Techniques in enhancing computation and understanding of convolutional neural networks

Abdiyeva, Kamila

2021

Abdiyeva, K. (2021). Techniques in enhancing computation and understanding of convolutional neural networks. Master's thesis, Nanyang Technological University, Singapore. <https://hdl.handle.net/10356/154072>

<https://hdl.handle.net/10356/154072>

<https://doi.org/10.32657/10356/154072>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 19:03:35 SGT

TECHNIQUES IN ENHANCING COMPUTATION AND
UNDERSTANDING OF CONVOLUTIONAL NEURAL
NETWORKS

KAMILA ABDIYEVA

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

2021

Techniques in Enhancing Computation and Understanding of Convolutional Neural Networks

Kamila Abdiyeva

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Master of Engineering

2021

Statement of Originality

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

27 April 2021

.....

Date

.....

Kamila Abdiyeva

Supervisor Declaration Statement

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

27 April 2021

.....

Date

.....

Yap Kim-Hui

Authorship Attribution Statement

This thesis **contains** material from 2 papers published at the conferences proceeding in which I am listed as the first author.

Chapter 3 is published as Kamila Abdiyeva, Kim-Hui Yap, Wang Gang, Narendra Ahuja, and Martin Lukac. "QL-Net: Quantized-by-Lookup CNN." *In 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, November 18-21, 2018*, pp. 413-418. IEEE, 2018.

The contributions of the co-authors are as follows:

- A/Prof Wang provided the initial project direction.
- I prepared the manuscript drafts. A/Prof Yap and Prof Ahuja revised the manuscript.
- I co-designed the study with A/Prof Wang and A/Prof Yap and performed all the coding and experimental work at the School of Electrical and Electronic Engineering, Nanyang Technological University.
- A/Prof Lukac advised on the procedure for dictionary construction.

Chapter 4 is published as Kamila Abdiyeva, Martin Lukac, and Narendra Ahuja. "Remove to Improve?" *In International Conference on Pattern Recognition (ICPR) International Workshops and Challenges, Virtual Event, January 10–15, 2021, Proceedings, Part III*, pp. 146-161. Springer, 2021.

The contributions of the co-authors are as follows:

- A/Prof Lukac and Prof. Ahuja N. suggested the idea and edited the manuscript drafts.
- I prepared the manuscript.
- I implemented the framework for conducting the study.

27 April 2021

.....

Date

.....

Kamila Abdiyeva

Acknowledgements

I want to thank my current supervisor, Associate Professor Yap Kim Hui, co-supervisor Professor Narendra Ahuja, former supervisors Dr. Wang Gang and Associate Professor Martin Lukac for their guidance, recommendations, and constructive discussions. I also wish to express my greatest gratitude to all my colleagues: Yasin, Mauricio, Roman, Lui Jun, Abrar, and Dipu for all the discussions that helped me to find unexpected solutions from various dead-ends encountered while conducting these studies. I want to express appreciation to the Rapid-Rich Object Search (ROSE) Lab staff, who supported my research by providing the computational resources and facilities.

More importantly, I want to thank my friends and family for their support during these challenging years. This thesis would not be possible without them. This thesis also would not be possible without Google, ArXiv, Github, coffee, and chocolate.

It's fine to celebrate success but it is more important to heed the lessons of failure.

—Bill Gates

To my dear family

Contents

Acknowledgements	ix
Abstract	xv
List of Figures	xv
List of Tables	xix
Symbols and Acronyms	xxi
1 Introduction	1
1.1 Motivation and Objective	1
1.2 Major Contributions	2
1.3 Outline of the Thesis	2
2 Background and Literature Review	5
2.1 Artificial Neural Networks	5
2.2 Convolutional Neural Network	7
2.2.1 Convolutional Layer (CONV)	8
2.2.2 Non-Linear Activation Functions	10
2.2.3 Pooling Layer (POOL)	10
2.2.4 Fully Connected (FC) Layer	11
2.3 Literature Review	11
2.3.1 Enhancing Computation	12
2.3.2 Enhancing CNN Understanding	15
3 Enhancing Computation Using Lookup Quantization	17
3.1 Introduction	17
3.2 Background	18
3.2.1 Convolution Operation	18
3.2.2 Lookup Operation	19
3.3 Proposed QL-Net	20
3.3.1 Codebook Construction	21
3.3.1.1 Codewords Generation	21

3.3.1.2	Lookup Response Generation	23
3.3.2	QL-Net Training	24
3.3.3	Inference	26
3.4	Experiments	27
3.4.1	Datasets	27
3.4.2	Ablation Study	29
3.4.2.1	Experimental Settings	29
3.4.2.2	Experimental Results	30
3.4.3	Experiments	32
3.4.3.1	Experimental Settings	32
3.4.3.2	Experimental Results	32
3.5	Complexity Analysis	33
3.6	Conclusion	35
4	Understanding CNN Filters and Interpretation	37
4.1	Introduction	37
4.2	Definitions	39
4.3	Experimental Study and Analysis	41
4.3.1	Class-wise Accuracy Changes	43
4.3.2	Identification of Filters Contribution to Each Class Recognition	44
4.3.3	Class-wise Pruned Filter Similarity and Semantic Similarity	47
4.3.4	Pruned Filter Similarity and Semantic Similarity between k-Closest Neighbors	48
4.3.5	Changes in Pruned Filter Similarity with k-Closest Neighbors with Pruning	50
4.4	Discussion and Conclusions	53
5	Conclusion and Future Work	55
5.1	Conclusion	55
5.2	Future Work	56
	List of Author's Publications	57
	Bibliography	59
	Appendix 1	67

Abstract

Convolutional Neural Networks (CNNs) are effective in solving a large number of complex tasks. The performance of CNNs is currently equaling or even surpassing the human performance level in a wide range of real-world problems. Such high performance is achieved at the cost of high computational and storage requirements. To satisfy these computational requirements, specialized hardware such as Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs) is required. Besides, CNNs are mainly used as a black-box tool, and only several attempts were made for their understanding.

In this thesis, two studies are provided to address the problems of lack of understanding and high computational requirements of CNNs.

The first study, introduced in Chapter 3, investigates and proposes a method for enhancing CNN computation by reducing the number of computational operations performed. We propose a new method for the computation enhancement in CNNs that substitutes Multiply and Accumulate (MAC) operations with a codebook lookup. The proposed method, Quantized-by-Lookup Network (QL-Net), combines several concepts: (i) a codebook construction, (ii) a layer-wise retraining strategy, and (iii) substitution of the MAC operations with the lookup of the convolution responses at inference time. The proposed QL-Net achieves good performance on datasets such as MNIST and CIFAR-10.

The second study provides a better CNN understanding by studying the importance of each learned feature for an individual object class recognition. The experimental work in Chapter 4 extends the current understanding of the CNN filters' roles, their mutual interactions, and their relationship to classification accuracy. Additionally, the study showed that the classification accuracy of some classes from the target objects' set could be improved by removing the sub-set of filters with the least contribution to these classes.

List of Figures

2.1	Example of (a) Perceptron, (b) Multi-Layer Perceptron	6
2.2	Detailed View of a Multi-Layer Perceptron	6
2.3	Convolutional Neural Network	8
2.4	An Example of Learned Convolutional Filter	8
2.5	Conventional CNN Layer	9
2.6	Convolution (MAC) between Input Region $\mathbf{P}$ and Filter $\mathbf{F}_i$	9
2.7	An Example of Applying Maximum Pooling	11
3.1	Conventional CNN Layer	19
3.2	Comparison of the Conventional CNN Operations with Proposed QL-Net	20
3.3	QL-Net Construction Overview.	21
3.4	Codewords Generation Process	22
3.5	Feature Patches $\mathbf{P}$ Spatial Partitioning into Feature Patch Columns $\mathbf{P}^{(j)}$	22
3.6	Lookup Response Computation	24
3.7	QL-Net Inference	26
3.8	QL-Net Lookup Function	27
3.9	MNIST Dataset Sample Images [1]	27
3.10	Fashion-MNIST Dataset Sample Images [2]	28
3.11	GTSRB Sample Images	28
3.12	CIFAR-10 Sample Images	29
3.13	Accuracy vs Total Codebook Size	30
3.14	Acceleration vs the Total Codebook Size	31
4.1	Convolutional Layer and Computation of the Accumulated Responses	40
4.2	ImageNet Sample Images	43
4.3	The Number of Classes for which Classification Accuracy Improved after Pruning	44
4.4	The Number of Classes that Pruned a Particular Filter at a Different θ	45
4.5	Accumulated Response for Each Filter	46
4.6	(A) Pruned Filter Similarity and (B) Semantic Similarity	47

List of Tables

2.1	Activation functions	10
2.2	Methods for Enhancing Computation	13
3.1	CNN Architecture used in the ablation study experiments	29
3.2	Performance of the Proposed QL-Net vs. Baseline Performance.	33
4.1	An Average Number of Semantically Related (SRN) Neighbors.	50
4.2	Average Fraction of Increased Similarity for Semantically Related and Unrelated Neighbors.	52
1	ImageNet Labels for classes ids=[350;450]	67

Symbols and Acronyms

Acronyms

CNN	Convolutional Neural Network
CONV	Convolution
MP	Maximum Pooling
FC	Fully Connected Layer
H-K	Hierarchical K-means
RGB	Red Green Blue
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
CPU	Central Processing Unit
FPGA	Field-Programmable Gate Array
MAC	Multiply and Accumulate Operation
QL-Net	Quantized-by-Lookup Network
RBP	Response Based Pruning

Chapter 1

Introduction

1.1 Motivation and Objective

With the world getting mobile, more and more wearable and embedded devices are continually emerging. On the one hand, mobile devices bring incredible opportunities; on the other, severe constraints. Manufacturers need to constantly compromise on the storage and computational capacity of the devices to provide the required computational time at an affordable price.

Convolutional Neural Networks (CNNs) proved to be effective in solving many quite complicated tasks similar to human-level performance. Such tasks include object recognition, language translation, speech generation, to name some. However, high performance is achieved at the cost of high computation requirements, which requires computationally expensive hardware such as Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs). Therefore, to fully utilize the potential of smart mobile devices and run computationally expensive algorithms such as CNN on smaller embedded and wearable devices, effective optimization methods must be studied and developed. In addition, CNN's are mainly used as a black-box tool, and only several attempts were made for their understanding. These limitations prevent CNN models from being widely integrated into industries that require short response time and transparency of decision-making.

The motivation of this work is to study methods for enhancing CNN computation and understanding to extend the usage of CNN toward embedded and wearable

devices. It will allow accelerating progress in such fields as augmented reality (HoloLens), self-driving, and medical diagnoses.

1.2 Major Contributions

Our main contributions can be stated as follows:

- *Contribution 1*: provided a study to enhance computational operations in CNN by replacing standard MAC operations for filter response computation with Dictionary Lookup of precomputed filter responses.
- *Contribution 2*: provided a study to enhance CNN understanding by:
 - Identification of filters closely associated with a specific class (whose pruning significantly decreases the classification accuracy of the class).
 - Experimentally determining the grouping of classes based on class-wise filter usage overlap and semantic relationship between classes.

1.3 Outline of the Thesis

Chapter 1 introduces the main motivation for studying methods for enhancing computation and understanding of Convolutional Neural Networks (CNNs).

Chapter 2 provides the basics of Convolutional Neural Networks (CNNs) and an overview of currently available approaches.

Chapter 3 introduces the first study for enhancing CNN computation by reducing the number of computational operations performed. In particular, the proposed approach Quantized-by-Lookup Network (QL-Net) investigates the accuracy-performance tread-offs of substituting Multiply and Accumulate (MAC) operations with a dictionary lookup.

Chapter 4 extends the current understanding of the CNN filters' roles, their mutual interactions, and their relationship to classification accuracy. This is motivated by observations that the classification accuracy for some classes increases instead

of decreasing when some filters are pruned from a CNN. We are interested in experimentally addressing the following question: Under what conditions do filter pruning increase classification accuracy? And, can we select a subset of CNN filters to classify the subset of learned classes more accurately?

Chapter 5 summarizes the main finding of the thesis and provides future research directions.

Chapter 2

Background and Literature Review

This chapter provides the background on Artificial Neural Network (ANN), their main variants, and components. After providing a general introduction to ANN, this chapter focuses more on a particular ANN variation, named Convolutional Neural Network (CNN). The chapter is concluded with an overview of the current methods for enhancing CNN computation and understanding.

2.1 Artificial Neural Networks

Definition 2.1 (Artificial Neural Network (ANN)). An Artificial Neural Network (ANN) is a set of interconnected input-output computational units. It is specified as a graph $G(V, E)$ with each vertex V is specified by $f : I \rightarrow O$ and each E is a connection between two vertices.

