Optimal Compression of Convolutional Neural Networks for Edge Devices

A Project Report

submitted by

ABISHEK S

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
May 2022

THESIS CERTIFICATE

This is to certify that the thesis entitled **Optimal Compression of Convolutional Neural Networks for Edge Devices**, submitted by **Abishek S** (**EE18B001**), to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Avhishek Chatterjee
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 27/05/2022

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to **Prof. Avhishek Chatterjee** who guided me throughout the course of this project. The interactions and regular meetings I had with him taught me a lot of things within and outside this research, kept me motivated and excited constantly, and transformed my perspective on research as a whole. I thank him for the opportunity to work on this project which is very relevant to the current times. I would also like to thank and acknowledge the support of Anirudh S (EE18B073), who worked with me on this work and contributed his ideas and observations.

I extend my sincere gratitude to IIT Madras for the quality of education and the opportunities that I got in my four amazing years of undergraduate at this place. The contribution of the university towards my growth as a knowledgeable student, as a researcher, and as an individual is immeasurable.

I would like to thank my parents, sister, and my friends for their never-ending support, encouragement, and care helped me reach where I am today that keeps me going. Finally, my special thanks to the Almighty, who has surrounded me with such good people and showered me with good health and wisdom.

ABSTRACT

KEYWORDS: Convolutional Neural Networks, Edge ML, Quantization, Principal Component Analysis, Clustering, Pruning, Two Nearest Neighbors, Mixed Precision Layer Design

Recently, there has been a lot of interest in deploying neural networks on mobiles, sensors, and other edge devices that are restrained in storage, computational power, and energy. The main success of the neural networks in tasks such as image classification can be attributed to their superlative performance over traditional machine learning methods. However, a major drawback of using these deep neural networks has been their high storage and processing requirements which most edge devices lack. Hence, there has been a constant trade-off between model performance and its memory and computational requirements. Despite this trade-off between accuracy and model size, several neural network model compression techniques have been introduced to reduce the discriminating power of this trade-off and achieve high performance along with high memory need reduction. We provide a detailed comparative study of some of the most relevant and useful methods for compressing convolutional neural networks, namely quantization (post-training and quantization aware training), pruning, parameter clustering, and hybrid layer precision design that show potential for insights and improvement with a focus on reducing the memory footprint for image classification tasks among other issues like latency and energy. We propose a pipeline to efficiently compress CNN-based models through analyzing the intrinsic dimensionalities of data representations across layers that achieves high performance and makes it less memory intensive to be suitable for deployment on limited resource edge devices. Using the same intrinsic dimension idea, we also optimally extend the hybrid layer precision design for memory crunch situations. The experiments are performed using **QuantNet** model on binary image classification tasks from the **CIFAR-10** dataset like Cat vs Dog and Horse vs Frog. Briefly put, this work puts forward a novel and promising approach to analyzing neural network layers for transfer learning setup (that applies to the source domain setup as well) and utilizing it to optimally combine different compression techniques for achieving great performance with a significant reduction in model size.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **i**

ABSTRACT **ii**

LIST OF TABLES **vi**

LIST OF FIGURES **viii**

ABBREVIATIONS **viii**

NOTATION **ix**

1 INTRODUCTION **1**

2 LITERATURE REVIEW & THEORY **5**

2.1 Quantization 5

2.1.1 Mathematical Formulation 6

2.1.2 Classification 6

2.1.3 Mixed Precision Quantization 10

2.2 Weight Clustering 12

2.3 Pruning 14

2.4 Intrinsic Dimension Estimation 16

2.4.1 PCA Method for Intrinsic Dimension Estimation 16

2.4.2 TwoNN Method for Intrinsic Dimension Estimation 17

3 RESEARCH METHODS **20**

3.1 Problem Statement Motivation 20

3.2 Description of Model used for Studies - QuantNet 22

3.3 TwoNN-based Hybrid Precision Quantization 24

3.3.1 Correct Scaling Factor and Gradients 26
3.3.2 TwoNN Method for Identifying Significant Layers 28
3.4 Proposed Model Pipelines 30

4 EXPERIMENTS 33
4.1 Setup 33
4.1.1 Dataset 33
4.1.2 Hyperparameter Tuning 35
4.1.3 Model Training Details 35
4.2 Results and Findings 36
4.2.1 Study on Conventional Quantization Approaches 36
4.2.2 Study on Mixed Precision Quantization 40
4.2.3 Study on Model Pruning and Weight Clustering 46
4.2.4 Performance of Proposed Model Pipeline 50

5 CONCLUSION AND FUTURE WORK 52

LIST OF TABLES

4.1 Summary of QuantNet model used in the experiments 33

4.2 Hyperparameter setting of QuantNET 36

4.3 Test performance and model size of QuantNet under different quantization schemes on Cat vs Dog and Horse vs Frog datasets (standalone) 37

4.4 Test performance and model size of QuantNet under different quantization schemes with Cat vs Dog source domain, Horse vs Frog and Rotated Cat vs Dog target domains (Transfer learning setup) . . . 38

4.5 Summary of the intrinsic dimension analysis on Quant-Net model for Cat vs Dog source domain and Horse vs Frog target domain . 41

4.6 Performance summary† of TwoNN and PCA approaches on Quant-Net model for Cat vs Dog Source Domain (SD) and Horse vs Frog Target Domain (TD) 44

4.7 Test performance and model size of QuantNet under different clustering schemes for Cat vs Dog 48

4.8 Test performance, number of clusters in each layer, and model size under different clustering + QAT schemes on QuantNet model for Cat vs Dog 49

4.9 Test performance and model size of QuantNet under the best performing compression schemes for Cat vs Dog dataset from CIFAR-10 in standalone setup 50

LIST OF FIGURES

2.1 Uniform vs Non-uniform Quantization 7

2.2 Symmetric vs Asymmetric Quantization 8

2.3 Illustration of Quantization Aware Training 9

2.4 Illustration of the working of PCA-based Hybrid-Net design which selectively quantizes different layers independently based on their significance denoted by higher increase in *nPC* from previous layer 13

2.5 Illustration of Weight Clustering 14

2.6 Illustration of Weight and Neuron Pruning 15

2.7 Illustration of ID estimation through TwoNN 18

3.1 Detailed Model Architecture of QuantNet for binary image classification task 23

3.2 The ID of data representations following a hunchback profile . . . 29

3.3 Illustration of identifying significant layers using TwoNN method on QuantNet model for the CIFAR-10 dataset, with threshold $\Delta = 20$ 30

3.4 Flow diagram of the optimal model pipeline proposed for limited resource edge devices 32

3.5 Flow diagram of the optimal model pipeline proposed for memory-constrained edge devices 32

4.1 Plot showing the effect of quantizing each layer on the test accuracy 39

4.2 Principal Component Analysis on the intermediate layers of QuantNet for Horse vs Frog of CIFAR-10 dataset 42

4.3 Identifying significant layers using TwoNN and PCA on QuantNet model for transfer domain in CIFAR-10 dataset, with threshold $\Delta =$ 20 42

4.4 Significant layers (shaded red) identified by TwoNN (left) and PCA (right) with threshold $\Delta = 20$, along with the values of the significance criteria 45

4.5 Variation of test accuracy with pruning 46

4.6 Variation of test accuracy with number of clusters (N_c) 47

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
CNN	Convolutional Neural Network
ML	Machine Learning
IC	Integrated Circuit
CPU	Central processing unit
GPU	Graphics processing unit
TPU	Tensor processing unit
FLOP	Floating Point Operations
DNN	Deep Neural Network
NLP	Natural Language Processing
QAT	Quantization Aware Training
PCA	Principal Component Analysis
TwoNN	Two Nearest Neighbors
ID	Intrinsic Dimension
CDF	Cumulative Distribution Function
MAC	Multiply and Accumulate Computations
NLG	Natural Language Generation

NOTATION

nPC	Number of Principal Components
fp32	32-bit Floating Point
n	Flattened dimensions of a general weight tensor
M	Minibatch size
O	Channel dimension of CNN output
W	Width dimension of CNN output
H	Height dimension of CNN output
N	Number of input samples used for training

CHAPTER 1

INTRODUCTION

The history of neural networks dates back to 1943 when Mcculloch and Pitts [1943] associated logic with neural activity and proposed the first artificial computational neuron for binary inputs. This was followed by Rosenblatt [1958] introducing the perceptron handling real inputs and an algorithm for learning the weights. The linear nature of the earlier models and the lack of powerful computers halted progress on deep neural networks (DNNs), and the traditional ML approaches dominated most of the late 20th century. On the image classification front, image features detectors, Naive Bayes, K-Means clustering, K-Nearest Neighbor, Support Vector Machines, and Decision Trees were widely used until the first Convolutional Neural Network (CNN) architecture was proposed by Lecun *et al.* [1998] in the form of LeNet-5 for handwritten character recognition in the late 20th century. The deep CNNs have been the dominant choice for image classification tasks ever since, with the constantly evolving processing power of computers making them more computationally viable and the sophisticated architectures making predictions more accurate.

In 1965, Gordon Moore stated empirically that the number of transistors packed in a dense integrated circuit (IC) doubles about every two years (*Moore's law*). However, electronic devices are getting increasingly smaller at an even more rapid rate resulting in limited memory and processing power. In addition, the CPUs, GPUs, and even the most recent TPUs that train or run inference on these models consume a lot of energy pertaining to the huge number of FLOP operations and memory accesses for read and write operations. The battery-powered devices still cannot afford to run the complex neural network models due to their energy budget. As a quantifying example, the ResNet-50 model [He *et al.*, 2015] with 50 convolution layers takes up over 100 MB memory and uses more than 3.8 billion floating-point operations while processing an image [Cheng *et al.*, 2017]. The latest smartphones cannot run image classification with AlexNet for more than an hour [Yang *et al.*, 2017]. All these claims might raise a question on whether DNNs are really necessary for the limited resource edge devices. The answer lies in the fact that data collected from such sources are usually noisy, large, and heterogeneous and the conventional ML methods fail in these settings. Hence with the advent of the Internet of Things (IoT) devices, augmented reality, embedded systems, real-time application devices, and distributed systems, the need for support of DNNs on such systems becomes a pressing issue for ensuring their purpose of autonomous intelligence.

A predominantly used approach employed on current edge devices is to transmit the data collected on the edge to the centralized cloud servers, where the model resides and runs on the input data, returning the output back to the device for further processing or usage. There are quite a few shortcomings to this approach, though. The first straightforward disadvantage is the latency delays that occur twice - while communicating the data to the cloud machine and while retrieving the output from the cloud. This does not go well with devices that run time-critical applications like braking a car, for instance. In 2021, there were over 10 billion active IoT devices alone, and it is estimated that by 2025, there will be around 152,200 IoT devices connecting to the Internet per minute [Jovanovic, 2022]. These deepen the latency problem further by creating huge load and traffic on the cloud servers. Secondly, there is the issue of the unreliability of the centralized cloud computing service. Failure of the cloud service in a region might affect a large number of devices relying on it, and even with replications on multiple regions, there is a problem of increased latency. Finally, there is also cost associated with cloud computing, and for applications involving deep learning, the clusters will be powerful and the cost will be considerable.

Hence, the need for a decentralized approach to deploying neural networks on edge devices after suitably compressing them is established. This needs to happen without losing out on the primary advantage of neural networks - their performance. Several important works in this regard have been popular since they were ideated, like quantization [Jacob *et al.*, 2018] [Rastegari *et al.*, 2016], hybrid precision layer design [Chakraborty *et al.*, 2020], pruning [Frankle and Carbin, 2018] [Han *et al.*, 2015], and knowledge transfer [Ahn *et al.*, 2019], to name a few. DNNs are known to be over-parametrized - as an example, AlexNet's performance is achieved at 50% of its size [Iandola *et al.*, 2016]. The model compression methods tap into this property of neural networks to achieve a high reduction in size with only a marginal reduction in accuracy. While previous works focus on a single kind of model compression technique for the entire model, we build over the idea of hybrid precision that uses different precisions for different layers to attempt using different techniques for different layers. The intuition is that different layers serve a different, specific purpose in the model for a given task and need to be treated differently. Especially when it comes to a transfer learning setup, there exist methods that freeze some layers before fine-tuning on the target dataset and achieve equal, if not better, performance, and probably exhibit different characteristics of parameter values from other layers. Though some of the existing works like information distillation [Ahn *et al.*, 2019] and SqueezeNet [Iandola *et al.*, 2016] attempt to minimizing model architecture, we focus on what needs to happen for a given, fixed model.

In this work, we perform a comparative study of various important model compression techniques like integer quantization, floating-point quantization, quantization aware training, pruning, and weight clustering by weighing their pros and cons. We then explore mixed precision quantization that achieves a very high size reduction of some layers and propose a novel approach to carry out the process of hybrid precision design to achieve better performance. Finally, we also propose a pipeline of model compression that utilizes a combination of well-suited methods by extending the novel criteria introduced for mixed precision.

Though this work is done through the lens of CNNs for image classification tasks, the need for compression techniques and the methods themselves can very much be extended to other important tasks that edge devices handle, including Natural Language Processing (NLP), Robotics, etc.

