Approximated Deep Neural Networks

A computationally efficient inference

MS (Research) Thesis

By

Siddharth Gupta

Under the supervision of

Dr. Kapil Ahuja and **Dr. Aruna Tiwari**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

AUGUST 2020

Approximated Deep Neural Networks

A computationally efficient inference

A THESIS

Submitted in fulfillment of the

requirements for the award of the degree

of

Master of Science (Research)

by

Siddharth Gupta

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

AUGUST 2020

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Approximated Deep Neural Networks** in the partial fulfillment of the requirements for the award of the degree of **Master of Science** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from July 2018 to August 2020 under the joint supervision of Dr. Kapil Ahuja, Associate Professor, Indian Institute of Technology Indore, Indore, India and Dr. Aruna Tiwari, Associate Professor, Indian Institute of Technology Indore, Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Siddharth Gupta

31-08-2020

Signature of the Student with Date

(Siddharth Gupta)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

31-08-2020 Signature of Thesis Supervisors with Date 31-08-2020

(Dr. Kapil Ahuja) **(Dr. Aruna Tiwari)**

Siddharth Gupta has successfully given his M.S. Oral Examination held on 04-12-2020

Dec 4, 2020

Signature of Chairperson, OEB

Date:

4-12-2020 4-12-2020

Signatures of Thesis Supervisors

Date:

Signature of Convener, DPGC

Date: 04-12-2020

Somnath Dey

Signature of Head of Discipline

Date: 04/12/2020

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable and enjoyable. At first, I would like to thank my supervisors **Dr. Aruna Tiwari** and **Dr. Kapil Ahuja**, who were a constant source of inspiration during my work. Without their constant guidance and research directions, this research work could not be completed. Their continuous support and encouragement has motivated me to remain streamlined in my research work.

I am thankful to **Dr. Swadesh Kumar Sahoo** and **Dr. Krushna Mavani**, my research progress committee members for taking out some valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages. I am also grateful to HOD of Computer Science for his help and support.

I am especially grateful to **Prof. Dr. Akash Kumar** and **Mr. Salim Ullah** for their insightful support, encouragement and comments, but also for the hard question which incented me to widen my research from various perspectives.

My sincere acknowledgement and respect to **Prof. Neelesh Kumar Jain**, Director, Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I am also grateful to the institute staffs for their unfailing support and assistance.

Finally, I am thankful to all who directly or indirectly contributed, helped and supported me. To sign off, I write a quote by Aristotle:

"Probable impossibilities are to be preferred to improbable possibilities."

Siddharth Gupta

To my family and friends

Abstract

For most of the pattern recognition and machine learning tasks, deep learning is performing well in recent years. Solving these machine learning tasks require large size deep neural networks (DNNs). Many state-of-the-art DNNs consists of millions of training parameters and arithmetic operations. Increase in size of deep neural network models results in high computational and memory space needs. Therefore, realization of these models on low power devices is possible with some approximation while retaining the accuracy of network. Many recent works have proposed different data quantization techniques. Most of these works require retraining of DNNs after quantization for reducing the accuracy loss due to quantization. Other works which do not retrain the network after quantization, suffer loss in accuracy.

This thesis presents a scalable technique for representing trained parameters of deep neural networks in such a way that these trained parameters can be used for computational and memory-efficient inference phase of deep neural networks. This technique consists of two variations i.e. *log_2_lead*, and *ALigN*. *log_2_lead* provides a single template for parameter representation and *ALigN* adaptively adjusts the template according to different layers of DNN for producing even better results. We have taken three different DNNs, AlexNet, VGG-16, and Resnet-18, and we quantize them using 8-bit version of our schemes and found minimal loss in accuracy compared to full precision network. For evaluating the efficiency of our technique, we have applied our proposed techniques for image classification and segmentation tasks. We have also presented a multiplier design for efficient multiplication of values represented in our proposed templates.

Keywords: Machine learning, deep neural networks, quantization, multipliers, classification, segmentation

List of Publications

A. In Refereed Journal

1. **Siddharth Gupta**, Salim Ullah, Kapil Ahuja, Aruna Tiwari, and Akash Kumar. *ALigN: A Highly Accurate Adaptive Layerwise Log_2_Lead Quantization of Pre-trained Neural Networks*, IEEE Access, vol. 8, pp. 118899 - 118911, 2020. (**IF: 4.098**)

B. In Refereed Conference

1. Salim Ullah, **Siddharth Gupta**, Kapil Ahuja, Aruna Tiwari, and Akash Kumar. *L2L: A Highly Accurate Log_2_Lead Quantization of Pre-trained Neural Networks*, Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, Grenoble (France), pp. 979 – 982, 2020.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning is the fast-growing field of machine learning which is widely used for state-of-the-art machine learning tasks like image classification, speech recognition, visual object recognition, object detection, and many others [12]. Deep Neural Networks (DNNs) are the models of deep learning composed of multiple processing layers to learn the data patterns in different layers of neural network. Large size DNNs having millions of computation units and learning parameters are required for complex machine learning tasks. These large size DNNs demand high computation and memory for training and inference phases. Therefore, training of these network models occurs on high-performance architectures like graphics processing units (GPUs). However, the realization of these DNNs on embedded systems and end-user devices is nearly infeasible due to excessive computational needs. There are many applications [13, 14] nowadays, which are using DNNs. A few of them are as follows.

In self-driving cars [13], deep learning based decision-making architectures process streams of observations coming from different on-board sources, such as cameras, radars, and other sensors. These streams are applied to the deep networks for training and inference. Further, image captioning [14] involves different phases, including the encoding of the image. For image encoding, deep neural networks like AlexNet, VGGNet, ResNet, GoogLeNet, and Inception networks are commonly used [3, 5, 11, 15]. These models require lot of memory and computations for encoding. Deep neural networks are also used in the healthcare domain for Cancer diagnostics [16] and tasks

like image recognition and segmentation [17].

In this thesis, we focus on the possible ways of reducing computations and memory needs to realize deep neural network models on low computation devices.

1.1 Background

Large size DNNs are used for solving complex classification task of machine learning. Training and inference are two phases of deep neural networks that use feed-forward and backpropagation. Due to the complex structure, these DNNs result in a higher number of training parameters. Therefore, high-performance parallel architectures, like GPUs are used for training the neural networks. These architectures use single-precision floating-point parameters for high network accuracy. However, dealing with so many network parameters on low computation devices need some approximation measures. A plethora of recent works has proposed different types of low bit-width quantization techniques and hardware accelerators to reduce the memory and computations of trained DNN models. Approximation in deep neural networks is also implemented using sparsity in networks [18], dynamic fixed point and logarithmic quantization schemes [19, 20], sampling-based technique [21] and clustering-based technique [22]. For example, Google cloud tensor processing units (TPUs) use the BFloat16 [23], a 16-bit number format with 7 bits for storing fraction part, for reducing the memory footprint and computation cost. The use of 16-bit format instead of 32-bit single-precision floating-point format allows the system to implement more complex deep learning models with a slight reduction in the precision of numbers. Further, few researchers propose DNN computation with 8 bits or even less number of bits. Some of these proposed methods involve retraining or fine-tuning of DNN for healing the accuracy loss due to low bit-width parameters.

1.2 Motivation

DNNs are resilient to small errors. A slight reduction in the precision of parameters in DNN does not show a significant impact on network accuracy. Leveraging this fact, recent research works have shown that large size deep neural networks can be implemented with low computation and storage needs using different network approximation techniques [4, 24, 25, 26]. However, these approximation techniques require computationally expensive retraining of the network for reducing the error that arises due to approximation. Hence, there is a need to develop approximation techniques without the requirement of retraining. Few researchers have developed the iterative quantization methods for finding reduced bit-width parameters of network [22, 10]. These iterative approaches can be time-consuming.

To overcome the retraining and iterative process, we propose a quantization based technique for pre-trained deep neural networks. Some researchers have proposed the approaches with layerwise analysis of network for compression [22, 27]. We also propose the parameter representation technique which analyses each layer of DNN and reduces the parameter quantization error even further. In deep neural networks, multiplication operation is the most computation hungry operation. Few researchers avoid multiplication operation in DNNs by using bit-shift and addition operations [28, 29, 30]. Our proposed technique also uses bit-shift and addition for efficient inference of DNN. Further, some works have developed the DNN accelerators based on GPU, FPGA, and ASIC hardware for various machine learning applications [31]. Hardware accelerator using our proposed parameter technique is a future direction for our work. Overall, this thesis seeks to address various issues of quantization based approximation for efficient training and inference of DNNs.

1.3 Objectives

In this thesis, we aim to achieve the following objectives:

(i) To develop a quantization based parameter representation technique for pre-

trained deep neural networks that retains the precision of parameters after the quantization.

(ii) To enable the technique scalable in terms of bit-width for different state-of-the-art DNNs.

(iii) To develop a quantization technique that can provide different configurations of quantization for different layers of DNN.

(iv) To develop quantization technique that allows multiplier less arithmetic for DNN inference.

1.4 Thesis Contributions

A brief overview of our research contributions is provided below, and more details are available in the later chapters.

Contribution I:

A lot of works show that large size deep neural networks can be compressed for efficient implementation using quantization based representations. In this thesis, we introduce a novel quantization method of DNN parameters for efficient inference of pre-trained DNNs. This scheme provides a single template to store the parameters with low bit-width while retaining the precision of parameters. Our technique does not require the retraining or fine-tuning of the network for maintaining the network accuracy.

Contribution II:

We further explore the quantization technique for deep networks with even higher accuracy. We observe that different layers of deep neural networks have different distribution of parameters. Therefore, we present the variation of our other contribution to provide different quantization templates for different layers of the network. By introducing this additional degree of freedom, our scheme reduces the quantization induced error even further. Our technique is optimized for DNN parameters, it provides better

accuracy than state-of-the-art quantization techniques for the dynamic range -1 to +1.

1.5 Organization of the Thesis

This thesis is organized into six chapters. A summary of each chapter is provided below:

Chapter 1 (Introduction)

This current chapter describes background knowledge of approximation in deep neural networks, the motivation of our work, and the contributions of this thesis.

Chapter 2 (Literature Survey and Research Methodology)

This chapter provides a detailed literature survey and a summary of various state-of-the-art DNN compression methods.

Chapter 3 (Quantization of Pre-trained Neural Networks)

In this chapter, proposed quantization based techniques are presented for DNNs.

Chapter 4 (Quantitative Analysis of the Proposed Techniques)

In this chapter, we present quantitative analysis of proposed methods using quantization error and decimal accuracy calculation.

Chapter 5 (Hardware Realization of Proposed Techniques)

In this chapter, we describe the proposed processing element and multiplier design for hardware realization of proposed technique.

Chapter 6 (Conclusions and Future Work)

This chapter concludes the contribution of this thesis and the possible future directions of our work.

Chapter 2

Literature Survey and Research Methodology

This chapter provides a detailed literature review for quantization in deep neural networks. We have divided the whole literature review into five sections. Section 2.1 provides preliminaries on deep neural networks, IEEE floating point format and fixed point format, before proceeding to the literature survey in further sections. Section 2.2 discusses current DNN quantization including the different schemes, Section 2.3 provides a brief literature on machine learning specific data formats, and Section 2.4 describes about hardware accelerators used for deep learning. In the Section 2.5, we discuss the quantitative analysis of number representation.

2.1 Preliminaries

This section provides preliminaries on deep neural networks, IEEE floating point format and fixed point format, which helps to understand further sections.

2.1.1 Deep Neural Networks

DNNs use two computation phases, i.e., forward propagation and backpropagation, for performing training and inference. Networks undergo supervised training procedures to find the optimal training parameters. Deep neural networks are the stack of

multiple types of layers of neurons. Type of these layers are as follows:

(i) **Convolutional layer:** Convolutional layers are used for feature extraction from input vectors. These layers consist of a set of convolutional kernels. Each kernel matrix slides over the input image, elementwise multiplication occurs, and the resultant matrix is passed on to the next layer of DNN.

(ii) **Pooling layer:** These layers sub-samples and reduces the size of the feature map. Max pooling and average pooling are frequent pooling layers. The below equation describes the max pooling operation calculating feature value f_{out} using n-sized pooling window.

Output of max pooling operation is:

$$f_{out} = max(f_1, f_2, f_3, ..., f_n) \tag{2.1}$$

(iii) **Fully connected layer:** A fully connected layer is also termed as a dense layer. Each neuron of a fully connected layer is connected to every previous layer neuron. Fully connected and convolutional layers perform a weighted sum of input features and weight parameter *'w'*. After that activation functions φ are applied to the addition of weighted sum and bias *'b'* for introducing the non-linearity. Sigmoid and Relu are the frequently used activation functions for DNNs. Every neuron consists of an activation function and output of that is referred to as activation. Below equation describes the output of the fully connected neuron:

$$y = \varphi\left(\sum_{i=1}^{n} f_i w_i + b\right) \tag{2.2}$$

Figure 2.1 of AlexNet [3] shows the example of typical architecture of deep neural network. 224x224 dimensional image is input to this network. This network consists of convolutional layers, max-pooling layers, and dense layers. The output layer is having 1000 nodes, each node represents one class of output.

