Ph.D. DISSERTATION

A Study on Deep Model Compression by Refining Gradients and Explicit Regularization

그라디언트 개선 및 명시적 정규화를 통한 심층 모델 압축에 관한 연구

BY

JANGHO KIM

FEBRUARY 2022

Intelligent Systems
Department of Transdisciplinary Studies
Graduate School of Convergence Science and Technology
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

A Study on Deep Model Compression by Refining Gradients and Explicit Regularization

그라디언트 개선 및 명시적 정규화를 통한 심층 모델 압축에 관한 연구

BY

JANGHO KIM

FEBRUARY 2022

Intelligent Systems
Department of Transdisciplinary Studies
Graduate School of Convergence Science and Technology
SEOUL NATIONAL UNIVERSITY

A Study on Deep Model Compression by Refining Gradients and Explicit Regularization

그라디언트 개선 및 명시적 정규화를 통한
심층 모델 압축에 관한 연구

지도교수 곽 노 준
이 논문을 공학박사 학위논문으로 제출함

2022년 2월

서울대학교 대학원

융합과학부 지능형융합시스템전공

김 장 호

김장호의 공학박사 학위 논문을 인준함

2022년 2월

위 원 장:	이 교 구	(인)
부위원장:	곽 노 준	(인)
위 원:	전 동 석	(인)
위 원:	최 상 일	(인)
위 원:	김 은 우	(인)

Abstract

Deep neural network (DNN) has been developed rapidly and has shown remarkable performance in many domains including computer vision, natural language processing and speech processing. The demand for on-device DNN, i.e., deploying DNN on the edge IoT device and smartphone in line with this development of DNN has increased. However, with the growth of DNN, the number of DNN parameters has risen drastically. This makes DNN models hard to be deployed on resource-constraint edge devices. Another challenge is the power consumption of DNN on the edge device because edge devices have a limited battery for the power. To resolve the above issues model compression is very important.

In this dissertation, we propose three novel methods in model compression including knowledge distillation, quantization and pruning. First, we aim to train the student model with additional information of the teacher network, named as knowledge distillation. This framework makes it possible to make the most of a given parameter, which is essential in situations where the device's resources are limited. Unlike previous knowledge distillation frameworks, we focus on distilling the knowledge indirectly by extracting the factor from features because the inherent differences between the teacher and the student, such as the network structure, batch randomness, and initial conditions, can hinder the transfer of appropriate knowledge.

Second, we propose the regularization method for quantization. The quantized model has advantages in power consumption and memory which are essential to the resource-constraint edge device. We non-uniformly rescale the gradient

of the model in the training time to make a weight distribution quantization-friendly. We use position-based scaled gradient (PSG) for rescaling the gradient. Compared with the stochastic gradient descent (SGD), our position-based scaled gradient descent (PSGD) mitigates the performance degradation after quantization because it makes a quantization-friendly weight distribution of the model.

Third, to prune the unimportant overparameterized model dynamic pruning methods have emerged, which try to find diverse sparsity patterns during training by utilizing Straight-Through-Estimator (STE) to approximate gradients of pruned weights. STE can help the pruned weights revive in the process of finding dynamic sparsity patterns. However, using these coarse gradients causes training instability and performance degradation owing to the unreliable gradient signal of the STE approximation. To tackle this issue, we propose refined gradients to update the pruned weights by forming dual forwarding paths. We propose a Dynamic Collective Intelligence Learning (DCIL) to avoid using coarse gradients for pruning.

Lastly, we combine proposed methods as a unified model compression training framework. This method can train a drastically sparse and quantization-friendly model.

keywords: Deep Model Compression, Knowledge Distillation, Quantization, Pruning, Deep Learning

student number: 2017-39082

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience"

– Albert Einstein, *1933*

1.1 Motivation

In recent years, deep neural networks (DNN) have shown remarkable developments in many artificial intelligence-related applications such as computer vision, signal processing, and natural language processing. For example, AlexNet [52] shows remarkable results with DNN in the 2012 ImageNet Challenge using 60 million parameters and many new DNN architectures are proposed every year for improving previous weaknesses in many ways , which generally need more parameters. For this reason, the performance of DNN is increasing rapidly in many tasks. These works rely on with millions or billions of parameters.

According to the rapid developments, the demands of deploying DNN on

INT8	Energy Saving	Area Saving
Add	30×	116×
Multiply	18.5×	27×

Table 1.1: The efficiency of INT8 operation compared to FP32 operation.

resource-restricted devices such as IoT devices and smartphones also increase. A vast amount of computational storage and cost makes them difficult to use in embedded systems with limited resources. For example, the number of parameters for the ResNet-50 [30] is 25.56M with 96MB memory for storage and AlexNet needs 1.4GOPS to inference a single 224 × 224 image. Given the size of the equipment we use, tremendous GPU computations are not generally available in real-world applications.

Another challenge in deploying DNN on the edge device is the power consumption. In general, resource-restricted devices have a limited battery, which is rapidly drained with the high computational operation. This power consumption issue is related to the bandwidth of operations. For instance, INT8 operations consume 4× less operation bandwidth than FP32 operations. This reduced-bandwidth operation with the quantization is faster and energy-efficient compared to the floating-point operation, shown in Table 1.1 [13].

To address above issues, many researchers studied DNN structures to make DNNs smaller and more efficient to be applicable for embedded systems. These studies can be roughly classified into three tasks: 1) **knowledge distillation** is to transfer a teacher model's information to a student network[1]. 2) **quantization** compresses the model by reducing the bandwidth used to represent the weight

[1]Knowledge distillation is often referred to as knowledge transfer (KT), to avoid meaning specific algorithm KD [35] than the domain.

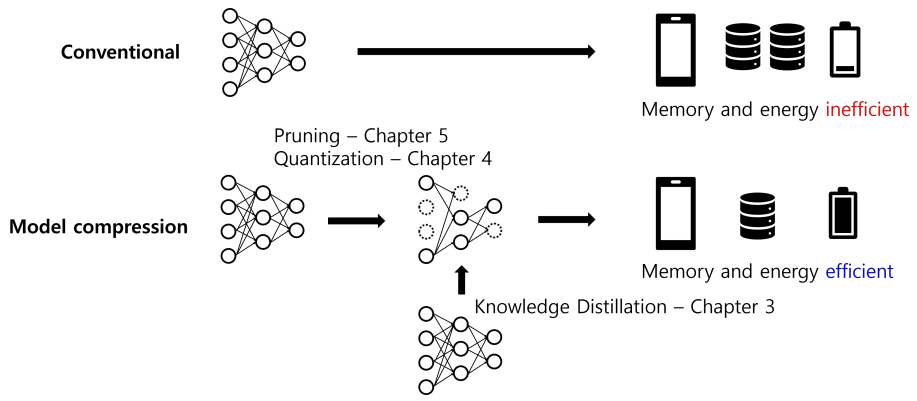


Figure 1.1: Outline design of dissertation. We propose model compression methods in three tasks including knowledge distillation, quantization and pruning. Compared to conventional training, model compression methods compress the model to make the model efficient in memory and the energy consumption.

parameters. 3) **pruning** is a way to reduce network complexity by pruning the redundant and non-informative weights. More details of the three tasks are stated in the next section.

However, KD methods do not compress the model size or reduce computational operations. They give efficiency in the model performance with the same given parameters so other compression methods are needed for compressing the model. Quantization reduces lots of computational costs but it is also relatively inefficient compared to the pruning method in terms of saving memory efficiency. Pruning only considers eliminating the model weights. For these reasons, we have a study on deep model compression in three ways containing knowledge distillation, quantization and pruning. Finally, we combine three methods in a single framework.

Fig. 1.1 illustrates the outline design of this dissertation. In this dissertation, firstly, we propose three efficient methods corresponding to each task includ-

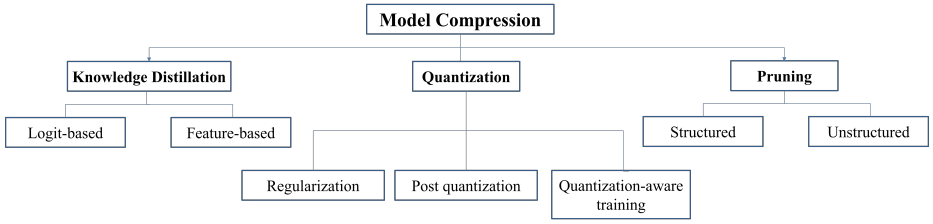


Figure 1.2: Three tasks and their subcategories in model compression.

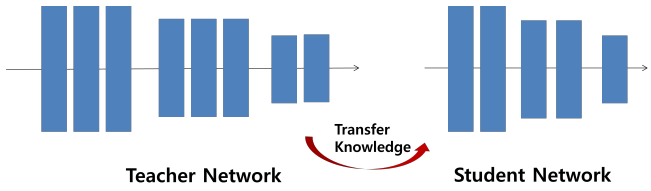


Figure 1.3: Knowledge distillation is the training framework using the information of the teacher model

ing knowledge distillation, network quantization and pruning for deep model compression to make the model efficient in terms of the storage and the bandwidth. Then, we combine the proposed pruning and quantization methods with knowledge distillation in the same training phase.

1.2 Tasks

In this section, we explain three model compression tasks and their subcategories. Fig. 1.2 shows three tasks related to the model compression and their subcategories. Note that subcategories can vary according to the many perspective so they are not limited to introduced subcategories in this dissertation.

Knowledge distillation (KD) method is to transfer the information of the teacher model to the student model (See Fig. 1.3), being also considered as a regularization method because KD method regularizes the student model with

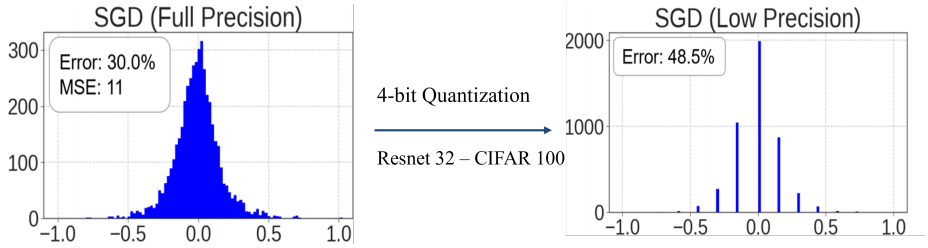


Figure 1.4: The quantization example where ResNet-32 trained on CIFAR-100 is quantized at 4bit.

many ways in the training. teacher model may have the same structure as the student model or may have a deeper structure than the student model.

KD is divided into two subcategories according to the type of information transferred, Logit-based KD and feature-based KD. Logit-based KD focus on transferring the output of the last fully connected layer formed as a probability. In general, Logit-based KD uses Kullback–Leibler (KL) divergence to make the distribution of the student model similar to that of the teacher model.

Feature-based KD utilizes the feature maps as the information for the knowledge transfer. Feature maps contain spatial information compared to a naive distribution generated from logits. Many distance metrics and pre-processing are used for the feature-based KD.

Quantization method quantizes floating-point weights of the model onto regular grids to reduce computational cost. The original weights are approximated by a set of integers and a scaling factor [48]. This approximation has advantages in using faster and more efficient integer operations in the deep learning inference, at the expense of lower representing power.

However, as quantization loses information because of lower representing power, the performance of the network degrades according to the representing

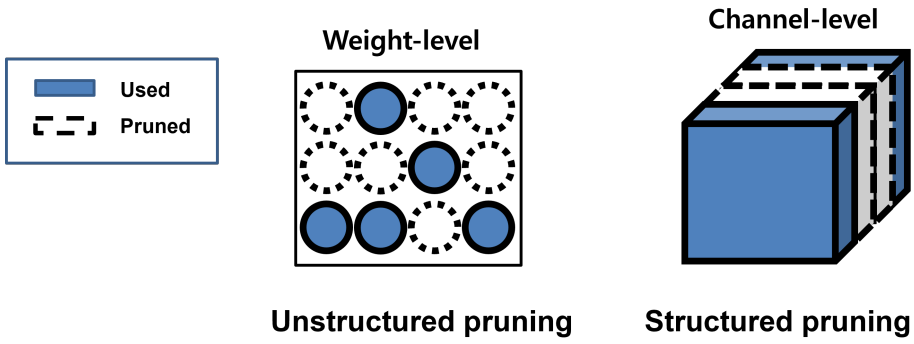


Figure 1.5: Unstructured pruning prunes parameters of the model in a weight-level. However, structured pruning prunes the whole filter or channel containing a set of parameters.

power. For example, Fig. 1.4 shows the naive symmetric quantization process. Two histograms depict the weights of a ResNet-32 trained with CIFAR-100 dataset. After quantizing the weight into 4-bit, the bell-shaped dense distribution becomes discrete values having 2^4 bins. Because of the discrepancy of distributions between full-precision and low precision, performance degradation is inevitable.

To resolve this problem, the regularization method and post quantization have been studied. When the model is trained, the regularization method introduces the explicit regularization term in the loss function or implicitly regularizes the model at the gradient level, to make the model quantization-friendly. Post quantization methods handle the pre-trained model to avoid drastic degradation. These methods do not need the training phase because they only use the small calibration dataset to rescale the weight and bias or eliminate outliers which bring severe effects in the performance degradation. The regularization method and the post quantization method start from the full-precision model or training the

full-precision model with the regularization term. On the contrary, Quantization-aware training (QAT) only focuses on the performance of the low-precision model. In the model training, QAT calculates the loss of an objective function from quantized weights by using a straight-through estimator (STE) [4] because the quantization process contains a non-differentiable part.

Pruning method prunes the unimportant weights or filters (channels) in the model according to various criteria. It is divided into two subcategories according to the unit used in the pruning. Fig. 1.5 shows two subcategories which are unstructured pruning and structured pruning. Unstructured pruning prunes the model at the weight-level unit. On the other hand, Structured pruning prunes the model at the filter-level unit containing a set of weights. In general, unstructured pruning outperforms structured pruning with a large margin. However, unstructured pruning is not a hardware-friendly method because it makes sparse weight matrices requiring dedicated hardware and libraries for speed up and the compression.

1.3 Contributions and Outline

For considering key challenges in the deploying DNN on the resource constraint device and three tasks in the model compression domain, in this dissertation, we propose three novel methods according to each task. Fig. 1.6 shows the outline of this dissertation.

In Chapter 2, prior works related to three tasks of model compression and their subcategories are reviewed.

In Chapter 3, we propose a novel knowledge transfer method which uses convolutional operations to paraphrase teacher's knowledge and to translate it

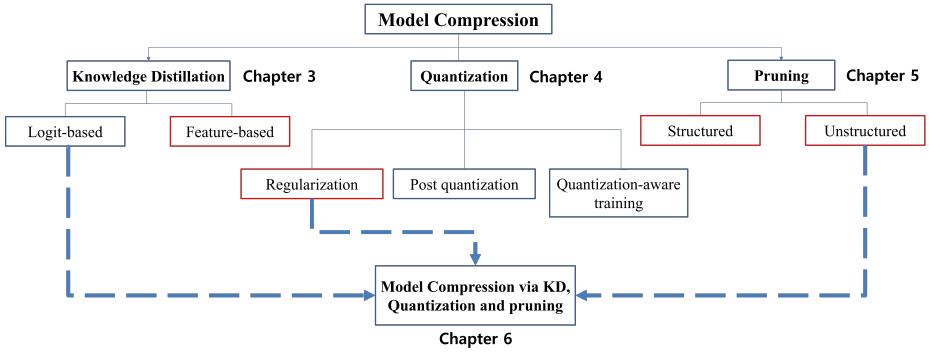


Figure 1.6: We propose three model compression methods. We introduce a indirect feature-based distillation in Chapter 3 and propose the regularization method for the quantization in Chaper 4. In Chapter 5, we propose a novel dynamic pruning method handling structured and unstructured pruning. Finally, we combine proposed methods in Chapter 6.

for the student where this method uses the feature map distillation categorized in feature-based KD [44]. This is done by two convolutional modules, which are called a *paraphraser* and a *translator*. The paraphraser is trained in an unsupervised manner to extract the *teacher factors* which are defined as paraphrased information of the teacher network. The translator located at the student network extracts the *student factors* and helps to translate the teacher factors by mimicking them. We observed that our student network trained with the proposed factor transfer method outperforms the ones trained with conventional knowledge transfer methods.

In Chapter 4, we propose the position-based scaled gradient (PSG) that scales the gradient depending on the position of a weight vector to make it more compression-friendly and this method is categorized in the regularization method of quantization [46]. First, we theoretically show that applying PSG to

the standard gradient descent (GD), which is called PSGD, is equivalent to the GD in the warped weight space, a space made by warping the original weight space via an appropriately designed invertible function. Second, we empirically show that PSG acting as a regularizer to the weight vectors is favorable for model compression domains such as quantization, pruning, and knowledge distillation. PSG reduces the gap between the weight distributions of a full-precision model and its compressed counterpart. This enables the versatile deployment of a model either as an uncompressed mode or as a compressed mode depending on the availability of resources. The experimental results on CIFAR-10/100 and ImageNet datasets show the effectiveness of the proposed PSG in model compression even for extremely low bits.

In Chapter 5, we introduce refined gradients to update the pruned weights by forming dual forwarding paths from two sets (pruned and unpruned) of weights. We propose a novel Dynamic Collective Intelligence Learning (DCIL) which makes use of the learning synergy between the collective intelligence of both weight sets [45]. We verify the usefulness of the refined gradients by showing enhancements in the training stability and the model performance on the CIFAR and ImageNet datasets. DCIL outperforms various previously proposed pruning schemes including other dynamic pruning methods with enhanced stability during training.

In Chapter 6, to leverage the advantages of three tasks of model compression including knowledge distillation, quantization and pruning, we combine proposed PSGD and DCIL with logit-based KD and propose a unified training framework named KQP.

In Chapter 7, we provide the summary and the future directions of this research toward on-device deep learning.

To sum up, we refine the gradient of the model by scaling gradients according to the position of weights for quantization and recalculating the approximated gradients with dual forwarding paths for pruning. Also, we introduce explicit regularization named as factor transfer for knowledge distillation.

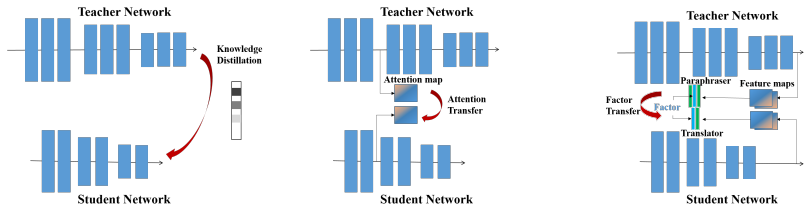
Chapter 2

Related work

In this chapter, we provide related works for each proposed method according to each task of three tasks in model compression. Firstly, we compare existing knowledge distillation methods with proposed factor transfer in section 2.1. Then, we introduce quantization methods and the sparse training and also compare proposed PSGD method in section 2.2. Finally, in section 2.3, pruning methods are described and we explain advantages of proposed DCIL.

2.1 Knowledge Distillation

'knowledge transfer' is a method of training a student network with a stronger teacher network. Knowledge distillation (KD) [35] is the early work of knowledge transfer for deep neural networks. The main idea of KD is to shift knowledge from a teacher network to a student network by leaning the class distribution via softened softmax. The student network can capture not only the information provided by the true labels, but also the information from the teacher. Yim *et al*. [84] defined the flow of solution procedure (FSP) matrix calculated by



(a) Knowledge Distillation (b) Attention Transfer (c) Proposed

Figure 2.1: The structure of (a) KD [35], (b) AT [86] and (c) the proposed method FT. Unlike KD and AT, our method does not directly compare the softened distribution (KD) or the attention map (AT) which is defined as the sum of feature maps of the teacher and the student networks. Instead, we extract factors from both the teacher and the student, whose difference is tried to be minimized.

Gram matrix of feature maps from two layers in order to transfer knowledge. In FitNet [72], they designed the student network to be thinner and deeper than the teacher network, and provided hints from the teacher network for improving performance of the student network by learning intermediate representations of the teacher network. FitNet attempts to mimic the intermediate activation map directly from the teacher network. However, it can be problematic since there are significant capacity differences between the teacher and the student. Attention transfer (AT) [86], in contrast to FitNet, trains a less deep student network such that it mimics the attention maps of the teacher network which are summations of the activation maps along the channel dimension. Therefore, an attention map for a layer is of its the spatial dimensions. Figure 2.1 visually shows the difference of KD [35], AT [86] and the proposed method, factor transfer (FT). Unlike other methods, our method does not directly compare the teacher and student networks' softend distribution, or attention maps.

As shown in Figure 3.1, our paraphraser is similar to the convolutional autoencoder [62] in that it is trained in an unsupervised manner using the reconstruction loss and convolution layers. Hinton *et al.*[36] proved that autoencoders produce compact representations of images that contain enough information for reconstructing the original images. In [53], a stacked autoencoder on the MNIST dataset achieved great results with a greedy layer-wise approach. Many studies show that autoencoder models can learn meaningful, abstract features and thus achieve better classification results in high-dimensional data, such as images [68, 74]. The architecture of our paraphraser is different from convolutional autoencoders in that convolution layers do not downsample the spatial dimension of an input since the paraphraser uses sufficiently downsampled feature maps of a teacher network as the input.

2.2 Quantization

QAT methods have shown increasingly strong performance in the low-precision domain even to 2,3 bit-width [19, 23, 5, 41]. Post-training quantization, on the other hand, aims to quantize weights and activation without additional training or using the training data. Majority of the works in recent literature starts from a pre-trained network trained by standard training scheme [90, 66, 3]. Many works on channel-wise quantization methods, which require storing quantization parameters per channel, have shown notable improvement in performance even at 4-bit [3, 11]. However, layer-wise quantization methods, which are more hardware-friendly as they store quantization parameters per layer (as opposed to per channel), still suffers at lower bit-widths [66, 48, 90]. [66] achieves near full-precision accuracy at 8-bit by bias correction and range equalization of

channels, while [90] splits channels with outliers to reduce the clipping error. However, both suffer from severe accuracy degradation under 6-bit. Our method improves on but is not limited to the uniform layer-wise quantization. Concurrent to ours, [67] and [65] propose to directly minimize the quantization error using a calibration dataset to achieve higher performance at under 6-bit. We show using PSGD pretrained model outperforms using SGD pretrained model in Section 4.4.

Meanwhile, another line of work in quantization has focused on quantization robustness by regularizing the weight distribution from the initial training phase. [58] focuses on minimizing the Lipshitz constant to regularize the gradients for robustness against adversarial attacks. Similarly, [2] proposes a new regularization term on the norm of the gradients for quantization robustness across different bit widths. This enables "on-the-fly" quantization to various bit widths. Our method does not have an explicit regularization term but scales the gradients to implicitly regularize the weights in the full-precision domain to make them quantization-friendly. Additionally, we do not introduce significant training overhead because gradient norm regularization is not necessary, while [2] necessitates double-backpropagation which increases the training complexity. Some other related works in quantization aims to quantize the gradient vectors for efficient training [15], propose more representative encoding formats [78], or learn the optimal mixed precision bit-width [80].

2.2.1 Sparse training

Another relevant line of research in model compression is pruning, in which unimportant units such as weights, filters, or entire blocks are pruned [39, 56]. Recent works have focused on pruning methods that include the pruning process in the training phase [71, 91, 61, 55]. Among them, substantial amount of works

utilize sparsity-inducing regularization. [61] proposes training with L0 norm regularizer on individual weights to train a sparse network, using the expected L0 objective to relax the otherwise indifferentiable regularization term. Meanwhile, other works focus on using saliency criterion. [55] utilizes gradients of masks as a proxy for importance to prune networks at a single-shot. Similar to [55] and [61], our method does not need a heuristic pruning schedule during training nor additional fine-tuning after pruning. In our method, pruning is formulated as a subclass of quantization because PSG can be used for sparse training by setting the target value as zero instead of the quantized grid points.

2.3 Pruning

Earlier works in model pruning have focused on pruning unimportant parts of a fully trained model. [54], and [29] have used second derivative information to identify unimportant neurons. [28] have established a general paradigm of iterative training-and-pruning to recover the degradation of accuracy due to pruning. Since then, many methods have been proposed in both unstructured pruning [71, 83] and structured pruning [34, 56]. [27] proposed a method to undo the pruning of important units by "splicing" during the pruning process as an alternative to greedy pruning. This avoids permanent pruning of important units as units may revive in the splicing process.

