IMPERIAL

The use of non-standard data representations in PolyLUT inference

Author
Bardia Mohammadzadeh
CID: 01855967

Supervised by
Professor George A Constantinides

Second Marker
Professor P.Y.K. Cheung

A Thesis submitted in fulfillment of requirements for the degree of
Master of Engineering in Electronic and Information Engineering

Department of Electrical and Electronic Engineering
Imperial College London
2024

Abstract

Neural networks have become integral to numerous computational tasks, yet their substantial resource demands pose significant challenges. Quantisation techniques have shown promise in reducing the computational load during neural network inference. However, the efficiency of these techniques is often constrained by the necessity for uniform quantisation, in order to facilitate efficient arithmetic particularly when deployed onto GPUs. PolyLUT, a novel lookup table (LUT)-based neural network deployment strategy, offers a solution by encapsulating neurons into input-output bit mappings, thereby mitigating the costs associated with non-standard data-type conversions and arithmetic. This project investigates the application of non-standard data representations within PolyLUT, focusing on parameterised non-uniform quantisation schemes that enable data representation learning. The findings show that learned non-uniform quantisation can improve network performance such that test accuracy is higher than uniformly quantised networks with larger bit-widths. Therefore, networks can be deployed with lower bit-widths with no accuracy loss, resulting in significant resource reductions. In some cases resources were reduced by a factor of two with no accuracy loss.

Declaration of Originality

I hereby declare that the work presented in this thesis is my own unless otherwise stated. To the best of my knowledge the work is original and ideas developed in collaboration with others have been appropriately referenced.

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

I have used ChatGPT 4, as an aid in the preparation of my report. I have used it as a search engine to find information easier than googling. All information was then confirmed through reputable sources. All technical content and references comes from my original text.

Copyright Declaration

Acknowledgments

First and foremost, I would like to extend my sincere thanks and gratitude to my supervisor Professor George A Constantinides for taking on my self-proposed project with enthusiasm. Your guidance allowed me to explore a truly fascinating project on an important real-world topic. I am also grateful to Marta Andronic for her invaluable support alongside my supervisor, both of whom have been indispensable to the project's progress. This project allowed me to delve into a field that truly interests me, and I could not have done it without their incredible and generous support.

I wish to also convey my immense gratitude to the Electrical Engineering department at Imperial College London. The staff have consistently demonstrated compassion, care, and generosity, especially during challenging times. I can say with absolute confidence that the support provided by the department has totally changed the course of my university experience for the better. The welcoming environment and the incredible individuals working there have made my time at Imperial a hugely transformative experience. The confidence I have gained in tackling real-world engineering problems is a testament to the excellent education and support I received, and for this, I will always be thankful.

Lastly, I would like to express my heartfelt gratitude to my family for their unwavering support throughout my university journey. If not for their support, I may never have made it to this point. Without their constant encouragement and kindness, this achievement would not have been possible. I am especially thankful to my father, whose guidance has been the foundation of my life, and whose sacrifices have allowed me to put my education first. His never-ending support and belief in me have been instrumental in shaping me into the person I am today. His presence and advice have never failed me, and for that, I am eternally grateful.

Contents

List of Acronyms

PTQ post-training quantisation

QR quantisation with retraining

QAT quantisation aware training

LUT lookup Table

FPGA Field Programmable Gate Array

DNN Deep Neural Network

GPU Graphics Processing Unit

PDF probability density function

CDF cumulative distribution function

SGD stochastic gradient descent

STE straight-through estimator

OOP Object-oriented programming

List of Figures

List of Tables

1 Introduction

Contents

This report begins with a background on the necessary theory relating to quantisation, neural networks, and PolyLUT. Subsequent sections discuss the design and implementation of various quantisation schemes and the rationale behind their selection. The key development of this project, namely the learnable non-uniform quantisation scheme, is presented, analysed and evaluated in a separate chapter. Finally, the overall results are presented, discussed, and evaluated.

1.1 Motivation

The incorporation of Machine Learning in modern technology continues to grow every year. Neural networks in particular have been at the forefront of recent breakthroughs in AI. Both of these points are explored further in Section 2.1. However, a significant limitation of neural networks is the large computational load required to train and deploy them effectively, which can limit their accessibility and scalability for real-time applications and for organisations with limited resources. This high computational demand often results in considerable energy consumption, which not only increases operational costs but also raises environmental concerns. One method to mitigate the computational load of a neural network, is quantisation (explored in Section 2.2 and Section 2.3).

Quantisation in neural networks aims to reduce the computational load, whilst maintaining the

task performance of a high precision network. In other words, the goal of neural network quantisation is to reduce the number of bits required to represent the signals within the neural network, whilst maintaining the expressive capabilities of the network such that the task performance does not decrease. This poses a significant research challenge, and many researchers have proposed different techniques to implement quantisation on neural networks [1][1].

Quantisation schemes (Section 2.2.2) can largely be divided into uniform, and non-uniform quantisation. Whilst non-uniform quantisation can be used to obtain the minimum squared quantisation error via the Lloyd-Max algorithm [2][3], deploying such a quantisation scheme is often costly. This is because whilst uniform quantisers have standardised hardware, a non-uniform quantiser requires logic specific to the exact quantisation scheme being implemented. Therefore, if one wishes to quantise each layer's activation in a neural network with its own non-uniform quantiser, the cost may outweigh the benefit.

This however, is not the case for neural networks such as PolyLUT (Section 2.5), where the non-uniform quantisation can be absorbed into the lookup Table (LUT) mapping, thereby incurring no additional cost. Furthermore, LUT-based deployment strategies such as PolyLUT suffer from exponential scaling of resource usage as the bit-width of activations and inputs grows. Therefore, PolyLUT in particular benefits significantly from quantisation efforts.

As a result, the aim of this project is to explore the use of non-standard data representations in PolyLUT inference. Note that because the PolyLUT framework absorbs all of a neuron's operations into the LUT mapping, quantisation is only applied to neuron activations. Therefore, the scope of this project is limited to quantisation of neuron activations.

1.2 Approach

Non-uniform quantisation encompasses all quantisation algorithms where the quantisation intervals are not of constant size. This is a broad category, and as a result there are many different quantisation schemes that could be explored. In this project, a few schemes were specifically chosen, and then a general framework was developed to implement an arbitrary non-uniform quantisation per layer or per neuron. This is explored further in the latter chapters of this report.

The quantisation schemes were developed to be compatible with the PyTorch framework [4]. To that end, custom software classes were created which emulate quantisation by changing floating-

[1] In this project, quantisation in neural networks is limited to scalar quantisation.

point data to specific fixed values - in other words, the corresponding quantisation levels. This allowed for network training and inference to be done on CUDA [5] devices.

When using the aforementioned non-uniform quantisation schemes, various approaches may be considered. These are categorised into the following: quantisation aware training (QAT), post-training quantisation (PTQ), and QR (Sections 2.3.1, 2.3.2 and 2.3.3 respectively).

Quantisation error (Section 2.2.1), can be analysed with various distances. These include Mean Squared Error (MSE) or Mean Absolute Error. However, the most relevant metric is the accuracy of the network, or the task loss. This is because whilst some quantisation schemes may have a high quantisation error, that does not always result in worse task performance. Therefore, in this project the *best quantisation scheme* is considered to be the one which yields the best network task performance, measured by the accuracy achieved on the test dataset.

1.3 Aims and Objectives

The objective of this project is to improve the predictive capabilities of low-precision neural networks by reducing the negative impact of quantisation when applied to layer activations.

In current neural network implementations, quantisation is implemented as a means to reduce the cost of deploying the network onto hardware (as explained in Section 2.3). Whilst various quantisation schemes are used, a standard implementation is uniform quantisation (described in Section 2.2.2).

The hypothesis driving this project is that uniform quantisation may not always be the ideal scheme to apply to a layer's activations. On the contrary, the quantisation scheme that results in the best network performance is likely to depend on the distribution of the activations themselves.

To investigate this hypothesis, this project aims to conduct a quantitative exploration and analysis of the effects of various low-precision quantisation schemes on neural networks. This analysis is then intended to be used to develop an automated framework which can implement and optimise low-precision quantisation schemes during training, such that the quantisation scheme can be learned to improve task performance.

The key deliverable of this project is a methodology compatible with a modern machine learning library such as PyTorch [4], which applies quantisation and optimises the quantisation scheme during network training. The scope of this project is limited to feed-forward neural networks. To be considered successful, the methodology should, when implemented, demonstrate a consistent improvement to network task performance.

Additionally, this enhancement should be achieved without significantly increasing resource usage. This criterion ensures that the methodology not only enhances task performance but also maintains efficiency in terms of computational resources. Achieving this balance is important for the practical application and effectiveness in real-world scenarios.

2

Background

Contents

2.1 Machine learning and Feed-forward Neural networks

Machine learning is a field of artificial intelligence that encompasses the development of algorithms which allow computer-based systems to perform tasks without specifically being programmed. Machine learning algorithms revolve around the concept of learning from data. This contrasts traditional programming where instructions are explicitly written by a designer.

Feed-forward Neural networks are machine learning models that employ a network of connected nodes, called *neurons* organised into layers, as shown in Figure 3.1. How neural networks are placed within the artificial intelligence field is illustrated in Figure 2.1.

Neural networks have been shown to achieve good performance when compared to traditional algorithms on several tasks including Image Classification [6], Natural Language Processing [7] and Generative Models [8]. Neural networks with as little as one hidden layer can be shown to be universal approximators [9], roughly meaning that they can approximate any function to an arbitrary level of accuracy (given that the layers are sufficiently wide). Neural networks generally have better accuracy when layers are wider [10] (meaning layers have more neurons), and the network is deeper [11] (meaning there are more layers in total).

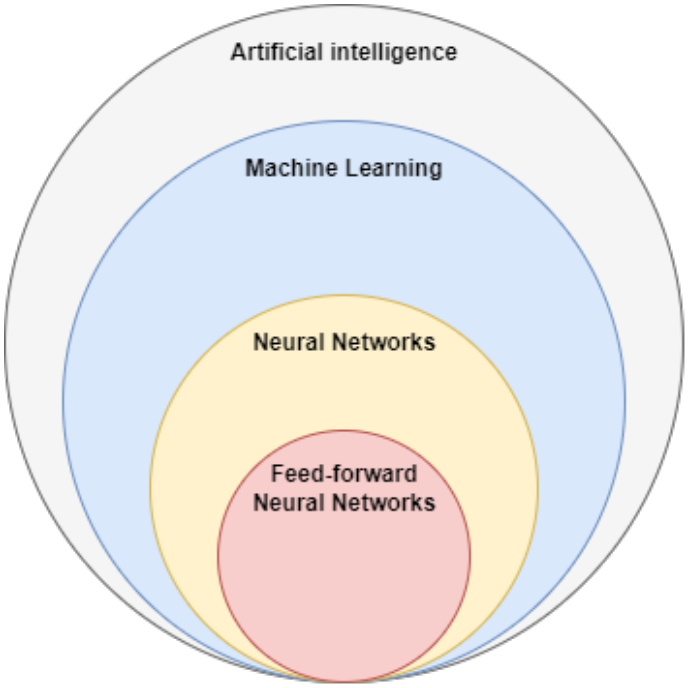


Figure 2.1: Neural networks within the Artificial Intelligence hierarchy

2.1.1 Neurons

As mentioned previously, neural networks employ a series of connected nodes called neurons. Neurons, depicted in Figure 2.2, are defined by their weights, bias and activation function. Each neuron can be connected to several inputs and will have a single output. Each of these inputs will have a corresponding weight parameter.

Inputs fed into the neuron are multiplied by their corresponding weight parameter and then

summed along with an additional parameter called the bias. The resulting value is then fed into a nonlinear activation function.

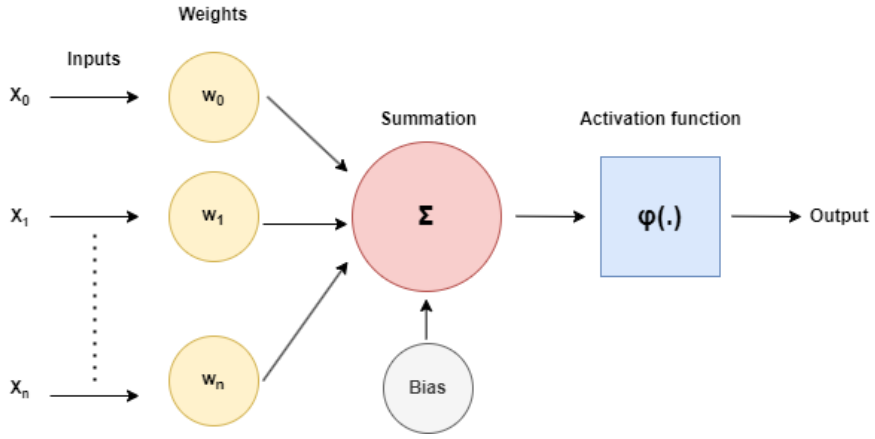


Figure 2.2: A standard neural network neuron

2.2 Basic Quantisation Theory

Quantisation can be described as the mapping of a set of continuous real-valued numbers across a finite discrete set of numbers. In this report, this finite discrete set is sometimes called the *representable range*. Quantisation was essential to the evolution of digital electronics and has been studied for more than half a century. Gray and Neuhoff offer a comprehensive overview of the history of quantisation and present the information clearly in their 1998 survey [12].

The goal of a quantisation algorithm can be viewed as firstly minimising the difference between the output of a computation using these discrete values, and the output of the same computation but using the continuous real values. And secondly to do so whilst also minimising the number of bits used to express the set of discrete values.

Quantisation is always necessary for computer arithmetic to work since the number of bits for a given data type must in practice be finite. However, as data types have more bit-width, the effects of quantisation are less disruptive. For example, the precision of 32 or 64-bit floats is such that in most use cases, the quantised value represented by the float is often considered to be negligibly close to the real value it is approximating.

However, in some use cases, bit-widths are far more limited. For example, 8-bit data types are commonly used in embedded systems applications due to limitations in compute and memory. In these cases, the difference between the real continuous set and the discrete representable set can

be significant. As a result, at smaller bit-widths, it is important to consider the data type used, which in turn can decide the quantisation scheme used.

2.2.1 Quantisation error

Quantisation error is the difference between the original real-valued number and the result after quantisation. Assuming the quantisation scheme does not allow for overflows, quantisation error can be divided into 2 sources - error due to clipping and error due to rounding - these are depicted in Figure 2.3. The term clipping refers to the bounding of a value between the maximum and minimum values of a representable range. Therefore, clipping occurs when a value is beyond the representable range of the quantisation scheme. When this is the case, the quantised output is chosen to be the closest bound of the representable range.

In some quantisation schemes such as quantisation to floating-point representation, when values beyond the upper limit of the representable range are encountered an error is thrown to let the user know that the value is beyond this range. This is because whilst error due to rounding is bounded, error due to clipping is unbounded. Therefore, in some applications, it is arguable that throwing an error is more suitable than clipping.

Rounding occurs when the input to quantisation lies between levels in the representable range. Via a rounding algorithm, the value is quantised to one of the two nearest quantisation levels. The specific rounding algorithm used can vary, but a common implementation is round to nearest-even. This is where the value is rounded to the nearest quantisation level and if it is exactly halfway between the two levels, it is quantised to the *even* level. In this scheme, the term *even* refers to the binary representation of the levels, therefore the level is *even* if the least-significant bit is zero.

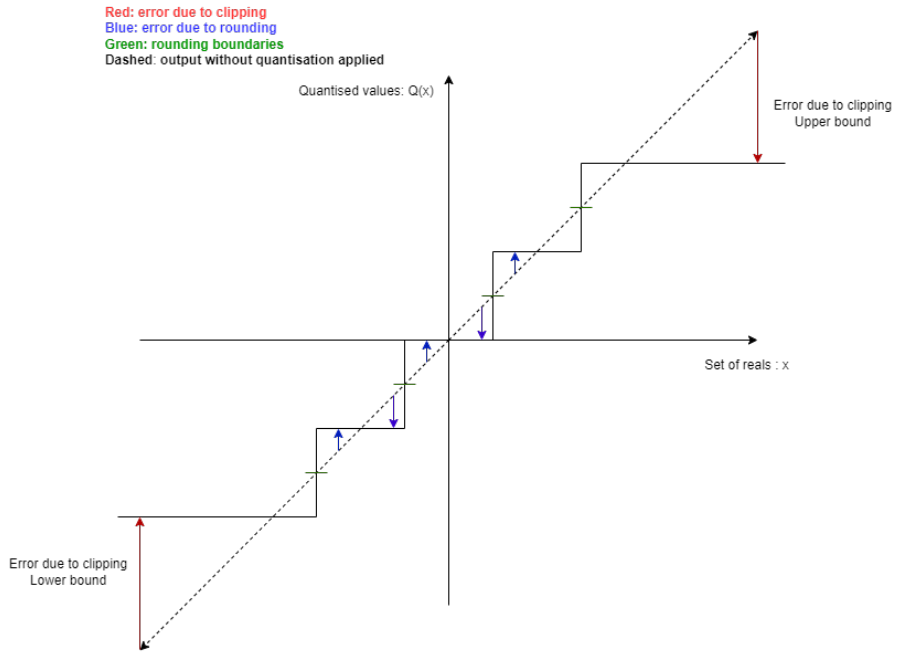


Figure 2.3: Error due to clipping and error due to rounding when uniform quantisation is applied

2.2.2 Quantisation schemes

A quantisation scheme is an algorithm that is followed to apply quantisation. A very common scheme is fixed-point quantisation otherwise known as uniform quantisation[1]. In this scheme, a real-valued range is divided into intervals of constant width. This can be seen in Figure 2.4a. Fixed-point quantisation can be extended with the idea of a bias[2] and a scale. When using a scale and a bias, sometimes referred to as a zero point, the following formula is used to quantise an input:

$$Q(R) = \text{round}\left(\text{clip}_B\left(\frac{R - Z}{S}\right)\right) \tag{2.1}$$

Where R represents the real and continuous input, Z represents the bias or zero point and S represents the scale. Furthermore, the $\texttt{clip}(x; B)$ is defined as follows for signed quantisation:

$$\texttt{clip}(x; B) = \begin{cases} x & \text{if } |x| < B, \\ \text{sgn}(x) \times B & \text{otherwise.} \end{cases}$$

The boundary, B is equal to the maximum representable value of a signed integer for the number of bits used in representation of the quantised output. By changing the zero-point, one

[1] In this report, the terms fixed-point quantisation and uniform quantisation are used interchangeably.
[2] Note that here bias refers to a completely different concept than when used in the context of a neuron's output.

can obtain asymmetrical clipping.

In unsigned uniform quantisation, clipping is defined slightly differently:

$$\texttt{clip}(x; B) = \begin{cases} 0 & \text{if } x \leq 0, \\ x & \text{if } 0 < x < B, \\ B & \text{otherwise.} \end{cases}$$

In this case, B is equal to the maximum representable unsigned integer for the number of bits used. This results in the following equation being used to return a value from its quantised representation:

$$\text{Output} = Y \times S + Z \tag{2.2}$$

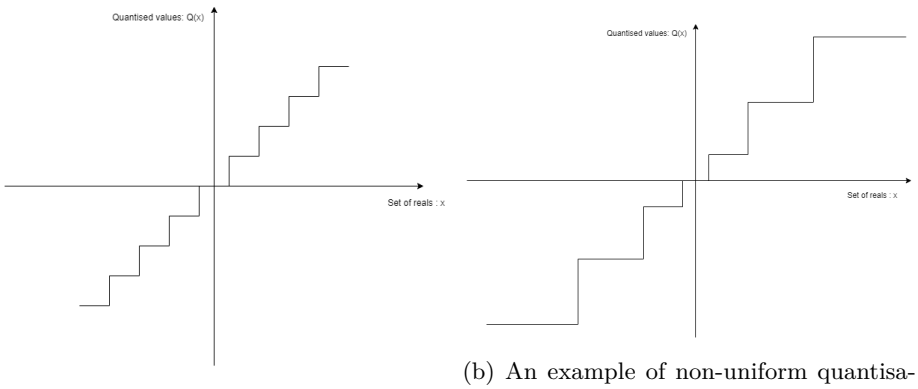
where Y refers to the integer value of the bit-string used to store the quantised representation.

The scale, S, allows for the size of the quantisation interval to be changed, and the zero point, Z, allows for shifting of the representable range. However, fixed-point quantisation is limited by the fact that all the quantisation intervals are the same size. This means that across a large range, whilst the absolute error due to rounding is constant, the relative error changes. Relative error is defined as the ratio of error from rounding to the number being rounded (the input to the quantiser). This is also known as the signal to quantisation noise ratio.

On the other hand, floating-point quantisation has varying absolute error, but constant relative error. This is because the quantisation intervals are not constant - they increase with the magnitude of the quantisation level. Floating point quantisation is an example of non-uniform quantisation, which is depicted in Figure 2.4b.

2.2.3 Companded Quantisation

Companded quantisation is a form of non-uniform quantisation implemented using a transform, a uniform quantiser, and the transform's inverse. Whilst some non-uniform quantisers are implemented by specifically choosing the quantisation levels, companded quantisation works by transforming the input and then passing the result through a uniform quantiser. The output of the



(a) Uniform quantisation

(b) An example of non-uniform quantisation

Figure 2.4: Depiction of uniform and non-uniform quantisation

uniform quantiser is then passed through the transform's inverse which obtains the final quantised result. Figure 2.5 illustrates the process (note that this figure considers the positive domain only - negative values are clipped to zero). Companded quantisation has been shown to be able to represent any non-uniform quantiser, given the input is bounded [3]. Further examples and mathematical reasoning about companded quantisation can be found in [3].

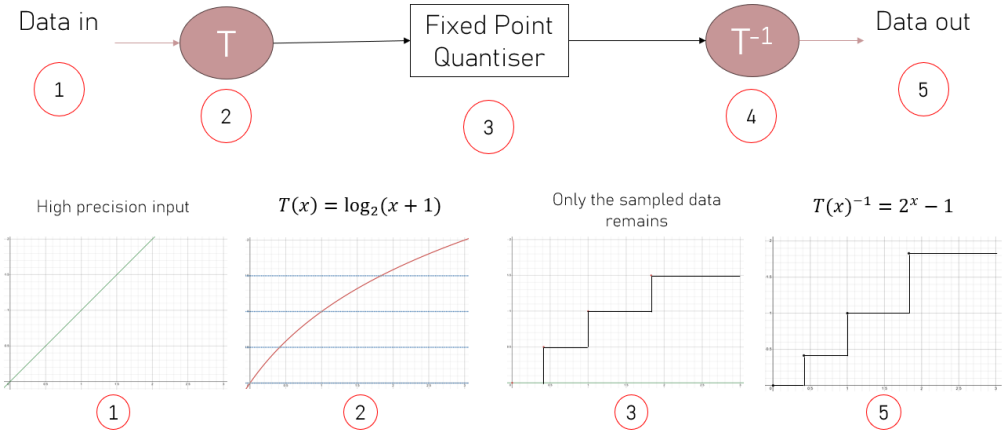


Figure 2.5: Implementing a non-uniform unsigned quantisation scheme through companding techniques.

2.2.4 Fixed-point quantisation with Brevitas

In this project, fixed-point quantisation was implemented using the Brevitas python package by Xilinx [13]. The Brevitas quantiser converts the floating-point outputs of the neural network node into a fixed-point representation along with a scale and bias value as explained in Section 2.2.2. The scale can be determined from statistics collected about the input to the quantiser at runtime. These can include things such as the max, the range or the absolute range. Otherwise, the scale

can be a learnable parameter which is updated via back-propagation and initialised using these runtime statistics. In this report, when using runtime statistics, the scale is calculated via the following formula:

$$\text{Scale} = \frac{|\text{Max}(D) - \text{Min}(D)|}{2^{\beta}} \tag{2.3}$$

Where D is the data being input to the quantiser and β is the number of bits in the quantised representation. Note that Scale was denoted by S in Equations 2.1 and 2.2.

To enable the scale to be a learnable parameter, the quantisation function must be continuous. However, fixed-point quantisation in the form described in Equation 2.1 does not have a continuous gradient because the `round` and `clip` operations are not continuous. As a result, the gradients of these operations must be approximated with a straight-through estimator (STE) [14], which means that the partial derivatives for these functions are equal to 1.

$$\frac{\partial \text{ round}(x)}{\partial x} = \frac{\partial \text{ clip(x)}}{\partial x} := 1$$

By doing this, the derivative of the scale (and the bias) with respect to the loss are computable, which means that both these parameters can be learned via back-propagation.

2.3 Deep Neural Network Approximation

Although Deep Neural Networks (DNNs) can provide good performance on various tasks, they have high computational costs. Examples of successful neural networks such as AlexNet [6] required over 60M weights and over 700M multiply-accumulate instructions. As a result, there has been considerable effort to reduce this computational cost. Architectures such as MobileNets [15] were able to achieve good accuracy with far less cost via various optimisations. As a result, MobileNets was significantly more feasible for deployment on low-compute devices such as mobile phones.