In this work, we only consider ANNs that are represented as Direct Acyclic Graphs (DAGs). This class of ANN is referred to as Feed-Forward ANN. The most popular representatives of Feed-Forward ANN are Perceptron, Multi-Layer Perceptron, and Convolutional Neural Networks (CNNs). These networks are shown in Figure 2.1 and Figure 2.3. The vertices of the ANN DAG are the computational units, also called **neurons**. Each neuron takes the sum of the weighted inputs and applies to it an activation function. The weighted sum of inputs is obtained by eq. 2.1.

$$r_j = \sum_i w_{ij}x_i \tag{2.1}$$

FIGURE 2.1: Example of (a) Perceptron, (b) Multi-Layer Perceptron

with x_i being inputs to the neuron j and w_{ij} is a connection from x_i to the neuron j .

A more detailed example of ANN is shown in Figure 2.2. The network (multi-layer perceptron) has two inputs labeled x_1 and x_2 and one output labeled y . Each connection between two nodes (neurons) in the ANN is represented as a weight value w_{ij} , with j representing the index of the target neuron and i being the index of the input neuron.

FIGURE 2.2: A detailed view of Multi-Layer Perceptron.

The network in Figure 2.2 takes the input variables at nodes x_1 and x_2 , multiplies them by the respective weights w_{11} , w_{21} , w_{12} and w_{22} to obtain $w_{11} * x_1$, $w_{12} * x_1$, $w_{21} * x_2$ and $w_{22} * x_2$. For each of the n_1 and n_2 nodes weighted sums are computed by $r_j = \sum_{i=1}^2 w_{ij} * x_i$. Each of the n_1 and n_2 nodes generates the output by applying

the activation function shown in eq. 2.2.

$$o_j = f(r_j) \quad (2.2)$$

Outputs of the nodes n_1 and n_2 are used as inputs to the node n_3 . The process of multiplying the input values by their weights, summing for each input neuron, and generating the output by applying the activation function is repeated for the node n_3 .

The learning in a feed-forward ANN is done in general by back-propagation [3]. The error E obtained at the output layer is back-propagated through each layer, and the weights are individually adjusted. Then network weights are updated by the equation 2.3 (For simplicity of visualization w_{ij} is denoted as w).

$$w^t = w^{t-1} - \lambda * \frac{\partial E}{\partial w} \quad (2.3)$$

where w^t is updated weights w , w^{t-1} represents old weights, $\frac{\partial E}{\partial w}$ is a partial derivative of the error with respect to weights w , λ is a scaling parameter often referred to as a *learning rate*.

2.2 Convolutional Neural Network

This thesis focuses on the most popular variation of feed-forward ANN called Convolutional Neural Network (CNN). CNN provides solutions to a range of problems, from computer vision [4–11], raw audio generation [12], to general data analytics [13], to name a few. Each network consists of a repeating set of layers (Convolution, Non-linear Activation, Pooling, Fully Connected) interconnected sequentially. Figure 2.3 provides a sample architecture for the case of image classification, which is of interest in this thesis. The first layer receives a multi-channel image as an input. Each subsequent layer then receives as an input a multi-channel output of the previous layer. Each layer transforms its input using a specific set of layer-specific functions (convolution, pooling, non-linear transformation) and produces a multi-channel output. The outputs at the last (output) layer provide the likelihood probability of an input image belonging to each of the given set of classes. The network's core is a set of filters located in different layers and tuned to different

FIGURE 2.3: Convolutional Neural Network

features (an example is provided in Figure 2.4). Each filter offers a different degree and type of evidence for each class indicated by their associated weights. The weights are obtained by the network training process in order to balance the needs of different classes and provide the best average results over the entire training data, e.g., the highest average accuracy of classification of given samples from all classes.

FIGURE 2.4: An Example of Learned Convolutional Filter and the Output of Applying this Filter to the Image.

Subsequent sections will look in more detail at each CNN layer type: Convolutional Layer, Non-linear Activation, Pooling, and Fully Connected Layer.

2.2.1 Convolutional Layer (CONV)

Definition 2.2 (Tensor Object). A $D_1 \times D_2 \times D_3$ dimensional object is called a tensor object, where D_1 and D_2 are spatial dimentions and D_3 is the depth of the tensor.

Figure 2.5 illustrates the main components of a convolutional layer (CONV) in a conventional CNN model. The main objective of the convolutional layer (CONV) is to transform an input tensor object to another tensor output object of neuron activations. Let us look at the layer l in CNN with the total number of layers L . The input to the layer l is a tensor object $\mathbf{B}$, where $\mathbf{B} \in \mathbb{R}^{X \times Y \times d}$. Each $X \times Y$

FIGURE 2.5: A conventional CNN layer, where all regions $\mathbf{P}$ from the input $\mathbf{B}$ are convolved by the set of filters $\mathbb{F}$ resulting in the output tensor $\mathbf{B}'$.

plane in $\mathbf{B}$ at a depth j is called a **feature map** and denoted as $\mathbf{B}_{(:, :, j)}$ or B_j for simplicity. $\mathbf{B}$ can be either an image (for $l = 1$) or a tensor object obtained from the previous layer $l - 1$. The layer l transfers $\mathbf{B}$ into a new tensor object $\mathbf{B}'$ (Figure 2.5), where $\mathbf{B}' \in \mathbb{R}^{X' \times Y' \times d'}$. In order to perform these transformations a convolutional layer performs MAC operations (multiplication and summation) between input tensor regions, denoted as $\mathbf{P} \in \mathbb{R}^{M \times M \times d}$ and a set of filters $\mathbb{F} = \{\mathbf{F}_1, \dots, \mathbf{F}_{d'}\}$, where $\mathbf{F}_i \in \mathbb{R}^{M \times M \times d}$. An example of convolving an input object region $\mathbf{P}$ with a filter $\mathbf{F}_i$ is provided in Figure 2.6 (for simplicity of visualization in Figure 2.6 the depth d of both $\mathbf{P}$ and $\mathbf{F}_i$ is set to 1). Each convolution involves one $\mathbf{P}$, extracted

$$MAC(F_i, P) \rightarrow 10 * 1.5 - 15 * 0.1 + 77 * 0 - 0 * 0.2 + 10 * 1.5 - 11 * 0.1 + 0 * 0 - 0 * 0.2 + 10 * 1.5 = 42.4$$

FIGURE 2.6: Convolution (MAC) between an input region $\mathbf{P}$ and a filter $\mathbf{F}_i$. For simplicity of visualization, depth d is 1 for both the input and the filter.

iteratively from $\mathbf{B}$ along the spatial dimensions X and Y , and all filters from $\mathbb{F}$. The convolutional output between $\mathbf{P}$ and $\mathbf{F}_i$ is a single value r_i (Figure 2.6) and the

convolutional output between $\mathbf{P}$ and $\mathbb{F}$ is a tensor object $\mathbf{r} \in \mathbb{R}^{1 \times 1 \times d'}$. Combining all $\mathbf{r}$ s for each $\mathbf{P}$ along the spatial coordinates results in the output tensor object $\mathbf{B}'$.

2.2.2 Non-Linear Activation Functions

To ensure a non-linear transformation of the output $\mathbf{B}'$ of the convolutional layer an activation function is applied. The activation function is applied in an element-wise manner (to every $r_i \in \mathbf{r}$). The most popular activation functions are Step Function, Sigmoid, Hyperbolic Tangent ($\tanh$), and Rectified Linear Activation Function (ReLU). Mathematical notations for each of the specified function are provided in Table 2.1, where $\exp$ is a natural exponential function, and $\max$ is a function returning a maximum of two given values.

Name	Mathematical Notation
Step function	$o_i = \begin{cases} 0 & \text{for } r_i \leq 0 \\ 1 & \text{for } r_i > 0 \end{cases}$
Sigmoid	$o_i = \frac{1}{1 + \exp(-r_i)}$
Tanh	$o_i = \frac{\exp(2r_i) - 1}{\exp(2r_i) + 1}$
ReLU	$o_i = \max(0, r_i)$

TABLE 2.1: Activation functions

2.2.3 Pooling Layer (POOL)

A pooling layer (POOL) performs downsampling along only the spatial dimensions of the tensor; hence it is depth independent. It takes an input tensor object of dimension $X \times Y \times d$ and returns an object of dimension $X' \times Y' \times d$, $X' = \frac{X-N}{s} + 1$ and $Y' = \frac{Y-N}{s} + 1$, where N is a pooling filter spatial dimension and s is stride.

Two functions are mainly used as a pooling operator:

Average Pooling (AP) function computes the average value over all values inside an input patch $P \in \mathbb{R}^{N \times N}$ of a feature map $B_j \in \mathbb{R}^{X \times Y}$, where $N < X$ and $N < Y$.

Maximum Pooling (MP) function returns the maximum value inside an input patch $P \in \mathbb{R}^{N \times N}$ of a feature map $B_j \in \mathbb{R}^{X \times Y}$, where $N < X$ and $N < Y$.

An example of applying a maximum pooling (MP) to a feature map B_j at depth j is provided in Figure 2.7.

FIGURE 2.7: An example of applying a maximum pooling with pooling filter spatial dimensions 2×2 and stride $s = 2$ to a single feature map B_j

2.2.4 Fully Connected (FC) Layer

Fully Connected (FC) layers are Neural Network layers in which all the inputs from one layer are connected to every computational unit (neuron) of the next layer. A multi-layer perception is one of the examples of ANN consisting mainly of Fully Connected layers. One can observe that the main difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region (input region $\mathbf{P}$) in the input object $\mathbf{B}$, while neurons in the FC layer are connected to all components of the input object $\mathbf{B}$.

2.3 Literature Review

In this chapter, we will briefly review the existing related methods on enhancing CNN computation and understanding.

While Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance in different computer vision tasks, such as classification [14–16], detection [17, 18] and semantic segmentation [19–21], they suffer from a high computational

cost and power consumption, making them challenging to use without a powerful Graphics Processing Unit (GPU). Thus, to allow small and embedded devices to use CNN technology, their power consumption, and computational complexity should be reduced.

2.3.1 Enhancing Computation

Several methods for CNN computation enhancement for acceleration purposes have been developed [22–26]. The early approaches involved restricting weights to be a number of a form 2^n , where n is an integer. This approximation allows substitution of all the multiplications by binary shifts [27]. In [28, 29] the authors proposed the CNN computation reduction by reducing the precision of the parameters. [22] proposed a stochastic binarization of the network weights and replacing the multiply and accumulate (MAC) operation used in convolution by the sign changes. In [22] weights are binarized only during the forward propagation, while feature maps remain unchanged. Further acceleration was achieved by binarizing both the network’s parameters and the feature maps [24]. The BNN [24, 30] uses bitwise operations, which allows substituting expensive multiplications. On the one hand, this approach reduces 32 times the memory usage; on the other, it reaches a lower accuracy than a model with full-precision weights. To address a drastic drop in accuracy, XNOR-Net [23] proposed to scale the binary parameters with a full-precision scaling factor to increase the network capacity while still exploiting faster bitwise operations (XOR or AND) instead of the conventional MAC. Finally, in ABC-Net [31], instead of using a single set of binarized weights, the approach approximates the full-precision weights with the linear combination of multiple binary weight sets, achieved by thresholding the original weights at different ranges. The feature maps were approximated in a similar manner - by multiple-range thresholding. ABC-Net contains full precision scaling factors for each set of binary weights, preventing it from achieving the maximum acceleration. DoReFa-Net [32] proposed to extend the precision reduction to gradients, therefore, allowing low-precision and higher speed training. The performance of some of the methods is provided in Table 2.2.

¹on hardware that support 64bit operations

Dataset	Model	Computational Gain	Accuracy
MNIST [1]	BNN [24]	32	98.6
ImageNet [33]	Baseline (AlexNet) [14]	1.0	56.6
	XNOR [23]	58 ¹	44.2
	Han [34]	3.0	57.23
ImageNet [33]	Baseline (ResNet-18) [35]	1.0	69.3
	ABC-Net [31]	32	42.7
	ABC-Net [31]	6.4	65.0
	DoReFa-Net [32]	8.0	59.2
	XNOR [23]	58	51.2

TABLE 2.2: The performance of methods for Enhancing Computation on MNIST/ImageNet dataset and AlexNet/ResNet-18 models

While reduced-precision networks minimize the number of computations and increase the inference speed, they require dedicated hardware. The current Central Processing Units (CPUs) and Graphics Processing Units (GPUs) are not well suited for the low-precision operations, and resources are wasted because the full power of these computational devices is not exploited. For optimal use, the binarized CNNs require low power computation hardware with limited reconfigurability. However, the design cost of such specific hardware is much higher than that of a standard high-speed FPGA or CPUs. Hence, it is challenging to minimize the CNN computation requirements without introducing hardware restrictions. Additionally, all of the available binarized networks preserve the first and the last layer in full-precision. Therefore, the network is not homogeneous and, when ported to a specialized or embedded device, requires specific hardware for the binarized and non-binarized network components, making such hardware less efficient and more expensive.

