Optimal Compression of Convolutional Neural Networks for Edge Devices

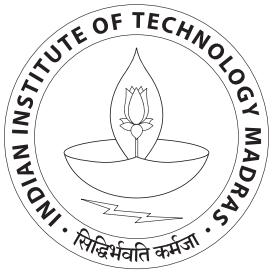
A Project Report

submitted by

ANIRUDH 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 **Anirudh S**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor 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.

Prof. Avhishek Chatterjee
Research Guide
Assistant Professor
Dept. of EE
IIT Madras, 600036

Place: Chennai

Date: 26 May, 2022

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude towards **Prof. Avhishek Chatterjee** who has played a phenomenal role in guiding me through the course of this work. I thank him for the opportunity to work on this exciting project and for the regular meetings where we he shared ideas that effectively guided me in this endeavour and transformed my outlook towards research as a whole.

I would like to acknowledge the support of Abishek S (EE18B001) who has contributed towards this project with his ideas and help in implementation. I also express my sincere gratitude towards IIT Madras for an amazing four years that I have spent here availing great opportunities for holistic growth and development.

Finally, I would like to thank my parents and sister for their unending support and encouragement as I pursued an exciting academic journey.

ABSTRACT

KEYWORDS: Convolutional Neural Networks, Edge ML, Quantization, Model Pruning, Principal Component Analysis, Clustering, Quantization-aware training, Two Nearest Neighbours

In recent times there has been a surge in the usage of Neural network based prediction models in Edge Devices like sensors, mobile phones etc. The improved prediction and generalization capabilities of the neural networks is responsible for the shift from traditional Machine learning (ML) models. A major drawback of using sophisticated neural networks in Edge devices is their limited data processing ability of these devices. In this work, we propose a efficient compression method for image classification models to decrease the data processing requirements and mitigate this liability. We show that model size and the number of computations required for training can be minimized using our compression technique which is inspired from existing works on quantization-aware training, weight sharing and model pruning. We perform a thorough comparative study with some of the major convolutional neural network (CNN) compression methods, namely **post-training quantization, quantization-aware training, model pruning, parameter clustering, and mixed precision networks**, that show potential for insights and improvement with a focus on reducing the memory footprint. We try to combine a subset of the above methods to optimally compress the model through analyzing the dimensionalities of feature channels across layers. Finally, we also propose a pipeline to efficiently compress image classification models for a transfer learning

setup that makes it memory efficient and less time intensive to use the model for predictions on small edge devices. We perform experiments on several binary image classification tasks taken from the **CIFAR-10** dataset (Cat vs Dog, Horse vs Frogs etc.). In summary, our work introduces an entirely new and promising approach to compress an image classification models using a combination of works from different domains.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
ML	Machine learning
DNN	Deep Neural Network
IoT	Internet of things
CNN	Convolutional Neural Network
QAT	Quantization Aware Training
Q-Aware	Quantization Aware
CPU	Central processing unit
GPU	Graphics processing unit
MAC	Multiply and accumulate computations
PCA	Principal Component Analysis
twoNN	Two Nearest Neighbors
ID	Intrinsic Dimension
TL	Transfer learning

NOTATION

nPC	Number of Principal Components
N	Number of input data samples for training
fp32	32-bit Floating Point
$\mathbf{n}$	Flattened dimensions of a general weight tensor
$\mathbf{N_c}$	Number of clusters

CHAPTER 1

INTRODUCTION

The idea of neural networks dates back to as early as 1943, when Mcculloch and Pitts [1943] tried to implement an artificial neuron that could replicate the activities of a human brain. That was the first artificial computational neuron that was developed and they could handle only binary inputs. Though primitive, this was a first step towards developing intelligent machines capable of replicating complex human behaviour. This was followed by Rosenblatt [1958] introducing the real inputs handling perceptron model and the perceptron learning algorithm to learn the weights for each input. A drawback of this model was it's inability to learn non-linearly separable functions. Due to lack of computational capabilities and the inherent limitations of the above mentioned methods, traditional ML algorithms were preferred over deep neural networks until late 20^{th} century.

Researchers at Gartner [Hung][1], estimated that there will be 20 billion IoT devices connected to the Internet by 2020 . The burgeoning of such devices has sparked many efforts into researching the optimal device design. In recent years, DNNs have achieved state-of-the-art performance in many domains, ranging from computer vision, natural language processing to autonomous vehicles. By exploiting these achievements, many industrial IoT applications such as robotics, smart factory, and warehouse, have emerged and attracted immense popularity.

Since most edge devices are constrained in terms of processing power and energy resources, training large amounts of data on them is not feasible most of the

[1]Link: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

time and the traditional approach has been to transmit data generated by the device to a cloud platform for server-based processing (cloud-computing). Hence, the deep learning models are trained in powerful on-premises or cloud server instances and then deployed on these edge devices. Although cloud computing has been successfully employed, it is sometimes not desirable due to concerns about latency, connectivity, energy, privacy, and security [R. K. Naha and Ranjan; A. Kumar and Varma, 2017]. To avoid the above issues of cloud computing and resolve the issue with limited computation resource and storage space of an edge device, many research works have proposed to design efficient DNNs. In the design of efficient DNNs, considerable efforts are spent on three major areas, which are **quantization** [[Mohammad Rastegari and Farhadi, 2016], [Benoit Jacob and Kalenichenko, 2018], [Kuan Wang and Han, 2019]], **pruning** [[Song Han and Dally, 2015], [Yihui He and Sun, 2017], [Hengyuan Hu and Tang, 2016]], and **network architecture design** [[Xiangyu Zhang and Sun], [Andrew G Howard and Adam, 2017]].

A key observation here is that while prior approaches select a fixed subset of weights or neurons from the model to quantize or prune, it is possible that different subsets play vital roles in processing different inputs and different methods utilizes them in different ways to get the best output. This motivates the idea of combining a few of the existing works (like Quantization-aware Training (QAT), clustering, pruning etc.) to come up with an efficient algorithm for model compression. The objective of our work is to perform efficient model compression on Image classification models used in transfer learning setup.

As the IoT and edge devices proliferate deeper into real-world applications, the goal of improving the prediction capabilities of them with the help of efficient DNNs gains prominence. While recent works in this area prescribe to modifying the

network architecture to obtain reduced network size and improved the inference speed using ideas like depth-wise separable convolution [Andrew G Howard and Adam, 2017], Squeeze networks [Jie Hu, 2017] and platform-aware neural architecture search, we opt for combining the traditional methods like quantization and pruning to obtain and simple yet efficient compression algorithm.

In our work, we conduct a thorough comparative study among different Quantization, Pruning and Clustering techniques generally used for model compression. Then we propose a novel method to compress DNNs used for image classification using the idea from Mixed precision network topology that incorporates different compression techniques on the significant and insignificant portions of the network.

The main contributions of this work are summarized as follows:

- We investigate several quantization approaches such as Integer quantization, Floating point quantization, Dynamic Range quantization and Quantization aware Training for edge devices and identify the advantages and disadvantages of these approaches.
- We compare the performances of pruning and weight clustering methods and list advantages and disadvantages of these approaches. We also infer the advantages of using a combination of Quantization and clustering together.
- We demonstrate the benefits of mixed precision network topology and evaluate the various existing approaches for transforming a model into a mixed precision network.
- We develop an efficient technique that detects a subset of significant layers, then applies QAT to the significant layer weights and weight clustering to the insignificant layer weights to get the best model compression and performance.

CHAPTER 2

RELATED WORK

In this chapter, we first describe the need for on-device computation and storage on edge devices. Then we move on to briefly review related literature in this area by discussing approaches that have earlier been proposed for compressing DNNs. Further, we provide a primer on the Mixed precision network topology methods and Intrinsic dimension identifying methods that we have used in this work.

2.1 Need for On-Device computation and storage

As previously stated, cloud computing is significantly used for processing and storage in growing artificial intelligence industrial based IoT applications. However, moving computation and storage to faraway cloud servers has a number of drawbacks. Firstly, the **network connections between edge devices and remote cloud servers might be unreliable** especially when edge devices are connected wirelessly. This shortcoming could fail some applications requiring fast and reliable network connections such as industrial robotics. Secondly, the **latency of computation is high** since a significant amount of data needs to be transferred to cloud servers, which could exhaust the network bandwidth and increases substantial network workload. Meanwhile, long latency also degrades the application's user experience dramatically. Thirdly, **transmitting data to remote cloud servers might pose privacy issues** since the data might contain private information. The shortage of privacy protecting protocols significantly degrades the safety to result

in avoidance of the application. Hence, because of all these issues with cloud computing, a fast and efficient DNN model running on edge devices is necessary.