CHAPTER 2

LITERATURE REVIEW & THEORY

In this chapter, we provide a background on the relevant techniques and theory that was explored as a part of our literature survey. We looked at the need for decentralized storage **??** and computation in the introduction that motivates our problem statement. We now proceed to explore the various classical and important techniques that have been used for neural network compression and conclude this chapter with a short introduction to intrinsic dimension and two ID estimation approaches that we focus on in this work.

2.1 Quantization

Quantization is the approach of performing some or all the computations and storing values at lower bit-widths than the conventional 32-bit floating-point (fp32) precision. This leads to compact model representations and fast operations, achieving reduced latency and storage. Neural networks show tremendous scope for quantization despite the high sensitivity of training and inference to the parameter values since they are over-parametrized. This additional degree of freedom renders the model robust to high differences between quantized and original unquantized models without significant degradation in generalizability, especially when the training process is aware of the quantization methods used.

2.1.1 Mathematical Formulation

Suppose there is a neural network with L layers and layer-wise parameters θ_l for $l \in \{0, \dots, L-1\}$. The quantization function for l^{th} layer Q_l is a map from $\mathbb{R}^d \to S^d$, where d is the flattened dimension of the parameter vector θ_l and S is a smaller set of values that a value of low precision can take (though the initial domain values are by itself only of fp32 precision, we treat them as $\in \mathbb{R}$ for simplicity). Model quantization attempts the following:

$$\min_{Q_0,\dots,Q_{L-1}} \sum_{l=0}^{L-1} ||Q_l(\theta_l) - \theta_l|| = \sum_{l=0}^{L-1} \min_{Q_l} ||Q_l(\theta_l) - \theta_l||$$

Each term in the above summation can be minimized independently since there are no dependencies among the functions and the parameters of different layers. This is also the basis for mixed/hybrid precision quantization (section 2.1.3).

2.1.2 Classification

There are a number of classifications used for quantization techniques, a few relevant of which are listed below:

A. ***Uniform and Non-Uniform Quantization***
Uniform/Affine quantization maps a given range of values $x \in [a, b]$ linearly to a set of discrete integers x_q. The process of quantization can be mathematically described as:

$$x_q = round\left(\frac{x}{c} - d\right)$$

And the process of de-quantization (converting the integers x_q back to the source domain of x) as:

$$\widetilde{x} = c(x_q + d)$$

($x \neq \widetilde{x}$ because of rounding-off errors)

If x_q assumes continuous integers in $[a_q, b_q]$,

$$c = \frac{b-a}{b_q - a_q} \qquad (c : \text{scaling factor})$$
$$d = \frac{ab_q - ba_q}{b-a} \qquad (d : \text{zero point})$$

Non-uniform quantization does not follow a linear map and is realized through several techniques like clustering and logarithmic quantization. Even the optimal quantization scheme proposed in Chatterjee and Varshney [2017] based on functional high-rate quantization theory adopts a non-linear approach.

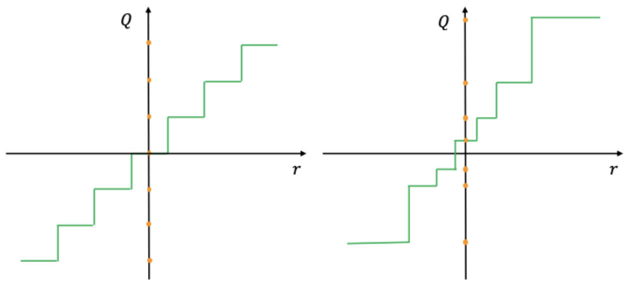


Figure 2.1: Uniform vs Non-uniform Quantization

Source: Gholami *et al.* [2022]

This work focuses on uniform quantization methods since non-uniform ones require more overhead storage and computation, which might not be feasible in edge devices, and the inputs at different times to these devices need not follow identical distributions.

B. ***Symmetric and Asymmetric Quantization***
In symmetric quantization, the source domain of x is symmetric (i.e.) of the form $[-a, a]$, whereas in asymmetric quantization it is not. Within symmetric quantization there are two types: *full range* symmetric quantization where the target domain of x_q is of the form $[-2^{b-1}, 2^{b-1} - 1]$ for b-bit precision using the whole range of possible signed integers vs *restricted range* symmetric quantization where the domain of x is restricted to usually $[-2^{b-1}+1, 2^{b-1}-1]$ for b-bit precision leading to target domain being symmetric and zero point d being equal to zero (has computational benefits during multiplication operation).

In our implementation of hybrid precision quantization introduced later (refer 3.3), restricted range symmetric quantization is used to make full use of the range available since it deals with very low precisions. The general quantization that PyTorch and Tensorflow adopts might be symmetric or

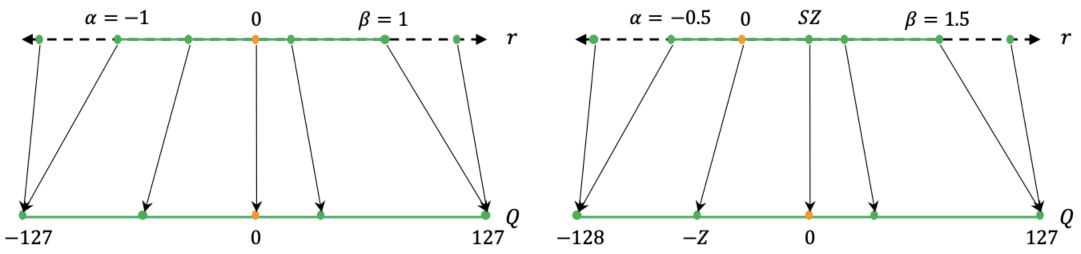


Figure 2.2: Symmetric vs Asymmetric Quantization

Source: Gholami *et al.* [2022]

asymmetric based on the dynamic range of values in the layer/channel it quantizes.

C. ***Static and Dynamic Quantization***
The values to be quantized theoretically can be from a wide floating-point range as supported by the hardware. However, the extreme values rarely occur, and even if they occur, they are very rarely useful. Hence the source domain of x is clipped to $[a, b]$.

Once a model is trained and ready for inference, the weights are fixed and have a fixed clipping range that can be determined. The biases are usually not quantized in any implementation since the mean of the data representation's distribution across layers is highly sensitive to biases. The activations can, however be quantized in two ways: *static quantization* decides the clipping range for the activations by calculating the (min, max, percentile) statistics for a representative dataset before actual inference; *dynamic quantization* decides the clipping range on the fly during forward pass in inference. Static quantization leads to some degradation in performance due to some sacrifice in the authenticity of clipping range, while dynamic quantization falls behind in latency due to large overheads though it is more accurate.

Our hybrid precision quantization implementation uses static quantization to avoid latency increase while simultaneously not allowing performance drops with quantization-aware training (refer D.). The built-in libraries in PyTorch and Tensorflow have both versions.

D. ***Post Training Quantization and Quantization Aware Training (QAT)***
Post-training quantization involves quantizing the model during or just before inference. The parameter values are discretized or reduced in precision by choosing an appropriate scaling factor and zero-point without any modification to the network. There are two types within this: *hybrid quantization*, where the weights alone are quantized, and *full quantization*, where both weights and activations are quantized. QAT operates by fine-tuning the network parameters in such a way that the negative effect of quantization on model performance is minimized. The training in Tensorflow and PyTorch

works by introducing *FakeQuant* layers that simulate low precision behavior in the forward pass and let the backward pass happen at full precision (to prevent vanishing and incorrect gradients), with the gradients calculated wrt the low precision values. Quantization aware training usually leads to better performance in practice at the cost of higher latency in forward-pass simulation and greater memory for storing quantized and non-quantized parameter values while training the model alone. But during deployment, the memory is reduced to the same level as post training quantization.

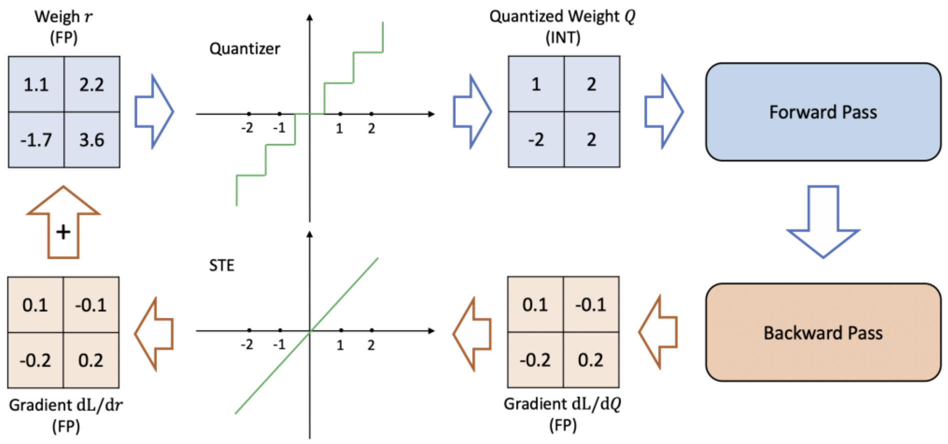


Figure 2.3: Illustration of Quantization Aware Training

Source: Gholami *et al.* [2022]

Our hybrid precision model uses quantization aware training to not yield significant performance drops due to very low precisions (like binary and 2-bit precisions) and realize easier quantized representations of parameters.

Among existing research, there has also been works with a focus on backpropagation: Difference Target Propagation that works for discrete setting [Lee *et al.*, 2015] and low bit-width gradients [Zhou *et al.*, 2016] in DoReFa-Net, to name a few. However, this work does not delve into backpropagation since the task at hand is to deploy a transfer learning model on the edge for inference.

2.1.3 Mixed Precision Quantization

The fact that each layer can be quantized independently has paved the way for mixed/hybrid precision networks. It has been observed that the first and last layers of a DNN are the most important layers that have to be left untouched for any significant accuracy drop to be avoided. This observation led to the introduction of very low precision middle layers that gave a marginal reduction in performance with a huge reduction in size. XNOR-Net proposed by Rastegari *et al.* [2016] is a major breakthrough in this regard.

Binary Quantization: XNOR-Net Design

In XNOR-Net, a layer's weight **W** is approximated by a binary filter **B** whose dimensions are the same as that of **W** (say n), multiplied by a positive scalar scaling factor α at full precision. It also binarizes the activations but without a scaling factor. Mathematically, the objective of the weight quantization is:

$$\begin{aligned} J(\alpha, \mathbf{B}) &= ||\mathbf{W} - \alpha\mathbf{B}||^2 \\ &= \alpha^2\mathbf{B}^T\mathbf{B} - 2\alpha\mathbf{W}^T\mathbf{B} + \mathbf{W}^T\mathbf{W} \\ \alpha^*, \mathbf{B}^* &= \underset{\alpha,\mathbf{B}}{\operatorname{argmin}} J(\alpha, \mathbf{B}) \end{aligned}$$

Since $\mathbf{B} \in \{-1, +1\}^n$, $\mathbf{B}^T\mathbf{B} = n$.

$$J(\alpha, \mathbf{B}) = \alpha^2 n - 2\alpha\mathbf{W}^T\mathbf{B} + \mathbf{W}^T\mathbf{W}$$

Since α is a positive arbitrary constant, $\min J(\alpha, \mathbf{B}) \equiv \max \mathbf{W}^T\mathbf{B}$. That together with the constraint $\mathbf{B} \in \{-1, +1\}^n$ implies $\mathbf{B}^* = sign(\mathbf{W})$.

Similarly the expression for $J(\alpha, \mathbf{B})$ is differentiated wrt α to get:

$$\begin{aligned}\alpha^* &= \frac{\mathbf{W}^T\mathbf{B}^*}{n}\\ &= \frac{\mathbf{W}^T sign(\mathbf{W})}{n}\\ &= \frac{||\mathbf{W}||_{l1}}{n} && \text{($l1$ denotes L1-norm)}\end{aligned}$$

All the intermediate layers except the first and final layer are set to binary precision through the quantization aware training technique where the forward pass is quantized, and backpropagation updates the parameters at full precision through the lens of the quantized values in the forward pass. When the model is deployed for inference, except for the bias and scaling factors for intermediate layers, and parameters of first and last layers, all parameters are stored in binary precision leading to a tremendous reduction in size. The activations are quantized to their *sign* (i.e.) either of $\{-1, +1\}$.

PCA-based Hybrid-Net Design

There was a lack of a systematic way to decide which intermediate layers had to be set to very low precisions to avoid significant performance drops and attain the most optimal trade-off between space reduction and performance drop. Chakraborty *et al.* [2020] proposed a mixed-precision network topology where layers identified as significant are set to k_b-bit precision (k_b 1), and the remaining intermediate layers are set to binary precision having known to be insignificant.

A neural network model is viewed as an iterative projection of input data to higher-dimensional manifolds with the ultimate aim of linear separability, and hence, a layer's significance is identified from the relevance of its transformation towards this objective. A layer is deemed significant if it causes a "significant" increase in the dimensionality of the data representation compared to the layer before it (i.e.) the change in the dimensionality of the data representation caused by the layer is greater than a certain fixed threshold. Principal Component Analysis (PCA) (refer 2.4.1) is used for obtaining the dimensionality of the layers' outputs, defined as the number of components required to explain 99% of the total variance. If the output of a convolution layer has $M \times O \times W \times H$ dimensions, where M is the size of a mini-batch, O is the number of filter maps, W and H are the width, and height respectively, PCA is performed on the channel activations (i.e.) $M * W * H$ activations of dimension O.

