Ultra low latency deep neural network inference for gravitational waves interferometer

M.C.B. de Rooij

Ultra low latency deep neural network inference for gravitational waves interferometer

by

Martijn Cornelis Bernardus de Rooij

A THESIS

submitted in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

at the Delft University of Technology,

To be defended publicly on Friday March 5, 2021

Student number:	4731743	
Project duration:	March 2, 2020 – March 5, 2021	
Company Supervisor:	Roel Aaij	Nikhef
Thesis committee:		
Chair:	Dr. ir. Z. Al-Ars,	TU Delft, supervisor
Members:	Dr. R. Aaij	Nikhef
	J. Petri-König	TU Delft, PhD Student, CE Group
	Dr. J.S. Rellermeyer	TU Delft, Distributed Systems group

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Delft University of Technology
Mekelweg 4
2600 GA Delft, The Netherlands

Nikhef (National Institute for Subatomic Physics)
Science Park 105
1098 XG Amsterdam, The Netherlands

TU Delft

Abstract

Low latency Convolutional Neural Network (CNN) inference research is gaining more and more momentum for tasks such as speech and image classifications. This is because CNNs have the ability to surpass human accuracy in classification of images. For improving the measurement setup of gravitational waves, low latency CNNs inference are researched. The CNN needs to process data from images to enable certain automatic controls for the control system. The data of these images need to be processed within 0.1 ms from the moment of taking the image to the control system obtaining the result of a deep neural network.

Hardware acceleration is needed to reduce the execution latency of the network and reach the 0.1 ms requirement. Field-Programmable Gate Arrays (FPGAs) in particular have the ability to provide the needed acceleration. This is because FPGAs have the ability to create highly customised layers of the network and obtain the lowest possible latency. To reduce the design effort and complexity of the machine learning design, Xilinx introduced the FINN (Fast, Scalable Quantized Neural Network Inference on FPGAs) framework. FINN is an end-to-end deep learning framework that generates dataflow-style architectures customised for each network. To establish if FINN can create the required ultra low latency CNN, some of FINN pretrained networks are used. The first neural network investigated is the Tiny Fully Connected (TFC) network. The TFC network is a multilayer perceptron (MLP) for MNIST classification with three fully connected layers. The other network investigated is the convolutional neural network named CNV. CNV is a derivative of the VGG16 topology. The VGG16 topology is used for deep learning image classification problems with multiple convolutional layers.

By using the analysis tools included with FINN, it can be determined if FINN is able to create the required ultra low latency CNN. The TFC network can be parallelised to a total of 5 expected cycles, with 1 expected cycle per layer. One cycle for the input quantization to standalone thresholding, one for the output layer and finally three for the fully connected layers. For the CNV network on the other hand, the initial convolution layer is unable to go below 8196 expected cycles, because of certain bottlenecks with FINN. These bottlenecks occur because of how FINN implements certain layers, moreover because certain layers can simply no longer be parallelised to lower the latency of that layer. To see if the CNV could achieve the latency requirements, a software emulation of the execution of the network has been done. This emulation showcased that by continuously increasing the parallelisation parameters, together with increasing clock frequencies, it is possible to create an ultra low latency pipeline of a CNN. This configuration has 45866 expected total cycles for the network, its expected cycles for the slowest layer is 8196 and needs a minimum frequency of 200 MHz. With those configuration it is possible to create a pipeline that has a latency of lower than 0.1 ms. From the resource analysis of this configuration it has been made clear that currently only the supported Alveo boards of FINN are able to fit this design on the board. This is because of the amount of Lookup Tables (LUTs) this configuration needs.

Preface

This thesis has been my longest project to date. Which already made it a challenge in it by itself. After just one month of working on this project another challenge had hit me and many others with the outbreak of the COVID-19 virus and the measures against the coronavirus. This forced me to work from home which was a bigger challenge than I originally expected and providing me with the lowest motivation of all time. This experience helped me appreciate the people that helped me complete this thesis even more. Luckily with the help of many people I got through this period.

I am writing my master thesis with the Accelerated Big Data Systems (ABS) group at TU Delft. The collaboration partner for this project is the National Institute for Subatomic Physics in Amsterdam, also called Nikhef. The department I did the collaboration with is working on gravitational wave detection. Almost all communication during this project needed to be online, which showed me how difficult it becomes to get everyone involved on the same page. I did learn a lot of how to approach these problems in the future and should be better at them now.

Now after all these months of work on this thesis project, I can finally show what I have accomplished and learned with this thesis. As this project would not have been doable without all the help I have received, I wish to thank all the people whose assistance was a milestone in the completion of this project.

First of all I wish to thank my TU Delft supervisor Zaid Al-Ars for supporting me in doing my master thesis in a company and all the knowledge I have learned from him. Next to him I would like to thank all the other people at the Accelerated Big Data Systems group for helping me further in my thesis and with all the questions I had. Especially Jakoba Petri-König, for her advice and tips throughout the project.

From the Nikhef side I would like to thank first of all Ruud Kluit for making this project for me possible. After which Roel Aaij was my supervisor from Nikhef, pushing me in the right direction. In addition, I also would like to thank Rob Walet for helping me with the project.

Lastly I would like to thank my mother for her continuous support when I was doing my master. Without her and many of my close friends such as Titus and their every day support, this project would not have been possible to complete.

M.C.B. de Rooij
Delft, March 5, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives and motivation

1.1.1 Project context

For a long time people have been aware of gravity. This is where most people will reference Newton's law of universal gravitation from 1687 with its famous story about the apple falling from the tree [1]. Only centuries later Einstein challenged this law with the theory of general relativity in 1916 [2]. In this law Einstein predicted the existence of gravitational waves. Gravitational waves are caused by the movement of mass, which propagate at the speed of light. On September 2015, gravitational waves were detected for the first time, which proved their existence. This was done by the LIGO-VIRGO (Variability of Solar Irradiance and Gravity Oscillations) collaboration with two advanced LIGO detectors [3]. The observatory VIRGO [4] is currently the only gravitational waves detector in Europe which is a collaboration between 6 countries (including the Netherlands). In the Netherlands, Nikhef (National Institute for Subatomic Physics) is part of this collaboration. The VIRGO department at Nikhef works on improving the gravitational wave detector.

The basic technology behind the detectors in the observatories is a Michelson laser interferometer, where the effective arm lengths are increased by Fabry-Pérot resonance cavities. In order for the interferometer to work, free-falling test masses (suspended mirrors) are needed. These mirrors then need to be kept at the correct orientation and position. The mirrors have a longitudinal degree of freedom and an angular degree of freedom. These degrees of freedom need to be fixed in order to conduct measurements. The process of fixing these degrees of freedom is called "locking" and is currently conducted manually.

The error signals in the system are strongly non-linear functions of the mirror positions and can only be linearised in a very small fraction of phase space. To solve this issue many complex tricks and manual actions are involved to enable classical controls of the mirror and keep the system working. This makes having the observatories into working condition a continuous challenge, relying on well trained, highly experienced people. Which indicates, if these people are not available no measurements can be done and no measurements can be done for very long periods of time. Attempts have been made in the past to completely automate the locking of the mirrors, however the experimental results of these attempts proved to be unsuccessful [5].

To solve this problem Nikhef has created an experimental setup for the improvement of the gravitational waves detector. With this setup, they try to reduce the challenge of continuously needing highly experienced people for angular alignment of optical cavities by using machine learning. In Figure 1.1 the experimental setup of the mirrors is displayed. In the upper left part of this figure you see the suspended mirrors that represent the Fabry-Pérot resonance cavities of the experimental setup and beneath it is a more detailed drawing on how the test setup works. The first part of the setup is in regard to the suspended mirrors. A laser will go through these mirrors. Then by having the laser go through a specific setup of mirrors, the near and far field distribution of the laser can be detected by special cameras. From the near and far field distribution the

control system will be able to determine if there is misalignment of the mirrors and adjust them accordingly.

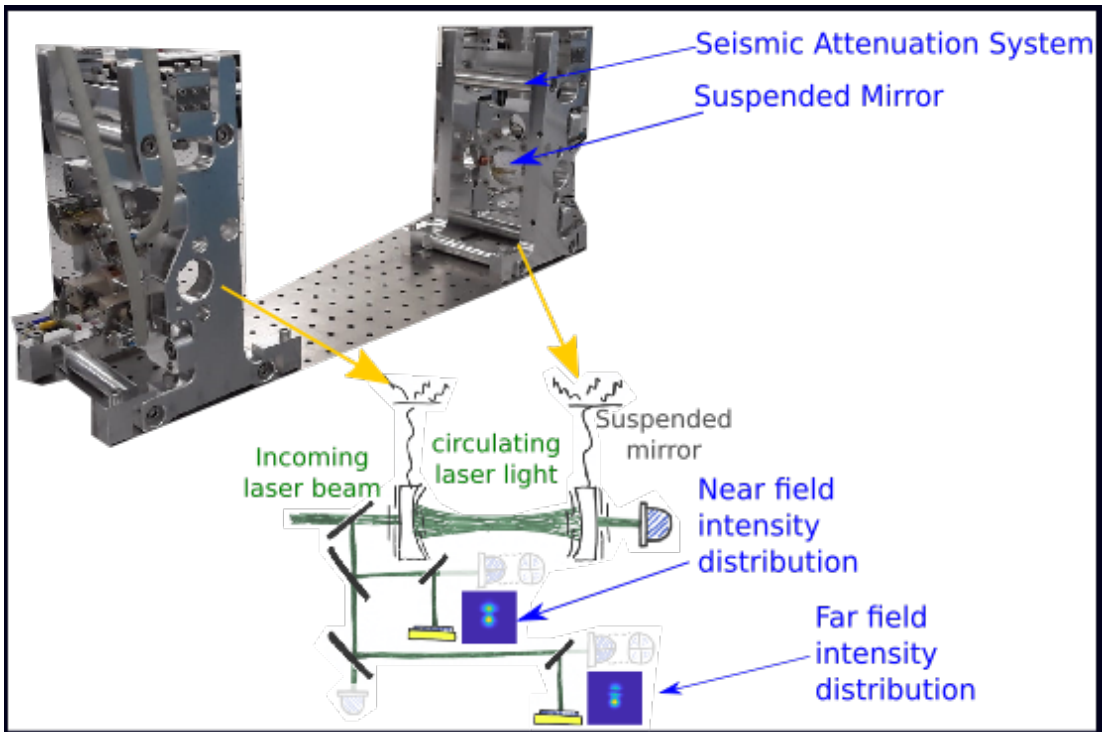


Figure 1.1: Experimental setup suspended mirrors.

The dSPACE MicroLabBox [6] is the control system for this experimental setup which adjusts the mirrors according to the information of the near and far field distributions. This is done to keep the mirrors in the free falling position and not let external factors influence the measurements. The complete experimental setup from Nikhef can be seen in Figure 1.2. The input for this experimental setup are the gravitational waves that can be seen on the left side of the figure. Figure 1.1 shows that within in the setup two images of the near and far field distribution are created. These images go through a CNN and determine the misalignment of the mirrors of the Fabry-Pérot resonance cavities. From there on the control system will readjust the mirrors in the correct position. For a more detailed visualisation of this setup and what kind of devices are involved can be seen in Appendix A.

The moment from taking the images, to determining the misalignment is a tight latency bound part of the control loop of the test setup. These latency requirements are there to ensure that the control system of the suspended mirrors is done with enough speed such that any measurement in this test setup can actually be done. With the current setup, the control loop needs to run at 10 kHz. This is because, in the experimental setup, the longitudinal cavity control of the mirrors is done at this speed by the seismic attenuation control system. In case the control of this part is slower, the experiments for detecting gravitational waves will not work. For this thesis the part to be researched is in regards to the tight latency bound part of this control loop, which is from taking the image to the result of the machine learning neural network.

The exploration of this research is in regards to what hardware platform can run the machine learning algorithm with a low enough latency and high enough throughput. For this multiple hardware fabrics are available such as field-programmable gate arrays (FPGAs), graphical processing units (GPUs) or central processing units (CPUs). This thesis will look at different hardware configurations with their respective frameworks to determine which one is able to meet the low latency requirements.

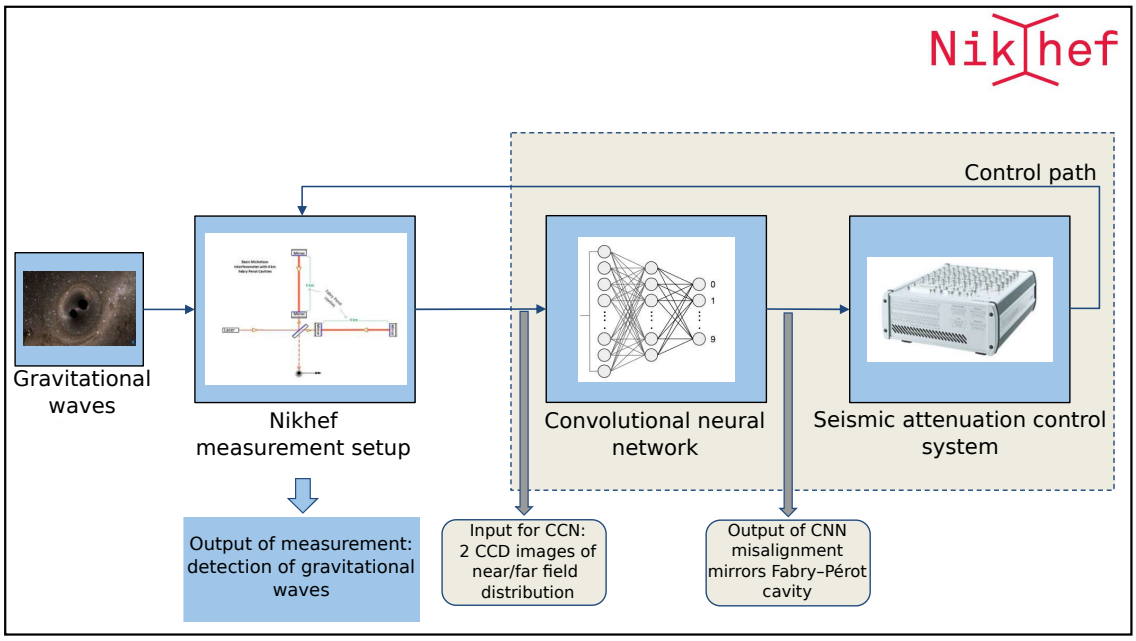


Figure 1.2: Experimental test setup for alignment of mirrors using machine learning.

1.1.2 Research questions

For completing this thesis and exploring the design space of low latency pipelines the following research question needs to be answered.

- How to use the appropriate hardware fabrics to achieve a low latency pipeline of a neural network for a future gravitational waves interferometer control system?

Answering this question requires first an investigation of the current pipeline of the gravitational waves interferometer control system. During the design, development and testing of this system the following questions need to be answered:

1. What is the type of machine learning algorithm running in the pipeline that has to be optimised?
2. How does using different hardware fabrics such as CPU, GPU or FPGAs affect the performance of the pipeline?
3. How effective are available machine learning design tools in reducing design effort and complexity?

1.1.3 Contributions

The main contribution of this thesis is the exploration of ultra low latency deep neural network inference on diverse hardware fabrics, for a future gravitational waves interferometer control system. In order to achieve this contribution, multiple smaller contributions have been made to explore the low latency pipeline. These smaller contributions are the following:

1. **Frameworks**. As the neural network for this pipeline is still changing, multiple frameworks for optimising this network are discussed. For each hardware fabric there are different tools available that have different advantages and disadvantages compared to low latency machine learning networks. The contribution of this thesis is, exploring the different frameworks and by selecting one of the frameworks to evaluate the possible achievable performance.

2. **Low latency**. The second contribution from this thesis is, investigating different configurations the framework can do and how they affect the latency of the neural network. To demonstrate if the low latency is achieved, different performance benchmarks are created. These benchmarks show the difference in latency between the different configurations and possible bottlenecks of the framework.

3. **Neural network**. The neural network for the future gravitational waves interferometer control system is still under development at the moment of writing. For the exploration of the low latency pipeline, different example neural networks with the chosen framework will be investigated. The contribution of this thesis is, determining the possible bottlenecks in the neural network of the pipeline, together with the chosen framework.

4. **Resources**. The final contribution will be some rough resource estimations of the neural network to show if it is possible to create the neural network on actual available hardware.

The behaviour of the framework is studied through testing and benchmarks of different possible configurations. To determine if it would be possible to create a low latency pipeline of a neural network for the future gravitational waves interferometer control system, an extensive evaluation and discussion will be done on the results of these tests. The contributions of this research should make it possible for upcoming researchers to have educated design choices to meet the low latency requirements and the hardware constraints of the gravitational waves interferometer control system.

1.2 Report structure

The thesis is structured in three main parts. The first part includes Chapters 2 and 3 and gives a theoretical background on gravitational waves, machine learning networks, differences between hardware platforms and the design methodology of this thesis. The second part includes Chapter 4 and presents how the different configurations of the pipeline are implemented and what kind of choices where made for them. The final part includes Chapters 5 and 6 and evaluates how the different hardware platforms performed and what conclusions can be drawn from them. To discuss the structure of the thesis in more detail, the information within each chapter will be shortly discussed.

Chapter 2 provides the necessary background for understanding the thesis and the design choices made to meet the performance requirements of this pipeline. To start, the theoretical background of gravitational waves, the Michelson laser interferometer and Fabry-Pérot resonance cavities are investigated and why machine learning is used in the experimental setup of Nikhef. After this concepts and techniques used for running low latency machine learning networks are introduced. For this the Convolutional Neural Network (CNN) is used as this is the neural network to optimize for the low latency pipeline. Finally some potential hardware platforms are discussed such as CPUs, GPUs, FPGAs and ASICs with their respective development tools for low latency machine learning networks.

Chapter 3 discusses the design methodology used in the thesis. It discusses the different hardware constraints the pipeline has and what limitations exists for the pipeline. From these constraints and background research, certain design choices will be made for this thesis and it will be explained why these choices have been made. With finally a discussion about the different neural networks that will be investigated and how they correspond to the possible future neural network for the gravitational waves interferometer control system.

Chapter 4 further explains the considerations and the motivation for the chosen framework to implement the pipeline design. This chapter explains what different design configuration will look like and will discuss how they are implemented. It will further explore how different configurations of the framework will be analysed and how the configurations possibly affect the performance of a complete pipeline design. Finally, this chapter determines the available hardware options of the framework.

Chapter 5 discusses the experimental setup used for the different configurations and what the measurements results are of the designs explored in Chapter 4. For the different configurations it will have a timing and resource analysis to discover possible bottlenecks of the framework and to detect if it is possible to create the low latency pipeline.

Finally, Chapter 6 will give a conclusion to this report and will reflect back on the original research question asked in this chapter. Concluding with a discussion about possible recommendations and possible avenues of future development of the pipeline of the future gravitational waves interferometer control system.

Chapter 2

Background research

2.1 Gravitational waves detection

2.1.1 Gravitational waves

As discussed in Section 1.1.1, Einstein predicted the existence of gravitational waves with the theory of general relativity in 1916 [2]. The gravitational waves are caused by the movement of mass, which propagate at the speed of light. In this section the basic properties of gravitational waves will be discussed as well as the detectable sources of gravitational waves [7]. For a more detailed explanation of gravitational waves refer to the literature [2, 8, 9].

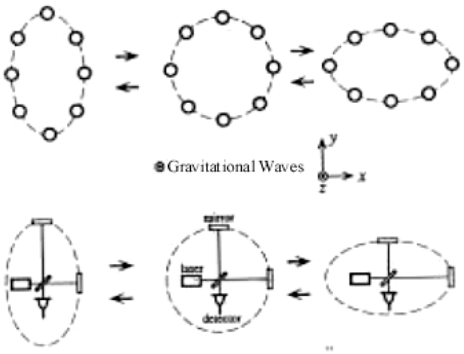
All objects with a mass produce contractions and expansions of space-time along the spacial dimensions perpendicular to the propagation of the mass. The stretching and squeezing of space-time in transverse directions are called gravitational waves. The effect of gravitational waves on matter can be seen in seen in three dimensions [10]. An example of how this works can be seen in Figure 2.1a. This figure shows that for the detection of gravitational waves, a Michelson laser interferometer is being used.

The gravitational waves only create very small disturbances that can be measured. Only some of the strongest sources of gravitational waves can create a big enough disturbance in order to be measured. For example the first measurement in September 2015 a measurement accuracy of gravitational-wave strain 10^{-21} was needed to measure the gravitational waves from two merging black holes [3]. This means that currently it is not possible to measure the gravitational wave of a car driving down the street. Other sources that are detectable originate from supermassive objects millions of light years away. The strongest gravitational waves are produced by binary black holes, supernovae, colliding neutron stars and possibly even the gravitational radiation created by the Big Bang [10, 11]. An example of how the first detection of the binary black holes looks like is shown in Figure 2.1b.

