EFFICIENT IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS ON EMBEDDED DEVICES

BARIŞ YILMAZ

EYLÜL 2022

ÇANKAYA UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

MASTER'S THESIS IN

ELECTRICAL AND ELECTRONICS ENGINEERING

EFFICIENT IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS ON EMBEDDED DEVICES

BARIŞ YILMAZ

EYLÜL 2022

ABSTRACT

EFFICIENT IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS ON EMBEDDED DEVICES

YILMAZ Barış

Master of Science in Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Erdem AKAGÜNDÜZ

September 2022, 69 pages

In the field of artificial intelligence, deep convolutional neural network models are very popular because they can yield results close to those of humans. Depending on the application, these deep learning models can be very simple and small, but also very complex and large. Hence, the performance of an embedded systems that implement these models may be poor and infeasible. Through the use of various methods, this thesis aims to improve deep convolutional neural architecture efficiency without a significant loss of in the performance. For this purpose, we first utilize feature dimension reductions in layer activations. We use methods such as Principal Component Analysis and Select K-Best functions for feature dimension reduction. In the following, in order to make a quantization-aware trained binary deep convolutional neural network model more efficient, we also utilize the "Regular Positive and Negative Inference" algorithm by replacing the fully connected layers of the deep learning model as a decision-making mechanism.

The ultimate aim of this thesis is to observe if these methods would make our models efficient without a significant loss of performance, and if we can further increase the efficiency of a binary quantized deep convolutional neural network.

Keywords: Embedded deep learning, Convolutional neural networks, Quantization, Principal component analysis

ÖZ

GÖMÜLÜ CİHAZLAR ÜZERİNDE EVRİŞİMSEL SİNİR AĞLARININ VERİMLİ UYGULAMASI

YILMAZ Barış

Elektrik-Elektronik Mühendisliği Yüksek Lisans

Danışman: Assoc. Prof. Erdem AKAGÜNDÜZ

Eylül 2022, 69 pages

Yapay zeka alanında, derin evrişimsel sinir ağı modelleri, insan sonuçlarına yakın sonuçlar verebildikleri için çok popülerdir. Uygulamaya bağlı olarak, bu derin öğrenme modelleri çok basit ve küçük olabilir, ancak aynı zamanda çok karmaşık ve büyük de olabilir. Bu nedenle, bu modelleri uygulayan gömülü sistemlerin performansı zayıf ve olanaksız olabilir. Bu tez, çeşitli yöntemlerin kullanılmasıyla, performansta önemli bir kayıp olmadan derin evrişimsel sinir mimarisi verimliliğini iyileştirmeyi amaçlamaktadır. Bu amaçla, ilk olarak katman aktivasyonlarında öznitelik boyutu küçültmelerinden yararlanıyoruz. Özellik boyut küçültme için Temel Bileşen Analizi ve Select-K-Best fonksiyonu gibi yöntemler kullanıyoruz. Sonrasında, niceliksel farkındalık eğitimli ikili derin evrişimli sinir ağı modelini daha verimli hale getirmek için, bir karar verme mekanizması olarak derin öğrenme modelinin tam bağlantılı katmanlarını değiştirerek "Düzenli Pozitif ve Negatif Çıkarım" algoritmasını da kullanıyoruz.

Bu tezin nihai amacı, bu yöntemlerin önemli bir performans kaybı olmadan modellerimizi verimli hale getirip getiremeyeceğini ve ikili nicemlenmiş derin evrişimsel sinir ağının verimliliğini daha da artırıp artıramayacağımızı gözlemlemektir.

Anahtar Kelimeler: Gömülü Derin Öğrenme, Evrişimsel sinir ağları, Niceleme, Temel bileşenler analizi

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my academic advisor, Assoc. Prof. Erdem Akagündüz, who worked tirelessly to push my limits of knowledge, inspired me with his working principle and provided the resources I needed.

I would like to express my sincere gratitude to my professors and department chair Prof. Dr. Yahya Kemal Baykal who generously provided knowledge and expertise during my time as a research assistant at the school.

Finally, I would like to thank my family and friends for always belief in me and kept my spirits and motivation high.

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS

CNN	: Convolutional Neural Network
FPGA	: Field Programmable Gate Array
SoC	: System on Chip
VM	: Virtual Machine
AI	: Artificial Intelligence
ANN	: Artificial Neural Network
RNN	: Recurrent Neural Network
LSTM	: Long Short-Term Memory
MLP	: Multilayer Perceptron
ReLU	: Rectified Linear Unit
UI	: User Interface
GUI	: Graphical User Interface
CPU	: Central Process Unit
GPU	: Graphics Processing Unit
FC Layer	: Fully Connected Layer
FSM	: Finite State Machine

CHAPTER I: INTRODUCTION

1.1 CONTEXT

Machine learning has a wide variety of applications such as image recognition, automatic language translation, speech recognition, traffic forecasting, to name a few. Several algorithms are utilized in machine learning applications to predict some values for a given input data.

The situation is slightly different in deep learning algorithms. Deep learning is a subclass of machine learning and obtains high-level features using multiple layers. Recently deep learning architectures like Convolutional Neural Network (CNN) have taken an important role [51]. When compared to other algorithms, CNNs perform much better at detecting and recognizing objects [1]. As applications have grown more complex and processing loads have increased, neural network models have become large and more computationally complex in embedded systems [2]. And reducing parameter sizes and weight and activation bits are being used to solve the problem of implementing large and complex networks in embedded system implementations. Research has shown that Convolutional Neural Networks can produce outstanding results in accuracy even when the weights and activation bits are reduced or binarized [3].

1.2 PROBLEM STATEMENT

Day by day, neural networks are starting to evolve into more complex structures. Not only is it complex, but it's also getting bigger and bigger. These structures find solutions to certain problems. In real world problems, these complex and big structures can be implemented and used in embedded systems. Our aim in this thesis is to integrate these structures into embedded systems in a more efficient way without any performance loss. In this direction, we aimed to use smaller and less complex structures that can work in harmony with FPGAs by modifying the decision-making mechanisms of CNNs. Although the quantization process performs this task very quickly and in line with FPGA, we compared it with some methods to achieve the same or better levels of performance and accuracy. In this thesis, we try 2 different

methods that we think can compete with quantization in terms of speed and accuracy in FPGA implementation.

- Quantization
- Feature Dimension Reduction
- Hybrid AI

The Quantization solution presents an approach that reduces the burden of activation and weight bits. We approached the Feature Dimension Reduction method in two different ways. First, the PCA method, which is not the primary task but is widely used for feature reduction, is used. Then the less popular Select K Best method is applied, and the results are compared. As a final solution, Hybrid AI is used, which is a different approach than other solutions.

1.2.1 Research Questions

Regarding the problem definition given above, we aim to understand whether more efficient results can be achieved by replacing the decision-making mechanisms of Convolutional Neural Network's with quantization, feature dimension reduction or hybrid AI methods.

- Is it possible to make the decision-making of a Convolutional Neural Network more efficient using Quantization, Feature dimension reduction, and Hybrid AI structures?
- Can Hybrid AI algorithms built with FSM structures yield better results than methods such as quantization commonly used in embedded deep learning?
- What sizes of activations are ideal for experimenting with an algorithm consisting of Finite State Machine structures?
- What are the pros and cons of PCA and Select K Best algorithms for feature dimension reduction?
- How do different activation lengths affect accuracy?

1.3 OUTLINE

The remainder of the thesis is as follow: Chapter 2 present the background theory of ANN, CNN, Hybrid AI, Quantization, and Hardware. Chapter 3 give used environments, tools, and their working logic and PYNQ board. And Chapter 4 used

model explanation and hardware architecture for this work. Chapter 5 present model accuracy results and compare results with Hybrid AI.

CHAPTER II: BACKGROUND

This section includes the basic hardware and theoretical information to start this thesis. The basic knowledge of artificial neural networks and the hardware parts used, as well as the methods by which networks can be made suitable for hardware implementation, are included in this section.

2.1 ARTIFICIAL NEURAL NETWORKS

The beginnings of artificial neural networks originated in the late 1800s when they are investigating how brain functions work. Warren McCulloch and Walter Pitts [4] pioneered these studies in 1943. The methods used are based on algorithms consisting of threshold-based operations. There are two different approaches to this topic. While one of these approaches pioneered biological research, the other pioneered artificial intelligence studies.

Most ANNs generally consist of 3 different layers. The first layer fetches data for processing in the system is called the input layer. It can be any information. Hidden layers are the layers between the input layers and output layers. These layers can appear in many different combinations. Their main task is to process the data they receive from the input layer and produce output to the output layer through the activation function. And the output layer is the layer that makes the outcome using the data from the hidden layers. This output can appear in different ways. Depending on the network's function, it can be an array of numbers or just a single integer. There are also networks created with more complex combinations than the ANN described here. Just to name a few, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), Multilayer Perceptrons (MLPs), etc.

2.1.1 Convolutional Neural Networks

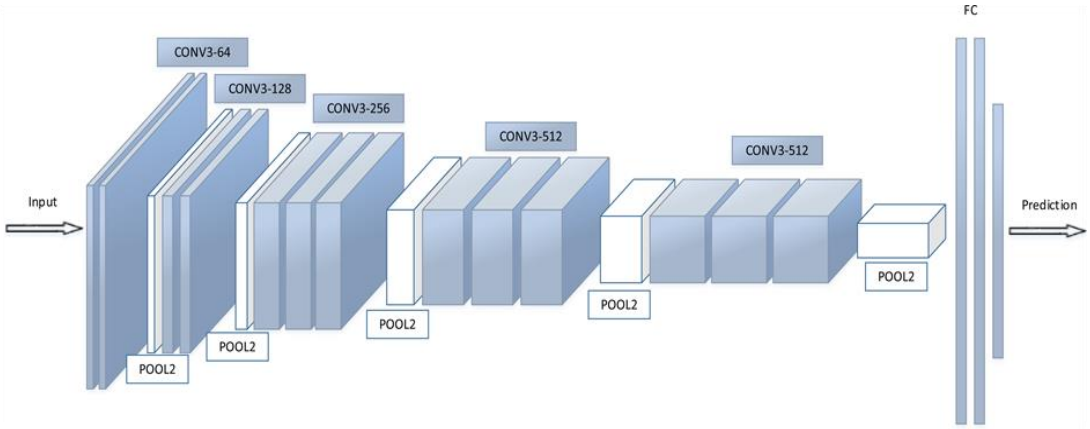


Figure 1: Convolutional Neural Network. From [5]

Convolutional Neural Networks are a class of ANN and are mainly used for image processing and object detection. It is based on the shared-weight architecture of the convolution kernels that slide among input data and provide responses known as feature maps [6]. Convolution of an image with different filters can perform operations such as edge detection and blur by applying filters. In CNNs, there are multiple types of hidden layers used.

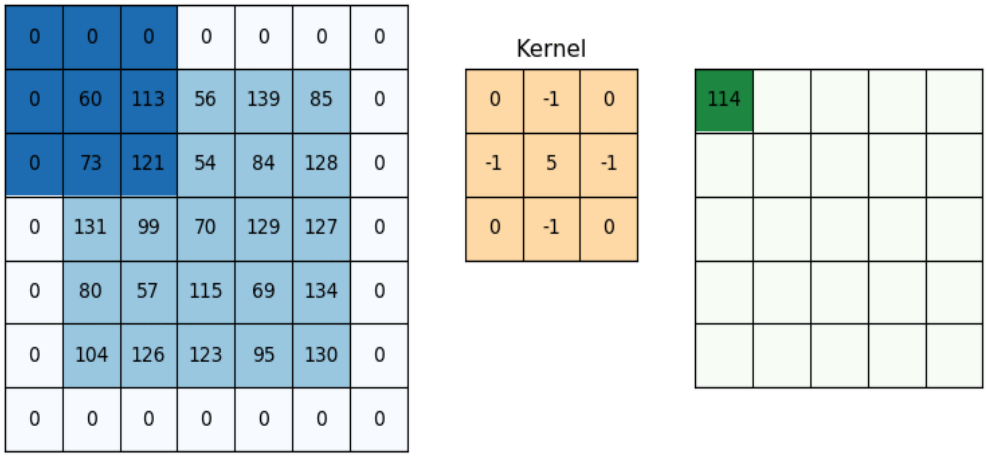


Figure 2: 2D Convolution operation with 3x3 kernel size. From [7]

The most common type of convolution that is used is the *2D Convolution Layer*. This filter scrolls through the input data and performs an element-by-element

multiplication. Then all these values are added together, and there is only one value in the result. The sliding process continues until our input is finished, repeated for each slide. As a result, our 2D feature matrix turns into a different 2D feature matrix.

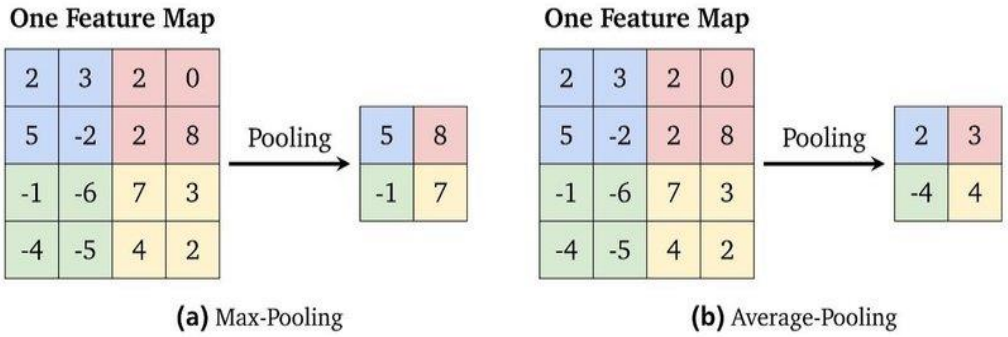


Figure 3: Example for the max-pooling and the average-pooling with a filter size of 2×2 and a stride of 2×2. From [8]

There are two types of *pooling layers*, which is another layer type. These are *max pooling* and *average pooling*. These layers are generally used to reduce the number of parameters and eliminate the computation load. For example, a kernel slides over the matrix in max pooling, and the maximum value is used on the layer's output. The process is repeated in average pooling, but the average value is used instead of the maximum value.

Fully connected layers are generally used after convolution layers or after pooling layers. When our parameters come to this layer, it should no longer be a 3D matrix. To be fed into fully connected layers, it must be turned into a completely one-dimensional vector. And this operation is called flattening. Pooling or convolution layer output is converted to a one-dimensional matrix by performing 3-dimensional matrix flattening.

2.1.1.1 Activation Functions

Activation functions in neural networks enable the received input to turn into output via nodes. Which activation function to choose in neural network applications can make a big difference in the model. At the same time, activation functions can be used more than once in any part of the model. Although neural networks are created to use the same activation function for each node in all its layers, activation functions can be implemented at the end or inside each node.