Quantizing the model's intermediate layers to low precisions in accordance with their significance minimizes the accuracy drop compared to setting all intermediate layers to binary precision while also minimizing the energy and memory footprint considerably.

2.2 Weight Clustering

Clustering is a weight sharing technique, implemented by Han *et al.* [2015], that attempts to reduce the number of unique weight values in a model. This is achieved by grouping the weights of each layer into a pre-determined number of clusters (say, N_c) and then sharing the cluster's centroid value for all the weights values belonging to the cluster. The technique is very similar to that of model quantiza-

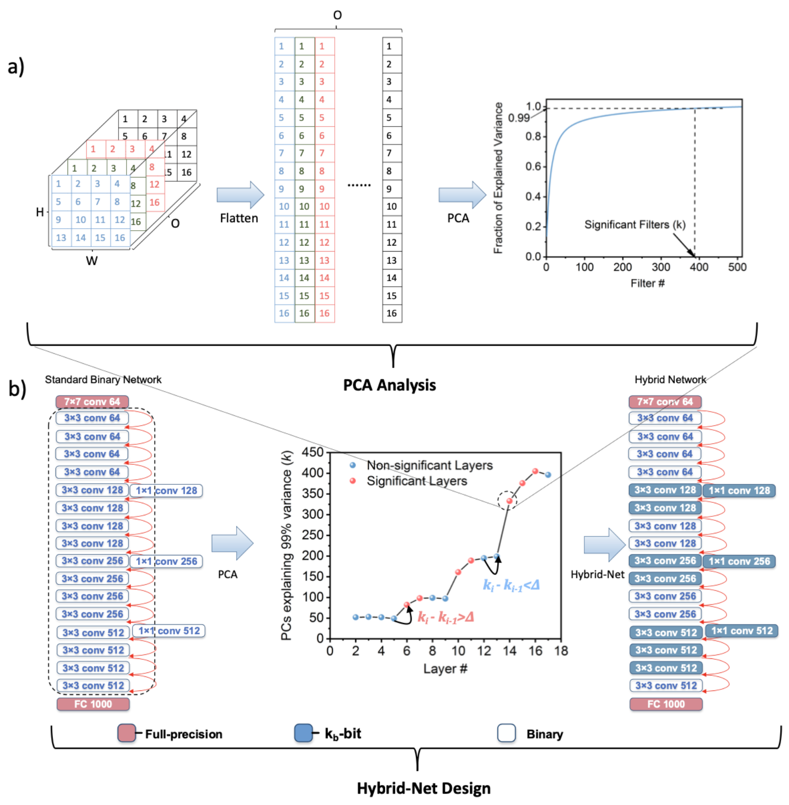


Figure 2.4: Illustration of the working of PCA-based Hybrid-Net design which selectively quantizes different layers independently based on their significance denoted by higher increase in *nPC* from previous layer

Source: Chakraborty *et al.* [2020]

tion in its intuition of reducing the number of unique values and, consequently, the information that is needed to be stored. This weight sharing technique has tremendous benefits when it comes to deployment through efficient model compression for embedded devices with resource crunch. This is achieved by creating a sparse distribution of weights that, in addition to compression benefits during model storage, also has reduced latency with specific hardware support. However, for convolution and dense layers preceding a batch-norm layer, the benefits offered by clustering are diminished.

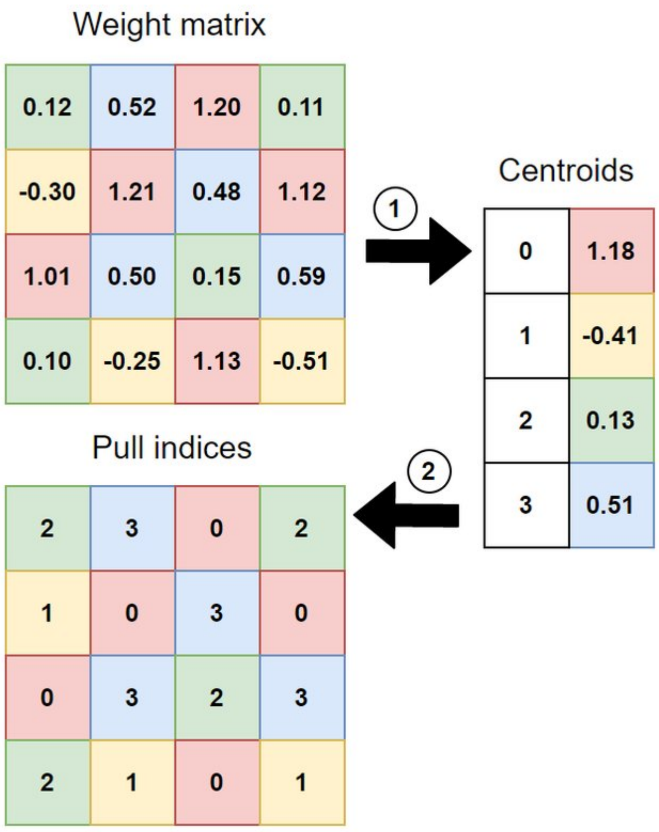


Figure 2.5: Illustration of Weight Clustering

Source: Stoychev and Gunes [2022]

2.3 Pruning

Model pruning is another technique that attempts to achieve an "efficient" model that is optimized in processing time and memory utilization. MIT researchers revealed in their 2019 study, *Lottery Ticket Hypothesis* Frankle and Carbin [2018], that some unnecessary or redundant connections do exist in neural networks that could be removed without affecting the accuracy. This gave birth to the idea of model pruning, which tries to uncover a sub-network within a dense neural network that, when trained in isolation, can achieve the performance of the original model. Model pruning also has a biological motivation - the concept of *synaptic pruning* in many animals, where the axons and dendrites deteriorate and die off between early childhood and the onset of puberty.

A winning ticket is a sub-network that is capable of learning. It is identified by training the given model, zeroing out (i.e.) pruning the parameters of lowest magnitudes, and constructing the sub-network from the remaining un-pruned parameters. The way pruning achieves smaller memory and latency is by creating this sparse winning ticket. Frankle and Carbin [2018] show winning tickets achieving less than 10-20% of the original feed-forward CNNs's size on MNIST and CIFAR-10. With sparse connections, the number of multiplication operations comes down, and with the zeroed-out parameter values, which can be left out while compressing and deploying the model, the size reduces.

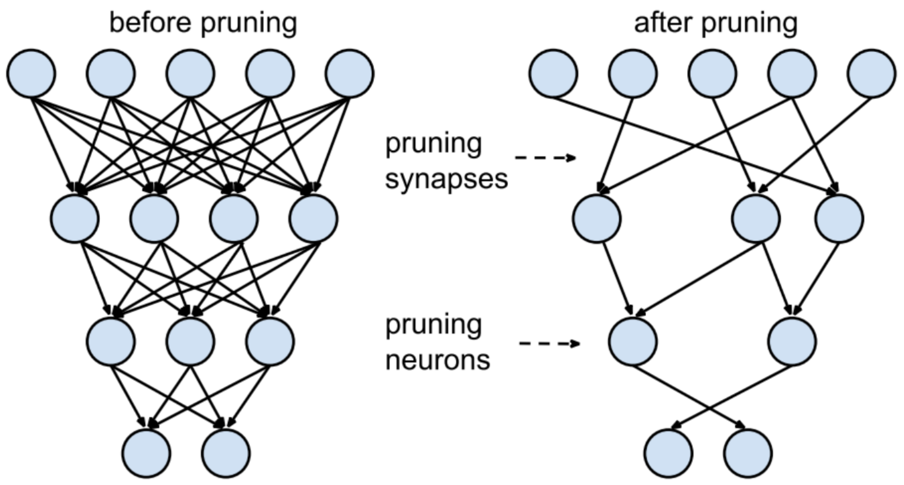


Figure 2.6: Illustration of Weight and Neuron Pruning

Source: [LeCun *et al.*, 1989] [Han *et al.*, 2015]

Pruning approaches differ in a number of ways, including in their focus on latency or memory, but we specifically use two important types in this work:

- ***Unstructured Pruning***: In this approach, the parameter values are individually pruned by setting a proportion. The pruned values need not be arranged in a manner that can be leveraged to achieve faster run-times using present hardware and libraries. Since we set the individual weight values to zero in the weight matrix, this approach is also known as ***weight pruning***.

- ***Structured Pruning***: This approach, also called by the name ***neuron pruning***, employs pruning parameters in groups - entire neurons in a layer, filter, or channel. This makes the network smaller and faster. The neurons are ranked according to L1/L2-norm of the neuron weights, and the lowest-ranked neurons are removed.

2.4 Intrinsic Dimension Estimation

Deep neural networks for supervised image classification, including CNNs, sequentially transform images through the linear and non-linear layers and reduce the representation to a linearly separable form for classification. The analysis of the distribution and dimensionality of data representation across layers offers several key insights to better understand the working of a DNN. The intrinsic dimension (ID) is a fundamental property of data representations and is defined as the minimum number of coordinates required to describe the vectors to satisfactory levels. PCA is the simplest and most convenient form of linear ID estimation.

2.4.1 PCA Method for Intrinsic Dimension Estimation

Principal Component Analysis (PCA) was predominantly developed as a method of dimensionality reduction to preserve as much information with reduced number of explanatory variables. It is based on two equivalent mathematical formulations that both yield the same results:

- Maximizing variance of data projected linearly onto a vector subspace
- Minimizing error when data is projected linearly onto a vector subspace

Optimizing either of the objectives tells us to pick the eigenvectors of the data's covariance matrix in decreasing order of eigenvalues for forming the reduced subspace to project the data. The choice of the number of eigenvectors (called principal components) for the subspace is arrived at by plotting the curve of cumulative explained variance vs the number of principal components. The curve is expected to rise steeply initially and then gradually saturate to 100% after a knee point. The dimensionality is then estimated as the number of principal components required to explain a certain threshold variance (usually > 90%), denoted as *nPC*.

Given the idea of using the difference in PCA across a layer as a measure of the significance of the layer's parameters and subsequently as a proxy to identify layers to quantize by Chakraborty *et al.* [2020], there is a need to explore other dimensionality estimation techniques that overcome the shortcomings of PCA and perform better at this use case. This work explores a better alternative method to estimate the ID of data representations: *TwoNN*.

2.4.2 TwoNN Method for Intrinsic Dimension Estimation

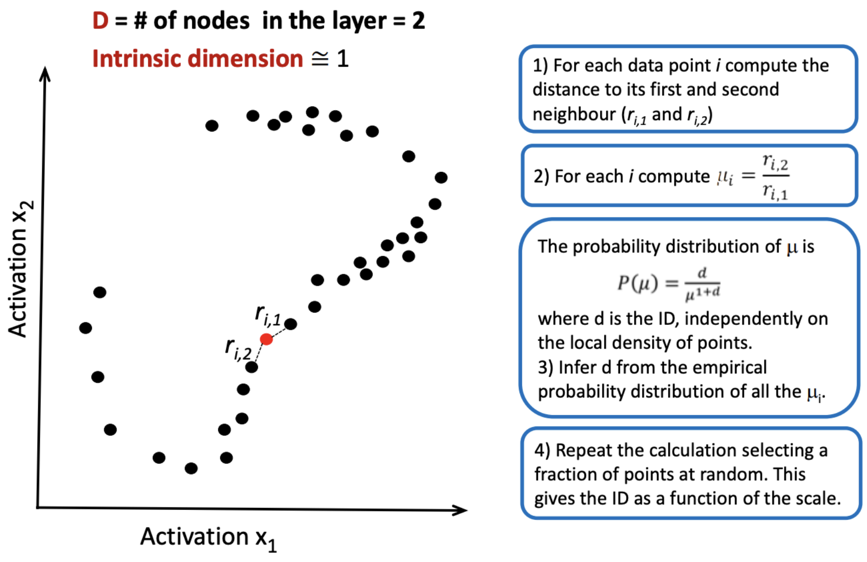
Two Nearest Neighbors (TwoNN) is a recently developed global intrinsic dimension estimator [Facco *et al.*, 2017], which is estimated using the information about merely the two nearest neighbors of every point. The ratio of nearest neighbors statistics $\mu_i = r_i^{(2)}/r_i^{(1)}$ for each point i is calculated where $r_i^{(1)}$ and $r_i^{(2)}$ are the distances to the nearest and second nearest points to point i respectively. Under a weak assumption that the density is constant on the scale of distance between each point and its second nearest neighbor, the distribution of μ_i's depend on the dimensionality of the data rather than the density. Specifically, it is derived that μ_i's follow *Pareto distribution* with parameter $d+1$ (d is the ID) on $[1, \infty)$. The Pareto distribution's CDF is given by:

$$F_X(x) = \begin{cases} 1 - \left(\frac{1}{x}\right)^{d+1} & x \geq 1, \\ 0 & x < 0 \end{cases}$$

With the knowledge about the distribution of data, the ID estimation is modeled as an objective of maximizing the log likelihood

$$P(\boldsymbol{\mu}|d) = d^N \prod_{i=1}^{N} \mu_i^{d+1}$$

where $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_N]$.