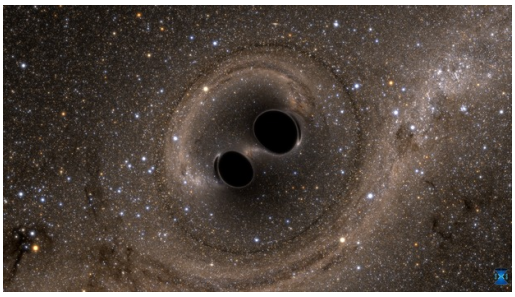
2.1.2 Michelson laser interferometer

From Section 2.1.1 it has been made clear that the Michelson laser interferometers is being used to detect the gravitational waves. The basic technology behind the gravitational waves detectors LIGO and VIRGO is a Michelson laser interferometer [14, 15], where the effective arm lengths are increased by Fabry-Pérot resonance cavities [3, 16]. The current detector is based on the original interferometer concept designed by Michelson and Morley (1887) [17]. The Fabry-Pérot resonance cavities will be further explained in Section 2.1.3.

In order to understand why the machine learning network is necessary and why the pipeline for this neural network needs to be created, a basic understanding of the Michelson laser interferometer is required. The Michelson detector needs free-falling masses to measure gravitational waves. To obtain the free-falling masses, the detector uses mirrors that are suspended by a control system. An example of the Michelson interferometer can be seen in Figure 2.2.



(a) Effect of gravitational waves for polarisation. With respective Michelson interferometer measurement setup below it [12].



(b) Example detection of first black holes by LIGO-VIRGO: GW1509014 [13].

Figure 2.1: Gravitational waves measurement and its first detection.

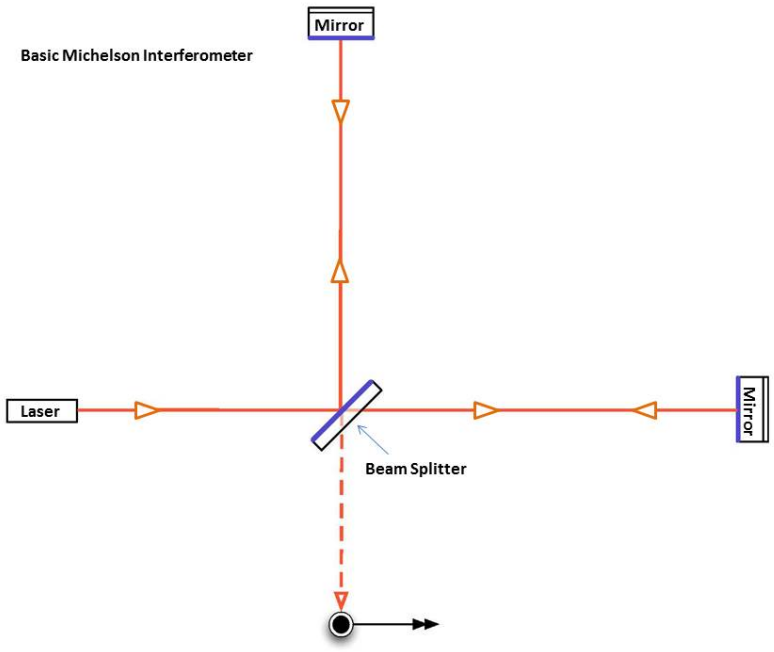


Figure 2.2: Michelson laser interferometer [18].

The Michelson interferometer consists of source of light which is often a laser, which can be seen on the left side in Figure 2.2. This laser beam is split in two using a partially reflecting mirror called a beam splitter. Each of the beams then travel a distance towards two mirrors. These two suspended mirrors are located at a distance from the beam splitter along two orthogonal directions. The distance between the mirrors and the beam splitter are called arms [5]. Once the laser beams reach the mirrors they reflect back to the beam splitter where they interfere. The interference depends on the phase difference between the beams due to the different distance of the arms traversed. At the output of the interferometer a photodiode is placed which converts the interfered light into a current signal.

Considering that the Michelson interferometer on its own does not have a high enough accuracy to measure gravitational waves, Fabry-Pérot cavities are implemented in the interferometer to increase its accuracy. The Michelson interferometer is able to detect gravitational waves by measuring the phase difference between the two laser beams created by the passage of a gravitational wave. This is because gravitational waves change the distance travelled by the laser beam considering the strain created by these waves [19].

2.1.3 Fabry-Pérot resonance cavities

As was mentioned in the Section 2.1.2 Fabry-Pérot resonance cavities are needed to achieve the accuracy required to measure gravitational waves. The cavity is a linear optical resonator which is formed by two partially-transparent mirrors, arranged parallel to each other. From there the Fabry-Pérot cavity is in simple configuration, a standing-wave resonator for the laser beam. If it is on resonance the laser beam will be reflected back and forth inside the cavity increasing the distance travelled of the beam, which enables the accuracy to be high enough for the experiments to work [5, 20]. This is the circulating laser light between the two suspended mirrors as shown in Figure 1.1. With an example of Michelson interferometer with Fabry-Pérot cavities on its arms shown in Figure 2.3.

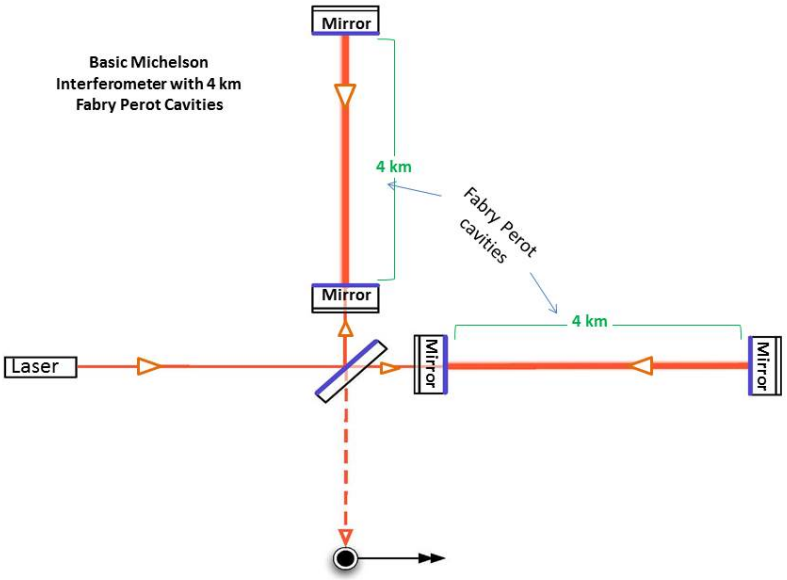


Figure 2.3: Michelson laser interferometer with Fabry-Pérot cavities on its arms used in the LIGO experiment [18].

In a Fabry-Perot cavity it is important that the mirrors are free to move. By allowing the

mirrors to freely move, the mirrors are able to be kept in the necessary free falling position. To show how a Fabry-Pérot cavity is kept at resonance, only a single arm will be looked at as shown in Figure 1.1. The input laser beam and the optical axis need to be aligned to prevent the occurrence of higher order modes (HOMs). The HOMs depend in turn on the shift and tilt of the optical axis as well as the length of the cavity [5]. In case there is a misalignment of the mirrors, error signals will be created. This can occur because of the movement of the suspended mirrors that change the length of the cavity. For that reason longitudinal control and angular control of the cavity is needed to keep the cavity in resonance.

To have longitudinal control of a Fabry-Pérot cavity an error signal is needed. In case of the Fabry-Pérot cavity this is the Pound-Drever-Hall error signal [21, 22]. In gravitational waves detectors the Pound-Drever-Hall error signal is used to control the cavity length of the arm, which has been previously stabilized [5]. The Pound-Drever-Hall technique uses a carrier frequency that is resonant inside the cavity, while the sidebands are anti-resonant, therefore they are reflected. The Pound-Drever-Hall error signal is the beating between the carrier and the sidebands. This is used as phase reference and carries the cavity length information of the Fabry-Pérot cavity.

For the angular control of the cavity, dithering line (mechanical modulation) will be used. This technique provides additional information about the misalignment of the Fabry-Pérot cavities. The Ward technique (Phase modulation) will be used for the global alignment of the suspended mirrors. This technique separates the information about the cavity angular degrees of freedom, tilt and shift. The spatial beam phase distribution is measured here. In other words the near field and far field intensity distributions that can be seen in Figure 1.1. Behind these positions cameras are placed to see if there is a misalignment. Currently as mentioned in Section 1.1.1 highly experienced people are needed to enable classical controls of the mirror and keep the system working. For that reason instead of using people, machine learning will be used to find the misalignment of the mirrors and control the system.

2.2 Convolutional neural network

An important part of the pipeline is the machine learning neural network to be used in it. This is also not the first time a neural network is being used or investigated to be used for the gravitational waves detectors. One application where the use of a neural network was being investigated was to filter noise to increase the accuracy of the detector [23–25]. To understand the choices made regarding the network in the pipeline, first some background knowledge will be presented regarding neural networks.

A majority of neural networks take their inspiration from the brain [26]. This is because just like any brain, a neural network consists of elements that are called artificial neurons, nodes, synapses or edges. A neural network can be split up into two parts. The first part of the neural network is the training or learning part. During this part the neural network uses data to train its network to create a trained model. This model then can be used in the second part, the inference of the neural network. During the inference part, the neural network applies the data from the trained neural network model on the input and then uses it to infer a result. This research will mostly focus on the second part of the neural network, the inference part. The design target of the neural network is to achieve low latency with an high enough throughput to meet the control constraints of the entire system. For a neural network the most important attributes are accuracy/ robustness, power/energy consumption, throughput/latency and cost [27]. All these metrics together show the trade-offs made in the design of a machine learning network and will be further discussed in Section 3.1.

Machine learning networks have applications in various fields from computer vision [28] towards natural language processing [29,30]. The most promising approach for image based neural networks are Convolutional Neural Networks (CNNs). Some state-of-the-art CNNs can already rival or even surpass the accuracy of humans when it comes to the classification of images [31]. The following subsections will give an overview of neural networks and in particular convolutional neural networks, but a full introduction to machine learning, neural networks and deep learning

is beyond the scope of this thesis. The reader can refer to the following references: [26, 27, 32–34].

2.2.1 Neural networks

For a neural network the basic element is an artificial neuron or node. The neuron consists of units that sums all its input signals x_i, which all are separately weighted w_i. The input signals come from other neurons. The weighted input signals are all summed up. These are then biased with a fixed constant w_b and fed into a non-linear activation function. This will then produce the neurons output signal as can be seen in Equation (2.1) [35]. Examples of non-linear activation functions are the sigmoid or the ReLU function, which can be seen in Equations (2.2) and (2.3).

$$y = w_b + \sum_{i=1}^{N} w_i * x_i \tag{2.1}$$

$$sigmoid(x) = \frac{1}{1 - e^{-x}} \tag{2.2}$$

$$ReLU(x) = max(0, x) \tag{2.3}$$

The mathematical model of a single neuron can be seen in Figure 2.4. The weights in the neuron model can be adjusted such that the neural network gives the desired output. Each of these neurons can be arranged into different layers. A neural network is created by connecting the neurons between the different layers.

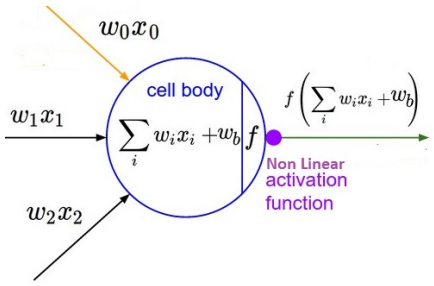


Figure 2.4: Single neuron mathematical model [35].

The neural networks are called N-layer neural networks, which do not count the input layer. Depending how large N is they are sometimes called deep neural networks. An example of a neural network is visible in Figure 2.5. This is a 2-layer neural network. Here one layer is the hidden layer, which are the layers between the input layer and the output layer. The last layer is the output layer which determines the outcomes of the neural network. These networks are then trained to determine the weights, which are needed to achieve an as high as possible accuracy of the network.

For neural network training the parameters or weights are not manually chosen but they are learned during the training phase of the neural network. For the training of the neural network there are many frameworks available that enable to train a custom network with little engineering effort. These frameworks depend on which hardware platform is being used for the training. Some examples of these frameworks are the Deep Learning Toolbox from MATLAB [36], Caffe [37], Tensorflow [38], Theano [39], Keras [40] and PyTorch [41]. The framework used for training the network needs to be compatible with the framework used for the neural network. The different hardware platforms together with the different frameworks will be further discussed in Section 2.3.

The training method often used is called supervised learning. This requires labeled training examples that the network can learn from. The training of the network starts with small, random initialised weights. Then one by one all training examples are given to the network. The results of the network are compared to a ground truth label using a loss function. This measures how

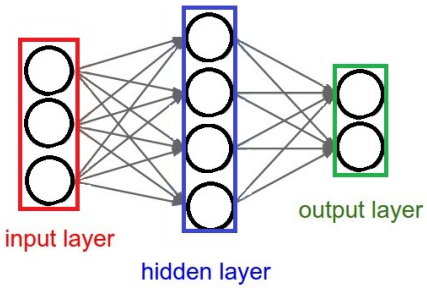


Figure 2.5: 2-layer neural network (one hidden layer of 4 neurons and one output layer with 2 neurons), and three inputs [35].

much the output deviates from the expected result. The training process then tries to minimise the loss function (error) on the training by changing the weights in the network.

Each weight in the network has a certain influence on the network. The stochastic gradient descent is an iterative method that describes each weights influence on the network. By using a backward pass the gradients of the network can be calculated by using the output error of the network. Then by continuously looping over the training examples the current loss function is determined and from that the gradient vector adjusts all the weights in the opposite direction of their respective gradient. One such loop is called an epoch and the change per update is the learning rate. This learning rate is typically large at the start and made smaller over time [35].

Network validation needs to be done before the network inference can be used. The network validation is done during the training of the network. This is done every few epochs with validation examples that are not used in the training examples. Once the accuracy is deemed good enough compared to the validation of the network, it can be used for neural network inference.

2.2.2 Introduction to convolutional neural networks

Convolutional neural networks is one of the most promising approaches for image based neural networks. From Section 1.1.1 it is made clear that the neural network of this system uses images from special cameras to determine the controls the system needs to take. A typical CNN is a feed-forward network that consists of a number of layers that run in sequence. The input and output of each layer consists of so called "feature maps". Within a layer the neural network can run in parallel to apply the weights determined in training of the neural network [26]. The first layer (input layer) of the neural network takes the dimensions of the image as input and produces the output feature map for the next layer of the CNN. With a CNN there is no need for a fully connected layers for each layer, because of the locality of information in images. In a typical CNN a single pixel cannot determine the output but the local neighbouring relations of pixels can. These layers that replace the fully connected layers are so called convolutional layers. Next to the convolutional layer (CONV layer), a typical CNN consists of multiple other layers such as non-linearity, pooling, input and fully connected layers [27,42–46]. In the case of a sequential fully connected neural network, billions of extra weights need to be stored.

The input layer will obtain the raw input data of the image. This is the width and height of the images and the amount of channels for each image. For example for a RGB image which has 3 channels with a width of 32 and a height of 32 the input feature map would be $[32 \times 32 \times 3]$.

The CONV layer consists of learnable filters of size (k × k). Commonly used filters are 1×1, 3×3, 5×5, 7×7. These are often called convolution kernels. The amount of kernels or filters used determines the depth of the output volume. For example if the input layer uses 64 filters with kernel sizes of 3×3, the output feature map would be $[30 \times 30 \times 64]$. This is the first

hyperparameter that can be tuned in a CONV layer of a CNN. An hyperparameter is a value that is set before training begins and a value that controls the learning process of the network.

For kernels larger than 1×1 the output feature map dimensions will be reduced. To avoid reducing the output dimensions, sometimes the input feature maps will be padded with zeros to preserve the spatial size of the input volume [43]. The zero-padding is the second hyperparameter.

The final parameter of the CONV layer is the stride used. If the stride is 1 then the filters are used for one pixel at a time, but when the stride is 2 or larger then the filters jump 2 or more pixels at a time. This will result in a smaller output feature map.

The non-linearity function applies the activation such as Equations (2.2) and (2.3) to the output of a layer. This function leaves the size of the feature maps unchanged. This function is sometimes called the activation in some literature [26]

The pooling layer reduces the spatial dimensions of the input feature map. It does this by summarising multiple input pixels into a single output pixel. The two most used pooling methods are max-pooling and avg-pooling. With max-pooling the output is determined by the taking the maximum of value of the pixels and for avg-pooling the average is taken. An example of max-pooling is where the filter of size $[2 \times 2]$ is applied with a stride of 2. This would mean that for 4 input pixels only a single pixel would remain making the output feature map of this layer $[15 \times 15 \times 64]$ as the depth once again does not change. An extreme case of pooling would be global pooling to the whole input image. This would reduce the spatial information to 1×1 pixels.

The Fully-Connected (FC) layer is often used as the last layer in a CNN. These layers usually have the most weights in CNNs, because they try to reduce the spatial information of the network to a single output. This means that each input of this layer needs a weight to know how it matches to each output. The FC layer is often the most memory intensive layer of the network.

2.2.3 CNNs complexity

CNNs have the ability to surpass human accuracy in classification of images [31]. This excellent performance of CNNs comes from the cost of a large computational complexity and storage capacity is high [26, 45–56]. The inference of a CNN may need up to billions or trillions of operations per second to achieve the required latency of the network. An example CNN model that uses this many operations uses 224×224 images. This model requires up to 39 billion floating point operations (FLOPs) and more than 500 MB of model parameters [57]. Another example is for a VGG-19 network. This network needs over 15 billion FLOPs to classify only a single image [58]. A final example is AlexNet, this contains 60 million parameters and storage size of 240 MB when the weights are stored as 32-bit numbers [59]. These large storage sizes and computational complexity can lead to problems in the deployment of CNN inference for low latency systems [60]. To achieve the lowest possible latency, several methods have been proposed to reduce the computational complexity and storage capacity of CNNs. Important to notice is that to achieve low latency is different from achieving high throughput in CNNs. Latency refers to the time taken to process a single input for the neural network, while throughput of a neural network is defined as the maximum number of inputs the network can process in a given timeframe. This means to achieve low latency pipelines the batch size is often small, while for high throughput the batch size is larger [61].

The methods to achieve the low latency include algorithmic optimizations for CNNs, acceleration or compression techniques which reduce the size of the weights used and the computational complexity. Researches then use these techniques to design more efficient networks models [26,45,52]. Examples of such networks are ShuffleNet [62,63], ShuffleNet V2 [64], Shift-Net [65], MobileNetV2 [66], AddressNet [67]. MobileNet is for example created by using depthwise separable convolutions [55].

2.2.4 CNNs complexity reduction techniques

Several methods have been proposed to reduce computational complexity and storage capacity of CNNs. Methods that are going to be discussed include pruning, reduced precision CNNs (quantization) and binary CNNs which are often called BNNs or BCNNs [47, 68–70].

Pruning is a technique used in neural networks where all small or unimportant weights are set to zero. This removes the unimportant connections, computational complexity and storage capacity [45, 49, 71, 72].

Quantization is being used to convert the usual 32-bit floating point precision numbers of weights and activations to low-precision numbers to reduce the computational complexity [52, 54, 72]. This also reduces the storage space needed for all the weights. Quantization can be split up into two different methods. Linear quantization and non-linear quantization. Where linear quantization tries to find the nearest fixed-point representation for each weight and activation. Non-linear quantization independently assigns values to different weights and activation's. The problem with using quantization is that it can lead to significant accuracy loss of the neural network. This is because, once the quantization converts the 32-bit floating point numbers to 8-bit fixed point numbers, the dynamic range of the weights and activations will be lower. Which in turn means less information can be shared between each layer in the network [49]. In quantized neural networks sometimes a thresholding layer is introduced at the input of the network. This layer converts the input quantization to standalone thresholding to ensure the neural network can use the input in the rest of the neural network. In this layer the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value. For more extensive introduction of quantized neural networks the reader can refer to the paper of Umuroglu [73] or Choukroun [74]. To further optimize the storage reduction techniques, networks such as AddNet have been develepod. AddNet is special in that it only uses adders, subtractors, bit shifts, and multiplexers. These are components that can be highly optimized for FPGA designs to ensure efficient utilisation [69].

BNNs or Binarised neural networks are a special kind of CNN. With a BNN instead of using 32-bit floating point precision for each layer for the weights and activation's, they are binarized to either +1 and -1 or 1 and 0 and stored in a single bit [45, 48, 75–78]. The BNNs that use the +1 and -1 are often called bipolar neural networks instead of binarized. BNNs have been proposed to challenge the two disadvantages of CNNs. First like quantization the storage for the weights and activations is reduced by using only a single bit instead of 32 bits. Secondly the BNNs also change multiply-accumulates (MACs) CNNs use to XNORs. The XNORs further reduce necessary compute sources [51]. A BNN does not always do this for every layer of a CNN. For the first and last layer a BNN sometimes still uses 32-bit floating point precision in order to achieve acceptable accuracy [79]. BNN type networks also introduce a new layer type. This is the batch normalization layer. This layer can be used instead of the non-linearity layer of the CNN. This layer reduces the information that is lost during binarization of the 32-bit floating integers by shifting and scaling the input distribution to have zero mean and unit variance [45, 48, 80].

