Ternary Neural Networks

Vivek, Vishnu

Award date:
2021

Eindhoven University of Technology

Department of Electrical Engineering
Electronic Systems group

Thesis Report

Ternary Neural Networks

Author:

Vishnu Vivek
1435884

Supervisor

prof. dr. Henk Corporaal

Abstract

Convolutional neural networks (CNNs) have achieved great successes in various domains of artificial intelligence, but they require large amounts of memory and computational power. This severely restricts their implementation on resource-constrained devices on the edge. One approach to solving this problem is CNN quantization. Within CNN quantization, weights and/or activations of a CNN are quantized to lower bit widths. This has a two-fold advantage: memory savings due to lesser bit width weights and simpler computational hardware. Binary neural network quantization is the most popular form of network quantization. In binary quantization, weights and activations are limited to two symbols. The symbols [-1, 1] or [0, 1] are most widely used. It provides the largest benefit in model compression and computational simplicity. Within this work, the focus is ternary CNN quantization. In ternary quantization, the weights and/or activations are limited to three symbols. The symbols [-1, 0, 1] or [0, 1, 2] are the most widely used. This quantization scheme has more representational capacity than binary quantization. Ternary quantization still retains a fairly high model compression and computational simplicity. Binary quantization schemes also have a range of performance enhancement methods. Such methods are not seen widely within the ternary quantization. Within this work, a range of ternary quantization methods are implemented with the focus of establishing the effectiveness of completely ternary CNNs. In addition, accuracy enhancement methods from binary quantization are applied to these completely ternary networks and their effects are evaluated. Experimental results from multiple datasets and network architectures clearly indicate the benefit of applying binary CNN accuracy recovery methods to ternary CNNs. An accuracy improvement of 2% is observed when applying a combination of binary accuracy recovery methods to a completely ternary CNN. This improvement is observed across CNN architectures and datasets, further motivating the use of accuracy enhancements methods on ternary CNNs.

Table of Contents

Abstract **i**

1 Introduction **1**
1.1 Problem statement . . . 1
1.1.1 Research questions . . . 2

2 Background **3**
2.1 Convolutional Neural Networks (CNNs) . . . 3
2.1.1 The convolution operation . . . 3
2.1.2 CNN architecture overview . . . 4
2.2 Quantization CNN models . . . 4
2.2.1 Quantization within a CNN structure . . . 5
2.3 Binary CNN quantization . . . 7
2.4 Ternary network quantization . . . 8
2.5 Training neural networks through quantizers . . . 9
2.6 Cost of accuracy recovery methods . . . 11
2.6.1 Extra "cost" at training time . . . 11
2.6.2 Extra "cost" at inference time . . . 12

3 Related Work **13**
3.1 Ternary CNN Quantization methods . . . 13
3.1.1 Ternary Weight Networks (TWN) . . . 13
3.1.2 Trained Ternary Quantization(TTQ) . . . 15
3.1.3 Fine Grain Quantization(FGQ) . . . 17
3.1.4 Embarrassingly Simple Approach(ESA) . . . 19
3.1.5 Fast and Accurate TNN (FATNN) . . . 21
3.2 Accuracy recovery methods from binary quantization . . . 23
3.2.1 Generalized activation functions . . . 23
3.2.2 Approximate derivative of Sign . . . 25
3.2.3 Progressively Hardening Quantizer . . . 26
3.2.4 Element wise gradient scaling (EWGS) . . . 28

4 Method **31**
4.1 Experiments . . . 31
4.1.1 CNN architectures . . . 31
4.1.2 Datasets . . . 34
4.2 Design Space . . . 34
4.2.1 Design space exploration (DSE) . . . 35

5 Results and Discussion **36**
5.1 Full precision CNN baselines . . . 36
5.2 Ternary quantized CNNs . . . 37
5.2.1 Ternary weight networks . . . 37
5.2.2 Ternary activation networks . . . 39
5.2.3 Completely ternary CNNs . . . 40
5.2.4 Ternary quantizers as weight regularizers . . . 42
5.3 Accuracy recovery methods from Binary Quantization . . . 44
5.3.1 Effect of ReAct PReLU on TNNs . . . 44
5.3.2 Ablation of ReAct PReLU's components . . . 45
5.3.3 Effect of Approximate sign quantizer on TNNs . . . 47
5.3.4 Effect of Progressively hardening quantizer on TNNs . . . 49
5.3.5 Effect of longer training times on quantizer enhancements . . . 50

5.3.6 Effect of Element wise gradient scaling on TNNs 52

6 Summary and Conclusions **53**
6.1 Reflecting on the research questions 53
6.1.1 Research question 1: 53
6.1.2 Research question 2 53
6.2 Conclusions 54
6.3 Suggestions and Future Work 54

List of Figures

2.1 Example of Convolution components . . . 3
2.2 Step 1 and 2 of the convolution process . . . 4
2.3 Step 3 and 4 of the convolution process . . . 4
2.4 Overview of a CNN architecture . . . 5
2.5 Overview of the quantization process . . . 6
2.6 The heart of quantizing CNNs: Quantizing the convolution layers . . . 6
2.7 Internal structure of a quantized convolution layer . . . 7
2.8 Example of a binary quantizer . . . 8
2.9 Summary of a ternary quantizer . . . 9
2.10 Binary quantizer with STE . . . 10
2.11 Ternary quantizer with STE . . . 11

3.1 Ternary Weight Quantization procedure . . . 15
3.2 TTQ training procedure[1] . . . 16
3.3 Trained Ternary Quantizer . . . 17
3.4 Fine grain Quantization Weight partitioning and scaling factors . . . 18
3.5 Fine Grain Ternary Quantization procedure . . . 19
3.6 Shape of WDR for a set of γ values . . . 20
3.7 Embarrassingly simple approach quantization procedure . . . 21
3.8 FATNN weight quantizer . . . 22
3.9 FATNN activation quantizer . . . 23
3.10 Generalized activation functions . . . 24
3.11 ReAct PReLU: in more detail . . . 24
3.12 Binary Sign and Approx Sign . . . 25
3.13 Ternary Quantizer and Ternary Approx Sign . . . 26
3.14 Binary Sign and Progressively hardening quantizer . . . 27
3.15 Ternary quantizer and Progressively hardening quantizer . . . 28
3.16 Element Wise Gradient Scaling working alongside the quantizer . . . 29
3.17 Intuition for Element Wise Gradient Scaling factor value . . . 30

4.1 Unified CNN architecture skeleton . . . 32
4.2 ResNet Convolution block . . . 32
4.3 EfficientNet Convolution block . . . 33

5.1 Full precision baseline on CIFAR-10 . . . 36
5.2 Full precision baseline on CIFAR-100 . . . 36
5.3 Ternary weight ResNet20 on CIFAR-10 . . . 37
5.4 Ternary weight EfficientNet on CIFAR-10 . . . 37
5.5 Ternary weight ResNet20 on CIFAR-100 . . . 38
5.6 Ternary weight EfficientNet on CIFAR-100 . . . 38
5.7 Ternary activation network: CIFAR-10 . . . 39
5.8 Ternary activation network: CIFAR-100 . . . 39
5.9 TNN ResNet on CIFAR-10 . . . 40
5.10 TNN EfficientNet on CIFAR-10 . . . 40
5.11 TNN ResNet on CIFAR-100 . . . 40
5.12 TNN EfficientNet on CIFAR-100 . . . 40
5.13 Regularization of weights under a ternary quantizer . . . 42
5.14 Resistance to parameter update in a ternary quantizer . . . 43
5.15 Movement of weights towards a decision boundary (Δ_1) in training . . . 43
5.16 Effect of RPReLU on TNNs for CIFAR-10 . . . 44
5.17 Effect of RPReLU on TNNs for CIFAR-100 . . . 44
5.18 R-PReLU ablation Tests 1 and 2: TNN-ReLU, TNN-PReLU . . . 46
5.19 R-PReLU ablation Tests 3 and 4: TNN-RReLU, TNN-RPReLU . . . 46

5.20 RPReLU ablation test results for ResNet-20 47
5.21 RPReLU ablation tests on scalar shifts 47
5.22 RPReLU scalar shift ablation test results for ResNet-20 48
5.23 Effect of Approximate sign on ResNet20 48
5.24 Effect of Approximate sign on EfficientNet 49
5.25 Effect of progressively hardening quantizer on ResNet20 49
5.26 Effect of progressively hardening quantizer on EfficientNet 50
5.27 Training curve for quantizer enhancement methods 51
5.28 Effect of longer training STE enhancements for ResNet20 51
5.29 Effect of Element wise gradient scaling on ResNet20 52
5.30 Effect of Element wise gradient scaling on EfficientNet 52

List of Tables

4.1 Common CIFAR characteristics . . . 34
4.2 CIFAR-10/100 characteristics . . . 34
4.3 Design space of this work . . . 34
4.4 Hyperparameter settings for CNN training . . . 35

5.1 Summary of full precision baseline results . . . 37
5.2 Summary of ternary weight method results: ResNet . . . 38
5.3 Summary of ternary weight method results: EfficientNet . . . 38
5.4 Accuracy degradation of the best ternary weight method . . . 39
5.5 Summary of Ternary activation method results . . . 39
5.6 Summary of TNN method results: ResNet . . . 41
5.7 Summary of TNN method results: EfficientNet . . . 41
5.8 Accuracy degradation of best TNN methods . . . 41
5.9 Summary of TNN method with RPReLU: ResNet . . . 45
5.10 Summary of TNN method with RPReLU: EfficientNet . . . 45
5.11 Accuracy improvement of TNN methods with RPReLU . . . 45

Chapter 1

Introduction

Deep Neural Networks (CNN)[2] have substantially pushed the state-of-the-art for a large range of learning-based tasks. This includes applications like speech recognition [3, 4], computer vision applications like object detection [5, 6, 7, 8, 9, 10] and segmentation [11, 12, 13]. This motivates the deployment of these CNNs for real-world applications based on smartphones, Internet of Things (IoT) sensors and other edge devices. Convolution Neural Networks (CNNs)[14] are a subset of CNNs that have become the cornerstone of many computer vision applications as they are well suited to process[5], transform[15] and even generate[16] 2-D data.

CNN-based applications typically require large memory and computational power. A very popular CNN architecture like ResNet-18 has a model size exceeding 45 MB and requires more than 2×10^9 floating-point operations (FLOPS). These requirements exceed the limited storage, battery and computational capabilities of small embedded devices. Recently, there has been significant research on making CNNs more accessible to such resource constrained devices. One of the most critical methods to achieve this is Model quantization. Model quantization limits the bit-width of the weights and activations of the network lower than 32-bits. This process has the advantage of making the CNN model smaller in memory and lowers the mathematical complexity of performing inference using this quantized model.

1.1 Problem statement

Full precision CNNs as well as Binary CNNs are both very heavily researched topics. Binary quantization represents the most extreme quantization scheme, as well as the one with the smallest model size[17] and largest speedup possibility[18, 19]. Binary networks also suffer from the highest amount of quantization error. Methods to improve the performance of these networks are also abundant[20, 21, 22]. In stark contrast, ternary neural networks have drastically fewer papers proposing quantization methods or improvements to learning methods. Within ternary quantization research, the core focus is ternary weight quantization[1, 23, 24] and not fully ternary networks[25]. In addition, almost no papers published on methods to improve the learning for completely ternary networks.

The goal of this thesis is to motivate the utility of ternary quantized networks for deployment on IoT and resource constrained systems. This utility is further emphasized with the addition of accuracy recovery methods from binary CNNs applied to ternary CNNs. This goal is accomplished in two steps. The first step is to implement state-of-the-art ternary quantization methods and extend them to fully ternary neural networks. Then secondly, enhance the state of the art by leverage learning improvement/accuracy recovery methods from binary neural network works and apply them to these ternary networks.

The first goal of the project is to explore in detail how CNNs respond to ternarization and understand their behavior. Using this understanding, the second goal of gauging the impact of binary CNN accuracy recovery methods on ternary CNNs can be achieved. With this approach, assumptions made when formalizing these accuracy recovery methods for binary networks can be verified for the ternary context.

1.1.1 Research questions

Within this work, we seek to answer the following research questions:

- **Research question 1:**
  How much accuracy degradation do completely ternary quantized CNNs experience compared to full-precision CNNs? Compared to completely binary CNNs how much accuracy improvement do completely ternary CNNs provide?
- **Research question 2:**
  Given the large amount of binary neural network accuracy recovery methods, can we utilize/extend these methods to enhance Ternary quantization? How much improvement/accuracy gain do these methods provide for a set of standard benchmarks

Chapter 2

Background

This work focuses on quantization methods for ternary CNNs, as well as accuracy recover/enhancement methods from binary quantization. This section will cover the basic concepts essential to these topics. These include convolutional neural networks (CNNs), binary and ternary neural network quantization and the central concepts like the straight through estimation and learning through quantizers.

2.1 Convolutional Neural Networks (CNNs)

Within the book *Deep learning* [2], CNNs are deep neural networks designed to process data with grid like topologies. This includes 1-D data like time series data and 2-D data like images. The defining factor of CNNs is the use of convolution operations instead of matrix multiplications within the network. Within this work, our focus remains on CNNs for image processing.

2.1.1 The convolution operation

The first building block of a CNN is the convolution operation itself. In the context of neural networks, convolution is a linear operation performed on a 2-D input X. In a convolution, the input X is transformed using a set of convolution filters also referrer to as weights W. This is done to obtain an output feature map Y. As an example, these variables can be visualized in Figure 2.1.

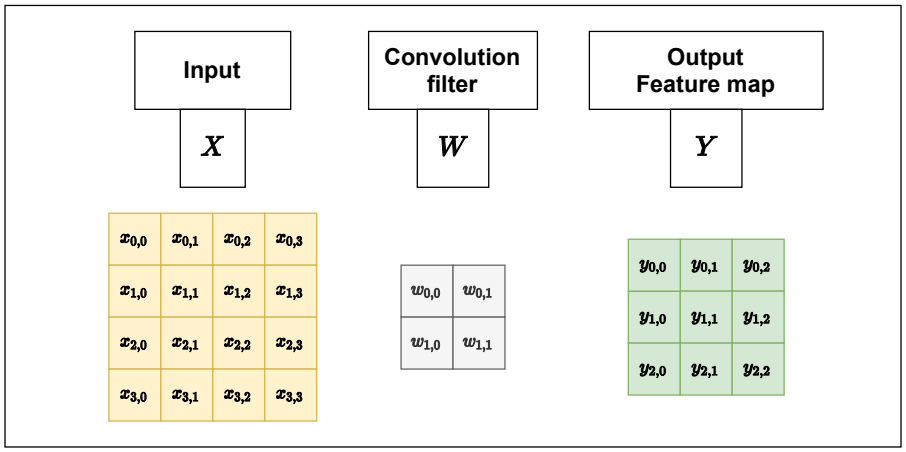


Figure 2.1: Example of Convolution components

The convolution operation between two matrices is represented by the Equation 2.1.

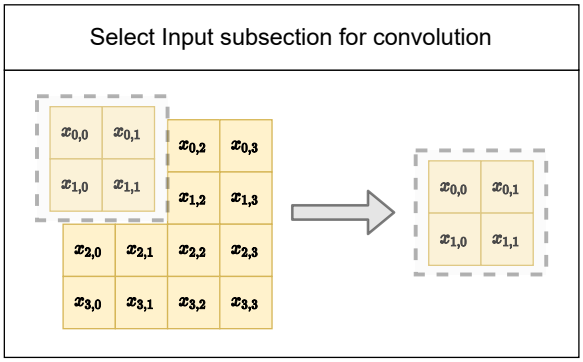
$$Y = X * W \tag{2.1}$$

Further expanding the equation above for individual elements of the Y, W and X give us the following representation. Where, i, j represent the coordinates of the current output element. m represents the height and width of the convolution filter.

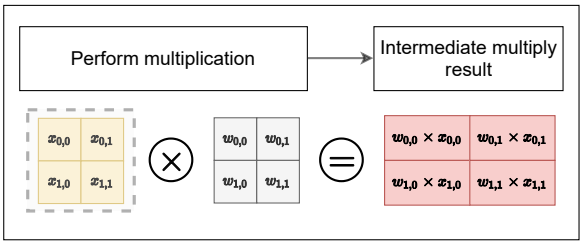
$$y_{(i,j)} = \sum_{k=1}^{m} \sum_{l=1}^{n} x_{(i+k,j+l-1)} \times w_{(k,l)} \tag{2.2}$$

Within the context of deep learning, the convolution operation involves the following steps:

1. Convolution Filter passes over the input in a sliding window manner. This can be represented as selecting a subset of the input matrix X which is the same dimension as W. As can be seen in Figure 2.2a.

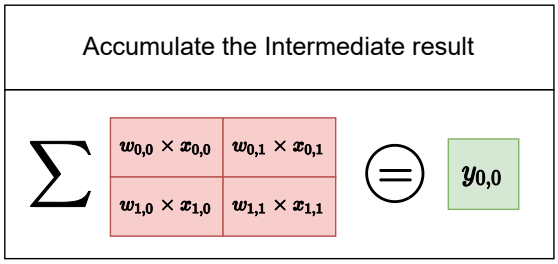


(a) Input matrix subsection selection

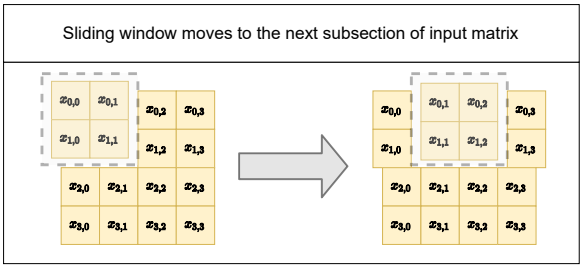


(b) Input and Conv. filter multiplication

Figure 2.2: Step 1 and 2 of the convolution process



(a) Input matrix subsection selection



(b) Input and Conv. filter multiplication

Figure 2.3: Step 3 and 4 of the convolution process

2. Multiply filter elements with input elements and obtain an intermediate result. This can be seen in 2.3a.
3. Then an accumulate action is performed on the intermediate result to obtain the result for that particular convolution output element. This can be seen in Figure 2.3a.
4. The next step is to advance the sliding window as can be seen in Figure 2.3b. After this step, the process restarts from Step 1 until the entire input matrix has been traversed.

This operation allows the CNN to learn 2-D features and distributions. Over multiple convolution layers and proper training, the network can specialize at detecting and highlighting important features that make decisions that reflect the ground truth more effectively. The convolution operation is simply a starting point to the end goal.

2.1.2 CNN architecture overview

An abstracted view of the architecture most CNNs follow can be seen in Figure 2.4.
It can be seen that the network is made up of two major components:

- **The feature extraction layers :** These layers consist of the convolution layers, pooling layers, non-linearities, batch-normalization etc. These are all operations that are performed on 2D data and are used to get the actionable features from an input image.
- **Classification layers :** This layer acts as the final decision-making portion of the network and is responsible for indicating the detected class or what output needs to be predicted from the input. This layer is made up of fully connected neurons.

2.2 Quantization CNN models

Quantization of CNNs is the process by which weights and/or activations are converted from continuous real valued[1] variables to discrete valued variables. These quantized variables are limited to lower precision. This

[1]In the context of this work, real valued implies that the variables are represented in 32-bit floating-point precision

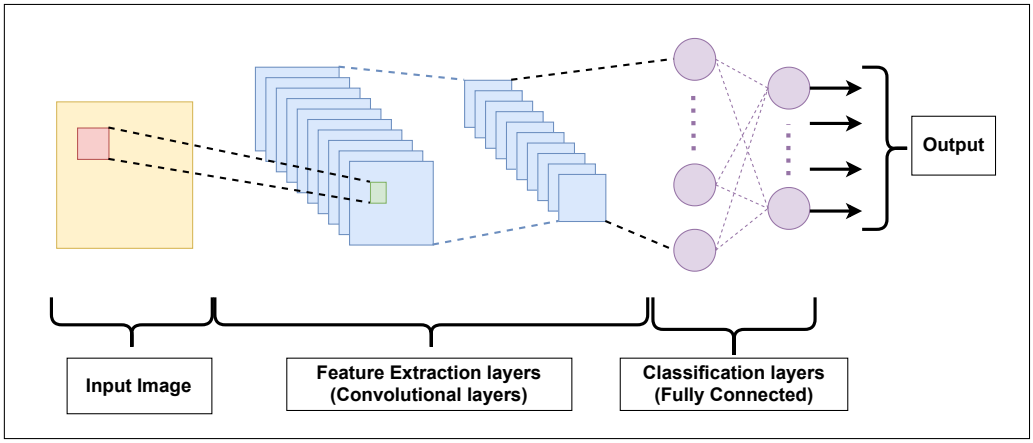


Figure 2.4: Overview of a CNN architecture

includes binary (1-bit)[17] and ternary (2-bit)[23] representations, along with other low-precision representations (4/8 bit)[26]. The key advantage of binary and ternary representations is the huge memory savings, as well as the ability to use bit-wise operations instead of integer/floating-point calculations. These makes quantized CNNs much smaller and computationally efficient. They are quite well suited to deployment on embedded computing systems as well as purpose-built hardware accelerators due to the memory and processing power limitations of such devices. Making these networks as efficient and accurate as possible is a very significant research focus.

A summary of the motivations for using quantized CNNs are as follows:

- **Smaller CNN Models:** By lowering the bit-width of weights for a CNN model, binary and ternary methods achieve up to 32× and 16× model compression. This has a twofold advantage:
  1. **Lower memory footprint:** These models can be compressed enough to fit on resource constrained hardware and can be used by Edge devices. A full precision ResNet-20 model is $\sim$ 1.7MB, the same model quantized to binary is $\sim$ 100KB and ternary is $\sim$ 120KB.
  2. **Energy savings on memory fetch:** Memory fetches[2] are 100× more expensive than MAC operations[27]. Smaller bit-width weights mean less memory fetches, this directly reduces the energy required for model inference. A smaller energy budget improves the efficiency and battery life for the system.[28, 29].
- **Simplified computation during inference:** Limiting the bit-width of weights and activations allows the replacement of 32-bit floating point Multiply Accumulate (MAC) operations with lower precision MACs or even bit-wise operations for binary and ternary cases. This can lead to large savings for compute energy usage and larger speedup due to simpler computation [23, 28, 29].

2.2.1 Quantization within a CNN structure

Quantization is the process by which some continuous valued variable X can be transformed into a variable X_q which has a finite number of values i. The quantization process represented in Equation 2.4 by a quantization function $F()$.

$$X \in \mathbb{R} \tag{2.3}$$

$$X_q = F(X), \text{ Where } X_q \in \{X_1, X_2 \ \ldots X_i\} \tag{2.4}$$

[2]Specifically refers to DRAM fetches required to access the weights of the CNN model

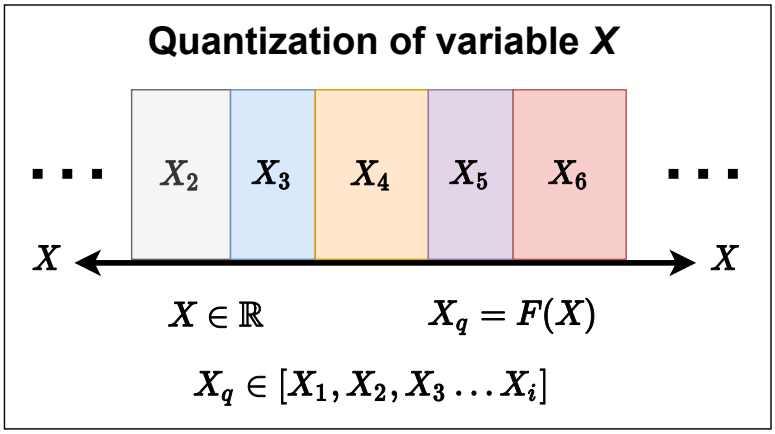


Figure 2.5: Overview of the quantization process
The process of quantization can be observed in this Figure where a continuous variable X is quantized into discrete variable X_q

With all the information presented in the previous sections, we can look into exactly where quantization is used in CNNs. The specific part we wish to quantize is the convolution operation itself. As the convolution operations account for most of the weights in a network. For ResNet-20[5], out of 270k parameters, 250k are convolution layer weights. Quantizing these parameters allows us to reduce a large amount of 32-bit full precision weights to a fraction of their bit-width. This reduction in size for the model parameters directly affects the final model size. A ResNet-20 model which is $\sim$ 1MB can become as small as $\sim$ 100KB when quantized to binary.

Within this work, the first convolution layer and the last fully-connected layer are both kept in 32-bit full precision. These two layers could also be represented in 4/8-bit with negligible accuracy degradation[26]. The rest of the convolution layers are ternarized. The quantization of a convolution layer is represented in the Figure 2.6. The internal structure of the quantized convolution layer looks like Figure 2.7. It can be seen that both the input to the convolution, referred to as the feature maps or activations and the weights of the convolution operation can be quantized. This can be done independently or at the same time. Depending on which variable is quantized, we can end up with ternary weight CNNs, ternary activation CNNs and completely ternary CNNs.

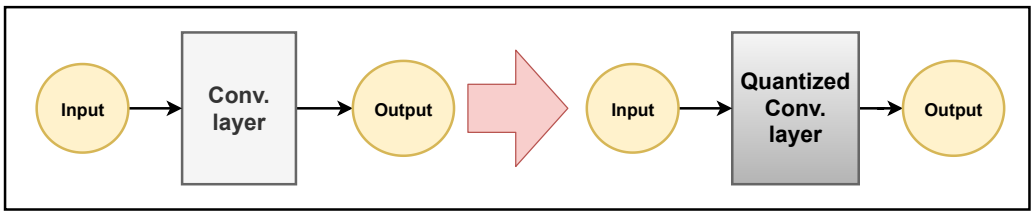


Figure 2.6: The heart of quantizing CNNs: Quantizing the convolution layers
This represents the most basic substitution that is made to quantize a CNN in a modular way.

