth-wise convolution [116] is used along with BatchNorm [117] in the first two stages and the proposed LSA, along with LayerNorm [118], is used in the latter two stages. Details and variants of our architecture can be found in Appendix 4.6.2.

4.4 Experiments

4.4.1 Datasets and Implementation Details

We implement our model with PyTorch [119] and the TIMM library [120]. All methods are benchmarked on ImageNet-1K [6] for image classification. For downstream tasks, MSCOCO [13] is used for object detection, and ADE20K [121] is used for semantic segmentation. For ImageNet training, we follow the training settings of related work [32, 101, 106]. We train the networks with the AdamW optimizer with a weight decay rate of 0.05 and batch size of 1024 for 300 epochs. The learning rate starts at 0.001 and decays under a cosine scheduler with 3 warm-up steps. Data augmentations such as random augmentation [122], Mixup [123], CutMix [124], and stochastic depth drop [125] are adopted as in previous methods [32, 101]. Unless otherwise stated, all ablation studies use the smallest variant with 12 layers, and the token mixer is a depth-wise convolution in the first two stages and self-attention in the last two stages. LayerNorm is

used as the normalization and we use a default local length of 49. To be noticed, for tasks such as image classification with input resolution 224×224, the size of the feature maps in the final block (7×7) will be the same as the default local length, which makes no difference for local and global self-attention. For all downstream tasks, we finetune the ImageNet-1K pre-trained model following [32, 106]. We use the Mask R-CNN [126] framework for object detection and Semantic FPN [127] for segmentation. More training details can be found in Appendix 4.6.3.

On-device latency is measured through the CoreML framework on an iPhone 13 Pro with iOS 16.0 with batch size 1, and with the TensorRT (TRT) framework on an NVIDIA A4000 GPU with batch size 8. The latency is evaluated at the scales, $224 \times 224, 512 \times 512, 1024 \times 1024$ for classification, segmentation, and object detection, respectively. Unless otherwise stated, the latency in all ablation studies is evaluated on an iPhone 13 Pro with iOS 16.0.

4.4.2 Evaluation of the proposed Local Self-Attention

4.4.2.1 Comparisons with Efficient Self-attention

We show the effectiveness of our proposed mobile-friendly LSA by comparing it to other efficient self-attention methods. We use the same default architecture, and only replace the token mixers in the third stage with the target attention method. Table 4.2 shows the Top-1 accuracy (%) on ImageNet for classification and the latency on CoreML. We can see that all LSA methods can achieve reasonable accuracy and latency with our re-implementation. In addition, our method is more accurate than Swin transformer (without shifting) and SRA with a similar inference speed. Ours has comparable accuracy to window and grid attention pairs (Win/Grid) from MaxViT [33] and cross-shape window attention from CSWin, with only half or even one-fourth of the reshape

operations, which demonstrates the effectiveness of our proposed method. More experiments on other downstream tasks such as segmentation and detection can be found in Appendix 4.6.4.

Methods	Accuracy	Latency	# Reshape
	Top-1 (%)	(ms)	
SRA	80.0	2.4	32
Swin	80.8	2.8	62
Win/Grid	81.3	2.8	62
CSWin	81.4	3.2	170
Mobile-friendly (Ours)	81.4	2.7	32

Table 4.2: Performance for different efficient attention mechanisms used with the MetaFormer architecture. We highlight the result of our proposed LSA in gray.

4.4.2.2 Comparisons with Global Attention

We compare our proposed LSA with global MHSA, where we use MHSA as token mixers in different stages. Table 4.3 shows that using MHSA only at the last stage, similar to EfficientFormer and TRT-ViT, may drop the performance of the model by a large margin. Compared to using MHSA at the latter two stages, Top-1 accuracy on ImageNet and the mAP on MSCOCO drops more than 2%, while mIoU on ADE20k segmentation tasks drops around 5%. Yet, by using the proposed LSA, we can achieve comparable performance on all these vision tasks. From Figure 4.2, we can see that our proposed LSA is much faster than using global MHSA, especially when the input resolution is high (ADE20K/MSCOCO). Therefore, our proposed LSA provides a better trade-off between model quality and efficiency.

Stage	ImageNet		ADE20k		MSCOCO	
	Accuracy	Latency	mIoU	Latency	mAP^{box}	Latency
CCCC	78.6	1.3	35.4	5.2	37.5	17.6
CCCA	79.0	1.9	37.0	9.3	41.3	47.1
CCAA	81.7	3.8	43.9	36.8	43.8	-
Ours	81.4	2.7	43.7	14.6	42.9	49.4

Table 4.3: Performance of using MHSA and LSA as token mixers in different stages. "C" denotes depth-wise convolution, "A" denotes MHSA and "L" denotes our proposed LSA. Our method can achieve comparable results to MHSA while dramatically reducing the latency (ms) on an iPhone ANE, especially for high-resolution inputs 1024×1024 which is too expensive to get a number for MHSA.

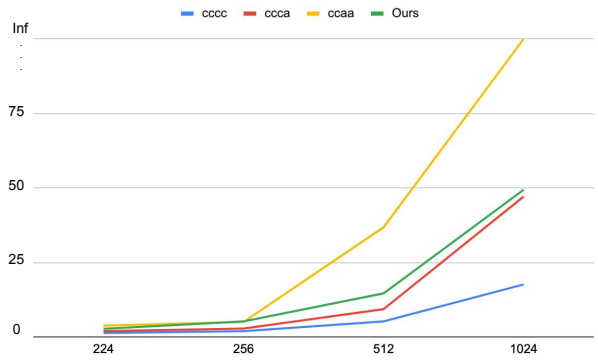


Figure 4.2: Latency (ms) of models using MHSA or our proposed LSA at different stages with inputs of resolution ranging from 224×224 to 1024×1024. The latency of LSA remains low when the resolution scales up.

4.4.3 Comparisons with SOTA Methods

4.4.3.1 Image Classification

Table 4.4 displays the top-1 accuracy on ImageNet as well as the latency on both TensorRT and CoreML. Rows are grouped by the block type applied in each network. As shown in Table 4.4, our proposed LSA can achieve comparable accuracy and latency compared to other SOTA mobile-friendly models, which use few attention layers. For example, compared to PoolFormer [101], which uses Pooling as token mixers, our model increases the latency by only around 50% while boosting the accuracy by at least 2% with the same architecture. Compared

to EfficientFormer, our method is as fast as the small-size variants, while roughly 15% faster and 0.3% better for the largest model - even though their model is trained with additional distillation which boosts accuracy by roughly 2%. Compared to Next-ViT, the accuracy is similar among all the variants, but our model can save around 30% in parameter count as well as strictly reduce FLOPs. Compared to other attention-heavily mobile models, such as MobileViTv2, with similar model sizes, our method is $2\times$ smaller in FLOPs and $2.2\times$ faster on CoreML while having the same accuracy. Notably, with the same model configuration as Swin-S, our model is 0.5% better in accuracy. Meanwhile, all these LSA-only methods are unavailable on ANE, with non-supported operations, and CSWin is roughly 40% slower on TensorRT under similar accuracy. These results emphasize again our proposed LSA is plausible on-device and brings optimism to the idea of reintroducing LSA for mobile transformers.

4.4.3.2 Object Detection and Semantic Segmentation

Table 4.5 shows the mean average precision (mAP) and mean intersection over union (mIoU) for object detection on MSCOCO and segmentation on ADE20K. Since higher resolution inputs are required for these frameworks, we also measure the latency on CoreML for backbones with different resolutions. As seen in Figure 4.3, a clear trade-off is observed, especially for object detection which has the highest input resolution. Even though SRA, which is used in Next-ViT, can reduce the token length, the computational cost is still quadratic to token size which worsens latency when the input resolution is high. Meanwhile, the cost for the proposed LSA is linear, which makes our model scale better with respect to resolution. Specifically, with similar inference time, our model outperforms EfficientFormer-L3 by 1.2% in box mAP and

Backbone	Block Type	Distill	Param (M)	FLOPs (G)	Latency (ms) TRT	Latency (ms) CoreML	Acc. Top-1
ResNet101	Conv		44.6	7.9	4.5	3.5	80.8
MobileNetv2-1.4	Conv		6.1	0.6	1.7	1.1	74.7
EfficientNet-B0	Conv		5.3	0.4	2.3	1.5	77.1
EfficientNet-B3	Conv		12.0	1.8	6.7	5.6	81.6
EfficientNet-B5	Conv		30.0	9.9	25.3	22.2	83.6
DeiT-T	MHSA	✓	5.9	1.2	2.0	5.1	74.5
DeiT-S	MHSA	✓	22.5	4.5	3.8	11.6	81.2
Swin-T	LSA		29.0	4.5	5.2	-	81.3
Swin-S	LSA		50.0	8.7	8.4	-	83.0
CSwin-T	LSA		23.0	4.3	7.9	-	82.7
CSwin-S	LSA		35.0	6.9	12.3	-	83.6
PoolFormer-S12	Pool		12.0	2.0	3.9	1.5	77.2
PoolFormer-S24	Pool		21.1	3.4	7.1	2.4	80.3
PoolFormer-S36	Pool		31.2	5.0	10.4	3.4	81.4
MetaFormer-S12(LSA)	Hybrid		16.6	2.5	3.7	2.1	81.1
MetaFormer-S18(LSA)	Hybrid		22.5	3.4	4.7	2.9	82.3
MetaFormer-S24(LSA)	Hybrid		30.6	4.7	6.2	3.8	82.8
MetaFormer-SwinS(LSA)	Hybrid		51.3	8.5	8.2	5.6	83.5
MobileViTv2-1.0	Hybrid		4.9	1.8	4.5	3.5	78.1
MobileViTv2-2.0	Hybrid		18.5	7.5	7.7	6.8	81.2
EfficientFormer-L1	Hybrid	✓	12.3	1.3	2.2	1.5	79.2
EfficientFormer-L3	Hybrid	✓	31.3	3.9	4.2	2.6	82.4
EfficientFormer-L7	Hybrid	✓	82.1	10.2	8.1	6.5	83.3
CMT-T	Hybrid		9.5	0.6	4.4	5.4	79.1
CMT-XS	Hybrid		15.2	1.5	7.7	8.2	81.8
CMT-S	Hybrid		25.1	4.0	13.1	10.0	83.5
Next-ViT-S	Hybrid		31.7	5.8	4.3	3.3	82.5
Next-ViT-B	Hybrid		44.8	8.3	5.8	4.1	83.2
Next-ViT-L	Hybrid		57.8	10.8	7.3	4.8	83.6
Next-ViT-S (LSA)	Hybrid		28.2	5.5	4.3	3.4	82.7
Next-ViT-B (LSA)	Hybrid		39.8	7.8	5.7	4.8	83.5
Next-ViT-L (LSA)	Hybrid		51.4	10.1	7.2	5.7	83.9

Table 4.4: Comparisons using image classification on ImageNet-1K. We highlight results in gray that use our proposed LSA. FLOPs and latency are measured with input size 224×224.

0.9% in mask mAP. In addition, our larger variant (S24) outperforms EfficientFormer-L7 by 2% box mAP and 1.6% mask mAP while the latency is 60% less. A similar trend is observed for semantic segmentation as well, where our larger variant beats EfficientFormer-L7 by 1.5% in mIoU with a slightly faster inference speed. Compared to Next-ViT, our model is roughly $2\times$ faster, but with a noticeable performance drop. Again, we compared our model with Swin-S, under comparable model sizes, where we achieve close performance in both object detection and semantic segmentation which demonstrates that our model can serve as a general-purpose backbone for vision tasks.

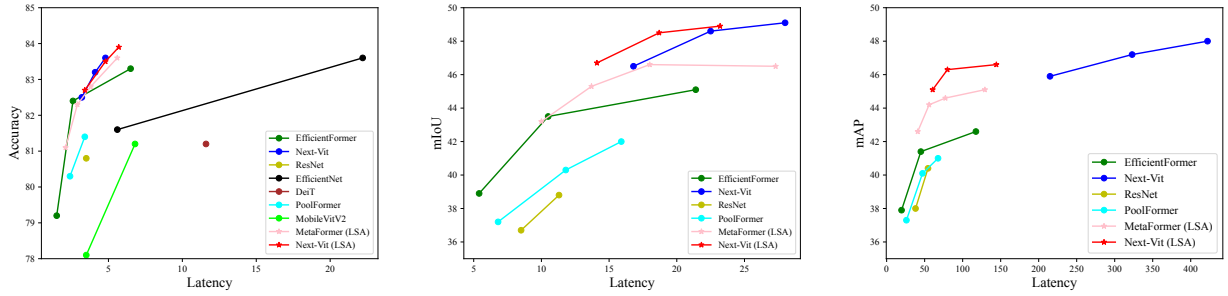
Backbone	MSCOCO			ADE20K	
	AP^b	AP^m	Latency	mIoU	Latency
ResNet101	40.4	36.4	54.4	38.8	11.3
Swin-T	42.2	39.1	-	41.5	-
Swin-S	44.8	40.9	-	45.2	-
EfficientFormer-L1	37.9	35.4	19.6	38.9	5.4
EfficientFormer-L3	41.4	38.1	45.0	43.5	10.5
EfficientFormer-L7	42.6	39.0	117.4	45.1	21.4
MetaFormer-S12(LSA)	42.6	39.0	40.9	43.2	10.0
MetaFormer-S18(LSA)	44.2	40.1	55.9	45.3	13.7
MetaFormer-S24(LSA)	44.6	40.6	77.3	46.6	18.0
MetaFormer-SwinS(LSA)	45.1	40.8	129.2	46.5	27.3
Next-ViT-S	45.9	41.8	214.9	46.5	16.8
Next-ViT-B	47.2	42.8	322.9	48.6	22.5
Next-ViT-L	48.0	43.2	422.1	49.1	28.0
Next-ViT-S (LSA)	45.1	41.0	60.5	46.7	14.1
Next-ViT-B (LSA)	46.3	42.3	80.0	48.5	18.7
Next-ViT-L (LSA)	46.6	42.4	144.3	48.9	23.2

Table 4.5: Comparison of object detection on MSCOCO and semantic segmentation on ADE20K. Latency is measured on CoreML for inputs with resolution 1024×1024 for object detection and 512×512 for segmentation.

4.4.3.3 Combining with Next-ViT

To further illustrate the efficiency and flexibility, we combine the proposed LSA with Next-ViT and apply the following modification: We replace all attention layers and their previous convolution layer with our proposed LSA, which allows swapping for dimensions in consecutive layers. In addition, we remove the complicated hybrid strategy within a transformer block and keep the channel dimension fixed within each stage. Table 4.4 and Table 4.5 show the results for Next-ViT and the variants with our proposed LSA in different vision tasks. For tasks with smaller input resolutions such as image classification, our model can achieve consistent improvement for all three variants with similar latency on CoreML, since the spatial reduction attention can aggressively shrink the token size to be as small as our local size. For tasks requiring higher input resolutions such as segmentation and object detection, our method is much faster and achieves

comparable results. As clearly illustrated in Figure 4.3, compared to other efficient networks, our Next-ViT variants can obtain significant performance boosts while enjoying low latency, which again shows the effectiveness and efficiency of the proposed LSA, especially for high-resolution applications.



(a) ImageNet Classification (b) ADE20K Segmentation (c) MSCOCO Detection

Figure 4.3: Trade-off between latency on CoreML and (a) classification accuracy on ImageNet, (b) mIoU for segmentation on ADE20K, (c) mAP for detection on MS-COCO. Latency is measured on input resolutions of $224 \times 224, 512 \times 512$ and 1024×1024 respectively. We compare our proposed LSA with other efficient models.

4.4.4 Ablation Studies

In this section, we conduct several ablation studies to highlight the benefits and crucial components of our proposed modified LSA.

4.4.4.1 Trade-offs between Local Size and Latency

We explore the trade-off between the local size used in the LSA and the latency on CoreML, as well as the classification accuracy. In Table 4.6, we collect the latency for blocks with various token mixers, including depth-wise convolution (DW-Conv), multi-head self-attention (MHSA), and proposed local self-attention (LSA) with different local lengths. The input resolutions are set as those used for different stages in image classification with the corresponding channel number.

Token Mixer	56x56	28x28	14x14	7x7
DW-Conv	0.5	0.46	0.41	0.44
MHSA	8.13	1.29	0.66	0.47
LSA-98	0.92	0.65	0.59	-
LSA-49	0.84	0.60	0.56	-
LSA-14	0.73	0.53	0.48	-
LSA-7	0.79	0.55	0.49	0.46

Table 4.6: Latency (ms) of LSA with different local lengths on input with the same resolution used in each stage for image classification. We also compare the latency with DW-Conv and MHSA.

As we can see in Table 4.6, the proposed LSA is notably faster than MHSA when the resolution is high, such as 56×56. However, when the resolution is as low as 7×7, the latency difference between MHSA and LSA is negligible, even compared with DW-Conv. Regarding accuracy, we can see from Table 4.7 that the accuracy drops slightly when shorter local partitions are used. In general, the longer local length, the slower the network but the better accuracy will get. More details can be found in Appendix 4.6.5.

To better understand the LSA mechanism, we evaluated models trained with a specific local size, ranging from 49 to 196 (full size), with various other local lengths. From Table 4.7, we observe that the pre-trained model works surprisingly well with the local size different from the one used during the training, especially for those trained with LSA. The accuracy drops roughly 1% when tested on a local size which is only one-fourth of the trained size. Even though the model is trained with global MHSA, it can still achieve reasonable accuracy when evaluated with a smaller local size, which shows that the local information is indeed important and useful for the self-attention mechanism.

4.4.4.2 Different Variants of MetaFormer

In this section, we test the performance of the proposed LSA under different architecture variants of MetaFormer, where different token mixers and normalization layers are evaluated. Following PoolFormer [101] and EfficientFormer [21], we use the pooling operation as the token mixer in the first two stages as well. As we can see in Table 4.8, similar results can be obtained compared to depth-wise convolutions, which shows that our model does not only rely on these convolution layers. In addition, depth-wise convolution is as efficient as pooling, which suggests that depth-wise convolution is a more natural choice for these hybrid models. With respect to normalization layers, contrary to previous work [21, 106] showing that Batch Normalization (BN) achieves similar results to Layer Normalization (LN), we observe a clear performance drop (around 0.5%) by using BN since our method is a more transformer-like approach. This performance drop happens on downstream tasks, such as object detection, as well. However, BN is much faster on mobile devices, given that it can be fused before inference. As a result, we adopt a hybrid mode where we use BN for convolution layers and LN for LSA layers, which provides the best trade-off. We also test our model with one more LSA stage, but no clear improvement is observed.

Length	196	98	49	28	14	7
196	81.7	79.5	75.8	70.1	57.3	28.6
98	-	81.5	81.1	80.1	78.0	68.1
49	-	-	81.4	81.2	80.7	78.0

Table 4.7: Accuracy of the networks trained with specific local size (196, 98, 49 respectively). We test the pre-trained model with various local sizes, and no finetuning is used during the evaluation.

Stages	Token Mixers	Norm	Accuracy	Latency
2	Pooling	LN	81.3	2.8 ms
2	DWConv	LN	81.4	2.7 ms
2	DWConv	BN	80.7	1.9 ms
2	DWConv	BN-LN	81.1	2.0 ms
3	DWConv	LN	81.2	3.2 ms

Table 4.8: Ablation study for different variants of MetaFormer, including different token mixers, normalization layers, and stage combinations.

4.4.4.3 Evaluating Position Bias and Dimension Swap

In this section, we show the effectiveness of the dimension swap and the relative position bias. Three models are compared in Table 4.9, including one with both methods, one with only dimension swap, and one with neither of them. Performance on downstream tasks is evaluated as well. Both methods improve the image classification accuracy on ImageNet by around 0.4%. Moreover, dimension swap dramatically boosts the performance of object detection and semantic segmentation, where high-resolution inputs are required. Swap dimension improves the mAP for object detection by 3% and mIoU for semantic segmentation by 5%. Since the local size is relatively small, 49 against 1024 in segmentation with input resolution 512×512, dimension swapping can increase the receptive field and thus improve the performance.

	ImageNet	MSCOCO		ADE20K
	Top-1	AP^b	AP^m	mIoU
	80.6	39.3	36.6	38.2
+ swap	81.0	42.2	38.9	43.5
+ pos + swap	81.4	42.9	39.2	43.7

Table 4.9: Benefits of dimension swapping and relative position bias. Dimension swapping can significantly improve performance especially on downstream tasks. Relative position bias can further boost performance.

4.5 Conclusion

Previous work has mostly dismissed local self-attention for transformers on mobile devices, largely because of problems caused by memory layout and tensor reshaping operations on mobile tools such as ANE. In this work, we revisit local self-attention for mobile vision transformers. We find that LSA can actually be modified to work quite well, and can succeed without tensor layout modification, using only our proposed mobile-friendly partitioning and dimension swapping. We validate this through extensive experiments. Overall, we find the benefits of our modified LSA to be especially apparent for high-resolution applications, such as object detection and semantic segmentation.

4.6 Appendix

4.6.1 Pytorch-Like Pseudocode

The PyTorch-Like pseudo code is displayed in Alg 2. Given an input $\mathbf{X}$ with dimension B, C, H, W, dimension swapping and local partition can be done with one reshape operation which is exactly the same as reshaping a 4D tensor to 3D.

4.6.2 Model Variants

Table 4.10 summarizes the configurations of the models used in the experiments. We mainly follow the structure of MetaFormer [101]. We use depthwise convolution as the token mixer for the shallow stages with higher input resolution. And we use our proposed LSA with a local length of 49 as the token mixer for the deep stages with lower input resolution. In addition,

Stage	Output Size	Layer	Configs	MetaFormer			
				S12	S18	S24	Swin-S
Stem	$\frac{H}{4} \times \frac{W}{4}$	Convolution	Kernal Size	3×3, stride 2			
			Embed Dim	64	64	64	96
		Convolution	Kernal Size	3×3, stride 2			
			Embed Dim	64	64	64	96
1	$\frac{H}{4} \times \frac{W}{4}$	MetaBlock	Token Mixer	DW Conv, 3×3, stride 1			
			Embed Dim	64	64	64	96
			MLP Ratio	4			
			# Blocks	2	3	4	2
Down Sample	$\frac{H}{8} \times \frac{W}{8}$	Convolution	Kernal Size	3×3, stride 2			
			Embed Dim	128	128	128	192
2	$\frac{H}{8} \times \frac{W}{8}$	MetaBlock	Token Mixer	DW Conv, 3×3, stride 1			
			Embed Dim	128	128	128	192
			MLP Ratio	4			
			# Blocks	2	4	4	2
Down Sample	$\frac{H}{16} \times \frac{W}{16}$	Convolution	Kernal Size	3×3, stride 2			
			Embed Dim	320	320	320	384
2	$\frac{H}{16} \times \frac{W}{16}$	MetaBlock	Token Mixer	LSA, length 49			
			Embed Dim	320	320	320	384
			MLP Ratio	4			
			# Blocks	6	8	12	18
Down Sample	$\frac{H}{32} \times \frac{W}{32}$	Convolution	Kernal Size	3×3, stride 2			
			Embed Dim	512	512	512	768
2	$\frac{H}{16} \times \frac{W}{16}$	MetaBlock	Token Mixer	LSA, length 49			
			Embed Dim	512	512	512	768
			MLP Ratio	4			
			# Blocks	2	3	4	2

Table 4.10: Configuration details of different model variants we used in the experiments.

we also provide a model variant that has the same number of channels and blocks for each stage as Swin-S [32].