D = # of nodes in the layer = 2
Intrinsic dimension ≅ 1
Activation x$_2$
Activation x$_1$
$r_{i,1}$
$r_{i,2}$
1) For each data point *i* compute the distance to its first and second neighbour ($r_{i,1}$ and $r_{i,2}$)
2) For each *i* compute $\mu_i = \frac{r_{i,2}}{r_{i,1}}$
The probability distribution of μ is $P(\mu) = \frac{d}{\mu^{1+d}}$ where d is the ID, independently on the local density of points.
3) Infer d from the empirical probability distribution of all the μ_i.
4) Repeat the calculation selecting a fraction of points at random. This gives the ID as a function of the scale.

Figure 2.7: Illustration of ID estimation through TwoNN

Source: Ansuini *et al.* [2019]

The minimal neighborhood size that the ID estimator relies on lowers the effect of data inhomogeneities in the estimation process. In addition, the theoretical setup of the estimation process and absence of usage of linear projections and assumptions make the method robust to curved, topologically complex data and non-uniform sampling of the data points. Ansuini *et al.* [2019] adopts the TwoNN approach for estimating and analyzing ID as a proxy to study the generalizability of neural networks. Their experiments also suggest that the TwoNN method is approximately scale-invariant, unaffected by embedding dimension (hence can be used to analyze data in any type of layer), and robust to hubs and outliers as long as enough data points are used for the estimation process.

CHAPTER 3

RESEARCH METHODS

3.1 Problem Statement Motivation

Image classification refers to labeling an input image as one of the many given pre-defined classes. It has become one of the most important tasks in digital image analysis and computer vision. Advancements in image classification have led to several applications like visual search, automated image organization, and facial recognition on social media. As mentioned earlier, neural networks have overtaken traditional machine learning methods with increasing computational power due to superlative performance. However, their requirement of high processing power and memory poses a hurdle to their use in edge devices. For instance, some state-of-the-art CNN models for image classification tasks, such as GoogeLeNet [Szegedy *et al.*, 2014] and AlexNet [Krizhevsky, 2009*a*] necessitates storing more than tens of millions of parameters and around a billion multiply and accumulate computations (MACs) [Shafique *et al.*, 2018]. These MAC operations account for over 99% of the total operations in these CNNs and are the dominant source of energy consumption and processing time, as stated by Yang *et al.* [2017]. Consequently, the computational and energy requirements of a model scale almost proportionally with the number of MAC operations in the worst case, discarding data reuse. When energy is normalized in terms of energy of a MAC operation, a relatively non-complex model like AlexNet consumes 3.97×10^9 energy. Storing such state-of-the-art models with such magnitude of parameters all require at least

50MB memory [Crefeda Rodrigues *et al.*, 2020]. This huge memory and energy footprint of these models discourages the use of deep neural networks, including CNNs, in small embedded devices and IoT sensors.

If the energy consumption of the IoT devices that behave as data collecting interfaces and transmit data to the cloud computing systems is significantly more than what it takes to classify images on the device hardware, it becomes more reasonable to run the computations on the device with limited resources. Added to that, for some real-time applications like self-driving cars, the delay that arises due to the communication with the centralized cloud server poses a huge risk to the safety of the passengers. The rapid scaling of such edge devices that connect to the same centralized server also perpetuates the need for decentralized systems and computations.

The contributions of this work are two-fold:

1. We first perform a comparative study of the classical and common existing methods for compressing deep neural networks, with a focus on convolutional neural networks for the image classification task in standalone and transfer learning settings. Techniques like Quantization, Mixed Precision Quantization, Weight Clustering, and Pruning are explored, and their advantages studied. We believe this will be helpful to the research community in deciding the approach that is most appropriate for the memory, cost, and performance considerations of the edge device.
2. We propose a novel neural network hybrid layer compression technique for very low memory and low memory setting each that can be used to achieve a reduction in model size without a considerable drop in performance.

3.2 Description of Model used for Studies - QuantNet

The convolutional neural network model for image classification tasks used throughout the work is **QuantNet**, which comprises two 2D convolution layers, followed by two fully connected dense layers, and a final output dense layer that predicts the label class the image belongs to.

Input image of size $32 \times 32 \times 3$ is fed into the model. The first two Conv2D layers are both initialized using *He normal* initialization, have *ReLU* activations, have kernel size of 3×3, have stride of 2, and padding set to same (in Tensorflow terms, “same” padding results in output size of $\lceil \frac{I}{S} \rceil$ where I is the input size and S is the stride). There is a dropout of 0.1 used after the two Conv2D layers as a regularization measure. The first conv2D layer has 32 output channels and the second Conv2D layer has 64 output filter maps, which capture various spatial features from the input image. These output feature maps are then flattened and passed to two dense layers, which further process the features to extract higher-level features that make the data more separable for the final output layer. The two fully connected layers are both initialized using *He normal* initialization, have *ReLU* activations, and have output dimensions of 128 and 16, respectively. The final output layer is again a dense layer initialized using *Glorot normal* initialization and with output dimensions equal to the number of target classes for the image classification task. The activation used for the final layer depends on the number of target classes: *sigmoid* function for binary classification tasks to predict the probability of image belonging to class 1 and *softmax* function for multi-class classification tasks to obtain the probability distribution for the image belonging to each class. The output from the model is used to infer the predicted class by taking an `argmax` of the output logits. Figure 3.1 illustrates a detailed architecture of the QuantNet model for a binary classification task.

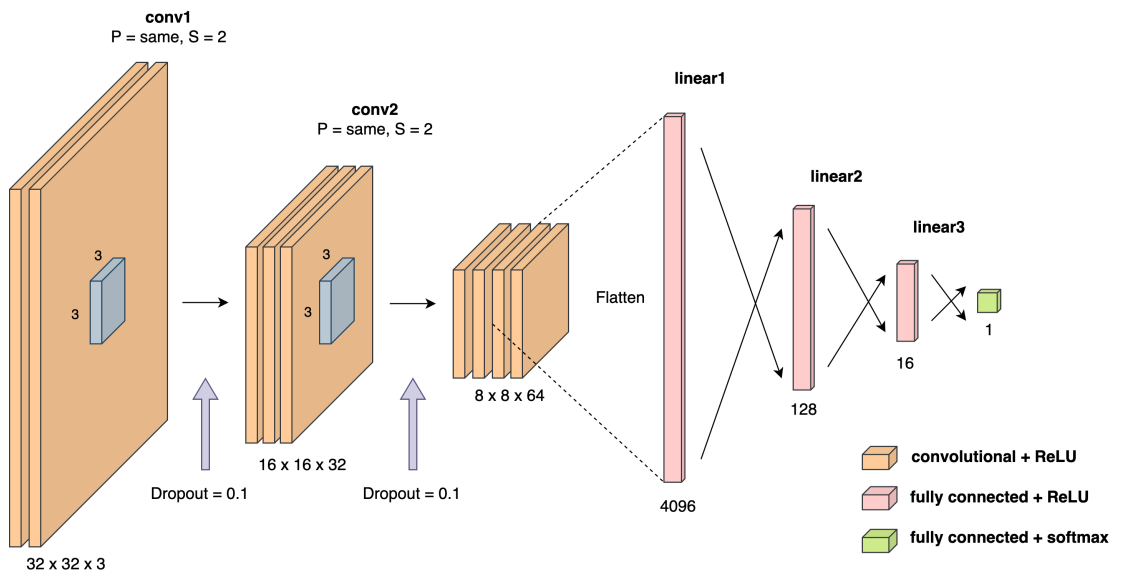


Figure 3.1: Detailed Model Architecture of QuantNet for binary image classification task

A plethora of past research works has demonstrated that the traditional neural network model compression techniques generate similar trends among models of different sizes and architectures. Models like ResNet with different depths of 50, 100, and 150, XNOR-Net are observed to follow alike trends by Liu *et al.* [2021], which corroborates the claim. Even the work on Hybrid-Net design by Chakraborty *et al.* [2020] suggest similar effects and results for different models like ResNet-18, ResNet-20, ResNet-32, and VGG-15. These observations allow us to focus on the analysis of different model compression algorithms on the single QuantNet model and extrapolate the results to models of different architecture and/or sizes.

Though the model employed by this work is a simple CNN with conv2D and dense layers, the model is sophisticated enough to extract various levels of features from the input image. The initial layers capture the general characteristics of the dataset (like face, nose, ear) and remove features irrelevant to the final prediction. The subsequent layers extract information that is specific to the learning task of image classification. When it comes to a transfer learning setup, the parameters learned on the source dataset facilitate learning on the target dataset through the relevance of some features of the source domain to the target domain, and the new features' extraction requiring only a small fine-tuning of the existing parameters. Hence, transfer learning saves a vast amount of time and computation by converging the training faster and learning better representations of the target domain inputs, having prior knowledge about what it did to the source domain.

3.3 TwoNN-based Hybrid Precision Quantization

We implement our own hybrid precision quantization methodology and propose a new Hybrid-Net design that improves over the current best Hybrid-Net design [Chakraborty *et al.*, 2020] in terms of correct identification of significant layers.

The model layers are quantized individually with different precisions based on their significance - first and last layers are set to full precision, insignificant layers are set to binary precision, and the significant layers are set to a higher *b*-bit precision, with the choice of *b* depending on the resource of the edge device and the performance level we expect. The quantization takes place through the quantization aware training principle where the forward pass happens at the determined mixed precisions while the backpropagation happens at full precision, wary of the quantized values in the forward pass. Every quantized layer's incoming activations are statically clipped to $[-1,+1]$ range and uniformly quantized within the range to the same precision of the layer that it passes through. Using any other range for clipping doesn't perform as well as $[-1,+1]$ and at times causes the model to diverge during training. Also, there are no scaling factors used for multiplying the quantized activations before they pass since they have to be evaluated dynamically during inference and cause unnecessary overheads, increasing latency. Hence, mathematically the quantization of activation A can be described as:

$$\widetilde{\mathbf{A}} = \begin{cases} \mathbf{A} & \mathbf{A} \text{ is not quantized} \\ sign(\mathbf{A}) & \mathbf{A} \text{ is binary precision quantized} \\ uniform\_dequant_b\,(uniform\_quant_b(\mathbf{A})) & \mathbf{A} \text{ is b-bit quantized} \end{cases}$$

The weights are mean centered, clipped to $[-1,+1]$ range, then quantized uniformly in a full range symmetric fashion, and dequantized to obtain back the discretized steps in $[-1,+1]$. It is then multiplied by a scaling factor (like in XNOR-Net) before being used for computations/forward pass.

$$\widetilde{\mathbf{W}} = \begin{cases} \mathbf{W} & \mathbf{W} \text{ is not quantized} \\ \alpha^{*} sign(\mathbf{W}) & \mathbf{W} \text{ is binary precision quantized} \\ \alpha^{*}\ uniform\_dequant_b\,(uniform\_quant_b(\mathbf{W})) & \mathbf{W} \text{ is b-bit quantized} \end{cases}$$

The mean centering and clipping of weights are like additional forms of restriction on the model that prevents skewed distributions and facilitates training convergence. The symmetric full range quantization ensures that the whole range of a

b-bit precision is used effectively since we are dealing with very low bit precisions. Now, let us see focus on two key components of the implementation:

3.3.1 Correct Scaling Factor and Gradients

The optimal scaling factor α^* was obtained by minimzing the following expression wrt α after we found $\mathbf{B}^*$:

$$\begin{aligned} J(\alpha, \mathbf{B}^*) &= ||\mathbf{W} - \alpha\mathbf{B}^*||^2 \\ &= \alpha^2\mathbf{B}^{*T}\mathbf{B}^* - 2\alpha\mathbf{W}^T\mathbf{B}^* + \mathbf{W}^T\mathbf{W} \\ \alpha^* &= \underset{\alpha}{\operatorname{argmin}} J(\alpha, \mathbf{B}^*) \end{aligned}$$

$$\mathbf{M}^* = \begin{cases} \mathbf{W} & \mathbf{W} \text{ is not quantized} \\ sign(\mathbf{W}) & \mathbf{W} \text{ is binary precision quantized} \\ uniform\_dequant_b\,(uniform\_quant_b(\mathbf{W})) & \mathbf{W} \text{ is b-bit quantized} \end{cases}$$

Now since $\mathbf{B}^*$ is no longer $\in -1, +1^n$ (taking the general case, it can have b-bit precision), differentiating $J(\alpha, \mathbf{B}^*)$ wrt α and equating to zero, we get optimal scaling factor as:

$$\alpha^* = \frac{\mathbf{W}^T\mathbf{B}^*}{||\mathbf{B}^*||_{l2}} \qquad (l2 \text{ denotes L2-norm})$$

However, this corrected scaling factor affects the convergence of training and either converges to a local optima or takes extra epochs with properly tunes learning rates to achieve same, if not better, performance as compared to $\alpha^* = \frac{||\mathbf{W}||_{l1}}{n}$. If the range of quantized outputs is anything other than $[-1, +1]$, the corrected scaling factor achieves better accuracy and loss. Hence, given the fact that our quantization output's range is $[-1, +1]$, we stick with the optimal scaling factor derived for the binary precision case for the higher precisions as well. Let us next look at the gradients for the backward pass.