Few major techniques used for achieving on-device computation and storage, that we will discuss in this work are quantization, which uses a few bits to represent each weight and activation in a DNN model. The second technique is pruning by eliminating redundant connections and neurons in a DNN model. The third technique is Clustering, or weight sharing, which reduces the number of unique weight values in a model, leading to benefits for deployment. In the following sections of this chapter, we will briefly cover each of these strategies.

2.2 Quantization techniques

Quantization, is the most commonly used technique to reduce model size and computational complexity. This technique involves **replacing floating points with integers** (or numbers of desired precision) inside the network. However, there is a trade-off: with quantization, we can lose significant accuracy.

The argument for quantization is that while adding and multiplying two numbers in scientific format, we can see that float arithmetic is more complicated than integer arithmetic. In actuality, the speed of each calculation is heavily influenced by the hardware used. A modern CPU in a desktop system, for example, can perform float arithmetic as quickly as integer arithmetic. GPUs, on the other hand, are better at single-precision float calculations (since this is the most prevalent type for computer graphics). Hence, without being completely precise, it can be said that **using int8 is typically faster and lighter than float32** for computations. The different types of precision conversions used in Quantization include **(i) float32**

to int8, (ii) float32 to float16, (iii) Quantize matrix multiplications to int8, while **activations to float16**.

Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network of low-bit width numbers. This dramatically reduces both the memory requirement and computational cost of using neural networks. Different types of Quantizations used in practice include:

- **Post-training Quantization**: Train the model using float32 weights and inputs, then quantize (modify precision) the weights (to precision of our choice as mentioned above). Its main advantage is that it is simple to apply. The downside is, it can result in accuracy loss.
- **Quantization-aware training (QAT)**: Here the forward pass is quantized and backpropagation updates the parameters at full precision through the lens of the quantized values in the forward pass. In general, studies have shown that when applying int8 quantization to the network, QAT has the best performance, but it is more involved than Post-training Quantization option.
- **BinaryConnect**: We Quantize the network parameters using a binarizing operation that can be either deterministic or stochastic, which maps float-valued parameters to +1 or -1. The deterministic operation is attractive because of how straightforward it is - simply take the sign of the float-valued parameter. For binary networks, the sign operation minimizes the Frobenius norm of the difference between binary-valued and float-valued parameter matrices thus making sign operator a natural choice for the binarizing operation.

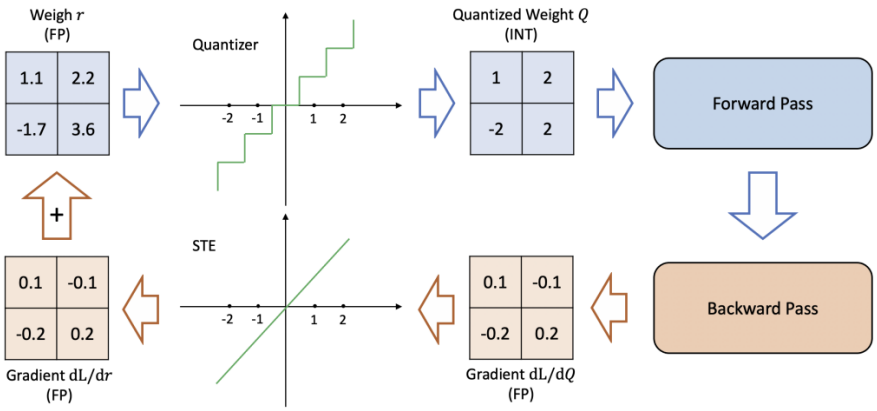


Figure 2.1: Illustration of Quantization Aware Training
Source: Gholami *et al.* [2022]

In further sections, we compare these three quantization techniques on the CIFAR-10 dataset for binary image classification. Other commonly used Quantization techniques for deep learning models include:

- **Continuous-discrete learning**: The forward pass of the neural network uses a scheme for rounding float-precision parameters to discrete levels, and in the backward pass, the float-precision parameters are updated using gradients calculated during the forward pass.
- **Ternary Parameter Networks**: Ternary parameter networks are an alternative to binary parameters that allow for higher accuracy, at cost of a larger model size. In particular, the method of [Fengfu Li, 2016] defines ternarization operation with scaling and threshold that is an effective approximation under the assumption of a parameter distribution which is between Gaussian and uniform.
- **Quantized activations**: We don't need to stop with just the parameters of the network being quantized, activations too are quantized to cut memory. Quantizing the inputs to each convolutional layer of the neural network results in a massive reduction of necessary computation. Furthermore, the convolution can be carried out in a bitwise manner, which is much more efficient, provided the relevant hardware acceleration is used.

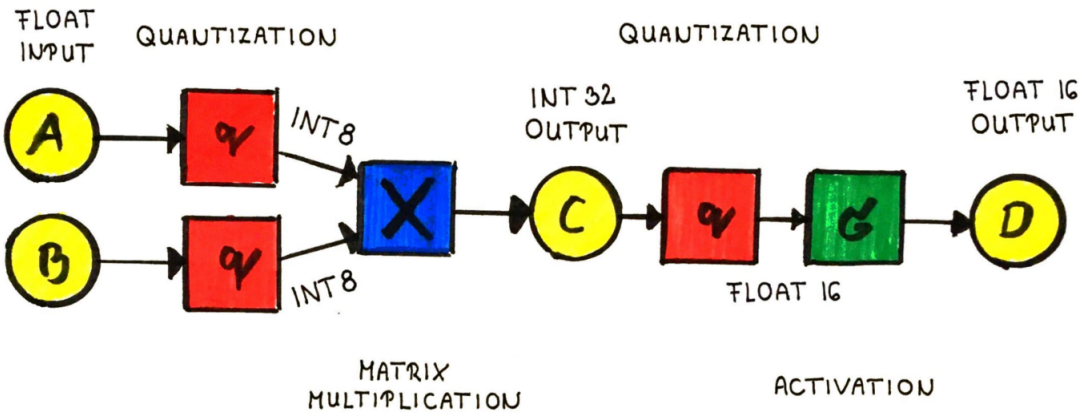


Figure 2.2: Illustration of Quantizing matrix multiplications to int8 and activations to float16

2.3 Model Pruning

The term Pruning is inspired from agriculture, where pruning refers to the act of cutting off unnecessary branches or stems of a plant. Similar to that, Model Pruning is the process of removing unwanted weight connections in a network to increase inference speed and decrease model storage size. Most commonly used pruning method is magnitude-based pruning. In Magnitude-based weight pruning we gradually zero out model weights during the training process to achieve model sparsity.

In Michael Zhu [2017]'s work, there is a comparison between two distinct methods of (i) training a large model, and performing pruning to obtain a sparse model with a small number of nonzero parameters (large-sparse) (ii) training a small-dense model with a size comparable to the large-sparse model. This comparison is to determine the trade-off between model accuracy and the model size of **large-sparse vs small-dense** models. It was observed that using a large-sparse model was more beneficial than using a small-dense model with same size. Sparse models are easier to compress, and we can skip the zeroes during inference for latency improvements.

Pruning in general can be categorized as:

- **Weight Pruning:** This is an unstructured pruning approach where individual parameters are randomly set to zero. This results in a sparse neural network, with lower parameter count. This method saves memory but may not necessarily improve computing performance because we end up conducting the same number of matrix multiplications as before. Because we set specific weights in the weight matrix to zero, this is method is known as Weight Pruning.
- **Neuron Pruning:** This is a structured pruning approach where algorithms consider parameters in groups, deleting entire neurons, filters, or channels. We set entire columns in the weight matrix to zero, thus removing the matching

output neuron. This is also known as Unit/Neuron Pruning. This is done to make use of technology and software that is specialized for dense processing.

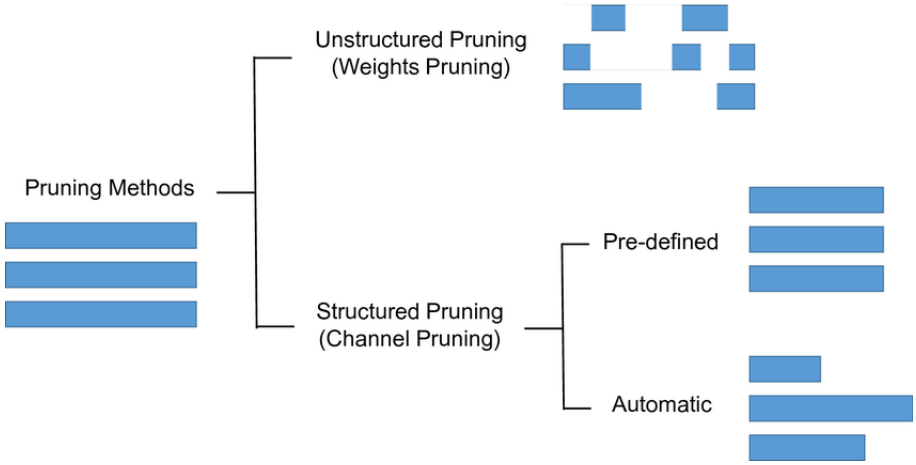


Figure 2.3: Structured and Unstructured Pruning

2.4 Weight Clustering

The weight clustering implementation of Neural networks is based on Song Han and Dally [2015] work. Clustering, or weight sharing, reduces the number of unique weight values in a model, leading to benefits for deployment. Weight clustering reduces the size of the model by replacing similar weights in a layer with the same value. These values are found by running a clustering algorithm over the model's trained weights. The number of clusters to be used is obtained through experiments.

