Alina Vasilciuc

Bachelor in Sciences of Electrical and Computer Engineering

Data Visualization for Benchmarking Neural Networks in Different Hardware Platforms

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Electrical and Computer Engineering

Adviser: Luís Gomes, Associate Professor with Habilitation,
NOVA University Lisbon

Examination Committee

Chair:	Rui Neves-Silva
Rapporteur:	André Damas Mora
Member:	Luís Gomes

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2021

Data Visualization for Benchmarking Neural Networks in Different Hardware Platforms

This document was created using the (pdf)LaTeX processor, based on the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To my family,

Acknowledgements

I would like to especially thank my supervisor Professor Luís Gomes, for all the excellent advice and for always pushing me to learn more, to work harder and for his support and guidance.

I want to thank FCT-UNL for hosting this course and making all of this possible.

I want to express my gratitude to my manager Michaela Blott who saw potential inside me and took the time to get to know me and for giving me this lifetime opportunity of doing an internship at Xilinx, Inc. I truly appreciate the confidence Michaela showed in me, for all the support, the insights and guidance. Thank you for our friendship, for lifting my spirits when hard times would come and for all advice provided.

A special thank you to Xilinx, Inc and all of my work colleagues, for their friendship, companionship and support throughout this journey.

I would like to sincerely thank my parents, their trust and confidence in me, for their support, for cheering me up through more challenging times and for making all of this possible.

And finally a big thank you to all the Professors at FCT-UNL that took the time to teach me, always pushing me harder and for the thousands of lessons they taught me throughout these five years.

Abstract

The computational complexity of Convolutional Neural Networks has increased enormously; hence numerous algorithmic optimization techniques have been widely proposed. However, in a space design so complex, it is challenging to choose which optimization will benefit from which type of hardware platform. This is why QuTiBench - a benchmarking methodology - was recently proposed, and it provides clarity into the design space. With measurements resulting in more than nine thousand data points, it became difficult to get useful and rich information quickly and intuitively from the vast data collected.

Thereby this effort describes the creation of a web portal where all data is exposed and can be adequately visualized. All the code developed in this project resides in an online public GitHub repository, allowing contributions.

Using visualizations which grab our interest and keep our eyes on the message is the perfect way to understand the data and spot trends. Thus, several types of plots were used: rooflines, heatmaps, line plots, bar plots and Box and Whisker Plots.

Furthermore, as level-0 of QuTiBench performs a theoretical analysis of the data, with no measurements required, performance predictions were evaluated. We concluded that predictions successfully predicted performance trends. Although being somewhat optimistic because predictions become inaccurate with the increased pruning and quantization. The theoretical analysis could be improved by the increased awareness of what data is stored in the on and off-chip memory. Moreover, for the FPGAs, performance predictions can be further enhanced by taking the actual resource utilization and the achieved clock frequency of the FPGA circuit into account. With these improvements to level-0 of QuTiBench, this benchmarking methodology can become more accurate on the next measurements, becoming more reliable and useful to designers.

Moreover, more measurements were taken, in particular, power, performance and accuracy measurements were taken for Google's USB Accelerator benchmarking EfficientNet S, EfficientNet M and EfficientNet L. In general, performance measurements were reproduced; however, it was not possible to reproduce accuracy measurements.

Keywords: Deep Learning, Field Programmable Gate Arrays, Graphics Processing Unit, Benchmarks, QuTiBench

Resumo

A complexidade computacional de Redes Neuronais Convulsionadas aumentou muito nos últimos anos, consequentemente inúmeras técnicas de otimização têm sido propostas para endereçar este assunto. No entanto, num espaço de desenvolvimento tão complexo, é difícil escolher que otimização vai beneficiar nestas plataformas de hardware, nomeadamente nas FPGAs, GPUs e ASICs. É por isto que QuTiBench — uma metodologia de benchmarking (avaliação comparativa) — foi recentemente proposta, a qual gerou um conjunto de dados com mais de nove mil data points, os quais se tornam difíceis de analisar e tirar informação útil de forma fácil e intuitiva.

Assim, este documento tem como objetivo descrever a criação de um portal web onde todos os dados estão expostos e podem ser devidamente visualizados. Todo o código desenvolvido para este documento está no GitHub, um repositório online e público, permitindo contribuições de terceiros. Usando visualizações que captam o interesse e passam a mensagem de uma forma intuitiva é a melhor maneira de entender os dados e detetar tendências. Para o efeito, diversos tipos de gráficos foram usados tais como: rooflines, mapa de calor, gráfico de linha, gráfico de barra e diagrama de caixa.

Como o nível 0 de QuTiBench faz uma análise teórica dos dados, foi efetuada uma análise entre o previsto e o medido. Concluiu-se que as previsões teóricas conseguiram prever as tendências gerais embora ainda sendo um pouco otimistas. A análise teórica pode ser melhorada ao manter o registo de se os dados estão na memória on-chip ou off-chip. Para as FPGAs, as previsões de performance podem ser melhoradas ao ter em conta a verdadeira utilização de recursos e a frequência de clock atingida pelo circuito. Com estes melhoramentos, o nível 0 de QuTiBench consegue prever a performance com maior grau de certeza e gerar dados mais confiáveis.

Também foram feitas mais medições de consumo de energia, desempenho e precisão ao USB Accelerator da Google das redes neuronais EfficientNet S, EfficientNet M e EfficientNet L. Em geral reproduziram-se os resultados apresentados pela Google em termos de desempenho, mas não foi possível reproduzir os de precisão.

Palavras-chave: Deep Learning, Field Programmable Gate Arrays, Graphics Processing Unit, Benchmarks, QuTiBench

Contents

List of Figures xv

List of Tables xvii

Acronyms xix

1 Introduction 1

1.1 Context 1

1.2 Motivation 2

1.3 Problem statement 4

1.4 Objectives and Contributions 4

1.4.1 Objective 4

1.4.2 Contributions 4

1.5 Document Structure 5

2 Related work 7

2.1 Definitions and Generic Background 7

2.1.1 Neural Networks 7

2.1.2 Deep Neural Networks 9

2.1.3 Convolutional Neural Networks 9

2.1.4 Popular DNN models 10

2.1.5 Popular Data Sets for Classification 12

2.2 Algorithmic Optimization techniques 13

2.3 Hardware for DNN Processing 14

2.3.1 Increased Efficiency in Temporal Architectures 14

2.3.2 Increased Efficiency in Spatial Architectures 15

2.4 Hardware Architectures for Deep Learning 16

2.4.1 FPGA: FINN and BISMO architetcures 18

2.4.2 Coral USB Accelerator 20

2.4.3 Intel NCS2 22

2.5 Characteristics and Challenges in Benchmarking 22

2.6 Existing Benchmarking 23

2.6.1 NN System Benchmarks 23

2.6.2 ML Benchmarks . . . 24
2.6.3 Performance Benchmarks . . . 25
2.7 Rooflines . . . 25
2.7.1 Context . . . 25
2.7.2 What is it? . . . 26
2.7.3 The Roofline Model . . . 26
2.7.4 How to improve performance . . . 27
2.8 Data Visualizations . . . 28
2.9 Previous implementation: QuTiBench . . . 29
2.9.1 Experimental Results and Evaluation . . . 31

3 Development: Benchmarking Hardware 35
3.1 Coral USB Accelerator . . . 35
3.1.1 Data acquisition . . . 35
3.1.2 Results . . . 39

4 Development: Website Visualizations 43
4.1 Theoretical Analysis - Level 0 of QuTiBench . . . 44
4.1.1 Methodology . . . 44
4.1.2 Results . . . 48
4.2 MNIST Classification - Level 3 of QuTiBench . . . 55
4.3 Raw Measurements . . . 57
4.4 FAIR and Open Data . . . 58
4.5 How to make contributions page . . . 58
4.6 Overview of the Experiments page . . . 58
4.7 Code and online repository . . . 58

5 Evaluation 61
5.1 Theoretical Pareto and Measured Pareto Overlapped . . . 61
5.2 Efficiency plots . . . 63
5.3 Evaluation between theoretical and measured . . . 64
5.3.1 FPGA Platforms . . . 65
5.3.2 GPU Platforms . . . 66
5.3.3 Edge TPU and NCS2 Platforms . . . 69

6 Conclusions and Future work 71

Bibliography 75

Annexes 81

I Annex 1 List of figures 81

List of Figures

1.1 Characteristics of winning Neural Networks model from the ImageNet Challenge . . . 3

2.1 History of Neural Networks [46]. . . . 8
2.2 Simple Neural Network example and terminology . . . 9
2.3 A diagram depicting all steps that a single grayscale input image endures in each layer. Figure taken from [2]. . . . 10
2.4 Model Scaling . . . 12
2.5 Highly parallel compute paradigms [46]. . . . 14
2.6 Mapping to matrix multiplication for convolutional layers . . . 15
2.7 Read and write access per MAC [46]. . . . 16
2.8 Memory hierarchy and data movement energy cost [46]. . . . 16
2.9 Floor Plan of Google's TPU [29]. . . . 18
2.10 Overview of BISMO's hardware architecture [56]. . . . 19
2.11 Overview of FINN flow for a BNN . . . 19
2.12 Matrix Multiplier + summation [36]. . . . 21
2.13 "Roofline model for (a) AMD Opteron X2 on left and (b) Opteron X2 vs. Opetron X4 on the right" [58]. . . . 27
2.14 Illustration of the multi-tiered approach proposed by the benchmark methodology - QuTiBench [7]. . . . 29
2.15 Rooflines provided by level 0 for GoogleNet and ResNet50 on TX2 and ZCU104 [7]. . . . 32
2.16 "Performance comparison layer0, layer1, layer2 and layer3 for TX2 (MaxN, FP16 configuration)"[7] for ResNet50. . . . 32
2.17 "System Performance evaluation" [7] for TX2 and ZCU104 for INT8, FP16, FP32, for both GoogleNetV1 and ResNet50 and for different batch sizes [7]. . . 32

3.1 Block diagram between the USB accelerator and the computer. . . . 36
3.2 UM34C Power meter used in the power measurements. . . . 39
3.3 Connection between the computer, the Power Meter and the USB Accelerator. 39
3.4 Average and standard deviation for power measurements on the USB Accelerator. . . . 41
3.5 Power consumption evolution over time for ImageNet task. . . . 42

4.1 Fastpages diagram [18]. 43
4.2 Performance estimate on a Roofline model [8]. 47
4.3 CNNs compute and memory requirements. Figure available at [5]. 49
4.4 Roofline Model. Figure available at [5]. 50
4.5 Performance prediction with roofline analysis. Figure available at [5]. 51
4.6 Performance Predictions with Heatmaps for MNIST classification 52
4.7 Effect of quantization on OI and performance [8]. 53
4.8 Theoretical Pareto for MNIST Classifications. Figure available at [5]. 54
4.9 Power consumption. Figure available at [5]. 55
4.10 Accuracy versus performance. Figure available at [5]. 56
4.11 Power bar charts for MNIST. Figure available at [5]. 57
4.12 Performance normalized Bar charts for MNIST. Figure available at [5]. 57
4.13 Website structure 59
4.14 GitHub repository overview 59

5.1 Overlapped pareto plot for MNIST classification. 63
5.2 Efficiency Bar charts for MNIST 64

I.1 Hardware platforms with their Peak Performance Predictions. Figure available at [5]. 82
I.2 Performance Predictions with Heatmap for ImageNet classification. Figure available at [5]. 83
I.3 Performance Predictions with Heatmap for CIFAR-10 classification. Figure available at [5]. 83
I.4 Theoretical Pareto for ImageNet Classifications. Figure available at [5]. 84
I.5 Theoretical Pareto for CIFAR-10 Classifications. Figure available at [5]. 85
I.6 Overlapped Pareto Plot for CIFAR-10 Classification. 86
I.7 Overlapped Pareto Plot for ImageNet Classification 87
I.8 Bar chart showing Efficiency for MNIST classification over NCS2 and the A53. Figure available at [5]. 88
I.9 Bar chart showing Efficiency for MNIST classification over the TX2. Figure available at [5]. 88

List of Tables

1.1 ResNet models' training time benchmarked with fb.resnet.torch using four M40 GPUs. Table taken from Lecture 15 – Efficient Methods and Hardware for Deep Learning. . . . 3

2.1 DNN models summarized [7]. . . . 13
2.2 Hardware platforms and neural networks model [7]. . . . 30

3.1 Latency Results 40
3.2 Accuracy results from benchmarking EfficientNets on USB Accelerator . . . 40
3.3 Power consumption results from benchmarking EfficientNets on USB Accelerator 41

4.1 Experimental CNNs and their accuracies. Table adapted from [8] 45
4.2 CNNs and their compute and memory requirements. Table Adapted from [8]. 48
4.3 Level 0 of QuTiBench. Hardware platforms with their Peak Performance Predictions [8]. . . . 50

5.1 Terminology explained 62
5.2 Efficiency results for the FPGAs for all pruned and quantized variants of CNV. 66
5.3 Efficiency results for the FPGAs for all pruned and quantized variants of MLP. 66
5.4 Efficiency results for the FPGAs for all topologies of ImageNet. . . . 66
5.5 Efficiency results for the TX2s for all pruned and quantized variants of CNV. 67
5.6 Efficiency results for the TX2s for all pruned and quantized variants of MLP. 68
5.7 Efficiency results for the TX2s for all topologies of ImageNet. . . . 68
5.8 Efficiency results for the NCS2 for all topologies of ImageNet. . . . 69
5.9 Efficiency results for the NCS2 for all pruned and quantized variants of MLP. 69
5.10 Efficiency results for the NCS2 and USB Accelerator for all topologies of ImageNet ML task. . . . 69

Acronyms

AI	Artificial Inteligence
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
BNN	Binarized Neural Network
CLB	Configurable logic BLock
CNN	Convolutional Neural Network
CONV	Convolutional
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
DPU	Deep Learning Processing Unit
DRAM	Dynamic random-access memory
EIE	Efficient Inference Engine
FC	Fully Connected
FET	Field-effect Transistors
FFT	Fast Fourier Transform
FLOP	Floating-point Operation
FPGA	Field Programmable Gate Array
GOP	Giga Operation
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	Internet of Things

MAC	Multiply Accumulate
ME	Millions of Elements
ML	Machine Learning
NN	Neural Network
OI	Operational Intensity
PCB	Printed Circuit Board
PE	Processing Engine
POP	Peta Operation
RF	Register File
SIMD	Single instruction multiple data
SIMT	Single instruction multiple threads
SoC	System on Chip
SRAM	Static Random Access Memory
TOP	Tera Operation
TPU	Tensor Processing Unit

Chapter 1

Introduction

Isn't it a pleasure to study and practice what you have learned?
-Confucious

1.1 Context

Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) are nowadays shaping the future of every industry and every human being. To be clear on the terms: Deep Learning forms a subset of Machine Learning, which forms a subset of Artificial Intelligence. AI is the main driver of emerging technologies like robotics, big data and IoT, and it is foreseen to continue to be a technological enabler. Some sectors are at the beginning of their AI journey, whereas others are veteran travellers. AI's impact is found on a large set of applications that goes from Transportation to Healthcare and Education.

Neural networks have been around for a while, and recently they have been on the spotlight for many machine learning applications. They can recognize patterns, clustering it and classifying it. They have been applied to all kinds of applications because they are so easy to use.

However, these algorithms strongly rely on computer power, and with it comes massive power consumption. Moreover, their heavy memory requirements make deployment rather difficult in energy-constrained environments such as embedded systems.

With addressing these issues, a new plethora of hardware architecture has emerged, specifically heterogeneous hardware architecture along with co-designed algorithms. Furthermore, to diminish computer requirements as well as more significant bandwidth requirements with the same accuracy, algorithmic and software optimizations, as well as architectural optimizations for NNs, are being searched. Although some optimizations consist of performing the same operations faster, other optimizations change entirely the way the training is done.

Comparing all these criteria is rather difficult due to the scarcity of standard evaluation criteria that examines all these considerations. This document is based on *QuTiBench* [7], a Benchmark suite that addresses this need. Hardware systems and system topologies can differ a lot in design options. Each component and system design quite often comes with its power, programmability and design trade-offs. For a significant amount of reasons, what may be convenient for one system designer's need may not be suitable for some other system designer. However, there is no way to compare different design options. While there is some work done in this area, it only covers very little of the embedded design space. Up until today, benchmarks have trouble supporting important algorithmic optimizations such as quantization, pruning and specialized heterogeneous hardware architectures.

Due to the difficulty of comparing all these criteria, this document will present data visualizations that could provide better understanding and comparison between all the plethora of hardware architecture and techniques that have arisen.

1.2 Motivation

Deep Learning relies on computer power to solve more complex problems, and with new computing technologies, companies can have AI models that can learn to solve complex problems. Deep Learning enables researchers to train "with billions of parameters" [4] "from massive datasets that improve in accuracy as the dataset grows" [10]. However, all this computer power comes with great power consumption [49], and considering that the world is experiencing an unprecedented growing electricity consumption by computers, data centres and a growing number of internet users [27], the need to turn towards greener solutions arises. To overcome the power issue makes it mandatory to shift from traditional computing paradigms to new ones.

Deep Learning is changing our lives. However, there is a recent trend in which, to achieve higher accuracy, models are getting larger and larger. Figure 1.1 shows that there was an increase of 16 times in the model size of the winners in the ImageNet Challenge from 2012 to 2015. Another example is Baidu's deep speech model which increased ten times the number of operations in only one year. All these heavy models make distribution through over-the-air update very difficult. In mobile phones, for instance, if the item is larger than one hundred megabytes, it can only be downloaded through wi-fi.

Another challenge that comes with larger models is speed. Figure 1.1 illustrates the training time taken by several types of ResNets. The lowest error rate of 6.16% takes 1.5 weeks to train on four M40 GPUs. Furthermore, it is only 0.1% better than the second-best error rate. This extended training time makes the researcher's productivity greatly limited.

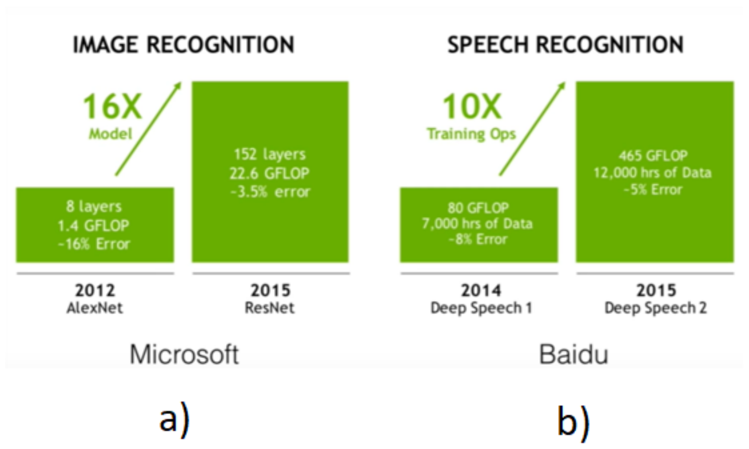


Figure 1.1: "Characteristics of winning Neural Networks model from the ImageNet Challenge. a)Increase of 16 times in neural networks' model size between winner AlexNet in 2012 to winner ResNet in 2015. b) Ten times increase in the number of training operations in Baidu's deep speech neural network from 2014 to 2015" [22].

Table 1.1: ResNet models' training time benchmarked with fb.resnet.torch using four M40 GPUs. Table taken from Lecture 15 – Efficient Methods and Hardware for Deep Learning.

	Error rate	Training time
ResNet18	10.76%	2.5 days
ResNet50	7.02%	5 days
ResNet101	6.21%	1 week
ResNet152	6.16%	1.5 weeks

The third most crucial challenge is energy efficiency. In the embedded environment with energy constraints, larger models can drain all the power. In data centres, power consumption is also crucial. The objective is to find the optimum balance that breaks the boundary between algorithm and hardware, thus, improving performance and power consumption.

The increasing need for more computer power has been fostering the increase in new technology targeting AI. However, existing benchmarks are not suitable enough to exhaustively analyze hardware as well as algorithms to provide knowledge on system problems and thus instigate greener solutions. Moreover, the lack of standard benchmarking methodology further aggravates the existing situation.

With more than nine thousand data points generated by QuTiBench, arises the need for adequate visualization of the data in a manner that trends can be spotted, and it can send a clear message. Moreover, there is a need for a web portal where all data can be publicly available online, and that could allow for contributions.

This is why QuTiBench [7], the benchmarking methodology, which was used to take all measurements taking into account hardware variety, numerous algorithmic optimization techniques and a plethora of neural networks now need a proper way to present results to the public in an insightful way.

1.3 Problem statement

During this work, it was verified that as AI is emerging in numerous use cases, there is a need to make it faster but especially greener. To achieve this, QuTiBench was proposed, and after it collected more than nine thousand data points now, there is a need to understand all data collected.

The following thesis is defended in this dissertation: *It is possible to improve existing benchmarking work by making use of QuTiBench, a benchmark methodology, that supports a different kind of algorithmic optimizations such as quantization and provides system designers with some insight about strengths and limitations on recent compute architectures regarding Neural Networks. Having so much data from more than nine hundred measurements creates the need for a powerful tool to understand them better. In this way, different kind of visualizations need to be created to gain better comprehension into all trade-offs between algorithmic optimizations and hardware choices. Moreover, these visualizations need to be available online inside a web portal where contributions can be made to the project.*

Besides a powerful way to see and understand all gathered data, there is also a need for it to be publicly accessible and to allow for contributions from third parties whom themselves have performed measurements and wish to contribute.

Moreover, more measurements need to be taken. As Google announces that EfficientNet S, EfficientNet M and EfficientNet L achieve very high accuracy on its low power, high throughput USB Accelerator, it becomes essential to include this device in the QuTiBench measurements.

1.4 Objectives and Contributions

1.4.1 Objective

Considering the problem statement and related considerations, the objective of the thesis is to display all data gathered by the benchmark suite to reveal the potential of different hardware platforms publicly. In the envisaged approach, the idea is to explore neural networks performance and hardware performance to maximize both for specific use cases.

1.4.2 Contributions

Related publications co-authored by this dissertation author are listed below:

- Michaela Blott, Nicholas J.Fraser, Giulio Gambardella, Lisa Halder, Johannes Kath, Zachary Neveu, Yaman Umuroglu, Alina Vasilciuc, Miriam Leeser, and Linda Doyle, “Evaluation of Optimized CNNs on Heterogeneous Accelerators using a Novel Benchmarking Approach” in IEEE Transactions on Computers, DOI: 10.1109/TC.2020.3022318.

- M. Blott, A. Vasilciuc, M. Leeser and L. Doyle, “Evaluating Theoretical Baselines for ML Benchmarking Across Different Accelerators,” in IEEE Design & Test, DOI: 10.1109/MDAT.2021.3063340.

1.5 Document Structure

The remaining of this report is divided into five chapters.

- **Chapter 2 - Related Work**: presents related work. It starts with a background on neural networks, enumerates optimization techniques, an existing hardware architecture for deep learning and describes existing Benchmarking. It presents the key concepts of QuTiBench. Describes the various types of visualizations used.
- **Chapter 3 - Development: Benchmarking Hardware** presents this work contribution; in particular, it describes how benchmarking Google’s USB accelerator was done.
- **Chapter 4 - Development: Website Visualizations**: describes the freshly build a web portal, all its contents and visualizations.
- **Chapter 5 - Evaluation**: shows specific visualizations that highlight the difference between what was predicted and what was measured. Moreover, it evaluates how accurate predictions were compared to the measured data points and concludes how to improve level 0 of QuTiBench.
- **Chapter 6 - Conclusions and Future work**: summarizes the main conclusions and presents future directions.

Chapter 2

Related work

This chapter introduces previous related work. It focuses on benchmarking deep learning algorithms and resents several state-of-the-art hardware architectures targeting neural networks.

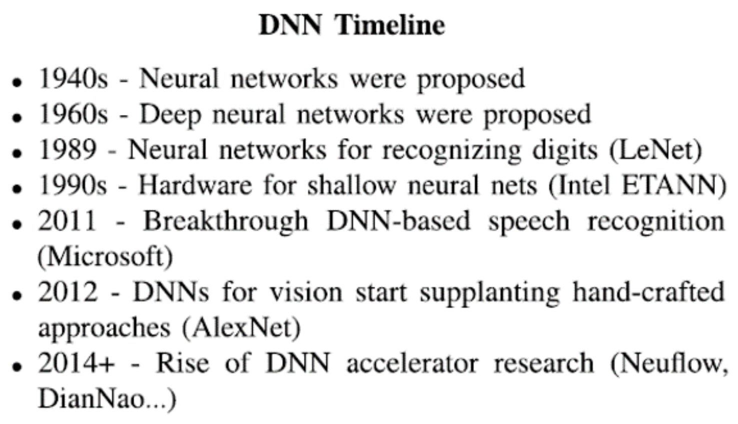
2.1 Definitions and Generic Background

2.1.1 Neural Networks

Neural networks are a class of machine learning algorithms based on several connected nodes called artificial neurons roughly modelled after the human brain that are designed to detect patterns.

Although they were proposed in the 1940s, their very first application did not appear until the 1980s when a network called LeNet was developed for handwritten digit recognition, which is still today used by ATMs to verify digit recognition on checks [31]. Later, in 2010 NN-based application started to grow exponentially until present days. Figure 2.1 shows a timeline of popular NNs.

Neural networks accomplish this by considering examples and by automatically finding specific characteristics from the samples they process. Over the past few years, neural networks have become a major success and have been used in a variety of tasks, including computer vision, robotics, speech recognition and machine translation. An exciting aspect of NNs is that they have the theoretical property of being a type of machine learning which requires zero domain expertise, which makes them an appealing solution both for unsolved problems and for solved ones. However, all this computational power comes at a cost, the tremendous demand for more computing power comes with enormous power consumption and memory requirements concerning capacity and access bandwidth, which becomes an obstacle, especially in the embedded space.


DNN Timeline

- 1940s - Neural networks were proposed
- 1960s - Deep neural networks were proposed
- 1989 - Neural networks for recognizing digits (LeNet)
- 1990s - Hardware for shallow neural nets (Intel ETANN)
- 2011 - Breakthrough DNN-based speech recognition (Microsoft)
- 2012 - DNNs for vision start supplanting hand-crafted approaches (AlexNet)
- 2014+ - Rise of DNN accelerator research (Neuflow, DianNao...)

Figure 2.1: History of Neural Networks [46].

Unfortunately, evaluating ML performance for training systems is undoubtedly more challenging than performance evaluation for classic software. ML systems are not like the traditional processing systems because they operate in a very different way [39]. Traditional systems are programmed with specific logic, whereas ML systems are fed massive datasets and are expected to apply algorithms that find patterns and features in all data. "The main goal of ML is to train a model that makes high-quality predictions on unseen data, which is referred to as *generalization*" [10]. This goal is achieved through the minimization of a loss-function to find a model that can learn weights from the training dataset and more importantly, *generalizes* to new data from a similar distribution. Existing Benchmarks measure proxy metrics, for instance, time to process a minibatch of data, but do not take into account whether the system as a whole will produce a high-quality result.

Technically described, neural networks are a connected graph of neurons. Each neuron processes a function to its input(s) to generate an output, thus being a processing component. "Edges" are the connections between two different neurons and are represented through a number. The output of each neuron is computed by a non-linear function of the sum of its inputs. Neurons, as well as edges, both have a weight, which can increase or decrease the strength of the signal, and it is continually adjusted as the learning process proceeds. Usually, neurons are assembled into many layers, in which each different layer applies a different transformation to its input. Figure 2.2 depictures a computational neural network. Values are propagated from the input layer to the hidden layer, also called the middle layer. The final value is presented to the user after all weighted sums from all hidden layers eventually propagated to the output layer. *Activations* are sometimes referred to as the outputs of neurons.

Computation at each layer is done with the following equation: $y_i = f(\sum_{i=1}^{3} W_{ij} \times x_i + b)$, where W_{ij} are the weights, x_i are the input activations, y_i are the output activations and f(.) is a nonlinear function. The bias term is absent in Figure 2.2 for simplicity.

