



University of Glasgow | School of
Computing Science

Deploying a neural network on a Xilinx FPGA PYNQ-Z1 board

Ashleigh Lyon

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfilment of the requirements
of the Degree of Master of Science at the University of Glasgow

9 September 2019

Abstract

Artificial Neural Networks (ANNs) can achieve human-like results in a range of speech, pattern and facial recognition applications. Unlike other algorithms, ANNs are not hard-coded to detect features and instead learn and develop from a given dataset. ANN models are designed in particular frameworks that most often utilise Graphics Processing Units for image processing. In this master thesis, the implementation of a deep neural network on a Xilinx FPGA is investigated, and a technique called quantization is utilised, which drastically cuts down the hardware consumption while maintaining acceptable accuracy.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Ashleigh Lyon

Signature: A Lyon

Acknowledgements

I would like to sincerely thank my supervisor Dr Wim Vanderbauwhede for all the help and guidance provided throughout this project. The discussions during meetings gave me great insight on how to improve and advance my work.

Would also like to thank my family and my bf for the continuous support throughout the past year.

Contents

Chapter 1	Introduction	1
1.1	Context	1
1.2	Existing Work	1
1.2.1	Under Water Image Recognition	1
1.2.2	Lung Nodule Detection	2
1.3	Project Objective	2
1.4	Document Outline	2
Chapter 2	Neural Network Architecture	3
2.1	Background	3
2.2	Neural Networks	3
2.2.1	Learning	4
2.2.2	Training	4
2.3	Convolutional Neural Network	5
2.4	Quantization	6
2.5	PYNQ-Z1 Board	7
Chapter 3	Design and Implementation	9
3.1	Training Environment	9
3.1.1	Amazon Web Services (AWS)	9
3.1.2	Software	9
3.1.3	Datasets	10
3.1.4	Training	11
3.2	Implementation on PYNQ-Z1 Board	12
Chapter 4	Evaluation	14
4.1	Speed and Accuracy Results	14
4.2	Tests	15
Chapter 5	Conclusion	17
5.1	Reflection	17
5.2	Future Works	17
References		19
Appendix A	Project Screenshots	1

Chapter 1 Introduction

1.1 Context

This report details the implementation of a deep neural network on a Xilinx PYNQ-Z1 FPGA. The report contains: the research completed as part of this project; details on the development of a trained neural network; an overview of how the network was deployed on an FPGA; and an evaluation of the project as a whole.

In today's world, artificial neural networks (ANNs) have become a popular and helpful model for classification, detection, speech, pattern and facial recognition, in many industries. Artificial neural networks are one type of model for machine learning that has become utilised in a multitude of industries ranging from finance to security, medical to engineering and in everyday real-world applications [11].

Due to the extensive use of ANNs, there is growing interest in enhancing their performance. Many hardware technologies can be used in accelerating machine learning algorithms, such as Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA). FPGA technology is a promising option for hardware acceleration [12]. Low power consumption, reconfigurability and real-time processing capability, make FPGAs particularly suitable for embedded systems (Liang, S. et al., 2018).

1.2 Existing Work

As part of the initial research for the project, I looked at several studies that explore the use of deploying neural networks on FPGAs. This allowed me to gain a greater understanding of how FPGAs are currently being used to aid and accelerate the implementation of deep learning networks across various disciplines.

1.2.1 Under Water Image Recognition

In January 2019, an article was published by Zhao et al. (2019) from Jilin University in China, outlining their work on the development of an embedded FPGA image recognition system on a Convolutional Neural Network. The system was designed to identify and analyse objects within the underwater environment. The problem they faced was the autonomous intelligent underwater vehicle (AUV), an underwater robot, could not process or analyse images captured by the camera in real-time, meaning it was unable to react fast enough to dangerous areas of the sea. Zhao et al. (2019) established that with the processing of a FPGA, AUV could systematically differentiate between driving regions like seawater and non-driving regions, such as rocks and ocean growth. When obstacles were found, AUV took steps to avoid impact. Therefore, with the acceleration of the FPGA, AUV could respond promptly to different situations [13].

1.2.2 Lung Nodule Detection

Early this year, researchers from the National University of Defence Technology in China took the acceleration of deep neural networks a step further by using not one, but multiple FPGA platforms. The focus of Shen's et al. (2019) research is aiding in lung nodule detection, which is a very effective way to identify possible cancer tissue. Convolutional neural networks have been successfully used to solve medical image analysis problems, over recent years; therefore, their focus was to accelerate 3D CNN-based lung nodule segmentation on a multi-FPGA platform. Multiple FPGAs were required as the CNNs adopted in lung nodule detection have high computational complexity, making it challenging to map the entire network on a single FPGA, developing a multi-system could, therefore, better balance the workload among FPGAs [14].

1.3 Project Objective

The overall objective of this project is to develop a thorough understanding of artificial neural networks, and the impact deploying on a FPGA has on performance compared to deploying on software. The primary objectives may be defined as follows:

1. Obtain a clear understanding of the theory and application of neural networks, with a particular focus on binarized neural networks.
2. Gain insight into software tools and libraries used in the process of deep learning.
3. Review the importance of hardware, particularly FPGAs, in the aid and acceleration of deep neural networks.
4. Set up a working training environment which successfully trains a convolutional neural network (CNN) or a large fully connected (LFC) network on a particular dataset.
5. Deploy a trained neural network on the Xilinx PYNQ-Z1 FPGA, classifying a test set of images.
6. Compare performance between hardware and software inference on the PYNQ-Z1 and evaluate the accuracy of the image classification.

1.4 Document Outline

The report is structured as follows: chapter 2 provides essential background research on neural networks with particular focus on convolutional neural networks and binarized neural networks. The Xilinx PYNQ-Z1 hardware is also introduced. Chapter 3 outlines the development of the neural network, with regards to training environment, dataset chosen and the implementation on the FPGA. Chapter 4 provides testing and an evaluation of the final network. Chapter 5 concludes the report with a brief reflection, along with limitations and a look at potential future works.

Chapter 2 Neural Network Architecture

This chapter provides the reader with background information regarding the essential topics for this thesis. First, artificial neural networks are introduced, which follow on to convolutional neural networks and finally binarized neural networks. Lastly, the FPGA hardware, Xilinx PYNQ-Z1, which is used to implement a neural network, is discussed along with the PYNQ framework.

2.1 Background

Machine-learning technology drives many aspects of modern society: from online fraud detection to web searches to content personalisation on social media platforms. It is becoming increasingly present in consumer devices such as laptops, tablets and smartphones. Machine learning algorithms are being used to identify faces in images, predict traffic congestion, personalise online search results and recommend products based on shopping habits [1]. Increasingly, these applications make use of a class of techniques called deep learning. What makes deep learning particularly special is that it learns from vast amounts of unstructured data that could potentially take humans decades to understand and process. This is made possible by Artificial Neural Networks (ANNs).

ANNs are biologically inspired computer programs designed to simulate how the human brain processes information. ANN's gather knowledge by identifying patterns and connections in data and learn through experience, not from programming (Agatonovic-Kustrin and Beresford, 2000). This is in stark contrast to traditional approaches to programming where we tell the computer what to do, and it breaks the problem down, executing each task. Here, we do not instruct the computer on how to solve the problem, the neural network instead learns from extensive collections of data, analysing its own solution to the present problem (Nielsen, 2019).