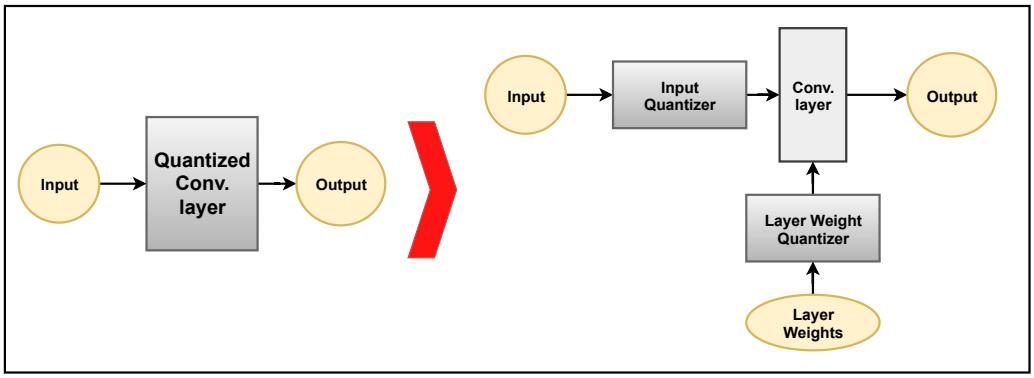


Figure 2.7: Internal structure of a quantized convolution layer
Within the scope of this work, the primary focus is on CNNs where both the activations and weights are always quantized to ternary. Ternary weight and ternary activation networks are used only to establish performance baselines.

In the following section, a more directed explanation of binary and ternary quantization will help further establish the premise of this work and some of the advantages and issues faced by those approaches.

2.3 Binary CNN quantization

Binary quantization methods constrain the weights and/or activations of a CNN from the full-precision range down to the binary symbols $\{-1, +1\}$ or $\{0, 1\}$ [17, 30]. These methods are the most extreme form of model compression and also suffer from the largest quantization error.

A generic binary quantizer can be described as a mathematical transform that converts a continuous real-valued variable into a discrete variable having two states. For some continuous variable X which can possess any real value within the range $(-\infty, \infty)$, a binary quantizer $F()$ will convert this variable into another that can only possess two distinct values. Within the context of binary CNNs, these values are $\{-1, 1\}$ or $\{0, 1\}$. This can be summarized in the Equation 2.5. Figure 2.8 shows an example of a binary quantizer.

$$X_q = F(X), \text{ Where } X_q \in \{-1, 1\} \text{ or } \{0, 1\} \tag{2.5}$$

Expressive ability: The binary states allow for limited representation within the convolutions filter maps. For a 3×3 convolution filter, a binary convolution filter has $2^{3\times3} = 512$ possible unique representations.

Model compression: With a 1-bit representation for the weights, binary networks achieve a $32\times$ model compression compared to full precision networks. The saving in memory read/writes for binarized networks is more than $32\times$ that of their full precision counterparts.

Computational savings: Binary networks can replace MAC operations with xnor and popcount operations. This makes the computation faster and simpler to execute on CPUs and GPUs. Multiply operations can be replaced with the *xnor* operation and accumulate operations can be replaced with the *popcount* operation. These operations take a fraction of the cycles of their full-precision MAC counterparts. With custom-built hardware [19], further speedup and energy savings can be observed.

In the full-precision case, a single output element of the convolution process is calculated in the following manner. Where, y represents the output element, x represents the input/feature maps and w represents the conv. filter and m represents the height and width of w filter.

$$y_{(i,j)} = \sum_{k=1}^{m}\sum_{l=1}^{n} x_{(i+k,j+l-1)} \times w_{(k,l)}$$

After binary quantization, this computation can be simplified into the following equation [17]. Where, x and w are quantized to two symbols.

$$y_{(i,j)} = popcount(xnor(x_{(i+k,j+l-1)}, w_{(k,l)})), \text{ where } k, l \in [1, m]$$

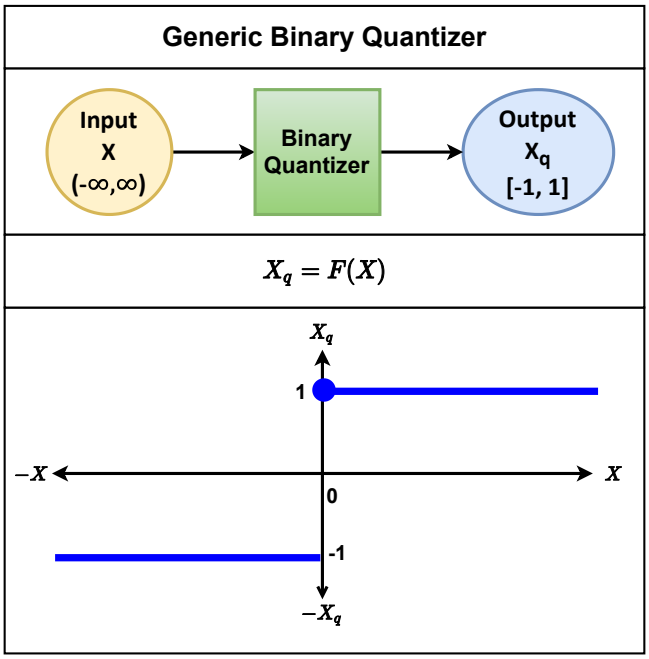


Figure 2.8: Example of a binary quantizer
This particular example variable X is quantized into {-1. 1}. Quantization to any two symbols is possible, but {-1, 1} and {0, 1} are preferred as *xnor* logic can be applied to speed up those multiplications.

2.4 Ternary network quantization

Ternary quantization methods constrain the weights and/oractivations of a CNN from the full-precision range down to the ternary symbols $\{-1, 0, +1\}$ or $\{0, 1, 2\}$. These methods claim to provide near full-precision performance and high model compression.

A generic ternary quantizer can be described as a mathematical transform that converts a continuous real-valued variable into a discrete variable having three states. This is a fundamentally more complicated problem to solve than a binary state division. Two quantization boundaries Δ_1 and Δ_2 are used to represent the points of state change. For some continuous variable X which can possess any real value within the range $(-\infty, \infty)$, a ternary quantizer $F_T()$ will convert this variable into another that can only possess three distinct values. Within the context of quantization, these values are $\{-1, 0, 1\}$ or $\{0, 1, 2\}$. This can be summarized in the equation 2.6 and Figure 2.9

$$X_q = F_T(X, \Delta_1, \Delta_2), \text{ Where } X_q \in \{-1, 0, 1\} \text{ or } \{0, 1, 2\} \tag{2.6}$$

Expressive ability: The ternary states allow the convolution filters to access an increased number of representational states. For a 3×3 convolution filter, a binary filter has $2^{3\times3} = 512$ possible representations. Whereas, a ternary filter has $3^{3\times3} = 19683$ possible representations. This allows a filter to have $38\times$ more possible states. Ternary representation also has in information entropy $1.6\times$ higher than binary symbols.

Model compression: With a maximum compression of 1.6^3 bits per symbol [31], ternary networks achieve a $16\times$ to $20\times$ compression compared to full precision networks. While not able to achieve the $32\times$ compression of binary networks, it is still sufficiently high compression for most State-of-the-art CNN models. There is still significant energy savings on memory fetches for weights compared to their full precision counterparts

Computational savings: Ternary computations can also be much more simplified compared to full-precision MACs. The bit representations of the two convolution operands (Weight filters and Input feature maps) are known beforehand and can be used to formulate the exact multiple and accumulate method to be used.

[3] Ternary states are stored in 2-bits but can be encoded to 1.6 bits per symbol[31]

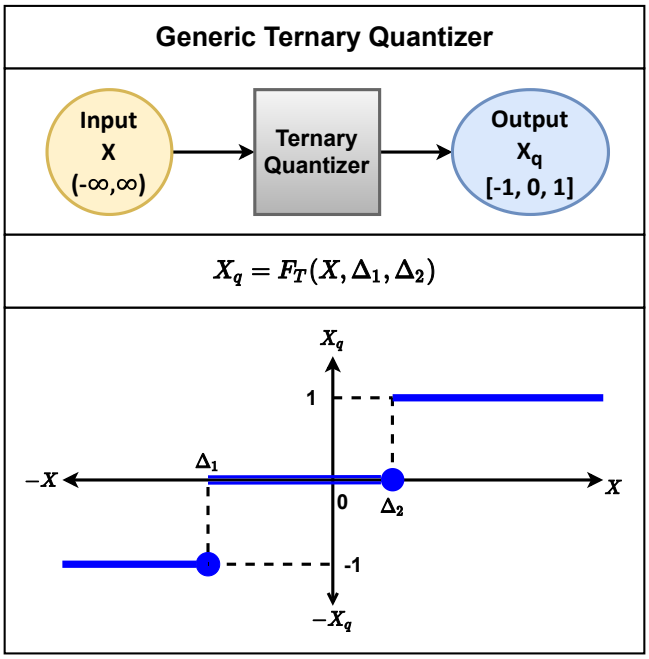


Figure 2.9: Summary of a ternary quantizer
This particular example variable X is quantized into {-1. 0, 1}. Quantization to any three symbols is possible, but {-1, 0, 1} and {0, 1, 2} are preferred as ternary multiplication logic can be applied to speed up those multiplications.

As an example, the ternarization of variable X is shown in we can Equation 2.7.

$$X_q = F_T(X, \Delta_1, \Delta_2) = \begin{cases} +1 & \text{-> } 2'b01 \\ 0 & \text{-> } 2'b01 \text{ or } 2'b10 \\ -1 & \text{-> } 2'b11 \end{cases} \tag{2.7}$$

In the full-precision case, a single output element of the convolution process is calculated in the following manner. Where, y represents the output element, x represents the input/feature maps and w represents the conv. filter and m represents the height and width of w filter.

$$y_{(i,j)} = \sum_{k=1}^{m} \sum_{l=1}^{n} x_{(i+k,j+l-1)} \times w_{(k,l)}$$

After ternary quantization, this computation can be simplified into the following equation [25, 28, 32]. Where, x and w are quantized to three symbols.

$$y_{(i,j)} = popcount(TM(x_{(i+k,j+l-1)}, w_{(k,l)})), \text{ where } k,\, l \in [1,\, m]$$

Where TM represents the ternary multiplication. This ternary multiplication can be accomplished using 2-bit multiply units [28] that can be made up of *xnor* and *and* gate logic[32, 33].

2.5 Training neural networks through quantizers

Training a CNN can be divided into three steps:

- **The forward pass:** In this step, the input images are given to the network. They propagate through the network and the last layer of the network provides us with a result. This result could be a class prediction, detected class bounding boxed or a segmented map of the input with highlighted classes. The nature of the result depends on the function expected from the CNN.
- **Loss calculation:** This forward pass result is compared to the ground truth within the loss function. This function creates a loss for the network. This loss is used by the training optimizers like Stochastic Gradient Descent or ADAM[34] to decide how much change the parameters of the network need. This measure of change is called the gradients.

- **The backwards pass:** In this step, the gradients are propagated through the network in a process called back propagation[35]. Within back propagation, we use the derivative chain rule to calculate how much each parameter affects the final loss. From these derivatives, the optimizers calculate the gradients for each parameter. The gradients are then applied to the parameters present in the network. These small changes over a number of training steps can help the network generalize and find optimal values for its parameters.

As the gradients within back propagation are calculated using the chain rule. This can be illustrated for a very simple expression: For some input X two mathematical transforms $F()$ and $G()$ are applied to it and we obtain an output Y.

$$Y = F(G(X)) \tag{2.8}$$

If we wish to see how much Y is affected by a change in X, we apply back propagation, This method uses partial differentials to obtain the desired gradient,

$$\frac{\partial Y}{\partial X} = \frac{\partial F(G(X))}{\partial X} = \frac{\partial F(G(X))}{\partial G(X)} \times \frac{\partial G(X)}{\partial X} \tag{2.9}$$

This essentially means that every mathematical transform applied within the deep learning framework requires that its derivative exist and not be infinitely large (infinitely large/exploding gradients) or small (vanishing or 0 gradients). This is challenging as the quantizers described in Figures 2.8, 2.9 are not compliant with our requirements. This is because a step function has a 0 derivative at all points except the step itself. On the step between states, it has an infinitely large gradient for an infinitely small interval. This causes both exploding gradients at the step location and vanishing gradients everywhere else.

An easy and effective solution to this is a Straight Through Estimator(STE)[17]. With this method, we "trick" the back propagation algorithm by specifying a differential term for our quantizers. In the experiments conducted within this work, the STE derivative has a value of 1 at all points, thereby simulating an $X = X$ operation in the forward pass. For the binary and ternary quantizers (Fig. 2.8, 2.9) the applied STE's are shown in Figures 2.10 and 2.11. These figures illustrate the simulated STE alongside the quantizer.

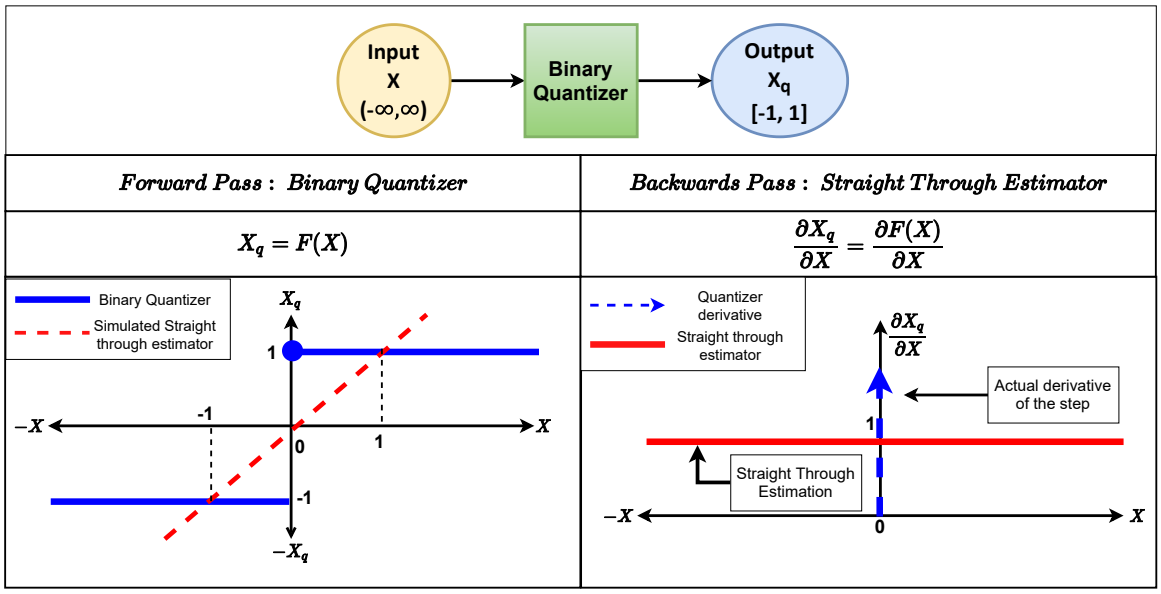


Figure 2.10: Binary quantizer with STE
This Figure represents the binary quantizer in the forwards and backwards pass of the CNN training process, shown in blue. In addition, the STE within the forwards and backwards pass is shown in red. During the CNN training process, the quantizer is used in the forward pass and the STE derivative is used in the backwards pass, these are shown by the solid lines.

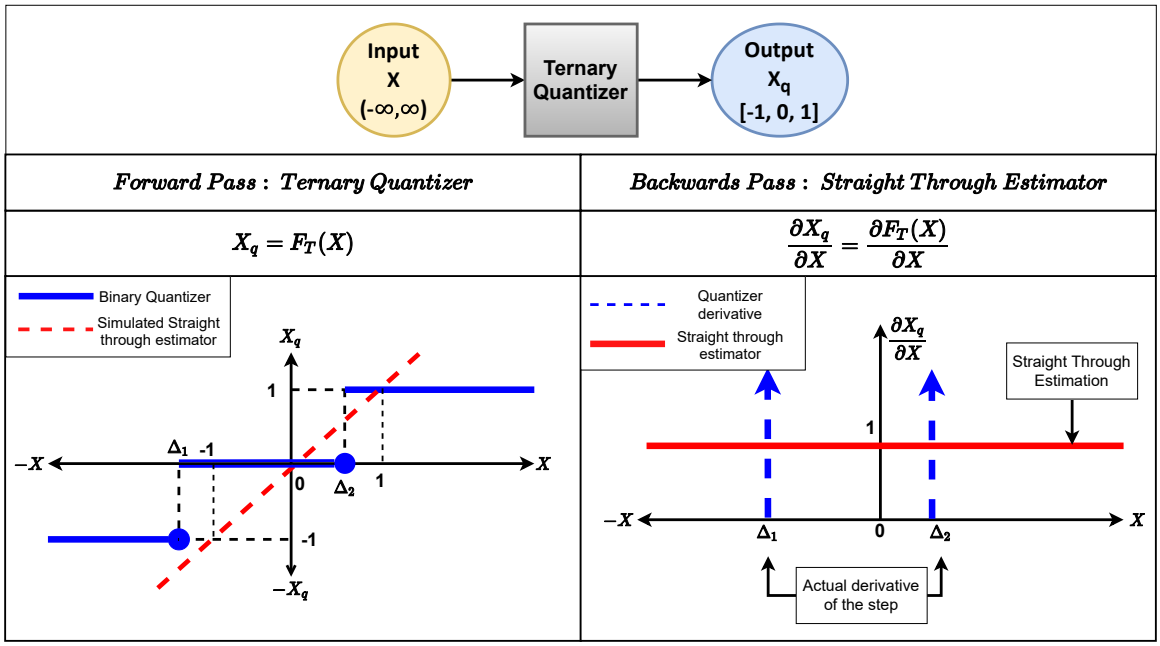


Figure 2.11: Ternary quantizer with STE
This Figure represents the ternary quantizer in the forwards and backwards pass of the CNN training process, shown in blue. In addition, the STE within the forwards and backwards pass is shown in red. During the CNN training process, the quantizer is used in the forward pass and the STE derivative is used in the backwards pass, these are shown by the solid lines.

2.6 Cost of accuracy recovery methods

Accuracy enhancement methods are used to extract better performance from quantized CNNs. These methods are another step towards bridging the accuracy gap from full-precision networks. Each accuracy enhancement method comes at some "cost". The "cost" attached with applying these methods can be broadly classified in these two categories:

1. **Extra "cost" at training time** : Covers methods that add computational elements to the training of the CNN but do not change that architecture of the network.
2. **Extra "cost" at inference time**: Covers methods that add computational elements to the architecture of the CNN.

2.6.1 Extra "cost" at training time

These accuracy recovery methods add extra complexity to the training process of quantized CNNs. These methods do not change the parameters or behavior of the model during inference/deployment. The goal of these methods is to extract as much performance as possible without changing the architecture or structure of the network in the forward pass. The methods that typically fall into this category are:

- **Quantizer enhancement methods:** Increases computational cost of training by either changing the behavior of the quantizer[36, 37] during training or the amount of quantization in the network through the process of training[38].
- **Teacher-Student approaches:** These methods use the knowledge distilled by a larger full-precision network to train a smaller quantized network[39]. This method increases complexity by adding another network in parallel that needs to be trained, as well as more computation elements.
- **Additional loss terms and regularization:** These methods add additional computation elements to the output classification or intermediate results of the network to optimize some aspect of the training.[39, 40, 41]

2.6.2 Extra "cost" at inference time

These accuracy recovery methods add extra parameters to the network or alter the behavior of the model during inference/deployment. The goal of these methods is to augment the performance by changing the architecture or structure of the network. These changes seek to add strategic representational capacity to the model and help it learn and generalize better. The methods that typically fall into this category are:

- **Increasing depth/width/resolution of the CNN architecture:** These methods add capacity to the network. This has a direct and measurable effect on the performance of these networks. This also adds a much larger computational and memory overhead to these methods. The effects of increasing these parameters also may have diminishing return for accuracy improvement and speedup/memory saving within the deployed model[21, 22, 42].
- **Additional computational modules/Architectural changes:** These methods can add additional components like batch normalization, additional scaling factors[36] or changes to the non linearities[20]. These methods increase the computational cost during inference by necessitating a higher number of mixed-precision mathematical operations depending on the architectural change.

Chapter 3

Related Work

Within the research space of quantized CNNs, most quantization methods seek to optimize for model performance and model compression. Literature on higher bit-width quantization (8 and 4 bit) [43, 44, 45], report model performance similar to full precision but have comparatively low model compression[1]. On the other extreme, binary quantization [17, 21, 30, 36, 46] methods report the highest possible model compression ($\sim 32x$) but also the worst model performance due to larger quantization error.

To combat this low model performance, binary quantization works also propose methods to enhance the accuracy of binary networks. These methods broadly involve architectural changes [20, 21, 36, 39] and improvement to training methods [36, 37, 47]. These methods can be collectively referred to as accuracy recovery methods. With these accuracy recovery methods, binary networks highlight the model performance vs. model compression tradeoff. We can judge how well any quantized CNN model perform and compresses using binary methods as a baseline.

Ternary network quantization on the other hand is a comparatively less explored quantization option. Theoretically, ternary methods should provide a higher degree of performance[2] with a lower degree of model compression[3]. This motivates a survey of ternary quantization methods to gauge the improvement in model performance that can be achieved. Additionally, the accuracy recovery methods used in binary works are explored and applied to ternary networks. This gives us an idea of how effectively these methods can enhance performance while not sacrificing model compression.

3.1 Ternary CNN Quantization methods

In this section, the key methods proposing ternary quantization are presented. All of the methods presented below provide some insight into the ternary decision boundary selection process and how it fits into the CNN training procedure. These methods are the primary basis for the completely ternary CNNs benchmarked within this work.

3.1.1 Ternary Weight Networks (TWN)

Ternary Weight Networks[23] laid the foundation for ternary quantization by establishing a formal description of the ternary quantization problem as well as an approximate solution. This approximate solution to ternary weight partitioning was the first of its kind and assumes the weights to be normally distributed.

Formulating the ternary weight quantization problem

The objective of ternary weight quantization is to find the most accurate ternary weight representation for given network architecture and training task. One way of accomplishing this is to minimize the representative distance between the full-precision weights W and the ternary-valued weights W_t along with a non-negative scaling factor α [46]. This is represented within the Eqn. 3.1. α^* and W_t^* represent the optimal scaling factor and ternary weight representation that are the closest to full precision performance.

$$\begin{gathered} \alpha^*, W_t^* = \underset{\alpha . W_t}{\operatorname{argmin}} \|W - \alpha W_t\|_2^2 \\ \text{where } \alpha \geq 0,\ W_t \in \{-1, 0, +1\},\ W \in \mathbb{R} \end{gathered} \tag{3.1}$$

[1] $2\times$ for 16-bit, $4\times$ for 8-bit

[2] Ternary quantization has 3 states compared to binary which has 2 states

[3] Binary compression: 32x, Ternary compression: 16x if 2 bits used per symbol and 20x if encoded 1.6 bit per symbol is applied

Approximate solution for Ternarization

Expanding and solving the optimization problem 3.1 gives us two interdependent equations for the optimal scaling factor α^* (3.4), and optimal ternary weight representation W_t^* (3.3). As no exact solution is possible, a threshold-based ternary function is suggested as an approximate solution. Where Δ^* is the optimal quantization threshold parameter, which helps us obtain the optimal weight distribution W_t^* from the full precision weights W.

$$W_t^* = f(W|\Delta^*) = \begin{cases} +1 & \text{if } W > \Delta^* \\ 0 & \text{if } |W| \leq \Delta^* \\ -1 & \text{if } W < -\Delta^* \end{cases} \tag{3.2}$$

The original problem described in Equation 3.1 can be further solved to obtain a numerical solution for the optimal scaling factor α^* and optimal threshold value Δ^* for a ternary quantization system.

$$\Delta^* = \underset{\Delta>0}{\operatorname{argmax}} \frac{1}{|I_\Delta|} \left(\sum_{i \in I_\Delta} |W_i| \right)^2 \tag{3.3}$$

$$\alpha_\Delta^* = \frac{1}{|I_\Delta|} \sum_{i \in I_\Delta} |W_i| \tag{3.4}$$

Where $I_\Delta = \{i \mid |W_i| > \Delta\}$ and $|I_\Delta|$ denotes the number of elements in I_Δ

Making the assumption that full-precision weights will be normally distributed, an approximate solution for Δ^* can be found in the Eqn. 3.5. Where n is the total number of weights present within the full precision weights W.

$$\Delta^* \approx \frac{0.7}{n} \sum_{i=1}^{n} |W_i| \tag{3.5}$$

The scaled ternary quantization from TWN[23] is represented in Eqn. 3.6. Where W is the full-precision weights, α^* is the positive scaling factor, Δ^* is the symmetric decision boundary and W_t^* are the ternarized weights

$$\alpha^* * W_t^* = \alpha^* * \begin{cases} 1 & \text{if } W > \Delta^* \\ 0 & \text{if } |W| \leq \Delta^* \\ -1 & \text{if } W < -\Delta^* \end{cases} = \begin{cases} \alpha^* & \text{if } W > \Delta^* \\ 0 & \text{if } |W| \leq \Delta^* \\ -\alpha^* & \text{if } W < -\Delta^* \end{cases} \tag{3.6}$$

This allows us to simplify the convolution process for TWN by allowing the scaling factors to be applied after the convolution with the input. Where Y is the convolution output, X is the input feature map, W_t represents the ternarized weights and α is the positive scaling factor.

$$Y = (X) * (\alpha^* W_t^*) = \alpha^* (X * W_t^*) \tag{3.7}$$

The Ternarization procedure can be visualized in the Figure 3.1. The procedure starts on the left with the full-precision weights of the CNN. Then using the Equations 3.5, 3.4 the optimal decision boundaries as well as scaling factors for the ternarization. Combining these factors gives the optimally ternarized weights along with an optimal scaling factor. These are then used within the convolution process.

