Model inversion for data-free quantization of neural networks

Author: Cillian Murphy
Industry Supervisor: Dr. Kate O' Bryne
Academic Supervisor: Dr. Andrew Keane

A thesis submitted in partial fulfilment of the requirements
for the degree of
MSc Mathematical Modelling and Machine Learning
School of Mathematical Sciences,
University College Cork,
Ireland

September 2024

Declaration of Authorship

This report is wholly the work of the author, except where explicitly stated otherwise. The source of any material which was not created by the author has been clearly cited.

Date: 02/10/2024

Signature: Cillian Murphy

Acknowledgements

I would first like to thank my supervisors Kate O' Byrne and Andrew Keane who's help and guidance have been invaluable thought this project. I would also like to thank Kate for the resources provided through Qualcomm as this project would not have been possible without it. A final thanks goes to my family who have supported me throughout the MSc program.

Abstract

Model quantization is a key technique for optimizing neural networks by reducing memory usage and computational cost, making them more efficient for deployment on resource constrained devices. However, quantization often requires access to calibration datasets, typically original training data, which are not always available. In this thesis the use of model inversion techniques to generate synthetic calibration datasets, from trained neural networks, for use in post training quantization is explored. It is demonstrated that statistical properties of the data used to train the network can be recovered using model inversion techniques and these recovered statistics can be used to generate a synthetic calibration dataset for use in quantization, achieving quantized model accuracies comparable to models quantized using original training data. In addition to this the use of the recovered training statistics to generate calibration datasets are a significant improvement over calibration datasets generated using random noise which is currently standard practice in industry. The results show the potential of model inversion techniques for data-free quantization of neural networks providing a possible solution when original datasets are unavailable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural networks have become an increasingly ubiquitous paradigm in machine learning, driving advancements across fields such as healthcare, autonomous systems and natural language processing. These models power real time applications such as medical imaging diagnostics, self driving cars and large scale language models where high accuracy and fast inference are important. However, as the size and complexity of these models grow, so do the demands of computational resources such as memory and power consumption. This increasing demand necessitates the development of techniques to reduce the memory and power usage of such models. The challenge of minimizing model size and energy consumption is not new and as far back as the 1980's and 1990's researchers were introducing model compression techniques such as model pruning [1] and weight sharing [2] in an attempt to address the relatively limited computation power available at the time. With the rapid advancement of neural networks over the past decade more sophisticated model compression techniques were required and have emerged.

Among the compression techniques model quantization has become one of the most effective and widely adopted methods. Quantization reduces the precision of model weights and activations, converting them from 32-bit floating point numbers to lower bit representations such as 8-bit integers, in a manner which aims to maintain the accuracy of the original model. Quantization significantly reduces both the memory requirements and the computational load during inference, making it a particularly valuable technique for deploying on devices with constraints on memory and power consumption such as mobile phones and embedded systems amongst others. Post-training quantization in particular, has gained significant traction as it enables model size reduction after the model is trained without the need to modify the training process of the network. This approach has potential to minimize any loss in model accuracy while achieving substantial gains in efficiency. Significant contributions to model quantization methods were made from 2015 on wards with many seminal papers including [3], [4], [5] contributing greatly to the field of model quantization.

To preform model quantization, especially when the activations of the model are to be quantized, a calibration dataset is required. This dataset, typically a representative subset of the original training data, is required to calculate the optimal parameters to quantize model activations in order to prevent drastic degradation of the model performance. The use of a calibration dataset ensures that quantized models maintain their accuracy by minimizing the degradation in performance that can arise from reduced precision. However, in many real world scenarios it may not be feasible to obtain the original training data to use as a calibration dataset for model quantization, due to privacy concerns, companies not wanting to share datasets with potential competitors or practical constraints such as the original dataset being too large to transfer efficiently. Without access to original training data, quantizing the activations of the model becomes more challenging as there is no way to directly calculate the optimal quantization parameters, which could lead to a significant drop in model performance.

After reviewing the previous work detailing model quantization, the main goals of this thesis were to explore techniques that allow post-training model quantization to be performed without access to the original training data, and to investigate methods for detecting any potential decrease in model accuracy post quantization again without access to original data. The main methods investigated to accomplish these tasks were model inversion techniques, with model inversion techniques using the information contained in the trained model itself, such as weights, activations and internal structures, to infer properties of the data used to train the model. In contrast to model compression techniques which were developed alongside neural networks, model inversion is a relatively new area of research. One of the first papers on model inversion by Fredrikson et al. [6] was published in 2015 which focused on using generative adversarial networks (GANs) trained on publicly available data to recreate images similar to those in the original models training set. Since this paper numerous other papers such as [7] and [8] have expanded on the method of using GANs for model inversion, however although this technique does manage to produce data which is representative of the original training data the reliance on large amounts of publicly available data makes this approach to model inversion unfeasible in a lot of cases.

More recent advancements in model inversion have provided alternative approaches that do not have the same reliance on external datasets. For instance, model inversion approaches presented in [9] and [10] provide a framework where the recovery of training data is framed as an optimization problem. These methods generate a set of representative images by by starting with an initial image of random noise and updating the image according to a specific loss function as it passed through the model. This process is iterative refining the input until it resembles the data used during training. These model inversion techniques offer a method to quantizing models effectively and detecting performance deviations post quantization.

This thesis aims to address two key challenges in neural network quantization, performing quantization without access to original training data and detecting accuracy degradation post quantization. By investigating and adapting real world techniques, this thesis seeks to provide novel solutions that are applicable to real-world scenarios, where privacy concerns or logistical constraints prevent the use of original data as calibration sets.

Chapter 2

Theoretical Background

This chapter provides an overview of the concepts required for understanding the methodologies used in this thesis. It begins by covering the basics of neural networks, including their architecture, activation functions, loss functions and optimization techniques such as gradient descent.Followed by a more in depth examination of two critical layer types, the convolutional and batch normalization layers, used in neural networks for image classification tasks. Furthermore model quantization is introduced as an optimization technique to improve efficiency of neural networks, as are model inversion techniques for use in generating synthetic calibration datasets when original training data is not available.

2.1 Basics of neural networks

Neural networks are a class of machine learning models which consist of interconnected layers of neurons that transform input data through a series of weighted connections and biases. This section provides an overview of the fundamental concepts of neural networks, including their architecture, learning process.

2.1.1 Neural network architecture

A neural network is composed of three main types of layers: input, hidden, and output layers.

The input layer receives the raw data and passes it to the subsequent layers of the network. Each neuron in the input layer represents a feature of the input data. For example, in an image classification task, each pixel value of an image might be represented by a neuron in the input layer. Hidden layers are intermediate layers between the input and output layers. They perform the majority of computations and learn complex features from the input data. Each hidden layer consists of neurons that apply nonlinear transformations to the data. These transformations are governed by activation functions, which introduce a non-linearity to the model, enabling it to learn complex patterns. The output layer produces the final predictions or classifications of the network. In a classification task, the output layer typically consists of neurons equal to the number of classes, with each neuron representing the probability of a specific class.

2.1.2 Activation functions

Activation functions determine the output of each neuron in the hidden and output layers. They introduce non-linearity into the network, allowing it to model complex relationships. Common activation functions include:

Sigmoid function The sigmoid function maps input values to a range between 0 and 1, making it suitable for binary classification tasks:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

ReLU (Rectified Linear Unit) The ReLU function outputs the input directly if it is positive; otherwise, it outputs zero:

$$ReLU(x) = \max(0, x)$$

with the ReLU activation function being almost exclusively used in the models presented in later sections of this thesis.

2.1.3 Loss function & optimization

Loss functions measure the difference between the network's predictions and the actual target values. The most common loss function for classification tasks such as those present in this thesis is the cross entropy loss function defined as:

$$L_{CE} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where y_i represents the true class label and $\hat{y}_i$ is the predicted probability for class i.

Once the loss function is defined, the process of minimizing it is typically done using an optimization algorithm, with one of the most basic algorithms being gradient descent. Gradient descent works by iteratively updating the model's parameters to minimize the loss function by moving in the direction of the steepest descent, which is the negative gradient of the loss with respect to the model's parameter's. Given parameters θ (e.g. weights and biases) of the model, gradient descent updates the parameters at each iteration t as :

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta)$$

where:

- θ_t are the parameters at iteration t
- η is the learning rate
- $\nabla_\theta L(\theta)$ is the gradient of the loss function with respect to the parameters θ

The gradient $\nabla_\theta L(\theta)$ represents the direction in which the parameters should be adjusted to minimize the loss, the learning rate η is important in ensuring stable convergence with a large learning rate possibly causing the minimum to be overshot, while a small learning rate may lead to slow convergence.

2.1.4 Convolutional layer

Convolutional layers are used extensively in modern neural network architectures especially in tasks which relate to computer vision such as image classification and are used throughout the networks presented later in the thesis. Convolutional layers are designed to learn spatial features of input images by using the cross-correlation operator which involves sliding a filter over the input data and calculating the dot product between the filter and the patch of the input data the filter is currently on as shown in fig. 2.1. This cross correlation operator can be expressed mathematically as:

$$y(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(i+m, j+n) \cdot F(m,n) + b$$

where:

- $y(i,j)$ is the output at position (i,j).
- M and N are the dimensions of the filter.
- $x(i,j)$ is the input value at position (i,j).
- $F(m,n)$ is the filter value at position (m,n).
- b is the bias term.

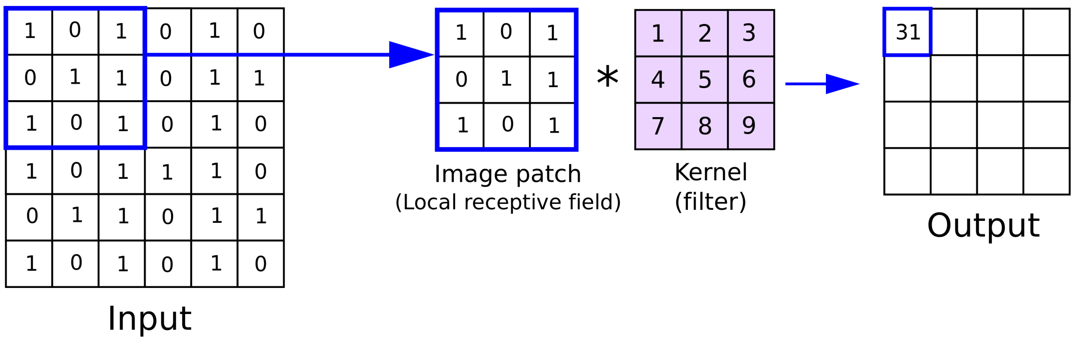


Figure 2.1: A visual representation of a convolutional layer in a neural network [11].

A more in depth look at convolutional layers in neural networks can be found in [12].