2.2 Neural Networks

An artificial neural network is a biologically inspired computational model formed from hundreds of single units, artificial neurons. Each connection between neurons has weights associated with them, creating the neural structure seen in Figure 1 (Agatonovic-Kustrin and Beresford, 2000).

Each ANN comprises an input and output layer at either end with hidden layers in the middle. The input layer provides data to the network; no computations are performed in these nodes; they are only there to pass data on to the hidden nodes. Within the hidden layer, the neurons take in a set of weighted inputs, perform computations, and transfer the results to the output layer. Finally, the output layer presents the results learned by the ANN [4].

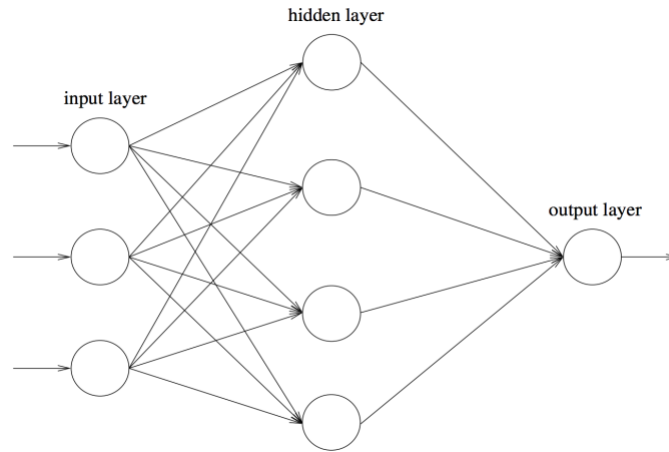


Figure 1: Model of an artificial neural network showing input, output and hidden layers. [10]

2.2.1 Learning

Artificial neural networks adjust and modify their architecture in order to learn. An ANN can change weights of connections based on input and desired output. In order to modify a networks architecture, several parameters can be adjusted. Connections between neurons can be created or deleted; for example, if we wish to remove a neuron, then the weights on all of its connections could be set to zero [5]. If the network is successful, the system learns not only the training examples but also the underlying relationship that allows it to produce the correct output in response to new inputs. Useful learning may occur despite noisy or incomplete data, and other defects in the training set (Reed and Marks, 1999).

2.2.2 Training

There are three approaches to training; supervised, unsupervised and reinforced learning. In supervised learning, we have access to examples of correct input-output pairs. The network can, therefore, check its calculated output matches the desired output, and if not adjust its connection weight combinations. Supervised learning can find solutions to several problems such as classification, forecasting, prediction and robotics (Sathya and Abraham, 2013).

A requirement of supervised learning is a strict data set of correct input and outputs that the network can learn from. Contrastingly, with unsupervised learning, the data is unlabelled and there are no targets. Instead, the network adapts to regularities in the data, finding patterns that we are unable to see, according to rules embedded in its design (Reed and Marks, 1999). Given the lack of labelled data in the world, the potential for unsupervised learning on these datasets is remarkably significant.

Lastly, reinforced learning refers to goal-oriented algorithms. The ANN learns from interacting with a specific environment, such as driving a vehicle or playing a game. The reinforced learning framework allows the network to learn via trial and error, receiving feedback in the form of rewards and punishments with the goal to learn and select the actions that maximise the rewards and reduce the fines [8, 9].

2.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in input data; images, videos or audio files, assign learnable parameters, weights and biases, to various aspects in the data and be able to discern one from the other (Medium, n.d). The CNNs fundamental role is to take input data and compress it into a form which is simpler to process, without losing aspects which are crucial for achieving a correct prediction [15]. A typical CNN has a pipeline-like architecture and is comprised of a stack of convolutional modules that implement feature extraction (shown in Figure 2) [17].

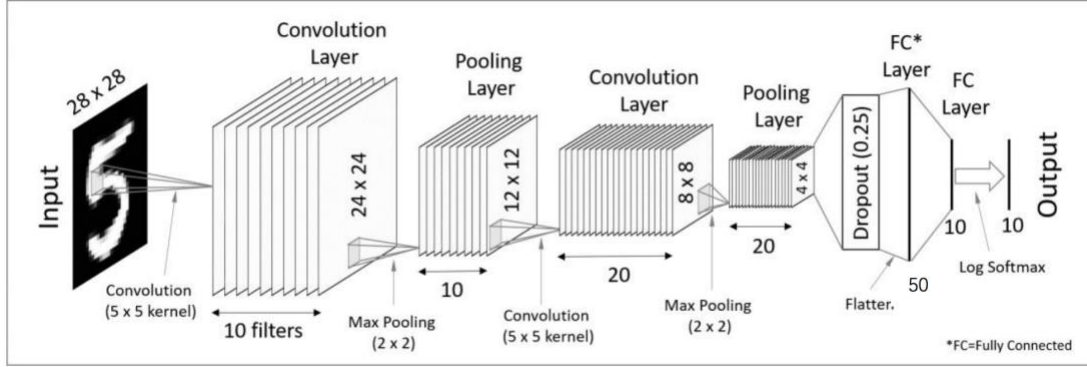


Figure 2: Example of a convolutional neural network model [17].

Each module comprises of a convolutional layer followed by a pooling layer. The final convolutional module has one or more dense layers that implement classification (Allaire, 2019). A brief description of the different layers is outlined below:

- **A convolution layer** is the first layer to extract characteristics from the input image. It carries the fundamental portion of the network's computational load. A convolution operation can be described as the production of a matrix smaller in size to that of the original image, representing pixels, by scanning a small window, called a filter or kernel, of size $x \times x$ over the image, to produce an output feature neuron value. As the filter scans over the image, it multiplies the values in the filter with the original pixel values and then sums the products, to produce a single number (Shawahna, Sait and El-Maleh, 2019). Once complete, the original input image is reduced to a final feature map as seen in figure 3.

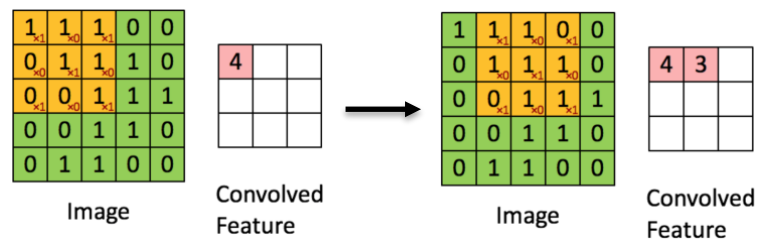


Figure 3: Illustration of the first two scans of a convolutional layer [18].

- **Within the pooling layer**, the dimensions of the output of the convolutional layer are compressed to lower the number of parameters and computation in the network. The pooling layer operates on each feature map independently [20]. According to Williams, T. and Li, R. (2018) max pooling and average pooling are the two most popular forms of pooling. Max pooling takes the largest value of a region and selects it for the condensed feature map. Average pooling calculates the average value of a region and selects it for the condensed feature map. An example of both max and average pooling methods is expressed in figure 4 [21].