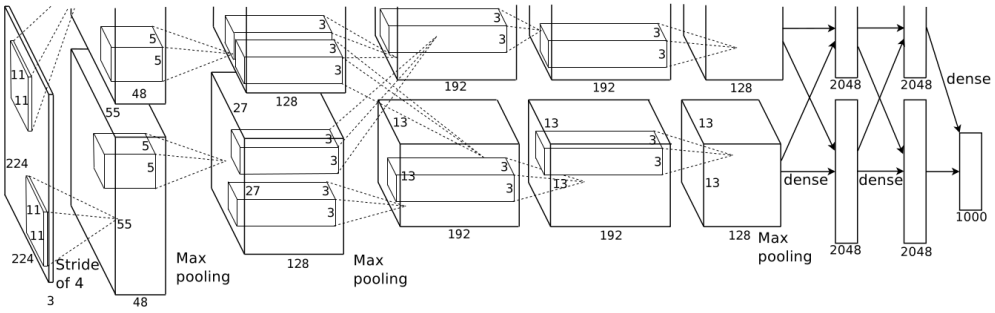


Figure 2.1: Network architecture of Alexnet by Krizhevsky et al. [3]

Since in this thesis, we propose the quantization-based parameter representation techniques for DNNs. Hence, the IEEE real value representation needs to be discussed in detail.

2.1.2 IEEE-754 Floating Point Number Representation [1]

IEEE-754 provides the standards for representing real values in floating-point format. Floating point format consists of sign, exponent, and mantissa bits as shown in figure 2.2. Exponent represents the range of format and mantissa represents the precision of format. In the table 2.1, we show the different variations of floating-point representation.

Figure 2.2: Example of floating point number [4]

Table 2.1: Variations of floating point representation [4]

Format	Total bit-width	Exponent bit-width	Mantissa bit-width
Double precision floating point	64	11	52
Single precision floating point	32	8	23
Half precision floating point	16	5	10

2.1.3 Fixed Point Approximation [2]

Deep neural network inference uses single-precision 32-bit floating-point numbers for parameter representation. This 32-bit number format limits the implementation of DNNs on low computation and memory devices. The fixed-point format offers to represent the parameters in reduced bits with different integer and fractional lengths. Fixed-point format consists of a N-bit signed integer mantissa and a global scaling factor f shared among all fixed-point values. That gives the decimal value of *Integer* x 2^{-f}. In the table 2.2, we show the examples of 8-bit and 10-bit fixed point format.

Table 2.2: Examples of 8-bit and 10-bit fixed point format. *fxp-x.y* shows x bits for integer length and y bits for fractional length.

***N* Bit-width**	**Format**	***Integer***	***f***	**Binary representation**	**Decimal value**
8-bit	*fxp-2.5*	93	5	10.11101	2.90625
	fxp-3.4	93	4	101.1101	5.8125
	fxp-4.3	93	3	1011.101	11.625
10-bit	*fxp-4.5*	463	5	1110.01111	14.46875
	fxp-5.4	463	4	11100.1111	28.9375
	fxp-6.3	463	3	111001.111	57.875

Further, we describe the neural network quantization for approximating the parameters.

2.2 Neural Network Quantization

Over the last few years, researchers have used different approximation techniques [32] for reducing the computations of deep neural networks while maintaining the accuracy of networks, such as dropout, network pruning, and quantization. In this thesis, we are mainly focusing on the quantization technique for DNN approximation that is quite popular. Doing a fixed-point approximation of the network parameters is one direction. Chen et al. [33] have used fixed-point format for representing the

parameters of DNNs with low bit-width. They have significantly reduced the computational efforts and memory costs in DNNs using fixed-point format compared to floating-point arithmetic. Quantization of neural network parameters can be applied during training or after training. In 2014, Courbariaux et al. [4] present the DNN training with low precision fixed-point parameters and shown very little accuracy loss compared to single-precision floating-point format. In this work, they use 20-bit and 12-bit DNN parameters to increase the efficiency of computation hungry multiplication operation in forward propagation and backpropagation. Further, in 2015, few researchers [24, 34] reduce the precision of network parameters even more by using binary and ternary parameters. Courbariaux et al. [25] present the binary quantization of DNN activations along with weights.

In XNOR-Net [26], authors use binary kernel and inputs for efficient convolutional operation in DNNs. Shuchang et al. stochastically quantize parameter gradients to low bit-width numbers during backward pass before propagating them to the convolutional layers in DoReFa-Net [35]. Gysel et al. [27] use dynamic fixed point technique for quantization along with retraining of network. In QUENN technique [22], authors find different bit-width for different layers of DNNs using an iterative approach. Vogel et al. [10] quantize the network by iteratively finding the optimal step size of the quantization of each layer's parameters. Mordido and Keirsbilck [21] use the Monte Carlo Method for sampling the pre-trained parameters of the network.

The most basic DNN approximation is to use fixed point quantized parameters [4, 24, 34]. However, fixed point quantization of DNN parameters suffers the loss of precision in quantized parameters. In our techniques, we retain the precision of parameters after the quantization. Some techniques use retraining of DNN with quantized parameters for reducing the error in accuracy introduced by quantization [4, 24, 34, 25, 26, 35, 27, 36]. However, these techniques cannot be applied to the pre-trained network. Few researchers [22, 10] use iterative approaches for finding the optimal quantization for network parameters. In our technique, we quantize the DNN in single pass and can be applied to pre-trained network. In the techniques [34, 25, 10, 28, 29, 30, 36], bit-shift operation is used in place of computationally

costly multiply-operation.

A survey is presented next for different types of quantization schemes applied in deep neural networks.

2.2.1 Linear Quantization

Linear quantization maps large set of values into lower set of values [37]. In DNNs, parameters are represented in single-precision 32-bit floating-point format. Using linear quantization, parameters can be represented less number of bits. Equations 2.3, 2.4 and 2.5 describes the method to convert float32 data tensor x to N-bit fixed point tensor x_{quant}.

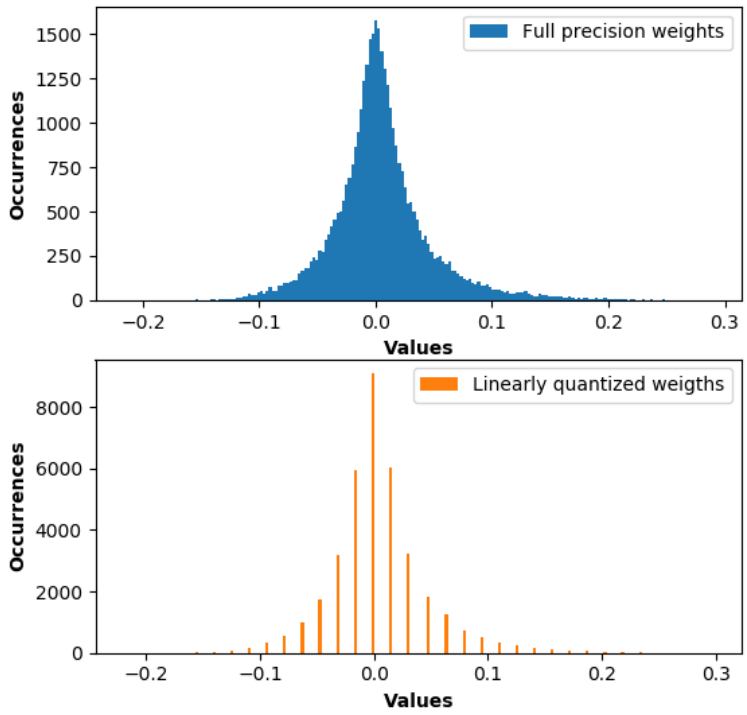


Figure 2.3: 7-bit Linear quantization of VGG-16 [5] Conv1_2 layer weights pre-trained on ImageNet dataset [6]

$$\Delta = clip\left(\frac{max\ (|\ x\ |)}{2^{N-1}}, 2^{N-1}, 2^{-(N-1)}\right) \tag{2.3}$$

$$\Delta = 2\ **\ clip\left(round(log_2(\Delta)), N-1, -N+1\right) \tag{2.4}$$

$$x_{quant} = clip\left(round\left(\frac{x}{\Delta}\right), 2^{N-1} - 1, -2^{N-1}\right)\Delta \tag{2.5}$$

where $round()$ function rounds the value to nearest integer and $clip()$ function is below

$$clip(x, Max, Min) = \begin{cases} x, & Min < x < Max \\ Max, & x \geq Max \\ Min, & \text{otherwise} \end{cases}$$

In Figure 2.3, we consider the weights of a pre-trained DNN convolutional layer of full precision and linearly quantize in 7-bit fixed-point representation. Linear quantization converts the wide set of floating-point values into a discrete set of values. Parameters of deep neural networks are normalized, so most of the weight parameter values are near 0.

2.2.2 Power of 2 Quantization (log_2 Quantization)

Power of 2 quantization is also referred to as logarithmic base 2 quantization [29]. Most of the DNN weights and activation values are non-uniformly distributed. Therefore, logarithmic-scaled quantization is used for DNNs. This quantization also allows to multiply the numbers with bit-shift operations. Equations 2.6 and 2.7 describe the method to convert the full precision data tensor to power of 2 quantized data tensor. Figure 2.4 shows the power of 2 quantization of DNN layer's full precision weights shown in the Figure 2.3. Compared to the linear quantization, power of 2 quantization provides more number representation within the same range.

$$\hat{x_{quant}} = clip\left(round\left(log_2(\mid x \mid)\right), 0, N\right) \tag{2.6}$$

$$\Delta = x_{quant} = sign(x)2^{\hat{x_{quant}}} \tag{2.7}$$

$$sign(x) = \begin{cases} +1, & \text{if x>0} \\ 0, & \text{if x=0} \\ -1, & \text{if x<0} \end{cases}$$

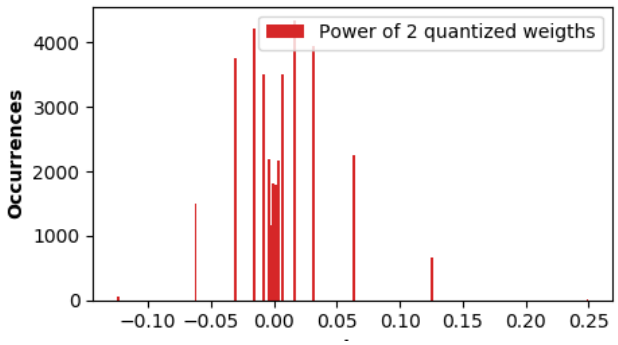


Figure 2.4: Power of 2 quantization of VGG-16 [5] Conv1_2 layer weights pre-trained on ImageNet dataset [6]

2.2.3 Two-Hot Quantization

The quantization technique proposed in [10] divides the Float32 parameter into two segments i.e., MSBs and LSBs. It then performs the log_2 quantization of the MSBs and LSBs separately and then adds the MSBs-quantized and LSBs-quantized values to achieve the final quantized value. This technique can recover some quantization induced errors; however, it also ignores all the significant bits after the leading 1 in both segments (MSBs and LSBs).

2.2.4 Dynamic Fixed Point Quantization

Dynamic fixed point is the variation of fixed point format [22, 27]. In this technique, fractional length (FL) for data representation is chosen based on the range of parameters. For quantizing the layers of DNN, different FLs are used according to the layer's parameter range. Number representation for dynamic fixed point is as follows:

$$Quantized\ value = (-1)^{sign}.2^{-FL}\sum_{i=0}^{W-2}2^{i}.m_i \tag{2.8}$$

where 'W' is bit-width and 'm' mantissa value.

In recent years, researchers have shown interest in providing number formats specific to deep neural networks, which are discussed in the following section.

2.3 DNN Specific Number Formats

In the past years, many application-specific data representation formats are introduced. Google's bfloat16 [7, 23] and minifloat [27] are machine learning specific number formats for speeding up the computation time and reducing the memory requirements.

(i) **Bfloat16, 16-bit brain floating point**: Google's bfloat16 [23] is a 16-bit number representation format. It utilizes reduced precision of the numbers to provide high-performance tensor processing units (TPUs). This format uses 8-bits for exponent and 7-bits for precision as shown in Figure 2.5. Compared to the half-precision floating-point format, bfloat16 uses more number of exponent bits to increase the dynamic range of values represented by the format. Range of bfloat16 is from $\sim 1e^{-38}$ to $\sim 3e^{38}$.

Figure 2.5: Bfloat16 format [7]

(ii) **Minifloat**: Gysel et al. [27] introduced an 8-bit format for representing DNNs weights and activation. Minifloat follows the IEEE-754 standards with some modification. However, 8-bit minifloat format [27] does not consider all the representations for normalized parameters and activations of DNNs.

Later, in the quantitative analysis, we compare our techniques with bfloat16 and minifloat to discuss the number representation capabilities of our techniques independent of DNNs. Since the proposed methods of this thesis can be used for hardware accelerators, we provide a brief discussion of the hardware accelerators used for deep neural networks in the subsequent Section 2.4.

2.4 Hardware Accelerators for DNNs

Over the years, the advancement of multicore processors and accelerators has made it possible to implement computationally heavy machine-learning algorithms such as deep neural networks [31]. GPU (graphics processing units), FPGA (Field programmable gate arrays), and ASIC (application-specific integrated circuit) based processors and accelerators are commonly used for machine learning algorithms.

(i) **GPU-based Accelerators**: As GPU's are high performance and memory architecture, many researchers consider GPU-based inference for deep learning. Denton et al. [38] propose the technique of GPU-based inference by finding the low-rank approximation of the convolutional layer and then fine-tuning. They achieve 2x speedup in the performance while maintaining the accuracy within 1% of the non-approximated model. NVIDIA [39] presents Turing architecture for training and inference for DNN models. Han et al. [40] reduce storage space of AlexNet and VGG-16 model by 35x and 49x respectively, by using network pruning and Huffman coding.

(ii) **FPGA-based Accelerators**: Compared to GPU based accelerators, FPGA accelerators produce less throughput, still, many researchers focus on implementing one or more DNN architectures on FPGA. This is because FPGAs are reconfigurable and programable. Xhang et al. [41] analyze computing and memory requirement of layers of convolutional neural network and accordingly find the optimal solution for design space. Suda et al. [42] implement the VGG network for classification on the OpenCL-based FPGA accelerator.