Motivated by reducing the computational costs of iterative pruning methods, some recent methods have focused on finding a sparse network while training. Many methods dynamically prune and regrow pruned weights while training instead of using a fixed mask as done in earlier work of [27]. Sparse Evolutionary Training [63] uses simple heuristics to prune unimportant weights and

subsequently regrow weights. Dynamic Sparse Reparameterization [64] adjusts sparsity patterns while training and automatically reallocates a sparsity level across layers. Sparse Momentum [17] uses gradient momentum to determine unimportant weights and layers and redistributes sparsity across layers using the momentum. Dynamic Feedback with Pruning [59] generalizes the simple scheme of [27] to prune while training and achieves state-of-the-art accuracy in unstructured pruning. While effective, the training curve is often unstable due to the coarse gradients, which makes model selection difficult for deployment. We propose a dynamic pruning method with superior final accuracy and is particularly stable when training by improving upon DPF via refining the gradients with two sub-networks.

Chapter 3

Factor Transfer (FT) for Knowledge Distillation

3.1 Introduction

In this work, we focus on the knowledge transfer. Previous studies such as *attention transfer* (AT) [86] and *knowledge distillation* (KD) [35] have achieved meaningful results in the field of knowledge transfer, where their loss function can be collectively summarized as the difference between the attention maps or softened distributions of the teacher and the student networks. These methods directly transfer the teacher network's softened distribution [35] or its attention map [86] to the student network, inducing the student to mimic the teacher.

While these methods provide fairly good performance improvements, directly transferring the teacher's outputs overlooks the inherent differences between the teacher network and the student network, such as the network structure, the number of channels, and initial conditions. Therefore, we need to re-interpret the output of the teacher network to resolve these differences. For example, from the perspective of a teacher and a student, we came up with a question that simply providing the teacher's knowledge directly without any explanation

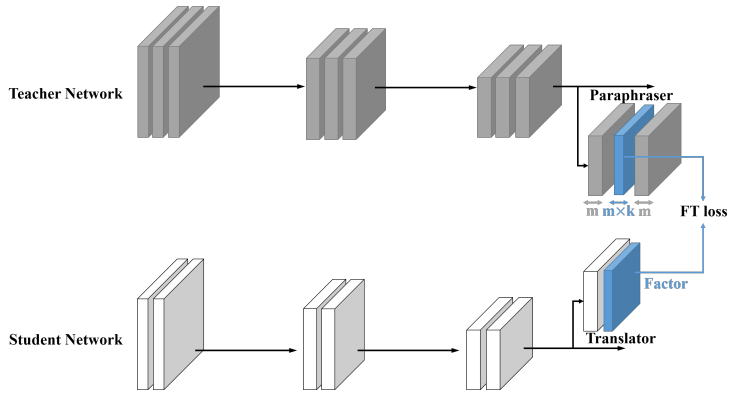


Figure 3.1: Overview of the factor transfer. In the teacher network, feature maps are transformed to the 'teacher factors' by a paraphraser. The number of feature maps of a teacher network (m) are resized to the number of feature maps of teacher factors ($m \times k$) by a paraphrase rate k. The feature maps of the student network are also transformed to the 'student factors' with the same dimension as that of the teacher factor using a translator. The factor transfer (FT) loss is used to minimize the difference between the teacher and the student factors in the training of the translator that generates student factors. Factors are drawn in blue. Note that before the FT, the paraphraser is already trained unsupervisedly by a reconstruction loss.

can be somewhat insufficient for teaching the student. In other words, when teaching a child, the teacher should not use his/her own term because the child cannot understand it. On the other hand, if the teacher translates his/her terms into simpler ones, the child will much more easily understand.

In this respect, we sought ways for the teacher network to deliver more understandable information to the student network, so that the student comprehends that information more easily. To address this problem, we propose a novel knowledge transferring method that leads both the student and teacher networks to

make transportable features, which we call 'factors' in this work. Contrary to the conventional methods, our method is not simply to compare the output values of the network directly, but to train neural networks that can extract good factors and to match these factors. The neural network that extracts factors from a teacher network is called a *paraphraser*, while the one that extracts factors from a student network is called a *translator*. We trained the paraphraser in an unsupervised way, expecting it to extract knowledges different from what can be obtained with supervised loss term. At the student side, we trained the student network with the translator to assimilate the factors extracted from the paraphraser. The overview of our proposed method is provided in Figure 3.1. With various experiments, we succeeded in training the student network to perform better than the ones with the same architecture trained by the conventional knowledge transfer methods.

Our contributions can be summarized as follows:

• We propose a usage of a paraphraser as a means of extracting meaningful features (factors) in an unsupervised manner.

• We propose a convolutional translator in the student side that learns the factors of the teacher network.

• We experimentally show that our approach effectively enhances the performance of the student network.

3.2 Proposed method

It is said that if one fully understands a thing, he/she should be able to explain it by himself/herself. Correspondingly, if the student network can be trained to replicate the extracted information, this implies that the student network is well informed of that knowledge. In this section, we define the output of paraphraser's

middle layer, as *'teacher factors'* of the teacher network, and for the student network, we use the translator made up of several convolution layers to generate '*student factors'* which are trained to replicate the *'teacher factors'* as shown in Figure 3.1. With these modules, our knowledge transfer process consists of the following two main steps: 1) In the first step, the paraphraser is trained by a reconstruction loss. Then, *teacher factors* are extracted from the teacher network by a paraphraser. 2) In the second step, these *teacher factors* are transferred to the *student factors* such that the student network learns from them.

3.2.1 Teacher Factor Extraction with Paraphraser

ResNet architectures [30] have stacked residual blocks and in [86] they call each stack of residual blocks as a 'group'. In this work, we will also denote each stacked convolutional layers as a 'group'. Yosinski *et al*.[85] verified lower layer features are more general and higher layer features have a greater specificity. Since the teacher network and the student network are focusing on the same task, we extracted factors from the feature maps of the last group as clearly can be seen in Figure 3.1 because the last layer of a trained network must contain enough information for the task.

In order to extract the factor from the teacher network, we train the paraphraser in an unsupervised way by assigning the reconstruction loss between the input feature maps x and the output feature maps $P(x)$ of the paraphraser. The unsupervised training act on the factor to be more meaningful, extracting different kind of knowledge from what can be obtained with supervised cross-entropy loss function. This approach can also be found in EBGAN [89], which uses an autoencoder as discriminator to give the generator different kind of knowledge from binary output.

The paraphraser uses several convolution layers to produce the teacher factor F_T which is further processed by a number of transposed convolution layers in the training phase. Most of the convolutional autoencoders are designed to downsample the spatial dimension in order to increase the receptive field. On the contrary, the paraphraser maintains the spatial dimension while adjusting the number of factor channels because it uses the feature maps of the last group which has a sufficiently reduced spatial dimension. If the teacher network produces m feature maps, we resize the number of factor channels as $m \times k$. We refer to hyperparameter k as a paraphrase rate.

To extract the teacher factors, an adequately trained paraphraser is needed. The reconstruction loss function used for training the paraphraser is quite simple as

$$\mathcal{L}_{rec} = \|x - P(x)\|^2, \tag{3.1}$$

where the paraphraser network $P(\cdot)$ takes x as an input. After training the paraphraser, it can extract the task specific features (teacher factors) as can be seen in the supplementary material.

3.2.2 Factor Transfer with Translator

Once the teacher network has extracted the factors which are the paraphrased teacher's knowledge, the student network should be able to absorb and digest them on its own way. In this work, we name this procedure as 'Factor Transfer'. As depicted in Figure 3.1, while training the student network, we inserted the translator right after the last group of student convolutional layers.

The translator is trained jointly with the student network so that the student network can learn the paraphrased information from the teacher network. Here,

the translator plays a role of a buffer that relieves the student network from the burden of directly learning the output of the teacher network by rephrasing the feature map of the student network.

The student network is trained with the translator using the sum of two loss terms, *i.e*. the classification loss and the factor transfer loss:

$$\mathcal{L}_{student} = \mathcal{L}_{cls} + \beta \mathcal{L}_{FT}, \tag{3.2}$$

$$\mathcal{L}_{cls} = \mathcal{C}(S(I_x), y), \tag{3.3}$$

$$\mathcal{L}_{FT} = \| \frac{F_T}{\|F_T\|_2} - \frac{F_S}{\|F_S\|_2} \|_p. \tag{3.4}$$

With (3.4), the student's translator is trained to output the student factors that mimic the teacher factors. Here, F_T and F_S denote the teacher and the student factors, respectively. We set the dimension of F_S to be the same as that of F_T. We also apply an l_2 normalization on the factors as [86]. In this work, the performances using l_1 loss ($p = 1$) is reported, but the performance difference between l_1 ($p = 1$) and l_2 ($p = 2$) losses is minor (See the supplementary material), so we consistently used l_1 loss for all experiments.

In addition to the factor transfer loss (3.4), the conventional classification loss (3.3) is also used to train student network as in (3.2). Here, β is a weight parameter and $C(S(I_x), y)$ denotes the cross entropy between ground-truth label y and the softmax output $S(I_x)$ of the student network for an input image I_x, a commonly used term for classification tasks.

The translator takes the output features of the student network, and with (3.2), it sends the gradient back to the student networks, which lets the student network absorb and digest the teacher's knowledge in its own way. Note that unlike the training of the teacher paraphraser, the student network and its translator are

trained simultaneously in an end-to-end manner.

3.3 Experiments

In this section, we evaluate the proposed FT method on several datasets. First, we verify the effectiveness of FT through the experiments with CIFAR-10 [50] and CIFAR-100 [51] datasets, both of which are the basic image classification datasets, because many works that tried to solve the knowledge transfer problem used CIFAR in their base experiments [72, 86]. Then, we evaluate our method on ImageNet LSVRC 2015 [73] dataset. Finally, we applied our method to object detection with PASCAL VOC 2007 [20] dataset.

To verify our method, we compare the proposed FT with several knowledge transfer methods such as KD [35] and AT [86]. There are several important hyperparameters that need to be consistent. For KD, we fix the temperature for softened softmax to 4 as in [35], and for β of AT, we set it to 10^3 following [86]. In the whole experiments, AT used multiple group losses. Alike AT, β of FT is set to 10^3 in ImageNet and PASCAL VOC 2007. However, we set it to 5×10^2 in CIFAR-10 and CIFAR-100 because a large β hinders the convergence.

We conduct experiments for different k values from 0.5 to 4. To show the effectiveness of the proposed paraphraser architecture, we also used two convolutional autoencoders as paraphrasers because the autoencoder is well known for extracting good features which contain compressed information for reconstruction. One is an undercomplete convolutional autoencoder (CAE), the other is an overcomplete regularized autoencoder (RAE) which imposes l_1 penalty on factors to learn the size of factors needed by itself [1]. Details of these autoencoders and overall implementations of experiments are explained in the supplementary

material.

In some experiments, we also tested KD in combination with AT or FT because KD transfers output knowledge while AT and FT delivers knowledge from intermediate blocks and these two different methods can be combined into one (KD+AT or KD+FT).

3.3.1 CIFAR-10

The CIFAR-10 dataset consists of 50K training images and 10K testing images with 10 classes. We conducted several experiments on CIFAR-10 with various network architectures, including ResNet [30], Wide ResNet (WRN) [87] and VGG [75]. Then, we made four conditions to test various situations. First, we used ResNet-20 and ResNet-56 which are used in CIFAR-10 experiments of [30]. This condition is for the case where the teacher and the student networks have same width (number of channels) and different depths (number of blocks). Secondly, we experimented with different types of residual networks using ResNet-20 and WRN-40-1. Thirdly, we intended to see the effect of the absence of shortcut connections that exist in Resblock on knowledge transfer by using VGG13 and WRN-46-4. Lastly, we used WRN-16-1 and WRN-16-2 to test the applicability of knowledge transfer methods for the architectures with the same depth but different widths.

In the first experiment, we wanted to show that our algorithm is applicable to various networks. Result of FT and other knowledge transfer algorithms can be found in Table 3.1. In the table, 'Student' column provides the performance of student network trained from scratch. The 'Teacher' column provides the performance of the pretrained teacher network. The numbers in the parentheses are the sizes of network parameters in Millions. The performances of AT and KD

Student	Teacher	Student	AT	KD	FT	AT+KD	FT+KD	Teacher
ResNet-20 (0.27M)	ResNet-56 (0.85M)	7.78	7.13	7.19	**6.85**	6.89	7.04	6.39
ResNet-20 (0.27M)	WRN-40-1 (0.56M)	7.78	7.34	7.09	**6.85**	7.00	6.95	6.84
VGG-13 (9.4M)	WRN-46-4 (10M)	5.99	5.54	5.71	4.84	5.30	**4.65**	4.44
WRN-16-1 (0.17M)	WRN-16-2 (0.69M)	8.62	8.10	7.64	7.64	**7.52**	7.59	6.27

Student	Teacher	$k = 0.5$	$k = 0.75$	$k = 1$	$k = 2$	$k = 4$	CAE	RAE
ResNet-20 (0.27M)	ResNet-56 (0.85M)	**6.85**	6.92	6.89	6.87	7.08	7.07	7.24
ResNet-20 (0.27M)	WRN-40-1 (0.56M)	7.16	7.05	7.04	**6.85**	7.05	7.26	7.33
VGG-13 (9.4M)	WRN-46-4 (10M)	**4.84**	5.09	5.04	5.01	4.98	5.85	5.53
WRN-16-1 (0.17M)	WRN-16-2 (0.69M)	**7.64**	7.83	7.74	7.87	7.95	8.48	8.00

Table 3.1: Mean classification error (%) on CIFAR-10 dataset (5 runs). All the numbers are the results of our implementation. AT and KD are implemented according to [86].

Student	Teacher	Student	AT	F-ActT	KD	AT+KD	Teacher	FT ($k = 0.5$)	Teacher
WRN-16-1 (0.17M)	WRN-40-1 (0.56M)	8.77	8.25	8.62	8.39	8.01	6.58	8.12	6.55
WRN-16-2 (0.69M)	WRN-40-2 (2.2M)	6.31	5.85	6.24	6.08	5.71	5.23	5.51	5.09

Table 3.2: Median classification error (%) on CIFAR-10 dataset (5 runs). The first 6 columns are from Table 1 of [86], while the last two columns are from our implementation.

are better than those of 'Student' trained from scratch and the two show better or worse performances than the other depending on the type of network used. For FT, we chose the best performance among the different k values shown in the bottom rows in the table. The proposed FT shows better performances than AT and KD consistently, regardless of the type of network used.

In the cases of hybrid knowledge transfer methods such as AT+KD and FT+KD, we could get interesting result that AT and KD make some sort of synergy, because for all the cases, AT+KD performed better than standalone

AT or KD. It sometimes performed even better than FT, but FT model trained together with KD loses its power in some cases.

As stated before in section 3.2.1, to check if having a paraphraser per group in FT is beneficial, we trained a ResNet-20 as student network with paraphrasers and translators combined in group1, group2 and group3, using the ResNet-56 as teacher network with $k = 0.75$. The classification error was 7.01%, which is 0.06% higher than that from the single FT loss for the last group. This indicates that the combined FT loss does not improve the performance thus we have used the single FT loss throughout the work. In terms of paraphrasing the information of the teacher network, the paraphraser which maintains the spatial dimension outperformed autoencoders based methods which use CAE or RAE.

As a second experiment, we compared FT with transferring FitNets-style hints which use full activation maps as in [86]. Table 3.2 shows the results which verifiy that using the paraphrased information is more beneficial than directly using the full activation maps (full feature maps). In the table, FT gives better accuracy improvement than full-activation transfer (F-ActT). Note that we trained a teacher network from scratch for factor transfer (the last column) with the same experimental environment of [86] because there is no pretrained model of the teacher networks.

3.3.2 CIFAR-100

For further analysis, we wanted to apply our algorithm to more difficult tasks to prove generality of the proposed FT by adopting CIFAR-100 dataset. CIFAR-100 dataset contains the same number of images as CIFAR-10 dataset, 50K (train) and 10K (test), but has 100 classes, containing only 500 images per classes. Since the training dataset is more complicated, we thought the number of blocks

Student	Teacher	Student	AT	KD	FT	AT+KD	FT+KD	Teacher
ResNet-56 (0.85M)	ResNet-110 (1.73M)	28.04	27.28	27.96	**25.62**	28.01	26.93	26.91
ResNet-20 (0.27M)	ResNet-110 (1.73M)	31.24	31.04	33.14	**29.08**	34.78	32.19	26.91

Student	Teacher	$k = 0.5$	$k = 0.75$	$k = 1$	$k = 2$	$k = 4$	CAE	RAE
ResNet-56 (0.85M)	ResNet-110 (1.73M)	**25.62**	25.78	25.85	25.63	25.87	26.41	26.29
ResNet-20 (0.27M)	ResNet-110 (1.73M)	29.20	29.25	29.28	29.19	**29.08**	29.84	30.11

Table 3.3: Mean classification error (%) on CIFAR-100 dataset (5 runs). All the numbers are from our implementation.

(depth) in the network has much more impact on the classification performance because deeper and stronger networks will better learn the boundaries between classes. Thus, the experiments on CIFAR-100 were designed to observe the changes depending on the depths of networks. The teacher network was fixed as ResNet-110, and the two networks ResNet-20 and ResNet-56, that have the same width (number of channels) but different depth (number of blocks) with the teacher, were used as student networks. As can be seen in Table 3.3, we got an impressive result that the student network ResNet-56 trained with FT even outperforms the teacher network. The student ResNet-20 did not work that well but it also outperformed other knowledge transfer methods.

Additionally, in line with the experimental result in [86], we also got consistent result that KD suffers from the gap of depths between the teacher and the student, and the accuracy is even worse compared to the student network in the case of training ResNet-20. For this dataset, the hybrid methods (AT+KD and FT+KD) was worse than the standalone AT or FT. This also indicates that KD is not suitable for a situation where the depth difference between the teacher and the student networks is large.

Paraphraser	Translator	CIFAR-10	CIFAR-100	Number of layers in Paraphraser	CIFAR-10	CIFAR-100
Yes	No	6.18	27.61	1 Layer [0.07M]	6.09	27.07
No	Yes	6.12	27.39	2 Layers [0.22M]	5.99	27.03
Yes	Yes	5.71	26.91	3 Layers [0.26M]	5.71	26.91
Student (WRN-40-1[0.6M])		7.02	28.81	Teacher (WRN-40-2[2.2M])	4.96	24.10

Table 3.4: Left: Ablation study with and without the paraphraser ($k = 0.5$) and the Translator. (Mean classification error (%) of 5 runs). Right: Effect of number of layers in the paraphraser.

3.3.3 Ablation Study

In the introduction, we have described that the teacher network provides more paraphrased information to the student network via factors, and described a need for a translator to act as a buffer to better understand factors in the student network. To further analyze the role of factor, we performed an ablation experiment on the presence or absence of a paraphraser and a translator. The result is shown in Table 3.4. The student network and the teacher network are selected with different number of output channels. One can adjust the number of student and teacher factors by adjusting the paraphrase rate k of the paraphraser. As described above, since the role of the paraphraser (making F_T with unsupervised training loss) and the translator (trained jointly with student network to ease the learning of Factor Transfer) are not the same, we can confirm that the synergy of two modules maximizes the performance of the student network. Also, we report the performance of different number of layers in the paraphraser. As the number of layers increases, the performance also increases.

Method	Network	Top-1	Top-5
Student	Resnet-18	29.91	10.68
KD	Resnet-18	33.83	12.55
AT	Resnet-18	29.36	10.23
FT ($k = 0.5$)	Resnet-18	**28.57**	**9.71**
Teacher	Resnet-34	26.73	8.57

Table 3.5: Top-1 and Top-5 classification error (%) on ImageNet dataset. All the numbers are from our implementation.

Method	mAP
Student(VGG-16)	69.5
FT(VGG-16, $k = 0.5$)	70.3
Teacher(ResNet-101)	75.0

Table 3.6: Mean average precision on PASCAL VOC 2007 test dataset.

3.3.4 ImageNet

The ImageNet dataset is a image classification dataset which consists of 1.2M training images and 50K validation images with 1,000 classes. We conducted large scale experiments on the ImageNet LSVRC 2015 in order to show our potential availability to transfer even more complex and detailed informations. We chose ResNet-18 as a student network and ResNet-34 as a teacher network same as in [86] and validated the performance based on top-1 and top-5 error rates as shown in Table 3.5.

As can be seen in Table 3.5, FT consistently outperforms the other methods. The KD, again, suffers from the depth difference problem, as already confirmed in the result of other experiments. It shows just adding the FT loss helps to lower about 1.34% of student network's (ResNet-18) Top-1 error on ImageNet.

3.3.5 Object Detection

In this experiment, we wanted to verify the generality of FT, and decided to apply it on detection task, other than classifications. We used Faster-RCNN

pipeline [70] with PASCAL VOC 2007 dataset [20] for object detection. We used PASCAL VOC 2007 trainval as training data and PASCAL VOC 2007 test as testing data. Instead of using our own ImageNet FT pretrained model as a backbone network for detection, we tried to apply our method for transferring knowledges about object detection. Here, we set a hypothesis that since the factors are extracted in an unsupervised manner, the factors not only can connote the core knowledge of classification, but also can convey other types of representations.

In the Faster-RCNN, the shared convolution layers contain knowledges of both classification and localization, so we applied factor transfer to the last layer of shared convolution layer. Figure 3.2 shows where we applied FT in the Faster-RCNN framework. We set VGG-16 as a student network and ResNet-101 as a teacher network. Both networks are fine-tuned at PASCAL VOC 2007 dataset with ImageNet pretrained model. For FT, we used ImageNet pretrained VGG-16 model and fixed the layers before conv3 layer during training phase. Then, by the factor transfer, the gradient caused by the $\mathcal{L}_{FT}$ loss back-propagates to the student network passing by the student translator.

As can be seen in Table 3.6, we could get performance enhancement of 0.8 in mAP (mean average precision) score by training Faster-RCNN with VGG-16. As mentioned earlier, we have strong belief that the latter layer we apply the factor transfer, the higher the performance enhances. However, by the limit of VGG-type backbone network we have used, we tried but could not apply FT else that the backbone network. Experiment on the capable case where the FT can be applied to the latter layers like region proposal network (RPN) or other types of detection network will be our future work.

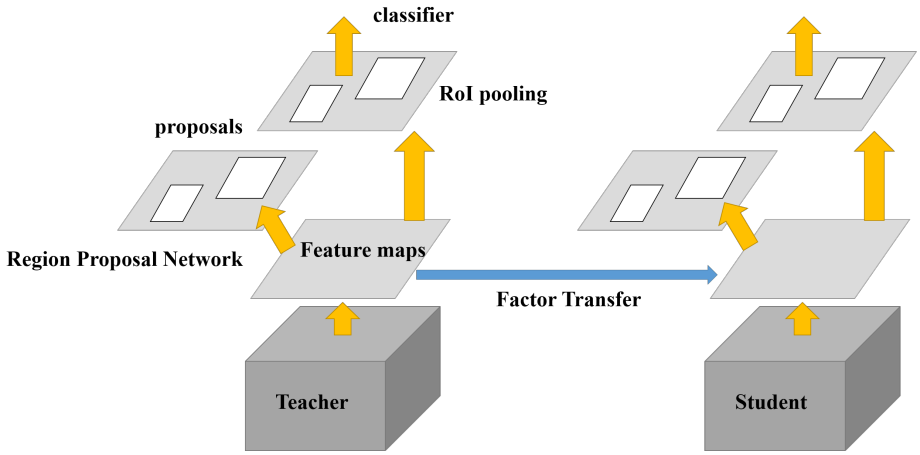


Figure 3.2: Factor transfer applied to Faster-RCNN framework

3.3.6 Discussion

In this section, we compare FitNet [72] and FT. FitNet transfers information of an intermediate layer while FT uses the last layer, and the purpose of the regressor in FitNet is somewhat different from our translator. More specifically, Romero *et al*. [72] argued that giving hints from deeper layer over-regularizes the student network. On the contrary, we chose the deeper layer to provide more specific information as mentioned in the work. Also, FitNet does not use the paraphraser as well. Note that FitNet is actually a 2-stage algorithm in that they initialize the student weights with hints and then train the student network using Knowledge Distillation.

3.4 Conclusion

In this work, we propose the factor transfer which is a novel method for knowledge transfer. Unlike previous methods, we introduce factors which contain paraphrased information of the teacher network, extracted from the paraphraser.