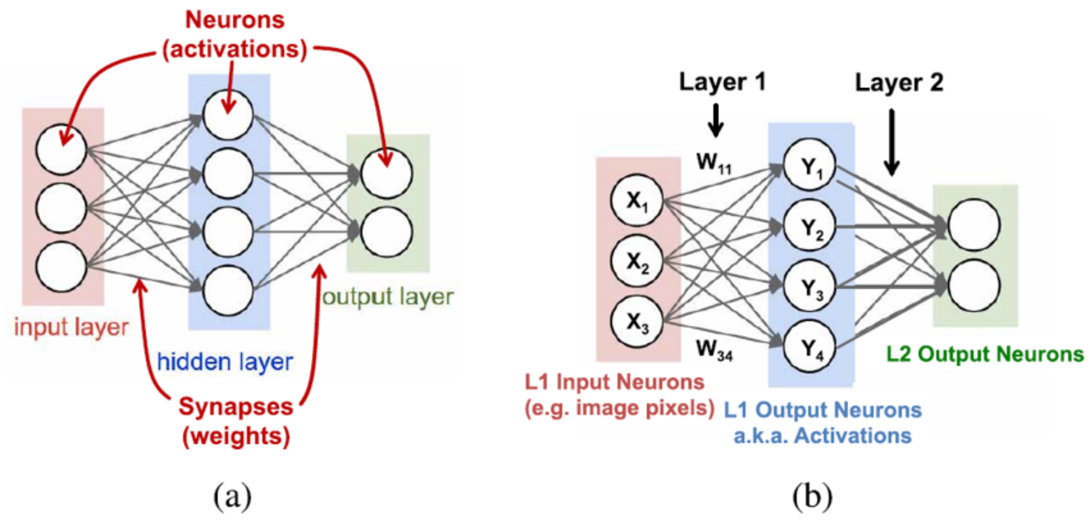


Figure 2.2: "Simple Neural Network example and terminology. (a) Neurons and Synapses. (b) Compute weighted sum of each layer" [46].

2.1.2 Deep Neural Networks

All the different connections between all different layers define the structure of the network. Deep Neural Networks (DNNs) are classified as such due to the number of layers, in this case, more than three layers. That means that there are one or more hidden layers. Nowadays, the usual range of hidden layers goes from five to more than a thousand [46]. In a fully connected network, all neurons from a particular layer are connected to every neuron from the subsequent layer.

DNNs are broadly used for image processing, as they can distinguish high-level features. When pixels from an image are fed to the DNN, the first layer can extract low-level features like lines and edges. At the next layer, all previous features are used to extract higher-level features like shapes. In the end, the output will be a probability that all those high-level characteristics belong to a particular object or scene. The use of all those layers enables DNNs to achieve more outstanding performance in many use cases.

2.1.3 Convolutional Neural Networks

A Convolutional Neural Network' structure is similar to a DNN except for the fact that convolutional neural networks (CNNs) are specialized for image-related tasks. A CNN is a deep learning algorithm that takes an input, usually, an image and drafts the importance of all features in the token image and, after it is trained, it is ready to differentiate any given images. CNNs can understand the Spatial and Temporal dependencies in an image through the application of relevant filters. In a CNN architecture, if the input image is on grayscale, then the input layer is 1-dimensional, but if the image comes in colours, then the input layer is 3-dimensional which corresponds to the colour channels like RGB.

Each image is represented as a 3D matrix with a dimension for the height, width and depth. Depth refers to the colour channels used in an image. The architecture comprises two types of layers: convolution and subsampling, also known as pooling layers.

In a convolutional layer, the mapping of activations from each layer to the subsequent is done using a filter. Thus, a convolution is defined, as it is shown in figure 2.3. Figure 2.3 shows an input image of $32x32$, which is convolved with a filter of weights of $5x5$. The learned weights are *kernels* that are convolved with the image to extract features. The convolution uses a multidimensional filter of weights, which means that numerous convolutions are performed on the input, in which each operation uses a different filter. The subsequent layers are calculated by doing the dot product between the weights in the filter and the image pixels. This results in several feature maps. In figure 2.3 C1 holds all six feature maps, which means that six different filters were applied. These filters is to highlight contrast, blur, edge and many other things. Performing the convolution between the filter of weights and pixels is done at all possible positions, from left to right and from top to down.

The objective of pooling is to extract only "interesting" features, which is achieved by downsampling each one of the feature map. These convolutions, followed by pooling layers, are repeated multiple times in a CNN to produce high-quality features present in the input images.

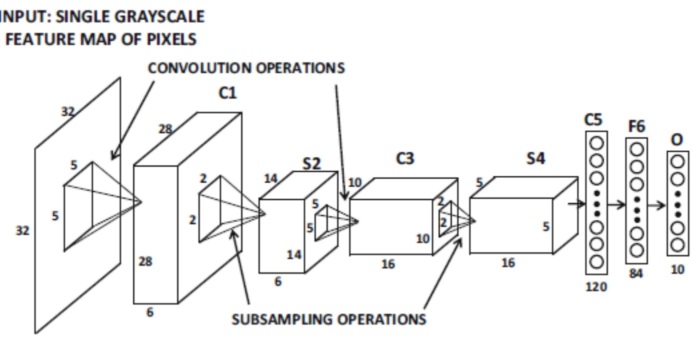


Figure 2.3: A diagram depicting all steps that a single grayscale input image endures in each layer. Figure taken from [2].

2.1.4 Popular DNN models

Apart from the neural networks described above, there are other different types, and newer ones continue to arise. In this section, an overview of some of the most popular NNs will be given, most of these models are available for download.

Some of the NNs described below competed in the ImageNet Challenge [41]. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition which uses the computer vision dataset called ImageNet. It accounts for more than 14 million images and around 1 million images with boxes around an identified object in each image. Accuracy has been improved significantly from the 2010 edition to the 2014 edition. There have been many important developments and academic publications relating to ILSVRC.

LeNet [31] was one of the first Convolutional Neural Network approaches introduced in 1989, it is what made CNNs successful for the first time. Back then it was used for digit recognition on checks used by ATMs. Its designated task was to recognize digit handwritten in greyscale images of size 28×28. Throughput rates obtained were of more than ten digits per second. LeNet-5, the most well-known version contains three convolutional (CONV) layers and two fully connected layers (FC) [32]. Each one of the convolutional layers uses filters of size 5×5, being one channel per filter. After the second CONV layer, there is a subsampling layer. For the nonlinearity, a sigmoid is used. Summing up, LeNet requires 60000 weights and 341000 multiply-and-accumulate (MACS) per one input (image) [32].

AlexNet [30] won the ImageNet Challenge in 2012 being the very first convolutional neural network to achieve this. It has five CONV layers and three FC layers, it also has pooling layers. A powerful GPU implementation of the convolution operation was used to speed up training, along with saturating neuron. The overfitting in the fully connecting layers was reduced due to the application of a method called "dropout". This NN achieved a winning top-1 test error rate of 37.5% and a top-5 test error rate of 17%.

Overfeat's [42] architecture is analogous to AlexNet, it comes with "five convolutional layers and three fully connected layers" [7]. The difference between Overfeat and AlexNet is the number of filters used in each layer. For layers 3, 4 and 5 Overfeat uses a larger amount of filters. Thus, there is a significantly higher amount of weights, one hundred and forty-six million to be exact. Overfeat has two main model variations: described above is fast, and there is also another model referred to accurate model used in the ImageNet Challenge. Although the accurate model achieves better accuracy than the fast model, it performs 1.9× more MACs.

VGG-16 [43] has sixteen layers, thirteen are convolutional layers and three FC layers. Because it has so many layers, 5×5 filters are assembled from other 3×3 filters. It is estimated that "VGG-16 requires one hundred and thirty-eight million weights and 15.5 MACs to process a 224×224 input image" [46]. There are two variant models of VGG: VGG-16 (described above) and VGG-19, which produces a lower top-5 error rate, however, it does 1.27× more MACS.

GoogLeNet [47] goes deeper with twenty-two layers composed of three CONV layers followed by nine inception layers and one FC layer. It introduces an inception module that, instead of a previously single serial connection, is composed of parallel connections. It uses multiple filter sizes (e.g., 1×1, 3×3, 5×5). As it uses several sizes for the filter, it processes the input at multiple scales. An interesting aspect of GoogLeNet is that the weights and activation fit into a single GPU memory. 1×1 filters are applied to diminish the number of weights by reducing channels for each filter.

ResNet [25], often referred to as Residual Net, benefits of residual connections and has around thirty-four layers. ResNet is made of one CONV layer, sixteen shortcut layers and one FC layer. It is estimated that ResNet requires "25.5 million weights and 3.9G MACs per image" [46]. ResNet comes in different depths. The winner of the ImageNet Challenge owned one hundred and fifty-two layers.

Efficientnet S, M and L introduced in 2019 [48], were chosen to be run on the USB Accelerator for many reasons. First of all, these NNs are pre-compiled models that Google made available specifically for the USB Accelerator. The other reason is that EfficientNets were recently proposed, along with a new scaling method to achieve higher accuracy.

Let us start with what scaling is and why it is essential. There are three scaling dimensions: depth, width and resolution. Depth relates to the number of layers a model owns. Width relates to how vast the network is, for instance, the number of channels in a convolutional layer. Furthermore, the resolution is the input image resolution. Figure 2.4 [48], illustrates each of these concepts. Scaling is done to improve a model's accuracy on a specific task such as ImageNet classification. If done correctly, it can also improve efficiency.

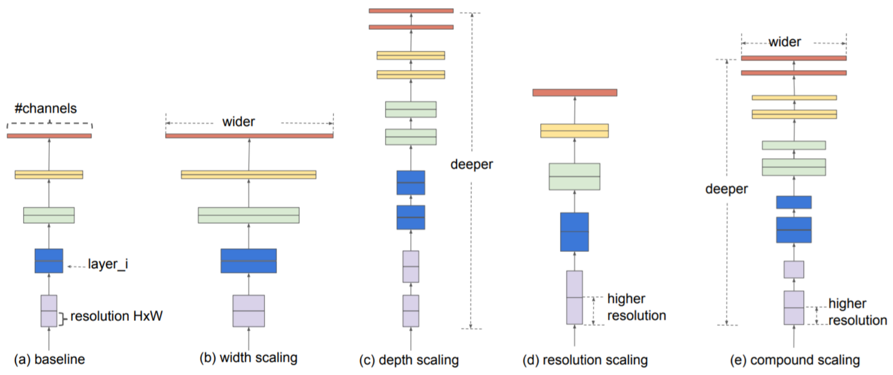


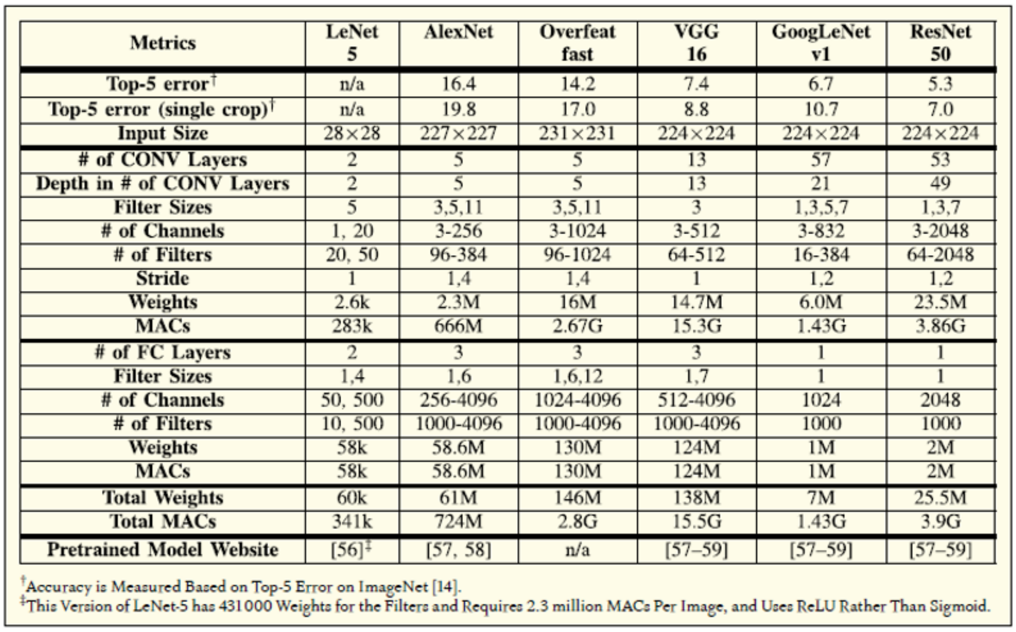
Figure 2.4: Model Scaling. (a) baseline network example; (b)-(d) conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is the proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio. Adapted from [48].

In this paper [48] the authors proposed a straightforward but effective technique to scale the network depth uniformly, width and resolution using a compound coefficient ϕ. This compound scaling method allowed to scale up a baseline ConvNet to produce an EfficientNet model that can surpass state-of-the-art accuracy while using fewer parameters and Floating-point Operations (FLOPs).

2.1.5 Popular Data Sets for Classification

MNIST was introduced in 1998 and is a commonly used dataset for digit classification. It is made of sixty thousand 28×28 pixel greyscale images of handwritten digits. When MNIST was first introduced, LeNet-5 was able to achieve a 99.05% accuracy. Since then the accuracy has increased, making MNIST a reasonably easy data set.

Table 2.1: DNN models summarized [7].



Metrics	LeNet 5	AlexNet	Overfeat fast	VGG 16	GoogLeNet v1	ResNet 50
Top-5 error†	n/a	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)†	n/a	19.8	17.0	8.8	10.7	7.0
Input Size	28×28	227×227	231×231	224×224	224×224	224×224
# of CONV Layers	2	5	5	13	57	53
Depth in # of CONV Layers	2	5	5	13	21	49
Filter Sizes	5	3,5,11	3,5,11	3	1,3,5,7	1,3,7
# of Channels	1, 20	3-256	3-1024	3-512	3-832	3-2048
# of Filters	20, 50	96-384	96-1024	64-512	16-384	64-2048
Stride	1	1,4	1,4	1	1,2	1,2
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G
# of FC Layers	2	3	3	3	1	1
Filter Sizes	1,4	1,6	1,6,12	1,7	1	1
# of Channels	50, 500	256-4096	1024-4096	512-4096	1024	2048
# of Filters	10, 500	1000-4096	1000-4096	1000-4096	1000	1000
Weights	58k	58.6M	130M	124M	1M	2M
MACs	58k	58.6M	130M	124M	1M	2M
Total Weights	60k	61M	146M	138M	7M	25.5M
Total MACs	341k	724M	2.8G	15.5G	1.43G	3.9G
Pretrained Model Website	[56]‡	[57, 58]	n/a	[57–59]	[57–59]	[57–59]

†Accuracy is Measured Based on Top-5 Error on ImageNet [14].
‡This Version of LeNet-5 has 431 000 Weights for the Filters and Requires 2.3 million MACs Per Image, and Uses ReLU Rather Than Sigmoid.

CIFAR [53] is a subset of Tiny Image data set consisting of colored images of different objects released around 2009. It holds sixty thousand of 32x32 images containing one of ten object classes, with six thousand images per class. There are fifty thousand training images and ten thousand test images.

ImageNet [41] was first popularized in 2010. It has 256×256 pixel coloured images with one thousand classes. The accuracy is calculated using two metrics: Top-1 and Top-5 error. The top-5 error means that if the first five answers provided include the correct answer, then it is considered to have provided the correct answer. The top-1 error implies that the answer provided is the correct answer.

2.2 Algorithmic Optimization techniques

Quantization and Numerical Representations is the idea of reducing the number of bits that represent a number. On small image classifications benchmarks show that quantized neural networks can attain results which can "achieve state of the art accuracy despite the reduction in precision" [7].

Pruning is an algorithm for efficient inference which drastically reduces memory requirements. The idea is to remove some of the unnecessary weights while still maintaining the same accuracy [13].

2.3 Hardware for DNN Processing

There has been a steep development in hardware platforms targeting DNNs along with new features being deployed. The goal is to minimize memory access to save power.

The Intel Knights Mill CPU will get into the DL field by introducing vector instructions into the design [28]. Nvidia Tegra, Samsung Exynos and FPGAs are an embedded system on chip (SoC) that runs inference. It is crucial to study how is the computation made to improve throughput and energy efficiency further. Multiply-and-accumulate (MAC) operations are a common component of both CONV and FC layers, which can easily be parallelized. There are two high-parallel compute paradigms that envision high performance: temporal and spatial architectures.

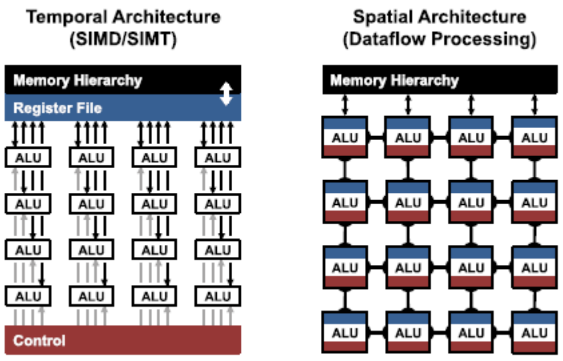


Figure 2.5: Highly parallel compute paradigms [46].

Temporal architectures are typical of both CPUs, and GPUs [46]. CPUs are latency oriented, whereas GPUs are throughput oriented. "A variety of techniques are applied to improve parallelisms, such as parallel threads (SIMT) or vectors (SIMD)"[46]. This type of architecture uses a centralized control for many ALUs. These ALUs are bound to fetch data and communicate only with Memory Hierarchy and not with each other. Spatial architectures, on the other hand, use dataflow processing which means that the ALUs can pass data one another. These ALUs that can communicate with one another and hold local memory through control logic are called as register file and are also referred to as Processing Engine (PE). Spatial architectures are most commonly used for DNN and found on ASICs and FPGAs designs.

2.3.1 Increased Efficiency in Temporal Architectures

This section discusses design optimizations for efficient computing. CPUs and GPUs use some techniques called SIMT and SIMD, which are used to process things, like MACs in parallel. All ALUs have access to the same memory (register file) and control.

The first approach is to make use of the Toeplitz matrix. It is possible to map the CONV layer to matrix multiplication in a DNN using a variant form of the Toeplitz matrix. However, the downside of this approach is that it is introduced unnecessary data into the input feature map matrix, which can lead to difficulties in accessing memory which implies inefficiency in storage.

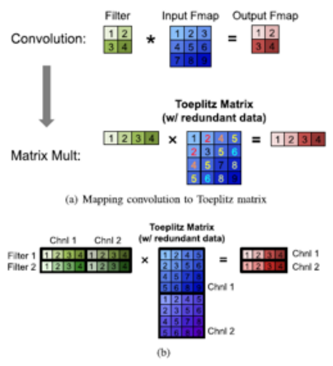


Figure 2.6: Mapping to matrix multiplication for convolutional layers. (a) Mapping convolution to Toeplitz matrix. (b) Extend Toeplitz matrix to multiple channels and filters. Adapted from [46].

Libraries that can ultimately improve matrix multiplications for CPUs are, for instance, openBLAS, and for GPUS there is cuBLAS.

Fast Fourier Transform (FFT) comprises numerous frameworks [35]. To perform the convolution, it first takes the FFT of the feature map and the filter, it then computes the multiplication in the frequency domain. Afterwards, the inverse FFT is applied to the resulting product to recover the output feature map in the spatial domain. This way reduces the number of multiplications from $O(N_o^2 N_f^2)$ to $O(N_o^2 log_2 N_o)$, in which the output size is $N_o N_O$ and the filter size is $N_f * N_f$. While FFT can reduce computation, it needs higher bandwidth and storage capacity.

2.3.2 Increased Efficiency in Spatial Architectures

For DNNs memory access is what originates most bottlenecks. In each MAC iteration, it is necessary to perform three memory reads as shown in figure 2.7 and then it is necessary to do a memory write to update the partial sum [46]. The worst-case scenario happens when all memory access is done to the off-chip DRAM, as accessing DRAM is costly and negatively impacts throughput and power efficiency.

The energy cost of internal cache access to functional operations costs about ten pJ, whereas the cost of DRAM access is about one to two nJ, which is a lot more [26]. “DRAM power consumption is crucial in contemporary computer systems as DRAM now accounts for almost half of the total system power consumption” [19]. All this is due to the growing demand for more memory bandwidth and compute power.

Making use of different levels of local memory hierarchy can greatly reduce the power cost of data movement, which is commonly done by numerous accelerators. This technique includes the introduction of a Global Buffer, a network of all PEs connected moving data between the ALU/s and a register file (RF) in each processing unit as shown in figure

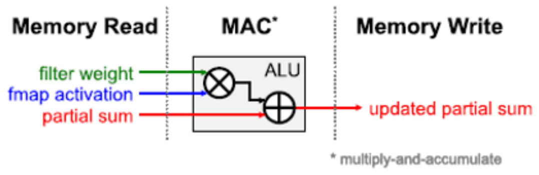


Figure 2.7: Read and write access per MAC [46].

2.8. The introduction of multiple levels of memory hierarchy can help diminish the power cost of data moving because accessing data costs less. Specifically, getting data from the RF or the neighbour PE will cost less than fetching from DRAM.

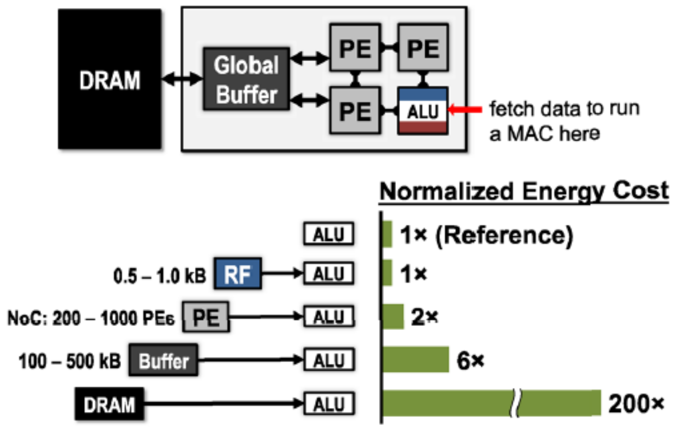


Figure 2.8: Memory hierarchy and data movement energy cost [46].

2.4 Hardware Architectures for Deep Learning

This section will present several types of hardware architectures and implementation alternatives. CPUs, GPUs, FPGAs and specialized architectures are amongst the most popular ones. There has been some research on computer architecture specialized on deep learning processing units (DPUs), and this can be implemented with FPGAs or ASICs. All these different types of architectures are organized according to: "basic type of compute operation, memory bandwidth, level of parallelism, degree of specialization and inherent precision support" [7].

CPU is latency oriented processing architecture which means that it is a serial compute engine. It tries to accomplish as many operations as possible belonging to a single serial thread. Possessing different memory hierarchies, it allows floating-point operations. The memory hierarchy splits the computer storage into a *hierarchy* based on the response time. Nowadays, CPUs are mainly multicore supporting parallel processing and can incorporate vector processing units, for instance, the Intel Knights Mill CPU. A vector processing unit is a CPU that implements an instruction set containing several instructions that operate on one-dimensional arrays of data, referred to as vectors. Depending on the workload, it can significantly improve performance.

GPUs are throughput-oriented processors. Their main aim is to maximize the total throughput of the system, rather than the latencies of all individual threads that they work on. Inherently, according to Blott, Halder, Leeser & Doyle [7] "GPUs are vector processors that support smaller floating-point formats (FP16) natively, most recently fixed point 8bit integer formats, and have a mix of implicitly and explicitly managed memory". Efficiently accessing memory is a crucial driver to fully explore the power of GPUs [37] [59]. For instance, NVIDIA's GTX980 holds a raw computational power of 4,612 GigaFLOPs/sec [1] , but its theoretical memory bandwidth is only 224 GB/s.

Field Programmable Gate Arrays (FPGAs) are reconfigurable hardware architecture that can adapt to every environment. Due to their massively parallel computing capacity are heavily demanded. FPGAs play an increasingly important role in data sampling and processing industry given their low power consumption and the fact that allow tailoring the amount of hardware parallelism necessary. There are many emerging new use cases for FPGA in the artificial intelligence field, for training and implement the neural networks and machine learning algorithms. Therefore, many companies have applied FPGAs into AI and Machine learning fields such as autonomous driving and Automatic Spoken Language Recognition. FPGAs are so flexible that they can even support bit-serial hardware architectures.

Google introduced its Tensor Processing Unit (TPU) [29] in 2016 designed to accelerate Google's Tensorflow framework. Google's TPU is a custom ASIC deployed in datacenters to specifically accelerate the inference phase of neural networks. As shown in picture 2.9, TPU's core is "a 65,536 8-bit MAC matrix multiply unit that offers a peak throughput of 92 TeraOPs/s [2] and a large (28 MiB) software-managed on-chip memory" [16]. Its design for dense matrices has more than 25× as many MACs vs GPU and more than 100× as many MACs vs CPU.

In 2016, Han et al. [23] extended their work by combining Network Pruning with quantization and Huffman encoding to create Deep Compression. Based on this deep compression, **the Efficient Inference Engine (EIE)** is a hardware accelerator that was designed to work on a compressed model. It was able to improve energy efficiency, and it achieved substantial speedup. By compressing the network through weight sharing or pruning, it was possible to fit recent networks like AlexNet in on-chip SRAM. Processing these compressed models on CPUs and GPUs is somewhat complex and challenging due to the sparse matrices and relative indices after pruning. Han et al. [23] apply a simple rule: "multiply each non-zero activation by all non-zero elements in its corresponding column" [23].

[1]GigaFLOPs/sec means Giga (10^9) floating-point operations per second
[2]TeraOPs/s means Tera (10^{12}) Operations per second

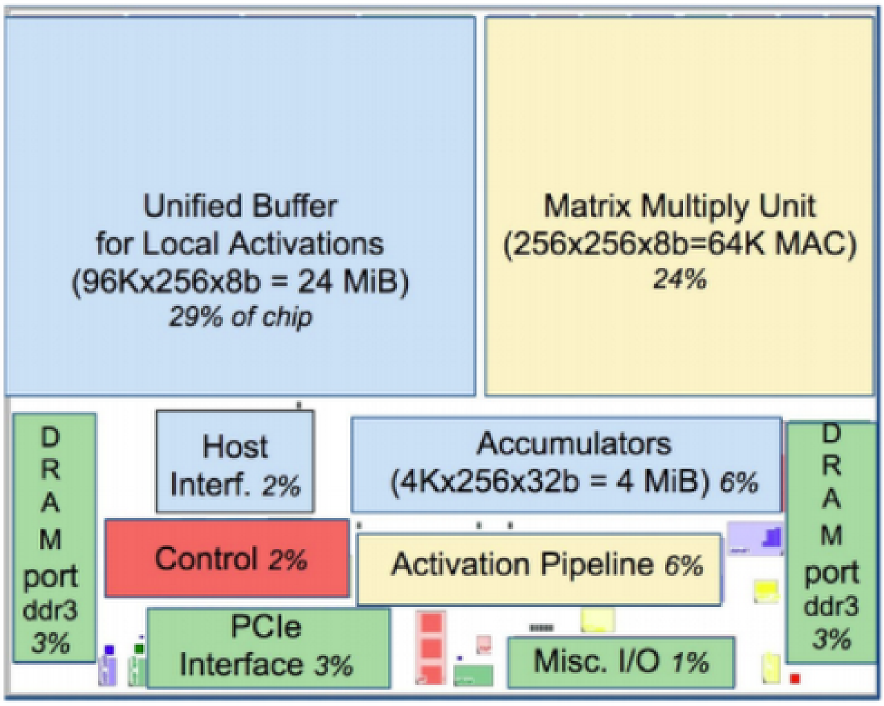


Figure 2.9: Floor Plan of Google's TPU [29].

2.4.1 FPGA: FINN and BISMO architetcures

The following subsections will explain some architectures presented by Xilinx Research Labs that was benchmarked as part of QuTiBench. These architectures were evaluated as part of QuTiBench project, and several results were obtained; thus, it is important to explain them beforehand.

