# FPGA implementation of an sEMG classifier

Master's Thesis in Biomedical Engineering

William Marnfeldt
2022


LUND
UNIVERSITY

Faculty of Engineering LTH Department of Biomedical Engineering

Main Supervisor: Christian Antfolk
Second Supervisor: Alexander Olsson

# Abstract

This master's thesis discusses the implementation of a convolutional neural network on a Field Programmable Gate Array (FPGA). It deals with implementation be describing a tool chain, starting with the designing of a model in Keras, transforming the model to Hardware Descriptive Language, and finally implementing it on an FPGA. Performance on three different scales of the same model topology are compared, in the following terms: accuracy, timing and power consumption. Findings show that timing is within acceptable ranges, with limitations lying in model capacity, and power consumption. Furthermore, the specific model is compared with a similar topology. Finally, suggestions for future attempts are proposed, suggesting new layer types.



**Keywords:** electromyography, convolutional neural networks, classifier, field programmable gate array, prosthetic hand

# Can artificial neural networks be deployed on FPGAs for prosthesis control? - Popular Science Summary

**With FPGAs, custom hardware can be designed for artificial neural networks. This may lead to performance improvements for artificial neural networks, thereby allowing for more advanced prosthesis control.**

For an amputee, a comfortable and easily controllable robotic prosthesis may significantly their improve life quality. One of the proposed inputs for controlling robotic prostheses is by surface electromyography. In other words, the prosthesis can be controlled by measuring the electrical potential in muscles from the surface of the skin. A neural network can then be trained to infer movement patterns from these signals. An issue with neural networks, is their large complexity, making them difficult to adopt to on smaller microcontrollers, typically desired on prostheses that run on battery. This puts a limit on the scale and speed of the neural networks that can be deployed. A good prosthesis controller should not only correctly determine the intended movement, but also do it within a short time frame. For this reason, the alternative approach of using FPGAs was attempted. An FPGA is in essence re-configurable hardware, where instead of programming software onto hardware, the hardware itself is programmed. This allows task specific hardware to be built which can do many calculations at once. In short, the main advantage would be allowing for multiple calculations to be performed in parallel as opposed to in a sequence, thus gaining a speedup. A workflow was proposed and followed in which a Convolutional Neural Network was constructed and trained on a computer. It was then, through several steps, translated into a corresponding hardware design, and deployed to an FPGA. The resulting implementation was fast enough to allow for multiple inferences within the desired timing constraint. This, in turn, meant that majority voting could be used to improve results. There still remains issues, notably in accuracy and a power consumption. Overall, this project successfully demonstrated the possibility of using FPGAs as controllers. In the future, this may open up for possibilities of using more dedicated hardware for neural networks, with FPGAs acting as design prototypes.

## Acknowledgements

I would like to thank my supervisors, Christian Antfolk and Alexander Olsson for offering guidance, feedback and plenty of patience throughout this project. I am also thankful to my friends and family for supporting me and encouraging me.

# Contents

# Acronyms

**ANN** Artificial Neural Network.

**ASIC** Application Specific Integrated Circuit.

**BRAM** Block Random Access Memory.

**CNN** Convolutional Neural Network.

**CPU** Central Processing Unit.

**DSP** Digital Signal Processing.

**EMG** Electromyography.

**FIFO** First In, First Out.

**FPGA** Field Programmable Gate Array.

**HDL** Hardware Descriptive Language.

**HLS** High Level Synthesis.

**RTL** Register Transfer Level.

**sEMG** Surface Electromyography.

# 1 Introduction

The loss of a limb is a devastating event to an individual. Although not the most common type of amputation, the consequences of a transradial amputation include functional and vocational limitations. This involve the complication of everyday actions, such as eating, washing or dressing [1]. Previously trivial tasks, may become very demanding. Therefore, one of the main goals of any prosthetic should be to restore functionality to its user. A common option for amputees are electronic prostheses, which may offer better control than the body powered alternative [2].

## 1.1 EMG-controlled prostheses

One approach to controlling robotic prostheses is through the use of surface electromyography (sEMG). This provides the benefit of being non-invasive, requiring no surgery. By measuring electrical signals in muscles on the surface of the skin, a prosthetic limb can be made controllable, allowing amputees to regain some functions of their lost limbs. Using electromyography to control prosthetic limbs is a technique that was pioneered in the late 1940's. Since then, the field has seen improvements: both in terms of prosthesis, being lighter and more dexterous, with a higher degree of freedom, but also in improving how they are controlled. [3].

One of the challenges of controlling the prosthesis is reading and classifying the sEMG signals, i.e. parsing the incoming signals and translating them to an appropriate output. Various approaches to controlling the prosthesis have been proposed. They can be categorized as pattern recognition based or non-pattern recognition based. Pattern recognition based control schemes, offer larger degrees of freedom in control, i.e. the ability to perform more actions with the prosthesis. Typically, some sort of classifier is used, i.e. an algorithm to determine which set an observation is given to [4].

## 1.2 Classification on an FPGA

Classification can be computationally heavy, making it difficult to implement in real time applications. As such, traditional CPU-architecture, being very sequential in nature, struggles with latency issues. Even for pre-trained networks, running classification may be problematic due to timing constraints. Controller delays should optimally be around 125-150 ms and no larger than

300 ms [5]. Many of these calculations required can be run in parallel providing room for speedup [6]. A possible route of implementation is therefore to run the model on an FPGA. The advantage of an FPGA implementation is the possibility to increase the level of parallel processing, thus allowing for a higher throughput of information. Since multiple sEMG signals are being measured in parallel, a faster system is expected. In certain cases, there may also be improved power usage, as compared to CPUs [7].

# 2 Aim and Research question

The main goal of the thesis is to investigate the use of FPGAs for classification of surface elecromyographic signals for the purpose of controlling robotic prostheses. The purpose for doing this is to examine potential benefits and issues related to its use for future research. The intent is to answer the following question: Are FPGAs a viable option for the control of sEMG based robotic prostheses? Another topic to be examined is the workflow. Developing hardware descriptions for FPGAs is time consuming. Therefore finding and describing a workflow would be beneficial.

# 3 Basic Concepts

A proposed control system would consist of, input acquisition of sEMG signals, preprocessing, classification and finally a control scheme. Figure 3.1 provides a simple overview for the steps in such a system. The larger portion of the project ascertains to producing a workflow to enable faster design of classifiers. This demands a principle understanding of each stage. As such, the fundamentals of each step in the project will be discussed.

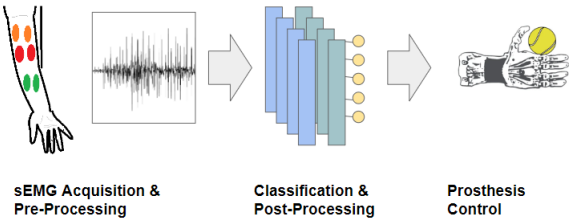


**Figure 3.1:** Overview of the main steps in a prosthesis control system [8].

## 3.1 Electromyography

Electromyography, or EMG is the measurement of electrical potential over skeletal muscles. It can be used to detect neural stimulation of a muscle, and as such, it provides a way to identify muscle activation. Typically, a distinction is made between intramuscular EMG and surface EMG (sEMG), which are invasive and non-invasive respectively. Invasive EMG involves using needle electrodes to measure the potential over the muscle of interest. The non-invasive nature of the surface EMG makes it an attractive alternative in prosthetic control as it only requires electrodes placed on the surface of the skin. Using sEMG is not without its downsides. A common issue is cross-talk, where energy from adjacent muscles groups is detected by a surface electrode. The recorded signal may therefore contain information from other muscle groups, making it difficult to isolate the muscle groups individually. Another issue arises from the amount of measurement points, limiting the amount of monitored muscle sites. Therefore, the measured signals will typically not represent all muscle activations [9]. As such, properly classifying movements

## 3.2 Data Preparation & Feature extraction

After acquiring a set of raw sEMG signals, they will need to be prepared. Typically, the main energy of the signal lies between 0-500 Hz, as illustrated in figure 3.2.