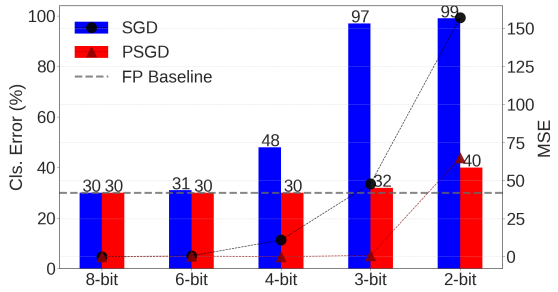
There are mainly two reasons that the student can understand information from the teacher network more easily by the factor transfer than other methods. One reason is that the factors can relieve the inherent differences between the teacher and student network. The other reason is that the translator of the student can help the student network to understand teacher factors by mimicking the teacher factors. A downside of the proposed method is that the factor transfer requires the training of a paraphraser to extract factors and needs more parameters of the paraphraser and the translator. However, the convergence of the training for the paraphraser is very fast and additional parameters are not needed after training the student network. In our experiments, we showed the effectiveness of the factor transfer on various image classification datasets. Also, we verified that factor transfer can be applied to other domains than classification. We think that our method will help further researches in knowledge transfer.

Chapter 4

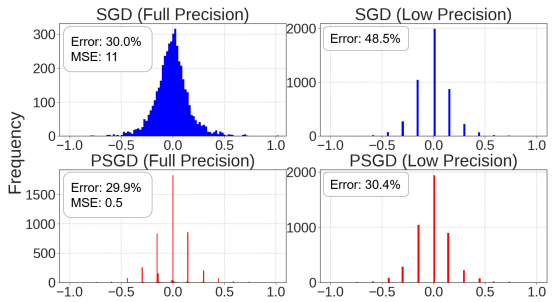
Position based Scaled Gradients (PSG) for Quantization

4.1 Introduction

Regularization strategies have been proposed to induce a prior to neural networks [37, 79, 35, 44, 43, 12]. Inspired by such regularization methods which induce a prior for a specific purpose, in this work we propose a novel regularization method that non-uniformly scales gradient for model compression problems. The scaled gradient, whose scale depends on the position of the weight, constrains the weight to a set of compression-friendly grid points. We replace the conventional gradient in the stochastic gradient descent (SGD) with the proposed position-based scaled gradient (PSG) and call it as PSGD. We show that applying PSGD in the original weight space is equivalent to optimizing the weights by the standard SGD in a warped space, to which weights from the original space are warped by an invertible function. The invertible warping function is designed such that the weights of the original space are forced to merge to the desired target positions by scaling the gradients.



(a) Classification and Quantization Error



(b) Weight Distribution of Full and 4-bit Precision

Figure 4.1: Results of ResNet-34 on CIFAR-100. (a) Mean-squared quantization error (line) and classification error (bar) across different bits. Blue: SGD, Red: PSGD. (b) Example of weight distribution (Conv2_1 layer [31]) trained with standard SGD and our PSGD. For PSGD, the distribution of the full precision weights closely resembles the low precision distribution, yet maintains its accuracy.

We are not the first to scale the gradient elements. The *scaled gradient method* which is also known as the *variable metric method* [14] multiplies a positive definite matrix to the gradient vector to scale the gradient. It includes a wide variety of methods such as the Newton method, Quasi-Newton methods and the natural gradient method [16, 69, 6]. Generally, they rely on Hessian estimation or Fisher information matrix for their scaling. However, our method is different from them in that our scaling does not depend on the loss function but it depends

solely on the current position of the weight.

We apply the proposed PSG method to the model compression problems such as quantization and pruning. In recent years, deploying a deep neural network (DNN) on restricted edge devices such as smartphones and IoT devices has become a very important issue. For these reasons, reducing bit-width of model weights (quantization) and removing unimportant model weights (pruning) have been studied and widely used for applications. Majority of the literature in quantization, dubbed as Quantization Aware Training (QAT) methods, fine-tunes a pre-trained model on the low precision domain without considering the full precision domain using the entire training dataset. Moreover, this scenario is restrictive in real-world applications because additional training is needed. In the additional training phase, a full-size dataset and high computational resources are required which prohibits easy and fast deployment of DNNs on edge devices for customers in need.

To resolve this problem, many works have focused on post-training quantization (PTQ) methods that do not require full-scale training [48, 66, 3, 90]. For example, [66] starts with a pre-trained model with only minor modification on the weights by equalizing the scales across channels and correcting biases. However, inherent discrepancy in the distribution of the pre-trained model and that of the quantized model is too large for the aforementioned methods to offset the fundamental difference in the distributions. As shown in Fig. 4.1, due to the differences in the two distributions, the classification error and the quantization error, denoted as the mean squared error increase as lower bit-width is used. Accordingly, when it comes to layer-wise quantization, existing post-training methods suffer significant accuracy degradation when it is quantized below 6-bit.

Meanwhile, another line of research in quantization has recently emerged

that approaches the task from the initial training phase [2]. Our method follows this scheme of training from scratch like standard SGD, but we attain a competent full-precision model that can also be effortlessly quantized to a low precision model with no additional post-processing. In essence, our main goal is to train a compression-friendly model that can be easily compressed when the resources are limited, without the need of re-training, fine-tuning and even accessing the data. To achieve this, we constrain the original weights to merge to a set of quantized grid points (Fig. 4.1(b)) by scaling their gradients proportional to the error between the original weight and its quantized version. For pruning, the weights are regularized to merge to zero. More details will be described in Sec 4.2.

Our contributions can be summarized as follows:

• We propose a novel regularization method for model compression by introducing the position-based scaled gradient (PSG) which can be considered as a variant of the variable metric method.

• We prove theoretically that PSG descent (PSGD) is equivalent to applying the standard gradient descent in the warped weight space. This leads the weight to converge to a well-performing local minimum in both compressed and uncompressed weight spaces (see Fig. 4.1).

• We interpret the warped space of PSGD using the steepest descent method with quadratic norm, which tries to make the space wide inversely proportional to quantization error (Eq 4.26). This phenomenon is also experimentally observed in Sec. 4.4.1.

• We verify the effectiveness of PSG on CIFAR and ImageNet datasets, applying PSG in quantization, pruning, and knowledge distillation. We also show that PSGD is very effective for extremely low bit quantization. Furthermore,

when PSGD-pretrained model is used along with a concurrent PTQ method, it outperforms its SGD-pretrained counterpart.

This work is the expanded version of our previous research [46]. We additionally verify PSGD with the recent iterative pruning framework. Also, we show that our PSGD as an implicit regularizer not modifying the objective function [82] works well with knowledge distillation which is one of the explicit regularizations. Finally, we interpret the geometry of the warped space from PSGD using steepest descent method.

4.2 Proposed method

In this section, we describe the proposed position-based scaled gradient descent (PSGD) method. In PSGD, a scaling function regularizes the original weight to merge to one of the desired target points which performs well at both uncompressed and compressed domains. This is equivalent to optimizing via SGD in the warped weight space. With a specially designed invertible function that warps the original weight space, the loss function in this warped space converges to a different local minima that are more compression-friendly compared to the solutions driven in the original weight space.

We first prove that optimizing in the original space with PSGD is equivalent to optimizing in the warped space with gradient descent. Then, we demonstrate how PSGD is used to constrain the weights to a set of desired target points. Lastly, we provide explanation on how this method is able to yield comparable performance with that of vanilla SGD in the original uncompressed domain, despite its strong regularization effect.

4.2.1 Optimization in warped space

Theorem 1. *Let $\mathcal{F} : \mathcal{X} \to \mathcal{Y},\ \mathcal{X}, \mathcal{Y} \subset \mathbb{R}^n$, be an arbitrary invertible multivariate function that warps the original weight space $\mathcal{X}$ into $\mathcal{Y}$ and consider the loss function $\mathcal{L} : \mathcal{X} \to \mathbb{R}$ and the equivalent loss function $\mathcal{L}' = \mathcal{L} \circ \mathcal{F}^{-1} : \mathcal{Y} \to \mathbb{R}$. Then, the gradient descent (GD) method in the warped space $\mathcal{Y}$ is equivalent to applying a scaled gradient descent in the original space $\mathcal{X}$ such that*

$$GD(\boldsymbol{y}, \nabla_{\boldsymbol{y}}^{\mathcal{L}'}) \equiv GD(\boldsymbol{x}, (\mathcal{J}_{\boldsymbol{x}}^{\mathcal{F}})^{-2} \nabla_{\boldsymbol{x}}^{\mathcal{L}}), \tag{4.1}$$

where $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x})$ and ∇_a^b and $\mathcal{J}_a^b$ respectively denote the gradient and Jacobian of the function b with respect to the variable a.

Proof. Consider the point $\boldsymbol{x}_t \in \mathcal{X}$ at time t and its warped version $\boldsymbol{y}_t \in \mathcal{Y}$. To find the local minimum of $\mathcal{L}'(\boldsymbol{y})$, the standard gradient descent method at time step t in the warped space can be applied as follows:

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t - \eta \nabla_{\boldsymbol{y}}^{\mathcal{L}'}(\boldsymbol{y}_t). \tag{4.2}$$

Here, $\nabla_{\boldsymbol{y}}^{\mathcal{L}'}(\boldsymbol{y}_t) = \frac{\partial \mathcal{L}'}{\partial \boldsymbol{y}}|_{\boldsymbol{y}_t}$ is the gradient and η is the learning rate. Applying the inverse function $\mathcal{F}^{-1}$ to $\boldsymbol{y}_{t+1}$, we obtain the updated point $\boldsymbol{x}_{t+1}$:

$$\begin{aligned} \boldsymbol{x}_{t+1} = \mathcal{F}^{-1}(\boldsymbol{y}_{t+1}) &= \mathcal{F}^{-1}(\boldsymbol{y}_t - \eta \nabla_{\boldsymbol{y}}^{\mathcal{L}'}(\boldsymbol{y}_t)) \\ &= \mathcal{F}^{-1}(\boldsymbol{y}_t) - \eta \mathcal{J}_{\boldsymbol{y}}^{\boldsymbol{x}}(\boldsymbol{y}_t) \nabla_{\boldsymbol{y}}^{\mathcal{L}'}(\boldsymbol{y}_t) \end{aligned} \tag{4.3}$$

where the last equality is from the first-order Taylor approximation around $\boldsymbol{y}_t$ and $\mathcal{J}_{\boldsymbol{y}}^{\boldsymbol{x}} = \mathcal{J}_{\boldsymbol{y}}^{\mathcal{F}^{-1}} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} \in \mathbb{R}^{n \times n}$ is the Jacobian of $\boldsymbol{x} = \mathcal{F}^{-1}(\boldsymbol{y})$ with respect to $\boldsymbol{y}$. By the chain rule, $\nabla_{\boldsymbol{y}}^{\mathcal{L}'} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \mathcal{J}_{\boldsymbol{y}}^{\boldsymbol{x}} \nabla_{\boldsymbol{x}}^{\mathcal{L}}$. Because $\mathcal{J}_{\boldsymbol{y}}^{\boldsymbol{x}} = (\mathcal{J}_{\boldsymbol{x}}^{\boldsymbol{y}})^{-1} = (\mathcal{J}_{\boldsymbol{x}}^{\mathcal{F}})^{-1}$, we can rewrite Eq. 4.3 as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta (\mathcal{J}_{\boldsymbol{x}}^{\mathcal{F}}(\boldsymbol{x}_t))^{-2} \nabla_{\boldsymbol{x}}^{\mathcal{L}}(\boldsymbol{x}_t). \tag{4.4}$$

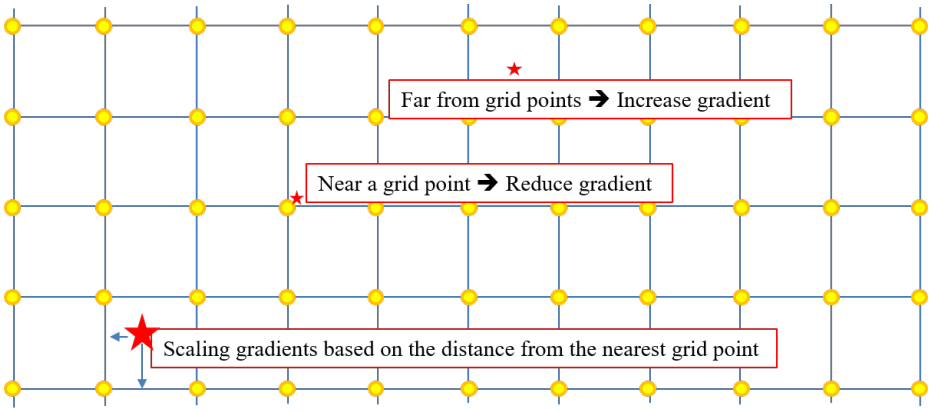


Figure 4.2: The main idea of PSGD. Suppose the yellow points indicate the quantization grid in a two-dimensional space. During training in the FP domain, if the position of the weight vector is close to a quantization grid, the gradient of that weight vector is scaled down proportionally to prevent it from escaping. Conversely, if it is distant, the gradient is scaled up so as to accelerate its escape from its original position.This idea is equivalent to multiplying a scaling factor to the gradients based on the distance from the nearest grid point.

Now Eq. 4.2 and Eq. 4.4 are equivalent and Eq. 4.1 is proved. In other words, the scaled gradient descent (PSGD) in the original space $\mathcal{X}$, whose scaling is determined by the matrix $(\mathcal{J}_{\boldsymbol{x}}^{\mathcal{F}})^{-2}$, is equivalent to gradient descent in the warped space $\mathcal{Y}$. □

4.2.2 Position-based scaled gradient

In this part, we introduce one example of designing the invertible function $\mathcal{F}(\boldsymbol{x})$ for scaling the gradients. This invertible function should cause the original weight vector $\boldsymbol{x}$ to merge to a set of desired target points $\{\bar{\boldsymbol{x}}\}$. These kinds of desired target weights can act as a prior in the optimization process to constrain the

original weights to merge at specific positions. The details of how to set the target points will be deferred to the next subsection.

The gist of weight-dependent gradient scaling is simple. For a given weight vector, if the specific weight element is far from the desired target point, a higher scaling value is applied so as to escape this position faster. On the other hand, if the distance is small, lower scaling value is applied to prevent the weight vector from deviating from the position (See Fig. 4.2). From now on, we focus on the design of the scaling function for the quantization problem. For pruning, the procedure is analogous and we omit the detail.

Scaling function: We use the same warping function f for each coordinate $x_i, i \in \{1, \cdots, n\}$ independently, i.e. $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}) = [f(x_1), f(x_2), \cdots f(x_n)]^T$. Thus the Jacobian matrix becomes diagonal ($\mathcal{J}_{\boldsymbol{x}}^{\mathcal{F}} = \text{diag}(f'(x_1), \cdots, f'(x_n))$ and our method belongs to the diagonally scaled gradient method.

Consider the following warping function

$$f(x) = 2 \text{ sign}(x - \bar{x})(\sqrt{|x - \bar{x}| + \epsilon} - \sqrt{\epsilon}) + c(\bar{x}) \tag{4.5}$$

where the target $\bar{x}$ is determined as the closest grid point from x, $\text{sign}(x) \in \{\pm 1, 0\}$ is a sign function and $c(\bar{x})$ is a constant dependent on the specific grid point $\bar{x}$ making the function continuous. We introduced $c(\bar{x})$ for making $f(x)$ continuous. If we do not add a constant $c(\bar{x})$, the $f(x)$ has points of discontinuity at every $\{(n + 0.5)\Delta | n \in \mathbb{Z}\}$ as depicted in Fig. 4.3, where Δ represents step size and $n\Delta$ means n-th quantized value identical to $\bar{x}$ corresponding to x. We can calculate the left sided limit and right sided limit at $n\Delta + 0.5\Delta$ using Eq. 4.5.

$$f(n\Delta + 0.5\Delta^-) = 2(\sqrt{0.5\Delta + \epsilon} - \sqrt{\epsilon}) + c(n\Delta)$$
$$f(n\Delta + 0.5\Delta^+) = -2(\sqrt{0.5\Delta + \epsilon} - \sqrt{\epsilon}) + c((n+1)\Delta)$$

Based on the condition that the left sided limit and the right sided limit should be the same, we can get the following recurrence relation:

$$c((n+1)\Delta) - c(n\Delta) = 4\ (\sqrt{0.5\Delta + \epsilon} - \sqrt{\epsilon}).$$

Using the successive substitution for calculating $c(\bar{x})$, it becomes

$$c(n\Delta) - c(0) = 4n\ (\sqrt{0.5\Delta + \epsilon} - \sqrt{\epsilon}).$$

Setting $c(0) = 0$ and because $n\Delta = \bar{x}$, $c(\bar{x})$ can be calculated as below:

$$c(\bar{x}) = \frac{4\bar{x}}{\Delta}\ (\sqrt{0.5\Delta + \epsilon} - \sqrt{\epsilon}). \tag{4.6}$$

ϵ is an arbitrarily small constant to avoid infinite gradient. Then, from Eq. 4.4, the elementwise scaling function becomes $s(x) = \frac{1}{[f'(x)]^2}$ and consequently

$$s(x) = |x - \bar{x}(x)| + \epsilon. \tag{4.7}$$

Using the elementwise scaling function Eq. 4.7, the elementwise weight update rule for the PSG descent (PSGD) becomes

$${x_i}^{t+1} = {x_i}^{t} - \eta s(x_i)\ \left.\frac{\partial \mathcal{L}}{\partial x_i}\right|_{\boldsymbol{x}^t} \tag{4.8}$$

where, η is the learning rate.[1] We further elaborate on the geometry of the warped space using the concept of steepest descent in the p-norm in Section 4.2.5.

PSGD operates independent of the type of the loss function as it does not modify the loss term, but rather non-uniformly scales the gradient elements. Therefore, it can be applied to KD loss containing task loss $\mathcal{L}$ (e.g. cross-entropy) and KL loss.Assuming that there are n classes, softmax posterior with temperature $\mathcal{T}$ can be calculated as follows:

[1] We set $\eta = \eta_0 \lambda_s$ where η_0 is the conventional learning rate and λ_s is a hyper-parameter that can be set differently for various scaling functions depending on their range.

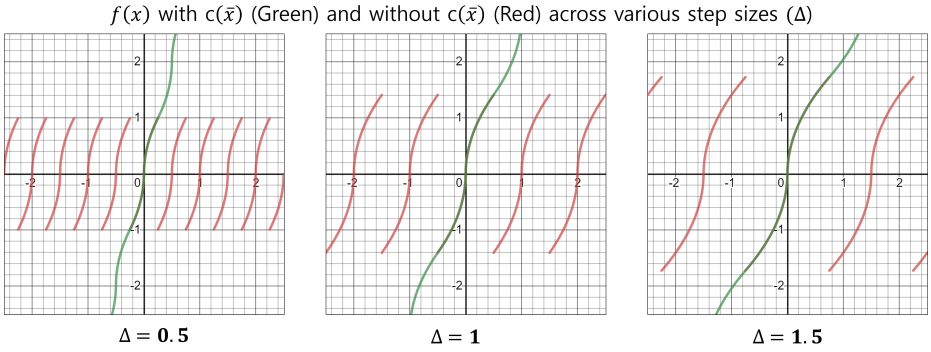


Figure 4.3: Scaling function $f(x)$ for different step size Δ. The red graph depicts $f(x)$ without $c(\bar{x})$ and the green graph depicts $f(x)$ with $c(\bar{x})$ (Eq. 4.6). Without $c(\bar{x})$, there are points of discontinuity at every $\{(n+0.5)\Delta | n \in \mathbb{Z}\}$. After adding $c(\bar{x})$ to the scaling function $f(x)$, it becomes a continuous function (green).

$$p_k(\mathbf{z}; \mathcal{T}) = \frac{e^{z_k/\mathcal{T}}}{\sum_j^n e^{z_j/\mathcal{T}}}, \tag{4.9}$$

where $\mathbf{z}_k$ represents a k-th logit. The temperature value, $\mathcal{T}$, is used to make soft logits for knowledge distillation. We can compute the KL loss between student and teacher network using following equation.

$$KL(z^T || z^S; \mathcal{T}) = \sum_{i=1}^{n} p_k(z^T; \mathcal{T}) \log(\frac{p_k(z^T; \mathcal{T})}{p_k(z^S; \mathcal{T})}). \tag{4.10}$$

Where Z^T and Z^S are teacher logit and student logit,respectively. Then, we can use PSGD with KD loss combining task loss and KL loss as below:

$$\mathcal{L}_{KD} = \mathcal{L} + \mathcal{T}^2 \times KL(z^T || z^S; \mathcal{T}) \tag{4.11}$$

$\mathcal{L}_{KD}$ refers to the KD loss. We multiply $\mathcal{T}^2$ because the decrease rate of the gradient scale is $1/\mathcal{T}^2$. Using KD loss, the update ruls for the PSGD with KD becomes

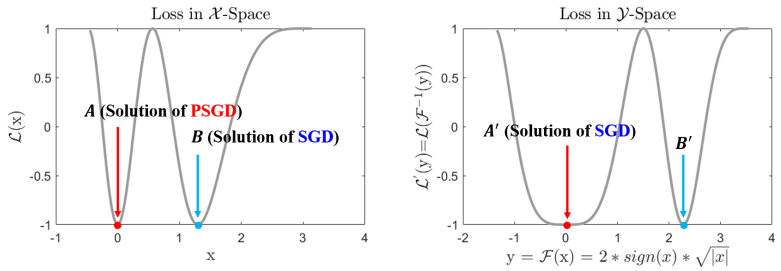


Figure 4.4: Toy example of warping a loss function $\mathcal{L}(x) = \cos\left((x - 3.07)^2\right)$. **Left** denotes the original loss function. **Right** is drawn by warping the original function by Eq. 4.5 with the target $\bar{x} = 0$.

$$x_i{}^{t+1} = x_i{}^{t} - \eta s(x_i) \left.\frac{\partial \mathcal{L}_{KD}}{\partial x_i}\right|_{\boldsymbol{x}^t} \tag{4.12}$$

4.2.3 Target points

Quantization: In this work, we use the uniform symmetric quantization method [48] and the per-layer quantization scheme for hardware friendliness. Consider a floating point range [min_x,max_x] of model weights. The weight x is quantized to an integer ranging [$-2^{n-1}+1$,$2^{n-1}-1$] for n-bit precision. Quantization-dequantization for the weights of a network is defined with step-size (Δ) and clipping values. The overall quantization process is as follows:

$$\begin{aligned} x_Q = Clip(\left\lfloor \frac{x}{\Delta} \right\rceil, -2^{n-1}+1, 2^{n-1}-1), \\ \Delta = \frac{max(-min_x, max_x)}{2^{n-1}-1} \end{aligned} \tag{4.13}$$

where $\lfloor \cdot \rceil$ is the round to the closest integer operation and $Clip(x, a, b) =$

$$\begin{cases} b & \text{if} \quad x > b \\ a & \text{if} \quad x < a \\ x & \text{elsewise.} \end{cases}$$

We can get the quantized weights with the de-quantization process as $\bar{x} = x_Q \times \Delta$ and use this quantized weights for target positions of quantization.

Pruning: For magnitude-based pruning methods, weights near zero are removed. Therefore, we choose zero as the target value (i.e. $\bar{x} = 0$).

4.2.4 PSGD for deep networks

Many literature focusing on the optimization of DNNs with stochastic gradient descent (SGD) have reported that multiple experiments give consistently similar performance although DNNs have many local minima (e.g. see Sec. 2 of [9]). [10] analyzed the loss surface of DNNs and showed that large networks have many local minima with similar performance on the test set and the lowest critical values of the random loss function are located in a specific band lower-bounded by the global minimum. From this respect, we explain informally how PSGD for deep networks works. As illustrated in Fig. 4.4, we posit that there exist many local minima (A, B) in the original weight space $\mathcal{X}$ with similar performance, only some of which (A) are close to one of the target points (0) exhibiting high performance also in the compressed domain. As in Fig. 4.4 left, assume that the region of convergence for B is much wider than that of A, meaning that there exists more chance to output solution B rather than A from random initialization. By the warping function $\mathcal{F}$ specially designed as described above (Eq. 4.5), the original space $\mathcal{X}$ is warped to $\mathcal{Y}$ such that the areas near target points are expanded while those far from the targets are contracted. If we apply gradient descent in this warped space, the loss function will have a better chance of converging to A'. Correspondingly, PSGD in the original space will more likely output A rather than B, which is favorable for compression. Note that $\mathcal{F}$ transforms the original weight space to the warped space $\mathcal{Y}$ not to the compressed

domain.