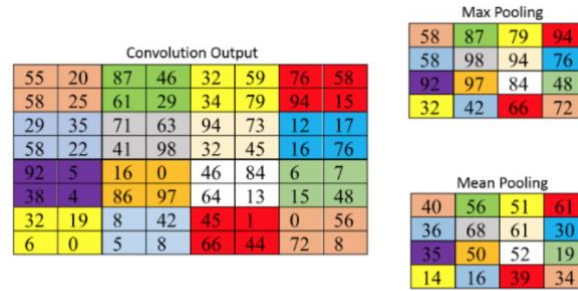


Figure 4: Max and Average pooling methods [21].

- **A fully connected layer** is typically the final layer at the end of the network that performs the classifications. The feature map matrix is converted into a vector and fed into the fully connected layer. Finally, an activation function is used to classify the outputs [18].

2.4 Quantization

For all its popularity, deep neural networks (DNN) are not without their disadvantages. DNNs consume vast amounts of memory, draining battery life from devices during training and inference, which is why networks are usually trained using powerful hardware [22]. Li et al., (2017) indicated that there is a growing interest in training and deploying neural networks directly on portable devices, such as mobile phones and tablets. In 2016 Apple installed an FPGA chip inside their iPhone 7 and more recently developed a custom chip called the A12 Bionic with a Neural Engine, explicitly designed for AI tasks [23, 24].

Li et al., (2017) suggested that in order to make neural networks efficient on embedded systems, many researchers have focused on training networks with coarsely quantized weights to condense the models, without reducing performance [25]. Quantized weights can significantly reduce memory size, increase power efficiency, and exploit hardware-friendly bitwise operations. The most extreme case of quantization is binarization [22]. Binarization compresses the neural network model, reducing the bit-width of inputs and weights from 32 bit (single-precision floating-point) to a single bit (Liang, S. et al., 2018). The development of binarized models makes it possible to implement a system on FPGAs with much greater performance than floating-point versions [12].

2.5 PYNQ-Z1 Board

As previously discussed, the popularity of deep neural networks, particularly CNNs and implementations on embedded devices, has prompted researchers to explore the possibilities of implementing neural networks on field-programmable gate arrays (FPGA). FPGAs are commercially available programmable devices that comprise mainly of configurable logic blocks (CLBs), a programmable interconnection network, and a set of programmable input and output cells around the device [19]. FPGAs are a flexible platform for implementing custom hardware functionality as they can be reprogrammed to desired applications after production [26]. This report will focus on training CNNs and LFC networks using GPUs in the AWS cloud and executing inference on the Digilent PYNQ-Z1 Python Productivity Board for Zynq-7000 FPGA.

The PYNQ-Z1 hardware board is a general-purpose, programmable platform for embedded systems that is designed to be used with the open-source framework PYNQ. The key technologies on the board are Jupyter Notebook, Linux and Python. The PYNQ framework allows developers to explore the capabilities of Xilinx Zynq without having to design programmable logic circuits by using the Python language and library. The model architecture of the PYNQ-Z1 FPGA is seen illustrated in figure 5. The PYNQ-Z1 features a Dual-core ARM Cortex A9 processor, 512MB of memory and a Micro SD card slot for storage [27].

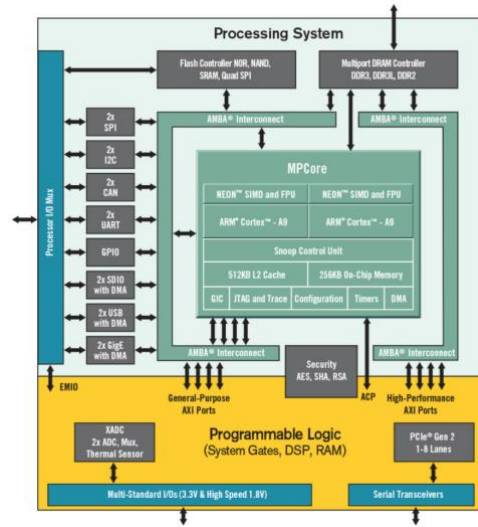


Figure 5: Architecture of the PYNQ-Z1 FPGA Board [27].

Another essential feature of the PYNQ-Z1 is the overlays. Overlays can be used to boost a software application or to tailor the hardware platform for a specific purpose. Overlays are configurable FPGA designs that expand the user application processing system of the Zynq into the Programmable Logic [28]. The overlay applied to the PYNQ-Z1 in this project was the Quantized Neural Network– Binary Neural Network (QNN-BNN) overlay, which included two different network topologies of CNV and LFC. Below are the block structure diagrams of both CNV and LFC overlays on the PYNQ-Z1 hardware (figure 6).

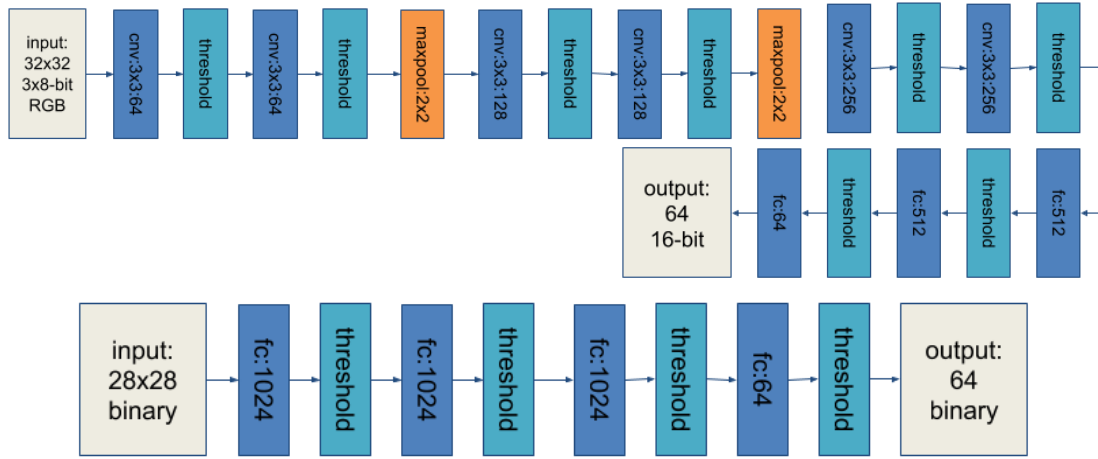


Figure 6: Block structure diagrams of CVN and LFC overlays. CVN binarized except first layer (8 bit inputs) and last layer (16 bit outputs). LFC fully binarized including inputs and outputs [29].

Chapter 3 Design and Implementation

In this chapter, the setup on which an experimental neural network model was implemented, is presented. First, the training environment is set up and dataset chosen. Then the models are trained using GPUs in the AWS cloud to obtain useful parameters for successful application execution. Next, the trained models are implemented on the Xilinx PYNQ-Z1 FPGA. Lastly, a Jupyter Notebook is created, showing inference performance on the hardware and software.

3.1 Training Environment

3.1.1 Amazon Web Services (AWS)

The decision to utilise the cloud service platform, Amazon Web Services (AWS) for the training of the neural network model was due to not having immediate access to GPU hardware. The training could also have been performed without GPU; however, this was not practical in terms of time. The Amazon Elastic Compute Cloud (Amazon EC2) allows users to develop and deploy applications in virtual computing environments, known as instances, thus providing the opportunity to run complex, high computation tasks without investing in hardware [29]. The first stage is to choose an Amazon Machine Image (AMI). An AMI is a preconfigured setting, which includes an operating system, applications and software [30]. For this study, the Deep Learning AMI was selected as it contained many of the software packages required to train the neural network effectively. The Deep Learning AMI comes preconfigured with the Linux Ubuntu 16.04 operating system, frameworks such as Theano and Keras as well as GPU accelerated libraries, cuDNN and NVIDIA CUDA.