4.6.3 Experiment Details

For model training on ImageNet, we follow previous work [22, 32, 101], as described in Section 4.4.1. For object detection on MSCOCO, we finetune the ImageNet-1K pre-trained

model following the training settings of previous work [32, 106]. We mainly focus on the "$1\times$" setting, where we finetune the model with 12 epochs in total. We use AdamW as the optimizer with a peak learning rate of 0.0001, and the learning rate decreases by $10\times$ at epoch of 8 and 11. We use 1000 warm-up iterations and a weight decay rate of 0.05. For semantic segmentation on ADE20k, we follow the training setting of Semantic FPN from Next-ViT [106], where we finetune the ImageNet-1K pre-trained model for 40000 iterations with the AdamW optimizer on 8 GPUs. We set the learning rate and weight decay as 0.0001. For the experiments regarding Next-ViT variants, we use the same hyper-parameters as Next-ViT, except one setting for Next-ViT large on semantic segmentation, where we use a larger weight decay (0.01) and stochastic path drop rate (0.3).

For the latency evaluation on CoreML, we evaluate both versions of `coremltools` (5.2 and 6.0), but we do not observe a big difference. Notably, when measuring the latency with extreme-high-resolution inputs, such as 1024×1024, Apple Neural Engine (ANE) is not always working for arbitrary dimensions. For example, we observe that several layers of Next-ViT (in stage 2) can only be runnable on CPU and GPU with the default model configurations. As a result, for fair comparison, we tweak their mixed block ratio to have a dimension of the embeddings that can be runnable on ANE for self-attention modules. This happens for our proposed LSA as well when a local length other than 49 is used, such as 64 together with the specific channel dimension 320 we used in stage 3. However, it is runnable on ANE with either a smaller dimension, 256, or a larger dimension 384, which shows that in general cases, our proposed method can still be deployed efficiently on mobile devices.

4.6.4 Comparison with Window-based Attention

Comparison between window-based attention and the proposed local self-attention on ADE20K segmentation and COCO detection. Our method outperforms window and grid attention adopted from MaxViT.

Method	MSCOCO		ADE20K
	AP^b	AP^m	mIoU
Win	39.1	36.3	38.4
Win/Grid	41.2	38.2	41.6
LSA(Ours)	**42.9**	**39.3**	**43.7**

Table 4.11: Comparison to different local attention methods on high-resolution tasks.

4.6.5 More Results for Ablation Studies

In this section, we provide more results for the ablation studies described in Section 4.4.4. Table 4.12 shows the top-1 accuracy on ImageNet-1K and latency on CoreML for proposed LSA with various local lengths. In general, with a longer local length, the network will be slower but achieve better accuracy. In addition, more results on downstream tasks for MetaFormer variants are provided in Table 4.13. We evaluate the backbones used in our experiments (See Section 4.4) with different normalization layers. For most experiments, the accuracy or the precision drops when we use Batch Normalization layers instead of Layer Normalization. For example, the accuracy of image classification on ImageNet-1K drops around 0.2%, and the box mAP of object detection on MSCOCO drops around 0.4% on average. However, the latency increases notably when LN is applied. For instance, the latency increases roughly 30%, 28%, and 20% on average, for inputs with resolution $224 \times 224, 512 \times 512, 1024 \times 1024$ respectively.

Local Length	Top-1 Accuracy (%)	Latency (ms)
Full	81.7	3.8
98	81.5	2.9
49	81.4	2.7
14	79.6	2.5
7	79.3	2.5

Table 4.12: ImageNet-1K classification accuracy and latency on CoreML for our proposed LSA with various local lengths.

Backbone	Norm Layer	ImageNet		ADE20k		MSCOCO	
		Accuracy	Latency	mIoU	Latency	mAP^{box}	Latency
MetaFormer-S12(LSA)	BN-LN	81.1	2.1	43.2	10.0	42.6	40.9
MetaFormer-S12(LSA)	LN	81.4	2.7	43.7	14.6	43.1	49.4
MetaFormer-S18(LSA)	BN-LN	82.3	2.9	45.3	13.7	44.2	55.9
MetaFormer-S18(LSA)	LN	82.3	3.8	45.0	17.1	44.3	70.6
MetaFormer-S24(LSA)	BN-LN	82.8	3.8	46.6	18.0	44.6	77.3
MetaFormer-S24(LSA)	LN	82.9	5.1	46.3	22.5	45.0	94.7
MetaFormer-SwinS(LSA)	BN-LN	83.5	5.6	46.5	27.3	45.1	129.2
MetaFormer-SwinS(LSA)	LN	83.7	7.3	46.9	31.2	45.8	136.9

Table 4.13: More ablation studies for MetaFormer variants on downstream tasks. "BN" denotes Batch Normalization, "LN" denotes Layer Normalization and "BN-LN" denotes the hybrid strategy where using Batch Normalization for convolution token mixers and using Layer Normalization for LSA token mixers.

Algorithm 2 PyTorch-like implementation of mobile-friendly LSA module.

```
# Pytorch-like Mobile Friendly Local Self-Attention

class LSA(nn.Module):
    def __init__(self, dim, head_dim, local_len):
        super().__init__()
        # Multi-Head Self-Attention for 3D tensor
        self.attn = MHSA(dim, head_dim)
        self.local_len = local_len

    def forward(self, x, swap):
        B, C, H, W = x.shape

        # Efficient token partition with dimension swap
        if swap:
            x = x.permute(0,3,2,1).reshape(-1, self.local_len,
C)
        else:
            x = x.permute(0,2,3,1).reshape(-1, self.local_len,
C)

        B_, N, C = x.shape
        # Conducting self-attention on each local partition
individually
        x = self.attn(x)

        if swap:
            x = x.view(-1, W, H, C).permute(0,3,2,1)
        else:
            x = x.view(-1, H, W, C).permute(0,3,1,2)

        return x
```

Chapter 5: Unraveling Meta-Learning: Understanding Feature Representations for Few-Shot Tasks

5.1 Introduction

Training neural networks from scratch requires large amounts of labeled data, making it impractical in many settings. When data is expensive or time-consuming to obtain, training from scratch may be cost-prohibitive [128]. In other scenarios, models must adapt efficiently to changing environments before enough time has passed to amass a large and diverse data corpus [129]. In both of these cases, massive state-of-the-art networks would overfit the tiny training sets available. To overcome this problem, practitioners pre-train on large auxiliary datasets and then fine-tune the resulting models on the target task. For example, ImageNet pre-training of large ResNets has become an industry standard for transfer learning [130]. Unfortunately, transfer learning from classically trained models often yields sub-par performance in the extremely data-scarce regime or breaks down entirely when only a few data samples are available in the target domain.

Recently, numerous few-shot benchmarks have been rapidly improved using meta-learning methods [131, 132]. Unlike classical transfer learning, which uses a base model pre-trained on a different task, meta-learning algorithms produce a base network that is specifically designed

for quick adaptation to new tasks using few-shot data. Furthermore, meta-learning is still effective when applied to small, lightweight base models that can be fine-tuned with relatively few computations.

The ability of meta-learned networks to rapidly adapt to new domains suggests that *the feature representations learned by meta-learning must be fundamentally different than feature representations learned through conventional training.* Because of the good performance that meta-learning offers in various settings, many researchers have been content to use these features without considering how or why they differ from conventional representations. As a result, little is known about the fundamental differences between meta-learned feature extractors and those which result from classical training. Training routines are often treated like a black box in which high performance is celebrated, but a deeper understanding of the phenomenon remains elusive. To further complicate matters, a myriad of meta-learning strategies exist that may exploit different mechanisms.

In this work [51], we delve into the differences between features learned by meta-learning and classical training. We explore and visualize the behaviors of different methods and identify two different mechanisms by which meta-learned representations can improve few-shot learning. In the case of meta-learning strategies that fix the feature extractor and only update the last (classification) layer of a network during the inner-loop, such as MetaOptNet [131] and R2-D2 [2], we find that meta-learning tends to cluster object classes more tightly in feature space. As a result, the classification boundaries learned during fine-tuning are less sensitive to the choice of few-shot samples. In the second case, we hypothesize that meta-learning strategies that use end-to-end fine-tuning, such as Reptile [1], search for meta-parameters that lie close in weight space to a wide range of task-specific minima. In this case, a small number of SGD steps can

transport the parameters to a good minimum for a specific task.

Inspired by these observations, we propose simple regularizers that improve feature space clustering and parameter-space proximity. These regularizers boost few-shot performance appreciably, and improving feature clustering does so without the dramatic increase in optimization cost that comes from conventional meta-learning.

5.2 Problem Setting

5.2.1 The Meta-Learning Framework

In the context of few-shot learning, the objective of meta-learning algorithms is to produce a network that quickly adapts to new classes using little data. Concretely stated, meta-learning algorithms find parameters that can be fine-tuned in few optimization steps and on few data points in order to achieve good generalization on a task $\mathcal{T}_i$, consisting of a small number of data samples from a distribution and label space that was not seen during training. The task is characterized as *n-way*, *k-shot* if the meta-learning algorithm must adapt to classify data from $\mathcal{T}_i$ after seeing k examples from each of the n classes in $\mathcal{T}_i$.

Meta-learning schemes typically rely on bi-level optimization problems with an *inner loop* and an *outer loop*. An iteration of the outer loop involves first sampling a "task," which comprises two sets of labeled data: the support data, $\mathcal{T}_i^s$, and the query data, $\mathcal{T}_i^q$. Then, in the inner loop, the model being trained is fine-tuned using the support data. Finally, the routine moves back to the outer loop, where the meta-learning algorithm minimizes loss on the query data with respect to the pre-fine-tuned weights. This minimization is executed by differentiating through the inner loop computation and updating the network parameters to make the inner loop fine-tuning

as effective as possible. Note that, in contrast to standard transfer learning (which uses classical training and simple first-order gradient information to update parameters), meta-learning algorithms differentiate through the entire fine-tuning loop. A formal description of this process can be found in Algorithm 3, as seen in Goldblum et al. [133].

Algorithm 3 The meta-learning framework

Require: Base model, F_θ, fine-tuning algorithm, A, learning rate, γ, and distribution over tasks, $p(\mathcal{T})$.
Initialize θ, the weights of F;
while not done **do**
  Sample batch of tasks, $\{\mathcal{T}_i\}_{i=1}^n$, where $\mathcal{T}_i \sim p(\mathcal{T})$ and $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$.
  for $i = 1, \ldots, n$ **do**
    Fine-tune model on $\mathcal{T}_i$ (inner loop).
    New network parameters are written $\theta_i = A(\theta, \mathcal{T}_i^s)$.
    Compute gradient $g_i = \nabla_\theta \mathcal{L}(F_{\theta_i}, \mathcal{T}_i^q)$
  Update base model parameters (outer loop):
  $\theta \leftarrow \theta - \frac{\gamma}{n} \sum_i g_i$

5.2.2 Meta-Learning Algorithms

A variety of meta-learning algorithms exist, mostly differing in how they fine-tune on support data during the inner loop. Some meta-learning approaches, such as MAML, update all network parameters using gradient descent during fine-tuning [45]. Because differentiating through the inner loop is memory and computationally intensive, the fine-tuning process consists of only a few (sometimes just 1) SGD steps.

Reptile, which functions as a zeroth-order approximation to MAML, avoids unrolling the inner loop and differentiating through the SGD steps. Instead, after fine-tuning on support data, Reptile moves the central parameter vector in the direction of the fine-tuned parameters during the outer loop [1]. In many cases, Reptile achieves better performance than MAML without

having to differentiate through the fine-tuning process.

Another class of algorithms freezes the feature extraction layers during the inner loop; only the linear classifier layer is trained during fine-tuning. Such methods include R2-D2 and MetaOptNet [2, 131]. The advantage of this approach is that the fine-tuning problem is now a convex optimization problem. Unlike MAML, which simulates the fine-tuning process using only a few gradient updates, last-layer meta-learning methods can use differentiable optimizers to exactly minimize the fine-tuning objective and then differentiate the solution with respect to feature inputs. Moreover, differentiating through these solvers is computationally cheap compared to MAML's differentiation through SGD steps on the whole network. While MetaOptNet relies on an SVM loss, R2-D2 simplifies the process even further by using a quadratic objective with a closed-form solution. R2-D2 and MetaOptNet achieve stronger performance than MAML and are able to harness larger architectures without overfitting.

Model	SVM	RR	ProtoNet	MAML
MetaOptNet-M	**62.64** $\pm$ 0.31 %	**60.50** $\pm$ 0.30 %	**51.99** $\pm$ 0.33 %	**55.77** $\pm$ 0.32 %
MetaOptNet-C	56.18 $\pm$ 0.31 %	55.09 $\pm$ 0.30 %	41.89 $\pm$ 0.32 %	46.39 $\pm$ 0.28 %
R2-D2-M	**51.80** $\pm$ 0.20 %	**55.89** $\pm$ 0.31 %	**47.89** $\pm$ 0.32 %	**53.72** $\pm$ 0.33 %
R2-D2-C	48.39 $\pm$ 0.29 %	48.29 $\pm$ 0.29 %	28.77 $\pm$ 0.24 %	44.31 $\pm$ 0.28 %

Table 5.1: Comparison of meta-learning and classical transfer learning models with various fine-tuning algorithms on 1-shot mini-ImageNet. "MetaOptNet-M" and "MetaOptNet-C" denote models with MetaOptNet backbone trained with MetaOptNet-SVM and classical training. Similarly, "R2-D2-M" and "R2-D2-C" denote models with R2-D2 backbone trained with ridge regression (RR) and classical training. Column headers denote the fine-tuning algorithm used for evaluation, and the radius of confidence intervals is one standard error.

Another last-layer method, ProtoNet, classifies examples by the proximity of their features to those of class centroids - a metric learning approach - in its inner loop [3]. Again, the feature extractor's parameters are frozen in the inner loop, and the extracted features are used to create

class centroids which then determine the network's class boundaries. Because calculating class centroids is mathematically simple, this algorithm is able to efficiently backpropagate through this calculation to adjust the feature extractor.

In this work, "classically trained" models are trained, using cross-entropy loss and SGD, on all classes simultaneously, and the feature extractors are adapted to new tasks using the same fine-tuning procedures as the meta-learned models for fair comparison. This approach represents the industry-standard method of transfer learning using pre-trained feature extractors.

5.2.3 Few-Shot Datasets

Several datasets have been developed for few-shot learning. We focus our attention on two datasets: mini-ImageNet and CIFAR-FS. Mini-ImageNet is a pruned and downsized version of the ImageNet classification dataset, consisting of 60,000, 84×84 RGB color images from 100 classes [134]. These 100 classes are split into $64, 16,$ and 20 classes for training, validation, and testing sets, respectively. The CIFAR-FS dataset samples images from CIFAR-100 [2]. CIFAR-FS is split in the same way as mini-ImageNet with 60,000 32×32 RGB color images from 100 classes divided into $64, 16,$ and 20 classes for training, validation, and testing sets, respectively.

5.2.4 Related Work

In addition to introducing new methods for few-shot learning, recent work has increased our understanding of why some models perform better than others at few-shot tasks. One such exploration performs baseline testing and discovers that network size has a large effect on the success of meta-learning algorithms [135]. Specifically, on some very large architectures, the

performance of transfer learning approaches that of some meta-learning algorithms. We thus focus on architectures common in the meta-learning literature. Methods for improving transfer learning in the few-shot classification setting focus on much larger backbone networks [135, 136].

Other work on transfer learning has found that feature extractors trained on large complex tasks can be more effectively deployed in a transfer learning setting by distilling knowledge about only important features for the transfer task [137]. Yet other work finds that features generated by a pre-trained model on data from classes absent from training are entangled, but the logits of the unseen data tend to be clustered [138]. Meta-learners without supervision in the outer loop have been found to perform well when equipped with a clustering-based penalty in the meta-objective [139]. Work on standard supervised learning has alternatively studied low-dimensional structures via rank [140, 141].

While improvements have been made to meta-learning algorithms and transfer learning approaches to few-shot learning, little work has been done on understanding the underlying mechanisms that cause meta-learning routines to perform better than classically trained models in data-scarce settings.

5.3 Are Meta-Learned Features Fundamentally Better for Few-Shot Learning?

It has been said that meta-learned models "learn to learn" [45], but one might ask if they instead learn to optimize; their features could simply be well-adapted for the specific fine-tuning optimizers on which they are trained. We dispel the latter notion in this section.

In Table 5.1, we test the performance of meta-learned feature extractors not only with their own fine-tuning algorithm, but with a variety of fine-tuning algorithms. We find that in

all cases, the meta-learned feature extractors outperform classically trained models of the same architecture. See Appendix 5.7.1 for results from additional experiments.

This performance advantage across the board suggests that meta-learned features are qualitatively different than conventional features and fundamentally superior for few-shot learning. The remainder of this work will explore the characteristics of meta-learned models.

5.4 Class Clustering in Feature Space

Methods such as ProtoNet, MetaOptNet, and R2-D2 fix their feature extractor during fine-tuning. For this reason, they must learn to embed features in a way that enables few-shot classification. For example, MetaOptNet and R2-D2 require that classes are linearly separable in feature space, but mere linear separability is not a sufficient condition for good few-shot performance. The feature representations of randomly sampled few-shot data from a given class must not vary so much as to cause classification performance to be sample-dependent. In this section, we examine clustering in feature space, and we find that meta-learned models separate features differently than classically trained networks.

5.4.1 Measuring Clustering in Feature Space

We begin by measuring how well different training methods cluster feature representations. To measure feature clustering (FC), we consider the intra-class to inter-class variance ratio

$$\frac{\sigma^2_{within}}{\sigma^2_{between}} = \frac{C}{N} \frac{\sum_{i,j} \|\phi_{i,j} - \mu_i\|_2^2}{\sum_i \|\mu_i - \mu\|_2^2},$$

where $\phi_{i,j}$ is a feature vector in class i, μ_i is the mean of feature vectors in class i, μ is the mean across all feature vectors, C is the number of classes, and N is the number of data points per class. Low values of this fraction correspond to collections of features such that classes are well-separated and a hyperplane formed by choosing a point from each of two classes does not vary dramatically with the choice of samples.

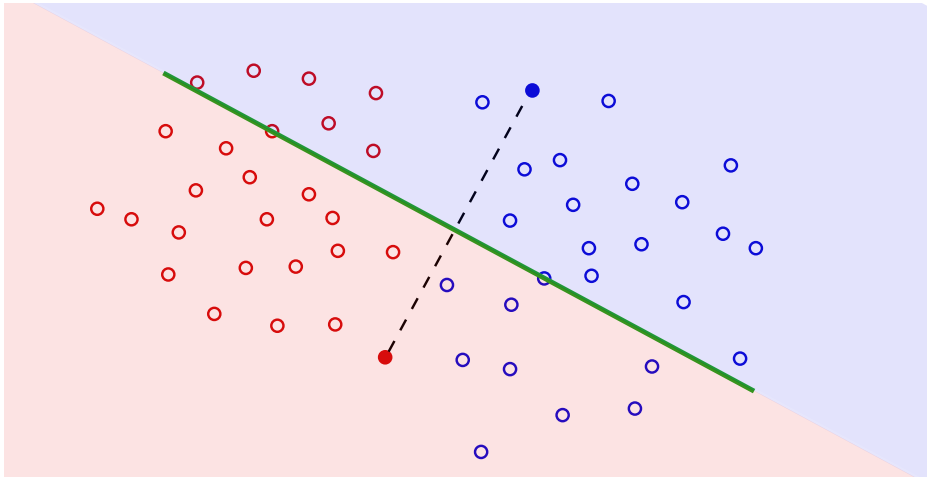
In Table 5.2, we highlight the superior class separation of meta-learning methods. We compute two quantities, R_{FC} and R_{HV}, for MetaOptNet and R2-D2 as well as classical transfer learning baselines of the same architectures. These two quantities measure the intra-class to inter-class variance ratio and invariance of separating hyperplanes to data sampling. Mathematical formulations of R_{FC} and R_{HV} can be found in Sections 5.4.4 and 5.4.5, respectively. Lower values of each measurement correspond to better class separation. On both CIFAR-FS and mini-ImageNet, the meta-learned models attain lower values, indicating that feature space clustering plays a role in the effectiveness of meta-learning.

Training	Dataset	R_{FC}	R_{HV}
R2-D2-M	CIFAR-FS	**1.29**	**0.95**
R2-D2-C	CIFAR-FS	2.92	1.69
MetaOptNet-M	CIFAR-FS	**0.99**	**0.75**
MetaOptNet-C	CIFAR-FS	1.84	1.25
R2-D2-M	mini-ImageNet	**2.60**	**1.57**
R2-D2-C	mini-ImageNet	3.58	1.90
MetaOptNet-M	mini-ImageNet	**1.29**	**0.95**
MetaOptNet-C	mini-ImageNet	3.13	1.75

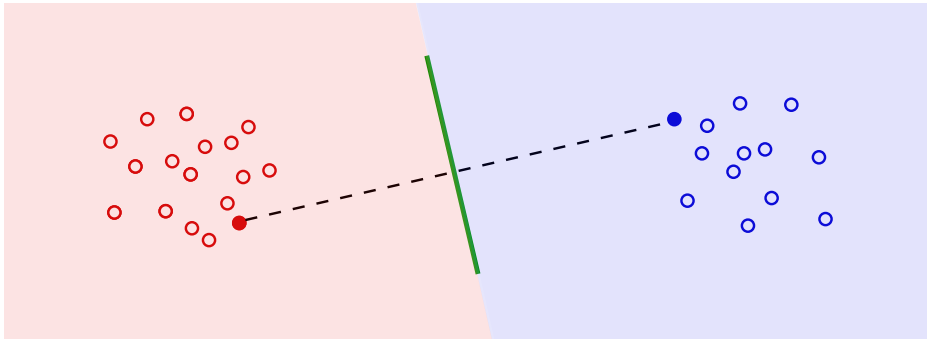
Table 5.2: Comparison of class separation metrics for feature extractors trained by classical and meta-learning routines. R_{FC} and R_{HV} are measurements of feature clustering and hyperplane variation, respectively, and we formalize these measurements below. In both cases, lower values correspond to better class separation. We pair together models according to dataset and backbone architecture. "-C" and "-M" respectively denote classical training and meta-learning. See Sections 5.4.4 and 5.4.5 for more details.

5.4.2 Why is Clustering Important?

To demonstrate why linear separability is insufficient for few-shot learning, consider Figure 5.1. As features in a class become spread out and the classes are brought closer together, the classification boundaries formed by sampling one-shot data often misclassify large regions. In contrast, as features in a class are compacted and classes move far apart from each other, the intra-class to inter-class variance ratio drops, and the dependence of the class boundary on the choice of one-shot samples becomes weaker.



(a)



(b)

Figure 5.1: a) When class variation is high relative to the variation between classes, decision boundaries formed by one-shot learning are inaccurate, even though classes are linearly separable. b) As classes move farther apart relative to the class variation, one-shot learning yields better decision boundaries.

This intuitive argument is formalized in the following result.

Theorem 1. *Consider two random variables, X representing class 1, and Y representing class 2. Let U be the random variable equal to X with probability $1/2$, and Y with probability $1/2$. Assume the variance ratio bound*

$$\frac{Var[X] + Var[Y]}{Var[U]} < \epsilon$$

holds for sufficiently small $\epsilon \geq 0$.

Draw random one-shot data, $x \sim X$ and $y \sim Y$, and a test point $z \sim X$. Consider the linear classifier

$$c(z) = \begin{cases} 1, & \text{if } z^T(x-y) - \frac{1}{2}\|x\|^2 + \frac{1}{2}\|y\|^2 \geq 0 \\ 2, & \text{otherwise.} \end{cases}$$

This classifier assigns the correct label to z with probability at least

$$1 - \frac{32\epsilon}{1-\epsilon}.$$

Note that the linear classifier in the theorem is simply the maximum-margin linear classifier that separates the two training points. In plain words, Theorem 1 guarantees that one-shot learning performance is effective when the variance ratio is small, with classification becoming asymptotically perfect as the ratio approaches zero. A proof is provided in Appendix 5.7.5.

5.4.3 Comparing Feature Representations of Meta-Learning and Classically Trained Models

We begin our investigation into the feature space of meta-learned models by visualizing features. Figure 5.2 contains a visual comparison of ProtoNet and a classically trained model of the same architecture on mini-ImageNet. Three classes are randomly chosen from the test set, and 100 samples are taken from each class. The samples are then passed through the feature extractor, and the resulting vectors are plotted. Because feature space is high-dimensional, we perform a linear projection into $\mathbb{R}^2$. We project onto the first two component vectors determined by LDA. Linear discriminant analysis (LDA) projects data onto directions that minimize the intra-class to inter-class variance ratio [142], and LDA is therefore ideal for visualizing the class separation phenomenon.

