Vector Quantization of Deep Convolutional Neural Networks with Learned Codebook

Siyuan Yang

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the degree of
Master of Applied Science

Electrical and Computer Engineering
Faculty of Engineering
University of Ottawa

Examining Committee

Supervisor:	Yongyi Mao
	Professor, Dept. of Engineering, University of Ottawa
Internal Members:	Robert Laganière
	Professor, Dept. of Engineering, University of Ottawa
	Babak Esfandiari
	Professor, Dept. of Engineering, Carleton University

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Deep neural networks (DNNs), particularly convolutional neural networks (CNNs), have been widely applied in the many fields, such as computer vision, natural language processing, speech recognition and etc. Although DNNs achieve dramatic accuracy improvements in these real-world tasks, they require significant amounts of resources (e.g., memory, energy, storage, bandwidth and computation resources). This limits the application of these networks on resource-constrained systems, such as mobile and edge devices. A large body of literature has been proposed to addresses this problem from the perspective of compressing DNNs while preserving their performance. In this thesis, we focus on compressing deep CNNs based on vector quantization techniques.

The first part of this thesis summarizes some basic concepts in machine learning and popular techniques on model compression, including pruning, quantization, low-rank factorization and knowledge distillation approaches. Our main interest is quantization techniques, which compress networks by reducing the precision of parameters. Full-precision weights, activations and even gradients in networks can be quantized to 16-bit floating point numbers, 8-bit integers, or even binary numbers. Despite a possible performance degradation, quantization can greatly reduce the model size while maintaining model accuracy.

In the second part of this thesis, we propose a novel vector quantization approach, which we refer to as Vector Quantization with Learned Codebook, or VQLC, for CNNs. Rather than performing scalar quantization, we choose vector quantization that can simultaneously quantize multiple weights at once. Instead of taking a pretraining/clustering approach as in most works, in VQLC, the codebook for quantization are learned together with neural network training from scratch. For the forward pass, the traditional convolutional filters are replaced by the convex combinations of a set of learnable codewords. During inference, the compressed model will be represented by a small-sized codebook and a set of indices, resulting in a significant reduction of model size while preserving the network's performance.

Lastly, we validate our approach by quantizing multiple modern CNNs on several popular image classification benchmarks and compare with state-of-the-art quantization techniques. Our experimental results show that VQLC demonstrates at least comparable and often superior per-

formance to the existing schemes. In particular, VQLC demonstrates significant advantages over the existing approaches on wide networks at the high rate of compression.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Mao for his patience, guidance and support during my graduate studies. I greatly appreciate his invaluable help and training, especially in how to think critically. The knowledge and skills I gained during my time in his group I will use for the rest of my life.

To the lab members of the group: Ziqiao, Runzhi, Chenjie, Boming, it has been a truly unforgettable experience working with all of you.

To my family: Dad, Mom, Grandma, Grandpa, and my brother Fengyuan, thank all of you for always giving me strength, supporting and believing in me every step of the way - I am truly grateful that we are family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the recent years, machine learning have shown remarkable improvement in computer vision, natural language processing and audio/video processing [67]. In particular, deep neural networks (DNNs) have been successfully applied to many complex real-world tasks such as image classification [61], segmentation [31], object detection [93], speech recognition [38] and language translation [105]. These successes often rely on neural networks with large number of parameters, in the scale of millions or billions. The deployment of such networks then requires significant memory, computation, storage, bandwidth and energy resources. This limits the application of such networks on resource-constrained systems, such as mobile phones and edge-devices. In such systems, the deployed neural networks are desirable to be resource-efficient, for example, having small number of parameters, using a small amount of memory and requiring less computation power.

This thesis addresses this problem from the perspective of reducing the network parameters. Such an approach is feasible since DNNs are traditionally over-parameterized, sometimes to the extent that 95% of the parameters can be predicted from the remaining 5% [22]. Although over-parameterization plays important roles in the training of DNNs (e.g., easier to train with stochastic gradient descent [33, 46, 59, 61], beneficial for optimization and generalization [3, 101]), it is in fact possible to remove the redundancy in the model parameters without sacrificing performance, as demonstrated by various model compression techniques [9, 10, 12, 35, 43] and other empirical observations [27].

A large body of literature has been proposed to compress neural network models. They can be broadly categorized as weight pruning, quantization, low-rank factorization and knowledge distillation. The weight pruning techniques [4, 11, 43, 50, 51, 70, 116] directly remove the "unimportant" model parameters ones without severely hurting the model performance. Quantization [10, 22, 25, 39, 56, 80, 86, 97, 98, 104] compresses the neural networks by reducing the number of bits required to represent the parameters. Low-rank factorization [1, 9, 19, 21, 26, 55, 109, 115] utilizes low-rank tensor decomposition to factorize the network weights. In knowledge distillation [6, 12, 36, 48, 114], a smaller network is learned to simulate the behaviour of a large trained network.

In this thesis, we focus on quantization techniques for model compression. Network quantization is an appealing approach, aiming at reducing the number of bits required to represent the parameters of the network [17, 30, 40, 72]. Existing network quantization approaches may be broadly categorized into two families: scalar quantization and vector quantization. The former quantizes each parameter individually whereas the latter directly quantizes a block of parameter simultaneously. It is well-known in information theory that scalar quantizers are in general inferior in their rate-distortion performance and that in general the optimal rate-distortion performance is only achievable via vector quantization [20]. As expected (and also shown in our experiments), in the context of neural network quantization, scalar quantizers are also shown to be limited in their achieved rate-accuracy performance. In this work, we focus on the design of vector quantization techniques for compressing convolutional neural networks.

Conventionally, a vector quantization scheme for network compression consists of three steps: pre-training, clustering, and fine-tuning. In the first step, a selected (usually over-parameterized) model is trained using SGD or its variants. In the second step, parameters of the learned model, grouped in "blocks", are clustered into a prescribed number of clusters (e.g. using K-means [44]), and the cluster centers are taken as the quantization codebook. In the third step, each parameter block in the network is replaced by its corresponding cluster center (or codeword), and the model is retrained to obtain a refined codebook. We hypothesize that such a pretrain-cluster-tune strategy limits the search space of the best quantization codebook. Specifically, the optimization objective in pretraining is the minimized classification error (or a surrogate loss reflecting the error) for image classification tasks. Such an objective may not be aligned with the objective of

optimizing the rate-accuracy trade-off achieved by a quantized codebook. Arguably, there are many parameter configurations that give a good classification accuracy and the one found by a single run of SGD may not be near the quantization codebook (in the sense of being reachable via the subsequent fine-tuning steps) that also optimizes the rate-accuracy trade-off.

Motivated by this hypothesis, this thesis sets out to develop a network quantization technique that search for a codebook in a larger space without relying the model parameters obtained from pre-training. To that end, we propose a novel vector quantization approach to the compression of convolutional neural networks, which we refer to as Vector Quantization with Learned Codebook, or VQLC. Instead of taking a pretrain-cluster-tune approach as in most works [35, 41, 78, 102, 103, 118], the codebook of VQLC is learned from scratch during the training of the neural network.

Moreover, we devise an additional loss in the training process, referred to as the "degeneracy-forcing loss", that forces the convex-combination coefficients to put all probability mass on a single codeword in the codebook. The model is trained using the degeneracy-forcing loss combined with the standard loss for classification. At the end of training, each filter block can be represented by a single codeword in the codebook and only a binary index of the codeword is required to store. Therefore, during the inference, a codebook and a sequence of indices are saved, leading to the significant reduction of the model size while preserving the network's performance.

Briefly, in VQLC, the convolutional filters in the neural network of interest are organized as filter blocks of regular sizes, in a way similar to [71, 102]. A qunatization codebook is randomly initialized to contain a prescribed number of filter blocks. Each filter block in the network is then expressed as a convex of blocks in the codebook using an attention mechanism [110]. The codebook and the convex-combination coefficients are then learned via minimizing the standard cross-entropy loss augmented by a "degeneracy-forcing loss" that aims at forcing each set of convex-combination coefficients to emphasize only one block in the codebook.

Through extensive experiments, we validate the proposed VQLC scheme and compare it with the current art of network quantization techniques. Our experiments suggest that VQLC demonstrates at least comparable and often superior performance to the existing schemes. In particular, we observe that VQLC demonstrates significant advantages over the existing schemes on wide

networks. For example, at about 56× compression rate, VQLC achieves the same classification accuracy on CIFAR10 as the uncompressed model; on CIFAR100, by sacrificing the accuracy by 1.2%, VQLC is capable of achieving nearly 36× compression rate. Our quantized model on MNIST task even outperform the umcompressed model at 117.54×. These results also, to a great extent, validates the hypothesis that motivates this research.

The contributions of this thesis are summarized as follows:

- We propose a novel vector quantization approach, VQLC, for compressing deep convolutional neural networks. It can be easily and flexibly generalize to various CNN architectures.
- We introduce a novel and effective training scheme in which an additional "degeneracy-forcing loss" is utilized combined with the standard cross entropy loss for classification, gradually forcing a smooth transition from soft to hard quantization.
- We validate the performance of our proposed VQLC in comparison with several start-of-the-art quantization techniques on popular some image classification benchmarks. The experimental results indicate that our approach is at least comparable and often superior to other techniques. Especially for the wide networks, VQLC shows significant advantages.

The rest of this thesis is organized as follows: Chapter 2 gives some backgrounds and basic concepts in machine learning, and provides some model compression techniques. Then we discuss quantization techniques in detail in Chapter 3. Starting from a brief history of quantization, some common quantization approaches are then introduced, including low-bit precision, stochastic quantization and weight-sharing techniques. In Chapter 4, we explain the proposed VQLC for compressing CNNs in detail. Chapter 5 discusses our experimental results.

Chapter 2

Background

In this chapter, we briefly describe the background of neural networks and introduce several types of networks. First, we present a brief introduction on machine learning. Then, we will give an overview of deep neural networks and describe different types of neural networks.

2.1 Machine Learning

Machine learning is a subfield of artificial intelligence (AI). As was first defined in [96], machine learning is the study that gives computers the ability to learn, making predictions or decisions without being explicitly programmed. This means that instead of creating handcrafted, custom and purpose-based programs to solve each individual problem, a single machine learning algorithm will be able to deal with different, new problems by simply learning how to do intelligent activities via a training process.

In this thesis, we primarily consider a "suppervised" machine learning setting, using image classification as an example task. In this task, given a set of images $\{\mathbf{x}_i\}$ and their ground-truth labels $\{\mathbf{y}_i\}$, we wish to find a predictor $f_{\mathbf{W}}(\cdot)$ in the form of

$$\hat{\mathbf{y}} = f_{\mathbf{W}}(\mathbf{x}) \tag{2.1}$$

that takes an image as input $\mathbf{x}$ and generates a label $\hat{\mathbf{y}}$ that hopefully matches the ground-truth label $\mathbf{y}$. Here $f_{\mathbf{W}}$ specifies a family of functions parametrized by $\mathbf{W}$. In the training process, we

find the "best" $\mathbf{W}$ via fitting the function $f_{\mathbf{W}}$ using the training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}$. For that purpose, a loss function $\mathfrak{L}()$ is usually needed, where $\mathfrak{L}(\hat{\mathbf{y}}, \mathbf{y})$ measures the mismatch between $\hat{\mathbf{y}}$ and $\mathbf{y}$. The most common function used in classification tasks is the Cross-Entropy loss.

Cross-entropy is a concept from information theory. According to [83], the uncertainty in a probability distribution $p(x)$ can be described as

$$H(p) = -\mathbb{E}_{x\sim p}[\log(p(x))] \tag{2.2}$$

If we have two different distribution $p(x)$ and $q(x)$ over the same random variable, the difference between them can be measured by Kullback-Leibler Divergence [63], also called relative entropy. The divergence of $p(x)$ from $q(x)$, also known as the relative entropy of $p(x)$ with respect to $q(x)$, is defined to be

$$D_{\mathrm{KL}}(p\|q) = \mathbb{E}_{x\sim p}[\log\left(\frac{p(x)}{q(x)}\right)] = \mathbb{E}_{x\sim p}[\log(p(x)) - \log(q(x))] \tag{2.3}$$

The cross-entropy of the distribution $q(x)$ relative to $p(x)$ is defined as follows:

$$H(p,q) = -\mathbb{E}_{x\sim p}[\log(q(x))] \tag{2.4}$$

The definition can be reformulated using 2.2 and 2.3

$$H(p,q) = H(p) + D_{\mathrm{KL}}(p\|q) \tag{2.5}$$

When we fit a model, modelling a distribution $q(x)$ to the sample data distribution $p(x)$, our aim is to minimize the misfit between $q(x)$ and $p(x)$, which is measured by the divergence 2.3. Since the sample distribution $p(x)$ is fixed, if we minimize the cross entropy, we have also minimized the divergence. With maximum likelihood estimation, the cross entropy loss for a g-class classifier $q(x)$ is defined as

$$\mathfrak{L}_{\mathrm{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{g} p_{j,i} log(q_{j,i}), \text{ for } g \text{ classes} \tag{2.6}$$

where $q_{j,i}$ is the predicted (Softmax) probability for the j^{th} class the i^{th} data pair, the corresponding ground truth probability is $p_{j,i}$ and N is the number of data pairs in the dataset.

However, in general, we add a penalty called a "regularizer" to the cross entropy loss function for reducing over-fitting and improving model performance when training complex models. Hence, the overall loss function, "regularized loss function" is denoted by $\mathfrak{L}$:

$$\mathfrak{L} = \mathfrak{L}_{\text{CE}} + \lambda \mathfrak{L}_{\text{reg.}} \tag{2.7}$$

where λ is is a hyper-parameter that controls the strength of regularizer and $\mathfrak{L}_{\text{reg.}}$ is the norm penalty term.

One of the simplest and most common choice of the regularizer is the L2 parameter norm penalty, also known as "weight decay". The regularization term is defined as

$$\mathfrak{L}_{\text{reg.}}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{2.8}$$

Therefore, to perform classification with weight decay, we minimize an overall loss $\mathfrak{L}$ including the cross entropy error on the training data and a regularizer that encourages the smaller weights.

2.2 Neural Networks

Since the brain is currently the best "machine" we know for learning and solving problems, it is natural to look for a machine learning approach from the brain mechanism. The main computational element of the brain is the neuron [106]. The neuron in our brains accepts the signals entering it as inputs, performs computations on them, and then outputs a signal to postneurons [29]. Different computational operations result in different responses to the input signals. Inspired from this brain information processing mechanism, the neural networks were proposed.

A neural network is a network composed of artificial neurons or nodes [53]. The node, as the basic unit of a neural network, receives signals from the connected prenodes, conducts a non-linear transformation on the weighted sum of the input signals, and then produces an output signal that will be sent to postnodes. We denote the weight from node i to the node j as w_{ij}, the

number of the connected prenodes of the node j as m, the input vector of node j as $\mathbf{x} \in \mathbb{R}^m$, the output of node j as y_j. Then the neural operation can be represented as:

$$y_j = \varphi(b_j + \sum_i^m x_i w_{ij}) \tag{2.9}$$

where b_j is a bias and $\varphi(\cdot)$ is the nonlinear activation function.

Neural networks are generally divided into three types: (a). multilayer perceptron (MLP); (b). convolutional neural networks (CNN); (c). recurrent neural networks (RNN). In this article, we only focus on the the former two model types because of the popularity in the computer vision field. We provide the model details as follows.

2.2.1 Multilayer Perceptrons

Multilayer perceptrons (MLPs), composed solely of fully connected (FC) layers, consists of three parts: input layer, hidden layers and output layer. An example of a MLP is shown in Figure 2.1. The basic calculation in each neuron is the same as given in Equ.(2.9). Note that the activation function generally adopts $ReLU(x) = max(x, 0)$.

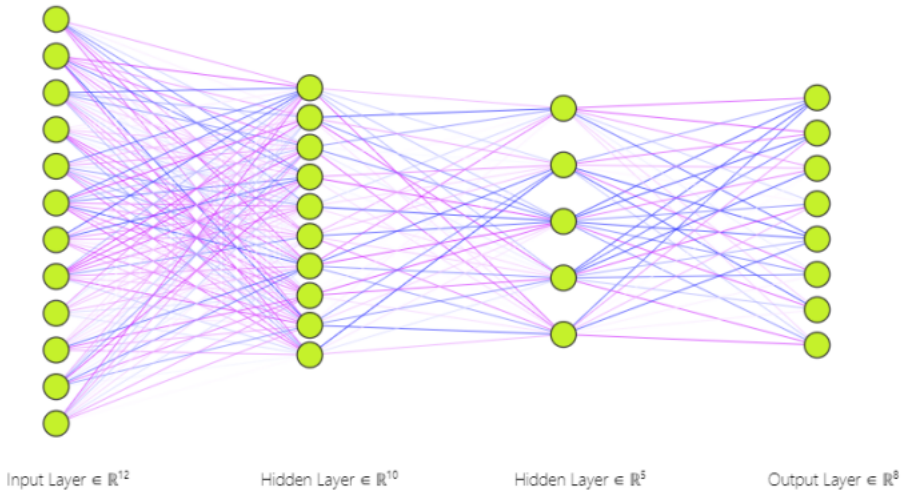


Figure 2.1: A four-layer MLP

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of artificial neural networks that have been successfully applied in many areas, especially in visual imagery [66]. A convolutional neural network usually contains three types of layers: convolutional layer, pooling layer, and fully-connected layer. Unlike MLPs processing the 1D data, CNNs target 3D data. A simple convolutional neural network is illustrated in Figure 2.2.

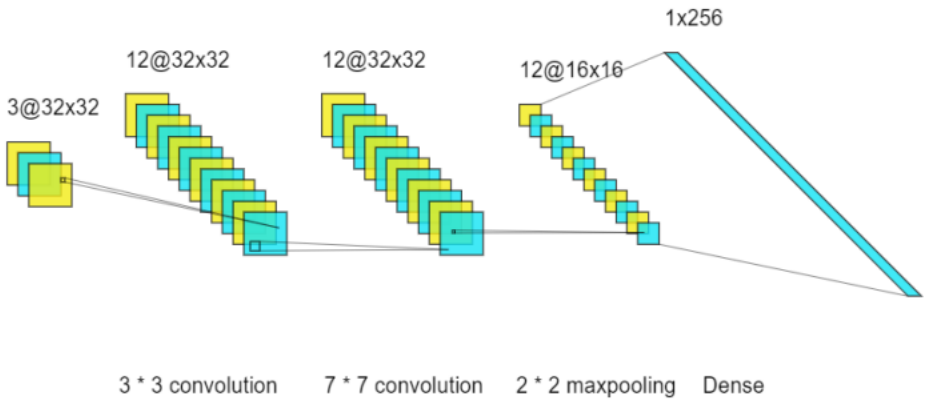


Figure 2.2: A simple convolutional neural network

Before going into details, we first summarizes terms and definitions used to describe convolutional neural networks, as shown in Table 2.2.

In computer vision tasks, convolutional layer is used to extract features from images.

$$\mathbf{A}_j^l = \varphi(\sum_{i=1}^{c_{in}} \mathbf{W}_{ij}^l * \mathbf{F}_i^l + b_j^l) \tag{2.10}$$

Equ.(2.10) represents the convolution operation of the l-th convolutional layer where $\mathbf{A}_j^l$ is the j-th activation map and $\mathbf{F}_i^l$ is the i-th feature map. $\mathbf{W}_{ij}^l$ represents the 2D kernel of the filter at the i-th input channel and j-th output channel. $\varphi(\cdot)$ is the nonlinear activation function and $*$ represents the convolution operation between two 2D tensors. The kernel $\mathbf{W}_{ij}^l$ slides across the feature map $\mathbf{F}_i^l$ and compute dot products between the kernel and local receptive field. Equ.(2.11) shows the convolution operation between the kernel and feature map, where $a_{k,g}$, $w_{k+m-1,g+n-1}$ and $f_{m,n}$ are respectively one element in the 2D activation map $\mathbf{A}_j^l$, kernel $\mathbf{W}_{ij}^l$ and input feature map $\mathbf{F}_i^l$.