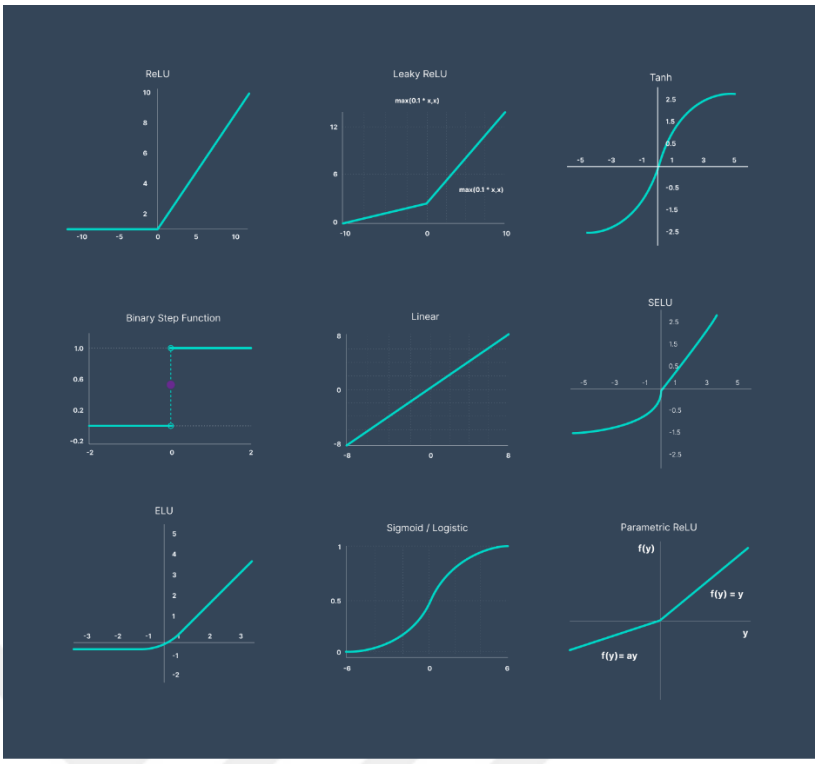


Figure 4: Activation Functions [9]

As mentioned before, neural networks consist of 3 main layers. Input layers, hidden layers, and output layers. An activation function is used in the output layer according to the prediction type in the model's output. And in general, a different activation function is used in the hidden layers than the activation function used in the output layer. Almost all the activation functions are differentiable. This is because neural networks are mostly trained with the backpropagation method, and in this method, the derivative of the prediction error must be taken to update the model weights.

In neural networks, hidden layers usually take the previous layer's output as input and transmit it to the next layer. And mostly, hidden layers do not encounter the model's input and output. There is no obligation to have a hidden layer in all neural networks. Generally, differentiable but non-linear activation functions are used in the hidden layers of neural networks. This is because it enables it to learn much more complex functions than a model trained using a linear activation function. The most used activation functions in the hidden layers of neural networks are;

- Rectified Linear Activation (ReLU)
- Logistic (Sigmoid)

- Hyperbolic (Tanh)

The layer that directly gives a prediction output in a neural network is called the output layer. All feed-forward neural networks have an output layer. The most used activation functions in the output layers of neural networks are:

- Linear
- Logistic (Sigmoid)
- Softmax

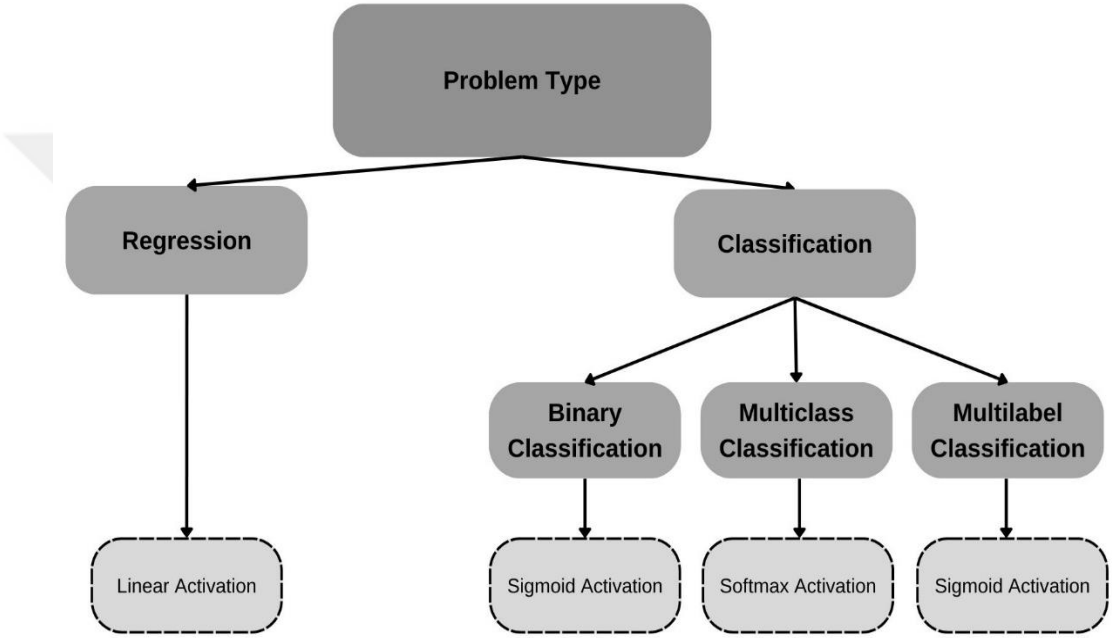


Figure 5: Activation function selection according to the problem type

The softmax function is a function that turns a vector of K real values into a vector of k real value that sum to 1 [10]. Even if the value in the softmax input is negative, the softmax function's task is to map this value between 0 and 1. So, it can be seen as probabilities obtained as a result. For example, if the incoming values are negative or very close to zero, this can be considered a low-probability prediction value. Unlike other methods, the softmax function outputs a normalized probability distribution after all operations, making it much more useful.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 6: Mathematical definition of the softmax function from [10]

2.1.2 Embedded Machine Learning

An embedded system is software running on a microcontroller. These systems have specific memories and can be programmable. Even though they are computing systems, embedded systems can have anything from no user interface (UI) to intricate graphical user interfaces (GUIs), like mobile devices. They typically include a processor, a power source, memory, and communication ports. This processor can be a microprocessor or microcontroller. The difference between a microcontroller and a microprocessor is having peripheral interfaces and integrated memory.

The term system on a chip refers to having multiple elements of a computer system on a single chip. In my thesis, the term SoC refers to the PYNQ-Z2 board, which includes an ARM processor and Zynq FPGA chip inside.

Machine learning allows us to teach computers to make predictions and decisions based on data and learn from experiences. From the past to the present, many optimizations have been made on both the software and hardware sides of machine learning [11]. This way, today's embedded systems can run these algorithms quickly, consuming little energy. And today, embedded devices can be much more efficient than cloud-based systems.

A lot of work has been done to make more efficient applications on FPGA. Y. Umuroglu et al. [46] created a streaming architecture called FINN to work more optimally on FPGA. Courbariaux et al. [47] showed that CNNs could give outstanding results when using binarized numbers instead of floating numbers by applying binarization. Nakahara et al. [48] implemented an object recognition application that can perform real-time processing on the FPGA. The proposed model executes CNN models on FPGA with its streaming structure. It also uses the ARM processor to calculate the softmax function and deliver the inputs to the FPGA.

These days, especially the big data companies use has increased a lot. And this big data can sometimes be very complex or big enough to force systems [12]. This can cause significant problems for embedded systems. Embedded devices have limited

memory, processing power, energy capacity, and more. With some methods, in some cases, this data can be made available for embedded systems such as quantization.

2.2 HYBRID AI

Symbols are things we use to represent other things and have an essential place in people's thinking and perception mechanisms. Symbolic AI is an approach that trains Artificial Intelligence the same way the human brain learns. Understands concepts with symbolic representations, unlike other methods [13].

Hybrid AI can be developed for different functions in different areas. For example, this field may be related to weather models, as in Rodriguez's study [49], while it can be used to obtain optimal management methods, as in Conte's study [50].

There is an apparent distinction between neural networks and symbolic AI applications. Symbolic AI applications need strict rules, and any incoming input can be easily converted into a symbol. But the problems encountered in daily life are not situations with strict rules. That's why symbolic AI doesn't always succeed in everyday world problems. Neural networks have become much more useful than Symbolic AI in this regard.

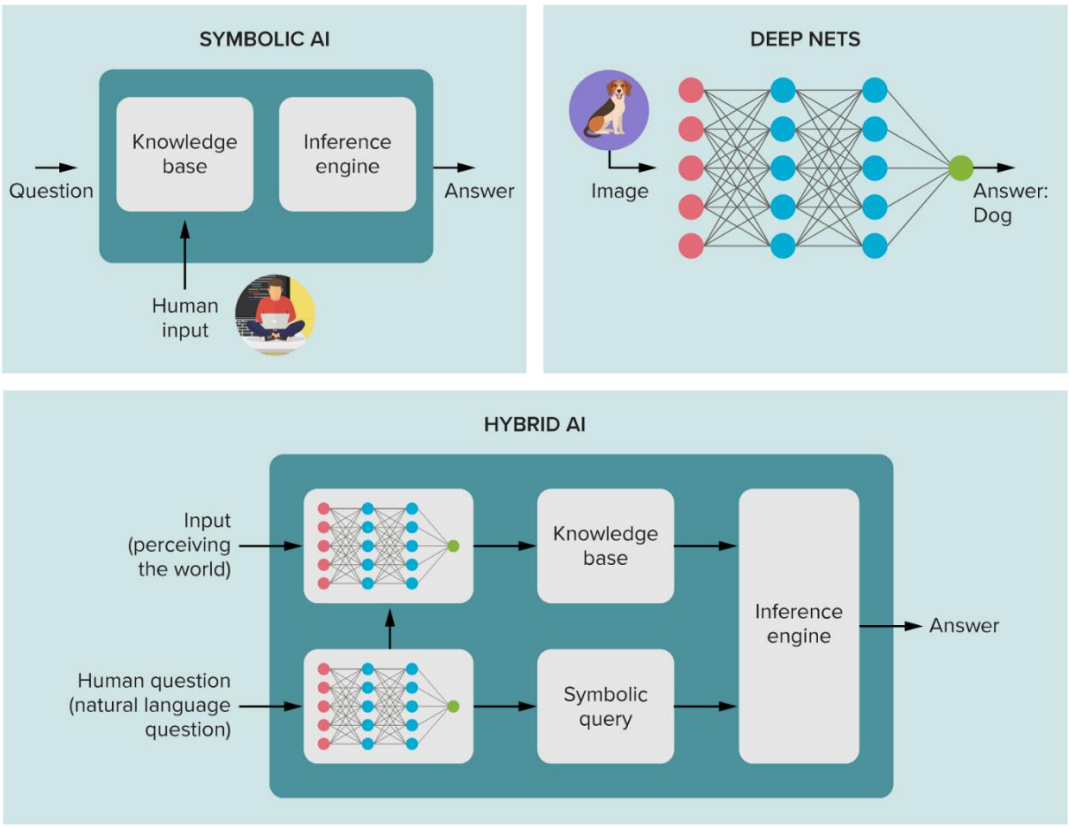


Figure 7: Symbolic AI (top left) knowledge base structure created by the user providing input and having AI respond. In deep networks (upper right), the strength of the connections between the layers is adjusted and trained to reach the correct answers. The hybrid uses deep networks to create the knowledge base from which the information is provided instead of only the user in that section. From Knowable Magazine [13]

Recent research has focused on an approach called Hybrid AI that combines the strengths of symbolic AI with neural networks. Hybrid AI allows neural networks to extract patterns combining substantial data sets. Then, rule-based AI systems can manipulate the retrieved information using algorithms to manipulate symbols.

2.2.1 RPNI Algorithm

The RPNI algorithm generates a transition map using finite state machine structures with any incoming symbolic sequence inputs. A different path is created for each symbolic input sequence to the RPNI algorithm. At the end of each path (the length of the path depends on the size of the incoming input sequence), the "True" state specified for that input is reached. For each element in the input, a different state is transitioned, and as a result, a transition map is produced according to which states it goes through. This way, it can be used instead of a decision-making mechanism in

deep learning applications by creating "True" states. The size of this map changes according to the number and length of the incoming input.

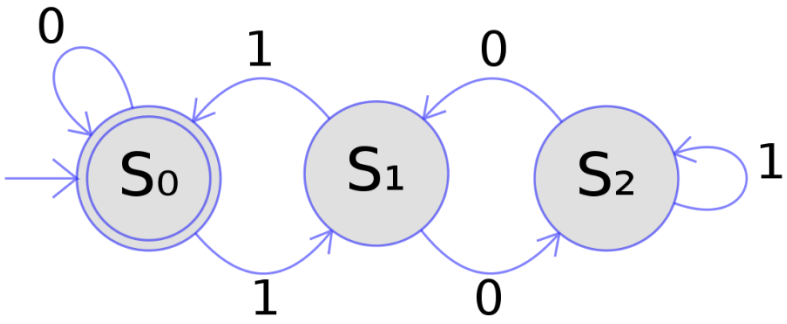


Figure 8: Finite State Machine

Using this algorithm, we observe how a neural network model trained on an embedded device eventually perform when used instead of FC layers (which is the decision-making mechanism).

2.3 METHODS TO MAKE THE NETWORK EFFICIENT

With deep learning methods becoming more accessible and developing day by day, it can be observed that more users or companies are starting to use these applications [14]. The development of neural networks solves more complex machine learning problems than before. However, using neural networks alone may not be a solution sometimes. Therefore, more intelligent solutions must be found and implemented when applying neural networks. It will become useless if some parameters are not chosen wisely in implementing the neural network.

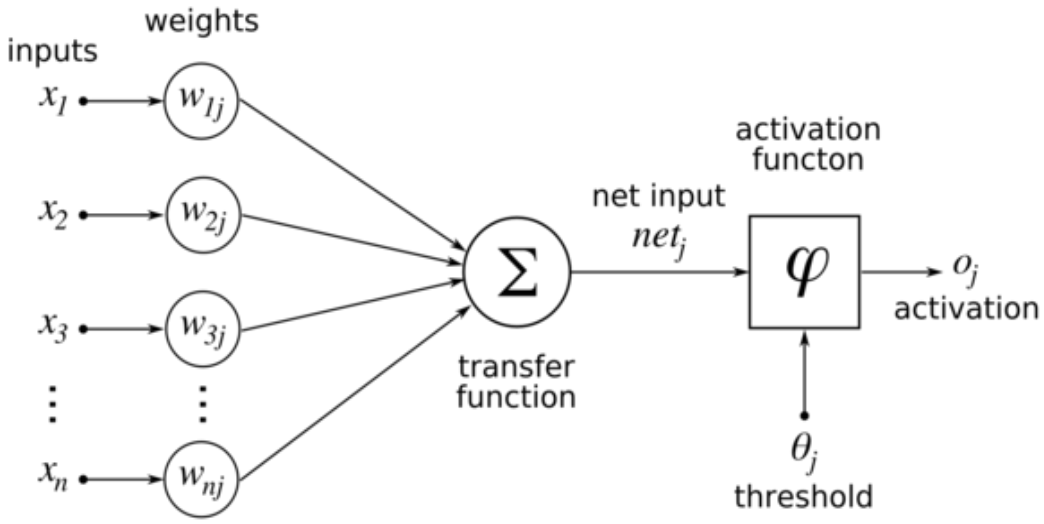


Figure 9: Artificial Neuron Model

Sometimes even the slightest change to the network can make a huge difference in output. In this section, it is mentioned which methods are essential for neural networks to be more efficient.

2.3.1 Quantization

While training and designing machine learning models is a topic, optimizing these models in the best way is an entirely different issue. Computational and memory resources spent by neural networks have increased noticeably from past to present. Especially projects involving extensive data can cost a lot. Therefore, the training and inference of deep neural networks need to be optimized at some point. When work is moved from servers to the edge, size and computational complexity must be resolved. And at this point, the quantization process is very logical and useful.

The primary purpose of network quantization is to provide ease of operation and consume less memory by converting weight and activation values to integer values. But there is a trade-off at this point, as in all matters. If the weights and activations are converted to integer values, there is likely to be a substantial decrease in the accuracy of the results. In the quantization process, the idea is quite simple. It is necessary to scale all possessed numeric values within a specific range and round this numeric value to a lower number. As shown in the example below.