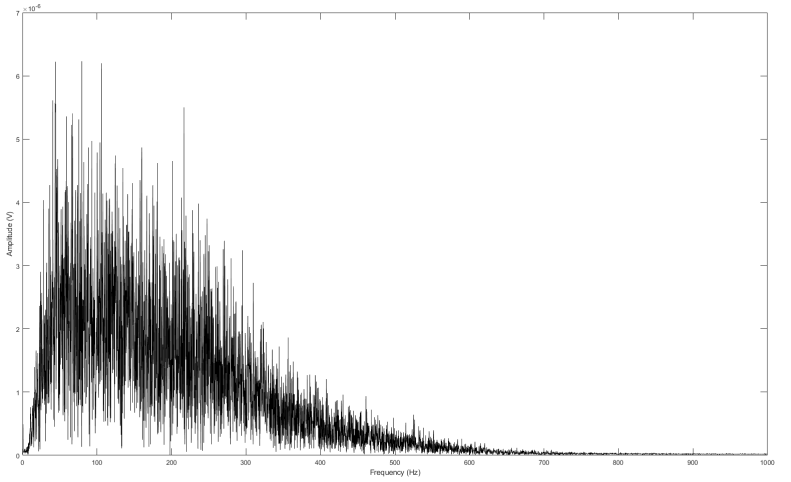


**Figure 3.2:** Spectrogram of the input signal. The vast majority of its content exists below 500 Hz.

As such, artifacts above this frequency are typically discarded [10]. Furthermore, due to power line interference, filtering out 50 Hz on systems is typically done [11]. Once noise artifacts have been removed, feature extraction is performed. Feature extraction refers to processing a signal in order to express a certain feature, such as an average amplitude, or a measurement of frequency. These features can then be used to determine the current class. For example, if all incoming signals are of a very low amplitude, it would be reasonable to assume that no movement was intended. Traditionally, features are hand crafted and are typically categorized as belonging to the time domain, the frequency domain, and spatial domain. In practice this means applying a known function to the input data, for example calculating the average amplitude over the sample window, or counting zero crossings in a window. Recently, there have been successful attempts at using convolutional neural networks to generate features instead of handcrafting them [12]. This has the benefit of reducing the time spent finding adequate features, but may also enable new features to be found.

## 3.3 Timing Constraints

Controller delays should optimally be around 125-150 ms and no larger than 300 ms. Anything larger than that will be degrade user experience [5]. As such, a primary timing constraint is given. Windowing, i.e. sampling the incoming the signals over a certain length of time, is typically used when classifying sEMG signals. The sampled windows are then processed. In order to increase the number of classifications performed, for example when using a majority voting scheme, the windows that are sampled can be overlapping. An example of this is given in figure 3.3.

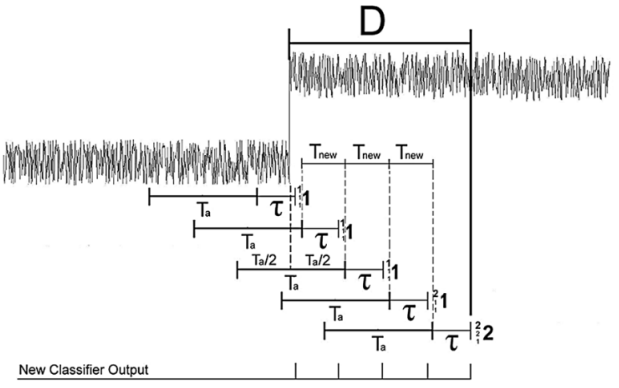


**Figure 3.3:** Majority voting with overlapping sampling [5].

In figure 3.3, $T_a$ refers to the window size, with $\tau$ being the classification time and $T_{new}$ the stride length. In regards to figure, the maximum delay $D_{max}$ can be described as [5]:

$$D_{max} = \frac{1}{2}T_a + (\frac{n+1}{2})T_{new} + \tau \tag{3.1}$$

Where *n* is the vote count, i.e.. the number of votes used.

## 3.4 Power Constraints

For any controller, low power consumption is desirable as the prosthesis would be battery powered. As such, lower power consumption allows for less battery weight or longer battery life. The batteries need to last for 12-16 hours before recharging [13]. The controller for the SmartHand transradial prosthesis is reported to have a power consumption of 1.44 W, although this is stated as being too large[14]. This gives a rough comparison point for a controllers power usage.

## 3.5 Classification

Once the incoming signals have been filtered, the next required step is to understand the user's intent from the signals. Simply put, how can the filtered signals be mapped to the intended motion? Classification refers to mapping a given combination of inputs to discrete classes, in this case, which movement is being performed. In the simplest case, classification involves only two classes, when more classes are available it is referred to as multi-class classification. Put in layman's terms, the classifier should be able to figure out what movement the sEMG data corresponds to. In order to determine the quality of the classifier some metrics will be introduced. Typical metrics of interest for a multi-class classifier are balanced accuracy, precision, recall, and f1-score. Before discussing these metrics, four terms commonly used when discussing binary classifier will be introduced. These are:

- True Positive - Classifier predicts positive, actual value is positive.
- False Positive - Classifier predicts positive, actual value is negative.
- True Negative - Classifier predicts negative, actual value is negative.
- False Negative - Classifier predicts negative, actual value is positive.

In a small confusion matrix the terms would be allocated as illustrated in figure 3.4.

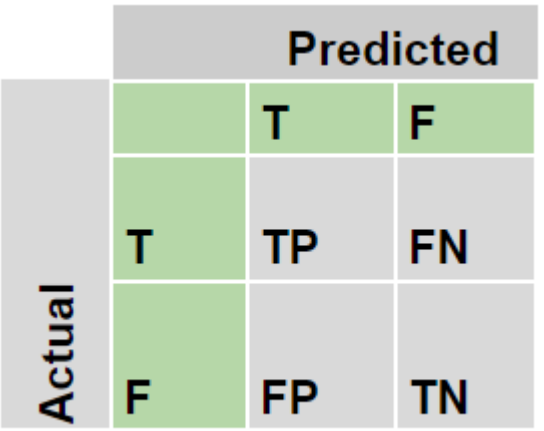


**Figure 3.4:** Confusion matrix for a binary classifier, each term allocated.

Although the terminology would seem to refer to the binary case, the concept can be expanded upon to work with multi class classifiers. This is done by iterating over the different classes and focusing on one at a time as the true class. As such, false positives exist on the same column as the class in question and false negatives on its row. As for true negatives, they fill the remaining spaces. This doesn't present an issue as they won't be used for any calculations. Figure 3.5 illustrates this with the second class being in focus.

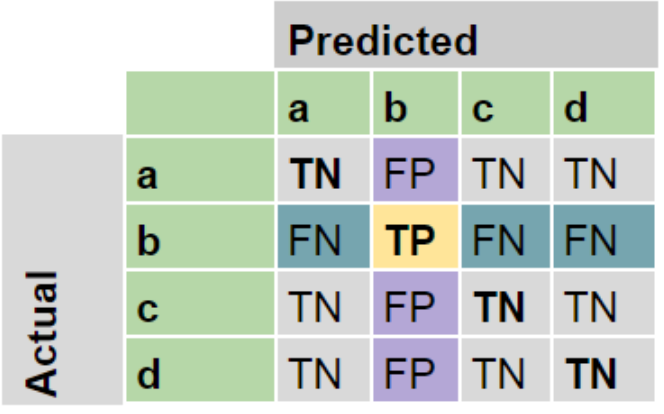


**Figure 3.5:** Confusion matrix for multiple classes with focus on class b.

For multi-class classification, regular accuracy, i.e. the total number of true positives for each category divided by the total classification attempts, may seem enough. However, if the data is unbalanced, one or more classes being over-represented may lead to a high number of correct classes for the over-represented class without the model even being able to classify the other categories. This justifies the need for balanced accuracy. By weighing each individual classes accuracy by its occurrence in the test data, no single category can outweigh the rest. Balanced accuracy can be expressed in the following way:

$$BalancedAccuracy = \frac{1}{N}\sum_{k=1}^{N}\frac{TP_k}{TP_k + FN_k} \tag{3.2}$$

Where N is the number of classes. This way, no single class is over-represented.

Precision aims to answer the question: Out of all the predictions made to a given category, how many were of that category? It can be expressed as the following for each element:

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{3.3}$$

Recall, in contrast, answers the similar but differing question: Out of all the occurrences of a specific class, how many were correctly classified?

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{3.4}$$

F1-score exists in two categories for multi class classification. For the sake of brevity, since only macro F1-score is used it will furthermore be referred to as F1-score. F1-score is the harmonic mean of the Precision and Recall of the classifier. It therefore gives a semblance of an average between the two scores. It can be expressed as the following:

$$F1 = 2 \cdot \frac{MAP \cdot MAR}{MAP^{-1} + MAR^{-1}} \tag{3.5}$$

In the equation, the terms MAP and MAV refer to the Macro Average Precision, and Macro Average Recall respectively. They are the average Precision and Recall of all classes [15]. Since the F1-score is a combination Precision and Recall, the metrics used in the project will be Balanced Accuracy and F1-score.

## 3.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a subset of machine learning based on emulating neurons. Although the concept is rather old, with early attempts at modeling a neural network dating back to the 1940's, the use of ANNs has become more frequent. Today, neural networks are used for a plethora of different applications, from pattern recognition of DNA to estimating the shelf life of food products [16].

Although artificial neural networks are commonly described as emulating the human brain, this description offers little help in understanding how they function. An ANN can be considered a composition of multivariate functions as illustrated in figure 3.6. The component functions are referred to as layers. Each layer contributes in complexity to the function and as such, the topology of the model determines the function [17].