From an AMI, the next stage is to launch an instance. This is a copy of the AMI running as a virtual server in the cloud. After exploring the instance options available, it was decided that the P2 instances were the most suitable for machine learning, providing GPU-based parallel compute capabilities [31]. The p2.xlarge instance was selected as the best compromise between cost and expected performance. The instance comes with the following specifications:

Name	GPUs	vCPUs	RAM (GiB)	ECUs	Network Bandwidth	Price/Hour
p2.xlarge	1	4	61	11.75	High	\$0.972

Finally, once the instance has been established, an environment is chosen from the selection of deep learning frameworks. The Theano with Python2 (CUDA 9.0) framework was used to finalise the setup of the virtual environment. The Theano software will be discussed further in the following section.

3.1.2 Software

As the application of deep learning continues to grow, so does the choice of frameworks available. Within the last few years, there has been numerous new

deep learning software launched, such as PyTorch, PlaidML and BigDL [31]. PyTorch was released in 2016 and is a Python-based scientific computing package which defines mathematical functions and calculates their gradients. Unlike frameworks such as Theano or Tensorflow, the PyTorch library operates with a dynamically updated gradient as opposed to a process of compiling the gradient. With PyTorch, the developer can make adjustments to the architecture in the process, as there is no compilation stage [32].

Despite there being a large number of options available in terms of frameworks, this study was limited to the tools and libraries compatible with the PYNQ BNN-QNN overlay. Brief descriptions of the software required in the training of the network are as follows:

- **Theano** is a Python library which allows users to define, improve and evaluate mathematical expressions involving multi-dimensional arrays effectively [34].
- **NumPy** is tightly integrated with Theano and is a scientific computing Python library that provides fast and efficient operations on arrays [35].
- **Lasagne** is a Python library which works with Theano to build and train neural networks [36].
- **PyLearn2** is a machine learning toolbox built on top of Theano. Users can write PyLearn2 plugins using mathematical expressions, and Theano will optimise and compile them [37].

3.1.3 Datasets

Sourcing or producing high-quality labelled training datasets for supervised deep learning algorithms is one of the emerging challenge’s researchers face in machine learning (Roh et al., 2017). Datasets can be difficult and costly to create due to the large amount of time required to collect, clean, analyse and label the data [38]. Due to this, researchers often use the widely popular and established MNIST, Cifar 10 and German Road Traffic Signs datasets, which allows researchers to prototype and test their algorithms quickly.

The Zalando Research team, conscious of how often the MNIST dataset is used, developed the Fashion-MNIST dataset with the intention of it serving as a direct drop-in replacement for the original. Zalando believes the MNIST set is too simple and overused by data scientists and the likes. The Fashion MNIST dataset comprises of 28 x 28 greyscale images of 70,000 fashion products, from 10 categories, keeping to the same format as the original set. The training set has 60,000 patterns, and the test set has 10,000 patterns. Table 1 provides a summary of the Fashion-MNIST images and classes [39].

During the early trial and research phase of the project, the German Traffic Road Signs dataset [48] was trained on the CNV model, which took approximately 16 hours to train on the environment outlined in the previous section. Therefore, it was decided that for the final outcome, the Fashion-MNIST dataset would be used as this was more practical in terms of time restrictions. The Fashion-MNIST dataset is a considerably smaller data set to that of other fashion-related datasets found, such as the DeepFashion Database. This dataset contains over 800,000 images with 50 labelled categories [40]. Therefore, training and deploying the

Fashion-MNIST model on a less complex LFC network, compared to that of a CNN, was more feasible in the time provided.

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Table 1: Class names and example images from the Fashion-MNIST dataset [39].

3.1.4 Training

According to Gavrilov et al., (2018), data set training has a significant influence on the accuracy of a network and is therefore important to create a network architecture that prevents overfitting and underfitting. The term overfitting refers to when a model predicts well on its test data, but poorly on new input data that it has not seen. When a model is unable to find relationships among inputs, it is called underfitting [41]. According to Bourez (2017), in order to get a precise measure of how the model behaves on unseen data during training, the validation dataset is used to compute a validation loss and accuracy during training. The validation dataset enables the developer to choose the best model while the test dataset is used at the end to get the final test accuracy.

As discussed above, the Fashion-MNSIT dataset was selected for training. The supervised learning technique was used, and the image seen in figure 8 shows the training in action. The epoch time can be seen along with learning rate, test error rate, trained and validation loss.


```

Epoch 69 of 1000 took 4.06352901459s
LR: 0.00160369307819
training loss: 0.142491195038
validation loss: 0.121659447402
validation error rate: 21.5100000352%
best epoch: 65
best validation error rate: 21.3700000346%
test loss: 0.123763032705
test error rate: 21.2899999693%
Epoch 70 of 1000 took 4.66219091415s
LR: 0.00158899033167
training loss: 0.141817371234
validation loss: 0.119345329255
validation error rate: 20.6599999666%
best epoch: 70
best validation error rate: 20.6599999666%
test loss: 0.121530635431
test error rate: 21.2099999785%
Epoch 999 of 1000 took 4.07728600502s
LR: 3.05577416416e-07
training loss: 0.0767596192732
validation loss: 0.0910925603658
validation error rate: 14.5400001109%
best epoch: 750
best validation error rate: 14.2199999727%
test loss: 0.0917598193884
test error rate: 14.439999979%
Epoch 1000 of 1000 took 4.07468008995s
LR: 3.02775865823e-07
training loss: 0.0768813100383
validation loss: 0.091412329115
validation error rate: 14.5900000036%
best epoch: 750
best validation error rate: 14.2199999727%
test loss: 0.0917598193884
test error rate: 14.439999979%

```

Figure 8: Training of the Fashion-MNIST model.

A number of these terminologies are explained in further detail below:

- **An epoch** value represents a number of times that the training architecture runs through each image [41]. During the fashion-MNIST training, the epoch duration remained under 5 seconds, resulting in a completion time of approximately 1 hour.
- **The Learning Rate (LR)** is a configurable hyper-parameter that controls how much to update the model's weights during training. If the learning rate is too low, training may become slow; however, if the rate is too high training could increase noise in the data (Bourez, 2017).
- **Training Loss** is a mathematical expression that measures how incorrect a prediction is. If predictions are predominantly incorrect, the output will be a higher value. If the training loss is lower than the *validation loss*, this could be an indication of overfitting [42].
- **Test Error Rate** is the percentage of error from the model running the test dataset, which is images the network was not exposed to during training.