Term	Definition
Parameter	A configuration variable that is internal to the network and whose value can be trained.
Hyperparameter	A predefined parameter before the training process and whose value is non-trainable.
Feature($\mathbf{F} \in \mathbb{R}^{c_{in} \times h_{in} \times w_{in}}$)	The input data of one convolutional layer, typically in height h_{in}, width w_{in}, and input channel c_{in}. The $h_{in} \times w_{in}$ is sometimes called a feature map.
Activation($\mathbf{A} \in \mathbb{R}^{c_{out} \times h_{out} \times w_{out}}$)	The output data of one convolutional layer, typically in height h_{out}, width w_{out}, and output channel c_{out}. The $h_{out} \times w_{out}$ is sometimes called an activation map. Generally, the activation of the current layer is the feature for the next layer.
Kernel($\mathbf{k} \in \mathbb{R}^{h \times w}$)	Convoluitonal weights for a channel. They are usually square and sized 1, 3, 7 in modern networks.
Filter($\mathbf{W} \in \mathbb{R}^{c_{out} \times c_{in} \times h \times w}$)	Convolutional coefficients of a convolutional layer. A filter consists of all 2D kernels corresponding to the c_{in} input channels. The filter's size c_{out} is the number of output channels.

Table 2.2: Some terms and definitions used in CNNs

$$a_{k,g} = \sum_{m=1}^{h} \sum_{n=1}^{w} w_{k+m-1,g+n-1} \times f_{m,n} \tag{2.11}$$

2.2.2.1 Popular CNNs

Many convolutional neural networks have been developed over the past decade and achieved breakthrough results in the computer vision tasks. In this article, we focus on the following three CNN architectures.

- VGGNet [100]

  The success of AlexNet [62] as the first CNN to win the ImageNet Challenge 2012, spawned a lot of novel CNN architectures. VGGNet was born out of the need to improve the model performance, and reduce the number of parameters in the network. This is achieved by increasing the depth of the network and replacing a large-sized kernel, such as 7×7 or 5×5, with multiple 3×3 kernels stacked together. Moreover, max pooling layers are added to reduce the dimensions of some layers. VGGNet is the first network structure that adopts block-based architecture [100].

- ResNet [46]

  Recent CNN architectures are designed to become deeper and deeper for the better performance. However, deep networks are hard to train due to the vanishing gradient problems which lead to the performance degradation of the network. To address these problems, [46] proposed ResNet which was implemented using the idea of Residual Blocks, with "shortcut connection" to fit the input from the previous layer to the next layer without modifying it.

  There are two types of "shortcut connections": Identity shortcut and Projection shortcut. When the shapes of the input and output of a residual block are the same, the identity shortcut is applied, just skipping the intermediate layers and adding the input directly to the last layer. For the case when the two shapes are different, the projection connection is implemented by using an additional 1×1 convolutional layer to transform the input into the desired shape for the next addition operation.

- DenseNet [54]

  [54] proposed DenseNet to reduce the vanishing gradient problems. DenseNet is composed of several dense blocks and within those blocks, each layer is densely connected to every other layer, treating the feature maps from all previous layers as the input and outputting activation maps to all subsequent layers.

  Instead of using element-wise addition operation like ResNets, DenseNet uses concatenation operation to add the features from all previous layers. In DenseNet, each convolutional layer has fewer number of filters, which makes the network thinner and compact.

There are 2 types of blocks in DenseNet: Dense blocks and Transition blocks. Each dense block consists of a 3×3 convolutional layer, a batch normalization layer and a ReLU activation function. And transition blocks are used for reducing the number of feature maps, each containing a 1×1 convolutional layer, an average pooling layer and a batch normalization layer. With this unique architecture, DenseNet has few number of feature maps and little redundancy in model parameters.

The performance of these models is impressive, however the huge model size prevents them from being widely used, especially in the mobile devices and embedded systems. This motivates lots of researchers to study compression methods to reduce the model size and required computation resources. More details can be found in Section 2.6.

2.3 Training and Inference

Deep neural networks (DNNs) and machine learning techniques have achieved significant success in solving traditionally challenging problems such as computer vision. DNNs are widely used in today's applications and systems. In general, DNNs are deployed as the following two steps: 1). Training and 2). Inference. We use image classification tasks as the example for explaining the training and inference processes.

Training process of a neural network involves feeding a data sample from the given dataset as an input into a network, propagating it through the model, predicting the classification result with the weights, and comparing it with the ground-truth label. Weights in the network are updated using a backpropagation strategy such as Stochastic Gradient Descent (SGD) to reduce classification errors. This performs a search for the best weight values. The learning algorithm keeps searching for the optimal weight values until the model reaching convergence.

Once trained, the weights and bias in the network are determined. Then the trained model can predict the classification class of a new image not from the previous dataset using the determined weights. This process is referred to as inference. An illustration of the training and inference can be found in Figure 2.3.

Since inference is often executed on edge, mobile, embedded or some resource-limited devices, we will focus on compressing models during the inference rather than the training process. We will discuss more details later.

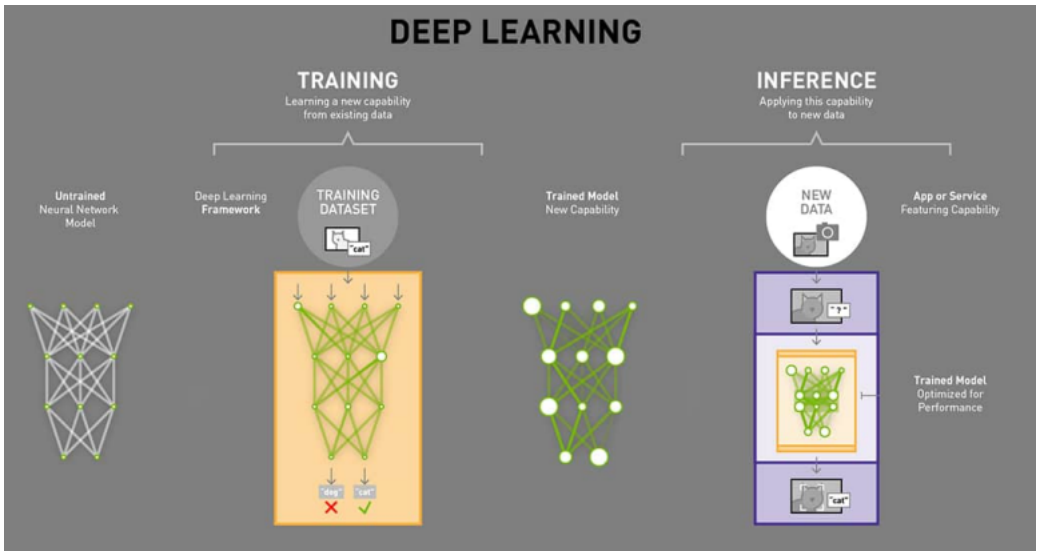


Figure 2.3: Training versus Inference [18]

2.4 Underfitting, Overfitting and Regularization

In machine learning, our goal is to train a model that perform well on "new" data—not the known training data. The ability for a model to perform well on unseen data is called generalization.

The error measured on the training set is called as the training error E_{in}, whereas the test error E_{out} is measured on the test set. The gap between the training and test error is defined as

$$E_{\text{gap}} = E_{\text{out}} - E_{\text{in}} \tag{2.12}$$

We train a model by minimizing its training error E_{in}, but we actually care about the test error E_{out}. Therefore, in practice, our aims are: $1)$. to minimize E_{in}; $2)$. to make E_{gap} small. There are two phenomena in the training process: Underfitting and Overfitting.

Underfitting occurs when the model is simple, and not able to obtain a low error value E_{in}. Overfitting occurs when the gap between training and test error E_{gap} is large. Whether it is

underfitting or overfitting in the training process of a model depends on the capacity of the model. Models with low capacity may be hard to fit the training set, whereas models with high capacity may overfit. Though E_{in} is small enough, the model cannot generalize well on the test set.

Regularization refers to any modifications we make to reduce over-fitting and generalize model when training models with high capacity.

There are many regularization strategies. One of the most common direction is to add an extra term in the loss function that can constrain the parameter values in the model, such as L-1 regularizer and weight decay. Regularization strategies can lead to better generalization and improved performance on new, unknown test data.

2.5 Attention Mechanism

2.5.1 Background

Attention mechanism, a simple method, has been widely applied to and obtained great improvement in many natural language processing(NLP) tasks, including machine translation, sentiment classification, language modeling, text summarization, question answering and so on. Transformer, an NLP network architecture, firstly proposed in [110], is a sequence transduction model based on the attention mechanism.

Unlike recurrent neural networks(RNNs), such as LSTMs, Transformers with attention mechanism can process all input tokens simultaneously, measure the importance among them, and focus on the important parts in a sequence while discarding the irrelevant or useless parts. The model has had great success and led to the development of many outstanding pretrained NLP models such as BERT [24] and RoBERTa [73].

2.5.2 Attention

First we denote some notations for the following description. We use $\mathbf{V} = \{\mathbf{v}_i\} \in \mathbb{R}^{d_v \times n}$ as a sequence of token values. Let $\mathbf{K} = \{\mathbf{k}_i\}$ be a $n \times d$ matrix, which we refer to as a "key" matrix of

$\mathbf{V}$. Each $\mathbf{k}_i \in \mathbb{R}^d$ is an embedding vector attached to corresponding token vector $\mathbf{v}_i$ as the index representation. Denote $\mathbf{Q}$ as a matrix of size $d \times m$, which we refer to as the "query" matrix, where the j^{th} column of the matrix corresponds to the embedding vector of the j^{th} output token.

The attention mechanism comprises of the following three steps:

- **Compatibility Function**

  In the first step, the compatibility function measures the similarity between the source and target tokens. The query vector $\mathbf{q}_j$, as the embedding representation of the j^{th} target token, matches with each source token in the sequence $\{\mathbf{v}_i\}$ based on the compatibility function $a(\mathbf{q}_j, \mathbf{k}_i)$ which outputs a score to measure the similarity between the target and corresponding source token.

  One common form of the compatibility function is the dot-product function. In practice, the compatibility function is measured on the set of queries simultaneously, i.e., $\mathbf{Q}$.

$$a(\mathbf{Q}, \mathbf{K}) = \mathbf{KQ} \tag{2.13}$$

- **Normalization**

  We apply scale and softmax functions to the output of the compatibility function to obtain the attention scores $\mathbf{A}$. See 2.14.

$$\mathbf{A} = softmax(\frac{\mathbf{KQ}}{\sqrt{d}}) \tag{2.14}$$

  where d is the dimension of the key/query vector.

  The attention score of a target token can be treated as the quality of match of it to each source token. We expect that similar source tokens can receive high attention scores and irrelevant tokens receive low scores.

- **Weighted-Sum**

  The normalized attention scores are then used to encode the entire source tokens into the target tokens $\mathbf{O}$.

$$\mathbf{O} = \mathbf{VA} \tag{2.15}$$

For each target token, the model focuses on the important tokens and discard the useless, irrelevant ones. As a result, the attention-based models achieve great success in NLP tasks.

An illustration of the attention mechanism can be found in Figure 2.4.

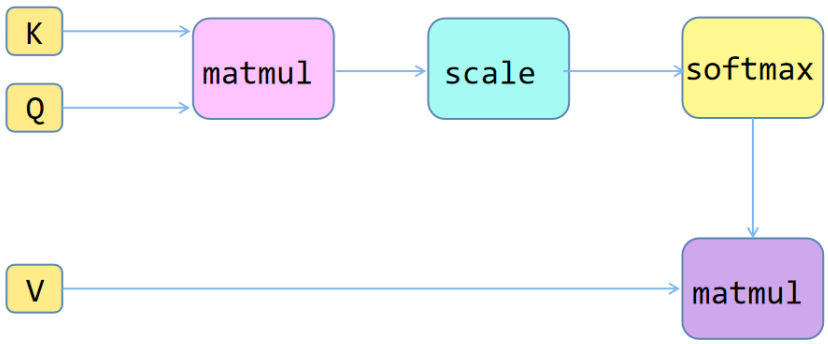


Figure 2.4: Attention mechanism

2.6 Compression of Neural Networks

Over the past decade, deep neural network have achieved significant performance improvements on many real-world tasks, such as computer vision [46] and natural language processing [121]. However, the outstanding performance of these DNNs is achieved by paying the cost of hungry memory consumption and high computational complexity. For example, ResNet-101 [46] requires 200MB memory, and training it usually requires the availability of graphics processing units (GPUs) with high processing parallelism and memory bandwidth. This cost prohibits the usage of DNNs on resource limited devices, especially edge, mobile or some embedded devices. Therefore, the network compression becomes an urgent and promising topic. Compression aims to reduce the model size and complexity without losing much accuracy of the original network. Various compression techniques have been proposed in recent years, widely used for resource saving and computation acceleration.

In this section, we will introduce some common compression approaches used in modern networks: a). Pruning; b). Low-rank Factorization; c). Knowledge Distillation. Quantization techniques will be discussed in the next chapter, chapter 3, in detail.

2.6.1 Pruning

Network pruning attempts to figure out the redundancies of weights and unnecessary parameters in a network, trying to remove the less important parameters and connections in the network while preserving the performance of the original model [43]. Network pruning is popular in compression works since it directly saves model storage and reduce the computational complexity. In some early works [68, 82], network pruning is used to reduce the network complexity and over-fitting.

When pruning a specific model in practice, there are mainly three questions that readers need to ask: (1) what to prune, i.e., pruning structure, (2) which to pruning, that is pruning criterion and (3) how to prune, i.e., pruning schedule.

Parameters can be pruned in some pattern. Based on the shape of the basic pruning element of a pruning algorithm, pruning techniques can be categorized into two types: unstructured and structured pruning [112]. The unstructured pruning performs on an element-by-element basis [72], i.e., no structure at all. The locations of these pruned, noncritical neurons generally are random. Although removing all these unnecessary parameters leads to very little impact on the model performance, this approach results in sparse matrix operations, which are known to be hard to accelerate [13, 28]. On the other hand, for the structured pruning, a group of parameters is removed. This type of pruning approach is popular in convolutional neural networks. We can flexibly choose to prune on a vector-by-vector, channel-by-channel, filter-by-filter, or layer-by-layer basis [72]. Besides, the locations of redundant groups are regular, which makes it easier to perform implementation-friendly algorithms and achieve acceleration. However, structured pruning often leads to significant accuracy degradation [11, 28, 51]. Therefore, a retraining/fine-tuning process is important and necessary for regaining the performance.

[76] explores the performance of pruned models under different pruning structures on ImageNet dataset. Figure 2.5 shows the accuracy curve of sparsity under the settings of unstruc-

tured and structured pruning for AlexNet on ImageNet. “Fine-grained(0D)”, “Vector(1D)”, “Kernel(2D)” and “Filter(3D)” denote individual weights, sub-kernel vectors, kernels, and filters respectively. We can find that when the size of pruning element is large (the “Filter(3D)” case), the accuracy loss is huge. Smaller size of a pruning elements results in higher accuracy. Pruned models with “Vector(1D)” and “Kernel(2D)” can reach similar compression rates unstructured pruning“Fine-grained(0D)”, and provides some advantages of hardware acceleration. Thus in practice, structured pruning is still more popular. Moreover, we can find that after removing over 50% parameters, the compressed models still outperform the uncompressed model in this example.

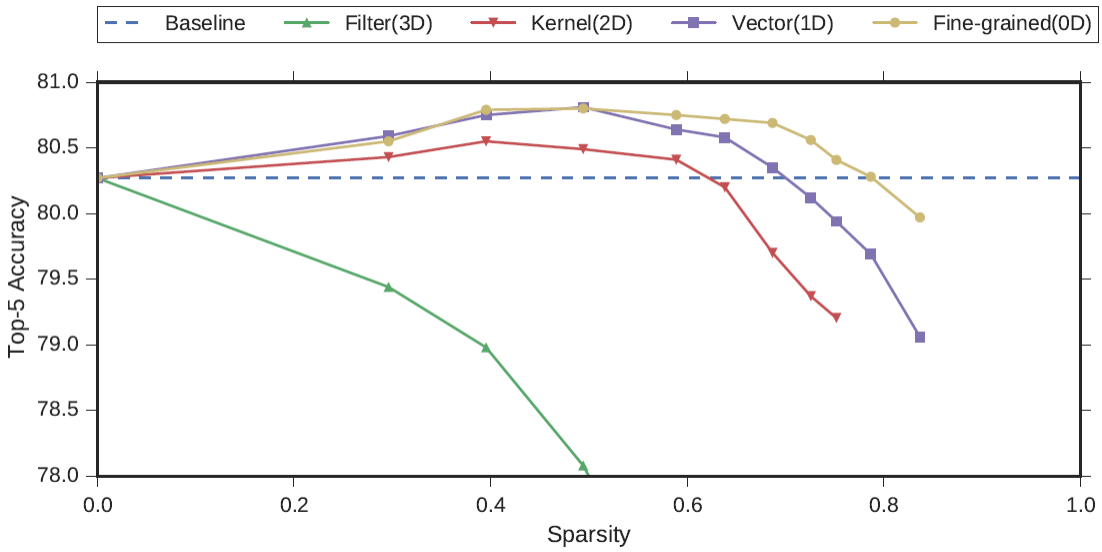


Figure 2.5: Accuracy-Sparity Curve of AlexNet [76]

X-axis: percentage of zero weights, Y-axis: top-5 accuracy on ImageNet

As for the pruning criteria, i.e., selecting which weights to be pruned, there are normally two kinds of strategies. First, magnitude-based pruning. For unstructured pruning [41, 43], the weights with the least absolute values are selected. For structured pruning [4,47,70,116,126], the Frobenius norm (typically L1-norm or L2-norm) of a weight group vector are used as the measurement. This is the simplest and most prevailing criterion in pruning now [112]. In addition, some works choose to prune those weights that induce the least loss change. The importance of weights or blocks can be measured by computing the change in loss or the sensitivity of loss

when the weight or block is removed, and then those causing the least change in loss will be pruned [17]. [68, 82] are two famous works based on this principle. [68] estimate the weight importance by making a local approximation of the loss with a Taylor series and use the second derivative of the loss with respect to the weight as a criterion. [82] measure the importance during training. Each weight in the network is assigned an importance weight α, where $\alpha = 0$ means that the weight becomes redundant, whereas $\alpha = 1$ the weight act as a standard weight. The author compute the loss derivative with respect to α. Weights are then pruned if the derivative falls below a threshold. [5, 117] adopt this idea to structured pruning, assigning an importance weights to weight blocks such as kernels or filters.

With the pruning structure and criterion determined, the next question is about how to prune. The normal training process for pruning consists of three steps: pre-training, pruning and fine-tuning. Since the second step will hurt the model performance due to the information loss,the retraining process is needed to recover the model accuracy.

There are three typical pruning choices [113]:

- One-shot pruning: network sparsity, which is defined as the ratio of zero weights in a network, goes from 0 to a target number in a single step, and then fine-tune.
- Progressive: network sparsity goes from 0 to the target gradually, and then fine-tune.
- Iterative: sparsity goes from 0 to an intermediate number, then fine-tune, and then repeat the process until the target sparsity is achieved.

There is no huge difference between the last two kinds of pruning strategies, thus many works use them interchangeably. Besides, when pruning the same number of weights for a specific model, the last two pruning approaches outperform the one-shot pruning, since they requires longer training time for the model to recover.