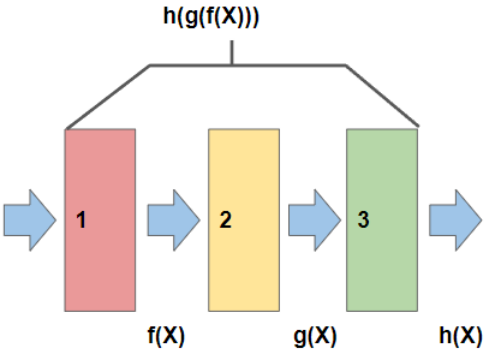


**Figure 3.6:** Each layer of the neural network acts as a component to the composite function.

The trainable weights behave as gradients and constants to each component function. The distance between the desired outcome and the given output is expressed as loss function. The distance between the two may be expressed in various ways, depending on the character of the problem. When the network is trained, the weights of each layer are adjusted so as to minimize the loss function. To put it simply, the ANN can be considered an adjustable function and the purpose of training is adjusting the functions weights so as the model behaves as well as possible [16].

Recently, the use of Convolutional Neural Networks (CNN:s) has gained popularity. One of the proposed advantages of using CNN:s is their ability

to extract features, thereby removing the need for handcrafted feature extractors [18]. What this means, is that raw or very lightly filtered sEMG data can be used as an input signal, thereby reducing the amount of pre-processing needed. Atzori et al. demonstrated such a network with a an average accuracy of 60.27 ±7.7% [19]. The following sections will provide a brief introduction to the layers used in the model.

### Dense layer

The dense layer is also commonly denoted as a fully connected layer. It functions by multiplying each incoming value with a trainable weight, summing up the weighted signals, and finally, applying a bias. The output for a dense node can be described by the following equation [20]:

$$y = X \cdot W^T + b \tag{3.6}$$

In equation 3.6 $y$, $X$, $W$ and $b$ represent the output, input, weights and bias respectively. $X$ and $W$ represent vectors, hence the capital notation. Figure 3.7 illustrates how a dense node performs it's operations.

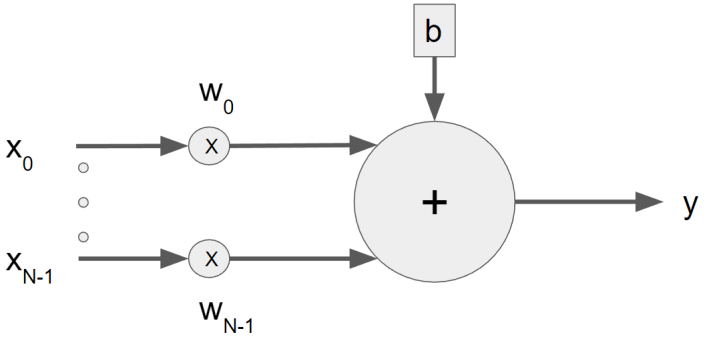


**Figure 3.7:** The dense node multiplies and accumulates every input signal with it's corresponding weight. A final bias is then applied.

### Convolutional Layer

The convolutional layer performs an n-dimensional convolution between the input tensor and an n dimensional weight tensor, the kernel. For each element in the input tensor, the kernel is centered atop the element, followed by the multiplication of each overlapping elements of the kernel and tensor. The products of each element pair are then summed together followed by the summation of an additional bias. This process is performed for every element in the vector. As such, it is computationally expensive to perform convolutions. In the case of the kernel and input tensor not overlapping completely, for example in the corners and edges of the input tensor, zero padding may be used. In effect this means that any range outside of the input tensor is considered a zero, as illustrated in figure 3.8. Alternatively wrap-around can be used, as with normal discrete convolutions. If this is not used, the output tensor would shrink in size. [20] .

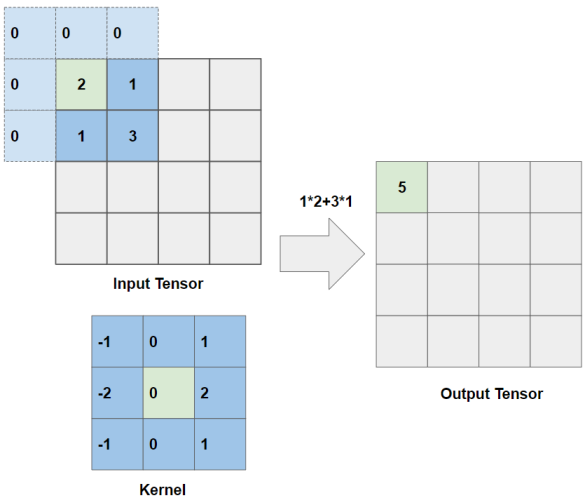


**Figure 3.8:** Example of zero padding being used to preserve the output tensor dimensions.

The resulting elements in the output tensor, in the case of a 2-dimensional operation can be expressed as follows:

$$y_{i,j} = \sum_{k=0}^{p}[\sum_{l=0}^{q}(x_{(i+k-\lceil\frac{p}{2}\rceil),(j+l-\lceil\frac{q}{2}\rceil)} \cdot w_{k,l})] + b$$

(3.7)

Where $x$, $w$ and $b$ are the input tensor, kernel and bias. Furthermore, $p$ and $q$ are the width and height of the kernel. It should be noted that the limits of summation In the case of multiples of kernels in a layer, the output will consist of several channels, each being a convolution between the input and one of the kernels. When the input tensor has multiple channels, a convolution is performed for every channel with the same kernel. The resulting outputs are then summed together, forming a single output tensor. As such, the number of channels of an output tensor is determined by the number of kernels that a convolutional layer has regardless of the number of input channels. It is possible to keep the channels separated with separable convolutional layers [21].

**ReLU**

Activation functions are functions placed on the output of a node meant to represent the firing of a neuron. One of their main purposes is providing non-linearity to each layer, thus allowing for better approximation of general functions. A Rectifying Linear Unit (ReLU) is a type of activation function that works like a rectifier. The output $f(x)$ is equal to the input $x$ as long as the input is larger than a predetermined threshold $z$, sometimes referred to as a zero point. This threshold is typically set to zero. The ReLU can be expressed as [22]:

$$f(x) = \begin{cases} x \mid x > z \\ 0 \mid x \leq z \end{cases} \tag{3.8}$$

**Softmax**

The softmax layer is another activation function that is typically used in the final layer of a neural network, as it outputs the normalized probability distribution of each output class. It can be expressed as [20]:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{3.9}$$

Where $f(x_i)$ is the probability of class $i$. As seen in equation 3.9, the softmax layer takes into account all of the signals of the previous layer.

### Pooling

A pooling layer is a downsampling layer. It downsamples by selecting a subset of input elements from a vector or matrix (depending on dimensionality) and outputs a single value corresponding with the specific pooling type. The two most common pooling types are average pooling, where the average value becomes the output and max pooling, where the largest value becomes the output [20].

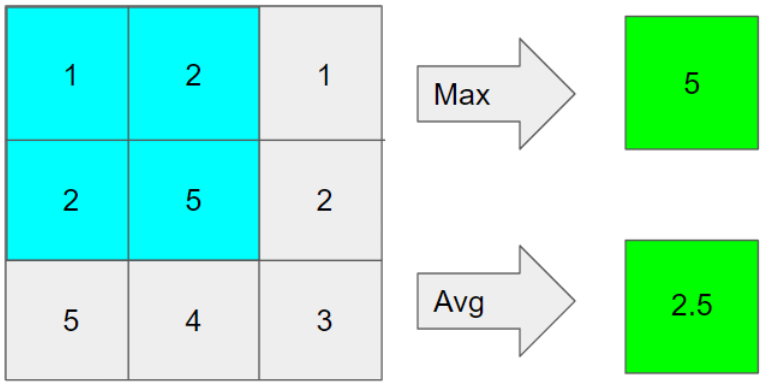


**Figure 3.9:** The values in the blue boxes are downsampled to one value

**Batch Normalization**

Batch normalization is a layer that normalizes the input, the resulting output being scaled and shifted so as to have a mean close to 0 and a standard deviation close to 1. It has been shown to allow the use of higher learning rates, as well as reduce the sensitivity to parameter initialization [23]. The equation used by Keras is the following [24]:

$$f(x) = \gamma * \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} + \beta \tag{3.10}$$

Here, $\beta$ and $\gamma$ are trainable parameters. Furthermore, $\epsilon$ corresponds to a small pre-defined constant. It should be noted that the mean and variance are only calculated during training and fixed during inference. As such, the expression simplified during inference, resulting in a much simpler affine function [25]:

$$f(x) = kx + m \tag{3.11}$$

where k can be denoted as:

$$k = \frac{\gamma}{\sqrt{\sigma_x^2 + \epsilon}} \tag{3.12}$$

and m as:

$$m = \beta - \frac{\mu_x}{\sqrt{\sigma_x^2 + \epsilon}} \tag{3.13}$$

This scaling becomes useful for data types with limited ranges, as it helps bound the ranges.