Quantisation is a form of neural network approximation that has been shown to yield a reduction in resource usage and an increase in throughput [1][16]. This is because quantisation reduces computational cost in multiple ways. Firstly, smaller data widths result in lower interconnect bandwidth requirements when moving data and lower memory requirements when storing data.

Bandwidth and memory are integral to neural network performance [17] because of the constant movements of weights and gradients to and from storage. Secondly, smaller data widths also enable the use of smaller arithmetic units resulting in lower latency and resource cost. This is because, with smaller arithmetic units, one can pack more units into the same area, resulting in higher parallelism and thus higher throughput.

In the PolyLUT case, although no arithmetic units are required, quantisation reduces the resource cost massively since smaller bit-widths result in smaller LUTs. In general, quantisation on neural networks can roughly be categorised into the following three categories.

2.3.1 Quantisation-aware Training

QAT is a technique used to improve the performance of quantised neural networks by simulating the effects of quantisation during the training process. In QAT, the forward pass of the network includes quantisation steps, which means the signals, such as weights and activations, are quantised as if the network were running in inference mode. This allows the network to learn how to compensate for the errors introduced by quantisation.

During QAT, the gradients are computed using the unquantised weights. This process is often facilitated by the use of STEs, which approximate the gradients of the uniform quantisation function by setting them to one (as explained in Section 2.2.4). By including quantisation effects during training, QAT helps in minimising the loss in accuracy that typically occurs when a network is quantised after being trained with full precision weights.

QAT is particularly useful for low bit-width quantisation where the quantisation errors are larger. It has been shown to significantly improve the accuracy of the quantised model compared to models that are quantised post-training [18]. This trend is also seen in this project's experiments (Section 5.6). The technique is widely used in scenarios where maintaining high accuracy is crucial, such as in mobile and embedded applications where computational resources are limited [1].

2.3.2 Post-Training Quantisation

PTQ is a method where a pre-trained neural network model is quantised after the training process is complete. This approach does not require modifying the training procedure and can be applied to any trained model. PTQ typically involves converting the weights and activations from floating-point to a lower bit-width representation, such as 8-bit integers.

The process of PTQ may include calibration using a small subset of the training data to determine the optimal scaling factors for the weights and activations. These scaling factors are important as they ensure that the dynamic range of the data is appropriately captured within the limited precision of the quantised format. The calibration step helps in reducing the quantisation error and improving the overall performance of the quantised model [19]. In this report's PTQ experiments (Section 5.6), the calibration of the scale is implicitly done via the Brevitas Runtime statistics scaling implementation (Section 2.2.4).

PTQ often performs worse than QAT, and may lead to a significant drop in accuracy, especially when aggressive quantisation (quantisation to very low bit-widths) is applied. Despite this, PTQ remains a popular choice due to its simplicity and ease of implementation [20]. PTQ has the added benefit of being financially cost-effective, since no retraining or fine-tuning is required to quantise a model which is important in large networks with large datasets such as LLMs where retraining can be very expensive.

2.3.3 Quantisation with Retraining

QR is a technique that combines elements of both QAT and PTQ. In this regime, a model is first trained using high precision. After the initial training, the model undergoes quantisation to reduce the bit-width of the weights and activations. Subsequently, the quantised model is fine-tuned using a few additional epochs of training. This has been shown to yield improvements in network task performance [21].

The fine-tuning phase aims to recover any accuracy lost during the post-training quantisation step. By exposing the model to training data and allowing the weights to adjust within the quantised constraints, the model can learn to mitigate the quantisation errors. However, it is more expensive than PTQ, since as mentioned previously, for large models retraining can be expensive.

This method may perform better than QAT, where quantisation is applied from the start of training, since by initially training with high precision, the network parameters (the weights) are able to learn with less quantisation noise.

2.4 DNN Acceleration on FPGAs

When examining the hardware options for accelerating neural networks, each type presents unique advantages and disadvantages. Currently, Graphics Processing Units (GPUs) are the predominant choice due to their high bandwidth interconnect and extensive parallel arithmetic processing capabilities, both essential for the computational demands of neural networks [22].

Despite their popularity, GPUs have notable drawbacks. They excel with dense inputs but struggle with sparse inputs, making them less effective for inference in sparsified neural networks achieved through methods like weight pruning. Additionally, GPUs are optimised for floating-point arithmetic and lack support for custom data types, which are often utilised in quantised DNNs. Consequently, GPUs are less efficient in terms of power usage when compared to specialised accelerators for approximated neural networks [16].

The rapid evolution of DNN algorithms further complicates the use of domain-specific ASICs. ASICs, while highly efficient for specific tasks, may become obsolete quickly if their architecture does not adapt to new state-of-the-art DNNs. Different tasks might also require varying architectures, making the fabrication of ASICs potentially cost-ineffective. These challenges highlight the advantages of the re-programmable nature of Field Programmable Gate Arrays (FPGAs) [23]. FPGAs offer the flexibility to adapt neural network designs to new architectures without incurring additional material costs.

FPGAs can be programmed to perform specific operations with greater efficiency, resulting in improved latency and throughput, and more significantly, higher energy efficiency [16][24][23]. Their customisability supports fine-grained parallelism, allowing concurrent execution of multiple DNN layers or operations. This type of parallelism accommodates irregular network architectures and enables optimisation tailored to specific task requirements.

In conclusion, FPGAs offer significant advantages in terms of flexibility, energy efficiency, and customisability. These attributes make FPGAs a compelling alternative, especially as DNN algorithms continue to evolve and diversify.

2.5 PolyLUT

PolyLUT [25] is an FPGA LUT-based neural network deployment strategy building upon similar work done in LUTnet and LogicNets [26][27], whereby the neurons are deployed onto hardware as input-output mappings. During training, the network is quantised to very low precision (less than 8 bits). Once trained, the input space of each neuron is enumerated, and every input possible is passed through the neuron to record the corresponding output, thereby obtaining the input-output mapping. Since the size of the input space grows exponentially with bit-width and neuron fan-in[3], this is not feasible at large bit-widths. For the same reason, the process of enumeration and tabulation of outputs is computationally expensive and time-consuming. The enumeration process is depicted in Figure 2.7.

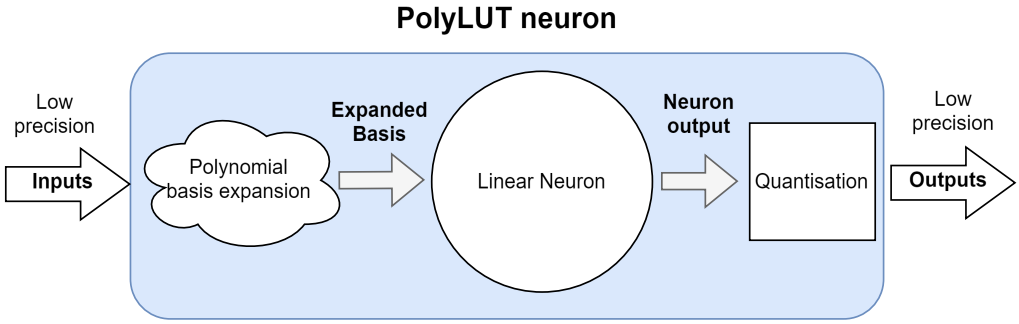


Figure 2.6: A diagram of the basic components within a PolyLUT neuron. Low precision inputs are basis expanded to include polynomial terms, and then are fed into a standard neuron. The blue region represents the functions that will be replaced by an input-output mapping when deployed onto hardware.

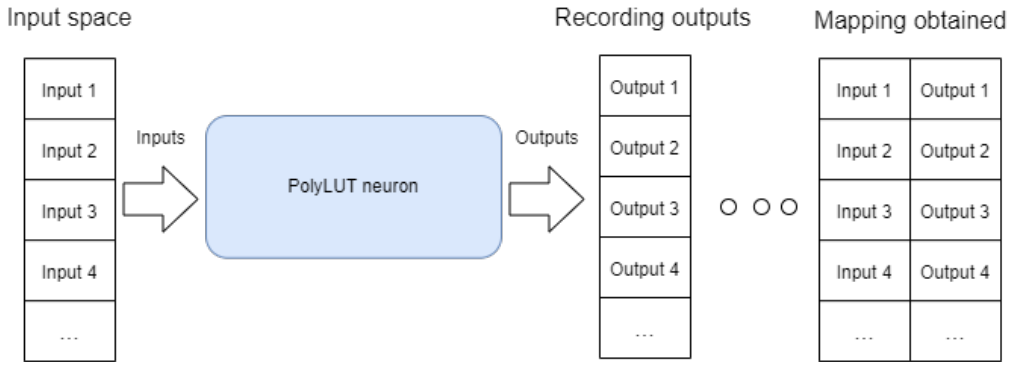


Figure 2.7: An input-output mapping is obtained from a trained neuron.

The benefit of deploying neurons as input-output mappings is that there is no longer any arithmetic required for the neuron to process an input, eliminating the need for DSPs—an invaluable FPGA resource. Consequently, this preserves DSP resources for other tasks.

[3]Fan-in describes the number of inputs a neuron is connected to.

Furthermore, the neurons can include a polynomial expansion of the input without any additional cost since this expansion is absorbed into the input-output mapping. This allows for neurons to learn more complex relationships and hence the network can achieve good accuracy with fewer layers [25]. PolyLUT neurons incorporate polynomial basis expansion as shown in Figure 2.6. This basis expansion is controlled by the *degree* hyper-parameter which dictates the order of the polynomial expansion. For example, a network with degree two expands the basis to include quadratic terms.

Once the input-output mappings have been obtained, they are deployed onto the FPGA hardware. The synthesis tool can make some optimisations since from this point onwards, neurons are effectively large truth tables. The benefit of deploying onto an FPGA is that the FPGA's physical LUTs can efficiently instantiate this input-output mapping with fast access speeds, allowing for ultra-low-latency inference.

In summary, PolyLUT offers an alternative solution for deploying neural networks with the benefit of ultra-low-latency inference, making it suitable for real-time applications. This strategy is particularly effective for small networks due to its exponential scaling, which limits its use for networks requiring large bit-widths. Furthermore, although not explored in the original paper, PolyLUT-style neurons can potentially be integrated with conventional designs when required, to reduce latency or in cases where arithmetic units are not easily available. PolyLUT and other LUT-based methodologies leverage the fundamental building blocks of FPGAs to achieve unique advantages, resulting in competitive machine learning performance.

2.5.1 Jet tagging dataset

Jet tagging involves distinguishing between jets originating from different types of particles emitted in the high-energy experiments conducted at the Large Hadron Collider (LHC). The development of machine learning techniques has significantly enhanced the ability to analyse jet substructure and improve jet tagging accuracy.

The Jet Tagging Substructure Dataset requires low latency inference for several reasons. In high-energy physics experiments, events occur at an extremely rapid pace. Low latency inference allows for real-time processing of data, enabling timely decisions about which events to record for further analysis. This allows for capturing the most relevant data without overwhelming the storage and processing capabilities. PolyLUT style deployment is ideal for classifiers targeting the

jet tagging task, due to its ultra-low-latency inference capabilities. As a result, it is the dataset used in this project.

2.6 Related works

Overall, there are various low-precision data types that networks commonly quantise to, in order to reduce cost whilst minimising accuracy loss. They include but are not limited to, binary representation [28], fixed-point representation, logarithmic representation [29] and various mini-float data types. However, choosing the *best* data representation to use is a difficult task, and additionally, there may not be a single data representation scheme that is best for every part of the network.

Methods such as Jung et al. [30] implement quantisation-aware training and attempt to avoid the issue described above by minimising the task loss with respect to the parameters of the quantisation scheme. This yielded improvements in test accuracy for low-precision networks. Whilst the authors were able to implement learned non-uniform quantisation schemes, this was limited to exponential distributions.

The LQ-Nets method [31] introduces a parameterised quantisation function that allows optimisation of quantisation levels and intervals during training. More specifically, the authors are able to train a *floating-point basis* which defines the quantised subspace. LQ-Nets uses the STE to allow gradient-based optimisation of the learnable basis. This approach allows values to be represented by linear combinations of learned discrete levels that are optimised alongside the network parameters.

The key benefit of this approach is that the authors are able to achieve non-uniform quantisation in a hardware-friendly manner. However, this advantage is not present in the PolyLUT setting, since (as mentioned previously) the quantisation function will be absorbed into the LUT mappings.

Finally, K.Yamamoto developed a learnable quantisation scheme through the use of parameterised companding functions [32]. In this work, the companding functions (Section 2.2.3) were learnable, meaning that they can be optimised against the task loss of the network. As a result, the quantisation scheme can be learned to improve task performance. Although this allows for learned non-uniform quantisation schemes, the scheme is constrained by the number of piecewise sections in the companding function. Furthermore, when deployed on hardware, the network requires specifically designed quantisers to implement the learned non-uniform quantisation, thereby adding additional cost. However, this disadvantage is not present in the PolyLUT setting.

2.7 PyTorch

PyTorch [4] is an open-source deep learning framework. It is widely used in the machine learning community for developing and training neural networks.

Unlike static computation graphs, PyTorch's dynamic graph allows for real-time construction and modification of neural networks. This makes debugging and experimentation more straightforward, which is beneficial for research applications where frequent changes and iterations are common. Additionally, PyTorch provides a comprehensive set of prebuilt layers and loss functions, simplifying the process of building neural networks. Users can define custom models by inheriting from the `nn.Module` class, which promotes code modularity and reusability.

Furthermore PyTorch also provides support for GPU acceleration through CUDA [5], which significantly reduces training and inference times.

In summary, PyTorch is a versatile framework for deep learning. Its design supports both academic research and practical applications in machine learning. PolyLUT (Section 2.5) is built using PyTorch, and hence so are the developments made in this project.

2.8 Object-Oriented Programming

Object-oriented programming (OOP) is a commonly used programming paradigm centred around objects and classes. The aim of the OOP paradigm is the creation of modular, reusable, and maintainable code. This section serves as a brief introduction to OOP principles. Notably, PyTorch leverages these principles to streamline the development and training of neural networks.

Encapsulation is one of the fundamental principles of OOP. It involves bundling data (attributes) and methods (functions) that operate on the data into a single entity called a class. In PyTorch, entire neural networks, layers, activation functions, optimisers and various other components are implemented as classes.

Other fundamentals include inheritance and polymorphism. Inheritance allows a child class to inherit attributes and methods from a parent class. Polymorphism enables methods to perform different operations based on the object they are acting upon, even if they share the same name. This provides flexibility and integration of various objects.

Another key concept is abstraction. Abstraction simplifies complex systems by using classes appropriate to the problem, working at the most relevant level of inheritance. In Python, abstract base classes define a blueprint for derived classes, ensuring they implement specific methods. In PyTorch, an abstract model class can define a forward method that must be implemented by any subclass, ensuring a consistent interface for all models.

As mentioned previously, PyTorch extensively utilises OOP principles to facilitate neural network development. The `nn.Module` class serves as a base for all neural network modules, encapsulating relevant methods and attributes within a single class. This approach promotes modularity and reusability. Inheritance extends the functionalities of existing models, enabling the creation of more complex architectures. Polymorphism allows different models to be used interchangeably within training or inference pipelines, enhancing flexibility. By adhering to these OOP principles, PyTorch ensures a structured and scalable approach to machine learning.

3

Project Setup

Contents

This chapter describes the experimental setup for this project. The setup remained the same for the results presented in Chapters 4 and 5, unless otherwise specified.

As mentioned previously, the neural network is implemented in Python using the PyTorch. During training, the network is quantised in the forward path. Since the end goal is to implement each node as an input-output mapping, quantisation is only required for the activation of each neuron meaning that gradients and weights remain unquantised. Note that input quantisation is only required for the input layer since all other layers have quantised inputs due to the output quantisation of the previous layer, as shown in Figure 3.1. Furthermore, when a bit-width is given for the input layer, this is the bit-width for both the input quantiser and output quantiser.

3.1 Datasets

The dataset used was the Jet Tagging Dataset [33](Section 2.5.1). This dataset was chosen since it allows for comparison against the original PolyLUT paper. The disadvantage of testing on only a single dataset is that there is a chance the results observed show trends which might not exist for

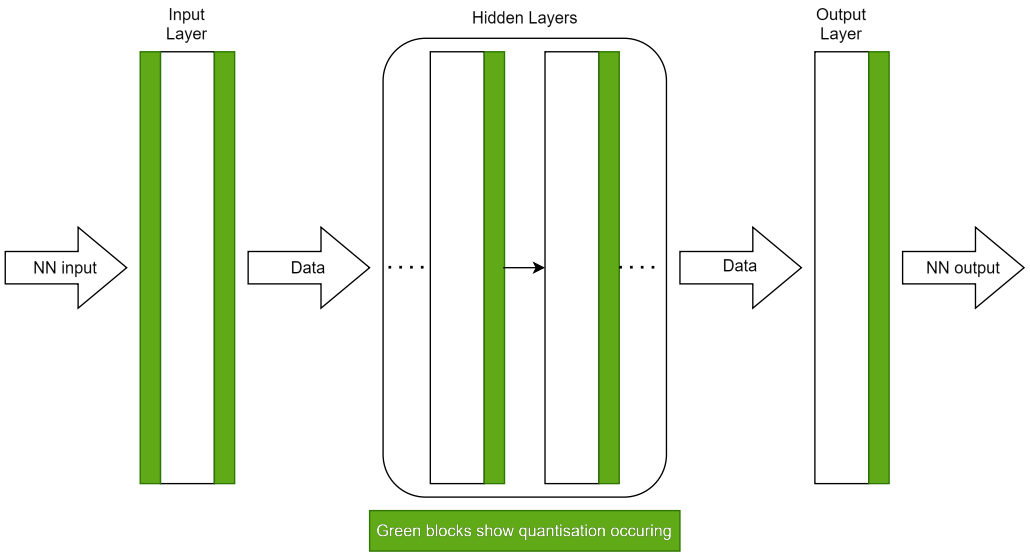


Figure 3.1: Neural network layers are quantised on input and output.

other machine learning tasks (involving different datasets). However, this disadvantage is mitigated when using a learnable quantisation scheme, since the scheme is by definition optimised against the task loss and therefore should be able to adapt to other tasks.

3.2 Implementing a quantisation scheme

To implement a custom quantisation scheme, a combination of a transformation and a fixed-point quantiser was used. For all experiments, the fixed-point quantiser was forced to have no bias, meaning that the zero-point was 0 and therefore every quantisation scheme had a quantisation level at zero.

Before being input to the fixed-point quantiser, the data was transformed via a function $T(x)$. Upon being outputted by the fixed-point quantiser, the data was passed through the inverse of the first transform: $T^{-1}(x)$. This means that to be a valid transform, $T(x)$ must have an inverse, although this inverse may only need to be defined in a specific domain (such positive values of x if the activation function is ReLU). As a result, this scheme is a form of companded quantisation as explained in Section 2.2.3.

The process is summarised in Figure 3.2 and occurs directly after the activation function is applied. However, if the activation function is ReLU [34], it can be applied at the same time as the fixed point quantisation.

In this report, $T(x)$ is referred to as the *forward transform*, and $T^{-1}(x)$ is referred to as the

inverse transform. Furthermore, all gradients and weights are unquantised - only activations in the forward path are quantised. This is because the weights are implicitly stored in the input-output mapping (due to PolyLUT). Therefore, when a bit-width is mentioned, this is the bit-width for the activations of a layer - all weights and gradients are at full precision (single precision floating point).

Figure 3.2: The structure of the quantiser used. The transform T and its corresponding inverse are changed to implement different quantisation schemes.

3.3 Ensuring PyTorch compatibility and autograd compatibility

The various quantisation schemes were implemented on software, and their performance was tested through running inference on GPUs. As mentioned previously, layer activations are quantised as they propagate through the forward path of the network. Both training and inference were facilitated through the use of PyTorch (Section 2.7).

To ensure PyTorch compatibility, quantisation schemes were implemented as `nn.module()` objects which applied their respective transforms during the `forward()` method. This allowed for `autograd`[1] to dynamically generate the computational graph, and therefore maintain the ability of the network to perform back-propagation. Note that this is only required in the case of quantisation-aware training and quantisation with retraining (Sections 2.3.1 and 2.3.3 respectively).

3.4 Evaluating performance

All evaluations were done on networks trained using the same seeds for random number generation and the same network architecture as the respective baseline used in the comparison. The benefit of using the same seed is that it reduces the amount of variation in performance due to differences in random processes such as weight initialisation. On the other hand, a potential drawback of keeping this constant is that the particular random seed in use may be favourable to one quantisation scheme more than others, thereby creating an unfair comparison. Ideally, the tests should be

[1] Autograd is the PyTorch utility which facilitates the generation of the dynamic computational graph.

repeated on a range of random seeds. However, this was not prioritised since other configuration parameters were deemed more important to vary during testing.

Additionally, it may be the case that one particular architecture is suited best to a certain quantisation scheme. For example, a layer with a sigmoid [35] activation function will have its output range bound between 0 and 1. As a result, it is less likely to have large quantisation errors due to clipping, when compared to a layer with a ReLU [34] activation function whose range is unbounded. Therefore, it may be beneficial in that case to use a quantisation scheme with smaller intervals and a smaller range. This means that one can't necessarily conclude that one scheme is always preferable to another based on the results obtained from one network architecture. However, the aim of developing a learnable quantisation scheme (Chapter 5) is that by optimising against network loss, the scheme can adapt to different architectures.

3.5 Metrics

To evaluate the performance and cost of a given quantisation scheme, various metrics were considered. Firstly, the maximum test accuracy was used to measure the inference capability of the network. Hence, in this report network task performance was measured via the maximum test accuracy.

For the experiments conducted in Chapter 4, the network training time is used as a measure of the cost of implementing a specific quantisation scheme, since the addition of the forward and inverse transform impacts the network's computational load. For example, the scheme described in Section 4.1.1 requires computing a logarithm and power operation for each element of the activation tensor at each layer, thereby adding additional computations whilst the network is training. Additional computing costs during training result in additional power utilisation and as a result, additional environmental consequences and monetary consequences (as mentioned in Section 7.2).

However, collecting accurate training time metrics is difficult. This is because the networks are trained on shared GPU resources, which results in the GPU load rising and falling at various points due to other users running programs. This ultimately affects the time taken to train and therefore makes the results observed less indicative of whether there was an increased cost due to implementing the current quantisation scheme.

The results in Chapter 4 show that the recorded training time is a noisy measure of cost. There-

fore, the experiments done in Chapter 5 offer the time and space complexity of each quantisation scheme as an alternative.

Finally, the resource consumption of the network when deployed onto an FPGA was considered. Since the network is deployed to hardware without any arithmetic units, the number of LUTs used by the design is a suitable measure of overall resource consumption (since no DSPs or BRAMS are consumed). Additionally, due to the nature of the PolyLUT style deployment, the resource consumption of the network with regards to FPGA resources such as LUTs has a fixed upper bound. This upper bound can be calculated per neuron by finding the size of the un-optimised logical lookup table generated. For example, a neuron with a fan-in of 3 and a bit-width of 2 has 6 input bits in total. Therefore the number of addresses in the un-optimised lookup table is 2^6.

Hence, the total upper bound can be calculated as the number of LUTs required to instantiate all the input-output mappings within the network (all of the neurons). However, in practice, the resource consumption once the network is deployed is less predictable since the synthesis tool can make optimisations whilst instantiating these input-output mappings.

These optimisations are far less predictable as they are only known once the network is fully trained and the input-output mappings are created. A more complex quantisation scheme may result in fewer optimisations being done since it adds complexity to the overall input-output mapping.

Data about resource consumption is quicker to collect than latency since for a reliable estimate, only design synthesis is required. However, it was still not prioritised, since it still substantially slows down the iteration cycle. Furthermore, experiments conducted in Chapter 4 show that resource utilisation is fairly consistent across the different quantisation schemes.

Similarly, whilst inference latency and throughput are important, since these metrics reflect the operational performance of the network, data for these metrics has not been recorded. This is because the critical path of the network, like the resource consumption, is upper bounded due to the PolyLUT style deployment.

3.6 Baseline model

To allow for comparison of various quantisation schemes, suitable baseline models were required. Furthermore, to ensure fair comparison when changing the quantisation scheme, all configuration

variables unrelated to quantisation are kept constant unless stated otherwise. The baseline models use fixed-point quantisation implemented via Brevitas. The common variables are shown in Table 3.1, and the model architectures are described in Tables 3.2, 3.3 and 3.4 for the JSC-M-LITE, JSC-M and JSC-XL+ architectures respectively. The majority of testing has been done with these baseline models, when other model architectures are used it will be stated clearly.

Note that in this report, the bit-widths given describe the precision of layer activations. They may be written in the form X-Y-Z where X is the bit-width of the input layer [2], Y is the bit-width of the hidden layers and Z is the bit-width of the output layer. Otherwise, where bit-width is given as a single digit, that is the bit-width for all layers.