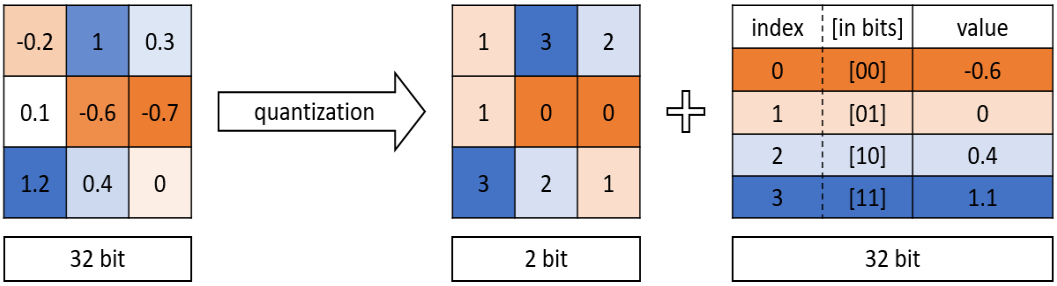


Figure 10: 2-bit weight quantization and a codebook mapping index to weight values from [15]

As seen in the example, reducing the values that can be expressed with 32 bits to 2 bits by applying a hard quantization process be very helpful in terms of ease of operation and memory. But there is also a large amount of information loss. Therefore, using different quantization techniques makes it possible to both gain speed and not be affected by a tremendous information loss. In practice, two possible quantization techniques can be applied.

2.3.1.1 Binary Quantization

The binary quantization method converts all weights and activations to binary numbers. Because of the operational speed of FPGAs working on the binary values, if you are making AI applications on an FPGA, you can achieve much more efficiency with this method than you would get on a CPU or GPU.

XNOR-Net [16] proposed two efficient approximations. The first one is Binary-Weight-Networks which offers 32x memory saving while approximating the filters with binary values. The other one is XNOR-Networks which offers 58x faster convolutional operations while approximating the convolutions using binary operations.

There are mainly two functions to binarize the values, deterministic and stochastic binarization. XNOR-Net uses deterministic binarization in CNN layers, while BinaryConnect [17] uses a stochastic binarization approach, which is a different method. In deterministic binarization, the signum function is used on real values.

$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise,} \end{cases}$$

Figure 11: Signum Function

In stochastic binarization, the hard sigmoid function is used.

$$x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

Figure 12: Hard sigmoid function

2.3.1.2 Post Training Quantization

It is the most straightforward quantization technique to implement. As the name suggests, the logic follows: Quantize the weights after training the weights and inputs in float32 format. Thus, a quantization operation is easily applied, which cause a loss of accuracy [18].

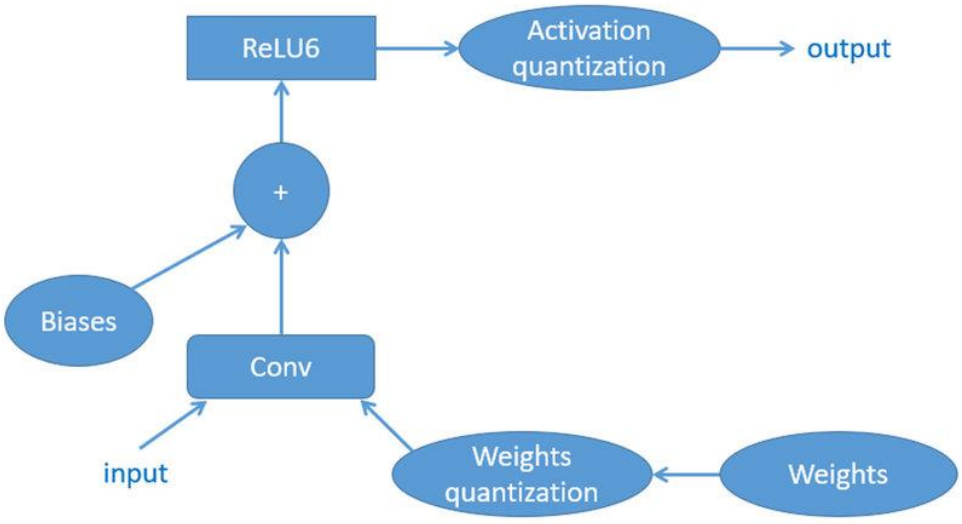


Figure 13: Post-training quantization scheme from [18]

2.3.1.3 Quantization-Aware Training

Although applying this process is a little more challenging, the method yields the best total results. The procedure done here is that the weights are quantized during training. In practice, int8 quantization provide the best results because the processors are designed much faster when performing integer operations [19].

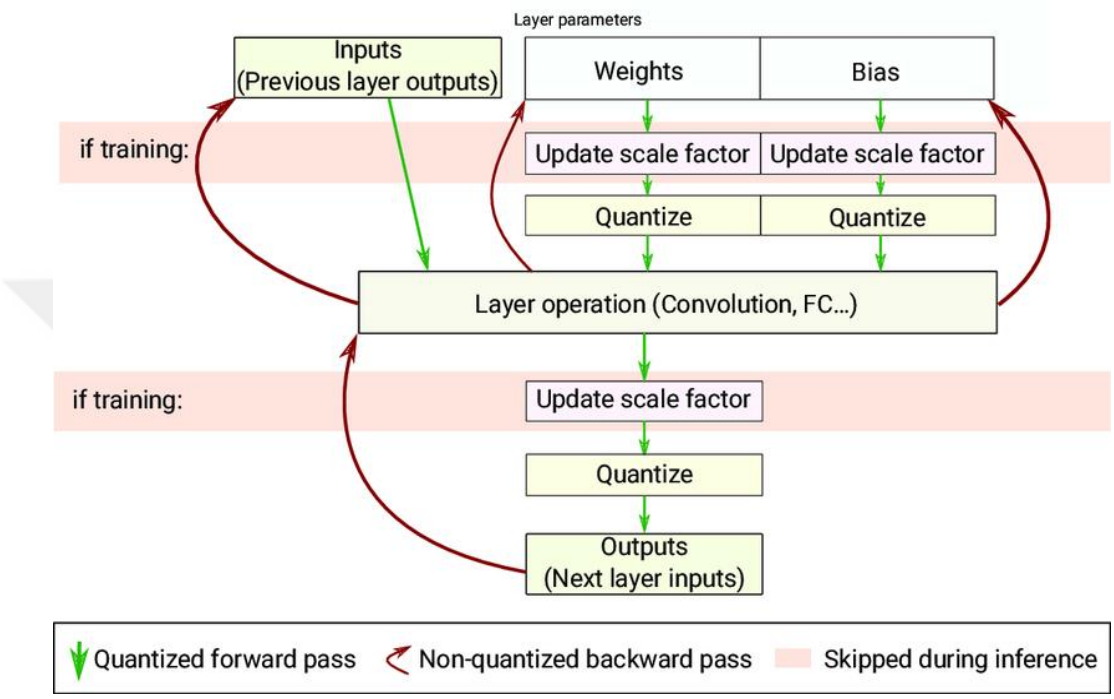


Figure 14: Quantization-aware training scheme taken from [19]

2.3.2 Channel Pruning

The channel pruning technique is one of the structured simplification techniques [36]. Tensor factorization [20], sparse connection [21] and channel pruning [22]. While other structured simplification techniques do not deal with the number of feature maps, the channel pruning technique aims to reduce the feature map width. And this technique is also very effective on embedded systems because it doesn't need any extra implementation.

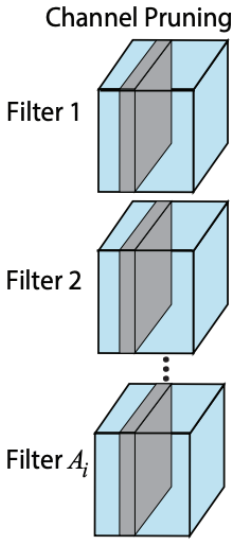


Figure 15: Channel Pruning from [23]

2.3.3 Using More Data

Overfitting is a significant problem when training neural networks. The training data to be used varies according to the problem encountered. So, there is no such thing as an optimal amount of data. If there is not enough data for the problem to be solved, then most likely, it may overfit the data.

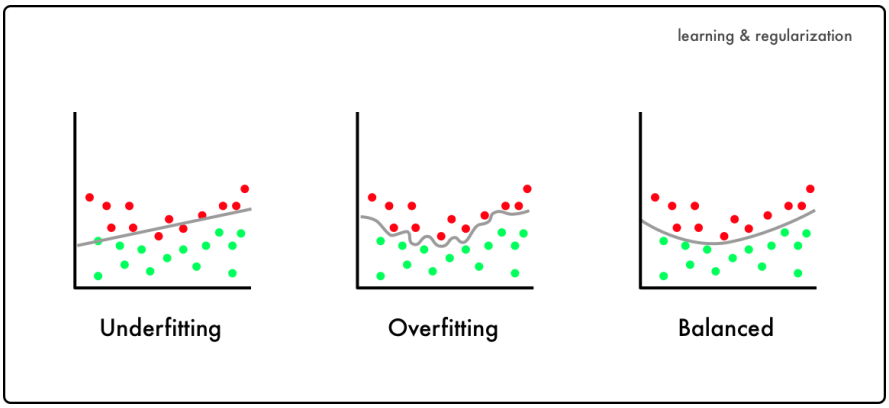


Figure 16: Underfitting, overfitting, and balanced data

If there is a very long training phase or if the model is too complex, irrelevant information starts to be learned, and it is observed that the validation error increases in a meaningless way. And if the model fits this unrelated data too close to the training set, the model becomes "overfitted" [24].

Some methods can be done to avoid overfitting. It can be an early stopping method. This method requires stopping the training before the model starts learning

irrelevant information. This time, however, an underfitting situation may be encountered while trying to avoid the overfitting situation. It is necessary to try to stop at the most efficient moment between these two points.

This situation can be avoided by using more data. Expanding the training set with efficient and relevant information can strengthen the relationship between input and output, allowing us to achieve more accurate results. It's essential to add data related to training data; otherwise, the complexity of the model and the overfitted status may increase. For example, data augmentation [25] is an excellent method to add relevant data.

The feature selection method can help avoid overfitting and increase the model's efficiency [26]. It is confused with the dimensionality reduction method in many ways. To predict an output, there are several parameters or features. Not all these features are required in some cases. Feature selection is a method in which the most important (best quality) features are selected according to their variance ratios, and the rest are discarded. There are different feature selection algorithms, and some are used for this thesis work. In Chapter 4, this is mentioned in more detail.

Figure 17: Feature selection taken from [27]

2.3.4 Changing Learning Parameters

Stochastic gradient descent is used to train deep learning neural networks. One type of optimization algorithm is stochastic gradient descent. It calculates error gradients in the current state using samples from the training set. After that, it updates the weights of the model with the backpropagation. The amount of weight updated during training is called the learning rate. And this learning rate can be configured in the training phase [28]. With the learning rate, instead of updating a whole weight at

once, it can be scaled according to the rate. For example, this means that 10% of the estimated weight error is updated every time the weights are updated.

Usually, choosing a small learning rate increase the training time but provide a more optimal learning method. A more significant learning rate allows for faster training and results in a non-optimal weight set.

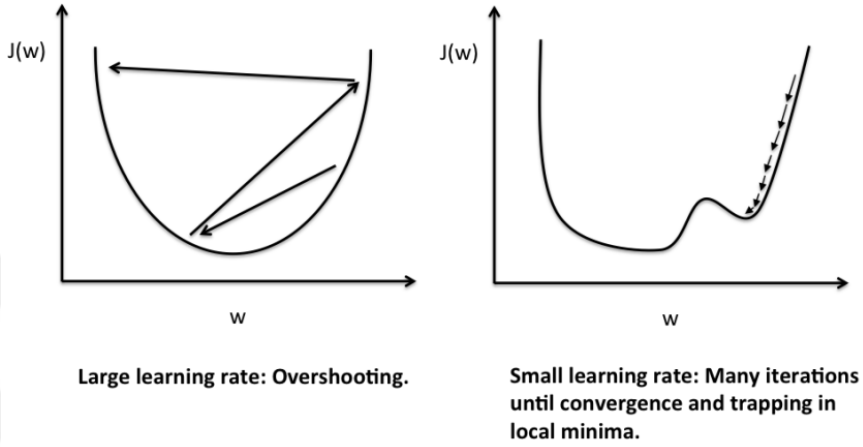


Figure 18: Effect of learning rate parameter values taken from [29]

Using different learning rates, it can be observed whether it gets stuck at the local minima. Also, changing the learning rate may improve the accuracy of training results. The images below visualize stuck in local minima and overshooting.

2.4 HARDWARE

This section covers preference priority and availability of different embedded devices over Neural networks. There is also an examination of the characteristics, strengths, and weaknesses of the board used in the thesis study. There are three main hardware options for AI applications. These are field programmable gate arrays (FPGAs), graphic processing units (GPUs), and central processing units (CPUs).

2.4.1 What is an FPGA?

FPGAs, also known as field-programmable gate arrays, have been used for several decades. FPGAs have an array of logic blocks, and users can program these blocks. To program an FPGA, you must use one of the HDLs (Hardware Description Languages) Verilog or VHDL.

[30] Intel has published research to see if next-generation FPGAs can compete with current NVIDIA GPUs. Intel demonstrates how FPGA AI accelerators perform better than an ordinary NVIDIA GPU. Using quantized compact data rather than 32-bit floating numbers makes this possible.

FPGA does not always show much better results in all AI applications. Before parallel task processing units like GPUs, it is nearly impossible to use neural networks for practical tasks. Because deep learning models need so many resources and complex models. To be successful in real-world applications, it is necessary to provide parameters such as flexibility, low latency, and low energy consumption [31][32]. And exactly at this point, FPGAs provide the best performance in AI applications, both in terms of performance and providing these parameters. An FPGA's ability to be reprogrammed provides the flexibility needed by artificial neural networks' dynamic structure. FPGAs also offer the custom parallelism and high-bandwidth memory for real-time inferencing of a model.

2.4.2 FPGA vs. CPU vs. GPU

The central processing unit (*CPU*) is the standard processor used in many devices. The CPU is the most straightforward unit that sequentially processes and completes the operations performed in machine learning applications [31][38].

Pros:

- Easy to access. It can be found almost anywhere.
- The most flexible and easy-to-use devices.

Cons:

- The biggest weakness for deep learning applications is their raw computational performance.
- CPUs are optimized for sequential processing with limited parallelism.

GPU is a processor architecture designed explicitly for graphical computing. It has more CU and ALU units according to the structure of the CPU architecture. In addition, commands are processed in parallel. In this way, it has more efficient and faster performance in graphical calculations.

Pros:

- Parallel computation, high-speed.

- Composed of hundreds of cores.

Cons:

- Weak data/memory access.
- High power consumption.

FPGA stands for field-programmable gate array. An FPGA is a hardware circuit that a user can program to carry out one or more logical operations [44].

FPGAs consist of three main parts:

- Configurable Logic Blocks
- Programmable Interconnects
- Programmable I/O Blocks

Pros:

- High-speeded parallel computation
- Energy efficiency
- High performance

Cons:

- Hard to configure
- Size of the dataset is limited

Apart from these, another concept that is not very popular but gain popularity in embedded machine learning applications day by day is **SOCs**. Deep learning is changing the structure of SoCs and bringing new generation investments into the semiconductor market. When the design of SOCs is considered, it combines a highly optimized CPU and a highly optimized FPGA. It incorporates both, and these structures are used in different functions in machine learning applications [33].

CHAPTER III: EMBEDDED ENVIRONMENT

The board selection brings some requirements. This section discusses the steps to follow to implement a quantized network on a Pynq board. I also mention which environments you should be familiar with when using the PYNQ board.

3.1 ENVIRONMENTS

This section gives which environments should be installed to get the system up and running.

3.1.1 Jupyter Notebook

The Jupyter Notebook is an interactive computing environment. This Notebook web application allows you to edit and run codes in the browser. And this Notebook supports different programming languages like Python, R, Ruby, Go, etc.