(iii) **ASIC-based Accelerators**: Despite larger design time, ASIC-based accelerators can achieve high performance and efficiency. DaDianNao [43] uses a custom multichip architecture for machine learning tasks. Chen et al. [44] introduce re-configurable accelerator with higher number of processing elements for Deep convolutional networks.

Next, we discuss the techniques for quantitave analysis of quantized number representations.

2.5 Quantitative Analysis of Number Representations

After the quantization process, quantized values have less precision compared to full precision value, due to the use of low bit-width. Therefore, quantization introduces some error in data while providing storage and processing benefits. For analyzing the efficacy of the quantization technique, many researchers have done quantitative analysis of quantization error.

Training of DNN occurs with full precision parameters. These trained full precision (exact) parameters are quantized and used for efficient inference. Absolute and relative errors provide the error difference between quantized and exact values as described in Equations 2.9 and 2.10, respectively.

$$absolute\ error = |X_{quantized} - X_{exact}| \tag{2.9}$$

$$relative\ error = \left| \frac{X_{quantized} - X_{exact}}{X_{exact}} \right| \tag{2.10}$$

where X is the scalar value.

In Ristretto framework [27], researchers use signal-to-quantization noise ratio (SQNR) for quantifying the quantization error. Further, John et al. [45] have proposed a decimal accuracy metric to compare the effectiveness of number representation as defined by the Equation 2.11.

$$decimal\ accuracy = -log_{10} \left| - log_{10} \left(\frac{X_{quantized}}{X_{exact}} \right) \right| \tag{2.11}$$

There are many measures available for quantitative analysis. Absolute error, relative error, and decimal accuracy are the commonly used measures which we use for our analysis.

Chapter 3

Quantization of Pre-trained Neural Networks

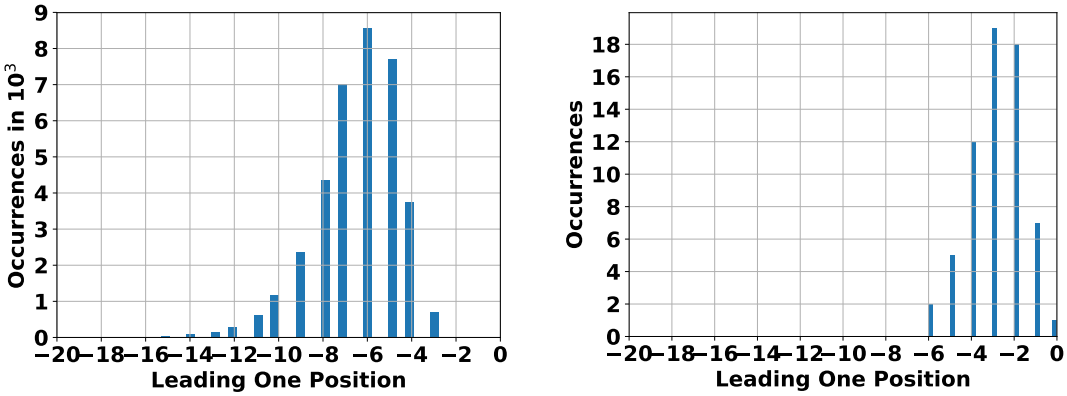
In this chapter, we present deep neural network parameter quantization techniques, which are named as L2L (log_2_lead) and ALigN. This proposed work is inspired by nonlinear quantization of values for reduced computations. As mentioned earlier, traditional quantization techniques require retraining or fine-tuning of DNN with quantized parameters to minimize the quantization induced errors. However, our quantization techniques can be applied directly to pre-trained DNN for inference. The proposed quantization methods also provide a unique representation of parameters to replace the multiplication operation with bit-shift and addition. For experimentation, we take different sizes of state-of-the-art DNNs and quantize them using proposed techniques. Their performance is evaluated for classification and segmentation tasks.

3.1 L2L: Log_2_Lead Quantization of Pretrained Neural Networks

This section describes the proposed quantization technique L2L (Log_2_Lead) for DNN. The proposed fixed-point quantization technique attempts to minimize the quantization induced errors by identifying and storing the most significant 1's in the parameters of a pre-trained DNN.

3.1.1 Template of Quantization

As the trained parameters are represented in single-precision floating-point scheme (32 bits), the identification of the 1's which have higher significance can notably reduce the quantization generated errors. To identify the significant 1's in Float32-based parameters, Figure 3.1 and Figure 3.2 give histograms of the leading 1's in all the weights and biases of two different layers of VGG-16 network. As shown, the leading 1 for most of the trained parameters (weights and biases) occurs at some distinct bit positions; for example, for the weights of Conv1_2 and Conv4_1 layers, the leading 1 frequently occurs at bit position -6 and -8 respectively. However, there are also some weights with very low-values and having the leading 1 occurring at bit position -15.



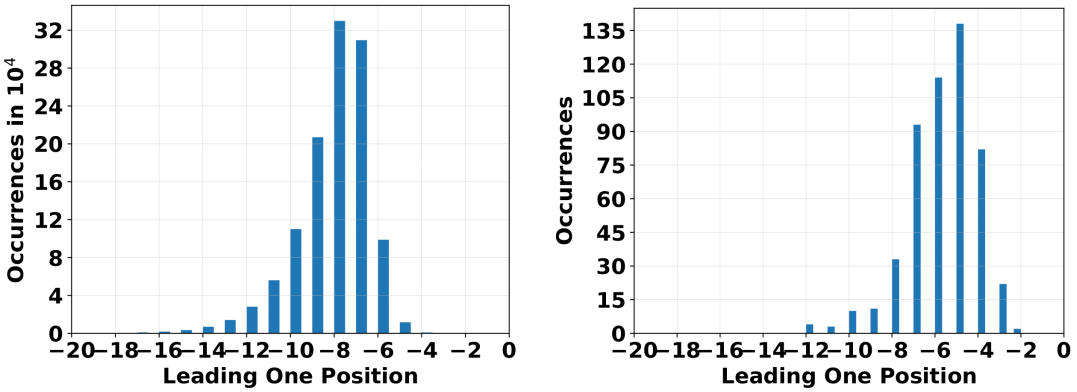
(a) Weights of a single layer

(b) Biases of a single layer

Figure 3.1: Histogram of leading 1's for all weights and biases for pre-trained VGG-16 [5] Conv1_2 layer

In our proposed *log_2_lead* technique, we utilize log_2 to detect the position of the leading 1 in a fraction. However, to limit the quantization errors, *log_2_lead* also observes the following bits locations after the leading 1 location. For N-bit quantization, Figure 3.3 shows the template of our proposed *log_2_lead* quantization scheme. The first bit is reserved for showing the sign of the quantized parameter. Following $\lceil \frac{N-1}{2} \rceil$ bits are reserved for storing the location of the leading 1 in the original non-quantized parameter. The remaining bits are used to store the values of the following $\lfloor \frac{N-1}{2} \rfloor$ bits after the leading 1.

For example, for the leading 1 histograms in Figure 3.1 and $N = 8$, the leading 1



(a) Weights of a single layer

(b) Biases of a single layer

Figure 3.2: Histogram of leading 1's for all weights and biases for pre-trained VGG-16 [5] Conv4_1 layer

at bit position −12 would be represented as a binary number '1100' by the '*leading one location*' field in the template shown in Figure 3.3. Moreover, to increase the precision of the quantized value, we always analyze $\lfloor \frac{N-1}{2} \rfloor + 1$ bits after the leading 1 and round the values (round towards $+\infty$) to $\lfloor \frac{N-1}{2} \rfloor$ bits.

The rounding of $\lfloor \frac{N-1}{2} \rfloor + 1$ bits further helps in retaining the precision of a rounded number. For example, Figure 3.4 shows an example of quantizing a fractional number 0.217884 using the proposed *log_2_lead* technique. The leading 1 is found at bit location −3 which is stored in our template as a binary number 0011. After the leading 1, following four bits are analyzed and rounded to binary pattern 110. The corresponding quantized value is 0.21875.

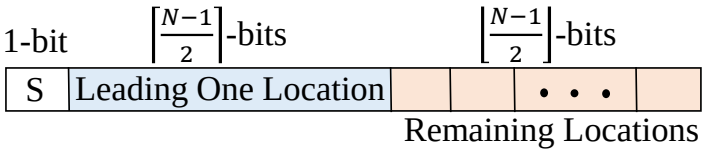


Figure 3.3: Template for proposed quantization technique

3.1.2 L2L Quantization Example

Compared to our proposed technique, a typical log_2 quantization as described in Eq. 2.6 and Eq. 2.7 (Chapter 2) would only find a single leading 1 for each weight and

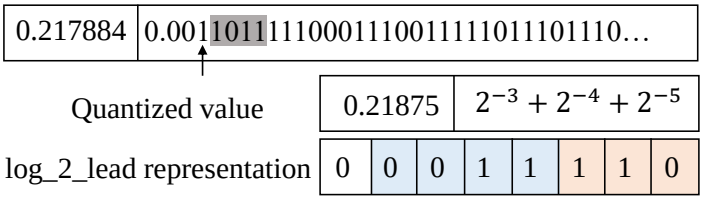
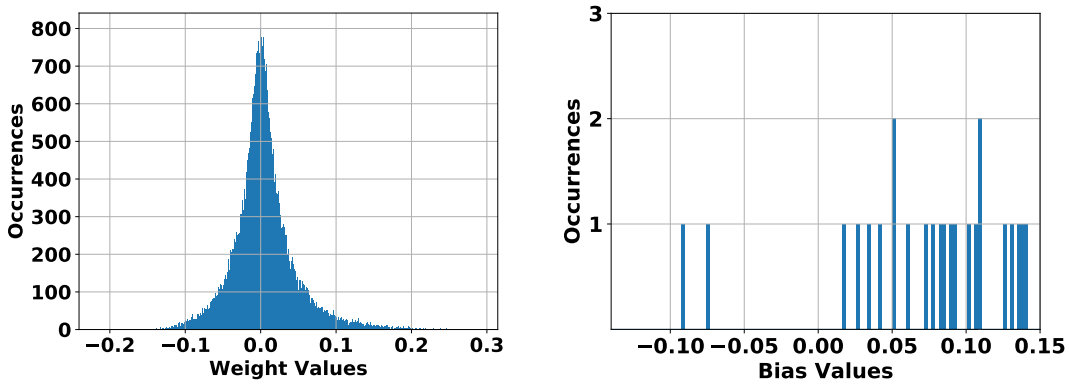


Figure 3.4: *log_2_lead* quantization example



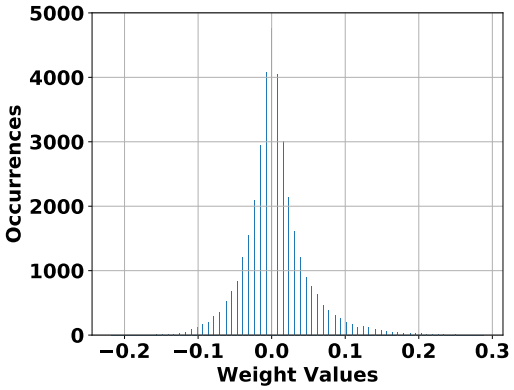
(a) Weights Distribution (b) Biases Distribution

Figure 3.5: Distribution of weights and biases for pre-trained VGG-16 [5] Conv1_2 layer

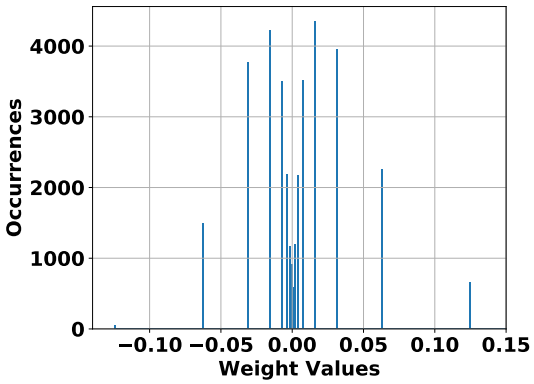
discard all other bit locations. Due to ignoring of all other bit locations for computing the quantized value, the log_2 quantization can introduce substantial quantization errors.

We have computed the classification accuracy of pre-trained ResNet-18 and VGG-16 DNNs and found that quantization errors can be healed significantly in the final output of a DNN by utilizing 8 bits for L2L quantization (discussed in Section 4.2 of Chapter 4).

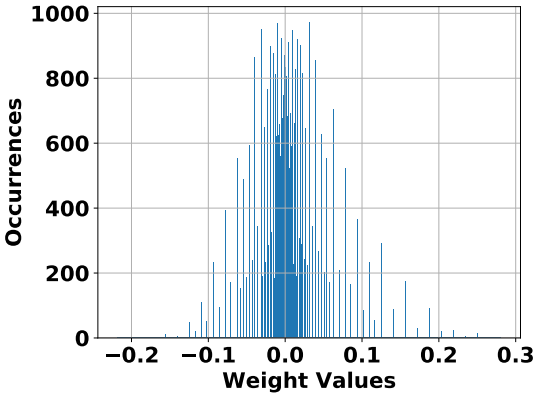
Figure 3.6 and Figure 3.7 show the quantization of weights and biases of two different layers of VGG-16 given in Figure 3.5 when using linear, power of 2, and L2L quantizations. It can be observed that the proposed L2L technique provides more discrete fixed-point values and better coverage for quantizing Float32-based parameters than the linear and power of 2 quantization schemes.