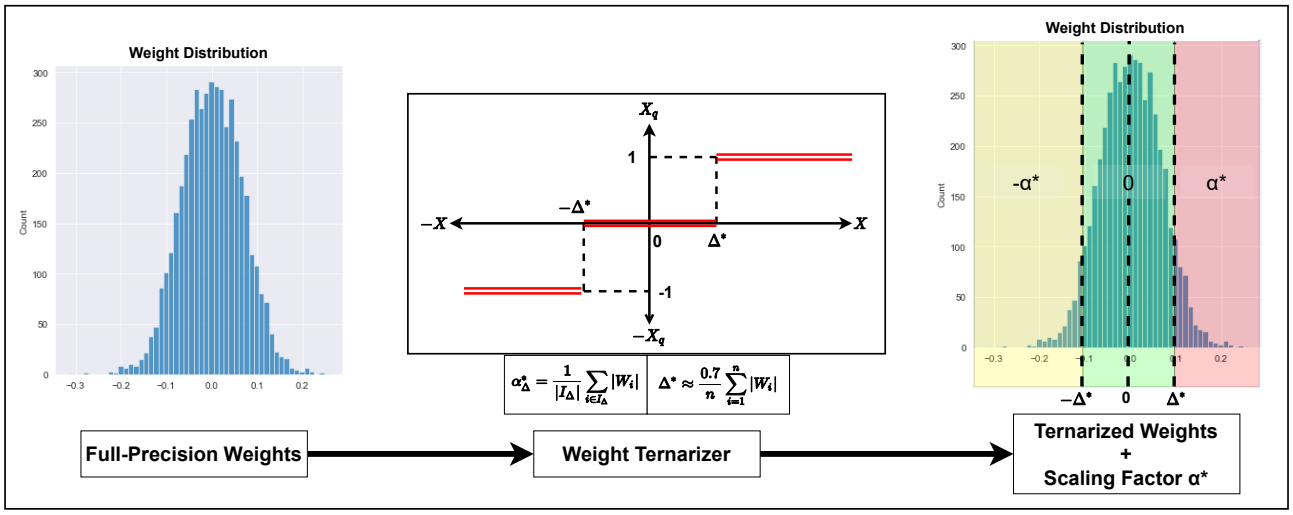


Figure 3.1: Ternary Weight Quantization procedure
This Figure describes the process by which the full-precision weight of a CNN are ternarized using the TWN method[23]. Both the decision boundaries and the scaling factors are calculated based on the weight distribution.

Training of Ternary Weights

The weights can be trained using the process Stochastic Gradient Descent(SGD). The weights are ternarized during the forward pass and during the backward pass a Straight Through Estimator(STE) is used to simulate a differentiable quantizer. Parameter update is performed on the full-precision version of the weights. The convolutions always use ternary weights.

3.1.2 Trained Ternary Quantization(TTQ)

TTQ[1] enhances TWN[23] with the introduction of two learned scaling factors α_{pos} and α_{neg} within each quantizer. This improvement allows for the positive and negative ternary bits to have more mixed precision representation.

The scaled ternary quantization from TTQ[1] is represented in Eqn. 3.8. Where W is the full-precision weights, , α_{pos} and α_{neg} are the two scaling factors, Δ is the symmetric decision boundary and W_t are the ternarized weights

$$\alpha * W_t = \begin{cases} \alpha_{pos} & \text{if } W > \Delta \\ 0 & \text{if } |W| \leq \Delta \\ -\alpha_{neg} & \text{if } W < -\Delta \end{cases} \tag{3.8}$$

Applying this weight scaling is also a more expensive mathematical operation than a single scaling factor. This is because multiple scaling factors cannot be easily separated from the multiply and accumulate operations. Unlike a single common factor in TWN that can be easily separated from the convolution itself, as seen in Eqn. 3.7.

Both α_{pos} and α_{neg} are learned through the training process instead of being calculated like in Eqn. 3.4. In TWN [23] the full-precision weights are directly ternarized using the decision boundaries as seen in Eqn. 3.2. Within TTQ, a normalization is applied to the weights before ternarization as seen in Eqn. 3.11

Ternarization method

The Training procedure in TTQ can be summarized by the following steps:

1. Start with Full precision weights W.
2. Normalize the weights to the range $[-1, 1]$. This is accomplished in the following manner:

$$W_{norm} = \frac{W}{max(|W|)} \tag{3.9}$$

3. Quantize the weights to $\{-1, 0, 1\}$ by using the thresholds $-\Delta, \Delta$. These thresholds are a constant value for CIFAR and ImageNet datasets

$$\Delta = 0.05^{4} \tag{3.10}$$

$$W_t = f(W_{norm}|\Delta) = \begin{cases} +1 & \text{if } W_{norm} > \Delta \\ 0 & \text{if } |W_{norm}| \leq \Delta \\ -1 & \text{if } W_{norm} < -\Delta \end{cases} \tag{3.11}$$

4. Apply scaling factors $\alpha_{pos}, \alpha_{neg}$ to obtain the weights for inference.

$$W'_t = \alpha * W_t = \begin{cases} \alpha_{pos} & \text{if } W_{norm} > \Delta \\ 0 & \text{if } |W_{norm}| \leq \Delta \\ -\alpha_{neg} & \text{if } W_{norm} < -\Delta \end{cases} \tag{3.12}$$

5. Perform inference and obtain a result. This result is used to calculate the loss L. In the case of CNNs this is most often Cross Entropy loss (CE). Y_{pred} is the output of the network ie the prediction of the expected class label. Y this is the ground truth or the expected output label.

$$L = CE(Y, Y_{pred}) \tag{3.13}$$

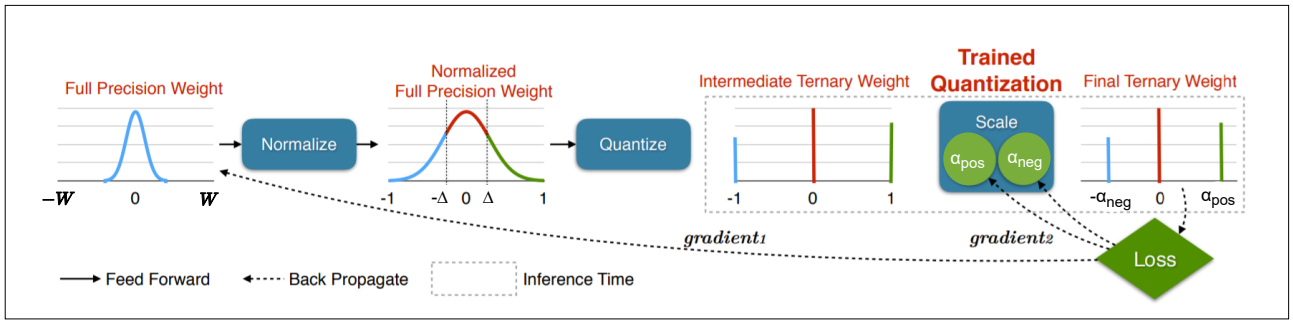


Figure 3.2: TTQ training procedure[1]
The process of training ternary weight alongside the learnable scaling factors is shown within this image. Starting with the full-precision weight and quantizing them as well as how the gradients from the loss get applied to both the weights and scaling factors.

6. The gradients for scaling factors are calculated a bit differently. As stated in [1], the gradients are calculated as shown in the Equations 3.14, 3.15

$$\frac{\partial L}{\partial \alpha_{pos}} = \sum_{i \in I_{pos}} \frac{\partial L}{\partial W_t(i)}, \text{ where } I_{pos} = \{i | W_{norm}(i) > \Delta\} \tag{3.14}$$

$$\frac{\partial L}{\partial \alpha_{neg}} = \sum_{i \in I_{neg}} \frac{\partial L}{\partial W_t(i)}, \text{ where } I_{neg} = \{i | W_{norm}(i) < -\Delta\} \tag{3.15}$$

7. The next step is to calculate the gradients for full precision weights W. This involves the multiplication using a normalization factor as seen in Equation 3.16. The other part of this calculation is the scaled gradient, which can be seen in Equation 3.17

$$\frac{\partial L}{\partial W} = \frac{\partial W_{norm}}{\partial W} \times \frac{\partial L}{\partial W_{norm}} = \frac{1}{max(|W|)} \times \frac{\partial L}{\partial W_{norm}} \tag{3.16}$$

$$\text{Where } \frac{\partial L}{\partial W_{norm}} = \begin{cases} \alpha_{pos} \times \frac{\partial L}{\partial W_t} & \text{if } W_{norm} > \Delta \\ 1 \times \frac{\partial L}{\partial W_t} & \text{if } |W_{norm}| \leq \Delta \\ \alpha_{neg} \times \frac{\partial L}{\partial W_t} & \text{if } W_{norm} < -\Delta \end{cases} \tag{3.17}$$

[4] as specified in [1]

As can be seen above, the scaling factors not only aim to increase model capacity but also serve as learning rate multipliers during back propagation. This ternarization procedure is visualized in Figure 3.3.

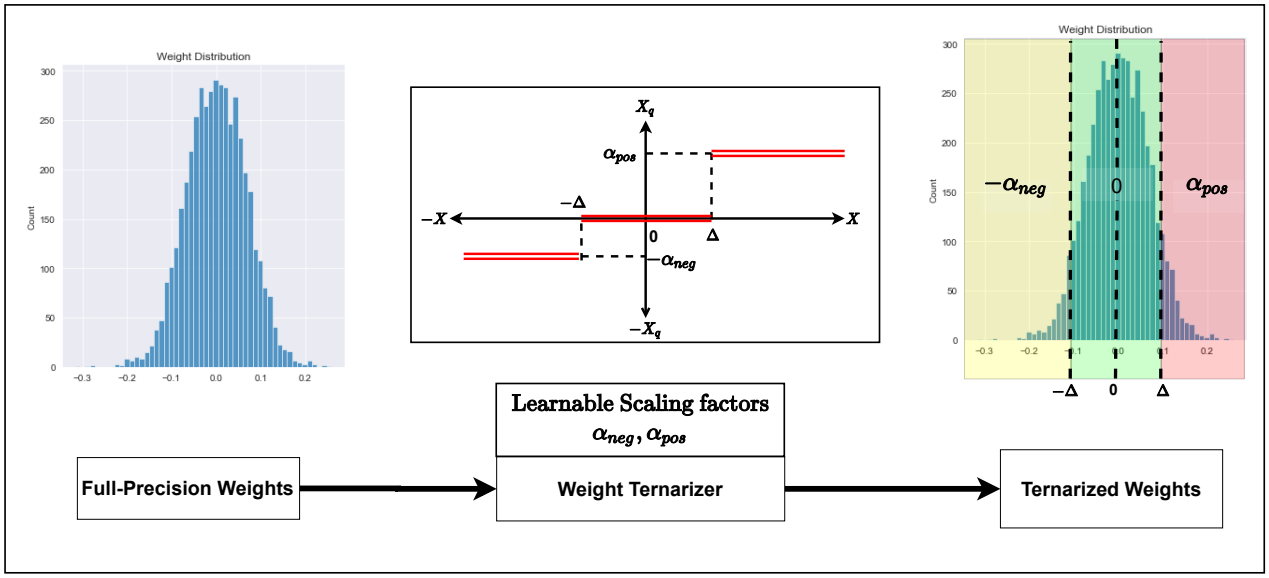


Figure 3.3: Trained Ternary Quantizer
This Figure describes the process by which the full-precision weight of a CNN are ternarized using the TTQ method[1]. The decision boundaries $\pm\Delta$ are calculated based on the weight distribution. The scaling factors α_{pos} and α_{neg} are both learned parameters.

3.1.3 Fine Grain Quantization(FGQ)

The Fine Grain Quantization (FGQ)[48] procedure further enhances ternary weight quantization with the addition of more scaling factors distributed across multiple weight subsets. This is done to improve the expressivity of a set of ternary weights by providing more mixed precision representation through the use of scaling factors.

Fine Grain Quantization Procedure

The key idea in FGQ is to separate the full-precision weights into smaller subsets and calculate "fine-grain" scaling factors for each of these subsets. This subset division is done per convolution layer. In every subset of full-precision weights, we calculate an optimal scaling factor (α) for that specific subset. This enhances the ternary weights W_t with more diversity within scaling factors to improve the mixed precision representation. This FGQ procedure can be represented in the following steps:

1. **Weight Partitioning**
   For a set of convolution layer weights W, we divide them into k unique subsets. The grouping of convolution weights is along the width and height of the convolution filter. Common scaling factors can be calculated for each weight subset.

   The weights of any convolution layer can be viewed as a tensor of $n \times m \times w \times h$ elements, where n is the number of input channels, m is the number of output channels and w, h represent the width and height of the convolution filter. Weights are grouped such that a convolution layer will have $w \times h$ scaling factors. This can be illustrated in the Figure 3.4. Weights with the same color across input and output channels belong in the same subset. Thus, for a 3x3 convolution later, there will be 9 subsets.
   With the weight partitioning complete we need to calculate the scaling factors.

2. **Ternarization of Weight Partitions**
   For a complete set of full-precision weights W, we can divide them into $w \times h$ sets. Each subset can be labelled as W^i, where $\bigcup_{i=1}^{w \times h} W^i = W$. Each subset W^i is used to calculate the scaling factor α^i for that subset. We use all the weights to calculate the ternarization threshold Δ for that convolution layer.

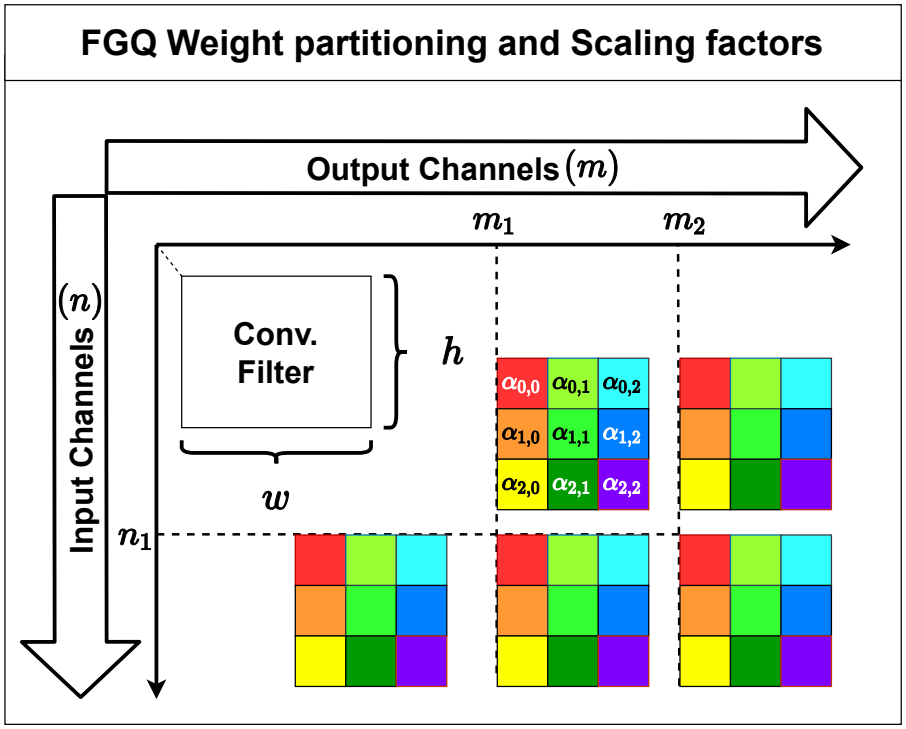


Figure 3.4: Fine grain Quantization Weight partitioning and scaling factors
The grouping is weights into FGQ subsets is illustrated within this image. It can be seen that across Input and output channels, weights present in similar positions within the filter are grouped into the same subset. The subset of the weights is indicated by the color of the block.

$I^i_\Delta = \{j \mid |W^i_j| > \Delta\}$ contain the indices of elements of W^i that are outside of the respective decision boundaries.

The analytical solution for the ternarization of the k sub-problems is:

$$\Delta^* \approx \frac{0.7}{N} \sum_{j=1}^{N} |W_j| \text{ where } N = n \times m \times w \times h \tag{3.18}$$

$$\alpha^i_\Delta = \frac{\left(\sum_{j \in I^i_\Delta} |W^i_j|\right)}{|I^i_\Delta|}, \forall i \in w \times h \tag{3.19}$$

The FGQ process is graphically represented in the Figure 3.5. Within this Figure, an overview of the FGQ process can be seen. Starting with the full-precision weights of a convolution layer and ternarizing the weight distribution. This is also accompanied with calculating the scaling factor matrix for the convolution filters. This results in much finer grain mixed-precision representative capability within the convolution.

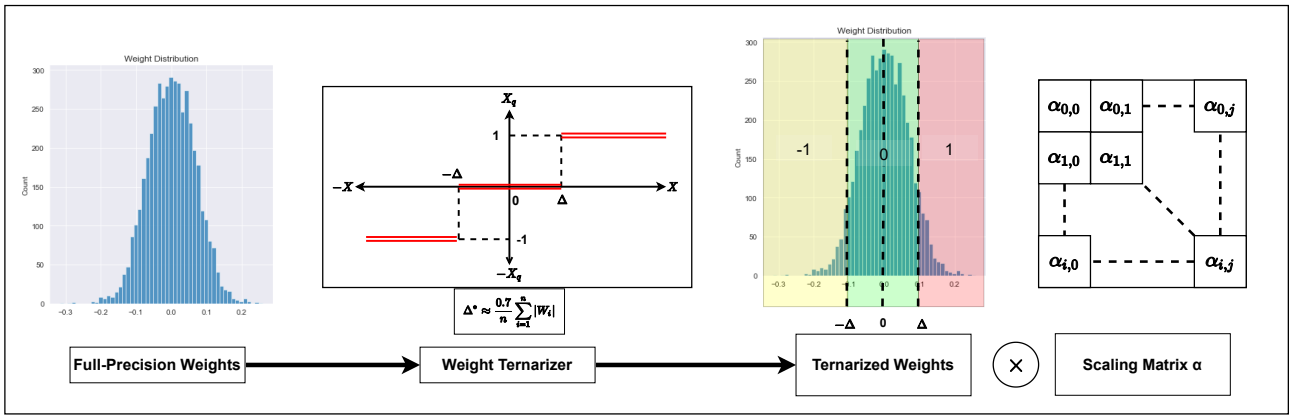


Figure 3.5: Fine Grain Ternary Quantization procedure
This Figure describes the process by which the full-precision weight of a CNN are ternarized using the FGQ method[48]. The decision boundaries $\pm\Delta$ are calculated based on the weight distribution. The scaling factor matrix α has multiple scaling factors calculated from subsets of the weights.

3.1.4 Embarrassingly Simple Approach(ESA)

As the name suggests, this method is centered around training a ternary network from scratch in an embarrassingly easy way[24]. This method differs from those mentioned above, as it uses a weight discretization regularizer(WDR) term in addition to the network's existing loss function for aiding in network ternarization. There is no scaling factor applied to the ternarized weights in this approach.

Weight parameterization

Instead of using full-precision weights directly, this approach first scales them to within the (-1, 1) range. This is accomplished by passing the weights through a $tanh(\ )$ non-linearity before quantization. This is done to pre-distribute the weights closer towards the ternary states of -1, 0 and 1..

The weights used by the network during ternarization are the pre-distributed weights W_{tanh}. These are obtained from the full-precision weights W using the Eqn. 3.20

$$W_{tanh} = tanh(W) \tag{3.20}$$

Weight Discretization term and Objective function

The weight discretization term forces the full precision weights towards ternary representation within the continuous space provided by weight parametrization[5]. For every convolution layer within the network, we can calculate the regularizer using the Equation 3.21. In this Equation, γ acts as a sparsity controller term and $|W|$ represents the number of weights in the convolution layer.

$$R(tanh(W)) = \sum_{j=1}^{|W|} \left[(\gamma - tanh^2(W_j)) \cdot tanh^2(W_j)\right] \tag{3.21}$$

This weight regularization can be visualized in Figure 3.6 for a few different values of γ. This Figure represents the relation between the parametrized weights $tanh(W)$ and the regularizer $R(tanh(W))$ from Equation 3.21.

[5] $tanh(W) \in (-1, 1)$

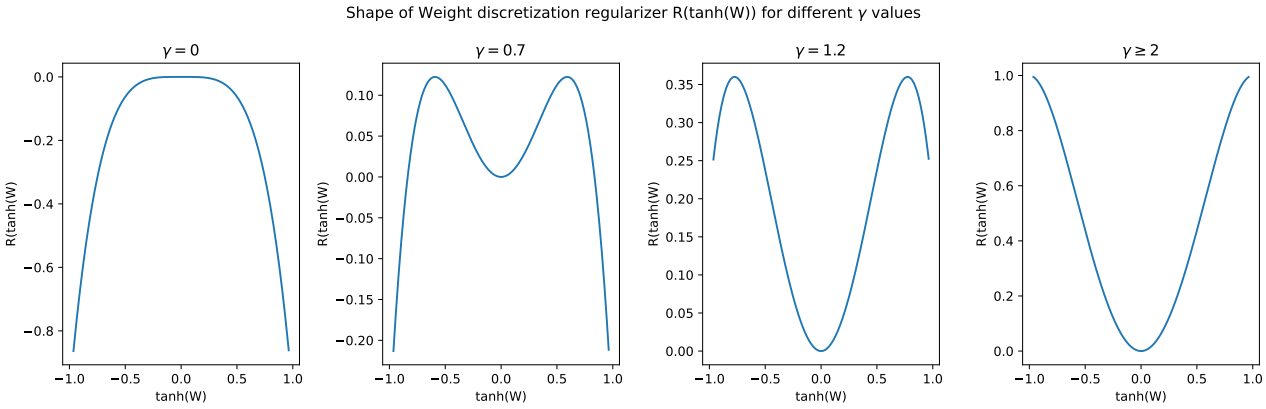


Figure 3.6: Shape of WDR for a set of γ values
Within this Figure, the relationship between the WDR ($R(tanh(W))$) and the parameterized weights $tanh(W)$ can be observed for different values of γ. This WDR term serves to push the weights towards the ternary values of -1, 0 and 1. This can be seen in the decreasing loss slopes when moving towards these ternary values. In the implementation of this method γ of 0.7 is used.

In the Figure 3.6 for a γ value of 0.7, the decreasing loss slopes when moving towards ternary values can be seen most clearly. This shape of the regularizer allows the gradient descent algorithm to optimize this regularizer by pushing weights towards these ternary values. By attaining the lowest regularizer loss, we shape the underlying weights towards ternary representation.

To further obtain the objective function of the entire training process, we add the regularizer to the existing Cross entropy loss function L_{CE}. The regularizer is also calculated for every convolution layer that is quantized, which we can represent as n. An additional loss balancing term λ is used to scale the regularization's influence on the loss.

$$Loss = L_{CE} + \lambda \times \sum_{i}^{n} R(tanh(W_i)) \tag{3.22}$$

We can then propagate this loss through the network and help the network learn a ternary distribution

Training and Ternarization

The training process under the ESA approach can be summarized in the following steps:

1. Specify Sparsity controller γ and Regularization controller λ for the Objective function specified in Equation 3.22. These variables are specified for a given network and dataset pair in [24].
2. Perform training on parameterized weights until Objective function converges. Obtain optimized weights $tanh(W^{opt})$
3. Obtain Ternary weights by rounding to the nearest integer, $W_T = round(tanh(W^{opt}))$

With this incredibly simple approach a ternary network can be trained from scratch while also having extremely competitive network performance.

The inference within the ESA network is performed in the manner illustrated in Figure 3.7

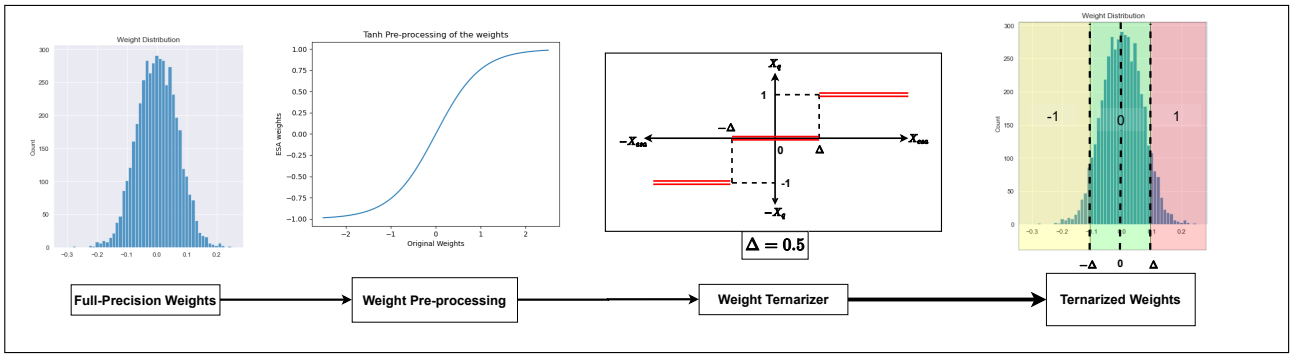


Figure 3.7: Embarrassingly simple approach quantization procedure
This Figure describes the process by which the full-precision weight of a CNN are ternarized using the ESA method[24]. The full-precision weights are always parameterized through a *tanh* non-linearity The decision boundaries $\pm\Delta$ are fixed at ± 0.5 as they represent a simple rounding function. No scaling factors are used in this approach.

3.1.5 Fast and Accurate TNN (FATNN)

FATNN proposes a way to leverage learnable ternary quantizers for both the weights and activations[25]. The quantizers use non-uniform step size quantization for increased expressive ability. The implementation of FATNN's quantizers borrows heavily from the work Learned Step size Quantization(LSQ)[45] and modifies the LSQ algorithm to accommodate non-uniform step-size ternarization.
The FATNN quantizers use two learnable parameters $\{\Delta_1$ and $\Delta_2\}$ as both quantization boundaries and step size widths.

The process of quantizing a variable using an FATNN quantizer is slightly different for both the weights (W) and activations (A) of a network. This is because of a difference in the structure of the quantizer for both of these variables.