2.1.5 Batch normalization layer

The batch normalization (BN) layer is another layer which is widely used throughout neural network architectures due to their ability to allow for increased training stability and higher learning rates. These layers work by normalizing the input data from mini batches during training of a model so that the output of the BN layer has zero mean and unit variance [13].

For a given mini-batch of training data $X = x_1, x_2, ...x_m$, where m is the batch size, batch normalization transforms each feature x_i by first computing the batch mean and variance:

$$\text{Batch mean: } \mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\text{Batch variance: } \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

The normalized output of each element of the mini-batch is then computed as:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where ϵ is a small constant added to prevent division by zero.

Furthermore, the BN layer also applies two learnable parameters, γ and β, which scale and shift the normalized values:

$$y_i = \gamma \hat{x}_i + \beta$$

.

During the training of a neural network the BN layer maintains a running average of the training data mean and variance, denoted as μ^{BN} and $(\sigma^{BN})^2$, respectively. These running statistics are used during inference to normalize the input data instead of using batch statistics, such that during inference the output of the BN layer is:

$$y_i = \gamma \cdot \frac{x_i - \mu^{BN}}{\sqrt{(\sigma^{BN})^2 + \epsilon}} + \beta$$

The running mean and variance approximate the actual mean and variance of the training dataset, which is important for maintaining model performance during inference. A more detailed explanation of the BN layer can be found in [13].

2.2 Model quantization

Quantization is an optimization technique for neural network models that reduces the precision of weights and layer outputs (known as activations) from 32-bit floating point precision to lower bit-widths such as 8-bit integer precision. This significantly reduces memory usage and computational complexity during inference, which is crucial for deploying neural networks on resource-constrained devices, such as mobile phones or embedded systems. Quantization allows models to run efficiently while aiming to maintain the accuracy of the original model. This section provides a theoretical overview of model quantization, with a particular focus on post-training quantization (PTQ).

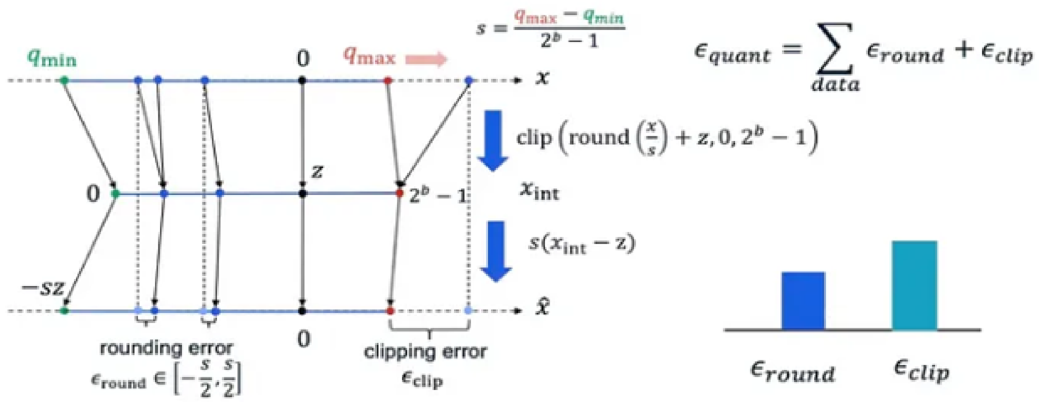
2.2.1 Quantizing network weights and activations

Quantizing a neural network involves approximating the 32-bit floating point representations of weights and activations with a discrete set of lower precision integers, typically in 8-bit integer representation. The general goal of quantization is to replace high precision floating point values with low-precision integer values to reduce memory usage and improve computational efficiency.

The key concepts of the quantization process are shown in fig. 2.2, which highlights the key components involved in converting floating point numbers into quantized integers and vise versa. These include:

- q_{min} & q_{max}: The minimum and maximum range of values to be quantized.

- b: The bit size used for quantization.
- s: The scale factor, representing the resolution of the quantized grid.
- z: The zero-point, which ensures that zero in floating point domain maps exactly to an integer in the quantized domain.
- ϵ_{round} & ϵ_{clip}: Errors introduced during quantization. ϵ_{round} refers to rounding errors whereas ϵ_{clip} refers to errors when values are clipped to fit within the quantization range.



$s = \frac{q_{max} - q_{min}}{2^b - 1}$
q_{min}
0
q_{max}
x
$\epsilon_{quant} = \sum_{data} \epsilon_{round} + \epsilon_{clip}$
clip $\left(\text{round}\left(\frac{x}{s}\right) + z, 0, 2^b - 1\right)$
0
z
$2^b - 1$
x_{int}
$s(x_{int} - z)$
$-sz$
$\hat{x}$
rounding error
$\epsilon_{round} \in \left[-\frac{s}{2}, \frac{s}{2}\right]$
0
clipping error
ϵ_{clip}
ϵ_{round}
ϵ_{clip}

Figure 2.2: Visual representation of the quantization process, including the range for quantization q_{min} & q_{max}, scale factor s, zero-point z, and errors due to rounding and clipping during quantization. [14]

Let x_{float} represent the original floating point value of a weight or activation, and let x_{quant} represent the quantized value. The most general form of quantization, known as asymmetric quantization, can be described by the following formula:

$$x_{\text{quant}} = \text{clip}\left(\text{round}\left(\frac{x_{\text{float}}}{s}\right) + z, q_{min}, q_{max}\right)$$

where s is the scaling factor, for quantizing to a bit width of b and is calculated as:

$$s = \frac{q_{max} - q_{min}}{2^b - 1}$$

and z is the integer zero-point that corresponds to the true zero in the floating-point range:

$$z = \text{round}\left(\frac{0 - \min(x_{\text{float}})}{s}\right)$$

Here, $\text{clip}\,(x, q_{min}, q_{max})$ ensures that the quantized values remain in the valid integer range and round() rounds the result to the nearest integer. In cases where asymmetric quantization is not required, symmetric quantization can be used, where x_{quant} is given by:

$$x_{\text{quant}} = \text{clip}\left(\text{round}\left(\frac{x_{\text{float}}}{s}\right)\right)$$

and the scale factor s is computed as:

$$s = \frac{\max(|x_{\text{float}}|)}{2^b - 1}$$

Asymmetric quantization is more flexible than symmetric quantization, as it handles skewed distributions better by using a non-zero z point, but incurs additional computational overhead whereas symmetric quantization is simpler and computationally more efficient, but less suitable for weight or activation distributions which are skewed and not symmetric around zero. When weights or activations are quantized, the values x_{quant} are constrained to the integer range $[-2^{b-1}, 2^{b-1} - 1]$ for signed integers and $[0, 2^{b-1} - 1]$.

2.2.2 Example calculation

The process of quantization can be demonstrated by considering a tensor of weights W and an input tensor X using the symmetric quantization approach for weights and asymmetric quantization for activations. The tensors are given by:

$$W = \begin{pmatrix} 1.2 & -0.8 & 0.5 & 2.3 \\ -1.0 & 0.7 & -0.3 & 0.4 \\ 2.0 & -1.5 & 1.1 & 0.9 \\ 0.2 & -0.4 & 1.6 & -1.3 \end{pmatrix}$$

$$X = \begin{pmatrix} 0.5 & -1.2 & 2.1 & -0.7 \\ 1.3 & -0.9 & 0.6 & 1.9 \\ 0.8 & -0.2 & -1.0 & 2.4 \\ -1.1 & 0.3 & 1.0 & -0.5 \end{pmatrix}$$

For 8-bit quantization, using asymmetric quantization the scale factor for the weights is calculated as:

$$s_W = \frac{\max(W) - min(W)}{255}$$

$$\Rightarrow s_W = \frac{2.3}{255} \approx 0.0149$$

and zero point for the weights is :

$$z_W = \text{round}\left(\frac{-min(W)}{s_W}\right)$$

$$\Rightarrow z_W = \text{round}(100.67) = 101$$

For the input tensor X, using asymmetric quantization, the scale factor is calculated as:

$$s_X = \frac{\max(X) - \min(X)}{255}$$

$$\Rightarrow s_X = \frac{3.6}{255} \approx 0.0141$$

and zero-point is:

$$z_X = \text{round}\left(\frac{-\min(X)}{s_X}\right)$$

$$\Rightarrow z_X = \text{round}(85.11) = 85$$

The components of the quantized weights and inputs, W_{quant} and X_{quant} respectively are computed using:

$$w^{i,j}_{\text{quant}} = \text{round}\left(\frac{w^{i,j}}{s_W}\right) + z_W$$

$$x_{\text{quant}}^{i,j} = \text{round}\left(\frac{x^{i,j}}{s_X}\right) + z_X$$

These give the quantized tensors $\hat{W}$ and $\hat{X}$:

$$W \approx \hat{W} = s_W W_{\text{quant}}$$

$$\Rightarrow W \approx \hat{W} = s_W \begin{pmatrix} 182 & 47 & 135 & 255 \\ 34 & 148 & 81 & 128 \\ 235 & 0 & 175 & 161 \\ 114 & 74 & 208 & 14 \end{pmatrix}$$

$$X \approx \hat{X} = s_X X_{\text{quant}}$$

$$\Rightarrow X \approx \hat{X} = s_X \begin{pmatrix} 120 & 0 & 234 & 35 \\ 177 & 21 & 127 & 220 \\ 142 & 71 & 14 & 255 \\ 7 & 106 & 156 & 50 \end{pmatrix}$$

If the multiplication of W and X are considered such that:

$$Y = W \cdot X$$

Then the output Y can be approximated by:

$$Y \approx \hat{Y} = s_W(W_{\text{quant}} - z_W) \cdot s_X(X_{\text{quant}} - z_X)$$

$$\Rightarrow \hat{Y} = s_W s_X W_{\text{quant}} \cdot X_{\text{quant}} - s_W s_X z_X W_{\text{quant}} - s_W s_X z_W X_{\text{quant}} + s_W s_X z_W z_X \quad (2.1)$$

which gives:

$$\hat{Y} = \begin{pmatrix} -2.57 & -0.14 & 3.85 & -2.3 \\ -0.27 & 0.74 & -0.98 & 1.13 \\ -1.07 & -0.99 & 3.10 & -2.07 \\ 2.28 & -0.57 & -2.73 & 3.56 \end{pmatrix}$$

To quantify the error due to quantization, the absolute error is computed as:

$$\epsilon = |Y - \hat{Y}|$$

where Y is the exact matrix multiplication result of $W \cdot X$ in floating point precision:

$$Y = \begin{pmatrix} -2.57 & -0.13 & 3.84 & -2.31 \\ -0.27 & 0.75 & -0.42 & 1.11 \\ -1.06 & -1.00 & 3.10 & -2.06 \\ 2.29 & -0.59 & -2.72 & 3.59 \end{pmatrix}$$

which gives an absolute error of:

$$\epsilon = \begin{pmatrix} 0.00 & 0.01 & 0.01 & 0.01 \\ 0.00 & 0.01 & 0.56 & 0.02 \\ 0.01 & 0.01 & 0.00 & 0.01 \\ 0.01 & 0.02 & 0.01 & 0.03 \end{pmatrix}$$

.