2.4.1.1 BISMO

BISMO [57] and [56] is a software-programmable scalable bit-serial matrix multiplication overlay that can be instantiated on an FPGA. In other words, BISMO is a programmable FPGA accelerator for few-bit integer matrix multiplication. It can achieve high performance for matrix multiplication where each matrix element is an integer of 2,3,4,.. bits. This architecture is beneficial for applications like quantized neural networks. It was developed as part of a collaboration between Xilinx Research Labs Ireland and the Norwegian University of Science and Technology Computer Architecture Lab. Its hardware is presented in figure 2.10. BISMO is composed both by a "hardware part and a software part. The hardware part is composed of a scalable bit-serial matrix multiplication datapath with the associated memory and control logic. The software part creates instructions for the hardware for the given matrix size and precision" [57].

It presents a few advantages, such as the fact that it supports several different precisions while performing matrix multiplications. The same hardware can be leveraged for

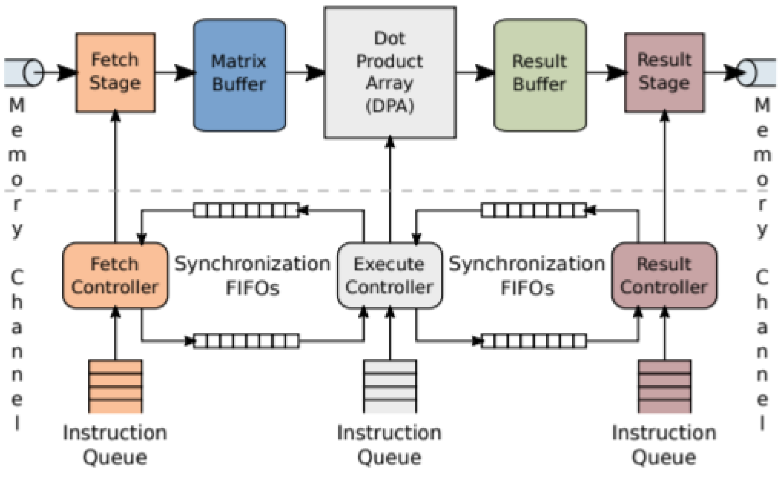


Figure 2.10: Overview of BISMO's hardware architecture [56].

many use cases, achieving higher speed with the lower precision matrix multiplication than with the higher precisions.

2.4.1.2 FINN

FINN ([55] and [6]) is an experimental framework built at Xilinx Research Labs which explores quantized neural networks inference on FPGAs. It is essentially an open-source framework for building Binarized Neural Networks (BNN) inference accelerators on FPGA. It is very flexible as it allows to dataflow-style architectures customized for each model. It is ideal for embedded applications as it can perform millions of classifications per second.

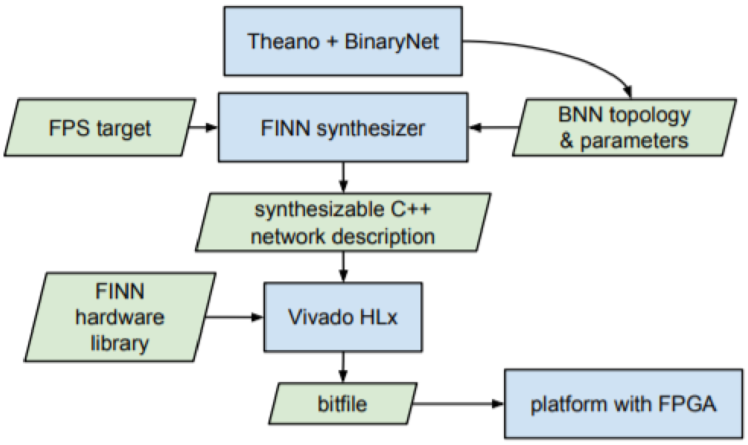


Figure 2.11: Overview of FINN flow for a BNN

Figure 2.11 shows how an FPGA accelerator can be built from a trained BNN with FINN. Everything starts with the user supplying a Theano-trained (Theano refers to a machine learning library) BNN to the FINN synthesizer, which in turn begins with determining the needed parameters that can meet the target fps and applies specific optimizations. After, it produces a synthesizable C++ network description which can then be loaded into Vivado HLS and deployed to an FPGA.

2.4.2 Coral USB Accelerator

The Coral USB Accelerator is a small PCB that contains an Edge TPU coprocessor and a USB socket to connect to the computer. It is capable of performing 4 TeraOPs/s [3], able to perform ML inference and lightweight transfer learning, but not training.

Installing the API library and the Edge TPU runtime on the host computer allows working with the USB accelerator. At this point, it is possible to select between the default clock frequency or the maximum clock frequency at which the device will operate. The maximum clock frequency is twice the default clock frequency which is translated in increased inference speed but also power consumption.

The ML runtime used to execute models is based on TensorFlow Lite. As EdgeTPU is only able to perform 8-bit math, as such, models need to be in a specific format, which means that the network needs to be trained using TensorFlow quantization technique. This technique ensures that the forward pass matches the 8-bit precision for both training and inference. The Edge TPU runtime is required to communicate with the Edge TPU and can be installed on a Linux, Mac or Windows computer host.

2.4.2.1 Context

In 2019 Google released TPU hardware to the market under the Coral brand. However, these are not as powerful as google's cloud TPUs which can run a NN at more than one hundred PetaFLOPs/sec [4]. These recently released TPU devices are meant to be used "at the edge".

Edge Computing means that the computation and data storage runs on local places like a computer, embedded system, IoT device or Edge Server. In contrast, Cloud Computing means that computing and data storage is done in the cloud. The access is done through the internet instead of the computer's hard drive. So data needs to be sent through an internet connection, then wait for the server to process everything and then it sends the results back. This process can be quick on a wired connection, but on bad connections can take up to a few seconds. Edge Computing avoids long-distance communication between a client and a server where bandwidth and latency are affected. This characteristic is important because we humans like to interact with fast things, more precisely, everything that runs faster than one hundred milliseconds.

Edge AI means that AI algorithms are running on a device that uses Edge Computing and all data needed is physically present on the device with no need for fetching data from the cloud. What usually runs at the edge is only inference whereas training is still

[3]TeraOPs/s means Tera (10^{12}) Operations per second

[4]PetaFLOPs/sec means Peta (10^{15}) floating-point operations per second

done on servers. However, it is possible to train the last layer of a model on this device. The inference is made with a lightweight version of TensorFlow models which have to be converted to this file type which is more power-efficient. Depending on the application type and especially on the AI workloads hardware options go from CPUs, GPUs, ASICs, FPGAs and SoC accelerators.

Some of the Advantages of Edge AI are reduced cost, security, increased speed, and easy to manage for beginners. Some examples of applications are surveillance and monitoring, autonomous vehicles, smart speakers and industrial IoT.

2.4.2.2 Edge TPU Processor

Being an ASIC, the Edge TPU is an Integrated Circuit (IC) chip customized for a very particular use rather than a general-purpose use. It combines small electronic circuits, of which field-effect transistors (FETs) and capacities, burned directly on the silicon layer. 5. The logic behind it is simple, and image 2.12 shows the basic principle upon which the Edge TPU was built.

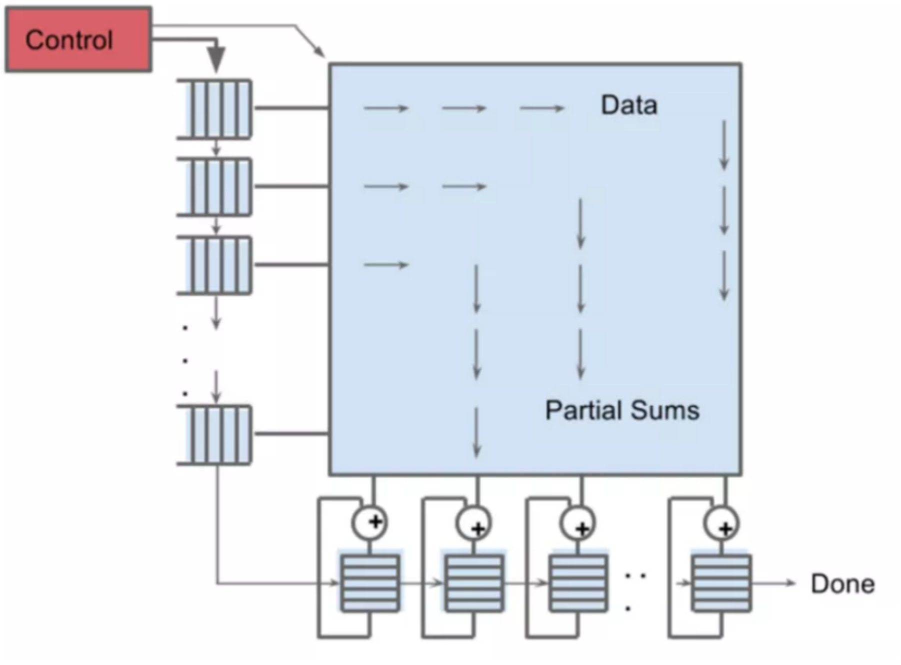


Figure 2.12: Matrix Multiplier + summation [36].

Many neural networks consist mainly of convolutional layers. A convolution is defined by the following equation:

$$(f \times g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

Each image pixel multiplies by each kernel pixel and then it is all summed up to create a new “image”. This type of arithmetic is precisely the reason why the Edge TPU was created. Whenever data is fed to the buffers, parallel multiplication followed by summation is done at very high speeds.

Being programmed for specific workloads, the Edge TPU can perform up to 4 TeraOPs/s [5] using 8-bit fixed point, and bandwidth capacity lies between 500 MB/s and 640 MB/s. Its size is about a fourth of a penny, and it only needs 500mA 5V USB port which means that its power consumption is at most 2.5 Watt. It is fast because it was greatly optimized for a specific application, unlike a CPU which can do almost everything. An ASIC is specially designed to perform multiplications so it can complete them in a very short time.

2.4.3 Intel NCS2

The Intel Neural Compute Stick 2 is Intel's deep learning inference development kit. It is packed in an USB-stick form factor and is powered by the VPU (vision processing unit) – the Intel Movidius Myriad X, which includes an on-chip neural network accelerator called the Neural Compute Engine. With 16 SHAVE cores and a dedicated hardware neural network accelerator it's set to deliver 4 trillion operations per second. Intel Distribution of OpenVINO Toolkit supports Ubuntu, CentOS, and Yocto Linux distributions along with Microsoft Windows and Raspbian 32-bit OS. OpenVINO Toolkit includes tools for generating an Intermediate Representation (IR) from TensorFlow, Caffe, and Apache MXNet models. It also supports Open Neural Network Exchange (ONNX) for importing and exporting deep learning models across multiple frameworks.

2.5 Characteristics and Challenges in Benchmarking

A Benchmark is a process of measuring the performance of a particular product, service or process, against those of another business. According to Blott, Halder, Leeser & Doyle [7] a more practical definition would imply that a benchmark is a "well-defined set of executable tests and measured regarding a specific set of figures of merit". Sometimes it can analyze some system aspects which can clarify system bottlenecks. The main objective is to identify opportunities for further improvement. Marketing also plays a part when it comes to benchmarking. It is important for a company's products to do well on a benchmark, otherwise it will lose popularity.

For software system designers it can identify which algorithms lead to better performance. For hardware designers, it can clarify what aspects of the architecture need improvement. As neural networks' development is progressing faster than ever, co-design of hardware and algorithms becomes crucial. Some key elements need to be taken into consideration to ensure a good benchmark.

1. Benchmarks need to be **relevant** to the user;

[5]TeraOPs/s means Tera (10^{12}) Operations per second

2. They need to be **objective and repeatable** in terms of the same results and time. Nevertheless, what does repeatable mean? Does it mean that it is enough for a third-party to validate the results? This is hard mainly due to the hardware cost. Benchmarking results have to be objective in order not to favour any specific system or hardware configuration [7];
3. Benchmarks need to **represent** true workloads in terms of applications, algorithms and computational patterns;
4. Benchmarks need to support **algorithmic optimization**. Researchers making DL computing cannot rely entirely on hardware. Combining hardware optimizations with software optimizations techniques such as model compression can lead to energy-efficiency and performance improvement;
5. **Portability** needs to be taken into consideration when it comes to benchmarking.
6. **Complexity vs Speed vs Accuracy**: The aim is to have a benchmark which is low in complexity, very high in speed and accuracy. However, these three criteria influence one another. To lower a benchmark complexity, there is a need to consider the speed and accuracy of the output;
7. **Adaptive**: The benchmark suite has to be adaptive because the ML research field is evolving faster than ever, so it can keep up with the emerging algorithms.

2.6 Existing Benchmarking

This section will distinguish three different benchmarking types: "**ML benchmarks, performance benchmarks** and **NN system benchmarks**" [7]. ML benchmarks are driven by the improvement on accuracy, which improves the overall system performance, with no regards to power efficiency, computing workload or execution time. "Performance benchmarks record hardware performance only, specifically throughput (measured in processed inputs per time or TOPs/s), latency or response time in milliseconds (ms), and power consumption in Watts" [7]. NN system benchmarks take both things into account, hardware performance as well as accuracy.

2.6.1 NN System Benchmarks

BenchIP [50] presents a Benchmark suite for intelligence processors, and contains two types of benchmarks: micro-benchmarks and macro benchmarks. The aim is to identify performance/energy bottlenecks for potential optimization. However, it does not provide an overall score. It also does not cover the stack of layers, which is important to isolate bottleneck in data movement. It also does not provide the comparison via Pareto curves.

Fathom [1] assembled a collection of eight archetypal DL workloads to be studied. It determines where time is spent and studies the effects of parallelism on scaling and focuses on understanding deep learning workloads. Nonetheless, it is rather limited at benchmarking. It contains CONV and FC layers but lacks Deconvolution, Unpooling and Batch Normalization. It does not provide a benchmarking method for customized hardware architectures. In the customized hardware field, Fathom does not process fine-grained architectures' performance.

TBD [60] proposes a benchmark suite for DNN training, it holds a set of eight DNN models covering six ML applications. It comprises numerous frameworks, more than Fathom, and numerous applications.

MLPerf [38] aims to provide a comprehensive benchmark suite for training as well as inference over ML hardware, software and services. MLPerf began in February 2018 with several meetings between engineers and researchers from Baidu, Google and some universities from the USA. MLPerf provides insights on how to build a solution. However, it does not address hyper-parameter optimizations.

DAWNBench [10] is a project led by Stanford University, and it allows us to compare many DL methodologies which are done by creating competitions. It introduces a benchmark focused on training time to achieve better accuracy. Unlike prior work that focuses solely on throughput metrics, this approach focuses primarily on a combined metric called time-to-accuracy (TTA), which assesses how long it takes to train for a target. However, it only considers ImageNet for both training and inference, making it limited in the application scope, and it does not provide insights into the full design space.

Collective Knowledge Framework [11] in conjunction with the **ASPLOS Request Tournament** [3] give space for different hardware accelerators while providing an assessment of correlations between accuracy, performance and power trade-offs. It also supports heterogeneous hardware architectures.

QuTiBench [7] is the benchmark proposal in which this work is inserted, so this was the benchmark used in this work. It will be discussed in details later in section 2.9.

2.6.2 ML Benchmarks

Machine Learning is fuelled by the insatiable demand of accuracy in generalization; it completely discards the computation load. One of the most recent considerably popular is the **ImageNet Challenge** [41] that consists of a computer vision competition in which a computer vision dataset called ImageNet is used. The evaluation is done through the calculation of an error-rate with no regards to compute power or energy consumption. **CortexSuite** [52] details a benchmark suite inspired by the brain. Using the cerebral cortex' lobes to organize and classify data, it analyses performance bottlenecks identifying the parts of the program mostly active during runtime with the increasing of data. However, it only analyses the algorithm's total runtime and is limited to measuring accuracy. **BenchNN** [9] present a benchmark suite to encourage research on hardware neural network accelerators. Although it emphasizes the potential of NNs, it looks more like a symbol rather than a benchmark suite. BenchNN provides only five NN applications, and the NNs used are classic models, like multi-layer perceptrons which are not very popular nowadays. **DjiNN and Tonic** [24] is an open infrastructure for DNN that supports several applications and NN architecture. It introduces seven end-to-end applications that use the DjiNN service, which are quite representative of the emerging workloads, such as computer vision, speech recognition and language processing systems. **MLBench** [34] presents a novel dataset to benchmark ML systems and services. It presents an example use-case by conducting a study of ML services like Amazon and Microsoft Azure.

2.6.3 Performance Benchmarks

DeepBench [15] is an open-source benchmarking tool that measures performance on a lower level, namely basic operations involved in training deep neural networks on distinct hardware platforms such as direct convolutions. DeepBench also provides insights in terms of which hardware provides the best performance on the specific operations. Most of the popular compute patterns are covered, but it does not support lower precision data types. It does not support data movement bottlenecks between layers. There are other hardware benchmarks such as **TPC** [54], which measures the performance of decision-support solutions targeting data processing communities. **SHOC** [14] consists of a benchmark suite targeting heterogeneous computing. It makes use of microbenchmarks to assess low-level architectural features of the system using OpenCL as design entry. However, SHOC's focus is on systems containing Graphics Processing Units (GPUs) and multicore processors. **SPEC** [44] is a non-profit corporation that endorses benchmarks and tools to evaluate performance and energy efficiency for the most recent generations of computing systems. It covers a vast range of applications including mail servers, MPI, virtualization, storage and graphics. **STREAM** [45], on the other hand, solely focuses on memory bandwidth in high-performance computers.

In conclusion, very few benchmarks include algorithmic optimizations such as quantization and pruning. None of the benchmarks gives proper insights on both hardware and software optimizations.

2.7 Rooflines

2.7.1 Context

Roofline models will be used throughout the entire document. Thus a brief description of what they are and what they consist of will be made.

There are multiple products on the market, each with a different number of cores that adds up to different trade-offs between performance and cost. However, such diversity, although it is suitable for competition, it complicates programmers and architect's job. For this reason, a need for an easy-to-understand model that offers insights on performance has risen. There are already models that predict multiprocessors performance accurately

[17], however, they hardly ever provide insights on how to improve performance, or they can be of problematic use for non-experts.

2.7.2 What is it?

A roofline is “an easy-to-understand, visual performance model that offers insights on improving parallel software and hardware for floating-point computations” [58].

2.7.3 The Roofline Model

The proposed model is a simple one because it does not attempt to predict performance. Instead, it performs bound and bottleneck analysis. It was specially designed to consider the off-chip memory traffic because off-chip memory bandwidth can often be the constraining resource. In figure 2.13 it is depicted the Roofline model for a 2.2 GHz AMD Opteron X2 model. The Y-axis represents the attainable floating-point performance, whereas the X-axis represents the operational intensity. Operational intensity means operations per byte of DRAM traffic. What is measured is the traffic between caches and DRAM instead of traffic between caches and processor. In Figure 2.13, operational intensity varies from ¼ FLOPs/DRAM byte accessed to sixteen FLOPs/DRAM byte accessed. The modelled system has a peak floating-point performance of 17.6 GigaFLOPs/sec and a peak memory bandwidth of fifteen GBytes/sec. As the X-axis is $GigaFLOPs/byte$ and the Y-axis is $GigaFLOPs/second$ then the line at 45 degrees is $bytes/second$ – which is equal to $(GigaFLOPs/second)/(GigaFLOPs/byte)$. This line shows the maximum floating-point performance that the computer can support for a given operational intensity. The plot is driven from the following equation:

$$AttainableGigaFLOPs/sec = Min(Peak\ Floating\ point\ Performance,$$
$$Peak\ Memory\ Bandwidth \times Operational\ Intensity)$$

This plot is created for a multicore computer and not for a single kernel. For a given kernel, based on its operational intensity it is possible to see the performance of that kernel on that computer. The way to do this is, based on its operational intensity, we find a point on the X-axis and see what the attainable floating-point performance is. Based on this, we can get insights on whether the kernel performance will be compute-bound if it hits the flat part of the plot or memory-bound if it hits the sloping part of the plot. In figure 2.13, it is possible to see that a kernel with an operational intensity of 0.75 will be memory bound, and a kernel with an operational intensity of 4 will be compute-bound. The turning/ridge point also provides meaningful insights. Its X-coordinate is the minimum operational intensity at which the kernel achieves maximum performance. If this point is far to the right, then only kernels with high operational intensity will achieve maximum performance. In contrast, if it is far to the left, then kernels with lower operational intensity can also reach maximum performance.

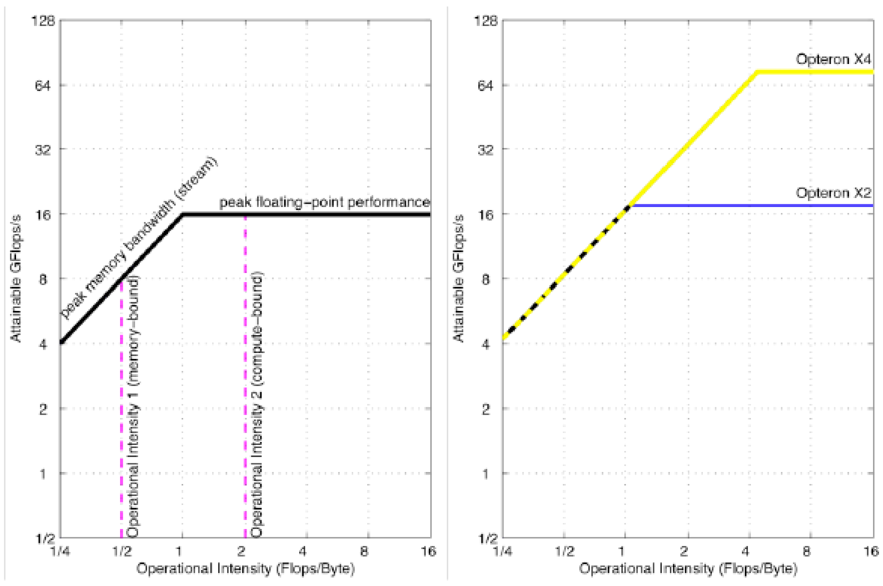


Figure 2.13: "Roofline model for (a) AMD Opteron X2 on left and (b) Opteron X2 vs. Opetron X4 on the right" [58].

2.7.4 How to improve performance

Suppose our program is performing far below the Roofline. How can it be optimized? The objective is to reduce computational bottlenecks along with memory bottlenecks.

It is possible to reduce computational bottlenecks by:

- "Improve Instruction Level parallelism (ILP) and apply SIMD" ([58]): Performance is increased when the number of instructions is increased. So improving the compiler code to increase ILP is an option. One way would be to unroll loops. For x86 architectures, one way of improving ILP would be to use floating-point SIMD instructions. These hardware components capable of performing SIMD instructions can perform the same operation on multiple data operands concurrently. Two input vectors are provided, the same operation is done on adjacent operands, and an output vector is provided with the results inside.

- "Balance Floating-point operation mix" ([58]): Due to many computers having an equal number of adders and multipliers, there is a need to have the same amount of floating-point additions and multiplications. This way, all resources can be fully utilized at the same time.

It is possible to reduce memory bottlenecks by:

- "Restructure loops for unit stride access" ([58]): which means accessing memory elements that are next to each other, which is desired because when they are non-sequential, the number of cache misses increases, and time is wasted for data to be transferred from the main memory (DRAM) to cache.

- “Memory affinity” ([58]): Many microprocessors nowadays have an on-chip memory controller. In a system with two multicore chips, some addresses are in the DRAM local to one multicore, and the rest is over an interconnect inside the other chip. The aim is to have all data needed inside the DRAM local to one multicore.

- “Use software prefetching” ([58]): Cache prefetching is a technique used by processors to improve memory access bandwidth. It consists of fetching instructions or data in chunks from slow main memory to a fast cache memory before it is needed. Accessing cache is much faster than accessing main memory because it was specially built for this, it is small (comparing to main memory) and it is fast. So, software prefetching is doing this with software. Sometimes it is even quicker than hardware prefetching.

2.8 Data Visualizations

Data visualization is a trending topic in the sense that before the digitalization era, data went from challenging to find and expensive to being abundant and cheap. What is difficult now is to process it in meaningful ways and understand it. Terabytes of unused data are sitting in data centres, but if it is correctly processed, it can become digital gold. This step is where data visualization comes in. Data visualizations consist of taking the raw data and transform it into all sorts of charts, graphs or even videos which explain the numbers and provides us with easy-to-understand insights on it. It comes down to making data visually more understandable. In this “age of Big Data” data visualizations is a powerful tool with which stories can be created out of the trillions of rows of data generated every day. However, this story is not an easy one to make, and a good visualization involves highlighting trends, removing noise and detecting outliers.

There are a huge plethora of ways to present data effectively, and interestingly, the one that will be used in throughout this work are: Rooflines; Heatmaps; Line plots; Bar plots and Box and Whisker plots.

Rooflines were explained in more details in section 2.7.2. Heatmaps are graphical representations of data where all data is represented not through a number but as a colour. Line plots are one of the simplest ways to display data. A Line plot can be defined as a graph that displays data as points above a number line, showing the frequency of each value. Bar plots display data using several bars of different height. It is an excellent way to show relative sizes/quantities, and it is not as good with continuous data; however, for that, a histogram would be more adequate. A histogram is used for continuous data, where the bins represent ranges of data, while a bar chart is a plot of categorical variables. Box and Whisker plots are a more complex way to display data; they are based on statistical information. Box and Whisker plots show groups of numerical data through quartiles. They can also have lines extending from the box, which indicate variability outside the upper and lower quartiles.

2.9 Previous implementation: QuTiBench

This section aims at describing the benchmark proposal and all its related considerations. Soon after it will describe some results that were achieved with this proposal.

"*QuTiBench* is a novel multi-tiered benchmarking methodology *(Ti)*" [7], it supports quantization *(Qu)*, which is an algorithmic optimization, and helps researchers understand in more depth the limitations and benefits of the underlying architecture. Its multi-tiered approach (fig:2.14) introduces the concept of abstraction levels, by subdividing the benchmark suite into these. For both inference and training, it supports several numerical representations which bring system-level insights.

By pairing hardware performance with accuracy it broader distinguishes itself from existing benchmarking. Specifically, reduced precision models are compared with floating-point implementation, which allows for higher performance at considerably lower power consumption. Optimal solutions can be visualized with Pareto graphs which will display the results.

A theoretical level will be introduced, which will allow performance estimation.

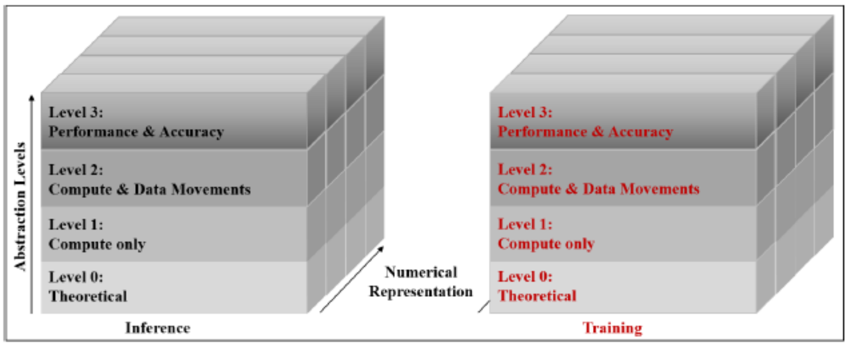


Figure 2.14: Illustration of the multi-tiered approach proposed by the benchmark methodology - QuTiBench [7].

QuTiBench is constituted by several test suites on every abstraction level. It also comprises quantization and algorithmic optimizations, namely awareness regarding datasets, framework challenges and hyperparameters, such as reproducibility and adaptability.

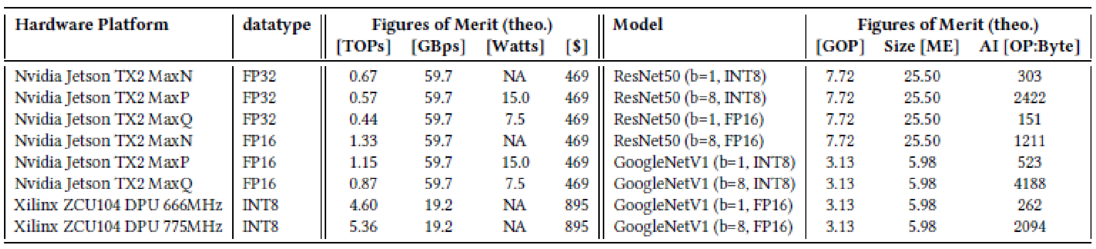
Multiple tiers – Abstraction levels It will be described all four levels of abstraction.