4.2.5 Geometry of the Warped Space

In this section, we further illustrate the exact geometry of the warped space when PSGD is applied to quantization. Recall from Eq. 4.4, Eq. 4.8, and Eq. 4.13 that the absolute magnitude of the quantization error is used to scale the gradient elements. This corresponds to left-multiplying a diagonal matrix with the elements determined by the magnitude of the quantization error. We use the concept of p-norm steepest descent [7] to illustrate why this leads to a warped space that induces the weight vectors to merge to the target points. First, we explain some necessary preliminary details for completeness.

Steepest Descent Method: For a first-order optimization method, the steepest descent direction, v is determined by minimizing the the first-order Taylor approximation of $L(x+v)$ around x.

$$\mathcal{L}(x+v) \approx \mathcal{L}(x) + \nabla\mathcal{L}(x)^T v \tag{4.14}$$

Since v can be chosen to have arbitrarily large magnitude in a particular direction, the magnitude is normalized as

$$v_{nsd} = \text{argmin}\{\nabla\mathcal{L}(x)^T v \mid \|v\|_p \leq 1\} \tag{4.15}$$

Naturally, using different values of p will yield distinct steepest directions. Additionally, other family of norms can also be used such as the quadratic norms, which is defined for a positive-definite matrix $\mathcal{A}$ as

$$\|v\|_{\mathcal{A}} = (v^t \mathcal{A} v^t)^{\frac{1}{2}} = \|\mathcal{A}^{\frac{1}{2}} v\|_2 \tag{4.16}$$

One can also consider the *unormalized* steepest descent, which scales the nor-

malized steepest descent by the dual norm.

$$v_{sd} = \|\nabla \mathcal{L}(x)\|_* v_{nsd} \tag{4.17}$$

where $\|\cdot\|_*$ denotes the dual norm

$$\|z\|_* = \sup_x \{|z^t x| \mid \|x\| \leq 1\} \tag{4.18}$$

For the Euclidean norm ($p = 2$), v_{nsd} corresponds to $-\nabla \mathcal{L}(x)$, which is denoted as gradient descent. Now we present our theorem by interpreting PSGD as steepest descent method in the quadratic norm.

Lemma 1. *For a fixed iteration t, the unormalized steepest descent direction in the quadratic norm $\|\cdot\|_{\mathcal{A}}$ is equivalent to the PSG descent direction if the symmetric, positive-definite matrix $\mathcal{A}$ is given by*

$$\mathcal{A}(t) = diag(\frac{1}{s(x_i^t)}, \cdots, \frac{1}{s(x_n^t)}) \tag{4.19}$$

where $s(x)$ is given by Eq. 4.7 and n is the dimension of the weight vectors.

Proof. First note that the normalized steepest descent in the Euclidean norm is simply given by the negative direction of the gradient scaled by its norm.

$$-\frac{\nabla \mathcal{L}(x)}{\|\nabla \mathcal{L}(x)\|_2} = \text{argmin}\{\nabla \mathcal{L}(x)^T v \mid \|v\|_2 \leq 1\} \tag{4.20}$$

The steepest descent in the quadratic norm can easily be formulated as above with change of variables.

$$\begin{aligned} v_{nsd} &= \text{argmin}\{\nabla \mathcal{L}(x)^T v \mid \|v\|_{\mathcal{A}} \leq 1\} \\ &= \text{argmin}\{\nabla \mathcal{L}(x)^T v \mid \|\mathcal{A}^{\frac{1}{2}} v\|_2 \leq 1\} \\ &= \text{argmin}\{\nabla \mathcal{L}(x)^T \mathcal{A}^{-\frac{1}{2}} h \mid \|h\|_2 \leq 1\} \end{aligned} \tag{4.21}$$

where the last equality follows from the change-of-variable $h = \mathcal{A}^{\frac{1}{2}} v$. Then, the descent direction is given by

$$h_{nsd} = -\frac{\mathcal{A}^{-\frac{1}{2}} \nabla \mathcal{L}(x)}{\|\mathcal{A}^{-\frac{1}{2}} \nabla \mathcal{L}(x)\|_2} \tag{4.22}$$

or equivalently,

$$v_{nsd} = -\frac{\mathcal{A}^{-1} \nabla \mathcal{L}(x)}{\|\mathcal{A}^{-\frac{1}{2}} \nabla \mathcal{L}(x)\|_2} \tag{4.23}$$

To yield the unormalized descent direction, we compute the dual norm of $\|\nabla \mathcal{L}\|_{\mathcal{A}}$, which is precisely $\sup_x \{|\nabla \mathcal{L}(x)^T x| \mid \|x\|_{\mathcal{A}} \leq 1\} = \|\mathcal{A}^{-\frac{1}{2}} \nabla \mathcal{L}(x)\|_2$. Thus the unormalized descent direction is

$$v_{sd} = -\mathcal{A}^{-1} \nabla \mathcal{L}(x) \tag{4.24}$$

, which written in element-wise for the ith element is equivalent to the PSGD update rule given in Eq. 4.8.

$$v_{sd}^i = -s(x_i) \frac{\partial \mathcal{L}}{\partial x_i} \tag{4.25}$$

□

Theorem 2. *Given weight spaces $\mathcal{X}, \mathcal{Y} \subset \mathbb{R}^n$, and a symmetric, positive-definite matrix $\mathcal{A} \in \mathbb{R}^{n*n}$, let $\mathcal{X}$ and $\mathcal{Y}$ be the weight spaces obtained by PSG descent method and the gradient descent method respectively. Then, the linear transformation from $\mathcal{X}$ to $\mathcal{Y}$ at iteration t is given by*

$$\mathcal{T}_t : \mathcal{X} \to \mathcal{Y} = \mathcal{A}(t)^{\frac{1}{2}} x \tag{4.26}$$

Thus, for a weight vector x_j^t with small quantization error, the jth basis is expanded inversely proportional to the error, rendering x_j^{t+1} in the vicinity of the target point for a given update.

Table 4.1: λ_s used in the sparse training experiment.

CIFAR-100 & ResNet-32	Sparsity (%)				
	20.0	50.0	70.0	80.0	90.0
λ_s	100	100	200	600	1200

Table 4.2: λ_s used in the quantization experiments.

ResNet-18	ImageNet			CIFAR-10		
	8-bit	6-bit	4-bit	8-bit	6-bit	4-bit
λ_s	500	500	1000	10	10	10

Proof. For the simplicity of notation, t is omitted below as the proof applies to any fixed t. Consider the loss function defined in $\mathcal{Y}$.

$$\widetilde{\mathcal{L}}(y) = \mathcal{L}(\mathcal{A}^{-\frac{1}{2}}y) = \mathcal{L}(x) \tag{4.27}$$

The gradient descent direction in y is given by

$$\begin{aligned} v_y &= -\nabla_y \widetilde{\mathcal{L}}(y) = -\frac{\partial \widetilde{\mathcal{L}}(y)}{\partial y} \\ &= -\frac{\partial x}{\partial y}\frac{\partial \mathcal{L}(x)}{\partial x} \\ &= -\mathcal{A}^{-\frac{1}{2}}\nabla_x \mathcal{L} \end{aligned} \tag{4.28}$$

Applying the inverse transformation of Eq. 4.26 yields the gradient descent direction in x, which is equivalent to the unormalized steepest descent direction in the qudratic norm given by Eq. 4.24 in $y \in \mathcal{Y}$.

$$v_x = \mathcal{T}^{-1}(v_y) = -\mathcal{A}^{-1}\nabla_x \mathcal{L} \tag{4.29}$$

By Lemma 1, this is equivalent to the PSG descent. □

Table 4.3: Test accuracy of ResNet-32 across different sparsity ratios (percentage of zeros) on CIFAR-100 after magnitude-based pruning [28] without any fine-tuning.

Method	Sparsity (%)				
	20.0	50.0	70.0	80.0	90.0
SGD	69.43	60.59	15.95	4.70	1.00
L0 Reg. [61]	67.56	64.49	49.73	23.95	2.85
SNIP [55]	**69.68**	68.73	66.76	65.67	60.14
PSGD (Ours)	69.63	**69.25**	**68.62**	**67.27**	**64.33**

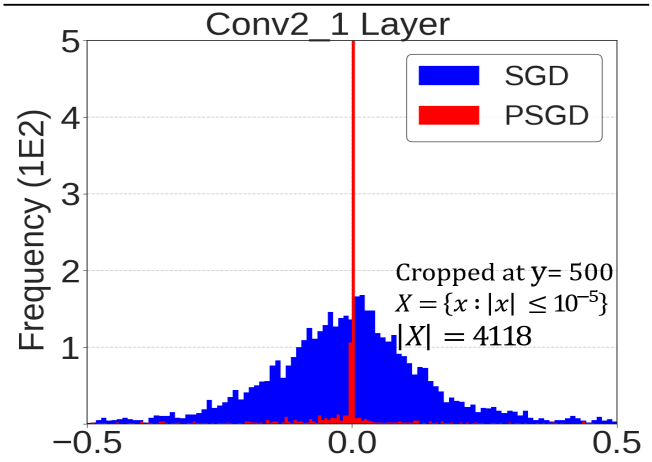


Figure 4.5: The weight distribution of SGD and PSGD models.

Table 4.4: Test accuracy of unstructured pruning for ResNet-20 on CIFAR 10 dataset. All numbers except ours are from [59]

Methods					Sparsity (%)
SNIP [55]	SM [17]	DSR [64]	DPF [59]	PSGD (Ours)	
88.50±0.13	89.76±0.40	87.88±0.04	**90.88±0.07**	90.47±0.03	90
84.91±0.25	83.03±0.74	–	88.01±0.30	**88.68±0.01**	95

4.3 Experiments

In this section, we experimentally show the effectiveness of PSGD. To verify our PSGD method, we first conduct experiments for sparse training by setting the target point as 0, then we further extend our method to quantization with CIFAR [49] and ImageNet ILSVRC 2015 [73] dataset. We first demonstrate the effectiveness in sparse training with magnitude-based pruning by comparing with L0-regularization [61] and SNIP [55]. [61] penalizes the non-zero model parameters and shares the scheme of regularizing the model while training. Like ours, [55] is a single-shot pruning method, which does not require pruning schedules nor additional fine-tuning. Then, we apply our PSGD with iterative pruning methods which contain pruning phase and finetuning phase at training.

For quantization, we compare our method with **(1)** methods that employ regularization at the initial training phase [2, 26, 58]. We choose gradient L1 norm regularization [2] method and Lipschitz regularization methods [58, 26] from the original paper [2] as baselines, because they propose new regularization techniques used at the training phase similar to us. Note that [26] adds an L2 penalty term on the gradient of weights instead of the L1 penalty like [2]. We also compare with **(2)** existing state-of-the-art layer-wise post-training quantization

methods that start from pre-trained models [66, 90] to show the improvement in lower bits (4-bit). Refer to Section 2.2 for the details on the compared methods. To validate the effectiveness of our method, we also train our model for extremely low bit (2,3-bit) weights.

For knowledge distillation, we conduct experiments to validate the compatibility of PSGD with knowledge distillation by applying it to the KD loss.

Lastly, we show the experimental results on various network architectures and applying PSG to the Adam optimizer [47] to show the adaptability of PSGD.

4.3.1 Implementation details

methods

We used the Pytorch framework for all experiments. For the pruning experiment of Table 4.3, we used ResNet-32 [31] on the CIFAR-100, following the training hyperparameters of [88]. We used released official implementations of [61] and re-implemented [55] for the Pytorch framework. In iterative pruning of Table 4.4, we followed the same setting of [59]. For quantization experiments of Table 4.5 and 4.6, we used ResNet-18 and followed [2] settings for CIFAR-10 and ImageNet. For [90], released official implementations were used for experiment. All other numbers are either from the original paper or re-implemented. For fair comparison, all quantization experiments followed the layer-wise uniform symmetric quantization [48] and when quantizing the activation, we clipped the activation range using batch normalization parameters as described in [66], same as [2]. PSGD is applied from the last 15 epochs for ImageNet experiments and from the first learning rate decay epoch for CIFAR experiments. We use additional 30 epochs for PSGD at extremely low bits experiments (Table 4.7).

Also, we tuned the hyper-parameter λ_s for each bit-widths and sparsity. Our search criteria is ensuring that the performance of uncompressed model is not degraded, similar to [2].

Datasets

We use CIFAR-10/100 and the ImageNet datasets for experiments. CIFAR-10 consists of 50,000 training images and 10,000 test images, consisting of 10 classes with 6000 images per class. CIFAR-100 consists of 100 classes with 600 images per class. The ImageNet dataset consists of 1.2 million images. We use 50,000 validation images for the test, which are not included in training samples. We use the conventional data pre-processing steps.[2] [3]

ImageNet / CIFAR-10 For ResNet-18, we started training with a L2 weight decay of 10^{-4} and learning rate of 0.1, then decayed the learning rate with a factor of 0.1 at every 30 epochs. Training was terminated at 90 epochs. We only used the last 15 epochs for training the model with PSGD similar to [2]. This means we applied the PSG method after 75 epochs with learning rate 0.001. For extremely low-bits experiments, we did not use any weight decay after 75 epochs. We tuned the hyper-parameters λ_s for target bit-widths. All numbers are results of the last epoch. We used the official code of [90] for comparisons with 0.02 for the Expand Ratio.[4]

CIFAR-100 For ResNet-32, the same weight decay and initial learning rate were used as above and the learning rate was decayed at 82 and 123 epoch following [88]. Training was terminated at 150 epoch. For VGG16 with batch-

[2] https://github.com/kuangliu/pytorch-cifar

[3] https://github.com/pytorch/examples/blob/master/imagenet/main.py

[4] https://github.com/cornell-zhang/dnn-quant-ocs

norm normalization (VGG16-bn), we decayed the learning rate at 145 epoch instead. We applied PSG after the first learning rate decay. The first convolutional layer and the last linear layer are quantizedat 8-bit for the 2-bit and the 3-bit experiments. For sparse training, training was terminated at 200 epoch and weight decay was not used at higher sparsity ratio, while all the other training hyperparameters were the same. For [61], we used the official implementation for the results.[5]

Hyper-parameter λ_s

We searched the appropriate λ_s with the criteria that the performance of the uncompressed model is not degraded, similar to [2]. For hyper-parameter tuning, we use two disjoint subsets of the training dataset for training and validation. Then we used the found λ_s to retrain on the whole training dataset. Table 4.1 and 4.2 show the values of λ_s used in experiments. The λ_s tended to rise for lower target bit-widths or for higher sparsity ratios. In CIFAR-10, we observe that same λ_s value yields fair performance across all bit-widths.

4.3.2 Pruning

Single-shot pruning As a preliminary experiment, we first demonstrate that PSG-based optimization is possible with a single target point set at zero. Then, we apply magnitude-based pruning following [28] across different sparsity ratios. As the purpose of the experiment is to verify that the weights are centered on zero, weights are pruned once after training has completed and the model is evaluated without fine-tuning for [61] and ours. Results for [55], which prunes the weights by single-shot at initialization, are shown for comparison on single-shot pruning.

[5] `https://github.com/AMLab-Amsterdam/L0_regularization`

Table 4.5: Test accuracy of regularization methods that do not have post-training process for ResNet-18 on the ImageNet and CIFAR dataset. PSGD@W# indicates the target number of bits for weights in PSGD is #. All numbers except ours are from [2]. At #-bit, PSGD@W# performs the best in most cases.

Method	ImageNet				CIFAR-10			
	FP	W8A8	W6A6	W4A4	FP	W8A4	W6A4	W4A4
SGD	69.70	69.20	63.80	0.30	93.54	85.51	85.35	83.98
DQ Regularization [58]	68.28	67.76	62.31	0.24	92.46	83.31	83.34	82.47
Gradient L2 [26]	68.34	68.02	64.52	0.19	93.31	84.50	84.99	83.82
Gradient L1 [2]	70.07	69.92	66.39	0.22	93.36	88.70	88.45	87.62
Gradient L1 ($\lambda = 0.05$) [2]	64.02	63.76	61.19	55.32	–	–	–	–
PSGD@W8 (Ours)	**70.22**	**70.13**	66.02	0.60	**93.67**	**93.10**	93.03	90.65
PSGD@W6 (Ours)	70.07	69.83	**69.51**	0.29	93.54	92.76	92.88	90.55
PSGD@W4 (Ours)	68.18	67.63	62.73	**63.45**	93.63	93.04	**93.12**	**91.03**

Table 4.6: Comparison with Post-training Quantization methods using ResNet-18 on the ImageNet dataset. Results of DFQ are from [66].

Method	W8A8	W6A6	W4A4
DFQ [66]	69.7	66.3	–
OCS + Best Clip [90]	69.37	66.76	44.3
PSGD (Ours)	**70.13**	**69.51**	**63.45**

Table 4.3 indicates that our method outperforms the two methods across various high sparsity ratios. While all three methods are able to maintain accuracy at low sparsity ($\sim$10%), [61] has some accuracy degradation at 20% and suffers severely at high sparsity. This is in line with the results shown in [21] that the method was unable to produce sparse residual models without significant damage to the model quality. Comparing with [55], our method is able to maintain higher accuracy even at high sparsity, displaying the strength in single-shot pruning, in

Table 4.7: Extremely low bits accuracy of ResNet-18 on the ImageNet dataset. The first convolutional layer and the last linear layer are quantized at 8-bit. Activation is fixed to 8-bit.

Method	(FP / W3A8)	(FP / W2A8)
SGD	**69.76** / 0.10	**69.76** / 0.10
PSGD (Ours)	66.75 / **66.36**	64.60 / **62.65**

Table 4.8: The performance of ResNet-32 on CIFAR-100. Teacher network is ResNet-56. In this experiment, we do not quantize the activation.

Method	(FP / 4-bit)	(FP / 3-bit)	(FP / 2-bit)
PSGD	70.08 / 69.57	68.96 / 68.56	67.16 / 60.76
PSGD + KD	**71.16 / 71.10**	**70.56 / 69.91**	**69.66 / 63.54**

which no pruning schedules nor additional training are necessary. Fig. 4.5 shows the distribution of weights in SGD- and PSGD-trained models.

Iterative pruning We also consider the iterative pruning case. Comparing single shot pruning methods, iterative pruning gradually increases the sparsity while training. This can help the recovery the performance via finetuning steps included in iterative pruning training schedule. We choose the iterative pruning schedule from [59]. Table 4.4 shows the performance of our method with SOTA pruning methods. PSGD is also very effective in the iterative pruning schemes. Even at 95% sparsity, PSGD outperforms the existing SOTA iterative pruning method with a simple training schedule and without any pruning schemes. These results verify that PSGD performs well with only scheduling comparable to the competitive iterative pruning. This can be possible because PSGD only scaled the gradient to regularize the weights to zero which is friendly to pruning.

Table 4.9: The performance of ResNet-18 on ImageNet, using ResNet-34 as a teacher network.

Method	(FP / W8A8)	(FP / W6A6)	(FP / W4A4)
PSGD	70.22 / 70.13	70.07 / 69.51	68.18 / 63.45
PSGD + KD	**71.28 / 71.23**	**71.40 / 70.82**	**69.76 / 63.70**

Table 4.10: The performances of various architectures with PSGD.

DataSet & Network	Method	(FP / W4A4)	(FP / W3A8)	(FP / W2A8)
CIFAR-100 & VGG16-bn	SGD	73.12 / 63.08	73.12 / 3.44	73.12 / 1.00
	PSGD	73.21 / 70.92	71.85 / 68.28	69.36 / 53.25
DataSet & Network	Method	(FP / W8A8)	(FP / W6A8)	(FP / W4A8)
ImageNet & DeseNet-121	SGD	74.43 / 73.85	74.43 / 70.57	74.43 / 0.36
	PSGD	75.16 / 75.03	75.12 / 74.84	72.60 / 72.26

4.3.3 Quantization

In the quantization domain, we first compare PSGD with regularization methods at the on-the-fly bit-widths problem, meaning that a single model is evaluated across various bit-widths. Then, we compare with existing state-of-the-art layer-wise symmetric post-training methods to verify handling the problem of accuracy drop at low bits due to the differences in weight distributions (See Fig. 4.1).

Regularization methods Table 4.5 shows the results of regularization methods on CIFAR-10 and ImageNet datasets, respectively. In the CIFAR-10 experiments of Table 4.5, we fix the activation bit-width to 4-bit and then vary the weight bit-widths from 8 to 4. For the ImageNet experiments of Table 4.5, we use equal bit-widths for both weights and activations, following [2]. In CIFAR-10

Table 4.11: ResNet-32 trained with Adam on the CIFAR-100 dataset. Vanilla Adam also suffers accuracy degradation on 4 bits, while applying PSG to Adam recovers the accuracy by more than 5%. Weight-only quantization is shown by W4A32.

Method	FP	W4A32	W4A4
Adam [47]	66.66	55.27	43.5
Adam with PSG	66.80	60.35	51.55

experiment, all methods seem to maintain the performance of the quantized model until 4-bit quantization. Regardless of target bit-widths, PSGD outperforms all other regularization methods. On the other hand, ImageNet experiment generally shows reasonable results until 6-bit but the accuracy drastically drops at 4-bit. PSGD targeting 8-bit and 6-bit marginally improves on all bits, yet also experiences drastic accuracy drop at 4-bit. In contrast, Gradient L1 ($\lambda = 0.05$) and PSGD @ W4 maintain the performance of the quantized models even at 4-bit. Comparing with the second best method Gradient L1 ($\lambda = 0.05$) [2], PSGD outperforms it at all bit-widths. At full precision (FP), 8-, 6- and 4-bit, the gap of performance between [2] and ours are about 4.2%, 3.9%, 1.5% and 8.1%, respectively. From Table 4.5, while the quantization noise may slightly degrade the accuracy in some cases, a general trend that using more bits leads to higher accuracy is demonstrated. Compared to other regularization methods, PSGD is able to maintain reasonable performance across all bits by constraining the distribution of the full precision weight to resemble that of the quantized weight. This quantization-friendliness is achieved by the appropriately designed scaling function. In addition, unlike [2], PSGD does not need additional overhead of

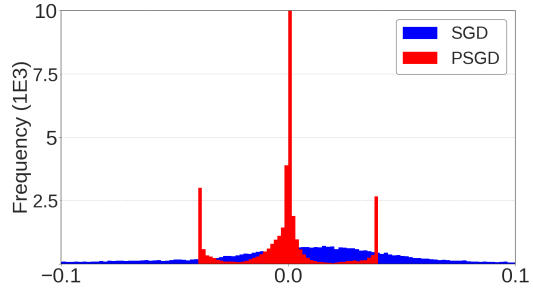
computing double-backpropagation.

Post-training methods Table 4.6 shows that OCS, state-of-the-art post-training method, has a drastic accuracy drop at 4-bit. For OCS, following the original paper, we chose the best clipping method for both weights and activation. DFQ also has a similar tendency of showing drastic accuracy drop under the 6-bit as depicted in Fig. 1 of the original paper of DFQ [66]. This is due to the fundamental discrepancy between FP and quantized weight distributions as stated in Fig. 4.1. On the other hand, models trained with PSGD have similar full-precision and quantized weight distributions and hence low quantization error due to the scaling function. Our method outperforms OCS at 4-bit by around 19% without any post-training and weight clipping to treat the outliers. Applying PTQ method to our PSG pre-trained model is shown in Sec 4.4.

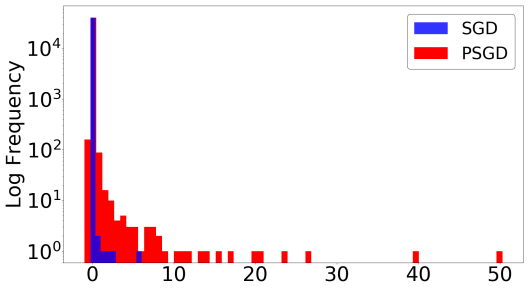
Extremely low bits quantization As shown in Fig. 4.1, SGD suffers drastic accuracy drop at extremely low bits such as 3-bit and 2-bit. To confirm that PSGD can handle extremely low bit, we conduct experiments with PSGD targeting 3-bit and 2-bit except the first and last layers which are quantized at 8-bit. Table 4.7 shows the results of applying PSGD. Although the full precision accuracy does drop due to the strong constraints, PSGD is able to maintain reasonable accuracy. This demonstrates the potential of PSGD as a key solution to post-training quantization at extremely low bits.