## 3.7 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA), is an integrated circuit consisting of non-specific configurable blocks called logic blocks or logic units (LU), along with memory and DSP-accelerators. In contrast to a microcontroller, it is not programmed in a traditional sense as there is no CPU on which it executes instructions. Instead, it emulates hardware on a register transfer level (RTL), which is described using Hardware Descriptive Languages (HDL). In summary, instead of programming a series of sequential instructions to perform a task, hardware that would perform those instructions is described and implemented. The primary motivation for implementing neural networks on FPGAs is parallelism.

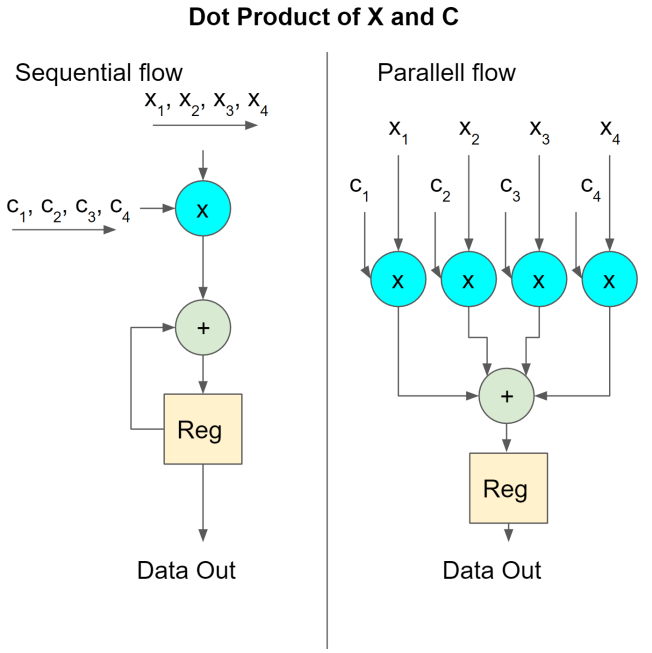


**Figure 3.10:** The sequential architecture requires less resources but performs slower.

Consider the following example, the dot product of vectors X and C, as illustrated in figure 3.10. In the example to the left, one element from each vector is multiplied and subsequently accumulated. Because the register, which behaves like a small memory, requires a positive flank from the clock to update it's value, a single multiplication and accumulation requires one clock cycle. In the example to the left, the operation is unrolled so that the multiplications

are performed simultaneously, thereafter summed together and finally saved. In the first example, four clock cycles would be required to perform the operation, whereas in the second example, all summations could be performed in one clock cycle. Thus, at the cost of more hardware usage, the operation could be sped up. Although the first example is not an equivalent to the architecture of a processor, a processor executes operations in a sequential matter. For the purpose of later results the term reuse factor is introduced as a measurement of how unrolled a process is. Reuse factor describes how much the same components are reused in regards to a given task. In regards to the previous example, the same multiplier and adder were used to perform 4 operations. As such it's reuse factor would be 4. In essence this term describes how many times a multiplier is used to perform operations. As with the example, low reuse factors therefore require more hardware resources but take less time.

Parallelizable architecture is not exclusive to FPGAs, Application Specific Integrated Circuits (ASICs) can also have multiple data-paths, and are typically faster and less power consuming. Kuon and Rose [26] measured a roughly 9 times larger power consumption on an FPGA than on an ASIC performing the same task. The difference is however that an FPGA is reusable and doesn't require extensive development time. As such, it allows for easy prototyping while still retaining many of the advantages of an Application Specific Integrated Circuit.

### Components

Naturally, there is a limit to what can be implemented on an FPGA. Only sufficiently small designs can be fitted on to an FPGA. Additionally, different FPGAs have different amounts of onboard building blocks. The three main building blocks of concern are Logic Slices, digital signal processors (DSP:s), and block RAM (BRAM).

The basic building block is the Logic Slice, also known as Logic Blocks or Configurable Logic Blocks. They contain a given number of flip flops and look up tables, depending on which device. They function by replacing conventional logic with look up tables, where the result of a combination of inputs are pre-stored. As such, instead of using logic gates, memory is used to represent the given output. The look up tables can in other words be considered truth tables. This is also what makes them re-configurable: by simply rewriting the truth table, the function they represent is changed. Figure 3.11 exemplifies a representation of an AND-gate. As such, they can

be made to emulate combinational logic. The flip flops further the concept by allowing the implementation of clocked logic.

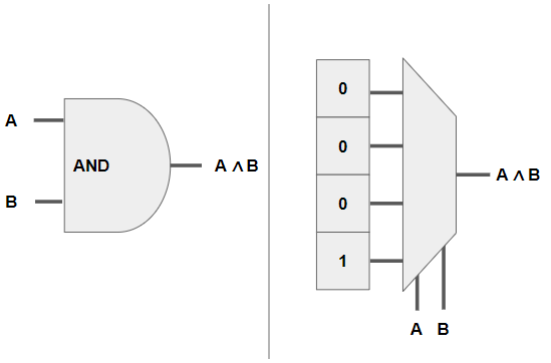


**Figure 3.11:** Logic gate equivalents can be designed using look up tables. The look up table in the figure is simplified for purposes of demonstration.

The DSP-slices can be considered accelerators, dedicated to performing multiplication and accumulation operations of binary numbers. Multiplication of binary numbers is performed as a series of additions and bit-shifts. For a wider bit-width, more area on the chip is required. For long numbers, this becomes very hardware demanding, as large numbers of adders need to be cascaded. As such, maintaining a high utilization of each DSP-slice is desirable, not only performance wise, but also in terms resource usage as it reduces the number of Logic Units needed to perform the same operation.

The final important building block is the BRAM or Block Random Access Memory. These are onboard blocks of RAM memory. In a similar fashion to the DSP-slices, these blocks exist to reduce resource usage of Logic Units, which could in fact also be used for memory, as they contain flip-flops. However, in many cases when larger memory storage is needed, such as in a buffer, BRAM is advantageous [27].

Table 3.1 presents the resources of the two devices used in the project. BRAM is represented in 36Kb blocks .

**Table 3.1:** Summary of resources available on the used devices. Each Logic Slice contains 4 LUTs and 8 Flip Flops. [28][29]

| **Board** | **Nexys Video** | **Zybo Z7** |
|---|---|---|
| FPGA | XC7A200T-1SBG484C | XC7Z020-1CLG400C |
| Logic Slices | 33650 | 13300 |
| LUTs | 134600 | 53200 |
| Flip Flops | 269200 | 106400 |
| DSP-Slices | 740 | 220 |
| BRAM | 365 (13Mb) | 140 (4.9Mb) |

Although the Zybo Z7 consists of fewer resources, it has an integrated ARM A9 processor, thus removing the need for a soft microprocessor core and certain peripheral drivers.

## 3.8 Numerical Representations & Quantization

A commonly used way to reduce hardware size and consumption, as well as increase speed, is to replacing floating point representations with fixed point representations [30]. Although the stated advantages, the use of floating point representations is still very common. In order to understand why one would be preferred over the other, an examination is required. In the following section a brief look at how values are represented will be given, as well as a comparison between a fixed point fractional representation and a floating point representation. There exists a multitude of different ways to represent values. Regardless of representation scheme, the various combinations are limited by the number of bits used to represent unique values. It can be described as follows:

$$n_{combinations} = 2^N \tag{3.14}$$

Where N is equal to the number of bits used for the representation. The first representation to be discussed is the fixed point fractional representation. It is an expansion of a binary representation where fractions are included by using bits to represent negative powers of two as opposed to just positive powers. It can be summarized by the following expression, in this case with an unsigned notation:

$$x_{N-1} * 2^{(N-1-D)} + ... + x_1 * 2^{(1-D)} + x_0 * 2^{(-D)} \tag{3.15}$$

In the expression, N denotes the number of total bits, and D denotes the number of fractional bits used. A large number of fractional bits implies a higher resolution, but subsequently demands a sacrifice in range. Additionally, if negative numbers are to be represented, the most significant bit can be altered to represent its negative complement. Since the largest bit is negative, the interval is thus shifted from $[0, 2^N - 1]$ to $[-(2^{N-1}), 2^{N-1} - 1]$. This negative notation is commonly referred to as two's complement.

The second type of representation is the floating point representation. It functions similarly to a scientific notation in that it consists of a significand and exponent. The representation is however in base 2, and it contains a negative offset to the exponent to accommodate for small numbers. In addition, negative numbers are represented by a sign bit. It can be expressed as the following:

$$(-1)^s * (1 + m) * 2^{n-Offset} \tag{3.16}$$

Where $s$ is the sign bit, $m$ the significant and $n$ the exponent. What is important to note is that the representable numbers are not evenly spaced. For

large numbers, the spacing is larger than for small numbers, as the exponent sets the ”step size”. For larger exponents, every increment of the significand results in a larger step [31]. This means that the standard allows for high precision for small numbers while still retaining a large range. This forms the basis for the IEEE standard for floating point arithmetic or IEEE 754 [32]. Figure 3.12 illustrates bit allocation for a 32 bit floating point number.