2.3 Potential hardware platforms

The system that has to be developed has certain constraints such as the low latency that has to be met for the control system to work. These constraints can be met by many different means. This section will explore a number of different options that are available for the implementation of the neural network. Several different hardware platforms will be investigated such as CPU, GPU, FPGA and ASICs. For each of these platforms, it will be determined what their most common use is, while also looking at the development time on these platforms. This development time can

increase and decrease depending on which tools are available for these platforms which will all be discussed.

2.3.1 Central processing unit

Central Processing Units (CPUs) are general purpose processors used in a very large range of devices such as desktop computers and smartphones. CPUs have a high flexibility and can be used for comparisons of different complex processes in a program. With many CPUs having optimised prediction optimisation to increase speed in processes. There are many types of CPUs available where they are designed with different trade-offs such as power and speed in consideration.

The reason why CPUs are often not used for low latency CNNs, is because they often compute the results sequentially, while a CNN is highly parallel in each layer. However CPUs can have multiple cores available with as example AMDs latest Ryzen Threadripper 3990X Processor which has up to 64 cores [81] making it possible for a certain level of parallelism. Nevertheless the NVIDIA GeForce RTX 2080 TI (GPU) has up to 4350 floating-point processing cores running. CPUs are highly flexible and can be used for many different use cases in a design. They have the ability to support for large storage capacities and sizeable amount of memory compared to GPUs or FPGAs. They do fail to match them on the raw compute capabilities. For that reason CPUs are often used in heterogeneous systems together with GPU or FPGAs for the inference of the system [63, 82]. The CPU has many tools available for creating a neural network inference. Each of these tools use some form of high-level programming such as C, C++ or Python. These frameworks are for example the Deep Learning Toolbox from MATLAB [36], Caffe [37], Tensorflow [38], Theano [39], Keras [40] and PyTorch [41]. The availability of all these different tools make the development of the neural network inference on a CPU one of the easiest of all hardware platforms.

2.3.2 Graphics processing units

The Graphics Processing Unit (GPU) has the ability to outperform the CPU for the inference of a neural network by using their raw compute capabilities. By using the GPU as accelerator for training a CNN model it can reduce the computational time needed compared to just using a CPU [26,83,84]. GPUs are being more and more used for general purpose computing tasks that are highly parallel [85–87]. For that reason these GPUs are referred as General-Purpose Computing on GPUs (GPGPUs) [45]. These computing task can then be used for neural network training and inference. High end GPUs such as the NVIDIA GeForce RTX 2080 TI [88] can have up to 4350 floating-point processing cores running at the same time with a clock of 1350 MHz. This GPU has 616 GB/s memory bandwidth, which lead to that the GPU can compute up to 11750 GFLOP/s. The costs of this performance of GPUs usually comes from power usage as the NVIDIA GeForce RTX 2080 TI uses up to 250W. State of the art GPUs have the ability to process the necessary billions or trillions operations per second, which may be necessary for a low inference CNN. GPUs are designed to be well suited for parallel workloads. Which means that the they are a good fit for the highly parallel workloads presented by CNNs. Some CNNs are even structured to be optimal for GPU efficiency [52]. The GPU constitutes one of the primary platforms for research in the area of CNNs because of how they are being structured and because GPUs are fully supported by most deep learning frameworks. Many GPU vendors also aggressively position the GPU as the compute platform of choice for machine learning by modifying their architecture [89].

The GPU consists of large arrays of arithmetic logic units (ALUs) or cores. There can be up to hundreds or thousands smaller cores [88] in a GPU while a CPU for example only has 8 cores. With an outlier being the Ryzen Threadripper 3990X Processor which has up to 64 cores [81]. By having this many cores the GPUs are generally suited for high-throughput computations that can exploit the singe instruction multiple data (SIMD) architecture of a GPU [54]. For the GPU the workloads would be broken up into thousands of parallel threads. Preferably as many as the GPU has available. This is because each thread needs to perform the same instruction at the same time. This means that for certain instructions many threads will be idle, resulting in a reduced

compute efficiency and making the top performance of a GPU dependent on how many threads can efficiently be used at the same time.

The GPU typically uses single-precision floating-points for each thread and, on occasion, double-precision floating-points. This double-precision floating-point often comes with a significant performance penalty compared to single precision floating-point calculations. For example for the NVIDIA GeForce RTX 2080 TI the single-precision floating-point has processing power up to 11750 GFLOP/s, while the double precision only has processing power of 367 GFLOP/s. The GPU does not always support the reduced-data-type precision techniques mentioned in Section 2.2.4. These techniques can however be used with FPGAs where it is easier to provide a lower precision data type [89].

The GPU system architecture has its own memory architecture which affects how the inference of the machine learning network will be on a GPU. An example of the GPUs memory hierarchy is visible in Figure 2.6. Each of the threads in the GPU has their own local memory and registers. Then a certain amount of threads will fit into a block. In this block the threads can communicate by using the shared memory. This memory is slower than the registers but faster than the local memory each thread has. Finally in a grid there is the global, constant and texture memory. Each of these memory spaces are optimised for different memory usages. The local memory of a thread is as slow as the use of global memory and will only be used if the register cannot fit all the necessary data. These differences in memory means that depending on which memory in the GPU is used, the latency can increase up to 100 times [90].

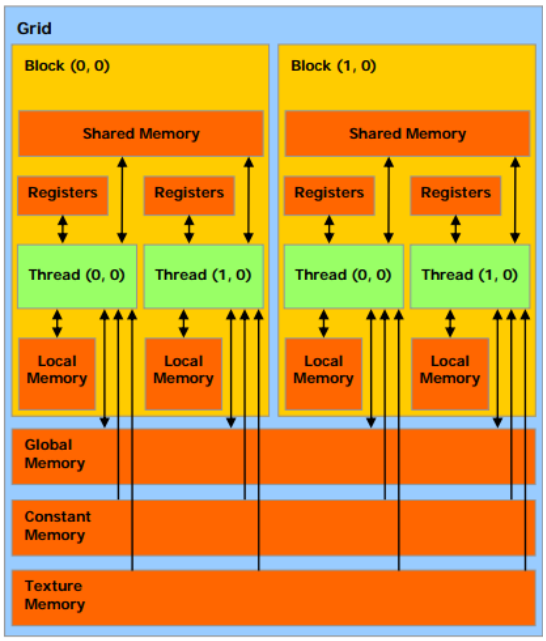


Figure 2.6: GPU memory model [91].

For a GPU, a CPU or host is still required to allocate workloads to the GPU [92]. This introduces one of the biggest bottlenecks of the GPU as the transfer of data to the GPU is dependent typically on the speed of the PCIe lanes [82]. For example the NVIDIA GeForce RTX 2080 TI supports PCIe 3.0 x16 limiting the GPU performance by only having 15.75GB/s of bandwidth. This bandwidth is 39.11 times slower than the memory bandwidth on the GPU card itself, making it one of the biggest bottlenecks for low latency driven systems. Considering the bottlenecks the GPU has it is more used for training than inference. As it delivers high performance compared to other hardware for batch computations but cannot meet the tight latency constraints for frame by frame processing [46, 84, 93, 94].

Just like the tools available for neural network inference on CPU there are several tools available

for the GPU. For this many of the same frameworks can be used such as Caffe [37] and Tensorflow [38]. For the general computing on graphical processing units (GPUs) the frameworks such as OpenCL and CUDA [91] have been developed to dramatically speed up computing applications [89]. By using these frameworks together with for example Caffe, the GPU can be utilised better for inference of the neural network.

2.3.3 Field-programmable gate arrays

The Field-Programmable Gate Array (FPGA) is another promising option as hardware platform for the inference of the CNN [53,72,95]. As accelerators FPGAs are commonly used to stream data from input devices and immediately perform computations on the streamed data. Examples of these computations are decompression [96], image processing [97], scientific computation [98] and simulation [99]. FPGAs have the ability to provide thousands of programmable logic blocks and configurable interconnect. This enables the possibility of a custom-tailored accelerator architecture in hardware that can achieve the necessary low latency [89]. For this FPGAs can use on-chip SRAM (Block RAM), USB, PCIe and Ethernet Transceivers, Digital Signal Processor (DSP) Slices, PLLs, Memory Interfaces and sometimes even full ARM processor cores [42,46]. The logic blocks, functions units and interconnects created for this device are programmed electronically by writing a configuration bitstream into the device. For that reason FPGAs are thought of as hardware devices where they are programmed in Hardware Description Languages (HDLs) such as VHDL or Verilog. This configuration that is typically held in the SRAM of the FPGA however can be be reprogrammed many times, giving it a advantage over dedicated circuitry [100]. The designs of the FPGAs are then described at Register Transfer Level (RTL). This is where the parallel processes which operate all the binary signals and simple data types are described. All these processes combined describe the basic arithmetic operations, logic blocks and registers. These processes are then driven by the rising and falling edges of a clock signal. For this FPGAs can have clock speeds up to 500 MHz [100]. This slow clock speed is an inherent disadvantage of FPGAs compared to GPUs which can have almost up to three times its clock speed [49]. For example the NVIDIA GeForce RTX 2080 TI can have a clock speed of 1350 MHz [88].

A design that can be made parallel by building custom processing engines using the programmable logic blocks is the best option for the FPGA. While workloads and algorithms that require complex data-dependent branching and decisions are more suited for the CPU instead of FPGAs. The FPGA can unlike the GPU increase performance by using the techniques mentioned in Section 2.2.3. This is because FPGAs have more flexibility in which data type precision will be used [89].

Like the GPU and CPU, the FPGA has its own unique memory which can be used. An example in Figure 2.7 from [89] tries to show how the memory hierarchy looks like for a Xilinx FPGA. This shows that kernels or programmable logic blocks can interface directly to the memory such as the LUTRAM, BRAM, UltraRAM or even external memory.

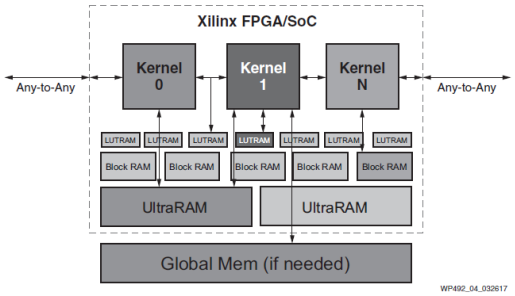


Figure 2.7: FPGAs programmable datapath memory hierarchy example [89].

This memory is an important factor to store the model parameters of the CNNs. As mentioned in Section 2.2.3 this can go up to 500 MB. For this the FPGA can use the UltraRAM that some

FPGAs have available. This memory can contain up to hundreds of megabits on-chip storage, storing all the model parameters for faster latency of a system [50, 89].

There have been many optimisations made to accelerate CNNs on FPGAs. To show some of the approaches to CNNs on FPGAs see Figure 2.8. This figure is based on figure 2 of the paper from Abdelouahab [54].

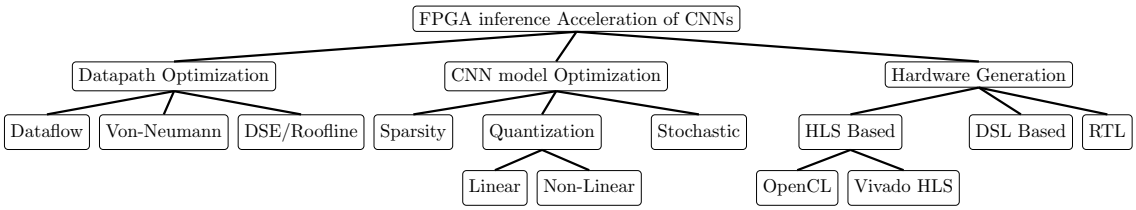


Figure 2.8: Main approaches to accelerate CNN inference on FPGAs [26, 54]

One important factor as to why FPGAs are not widely used for machine learning networks is because of their development time compared to GPUs or CPUs. As mentioned before the designs of the FPGAs are described at RTL. While RTL descriptions are very close to the logic gates and RTL synthesis can be closely controlled, the process of creating the necessary processes in logic blocks is a very tedious and error-prone process. Making the engineering effort of a CNN on FPGAs higher than for example a CPU or a GPU [26, 52, 101].

In order to solve this problem multiple vendors have created frameworks and libraries that target to lower the development time of neural networks on FPGAs. The paper by Shawahna, [46] discusses many cases of FPGA implementations of CNNs. Here the paper discusses different frameworks and techniques used for development of the CNNs on FPGAs. As there have been many efforts by the academic community for the development on FPGA-based CNN accelerators [56, 102, 103]. One of these developments is the use of low precision CNNs like BNNs mentioned in Section 2.2.4.

There are many CNNs to FPGA frameworks used for reducing design effort and complexity. One of them is for example the Merlin Compiler of Falcom computing [104]. This framework requires the rewriting of the algorithms in OpenCL or lower level hardware description languages. Additionally generic frameworks such as Intel OpenCL [105] and Xilinx SDAccel [106] allow computational kernel to be offloaded from host processor onto FPGA-based accelerators. OpenCL is a higher-level programming language that can be used between different hardware platforms while hiding which hardware is going to be used [82].

Both Xilinx and Intel mention that these frameworks have acceleration for deep learning algorithms such as CNNs as major use case, but both do not directly accelerate CNNs. Vivado HLS is a framework that can be used to implement accelerators. For example the paper by Liu, [63] uses Vivado HLS (v2016.4) to implement the neural network in C++ and convert it to RTL as a Vivado Intellectual Property (IP) core. There have been many more automated frameworks in development that map CNNs to FPGAs. Table 2.1 lists some of the CNN-to-FPGA frameworks. The table also mentions some of the supported FPGAs that can be used with the framework. Depending on the framework used, an optimised FPGA-based CNN accelerator can be generated. Most of these frameworks are discussed in papers such as: [46, 56, 107]. The majority of the frameworks mentioned in the table use a high-level description of the network in Caffe or Tensorflow and generate an accelerator for an FPGA. The architectures of these frameworks can for the most part be categorised into streaming architectures and single computation engines [56].

The streaming architecture generates feed-forward dataflow-style architectures that will be optimised for each network. In this architecture each layer of the neural network will be implemented separately to exploit the parallelism of these layers. These layers together will create a pipeline that enables the data to be streamed through the architecture. The advantage of this architecture is that it exploits the parallelism between layers by means of pipelining, which lowers the latency by reducing the communication time between layers. Its disadvantage is the long compilation times that are needed. This comes from that for each configuration of the CNN a new bitstream

needs to be created. Some of the frameworks that use the streaming architecture are fpgaConvNet, DeepBurning, FINN and Haddoc2 [56].

The single computation engines consists of a matrix of processing engines. This architecture is made up of a single computation engine, that is often in the form of a systolic array of processing elements or a matrix multiplication unit [56]. This unit will then execute each of the layers of the neural network sequentially, increasing the latency of network compared to streaming architectures. The advantage of this architecture is that is often more flexible reducing the time to create multiple neural networks. The frameworks that use this architecture are Angel-Eye, ALAMO, DnnWeaver, Caffeine, FP-DNN, Snowflake, SysArrayAccel and FFTCodeGen [56].

Framework	**Interface**	**Supported FPGAs**
ALAMO [95, 108–111]	Caffe	Intels: Stratix V GXA7 and the Arria 10 GX115 SoC
Angel-Eye [103, 112, 113]	Caffe	Xilinx SoCs: Zynq XC7Z045 Zynq XC7Z020
AutoCodeGen [114]	Proprietary Input Format	Xilinx: -
Caffeine [44]	Caffe	Xilinx: KU060 FPGA board Xilinx: Virtex 7 VX690T
DeepBurning [115]	Caffe	Xilinx SoCs: Zynq XC7Z045 Zynq XC7Z020
DNNWEAVER [116, 117]	Caffe	Intels: Stratix V GSD5 and the Arria 10 GX115 SoC Xilinx Zynq XC7Z020 SoC
FFTCodeGen [118–121]	Proprietary Input Format	Intel: Stratix V GXA7, With Intel Xeon E5-2600 v2 CPU
FINN [80, 122–125]	(Theano past), Brevitas	Xilinx SoCs: Zynq XC7Z045 Zynq XC7Z020 Pynq-Z1, Pynq-Z2, Ultra96, ZCU104 Alveo Boards
fpgaConvNet [61, 126–128]	Caffe & Torch	Xilinx SoCs: Zynq XC7Z045 Zynq XC7Z020
FP-DNN [101]	Tensorflow	Intel: Stratix-V GSMD5
Haddoc2 [129]	Caffe	Intel Cyclone V FPGA Xilinx Kintex 7 FPGA
Snowflake [130, 131]	Torch	Xilinx SoCs: Zynq XC7Z045
SysArrayAccel [132]	C program	Intel Arria 10 GT115
TABLA [133]	Template-Based	a Xilinx Zynq FPGA
Vitis AI [134, 135]	TensorFlow and Caffe	Xilinx: ZCU102 ZCU102 Alveo: U50 Alveo: U200 Alveo: U250 Ultra96

Table 2.1: CNN to FPGA inference frameworks.

From the frameworks mentioned in Table 2.1 some will be shortly further discussed that have an open-source version. These frameworks to be discussed include DNNWEAVER [116], Haddoc2 [129], FINN [125] and Vitis AI [135]. Only the open-source version will be discussed, considering the neural network has yet to be created for the pipeline. Meaning that in the future maybe some specific adjustments must be made to the tool in order for the framework to work.

DNNWEAVER is a framework for accelerating Deep Neural Networks (DNNs) on FPGAs. The programmer that uses DNNWEAVER specifies the DNN using Caffe. From the Caffe format of the DNN the framework automatically generates the accelerator Verilog code. It does this by using special hand-optimized Verilog templates.

Haddoc2 is an framework for accelerating Convolutional Neural Networks (CNNs) on FPGAs. Just as with DNNWEAVER, Haddoc2 uses a Caffe model to generate a hardware description of the network. This hardware description is in VHDL-2008 and the framework claims that the generated code is constructor and device independent. Haddoc2 implements the target CNN by directly mapping all the actors involved in CNN processing are physically mapped on the FPGA.

FINN is a framework developed by Xilinx. The FINN framework explores deep neural networks inference on FPGAs. For this FINN specifically targets quantized neural networks (QNN). By focusing on QNNs it is able to trade-off between very high throughput and low latency networks. For this the developer needs to use Brevitas trained nework. Brevitas is a Pytorch library made by Xilinx for quantization-aware training

Vitis AI is another framework that is being developed by Xilinx. For further accelerating productivity for hardware designers Xilinx released the Vitis Unified Software Platform [134,135]. Vitis AI is an integral part of Vitis. This platform enables the development environment of accelerating neural network inference on Xilinx embedded platforms. The Vitis libraries enable the accelerating with minimal changes to existing code written in C, C++ or Python by using frameworks such as Tensorflow and Caffe. It further consists of tools to optimise networks. Tools that supports model quantization, calibration, and fine tuning.

Vitis AI and FINN are currently the most promising frameworks to look further into. For example a network build with FINN was able to reach a latency off 0.31 μs and 12.3 million images per second [80]. Another reason is that both Vitis AI and FINN are both frameworks that at the moment of writing are still actively being developed.

2.3.4 Application-specific integrated circuits

Application-Specific Integrated Circuits (ASICs) are custom-tailored semiconductor devices. Generally ASICs do not suffer from any area or timing overhead compared to the other hardware platforms discussed. This results in that ASICs typically are the smallest and fastest systems for inference of neural networks [53, 93].

However next to the advantages of ASICs designs they also have several disadvantages compared to other hardware platforms. The most important reason is that the development of an ASIC design is a long and complex one [53, 93]. This is due to the fact that ASIC designs do not use any form of high-level languages. The ASIC design also would have the most difficult time communicating with the other components of the system as those parts also needs to be specifically designed for this project, making the design time even longer.

An example of an ASIC design to accelerate deep neural network inference tasks is the Google's Tensor Processing Unit (TPU) [94]. The TPU of Google has the ability to support various deep neural networks by having a integration with the Tensorflow framework. However the development of this ASIC is still a long one and cannot be easily reconfigured. Another example of the ASIC design flow could go for a CNN on an ASIC is discussed in the paper [93] by Boutros. This paper uses Synopsys Design Compiler 2013.03 to synthesize computing architectures for CNNs using 28nm STMicroelectronics standard-cell libraries.

2.3.5 Comparison and combination of different platforms

In order to find out which hardware platform will have the highest change to meet the system requirements certain metrics of each platform will need to be compared to one another. These

metrics will be further discussed in Section 3.1. An important thing to note is when different platforms such as GPU and CPU are being combined in a heterogeneous system. An heterogeneous system mainly aims to optimise throughput of a neural network and often has communication between platforms as bottleneck for latency. On the other hand an homogeneous system target the optimisation of the latency [50]. In order to tackle this problem Xuechao describes in the paper [50] a tile-grained pipeline architecture (TGPA) for low latency inference of CNN models.