(a) Linear quantization

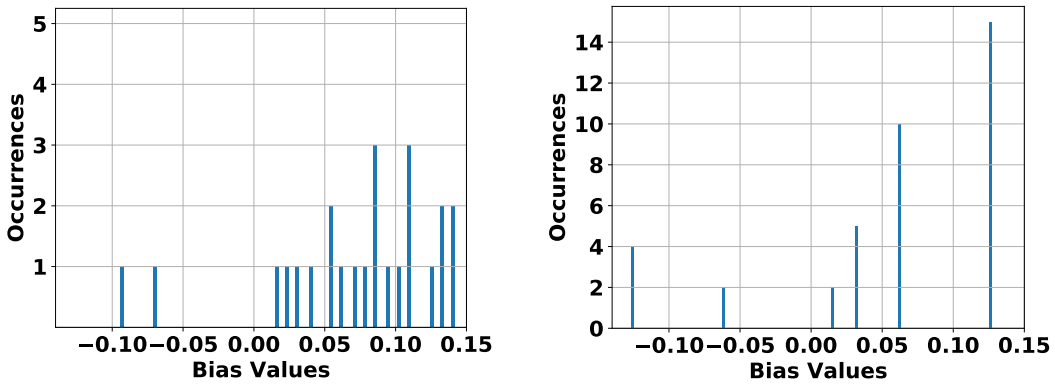


(b) Power of 2 quantization



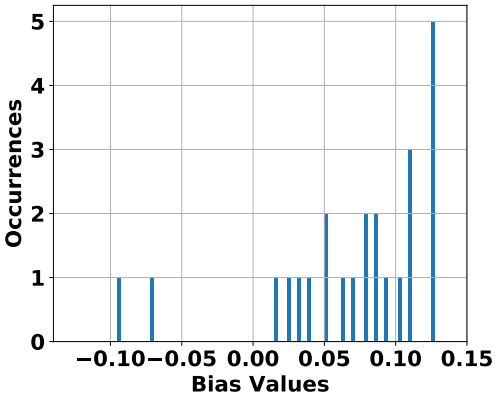
(c) *log_2_lead* quantization

Figure 3.6: Quantization of pre-trained weights of Conv1_2 layer of VGG-16 [5]



(a) Linear quantization

(b) Power of 2 quantization



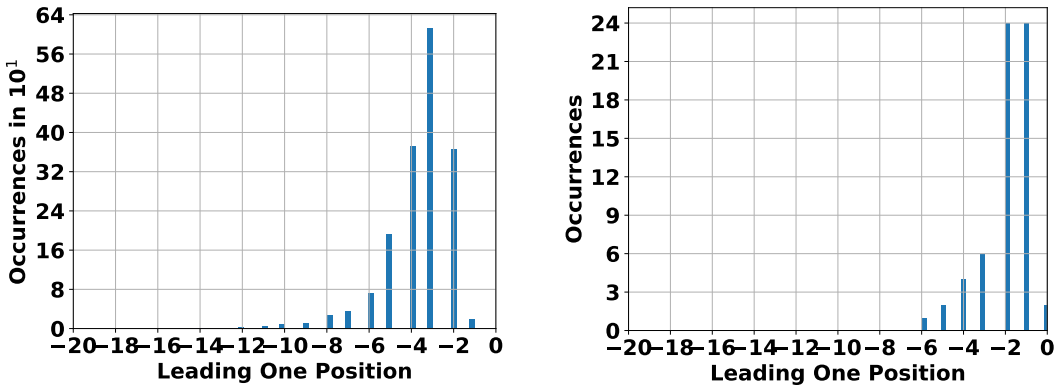
(c) *log_2_lead* quantization

Figure 3.7: Quantization of pre-trained biases of Conv1_2 layer of VGG-16 [5]

3.2 ALigN: Adaptive Layerwise Quantization of Pre-trained Neural Networks

In this section, we describe the adaptive layerwise variation of our L2L technique, i.e., ALigN. ALigN scheme provides multiple templates for DNN parameter representation and reduces the quantization induced error even further.

3.2.1 Template of Quantization



(a) Weights of a single layer (b) Biases of a single layer

Figure 3.8: Histogram of leading 1's for all weights and biases for pre-trained VGG-16 [5] Conv1_1 layer

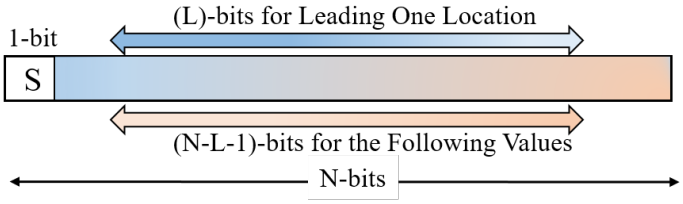


Figure 3.9: *ALigN* template for N-bit quantization

The quantization induced errors can be minimized by utilizing the available quantization bit-width intelligently. The proposed L2L technique, although minimizes the quantization induced errors, has a fixed template for representing the quantized parameters of all layers in a pre-trained DNN. For example, as shown in Figure 3.3,

it always utilizes $\lceil \frac{N-1}{2} \rceil$-bits for storing the position of leading 1 in an N-bit quantization. We analyze the distribution of parameters for various layers of pre-trained DNNs. We observe that for some layers, the fixed $\lceil \frac{N-1}{2} \rceil$-bits for storing the leading 1 position are not used efficiently. For example, Figure 3.8 presents the leading 1 histogram of the weights and biases of the Conv1_1 layer of a pre-trained VGG-16. It can be observed that the leading 1 for these parameters mostly occurs at bit position -3. Bit position -3 can be stored in 2 bits (magnitude of bit position is stored) while log_2_lead uses a fixed template of 4 bits for storing leading 1 position (when taking N= 8). To resolve this problem, we propose '*ALigN*', an adaptive log_2_lead quantization technique. ALigN technique can provide multiple configurations of leading one location storage and following bits storage, as described in Figure 3.9. Further, we describe the layerwise analysis for choosing appropriate ALigN configurations.

3.2.2 Adaptive Analysis of Neural Network

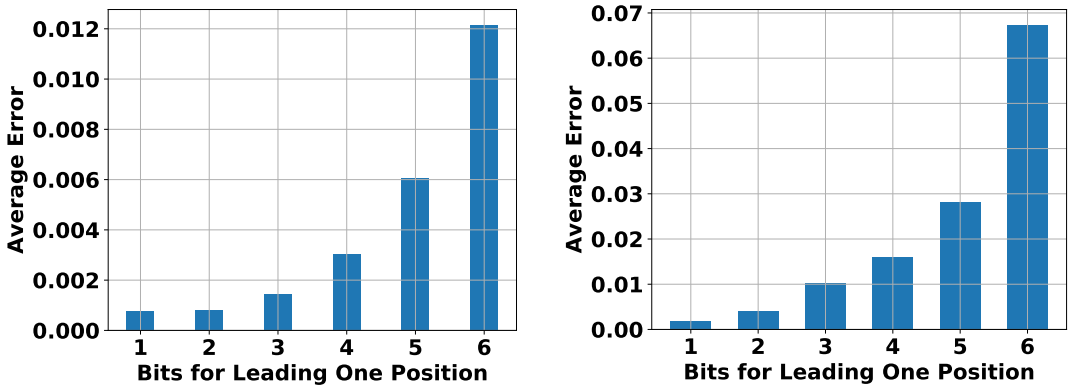
In ALigN technique, we analyze the parameters of each layer to identify a suitable number of bits for storing the leading 1 in the original non-quantized parameters of each layer to reduce the corresponding quantization induced errors. Figure 3.10 and Figure 3.11 show the average quantization induced errors by varying the number of bits reserved for storing the leading 1 location in L2L scheme for the parameters of two different layers of a pre-trained VGG-16 network. For the weights of the Conv1_1 layer, assigning a single bit for storing the leading 1 location produces minimum quantization errors. However, the quantization of the parameters of the Conv1_2 layer requires 3-bits for generating the smallest error. Therefore, it is advantageous to align the L2L scheme according to the distribution of the parameters of each layer of a pre-trained network. Eq. 3.1 and Eq. 3.2 show our technique for finding the optimal number of bits L_w and L_b for storing the leading 1 locations of weights and biases, respectively, for each layer. After determining L_w and L_b adaptively for each layer, we quantize the parameters of each layer by storing leading 1 in L-bits and utilizing the remaining locations for storing $N-L-1$ following bits. Figure 3.9 shows the updated template of our proposed quantization scheme, *ALigN*, to represent the quantized

values. As shown, for an N-bit $ALigN$ quantization, there is no fixed boundary for storing the leading 1 location.

$$L_w = argmin_{L_w}(Average\{|w_{quant} - w|\}) \tag{3.1}$$

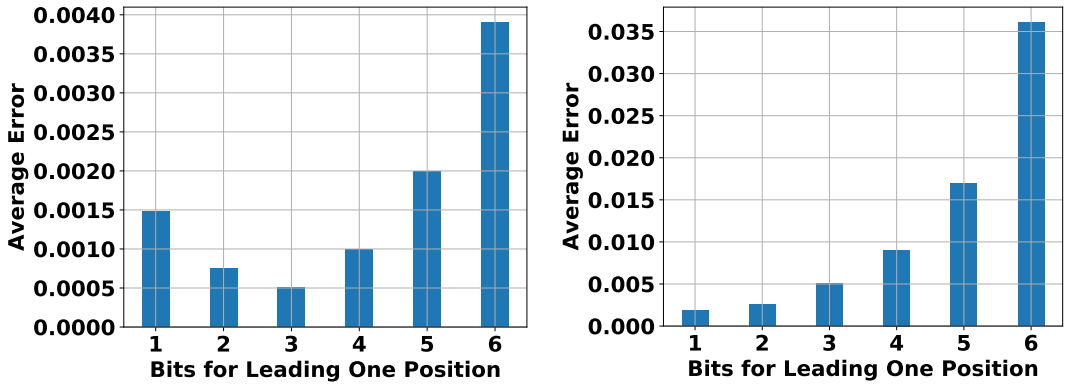
$$L_b = argmin_{L_b}(Average\{|b_{quant} - b|\}) \tag{3.2}$$

where $argmin_L$ selects the L (number of bits for storing leading 1 location) which gives minimum average difference.



(a) Weights of a single layer (b) Biases of a single layer

Figure 3.10: Average error for different number of bits for leading one position for weights and biases for pre-trained VGG-16 [5] Conv1_1 layer



(a) Weights of a single layer (b) Biases of a single layer

Figure 3.11: Average error for different number of bits for leading one position all weights and biases for pre-trained VGG-16 [5] Conv1_2 layer

3.3 Experiments

For the application-level evaluation of linear, power of 2, dynamic fixed point, L2L, and ALigN quantization, we have used TensorFlow framework [46]. Using the framework, we have implemented a variety of networks with different datasets to test the efficacy of our proposed quantization techniques for image classification and semantic segmentation accuracies of quantized networks. These results are discussed in Sections 3.3.1 and 3.3.2. In the experiments below, we show the superiority of our ALigN scheme. All the experiments below have been done with 8-bit ALigN technique. This is because once we use more bits, for example, 10-bits, we do not see a definite effect on classification accuracy. This aspect has been discussed in more detail in the next chapter.

3.3.1 Image Classification

For image classification task, we have taken small and large size deep neural network for variety of datasets. For the MNIST dataset [47], we have trained a lightweight neural network consisting of convolution layers (Conv), Maxpool layer, and fully connected layers (FC), as described in Table 3.1. The MNIST dataset contains 60,000 and 10,000 greyscale training and testing images, respectively, at a resolution of 28 x 28 pixels. After training the network on 60,000 images, we have applied different 8-bit (N=8) quantization schemes on the trained parameters. The application-level accuracy results are reported in Table 3.2. The linear and L2L quantization have nearly similar output accuracy compared to the single-precision Float32 representation. The power of 2 quantization considers only the most significant 1 in the fraction; therefore, it has produced comparatively reduced output accuracy. ALigN quantization technique offers the same classification accuracy as produced by the baseline Float32-based implementation.

We also test the lightweight neural network described in Table 3.1 on CIFAR-10 dataset [48] to assess the accuracy of our proposed quantization schemes. CIFAR-10 dataset contains 50,000 training and 10,000 32x32 RGB images. These images are

Table 3.1: Description of networks used for classification of MNIST and CIFAR-10 datasets

Layer	**MNIST Network**		**CIFAR-10 Network**	
	Kernel Size	Total Kernels	Kernel Size	Total Kernels
Conv1	5x5	32	5x5	64
Maxpool	-	-	3x3	-
Conv2	5x5	64	5x5	64
Maxpool	2x2	-	3x3	-
FC1	-	512	-	384
FC2	-	512	-	192
Softmax	-	10	-	10

Table 3.2: Classification accuracy of lightweight neural networks on MNIST and CIFAR-10 dataset with 8-bit quantized weights and biases for different quantization schemes

Quantization	**MNIST**	**CIFAR-10**
w, b quantized	Accuracy [%]	Accuracy [%]
Float32	98.18	85.1
8-bit linear	98.13	85.1
Power of 2	97.6	73.5
L2L	98.12	85.1
ALigN	98.18	85.1
Float32 vs L2L	-0.06	0.0
Float32 vs ALigN	0.0	0.0

classified into 10 labeled classes. Lightweight neural network classification accuracy results for 10,000 images using different 8-bit quanitzation schemes are reported in Table 3.2 (see last column). The power of 2 quantization has shown significant accuracy drop due to limited coverage of the dynamic range of fractional numbers under consideration. The L2L and ALigN quantization have produced comparable results to the single-precision Float32-based results.