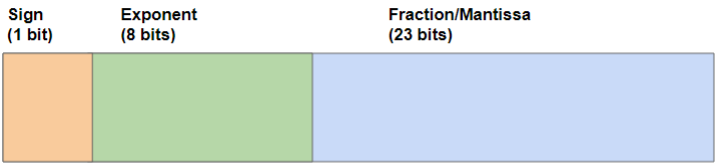


**Figure 3.12:** Bit allocations for a IEEE 754 floating point value [32].

When comparing the two presented types of representations, the first apparent difference is numerical spacing. The fixed point uses fixed spacing, whereas the floating point representation uses variable spacing. The second difference between the representations, although less obvious, is how operations are performed on the numbers. For a fixed point representation, the majority of operations can be performed on integer based hardware, whereas floating point operations typically requires specialized floating point hardware [31].

### Quantization & Pruning

When implementing classifiers on FPGA:s, a large bit-size may be very taxing in terms of resources [33], 32-bit numerical representations require more resources than 16- or 8-bit numerical representations. Furthermore, floating point mathematics are more resource intensive, requiring up to 5 times as many DSP-slices as the equivalent fixed point operation [31]. In order to reduce hardware usage, a preferred architecture uses as few bits as possible to represent values as well as a fixed point representation. This comes with a trade-off, however. For every bit removed, the number of possible numerical representations is halved. A fixed point representation requires careful handling of range to avoid overflows or saturation errors.

The tools commonly used to develop and train ANN:s commonly use 32-

bit floating point values. As such, quantization is necessary. Quantization aims to map a given set of numbers to a smaller subset. This is done by approximating each value to a similar existing value. It is typically performed by rounding. However, when quantizing a model, classification accuracy may be reduced due to the rounding errors introduced. This drop in accuracy can be reduced by performing Quantization Aware Training (QAT), which takes into account a desired quantization of parameters when initially training the neural network. The model size can be reduced additionally by pruning. This works by identifying weights with values close to zero, and subsequently setting them to zero. This can be done during training, allowing the model to adjust to the reduction in weights [34].

# 4 Methodology

The following sections will describe the various tools used in the workflow throughout the process. As one of the goals of the project is identifying a workflow that would allow low development times, an automated workflow is advantageous as compared to handwriting neural networks in Hardware Descriptive Language. The ideal workflow would allow a high level model of a neural network to be immediately deployed onto an FPGA. As such, automating the design flow is the preferred method. The following sections aims to describe each tool used, as well as to explain their purpose in the workflow.

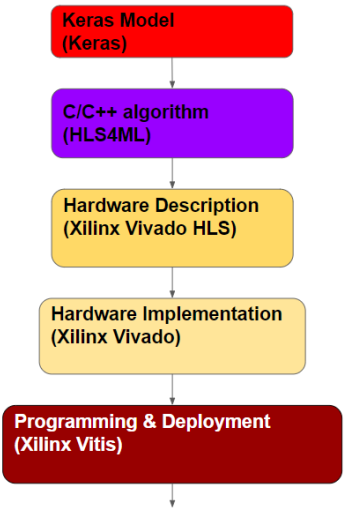


**Figure 4.1:** Summary of the proposed workflow, starting with a Keras model and ending with a deployable bitstream.

Figure 4.1 summarizes the tools used throughout the project. It should be noted that the workflow was iterative, and many iterations were pushed through the workflow. As will be discussed, not all layers could be implemented using the tool set, something that was discovered during the course of the project.

## 4.1 Model Design and Training

### Dataset and Pre-processing

The NinaPro DB2 [8] database was selected as the data set for the model. This was in large due to it being sampled at a sufficiently high frequency, at 2 kHz, as well as being one of the more commonly used data sets. Out of the 49 classes available in the data set, the 23 grasp movements from exercise C, as illustrated in figure 4.2, along with the class for resting were selected. For each movement class selected, repetitions 1, 3, 4 and 6 were used for training. The remaining repetitions, 2 and 5 were used for testing. This was done to ensure that the full movement sequences would be unchanged. Due to the same rationale, the data remained unshuffled.

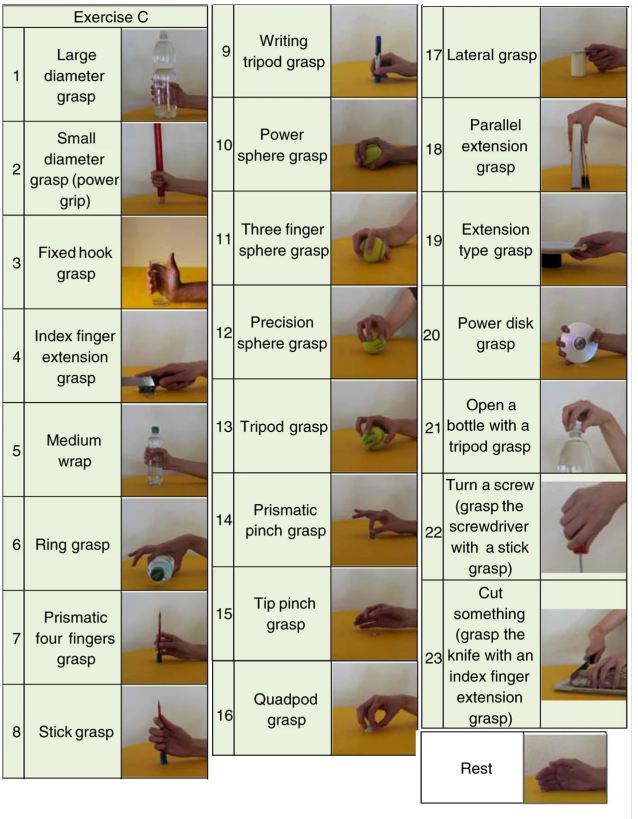


**Figure 4.2:** The 23 functional grasps from exercise C of NinaPro DB2 [8].

During training sessions different ways of processing the data were experi-

mented with. Originally the data was filtered, rectified and normalized but the filtering and rectification were ultimately dropped in favor of normalizing the data and downsampling it to 1 kHz. As such, the input data was very close to being ”raw”. The data was divided into windows of 128 samples, with a 32 sample stride length. In the second iteration of it’s implementation, the stride length was reduced to 4 samples. As such, every window represents 128 ms, with each stride being a step forward in of 32 ms or 4 ms respectively. The input ultimately consisted of a 2-dimensional matrix, with each column being a time-series of one of the input channels as illustrated in figure 4.3. The figure does not represent the final dimensions.

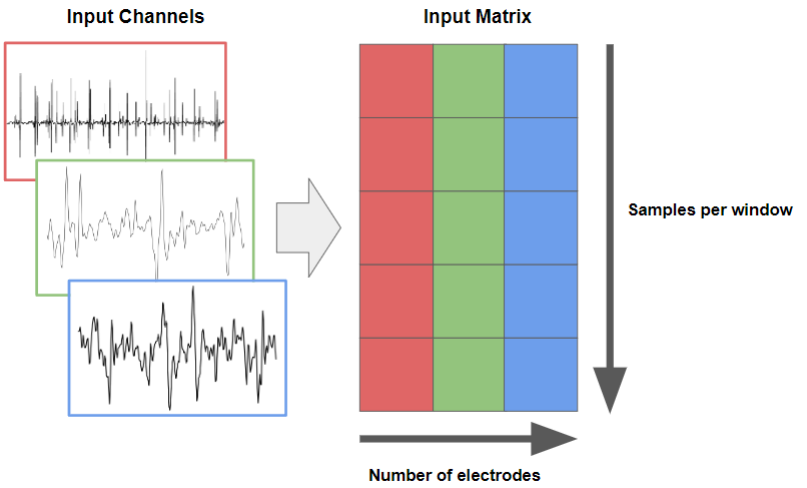


**Figure 4.3:** The time-series of each channel constructs the image. Each row represents a time series for a specific channel.

### Model

The model was initially designed in Keras. Keras is an API built on top of TensorFlow to simplify the implementation of artificial neural networks. Simply put, it is easy to use. It includes libraries of commonly used pre-implemented network layers. It was chosen due to its accessibility, with a large set of layers, as well as being compatible with tools used down streams in the workflow. It should be noted that an additional add-on tool QKeras was used in conjunction, to enable quantization aware training by providing drop in replacement layers. This meant that layers were typically denoted with the prefix Q if they used quantized weights. This notation will hereafter

be used in descriptions when referring to quantized layers.