3.2 Implementation on PYNQ-Z1 Board

The final part of the development stage is the implementation of the trained network on the Xilinx PYNQ-Z1 FPGA. This section delves into the steps taken to execute the finished model on both the PYNQ-Z1's hardware and software. First, the connection to the PYNQ-Z1 is made by navigating to the boards IP address, and then the Jupyter Notebook software is launched. Jupyter Notebook is an interactive environment for writing and running multiple programming languages. The Notebook server runs on the ARM processor of the PYNQ-Z1 and is associated with the iPython kernel, which runs the Python language [44]. Next, the finished training file containing a list of NumPy arrays which correspond to the real trained weights in each layer, require converting from real point values into binary values [45]. This script was executed in the Jupyter notebook terminal, as illustrated in figure 9. From here, a new Jupyter Notebook was created to begin inferencing in Python.

```

root@pyng:/home/xilinx/jupyter_notebooks/BNN-PYNQ/bnn/src/training# python fashion-mnist-gen-binary-weights.py
Extracting FCBN complex, ins = 784 outs = 1024
Layer 0: 1024 x 832, SIMD = 64, PE = 32
WMem = 416 TMem = 32
Extracting FCBN complex, ins = 1024 outs = 1024
Layer 1: 1024 x 1024, SIMD = 32, PE = 64
WMem = 512 TMem = 16
Extracting FCBN complex, ins = 1024 outs = 1024
Layer 2: 1024 x 1024, SIMD = 64, PE = 32
WMem = 512 TMem = 32
Extracting FCBN complex, ins = 1024 outs = 10
Layer 3: 16 x 1024, SIMD = 8, PE = 16
WMem = 128 TMem = 1
Config header file:

```

Figure 9: Running script to generate binary weight files on Jupyter Notebook.

Inferencing uses the models built during the training phase to make decisions about new data to which it is presented. The model classifies new input data, predicting what category it belongs to based on what it has already learned [46]. The BNN package is imported into the notebook to begin the inferencing. The LFC classifier is then instantiated for both the software and hardware, as seen in figure 10.

```

hw_classifier = bnn.LfcClassifier(bnn.NETWORK_LFCW1A1, "fashion-mnist", bnn.RUNTIME_HW)
sw_classifier = bnn.LfcClassifier(bnn.NETWORK_LFCW1A1, "fashion-mnist", bnn.RUNTIME_SW)

print(hw_classifier.classes)

['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

```

Figure 10: Instantiating the software and hardware classifiers.

Next, the image is loaded, cropped and converted to BNN input format for processing through the LFC network, where it was correctly classified. From there inferencing was complete on the 10,000-test set batch of images on both the software (executed on the PYNQs ARM cortex A9 processor) and hardware. The results and accuracy of the network are discussed in the following section.

Chapter 4 Evaluation

In this chapter, the performed tests on the FPGA will be presented. Additionally, the results of the inference and training of the LFC model will be discussed and analysed.

4.1 Speed and Accuracy Results

First, the network was tested with a single image of a dress. This was to ensure the Python code correctly formatted input data into 28 x 28 pixels, inverted colours to white on black and successfully ran the image through the network. The results showed the image was correctly rescaled and identified as a dress from class number 3. Inference time on hardware was *24.00 microseconds* and *79804.00 microseconds* on software, which is an early indication of the vast difference in inferencing speed between the two. Next the model was tested on a batch of 10,000 fashion product images to gain a better insight into the overall accuracy of the network. The network inferencing on the hardware took 0.08 seconds compared to 13.16 minutes on the software (figure 11).

Inference took 84121.00 microseconds, 8.41 usec per image
Classification rate: 118876.38 images per second
Inference took 789958750.00 microseconds, 78995.88 usec per image
Classification rate: 12.66 images per second

Figure 11: Hardware inference results on top and software inference results on bottom.

There is an extensive gap between inference times, providing clear evidence of the speed and performance capabilities of the FPGA. Finally, accuracy is assessed. The result for both hardware and software's overall accuracy was *84.81%*, as shown in figure 12.

Hardware Accuracy Fashion MNIST: 84.81
Software Accuracy Fashion MNIST: 84.81

Figure 12: Overall network accuracy on hardware and software.

A factor believed to be affecting the accuracy results is that the current model's architecture does not contain any convolutional layers. The resizing of images beforehand, in order to meet input standards, is potentially causing a loss of features. Features likely important to the prediction process. Having said that, the network has performed reasonably well considering the more basic architecture to that of a CNN. Chou et al., (2019) published their results when identifying Fashion-MNIST images, using a 16-layer and 8-layer CNN architecture. The average correct prediction rates for a training set of 60,000 samples on the 16-layer CNN was 96.57% and 93.94% on the 8-layer CNN. The test set of 10,000 samples achieved an average correct prediction rate of 91.68% on the 16-layer CNN and 90.11% on the 8-layer CNN. Therefore, the LFC network accuracy results of 84.81% on the test set of 10,000 images are not that dissimilar to the results of the

8-layer CNN. However, the omission of convolutional layers has played a significant role in the loss of accuracy within the network.

4.2 Tests

This section analyses the possible weaknesses in the network, looking at input data's actual and predicted results. First, an image from each category was tested. The initial test was carried out to determine if there were any obvious indications of groups, such as clothing, footwear or accessories, that perhaps did not perform as well as others. All input images contained a white background with a single product image. Each image was resized to input standards, and inference ran on both hardware and software. Table 2 displays the results from the initial testing of each category.

	Test Number	Actual	Predicted	Inference (microseconds)
Hardware	1	Dress	Dress	24.00
Software		Dress	Dress	79657.00
Hardware	2	Ankle Boot	Bag	23.00
Software		Ankle Boot	Bag	79808.00
Hardware	3	Bag	Bag	22.00
Software		Bag	Bag	79462.00
Hardware	4	Coat	Tshirt/top	23.00
Software		Coat	Tshirt/top	79733.00
Hardware	5	Pullover	Pullover	24.00
Software		Pullover	Pullover	79372.00
Hardware	6	Shirt	Shirt	22.00
Software		Shirt	Shirt	79311.00
Hardware	7	Sneaker	Bag	24.00
Software		Sneaker	Bag	80101.00
Hardware	8	Trouser	Trouser	23.00
Software		Trouser	Trouser	79741
Hardware	9	Sandal	Bag	23.00
Software		Sandal	Bag	80097.00
Hardware	10	T-shirt/top	T-shirt/top	23.00
Software		T-shirt/top	T-shirt/top	79788.00

Table 2: Inference results of the 10 images testing each category.

The outcome reveals that four categories were incorrectly predicted, 'sneaker', 'ankle boot', 'sandal' and 'coat'. These results are interesting and suggest that the network may have difficulty in correctly predicting footwear. In order to investigate this theory further, an additional ten images containing male and

female footwear from the three categories previously specified were submitted to the network (table 3).

	Test Number	Actual	Predicted	Inference (microseconds)
Hardware	1	Ankle Boot	Bag	23.00
Software		Ankle Boot	Bag	81722.00
Hardware	2	Ankle Boot	Bag	23.00
Software		Ankle Boot	Bag	80058.00
Hardware	3	Ankle Boot	Bag	23.00
Software		Ankle Boot	Bag	79520.00
Hardware	4	Ankle Boot	Sandal	24.00
Software		Ankle Boot	Sandal	79503.00
Hardware	5	Sandal	Bag	23.00
Software		Sandal	Bag	79545.00
Hardware	6	Sandal	Bag	23.00
Software		Sandal	Bag	79804.00
Hardware	7	Sandal	Sandal	23.00
Software		Sandal	Sandal	79534.00
Hardware	8	Sneaker	Sneaker	23.00
Software		Sneaker	Sneaker	79683.00
Hardware	9	Sneaker	T-shirt/top	23.00
Software		Sneaker	T-shirt/top	79557.00
Hardware	10	Sneaker	Bag	23.00
Software		Sneaker	Bag	79620.00

Table 3: The results of inference ran on footwear only images.