Let H be the cost function with which the neural network is trained. The XNOR-Net design uses the following expression for gradients wrt the quantized weights during the backpropagation:

$$\begin{aligned}\frac{\partial H}{\partial W_i} &= \frac{\partial H}{\partial \widetilde{W_i}} \cdot \frac{\partial \widetilde{W_i}}{\partial W_i} \\ &= \frac{\partial H}{\partial \widetilde{W_i}} \cdot \frac{\partial\left(\alpha \cdot \operatorname{sign}\left(W_i\right)\right)}{\partial W_i} \\ &= \frac{\partial H}{\partial \widetilde{W_i}} \cdot \left[\operatorname{sign}\left(W_i\right) \cdot \frac{\partial \alpha}{\partial W_i} + \alpha \cdot \frac{\partial \operatorname{sign}\left(W_i\right)}{\partial W_i}\right] \\ &= \frac{\partial H}{\partial \widetilde{W_i}} \left[\frac{1}{n} + \frac{\partial \operatorname{sign}\left(W_i\right)}{\partial W_i} \cdot \alpha\right]\end{aligned}$$

In our implementation of backpropagation, we use the following corrected version which takes into account the dependency of the optimal scaling factor α^*

on other weight values:

$$\begin{aligned}
\frac{\partial H}{\partial W_i} &= \sum_{j=1}^{n} \left(\frac{\partial H}{\partial \widetilde{W}_j} \cdot \frac{\partial \widetilde{W}_j}{\partial W_i} \right) \\
&= \sum_{j=1}^{n} \left[\frac{\partial H}{\partial \widetilde{W}_j} \cdot \frac{\partial \left(\alpha \cdot \operatorname{sign}\left(W_j \right) \right)}{\partial W_i} \right] \\
&= \sum_{j=1}^{n} \left[\frac{\partial H}{\partial \widetilde{W}_j} \cdot \left(\operatorname{sign}\left(W_j \right) \cdot \frac{\partial \alpha}{\partial W_i} + \frac{\partial \operatorname{sign}\left(W_j \right)}{\partial W_i} \cdot \alpha \right) \right] \\
&= \sum_{j=1}^{n} \left[\frac{\partial H}{\partial \widetilde{W}_j} \cdot \operatorname{sign}\left(W_j \right) \cdot \frac{\partial \alpha}{\partial W_i} \right] + \frac{\partial H}{\partial \widetilde{W}_i} \cdot \frac{\partial \operatorname{sign}\left(W_i \right)}{\partial W_i} \cdot \alpha \\
&= \frac{\partial \alpha}{\partial W_i} \cdot \sum_{j=1}^{n} \left[\frac{\partial H}{\partial \widetilde{W}_j} \cdot \operatorname{sign}\left(W_j \right) \right] + \frac{\partial H}{\partial \widetilde{W}_i} \cdot \frac{\partial \operatorname{sign}\left(W_i \right)}{\partial W_i} \cdot \alpha \\
&= \frac{1}{n} \cdot \operatorname{sign}\left(W_i \right) \cdot \sum_{j=1}^{n} \left[\frac{\partial H}{\partial \widetilde{W}_j} \cdot \operatorname{sign}\left(W_j \right) \right] + \frac{\partial H}{\partial \widetilde{W}_i} \cdot \frac{\partial \operatorname{sign}\left(W_i \right)}{\partial W_i} \cdot \alpha
\end{aligned}$$

3.3.2 TwoNN Method for Identifying Significant Layers

The two main disadvantages of PCA in the context of analyzing data representation in neural networks are:

1. The output from a few layers might have a sparse set of outliers than don't affect the final results but are caused due to noise in input and/or irrelevant neurons in the over-parametrized neural network models.
2. The data passes through a series of linear weights and non-linear activations that might cause the data to form curved manifolds, especially in the middle layers (since the effect of non-linearity sets in after the activation following the first layer and the model's objective is to make the data linearly separable for classification in the last layer; the objective of middle layers are not well studied).

Hence, we propose the usage of TwoNN intrinsic dimension estimation method (refer 2.4.2) for the purpose of identifying significant layers in a neural network and subsequently quantizing each layer accordingly. The method is robust to non-linear data manifolds and is approximately insensitive to outliers.

The study of Ansuini *et al.* [2019] suggests that for a general neural network, the intrinsic dimensions of data representations across layers follow a characteristic hunchback shape as seen in figure 3.2. The hypothesis is that the initial layers of a neural network prune the highly correlated features irrelevant to the final predictions (like luminescent gradients, contrast, saturation, etc.) by projecting them onto curved manifolds with higher IDs, and the subsequent layers do the advanced processing of making the data linearly separable at final layer for correct predictions, which decreases the IDs.

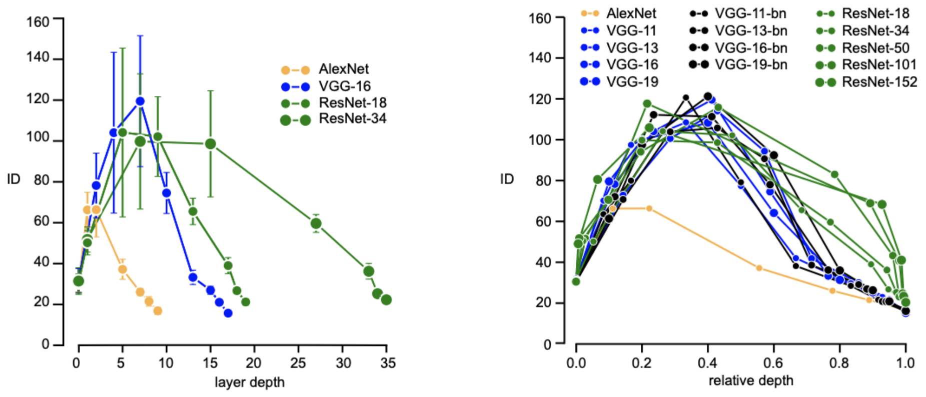


Figure 3.2: The ID of data representations following a hunchback profile

Source: Ansuini *et al.* [2019]

Following this intuition, we come up with a new criteria for identifying significant layers:

$$|ID_l - ID_{l-1}| > \Delta \implies \text{significant layer}$$

where ID_i is the intrinsic dimension of data output from layer i, and Δ is the significant threshold that is determined according to our needs for size reduction trading-off with performance.

Also, the TwoNN analysis is performed on the vector of dimension $O * W * H$ obtained by flattening the entire output of a convolution layer instead of following the procedure on channel outputs of dimension O. The reason is each filter extracts a different feature, and each pixel within a filter is needed to properly characterize the filter and, subsequently, the data representation corresponding to a given image at the end of this layer. The linear layer gives a flattened output; hence there are no ambiguities there. For a standalone setup, TwoNN analysis is performed on a mini-batch of the source dataset while for a transfer learning setup, a mini-batch of the target dataset is used, both on the model pre-trained using the source dataset. The application of the ID estimation method on our QuantNet model for the CIFAR-10 dataset is shown in figure 3.3. The experiment corroborates the intuition of the characteristic hunchback profile for our setting as well.

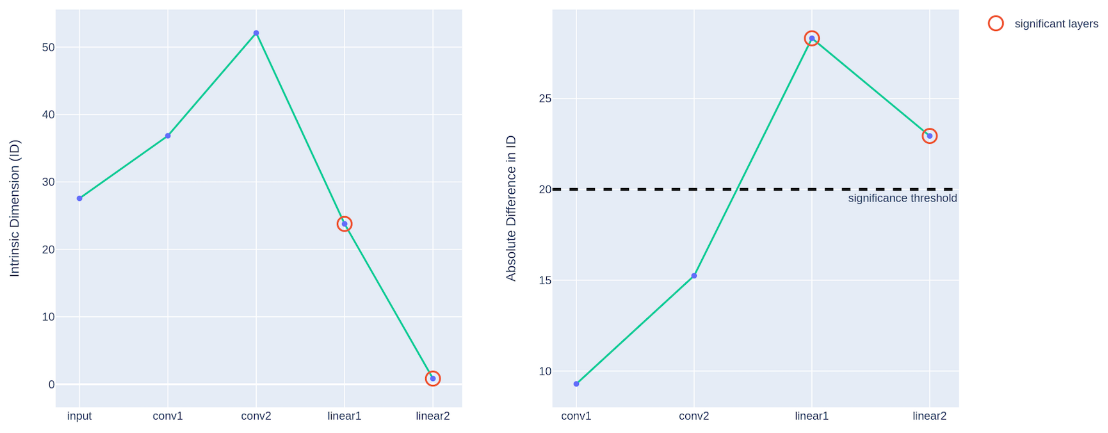


Figure 3.3: Illustration of identifying significant layers using TwoNN method on QuantNet model for the CIFAR-10 dataset, with threshold $\Delta = 20$

3.4 Proposed Model Pipelines

Based on the results from our comparative study that we shall see in the next chapter, we propose a model pipeline for limited memory edge devices as follows:

- ***For standalone setup***: The model is trained on the dataset, and subsequently, the significance of layers is analyzed using the TwoNN approach outlined in section 3.3.2. On the insignificant layers, weight clustering is employed, and on the significant layers, cluster-preserving QAT is used.
- ***For transfer learning setup***: The model is pre-trained on the source dataset, and the significance of layers is analyzed using the TwoNN approach by passing a mini-batch of the target dataset on the pre-trained model. On the insignificant layers, weight clustering is employed, and on the significant layers, cluster-preserving QAT is used.

The choice of vanilla clustering method for the insignificant layers is due to the presence of a systematic method to choose the number of clusters and the memory reduction that follows. We also prefer it over very low precision quantization since clustering creates a sparser weight distribution that incorporates the information from original weight distribution through the centroid calculation and does not limit to uniform steps. Cluster-preserving QAT was chosen for significant layers because QAT preserves the desirable precision for such important parameters, and this clustering-preserved QAT specifically achieves the best test performance with optimized memory (refer section 4.2.3). A pictorial summary of the model pipeline we propose in this work for transfer learning setup is shown in figure 3.5. For a standalone setup, the target dataset will be the same as the source dataset, which is why the pipeline is more naturally applicable to transfer learning setups.

For edge devices with extreme memory constraints, we propose a hybrid precision model design: the model is trained on the source dataset, and then its layers' significance is analyzed using the TwoNN approach. The insignificant layers are set to binary precision, and significant layers are set to slightly higher precision (we choose 2-bit due to memory limitation). Finally, the model is trained on the

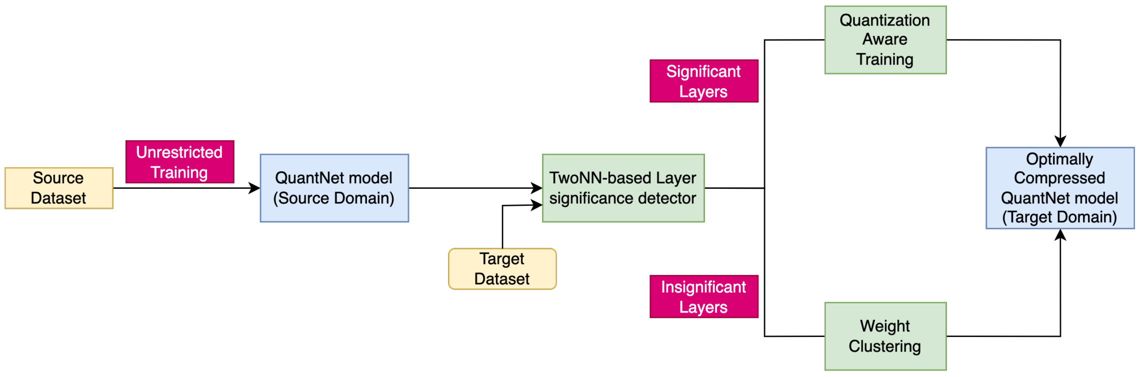

Source Dataset
Unrestricted Training
QuantNet model (Source Domain)
TwoNN-based Layer significance detector
Target Dataset
Significant Layers
Insignificant Layers
Quantization Aware Training
Weight Clustering
Optimally Compressed QuantNet model (Target Domain)

Figure 3.4: Flow diagram of the optimal model pipeline proposed for limited resource edge devices

source dataset again from scratch with the quantization for a standalone setup. For a transfer learning setup, the model is fine-tuned on the target dataset with this mixed quantization. We do not attempt introducing other techniques like clustering for insignificant layers here due to the advantage of mixed layer precision training happening in one go, but it is an idea to explore in the future.