Another approach to the reduction of computation in CNNs is pruning. Currently, available pruning methods [36–41] can be split mainly into three groups, based on weights, feature maps’ values and accuracy/loss-based optimizations. The simplest line of work is based on removing filters with the smallest values [42]. Pruning based on feature maps’ values focuses on removing the least active filters. One of such works is [43]. This method removes those filters that do not change the

feature maps significantly. Accuracy/loss-based optimization [44, 45] methods try to estimate the contribution of a filter to the overall loss function or the model accuracy and usually involve a training component. For instance, Taylor pruning of [44] is optimized towards the final loss function with additional retraining. [46] proposed “lottery ticket hypothesis”. In particular, it states that one can identify sub-networks ("winning tickets") by pruning a trained model, and when such sub-networks are trained from scratch with the original parameter initialization, they achieve similar performance to the original network.

2.3.2 Enhancing CNN Understanding

A number of attempts have been made to interpret the operation of CNNs [40, 47–53]. Many of these [50, 52–54] focused on explaining the functions of individual neurons. They search for the inputs/objects that maximize the response of a specific neuron, which is then seen as a “single-cell object identifier”. Since neurons work together as a network, a single-neuron feature visualization provides limited insight into understanding neural network decision-making. However, it is also challenging to analyze decisions made by groups of neurons due to the combinatorial complexity created by a large number of network parameters. Visualization of groups of neurons was studied through visualization of filter responses in 2D space using PCA [55] or t-SNE [56] compression [57].

Lastly, in [58], in addition to a single-neuron visualization, authors approximate how a variation in neuron value changes the probabilities of each class.

Chapter 3

Enhancing Computation Using Lookup Quantization

3.1 Introduction

While Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance in different computer vision tasks, such as classification[14–16], detection [17, 18], and semantic segmentation [19–21], they suffer from the high computational cost and the power consumption, making them challenging to use without a powerful Graphics Processing Unit (GPU). Additionally, with new intelligent devices appearing every day, systems without powerful GPUs are becoming very common. Thus, to allow small and embedded devices to use CNN technology, their power consumption, and computational complexity should be reduced.

This chapter presents a study to reduce the number of operations performed by Convolutional Neural Network (CNN) using an alternative to Multiply and Accumulate (MAC) operations. We propose a novel framework, called QL-Net, which stands for Quantized-by-Lookup CNN. QL-Net integrates three main concepts: (i) a dictionary construction to approximate patches of CNN feature maps, (ii) a layer-wise retraining strategy to minimize the approximation error, and (iii) performing dictionary lookup instead of convolutional MAC operations to reduce the number of operations during the inference.

QL-Net takes an image as an input and extracts the feature maps, similar to the conventional CNN. Unlike the conventional CNN, QL-Net extracts feature maps by lookup operations rather than by convolution. In addition, QL-Net can be used on a general-purpose CPU because all the operations are performed using the same precision as in normal CNN. The QL-Net was evaluated on the MNIST, Fashion MNIST, German Traffic Sign Recognition Benchmark (GTSRB) and CIFAR-10 datasets. Compared to the current MAC-based CNN performance, a proposed implementation resulted in a 39x, 39x, 33.16x and 27.9x reduction in computational time complexity while the accuracy drop was only 0.13%, 2.63%, 1.21% and 4.5% for MNIST, Fashion MNIST, GTSRB and CIFAR-10, respectively.

This chapter is published as Kamila Abdiyeva, Kim-Hui Yap, Gang Wang, Narendra Ahuja, and Martin Lukac. "QL-Net: Quantized-by-Lookup CNN." *In 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, November 18-21, 2018*, pp. 413-418, IEEE, 2018.

The rest of the chapter is organized as follows. Section 3.2 provides background for the method. Section 3.3 describes the key components and the main steps of the proposed model. Section 3.4 describes and discusses the experimentation and results. Finally, the chapter is concluded by Section 3.6.

3.2 Background

3.2.1 Convolution Operation

Definition 3.1 (Tensor Object). A $D_1 \times D_2 \times D_3$ dimensional object is called a tensor object, where D_1 and D_2 are spatial dimensions and D_3 is the depth of the tensor.

Figure 3.1 illustrates the main components of a convolutional layer in a conventional CNN model. The main objective of the layer is to transform tensor objects (See Definition 3.1). Let's look at the layer l in CNN with the total number of layers L . The input to the layer l is a tensor object $\mathbf{B}$, where $\mathbf{B} \in \mathbb{R}^{X \times Y \times d}$. Each $X \times Y$ plane in $\mathbf{B}$ along the depth dimension d is called a feature map. $\mathbf{B}$ can be either an image (for $l = 1$) or a tensor object obtained from the previous layer

FIGURE 3.1: Conventional CNN layer. (a) Input feature maps. (b) Feature Patch. (c) Set of Filters. (d) Convolution output. (e) Convolution response tensor. (f) Output feature maps.

$l - 1$. The layer l transform $\mathbf{B}$ (Figure 3.1(a)) into a new tensor object $\mathbf{B}'$ (Figure 3.1(f)), where $\mathbf{B}' \in \mathbb{R}^{X' \times Y' \times d'}$. Let's denote a 3D region in $\mathbf{B}$ as a Feature Patch $\mathbf{P}$ (Figure 3.1 (b)) with $\mathbf{P} \in \mathbb{R}^{M \times M \times d}$, where $M < X$ and $M < Y$. Additionally, let's refer to a set of tensor objects $\mathbb{F} = \{\mathbf{F}_1, \dots, \mathbf{F}_{d'}\}$ as a set of filters (Figure 3.1(c)), with $\mathbf{F}_i \in \mathbb{R}^{M \times M \times d}$. The mapping of $\mathbf{B}$ to $\mathbf{B}'$ is done as follows. At layer l , $\mathbf{B}$ is convolved with $\mathbb{F}$. Each convolution involves one $\mathbf{P}$, extracted iteratively from $\mathbf{B}$ along the spatial dimensions X and Y , and all filters from $\mathbb{F}$. The output of convolution between one $\mathbf{P}$ and a filter $\mathbf{F}_i$ is a single output value r_i (Figure 3.1 (d)). The convolutional output between $\mathbf{P}$ and $\mathbb{F}$ is a tensor object $\mathbf{r} \in \mathbb{R}^{1 \times 1 \times d'}$ (Figure 3.1 (e)). Combining all $\mathbf{r}$ s for each $\mathbf{P}$ along the spatial coordinates results in the output tensor object $\mathbf{B}'$.

3.2.2 Lookup Operation

To study alternative approaches to enhance computation performed in CNN, we propose using a Lookup function.

Definition 3.2 (Lookup Table (C)). A lookup table $\mathbb{C}$ is an array of data that maps input values (keys K) to output values (values V), thus providing an approximation for any given mathematical function.

Definition 3.3 (Lookup Operation($\mathcal{L}$)). Given a Lookup table $\mathbb{C}$ and a tensor object $\mathbf{P}$ lookup operation returns the value V associated with the closest key K from $\mathbb{C}$ to $\mathbf{P}$.

Definition 3.4 (Quantization). A process of substituting a feature patch $\mathbf{P}$ by the closest key K from the Lookup Table $\mathbb{C}$.

3.3 Proposed QL-Net

QL-Net takes an image $\mathbf{I}$ as an input and extracts the feature maps $\mathbf{B}_l$ at each layer l , similar to the conventional CNN. Unlike the conventional CNN, QL-Net extracts feature maps by lookup operations rather than by convolution. In conventional CNN, when a convolutional filter is applied to the input patch, MAC operations between the filter $\mathbf{F}_i$ and the input patch $\mathbf{P}$ are performed to calculate the filter response r_i . When multiple filters are applied, responses of all filters are concatenated together ($\mathbf{r}$). In contrast, QL-Net approximates the convolution between an input patch and a set of filters by a dictionary lookup using the patch as the key in the Lookup Table, later referred to as Codebook $\mathbb{C}$. By implementing a hierarchical tree as a Codebook, we are able to find the nearest neighbor in a smaller amount of computations than it would take to perform the convolutional MAC operations on a set of filters $\mathbb{F}$.

FIGURE 3.2: Comparison of the Conventional CNN operations with Proposed QL-Net. We target to reduce the cost of MAC [most expensive operations in the CNN] by replacing MAC with a faster and cheaper LookUp approximation

The proposed framework can be summarized as follows (Figure 3.3 illustrates steps 1-5).

1. Train a conventional CNN model on a selected dataset.
2. Save the input tensor $\mathbf{B}_l$ of the target layer l ($l = \{2, \dots, L-1\}$, where L is a total number of

layers in CNN). 3. Construct a Lookup table (Codebook) $\mathbb{C}_l$ for layer l . In this work hierarchical k-means is used for the Codebook construction. 4. Approximate feature patches in $\mathbf{B}_l$ with the nearest key from $\mathbb{C}_l$. 5. Fine-tune network to minimize error caused by $\mathbf{B}_l$ approximation. 6. Convolve keys from $\mathbb{C}_l$ with the fine-tuned weights and store them as corresponding to these keys values. 7. Repeat steps 2-6 for all layers in the trained CNN. 8. At inference time, use lookup values instead of convolution (MAC).

FIGURE 3.3: QL-Net Construction Overview.

3.3.1 Codebook Construction

A Codebook($\mathbb{C}$) is a lookup table to approximate the input feature maps in the convolutional neural networks and their corresponding responses. Given an input patch $\mathbf{P}$, we can represent it by its closest neighbor in the Codebook, later referred to as Codeword $\mathbf{K}$.

The codebook generation process consists of the following steps:

- generating codewords $\mathbf{K}$ by clustering,
- calculating lookup responses by convolving codewords.

3.3.1.1 Codewords Generation

Codeword($\mathbf{K}$) is an entry to the Codebook $\mathbb{C}$ to approximate the feature patch $\mathbf{P}$.

We start the codewords generation process (Figure 3.4) by sampling a subset of images from the training set and applying a pre-trained CNN model on it to extract resultant feature maps for layer l . Afterward, several Feature Patches $\mathbf{P}$ are selected at random to obtain the data samples (data points) for the codebook construction.

FIGURE 3.4: Codewords Generation Process

With each successive layer, the number of filters in the layer increases, significantly increasing the depth (d) of the input to the next layer. As a result, performing approximation directly on the $M \times M \times d$ dimensional feature patch results in a too coarse approximation and a high quantization error. To improve the coarse approximation, we propose to partition the $M \times M \times d$ vector into smaller parts of size $1 \times 1 \times d$

1.1) Data Samples Partitioning: To improve the representational power of the constructed codebook we propose to use small dimensional codewords. Hence, sampled $\mathbf{P}$ are partitioned along the spatial dimensions into $N = M^2$ lower dimensional tensors (Figure 3.5) referred to as the Feature Patch Column $\mathbf{P}^{(j)} \in \mathbb{R}^{1 \times 1 \times d}$ for $j = 1 \dots N$. This type of partitioning will be referred to as a **spatial partitioning**.

FIGURE 3.5: Feature Patches $\mathbf{P}$ Spatial Partitioning into Feature Patch Columns $\mathbf{P}^{(j)}$

1.2) Codewords Generation by Clustering: The codewords are generated by clustering the collected feature patch columns. In particular, the hierarchical k-means clustering (H-K-means) was used since it provides an efficient codewords' search. H-K-means algorithm subdivides the sampled feature patch columns into k subsets using the conventional k-means algorithm. Subsequently, each subset is divided into other k subsets. The process is repeated recursively until the termination criteria are attained. In our implementation, the first termination criterion is the depth of the tree H . The second termination condition is the density threshold ρ . ρ was introduced to avoid the creation of clusters containing only a single data sample.

The last criterion will guarantee that the clustering is applied to the subset if its density exceeds ρ . The clustering will result in a codebook $\mathbb{C}$ with Q number of codewords $\mathbf{K}_v \in \mathbb{R}^{1 \times 1 \times d}$ (for $v = 1, \dots, Q$).

3.3.1.2 Lookup Response Generation

Once all layers in CNN were approximated with codewords and their corresponding filters were fine-tuned, then we modify our dictionary (Codebook $\mathbb{C}$) by convolving each key (codeword) $\mathbf{K}$ entry in $\mathbb{C}$ by the set of filters ($\mathbb{F}$) of the layer l (Figure 3.6), so we can use resultant $1 \times 1 \times d'$ vector to approximate convolution operations (d' is the number of convolutional filters $\mathbf{F}_i$ we applied to the input tensor $\mathbf{B}_l$ of the layer l). During the inference, we approximate each patch $\mathbf{P}$ from the input with the closest key $\mathbf{K}$ from the Codebook and use calculated convolution output for the $\mathbf{K}$ instead of performing convolution on the input patch.

Lookup Response Vector($\mathbf{r}$) is an output of convolution between a Codeword $\mathbf{K}$ and set of filters $\mathbb{F}$ of the layer l .