3.1.2 Google Colab

Google Colab is a free Jupyter Notebook environment that runs entirely in Google cloud servers. It supports the most popular machine learning libraries. Data and results are stored in Google Drive because they are accessible from Colab, and Google Colab is used to retrain FC layers and observe graphs.

3.2 EMBEDDED ENVIRONMENTS

3.2.1 Selection of Hardware (PYNQ-Z2)

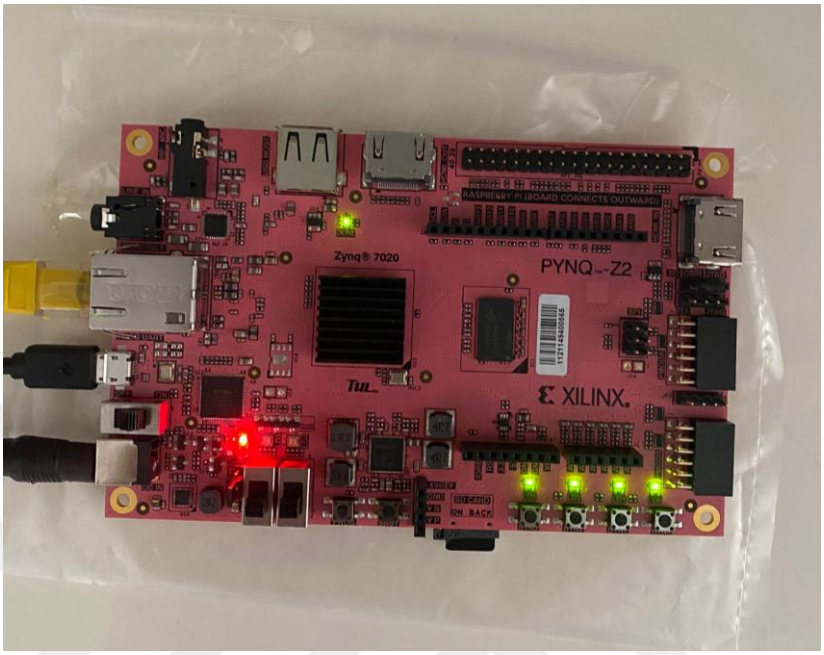


Figure 19: Pynq Z2 Board

PYNQ-Z2 is a SOC development board with ZYNQ XC7Z020 FPGA and Arm processor, developed by Xilinx University Program. With this SOC, you can perform operations without designing the programming logic circuits on the FPGA, which provides great convenience. Also, one of its very powerful features is being able to program, edit or test the PYNQ-Z2 board directly in Python.

Table 3.1: Comparison of PYNQ Z1, PYNQ Z2, and ZCU104 FPGA-based boards on different features. Information is taken from [34]

	PYNQ-Z1	**PYNQ-Z2**	**ZCU 104**
Device	Zynq Z7020	Zynq Z7020	Zynq Ultrascale+ XCZU7EV
Memory	512MB DDR3	512MB DDR3	2GB DDR, PL DDR4 SODIMM
Storage	Micro SD	Micro SD	Micro SD

Table 3.1 Cont.

Audio	PDM integrated mic, 3.5mm PWM audio jack	ADAU1761 codec with HP + mic	-
Video	In & Out HDMI	In & Out HDMI	In & Out HDMI, Display Port
Network	10x1, x10, x100 Ethernet	10x1, x10, x100 Ethernet	10x1, x10, x100 Ethernet
Expansion	USB host (PS)	USB host (PS)	USB 2.0/3.0 host (PS)
GPIO	1x Arduino Header 2x Pmod 16x GPIO Pins	1x Arduino Header 2x Pmod 1x RasperryPi	LPC FMC 3x Pmod (2x PL)
Other	6x User LEDs 4x Push Buttons 2x Dip switches	6x User LEDs 4x Push Buttons 2x Dip switches	4x User LEDs 4x Push Buttons 4x Dip switches
Other	Enables to program the onboard SoC with Python	Enables to program the onboard SoC with Python	-

We decided to use the Xilinx PYNQ Z2 board, and there are some reasons why we chose this board. We chose this option because we wanted to make comparisons on tools such as BNN-PYNQ and QNN-MO-PYNQ developed by Xilinx. To work with these tools, you must use the PYNQ board. In addition, running the codes written in python directly on the board provided great ease of operation.

3.2.2 Pynq Getting Started

There are some prerequisites for installing and using the Pynq board. First, an ethernet, micro-USB cable, and a minimum 8 GB micro-SD card are required. To install the PYNQ board, the required image file must be loaded into the micro-SD card beforehand. To use the QNN-MO_PYNQ tool, image version 2.3 or higher needs to be installed. Version 2.5 has been installed.

And since the operating system supported by this version is Ubuntu 18.04, we installed Virtual Box on the host Windows10 operating system. After making the necessary adjustments on the virtual machine, Vivado 2019.1, SDK 2019.1, and PetaLinux 2019.1, which are Xilinx Tools suitable for the version, are installed. After meeting these requirements, the image building part is completed.

To access the jupyter notebook via the board, the board must be on the same internet network as your computer. This way, clipboard updates and improvements can be made on the computer. On the board, a file-sharing program called Samba is active. You can access the Pynq home directory and transfer files to and from the board using it as a network drive. Since the Pynq board is not very large, a limited size should be uploaded

.

CHAPTER IV: EXPERIMENTAL SETUP

The implementation of artificial neural networks on embedded devices can be divided into two general parts: software and hardware. In this thesis process, we needed more theoretical knowledge on the hardware side, but we did more work on the software side.

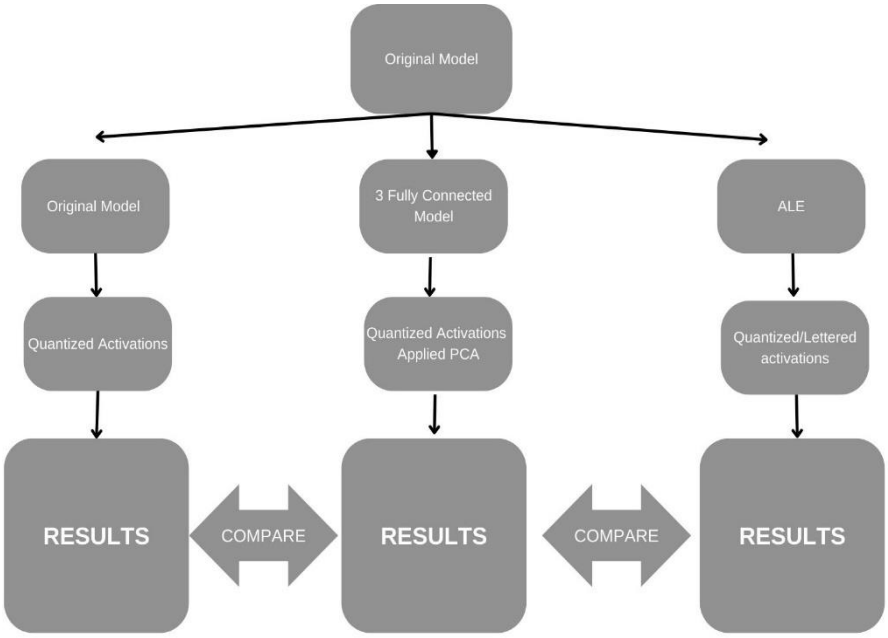


Figure 20: Overview of experiments

4.1 QUANTIZED NEURAL NETWORK PROPERTIES

Xilinx's QNN-MO-PYNQ package enables us to build a Quantized neural network on PYNQ boards using Multi-Layer Offload architecture [35][37]. Two different options are offered for the quantization level. The first is W1A2 (1-bit weights, 2-bit activations), and the other is W1A3 (1-bit weights,3-bit activations). It offers the ability to automatically download the bitstream to the PYNQ device using the available classifier. At the same time, with this designed classifier, memory buffers can be allocated, and hyperparameters and weights of the pre-trained network can be loaded.

In addition, instead of automatically installing the software and hardware builds, you can do it manually. The software building process has two ways: it can be built as a library or testbench. If testbench is used for software build, it is possible to do this process directly from the command line. But if a library is used, then a python jupyter notebook should be used to load images to the board.

Quantized Neural Network features are the first steps of the software part of this thesis. A pre-trained quantization neural network provided by Xilinx, a variant of DoReFaNet [39], used for the experiments. The network topology is illustrated in the following picture.

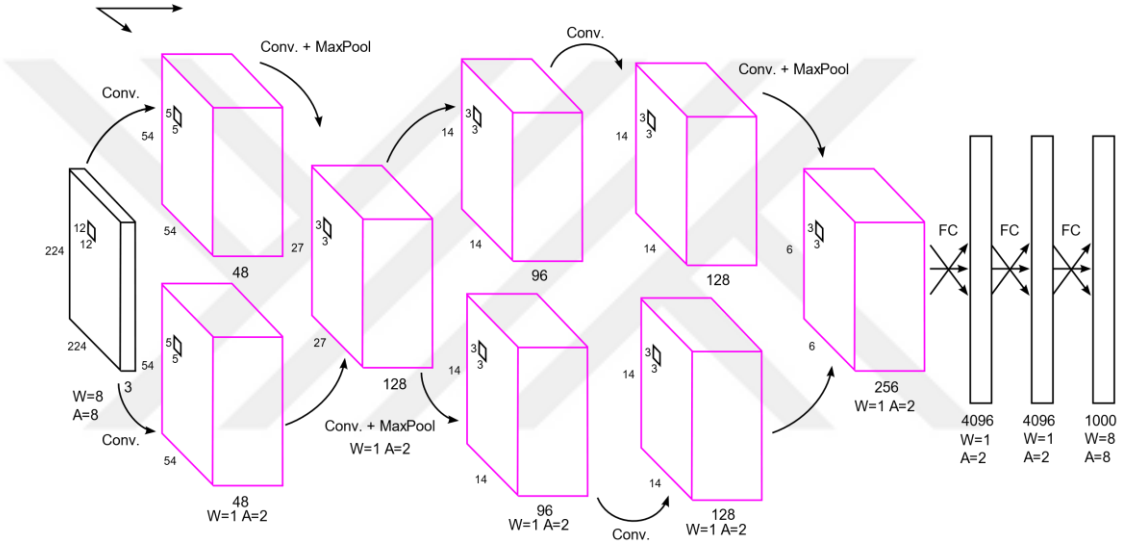


Figure 21: Xilinx QNN-MO-PYNQ version of DoReFa-Net from [37]

This is a pruned version of the DoReFa-Net network, trained on the ImageNet dataset with 1-bit weights and 2-bit activations. It contains 5 Convolution layers and 3 Fully Connected layers. It also has split and merge layers. The pink layers are executed in FPGA, while the others are performed in Python.

DoReFa-Net proposed a method to train convolutional neural networks with low bitwidth weights and activations using low bitwidth parameter gradients. The technique uses bit convolution kernels and a straight-through estimator [40].

4.2 IMAGE CLASSIFICATION APPLICATIONS

This section describes how the data set prepared, how problems encountered from the first step to the last step of the experiments, and what methods used to solve them.

4.2.1 Data Preparation

First, the classifier is created to automatically download the data stream to the device. Bitstream is a file containing the programming data associated with FPGA. And on here, bitstream allocates memory buffers and loads network parameters and weights. The neural network is specified in a JSON file provided by Xilinx. And the weights that won't be downloaded to the chip are loaded into a NumPy dictionary to be used for execution in Python.

Table 4.1: Comparison of the original and customized dataset

	Original Dataset	Customized Dataset
Total Images	1.028.000	100.000
Total Classes	1000	1000
Images per class	Unevenly distributed	100

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 is then used for the dataset. The number of images for each class has been reduced because there is too much data in ImageNet. In total, there are 1,028,000 images organized in 1,000 categories.

```
# Top-5 results
topn =  utils.get_topn_indexes(out, 5)

if img_class in synsets.keys():

    if synsets[img_class] in (names[k] for k in topn):
        print("\nMatch!")
        OK=OK+1
        print(OK)
    else:
        NOK=NOK+1
        print(NOK)
else:
    for k in topn:
        accuracy = ((OK)/(OK+NOK))*100
        print(accuracy)
```

```
accuracy = ((OK)/(OK+NOK))*100
print(accuracy)
```

```
81.39999999999999
```

Figure 22: Code pieces of accuracy consistency check and accuracy score

After customizing the dataset, 1 sample from each class and 1,000 test images are taken, resulting in the first 5 accuracy scores of 81.4%. The result is compared to the AlexNet top 5 accuracy rates on the Imagenet dataset for accuracy consistency.

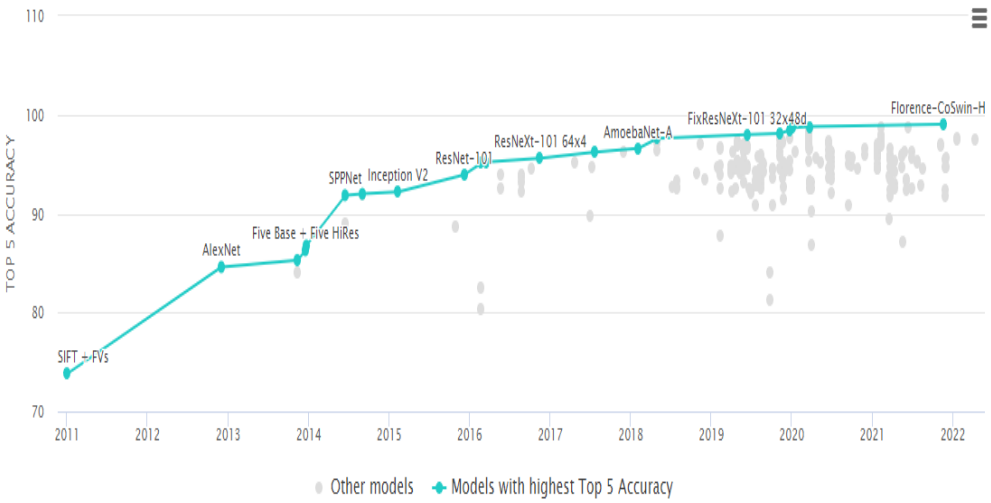


Figure 23: Image classification Top-5 accuracy results on Imagenet dataset from [41]

When there is no doubt about the accuracy consistency, it is possible to move on to the next experiment.

4.2.2 Getting Activations from the Original Model

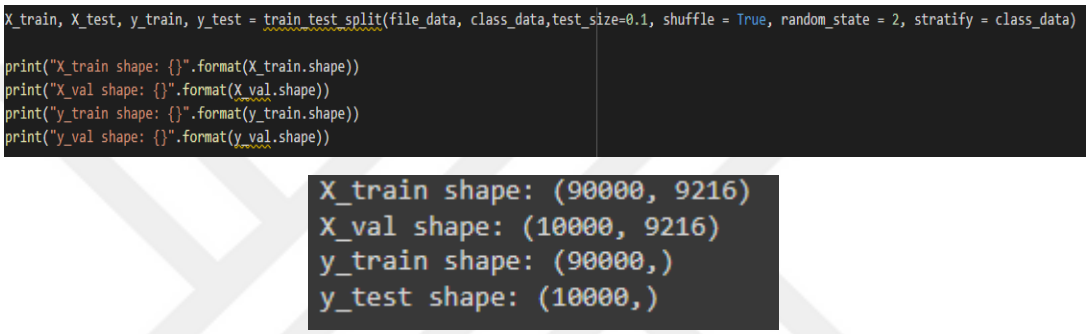
There are two most popular ways to save activation values from the original model to a text file. These values can be easily saved using the pandas or NumPy libraries. Activations with a total sequence size of 100,000 x 9216 are obtained using the Numpy library. There are some difficulties with working on an embedded device at this stage.