FATNN quantization process for weights

Weights typically have $(-\infty, \infty)$ range and their distribution is centered around 0. With these properties in mind, the steps for quantizing weights can be summarized here:

1. Start with full precision weights W
2. Partition the weight distribution into two parts W_1 and W_2. These partitions divide the weights into positive and negative values.
   - $W_1 = W \leq 0$
   - $W_2 = W > 0$
3. The learnable decision boundaries Δ_1 and Δ_2 are used to quantize the respective weight partitions W_1 and W_2. The quantization itself can be seen in Equation 3.23. It can be seen that the ternarization is performed to the symbols $\{-1, 01\}$, the exact values of the symbols are not relevant for accuracy of the network. Any alternate 3 symbol notation can be used to represent this quantization without accuracy loss[6].

$$W_t = \begin{cases} +1 & \text{if } W_2 > \Delta_2 \\ 0 & \text{if } W_2 \leq \Delta_2 \text{ and } W_1 > -\Delta_1 \\ -1 & \text{if } W_1 \leq -\Delta_1 \end{cases} \tag{3.23}$$

The process of FATNN weight quantization can be visualized in the Figure 3.8. It should be noted that due to the distribution properties of weights and how the quantizer is centered around 0, the non-uniform step size quantization does not have an impact on the ternarization.

[6]Ternary $\{0, 1, 2\}$ notation was also tested for the weights, no accuracy loss was found.

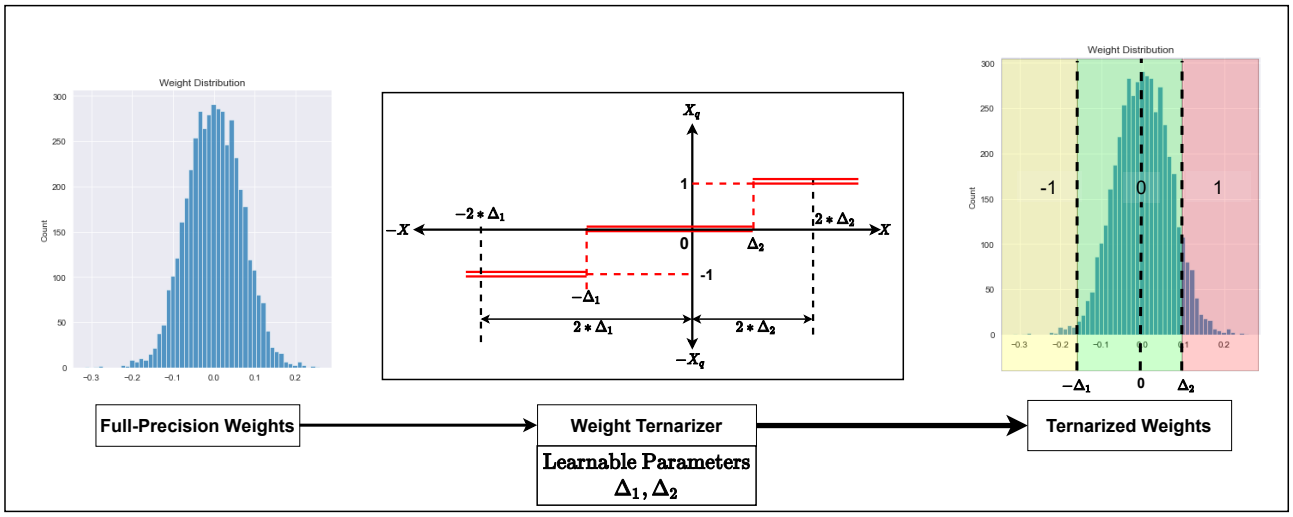


Figure 3.8: FATNN weight quantizer
This Figure describes the process by which the full-precision weight of a CNN are ternarized using the FATNN method[25]. The full-precision weights are quantized by two learnable decision boundaries Δ_1 and Δ_2. No scaling factors are used in this approach.

FATNN quantization process for activations

The distributions of activations of a CNN are very heavily dependent on the non-linearity functions used within the network. If a ReLU is used activations have a [0, ∞) range while if a HardTanh is used the range becomes [-1, 1]. For the purposes of this thesis, all the network architectures used rely on ReLU or ReLU like non linearities. Hence the Possible activation range of [0, ∞) is used. This significantly changes the structure of the quantizer. The steps taken to quantize the activations are summarized below:

1. Start with full precision activations A
2. Partition the activation distribution into two parts A_1 and A_2. These partitions divide the activations into the two quantization boundaries.
   - $A_1 = A \leq 2 * \Delta_1$
   - $A_2 = A > 2 * \Delta_1$
3. The learnable decision boundaries Δ_1 and Δ_2 are used to quantize the respective activation partitions A_1 and A_2. The quantization itself can be seen in Equation 3.24. Within this example the ternary symbols are $\{0, 1, 2\}$, these symbols could be replaced with $\{-1, 0, 1\}$ or any other three symbol representation with no accuracy loss.

$$A_t = \begin{cases} 2 & \text{if } A_2 > \Delta_2 + 2 * \Delta_1 \\ 1 & \text{if } A_2 < \Delta_2 + 2 * \Delta_1 \text{ and } A_1 > \Delta_1 \\ 0 & \text{if } A_1 \leq \Delta_1 \end{cases} \tag{3.24}$$

The process of FATNN activation quantization can be visualized in the Figure 3.9. It should be noted that due to the distribution properties of activations, the non-uniform step size quantization is fully utilized and is the major driver of the ternarization.

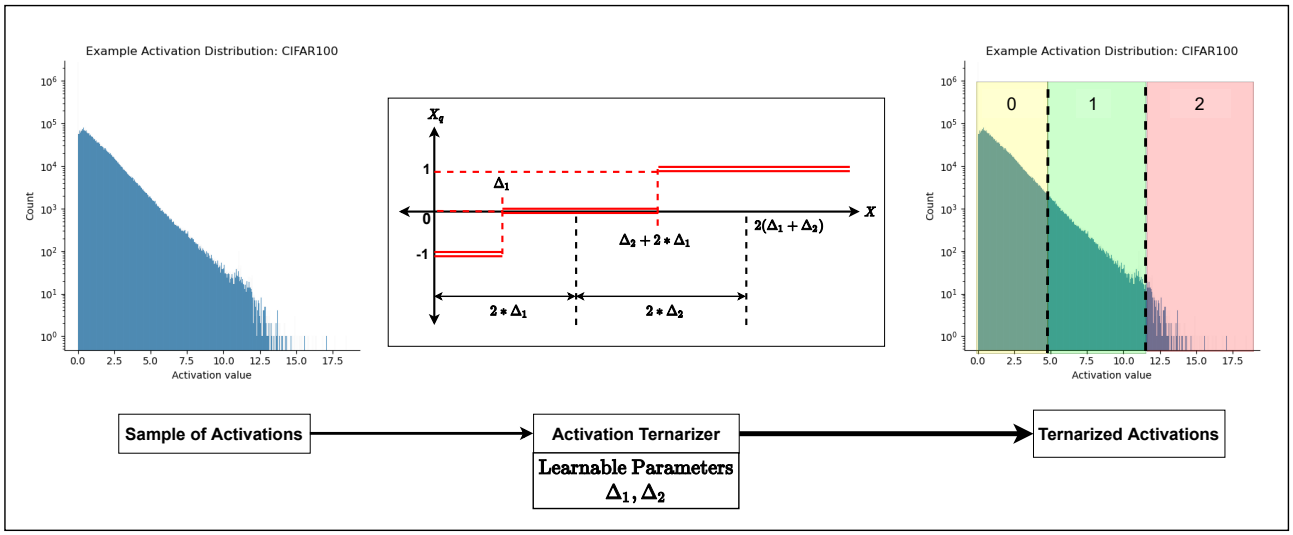


Figure 3.9: FATNN activation quantizer
This Figure describes the process by which the full-precision activations of a CNN are ternarized using the FATNN method[25]. The full-precision activations are quantized by two learnable decision boundaries Δ_1 and Δ_2. No scaling factors are used in this approach.

Combining these learnable parameters with the concepts of learned step-size quantization allows us to create a uniquely customizable quantizers that can learn the optimal decision bounds for ternary representation. These bounds are learned through the process of Stochastic Gradient Descent(SGD) without the need for any special loss terms or change to the learning process.

3.2 Accuracy recovery methods from binary quantization

Binary neural network methods have to contend with a much harsher quantization error as well as information lost within the activations through convolutions. This has prompted a large amount of research into methods that can help these networks to perform better and extract more accuracy from these CNNs. Within the research space, the following methods are found to be used most:

- **Increasing capacity of the model** [20, 21, 22, 39, 49]: In these methods, more parameters are introduced in such a way that it boosts the representative capability of the network.
- **Improve the learning characteristics of the network** [36, 37, 47]: These methods try to improve the performance of binary networks by changing some as part of the training process. These include new loss terms, regularization, straight through estimator enhancements, teacher-student learning or other enhancements. These methods don't add capacity to the model but instead focus on extracting more accuracy/performance in existing capacity.

In this section, the accuracy recovery methods from some binary works are discussed in the ternary context. Some preliminary hypothesis on their expected performance will also be included with each method.

3.2.1 Generalized activation functions

Generalized activation functions were first introduced in ReActNet[20]. They highlight the critical role mixed precision activations play for the representative ability of quantized network. The key idea is to explicitly reshape the distributions of these activations to boost the amount of data and actionable features within the propagated activation.

This is accomplished through two enhancements:

- Learnable shift-based quantizer: **ReAct Sign**
- Learnable shift-based activation function: **ReAct PReLU**

These are visualized in Figure 3.10. Implementing the ReAct Sign in ternary overlaps with the quantization

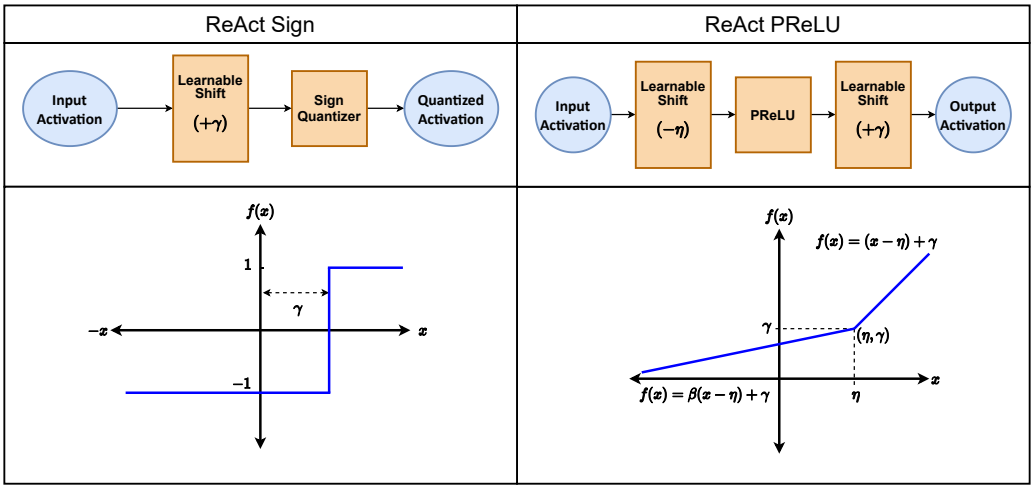


Figure 3.10: Generalized activation functions
Within this Figure, both of the generalized activation functions from [20] can be seen. ReAct Sign: learnable shift-based quantizer is to the left. ReAct PReLU: generalized activation conditioning function is to the right. Both these modules seek to improve quantized CNN components by adding more control over the activation distributions passing through the network.

methods presented in FATNN[25]. Both methods use decision boundaries that learn and shift while the network is training to provide optimal quantization.

The ReAct PReLU serves as a replacement for the existing ReLU like activation functions within the quantized network architecture. It is independent of the quantization scheme and can be broadly applied to any quantized network. The ReAct-PReLU combines two properties that make its presence within a network extremely beneficial:

- Two linear shift factors : η and γ
- A leaky non-linearity : PReLU

This adds 3 parameters per channel, but in return provides much more control over the scaling shifting and scaling of the activation distribution when training the network. A detailed look at the transformation of activation distribution done under the ReAct PReLU can be seen in Figure 3.11.

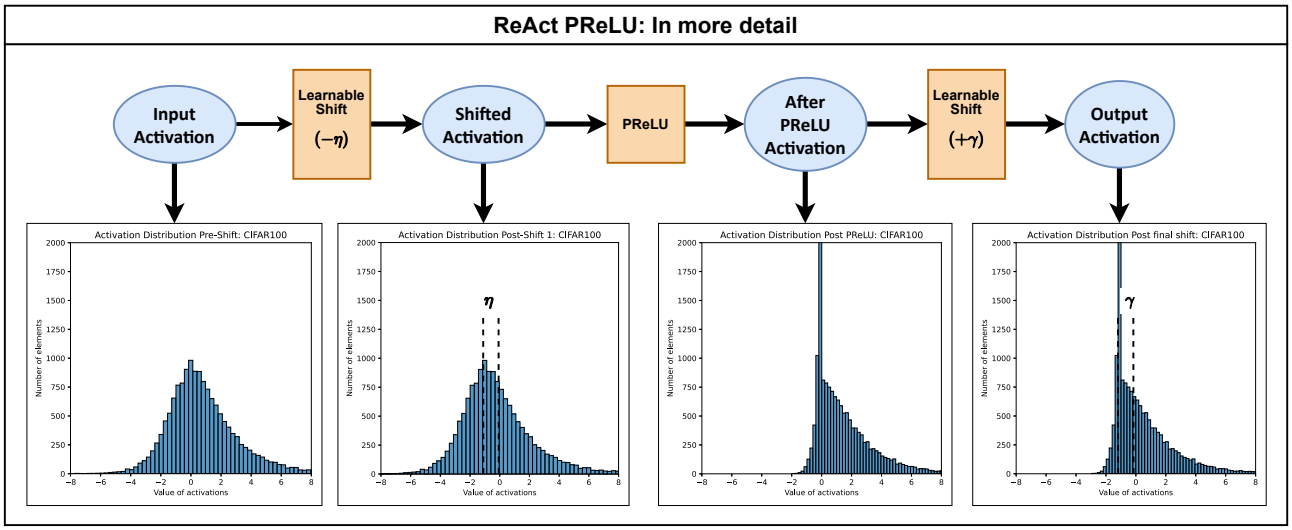


Figure 3.11: ReAct PReLU: in more detail
Within this Figure, the drastic effects a ReAct-PReLU can have on the distribution of the activations can be seen in great detail. An ablation study within Section 5.3.2 goes into detail of which components are the most effective at improving accuracy in the model.

ReActNet reported a +4% improvement in network top-1 accuracy for the ImageNet dataset[50] when using ReAct Sign as well as ReAct PReLU within their CNNs. Within ternary weight networks, we can expect a similar improvement of up to 2-4% when using just ReAct PReLU. ReAct Sign is not included due to the large overlap between FATNN and ReAct Sign quantization method. They both use learnable quantizer bounds to augment the quantizer. In ternary networks with both weights and activations quantized, this improvement is more likely to be in the 1 to 2% range due to the loss of information due from activation quantization.

3.2.2 Approximate derivative of Sign

The approximate derivative to the sign function is specified within the work BI-Real Net[36]. This approx. Sign is a differentiable approximation of the non-differentiable Sign function. We can see from the Figure 3.12, that the approx. Sign is a much closer representation of the Sign quantizer than the STE. This approx. Sign can be defined with a piece-wise linear function, as seen in Equation 3.26. The derivative of this approx. Sign is also seen in the Equation. 3.27.

$$Sign(X) = F(X) = \begin{cases} +1 & \text{if } X \geq 0 \\ -1 & \text{if } X < 0 \end{cases} \tag{3.25}$$

$$Approx\ F(X) = \begin{cases} +1 & \text{if } X \geq 1 \\ 2X + 2X^2 & \text{if } 0 \leq X < 1 \\ 2X - 2X^2 & \text{if } -1 \leq X < 0 \\ -1 & \text{if } X < -1 \end{cases} \tag{3.26}$$

$$Approx\ \frac{\partial F(X)}{\partial X} = \begin{cases} 0 & \text{if } X > 1 \text{ and } X < -1 \\ 2 + 2X & \text{if } 0 \leq X < 1 \\ 2 - 2X & \text{if } -1 \leq X < 0 \end{cases} \tag{3.27}$$

With a little modification we can apply the same idea to a ternary quantizer. A generic ternary quantizer can be

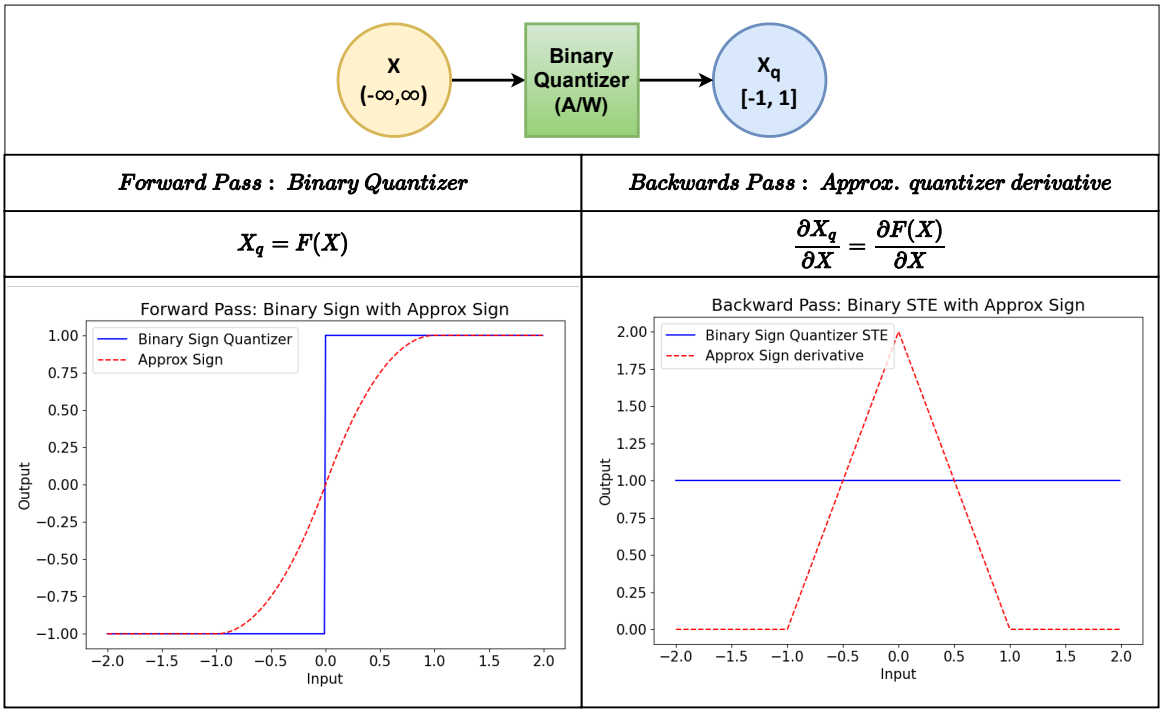


Figure 3.12: Binary Sign and Approx Sign
This Figure represents the binary quantizer in the forwards and backwards pass of the CNN training process, shown in blue. In addition, the approx. Sign within the forwards and backwards pass is shown in red.

expressed in the Equations 3.28. Using the Approx sign from Equations 3.26, a ternary version can be derived. This approximate representation of the ternary quantizer takes the form of Equations 3.29.

$$Ternary\_Quantizer(X, \Delta_1, \Delta_2) = F_T(X) = \begin{cases} +1 & \text{if } X > \Delta_2 \\ 0 & \text{if } \Delta_1 < X \leq \Delta_2 \\ -1 & \text{if } X \leq \Delta_1 \end{cases} \tag{3.28}$$

$$Approx\ F_T(X) = \begin{cases} +1 & \text{if } X \geq \Delta_2 + 0.5 \\ 2X_2^2 & \text{if } \Delta_2 \leq X < (\Delta_2 + 0.5) \\ 4X_2 - 2X_2^2 - 1 & \text{if } (\Delta_2 - 0.5) \leq X < \Delta_2 \\ 2X_1^2 - 1 & \text{if } \Delta_1 \leq X < (\Delta_1 + 0.5) \\ 4X_1 - 2X_1^2 - 2 & \text{if } (\Delta_1 - 0.5) \leq X < \Delta_1 \\ -1 & \text{if } X < (\Delta_1 - 0.5) \end{cases}, \text{ where } \begin{array}{l} X_1 = X - \Delta_1 + 0.5 \\ X_2 = X - \Delta_2 + 0.5 \end{array} \tag{3.29}$$

In the backward pass over the ternary quantizer, the following Approximate derivative of the quantizer is expected.

$$Approx\ \frac{\partial F_T(X)}{\partial X} = \begin{cases} 4X_2 & \text{if if } (\Delta_2 - 0.5) \leq X < \Delta_2 \\ 4 - 4X_2 & \text{if if } \Delta_2 \leq X < (\Delta_2 + 0.5) \\ 4X_1 & \text{if } (\Delta_1 - 0.5) \leq X < \Delta_1 \\ 4 - 4X_1 & \text{if } \Delta_1 \leq X < (\Delta_1 + 0.5) \\ 0 & \text{Otherwise} \end{cases}, \text{ where } \begin{array}{l} X_1 = X - \Delta_1 + 0.5 \\ X_2 = X - \Delta_2 + 0.5 \end{array} \tag{3.30}$$

The forwards and backward pass of the existing ternary quantizer can be visualized in the Figure 3.13. Applying this STE enhancement to the network provides an additional transform to the gradients as they pass through the approx. Sign. This should help the network will converge better, as the training process now also accounts for the derivative of the quantizer when training.

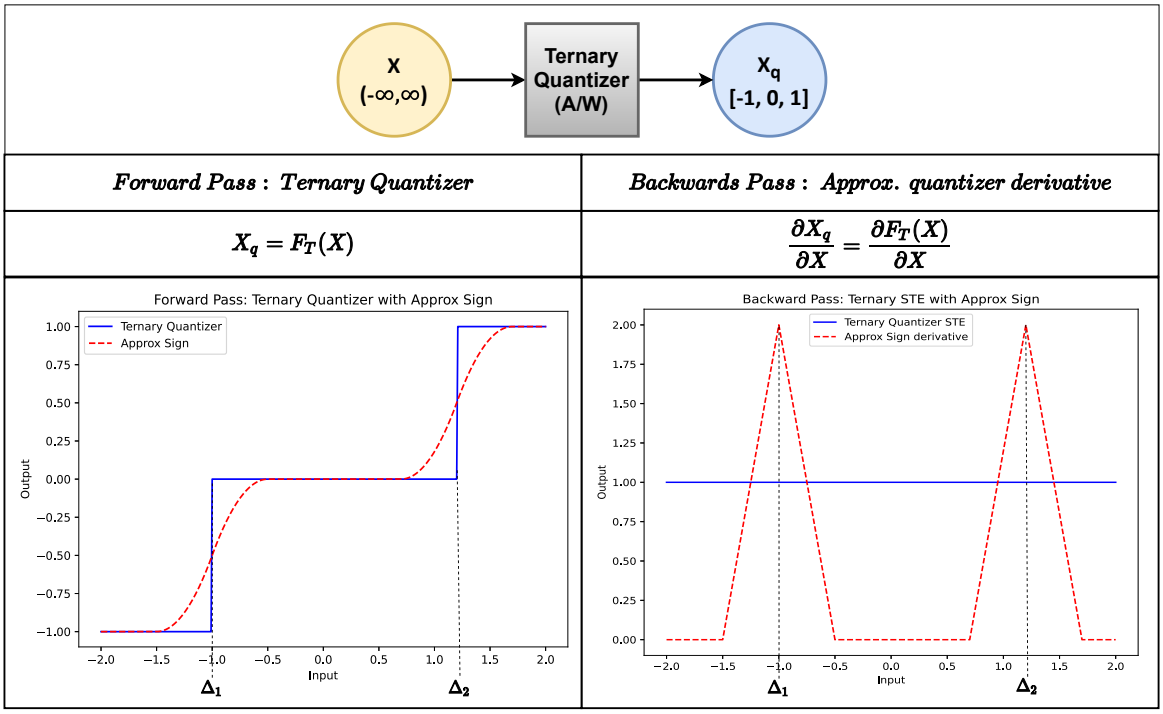


Figure 3.13: Ternary Quantizer and Ternary Approx Sign
This Figure represents the ternary quantizer in the forwards and backwards pass of the CNN training process, shown in blue. In addition, the approx. Sign within the forwards and backwards pass is shown in red.

3.2.3 Progressively Hardening Quantizer

Previous work on incrementally quantizing a network by partitioning and increasing the number of quantized weights[38] has shown an increase in network performance. This motivates the use of progressivlely ternarizing a network. The Prog. Hardening approach focuses its efforts on progressively increasing the amount of quantization for the weights and activations in the network experience. This process pushes towards more discrete representations as the training goes on [37]. This is achieved through a control factor λ within the Equations

3.32 that describe the progressively hardening quantizer.

Within the binary case, the Equation 3.31 represents a binary quantizer. While, Equation 3.32 are used to simulate the progressively hardening quantizer.

$$Sign(X) = F(X) = \begin{cases} +1 & \text{if } X \geq 0 \\ -1 & \text{if } X < 0 \end{cases} \tag{3.31}$$

$$Progressive\ F(X) = tanh(\lambda X)\ \text{as}\ Sign(X) \approx \lim_{\lambda \to \infty} tanh(\lambda X) \tag{3.32}$$

$$Progressive\ \frac{\partial F(X)}{\partial X} = \lambda(1 - tanh^2(\lambda X)) \tag{3.33}$$

The Equations 3.31, 3.32, 3.33, are visualized in the Figure 3.14.It can be seen that the backwards pass for the progressively hardening quantizer serves as a good approximation of the non-differentiable impulse response seen in Figure 2.11. Something to keep in mind for larger λ factors is the value of $max(\frac{\partial F(X)}{\partial X})$. This value increases exponentially with the λ. In the backwards pass, these large peaks can lead to extremely high gradients for certain weights that can significantly hurt training attempts. Within this thesis, the value of γ is increased 0 to 9 over the training epochs. The value of γ is increased in steps of $+1$ every 15 training epochs.