In conclusion, pruning is an important technique for compressing neural networks. Unstructured and structured pruning reduces model required storage and computational complexity [72]. An detailed pruning survey [50] shows that today's sparsification methods can lead to a $10-100\times$ reduction in model size, computational complexity and energy efficiency, all without significant loss of accuracy.

2.6.2 Low-rank Factorization

Low-rank factorization is a technique that decomposes the original, higher-rank weight tensor into smaller, lower-rank tensors for efficient computation [57]. It works by breaking a large layer into multiple smaller ones, eliminating the redundant parameters, thereby reducing the model size and accelerating the computation. This technique can be applied to both fully connected and convolutional layers [17].

Performing singular value decomposition (SVD) to the weight tensors in MLPs and CNNs is a common and popular factorization scheme in early works [17, 23, 58, 124]. The original weight matrix is replaced with three smaller matrices. For any given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, there exists a SVD decomposition [7],

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{2.16}$$

where $\mathbf{U} \in \mathbb{R}^{m \times r}$ and $\mathbf{V}^T \in \mathbb{R}^{r \times n}$ are orthogonal matrices and $\mathbf{S} \in \mathbb{R}^{r \times r}$ is a diagonal matrix with the singular values on the diagonal.

[23] performs SVD to the weight tensors in the convolutional and the fully-connected layers, removing approximately 95% weights within the layer and achieving $2\times$ speedup for a single convolutional layer with 1% drop in classification accuracy. Another work [58] reports a $4.5\times$ speedup with 1% accuracy drop in CNNs using SVD. Moreover, [124] proposes a improved SVD method for accelerating non-linear CNN, showing a more accurate result than [58]. All above works perform the factorization technique layer by layer. After one layer is approximated by the low-rank filters, all layers above are fine-tuned for the accuracy reconstruction.

In [64], a new approach-canonical polyadic (CP) decomposition is proposed. They use the non-linear least squares to decompose the 4D weight tensors of convolutional layers into a sum of rank-one tensors, replacing an original layer with a sequence of four layers with rank-one filters. Figure 2.6 illustrates the two different low-rank decompositions for speeding up a generalized convolution in [64].

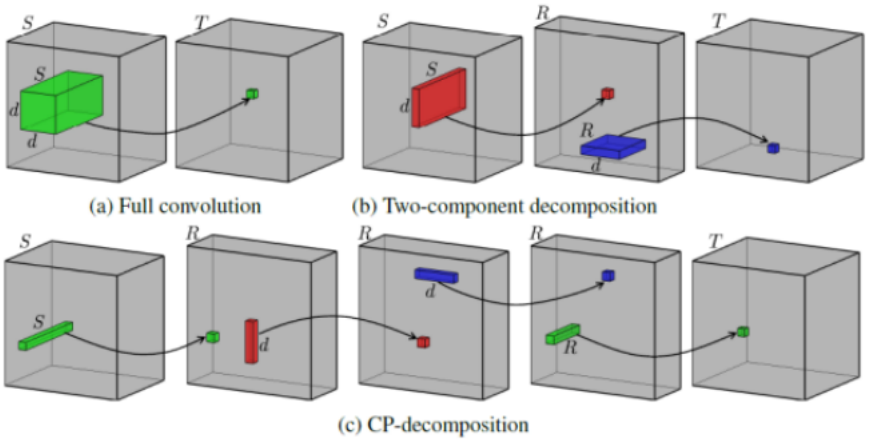


Figure 2.6: Low-rank Factorizations for speeding up a generalized convolution [64].

Gray boxes conrespond to 3D tensors in a convolutional layer. Colored boxes illustrate the original or low-rank approximated filters. Arrows shows linear tranformation mappings. (a) initial full convolution, the original 3D convolutional filter $\in \mathbb{R}^{S\times d\times d}$. (b) Low-rank approximation [58]. The initial convolution is approximated as a composition of two linear mappings, the low-rank filters are $\in \mathbb{R}^{S\times d\times 1}$ or $\in \mathbb{R}^{S\times 1\times d}$. (c) CP decomposition [64]. Approximate the convolution as four convolutions, all decomposed filters are rank-one tensors.

2.6.2.1 Filter Decomposition in Convolution

For the low-rank factorization in convolutional layers, the original filter can be approximated with a lightweight convolution and a linear projection [71]. Current decomposition methods usually operate in some pattern, either directly performing on the 1D column-wise, 2D kernel-wise or 3D filter-wise basis.

We first denote some notations for the following description. The filter of the l^{th} convolution layer is denoted as $\mathbf{W}^l \in \mathbb{R}^{c_{\text{out}}\times c_{\text{in}}\times h\times w}$ where $c_{\text{out}}, c_{\text{in}}$ are the numbers of input and output channels, respectively, and the size of the receptive fiel is $h \times w$.

CP decomposition [64] can be regarded as a column-wise basis factorization, splitting $\mathbf{W}^l$ into multiple rank-one tensors. [58,102,115] directly operate on the 2D kernel-wise $h \times w$ kernels. [58,115] replace the 2D kernels with a linear combination of a smaller 2D filter set. In their work, each original convolutional layer is decomposed into multiple depth-wise convolutional layers and a convolutional layer with 1×1 kernels. [102] combines the decomposition and weight-

sharing techniques to reduce the model size. The author decomposes the original 4D tensors into multiple 2D kernels, considering each kernel as a clustering element and perform k-means algorithm over all 2D kernels. The kernels that fall in the same cluster share the same weights.

[71, 90, 123] directly deal with the 3D $c \times h \times w$ filters, considering the input channel as the third dimension, leading to an impressive reduction in terms of compression rate. However, [123] cannot reduce the input channel, i.e., $c = c_{\text{in}}$, which means that it cannot perform in narrow networks with large c_{in} but much fewer c_{out}, such as DenseNet. [71, 90] decompose each 3D $c_{\text{in}} \times h \times w$ into n groups along input channel dimension, i.e., $c = \frac{c_{\text{in}}}{n}$, then perform convolution on 3D $c \times h \times w$ filters.

Low-rank factorization can support training from scratch and pertrained model. However, the implementation is not that easy since it involves computationally-expensive decomposition operations. Moreover, most methods perform on a layer-wise basis, thus cannot perform global compression.

2.6.3 Knowledge Distillation

Knowledge distillation, as another technique of model compression and acceleration, involves training a teacher model and then train a more compact, smaller student model under the teacher's supervision. The teacher model can be a single large model, or it can be an ensemble of separately trained models [17]. The main objective for the student model is to learn the teacher model's generalization capability and obtain a competitive or even a superior performance while being compact.

[12] first introduces the knowledge transfer concept, compressing the large, complex ensemble models into a smaller and faster model without substantial loss in the performance. The authors claim that the knowledge learned from the bigger ensemble model can be transferred to a smaller one. Later [6] extends this idea. By training a large network first and then transferring the learned knowledge to a small shallow network, the smaller network can achieve the same accuracy. Figure 2.7 shows the a general teacher-student framework for knowledge distillation.

In knowledge distillation, there are mainly three different knowledge types [36]: response-

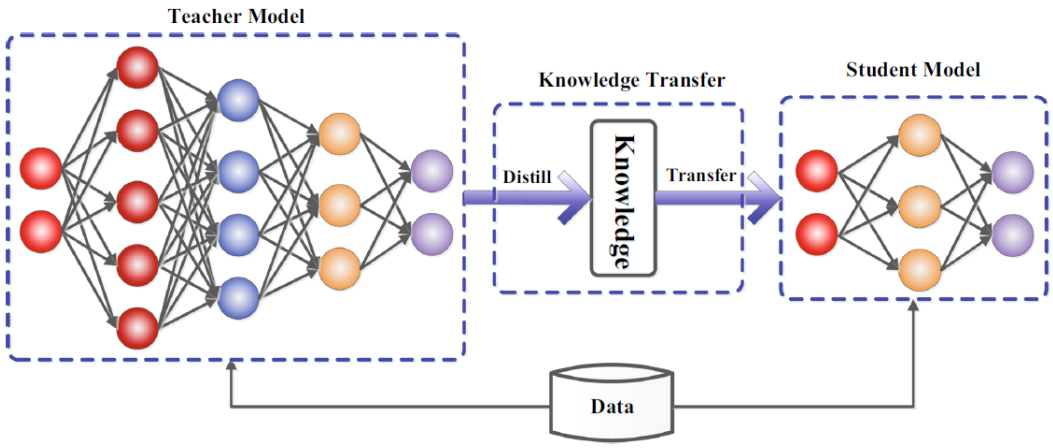


Figure 2.7: The generic teacher–student framework for knowledge distillation [36]

based, feature-based and relation-based knowledge. Response-based knowledge usually refers to the neural response of the last output layer of the teacher model [36]. The main idea is to directly mimic the final prediction of the teacher model. This approach is easy but effective, and has been widely used in different applications. Feature-based approach focus on not only the output of the last layer, but also the output of intermediate layers. It is a good extension of response-based knowledge, especially for the thinner and deeper networks [36]. Both of these two types of knowledge use the output of specific layers, whereas the relation-based knowledge explores the relationships among layers. [120] proposed a flow of solution process (FSP) which is calculated by the inner products between features of different layers.

[48] is commonly known as an important work in knowledge distillation for image classification tasks. The authors chose the response-based knowledge. Instead of using the hard labels (ground-truth label), the authors believe soft labels (predicted probabilities of inputs belonging to image classes) contain more information. Even the probabilities of incorrect answers are very small (close to 0), some of them are still much different from others. The relative probabilities of incorrect answers tell the student model a lot about how the teacher model tends to generalize. For example, an image of a car may have a very small chance of being mistaken for a truck, but that mistake is still many times more than mistaking it for a bird.

The author also introduce the concept of temperature (T) to generate the soft labels. The softmax function with temperature is defined in Equation (2.17), where z_i is the logit for i^{th}

class and q_i is the soft prediction. By raising the temperature, the probability distribution over classes become softer, which means higher entropy in soft labels, so the student can learn more information from the teacher.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{2.17}$$

Figure 2.8 shows the training process in [48]. The large teacher model is first trained on a given training dataset. When the pre-trained teacher model generates the soft labels under the high temperature, the student model is trained under the same high temperature to match these soft labels. The first loss term $\mathfrak{L}_{\text{soft}}$ in Equation (2.18) is the cross entropy between the soft labels and soft predictions. The normal training process is also added for the student using the hard labels under the normal temperature, i.e., $T = 1$. Thus the second loss term $\mathfrak{L}_{\text{hard}}$ is the cross entropy between the ground-truth labels and hard predictions. The overall loss is a weighted sum of two losses, see Equation (2.18). The authors also found that in practice, assigning a relatively lower weight to $\mathfrak{L}_{\text{hard}}$ can obtain the better results.

$$\mathfrak{L} = \alpha \mathfrak{L}_{\text{soft}} + \beta \mathfrak{L}_{\text{hard}} \tag{2.18}$$

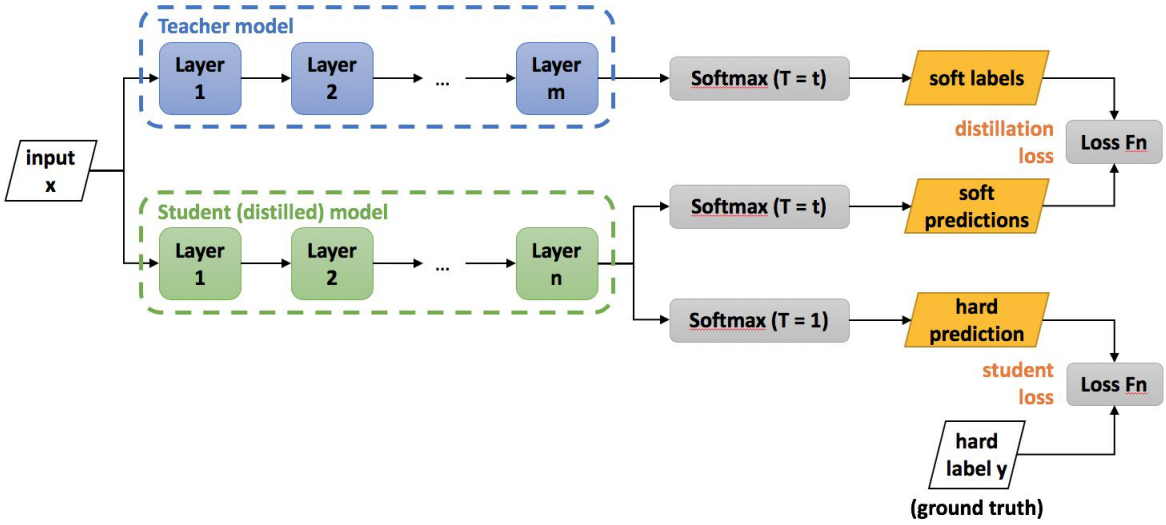


Figure 2.8: The training process in [48]

Knowledge distillation allows small models to obtain good generalization ability. However,

this approach can only be applied to classification tasks with softmax loss function, which hinders its usage. Since this technique is not our focus, the more comprehensive survey can be found in [15, 17, 36, 114].

2.6.4 Quantization

Network quantization is recognized as one of the most effective approaches to compress deep neural networks and has achieved great success in both training and inference of models. It focuses on reducing the number of bits to represent each parameter, storing weights using more compact formats such as integers, low-precision floating numbers or even binary numbers. Quantization provides a solution to greatly reduce the model size and the energy consumption without significantly performance degradation.

While in early works, quantization was used in the digital computing, this concept can be extended to the field of deep neural networks. In this article, we focus on exploring quantization techniques for compressing deep neural networks, and propose a novel quantization approach for compressing deep convolutions. Some common quantization techniques will be discussed in detail in Chapter 3.

Chapter 3

Quantization

In this chapter, we discuss quantization techniques in detail. The organization is summarized as follows: In Section 3.1, we will first provide the background of quantization in digital communication field, including a brief history of quantization, its development in the information theory. And then Section 3.2 discusses this concept in deep neural networks, introducing some common quantization techniques used in recent works.

3.1 Quantization in Digital Communication

Quantization, in mathematics, is a method of mapping input values from a large, often continuous set to output values in a countable, smaller set often with a finite number of elements. It has a long history, dating back to the 1800s. In the oldest work, quantization was used in the approximation of integral calculation. [99] first used rounding for the integration. More recently, quantization became popular in digital signal processing field. Especially in 1948, [97] were published, greatly contributed to quantization theory.

Shannon in [97] discussed the effect of quantization and formally used it in coding theory. The author pointed out that assigning equal number of bits to all quantization cells is wasteful if the cells follow an non-uniform probability distribution. Instead, a lossless coding approach - **variable-rate quantization** - was proposed: to vary the number of bits based on the corresponding cell's probability, specifically, less number of bits assigned to higher probability cells.

Inspired by this idea, Huffman coding was proposed [56]. Later, Shannon's idea was fully developed in [98], where the distortion-rate function and vector quantization concept were published. **Rate-distortion theory** provided the theoretical foundations for lossy data compression, determining the lower bound of rates subject to a signal distortion constraint. If the source can be approximately reconstructed at the receiver without exceeding the expected distortion, then each input symbol should be represented with at least the determined number of bits. **Vector quantization** defined a novel joint quantization approach handling a block of signal values simultaneously, which can achieve rate-distortion limit.

From information theory [20], scalar quantizers which handle weights individually are inferior in their rate-distortion performance and the optimal rate-distortion performance is only achievable via vector quantization. Due to its better performance, vector quantization has become practical in [25, 95] and been widely used in many fields, such as signal and image processing.

In conclusion, quantization as the process of representing a signal in digital form, plays a critical role in digital communication and forms the core of all lossy compression techniques. We just briefly list some of the important points here. A detailed discussion can be found in [39].

3.2 Quantization in Deep Neural Networks

Quantizing neural networks dates back to the 1990s [77, 107]. In the early works, the motivation of quantizing models was that the digital hardware implementation of such networks was easier. For example, by performing the powers-of-two weight quantization approach in [107], the original bulky multipliers were replaced by shift registers, thus the ship ares and computation time were saved. Recently, the research of quantization in deep neural networks focuses on compressing models without performance degradation. Achieving this aim usually calls for joint solutions from machine learning, optimization strategies, hardware design, etc. As discussed before, quantization has not been invented in machine learning, but has been widely used in digital communication field as a compression tool. However, quantization in neural networks is different from it in communication.

In digital communication, the objective is to compress signals with minimal error compared

to the original ones. In other words, we focus on finding a quantization method that would preserve the information in original signals as much as possible. However, things are different in deep neural networks.

First, deep neural networks are in general heavily over-parameterized. Many works [22, 69, 80, 104] have shown that deep neural networks are very robust to aggressive compression. [22] pointed out that in the best cases more than 95% of the parameters in a neural network can be predicted from the remaining 5% without a drop in predictive performance. Second, the objective for the training of neural networks is normally measured by loss function. Hence, in neural networks, instead of trying to preserving as much as possible information of original models, our goal is to find a quantized representation that leads to as small loss as possible. Due to the high degrees of freedom in networks, it is highly possible that there exists some very different models that can simultaneously minimize the loss. Therefore, it is possible for a quantized model to have a huge difference with the original, non-quantized one, while still achieving very good performance.

Recent quantized neural networks have achieved good performance, with similar accuracy to their full-precision counterparts. In this section. We will brief introduce some common quantization techniques. They can be categorized into three types: deterministic, stochastic quantization and weight-sharing. In deterministic quantization, there is a deterministic one-to-one mapping between the full-precision value and the quantized value. While in stochastic quantization, compressed parameters are discretely distributed. The real value is quantized to some discrete value with a stochastic probability. We will introduce a typical deterministic quantization approach - low-bit precision quantization in Section 3.2.1. Then we discuss stochastic quantization in Section 3.2.2. In Section 3.2.3, we focus on weight sharing approaches, which also fall within the quantization area.

3.2.1 Low-bit Precision Quantization

Most neural networks are trained using float32 (FP32) numbers [72]. Low-bit precision quantization aims to convert FP32 parameters to lower bit representations [9], which can significantly

reduce bandwidth, model size, energy and running time. In this subsection, We categorize quantization techniques based on bit-width.

Low-bit precision quantization works were proposed as far back as the 1990s [1, 26]. Recent research interests focus on 8-bit integer (INT8) and half-precision (FP16) quantization. FP16 numbers are widely used in nVidia GPUs and ASIC accelerators [21]. And many works [75, 109] have shown that full-precision (FP32) parameters produced in the training process can be quantized to INT8 numbers for inference without a significant drop in accuracy.

We start from the most special case: extreme low-bit quantization.

3.2.1.1 Extreme Quantization

Extreme low-bit quantization reduces all weights in the network to 1-bit or 2-bit representations [72]. Binarization, drastically reducing the model size by up-to 32×, is the most extreme quantization method. Besides the memory advantages, normal expensive multiplications are replaced with efficient bit-wise operations, thereby binary(1-bit) and ternary(2-bit) operations significantly accelerating the computation and lowering energy consumption. However, reducing 32-bit parameters into a 1 or 2 bit usually would result in significant performance degradation [72].