4.3.4 Knowledge Distillation

In this part, we show the adaptability of PSGD, which only manipulates the magnitude of the gradients from the loss function. We apply PSGD with another regularizer, Knowledge Distillation. We follow the update rule (Eq. 4.12) for quantization, using a KD framework. We utilize a powerful teacher network to



(a) Weight distribution (1st layer)



(b) Histogram of eigenvalues

Figure 4.6: Weight distribution and histogram of eigenvalues for MNIST dataset. The two-layered fully connected network consists of 50 and 20 hidden nodes. Target bit of PSGD is 2. Note that both solutions yield relatively small negative eigenvalues ($\lambda > -1$).

train a relatively small student network. We conduct two experiments on CIFAR-100 and ImageNet. In CIFAR-100, we use ResNet-32 as a student and ResNet-54 as a teacher network. In ImageNet, we use ResNet-18 and ResNet-34 as a student and teacher, respectively. Table 4.8 and 4.9 show similar tendency. Regardless of bit-width, network, and dataset, Combining KD and PSGD (Eq. 4.12) outperform using PSGD alone (Eq. 4.8). From this respect, we validate that PSGD can be used alongside with other regularizer because of its adaptability.

4.3.5 Various architectures with PSGD

In this section, we show the results of applying PSGD to various architectures. Table 4.10 shows the quantization results of VGG16 [75] with batch normalization on the CIFAR-100 dataset and DenseNet-121 [38] on the ImageNet dataset, respectively.

For DenseNet, we run additional 15 epochs from the pre-trained model to reduce the training time.[6] For fair comparisons in terms of the number of epochs, we also trained for additional 15 epochs for SGD with the same last learning rate (0.001). However, we only observed oscillation in the performance during the additional epochs. Similar to the extremely low-bits experiments, we fixed the activation bit-width to 8-bit.

For VGG16 on the CIFAR-100 dataset, similar tendency in performance was observed with ResNet-32. The 4-bit targeted model was able to maintain its full-precision accuracy, while the model targeting lower bit-widths had some accuracy degradation.

4.3.6 Adam optimizer with PSG

To show the applicability of our PSG to other types of optimizers, we applied our PSG to the Adam optimizer by using the same scaling function with ResNet-32 on 4-bits with the CIFAR-100 dataset. Following the convention, the initial learning rate of 10^{-3} was used and the first and the last layer of the model were fixed to 8-bits. All the other training hyperparameters remained the same. Table 4.11 compares the quantization results of models trained with vanilla Adam and applying PSG to Adam.

[6] `https://download.pytorch.org/models/densenet121-a639ec97.pth`

4.4 Discussion

In this section, we first focus on the local minima found by PSG with a toy example to gain a deeper understanding. In this toy example, we train with SGD and PSGD on 2-bit on the MNIST dataset with a fully-connected network consisting of two hidden layers (50, 20 neurons). We show the weight distributions of the two models trained with SGD and PSGD at the first layer. Then, we calculate the eigenvalues of the entire Hessian matrix to analyze the curvature of a local loss surface. In the subsequent sections, we clarify the differences of the purpose of PSGD and that of QAT and analyze why PSGD does not achieve near-FP performance in LP. Also, we demonstrate the potential application of PSG by applying it to a concurrent PTQ method.

4.4.1 Toy Example

Quantized and sparse model

SGD generally yields a bell-shaped distribution of weights which is not adaptable for low bit quantization [90]. On the other hand, PSGD always provides a multi-modal distribution peaked at the quantized values. For this example, three target points are used (2-bit) so the weights are merged into three modes as depicted in Fig. 4.6a. A large proportion of the weights are near zero. Similarly, we note that the sparsity of ResNet-18@W4 shown in Table 4.5 is 72.4% at LP. This is because symmetric quantization also contains zero as the target point. PSGD has nearly the same accuracy with FP ($\sim$96%) at W2A32. However, the accuracy of SGD at W2A32 is about 9%, although the FP accuracy is 97%. This tendency is also shown in Fig. 4.1b, which demonstrates that PSGD reduces the quantization error.

Curvature of PSGD solution

In Sec 4.2.4 and Fig. 4.4, we claimed that PSG finds a minimum with sharp valleys that is more compression friendly, but has a less chance to be found. As the curvature in the direction of the Hessian eigenvector is determined by the corresponding eigenvalue [24], we compare the curvature of solutions yielded by SGD and PSGD by assessing the magnitude of the eigenvalues, similar to [8]. SGD provides minima with relatively wide valleys because it has many near-zero eigenvalues and the similar tendency is observed in [8]. However, the weights trained by PSGD have much more large positive eigenvalues, which means the solution lies in a relatively sharp valley compared to SGD. Specifically, the number of large eigenvalues ($\lambda > 10^{-3}$) in PSGD is 9 times more than that of SGD. From this toy example, we confirm that PSG helps to find the minima which are more compression-friendly (Fig 4.6a) and lie in sharp valleys (Fig. 4.6b) hard to reach by vanilla SGD. we have also used official code[7] of [57] to qualitatively assess the curvature of Fig. 4.6, using the same experimental setting, which is depicted in Fig. 4.7 and it shows a similar tendency. The solution of PSGD is in the more sharp valley than it of SGD.

4.4.2 Weight Distributions

In Table 4.12, we show the classification accuracies and the corresponding mean squared error (MSE) of PSGD depicted in Fig. 4.1. Also, the weight distributions of the model at various layers are shown in Fig. 4.8. In this experiment, we only quantize the weights, not the activations, to compare the performance degradation as weight bit-width decreases. The mean squared errors (MSE) of

[7] `https://github.com/tomgoldstein/loss-landscape`

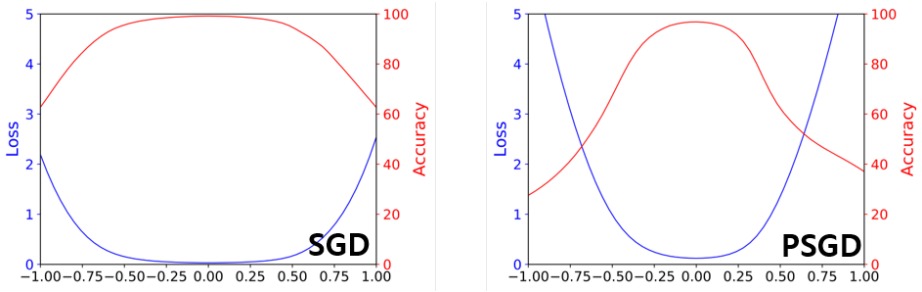


Figure 4.7: Visualizing the loss spaces of Fig. 4.6 using [57]; Left: Loss space of SGD solution; Right: Loss space of PSGD solution.

Table 4.12: 8-, 6-, 4-, 3- and 2-bit weight quantization results of ResNet-32 learned by PSGD on the CIFAR-100 dataset. This is also reported in Fig. 4.1. For lower bit-widths, the mean and standard deviation over five runs are reported. MSE between the learned weight and the target weight is reported. The numbers in the parentheses are the MSEs of SGD.

Bit-width	8-bit	6-bit	4-bit	3-bit	2-bit
Full precision	70.14	70.59	70.08	68.96±0.34	67.16±0.40
Low precision	70.05	70.33	69.57	68.56±0.19	60.76±2.18
MSE (SGD MSE)	0.03 (0.03)	0.41 (0.58)	0.5 (11)	0.84 (48)	67 (157)

the weights across different bit-widths are also reported. The MSE is computed by the squared mean of the differences in full-precision weights and the low-precision weights across layers. As some variance in performance was observed for lower bit-widths, we report the mean±standard deviation for the 2-bit and the 3-bit experiments. As stated, PSGD successfully merges the weights to the target points and obtains quite low MSE until 3-bit and the 2-bit MSE of PSGD is more than 2 times smaller than that of conventional SGD.

In Fig. 4.8, we display the full-precision weight distributions of the PSGD

models and compare them against vanilla SGD-trained distributions. Four random layers of each model are shown column-wise. The first row displays the model trained with SGD and L2 weight decay. Below are distributions trained with PSGD with target points for the sparse training case, 2-bit, 3-bit, and 4-bit respectively. Note that all the histograms are plotted in the full-precision domain, rather than the low-precision domain. For 2-bit, all three target bins ($2^2 - 1$) are visible. For 3-bit, only five target bins are visible as the peripheral two bins contain relatively low numbers of weight components.

4.4.3 Quantization-aware training vs PSGD

Conventional QAT methods [19, 23, 40] starts with a pre-trained model initially trained with SGD and further update the weights by only considering the low precision weights. In contrast, regularization methods such as our work and [2] starts from scratch and update the full-precision weights *analogous to SGD*. In our work, the sole purpose of PSGD is to find a set of full precision weights that are quantization-friendly so that versatile deployment as low precision (LP) is possible without further operation. Therefore, regularization methods start from the initial training phase analogous to SGD, whereas QAT methods starts with a pre-trained model after the initial training phase such as SGD and PSGD. The timeline comparing different quantization training methods is depicted in Fig. 4.9. The purpose of QAT methods is solely focused on LP weights. In general, a coarse gradient is used to update the weights attained by forwarding the LP weights, instead of the FP weights by using the straight-through-estimator (STE) [48]. Additionally, the quantization scheme is modified to include trainable parameters dependent on the low-precision weights and activations. Thus, QAT cannot maintain the performance of full-precision as it only focuses on that of

low-precision such as 4 bit-width.

4.4.4 Post-training with PSGD-trained model

Our model attains similar full-precision performance with SGD and reasonable performance at low-precision even with naive quantization. Thus, PSGD-trained model can be potentially used as a pre-trained model for QAT or PTQ methods. We performed additional experiments using the model trained with PSGD in Table 4.6 by applying a concurrent PTQ work, LAPQ [67], using the official code.[8] This attains 66.5% accuracy for W4A4, which is more than 3.1% and 6.2% points higher than that of PSGD-only and LAPQ-only respectively. This shows that PTQ methods can benefit from using our pretrained model.

4.5 Conclusion

In this work, we introduce the position-based scaled gradient (PSG) which scales the gradient proportional to the distance between the current weight and the corresponding target point. We prove the stochastic PSG descent (PSGD) is equivalent to applying the SGD in the warped space. Based on the hypothesis that DNN has many local minima with similar performance on the test set, PSGD is able to find a compression-friendly minimum that is hard to reach by other optimizers. PSGD can be a key solution to low bit post training quantization, because PSGD reduces the quantization error bridging the discrepancy between the distributions of the compressed and uncompressed weights. Because target points act as a prior to constrain original weights to be merged at specific

[8] https://github.com/ynahshan/nn-quantization-pytorch/tree/master/lapq

positions, PSGD also can be used for sparse training by simply changing the target point as 0. In our experiments, we verify PSGD in the domain of pruning and quantization by showing the effectiveness on various image classification datasets such as CIFAR-10/100 and ImageNet. Also, we empirically show that PSGD finds minima which are located in sharper valleys than SGD. We believe that PSGD will help further researches in model quantization and pruning.

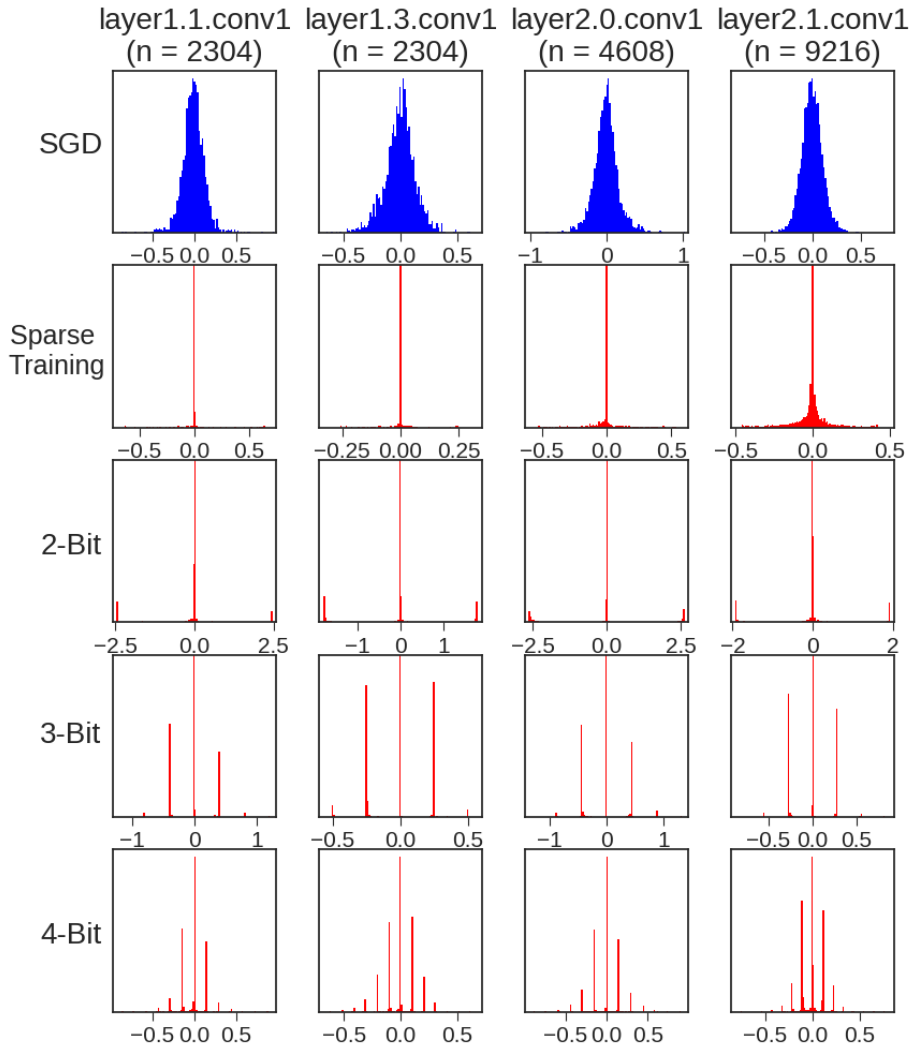


Figure 4.8: Weight distributions in the full-precision domain of four random layers for sparse training, 2-bits, 3-bits, and 4-bits. The name of the layer and the number of parameters in parenthesis are shown in the column. The y-axis and x-axis of PSGD distributions are clipped appropriately for visualization purposes and the number of bins is all set to 100 for both PSGD and SGD.

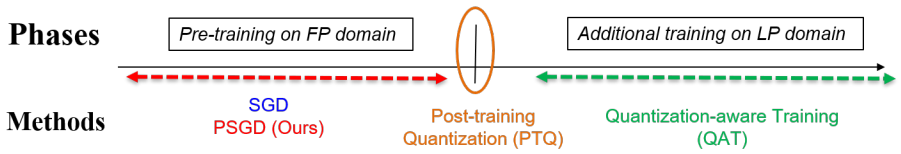


Figure 4.9: The regularization methods such as PSGD and [2] lie in the initial training phase same as SGD. Generally, PTQ and QAT starts after an initial training phase.

Chapter 5

Dynamic Collective Intelligence Learning (DCIL) for Pruning

5.1 Introduction

Massive improvements in various computer vision tasks using deep neural networks have been made at a cost of increase in millions of model parameters. While the task performances look promising, using such over-parameterized neural networks in low-end devices is infeasible due to demanding memory requirements and high latency. To resolve these issues, pruning unimportant units – individual weights (unstructured pruning) and neurons (structured pruning) – of neural networks has been well explored. Following its application to modern neural networks [28], many works have shown success through various pruning criteria in attaining a much sparse network that performs on par with or even better than the original unpruned network [27, 83, 71].

While effective, the aforementioned methods require a finetuning phase after the training phase and use a single fixed sparsity mask to obtain the pruned model. To tackle these problems, some works have focused on pruning methods *during*

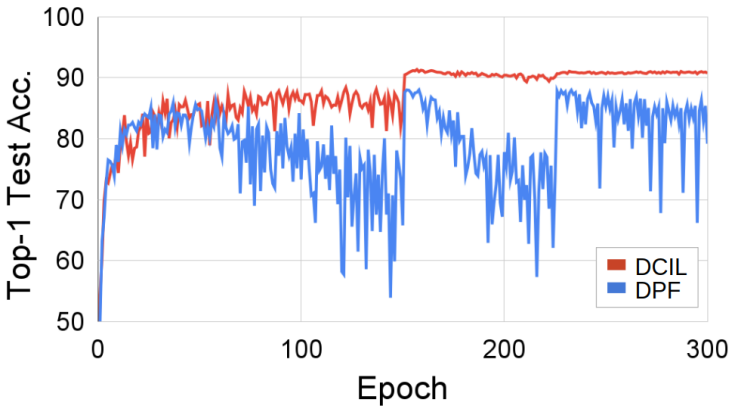


Figure 5.1: Test accuracy vs. epoch with ResNet-20 on CIFAR-10 by 95% pruning. Blue: Dynamic pruning with feedback (DPF) [59], a state-of-the-art dynamic pruning method, Red: Dynamic collective intelligence learning (DCIL; Ours). DPF is unstable and shows a degradation in the performance because of coarse gradients. On the other hand, DCIL utilizes refined gradients to tackle the above issues. More details of DCIL and the stability analysis are stated in the *Method* and *Experiment* sections, respectively.

training with dynamic sparsity patterns. Exploring diverse sparsity patterns is very important as it has a good chance of outperforming algorithms using a single sparsity pattern. These methods called *dynamic pruning* discover sparse masks $(\mathbf{M})$ that are applied to the parameters of the neural network during training. Because fixing the masks discovered in the early training phase may have a detrimental effect on the training process, the masks and hence the sparsity pattern $(\overline{\mathbf{W}} = \mathbf{M} \odot \mathbf{W})$ are designed to dynamically change every few iterations $(\mathbf{F})$ by a certain criterion (e.g. L2 norm) on the real weights $(\mathbf{W})$ as the training progresses.

Trivially, the gradient with respect to the weight of a pruned unit is zero

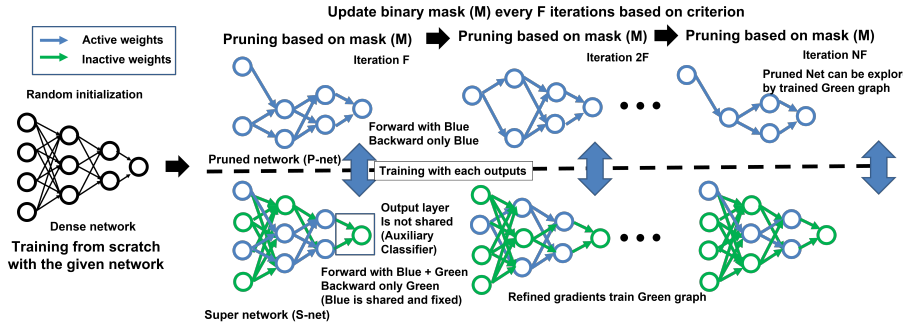


Figure 5.2: The overall process of Proposed DCIL. The blue and green graph refer to active and inactive weights, respectively. At all time, the S-net contains the P-net by sharing the weights of the P-net except a linear classifier and batch norm layers. Refined gradients make the inactive weights good candidates for future inclusion to the active weights.

as the pruned weight does not contribute anything to the output of the neural network nor the loss function ($\frac{\partial L}{\partial w}|_{m=0} = 0$). However, to update the weights of the pruned unit, several works [27, 59] in *dynamic pruning* have resorted to using a form of coarse gradient, in which the derivative of the masked weight with respect to the real weight is simply approximated to 1 (i.e $\frac{\partial \overline{w}}{\partial w}$=1). By doing so, the gradient $\frac{\partial L}{\partial w}$ is approximated by $\frac{\partial L}{\partial \overline{w}}$. This empirical practice has been applied in other literature such as quantization in the name of Straight-Through-Estimator (STE) [4]. The mask $\mathbf{M}$ is updated every $\mathbf{F}$ iterations and some of the inactive (pruned) weights trained with coarse gradients may be revived to active (unpruned) weights. While using this form of coarse gradient is a convenient practice to revive the inactive weights, it may update the weights to an unintended direction harming the training stability and performance. This phenomenon is depicted in Fig. 5.1 and 5.5.

In this work, we show that using coarse gradient to update the inactive weights

can be improved upon by forming a separate path for the inactive weights and updating them separately using an auxiliary loss function. Fig. 5.2 shows the overall process of the proposed method. We form two different networks in size, called Super network (S-net) that has the entire set of real weights ($\mathbf{W}$) and Pruned network (P-net) that only utilizes the masked weights ($\overline{\mathbf{W}}$). We update the inactive weights (Green graph in Fig. 5.2) using the refined gradients of the S-net, whereas the same set of weights in the P-net is masked out to zero and not updated in back-propagation. Updating the inactive weights with refined gradients makes them qualified candidates of active weights for the ensuing training phase that can be readily used without destabilizing the task performance. A similar motivation, in which subsets of the neural network are stochastically selected at training for a better performance, is realized in Dropout [76] to incorporate the effect of model ensemble. In contrary, we update the inactive weights by back-propagation via the forward path of the S-Net for future inclusion in the set of active weights. Note that, the S-net is an unpruned model, whose weights are shared with the P-net, meaning negligible additional memory is needed for maintaining both the S-net and the P-net. In other words, the two networks are subnetworks of the original network whose structure is identical to the S-net. The S-net and P-net collaborate closely to find an efficient sparse network using refined gradients. Based on the learning synergy from the collective intelligence of both networks, we propose a novel sparse training framework named Dynamic Collective Intelligence Learning (DCIL). This allows even the inactive weights to obtain a much meaningful feedback than when using STE, which enhances training stability and achieves higher final performance in dynamic pruning as depicted in Fig. 5.1.

5.2 Proposed method

In this section, we first describe incremental and dynamic pruning. Using incremental pruning, we gradually adjust the pruning sparsity ratio. We explain dynamic pruning method used in [59] to introduce our intuition of refined gradient. Then, we explain our proposed DCIL.

5.2.1 Backgrounds

Incremental pruning One-shot pruning methods usually prune the network and finetune the pruned model for additional epochs to improve the performance. On the other hand, incremental pruning prunes the model each epoch by increasing sparsity ratio based on the current epoch. It is widely used in common pruning methods because it does not need an additional finetuning phase. In this work, we adopt the gradual pruning scheduling proposed in [91] following the setting of [59]:

$$S_c = S_t + (S_i - S_t)(1 - \frac{c - c_0}{n})^3. \tag{5.1}$$

We gradually increase the sparsity ratio from an initial sparsity ratio (S_i at $c_0 = 0$) based on the current epoch c to the target sparsity ratio (S_t). S_c means the current sparsity at c, where $c \in \{c_0, ..., c_0 + n\}$ and n is the number of epochs for the overall training.

Dynamic pruning We consider the binary mask $\mathbf{M} \in \{0, 1\}^P$ for obtaining the sparse weights by masking out the dense weights ($\overline{\mathbf{W}} = \mathbf{M} \odot \mathbf{W}, \mathbf{W} \in \mathbb{R}^P$ where P is the number of parameters and $\odot$ denotes the Hadamard product). A pruned network consists of these sparse weights ($\overline{\mathbf{W}}$). Dynamic pruning updates the binary mask $\mathbf{M}$ every $\mathbf{F}$ iterations based on a certain criterion such as the L2 norm and increases the sparsity with the incremental pruning scheme (5.1). The

update rule of DPF [59] can be written by the following equation for a given loss function $\mathcal{L}$.

$$w_i \leftarrow w_i - \eta \frac{\partial \mathcal{L}}{\partial \overline{w}_i} \frac{\partial \overline{w}_i}{\partial w_i}, \ \ \forall i \in \{1, \ldots, P\} \tag{5.2}$$

where i is the index of weights and mask and $\overline{w}_i = m_i \times w_i$. Sparse weights consist of two mutually exclusive sets which are active (unpruned) ($\{w_i | w_i = \overline{w}_i, m_i = 1\}$) and inactive (pruned) ($\{w_i | w_i \neq \overline{w}_i, \ \overline{w}_i = m_i = 0\}$). Many dynamic pruning methods [27, 59] utilize the straight-through estimator [4, STE] to flow the gradients to the inactive weights. Specifically, if the binary mask prunes the weight (w_i) as zero, the forward path is calculated by $\overline{w}_i = 0$ and the backward path updates w_i. Thus, the gradient $\frac{\partial \overline{w}_i}{\partial w_i}$ of inactive weights is approximated as 1 using STE.

The instability in the test performance curve of DPF can be observed in Fig. 5.1 and Fig. 5.5 as the masking pattern dynamically revives the ill-updated inactive weights. In other words, the inactive weights updated with the STE-approximated gradient are candidates for being active weights based on a certain criterion. When ill-updated weights are selected, the pruned network is not prepared to perform well so it needs some iterations to recover its degraded accuracy. We tackle these issues by refining the gradients in the following section.