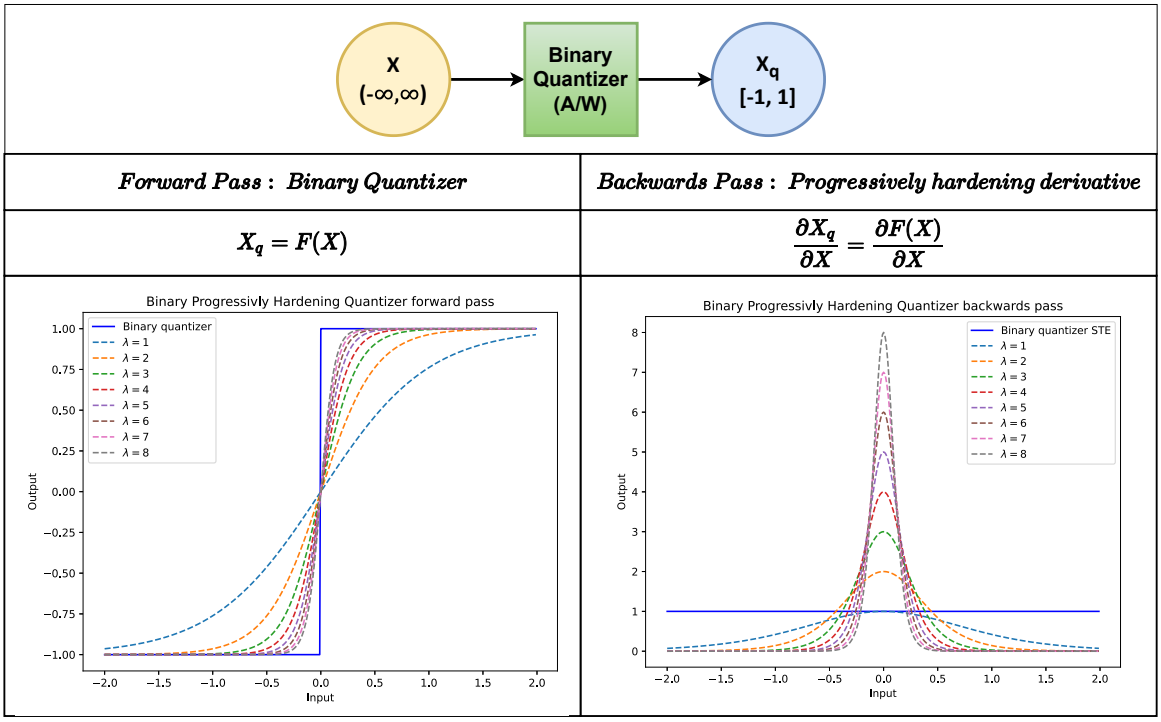


Figure 3.14: Binary Sign and Progressively hardening quantizer
This Figure represents the binary quantizer in the forwards and backwards pass of the CNN training process, shown in blue. In addition, the progressively hardening quantizer within the forwards and backwards pass is shown in dashed lines of various colors.

In the ternary context, we can represent the basic quantizer with the Equation 3.34. The progressively hardening quantizer is represented with the Equation 3.35.

$$Ternary\_Quantizer(X, \Delta_1, \Delta_2) = F_T(X) = \begin{cases} +1 & \text{if } X > \Delta_2 \\ 0 & \text{if } \Delta_1 < X \leq \Delta_2 \\ -1 & \text{if } X \leq \Delta_1 \end{cases} \tag{3.34}$$

$$Progressive\ F_T(X) = \left(\frac{\exp(\lambda X_1)}{1 + \exp(\lambda X_1)}\right) + \left(\frac{\exp(\lambda X_2)}{1 + \exp(\lambda X_2)}\right) - 1, \text{ where } \begin{matrix} X_1 = X - \Delta_1 \\ X_2 = X - \Delta_2 \end{matrix} \tag{3.35}$$

In the backward pass over the ternary quantizer, the progressive hardening takes on the familiar double peaked form that we can see in the Equation 3.36.

$$Progressive\ \frac{\partial F_T(X)}{\partial X} = \left(\frac{\lambda \exp(\lambda X_1)}{(1+\exp(\lambda X_1))^2}\right) + \left(\frac{\lambda \exp(\lambda X_2)}{(1+\exp(\lambda X_2))^2}\right), \text{ where } \begin{matrix} X_1 = X - \Delta_1 + 0.5 \\ X_2 = X - \Delta_2 + 0.5 \end{matrix} \tag{3.36}$$

The Equations 3.35 and 3.36 are visualized in the Figure 3.15. The progressively hardening quantizer within the ternary context is described in this Figure.

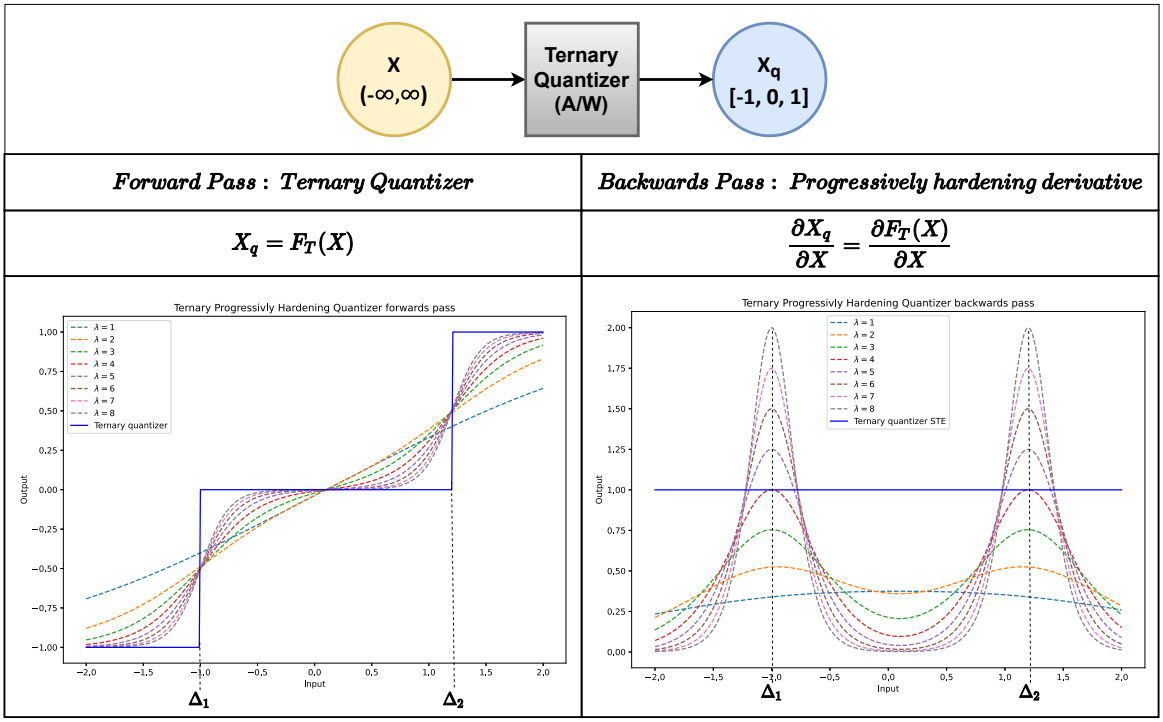


Figure 3.15: Ternary quantizer and Progressively hardening quantizer
This Figure represents the ternary quantizer in the forwards and backwards pass of the CNN training process, shown in blue. In addition, the progressively hardening quantizer within the forwards and backwards pass is shown in dashed lines of various colors.

Applying these methods to a Ternary CNN should lead to an increase in the performance of the network. This is likely caused by the backward pass slowly becoming a better and better approximation of the original derivative of the quantizer. In addition to this, due to the shape of the progressively hardening quantizer, weights closer to the decision boundaries gain an increase in their mobility due to the large multiplicative factor for those weight gradients. This is also likely to help in the better settling of weights into their most effective quantized symbol.

3.2.4 Element wise gradient scaling (EWGS)

Element wise gradient scaling seeks to enhance the quantization of a network by modifying the gradients in such a way that they account for the difference between the quantized ternary symbol and full-precision value of the weight being quantized[47]. This approach serves as an enhancement to the STE. When training a network with quantized weights, the gradients generated are always with respect to the quantized value of the weight. This approach is still quite effective, but does not account for the quantization error between the real valued weights and quantized weight representations used during training.

The EWGS approach leverages the following variables during back propagation to scale the gradients in such a way that quantization error is minimized:

- **The quantization error experienced by a weight**: This is expressed as the difference between the quantized variable W_q and the real valued variable W. This can be represented as $QE = W - W_q$
- The value and direction of the gradient calculated for the quantized weight W_q. This is represented as $\mathcal{G}_{W_Q}$.

- The scaling factor δ. This factor dictates how much of the calculated EWGS scaling is to be applied to the network.

The EWGS can be mathematically represented by the Equations 3.37.

$$\mathcal{G}_W = \mathcal{G}_{W_q}\left(1 + \delta Sign(\mathcal{G}_{W_q})(QE)\right) \tag{3.37}$$

The process of applying EWGS to a quantized CNN is visualized in the Figure 3.16. This method serves as an STE enhancement and does not affect the training or functioning of the CNN in any way except applying a multiplicative scale to the gradients of weights.

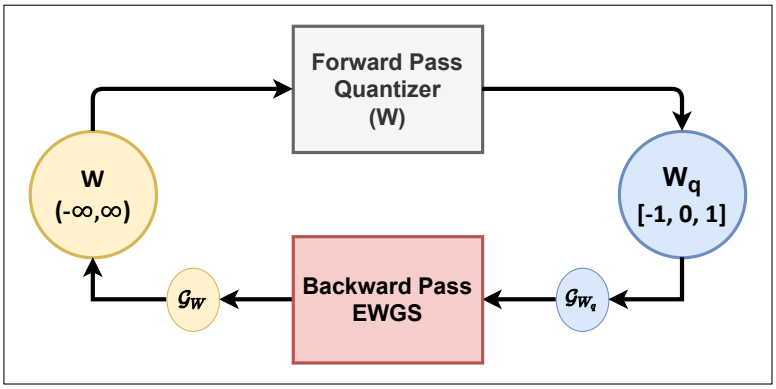


Figure 3.16: Element Wise Gradient Scaling working alongside the quantizer
The EWGS components are only active within the backwards pass of the training process. The scaling of the gradients is the only change made by the EWGS module.

The interplay between the quantization error and the gradients for the discrete weights can be generalized within the Figure 3.17. This allows us to establish an intuition for how EWGS operates. The key idea being that EWGS helps weights converge with their currently quantized value and obstructs a weight from moving further away from it's quantized state, This is seen in Figure 3.17. The Figure can be understood based on two factors

- The quantization error: QE
- The gradient of the quantized weight: $\mathcal{G}_{W_q}$

The relation between these factors is summarized as follows: If QE is positive: the value of the real weigh exceeds the quantized value, $W > W_q$. If the gradient of the quantized weight is negative, the real value moves towards the quantized value. This is a desirable outcome and is encouraged with a gradient scaling factor greater than 1. If the gradient is positive. The real value moves further from the quantized value. This is not desirable and thus, the gradient is obstructed by lowering the gradient scaling factor less than 1.

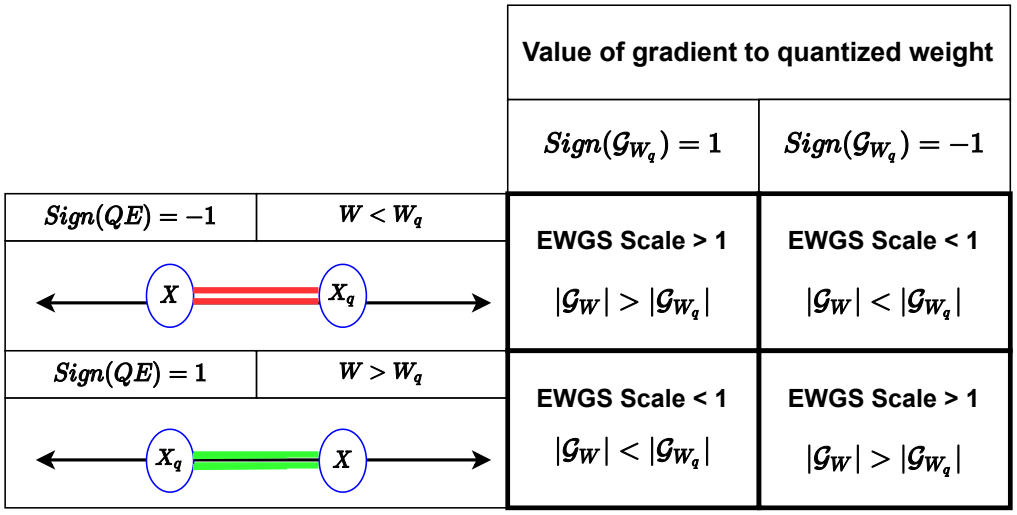


Figure 3.17: Intuition for Element Wise Gradient Scaling factor value
This grid represents how the gradient will be scaled when it passes through EWGS. Depending on the quantization error and original gradient direction, a scale is applied to the gradient.

The behavior seen within EWGS may have mixed effects on network performance, as it increases the mobility of weights converging with their quantized value but it also provides extra inertia to weights that are moving away from their quantized value. In both cases, the gradients over multiple training steps will still lead to a meaningful change and may allow the network to settle on better values.

Chapter 4

Method

The experimental steps taken to understand the behavior of ternary quantized CNNs involve testing multiple CNN architectures on multiple datasets while actively ternarizing their weights and activations. The exact mechanism of how this is achieved is discussed in the following sections:

1. **Section 4.1** goes into details about the experimentation and benchmarking process. The goal being to create a battery of experiments that help gain insight into the performance and reproducibility of ternary quantization and accuracy recovery methods.
2. **Section 4.2** seeks to establish the global context of the experiments and specify the exact methods to be explored. This establishes the bounds of the design space for this work. The structure of the experiments used to explore this design space is also presented.

The objective of this section is to establish a set of experiments as well as a method for executing those experiments such that a clear picture of how ternary quantization methods perform within CNN models can be established. The primary metric used to judge these models is the test accuracy after training is completed.

4.1 Experiments

The structure of the experiments, as well as the CNN architectures, used to explore the design space of this work are specified in this section. Within these experiments, we compare and contrast the different ternary quantization methods as well accuracy recovery methods from binary applied to ternary networks. This is done to gauge the effectiveness of their ternary quantization and accuracy recovery methods in a manner that conveys their generalizability and reproducibility across multiple CNN architectures and datasets.

4.1.1 CNN architectures

Within this battery of experiments, the methods under study are applied to two CNN architectures: ResNet[5] and EfficientNet[42]. These architectures are chosen because of their widespread use and relatively lower parameter count while still retaining high performance. Our goal is to present quantized CNNs that can be deployed on edge devices/accelerators. Selecting lightweight and expressive networks aligns well with the goal of deployment on the edge.

Unified global CNN skeleton

For the purpose of standardization, a unified skeleton architecture was implemented. This architecture represents a generic CNN and consists of a collection of "Convolution blocks". These convolution blocks are substituted with the relevant ResNet or EfficientNet convolution blocks during the training/inference time to represent those architectures. A representation of the unified architecture is shown in Figure 4.1.

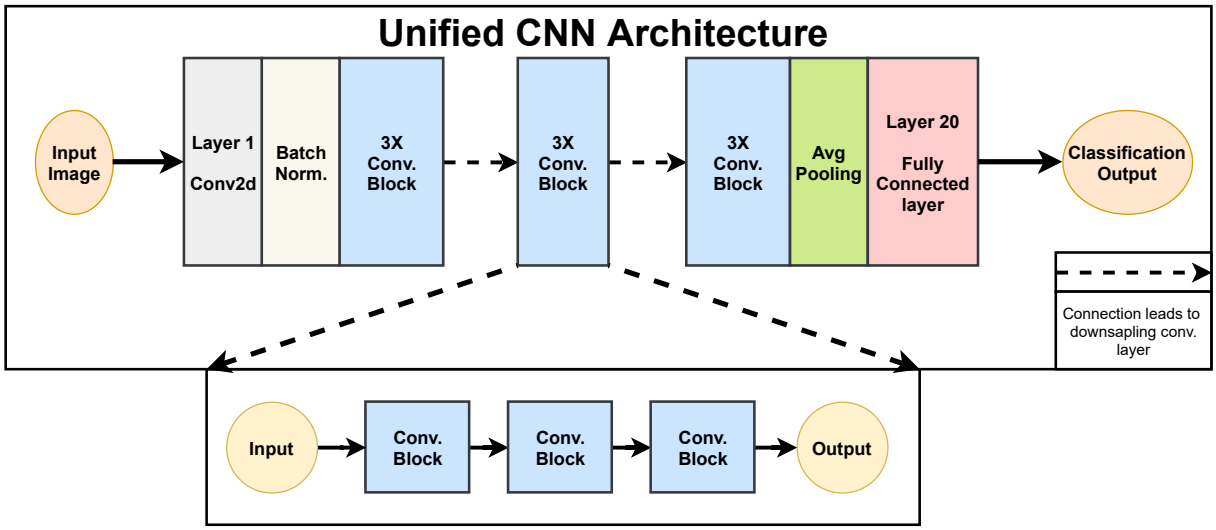


Figure 4.1: Unified CNN architecture skeleton

Within this Figure, the common CNN architectural skeleton is presented. This skeleton consists of a full-precision 3x3 convolution layer followed by 9 convolution blocks and concludes with a full-precision fully connected layer. The first and last layer are kept in full-precision within this thesis. The 9 convolution blocks can be populated with the appropriate quantized CNN blocks when training to get a quantized CNN.

ResNet architectural block

The ResNet[5] architecture is one of the most popular CNN architectures in the world. It has very high expressive capability, coupled with a much lower parameter count than similar performing networks. The architectural block that represents the ResNet contains $2\times$ $3x3$ kernel convolutions and a residual connection. The basic block of this architecture is seen in Figure 4.2. When combined with the unified CNN skeleton, the obtained network is that of a ResNet-20 with 270k total parameters.

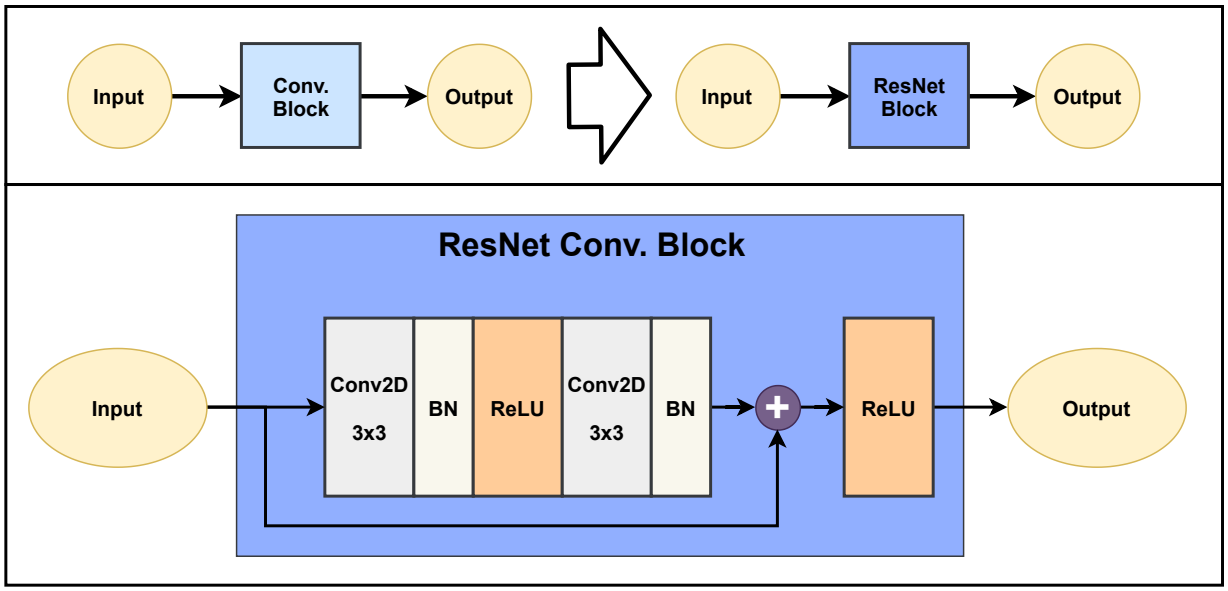


Figure 4.2: ResNet Convolution block

The ResNet convolution block is consists of 2 3x3 convolution layers followed by Batch Normalization and ReLU activation functions. The block also includes a residual connection to further enhance the performance.

EfficientNet architectural block

The EfficientNet[42] architecture is an extension of the MobileNet V2[51] architecture. The focus of EfficientNet is to allow the parameterized scaling of networks (width, depth, number of channels) to provide the lightest and most accurate CNNs. The advantage of being a MobileNet derivative is that these networks have even less parameters than ResNet but still perform to a very high degree. The advantage of including such a network in this quantization study is twofold:

1. It allows easy extension of further experiments by increasing depth, width, resolution of the network.
2. It allows the research of how the number of convolution parameters affect the network performance under quantization.

Thus, the EfficientNet architecture serves as another great comparison point to the ResNet and can further inform us of the quantized behavior of ternary networks. The EfficientNet convolution block contains 3 convolutions. The initial 1×1 convolution that serves as a channel expansion layer. This is followed by a 3×3 depth-wise convolution and a 1×1 point-wise convolution. A representation of the EfficientNet architectural block can be seen in Figure 4.3.

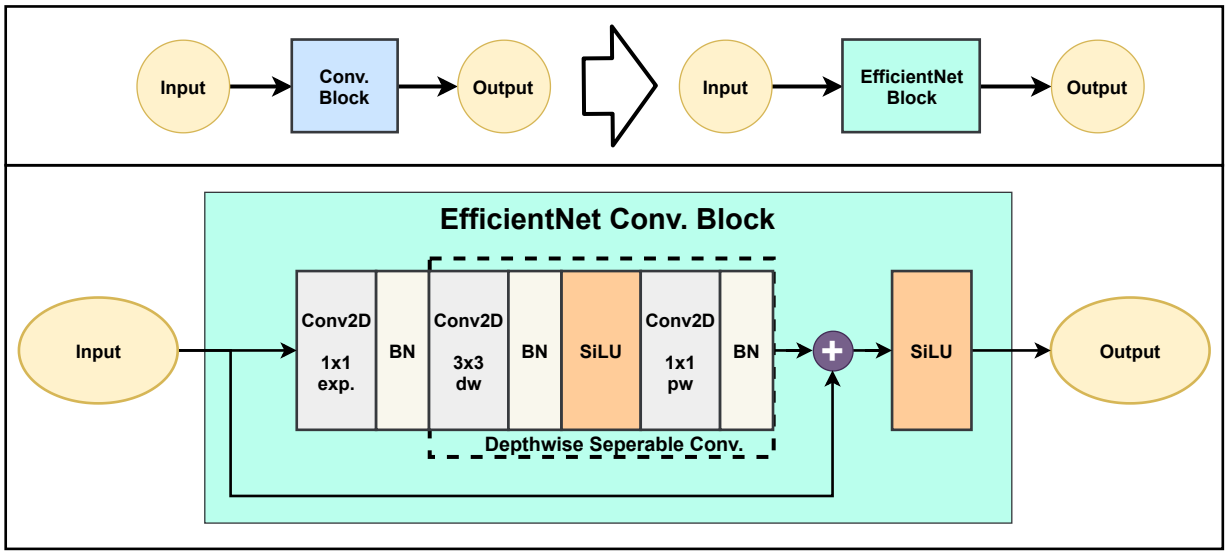


Figure 4.3: EfficientNet Convolution block
The EfficientNet convolution block is consists of 3 individual convolutions. The first is a 1x1 convolution that serves to expand the number of channels within the input. The following two convolution layers form a depth wise separable convolution. A 3x3 depth wise convolution is followed by a 1x1 point wise convolution. These convolution blocks are also followed by Batch Normalization and Swish(SiLU) non-linearity.

4.1.2 Datasets

The datasets used within this set of experiments are CIFAR-10 and CIFAR-100 [52]. These datasets were chosen because of their universal acceptance as introductory learning problems to establish the effectiveness of the CNN quantization. These experiments allows us to judge the effectiveness of ternary quantization methods. These datasets while being much smaller than ImageNet [50], but still serve as a decent challenge to gauge the generalization potential of ternary quantization methods. The characterisitic of the CIFAR datasets used are specified in Tables 4.1 and 4.2.

Input image size	Total images	Training images	Test images
32x32x3	60,000	50,000	10,000

Table 4.1: Common CIFAR characteristics

Dataset name	Number of classes	Total images per class	Training images per class	Test images per class
CIFAR-10	10	6000	5000	1000
CIFAR-100	100	600	500	100

Table 4.2: CIFAR-10/100 characteristics

4.2 Design Space

The methods discussed in Chapter 4 include both ternary quantization works and accuracy recovery methods from binary CNN works. The goal of this work is to perform an ablation study on the different ternary quantization methods as well as relevant accuracy enhancement methods. The final design space of this work are summarized in Table 4.3.

Ternary Quantization methods		Quantizer Enhancements	Architectural changes	Datasets	CNN Architectures
Weight Quantization	Activation Quantization				
Ternary Weight Network	Fast and Accurate TNN	Straight Through Estimator	ReAct PReLU	CIFAR-10	ResNet
Trained Ternary Quantization		Approximate Sign Quantizer		CIFAR-100	EfficientNet
Fine-Grained Quantiation		Progressively Hardening Quantizer			
Embarassingly Simple Approach		Element wise gradient scaling			

Table 4.3: Design space of this work

4.2.1 Design space exploration (DSE)

A brute force approach is used to explore the design space presented above. This is done so that we can ascertain the effectiveness of multiple methods as well as all their combinations and use that extensive experimental data to establish trends and generalized behavior.

Structure of experiments

In this DSE, each combination of methods specified in Table 4.3 are bench-marked on both CIFAR-10/100 as well as ResNet and EfficientNet architectures. Multiple iterations of the same experiment are performed with unique seeds. This allows us to compare any combination of methods, datasets and architectures.
With this setup, we can standardize our experiments and increase our chances of observing both macro scale behavior (across multiple datasets and methods) as well a micro-scale behaviors (seen only in specific situations or specific to certain methods) of ternary quantized networks. The hyperparameter settings used across all experiments are seen in Table 4.4. These hyperparameter settings are common across all experiments and datasets.