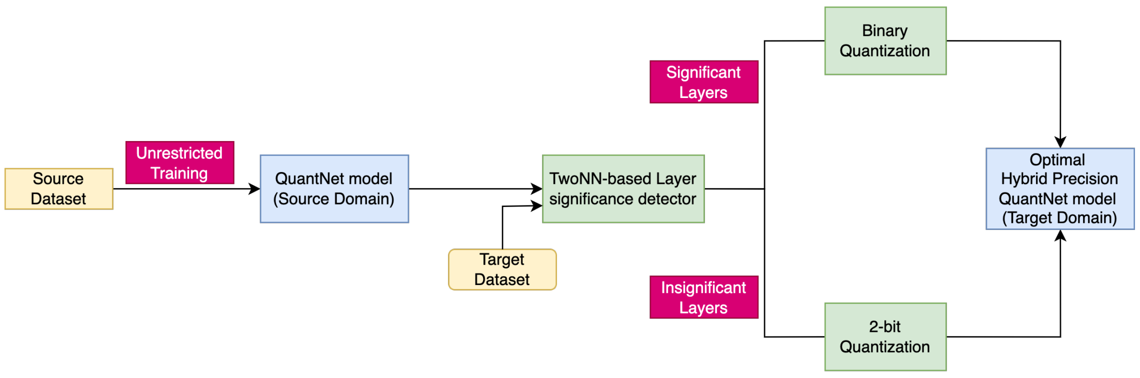

Source Dataset
Unrestricted Training
QuantNet model (Source Domain)
TwoNN-based Layer significance detector
Target Dataset
Significant Layers
Insignificant Layers
Binary Quantization
2-bit Quantization
Optimal Hybrid Precision QuantNet model (Target Domain)

Figure 3.5: Flow diagram of the optimal model pipeline proposed for memory-constrained edge devices

CHAPTER 4

EXPERIMENTS

4.1 Setup

4.1.1 Dataset

We use the QuantNet model described thoroughly in section 3.2 for all analysis with a binary image classification head and sigmoid activation in the output layer (refer figure 3.1). The details of the exact parameter count of the model layers are mentioned in table 4.1.

Table 4.1: Summary of QuantNet model used in the experiments

Layer	**Trainable Params**
conv1	896
conv2	18,496
linear1	534,416
linear2	2,064
linear3	17
Total	545,889

All the experiements in this work are performed on the publicly available CIFAR-10 dataset[1] released by Krizhevsky [2009*b*]. The dataset contains 50,000 training set of images and 10,000 test set of images, all colored and of dimensions

[1]Link: https://www.cs.toronto.edu/ kriz/cifar.html

$32 \times 32 \times 3$. As the name suggests, the images in the dataset belong to the following 10 classes:

- Airplane
- Automobile
- Ship
- Truck
- Bird
- Cat
- Deer
- Dog
- Horse
- Frog

Both the training and test sets have equal representation of all 10 classes (i.e.) the training set has 5,000 images of each class, and the test set has 1,000 images of each class. The test set is created by randomly sampling those 1000 images of each class from a total of 60,000 images. The images in the training set are randomly shuffled to avoid over-fitting to some classes within the first few iterations of a mini-batch, which might cause convergence to a local minima. Finally, all classes are mutually exclusive in the domains they capture and the labels they represent.

We filter both the training and test sets for two different yet closely related pairs of labels to be used for the experiments:

1. **Cat vs Dog** - This binary classification dataset is used for standalone experiments and as the source domain on which model is pre-trained before fine-tuning on the target domain for transfer learning setup.
2. **Horse vs Frog** - This binary classification dataset is used for standalone experiments and as the target domain on which the model is fine-tuned for transfer learning setup.

In each of the above filtered datasets, we randomly split the training dataset in 80:20 ratio to get the actual training and validation sets, respectively, to be used for hyperparameter tuning and training of the models.

4.1.2 Hyperparameter Tuning

As stated before, DNNs are over-parametrized models that are sensitive to the choice of hyperparameters used for training a model. These hyperparameters cannot be obtained directly by training a model but rather arrived at by searching through a limited space of values. We attempt this hyperparameter tuning to get the values for which the baseline model achieves the best performance so that we will be able to get distinct, accurate, and reliable from the further compression experiments.

Though there are a number of other methods for hyperparameter search like Grid search and Bayes search, we choose to go with Random search based on the theoretical backings from Bergstra and Bengio [2012], which using a Gaussian process analysis of the function from hyper-parameters to validation set performance, divulge that only a few of the hyper-parameters really matter for most data sets. Hence, we minimize the resources spent on tuning these hyperparameters by adopting random search, for which the search space used is mentioned in table 4.2. Apart from those mentioned in the table, we also manually run a search on the type of parameter initializer to use out of *He Normal*, *Glorot Normal*, *Glorot Uniform*, and *He Uniform*. Based on this, *Glorot Normal* was chosen for the output layer that uses sigmoid activation, and *He Normal* was chosen for all other layers.

4.1.3 Model Training Details

We use Adam optimizer for training the QuantNet for the experiments that follow, with a constant learning rate of 0.001. The loss function used is the Binary Cross-

Table 4.2: Hyperparameter setting of QuantNET

Parameter	**Range**	**Optimal Value**
Filters (Conv Layer 1)	Range: [16, 128]; Step size = 16	32
Filters (Conv Layer 2)	Range: [16, 128]; Step size = 16	64
Kernel Size (Conv Layers)	[3,5,7,9]	3
Dropout	[0.1, 0.2, 0.3, 0.4, 0.5]	0.1
Optimization Function	Adam, SGD, RMSProp	Adam
Learning Rate	[1e-5, 1e-4, 1e-3, 1e-2, 1e-1]	0.001
Number of epochs	Range: [10, 100]; Step size = 10	100
Batch size	[32, 64, 128, 256, 512]	64

Entropy Loss since the task is of binary classification. The first two convolution layers of the model generate output channel sizes of 32 and 64, respectively. The model is trained for 100 epochs coupled with early stopping based on test accuracy with patience parameter 5. We restrict to using Top-1 accuracy as the metric instead of Recall, F1-score, etc., since the dataset is balanced. All the experiments were done on Google Colab Notebook with an NVIDIA Tesla K80 GPU.

4.2 Results and Findings

The values given in the section are the values averaged over 20 independent runs.

4.2.1 Study on Conventional Quantization Approaches

Quantization was introduced in section 2.1 as a means to reduce the precision of parameters and corresponding activations in neural networks that originally had high space and computational complexity. The conventional implementations of quantization involve reducing precision from fp32 to 16-bit floating-point/integer and 8-bit integer values. Such compressions have a number of benefits apart from mere size reduction - integer operations consume lesser energy than FLOPs, hardware is made less complex, the number of off-chip memory access is reduced, and consequently, the communication cost and latency is decreased.

We first run a set of quantization methods separately on the Cat vs Dog and Horse vs Frog datasets and evaluate the test performance for this **standalone experiment** without any transfer learning. The results of this experiment are shown in table 4.3. Note that post-training quantization does not involve training the model again.

Table 4.3: Test performance and model size of QuantNet under different quantization schemes on Cat vs Dog and Horse vs Frog datasets (standalone)

Set Up	**Test Acc (Cat vs Dogs)**	**Test Acc (Horse vs Frogs)**	**Model Size (MB)**
Base Model	66.00	92.35	2.0852
Post Training - Float 16 Quantization	64.65	91.65	1.0453
Post Training - Dynamic Range Quantization	64.55	91.30	0.5282
Post Training - Integer Quantization	63.70	90.25	0.5271
Post Training - Int Quantization + Int 16 Activation	64.80	92.05	0.5292
Quantization Aware Training	65.90	93.95	4.1784
Post Training Quantization on QAT model	65.80	93.90	0.5278

Simple 8-bit integer post training quantization leads to the least accuracy since it loses the most information without the model being fine-tuned to be prepared for mitigating this information loss. With only quantization aware training (QAT), we observe that the model size almost becomes double that of the baseline because parameters are stored at both low and original precisions. But it also exhibits the highest performance among all techniques. Hence, we run **post training integer quantization on the quantization aware trained model** to achieve the best trade-off between accuracy and model size.

Now, let us venture into the **transfer learning setup**, which is our primary focus for improvements. The same experiment as above is repeated, but this time after training on Cat vs Dog (source domain), we fine-tune on Horse vs Frog (target domain) by transferring knowledge learned from the Cat vs Dog dataset. Post training quantization happens at the end of the fine-tuning, and QAT runs just after the pre-training step (during the fine-tuning).

Table 4.4: Test performance and model size of QuantNet under different quantization schemes with Cat vs Dog source domain, Horse vs Frog and Rotated Cat vs Dog target domains (Transfer learning setup)

Set Up	**Test Acc (SD)**	**Test Acc (TD1)**	**Test Acc (TD2)**	**Model Size (MB)**
Base Model	66.00	92.35	59.40	2.0852
Post Training - Float 16 Quantization	64.65	90.45	58.30	1.0453
Post Training - Dynamic Range Quantization	64.55	90.55	58.05	0.5282
Post Training - Integer Quantization	63.70	89.70	57.80	0.5271
Post Training - Int Quantization + Int 16 Activation	64.80	90.45	58.15	0.5292
Quantization Aware Training	65.90	92.95	59.20	4.1784
Post Training Quantization on QAT model	65.80	92.80	59.05	0.5278

SD: Source Domain (Cat vs Dog), TD1: Target Domain 1 (Horse vs Frog), TD2: Target Domain 2 (Rotated Cat vs Dog)

From table 4.4, we observe that the quantization trends for transfer learning setup mirror that of standalone training. In fact the results in the upcoming sections also highlight and reflect the same. Hence, we can assume what holds for standalone setup to hold good for the transfer learning setup and vice versa. This observation arises because model compression techniques seem to rely on the model mostly than the underlying dataset, as we shall see in section 4.2.2. For the different quantization schemes, the model sizes are nearly the same as that obtained in a standalone setting, as should be the case, and **post training quantization + QAT** gives the best performance with desired size reduction here as well.

Choice of Layers to Quantize

Let us examine figure 4.1 that demonstrates the effect of quantizing each layer individually on the test performance.

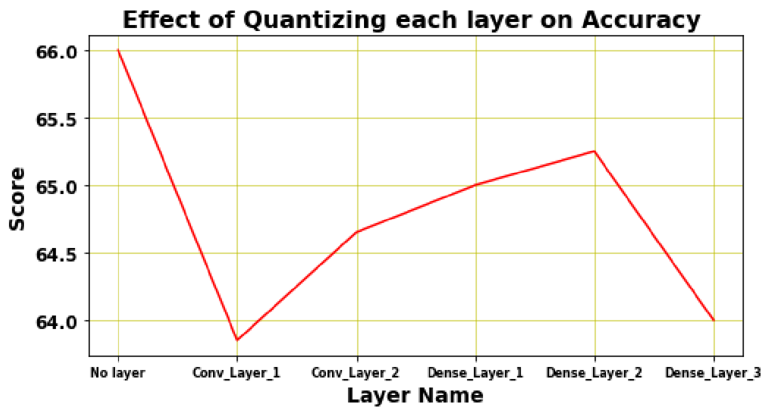


Figure 4.1: Plot showing the effect of quantizing each layer on the test accuracy

Just was the intuition behind mixed layer precision design, we expect that each layer has a different role to play in a CNN and, as a result, will have different effects on the model's performance when quantized. As observed in various literature, quantizing either the first (input) or last (output) layer leads to the high drops in accuracy and should be avoided for any mixed layer precision or hybrid layer approach design to prevent considerable information loss right at the beginning. Hence, we next move on to try quantizing different layers differently, especially in extreme low precision settings, in which case, a method of quantifying each layer's importance through the new TwoNN approach that we introduced in section 3.3.2 becomes pertinent.

4.2.2 Study on Mixed Precision Quantization

In section 2.1.3, we looked at existing mixed precision techniques that are popular. While XNOR-Net design merely quantizes all intermediate layers to binary precision, PCA-based Hybrid-Net design intelligently decides to quantize the insignificant intermediate layers alone to binary precision and the significant intermediate layers to a higher bit precision (most of the time 2-bit precision works but is dependent on the difficulty of the task and model layer's learning capacity). However, section 3.3.2 debunks the use of PCA for identifying significant layers and introduces the usage of TwoNN with the significance criteria incorporating an absolute difference rather than a normal difference. Though theoretically justified, we check the experimental validity for the same.

Before proceeding to the intended transfer learning setup, let us look at why the layer significance identification approach that originated for standalone setup becomes applicable to the transfer learning setup as well. We train the QuantNet model on the source domain dataset of Cat vs Dog and then pass a minibatch of the source or target/transfer (Horse vs Frog) dataset through the model to note the IDs of the layer outputs. PCA and TwoNN[*] are performed on vectors across channels of dimension O, and TwoNN on flattened tensors of dimension $O*W*H$. For PCA, explaining 95% of the total variance is used as the threshold.

Table 4.5: Summary of the intrinsic dimension analysis on Quant-Net model for Cat vs Dog source domain and Horse vs Frog target domain

Layer	**ID (Source domain)**			**ID (Target domain)**		
	PCA	*TwoNN*	*TwoNN**	*PCA*	*TwoNN*	*TwoNN**
Input	2	25.61	4.15	2	27.57	3.95
conv1	13	33.19	6.72	14	36.85	6.45
conv2	48	49.48	5.25	49	52.10	5.28
linear1	31	23.76	19.54	31	23.78	15.35
linear2	1	0.98	0.82	1	0.84	1.20

From table 4.5, We can observe that the trends of the ID of all methods are the same for the source and target domains, and even the value of the IDs are very close (less than the error in the measurements). This might indicate that the source and target domain input data belong to approximately the same vector space, which the model layers transform sequentially into nearly the same spaces correspondingly. Such a setup is when transfer learning is usually employed, supporting the reliability of this method when extended to the transfer learning setup.