Architecture	JSC-M-LITE, JSC-M, JSC-XL+
Batch size	1024
Epochs	200
Degree	2
Quantisation scheme	Brevitas Fixed-Point
Scale implementation	Learned, Runtime statistics

Table 3.1: The baseline model.

Hidden layers	[64,32]
Input layer	[16, 64]
Output layer	[32, 5]
Hidden fan-in	4
Output fan-in	4
Input fan-in	4
Learning rate	0.001

Table 3.2: The JSC-M-LITE architecture.

Hidden layers	[64,32,32]
Input layer	[16, 64]
Output layer	[32, 5]
Hidden fan-in	4
Output fan-in	4
Input fan-in	4
Learning rate	0.001

Table 3.3: The JSC-M architecture.

[2]Both the input, and output quantiser of the input layer.

Hidden layers	[64, 32, 32, 32, 16, 8]
Input layer	[16, 64]
Output layer	[8, 5]
Hidden fan-in	2
Output fan-in	2
Input fan-in	2
Learning rate	0.01

Table 3.4: The JSC-XL+ architecture.

4

Implementation and Results - Initial transforms

Contents

This section goes into detail about the initial quantisation schemes implemented. The method by which quantisation schemes are implemented is as explained in Section 3.2. Initially, some quantisation schemes were manually chosen and implemented, and then a learnable scheme was developed which used a piecewise linear transform. The primitive version of this piecewise linear transform is included in this chapter, but this scheme is expanded upon in the next chapter and developed into the main outcome of this project. The final section of this chapter describes the conclusions obtained from the experiments done with the initial transforms, and the motivations for developing the piecewise learnable transform further.

In each section, the motivation, implementation and results of each scheme are discussed. Note

that each scheme incurs an additional computational load during training, however if deployed onto an FPGA for inference there is no additional logic required to implement any scheme aside from the implementation of the neuron LUT.

The first subsection goes into more detail about the process of applying the quantisation scheme to serve partly as an example of how these schemes are implemented. Note that all results shown in this chapter were obtained via quantisation aware training from scratch.

Each scheme is tested with different configurations, and compared with the corresponding baseline model performance. The baseline model was able to achieve the results shown in Table 4.1.

Architecture	Scale implementation	Degree	Bit-width	Max Test Accuracy (%)	Resource usage (CLBs)	Time to train (mins)
JSC-M-LITE	Learned	2	3	71.86	11362	184
JSC-M-LITE	Runtime statistics	2	3	71.775	-	175
JSC-M-LITE	Learned	2	2	71.332	-	177
JSC-M-LITE	Learned	6	3	72.18	-	1345
JSC-XL+	Learned	2	3	71.8	-	430
JSC-XL+	Learned	2	8	75.141	-	504

Table 4.1: Results of the baseline model on the JSC dataset.

4.1 Quantisation using a logarithmic transform

In this report, logarithmic representation is the interpretation of a bit-string's integer value as a base-2 exponent. Meaning that to return the value the quantised representation is storing, the following formula is used:

$$\text{Output} = 2^{Q(x)}$$

where $Q(x)$ is the stored quantised representation. Note that this is an unsigned interpretation.

Logarithmic representation in low-precision neural networks has been shown to be capable of yielding greater accuracy than uniformly quantised representations in some cases [29]. As a result, the first quantisation scheme that was tested in this investigation was emulating quantisation to logarithmic representation.

In theory, logarithmically distributed quantisation levels are helpful when the error due to clipping is high. This is because logarithmic representation can express a larger range than fixed-point quantisation for the same scale and number of bits. Logarithmic quantisation was implemented on the hidden layers only which employ a ReLU activation function [34]. This is because the input

and output layers use a Tanh activation function [36], which restricts the output range to between -1 and 1. Therefore, a logarithmic representation was deemed to be less effective since the range of the output space is bounded and small.

4.1.1 Applying the transform

To quantise to logarithmic representation, the following transform was applied:

$$T(x) = \begin{cases} 0 & x < 0 \\ \log_2(x+1) & x \geq 0 \end{cases}$$

The effect of this transform is shown in Figure 4.1.

This is because for $x < 1$, $\log_2(x)$ has a gradient which tends to infinity as $x \to 0$, resulting in an exploding gradient problem when included in the back-propagation chain. Therefore, $\log_2(x+1)$ is more suitable. Additionally, the negative codomain of the function is unimportant, because the output of the quantisation scheme will be passed through a ReLU activation function.

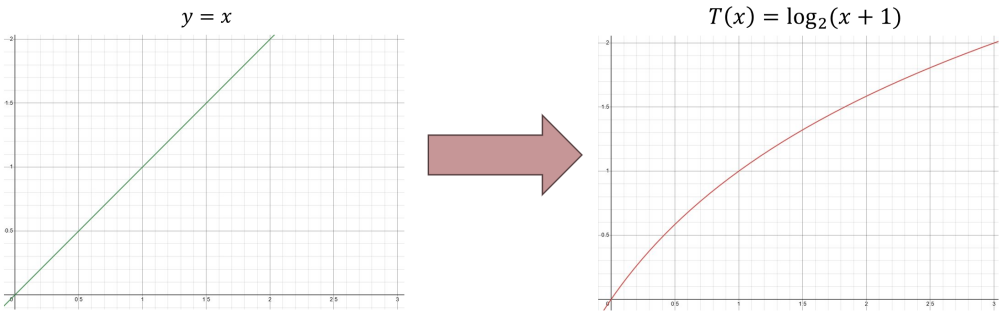


Figure 4.1: The output space is changed by applying the transform.

Following this, fixed-point quantisation was applied via Brevitas. The quantiser samples the transformed data at constant intervals, which is depicted in Figure 4.2. As a result, the output of the quantiser is the transformed data sampled in fixed intervals. The next step is to undo the augmentation of the data distribution.

Figure 4.3 depicts the undoing of the data augmentation via the application of the inverse transform. The sampled points lie on a straight line with a gradient of 1 and hence the distribution is no longer augmented. However, the quantisation intervals are now exponentially increasing. Therefore, non-uniform quantisation has successfully been applied without data augmentation.

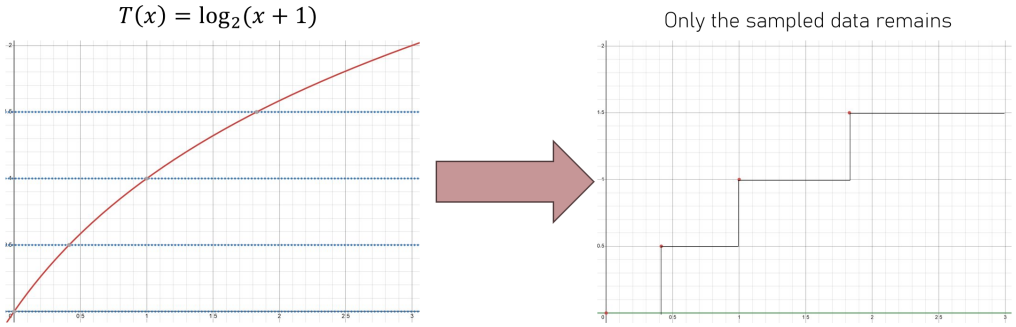


Figure 4.2: The fixed-point quantiser samples the output space of the transform at constant intervals.

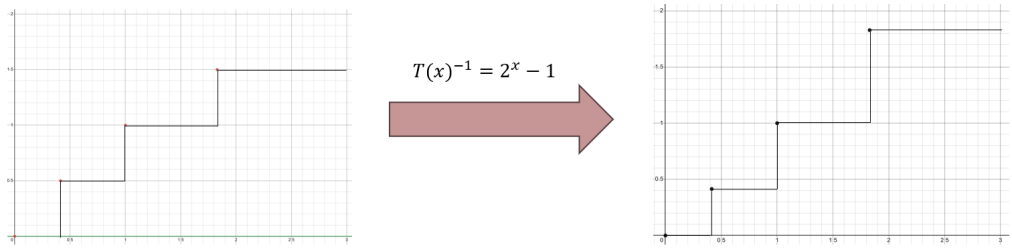


Figure 4.3: The quantised levels undergo the inverse-transform, thereby undoing the data augmentation.

4.1.2 Results and evaluation

The logarithm transform achieved a max test accuracy of 72.073 when trained with degree 6 and bit-width 3. The full results are shown in Table 4.2.

Architecture	Scale implementation	Degree	Bit-width	Max Test Accuracy (%)	Resource usage (CLBs)	Time to train (mins)
JSC-M-LITE	Learned	2	3	71.854	10274	214
JSC-M-LITE	Learned	6	3	72.073	-	1254
JSC-M-LITE	Learned	2	2	70.988	-	209
JSC-M-LITE	Runtime statistics	2	3	71.766	-	186

Table 4.2: Results of the logarithm quantised model on the JSC dataset.

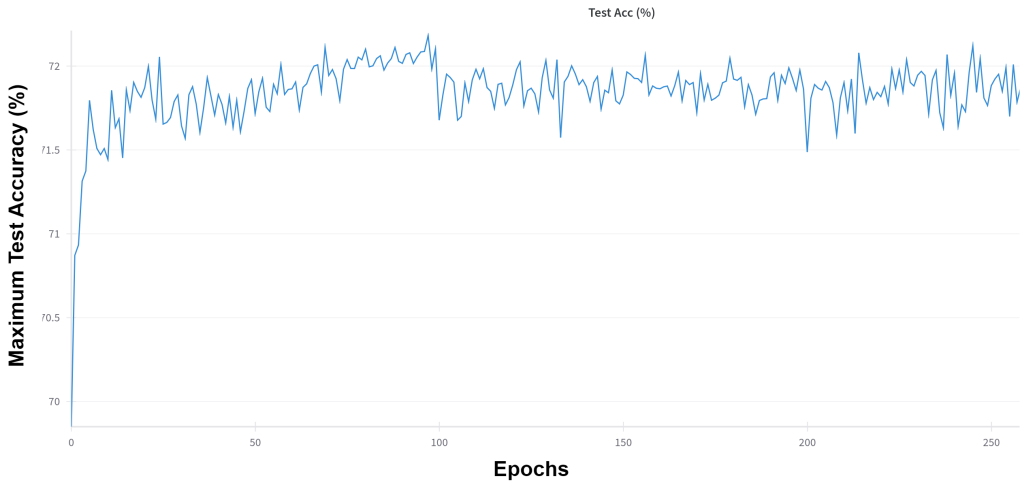
For the same initialisation and model, the logarithm transform performed almost exactly the same as the baseline, having a worse maximum test accuracy by 0.1%. This implies that the network performance did not improve through the addition of this quantisation scheme.

Investigating why this is the case is difficult, since under a QAT approach, the parameter values learned are influenced by the quantisation being applied. The learning of the network parameters and the quantisation scheme are effectively coupled.

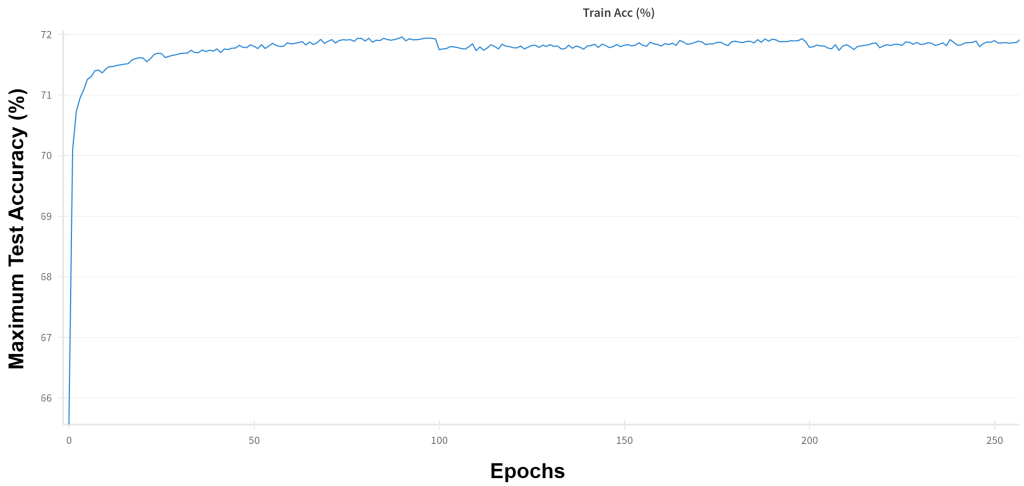
The test and train accuracy for both the baseline and logarithm transform models can be

seen in Figures 4.4 and 4.5 respectively. Interestingly, the test accuracy of the baseline model was more volatile during training than its logarithm counterpart. This may be because logarithm quantisation intervals are likely to grow to be larger than fixed point quantisation intervals, meaning that the inputs to quantisation need to change significantly to be quantised to a different level. Therefore, towards the end of training when weight values are not changing significantly, neurons are less likely to output different values and hence the output of the network is more stable.

This stability is beneficial, since it implies that the logarithmically quantised network may provide more reliable performance in real-world scenarios. This is because the test set is only an estimation of the population distribution of the dataset, therefore a volatile test accuracy may result in worse performance when the network is actually deployed.

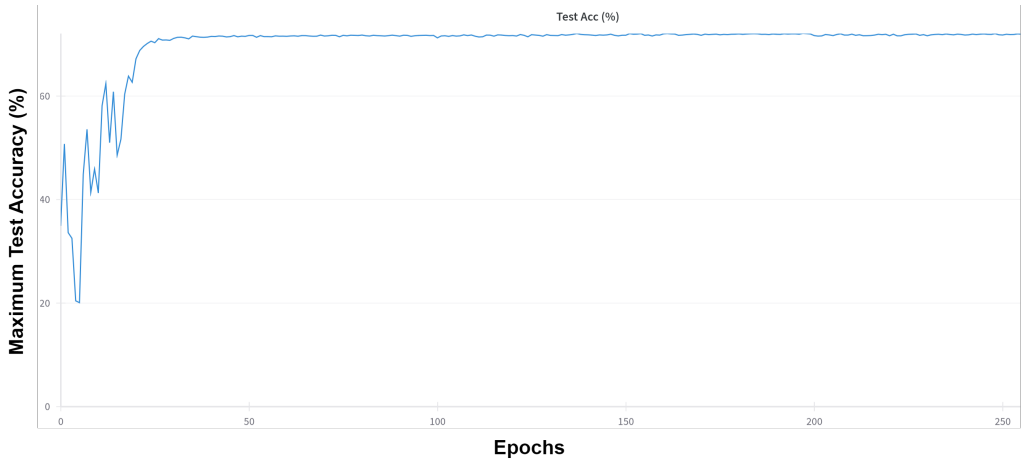


(a) Baseline model test accuracy versus epochs.

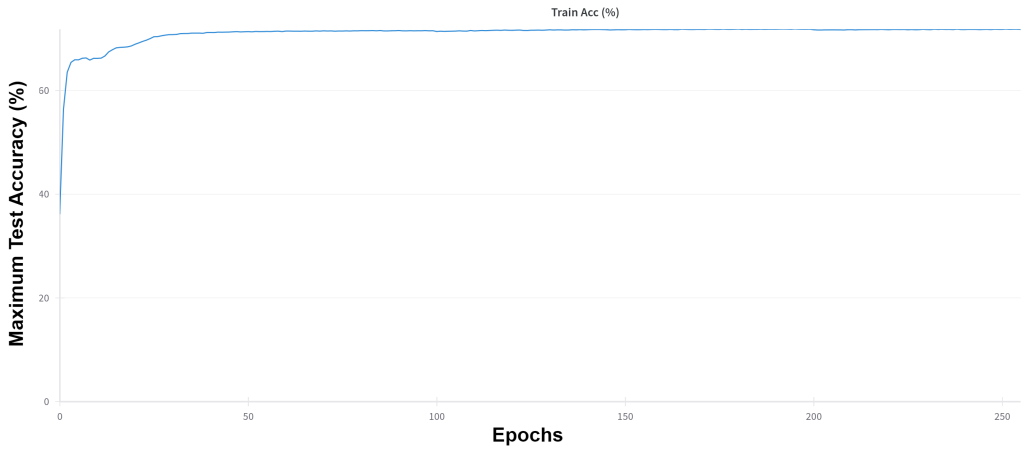


(b) Baseline model train accuracy versus epochs.

Figure 4.4: The train and test accuracy of the baseline JSC-M-LITE model.



(a) Logarithm quantisation model test accuracy versus epochs.



(b) Logarithm quantisation model train accuracy versus epochs.

Figure 4.5: The train and test accuracy of the logarithm transform JSC-M-LITE model.

4.1.3 Complexity

The time complexity of a quantisation scheme is the sum of the time complexity of the forward transform, the uniform quantiser and the inverse transform. Since the uniform quantiser is present in all quantisation schemes tested, its complexity will be denoted by C.

The complexity of the forward transform is $\mathrm{O}(3(N \times M \times K))$, where M, N and K are the dimensions of the input tensor. This is because for every value in the input tensor, there are 3 operations: a greater than comparison with zero, an addition by 1, and the application of $\log_2(\cdot)$.

The inverse transform has the same time complexity as the forward transform. Therefore, the time complexity simplifies to $\mathrm{O}(C + 6NMK)$. The memory complexity during training is zero since the transform and its inverse have no parameters, and hence do not require gradients to be stored. Therefore the additional complexity incurred is $\mathrm{O}(6NMK)$ since the uniform quantiser is present in all quantisation schemes (including the baseline). It is important to note that comparison operations are particularly expensive since they result in branching behaviour.

4.2 Quantisation using a linear-then-logarithm transform

This scheme involves applying a transform consisting of a piecewise function of 2 components. The transform is defined by the following equation:

$$T(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x < 1 \\ \log_2(x+1) & x \geq 1 \end{cases}$$

The transform is shown in Figure 4.6. The transform resembles the full logarithm transform described in the previous section, however, the addition of the linear section allows for a uniform quantisation when the inputs to quantisation are between a certain range. This is beneficial because it means that the network can change to a uniform quantisation scheme by changing the activation distribution, which it does through learning different network parameters (weights). However, since the transform is piecewise, the gradient function is discontinuous. Therefore, the network relies on the stochastic behaviour of training to move between the two regions of the transform.

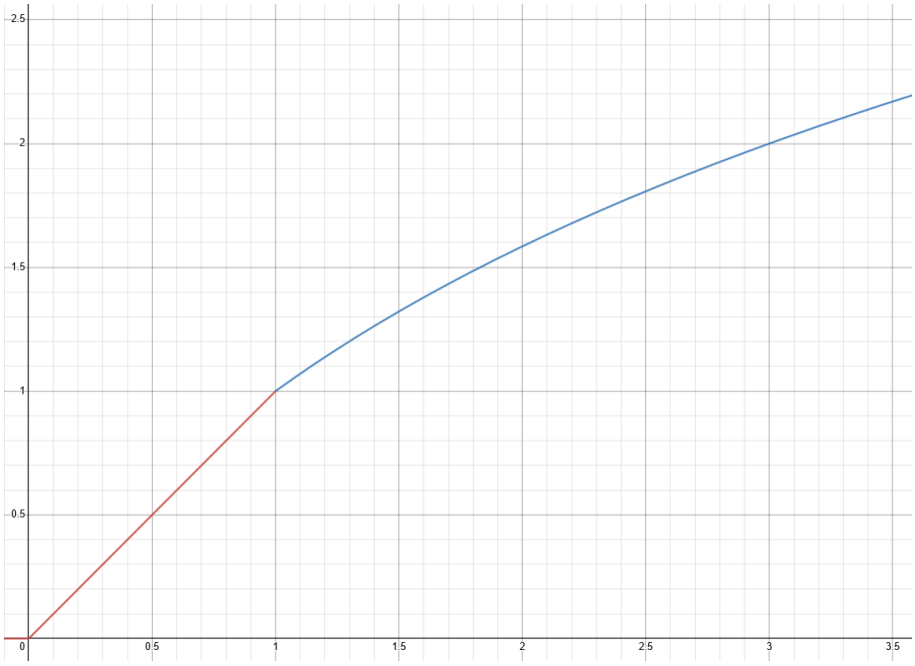


Figure 4.6: The linear-then-logarithm transform.

4.2.1 Results and evaluation

The linear-then-logarithm scheme performed better than the baseline in all common test scenarios. Although, the improvement is not very large - the best improvement was 0.1%. This is within the range of the fluctuations in test accuracy, and as a result, does not indicate that this scheme is an improvement over the baseline. Ideally, more data would be collected using many random seeds, network architectures and datasets, as this would help clarify if the improvement is due to the scheme.

Architecture	Scale implementation	Degree	Bit-width	Max Test Accuracy (%)	Resource usage (CLBs)	Time to train (mins)
JSC-M-LITE	Learned	2	3	71.96	-	289
JSC-M-LITE	Learned	6	3	72.194	-	2520

Table 4.3: Results of the linear-then-logarithm quantised model on the JSC dataset.

4.2.2 Complexity

The time complexity of this quantisation scheme is $\mathrm{O}(C + 8NMK)$ because in the case of $x \geq 1$, the forward and inverse transform each require 2 arithmetic operations and 2 comparisons per value in the input tensor. Furthermore, the memory complexity is zero since the transform and its inverse have no parameters, and hence do not require gradients to be stored.

4.3 Initial Parametrised Transforms

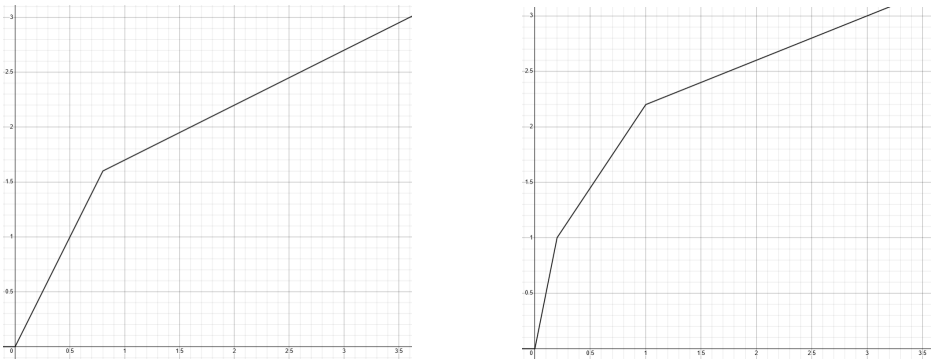
To have a transform be learnable, its formulation should be influenced by trainable parameters. The learnable transform developed is composed of piecewise linear sections. These sections use learnable parameters to define their slopes. Having these parameters control the shape of the transform means that the transform is learnable, and hence can be optimised via stochastic gradient descent (SGD). The computational graph for this transform and its derivative are shown later in this section.

Because of this, the network can use back-propagation to influence the quantisation scheme towards a lower loss. The benefit of using linear components is that with enough linear sections, the transform should be able to approximate any quantisation scheme.

Additionally, the fact that the transform is learnable, means that layers can have different quantisation schemes since each layer will have its own set of parameters which the quantisation scheme uses. Furthermore, the formulation shown in this section guarantees the transform to be invertible, which is essential for companded quantisation.

Initially, data was collected for both a 2-section piecewise function, and a 3-section piecewise function, which are depicted in Figure 4.7. In the next section, the piecewise transform was developed so that it supports any number of sections, which meant that the number of sections was a hyper-parameter that could be tuned.

4.3.1 Mathematical Formulation



(a) Piecewise linear transform with 2-sections. $b_a = 0.8, g_a = 2, g_b = 0.5$

(b) Piecewise linear transform with 3-sections. $b_a = 0.2, b_b = 0.8, g_a = 5, g_b = 1.5, g_c = 0.4$

Figure 4.7: The 2-section piecewise, and 3-section piecewise transforms.

The transform with 3 distinct linear sections is defined by the following equation:

$$T(x) = \begin{cases} 0 & \text{if } x < 0, \\ g_0 x & \text{if } 0 \leq x < b_0, \\ g_1 x - (g_1 b_0 - g_0 b_0) & \text{if } b_0 \leq x < b_1, \\ g_2 x - (g_2 b_1 - (g_1 b_1 - (g_1 b_0 - g_0 b_0))) & \text{if } x > b_1 \end{cases} \tag{4.1}$$

Where

$$g_0 = abs(g_a + \epsilon)$$
$$g_1 = abs(g_b + \epsilon)$$
$$g_2 = abs(g_c + \epsilon)$$
$$b_0 = b_a$$
$$b_1 = b_a + b_b$$

In this formulation, g_a, g_b, g_c are the learnable slope parameters, ϵ is a small number and b_a, b_b are the constant, positive boundary parameters which are fixed and not learnable. The slope parameters can be updated via back-propagation since they have a valid partial derivative. However, the boundary parameters are not learnable because when included in the transform expression, they are part of Heaviside functions [37] and hence their partial derivatives involve the Dirac Delta function [38].

Ensuring invert-ability

To ensure that the transform does not become invalid as a function, $g_0, g_1,$ and g_2 must all be entirely positive or entirely negative. This is because otherwise, the transform becomes a many-to-one mapping, and hence has no inverse. In this project, they were chosen to be positive.