The error accumulation present in this case is due to rounding only as no clipping of values outside the maximum range for quantization was performed.

2.2.3 Calibration datasets

Although Eq. 2.1 does not appear simplify computation, it enables pre-computation of terms such as $s_W s_X z_X W_{\text{quant}}$, reducing the the number of computations which must be carried out during inference, which leaves the multiplication of the quantized tensors W_{quant} and X_{quant}. A critical part of this process is selecting a calibration dataset, which is a representative subset of the training data. Passing this data through the network helps determine the optimal scale and zero-point values for activations, allowing quantization parameters to be set before inference which reduces the memory usage and .latency during inference

In cases where a representative dataset is unavailable (e.g., due to privacy concerns), model inversion techniques can be used to estimate appropriate activation distributions, allowing for data-free model quantization. A more in depth look at quantization procedure can be found in [15].

2.2.4 Cosine similarity

Cosine similarity is a method of measuring the similarity between two vectors often used in industry to compare the output vectors of two models, typically the outputs of a quantized model with the outputs of the original model. The cosine similarity between two vectors A and B is given by the formula:

$$\text{Cosine similarity} = \frac{A \cdot B}{||A||||B||}$$

where $A \cdot B$ is the dot product of the vectors and $||A||$ and $||B||$ representing the magnitude of the vectors. The result is a value between -1 and 1, where 1 indicates that the vectors are perfectly aligned, 0 means they are orthogonal to each other and -1 means they are pointing in opposite directions.

2.3 Model inversion

With model quantization becoming an increasingly popular technique to decrease model size and latency, the lack of available datasets which can readily be used as calibration datasets is major hindrance to the quantization process. This section focuses on model inversion techniques which aim to recover information about the data used to train the network and it is hoped that the information recovered from the trained network can be used to generate a synthetic dataset for use as a calibration dataset.

2.3.1 DeepInversion

Deep Inversion is a model inversion technique designed to generate realistic synthetic images from a trained neural network without direct access to the original training data. The method uses a combination of losses designed to reconstruct images that are close to the original data distribution and match the statistical properties of features learned by the network. This section outlines the various components of the DeepInversion loss and their role in generating images.

2.3.2 Loss Function Composition

The loss function used in DeepInversion combines multiple parts that help guide the generated image towards satisfying characteristics of the target class while maintaining image realism. The overall loss is defined as [9]:

$$L = L_{\mathrm{CE}} + \alpha_v L_{\mathrm{v2}} + \alpha_f L_{\mathrm{feat}} + \alpha_{l2} L_{\mathrm{l2}}$$

The individual terms present in the loss are:

Cross-Entropy Loss ($L_{\mathbf{CE}}$): This term ensures that the model generates an image that is classified by the network as the desired target class. Specifically, it measures the difference between the model's predicted probability distribution for the generated image and a one-hot encoding of the target class. Mathematically, this is expressed as:

$$L_{\mathrm{CE}} = -\sum_c y_c \log(p_c)$$

where y_c is the one-hot encoded target class and p_c is the predicted probability for class c. This term drives the generated image to maximize the model's confidence in classifying the image as belonging to the target class.

Total Variation Loss ($L_{\mathbf{v2}}$): The total variation loss encourages smoothness in the generated image by penalizing pixel intensity differences along the horizontal, vertical, and diagonal directions. This regularization term reduces noise and artifacts, producing visually coherent images. It is defined as:

$$L_{\mathrm{v2}} = \sum_{i,j} \sqrt{(y_{i+1,j} - y_{i,j})^2 + (y_{i,j+1} - y_{i,j})^2 + (y_{i+1,j+1} - y_{i,j})^2 + (y_{i+1,j-1} - y_{i,j})^2}$$

where $y_{i,j}$ represents the pixel value at position (i, j). The terms ,$(y_{i+1,j} - y_{i,j})^2$ and $(y_{i,j+1} - y_{i,j})^2$, promotes image smoothness in the horizontal and vertical axes, with the inclusion of both diagonal terms, $(y_{i+1,j+1} - y_{i,j})$ and $(y_{i+1,j-1} - y_{i,j})$, promoting smoothness also across diagonal pixel transitions. Minimizing this loss encourages spatial continuity between neighboring pixels, which is important for generating visually plausible images with smooth gradients.

Feature Regularization Loss (L_{feat}): To ensure that the generated image aligns with the statistical properties of the network's learned feature representations, DeepInversion incorporates a feature regularization loss based on the Batch Normalization (BN) statistics. During training, batch normalization layers compute and store the mean and variance of feature activations across batches. The feature regularization loss is defined as:

$$L_{\text{feat}} = \sum_{l} \left(|\mu_l - \mu_l^{BN}|_2 + |\sigma_l - \sigma_l^{BN}|_2 \right)$$

where μ_l and σ_l are the mean and variance of the feature activations at layer l for the generated image, and μ_l^{BN} and σ_l^{BN} are the corresponding stored values from the batch normalization layers. By aligning the feature statistics of the generated image with those learned during training, the network is encouraged to produce images with the data distribution on which the model was originally trained.

L2 Regularization Loss (L_{l2}): This term introduces additional regularization by penalizing the L2 norm of the pixel values in the generated image. It helps prevent the generation of extreme pixel values, which can disrupt the learning process. The L2 loss is computed as:

$$L_{\text{l2}} = \frac{1}{C} \sum_{c} \|x_c\|_2$$

where C is the number of input channels (e.g., 1 for gray-scale images, 3 for RGB images) and x_c represents the pixel values in channel c. This loss encourages the generated images to have a moderate range of pixel intensities, contributing to the overall stability of the image generation process.

The coefficients α_v, α_f, and α_{l2} control the relative importance of the different loss components. Selecting optimal values for these coefficients is important to generating images which are recognisable and are distinct between separate classes. A common approach to determine these hyper-parameters is a grid search, where different combinations of the α values are evaluated to find the best combination.

2.3.3 Enhancing DeepInversion with image loss

A further enhancement to the DeepInversion loss involves incorporating an additional term that directly aligns the mean and variance of the generated image with the target statistics. This is particularly useful when the true distribution of the training data is known or can be estimated from the network. The image loss is defined as:

$$L_{\text{image}} = \text{MSE}(\mu_{\text{image}}, \mu_{\text{target}}) + \text{MSE}(\sigma_{\text{image}}, \sigma_{\text{target}})$$

where μ_{image} and σ_{image} represent the mean and standard deviation of the generated image, and μ_{target} and σ_{target} are the known or estimated mean and standard deviation of the original training data. This term ensures that the overall statistics of the generated image match those of the training data, further improving the fidelity of the synthesized images.

2.3.4 Recovering Training Data Statistics

Instead of focusing on recovering recognizable images from the network, an alternative approach is to recover the mean and standard deviation of pixel values from the training dataset. This is particularly useful for incorporating the image loss discussed in the previous section, and can be used to generate a synthetic calibration dataset on their own.

To recover these statistics, the running mean and standard deviation stored in the first batch normalization layer of the network can be used. However, it is important to note that the first batch normalization layer in most neural networks typically occurs after a convolutional layer. Since the convolution operator is generally not invertible, a full analytic approach is not possible and an optimization based approach must be used to approximate the statistics of the training data.

The process begins by initializing a random noise image. This image is passed through the initial layers of the network, which typically consist of a convolutional layer followed by a batch normalization layer. Once the input reaches the first batch normalization layer, the mean and standard deviation of the image at this point are calculated. To recover the training data statistics, the following loss function is used:

$$L_{\text{stat}} = \text{MSE}(\mu_r, \mu_{\text{BN}}) + \text{MSE}(\sigma_{\text{r}}, \sigma_{\text{BN}})$$

where μ_r and σ_r are the recovered mean and standard deviation of the input image at the batch normalization layer, and μ_{BN} and σ_{BN} are the running mean and standard deviation stored in the batch normalization layer.

The error computed from this loss function is back-propagated to the input image, updating its pixel values accordingly. With each update, the input image becomes more representative of the mean and standard deviation of the original training data. After several iterations of this optimization process, the input image's pixel statistics should theoretically converge to match those of the training dataset.

Chapter 3

Results

This chapter presents the results obtained from the model inversion process, aimed at recovering both images and image statistics from a pre-trained neural network. The outcomes of the model inversion are further applied to model quantization, demonstrating that using the recovered mean and standard deviation of the training data as a calibration dataset significantly outperforms models quantized with random noise as calibration data. It is also shown that cosine similarity or investigating layer activations are not reliable methods for detecting degradation in model performance.

3.1 Model inversion for recovering training images

After training a model on the MNIST dataset, which consists of single-channel images with dimensions of 28×28 pixels representing handwritten digits from 0 to 9, the model inversion method and loss function described in section 2.3.1 were implemented to generate images from each of the ten classes in the dataset. Fig. 3.1 presents a sample of the generated images, where fig. 3.1a shows an initial random noise image to be optimized using the loss function outlined in section 2.3.1. Figs. 3.1b, 3.1c, and 3.1d display the recovered images of handwritten digits '0', '3', and '5', respectively, each generated from an initial noise image similar to that shown in 3.1a. From figs. 3.1a to 3.1d, it can be seen that this method successfully recovers images that resemble the distribution of the training data, with clear separation between the different classes present in the dataset. The images presented here are representative of all the digits generated using this method with all digits being recognisable with a clear separation between classes.

Having confirmed the efficacy of this method for simple single channel images such as those present in the MNIST dataset, the same approach was applied to the more complex CIFAR10 dataset, which consists of three-channel images (RGB) with dimensions of 32×32 pixels. The CIFAR10 dataset contains 10 classes, representing images of airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Fig.3.2 shows a sample of the images generated using this approach. In fig. 3.2, 3.2a depicts a random noise image to be optimized, while figs. 3.2b, 3.2c, and 3.2d display the generated images corresponding to the classes 'airplane', 'car', and 'bird', respectively, all derived from initial random noise images like the one in fig. 3.2a. However, unlike the images generated from the network trained on the MNIST dataset, the images produced from the CIFAR10 trained network did not resemble any of the training data. Attempts were made to optimize the hyper-parameters in the loss function to produce more representative images for the CIFAR10 classes, but a sufficiently fine grid search proved computationally prohibitive. The coarse grid search that was performed did not yield noticeable improvements in image quality.

Furthermore, as discussed in [9], generating images for the Imagenet dataset using this method required 128 GPUs and 22 hours of training to produce recognizable images. Despite the theoretical validity of this approach, as shown in the results in fig. 3.1, the computational demands of optimizing the loss function hyper-parameters and generating higher resolution images made it infeasible to carry out within the scope of this thesis, resulting in an alternative model inversion technique being sought for data-free model quantization.