The results displayed in table 3 exhibit the network correctly identifying two out of ten images as footwear. In addition, the network has identified footwear as category bag on all occasions during the first test and on six occasions during the second. As only a small batch of footwear images have been tested, there is not enough evidence to fully support the claim that the majority of incorrect predictions are due to footwear. However, it does provide a compelling argument as to the direction in which further error analysis should be performed.

Chapter 5 Conclusion

5.1 Reflection

This document has described how the development process was undertaken, from background research, through training, implementation and evaluation, culminating in a successful working neural network trained to identify fashion products on the Xilinx FPGA. This project was not without its challenges, having found neural networks and the topic of deep learning to be a fruitful yet steep learning curve. In hindsight during the initial research and acquisitions stages, a small dataset on a CNN architecture would have been trialled, leaving time for a more complex CNN model to be implemented during the development stages. This would have probably yielded better results in the final outcome.

The practicality of the network could probably be vastly improved and better utilised with a deeper understanding of the Python programming language. It would have been beneficial to be able to have programmed an algorithm to process a batch of approximately 100 images, which resizes to input standards and runs through the network then prints the actual and predicted results. As this had to be executed manually during the evaluation stages, the results could be seen as inconclusive. However, having said that, the PYNQ QNN-BNN overlay and repository was an invaluable learning tool, which aided and guided the deep learning process from the environment set up to hardware implementation. The example Jupyter notebooks provided with the PYNQ framework were an excellent guidance tool navigating the python commands required to instantiate a classifier and run inference on software and hardware. The PYNQ-Z1 BNN project does an impressive job connecting the gap between hardware and software engineering.

Finally, there were a few limitations during the project that needed to be acknowledged. First, there were cost limitations to consider. As the AWS cloud was being utilised for training, this could potentially accrue to a considerable cost. The larger AWS instances were rather expensive to use, meaning that the project was limited to using smaller instances. However, that being said, the smaller-sized instances were then being used for more hours. This had to be taken into consideration when selecting an instance. Not only does dataset training time need to be considered but also the time taken to become familiar with the instance environment. The second limitation, which is closely related to cost, is time limitations. Large datasets of over 70,000 images, depending on the size and detail, could take significantly longer to train, even with the cudNN library installed. This meant that smaller datasets had to be selected as it was not time, nor cost practical to train larger ones.

5.2 Future Works

Whilst a successful neural network model was developed and implemented on an FPGA, there are still areas where more work could be done to improve the architecture and further the research started in this project. Based on the network's overall accuracy, more error analysis could be conducted. The evaluation could be developed further looking at technical steps to assess the model's performance, establishing the precision, recall and F1 metrics used in machine

learning. Further work could also include training the Fashion-MNIST dataset on a CNN model and deploying on the Xilinx FGPA and comparing results to that of the LFC model. It would also be interesting to see if training the LFC model with more data or altering parameters would improve the network and if so, how it would hold up in comparison to a CNN model trained on the same dataset.

In general, future research should evaluate different neural network models on a variety of FPGA hardware devices. The current implementation is based on the Xilinx PYNQ-Z1, which, as discovered, enables hardware and software engineers to develop embedded systems at a higher abstraction level without having the necessary hardware design background. This is vital in bridging the gap between hardware engineering and software development. Furthermore, as mentioned in chapter 3, there is a wide selection of deep learning frameworks, which could also be explored and implemented in future projects.

Finally, a subject of particular interest and, thus, a potential topic of future research projects is deploying a neural network model to a mobile application and inferencing on a mobile device. Recent years have witnessed increased usage of mobile devices and smart applications, making mobile deep learning an interesting focal point for research. Machine learning has become increasingly used in mobile applications with many of the large companies such as Apple, continuously investing in it [24].

References

- [1] Medium. (n.d.). *9 Applications of Machine Learning from Day-to-Day Life*. [online] Available at: <https://medium.com/app-affairs/9-applications-of-machine-learning-from-day-to-day-life-112a47a429d0>
- [2] Agatonovic-Kustrin, S. and Beresford, R. (2000). Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis*, [online] 22(5), pp.717-727.
- [3] Nielsen, M. (2019). Neural Networks and Deep Learning. [online] [Neuralnetworksanddeeplearning.com](http://neuralnetworksanddeeplearning.com). Available at: <http://neuralnetworksanddeeplearning.com/> [Accessed 9 Aug. 2019].
- [4] The Data Science Blog. (n.d.). *A Quick Introduction to Neural Networks*. [online] Available at: <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/> [Accessed 9 Aug. 2019].
- [5] Rubik's Code. (n.d.). *How do Artificial Neural Networks learn? | Rubik's Code*. [online] Available at: <https://rubikscore.net/2018/01/15/how-artificial-neural-networks-learn/>.
- [6] Reed, R. and Marks, R. (1999). *Neural Smithing*. Cambridge, Mass.: CogNet.
- [7] Sathya, R. and Abraham, A. (2013). Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2).
- [8] Mousavia, S., Schukat, M. and Howley, E. (2018). Deep Reinforcement Learning: An Overview. *Conference: Proceedings of SAI Intelligent Systems Conference*. [online] Available at: https://www.researchgate.net/publication/319251315_Deep_Reinforcement_Learning_An_Overview [Accessed 12 Aug. 2019].
- [9] Medium. (n.d.). *Unsupervised Learning and Data Clustering*. [online] Available at: <https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a>
- [10] Engelbrecht, A. (2008). *Computational Intelligence*. Hoboken: John Wiley & Sons, Ltd.
- [11] Abiodun, O., Jantan, A., Omolara, A., Dada, K., Mohamed, N. and Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, [online] 4(11), p.e00938. Available at: <https://www.sciencedirect.com/science/article/pii/S2405844018332067>
- [12] Liang, S., Yin, S., Liu, L., Luk, W. and Wei, S. (2018). FP-BNN: Binarized neural network on FPGA. *Neurocomputing*, 275, pp.1072-1086.
- [13] Zhao, M., Hu, C., Wei, F., Wang, K., Wang, C. and Jiang, Y. (2019). Real-Time Underwater Image Recognition with FPGA Embedded System for Convolutional Neural Network. *Sensors*, 19(2), p.350.
- [14] Shen, J., Wang, D., Huang, Y., Wen, M. and Zhang, C. (2019). Scale-out acceleration for 3D CNN-based lung nodule segmentation on a multi-FPGA system. *Proceedings - Design Automation Conference*, pp.1 - 6.