Additionally, g_0, g_1 and g_2 must always be greater than zero to avoid divide by zero errors since the sections of the corresponding inverse transform have slopes equal to $\frac{1}{g_0}, \frac{1}{g_1}, \frac{1}{g_2}$. Therefore these values are defined by the absolute value of the slope parameters plus some epsilon.

Furthermore, boundaries such as b_1 [1] are defined through summations. This is because this

[1] Note that boundaries are not the same as boundary parameters.

formulation requires that the boundary parameters (such as b_a) are positive, so a summation ensures that each boundary is larger than the previous. Therefore, this guarantees that $b_1 > b_0$, which is necessary because if $b_1 < b_0$, the transformation becomes a one-to-many mapping and hence is not a valid function (so it has no inverse). Note that the formulation of the inverse transform does not require any additional parameters.

Similarly, the transform with 2 distinct linear sections is defined by the following equation:

$$T(x) = \begin{cases} 0 & \text{if } x < 0, \\ g_0 x & \text{if } 0 \leq x < b_0, \\ g_1 x - (g_1 b_0 - g_0 b_0) & \text{if } b_0 \leq x, \end{cases} \tag{4.2}$$

Where

$$g_0 = abs(g_a + \epsilon)$$
$$g_1 = abs(g_b + \epsilon)$$
$$b_0 = b_a$$

Note that as expected, the transform with 2-sections requires 1 less slope parameter and 1 less boundary parameter. In other words, each additional section adds a small cost to implementing the transform.

As mentioned previously, by changing these parameters, one can essentially change the shape of the transform and hence the quantisation scheme. For example, Figure 4.8 highlights how the shape of this transform (and hence the resulting quantisation scheme) can vary greatly depending on the parameters.

Ensuring continuity

As shown in Equations 4.1 and 4.2, each piecewise section has an additional constant subtracted from the linear term. This value is referred to as the *adjustment* in this report, and is required to maintain the transform's continuity. The term guarantees that the start of each section (on the x-y plane) is at the point where the previous section ended. Without this term, the transforms would be discontinuous, an example of this is shown in Figure 4.9 where the parameters of the

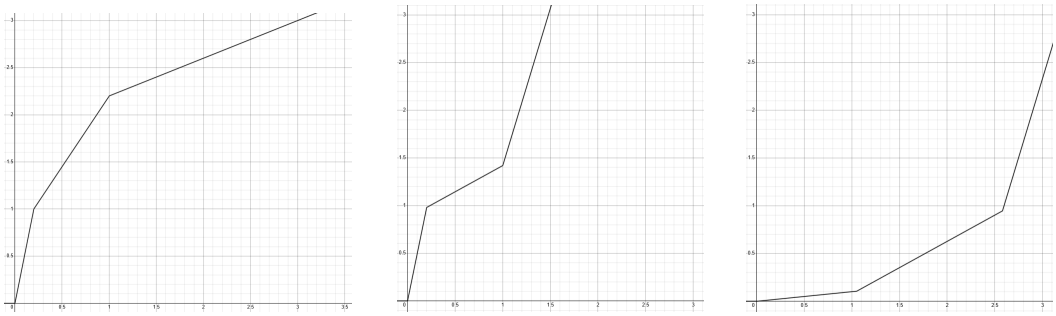


Figure 4.8: The piecewise transform is versatile and can have different shapes depending on the parameter values.

transform are the exact same as those in Figure 4.7b.

Figure 4.9: Piecewise linear transform with 3-sections. $b_a = 0.2, b_b = 0.8, g_a = 5, g_b = 1.5, g_c = 0.4$. The transform is discontinuous due to absence of the *adjustment* term.

Derivative and computational graph

The computational graph for the transform when the input is in the domain of the first piecewise section is shown in Figure 4.10. The partial derivative of y with respect to g_0 is shown in Equation 4.3.

$$
\begin{aligned}
\frac{\partial y}{\partial g_0} &= \frac{x}{g_0} - \frac{Q(g_0 \cdot x)}{{g_0}^2} \\
Q(g_0 \cdot x) &= g_0 \cdot x_q \\
\frac{\partial y}{\partial g_0} &= \frac{x}{g_0} - \frac{a \cdot x_q}{{g_0}^2} \\
&= \frac{x}{g_0} - \frac{x_q}{g_0} \\
&= \frac{1}{g_0}(x - x_q)
\end{aligned}
\tag{4.3}
$$

In this formulation, $Q(\cdot)$ represents the uniform quantiser. Therefore, $Q(g_0 \cdot x) = g_0 \cdot x_q$ because the multiplication by g_0 is a uniform scaling of the input, and hence is absorbed into the scale variable of the quantiser. The derivation shows that the partial derivative with respect to the parameter g_0 depends on the difference between the quantised output, and the unquantised input. In other words, when the quantised output is close to the unquantised input, the partial derivative is small.

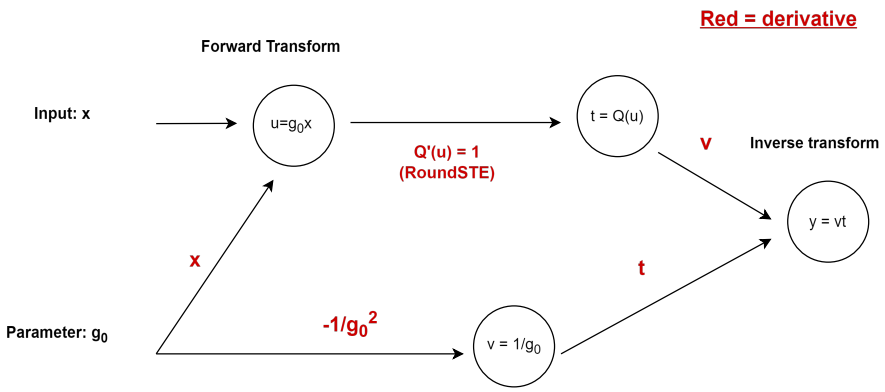


Figure 4.10: The computational graph of the transform when the input is in the domain of the first piecewise section.

4.3.2 Results and evaluation

The full results can be seen in Tables 4.4 and 4.5 for the 2 and 3-section transforms respectively. Overall, the piecewise transform scheme did show some improvement over the baseline, although this was only the case for the 3-section transform. The 3-section transform was slightly better than the baseline in almost all cases with the exception being in the JSC-XL+ bit-width 8 scenario.

The reason why the 3-section transform performed better than the 2-section might be because with more sections, the transform is more versatile and hence the network has more freedom to influence the quantisation scheme.

As expected, the 3-section transform performed better for degree 6 models than degree 2. Additionally, the largest improvement over the baseline was for bit-width 2 and degree 2 in the JSC-M-LITE model. However, this improvement was still relatively small at around 0.2%.

Interestingly, the resource usage is slightly less than the baseline and the logarithm transform. This is difficult to interpret and, since the difference is not significant, could simply be attributed to the unpredictable nature of the synthesiser.

Multiple tests were run on the 3-section piecewise transform, and in most cases there was some slight improvement over the baseline. However, in some initialisations, this was not the case. This implies that like many learnable schemes, the performance of the transform is affected by its initialisation as shown by the results in Table 4.6.

The table also shows that even though the slope parameters had significantly different final values, the test accuracy was similar. This implies that the slope values and hence the quantisation scheme are indeed coupled with the network weights.

Architecture	Scale implementation	Degree	Bit-width	Max Test Accuracy (%)	Resource usage (CLBs)	Time to train (mins)
JSC-M-LITE	Learned	2	3	71.832	-	201

Table 4.4: Results of the 2-section piecewise transform on the JSC dataset.

Architecture	Scale implementation	Degree	Bit-width	Max Test Accuracy (%)	Resource usage (CLBs)	Time to train (mins)
JSC-M-LITE	Learned	2	3	72.012	10525	197
JSC-M-LITE	Learned	6	3	72.124	-	2314
JSC-M-LITE	Learned	2	2	71.553	-	212
JSC-M-LITE	Runtime statistics	2	3	71.803	-	274
JSC-XL+	Learned	2	3	71.941	-	430
JSC-XL+	Learned	2	8	75.125	-	601

Table 4.5: Results of the 3-section piecewise transform on the JSC dataset.

Initialisation	Slope 1 Final value	Slope 2 Final value	Slope 3 Final value	Bound 1	Bound 2	Maximum Test Accuracy	Better Than Baseline
1	11.0247	1.0385	10.1727	0.08	0.12	72.012	✓
2	0.5433	0.5986	2.0127	0.315	0.7556	71.931	✓
3	4.449	0.0156	4.3796	0.08	0.12	71.885	✓
4	15.9747	0.0825	12.4428	0.08	0.12	71.833	×
5	0.4197	3.8522	7.0346	0.7736	0.7672	71.871	✓
6	4.7291	2.9209	2.6404	0.0105	0.4369	71.815	×

Table 4.6: Results of 3-bit JSC-M-LITE networks trained with the 3-section piecewise transform under different initialisations.

4.3.3 Complexity

The time complexity for the 3-section piecewise forward transform is O($5NMK$) since there are 3 comparison operations in the worst case, 1 multiplication and 1 addition performed per value in the input tensor to find which bounds the value lies between. For a transform with a large number of sections, this comparison could be optimised using binary search. The inverse transform has the same complexity as the forward, therefore the total complexity is O($C + 10NMK$). The memory complexity is O(3) since the transform and its inverse share the three slope parameters, and hence do not require gradients to be stored.

4.4 Conclusions

The maximum test accuracy and resource consumption of the various quantisation schemes covered so far are shown in Figures 4.11 and 4.12 respectively. The results imply that overall, the 3-section piecewise transform seems to yield the best performance, with the largest improvement being for bit-width 2 trials.

In terms of cost, none of the quantisation schemes had a significant impact on resource consumption. As mentioned previously, the resource consumption of PolyLUT models has a fixed upper bound that is a function of the bit-widths of the network, and hence the only way the quantisation scheme can affect resource consumption is if it reduces or increases the amount of optimisation the synthesiser can do.

In evaluation, there are some flaws in the implementation and testing methodologies. Firstly, the data collected for the time taken to train is unreliable. As mentioned previously, this is because the hardware resources used to train the model (GPUs) were shared. As a result, the time taken to train was affected heavily by other users putting the resource under load.

These findings raise the question of why there is relatively little difference in the performance of the various quantisation schemes when compared to the baseline. One possibility is that the exact quantisation scheme may not be important because the network can influence the output of quantisation by changing the weights.

In other words, instead of changing the quantisation scheme, the network can change the distribution of the input to the quantiser. From this perspective, the benefit of adding a learnable quantisation scheme is that the network parameters (weights) are no longer constrained by the

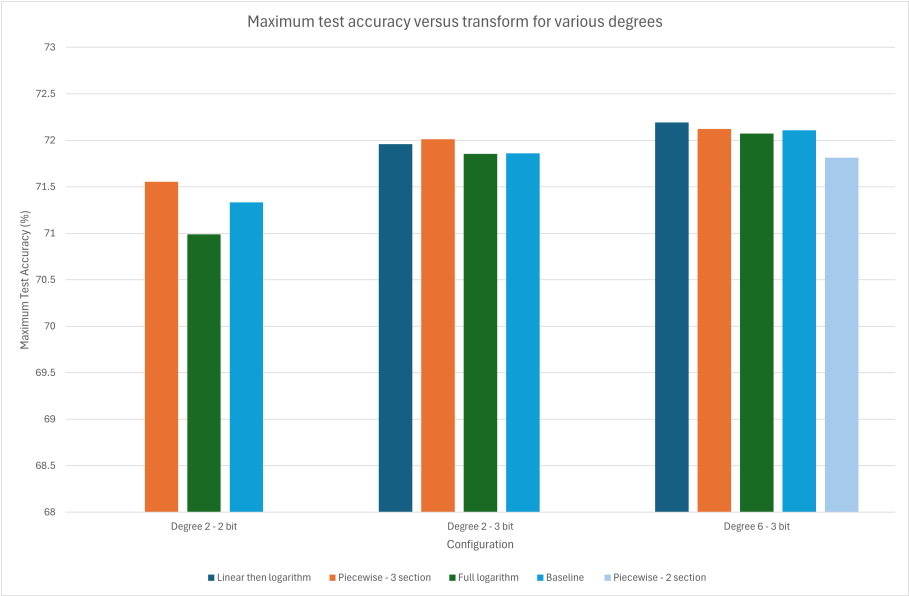


Figure 4.11: The performance of the various quantisation schemes when tested on the JSC-M-LITE model.

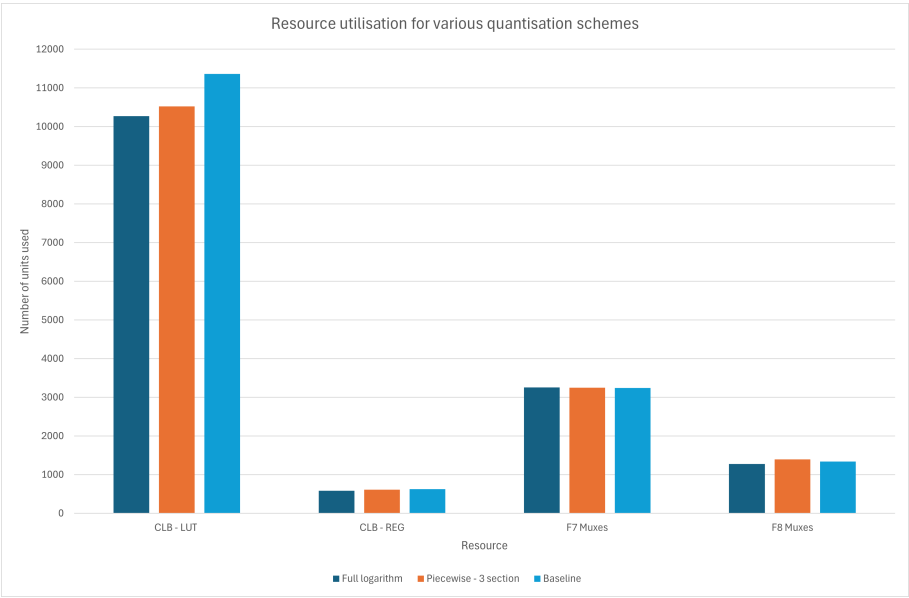


Figure 4.12: The resource consumption of the various quantisation schemes when tested on the JSC-M-LITE model.

uniform quantiser - in other words, the weights are less motivated to learn values that reduce quantisation error, since this can be done by the transform parameters.

The benefit of non-uniform quantisation may require tuning of the quantisation hyper-parameters, such as the number of piecewise sections. The following chapter expands on the piecewise linear transform quantisation scheme, and explains the various developments that led to an improvement in network task performance over the baseline.

5

Implementation and Results - Variable linear piecewise transform

Contents

This section details the various developments made to the piecewise learnable quantisation scheme throughout the project. Note that the implementation of this quantisation scheme is the same as in Section 3.2, and the mathematical formulation remains the same as explained in Section 4.3.1.

During the testing of each development, the control variables remained constant, with only the feature under development being modified unless stated otherwise in the results. However,

control variables may differ between subsections so one should not compare the results between subsections. Finally, the term *piecewise transform* by default refers to the transform applied per-tensor. All developments in this chapter, unless specified otherwise such as in Section 5.10, were tested on the per-tensor piecewise transform. Furthermore, to reduce the time required per trial, only the JSC-M-LITE and JSC-M networks were used for testing (larger networks took too long to train).

In these experiments, resource usage was not measured since previous results indicate that quantisation scheme has little effect on the resource usage, likely because the resource usage has a fixed upper bound regardless of the quantisation scheme (Section 3.5). For the same reason, the inference latency once deployed was not measured. Additionally, training time was not recorded due to the unreliability of its measurement (as mentioned in previous sections).

Additionally, all layers on networks with no transform applied were quantised using a Brevitas quantiser with a runtime statistics scale implementation, whereas networks with transforms used a constant scale of 1.0. This is explained further in Section 5.7. The results shown in the following subsections may not draw direct comparison to the baseline, since this is left to Chapter 6 where the final results are compared and evaluated.

5.1 Automatic construction

In the initial implementation of the piecewise transform, the control logic was hard-coded. This method involved explicitly checking which pair of bounds the input fell between and then applying the corresponding transform. Although functional, this approach was rigid and lacked flexibility, necessitating changes to the code whenever the configuration of the piecewise function needed to be altered.

To reduce the development time required when implementing piecewise transforms, a Python class was developed based on OOP fundamentals (Section 2.8) which automates the construction of the transform by automatically creating the 3 required arrays: the bounds array, the slopes array, and the adjustments array. This approach abstracts the logic required to apply the transform, allowing any configuration to be utilised without modifying the underlying code. The Python class leverages PyTorch methods in order to preserve automatic differentiation, ensuring that the gradients are correctly propagated through the piecewise function.

5.2 Enabling on hidden layers only

Applying the piecewise transform incurs a computational cost which results in slower training. This is proven by the complexity analysis done in Section 5.8. Therefore, in some experiments only the hidden bit-width was varied and hence the transform (and its inverse) were only applied to the hidden layers. In experiments where all bit-widths are varied, the transform is applied on all layers.

The hidden layers were chosen over the input and output layers since the input and output layers are likely to have data distributions largely influenced by the dataset. This is because the input layer interacts directly with the dataset and the output layer aims to reproduce the distribution of the dataset. Therefore, since testing is done only on one dataset, this may limit the range of results observed.

5.3 Measuring the negative impact of quantisation

As mentioned in previous sections, to benefit from non-uniform quantisation, there should be a measurable detriment due to quantisation error. For example, if a PolyLUT network trained with 10-bit activations achieves the same accuracy as a network trained with 8-bit activations, it can be argued that there is no room for accuracy improvements made through alternate quantisation schemes. This is because the lower precision model performs just as well as the higher precision model, even with larger quantisation error.

Therefore, to evaluate the efficacy of non-uniform quantisation, one should first measure if there is a loss in task performance due to quantisation. To that end, experiments were conducted to measure the accuracy of the JSC-M-LITE network at various bit-width configurations.

These results can be seen in Figure 5.1. The results show that for a JSC-M-LITE network with eight bit input and output layers, there is no benefit in having a hidden bit-width larger than five bits. Likewise, for a JSC-M-LITE network with three bit input and output layers, there is no benefit in having a hidden bit-width larger than four bits.

The results also show that hidden bit-width has a larger impact on test accuracy per bit when the network has eight bit input and output layers, rather than three bit. This relates to the results observed in Section 5.4. Therefore, the majority of experiments were done using eight bit input

and output layers, since this configuration is more likely to benefit from non-uniform quantisation.

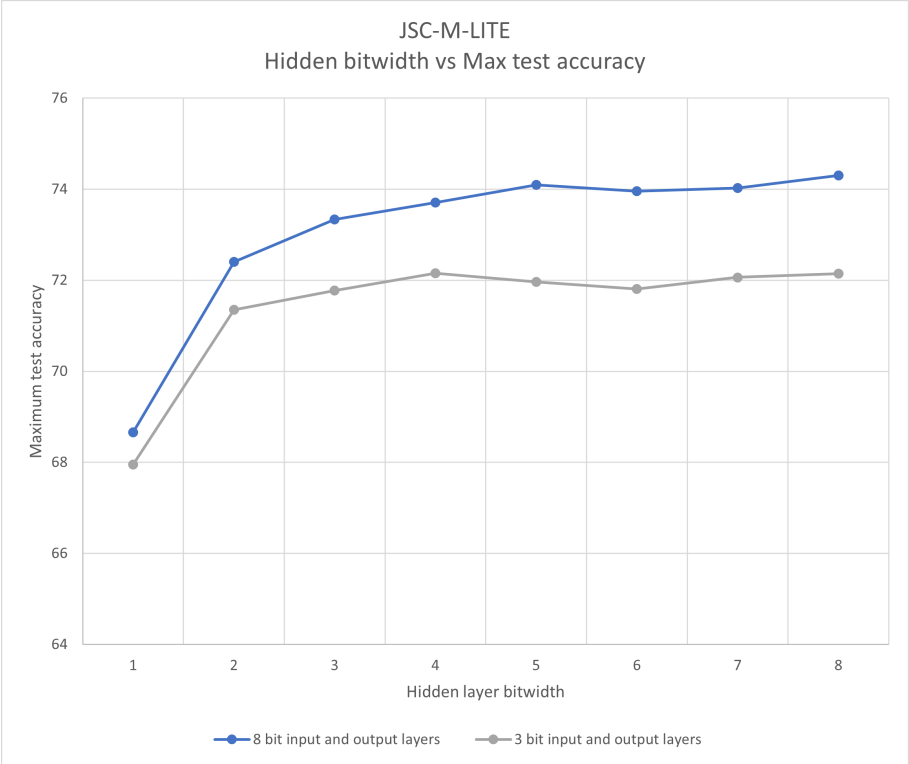


Figure 5.1: The maximum test accuracy of the JSC-M-LITE network for different hidden layer bit-widths.

5.4 Network input quantisation

Throughout the experiments, one interesting result observed is that quantisation of the input to the network results in an accuracy bottleneck. In other words, if the precision on the input layer is too low, the network does not gain significant benefit from reduced quantisation error in the latter layers. Figure 5.2 shows that networks are more sensitive to input layer quantisation than output layer quantisation. Additionally, the figure shows that the gap in accuracy between transform-enabled networks and transform-disabled networks shrinks when the input layer is quantised. Essentially, input layer quantisation acts as a bottleneck for accuracy improvements achieved through reduced quantisation error in later layers.

One reason for this is that the input layer of a neural network receives raw data, which is the source of information for the entire network. Quantising the input data means that some of the original information is lost or significantly altered right at the beginning. This loss of information cannot be recovered or compensated for by subsequent layers, leading to a degradation in the

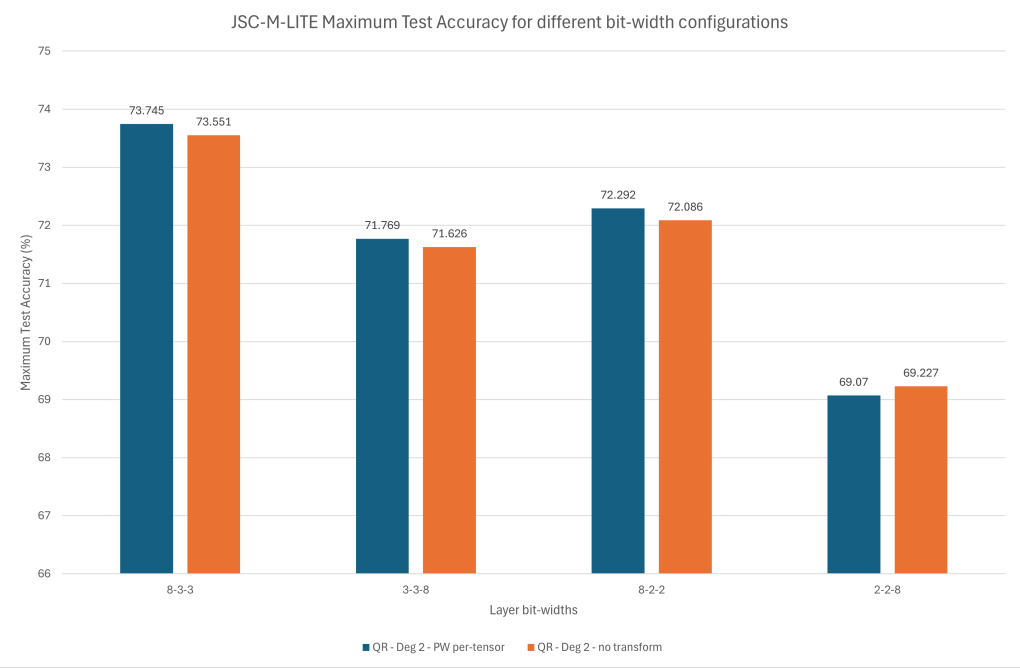


Figure 5.2: The test accuracy of the networks for different bit-width configurations. Performance is significantly worse when the input layer is severely quantised, whereas the output layer is less sensitive to quantisation error.

overall performance of the network.

Furthermore, when quantisation occurs at the input layer, the errors introduced are propagated through all subsequent layers. This can lead to a cumulative effect where the errors compound as the data moves through the network, further degrading performance. In contrast, if quantisation is applied only to hidden and output layers, the initial input information is still accurate, and the network can still perform meaningful transformations on the original data.

Additionally, the initial layers of a neural network are responsible for extracting basic features from the input data. Quantisation at this stage can distort these basic features, making it difficult for subsequent layers to recognize more complex patterns and structures. This is less problematic when quantisation is applied to deeper layers, where the features being processed are already more abstract and less sensitive to small perturbations.

In summary, quantising the input layer leads to a fundamental loss of information and introduces errors early in the processing pipeline, which cannot be corrected by subsequent layers. This results in a significant accuracy bottleneck. On the other hand, quantising hidden layers affects the activations after initial feature extraction has taken place, allowing the network to better manage and mitigate the impact of reduced precision through its learned parameters and structure.