First, we aim to take 1000 samples from each class and process a set of 1,000,000 pieces in total. But unfortunately, such a large operation cannot be done on the PYNQ embedded device simultaneously. Therefore, 100,000 samples are divided into 10 files during the experiment, and the experiments are performed separately. In the end, the obtained results are combined again. If working with 1,000,000 datasets, the number of files to split would be 100, which is nearly impossible with current clipboard resources. Quantized activations before the 3 FC layers at the end of the

model are obtained separately for the training set (100,000 x 9216) and the test set (1000 x 9216).

4.2.3 Making the Activation Values Suitable for the Experiments by Using Different Methods

Activations from the original model are modified differently to be used in different experiments. To train a fully connected layer consisting of 3 layers, previously obtained activations are split into validation (10,000 x 9216) and training (90,000 x 9216) sets. A test set (1.000 x 9216) contained 1 sample from each class.



```
X_train, X_test, y_train, y_test = train_test_split(file_data, class_data,test_size=0.1, shuffle = True, random_state = 2, stratify = class_data)

print("X_train shape: {}".format(X_train.shape))
print("X_val shape: {}".format(X_val.shape))
print("y_train shape: {}".format(y_train.shape))
print("y_val shape: {}".format(y_val.shape))
```

```
X_train shape: (90000, 9216)
X_val shape: (10000, 9216)
y_train shape: (90000,)
y_test shape: (10000,)
```

Figure 24: Code pieces of splitting train and validation data

In the Hybrid AI experiment, on the other hand, it is necessary to modify all the 100,000 activations in a row, without spaces, and by assigning a letter instead of each quantized number. And at the end of each line, the number indicating which class it belongs to should be written with a space.

```
acaaaaadcaa 939
adaaaaddaab 939
acaacaaccaa 939
adaaacbbaaa 939
abbbcacacac 939
```

Figure 25: 2-bit Quantized Hybrid AI activations

	0	1	2	3	4	5	6	7	8	9	...	9207	9208	9209	9210	9211	9212	9213	9214	9215	class
0	3	3	1	1	3	3	3	3	3	3	...	1	0	0	0	1	1	2	0	0	475
1	3	3	1	0	2	3	3	2	3	3	...	1	0	2	1	3	2	2	0	0	475
2	3	3	0	0	3	3	3	3	3	3	...	3	0	0	0	0	2	1	0	0	475
3	3	3	0	0	3	3	3	3	3	3	...	0	0	0	1	0	1	1	0	0	475
4	3	3	0	0	3	3	3	3	3	3	...	1	0	0	0	0	0	2	0	0	475

Figure 26: 2-bit Quantized FC model activations

4.2.3.1 Principal Component Analysis

Principal Component Analysis or PCA turns a large set of variables into a smaller set that contains most of the information. Thus, this method is often used as a dimensionality reduction method to reduce the dimensionality of large datasets. [42].

And it could only help analyze how much information would be lost as features are reduced because PCA does feature extraction and means that, which reduces feature sizes but derives new values for features. And this principle of operation is not suitable for preparing input for hybrid AI in one of the experiments below. Compared with hybrid AI, quantized values should not change their quantized states.

The variance ratios of our 9216 features in total are as follows:

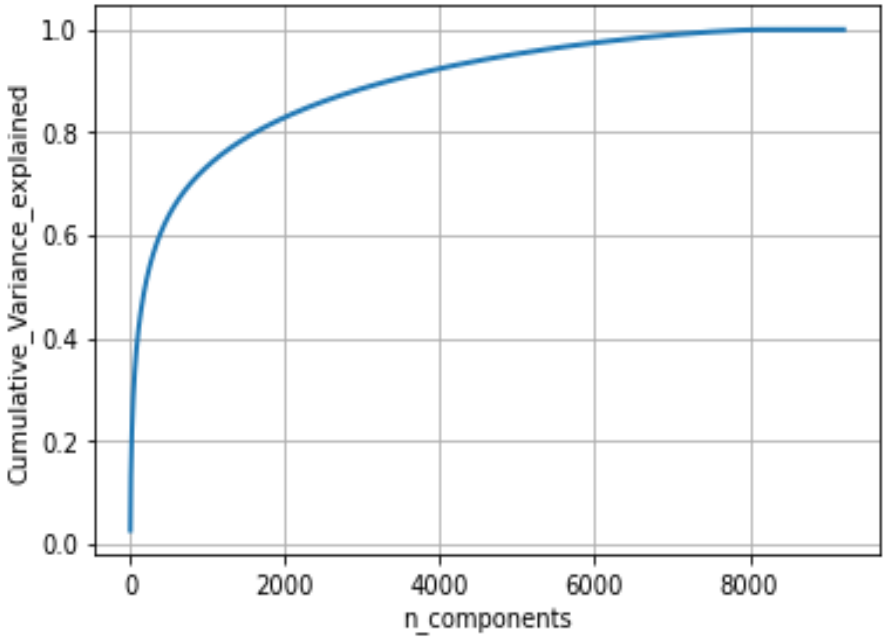


Figure 27: Cumulative variance ratio by number of feature size

This figure shows how much information can be lost in feature extraction experiments from 0 to 9216 feature sizes. The working principle of feature extraction with PCA is as follows:

The first step is standardizing the range of the input values so that each value contributes equally to the analysis. After that, the covariance matrix is computed because sometimes values could be very similar to each other, and this causes them to store some redundant information. Then the identify the principal components; eigenvalues computation is performed. And the last step is ordering the eigenvectors by their eigenvalues in descending order. In this step, it is chosen to keep or not discard some values.

4.2.3.2 Select K Best Features

Scikit-learn API [45] provides SelectKBest class for extracting the best features of a given dataset [43]. SelectKBest selects the features according to the k highest score. This function is used for preparing a large dataset for classification training.

```
bestfeatures = SelectKBest(score_func=chi2, k=1000)
```

Figure 28: SelectKBest code line

For classification, the score function is selected as “chi2” and the k parameter defines the number of features. Top 1000 features selected in our training data (100,000 x 9216). The reason for wanting to access the first 1000 features using this method instead of PCA is to detect and use only the highest value features without changing the quantized activation values from the original model.

4.2.4 Training FC Model with the Activations

We use the Keras library when training the FC model. Although less customizable, it is sufficient for simple training - it used 64 batch sizes and 100 epochs. The dimensions of the 3 Fully Connected layers in the original model and extracted in the new experiment are 4096, 4096, and 1000, respectively. While training these new 3 FC layers, these values had to change, and as a result, 3 FC layers are added in 1024, 1024, and 1000 sizes, respectively.

Figure 29: Previous model's FC layers and re-trained FC layers

By applying the PCA and the Select-K-Best function separately, different training results are obtained for each component size, and the test set activations obtained from the original model are used. In Chapter 5, the results are analyzed in detail.

CHAPTER V: ANALYSIS OF THE RESULTS

In this section, the results of the performed experiments are presented with a benchmark of the utilized methods and parameters.

The results of the original activations with a feature size are 9216 are as follows:

Top-5 Accuracy = % 48.6

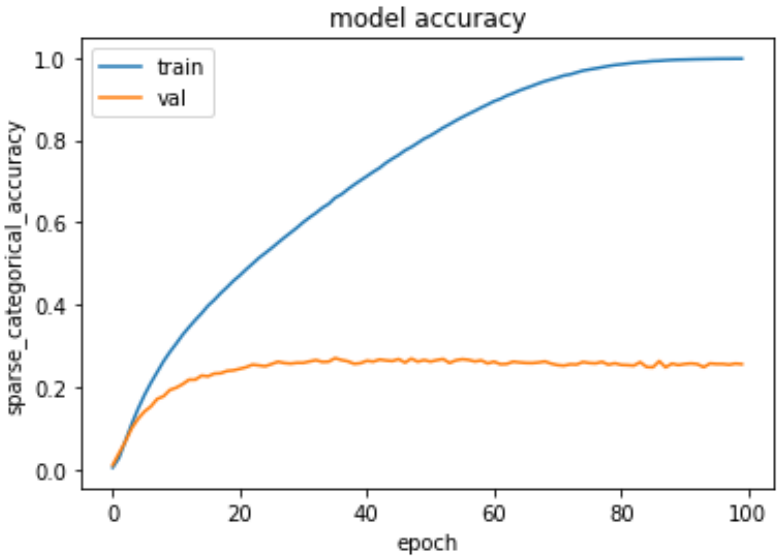


Figure 30: Model accuracy with 9216 feature size

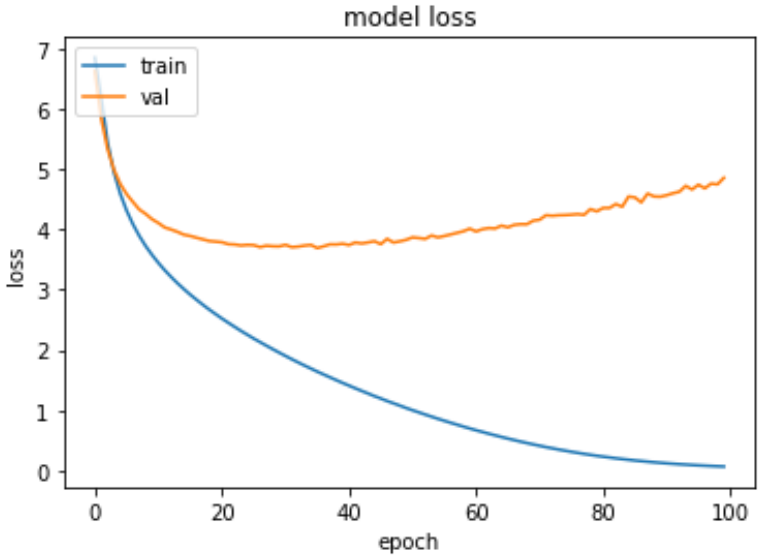


Figure 31: Model loss with 9216 feature size

The model accuracy graph shows that the training curve reaches approximately 99%, while the validation curve remains at only 25%. The top5 accuracy value obtained with the test data consisting of 1000 samples, obtained by taking one from each class, is 48.6%. The early stop method can be applied, but in this case the model accuracy could be very low. This experiment with a total of 100,000 samples shows that training results could be better if the number of samples could be increased.

5.1 RESULTS OF ACTIVATIONS WITH PCA

In this part, the results of the experiments with the features obtained using the PCA method are compared.

When the feature size is reduced to *1000 features* with the PCA method, the results are as follows:

Top-5 Accuracy = %45,1

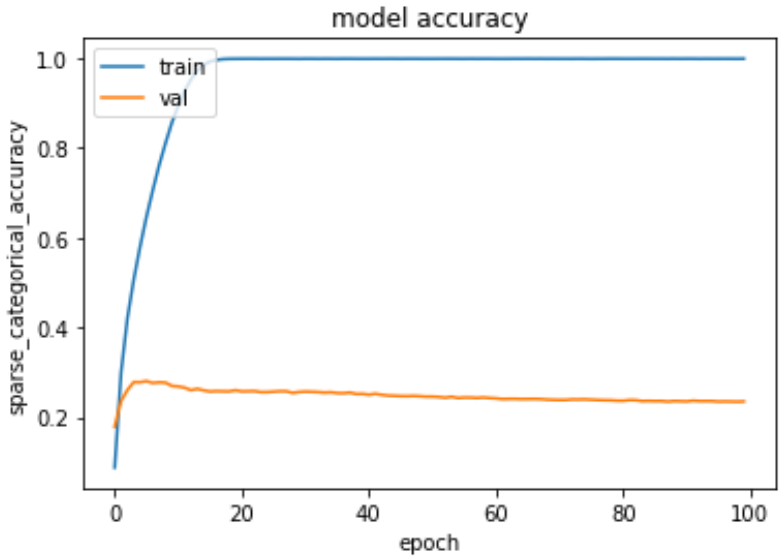


Figure 32: Model accuracy with 1000 feature size

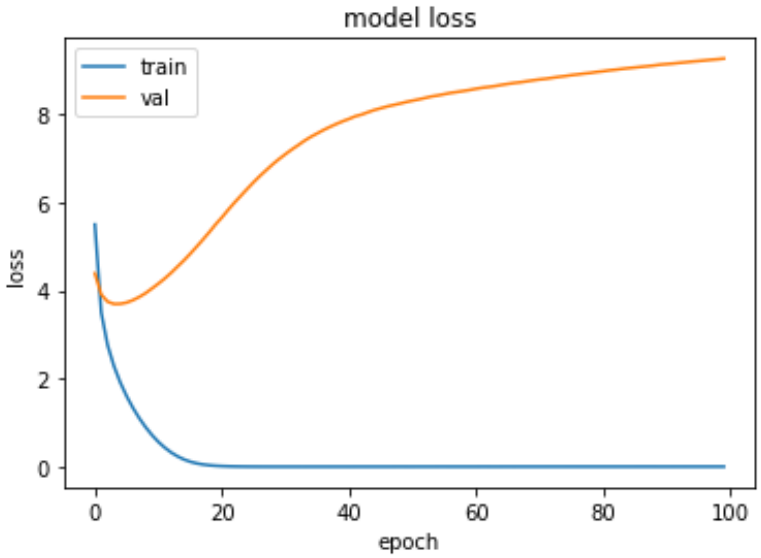


Figure 33: Model loss with 1000 feature size

There is a low decrease in Top5 accuracy when the feature selection is applied by selecting the best 1000 features with feature sizes totaling 9216 using PCA. However, when we look at the model loss graph, the experiment starts to overfit very quickly.

When the feature size is reduced to *3000 features* with the PCA method, the results are as follows:

Top-5 Accuracy = %43,6

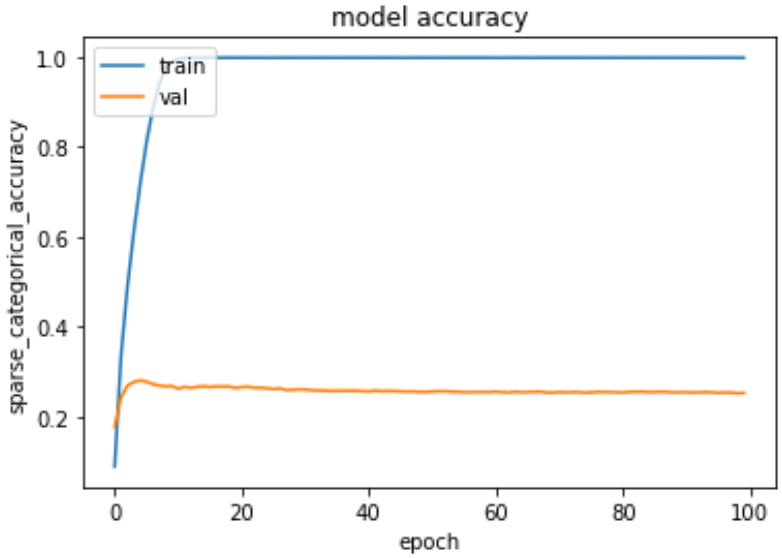


Figure 34: Model accuracy with 3000 feature size

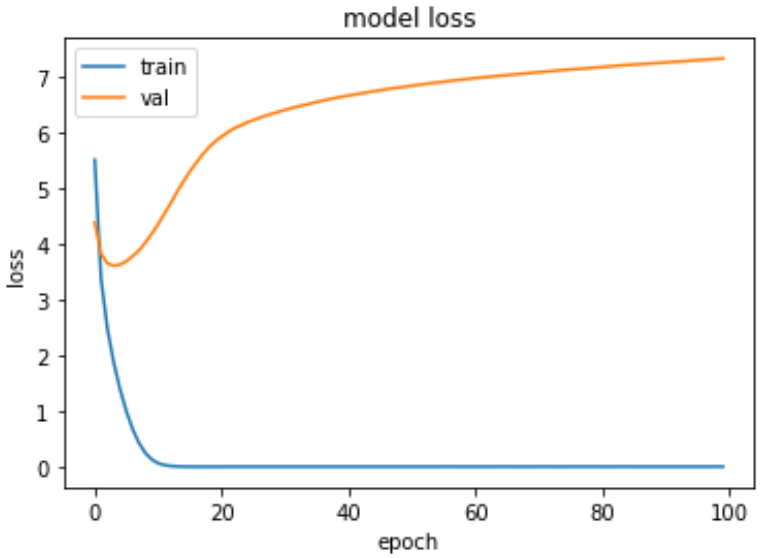


Figure 35: Model loss with 3000 feature size

When the feature selection is applied by selecting the best 3000 features for the feature sizes, a minimal increase in the top5-accuracy value is observed. However, when we examine the model loss chart, we observe that there is still an overfitting problem.