The network designed was a 5-layer convolutional neural network, with similarities to the models used by Gijsberts et al. [35] and Pinzón-Arenas et al. [36]. Similar to the network used by Gijsberts et al., the network used raw inputs from which the convolutional layers extracted features. In order to reduce the total number of output weights required, a convolutional layer with a 1x1 kernel was used as the final convolutional layer. This effectively acted as a way to reduce the number of channels before flattening. As such it reduced the number of weights required by the dense layer. From the initial model design, the convolutional and dense layers were replaced with 16 bit quantized layers. Furthermore, the average pooling layers were replaced with max pooling layers. This was due to conversion issues further downstream in the workflow. The broad topology is illustrated in figure 4.4.

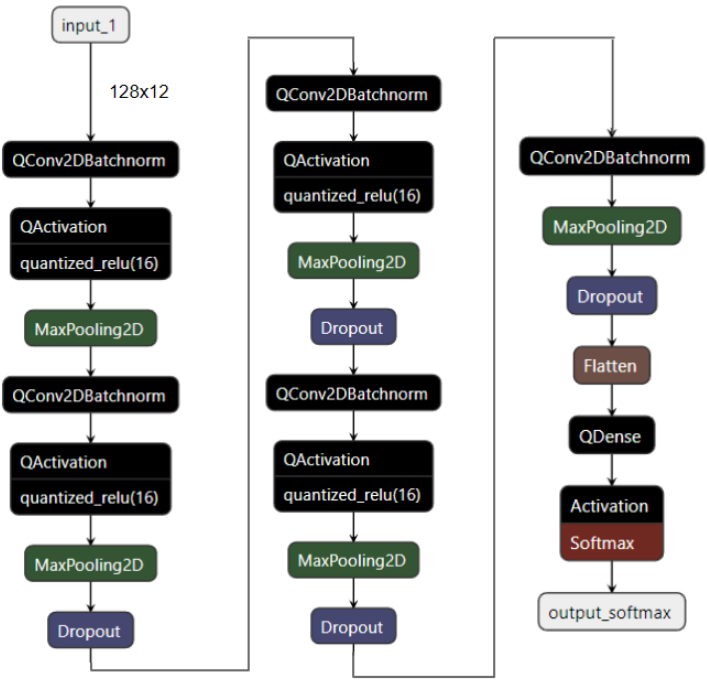


**Figure 4.4:** Overview of model architecture.

The model was tested with three scale configurations: small, medium and large. The difference was only in the size of its layers, meaning that larger models had more neurons in each layer. Table 4.1 presents a summary of the layer sizes for the three models.

**Table 4.1:** Table of layer sizes for each model.

| Layer | Kernel Size | Small | Medium | Large |
|---|---|---|---|---|
| QConvolutional | 5x5 | 8 | 10 | 16 |
| QConvolutional | 3x3 | 8 | 10 | 15 |
| QConvolutional | 3x3 | 16 | 20 | 32 |
| QConvolutional | 2x2 | 16 | 20 | 32 |
| QConvolutional | 1x1 | 8 | 8 | 8 |
| QDense | - | 24 | 24 | 24 |
| Total Parameters | - | 7,880 | 9,546 | 16,173 |

Training was performed over 50 epochs, ie. 50 passes over the entire dataset, with a batch size of 50 windows. In order to avoid over-fitting, the training was set to stop early if the loss didn't decrease after 10 epochs. Furthermore, the training rate was gradually halved after a five epoch loss plateau. After initial training, the model was additionally pruned and retrained an additional 10 epochs with the same training hyperparameters. The target pruning sparsity was set to 30%. For the purpose of producing a comparison to other works, the large model was trained on all 40 patients. The small and medium models, with the main purpose of providing comparisons in terms of scale, were trained on subject 1.

## 4.2 Model Deployment

Once the model was trained and pruned, it was exported to HLS4ML where it was converted into a C++ model. HLS4ML is a tool to enable deployment of ANN:s to FPGAs. It functions by converting the Keras models into a C++ equivalent model, which can then be used by the Vivado High Level Synthesis (HLS) workflow. In addition, the software also provides profiling tools, to analyze the weights, and outputs of each layer of the models. Using the tool, the data widths were profiled in order to avoid situations when the weights or results would be out of range, which would result in overflow issues. In order to ensure no accuracy loss at conversion, seeing as the model was already quantized, the Keras model was compared along side a C++ simulation of the hardware model. It was also at this stage that each layer's reuse factor was configured.

## 4.3 High Level Synthesis

When the converted and quantized model was verified, it was run through Vivado HLS to synthesize a hardware description. Xilinx Vivado HLS is a high level synthesis tool, which allows deploying C, C++, and System C algorithms on Xilinx FPGA:s without having to manually write RTL. It is commonly used to implement digital signal processing algorithms into hardware. Parameters for this stage were set through HLS4ML:s configuration files. This step became mostly automated with the exception of the models input and output. A wrapper file was written to reduce the number of inputs and outputs by implementing a streaming interface. For an input tensor with 12 channels and 128 time samples, a total of 1536 values are required. However, since convolutions function by having kernel traverse the tensor, not all inputs are required immediately [37], as illustrated in figure 4.5. Therefore, streaming is utilized and the values can be fed sequentially. The CNN model was exported as an IP-block.

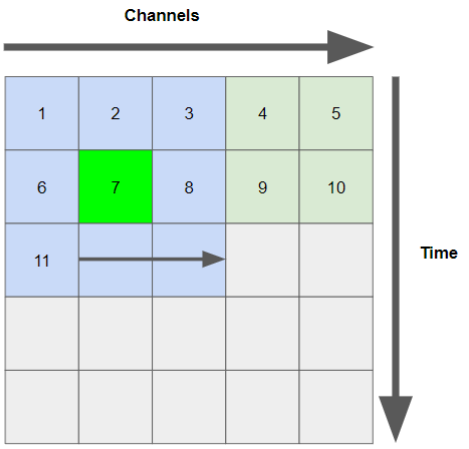


**Figure 4.5:** Input sequence for the streaming scheme. The convolutional layer doesn't require the full tensor to begin computing, only the elements covered by the kernel.

## 4.4 Hardware Design

Once the IP was generated for the design, it could be implemented into hardware. The resulting block required a surrounding infrastructure to manage dataflow and input/output. As such, a surrounding infrastructure was designed. For the larger Nexys Video, this involved implementing a MicroBlaze soft processor core and connecting the exported IP-block to it via a streaming interface. The MicroBlaze was connected to desired inputs and outputs. For testing purposes, this primarily meant a UART interface, as it could be connected to a computer via USB allowing for quick validation with scripts. The main reason why a processor core was included was to allow for easier integration of different accelerators and methods of input/output without needing to change the underlying architecture. This would have been required if a specialized overhead had been used. Hence the used architecture allows for interchangeability. Figure 4.6 illustrates the difference between using a custom overhead and the more generalized design which was implemented.

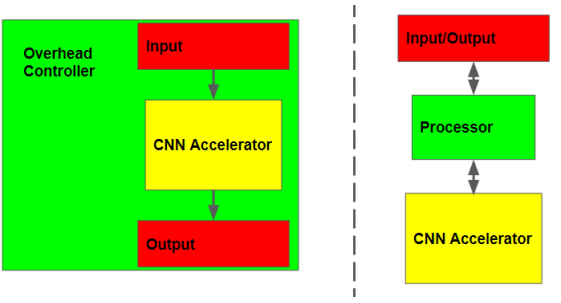


**Figure 4.6:** Using an interconnected processor grants larger interchangeability between the blocks.

For the Zybo Z7 the equivalent models were implemented, with the only major difference being an ARM core instead of the MicroBlaze, thus requiring less setup. Once the design was synthesized and implemented, its hardware specifications were exported to Xilinx Vitis for programming.

## 4.5 Design Programming

The imported device was programmed in Xilinx Vitis. Xilinx Vitis is a tool for programming processor architectures implemented on Xilinx FPGA:s. It is in essence an IDE with capabilities for generating bit-streams for a given

hardware architecture on a processor core. A simple program was designed: The device would receive the input data via UART from a computer. It would then package it and stream it to the designed accelerator. After fetching the results, these would then be transmitted back to the computer where they could be evaluated. A small segment of code was also written for temporal majority voting, but for testing purposes this was ultimately done offline. The majority voting scheme consisted of a FIFO-queue, where the most frequently occurring result was selected.

# 5 Results

Because of the broad scale of the project, the results describe both the model, but also it's hardware implementation. Due to the nature of the models being implemented on an FPGA, key aspects of its implementation will be presented along with the models actual performance.

## 5.1 Resource Usage and Reuse Factor

As can be seen in the following tables, resource usage differs not only on scale, but also on reuse factor. The result of having to vary the reuse factor in order to accommodate the differing resource availabilities of the two devices is a difference in resource usage. In order to achieve consistency between the models the default reuse factor used was 64 with the exception of the output layer, which had a reuse factor of 24. For the smaller Zybo Z7 the reuse factor used was 256, with the output layer maintaining a reuse factor of 24. It should be noted that the reuse factor was determined empirically.