In the plots, we see that relative to the size of the point clusters, the classically trained model mashes features together, while the meta-learned models draw the classes farther apart. While visually separate class features may be neither a necessary nor sufficient condition for few-shot performance, we take these plots as inspiration for our regularizer in the following section.

5.4.4 Feature Space Clustering Improves the Few-Shot Performance of Transfer Learning

We now further test the feature clustering hypothesis by promoting the same behavior in classically trained models. Consider a network with feature extractor f_θ and fully-connected layer g_w. Then, denoting training data in class i by $\{x_{i,j}\}$, we formulate the feature clustering

regularizer by

$$R_{FC}(\theta, \{x_{i,j}\}) = \frac{C}{N} \frac{\sum_{i,j} \|f_\theta(x_{i,j}) - \mu_i\|_2^2}{\sum_i \|\mu_i - \mu\|_2^2},$$

where $f_\theta(x_{i,j})$ is a feature vector corresponding to a data point in class i, μ_i is the mean of feature vectors in class i, and μ is the mean across all feature vectors. When this regularizer has value zero, classes are represented by distinct point masses in feature space, and thus the class boundary is invariant to the choice of few-shot data.

We incorporate this regularizer into a standard training routine by sampling two images per class in each mini-batch so that we can compute a within-class variance estimate. Then, the total loss function becomes the sum of cross-entropy and R_{FC}. We train the R2-D2 and MetaOptNet backbones in this fashion on the mini-ImageNet and CIFAR-FS datasets, and we test these networks on both 1-shot and 5-shot tasks. In all experiments, feature clustering improves the performance of transfer learning and sometimes even achieves higher performance than meta-learning. Furthermore, the regularizer does not appreciably slow down classical training, which, without the expense of differentiating through an inner loop, runs as much as 13 times faster than the corresponding meta-learning routine. See Table 5.3 for numerical results, and see Appendix 5.7.2 for experimental details including training times.

In addition to performance evaluations, we calculate the similarity between feature representations yielded by a feature extractor produced by meta-learning and that of one produced by the classical routine with and without R_{FC}. To this end, we use centered kernel alignment (CKA) [143]. Using both R2-D2 and MetaOptNet backbones on both mini-ImageNet and CIFAR-FS datasets, networks trained with R_{FC} exhibit higher similarity scores to meta-learned networks than networks trained classically but without R_{FC}. These measurements provide further

Training	Backbone	mini-ImageNet		CIFAR-FS	
		1-shot	5-shot	1-shot	5-shot
R2-D2	R2-D2	$\mathbf{51.80} \pm 0.20\%$	$68.40 \pm 0.20\%$	$65.3 \pm 0.2\%$	$79.4 \pm 0.1\%$
Classical	R2-D2	$48.39 \pm 0.29\%$	$68.24 \pm 0.26\%$	$62.9 \pm 0.3\%$	$82.8 \pm 0.3\%$
Classical w/ R_{FC}	R2-D2	$50.39 \pm 0.30\%$	$\mathbf{69.58} \pm 0.26\%$	$\mathbf{65.5} \pm 0.4\%$	$\mathbf{83.3} \pm 0.3\%$
Classical w/ R_{HV}	R2-D2	$50.16 \pm 0.30\%$	$69.54 \pm 0.26\%$	$64.6 \pm 0.3\%$	$83.1 \pm 0.3\%$
MetaOptNet-SVM	MetaOptNet	$\mathbf{62.64} \pm 0.31\%$	$\mathbf{78.63} \pm 0.25\%$	$72.0 \pm 0.4\%$	$84.2 \pm 0.3\%$
Classical	MetaOptNet	$56.18 \pm 0.31\%$	$76.72 \pm 0.24\%$	$69.5 \pm 0.3\%$	$85.7 \pm 0.2\%$
Classical w/ R_{FC}	MetaOptNet	$59.38 \pm 0.31\%$	$78.15 \pm 0.24\%$	$\mathbf{72.3} \pm 0.4\%$	$\mathbf{86.3} \pm 0.2\%$
Classical w/ R_{HV}	MetaOptNet	$59.37 \pm 0.32\%$	$77.05 \pm 0.25\%$	$72.0 \pm 0.4\%$	$85.9 \pm 0.2\%$

Table 5.3: Comparison of methods on 1-shot and 5-shot CIFAR-FS and mini-ImageNet 5-way classification. The top accuracy for each backbone/task is in bold. Confidence intervals have radius equal to one standard error. Few-shot fine-tuning is performed with SVM except for R2-D2, for which we report numbers from the original paper.

evidence that feature clustering makes feature representations closer to those trained by meta-learning and thus, that meta-learners perform feature clustering. See Table 5.4 for more details.

Backbone	Dataset	C	R_{FC}	R_{HV}
R2-D2	CIFAR-FS	0.71	**0.77**	0.73
MetaOptNet	CIFAR-FS	0.77	**0.89**	0.87
R2-D2	mini-ImageNet	0.69	**0.72**	0.70
MetaOptNet	mini-ImageNet	0.70	**0.82**	0.79

Table 5.4: Similarity (CKA) representations trained via meta-learning and via transfer learning with/without the two proposed regularizers for various backbones and both CIFAR-FS and mini-ImageNet datasets. "C" denotes the classical transfer learning without regularizers. The highest score for each dataset/backbone combination is in bold.

5.4.5 Connecting Feature Clustering with Hyperplane Invariance

For further validation of the connection between feature clustering and invariance of separating hyperplanes to data sampling, we replace the feature clustering regularizer with one that penalizes variations in the maximum-margin hyperplane separating feature vectors in opposite classes. Consider data points x_1, x_2 in class A, data points y_1, y_2 in class B, and feature extractor

f_θ. The difference vector $f_\theta(x_1) - f_\theta(y_1)$ determines the direction of the maximum margin hyperplane separating the two points in feature space. To penalize the variation in hyperplanes, we introduce the hyperplane variation regularizer,

$$R_{HV}(f_\theta(x_1), f_\theta(x_2), f_\theta(y_1), f_\theta(y_2)) \quad = \quad \frac{\|(f_\theta(x_1) - f_\theta(y_1)) - (f_\theta(x_2) - f_\theta(y_2))\|_2}{\|(f_\theta(x_1) - f_\theta(y_1)\|_2 + \|f_\theta(x_2) - f_\theta(y_2)\|_2}.$$

This function measures the distance between distance vectors $x_1 - y_1$ and $x_2 - y_2$ relative to their size. In practice, during a batch of training, we sample many pairs of classes and two samples from each class. Then, we compute R_{HV} on all class pairs and add these terms to the cross-entropy loss. We find that this regularizer performs almost as well as R_{FC} and conclusively outperforms non-regularized classical training. We include these results in Table 5.3. See Appendix 5.7.2 for more details on these experiments, including training times (which, as indicated in Section 5.4.4, are significantly lower than those needed for meta-learning).

5.4.6 MAML Does Not Have the Same Feature Separation Properties

Remember that the previous measurements and experiments examined meta-learning methods that fix the feature extractor during the inner loop. MAML is a popular example of a method that does not fix the feature extractor in the inner loop. We now quantify MAML's class separation compared to transfer learning by computing our regularizer values for a pre-trained MAML model as well as a classically trained model of the same architecture. We find that, in fact, MAML exhibits even worse feature separation than a classically trained model of the same architecture. See Table 5.5 for numerical results. These results confirm our suspicion that the feature clustering phenomenon is specific to meta-learners which fix the feature extractor during the inner loop

of training.

Model	R_{FC}	R_{HV}
MAML-1	3.9406	1.9434
MAML-5	3.7044	1.8901
MAML-C	**3.3487**	**1.8113**

Table 5.5: Comparison of regularizer values for 1-shot and 5-shot MAML models (MAML-1 and MAML-5) as well as MAML-C, a classically trained model of the same architecture on mini-ImageNet training data. The lowest value of each regularizer is in bold.

5.5 Finding Clusters of Local Minima for Task Losses in Parameter Space

Since Reptile does not fix the feature extractor during fine-tuning, it must find parameters that adapt easily to new tasks. One way Reptile might achieve this is by finding parameters that can reach a task-specific minimum by traversing a smooth, nearly linear region of the loss landscape. In this case, even a single SGD update would move parameters in a useful direction. Unlike MAML, however, Reptile does not backpropagate through optimization steps and thus lacks information about the loss surface geometry when performing parameter updates. Instead, we hypothesize that Reptile finds parameters that lie very close to good minima for many tasks and is therefore able to perform well on these tasks after very little fine-tuning.

This hypothesis is further motivated by the close relationship between Reptile and *consensus optimization* [144]. In a consensus method, a number of models are independently optimized with their own task-specific parameters, and the tasks communicate via a penalty that encourages all the individual solutions to converge around a common value. Reptile can be interpreted as

approximately minimizing the consensus formulation

$$\frac{1}{m}\sum_{p=1}^{m}\mathcal{L}_{\mathcal{T}_p}(\tilde{\theta}_p)+\frac{\gamma}{2}\|\tilde{\theta}_p-\theta\|^2,$$

where $\mathcal{L}_{\mathcal{T}_p}(\tilde{\theta}_p)$ is the loss for task $\mathcal{T}_p$, $\{\tilde{\theta}_p\}$ are task-specific parameters, and the quadratic penalty on the right encourages the parameters to cluster around a "consensus value" θ. A stochastic optimizer for this loss would proceed by alternately selecting a random task/term index p, minimizing the loss with respect to $\tilde{\theta}_p$, and then taking a gradient step $\theta \leftarrow \theta - \eta\tilde{\theta}_p$ to minimize the loss for θ.

Reptile diverges from a traditional consensus optimizer only in that it does not explicitly consider the quadratic penalty term when minimizing for $\tilde{\theta}_p$. However, it implicitly considers this penalty by initializing the optimizer for the task-specific loss using the current value of the consensus variables θ, which encourages the task-specific parameters to stay near the consensus parameters. In the next section, we replace the standard Reptile algorithm with one that explicitly minimizes a consensus formulation.

5.5.1 Consensus Optimization Improves Reptile

To validate the weight-space clustering hypothesis, we modify Reptile to explicitly enforce parameter clustering around a consensus value. We find that directly optimizing the consensus formulation leads to improved performance. To this end, during each inner loop update step in Reptile, we penalize the squared ℓ_2 distance from the parameters for the current task to the

average of the parameters across all tasks in the current batch. Namely, we let:

$$R_i\big(\{\tilde{\theta}_p\}_{p=1}^m\big) = d\big(\tilde{\theta}_i, \frac{1}{m}\sum_{p=1}^m \tilde{\theta}_p\big)^2,$$

where $\tilde{\theta}_p$ are the network parameters on task p and d is the filter normalized ℓ_2 distance (see Note 1). Note that as parameters shrink towards the origin, the distances between minima shrink as well. Thus, we employ filter normalization to ensure that our calculation is invariant to scaling [145]. See below for a description of filter normalization. This regularizer guides optimization to a location where many task-specific minima lie in close proximity. A detailed description is given in Algorithm 4, which is equivalent to the original Reptile when $\alpha = 0$. We call this method "Weight-Clustering."

Note 1. *Consider that a perturbation to the parameters of a network is more impactful when the network has small parameters. While previous work has used layer normalization or even more coarse normalization schemes, the authors of Li et al. [145] note that since the output of networks with batch normalization is invariant to filter scaling as long as the batch statistics are updated accordingly, we can normalize every filter of such a network independently. The latter work suggests that this scheme, "filter normalization", correlates better with properties of the optimization landscape. Thus, we measure distance in our regularizer using filter normalization, and we find that this technique prevents parameters from shrinking towards the origin.*

We compare the performance of our regularized Reptile algorithm to that of the original Reptile method as well as first-order MAML (FOMAML) and a classically trained model of the same architecture. We test these methods on a sample of 100,000 5-way 1-shot and 5-shot mini-ImageNet tasks and find that in both cases, Reptile with Weight-Clustering achieves higher

Algorithm 4 Reptile with Weight-Clustering Regularization

Require: Initial parameter vector, θ, outer learning rate, γ, inner learning rate, η, regularization coefficient, α, and distribution over tasks, $p(\mathcal{T})$.
for *meta-step* $= 1, \ldots, n$ **do**
  Sample batch of tasks, $\{\mathcal{T}_i\}_{i=1}^{m}$ from $p(\mathcal{T})$
  Initialize parameter vectors $\tilde{\theta}_i^0 = \theta$ for each task
  for $j = 1, \ldots, k$ **do**
    for $i = 1, \ldots, m$ **do**
      Calculate $\mathcal{L} = \mathcal{L}_{\mathcal{T}_i}^j + \alpha R_i\big(\{\tilde{\theta}_p^{j-1}\}_{p=1}^{m}\big)$
      Update $\tilde{\theta}_i^j = \tilde{\theta}_i^{j-1} - \eta \nabla_{\tilde{\theta}_i} \mathcal{L}$
  Compute difference vectors $\{g_i = \tilde{\theta}_i^k - \tilde{\theta}_i^0\}_{i=1}^{m}$
  Update $\theta \leftarrow \theta - \frac{\gamma}{m} \sum_i g_i$

performance than the original algorithm and significantly better performance than FOMAML and the classically trained models. These results are summarized in Table 5.6.

Framework	1-shot	5-shot
Classical	$28.72 \pm 0.16\%$	$45.25 \pm 0.21\%$
FOMAML	$48.07 \pm 1.75\%$	$63.15 \pm 0.91\%$
Reptile	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$
W-Clustering	$\mathbf{51.94} \pm 0.23\%$	$\mathbf{68.02} \pm 0.22\%$

Table 5.6: Comparison of methods on 1-shot and 5-shot mini-ImageNet 5-way classification. The top accuracy for each task is in bold. Confidence intervals have width equal to one standard error. W-Clustering denotes the Weight-Clustering regularizer.

We note that the best-performing result was attained when the product of the constant term collected from the gradient of the regularizer R_i and the regularization coefficient α was 5.0×10^{-5}, but a range of values up to ten times larger and smaller also produced improvements over the original algorithm. Experimental details, as well as results for other values of this coefficient, can be found in Appendix 5.7.3.

In addition to these performance gains, we found that the parameters of networks trained using our regularized version of Reptile do not travel as far during fine-tuning at inference as those trained using vanilla Reptile. Figure 5.3 depicts histograms of filter normalized distance traveled

by both networks fine-tuning on samples of 1,000 1-shot and 5-shot mini-ImageNet tasks. From these, we conclude that our regularizer does indeed move model parameters toward a consensus which is near good minima for many tasks. Interestingly, we applied these same measurements to networks trained using MetaOptNet and R2-D2, and we found that these feature extractors lie in wide and flat minimizers across many task losses. Thus, when the whole network is fine-tuned, the parameters move a lot without substantially decreasing loss. Previous work has associated flat minimizers with good generalization [146].

5.6 Discussion

In this work, we shed light on two key differences between meta-learned networks and their classically trained counterparts. We find evidence that meta-learning algorithms minimize the variation between feature vectors within a class relative to the variation between classes. Moreover, we design two regularizers for transfer learning inspired by this principle, and our regularizers consistently improve few-shot performance. The success of our method helps to confirm the hypothesis that minimizing within-class feature variation is critical for few-shot performance.

We further notice that Reptile resembles a consensus optimization algorithm, and we enhance the method by designing yet another regularizer, which we apply to Reptile, in order to find clusters of local minima in the loss landscapes of tasks. We find in our experiments that this regularizer improves both one-shot and five-shot performance of Reptile on mini-ImageNet.

5.7 Appendix

5.7.1 Mixing Meta-Learned Models and Fine-Tuning Procedures: Additional Experiments

Model	SVM	RR	ProtoNet	MAML
MetaOptNet-M	**78.63** $\pm$ 0.25 %	**76.96** $\pm$ 0.23 %	**76.17** $\pm$ 0.23 %	70.14 $\pm$ 0.27 %
MetaOptNet-C	76.72 $\pm$ 0.24 %	74.48 $\pm$ 0.24 %	73.37 $\pm$ 0.24 %	**71.32** $\pm$ 0.26 %
R2-D2-M	**68.40** $\pm$ 0.20 %	**72.09** $\pm$ 0.25 %	**70.74** $\pm$ 0.25 %	**71.43** $\pm$ 0.27 %
R2-D2-C	68.24 $\pm$ 0.26 %	67.04 $\pm$ 0.26 %	60.93 $\pm$ 0.29 %	65.30 $\pm$ 0.27 %

Table 5.7: Comparison of meta-learning and transfer learning models with various fine-tuning algorithms on 5-shot mini-ImageNet. "MetaOptNet-M" and "MetaOptNet-C" denote models with MetaOptNet backbone trained with MetaOptNet-SVM and classical training. Similarly, "R2-D2-M" and "R2-D2-C" denote models with R2-D2 backbone trained with ridge regression (RR) and classical training. Column headers denote the fine-tuning algorithm used for evaluation, and the radius of confidence intervals is one standard error.

5.7.2 Transfer Learning and Feature Space Clustering

We evaluate the proposed regularizers and classically trained baseline on two backbone architectures: a 4-layer convolutional neural network with number of filters per layer 96-192-384-512 originally used for R2-D2 [2] and ResNet-12 [10, 131, 147]. We run experiments on the mini-ImageNet and CIFAR-FS datasets.

When training the backbone feature extractors, we use SGD with a batch size of 128 for CIFAR-FS and 256 for mini-ImageNet, Nesterov momentum set to 0.9 and weight decay of 10^{-4}. For training on CIFAR-FS, we set the initial learning rate to 0.1 for the first 100 epochs and reduce by a factor of 10 every 50 epochs. To avoid gradient explosion problems, we use 15 warm-up epochs for mini-ImageNet with a learning rate of 0.01. We train all classically trained networks

for a total of 300 epochs. We employ data parallelism across 2 Nvidia RTX 2080 Ti GPUs when training on mini-ImageNet, and we only use one GPU for each CIFAR-FS experiment. For few-shot testing, we train two classification heads, a linear NN layer and SVM [131] on top of the pre-trained feature extractors. The evaluation results of these models are given in Table 5.9. Table 5.8 shows the running time per training epoch as well as total training time on both datasets and backbone architectures to achieve the results in Table 5.3. The training speed of the proposed regularizers is nearly as fast as classical transfer learning and up to almost 13 times faster than meta-learning methods. For meta-learning methods, we follow the training hyperparemeters from Lee et al. [131].

		mini-ImageNet	CIFAR-FS
Training	Backbone	runtime	runtime
R2-D2	R2-D2	16m/16.8h	44s/45m
Classical	R2-D2	20s/1.7h	4s/22m
Classical w/ R_{FC}	R2-D2	20s/1.7h	4s/24m
Classical w/ R_{HV}	R2-D2	20s/1.7h	4s/23m
MetaOptNet-SVM	MetaOptNet	1.5h/88.0h	4m/4.5h
Classical	MetaOptNet	1.4m/7.0h	14s/1.2h
Classical w/ R_{FC}	MetaOptNet	1.5m/7.4h	15s/1.3h
Classical w/ R_{HV}	MetaOptNet	1.3m/7.2h	16s/1.4h

Table 5.8: Runtime (training time per epoch/total times) comparison of methods on CIFAR-FS and mini-ImageNet 5-way classification on a single GPU.

				mini-ImageNet		CIFAR-FS	
Backbone	Regularizer	Coeff	Head	1-shot	5-shot	1-shot	5-shot
R2-D2	R_{FC}	0.02	NN	$48.27 \pm 0.29\%$	$69.13 \pm 0.26\%$	$63.11 \pm 0.35\%$	$83.31 \pm 0.25\%$
		0.05	NN	$48.75 \pm 0.29\%$	$69.50 \pm 0.26\%$	$64.49 \pm 0.35\%$	$83.32 \pm 0.25\%$
		0.1	NN	$48.72 \pm 0.29\%$	$67.39 \pm 0.25\%$	$62.98 \pm 0.36\%$	$81.07 \pm 0.26\%$
	R_{HV}	0.02	NN	$46.74 \pm 0.28\%$	$68.19 \pm 0.27\%$	$62.50 \pm 0.34\%$	$82.90 \pm 0.25\%$
		0.05	NN	$49.11 \pm 0.29\%$	$68.88 \pm 0.26\%$	$63.61 \pm 0.35\%$	83.21 ± 0.25 %
		0.1	NN	$48.87 \pm 0.29\%$	$69.67 \pm 0.26\%$	$63.50 \pm 0.35\%$	$83.17 \pm 0.25\%$
	R_{FC}	0.02	SVM	$49.05 \pm 0.30\%$	$68.94 \pm 0.26\%$	$64.48 \pm 0.34\%$	$83.11 \pm 0.25\%$
		0.05	SVM	$50.39 \pm 0.30\%$	$69.58 \pm 0.26\%$	$65.53 \pm 0.35\%$	$83.30 \pm 0.25\%$
		0.1	SVM	$50.71 \pm 0.30\%$	$68.46 \pm 0.25\%$	$64.25 \pm 0.36\%$	$81.57 \pm 0.26\%$
	R_{HV}	0.02	SVM	$47.81 \pm 0.29\%$	$68.08 \pm 0.27\%$	$63.71 \pm 0.33\%$	$82.77 \pm 0.26\%$
		0.05	SVM	$49.28 \pm 0.30\%$	$68.62 \pm 0.26\%$	$64.52 \pm 0.34\%$	$82.99 \pm 0.26\%$
		0.1	SVM	$50.16 \pm 0.30\%$	$69.54 \pm 0.26\%$	$64.62 \pm 0.34\%$	$83.08 \pm 0.26\%$
ResNet-12	R_{FC}	0.02	NN	$57.54 \pm 0.32\%$	$77.31 \pm 0.25\%$	$71.69 \pm 0.36\%$	$86.13 \pm 0.23\%$
		0.05	NN	$56.59 \pm 0.33\%$	$74.81 \pm 0.25\%$	$71.78 \pm 0.37\%$	$85.30 \pm 0.24\%$
		0.1	NN	$52.26 \pm 0.35\%$	$69.93 \pm 0.28\%$	$71.85 \pm 0.39\%$	$83.74 \pm 0.25\%$
	R_{HV}	0.02	NN	$53.75 \pm 0.30\%$	$76.11 \pm 0.25\%$	$70.12 \pm 0.35\%$	$86.37 \pm 0.23\%$
		0.05	NN	$57.15 \pm 0.31\%$	$77.27 \pm 0.25\%$	$71.49 \pm 0.36\%$	$85.85 \pm 0.24\%$
		0.1	NN	$57.76 \pm 0.33\%$	$76.05 \pm 0.26\%$	$71.56 \pm 0.37\%$	$84.80 \pm 0.25\%$
	R_{FC}	0.02	SVM	$59.38 \pm 0.31\%$	$78.15 \pm 0.24\%$	$72.32 \pm 0.30\%$	$86.31 \pm 0.24\%$
		0.05	SVM	$59.05 \pm 0.32\%$	$76.36 \pm 0.24\%$	$71.94 \pm 0.36\%$	$85.28 \pm 0.24\%$
		0.1	SVM	$56.73 \pm 0.35\%$	$73.70 \pm 0.26\%$	$71.08 \pm 0.36\%$	$83.49 \pm 0.25\%$
	R_{HV}	0.02	SVM	$56.95 \pm 0.30\%$	$77.06 \pm 0.24\%$	$71.34 \pm 0.35\%$	$86.54 \pm 0.23\%$
		0.05	SVM	$59.36 \pm 0.31\%$	$77.97 \pm 0.24\%$	$72.00 \pm 0.36\%$	$85.87 \pm 0.24\%$
		0.1	SVM	$59.37 \pm 0.32\%$	$77.05 \pm 0.25\%$	$71.92 \pm 0.37\%$	$84.84 \pm 0.25\%$

Table 5.9: Hyper-parameter tuning for R_{FC} and R_{HV} regularizers with various backbone structures and classification heads on 1-shot and 5-shot CIFAR-FS and mini-ImageNet 5-way classification. Regularizer coefficients include the C/N factor.