To evaluate the efficacy of our proposed techniques on a large dataset, we also test a variety of deep neural networks (see below) on ImageNet dataset [6]. The ImageNet dataset hosts 1.2 million and $50,000$ RGB images for training and validation, respectively. These images are classified into 1000 different categories. For our experimentation, we take pre-trained AlexNet [3], VGG-16 [5] and ResNet-18 net-

works [11]. For effective comparison, we improve the inference accuracy of power of 2 quantization scheme by utilizing the post-quantization retraining, which minimizes its quantization induced errors. AlexNet network contains 5 convolutional layers followed by 3 fully connected layers. For the ImageNet validation set, the pre-trained AlexNet has 53.89% and 77.62% Top-1 and Top-5 accuracies, respectively, using the baseline single-precision Float32 weights, biases, and activations. We quantize the weights and biases of this network with linear, power of 2, dynamic fixed-point, L2L, and ALigN quantization schemes. Except for the dynamic fixed-point scheme, all quantization schemes have an 8-bit width. The dynamic fixed-point quantization employs different bit-widths for different layers. For example, for weights and biases of the conv1, conv2, conv3, and conv4 layers, it applies twelve bits, whereas, for the parameters of the conv5 layer, it utilizes ten bits. Similarly, for the parameters of FC6, FC7, and FC8 layers, it uses sixteen, twelve, and six bits, respectively. The Top-5 and Top-1 classification accuracies are shown in Table 3.3. The L2L quantization scheme provides better output accuracies than the linear, power of 2, and dynamic fixed-point quantization schemes. When compared with Float32, the drop in Top-5 and Top-1 percentage classification accuracies are only 0.12 and 0.01 respectively for the L2L scheme. Based upon the adaptive layerwise analysis of the network, Table 3.4 shows the number of bits (L_x) delimited to store the leading 1 location for parameters of each layer for an 8-bit ALigN quantization scheme. The remaining $8 - L_x - 1$ bits are assigned for storing the bit values following the leading 1. Compared to the baseline Float32 accuracy, the ALigN scheme produces the same output accuracy.

The VGG-16 network mainly consists of 13 convolution layers with kernel size 3×3 and 3 fully connected layers. For the $50,000$ validation images, the pre-trained network has 85.74% and 64.72% Top-5 and Top-1 classification accuracies using the single-precision Float32-based parameters and activations. We apply various 8-bit quantization schemes on the pre-trained parameters and evaluate for classification accuracy. These results are presented in Table 3.5. In this experiment with VGG-16 network, the activations have Float32 precision, and the weights and biases are quantized to 8-bit precision. For the dynamic fixed-point quantization scheme, except

Table 3.3: Classification accuracy of AlexNet network [3] on ImageNet dataset [6] with quantization of weights and biases using different schemes

AlexNet [3]	**Top-5 [%]**	**Top-1 [%]**
Float32	77.62	53.89
8-bit linear	76.95	53.11
Power of 2 (without retraining)	65.45	40.72
Power of 2 (with retraining)	75.63	47.19
Dynamic fixed point	74.21	50.24
L2L	77.50	53.88
ALigN	77.67	53.99
Float32 vs L2L	-0.12	-0.01
Float32 vs ALigN	+0.05	+0.1

Table 3.4: No. of bits selected for storing the leading 1 location of parameters of AlexNet [3] and VGG-16 [5] networks using 8-bit *ALigN* technique

AlexNet	**Layer**	Conv1		Conv2		Conv3		Conv4	
	Bits for Leading	2		3		3		3	
VGG-16	**Layer**	Conv1_1	Conv1_2	Conv2_1	Conv2_2	Conv3_1	Conv3_2	Conv3_3	Conv4_1
	Bits for Leading	1	2	3	3	3	3	3	3

AlexNet	**Layer**	Conv5		FC6		FC7		FC8	
	Bits for Leading	3		3		3		3	
VGG-16	**Layer**	Conv4_2	Conv4_3	Conv5_1	Conv5_2	Conv5_3	FC6	FC7	FC8
	Bits for Leading	3	3	3	3	3	3	3	3

for the FC6 layer, which is quantized to 8 bits, all other layers have Float32 precision. Compared to the linear quantization, power of 2 quantization, and dynamic fixed-point quantization, our proposed techniques L2L and ALigN produce better Top-5 and Top-1 classification accuracies. The drop in Top-5 and Top-1 percentage classification accuracies for L2L scheme is only by 0.1 and 0.21, respectively. The ALigN scheme also produces better results than others and reduces the Top-5 and Top-1 percentage classification accuracies (drop by 0.03 and 0.1 only, respectively). The corresponding number of bits, to store the leading 1 location of the trained parameters of each layer, are again shown in Table 3.4.

We also evaluate our proposed quantization schemes on the residual network ResNet-18 with the ImageNet validation set. Table 3.6 shows the corresponding Top-5 and Top-1 accuracies of the quantized pre-trained network with 8-bit linear, power of 2, L2L, and ALigN schemes. Our proposed quantization schemes again outperform other quantization techniques in the respective classification accuracies. For example,

compared to the baseline Float32-based classification accuracies, the L2L has only 0.38 and 0.8 drops in the Top-5 and Top-1 percentage classification accuracies, respectively. The ALigN scheme improves the Top-1 percentage classification accuracy by 0.12 when compared with the corresponding baseline accuracy. However, the Top-5 percentage accuracy is dropped by 0.1, which is still better than those achieved with the other quantization schemes.

3.3.2 Image Segmentation

To further evaluate the efficacy of our proposed quantization techniques, we also consider the semantic segmentation task. For this purpose, we utilize the Fully Convolutional Network (FCN) [8] and PASCAL Visual Object Classes (VOC) 2012 validation set [9]. The FCN utilizes the thirteen convolutional layers of VGG-16 [5] for performing semantic segmentation. The PASCAL VOC 2012 validation dataset contains 1449 images in 20 different classes. Further, we use the intersection over union (IU) metric for measuring the efficiency of presented quantization techniques for the semantic segmentation task. Table 3.7 shows the results of applying 8-bit linear, power of 2, dynamic fixed point, L2L, and ALigN quantization schemes on the pre-trained FCN. For the dynamic fixed-point quantization scheme, except for the FC6 layer, which is linearly quantized to 8 bits, all other layers have Float32 precision. As shown by the results, the pre-trained FCN has a mean IU of 61.8% using single-precision Float32-

Table 3.5: Classification accuracy of VGG-16 network [5] on ImageNet dataset [6] with quantization of weights and biases using different schemes

VGG-16 [5]	**Top-5 [%]**	**Top-1 [%]**
Float32	85.74	64.72
8-bit linear	82.55	59.8
Power of 2 (without retraining)	0.63	0.1
Power of 2 (with retraining)	56.25	30.78
Dynamic fixed point	83.63	61.43
L2L	85.64	64.51
ALigN	85.71	64.62
Float32 vs L2L	-0.1	-0.21
Float32 vs ALigN	-0.03	-0.1

Table 3.6: Classification accuracy of ResNet-18 network [11] on ImageNet dataset [6] with quantization of weights and biases using different schemes

Resnet-18 [11]	**Top-5 [%]**	**Top-1 [%]**
Float32	88.94	69.30
8-bit linear	88.47	68.70
Power of 2 (without retraining)	5.37	1.49
Power of 2 (with retraining)	86.84	65.45
L2L	88.56	68.41
ALigN	88.84	69.42
Float32 vs L2L	-0.38	-0.8
Float32 vs ALigN	-0.1	+0.12

based parameters. Compared to this baseline result, the 8-bit linear quantization displays a loss of 1% in the mean IU, whereas The dynamic fixed-point quantization schemes produce no loss in the mean IU. L2L scheme has only a 0.5% loss in the mean IU, while ALigN quantization schemes produce no loss. Figure 3.12 shows the visual results of applying different quantization schemes on the pre-trained FCN for one single image from the PASCAL VOC 2012 validation dataset. As shown by the images, the ALigN produces minimal differences in the pixel values when compared with the Float32-based representation.

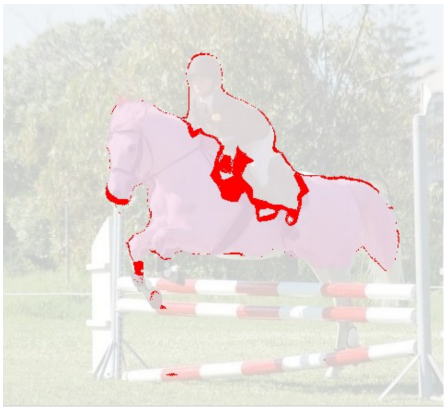
Table 3.7: Mean IU of FCN8 [8] network on PASCAL VOC 2012 validation set [9] with quantization of weights and biases using different schemes

FCN8 [8]	**Mean IU [%]**
Float32	61.8
8-bit linear	60.8
Power of 2 (without retraining)	3.5
Power of 2 (with retraining)	21.1
Dynamic fixed point	61.8
L2L	61.3
ALigN	61.8
Float32 vs L2L	-0.5
Float32 vs ALigN	0.0

(a) Original Image

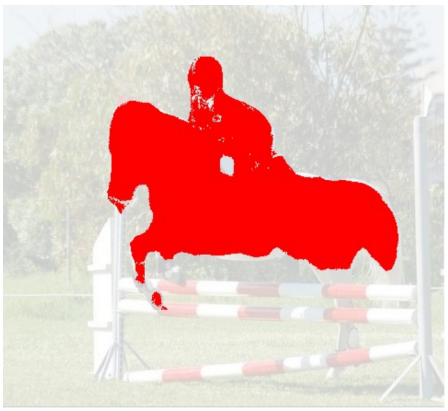
(b) Single precision float32

(c) 8-bit linear quantization



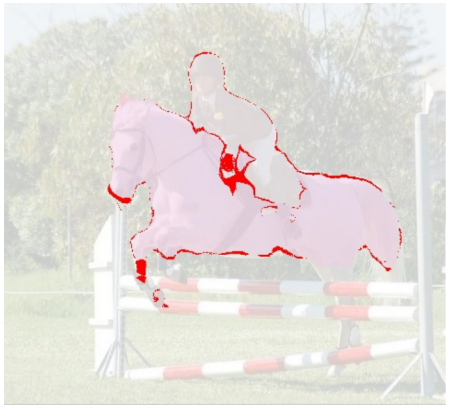
(d) Pixel difference between single-precision Float32 and 8-bit linear quantization

(e) Power of 2 quantization



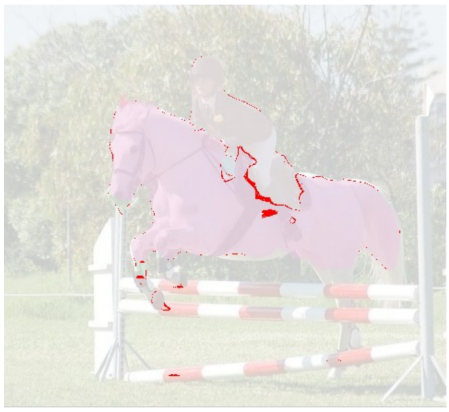
(f) Pixel difference between single-precision float32 and Power of 2 quantization

(g) Dynamic fixed point quantization



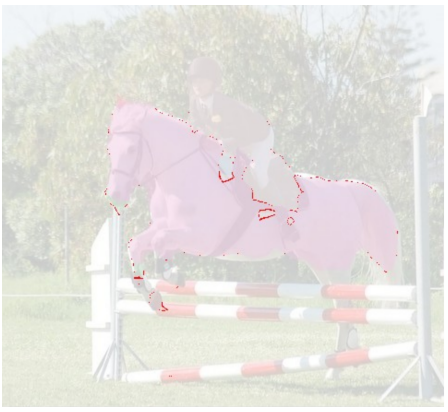
(h) Pixel difference between single-precision float32 and Dynamic fixed point quantization

(i) *log_2_lead* quantization



(j) Pixel difference between single-precision float32 and *log_2_lead* quantization

(k) *ALigN* quantization



(l) Pixel difference between single-precision float32 and *ALigN*

Figure 3.12: Object detection comparison with different quantization schemes using FCN8 [8] network on PASCAL VOC 2012 validation set [9]

3.4 Summary

In this chapter we have proposed two quantization based techniques for parameter representation of pre-trained deep neural networks. First technique (L2L) provides a fixed template for quantizing the DNN parameters by storing the most significant bits of values after the leading 1 position, and gives less quantization errors than existing techniques. Second technique (ALigN) adaptively analyses each layer of the neural network and provides efficient configuration of storing substantial bits. ALigN technique reduces the quantization error even further. We quantize small and large neural networks and perform classification tasks using different datasets to evaluate the proposed methods. Under the large DNNs category, state-of-the-art networks of AlexNet, VGG-16, and Resnet-18 are tested on the ImageNet dataset for inference.

Further, we have also performed the segmentation task using the FCN8 network for PASCAL VOC dataset. For both the classification and segmentation tasks, the accuracy and mean IU of DNNs quantized with 8-bit proposed quantization techniques is only slightly less than a full precision Float32 based network. In this way, we achieve a considerable reduction in the computational and memory requirements of DNN inference at the cost of a slight drop in accuracy. In the next Chapter 4, we discuss the quantitative analysis of the proposed techniques to evaluate their efficacy.

Chapter 4

Quantitative Analysis of the Proposed Techniques

Until now, we have tested our techniques on a variety of deep neural network models using different datasets. Here, we perform error analysis in a different way. First, In Section 4.1, without considering the classification and segmentation tasks, we check the average error in different layers on two types of pre-trained networks AlexNet and VGG-16. For this, we use the error terms defined in Chapter 2. Further, in Section 4.2, we discuss the decimal accuracy measures for 8-bit and 16-bit ALigN technique. For this, we generate equally spaced points in the range of -1 to +1 and quantize these points using different configurations of our ALigN technique.