On account of introducing the spatial partitioning, the filters $\mathbf{F}_i$ should be subdivided into N sub-filters $\mathbf{F}_i^{(j)}$ (for $i = 1 \dots d'$ and $j = 1 \dots N$) in a similar to the feature patch partitioning manner (Figure 3.5). Afterwards, the lookup responses are calculated for each sub-filter (sub-filter at position j in $\mathbf{F}_i$) separately (Figure 3.6).

Each sub-filter is applied to each codeword ($\mathbf{K}_v$ for $v = 1 \dots Q$) in the codebook as can be seen in Figure 3.6. The respective convolution output is obtained by eq. 3.1.

$$r_{i,v}^{(j)} = \mathbf{K}_v * \mathbf{F}_i^{(j)} \quad (3.1)$$

where $r_{i,v}^{(j)}$ is the scalar convolution output between the v -th codeword in the codebook and the j -th sub-filter of the $\mathbf{F}_i$ filter (a sub-filter at the spatial position j in $\mathbf{F}_i$). The convolution outputs between a codeword and the sub-filters at the position j for all d' filters in the set $\mathbb{F}$ are concatenated together into one lookup response $\mathbf{r}_v^{(j)} \in \mathbb{R}^{1 \times 1 \times d'}$ (Eq. 3.2).

$$\mathbf{r}_v^{(j)} = r_{1,v}^{(j)} \parallel \dots \parallel r_{d',v}^{(j)}, \quad (3.2)$$

FIGURE 3.6: Lookup Response computation for a Codeword $\mathbf{K}_v$ and a layer l . (a) Select a Codeword $\mathbf{K}_v$ from the Codebook. (b) Perform Spatial Partitioning of all Filters of the layer l into sub-filters $\mathbf{F}_i^{(j)}$ (c) Convolve the Codeword $\mathbf{K}_v$ with each sub-filter $\mathbf{F}_i^{(j)}$ to get the response $r_{i,v}^{(j)}$ (d) Concatenate responses $r_{i,v}^{(j)}$ of all d' filters in the layer l for the spatial position j (e) Repeat (d) for all j -th spatial positions in the Filter $\mathbf{F}_i$ and store the responses of each j -th position in the Codebook as Lookup Responses for the Codeword $\mathbf{K}_v$.

where $\parallel$ denotes the concatenation operation. Consequently, each codeword will have N lookup responses $\mathbf{r}_v^{(j)}$, one for each j -th spatial position in the original $\mathbf{F}_i$.

3.3.2 QL-Net Training

So far, we have presented an algorithm for codebook construction. This codebook is used to approximate the CNN feature maps. However, approximation error (also referred to as quantization error) is introduced after approximating the input tensor, and hence the QL-Net model must be retrained.

The reason and challenges for retraining an approximated network are:

1. An approximation error is introduced each time an input is approximated by the codewords. This error causes the classification accuracy to drop. Therefore, CNN filters need to be retrained to minimize this error.

2. The approximation of a current layer depends on the quantization of earlier layers.

To address these challenges we propose the following QL-Net training procedure (Algorithm 1). We start with a pre-trained baseline model and perform the quantization process for one layer at a time. Let's denote the layer for quantization as a layer l and the total number of layers in the network by L . We start quantization and retraining with the $l = 2$ and finish at $l = L - 1$. Every time a layer l is quantized, all layers from 1 to $l - 1$ are treated as a fixed feature extractor.

A similar procedure for the quantization and re-training is repeated for every $1 < l < L$ layer :

1. approximate $\mathbf{B}_l$ with codewords from the codebook $\mathbb{C}_l$,
2. retrain parameters of filters from layers l to L .

We start the QL-Net training procedure by approximating $\mathbf{B}_l$ with the codewords from the layer l codebook $\mathbb{C}_l$. Let's denote this quantized input as $\hat{\mathbf{B}}_l$. The $\hat{\mathbf{B}}_l$ is used instead of $\mathbf{B}_l$ to calculate the layer output $\mathbf{B}'_l$. The substitution of $\mathbf{B}_l$ with $\hat{\mathbf{B}}_l$ increases the classification error. To minimize this error, retraining of the set of filters for the layer l till L should be performed. The pseudo-code of the algorithm for QL-Net training is described in Algorithm 1.

Algorithm 1: QL-Net Training

Input: images $\mathbf{I}$ selected at random from the dataset , labels t

model = train_baseline($\mathbf{I}$, t);

for l in range ($2, L - 1$) **do**

$\mathbf{B}_l \leftarrow \text{model}(\mathbf{I}, l)$;

$\mathbb{C}_l \leftarrow \text{construct}(\mathbf{B}_l, k, H)$;

for i in range ($1, \text{epochs}$) **do**

$\hat{\mathbf{B}}_l \leftarrow \text{lookup}(\mathbf{B}_l, \mathbb{C}_l, l)$;

model $\leftarrow \text{train}(\text{model}, \hat{\mathbf{B}}_l, t)$;

After the filters of layers l till L were modified, the retrained model is used to quantize the consecutive layers, from $l + 1$ to $L - 1$. For instance, to extract $\mathbf{B}_{l+1}$ for the layer $l + 1$, the retrained QL-Net model is used, with the layers 2 to l already quantized and properly restrained, and with lookup responses for these layers calculated.

3.3.3 Inference

During the inference, we approximate each patch $\mathbf{P}$ from the input tensor $\mathbf{B}$ with the closest key $\mathbf{K}$ from the Codebook and use calculated convolution output (lookup response vector) for the $\mathbf{K}$ instead of performing convolution on the input patch $\mathbf{P}$. With the spatial partitioning strategy introduced, the convolution, between the

FIGURE 3.7: QL-Net Inference. (a) Each feature patch is split into Feature Patch Columns of size $1 \times 1 \times d$. (b) These Feature Patch Columns and their spatial positions j are used to find the closest codewords from the Codebook. Once the closest codeword for a Feature Patch Column is found, the corresponding to the codeword lookup response vector at position j is retrieved. (c) Retrieved from the Codebook pre-computed lookup response vectors for each Feature Patch j -th spatial position are summed together to get the final approximation for the output of the convolution between the input Feature Patch and a set of filters of the layer l .

$M \times M \times d$ dimensional feature patch $\mathbf{P}$ and an equally sized filter $\mathbf{F}_i$, is substituted by a sum of N separate convolutions between the lower dimensional sub-filters $\mathbf{F}_i^{(j)}$ and the feature patch columns $\mathbf{P}^{(j)}$ (Figure 3.8(a)). $\mathbf{P}^{(j)}$ and $\mathbf{F}_i^{(j)}$ are components at j -th position in the Feature Patch and the filter $\mathbf{F}_i$, respectively. Accordingly, to replace the convolution by a lookup operation, we extract the lookup responses for each $\mathbf{P}^{(j)}$ ($j = 1 \dots N$) independently (Figure 3.8(b)) and sum them together (Figure 3.8(c)), to get the final lookup response for the whole feature patch $\mathbf{P}$. Since $\mathbf{P}$ is extracted from the input $\mathbf{B}$ in a spatially overlapping manner, the same codewords can be re-used to estimate the lookup responses for the multiple j positions.

FIGURE 3.8: QL-Net Lookup Function. (a) Partition Feature Patch into the Feature Patch Columns. (b) Lookup Feature Patch Column for Lookup Response. (c) Sum Lookup Responses.

3.4 Experiments

The proposed model was evaluated on four image classification datasets: MNIST [1], Fashion-MNIST [2], German Traffic Sign Recognition Benchmark [59] and CIFAR-10 [60]. Section 3.4.1 provides short description of each dataset along with sample images. The ablation study is provided in Section 3.4.2 and extended experimental results are provided in Section 3.4.3.

3.4.1 Datasets

MNIST Dataset is an image classification dataset [1]. It consists of 60000 training and 10000 test 28×28 gray-scale images representing digits ranging from 0 to 9. Sample images are provided in Figure 3.9.

FIGURE 3.9: MNIST Dataset Sample Images [1]

Fashion-MNIST Dataset is an image classification dataset [2]. It consists of 60000 training and 10000 test 28×28 gray-scale images representing 10 object classes, such as t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneakers, bag, and ankle boot. Sample images are provided in Figure 3.10.

FIGURE 3.10: Fashion-MNIST Dataset Sample Images [2]

German Traffic Sign Recognition Benchmark (GTSRB) is an image classification dataset [59]. It consists of 39209 training and 12630 test color images containing one traffic sign each from 43 classes. Image sizes vary between 15×15 to 250×250 pixels, however for our experiments we scaled all images to 48×48 pixels. The images have varying light conditions and backgrounds. Sample images are provided in Figure 3.11.

FIGURE 3.11: Sample Images from German Traffic Sign Recognition Benchmark (GTSRB) [59]

CIFAR-10 Dataset is an image classification dataset [60]. It consists of 50000 training and 10000 test 32×32 color images representing objects from 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Sample images are provided in Figure 3.12.

FIGURE 3.12: Sample Images from CIFAR-10 Dataset [60]

3.4.2 Ablation Study

3.4.2.1 Experimental Settings

The first set of experiments was conducted to verify the proposed design choices for the QL-Net look-up function and training. We performed an ablation study on the MNIST dataset and CNN architecture described in Table 3.1. The first column in

TABLE 3.1: CNN Architecture used in the ablation study experiments

Layer	Size	Filters	Codewords size
CONV	$5 \times 5 \times 1$	20	-
MP	2×2	-	-
CONV	$5 \times 5 \times 20$	40	$1 \times 1 \times 20$
MP	2×2	-	-
CONV	$4 \times 4 \times 40$	50	$1 \times 1 \times 40$
FC	50	10	-

Table 3.1 shows the type of the layer, with either CONV for a convolutional layer, MP for max-pooling, or FC for a fully connected layer. The second column indicates the size of the filters for the CONV layers, the size of the connectivity matrix for the FC layers, and a downsampling ratio for the MP layer. The third column indicates the number of filters for each layer. The last column demonstrates the codewords’ size. During training, the cross-entropy loss is minimized with ADAM[61]. Batch Normalization [62] with a minibatch size of 100 were used. To construct the H-K tree, we randomly selected 80000 data points extracted from 20000 images from the training set. The number of clusters each subset is divided into is $k = 2$.

3.4.2.2 Experimental Results

Experiments on the MNIST dataset show that the proposed model achieved 98.6% accuracy, compared to the MAC-based CNN model with 99.2% accuracy. The acceleration obtained using the same settings as for the model with the highest accuracy was 5.8 times.

FIGURE 3.13: Accuracy vs Total Codebook Size

Figure 3.13 shows the relation between the dictionary size and the accuracy. In the figure, “QL-Net” is the proposed model with the spatial partitioning and retraining strategies applied. “QL-Net no splitting” is a model where only the retraining

was applied, and the codebooks were constructed without the spatial partitioning. “QL-Net no Retraining” is the model where the codebooks were constructed using spatial partitioning, but no retraining was applied. As shown in Figure 3.13, the increase in the codebook size improves the model performance. However, after the total codebook size exceeds 100 codewords, the accuracy graph flattens. In fact, there are improvements with the increase in the codebook size but the accuracy improvement compared to the size increase is negligible. The possible explanation for this behavior is that the dictionary representational capacity is not improving anymore, and a more advanced method should be used for the codebook construction. The effect of the retraining can be observed in Figure 3.13, without the layer-wise retraining, the accuracy is around 20-22% lower than the accuracy of the retrained QL-Net. Experiments in Figure 3.13 demonstrate as well that the QL-Net performance is higher when the approximation of the Feature Patches is made using the multiple codewords. Additionally, the retrained QL-Net with smaller codewords is only slightly affected by the different sizes of the codebook and thus can achieve the highest relative speed up.

FIGURE 3.14: Acceleration vs the Total Codebook Size

Figure 3.14 shows the acceleration obtained when evaluating the MAC operation with respect to the lookup operation. As can be seen from Figure 3.14, a smaller codebook provides a higher acceleration, as a smaller number of operations are required to find a respective codeword. This is as expected because the number of MAC operations required for the one convolutional layer is constant. While a codebook with a smaller dictionary requires a smaller number of comparisons to

approximate a feature patch. Acceleration is achieved when the number of lookup comparisons is smaller than the number of operations required for the convolution.

Looking closer at both Figure 3.13 and Figure 3.14 it can be observed that the effective acceleration obtained by the proposed model is around 4 to 6 times. This corresponds to the best accuracy of the QL-Net in Figure 3.13.

3.4.3 Experiments

To show how the proposed method can be applied to different datasets we performed experiments on four benchmark image classification datasets (MNIST [1], Fashion-MNIST[2], German Traffic Sign Recognition Benchmark[59] and CIFAR-10[60]).

3.4.3.1 Experimental Settings

As a baseline for this experiments we selected ResNet-11 [35]. For QL-Net codebook construction, we selected $k = 8$ (the number of clusters each node in the H-K tree is divided into) and $H = 2, 3, 4, 5$ (depth of the H-K tree). To construct the H-K tree, we randomly selected 200000 data points. Acceleration was computed as a ratio between the number of Floating Point Operations (FLOPs) performed by the baseline model over the number of FLOPs performed by QL-Net.