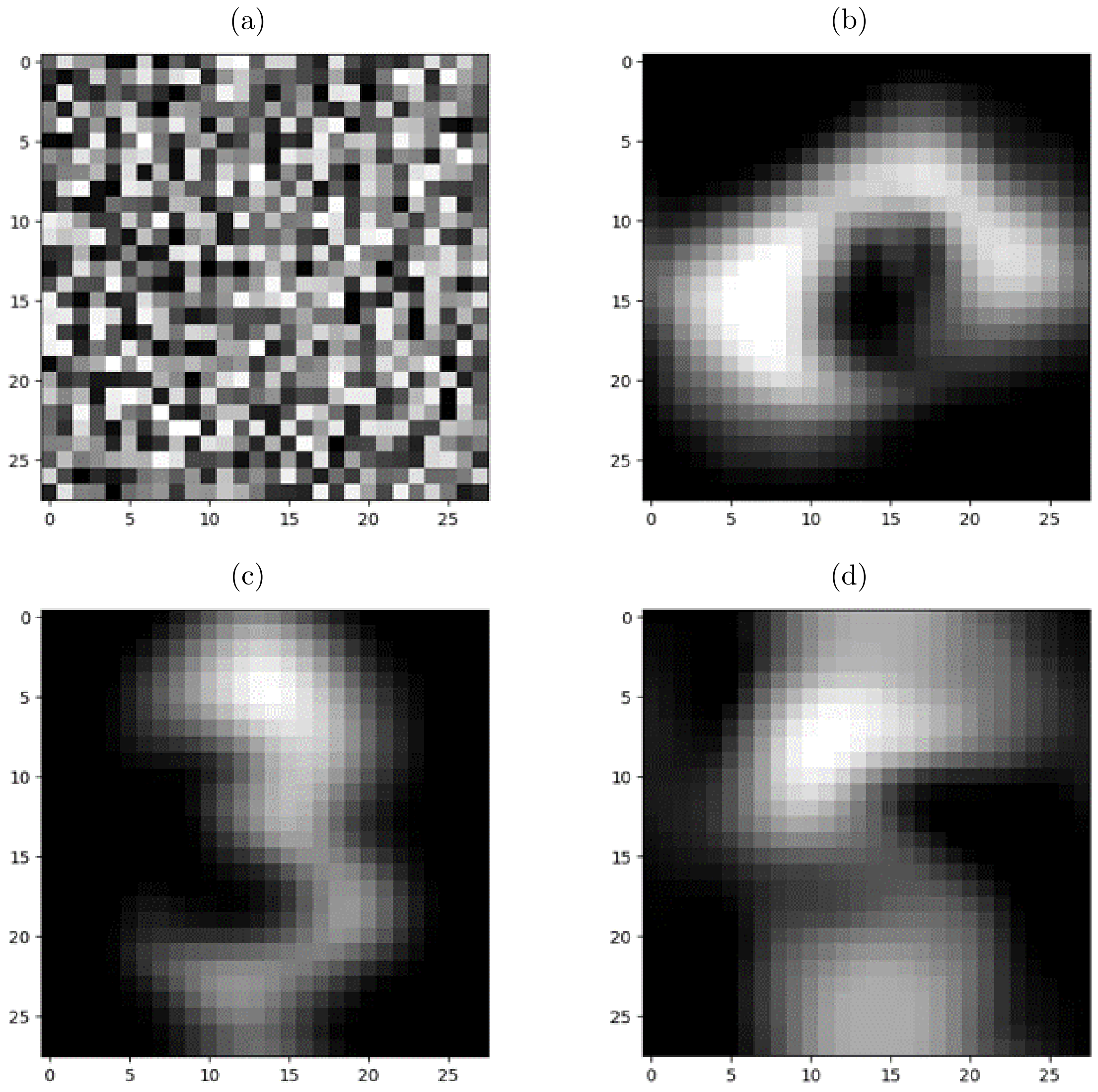


Figure 3.1: (a) Example initial image of random noise to be optimized using DeepInversion loss.(b) Recovered image from class 0 of MNIST data set, generated using DeepInversion loss. (c) Recovered image from class 3 of MNIST data set, generated using DeepInversion loss. (d) Recovered image from class 5 of MNIST data set, generated using DeepInversion loss.

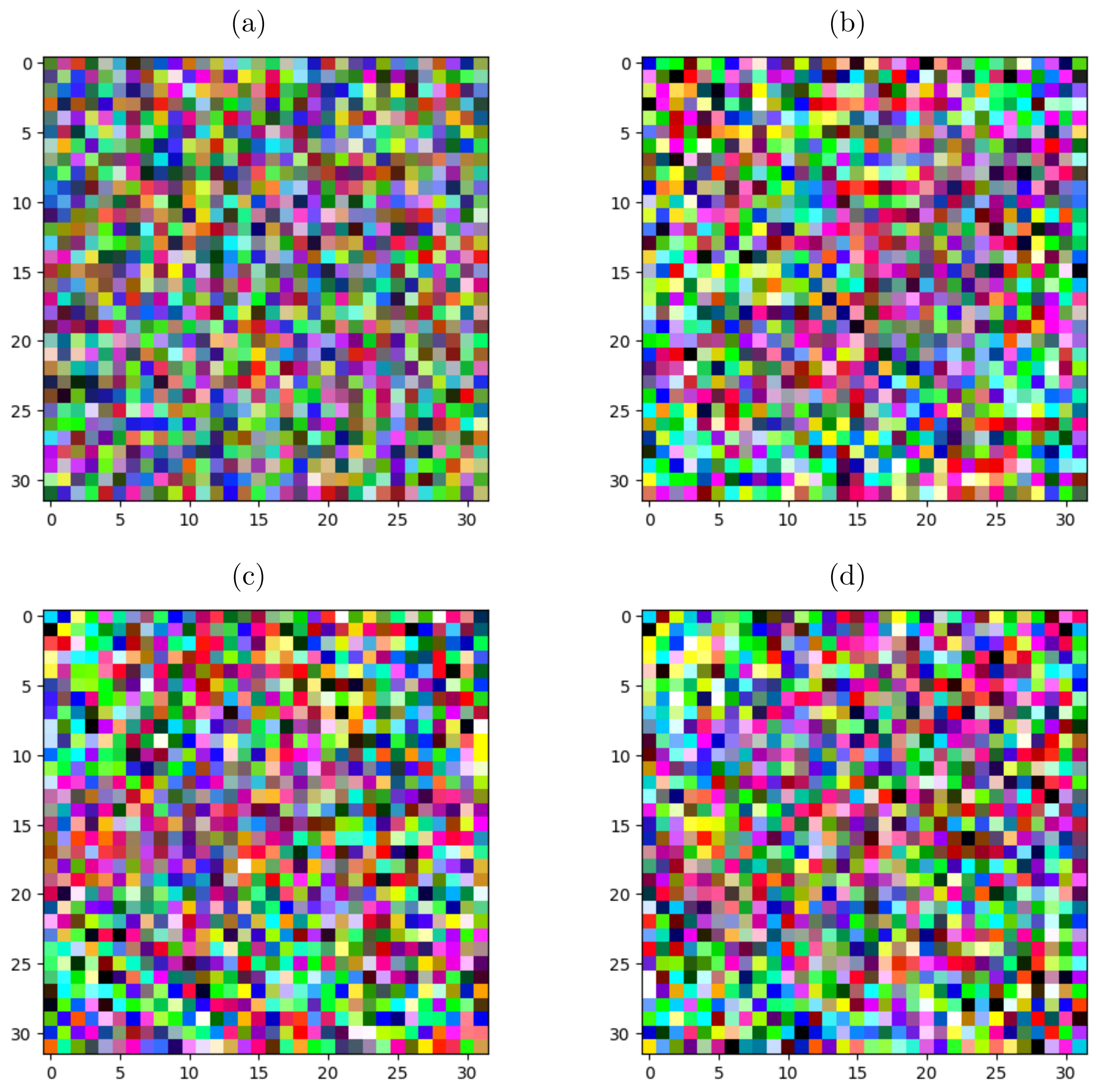


Figure 3.2: (a) Example initial image of random noise to be optimized using DeepInversion loss. (b) Recovered image from class 0 of CIFAR10 dataset, generated using DeepInversion loss. (c) Recovered image from class 1 of CIFAR10 dataset, generated using DeepInversion loss. (d) Recovered image from class 2 of CIFAR10 dataset, generated using DeepInversion loss.

3.2 Model inversion for recovering image statistics

Given the prohibitive computational expense of generating entire images from a trained network, the approach outlined in section 2.3.4 was used. This approach aimed to recover the mean and standard deviation of the training images from a trained network, and to use images generated with these recovered statistics as a calibration dataset for model quantization. As a proof of concept, this method where back-propagation was applied from the first batch normalization layer only was first tested on a model trained with the MNIST dataset. The results, presented in table 3.1, show that while the recovered mean and standard deviation do not exactly match the actual values, they are reasonably close to the expected values.

	Actual	Recovered
Mean	0.13	0.17
Std	0.31	0.33

Table 3.1: Table of actual vs recovered statistics of training images for a network trained on MNIST dataset.

Once the mean and standard deviation of the MNIST training images were found to be approximately in line with the actual values, the same procedure was applied to a model trained on the CIFAR10 dataset. The results for the CIFAR10 model are shown in table 3.2, with tables 3.2a, 3.2b, and 3.2c detailing the actual versus recovered statistics for channels 0, 1, and 2 of the CIFAR10 training images. Similar to the results for the MNIST model in table 3.1, the recovered means and standard deviations for the CIFAR10 model do not perfectly align with the actual values but remain relatively close to the expected values.

	Actual	Recovered
Mean	0.49	0.46
Std	0.25	0.29

(a)

	Actual	Recovered
Mean	0.48	0.45
Std	0.24	0.27

(b)

	Actual	Recovered
Mean	0.45	0.42
Std	0.26	0.27

(c)

Table 3.2: Tables of actual vs recovered statistics for each channel of training images for a network trained on CIFAR10 dataset with tables (a), (b) and (c) showing results for channels 0, 1 and 2, respectively.

3.3 Model inversion results applied to quantization

Having demonstrated that the mean and standard deviation of training data images can be recovered with a high degree of accuracy, the next step was to investigate the use of images generated with the recovered statistics as a calibration dataset for model quantization. For this section, only CIFAR10 models were explored, as models trained on the MNIST dataset are relatively simple and can be quantized using random noise with little to no drop in accuracy. Three different models trained on the CIFAR10 dataset were tested: ResNet32, VGG13, and MobileNetV2-x1-0, in order to ensure that the generated images can be used as a calibration set across different architectures, rather than the results being specific to a particular model type.

For each of the three models, four different calibration datasets were investigated:

- The original training data.

- Images generated using the recovered mean and standard deviation, modeled as a normal distribution $N(\mu_r, \sigma_r)$, where μ_r and σ_r represent the recovered mean and standard deviation, respectively, from the trained model.
- Random noise sampled from a uniform distribution between 0 and 1, denoted $U(0, 1)$.
- Random noise sampled from a uniform distribution between -1 and 1, denoted $U(-1, 1)$.