4.1 Quantization Induced Errors

In this section, we discuss quantization error in the deep neural network layers and individual parameters. Section 4.1.1 describes the average error induced in the DNN layers and section 4.1.2 presents the relative error in DNN parameters.

4.1.1 Average Error in DNN Layers

We quantize different layers of pre-trained VGG-16 and AlexNet networks, using L2L, ALigN, and various other quantization schemes to evaluate the average quanti-

zation error in these layers. We compute the average error of each layer using equation 4.1. The original weights, in single-precision Float32, are quantized to 8-bits. Figures 4.1 and 4.2 show the average errors in different layers of AlexNet and VGG-16 networks, respectively. The traditional power of 2 quantization suffers the most from quantization produced errors. It is found that, for both networks, our proposed techniques always bear a minimal amount of quantization error in each layer.

$$Average\ error = \frac{1}{n_j} \sum_{i=1}^{n_j} \left| X_{iquant} - X_{ifloat} \right| \tag{4.1}$$

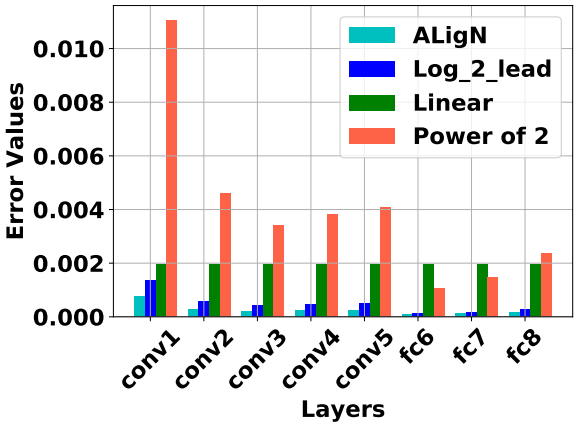


Figure 4.1: Comparison of average quantization induced errors for quantized weights of convolution layers of AlexNet [3] between *ALigN*, *log_2_lead*, linear and power of 2 quantization schemes.

Further, ALigN always generates lesser errors than the L2L quantization scheme as it efficiently utilizes the available quantization bit-width. In Figure 4.2, it can be observed that linear quantization performs better than the L2L quantization for layer conv1_1. In the L2L technique, the allocation of fixed four bits for storing the leading 1 location of the conv1_1 layer has resulted in under-utilization of these bits. This layer parameters do not require more bits for storing the leading 1 location. However, ALigN adaptively selects the number of bits for storing the position of the leading 1 for each layer, and it always performs better than the linear quantization.

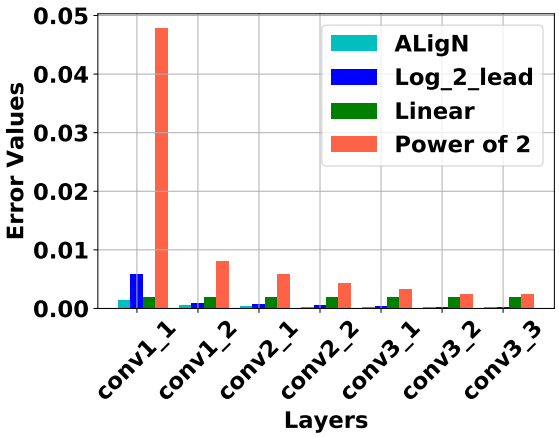


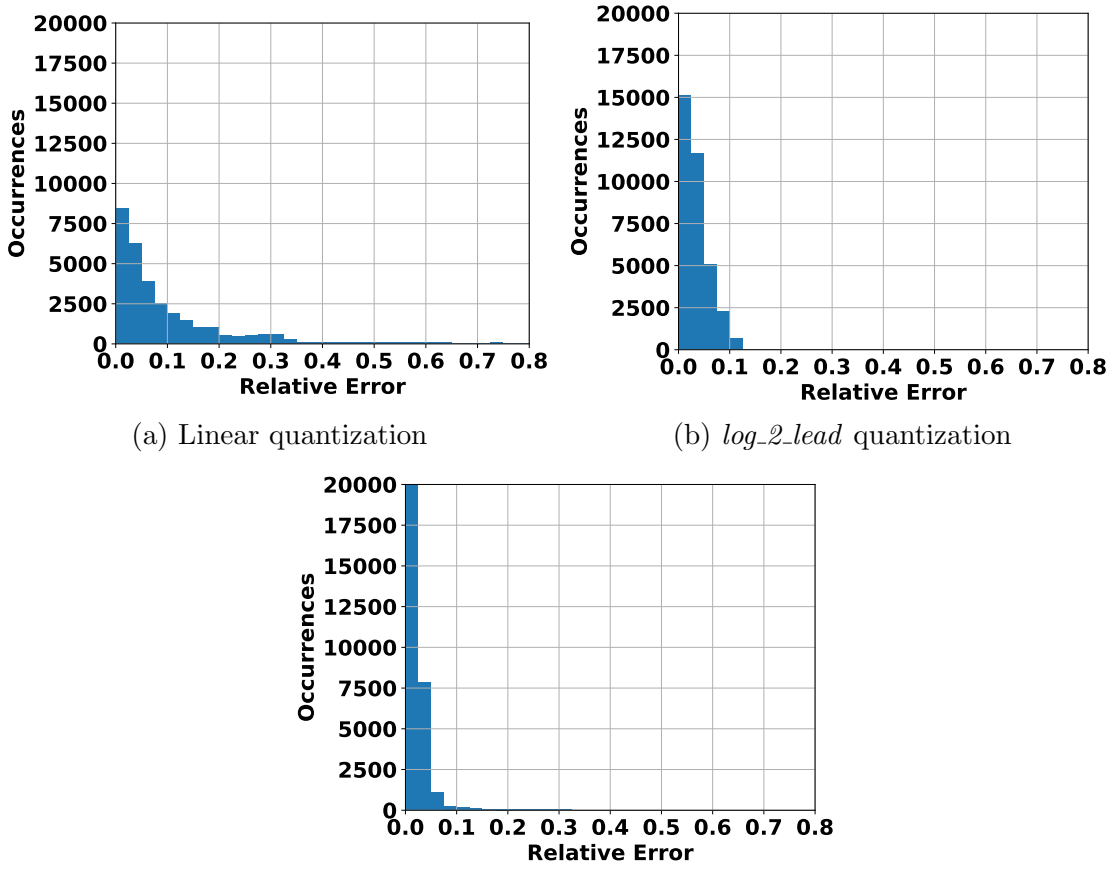
Figure 4.2: Comparison of average quantization induced errors for quantized weights of convolution layers of VGG-16 [5] between *ALigN*, *log_2_lead*, linear and power of 2 quantization schemes

4.1.2 Relative Error in DNN Parameters

To further investigate the distribution of quantization induced errors, we compare the relative error distribution of quantized weights for two different layers of VGG-16 and AlexNet networks using different quantization schemes as shown in Figure 4.3 and 4.4. Quantized relative error is calculated using equation 2.10, which is again summarized below [45]

$$relative\ error = \left| \frac{X_{quant} - X_{float}}{X_{float}} \right| \tag{4.2}$$

It is observed that for both the networks, most of the quantization generated errors for L2L are limited to a narrow band of values (0 — 0.1), whereas the relative errors generated by linear quantization are spread over a broader spectrum (0 — 0.8). The ALigN further reduces these errors and limits them to even a narrower band than L2L.

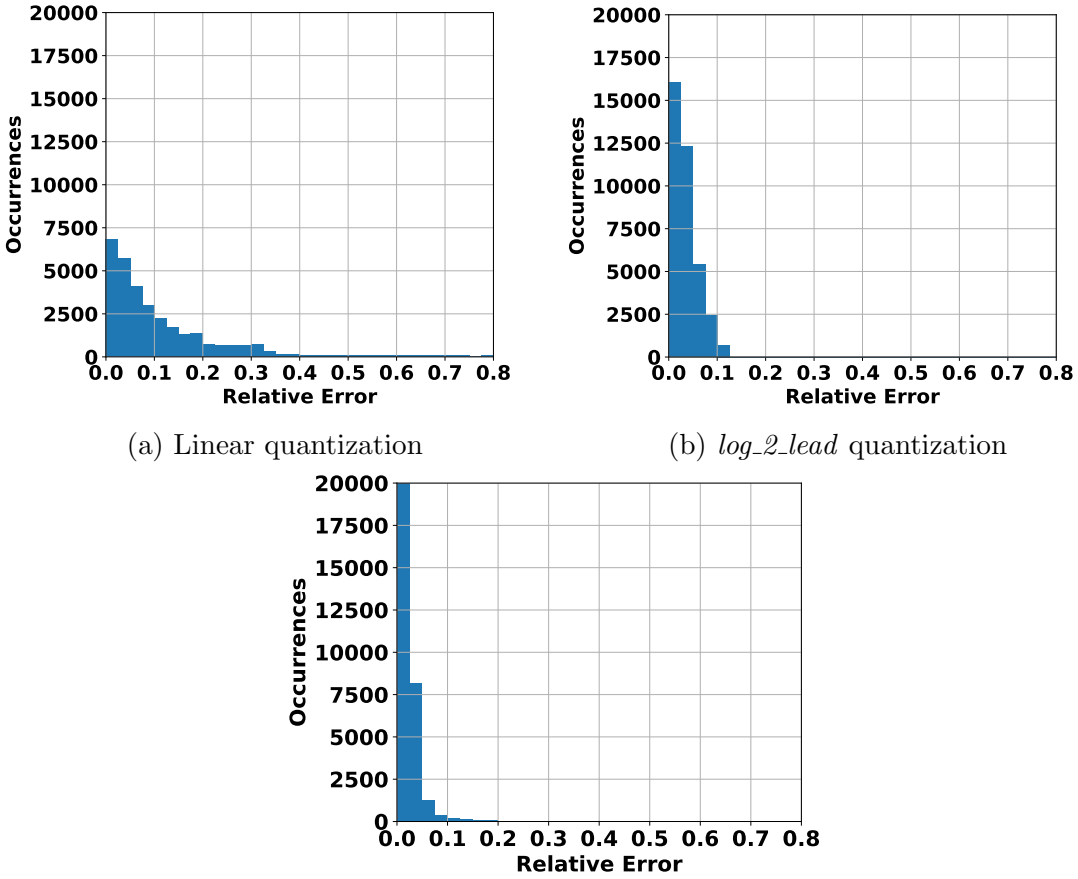


(a) Linear quantization

(b) *log_2_lead* quantization

(c) *ALigN* quantization

Figure 4.3: Relative error distribution of quantized weights of Conv1 layer from AlexNet [3] using different quantization schemes



(a) Linear quantization

(b) *log_2_lead* quantization

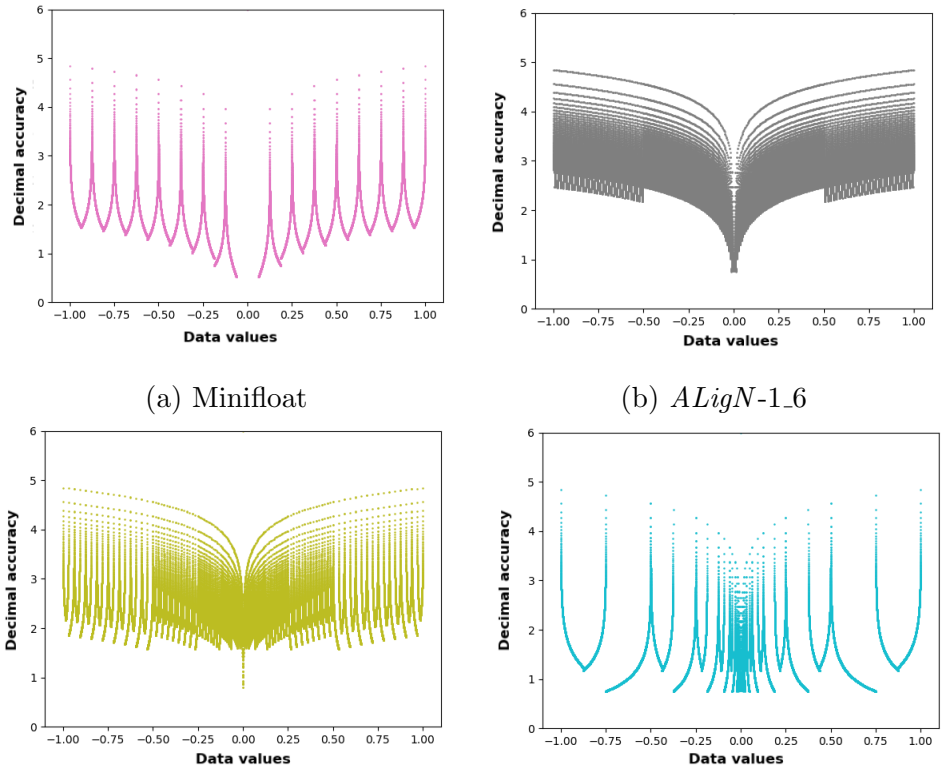
(c) *ALigN* quantization

Figure 4.4: Relative error distribution of quantized weights of Conv1_2 layer from VGG-16 [5] using different quantization schemes

4.2 Decimal Accuracy Measures

To show the efficacy of our proposed quantization schemes for the parameters of pre-trained DNNs, we compare its decimal accuracy in the range of -1 to +1 with bfloat16 [23] and minifloat [27] (this is independent of DNNs). The decimal accuracy, defined in equation 4.3, represents the capability of a number system to represent Float32-based numbers [45].

$$Decimal\ Accuracy = -log_{10}\left| - log_{10}\left(\frac{X_{quant}}{X_{float}}\right)\right| \tag{4.3}$$