5.7.3 Reptile Weight Clustering

We train models via our weight-clustering Reptile algorithm with a range of coefficients for the regularization term. The model architecture and all other hyperparameters were chosen to match those specified for Reptile training and evaluation on 1-shot and 5-shot mini-ImageNet in Nichol and Schulman [1]. The evaluation results of these models are given in Table 5.10. All models were trained on Nvidia RTX 2080 Ti GPUs.

Coefficient	1-shot	5-shot
0 (Reptile)	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$
1.0×10^{-5}	$51.42 \pm 0.23\%$	$67.16 \pm 0.22\%$
2.5×10^{-5}	$51.25 \pm 0.24\%$	$67.55 \pm 0.22\%$
5.0×10^{-5}	$\mathbf{51.94} \pm 0.23\%$	$\mathbf{68.02} \pm 0.22\%$
7.5×10^{-5}	$51.40 \pm 0.24\%$	$67.59 \pm 0.22\%$
1.0×10^{-4}	$50.92 \pm 0.23\%$	$67.91 \pm 0.22\%$
2.5×10^{-4}	$50.65 \pm 0.23\%$	$65.95 \pm 0.23\%$
5.0×10^{-4}	$51.37 \pm 0.23\%$	$66.98 \pm 0.23\%$

Table 5.10: Comparison of test accuracy for models trained with the weight-clustering Reptile algorithm with various regularization coefficients evaluated on 1-shot and 5-shot mini-ImageNet tasks. The results for vanilla Reptile are those given in Nichol and Schulman [1].

5.7.4 Architectures

For our experiments using MAML, R2-D2, MetaOptNet, and Reptile, we use the architectures originally used for experiments in the respective papers [1, 2, 45, 131]. Specifically, Nichol and Schulman [1], Finn et al. [45] use the same network with 4 convolutional layers. Bertinetto et al. [2] uses a modified version of this convolutional network, while Lee et al. [131] employs a ResNet-12 architecture.

5.7.5 Proof of Theorem 1

Consider the three conditions

$$\|x - \overline{X}\| < \delta, \quad \|y - \overline{Y}\| < \delta, \quad \|z - \overline{X}\| < \delta,$$

where $\delta = \|\overline{X} - \overline{Y}\|/4$, and $\overline{X}$ is the expected value of X. Under these conditions,

$$\|z - x\| \leq \|z - \overline{X}\| + \|x - \overline{X}\| < 2\delta$$

and

$$\|z - y\| \geq \|\overline{X} - \overline{Y}\| - \|y - \overline{Y}\| - \|z - \overline{X}\| > 4\delta - 2\delta = 2\delta.$$

Combining the above yields

$$\|z - x\| < \|z - y\|.$$

We can now write

$$\begin{aligned} z^T(x-y) - \frac{1}{2}\|x\|^2 + \frac{1}{2}\|y\|^2 & \\ &= -\|z-x\|^2 + \frac{1}{2}\|z-y\|^2 + \frac{1}{2}\|z-x\|^2 \\ &\geq -\|z-x\|^2 + \frac{1}{2}\|z-x\|^2 + \frac{1}{2}\|z-x\|^2 \\ &= 0, \end{aligned}$$

and so z is classified correctly if our three conditions hold. From the Chebyshev bound, these conditions hold with probability at least

$$\left(1 - \frac{\sigma_x^2}{\delta^2}\right)^2 \left(1 - \frac{\sigma_y^2}{\delta^2}\right) \geq \left(1 - \frac{2\sigma_x^2}{\delta^2}\right)\left(1 - \frac{\sigma_y^2}{\delta^2}\right) \geq 1 - \frac{2\sigma_x^2 + \sigma_y^2}{\delta^2}, \tag{5.1}$$

where we have twice applied the identity $(1-a)(1-b) \geq (1-a-b)$, which holds for $a, b \geq 0$, (this also requires $\sigma_y^2/\delta^2 < 1$, but this can be guaranteed by choosing a sufficiently small ϵ as in

the statement of the theorem).

Finally, we have the variation ratio bound

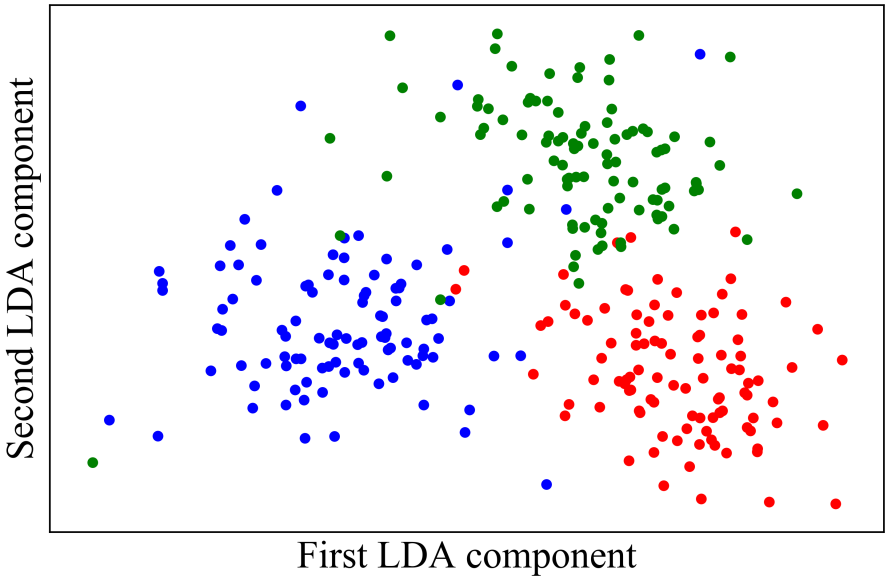
$$\frac{\text{var}[X] + \text{var}[Y]}{\text{var}[U]} = \frac{\sigma_x^2 + \sigma_y^2}{\sigma_x^2 + \sigma_y^2 + 16\delta^2} < \epsilon.$$

And so

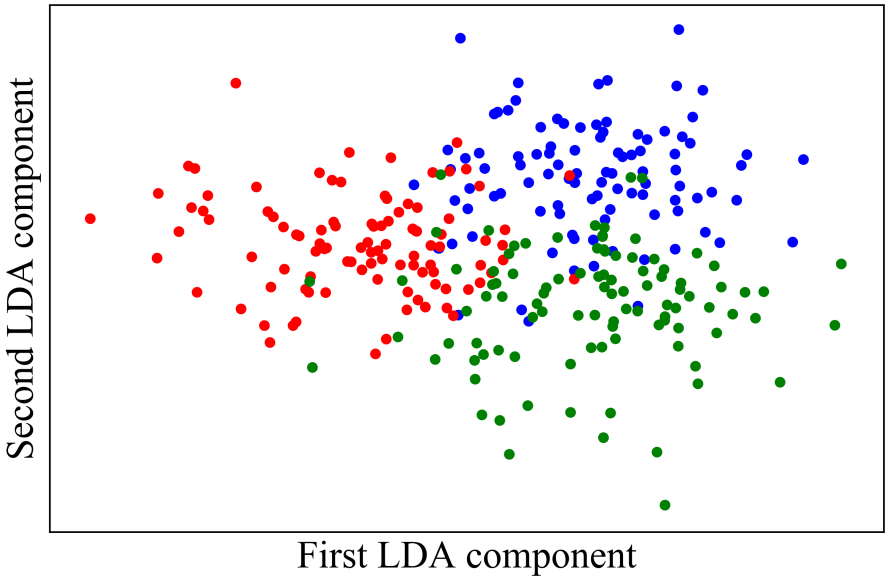
$$\delta^2 \geq \frac{(1-\epsilon)(\sigma_x^2 + \sigma_y^2)}{16\epsilon}.$$

Plugging this into equation 5.1 we get the final probability bound

$$1 - \frac{32\epsilon\sigma_x^2 + 16\epsilon\sigma_y^2}{(\sigma_x^2 + \sigma_y^2)(1-\epsilon)} \geq 1 - \frac{32\epsilon\sigma_x^2 + 32\epsilon\sigma_y^2}{(\sigma_x^2 + \sigma_y^2)(1-\epsilon)} = 1 - \frac{32\epsilon}{(1-\epsilon)}. \tag{5.2}$$

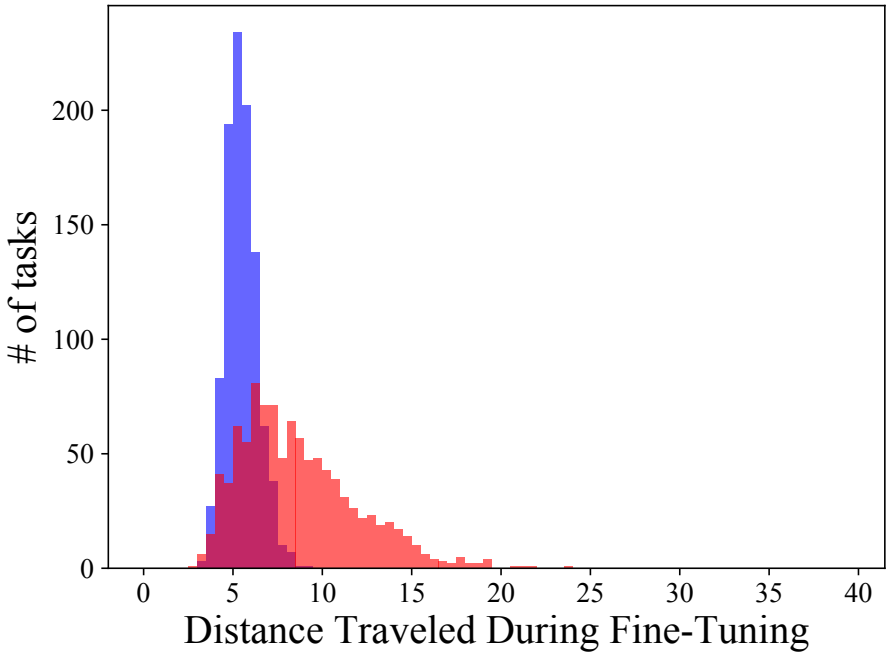


(a) Meta-Learning

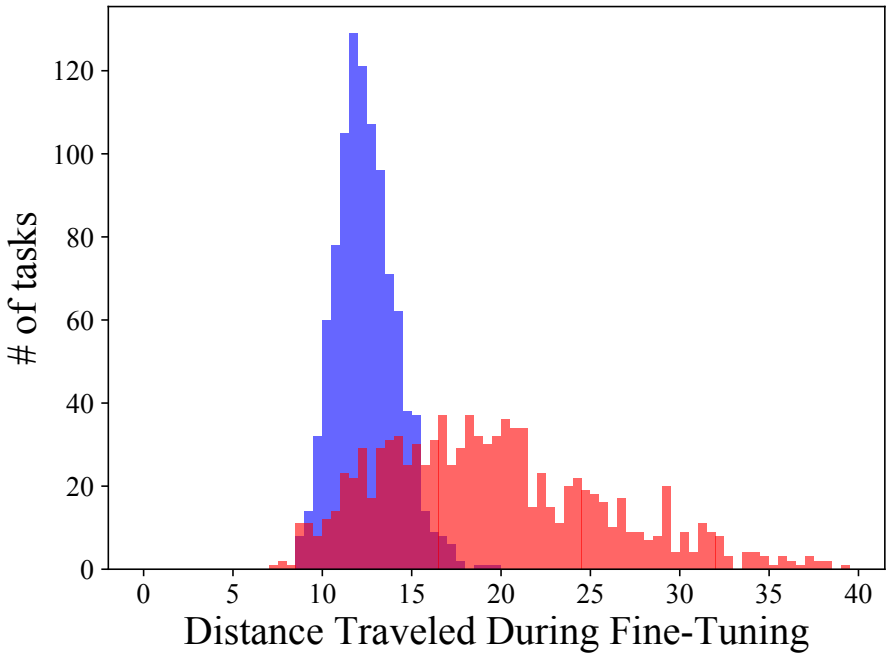


(b) Classically Trained

Figure 5.2: Features extracted from mini-ImageNet test data by a) ProtoNet and b) classically trained models with identical architectures (4 convolutional layers). The meta-learned network produces better class separation.



(a)



(b)

Figure 5.3: Histogram of filter normalized distance traveled during fine-tuning on a) 1-shot and b) 5-shot mini-ImageNet tasks by models trained using vanilla Reptile (red) and weight-clustered Reptile (blue).

Chapter 6: Data Augmentation for Meta-Learning

6.1 Introduction

Data augmentation has become an essential part of the training pipeline for image classifiers and similar systems, as it offers a simple and efficient way to significantly improve performance [123, 148]. In contrast, little work exists on data augmentation for meta-learning. Existing frameworks for few-shot image classification use only horizontal flips, random crops, and color jitter to augment images in a way that parallels augmentation for conventional training [2, 131]. Meanwhile, meta-learning methods have received increasing attention as they have reached the cutting edge of few-shot performance. While new meta-learning algorithms emerge at a rapid rate, we show that, like image classifiers, meta-learners can achieve significant performance boosts through carefully chosen data augmentation strategies that are injected into various stages of the meta-learning pipeline.

Meta-learning frameworks use data for multiple purposes during each gradient update, which creates the possibility for a diverse range of data augmentations that are not possible within the standard training pipeline. At the same time, it is still unclear how different categories of data within the training pipeline impact meta-learning performance. In this work [52], we explore these possibilities and discover combinations of augmentation types that improve performance over existing methods. Our contributions can be summarized as follows:

- First, we break down the meta-learning pipeline and find that each component contributes differently to meta-learning performance: meta-learners are very sensitive to the amount of query data and number of tasks and less sensitive to the amount of support data.

- Based on these findings, we uncover four modes of augmentations for meta-learning that differ in where in the training pipeline they are applied: support augmentation, query augmentation, task augmentation, and shot augmentation.

- We test these four modes using a pool of image augmentations, and we confirm that query augmentation is critical, while support augmentation often does not provide performance benefits and may even degrade accuracy in some cases.

- Finally, we combine augmentations and implement a MaxUp strategy, which we call Meta-MaxUp, to maximize performance. We achieve significant performance boosts for popular meta-learners on few-shot benchmarks such as mini-ImageNet, CIFAR-FS and Meta-Dataset.

6.2 Background and Related Work

6.2.1 The Meta-Learning Framework

Meta-learning algorithms aim to learn a network that can easily adapt to new tasks with limited data and generalize to unseen examples. In order to achieve this, they simulate the adaptation and evaluation procedure during meta-training. To simulate an N-way classification task, $\mathcal{T}_i$, we sample *support* data $\mathcal{T}_i^s$ and *query* data $\mathcal{T}_i^q$, so that $\mathcal{T}_i = \{\mathcal{T}_i^s, \mathcal{T}_i^q\}$. As we will detail in the following paragraph, support will be used to simulate few-shot training data, while query will be

used to simulate unseen testing data. Note that *shot* denotes the number of training samples per class available for fine-tuning on a given task during the testing phase.

Adopting common terminology from the literature, the archetypal meta-learning algorithm contains an *inner loop* and an *outer loop* in each parameter update of the training procedure. In the inner loop, a model is first fine-tuned or adapted on support data $\mathcal{T}_i^s$. Then, in the outer loop, the updated model is evaluated on query data $\mathcal{T}_i^q$, and minimizes loss on the query data with respect to the model's parameters before fine-tuning. This loss minimization step may require computing the gradient through the fine-tuning procedure. Existing meta-learning algorithms apply various methods for fine-tuning on support data during the inner loop. Some algorithms, such as MAML and Reptile [45, 46], update all the parameters in the network using gradient descent during fine-tuning on support data. Other algorithms, such as MetaOptNet and R2-D2 [2, 131], only update the parameters from the linear classifier layer during the fine-tuning while keeping the feature extraction layers frozen. These methods benefit from the simplicity and the convexity of the inner loop optimization problem. Similarly, metric learning approaches, such as [3, 149], freeze the feature extraction layers as well, and create class centroids from the support data during the inner loop. These methods have low-cost training iterations, and can be applied on deeper architectures to achieve better performance. In this work, we mainly focus on the latter algorithms due to their stronger performance. Further details of the algorithms used in our experiments can be found in Section 6.4.1.

6.2.2 Preventing Overfitting in Meta-Learning

Meta-learners are known to be particularly vulnerable to overfitting [150]. One work, MetaMix, proposes averaging support and query features to prevent the model from memorizing the query data and ignoring support [151]. Recently, another work adds random noise to the label space to make the model rely on support data [150]. In the context of few-shot classification, random shuffling labels within tasks alleviate this kind of overfitting and is commonplace in meta-learning algorithms [150, 152]. However, as shown in Figure 6.1, overfitting to training tasks remains a problem. One recent work has developed a data augmentation method to overcome this problem [4]. This method simply rotates all images in a class by a large degree and considers this new rotated class distinct from its parent class. This effectively increases the number of possible few-shot tasks that can be sampled during training.

A different line of work instead applies regularizers to prevent overfitting and improve few-shot classification [51, 152]. Yet additional work has developed methods for labeling and augmenting unlabeled data [153, 154], generative models for deforming images in one-shot metric learning [155], and feature space data augmentation for adapting language models to new unseen intents [156].

6.2.3 Few-shot Benchmarks

In this paper, we perform our experiments on the mini-ImageNet and CIFAR-FS datasets as well as the Meta-Dataset benchmark [2, 134, 157]. Mini-ImageNet is a few-shot learning dataset derived from the ImageNet classification dataset [6], and CIFAR-FS is derived from CIFAR-100 [158]. Each of these datasets contains 64 training classes, 16 validation classes, and 20

classes for testing. In each class, there are 600 images, and both Mini-ImageNet and CIFAR-FS have 60000 images in total. Meta-Dataset is a large-scale diverse benchmark consisting of 10 different image classification subdatasets with distinct data distributions. This diversity allows us to measure cross-domain generalization.

6.3 The Anatomy of Data Augmentation for Meta-Learning

6.3.1 Where Does Dataset Diversity Matter Most? In the Support, Query or Tasks?

Since data augmentation techniques aim to increase the amount of training samples, learning algorithms that are sensitive to the amount of training data may benefit more from these techniques. In this section, before we introduce data augmentations, we investigate how sensitive meta-learning algorithms are to the amount of support data, query data, and tasks. Typically, support and query data are sampled from the same pool (the entire training set).

To examine the impact of dataset diversity on various stages of meta-learning, we perform an ablation where we limit the diversity of each stage. We first reduce the pool of support data to a fixed subset of only five independent samples per class while sampling query data from the entire training set. That is, whenever a support image is sampled from class c, it is only sampled from the five-image subset associated with that class instead of from all training data in that class. Interestingly, we find that test accuracy remains almost the same as baseline performance (see Table 6.1). In fact, if we replace those five support images per class with fixed random noise images, we still only observe a small degradation in performance. We then instead shrink the pool of query data (but not support), and we see a much larger decrease in test accuracy. These

experiments suggest that meta-learning is fairly insensitive to the amount and quality of support but not query data. This observation agrees with our following finding that augmenting query data is far more beneficial than augmenting support.

Since we also consider task-level augmentation, we now examine how sensitive meta-learning is to a decrease in task diversity. As CIFAR-FS contains 64 training classes, there are $\binom{64}{5} = 7624512$ 5-way classification problems that can be sampled during each iteration of meta-learning. We reduce the number of tasks by randomly batching classes into just 13 distinct 5-way classification tasks before training, and we only train on these 13 tasks. We do this in such a way that all classes, and therefore training data, are used during training. We observe that this process noticeably degrades test accuracy, and we conclude that there may be room to improve performance by augmenting the number of tasks (see Table 6.1). To verify that this impact of dataset diversity generalizes, we run additional experiments on Mini-ImageNet and with other backbones. The results are shown in Appendix A, and these experiments support the aforementioned findings as well.

Table 6.1: Few-shot classification accuracy (%) using R2-D2 and a ResNet-12 backbone for various data size manipulations on CIFAR-FS. "Support", "Query" and "Task" columns denote the number of samples per class for support and query data and the number of total tasks available for sampling. The first row contains baseline performance. Confidence intervals have radius equal to one standard error.

Support	Query	Task	1-shot	5-shot
600	600	full	71.73 ± 0.37	84.39 ± 0.25
5	600	full	70.97 ± 0.36	84.51 ± 0.24
5 (random)	600	full	58.15 ± 0.36	76.26 ± 0.27
600	5	full	60.25 ± 0.37	77.05 ± 0.28
600	600	13	68.24 ± 0.38	81.77 ± 0.26

6.3.2 Data Augmentation Modes

Motivated by the observation that meta-learning is more sensitive to the amount of query data and tasks than support, we delineate four modes of data augmentation for meta-learning which may be employed individually or combined.

Support augmentation: Data augmentation may be applied to support data in the inner loop of fine-tuning. This strategy enlarges the pool of fine-tuning data.

Query augmentation: Data augmentation alternatively may be applied to query data. This strategy enlarges the pool of evaluation data to be sampled during training.

Task augmentation: We can increase the number of possible tasks by uniformly augmenting whole classes to add new classes with which to train. For example, a vertical flip applied to all car images yields a new upside-down car class which may be sampled during training.

Shot augmentation: At test time, we can artificially amplify the shot by adding additional augmented copies of each image. Shot augmentation can also be used during training by adding copies of each support image via augmentation. Shot augmentation during training may be needed to prepare a network for the use of test-time shot augmentation.

Existing meta-learning algorithms for few-shot image classification typically apply standard augmentations (horizontal flips, random crops, and color jitter) on all images that come from the data loader without considering the purpose of each image. As a result, the same augmentation occurs on both support and query images [159, 160]. In Section 6.4, we test the four modes of data augmentation enumerated above in isolation across a large array of specific augmentations. We find that query augmentation is far more critical than support augmentation for increasing performance. In fact, support augmentation often hurts performance. Additionally, we

find that task augmentation, when combined with query augmentation, can offer further boosts in performance when compared with existing frameworks.

6.3.3 Data Augmentation Techniques

For each of the data augmentation modes described above, we try a variety of specific data augmentation techniques. Some techniques are only applicable to support, query, and shot modes or solely to the task mode. We use an array of standard augmentation techniques as well as CutMix [124], MixUp [123], and Self-Mix [161]. In the context of the task augmentation mode, we apply these the same way to every image in a class in order to augment the number of classes. For example, we use MixUp to create a half-dog-half-truck class where every image is the average of a dog image and a truck image. We also try combining multiple classes into one class as a task augmentation mode.

In general, techniques that greatly change the image distribution (i.e. a vertical flip, which does not naturally appear in the dataset) are better suited for task augmentations while techniques that preserve the image distribution (e.g., random crops, which produce images that are presumably within the support of the image distribution) are typically better suited for the support, query, and shot augmentation modes. The baseline models we compare to use horizontal flip, random crop, and color jitter augmentation techniques at both the support and query levels since this combination is prevalent in the literature. More details on our pool of augmentation techniques can be found in Appendix B.