The first random noise dataset, $U(0, 1)$, was chosen because pixel values in typical image datasets are scaled to the range $[0, 1]$. The second random noise dataset, $U(-1, 1)$, was selected because examining the recovered statistics showed that the normal distribution $N(\mu_r, \sigma_r)$ approximately falls within this range for most pixel values, except for some outliers. Images generated from this normal distribution $N(\mu_r, \sigma_r)$ thus represent a better approximation of the training data.

These calibration datasets were used to determine the optimal scale and zero-point for activation quantization, as discussed in section 2.2, for each of the three CIFAR10 models.

The resulting accuracies for each calibration set and model are shown in table 3.3. For the ResNet32 model, with an original accuracy of 0.9321, as can be seen in table 3.3a using the original training data as the calibration set led to the highest quantized model accuracy of 0.9297, quantizing the model using images generated with the recovered mean and standard deviation $N(\mu_r, \sigma_r)$ resulted in the second-highest accuracy of 0.9011, using random noise from $U(0, 1)$ produced the lowest accuracy of 0.5749, while random noise from $U(-1, 1)$ yielded a quantized accuracy of 0.8876. A similar trend was observed for the VGG13 model as shown in table 3.3b, with an original accuracy of 0.9369, the quantized accuracies were 0.9367, 0.9115, 0.6126, and 0.9041 for the respective calibration datasets of original training images, images generated from $N(\mu_r, \sigma_r)$, images generated from $U(0, 1)$ and images generated from $U(-1, 1)$. The MobileNetV2-x1-0 model, with an original accuracy of 0.9363, as can be seen in table 3.3c showed quantized accuracies of 0.9345, 0.9117, 0.6094, and 0.9016 for the same respective calibration sets.

Calibration Dataset	Quantized Model Accuracy
Original Training Data	0.9297
Recovered Mean/Std $N(\mu_r, \sigma_r)$	0.9011
Random Noise $U(0, 1)$	0.5749
Random Noise $U(-1, 1)$	0.8876

(a) ResNet32, original model accuracy 0.9321.

Calibration Dataset	Quantized Model Accuracy
Original Training Data	0.9367
Recovered Mean/Std $N(\mu_r, \sigma_r)$	0.9115
Random Noise $U(0, 1)$	0.6126
Random Noise $U(-1, 1)$	0.9041

(b) VGG13, original model accuracy 0.9369.

Calibration Dataset	Quantized Model Accuracy
Original Training Data	0.9345
Recovered Mean/Std $N(\mu_r, \sigma_r)$	0.9117
Random Noise $U(0, 1)$	0.6094
Random Noise $U(-1, 1)$	0.9016

(c) MobileNetV2-x1-0, original model accuracy 0.9363.

Table 3.3: Quantized model accuracies for different calibration datasets.

As shown in tables 3.3a, 3.3b, and 3.3c, the ideal case of using the original training data for calibration resulted in the best quantized model accuracies, with only a slight drop in accuracy compared to the original models. However, using images generated with the recovered mean and standard deviation $N(\mu_r, \sigma_r)$ provided an effective alternative calibration dataset, with only a 2–3% drop in accuracy for all models. In contrast, using random noise sampled from $U(0, 1)$ resulted in a significant accuracy drop of over 30% for all models, while random noise sampled from $U(-1, 1)$ performed better but still worse than the images generated using the recovered mean and standard deviation.

3.4 Determining model accuracy post quantization from model outputs

As outlined in section 2, cosine similarity between the outputs of the original and quantized models is often used to measure any deviation in model accuracy post-quantization. This section investigates whether any decrease in model accuracy can be determined using cosine similarity without relying on original test images.

To determine if cosine similarity could be used as a measure of model accuracy between original and quantized models, three sets of 100 images were passed through both the original and quantized Resnet32, VGG13, and Mobilenetv2-x1-0 models, and the average cosine similarities were found. The first dataset consisted of images generated from random noise sampled from $U(0,1)$, the second dataset comprised images generated using a normal distribution $N(\mu_r, \sigma_r)$ with recovered mean μ_r and standard deviation σ_r from the models as shown in section 3.2, and the last set consisted of original test images. The quantized models compared to the original unquantized models in this thesis were selected based on their performance during quantization using different calibration datasets of:

- Original training data, which served as a baseline calibration set.
- Images generated using a normal distribution $N(\mu_r, \sigma_r)$, where μ_r and σ_r denote the recovered mean and standard deviation, respectively. This was identified as the best data-free method for quantization in the previous section.
- Random noise sampled from $U(0,1)$, found to be the worst performing data-free method in the previous section.

These choices allowed for the effectiveness of each quantization method to be evaluated by comparing the accuracy of the quantized models with that of the original unquantized models.
The results for the Resnet32 model are presented in table 3.4, with tables 3.4a, 3.4b, and 3.4c showing the cosine similarity of the outputs of quantized models

compared to the original model when passing images generated from random noise, images generated using $N(\mu_r, \sigma_r)$, and original test images, respectively.

Quantized Model Accuracy	Cosine similarity
0.5749	0.9981
0.9011	0.9977

(a) Using images generated from random noise sampled from a uniform distribution between 0 and 1 to test cosine similarity between original and quantized models.

Quantized Model Accuracy	Cosine similarity
0.5749	0.9942
0.9011	0.9977

(b) Using images generated using $N(\mu_r, \sigma_r)$ to test cosine similarity between original and quantized models.

Quantized Model Accuracy	Cosine similarity
0.5749	0.6370
0.9011	0.9741

(c) Using original test images of CIFAR10 dataset to test cosine similarity between original and quantized models.

Table 3.4: Tables showing the cosine similarity between original and quantized versions of Resnet32 network trained on CIFAR10 images, original model accuracy 0.9321.

The results for the VGG13 model are presented in table 3.5, with tables 3.5a, 3.5b, and 3.5c showing the cosine similarity of the outputs of quantized models compared to the original model under the same calibration datasets as the case for table 3.4.

The results for the MobileNetv2-x1-0 model are presented in Table 3.6, with Tables 3.6a, 3.6b, and 3.6c showing the cosine similarity of the outputs of quantized models compared to the original model using the same calibration set as in the previous two tables 3.4 and 3.5.

From the tables 3.4, 3.5, and 3.6 presented above, it can be seen that using images generated from random noise or from $N(\mu_r, \sigma_r)$ does not result in any measurable difference between the outputs of quantized and original models, even

Quantized Model Accuracy	Cosine similarity
0.6126	0.9981
0.9115	0.9977

(a) Using images generated from random noise sampled from a uniform distribution between 0 and 1 to test cosine similarity between original and quantized models.

Quantized Model Accuracy	Cosine similarity
0.6126	0.9942
0.9115	0.9977

(b) Using images generated using $N(\mu_r, \sigma_r)$ to test cosine similarity between original and quantized models.

Quantized Model Accuracy	Cosine similarity
0.6126	0.6370
0.9115	0.9741

(c) Using original test images of CIFAR10 dataset to test cosine similarity between original and quantized models.

Table 3.5: Tables showing the cosine similarity between original and quantized versions of VGG13 network trained on CIFAR10 images, original model accuracy 0.9369.

when the quantized models have significantly lower accuracies. The only apparent way to use cosine similarity to detect any decrease in model performance post quantization is by using original test data to compare both the original and quantized models.

Other methods of comparing outputs, such as Jensen-Shannon Divergence and MSE between the outputs of the original and quantized models, were also investigated. However, differences were only detectable when using original test data, so these methods have been omitted from this dissertation.

3.5 Determining model accuracy post quantization from model activations

Instead of restricting the analysis of model accuracy to comparing the outputs of the quantized and original models, a more in depth look at the layer activations was

Quantized Model Accuracy	Cosine similarity
0.6094	0.9981
0.9117	0.9977

(a) Using images generated from random noise sampled from a uniform distribution between 0 and 1 to test cosine similarity between original and quantized models.

Quantized Model Accuracy	Cosine similarity
0.6094	0.9942
0.9117	0.9977

(b) Using images generated using $N(\mu_r, \sigma_r)$ to test cosine similarity between original and quantized models.

Quantized Model Accuracy	Cosine similarity
0.6094	0.6370
0.9117	0.9741

(c) Using original test images of CIFAR10 dataset to test cosine similarity between original and quantized models.

Table 3.6: Tables showing the cosine similarity between original and quantized versions of MobileNetv2-x1-0 network trained on CIFAR10 images, original model accuracy 0.9363.

investigated to determine if the accuracy of a quantized model could be inferred using this method. To gather the layer activations of the quantized and original models, separate datasets than those used in the quantization of the models were required. These datasets will be referred to as activation datasets. The following activation datasets were used:

- Original test images.
- Images generated using $N(\mu_r, \sigma_r)$, where μ_r and σ_r denote the recovered mean and standard deviation, respectively.
- Random noise sampled from $U(0, 1)$.

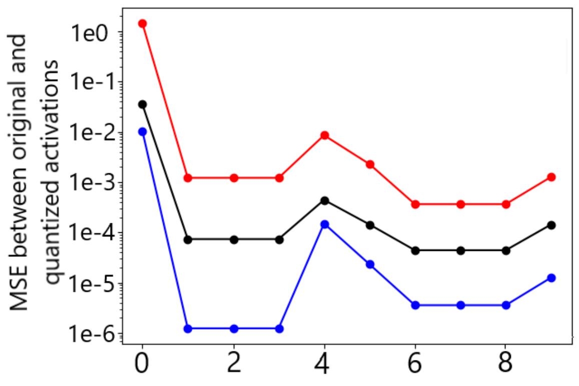
In each of the figures 3.3, 3.4, and 3.5, three models quantized with calibration datasets of original training images, images generated using $N(\mu_r, \sigma_r)$, and images generated from $U(0, 1)$ were compared with the original model. The Mean Squared Error (MSE) between the layer activations of the quantized and original models was calculated to compare the activations.