3.4.3.2 Experimental Results

Table 3.2 provides experimental results on four image classification datasets (MNIST [1], Fashion-MNIST[2], German Traffic Sign Recognition Benchmark[59] and CIFAR-10[60]) with different QL-Net configurations.

As can be seen from Table 3.2 applying QL-Net to four different datasets results in high computational savings with a small drop in performance on all four datasets.

Dataset	Model	Acceleration	Accuracy
MNIST[1]	ResNet-11 [35]	1.0	99.58
	QL-Net ($H = 4$)	39.0	99.45
	QL-Net ($H = 3$)	50.1	99.28
	QL-Net ($H = 2$)	70.1	98.91
Fashion-MNIST[2]	ResNet-11 [35]	1.0	93.54
	QL-Net ($H = 4$)	39.0	90.91
	QL-Net ($H = 3$)	50.1	90.13
	QL-Net ($H = 2$)	70.1	89.06
CIFAR-10[60]	ResNet-11 [35]	1.0	86.99
	QL-Net ($H = 5$)	27.9	82.49
	QL-Net ($H = 4$)	33.15	80.83
GTSRB [59]	ResNet-11 [35]	1.0	98.8
	QL-Net ($H = 4$)	33.16	97.59

TABLE 3.2: Performance of the Proposed QL-Net vs. Baseline Performance.

3.5 Complexity Analysis

In the previous section, acceleration was computed as a ratio between the number of Floating Point Operations (FLOPs) performed by the baseline model over the number of FLOPs performed by QL-Net. In this section, we will provide a complexity analysis for the computational gain.

Let X, Y be a spatial dimensions of the input tensor $\mathbf{B}$ (Figure 3.1), such that $X = Y$. Let M be a spatial dimension of a filter $\mathbf{F}_i$, d be a depth of the filter and d' be a number of filters in a given convolutional layer l . Let S be a stride parameter during convolution operations and P be padding size. Hence, to perform a convolution between one input tensor $\mathbf{B}$ and a set of filters $\mathbb{F} \in \mathbb{R}^{M \times M \times d \times d'}$ of a layer l it will require N number of multiplications during convolution, where N is defined according to the equation 3.3

$$\begin{aligned}
N &= \left(\frac{X - M + 2P + S}{S} \times \frac{Y - M + 2P + S}{S} \right) \times M \times M \times d \times d' \\
&= \left(\frac{X - M + 2P + S}{S} \right)^2 \times M^2 \times d \times d'
\end{aligned} \tag{3.3}$$

Frequently, the stride $S = 1$, hence equation 3.3 can be simplified as following:

$$N = (X - M + 2P + 1)^2 \times M^2 \times d \times d' \tag{3.4}$$

Hence, to perform a convolution between an input tensor $\mathbf{B}$ and a set of filters $\mathbb{F}$ of a layer l it will require N multiplications and $(X - M + 2P + 1)^2 \times (M \times M - 1) \times (d - 1) \times d'$ summations. As multiplications are more expensive operations, in the rest of the section we will use only a number of multiplications for computational gain estimation.

To compare the performance of standard convolution with the proposed QL-Net look-up function, let's denote the depth of the codebook H-K tree as H and the number of leaf-nodes each node is split into as k . Then to perform approximation of one codeword $\mathbf{K} \in \mathbb{R}^{1 \times 1 \times d}$ we need N' operations, where N' is defined according to the equation 3.5

$$N' = X^2 \times f(d) \times H \times k \tag{3.5}$$

where $f(d)$ is the number of operations necessary for a lookup function. For look-up we use $L1$ distance, hence we can assume that $f(d) \approx d$ and the equation for computing the computational gain λ can be approximated as follows:

$$\lambda = \frac{N}{N'} = \frac{(X - M + 2P + 1)^2 \times M \times M \times d \times d'}{X^2 \times d \times H \times k} \approx \frac{M^2 \times d'}{H \times k} \tag{3.6}$$

For instance, for an input tensor of size $3 \times 3 \times 3$ ($X = 3, Y = 3, d = 3$) and for 512 filters ($d' = 512$) of size $3 \times 3 \times 3$ the resulting tensor object will be of the size $1 \times 1 \times 512$. In such case, the number of operations to perform the convolution is $3 \times 3 \times 3 \times 512$ multiplications, assuming $S = 1$ and $P = 0$.

Using an H-K tree as a codebook allows a codebook to have a hierarchical structure. Lets take codebook with depth $H = 5$ and with $k = 2$. Therefore, at maximum, we need to perform $H \times k = 10$ comparisons to find the codeword that approximates the input tensor. The comparison was performed using the l^1 -distance. Thus to transform an input tensor of size $3 \times 3 \times 3$ into a tensor object of size $1 \times 1 \times 512$ we perform at maximum $X^2 \times k \times H = 90$ comparisons with codewords $\mathbf{K} \in \mathbb{R}^{1 \times 1 \times 3}$, which means in total we need to perform $1 \times 1 \times 3 \times 90$ operations.

Thus, the computational gain can be estimated as $\frac{3 \times 3 \times 3 \times 512}{1 \times 1 \times 3 \times 90} \approx 51$. This shows that theoretically there is a fifty times acceleration using our method. These results agree with the experimental results. Naturally, this is only an approximate count because the different arithmetic operations may have different implementation costs.

3.6 Conclusion

The approach studied in this thesis targets to enhance computation in the convolutional neural networks (CNNs) to reduce the number of operations performed during the inference. We have evaluated the proposed algorithm on MNIST [1], Fashion-MNIST [2], German Traffic Sign Recognition Benchmark [59] and CIFAR-10 [60] datasets and obtained promising results. We demonstrated that the quantization by lookup is a promising solution for the network acceleration if the codebook can capture the data diversity and the network can compensate for the information loss.

Chapter 4

Understanding CNN Filters and Interpretation

This chapter extends the current understanding of the roles played by the filters in Convolutional Neural Network (CNN) and their interactions.

4.1 Introduction

Convolutional Neural Networks (CNN) provide solutions to a range of problems, from computer vision [4–11], raw audio generation [12], to general data analytics [13], to name some. Each network consists of a set of layers, where each layer transforms its input using a set of filters and produces a multi-channel output. The outputs of the filters are combined to obtain a likelihood that an input image belongs to a given set of classes. The network’s core is the set of filters located in different layers and tuned to different features. Each filter offers a different contribution for each class recognition indicated by filter weights. The weights are obtained by the network training process in order to balance the needs of different classes and provide the best average results over the entire training data, e.g., the highest average accuracy of classification of given samples from all classes. This chapter extends the current understanding of the roles played by the filters in Convolutional Neural Network (CNN) and their interactions.

During our work on CNN compression, we discovered that when some filters with a weak contribution to the classification of an object are removed, the classification accuracy for a substantial subset of classes increases against the expectation of a decrease. Therefore the present study investigates the usage of pruning for enhancing CNN filter interpretability and understanding.

Specifically, we are interested in answering the following questions:

- How significant is a filter’s presence for accurate classification of a particular class?
- What type of evidence does the filter provide for a class classification?
- Which subsets of classes share the same subsets of filters?

Answers to such questions could help extend solutions of a given problem to other related problems, e.g., obtain a good initial (or partial) estimate for retraining a classification network for fewer classes than the original, or from a given set of familiar filters, intelligently selecting an ensemble that covers the major characteristics of the new class (like a mixture of experts). Understanding the significance and connections among filters used could also contribute to the interpretability of the network.

This chapter is partly based on the work published as Kamila Abdiyeva, Martin Lukac, and Narendra Ahuja. "Remove to Improve?" *In International Conference on Pattern Recognition (ICPR) International Workshops and Challenges, Virtual Event, January 10–15, 2021, Proceedings, Part III*, pp. 146-161. Springer, 2021.

Section 4.2 provides an overview of the pruning algorithm we use to find answers to the aforementioned questions. In addition, Section 4.2 presents some definitions and measures used in the experiments. Section 4.3 presents the motivation and objectives of the experiments performed, their details, and their findings, vs the questions raised in the introduction section. Section 4.4 presents concluding remarks.

4.2 Definitions

In order to study the roles and relationships among filters in a convolutional network, we propose to use an activation-based pruning method, referred to as *Response Based Pruning (RBP)*. Below we first briefly present the main aspects and definitions related to RBP.

Let $\mathbb{F} = \{\mathbf{F}_1 \dots, \mathbf{F}_t\}$ be the set of all filters in a convolutional neural network (CNN) and $\mathbf{F}_i \in \mathbb{R}^{M \times M \times d}$ be a filter, where M is the filter's spatial dimension and d is the depth. Also, let a subset of filters for a given CNN layer l be denoted as $\mathbb{L}$ with $\mathbb{L} \subseteq \mathbb{F}$. Let $\mathbf{B} \in \mathbb{R}^{X \times Y \times d}$ be an input tensor to l (X and Y are spatial dimensions). To calculate the output tensor $\mathbf{B}'$ of the layer l , the input tensor $\mathbf{B}$ is convolved with the set of filters $\mathbb{L} = \{\mathbf{F}_1 \dots, \mathbf{F}_{d'}\}$ (as illustrated in Figure 4.1a). The output of the convolution between the input tensor $\mathbf{B}$ and a single filter $\mathbf{F}_i$ is $\mathbf{R}_i \in \mathbb{R}^{X' \times Y'}$ (later referred to as a **feature map**) given by eq. 4.1.

$$\mathbf{R}_i = \mathbf{F}_i \odot \mathbf{B}, \text{ and } \mathbf{B}'_{(:, :, i)} = \mathbf{R}_i \quad (4.1)$$

Let $\mathbb{D}$ be a dataset containing the N data samples, $\mathbb{C} = \{c_1, \dots, c_m\}$ be the set of all object classes in the dataset. Let $\mathbb{D}_c = \{\mathbf{I}_1^c \dots \mathbf{I}_{N'}^c\}$ be a subset of $\mathbb{D}$ ($\mathbb{D}_c \subset \mathbb{D}$) containing all data samples ($\mathbf{I}_j^c$) with the class label $c \in \mathbb{C}$. The number of samples in $\mathbb{D}_c$ is denoted as N' , such that $N' \leq N$. Finally, we denote by $\mathbf{R}_i(\mathbf{I}_j^c)$ the feature map for the filter $\mathbf{F}_i$ and the input data sample $\mathbf{I}_j^c \in \mathbb{D}_c$.

Definition 4.1 (Pruning, $\wp(\cdot)$). Specifically, pruning a layer l is a process of removing one or more $\mathbf{F}_i$ from $\mathbb{L}$ using some predefined criteria, resulting in a new set of filters $\wp(\mathbb{L}) = \hat{\mathbb{L}} = \{\mathbf{F}_1, \dots, \mathbf{F}_{\hat{d}}\}$. Hence, a new output is $\hat{\mathbf{B}}' = \hat{\mathbb{L}} \odot \mathbf{B}$ with $\hat{\mathbf{B}}' \in \mathbb{R}^{X' \times Y' \times \hat{d}}$ and $\hat{d} \leq d'$.

As a pruning criteria - conditions to determine which filters to remove from the network - we propose to threshold filters' **accumulated response** values.

Definition 4.2 (Accumulated response (r_i^c)). Accumulated response (r_i^c) for a class c and a filter $\mathbf{F}_i$ is defined as an average sum of all elements in a feature map $\mathbf{R}_i$ for all samples in $\mathbb{D}_c$. It is computed by eq. 4.2

$$r_i^c = \frac{1}{X'Y'N'} \sum_{j=1}^{N'} \sum_{x=1}^{X'} \sum_{y=1}^{Y'} \mathbf{R}_{x,y,i}(\mathbf{I}_j^c) \quad (4.2)$$

where $\mathbf{R}_{x,y,i}$ is a single element at a position x, y ($x \in X', y \in Y'$) in a feature map $\mathbf{R}_i$. A schematic representation for calculating r_i^c for an input B is shown in Figure 4.1b.

FIGURE 4.1: (a) Convolutional layer operation and (b) computation of the accumulated response r_i^c (for simplicity of visualization superscript c was omitted in the figure).

Definition 4.3 (Response Based Pruning (RBP)). Let $\mathbf{r}^c = \{r_1^c, r_2^c, \dots, r_t^c\}$ be a vector of accumulated responses of all filters in the network for a class c . For a given pruning ratio $\theta \in [0, 1]$, Response Based Pruning (RBP) refers to a process of removing $|\mathbf{r}^c| * \theta$ number of filters $\mathbf{F}_i$ with the smallest r_i^c values in $\mathbf{r}^c$, where $|\mathbf{r}^c|$ is a cardinality of $\mathbf{r}^c$.

For instance, for $\theta = 0.1$, the 10% of filters with the lowest accumulated responses will be removed.

The experiments will throughout involve several measures, which we list below.