5.5 Quantisation with Retraining - Experimentation

As explained in Section 2.3.3, QR can be an effective way to mitigate the accuracy losses incurred from neural network quantisation. Therefore, experiments were conducted on the JSC-M and JSC-M-LITE network architectures to investigate this quantisation approach.

Networks were trained at 8-bit precision on each layer, for the reasons explained in Section 5.3, until the test accuracy plateaued. Then the hidden layers were quantised to lower precision. Following this, the networks were trained for more epochs. This proved to yield accuracy improvements over networks trained with hidden layers at low precision from scratch.

Figure 5.3 shows the results for networks quantised using this approach versus networks trained at low precision from scratch. The results show that there is a consistent increase in test accuracy when using a QR approach as opposed to training with low precision from the scratch. This effect is observed regardless of whether the transform is enabled.

However, the results also show that the networks with the piecewise learnable transform enabled perform better than networks without under the QR. Note that in these results, the transform parameters were learned during the high precision portion of training. This increase in performance may be because the transform helps to retain information about the high precision distribution of the activations.

5.6 Post-Training Quantisation - Experimentation

As explained in Section 2.3.2, PTQ is a simple and cheap method by which to implement neural network quantisation. Experiments showed that networks with the piecewise learnable transform enabled performed significantly better under the PTQ approach than networks without. These results can be seen in Figure 5.7.

Networks were trained at 8-bit precision on each layer, and then the hidden layers were quantised to lower precision and the network accuracy was tested. Furthermore, the transform parameters were trained during the high precision portion of training.

The results show that for quantisation to low precisions, PTQ performs significantly worse than QAT when no transform is applied. However, networks trained with the transform are able to achieve good results when quantised via PTQ, similar to results obtained via QAT from

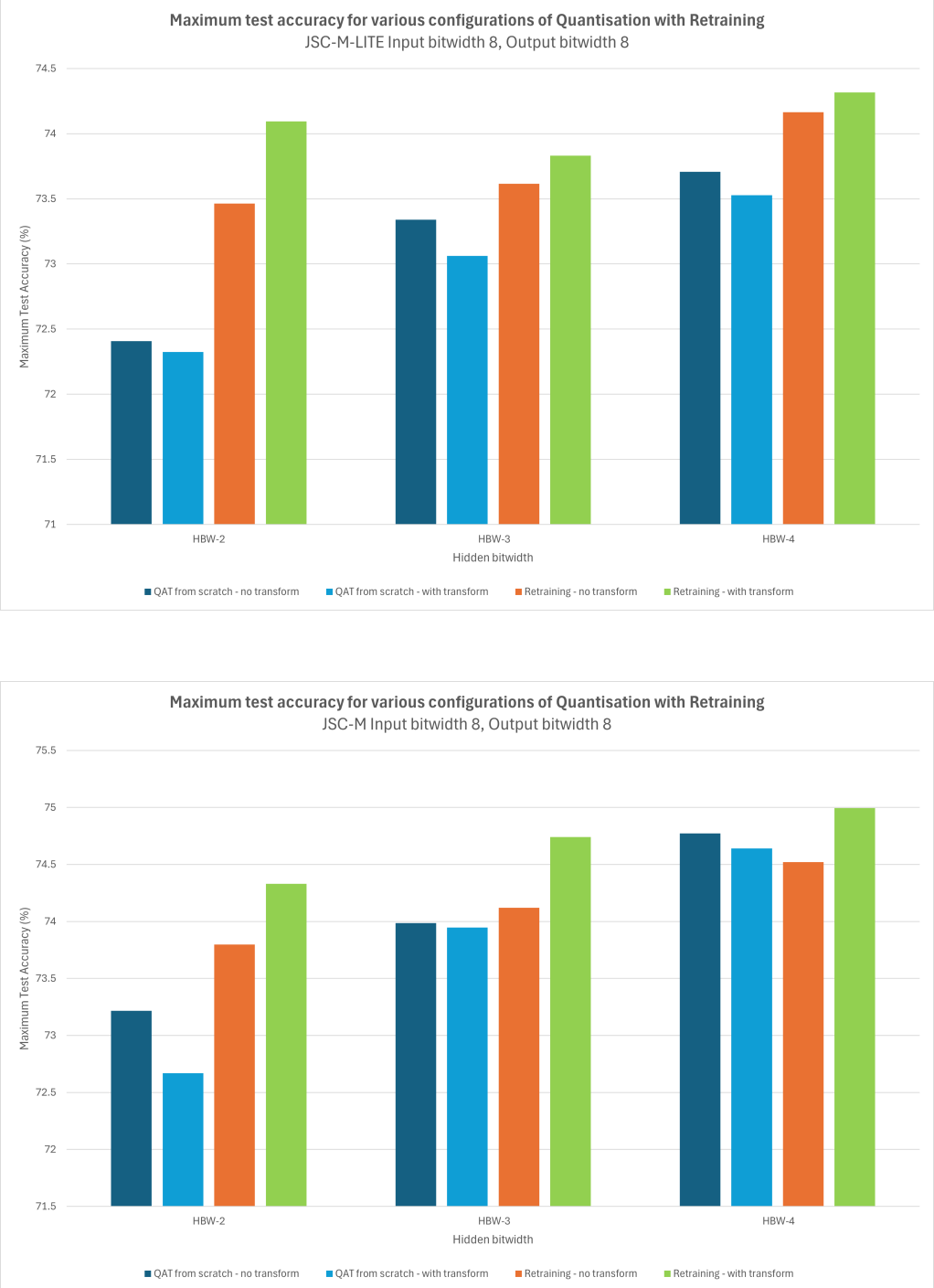


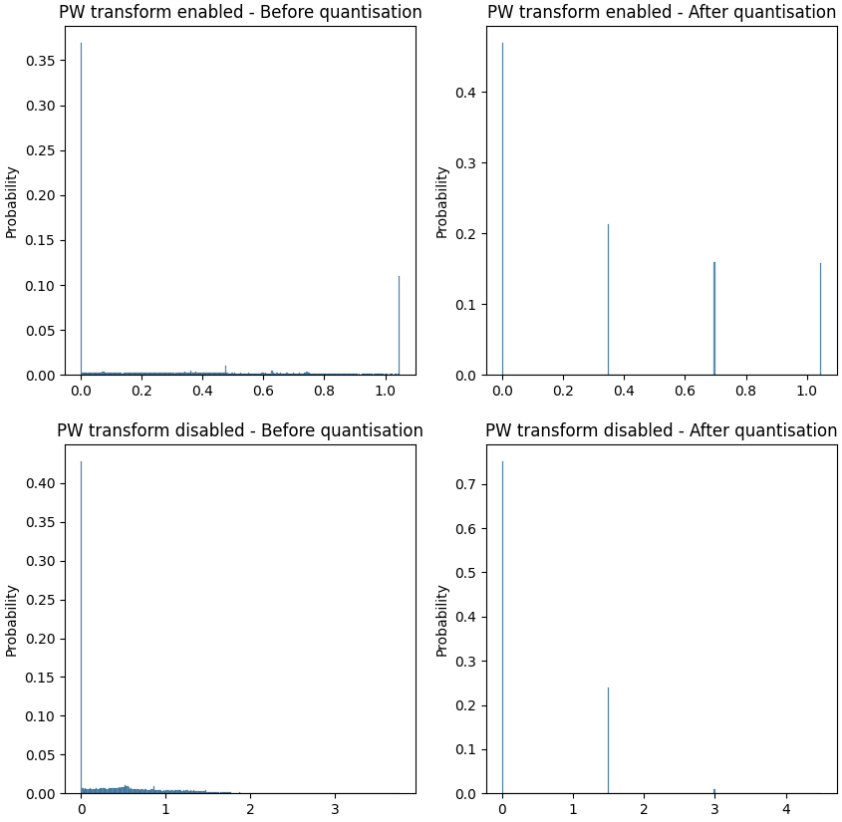
Figure 5.3: The accuracy achieved through QR for JSC-M-LITE and JSC-M networks across various hidden bit-widths.

scratch. This may be because (as mentioned in Section 5.5) by allowing the transform parameters to be learned at high precision, the network is able to retain information about the high precision distributions of the activations. Therefore, when the network is quantised, the distribution of the low precision activations is more similar to that of the high precision activations, which improves the accuracy of the network.

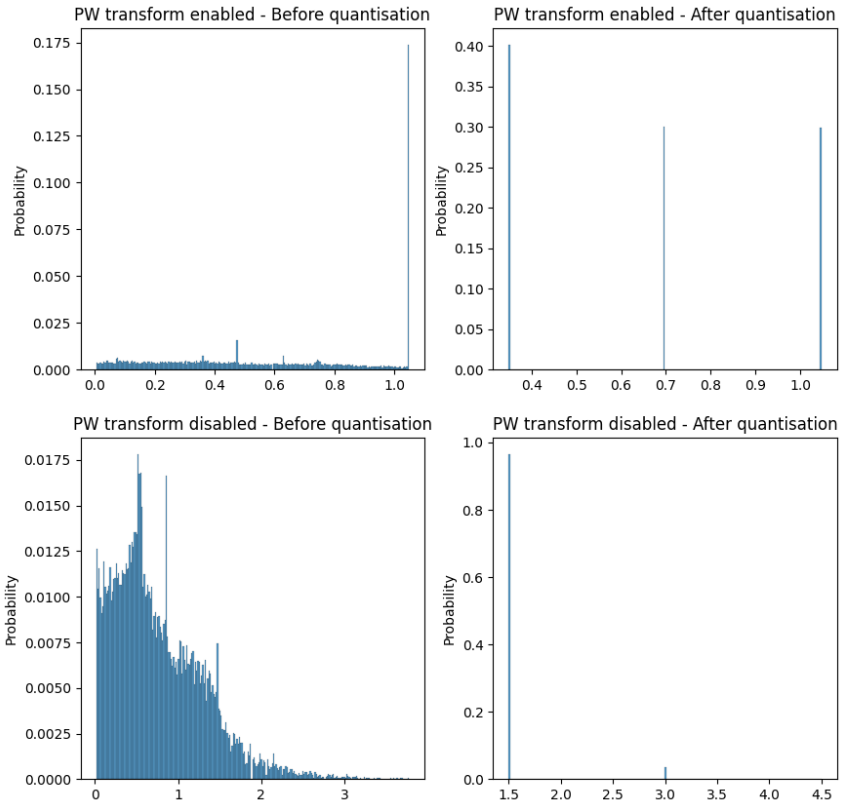
Figures 5.4a and 5.4b compare the hidden layer activation distributions before and after quantisation. Figure 5.4a includes values at zero, while Figure 5.4b excludes them. Since the quantiser always includes a level at zero, the distribution of non-zero values determines the variable quantisation levels. When the transform is enabled, the activation distribution resembles a uniform distribution, making uniform quantisation less detrimental. This means the post-quantisation plot closely approximates the pre-quantisation plot when linearly interpolated. This is not observed in the network trained without the transform.

Figure 5.5 shows the cumulative distribution of hidden layer activations before and after quantisation for networks trained with and without the transform. This figure demonstrates that, with the transform enabled, the post-quantisation distribution closely matches the pre-quantisation distribution, unlike in the network trained without the transform.

The results from Section 5.5 show that if allowed to retrain, the networks without the transform are able to re-learn the network parameters such that the performance is comparable to networks with the transform enabled. This suggests that by changing the weights, the network is able to change the distribution of the activations such that quantisation is less detrimental to network accuracy. Figure 5.6 evidences this claim by showing that after retraining, the quantised distribution far better approximates the high precision distribution (as opposed to the distribution just after quantisation).



(a) The probability density distribution of the hidden layer activations, including values at zero.



(b) The probability density distribution of the hidden layer activations, excluding values at zero.

Figure 5.4: The distributions of the hidden layer activations, before and after quantisation from 8-bit to 2-bit, for networks trained with the transform enabled and disabled. The distribution includes values at zero.

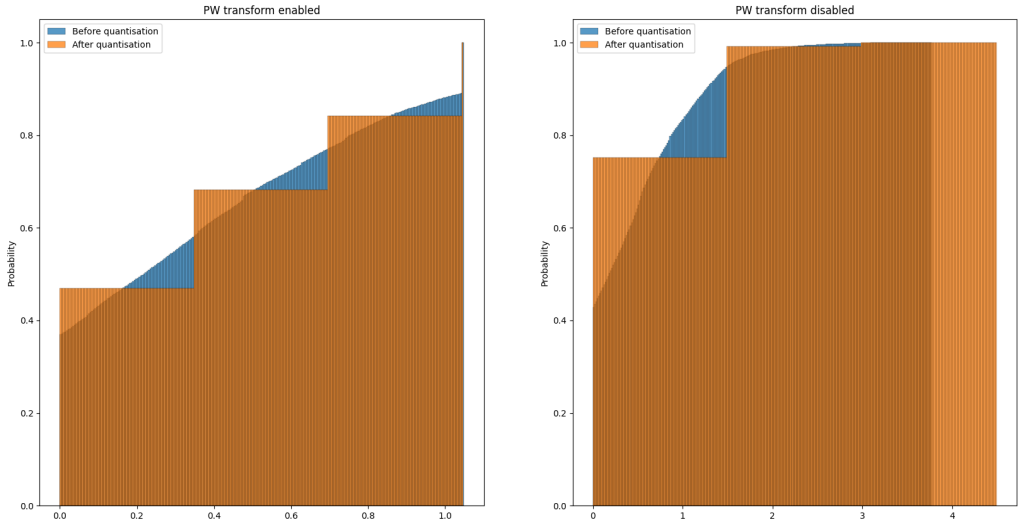


Figure 5.5: The cumulative distribution of the hidden layer activations, including values at zero.

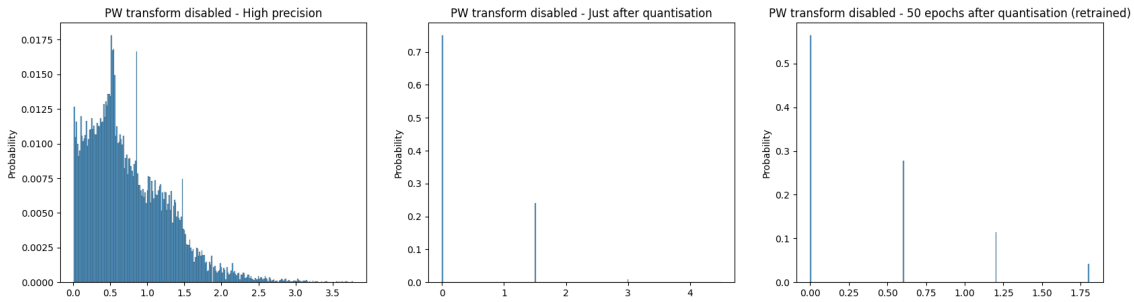


Figure 5.6: The probability density distribution of the hidden layer activations before quantisation, after quantisation, and after retraining.

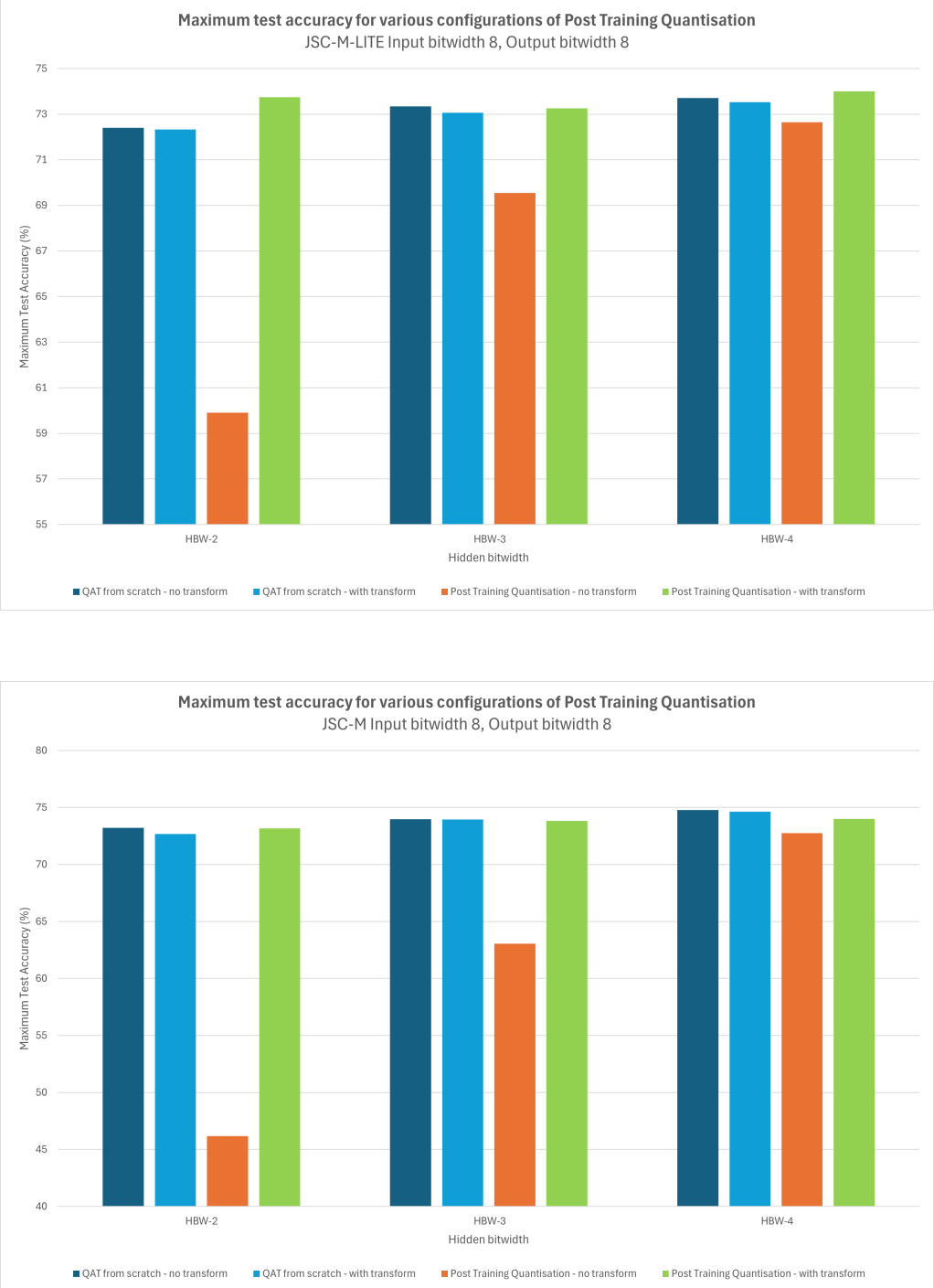


Figure 5.7: The accuracy achieved through Post-Training Quantisation for JSC-M-LITE and JSC-M networks across various hidden bit-widths.

5.7 Brevitas scaling implementation

As mentioned previously, companded quantisation requires a uniform quantiser between the forward and inverse transforms. In this project, uniform quantisation is implemented by the Brevitas python package as explained in Section 2.2.4. The scale implementation of the Brevitas quantiser can vary - in previous sections experiments were ran with scales calculated using runtime statistics. However, experiments showed that the maximum test accuracy of networks trained with the piecewise transform quantisation scheme improved when using a constant scaling implementation. The results are shown in Figure 5.8.

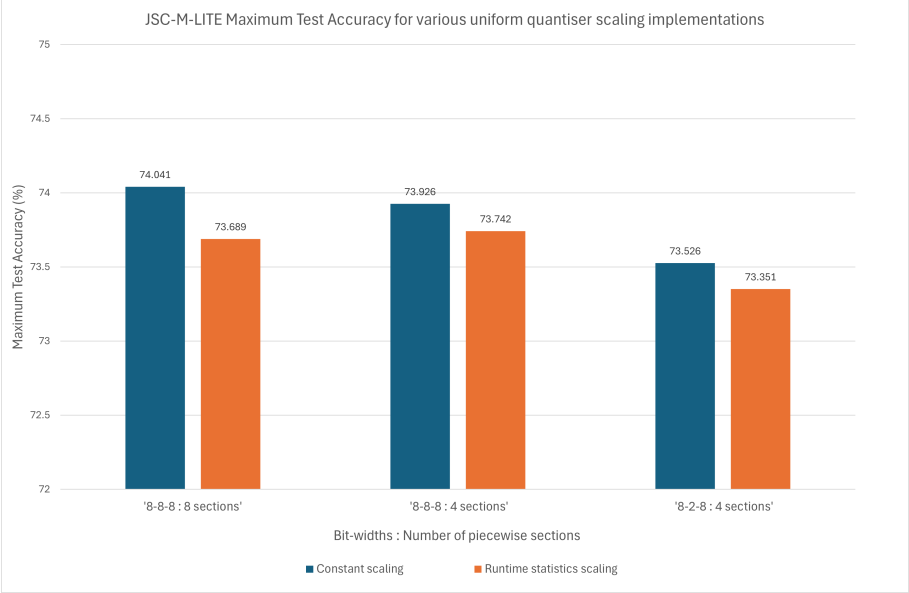


Figure 5.8: The maximum test accuracy of networks trained with the piecewise per-tensor transform for different scaling implementations, across various network configurations.

This may be because per batch, a scale based on runtime statistics changes. Therefore, the gradient function of the learnable transform parameters also changes. This could result in the transform parameters oscillating. Alternatively, it may be because with both the piecewise transform and the Brevitas scaling implementation, there is redundancy. This is because the slope of the piecewise transform effectively implements a scale for values in the domain of the corresponding piecewise section. Training with redundancy can result in worse accuracy because the redundancy introduces additional complexity and interdependence between parameters, making the optimisation process more challenging.

5.8 Number of sections as a hyper-parameter

With the automated construction of piecewise transforms, it is possible to experiment with various configurations. This capability is particularly advantageous for hyper-parameter tuning, which is important since the number of sections in the piecewise function is likely to influence the task performance of the network. This is because the number of piecewise sections used impacts the range of non-uniform quantisation schemes which can be represented. Therefore, by adjusting the number of segments, it is possible to tune the transform to achieve better results.

However, as the number of piecewise sections increases, the computational load of the transform also increases. This is because as mentioned in Section 4.3.3, the number of piecewise sections is a factor in the time complexity of the algorithm since each additional bound is another potential comparison operation which must be done. Therefore, the complexity of the forward piecewise transform with S sections is $\mathrm{O}(SNMK)$ or $\mathrm{O}(\log_2(S)NMK)$ when using binary search. However, binary search may yield slower performance for small values of S due to the additional overhead incurred. Furthermore, the memory complexity of the transform increases linearly with the number of piecewise sections, and hence the memory complexity is $\mathrm{O}(S)$. The total time complexity of the quantisation scheme is therefore $\mathrm{O}(2SNMK + C)$, since the inverse transform requires the same number of operations.

Figure 5.9 shows the network task performance for different numbers of piecewise sections, and Figure 5.10 shows the plotted test accuracy for a subset of experiments. The figure shows that the transforms with eight and four piecewise sections consistently perform better than the transform with 2 piecewise sections. This implies that the network benefits from the transform with more piecewise sections, which indicates that the distribution of data is such that a non-uniform quantisation is beneficial. Moving forward, the piecewise transforms were trained with 4 sections since 8 sections causes significant slowdown during training, and in some cases performs worse than 4 sections. Note that a transform with 1 piecewise section is equivalent to uniform quantisation.

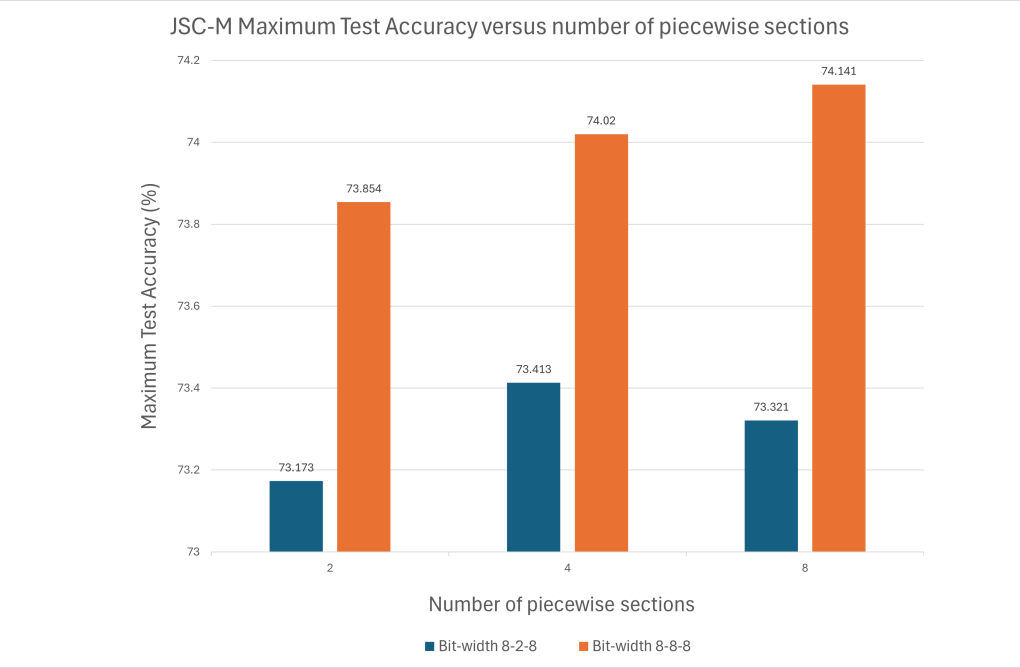


Figure 5.9: The maximum test accuracy achieved for networks trained under a QR approach for different numbers of piecewise sections. The results were collected from networks trained using the JSC-M architecture.