Figs. 3.3, 3.4, and 3.5 show the MSE between the activations of quantized and original models for a Resnet32, VGG13, and Mobilenetv2-x1-0 model, respectively. Specifically:

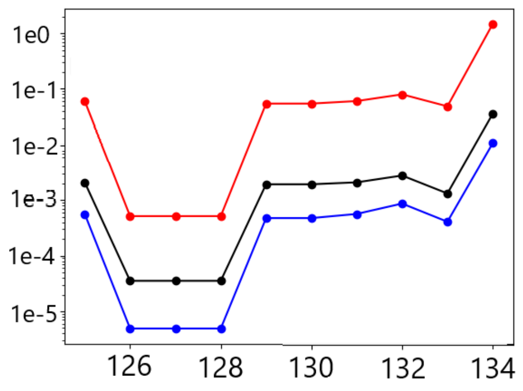
- Figs. 3.3a, 3.4a, and 3.5a show the activations of the first 10 layers of the Resnet, VGG, and Mobilenet models, respectively, when an activation dataset of original test images was used.
- Figs. 3.3c, 3.4c, and 3.5c show the activations of the first 10 layers of the Resnet, VGG, and Mobilenet models, respectively, when an activation dataset of images generated with $N(\mu_r, \sigma_r)$ was used.
- Figs. 3.3e, 3.4e, and 3.5e show the activations of the first 10 layers of the Resnet, VGG, and Mobilenet models, respectively, when an activation dataset of images generated from $U(0, 1)$ was used.
- Figs. 3.3b, 3.4b, and 3.5b show the activations of the last 10 layers of the Resnet, VGG, and Mobilenet models, respectively, when an activation dataset of original test images was used.
- Figs. 3.3d, 3.4d, and 3.5d show the activations of the last 10 layers of the Resnet, VGG, and Mobilenet models, respectively, when an activation dataset of images generated with $N(\mu_r, \sigma_r)$ was used.
- Figs. 3.3f, 3.4f, and 3.5f show the activations of the last 10 layers of the Resnet, VGG, and Mobilenet models, respectively, when an activation dataset of images generated from $U(0, 1)$ was used.

From figs. 3.3, 3.4, and 3.5, it can be observed that there is only a clear separation between the model activations when original test images are used to gather the activations. Quantized models with higher accuracies show lower MSE between the quantized and original models. When activation sets of images generated with $N(\mu_r, \sigma_r)$ or images generated from $U(0, 1)$ are used, no clear separation is present between the MSE of the quantized and original model activations, despite some cases showing almost 35% differences in model accuracy. These results suggest that the activation datasets presented here, except for original test images, cannot be used to infer the model accuracy of quantized models.

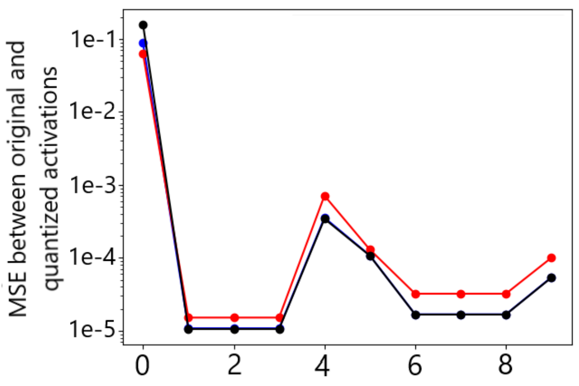
Other methods of comparing layer activations between quantized and original models were also considered, such as the structural similarity index measure (SSIM) and the Jensen-Shannon divergence, were also investigated. However, as was the case for comparing the MSE between layer activations differences were only detectable when using original test data, so these methods have been omitted from this thesis.



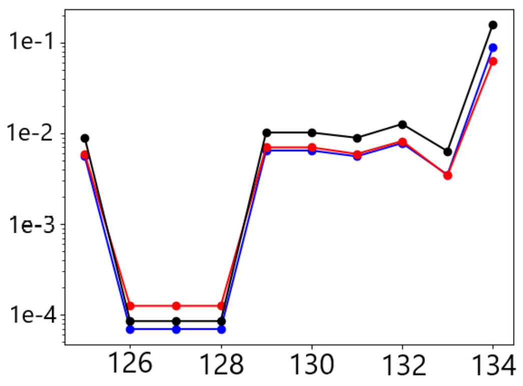
(a)



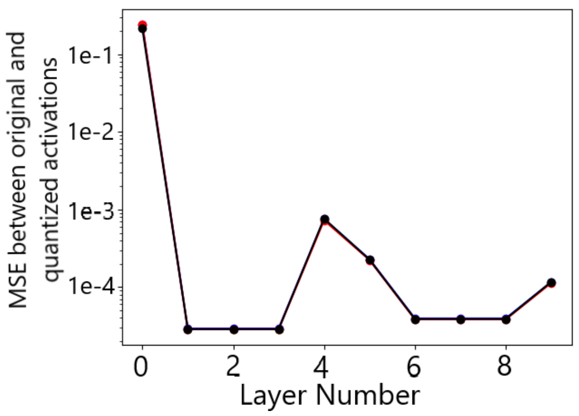
(b)



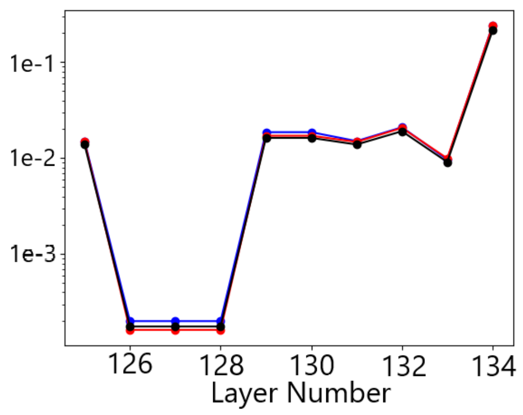
(c)



(d)



(e)



(f)

Figure 3.3: MSE between the original Resnet32 model and three models quantized with calibration datasets of original training images (blue line), recovered image statistics, $N(\mu_r, \sigma_r)$, (black line) and random noise, $U(0,1)$, (red line). (a) & (b) First and last 10 layers, respectively, with original test images as activation set; (c) & (d) First and last 10 layers, respectively, with images from $N(\mu_r, \sigma_r)$ as activation set; (e) & (f) First and last 10 layers, respectively, with images from $U(0,1)$ as activation set.

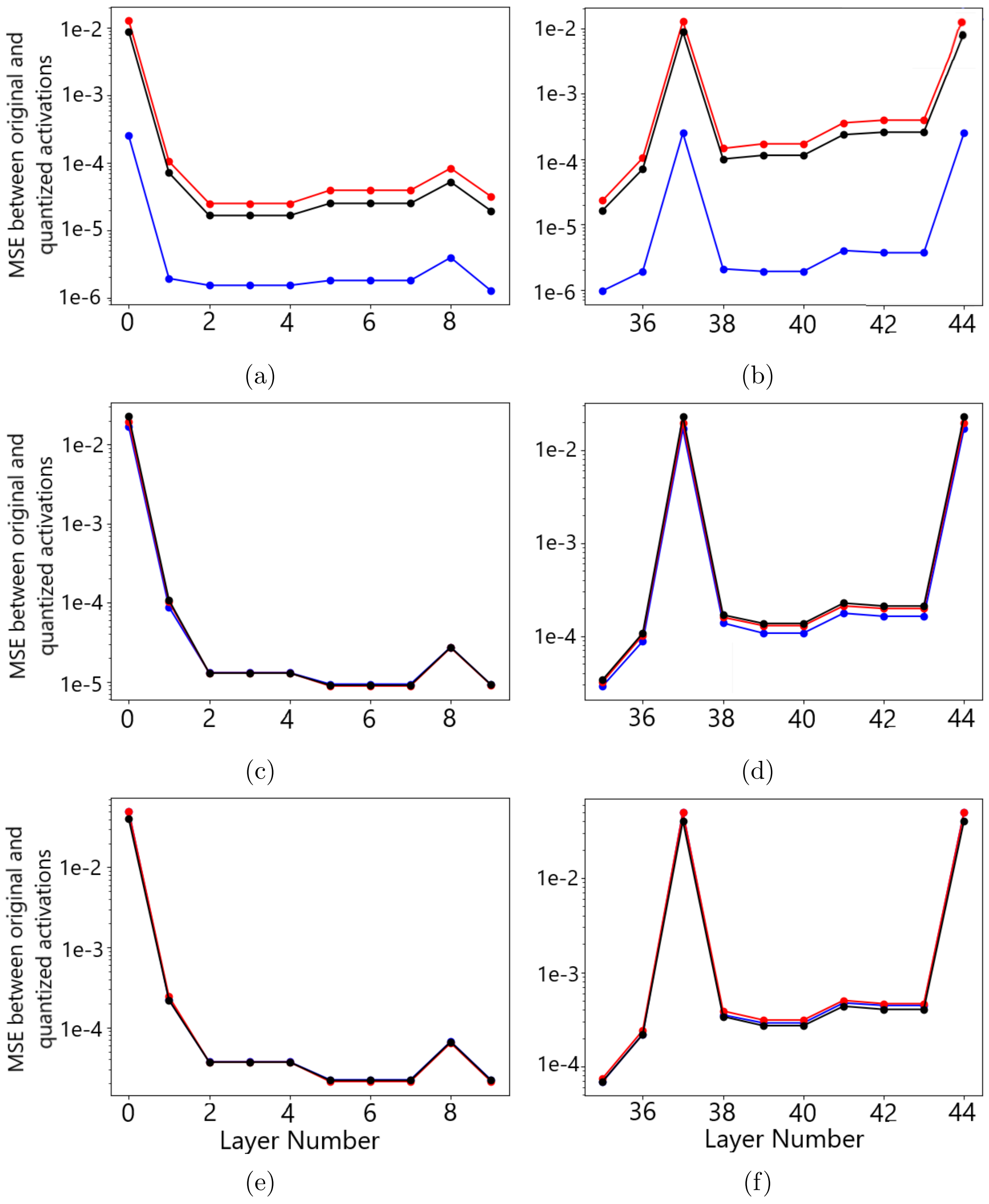
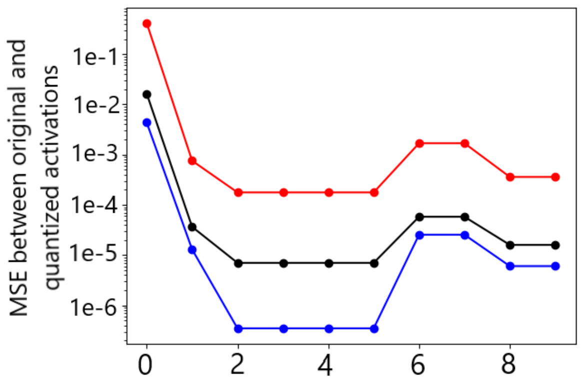
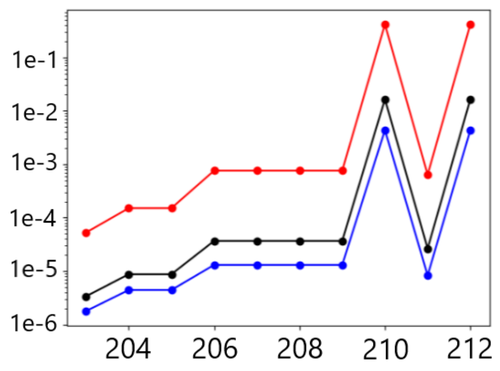


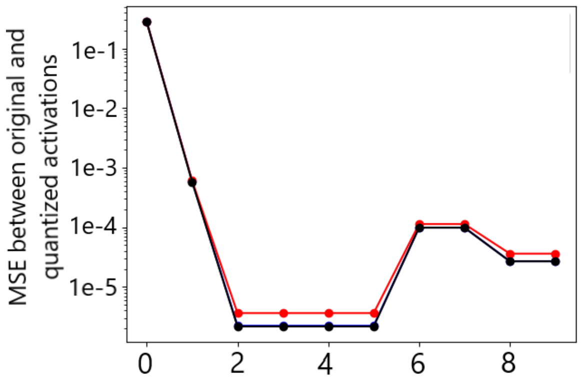
Figure 3.4: MSE between the original VGG13 model and three models quantized with calibration datasets of original training images (blue line), recovered image statistics, $N(\mu_r, \sigma_r)$, (black line) and random noise, $U(0, 1)$, (red line). (a) & (b) First and last 10 layers, respectively, with original test images as activation set; (c) & (d) First and last 10 layers, respectively, with images from $N(\mu_r, \sigma_r)$ as activation set; (e) & (f) First and last 10 layers, respectively, with images from $U(0, 1)$ as activation set.