(a) Minifloat

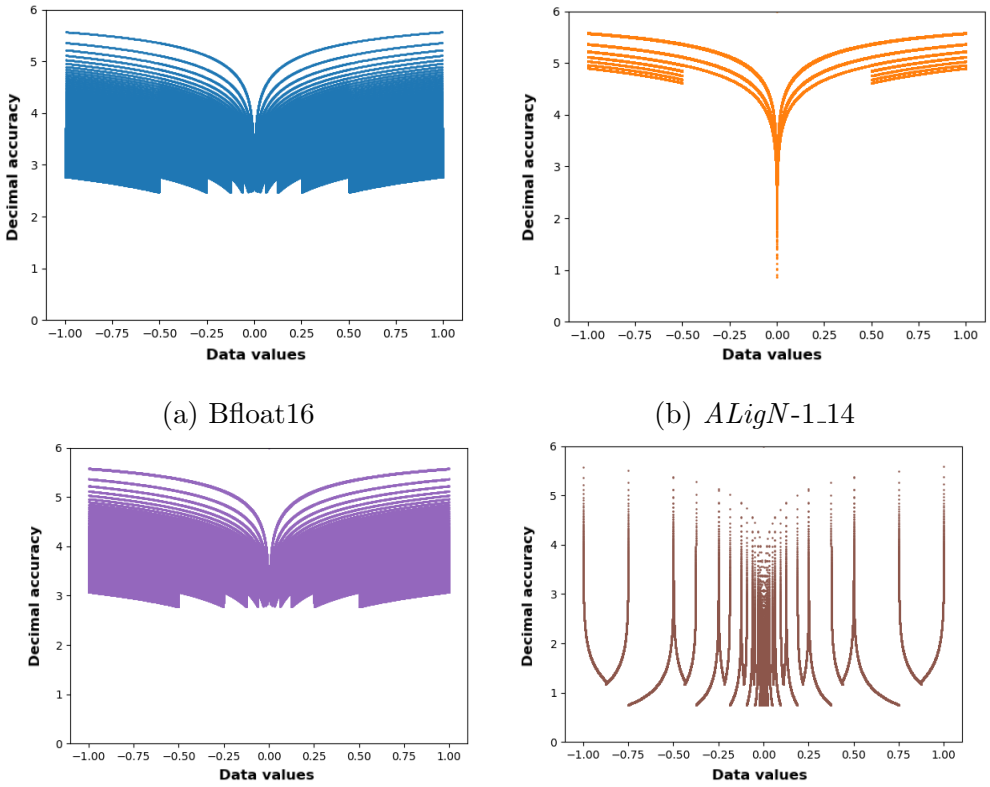
(b) *ALigN*-1_6

(c) *ALigN*-3_4

(d) *ALigN*-6_1

Figure 4.5: Decimal accuracy of different 8-bit quantization schemes

First, we compare the decimal accuracy of 8-bit ALigN representations with minifloat in Figure 4.5. Higher decimal accuracy shows better number representation capability. It can be observed that 8-bit ALigN representations provide higher decimal accuracy compared to minifloat. Further, we compare the decimal accuracy of 16-bit



(a) Bfloat16

(b) *ALigN*-1_14

(c) *ALigN*-7_8

(d) *ALigN*-14_1

Figure 4.6: Decimal accuracy of different 16-bit quantization schemes

ALigN representations with bloat16 in Figure 4.6. As evident from various subfigures, ALigN again performs better than that.

For better comparative analysis, we show the worst decimal accuracy of different quantization schemes in Figure 4.7. For 8-bit quantization, the ALigN-1_6 (1 bit reserved for storing the leading one location and 6 bits allocated to store the following bits) provides the best results among all other 8-bit variations. Similarly, ALigN-1_14 scheme provides the highest decimal accuracy. The ALigN-7_8 scheme provides slightly better performance than the bfloat16.

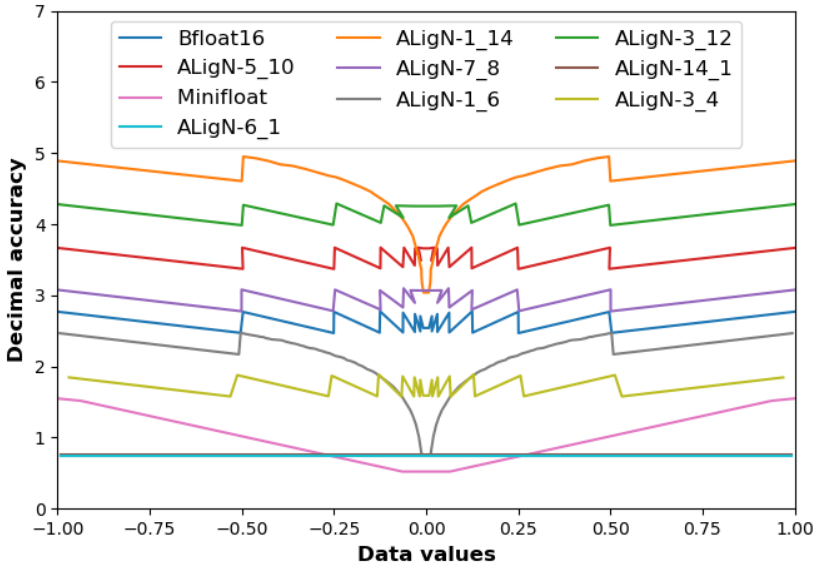


Figure 4.7: Decimal accuracy of different quantization schemes for the active range of values of pre-trained parameters (−1 to +1). *ALigN*-x_y shows x bits reserved for storing the leading 1 location and y bits allocated to store the following values after leading 1 location.

Figure 4.8 shows the average decimal accuracy of the pre-trained weights of a single layer, Conv1_2, of VGG-16 network using different quantization schemes. For 8-bit quantization, the ALigN-3_4 scheme produces better results than other corresponding 8-bit schemes. The ALigN-3_12 configuration represents the quantized weights with the highest accuracy. Similar results can be generated for other layers of the deep neural network.

Further, we have observed that quantization-induced errors can be significantly

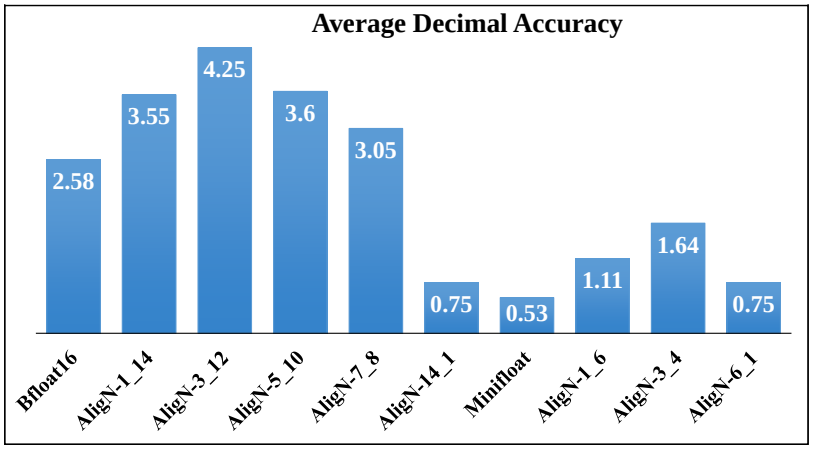


Figure 4.8: Average decimal accuracy of the weights of Conv1_2 layer of VGG-16 [5] network using different quantization schemes. *ALigN*-x_y shows x bits reserved for storing the leading 1 location and y bits allocated to store the following values after leading 1 location.

reduced by deploying an 8-bit ALigN scheme. Table 4.1 presents the Top-1 and Top-5 percentage classification accuracies using various bit-widths for the ALigN scheme for two state-of-the-art DNNs. As shown by the results, the classification accuracy improves for the both networks by increasing the quantization bit-width.

However, increasing quantization bit-width beyond 8 bits does not have a significant effect on classification accuracy. The corresponding cost in terms of resource utilization, critical path delay, and energy consumption of the hardware implementation of the quantized DNN also grows with the rising quantization bit-width.

Table 4.1: ALigN-based quantization of two state-of-the-art DNNs using different bit-widths

Quantization Bit-width	**ResNet-18**		**VGG-16**	
	Top-5 %	Top-1 %	Top-5 %	Top-1 %
Float32	88.94	69.3	85.74	64.72
6	86.59	65.71	85.13	63.65
7	88.61	68.33	85.64	64.53
8	88.84	69.42	85.71	64.62
9	89.05	69.2	85.74	64.71
10	88.95	69.34	85.7	64.67

4.3 Summary

This chapter provides a detailed quantitative analysis of proposed quantization techniques. We have presented quantization induced errors in various layers of DNN and error distribution on the parameter level. Decimal accuracy describes the number of representation capabilities of parameter representation. Therefore, we have compared the decimal accuracy of the proposed techniques with bfloat16 and minifloat. We have also discussed the quantization error by varying the bit-width of the ALigN scheme. Our proposed techniques significantly reduce the quantization error and provide higher decimal accuracy compared to other quantization techniques.

Chapter 5

Hardware Realization of Proposed Techniques

In this chapter, we present the hardware realization of the proposed quantization techniques, which reduces the hardware resources needed for quantized parameter processing. For the multiplication operation involved in convolutional and fully-connected layers, we have designed a processing element (PE) that reduces the hardware resources. In this PE, computation hungry multiplication operation turns into low computation bit-shift and addition operations. We also present the comparison of resource utilization for the proposed multiplier and linear multiplier.

5.1 Processing Element

For accelerated computation in hardware, processing elements are used that represent the computational core. An increase in computational efficiency of processing elements results in an overall increase in the hardware design performance. Linear and logarithmic PEs are commonly used processing elements for quantized deep neural network parameters [29].

Processing elements usually refer to a combination of components to perform some specific task. Vogel et al. [10] proposed a processing element that uses shift and addition operations for the multiplication of quantized weights W and activations X, and the same is shown Figure 5.1, which forms our base. In this section, we first describe

the algorithm for processing the proposed quantized parameters for multiplication, and then we discuss the new multiplier design.

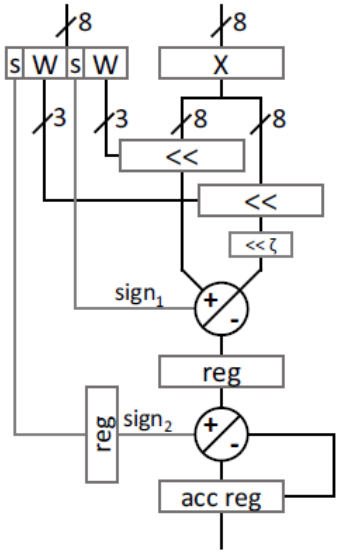


Figure 5.1: PE for two-hot quantized weights multiplication [10]

5.1.1 Algorithm for DNN Parameters Processing

The quantized weights consist of a dynamic range of values between -1 to +1. Therefore, it is not possible to realize the multiplier with fixed number of shift operations. This variable number of shift operations may lead to an increase in overall hardware resources. To tackle this problem, Algorithm 1 describes a novel resource-efficient implementation of the multiplication of an N-bit fixed-point feature with an N-bit L2L- or ALigN-based quantized weight. First, it analyses the $\lfloor \frac{N-1}{2} \rfloor$ 'remaining locations' bits for quantized weight, for each set bit, it performs the shift operation and adds the value to result (lines 2-4). Lines 5-7 show the shifting of the output according to the 'leading one location' bits. For ALigN-based implementation, different layers can have different configurations for storing the leading 1 location and the remaining bits. Algorithm 1 allows us to adapt the multiplier for each layer independently by changing the number of bits for storing leading 1 location and remaining bits. We have proposed lookup tables for implementing Algorithm 1.

Algorithm 1: Multiplication using *L2L* and *ALigN*

Input: Feature: x, Weight: w encoded in *L2L* or *ALigN* format

Output: y

$y \leftarrow x$

for $i \leftarrow 1$ ***to*** $\lfloor \frac{N-1}{2} \rfloor$ **do**

  if $w \left[\lfloor \frac{N-1}{2} \rfloor - i \right] \neq 0$ **then**

    $y \leftarrow y + (x >> i)$

for $j \leftarrow 0$ ***to*** $\lceil \frac{N-1}{2} \rceil - 1$ **do**

  if $w \left[\lfloor \frac{N-1}{2} \rfloor + j \right] \neq 0$ **then**

    $y \leftarrow y >> 2^j$

return y

5.1.2 Multiplier Design

In the convolutional and fully connected layers of DNN, multiplication and accumulation of activation and parameters occur. Equation (5.1) and Figure 5.2 describe the multiplication of single-precision Float32 based parameters and activations. In Ristretto [27], authors replace the weight parameter w with quantized weights parameters q as shown in Equation (5.2). Then, they multiply the power of 2 quantized value q to x by using shift operation, and each multiplied value is further added to get the approximated output of a particular neuron as described in Equation (5.3). Similarly, Vogel et al. [29] propose a multiplier design using bit-shift and addition operations to reduce the computational resources required for multiplication.

$$output = \sum_i x_i.w_i \tag{5.1}$$

$$q_i = round(log2(w_i)) \tag{5.2}$$

$$output \approx \sum_i x_i >> q_i \tag{5.3}$$

However, just considering the single-bit binary for parameter representation reduces the precision of values. In our techniques, we retain the precision of parameters and

use bit-shift and addition operations for the multiplication operation.