CNN Initialization	**Training Epochs**	**Learning rate (LR) scheduler**	**Optimizer**	**Other hyperparameter settings**
Kaiming Init.	200 Warm up epochs: 2	Cosine LR scheduler Initial lr: 0.1 Final lr: 0	Stochastic gradient descent (SGD)	Momentum: 0.9 Weight decay: 1e-4

Table 4.4: Hyperparameter settings for CNN training

Chapter 5

Results and Discussion

The results of exploring the design space from Section 4.2 are presented and analyzed. In some cases, additional experiments are performed to further strengthen the observations made or further establish observed trends. The results of this design space exploration are divided into sections that highlight the core lessons learned from the relevant subset of experiments. These subsets are:

- **Full precision CNN baselines:** The full precision CNNs trained within this section set the baseline for further experiments within this work.
- **Ternary quantized CNNs:** The ternary quantized networks within this section are used to establish the performance characteristics of ternary quantization, including completely ternary networks.
- **Accuracy recovery methods from Binary Quantization:** The various methods to boost the accuracy of binary CNNs are used to enhance ternary CNNs in this section. Various experiments are also performed to verify the working of these methods and motivate further accuracy recovery for ternary CNNs.

In all the experiments that will be presented, the average accuracy and standard deviation across multiple experiments is presented. This is done to provide a realistic expectation of CNN performance. In addition, reporting the statistical behavior of the network performance supports the reproducibility of this work.

5.1 Full precision CNN baselines

The full-precision baselines presented here serve as a starting point for our experiments. They act as an unmodified maximum performance indicator and help judge how much quantization affects the performance of a network and how much accuracy can be recovered using various improvement methods. As stated in section 4.1 we test 2 different CNN architectures: ResNet and EfficientNet as well as two datasets: CIFAR10 and CIFAR100.

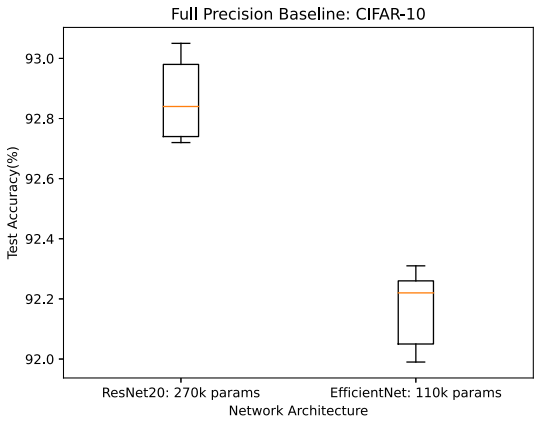


Figure 5.1: Full precision baseline on CIFAR-10

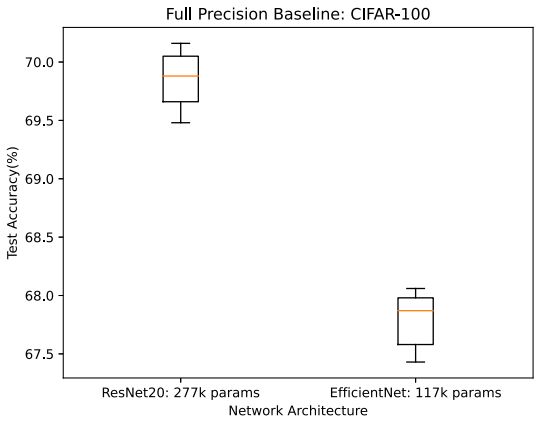


Figure 5.2: Full precision baseline on CIFAR-100

From the Table 5.1, we can see that the EfficientNet despite having less than half the parameters of ResNet is still able to achieve a comparable performance. With this baseline established, we move to quantized versions of these networks. The reported average accuracy for ResNet-20 and CIFAR-10/100 are both in line with those reported in the source material.

Dataset	Network Architecture	Avg. Test Accuracy(%)	Standard deviation
CIFAR-10	ResNet-20	92.86	0.0013
	EfficientNet	92.16	0.0012
CIFAR-100	ResNet-20	69.84	0.0024
	EfficientNet	67.78	0.0024

Table 5.1: Summary of full precision baseline results

5.2 Ternary quantized CNNs

The various types of ternary quantization are explored in this section. As stated in section 2.2, we can quantize the weights, activations or both to obtain three differently quantized CNNs. The three key experiments that will be performed are:

- **Ternary weight networks:** In these networks, only the weights are quantized. These experiments are used to verify the implementation of ternary weight methods with their source material.
- **Ternary activation networks:** In these networks, only the activations are quantized. These experiments are primarily to judge the relative effects of activation quantization on the CNNs.
- **Completely ternary CNNs:** In these networks, both the weights and activations are quantized. these networks are the core interest of this work and can theoretically provide the highest possible memory and speedup benefit within ternary quantization.

5.2.1 Ternary weight networks

Ternary weight methods are those where the weights of the network are quantized to ternary symbols. The most commonly used ternary symbols are $\{-1, 0, 1\}$ but any 3 symbol representation can be used. Ternary quantization is a much harder problem to solve than binary quantization, this is due to the added complexity of two decision boundaries that need to be optimally placed. The somewhat predictable distribution of weights within a CNN makes this problem a bit easier to theorize and attempt to solve, this warranting the large research focus.

The results of CNNs with weight quantization methods discussed in Section 3.1 are summarized in the following figures. The results for CIFAR-10 dataset are shown in Figures 5.3 and 5.4. The results for CIFAR-100 are shown in Figures 5.5 and 5.6. The results for all quantization methods are in line with those presented in their respective source materials.

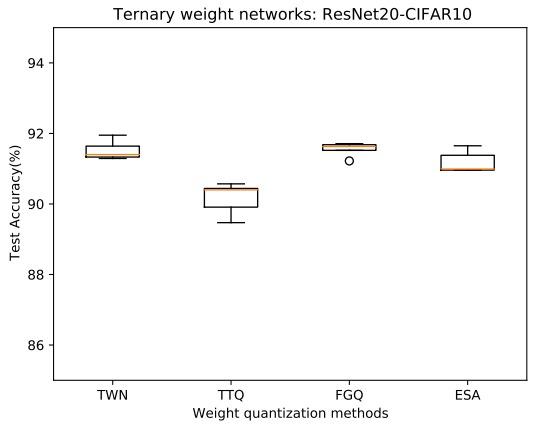


Figure 5.3: Ternary weight ResNet20 on CIFAR-10

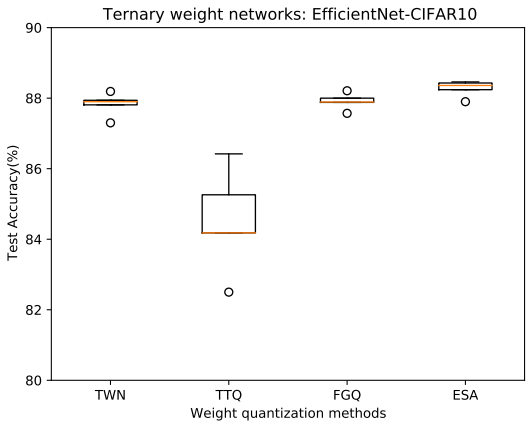


Figure 5.4: Ternary weight EfficientNet on CIFAR-10

The effects of weight ternarization of ResNet and EfficientNet CNNs on the CIFAR-10 datasets are summarized in these figures.

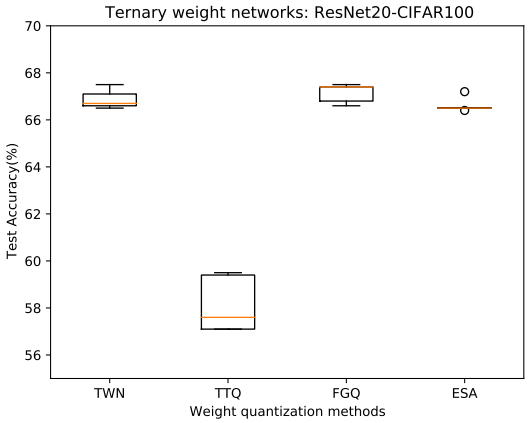


Figure 5.5: Ternary weight ResNet20 on CIFAR-100

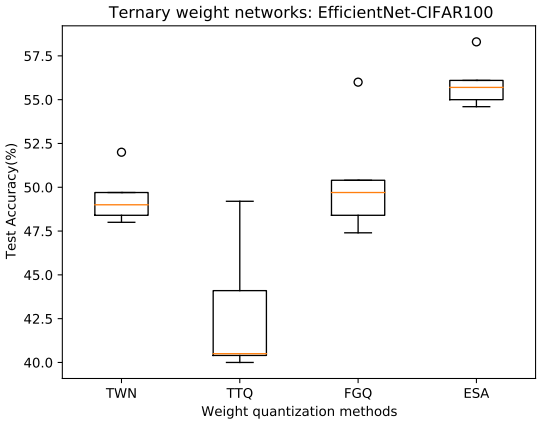


Figure 5.6: Ternary weight EfficientNet on CIFAR-100

The effects of weight ternarization of ResNet and EfficientNet CNNs on the CIFAR-100 dataset are summarized in these figures.

A summary of the ternary weight CNN experiment results can be found in Tables 5.2 and 5.3. These tables present the average accuracy and standard deviation of the ternary weight methods across many experiments.

ResNet-20			
Dataset	**Weight quantization method**	**Avg. Test Accuracy(%)**	**Standard deviation**
CIFAR-10	TWN	91.522	0.00246
	TTQ	90.082	0.003592
	FGQ	91.52	0.001756
	ESA	91.06	0.001611
CIFAR-100	TWN	66.88	0.003709
	TTQ	58.02	0.034383
	FGQ	67.14	0.003666
	ESA	66.48	0.0004

Table 5.2: Summary of ternary weight method results: ResNet

EfficientNet			
Dataset	**Weight quantization method**	**Avg. Test Accuracy(%)**	**Standard deviation**
CIFAR-10	TWN	87.828	0.002925
	TTQ	84.122	0.00902
	FGQ	87.854	0.002427
	ESA	88.184	0.002196
CIFAR-100	TWN	49.42	0.014119
	TTQ	43.48	0.046995
	FGQ	48.76	0.011218
	ESA	55.46	0.005678

Table 5.3: Summary of ternary weight method results: EfficientNet

A summary of the best case accuracy degradation suffered by ternary weight methods can be seen in Table 5.4.

Dataset	Network Architecture	Acc. degradation from Baseline
CIFAR-10	ResNet-20	∼2%
	EfficientNet	∼4%
CIFAR-100	ResNet-20	∼4%
	EfficientNet	∼16%

Table 5.4: Accuracy degradation of the best ternary weight method

The ternary weight quantization methods for a well parameterized network like ResNet suffer from 2% worse performance on CIFAR10 and 4% worse performance on CIFAR 100 than the full precision ResNet. The much lighter and less parametrized, EfficientNet network performs 4% worse on CIFAR-10 than full precision. In contrast, EfficientNet performs 16% worse than full-precision on the harder CIFAR-100 dataset. This indicates that a lower number of total parameters can quickly become insufficient for the learning task when they are quantized.

5.2.2 Ternary activation networks

Ternary activation methods are those within which the activation or feature maps of the CNN are quantized to a 3 symbol form. The most commonly used symbols are $[-1, 0, 1]$ or $[0, 1, 2]$. In all of these experiments The activations of a CNN follow no predictable distribution unlike weights. Every input to a CNN can have a completely unique distribution. This means the quantizer has to generalize over a much more varied and unpredictable set of variables. The quantizer also has to ensure enough information is preserved so that inference can still be performed. Thus, within this section only one work is presented: FATNN[25].
The results of the experiments on CIFAR-10 are presented in Figure 5.7. The results of the experiments on CIFAR-100 are presented in Figure 5.8.

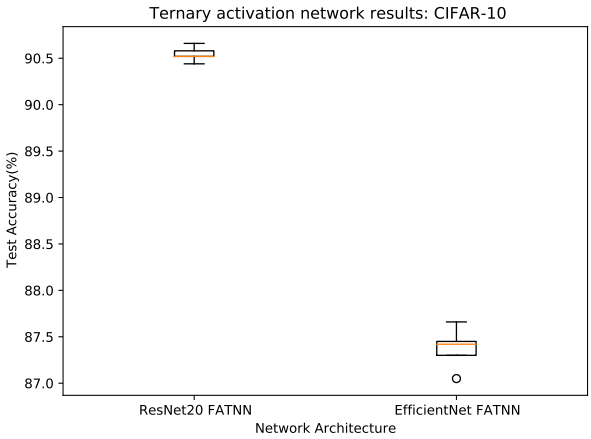


Figure 5.7: Ternary activation network: CIFAR-10

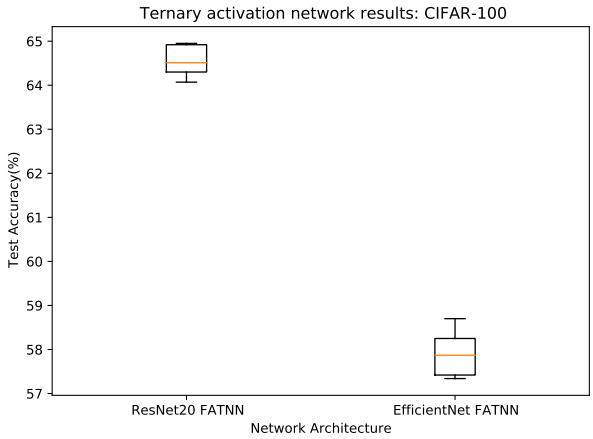


Figure 5.8: Ternary activation network: CIFAR-100

The effects of activation ternarization of ResNet and EfficientNet CNNs on the CIFAR-10 and 100 datasets are summarized in these figures.

The results above can be summarized in the Table 5.5. This Table also includes the accuracy degradation from full-precision baseline.

Dataset	Network architecture	Avg. Test accuracy	Standard deviation	Accuracy degradation from Baseline
CIFAR-10	ResNet-20	90.494	0.001084	∼2%
	EfficientNet	87.408	0.001981	∼5%
CIFAR-100	ResNet-20	64.496	0.002945	∼3%
	EfficientNet	57.662	0.003479	∼10%

Table 5.5: Summary of Ternary activation method results

Similar to results seen within ternary weight experiments (Table 5.4), the well parametrized ResNet networks experiences much less accuracy degradation. 2% for CINFAR-10 and 3% for CIFAR-100. Similarly, the less parametrized EfficientNet shows a much steeper accuracy degradation with increasing complexity of learning task. The EfficientNet performs 5% worse on CIFAR-10 and 10% worse for CIFAR-100, when compared to its full-precision version. With the use of FATNN's quantizer which is completely learnable, we see quite good performance compared to ternary weight experiments in Table 5.4. This motivates the deployment of completely ternary networks combining the ternary weight and activation methods.

5.2.3 Completely ternary CNNs

Completely ternary neural networks (TNN) quantize both activation and weights from full-precision to 3 symbols. In all the fully ternary networks within the design space, weights are quantized to $[-1, 0, 1]$ and activations are quantized to $[0, 1, 2]$. Any other symbols can be used but these are the most popular in literature. This section brings together the ternary weight and activation methods tested in Sections 5.2.1 and 5.2.2 combines them. This leads to completely ternary convolutions and should provide the highest inference time model compression and open the doors to using bit-wise operations and low/mixed precision MACs to accelerate the inference on these networks [25, 28, 32].

The results of experiments for TNNs on CIFAR-10 can be found in Figure 5.9 and 5.10. The results of experiments for TNNs on CIFAR-100 can be found in Figure 5.11 and 5.12.

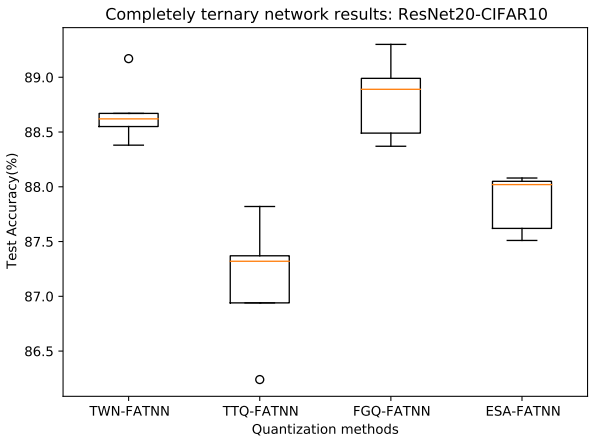


Figure 5.9: TNN ResNet on CIFAR-10

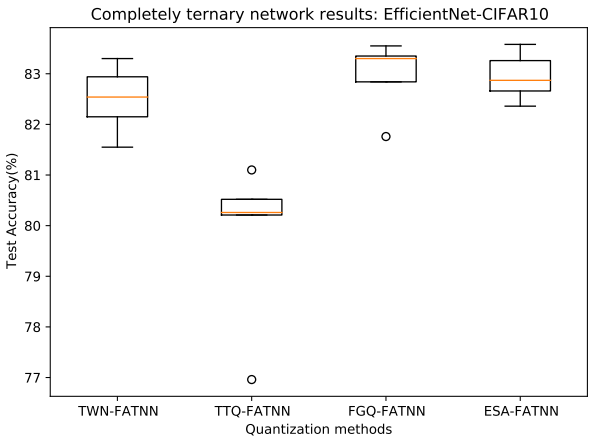


Figure 5.10: TNN EfficientNet on CIFAR-10

The effects of complete ternarization of ResNet and EfficientNet CNNs on the CIFAR-10 dataset are summarized in these figures.

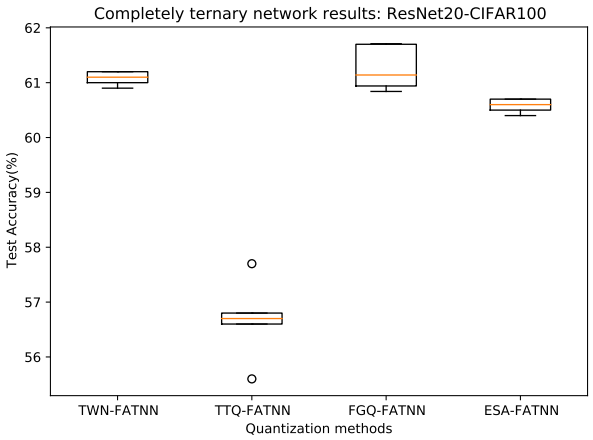


Figure 5.11: TNN ResNet on CIFAR-100

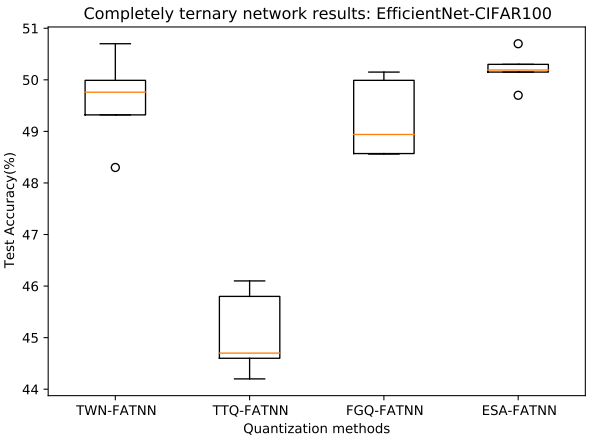


Figure 5.12: TNN EfficientNet on CIFAR-100

The effects of complete ternarization of ResNet and EfficientNet CNNs on the CIFAR-100 dataset are summarized in these figures.

These results of the experiments presented above are summarized in the Tables 5.6 and 5.7.

ResNet20				
Dataset	**Weight quantization method**	**Activation quantization method**	**Avg. Test Accuracy(%)**	**Standard deviation**
CIFAR-10	TWN	FATNN	88.632	0.00182
	TTQ	FATNN	86.992	0.004
	FGQ	FATNN	88.7	0.0023
	ESA	FATNN	87.76	0.0022
CIFAR-100	TWN	FATNN	61	0.0026
	TTQ	FATNN	56.7	0.0063
	FGQ	FATNN	61.2	0.0036
	ESA	FATNN	60.42	0.0033

Table 5.6: Summary of TNN method results: ResNet

EfficientNet				
Dataset	**Weight quantization method**	**Activation quantization method**	**Avg. Test Accuracy(%)**	**Standard deviation**
CIFAR-10	TWN	FATNN	82.63	0.0068
	TTQ	FATNN	79.73	0.014
	FGQ	FATNN	82.66	0.0075
	ESA	FATNN	82.95	0.0043
CIFAR-100	TWN	FATNN	49.55	0.0071
	TTQ	FATNN	44.76	0.0073
	FGQ	FATNN	49.46	0.0061
	ESA	FATNN	50.13	0.0033

Table 5.7: Summary of TNN method results: EfficientNet

Based on the average performance of the networks when in fUll-precision and when completely quantized to ternary, the accuracy degradation can be summarized in Table 5.8.

Dataset	**Network Architecture**	**Acc. degradation from Baseline**
CIFAR-10	ResNet-20	∼4%
	EfficientNet	∼9%
CIFAR-100	ResNet-20	∼8%
	EfficientNet	∼18%

Table 5.8: Accuracy degradation of best TNN methods

Following the same trend seen in ternary weight and ternary activation experiments, completely ternary ResNet20 experience a 4% and 8% degradation in performance from full precision for CIFAR-10 and CIFAR-100. In the same vein, completely ternary EfficientNet suffers a 9% degradation from full precision for CIFAR-10 and a staggering 18% degradation from full-precision for CIFAR-100.

Across all methods, TTQ as a weight quantizer under-performs quite severely. Often performing 4 to 5% worse than other methods. This is primarily due to the way TTQ's quantization bounds are calculated. The decision bounds are a constant factor of the maximum weight value. The ternarization bounds do not account for the distribution of the weights while training. TTQ is not a competitive quantization method[1], Thus TTQ is not considered for the accuracy recovery experiments within the design space.

[1] As currently implemented, TTQ is not as performant as other methods.

5.2.4 Ternary quantizers as weight regularizers

The learning process employed by CNNs relies on making small changes to the weights, which eventually push them into values that provide the lowest loss and high model accuracy/representation. Within the context of ternary quantization, the weights are represented by just three states. The state of a variable is decided by the quantizer using the decision boundaries (Δ_1 and Δ_2). Unless a weight crosses one of these decision boundaries, it cannot change its quantized representation. This quantizer exerts and implicit force on the weight parameters, they must move close to the decision boundaries to be able to change with value. This motivates the optimizer to move weights close to the decision boundaries forming a bi-modal distribution. This process is referred to as the regularization effect of the ternary quantizer. This process is labelled as a regularization because weights far away from the decision bounds are not contributing to the learning process. The SGD optimizer is penalized for these weights by observing no change in loss when moving these weights away from the decision boundaries.

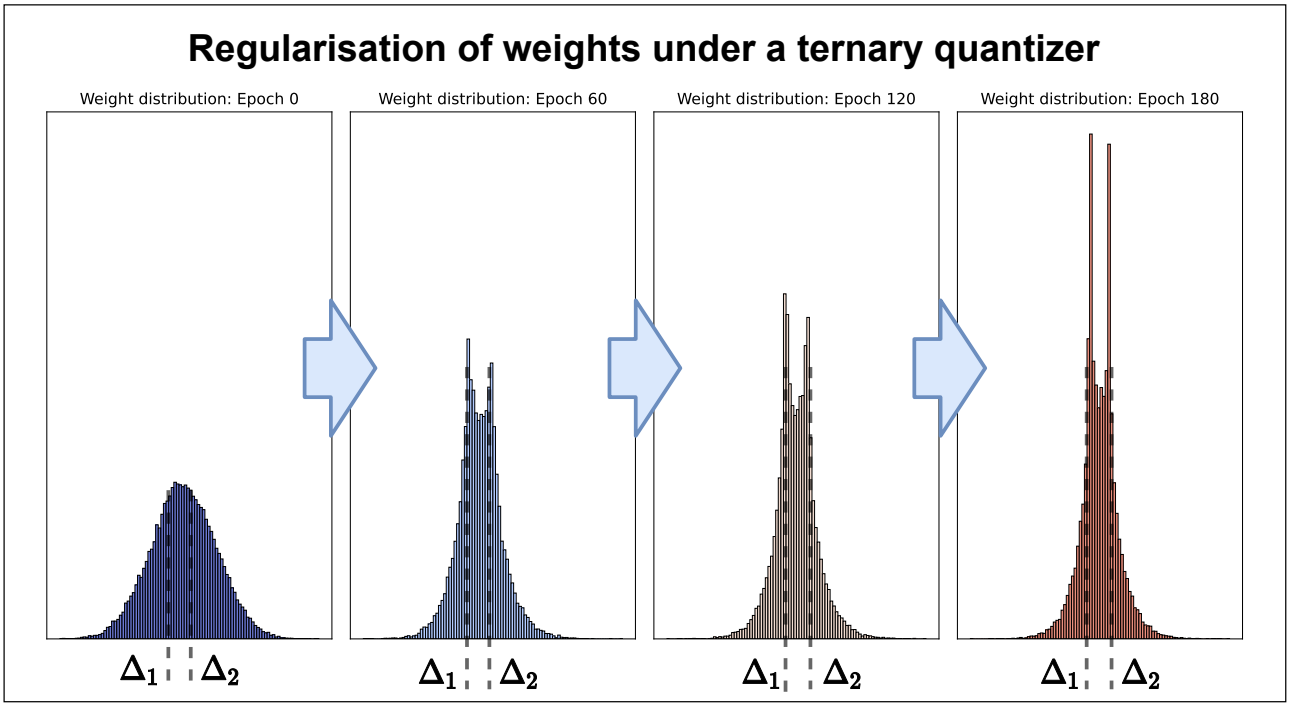


Figure 5.13: Regularization of weights under a ternary quantizer
The regularization effect of the quantizer is clearly seen, as the weights present in this convolution layer migrate towards the decision bounds and form a bimodal distribution through the process of training the network.

The regularization effect of the ternary quantizer can be visualized in Figure 5.13. The presence of a quantizer being applied to the weights adds inertia to the system, where making a change in variable state requires crossing a decision boundary. As seen in Figure 5.13, weights that are crossing decision boundaries undergo a change in state and thus contribute to a differnt loss value. If weights do not cross decision boundaries their contribution to the loss remains constant. Thus, weights appear to migrate towards decision boundaries and actively move across them. This allows the optimizer to find the right quantized value for each weight and the system can converge on a generalized solution.