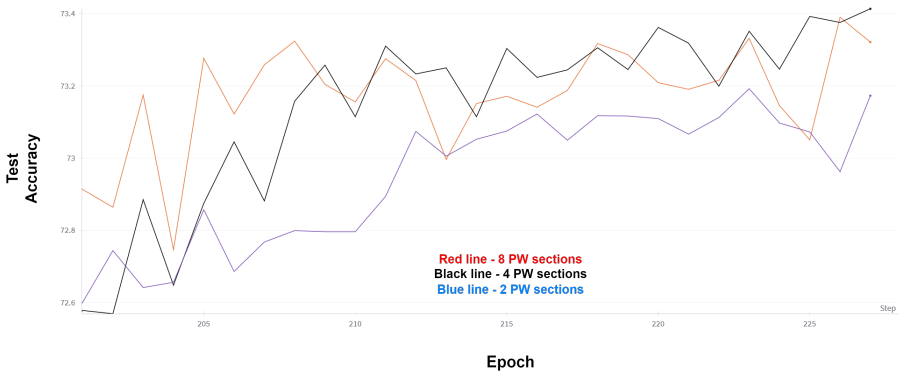


Figure 5.10: Test accuracy versus epoch under a QR approach for different numbers of sections in the piecewise transform. The results were collected from networks trained using the JSC-M architecture.

5.9 Varying the degree hyper-parameter

One benefit of PolyLUT is that, at inference, the network is able to implement polynomial basis expansion without the use of arithmetic units. This is because the polynomial basis expansion is absorbed into LUT mapping through the process described in Section 2.5. This basis expansion allows for complex relationships between features to be learned, with fewer layers [25].

The degree hyper-parameter is what controls the order of the expanded basis (as explained in Section 2.5). A polynomial basis expansion with a higher degree may result in more complicated distributions of activations, since higher order terms enable more complex functions to be mapped in a single layer. As a result, it may be the case that non-uniform distribution is more beneficial for networks with a higher degree hyper-parameter.

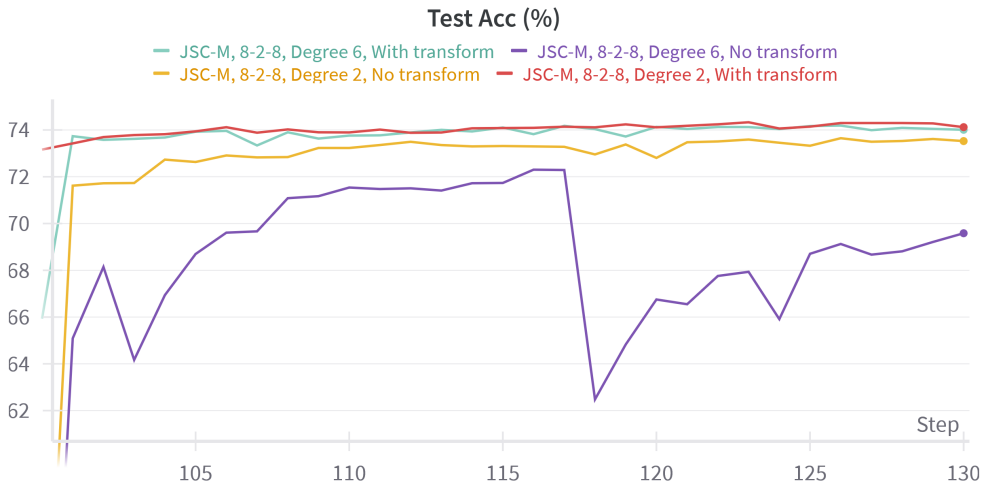
Figure 5.12 shows the accuracy for networks trained with a quantisation with re-training approach, across various bit-widths and degrees. The figure shows results for networks trained with the piecewise learnable transform quantisation scheme, and networks without. For the JSC-M-LITE network, the figure shows that both having a higher degree, and training with the transform is beneficial to test accuracy. Furthermore, the plots show that for some low bit-width configurations (8-2-8 and 8-2-2), training with the transform enabled is more beneficial than training with a higher degree.

The JSC-M plots show that the performance of the degree 6 network with the transform disabled is relatively poor in comparison to other configurations, particularly for the 8-2-8 and 8-2-2 bit-width configurations. This is highlighted in Figure 5.11a, which shows the test accuracy per epoch for the trials conducted with bit-widths 8-2-8 on the JSC-M network (again using a quantisation with re-training approach).

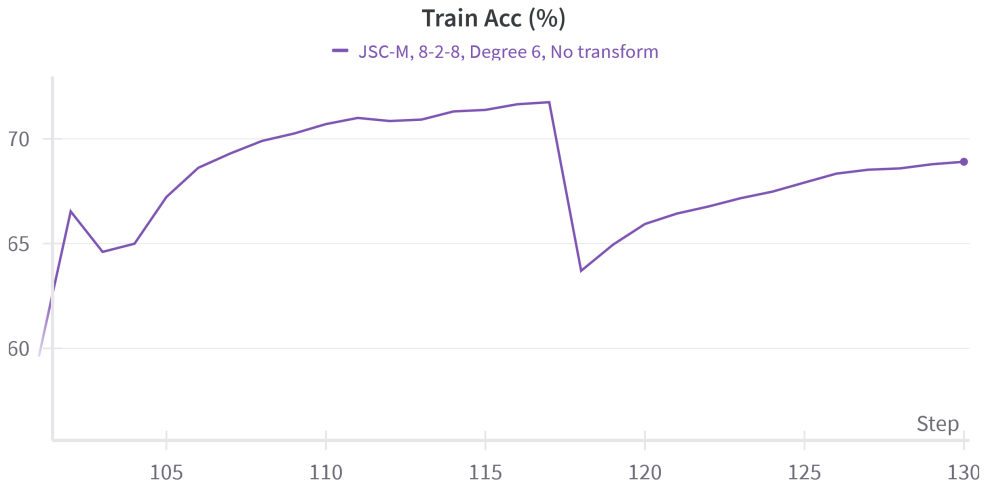
The graph shows that the test accuracy of the degree 6 network without a transform applied is significantly more volatile than the other configurations. Figure 5.11b shows the accuracy on the train dataset for this configuration which is also volatile.

As mentioned previously, a higher degree hyper-parameter induces more complex distributions of activations which may benefit from non-uniform quantisation. Furthermore, without the transform, the network relies on its weight parameters to reduce the negative impact of quantisation, through changing the distribution of the activations. The combination of these two factors may result in a difficult learning scenario, and hence cause the detriment to the accuracy observed in

the figures.



(a) The test accuracy for the 8-2-8 JSC-M network per epoch with and without the transform applied and across different degrees.



(b) The train accuracy of the JSC-M network with no transform, degree 6 and bit-widths 8-2-8.

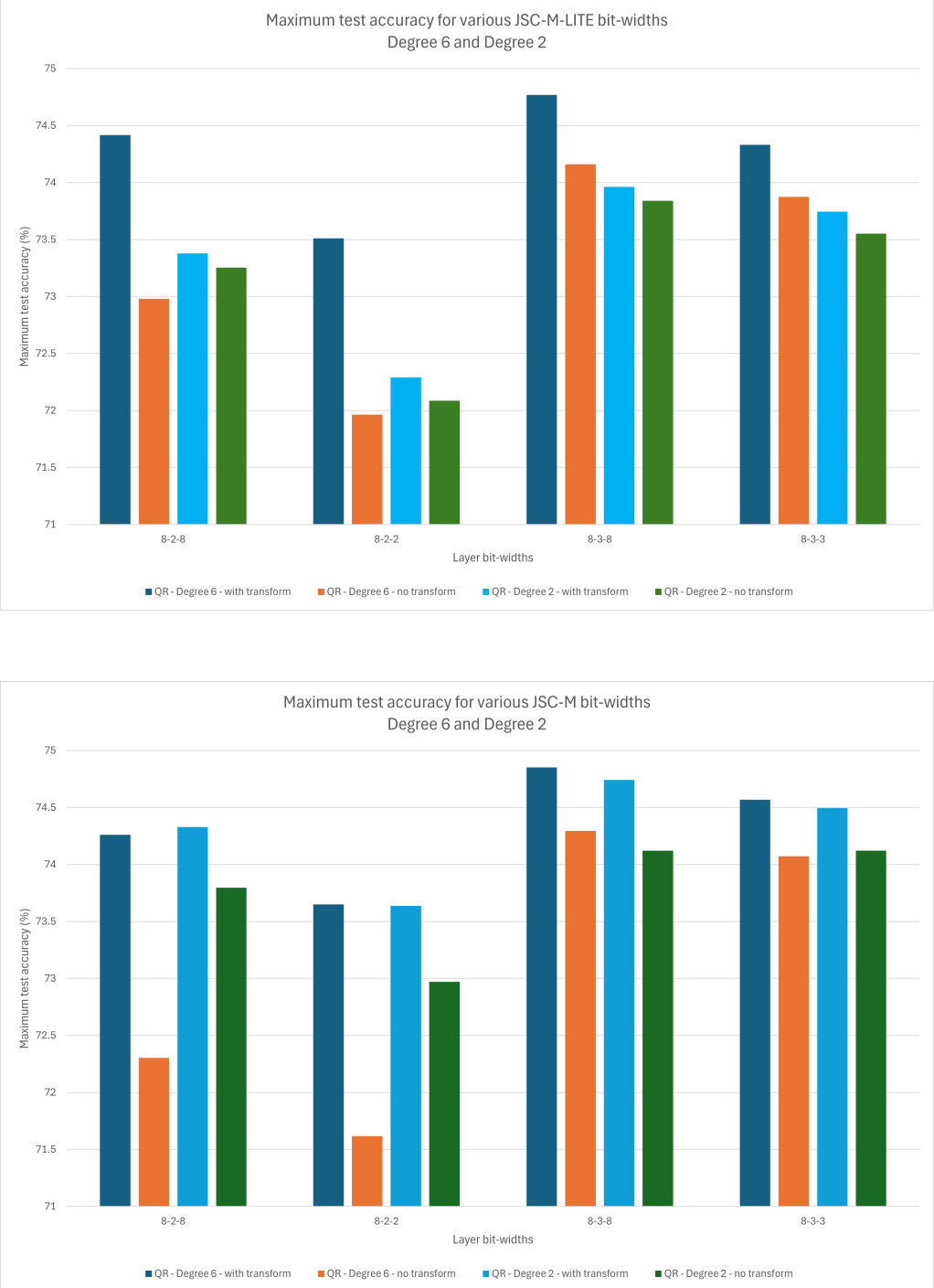


Figure 5.12: The accuracy achieved through quantisation with re-training for JSC-M-LITE and JSC-M networks across various bit-widths and degrees.

5.10 Per-neuron non-uniform quantisation

The piecewise transform scheme described in previous sections was applied per-tensor, meaning that all activations from a layer in a mini-batch were passed into the same transform with the same set of parameters. However, this means that the non-uniform quantisation learned is based upon the aggregate distribution of the activations from each neuron in a layer. This is potentially limiting, because neurons may have largely different distributions, and hence benefit from different quantisation schemes. To test this theory, a JSC-M-LITE network was trained at 8-bit precision for 200 epochs (since this is when test accuracy no longer seems to improve), and the distribution of the activations was recorded. This data was then plotted as a histogram to model its probability density function (PDF).

Figure 5.13 shows the distribution of a single neuron's activations in the hidden layer. Figure 5.14 shows the distributions of each neuron in the hidden layer, excluding zeros. The left plots show histograms with 256 bins (2^8) and the right plots show histograms with 4 bins (2^2) to emulate 8-bit and 2-bit uniform quantisation respectively[1].

The plots show that the distributions of the neurons at both 8-bit and 2-bit representations vary significantly depending on the neuron. Additionally, the range changes significantly per plot. Both of these points imply that using the same quantiser, meaning the same quantisation levels and clipping boundaries, is sub-optimal since the distribution of the neuron activations are significantly different.

Therefore, the piecewise transform quantisation scheme was developed to apply a different transform to each neuron's activations, meaning for a layer with 32 neurons, there would be 32 different transformations. The joint application of these transforms is referred to as applying the *per-neuron piecewise transform* in this report.

The memory complexity of this design is larger than the per-tensor transform, since for each neuron there is a different set of learnable parameters. The memory complexity is now $\mathrm{O}(WS)$, where S is the number of piecewise sections, and W is the number of neurons in the layer (the *width* of the layer). The time complexity remains the same since the total number of operations is unchanged.

Although the time complexity is the same as for the per-tensor transform, training time was

[1] In actuality, 2-bit representation only allows for three bins for values excluding zeros, since one quantisation level is always at zero (Section 3.5). However these plots are just to illustrate roughly how the distributions of each neuron varies, so this is not a significant problem.

significantly increased. This is because the per-neuron transform was executed sequentially, by looping through the activations of each neuron in the input tensor.

Unfortunately, PyTorch did not have the appropriate methods to enable easy vectorisation of this loop, which meant that an independent solution was required. However, due to time constraints during the development phase of this project, no such solution was found. As a result, the computational load during training made tuning this quantisation scheme infeasible.

Results are shown in Table 5.1 and the scheme is compared against others in Figure 6.1. The figures show that the piecewise per-neuron transform performs slightly worse than the piecewise per-tensor transform. Theoretically, the set of quantisation schemes representable by the per-neuron piecewise transform, is a super-set of those representable by the per-tensor piecewise transform. Therefore, with the correct training hyper-parameters, networks trained with this scheme should be able to achieve the same task performance as those trained with the per-tensor transform.

However, finding the correct hyper-parameters requires running a large number of trials, which is infeasible due to the computational requirements of the per-neuron piecewise transform. Because the results found in these experiments indicate that the benefit of a per-neuron non-uniform quantisation is not significant, the decision was made not to run further trials.

Number of piecewise sections	Bit width	Max test accuracy (%) JSC-M-LITE	Max test accuracy (%) JSC-M
4	8-8-8	74.032	75.045
4	8-2-8	73.772	74.291

Table 5.1: Maximum test accuracy for the per-neuron piecewise transform

5.11 Per-neuron uniform quantisation

The histogram plots in Section 5.10 show that the range of activations varies greatly per-neuron. As a result, using a single uniform quantiser is inefficient because the quantisation scheme will either have significant errors due to clipping, or some neurons will never hold values in certain quantisation levels.

However, this issue is alleviated through the usage of per-neuron uniform quantisation, where each neuron has its own uniform quantiser. To implement a per-neuron uniform quantisation, a linear transform is applied per-neuron before the activation tensor is inputted to the uniform quantiser. This transform is referred to as the *simple transform* in this report, and is defined as follows:

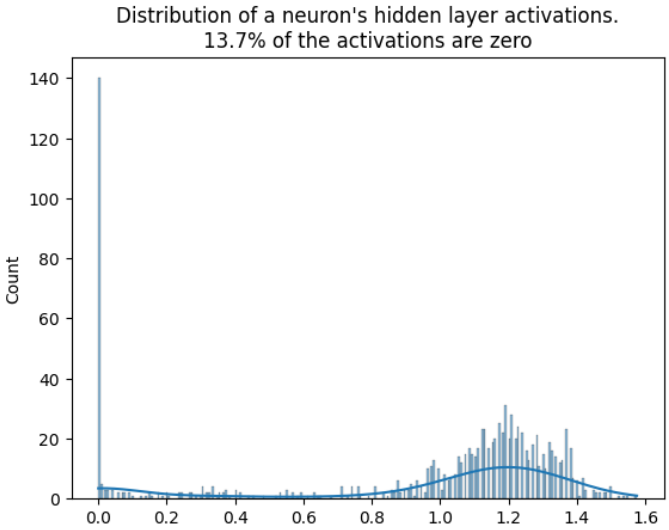


Figure 5.13: A single hidden layer neuron's PDF. A large portion of the distribution is at zero due to the ReLU activation function.

$$T(x) = ax + c \tag{5.1}$$

$$T^{-1}(x) = \frac{x - c}{a} \tag{5.2}$$

Both a and c are learnable parameters. This is an implementation of fixed-point quantisation as explained in Section 2.2.2, but with a learnable bias (not forced to be zero). The computational load of this network during training is minimal since there are no comparisons required. The time complexity of the quantisation scheme is $\mathrm{O}(4NMK + C)$ since there are 4 operations per value in the tensor. The memory complexity is $\mathrm{O}(2W)$ where W is the width of the layer as in Section 5.10. Figure 5.15[2] shows the results for the per-neuron simple transform for various learning rates and bit-widths.

5.12 Reducing the learning rate

The first enhancement to the learnable quantisation involved experimenting with different learning rates specifically for the transform parameters. Since the transform slope parameters are initialised to 1.0, theoretically, networks with the transform enabled are unlikely to perform worse than those without it. This is because if uniform quantisation is more effective, the parameters are motivated

[2]Note that TLR is an acronym for Transform Learning Rate

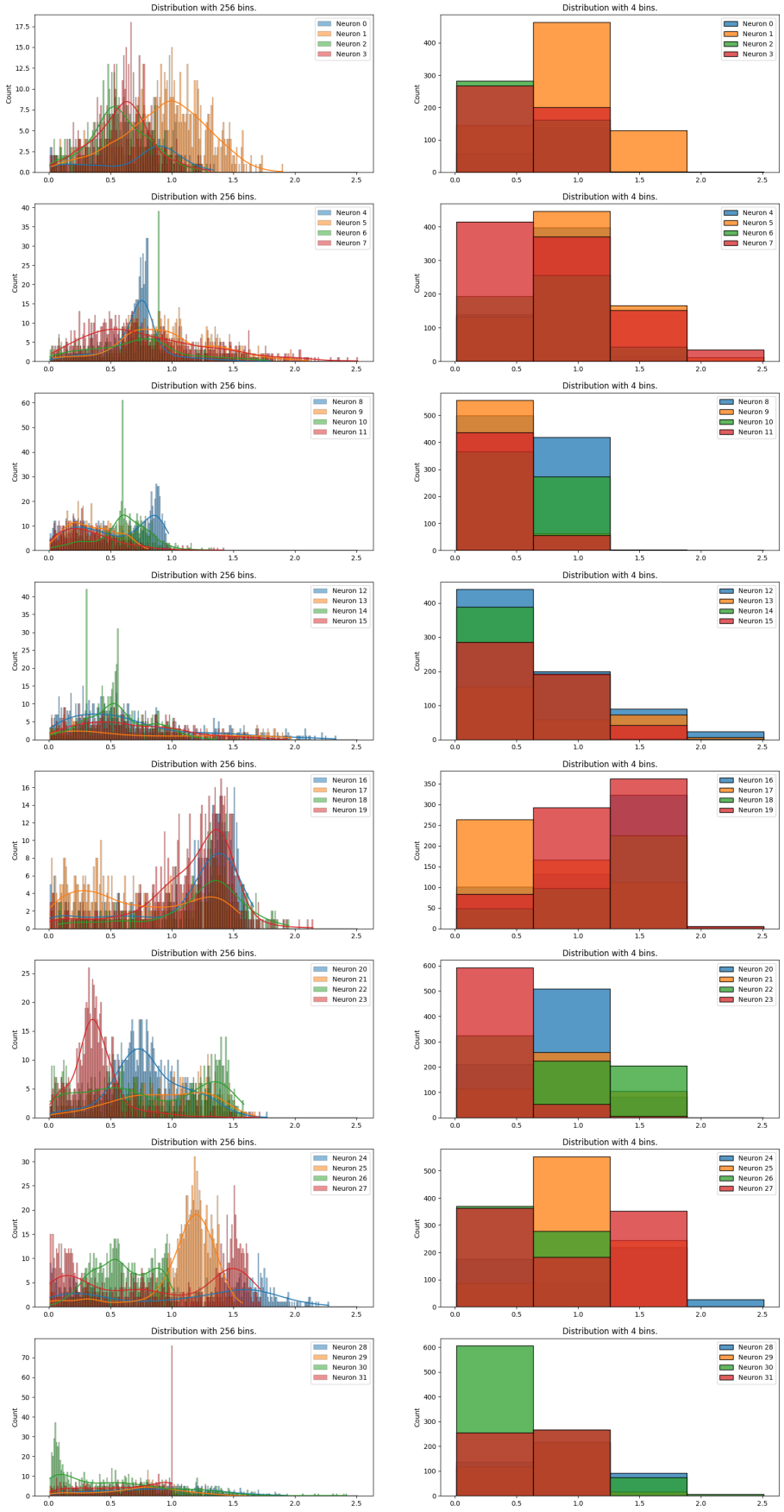


Figure 5.14: The distributions of the hidden layer's activations.

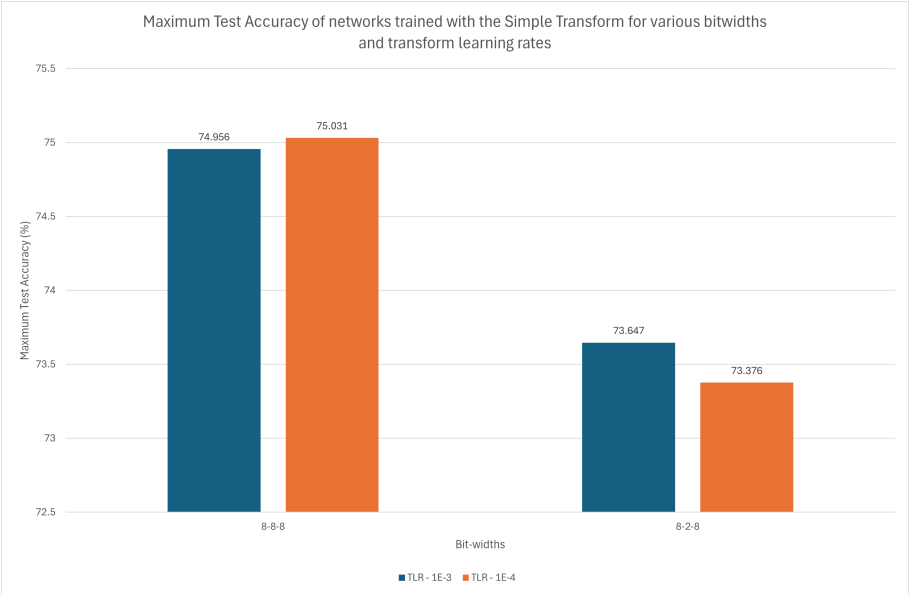


Figure 5.15: The maximum test accuracy of the per-neuron simple transform for various bit-widths and transform LRs.

by the loss to remain at 1.0. However, when trained with the same learning rate as the other network parameters (weights), the observed performance was worse. This result could be due to the transform parameters oscillating around their optimal values.

It is plausible that the transform parameters may benefit from a different learning rate because the transform parameters are far fewer in number than the input tensor, causing each parameter to accumulate gradients from many inputs. Therefore, the learning rate used for the network weights might not be optimal for the transform parameters. Experiments with different learning rates were conducted on the JSC-M-LITE network and the JSC-M network, and the results are shown in Figure 5.16.

The gains made by tuning the learning rate are relatively small when maximum test accuracy is compared. However, the transform learning rate has a big impact on the stability of network task performance, as shown by the fluctuations in Figure 5.17. The figure shows that although for some epochs the test accuracy is similar, the transform with the higher learning rate is more volatile and has large downward spikes at certain epochs. These results show that if the transform parameter learning rate is too high, the quantisation scheme can change significantly per epoch which results in detriment to the accuracy. These results are examples of the disadvantages of using maximum test accuracy as the sole metric for network task performance.