First, we introduce an early binary network, BinaryConnect [19], which constrains all weights to either $+1$ or -1 in both forward and backward propagation. During the forward pass, the full precision weights w_r are converted into $+1$ or -1 based on the sign function. For the backward pass, due to the non-differentiability of binarization function, the gradient vanishes almost everywhere. Therefore, the common gradient descent algorithm cannot be directly applied to update the binary weights. The common solution is to approximate the gradient by "Straight Through Estimator"(STE) [9]. STE ignores the binarization operation and approximates it with an identify function.

$$\text{Forward Pass:} \qquad w_{\mathrm{b}} = \operatorname{sign}(w_{\mathrm{r}}) = \begin{cases} +1 & w_{\mathrm{r}} \geqslant 0 \\ -1 & \text{otherwise} \end{cases} \tag{3.1}$$

$$\text{STE:} \qquad \frac{\partial w_{\mathrm{b}}}{\partial w_{\mathrm{r}}} = \mathbf{1}_{|w_{\mathrm{r}}\leqslant 1|} = \begin{cases} 1 & -1 \leqslant w_{\mathrm{r}} \leqslant 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$\text{Backward Pass:} \qquad \frac{\partial \mathfrak{L}}{\partial w_{\mathrm{r}}} = \begin{cases} \dfrac{\partial \mathfrak{L}}{\partial w_{\mathrm{b}}} & -1 \leqslant w_{\mathrm{r}} \leqslant 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

[55] extends this idea by binarizing the activations. BinaryNets(BNN) is usually recognized as the first binary neural network. Quantizing both weights and activations to binary values brings the additional benefit of less latency, since lightweight XNOR operations replace the costly multiplications between weights and activations. Experiments show that BNN can achieve good performance on MNIST, SVHN, and CIFAR-10 datasets without a significant accuracy drop compared to full-precision competitors.

Another interesting work Binary Weight Network (BWN) [91] adds a scaling factor $\alpha \in \mathscr{R}^{+}$ to weights and uses $+\alpha$ or $-\alpha$ instead of $+1$ or -1 to represent binary weights. The original full precision weight $\mathbf{W}$ is approximated with $\alpha\mathbf{B}$, where $\mathbf{B}$ is a binary weight matrix and α is chosen to minimize the difference, see Equ.(3.4).

$$\alpha, \mathbf{B} = \arg\min \|\mathbf{W} - \alpha\mathbf{B}\|^2 \tag{3.4}$$

[91] also extends this idea to activations, builds XNOR-Net. Compared to BNN, the additional scaling factor in XNOR-Net compensates for the performance degradation and leads to better accuracy.

3.2.1.2 Logarithmic Quantization

Logarithmic quantization [79] is a kind of non-uniform quantization with a predefined bit-width. Compared to uniform/linear quantization, non-uniform quantization usually can achieve higher accuracy, because the latter could better capture the real weights' distributions by focusing more on important value regions or finding appropriate dynamic ranges [30]. A typical example is the logarithmic quantization where all weights and activations are constrained to be power-of-two values. On the one hand, logarithmic quantization enjoys a good match with the distribution

of weights and activations which is bell-shaped and long-tailed [41]. The technique assigns higher resolution, smaller quantization interval around the mean and less quantization levels at the tail, which matches weights' distribution. On the other hand, quantizing all weights and activations to powers-of-two means that original multiplications are replaced with the cheap bit-shift operations, thus brings in hardware efficiency and significantly reduced training and inference time.

Logarithmic quantization levels in [79] can be defined as

$$\mathbf{Q}(\alpha, b) = \alpha \times \{0, \pm 2^{-2^{b-1}+1}, \pm 2^{-2^{b-1}+2}, \pm 2^{-2^{b-1}+3}, ..., \pm 2^{-1}, \pm 1\} \tag{3.5}$$

where $\mathbf{Q}(\alpha, b)$ is a set of quantization levels, α is the clipping threshold and b is the bit-width. Each original full-precision parameter is clipped into $[-\alpha, \alpha]$ first and then is mapped onto the quantization levels $\mathbf{Q}(\alpha, b)$.

Another interesting work [125] performs weight quantization iteratively, named as Incremental Network Quantization (INQ). In one iteration, a subset of weights are converted into power-of-two format, while the other weights keep full-precision format to preserve ensemble performance. After multiple iterations, most of weights are converted to power-of-two values. Experiment results show that [125] has better performance than [79]. Specifically, a 5-bit low-precision version AlexNet [125] obtains only 0.15% top-1 accuracy drop on the ILSVRC-2012 compared to the full-precision one.

3.2.2 Stochastic Quantization

During the inference, the quantization scheme is usually deterministic. However, some works explore stochastic quantization.

3.2.2.1 Random Rounding

In the deterministic low-bit precision quantization, it is possible that no weights change since the rounding operation may always return the same values especially for the small weights [40]. However, random rounding may allow a neural network to explore more, and provide more opportunities for updating parameters.

In [19], the authors propose the following random rounding function for bianry networks,

$$w_b = \begin{cases} +1 & \text{with probability} \quad p = \sigma(w_r) \\ -1 & \text{with probability} \quad 1-p \end{cases} \tag{3.6}$$

where σ is the hard sigmoid function:

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2})) \tag{3.7}$$

If w_r is positive, then there is a higher probability to quantize it to $+1$, otherwise to -1. The trained Binary network shows 1.18% classification error on the small dataset MNIST, but worse performance on larger dataset CIFAR10.

3.2.2.2 Probabilistic Quantization

Empirical study [41], has shown that weights in a trained neural network often follow some Gaussian-like distributions. Inspired by this observation, researchers try to compress network from a probabilistic perspective. One important direction is to utilize some recent Bayesian methods for network compression [74, 81, 108]. Let $\mathfrak{D}$ be a dataset of N pairs $(x_n, y_n)_{n=1}^N$ and $p(y|x,w)$ be a weighted model that predicts output y given input x and weights w. In a Bayesian neural network, we try to estimate the posterior distribution over weights w given data. The posterior is given by Bayes' rule: (3.8), $p(w)$ is the prior distribution over weights.

$$p(w|\mathfrak{D}) = p(\mathfrak{D}|w)p(w)/p(\mathfrak{D}) \tag{3.8}$$

Computation of the true posterior is in general intractable. One common solution is to use variational inference approaches to approximate it. Assuming that the true posterior distribution is approximated by a distribution $q_\phi(w)$. To find $q_\phi(w)$, we need to optimize the variational parameters ϕ by minimizing the Kullback-Leibler (KL) divergence [63] between the true and the approximated posterior distribution which measures the information divergence of two probability distributions, $D_{\text{KL}}(q_\phi(w)||p(w|\mathfrak{D}))$ [63]. Since $p(w|\mathfrak{D})$ is intractable, the minimization of KL divergence can be approximately converted to maximize the "evidence lower bound"

(ELBO) [2, 108]. The lower bound $\mathfrak{L}_{\text{ELBO}}$ is composed of two terms: the first term of the right-hand side of Equ.(3.9) is the negative of reconstruction error, which measures the approximated model performance and the second term regularizes the approximated posterior to be close to the prior distribution.

$$\mathfrak{L}_{\text{ELBO}}(\phi) = \sum_{n=1}^{N} \mathbb{E}_{q_\phi(w)}[\log p(y_n|x_n, w)] - D_{\text{KL}}(q_\phi(w)||p(w)) \tag{3.9}$$

Compression is directly related to variational Bayesian inference and the minimum description (MDL) principle [49]. MDL is a model selection principle, indicating that the best model class for a set of observed data is the one whose representative permits the shortest coding of the data [8]. In practice, when we train a model according to MDL principle, the objective is to minimize the sum of the cost to describe the model (model complexity cost $\mathfrak{L}_{\text{C}}$) and the misfit between model and data (error cost $\mathfrak{L}_{\text{E}}$) [94]. Many works [37, 49, 52, 111] have shown that the objectives of compression based on MDL sense and variational Bayesian inference are well-aligned. Maximizing the $\mathfrak{L}_{\text{ELBO}}$ minimizes the total cost: $\max \mathfrak{L}_{\text{ELBO}} = \min \mathfrak{L}_{\text{C}} + \mathfrak{L}_{\text{E}}$, leading to an optimal trade-off between short description length of the data and the model.

[2, 108] solve the model compression problem based on variational inference and MDL principle. Authors place a prior which can induce clustering and sparsity (a Gaussian mixture in [108] and a multi-spike-and-slab prior in [2]) on the network weights. Then by retraining model using variational inference, a multi-modal and sparse posterior distribution is obtained.

3.2.3 Weight-Sharing

We have introduced low-bit precision quantization which reduces required resources by constraining the bit-width of model parameters and stochastic quantization. Now, we discuss quantization techniques from another perspective, weight-sharing. The basic idea of is to cluster the weights into groups and all weights in the same group share their values with the centroid of this group. Therefore, redundant weights are replaced with their centroids. Instead of storing the whole full-precision neural network, we represent the compressed model with multiple, discrete, shared values - a small sized codebook, and a set of indices indicating the weight clustering assignments, leading to the reduced storage cost.

Quantization can be categorized into two types:

- Scalar Quantization (Section 3.2.3.1): quantizing a scalar weight at a time.
- Vector Quantization (Section 3.2.3.2): jointly quantizing multiple weights at once; working on weight blocks; benefiting from the correlation induced by the block structure.

From the perspective of quantization codebook, the weight-sharing works can be roughly classified into two categories: fixed codebook and adaptive codebook quantization [40]. In fixed codebook quantization, the weights are quantized into some predefined codebook while in adaptive codebook quantization the codebook is learned from the data based on the applied clustering algorithm, such as K-means [44].

Some low-bit precision techniques we discussed in the previous section can also be divided into this category. For example, based on scalar and fixed codebook quantization strategy, some binary, ternary and power-of-two networks use $[-1,1]$, $[-1,0,1]$ and a set of power-of-two values codebooks separately, to quantize every scalar weight in the networks. However, in this section, we focus on exploring those works on clustering algorithm based vector quantization.

The typical procedure in weight-sharing works is shown as follows:

1. Pre-training: The authors start from training a full-precision model with optimization algorithms, such as stochastic gradient descent (SGD).
2. Clustering: By performing clustering algorithm, such as K-means or EM algorithm, the learned parameters are grouped in blocks, and clustered into several clusters, and the cluster centers are taken as the quantization codebook.
3. Fine-tuning: Each block is replaced by a codeword. This step will hurt the model performance due to the information loss. Therefore, the model is retrained to regain performance and obtain a retained codebook.

Many works [16,41,87,102,118] focus on exploiting different kinds of fine-tuning strategies for compensating for the accuracy loss.

3.2.3.1 Scalar Quantization

In scalar quantization, a scalar weight is quantized from a weight tensor at a time, whereas in vector quantization, multiple weight values are simultaneously quantized. In this example, we perform k-means algorithm to a pre-trained weight tensor of a convolutional layer $\mathbf{W} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times h \times w}$.

With scalar quantization, we perform K-means algorithm over all scalar values W_{ijmn} in $\mathbf{W}$:

$$\min \sum_{i}^{c_{\text{out}}} \sum_{j}^{c_{\text{in}}} \sum_{\text{m}}^{w} \sum_{n}^{h} \sum_{l}^{k} \|W_{ijmn} - c_l\|_2^2 \tag{3.10}$$

where W_{ijmn}, c_l are both scalars and $\mathbf{c} \in \mathbb{R}^{1 \times k}$ is a k-sized codebook. After the clustering, all weights that fall into the same cluster will share the same codeword. So that we can use the codebook to approximate the original network $\hat{\mathbf{W}}$.

$$\hat{W}_{ijmn} = c_z, \quad \text{where} \quad z = \arg\min_{z} \|W_{ijmn} - c_z\|_2^2 \tag{3.11}$$

With scalar quantization, we only need to store the indices and the trained codebook during the inference. Many works [14, 16, 41, 108] compress models through weight-sharing based on scalar quantization. [41] proposed the DeepCompression, a three stage pipeline, including pruning, weight-sharing, and Huffman coding. After compression, memory requirement of network can be reduced by 35× to 49× without accuracy degradation. Figure 3.1 shows the pipline of DeepCompression in [41].

[16] proposed the Entropy-constrained scalar quantization (ECSQ) to improve the weight quantizaiton by taking the the importance of network parameters into account. The authors use the second-order information of the loss function (Hessian matrix) to measure the importance. In the k-means clustering step, the clustering errors are weighted by the corresponding entries in the Hessian matrix that indicate the weight importance. The experiment results in [16] show that this method can achieve nearly the same accuracy level as a full-precision network on ImageNet dataset.

Another interesting scalar quantization work [14] is designed based on a hashnet. Different from other weight-sharing works based on K-means clustering, HashNet uses a low-cost hash

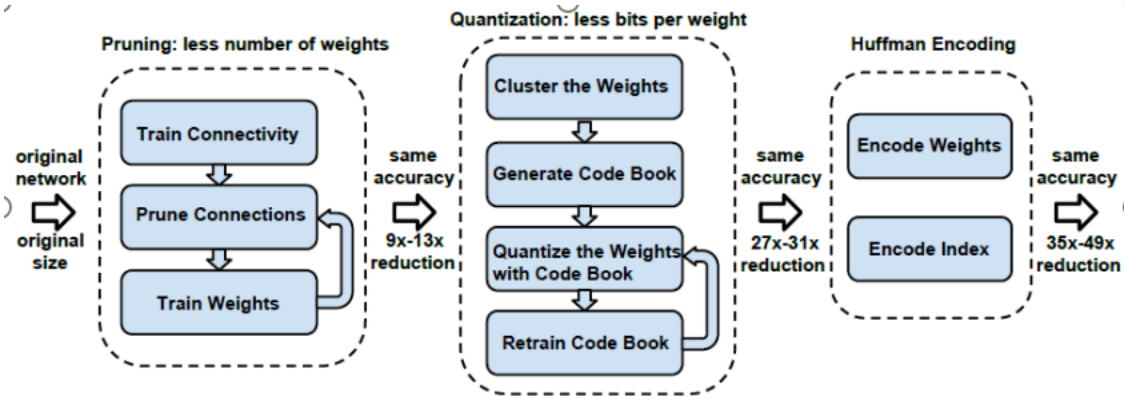


Figure 3.1: DeepCompression: three stage compression pipeline [41]

function to cluster weights before the model sees any training data. Each hash group is replaced with a single floating-point weight value. This approach can be applied to MLP and shallow CNN models and outperforms equivalent-sized networks on small MNIST.

3.2.3.2 Vector Quantization

In vector quantization, several weights are quantized jointly. Compared to scalar quantization, vector quantization normally can achieve lower compression rate by quantizing weight blocks. Moreover, vector quantization provides more flexibility. Due to the high-dimensionality of convolutional filters, we can flexibly select the most suitable shape of quantization blocks through different filter decomposition approaches, such as 1D, 2D, or 3D decomposition methods.

A regular K-means based vector quantization objective can be represented as follows. We suppose that the convolutional filter is split into multiple blocks $\mathbf{b}$ and a codebook $\mathbf{C}$ is a set of k centroids, each with the same size as $\mathbf{b}$.

$$\min \sum_{\mathbf{b}_i \in \mathbf{W}} \sum_{j}^{k} \|\mathbf{b}_i - \mathbf{c}_j\|_2^2 \tag{3.12}$$

Many vector quantization works combines low-rank factorization with weight-sharing approaches. In Section 2.6.2.1, we discussed some common convolution filter factorization approaches. In this section, we introduce three weight-sharing techniques based on 1D, 2D, and 3D factorization blocks.

[118] performs the 1D vector-level decomposition, splitting $\mathbf{W}$ into n columns, where $n = c_{\text{out}} \times c_{\text{in}} \times h$. Each column vector is treated as a clustering element. By applying a regularized k-means algorithm, n column vectors are mapped to a small codebook.

[102] and [71] explores the redundancy in the spatial structure of convolutional filters, either performing 2D kernel-level or 3D filter-level decomposition strategies. [102] applies 2D decomposition approach, treating $\mathbf{W}$ as $c_{\text{out}} \times c_{\text{in}}$ kernels. Then perform K-means over all 2D kernels in the entire network. [71] takes the input channel of the convolutional filters into account. By splitting input channels into s groups, the authors reshapes a 4D weight tensor $\mathbf{W}$ into $s \times c_{\text{out}}$ 3D filters $\mathbf{W}_q \in \mathbb{R}^{c \times h \times w}$, where $c = \frac{c_{in}}{s}$. Instead of performing K-means to extract the codebooks, the authors directly learn them and reconstruct original convolutional filters with the linear combination of the learned basis.

3.2.4 Soft and Hard Quantization

Quantization techniques can also be classified into two groups: hard and soft quantization. In hard quantization works [35,41,87,102,118], each full-precision weight will be quantized exactly to be one value. In soft quantization works, each weight can potentially belong to more than one groups.

[71] adopts the soft weight assignments. Although authors solve the convolution compression problem from the low-rank filter decomposition perspective, we can still treat it as a soft quantization work. Each convolutional filter is represented as a linear combination of shared codewords. A standard convolution is decomposed into a convolution with respect to a codebook and a 1×1 convolution which includes all coefficients of the linear combination. Hence, the corresponding compression rates become much smaller than hard quantization works.

There exists an interesting type of hard quantization works [2, 34, 85, 108, 119]: from soft to hard quantization. [85] introduces a "soft weight-sharing" strategy as a regularization approach for better generalization. By modeling the distribution of weight values as a mixture of multiple Gaussians, weights are encouraged to concentrated around mean values during training. [108] extends this idea for neural network compression using Minimum Description Length (MDL) Principle which we discussed in Section 3.2.2. Starting from a pre-trained model, authors fit

a Gaussian mixture model prior over weights. By performing an EM algorithm, a "softened" K-means algorithm, the distribution of weights is encouraged to be close to K clusters. After training stage, each weight was quantized to the mean of the mixture model component which takes most responsibility. In another Bayesian network compression work [2], the authors use a different prior, "multi-spike-and-slab" prior which has multiple spikes at locations c_k, $k \in 1,..,K$ to learn a multi-modal posterior. After a MDL-based retraining, most weights of low variance are distributed very closely around the quantization target values c_k and can thus be replaced by the corresponding value $c_k$without significant loss in accuracy. Weights of large variance can be pruned.

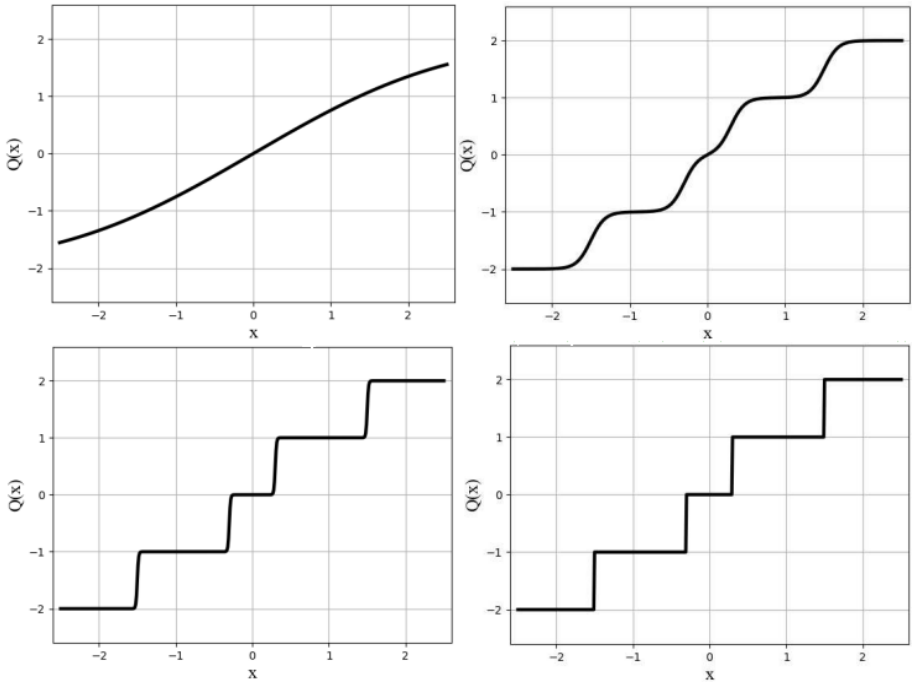


Figure 3.2: The transition process of a soft quantization function during training [119]

$T = 1, 11, 121, 200$ respectively. As T increases, soft quantization function becomes harder and harder until hard quantization is completed.