Also, it is clear that all three methods reflect the characteristic hunchback profile of data representations' dimensionalities that Ansuini *et al.* [2019] stated. This supports the need for a better significance criterion that allows layers that also cause a reduction in ID to be significant. In figure 4.2, we notice the absence of a sharp knee point in the PCA curve of the intermediate layers, especially (layers *conv2* and *linear1*) that is characteristic of data in linear space, and this suggests curved data manifolds. Both these observations strengthen the idea of using

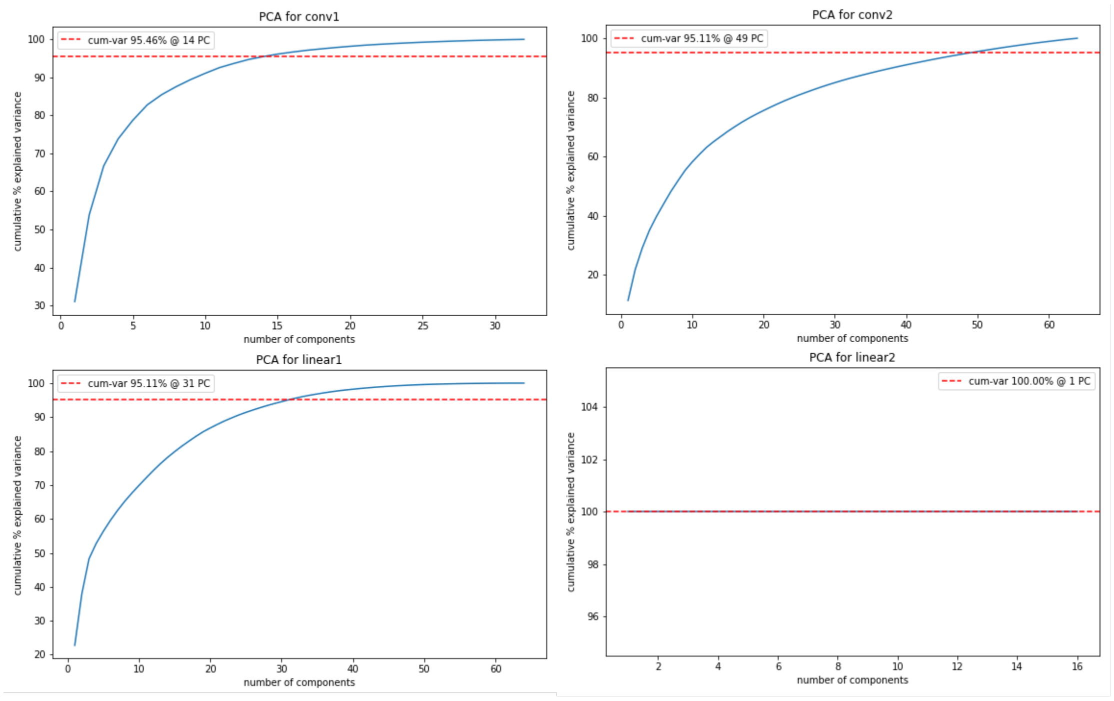


Figure 4.2: Principal Component Analysis on the intermediate layers of QuantNet for Horse vs Frog of CIFAR-10 dataset

TwoNN ID estimation and absolute difference criteria for quantifying the layer significance.

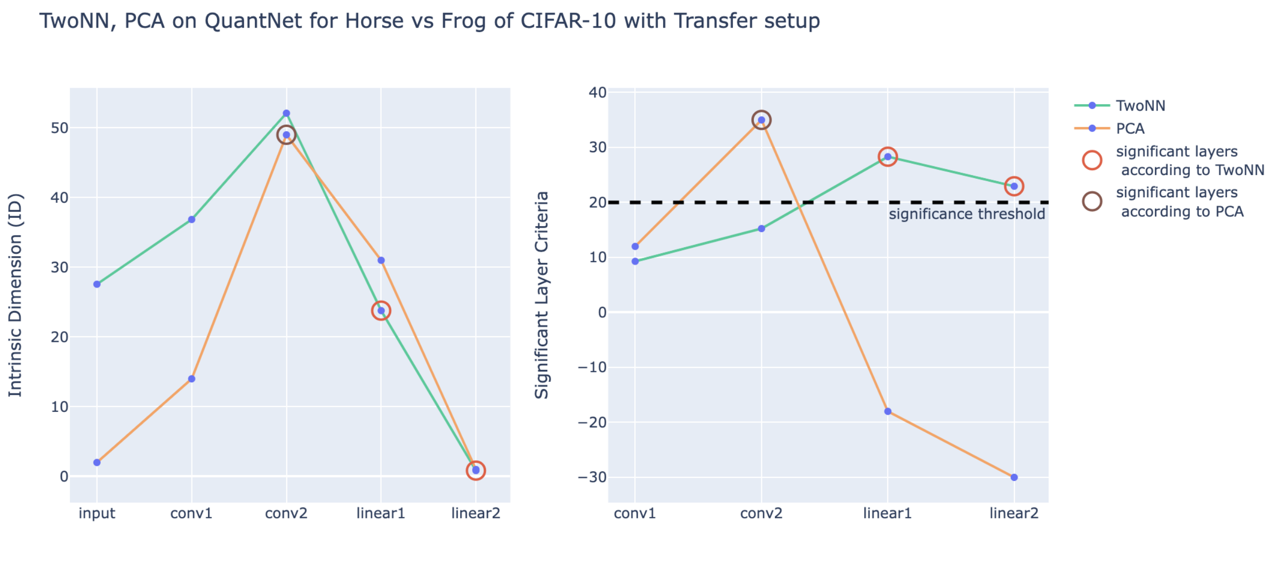


Figure 4.3: Identifying significant layers using TwoNN and PCA on QuantNet model for transfer domain in CIFAR-10 dataset, with threshold $\Delta = 20$

We now evaluate the PCA and TwoNN approaches on the target and source domains. As we saw in table 4.5, the ID values and, more importantly, the trends in the significance criteria seem to be characteristic more of the model itself for closely-related data, and hence we focus on the analysis for the transfer learning setup that we mainly seek to propose in this work and verify it with the source domain numbers. Both methods only apply to the intermediate layers, with the first and last layers always set to full precision in these very low precision settings to avoid significant accuracy drops (as observed in 4.1 and from the very low accuracy in row 3 of table 4.6). From figure 4.3's right side plot and 4.4, we observe that the *TwoNN* approach chooses *conv2* as the least significant and *linear1* as the most significant layer, while *PCA* suggests *conv2* is the most significant, *linear1* layer is less significant, and *linear2* is least significant. Table 4.6 shows that the model with *conv2* binarized achieves better performance than the one with *linear1* binarized in both source and target domains, as suggested by *TwoNN*. This result verifies the theoretical arguments for using TwoNN.

Hybrid Layer Precision Design for Very Low Memory Setting

With the layer significance identification using the *TwoNN* approach working as expected, we attempt a hybrid precision layer design on the QuantNet model, pre-trained on the Cat vs Dog dataset, for the Horse vs Frog transfer learning setup. Setting a threshold $\Delta = 20$, we categorize the layers with a significance value above it to be important and set them to a higher precision of 2 bits while binarizing the layers with a significance value below Δ (as in the left side of figure 4.4). We can see the performance for the model with this configuration in the last row of table 4.6, and as one might expect, it achieves a high empirical reduction in size

Table 4.6: Performance summary[†] of TwoNN and PCA approaches on Quant-Net model for Cat vs Dog Source Domain (SD) and Horse vs Frog Target Domain (TD)

Layer Precisions				**Test Acc (SD)**	**Test Acc (TD)**	**Memory Reduction**
conv1	***conv2***	***linear1***	***linear2***	66.00	92.30	1x
full	full	full	full			
conv1	***conv2***	***linear1***	***linear2***	64.20	90.55	30.42x
full	binary	binary	binary			
conv1	***conv2***	***linear1***	***linear2***	62.60	89.55	1.01x
binary	full	full	full			
conv1	***conv2***	***linear1***	***linear2***	65.70	91.85	1.03x
full	binary	full	full			
conv1	***conv2***	***linear1***	***linear2***	65.00	91.35	14.4x
full	full	binary	full			
conv1	***conv2***	***linear1***	***linear2***	65.80	91.6	15.87x
full	binary	2-bit	2-bit			

[†] The memory reduction mentioned in an empirical calculation using the parameter count and precisions since Tensorflow and PyTorch does not yet support memory calculations for custom implementations

with only a marginal reduction in accuracy compared to the baseline. IN terms of size reduction, which the very low precision methods attribute slightly more importance to, the design achieves reduction next to XNOR-Net design (row 2 of table 4.6 with all intermediate layers set to binary precision) but with considerably better generalizability and accuracy.

Consequently, this hybrid layer precision design for CNNs using the novel TwoNN approach for quantifying layer significance can be used to **deploy models on extremely memory-constrained edge devices** by still retaining a very high proportion of the baseline's performance.

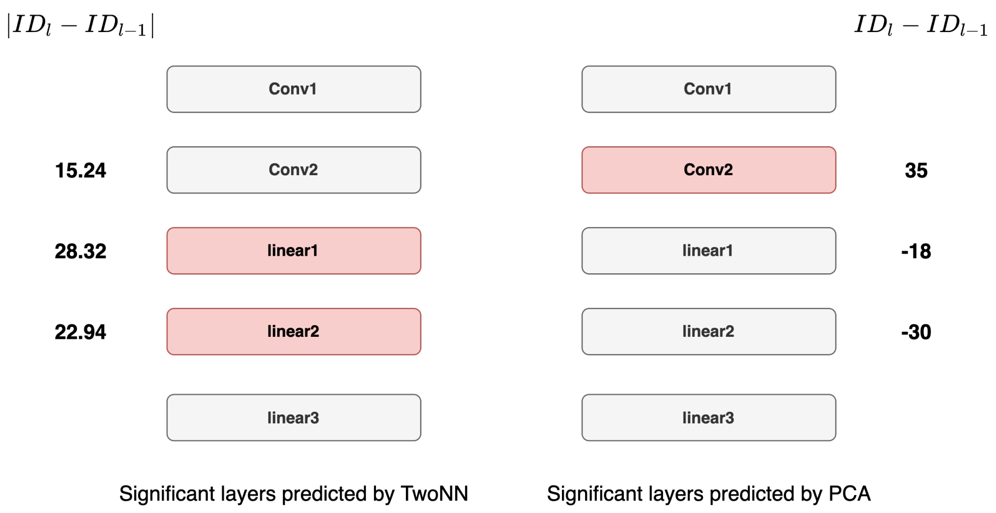


Figure 4.4: Significant layers (shaded red) identified by TwoNN (left) and PCA (right) with threshold $\Delta = 20$, along with the values of the significance criteria

These extremely low precision settings do not usually have direct memory gains from the hardware since a word is the fundamental unit of storage and operation on most processors and memory. Hence, the need for specialized hardware arises, which is outside the scope of this work, and currently, there are no in-built deployment pipelines in Tensorflow and PyTorch for the same. So what if the edge device isn't with that small a memory to attempt such extreme reductions? We seek to explore methods that share the same motivation of reducing unique parameter values and achieving better performance than the very low memory approaches, leading us to pruning and weight clustering.

4.2.3 Study on Model Pruning and Weight Clustering

Model Pruning

Pruning was introduced in section 2.3 as a method that stemmed from lottery ticket hypothesis. We employ neuron and weight pruning techniques on QuantNet model and summarize the results of the analysis in figure 4.5.

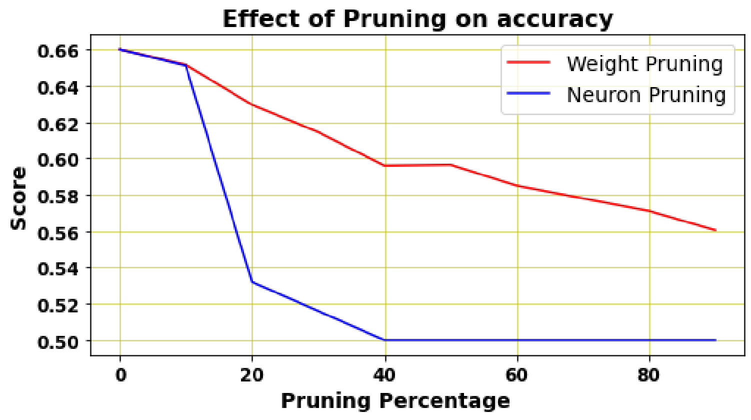


Figure 4.5: Variation of test accuracy with pruning

We observe that weight pruning surpasses neuron pruning throughout the range of percentage. This might be because our model isn't very complex or large, which necessitates removing whole units or filters, which is supported by the fact that the accuracy of neuron pruning rapidly falls to 50% (corresponding to a random choice). Whereas weight pruning only individually prunes weight values, and it works decently - we get only about a 6% reduction in accuracy with 50% of the weights pruned. But compared to the methods we saw earlier, this might not be enough for both memory reduction and performance preservation. Pruning completely discards information by zeroing them out in the path to reducing unique values and losses in performance to quantization techniques. So we turn back to methods that reduce the number of unique discrete steps instead, specifically weight clustering since it is similar in its idea to quantization.