Definition 4.4 (Accuracy ($\mathcal{A}_c^\theta$)). Accuracy is the performance of a model on a class c (viewed as a binary classification problem) for a given pruning ratio θ and computed using 4.3.

$$\mathcal{A}_c^\theta = \frac{tp + tn}{tp + fp + fn + tn}, \quad (4.3)$$

where tp (true positive) stands for correctly classified samples (belonging to a class c), tn (true negative) stands for correctly classified samples (not belonging to a class c), fn (false negative) stands for misclassified samples (not belonging to a class c), and fp (false positive) stands for misclassified samples (belonging to a class c).

Definition 4.5 (Pruned Filter Similarity ($\mathcal{P}$)). Let $\bar{\mathbb{F}}_c = \{\mathbf{F}_1, \dots, \mathbf{F}_z\}$ be a set of filters pruned at some θ for a class c . Then the similarity between two sets of filters $\bar{\mathbb{F}}_{c_1}$ and $\bar{\mathbb{F}}_{c_2}$ pruned separately for two classes (c_1, c_2) is computed by eq. 4.4.

$$\mathcal{P}(c_1, c_2) = \frac{|\bar{\mathbb{F}}_{c_1} \cap \bar{\mathbb{F}}_{c_2}|}{|\bar{\mathbb{F}}_{c_1} \cup \bar{\mathbb{F}}_{c_2}|}, \quad (4.4)$$

where $|\cdot|$ is a cardinality of a set.

Definition 4.6 (Highly Overlapping Classes (HOC)). If two classes have pruned filter similarity higher than a specified threshold σ ($\mathcal{P}(c_1, c_2) \geq \sigma$, where σ is a similarity threshold), then classes c_1 and c_2 are referred to as Highly Overlapping Classes (HOC).

Definition 4.7 (Group of Classes ($\mathbb{G}$)). A group of classes $\mathbb{G}$ is a set consisting of multiple highly overlapping classes. $\mathbb{G} = \{c_i, c_j \in \mathbb{C} \mid \mathcal{P}(c_i, c_j) \geq \sigma \text{ and } 1 \leq i, j \leq m \text{ and } i \neq j\}$, where m is the number of classes in the dataset.

Definition 4.8 (k-Closest Neighbors($\mathbb{K}$)). k classes with the highest pruned filter similarity $\mathcal{P}$ to a class $c \in \mathbb{C}$ are referred to as k-Closest Neighbors $\mathbb{K}$ for a class c .

Definition 4.9 (Semantic Similarity ($\mathcal{S}$)). Semantic similarity between two classes $c_1, c_2 \in \mathbb{C}$ is measured using Wu-Palmer [63] similarity between nodes in the WordNet graph structure corresponding to the ImageNet class names.

We use Wu-Palmer [63] similarity as it provides a normalized, bounded value for the similarity, $\mathcal{S} \in (0, 1]$.

Definition 4.10 (Semantically Related Neighbors (SRN)). If semantic similarity between two classes $\mathcal{S}(c_1, c_2) \geq \sigma$, where σ is a similarity threshold, then classes c_1 and c_2 are semantically related, and are referred to as Semantically Related Neighbors (SRN).

Definition 4.11 (Semantically Unrelated Neighbors (SUN)). If semantic similarity between two classes $\mathcal{S}(c_1, c_2) < \sigma$ then classes c_1 and c_2 are referred to as Semantically Unrelated Neighbors (SUN), where σ is a similarity threshold.

4.3 Experimental Study and Analysis

The first set of experiments was conducted to observe the changes in classification accuracy at different pruning ratios θ (Section 4.3.1). As expected, the majority of classes started to decrease their classification accuracy. However, we noticed that for specific values of θ , there are sets of classes $\mathbb{I}$ for which classification accuracy improves with pruning (Section 4.3.1). To determine the conditions under which an

accuracy improves, we experimentally investigated the usage and contribution of individual filters for the classification of classes in $\mathbb{I}$ (Section 4.3.2). As a result, we extracted groups of the most active filters specific for each class. A natural next step was to determine a) the possible overlaps of used filters between classes in $\mathbb{I}$ (Section 4.3.3) and b) the possible semantic relation between classes in $\mathbb{I}$ (Section 4.3.3). We observed a) the existence of groups of classes $\mathbb{G}$ for which the classification accuracy improves by pruning similar sub-sets of filters (Section 4.3.3) and b) that semantically related classes tend to prune similar sub-sets of filters (Section 4.3.3). In the following experiment we study the structure of $\mathbb{G}$ in more details by looking at semantic relation with closest neighbouring classes (Section 4.3.4) and how the filter overlap with these classes changes at different θ (Section 4.3.5). This experiment allowed us to determine that a) classes for which the classification accuracy is improved by pruning have, on average, a higher number of semantically related neighbors, and b) improved classes have, on average, a higher increase in correlation with semantically related classes. Besides, this experiment showed an apparent interference between filters learned for different classes, as classification accuracy improved by reducing the filter overlap between the classes.

For all experiments, we use AlexNet [64] trained on the ImageNet [33] dataset.

ImageNet Dataset is a benchmark image classification dataset [33] with roughly 1.2 million training images, 50000 validation color images representing 1000 object classes. For evaluation, we used the ImageNet validation set. Following the convention, all images were resized to 256×256 resolution. Sample images from the dataset are provided in Figure 4.2.

AlexNet consists of five convolutional and three fully connected layers. RBP prunes only the convolutional layers, also referred to as the feature extractor. Filters of the first convolutional layer ($l = 1$) are referred to as early-layer filters. Filters of the second and third convolutional layers ($l = [2, 3]$) are referred to as intermediate-layers filters. Finally, the filters of the fourth and fifth convolutional layers ($l = [4, 5]$) are referred to as late-layers filters. Pruning ratio θ is set to 0 and increased by 0.05 until reaching $\theta = 0.5$ (representing pruning 50% of filters in the network). No retraining is performed after pruning.

FIGURE 4.2: Sample images from ImageNet dataset

4.3.1 Class-wise Accuracy Changes

We applied RBP to all classes $c \in \mathbb{C}$ in the ImageNet dataset and recorded their classification accuracies $\mathcal{A}_c^\theta$ for various pruning ratios $\theta \geq 0$. As expected, similar to the previous works, an average classification accuracy over all classes monotonically decreases with increasing θ . However, we observed that there are sets of classes for which classification accuracy improved by pruning for specific values of θ .

Figure 4.3 shows the number of classes that increased the classification accuracy after pruning. One can observe from Figure 4.3 that despite the global trend of accuracy reduction with pruning there is a significant number of classes $c \in \mathbb{C}$ for which accuracy improved after pruning ($\mathcal{A}_c^\theta - \mathcal{A}_c^{\theta=0} > 0$, for some values of θ , and where $\mathcal{A}_c^{\theta=0}$ is original accuracy for the class c before pruning). For simplicity in the rest of the chapter $\mathcal{A}_c^{\theta=0}$ will be referred to as $\mathcal{A}_c^0$. According to Figure 4.3 the highest number of classes with improved classification accuracy observed at a smaller $\theta = [0.05, 0.1]$, is 272 and 208 respectively. As expected, this number decreases with an increase in the pruning ratio value θ and achieves only seven classes at $\theta = 0.5$.

The possible reason for improving accuracy with pruning is that improved accuracy classes had interference from the pruned filters. To investigate this hypothesis in

FIGURE 4.3: The number of classes $c \in \mathbb{C}$ for which the classification accuracy improved after pruning for different pruning ratios θ .

more detail, we study each filter's contribution to class classification and what filters are being removed by pruning.

4.3.2 Identification of Filters Contribution to Each Class Recognition

The next step was to investigate the accumulated response magnitudes (as defined in Definition 4.2) of each filter $\mathbf{F}_i$ while classifying each class $c \in \mathbb{C}$, and identify what groups of filters strongly contributed (have a high value of accumulated response) to each class classification.

Figure 4.4 illustrates how many classes pruned a particular filter at a pruning ratio θ (horizontal axis (x-axis) represents $\mathbf{F}_i$ and vertical axis (y-axis) is θ). The intensity of each bar in the graph is proportional to the number of classes that pruned the filter. Lighter color indicates a higher number of classes.

One can observe from Figure 4.4 that early-layer filters (the blue rectangle in Figure 4.4) were barely pruned at any pruning ratio θ . At the same time for a small $\theta = [0.05; 0.15]$, mainly intermediate-layers (mid-layers) filters (the green

rectangle in Figure 4.4) were pruned. However, once a bigger $\theta \geq 0.2$ is reached more and more late-layers filters (the purple rectangle in Figure 4.4) were pruned.

FIGURE 4.4: The number of classes that pruned a particular filter at a different θ . The intensity of the graph color is proportional to the number of classes. Lighter colors indicate the higher numbers.

To investigate the contribution of accumulated responses (as defined in Definition 4.2) in more details we plotted accumulated response values r_i^c of each filter $\mathbf{F}_i$ for each class $c \in \mathbb{C}$ in Figure 4.5 (the horizontal axis (x-axis) represents $\mathbf{F}_i$, while the vertical axis (y-axis) represent classes in $\mathbb{C}$). For a better visualization values of r_i^c below -10 were capped at -10.

On one hand, as can be seen from Figure 4.5, early-layer filters (the blue rectangle in Figure 4.5) were highly used and contribute almost equally to the recognition of all classes. Therefore, they were not pruned at any pruning ratio θ (Figure 4.4). On the other hand, intermediate-layers (mid-layers) filters (the green rectangle in Figure 4.5) and late-layers filters (the purple rectangle in Figure 4.5) are more class specific.

FIGURE 4.5: Accumulated Response r_i^c of each filter $\forall \mathbf{F}_i \in \mathbb{F}$ for each class $\forall c \in \mathbb{C}$. (A) Filters ranges 200-600 have a high accumulated responses for classes 0-400. (B) Filters ranges 200-600 have small accumulated responses for classes 400-900

Additionally, the rectangle A in Figure 4.5 shows that filters in the range between 200 and 600 have high accumulated responses for classes 0-400. While the rectangle B shows that the same filters (filters ranges 200-600) have small accumulated responses for classes 400-900. The observations in Figure 4.5 suggest that there are groups of classes $\mathbb{G}$ which use and prune similar sub-sets of filters. It can be explained by the fact that class labels in ImageNet are organized according to the WordNet semantic hierarchy. Based on the observations from Figure 4.5 we hypothesize that classes that use and prune similar filters are located closely in the WordNet semantic hierarchy.

Therefore, the next set of experiments will provide a more precise quantitative evaluation of groups formed, intra-group pruned filter similarity ($\mathcal{P}$), and semantic similarity.

4.3.3 Class-wise Pruned Filter Similarity and Semantic Similarity

Previous experiments showed that there are groups of classes $\mathbb{G}$ which share and prune similar sub-sets of filters. The next set of experiments was conducted to identify a more precise quantitative evaluation of groups formed and their properties. In particular, we focused on investigating the dependencies between a class-wise pruned filter similarity $\mathcal{P}$ and a semantic similarity $\mathcal{S}$.

Figure 4.6a shows results of a pruned filter similarity $\mathcal{P}$ (computed according to the Eq. 4.4) between all classes in $\mathbb{C}$ for $\theta = 0.05$. To obtain Figure 4.6a, we pruned each class c independently at $\theta = 0.05$, and we computed the pruned filter similarity $\mathcal{P}$ for the pruned filters $\bar{\mathbb{F}}_c$ of all $c \in \mathbb{C}$. Both axes in Figure 4.6a represents classes from the dataset. The intensity of each pixel in Figure 4.6a is proportional to the pruned filter similarity between classes corresponding to the rows and columns. For instance, a lighter color pixel on an intersection between a row and a column means that two classes represented by the column and the row have a high pruned filter similarity. Figure 4.6a supported the hypothesis that indeed there are easily identifiable groups of classes that have a similar value of pruned filter similarity.

FIGURE 4.6: (A) Pruned Filter Similarity $\mathcal{P}$ for $\theta = 0.05$, the intensity of a pixel is proportional to the number of common pruned filters between the corresponding classes (B) Semantic Similarity $\mathcal{S}$

Since the pruned filter similarity (Figure 4.6a) resulted in such a clear overlap pattern, we pursue a deeper study by analyzing semantic relations between classes

in the dataset. Since ImageNet classes follow the WordNet class hierarchy, we measure semantic similarity using corresponding metrics used in WordNet, e.g., distances along the Wordnet ontological graph structure. We use Wu-Palmer [63] similarity between WordNet nodes corresponding to the ImageNet class labels. Figure 4.6b shows the magnitude of Wu-Palmer semantic similarity between all classes in $\mathbb{C}$.

The high similarity between Figure 4.6a and Figure 4.6b is supporting the hypothesis that semantically similar classes prune similar filters.