Level 0 - Theoretical Level 0 is a baseline that provides instant results. It introduces the awareness of performance for each data type operation, which is crucial to algorithmic optimizations such as quantization. Two tables are presented (table 2.2), the first presents characteristics of the hardware and the second presents the characteristics of NNs. The hardware table lists different hardware platforms and supportive native datatype.

In the neural networks table for each model of CNN, there are four values recorded: "total number of compute operations for a single input, the model size, the size of the state and the total amount of tensors in between layers that require buffering. These values can be used as a basis to derive memory requirements and compute requirements for both inference and training" [7].

All these values can help with seeing memory and compute requirements for inference as well as training. Assuming that weights, weight updates, tensors and gradients will be held somewhere in the hardware platform and knowing the size of the datatype, it is possible to estimate the intensity of the calculations during training and inference. Determining the estimate and knowing the Roofline of a given hardware platform makes it possible to draw insights on the bound type, whether it is a memory or compute-bound.

Table 2.2: Hardware platforms and neural networks model [7].



Hardware Platform	datatype	Figures of Merit (theo.) [TOPs]	[GBps]	[Watts]	[$]	Model	Figures of Merit (theo.) [GOP]	Size [ME]	AI [OP:Byte]
Nvidia Jetson TX2 MaxN	FP32	0.67	59.7	NA	469	ResNet50 (b=1, INT8)	7.72	25.50	303
Nvidia Jetson TX2 MaxP	FP32	0.57	59.7	15.0	469	ResNet50 (b=8, INT8)	7.72	25.50	2422
Nvidia Jetson TX2 MaxQ	FP32	0.44	59.7	7.5	469	ResNet50 (b=1, FP16)	7.72	25.50	151
Nvidia Jetson TX2 MaxN	FP16	1.33	59.7	NA	469	ResNet50 (b=8, FP16)	7.72	25.50	1211
Nvidia Jetson TX2 MaxP	FP16	1.15	59.7	15.0	469	GoogleNetV1 (b=1, INT8)	3.13	5.98	523
Nvidia Jetson TX2 MaxQ	FP16	0.87	59.7	7.5	469	GoogleNetV1 (b=8, INT8)	3.13	5.98	4188
Xilinx ZCU104 DPU 666MHz	INT8	4.60	19.2	NA	895	GoogleNetV1 (b=1, FP16)	3.13	5.98	262
Xilinx ZCU104 DPU 775MHz	INT8	5.36	19.2	NA	895	GoogleNetV1 (b=8, FP16)	3.13	5.98	2094

Level 1 – Compute Patterns This level reports the attainable compute performance for common neural network layers, such as fully connected, recurrent, residual, squeeze and convolutional layers. Being analogous to Deepbench, the main difference is that *QuTiBench* provides a much broader scope onto specialized numerical representations. "For each of these compute patterns, and for both inference and training, we record the following figures of merit: measured performance (TOPs/s [6] or GOPs/s [7]), latency (ms), power consumption (Watts) of the full platform in the embedded space, and the board excluding the host system in the cloud" [7]. Even though level 1 does not provide insights about accuracy at the application level, it verifies that the system is functioning correctly. Test speed is relatively high, and it considers thread and batch size.

Level 2 – Compute and Data Movement This level computes possible combinations of tests done by level 1, and with that, it provides insights on potential bottlenecks and storage requirements. Figures of merit are the same as indicated for level 1.

Level 3 – Applications Level 3 provides application coverage, as it becomes possible to optimize algorithms which can attain system-level performance and can only be confirmed with results provided by the application. This level includes the initial planned dataset and models that perform correctly with quantization and pruning. For inference, performance measures are included for a single image. For training a single image, this level reports throughput, power and latency. It is possible to perform measurements with a specific accuracy target. Finally, it is also possible to optimize the algorithm and the network itself with the possibility to save data point in a graph.

Regarding **quantization** this methodology supports algorithmic optimizations such as pruning and quantization. Thus, on every level, there are included all the next numerical

[6]TOPs/s means Tera (10^{12}) Operations per second
[7]GOPs/s means Giga (10^{9}) Operations per second

representations such as “FP32 [8], FP16, INT8, BIN, TERN, and allow for arbitrary choices to be included, for example, Microsoft’s custom floating-point” [7].

Frameworks and Datasets Datasets are very important because they directly impact accuracy. Providing framework support is a significant challenge because all frameworks are linked with a specific neural network. Thus, each framework may need its dedicated hardware backend which may not be available.

Power and Energy This benchmarking methodology represents power measured at the socket. Although it is not entirely accurate, this was chosen because measurements need to be fair and so measuring subsystems, including memory, needs to be considered. Power measurements at a socket have an accuracy of around 10%, so an average over ten measurements is performed to address this issue.

2.9.1 Experimental Results and Evaluation

This section presents results that will help evaluate the proposed benchmarking methodology and figures of merit.“For both platforms, we carried out all levels of tests on one specific Machine Learning task, ImageNet classification, for two different neural networks, GoogleNetV1 and ResNet50. We use FP32, FP16 (supported by GPU), and INT8 (supported by FPGA) as numerical representations, a form of algorithmic optimization. We run GPU platforms with a spectrum of batch sizes and different operating modes (MaxN, MaxQ, MaxP), which are optimized for different performance and power consumption targets6. For FPGAs, there is a spectrum of implementations available. We exercise the Deephi DPU overlay, which uses threads instead of batch sizes to achieve high system utilization, and therefore exercise a spectrum of thread counts. For FPGAs, we show the theoretical limits of the current implementation (which is clocked at 666MHz), as well as the datasheet peak performance of 750MHz. For GPUs, we use the theoretical peak as dictated by the clock frequencies defined by the operating mode” [7].

For all levels of the benchmarking methodology was done a critical review as follows.

Level 0 In figure 2.15 it is possible to see that GoogleNet and ResNet50 are compute bound for FP32, FP16 and INT8. FPGAs can achieve much greater performance for bigger operational intensity. The larger the batch size the larger the operational intensity, however the estimated performance for bigger batch sizes is similar.

Level 1 and 2 In figure 2.16 it is possible to see the performance of the different layers for ResNet50. This neural network was chosen due to its regularity in structure as it consists of a convolutional layer followed by pooling combinations, a fully connected layer and sixteen residual blocks. Even though the compute requirements in each layer are similar, there is a big range in the throughput of each layer, which means that some of them are taking a lot longer execution times. The variation in data movements can explain this particularity, which is likely to occur for other types of NNs as ResNet50 has

[8]FP32 refers to a floating-point precision of 32 bits which means that there are 32 bits or 8 bytes used to store decimals

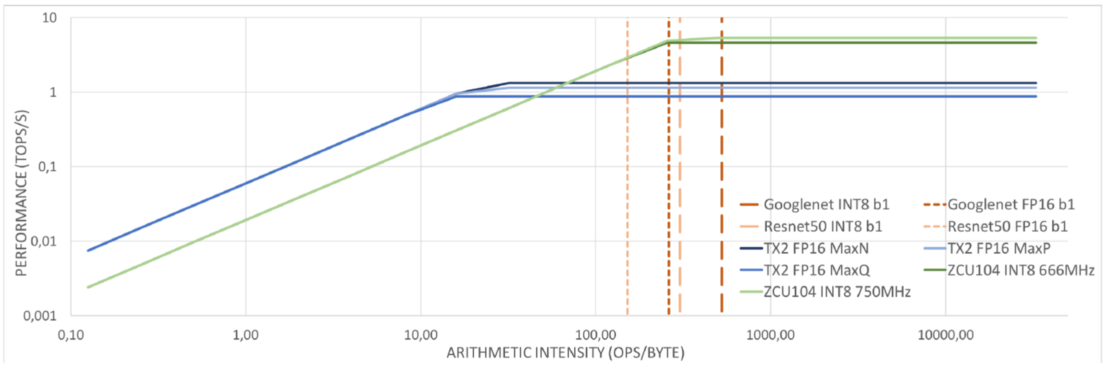


Figure 2.15: Rooflines provided by level 0 for GoogleNet and ResNet50 on TX2 and ZCU104 [7].

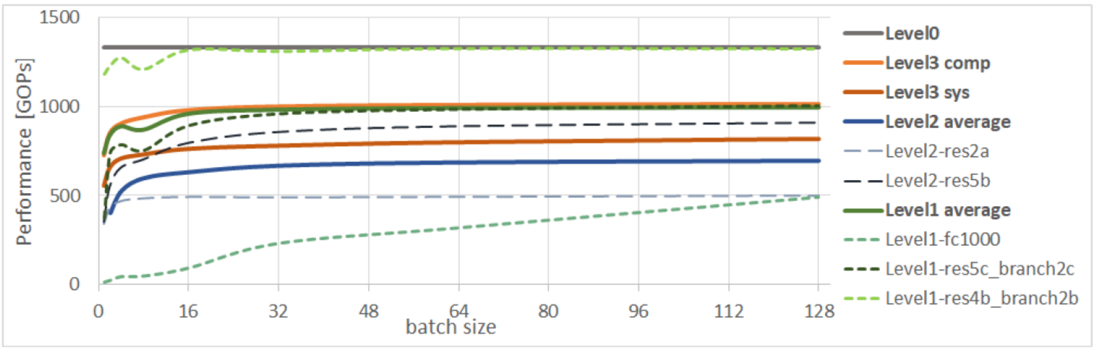


Figure 2.16: "Performance comparison layer0, layer1, layer2 and layer3 for TX2 (MaxN, FP16 configuration)"[7] for ResNet50.

a more balanced topology. It is also visible that different convolutional layers have very different performance, especially as the batch size increases. This difference is because compute requirements differ from one another.

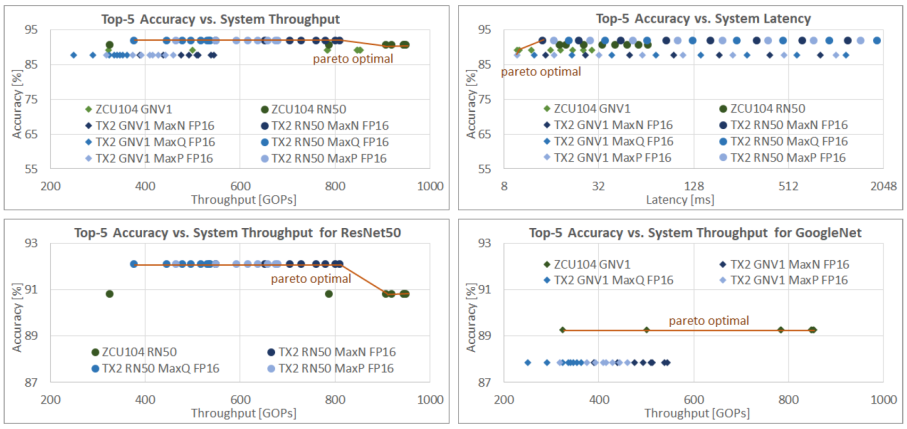


Figure 2.17: “System Performance evaluation” [7] for TX2 and ZCU104 for INT8, FP16, FP32, for both GoogleNetV1 and ResNet50 and for different batch sizes [7].

Level 3 This level explores possible optimization solutions. Results show (fig.2.17) that the FPGA has the highest performance both system and compute level (948GOPs/s and 1067GOPs/s). In comparison, the GPU shows a system performance of 809GOPs/s and a compute level performance of 1011GOPs/s. In figure 2.17 it is possible to see that the ZCU104 has considerably smaller latency compared to the TX2. The ZCU104 also has better performance and accuracy compared to TX2. Regarding ResNet50 although the FPGA achieves better performance, the GPU achieves better accuracy under worse performance.

Summarizing QuTiBench is a fresher benchmarking methodology to give a clearer vision on hardware innovation and algorithmic optimization. This work is in its early stages, and future work will consider more in-depth development and experimentation. The objective is to, firstly, further develop level0 and then building test suites targeting CPUs, FPAs and DPUs.

CHAPTER 3

Development: Benchmarking Hardware

The following chapters will describe the work completed during the internship at Xilinx, Inc, which included benchmarking a hardware platform, creating visualizations to understand better all the present data that there is and deployed them to a web portal that was made.

This chapter will focus on describing the work done on benchmarking Google USB Accelerator.

3.1 Coral USB Accelerator

The hardware platform that was benchmarked was Google's USB Accelerator, and the neural networks that were benchmarked on this hardware platform were: EfficientNet Small, EfficientNet Medium and EfficientNet Large. All models that are to run inside the USB Accelerator need to be on a 'tflite' format. So it is possible to download these models in this format or convert from another format to tflite format. EfficientNet S, M and L can be downloaded from Google Coral webpage directly, specifically from here [12].

3.1.1 Data acquisition

To start, all necessary software that comes with the USB accelerator needs to be installed to be later used. The EdgeTPU runtime was installed which provides the interface with the Edge TPU and it can either be installed on Linux, on Mac, or Windows. It was installed on Windows. Along with the installation of the Edge TPU runtime, there is an option to install either the slow or maximum operating frequency. Both modes were installed. There was also the need to install the tflite_runtime library. By cloning the tflite repository from google-coral GitHub [51], there are a couple of getting started with the USB Accelerator guides.

The figures of merit that were measured on the USB Accelerator were performance, power consumption and accuracy. This was done with a python script which is inspired on this script [20], that directly interacted with the *ClassificationEngine*. These scripts were not created as part of this work but were modified while taking measurements. The USB Accelerator only supports batch sizes equal to one, which means that it can only perform inference over one image at the time.

A block diagram between the USB Accelerator and the computer is shown in picture 3.1. The USB Accelerator connects to the computer via a USB Type-C cable. This is a very short cable (about 31 cm) specially designed to have very low resistance.

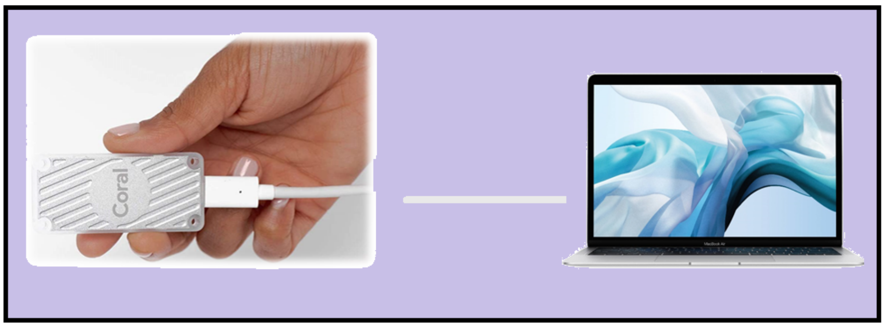


Figure 3.1: Block diagram between the USB accelerator and the computer.

3.1.1.1 Performance Measurements

Two types of performance were measured, system performance and computing performance, both illustrated with pseudo code. System performance (algorithm 1) accounts for opening the image, creating an input tensor from it and calling inference. Whereas computing performance (algorithm 2) only accounts for the time the inference call takes. There is no need to measure performance on the entire 50000 images, only on a few thousands.

Upon starting the inference the required arguments are the following:

- Model;
- Validation set: Folder with image(s) to run inference on;
- Path to the text file which holds all 1000 categories listed.

3.1.1.2 Accuracy Measurements

The accuracy measurement is done over the entire test set, which is made of 50000 images. The pseudo-code shown in algorithm 3 creates a file with results. All images were preprocessed before being fed to the USB Accelerator, in particular, images were resized to 256×256 and then cropped to 224×224, which is the recommended input size. For

Algorithm 1 Algorithm for benchmarking system performance.

```
Get the input arguments (path to model, path to the labels and path to pictures to test on).
Initialize the engine.
$Accumulated\ time\ = 0$
for $Each\ image\ldots$ do
    Get current (start) time.
    Open the image.
    Preprocess the image (resize and crop).
    Flatten the image into an input tensor.
    Run inference over the image.
    Get current (end) time.
    $Time\ [ms] = (end\ time - start\ time) * 1000$
    $Accumulated\ time\ [ms] = Accumulated\ time + Time\ [ms]$
end for
$Average\ System\ Latency\ [ms/img] = \frac{Accumulated\ time\ [ms]}{Number\ of\ Images}$
$System\ Throughput\ [GFLOPs/sec] = GFLOPs * \frac{Number\ of\ images}{Accumulated\ time\ [ms]}$
Save/print all results.
```

Algorithm 2 Algorithm for benchmarking computing performance.

```
Get the input arguments (path to model, path to the labels and path to pictures to test on)
Initialize the engine.
$Accumulated\ time\ = 0$
for $Each\ image\ldots$ do
    Open the image.
    Preprocess the image (resize and crop).
    Flatten the image into an input tensor.
    Get current (start) time.
    Run inference over the image.
    Get current (end) time.
    $Time\ [ms] = (end\ time - start\ time) * 1000$
    $Accumulated\ time\ [ms] = Accumulated\ time + Time\ [ms]$
end for
$Average\ Computing\ Latency\ [ms/img] = \frac{Accumulated\ time\ [ms]}{Number\ of\ Images}$
$Computing\ Throughput\ [GFLOPs/sec] = GFLOPs * \frac{Number\ of\ images}{Accumulated\ time\ [ms]}$
Save/print all results.
```

each input (image) the USB Accelerator outputs the probability of each one of the 1000 categories. Only the five categories with the highest score are registered. Then, with another script, it is possible to compare the result obtained with the correct results and derive a top-1 and top-5 accuracy percentage.

The accuracy obtained, however, was lower than the one documented by Google Coral.

Algorithm 3 Algorithm for benchmarking Accuracy.

```
Get the input arguments (path to model, path to the labels and path to pictures to
test on)
Initialize the engine.
$Accumulated\ time\ = 0$
for $Each\ image\ldots$ do
    Open the image.
    Preprocess the image (resize and crop).
    Flatten the image into an input tensor.
    Run inference over the image.
    Get top-5 results as a list of tuples (Eg.: (354, 0.43210), ..., (458, 0.00001))
    Write them orderly to a file, the first category has the highest result. Eg.: 1st image:
222,333,444,555,666; 2nd image: 222, 333, 444, 555, 666; ...; 50000th image:222, 333,
444, 555, 666.
end for
```

For top-1 accuracy, it only matters the category with the highest probability for that image. For top-5 accuracy, what matters are the first five categories which hold the five highest probabilities. Then, another script is run to calculate the accuracy based on the results file and another file which has the correct answers. If the first answer is correct, then that counts as top-1, if any of the following four answers is correct, then that counts as top-5. In the end, the accuracies are calculated according to the following equations. $Top-1\ Accuracy = \frac{Top-1\ correct\ answers}{Total\ number\ of\ images}$. $Top-5\ Accuracy = \frac{Top-5\ correct\ answers}{Total\ number\ of\ images}$.

3.1.1.3 Power Measurements

Power measurements were taken with UM34C USB Power Meter illustrated in figure 3.2. The pseudo-code used is shown in algorithm 4. Measurements can be taken by plugging the UM34C into the computer and then plugging the USB Accelerator into the UM34C, which is illustrated in figure 3.3. Then, whatever current flows into the USB Accelerator it is detected by the power meter. What this little device measures are instant Voltage (V) and instant Amperage (A) at each second that passes. Being able to connect through Bluetooth to its app, which also helps to track these values, this device allows for recording measured data into a .xlsx file. Power [W] can be derived from instant voltage and current measurements.

Algorithm 4 Algorithm for benchmarking power consumption.

```
Get the input arguments (path to model, path to the labels and path to pictures to test on)
Initialize the engine.
for Each image... do
    Open the image.
    Flatten the image into an input tensor.
    Run inference over the image.
    Get the results.
end for
```

Figure 3.2: UM34C Power meter used in the power measurements.

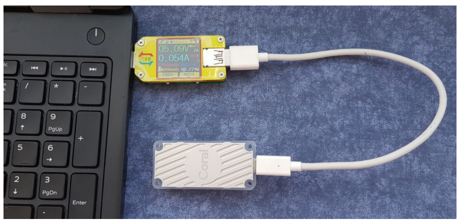


Figure 3.3: Connection between the computer, the Power Meter and the USB Accelerator.

3.1.2 Results

This section will begin with performance results. It is worth noting that these results represent the maximum value of performance achieved. Google results can be found here [21]. Table 3.1 shows latency results. Throughput results were also calculated, but latency results are displayed to compare with Google's results. EfficientNet S owns the smallest latency per image, while EfficientNet L owns the highest one. The higher the clock frequency of this device, the higher the performance it achieves, and vice-versa. Obtained results were slightly (between 0.2 to 0.6 ms) better for EfficientNet S and EfficientNet M than what Google reports out. Results obtained for EfficientNet L were somewhat worse than what Google reports with a difference of 0.6 ms. These differences can be explained by the way tests were done, in which case were different. Google's tests are performed using C++ benchmark tests. In contrast, these tests were conducted using Google's edge TPU Python API, specifically the base inference engine that executes TensorFlow Lite models on the EdgeTPU. These different platforms affect performance. Therefore mismatches are expected. Another thing to consider is that these tests were conducted in a Windows Operating System, whereas Google does not disclose which operating system was used while taking measurements.

Regarding accuracy results, table 3.2 depicts all accuracy numbers that were obtained

Table 3.1: Latency Results

	Latency				
	Full Frequency			Half Frequency	
	Computing		System [ms]	Computing [ms]	System [ms]
	Google [ms]	Obtained [ms]			
EffcientNet S	5.1	4.985	49.994	8.998	52.993
EfficientNet M	8.7	7.993	53.015	13.990	58.030
EfficientNet L	25.3	25.998	65.003	35.976	76.988

and the numbers reported by Google Coral. Obtained results are lower than what was reported by Google. It was not possible to replicate Google accuracy results.

Table 3.2: Accuracy results from benchmarking EfficientNets on USB Accelerator

	Top-1 Accuracy [%]		Top-5 Accuracy [%]	
	Google	Obtained	Google	Obtained
EfficientNet S	77.62	**49.27**	93.77	**71.81**
EfficientNet M	78.98	**61.21**	94.51	**82.71**
EfficientNet L	80.47	**57.30**	95.19	**78.71**

Regarding power results, the same can be found in table 3.3. For power measurements, what is listed in table 3.3 is the Peak Power consumption [W], which refers to the maximum value achieved. It is true that a spike of current might occur and might not be taken into account because measurements are done from one second to one second. However, as this was iterated through ten thousand images, the probability of this spike not being noticed is highly decreased.

The inference was executed over ten thousand times, one image at the time since the EdgeTPU only supports batch size equal to one. Statistical analysis with mean and standard deviation was derived from these experiments, and the same can be found in figure 3.4. This figure shows a bar plot where on the x-axis all topologies are listed, and on the y-axis, there is power in Watts. Red bars correspond to running the model in fast clock mode, which uses more energy, and the blue bars correspond to running the model in slow clock mode. The height of the bar corresponds to the average of the power consumption throughout the entire ten thousand experiments. The standard deviation displayed is the standard deviation of the whole ten thousand inputs processed serially. Idle mode power consumption is shown so there can be a comparison between what the device consumes while in idle mode and while doing inference. This plot is not displayed on the website.

There is the possibility that abnormal spikes might occur which introduces outliers in the data; for that reason, an algorithm was developed in python to eliminate all these outliers. The algorithm eliminates all values higher than a certain threshold. This threshold is calculated according to the following equation:

$Threshold = input\ coeficient * standard\ deviation$

Where:

Table 3.3: Power consumption results from benchmarking EfficientNets on USB Accelerator

	Peak Power [W]			
	Half Frequency		Maximum Frequency	
	Idle mode	Board Load	Idle mode	Board Load
EfficientNet S	0.83	1.49	0.81	1.99
EfficientNet M	0.83	1.68	0.81	2.01
EfficientNet L	0.83	1.63	0.80	2.88

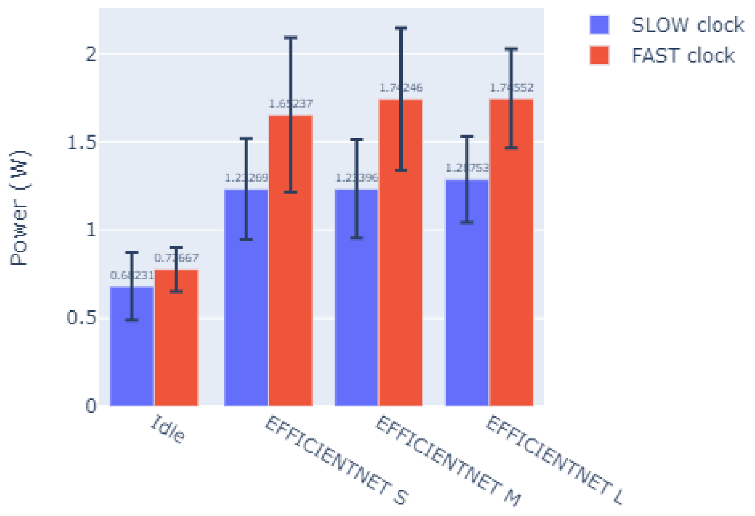


Figure 3.4: Average and standard deviation for power measurements on the USB Accelerator. There were ten thousand measurement repetitions over batch size of one image.

$Input\ coefficient =]-\infty, +\infty[$ and standard deviation is the standard deviation of the entire data. This code can be found on the online repository where all code is available.

To show the evolution of power over time, figure 3.5 was created. This figure shows several line plots with the evolution of the power consumption split by topology and clock mode. On the x-axis, there is time evolution on seconds, and on the y-axis, there is the power consumption in Watts. This plot shows the first two hundred seconds of the experiments. However, some experiments' duration exceeded this time, but they are not shown because the intention was to zoom in the first two hundred seconds. This plot can be found on the website, and it is interactive, so besides zooming in, zooming out and dragging the plot around, it is possible to see the entire duration of the experiments.

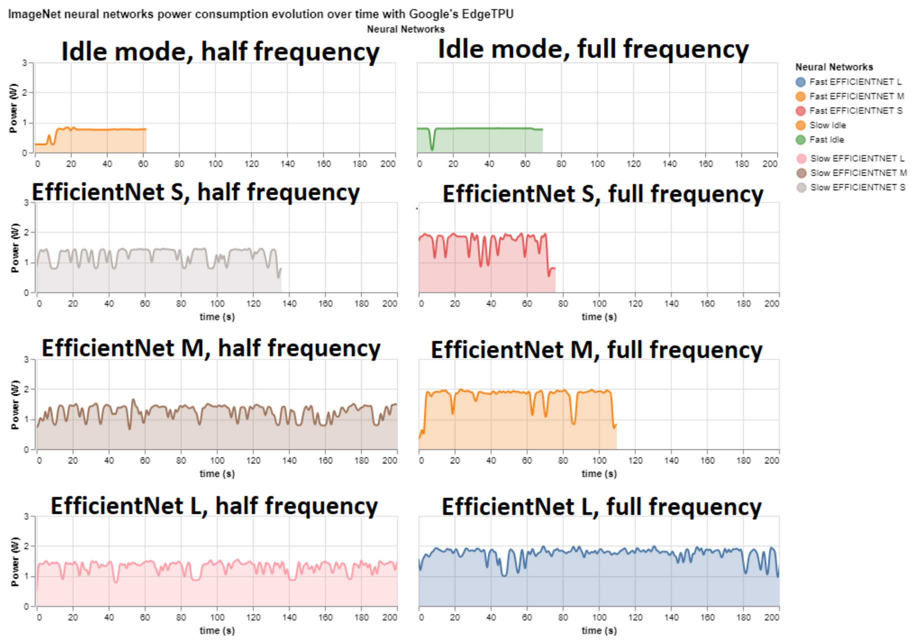


Figure 3.5: Line plot with power consumption evolution over time for ImageNet task on Google's USB Accelerator. Figure available at [5].

CHAPTER 4

Development: Website Visualizations

This chapter will focus on describing the website development and all visualizations that were created.

To provide full access to data, visualizations and to allow for contributions a web portal was deployed, which can be found at [5]. The GitHub repository with all code open-sourced can be found at [40].

The website was developed with fast pages which is an "easy to use blogging platform, with support for Jupyter notebooks, Word Documents and Markdown" [18]. This technology was chosen because is one of the fastest and easiest way to render a blogging website directly from jupyter notebooks. With GitHub, an entire blog can be created, so a GitHub account is needed. The setup process of fast pages includes creating a repo from the fast pages template through a pull request, then a couple of steps need to be followed. Finally, the webpage with a standard template is available. From then on, GitHub actions process notebooks and each notebook is presented as a page inside the website. Figure 4.1 shows a representation of the website rendering.