Apart from these "soft weight-sharing" works, some low-bit precision quantization works [34, 119] adopt a similar training process encouraging the soft quantization to be harder and harder. The motivation of these works is mainly from the prevention from the gradient mismatch problem confronted by many low-bit works. An ideal low-bit quantization operation can be rep-

resented as a combination of several step functions. Since step functions are not smooth, the derivative of quantization functions is zero almost everywhere, which means the training process will be unstable. [119] replaces step functions with smooth, differential Sigmoid functions, Equation (3.13), where T is a temperature factor determining the function shape. Larger T means the smaller difference between the real and approximated quantizaiton functions.

$$\sigma(Tx) = \frac{1}{1 + exp^{-Tx}} \tag{3.13}$$

Thus, in the training process, the author starts from a small T to ensure the quantized models can be well learned. And then T is gradually increased so that the gap between the ideal and Sigmoid functions is narrowing until the expected hard quantization function is obtained. Figure 3.2 illustrates the gradual transition for the quantization functions during training process. [34] replaces Sigmoid with the nonlinear $tanh(Tx)$ function and adopts the similar approach.

Chapter 4

Vector Quantization with Learned Codebook

In this chapter, we develop a novel vector quantization approach to the compression of convolutional neural networks, which we refer to as Vector Quantization with Learned Codebook or VQLC.

Unlike scalar quantization which quantize each network weight individually, vector quantization simultaneously quantizes multiple weights at once. Similar to [71, 102], we choose the similar weight decomposition strategies, namely, considering each channel(2D slice) of the convolutional filters or a 3D group concatenated by several 2D slices along the input-channel dimension as a basic block, and quantizing the weights in each block altogether. Instead of taking a pretraining/clustering approach as in [35, 41, 102], in VQLC, the codebook for quantizing the blocks are learned together with neural network training.

Briefly, VQLC works as follows. The quantization codebook (or dictionary) consists of a prescribed number of learnable codewords, where each codeword is a vector having the same size as the filter blocks considered for quantization. Each filter block is then represented a convex combination of these codewords and the convex-combination coefficients are expressed via a learnable low-rank factorization. In the training process, we devise an additional loss, referred to as the "degeneracy-forcing loss", that forces the convex-combination coefficients to put all probability mass on a single codeword in the dictionary. The degeneracy-forcing loss is used

to train the network together with the standard cross-entropy loss for classification. At the end of training, each filter block can then be taken as the high-probability codeword in the convex combination, and only the index of the codeword is stored.

4.1 Model

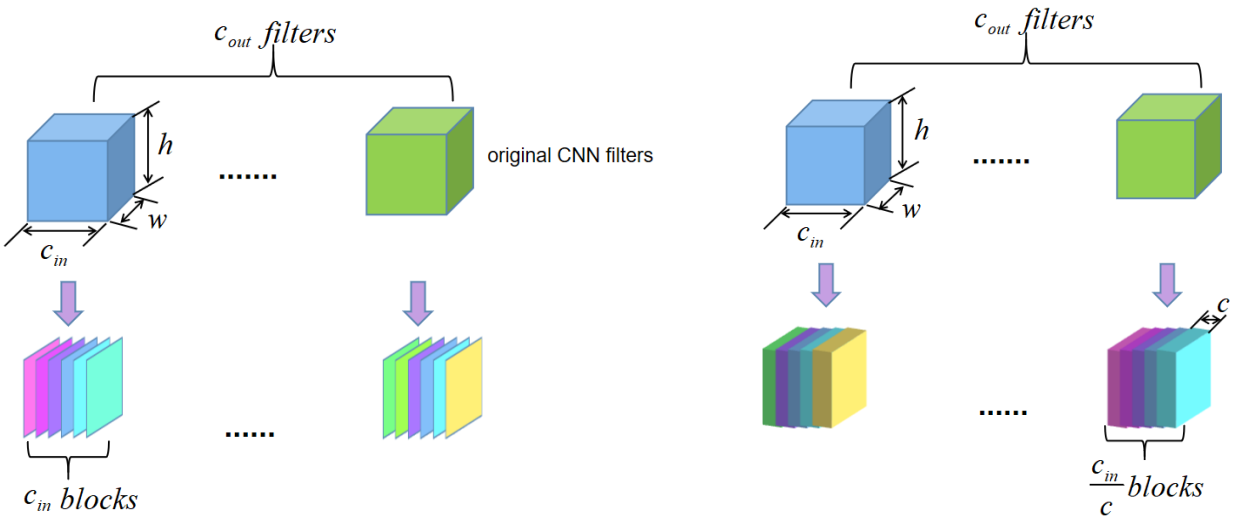
We consider an arbitrary back-bone CNN model, such as VGG [100], ResNet [46], DenseNet [54], that includes L convolutional layers. We will use l to index the convolutional layers, and a quantity denoted with the suprescript (l) indicates that it is in the l^{th} layer.

For the ease of explanation, we consider that the receptive field (RF) of all convolution filers have size $h \times w$, for some h and w. That is, the weights of the l^{th} convolutional layer are structured as a tensor of size $c_{\text{out}}^{(l)} \times c_{\text{in}}^{(l)} \times h \times w$, where $c_{\text{in}}^{(l)}$ and $c_{\text{out}}^{(l)}$ are respectively the number of input channels and the number of output channels in the l^{th} layer. It is possible that $c_{\text{in}}^{(l)}$ (resp. $c_{\text{out}}^{(l)}$ varies with l). Minor modification is needed when the convolutional layers involve multiple RF sizes, which we discuss in Section 4.5.

Each convolutional filter is decomposed into filter blocks of size $c \times h \times w$, where c is a divisor of all $c_{\text{in}}^{(l)}$'s. We say that the blocks are $2D$ blocks if $c = 1$, and $3D$ blocks if $c > 1$. Specifically, if the weights $\mathbf{W}^{(l)}$ is a $c_{\text{out}}^{(l)} \times c_{\text{in}}^{(l)} \times h \times w$ tensor, and $c_{\text{in}}^{(l)} = n \cdot c$ for an integer n. Then for each output channel, the filter (structured as a tensor of size $c_{\text{in}}^{(l)} \times h \times w$) is decomposed into n blocks. We will denote these fitler blocks by $\mathbf{w}_1^{(l)}, \mathbf{w}_2^{(l)}, \ldots, \mathbf{w}_n^{(l)}$. The entire layer then is decomposed into $n \times c_{\text{out}}^{(l)}$ blocks. See Figure 4.1.

With this decomposition, the entire network consists of $\sum_{l=1}^{L} \frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c}$ filter blocks. The assumption in the proposed VQLC method is that these blocks need not to be all different and that they can be taken from a much smaller dictionary of filter blocks.

Let $\mathbf{V}$ be a dictionary (or codebook) consisting of K filter blocks, each having size $c \times h \times w$. We will express $\mathbf{V}$ as $c \times h \times w \times K$ tensor. The overall idea is that we hope by an appropriate training process, each filter block in the CNN will automatically learn to take one of the blocks in the dictionary.



(a) Decomposition into 2D blocks.

(b) Decomposition into 3D blocks.

Left: $2D$ slice-wise decomposition, $c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}$ $2D$ blocks in total. **Right**: $3D$ group-wise decomposition, $\frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c}$ $3D$ blocks in total.

Figure 4.1: Decomposition Approaches

Let $\mathbf{K}$ be a $K \times d$ matrix, which we will refer to as the "key matrix" of the dictionary $\mathbf{V}$. At the l^{th} layer, let $\mathbf{Q}^{(l)}$ be a matrix of size $d \times \frac{c_{\text{in}}^{(l)} \times c_{\text{in}}^{(l)}}{c}$, where the j^{th} column of the matrix corresponding to the embedding vector of the j^{th} filter block in $\mathbf{W}^{(l)}$, which we refer to as the "query" for this filter block.

Let

$$\mathbf{P}^{(l)} := \text{softmax}\mathbf{K}\mathbf{Q}^{(l)} \tag{4.1}$$

Note that $\mathbf{P}^{(l)}$ is a matrix of size $K \times \frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c}$, where each column is a probability vector. Then we express each filter block $\mathbf{w}_j^{(l)}$ in the layer l by

$$\mathbf{w}_j^{(l)} = \mathbf{V}\mathbf{P}^{(l)}[j] \tag{4.2}$$

where $\mathbf{P}^{(l)}[j]$ denotes the j^{th} column of $\mathbf{P}^{(l)}$.

In this model, we randomly initialize the matrix $\mathbf{K}$ and have it fixed during training. Parameter $(\mathbf{V}, \mathbf{Q})$ are made learnable.

Figure 4.2 illustrates the structure of this model.

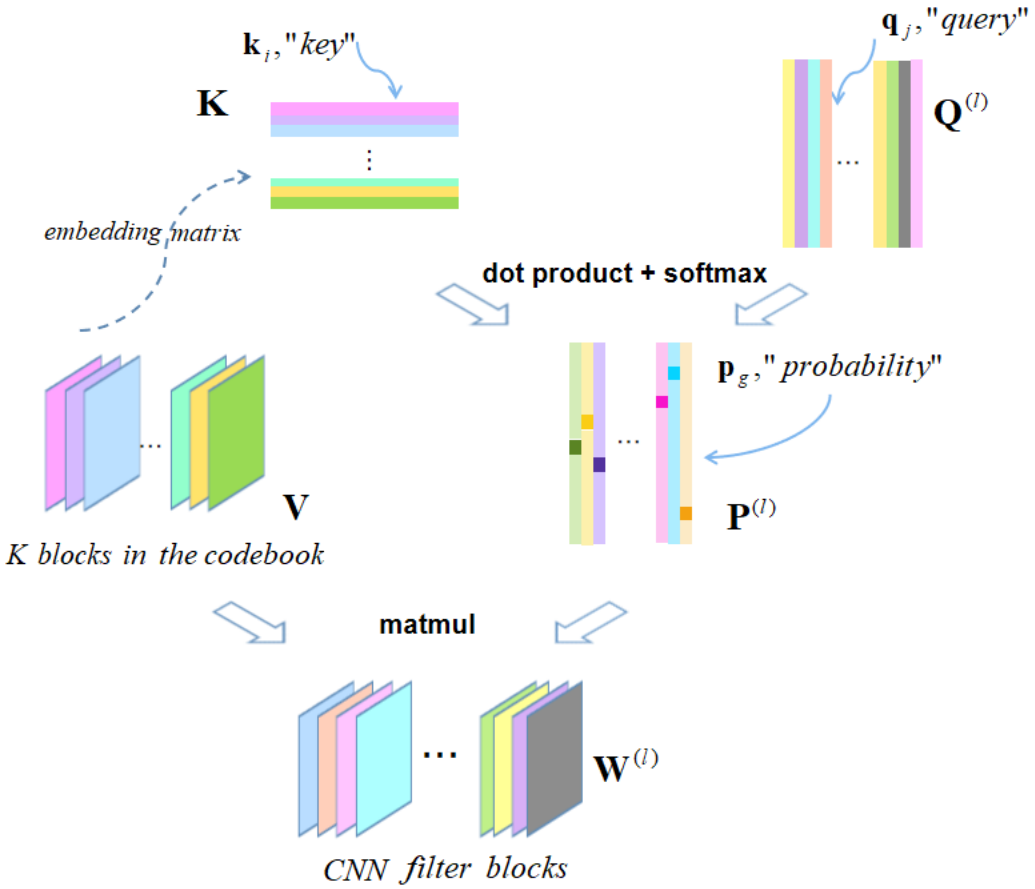


Figure 4.2: CNN Filters in VQLC framework

For the simplicity, we use the 2D slice-wise decomposition in this example. $\mathbf{V}$: K-sized codebook; $\mathbf{K}$: key matrix of the codebook; $\mathbf{Q}^{(l)}$: query matrix for the convolutional filter blocks in the l^{th} layer; $\mathbf{P}^{(l)}$: probability matrix for the l^{th} layer, where the highlighted part in each column indicates that each convolutional filter block eventually selects only one codeword in the codebook; $\mathbf{W}^{(l)}$: convolutional filters in the l^{th} layer.

We remark that the expression of each filter block as a weighted sum of the blocks in the dictionary in (4.2) using an attention module resembles a low-rank decomposition of filrers in [71, 123]. However, an extra mechanism will be introduced during training to force each probability vector in $\mathbf{P}^{(l)}$ to be near a one-hot vector which effectively selects only one block from the dictionary.

4.2 Loss Function

To force each filter block to be taken as a single block in the dictionary, we introduce an additional loss function, which we refer to as the "degeneracy-forcing" loss.

To that end, let $\mathbf{P}^{(l)}[j][k]$ denote the element in the probability vector $\mathbf{P}^{(l)}[j]$ that corresponds to the k^{th} block in the dictionary. The degenercy-forcing loss is then defined as

$$\mathfrak{L}_{\text{DF}} := \frac{1}{\sum_{l=1}^{L} \frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c}} \sum_{l=1}^{L} \sum_{j=1}^{\frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c}} \left(1 - \max_{k=1,\ldots,K} \mathbf{P}^{(l)}[j][k]\right)^2 \tag{4.3}$$

It is easy to verify that when this loss is minimized to its minimum (i.e., 0), each probability vector $\mathbf{P}^{(l)}[j]$ will be forced to the form of a one-hot vector.

In the training process, the model is trained using the degeneracy-forcing loss $\mathfrak{L}_{\text{DF}}$ combined with the cross-entropy loss $\mathfrak{L}_{\text{CE}}$ of the classifier. That is, the overall loss function takes the form

$$\mathfrak{L} = \mathfrak{L}_{\text{CE}} + \lambda \mathfrak{L}_{\text{DF}} \tag{4.4}$$

for some choice of parameter λ.

4.3 Training

The hyper-parameter λ in the overall loss function (4.4) governs the extent to which the degeneracy-forcing loss $\mathfrak{L}_{\text{DF}}$ is minimized relative to the minimization of the cross-entropy loss $\mathfrak{L}_{CE}$. The settings of λ and other hyperparameters (such as the size K of the dictionary, the dimension d of keys, and learning rate) interact and impact the training dynamics in an arguably complex way.

Through extensive investigation, we decided on a two-phase training strategy. The first training phase ("accuracy-oriented training") aims only at improving model accuracy and the second phase ("block-hardening training") aims at forcing each filter block to a single block in the dictionary while maintaining the model accuracy.

- Accuracy-Oriented Training: in this phase, λ is set to 0 for n_0 epochs.

- Block-Hardening Training: in this phase, the value of λ is first set to a small value λ_0 and then is increased by $a\%$ in each epoch.

The training algorithm of VQLC is given in Algorithm 4.1.

Algorithm 4.1 *Vector Quantization with Learned Codebook, VQLC.*

Require: Training dataset $\mathbb{D} = \{(x_i, y_i, i = 1, 2, .., N)\}$, Batch size m, total number of layers in the network L, hyperparameters Θ for gradient decent scheme such as learning rate, weight decay coefficient and momentum

Require: Codebook size K, dimension of keys and queries d, filter block size $c \times h \times w$

Require: Initial trade-off hyperparameter λ_0, the increasing rate $a\%$ for the trade-off hyperparameter

initialize the codebook $\mathbf{V}$

initialize the query tensors in each CNN layers $\mathbf{Q} = \{\mathbf{Q}^{(l)}\}_{l=1}^{L}$

for $n \leftarrow 1$ to N_0 **do** ▷ *Accuracy-Oriented Training*

    $\mathbf{V}, \mathbf{Q} \leftarrow \nabla_{\mathbf{V},\mathbf{Q}} \mathfrak{L}_{\text{CE}}$

end for

$\lambda \leftarrow \lambda_0$

for $n \leftarrow 1 + N_0$ to N **do** ▷ *Block-Hardening Training*

    $\lambda \leftarrow (1 + a\%)\lambda$

    $\mathbf{V}, \mathbf{Q} \leftarrow \nabla_{\mathbf{V},\mathbf{Q}} \mathfrak{L}_{\text{CE}} + \lambda \nabla_{\mathbf{V},\mathbf{Q}} \mathfrak{L}_{\text{DF}}$

end for

4.4 Storing Model and Prediction

After the two training stages, we obtain the optimized codebook $\mathbf{V}^*$ and the corresponding query tensors $\mathbf{Q}^{(l)*}, l = 1, 2, ..., L$. We will then only store the learned codebook $\mathbf{V}^*$ and for each filter block in the network only the index of the block in the codebook that has the highest probability under the resulting $\mathbf{P}^{(l)}$, which we denote by $\mathbf{P}^{(l)*}$.

More precisely, we perform the "einsum" operations between $\mathbf{Q}^{(l)*}$ and $\mathbf{K}$ for each layer l and compute

$$\mathbf{P}^{(l)*} = \text{softmax}\mathbf{K}\mathbf{Q}^{(l)*},\ l = 1,2,...,L \tag{4.5}$$

Let

$$i_j^{(l)} = \arg\max \mathbf{P}^{(l)*}[j],\ j = 1,2,...,\sum_{l=1}^{L} \frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c} \tag{4.6}$$

We will take the filter block $\mathbf{w}_j^{(l)}$ as the ${i_j^{(l)}}^{\text{th}}$ block in the dictionary $\mathbf{V}^*$ and only store the binary representation of $i_j^{(l)}$.

When the stored model is utilized for prediction, we reconstruct the filter block $\mathbf{w}_j^{(l)}$ via

$$\mathbf{w}_j^{(l)*} = \mathbf{V}^* \widetilde{\mathbf{P}}^{(l)}[j],\ l = 1,2,...,L \tag{4.7}$$

where $\widetilde{\mathbf{P}}^{(l)}[j]$ is the "hardened" version of $\mathbf{P}^{(l)*}[j]$ by replacing it with the one-hot vector with 1 located at the ${i_j^{(l)}}^{\text{th}}$ place.

4.5 Additional Considerations

4.5.1 Biases and the First Layer

Through extensive investigation, we decided to keep all biases and the convolutional weights in the first layer at their original precision without further quantization. This presents negligible influences on the rate of compression since there are usually only a small number of input channels (one or three) at the first layer and leaving the filter blocks there intact does not cost a large storage. This choice is motivated by the observation [42, 92, 127] that the first-layer filters of a CNN usually appear quite different from those in the other layers and shall be treated differently from the rest of the network. On the hand, it is observed that converting the network weights in the first layer to a sparse tensor or compressing it with low-bitwidth representation causes severe degradation in the prediction accuracy.

4.5.2 Multiple RF sizes

In the modern CNN architectures, convolutional layers may involves multiple RF sizes. For example, besides the traditional 3×3 RF size, ResNet uses an additional 1×1 layer as "shortcut"

from the previous layer to the next layer. A similar design is also adopted in DenseNet, which contains 1×1 RF in the transition blocks.

Through extensive investigation, we decided a "One Codebook Per RF" strategy, namely, all convolutional layers with the same RF size are made to share the same codebook, and when there are more than one RF sizes, we correspondingly create multiple codebooks, each shared across the entire network. This strategy allows our proposed VQLC approach to easily and flexibly generalize to different CNN architectures.

4.6 Implementation

We now describe some implementation details of the VQLC framework.

4.6.1 Hyperparameter Setup

Algorithm 4.1 lists the hyperparameters required in VQLC framework. We now will show some important hyperparameter settings used in our experiments (Chapter 5).

- K: the codebook size.
  In practice, we chose K from the set $\{64, 128, 256, 512, 1024, 2048\}$.