Weight Clustering

We employ the clustering technique on QuantNet model for Cat vs Dog dataset from CIFAR-10 in a standalone setup. Figure 4.6 shows the results of the experiment performed with varying numbers of clusters. This simple plot serves as a systematic approach to optimally choosing the number of clusters, if necessary.

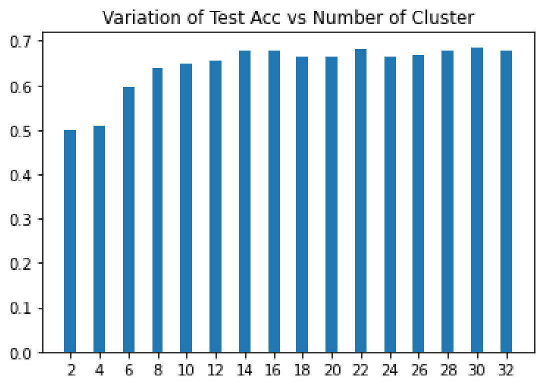


Figure 4.6: Variation of test accuracy with number of clusters (N_c)

We notice from figure 4.6 that the ideal number of clusters to use is **16**. The accuracy after $N_c = 16$ oscillates, and though there are a few higher accuracy points, like at 32, the increase is very negligible for the cost of an extra bit. One might argue $N_c = 14$ doesn't fall behind much from 16, but both require 4 bits to represent normally unless we use other compact forms of coding, which we don't get into.

In table 4.7, clustering and clustering + quantization techniques are performed on the QuantNet model. The use of the basic weight clustering technique with $N_c = 16$ results in over 8x reduction in model size. The general trends here

Table 4.7: Test performance and model size of QuantNet under different clustering schemes for Cat vs Dog

Set Up	Test Acc	Model Size (MB)
Base Model	66.00	2.0852
Clustered (N_c = 16)	65.30	0.2562
Clustered + Post Training - Float 16 Quantization	64.40	0.2153
Clustered + Post Training - Dynamic Range Quantization	63.80	0.1921
Clustered + Post Training - Integer Quantization	63.30	0.1920
Clustered + Post Training - Int Quantization + Int 16 Activation	64.90	0.1922
Clustered + QAT model	65.75	1.6755
Clustered + Post Training Quantization on QAT model	65.60	0.1922

reflect what was obtained for quantization techniques without clustering. In fact, **post-training quantization on QAT after clustering** gives the best Accuracy per MB value here as well. The accuracy has dropped by 0.2% with prior clustering compared to the earlier QAT + post-training quantization numbers. The reason could be that clustering creates a new, sparse distribution of weight values. This distribution shift might impede the QAT from converging to the previous optimal distribution of weights. Can we do better?

Cluster-Preserving QAT

Table 4.8 shows the number of clusters each layer's weight values form at the end of using a particular scheme. In the previous experiment, QAT was performed on the clustered model, treating it as a new model without taking complete advantage of the distribution shift or reduction in unique values brought about by clustering. Row 3 of the table verifies this claim showing that the number of clusters representing the number of unique weight values is very high to the extent that the effect of clustering is not felt at all at the end.

Table 4.8: Test performance, number of clusters in each layer, and model size under different clustering + QAT schemes on QuantNet model for Cat vs Dog

Set Up	**Test Acc**	**Number of clusters**	**Model Size (MB)**
Base Model	66.00	-	2.0852
Clustered (N_c = 16)	65.30	16, 16, 16, 16, 16	0.2562
Clustered (N_c = 16) + QAT	65.75	864, 18430, 361277, 1308, 16	1.6755
Clustered (N_c = 16) + Cluster-Preserving QAT	65.85	16, 16, 16, 16, 16	0.2394

Therefore, we perform clustering-aware QAT on the clustered model (row 4 of table 4.8) that preserves the number of clusters per layer. By preserving the number of unique values per layer, we have not only brought the model size to merely 0.24 MB (over 8x reduction from the base model size of 2.1MB) but also achieved the best test performance of **65.85** on Cat vs Dog dataset, only slightly higher than what we obtain using QAT + post-training quantization alone and TwoNN-based hybrid layer precision. Our intuition for this slight increase over the normal quantization aware training is that the CNN model is sort of regularized by this restriction on the number of unique values since most neural network models are over-parametrized. This over-parametrization is also verified by the pruning experiment (refer section 4.2.3), where the accuracy does not fall significantly till a pruning fraction of about 50%. Hence, we expect that this method, coupled with our novel approach for identifying significant layers, will be an ideal pipeline for general low-resource edge devices.

4.2.4 Performance of Proposed Model Pipeline

We introduced our novel model pipeline in section 3.4 based on the theoretical background of the methods that we explored in this work and the results we obtained during the comparative study. In this section, we mainly compare the results of this model pipeline to the best values we obtain from our work using the classical approaches as well as the proposed TwoNN-based hybrid layer precision approach for limited memory settings.

Table 4.9: Test performance and model size of QuantNet under the best performing compression schemes for Cat vs Dog dataset from CIFAR-10 in standalone setup

Set Up	Test Acc	Model Size (MB)
Base Model	66.00	2.0852
Post Training Quantization on QAT model	65.80	0.5278
TwoNN + Mixed Precision (1-bit, 2-bit) Quantization	65.80	0.1321
Clustering + Post Training Quantization on QAT model	65.60	0.1922
Cluster-Preserving QAT	65.85	0.2394
TwoNN + Clustering, Cluster-Preserving QAT	65.95	0.1625

From table 4.9, we clearly see that the model pipeline we proposed outperforms other techniques that we explore in this work. Both methods adopted for significant and insignificant layers have a clustering part which is common, and that helps when implementing this pipeline. It also acts as a regularization by creating a sparser weight distribution that prevents over-fitting. The TwoNN-based mixed precision model achieves the next best accuracy with even lesser model size, justifying its application in memory-constrained systems. However, whenever memory permits, it seems better to employ the clustering-based model pipeline proposed for increased performance. The difference between the two might be higher for complex models and/or difficult tasks.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this work, we propound two novel model pipelines for two memory settings of the edge device - TwoNN + Clustering, Cluster-Preserving QAT for limited memory and TwoNN + Mixed Precision (1-bit, 2-bit) Quantization for extremely low memory. We compare our proposed pipelines in model size and performance to the best results from classical and notable techniques that we discuss in this work to validate our claim. These pipeline designs work for both a standalone and a transfer learning setup. Apart from the trade-off between accuracy and storage size, the proposed designs are expected to have reduced latency and acceleration advantages with specialized hardware. The comparative study that was presented backed the choice of such a model pipeline for CNN compression along with intuitions drawn from the theoretical study. The seed of the idea was the realization that different layers of a neural network specialize in different sub-tasks within the framework of the main task and have to be treated differently while employing the compression techniques.

In the course of our comparative study, we come across the shortcomings of the existing PCA-based method for identifying significant layers for a hybrid precision model design. Consequently, we introduce the approach of using TwoNN, which proves to be a better metric for arriving at significant layers and forms the basis for the model pipelines presented. We hope that this work helps the community to analyze the layer significance of a CNN using a robust method and tap into the advantages of using hybrid layer compression techniques.

In the future, we intend to compare and extend this hybrid layer approach to more techniques that might not have gained popularity but show potential. Also, we hope to explore the different methods in depth from the perspective of optimizing constraints in edge devices other than memory like latency, computations, etc. Given the carbon footprint of training state-of-the-art models, we wish to study the methods for optimizing online training on the edge device itself and explore techniques for a federated learning setting. There is also the angle about the hardware and the relative pros and cons of the various techniques when they actually run on the hardware that we seek to analyze. We also wish to focus on completely new techniques developed for specific use-cases within the world of edge devices. Finally, we aim to formally extend this approach and study on CNNs to neural network architectures for other important tasks like NLP, Natural Language Generation (NLG), time-series analysis, motion control, etc. Given that this work deals with a very hot topic in deep learning research - model compression, there are a number of follow-ups and orthogonal directions that one can venture into, and we believe that this work will instigate many more related research.

REFERENCES

Ahn, S., **S. X. Hu**, **A. C. Damianou**, **N. D. Lawrence**, and **Z. Dai** (2019). Variational information distillation for knowledge transfer. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 9155–9163.

Ansuini, A., **A. Laio**, **J. H. Macke**, and **D. Zoccolan** (2019). Intrinsic dimension of data representations in deep neural networks. *arXiv preprint arXiv:1905.12784*.

Bergstra, J. and **Y. Bengio** (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, **13**(null), 281–305. ISSN 1532-4435.

Chakraborty, I., **D. Roy**, **I. Garg**, **A. Ankit**, and **K. Roy** (2020). Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence. *Nature Machine Intelligence*, **2**(1), 43–55. URL `https://doi.org/10.1038%2Fs42256-019-0134-0`.

Chatterjee, A. and **L. R. Varshney**, Towards optimal quantization of neural networks. *In 2017 IEEE International Symposium on Information Theory (ISIT)*. 2017.

Cheng, Y., **D. Wang**, **P. Zhou**, and **T. Zhang** (2017). A survey of model compression and acceleration for deep neural networks. *ArXiv*, **abs/1710.09282**.

Crefeda Rodrigues, **G. Riley**, and **M. Luján** (2020). Energy predictive models for convolutional neural networks on mobile platforms. URL `http://rgdoi.net/10.13140/RG.2.2.15224.80644`.

Facco, E., **M. d'Errico**, **A. Rodriguez**, and **A. Laio** (2017). Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports*, **7**.

Frankle, J. and **M. Carbin** (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. URL `https://arxiv.org/abs/1803.03635`.

Gholami, A., **S. Kim**, **Z. Dong**, **Z. Yao**, **M. W. Mahoney**, and **K. Keutzer** (2022). A survey of quantization methods for efficient neural network inference. *ArXiv*, **abs/2103.13630**.

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

He, K., **X. Zhang**, **S. Ren**, and **J. Sun** (2015). Deep residual learning for image recognition. URL `https://arxiv.org/abs/1512.03385`.

Iandola, F. N., **S. Han**, **M. W. Moskewicz**, **K. Ashraf**, **W. J. Dally**, and **K. Keutzer** (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and &lt;0.5mb model size. URL `https://arxiv.org/abs/1602.07360`.

Jacob, B., **S. Kligys**, **B. Chen**, **M. Zhu**, **M. Tang**, **A. Howard**, **H. Adam**, and **D. Kalenichenko** (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2704–2713.

Jovanovic, B. (2022). Internet of Things statistics for 2022 - Taking Things Aparte. `https://dataprot.net/statistics/iot-statistics/#:~:text=In%202021%2C%20there%20were%20more,in%20economic%20value%20by%202025.`

Krizhevsky, A., Learning multiple layers of features from tiny images. 2009*a*.

Krizhevsky, A. (2009*b*). Learning multiple layers of features from tiny images. Technical report.

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

LeCun, Y., **J. Denker**, and **S. Solla**, Optimal brain damage. *In* **D. Touretzky** (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL `https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf`.

Lee, D.-H., **S. Zhang**, **A. Fischer**, and **Y. Bengio**, Difference target propagation. *In* **A. Appice**, **P. P. Rodrigues**, **V. Santos Costa**, **C. Soares**, **J. Gama**, and **A. Jorge** (eds.), *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, Cham, 2015. ISBN 978-3-319-23528-8.

Liu, S., **D. S. Ha**, **F. Shen**, and **Y. Yi** (2021). Efficient neural networks for edge devices. *Computers & Electrical Engineering*, **92**, 107121. ISSN 0045-7906. URL `https://www.sciencedirect.com/science/article/pii/S0045790621001257`.

Mcculloch, W. and **W. Pitts** (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 127–147.

Rastegari, M., **V. Ordonez**, **J. Redmon**, and **A. Farhadi**, Xnor-net: Imagenet classification using binary convolutional neural networks. *In* **B. Leibe**, **J. Matas**, **N. Sebe**, and **M. Welling** (eds.), *Computer Vision – ECCV 2016*. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46493-0.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65 6**, 386–408.

Shafique, M., **T. Theocharides**, **C. Bouganis**, **M. Hanif**, **F. Khalid**, **R. Hafiz**, and **S. Rehman**, An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era. 2018.

Stoychev, S. and **H. Gunes** (2022). The effect of model compression on fairness in facial expression recognition. URL `https://arxiv.org/abs/2201.01709`.

Szegedy, C., **W. Liu**, **Y. Jia**, **P. Sermanet**, **S. Reed**, **D. Anguelov**, **D. Erhan**, **V. Vanhoucke**, and **A. Rabinovich** (2014). Going deeper with convolutions. URL `https://arxiv.org/abs/1409.4842`.

Yang, T.-J., **Y.-H. Chen**, and **V. Sze**, Designing energy-efficient convolutional neural networks using energy-aware pruning. *In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

Zhou, S., **Y. Wu**, **Z. Ni**, **X. Zhou**, **H. Wen**, and **Y. Zou** (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. URL `https://arxiv.org/abs/1606.06160`.