The migration of weights to decision boundaries is seen in Figure 5.15. This figure shows a subset of a convolution layers weights converging on a quantizer bound through the training process. An average of 60% of the convolution weights will settle to a value within 10% of the decision boundary. This motivates the strength of the regularization effect even further.

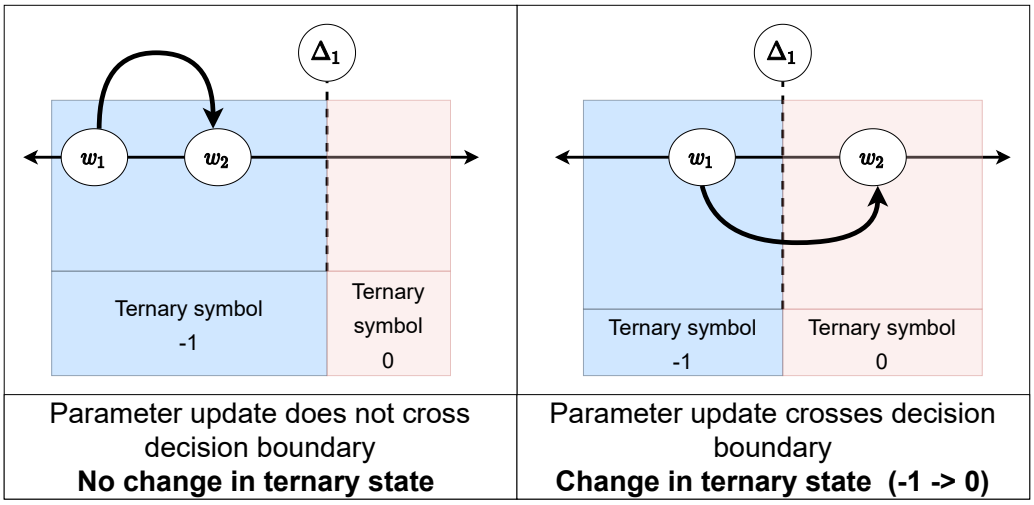


Figure 5.14: Resistance to parameter update in a ternary quantizer
The full precision weights present when training a quantizer experience a form of state inertia. This is illustrated by the fact that they cannot make a change to their quantized state unless they cross one of the decision boundaries within the quantizer. This also adds an element of regularization for the trainig optimizers, as only weights close to the decision boundaries can be used to actively minimize the loss. This is because with small parameter updates, the quantized state of those weight can be changed.

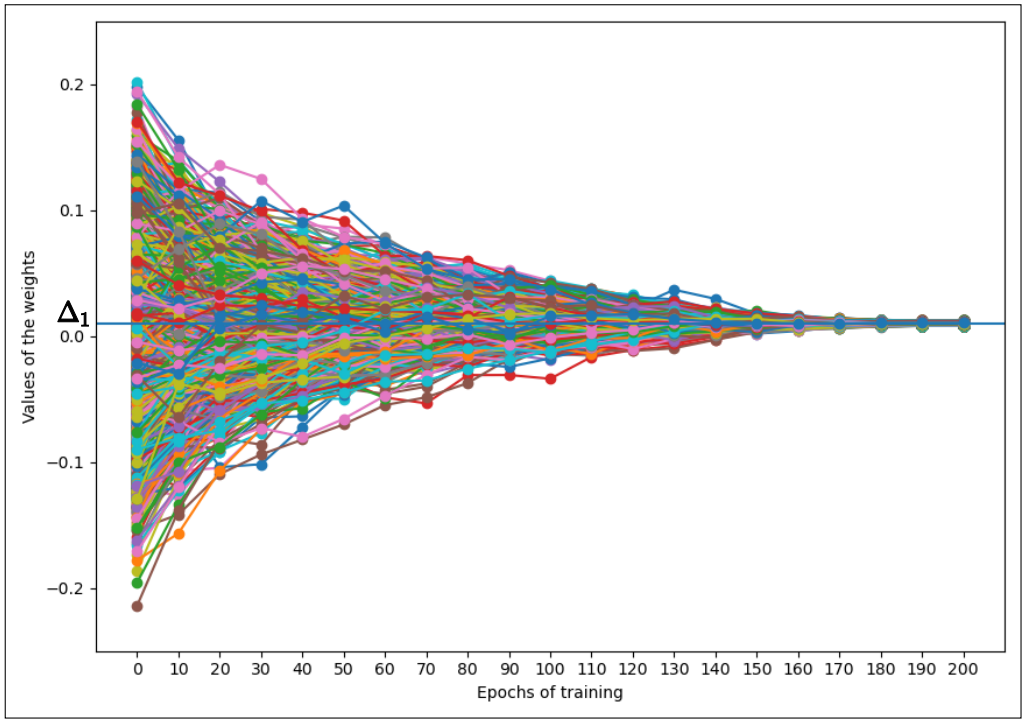


Figure 5.15: Movement of weights towards a decision boundary (Δ_1) in training
The regularization of the optimizer is clearly visible within this image. Weights that are far away from the decision boundaries, migrate closer in the course of training. The large number of weights (over 60%) settle to values very close to the decision boundaries of the quantizer.

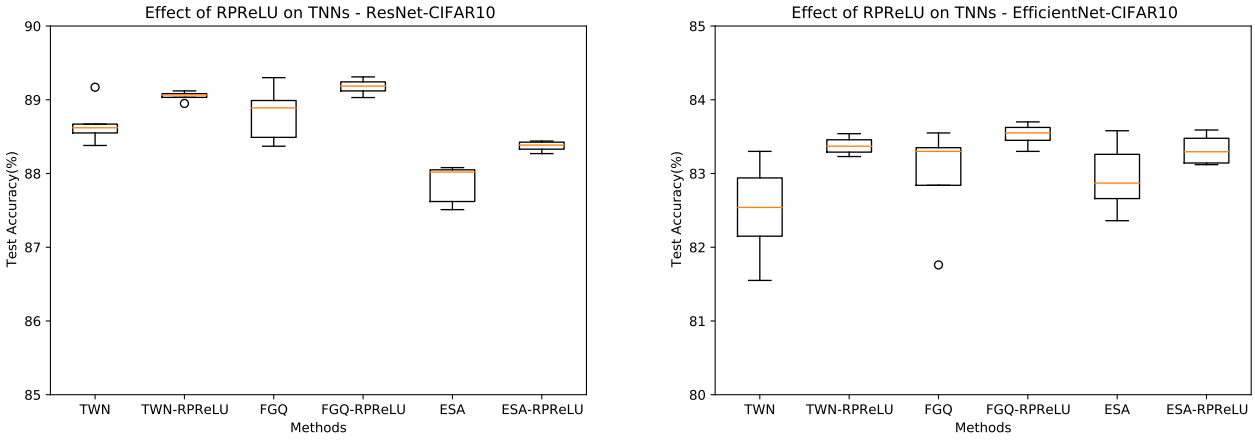
All these factors are observed in all ternary weight and completely ternary networks across data-set and network architecture. This also strongly indicates that the ternary distribution can be shaped by the decision boundaries.

5.3 Accuracy recovery methods from Binary Quantization

In the previous sections, the accuracies of various ternary quantized networks has been measured. The next step is to implement accuracy recovery methods from binary CNNs to completely ternary CNNs and benchmark their effectiveness.

5.3.1 Effect of ReAct PReLU on TNNs

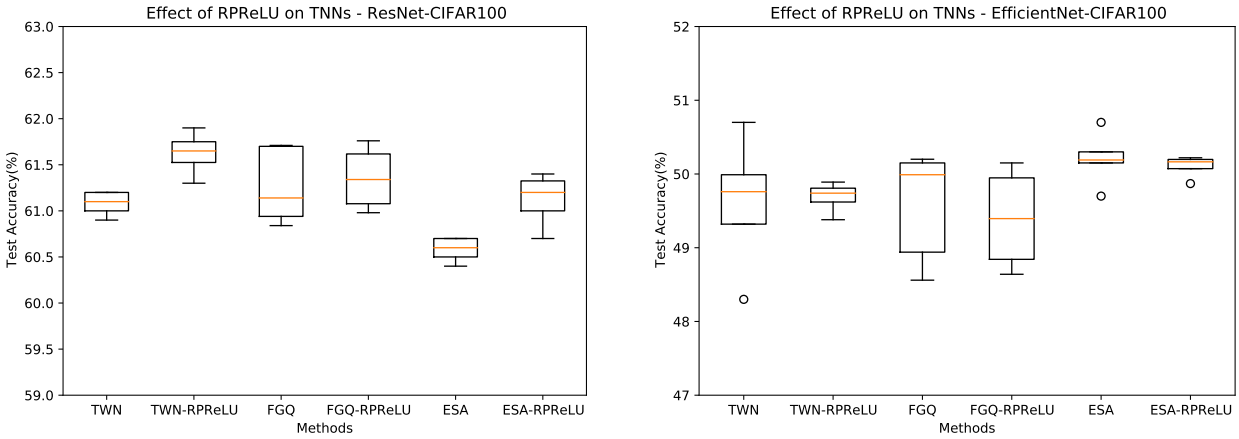
Within a completely ternary CNN, the activations are carrying a reduced amount of information due to the quantization of feature maps. The ReAct PReLU (RPReLU) creates a unique opportunity where the shifting and scaling of these feature maps can be performed along with the introduction of a leaky non-linearity. This also allows the RPReLU to affect the activations going through the network and transform them in a way that network accyracy is boosted. The addition of the ReAct-PReLU (RPReLU) adds some representative capacity to the network, 3 parameters per input channel per RPReLU. The results of these RPReLU experiments can be seen in figures 5.16 for CIFAR-10 and 5.17 for CIFAR-100.



(a) ResNet on CIFAR-10

(b) EfficientNet on CIFAR-10

Figure 5.16: Effect of RPReLU on TNNs for CIFAR-10
The effects of ReAct PReLU on ResNet and EfficientNet TNNs on the CIFAR-10 dataset are summarized in this figure.



(a) ResNet on CIFAR-100

(b) EfficientNet on CIFAR-100

Figure 5.17: Effect of RPReLU on TNNs for CIFAR-100
The effects of ReAct PReLU on ResNet and EfficientNet TNNs on the CIFAR-100 dataset are summarized in this figure.

These results of Figures 5.17 and 5.16 can be summarized within the Tables 5.9 and 5.10 for ResNet and EfficientNet respectively.

In addition to the average accuracies observed, the Figures 5.17a, 5.17b, 5.16a, 5.16b, it can be observed that the best performing method depends on a combination of the quantization methods, CNN architecture and test dataset.

ResNet20 with RPReLU				
Dataset	**Weight quantization method**	**Activation quantization method**	**Avg. Test Accuracy(%)**	**Standard deviation**
CIFAR-10	TWN	FATNN	89.074	0.0007
	FGQ	FATNN	89.2	0.0010
	ESA	FATNN	88.3	0.0013
CIFAR-100	TWN	FATNN	61.48	0.0038
	FGQ	FATNN	61.42	0.0031
	ESA	FATNN	61.02	0.0031

Table 5.9: Summary of TNN method with RPReLU: ResNet

EfficientNet with RPReLU				
Dataset	**Weight quantization method**	**Activation quantization method**	**Avg. Test Accuracy(%)**	**Standard deviation**
CIFAR-10	TWN	FATNN	83.21	0.0035
	FGQ	FATNN	83.48	0.0016
	ESA	FATNN	83.23	0.0025
CIFAR-100	TWN	FATNN	49.89	0.0043
	FGQ	FATNN	49.55	0.0065
	ESA	FATNN	50.14	0.0014

Table 5.10: Summary of TNN method with RPReLU: EfficientNet

Dataset	**Network Architecture**	**Acc. improvement from TNN Baseline**
CIFAR-10	ResNet-20	+0.4%
	EfficientNet	+0.4%
CIFAR-100	ResNet-20	+0.5%
	EfficientNet	+0.09%

Table 5.11: Accuracy improvement of TNN methods with RPReLU

The addition of the generalized activation function ReAct PReLU has a positive effect on the performance of all the models across all datasets. This is most clearly seen in Table 5.11. These experiments underline the important role the activation functions play in maximizing accuracy for extremely quantized networks.

5.3.2 Ablation of ReAct PReLU's components

The RPReLU consists of 3 components that act in concert to improve the accuracy of the quantized networks. These are the two scalar shifts that are applied before and after the activation function and the leaky non-linearity (PReLU). To ascertain which part of the RPReLU is actually critical to the accuracy improvement, we perform a small ablation test on the ResNet architecture. This test has 4 cases:

1. TNN : ReLU activation function (default case) (Fig 5.18)
2. TNN : PReLU activation function (Fig 5.18)

3. TNN : ReLU with both scalar shifts (RReLU) (Fig 5.19)
4. TNN : PReLU and both scalar shifts (RPReLU) (Fig 5.19)

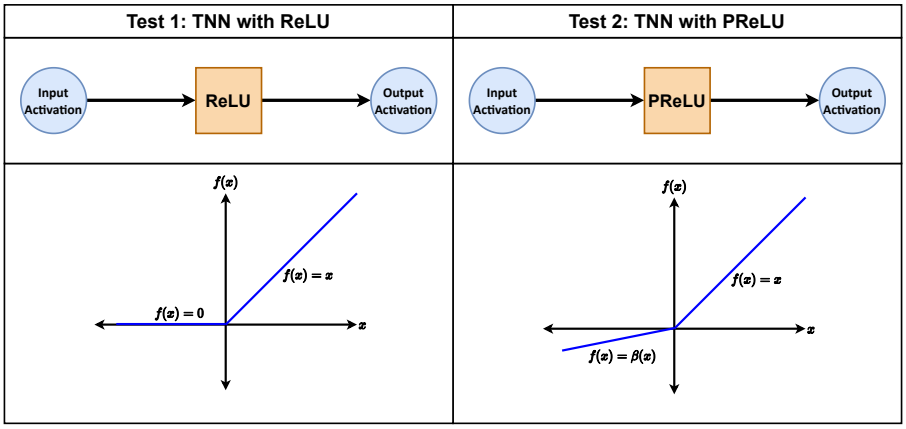


Figure 5.18: R-PReLU ablation Tests 1 and 2: TNN-ReLU, TNN-PReLU
Within the ResNet architecture, the activation functions are originally ReLU (left). The PReLU activation function (right) allows the negative activations to also pass through in a scaled manner. This is done through a learnable scaling parameter β, this leaky non-liearity is considered an improvement over ReLU.

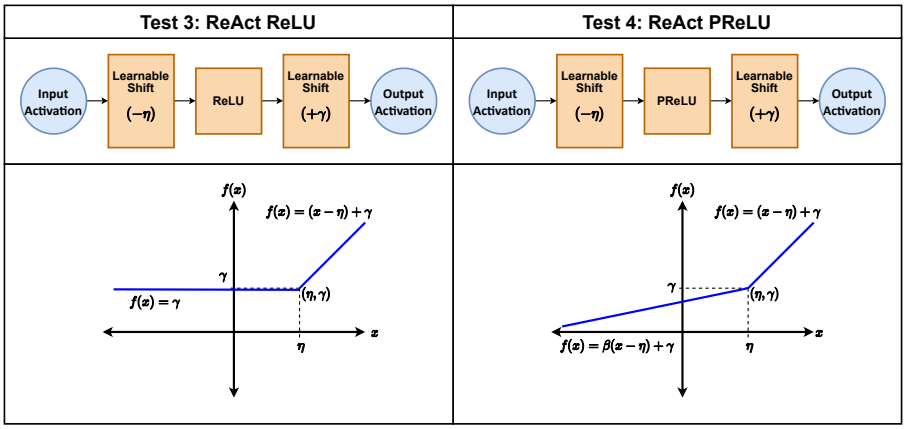
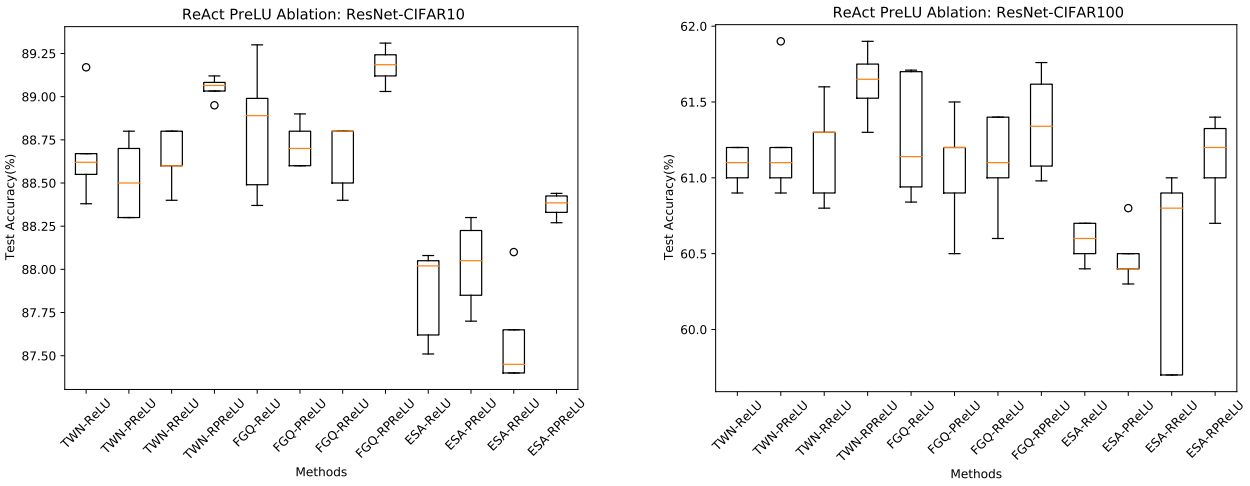


Figure 5.19: R-PReLU ablation Tests 3 and 4: TNN-RReLU, TNN-RPReLU
To test the contribution of the leaky non-linearity in the ReAct PReLU, this additional test is performed with a ReAct- ReLU (left) and the original ReAct-PReLU (right).

The results of the ablation test are shown in Figures 5.20a and 5.20b. It is clear that the ReAct-PReLU (test 4) performs the best overall compared to the other versions. A few high performing outliers are also seen within the training data, their performance could be the result of a favorable initialization.

To further analyze which scalar shift is more important for increasing model performance, another smaller set of experiments was performed. These tests overlap with the ones presented previously:

1. TNN : PReLU activation function (Fig 5.18)
2. TNN with one scalar shift before PReLU. (Fig. 5.21)
3. TNN with one scalar shift after PReLU. (Fig. 5.21)
4. TNN : PReLU and both scalar shifts (RPReLU) (Fig 5.19)



(a) Ablation tests: ResNet on CIFAR-10

(b) Ablation tests: ResNet on CIFAR-100

Figure 5.20: RPReLU ablation test results for ResNet-20
The results of the ablation test involving the RPReLU variants described in Figures 5.18, 5.19 are presented.

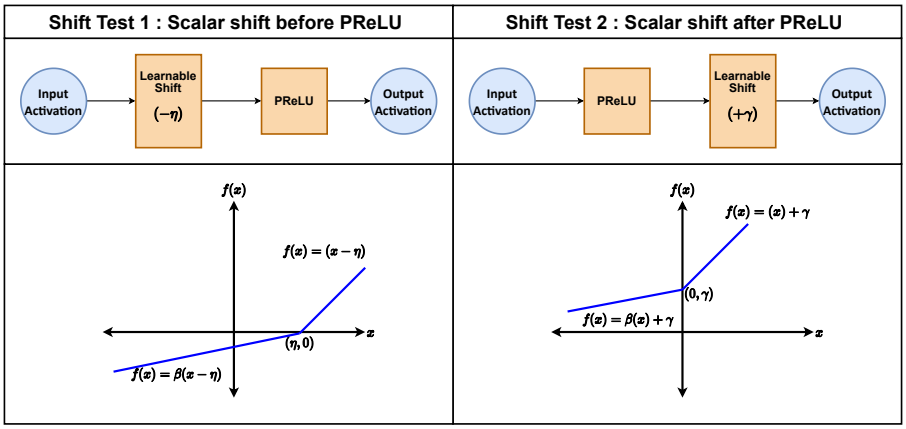


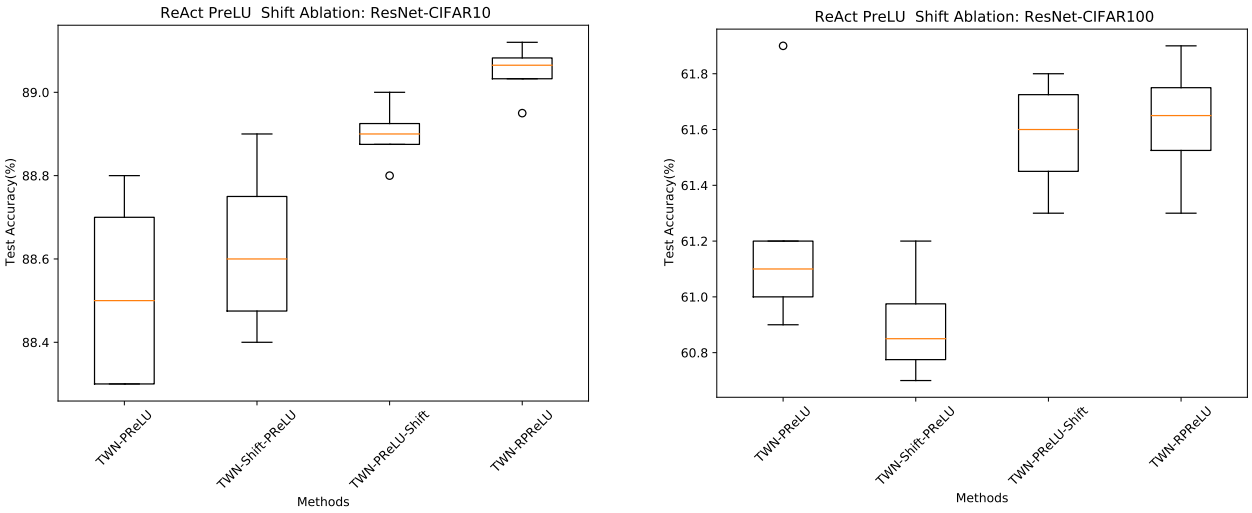
Figure 5.21: RPReLU ablation tests on scalar shifts
This ablation test is used to gauge the effectiveness of individual shifts within the ReAct-PReLU. Scalar shift before the non linearity (left) and scalar shift after non-linearity (right) are shown here.

The results presented in 5.22 show us that in addition to RPReLU being 0.5% better than the PReLU versions of TNNs. The scalar shift added after applying the PReLU non-linearity plays a much bigger role in the representative capacity of the network. The shift after the PReLU accounts for over 70% of the accuracy improvement provided by RPReLU.

All of the results presented above substantiate the idea that both the shifting and scaling of the activation together play a complimentary role in conditioning the activations and enhancing learning.

5.3.3 Effect of Approximate sign quantizer on TNNs

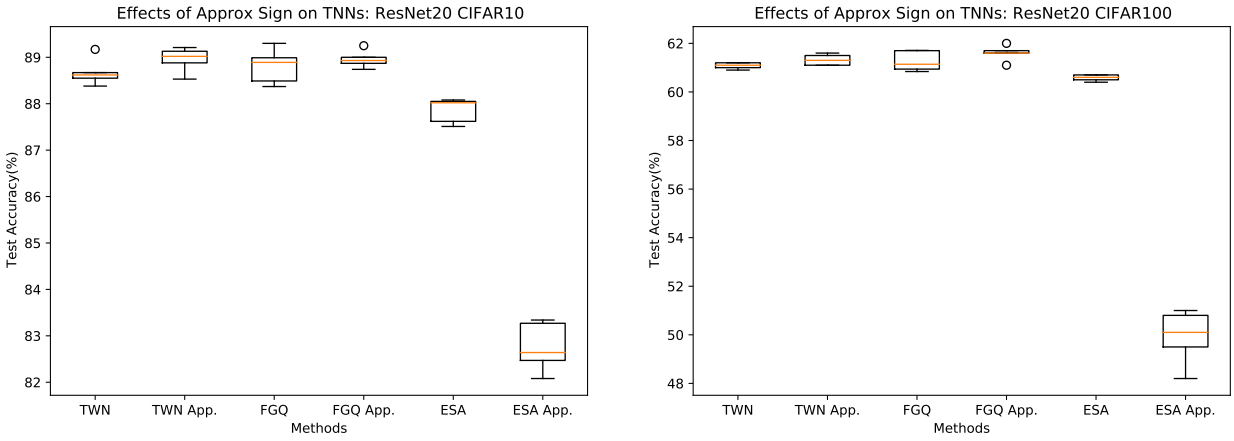
The approximate sign quantizer attempts to bring the derivative of quantizers used in CNN training closer to their true derivative, as opposed to using a Straight through estimator. This more faithful representation of the quantizer can lead to better convergence and better training through the quantizer.



(a) Ablation test: ResNet on CIFAR-10

(b) Ablation test: ResNet on CIFAR-100

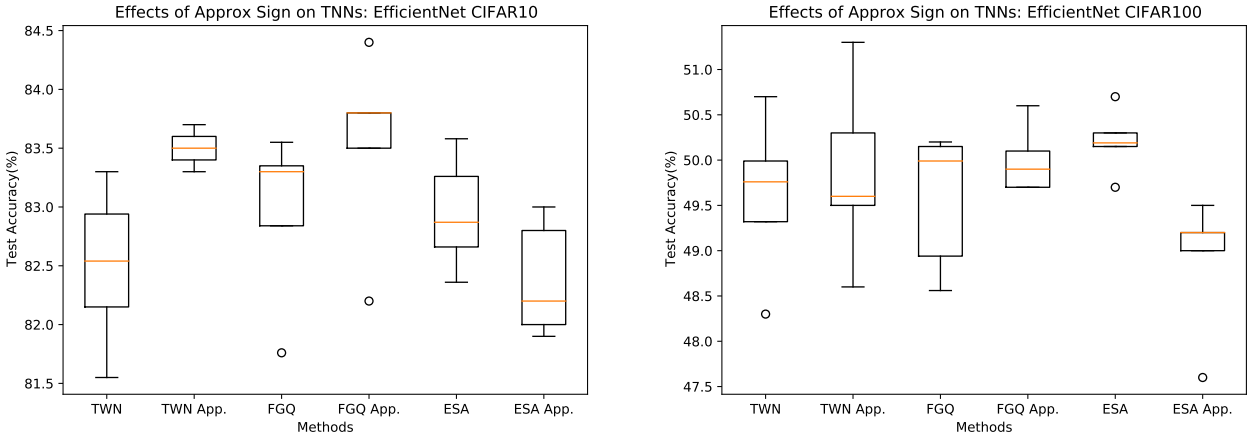
Figure 5.22: RPReLU scalar shift ablation test results for ResNet-20
The results of the ablation test involving the RPReLU variants described in Figures 5.18, 5.19, 5.21 are presented.