The paper from Nurvitadhi [76] looks for example to compare BNNs on CPU, GPU, FPGA and ASIC. In this paper each hardware platforms acceleration is compared to the CPU implementation of the BNN. Another paper where CPU, GPUs and FPGAs are compared with each other is in the paper from Yu [136]. In this paper the hardware platforms are implemented in data-centers. This paper concludes that the FPGA has the lowest latency on the different networks tested. The GPU on the other hand achieved the highest throughput. The GPU reached this performance at a batch size of 128, while until the batch size reaches 64, the FPGA implementation has an higher throughput.

It can be seen from the paper of T. Wang [53], that the FPGA has the highest change as hardware platform to succeed. This paper did a survey on FPGA based neural networks. The general conclusion from this was that the ASIC has the highest theoretical performance but has too large of an development time and is too complex. The CPU and GPU both had an easier development time than the FPGA but have general less performance. This meant that an FPGA inference of a CNN has the highest change to succeed the reach the required low latency, while having a low enough development time to be made in this thesis. This general thought was shared when comparing RNNs (Recurrent Neural Networks) and BNNs for FPGA, CPU, GPU and ASICs [76, 137].

Another paper that compares FPGA and ASIC hardware accelerators is written by Shawahna [46]. This paper also mentions that ASIC designs have in theory an higher performance however the development time and flexibility of ASIC designs are not adequate enough for this thesis. While an FPGA does have enough flexibility and a low enough development time. Many more papers believe that an FPGA design will be the best design for a CNN accelerator such as [56, 61, 82].

Some disadvantages from FPGA acceleration come from how it is implemented. This comes from that most FPGA accelerators only implement the convolution layer or the Fully-Connected layer [44]. However only accelerating certain layers and not all of them brings certain limitations to the acceleration. The biggest limitation is the data communication overhead. This overhead comes from that the unaccelerated layers are executed on the CPU and need to communicate back and forth with the FPGA on for example the PCIe connection [44]. An example of an FGPA network which can meet the requirements of this project is discussed in the paper by Zhou [78]. There the BNN has an accuracy of 86.06 % on CIFAR-10 dataset while reaching 332,158 images per second with a constant latency of 4.9μs. Or another example where the latency of the network is only 75ns is in the paper of Duarte [68].

The paper written by Abdelouahab [26], does another analysis when comparing available hardware to accelerate a CNN workload. This paper shows that for different networks, different hardware platforms will work better for low inference times. As in this paper in table 2.5 the GPU platform has the lowest inference time, while as mentioned before FPGA platforms have the highest change to succeed.

The paper written by Rush [82] compares CPU, FPGA and GPU hardware platforms extensively and gives a good analysis about which metrics for different platforms will perform better. Figure 3 of this paper shows why for different features FPGA such as timing latency is better but for DNN training a GPU is better.

Chapter 3

Use case requirements

This chapter will explain the most important choices for this thesis and how they have been made. It will start with discussing the design methodology that is going to be used for the rest of the thesis. In this section it will explain how the different metrics of the pipeline will be measured and show on which metrics a performance analysis will be done. Once the metrics are determined together with the performance requirements of the pipeline, the different hardware constraints of the pipeline will discussed. The hardware constraints determine what the input and output need to be for be for the eventual neural network of the system. From the hardware constraints together with the background research, certain design choices will be made for the rest of this thesis. Finally the design choices of this thesis will determine which neural networks will be investigated. These neural networks will be investigated to show how they correspond to the possible future neural network for the gravitational waves interferometer control system.

3.1 Design methodology

In order to properly design the pipeline and see if it can meet the performance requirements, some evaluation metrics will need to be determined. Together with how these metrics will be measured. From Section 2.2 it has been made clear some important metrics to look at with network inference are accuracy/ robustness, power/energy consumption, throughput/latency and cost (resources) [138]. Considering the neural network is still under development, another important metric is the development time of the accelerated neural network.

The first metric to discuss is the latency of the network. For the network that is going to be developed it is important that the inference is low latency. Within this project low latency is defined as that from the input of the camera till the output of the neural network the latency is equal or lower than 0.1 ms.

The next metric to consider is the throughput. Often in other literature throughput of a system is usually expressed by the number of Multiply Accumulates (MACs) an accelerator can perform per second [26]. Now because this pipeline is about using cameras the throughput is measured using frames per second (fps). The two different metrics can be directly related, but MACs are in the case of this thesis not of interest. For this system there are two different cameras measuring each respectively the near and far field. These two cameras each correspond to the same output of the neural network. This means that within the 0.1 ms two different images need to go through the network. By using Equation (3.1) the amount of fps or throughput can be determined for this project.

$$Frames\ per\ seconds = \frac{Frames}{Inference\ time} = \frac{2}{0.0001\ s} = 20000\ fps \tag{3.1}$$

The final metric this project will discuss, is the cost or in other words the amount resources to be used by the network. This metric will be explored, to ensure that eventual system is realistic

and if it can fit on available hardware. For the sake of this project there is no limitation on the cost.

As the neural network is still under development, the accuracy/ robustness of the network will not be investigated. However as is discussed in Section 2.2.4, if complexity reduction techniques will be applied, then this needs to be accounted for in the conclusion in case a certain accuracy target needs to be met for this network in the future.

Finally the metric of power/energy consumption will not be discussed in this thesis as it is not of interested in regards to the research question.

3.2 Hardware constraints for pipeline

The pipeline that must be created for the neural network for a future gravitational waves interferometer control system has certain constraints to reach the required latency of 0.1 ms. These constraints do not only come from the pipeline itself, but also from the input and output of the pipeline. The pipeline to be optimised can be seen in Figure 3.1.

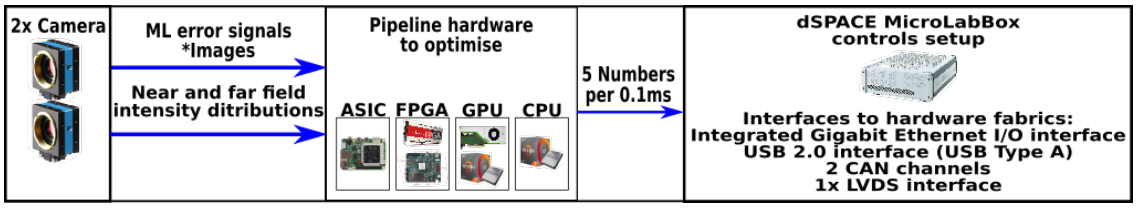


Figure 3.1: Pipeline to optimise.

This pipeline consist of 2 cameras which measure the error signals that are created from the misalignment of the Fabry-Pérot cavities. Next to the cameras the pipeline consists of hardware fabrics that runs the machine learning network and obtains the results. These results must be communicated with the dSPACE MicroLabBox which does the control of the system for the mirrors.

3.2.1 dSPACE MicroLabBox

The first constraint of the pipeline comes from the dSPACE MicroLabBox. The dSPACE MicroLabBox needs to receive data every 0.1 ms. This is because the control loop of the longitudinal control of a Fabry-Pérot cavity is 10 kHz. In case the loop is made slower the longitudinal control will fail and the experiments cannot be conducted. The datasheet of the dSPACE MicroLabBox informs the user which interfaces can be used to communicate with the possible hardware fabrics [6]. Depending on the hardware fabrics chosen to run the machine learning network on, one of these interface will be used to send the necessary control data every 0.1 ms. This data exist out of 5 numbers for the mirror misalignment, which is the result of the machine learning network. The interfaces and their speeds can be seen in Table 3.1.

Communication Protocols	**Integrated Gigabit Ethernet**	**USB 2.0**	**CAN channels**	**LVDS interface**
Transfer speed:	1Gbit/s [139]	480Mbit/s [140]	1Mbit/s [141]	655Mbit/s [142]

Table 3.1: Communication transfer speeds of different protocols in the dSPACE MicroLabBox.

The transfer speeds in table 3.1 are needed to calculate how much time is being used to send the result of the neural network to the dSPACE MicroLabBox. This time is then substracted from the 0.1 ms to determine how much time is left for the actual running of the neural network.

These performance requirements of the pipeline will be discussed in Section 3.3. For the dSPACE MicroLabBox together with its fastest protocol in Equation (3.2) it is determined that it needs

$$Data\ Trasfer\ Time = \frac{bits\ to\ Transfer}{interface\ speed} \times 1000 = \frac{8\ bit}{1000 \times 10^6\ bit/s} \times 1000 = 0.000008\ ms \quad (3.2)$$

3.2.2 Cameras

The second part that constraints the pipeline is the input. The input comes from two different cameras that take images from the near and far field distribution. Considering that the dSPACE MicroLabBox needs a result every 0.1 ms for the control system, the cameras also influence the final latency. From this can be concluded, that first the cameras need a result within 0.1 ms and then the remaining time is used to run the neural network and transfer the result to the dSPACE MicroLabBox. The longitudinal control needs a control loop of 10 kHz. This would mean in turn that the frames per second (fps) of the cameras at minimum needs to be 10000 fps. By having 10000 fps, every frame can then run through the neural network giving a new result. For obtaining the correct data from the setup, special cameras are needed that are able to obtain the data from the far field and near field intensity distributions. Currently for this experimental test setup the available cameras and their constraints can be seen in Table 3.2.

Cameras specs	Budget	Moderate	Professional
Supplier	Basler	Mikroton	Optronics
Resolution	64x64 32x32	90x90 60x60	192x192 192x120
Frames/second (fps) Max@res	4081 fps 4830 fps	22900 fps 33900 fps	20000 fps 30366 fps
Communication	USB 3.0 5 Gbit/s [143]	Cameralink interface 850 MB/s	CXP-6 6.25 Gbit/s
Pixel sizes (in bits)	8, 10, 16	8, 10	8, 12
Total Cost (two cameras)	1000,-	11.000,-	15455,-

Table 3.2: The different camera constraints.

Each of these cameras in Table 3.2 have different specifications which provide different constraints for the hardware fabrics that run the machine learning network. The constraints are first the resolution of the cameras together with the pixel size. This determines how much information the neural network has, to create a correct solution for the control system. On top of that the resolution and the pixel size determine the amount of data the camera needs to transfer over a specific communication protocol. The time needed to transfer all this data with the time needed to take the frame is then again subtracted from the time the neural network has to operate.

Lastly from Table 3.2 it already has been made clear that the Basler camera is not able to be used in the experimental setup. This is because this camera is not able to reach the required fps of 10000. For this research the Miktron camera with a resolution of 90x90 can be used to determine if the low latency pipeline can be created. Now because both cameras of the network will be running concurrently it can be determined how much time is needed from taking the image to the input of the pipeline. Each image is RGB with each channel having 8 bits. Then from Equation (3.3) it is noticed that the pipeline will need to transfer 24300 bytes per frame. The camera has an interface of 850 MB/s, meaning from Equation (3.4) it can be determined the pipeline has 0.0714 ms for obtaining the necessary output. This calculation has been done for how much time one frame takes to be transferred to the pipeline.

$$Total\ bytes\ to\ transfer = \frac{Resolution \times\ Channels \times bits}{8} = \frac{90 \times 90 \times 3 \times 8}{8} = 24300 \quad (3.3)$$

$$Frame\ transfer\ time = \frac{Bytes\ to\ Transfer}{interface\ speed} \times 1000 = \frac{24300}{850 \times 10^6} \times 1000 = 0.0286\ ms \quad (3.4)$$

However for this project the neural network is currently under development and for that reason also no camera has been fully decided on. This means that for this project the investigation will be done in getting the neural network to have a result within 0.1 ms, while for follow up research this time lost in communication needs to be taken into account.

3.2.3 The neural network input and output

From the previously discussed constraints the input and output of the pipeline or in other words neural network can be discussed. This part will be briefly discussed in order to show what limitations the pipeline needs to deal with and what possible bottlenecks can be for the pipeline when trying to reach the necessary latency. The neural network I/O can be seen in Figure 3.2.

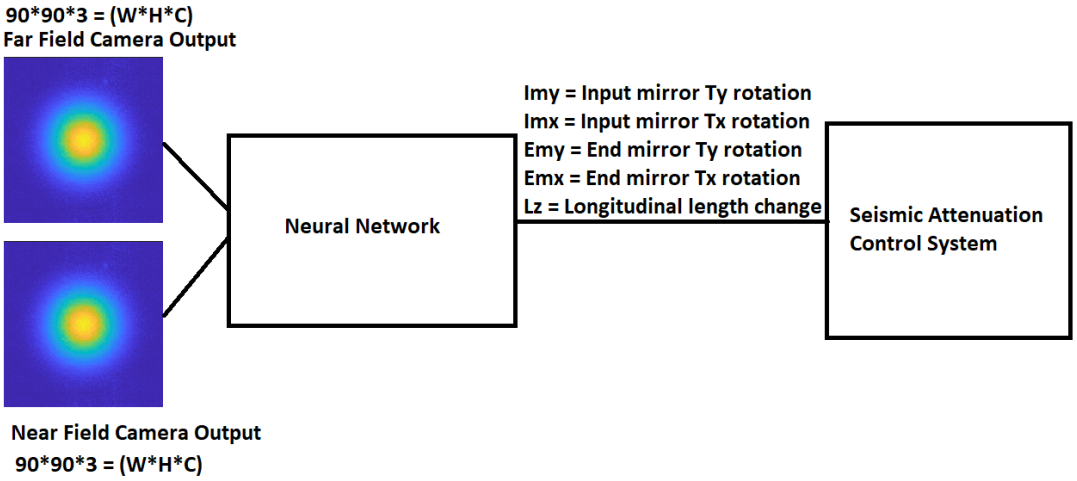


Figure 3.2: Neural network I/O.

As can be see in the image the input of the neural network are two different images of near field and far field each of size 90*90*3. The variables Imy, Imx, Emy, Emx and Lz of the mirror can be derived from the images. These abbreviations can be seen in Table 3.3. Each of these outputs are 8 bits in size. With this information the dSPACE MicroLabBox should know what to do to keep the mirrors in the correct position and keep the influence of external factors on the measurements to a minimal.

Abbreviation	Equivalents
Imy	Input mirror Ty rotation
Imx	Input mirror Tx rotation
Emy	End mirror Ty rotation
Emx	End mirror Tx rotation
Lz	Longitudinal length change

Table 3.3: Output neural network abbreviation meaning

3.3 Design choices

From the design methodology and the performance requirements of the pipeline already some design choices can be made in order to reach those requirements. The first design choice is that the chosen hardware platform is the FPGA. For a neural network the best structure of each layer is highly dependent on the hardware platform used. Consequently being able to optimise every layer of the neural network would be a great advantage in reaching a low enough latency. For that reason a framework that uses a streaming architecture as discussed in Section 2.3.3 will be the best choice. From these frameworks, FINN was able to reach a latency of 0.31 μs [80] for a neural network. Henceforward FINN will be used to investigate different configurations of the framework and how they affect the latency of the neural network. To obtain the best possible structure of the neural network on the FPGA however, involves exploring to large of a design space and is to time consuming [47]. This means that when investigating FINN, which uses streaming architectures, not the entire design space can be investigated because of the long compilation times that are needed for each configuration. Consequently educated design choices need to be made for investigating different configurations.

3.3.1 FPGA choice

At the moment of evaluating different hardware platforms to accelerate a neural network a trade off between the metrics mentioned in Section 3.1 is always considered. For this thesis the FPGA is chosen as hardware platform to be further investigated. On the subject of implementation a neural network on the FPGA, there are numerous choices with different FPGAs each of which can run different implementation alternatives and different deployment parameters including batch sizes and power modes. All of the implementation alternatives will deliver different performance characteristics.

The paper from Blott, [138] compares different implementation alternatives and determines that pruning and quantization are orthogonal, and yield optimal design points when combined for the pipeline. As mentioned before in Section 2.3.2 the FPGA benefits the most of the reduced-data-type precision techniques. On the grounds of that an FPGA where it is easier to provide a lower precision data types [89] will be a better hardware platform.

Neural networks on FPGAs as hardware platform has multiple examples of reaching the required latency of this project as has been show in Section 2.3.5. From the research question it was asked what the appropriate hardware fabrics are in order to reach the low latency pipeline. While from Section 2.3.5 it has been made clear that ASICs have the highest theoretical performance of reaching the required latency the current neural network is still under development. Considering that FPGAs will be a better option as they are more focused on reconfigurability than ASICs (which are for low latency) and more low latency than throughput which is for GPUs. The final comparison between all different hardware fabrics on the metrics can be seen in Table 3.4. This table is based on the background research and the papers from Rush and Abdelouahab [26, 82].

3.3.2 Framework neural network inference

The chosen framework to be used together with the FPGA is the experimental framework from Xilinx Research Labs FINN. FINN specifically targets quantized neural networks, with emphasis on generating dataflow-style architectures customized for each network. Each of the network are trained and created using Brevitas. Brevitas is a Pytorch library made by Xilinx for quantization-aware training. Next to that FINN includes the FINN compiler and the finn-hlslib Vivado HLS library of FPGA components for QNNs.

With the generating of dataflow-style architectures in FINN every layer will be implemented separately on the FPGA as discussed in Section 2.3.3. This means that when running a neural network on the FPGA, the network will have a fixed latency which only is determined by the length of the pipeline. This also means that no piece of hardware will be used for two separate layers and there will be no overhead in reloading weights and intermediate results going between

Metric/ Feature	**Analysis**	**Hardware**
Neural network training	GPUs can be highly parallelised and are supported by many frameworks for training [26, 83, 84]	GPU
Neural network inference (batch size small)	FPGAs can be highly customised per layer to obtain the lowest latency	FPGA
Neural network inference (batch size large)	GPUs have large parallelisation capabilities	GPU
Interfaces	As FPGAs are close to hardware it can connect to many interfaces	FPGA
Resources/Size	FPGAs can be highly customised with minimal overhead	FPGA
Customization	FPGAs enable the possibility of a custom-tailored accelerator architecture in hardware	FPGA
Development/ Ease of use	CPUs are easier to program than GPUs with frameworks and both are easier than FPGAs	CPU

Table 3.4: Comparison between FPGA, GPU and CPU for neural networks.

layers [80]. Consequently the latency and overhead of the neural network will be lower. Next to dataflow-style architectures, there are layer-by-layer style architecture. However these types of architectures result in much higher latency and latency variation because of the reloading of the weights and reusing hardware. The FINN dataflow-style architecture does however has a limitation, because everything of the neural network needs to be implemented using the on chip resources. As a consequence it is not possible for arbitrarily large CNNs to run on most FPGAs and must be taken into account when developing low latency accelerators [124].

FINN targets quantized neural networks. With a special variant of quantized neural networks being the BNN as discussed in Section 2.2.4. For FINN this is a quantized network with 1-bit bipolar (-1, +1 values) precision. Now the higher the quantized bits the higher network accuracy can be expected as more information is being processed per layer. However this can lead to more resources to be used to store these weights and can include more latency as more information need to be processed. To investigate if the required latency can be obtained, some of FINNs pretrained BNNs will be investigated. This is because these networks have the highest chance to succeed [138].

To conclude FINN was chosen for the following reasons. Firstly each layer will be implemented separately. Meaning every layers latency can be optimised separately to a certain extend. Secondly, FINN has multiple analysis tools, together with some pretrained example networks from Brevitas to find out any bottlenecks FINN may have. Some of these analysis tools include resource estimation or simulation of the networks for runtime with different batch sizes. Thirdly, FINN is an open-source project that is still under development during the time of the thesis. This means that when having to implement a specific layer that is not implemented in FINN, then it is possible to implement the layer yourself into FINN and contribute to the FINN project. Finally FINN has already proven in past research that it can reach the required latency as can be seen in table 8 of the paper discussed by Blott [138] or in another paper discussed by Blott [124].

The other framework that was a possibility was Vitis AI as mentioned in Section 2.3.3. This was the other framework that is open-source and being actively being developed by Xilinx. Vitis AI uses the Xilinx Deep Learning Processor Unit (DPU). This is a programmable engine dedicated to convolutional neural networks. While Vitis AI is a framework that has many analysis tools and can use quantized neural networks it does not give any indication that it can reach the required latency of 0.1 ms. This can be seen from all the performance numbers on many different networks, on different boards where none of them reach the 0.1 ms requirement [144]. This could be happening because of how Vitis AI is implemented. It appears as Vitis AI uses an overlay architecture for implementing the neural networks. This architecture is flexible as it enables many types of CNNs

to be executed on an FPGA but is not efficient as it uses off-chip weights and activations. Which means that there is a need to transfer the weights and activations on chip making the real-time low latency inference difficult [145].

FINN however does have some limitations as discussed before and in the papers [80, 124]. The first limitation is in the accuracy the neural networks can achieve. As FINN is based on quantized networks. By having limited precision available for the weights and the activations, the amount of information that can go between layers is also limited. Meaning as more information is lost between layers in general the lower the accuracy will be. The second limitation is that FINN based neural networks are limited by the resources on the FPGA board. As the board is required to implement compute units for each of the CNN layers on the board itself. However by having all information close by there will be less communication overhead then when needing to use off-chip resources. Finally as FINN is a framework for FPGAs it is limited by the frequency the FPGAs can run on, which are in general lower than a CPU or GPU.