**Table 5.1:** Resource Usage for the small sized model. The difference in resource usage between the devices can be explained by differing reuse factors.

| Component | Nexys Video | Zybo Z7 |
|---|---|---|
| LUT | 44147 (32%) | 35162 (66%) |
| Flip Flop | 54046 (20%) | 43084 (40%) |
| BRAM | 180 (49%) | 140 (100%) |
| DSP | 244 (30%) | 121 (86%) |

For the medium and large sized models, implementations on the Zybo Z7 were unsuccessful due to insufficient resources. As such, only the results for the Nexys Video are presented.

**Table 5.2:** Resource usage for the medium and large sized models.

| Component | Medium | Large |
|---|---|---|
| LUT | 48961 (36%) | 62372 (46%) |
| Flip Flop | 58985 (21%) | 74386 (28%) |
| BRAM | 212 (58%) | 323 (88%) |
| DSP | 274 (37%) | 381 (51%) |

From just a glance at the tables it becomes clear that the heaviest resource usage lies in BRAM. This was also the reason why the medium and large models were not successfully implemented on the Nexys Video.

## 5.2 Timing

It was discovered that implementations of the model failed at frequencies above 60 MHz. The failures occurred to due insufficient slack times within the circuit. In order to mitigate, as well as gain some additional slack, the implemented models' clock frequency was set to 50 MHz.

The time for one prediction for each accelerator is depicted in the table below, timings used were the maximum:

**Table 5.3:** Inference delay for the differing implementations at 50 MHz.

| Size | Nexys Video | Zybo Z7 |
|---|---|---|
| Small | 1.213 ms | 2.734 ms |
| Medium | 1.213 ms | - |
| Large | 1.406 ms | - |

## 5.3 Power Usage

The following table illustrates the expected power usage as reported by Vivado post-implementation. Surprisingly, the design requiring the least amount of hardware resources, namely the small model, as implemented on the Zybo, has the largest power consumption.

**Table 5.4:** Estimated power usage as reported after device implementation.

| Size | Nexys Video | Zybo Z7 |
|---|---|---|
| Small | 1.41 W | 1.972 W |
| Medium | 1.498 W | - |
| Large | 1.96 W | - |

## 5.4 Classification Results

The following table lists the balanced accuracy and F1-scores for the 3 models. For the large model, the results reflect all 40 patients. For the smaller models, patient 1 was used. Since the results are numerically the same, regardless of which device they were tested on, the results apply for both devices.

**Table 5.5:** Classification results before post-processing.

| Model Size | Balanced Accuracy (%) | F1-Score (%) |
|---|---|---|
| Small | 62.2 | 63.2 |
| Medium | 62.1 | 63.3 |
| Large | 66.3±6.1 | 64±6.3 |

A more in-depth look at the results illustrates some of the limitations of the model. The classifier completely fails to determine the resting state. Apart from the failing rest state, the classifier commonly predicts the precision sphere grasp as a tripod grasp, and struggles with the quadpod grasp. A confusion matrix for the large model is presented in figure 5.1, followed by a figure of the mentioned grasps in figure 5.2.

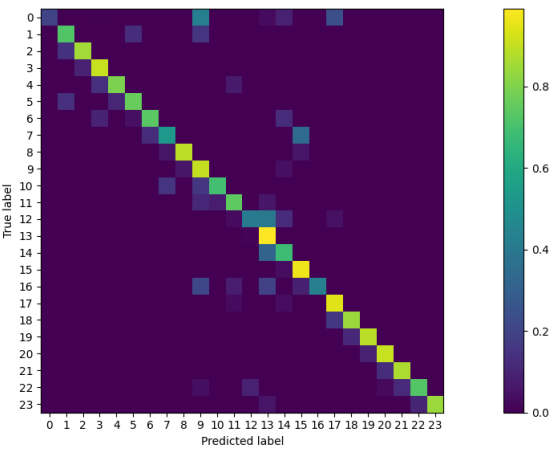


**Figure 5.1:** Confusion matrix for the large model. The conflicting classes can be identified by the disruption in the main diagonal.

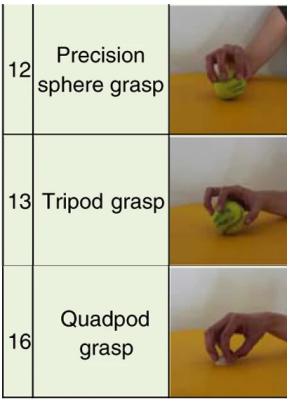


**Figure 5.2:** The precision sphere grasp is commonly classified as a tripod grasp. Apart from the resting class, the quadpod grasp is the weakest class in terms of correct classifications.

With majority voting, the results improve significantly. With a window size of 128 ms and a stride length on windows of 4 ms, 41 vote counts can be considered and still meet timing demands using the equation presented earlier. However, since the minimum stride length is limited by the delay of the classifier, the stride length could be reduced to 2-3 ms, depending on the device. This would in turn allow for more vote counts, and potentially more stable performance. In order to preserve some margin of error this was not attempted. Figures 5.3 and 5.4 illustrate performance increases in regard to larger vote counts.

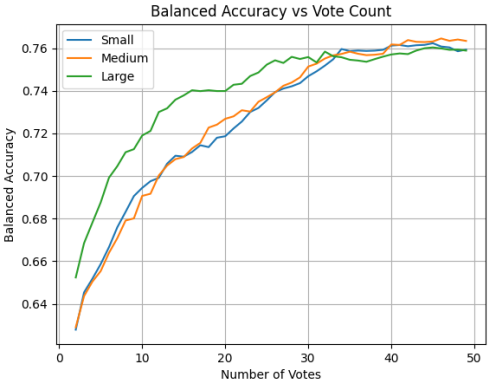


**Figure 5.3:** Accuracy improves with larger vote counts. Illustrated on subject 1.

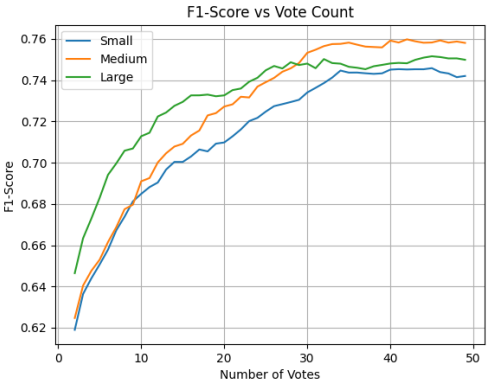


**Figure 5.4:** The F1-score improves with larger vote count. Illustrated subject 1.

It should be noted that with very large vote counts, the final accuracy and f1-score seem to converge.

# 6 Discussion

While the implementation onto the FPGA was successful, demonstrating a relatively effective tool flow, as well as including a variety of commonly used layers, there still exists room for improvement. What is probably the most difficult task is comparing the results of the project to similar works. Other works have utilized different data sets, and more successful yet more complex architectures. The model was however chosen in part to determine if the workflow could handle CNN:s, hence its implementation. As illustrated in the results section, there are classes for which the model in large part fails to classify properly. For the resting class, this error could be mitigated by implementing an additional single class classifier to augment the model. When comparing the model to works by Atzori et. al. [19], its results are slightly better. Accuracy before post-processing is relatively similar, albeit performing slightly better. It should, however be noted that fewer classes were used in this work. In terms of timing, the design was fast enough to not only achieve real time inference, but also obtain a large number of votes in the allotted time. An issue arises beyond prototyping, and that is the power consumption. The almost 2 W power figure may make such an implementation difficult to run on batteries for larger times. For an ASIC equivalent the power consumption would decrease quite dramatically. There are furthermore commonly used methods to reduce power consumption, such as clock gating which is however beyond the scope of this project.

## 6.1 Tool flow

Although the tool flow, when functioning properly, allowed for quick implementations, a multitude of issues had to be overcome throughout the project. For HLS4ML, documentation was sparse and at times outdated. Not all layers functioned properly and as such, replacements had to be made at the cost of model performance. In many cases, the internal bit widths of accumulators would overflow, requiring manual debugging. Furthermore, setting reuse factors become an exercise in trial and error. As the performance of an iteration required synthesis to calculate resource usage, a process which in worst cases took over an hour, optimizing resource usage became time consuming. As such, there are still aspects of the workflow that are not fully automated. In addition, the implementations provided by this tool flow are pre-trained. This implies that all of the steps must be iterated over in order to modify the model. Quantization Aware Training of pre-quantized layers allowed for a ”what you see is what you get” approach, where the model results after

training don't decline further. This was desirable, as it removed the risk for unexpected loss further downstream.