6.3.4 Meta-MaxUp Augmentation for Meta-Learning

Recent work proposes MaxUp augmentation to alleviate overfitting during the training of classifiers [162]. This strategy applies many augmentations to each image and chooses the augmented image which yields the highest loss. MaxUp is conceptually similar to adversarial training [163]. Like adversarial training, MaxUp involves solving a saddlepoint problem in which loss is minimized with respect to parameters while being maximized with respect to the input. In the standard image classification setting, MaxUp, together with CutMix, improves generalization and achieves state-of-the-art performance on ImageNet. Here, we extend MaxUp to the setting of meta-learning. Before training, we select a pool, $\mathcal{S}$, of data augmentations from the four modes as well as their combinations. For example, $\mathcal{S}$ may contain horizontal flip shot augmentation, query CutMix, and the combination of both. During each iteration of training, we first sample a batch of tasks, each containing support and query data, as is typical in the meta-learning framework. For each element in the batch, we randomly select m augmentations from the set $\mathcal{S}$, and we apply these to the task, generating m augmented tasks with augmented support and query data. Then, for each element of the batch of tasks originally sampled, we choose the augmented task that maximizes loss, and we perform a parameter update step to minimize training loss. Formally, we solve the minimax optimization problem,

$$\min_{\theta} \mathbb{E}_{\mathcal{T}} \Big[\max_{M \in \mathcal{S}} \mathcal{L}(F_{\theta'}, M(\mathcal{T}^q)) \Big],$$

where $\theta' = \mathcal{A}(\theta, M(\mathcal{T}^s))$, $\mathcal{A}$ denotes fine-tuning, F is the base model with parameters θ, $\mathcal{L}$ is the loss function used in the outer loop of training, and $\mathcal{T}$ is a task with support and query data

$\mathcal{T}^s$ and $\mathcal{T}^q$, respectively. Algorithm 5 contains a more thorough description of this pipeline in practice (adapted from the standard meta-learning algorithm in Goldblum et al. [164]).

Algorithm 5 Meta-MaxUp

Require: Base model F_θ, fine-tuning algorithm $\mathcal{A}$, learning rate γ, set of augmentations $\mathcal{S}$, and distribution over tasks $p(\mathcal{T})$.

Initialize θ, the weights of F;

while not done **do**
  Sample batch of tasks, $\{\mathcal{T}_i\}_{i=1}^n$, where $\mathcal{T}_i \sim p(\mathcal{T})$ and $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$.

  for $i = 1, ..., n$ **do**
    Sample m augmentations, $\{M_j\}_{j=1}^m$, from $\mathcal{S}$.

    Compute $k = \text{argmax}_j \, \mathcal{L}(F_{\theta_j}, M_j(\mathcal{T}_i^q))$, where $\theta_j = \mathcal{A}(\theta, M_j(\mathcal{T}_i^s))$.
    Compute gradient $g_i = \nabla_\theta \mathcal{L}(F_{\theta_k}, M_k(\mathcal{T}_i^q))$.
  Update base model parameters: $\theta \leftarrow \theta - \frac{\gamma}{n} \sum_i g_i$.

6.4 Experiments

In this section, we empirically demonstrate the following:

1. Augmentations applied in the four distinct modes behave differently. In particular, query and task augmentation are far more important than support augmentation. (Section 6.4.2)

2. Meta-specific data augmentation strategies can improve performance over the generic strategies commonly used for meta-learning. (Section 6.4.3)

3. We further boost performance by combining augmentations with Meta-MaxUp. (Section 6.4.4)

4. Our proposed augmentation Meta-MaxUp greatly improves performance on cross-domain benchmarks as well. (Section 6.4.7)

6.4.1 Experimental Setup

We conduct experiments on four meta-learning algorithms: ProtoNet [3], R2-D2 [2], MetaOptNet [131], and MCT [149]. ProtoNet is a metric-learning method that uses a prototype learning head, which classifies samples by extracting a feature vector and then performing a nearest-neighbor search for the closest class prototype. R2-D2 and MetaOptNet instead use differentiable solvers with a ridge regression and SVM head, respectively. These methods extract feature vectors and then apply a standard linear classifer to assign class labels. MCT improves upon ProtoNet by meta-learning confidence scores. We experiment with all of these different classifier head options, all using the ResNet-12 backbone proposed by Oreshkin et al. [147] as well as the four-layer convolutional architectures proposed by Snell et al. [3] and Bertinetto et al. [2].

We perform our experiments on the aforementioned benchmark datasets, mini-ImageNet, CIFAR-FS, and Meta-Dataset. A description of training hyperparameters and computational complexity can be found in Appendix C. We report confidence intervals with a radius of one standard error.

Few-shot learning may be performed in either the inductive or transductive setting. Inductive learning is a standard method in which each test image is evaluated separately and independently. In contrast, transduction is a mode of inference in which the few-shot learner has access to all unlabeled testing data at once and therefore has the ability to perform semi-supervised learning by training on the unlabelled data. For fair comparison, we only compare inductive methods to other inductive methods. A PyTorch implementation of our data augmentation methods for meta-learning can be found at: `https://github.com/RenkunNi/MetaAug`

6.4.2 An Empirical Comparison of Augmentation Modes

We empirically evaluate the performance of all four different augmentation modes identified in Section 6.3.2 on the CIFAR-FS dataset using an R2-D2 base-learner paired with both a 4-layer convolutional network backbone (as used in the original work [2]) and a ResNet-12 backbone. We report the results of the most effective augmentations for each mode on the ResNet-12 backbone in Table 6.2. Appendix D contains an extensive table with various augmentations and both backbones.

Table 6.2 demonstrates that each mode of augmentation individually can improve performance. Augmentation applied to query data is consistently more effective than the other augmentation modes. In particular, simply applying CutMix to query samples improves accuracy by as much as 3% on both backbones. In contrast, most augmentations on support data actually damage performance. The overarching conclusion of these experiments is that the four modes of data augmentation for meta-learning behave differently. Existing meta-learning methods, which apply the same augmentations to query and support data without using task and shot augmentation, may achieve suboptimal performance.

6.4.3 Combining Augmentations

After studying each mode of data augmentation individually, we combine augmentations in order to find out how augmentations interact with each other. We build on top of query CutMix since this augmentation was the most effective in the previous section. We combine query CutMix with other effective augmentations from Table 6.2, and we conduct experiments on the same backbones and dataset. Results on the ResNet-12 backbone are reported in Table 6.3, and a full

Table 6.2: Few-shot classification accuracy (%) using R2-D2 and a ResNet-12 backbone on the CIFAR-FS dataset with the most effective data augmentations for each mode shown. Confidence intervals have radii equal to one standard error. The best performance in each category is bolded. Query CutMix is consistently the most effective single augmentation for meta-learning.

Method	Mode	1-shot	5-shot
Baseline	-	71.95 ± 0.37	84.56 ± 0.25
CutMix	Support	72.79 ± 0.37	84.70 ± 0.25
Self-Mix	Support	71.96 ± 0.36	84.84 ± 0.25
CutMix	Query	$\mathbf{75.97 \pm 0.34}$	$\mathbf{87.28 \pm 0.23}$
Self-Mix	Query	73.59 ± 0.35	86.14 ± 0.24
Large Rotation	Task	73.79 ± 0.36	85.81 ± 0.24
MixUp	Task	72.05 ± 0.37	85.27 ± 0.25
Random Crop	Shot	70.56 ± 0.37	83.87 ± 0.25
Horizontal Flip	Shot	73.25 ± 0.36	85.06 ± 0.25

table with additional results can be found in Appendix E.

Interestingly, when we use CutMix on both support and query images, we observe worse performance than simply using CutMix on query data alone. Again, this demonstrates that meta-learning demands a careful and meta-specific data augmentation strategy. In order to further boost performance, we will need an intelligent method for combining various augmentations. We propose Meta-MaxUp as this method.

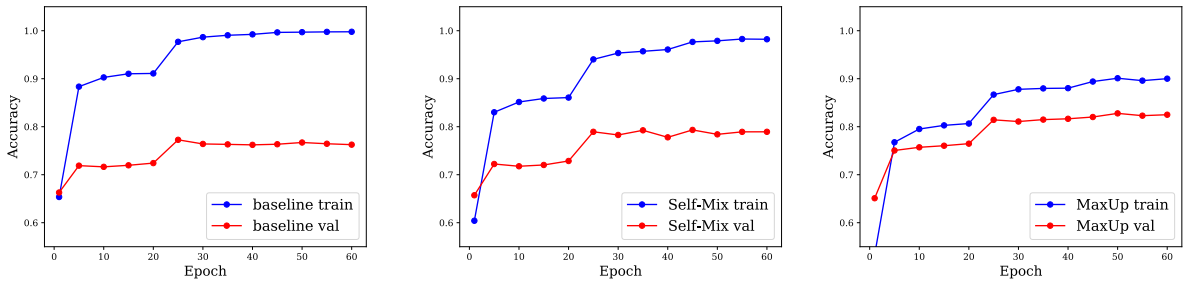


Figure 6.1: Training and validation accuracy for R2-D2 meta-learner with ResNet-12 backbone on the CIFAR-FS dataset. (Left) Baseline model (Middle) query Self-Mix (Right) Meta-MaxUp. Better data augmentation strategies, such as MaxUp, narrow the generalization gap and prevent overfitting.

Table 6.3: Few-shot classification accuracy (%) using R2-D2 and a ResNet-12 backbone on the CIFAR-FS dataset with combinations of augmentations and query CutMix. "S","Q","T" denote "Support", "Query", and "Task" modes, respectively. While adding augmentations can help, it can also hurt, so additional augmentations must be chosen carefully.

Mode	1-shot	5-shot
CutMix	75.97 ± 0.34	87.28 ± 0.23
+ CutMix (S)	75.00 ± 0.37	85.37 ± 0.25
+ Random Erase (S)	75.84 ± 0.34	87.19 ± 0.24
+ Random Erase (Q)	75.08 ± 0.35	87.14 ± 0.23
+ Self-Mix (S)	$\mathbf{76.27 \pm 0.34}$	87.52 ± 0.24
+ Self-Mix (Q)	76.04 ± 0.34	87.45 ± 0.24
+ MixUp (T)	75.97 ± 0.34	86.66 ± 0.24
+ Rotation (T)	75.74 ± 0.34	$\mathbf{87.68 \pm 0.24}$
+ Horizontal Flip (Shot)	76.23 ± 0.34	87.36 ± 0.24

6.4.4 Meta-MaxUp Further Improves Performance

In this section, we evaluate our proposed Meta-MaxUp strategy in the same experimental setting as above for various values of m and different data augmentation pool sizes. Table 6.4 contains the results, and a detailed description of the augmentation pools as well as the full results can be found in Appendix F. Rows beginning with "CutMix" denote experiments in which the pool of augmentations simply includes many CutMix samples. "Single" denotes experiments in which each augmentation in $\mathcal{S}$ is of a single type, while "Medium" and "Large" denote experiments in which each element of $\mathcal{S}$ is a combination of augmentations, for example CutMix+rotation. Combinations greatly expand the number of augmentations in the pool. Rows with $m = 1$ denote experiments where we do not maximize loss in the inner loop and thus simply apply randomly sampled data augmentation for each task. As we increase m and include a large number of augmentations in the pool, we observe performance boosts as high as 4% over the baseline, which uses horizontal flip, random crop, and color jitter data augmentations from

the original work corresponding to the R2-D2 meta-learner [2].

Table 6.4: Few-shot classification accuracy (%) using R2-D2 and a ResNet-12 backbone on the CIFAR-FS dataset for Meta-MaxUp over different sizes of augmentation pools and numbers of samples. As m and the pool size increase, so does performance. Meta-MaxUp is able to pick effective augmentations from a large pool.

Pool	m	1-shot	5-shot
Baseline	-	71.95 ± 0.37	84.56 ± 0.25
CutMix	1	75.97 ± 0.34	87.28 ± 0.23
Single	1	75.71 ± 0.35	87.44 ± 0.43
Medium	1	75.60 ± 0.34	87.35 ± 0.23
Large	1	75.44 ± 0.34	87.47 ± 0.23
CutMix	2	74.93 ± 0.36	87.14 ± 0.24
Single	2	75.81 ± 0.34	87.33 ± 0.23
Medium	2	76.49 ± 0.33	88.20 ± 0.22
Large	2	76.59 ± 0.34	88.11 ± 0.23
CutMix	4	75.08 ± 0.23	87.60 ± 0.24
Single	4	76.82 ± 0.24	88.14 ± 0.23
Medium	4	76.30 ± 0.24	88.29 ± 0.22
Large	4	$\mathbf{76.99 \pm 0.24}$	$\mathbf{88.35 \pm 0.22}$

We explore the training benefits of these meta-specific training schemes by examining saturation during training. To this end, we plot the training and validation accuracy over time for R2-D2 meta-learners with ResNet-12 backbones using baseline augmentations, query Self-Mix, and Meta-MaxUp with a medium sized pool and $m = 4$. See Figure 6.1 for training and validation accuracy curves. With only baseline augmentations, validation accuracy stops increasing immediately after the first learning rate decay. This suggests that baseline augmentations do not prevent overfitting during meta-training. In contrast, we observe that models trained with Meta-MaxUp do not quickly overfit and continue improving validation performance for a greater number of epochs. Meta-MaxUp visibly reduces the generalization gap.

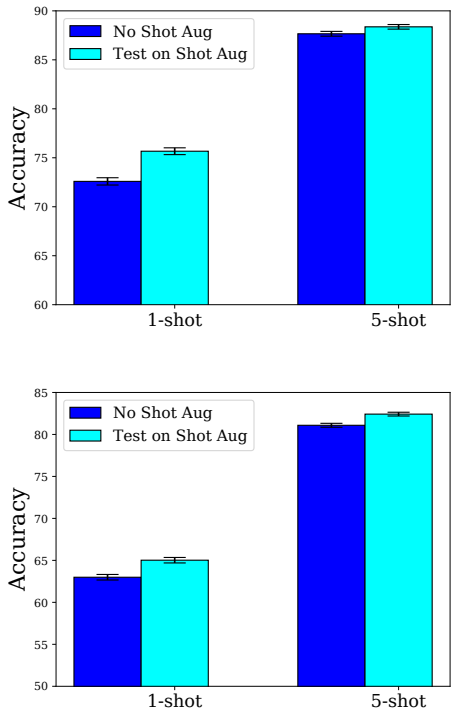


Figure 6.2: Performance with shot augmentation using MetaOptNet trained with the proposed Meta-MaxUp. (Top) 1-shot and 5-shot on CIFAR-FS (Bottom) 1-shot and 5-shot on mini-ImageNet.

6.4.5 Shot Augmentation for Pre-Trained Models

In the typical meta-learning framework, data augmentations are used during meta-training but not during test time. On the other hand, in some transfer learning work, data augmentations, such as horizontal flips, random crops, and color jitter, are used during fine-tuning at test time [135]. These techniques enable the network to see more data samples during few-shot testing, leading to enhanced performance.

We propose shot augmentation (see Section 6.3) to enlarge the number of few-shot samples during testing, and we also propose a variant in which we additionally train using the same augmentations on support data in order to prepare the meta-learner for this test time scenario. Figure 6.2 shows the effect of shot augmentation (using only horizontal flips) on performance

Table 6.5: Few-shot classification accuracy (%) on CIFAR-FS and mini-ImageNet. "+ DA" denotes training with CutMix (Q) + Rotation (T), and "+ MM" denotes training with Meta-MaxUp. "CNN-4" denotes a 4-layer convolutional network with 96, 192, 384, and 512 filters in each layer [2]. "64-64-64-64" denotes the 4-layer CNN backbone from Snell et al. [3].

		CIFAR-FS		mini-ImageNet	
Method	Backbone	1-shot	5-shot	1-shot	5-shot
R2-D2	CNN-4	67.56 ± 0.35	82.39 ± 0.26	56.15 ± 0.31	72.46 ± 0.26
+ DA	CNN-4	70.54 ± 0.33	84.69 ± 0.24	57.60 ± 0.32	74.69 ± 0.25
+ MM	CNN-4	**71.10 ± 0.34**	**85.50 ± 0.24**	**58.18 ± 0.32**	**75.35 ± 0.25**
R2-D2	ResNet-12	71.95 ± 0.37	84.56 ± 0.25	60.46 ± 0.32	76.88 ± 0.24
+ DA	ResNet-12	76.17 ± 0.34	87.74 ± 0.24	**65.54 ± 0.32**	81.52 ± 0.23
+ MM	ResNet-12	**76.65 ± 0.33**	**88.57 ± 0.24**	65.15 ± 0.32	**81.76 ± 0.24**
ProtoNet	64-64-64-64	60.91 ± 0.35	79.73 ± 0.27	47.97 ± 0.32	70.13 ± 0.27
+ DA	64-64-64-64	62.21 ± 0.36	80.70 ± 0.27	**50.38 ± 0.32**	**71.44 ± 0.26**
+ MM	64-64-64-64	**63.01 ± 0.36**	**80.85 ± 0.25**	50.06 ± 0.32	71.13 ± 0.26
ProtoNet	ResNet-12	70.21 ± 0.36	84.26 ± 0.25	57.34 ± 0.34	75.81 ± 0.25
+ DA	ResNet-12	74.30 ± 0.36	86.24 ± 0.24	60.82 ± 0.34	78.23 ± 0.25
+ MM	ResNet-12	**76.05 ± 0.34**	**87.84 ± 0.23**	**62.81 ± 0.34**	**79.38 ± 0.24**
MetaOptNet	ResNet-12	70.99 ± 0.37	84.00 ± 0.25	60.01 ± 0.32	77.42 ± 0.23
+ DA	ResNet-12	74.56 ± 0.34	87.61 ± 0.23	64.94 ± 0.33	82.10 ± 0.23
+ MM	ResNet-12	**75.67 ± 0.34**	**88.37 ± 0.23**	**65.02 ± 0.32**	**82.42 ± 0.23**
MCT	ResNet-12	75.80 ± 0.33	89.10 ± 0.42	64.84 ± 0.33	81.45 ± 0.23
+ MM	ResNet-12	**76.00 ± 0.33**	**89.54 ± 0.33**	**66.37 ± 0.32**	**83.11 ± 0.22**

for MetaOptNet with ResNet-12 backbone trained with Meta-MaxUp. Shot augmentation consistently improves results across datasets, especially on 1-shot classification ($\sim 2\%$). To be clear, in this figure, we are not using shot augmentation during the training stage. Rather, we are using conventional low-shot training, and then deploying our models with shot augmentation at test time. These post-training performance gains can be achieved by directly applying shot augmentation to pre-trained/existing models during testing. For additional experiments, see Appendix G.

6.4.6 Improving Existing Meta-Learners with Better Data Augmentation

In this section, we improve the performance of four different popular meta-learning methods including ProtoNet [3], R2-D2 [2], MetaOptNet [131], and MCT [149]. We compare their baseline performance to query CutMix with task-level rotation as well as Meta-MaxUp data augmentation strategies on both the CIFAR-FS and mini-ImageNet datasets. See Table 6.5 for the results of these experiments. In all cases, we are able to improve the performance of existing methods, sometimes by over 5%. Even without Meta-MaxUp, we improve performance over the baseline by a large margin. The superiority of meta-learners that use these augmentation strategies suggests that data augmentation is critical for these popular algorithms and has largely been overlooked.

In addition, we compare our method to augmentation by Large Rotations at the task level – the only competing work to our knowledge – in Table 6.6. Note that using Large Rotations to create new classes is referred to as "Task Augmentation" in [4]; we refer to it here as "Large Rotations" to avoid confusion since we study a myriad of augmentations at the task level. We observe that with the same training algorithm (MetaOptNet with SVM) and the ResNet-12 backbone, our method outperforms the Large Rotations augmentation strategy by a large margin on both the CIFAR-FS and mini-ImageNet datasets. Together with the same ensemble method as used in Large Rotations, marked by "+ens", we further boost performance consistently above the MCT baseline, the current highest performing meta-learning method on these benchmarks, despite using an older meta-learner previously thought to perform worse than MCT. Moreover, when both training and validation datasets are used for meta-training, we can achieve the state-of-art results for few-shot classification on mini-ImageNet in inductive setting.

Table 6.6: Few-shot classification accuracy (%) on CIFAR-FS and mini-ImageNet with ResNet-12 backbone. "M-SVM" denotes MetaOptNet with the SVM head. "+ens" denotes testing with ensemble methods as in Liu et al. [4]. "LargeRot" denotes task-level augmentation by Large Rotations as described in Liu et al. [4].

	CIFAR-FS		mini-ImageNet	
Method	1-shot	5-shot	1-shot	5-shot
M-SVM + LargeRot	72.95 ± 0.24	85.91 ± 0.18	62.12 ± 0.22	78.90 ± 0.17
M-SVM + MM (ours)	75.67 ± 0.34	88.37 ± 0.23	65.02 ± 0.32	82.42 ± 0.23
M-SVM + LargeRot + ens	75.85 ± 0.24	87.73 ± 0.17	64.56 ± 0.22	81.35 ± 0.16
M-SVM + MM + ens (ours)	76.38 ± 0.33	89.16 ± 0.22	66.42 ± 0.32	83.69 ± 0.21
M-SVM + LargeRot + ens + val	$\mathbf{76.75 \pm 0.23}$	88.38 ± 0.17	65.38 ± 0.23	82.13 ± 0.16
M-SVM + MM + ens + val (ours)	76.38 ± 0.34	$\mathbf{89.25 \pm 0.21}$	$\mathbf{67.37 \pm 0.32}$	$\mathbf{84.57 \pm 0.21}$

6.4.7 Out-of-Distribution Testing on Meta-Dataset

In this section, we examine the effectiveness of our methods on cross-domain few-shot learning benchmarks. Few-shot learners may be successful on similar tasks to their training data but fail on tasks that deviate. Thus, testing on diverse distributions is crucial. To this end, we leverage Meta-Dataset, a collection of sub-datasets used for testing meta-learners across diverse tasks [157]. Among the 10 subdatasets, we train the networks only on ILSVRC-2012 [165], the largest dataset in the collection, and we evaluate the cross-domain few-shot classification performance on the other 9 datasets with R2-D2 and MetaOptNet learners and ResNet-12 backbones. Training and evaluation details can be found in Appendix H.

We observe that on all subdatasets except for Omniglot, our proposed methods can improve test accuracy over the baseline by as much as 7%. Additionally, we improve performance by a large margin (more than 3%) on more than half of the subdatasets. On average, Meta-MaxUp improves accuracy by around 3%. Omniglot suffers under our strategies since this dataset comprises handwritten letters which are not invariant to strong augmentations. Specially designed

augmentations for handwritten letters are necessary to optimize performance on Omniglot. The success of Meta-MaxUp on cross-domain benchmarks demonstrates that the proposed strategy is effective even on diverse testing distributions which do not resemble the learner's training data.

Table 6.7: Few-shot classification accuracy (%) on Meta-Dataset with both MetaOptNet and R2-D2 learner. "+ DA" denotes training with CutMix (Q) + Rotation (T), and "+ MM" denotes training with Meta-MaxUp. Confidence intervals have radius equal to one standard error.

Test Source	R2-D2	+ DA	+ MM
ILSVRC	69.04 ± 0.31	70.30 ± 0.31	**71.68 ± 0.30**
Birds	75.22 ± 0.30	77.27 ± 0.28	**77.95 ± 0.30**
Omniglot	**97.46 ± 0.08**	96.10 ± 0.11	96.71 ± 0.09
Aircraft	54.28 ± 0.28	58.93 ± 0.30	**60.83 ± 0.28**
Textures	63.47 ± 0.24	65.98 ± 0.24	**67.34 ± 0.26**
Quick Draw	76.39 ± 0.27	78.44 ± 0.27	**80.83 ± 0.25**
Fungi	50.41 ± 0.22	52.29 ± 0.20	**54.12 ± 0.22**
VGG Flower	86.26 ± 0.21	87.79 ± 0.19	**90.29 ± 0.17**
Traffic Signs	83.98 ± 0.34	**84.23 ± 0.36**	83.59 ± 0.36
MSCOCO	70.29 ± 0.30	71.59 ± 0.31	**72.83 ± 0.29**

Test Source	MetaOptNet	+ DA	+ MM
ILSVRC	68.92 ± 0.30	71.17 ± 0.30	**72.19 ± 0.30**
Birds	75.58 ± 0.39	**77.49 ± 0.29**	77.47 ± 0.2
Omniglot	**97.43 ± 0.10**	95.97 ± 0.10	96.59 ± 0.09
Aircraft	53.40 ± 0.37	60.43 ± 0.29	**60.57 ± 0.29**
Textures	63.29 ± 0.33	65.70 ± 0.24	**69.42 ± 0.25**
Quick Draw	78.00 ± 0.33	79.56 ± 0.25	**80.67 ± 0.25**
Fungi	50.56 ± 0.21	53.80 ± 0.22	**53.82 ± 0.22**
VGG Flower	88.16 ± 0.25	89.92 ± 0.18	**91.13 ± 0.15**
Traffic Signs	85.12 ± 0.33	**85.25 ± 0.33**	83.38 ± 0.37
MSCOCO	69.52 ± 0.32	71.90 ± 0.31	**73.49 ± 0.30**

6.5 Discussion

In this work, we break down data augmentation in the context of meta-learning. In doing so, we uncover possibilities that do not exist in the classical image classification setting. We identify four modes of augmentation: query, support, task, and shot. These modes behave differently

and are of varying importance. Specifically, we find that augmenting query data is particularly important. After adapting various data augmentations to meta-learning, we propose Meta-MaxUp for combining various meta-specific data augmentations. We demonstrate that Meta-MaxUp significantly improves the performance of popular meta-learning algorithms. As shown by the recent popularity of frameworks like AutoAugment [148] and MaxUp [162], data augmentation for standard classification is still an active area of research. We hope that this work opens up possibilities for further work on meta-specific data augmentation and that emerging methods for data augmentation will boost the performance of meta-learning on progressively larger models with more complex backbones.