- [15] Medium. (n.d). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. [online] Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Accessed 28 Aug. 2019].
- [16] Allaire, J. (2019). TensorFlow for R. [online] Tensorflow.rstudio.com. Available at: <https://tensorflow.rstudio.com/tfestimators/articles/layers.html> [Accessed 1 Sep. 2019].
- [17] Shi, H. (2018). An Architecture Combining Convolutional Neural Network and Support Vector Machine for Image Recognition. Research School of Computer Science, Australian National University.
- [18] Medium. (2019). *Understanding of Convolutional Neural Network (CNN) — Deep Learning*. [online] Available at: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148> [Accessed 1 Sep. 2019].
- [19] Shawahna, A., Sait, S. and El-Maleh, A. (2019). FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. IEEE Access, 7, pp.7823-7859.
- [20] Medium. (n.d). *The best explanation of Convolutional Neural Networks on the Internet!*. [online] Available at: <https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8> [Accessed 2 Sep. 2019].
- [21] Williams, T. and Li, R. (2018). Wavelet Pooling for Convolutional Neural Networks. In: *International Conference on Learning Representations*. [online] North Carolina. Available at: https://www.researchgate.net/publication/324090115_Wavelet_Pooling_for_Convolutional_Neural_Networks
- [22] Li, H., De, S., Xu, Z., Studer, C., Samet, H., Goldstein, T., (2017). Training quantized nets: A deeper understanding, in: *Advances in Neural Information Processing Systems*. Neural information processing systems foundation, pp. 5812–5822.
- [23] Tilley, A. (2016). *This Mysterious Chip In The iPhone 7 Could Be Key To Apple's AI Push*. [online] Forbes. Available at: <https://www.forbes.com/sites/aarontilley/2016/10/17/iphone-7-fpga-chip-artificial-intelligence/#5a6b34d23c69>.
- [24] Apple (United Kingdom). (n.d). *iPhone XS - A12 Bionic*. [online] Available at: <https://www.apple.com/uk/iphone-xs/a12-bionic/> [Accessed 3 Sep. 2019].
- [25] Guo, Y., (2018). A Survey on Methods and Theories of Quantized Neural Networks.
- [26] Xilinx. (n.d). *What is an FPGA? Field Programmable Gate Array*. [online] Available at: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>
- [27] Mouser (n.d). *PYNQ-Z1 Python Productivity Board - Digilent | Mouser Europe*. [online] Available at: <https://eu.mouser.com/new/digilent/digilent-pynq-z1-board/>
- [28] Pynq (n.d). PYNQ Overlays — Python productivity for Zynq (Pynq) v1.0. [online] Available at: https://pynq.readthedocs.io/en/v2.1/pynq_overlays.html
- [29] Amazon Docs (n.d). *What Is Amazon EC2? - Amazon Elastic Compute Cloud*. [online] Available at: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>

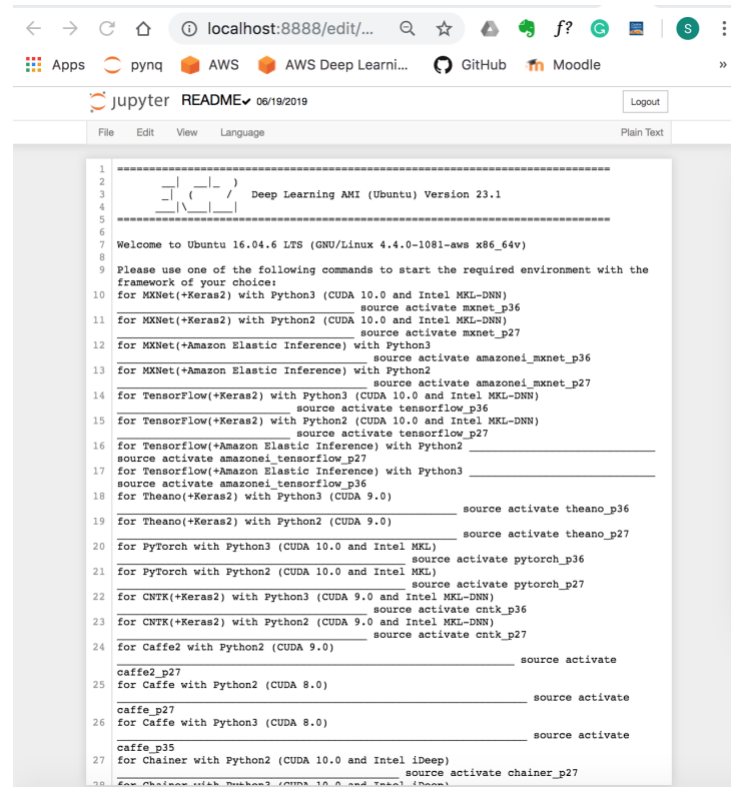
- [30] Amazon Docs (n.d.). Instances and AMIs - Amazon Elastic Compute Cloud. [online] Available at: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html>
- [31] Amazon Web Services, Inc. (n.d.). *Amazon EC2 - P2 Instances*. [online] Available at: <https://aws.amazon.com/ec2/instance-types/p2/>
- [32] Wikipedia (n.d.). *Comparison of deep-learning software*. [online] Available at: https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software
- [33] Ketkar N. (2017) Introduction to PyTorch. In: Deep Learning with Python. Apress, Berkeley, CA
- [34] Deeplearning.net. (2017). *Welcome — Theano 1.0.0 documentation*. [online] Available at: <http://deeplearning.net/software/theano/> [Accessed 4 Sep. 2019].
- [35] Docs.scipy.org (2019). *What is NumPy? — NumPy v1.17 Manual*. [online] Available at: <https://docs.scipy.org/doc/numpy/user/whatisnumpy.html>
- [36] Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., Fauw, J., Heilman, M., McFee, B., Weideman, H., Rasul, D. and Degraeve, J. (2015). *Lasagne: First release*. [online] Zenodo. Available at: <http://dx.doi.org/10.5281/zenodo.27878>
- [37] Goodfellow, I., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F. and Bengio, Y. (2013). *Pylearn2: a machine learning research library*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1308.4214>
- [38] Roh, Y., Heo, G. and Whang, S. (2019). A Survey on Data Collection for Machine Learning: a Big Data -- AI Integration Perspective.
- [39] Xiao, H., Rasul, K., Vollgraf, R., (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.
- [40] Liu, Z., Luo, P., Qiu, S., Wang, X. and Tang, X. (2016). DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [41] Gavrilov, A., Jordache, A., Vasdani, M. and Deng, J. (2018). Preventing Model Overfitting and Underfitting in Convolutional Neural Networks. *International Journal of Software Science and Computational Intelligence*, 10(4), pp.19-28.
- [42] Algorithmia Blog. (n.d.). *Introduction to Loss Functions | Algorithmia Blog*. [online] Available at: <https://blog.algorithmia.com/introduction-to-loss-functions/>
- [43] Bourez, C. (2017). Deep Learning with Theano. Packt Publishing Ltd.
- [44] Pynq. (n.d.). *Jupyter Notebooks — Python productivity for Zynq (Pynq) v1.0*. [online] Available at: https://pynq.readthedocs.io/en/v2.1/jupyter_notebooks.html [Accessed 5 Sep. 2019].
- [45] Umuroglu, Y., Fraser, N., Gambardella, G., Blott, M., Leong, P., Jahre, M. and Vissers, K. (2017). FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp.65-74.
- [46] Copeland, M. (2016). *What's the Difference Between Deep Learning Training and Inference?* | *The Official NVIDIA Blog*. [online] The Official NVIDIA Blog. Available

at: <https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/> [Accessed 5 Sep. 2019].