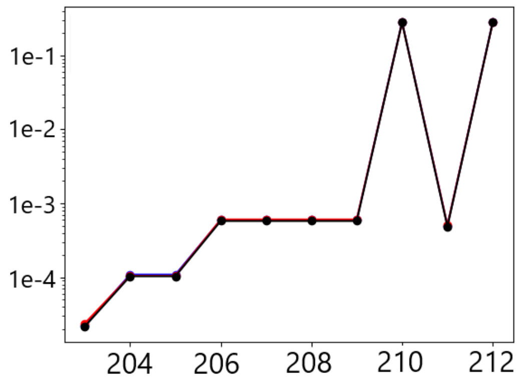
(a)



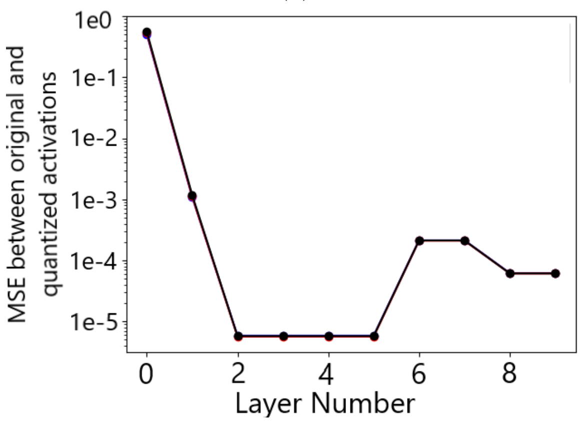
(b)



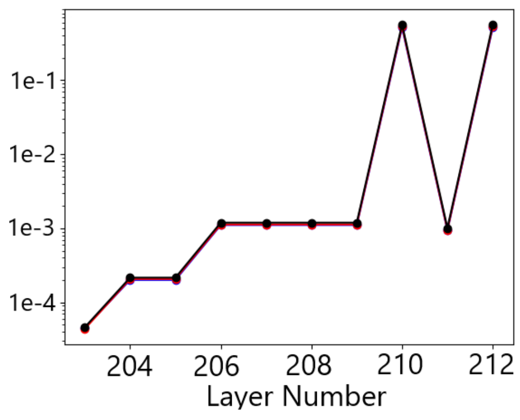
(c)



(d)



(e)



(f)

Figure 3.5: MSE between the original Mobilenetv2-x1-0 model and three models quantized with calibration datasets of original training images (blue line), recovered image statistics, $N(\mu_r, \sigma_r)$, (black line) and random noise, $U(0, 1)$, (red line). (a) & (b) First and last 10 layers, respectively, with original test images as activation set; (c) & (d) First and last 10 layers, respectively, with images from $N(\mu_r, \sigma_r)$ as activation set; (e) & (f) First and last 10 layers, respectively, with images from $U(0, 1)$ as activation set.

Chapter 4

Discussion

This chapter interprets the results from applying model inversion techniques to recover training data and image statistics, as well as their impact on model performance during model quantization. It discusses the effectiveness of using recovered statistics to generate calibration datasets as well as the limitations of this method. The reliability of different metrics, including cosine similarity and MSE, for detecting performance degradation in quantized models using both model outputs and model activations. Challenges faced when applying these methods to complex data sets are highlighted and outlines potential areas of improvement for the methods presented.

4.1 Interpretation of results

In the results section, the use of model inversion methods for recovering training data and image statistics was investigated, along with the effects of these recovered statistics on model performance during quantization. Additionally, methods for inferring model accuracy from outputs and activations were explored. This section provides an interpretation of the results and their impact on data-free model quantization.

The model inversion method described in section 2.3.1 was first applied to a model trained on the MNIST dataset, which consists of 28x28 pixel gray-scale images of handwritten digits. The method successfully reconstructed images from random noise, producing clear representations for each digit class, as shown in fig. 3.1. The results, illustrated in figs. 3.1b, 3.1c, and 3.1d, demonstrate that the recovered images closely resembled the original training data, with distinct differences between the digit classes. After the success on MNIST, a relatively simple dataset, the same method was applied to the more complex CIFAR-10 dataset, which contains 32x32 pixel RGB images across 10 classes, including airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

When this approach was applied to a model trained on the CIFAR10 dataset, the inversion method struggled to generate high quality images as shown in fig. 3.2. Despite attempts to optimize the loss function hyper-parameters, the generated images shown in figs. 3.2b, 3.2c, 3.2d, did not capture the details of the training data. The poor image quality, combined with the high computational cost of hyper-parameter tuning, showed a limitation of this approach for more complex datasets, particularly large-scale datasets as demonstrated in [9].

To overcome the limitations of the above model inversion approach, an alternative method focusing on the recovery of image statistics, in particular the mean and standard deviation, was explored. This method was applied to neural networks trained on both MNIST and CIFAR-10 datasets with the recovered statistics being found to be close to the actual values. These statistics were then used to generate calibration datasets for model quantization. The results indicated that images generated from the recovered statistics served as effective calibration data, leading to quantized model accuracies comparable to those achieved using the original training data. This demonstrates that using generated images based on recovered statistics offers a viable alternative for data-free model quantization, significantly outperforming datasets generated from random noise.

To assess performance degradation due to quantization, cosine similarity between the outputs of original and quantized models was initially investigated. The objective was to measure the alignment between the predictions of the quantized models and the original models without relying on the original test data. However, the results as shown in tables 3.4a, 3.5a, 3.6a, indicated that cosine similarity remained high even when the quantized models had significantly lower accuracies. This shows that cosine similarity is not a reliable measure of model performance degradation when using images generated from random noise or normal distributions with the recovered mean and standard deviation. In contrast to this, when cosine similarity was calculated using original test images as shown in tables 3.4c, 3.5c and 3.6c, a clear correlation with model performance degradation was observed, showing the importance of representative datasets in evaluating the effects of quantization.

Further investigation into model performance post quantization was carried out by comparing the MSE between the activations of quantized and original models. Three different activation datasets were used: original test images, images generated from a normal distribution $N(\mu_r, \sigma_r)$ using the recovered mean and standard deviation, and random noise. This analysis showed that using original test images had a strong correlation between model accuracy and MSE. Models with higher post quantization accuracy had a lower MSE in their activations, demonstrating that original test images provide a reliable metric for evaluating the impact of quantization on model performance. In contrast, datasets generated from recovered image statistics or random noise did not show a clear distinction in MSE between quantized and original models. Although the generated images were statistically similar to the training data by design, they failed to capture activation changes resulting from quantization. Furthermore, random noise datasets were found to be ineffective in distinguishing MSE values for quantized and original models.

Overall, the results show the importance of using activation datasets that closely resemble the distribution of the training data to infer model accuracy degradation. Original test images were most effective for both quantizing and evaluating the impact of quantization on model accuracy, while synthetic or random datasets showed limitations in reliably predicting performance degradation due to quantization, however synthetic images generated with the recovered mean and standard deviation appear to be viable in use as calibration data sets.

4.2 Limitations and future work

4.2.1 Training image recovery

While the methods used in this study provided insights into model inversion and data-free model quantization, there were some limitations in the methods used. One of the primary limitations lies in the computational expense associated with image generation for complex datasets like CIFAR10 or Imagenet, as discussed in section 2.3.1. The high computational demand for hyper-parameter tuning and image generation suggests that this methods may not be scalable for larger datasets or more complex models, which is shown by the poor performance on CIFAR10 dataset compared to the MNIST dataset. Future work could explore more efficient optimization techniques for this inversion method to recover images which are representative of the original training images as this could allow for better data-free model quantization and provide a method to detect accuracy degradation of models post quantization.

4.2.2 Training statistics recovery

Additionally, the use of recovered image statistics, while effective for generating calibration data did not allow for the detection in accuracy degradation post model quantization. This inability to capture changes in model accuracy post quantization, could lead to a limitation in the ability of the recovered statistics to effectively perform as a calibration dataset for more complex networks or datasets. Another factor which may limit the effectiveness of this approach is the assumption that the training images follow a normal distribution. If the actual distribution of the training data does not agree with this assumption the effectiveness of using these recovered statistics may be limited, particularly with more complex datasets. Future work could investigate more advanced methods of statistical recovery which do not rely on thisd assumption, or explore incorporating additional information about the training data, such as class specific features through the use of sensitivity and uncertainty analysis as presented in papers such as [16], [17] and [18], to better approximate the original training data which may allow for both better data-free model quantization and for the detection of accuracy degradation without the use of original training data .

4.2.3 Accuracy metrics

Another limitation concerns the evaluation metrics used for both comparing model outputs and model layer activations. For comparing model outputs cosine similarity proved inadequate in detecting performance degradation, when datasets other than the original images were used, and other metrics such as the Jensen-Shannon divergence and MSE also showed the same limited effectiveness with datasets generated using the recovered mean and standard deviation and those generated from random noise. For comparing layer activations metrics such as MSE, SSIM and Jensen-Shannon divergence again proved inadequate for detecting performance degradation, when datasets other than the original images were used. Future work could focus on using more advanced evaluation metrics such as other metrics presented in [19] that could potentially better capture performance degradation without original test data, enabling more accurate detection of model accuracy post quantization.

4.2.4 Further model testing

The analysis on model quantization in this thesis focused on the quantization of three distinct model architectures which included Resnet, VGG and Mobilenet models. Future work could focus on expanding the number of models investigated using recovered image statistics as calibration data sets for quantization, and incorporating some of the mention improvements in the above sections for improving image statistics recovery and the accuracy metrics used for accuracy evaluation post quantization may provide further insights into data-free model quantization.

4.3 Conclusion

In this thesis, the effectiveness of model inversion techniques and the recovery of training image statistics for data-free model quantization were investigated. The study demonstrated that while model inversion methods can successfully recover images from simpler datasets such as MNIST, there were challenges when applied to more complex datasets like CIFAR10 due to high computational costs and poor image quality without large computational power available. The use of recovered training image mean and standard deviation, provided an alternative for generating calibration datasets for model quantization. This approach showed significant improvements over random noise datasets, allowing quantized models to maintain accuracy levels close to those achieved with original training data. However, the use of these recovered image statistics provided limited abilities in detecting model performance degradation, indicating a need for further improvement in the recovery methods of image statistics and evaluation metrics.