- d: the dimension of a key or query
  The dimension d of the keys and queries governs the model's capacity to express each convex combination coefficients (or attention weights). In general, larger codebook size K requires a larger dimension d. We examined different settings of (K, d) as listed in Table 4.1.

- c: the number of input channels of a codeword or filter block
  There is a freedom in deciding a decomposition schemes of the CNN filters, namely, either into 2D blocks or into 3D blocks, and in the latter case, there is also the freedom in selecting the number c of input channels. c is a divisor of all $c_{\text{in}}^{(l)}$'s in the network. It is possible to adjust the decomposition strategy to arrive at a different rates of compression. The settings

of c for each network used in our experiments are summarized in Table 4.2. Note that $c = 1$ corresponds to decomposition into 2D blocks.

K	64	128	256	512	1024	2048
d	40	45	50	55	150	200

Table 4.1: Choices of (K, d)

Network	c
VGG-16	$\{1, 64, 128\}$
ResNet-56	$\{1, 4, 8, 16\}$
DenseNet-12-40	$\{1, 4, 12\}$
WideVGG-16	$\{1, 128, 256\}$

Table 4.2: Choices of c for different networks

4.6.2 Initialization of $\mathbf{K}, \mathbf{Q}^{(l)}, \mathbf{V}$

Appropriate weight initialization not only prevents some vanishing or exploding gradient problems but also results in faster convergence of deep neural networks. In VQLC, we initialize the codebook $\mathbf{V}$, key $\mathbf{K}$ and query tensors $\mathbf{Q}^{(l)}$ all from normal distributions, while also investigating various strategies in setting up the standard deviation (std) of the normal distribution, namely, random, Xavier [32], Kaiming initialization [45].

1. **Random initialization**:

   Random initialization is the most common initialization approach which randomly chooses parameters from the standard normal distribution, $\mathcal{N}(0, 1)$.

2. **Xavier initialization**:

   Xavier initialization [32] sets a layer's parameters to values chosen from a normal distribution $\mathcal{N}(0, std^2)$ where

$$std = \frac{2}{\sqrt{fan\_in + fan\_out}} \tag{4.8}$$

where fan_in is the number of incoming network connections to the layer, and fan_out is the number of outgoing network connections from that layer.

3. **Kaiming initialization**:

   There are 2 modes to choose in Kaiming initialization [45]: 1).fan_in, 2).fan_out. Kaiming initialization samples parameters from the distribution $\mathcal{N}(0, std^2)$, where

$$std = \sqrt{\frac{2}{(1+a^2) \times fan\_mode}} \tag{4.9}$$

   a is the negative slope of the rectifier used after the layer, 0 for Relu by default, fan_mode is either fan_in or fan_out.

We compare the performance of these three initialization approaches. Through extensive investigation, the last two approaches, commonly outperform in many tasks, dose not perform well, whereas random initialization works best. Then we adjust the variance std^2 of the normal distribution. The settings of fine-tuned hyperparameter std used in our experiments are listed in Table 4.3.

tensor	std
$\mathbf{Q}^{(l)}$	0.5
$\mathbf{K}$	1.5
$\mathbf{V}$	1.0

Table 4.3: Choices of std for different tensors

4.6.3 Implementation of a VQLC Layer

Rather than taking the existing convolutional layers in the standard deep learning libraries (e.g. TensorFlow or PyTorch), the proposed VQLC approach requires a redesign of each convolutional layer, which we refer to as a VQLC layer. Specifically, we use PyTorch framework [89] to design

each VQLC layer, which takes $\mathbf{K}, \mathbf{V}$ and the output of the previous layer as inputs and generates activation and degeneracy-forcing loss as outputs, details given below.

The learnable $\mathbf{V}$ and fixed parameter $\mathbf{K}$ are set to be global and fed into each VQLC layer. The convolutional filter block in the VQLC layer is computed by a convex combination of all codewords in $\mathbf{V}$, where the coefficient assigned to each codeword is computed by a compatibility function(Equ.(4.1)) of the learnable query of the block with the key of the codeword. The degeneracy-forcing loss of each layer is calculated by Equ.(4.3) based on $\mathbf{P}^{(l)}$, and then is utilized for the overall loss.

Due to the automatic differentiation module [88] in Pytorch, our model can automatically collect and calculate the parameters' gradients. In the backward pass, the model updates the trainable parameters $\mathbf{V}$ and $\mathbf{Q} = \{\mathbf{Q}^{(l)}\}_{l=1}^{L}$ based on the standard stochastic gradient decent (SGD) learning algorithm with the gradients. Since we perform weight-sharing over the entire network, every codeword in the codebook $\mathbf{V}$ will take gradients from multiple layers.

Our proposed VQLC approach can be effectively and easily embedded into different network architectures and compress any RF-sized convolutional layers with flexible choice of rate of compression.

Chapter 5

Experiments

In this chapter, we evaluate the proposed VQLC method on a variety of network architectures over several popular image-classification datasets and compare with some state-of-the-art quantization techniques. Our experimental results indicate that our approach is at least comparable, often superior to other techniques. Especially for the wide networks, VQLC shows the obvious advantages over other approaches at high rates of compression.

5.1 Datasets

We evaluate our method on four common datasets for image classification tasks: MNIST [65], CIFAR10 [60], CIFAR100 [60], SVHN [84].

1. MNIST [65]

   MNIST is one of the most popular deep learning datasets, consisting of grayscale images of handwritten digits, which is commonly used for training various image processing systems. The database contains $70,000$ 28×28 images in 10 classes, including $60,000$ training images and $10,000$ testing images.

2. CIFAR-10 and CIFAR-100 [60]

   The CIFAR-10 dataset is another widely used dataset, consisting of 60,000 32×32 color

images in 10 classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 50,000 training images and 10,000 test images.

CIFAR-100 is similar to CIFAR-10 dataset, except it has 100 classes, each containing 600 32×32 color images. The 100 classes are grouped into 20 superclasses. Each image has a fine label (class) and a coarse label (superclass).

3. SVHN [84]

   SVHN is a real-world image dataset, collected from house numbers in Google Street View images. It is similar to the MNIST dataset, but has more labeled data, 630,420 32×32 color images of printed digits (from 0 to 9). SVHN has three sets: 73,257 images in the training set, 26,032 in the testing set and an extra training set with 531,131 images that are less difficult and can be used in the training process.

5.2 Models

We apply our proposed method to common CNN architectures: VGG-16 [100], ResNet-56 [46], and DenseNet-12-40 [54].

We apply VQLC to various modern CNN architectures including VGG-16 [100], ResNet-56 [46] and DenseNet-12-40 [54]. In order to explore the influence of model width on the performance of VQLC, we add some experiments on "WideVGG-16", in which the width of every convolutional layers is doubled compared to the standard VGG-16 model.

- VGG-16 [100]

  VGG-16 model contains 13 convolutional layers with 3×3 kernels, 5 max pooling layers distributed in the whole convolutional block and 1 fully connected layer followed by a softmax layer for output at the end of the network. The convolutional layers are relatively wide, the width starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer until reaching 512. VGG-16 architecture contains $1,634,496$ 3×3 2D kernels.

- WideVGG-16

  WideVGG-16 follows the most settings of VGG-16 architecture, except that the width doubles in each convolutional layer. Starting from 128, the width increases by a factor of 2 until reaching 1024. The model consists of $6,537,600$ 3×3 2D kernels.

- ResNet-56 [46]

  ResNet-56 contains 55 convolutional layers with 3×3 kernels and 1 fully connected layer. The model consists of 3 parts, each with 9 residual blocks. Each residual block includes two 3×3 convolutional layers with the same number of output channels, a batch normalization layer and a ReLU activation function. Note that for the projection shortcut, we use the maxpooling instead of 1×1 convolution operation to adjust the input shape. A $16 \times 3 \times 3$ convolutional layer is placed at the top of the network, then followed by 3 residual block groups with increasing output channels $\{16, 32, 64\}$. The model ends with a average pooling and a fully-connected layer. The baseline ResNet-56 has 94,256 3×3 kernels in total.

- DenseNet-12-40 [54]

  There are 3 dense block groups, each with 12 dense blocks, and 2 transition blocks. In each dense block, all RF sizes are 3×3, the number of output channels are fixed to 12 and thus the number of input channels starts from 24, gradually increasing with a step 12. Each transition block contains a 1×1 convolutional layer, evenly distributed in the model. The baseline contains 101,160 3×3 kernels and 125,568 1×1 kernels.

5.3 Performance Metrics

We evaluate VQLC and other network compression methods using Top-1 Accuracy and Compression Rate γ.

5.3.1 Classification Accuracy

We use Top-1 Classification Accuracy, defined below, to evaluate the performance of compared compression models on the testing set.

$$\text{Classification Accuracy} = \frac{n_0}{N} \times 100 \tag{5.1}$$

where n_0 is the number of correct predictions, N is the total number of data pairs.

5.3.2 Rate of Compression

When evaluating compression rates, we only consider the compression of convolutional layers, thus only storage required for representing the convolutional layers is counted. Moreover, motivated by the observation in [72], we keep all biases and the first layer uncompressed, which results in negligible influences on the compression rate but maintains a higher accuracy. Similar to the approach taken in [78, 103], we train our VQLC models in full precision (32-bit floats) but store our final codebook in half precision (16-bit floats). In our experiments, this leads to a drop of validation accuracy by less than 0.01%. During the inference, VQLC stores the codebook as well as the codeword indices (using $\log_2(K)$ bits per index). The compression rate γ is defined as the ratio of the bits required for storage before compression to that after compression, the expression of which is given in Equation (5.2) (for case when all convolutional layers use the the same RF).

$$\gamma = \frac{32 \times \sum_{l=1}^{L} c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)} \times w^{(l)} \times h^{(l)}}{16 \times Kcwh + \sum_{l=1}^{L} \frac{c_{\text{in}}^{(l)} \times c_{\text{out}}^{(l)}}{c} \times \log_2(K)} \tag{5.2}$$

Notably, the 2D decomposition scheme ususally requires a large codebook to achieve a good accuracy, and storing the codeword indices may dominate the required bits after compression. In the 3D decomposition schemes, particularly with high value of c, bit budget is shifted mostly for storing the codewords instead.

5.4 Experimental Details

We show the experimental results in this section, and compare with several start-of-the-art quantization strategies.

5.4.1 General Procedures

First, we apply VQLC to several popular CNN architectures, VGG [100], ResNets [46], and DenseNets [54] with various decomposition approaches and different-sized codebooks, and evaluate their performance on several popular classification benchmarks MNIST [65], CIFAR10 [60], CIFAR100 [60] and SVHN [84]. Then we compare our models with some state-of-the-art competitors at the same level of the rates of compression.

We normalize all images in each dataset using channel-wise mean and standard deviation of the corresponding training set. Then two additional geometrical transformation augmentation strategies including random crops and horizontal flips [61] are adopted in the training process. SGD is utilized as the optimizer with momentum 0.9 and a step-wisely decaying learning rate. Specifically, the initial learning rate is 0.1 and then we reduce it by 10 after 50% and 75% of the training epochs. For the regularization approaches, dropout rate and weight decay are set to 0.1 and 5×10^{-5} respectively for all compressed models. We train each model with 50 samples per batch.

The training process is divided into two stages. In the first training phase ("accuracy-oriented training"), λ is set to 0, we train each VGG-16 and WideVGG-16 model for 400 epochs with three learning rates, whereas ResNet-56 and DenseNet-12-40 models are trained for 200 epochs with the same learning rate 0.1. In the second phase ("block-hardening training"), λ_0[1] is set to 0.1 (for VGG-16 and WideVGG-16) or 0.2 (for ResNet-56 and DenseNet-12-40), and then is increased by $a\%$[2] in each epoch. For the VGG-16 and WideVGG-16 models, $a\%$ is initialized as 1% for the first 300 training epochs and then set to 0.3%. For the cases of ResNet-56 and DenseNet-12-40, $a\%$ is set to 2% for the first 100 epochs and then to 0.2%.

[1] An insensitive hyperparameter.

[2] A sensitive hyperparameter, the higher value may hurt model performance whereas a lower one results in slow convergence and long training time.

5.4.2 Compressing CNNs with VQLC

We first show the performance of VQLC on some common CNN architectures, VGG-16 [100], WideVGG-16, ResNet-56 [46], and DenseNet-12-40 [54], for image classification tasks. Some results are summarized in Table 5.1. Note that these results are selected based on an "accuracy-centric" criterion, namely, for a good accuracy, we demonstrate the compression performance of VQLC.

It is clear that for each classification task, VQLC maintains a high accuracy at high compression rates. The results in the bold font indicate the cases where VQLC even improves classification accuracy over the uncompressed models. For example, on MNIST dataset, our compressed VGG-16 model beats the baseline at 94.5$\times$ compression rate.

Model	Dataset	Codebook	Top-1 Accu./Baseline(%)	Compression Rate($\times$)
ResNet-56	CIFAR10	C1K1024	93.35/93.72	23.3
	CIFAR100	C1K1024	71.51/74.01	23.3
	SVHN	C1K256	96.8/97.2	31.3
	MNIST	C1K128	**99.68**/99.54	36.0
DenseNet-12-40	CIFAR10	C1K512-C1K1024	93.87/94.74	14.4
	CIFAR100	C1K512-C1K1024	73.24/75.58	14.4
	SVHN	C4K128-C1K256	96.76/97.253	25.3
	MNIST	C4K128-C1K256	99.62/99.66	25.3
VGG-16	CIFAR10	C128K1024	93.92/94.02	20.7
	CIFAR100	C128K1024	72.09/74.21	20.7
	SVHN	C128K512	**97**/96.973	35.5
	MNIST	C128K64	**99.61**/99.5	94.5
WideVGG-16	CIFAR10	C256K512	**94.59**/94.59	56.0
	CIFAR100	C128K2048	74.86/76.06	35.6

Table 5.1: Some results of VQLC on different datasets.

"K" and "C" in the "Codebook" column denote the codebook size and number of input channels of each block."C1" indicates the 2D decomposition scheme. For DenseNet models, the settings of 3×3 and 1×1 codebooks are placed before and after the dash "-", respectively.

5.4.2.1 Experiments on CIFAR10

The experiment results of VQLC for different models on CIFAR10 are shown in Table 5.2, Table 5.3, Table 5.4.

5.4.2.1.1 VGG-16

We found that 3D decomposition approach performs better than 2D quantization. Especially when we use the "C128K1024" codebook, the quantized model only loses 0.1% accuracy compared with the uncompressed model at 20.71× compression rate. Moreover, as we expect, with the same decomposition approach, using larger codebooks results in higher accuracy.

Model	Top-1 Accu.(%)	Compression Rate(×)
VGG-16-baseline	94.02	—
VGG-16-C1K128	90.3	40.42
VGG-16-C1K256	91.11	35.399
VGG-16-C1K512	92.23	31.45
VGG-16-C128K128	92.45	76.225
VGG-16-C128K256	93.3	55.077
VGG-16-C64K1024	93.25	47.644
VGG-16-C128K512	93.69	35.455
VGG-16-C128K1024	**93.92**	20.72

Table 5.2: Results of VQLC for compressing VGG-16 on CIFAR10

5.4.2.1.2 ResNet-56

Compared with models using 3D codebooks, 2D decomposition strategy in ResNet-56 can obtain better performance. Especially when we use the codebook with 512 codewords, the model at 27.2× compression rate performs even better than the model with the 3D codebook at 10.6×.

Moreover, the performance of compressed models with 3D decomposition not only depends on the codebook size K, but also the number of input channels of each block, C. It is possible that a model with a small K at the lower compression rate outperforms a model with a large K.

Model	Top-1 Accu.(%)	Compression Rate(×)
ResNet-56-baseline	93.72	—
ResNet-56-C1K64	91.73	41.65
ResNet-56-C1K128	91.85	35.96
ResNet-56-C1K256	92.49	31.3
ResNet-56-C1K512	**93.12**	27.2
ResNet-56-C16K512	91.85	20.75
ResNet-56-C4K2048	92.19	17.93
ResNet-56-C16K1024	92.79	10.9
ResNet-56-C8K2048	92.51	10.6

Table 5.3: Results of VQLC for compressing ResNet-56 on CIFAR10

5.4.2.1.3 DenseNet-12-40

In DenseNets, models with larger larger size obtain higher accuracy.

Model	Top-1 Accu.(%)	Compression Rate(×)
DenseNet-12-40-baseline	94.74	—
DenseNet-12-40-C1K256-C1K256	93.28	17.42
DenseNet-12-40-C1K512-C1K256	93.48	16.24
DenseNet-12-40-C1K512-C1K512	93.55	15.27
DenseNet-12-40-C1K512-C1K1024	**93.77**	14.39
DenseNet-12-40-C12K64-C1K256	92.09	27.19
DenseNet-12-40-C4K128-C1K256	92.64	25.33
DenseNet-12-40-C12K64-C1K512	92.1	24.58
DenseNet-12-40-C4K128-C1K512	93.06	23.05

Table 5.4: Results of VQLC for compressing DenseNet-12-40 on CIFAR10

5.4.2.2 Experiments on CIFAR100

5.4.2.2.1 VGG-16

We adopt 3D decomposition for VGG-16 networks on CIFAR100 dataset. The model at 11.31× rate of compression achieves 1.27% drop in accuracy compared with the baseline.

Model	Top-1 Accu.(%)	Compression Rate(×)
VGG-16-baseline	74.21	—
VGG-16-C64K1024	68.21	47.644
VGG-16-C128K512	69.21	35.455
VGG-16-C64K2048	69.45	24.34
VGG-16-C128K1024	71.5	20.72
VGG-16-C128K2048	**72.94**	11.31

Table 5.5: Results of VQLC for compressing VGG-16 on CIFAR100

5.4.2.2.2 ResNet-56

2D decomposition approach is more suitable than 3D decomposition for ResNet models.

Model	Top-1 Accu(%)	Compression Rate(×)
ResNet-56-baseline	74.01	—
ResNet-56-C1K128	70.02	36.0
ResNet-56-C1K256	70.17	31.3
ResNet-56-C1K512	71.1	27.2
ResNet-56-C1K1024	**72.1**	23.3
ResNet-56-C16K512	66.6	20.75
ResNet-56-C4K2048	67.04	17.93
ResNet-56-C16K1024	67.19	10.9
ResNet-56-C8K2048	68.55	10.6

Table 5.6: Results of VQLC for compressing ResNet-56 on CIFAR100

5.4.2.2.3 DenseNet-12-40

For 3D decomposition, models with "C4" codebooks perform better.

Model	Top-1 Accu.(%)	Compression Rate(×)
DenseNet-12-40-baseline	75	—
DenseNet-12-40-C1K256-C1K256	73.01	17.42
DenseNet-12-40-C1K512-C1K256	72.9	16.24
DenseNet-12-40-C1K512-C1K512	73.05	15.27
DenseNet-12-40-C1K512-C1K1024	**73.41**	14.39
DenseNet-12-40-C12K64-C1K512	70.08	24.58
DenseNet-12-40-C4K128-C1K512	71.2	23.05
DenseNet-12-40-C4K256-C1K512	71.99	21.57
DenseNet-12-40-C12K128-C1K512	70.68	22.59

Table 5.7: Results of VQLC for compressing DenseNet-12-40 on CIFAR100

5.4.2.3 Experiments on SVHN

5.4.2.3.1 VGG-16

Most of our quantized VGG-16 models outperform the baseline on SVHN dataset. In particular, the model at 35.5× compression rate achieves 0.087% accuracy rise compared to the uncompressed baseline.