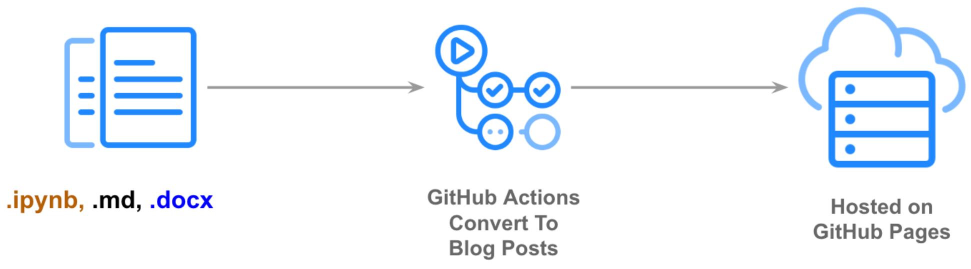


Figure 4.1: Fastpages diagram [18].

The website created is divided into several pages. The initial page shows an introduction to QuTiBench, the benchmarking methodology, then all pages available are presented.

There is an “About this Work” section where all contributors to the QuTiBench project can be found.

4.1 Theoretical Analysis - Level 0 of QuTiBench

Since level 0 of QuTiBench is the theoretical analysis, it makes sense to start describing the website from this point on. A theoretical analysis is presented by applying level 0 of the benchmarking methodology. Both compute, and memory requirements are analyzed for both hardware platforms and applications. Performance predictions are made through Heatmaps with the help of Roofline models. All these predictions are made with no measurements required.

4.1.1 Methodology

Theoretical analysis is a simple tool from which preliminary conclusions can be drawn with no actual data measured. It should provide instant feedback and performance predictions quickly. The theoretical analysis was applied to both hardware platforms (Peak Performance in TOPS/sec, external memory bandwidth in GBps and Thermal Design Power in Watts) as well as algorithms in the form of compute and memory requirements for all CNN topologies. Combining hardware platform characteristics with algorithm requirements makes it possible to create performance predictions with the help of roofline models.

4.1.1.1 CNNs and their compute and memory requirements

As introduced in chapter 2, three machine learning tasks were selected: MNIST, ImageNet and CIFAR-10. Within these machine learning tasks, there are different topologies with several variants. Some models can be found online; other models were trained in-house.

Performance predictions can be formed by combining hardware platform characteristics with algorithm requirements, which can be achieved through roofline models. Determining algorithm requirements can be done by calculating the operational intensity for each topology and each datatype.

Operational intensity is a concept introduced by the Rooflines. It is generally seen as the ratio between Work (W) and Memory Traffic (M) as in equation 4.1. In this particular context, work refers to the number of floating-point instructions performed on the data and Memory Traffic refers to the amount of data bytes moved to and from the off-chip memory. Note that for general-purpose processors, the Memory Traffic is considered to be between the DRAM and the cache hierarchy and not between the cache hierarchy and the processor because the former is often much slower than the latter. Moreover, the intent is to look for the worst-case scenario. One fundamental assumption is that all parameters are staying off-chip and all intermediate values, i.e., activations, are on-chip memory.

The operational intensity for each datatype was calculated according to equation 4.1, which is equivalent to equations 4.2,4.3,4.4,4.5, meaning that they all mean the same, just written in a different way.

$$Operational\ Intensity = \frac{W}{M} \tag{4.1}$$

$$Operational\ Intensity = \frac{Compute\ Requirements}{Memory\ Requirements} \tag{4.2}$$

$$Operational\ Intensity = \frac{Number\ of\ Floating\ point\ operations}{Off-chip\ bytes\ transferred} \tag{4.3}$$

$$Operational\ Intensity = \frac{Number\ of\ Floating\ point\ operations}{Number\ of\ parameters * bytes} \tag{4.4}$$

$$Operational\ Intensity = \frac{Total\ Operations}{Model\ size * \frac{Datatype\ bits}{8}} \tag{4.5}$$

Where:
W = Work
M = Memory traffic

The total amount of operations and the model size are inherent characteristics of each topology and can be found on table 4.2. Table 4.1 shows a complete overview of the topologies leveraged in the experimentation and also the acquired accuracies. Some models were trained by us; others were not.

Table 4.1: Experimental CNNs and their accuracies. Table adapted from [8]

	INT2	**INT4**	**INT8**	**FP16**	**FP32**
	top1 (top5) [%]	**top1 (top5) [%]**	**top1 (top5) [%]**	**top1 (top5) [%]**	**top1 (top5) [%]**
GoogLeNetv1 100	nm	nm	69.24 (88.45)	66.93 (87.83)	66.96 (87.84)
MobileNetv1 100	nm	nm	69.57 (87.71)	nm	nm
EfficientNet-S 100	nm	nm	77	nm	nm
EfficientNet-M 100	nm	nm	78.6	nm	nm
EfficientNet-L 100	nm	nm	80.2	nm	nm
ResNet-50 100	nm	nm	73.29 (91.26)	75.14 (92.12)	75.15 (92.11)
ResNet-50 80	nm	nm	73.30 (91.40)	nm	nm
ResNet-50 50	nm	nm	69.49 (91.00)	nm	nm
ResNet-50 30	nm	nm	68.83 (90.16)	nm	nm
CNV 100	86.86	87.4	nm	87.02	87.06
CNV 50	84.29	84.88	nm	85.55	85.6
CNV 25	79.89	81.09	nm	83.28	83.25
CNV 12.5	73.64	75.85	nm	77.82	77.84
MLP 100	98.75	98.77	nm	97.3	97.31
MLP 50	98.49	98.62	nm	97.45	97.46
MLP 25	98.04	98.29	nm	97.49	97.44
MLP 12.5	96.85	97.54	nm	97.95	97.15

Looking at these equations, it is possible to understand why it is essential that the lower the precision, the higher is the operational intensity. Moreover, vice versa is also visible, the higher the denominator, the lower the operational intensity becomes. Thereby, mathematically quantization makes a difference in the operational intensity.

4.1.1.2 Hardware Platforms Performance

In this study, several types of hardware were studied. FPGAs are one of the most flexible among other hardware platforms and were configured to support all numerical representations.

For FPGA platforms, it was assumed a 100% resource utilization and a frequency of 666MHz. There are three different types of implementations. The first implementation is the Deep Learning Processor Unit (DPU) which consists of a matrix of processing engines instantiated on the device. The second is FINN which mimics the CNN topology in the hardware by creating a customized dataflow compute architecture and was previously presented in subsection 2.4.1.2. The third one is BISMO which is a generic matrix multiply accelerator and was previously introduced in subsection 2.4.1.1. More information on these hardware platforms can be found here [6].

4.1.1.3 Methodology

Rooflines model the theoretical peak performance of given hardware, taking into account two things: the off-chip memory bandwidth (mem_bw) and the peak compute performance (p_hw_pp) of the hardware. There are different peak performances for each numerical representation. For example, a GPU has different peak performance for FP32, FP16, INT8, INT4 and INT2. Hardware performance (hw_pp) can be thought of as a function of the operational intensity of an application, which represents the ratio between work and memory traffic, more specifically, between compute requirements *cmp_req* and memory requirements *mem_req*: $Operational\ Intensity = \frac{cmp\_req}{mem\_req}$. Hardware performance *hw_pp* was calculated through formula 4.6.

$$hw\_pp = Min(p\_hw\_pp\ , mem\_bw * \frac{cmp\_req}{mem\_req}) \tag{4.6}$$

Where *hw_pp* is calculated as the minimum between peak hardware performance and memory bandwidth *mem_bw* multiplied by an application's operational intensity.

A roofline graph, presented in figure 4.2 is a 2D line graph where the x-axis shows the operational intensity of applications and the y-axis represents hardware performance (hw_pp). In this plot, the slope part of the chart represents the memory-bound area, and the flat part of the graph represents the compute-bound. By combining theoretical peak performance, memory bandwidth and the operational intensity of all CNNs, it is possible to understand if a model will be compute or memory bound.

As presented in chapter 2, Level 0 is achieved with a Theoretical Analysis that provides instant results. Theoretical Peak Performance (TOP/sec) and Memory Bandwidth (GBps) are performance upper bounds and are architecture-specific, do not depend on the given application. However, what limits the application's performance (either peak performance or memory bandwidth) much depends on the application itself, and so, on the type of computation that is performed.

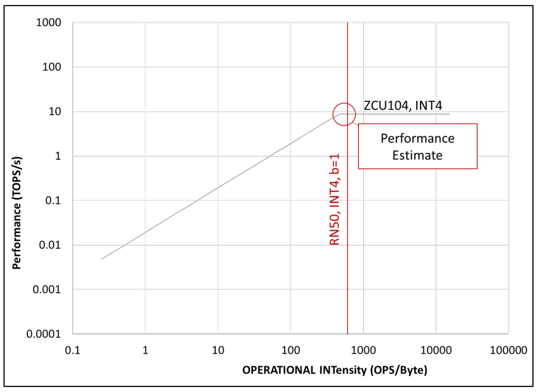


Figure 4.2: Performance estimate on a Roofline model [8].

Although Roofline models were first introduced for general-purpose processors, the same concept is being applied to FPGAs. However, estimating a Roofline model for FPGAs is somewhat tricky because FPGAs do not define a fixed architecture. As there are a lot of possible FPGA architectures that can be implemented for the same application, it becomes difficult to estimate whether the application will be compute-bound or memory-bound. This is opposed to a general-purpose processor in which the architecture is well defined and known. Even so, the Roofline model can still be used to understand which is the maximum level of performance that can be achieved on a target FPGA. For this, the final implementation has to be fully optimized.

Having each CNN's operational intensity obtained by equation 4.1 and replacing this in equation 4.6 performance estimates can be made. The performance estimate is indicated in figure 4.2 with the red circle at the intersection between the roofline and the line representing the application. To compute the operational intensity of each CNN, some assumptions had to be made, for instance where weight, weight updates, tensor, gradients and state of a neural network are stored and combine all with the size of the datatypes.

To facilitate comparison between all CNNs of the same ML task performance predictions pp were created, measured in inputs/sec, through equation 4.7. pp is the performance prediction, hw_pp represents the theoretical peak performance calculated with the roofline model, cmp_req is amount of compute necessary to process one input, mem_bw is available memory bandwidth, mem_req are memory requirements in millions of elements (ME) and p_hw_pp is hardware's absolute peak performance.

$$pp = \frac{hw\_pp}{cmp\_req} = Min(\frac{p\_hw\_pp}{cmp\_req}, \frac{mem\_bw}{mem\_req}) \tag{4.7}$$

Results are computed with this equation and are visualized with heatmaps as will be shown in section 4.1.2.4.

4.1.2 Results

Three ML tasks were benchmarked, namely MNIST, ImageNet and CIFAR-10 with different datasets and different topologies per ML task. Regarding hardware platforms, several categories were benchmarked, particularly GPUs, CPUs, VLIWs, FPGAs and Edge TPU. There are pruned and quantized versions of several topologies on all three machine learning tasks to cover a broader scope of optimization techniques across several types of ML tasks and topologies. A sweep over batch size, thread count and stream size was made over all operating modes supported by hardware. What was measured was throughput, power, accuracy and latency. More details and all results can be found on the web portal [5].

4.1.2.1 CNN Topologies

All results for level-0 of QuTiBench are shown in table 4.2 which includes, for each topology, the total number of operations per one input in Giga Operations (GOPs), the model size in Millions of Elements (ME), and finally, the Operational Intensity in operations per byte read or written from memory. The operational intensity for each datatype was calculated according to equations 4.1,4.2,4.3,4.4 and 4.5.

Table 4.2: CNNs and their compute and memory requirements. Table Adapted from [8].

	Total OPs	**Total Model Size**	**OI (INT2)**	**OI (INT4)**	**OI (INT8)**	**OI (FP16)**	**OI (FP32)**
	GOPs	**[ME]**	**[Ops/Byte]**	**[Ops/Byte]**	**[Ops/Byte]**	**[Ops/Byte]**	**[Ops/Byte]**
GoogLeNetv1 100%	3.1	6	2093.97	1046.99	523.49	261.75	130.87
MobileNetv1 100%	1.1	4.2	1075.47	537.74	268.87	134.43	67.22
ResNet-50 100%	7.7	25.5	1210.84	605.42	302.71	151.36	75.68
ResNet-50 80%	6.5	23.7	1086.59	543.3	271.65	135.82	67.91
ResNet-50 50%	3.8	15.8	949.85	474.93	237.46	118.73	59.37
ResNet-50 30%	2.5	10.1	970.16	485.08	242.54	121.27	60.64
EfficientNet-S 100%	4.7	5.4	3481.48	1740.74	870.37	435.18	217.59
EfficientNet-M 100%	7.4	6.9	4289.86	2144.93	1072.46	536.23	268.12
EfficientNet-L 100%	19.4	10.6	7313.21	3656.6	1828.3	914.15	457.08
CNV 100%	0.47	6.16	304.95	152.48	76.24	38.12	19.06
CNV 50%	0.12	1.54	308.32	154.16	77.08	38.54	19.27
CNV 25%	0.03	0.39	315.01	157.51	78.75	39.38	19.69
CNV 12.5%	0.01	0.1	332.61	166.3	83.15	41.58	20.79
MLP 100%	0.02	10.01	8	4	2	1	0.5
MLP 50%	0.00582	2.91	8	4	2	1	0.5
MLP 25%	0.0019	0.93	8	4	2	1	0.5
MLP 12.5%	0.0007	0.33	8	4	2	1	0.5

Regarding Total Operations executed per DRAM traffic EfficientNet Large is the highest followed by ResNet-50 while MLPs are by far the lowest ones. ResNets own the most massive model size while CNVs own the smallest model size.

The operational intensity is calculated for each data type, and the assumptions are that weights reside off-chip, intermediate results reside on-chip, and the batch size is equal to one. Regarding the operational intensity (Ops/Byte), EfficientNets have the highest operational intensity because they were specifically created based on a new scaling method to achieve high accuracies but also high performance. MLPs, however, own the lowest operational intensity essentially because of its three fully connected hidden layers, which turn out to be very memory intensive.

Some conclusions can be drawn early on. Looking at the operational intensity of each topology, it is predicted that EfficientNets will probably be compute-bound, due to the extremely high OI. MLPs will probably be memory bound due to the extremely low OI. Pruning does not affect operational intensity. However, as the compute per frame decreases, the frame rate of the entire algorithm should increase proportionally to the pruning factor. In general, quantization doubles the OI. This is because quantization decreases the number of elements that need to be read and written to memory, thus increasing the OI.

To better understand table 4.2, figure 4.3 was created and shows the two bar charts that were created based on it. Both charts show different figures of merit in the function of the corresponding CNN. The first chart shows the number of operations (GOPs) per CNN, and the second one shows the total model size (ME). Please note the logarithmic scale. Looking at these charts, it becomes obvious that EfficientNet Large owns the highest number of operations, whereas ResNets own the highest number of elements.

Please note that all these charts are interactive, and so it is possible to zoom in or move them around, everybody is encouraged to visit the website to give it a try.

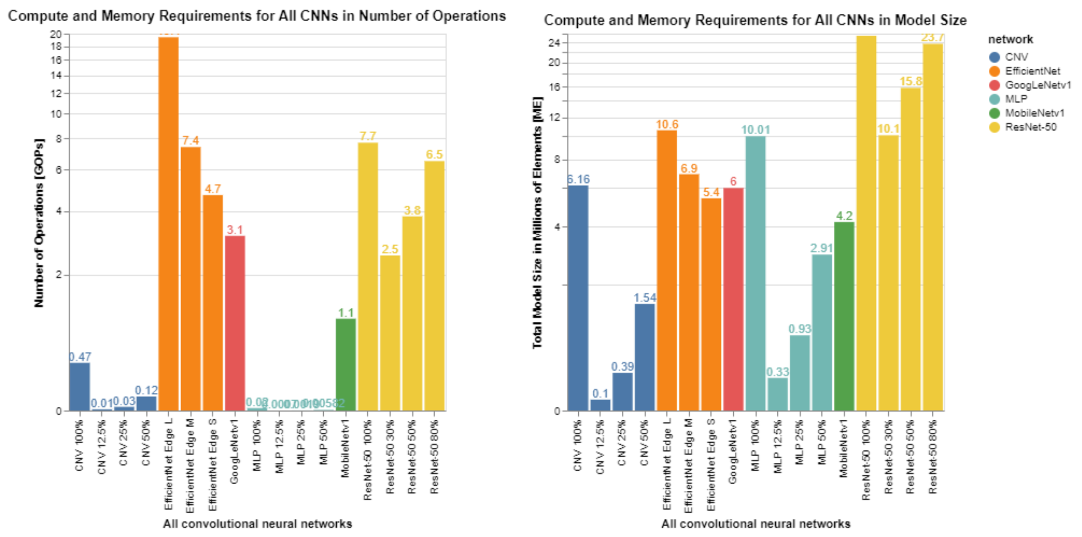


Figure 4.3: CNNs compute and memory requirements. Figure available at [5].

4.1.2.2 Hardware Architectures

Table 4.3 summarizes level 0 of QuTiBench for hardware platforms. In this table, theoretical peak performance (TOP/sec) for the designated data types, memory bandwidth in Giga Bytes per second (GBps), memory capacity (GB) and power consumption (Watt) can be visualized.

Regarding this table, some conclusions can be made. GPUS are the highest in terms of Memory Bandwidth, but also power consumption, in this field, USB devices such as Google's EdgeTPU and Intel's NCS are the lowest. Regarding performance, the highest

Table 4.3: Level 0 of QuTiBench. Hardware platforms with their Peak Performance Predictions [8].

Hardware Platforms	INT2	INT4	INT8	FP16	FP32	Memory Bandwidth	Memory Capacity	Power
	[TOP/sec]	[TOP/sec]	[TOP/sec]	[TOP/sec]	[TOP/sec]	[GBps]	[GB]	[Watt]
Ultra96-DPU	na	na	0.96	na	na	4.26	2	na
ZCU104-DPU	na	na	4.6	na	na	19.2	4	na
ZCU102-DPU	na	na	6.71	na	na	19.2	4	na
ZCU104-FINN	30.7	8.8	na	na	na	19.2	4	na
ZCU104-BISMO	30.7	8.8	na	na	na	19.2	4	na
TX2 max-n	na	na	na	1.33	0.67	59.7	8	15
TX2 max-p	na	na	na	1.15	0.57	59.7	8	15
TX2 max-q	na	na	na	0.87	0.44	59.7	8	15
EdgeTPU-fast	na	na	4	na	na	25.6	1	2
EdgeTPU-slow	na	na	2	na	na	25.6	1	2
NCS (MyriadX)	na	na	1	0.5	na	12.8	2	1
U96-Quadcore A53-INT8	0.192	0.192	0.192	na	na	4.26	2	na

one is achieved with the lowest precision (INT2) by the FPGAs, which are the only hardware platforms that support reduced precision. In general, reduced precision increases performance. In contrast, increased precision reduces performance, and this is quite visible for the TX2, where 32-bit Floating-Point Representation (FP32) performance is half the 16-bit Floating-Point Representation (FP16) performance.

Some bar charts were also created to visually illustrate this table shown in figure I.1. Please note the logarithmic scale.

4.1.2.3 Roofline model

Having each CNN's operational intensity, it is possible to plot these neural networks along with hardware platforms to get insights to whether models will be compute or memory bound. With this in mind, figure 4.4 was created.

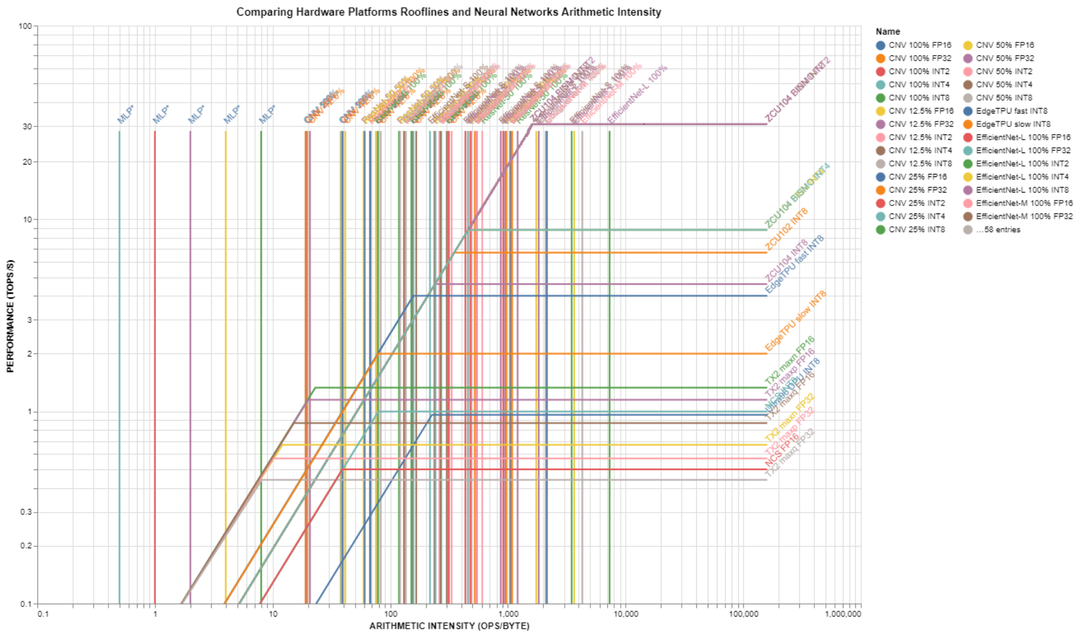


Figure 4.4: Roofline Model. Figure available at [5].

By combining hardware platform characteristics with application requirements, this plot presents insights into performance predictions.

With Rooflines, performance predictions can be made, where the maximum performance for a CNN in that specific hardware is given by the value that lies at the intersection point between the operational intensity and the roofline of the hardware platform as shown in figure 4.2. ResNet50 is one of the highest in compute and memory requirements, followed by GoogLeNet and then MobileNetv1. Figure 4.5 shows the operational intensity for all neural networks datatypes leveraged in this experimentation.

CNN's operational intensity for FP32 ranges from 0.5 to 458. In contrast, CNN's operational intensity for INT2 ranges from 8 to 7313, with an increased incidence near the 7313 ops/byte. The range is also shown for FP16, INT8 and INT4. For FP32, most neural networks are memory bound for most hardware platforms. With reduced precision, however, this changes, pushing the intersection points to the right of the plot, turning these neural networks into compute-bound. Even so, even with INT2, there are still some models that are memory bound. A possible solution for this could be to increase batch size.

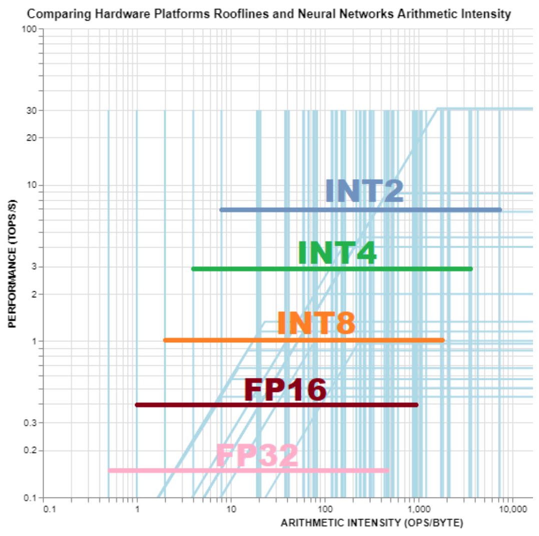


Figure 4.5: Performance prediction with roofline analysis. Figure available at [5].

With this model (fig 4.4 and 4.5), insights on how to scale up performance can be made. There are several options available for improving the application's performance. If the application is memory bound in an FPGA, there are several options to improve performance. The first one would be to increase the bit width of data transfers. Each bit of data must be transferred on a different wire of the data bus because sending more than one electrical signal on the same wire causes a short circuit. Then, if the data bus consists of 8 bits it can transfer 1 byte of data per reading or write operation; if it consists of 16 bits, it can move 2 bytes of data per reading or write operation; and so on. Therefore, the width of the data bus determines the amount of data transferred per memory transfer operation. Another option would be to increase the number of memory ports used when transferring data to and from off-chip memory [33]. Now, another option would be to increase the operational intensity of the application to make it compute-bound instead of memory bound. This can be done leveraging local memories available on the FPGA to optimize the data transfer required. As a result, increasing the number of operations performed concerning the amount of data moved to and from the off-chip memory could be a solution. Another technique used to improve operational intensity is to compress the data input and output so that the overall memory traffic is reduced. In essence, the aim is to reduce the number of bits needed to encode data, also known as quantization.

What is interesting to note is that reducing precision pulls the application to the right side of the plot, making it more compute-bound. This makes sense because reducing precision inherently increases the number of operations because of the amount of external memory required for storage weights decreases. This is the same as saying that the processor no longer needs to be fetching so much data from external memory. Since external memory is extremely slow, it can now spend that time doing operational operations. However, even for INT2, there are still memory-bound topologies, namely MLP. What needs to be taken into account is the initial assumption that weights reside off-chip and batch size is equal to one.

4.1.2.4 Performance Predictions

Heatmaps (figures 4.6, I.2 and I.3,) show the input/sec for all leveraged neural networks across all hardware platforms. In order to allow for comparisons between all different models, performance predictions expressed in input/sec are used as a metric for theoretical performance, which is calculated with equation 4.7.

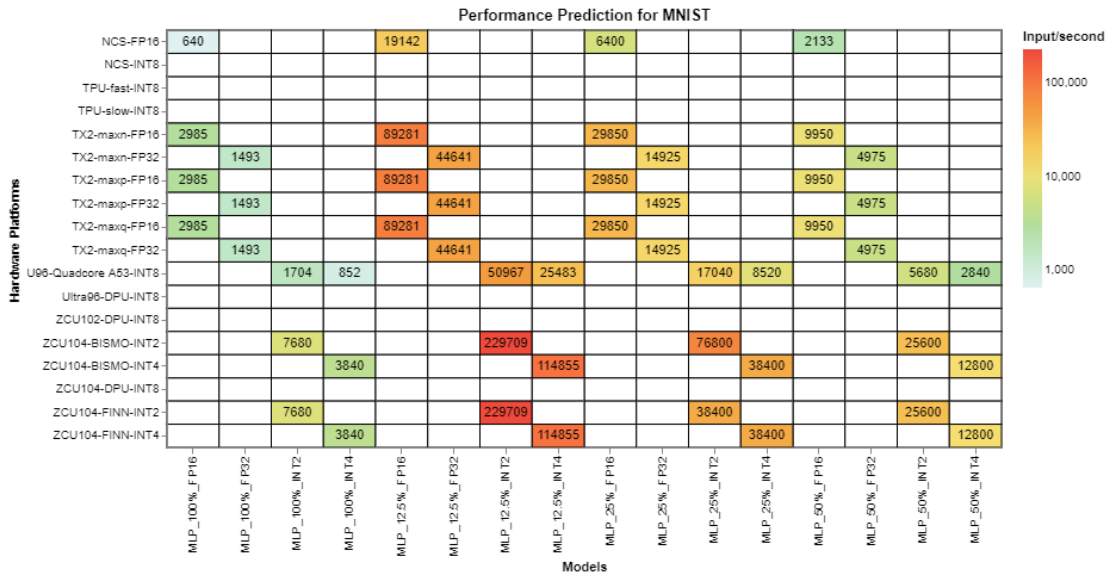


Figure 4.6: Performance Predictions with Heatmaps for MNIST classification

The colour of each cell represents the intensity of the performance predictions according to the legend. These results were plotted using heatmaps, and there is one heatmap per ML task. All these plots are interactive, meaning that a tooltip is displayed when hovering with the mouse showing the designated input/second, the model name and hardware platform.

For MNIST, the prediction made is that the highest performance will be achieved by MLP pruned at 12.5% with INT2 representation on the ZCU104.