6.6 Appendix

6.6.1 Impact of Dataset Diversity on Various Stages of Meta-learning

Following the settings in Section 6.3.1, we further investigate how dataset diversity matters in the various stages of meta-learning on Mini-ImageNet, CNN-4 backbones, and with the ProtoNet [3] head. The results are shown in Table 6.8, and all these results support our findings that meta-learning algorithms are more sensitive to the amount of query data and the number of tasks and less sensitive to the amount of support data.

6.6.2 Details About Data Augmentation Techniques

In this section, we provide more details about the different data augmentation techniques we use in this work. We employ the following pool of data augmentations:

CutMix: Yun et al. [124] introduce the CutMix augmentation strategy where patches are

Table 6.8: Few-shot classification accuracy (%) using different meta-learning algorithms and backbones for various data size manipulations on Mini-ImageNet. "Support", "Query" and "Task" columns denote the number of samples per class for support and query data and the number of total tasks available for sampling. Confidence intervals have radius equal to one standard error.

Method	Backbone	Support	Query	Task	1-shot	5-shot
R2-D2	CNN-4	600	600	full	55.94 ± 0.32	72.32 ± 0.25
R2-D2	CNN-4	5	600	full	55.05 ± 0.31	71.66 ± 0.26
R2-D2	CNN-4	5 (random)	600	full	42.09 ± 0.29	60.12 ± 0.27
R2-D2	CNN-4	600	5	full	49.87 ± 0.30	66.00 ± 0.27
R2-D2	CNN-4	600	600	13	53.34 ± 0.31	69.43 ± 0.25
R2-D2	ResNet-12	600	600	full	60.50 ± 0.33	76.60 ± 0.24
R2-D2	ResNet-12	5	600	full	58.79 ± 0.32	76.45 ± 0.25
R2-D2	ResNet-12	5 (random)	600	full	43.80 ± 0.30	62.26 ± 0.28
R2-D2	ResNet-12	600	5	full	48.02 ± 0.31	65.45 ± 0.26
R2-D2	ResNet-12	600	600	13	57.65 ± 0.33	73.18 ± 0.28
ProtoNet	ResNet-12	600	600	full	57.46 ± 0.38	75.61 ± 0.29
ProtoNet	ResNet-12	5	600	full	57.03 ± 0.33	75.46 ± 0.26
ProtoNet	ResNet-12	5 (random)	600	full	43.36 ± 0.32	57.20 ± 0.28
ProtoNet	ResNet-12	600	5	full	48.40 ± 0.34	64.79 ± 0.29
ProtoNet	ResNet-12	600	600	13	51.88 ± 0.35	66.41 ± 0.29

cut and pasted among training images, and the ground truth labels are also mixed proportionally to the area of the patches.

MixUp: Zhang et al. [123] propose mixup, a simple learning principle to alleviate memorization and sensitivity to adversarial examples. Mixup trains a neural network on convex combinations of pairs of examples and their labels. By doing so, mixup regularizes the neural network to favor simple linear behavior in between training examples.

Self-Mix: Seo et al. [161] introduce the self-mix augmentation strategy in which a patch of an image is substituted into other values in the same image to improve the generalization ability of few-shot image classification models.

In addition, we use some standard and simple data augmentation techniques:

Rotation: augments the data by rotating the images.

Horizontal Flip: augments the data by horizontally flipping images.

Random Erase: augments the data by randomly erasing patches from the image.

Finally, we also experimented with the following data augmentation techniques:

Combining Labels: augments the data by combining two different labels into a single class. For instance, we may combine the "dog" and "cat" labels to create a new "dog or cat" class.

Feature Mixup: similar to the "Mixup" augmentation technique we describe above, however, we perform the mixup strategy on the feature representation for the image.

Drop Channel: augments the data by dropping color channels in the image.

Solarize: inverts all pixels above a threshold value of magnitude.

6.6.3 Training Details

For MetaOptNet, we use the same training procedure as [131] including SGD with Nesterov momentum of 0.9 and weight decay coefficient of 0.0005. The model was meta-trained for 60 epochs, with an initial learning rate of 0.1, then changed to 0.006, 0.0012, and 0.00024 at epochs 20, 40 and 50, respectively. In each epoch, we train on 8000 episodes and use mini-batches of size 8. Following [131], we use a larger shot number (15) to train mini-ImageNet for both 1-shot and 5-shot classification. For MCT, we use the same optimizer but with batch size 1 and maximum iteration 50000. Following [149], we enlarge the training classification ways to 15 for a 5-way testing. We use instance-wise metrics for all inductive learning.

Table 6.9 compares the training time of meta-learning methods with baseline data augmentations, with our proposed data augmentations (DA) and Meta-MaxUp strategy (MM) on the CIFAR-FS dataset. We employ data parallelism across 4 Nvidia RTX 2080 Ti GPUs for all experiments. The training time of meta algorithms with our proposed data augmentation is almost the same as with baseline methods. Although Meta-MaxUp requires m times as many forward passes, here $m = 4$, it does not require any extra backward passes. Thus, Meta-MaxUp typically runs roughly 2-3 times longer than baseline methods.

Table 6.9: Runtime (training time in hours for 60 epochs) comparison of data augmentation strategies on CIFAR-FS

Method	Backbone	Runtime	Backbone	Runtime
R2D2	CNN-4	2.5h	ResNet-12	3.2h
+ DA	CNN-4	2.6h	ResNet-12	3.7h
+ MM	CNN-4	4.1h	ResNet-12	8.2h
MetaOptNet	CNN-4	8.6h	ResNet-12	8.9h
+ DA	CNN-4	8.8h	ResNet-12	9.2h
+ MM	CNN-4	14.5h	ResNet-12	18.5h

6.6.4 Results for All Data Augmentation Techniques

Table 6.6.4 shows the few-shot classification accuracy on CIFAR-FS of an R2-D2 meta-learner with all single data augmentation techniques used in our paper. We highlight the best result in each mode. Data augmentation on query images significantly improves the baseline performance as well as data augmentations on other modes.

Table 6.10: Few-shot classification accuracy (%) on the CIFAR-FS dataset for all data augmentations with an R2-D2 learner. Confidence intervals have radius equal to one standard error. "CNN-4" denotes a 4-layer convolutional network with 96, 192, 384, and 512 filters in each layer [2]. Best performance in each category is bolded.

		CNN-4		ResNet-12	
Mode	Level	1-shot	5-shot	1-shot	5-shot
Baseline	-	67.56 ± 0.35	82.39 ± 0.26	71.95 ± 0.37	84.56 ± 0.25
Random Erase	Support	67.71 ± 0.36	82.25 ± 0.26	72.30 ± 0.37	84.50 ± 0.25
Self-Mix	Support	$\mathbf{69.61 \pm 0.35}$	$\mathbf{83.43 \pm 0.25}$	71.96 ± 0.36	$\mathbf{84.84 \pm 0.25}$
CutMix	Support	69.05 ± 0.36	83.12 ± 0.26	$\mathbf{72.79 \pm 0.37}$	84.70 ± 0.25
MixUp	Support	68.64 ± 0.37	82.72 ± 0.27	71.86 ± 0.37	84.11 ± 0.25
Feature Mixup	Support	67.88 ± 0.35	82.40 ± 0.25	71.21 ± 0.37	83.38 ± 0.25
Rotation	Support	68.65 ± 0.35	82.86 ± 0.25	71.13 ± 0.37	83.84 ± 0.25
Combining labels	Support	68.27 ± 0.36	82.53 ± 0.26	71.00 ± 0.38	83.12 ± 0.25
Drop Channel	Support	68.21 ± 0.35	82.76 ± 0.25	69.65 ± 0.73	83.15 ± 0.25
Solarize	Support	68.65 ± 0.35	82.68 ± 0.26	70.88 ± 0.37	83.45 ± 0.25
Random Erase	Query	69.73 ± 0.34	84.04 ± 0.25	73.05 ± 0.36	85.67 ± 0.25
Self-Mix	Query	69.55 ± 0.35	84.20 ± 0.25	73.59 ± 0.35	86.14 ± 0.25
CutMix	Query	$\mathbf{70.54 \pm 0.33}$	$\mathbf{84.69 \pm 0.24}$	$\mathbf{75.97 \pm 0.34}$	$\mathbf{87.28 \pm 0.23}$
MixUp	Query	67.70 ± 0.34	83.13 ± 0.25	72.93 ± 0.35	86.13 ± 0.24
Feature Mixup	Query	70.16 ± 0.35	83.80 ± 0.28	73.38 ± 0.35	85.87 ± 0.23
Rotation	Query	68.17 ± 0.35	83.01 ± 0.25	72.02 ± 0.36	84.42 ± 0.25
Combining labels	Query	66.01 ± 0.34	81.99 ± 0.26	69.77 ± 0.37	82.99 ± 0.26
Drop Channel	Query	68.34 ± 0.35	83.25 ± 0.25	69.60 ± 0.37	83.01 ± 0.26
Solarize	Query	67.51 ± 0.35	82.65 ± 0.25	72.45 ± 0.36	84.97 ± 0.24
MixUp	Task	67.21 ± 0.35	82.72 ± 0.26	72.05 ± 0.37	85.27 ± 0.25
Large Rotation	Task	$\mathbf{68.96 \pm 0.35}$	$\mathbf{83.65 \pm 0.25}$	$\mathbf{73.79 \pm 0.36}$	$\mathbf{85.81 \pm 0.24}$
CutMix	Task	68.78 ± 0.36	82.99 ± 0.50	72.72 ± 0.37	84.62 ± 0.25
Combining labels	Task	68.08 ± 0.35	82.33 ± 0.26	69.64 ± 0.37	83.79 ± 0.26
Random Erase	Task	68.39 ± 0.36	83.26 ± 0.25	71.09 ± 0.37	84.49 ± 0.25
Drop Channel	Task	67.54 ± 0.36	81.97 ± 0.25	70.24 ± 0.37	83.52 ± 0.26
Horizontal Flip	Shot	$\mathbf{68.13 \pm 0.35}$	82.95 ± 0.25	$\mathbf{73.25 \pm 0.36}$	$\mathbf{85.06 \pm 0.25}$
Random Crop	Shot	67.33 ± 0.36	$\mathbf{83.04 \pm 0.25}$	70.56 ± 0.37	83.87 ± 0.25
Random Rotation	Shot	67.57 ± 0.35	83.00 ± 0.25	70.32 ± 0.37	83.75 ± 0.25

6.6.5 Results for Combination of Data Augmentations

Table 6.11 shows the few-shot classification accuracy for combinations of data augmentations building on the top of query CutMix, with both CNN-4 and ResNet-12 backbones.

Table 6.11: Few-shot classification accuracy (%) on the CIFAR-FS dataset with combinations of augmentations and query CutMix. "S","Q","T" denote "Support", "Query", and "Task" modes, respectively. While adding augmentations can help, it can also hurt, so additional augmentations must be chosen carefully.

	CNN-4		ResNet-12	
Mode	1-shot	5-shot	1-shot	5-shot
CutMix	70.54 ± 0.33	84.69 ± 0.24	75.97 ± 0.34	87.28 ± 0.23
+ CutMix (S)	69.50 ± 0.35	82.64 ± 0.26	75.00 ± 0.37	85.37 ± 0.25
+ Random Erase (S)	70.12 ± 0.35	84.48 ± 0.25	75.84 ± 0.34	87.19 ± 0.24
+ Random Erase (Q)	69.68 ± 0.34	84.36 ± 0.24	75.08 ± 0.35	87.14 ± 0.23
+ Self-Mix (S)	70.65 ± 0.34	84.68 ± 0.25	$\mathbf{76.27 \pm 0.34}$	87.52 ± 0.24
+ Self-Mix (Q)	69.94 ± 0.34	84.38 ± 0.24	76.04 ± 0.34	87.45 ± 0.24
+ MixUp (T)	70.33 ± 0.35	84.57 ± 0.25	75.97 ± 0.34	86.66 ± 0.24
+ Rotation (T)	70.35 ± 0.34	84.73 ± 0.24	75.74 ± 0.34	$\mathbf{87.68 \pm 0.24}$
+ Horizontal Flip (Shot)	$\mathbf{70.90 \pm 0.33}$	$\mathbf{84.87 \pm 0.24}$	76.23 ± 0.34	87.36 ± 0.24

6.6.6 Augmentation Pool for Meta-MaxUp

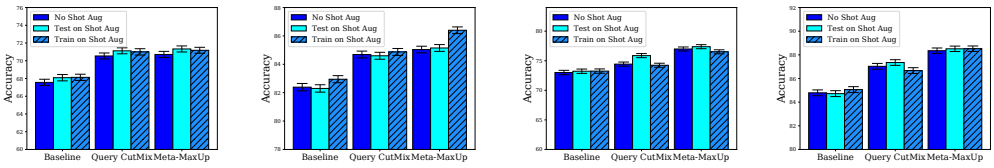
For all the benchmark results of Meta-MaxUp training, we use a medium-size data augmentation pool with $m = 4$, including CutMix (Q), Random Erase (Q), Self-Mix (S), Rotation (T), CutMix (Q) + Rotation (T), and Random Erase (Q) + Rotation (T). For the large-size pool, we add more techniques and combinations of the mentioned techniques into the pool, including Random Erase (Q) + Random Erase (S), CutMix (Q) + Random Erase (S), CutMix (Q) + Random Erase (Q), and CutMix (Q) + Self-Mix (S). Table 6.12 shows the few-shot classification accuracy on CIFAR-FS using R2D2 meta-learner and both CNN-4, ResNet-12 backbones, with various augmentations pool and hyper-parameter m.

Table 6.12: Few-shot classification accuracy (%) on the CIFAR-FS dataset for Meta-MaxUp over different sizes of augmentation pools and numbers of samples. As m and the pool size increase, so does performance. Meta-MaxUp is able to pick effective augmentations from a large pool.

		CNN-4		ResNet-12	
Pool	m	1-shot	5-shot	1-shot	5-shot
Baseline	-	67.56 ± 0.36	82.39 ± 0.26	71.95 ± 0.37	84.56 ± 0.25
CutMix	1	70.54 ± 0.34	84.69 ± 0.24	75.97 ± 0.34	87.28 ± 0.23
Single	1	70.76 ± 0.35	84.70 ± 0.25	75.71 ± 0.35	87.44 ± 0.43
Medium	1	70.50 ± 0.34	84.59 ± 0.24	75.60 ± 0.34	87.35 ± 0.23
Large	1	70.84 ± 0.34	85.04 ± 0.24	75.44 ± 0.34	87.47 ± 0.23
CutMix	2	70.56 ± 0.34	84.78 ± 0.24	74.93 ± 0.36	87.14 ± 0.24
Single	2	70.86 ± 0.34	85.06 ± 0.25	75.81 ± 0.34	87.33 ± 0.23
Medium	2	70.75 ± 0.34	85.02 ± 0.24	76.49 ± 0.33	88.20 ± 0.22
Large	2	70.63 ± 0.34	85.07 ± 0.24	76.59 ± 0.34	88.11 ± 0.23
CutMix	4	70.48 ± 0.34	84.76 ± 0.24	75.08 ± 0.23	87.60 ± 0.24
Single	4	$\mathbf{71.10 \pm 0.34}$	$\mathbf{85.50 \pm 0.24}$	76.82 ± 0.24	88.14 ± 0.23
Medium	4	70.58 ± 0.34	85.32 ± 0.24	76.30 ± 0.24	88.29 ± 0.22
Large	4	70.71 ± 0.34	85.04 ± 0.23	$\mathbf{76.99 \pm 0.24}$	$\mathbf{88.35 \pm 0.22}$

6.6.7 Bar Plots for Shot Augmentation

Figure 6.3 shows the effect of the shot augmentation in few-shot evaluation. In general, shot augmentation enhances the performance of meta-learners.



(a) (b) (c) (d)

Figure 6.3: Performance with shot augmentation using different backbones and training strategies on CIFAR-FS. (a) 1-shot classification with CNN-4 (b) 5-shot classification with CNN-4 (c) 1-shot classification with ResNet-12 (d) 5-shot classification with ResNet-12

6.6.8 Meta-Dataset Training and Evaluation

Details concerning each dataset in the Meta-Dataset benchmark can be found in Triantafillou et al. [157]. We use the same training procedure as mentioned above in Appendix 6.6.3 for both meta-learners. In each epoch, we train on 8000 episodes with shot of 5 and images of spacial dimensions $84 \times 84 \times 3$, and we use mini-batches of 8 tasks each. When training with Meta-MaxUp, we use the same augmentation pool as in Appendix 6.6.6 and set $m = 4$. During evaluation, we test 5-shot performance on 1000 tasks consisting of 15 query samples each. Due to the small number of sample size for several classes in the Fungi dataset, we use 1-shot classification with 5 query samples instead.

Chapter 7: The Close Relationship Between Contrastive Learning and Meta-Learning

7.1 Introduction

Self-supervised visual representation learning (SSL) has recently gathered attention due to its ability to learn image features without manual supervision, thus allowing for efficient learning on downstream tasks such as detection and segmentation [5, 40, 41, 47, 48, 49, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178]. Among SSL approaches, contrastive learning based methods [39, 40, 47] show particularly strong potential and achieve promising results which are close to those of fully supervised methods on numerous computer vision benchmarks.

These methods rely on applying various data augmentations such as random crops, flips, color distortion, and Gaussian blur on the same training sample to create different views of an image. Two such example methods, SimCLR [40] and MoCo [47], involve reducing the distance between features corresponding to positive pairs (different augmented views of the same image), and increasing the distance between features corresponding to negative pairs (augmented views of different images).

Meanwhile, meta-learning is an established popular framework for learning models that quickly adapt to on-the-fly tasks given a small number of examples [2, 45, 46, 131, 179]. The

training loop for meta-learners typically involves (i) sampling a random batch of classes and (ii) updating a feature extractor to distinguish between these classes. This procedure mirrors that of contrastive learning, which proceeds by (i) sampling a batch of images and augmenting them to generate classes (each "class" is an image plus all of its views), and (ii) updating a feature extractor to distinguish between these classes. Conceptually, this contrastive learning procedure resembles meta-learning where the training tasks are generated by computing multiple views of individual images.

In this work [53], we discuss the close relationship between contrastive learning and meta-learning. Concretely, we show that established meta-learning algorithms, originally designed for few-shot learning, can achieve the same performance as recent contrastive learning algorithms on standard SSL problems when paired with the same data sampling strategy. In addition, we explore techniques, originally designed for meta-learning, that can improve contrastive learning. Specifically, we explore ways by which we can adapt data augmentation strategies inspired by recent work in meta-learning [52, 180] to SSL and find that this approach can yield significant performance boosts.

Our contributions can be summarized as follows:

- We formulate a meta-learning based framework for understanding self-supervised learning, and we show that meta-learners can achieve comparable self-supervised performance to contrastive learning methods.

- We propose a meta-specific task augmentation strategy which boosts the performance of self-supervised learning. This data augmentation method generalizes to methods with no negative pairs, such as BYOL [48], as well.

7.2 Related Work and Background

7.2.1 Meta-Learning for Few-shot Learning

Meta-learning algorithms for few-shot learning aim to learn a model that can quickly adapt to new tasks with limited data and generalize to unseen examples. To achieve this, the adaptation and evaluation procedures are both simulated during meta-training. During each episode of meta-learning, we sample a task, $\mathcal{T}_i$, from a distribution of tasks, often corresponding to combinations of training classes formed into classification problems. Each task consists of *support* data $\mathcal{T}_i^s$ and *query* data $\mathcal{T}_i^q$, so that $\mathcal{T}_i = \{\mathcal{T}_i^s, \mathcal{T}_i^q\}$. When applied to few-shot classification, this task is called a k-shot, N-way classification problem, where k denotes the number of training samples per category in the support data. Then, support data will be used to simulate few-shot training data, while query data will be used to simulate novel testing samples.

A meta-learning model F, in this setting, contains a feature extractor and a classification strategy, $\mathcal{A}$. This classification strategy can take various forms, such as adding a linear classifier on top of the feature extractor and fine-tuning either the linear layer or the whole network end-to-end, or this strategy may simply classify samples by selecting the nearest class prototype. Meta-learning training algorithms have an *inner loop* and an *outer loop* in each parameter update. In the inner loop, the model is first fine-tuned on support data $\mathcal{T}_i^s$. Then, in the outer loop, the updated model is used to predict on query data $\mathcal{T}_i^q$, and a loss is minimized with respect to the model's parameters before fine-tuning.[1] Intuitively, we update parameters so that the feature extractor extracts better features for the classification strategy, often resulting in tightly clustered

[1]Note that algorithms in the vein of Reptile [46] do not split the tasks into support and query.

features corresponding to each class [51]. Existing works apply various methods for fine-tuning on support data during the inner loop. In a line of algorithms, such as MAML and Reptile [45, 46], all the parameters in the model are updated using gradient descent during fine-tuning on support data. Other algorithms, such as MetaOptNet and R2-D2 [2, 131], keep the feature extractor frozen during fine-tuning; MetaOptNet uses SVM, and R2-D2 uses ridge regression on top of the feature extractor. Similarly, metric learning approaches, such as ProtoNet [3, 149], freeze the feature extractor as well, and create class centroids from the support data in the inner loop. In this work, we primarily focus on the latter algorithms due to their efficiency and performance as well as the similarity of contrastive learning to metric learning.

7.2.2 self-supervised learning

Contrastive SSL. Contrastive methods [39, 40, 47, 168, 170, 173] achieve promising performance in self-supervised learning. As mentioned previously, contrastive learning maximizes agreement on different augmented views of the same image (called *positive pairs*) while ensuring disagreement on samples generated by different base images (called *negative pairs*). Given a batch of input images $\boldsymbol{x}$, m random data augmentations are applied on the same batch, generating a set of training samples $\{\tilde{\boldsymbol{x}}^i\}_{i=1}^m$. These samples are fed into a backbone network $f(\cdot)$ to obtain the feature representations $\{\boldsymbol{h}^i\}_{i=1}^m$. Then, a small neural network $g(\cdot)$, usually a non-linear MLP, is applied to project $\{\boldsymbol{h}^i\}_{i=1}^m$ to the latent representations $\{\boldsymbol{z}^i\}_{i=1}^m$ in the space where a contrastive loss $l(\cdot)$ is applied, ensuring that latent representations of positive pairs are similar while latent representations of negative pairs are different. In this work, we mainly focus on this type of self-supervised learning and show its close relation with meta-learning. The batch sam-

pling procedure of contrastive learning can be viewed as sampling a new classification problem with a number of classes equal to the number of base images used to generate augmented views. We will see that this on-the-fly sampling of classification problems closely mirrors a common meta-learning setup.