3.4 Neural network architecture

In order to see if FINN can reach the required latency of 0.1 ms certain neural network architecture need to be investigated. As the neural network for the gravitational waves need images one of the networks to be investigated has convolutional layers. These layers are often used in neural networks that need to identify images because of the locality of the information. The other neural network that will be investigated is the one that according to the paper discussed by Blott [124] was able to pass the 0.1 ms requirement. There are a lot of different neural network topologies, each of these can be trained with different datasets. These topologies also have different numerical representations, learning techniques and hyperparameter selection [138] as is discussed in Section 2.2. All of these topologies can again produce different results in terms of latency, accuracy and cost. As a means to determine if a neural network can reach certain requirements, FINN has several pretrained neural networks available for testing and analysing the FINN framework. These networks are trained with different quantizations. For that reason the pretrained networks TFC and CNV of the FINN framework will be used to determine if the required latency can be reached. Both of the network architectures will be shown in Figure 3.3 and Figure 3.4[1].

From Section 2.2.3 it has been made clear that each neural network has a different computational complexity and required storage capacity. The number of parameters each network has is defined by the total number of weight and biases. The higher the number of parameters the more features are used to determine the result. As more information is passed in each layer, often the higher accuracy can achieved for the neural network. Table 3.5 shows the number of parameters and operations used for the example neural networks of FINN. The SFC and LFC networks are family of the TFC network, where respectfully they use 256 and 1024 instead of 64 neurons per fully connected layer. In this table these networks are included to show that by just changing the amount of neurons per layer the total amount of parameters can grow larger than the CNV network which has more layers, while the amount of operations to be executed is still lower than the CNV. For all the networks that are investigated a thresholding layer is introduced when creating the network in FINN to ensure that the input quantization of these networks is converted to standalone thresholding that the rest of the layers can use. This does introduce some latency, but from the small precision that is investigated this cost practically disappears.

Topology:	TFC	SFC	LFC	CNV
Parameters (Mbits)	0.06	0.3	2.9	1.5
Operations (M)	0.12	0.6	5.8	112.5

Table 3.5: Neural network input and output layer information.

[1]This figure is generated by adapting the code from `https://github.com/gwding/draw_convnet`

3.4.1 Tiny fully connected network

The first neural network to be investigated is the Tiny Fully Connected network (TFC) network. From the paper discussed by Blott [124], it is known that a likewise neural network was able to reach the required latency. The TFC network is a multilayer perceptron (MLP) for MNIST classification with three fully connected layers [138]. In the TFC variant of this MLP network only 64 neurons are used per fully connected layer. Which means the parallelisation and by that the acceleration is limited to the amount of neurons per layer. The network architecture can be seen in Figure 3.3. This architecture does not show the threshold layer that will be implemented with FINN, because this layer will only be implemented for a small quantization of the weights and activation.

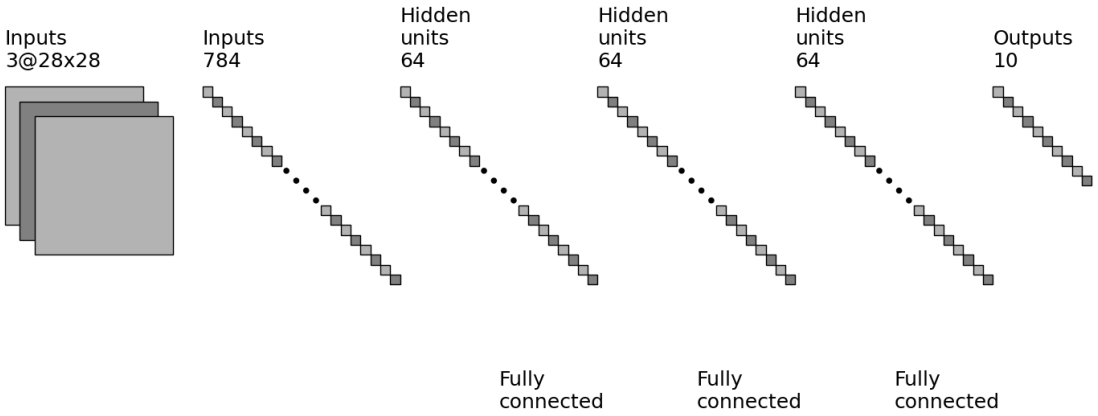


Figure 3.3: TFC architecture [1].

3.4.2 CNV

The other network that will be investigated in this thesis is the convolutional neural network named CNV. CNV is a derivate of the VGG16 topology. The VGG16 topology is used for deep learning image classification problems with multiple convolutional layers [58]. The CNV variant is in particular trained on the CIFAR-10 dataset as it it classifies 32x32 RGB images. This network contains a succession of (3x3 convolution, 3x3 convolution, 2x2 maxpool) layers repeated three times with 64-128-256 channels, with the final time not having the maxpool layer but a fully connected layer. This is followed by two fully connected layers of 512 neurons each. In the CNV-w1a1 variant where the weights and activations are quantized to bipolar values (either -1 or +1), the exception is in the input (which is RGB with 8 bits per channel). To summarise CNV-w1a1 utilizes binary (1-bit) quantization, while CNV-w2a2 utilizes ternary (2-bit) quantization. This network can be used to determine if a future neural network can reach the 0.1 ms requirement or determine where possible bottlenecks are in FINN if it cannot reach it. The neural network architecture can be seen in Figure 3.4. This architecture does not show the threshold layer that will be implemented with FINN, because this layer will only be implemented for a small quantization of the weights and activation.

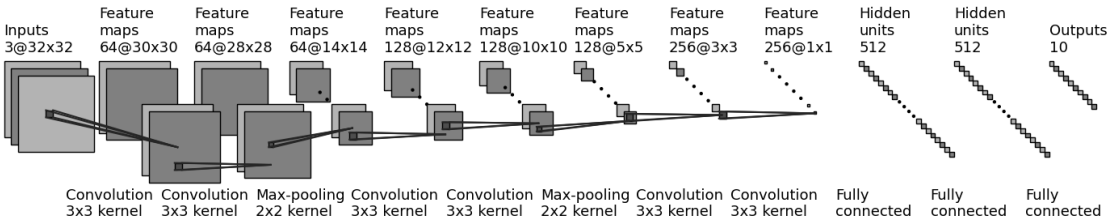


Figure 3.4: CNV architecture [1].

Chapter 4

Exploration pipeline designs

In order to explore the design space of the pipeline, analysis tools are necessary that can measure the metrics mentioned in section 3.1. This chapter further discusses certain consideration that have to be made with those tools and the motivation behind using them. The design analysis tools that are build in FINN will be discussed together with how the performance of the metrics can be improved or changed by using the framework FINN. Finally all the current hardware options of FINN will be discussed to find out in Chapter 5 which boards will be able to fit a design that can have a latency of less than 0.1 ms.

4.1 Considerations and motivation

To penetrate the design space of this project certain considerations need to be made. This is because there are numerous different neural network topologies that can be investigated. The first consideration to be made includes the large number of parameters that can be set by the use for the neural network. As mentioned in Section 2.2.2 these are called hyperparameters. The hyperparameters are values that are set before training begins and are values that control the learning process of the network. Which in turn influence the metrics mentioned in Section 3.1. The neural network architectures mentioned in Section 3.4 will help in penetrating the design space of these parameters and give an idea on how they influence the latency. The second consideration for exploring the design space of this project is in choosing the hardware platforms that can achieve the wanted latency. As the available hardware platforms is framework dependend only FPGAs from Xilinx will be investigated. This is because these are the only ones that are supported by the framework FINN. Each hardware platform has different advantages and disadvantages when trying to achieve certain requirements.

To be able to evaluate the performance of the neural networks, the framework-specific resource allocator needs to be investigated. This allocator in FINN sets the parallelisation parameters for each layer in the CNN dataflow. By optimising the parallelisation parameters the low latency needs to be achieved, while remaining in the resource limitations of the hardware platforms.

4.2 Parallelisation parameters FINN

In order to achieve the required low latency the parallelisation parameters of FINN need to be optimised. In the FINN framework folding describes how much a layer can be parallelised. For each layer in a neural network there are several folding factors. The vector Processing Elements (PE) determines the parallelisation over the outputs of the layer (Output Feature Maps), while the Single Instruction, Multiple Data (SIMD) determines the parallelisation over the inputs of the layer (Input Feature Maps). The higher the parallelisation that can be achieved per layer the lower the latency will be. However the more FPGA resources will be used, meaning that there will be a trade-off between available resources and latency.

To further understand how these parallelisation parameters and folding work, the convolutional layer will be used to explain them. In FINN a convolution is converted to matrix multiplications, where one of the matrices is generated by sliding a window over the input image. The key part of this operation is the sliding window operation. This operation extracts a group of elements located next to each other from an array, then goes on to the next group of elements. The two key parameters for this operator are window size, which is how many elements there are in each window, and stride, which is how many elements the window moves at every step. The matrix multiplications that need to be calculated are then converted to multiply-accumulate (MAC) circuitry on the FPGA fabric. To then determine total number of MAC operations, the parallelism variables: PE, SIMD and the number of pixels processed in parallel are used. The throughput is the determine by the total of MACs that can be executed in parallel. Equation (4.1) is used to determine the total number of MACs per layer.

$$Total\ MACs = PE \times\ SIMD \times pixels\ in\ parallel \tag{4.1}$$

The variables of eq. (4.1) are the key parameters in FINN in determining the latency, throughput and amount of resources used. From the the number of PEs (P) and the number of SIMD lanes per PE (S) the total folding factor of a layer can be determined. The total folding is also the number of cycles required to complete that specific layer. For a X $\times$ Y matrix the neuron fold $F^n = X/P$ and $F^s = Y/S$ as the synapse fold [80]. The total folding of a layer is then obtained as $F = F^n \times F^s$. Each layer has a different network dependent constant F^m which will affect the total folding. Considering that the total folding factor of a layer is also the number of cycles required to complete that specific layer, the expected throughput and latency can be determined from this factor. The latency will be the total folding factors of all the layers combined, while the throughput of the neural network will be determined by the highest total folding factor, because the network is pipelined.

The amount of PE and SIMD elements the user can assign to a layer is limited by how FINN has implemented that specific layer. For example for the CNV architecture the first layer is a convolutional layer. This layer is implemented by a sliding window and has an IFM of 3. The IFM for the first layer is determine by the amount of input channels and for an RGB image this is 3 channels. As IFM equals to 3 for the first layer the amount of SIMD is ≤ 3. While the amount of PE that can be assigned to a layer is limited to the OFM. In FINN the eq. (4.2) and eq. (4.3) need to be true, as otherwise FINN would not be able to create the layer for the network. While the total amount of PE and SIMD the user can use is limited by the resources available on the hardware.

$$OFM\ \%\ PE == 0 \tag{4.2}$$

$$IFM\ \%\ SIMD == 0 \tag{4.3}$$

4.3 Neural network analysis tools TFC/CNV

To see if the neural network can achieve the required latency the neural network needs to be analysed. For this FINN has several analysis tools available that can analyse the metrics mentioned in section 3.1. Each of the results will then be compared via graphs in section 5.2. To achieve the lowest latency the strategy in table 4.1 will be used for determining the configurations for the timing and resource analysis.

4.3.1 Timing analysis

The first analysis is the timing analysis for each neural network. This analysis will tell us if a specific neural network can achieve the required latency. In case the pipeline is unable to reach the required latency of 0.1 ms, then it is also unnecessary to do any resource analysis as the pipeline

Current Step	Description Step
1.	Calculate latency for all layers using the folding factors
2.	Pick the slowest layer
3.	Try to accelerate the layer by increasing SIMD or PE by factors of two
4.	Depending on setup start with SIMD or PE first, only switch to the other if the first one is maxed out
5.	If the layer cannot be accelerated any further; Choose the following slowest layer and go back to 3.
6.	Once the layer is accelerated go back to 1. If no layer can be accelerated anymore end the exploration

Table 4.1: Low latency strategy FINN.

is unable to reach its requirements. In order to do the analysis certain tools need to be available that can give the user estimations on when the latency can be reached.

The first step of the timing analysis is to determine how many cycles each layer needs to execute the neural network. The layer that uses the most amount of cycles will be the slowest layer and as such will need to be the first layer to be optimised. In order to do this analysis FINN has a function called: "exp_cycles_per_layer". This function estimates the number of cycles per sample for dataflow layers in the given model by calculating the folding factor of that layer.

Once the cycles per layer are determined it is possible to give a rough estimate of what the latency would be for the pipeline. The latency would be calculated by using eq. (4.4). This calculation will be using the frequency of the FPGA and the latency is shown in seconds. The calculation shows us that the higher the frequency the lower the latency will be. It is important to notice this analysis is an initial estimation, meaning that the actual latency of the neural network can be lower.

$$Latency = \frac{cycles}{frequency} \tag{4.4}$$

The second step of the timing analysis is a software emulation of the execution of the network. This will tell us more about the latency, while taking less time then fully creating the hardware and execution that. This software emulation can also be used to verify if the generated network is working correctly with the framework FINN. For the software emulation FINN uses "PyVerilator". This is a package that makes it possible to simulate verilog files using verilator [146] via a python interface. This simulation can give an estimation in terms of cycles for the pipeline. Then eq. (4.4) can be used again to determine the latency.

The final step in the timing analysis is the hardware timing. In this step the actual hardware for the FPGA will be generated to execute the neural network. FINN also offers the possibility to measure the network performance directly on the hardware by using the "finn.core.throughput_test" function. When running this function the metrics of the network are returned as dictionary. Considering FINN uses software for this test instead of hardware timing with cycles, there can be some software overhead in the measurements. This should be taken into account when the measurements are done and the results are discussed. One final part to look at before the hardware timing is done is by looking at the Worst Negative Slack (WNS) of the generated hardware. This tells us at which frequency the generated hardware can run as it is the slack of the the critical path of the neural network.

4.3.2 Resource Analysis

Following the timing analysis, a resource analysis will be done to discover, if the generated hardware can fit on real hardware. The resource analysis will identify how many resources the neural network will need. In case a network does not fit on the board, specific layers can be optimised to

be balanced between latency and resources used. There are four different steps the resource analysis can take to determine if it can fit in available boards, that are compatible with FINN. For the resource analysis, the parallelisation parameters will be used to optimise the network performance within a given resource budget.

The first step for the resource analysis will be a rough estimation by means of simple calculation based on the resources given. This is an empirical fit of the LUT model based on the resources given [124]. It uses the calculation shown in eq. (4.5) and should show an estimate of the post-synthesis LUT usage. In the calculation, abits is the size of the activation bits used and wbits of the weights. So depending on how much information the weights have, it is already possible to see that more resources are needed.

$$Post - Synth\ LUT \quad = 1.1 \times abits \ \times wbits \times PE \times SIMD + 300 \tag{4.5}$$

From the calculation the following step of the resource analysis will be a High Level Synthesis (HLS) estimation. This step is done at the same time as the software timing analysis, as it shows if a particular configuration fits on an FPGA. The software emulation step generates IP blocks from the corresponding HLS layers. The IP blocks are then used by Vivado to estimate the amount of resources used. This estimation is a really generous estimation, because when generating the actual hardware, Vivado will optimise the resources. From this can be concluded that there is a design area which will not fit in the HLS estimation of resources, but will fit on actual hardware.

The final resource analysis is the post-synthesis analysis. This step extracts the FPGA resource results from the Vivado synthesis. In this step FINN will have determined if a design is feasible to be created for hardware. From there on the hardware can be created for actual hardware. The post-synthesis results are still overestimation compared to the post implementation step. The reason for this is that Vivado still optimises the resource usage when creating the configuration bitstream for the FPGA.

4.4 Hardware options FINN

The FINN project is created by Xilinx Research Labs to explore deep neural network inference on FPGAs. Considering that FINN is created by Xilinx, only FPGAs created by Xilinx are currently compatible with the FINN framework. FINN targets boards supported by "Python Productivity for Zynq" (PYNQ). PYNQ has been created to make it easier for designers of embedded systems to exploit the unique benefits of Xilinx devices in their applications. At the moment of writing FINN supports the Pynq-Z1, Pynq-Z2, Ultra96, ZCU102 and ZCU104 boards, with some preliminary support for Xilinx Alveo boards. These Alveo boards include the Alveo U25, Alveo U50, Alveo U200, Alveo U250 and the Alveo U280 [147]. To use FINN generated accelerators, PYNQ is not a neccasity to use as it is possible to use the Vivado IP integrator to put the design together with other RTL/IP blocks. In this section the capabilities and available resources of these board will be investigated, to determine in the evaluation which boards can be used for the low latency pipeline.

Each of the aforementioned boards have different use cases based on their resources. Each FPGA has multiple resources to report which can be important when comparing different results. These resources are LUT (LookUp Tables), DSP units, FF (FLip Flops (registers)) and BRAM. Together with the on-chip memory, the off-chip memory of the boards will also be discussed as it can be that the implementation needs the extra resources. Finally the max frequency of each of the FPGAs will be discussed, as from eq. (4.4) it has been made clear the frequency has a direct correlation between itself and latency. Most of the resources of the boards will be visualised in table 4.2. In the following subsections the resources of some of the boards will be further explained to understand what they mean for this thesis.

Pynq-Z1 and Pynq-Z2

Here the capabilities of the PYNQ-Z1 and PYNQ-Z2 that might be interesting for this thesis will be discussed. For example the type of off-chip memory and other I/O these boards have. Both of

Board	Freq	LUT	BRAM	FF	DSP slices
Pynq-Z1/Z2 [148–150]	667 MHz 766 MHz 866 MHz	53200	140 (4.9Mb)	106400	220
ZC607 [150]	667 MHz 800 MHz 1000 MHz	218600	545 (19.2Mb)	437200	900
Ultra96 [151, 152]	600 MHz 667 MHz 1500 MHz	70560	216 (7.6Mb)	141120	360
ZCU102 [151, 153]	600 MHz 667 MHz 1500 MHz	274000	934 (32.1Mb)	548000	2520
ZCU104 [151, 154]	667 MHz 766 MHz 866 MHz	230400	320 (11.0 Mb)	460800	1728
ALVEO U50 (HBM) [155, 156]	*	872000	*	1743000	5952
ALVEO U280 (HBM) [156–158]	*	1304000 [157] 1079000 [159]	2016	2607000	9024
ALVEO U200 [156, 157, 160]	*	1182000	*	2364000	6840
ALVEO U250 [160]	*	1341000 [159] 1728000 [156, 157]	*	3456000	12228

Table 4.2: Resources different boards FINN.

these boards use the ZYNQ XC7Z020-1CLG400C [148, 149]. This chip belongs to the Zynq-7000 SoC architecture and consists of dual-core Cortex-A9 processor [150]. From the datasheet in [150] it can be seen the XC7Z020 has three different maximum frequencies and is a reoccurring theme with the datasheets of Xilinx. The reason for this is that multiple parts of the FPGA board can define the maximum frequency of the device. Given that information the maximum clock of the FPGA is dependent on multiple factors and will almost never be reached. As was discussed in section 4.3.1 the WNS of the eventual design will tell us more of what the maximum frequency of that design can be.

The boards have furthermore 512MB DDR3 memory with an 16-bit bus @ 1050Mbps. Finally some intersting I/O both boards have are the 1G Ethernet, USB-JTAG Programming circuitry, USB-UART bridge and 16 total FPGA I/O. The FPGA I/O can for example be used to be further connected to the cameras or to the control system.

ZC706, ZCU102 and ZCU104

The following boards to be discussed are the ZC706, ZCU102 and ZCU104. The ZCU706 is used in the paper of Umuroglu [80] for the experiments and was the first boards that used a version of the TFC-network that passed the 0.1 ms requirement. The ZC706 uses part number XC7Z045 and just like the PYNQ-Boards belong to the Zynq-7000 SoC architecture [150]. As can be seen in table 4.2 this board has more BRAM, but less LUT than the ZCU104. This can show us already that depending on the design the wanted board may change. This change in resources is also because the ZCU102 and ZCU104 use a different Zynq architecture. The ZCU102 and ZCU104 use the Zynq UltraScale+ MPSoCs [151] architecture instead. The ZCU102 uses the XCZU9EG part and the ZCU104 uses the ZU7EV part. Each of these boards can be optimised for different applications and for that reason also have different resources and I/O that can be used. For the details of the I/O and off-chip memory their respective product pages of ZCU102 and ZCU104

can be conducted [153, 154].

Alveo boards

The final hardware platforms to be discussed are the Alveo boards that at the moment of writing only has some preliminary support from FINN. The first thing to notice from table 4.2 is that depending on which reference is considered the amount of resources the board have is different. For that reason in the experiments only the numbers that come from the datasheet will be used. The Alveo boards are designed for use in datacenters and as discussed in section 3.3.1, having no direct connection to the camera can already lead to much of a latency time. For this reason these cards are not used to determine if the neural network can achieve the wanted latency. What these cards can be used for is for the resource analysis. As in case neural network will not fit in the smaller boards then needs to be figured out if there exist FPGAs that can fit these designs. The Alveo boards can then be used as reference point to see if it can fit in existing boards and from there a new FPGA can be designed that has these resources.