For ImageNet ML task it is predicted that the highest performance will be for MobileNetv1 on the TPU followed by GoogLeNetv1 with INT8 precision on the ZCU102-DPU, followed by ResNet50 pruned at 30% with INT8 representation on the ZCU102-DPU.

For CIFAR-10, the highest predictions are for CNV pruned at 12.5% with INT2 representation achieving 638592 inputs/sec on both ZCU104-FINN and ZCU104-BISMO.

It is predicted that the highest performance is achievable with the lowest precisions on the FPGAs. What is clear from these results is that in general pruning combined with quantization provides the highest performance. One thing to note is that FPGAs are the only hardware platform where reduced precision is supported.

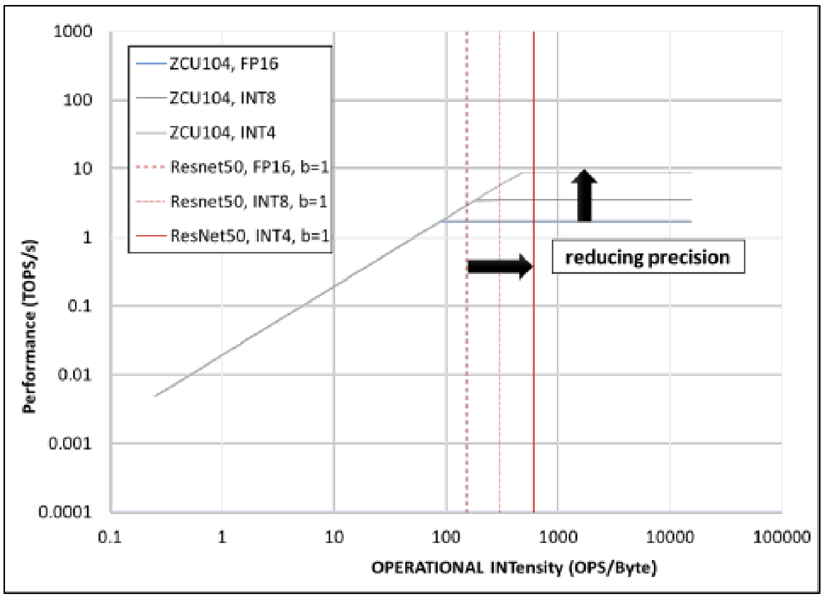


Figure 4.7: Effect of quantization on OI and performance [8].

Quantization works by pulling the ceiling upwards, because reduced precision, in general, increases performance; this is illustrated in figure 4.7. Besides that, quantization also improves the operational intensity of the application making it more likely to be compute-bound. Thus, the benefits of quantization are twofold.

Some conclusions can be derived from these early results. Pruning does not affect operational intensity, but as the compute per input is reduced the overall input/sec should increase in proportion to the amount of pruning. Quantization, on the other side, does affect the operational intensity by lowering the amount of traffic to and from memory.

Besides this, quantization also increases the theoretical performance of a given hardware platform. Thus a double benefit can be achieved.

4.1.2.5 Theoretical Pareto

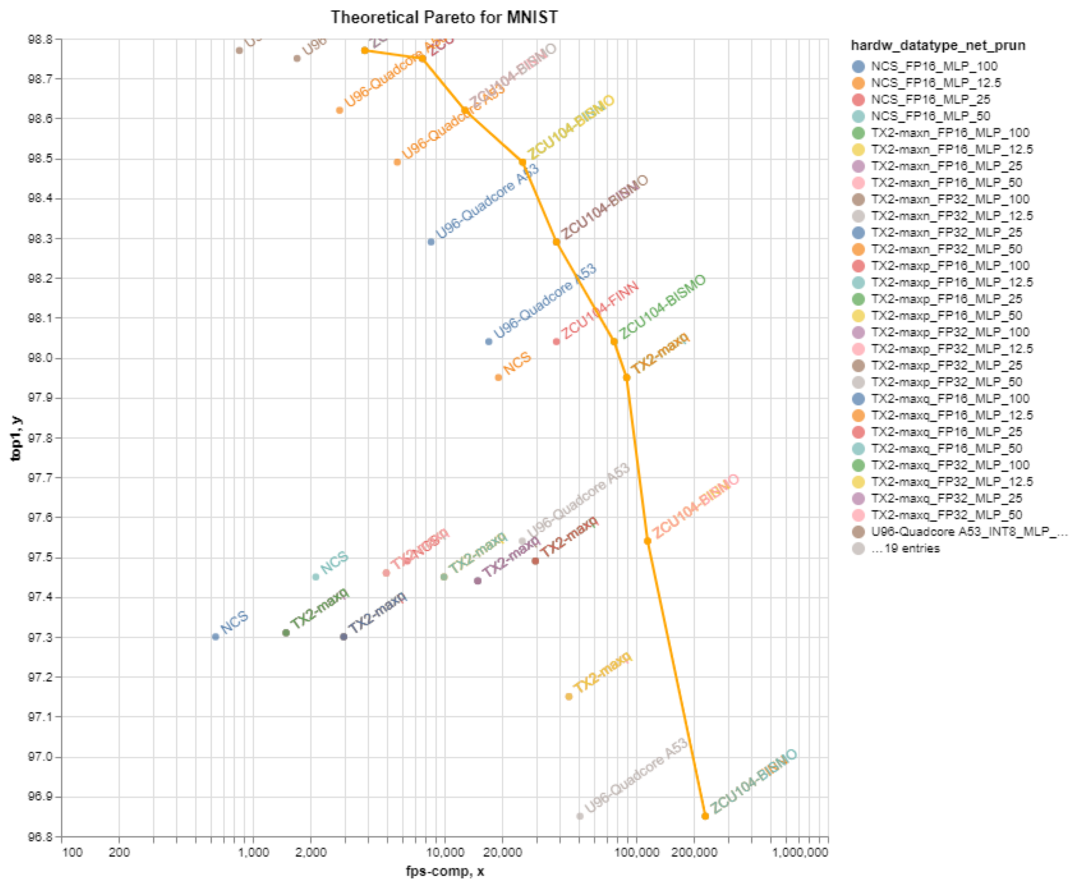


Figure 4.8: Theoretical Pareto for MNIST Classifications. Figure available at [5].

Combining performance results from heatmaps with the accuracies of the topologies, a theoretical Pareto plot can be created, which is shown in figures 4.8, I.4 and I.5. In these figures, a theoretical Pareto curve for each machine learning task is shown. These figures present theoretical data which was predicted with no actual data collected. On the y-axis, there is top-1 accuracy and on the x-axis throughput, expressed as frames per second. The best solution would be situated on the top right corner because that is where the accuracy and the throughput hold the highest value. However, such a solution does not exist. The higher the throughput, the lower accuracy. There is always a compromise between accuracy and throughput. A way to make use of this kind of plot is described as the following. Suppose the application is accuracy bound. When a horizontal line is drawn, below which the accuracy cannot be lower, then the data point with the highest throughput above that horizontal line is the optimal solution for the application.

Imagine a particular application that is bound by throughput; it could be a real-time embedded application. If a vertical line is drawn and stipulated that the throughput cannot be lower then that exact value, then the data point with the highest accuracy at the right of that vertical line is the optimal solution for the application.

4.2 MNIST Classification - Level 3 of QuTiBench

After theoretical analysis presenting level 0 of QuTiBench is described, this effort will carry on describing level 3 of QuTiBench. Only level 0 and level 3 results are presented on the web portal. This section will start with describing level 3 for MNIST classification, and since all three pages (one per ML task) have the same structure, CIFAR-10 and ImageNet will not be described, in order not to repeat the same kind of content.

The MNIST page starts by presenting a theoretical analysis which was introduced in the theoretical analysis page. Here data points were filtered out to the MNIST machine learning task, so all data points show MLP and its variants across the full spectrum of hardware platforms and configurations. Then there are performance predictions with heatmaps which were already presented in the theoretical analysis. Next, there is the experimental data analysis were all data points were no longer predicted but indeed measured.

4.2.0.1 Box Plots

Figure 4.9 presents an overview of power consumption across all the design space. A pruning factor of one hundred means that the model was not pruned. The more pruned the network is the less power it consumes, this is a trend across all Hardware platforms.

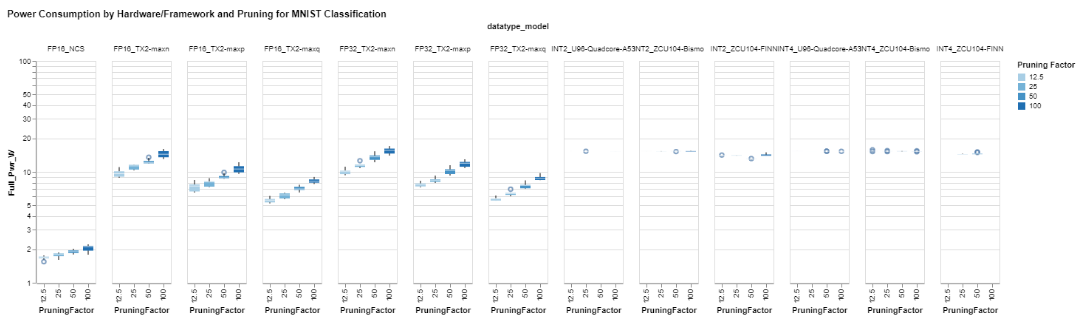


Figure 4.9: Power consumption. Figure available at [5].

4.2.0.2 Pareto Graphs

In figure 4.10 there is a Pareto graph which presents the accuracy vs performance in frames per second for all Hardware platforms across several pruning and quantization configurations. It can provide insights into accuracy based configurations and also performance-based configurations.

The main question that this plot tries to answer is for a certain accuracy what brings the best performance, is it pruning or reduced precision? The way that this plot was created was the following. It is a standard line plot, and the Pareto frontier was created connecting all the data points that appeared to be on the frontier. It is not easy to identify all the data points in figure 4.10, and so the reader is invited to visit the website to visualize better the plot and all its data. This plot is interactive as it can zoom in, zoom out and move the chart around. There are also tooltips when hovering with the mouse on top of each data point.

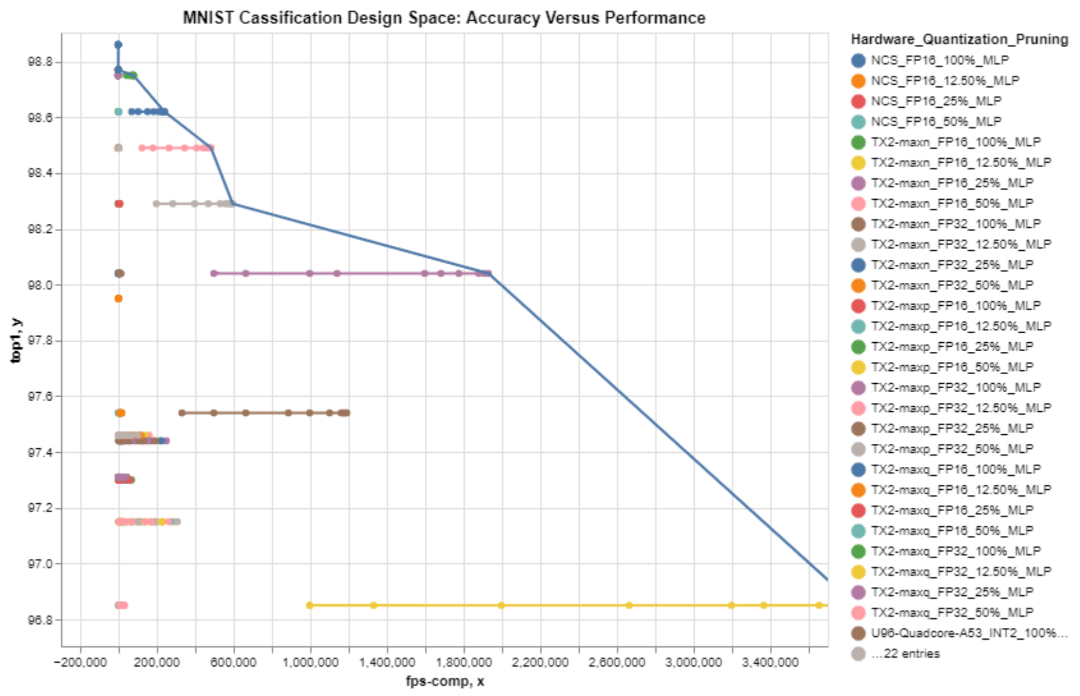


Figure 4.10: Accuracy versus performance. Figure available at [5].

4.2.0.3 Power Plots

Based on the power consumption of each model on each platform, it was created a rather simple bar chart on figure 4.11. This bar chart answers the following questions. In each platform, which model consumes more power? Which model consumes less power? Furthermore, does pruning have any impact on power consumption?

In this bar chart, the height of the bar represents the peak power consumption that that model has consumed on that specific platform. In the MNIST plot, pruning plays an important factor in power consumption, as there is a drop in power consumption for the pruned variants of MLP. This is visible for almost all hardware platforms.

4.2.0.4 Performance Normalized by Power Plots

After having the power plots mentioned above, it is important to be able to compare performance while having an idea of the power consumption. This kind of plot is illustrated

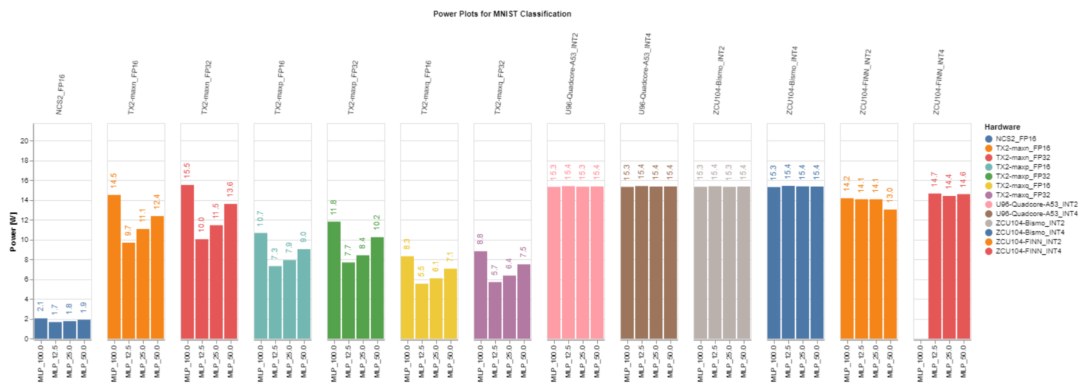


Figure 4.11: Power bar charts for MNIST. Figure available at [5].

in figure 4.12. The height of the bar represents the performance [fps] normalized by power.

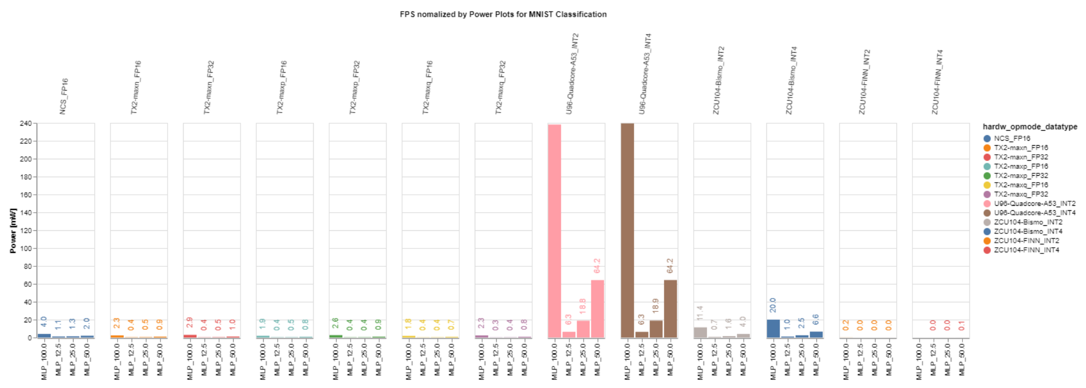


Figure 4.12: Performance normalized Bar charts for MNIST. Figure available at [5].

All these plots are created for ImageNet and CIFAR-10 classification. In order not to repeat the same thing, it will not be explained all the details of the other two pages because they are quite very similar to this one in structural terms, to which the reader is invited to visit.

As a last note, the line plots inside the website are not part of this work so they will not be described here.

4.3 Raw Measurements

This page presents all the raw data that there is. The data is separated by machine learning task. Inside each machine learning task, data is separated by the hardware platform. It is possible to download all data related to a specific machine learning task into a CSV file.

4.4 FAIR and Open Data

FAIR and Open data are two very similar concepts; however, they are not the same. FAIR is the abbreviation of Findable, Accessible, Interoperable and Reusable. FAIR data means that data generated complies with some or all of the principles for scientific data management and stewardship. This project has adhered to the FAIR principles, by:

- Making all data open and freely available. All data can be viewed and downloaded through an open-source repository and a web portal;
- Created a Unique and Persistent Identifier (DOI) with ResearchGate associated with the data: DOI: 10.13140/RG.2.2.35785.57448;
- Created data-level and project-level documentation also metadata associated with the data through the Dublin Core Generator which can be found in the repository.

4.5 How to make contributions page

There is a page showing how everybody can contribute to the QuTiBench project. For this, there are a couple of steps that anybody who wishes to contribute can easily follow.

4.6 Overview of the Experiments page

On this page, an overview of all experiments is presented. It consists of three tables, one per machine learning task. It is possible to visualize which model ran into which hardware platform with what precision and pruning factor. The batch size is also indicated.

4.7 Code and online repository

Full code will not be accompanying this document, as it is available in an online GitHub repository [40]. There are 5 jupyter notebook files, namely the following: “CIFAR-10_Classification.ipynb”; “ImageNet_Classification.ipynb”; “MNIST_Classification.ipynb”; “Contributing_Measurements.ipynb” and “Overview_of_experiments.ipynb”.

Website structure can be found in figure 4.13, where each square denotes a web page and inside each square/rectangle there are all that page’s contents. MNIST, ImageNet and CIFAR-10 web pages have the same structure, so only one is shown to avoid repetition. In total, there are eight web pages besides the “About us” page.

An overview of the GitHub project is shown in figure 4.14. Each web page is codified in each Jupyter notebook, as shown in figure 4.14.

What is needed to create a Roofline is to load into memory a python file, where all methods are defined and then call the “rooflines” method specifying the desired machine learning task. So the code to create a roofline stands in a script and then whenever that

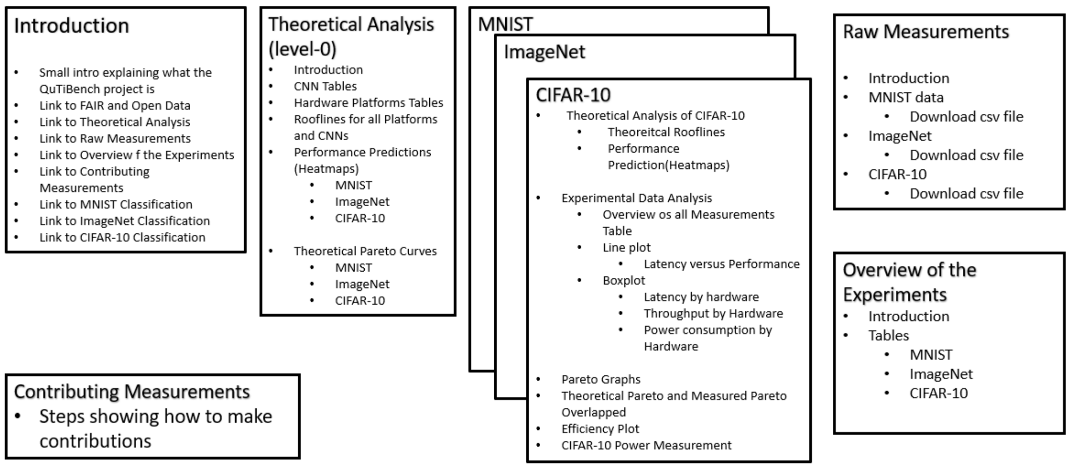
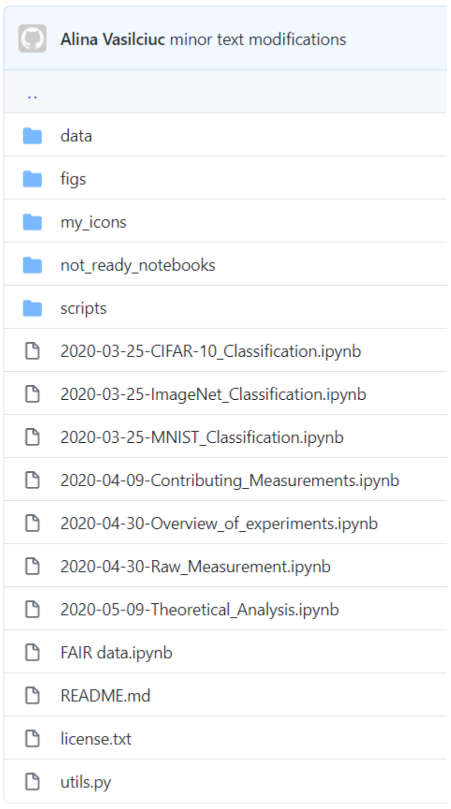

Introduction
Small intro explaining what the QuTiBench project is
Link to FAIR and Open Data
Link to Theoretical Analysis
Link to Raw Measurements
Link to Overview f the Experiments
Link to Contributing Measurements
Link to MNIST Classification
Link to ImageNet Classification
Link to CIFAR-10 Classification
Theoretical Analysis (level-0)
Introduction
CNN Tables
Hardware Platforms Tables
Rooflines for all Platforms and CNNs
Performance Predictions (Heatmaps)
MNIST
ImageNet
CIFAR-10
Theoretical Pareto Curves
MNIST
ImageNet
CIFAR-10
MNIST
ImageNet
CIFAR-10
Theoretical Analysis of CIFAR-10
Theoreitcal Rooflines
Performance Prediction(Heatmaps)
Experimental Data Analysis
Overview os all Measurements Table
Line plot
Latency versus Performance
Boxplot
Latency by hardware
Throughput by Hardware
Power consumption by Hardware
Pareto Graphs
Theoretical Pareto and Measured Pareto Overlapped
Efficiency Plot
CIFAR-10 Power Measurement
Raw Measurements
Introduction
MNIST data
Download csv file
ImageNet
Download csv file
CIFAR-10
Download csv file
Overview of the Experiments
Introduction
Tables
MNIST
ImageNet
CIFAR-10
Contributing Measurements
Steps showing how to make contributions

Figure 4.13: Website structure


Alina Vasilciuc minor text modifications

..
data
figs
my_icons
not_ready_notebooks
scripts
2020-03-25-CIFAR-10_Classification.ipynb
2020-03-25-ImageNet_Classification.ipynb
2020-03-25-MNIST_Classification.ipynb
2020-04-09-Contributing_Measurements.ipynb
2020-04-30-Overview_of_experiments.ipynb
2020-04-30-Raw_Measurement.ipynb
2020-05-09-Theoretical_Analysis.ipynb
FAIR data.ipynb
README.md
license.txt
utils.py

Figure 4.14: GitHub repository overview

plot needs to appear on a specific page only then this code is called. This is because the same kind of plot is needed in several web pages, with minor modifications, therefore, to avoid code repetition, the code for the plot is defined only once in a python file and then called with the desired parameters from several web pages. Several script files store all methods needed to create all plots.

Chapter 5

Evaluation

This chapter will show certain visualizations that highlight the difference between what was predicted (level 0) and what was measured (level 3). Moreover, it will evaluate how accurate predictions were compared to the measured data points and if there are meaningful performance optimization techniques that can be drawn from this experimentation.

The terminology used in the plots and throughout this text will be the following explained in table 5.1.

5.1 Theoretical Pareto and Measured Pareto Overlapped

Concerning differentiating predicted from measured data points an overlapped Pareto plot was created, and it is illustrated in figures 5.1, I.6 and I.7 which show a theoretical Pareto and a measured Pareto plot overlapped, for all three machine learning tasks (MNIST, ImageNet and CIFAR-10). This plot can easily illustrate the differences between what was predicted and what was measured.

As this is a Pareto plot, the x-axis shows performance in frames per second and the y-axis shows the top-1 accuracy. Moreover, the orange Pareto frontier shows the theoretical data points, while the blue Pareto frontier shows the measured ones.

For MNIST, on the measured Pareto frontier, most data points are BISMO and FINN (these two have the same predicted performance; hence they are overlapped) and one TX2 data point. On the measured frontier, most data points are ZCU-104FINN; there is also one ZCU-104BISMO data point and one NCS2 data point.

It would be expected that the theoretical Pareto frontier would be at the right of the measured Pareto frontier, which happens both for ImageNet and CIFAR-10 overlapped Pareto plots, but not for MNIST. MNIST classification got better measurements than what was predicted. The reason for this will be explained in section 5.3.

Table 5.1: Terminology used throughout this effort is explained in this table. For instance u96-quadcore-a53_int4_mlp_25 refers to the hardware platform U96 Quadcore A53, the model implemented on this platform is the MLP with the INT4 data type and pruning scale at 25%.

Keyword	Explanation
u96-quadcore-a53_int4_mlp_25	Hardware Platform: U96 Quadcore A53 Datatype: INT4 Neural Network: MLP Pruning scale: 25%
u96-quadcore-a53_int4_mlp_50	Hardware Platform: Ultra96 Quadcore A53 Datatype: INT4 Neural Network: MLP Pruning scale: 50%
tx2-maxq_fp32_mlp_25	Hardware Platform: TX2 Hardware mode: max-q Datatype: FP32 Neural Network: MLP Pruning scale: 25%
tx2-maxp_fp16_cnv_12.5	Hardware Platform: TX2 Hardware mode: max-p Datatype: FP16 Neural Network: CNV Pruning scale: 12.5%
edgetpu-fast_int8_efficientnet-l_100	Hardware Platform: USB Accelerator Hardware mode: Fast Clock Datatype: INT8 Neural Network: EfficientNet L Pruning scale: 100% (no pruning)
zcu104-bismo_int2 mlp_12.5	Hardware Platform: ZCU104 Implemented Architecture: BISMO Datatype: INT2 Neural Network: MLP Pruning scale: 12.5%

This plot is interactive, so it is possible to zoom in, zoom out, drag the chart around, see on-plot tooltips, and select which platform types it is desired to hide.

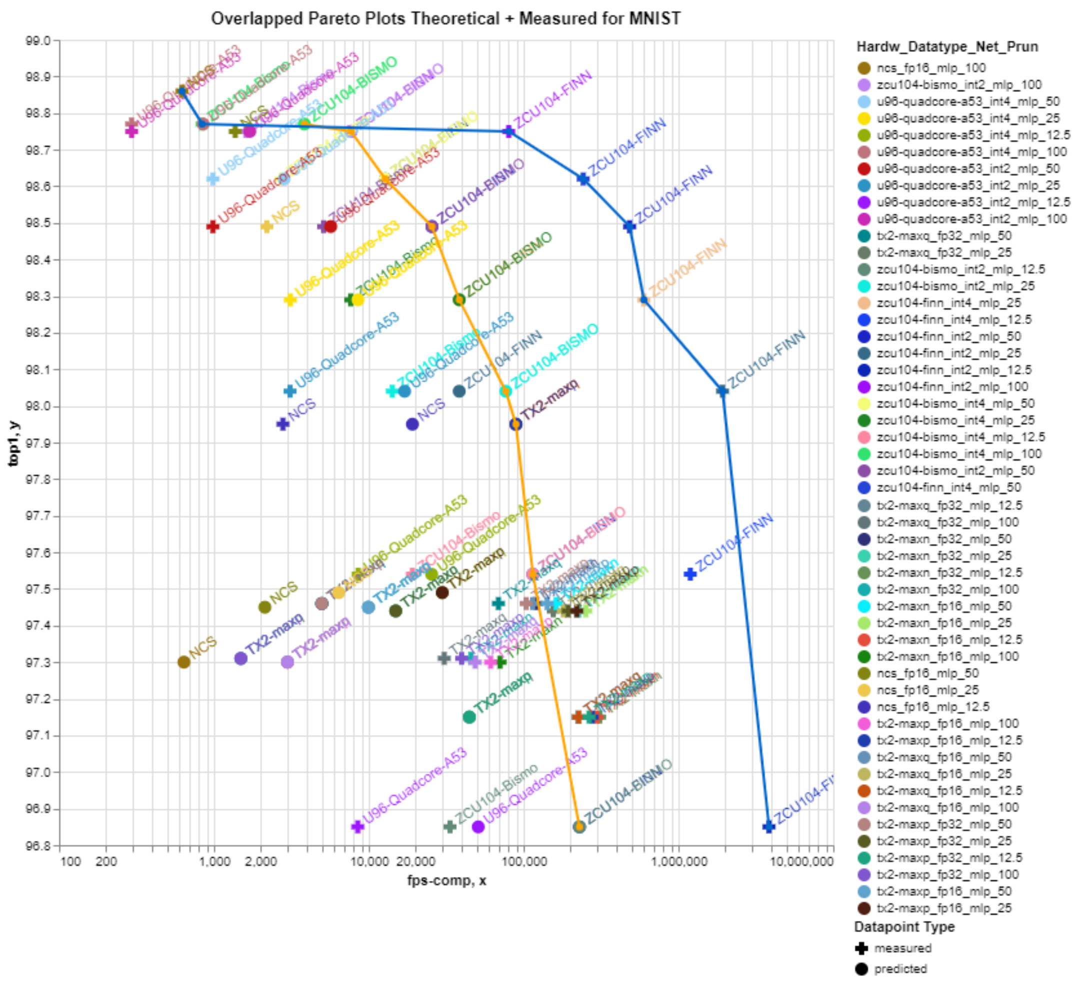


Figure 5.1: Overlapped pareto plot for MNIST classification. Orange pareto frontier shows the theoretical data points and the blue pareto frontier shows the measured data points. Figure available at [5].