When the feature size is reduced to *5000 features* with the PCA method, the results are as follows:

Top-5 Accuracy = %42,4

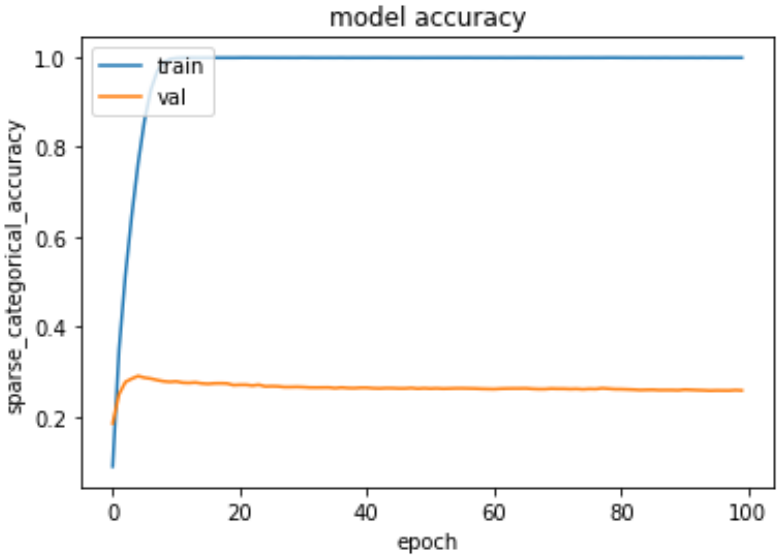


Figure 36: Model accuracy with 5000 feature size

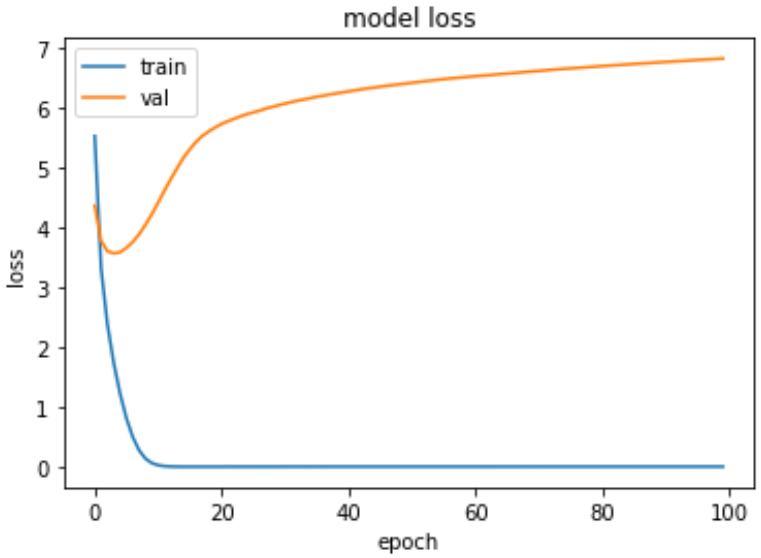


Figure 37: Model loss with 5000 feature size

It is observed that the top5-accuracy value is slightly lower than the previous experiment when the feature selection is applied by selecting the best 3000 features for the feature sizes. However, no change is observed if the graphs are examined compared to the previous experiment.

When the feature size is reduced to *8000 features* with the PCA method, the results are as follows:

Top-5 Accuracy = %41,0

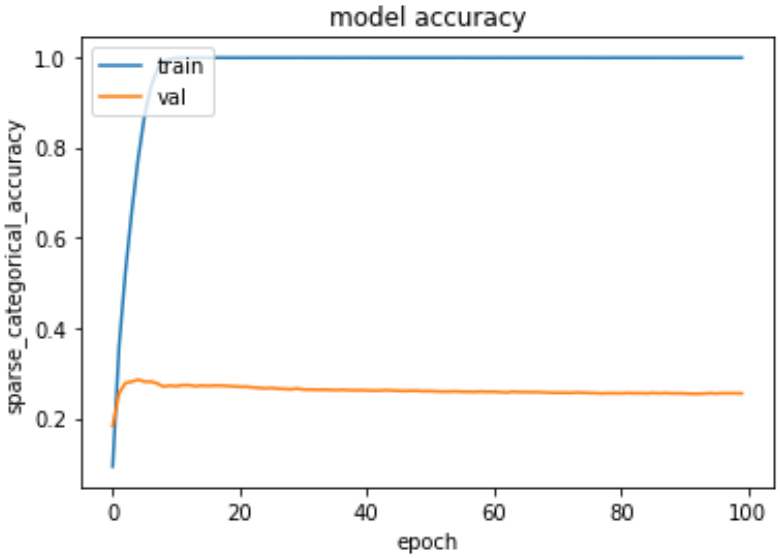


Figure 38: Model accuracy with 8000 feature size

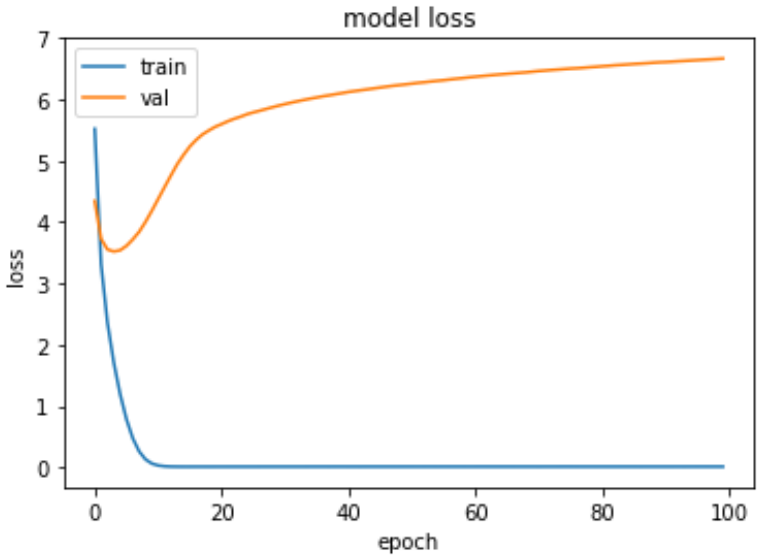


Figure 39: Model loss with 8000 feature size

Suppose we ignore the experiment with a total feature size of 9216. In that case, it is observed that the Top-5 accuracy value decreases slightly in the cases between feature selection 1000 and 9216, and the overfitting status does not change.

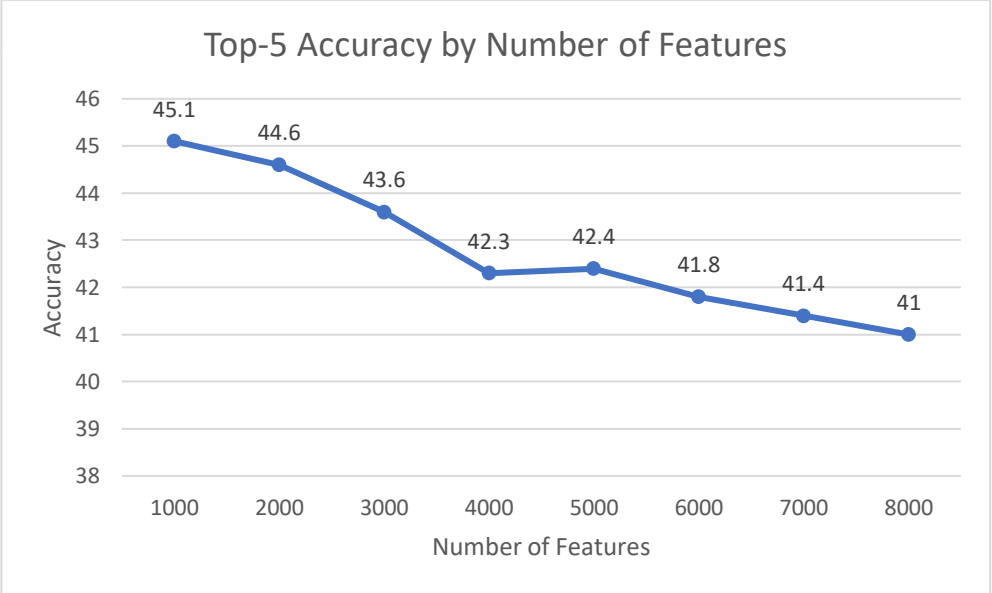


Figure 40: Top-5 Accuracy by Number of Features (PCA)

5.2 RESULTS OF ACTIVATIONS WITH SELECT K BEST FUNCTION

In this part, the results of the experiments with the features obtained using the Select K Best method are compared. The most significant difference between the

Select K Best method and the PCA method in these experiments is that while the values of the activations change in the PCA method, the activation values do not change in the Select K Best method. The new values formed after the PCA method are no longer quantized. Thanks to the Select K Best method, quantized activations retain their values. This allows it to be used in the upcoming Hybrid AI experiment.

When the feature size is reduced to *1000 features* with the Select K Best method, the results are as follows:

Top-5 Accuracy = %36,7

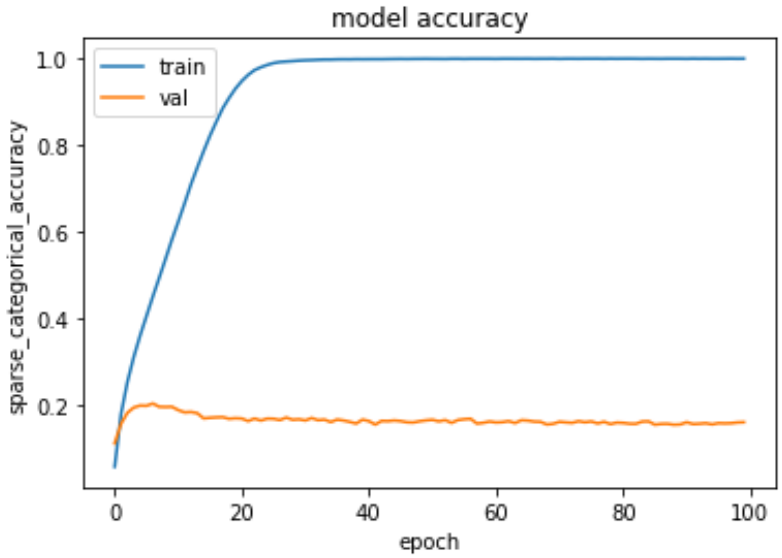


Figure 41: Model accuracy with 1000 feature size

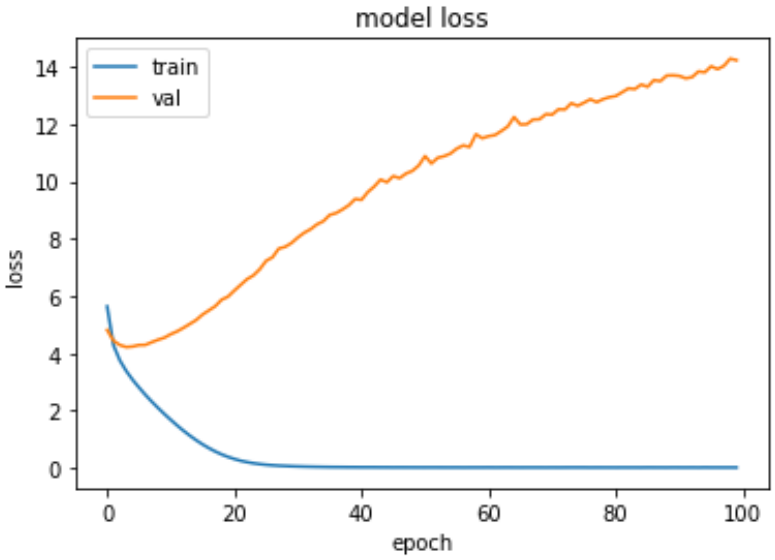


Figure 42: Model loss with 1000 feature size

The top 5 accuracy metrics are slightly less accurate than 1000 activation experiments obtained with PCA. This is because of the way the PCA and SelectKBest methods work. We can still observe the problem experienced due to the low data in these graphs. These issues are discussed in more detail in the Discussion section.

Top-5 Accuracy = %44,6

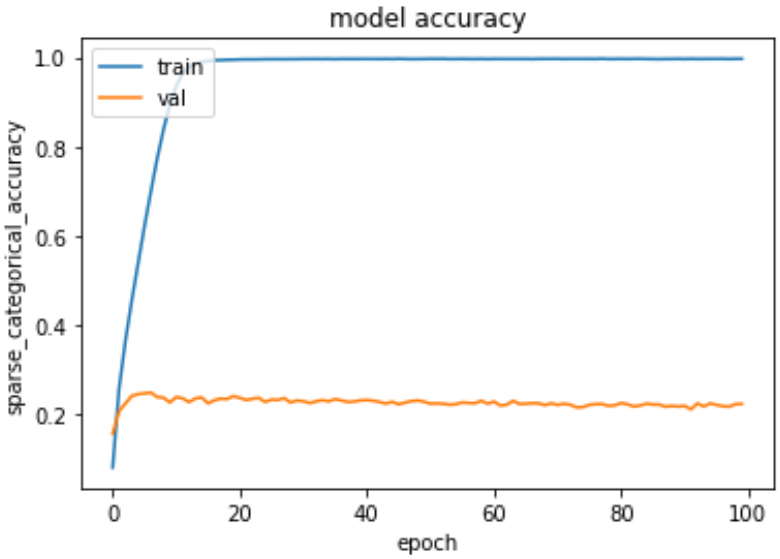


Figure 43: Model accuracy with 3000 feature size

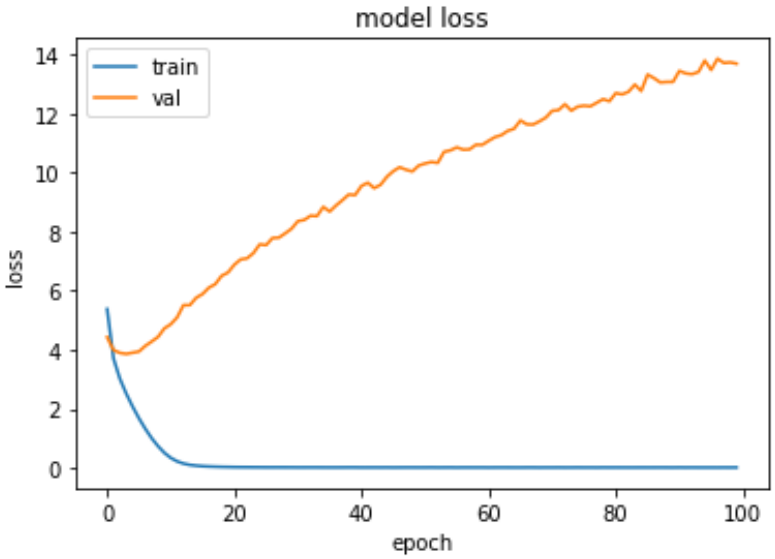


Figure 44: Model loss with 3000 feature size

When the Select K Best method is applied by selecting the best 3000 features for the feature sizes, a noticeable increase in the top5-accuracy is observed.