Chapter 5

Evaluation

This chapter evaluates the performance of the neural networks discussed in Section 3.4 and discusses how these results can be used to explore the low latency pipeline. First this chapter discusses the experimental setup that is used to explore the different configurations. After that the measurement results will be further explained. For this first a timing analysis will be done, to find out if the pipeline is possible. This will be followed up by a resource analysis to find out, what kind of hardware is necessary to implement the pipeline. Finally this chapter will have a short discussion about the most important results and what they mean for the low latency neural network of the future gravitational waves interferometer.

5.1 Experimental setup

To evaluate FINN, for the low latency pipeline multiple example neural networks will be used. These neural network architectures exists off the TFC network discussed in Section 3.4.1 and the CNV architecture discussed in Section 3.4.2. Each of these neural networks can exists of different parallelisation parameter configurations that will result in different folding factors. These factors will be determined according to the steps in Table 4.1 for the experiments. After that the FINN design flow will be used to explore the design space of the neural networks.

The version of FINN to be used in the measurements is version v0.4b [161]. In this version primary support for the Alveo boards have been implemented. At the time of writing version v5.0b has already been released showing that FINN is still actively being developed on. For FINN, Vivado version 2020.1 is being used to generated High-Level Synthesis (HLS) files and used for the bifile synthesis.

Unless otherwise noted the target clock frequency for the measurement is 100 MHz to evaluate the resulting accelerators. Most of the experiments to be run will be fully done in the FINN framework to show the capabilities of FINN and to further explore the design space of the networks. Finally, for all the hardware experiments the previously discussed Pynq-Z1 in Section 4.4 will be used. This is a small FPGA and gives a general idea of what is possible with FINN.

5.2 Measurements

The experiments were performed to explore the inference of the two neural networks, with their respective data sets. For the inference tests, a batch size of 1 was used for determining the possible bottlenecks the networks may have and to discover how to design the neural network for the future gravitational waves interferometer.

5.2.1 Timing analysis

The first conducted experiments are regarding the timing analysis discussed in Section 4.3.1. For every experiment first the smaller TFC network will be evaluated and afterwards the CNV network will be evaluated. This will show where possible bottlenecks may occur when developing the future neural network with FINN.

The first experiment will use the "exp_cycles_per_layer" together with the parallelisation parameters discussed in Section 4.2 to determine the total folding factor of each layer. By adding the total folding factor of each layer, the total amount of expected cycles of the neural network can be calculated. For the first experiment the calculated latency will be compared to parallelisation parameters. The Figure 5.1 shows that by increasing the parallelisation parameters for each of the layers in the neural network the latency will go down. This figure compares the calculated latency of the TFC,SFC and LFC neural network based on the folding factors with increasing amount of parallelisation parameters. The figure uses logarithmic scales for both the x and y-axis. On the x-axis is the latency in ms from low to high and on the y-axis is the amount of parallelisation parameters from low to high used. This figure shows that depending on how many resources the FPGA has each of the Fully-Connected (FC) neural networks can be achieve the wanted low latency of 0.1 ms. Which comes in agreement with the paper of Umuroglu [80]. From Figure 5.1 can be observed that the more operations and parameters a neural networks has, the slower the network will be for the same configuration of the layers. This is because these layers can be further parallelised.

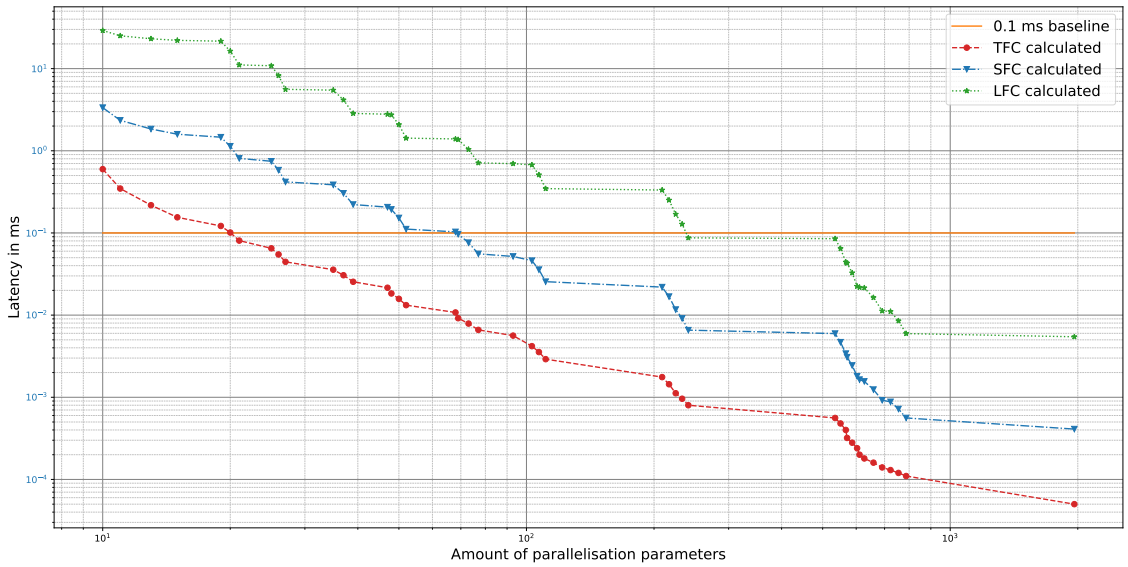


Figure 5.1: FC calculated latency compared to parallelisation parameters, for TFC, SFC, LFC network.

From the previous determined total folding factors of the network the expected throughput can be calculated for TFC,SFC and LFC neural network. For this the optimal throughput and the minimum throughput will be determined. The optimal throughput of this neural network is determined by the slowest layer. This layer will have the highest total folding factor. For a fully pipelined network this layer will be the bottleneck. The minimum throughput however is determined by the latency of the neural network. In this case only a single image can go through the network at a time. By then looking at the difference between the minimum and optimal throughput the design space of this metric can be observed. Figure 5.2 shows the results of these calculations for throughput. The figure uses logarithmic scales for both the x and y-axis. On the x-axis is the throughput in images/s from low to high and on the y-axis is the amount of parallelisation parameters from low to high used for that configuration.

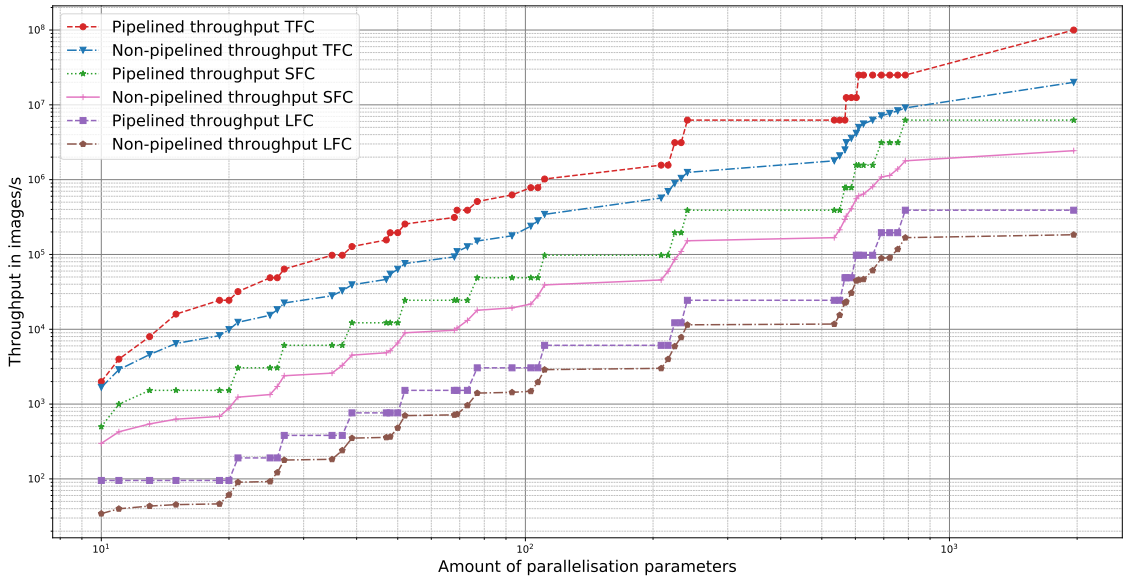


Figure 5.2: FC throughput calculated for TFC, SFC, LFC networks fully pipelined and non-pipelined compared to parallelisation parameters.

After looking at calculated latency of the FC-networks, the second step of the timing analysis will be done. In the second step for both the TFC and CNV networks a software emulation is done to determine the expected latency of the neural networks together with the throughput. This will be then be compared to the calculated latency and throughput. For the software emulation experiment of determining the expected throughput a batch size of 1 is chosen. While this will not show the maximum expected throughput of the neural networks, it will show what can be expected of the future neural network. Figure 5.3 shows a comparison between the software emulated latency and the calculated latency of the TFC network for the same configurations. The figure uses logarithmic scales for both the x and y-axis. On the x-axis is the latency in ms from low to high and on the y-axis is the amount of parallelisation parameters from low to high used. From Figure 5.3 can be observed that once again increasing amount of parallelisation parameters the latency will go down for the emulated latency. Another thing to notice in this figure on the bottom right is that while the emulated latency goes down, the calculated latency which is derived from total amount of expected cycles goes up. At the same time the parallelisation parameters for this configuration went up. One of the reasons this could be happening is that on this final configuration, one of the layers has not been fully optimised as indicated from the strategy of Table 4.1. Another reason as to why this can be happening is that in this final configuration the resources are more divided over all the layers. In that case the communication that is between each of the the layers can be better optimised and lead to an overall lower latency. Meaning that while each individual layer overall is slower, the communication between layers is faster.

Figure 5.4 which is the throughput measurement of the TFC network shows in more detail what is happening at that configuration of parallelisation parameters. Like previously the optimal throughput of the TFC will be determined by the highest folding factor of the network which is the pipelined. The non-pipelined throughput is determined by the calculated latency. The figure uses logarithmic scales for both the x and y-axis. On the x-axis is the throughput in images/s from low to high and on the y-axis is the amount of parallelisation parameters from low to high used for that configuration. From Figure 5.4 can be observed that in the final configuration, while the calculated throughput goes down as the slowest layer has a higher total folding factor, the emulated throughput still went up.

Figure 5.5 shows a comparison between the software emulated latency and the calculated latency of the CNV network for the same configurations with different clock frequencies. The

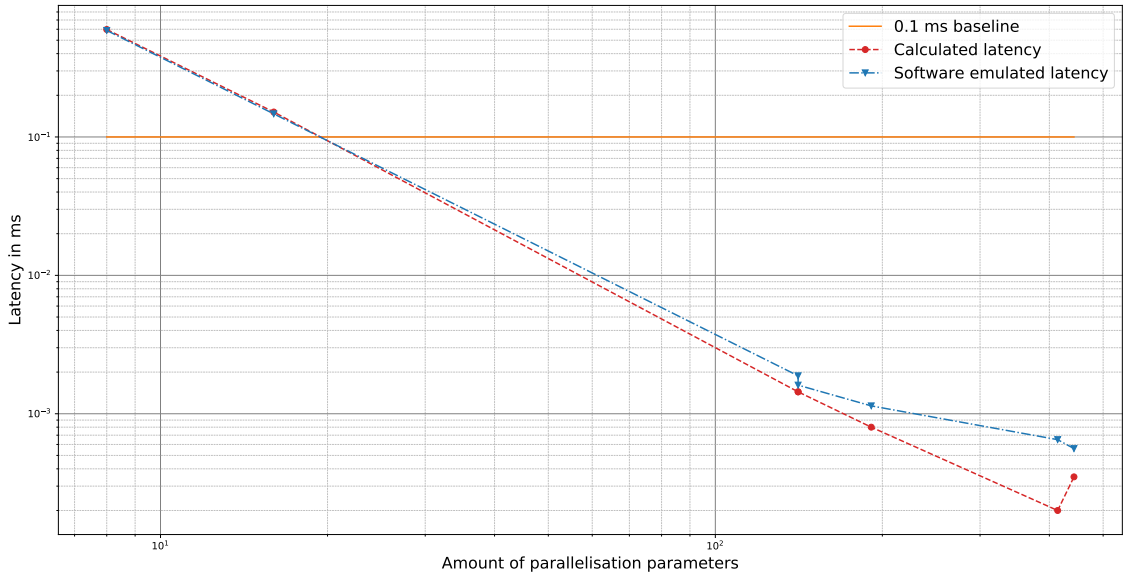


Figure 5.3: TFC software emulated latency compared to the amount of parallelisation parameters.

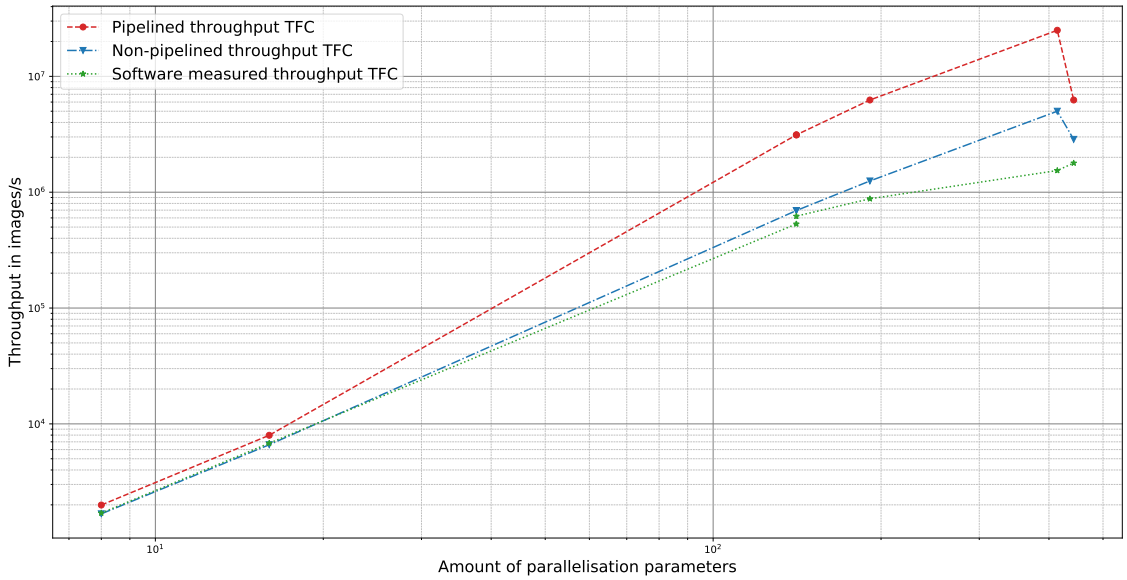


Figure 5.4: TFC software emulated throughput compared to calculated fully pipelined and non-pipelined throughput for different parallelisation parameters.

figure uses linear scales for both the x and y-axis. On the x-axis is the latency in ms from low to high and on the y-axis is the amount of parallelisation parameters from low to high used. From Figure 5.5 can be observed that once again increasing amount of parallelisation parameters the latency will go down for the emulated latency. Another thing to notice in this figure on the bottom right is that the software emulated latency with a 200 MHz clock is able to pass the 0.1 ms baseline requirement.

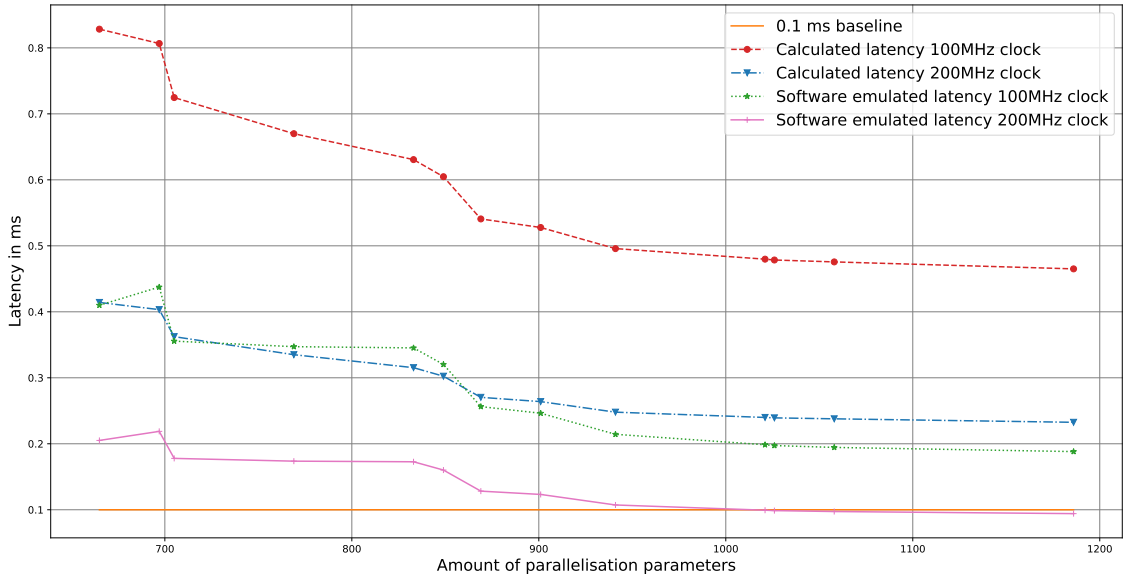


Figure 5.5: CNV software emulated latency compared to the amount of parallelisation parameters.

Figure 5.6 shows the throughput measurement of the CNV network for different configurations of parallelisation parameters. Like previously the optimal throughput of the CNV will be determined by the highest folding factor of the network which is the pipelined version. The non-pipelined throughput is determined by the calculated latency. The figure uses linear scales for both the x and y-axis. On the x-axis is the throughput in images/s from low to high and on the y-axis is the amount of parallelisation parameters from low to high used for that configuration. From Figure 5.6 can be observed that the optimal throughput has already been reached with less than 700 parallelisation parameters. This is a consequence of the fact that the slowest layer of the CNV network no longer can be optimised because of certain bottlenecks with FINN and the layer itself. This layer could no longer be parallelised, because otherwise the Equation (4.3) would no longer be true.

Figure 5.7 is used to further enhance where this bottleneck comes from. Each of different layers that has been implemented using the FINN-hls library. The figure is linear for the x-axis and is logarithmic for the y-axis. On the y-axis the the expected cycles per layer, which is in this case also the total folding factor of that layer. On the x-axis is the amount of parallelisation parameters used for all layers combined. As can been seen in this figure from left to right the more layers are being optimised the lower, the total expected cycles of all the layers will become. However as also can be seen in the figure the layer "ConvolutionInputGenerator_0" has for all the different configurations the same amount of expected cycles. This layer corresponds to the first convolutional layer in the CNV network and the input of this layer is the CIFAR-10 image. The architecture can be seen in Figure 3.4. This shows that for the layer "ConvolutionInputGenerator_0" the input is "3@32x32". In turn this means that input feature maps or IFM for this layer is 3, which consequently means that Equation (4.3) only a maximum of SIMD = 3 can be assigned to this layer. Resulting in that this layer no longer can be optimised and is a bottleneck for the entire design. This means that the input of a CNN network can have a big impact on how low the latency can for the network. From Figure 3.2 it can be seen the amount of channels for these images is also 3, meaning that this

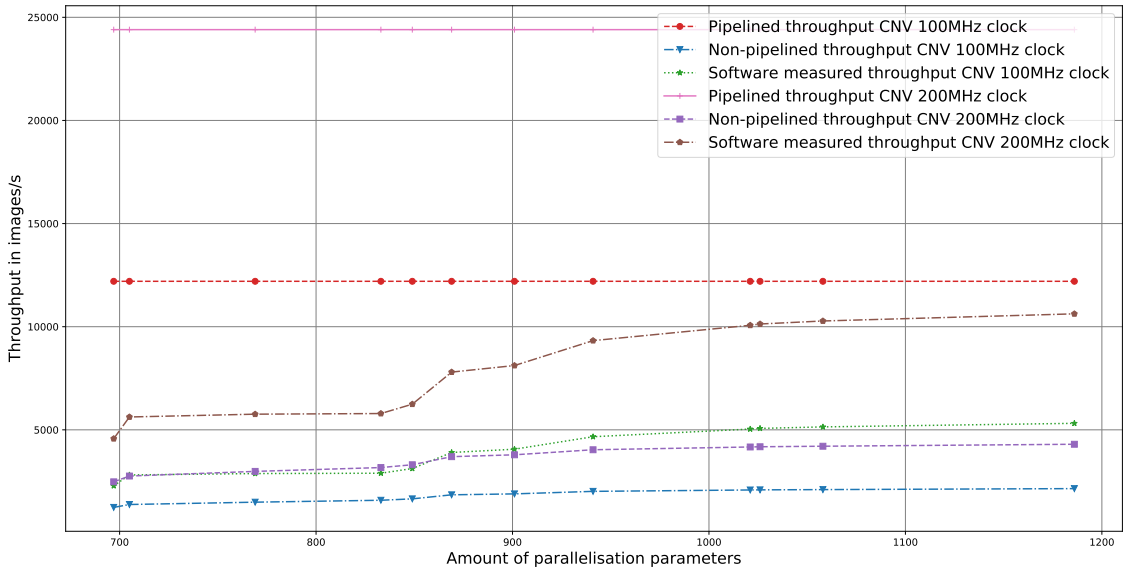


Figure 5.6: CNV software emulated throughput compared to calculated fully pipelined and non-pipelined throughput for different parallelisation parameters.

is also a bottlenecks where the researcher needs to design around for the future neural network.