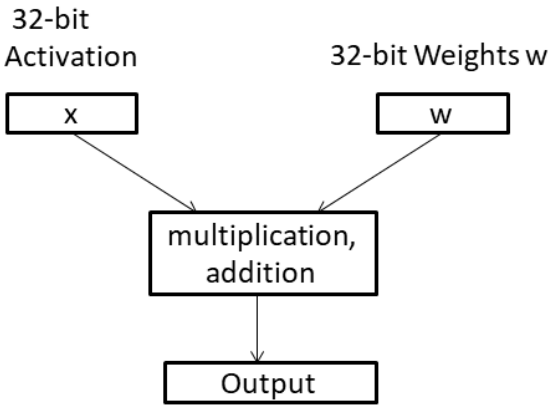


Figure 5.2: single precision floating point multiplication of activation weights of a DNN layer

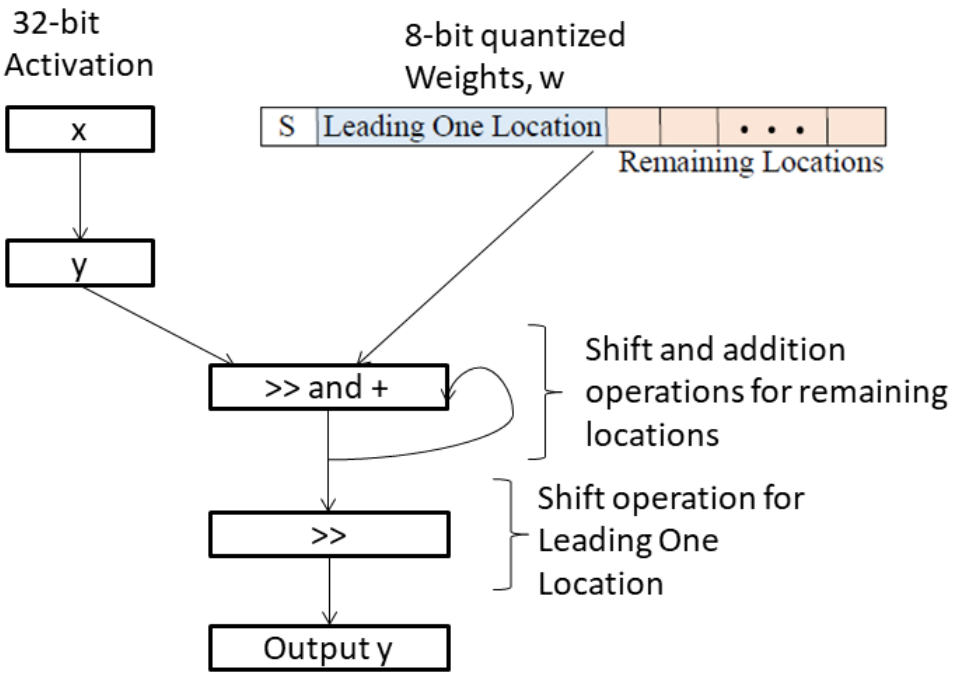


Figure 5.3: Multiplier-free multiplication of 8-bit quantized weights and single precision float activation

Figure 5.3 shows the multiplication of full precision activation x and 8-bit quantized weight w using shift and addition operations. First, the remaining location bits are

compared for shift and addition. Then, we shift the result by analyzing leading one location bits.

5.2 Experimentation

We have implemented proposed Processing Element using Algorithm 1 in VHDL for Xilinx Virtex-7 xc7v585tffg1157-3 FPGA using Xilinx-Vivado-17.4. Xilinx state-of-the-art FPGAs provide 6-input lookup tables (LUTs) for implementing various types of combinational and sequential logic. Our processing element utilizes these lookup tables. We compare the resource utilization of our proposed PE with Vivado standard multiplier IPs in Table 5.1. That is, we show the classification accuracies for VGG-16 network on ImageNet dataset using L2L and linear quantization with different bit-width for weights. We have implemented an 8-bit configuration of the proposed technique, i.e., L2L (4 bits for storing leading one location and 3 bits for storing remaining location). However, proposed implementation can be easily converted to other adaptive ALigN configurations. We have implemented various bit-width linear quantized multipliers and compared the accuracies as shown in Table 5.1.

The baseline for experimentation is single-precision Float32 based Top-5 and Top-1 accuracies i.e., 85.74% and 64.72%, respectively, which are not listed in the table. Top-5 and Top-1 accuracies of L2L based parameters and activation are higher, and lookup table resources are also lower than an 8-bit linear quantized multiplier. With the increase of bits, the number of LUT resources for the Vivado multiplier IP increase rapidly. Even with 18 bits of precision, the accuracy of linearly quantized weights design is only marginally better than L2L, despite consuming more than twice the area of L2L design.

Table 5.1: Comparison of resource utilization of proposed PE and Vivado multiplier IPs along with corresponding classification accuracies for VGG-16 [5] network on ImageNet dataset [6]

Design	**Quantization Scheme**	**Feature size (bits)**	**Weight size (bits)**	**Resources (LUTs)**	**Achieved Accuracy**	
					Top-5 [%]	**Top-1** [%]
PE	*log_2_lead* (L2L)	8	8	67	85.63	64.47
IP	Linear	8	8	85	82.55	59.83
IP	Linear	8	9	89	85.34	64.26
IP	Linear	8	10	109	85.56	64.52
IP	Linear	8	11	111	85.72	64.65
IP	Linear	8	18	166	85.7	64.67

5.3 Summary

This chapter describes the hardware realization of the proposed parameter representation techniques. We have presented a processing element for the multiplication of parameters quantized with the proposed quantization techniques. We have also shown that the proposed PE efficiently utilizes the bit-shift and addition operations instead of the computationally heavy multiplication operation. We have also compared the resources required for implementing the proposed 8-bit log_2_lead based processing element with different bit-width Vivado standard multiplier IPs. Our technique requires the least number of resources as well.

Chapter 6

Conclusions and Future Work

This thesis has primarily investigated different approximation techniques used in deep neural networks. In particular, we have developed two quantization based approximation techniques that can be applied to pre-trained deep neural networks. First, we have proposed a single template based DNN parameter quantization technique called L2L. Further, we have introduced a quantization technique called ALigN that involves adaptive layerwise analysis of DNNs to find the most significant bits of parameters. The layerwise analysis further reduces quantization induced error. These proposed quantization techniques have been evaluated on various benchmark datasets and results compared with various state-of-the-art quantization techniques.

We have tested the proposed techniques on AlexNet, VGG-16, and ResNet-18 DNNs for image classification on ImageNet dataset. For image segmentation task, we have used FCN8 DNN on PASCAL Visual Object Classes dataset. For both the tasks, image classification, and segmentation, we have observed only a slight reduction in accuracy using 8-bit L2L and ALigN based quantized parameters compared to Float32 based parameters. Using our quantization schemes, we have also proposed a multiplier design that uses bit-shift and addition operations. Our proposed multiplier has shown $\approx 39\%$ reduction in hardware resource utilization compared to Vivado area-optimized multiplier IP.

6.1 Summary of Research Achievements

The objectives specified in Section 1.3 have been successfully fulfilled by the following contributions:

(i) **L2L: Single template based quantization:** We have proposed a quantization-based parameter representation technique, L2L, for deep neural networks in Chapter 3, which can be applied on pre-trained neural networks. This technique analyzes the Float32-based DNN parameters and identifies the bit positions of the most important bits. These bits help in retaining the precision of parameters. The proposed method is less computationally expensive compared to retraining, fine-tuning, and iteration based traditional DNN quantization methods. For the evaluation of our technique, we have quantized small and large deep neural networks and compared the accuracy of image classification on various datasets. For VGG-16 and ResNet-18 network, we observe around 0.24 drop in Top-5 accuracy using our 8-bit L2L technique, compared to full precision 32-bit parameters. L2L also significantly heals the quantization induced error in parameters of these DNNs.

(ii) **ALigN: Adaptive multi-template based quantization:** We have explored multi-template based parameter representation technique called ALigN for DNN quantization in Chapter 3. This approach adaptively aligns the available quantization bit-width according to leading 1 position of parameters of each layer. By providing different quantization templates for each layer, this approach reduces the quantization error further as compared to L2L technique. ALigN technique also does not require retraining or iterative approach for quantization. Further, we have presented various bit-width of the proposed scheme for two state-of-the-art DNNs. VGG-16 and ResNet-18 DNNs show only 0.065 drop in Top-5 accuracy using 8-bit ALigN technique compared to full precision 32-bit parameters.

6.2 Future Research Directions

Despite significant progress in the field of approximating deep neural networks, it can be explored in several interesting future directions as follows:

(i) **DNN quantization using sampling techniques**: Sampling is the process of selecting a subset of a large population that possesses the behavior of the entire population. Importance sampling has been used for different convex optimization problem. Katharopoulos and Fleuret [49, 50] propose the methods to reduce the gradient variance of stochastic gradient descent by using importance sampling on training set of deep learning networks. Using these methods, they speed up the training process of deep neural networks and reduces training loss. Further, Mordido et al. [21] propose a quantization scheme for pre-trained training using importance sampling. With their quantization approach, they quantize the weight parameters and activation of the neural network and reduce the memory and computation requirement for inference. Seo et al. [51] use the kernel density estimation to obtain the probability density function (PDF) of convolutional neural network parameters and perform sampling through this PDF. With this approach, they quantize the deep convolutional neural network parameters in low bit-width. However, their approach suffers the computation of fine-tuning the quantized parameters. By taking a cue from the approaches mentioned above, other sampling techniques such as deterministic sampling and probability density-based sampling techniques can be explored for the quantization of deep neural networks.

(ii) **Training of DNN with log_2_lead and ALigN quantized parameters**: In our approach, we have applied our proposed quantization techniques on pre-trained deep neural networks to reduce the computation and memory needs during the inference phase of DNNs. Our approach can be further enhanced for even training with quantized parameters, as our technique retains the precision of parameter and provides a dynamic range for training. Hubara et al. [36] propose training of deep neural network models with reduced bit-width parameters.

Training with these quantized parameters reduces the computations involved in forward and backpropagation. Few researchers also present efficient training using quantized parameters [4, 34].

(iii) **Hardware accelerator for DNN using log_2_lead and ALigN techniques**: We have developed a processing element for multiplication using bit-shift and addition operations for values quantized with the proposed technique. Some works have proposed full hardware accelerators for deep neural network processing [40, 44, 42]. Our proposed processing element can also be enhanced for a full hardware accelerator for DNNs.

For other possible future directions, we can also consider the proposed L2L and ALigN quantization techniques for natural language processing and text generation tasks of machine learning.

Bibliography

[1] William Kahan. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE 754, UC Berkeley*, page 11, 1996.

[2] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Advances in neural information processing systems*, pages 1742–1752, 2017.

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

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv:1412.7024*, 2014.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

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

[7] Using Bfloat16 with tensorflow models. `https://cloud.google.com/tpu/docs/bfloat16`, 2020. [Online; accessed 31-May-2020].

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[9] Mark Everingham, S.M. Ali Eslami, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[10] Sebastian Vogel, Jannik Springer, Andre Guntoro, and Gerd Ascheid. Self-supervised quantization of pre-trained neural networks for multiplierless acceleration. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1094–1099. IEEE, 2019.

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

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

[13] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[14] Md. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[16] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the*

international conference on machine learning, volume 28. ACM New York, USA, 2013.

[17] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *arXiv:2001.05566*, 2020.

[18] Helmut Bölcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. Optimal approximation with sparsely connected deep neural networks. *SIAM Journal on Mathematics of Data Science*, 1(1):8–45, 2019.

[19] Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *arXiv:1605.06402*, 2016.

[20] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv:1603.01025*, 2016.

[21] Gonçalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. Instant quantization of neural networks using monte carlo methods. *arXiv:1905.12253*, 2019.

[22] Miguel de Prado, Maurizio Denna, Luca Benini, and Nuria Pazos. Quenn: Quantization engine for low-power neural networks. In *Proceedings of the 15th ACM International Conference on Computing Frontiers*, pages 36–44, 2018.

[23] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv:1905.12322*, 2019.

[24] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv:1510.03009*, 2015.

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

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

[27] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5784–5789, 2018.

[28] Hokchhay Tann, Soheil Hashemi, R. Iris Bahar, and Sherief Reda. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[29] Sebastian Vogel, Mengyu Liang, Andre Guntoro, Walter Stechele, and Gerd Ascheid. Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–8, 2018.

[30] Syed Shakib Sarwar, Swagath Venkataramani, Anand Raghunathan, and Kaushik Roy. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 145–150. IEEE, 2016.

[31] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. *arXiv:1908.11348*, 2019.

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

[33] Xi Chen, Xiaolin Hu, Hucheng Zhou, and Ningyi Xu. Fxpnet: Training a deep convolutional neural network in fixed-point representation. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2494–2501. IEEE, 2017.

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

[35] 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:1606.06160*, 2016.

[36] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[37] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858, 2016.

[38] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[39] Emmett Kilgariff, Henry Moreton, Stam Nick, and Brandon Bell. NVIDIA turing architecture in-depth. `https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth`, 2018. [Online; accessed 31-May-2020].

[40] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149*, 2015.

[41] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170, 2015.

[42] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 16–25, 2016.

[43] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE, 2014.

[44] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[45] John L Gustafson and Isaac T Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, 2017.

[46] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467*, 2016.

[47] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. AT&T Labs, 2010.

[48] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research), 2009.

[49] Angelos Katharopoulos and François Fleuret. Biased importance sampling for deep neural network training. *arXiv:1706.00043*, 2017.

[50] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. *arXiv:1803.00942*, 2018.

[51] Sanghyun Seo and Juntae Kim. Efficient weights quantization of convolutional neural networks using kernel density estimation based non-uniform quantizer. *Applied Sciences*, 9(12):2559, 2019.