5.2 Efficiency plots

In order to have a better comprehension of what was theoretically predicted and what was really measured, efficiency bar-charts were created (figures 5.2, I.8 and I.9). There are efficiency bar charts for MNIST, ImageNet and CIFAR-10, but for a matter of space optimization only MNIST bar charts are showcased, the rest can be fully visualized on the website. In these plots, what was predicted was usually higher than what was measured. The size of each bar corresponds to the absolute performance. All predicted are shown in red, all measured data points in orange and the theoretical peak performance of the hardware platform in blue. The percentages shown on top of the blue bars correspond to the efficiency achieved as a percentage of the predicted performance. Measured performance can exceed 100% of the predicted performance because all theoretical predictions take memory bottlenecks into account.

The percentage which is shown on top of each bar is the efficiency, and it was calculated according to $Efficiency = \frac{measured\ performance}{predicted\ performance} * 100$.

Efficiency plots were drawn separately per machine learning task (ImageNet, MNIST and CIFAR-10), afterwards split per hardware type. For each machine learning task, there is the FPGAs plot, the GPUs plot and the USB devices plot.

An interesting phenomenon occurs in these plots. In the TX2 bar chart (fig. I.9) for MNIST all measured values for hardware and topology combinations are above the predicted ones, which also happens for FINN in figure 5.2. This phenomenon takes place for MNIST classification alone. A more in-depth explanation will be given in section 5.3 as to why this happens.

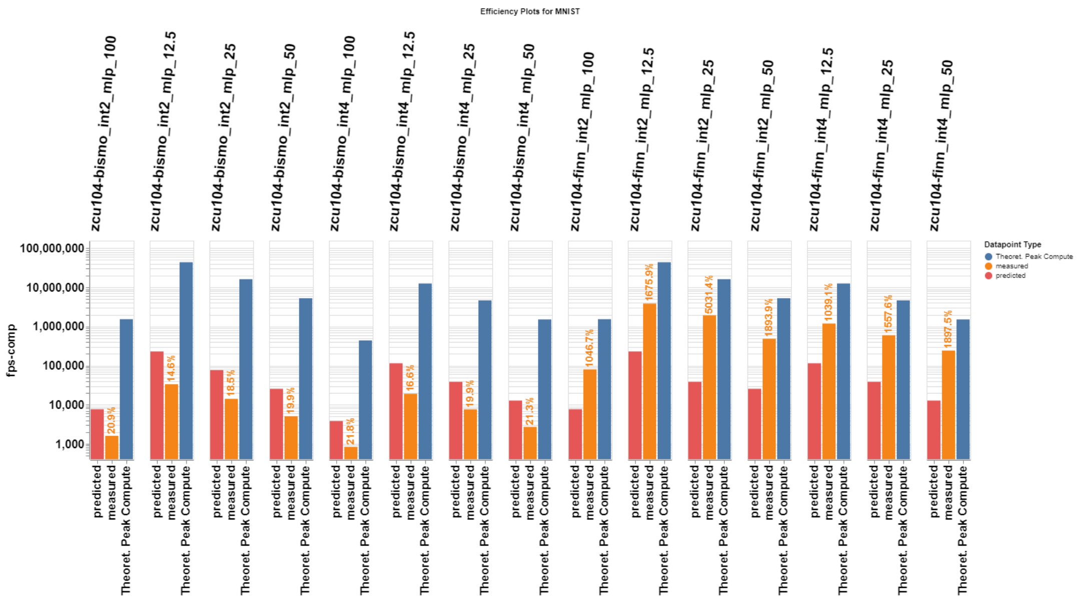


Figure 5.2: Bar chart showing Efficiency for MNIST classification over the FPGAs, more specifically on the ZCU104-FINN, ZCU104-BISMO. Figure available at [5].

5.3 Evaluation between theoretical and measured

Regarding performance predictions, heatmaps were created as an easy, visual way to present them. Measured values are displayed in a different variety of visualizations. For instance, several boxplots show latency, throughput and power per hardware platform and topology combination. Moreover, a Pareto frontier is also presented with the performance (x-axis) versus the accuracy (y-axis) for the measured data points.

For the comparison between predicted data points versus the measured ones, an overlapped Pareto plot was created with both predicted and measured values. Efficiency plots in the form of bar charts were also created where predicted performance is measured in frames per second on the y-axis and compare to the measured performance (frames per second) for the same topology and hardware combination.

5.3.1 FPGA Platforms

Three FPGA platforms were benchmarked: BISMO, FINN and DPU. BISMO and FINN were benchmarked on CIFAR-10 and MNIST classification, while the DPU was benchmarked on the ImageNet classification.

BISMO on the CIFAR-10 classification had very low measured values compared to the predicted ones (between 0% and 1% as per table 5.2). The reason behind this is that the implementation only uses a fraction of BISMO resources, with no time to implement a larger prototype. In the future, it could be beneficial and exciting to implement it with the full range of FPGA resources. On the MNIST classification, however, got values between 15% and 22% of what was predicted, as per table 5.3.

FINN on the CIFAR-10 classification achieved a performance between 8% and 35% of the predicted values, as per table 5.2. On the MNIST classification, however, got between 10 to 50 times the predicted values, as per table 5.3. After performing quantization, the entire MLP model fit in the on-chip memory, contrary to what was initially assumed for this model. Initially, it was considered that the whole model would reside in the off-chip memory, being memory-bound and that memory would bottleneck performance. So the operational intensity would be very low due to this. However, as the entire model fit in the on-chip memory the operational intensity was much higher, memory bottleneck was avoided altogether, and consequently, the performance was way higher than what was predicted, between 10 to 50 times higher.

The DPU (Ultra96 and ZCU102 using the DPU architecture) on the ImageNet Classification got measured values between 6% and 26% of the predicted ones, as per table 5.4.

Overall, the efficiency across all platforms seems to be constant at an average of roughly around 19%. This low efficiency on the FPGAs might be due to its vast implementation flexibility, where performance predictions assume 100% resource utilization; however, in reality, that might not be the case. Also, the clock frequency might not be at the maximum theoretical peak.

In general, efficiency predictions decrease with increased pruning. This observation is precise for ultra96-dpu-resnet50, ncs2-cnv, zcu104-finn-cnv and tx2-cnv. Another critical aspect can be drawn: the increased the quantization, the less accurate predictions become, which is also very visible for resnet-50-cnv and tx2-mlp.

Efficiency predictions can be improved by adjusting the resource utilization and clock frequency for peak performance within the model assumption. Regarding MNIST, measured performance exceeds predictions, because predictions were made assuming that the model would remain in the off-chip memory. However, this was not the case as the quantized versions of the model stayed in the on-chip memory, eradicating the memory bottleneck and increasing performance. To improve the prediction quality for heavily quantized models, the solution would be to either ignore memory constraints altogether or introduce an on-chip memory capacity assumption in future work.

Overall it is visible that level-0 predictions show an average of around 10%-20% of the measured performance. As the pruning scale goes up, predictions become less accurate, and the same can be verified for quantization. For future work, performance predictions can be improved by taking the resource utilization into account and the clock frequency of the FPGA circuit also into account. Another improvement would be to create an on-chip memory, and then it would only be necessary to compare the model size with the on-chip memory size and decide to ignore the memory access.

Table 5.2: Efficiency results for the FPGAs for all pruned and quantized variants of CNV.

Hardware	Maximum	Median	Average	Minimum
zcu104-finn	34.5	15.1	18.7875	7.9
zcu104-bismo	0.4	0.3	0.3	0.1

Table 5.3: Efficiency results for the FPGAs for all pruned and quantized variants of MLP.

Hardware	Median	Average	Minimum	Maximum
zcu104-bismo	19.9	19.18	14.6	21.8
zcu104-finn	1675.9	2020.3	1039.1	5031.4

Table 5.4: Efficiency results for the FPGAs for all topologies of ImageNet.

Hardware	Median	Average	Minimum	Maximum
ultra96-dpu	20.0	18.259	12.7	21.7
zcu102-dpu	25.549	25.5497	25.4	25.7
zcu104-dpu	14.55	14.55	5.9	23.2

5.3.2 GPU Platforms

The GPU platform benchmarked is the Jetson TX2 with its three modes: max-n, max-p and max-q. The max-n mode is meant for full clock speed, meaning that it is throughput oriented. The max-q mode is very power consumption-oriented. The max-p mode is a hybrid between max-q and max-n promoting power and performance balance.

5.3.2.1 CIFAR-10

The topology benchmarked is the CNV not pruned and pruned at 50%, 25% and 12.5%. For CIFAR-10 classification, efficiency ranges between 25.7% and 88.9% with an average of 60%, as per table 5.5; this means that measured values were, on average, 60% of the predicted values.

Results make evident that the more pruned the model is, the more inaccurate predictions are, this is visible across all model variants, for instance, on TX2-maxn-FP16-CNV-100 had an efficiency of 79%. In contrast, TX2-maxn-FP16-CNV-12.5 had an efficiency of 26%.

Regarding quantization, prediction accuracy decreases with the increase in quantization; this is also evident across all model variants; for instance, the tx2-maxn-fp16-cnv-100 got 78.8% efficiency, whereas the tx2-maxn-fp32-cnv-100 got 86.7% efficiency.

Regarding performance (fps), the maximum measured value achieved by the GPUs in the CIFAR-10 machine learning task was achieved by TX2-maxn with a total of 34197.2 fps on tx2-maxn-fp16-cnv-12.5. The second best 29763.1 fps tx2-maxp-fp16-cnv-12.5. Which makes sense according to the max-n, max-p and max-q characteristics.

Overall predictions are relatively constant across all CNV variants. Predictions efficiency decreases with pruning and quantization, which, again, could be caused by the limitations of the memory model in the theoretical analysis.

Table 5.5: Efficiency results for the TX2s for all pruned and quantized variants of CNV.

Hardware	**Maximum**	**Median**	**Average**	**Minimum**
TX2 max-n	86.7	63.8	59.75	25.7
TX2 max-p	88.4	64.15	60.2375	25.9
TX2 max-q	88.9	65.3	61.425	26.8

5.3.2.2 MNIST

The topology benchmarked was MLP not pruned and pruned at 50%, 25% and 12.5%. For MNIST classification, efficiency values range between 254.8% and 3069.4% averaging at 1362%, as per table 5.6. These values mean that measured performance exceeds predictions by 2.5 and 30.7 times which is not usual. However, this also happened with the FPGAs. Again, this occurred because performance predictions assume a very low operational intensity for MLP. After all, they are quite memory heavy, predicting that they will be memory bound. However, this did not happen because the model fit in the on-chip memory of the GPU, eradicating the memory bottleneck, which increased the measured performance considerably above the predicted one.

In this case, the more pruned the topology is, the more inaccurate predictions are, for instance, tx2-maxn-fp16-mlp-100 got 2365.5%, whereas the tx2-maxn-fp16-mlp-12.5 got 333.4%.

Drawn plots also make evident that the more quantized the topology is, the less accurate predictions are, for instance, tx2-maxn-fp32-mlp-100 got 3069.4% whereas the tx2-maxn-fp16-mlp-100 got 2365.5%.

The maximum measured performance (fps) achieved by the GPUs on the MNIST classification was achieved by tx2-maxn-fp32-mlp-12.5 with 309646 fps, and the second best-measured performance was achieved by tx2-maxn-fp16-mlp-12.5 with 297674 fps.

The best result was achieved on the max-n mode, which was expected as this is the model with the highest performance. Another thing is that the best results and the second-best result were both obtained with a severely pruned variant of the topology.

Overall predictions were constant across MLP variants. In this classification, it is concluded that prediction efficiency decreases with pruning and quantization. This observation could be due to the limitations in the memory model in the assumptions for the theoretical analysis.

Table 5.6: Efficiency results for the TX2s for all pruned and quantized variants of MLP.

Hardware	Median	Average	Minimum	Maximum
tx2-maxn	1577.75	1610.3125	333.4	3069.4
tx2-maxp	1361.35	1397.11	311.4	2666.8
tx2-maxq	1095.1	1078.41	254.8	2060.8

5.3.2.3 ImageNet

The topologies benchmarked were GoogLeNet-v1 and ResNet-50 both on full precision (FP32) and half-precision (FP16). They were benchmarked across all three operating modes of the GPU (max-n, max-p and max-q). Pruning was not applied to any of these models, so it is not possible to draw any conclusions from that.

For ImageNet classification, the efficiency values range between 65% and 84.6%, averaging at around 75%, as per table 5.7, which means that measured values were, on average, 75% of the predicted values.

Table 5.7: Efficiency results for the TX2s for all topologies of ImageNet.

Hardware	Median	Average	Minimum	Maximum
tx2-maxn	74.6	73.925	65.0	81.5
tx2-maxp	75.75	75.075	65.5	83.3
tx2-maxq	77.55	76.824	67.6	84.6

GoogLeNet-v1 efficiency values ranged between 65% and 75.8%. ResNet-50 efficiency values ranged between 76% and 84.6%. These values mean that estimations were relatively accurate

The increase in quantization means the decrease in efficiency predictions, for instance tx2-maxn-fp32-googlenetv1-100 got 73.2% whereas the tx2-maxn-fp16-googlenetv1-100 got 65%. This observation is seen across all ImageNet measurements.

Overall, predictions were quite accurate for ImageNet classification with all prediction efficiencies above 65%. What is visible both for CIFAR-10 and MNIST is that the higher the pruning scale, the lower the prediction efficiency. This is visible for TX2 max-n, max-p and max-q. It is visible for all experiments (CIFAR-10, MNIST and ImageNet) that the more quantized the topology is, the lower the prediction efficiency is.

In summary, theoretical predictions were leveraged to predict performance over several pruning variants and quantization variants. Predictions reflect the benefits of optimizations - pruning and quantization - even though a bit too optimistically. Moreover, it is predicted that the built-in memory assumptions negatively influenced results and is key to the differences between predictions and measured performance.

5.3.3 Edge TPU and NCS2 Platforms

In regards to CIFAR-10 the NCS2 got efficiency values between 2.9% corresponding to ncs2-fp16-cnv-12.5, and 25.6% corresponding to ncs2-fp16-cnv-100. This shows, once again, that the increased pruning lowers the efficiency of the prediction. For MNIST the NCS2 got values between 97%, corresponding to ncs2-fp16-mlp-100, and 14.6% corresponding to the pruned version (ncs2-fp16-mlp-12.5). For ImageNet classification, the NCS2 achieved a 28.2% of the predicted performance on the ResNet-50 topology.

The EdgeTPU, on the other hand, was only benchmarked with the ImageNet classification. It was also not possible to check for pruning effect on predictions efficiency because there was no pruning applied or quantization. It is only possible to conclude that MobileNet-v1 achieved an average of 13% of the predicted performance, whereas GoogLeNet-v1 achieved an average of 24% of the predicted performance. EfficientNets also achieved an average of 24% of the predicted performance. What it is possible to observe is that smaller topologies like MobileNet-v1 achieved a lower efficiency.

In conclusion, performance predictions for smaller topologies are relatively low, which is quite visible on the efficiency numbers for the pruned variants (12.5%). For this reason, it is concluded that predictions were overly optimistic. Overall efficiency values for the NCS2 and the EdgeTPU are lower on average than the FPGAs and the GPUs.

Table 5.8: Efficiency results for the NCS2 for all topologies of ImageNet.

Hardware	**Maximum**	**Median**	**Average**	**Minimum**
ncs	25.6	12.65	13.45	2.9

Table 5.9: Efficiency results for the NCS2 for all pruned and quantized variants of MLP.

Hardware	**Median**	**Average**	**Minimum**	**Maximum**
ncs	49.3	52.575	14.6	97.1

Table 5.10: Efficiency results for the NCS2 and USB Accelerator for all topologies of ImageNet ML task.

Hardware	**Median**	**Average**	**Minimum**	**Maximum**
edgetpu-fast	20.6	19.04	11.1	23.5
edgetpu-slow	26.5	24.18	14.0	27.4
ncs	28.2	28.2	28.2	28.2

Chapter 6

Conclusions and Future work

This effort presents the work done on creating several kinds of data visualizations, their deployment to a web portal and the results from benchmarking the EfficientNets on Google's USB Accelerator.

A way to better understand all gathered data was proposed and successfully created. More specifically, several kinds of visualizations were designed to provide insights into the recent compute architectures and the trade-offs between algorithmic optimizations and hardware choices. Almost all visualizations are interactive and were deployed to a web portal where all predicted and measured data is available. The web portal adheres to the FAIR guiding principles, by owning metadata and a DOI and by supporting contribution (a set of guiding steps are available for anyone who wishes to contribute).

As part of this effort, the EffcientNets were benchmarked on Google's USB Accelerator, and accuracy, performance and power consumption measurements were taken. There were two types of measured performances: system performance and computing performance. An inference script was used to which the model and the validation set were fed. This script outputted the results from the inference over one image, and the process repeats for all them. At the same time, this process is timed to get performance results. Obtained results were very close to the ones that Google reports out, with the computing performance always being faster system performance. This makes sense because computing performance only accounts for the time the inference call takes. In contrast, system performance accounts for opening the image, creating a tensor from it and calling inference. Moreover, EfficientNet S is the fastest model benchmarked out of the three, due to its small size. At the same time, EfficientNet L is the slowest model because it is the biggest one with over 10 million elements.

Power measurements were taken with the UM34C power meter. The USB Accelerator was plugged into the power meter, which was plugged into the computer, and the instant power was recorded at every one second. The experiments were repeated several times in order not to miss any critical peak power consumption value. The board consumes the most running at maximum frequency, which is the fastest mode, and EfficientNet L is the one that drains more power, as it was expected, again, due to its size and GOPs. In general, the board drains around 2 Watts while doing inference.

Accuracy measurements were taken using two scripts, one written in python that performs inference and records the outputs and another one written in bash scripting that verifies the outputs from the previous script and calculates the accuracy. Accuracy results obtained were relatively lower than the ones reported by Google, as it was not possible to replicate Google's results.

Besides plotting measured data points, predicted data points that were created by level 0 of QuTiBench were also plotted. The predicted data points were later compared with the measured ones with the help of overlapped Pareto plots and efficiency bar charts.

In general theoretical predictions successfully predict performance trends. BISMO had very low measured values compared to the predicted ones due to the lack of resources in the implementation. The MLP variants got higher performance than what was predicted because the memory bottleneck was avoided altogether. Therefore FINN measured performance exceed 10 to 50 times the predicted performance and the TX2 measured performance exceeds 2.5 to 30.7 times the predicted performance. Again, since MLPs are so memory-heavy, performance predictions assume an extremely low operational intensity expecting that they will be memory bound. However, this did not happen as these models fit in the hardware memory as they were heavily quantized. The FPGAs' efficiency is around 19%; this can be due to the vast implementation flexibility, where performance predictions assume a resource utilization of 100%, which may not be happening and the clock speed might also not be at the maximum theoretical peak. Moreover, efficiency numbers for the USB Accelerator and the NCS2 are lower than for FPGAs.

Overall level-0 predictions show an average of 10%-20% of the measured performance. What is sensed across all hardware platforms is that performance predictions become more inaccurate with the increased pruning. Performance predictions also become more inaccurate with the increased quantization.

In future work, level 0 of QuTiBench could be improved. More precisely, the memory model could be refined regarding assumptions about what data is stored in the on and off-chip memory. Meaning that we could introduce an on-chip memory capacity assumption, if the model fits on-chip, memory access would then be ignored. This solution could provide more accurate estimates for the heavily quantized CNNs. Alternatively, memory bottleneck could be removed altogether as given through the roofline model and only consider the peak compute performance for performance predictions. Moreover, for the FPGAs, performance predictions can be further improved by taking the actual resource utilization and the achieved clock frequency of the FPGA circuit into account.

Concluding, for the pruned variants, there are limitations in regards to the memory model. While performance estimation was overly optimistic, especially for the pruned and quantized variants, predictions for the Pareto optimal points provide rich insights into all the combinations between topologies, hardware and optimization strategies which can save a considerable amount of implementation time to the designer. Therefore, it is concluded that theoretical baselines can provide good baselines in ML benchmarking. In conclusion, several improvements to level 0 of QuTiBench were drawn from these visualizations, which could make it more accurate at predicting performance in the future.

Regarding power consumption from all measurements, it is possible to conclude that the NCS2 and the USB Accelerator are among the platforms that consume less energy, in particular between 1 and 3 Watts. The peak power consumption for TX2 is between 5 and 20 Watts and for the FPGAs is between 10 to 20 Watts. It is concluded that, as is was expected, ASICs own lower power consumption in comparison with GPUs and FPGAs.

Regarding the throughput visualization with box and whiskers, both for MNIST and CIFAR-10, the ZCU104-FINN-INT2 achieves the higher throughput outperforming TX2, NCS2, A53, and other less quantized versions of MLP on the ZCU104-FINN. With ImageNet, it is the INT8-ZCU102-DPU that owns the highest throughput. In general, the more quantized the model is, the higher the throughput achieved. Regarding pruning, the more pruned the model is, the higher the throughput achieved.

Regarding latency, ZCU104-FINN-INT2 shows the lowest latency for the MNIST classification, whereas, for the CIFAR-10, both TX2 and FPGAs show quite the same latency. On the ImageNet classification, however, the NCS2 is leading the way. Overall latency increases with the less pruned and quantized versions of the model.

The web portal is up and running, with almost all visualizations being interactive, which provide insights into what topology will give what performance in what hardware. More measurements were successfully taken, in particular to the USB Accelerator. In conclusion, all objectives were met.

Bibliography

[1] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks. "Fathom: Reference workloads for modern deep learning methods." In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2016, pp. 1–10.

[2] C. C. Aggarwal. *Neural Networks and Deep Learning. A Textbook*. Springer, 2018. ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.

[3] "asplos." In: (). URL: https://dl.acm.org/action/showFmPdf?doi=10.1145\%2F3229762. Accessed: 24.11.2020.

[4] A. Baevski and M. Auli. "Adaptive input representations for neural language modeling." In: *arXiv preprint arXiv:1809.10853* (2018).

[5] M. Blott, M. Leeser, L. Doyle, J. Kath, L. Halder, Z. Neveu, and A. Vasilciuc. "QuTiBench: Benchmarking Neural Networks on Heterogeneous Hardware." In: (). URL: https://rcl-lab.github.io/QutibenchWeb/. Accessed: 21.09.2020.

[6] M. Blott, T. B. Preusser, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers. "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks." In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11.3 (2018), p. 16.

[7] M. Blott, L. Halder, M. Leeser, and L. Doyle. "QuTiBench: Benchmarking neural networks on heterogeneous hardware." In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.4 (2019), pp. 1–38.

[8] M. Blott, N. J.Fraser, G. Gambardella, L. Halder, J. Kath, Z. Neveu, Y. Umuroglu, A. Vasilciuc, M. Leeser, and L. Doyle. "Evaluation of Optimized CNNs on Heterogeneous Accelerators using a Novel Benchmarking Approach." In: (2020). DOI: 10.1109/TC.2020.3022318.

[9] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam. "BenchNN: On the broad potential application scope of hardware neural network accelerators." In: *2012 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2012, pp. 36–45.

[10] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. "Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark." In: *ACM SIGOPS Operating Systems Review* 53.1 (2019), pp. 14–25.

[11] "Collective Knowledge Framework." In: (). URL: http://cknowledge.org/. Accessed: 24.11.2020.

[12] "Coral." In: (). URL: https://coral.ai/models/. Accessed: 26.10.2020.

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

[14] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. "The scalable heterogeneous computing (SHOC) benchmark suite." In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM. 2010, pp. 63–74.

[15] "DeepBench." In: (). URL: https://svail.github.io/DeepBench/. Accessed: 24.11.2020.

[16] C. Digital. "Machine for Machine Intelligence." In: (2020).

[17] M. Dubois and F. A. Briggs. "Performance of synchronized iterative processes in multiprocessor systems." In: *IEEE Transactions on Software Engineering* 4 (1982), pp. 419–431.

[18] "fastai." In: (). URL: https://github.com/fastai/fastpages. Accessed: 24.11.2020.

[19] S. Ghose, A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, et al. "What your DRAM power models are not telling you: Lessons from a detailed experimental study." In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2.3 (2018), p. 38.

[20] "Google Coral classification image example." In: (). URL: https://gist.github.com/ayyucedemirbas/2901b48a1b33eec1fd4794a522c7e204. Accessed: 15.11.2020.

[21] "Google Performance results." In: (). URL: https://coral.ai/docs/edgetpu/benchmarks/. Accessed: 19.11.2020.

[22] S. Han. "Efficient Methods and Hardware for Deep Learning." In: (). URL: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/company/Events/summits/enns/enn2017-son-hang-stanford-cp.pdf. Accessed: 25.11.2020.

[23] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. "EIE: efficient inference engine on compressed deep neural network." In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 243–254.

[24] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang. "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers." In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. June 2015, pp. 27–40. DOI: 10.1145/2749469.2749472.

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

[26] M. Horowitz. "1.1 computing's energy problem (and what we can do about it)." In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE. 2014, pp. 10–14.

[27] *ICT-Energy Strategic Research Agenda*. 2016.

[28] "Intel patent." In: (). URL: https://patents.justia.com/patent/10152325. Accessed: 24.11.2020.

[29] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. "In-datacenter performance analysis of a tensor processing unit." In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 2017, pp. 1–12.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[31] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. "Handwritten digit recognition: Applications of neural network chips and automatic learning." In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[33] J. Lin and B. C. Lai. "BRAM efficient multi-ported memory on FPGA." In: *VLSI Design, Automation and Test(VLSI-DAT)*. 2015, pp. 1–4. DOI: 10.1109/VLSI-DAT.2015.7114526.

[34] Y. Liu, H. Zhang, L. Zeng, W. Wu, and C. Zhang. "MLbench: benchmarking machine learning services against human experts." In: *Proceedings of the VLDB Endowment* 11.10 (2018), pp. 1220–1232.

[35] M. Mathieu, M. Henaff, and Y. LeCun. "Fast training of convolutional networks through ffts." In: *arXiv preprint arXiv:1312.5851* (2013).

[36] "Matrix Multiplier." In: (). URL: https://blog.raccoons.be/coral-tpu-jetson-nano-performance. Accessed: 24.11.2020.

[37] X. Mei and X. Chu. "Dissecting GPU memory hierarchy through microbenchmarking." In: *IEEE Transactions on Parallel and Distributed Systems* 28.1 (2016), pp. 72–86.

[38] "MLPerf." In: (). URL: https://mlperf.org/about/#philosophy. Accessed: 24.11.2020.

[39] M. P. Raghunath Nambiar. *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence*. https://www.springer.com/gp/book/9783030114039. Springer, 2018.

[40] "RCL-lab GitHub repository." In: (). URL: https://github.com/RCL-lab. Accessed: 02.11.2020.