4.3.4 Pruned Filter Similarity and Semantic Similarity between k-Closest Neighbors

Previous experiments showed that there are groups of classes $\mathbb{G}$ which share and prune similar sub-sets of filters and have high semantic similarity. To quantify the pruned filter similarity and semantic similarity between classes within each group $\mathbb{G}$, for each class $c \in \mathbb{G}$ we take $k = 5$ classes with the highest pruned filter similarity $\mathcal{P}$ (k-Closest neighbors). We then calculate how many out of $k = 5$ neighbouring classes are semantically related to c . We also look at how the number of semantically related classes changes for different pruning ratios θ .

Let $\mathbb{K} = \{c_1, \dots, c_k\}$ be a set containing k classes with the highest pruned filter similarity $\mathcal{P}$ to a class c . Then, let $\mathbf{s}^c = \{\mathcal{S}_1, \dots, \mathcal{S}_k\}$ be a vector containing the semantic similarity between a class c with all classes in $\mathbb{K}$, where $\mathcal{S}_j = \mathcal{S}(c, c_j)$ and $k = 5$.

We count how many classes in $\mathbb{K}$ are Semantically Related Neighbors (SRN : $\mathcal{S}(c, c_j) > \sigma$) and Semantically Unrelated Neighbors (SUN) to class c . We selected a similarity threshold $\sigma = 0.5$ for identifying SRN and SUN.

The column AV in Table 4.1 shows how many neighbors are SRN (Eq. 4.5) on average over the whole set of classes $\mathbb{C}$ (including improved, reduced, and classes that didn't change classification accuracy). The average number of SUN is complementary to the average number of SRN in the sense that $\text{AV}(\text{SUN}) = k - \text{AV}(\text{SRN})$, where k is a number of classes in $\mathbb{K}$ and $k = 5$ in our experiments. Hence, for simplicity of visualization, we provided only numbers for SRN in Table 4.1. Equation 4.6 provides an equation for computing $\text{AV}(\text{SUN})$ independently from $\text{AV}(\text{SRN})$.

$$AV(SRN) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k g(\mathcal{S}(c_i, c_j)) , \text{ where } g(x) := \begin{cases} 0, & \text{if } x \leq \sigma \\ 1, & \text{if } x > \sigma \end{cases} \quad (4.5)$$

where m is a total number of classes in $\mathbb{C}$ and k is a number of classes in $\mathbb{K}$ for each class c_i .

$$AV(SUN) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k h(\mathcal{S}(c_i, c_j)) , \text{ where } h(x) := \begin{cases} 1, & \text{if } x \leq \sigma \\ 0, & \text{if } x > \sigma \end{cases} \quad (4.6)$$

where m is a total number of classes in $\mathbb{C}$ and k is a number of classes in $\mathbb{K}$ for each class c_i .

The column I in Table 4.1 shows how many SRN classes are on average among the five closest neighbors for classes that improve classification accuracy after pruning at θ ($\mathcal{A}_c^\theta - \mathcal{A}_c^0 > 0$). The column entitled R shows the average number of SRN for classes for which the classification accuracy is reduced after pruning at θ ($\mathcal{A}_c^\theta - \mathcal{A}_c^0 < 0$). As the number of classes that didn't change classification accuracy by pruning was too small, we did not study them in more details. The number of semantically unrelated neighbors (SUN) is complementary to SRN for each case in Table 4.1.

Previous experiments (Figure 4.6) showed that semantically similar classes tend to prune similar filters. Hence, one can expect that the number of SRN in $\mathbb{K}$ will be approaching k . This hypothesis is supported by the results in Table 4.1, where values for SRN are much higher than for SUN. One can observe from Table 4.1 that for the first two pruning ratios $\theta = 0.05$ and $\theta = 0.10$, the highest values for SRN (4.033 and 4.192 respectively) are observed for classes for which the classification accuracy increased after pruning (Column I). However, once the pruning ratio gets above $\theta \geq 0.1$, the highest values of SRN (4.077, 4.115, 4.124, and 4.137 respectively) are observed for classes with the reduced accuracy (Table 4.1 Column R). This observation can be explained as follows. At a small θ , pruning removes interfering filters that were introducing distortion into the target class classification. Therefore, for a small θ pruning moves SRNs closer together. However, once we start to prune a more significant portion of the filters ($\theta \geq 0.1$), pruning starts to

remove filters that allowed the discrimination between similar/semantically related classes. Now, the network tends to confuse more classes belonging to SRN and, hence, reduces the classification accuracy.

TABLE 4.1: An Average Number of Semantically Related Neighbors (SRN). (I) and (R) represent average numbers of SRN for improved and reduced accuracy classes. AV is an average number of SRN for all classes.

Pruning ratio	AV	I	R
$\theta = 0.05$	3.932	4.033	3.935
$\theta = 0.10$	4.030	4.192	3.994
$\theta = 0.15$	4.051	3.922	4.077
$\theta = 0.20$	4.079	3.991	4.115
$\theta = 0.25$	4.110	3.973	4.124
$\theta = 0.30$	4.125	3.918	4.137

One of the possible conclusion from Table 4.1 is that for a small θ classes for which the classification accuracy improves, the pruning tends to group SRN together. Hence, the next set of experiments investigates how pruned filter similarity is changing by pruning.

4.3.5 Changes in Pruned Filter Similarity with k-Closest Neighbors with Pruning

We have seen that there are identifiable groups of classes $\mathbb{G}$ with a high pruned filter similarity. We also have observed that a high pruned filter similarity between classes also implies a high semantic similarity. Besides, it turns out that classes for which the classification accuracy had improved at smaller pruning ratios ($\theta = [0.05, 0.1]$) also had a higher number of semantically related neighbors (SRN) than classes for which the classification accuracy was not improved by pruning. Based on this observation, we hypothesize that for a small θ ($\theta = [0.05, 0.1]$), classes for which pruning increases the number of closest SRN have higher chances to increase the classification accuracy. In addition, classes with increased accuracy by pruning have less SUN than classes for which the classification accuracy is reduced by pruning

at lower θ . To verify this hypothesis, in this section, we experimentally determine how the pruned filter similarity between the semantic neighbors of improved and non-improved classes changes with an increase in θ .

Let $\mathbf{p}_c^\theta$ be a vector containing the pruned filter similarity between a class c and its k -Closest neighbors $\mathbb{K}$ at θ ($k = 5$). For the simplicity of visualization $\mathbf{p}_c^\theta$ will be denoted as $\mathbf{p}^\theta$ in the rest of the chapter. To determine the change in the pruned filter similarity, we calculate the element-wise difference (Δ) between $\mathbf{p}^{\theta_t}$ and $\mathbf{p}^{\theta_{t-1}}$ for class $c_i \in \mathbb{C}$ using eq 4.7 and eq 4.8, where θ_t and θ_{t-1} stands for the current and previous pruning ratios respectively. We separate $\mathbb{K}$ into Semantically Related neighbors (SRN) and Semantically Unrelated neighbors (SUN). Then we compute the average fraction of increased similarity separately for SRN (Δ_{SRN}) and SUN(Δ_{SUN}).

$$\Delta_{SRN}(\mathbf{p}^{\theta_t}, \mathbf{p}^{\theta_{t-1}}) = \frac{1}{k'} \sum_{j=1}^k g(\mathcal{S}(c_i, c_j)) * f(p_j^{\theta_t} - p_j^{\theta_{t-1}}), \text{ where } f(x) := \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases} \quad (4.7)$$

where k is a total number of classes in $\mathbb{K}$, k' is a number of SRN in $\mathbb{K}$, $g(\cdot)$ is the same function as defined in eq. 4.5, $\mathcal{S}$ is the semantic similarity between two classes and $p_j^{\theta_t} = \mathcal{P}(c_i, c_j)$.

$$\Delta_{SUN}(\mathbf{p}^{\theta_t}, \mathbf{p}^{\theta_{t-1}}) = \frac{1}{k''} \sum_{j=1}^k h(\mathcal{S}(c_i, c_j)) * f(p_j^{\theta_t} - p_j^{\theta_{t-1}}), \text{ where } f(x) := \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases} \quad (4.8)$$

where k is a total number of classes in $\mathbb{K}$, k'' is a number of SUN in $\mathbb{K}$, $h(\cdot)$ is the same function as defined in eq. 4.6, $\mathcal{S}$ is the semantic similarity between two classes and $p_j^{\theta_t} = \mathcal{P}(c_i, c_j)$.

Table 4.2 shows the results of calculating the average fraction of increased similarity for Δ_{SRN} and Δ_{SUN} in pruned filter similarity over all classes $c_i \in \mathbb{C}$ and their corresponding k -closest neighbors for $\theta_t = 0.1, 0.15, 0.2, 0.25, 0.3$.

The values of average fraction of increased similarity in Table 4.2 can be interpreted as follows. For example, let $\mathbb{K} = \{c_1, c_2, c_3, c_4, c_5\}$ be the 5 closest neighbors of the class c at θ_{t-1} . Let classes c_1, c_2, c_3 be SRN to c ($k' = 3$) and classes c_4, c_5 be SUN ($k'' = 2$). Then, for both θ_{t-1} and θ_t , we measure the pruned filter similarity

$\mathcal{P}$ between the class c and its SRN classes ($\mathcal{P}(c, c_1)$, $\mathcal{P}(c, c_2)$, $\mathcal{P}(c, c_3)$). If the magnitude of $\mathcal{P}$ increased when moving from θ_{t-1} to θ_t for all SRN classes then Δ_{SRN} will be equal to 1. Otherwise, we take a ratio between the number of SRN classes for which the pruned filter similarity increased vs the total number of SRN (k'). For instance, assume that the value of $\mathcal{P}$ increased only for c_1 and c_2 then $\Delta_{SRN} = \frac{2}{3} = 0.67$. The value of Δ_{SUN} can be interpreted in a similar manner.

Table 4.2 shows the values of Δ_{SRN} and Δ_{SUN} averaged over all classes $c \in \mathbb{C}$ (column AV), all improved classes $c \in \mathbb{I}$ (column I) and all reduced classes $c \in \mathbb{R}$ (column R).

TABLE 4.2: Average Fraction of Increased Similarity for semantically related (Δ_{SRN}) and unrelated (Δ_{SUN}) neighbors. The higher the value in the table, the higher is the number of neighbouring classes with which the pruned filter similarity increases. (I) and (R) represent average change in pruned filter similarity for improved and reduced classes.

Pruning ratios	AV		I		R	
	Δ_{SRN}	Δ_{SUN}	Δ_{SRN}	Δ_{SUN}	Δ_{SRN}	Δ_{SUN}
$\theta_{t-1} = 0.05, \theta_t = 0.10$	0.948	0.918	0.969	0.905	0.943	0.918
$\theta_{t-1} = 0.10, \theta_t = 0.15$	0.647	0.672	0.688	0.659	0.637	0.667
$\theta_{t-1} = 0.15, \theta_t = 0.20$	0.572	0.590	0.654	0.642	0.560	0.581
$\theta_{t-1} = 0.20, \theta_t = 0.25$	0.710	0.738	0.732	0.790	0.708	0.731
$\theta_{t-1} = 0.25, \theta_t = 0.30$	0.697	0.696	0.699	0.638	0.695	0.700

The most notable observation in Table 4.2 is that for the majority of θ_t , the values in Column I (Δ_{SRN}) are higher than the values in Column I (Δ_{SUN}). At the same time, the majority of the values in Column R (Δ_{SRN}) are lower than the values in Column R (Δ_{SUN}). In addition, all values in Column I (Δ_{SRN}) are always higher than the values in Column R (Δ_{SRN}). This observation suggests that the classes for which Δ_{SRN} is higher than Δ_{SUN} have higher chances of classification accuracy to be improved by pruning.

Once the pruning is performed, we are left with a smaller number of features to discriminate classes. We hypothesize that with fewer filters, it is easier to encode semantically similar classes rather than classes from completely different groups. This hypothesis is supported by the results observed in Table 4.2. On the one

hand, for classes where pruning increases the pruned filter similarity with SUN, the classification accuracy decreases. The possible explanation for such behavior is that it is harder to capture the diversity of unrelated classes with a smaller number of filters. On the other hand, for classes where pruning increases the pruned filter similarity with SRN, we observed an increase in the classification accuracy. This observation supported the original hypothesis that semantically similar classes are easier to encode with a smaller set of filters. However, once we prune too many filters, it is becoming much harder to discriminate between similar classes as well.

4.4 Discussion and Conclusions

The results presented in this work elicit several trends. First, for small pruning ratios θ , classes whose closest neighbors are semantically related have a higher probability of being improved by RBP (Experiments 4.3.4). Second, as a result of RBP in Experiment 4.3.5 classes whose accuracy improved have more increase in pruned filter similarity with semantically related neighbors than with semantically unrelated.

Based on our experiments, we can conclude that whether the classification accuracy for a given set of classes can be potentially improved by pruning can be determined by checking if a large fraction of neighbors is semantically related and there is an initial increase in pruned filter similarity with semantically related neighbors.