Non-Contrastive SSL. Some non-contrastive methods are generative approaches, such as auto-encoders [181, 182, 183], and adversarial learning [184], where a distribution is learned over data and a latent embedding. These methods are typically computationally expensive as they require training a learned model which maps latent representations to pixel space. Other non-contrastive methods rely on using heuristic designed pretext tasks [166, 167, 171, 185] to learn the representation. More recently, BYOL [48] showed that by bootstrapping a target representation prediction, feature representations can be learned without negative pairs. However, BYOL still adopts the data augmentation procedure from contrastive learning, where different augmented views are used as training samples. In section 7.5, we show that BYOL still benefits from our proposed task augmentation strategy.

Data Augmentation in Meta-Learning and SSL. Data augmentations play an essential role in both meta-learning and self-supervised learning. In meta-learning, Liu et al. [4], Ni et al. [52], Su et al. [180] show that proper data augmentation and meta-specific task augmentations dramatically improve few-shot learning performance by expanding the number of classes available for sampling. In self-supervised learning, Tian et al. [186] show contrastive learners can find better feature representations when views contain less mutual information. In addition, Shen et al. [5], Kim et al. [174], Li et al. [177] show that adding harder examples such as cut-mixed samples into the training pipeline can improve self-supervised performance. In this work, we show that similarly to meta-learning, self-supervised learners can benefit from carefully applied

data augmentation techniques which mirror task augmentations from the meta-learning literature.

7.2.3 self-supervised learning for meta-learning and few-shot learning

Another line of research [187, 188, 189, 190] focuses on self-supervised learning for meta-learning and few-shot learning. Hsu et al. [187] focuses on partitioning samples from a dataset to construct meta-learning tasks and using MAML or ProtoNet on 4-layer architectures to solve few-shot problems. Similarly, Khodadadeh et al. [188] and Medina et al. [190] add more data augmentations, such as Auto Aug [148] and use MAML and ProtoNet, respectively, to learn a few-shot representation. To further improve few-shot performance, Ye et al. [189] sample harder mixed support examples and apply a task-specific projection head to help generalize to unseen classes. These methods focus on few-shot learning performance, which entails up to 50 training examples per class. In contrast, we focus on the unsupervised learning paradigm where large models are pre-trained on samples generated via data augmentation and are applied to downstream tasks such as ImageNet classification.

7.3 Experimental Setup

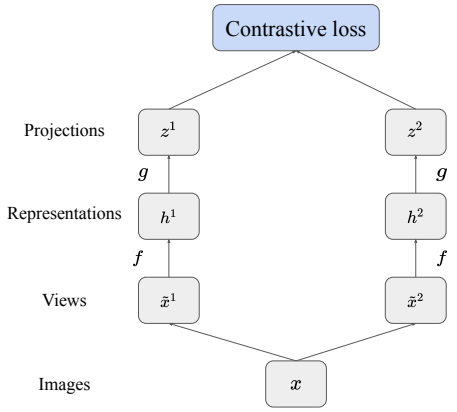
Datasets and Evaluation We conduct self-supervised training on both the CIFAR-10 and ImageNet datasets [6, 158]. Following Chen et al. [40], we evaluate pre-trained representations in a linear evaluation setting, where feature extractors are frozen, and a classification head is stacked on top and tuned. In addition, we test the performance of the pre-trained feature extractors on downstream tasks such as transfer learning and semi-supervised learning with 1% and 10% of labeled data. Evaluation and dataset details can be found in Appendix 7.7.2.

Pre-Training Details We use a ResNet-18 backbone for all experiments on CIFAR-10 and ResNet-50 for those on ImageNet. We train the model on CIFAR-10 with the LARS optimizer [191] and batch size 1024 for 1000 epochs (with 4 GPUs). On ImageNet, we use the same optimizer and batch size of 256, and we train for 100 epochs (with 8 GPUs). For ImageNet pre-training, we follow the hyperparameter setting in Chen et al. [40], including baseline data augmentation methods, dimension of the latent space, and learning rate decay schedule. For CIFAR-10 pre-training, we use the same CIFAR-10 specific hyperparameters as SimCLR again. For BYOL, we use the same learning rate schedule as meta-learners and start with learning rate 4. In addition, both the projector and predictor in BYOL are two-layer MLPs with hidden dimension 2048 and output dimension 256. More details can be found in Appendix 7.7.1.

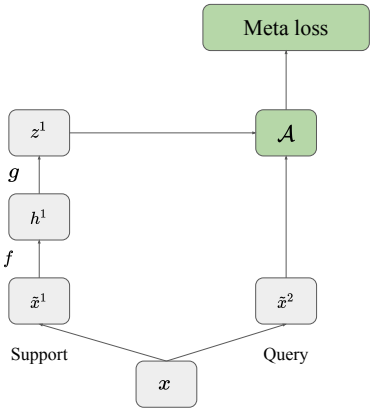
7.4 A Meta-Learning Framework for SSL

Meta-learners, designed for few-shot learning, and contrastive learners for SSL are built on similar intuitions. Both approaches learn to solve new tasks on-the-fly with each batch – new classification problems in the case of meta-learning and differentiating a new batch of images in the case of contrastive learning. Furthermore, both approaches hold a goal of learning invariances which generalize to novel problems at inference; meta-learners should extract similar features for each instance of a novel test class, and contrastive learners should extract similar features for each view of an image sample. In this section, we show how one can construct a meta-learning framework for SSL which closely mirrors the strategy adopted by recent contrastive learning methods.

We now describe how to generate meta-learning task distributions $p(\mathcal{T})$ for self-supervised



(a) Training procedure of contrastive learning



(b) Meta-Learning framework for SSL

Figure 7.1: (a) Training procedure of contrastive learning. Two augmented views are generated by applying random transformations to the same input batch. A backbone $f(\cdot)$ and a projector $g(\cdot)$ is learned through contrastive prediction tasks. (b) Meta-Learning framework for SSL. We adopt the same data augmentation operations as contrastive learning. We generate a b-way classification problem, b is the batch size, by treating each image itself as a class. Two views are separated as *support* data, on which the network is fine-tuned and the classification strategy is learned by $\mathcal{A}$; *query* data, on which we apply the updated model and calculate the meta-loss. At the end of training, everything but $f(\cdot)$ is discarded, and $\boldsymbol{h}$ is used as the image representation.

learning. We adopt the data augmentation operations from contrastive learning, where different random augmentations are applied to the input batch to generate alternative views. In general, given m random augmented views of a batch of b input images $\boldsymbol{x}$, we can create a b-way classification problem by treating all images generated by the same base image as a class. Then, we can divide the data from each class into m_1 support and m_2 query samples so that $m_1 + m_2 = m$. This framework for sampling a batch containing augmented views of base images and dividing them into support and query samples yields a task distribution $p(\mathcal{T})$. In few-shot learning terminology, each training task $\mathcal{T}_i$ is a m_1-shot-b-way classification problem, since we have b base images which generate classes, and for each such image, we have m_1 support samples. Viewed in this way, SimCLR sets $m = 2$ and $m_1 = m_2 = 1$, but SimCLR has important differences from common meta-learners.

Unlike typical meta-learning methods, SimCLR compares every sample with every other

sample, while methods like ProtoNet only compare each query sample with each support prototype and not with each other. Moreover, SimCLR samples a single large batch of samples for each episode of training which corresponds to sampling a single task, while meta-learners typically sample a batch of many tasks, i.e., classification problems, during each episode.

Now that we have established a framework for sampling tasks, we can directly apply various meta-learning algorithms, such as R2-D2 and ProtoNet described in Section 7.2.1, in order to learn the parameters θ of the base model F. Recall that the base model contains a classification strategy and a feature extractor, which is the learning target of SSL. We use the same feature extractor here used for contrastive learning in Section 7.2.2, which consists of a backbone $f(\cdot)$ followed by a projection head $g(\cdot)$. Formally, we solve the meta-learning optimization problem,

$$\min_{\theta} \mathbb{E}_{\mathcal{T}}\Big[\mathcal{L}_{SSL}\Big], \quad \mathcal{L}_{SSL} = l(F_{\theta'}, \mathcal{T}^q),$$

where $\theta' = \mathcal{A}(\theta, \mathcal{T}^s)$ are parameters updated by training on support tasks, and l is the loss function, e.g., cross-entropy loss in our work, used in the outer loop of training. After pre-training, only the backbone $f(\cdot)$ will be kept for self-supervised evaluation. This meta-learning framework for self-supervised learning is summarized in Algorithm 6 and Figure 7.1.

We compare the performance of representations learned by meta-learners with SimCLR

Table 7.1: Linear evaluation on CIFAR-10 for representations learned via contrastive learning and our meta-learning framework.

Method	Backbone	Top-1 Acc(%)
SimCLR	ResNet-18	91.4
ProtoNet	ResNet-18	91.8
R2-D2	ResNet-18	91.6

Table 7.2: Linear evaluation on ImageNet for representations learned via contrastive learning and our meta-learning framework.

Method	Backbone	Top-1 Acc(%)
SimCLR	ResNet-50	58.8
ProtoNet	ResNet-50	57.6
R2-D2	ResNet-50	55.5

Algorithm 6 Meta-Learning Framework for Self-Supervised Learning

Require: Base model F_θ, classification strategy $\mathcal{A}$, learning rate γ, and distribution of data augmentations $\mathcal{D}$.
Initialize: θ, the weights of F
while not done **do**
  for $j = 1, ..., n$ **do**
    Sample a batch of base images $\boldsymbol{x}$
    Sample m random data augmentations from $\mathcal{D}$ to obtain augmented views $\{\tilde{\boldsymbol{x}}^{\boldsymbol{i}}\}_{i=1}^{m}$
    Separate $\{\tilde{\boldsymbol{x}}^{\boldsymbol{i}}\}_{i=1}^{m}$ into support set $\mathcal{T}_j^s = \{\tilde{\boldsymbol{x}}^{\boldsymbol{i}}\}_{i=1}^{m_1}$ and query set $\mathcal{T}_j^q = \{\tilde{\boldsymbol{x}}^{\boldsymbol{i}}\}_{i=m_1+1}^{m}$
    Fine-tune model on $\mathcal{T}_j$: $\theta_j = \mathcal{A}(\theta, \mathcal{T}_j^s)$
    Compute gradient $g_j = \nabla_\theta \mathcal{L}(F_{\theta_j}, \mathcal{T}_j^q)$;
  Update base model parameters: $\theta \leftarrow \theta - \frac{\gamma}{n} \sum_j g_j$.

under the default setting described in Section 7.3. Table 7.1 and Table 7.2 show the linear evaluation top-1 accuracy for the feature representations trained and tested on the CIFAR-10 and ImageNet datasets, respectively. We observe that representations learned via meta-learning (R2-D2 and ProtoNet) can achieve performance on par with SimCLR on CIFAR-10 but worse on ImageNet. Note that during training, we use the same exact hyperparameter as SimCLR due to computational constraints, which may not be optimal specifically for our proposed method. We will see in the following experiments that although R2-D2 achieves worse linear evaluation on ImageNet with this hyperparameter setting, it actually performs better than SimCLR on downstream tasks, such as semi-supervised learning and transfer learning, other popular (and plausibly more realistic) evaluation scenarios for SSL methods.

Following Chen et al. [40], we first evaluate the pre-trained model by semi-supervised learning, where we fine-tune the pre-trained model with only a fraction of labeled ImageNet data (1% and 10%). As we see from Table 7.3, with the same fine-tune setting (See Appendix 7.7.2), models pre-trained by R2-D2 can achieve $\sim 5\%$ higher top-1 accuracy than those pre-trained by SimCLR after fine-tuning on labeled data. Notably, the supervised baseline from Zhai et al. [192] is strong due to exhaustive hyperparameter searching and stronger data augmentations used

during the training.

To further compare the feature representations learned by different methods, we apply the pre-trained weights to transfer learning. We consider 8 datasets with natural images of various categories [158, 193, 194, 195, 196, 197, 198]. For each dataset, we use the backbone (ResNet-50) pre-trained on ImageNet as an initialization for the feature extractor of the downstream classification model. In contrast to linear evaluation, we fine-tune the entire model on the given dataset for 20,000 iterations with the best hyperparameter setting selected on its validation split. Details of our hyperparameter selection are included in Appendix 7.7.2. All models are pre-trained on ImageNet for 100 epochs. We also include a baseline model provided in Chen et al. [40] which does not use pre-trained weight as an initialization. Note that the baseline model is tuned to achieve comparable performance with a larger search space for hyperparameters, and it is trained for a longer duration. From the results in Table 7.4, we find that R2-D2 initialized model consistently outperforms its contrastive counterpart on all 8 datasets. These results suggest that for the same number of epochs, a model trained with R2-D2 works better as an initialization for downstream tasks than one trained with SimCLR. We speculate that this property is connected to R2-D2's few-shot learning driven design and simulation of adapting to new tasks during its inner loop.

7.5 Boosting SSL with Meta-Specific Augmentation

Now that we have established a relationship between contrastive learning and meta-learning, we will apply tools developed in the latter discipline to enhance contrastive learners. Previous work has shown that data augmentations such as crops and colorizing play an important role

Table 7.3: ImageNet Top-1 accuracy (%) of models fine-tuned with few labels.

Method	Backbone	Label fraction	
		1%	10%
Supervised baseline	ResNet-50	25.4	56.4
SimCLR	ResNet-50	32.4	53.6
ProtoNet	ResNet-50	31.0	52.9
R2-D2	ResNet-50	**37.9**	**58.8**

Table 7.4: Transfer learning using ImageNet pre-trained weights. We report mean per-class accuracy (%) on the Flowers and Aircraft datasets, mean average precision (mAP) on the VOC2007 classification dataset, and Top-1 accuracy on the remaining datasets.

	Flowers102	DTD	VOC2007	Aircraft	Food101	SUN397	CIFAR-10	CIFAR-100
Baseline	92.0	64.8	67.3	85.9	**86.9**	53.6	95.9	80.2
SimCLR	92.4	72.7	66.0	83.7	86.3	57.4	94.8	79.1
ProtoNet	92.7	71.5	64.7	83.9	86.2	56.4	96.0	79.1
R2-D2	**94.5**	**73.8**	**69.9**	**86.2**	**86.9**	**59.7**	**96.7**	**82.8**

in both contrastive learning and meta-learning. We focus on a particular augmentation strategy from the meta-learning literature, termed *task augmentation*, which aims to expand the number of classes available for sampling rather than expanding the number of samples per class. Liu et al. [4], Ni et al. [52] show empirically that data augmentations work best when applied to carefully chosen parts of the meta-learning batch, and large rotations can only work as a task augmentation, in which rotation by a chosen degree is applied to all images in an entire class, and we then treat them as a new class. Large rotations, and other dramatic transformations used for task augmentation, actually decrease performance when instead applied independently on support and query samples (as a way to increase data within a class) rather than uniformly on an entire class (therefore defining a new class). Augmentations that exhibit this task augmentation behavior are typically those which transform an image so much that its semantic content looks different to a

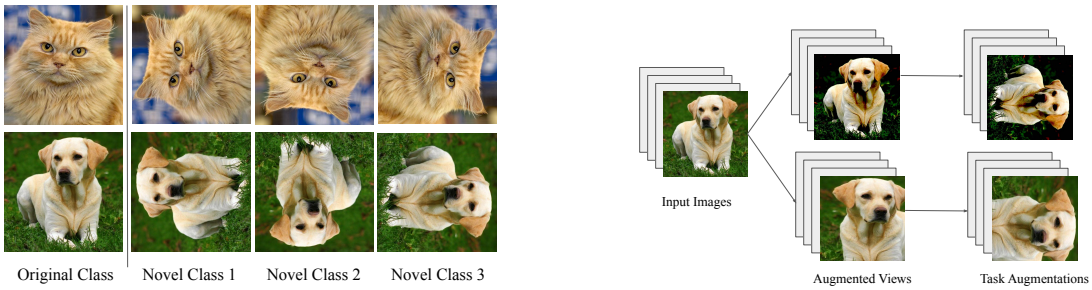
human. By keeping images with very large augmentations in the same class, we may accidentally encourage models to learn overly strong invariances which do not naturally exist in the data.

Table 7.5: Linear evaluation (Top-1 accuracy (%)) on CIFAR-10 with feature representations learned by SimCLR and BYOL in default setting (see Section 7.3). Simply adding large rotations to data augmentation hurts the performance of self-supervised learning.

Rotation	SimCLR	BYOL
No	91.4	92.1
Yes	89.7	90.6

Although data augmentations such as large rotations have been shown effective for visual pre-training [171, 199], how to encode that into the data augmentation framework of contrastive learning remains unclear. In addition, we observe that when applied along with contrastive learning, the phenomenon mentioned above occurs as well. Namely, the same large rotation data augmentation, which can improve the performance of meta-learners via task augmentation, also degrades the performance of contrastive learners when applied to samples independently (instead of uniformly to an entire class). Table 7.5 shows linear evaluation accuracy on CIFAR-10 when we add this augmentation to the training pipeline of SimCLR. In this experiment, we randomly rotate every augmented view by $\{90°, 180°, 270°\}$ with probability 0.25 each. In Table 7.5, we see that the accuracy of SimCLR drops by $\sim 2\%$, and this degradation also occurs in self-supervised learning methods without negative pairs such as BYOL, which adopts the same augmentation pipeline as SimCLR. Driven by the observation and insights from the meta-learning literature, we are motivated to apply strong augmentations, such as large rotations, to contrastive learning at the task level so that models can benefit from the additional augmentation without learning overly strong and harmful invariances.

In the literature on meta-learning literature for few-shot classification, large rotations can be



(a) (b)

Figure 7.2: (a) Examples of novel classes created by rotation by $\{90°, 180°, 270°\}$. (b) Adding task augmentation into the data augmentation pipeline. We first apply random transformations on the input batch, then for each augmented view from the same image, we rotate them by the same degree.

used either as a task augmentation or an auxiliary loss [4, 180, 200]. We adopt both methods into our meta-learning inspired pipeline for self-supervised learning and describe the details below. This procedure is illustrated in Figure 7.2 and Figure 7.3.

Large Rotation as Task Augmentation. Instead of randomly rotating each training sample independently, we rotate all images from the same class (different augmented views of the same original base image) by the same degree (chosen randomly from $\{0°, 90°, 180°, 270°\}$) in both the support and the query data. In such a way, the number of potential classes is enlarged by 3 times. We combine large rotation augmentation with the basic augmentations used in contrastive learning during the sampling stage and keep other components of the training procedure unchanged.

Large Rotation as Auxiliary Loss. In addition to task augmentation, large rotations can be used as an auxiliary prediction task for the original self-supervised problem, where the angle of a rotated image is used as the target label. To this end, we spin the input batch $\boldsymbol{x}$ by an angle to generate the rotated training samples, $\{\boldsymbol{x}_d, \boldsymbol{y}\}$, where $d \in \{0°, 90°, 180°, 270°\}, \boldsymbol{y} = \{d/90\}$. Then, we stack a 4-way classification head r on top of the shared backbone f and projection head

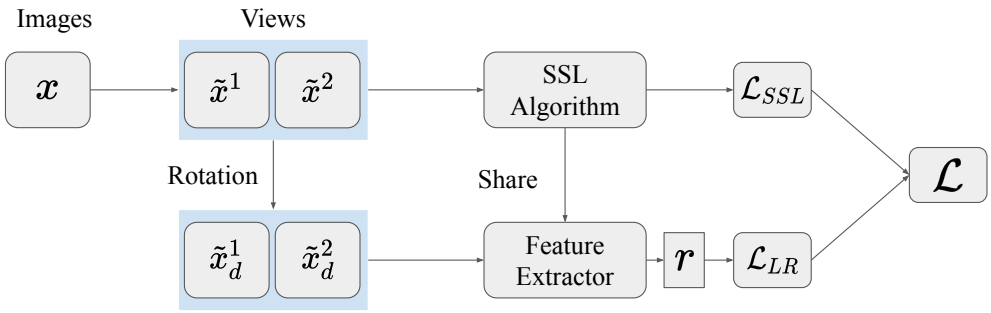


Figure 7.3: Training procedure with Large Rotation augmentation as an auxiliary loss. We randomly rotate each augmented view by a degree in $\{0°, 90°, 180°, 270°\}$. We share the feature extractor used in the SSL algorithm and apply it along with a 4-way linear head $r(\cdot)$ to predict the rotation angle. We sum up the SSL loss, $\mathcal{L}_{SSL}$, with the angle prediction loss, $\mathcal{L}_{LR}$, as our ultimate objective, $\mathcal{L}$.

g, to predict the angle of the rotations evaluated with cross-entropy loss l_{LR}:

$$\mathcal{L}_{LR} = \sum_{(x,y)\in\{\boldsymbol{x}_d,\boldsymbol{y}\}} l_{LR}(r(g(f(x))), y). \tag{7.1}$$

During training, we sum this loss with the original self-supervised loss to achieve the final objective, $\mathcal{L} = \mathcal{L}_{SSL} + \lambda\mathcal{L}_{LR}$, where λ is a coefficient that controls the influence of our prediction task. In contrast to few-shot classification problems, the batch size on self-supervised tasks is often very large. If we rotate all images individually by four different degrees, we will triple the batch size which will consume valuable memory. Instead, we propose to only sample a single rotation for each image. We conduct an ablation study on the number of rotations used to augment each batch on CIFAR-10 pre-raining, and we show that when the batch size is large, rotating each augmented view in the batch once is enough (see Appendix 7.7.3).

Table 7.6 highlights the effectiveness of treating large rotations as a task augmentation or as an auxiliary loss on pre-trained CIFAR-10 representation in the default setting. Notably, our method enables BYOL to achieve 94.6% accuracy with a backbone trained entirely on unlabeled

Table 7.6: Linear evaluation (Top-1 accuracy (%) with one standard error) on CIFAR-10 with representations learned via different augmentations."Baseline" denotes augmentations used in SimCLR, "+ TA" denotes adding large rotation as task augmentations, "+ $\mathcal{L}_{LR}$" denotes adding large rotation as an auxiliary loss, and "+ Mix" denotes adding image mixing augmentation [5].

Augmentations	SimCLR	R2-D2	ProtoNet	BYOL
Baseline	91.52 ± 0.19	91.46 ± 0.11	91.84 ± 0.19	92.08 ± 0.13
+ TA (ours)	91.46 ± 0.13	91.24 ± 0.06	91.98 ± 0.13	92.03 ± 0.15
+ $\mathcal{L}_{LR}$ (ours)	93.04 ± 0.17	92.74 ± 0.11	93.16 ± 0.13	93.18 ± 0.16
+ Mix	92.98 ± 0.13	93.02 ± 0.11	93.36 ± 0.11	93.83 ± 0.06
+ Mix + $\mathcal{L}_{LR}$ (ours)	$\mathbf{93.88 \pm 0.13}$	$\mathbf{93.70 \pm 0.20}$	$\mathbf{94.12 \pm 0.13}$	$\mathbf{94.57 \pm 0.15}$

data, just as high as models of the same architecture trained in a fully supervised setting (but without our augmentations) [201]. For task augmentation, we apply large rotations with the probability of 0.25 each on top of the baseline augmentations. For the rotation angle predictor loss, we weight the additional loss term with coefficient $\lambda = 1$ for all experiments except for pre-training with R2-D2, where we set $\lambda = 0.01$. In Table 7.6, we see that by using large rotations as a task augmentation, we can avoid the massive accuracy drop observed in Table 7.5. In addition, when we add the angle prediction loss for large rotation as an auxiliary loss, we boost the linear evaluation of all the pre-trained representations by at least 1%.

Table 7.7: Linear evaluation on ImageNet with representations learned by SimCLR and with proposed augmentations.

Model	Top-1 (%)
SimCLR	58.8
SimCLR + $\mathcal{L}_{LR}$ (ours)	**59.6**

We further show that our proposed method can be combined with existing data augmentation methods by adding harder examples with mix-up or cut-mix in Table 7.6. Following Shen et al. [5], we mix one augmented view from each base image with an augmented view from a second image where the indices used for the second augmented view are reverse ordered. In order

to mix images, we randomly apply mix-up or cut-mix with equal probability. Our proposed large rotation prediction loss consistently improves performance on top of the mixing augmentation by $\sim 1\%$ on all pre-training methods, thus leading to more than 2% improvement from the baseline augmentations. On ImageNet, Table 7.7 displays the top-1 linear evaluation accuracy for methods with and without our proposed rotation angle prediction loss. With the proposed auxiliary loss, we achieve 0.8% improvement on the representation trained for 100 epochs. Additional evaluations for transfer learning and semi-supervised learning can be found in Appendix 7.7.4.