Top-5 Accuracy = %46,5

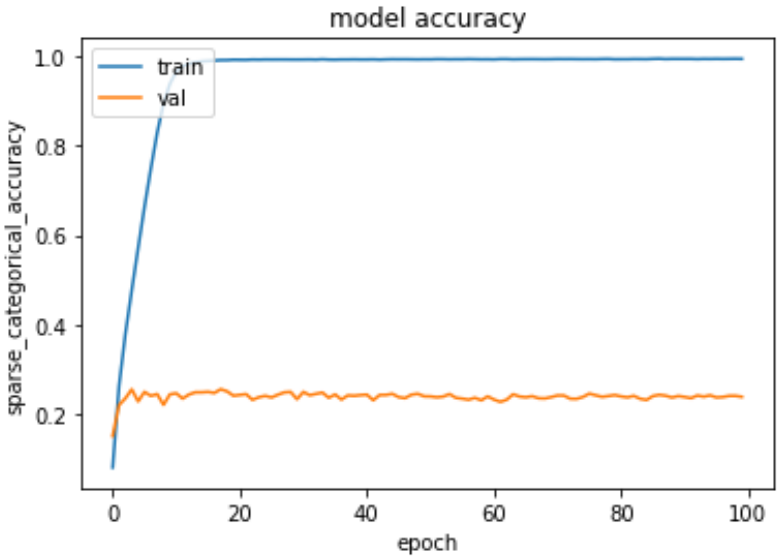


Figure 45: Model accuracy with 5000 feature size

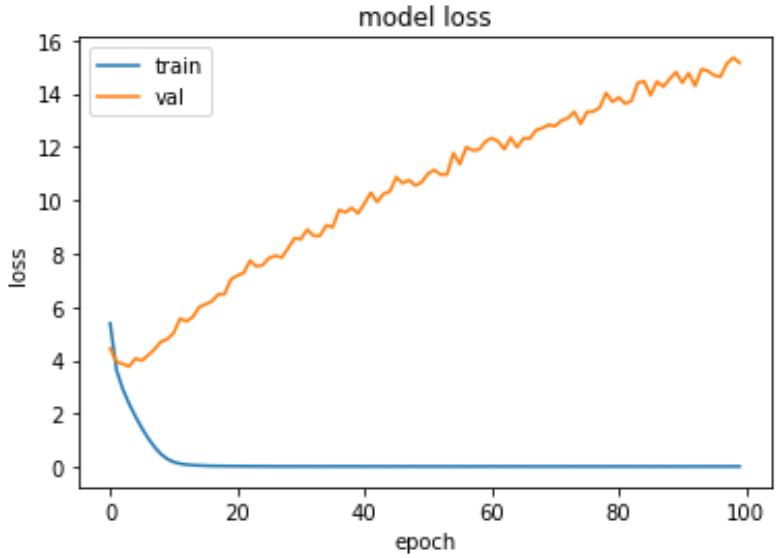


Figure 46: Model loss with 5000 feature size

It can be observed that the top 5 accuracy value, which increased up to around 6000 features, decreased after these values.

Top-5 Accuracy = %42,2

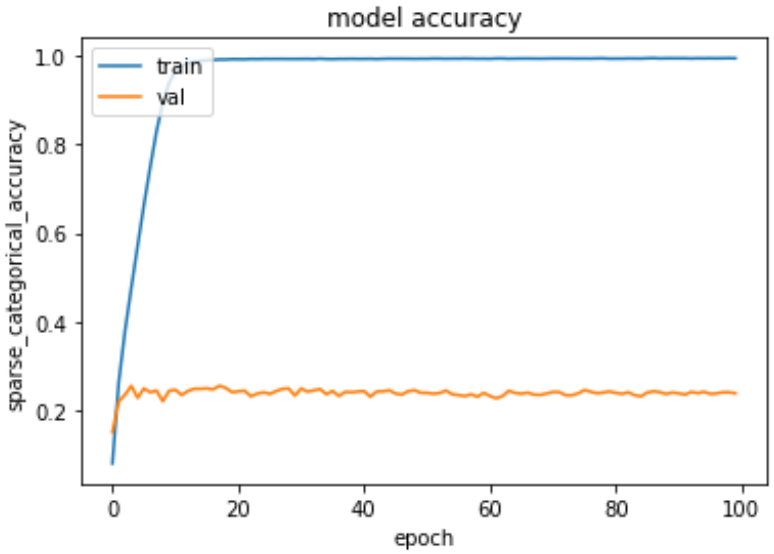


Figure 47: Model accuracy with 8000 feature size

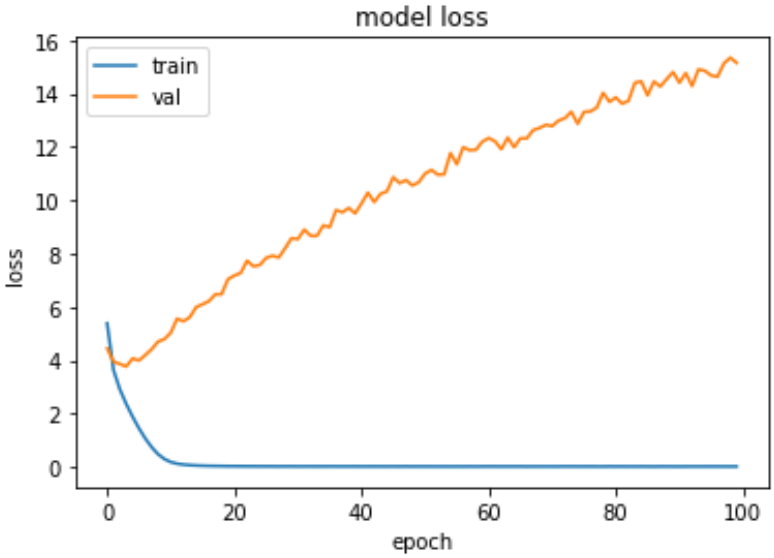


Figure 48: Model loss with 8000 feature size

With this Select K Best method experiment, in cases between 1000 and 9216 feature selection, it is seen that the Top-5 accuracy value increases until it reaches a certain number of features, then decreases, and the overfitting condition does not change.

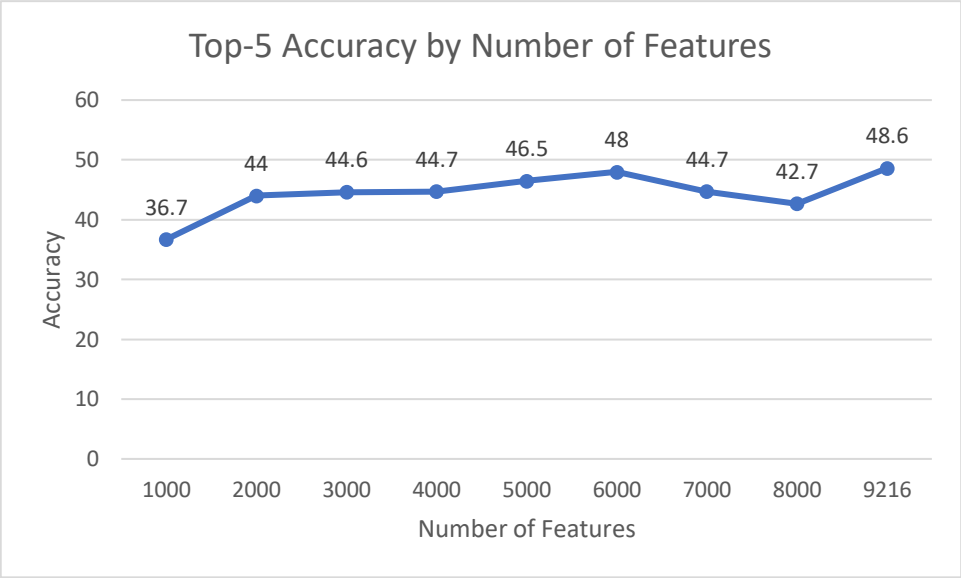


Figure 49: Top-5 Accuracy by Number of Features (SelectKBest)

5.3 RESULTS OF ACTIVATIONS WITH HYBRID AI

The RPNI algorithm is applied instead of the previously used FC layers in this part of the experiment.

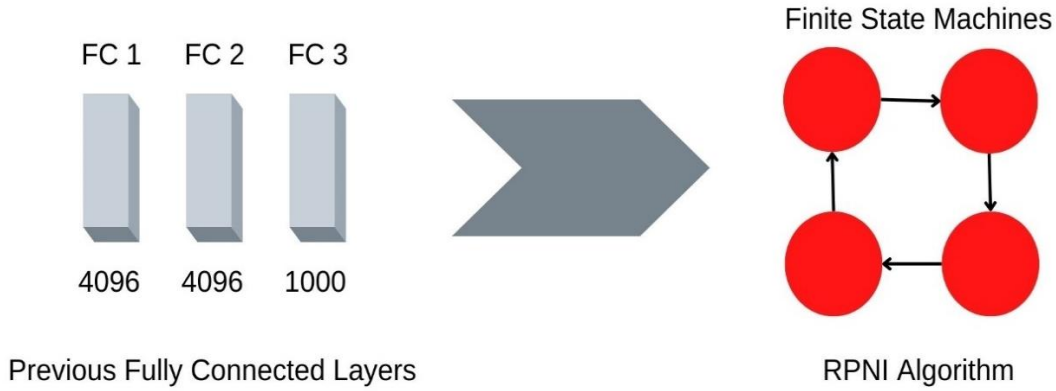


Figure 50: Previous model's FC layers and RPNI algorithm

Unlike previous experiments, instead of top5 accuracy, the one versus all classification required by this algorithm is applied. RPNI and Fully Connected layer outputs are compared using inputs with 1000 activation lengths. There are 3 FC layers used in this experiment. The first two are 1024, and the last is 6. The last layer is 6 in size because 6 different classes are used in total. Activations of 1300 images in total are obtained via the embedded device without being included in the decision

mechanism at the end of the model. While 650 of 1300 images belong to class 954 in the Imagenet dataset, the remaining 650 are images obtained in equal amounts from 5 other different classes.

The results of the first FC experiment with 1000 activation lengths are as follows.

	Precision	Recall	F1-Score	Support
715	0.85	0.88	0.86	130
717	0.85	0.88	0.87	130
719	0.72	0.75	0.73	130
723	0.82	0.67	0.74	130
726	0.84	0.69	0.76	130
954	0.89	0.94	0.91	650
Accuracy			0.86	1300
Macro Average	0.83	0.80	0.81	1300
Weighted Average	0.85	0.86	0.85	1300

Figure 51: 1000 Activation Length One vs. All Classification Report

The results of the RPNI algorithm experiment with the same activation lengths are as follows. The results of the RPNI algorithm in the tables below are calculated for class 954, and this class constitutes 650 of 1300 images in total.

ACTUAL			
Positive	*Negative*		
374	*356*	*Positive*	PREDICTION
276	*294*	*Negative*	

Figure 52: 1000 Activation Length RPNI Report

Recall	Precision	F1-Score
0.58	0.51	0.60

Figure 53: Recall, Precision, and F1-Score of RPNI Experiment

5.4 EVALUATION OF THE RESULTS

There are some requirements for the results obtained to be acceptable accuracy. A neural network model trained with correct hyperparameters with quantized layers can achieve these values. The top5 accuracy of the pre-trained network which has 100.000 samples from Imagenet, which consists of quantized layers, was calculated as 81.4%. We test if we could achieve the same or similar results by modifying some layers of this model, applying feature reducing methods, or even using a different algorithm, which is not very common.

Considering the Top5 Accuracy values of the experiment using the reduced activations obtained by the PCA method and using the Select K Best function, it is seen that the common problem is not enough input. The solution to this is, of course, to provide enough input, but because the size of the embedded device used and the number of activations it can store, more inputs could not be used. We think that this is the reason why the accuracy values of the experiments we used PCA and Select K Best were around 40-45%. Although overfitting is observed from the graphs of the experiments using PCA and Select K Best, a slight difference can be observed when

looking at the accuracy values. We assume that this is because PCA changes all these values while the Select K best function performs feature dimension reduction without changing the original activation values.

In Table 5.1, the network outputs consisting of custom FC layers for one vs. all classification are observed. When the RPNI algorithm is applied, the results obtained for class 954 are as in Table 5.2 and Table 5.3. While the one vs. all classification accuracy applied with FC layers can reach up to 91%, the RPNI algorithm is observed at a value of 60%. According to RPNI results it is observed that the results are usable but not enough. We think that this is because these values are limited due to the insufficient number of inputs. In addition, Fully Connected layers are more successful in resolving complex activations than the RPNI algorithm and it is possible to achieve much better results with less complex activations. To work with much larger inputs, it is necessary to have much stronger processing power. In addition, this RPNI algorithm, written in Python, must be completely changed and written in C language in order to work with longer activation sizes. The reason for this is that there is a huge difference between recursive depth in Python and recursive depth in C language, and Python supports this process for a maximum length of 1000. This subject is also included in the future works section.

CHAPTER VI: CONCLUSIONS AND FUTURE DIRECTIONS

In this thesis, 3 different methods are used to make the decision-making mechanisms of CNNs more efficient. Some challenges are encountered when performing these experiments. Regardless of the experiments, choosing a suitable CNN model at the beginning and then retraining this model with the appropriate hyperparameters and dataset has the potential to be a challenge.

First, the decision-making mechanism of the pre-trained neural network is tested as a quantized fully connected structure. Thus, we aimed to obtain an accuracy score on quantized layers, which are classically used in embedded deep learning applications. The application made while obtaining the Accuracy score is a classification process using the ImageNet data set to obtain a top 5 accuracy score.

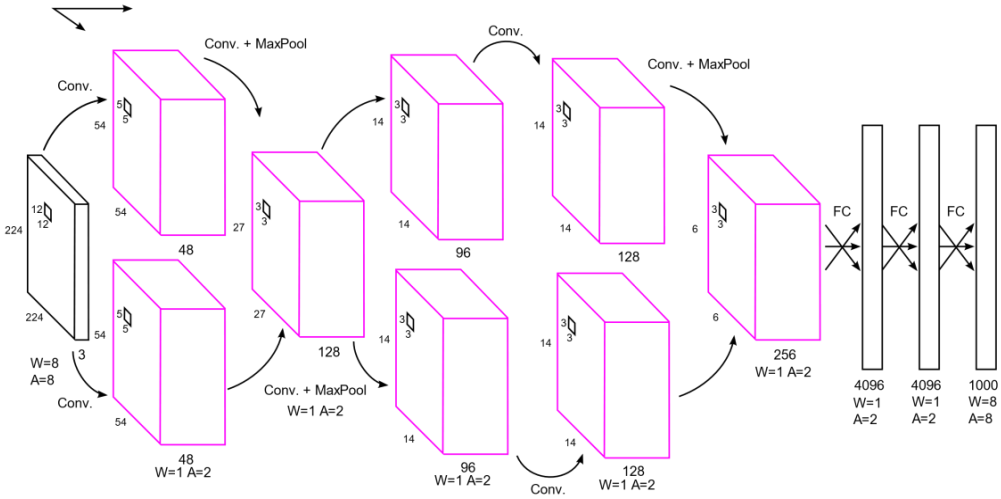


Figure 54: Pre-trained neural network with quantized layers

All activations of the pre-trained neural network (DoReFa-Net) are obtained using an embedded device (PYNQ-Z2). These obtained activations are reduced in dimension with the help of PCA and Select K Best functions and are prepared for use in experiments. Afterwards, we aimed to retrain this CNN model with modified FC layers using these activations. The application made while obtaining the accuracy score is a classification process using the ImageNet data set to obtain a top 5 accuracy score.