As final timing analysis experiment is to discover possible bottlenecks with FINN or the neural networks for hardware measured latency. For this both neural networks are being compared. For the TFC network two different configurations have been designed. The basic configuration of FINN for the parallelisation parameters and a configuration where the minimum amount of parallelisation parameter is used. Furthermore different weights and activations have been tested to see how they can affect the latency. Figure 5.8 shows on the logarithmic y-axis the run time in ms of the different networks and on the linear x-axis the corresponding batch size of that run time. This experiment has been done on the Pynq-Z1 board with a frequency of 100 MHz. The figure shows while the TFC network has different weights, it hardly shows an effect on the latency of the neural network. This could be because of how small the TFC network is or because the difference in size in weights is not big enough to become a bottleneck in the design.

5.2.2 Resource analysis

Following the timing analysis a small resource analysis is done to find out if the the configuration that passed the 0.1 ms latency baseline can be implemented using the currently supported boards of FINN and how much different weight and activation sizes influence the resources used.

First as explained in Section 4.3.2, the amount of LUT used with different parallelisation parameters will evaluated. In Figure 5.9 this is done in comparison with the corresponding software emulated LUT resources used. This figure shows on the logarithmic y-axis the amount of LUT resources needed and on the linear x-axis the amount of parallelisation parameters used for that configuration. In the upper left of this image which is the zoomed in part of the HLS resources used, it is visible that there are two small dips in resource usage, while increasing the amount of parallelisation parameters. This happens when the PE and SIMD, together are less balanced, resulting in less resources needed. However for these configuration in particular do have a higher latency, making it a compromise between latency and resources used for different layers. This means that optimising the future neural network will be a challenging task as more parallelisation parameters does not always mean more resources used. It also matters on which layer these parameters are used and how the different parallelisation parameters are balanced.

Table 5.1 provides a further in depth overview of how different weight and activation sizes

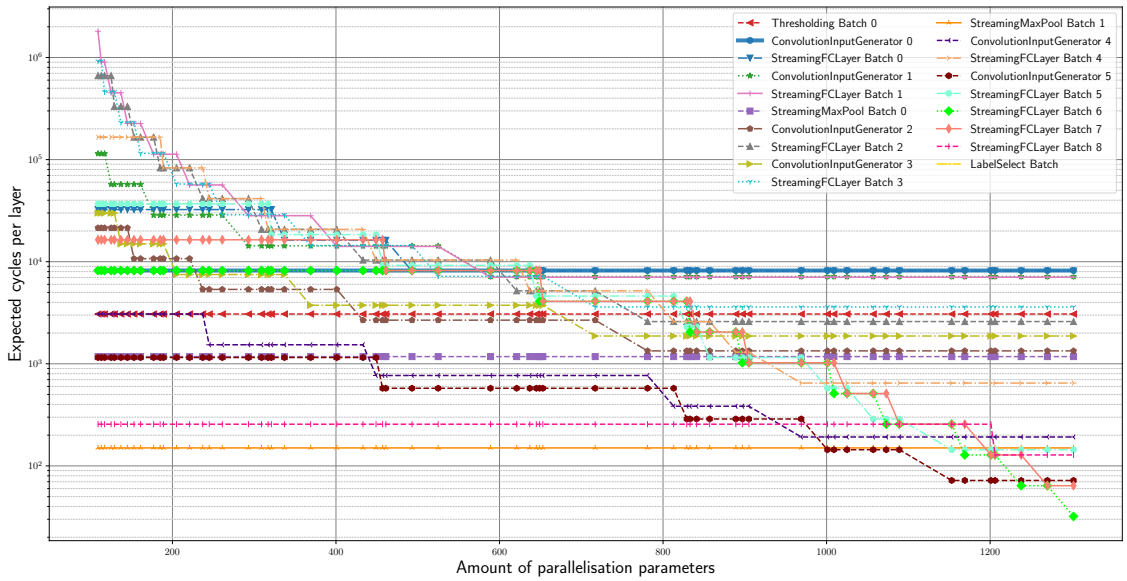


Figure 5.7: CNV layer bottlenecks.

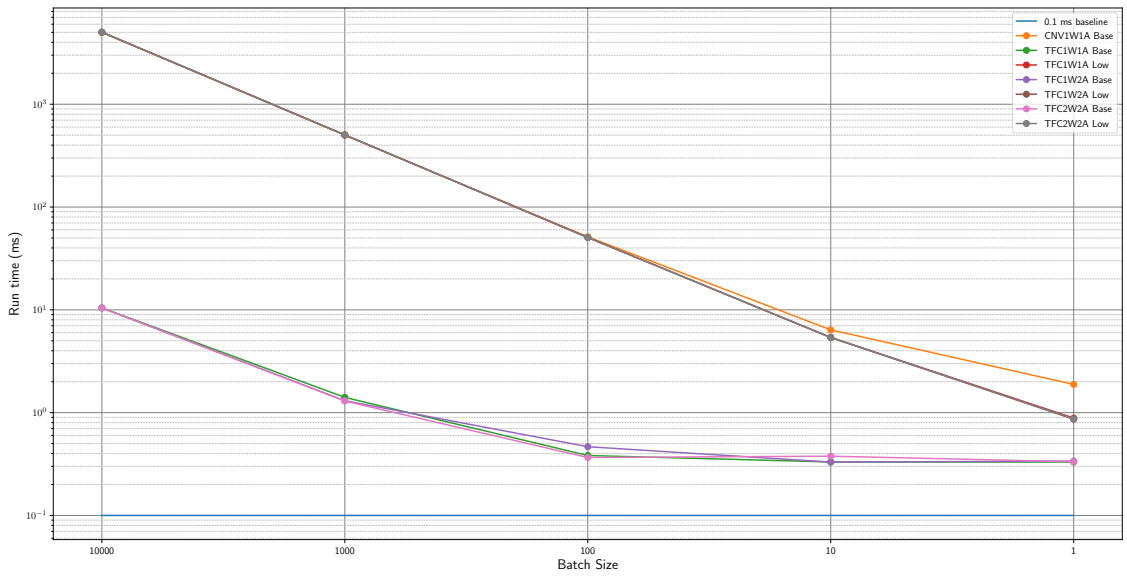


Figure 5.8: Hardware generated examples and differences batch sizes.

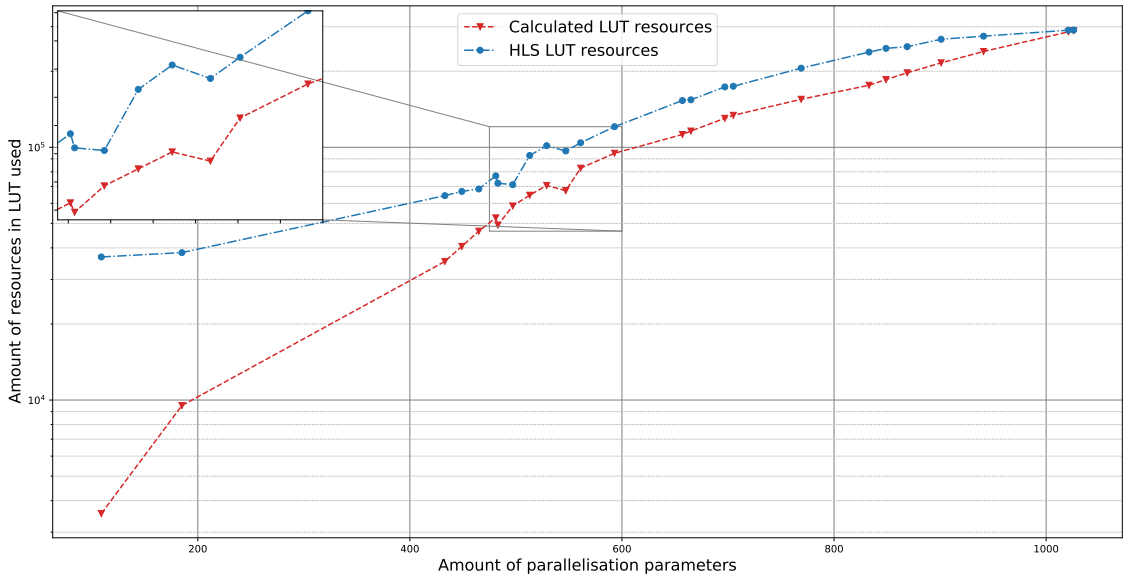


Figure 5.9: CNV1W1A LUT resource estimation.

influence the resources used for a neural network. This table shows that the higher the weight and activation sizes, the more accuracy these networks will have. However this table also shows that comes at the cost of needing more resources. The results in Table 5.1 all use the same configuration for parallelisation parameters and was build for the ZCU104 board. This configuration was the first configuration that passed the 0.1 ms requirement. From this table can also be seen that depending on which board is chosen for building the HLS, the amount of resources used by Vivado can also change as in this table the resources are higher that the ones can be seen in Figure 5.9.

CNV Network	LUT	BRAM	FF	URAM	DSP Slices	Accuracy
CNV1W1A	306717	12	24068	0	0	84.22%
CNV1W2A	353626	0	52287	0	0	87.80%
CNV2W2A	384676	0	68443	0	0	89.03%

Table 5.1: Resources used different CNV weight sizes and activation sizes.

Furthermore from Table 5.1 can be observed, that depending on the available boards, only a certain accuracy can be reached which can fit the design. A clear view of what this means can be seen in Table 5.2. This table shows how much of the resources needs to be utilised in order to be implemented on the board. From this can be concluded that if more than 100% is needed, then this particular network configuration cannot be implemented on that board. This shows from the current available boards only the Alveo U280 will be able to fit the entire design of the configuration that passed the 0.1 ms baseline.

5.3 Discussion

For creating a CNN model that is able to provide an acceptable accuracy level while reaching a low latency of 0.1 ms is not trivial tasks. This is because of the enormous design space that can be explored for the neural network and all the different configurations that can be done using FINN on this network. Figure 5.5 has shown that it is possible to design a CNN accelerator that is able to obtain results every 0.1 ms with a clock of 200 MHz.

To obtain this result, it was necessary to work around the limitations and bottlenecks of FINN and how FINN implements CNNs. As the timing analysis has shown that simply adding more

CNV Network	FPGA Board	LUT	BRAM	FF	URAM	DSP Slices
CNV1W1A	Pynq-Z1/Z2	576.54%	8.57%	22.62%	0%	0%
	Ultra96	434.69%	5.56%	17.05%	0%	0%
	ZCU104	133.12%	2.21%	5.22%	0%	0%
	ZC706	140.31%	2.20%	5.51%	0%	0%
	Alveo U280	23.52%	0.60%	0.92%	0%	0%
CNV1W2A	Pynq-Z1/Z2	664.71%	0%	49.14%	0%	0%
	Ultra96	501.17%	0%	37.05%	0%	0%
	ZCU104	153.48%	0%	11.35%	0%	0%
	ZC706	161.77%	0%	11.96%	0%	0%
	Alveo U280	27.12%	0%	2.01%	0%	0%
CNV2W2A	Pynq-Z1/Z2	723.08%	0%	64.33%	0%	0%
	Ultra96	545.18%	0%	48.50%	0%	0%
	ZCU104	166.96%	0%	14.85%	0%	0%
	ZC706	175.97%	0%	15.65%	0%	0%
	Alveo U280	29.50%	0%	2.63%	0%	0%

Table 5.2: Resources utilisation percentage on different board for first passing CNV1W1A configuration.

parallelisation parameters does not mean the neural network will accelerate more. The analysis further presented that each layer has a limitation to how much they can be accelerated. This is partly because of how FINN is implemented and partly because of how much the layer can be parallelised in the first place. For example the first convolutional layer of the neural network can have significant impact on the latency as this layer will always be limited by the input of the neural network, which cannot be changed.

Finally implementing the configuration that can pas the required 0.1 ms baseline on actual hardware is also no trivial task. As for the CNV-network the configuration that has passed this baseline is at the current moment only able to fit on the Alveo Boards, because of the amount of LUT it needs. This is still part of the design space that can be further explored on how much effect larger weight and activation bits sizes have on the accuracy and resources used. Additionally FINN is able distribute the resources more than is currently shown in Table 5.2. FINN has for example the ability to store the weight and activations in BRAM instead of LUT. Making the ZCU104 a possibility for a design that can fit the 0.1 ms configurations. Figure 5.9 has also shown that the parallelisation parameters can also be further explored with how they effect the neural network in terms of latency and resources used.

Chapter 6

Conclusions and recommendations

This final chapter consists of the conclusions of this thesis and reflects back on the original research question. At the end there is a discussion about possible recommendations and possible avenues of future development of the pipeline of the future gravitational waves interferometer control system.

6.1 Conclusions

By using the appropriate hardware fabrics together with their respective frameworks, it is possible to further the development of the low latency pipeline of a neural network for a future gravitational waves interferometer control system. The control system needs to keep free-falling test masses (suspended mirrors) of the Fabry-Pérot resonance cavities in the correct alignment. By having a laser go through a specific setup of mirrors, the near and far field distribution can be detected by special cameras. From the near and far field distribution images together with a neural network, the control system will be able to determine if there is misalignment of the mirrors and adjust them accordingly. For the neural network a Convolutional Neural Network (CNN) will be used, because the most promising approach for image focused neural networks are CNNs

For creating a CNN model that is able to provide an acceptable accuracy, while reaching a low latency of 0.1 ms is not a trivial task. This is because of the enormous design space that can be explored with neural networks and all the different configurations that can be done. To meet the requirements, an FPGA was chosen as hardware platform. This is because with an FPGA the neural network can be highly customised per layer to obtain the lowest latency, while having the reconfigurability for ease of development. Together with an FPGA, FINN is used to develop the low latency pipeline. FINN generates dataflow-style architectures, to have a fully pipelined neural networks. FINN uses parallelisation parameters to optimise each layer separately. From these parallelisation parameters the total folding factor of each layer can be determined, which are used in the analysis of the neural networks.

To explore the large design space of low latency inference deep learning neural networks, the pretrained neural networks available for testing and analysing the FINN framework have been used. From the pretrained neural networks, the TFC and CNV neural networks have been further investigated. The TFC network because FINN was able to reach a latency off 0.31 μs with a variant of this network. The CNV network because it is used for deep learning image classification problems with multiple convolutional layers. These neural networks will help discover possible bottlenecks for achieving the low latency of 0.1 ms with FINN. Both pretrained neural network can be optimised in term of accuracy, resources and latency. The most important requirement of the neural network is to reach a latency of 0.1 ms or lower. For that reason, the first analysis on the neural networks is a timing analysis. This is followed up by the resource analysis to see if this design is feasible on available boards.

The timing analysis starts with determining the total folding factors per layer for different configurations and from that the total amount of expected cycles used in the whole neural network.

For the TFC network, it became clear that the lowest total amount of expected cycles, would not necessary provide the lowest possible latency of the network. This can be observed that for one specific configuration the emulated latency goes down, while the calculated latency which is derived from total amount of expected cycles goes up. There are two reasons as to why this development can occur. The first reason is that this configuration is not been fully optimised as indicated from the strategy of achieving the lowest latency based on the folding factors. Another reason is that in this final configuration the resources are more divided over all the layers. In that case the communication that is between each of the the layers can be better optimised and lead to an overall lower latency. Meaning that while each individual layer overall is slower, the communication between layers is faster, lowering the total latency of the network. In case the TFC network could not reach the required latency, it would be deemed impossible to develop the required low latency pipeline using FINN. For the TFC network, it was possible to design a pipeline that could achieve the 0.1 ms requirement.

Subsequently, a timing analysis has been done on the CNV network to discover if a CNN could reach the 0.1 ms requirement with FINN. By optimising the parallelisation parameters and increasing the clock frequency, it was possible to create a configuration that was able to pass the 0.1 ms requirement for a CNN. This was reached by further optimising all layers, because the highest total folding factor of a specific layer could not go lower. For this reason, the bottlenecks of the CNV have been investigated to see where the latency possibly could be limited. The bottlenecks for the latency come from how much the layer can be parallelised in the first place to accelerate the pipeline. Every layer of the neural network is limited in its parallelisation by the input and output of the layer. For the CNV network and for the future gravitational waves interferometer control system neural network, this limitation will come from the input of the neural network. This is because the input cannot be changed in such a way to make the layer more parallel and accelerate the entire pipeline.

For the CNV network configuration that was able to pass the 0.1 ms requirements, a resource analysis was done to see if this design could fit on available hardware. The analysis shows that the more accuracy is required from the neural network, more resources would be needed to generate this network. Currently only the Alveo boards are able to fit the design that can pass the 0.1 ms requirement. In case different hardware options want to be used, the configuration needs to be developed more using the available resources in mind.

6.2 Recommendations

There are multiple avenues that can be taken for the future development of the low latency neural network using FINN. The first recommendation is further development on how different weights and activation bits for quantized neural networks affect the accuracy. This can be done together with a resources analysis and investigate if the increase in size of the weight and activation bits increase the latency of the network.

Currently in the FINN framework, Vivado decides if the weights and activation bits of the layers would be stored in LUT or BRAM. As was discovered in the resource analysis the LUT resource utilisation for different boards exceeds the available resources for almost every board. Meanwhile the BRAM utilisation is almost always close to zero percent. Follow up research can look at how the LUT and BRAM utilisation can be balanced more, by changing where FINN and Vivado will store the weights and activation bits of the layers

The third recommendation for follow up research is in regards to power measurements. Once a neural network has been developed that can achieve an acceptable accuracy with FINN, different hardware configurations of the neural network can be generated and see how they affect the power on different FPGAs.

The final recommendation is the further development of the interface from camera to neural network, together with the control system. This thesis has made some initial analysis on this subject, but as the development on the neural network is still ongoing, the research on this topic is not finished.

Bibliography

[1] I. Newton, *Philosophiae naturalis principia mathematica.* William Dawson & Sons Ltd., London, 1687.

[2] A. Einstein, "Die grundlage der allgemeinen relativitätstheorie," *Annalen der Physik*, vol. 354, no. 7, pp. 769–822, 1916.

[3] B. P. Abbott, R. Abbott, T. Abbott, M. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. Adhikari *et al.*, "Observation of gravitational waves from a binary black hole merger," *Physical review letters*, vol. 116, no. 6, p. 061102, 2016.

[4] "Virgo website." [Online]. Available: http://www.virgo-gw.eu/

[5] J. C. Diaz, *Control of the gravitational wave interferometric detector Advanced Virgo.* Springer, 2018.

[6] *MicroLabBox Hardware*, [Online], Available: http://www.dspace.com/en/ltd/home/products/hw/microlabbox.cfm#145_23644, Accessed: 05 March 2020.

[7] J. D. Creighton and W. G. Anderson, *Gravitational-wave physics and astronomy: An introduction to theory, experiment and data analysis.* John Wiley & Sons, 2012.

[8] R. M. Wald, *General relativity.* University of Chicago press, 2010.

[9] J. Weber, *General Relativity and gravitational waves.* Courier Corporation, 2004.

[10] B. F. Schutz and F. Ricci, "Gravitational waves, sources, and detectors," *arXiv preprint arXiv:1005.4735*, 2010.

[11] K. Riles, "Gravitational waves: Sources, detectors and searches," *Progress in Particle and Nuclear Physics*, vol. 68, pp. 1–54, 2013.

[12] B. C. Barish, "The Science and Detection of Gravitational Waves," *Brazilian Journal of Physics*, vol. 32, pp. 831 – 837, 12 2002. [Online]. Available: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-97332002000500003&nrm=iso

[13] *Two Black Holes Merge into One*, [Online], Available: https://www.ligo.caltech.edu/image/ligo20160211d, Accessed: 04 March 2020.

[14] A. Abramovici, W. E. Althouse, R. W. Drever, Y. Gürsel, S. Kawamura, F. J. Raab, D. Shoemaker, L. Sievers, R. E. Spero, K. S. Thorne *et al.*, "Ligo: The laser interferometer gravitational-wave observatory," *science*, vol. 256, no. 5055, pp. 325–333, 1992.

[15] C. Bradaschia, R. Del Fabbro, A. Di Virgilio, A. Giazotto, H. Kautzky, V. Montelatici, D. Passuello, A. Brillet, O. Cregut, P. Hello *et al.*, "The virgo project: a wide band antenna for gravitational wave detection," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 289, no. 3, pp. 518–525, 1990.