Weight clustering has an immediate advantage in reducing model storage and transfer size across serialization formats, as a model with shared parameters has a much higher compression rate than one without. This is similar to a sparse (pruned) model, except that the compression benefit is achieved through reducing the number of unique weights, while pruning achieves it through setting weights

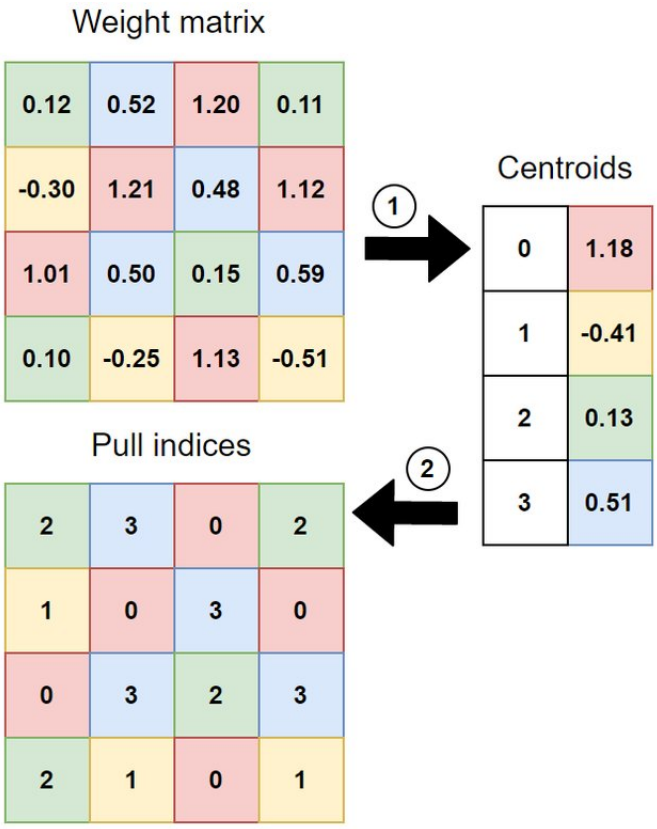


Figure 2.4: Illustration of Weight Clustering

Source: Stoychev and Gunes [2022]

below a certain threshold to zero.

2.5 Mixed Precision Quantization networks

We observed that in both, BinaryConnect algorithm from section 2.2 and the XNOR-Net paper [Mohammad Rastegari and Farhadi, 2016] the network parameters were converted to +1 or -1 to store them in 1 bit which gave 32x memory saving. Inspired from the success of the binary quantization, there were several multi-bit quantization approaches proposed. These techniques had much smaller accuracy gap between the quantized network and the floating-point counterpart due to extra information they hold (due to higher precision). The fact that each layer can be

quantized independently has paved the way for mixed precision networks.

The multi-bit quantization uses more bits to represent weights and activation compared to the binarized quantization, but the accuracy in muti-bit setup is also much higher than the binarized quantization setup, especially on complicated datasets such as the ImageNet dataset. Besides the image classification task, the multi-bit quantization approach has been extended to other tasks such as object detection, face detection, and face attributes. Two major works in this regard are the XNOR-Net paper [Mohammad Rastegari and Farhadi, 2016] and PCA Hybrid paper [Chakraborty *et al.*, 2020].

XNOR-Net Design

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

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

$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}}\, J(\alpha, \mathbf{B})$$

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

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

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

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 $\mathbf{B}^* = sign(\mathbf{W})$ (Similar to the explanation given in BinaryConnect part of section 2.2).

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

$$\alpha^* = \frac{\mathbf{W}^T\mathbf{B}^*}{n}$$

$$= \frac{\mathbf{W}^T sign(\mathbf{W})}{n}$$

$$\alpha^* = \frac{\|\mathbf{W}\|_{l1}}{n} \qquad (l1 \text{ denotes L1-norm})$$

All the intermediate layers except the first and final layer [Lee *et al.*, 2015] are set to binary precision through the quantization aware training (QAT) 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 the bias and scaling factors for intermediate layers, and parameters of first and last layers, all parameters are stored in binary precision leading to tremendous reduction in size.

PCA-based Hybrid-Net Design

There was no systematic way to decide which intermediate layers had to be set to very low precision 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 ($k_b \geq 2$) and the remaining intermediate layers are set to binary precision.

In general, 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 preceding it (i.e.) the change in the dimensionality of the data representation caused by the layer is greater than a certain fixed threshold.

In Chakraborty *et al.* [2020] work, Principal Component Analysis (PCA) is used for obtaining the dimensionality of the layer's outputs, defined as the number of principal components (nPC) required to explain 99% of the total variance. Hence from these works we can conclude that, Quantizing the model's intermediate layers to low precision in accordance with their significance decreases the accuracy drop compared to setting all intermediate layers to binary precision, while also minimizing the energy and memory footprint considerably.

2.6 Intrinsic Dimension Estimation

Deep neural networks for supervised image classification sequentially transform the input images through linear and non-linear layers and reduce the representation to a linearly separable form to make it suitable 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 Deep Neural Network. 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 conventional form of linear

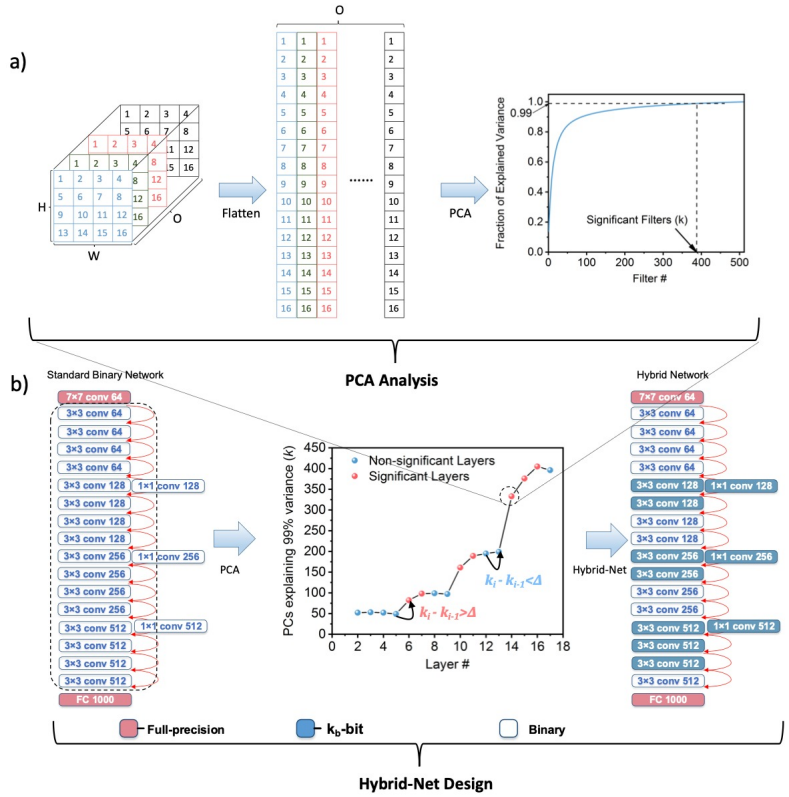


Figure 2.5: Illustration of the working of PCA-based Hybrid-Net design which selectively quantizes each layer independently based on their significance denoted by change in principal components from previous layers. Source: Chakraborty *et al.* [2020]

ID estimation and it measures the number of Principal Components required to explain satisfactory level of total variance.

2.6.1 PCA Method for Intrinsic Dimension Estimation

Principal component analysis (PCA) is a technique that generally transforms high-dimensional data into lower-dimensions while retaining as much information as possible. In other words, PCA can be defined as an orthogonal linear transformation that modifies the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate,

and so on. It's based on the idea to minimize the error when we project the data onto a new vector subspace.

Solving this objective to minimize the Projection error provides us with a solution to pick the eigenvectors of the data's covariance matrix in decreasing order of their corresponding eigenvalues for forming the best reduced subspace to project the data. The choice of number of eigenvectors (called principal components) for the subspace is arrived at by plotting the curve of cumulative explained variance vs number of principal components. The curve is expected to rise steeply initially and then gradually saturate to 100%. A plot which displays the individual contribution of explained variance by each Principal component (PC) is represented through a Scree plot. The dimensionality is then estimated as the number of principal components required to explain a certain threshold percentage of total variance (usually > 95%), denoted as nPC.