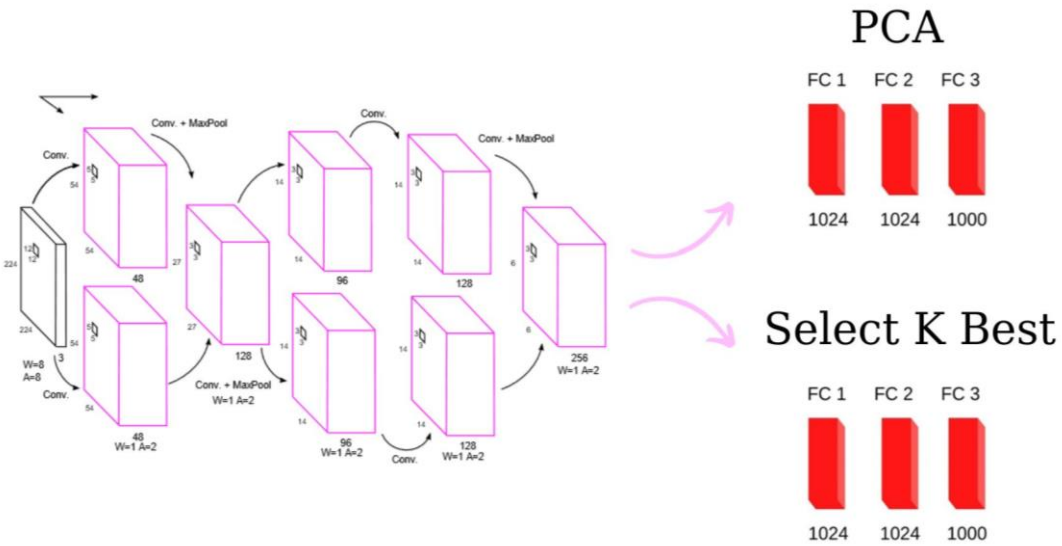


Figure 55: Pre-trained neural network with modified FC layers

Finally, we aimed to observe the accuracy score by trying the RPNI algorithm, which was created with the logic of the Finite state machine structure, as a decision-

making mechanism. Unlike previous experiments, the accuracy score to be obtained for this algorithm, which decides using a state structure logic, is different. Because the "true state" obtained in the algorithm used can reach a single answer. Therefore, instead of the top 5 accuracy, one vs. all classification is applied for a total of 6 classes, 1 of which is the correct class, obtained from the Imagenet dataset. In order to compare the results obtained, we modify the Fully connected layers and retrain the model for one vs. all classification. The sizes of custom FC layers applied here are 1000, 1000, and 6, respectively. The activation length in this experiment is 1000, and they are obtained from the original network (DoReFa-Net) used in the thesis using an embedded device (Pynq-Z2).

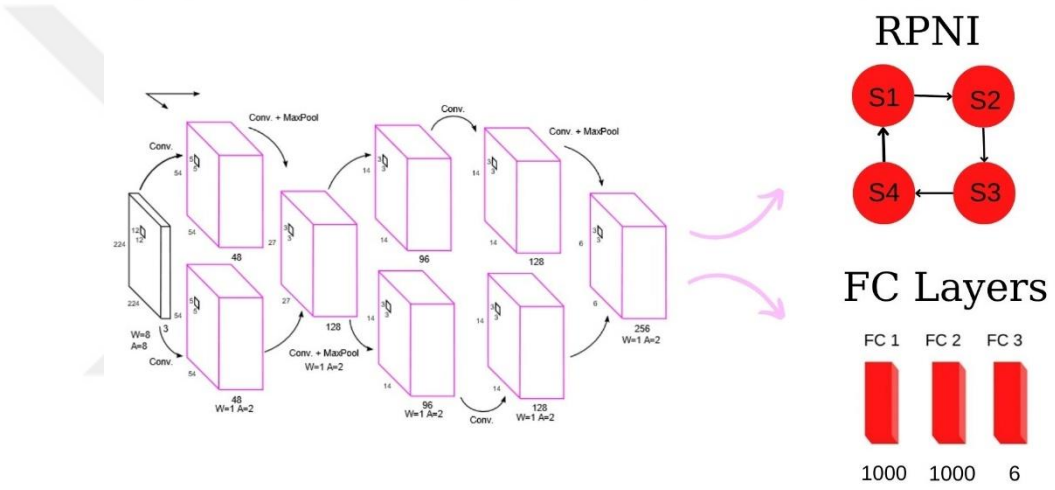


Figure 56: Pre-trained neural network with modified FC layers and RPNI

For future works, the first experiment can be tried on a different and larger embedded device with much more input numbers, thus improving the results. In the RPNI experiment, the same algorithm can be written in C code, thus avoiding the recursive depth problem. As a result, the results can be observed using longer activations. At the same time, more inputs for the RPNI experiment can be obtained and tested on a machine with more processing power. The RPNI algorithm to be used as a decision-making mechanism can be tested with much less complex activations. Also, experiments can be observed by increasing the number of inputs used. Since the RPNI algorithm is written in Python, we were able to perform RPNI experiments using a maximum activation length of 1000. This is because the Python language does not support large recursive depths. As future work, we want to rewrite this algorithm in C

language, which has a much larger recursive depth, and try it with different activation lengths. The fact that the RPNI algorithm works with the logic of the FSM structure overlaps with the fact that FPGAs also can be coded with the logic of FSM. When tested with experiments with less complexity, it can be a faster while used in embedded deep learning applications. When we can provide all these possibilities mentioned in the future works section, we aim to compare the results not only on accuracy scores, but also on the speeds on FPGA board for real life problems.

.

REFERENCES

[1] Patel, S., & Patel, A. (2020). Object detection with Convolutional Neural Networks. Machine Learning for Predictive Analysis, 529–539. https://doi.org/10.1007/978-981-15-7106-0_52

[2] Kasemi, Roni & Karahoda, Bertan. (2020). Implementation of Artificial Neural Network in Embedded Systems. 10.33107/ubt-ic.2020.429.

[3] Ji, Z., Ovsiannikov, I., Wang, Y., Shi, L., & Zhang, Q. (2015). Reducing weight precision of convolutional neural networks towards large-scale on-chip image recognition. SPIE Proceedings. https://doi.org/10.1117/12.2176598

[4] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115–133. https://doi.org/10.1007/bf02478259

[5] El Khiyari, H., & Wechsler, H. (2016). Face recognition across time lapse using Convolutional Neural Networks. Journal of Information Security, 07(03), 141–151. https://doi.org/10.4236/jis.2016.73010

[6] Zhang, Wei (1988). "Shift-invariant pattern recognition neural network and its optical architecture". Proceedings of Annual Conference of the Japan Society of Applied Physics.

[7] Hussain, A., Rosebrock, A., Jaffar, Hilman, Kabare, M., Gay, A., Wang, X., Alejandro, Agarwal, P., Klein, K., Walid, Manuel, Khan, E., Martin, Jakub, Ibrahim, & Sucanthudu. (2021, April 17). Keras conv2d and convolutional layers. PyImageSearch. Retrieved June 12, 2022, from https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[8] Stanford Lecture CS231n. [Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Lecture Notes to CS231n: Convolutional Neural Networks for Visual Recognition, 2015. [online]. https://cs231n.github.io. Date of Access: 20.06.2022

[9] Baheti, P. (n.d.). Activation functions in neural networks [12 types & use cases]. V7. Retrieved July 6, 2022, from https://www.v7labs.com/blog/neural-networks-activation-functions

[10] Wood, T. (2019, May 17). Softmax function. DeepAI. Retrieved July 6, 2022, from https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[11] Moons, B., Bankman, D., & Verhelst, M. (2018). Embedded Deep Neural Networks. Embedded Deep Learning. https://doi.org/10.1007/978-3-319-99223-5_1

[12] Lu, J., & Zhou, H. (2021). Implementation of artificial intelligence algorithm in embedded system. Journal of Physics: Conference Series, 1757(1), 012015. https://doi.org/10.1088/1742-6596/1757/1/012015

[13] Ananthaswamy, A. (n.d.). Ai's next big leap. Knowable Magazine | Annual Reviews. Retrieved July 6, 2022, from https://knowablemagazine.org/article/technology/2020/what-is-neurosymbolic-ai

[14] Karayiannis, N. B., & Venetsanopoulos, A. N. (1993). Efficient learning algorithms for Neural Networks (ELEANNE). IEEE Transactions on Systems, Man, and Cybernetics, 23(5), 1372–1383. https://doi.org/10.1109/21.260668

[15] Stecklina, M. (2018, February 21). Step by step to a quantized network. Medium. Retrieved July 6, 2022, from https://medium.com/@marianne.stecklina/step-by-step-to-a-quantized-network-5d7da6c52af1

[16] Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-net: ImageNet classification using binary convolutional Neural Networks. Computer Vision – ECCV 2016, 525–542. https://doi.org/10.1007/978-3-319-46493-0_32

[17] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. CoRR, abs/1511.00363, 2015. 10, 13

[18] Wang, J. (2021). Lightweight and real-time object detection model on EDGE devices with model quantization. Journal of Physics: Conference Series, 1748(3), 032055. https://doi.org/10.1088/1742-6596/1748/3/032055

[19] Novac, P.-E., Boukli Hacene, G., Pegatoquet, A., Miramond, B., & Gripon, V. (2021). Quantization and deployment of deep neural networks on microcontrollers. Sensors, 21(9), 2984. https://doi.org/10.3390/s21092984

[20] Jaderberg, M., Vedaldi, A., & Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. Proceedings of the British Machine Vision Conference 2014. https://doi.org/10.5244/c.28.88

[21] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for the efficient neural network. In Advances in Neural Information Processing Systems, pages 1135–1143, 2015. 1, 2, 3

[22] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In Advances, In 9 Neural Information Processing Systems, pages 2074–2082, 2016. 1, 2, 3

[23] Ma, X., Lin, S., Ye, S., He, Z., Zhang, L., Yuan, G., Tan, S. H., Li, Z., Fan, D., Qian, X., Lin, X., Ma, K., & Wang, Y. (2022). Non-structured DNN weight pruning—is it beneficial in any platform? IEEE Transactions on Neural Networks and Learning Systems, 33(9), 4930–4944. https://doi.org/10.1109/tnnls.2021.3063265

[24] Gavrilov, A. D., Jordache, A., Vasdani, M., & Deng, J. (2018). Preventing model overfitting and underfitting in Convolutional Neural Networks. International Journal of Software Science and Computational Intelligence, 10(4), 19–28. https://doi.org/10.4018/ijssci.2018100102

[25] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for Deep Learning. Journal of Big Data, 6(1). https://doi.org/10.1186/s40537-019-0197-0

[26] Survey on methodologies and techniques involved in feature selection. (2016). International Journal of Science and Research (IJSR), 5(2), 948–952. https://doi.org/10.21275/v5i2.nov161320

[27] Kumar, A. (2021, August 8). Machine Learning - Feature Selection vs feature extraction. Data Analytics. Retrieved July 6, 2022, from https://vitalflux.com/machine-learning-feature-selection-feature-extraction/

[28] Brownlee, J. (2019, August 6). How to configure the learning rate when training deep learning neural networks. Machine Learning Mastery. Retrieved July 6, 2022, from https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[29] Ved, A. (2016, October 5). How to improve performance of Neural Networks. d4datascience.com. Retrieved July 6, 2022, from https://d4datascience.com/2016/09/29/fbf/

[30] Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., Liew, Y. T., Srivatsan, K., Moss, D., Subhaschandra, S., & Boudoukh, G. (2017). Can fpgas beat gpus in accelerating next-generation Deep Neural Networks? Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. https://doi.org/10.1145/3020078.3021740

[31] Asano, Shuichi & Maruyama, Tsutomu & Yamaguchi, Yoshiki. (2009). Performance comparison of FPGA, GPU and CPU in image processing. FPL 09: 19th International Conference on Field Programmable Logic and Applications. 126-131. 10.1109/FPL.2009.5272532.

[32] FPGA vs. GPU vs. CPU – hardware options for AI applications. Avnet. (n.d.). Retrieved July 6, 2022, from https://www.avnet.com/wps/portal/silica/resources/article/fpga-vs-gpu-vs-cpu-hardware-options-for-ai-applications/

[33] Springer, T., Eiroa-Lledo, E., Stevens, E., & Linstead, E. (2021). On-device deep learning inference for system-on-chip (SOC) architectures. Electronics, 10(6), 689. https://doi.org/10.3390/electronics10060689

[34] Python productivity for Zynq. PYNQ. (n.d.). Retrieved July 6, 2022, from http://www.pynq.io/board.html

[35] Blott, M., Preußer, T. B., Fraser, N. J., Gambardella, G., O'brien, K., Umuroglu, Y., Leeser, M., & Vissers, K. (2018). FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems, 11(3), 1–23. https://doi.org/10.1145/3242897

[36] Julian Faraone, Giulio Gambardella, David Boland, Nicholas J. Fraser, Michaela Blott, and Philip H.W. Leong. 2018. Hardware-Optimized Pruning Methods for Efficient Low Precision Deep Neural Networks on FPGAs. In Under Review.

[37] Xilinx. (n.d.). Xilinx/QNN-MO-PYNQ. GitHub. Retrieved July 6, 2022, from https://github.com/Xilinx/QNN-MO-PYNQ

[38] Nurvitadhi, E., Sheffield, D., Jaewoong Sim, Mishra, A., Venkatesh, G., & Marr, D. (2016). Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. 2016 International Conference on Field-Programmable Technology (FPT). https://doi.org/10.1109/fpt.2016.7929192

[39] Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., & Zou, Y. (2016). DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. ArXiv, abs/1606.06160.

[40] Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., & Xin, J. (2019, May 2). Understanding straight-through estimator in training activation quantized neural nets. arXiv.org. Retrieved July 6, 2022, from https://arxiv.org/abs/1903.05662v2

[41] Papers with code - imagenet benchmark (image classification). The latest in Machine Learning. (n.d.). Retrieved July 6, 2022, from https://paperswithcode.com/sota/image-classification-on-imagenet

[42] Mishra, S., Sarkar, U., Taraphder, S., Datta, S., Swain, D., Saikhom, R., Panda, S., & Laishram, M. (2017). Principal component analysis. International Journal of Livestock Research, 1. https://doi.org/10.5455/ijlr.20170415115235

[43] Nair, R., & Bhagat, A. (2019). Feature Selection Method To Improve The Accuracy of Classification Algorithm. https://www.ijitee.org/wp-content/uploads/papers/v8i6/F3421048619.pdf

[44] Touger, E. (2022, June 29). What Is an FPGA and Why Is It a Big Deal? . Prowess Corp. Retrieved July 6, 2022, from https://www.prowesscorp.com/what-is-fpga/

[45] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., Vanderplas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. ArXiv, abs/1309.0238.

[46] Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., & Vissers, K. (2016, December 1). Finn: A framework for fast, scalable Binarized neural network inference. arXiv.org. Retrieved July 6, 2022, from https://arxiv.org/abs/1612.07119v1

[47] Courbariaux, M., Bengio, Y., & David, J.-P. (2016, April 18). BinaryConnect: Training deep neural networks with binary weights during propagations. arXiv.org. Retrieved July 6, 2022, from https://arxiv.org/abs/1511.00363

[48] H. Nakahara, H. Yonekawa, and S. Sato. An object detector based on multiscale sliding window search using a fully pipelined binarized CNN on an FPGA. In 2017 International Conference on Field Programmable Technology (ICFPT), pages 168–175, Dec 2017. 13

[49] Corchado Rodríguez, Juan & Aiken, J.. (2002). Hybrid artificial intelligence methods in oceanographic forecast models. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 32. 307 - 313. 10.1109/TSMCC.2002.806072.

[50] Conte, F., D'Antoni, F., Natrella, G., & Merone, M. (2022). A new hybrid AI optimal management method for Renewable Energy Communities. Energy and AI, 10, 100197. https://doi.org/10.1016/j.egyai.2022.100197

[51] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 16, 2015.