5.2.2 Dynamic Collective Intelligence Learning

In this work, we propose Dynamic Collective Intelligence Learning (DCIL) which updates network by using the knowledge and refined gradients from two sub-networks. In the case of inactive weights, the gradient $\frac{\partial \overline{w}}{\partial w}$ should be approximated in order to update the inactive weights (pruned weights). To resolve this problem, DCIL calculates the refined gradient using an auxiliary classifier rather than relying on the approximated gradient ($\frac{\partial L}{\partial \overline{w}}$) by introducing dual forwarding

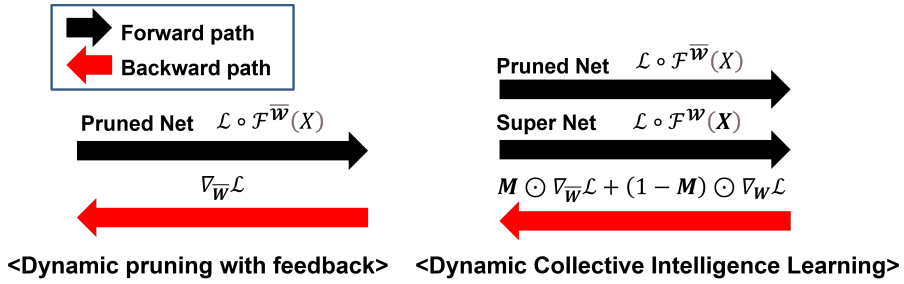


Figure 5.3: Comparison between DPF and DCIL of the forward and backward paths.

paths corresponding to sub-networks. DCIL has two sub-networks, the Pruned network (P-net) and the Super network (S-net). The P-net (Blue graph in Fig. 5.2) consists of weights after pruning ($\overline{\mathbf{W}} = \mathbf{M} \odot \mathbf{W}$) and the S-net (Blue + Green graph in Fig. 5.2) consists of the P-net and the complementary inactive weights ($\mathbf{W} = (\mathbf{1} - \mathbf{M}) \odot \mathbf{W} + \mathbf{M} \odot \mathbf{W}$). Once pruning is done, the weight set of the S-net is a proper superset of that of the P-net (S-net $\supset$ P-net) because a set of inactive weights is a complementary set of active weights building the P-net (inactive weights $=$ active weights^{C}). Note that the S-net contains the P-net and the size of the S-net is equal to that of the unpruned network. The forms of both the P-net and S-net vary every $\mathbf{F}$ iteration based on (5.1) as depicted in Fig. 5.2.

In contrary to the conventional dynamic pruning network, our DCIL does not use STE to update the inactive weights. We introduce a separate auxiliary classifier and batch norm layers for the S-net because the activation statistics are distinct throughout the forward paths of the S-net and P-net. The S-net computes the refined gradient of inactive weights with this auxiliary classifier. In this stage, the inactive weights collaborate with the P-net (active weights) to forward the output of the auxiliary classifier by sharing the weights of the P-net (See

Table 5.1: Top-1 test accuracy of various SOTA pruning methods on **CIFAR-10** for unstructured weight pruning. The numbers of SM, DSR and DPF methods are from [59]. The $\star$ means that the model does not converge. † indicates our numbers. DPF† is from using the official code and we report the last and best accuracy of both DPF† and our DCIL. We also report accuracy of the Dense† models which are not pruned at all. All experiments were conducted three times.

Model	Dense†	Methods							Target Pr. ratio
		SM (DZ,)	DSR (MW,)	DPF (Lin,)	DPF† (Lin,)		DCIL (Ours)		
					Last	Best	Last	Best	
ResNet-20	92.36 ± 0.10	89.76 ± 0.40	87.88 ± 0.04	90.88 ± 0.07	88.02 ± 0.12	90.87 ± 0.25	$\mathbf{91.61} \pm 0.20$	91.93 ± 0.11	90%
		83.03 ± 0.74	$\star$	88.01 ± 0.30	81.46 ± 2.26	87.84 ± 0.11	$\mathbf{90.54} \pm 0.19$	90.80 ± 0.23	95%
ResNet-32	93.22 ± 0.07	91.54 ± 0.18	91.41 ± 0.23	92.42 ± 0.18	91.14 ± 0.12	92.39 ± 0.09	$\mathbf{93.05} \pm 0.15$	93.23 ± 0.06	90%
		88.68 ± 0.22	84.12 ± 0.32	90.94 ± 0.35	86.52 ± 1.38	90.91 ± 0.45	$\mathbf{92.04} \pm 0.24$	92.27 ± 0.27	95%
ResNet-56	94.34 ± 0.19	92.73 ± 0.21	93.78 ± 0.20	93.95 ± 0.11	93.62 ± 0.14	93.97 ± 0.14	$\mathbf{94.16} \pm 0.08$	94.29 ± 0.05	90%
		90.96 ± 0.40	92.57 ± 0.09	92.74 ± 0.08	90.59 ± 0.38	92.81 ± 0.06	$\mathbf{93.75} \pm 0.14$	93.98 ± 0.12	95%
WideResNet-28-2	94.73 ± 0.03	93.41 ± 0.22	93.88 ± 0.08	94.36 ± 0.24	94.08 ± 0.34	94.32 ± 0.22	$\mathbf{94.69} \pm 0.13$	94.84 ± 0.28	90%
		92.24 ± 0.14	92.74 ± 0.17	93.62 ± 0.05	93.13 ± 0.24	93.61 ± 0.08	$\mathbf{94.01} \pm 0.21$	94.21 ± 0.04	95%
		85.36 ± 0.80	$\star$	88.92 ± 0.29	85.82 ± 0.45	88.77 ± 0.23	$\mathbf{91.19} \pm 0.16$	91.35 ± 0.24	99%

Fig. 5.2). The refined gradient helps the inactive weights to be prepared for the future inclusion to the P-net because the auxiliary classifier computes the refined gradient by forwarding with real weights ($\mathbf{W}$) rather than masked weights ($\overline{\mathbf{W}}$). In doing so, the inactive weights can be updated using the refined gradients with actual values of inactive weights. When the binary mask $\mathbf{M}$ changes every $\mathbf{F}$ iterations, inactive weights trained with refined gradients become better qualified candidates of the P-Net, which have better performance and stability than the inactive weights trained with approximated gradients with STE for active weights.

Let $\mathcal{F}^{\mathbf{W}} : \mathcal{X} \rightarrow \mathcal{Y}$ and $\mathcal{L} : \mathcal{Y} \rightarrow \mathbb{R}$ be the forwarding path with given weights ($\mathbf{W}$) and the loss function, respectively. Then, we can define the update rule of DCIL as follows:

Table 5.2: Top-1 test accuracy of our DCIL and DPF† on **CIFAR-100** for unstructured weight pruning. † indicates our numbers. DPF† indicates our numbers using the official code and all reported numbers are averaged over three times.

Model	Dense†	DPF†		DCIL (Ours)		Target Pr. ratio
		Last	Best	Last	Best	
ResNet-20	67.81 ± 0.54	53.7 ± 1.95	63.94 ± 0.02	$\mathbf{67.12} \pm 0.06$	67.51 ± 0.10	90%
		46.15 ± 0.16	58.17 ± 0.46	$\mathbf{64.10} \pm 0.53$	64.68 ± 0.37	95%
ResNet-32	69.50 ± 0.20	61.12 ± 1.73	67.82 ± 0.26	$\mathbf{69.96} \pm 0.28$	70.40 ± 0.36	90%
		50.08 ± 6.10	63.22 ± 0.39	$\mathbf{67.90} \pm 0.14$	68.53 ± 0.16	95%
ResNet-56	74.23 ± 0.13	67.99 ± 1.15	72.99 ± 0.92	$\mathbf{74.59} \pm 0.36$	74.95 ± 0.24	90%
		56.88 ± 0.85	69.83 ± 0.42	$\mathbf{73.30} \pm 0.34$	73.63 ± 0.39	95%
WideResNet-28-2	74.81 ± 0.15	71.32 ± 0.59	73.45 ± 0.25	$\mathbf{74.61} \pm 0.02$	74.97 ± 0.13	90%
		68.56 ± 0.50	71.69 ± 0.36	$\mathbf{73.70} \pm 0.23$	74.05 ± 0.13	95%
		51.27 ± 0.33	60.02 ± 0.04	$\mathbf{65.55} \pm 0.18$	66.00 ± 0.10	99%

$$\begin{aligned} \mathbf{W} &\leftarrow \mathbf{W} - \eta\{\mathbf{M} \odot \nabla_{\overline{\mathbf{W}}}\mathcal{L} + (1-\mathbf{M}) \odot \nabla_{\mathbf{W}}\mathcal{L}\} \\ \nabla_{\overline{\mathbf{W}}}\mathcal{L} &\triangleq \frac{\partial \mathcal{L} \circ \mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X})}{\partial \overline{\mathbf{w}}}, \quad \nabla_{\mathbf{W}}\mathcal{L} \triangleq \frac{\partial \mathcal{L} \circ \mathcal{F}^{\mathbf{W}}(\mathbf{X})}{\partial \mathbf{w}}. \end{aligned} \tag{5.3}$$

In this setting, note that the composite functions $\mathcal{L} \circ \mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X})$ and $\mathcal{L} \circ \mathcal{F}^{\mathbf{W}}(\mathbf{X})$ correspond to the model with different classifiers of which the latter uses the auxiliary classifier. Fig. 5.3 shows the forward and backward paths according to each algorithm. DPF [59] updates $\mathbf{W}$ with the gradient of $\overline{\mathbf{W}}$ by the approximation using STE ($\nabla_{\overline{\mathbf{W}}}\mathcal{L}$). On the other hand, DCIL updates $\mathbf{W}$ with gradients of $\overline{\mathbf{W}}$ and the refined gradients of inactive weights without an approximation ($\mathbf{M} \odot \nabla_{\overline{\mathbf{W}}}\mathcal{L} + (1-\mathbf{M}) \odot \nabla_{\mathbf{W}}\mathcal{L}$). Generally, the cross entropy term $\mathcal{L}_{ce}$ is used for the loss function. However, we want separate paths to share more information then collaborate better with each other. Since DCIL has two forward paths with two classifiers, DCIL can expand the cross entropy loss to Knowledge distillation (KD) loss [35, 44] by using each output as a soft target for each Kullback–Leibler

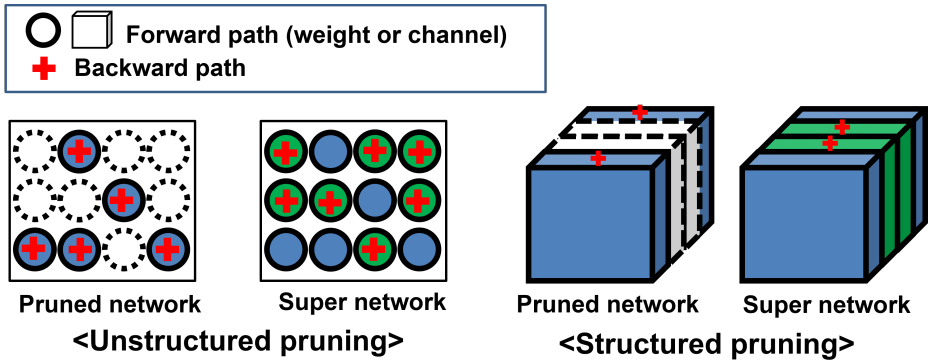


Figure 5.4: Diagram of applying DCIL to two pruning types, unstructured pruning and structured pruning.

(KL) divergence term ($KL(p||q;\mathcal{T})$) with a temperature value $\mathcal{T}$:

$$\begin{aligned}\mathcal{L}\circ\mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X}) &= \mathcal{L}_{ce}\circ\mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X}) + \lambda\mathcal{T}^2\cdot KL(\mathcal{F}^{\mathbf{W}}(\mathbf{X})||\mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X});\mathcal{T})\\ \mathcal{L}\circ\mathcal{F}^{\mathbf{W}}(\mathbf{X}) &= \mathcal{L}_{ce}\circ\mathcal{F}^{\mathbf{W}}(\mathbf{X}) + \lambda\mathcal{T}^2\cdot KL(\mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X})||\mathcal{F}^{\mathbf{W}}(\mathbf{X});\mathcal{T})\end{aligned} \tag{5.4}$$

where λ is a hyper-parameter for balancing loss functions. We compare the effect of the KL loss and the cross entropy loss in our ablation study.

DCIL can be applied regardless of the pruning type, i.e, it can be applied to both unstructured or structured pruning. In the unstructured pruning case, inactive and active weights coexist in a filter so the pruning is done in the weight-level whereas DCIL of structured pruning operates in the filter-level (neuron) as shown in Fig. 5.4. We set the mask frequency $\mathbf{F}$ as 16 and use the L2 magnitude for the pruning criterion. Algorithm 1 shows the pseudocode of the proposed DCIL. At every $\mathbf{F}$ iterations, binary mask $\mathbf{M}$ is updated based on Eq. 5.1 and the pruning criterion. During the update of the dynamic mask, inactive weights which are trained with refined gradients can be altered into active weights. In this process, prepared candidates trained with refined gradients maintain the training stability.

5.2.3 Convergence analysis

Theorem 3. *Suppose optimal solution w^* that minimizes the convex loss f exists and the approximated gradient $g(w)$ used to update DCIL has limited power ($\mathbb{E}\|g(w)\|^2 \leq G^2$) for all w, then the loss for the uncompressed model of DCIL (S-net) is upper bounded by:*

$$\mathbb{E}\{\frac{1}{T}\sum_{t=1}^{T} f(w_t)\} \leq f(w^*)\frac{R^2}{2\eta T} + \frac{\eta G^2}{2} + \frac{R}{T}\sum_{t=1}^{T}\mathbb{E}\|\nabla f_t - g_t\|. \tag{5.5}$$

where $R^2 = \mathbb{E}\|w_1 - w^*\|^2$ and ∇_f is the exact gradient of S-net.

Proof. Let's first consider w_t is fixed. Then it becomes

$$\begin{aligned}
&\mathbb{E}\{\|w_{t+1} - w^*\|^2 | w_t\} = \mathbb{E}\{\|w_t - \eta g_t - w^*\|^2 | w_t\} \\
&= \mathbb{E}\{\|w_t - w^*\|^2 + \eta^2\|g_t\|^2 - 2\eta(g_t - \nabla f_t + \nabla f_t)^T(w_t - w^*)|w_t\} \\
&\leq \|w_t - w^*\|^2 + \eta^2 G^2 + 2\eta(\nabla f_t - g_t)^T(w_t - w^*) - 2\eta(f(w_t) - f(w^*))
\end{aligned} \tag{5.6}$$

where ∇f_t is the exact gradient at time t and the last term is from the convexity of f. Taking expectation over w_t,

$$\begin{aligned}
\mathbb{E}\|w_{t+1} - w^*\|^2 &\leq \mathbb{E}\|w_t - w^*\|^2 + \eta^2 G^2 - 2\eta(\mathbb{E}\{f(w_t)\} - f(w^*)) \\
&\qquad + 2\eta\mathbb{E}\{\|\nabla f_t - g_t\|\}\mathbb{E}\{\|w_t - w^*\|\} \\
&\leq \mathbb{E}\|w_1 - w^*\|^2 + t\eta^2 G^2 - 2\eta\sum_{k=1}^{t}(\mathbb{E}\{f(w_k)\} - f(w^*)) \\
&\quad + 2\eta\sum_{k=1}^{t}\mathbb{E}\{\|\nabla f_k - g_k\|\}\mathbb{E}\{\|w_k - w^*\|\}.
\end{aligned} \tag{5.7}$$

Reordering terms, it becomes

$$2\eta\sum_{k=1}^{t}(\mathbb{E}\{f(w_k)\} - f(w^*)) \leq R^2 + t\eta^2 G^2 + 2\eta\sum_{k=1}^{t}\mathbb{E}[\epsilon_k]\mathbb{E}\|w_k - w^*\| \tag{5.8}$$

where $\epsilon_k = \|\nabla f_k - g_k\|$. By the convexity of f it becomes

$$\mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} f(w_t)] - f(w^*) \leq \frac{R^2}{2\eta T} + \frac{\eta G^2}{2} + \frac{R}{T}\sum_{t=1}^{T}\mathbb{E}[\epsilon_t]. \tag{5.9}$$

□

This analysis applies to uncompressed models of both DPF (DPF-U) and our DCIL (S-net). Because the first two terms in the right side are identical for both models, the key difference between the two models is the magnitude of ϵ_t. While both DPF-U and S-net update their masked surviving weights with a different stochastic gradient $M \odot g$ than the true gradient $M \odot \nabla f$, the pruned weights are updated differently, i.e, for DPF-U, $\bar{M} \odot g \neq \bar{M} \odot \nabla f$, but S-net uses exact gradient $\bar{M} \odot g = \bar{M} \odot \nabla f$. Therefore, for the same w_t, $\epsilon_t^{DPF} \geq \epsilon_t^{DCIL}$ and DPF-U is always inferior to S-net, thus DCIL shows a better convergence behavior.

5.3 Experiments

We compare our DCIL with the state-of-the-art methods on commonly used datasets. Our model significantly improves the performance and the stability in both unstructured and structured pruning regardless of the architecture.

We also show the stability analysis and the effect of the KL divergence through an ablation study. As a result of the ablation study, we demonstrate that updating the inactive weights with refined gradients plays a key role in improving performance and training stability than using KL divergence.

Algorithm 1 Dynamic Collective Intelligence Learning (DCIL)

Input: Mask update frequency $\mathbf{F}$, Binary mask $\mathbf{M} \in \{0,1\}^P$, S-Net $\mathbf{W} \in \mathbb{R}^P$, P-Net $\overline{\mathbf{W}} = \mathbf{M} \odot \mathbf{W}$ ▷ P is the number of parameters and $\odot$ denotes the Hadamard product

1: **[Train]**

2: **for** Iter = 1 ,..., T **do**

3: **if** Iter $\%$ $\mathbf{F} == 0$ **then**

4: compute binary mask $\mathbf{M}$ with Eq.(1) and the pruning criterion ▷ Update mask every $\mathbf{F}$ iterations

5: **end if**

6: $\mathbf{W} \leftarrow \mathbf{W} - \eta\{\mathbf{M} \odot \nabla_{\overline{\mathbf{W}}}\mathcal{L} + (1-\mathbf{M}) \odot \nabla_{\mathbf{W}}\mathcal{L}\}$ ▷ Update weights which are forwarded with S-Net and P-Net

7: **end for**

Output: S-Net, P-Net ▷ P-Net is the efficient network trained from DCIL

8: **[Test]**

9: Inference with P-Net

5.3.1 Experiment Setting

Datasets

We conduct experiments on three image classification datasets CIFAR-10, CIFAR100 [49], and ImageNet [73].

Baselines

We compare our method against DPF [59], SM [17], DSR [64] and SFP [32]. Please refer to the section on *Related Work* for the details.

Table 5.3: Top-1 and Top-5 test accuracy of ResNet-18 and ResNet-50 on **ImageNet**. DPF† indicates our numbers using the official code. We report the best accuracy and the accuracy from the last epoch. Difference refers to difference between pruned and dense accuracy ('Last - Dense'). Best accuracy of Top-5 is selected based on Top-1 best epoch.

Model	Method	Top-1 accuracy				Top-5 accuracy				Target Pr. ratio
		Dense	Last	Best	Difference	Dense	Last	Best	Difference	
ResNet-18	DPF† (Lin,)	70.50	69.08	69.19	-1.31	89.54	88.93	88.88	-0.66	80%
	DCIL (Ours)	70.50	**69.57**	69.62	-0.88	89.54	**89.26**	89.29	-0.25	80%
	DPF† (Lin,)	70.50	67.37	67.66	-2.83	89.54	87.83	87.88	-1.66	90%
	DCIL (Ours)	70.50	**68.66**	68.66	-1.84	89.54	**88.57**	88.57	-0.97	90%
	DCIL w/o KL (Ours; $\lambda = 0$)	70.50	68.37	68.41	-2.09	89.54	88.32	88.26	-1.28	90%
ResNet-50	Incremental (ZG,)	75.95	–	74.25	-1.70	92.91	–	91.84	-1.07	80%
	DSR (MW,)	74.90	–	73.30	-1.60	92.40	–	92.40	0	80%
	SM (DZ,)	75.95	–	74.59	-1.36	92.91	–	92.37	-0.54	80%
	DPF (Lin,)	75.95	–	75.48	-0.47	92.91	–	92.59	-0.32	80%
	DPF† (Lin,)	76.06	75.59	75.61	-0.47	92.85	92.66	92.71	-0.19	80%
	DCIL (Ours)	76.06	**76.16**	76.17	0.10	92.86	**92.94**	93.00	0.09	80%
	Incremental (ZG,)	75.95	–	73.36	-2.59	92.91	–	91.27	-1.64	90%
	DSR (MW,)	74.90	–	71.60	-3.30	92.40	–	90.50	-1.90	90%
	SM (DZ,)	75.95	–	72.65	-3.30	92.91	–	91.26	-1.65	90%
	DPF (Lin,)	75.95	–	74.55	-1.44	92.91	–	92.13	-0.78	90%
	DPF† (Lin,)	76.06	74.39	74.48	-1.67	92.85	92.08	92.05	-0.77	90%
	DCIL (Ours)	76.06	**75.34**	75.40	-0.72	92.85	**92.58**	92.63	-0.27	90%
	DCIL w/o KL (Ours; $\lambda = 0$)	76.06	74.75	74.87	-1.31	92.85	92.31	92.24	-0.54	90%

Implementation Details

For a fair comparison, we followed the same experimental settings with DPF [59] for all experiments. We set the mask update frequency (**F**) to 16 iterations like DPF. We applied our pruning method and other pruning methods to ResNet [31] and WideResNet [87]. Following DPF, we performed pruning across all convolutional layers except for the batch normalization layers and the last fully connected layer. We used SGD with the Nesterov momentum of 0.9. For CIFAR, the ResNet models were trained for 300 epochs with the initial learning rate of 0.2 and the L2 weight decay parameter of 1e-4. We decayed the learning rate by

Table 5.4: Top-1 test accuracy of our DCIL and other baseline methods on **CIFAR-10** for structured weight pruning. † indicates our numbers. We ran DPF† using the official code and official training schedule. Note that for a given pruning ratio DCIL and DPF prune filters across layers, meanwhile SFP prunes filters within the layer. Furthermore, we report the differences ('Pruned - Dense') in the Appendix. When comparing the differences, DCIL outperform with the higher gap in most cases. All reported numbers are averaged over three times.

Model	Dense†	SFP (H^+,)	DPF (Lin,)	DPF†		DCIL (Ours)		Target Pr. ratio
				Last	Best	Last	Best	
ResNet-32	93.22 ± 0.07	92.07 ± 0.22	92.18 ± 0.16	91.61 ± 0.15	91.81 ± 0.19	$\mathbf{93.02} \pm 0.21$	93.10 ± 0.21	30%
		91.14 ± 0.45	91.50 ± 0.21	90.01 ± 0.40	90.28 ± 0.40	$\mathbf{92.61} \pm 0.07$	92.76 ± 0.16	40%
WideResNet-28-2	94.73 ± 0.03	94.02 ± 0.24	94.52 ± 0.08	94.11 ± 0.17	94.30 ± 0.09	$\mathbf{94.55} \pm 0.04$	94.61 ± 0.08	40%
		86.00 ± 1.09	90.53 ± 0.17	88.99 ± 0.45	89.76 ± 0.64	$\mathbf{92.10} \pm 0.11$	92.32 ± 0.15	80%
WideResNet-28-4	95.50 ± 0.07	95.15 ± 0.11	95.50 ± 0.05	95.36 ± 0.16	95.52 ± 0.14	95.46 ± 0.06	95.59 ± 0.06	40%
		91.88 ± 0.59	93.79 ± 0.09	93.41 ± 0.07	93.51 ± 0.11	$\mathbf{94.18} \pm 0.04$	94.38 ± 0.08	80%
WideResNet-28-8	95.93 ± 0.07	95.62 ± 0.04	96.06 ± 0.12	95.86 ± 0.17	96.03 ± 0.18	95.73 ± 0.07	95.91 ± 0.10	40%
		94.22 ± 0.21	95.15 ± 0.03	94.97 ± 0.20	95.16 ± 0.12	$\mathbf{95.43} \pm 0.09$	95.49 ± 0.05	80%

10 at 150 and 225 epochs. WideResNet models were trained for 200 epochs with the initial learning rate of 0.1 which is decayed by 5 at 60, 120 and 160 epochs. The weight decay parameter was set to 5e-4. We used 128 mini-batch size for all experiments of CIFAR. In the ImageNet training, we trained models with 90 epochs and decayed the learning rate by 10 at 30, 60 and 80 epochs with the initial learning rate of 0.1. We used 128 mini-batch size for ResNet-18 and 1,024 mini-batch size for ResNet-50 following [25]. We introduce the warm-up epoch, an early period of learning without KL loss but only using cross-entropy ($\lambda = 0$), because regularizing the model with KL loss using an untrained teacher model may hinder convergence. We set the warm-up epoch to 70 and 10 for CIFAR and ImageNet datasets, respectively. However, we confirmed that the scale of warm-up epoch does not significantly affect the test accuracy as shown in the

Appendix. We set the $\lambda = 1$ and $\mathcal{T} = 2$ for the KD loss. We used the Pytorch framework for all experiments. All experiments on CIFAR and the experiments of ResNet-18 on ImageNet were conducted with a GeForce GTX 1080 Ti GPU. The experiments of ResNet-50 on ImageNet were conducted with four NVIDIA RTX A6000 GPUs.