- [47] Chou, F., Tsai, Y., Chen, Y., Tsai, J. and Kuo, C. (2019). Optimizing Parameters of Multi-Layer Convolutional Neural Network by Modeling and Optimization Method. *IEEE Access*, 7, pp.68316-68330.
- [48] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M. and Igel, C. (2013). Detection of Traffic Signs in Real-World Images: The {G}erman {T}raffic {S}ign {D}etection {B}enchmark. *International Joint Conference on Neural Networks*, (1288).

Appendix A Project Screenshots

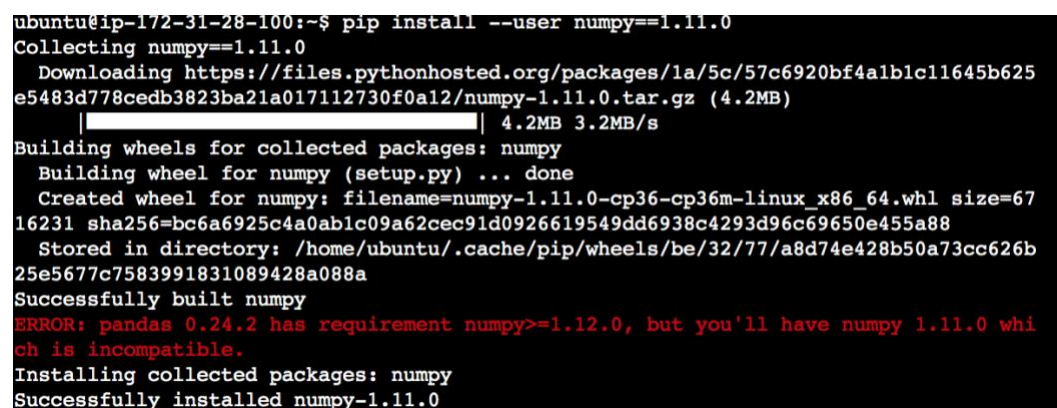
1. Connection to AWS via Jupyter Terminal.



The screenshot shows a Jupyter Notebook interface with a terminal window. The terminal output displays the AWS CLI commands and their results for connecting to an AWS instance. The commands include `aws s3 cp` to upload a file, `aws s3 mv` to move a file, and `aws s3 rm` to remove a file. The output shows the file being uploaded and moved successfully, and the removal command being executed.

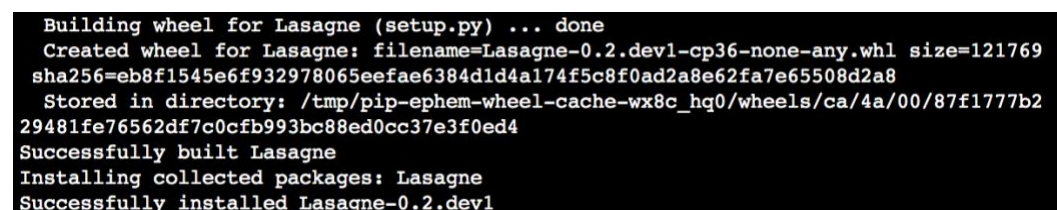
```
1 =====
2      |  _  |  )
3      | (  |  /
4      | \_ |  |
5      |  _ |  |
6      |  _ |  |
7
8 Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1081-aws x86_64v)
9
10 Please use one of the following commands to start the required environment with the
11 framework of your choice:
12 for MXNet(+Keras2) with Python3 (CUDA 10.0 and Intel MKL-DNN)
13     source activate mxnet_p36
14 for MXNet(+Keras2) with Python2 (CUDA 10.0 and Intel MKL-DNN)
15     source activate mxnet_p27
16 for MXNet(+Amazon Elastic Inference) with Python3
17     source activate amazonai_mxnet_p36
18 for MXNet(+Amazon Elastic Inference) with Python2
19     source activate amazonai_mxnet_p27
20 for TensorFlow(+Keras2) with Python3 (CUDA 10.0 and Intel MKL-DNN)
21     source activate tensorflow_p36
22 for TensorFlow(+Keras2) with Python2 (CUDA 10.0 and Intel MKL-DNN)
23     source activate tensorflow_p27
24 for TensorFlow(+Amazon Elastic Inference) with Python2
25     source activate amazonai_tensorflow_p27
26 for TensorFlow(+Amazon Elastic Inference) with Python3
27     source activate amazonai_tensorflow_p36
28 for Theano(+Keras2) with Python3 (CUDA 9.0)
29     source activate theano_p36
30 for Theano(+Keras2) with Python2 (CUDA 9.0)
31     source activate theano_p27
32 for PyTorch with Python3 (CUDA 10.0 and Intel MKL)
33     source activate pytorch_p36
34 for PyTorch with Python2 (CUDA 10.0 and Intel MKL)
35     source activate pytorch_p27
36 for CNTK(+Keras2) with Python3 (CUDA 9.0 and Intel MKL-DNN)
37     source activate cntk_p36
38 for CNTK(+Keras2) with Python2 (CUDA 9.0 and Intel MKL-DNN)
39     source activate cntk_p27
40 for Caffe2 with Python2 (CUDA 9.0)
41     source activate
42 caffe2_p27
43 for Caffe with Python2 (CUDA 8.0)
44     source activate
45 caffe_p27
46 for Caffe with Python3 (CUDA 8.0)
47     source activate
48 caffe_p36
49 for Chainer with Python2 (CUDA 10.0 and Intel iDeep)
50     source activate chainer_p27
51 for Chainer with Python3 (CUDA 10.0 and Intel iDeep)
52     source activate chainer_p36
```

2. Installation of python packages.



The screenshot shows a terminal window with the following output:

```
ubuntu@ip-172-31-28-100:~$ pip install --user numpy==1.11.0
Collecting numpy==1.11.0
  Downloading https://files.pythonhosted.org/packages/1a/5c/57c6920bf4a1b1c11645b625e5483d778cedb3823ba21a017112730f0a12/numpy-1.11.0.tar.gz (4.2MB)
    | 4.2MB 3.2MB/s
Building wheels for collected packages: numpy
  Building wheel for numpy (setup.py) ... done
  Created wheel for numpy: filename=numpy-1.11.0-cp36-cp36m-linux_x86_64.whl size=6716231 sha256=bc6a6925c4a0ab1c09a62cec91d0926619549dd6938c4293d96c69650e455a88
  Stored in directory: /home/ubuntu/.cache/pip/wheels/be/32/77/a8d74e428b50a73cc626b25e5677c7583991831089428a088a
Successfully built numpy
ERROR: pandas 0.24.2 has requirement numpy>=1.12.0, but you'll have numpy 1.11.0 which is incompatible.
Installing collected packages: numpy
Successfully installed numpy-1.11.0
```



The screenshot shows a terminal window with the following output:

```
Building wheel for Lasagne (setup.py) ... done
Created wheel for Lasagne: filename=Lasagne-0.2.dev1-cp36-none-any.whl size=121769 sha256=eb8f1545e6f932978065eefae6384d1d4a174f5c8f0ad2a8e62fa7e65508d2a8
Stored in directory: /tmp/pip-ephem-wheel-cache-wx8c_hq0/wheels/ca/4a/00/87f1777b229481fe76562df7c0cfb993bc88ed0cc37e3f0ed4
Successfully built Lasagne
Installing collected packages: Lasagne
Successfully installed Lasagne-0.2.dev1
```