Model	Top-1 Accu.(%)	Compression Rate(×)
VGG-16-baseline	96.973	—
VGG-16-C1K128	**97.04**	40.42
VGG-16-C1K256	96.85	35.4
VGG-16-C128K128	96.854	76.225
VGG-16-C128K256	**96.99**	55.077
VGG-16-C128K512	**97.06**	35.455

Table 5.8: Results of VQLC for compressing VGG-16 on SVHN

5.4.2.3.2 ResNet-56

Models with 2D decomposition approach on ResNet-56 can achieve relatively high compression rates while preserving the model performance.

Model	Top-1 Accu.(%)	Compression Rate(×)
ResNet-56-baseline	97.23	—
ResNet-56-C1K64	96.55	41.65
ResNet-56-C1K128	96.773	36.0
ResNet-56-C1K256	96.804	31.3
ResNet-56-C1K512	96.89	27.2
ResNet-56-C16K512	96.696	20.75
ResNet-56-C4K2048	96.735	17.93
ResNet-56-C16K1024	**97.01**	10.9
ResNet-56-C8K2048	96.9	10.6

Table 5.9: Results of VQLC for compressing ResNet-56 on SVHN

5.4.2.3.3 DenseNet-12-40

The DenseNet at 15.27× rate of compression obtains 96.923% accuracy on SVHN.

Model	Top-1 Accu.(%)	Compression Rate(×)
DenseNet-12-40-baseline	97.253	—
DenseNet-12-40-C1K256-C1K256	96.754	17.42
DenseNet-12-40-C1K256-C1K512	96.776	16.31
DenseNet-12-40-C1K512-C1K512	**96.923**	15.27
DenseNet-12-40-C12K64-C1K256	96.59	27.19
DenseNet-12-40-C4K128-C1K256	96.759	25.33
DenseNet-12-40-C12K64-C1K512	96.61	24.58
DenseNet-12-40-C4K128-C1K512	96.72	23.05

Table 5.10: Results of VQLC for compressing DenseNet-12-40 on SVHN

5.4.2.4 Experiments on MNIST

5.4.2.4.1 VGG-16

All quantized VGG-16 models outperform the baseline on MNIST dataset at the extremely high rate of compression.

Model	Top-1 Accu.(%)	Compression Rate(×)
VGG-16-baseline	99.5	—
VGG-16-C128K64	**99.61**	94.5
VGG-16-C128K128	99.6	76.225
VGG-16-C128K256	99.61	55.077
VGG-16-C128K512	99.61	35.455

Table 5.11: Results of VQLC for compressing VGG-16 on MNIST

5.4.2.4.2 ResNet-56

Compressed ResNet-56 models also perform better than the uncompressed baseline on MNIST dataset at high compression rates.

Model	Top-1 Accu.(%)	Compression Rate(×)
ResNet-56-baseline	99.54	—
ResNet-56-C1K64	99.64	41.65
ResNet-56-C1K128	99.64	35.96
ResNet-56-C1K256	99.67	31.3
ResNet-56-C1K512	**99.7**	27.2

Table 5.12: Results of VQLC for compressing ResNet-56 on MNIST

5.4.2.4.3 DenseNet-12-40

Model	Top-1 Accu.(%)	Compression Rate(×)
DenseNet-12-40-baseline	99.66	—
DenseNet-12-40-C1K256-C1K256	99.59	17.42
DenseNet-12-40-C1K256-C1K512	99.62	16.31
DenseNet-12-40-C1K512-C1K256	**99.63**	16.24
DenseNet-12-40-C1K512-C1K512	99.63	15.27

Table 5.13: Results of VQLC for compressing DenseNet-12-40 on MNIST

5.4.2.5 The trade-off between accuracy and compression rate

Figure 5.1 and Figure 5.2 show how VQLC trades in accuracy for higher rate of compression using CIFAR10 and SVHN as examples. In general, when the size of filter blocks is fixed, increasing the codebook size is expected to improve the accuracy. This trend is observed clearly in VGG-16 and DenseNet-12-40. Specifically, in the VGG-16 models, adopting 3D decomposition

scheme achieves high compression rates ($21\times \sim 76\times$) with a small sacrifice of accuracy. In the case of DenseNet-12-40, 3D decomposition schemes achieve higher compression rates while the accuracy decreases more rapidly.

The visible non-monotonic behaviour of ResNet-56 is due to the fact that the left-most point in the curve uses a 3D codebook whereas other points use a 2D codebook. Utilizing 3D codebooks in ResNet-56 models requires larger storage but appears to perform worse than those compression using 2D codebooks but at higher compression rates. We note that a similar behavior has been observed with PQF [78]. At the same level of compression rates, the performance of ResNet-56 models in the "small blocks regime"(similar with our 2D decomposition scheme) is better than it in "large blocks regime" (similar with our 3D decomposition scheme).

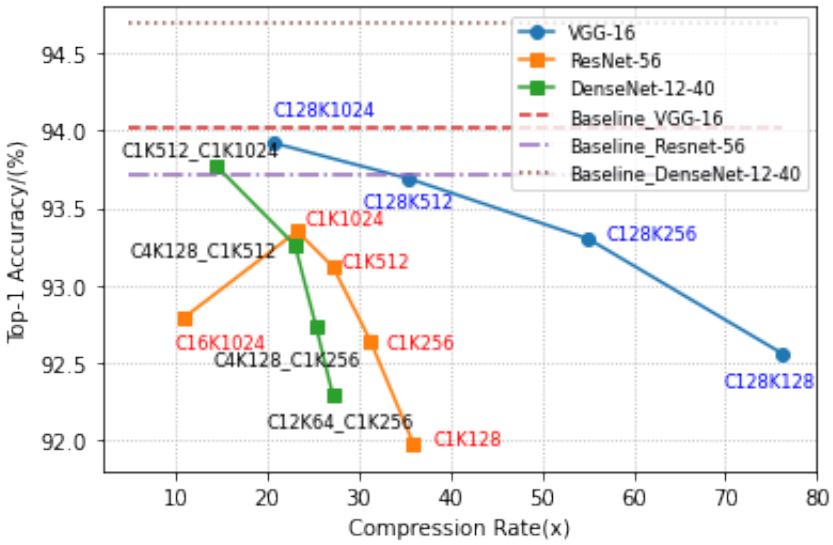


Figure 5.1: The performance of VQLC on CIFAR10 in two decomposition schemes.

5.4.3 Comparison with other works

5.4.3.1 Baselines

We now compare VQLC with various start-of-the-art network quantization techniques, including Clustering Convolutional Kernels (CCK) [102], Learning Filter Basis (LFB) [71], Bit Goes Down (BGD) [103], Any-Precision DNN (APD) [122] and Permute, Quantize and Fine-tune (PQF) [78].

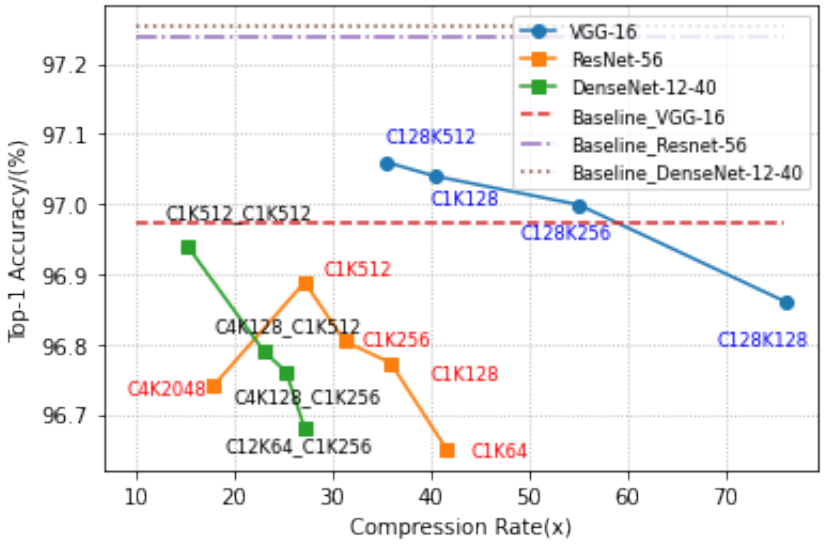


Figure 5.2: The performance of VQLC on SVHN in two decomposition schemes.

Noting that the highest compression rate for scalar quantization (SQ) is only 32× (1-bit model), thus we only include one SQ scheme in comparison, APD [122]. We will focus on vector quantization techniques, including CCK [102], LFB [71], BGD [103] and PQF [78]. CCK [102] extracts normalized 2D codewords based on K-means clustering from a pre-trained model; then a trainable scale for every 2D filter block is introduced and retrained with the network-wise codebook. In LFB [71], each convolutional filter is represented as a linear combination of codewords; both the codebook and the combination coefficients for each filter are learned and stored. On VGG-16, LFB [71] adopts the layer-wise codebooks, whereas in ResNet-56 and DenseNet-12-40, block-wise and network-wise codebooks are selected respectively. Both of PQF [78] and BGD [103] may adopt a 2D or 3D decomposition of filters, where in the latter case, the maximal number c of input channels is only 2. Both methods first exact layer-wise codebooks from a pre-trained model and then retrain the codebooks. In BGD [103], the EM algorithm is used for clustering and minimizing reconstruction errors. In PQF [78], before quantization, the algorithm first searches for permutations of filter parameters under which the network is easier to compress.

5.4.3.2 Assurance of Fair Comparison

All compressed models by different schemes for comparison need to be kept at the same level of compression rates. Since we are interested in high rates of compression, we reduce the codebooks' sizes of competitor schemes from those provided in their original papers. Notably, we keep all linear layers uncompressed, and store the final codebooks using 16-bit floats for all competitors. Moreover, CCK [102] only provided the code for compressing same RF-sized convolutional layers. We modify their code allowing it to compress CNN layers with different RF-sizes for the training of DenseNet-12-40 models. In BGD [103], due its the high sensitivity to the choices of hyperparameters in the EM algorithm, we are unable to find a hyper-parameter setting for which iteration in the EM algorithm will properly terminate. Thus for BGD, we only provide results for ResNet-56 on CIFAR10 and CIFAR100.

Except the codebook size, most settings of hyperparameters for the competitors are taken from their original papers. For each model type on every dataset, we compare the performance of quantized models at more than three different compression rates.

5.4.3.3 Experimental Results

The comparison results for VGG-16, ResNet-56, and DenseNet-12-40 are shown in the Figure 5.3, Figure 5.4 and Figure 5.5 respectively, and is summarized in the Table 5.14.

Since our approach beats LFB [71], BGD [103] and APD [122] on every model and dataset, we only focus on analyzing the comparison results with CCK [102] and PQF [78].

For VGG-16 models, our approach outperforms all competitors on most tasks at high compression rates ($35 \sim 95\times$). Moreover, all models from PQF [78] with 3D codebooks outperform models from CCK [102] with 2D codebooks. 3D decomposition scheme not only makes compression rates higher ($76 \sim 90\times$) but also better preserves the model accuracy. However, the best scheme option in ResNet-56 is opposite.

For ResNet-56 models over all datasets, both of our method and CCK [102] perform better than PQF [78], since we compress all convolutional layers simultaneously (i.e., only one codebook) rather than handling each layer individually, which leads to the higher compression rate.

For DenseNet-12-40 models, we only try a 2D codebook in pointwise convolutions, which may result in the relatively lower compression rate compared to it in other models.

Method	CIFAR10			CIFAR100			SVHN			MNIST		
	V	R	D	V	R	D	V	R	D	V	R	D
CCK [102]	√	×	×	×	−	×	√	√	√	√	−	×
PQF [78]	√	√	×	×	√	−	√	√	√	√	√	×
LFB [71]	√	√	√	√	√	√	√	√	√	√	√	√
APD [122]	√	√	√	√	√	√	√	√	√	√	√	√
BGD [103]		√			√							

Table 5.14: Comparison results on four datasets.

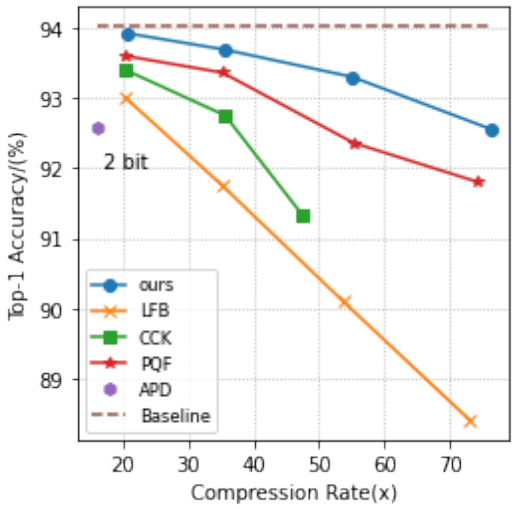
"V", "R" and "D" denote VGG-16, ResNet-56 and DenseNet-12-40 models. "√" means that our models wins at all compression rates. "×" indicates that competitors perform better at some compression rate. "−" means that our models are comparable to competitors.

5.4.4 Additional Experiments: WideVGG-16

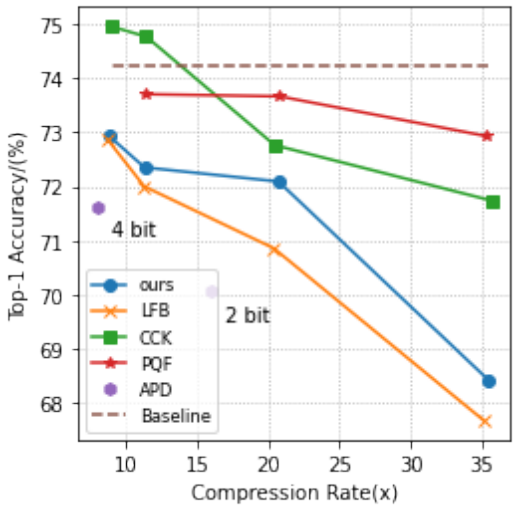
A common wisdom in information theory suggests that when there is more redundancy, there is larger room for compression. This motivates us to investigate how VQLC and its competitors perform on wider networks, which are arguably "more over-parameterized". For this purpose, we perform experiments compressing the WideVGG-16 model.

5.4.4.1 Experiments on CIFAR10

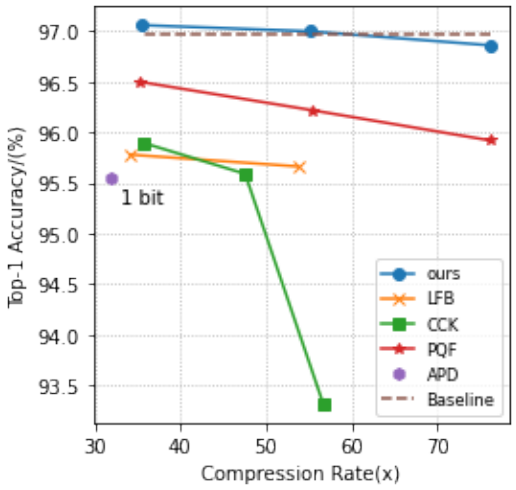
The experimental results of VQLC for WideVGG-16 models on CIFAR10 dataset is listed in Table 5.15. The wider VGG-16 models perform better than the original VGG-16 models with VQLC on the CIFAR10 dataset. WideVGG-16 reaches the same accuracy with the uncompressed baseline at 56.0× compression rate, whereas the best quantized VGG-16 only obtains 0.1% accuracy drop compared with its corresponding baseline at a much lower compression rate 20.72×.



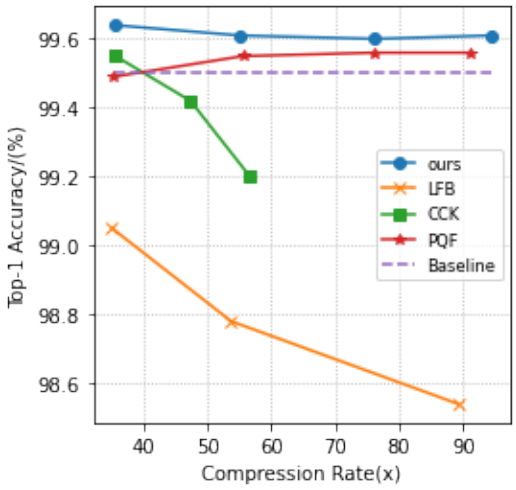
(a) VGG-16 on CIFAR10



(b) VGG-16 on CIFAR100



(c) VGG-16 on SVHN

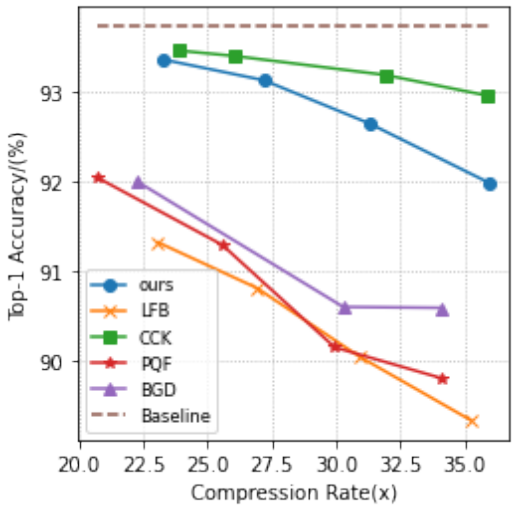


(d) VGG-16 on MNIST

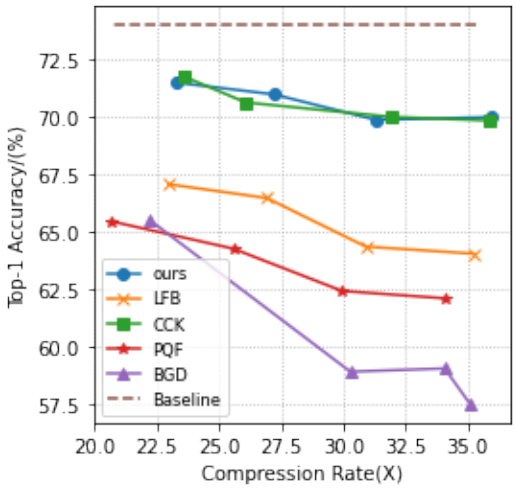
Figure 5.3: Comparison with CCK [102], PQF [78], LFB [71], BGD [103], APD [122] for VGG-16 on different datasets.