(a) CIFAR-10 Results

(b) CIFAR-100 Results

Figure 5.23: Effect of Approximate sign on ResNet20
These figures summarize the results of training ResNet20 with all the weight quantizer STEs replaced with the backwards pass of the approximate Sign quantizer.



(a) CIFAR-10 Results

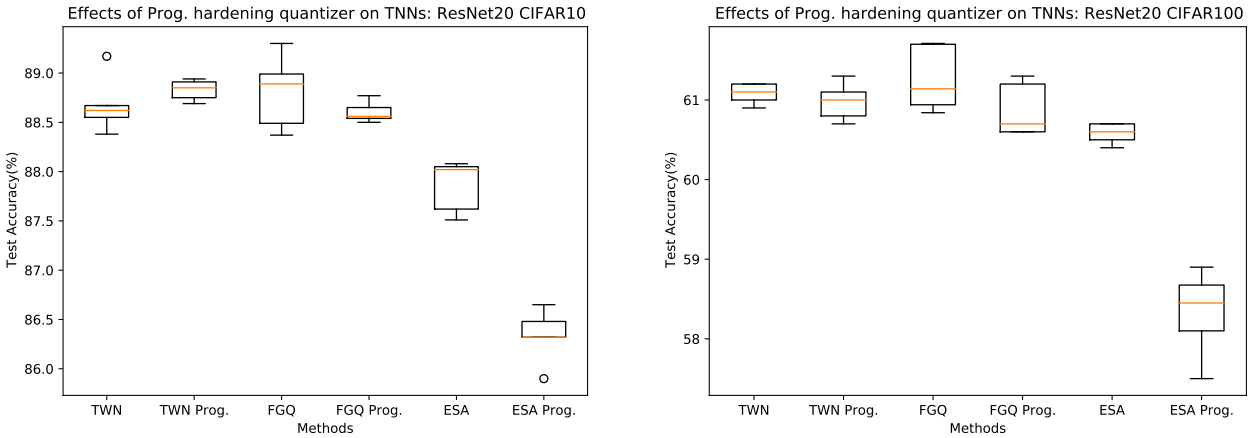
(b) CIFAR-100 Results

Figure 5.24: Effect of Approximate sign on EfficientNet
These figures summarize the results of training EfficientNet with all the weight quantizer STEs replaced with the backwards pass of the approximate Sign quantizer.

The approximate Sign quantizer consistently improves the performance of TWN and FGQ quantization methods. In contrast, there is a significant performance decrease when used alongside the ESA quantization method. This discrepancy is caused by the pre-distribution of weights using a *tanh* within ESA. This *tanh* adds an additional derivative term in the backwards pass. This appears to significantly affect the approximate Sign derivative from performing its function. This interference implies a very close coupling between the quantizer and its backward pass enhancement method. Both methods need to work without interference from each other to provide the maximum benefit.
High performing outliers can also be observed in both the ResNet20 and EfficientNet experiments, these can be caused by favorable initialization conditions,

5.3.4 Effect of Progressively hardening quantizer on TNNs

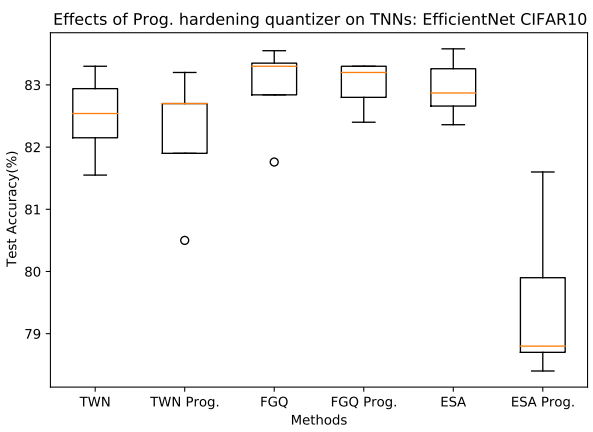
The progressively hardening quantizer attempts to bridge the gap between the STE and the impulse response derivative of a quantizer. This method proposed a backwards pass enhancement which simulates the quantizer slowly becoming more and more discrete as training goes on. This simulated progressive discretization should also help the network learn the discrete distribution reflected by the quantizer.



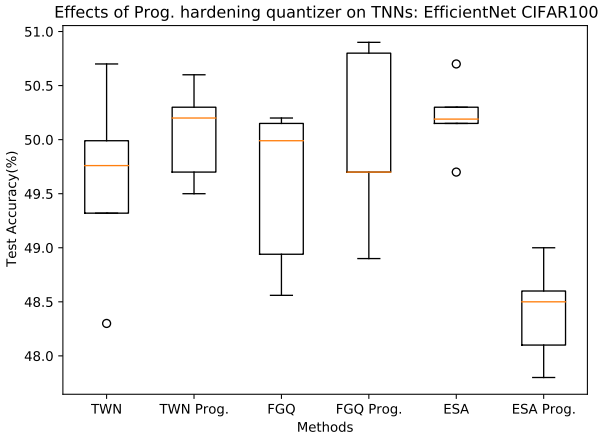
(a) CIFAR-10 Results

(b) CIFAR-100 Results

Figure 5.25: Effect of progressively hardening quantizer on ResNet20
These figures summarize the results of training ResNet20 with all the weight quantizer STEs replaced with the backwards pass of the progressively hardening quantizer.



(a) CIFAR-10 Results



(b) CIFAR-100 Results

Figure 5.26: Effect of progressively hardening quantizer on EfficientNet
These figures summarize the results of training EfficientNet with all the weight quantizer STEs replaced with the backwards pass of the progressively hardening quantizer.

High performing outliers can also be observed in both the ResNet20 and EfficientNet experiments, these can be caused by favorable initialization conditions.

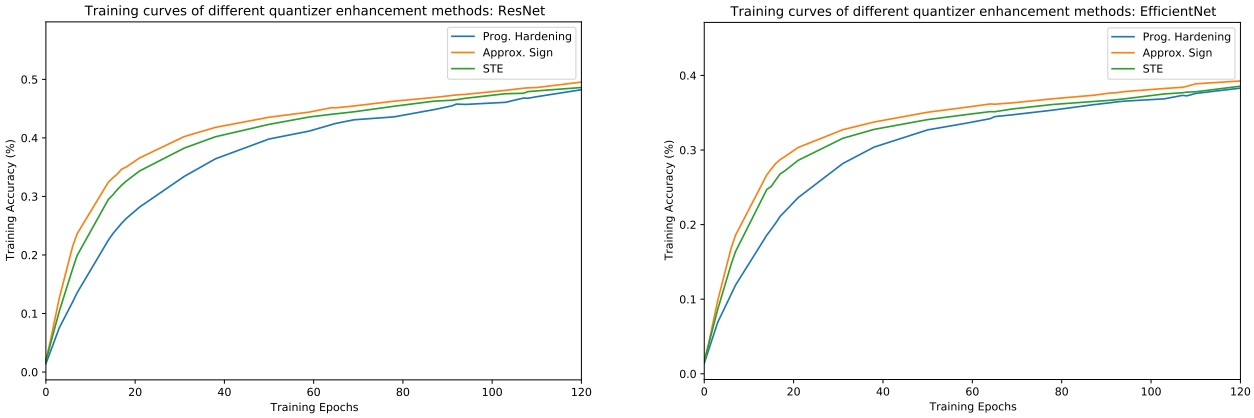
The results for progressive hardening echo those of the Approximate sign quantizer enhancement. TWN and FGQ benefit from this method, but ESA's performance is degraded. This mirroring of performance further strengthens the intuition that the quantizer and the quantizer enhancement method used have very close coupling between them. An element of co-design will help improve the performance of these networks even further.

5.3.5 Effect of longer training times on quantizer enhancements

In the sections 5.3.3, 5.3.4 the performance of the approximate Sign quantizer and progressively hardening quantizer STE enhancements showed some mixed results. The approx. Sign performs better than the STE version. Whereas, the Progressively hardening quantizer does not perform as well the STE version. This motivates a deeper look into how their performance evolves during the training process.

Motivation for longer training time tests

The training curves for the two quantizer enhancement methods paint a different picture. The training curves can be seen in Figure 5.27. The approximate Sign quantizer leads to a much faster initial convergence, while the progressively hardening quantizer converges much slower. All three methods appear to converge quite close to each other by the end of training. This close convergence motivates further experiments to judge the maximum realizable accuracy gain provided by these quantizer enhancements. The easiest way to do that is to increase the amount of training epochs and then compare how well STE, approx. Sign and progressively hardening quantizers perform.



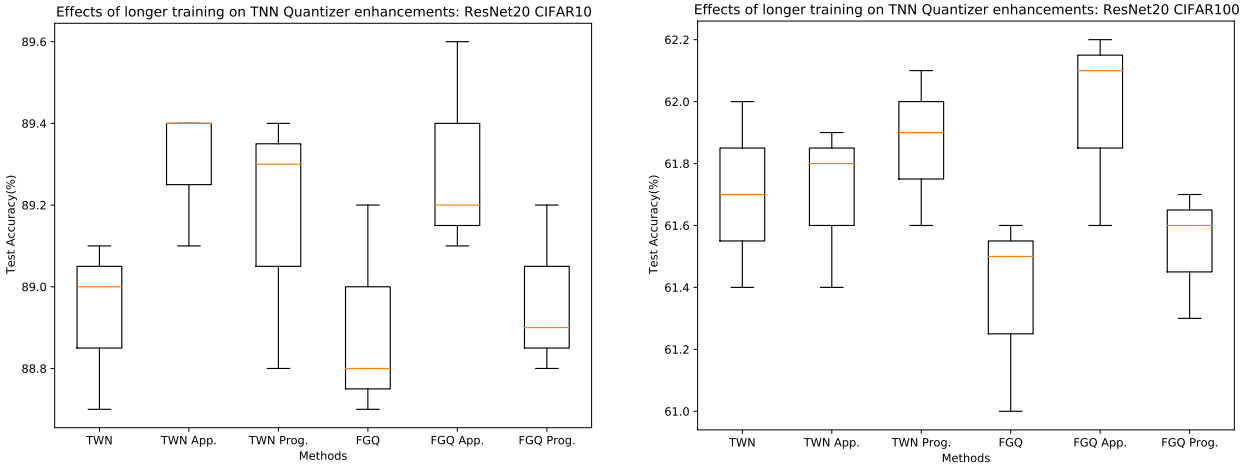
(a) Training curve for ResNet20-CIFAR 100

(b) Training curve for EfficientNet-CIFAR 100

Figure 5.27: Training curve for quantizer enhancement methods
The training accuracy curves for STE, approx. Sign and progressively hardening quantizer are shown for both the ResNet20 and EfficientNet architectures on the CIFAR-100 dataset.

Results of the longer training time tests

To verify if the lengthening of training will help quantizer enhancements exceed the performance of the STE version, training TNNs was done for 25% more training epochs (total 250 epochs). This experiment set included all three verions of the quantizer: baseline TNN with STE, TNNs with approx. Sign and TNN with progressively hardening quantizer.



(a) CIFAR-10 Results

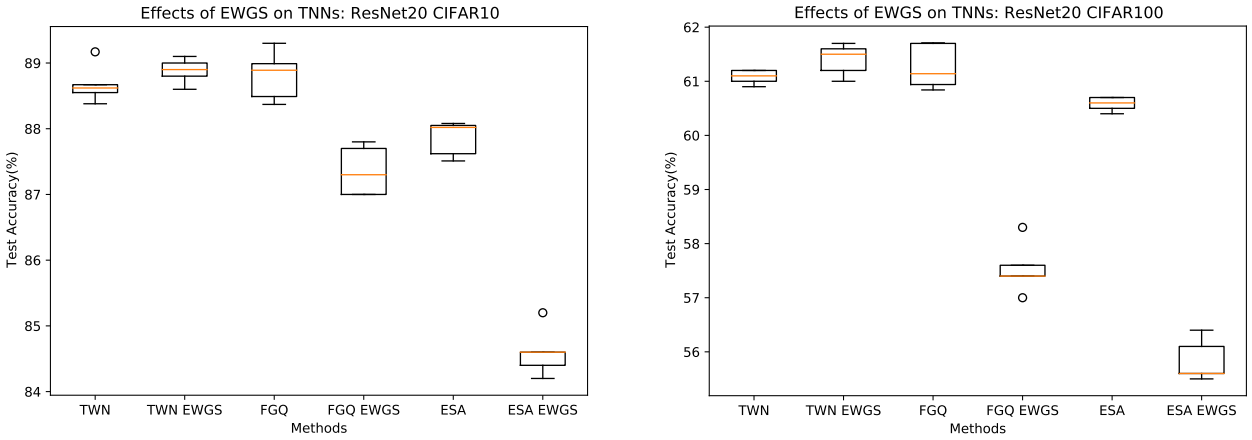
(b) CIFAR-100 Results

Figure 5.28: Effect of longer training STE enhancements for ResNet20
The effects of longer training times on test accuracy of ResNet TNNs with quantizer enhancements on the CIFAR-100 dataset are summarized in these figures.

It is seen in Figure 5.28 that training for a longer period of time favors the approx. Sign and progressively hardening quantizers over the STE. This further strengthens the case for the use of backward pass quantizer enhancements for ternary quantizers.

5.3.6 Effect of Element wise gradient scaling on TNNs

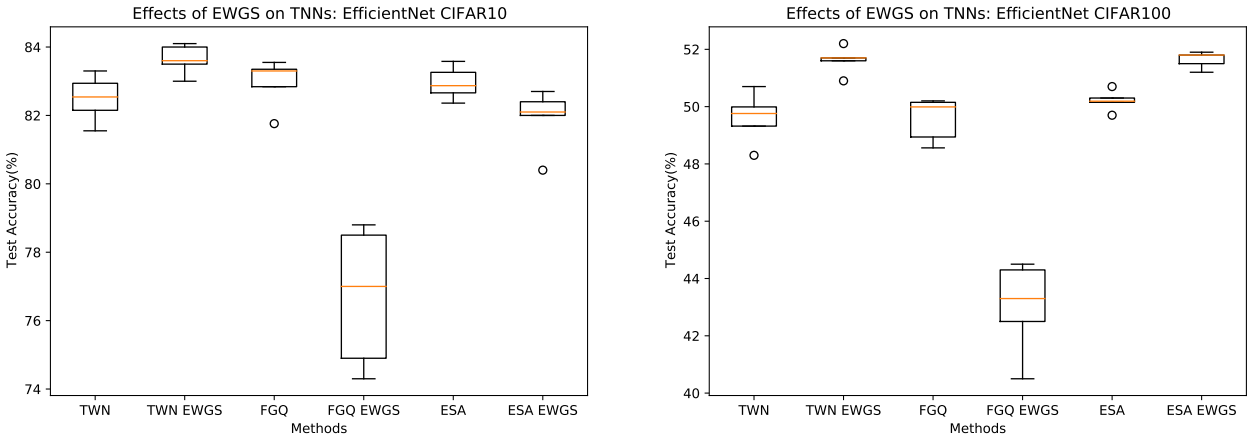
Element wise gradient scaling attempts to reduce the quantization error experienced between a variable and its quantized value. This in addition with the regularization provided by the ternary quantizer should help the weights of a network distribute and generalize better. The results of EWGS's interaction with TNNs appears



(a) CIFAR-10 Results

(b) CIFAR-100 Results

Figure 5.29: Effect of Element wise gradient scaling on ResNet20
These figures summarize the results of training ResNet20 with Element wise gradient scaling applied to the weights in the backwards pass quantizer.



(a) CIFAR-10 Results

(b) CIFAR-100 Results

Figure 5.30: Effect of Element wise gradient scaling on EfficientNet
These figures summarize the results of training EfficientNet with Element wise gradient scaling applied to the weights in the backwards pass quantizer.

quite deeply tied to the mixed precision representation provided by the quantizer. In the case of FGQ the output of the quantizer has many different scaling factors for the same weight quantizer. This may provide misleading EWGS scaling factors and worse convergence. A similar issue can be observed in the ESA case. The interference of the *tanh* distributes the weights in a manner that may lead to a gradient mismatch issue. Overall, an alternate implementation of EWGS that could take into account relative distance from multiple possible quantization states may lead to better performance.

Chapter 6

Summary and Conclusions

The ternary quantization of CNNs as well as methods to enhance this quantization were discussed within this work. The mechanisms of quantization and the effects it has on CNNs are key to creating more accurate and higher performing ternary quantizers. Methods to improve the accuracy of these ternary CNNs were also studied and the encouraging experimental results motivate a greater use of these methods. The computational cost of these methods must also be considered when creating a deployable CNN solution.

6.1 Reflecting on the research questions

After a thorough examination of the design space specified for this project and presentation of the results obtained, we can reflect on the central research questions posed at the start of the project.

6.1.1 Research question 1:

How much accuracy degradation do completely ternary quantized CNNs experience compared to full-precision CNNs? Compared to completely binary CNNs how much accuracy improvement do completely ternary CNNs provide?

In the best cases, ternary weight networks achieve performance within 3 to 4% of full-precision networks, as seen in section 5.2.1. The best performing completely ternary networks achieve accuracies within 4 to 8% of their full precision counterparts, as seen in section 5.2.3. This accuracy degradation is most critically dependent on the number of parameters within the network. As seen in the results, the EfficientNet architecture with half the parameters of ResNet suffered almost double the accuracy degradation at 18%. Thus care must be taken to properly judge the learning task and select well parameterized networks.

All ternary networks tested within this work are more performant than an equivalent binary network. For CIFAR-10 experiments, completely ternary networks ($\sim$ 88.5% acc.) performed 4-5% better than equivalent binary networks ($\sim$ 85% acc.). This improvement is more apparent for larger datasets, completely ternary networks ($\sim$ 60% acc.) performed 10-15% better than equivalent binary networks ($\sim$ 45% acc.). This gap between ternary and binary is most apparent for EfficientNet on CIFAR-100. The EfficientNet performs well in full precision ($\sim$ 67% acc.) and for ternary quantization we see a large drop of -18% to $\sim$ 51.5% accuracy, for binary the performance was observed even lower at $\sim$ 40% accuracy. The performance gap in representational capability that only widens with harder learning tasks and less parameterized networks. Ternary quantization states provide much more fine grain representative capacity within the convolution than is possible with binary values.

6.1.2 Research question 2

Given the large amount of binary neural network accuracy recovery methods, can we utilize/extend these methods to enhance Ternary quantization? How much improvement/accuracy gain do these methods provide for a set of standard benchmarks

A large amount of accuracy enhancement methods exist for binary CNN quantization methods. A few of these methods are described in Section 3.2. Most binary accuracy recovery methods can be applied to ternary CNNs with minimal changes. Methods that enhance the capacity of the network [20, 21, 22] can be directly applied to ternary and will provide an improvement similar to those proposed in binary. Methods that enhance the quantizer during training [36, 37, 47], need some modifications to be applied to ternary networks. Quantizer enhancement methods also may require longer training time as seen in Section 5.3.5. Proper synergy between enhancement methods and quantizer used is important to maximize performance. This is underlined in the poorer performance of ESA quantization method in certain quantizer enhancements (Fig 5.23, 5.25) than other

ternary methods. ESA often suffered a 6% drop in effectiveness compared to the 1 - 2% improvement seen by other methods.

Within the results discussed in this work, the application of methods presented in section 3.2 cumulatively lead to an average improvement of $\sim 2\%$ across datasets and both ResNet20 and EfficientNet architectures for completely ternary CNNs. This is an encouraging result, motivating that more accuracy recovery methods be applied to ternary networks.

6.2 Conclusions

Quantization of neural networks allow artificial intelligence to be deployed on a range of resource constrained edge devices by reducing the size of CNN models and in some cases, computational simplicity. Within this range of quantization, binary and ternary quantization enjoy the highest model compression (32x and 16x-20x). These methods also benefit from simplification in computation through bitwise operation and even lower bit-width MAC operations.

In the survey of ternary quantization works specified in section 3.1, it is clear that the performance of ternary networks is higher than that of binary[1]. This motivates the use of ternary quantized networks in any application, where binary networks may be considered.

When deciding which accuracy recovery method from binary CNNs (Section 3.2) to apply on ternary CNNs, two core points must be kept in mind:

- Quantizer enhancement methods may require additional training time, a more favorable initialization and synergy with the ternary quantization method to achieve more significant accuracy improvement.
- Architectural changes and capacity increases have the most direct positive effect on model performance but may have diminishing returns if too many parameters are added.

The accuracy recovery methods discussed in this work cumulatively can lead to a $\sim$2% improvement for fully ternarized networks, across architectures and datasets. This motivates the application of more accuracy recovery methods to ternary CNNs that can help bridge the gap to full-precision CNNs.

6.3 Suggestions and Future Work

From the results presented within this work and trends within the behavior of ternary CNNs, the following research directions could lead to significant progress for ternary network quantization:

- Quantization methods combining bimodal distribution weight initialization, weight regularizers and learnable quantizer bounds[25] can be created and with inclusion of accuracy recovery enhancements should match or exceed the state of the art for TNNs.
- Ternary training with higher cost accuracy recovery methods to further bridge that gap from full-precision CNNs. The addition of architectural changes like those specified in [21, 22, 42, 53] can be used to attain the optimal mix of parameterization and network compression.
- Hardware accelerators to verify the performance for TNNs in terms of accuracy and energy efficiency when applying accuracy recovery methods. This can help further motivate the enhancement of TNNs and their adoption within the industry. Works similar to CUTIE[28] and Chewbaccann[19] with custom ternary and mixed precision hardware can help establish that tradeoff between accuracy enhancement and energy usage.
- As we formalize ternary quantization as a viable option for widespread use, we must also account for the reliability and error resilience of such a network. The concerns of accuracy across classes, implicit biases within the network and resistance to adversarial/directed attacks are a very prominent topic of research within the wider artificial intelligence space. Using methods like [54, 55, 56] within the ternary context would be a large step forward in formalizing the reliability and trustworthiness of quantized networks.

[1] Ternary networks perform 4% better in CIFAR-10 and 10% better in CIFAR-100

Bibliography

[1] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

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

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8614–8618.

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

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

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

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[12] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle, "Brain tumor segmentation with deep neural networks," *Medical image analysis*, vol. 35, pp. 18–31, 2017.

[13] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[14] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[15] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[16] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[17] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[18] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 236–241.

[19] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "Chewbaccann: A flexible 223 tops/w bnn accelerator," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[20] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, "Reactnet: Towards precise binary neural network with generalized activation functions," in *European Conference on Computer Vision*. Springer, 2020, pp. 143–159.

[21] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured binary neural networks for accurate image classification and semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 413–422.

[22] A. Bulat, B. Martinez, and G. Tzimiropoulos, "High-capacity expert binary networks," *arXiv preprint arXiv:2010.03558*, 2020.

[23] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[24] X. Deng and Z. Zhang, "An embarrassingly simple approach to training ternary weight networks," *arXiv preprint arXiv:2011.00580*, 2020.

[25] P. Chen, B. Zhuang, and C. Shen, "Fatnn: Fast and accurate ternary neural networks," *arXiv preprint arXiv:2008.05101*, 2020.

[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[27] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 1921–1925.

[28] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "Cutie: Beyond petaop/s/w ternary dnn inference acceleration with better-than-binary energy efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[29] S. Jain, S. K. Gupta, and A. Raghunathan, "Tim-dnn: Ternary in-memory accelerator for deep neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 7, pp. 1567–1577, 2020.

[30] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *arXiv preprint arXiv:1511.00363*, 2015.

[31] O. Muller, A. Prost-Boucle, A. Bourge, and F. Pétrot, "Efficient decompression of binary encoded balanced ternary sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1962–1966, 2019.

[32] N. Nazari, M. Loni, M. E. Salehi, M. Daneshtalab, and M. Sjodin, "Tot-net: An endeavor toward optimizing ternary neural networks," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 305–312.

[33] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Networks*, vol. 100, pp. 49–58, 2018.

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

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[36] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 722–737.

[37] A. Bulat, G. Tzimiropoulos, J. Kossaifi, and M. Pantic, "Improved training of binary networks for human pose estimation and image recognition," *arXiv preprint arXiv:1904.05868*, 2019.

[38] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[39] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," *arXiv preprint arXiv:2003.11535*, 2020.

[40] L. Hou and J. T. Kwok, "Loss-aware weight quantization of deep networks," *arXiv preprint arXiv:1802.08635*, 2018.

[41] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[42] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[43] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[44] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

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

[46] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[47] J. Lee, D. Kim, and B. Ham, "Network quantization with element-wise gradient scaling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6448–6457.

[48] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary neural networks with fine-grained quantization," *arXiv preprint arXiv:1705.01462*, 2017.

[49] P. Xue, Y. Lu, J. Chang, X. Wei, and Z. Wei, "Self-distribution binary neural networks," *arXiv preprint arXiv:2103.02394*, 2021.

[50] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[52] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[53] Z. Xu, M. Lin, J. Liu, J. Chen, L. Shao, Y. Gao, Y. Tian, and R. Ji, "Recu: Reviving the dead weights in binary neural networks," *arXiv preprint arXiv:2103.12369*, 2021.

[54] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[55] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," in *International Conference on Learning Representations*, 2018.

[56] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. T. Shen, "Binary generative adversarial networks for image retrieval," in *Thirty-second AAAI conference on artificial intelligence*, 2018.