## 6.2 Related Work

In terms of the broader ambition to implement neural networks on FPGA:s, there exists a large portion of projects and attempts, both in terms of approaching it from high level synthesis as well as handcrafted designs that can be accommodate different network architectures [38][39]. Using FPGA:s for electromyographic classification has also been performed, but implementing traditional classifiers[40]. Using CNN:s for classification has been performed quite extensively, Triwiyanto et al. [41] provides a review containing both examples with handcrafted features, as well as with raw inputs. More recently, CNN:s have been used in conjunction with Long Short Term Memory, a type of layer which allows the model to retain memory of previous inputs with promising results [42].

## 6.3 Future Work

There is still much to be desired in terms of both model performance and tool flow improvements. Although this project laid out design considerations and a tool flow, not much is given in comparison with the existing alternative of using a microcontroller based approach. As such, future work should include comparisons of equal models implemented side by side on microcontrollers and FPGA:s, to further answer the question of whether the FPGA is a suitable alternative to a microcontroller. This idea of comparability should also be applied in terms of models and data sets, to simplify comparisons. Future work should aim to include recurrent neural networks with LSTM integration, as they have been shown to be promising in classifier architecture.

## 6.4 Conclusion

This project demonstrates that it is possible to implement relatively large CNN:s on FPGA:s for the purposes of sEMG classification. While the tool flow still requires work, it is usable, and possibly even useful, for prototyping. The CNN, was comparable to similar architectures using raw input data. That being said, although the FPGA implementation is feasible, it still remains undetermined if it is preferable to using micro-controllers.

# 7 Bibliography


[1] Diane W. Braza and Jennifer N. Yacub Martin. Upper Limb Amputations. *Essentials of Physical Medicine and Rehabilitation: Musculoskeletal Disorders, Pain, and Rehabilitation*, pages 651–657, 1 2020.

[2] Stefan Grushko, Tomáš Spurný, and Martiň Cerný. Control Methods for Transradial Prostheses Based on Remnant Muscle Activity and Its Relationship with Proprioceptive Feedback. *Sensors*, 20(17), 2020.

[3] Kevin J Zuo and Jaret L Olson. The evolution of functional hand replacement: From iron prostheses to hand transplantation. *Plastic surgery (Oakville, Ont.)*, 22(1):44–51, 2014.

[4] Maria Hakonen, Harri Piitulainen, and Arto Visala. Current state of digital signal processing in myoelectric interfaces and related applications. *Biomedical Signal Processing and Control*, 18:334–359, 2015.

[5] T R Farrell and R F Weir. The Optimal Controller Delay for Myoelectric Prostheses. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(1):111–118, 2007.

[6] Afef Saidi, Slim Ben Othman, Meriam Dhouibi, and Slim Ben Saoud. FPGA-based implementation of classification techniques: A survey. *Integration*, 81:280–299, 2021.

[7] A Shawahna, S M Sait, and A El-Maleh. FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access*, 7:7823–7859, 2019.

[8] Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific Data*, 1(1):140053, 2014.

[9] R H Chowdhury, M B I Reaz, M A Bin Mohd Ali, A A A Bakar, K Chellappan, and T G Chang. Surface Electromyography Signal Processing and Classification Techniques. *Sensors*, 13(9):12431–12466, 9 2013.

[10] Jiajia Wu, Xiaoou Li, Wanyang Liu, and Z J Wang. sEMG Signal Processing Methods: A Review. In *Journal of Physics: Conference Series*, volume 1237, page 032008, Place of Publication: Weihai, China. Country of Publication: UK., 1 2019. IOP Publishing.

[11] J Liu, A Hu, C An, and S Lv. Review of Electromyography Acquisition and Control System Based on Flexible Electrode. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 5841–5846, 2019.

[12] Hongfeng Chen, Yue Zhang, Gongfa Li, Yinfeng Fang, and Honghai Liu. Surface electromyography feature extraction via convolutional neural network. *International Journal of Machine Learning and Cybernetics*, 11, 1 2020.

[13] Daniel A Bennett, Skyler A Dalley, Don Truex, and Michael Goldfarb. A Multigrasp Hand Prosthesis for Providing Precision and Conformal Grasps. 2014.

[14] Christian Cipriani, Marco Controzzi, and Maria Chiara Carrozza. The SmartHand transradial prosthesis. *Journal of NeuroEngineering and Rehabilitation*, 8(1):29, 2011.

[15] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for Multi-Class Classification: an Overview. *arXiv e-prints*, page arXiv:2008.05756, 8 2020.

[16] I. A. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 12 2000.

[17] Zhongkui Ma. The Function Representation of Artificial Neural Network. 8 2019.

[18] Frank Kulwa, Oluwarotimi Williams Samuel, Mojisola Grace Asogbon, Olumide Olayinka Obe, and Guanglin Li. Analyzing the Impact of Varied Window Hyper-parameters on Deep CNN for sEMG based Motion Intent Classification. In *2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, pages 81–86, 2022.

[19] Manfredo Atzori, Matteo Cognolato, and Henning Müller. Deep Learning with Convolutional Neural Networks Applied to Electromyography Data: A Resource for the Classification of Movements for Prosthetic Hands. *Frontiers in Neurorobotics*, 10, 2016.

[20] Courville Aaron Goodfellow lan, Bengio Yoshua. *Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books*. 2016.

[21] Keras. SeparableConv2D layer. *Keras API Reference*, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Technical report, 2015.

[23] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Technical report, 2015.

[24] Keras. BatchNormalization layer, 2015.

[25] Sioni Summers. Feature Request: hls4ml-specific QBatchNormalization layer, 10 2020.

[26] Ian Kuon and Jonathan Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, 2007.

[27] NI. FPGA Fundamentals, 8 2021.

[28] Xilinx. Zynq-7000 SoC Data Sheet: Overview (DS190), 7 2018.

[29] Xilinx. 7 Series FPGAs Data Sheet: Overview (DS180), 9 2020.

[30] MathWorks. Benefits of Fixed-Point Hardware, 2022.

[31] Ambrose Finnerty and Hervé Ratigner. Reduce Power and Cost by Converting from Floating Point to Fixed Point. Technical report, 3 2017.

[32] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, Boston, 2010.

[33] P Banerjee, D Bagchi, M Haldar, A Nayak, V Kim, and R Uribe. Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design. In *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, pages 263–264, 2003.

[34] Benjamin Hawks, Javier Duarte, Nicholas J. Fraser, Alessandro Pappalardo, Nhan Tran, and Yaman Umuroglu. Ps and Qs: Quantization-aware pruning for efficient low latency neural network inference. 2 2021.

[35] Arjan Gijsberts, Manfredo Atzori, Claudio Castellini, Henning Müller, and Barbara Caputo. Measuring Movement Classification Performance with the Movement Error Rate. *IEEE Transactions on neural systems and rehabiliation engineering*, 2014.

[36] Javier Orlando Pinzón-Arenas, Robinson Jiménez-Moreno, and Julian Esteban Herrera-Benavides. Convolutional Neural Network for Hand Gesture Recognition using 8 different EMG Signals; Convolutional Neural Network for Hand Gesture Recognition using 8 different EMG Signals. *2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA)*, 2019.

[37] Lara Hoffmann, Ines Fortmeier, Clemens Elster, Thea Aarrestad, Vladimir Loncar, Nicoì Ghielmetti, Maurizio Pierini, Sioni Summers, Jennifer Ngadiuba, Christoffer Petersson, Hampus Linander, Yutaro Iiyama, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergo Jindariani, Kevin Pedro, Nhan Tran, Mia Liu, Edward Kreinar, Zhenbin Wu, and Duc Hoang. Fast convolutional neural networks on FPGAs with hls4ml Learning from survey propagation: a neural network for MAX-E-3-SAT Raffaele Marino-Fast convolutional neural networks on FPGAs with hls4ml. *Mach. Learn.: Sci. Technol*, 2:45015, 2021.

[38] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. 2018.

[39] Kingshuk Majumder and Uday Bondhugula. A Flexible FPGA Accelerator for Convolutional Neural Networks. Technical report, 2019.

[40] Mariano Majolo and Alexandre Balbinot. Proposal of a Hardware SVM Implementation for Fast sEMG Classification. In Rodrigo Costa-Felix, Machado, Carlos João, and André Victor Alvarenga, editors, *XXVI Brazilian Congress on Biomedical Engineering*, pages 381–386, Singapore, 2019. Springer Singapore.

[41] Triwiyanto Triwiyanto, Triana Rahmawati, Andjar Pudji, M Ridha Mak'ruf, and others. Deep Learning Approach in Hand Motion Recognition Using Electromyography Signal: A Review. In *Proceedings of the 2nd International Conference on Electronics, Biomedical Engineering, and Health Informatics*, pages 135–146, 2022.

[42] Naveen Kumar Karnam, Shiv Ram Dubey, Anish Chand Turlapaty, and Balakrishna Gokaraju. EMGHandNet: A hybrid CNN and Bi-LSTM

architecture for hand activity classification using surface EMG signals. *Biocybernetics and Biomedical Engineering*, 42(1):325–340, 2022.