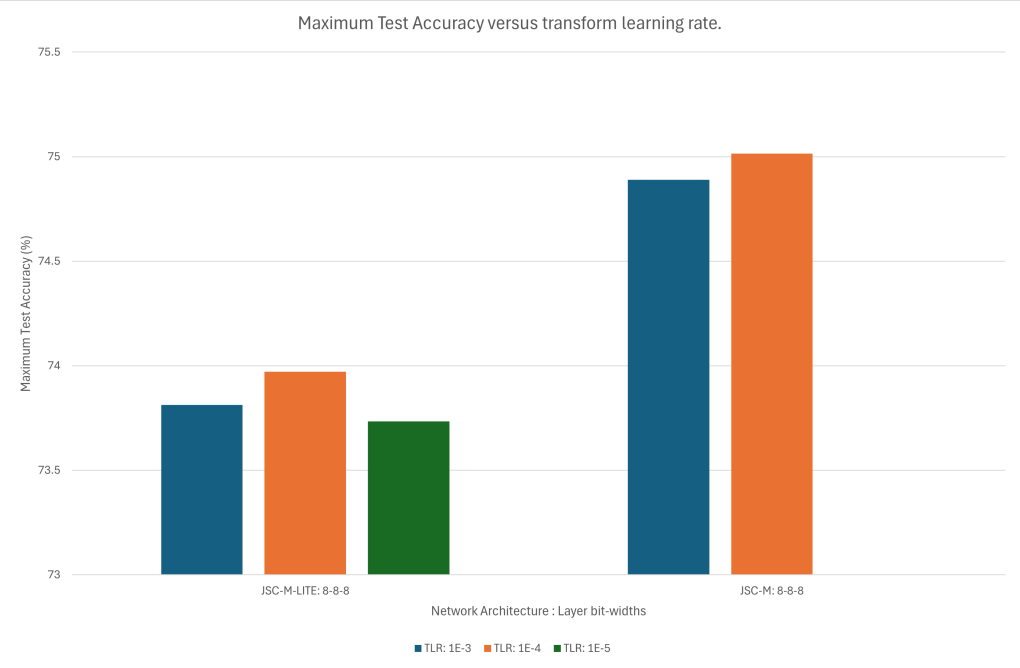


Figure 5.16: Maximum test accuracy versus transform learning rate for various configurations.

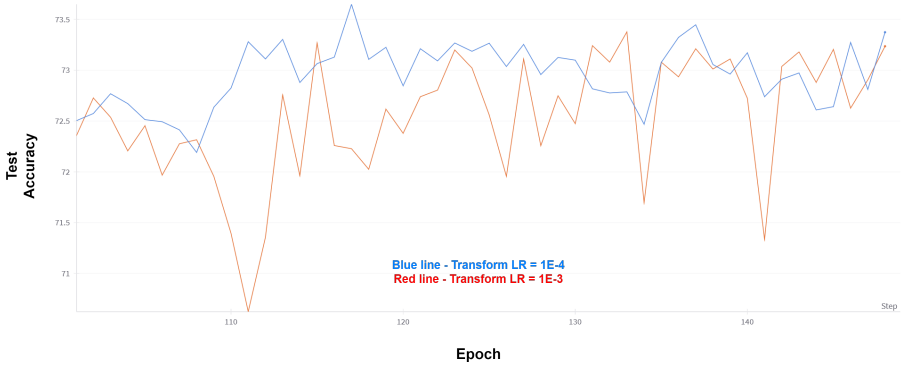


Figure 5.17: Test accuracy for the JSC-M model with bit-widths 8-2-8 for various transform learning rates.

5.13 Histogram equalisation

Histogram equalisation is a technique mainly employed in image processing to enhance the contrast of an image. The method works by adjusting the intensity distribution of the image's histogram, redistributing the intensity levels to span the entire range of possible values more uniformly. The primary aim is to achieve a histogram that is as flat as possible, meaning each intensity value occurs with equal probability.

The motivation behind applying histogram equalisation before uniform quantisation is that the distributions of activations will occupy each quantisation level with even counts. This means that the amount of aliasing[3] that occurs post quantisation is minimised. The hope is that by reducing aliasing, the accuracy of the network improves.

The process of histogram equalisation involves several steps. Initially, the histogram of the input is calculated. Following this, the cumulative distribution function (CDF) of the histogram is computed. Inputs are passed through the histogram's CDF to obtain their corresponding percentile. Following this, the percentile is then passed into an inverse uniform CDF which returns the corresponding equalised value for the original input.

The CDF can be implemented through a linear piecewise approximation, which allows the equalisation process to be done through the current piecewise transform setup.

After the equalisation process is complete, the tensor is passed into the uniform quantiser, and then into the inverse transform. The inverse transform undoes the equalisation process to return the post-quantisation approximation of the original distribution.

To test histogram equalisation, it was implemented on the hidden layer of a trained JSC-M-LITE network. To obtain an approximation for the hidden layer's CDF, the network was trained at 8-bit precision until test accuracy no longer improved, at which point one batch of activations was collected.

This was then used to approximate the CDF using an N-section piecewise linear function. The CDF of this data was divided into $N-1$ equal parts, and the coordinates of the boundaries were obtained using interpolation techniques. These coordinates were connected which formed the linear piecewise approximation of the CDF. Figure 5.19 shows the equalisation process, and the CDF of the final output.

[3] In this context, aliasing is when two or more different inputs map to the same quantisation level and hence result in the same output post-quantisation.

To test the results of the histogram equalisation, networks were trained with the transform disabled, at 8-bit precision for 200 epochs. Then the hidden layer was quantised using the per-neuron piecewise transform scheme. The transform parameters were initialised to values such that the transform implements histogram equalisation per-neuron. Figure 5.18 shows the accuracy achieved through this technique compared to the same network with no quantisation transforms enabled.

The figure shows that histogram equalisation was unable to achieve an improvement over the baseline network, and therefore was ineffective. Experiments showed that mean absolute quantisation error and mean squared quantisation error increase under this scheme (compared to uniform quantisation without equalisation). The quantisation error likely increases because firstly, the CDF is approximated. Secondly, when the approximate inverse CDF is applied, the quantisation errors may be enlarged. This test indicates that the benefit of reducing aliasing is less than the detriment incurred by increasing the absolute quantisation error.

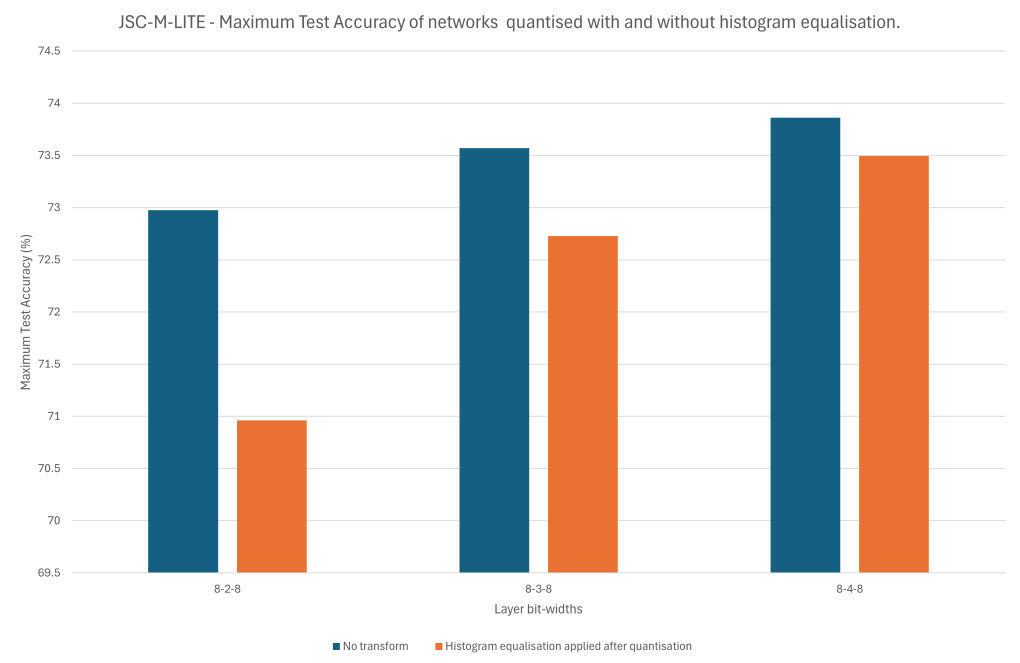


Figure 5.18: The maximum test accuracy of the JSC-M-LITE network when quantised using histogram equalisation. The results were measured after 20 epochs of retraining. The figure includes the results of networks trained with the quantisation transforms disabled (and hence no histogram equalisation).

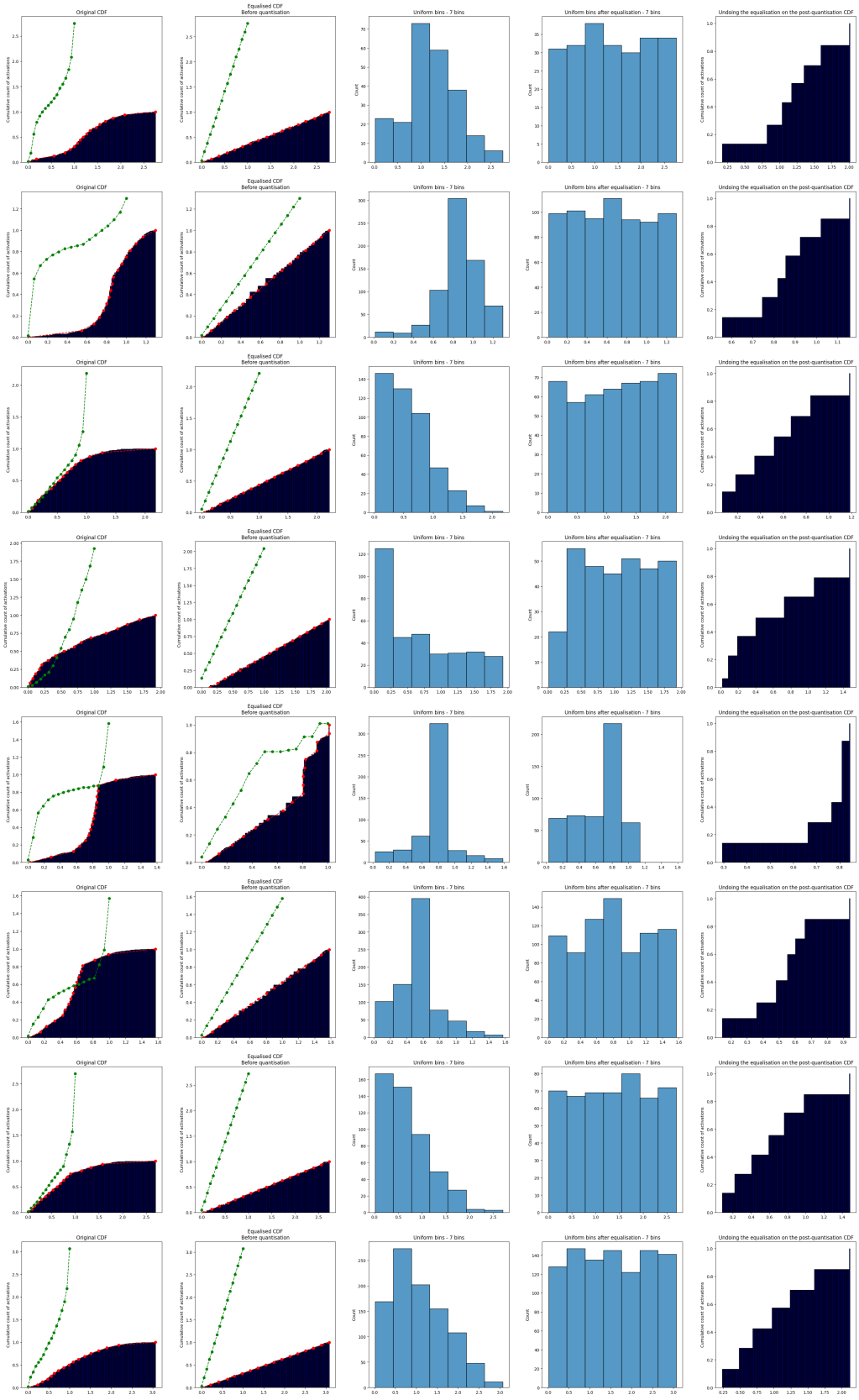


Figure 5.19: The histogram equalisation process on a hidden layer's activations. The CDF is approximated using a linear piecewise function. The histograms named "Uniform bins" emulate 3 bit uniform quantisation applied to the activation tensor with and without equalisation. The final plot shows the post-quantisation CDF after being passed into the inverse transform which undoes the equalisation.

6

Overall results, Evaluation and Conclusions

Contents

The primary objective of this project was to evaluate the effectiveness of different number representations in a setting where the cost of all representations were roughly equal. As mentioned previously, in the case of PolyLUT and NeuraLUT [25][39], the implementation of non-standard representation is absorbed into the mapped LUT. Therefore, the resource cost is upper bounded by the theoretical maximum size of the LUT generated.

Implementing non-standard representations results from using non-uniform quantisation. To achieve this, various specific quantisation schemes, such as logarithmic quantisation, were tested. Ultimately, a software module was developed to implement a learned non-uniform quantisation scheme. This module uses a learned piecewise transform and a uniform quantiser to implement an arbitrary quantisation scheme through the concept of companded quantisation. The effectiveness of

the learned quantisation scheme depends on the number of piecewise sections in the transform. In other words, with enough piecewise sections, the module can implement any quantisation scheme.

6.1 Results

Overall, various quantisation schemes were developed, and each scheme was tested across many different scenarios. The full results for the quantisation from 8-bit to lower precision can be seen in Table A.1 and Figure 6.1[1]. These results were all collected using a QR approach, since this proved to yield the best test accuracy in each scenario.

The results for the JSC-M and JSC-M-LITE architectures show some common trends. For the JSC-M-LITE tests, the best performing network was the 8-3-8 configuration with degree 6 and the piecewise per-tensor transform enabled. This network, was expected to perform well, but the fact that it performs better than the 8-4-8 configuration with degree 2, implies that more hidden bit-width was less beneficial than a higher degree. In the JSC-M tests, this is not the case. Although the higher degree network performs well, having more hidden bit-width is more beneficial. This is likely due to the fact that there are two hidden layers in the JSC-M network, which means that hidden layer quantisation error is more detrimental.

The results also reveal a significant difference between networks with the transform enabled and those without. This suggests that the network benefits from learned non-uniform quantisation. Additionally, the 8-3-3 network with the transform enabled performs better than the 8-3-8 no transform network in both degree 2 and degree 6 cases. This is notable because it indicates that the transform allows for a reduction of 5 bits in the output layer. This is particularly advantageous for PolyLUT networks, as resource costs in this regime scale exponentially with bit-width (evidenced by the resource analysis shown in Section 6.1.2).

For both architectures and across all the bit-widths, networks trained with the piecewise per-tensor transform consistently performs the best. In the cases where the hidden and output bit-widths are reduced to below 3 bits, increasing the degree parameter results in worse performance. This effect is far more significant in the networks trained without a transform. This result was explored in Section 5.9 - networks trained with a higher degree seem to exhibit higher volatility at the lower bit-widths when trained without the transform.

Both of the per-neuron schemes performed worse than the per-tensor scheme. In theory, the

[1]Note that in this figure *Deg* is the degree hyper-parameter the network was trained with.

piecewise per-neuron transform is capable of implementing the same quantisation schemes as the piecewise per-tensor transform. The reasons for its performance not matching that of the piecewise per-tensor transform is explored in Section 5.10.

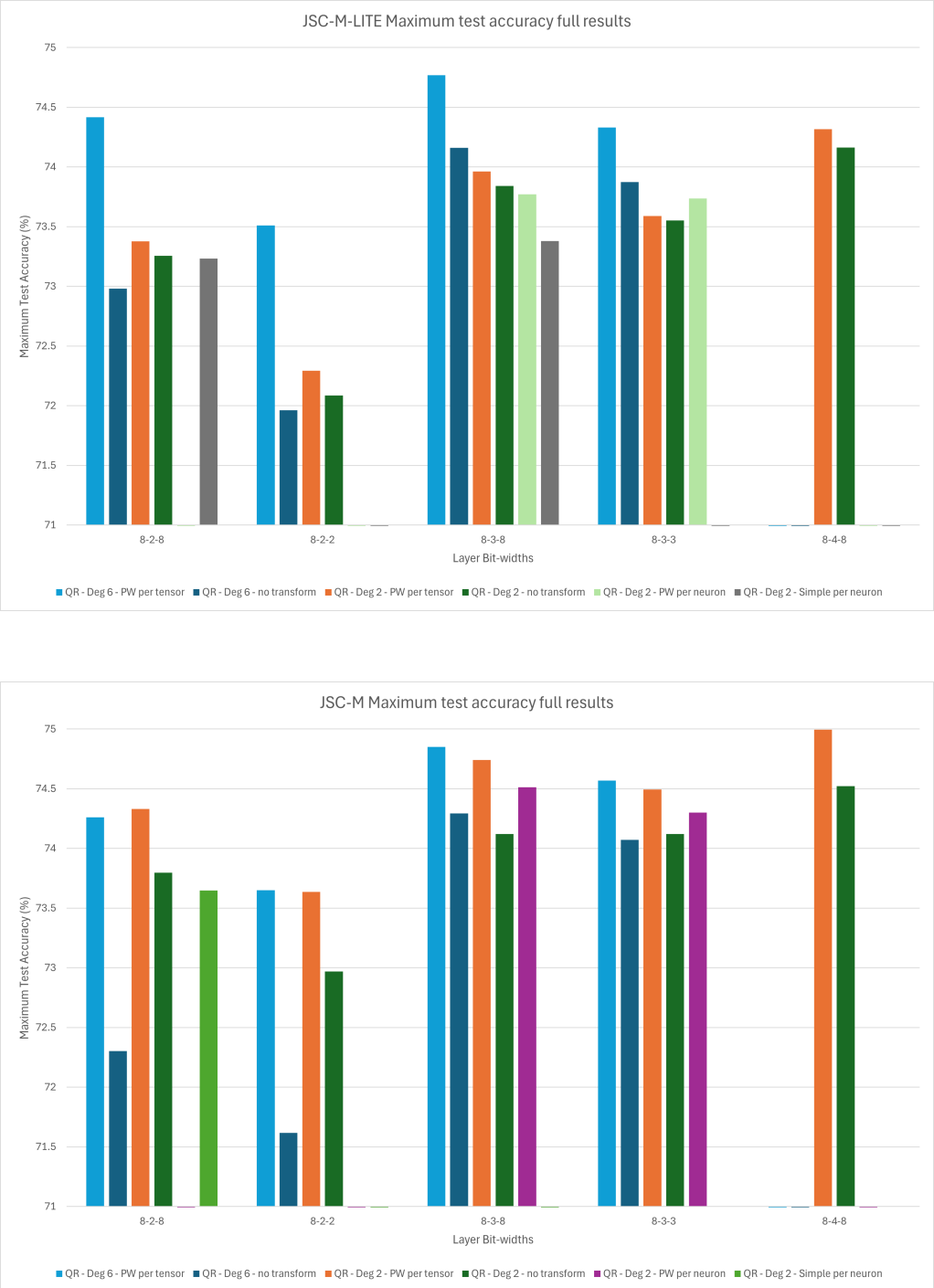


Figure 6.1: The maximum test accuracy across various network configurations. All results were collected using a QR approach. 'PW' denotes networks trained with the piecewise transform

6.1.1 Comparison with PolyLUT

After 200 epochs of training, the original PolyLUT configuration for the degree 2 JSC-M-LITE network achieved a maximum test accuracy of 71.858% with 3-3-3 bit-widths. For this bit-width configuration, a QAT approach offered the best test accuracy.

With the transform enabled, the same network achieved a slightly higher maximum test accuracy of 72.012%. Although the difference in maximum test accuracy is relatively small, Figure 6.3 shows that the network trained with the transform consistently performed better, and in some cases the gap in test accuracy was significantly larger. The gap in performance is likely smaller due to the bottleneck caused by the low precision of the input layer. The reasons for this are explored in Section 5.4.

Figure 6.2 summarises the results of each quantisation scheme when trained on one of the original PolyLUT configurations: JSC-M-LITE with 3-3-3 layer bit widths and degree 2.

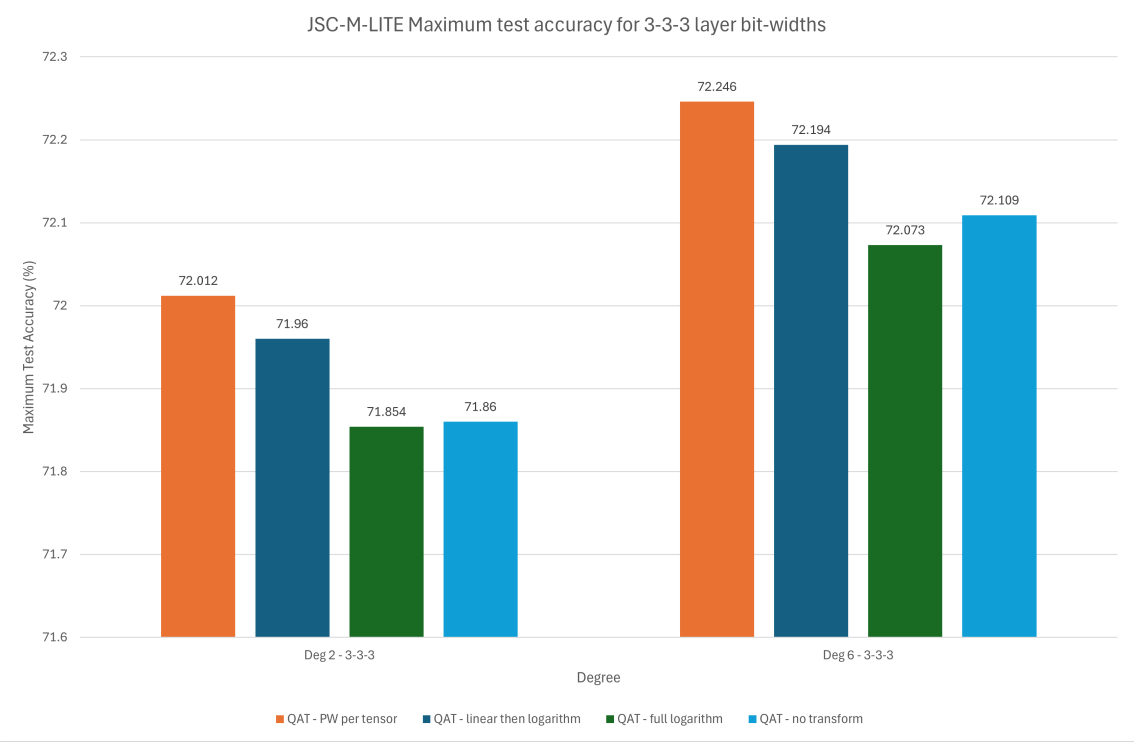


Figure 6.2: The best test accuracy achievable for each quantisation scheme on a JSC-M-LITE network with 3-3-3 layer bit-widths.

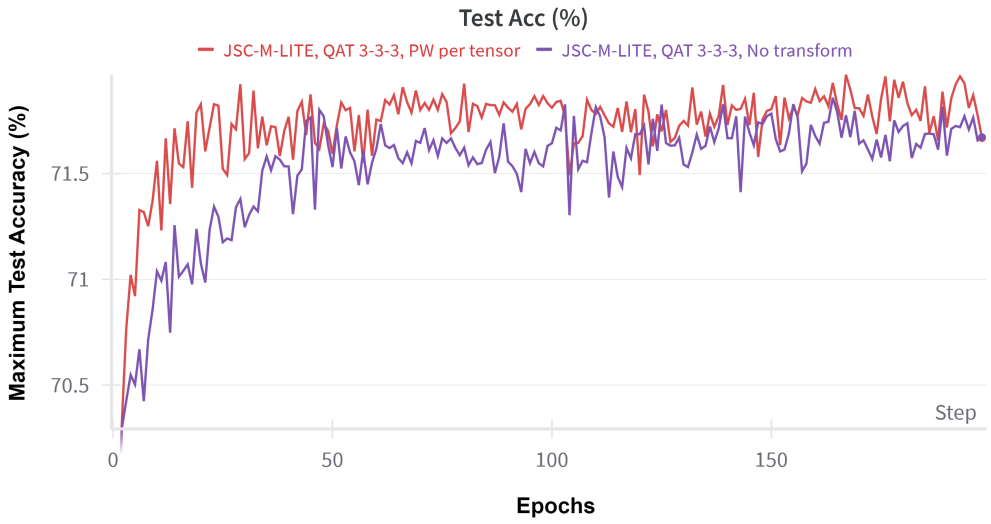


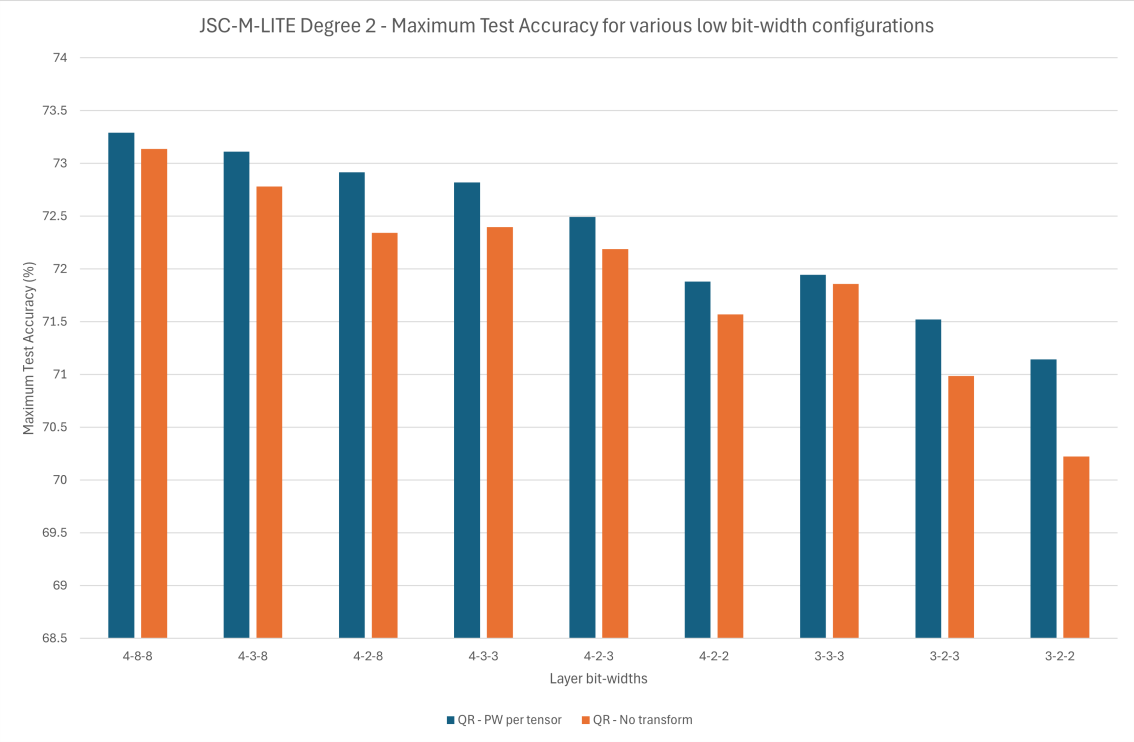
Figure 6.3: The test accuracy of JSC-M-LITE network when trained from scratch with bit-width 3 for each layer. The blue line represents the performance under uniform quantisation, and the red line represents the performance under non-uniform quantisation implemented via the piecewise learnable transform scheme.