[16] C. Fabry, "Theorie et applications d'une nouvelle methods de spectroscopie intereferentielle," *Ann. Chim. Ser. 7*, vol. 16, pp. 115–144, 1899.

[17] A. A. Michelson and E. W. Morley, "Lviii. on the relative motion of the earth and the luminiferous æther," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 24, no. 151, pp. 449–463, 1887.

[18] *LIGO's Interferometer*, [Online], Available: https://www.ligo.caltech.edu/page/ligos-ifo, Accessed: 04 March 2020.

[19] M. Granata, *Optical development for second-and third-generation gravitational-wave detectors: stable recycling cavities for advanced virgo and higher-order Laguerre-Gauss modes*, 2011.

[20] C. Bond, D. Brown, A. Freise, and K. A. Strain, "Interferometer techniques for gravitational-wave detection," *Living reviews in relativity*, vol. 19, no. 1, p. 3, 2016.

[21] M. Nickerson, "A review of pound-drever-hall laser frequency locking," *JILA, University of Colorado and Nist*, 2019.

[22] R. Drever, J. L. Hall, F. Kowalski, J. Hough, G. Ford, A. Munley, and H. Ward, "Laser phase and frequency stabilization using an optical resonator," *Applied Physics B*, vol. 31, no. 2, pp. 97–105, 1983.

[23] M. Razzano and E. Cuoco, "Image-based deep learning for classification of noise transients in gravitational wave detectors," *Classical and Quantum Gravity*, vol. 35, no. 9, p. 095016, 2018.

[24] R. Biswas, L. Blackburn, J. Cao, R. Essick, K. A. Hodge, E. Katsavounidis, K. Kim, Y.-M. Kim, E.-O. Le Bigot, C.-H. Lee *et al.*, "Application of machine learning algorithms to the study of noise artifacts in gravitational-wave data," *Physical Review D*, vol. 88, no. 6, p. 062003, 2013.

[25] D. George and E. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," *Physical Review D*, vol. 97, no. 4, p. 044039, 2018.

[26] K. Abdelouahab, "Reconfigurable hardware acceleration of cnns on fpga-based smart cameras," Ph.D. dissertation, Clermont Auvergne, 2018.

[27] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *CoRR*, vol. abs/1703.09039, 2017. [Online]. Available: http://arxiv.org/abs/1703.09039

[28] 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.

[29] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[30] A. Bordes, X. Glorot, J. Weston, and Y. Bengio, "Joint learning of words and meaning representations for open-text semantic parsing," in *Artificial Intelligence and Statistics*, 2012, pp. 127–135.

[31] A. Karpathy, "What i learned from competing against a convnet on imagenet," *Andrej Karpathy Blog*, vol. 5, pp. 1–15, 2014.

[32] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[34] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[35] A. Karpathy, *Neural Networks Part 1: Setting up the Architecture*, [Online], Available: https://cs231n.github.io/neural-networks-1/, Accessed: 24 Mar 2020.

[36] The MathWorks, Inc, *Deep Learning Toolbox*, [Online], Available: https://www.mathworks.com/products/deep-learning.html, Accessed: 24 Mar 2020.

[37] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[39] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.

[40] F. Chollet *et al.*, "Keras: Deep learning library for theano and tensorflow," *URL: https://keras. io/k*, vol. 7, no. 8, p. T1, 2015.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[42] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[43] A. Karpathy, *Convolutional Neural Networks (CNNs / ConvNets)*, [Online], Available: http://cs231n.github.io/convolutional-networks/, Accessed: 24 Mar 2020.

[44] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.

[45] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li, "Computer vision algorithms and hardware implementations: A survey," *Integration*, 2019.

[46] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *CoRR*, vol. abs/1901.00121, 2019. [Online]. Available: http://arxiv.org/abs/1901.00121

[47] X. Zhang, A. Ramachandran, C. Zhuge, D. He, W. Zuo, Z. Cheng, K. Rupnow, and D. Chen, "Machine learning on fpgas to face the iot revolution," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 894–901.

[48] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 15–24.

[49] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[50] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "Tgpa: tile-grained pipeline architecture for low latency cnn inference," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[51] D. J. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "High performance binary neural networks on the xeon+ fpga™ platform," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–4.

[52] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. A. Vissers, J. Wawrzynek, and K. Keutzer, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas," *CoRR*, vol. abs/1811.08634, 2018. [Online]. Available: http://arxiv.org/abs/1811.08634

[53] T. Wang, C. Wang, X. Zhou, and H. Chen, "A survey of fpga based deep learning accelerators: Challenges and opportunities," *arXiv preprint arXiv:1901.04988*, pp. 1–10, 2018.

[54] K. Abdelouahab, M. Pelcat, J. Sérot, and F. Berry, "Accelerating CNN inference on fpgas: A survey," *CoRR*, vol. abs/1806.01683, 2018. [Online]. Available: http://arxiv.org/abs/1806.01683

[55] G. B. Hacene, C. E. R. K. Lassance, V. Gripon, M. Courbariaux, and Y. Bengio, "Attention based pruning for shift networks," *CoRR*, vol. abs/1905.12300, 2019. [Online]. Available: http://arxiv.org/abs/1905.12300

[56] S. I. Venieris, A. Kouris, and C. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *CoRR*, vol. abs/1803.05900, 2018. [Online]. Available: http://arxiv.org/abs/1803.05900

[57] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

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

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[60] S. Ghaffari and S. Sharifian, "Fpga-based convolutional neural network accelerator design using high level synthesize," in *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*. IEEE, 2016, pp. 1–6.

[61] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[62] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[63] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An fpga-based cnn accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, no. 3, p. 281, Mar 2019. [Online]. Available: http://dx.doi.org/10.3390/electronics8030281

[64] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[65] Z. Yan, X. Li, M. Li, W. Zuo, and S. Shan, "Shift-net: Image inpainting via deep feature rearrangement," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[66] L. Bai, Y. Zhao, and X. Huang, "A cnn accelerator on fpga using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[67] Y. He, X. Liu, H. Zhong, and Y. Ma, "Addressnet: Shift-based primitives for efficient convolutional neural networks," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019, pp. 1213–1222.

[68] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[69] J. Faraone, M. Kumm, M. Hardieck, P. Zipf, X. Liu, D. Boland, and P. H. W. Leong, "Addnet: Deep neural networks using fpga-optimized multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 115–128, 2020.

[70] E. Delaye, A. Sirasao, C. Dudha, and S. Das, "Deep learning challenges and solutions with xilinx fpgas," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 908–913.

[71] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, Feb. 2017. [Online]. Available: https://doi.org/10.1145/3005348

[72] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 75–84.

[73] Y. Umuroglu and M. Jahre, "Streamlined deployment for quantized neural networks," *arXiv preprint arXiv:1709.04060*, 2017.

[74] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference." in *ICCV Workshops*, 2019, pp. 3009–3018.

[75] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[76] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 77–84.

[77] Y. Li, L. Zichuan, K. Xu, H. Yu, and F. Ren, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks," *FPGA '17: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 02 2017.

[78] Y. Zhou, S. Redkar, and X. Huang, "Deep learning binary neural network on an fpga," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2017, pp. 281–284.

[79] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[80] 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*, 2017, pp. 65–74.

[81] Advanced Micro Devices, Inc, *AMD Ryzen™ Threadripper™ 3990X Processor*, Jan 2020, [Online], Available: https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3990x, Accessed: 25 Mar 2020.

[82] A. Rush, A. Sirasao, and M. Ignatowski, "Unified deep learning with cpu gpu and fpga technologies," *White paper, AMD and Xilinx*, 2017.

[83] 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," *arXiv preprint arXiv:1602.02830*, 2016.

[84] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks," in *FPGA*, 2017, pp. 290–291.

[85] S. Ren, K. Bertels, and Z. Al-Ars, "Efficient acceleration of the pair-hmms forward algorithm for gatk haplotypecaller on graphics processing units," *Evolutionary bioinformatics*, vol. 14, 2018.

[86] N. Ahmed, J. Lévy, S. Ren, H. Mushtaq, K. Bertels, and Z. Al-Ars, "Gasal2: a gpu accelerated sequence alignment library for high-throughput ngs data," *BMC Bioinformatics*, vol. 20, no. 1, p. 520, Oct 2019. [Online]. Available: https://doi.org/10.1186/s12859-019-3086-9

[87] S. Ren, N. Ahmed, K. Bertels, and Z. Al-Ars, "Gpu accelerated sequence alignment with traceback for gatk haplotypecaller," *BMC Genomics*, vol. 20, no. 2, p. 184, Apr 2019. [Online]. Available: https://doi.org/10.1186/s12864-019-5468-9

[88] NVIDIA, *RTX. IT'S ON. GEFORCE RTX 2080 Ti*, 2019, [Online], Available: https://www.nvidia.com/nl-nl/geforce/graphics-cards/rtx-2080-ti/, Accessed: 25 Mar 2020.

[89] C. Murphy and Y. Fu, *Xilinx All Programmable Devices: A Superior Platform for Compute-Intensive Systems*, 2017, [Online], Available: https://www.xilinx.com/support/documentation/white{_}papers/wp492-compute-intensive-sys.pdf, Accessed: 3 Apr 2020.

[90] X. Mei and X. Chu, "Dissecting gpu memory hierarchy through microbenchmarking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 72–86, 2017.

[91] C. Nvidia, "Nvidia cuda programming guide (version 1.0)," *NVIDIA: Santa Clara, CA*, 2007.

[92] M. Amarís, R. Y. de Camargo, M. Dyab, A. Goldman, and D. Trystram, "A comparison of gpu execution time prediction using machine learning and analytical modeling," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, 2016, pp. 326–333.

[93] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, Dec. 2018. [Online]. Available: https://doi.org/10.1145/3242898

[94] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[95] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 45–54.

[96] J. Fang, J. Chen, J. Lee, Z. Al-Ars, and H. P. Hofstee, "An efficient high-throughput lz77-based decompressor in reconfigurable logic," *Journal of Signal Processing Systems*, vol. 92, no. 9, pp. 931–947, Sep 2020. [Online]. Available: https://doi.org/10.1007/s11265-020-01547-w

[97] J. Hoozemans, R. de Jong, S. van der Vlugt, J. Van Straten, U. K. Elango, and Z. Al-Ars, "Frame-based programming, stream-based processing for medical image processing applications," *Journal of Signal Processing Systems*, vol. 91, no. 1, pp. 47–59, Jan 2019. [Online]. Available: https://doi.org/10.1007/s11265-018-1422-3

[98] E. Houtgast, V. Sima, and Z. Al-Ars, "High performance streaming smith-waterman implementation with implicit synchronization on intel fpga using opencl," in *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, 2017, pp. 492–496.

[99] G. Smaragdos, G. Chatzikonstantis, R. Kukreja, H. Sidiropoulos, D. Rodopoulos, I. Sourdis, Z. Al-Ars, C. Kachris, D. Soudris, C. I. D. Zeeuw, and C. Strydis, "BrainFrame: a node-level heterogeneous accelerator platform for neuron simulations," *Journal of Neural Engineering*, vol. 14, no. 6, p. 066008, nov 2017. [Online]. Available: https://doi.org/10.1088/1741-2552/aa7fc5

[100] Xilinx, Inc, *What is an FPGA?*, 2019, [Online], Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html, Accessed: 25 Mar 2020.

[101] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 152–159.

[102] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170.

[103] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[104] Falcon Computing Solutions, *MERLIN COMPILER*, [Online], Available: https://www.falconcomputing.com/merlin-fpga-compiler/, Accessed: 24 Mar 2020.

[105] Intel Corporation, *Intel® FPGA SDK for OpenCL™ Software Technology*, [Online], Available: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html, Accessed: 24 Mar 2020.

[106] Xilinx, Inc, *The Xilinx SDAccel development environment: Bringing the best performance/watt to the data center*, 2014, [Online], Available: https://www.xilinx.com/support/documentation/backgrounders/sdaccel-backgrounder.pdf, Accessed: 24 Mar 2020.

[107] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, and D. Qian, "The deep learning compiler: A comprehensive survey," *arXiv preprint arXiv:2002.03794*, 2020.

[108] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[109] Y. Ma, M. Kim, Y. Cao, S. Vrudhula, and J.-s. Seo, "End-to-end scalable fpga accelerator for deep residual networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[110] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–8.

[111] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J.-s. Seo, "Alamo: Fpga acceleration of deep learning algorithms with a modularized rtl compiler," *Integration*, vol. 62, pp. 14–23, 2018.

[112] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto customized hardware," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 24–29.

[113] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[114] Z. Liu, Y. Dou, J. Jiang, and J. Xu, "Automatic code generation of convolutional neural networks in fpga implementation," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 61–68.

[115] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: automatic generation of fpga-based learning accelerators for the neural network family," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[116] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[117] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to fpga acceleration," in *the Workshop on Cognitive Architectures*, 2016.

[118] H. Zeng, C. Zhang, and V. Prasanna, "Fast generation of high throughput customized deep learning accelerators on fpgas," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–8.

[119] H. Zeng, R. Chen, and V. K. Prasanna, "Optimizing frequency domain implementation of cnns on fpgas," *University of Southern California, Tech. Rep*, 2017.

[120] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 35–44.

[121] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A framework for generating high throughput cnn implementations on fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 117–126.

[122] N. J. Fraser, Y. Umuroglu, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Scaling binarized neural networks on reconfigurable logic," in *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, 2017, pp. 25–30.

[123] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott, "FINN-L: library extensions and design trade-off analysis for variable precision LSTM networks on fpgas," *CoRR*, vol. abs/1807.04093, 2018. [Online]. Available: http://arxiv.org/abs/1807.04093

[124] 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," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[125] Xilinx, Inc, *FINN*, 2020, [Online], Available: https://finn.readthedocs.io/en/latest/index.html, Accessed: 25 Apr 2020.

[126] S. I. Venieris and C. Bouganis, "fpgaconvnet: A toolflow for mapping diverse convolutional neural networks on embedded fpgas," *CoRR*, vol. abs/1711.08740, 2017. [Online]. Available: http://arxiv.org/abs/1711.08740

[127] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 40–47, 2016.

[128] ——, "fpgaconvnet: Automated mapping of convolutional neural networks on fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 291–292.

[129] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, and F. Berry, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, pp. 1–4, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8015156/

[130] A. X. M. Chang, A. Zaidy, V. Gokhale, and E. Culurciello, "Compiling deep learning models for custom hardware accelerators," *arXiv preprint arXiv:1708.00117*, 2017.

[131] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[132] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[133] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "Tabla: A unified template-based framework for accelerating statistical machine learning," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 14–26.

[134] Xilinx, Inc, *Vitis Unified Software Platform*, 2019, [Online], Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html, Accessed: 24 Mar 2020.

[135] V. Kathail, "Xilinx vitis unified software platform," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 173–174. [Online]. Available: https://doi.org/10.1145/3373087.3375887

[136] X. Yu, Y. Wang, J. Miao, E. Wu, H. Zhang, Y. Meng, B. Zhang, B. Min, D. Chen, and J. Gao, "A data-center fpga acceleration platform for convolutional neural networks," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 151–158.

[137] E. Nurvitadhi, Jaewoong Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[138] M. Blott, N. 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," *IEEE Transactions on Computers*, 2020.

[139] H. Frazier, "The 802.3 z gigabit ethernet standard," *Ieee network*, vol. 12, no. 3, pp. 6–7, 1998.

[140] J. Axelson, "Usb complete: Everything you need to develop usb peripherals," *Lakeview Research*, 2005.

[141] K. Tindell, H. Hanssmon, and A. J. Wellings, "Analysing real-time communications: Controller area network (can)." in *RTSS*. Citeseer, 1994, pp. 259–263.

[142] F. Dehmelt, "Performance of lvds with different cables," *Analog Applications*, 2000.

[143] I. HP *et al.*, "Universal serial bus 3.0 specification," 2008.

[144] Xilinx, Inc, *Xilinx Vitis AI Model Zoo*, 2019, [Online], Available: https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo, Accessed: 18 Dec 2020.

[145] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana, and M. Blott, "Memory-efficient dataflow inference for deep cnns on fpga," *arXiv preprint arXiv:2011.07317*, 2020.

[146] Y. Umuroglu, *PyVerilator*, November 2020, [Online], Available: https://github.com/maltanar/pyverilator, Accessed: 22 Jan 2021.

[147] Xilinx, Inc, *FINN*, 2020, [Online], Available: https://finn.readthedocs.io/en/latest/getting_started.html, Accessed: 25 Apr 2020.

[148] Xilinx, *PYNQ-Z1 Overlays)*, 2018, [Online], Available: https://pynq.readthedocs.io/en/latest/pynq_overlays/pynqz1.html, Accessed: 22 Jan 2021.

[149] ——, *PYNQ-Z2 Overlays)*, 2018, [Online], Available: https://pynq.readthedocs.io/en/latest/pynq_overlays/pynqz2.html, Accessed: 22 Jan 2021.

[150] *Zynq-7000 SoC Data Sheet: Overview*, Xilinx, July 2018, [Online], Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, Accessed: 22 Jan 2021.

[151] Xilinx, *Zynq UltraScale+ MPSoC*, 2018, [Online], Available: https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf, Accessed: 22 Jan 2021.

[152] L. Limited, *Ultra96*, 2021, [Online], Available: https://www.96boards.org/product/ultra96/, Accessed: 22 Jan 2021.

[153] Xilinx, *Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit*, [Online], Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#hardware, Accessed: 22 Jan 2021.

[154] ——, *Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit*, [Online], Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html#hardware, Accessed: 22 Jan 2021.

[155] ——, *Alveo U50 Data Center Accelerator Card*, [Online], Available: https://www.xilinx.com/products/boards-and-kits/alveo/u50.html#specifications, Accessed: 22 Jan 2021.

[156] P. kennedy, *Xilinx Alveo U50 FPGA Card for Data Center Acceleration*, August 2019, [Online], Available: https://www.servethehome.com/xilinx-alveo-u50-fpga-card-for-data-center-acceleration/, Accessed: 22 Jan 2021.

[157] Xilinx, *Alveo U200 and U250 Data Center Accelerator Cards Data Sheet*, May 2020, [Online], Available: https://www.xilinx.com/support/documentation/data_sheets/ds962-u200-u250.pdf, Accessed: 22 Jan 2021.

[158] ——, *Alveo U280 Data Center Accelerator Card*, [Online], Available: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html#specifications, Accessed: 22 Jan 2021.

[159] ——, *Alveo U250 Data Center Accelerator Card*, [Online], Available: https://www.xilinx.com/products/boards-and-kits/alveo/u250.html#specifications, Accessed: 22 Jan 2021.

[160] ——, *Adaptable Accelerator Cards for Data Center Workloads*, 2018, [Online], Available: https://www.xilinx.com/publications/product-briefs/alveo-product-brief.pdf, Accessed: 22 Jan 2021.

[161] Y. Umuroglu, *FINN v0.4b (beta) is released*, September 2020, [Online], Available: https://xilinx.github.io/finn/2020/09/21/finn-v04b-beta-is-released.html, Accessed: 22 Jan 2021.

Appendices

Appendix A

Measurement setup Nikhef

This appendix presents the detailed version of the experimental setup from Nikhef.

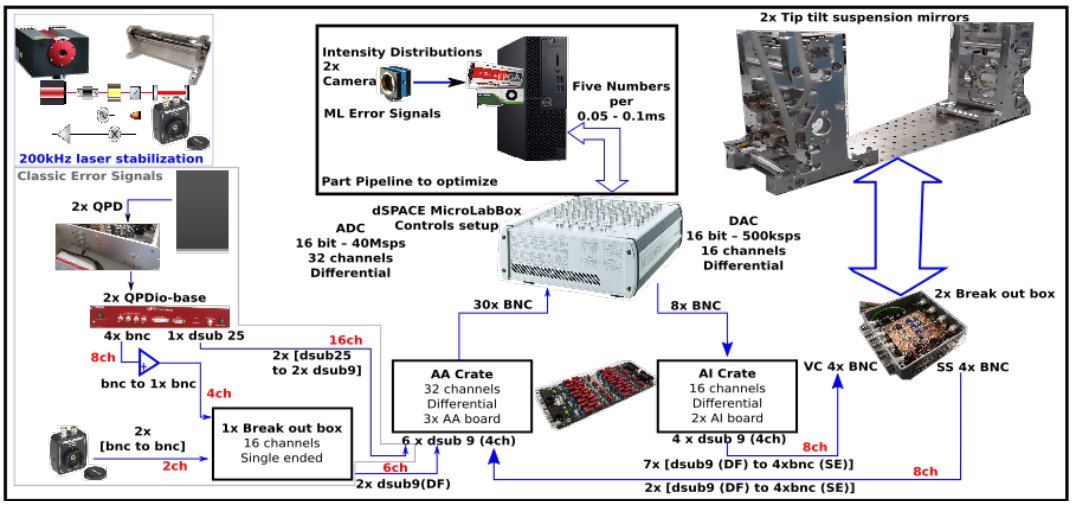


Figure A1: Detailed experimental setup suspended mirrors.