Our results could be used to design filter selection masks to increase the classification accuracy of a group of classes.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis provides two studies for enhancing the computation and understanding of CNN. The first approach provides a study to enhance computation in the convolutional network by reducing the number of operations during the inference. It demonstrated that the quantization by lookup is a promising solution for the CNN network operation enhancement if the codebook can capture the data diversity and the network can compensate for the information loss. Future work will include investigating techniques to improve the codebook quality and alternative approaches for lookup, such as fast hash functions.

The second study on enhancing CNN understanding showed that for small pruning ratios, the classification accuracy for a given set of classes could be improved by pruning if a significant fraction of neighbors is semantically related, and there is an initial increase in pruned filter similarity with semantically related neighbors. Future work will use the study results to design filter selection masks to increase the classification accuracy of a group of classes. Such masked networks could be combined into ensembles for decision-making.

5.2 Future Work

As future work, we want to extend the research on network understanding through the study of different optimization approaches. In particular, we propose to study how knowledge stored inside CNN is compressed by different optimization methods such as pruning (structural transformations), quantization/binarization (precision reduction), or knowledge transfer methods. The target of this approach is to obtain a highly compressed knowledge representation and to remove redundant information accumulated during the training process. Finding the right compression approach will help to develop a general methodology for designing compact, task-specific neural networks.

Concretely, the first direction for future works is to study the limits of CNN model capacity by using precision reduction methods. For instance, Fully Binary Networks, which provide the highest possible model information compression, can be used to determine how much variety in the input information is required for high-quality knowledge storage. In addition, this approach can be combined with both QL-Net and pruning methods. As a result of these approaches, we expect that the combination of structural and precision reduction methods will lead to a better understanding of how neural networks store information and how it should be compressed.

The second future work direction is to study alternative approaches for compact network design such as knowledge distillation methods [65, 66]. In particular, we are interested in studying the optimal ways to transfer information stored inside a full-precision model to a reduced precision model (the binary model in particular). The ultimate goal is to combine several optimization methods to better evaluate the limits of the neural network knowledge storage capacity.

List of Author's Publications

Thesis Publications

- **Kamila Abdiyeva**, Martin Lukac and Narendra Ahuja, "Remove to Improve?" in *International Conference on Pattern Recognition (ICPR) International Workshops and Challenges, Virtual Event, January 10–15, 2021, Proceedings, Part III*, pp. 146-161, Springer, 2021.
- **Kamila Abdiyeva**, Kim-Hui Yap, Gang Wang, Narendra Ahuja and Martin Lukac, "QL-Net: Quantized-by-Lookup CNN," in *15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, November 18-21, 2018*, pp. 413-418, IEEE, 2018.

Other Publications

- **Kamila Abdiyeva**, Timur Tibeyev and Martin Lukac, "Capacity Limits of Fully Binary CNN," in *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL), Miyazaki, November 9-11, 2020*, pp. 206-211, IEEE, 2020. doi: 10.1109/ISMVL49045.2020.000-4.
- Martin Lukac, Ayazkhan Bayanov, Albina Li, **Kamila Abdiyeva**, Nadira Izbassarova, Magzhan Gabidolla and Michitaka Kameyama, "Selecting Algorithms Without Meta-features," in *International Conference on Pattern Recognition (ICPR) International Workshops and Challenges, Virtual Event, January 10–15, 2021, Proceedings, Part IV*, pp. 607-621, Springer, 2021.

- Martin Lukac, **Kamila Abdiyeva** and Michitaka Kameyama, "CNOT-Measure Quantum Neural Networks," *in 2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL), Linz, May 16-18, 2018*, pp. 186-191, IEEE, 2018. doi: 10.1109/ISMVL.2018.00040.
- Jun Liu, Gang Wang, Ling-Yu Duan, **Kamila Abdiyeva** and Alex C. Kot, "Skeleton-Based Human Action Recognition With Global Context-Aware Attention LSTM Networks," *in IEEE Transactions on Image Processing*, vol. 27, no. 4, pp. 1586-1599, April 2018. doi: 10.1109/TIP.2017.2785279.

Technology Disclosure

- Object Detection And Classification System For Near Real-Time Surveillance Applications (TD/067/17)

Bibliography

- [1] 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. [xvii](#), [13](#), [27](#), [32](#), [33](#), [35](#)
- [2] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. URL <http://arxiv.org/abs/1708.07747>. cite arxiv:1708.07747Comment: Dataset is freely available at <https://github.com/zalandoresearch/fashion-mnist> Benchmark is available at <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/>. [xvii](#), [27](#), [28](#), [32](#), [33](#), [35](#)
- [3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL <http://dl.acm.org/citation.cfm?id=104279.104293>. [7](#)
- [4] Wanli Ouyang, Xiaogang Wang, Xingyu Zeng, Shi Qiu, Ping Luo, Yonglong Tian, Hongsheng Li, Shuo Yang, Zhe Wang, Chen-Change Loy, and Xiaoou Tang. DeepID-Net: Deformable deep convolutional neural networks for object detection. *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 2403–2412, 2015. ISSN 1550-5499. doi: 10.1109/CVPR.2015.7298854. [7](#), [37](#)
- [5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. (9), 2015. ISSN 0163-6804. doi: 10.1109/MCOM.2016.7378425. URL <http://arxiv.org/abs/1511.06581>.
- [6] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. *ICCV*, 2017-Octob:2999–3007, 2017. ISSN 15505499. doi: 10.1109/ICCV.2017.324.
- [7] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. 2017. URL <http://arxiv.org/abs/1709.01507>.
- [8] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In

- Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 487–495, Cambridge, MA, USA, 2014. MIT Press. URL <http://dl.acm.org/citation.cfm?id=2968826.2968881>.
- [9] Haoyu Ren and Ze-Nian Li. Object Detection Using Generalization and Efficiency Balanced Co-Occurrence Features. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 46–54, 2015. doi: 10.1109/ICCV.2015.14. URL <http://ieeexplore.ieee.org/document/7410371/>.
- [10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE CVPR*, pages 580–587, 2014. ISBN 9781479951178. doi: 10.1109/CVPR.2014.81.
- [11] Farzad Husain, Babette Dellen, and Carme Torras. Chapter 20 - scene understanding using deep learning. In Pijush Samui, Sanjiban Sekhar, and Valentina E. Balas, editors, *Handbook of Neural Computation*, pages 373 – 382. Academic Press, 2017. ISBN 978-0-12-811318-9. URL <http://www.sciencedirect.com/science/article/pii/B978012811318900020X>. 7, 37
- [12] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016. URL <https://arxiv.org/abs/1609.03499>. 7, 37
- [13] Maryam M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, Feb 2015. ISSN 2196-1115. 7, 37
- [14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012. 11, 13, 17
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. 2016. URL <http://arxiv.org/abs/1603.05027>.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016. 11, 17
- [17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, 2015. 11, 17
- [18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 11, 17

- [19] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 5 2015. ISSN 0028-0836. doi: 10.1038/nature14539. [11](#), [17](#)
- [20] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *TPAMI*, 2016.
- [21] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv preprint arXiv:1703.06870*, 2017. [11](#), [17](#)
- [22] Matthieu Courbariaux, Yoshua Bengio, and J David. Binaryconnect: Training deep neural networks with binary weights during propagations. *NIPS*, 2015. [12](#)
- [23] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*. 2016. [12](#), [13](#)
- [24] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El Yaniv, and Yoshua Bengio. Binarized neural networks. 2016. URL <http://arxiv.org/abs/1602.02505>. [12](#), [13](#)
- [25] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *ICLR*, 2016.
- [26] 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. 2016. URL <http://arxiv.org/abs/1606.06160>. [12](#)
- [27] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast neural networks without multipliers. *Trans. Neur. Netw.*, 4(1):53–62, January 1993. ISSN 1045-9227. doi: 10.1109/72.182695. URL <http://dx.doi.org/10.1109/72.182695>. [12](#)
- [28] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016. URL <http://arxiv.org/abs/1605.04711>. [12](#)
- [29] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *ICLR*, 2017. [12](#)
- [30] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016. [12](#)
- [31] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems 30*, pages 345–353. Curran Associates, Inc., 2017. [12](#), [13](#)
- [32] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. [12](#), [13](#)

- [33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015. [13](#), [42](#)
- [34] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL <https://proceedings.neurips.cc/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf>. [13](#)
- [35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016. [13](#), [32](#), [33](#)
- [36] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *CoRR*, abs/1512.08571, 2015. URL <http://arxiv.org/abs/1512.08571>. [13](#)
- [37] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. URL <http://arxiv.org/abs/1608.08710>.
- [38] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. 2017. URL <https://arxiv.org/abs/1710.01878>.
- [39] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability.
- [40] Ari S. Morcos, David G. T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. art. arXiv:1803.06959, Mar 2018. [15](#)
- [41] Xiaolong Ma, Geng Yuan, Sheng Lin, Zhengang Li, Hao Sun, and Yanzhi Wang. ResNet Can Be Pruned 60x: Introducing Network Purification and Unused Path Removal (P-RM) after Weight Pruning. Apr 2019. [13](#)
- [42] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. [13](#)
- [43] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017. [13](#)

- [44] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. 2016. URL <http://arxiv.org/abs/1611.06440>. 14
- [45] Taiji Suzuki, Hiroshi Abe, Tomoya Murata, Shingo Horiuchi, Kotaro Ito, Tokuma Wachi, So Hirai, Masatoshi Yukishima, and Tomoaki Nishimura. Spectral-pruning: Compressing deep neural network via spectral analysis, 2018. 14
- [46] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=rJl-b3RcF7>. 14
- [47] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017. 15
- [48] Joel Zylberberg. Untuned but not irrelevant: The role of untuned neurons in sensory information coding. 2017. doi: 10.1101/134379.
- [49] Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks. *ArXiv e-prints*, 2017.
- [50] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. 15
- [51] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. <https://distill.pub/2017/feature-visualization>.
- [52] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. 2015. 15
- [53] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *TPAMI*, 41(9):2131–2145, 2019. 15
- [54] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE CVPR*, Jun 2014. doi: 10.1109/cvpr.2014.81. 15
- [55] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*, 2016. 15
- [56] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. 15

-
- [57] A. Karpathy. t-sne visualization of cnn codes. 2014. <https://cs.stanford.edu/people/karpathy/cnnembed/>. 15
- [58] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. 15
- [59] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011. 27, 28, 32, 33, 35
- [60] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 27, 29, 32, 33, 35
- [61] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 30
- [62] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL <http://arxiv.org/abs/1502.03167>. 30
- [63] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. pages 133–138, 01 1994. 41, 48
- [64] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012. 42
- [65] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541. ACM, 2006. ISBN 1-59593-339-5. doi: 10.1145/1150402.1150464. URL <http://doi.acm.org/10.1145/1150402.1150464>. 56
- [66] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL <http://arxiv.org/abs/1503.02531>. 56

Appendices

Appendix 1

The labels of classes around class 400 are provided in Table 1. According to Table 1, we can see that all classes from 350 till 397 correspond to the animals' names, while classes after 397 belong to the items produced by humans. Hence, these two groups have a low semantic similarity.

TABLE 1: ImageNet Labels for classes ids=[350;450]

Class id	Class name	Class id	Class name
350	ibex	372	baboon
351	hartebeest	373	macaque
352	impala	374	langur
353	gazelle	375	colobus
354	arabian camel	376	proboscis monkey
355	llama	377	marmoset
356	weasel	378	capuchin
357	mink	379	howler monkey
358	polecat	380	titi
359	black-footed ferret	381	spider monkey
360	otter	382	squirrel monkey
361	skunk	383	madagascar cat
362	badger	384	indri
363	armadillo	385	indian elephant
364	three-toed sloth	386	african elephant
365	orangutan	387	lesser panda
366	gorilla	388	giant panda
367	chimpanzee	389	barracouta
368	gibbon	390	eel
369	siamang	391	coho
370	guenon	392	rock beauty
371	patas	393	anemone fish
394	sturgeon	431	bassinet

Class id	Class name	Class id	Class name
395	gar	432	bassoon
396	lionfish	433	bathing cap
397	puffer	434	bath towel
398	abacus	435	bathtub
399	abaya	436	beach wagon
400	academic gown	437	beacon
401	accordion	438	beaker
402	acoustic guitar	439	bearskin
403	aircraft carrier	440	beer bottle
404	airliner	441	beer glass
405	airship	442	bell cote
406	altar	443	bib
407	ambulance	444	bicycle-built-for-two
408	amphibian	445	bikini
409	analog clock	446	binder
410	apiary	447	binoculars
411	apron	448	birdhouse
412	ashcan	449	boathouse
413	assault rifle	450	bobsled
414	backpack		
415	bakery		
416	balance beam		
417	balloon		
418	ballpoint		
419	band aid		
420	banjo		
421	bannister		
422	barbell		
423	barber chair		
424	barbershop		
425	barn		
426	barometer		
427	barrel		
428	barrow		
429	baseball		
430	basketball		