5.3.2 Experiment Results

In all experiments, we report the numbers with same target pruning ratio from the original paper of DPF [59] and used the same L2 pruning criterion as DPF. We also re-implemented DPF referred as DPF† to compare training stability. We report the test accuracy of the last and the best epoch of DPF and DCIL. Our last epoch accuracy is made bold if our number is higher than those of all the methods.

Unstructured

Table 5.1 shows the top-1 test accuracy of various SOTA pruning methods on the CIFAR-10 dataset. Along with the accuracy reported in the original papers of each method, we report the performance of DPF we re-implemented. DCIL significantly outperforms all other methods for all models, datasets, and a variety of pruning sparsities. In particular, the test accuracy in the last epoch of DCIL is 1-3% higher than the best accuracy of re-implemented DPF which is nearly the same as the performance of the original paper. DCIL has a far superior performance on the CIFAR-100 dataset as shown Table 5.2. The last accuracy of DCIL outperforms the last accuracy of DPF by 3-18%, the best accuracy of DPF by 1-6%, and sometimes the performance of the dense model due to the regularization effect. In addition, the training instability of DPF is severe in the

CIFAR datasets on unstructured pruning, resulting in a very large difference between the last and the best accuracies as shown in Table 5.1 and Table 5.2. On the other hand, our DCIL has a stable training curve, demonstrated by the small margin between the last accuracy and the best accuracy. And as a result, in the absence of an additional fine-tuning phase or validation sets, DCIL has a good chance to work as good as the optimal pruned model even on datasets where training can be relatively unstable.

Also, we conducted ImageNet experiments to verify the effectiveness of DCIL in large-scale datasets as shown in Table 5.3. DCIL outperforms other pruning methods with a large margin in test accuracy. In ResNet-18 with 90% pruning, the performance gap between DPF and DCIL is around 1.3%. The tendency of stability during training is similar to that of CIFAR, which is depicted in Appendix. We also report the results of ablation studies with respect to the KL divergence term. 'DCIL w/o KL' in Table 5.3 means a model trained with DCIL by only cross-entropy. KL helps improve the performance by a range of 0.3-0.6%. However, DCIL without KL also surpasses DPF by about 0.4-1%, which shows the effectiveness of the refined gradients. A similar tendency is observed in the following stability analysis section.

Structured

To show a broad applicability of our algorithm, we applied DCIL at the filter level. Unlike unstructured pruning, filter pruning or *structured pruning* could take full advantage of the BLAS library and accelerate model inference.

Table 5.4 shows a comparison of different methods on CIFAR-10 for ResNet and WideResNet variants. Both DPF and SFP dynamically allocate sparsity pattern, updating the filter with a coarse gradient ($\frac{\partial L}{\partial \widetilde{w}}$). DCIL outperforms all the

Table 5.5: Standard Deviation of the top-1 test accuracy over the last 10% epochs. To compare the training stability of our DCIL and DPF on **CIFAR-10**, we report the standard deviation of the top-1 test accuracy over the last 10% epochs (i.e. 30 epochs for ResNet and 20 epochs for WideResNet), which is expected to be stable with training for several epochs after the last learning rate decay. The results are averaged over three runs.

Model	CIFAR-10			Target Pr.ratio
	Dense	DPF	DCIL (Ours)	
ResNet-20	0.070 ± 0.012	1.179 ± 0.310	0.072 ± 0.013	90%
		3.839 ± 1.455	0.114 ± 0.007	95%
ResNet-32	0.061 ± 0.013	0.344 ± 0.036	0.080 ± 0.005	90%
		1.947 ± 0.362	0.097 ± 0.011	95%
ResNet-56	0.060 ± 0.009	0.179 ± 0.055	0.063 ± 0.007	90%
		0.990 ± 0.078	0.082 ± 0.009	95%
WideResNet-28-2	0.061 ± 0.004	0.120 ± 0.033	0.093 ± 0.011	90%
		0.286 ± 0.067	0.086 ± 0.015	95%
		2.790 ± 0.723	0.096 ± 0.007	99%

baselines in most settings except for WideResNet-28-8 (40% sparsity), where DCIL is comparable to the official results of DPF. DCIL's superior performance for structured pruning can be seen in terms of architecture search as structured pruning is considered as performing implicit architecture search [60]. DCIL searches for better subnetwork candidates obtained through the refined gradients, whereas existing methods using coarse gradients could be updated in a direction that is harmful to the performance and reach a suboptimal solution.

We also report the full results on CIFAR-100 with other structured pruning methods [32, 18, 33]. Briefly, when compared with the existing methods for

ResNet-56 (40% and 50% sparsity), DCIL shows higher accuracies in all settings. This result validates the effectiveness of our DCIL's refined gradient for structured pruning. The details of structured pruning is same as above settings. Table 5.6 shows the numbers of structured pruning experiments on CIFAR-100.

Table 5.6: Top-1 test accuracy of our DCIL and other baseline methods on CIFAR-100 for structured weight pruning. The same training scheme was used as on CIFAR-10. To compare the stability of pruning methods we reported the accuracy of the last epoch and the best accuracy of DCIL.

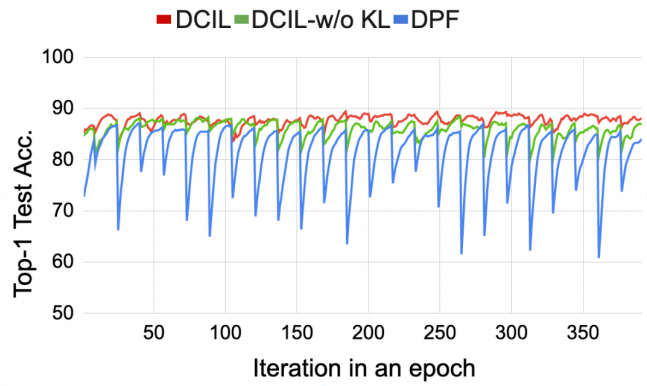
Model	DCIL (Ours)				SFP		MIL		FPGM		Target Pr. ratio
	Dense	Last	Best	Diff.	Dense	Diff.	Dense	Diff.	Dense	Diff.	
ResNet-56	74.23 ± 0.13	73.49 ± 0.16	73.80 ± 0.23	$\mathbf{-0.74}$	71.4	–	71.33	-2.96	71.41	–	40%
		73.17 ± 0.04	73.59 ± 0.15	$\mathbf{-1.06}$		-2.61		–		-1.75	50%

5.3.3 Differences between Dense and pruned model

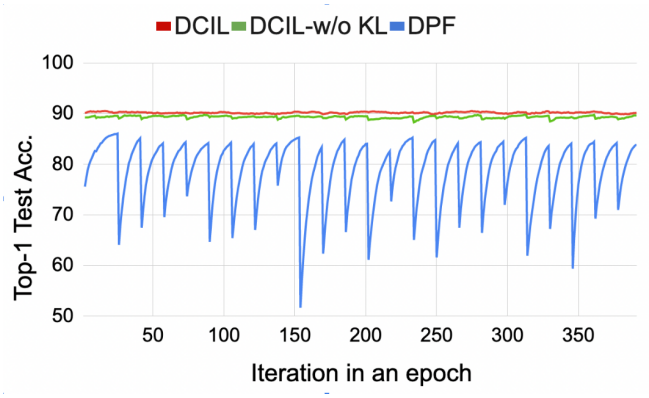
In this section, we provide the performance differences between the dense model and the pruned model ('Pruned - Dense') to compare the degree of the performance degradation. Table 5.7 and Table 5.8 show the performance degradation according to the experiments in the main experiment.

5.3.4 Analysis of the stability

Table 5.5 summarizes the standard deviation of top-1 test accuracy of ours and DPF over the last 30 and 20 epochs, which is the last 10% epochs for ResNet and WideResNet, respectively. Despite the low learning rate since it is after the last decaying epoch, DPF has a large variance in top-1 test accuracy, while our method has a much smaller variance. We conjecture that this is due to the differences in the gradients used to update the inactive weights, i.e, our method



(a) 120th epoch



(b) 200th epoch

Figure 5.5: Top-1 test accuracy of every iteration in an epoch. To show the effectiveness of changes in active weights caused by changes in pruning masks, we evaluated ResNet-20 pruned with 95% sparisty using Red: DCIL, Green: DCIL without KL loss, and Blue: DPF on CIFAR10 every iteration at (a) 120th epoch and (b) 200th epoch. The accuracy of all methods shows a repetitive pattern with a pruning frequency (**F**) of 16.

updates inactive weights using refined gradients calculated on S-net, whereas DPF approximately estimate gradients that can lead to a degradation of the pruned model.

Table 5.7: Top-1 test accuracy *differences* ('Pruned - Dense') of our DCIL and other baseline methods on CIFAR-10 for unstructured pruning. $\star$ means that the model does not converge. † indicates our numbers while the unmarked others are calculated directly from the DPF paper. We ran DPF† using the official code and official training schedule and all reported numbers are averaged over three times.

Model	Dense	SM	DSR	DPF	Dense†	DPF†		DCIL (Ours)		Target Pr.
						Last	Best	Last	Best	
ResNet-20	92.48 ± 0.20	-2.72	-4.60	-1.6	92.36 ± 0.10	-4.34	-1.49	$\mathbf{-0.75}$	-0.43	90%
		-9.45	$\star$	-4.47		-10.9	-4.52	$\mathbf{-1.82}$	-1.56	95%
ResNet-32	93.83 ± 0.12	-2.29	-2.42	-1.41	93.22 ± 0.07	-2.08	-0.83	$\mathbf{-0.17}$	0.01	90%
		-5.15	-9.71	-2.89		-6.7	-2.31	$\mathbf{-1.18}$	-0.95	95%
ResNet-56	94.51 ± 0.20	-1.78	-0.73	-0.56	94.34 ± 0.19	-0.72	-0.37	$\mathbf{-0.18}$	0.05	90%
		-3.55	-1.94	-1.77		-3.75	-1.53	$\mathbf{-0.59}$	-0.36	95%
WideResNet-28-2	95.01 ± 0.04	-1.6	-1.13	-0.65	94.73 ± 0.03	-0.65	-0.41	$\mathbf{-0.04}$	0.11	90%
		-2.77	-2.27	-1.39		-1.6	-1.12	$\mathbf{-0.72}$	-0.54	95%
		-9.65	$\star$	-6.09		-8.91	-5.96	$\mathbf{-3.54}$	-3.38	99%

Table 5.8: Top-1 test accuracy *differences* ('Pruned - Dense') of our DCIL and other baseline methods on CIFAR-10 for structured pruning. The settings and symbols are the same as in Table 5.7

Model	Dense	SFP	DPF	Dense†	DPF†		DCIL (Ours)		Target Pr.
					Last	Best	Last	Best	
ResNet-32	93.52 ± 0.13	-1.34	-1.45	93.22 ± 0.07	-1.61	-1.40	$\mathbf{-0.19}$	-0.12	30%
		-2.38	-2.02		-3.20	-2.93	$\mathbf{-0.61}$	-0.4	40%
WideResNet-28-2	95.01 ± 0.04	-0.99	-0.49	94.73 ± 0.03	-0.63	-0.43	$\mathbf{-0.19}$	-0.13	40%
		-9.01	-4.48		-0.91	-0.74	$\mathbf{-0.52}$	-0.43	80%
WideResNet-28-4	95.69 ± 0.10	-0.54	-0.19	95.50 ± 0.07	-0.14	0.01	-0.04	0.09	40%
		-3.81	-1.9		-2.09	-1.99	$\mathbf{-1.32}$	-1.13	80%
WideResNet-28-8	96.06 ± 0.06	-0.44	0.00	95.93 ± 0.07	-0.06	0.10	-0.20	-0.01	40%
		-1.84	-0.91		-0.96	-0.77	$\mathbf{-0.50}$	-0.44	80%

In order to demonstrate the effect of using refined gradient to dynamically allocate sparsity masks on performance directly, we evaluated the pruned model

using DCIL, DCIL without KL, and DPF at every iteration as in Fig 5.5. Fig 5.5 shows the top-1 test accuracy of all iterations throughout one epoch (e.g. 120, 200) during training. In the 120th epoch before the first learning rate decaying, all the methods have a cyclic pattern every 16 iterations due to the pruning frequency of 16. However, there is a clear difference in the scale of performance degradation between DCIL-based methods and DPF. In DPF, the test accuracy decreases drastically by 10-20% after the pruning mask changes and restores performance before the next pruning mask iteration, whereas DCIL has a stable pattern with much less performance reduction of around 2-5%. Performance reduction of DCIL without KL is around 3-7% which is greater than that of DCIL with KL, but the gap is also much smaller than DPF. In the 200th epoch, the difference becomes more pronounced as training progresses using the refined gradient. Although DPF still suffers significant performance degradation each time the pruning mask changes, our DCIL exhibits a much more stable pattern similar to the dense models, because refined gradients make inactive weights as prepared candidates for active weights. The results of structured pruning shows a similar tendency with that of unstructured pruning. We provide the overall Top-1 test accuracy vs epoch in Fig. 5.6 and Fig. 5.7.

5.3.5 Cost of training

One limitation of DCIL in the current implementation is that the model requires two forward and backward paths through the P-Net and the S-Net, taking around 1.5x wall-clock time for training than DPF. Nonetheless, with an efficient implementation, the cost of backward paths can be significantly reduced, because essentially the same number of weights (inactive + active) requires update like DPF. Some relevant methods [81, 77] utilizing sparse gradients have been pro-

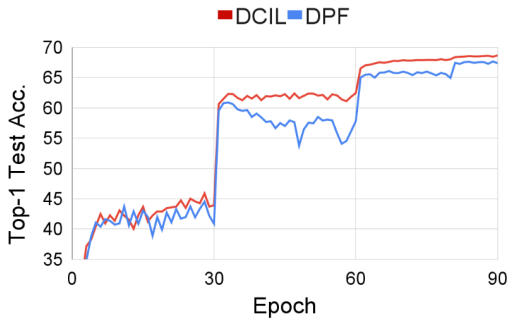
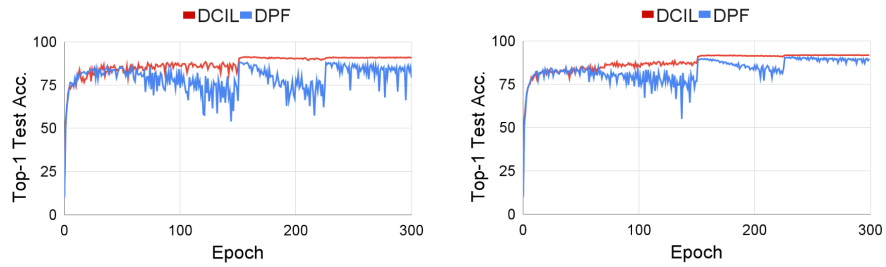


Figure 5.6: Training stability, unstructured pruning with 90%, ResNet-18 on Imagenet.



(a) Unstructured pruning

(b) Structured pruning

Figure 5.7: Top-1 test accuracy vs. epoch to show the training stability. ResNet-20 with 95% pruning on CIFAR-10 dataset, using unstructured and structured pruning.

posed to reduce computation in the backward phase. Note that the backward path accounts for the majority of the computation requiring around 2-3x more time than the forward path for residual networks [22]. Most importantly, a better performing model with the same number of parameters can be obtained *for inference* with our method.

Interestingly, even in the current setting, our method achieves on par or superior performance than the SOTA method with equal computation cost due to the faster convergence and training stability. For instance, in CIFAR-10 ResNet-20 for target sparsity 95%, DPF achieves 90.34% with additional finetuning epochs to improve performance (300+60 epochs), whereas ours achieves 90.93% at 240 epoch, two thirds of the 360 epochs of DPF.

5.3.6 Fast convergence of DCIL

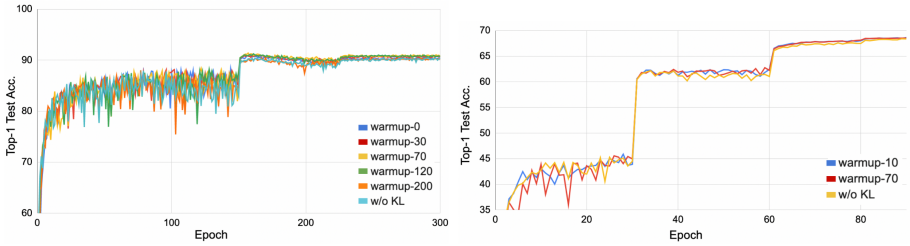
In the *Cost of training* subsection, we noted that the limitation of DCIL is that the model takes around 1.5x wall-clock time for training than DPF. To address this limitation, we suggested that training could be stopped earlier due to the training stability, citing that the accuracy of DCIL at the 240th epoch is higher than that of DPF which is fine-tuned for 60 epochs additionally. Moreover, we can finish the training much earlier by adjusting the target epoch, the epoch at which the sparsity of the model reaches the target sparsity. We additionally conducted experiments by adjusting the target epochs smaller than 225 used in the main experiment, and the learning rate decaying epochs were also adjusted so that the second learning rate decaying epochs are the same as the target epochs. Table 5.9 compares the accuracy and training time between DCIL with reduced training time and DPF which is fine-tuned. For DCIL, we reduced the target epoch to 100, 125, 150, 175, 200 and trained only 25 epochs additional after that target epoch. Since DCIL is much more stable in training, it has higher performance with only 0.6-0.7 times the training time compared to DPF.

Table 5.9: Top-1 test accuracy on CIFAR10 dataset in unstructured pruning of target sparsity 95%. For DCIL, we reduced the training time by adjusting the target epoch and training only 25 additional epochs after the target epoch. For DPF, fine-tuning for 60 epochs is required to improve performance. We compare the accuracy and training time of DCIL with reduced training time to both original DPF and fine-tuned DPF in ResNet-20 and ResNet-32. *Time* column refers to the ratio compared to the DPF training and fine-tuning time. All numbers of DPF are from the original paper of DPF [59].

Method	Total Epoch	Target Epoch	Top-1 Test Acc.		Time
			ResNet-20	ResNet-32	
DCIL	125	100	89.63	92.07	$\times 0.52$
	150	125	90.04	92.56	$\times 0.63$
	175	150	90.50	92.93	$\times 0.73$
	200	175	90.62	93.15	$\times 0.83$
	225	200	90.49	93.03	$\times 0.94$
	250	225	91.06	93.23	$\times 1.04$
DPF	300	225	88.01	90.94	$\times 0.83$
	360 $(300 + 60)$	225	90.34	92.18	$\times 1.0$

5.3.7 Tendency of warm-up

We introduced the warm-up epoch, an early period of learning without KL loss but only using the cross-entropy loss, and confirmed that the performance is stable across various warm-up epochs. The length of the warm-up epoch does not significantly affect performance and we did not search the hyper-parameter of warm-up in this work. Also, the KL loss help enhance the performance somewhat.



(a) ResNet-20 with 95% pruning on CIFAR-10

(b) ResNet-18 with 90% pruning on ImageNet

Figure 5.8: Top-1 test accuracy on various warm-up epochs.

5.3.8 CIFAR10

Table 5.10 shows the top-1 test accuracy for unstructured pruning of 95% sparsity according to the warm-up epoch in ResNet-20 and ResNet-32 and Fig. 5.8(a) compares the top-1 test accuracy across different warm-up epochs. The performance gap due to the different warm-up epoch is not significant, and the value of 70 which is used in the main experiment for warm-up epoch may be tuned.

Table 5.10: Top-1 test accuracy on CIFAR10 according to warm-up epoch. We report the *last accuracy* for each warm-up epoch value. All reported numbers are averaged over three times.

Model	Target Pr.ratio	$w = 0$	$w = 30$	$w = 70$	$w = 120$	$w = 200$	$w = 300$ (w/o KL)
ResNet-20	95%	90.43 ± 0.31	90.46 ± 0.14	90.54 ± 0.19	90.81 ± 0.22	90.14 ± 0.22	90.07 ± 0.31
ResNet-32	95%	91.85 ± 0.38	92.02 ± 0.06	92.04 ± 0.24	92.01 ± 0.25	91.92 ± 0.10	91.41 ± 0.22

5.3.9 ImageNet

We also applied different warm-up epochs (10, 70 and DCIL w/o KL) on the ImageNet dataset to confirm the effect of warm-up epoch. Table 5.11 and Fig.

5.8(b) shows the numbers and curves of ResNet-18 on the ImageNet dataset.

Table 5.11: Top-1 test accuracy on ImageNet according to warm-up epoch. We report the *last accuracy* for each warm-up epoch value.

Model	Target Pr.ratio	$w = 10$	$w = 70$	$w = 90$ (w/o KL)
ResNet-18	90%	68.66	68.55	68.37

5.3.10 Analysis of training and inference overheads

We analyze overhead in Table 5.12. During training, our model needs only 0.67% more parameters than DPF by adding BatchNorm and the last FC layer. Despite the two forward-backward, the overall training time and memory usage actually take about 1.5x due to other process factors such as computing the mask in Pruned net. Note that, there is no additional cost of inference compared to DPF. Furthermore, 'Fast convergence of DCIL' shows that DCIL outperforms DPF with the same training time.

Table 5.12: We report the numbers for ResNet32 model on CIFAR10 using a single TITAN RTX GPU.

		During Training			During Inference	
Model	Pr.ratio	Param. #	Memory	Time / Epoch	Param. #	MAC
Dense	-	0.467M	1056MiB	7.85s	0.467M	70.06M
DPF	90%	0.467M	1177MiB	9.10s	0.047M	11.66M
unstructured	95%	0.467M	1177MiB	9.10s	0.023M	6.35M
Ours	90%	0.470M	1511MiB	14.16s	0.047M	11.49M
unstructured	95%	0.470M	1511MiB	14.16s	0.023M	6.26M

5.4 Conclusion

To get the best out of model efficiency, *dynamic pruning* methods have been studied, which find an efficient sparse network with dynamic sparse patterns. To make diverse sparse patterns, reviving inactive weights with coarse gradients using STE has been used. This causes instability during training and performance degradation due to the gradient approximation. In this work, we propose a novel Dynamic Collective Intelligence Learning (DCIL) which finds a sparse model by training inactive weights with refined gradients rather than using approximated coarse gradients. This can help make inactive weights be superior candidates for future active weights. DCIL outperforms other pruning methods with enhanced stability for various architectures on CIFAR and ImageNet.

Chapter 6

Deep Model Compression via KD, Quantization and Pruning (KQP)

We introduced novel methods in three tasks through previous chapters, including knowledge distillation, quantization, and pruning. Knowledge distillation can improve the performance of a student network with a simple regularization by using the information from the teacher network. Quantization and pruning give power and memory efficiency to the model. In this chapter, we propose a unified model compression framework via KD, quantization and pruning (KQP) to leverage these advantages from three tasks.

6.1 Method

We design KQP algorithm by combining the proposed DCIL and PSGD. Concerning the distilling the knowledge, we use logit-based knowledge distillation than the proposed feature-based knowledge distillation because distilling the information of feature maps can be harmful if the types of representing the precision of the teacher and the student network are different e.g., full-precision

teacher and low-precision student networks [40].