Future work could address the above limitations by investigating more efficient model inversion techniques, additional methods for image statistics recovery, and the use of better evaluation metrics for assessing quantized model performance. Expanding the study to include more complex datasets and a wider range of model architectures could also provide a better understanding of data-free model quantization. By improving both the inversion methods and evaluation techniques, a more scalable and effective solution for data-free model quantization may be possible, in scenarios where access to original training data is not available.

Appendix A

Pseudo-code for model inversion and quantization

The methods outlined in chapter 2 to gather the results presented in chapter 3 were implemented using Python in VS Code, and used the packages random, torch, torchvision, numpy and matplotlib. The methods used are provided as pseudo-code below.

A.1 DeepInversion image generation

The original python code used to generate the results shown in section 3.1 was taken and modified from the GitHub repository [20] as shown in algorithm 1.

```
Data: Trained neural network model, target class label.
Result: Optimized image for the target class.
Initialise random noise image $x_{random}$. Define the total loss function:
 Define the total loss function:

    $$L = L_{\text{CE}} + \alpha_v L_{\text{v2}} + \alpha_f L_{\text{feat}} + \alpha_{l2} L_{\text{l2}}$$

 where:
    $$L_{\text{CE}} = -\sum_c y_c \log(p_c)$$
    $$L_{\text{v2}} = \sum_{i,j} \sqrt{(y_{i+1,j} - y_{i,j})^2 + (y_{i,j+1} - y_{i,j})^2 + \cdots}$$
    $$L_{\text{feat}} = \sum_l \left(|\mu_l - \mu_l^{BN}|_2 + |\sigma_l - \sigma_l^{BN}|_2\right)$$
    $$L_{\text{l2}} = \frac{1}{C} \sum_c \|x_c\|_2$$

 for number of iterations do
     Forward pass the random noise image $x_{random}$ through the model;
     Compute the total loss $L$;
     Backpropagate the loss to adjust the pixel values of $x_{random}$;
     Update the image $x_{random}$ via Adam optimization algorithm.
end
return Final optimized image $x_{optimized}$
```

Algorithm 1: DeepInversion image generation algorithm

A.2 Recovering training image statistics

The method to generate the results shown in section 3.2 is shown in algorithm 2.

```
**Data:** Trained neural network model, target class label.
**Result:** Recovered mean $\mu_r$ and standard deviation $\sigma_r$ of training data.
Initialise random noise image $x_{random}$. Define the loss function:

    $$L_{\text{stat}} = \text{MSE}(\mu_r, \mu_{\text{BN}}) + \text{MSE}(\sigma_{\text{r}}, \sigma_{\text{BN}})$$

for *number of iterations* do
    Forward pass the random noise image $x_{random}$ through the model until
      it reaches first batch normalization layer.;
    Compute the loss $L_{\text{stat}}$;
    Backpropagate the loss to adjust the pixel values of $x_{random}$;
    Update the image $x_{random}$ via Adam optimization algorithm.
end
Calculate new mean and standard deviation of $x_{random}$;
return *Recovered mean $\mu_r$ and standard deviation $\sigma_r$ from optimized
  noise image*
```

Algorithm 2: Recovering training data statistics

A.3 Model quantization using recovered statistics

The models which algorithm 2 was applied to were taken from the GitHub repository [21] and were then quantized to generate the results shown in section 3.3 as shown in algorithm 3.

```
Data: Trained neural network model, recovered statistics $\mu_r$ and $\sigma_r$,
      original dataset.
Result: Models quantized using different calibration datasets.
Generate synthetic calibration data sets $N(\mu_r, \sigma_r)$, $U(0,1)$ and $U(-1,1)$ ;
Define the loss function: Define the total loss function:

        $L_{\text{stat}} = \text{MSE}(\mu_r, \mu_{\text{BN}}) + \text{MSE}(\sigma_{\text{r}}, \sigma_{\text{BN}})$

for each model (Resnet32, VGG13, MobilenetV2) do
    for each calibration dataset (original training data, $N(\mu_r, \sigma_r)$, $U(0,1)$
      and $U(-1,1)$) do
        Quantize model weights and activations using calibration dataset;
        Evaluate model accuracy using original test dataset;
        record accuracy of quantized model;
    end
end
```

Algorithm 3: Model quantization.

A.4 Determining model accuracy post quantization from model outputs

To calculate the cosine similarity of model outputs shown in section 3.4 algorithm 4 was implemented.

```
Data: Original model, quantized model, datasets of $N(\mu_r, \sigma_r)$, $U(0,1)$
      and original test data.
Result: Cosine similarity between original and quantized model outputs.
for each model (Resnet32, VGG13, MobileNetV2) do
    for each dataset ($N(\mu_r, \sigma_r)$, $U(0,1)$ and original test data. ) do
        for each image in dataset do
            Pass image through original model and record the output $O_{orig}$;
            Pass image through quantized model and record the output
             $O_{quant}$;
            Compute cosine similarity between $O_{orig}$ and $O_{quant}$ using:

                $\text{cosine\_sim}(O_{orig}, O_{quant}) = \frac{O_{orig} \cdot O_{quant}}{\|O_{orig}\| \|O_{quant}\|}$

        end
        Compute average cosine similarity for the dataset;
    end
end
```
Algorithm 4: Determining model accuracy post quantization from model outputs using cosine similarity.

A.5 Determining model accuracy post quantization from model activations

To calculate the mean squared error (MSE), of model outputs shown in section 3.5 algorithm 5 was implemented.

```
Data: Original model, quantized model, datasets of $N(\mu_r, \sigma_r)$, $U(0,1)$
      and original test data.
Result: MSE between original and quantized model outputs.
for each model (Resnet32, VGG13, MobileNetV2) do
    for each dataset ($N(\mu_r, \sigma_r)$, $U(0,1)$ and original test data. ) do
        for each layer in the model do
            for each image in dataset do
                Pass image through original model and record activation for
                  the layer $A_{orig}$;
                Pass image through quantized model and record activation
                  for the layer $A_{quant}$;
                Compute MSE between $A_{orig}$ and $A_{quant}$:
                    $$\text{MSE}(A_{orig}, A_{quant}) = \frac{1}{n}\sum_{i=1}^{n}(A_{orig}^{(i)} - A_{quant}^{(i)})^2$$
            end
            Compute average MSE for each layer;
        end
    end
end
```
Algorithm 5: Determining model accuracy post quantization from model activations using MSE.

Bibliography

[1] Yann LeCun, John S. Denker, and Sara A. Solla. "Optimal Brain Damage". In: (1989). URL: `https://api.semanticscholar.org/CorpusID:7785881`.

[2] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: `10.1162/neco.1989.1.4.541`.

[3] Song Han et al. "EIE: efficient inference engine on compressed deep neural network". In: ISCA '16 44.3 (2016), pp. 243–254. ISSN: 0163-5964. DOI: `10.1145/3007787.3001163`. URL: `https://doi.org/10.1145/3007787.3001163`.

[4] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. "Fixed Point Quantization of Deep Convolutional Networks". In: Proceedings of Machine Learning Research 48 (20–22 Jun 2016). Ed. by Maria Florina Balcan and Kilian Q. Weinberger, pp. 2849–2858. URL: `https://proceedings.mlr.press/v48/linb16.html`.

[5] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: (June 2018). DOI: `10.1109/cvpr.2018.00286`. URL: `http://dx.doi.org/10.1109/CVPR.2018.00286`.

[6] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures". In: (2015). DOI: `10.1145/2810103.2813677`. URL: `https://doi.org/10.1145/2810103.2813677`.

[7] Ngoc-Bao Nguyen et al. “Re-Thinking Model Inversion Attacks Against Deep Neural Networks”. In: (June 2023). DOI: 10.1109/cvpr52729.2023.01572. URL: http://dx.doi.org/10.1109/CVPR52729.2023.01572.

[8] Si Chen et al. “Knowledge-Enriched Distributional Model Inversion Attacks”. In: (Oct. 2021). DOI: 10.1109/iccv48922.2021.01587. URL: http://dx.doi.org/10.1109/ICCV48922.2021.01587.

[9] Hongxu Yin et al. “Dreaming to Distill: Data-Free Knowledge Transfer via DeepInversion”. In: (June 2020). DOI: 10.1109/cvpr42600.2020.00874. URL: http://dx.doi.org/10.1109/cvpr42600.2020.00874.

[10] Yaohui Cai et al. “ZeroQ: A Novel Zero Shot Quantization Framework”. In: (June 2020). DOI: 10.1109/cvpr42600.2020.01318. URL: http://dx.doi.org/10.1109/cvpr42600.2020.01318.

[11] ANN H. Reynolds. *Convolutional Nerual Networks (CNNs)*. 2019. URL: https://anhreynolds.com/blogs/cnn.html.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[13] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: (2015). arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[14] MultiMedia LLC. *Hessian Eigenvalues for Asymmetric Quantization*. Ed. by Medium.com. 2024. URL: https://mikkicon.medium.com/hessian-eigenvalues-for-asymmetric-quantization-f4aa1a41dcf4.

[15] Markus Nagel et al. “A White Paper on Neural Network Quantization”. In: (2021). arXiv: 2106.08295 [cs.LG]. URL: https://arxiv.org/abs/2106.08295.

[16] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for interpreting and understanding deep neural networks”. In: *Digital Signal Processing* 73 (2018), pp. 1–15. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2017.10.011. URL: https://www.sciencedirect.com/science/article/pii/S1051200417302385.

[17] Wenchong He et al. "A Survey on Uncertainty Quantification Methods for Deep Learning". In: (2024). arXiv: 2302.13425 [cs.LG]. URL: https://arxiv.org/abs/2302.13425.

[18] Hai Shu and Hongtu Zhu. "Sensitivity analysis of deep neural networks". English (US). In: *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*. 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019. Publisher Copyright: © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org).; 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Annual Conference on Innovative Applications of Artificial Intelligence, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019 ; Conference date: 27-01-2019 Through 01-02-2019. AAAI press, 2019, pp. 4943–4950.

[19] Max Klabunde et al. *Similarity of Neural Network Models: A Survey of Functional and Representational Measures*. 2024. arXiv: 2305.06329 [cs.LG]. URL: https://arxiv.org/abs/2305.06329.

[20] NVlabs. *DeepInversion: Inverting deep neural networks with deep feature losses*. https://github.com/NVlabs/DeepInversion. Accessed: 2024-09-30. 2020.

[21] Yaofo Chen. *PyTorch CIFAR Models*. https://github.com/chenyaofo/pytorch-cifar-models. Accessed: 2024-09-30. 2020.