Given the idea of using difference in dimensionalities across a layers as a measure of significance of the layer's parameters and subsequently as a proxy to identify layers to quantize by Chakraborty *et al.* [2020], we explore other dimensionality estimation techniques that overcome the shortcomings of PCA (not restricting to linear subspaces) and perform better at this use case. This work explores a better alternative method to estimate the ID of data representations: **Two Nearest Neighbours (*TwoNN*).**

2.6.2 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 merely from 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 $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_N]$.

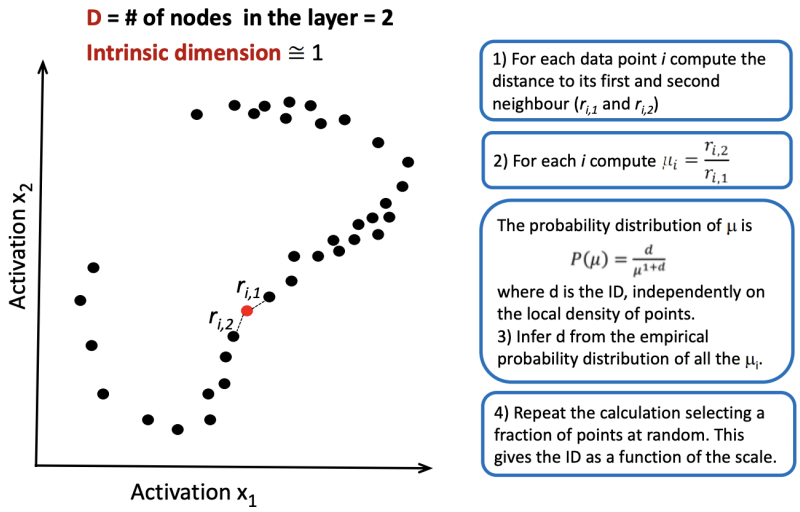


Figure 2.6: Illustration of ID estimation through TwoNN

Source: Ansuini *et al.* [2019]

The minimal neighbourhood size that this dimensionality estimator relies on, lowers the effect on data inhomogeneities on 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 datapoints. Ansuini *et al.* [2019] adopts the TwoNN approach for estimating and analyzing ID as a proxy to study generalizability of neural networks. Their experiments also suggest that 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 datapoints are used for the estimation process.

CHAPTER 3

METHODOLOGY

3.1 Problem Statement

Various machine learning (ML) algorithms, offering different accuracy and complexity, have been proposed to tackle the challenges of image classification. Despite their exceptional performance, they require high processing power and large storage. For example, some state-of-the-art neural network architectures, such as AlexNet [Krizhevsky *et al.*, 2012], GoogLeNet [Szegedy *et al.*, 2014], and ResNet [He *et al.*, 2015] require over a million parameters to represent them and more than a billion multiply and accumulate computations (MAC) [Shafique *et al.*, 2018]. Each MAC operation is generally associated with a number of memory accesses. In the worst case scenario, where there is no data re-use, each operation requires 3 reads and 1 write to memory. The simplest neural network from the aforementioned models requires around 2172M memory reads and 724M memory writes. Since these operations consume a considerable amount of processing power, the energy consumption of these algorithms might not meet the requirements of various application scenarios. If the energy consumed to classify a single image on the device was considerably less than the energy consumed to transmit the image to the cloud and receive the result, then, as one scales, it becomes advantageous to compute locally. This is exactly what we want achieve.

The contributions of this work are two-fold. First, we perform a thorough comparative study of the existing common and classical approaches used for DNN

compression like Quantization, Pruning, Weight clustering and also understand the advantages of each technique. This will equip the research community with the tools necessary to make an informed decision about the techniques to use on their edge systems in a way that balances cost and complexity with performance. Second, we present a novel model compression method deriving ideas from these existing classical approaches for a transfer learning setup.

3.2 QuantNET - Model for image classification

There are many research works which has shown that the classical model compression approaches follow similar trends across models with varied model sizes. An important study in that front was published in 2021, based on a comparative study on **Efficient neural networks on Edge devices** [Liu *et al.*, 2021]. In this work, we could observe that models like ResNet with depth of 50, 100, and 150 and XNOR-Net followed similar trends under different compression techniques. Hence, in our work we focus developing an optimal compression algorithm on a small network (size similar to that of XNOR-Net) used for Image classification called as **QuantNET**.

All the experiments are performed on the CIFAR-10 dataset [Krizhevsky *et al.*]. More details about the input dataset is provided in section 4.1.1.The image classification model designed in this work (QuantNET) consists of two convolutional layers and two dense layers followed by an output layer. The input array of size 32*32*3 is fed into a 2D-convolutional layer with 32 filters and a kernel size of 3 using 'He normal' kernel weight-initializer. A dropout of 0.1 is used after the convolutional layer to avoid overfitting as the dropout layers provide some form of

regularization. The output from this layer is then passed into another CNN layer with 64 filters and a kernel size of 3 and the same weight intializer, this layer is also followed by a dropout layer. CNN layers act like a feature extractors for the upcoming layers by learning the internal representation of the input sequence. The spatial features in the input extracted by the convolutional layers are flattened and then passed into the fully connected Dense layers to produce a higher-order feature representation for better separability among the different classes in the image data. The fully connected layers are modeled as (i) first dense layer with 128 units and (ii) second dense layer with 16 units, both with 'relu' activation, and 'He normal' weight initialization technique. The output from the two dense layers is then passed into the output layer with units equal to the number of target classes and a 'softmax' activation (in case of multi-class classification) or 'sigmoid' activation (in case of binary classification) is used to calculate the probability of each class. The model is trained using an Adam optimizer with a learning rate of 0.001 and cross entropy loss function. The predicted probabilities are then passed into an 'argmax' function to find the class with maximum predicted probability for the considered input. The hyperparameters of the Quant-NET architecture are fine-tuned using random search and Keras-Tuner and are discussed in section 4.2.1.

Figure 3.1 presents a detailed view on the architecture of the proposed Quant-NET model. This work employs a simple CNN and Dense layers based image classification model. This model is capable of exploiting different levels of information from the different layers. The layers closer to the input data captures the specific characteristics (image features like ears, nose, face etc.) of the dataset, whereas the deeper layers of this architecture capture significant information that are more relevant to the specified learning task, i.e., image classification. In case of transfer learning setups, the target classifier uses the well-learned knowledge of

parameter weights from the source domain to learn better representation to make accurate predictions the target dataset. The importance of the information carried by each layer of the Quant-NET is studied using a simple Quantization experiment which is discussed in section 4.2.2

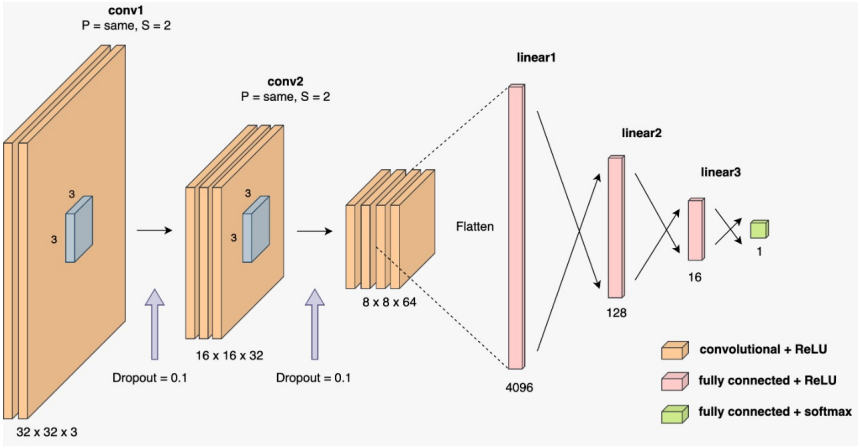


Figure 3.1: QuantNET: Model architecture

3.3 Extracting Significant layers of the network

In section 2.5, the Chakraborty *et al.* [2020] paper had introduced the idea of using PCA for significant layer extraction. In this work, we implement a better method for layer extraction based on Two Nearest Neighbours (TwoNN) which we briefly explained in section 2.6.2 and observed TwoNN to outperform the existing PCA based method.

In our implementation of the Hybrid-Net design, the model layers are quantized individually with different precisions based on their significance - first and last layers are set to full precision [Lee *et al.*, 2015], 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 Q-aware training and 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 good 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 is like an additional form 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.