In this unified KQP framework, we utilize the PSGD optimization at the P-net training and SGD optimization at S-net training. When applying the PSGD, we prune the network first and then calculate the quantization range and the step size. This can help to make the distribution of the active weights quantization-friendly. To exploit the PSGD optimization with P-net, the scaling function $s(\bar{w})$ calculates the l1 distance between the active weights ($\bar{w}$) and the quantized active weights ($\hat{w}$).

$$s(\bar{w}) = |\bar{w} - \hat{w}(\bar{w})| + \epsilon. \tag{6.1}$$

In the S-net training, we use naive SGD optimization because inactive weights have no reason to be quantized. Based on these rules, we can write the update rule of the unified framework, called KQP.

$$\begin{aligned} &\mathbf{W} \leftarrow \mathbf{W} - \eta\{\mathbf{S}(\overline{\mathbf{W}}) \odot \mathbf{M} \odot \nabla_{\overline{\mathbf{W}}}\mathcal{L} + (1-\mathbf{M}) \odot \nabla_{\mathbf{W}}\mathcal{L}\} \\ &\nabla_{\overline{\mathbf{W}}}\mathcal{L} \triangleq \frac{\partial \mathcal{L} \circ \mathcal{F}^{\overline{\mathbf{W}}}(\mathbf{X})}{\partial \overline{\mathbf{w}}}, \quad \nabla_{\mathbf{W}}\mathcal{L} \triangleq \frac{\partial \mathcal{L} \circ \mathcal{F}^{\mathbf{W}}(\mathbf{X})}{\partial \mathbf{w}}. \end{aligned} \tag{6.2}$$

where $\mathbf{S}(\overline{\mathbf{W}}) = [s(\bar{w}_1), s(\bar{w}_2), \cdots s(\bar{w}_n)]^T, \overline{\mathbf{W}} \in \mathbf{R}^{\mathbf{n}}$.

The overall process of the unified KQP framework is depicted in Fig. 6.1.

6.2 Experiment

We conducted the KQP experiment with various bit-with and both 90% and 95% pruning ratios on CIFAR-100 dataset with ResNet-32. We followed the same training setting of DCIL [45]. We compared our KQP method with the DCIL as a baseline. We trained the KQP model with each target whose bit-rates are 3,4,6 and 8 bit. We did not quantize the last linear layer and activation layer in this experiment. Table 6.1 shows accuracies of ResNet-32 on CIFAR-100 dataset

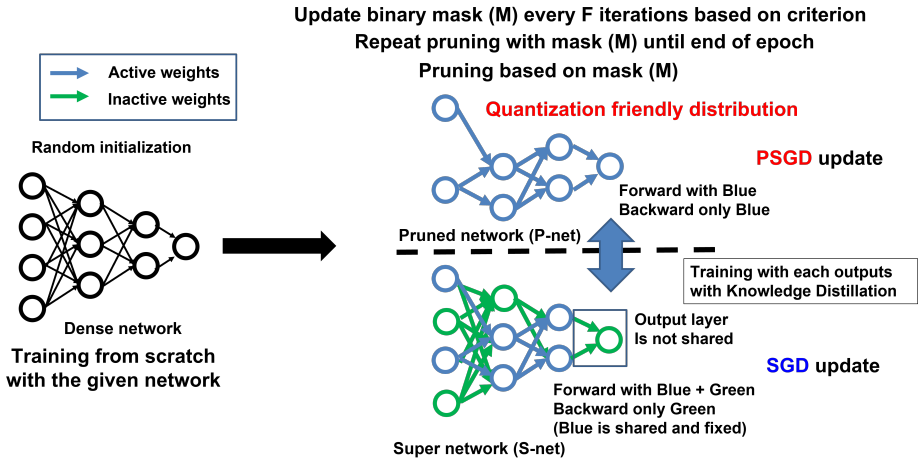


Figure 6.1: The overall process of KQP. The distribution of active weights needs to be quantization-friendly. To this end, In the P-net training, we utilize PSGD optimization.

according to full-precision and low-precision using the symmetric quantization used in the previous PSGD method. In the relatively higher bit-width such as 6 and 8 bit, DCIL outperforms the KQP because PSGD acts as a regularization to make model distribution quantization-friendly. In a higher bit-width, because of the powerful representing precision of the bit-with, PSGD is harmful to train a P-Net. On the other hand, in the case of the low bit-width including 3 and 4 bit, PSGD is very effective with respect to preventing performance degradation from the quantization. This is because the lack of the representing power of the low bit-width brings performance degradation from the discrepancy between the distribution of full-precision and that of low-precision. PSGD can relax this discrepancy compared to naive SGD optimization even at the 90% and 95% pruning ratios. From this experiment, KQP shows the effectiveness at the low-bits such as 3 and 4 bit even at 90% and 95% pruning. In low-bit experiments, KQP

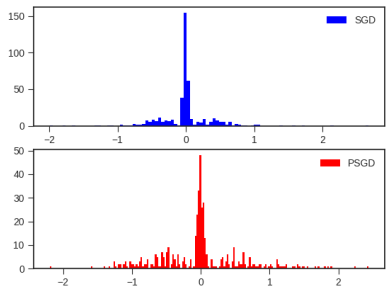
maintains the accuracy of full-precision compared to the vanilla SGD trained model (69.50 ± 0.20) in a 90% pruning ratio.

Fig. 6.2 and Fig. 6.3 show weight distributions from the model trained with DCIL and the model trained with KQP at the conv1 and conv1 of layer3 in the ResNet architecture. In the figure, @xbit represents the target bit-width of the KQP method. In the case of relatively higher precision (6,8 bit), weight distributions between the KQP and DCIL are not significantly different because of the representing power of bit-width. However, the regularization effect of PSGD adversely affects the performance. On the contrary, In the case of the low precision (3, 4 bit), the two distributions from DCIL and KQP show remarkably different because generally SGD trained model has a unimodal bell-shaped distribution but PSGD trained model has a multimodal distribution according to the number of the bin of quantization [46].

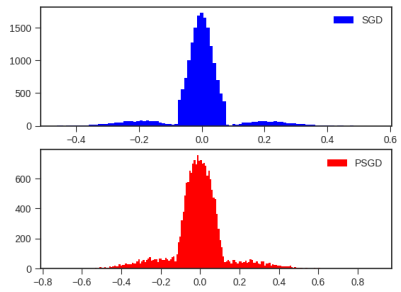
Unlike the original SGD method where pruning is not applied, DCIL-SGD normally has a multimodal distribution having three peaks. This phenomenon comes from magnitude-based pruning of DCIL. DCIL prunes the weights located in a certain pruning range satisfying the target pruning ratio. In the Fig. 6.2 and Fig. 6.3, There are three peaks colored with the blue according to each figure. A relative significant distribution, inactive weights, near the zero value are in the certain pruning range, which will be pruned with a mask $\mathbf{M}$. The other two insignificant distributions which are active weights, are located in the out of the pruning range. KQP controls these active weights with PSGD for making a compression-friendly distribution. In every figure, active weights are clustered according to their target bit.

Table 6.1: The accuracy of ResNet-32 with on the CIFAR-100. SGD & DCIL refers pure DCIL method where quantization is not considered. PSGD & DCIL, KQP, is the unified training framework considering quantization and pruning.

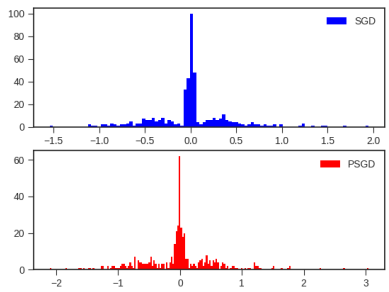
Method	Pruning ratio	(FP / 3bit)	(FP / 4bit)	(FP / 6bit)	(FP / 8bit)
SGD & DCIL	90%	70.01 / 25.76	70.41 / 62.11	70.51 / 70.04	70.40 / 70.41
	95%	67.90 / 24.76	67.68 / 58.27	67.76 / 67.47	68.07 / 67.88
(PSGD & DCIL	90%	70.18 / 58.22	69.50 / 68.08	69.38 / 68.98	68.61 / 68.43
= KQP)	95%	67.23 / 55.09	67.30 / 65.25	65.78 / 65.99	64.51 / 64.54



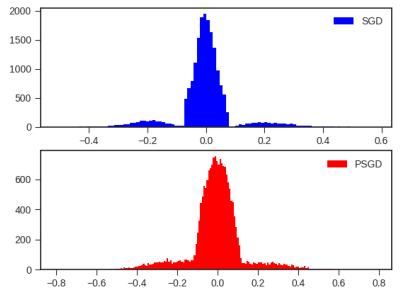
(a) Conv1 @6bit



(b) Conv1-layer3 @6bit

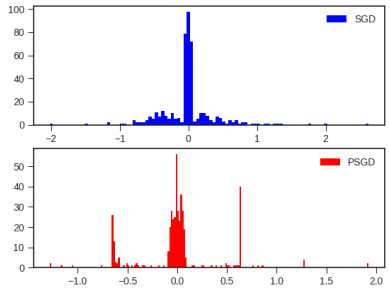


(c) Conv1 @8bit

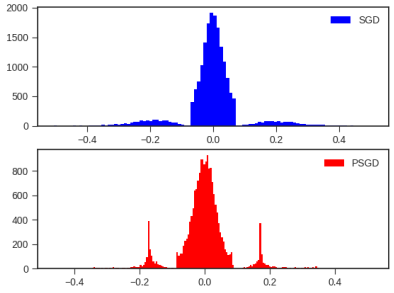


(d) Conv1-layer3 @8bit

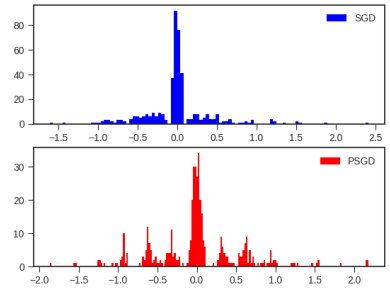
Figure 6.2: Weight distributions from the Blue:DCIL and Red:KQP with 6 and 8 target bit.



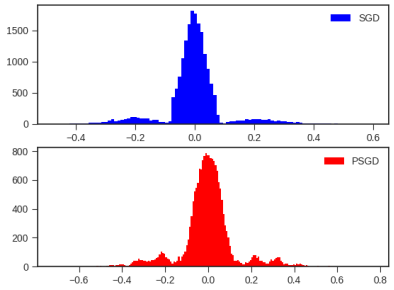
(a) Conv1 @3bit



(b) Conv1-layer3 @3bit



(c) Conv1 @4bit



(d) Conv1-layer3 @4bit

Figure 6.3: Weight distributions from the Blue:DCIL and Red:KQP with 3 and 4 target bit.

6.3 Conclusion

In this chapter, we propose a unified model compression framework named as KQP. KQP appropriately utilizes each advantage from KD, Quatization and pruning. Although PSGD acts as a strong regularizer which has a bad effect in the high precision, PSGD is effective even with 90% and 95% pruning in the low precision.

Chapter 7

Conclusion

Toward on-device deep learning, the model compression has been studied in three ways: knowledge distillation, quantization, and pruning. In this dissertation, we propose three novel model compression methods. Here, we summarize the proposed three methods and discuss the limitation of our works and the future directions of our work for on-device deep learning.

7.1 Summary

In chapter 3, we consider indirect knowledge distillation for enhancing the performance of the student model without any inherent differences between the teacher and the student model. We introduce two convolutional modules, the paraphraser and the translator. The paraphraser is attached at the last layer of the teacher model and extracts factors from the teacher's feature maps. Similarly, the translator is attached at the last layer of the student model and tries to absorb the teacher factor by minimizing a l1 distance between the teacher factor and student factor. By doing so, the student model can be trained with the factor information

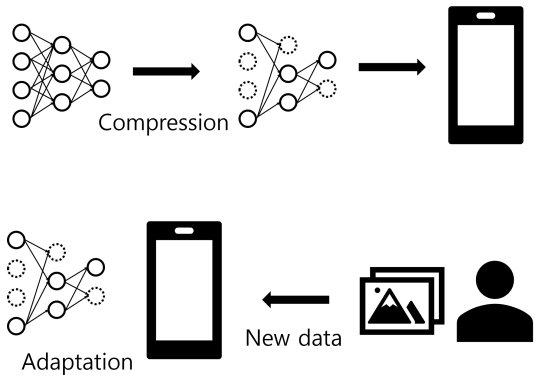


Figure 7.1: Toward on-device deep learning, the adaptation as well as model compression is a very important issue in the real application.

from the teacher model by distilling the knowledge of feature maps indirectly.

In chapter 4, we show the fundamental discrepancy in weight distribution between the full-precision model and the quantized low-precision model. This brings inevitable performance degradation when the model is quantized. To tackle this problem, we propose a regularization method to make model weights quantization-friendly. We introduce the scaling function that rescale the gradient based on the l1 distance between original weights and quantized weights. This scaling function can help to escape the position of the weight not appropriate for the quantization.

In chapter 5, we tackle the radical problem of the general dynamic pruning method which finds the sparse pattern with approximated gradients. This coarse gradient can bring the unintended direction of the gradient, causing training instability and performance degradation. We propose dual forwarding paths training framework to calculate the refined gradients which are very stable in the training and good effect in enhancing the performance compared to the coarse gradient.

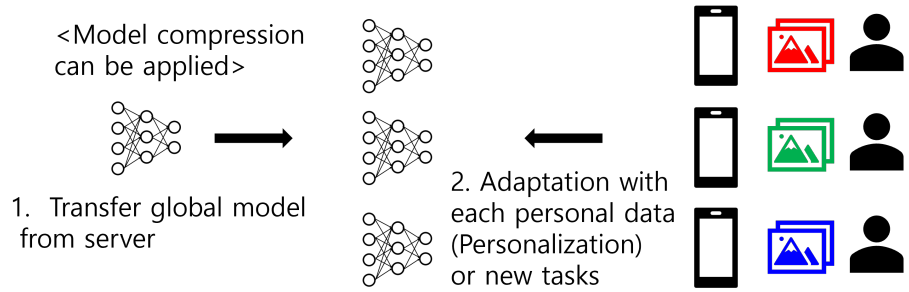


Figure 7.2: Model compression methods can be applied at the global model training. The global model should adapt the personal data at the expense of minimum costs.

In chapter 6, we consider a unified model compression framework combining logit-based knowledge distillation, PSGD and DCIL, called KQP. KQP is very effective with a high pruning ratio and low-bit quantization.

7.2 Limitations and Future Directions

In this dissertation, we propose indirect and implicit regularizations by rescaling and refining the gradient. Obviously, These methods are very effective in the model compression tasks. However, although the regularization method can maintain reasonable accuracy in the full precision model, the regularization method has a lower performance compared to quantization-aware training. With respect to extremely low bit quantization and the performance of the low precision model, the regularization method should consider enhancing the performance of the low precision model for the real application in further research.

Proposed model compression methods are well designed in terms of compressing the model size and reducing the power consumption. To achieve the main goal, deploying the deep learning on the edge device also considers the task

or data adaptation problem depicted in Fig. 7.1. In the real-world application, the pretrained model needs to adapt incoming datasets or a new task.

For example, Fig. 7.2 shows the real application scenario about on-device deep learning. First, the global model is learned on the server, where it can be compressed using the model compression technique considering the operation at the resource-constraint device. Then, the pretrained global model should adapt the new task and personal data generated from each user called personalization [42].

We believe the model compression methods and the adaptation process should be studied simultaneously and meta-learning could be a good solution to give a model flexibility.

Bibliography

[1] G. Alain and Y. Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, 2014.

[2] M. Alizadeh, A. Behboodi, M. van Baalen, C. Louizos, T. Blankevoort, and M. Welling. Gradient ℓ_1 regularization for quantization robustness. In *International Conference on Learning Representations*, 2020.

[3] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, pages 7948–7956, 2019.

[4] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[5] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–697, 2020.

[6] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[7] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

[9] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. T. Chayes, L. Sagun, and R. Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations*, 2017.

[10] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204, 2015.

[11] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev. Low-bit quantization of neural networks for efficient inference. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct 2019.

[12] I. Chung, S. Park, J. Kim, and N. Kwak. Feature-map-level online adversarial knowledge distillation. In *International Conference on Machine Learning*, pages 2006–2015. PMLR, 2020.

[13] W. Dally. High-performance hardware for machine learning. *NIPS Tutorial*, 2, 2015.

[14] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[15] C. De Sa, M. Leszczynski, J. Zhang, A. Marzoev, C. R. Aberger, K. Olukotun, and C. Ré. High-accuracy low-precision training. *arXiv preprint arXiv:1803.03383*, 2018.

[16] J. E. Dennis, Jr and J. J. Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[17] T. Dettmers and L. Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[18] X. Dong, J. Huang, Y. Yang, and S. Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[20] 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.

[21] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *ArXiv*, abs/1902.09574, 2019.

[22] N. Goli and T. M. Aamodt. Resprop: Reuse sparsified backpropagation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1548–1558, 2020.

[23] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. 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.

[24] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[25] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[26] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[27] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. *Advances in Neural Information Processing Systems*, 29:1379–1387, 2016.

[28] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 2015.

[29] B. Hassibi and D. G. Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

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

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

[32] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[33] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[34] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

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

[36] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[37] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

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

[39] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[40] J. Kim, Y. Bhalgat, J. Lee, C. Patel, and N. Kwak. Qkd: Quantization-aware knowledge distillation. *arXiv preprint arXiv:1911.12491*, 2019.

[41] J. Kim, S. Chang, and N. Kwak. Pqk: Model compression via pruning, quantization, and knowledge distillation. *arXiv preprint arXiv:2106.14681*, 2021.

[42] J. Kim, S. Chang, S. Yun, and N. Kwak. Prototype-based personalized pruning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3925–3929. IEEE, 2021.

[43] J. Kim, M. Hyun, I. Chung, and N. Kwak. Feature fusion for online mutual knowledge distillation. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4619–4625. IEEE, 2021.

[44] J. Kim, S. Park, and N. Kwak. Paraphrasing complex network: Network compression via factor transfer. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[45] J. Kim, J. Yoo, Y. Song, K. Yoo, and N. Kwak. Dynamic collective intelligence learning: Finding efficient sparse model via refined gradients for pruned weights. *arXiv preprint arXiv:2109.04660*, 2021.

[46] J. Kim, K. Yoo, and N. Kwak. Position-based scaled gradient for model quantization and pruning. In *Advances in Neural Information Processing Systems*, volume 33, pages 20415–20426. Curran Associates, Inc., 2020.

[47] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018.

[49] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[50] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[51] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research).

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[53] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[54] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[55] N. Lee, T. Ajanthan, and P. Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019.

[56] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations*, 2017.

[57] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[58] J. Lin, C. Gan, and S. Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations*, 2019.

[59] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. *International Conference on Learning Representations*, 2020.

[60] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *InternationalConference on Learning Representations*, 2019.

[61] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[62] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[63] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[64] H. Mostafa and X. Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655, 2019.

[65] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort. Up or down? adaptive rounding for post-training quantization. *arXiv preprint arXiv:2004.10568*, 2020.

[66] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of*

the IEEE International Conference on Computer Vision, pages 1325–1334, 2019.

[67] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson. Loss aware post-training quantization. *arXiv preprint arXiv:1911.07190*, 2019.

[68] W. W. Ng, G. Zeng, J. Zhang, D. S. Yeung, and W. Pedrycz. Dual autoencoders features for imbalance classification problem. *Pattern Recognition*, 60:875–889, 2016.

[69] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[70] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[71] A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. *InternationalConference on Learning Representations*, 2020.

[72] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[73] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[74] H.-C. Shin, M. R. Orton, D. J. Collins, S. J. Doran, and M. O. Leach. Stacked autoencoders for unsupervised feature learning and multiple organ

detection in a pilot study using 4d patient data. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1930–1943, 2013.

[75] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[77] X. Sun, X. Ren, S. Ma, and H. Wang. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *International Conference on Machine Learning*, pages 3299–3308. PMLR, 2017.

[78] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei. Adaptivfloat: A floating-point based data type for resilient deep learning inference. *arXiv preprint arXiv:1909.13271*, 2019.

[79] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[80] M. van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling. Bayesian bits: Unifying quantization and pruning. *arXiv preprint arXiv:2005.07093*, 2020.

[81] B. Wei, X. Sun, X. Ren, and J. Xu. Minimal effort back propagation for convolutional neural networks. *arXiv preprint arXiv:1709.05804*, 2017.

[82] C. Wei, S. Kakade, and T. Ma. The implicit and explicit regularization effects of dropout. In *International Conference on Machine Learning*, pages 10181–10192. PMLR, 2020.

[83] X. Xiao and Z. Wang. Autoprune: Automatic network pruning by regularizing auxiliary parameters. *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 32, 2019.

[84] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[85] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[86] S. Zagoruyko and N. Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

[87] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[88] D. Zhang, J. Yang, D. Ye, and G. Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.

[89] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

[90] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. *International Conference on Machine Learning (ICML)*, pages 7543–7552, June 2019.

[91] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

초 록

Deep Neural Network (DNN)은 빠르게 발전하여 컴퓨터 비전, 자연어 처리 및 음성 처리를 포함한 많은 영역에서 놀라운 성능을 보여 왔다. 이러한 DNN의 발전에 따라 edge IoT 장치와 스마트폰에 DNN을 구동하는 온디바이스 DNN에 대한 수요가 증가하고 있다. 그러나 DNN의 성장과 함께 DNN 매개변수의 수가 급격히 증가했다. 이로 인해 DNN 모델을 리소스 제약이 있는 에지 장치에 구동하기가 어렵다. 또 다른 문제는 에지 장치에서 DNN의 전력 소비량이다 왜냐하면 에지 장치의 전력용 배터리가 제한되어 있기 때문이다. 위의 문제를 해결하기 위해서는 모델 압축이 매우 중요하다.

이 논문에서 우리는 지식 증류, 양자화 및 가지치기를 포함한 모델 압축의 세 가지 새로운 방법을 제안한다. 먼저, 지식 증류라고 불리는 방법으로써, 교사 네트워크의 추가 정보를 사용하여 학생 모델을 학습시키는 것을 목표로 한다. 이 프레임워크를 사용하면 주어진 매개변수를 최대한 활용할 수 있으며 이는 장치의 리소스가 제한된 상황에서 중요하다. 기존 지식 증류 프레임워크와 달리 네트워크 구조, 배치 무작위성 및 초기 조건과 같은 교사와 학생 간의 고유한 차이가 적절한 지식을 전달하는 데 방해가 될 수 있으므로 피쳐에서 요소를 추출하여 지식을 간접적으로 증류하는 데 중점을 둔다.

둘째, 양자화를 위한 정규화 방법을 제안한다. 양자화된 모델은 자원이 제한된 에지 장치에 중요한 전력 소모와 메모리에 이점이 있다. 파라미터 분포를

양자화 친화적으로 만들기 위해 훈련 시간에 모델의 기울기를 불균일하게 재조정한다. 우리는 그라디언트의 크기를 재조정하기 위해 position-based scaled gradient (PSG)를 사용한다. Stochastic gradient descent (SGD) 와 비교하여, 우리의 position-based scaled gradient descent (PSGD)는 모델의 양자화 친화적인 가중치 분포를 만들기 때문에 양자화 후 성능 저하를 완화한다.

셋째, 중요하지 않은 과잉 매개 변수화 모델을 제거하기 위해, 가지치기된 가중치의 대략적인 기울기에 Straight-Through-Estimator (STE)를 활용하여 훈련 중에 다양한 희소성 패턴을 찾으려고 하는 동적 가지치기 방법이 등장했다. STE는 동적 희소성 패턴을 찾는 과정에서 제거된 파라미터가 되살아나도록 도울 수 있다. 그러나 이러한 거친 기울기 (coarse gradient)를 사용하면 STE 근사의 신뢰할 수 없는 기울기 방향으로 인해 훈련이 불안정해지고 성능이 저하된다. 이 문제를 해결하기 위해 우리는 이중 전달 경로를 형성하여 제거된 파라미터 (pruned weights)를 업데이트하기 위해 정제된 그라디언트를 제안한다. 가지치기에 거친 기울기를 사용하지 않기 위해 Dynamic Collective Intelligence Learning (DCIL)을 제안한다.

마지막으로 제안된 방법들을 이용하여 통합 모델 압축 훈련 프레임워크로서 결합한다. 이 방법은 극도로 희소하고 양자화 친화적인 모델을 훈련할 수 있다.

주요어: 심층 모델 압축, 지식 증류, 양자화, 가지치기, 딥러닝

학번: 2017-39082

감사의 글

학위를 무사히 마칠 수 있도록 도와주신 많은 분들께 감사인사를 올립니다. 먼저, 존경하는 곽노준 지도교수님과 학위 논문의 심사를 맡아주신 교수님들께 감사드립니다. 훌륭한 MIPAL 선후배님들, 친구들 그리고 사랑하는 가족들에게 감사합니다. 언제나 응원해준 아내, 사랑합니다.

中石沒鏃. 항상 배우는 자세로 초심을 지키며 정진하는 삶을 살겠습니다.