6.1.2 Benefits achieved

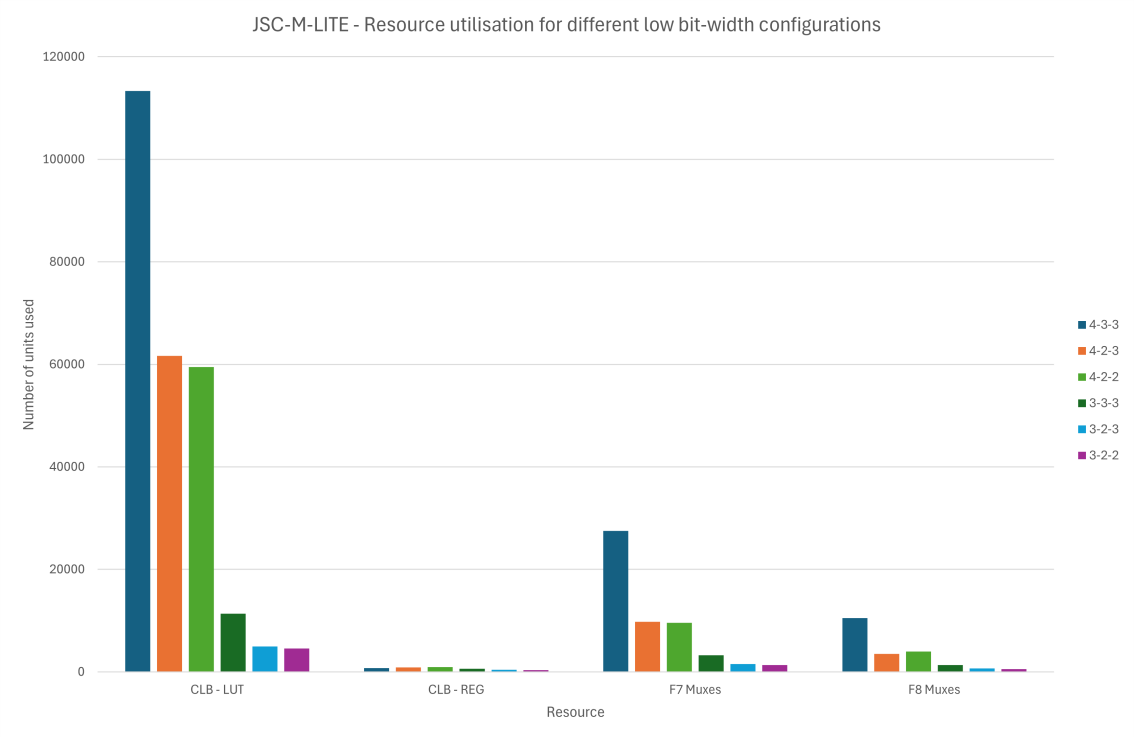
Further tests were conducted on various low bit-width configurations, since at these configurations FPGA synthesis was possible. The accuracy achieved in these configurations is shown in Figure 6.4a. The resource utilisation for each of these bit-width configurations is shown in Figure 6.4b. Resource utilisation was only able to be collected for networks trained with the transform disabled. However, it is expected that whether the transform is enabled or not has low impact on the resource utilisation (for the reasons explained in Section 6.2.2).

The results shown in Figure 6.4a continue the trend seen in the 8-bit tests (Figure 6.1). Enabling the piecewise linear transform consistently yields non-negligible accuracy improvements. Furthermore, like the 8-bit tests, these results also show that the addition of the transform can yield improvements equivalent to adding a bit to a layer's activations. Saving one bit results in significant reductions in resource utilisation, evidenced by the results in Figure 6.4b. For example, the 4-2-8 network trained with the transform enabled achieves greater accuracy than the 4-3-8 network with the transform disabled meaning the hidden layer bit-width can be reduced by one, without accuracy loss. Similarly, the 4-2-3 network with the transform enabled performs better than the 4-3-3 network with the transform disabled. The 4-3-3 and 4-2-3 resource consumption shown in Figure 6.4b show that reducing the hidden bit-width by one results in a large resource reduction (the CLB usage is approximately reduced by a factor of 2). This is a significant reduction,

since it means for the same resource usage, you can have double the neurons.



(a) Maximum test accuracy for various low bit-width JSC-M-LITE networks.



(b) Resource consumption for various low bit-width JSC-M-LITE networks.

The increase in test accuracy achieved via the piecewise learnable transform allows for fewer bits to be used while maintaining the same level of accuracy. The JSC-M-LITE network, when trained without the transform enabled and quantised to a 6-3-3 bit-width configuration achieved a maximum test accuracy of 73.331%, and 73.551% with 8-3-3 bit-widths (as shown in Table A.1). In contrast, the same architecture, when quantised to 6-3-3 and trained with the transform enabled, achieved 73.643% - a higher accuracy than the 8-3-3 network trained with the transform disabled.

The input layer of the JSC-M-LITE network has sixteen neurons. Each neuron has a fan-in of 4, which means that it is connected to four inputs. With four inputs and eight bits, the size of LUT input for one neuron is $4 \times 8 = 32$. Therefore, the sum of the number of entries in the lookup tables for all sixteen neurons is $2^{32} \times 16 = 2^{36}$. In comparison, with six bits the number of entries is 2^{28}. The number of entries in the logical look-up table correlates with the size of the physical LUT generated, meaning that the resource consumption for the input layer has decreased by roughly 256 times with zero accuracy reduction. The network can achieve better performance with two fewer bits in the input layer, representing a significant resource reduction due to the exponential scaling of PolyLUT-style deployments.

6.2 Evaluation

6.2.1 Inefficient testing methodology

The project encountered several challenges, primarily due to the complexity and depth of understanding required about neural network quantisation. One significant issue was the lack of a robust testing regime early in the project. This oversight led to repeated experiments, which were time-consuming and inefficient.

Furthermore, inefficient testing is particularly detrimental to progress in this project due to the time required to run each trial. As in many machine learning projects, network training is a lengthy process. The need for extensive training time is compounded by the iterative nature of machine learning research. In the case of this project, progress was often blocked whilst waiting for simulation results, since each development made was in response to some observed behaviour.

GPU resources were shared with other members of the department, which meant that the number of simulations started at a given time was limited. Shared computational resources introduced a slight bottleneck in the experimentation process. This limitation meant that scheduling and

prioritisation of experiments was required to maximise the productive use of the available GPU time.

Network training for the majority of tests was conducted overnight, which meant that to keep up progress, testing was done concurrently with analysis. Running training sessions overnight allowed utilisation of otherwise idle computational resources effectively. However, this also introduced a delay in feedback, as results from overnight runs were only available the following day. To mitigate this delay and maintain steady progress, it was important to analyse the previous day's results and set up new experiments promptly.

In other words, there was a pipeline. This pipeline was essential for maintaining momentum in the project. It involved a continuous loop of training, testing, analysing results, and setting up new experiments. The efficiency of this pipeline directly impacted the progress of the project. Any delays or inefficiencies in one stage of the pipeline would cascade, slowing down the entire process.

Regrettably, the effectiveness of this testing methodology was realised through difficult experience. In the early phases of the project, the testing regime was less systematic, leading to a higher number of unproductive experiments. As a result, progress was considerably slower during these initial stages.

6.2.2 Difficulties when collecting results

When evaluating the effectiveness of a quantisation scheme, or an improvement made to it, the ideal approach is to test all combinations of control variables. This allows one to rule out potential interdependencies and gain a clear understanding of the impact of the change being tested. However, as mentioned in the previous section, testing was time-consuming. Consequently, when testing a particular improvement, it was necessary to select a search space. Often, this meant only varying one or two control variables. This is disadvantageous because it restricts the ability to fully grasp the impact of the changes and may overlook important interdependencies between variables.

The reliance on maximum test accuracy as the sole metric for network task performance had disadvantages. This metric does not account for volatility in test accuracy per epoch, which is suboptimal because the test set is merely an estimation of the population distribution of the dataset. Consequently, networks with higher but volatile test accuracy might not perform consistently in practice. Incorporating additional metrics that measure stability and consistency across epochs, such as a moving average of test accuracy, would provide a more reliable assessment of the network's

performance.

The project suffered from a lack of synthesis results for two reasons. Firstly, the enumeration process required to develop the hardware description of the network was infeasible for network's with one or more 8-bit layers. This is because the enumeration space required over one terabyte of GPU memory to tabulate. This also means that it would be infeasible to deploy on hardware. Regardless, most testing was done on configurations with such bit-widths, as they showed the largest improvements when trained with the transform enabled, compared to without. This was important because it meant that trends were more easily observable, which allowed for better guidance during the development.

Secondly, delays in the development resulted in a lack of time to run many synthesis flows. Collecting synthesis results was not prioritised since experiments in Chapter 4 combined with results from PolyLUT [25] and NeuraLUT [39] indicated that changing the quantisation scheme was likely to have a low impact on resource utilisation. However, on deeper networks there is more motivation to obtain synthesis results because the quantisation schemes are implemented per layer, therefore, as the number of layers increases, the effect on the network's resource consumption might increase. None of the architectures tested in this project were particularly deep but this may be an important consideration for future work.

6.2.3 Task prioritisation

A key lesson is the importance of prioritising tasks and transitioning from less effective methods more swiftly. In hindsight, focusing earlier on developing the linear piecewise transform scheme and conducting a thorough hyper-parameter search would have yielded better results. The correct learning rate and the number of sections for the transform improved accuracy significantly. Tuning these parameters earlier would have sped up development. Additionally, developing versatile software modules ahead of time would have allowed faster updates to the simulation environment, thereby reducing the time spent on debugging and adding features.

6.2.4 Dataset limitations

Testing was limited to a single dataset and small network architectures. This constraint is significant because the effectiveness of quantisation can vary greatly with different data distributions and network structures. Evaluating the quantisation methods across multiple datasets and a wider

variety of architectures would have provided a more thorough and generalisable assessment. Understanding the interplay between data distributions, network complexities, and quantisation is important for extending findings to broader applications.

The results in Section 5.6 suggested that networks could implicitly implement non-uniform quantisation by adjusting the distribution of activations to minimise quantisation error, since quantisation error is likely to increase the loss of a network. If this is the case, through SGD, the weight parameters learn values which strike a balance between minimising the quantisation error of the layer, and learning effective features. From this perspective, the advantage of learnable non-uniform quantisation is that the network can reduce the quantisation error without changing the weights. Therefore, the weights may be able to learn better features.

However, this theory implies that the learnable non-uniform quantisation scheme can be replaced by increasing the number of weights in the layers. Therefore, the advantage of learnable non-uniform quantisation may be most present on networks which are not over-parameterised. To test this, it would have been beneficial to run trials on a wide range of network architectures, and different datasets.

6.3 Further work

Building on the findings and limitations of this project, several avenues for further research are recommended.

Extending the experiments to other network types, such as Convolutional Neural Networks (CNNs), would be insightful. CNNs, due to their weight-sharing properties, might exhibit different activation distributions, impacting quantisation. Additionally, examining the effects of skip connections, which merge outputs from multiple layers, on quantisation would be interesting. This is because skip connections merge output distributions from different layers, which impacts what quantisation scheme is best to use.

Exploring the vectorisation of piecewise per-neuron transform (Section 5.10) would make this scheme more feasible. Thereby significantly enhancing the variety of quantisation schemes available at no additional cost for PolyLUT. This approach would allow for more tailored and potentially more effective quantisation strategies, leveraging the unique characteristics of each neuron.

Another promising direction for future research would be to investigate the efficacy of applying

the transform without its inverse, similar to the approach taken by Jung et al. [30]. The primary reason for including the inverse transform is to decouple the transform from other network parameters, but this decoupling may not be necessary since the network parameters can simply learn different relationships which incorporate the application of the transform when trained under a QAT approach.

Furthermore, excluding the inverse transform opens up the possibility of combining the transform with a ReLU activation to create a learnable activation function. This approach could lead to more adaptable and efficient quantisation, potentially improving overall performance since learnable activation functions have been demonstrated to enhance the capabilities of neural networks [40].

On the topic of activation functions, it would be interesting to experiment with an activation function that implements histogram equalisation. Layer activations are generally normally distributed, which means that an activation function that incorporates the CDF of a normal distribution to equalise the input tensor may improve performance in quantised neural networks. This is because the equalised distribution will have less aliasing post-quantisation and if applied to the network from scratch (QAT), may result in higher accuracy. Unlike the implementation in Section 5.13, this scheme would not apply the inverse transform post quantisation.

6.4 Conclusions

The primary development of this project, the piecewise learnable transform, has demonstrated significant accuracy improvements when applied to various networks for the jet substructure tagging task. Networks with the transform enabled outperform those without it, even when the latter have higher bit-widths. This indicates that learned quantisation allows the network to use its bits more efficiently, resulting in better performance.

One of the most notable findings is the significant performance difference between networks with and without the transform when using a PTQ approach, as discussed in Section 5.6. Networks trained with the transform maintained almost all of their accuracy after quantisation in the hidden layer, a result not seen in networks without the transform unless the quantisation was non-severe. PTQ is advantageous because it avoids the significant financial and computational costs of retraining. However, the transform introduces a small computational cost during training, as explained in Section 5.8. This cost is incurred only once since the transform needs to be trained

at high precision only once. Subsequently, PTQ can be applied to lower precisions as needed, avoiding the need for retraining with every new bit-width combination.

Furthermore, the results show that under a quantisation with retraining approach, there are gains to be made through non-uniform quantisation of layer activations. The experiments in Section 5.5 highlight the advantages of using this approach, showing that there is a performance increase compared to networks implemented with QAT from scratch, even with the transform disabled.

Finally, the initial developments of this project, discussed in Chapter 4, did not yield significant advantages. These early schemes were fixed and non-parameterised, which prevented them from adapting to the dataset. As a result, certain quantisation schemes, such as logarithmic quantisation, might not have inherently benefited the network. Additionally, the limited testing regime, which included only one dataset, means that the ineffectiveness of logarithmic quantisation and the other schemes cannot be conclusively determined. These schemes might still be effective for other network architectures and datasets. Therefore, broader testing across diverse datasets and architectures is necessary to fully evaluate their potential.

In conclusion, despite facing several challenges, the project yielded positive results, demonstrating a consistent improvement in test accuracy over the baseline. This improvement also led to significant resource savings, a crucial factor in the field of LUT-based neural network design. The insights gained from this project lay a solid foundation for future research. By addressing the identified limitations and exploring the suggested avenues for further investigation, we can make substantial advancements in the field of neural network quantisation.

7

Ethical, Legal and Safety plan

Contents

In general, this project does not have significant ethical concerns outside of those associated with machine learning. However, there are a few key points that are important to consider.

7.1 Safety

Quantisation impacts the precision of a model which affects the performance of a model. As a result, it is more important to thoroughly test quantised models to ensure that performance degradation does not lead to unsafe outcomes, especially in critical applications like autonomous vehicles or medical diagnosis systems.

Interpretability

The process of quantisation can make the internal workings of a model more opaque. This is because quantisation results in reduced precision, which may lead to unexpected model behaviour that is difficult to understand. This is especially critical in models that were originally designed with high precision in mind, where small changes in weights or activations can significantly impact outcomes.

Stakeholders should be informed about how quantisation might affect model interpretability. Providing clear documentation and explaining features helps in developing transparency. This is especially important in domains where understanding model decisions is critical. These areas include healthcare, criminal justice, and financial services. Where model decisions can have significant consequences, understanding the basis of these decisions is crucial. Stakeholders in these industries might prefer a higher level of interpretability to trust and effectively use the models.

7.2 Ethical

Quantisation reduces the size and computational requirements of a model, as explained in Section 2.3. This has ethical advantages in that by reducing the cost of deploying a model, quantisation is capable of reducing the amount of energy consumed to run inferences. This is an important issue, as deploying large machine learning models can have significant environmental consequences [41]. As a result, there is an environmental benefit in implementing quantisation.

However, it can add computational overhead during the training phase. Especially in this project's implementation where each layer needs to compute a transform and inverse transform which, as shown by the complexity analysis, can incur additional non-negligible computational load. This increase in computation translates directly into higher energy consumption, contributing to a larger carbon footprint. In a world increasingly aware of climate change and environmental sustainability, the ethical implications of such increased energy demands are significant.

A

Title of the Appendix

Bitwidth	Degree	Transform	Accuracy JSC-M-LITE	Accuracy JSC-M
8-2-8	2	No transform	73.255	73.797
8-2-8	2	PW per tensor	73.378	74.33
8-2-8	2	PW per neuron	-	-
8-2-8	2	Simple per neuron	73.233	73.647
8-2-8	6	No transform	72.981	72.303
8-2-8	6	PW per tensor	74.417	74.26
8-2-8	6	PW per neuron	-	-
8-2-8	6	Simple per neuron	-	-
8-2-2	2	No transform	72.086	72.97
8-2-2	2	PW per tensor	72.292	73.637
8-2-2	2	PW per neuron	-	-
8-2-2	2	Simple per neuron	-	-
8-2-2	6	No transform	71.963	71.616
8-2-2	6	PW per tensor	73.511	73.649
8-2-2	6	PW per neuron	-	-
8-2-2	6	Simple per neuron	-	-
8-3-8	2	No transform	73.84	74.121
8-3-8	2	PW per tensor	73.962	74.741
8-3-8	2	PW per neuron	73.772	74.513
8-3-8	2	Simple per neuron	73.38	-
8-3-8	6	No transform	74.161	74.294
8-3-8	6	PW per tensor	74.769	74.851
8-3-8	6	PW per neuron	-	-
8-3-8	6	Simple per neuron	-	-
8-3-3	2	No transform	73.551	74.122
8-3-3	2	PW per tensor	73.59	74.494
8-3-3	2	PW per neuron	73.737	74.3
8-3-3	2	Simple per neuron	-	-
8-3-3	6	No transform	73.874	74.072
8-3-3	6	PW per tensor	74.331	74.568
8-3-3	6	PW per neuron	-	-
8-3-3	6	Simple per neuron	-	-
6-3-3	2	No transform	73.331	-
8-4-8	2	No transform	74.164	74.521
8-4-8	2	PW per tensor	74.317	74.995

Table A.1: Table of the maximum test accuracy across various network configurations.

Bibliography

[1] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *CoRR*, vol. abs/2103.13630, 2021. arXiv: 2103.13630. [Online]. Available: https://arxiv.org/abs/2103.13630.

[2] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[3] K. Sayood, *Introduction to Data Compression, Fourth Edition*, 4th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012, ch. 9.6.2, pp. 282–287, ISBN: 0124157963.

[4] A. Paszke, S. Gross, F. Massa, *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 [cs.LG].

[5] NVIDIA, P. Vingelmann, and F. H. Fitzek, *Cuda, release: 10.2.89*, 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper%5C_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[7] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper%5C_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper%5C_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[9] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[10] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "The role of over-parametrization in generalization of neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=BygfghAcYX.

[11] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[12] R. Gray and D. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998. DOI: 10.1109/18.720541.

[13] A. Pappalardo, *Xilinx/brevitas*, 2023. DOI: 10.5281/zenodo.3333552. [Online]. Available: https://doi.org/10.5281/zenodo.3333552.

[14] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013. arXiv: 1308.3432. [Online]. Available: http://arxiv.org/abs/1308.3432.

[15] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[16] E. Wang, J. J. Davis, R. Zhao, *et al.*, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Computing Surveys*, vol. 52, no. 2, pp. 1–39, May 2019, ISSN: 1557-7341. DOI: 10.1145/3309551. [Online]. Available: http://dx.doi.org/10.1145/3309551.

[17] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos, "Memory requirements for convolutional neural network hardware accelerators," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 111–121. DOI: 10.1109/IISWC.2018.8573527.

[18] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2018, pp. 2704–2713. DOI: 10.1109/CVPR.2018.00286. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00286.

[19] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018. arXiv: 1806.08342. [Online]. Available: http://arxiv.org/abs/1806.08342.

[20] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[21] B. Zhuang, M. Tan, J. Liu, L. Liu, I. Reid, and C. Shen, "Effective training of convolutional neural networks with low-bitwidth weights and activations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6140–6152, 2022. DOI: 10.1109/TPAMI.2021.3088904.

[22] S. M. Nabavinejad, M. Baharloo, K.-C. Chen, M. Palesi, T. Kogel, and M. Ebrahimi, "An overview of efficient interconnection networks for deep neural network accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 268–282, 2020. DOI: 10.1109/JETCAS.2020.3022920.

[23] E. Chung, J. Fowers, K. Ovtcharov, *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018. DOI: 10.1109/MM.2018.022071131.

[24] E. Nurvitadhi, G. Venkatesh, J. Sim, *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, Monterey, California, USA: Association for Computing Machinery, 2017, pp. 5–14, ISBN: 9781450343541. DOI: 10.1145/3020078.3021740. [Online]. Available: https://doi.org/10.1145/3020078.3021740.

[25] M. Andronic and G. A. Constantinides, "PolyLUT: Learning Piecewise Polynomials for Ultra-Low Latency FPGA LUT-based Inference," in *2023 International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 60–68. DOI: 10.1109/ICFPT59805.2023.00012.

[26] E. Wang, J. J. Davis, P. Y. Cheung, and G. A. Constantinides, "Lutnet: Rethinking inference in fpga soft logic," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2019, pp. 26–34.

[27] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "Logicnets: Co-designed neural networks and circuits for extreme-throughput applications," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 291–297. DOI: 10.1109/FPL50879.2020.00055.

[28] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 3123–3131.

[29] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5900–5904. DOI: 10.1109/ICASSP.2017.7953288.

[30] S. Jung, C. Son, S. Lee, *et al.*, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4345–4354. DOI: 10.1109/CVPR.2019.00448.

[31] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.

[32] K. Yamamoto, "Learnable companding quantization for accurate low-bit neural networks," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 5027–5036. DOI: 10.1109/CVPR46437.2021.00499.

[33] J. Duarte, S. Han, P. Harris, *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, P07027–P07027, Jul. 2018, ISSN: 1748-0221. DOI: 10.1088/1748-0221/13/07/p07027. [Online]. Available: http://dx.doi.org/10.1088/1748-0221/13/07/P07027.

[34] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 1969. DOI: 10.1109/TSSC.1969.300225.

[35] E. W. Weisstein, *Sigmoid function.* https://mathworld.wolfram.com/HyperbolicTangent.html.

[36] E. W. Weisstein, *Hyperbolic tangent.* https://mathworld.wolfram.com/HyperbolicTangent.html.

[37] E. W. Weisstein, *Heaviside step function.* https://mathworld.wolfram.com/HeavisideStepFunction.html.

[38] E. W. Weisstein, *Delta function.* https://mathworld.wolfram.com/DeltaFunction.html.

[39] M. Andronic and G. A. Constantinides, *Neuralut: Hiding neural network density in boolean synthesizable functions*, 2024. arXiv: 2403.00849 [cs.AR].

[40] Z. Liu, Y. Wang, S. Vaidya, *et al.*, "Kan: Kolmogorov-arnold networks," *arXiv preprint arXiv:2404.19756*, 2024.

[41] D. A. Patterson, J. Gonzalez, Q. V. Le, *et al.*, "Carbon emissions and large neural network training," *CoRR*, vol. abs/2104.10350, 2021. arXiv: 2104.10350. [Online]. Available: https://arxiv.org/abs/2104.10350.