3.3.1 TwoNN Method for Identifying Significant Layers

We have already briefly seen about TwoNN method in section 2.6.2 and we have also mentioned about the fact that this outperforms the existing PCA based method. Looking at why, we can find two main disadvantages with PCA in the context of analyzing data representation in neural networks. First, the output from few layers might have a sparse set of outliers that does not affect the final results but are caused due to noise in the input and due to the presence of irrelevant neurons in over-parametrized models. Second, the data passes through a series of linear weights and non-linear activations that might cause the data to form curved manifolds (since the effect of non-linearity sets in after the activation following 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), especially in the middle layers and TwoNN is better equipped to handle this.

Hence, we propose the use of TwoNN intrinsic dimension estimation method 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 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.

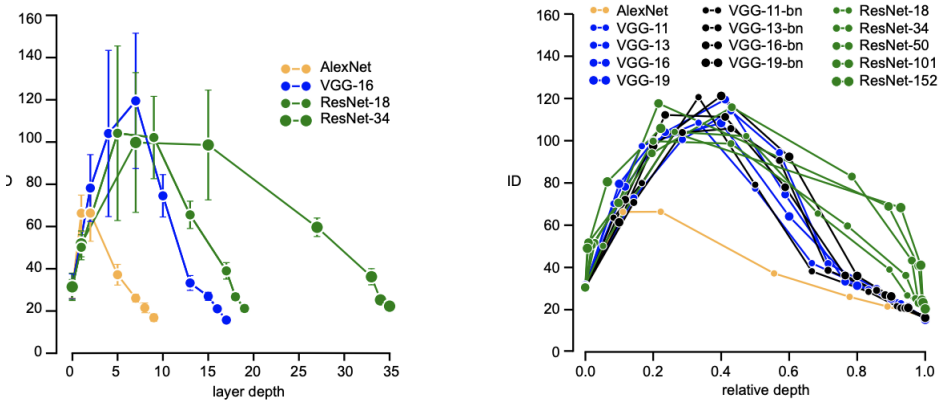


Figure 3.2: The ID of data representations following a hunchback profile

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.

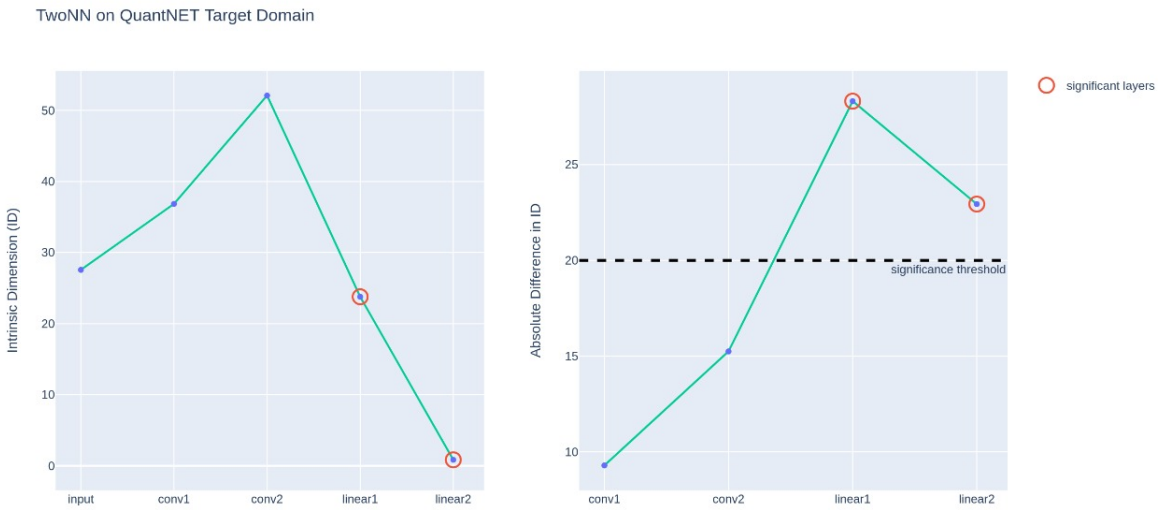


Figure 3.3: (a)Illustration of Intrinsic dimensions obtained for Quant-NET on CIFAR-10 binary classification dataset(Cats vs Dogs) (b) Thresholding of difference in Layer Dimensions to obtain significant layers

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 featuremap and all of them together characterize the filter, and subsequently the data representation corresponding to a given image at the end of this layer. The Dense layers gives a flattened output, hence there are no ambiguities there. Application of the ID estimation method on our Quant-NET 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.

3.4 Model Flow

We will discuss the entire flow briefly in this section. In our work, we make use of this TwoNN dimension estimation method to identify the significant layers from the input network (QuantNET) for a given input dataset (CIFAR-10). In case of transfer-learning setup, the trained QuantNET model (from the source domain) is transferred to target domain, where we identify the significant layers with the help of TwoNN method and target domain data. The weights of the insignificant layers are clustered using the **weight clustering** technique. The significant layers from the original model is then trained using **Cluster preserving Quantization-aware training** to obtain optimal model compression with minimal drop in accuracy. The entire algorithm is shown below as a flow-diagram in Fig:3.4.

The use of Q-aware training ensures minimal accuracy loss but increases model size as well, hence we also provide another technique for memory sensitive settings, in which we use the TwoNN method to identify the significant layers and a custom

mixed precision network instead of Clustering and Q-Aware training to get both accurate and memory precise network.

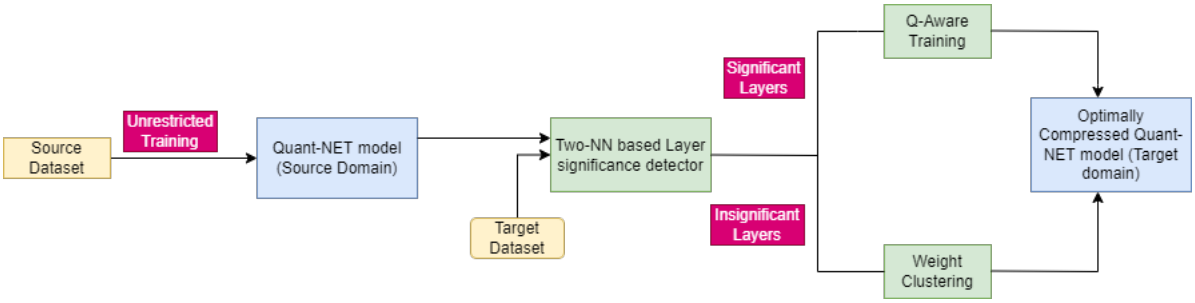


Figure 3.4: Flow diagram of the proposed optimal compression technique

CHAPTER 4

EXPERIMENTS

4.1 Experimental Setup

4.1.1 Data

All our experiments are performed on the CIFAR-10 dataset [Krizhevsky *et al.*]. We restrict our models to binary image classification tasks as the performance of our individual model compression techniques are independent of number of classes in the input data.

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

The 10 different classes that exists in the dataset are: **airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck**. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. A point to note is that 'Automobile' includes sedans, SUVs while 'Truck' includes only big trucks and neither includes pickup trucks.

Figure 4.1: (a) Sample Cat image from CIFAR-10 and (b) Sample Dog image from CIFAR-10

To test our model's performance we make use of following classification tasks:

- **Cats vs Dogs:** Image classification model trained to distinguish between cats and Dogs images from the CIFAR datset.
- **Horses vs Frogs:** Image classification model trained to distinguish between horses and frogs images from the CIFAR datset.
- **Rotated Cats vs Dogs:** Image classification model trained to distinguish between cats and dogs when the images have bee modified random amount of rotation, flip, zoom from the original images from the CIFAR datset.

Listing 4.1: Augmentation Snippet

```
data_augmentation = keras.Sequential([
                    preprocessing.RandomFlip('horizontal'),
                    preprocessing.RandomRotation(np.pi,
                                        fill_mode = 'nearest',
                                    interpolation ='bilinear'),
                    preprocessing.RandomZoom(0.1)])
```

The code used for creating the modified versions of the Cat and Dog images from the original cifar images is mentioned above. The dataset for each experiment is prepared by combining equal number samples from each class (balanced dataset) for which the complete network generates the predictions.

4.1.2 Implementation details

The first two convolutional layers in Quant-NET have a kernel size of 3 and generate 32 and 64 output feature maps respectively. The two fully connected layers have output dimensions of 128 and 16. We use the binary cross-entropy loss function and the Adam optimizer for training with a learning rate of 0.001. The model is trained for 100 epochs with early stopping. All experiments were performed with a train-validation-test split of 70-10-20 on an NVIDIA Tesla K80 GPU.

4.2 Results and Discussion

The performance of the QuantNET model on the tasks of binary image classification on the datasets mentioned in section 4.1.1 have been discussed next. The performances were assessed based on Top-1 Accuracy, metrics like precision and recall were not considered as the input data was balanced (equal number of samples from each class). The error values given in this section are the average error values obtained after 20 iterative and independent runs.