7.6 Conclusion

In this work, we discuss the close relationship between contrastive learning and meta-learning. In doing so, we propose a new meta-learning framework for self-supervised learning by converting the contrastive setup into on-the-fly image classification tasks. We show that feature extractors pre-trained via meta-learning achieve comparable results to contrastive learning methods and, in fact, they transfer better to downstream tasks in some cases. In addition, we leverage data augmentation ideas from meta-learning and incorporate them into contrastive learning. We demonstrate that the proposed meta-specific data augmentation consistently improves the performance of popular self-supervised learning algorithms on various datasets.

7.7 Appendix

7.7.1 Pre-training Details

For ImageNet, we follow a similar procedure as Chen et al. [40] for both SimCLR and our proposed meta-learning algorithms, where we use a batch size 256, a starting learning rate of 0.3

(following LR = 0.3 × BatchSize/256), weight decay 10^{-6} and temperature 0.1. For Cifar10, we use a batch size 1024, a starting learning rate of 4. (following LR = 1. × BatchSize/256), weight decay 10^{-6} and temperature 0.5.

7.7.2 Evaluation Details

Linear Evaluation. For ImageNet, we follow a similar procedure as Chen et al. [40], where we use a batch size 4096, a larger learning rate of 1.6 (following LR = 0.1 × BatchSize/256) and longer training of 90 epochs. For CIFAR-10, we train the linear classifier for 200 epochs with batch size 1024, and a linear decay learning rate schedule starts with rate 0.1. We use SGD as the optimizer for linear evaluation on both datasets.

Semi-supervised Learning. Following Chen et al. [40], we fine-tune the pre-trained model with 1% and 10% labeled ImageNet data. Due to the computational limit, instead of using batch size 4096, we use the SGD optimizer with a batch size of 256 and momentum of 0.9. We use a linear decay learning rate schedule starts with rate 0.05. We only use the regular data augmentation for ImageNet training (random crop and resize). We do not use any regularization such as weight decay. For both 1% and 10% of the labeled data, we fine-tune for 60 epochs. The supervised baseline results are imported from Zhai et al. [192], where they trained several thousand models for hyperparameter tuning and using strong data augmentations.

Transfer Learning. We evaluate the performance of models in transfer learning through fine-tuning. We use SSL pre-trained feature extractors with a linear classification head as our model in transfer learning, and we update the entire network during fine-tuning. In this experiment,

we use an SGD optimizer with Nesterov momentum (momentum=0.9). We set the batch size to be 256 and train our models for 20,000 iterations. For each dataset, we follow the training/validation/testing split in Chen et al. [40] and perform the hyperparameter search on the validation set. We search through different learning rates (ranging from 10^{-2} to 10^{-1}) and weight decays (ranging from 10^{-6} to 10^{-3}, and 0), and choose the best setting by comparing the performance on the validation set. Results in Table 7.4 are evaluated on the test split, and our models are trained on the union of training and validation set, using the best hyperparameter settings we have found.

7.7.3 The sample size for Large Rotation Auxiliary Loss

Table 7.8 shows the linear evaluation accuracy for representations learned by different self-supervised learning algorithms with different rotation methods. A full rotation method rotates all the augmented views by 3 times, a half rotation method rotates half of the augmented views by 3 times, and the random rotation only rotates all the augmented views by an angle randomly chosen from $\{0°, 90°, 180°, 270°\}$. With different rotation methods, the number of training examples for rotation angle prediction is different. Among all the experiments, we use the same batch size 1024, and the full rotation will generate $1024 \times 2 \times 4$ training examples, the half rotation will generate $512 \times 2 \times 4$ examples and random rotation will only generate 1024×2 samples, where 2 represents for the number of the random augmentations and 4 is the length of the degree set. From Table 7.8, we can see that even if we have much fewer training samples when applying random rotation, we can still achieve comparable results. As a result, we use random rotation in all the experiments as it makes the training more memory efficient.

Table 7.8: Linear evaluation (top-1 accuracy (%)) on CIFAR-10 with representations learned by various SSL methods with different rotation methods.

Method	SimCLR	R2-D2	PN
Random Rotation	93.0	92.9	93.0
Full Rotation	92.6	92.8	92.9
Half Rotation	92.9	92.6	92.7

7.7.4 More evaluations for large rotation and gradient accumulation

In this section, we provide additional evaluations for the model pre-trained with our proposed large rotation and gradient accumulation methods in Table 7.9 and Table 7.10. We conduct the same transfer learning and semi-supervised learning experiments as in Section 7.4.

Table 7.9: ImageNet Top-1 accuracy (%) of models fine-tuned with few labels. "+ GA" denotes using gradient accumulation and "+ $\mathcal{L}_{LR}$" denotes adding large rotation as an auxiliary loss during pre-training

Method	Backbone	Label fraction	
		1%	10%
Supervised baseline	ResNet-50	25.4	56.4
SimCLR	ResNet-50	31.7	53.3
SimCLR + GA	ResNet-50	32.4	53.6
SimCLR + GA + $\mathcal{L}_{LR}$	ResNet-50	32.0	53.5

Table 7.10: Transfer learning using ImageNet pre-trained weights. We report mean per-class accuracy (%) on the Flowers and Aircraft datasets, mean average precision (mAP) on the VOC2007 classification dataset, and Top-1 accuracy on the remaining datasets. "+ GA" denotes using gradient accumulation and "+ $\mathcal{L}_{LR}$" denotes adding large rotation as an auxiliary loss during pre-training

	Flowers102	DTD	VOC2007	Aircraft	Food101	SUN397	CIFAR-10	CIFAR-100
Baseline	92.0	64.8	67.3	85.9	86.9	53.6	95.9	80.2
SimCLR	92.5	71.4	65.4	84.9	85.9	56.5	95.9	80.3
SimCLR + GA	92.4	72.7	66.0	83.7	86.3	57.4	94.8	79.1
SimCLR + GA + $\mathcal{L}_{LR}$	93.1	73.0	64.4	85.8	86.3	57.0	96.2	82.0

7.7.5 Pre-training for tabular dataset

Outside of computer vision, contrastive learning pre-training has been shown effective in the tabular data domain as well, especially for fine-tuning with limited data [202]. To further show the effectiveness of meta-learning in this setting, we pre-train on tabular data with the R2D2 head and evaluate on semi-supervised tasks given only 50 training samples. We apply our methods to 5 datasets from OpenML [203], and we average over 5 runs with random train-test split on each dataset. Details of the selected datasets can be found in Somepalli et al. [202]. For datasets with binary classification tasks, we evaluate the algorithm with mean AUROC scores, and for datasets with multi-class classification tasks, we evaluate the methods with mean accuracy. Table 7.11 shows that our meta-learning based pre-training method can indeed achieve comparable accuracy to state-of-the-art contrastive learning methods on tabular data.

Table 7.11: Semi-supervised evaluation with 50 labeled training samples on 5 tabular datasets.

Dataset	Pre-train Nethods	Mean Score	Std
adult	-	84.46	1.82
	Contrastive	86.66	1.62
	R2-D2	84.26	4.34
arrhythmia	-	72.36	6.20
	Contrastive	75.94	4.32
	R2-D2	75.64	3.26
telco-customer-churn	-	79.76	2.22
	Contrastive	80.34	1.30
	R2-D2	81.38	1.69
eucalyptus	-	45.06	3.68
	Contrastive	46.8	8.76
	R2-D2	49.17	9.03
volkert	-	40.80	2.57
	Contrastive	39.22	2.07
	R2-D2	40.44	1.76

Bibliography

[1] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2:2, 2018.

[2] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[3] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[4] Jialin Liu, Fei Chao, and Chih-Min Lin. Task augmentation by rotating for meta-learning. *arXiv preprint arXiv:2003.00804*, 2020.

[5] Zhiqiang Shen, Zechun Liu, Zhuang Liu, Marios Savvides, Trevor Darrell, and Eric Xing. Un-mix: Rethinking image mixtures for unsupervised visual representation learning. *arXiv preprint arXiv:2003.05438*, 2020.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[8] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[13] 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 *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[14] 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*, pages 2704–2713, 2018.

[15] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

[16] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[17] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[18] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[19] 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.

[20] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[21] Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *Advances in Neural Information Processing Systems*, 35:12934–12949, 2022.

[22] 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*, pages 10347–10357. PMLR, 2021.

[23] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *Advances in Neural Information Processing Systems*, 34:15908–15919, 2021.

[24] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 558–567, 2021.

[25] Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 357–366, 2021.

[26] Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Jürgen Gall. Adaptive token sampling for efficient vision transformers. In *European Conference on Computer Vision*, pages 396–414. Springer, 2022.

[27] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5270–5279, 2022.

[28] Wei Li, Xing Wang, Xin Xia, Jie Wu, Xuefeng Xiao, Min Zheng, and Shiping Wen. Sepvit: Separable vision transformer. *arXiv preprint arXiv:2203.15380*, 2022.

[29] Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[30] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[31] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 568–578, 2021.

[32] 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*, pages 10012–10022, 2021.

[33] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. In *European conference on computer vision*, pages 459–479. Springer, 2022.

[34] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone

with cross-shaped windows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12124–12134, 2022.

[35] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *Advances in Neural Information Processing Systems*, 34:9355–9366, 2021.

[36] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[37] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

[38] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International conference on learning representations*, 2016.

[39] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[40] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[41] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*, 2021.

[42] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[43] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[44] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[45] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[46] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[47] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[48] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[49] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[50] Renkun Ni, Hong-min Chu, Oscar Castañeda Fernández, Ping-yeh Chiang, Christoph Studer, and Tom Goldstein. Wrapnet: Neural net inference with ultra-low-precision arithmetic. In *International Conference on Learning Representations ICLR 2021*. OpenReview, 2021.

[51] Micah Goldblum, Steven Reich, Liam Fowl, Renkun Ni, Valeriia Cherepanova, and Tom Goldstein. Unraveling meta-learning: Understanding feature representations for few-shot tasks. In *International Conference on Machine Learning*, pages 3607–3616. PMLR, 2020.

[52] Renkun Ni, Micah Goldblum, Amr Sharaf, Kezhi Kong, and Tom Goldstein. Data augmentation for meta-learning. In *International Conference on Machine Learning*, pages 8152–8161. PMLR, 2021.

[53] Renkun Ni, Manli Shu, Hossein Souri, Micah Goldblum, and Tom Goldstein. The close relationship between contrastive learning and meta-learning. In *International conference on learning representations*, 2021.

[54] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[55] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[56] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[57] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[58] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[59] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Jun Zhu, and Hang Su. Learning accurate low-bit deep neural networks with stochastic quantization. *arXiv preprint arXiv:1708.01001*, 2017.

[60] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. Adaptive layerwise quantization for deep neural network compression. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018.

[61] Jungwook Choi, Pierce I-Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Bridging the accuracy gap for 2-bit quantized neural networks (qnn). *arXiv preprint arXiv:1807.06964*, 2018.

[62] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[63] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In *Advances in Neural Information Processing Systems*, pages 5811–5821, 2017.

[64] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[65] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[66] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[67] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4852–4861, 2019.

[68] Fabrizio Pedersoli, George Tzanetakis, and Andrea Tagliasacchi. Espresso: Efficient forward propagation for binary deep neural networks. 2018.

[69] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863*, 2019.

[70] Benoit Jacob, Pete Warden, Miao Wang, David Andersen, Maciek Chociej, Justine Tunney, Mark Matthews, Marie White, Suharsh Sivakumar, Sagi Marcovich, Sarah Knepper, Mourad Gouicem, Richard Winterton, David Mansell, Andreas Gal, Alexey Frunze, and Alexey Frunze. Google gemmlowp, 2016. URL `https://github.com/google/gemmlowp`. https://github.com/google/gemmlowp.

[71] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[72] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

[73] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[74] Daya Khudia, Jianyu Huang, Protonu Basu, Summer Deng, Haixin Liu, Jongsoo Park, and Mikhail Smelyanskiy. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

[75] Barry de Bruin, Zoran Zivkovic, and Henk Corporaal. Quantization of deep neural networks for accumulator-constrained processors. *Microprocessors and Microsystems*, 72: 102872, 2020.

[76] Charbel Sakr, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Ankur Agrawal, Naresh Shanbhag, and Kailash Gopalakrishnan. Accumulation bit-width scaling for ultra-low precision training of deep networks. *arXiv preprint arXiv:1901.06588*, 2019.

[77] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[78] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pages 7675–7684, 2018.

[79] Marcel Simon, Erik Rodner, and Joachim Denzler. Imagenet pre-trained models with batch normalization. *arXiv preprint arXiv:1612.01452*, 2016.

[80] Dong Wang, Xiaodong Wang, and Shaohe Lv. An overview of end-to-end automatic speech recognition. *Symmetry*, 11(8):1018, 2019.

[81] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[82] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5884–5888. IEEE, 2018.

[83] Peter Kairouz et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[84] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[85] Dhruv Guliani, Lillian Zhou, Changwan Ryu, Tien-Ju Yang, Harry Zhang, Yonghui Xiao, Françoise Beaufays, and Giovanni Motta. Enabling on-device training of speech recognition models with federated dropout. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8757–8761. IEEE, 2022.

[86] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[87] Rongmei Lin, Yonghui Xiao, Tien-Ju Yang, Ding Zhao, Li Xiong, Giovanni Motta, and Françoise Beaufays. Federated pruning: Improving neural network efficiency with federated learning. *arXiv preprint arXiv:2209.06359*, 2022.

[88] Tien-Ju Yang, Yonghui Xiao, Giovanni Motta, Françoise Beaufays, Rajiv Mathews, and Mingqing Chen. Online model compression for federated learning with large models, 2022.

[89] Shaojin Ding, Phoenix Meadowlark, Yanzhang He, Lukasz Lew, Shivani Agrawal, and Oleg Rybakov. 4-bit conformer with native quantization aware training for speech recognition. *arXiv preprint arXiv:2203.15952*, 2022.

[90] Kai Zhen, Martin Radfar, Hieu Nguyen, Grant P Strimel, Nathan Susanj, and Athanasios Mouchtaris. Sub-8-bit quantization for on-device speech recognition: a regularization-free approach. In *2022 IEEE Spoken Language Technology Workshop (SLT)*, pages 15–22. IEEE, 2023.

[91] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016. URL `http://arxiv.org/abs/1604.06174`.

[92] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[93] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[94] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2021–2031. PMLR, 2020.

[95] Aleksei Triastcyn, Matthias Reisser, and Christos Louizos. Dp-rec: Private & communication-efficient federated learning. *arXiv preprint arXiv:2111.05454*, 2021.

[96] Kartik Gupta, Marios Fournarakis, Matthias Reisser, Christos Louizos, and Markus Nagel. Quantization robust federated learning for efficient inference on heterogeneous devices. *arXiv preprint arXiv:2206.10844*, 2022.

[97] Sashank Macha, Om Oza, Alex Escott, Francesco Caliva, Robbie Armitano, Santosh Kumar Cheekatmalla, Sree Hari Krishnan Parthasarathi, and Yuzong Liu. Fixed-point quantization aware training for on-device keyword-spotting. In *ICASSP 2023*. IEEE.

[98] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[99] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[100] Dhruv Guliani, Françoise Beaufays, and Giovanni Motta. Training speech recognition models with federated learning: A quality/cost framework. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3080–3084. IEEE, 2021.

[101] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10819–10829, 2022.

[102] Xudong Wang, Li Lyna Zhang, Yang Wang, and Mao Yang. Towards efficient vision transformer inference: a first study of transformers on mobile devices. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, pages 1–7, 2022.

[103] S Mehta and M Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. arxiv 2021. *arXiv preprint arXiv:2110.02178*, 2021.

[104] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12259–12269, 2021.

[105] Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, Vikas Chandra, et al. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training. In *International Conference on Learning Representations*, 2021.

[106] Jiashi Li, Xin Xia, Wei Li, Huixia Li, Xing Wang, Xuefeng Xiao, Rui Wang, Min Zheng, and Xin Pan. Next-vit: Next generation vision transformer for efficient deployment in realistic industrial scenarios. *arXiv preprint arXiv:2207.05501*, 2022.

[107] Chun-Fu Chen, Rameswar Panda, and Quanfu Fan. Regionvit: Regional-to-local attention for vision transformers. *arXiv preprint arXiv:2106.02689*, 2021.

[108] Sachin Mehta and Mohammad Rastegari. Separable self-attention for mobile vision transformers. *arXiv preprint arXiv:2206.02680*, 2022.

[109] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[110] Jianyuan Guo, Kai Han, Han Wu, Yehui Tang, Xinghao Chen, Yunhe Wang, and Chang Xu. Cmt: Convolutional neural networks meet vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12175–12185, 2022.

[111] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16519–16529, 2021.

[112] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021.

[113] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34: 3965–3977, 2021.

[114] Xin Xia, Jiashi Li, Jie Wu, Xing Wang, Mingkai Wang, Xuefeng Xiao, Min Zheng, and Rui Wang. Trt-vit: Tensorrt-oriented vision transformer. *arXiv preprint arXiv:2205.09579*, 2022.

[115] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11936–11945, 2021.

[116] 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*, pages 4510–4520, 2018.

[117] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[118] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[119] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[120] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[121] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.

[122] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[123] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[124] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6023–6032, 2019.

[125] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[126] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[127] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6399–6408, 2019.

[128] Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293, 2017.

[129] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[130] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[131] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[132] Liang Song, Jinlu Liu, and Yongqiang Qin. Fast and generalized adaptation for few-shot learning. *arXiv preprint arXiv:1911.10807*, 2019.

[133] Micah Goldblum, Liam Fowl, and Tom Goldstein. Robust few-shot learning with adversarially queried meta-learners. *arXiv preprint arXiv:1910.00982*, 2019.

[134] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[135] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

[136] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *arXiv preprint arXiv:1909.02729*, 2019.

[137] Kafeng Wang, Xitong Gao, Yiren Zhao, Xingjian Li, Dejing Dou, and Cheng-Zhong Xu. Pay attention to features, transfer learn faster CNNs. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ryxyCeHtPB`.

[138] Nicholas Frosst, Nicolas Papernot, and Geoffrey Hinton. Analyzing and improving representations with the soft nearest neighbor loss. *arXiv preprint arXiv:1902.01889*, 2019.

[139] Gabriel Huang, Hugo Larochelle, and Simon Lacoste-Julien. Centroid networks for few-shot clustering and unsupervised few-shot classification. *CoRR*, abs/1902.08605, 2019. URL `http://arxiv.org/abs/1902.08605`.

[140] Micah Goldblum, Jonas Geiping, Avi Schwarzschild, Michael Moeller, and Tom Goldstein. Truth or backpropaganda? an empirical investigation of deep learning theory. In *International Conference on Learning Representations*, 2019.

[141] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

[142] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468)*, pages 41–48. Ieee, 1999.

[143] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[144] 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.

[145] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[146] W Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K Terry, Furong Huang, and Tom Goldstein. Understanding generalization through visualizations. *arXiv preprint arXiv:1906.03291*, 2019.

[147] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018.

[148] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[149] Seong Min Kye, Hae Beom Lee, Hoirin Kim, and Sung Ju Hwang. Transductive few-shot learning with meta-learned confidence. *arXiv preprint arXiv:2002.12017*, 2020.

[150] Janarthanan Rajendran, Alex Irpan, and Eric Jang. Meta-learning requires meta-augmentation. *arXiv preprint arXiv:2007.05549*, 2020.

[151] Huaxiu Yao, Longkai Huang, Ying Wei, Li Tian, Junzhou Huang, and Zhenhui Li. Don't overlook the support set: Towards improving generalization in meta-learning. *arXiv preprint arXiv:2007.13040*, 2020.

[152] Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. Meta-learning without memorization. *arXiv preprint arXiv:1912.03820*, 2019.

[153] Antreas Antoniou and Amos Storkey. Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation. *arXiv preprint arXiv:1902.09884*, 2019.

[154] Zitian Chen, Yanwei Fu, Kaiyu Chen, and Yu-Gang Jiang. Image block augmentation for one-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3379–3386, 2019.

[155] Zitian Chen, Yanwei Fu, Yu-Xiong Wang, Lin Ma, Wei Liu, and Martial Hebert. Image deformation meta-networks for one-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8680–8689, 2019.

[156] Varun Kumar, Hadrien Glaude, Cyprien de Lichy, and William Campbell. A closer look at feature space data augmentation for few-shot intent classification. *arXiv preprint arXiv:1910.04176*, 2019.

[157] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[158] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[159] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

[160] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7229–7238, 2018.

[161] Jin-Woo Seo, Hong-Gyu Jung, and Seong-Whan Lee. Self-augmentation: Generalizing deep networks to unseen classes for few-shot learning. *arXiv preprint arXiv:2004.00251*, 2020.

[162] Chengyue Gong, Tongzheng Ren, Mao Ye, and Qiang Liu. Maxup: A simple way to improve generalization of neural network training. *arXiv preprint arXiv:2002.09024*, 2020.

[163] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[164] Micah Goldblum, Liam Fowl, and Tom Goldstein. Adversarially robust few-shot learning: A meta-learning approach. *arXiv*, pages arXiv–1910, 2019.

[165] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[166] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer, 2016.

[167] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[168] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[169] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[170] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018.

[171] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[172] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717, 2020.

[173] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 776–794. Springer, 2020.

[174] Sungnyun Kim, Gihun Lee, Sangmin Bae, and Se-Young Yun. Mixco: Mix-up contrastive learning for visual representation. *arXiv preprint arXiv:2010.06300*, 2020.

[175] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*, 2020.

[176] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. *arXiv preprint arXiv:2010.01028*, 2020.

[177] Chunyuan Li, Xiujun Li, Lei Zhang, Baolin Peng, Mingyuan Zhou, and Jianfeng Gao. Self-supervised pre-training with hard examples improves visual representations. *arXiv preprint arXiv:2012.13493*, 2020.

[178] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. *arXiv preprint arXiv:2103.03230*, 2021.

[179] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[180] Jong-Chyi Su, Subhransu Maji, and Bharath Hariharan. When does self-supervision improve few-shot learning? In *European Conference on Computer Vision*, pages 645–666. Springer, 2020.

[181] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[182] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[183] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[184] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[185] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[186] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *arXiv preprint arXiv:2005.10243*, 2020.

[187] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

[188] Siavash Khodadadeh, Ladislau Bölöni, and Mubarak Shah. Unsupervised meta-learning for few-shot image classification. *arXiv preprint arXiv:1811.11819*, 2018.

[189] Han-Jia Ye, Lu Han, and De-Chuan Zhan. Revisiting unsupervised meta-learning: Amplifying or compensating for the characteristics of few-shot tasks. *arXiv preprint arXiv:2011.14663*, 2020.

[190] Carlos Medina, Arnout Devos, and Matthias Grossglauser. Self-supervised prototypical transfer learning for few-shot classification. *arXiv preprint arXiv:2006.11325*, 2020.

[191] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[192] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1476–1485, 2019.

[193] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[194] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[195] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[196] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[197] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

[198] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010. doi: 10.1109/CVPR.2010.5539970.

[199] Zeyu Feng, Chang Xu, and Dacheng Tao. Self-supervised representation learning by rotation feature decoupling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10364–10374, 2019.

[200] Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Pérez, and Matthieu Cord. Boosting few-shot visual learning with self-supervision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8059–8068, 2019.

[201] Liu Kuang, Yang Wei, Yang Peiwen, and Ducau Felipe. Train cifar10 with pytorch, 2017. URL `https://github.com/kuangliu/pytorch-cifar`. https://github.com/kuangliu/pytorch-cifar.

[202] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[203] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.