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[42] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. "Overfeat: Integrated recognition, localization and detection using convolutional networks." In: *arXiv preprint arXiv:1312.6229* (2013).

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

[44] "SPEC." In: (). URL: http://spec.org/. Accessed: 24.11.2020.

[45] "STREAM." In: (). URL: https://www.cs.virginia.edu/stream/. Accessed: 24.11.2020.

[46] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. "Efficient processing of deep neural networks: A tutorial and survey." In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[48] M. Tan and Q. V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In: *arXiv preprint arXiv:1905.11946* (2019).

[49] R. Tang, W. Wang, Z. Tu, and J. Lin. "An experimental analysis of the power consumption of convolutional neural networks for keyword spotting." In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 5479–5483.

[50] J.-H. Tao, Z.-D. Du, Q. Guo, H.-Y. Lan, L. Zhang, S.-Y. Zhou, L.-J. Xu, C. Liu, H.-F. Liu, S. Tang, et al. "BenchIP: Benchmarking Intelligence Processors." In: *Journal of Computer Science and Technology* 33.1 (2018), pp. 1–23.

[51] "Tflite repository inside google coral github." In: (). URL: `https://coral.ai/docs/edgetpu/benchmarks/`. Accessed: 20.11.2020.

[52] S. Thomas, C. Gohkale, E. Tanuwidjaja, T. Chong, D. Lau, S. Garcia, and M. B. Taylor. "CortexSuite: A synthetic brain benchmark suite." In: *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2014, pp. 76–79.

[53] A. Torralba, R. Fergus, and W. T. Freeman. "80 million tiny images: A large data set for nonparametric object and scene recognition." In: *IEEE transactions on pattern analysis and machine intelligence* 30.11 (2008), pp. 1958–1970.

[54] "TPC." In: (). URL: `https://svail.github.io/DeepBench/`. Accessed: 24.11.2020.

[55] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference." In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '17. ACM, 2017, pp. 65–74.

[56] Y. Umuroglu, L. Rasnayake, and M. Sjalander. "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing." In: *Field Programmable Logic and Applications (FPL), 2018 28th International Conference on*. FPL '18. 2018.

[57] Y. Umuroglu, D. Conficconi, L. Rasnayake, T. Preusser, and M. Sjalander. "Optimizing Bit-Serial Matrix Multiplication for Reconfigurable Computing." In: *ACM Transactions on Reconfigurable Technology and Systems* (2019).

[58] S. Williams. "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore." In: (2009).

[59] K. Zhao and X. Chu. "G-BLASTN: accelerating nucleotide alignment by graphics processors." In: *Bioinformatics* 30.10 (2014), pp. 1384–1391.

[60] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko. "Benchmarking and analyzing deep neural network training." In: *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2018, pp. 88–100.

ANNEX I

Annex 1 List of figures

This section will provide additional figures.

Figure I.1 shows the theoretical peak performance of a given hardware platform in a visual way.

Figures I.2 and I.3 show Performance Predictions for ImageNet and CIFAR-10 classification.

Figures I.4 and I.5 show the theoretical pareto for ImageNet and CIFAR-10 respectively.

Figures I.6 and I.7 show an overlapped pareto plot with the theoretical and the measured data points.

Figures I.8 and I.9 show the efficiency bar charts for the NCS2, TX2 and the A53.

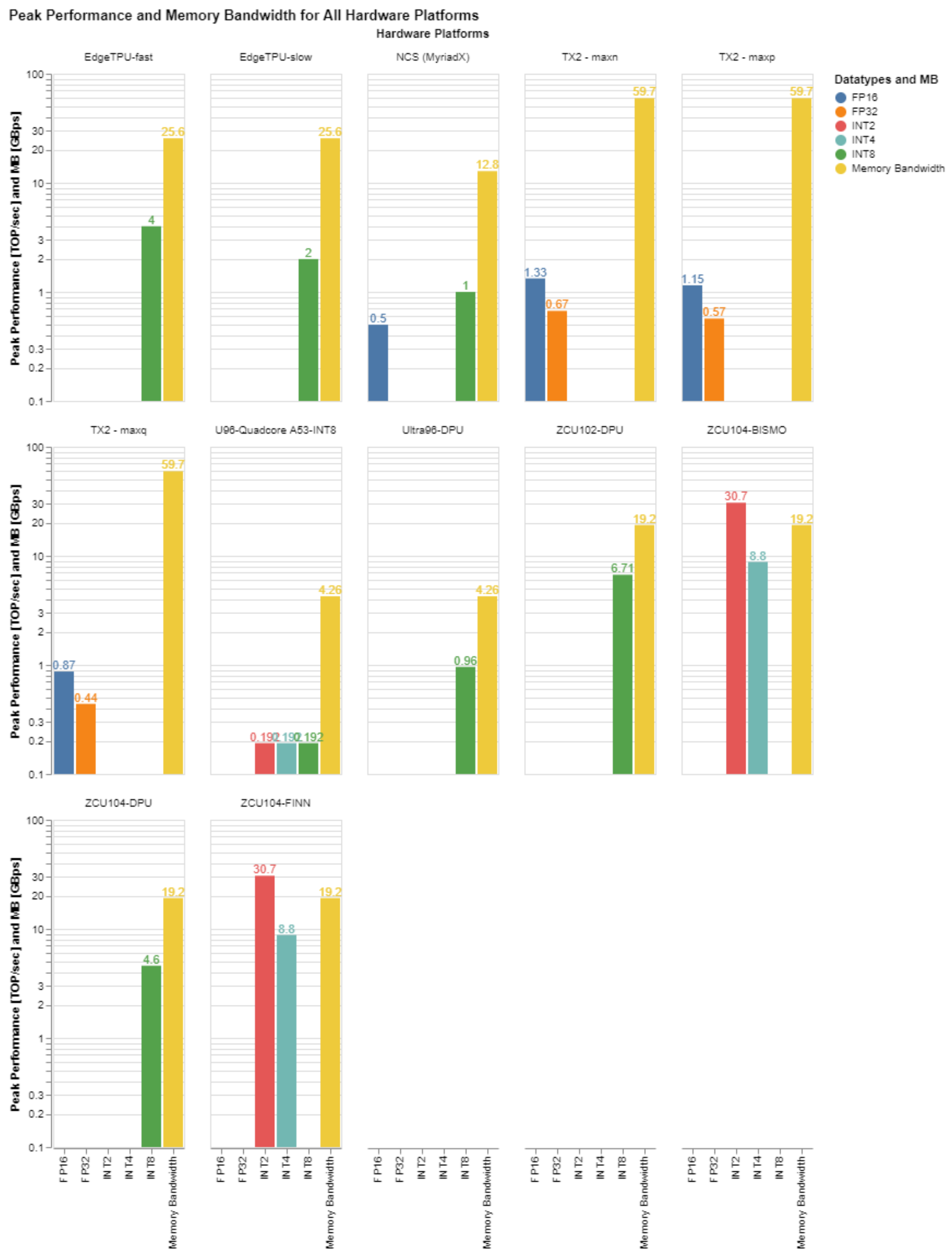


Figure I.1: Hardware platforms with their Peak Performance Predictions. Figure available at [5].

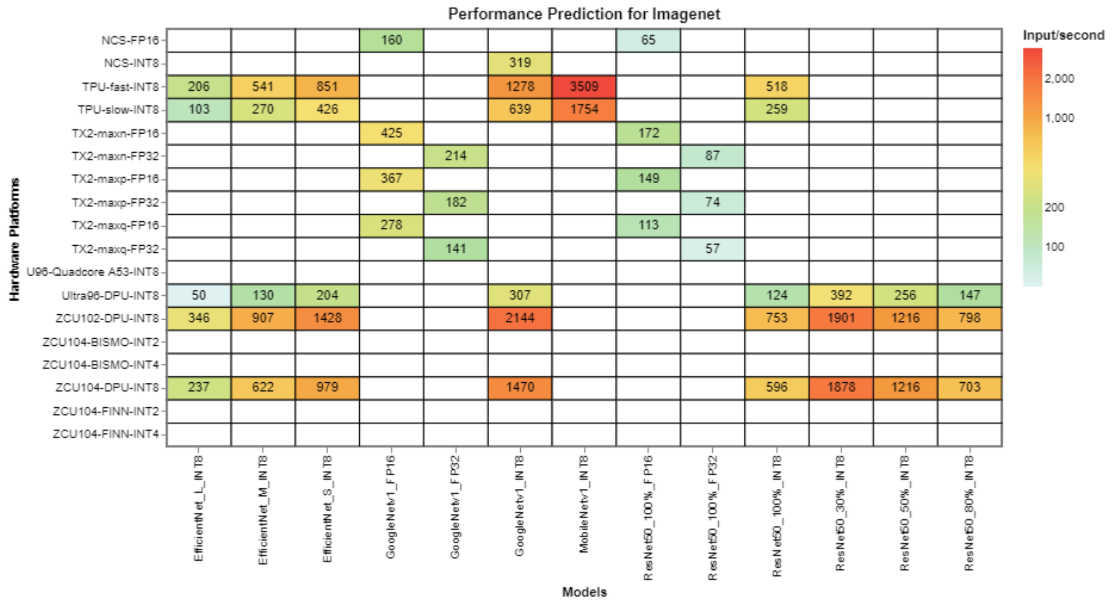


Figure I.2: Performance Predictions with Heatmap for ImageNet classification. Figure available at [5].

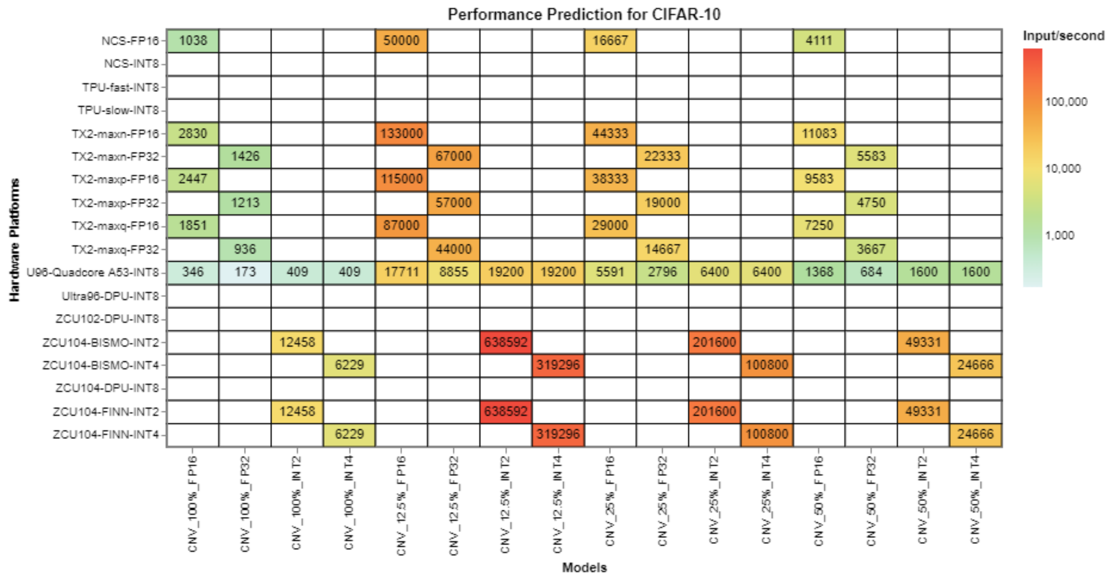


Figure I.3: Performance Predictions with Heatmap for CIFAR-10 classification. Figure available at [5].

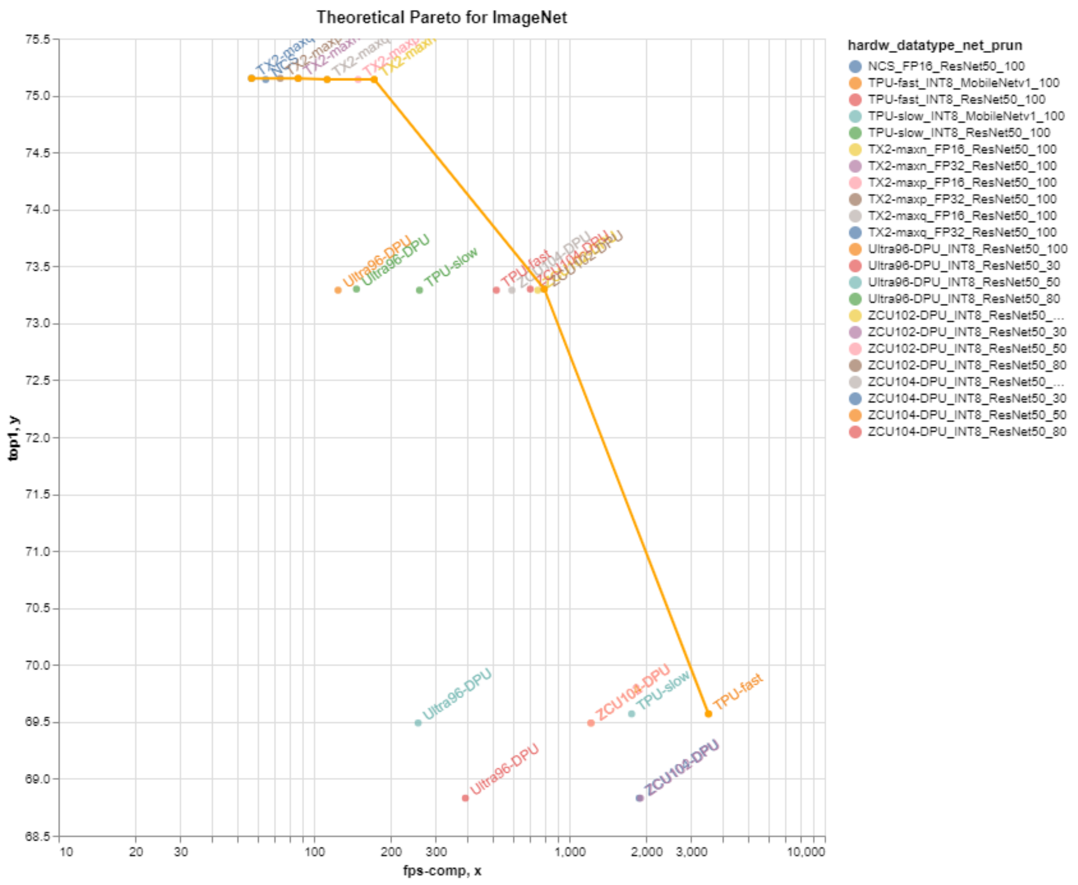


Figure I.4: Theoretical Pareto for ImageNet Classifications. Figure available at [5].

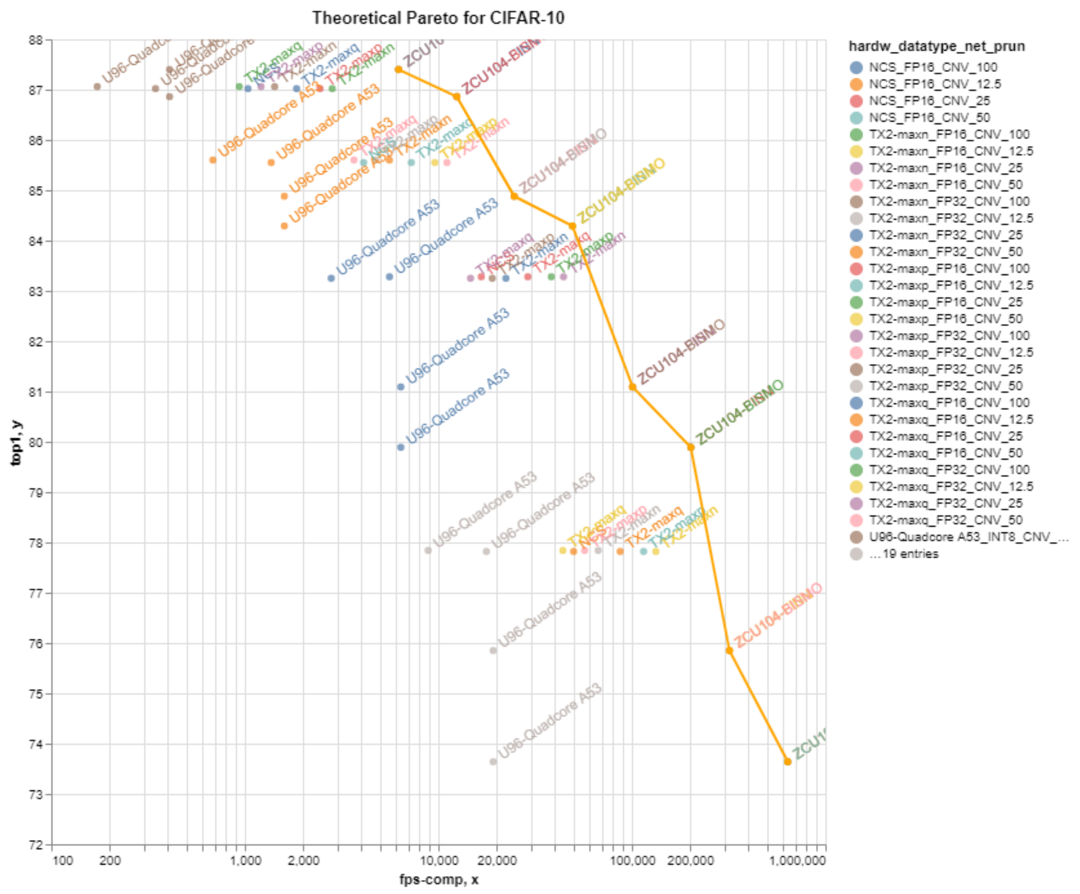


Figure I.5: Theoretical Pareto for CIFAR-10 Classifications. Figure available at [5].

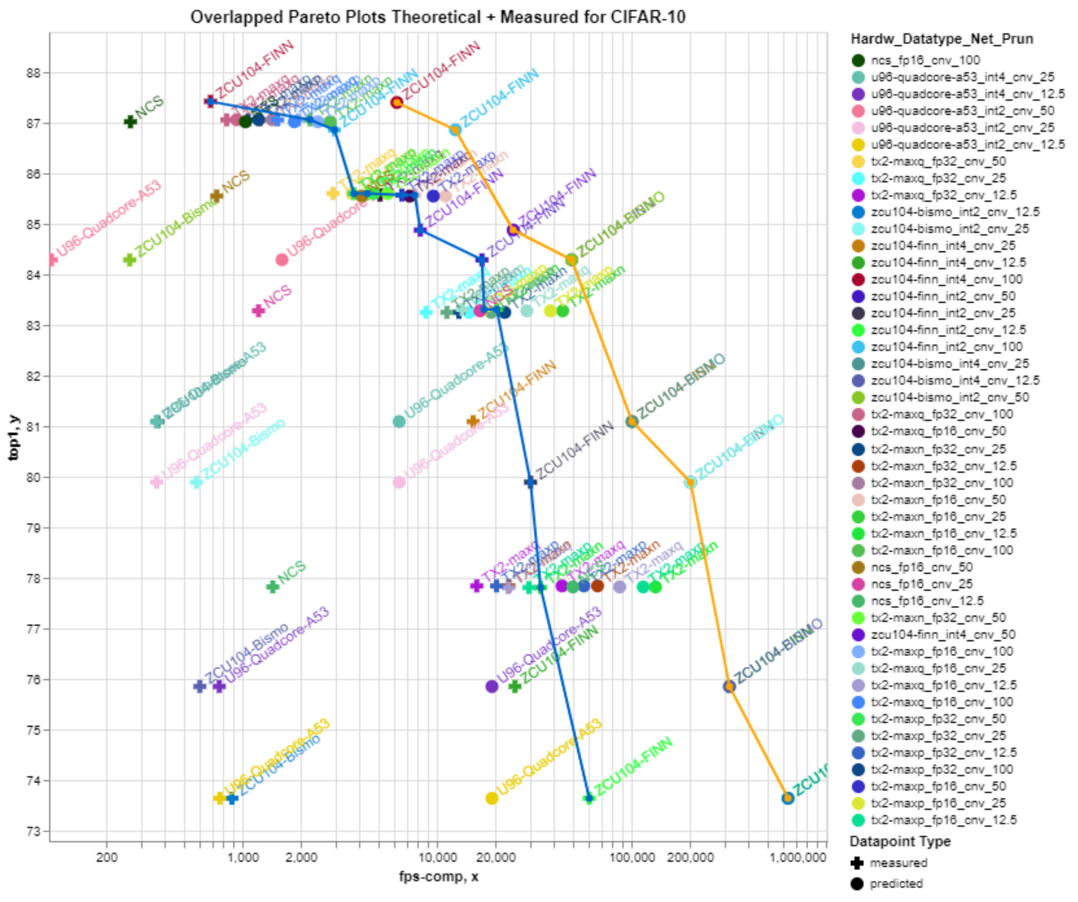


Figure I.6: Overlapped Pareto Plot for CIFAR-10 Classification. Orange pareto frontier shows the theoretical data points and the blue pareto frontier shows the measured data points. Figure available at [5].

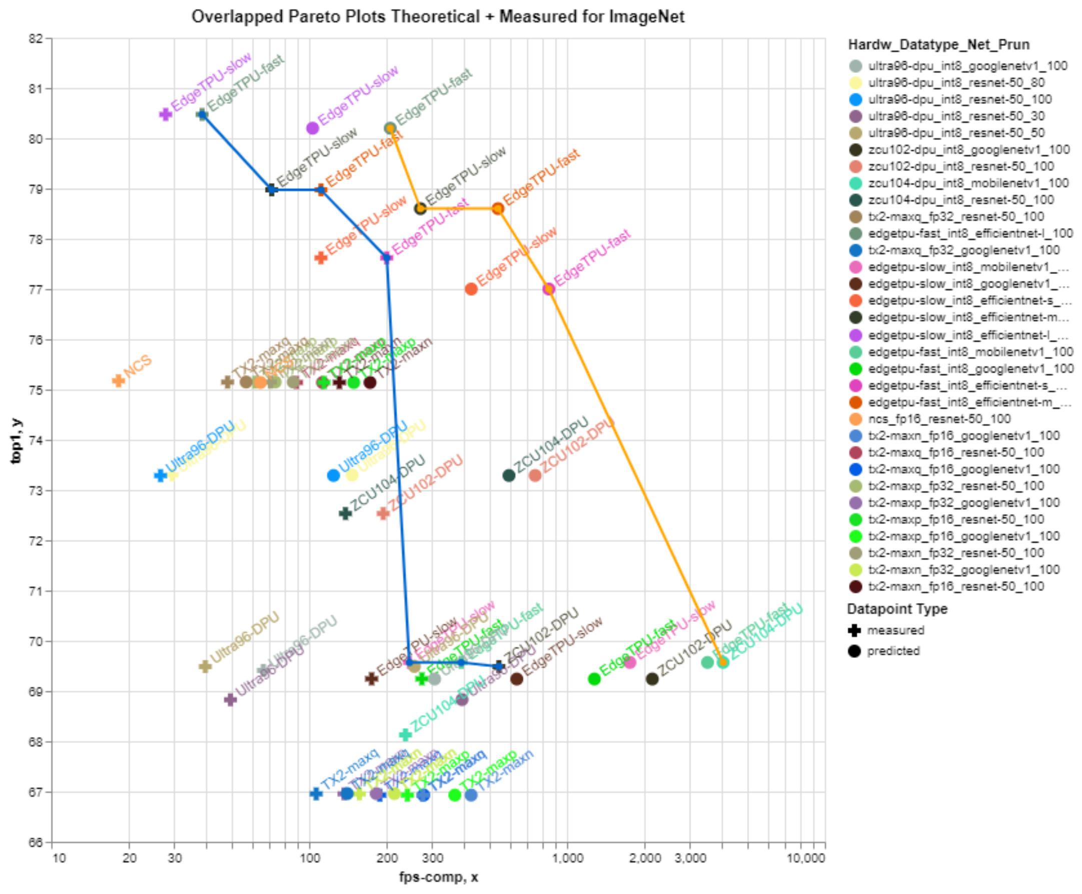


Figure I.7: Overlapped Pareto Plot for ImageNet Classification. Orange pareto frontier shows the theoretical data points and the blue pareto frontier shows the measured data points. Figure available at [5].

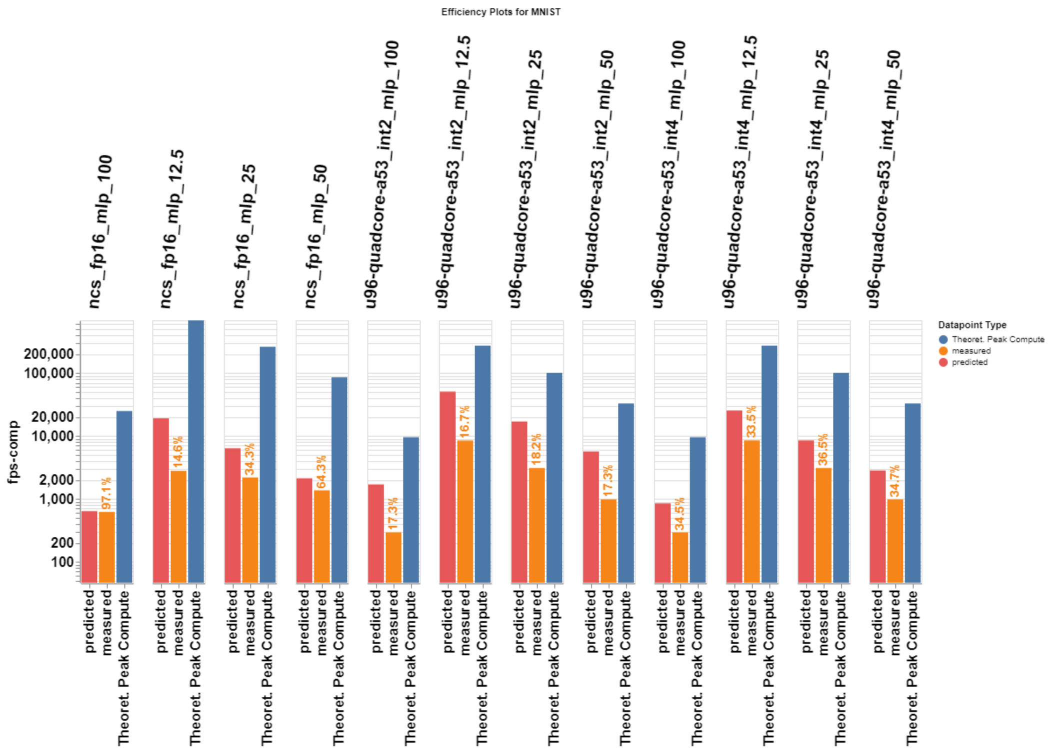


Figure I.8: Bar chart showing Efficiency for MNIST classification over NCS2 and the A53. Figure available at [5].

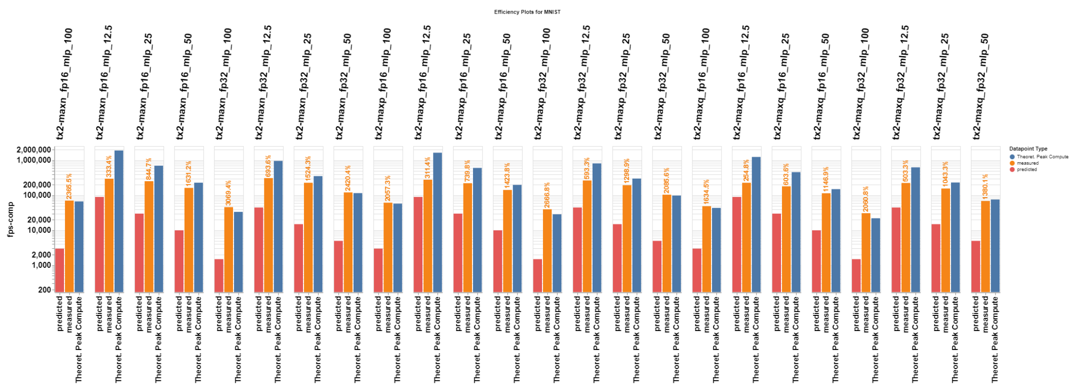


Figure I.9: Bar chart showing Efficiency for MNIST classification over the TX2. Figure available at [5].