4.2.1 Hyperparameter Tuning - QuantNET

Before we get into the results of performance of Quant-NET under different model compression techniques, we make sure that this model is tuned to give us the best possible performance when no compression is applied.

The hyperparameters of this QuantNET model was fine-tuned using random search algorithm (Refer Table 4.1). As the name suggests Random Search algorithm does a random search in the given parameter space to pick a point randomly from

the configuration space. The intuition of how it works better is that we can explore the hyper-parameters space more widely with Random Search (especially for the more important variables). This will help us to find the best configuration in fewer iterations. This work, Bergstra and Bengio [2012] shows empirically and theoretically that randomly chosen trials are more efficient for hyper-parameter optimization than trials on a grid and manual selection.

Table 4.1: 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

The weight intializer parameter of all the layers except the output layer was also tuned, but using manual search among **'He Normal', 'Glorot Normal', 'Glorot Uniform', 'He Uniform'** options and **'He Normal'** was found to be the optimal one. In case of output layer, because the layer had sigmoid activation, prior works suggest **'Glorot Normal'** intializers works best on layers with Sigmoid activation, hence that was used.

It is important to understand the significance of Hyperparameter tuning of QuanNET. We employ model compression techniques like model pruning on QuantNET to compare the Top-1 accuracy we get on the input data, however, if the model is not tuned to its best performing state, what these compression techniques identify may simply be the tuned and best performing version of untuned QuantNET, which is not what we want to find.

4.2.2 Comparative study of Quantization techniques

The computational and space complexity of a high-performance neural network is high, and thus it is challenging to deploy a high-performance neural network on edge devices. A common approach to address the high computational and space complexity is to compress neural network models using quantization techniques which use a few bits to represent each weight and activation in a neural network.

By replacing floating-point numbers with fixed-point numbers, arithmetic operations consume much less resource and power compared to floating-point arithmetic operations. Also, hardware complexity is reduced significantly. Moreover, the number of off-chip memory access is decreased, which leads to immense cost reduction in communication. The Quantization techniques that we will be using for this comparative study are already discussed in Section 2.2.

Mathematical Formulation

Suppose there is a neural network with L layers and layer-wise parameters θ_l for $l \in \{0, \ldots, L-1\}$. 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 values are by itself only of fp32 precision, we treat them as $\in \mathbb{R}$ for simplicity). Model quantization attempts the following:

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

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 2.5).

Quantization techniques used in Practice: (i) **Post-training:** Train the model using float32 weights and inputs, then quantize the weights. Its main advantage is that it is simple to apply. The downside is, it can result in accuracy loss. (ii) **Quantization-aware training:** Quantize the weights during training. Here, even the gradients are calculated for the quantized weights. This is more involved than the other option.

The results in Table 4.2 are obtained by stand-alone training of the QuantNET model on input CIFAR data (cats vs dogs or Horse vs frogs) using cross entropy loss and testing the model performance after applying the Quantization technique of our choice.

Table 4.2: Test accuracy, Size (MB) of different Quantization methods on Cat vs Dogs and Horse vs Frogs datasets (Stand-alone)

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

Few points we can infer from Table 4.2 is that Quantization-aware training outperforms Post-training quantization methods but we also that the Size of the quantization-aware training model is the largest among all the models in comparison. As expected, the simple integer post-training qunatization resulted in maximum drop in accuracy as it has the maximum amount of information loss, but again it's the smallest when compared size wise. Hence, the model with a combination of **post-training integer quantization and Q-aware training** is the most optimized with respect to Accuracy and Size of the model.

Trends in Transfer learning setup

The trends observed in table 4.2 are for stand-alone Quant-NET models. We now perform an experiment to illustrate that the same trends hold true for a transfer-learning set up 4.3. We train the model and Cat vs Dogs and test the model on Horse vs Dogs or Rotated Cats vs Dogs dataset (after training and quantizing).

Observations from Table 4.3 show that the general trends observed with different Quantization techniques for Source domain only set-up seems to be valid for the TL set up too and hence experiments and trends observed in the coming sections for stand-alone model equally holds good for TL setup. Note that the model trained on cat vs Dog and then fine-tuned for Horse vs Frog performed almost at par with a stand-alone Horse vs Frog classifier (Avg Acc drop = 0.8%). We can see that the size of the models for the target-domain are same as that of stand-alone source domain model, hence again **Post Training Quantization on Q-Aware model** provide the best accuracy for a small model.

Table 4.3: Test accuracy and Size of model (MB) under different Quantization schemes with Source: Cat vs Dogs and Target: Horse vs Frogs data or Rotated Cats vs Dogs (Transfer learning set up)

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

Source Domain (SD): Cat vs Dogs, Target Domain 1 (TD1) : Horse vs Frogs,
Target Domain 2 (TD2) : Rotated Cats vs Dogs

Which layers to Quantize ?

It is fairly intuitive that different layers of the Quant-NET model play different roles in performing our image classification tasks.We expect the CNN layers to act like a feature extractors for the upcoming layers and the Dense layers to produce a higher-order feature representation for better separability among the different classes in the image data. So we postulate that quantizing different layers will have different hit on model performance. This is the idea behind model pruning as well.

From figure 4.2 we can observe that Quantizing the first and last layers have a dire effect on model performance and this result has been reported in many prior

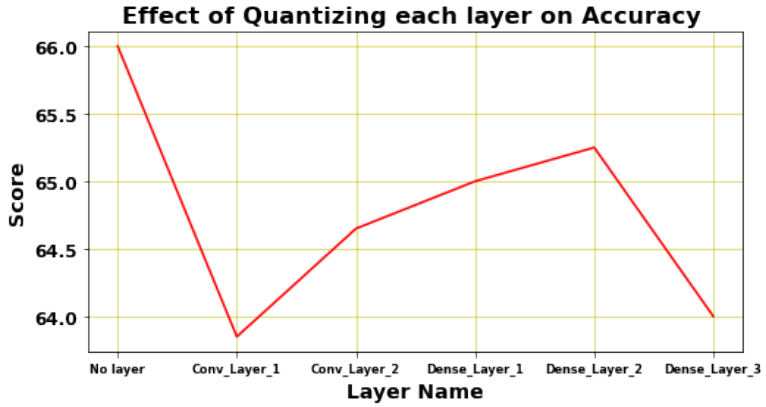


Figure 4.2: Effect of Quantizing each layer on Accuracy

works [Lee *et al.*, 2015]. Also, as expected Quantizing Convolutional layers cause more loss in information than quantizing a dense layer. Hence, we can conclude that instead of quantizing or compressing all the layers uniformly we can selectively quantize/compress certain layers to mininmize loss in accuracy, this is exactly what we plan to do next through model pruning.

4.2.3 Comparative study of Pruning Techniques

We know that model Pruning is the process of removing unwanted weight connections in a network to increase inference speed and decrease model storage size. In section 2.3, we briefly described the two commonly used types of pruning, namely Weight and Neuron pruning. Here, we compare the drop is accuracy as we try to make the model as sparse as possible.

From figure 4.3, we can observe that Weight pruning outperforms neuron pruning for all values of k(% pruning). This can be attributed to that fact that pruning a neuron implies we zero out all the weights that go through it, thereby stalling the flow of information to a greater extent than setting a small portion of random weights to zero (in this case there is still a guaranteed path of information

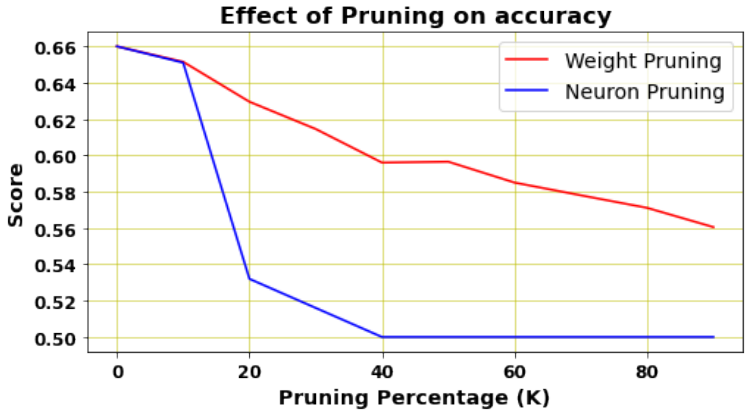


Figure 4.3: Effect of Pruning on accuracy

flow in most cases).

We can also observe that we are able to set upto 50% of weights to zero with an accuracy drop of less than 5%. From this we get the idea of using lesser number of weights to pass on sufficient information required by the model. This is the idea of Weight Sharing or clustering, which we will discuss in next section.