Model/Method	Top-1 Accu.(%)	Compression Rate(×)
WideVGG-16/baseline	94.59	—
WideVGG-16-C256K256	94.04	77.89
WideVGG-16-C128K512	94.19	77.08
WideVGG-16-C256K512	**94.59**	56.0
WideVGG-16-C128K1024	94.49	55.52

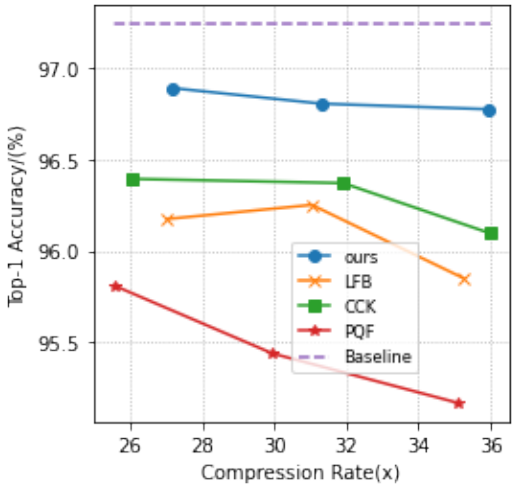
Table 5.15: Results of our method for compressing WideVGG-16 on CIFAR10



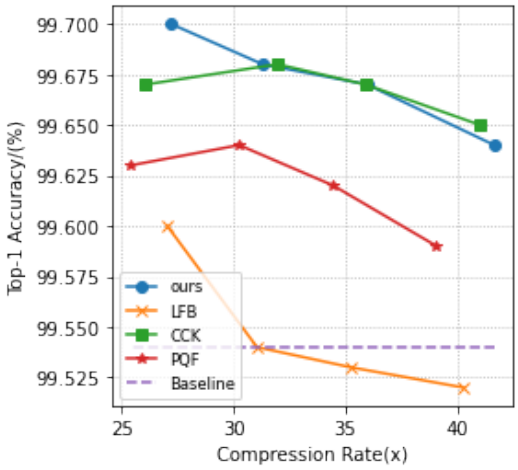
(a) ResNet-56 on CIFAR10



(b) ResNet-56 on CIFAR100



(c) ResNet-56 on SVHN

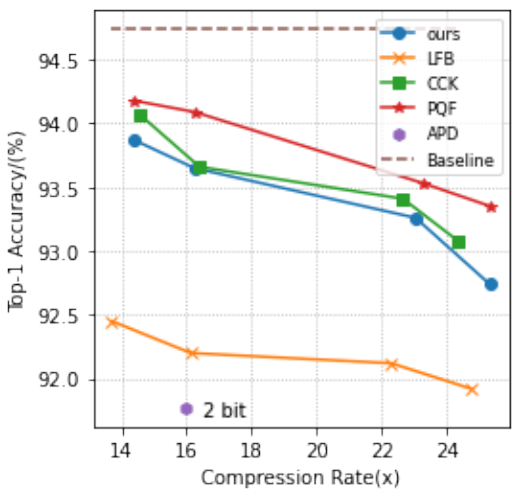


(d) ResNet-56 on MNIST

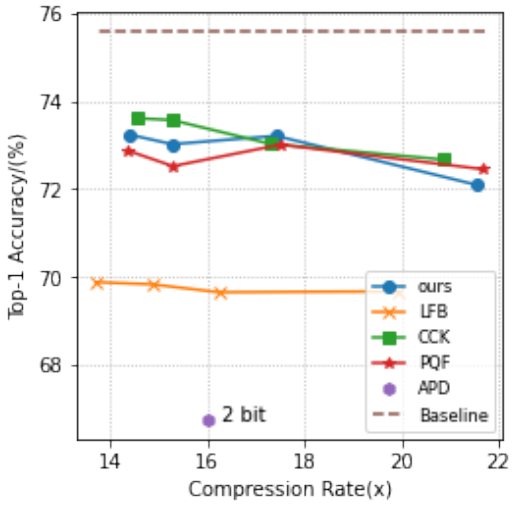
Figure 5.4: Comparison with CCK [102], PQF [78], LFB [71], BGD [103], APD [122] for ResNet-56 on different datasets.

5.4.4.2 Experiments on CIFAR100

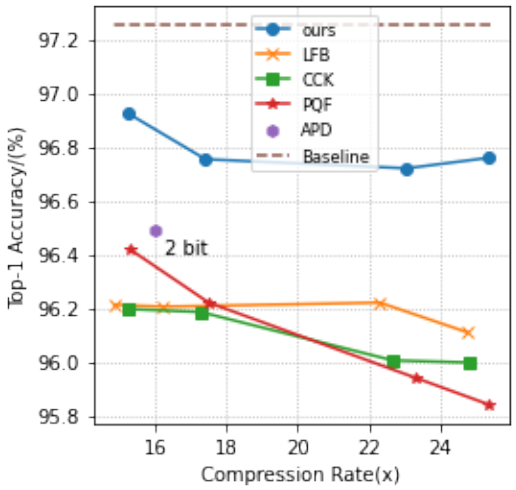
The wider VGG-16 models also perform better than the VGG-16 models on the CIFAR100 dataset. The best compressed WideVGG-16 model obtains only 1.28% accuracy drop at 35.6× compression rate.



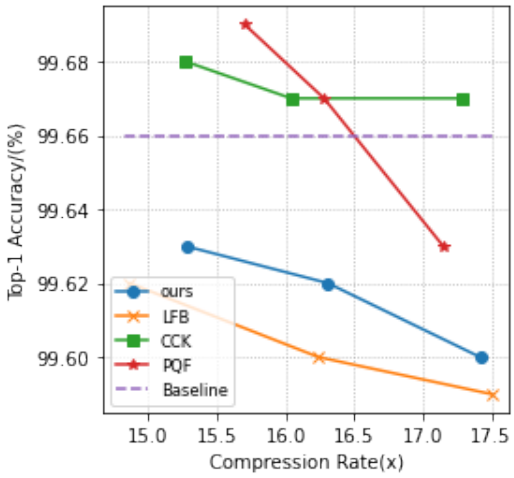
(a) DenseNet-12-40 on CIFAR10



(b) DenseNet-12-40 on CIFAR100



(c) DenseNet-12-40 on SVHN



(d) DenseNet-12-40 on MNIST

Figure 5.5: Comparison with CCK [102], PQF [78], LFB [71], BGD [103], APD [122] for DenseNet-12-40 on different datasets.

5.4.4.3 Experiments on SVHN

The experimental results of VQLC for WideVGG-16 on SVHN are listed in Table 5.17. Our quantized model can reach almost 120× compression rate while preserving the model accuracy.

Model/Method	Top-1 Accu.(%)	Compression Rate(×)
WideVGG-16/baseline	76.06	—
WideVGG-16-C256K256	71.14	77.89
WideVGG-16-C256K512	73.19	56.0
WideVGG-16-C128K1024	73.84	55.52
WideVGG-16-C256K1024	74.44	35.84
WideVGG-16-C128K2048	**74.78**	35.64

Table 5.16: Results of our method for compressing WideVGG-16 on CIFAR100

Model/Method	Top-1 Accu.(%)	Compression Rate(×)
WideVGG-16/baseline	97.16	—
WideVGG-16-C128K64	96.45	117.54
WideVGG-16-C128K128	96.55	109.2
WideVGG-16-C128K256	96.773	95.78
WideVGG-16-C128K512	96.9	77.08
WideVGG-16-C128K1024	97.13	55.52

Table 5.17: Results of our method for compressing WideVGG-16 on SVHN

5.4.4.4 Experiments on MNIST

All compressed WideVGG-16 models outperform the uncompressed baseline, even at 117.54× compression rate!

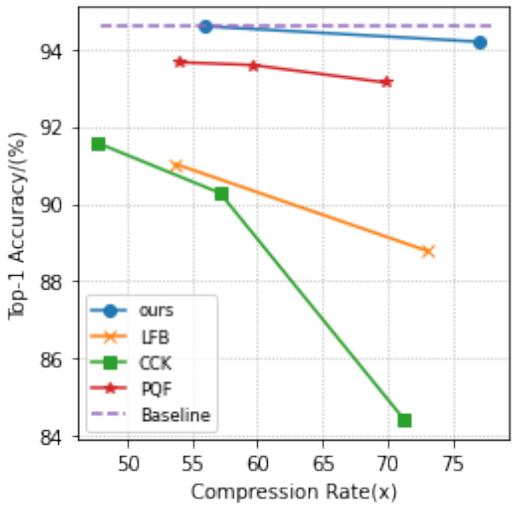
Model/Method	Top-1 Accu.(%)	Compression Rate(×)
WideVGG-16/baseline	99.65	—
WideVGG-16-C128K64	**99.69**	117.54
WideVGG-16-C128K128	**99.7**	109.2
WideVGG-16-C128K256	**99.7**	95.78
WideVGG-16-C128K512	**99.72**	77.08

Table 5.18: Results of our method for compressing WideVGG-16 on MNIST

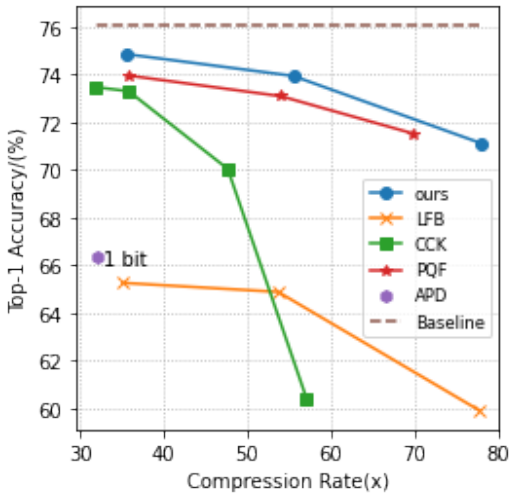
5.4.4.5 Comparison with other works

We compare our compressed WideVGG-16 models with the state-of-the-art. Figure 5.6 shows the comparison results on CIFAR10, CIFAR100, SVHN and MNIST datasets.

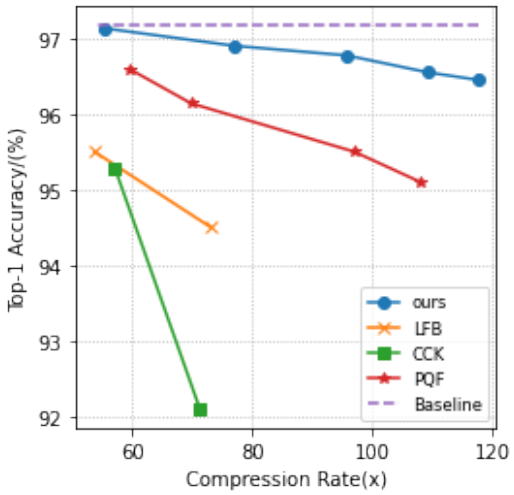
From these figures, it is clear to see that our approach outperforms all competitors on WideVGG-16 models at high compression rates ($55 \sim 120\times$). Compared with the performance of our VGG-16 models on CIFAR100 in Figure 5.3b, the performance of VQLC is much better in WideVGG-16 models, as shown in Figure 5.6b, improving $\sim 1\%$ accuracy over the state of the art. On CIFAR10 dataset, although our VGG-16 models outperform other competitors as shown in Figure 5.3a, VQLC is more effective on wide networks. One of our models obtains the same validation accuracy as the uncompressed model at $56\times$ compression rate, and $\sim 3\%$ accuracy rise compared with the best competitor, as shown in Figure 5.6a. On MNIST dataset, all of our compressed models even outperform the uncompressed model.



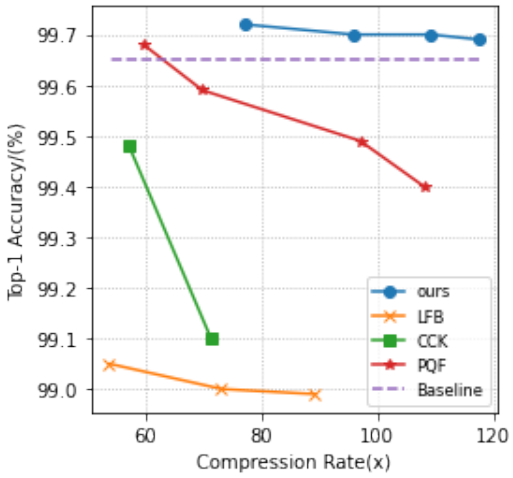
(a) WideVGG-16 on CIFAR10



(b) WideVGG-16 on CIFAR100



(c) WideVGG-16 on SVHN



(d) WideVGG-16 on MNIST

Figure 5.6: Comparison with CCK [102], PQF [78], LFB [71], BGD [103], APD [122] for WideVGG-16 on different datasets.

Chapter 6

Conclusion and Future Work

In this thesis, we propose a novel neural network compression scheme, referred to as Vector Quantization with Learned Codebook or VQLC. Our motivation is the hypothesis that by searching over a larger codebook space than the neighborhood of a pretrained model, one may potentially find better quantization codebook. This leads to the development of VQLC, in which the quantization codebook is randomly initialized and learned together with the training of the model. We demonstrate the state-of-the-art performance of VQLC on several popular image classification benchmarks. In particular, we observe that VQLC appears to be superior to the existing compression schemes on wide networks. This to an extent validates our motivating hypothesis.

Our proposed method can be easily and flexibly generalize to various CNN architectures with the same or multiple RF sizes. Moreover, VQLC has multiple and flexible design choices for compression.

There are situations in which VQLC performs similarly or somewhat worse than an existing scheme. This may be due to the fact that the search of best codebook in VQLC is nonetheless optimal so that there is the possibility that a codebook near a pretrained model may turn out to achieve a better rate-accuracy tradeoff than what VQLC finds. Moreover, it remains unclear to us how the shape of the filter blocks in filter decomposition (i.e., the number c of the input channels) impacts the performance of VQLC (and other compression schemes). More insight into this will help to decide an appropriate filter decomposition scheme without the need of a heuristic search.

In future work, we intend to explore why wider CNNs are more efficient for quantization.

Whether there exists a thinner and smaller subnetwork that can reach the similar test accuracy with the original CNN, as proposed in [27].

In practice, we only explore the performance of models with our proposed quantization approach at some chosen rates of compression for a given dataset. In future work, we intend to study the theoretical analysis on the trade-off between the compression rate and the performance. Whether there exists a specific measurement for the best trade-off between the performance and the rate of compression, for a given data distribution and data sample size.

The final plan for future work centers on the application of our proposed approach, VQLC. Beyond convolutional neural networks, whether our approach can applied to other architectures, such as the Transformer in NLP tasks, and how good the performance of our approach.

References

[1] Weight quantization in boltzmann machines. *Neural Networks*, 4(3):405–409, 1991.

[2] Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *ICLR*, 2018.

[3] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers, 2020.

[4] Jose M. Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. *CoRR*, 2016.

[5] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks, 2015.

[6] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep?, 2014.

[7] S. Banerjee and A. Roy. *Linear Algebra and Matrix Analysis for Statistics*. 2014.

[8] A. Barron, J. Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

[9] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

[10] W. R. Bennett. Spectra of quantized signals. *The Bell System Technical Journal*, 27(3):446–472, 1948.

[11] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *CoRR*, 2020.

[12] Cristian Buciluundefined, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 535–541. Association for Computing Machinery, 2006.

[13] Aydin Buluc and John R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *International Conference on Parallel Processing*, pages 503–510, 2008.

[14] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, volume 37, pages 2285–2294, 2015.

[15] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

[16] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Towards the limit of network quantization, 2017.

[17] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and S. Jagannathan. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pages 1–43, 2020.

[18] Michael Copeland. What's the difference between deep learning training and inference?, 2016. [Online].

[19] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2016.

[20] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory 2nd Edition*. Wiley-Interscience, 2006.

[21] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, Alexander Heinecke, Pradeep Dubey, Jesus Corbal, Nikita Shustrov, Roma Dubtsov, Evarist Fomenko, and Vadim Pirogov. Mixed precision training of convolutional neural networks using integer operations, 2018.

[22] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

[23] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, 2014.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[25] W.H. Equitz. A new vector quantization clustering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(10):1568–1575, 1989.

[26] Emile Fiesler, Amar Choudry, and H. John Caulfield. Weight discretization paradigm for optical neural networks. In *Optical Interconnections and Networks*, volume 1281, pages 164 – 173. International Society for Optics and Photonics, 1990.

[27] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[28] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, 2019.

[29] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[30] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[31] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[32] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010.

[33] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

[34] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *ICCV*, 2019.

[35] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization, 2014.

[36] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *IJCV*, 129(6):1789–1819, Mar 2021.

[37] Alex Graves. Practical variational inference for neural networks. In *NIPS*, volume 24, 2011.

[38] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[39] R.M. Gray and D.L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998.

[40] Yunhui Guo. A survey on methods and theories of quantized neural networks, 2018.

[41] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[42] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, volume 28, 2015.

[43] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, 2015.

[44] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, December 2015.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, June 2016.

[47] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, Oct 2017.

[48] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[49] Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Annual Conference on Computational Learning Theory*, page 5–13, 1993.

[50] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *CoRR*, 2021.

[51] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *CoRR*, 2021.

[52] A. Honkela and H. Valpola. Variational learning and bits-back coding: an information-theoretic view to bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810, 2004.

[53] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[54] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, July 2017.

[55] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, volume 29, 2016.

[56] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[57] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, 2014.

[58] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, 2014.

[59] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[60] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 25, 2012.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[63] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[64] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition, 2015.

[65] Y. LECUN. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

[66] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[68] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *NIPS*, volume 2, 1990.

[69] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding, 2017.

[70] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets, 2017.

[71] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *ICCV*, October 2019.

[72] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. Pruning and quantization for deep neural network acceleration: A survey. *CoRR*, 2021.

[73] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[74] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning, 2017.

[75] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In *International Conference on Field Programmable Logic and Applications*, pages 1–8, 2016.

[76] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. Exploring the granularity of sparsity in convolutional neural networks. In *CVPRW*, pages 1927–1934, 2017.

[77] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast neural networks without multipliers. *IEEE Transactions on Neural Networks*, 4(1):53–62, 1993.

[78] Julieta Martinez, Jashan Shewakramani, Ting Wei Liu, Ioan Andrei Bârsan, Wenyuan Zeng, and Raquel Urtasun. Permute, quantize, and fine-tune: Efficient compression of neural networks, 2021.

[79] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation, 2016.

[80] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, volume 70, pages 2498–2507, 2017.

[81] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks, 2017.

[82] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NIPS*, volume 1, 1989.

[83] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2012.

[84] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop*, 2011.

[85] S. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4:473–493, 1992.

[86] B.M. Oliver, J.R. Pierce, and C.E. Shannon. The philosophy of pcm. *Proceedings of the IRE*, 36(11):1324–1331, 1948.

[87] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *CVPR*, pages 7197–7205, 2017.

[88] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

[89] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NIPS*, volume 32, 2019.

[90] Bo Peng, Wenming Tan, Zheyang Li, Shun Zhang, Di Xie, and Shiliang Pu. Extreme network compression via filter group approximation. In *ECCV*, September 2018.

[91] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542, Cham, 2016.

[92] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV*, pages 525–542, 2016.

[93] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *PAMI*, 39(6):1137–1149, 2017.

[94] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[95] Kenneth Rose, Eitan Gurewitz, and Geoffrey Fox. A deterministic annealing approach to clustering. *Pattern Recognition Letters*, 11(9):589–594, 1990.

[96] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[97] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[98] Claude E Shannon. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat.Conv.Rec*, 4:142–163, 1959.

[99] W. F. Sheppard. On the calculation of the most probable values of frequency-constants, for data arranged according to equidistant division of a scale. *Proceedings of the London Mathematical Society*, s1-29(1):353–380, 1897.

[100] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[101] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D. Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65(2):742–769, 2019.

[102] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. Clustering convolutional kernels to compress deep neural networks. In *ECCV*, September 2018.

[103] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks, 2020.

[104] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of deep neural networks under quantization, 2016.

[105] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, volume 27, 2014.

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

[107] C.Z. Tang and H.K. Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8):2724–2727, 1993.

[108] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression, 2017.

[109] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011.

[110] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[111] C. S. Wallace. Classification by minimum-message-length inference. In *ICCI*, pages 72–81, 1990.

[112] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Emerging paradigms of neural network pruning. *CoRR*, 2021.

[113] Huan Wang, Qiming Zhang, Yuehai Wang, and Haoji Hu. Structured pruning for efficient convnets via incremental regularization, 2018.

[114] Lin Wang and Kuk-Jin Yoon. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *PAMI*, pages 1–1, 2021.

[115] Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *ICCV Workshops*, Oct 2017.

[116] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *CoRR*, 2016.

[117] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks, 2016.

[118] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *ICML*, Jul 2018.

[119] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *CVPR*, 2019.

[120] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *CVPR*, pages 7130–7138, 2017.

[121] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[122] Haichao Yu, Haoxiang Li, Honghui Shi, Thomas S. Huang, and Gang Hua. Any-precision deep neural networks, 2021.

[123] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *PAML*, 38(10), 2016.

[124] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *CVPR*, June 2015.

[125] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights, 2017.

[126] Hao Zhou, Jose M. Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *ECCV*, pages 662–677, 2016.

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