4.2.4 Effect on Model performance due to Weight clustering

From the previous two sections we have observed the advantage of selectively quantizing certain layers and minimizing the number of unique values in the network to efficiently decrease the size of the model without hurting the accuracy too much. In case of clustering, we reduce the number of unique weight values in a model by replacing similar weights in a layer with the same value.

In this section, we will perform experiments using the stand-alone Cat vs Dogs dataset from CIFAR-10 on Quant-NET model to cluster it's weight into 16 clusters. We also combine the methods of Quantization and weight clustering to check if it leads to any improvement. We also experimentally show in figure 4.4 why N_c=16 is the best number of clusters to group the weights into.

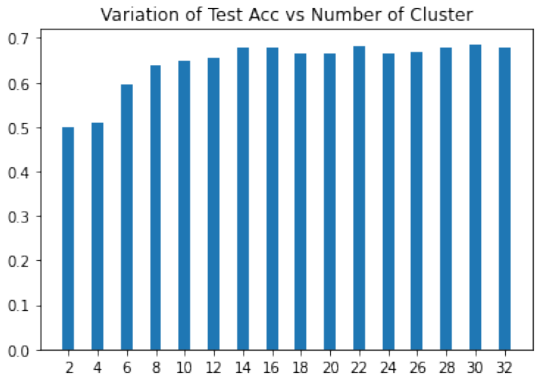


Figure 4.4: Variation of Test accuracy with Number of clusters

From the plot we can observe that optimal value for **N_c is 16**. There are cases with similar or higher accuracy for larger N_c (Eg: N_c = 30, 32), but we will be compromising on the Model compression factor by opting for larger N_c for a very small increase in model performance.

Table 4.4: Test accuracy and Size of model (MB) under different Clustering schemes schemes

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

From Table 4.4, we can see up to 9x shrinkage in model size by using simple

weight clustering technique with a small loss (0.7%) in accuracy. The sizes of Clustering Quantization aware-training models are lesser than the base-model unlike Table 4.2. General trends observed with Quantization techniques with base-model seems to be followed with Clustered model as well. From this table as well we can observe that **Post-training Quantization on Q-aware trained model** outperforms other techniques and gives the maximum **Accuracy per MB value**. We get a 12x decrease in model size from the base model for a mere drop of 0.4% in accuracy for the Clustering + Post-training Quantization on Q-Aware trained model.

Cluster preserving Q-Aware Training

The best method from table 4.4 can be considered as two-layered Compression with Clustering first and Q-aware training next. Even though this gives good accuracy and decrease in model size, usage of Q-aware training nullifies the Clusters the weights had been put into by the clustering technique in the previous step, thereby decreasing the overall effect. Hence, we now go ahead with Cluster preserving Q-aware training such that the number of clusters (unique weights) per layer remains the same even after Q-aware training.

Table 4.5: Comparative study of efficient clustering techniques

Set Up	**Test Acc**	**Number of clusters (Layerwise Unique Weights)**	**Model Size (MB)**
Base Model	66.00	896, 18496, 534416, 2064, 17	2.0852
Clustered (N_c = 16)	65.30	16, 16, 16, 16, 16	0.2562
Clustered (N_c = 16) + Q-aware Training	65.75	864, 18430, 361277, 1308, 16	1.6755
Cluster Preserving + Q-aware Training	65.85	16, 16, 16, 16, 16	0.2394

From table 4.5 we can observe that using cluster preserving version for Q-aware training helps us to reduce the model size drastically from 1.68 MB to 0.24 MB (7x decrease) and also not compromise on accuracy. Hence we go ahead with using a combination of cluster preserving Q-aware training for our final model. We come up with an idea to apply Clustering to insignificant layers of Quant-NET and Q-aware training for the remaining layers to strike an optimal balance between size reduction and accuracy maintenance while using TwoNN method to obtain the important layers. The idea to separate significant and insignificant layers was inspired from the success of Mixed precision networks, which we will see in our next section.

4.2.5 Study on Mixed Precision Quantization

In section 2.5, we looked at existing and popular mixed precision techniques like the XNOR-Net design and PCA Hybrid. 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. In our work we make use of TwoNN instead of PCA and reasons for the same are listed in section 3.3.1.

Next, the table 4.6 shows why the layer significance identification approach that for the standalone setup becomes applicable to the transfer learning setup as well. From table 4.6, 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

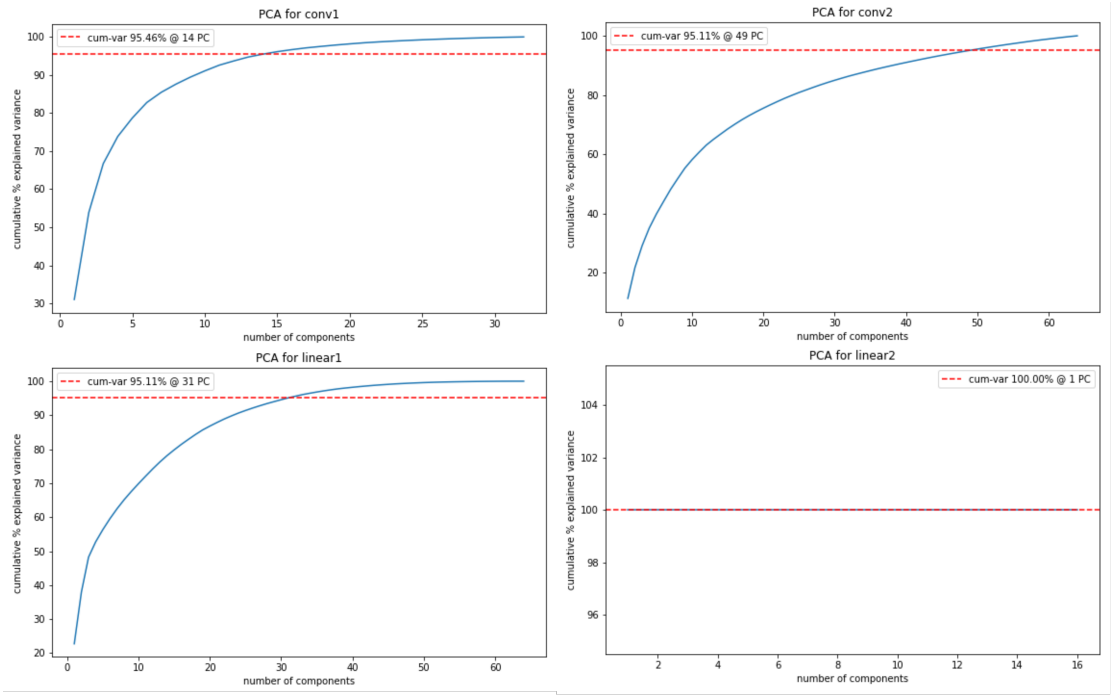


Figure 4.5: Principal Component Analysis on the intermediate layers of QuantNet for Horse vs Frogs dataset

Table 4.6: 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

model layers transform sequentially into nearly 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 representation's dimensionalities that Ansuini *et al.* [2019] stated. This supports the need for a better significance criteria that allows layers that also cause

a reduction in ID to be significant. In figure 4.5, 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 TwoNN method and absolute difference criteria for quantifying the layer significance.

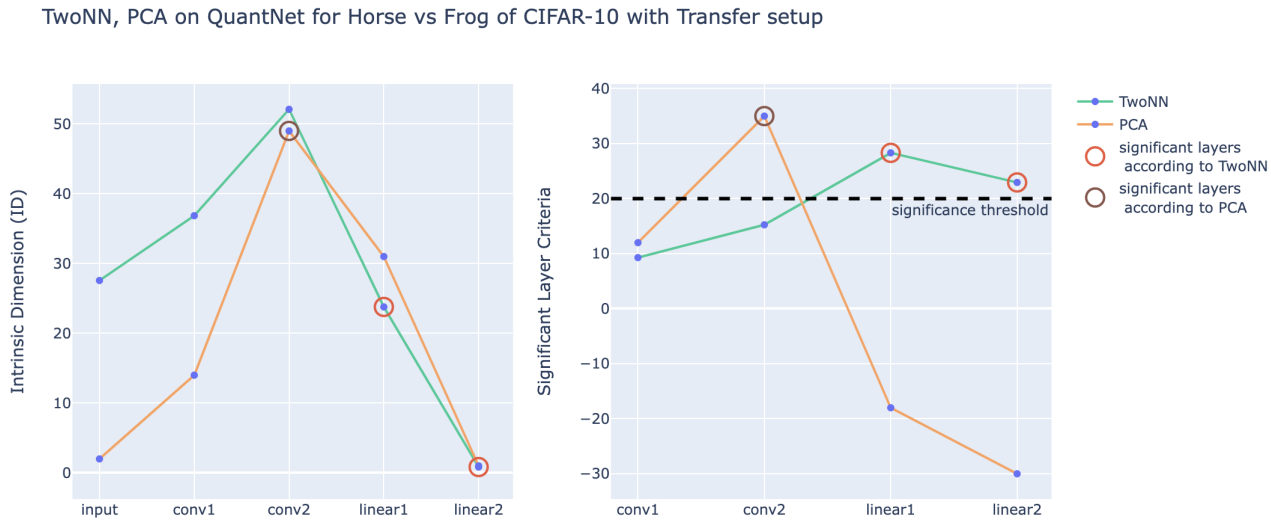


Figure 4.6: Identifying significant layers using TwoNN and PCA on QuantNet model for transfer domain in CIFAR-10 dataset, with threshold $\Delta = 20$

We now evaluate the PCA and TwoNN approach on the target and source domains. As we saw in table 4.6, 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 by Lee *et al.* [2015]. From figure 4.6, we can 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 4.7 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 to use TwoNN.

Table 4.7: Performance summary† of TwoNN and PCA approaches on Quant-Net model for Cat vs Dog source domain and Horse vs Frog target domain

Layer Precisions				**Test Acc (Source Domain)**	**Test Acc (Target Domain)**	**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***	65.60	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	91.60	15.87x
full	binary	2-bit	2-bit			

† 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

Model for Memory constrained settings

As mentioned in the later parts of section 3.4 we now present a model suited for memory constrained settings. With the layer significance identification using *TwoNN* approach working as expected, we attempt a hybrid precision layer design

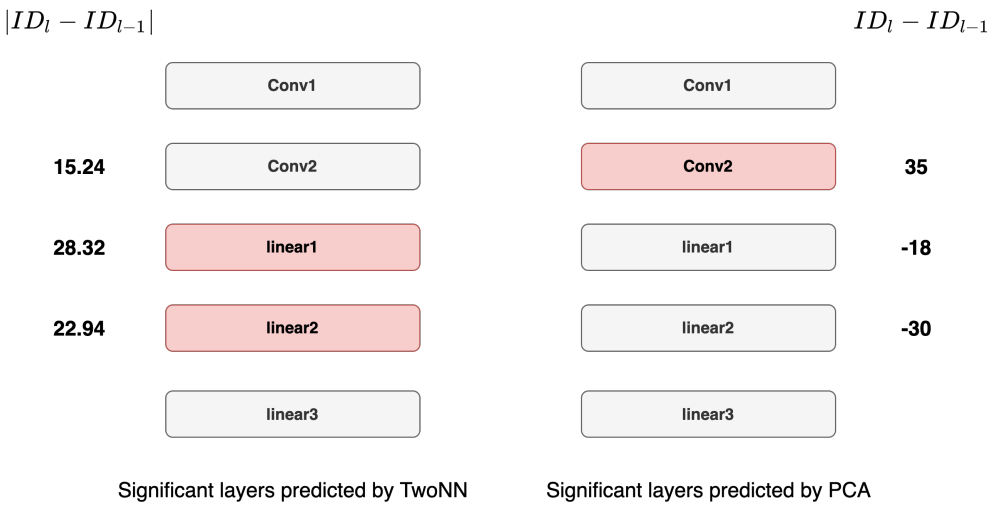


Figure 4.7: Significant layers (shaded red) identified by TwoNN (left) and PCA (right) with threshold $\Delta = 20$, along with the values of the significance criteria

on the QuantNet model. Setting a threshold $\Delta = 20$, we categorize the layers with significance value above it to be important and set to a higher precision of 2 bits, while binarizing the layers with significance value below Δ. We can see the performance for the model with this configuration in the last row of table 4.7 and as one might expect, it achieves high empirical reduction in size 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, this design achieves reduction next to XNOR-Net design (row 2 of table 4.7 with all intermediate layers set to binary precision) but with a considerably better generalizability and predicting capability.

4.2.6 Proposed Model vs Other best performing models

In this section we will compare the performances of our proposed compressing technique (TwoNN + Clustering + Q-Aware training) with the best performing models from each technique discussed so far.

Table 4.8: Test accuracy and Size of model (MB) of best performing compression schemes

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

All numbers are reported for Cat vs Dogs stand-alone setup

Table 4.8 clearly shows that our proposed model outperforms the best performing models from each of technique discussed so far. The closest to our model is the plain (without differentiating between the layers) Cluster preserving Q-aware training. In general, most of these constraints on the network is observed to have a regularizing effect aiding in improvement of better Test accuracy and model generazability. Also, as mentioned in the previous subsection, for memory constrained setting the TwoNN + Mixed Precision model is most suitable given it's model size and accuracy. We still persist with our proposed model for normal edge settings because of the compression we are able to achieve and the accuracy we are able to maintain. We firmly believe that this difference in accuracies will be much larger in case of complex datasets and complex tasks.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this work, we present an efficient approach to compress an image classification model in transfer learning setup. We introduce a new approach to combine the ideas of TwoNN dimension estimation method, Weight clustering and Quantization aware training to come up with an optimal pipeline for model compression.

We benchmark our model against the best models from all the other classical approaches discussed in the Experiments and results section. Subsequently, we demonstrate that our approach outperforms all these commonly used techniques in terms of accuracy and also achieves very good model compression.

Further, we explain our observations by analysing the results we get from various experiments and justify our choices of techniques that we choose to form our final model. We first show how the Post training + Q-aware training model outperformed the other Quantization techniques both in stand-alone and transfer learning setup. We then demonstrated that different layers in play different roles and selectively modifying certain parts of model will help in both model size reduction and also accuracy maintenance.

Further, we perform an important comparative study among the clustering and Clustering + Quantization techniques. We also demonstrated the advantage of using Cluster preserving Q-Aware training due to their cascading effect on model compressing. Then, we compare TwoNN and PCA methods for ID estimation and conclude that TwoNN is the better alternative and helps in improving generaliz-

ability in the compressed models. Finally we also compare the performance of our model with all the other best performing models.

In the future, we hope to extend the applicability of our approach to multi-class image classification models as well. We also intend to extend this work to tasks beyond classification like object detection, face detection etc. Given the surge in the usage of edge devices in recent times and there is a strong need for a fast and efficient Neural network model running on these devices and we wish to explore further avenues where the learnings from this work can be applied. Further, given that reinforcement learning is used in recent times for similar tasks we wish use that as a comparative to benchmark as well. We also want to see how the findings of our research might be used to design more efficient approaches for environments with far more limited memory.

REFERENCES

A. Kumar, S. G. and **M. Varma**, Resource-efficient machine learning in 2 kb ram for the internet of things. *In International Conference on Machine Learning*. 2017.

Andrew G Howard, B. C. D. K. W. W. T. W. M. A., Menglong Zhu and **H. Adam** (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

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*.

Benoit Jacob, B. C. M. Z. M. T. A. H. H. A., Skirmantas Kligys and **D. Kalenichenko** (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2704–2713.

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`.

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**.

Fengfu Li, B. L., Bo Zhang (2016). Ternary weight networks. *arXiv preprint arXiv:1605.04711*.

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**.

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

Hengyuan Hu, Y.-W. T., Rui Peng and **C.-K. Tang** (2016). "network trimming: A data-driven neuron pruning approach towards efficient deep architectures". *arXiv preprint arXiv:1607.03250*.

Hung, M. ().

Jie Hu, S. A. G. S. E. W., Li Shen (2017). Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*.

Krizhevsky, A., **V. Nair**, and **G. Hinton** (). Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Krizhevsky, A., **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12. Curran Associates Inc., Red Hook, NY, USA, 2012.

Kuan Wang, Y. L. J. L., Zhijian Liu and **S. Han** (2019). Haq: Hardware aware automated quantization with mixed precision, 8612–8620.

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.

Michael Zhu, S. G. (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

Mohammad Rastegari, J. R., Vicente Ordonez and **A. Farhadi** (2016). Xnor-net: Imagenet classification using binary convolutional neural networks, 525–542.

R. K. Naha, D. G. P. P. J. L. G. Y. X., S. Garg and **R. Ranjan** (). Fog computing: Survey of trends, architectures, requirements, and research directions.

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.-S. Bouganis**, **M. A. Hanif**, **F. Khalid**, **R. Hafız**, and **S. Rehman**, An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era. *In 2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018.

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

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. E. Reed**, **D. Anguelov**, **D. Erhan**, **V. Vanhoucke**, and **A. Rabinovich** (2014). Going deeper with convolutions. *CoRR*, **abs/1409.4842**. URL `http://arxiv.org/abs/1409.4842`.

Xiangyu Zhang, M. L., Xinyu Zhou and **J. Sun** (). Shufflenet: An extremely efficient convolutional neural network for mobile devices.

Yihui He, X. Z. and **J. Sun** (2017). *arXiv preprint arXiv:1707.06168*.