Near-Realtime Low Power Epileptic Seizure Detection Using ANNs

by

Joey van Rijn

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday August 30, 2021 at 10:30 AM.

Student number:	4321413	
Project duration:	December 1, 2019 – August 30, 2021	
Thesis committee:	Prof. dr. ir. W.A. (Wouter) Serdijn,	TU Delft
	Dr. Zaid Al-Ars,	TU Delft
	Dr. O.C. Akgun,	TU Delft, supervisor
	Dr. ir. Christos Strydis	Erasmus MC

Preface

0.1. Abstract

Outstanding seizure detection algorithms using electroencephalogram (EEG) recordings have been developed over the past decade. These works mainly focus on best of class performance, which leads to computationally heavy solutions. This limits the applicability of these detection algorithms for hardware implementations such as field-programmable gate arrays (FPGAs). Which in turn limits its use in real-world applications such as warning systems or neural-stimulation systems.

In this work, a convolutional neural network (CNN) is trained and its properties reduced to minimize its footprint on hardware. The network is trained based on 1423 EEG recording sessions of 637 different patients sourced from the TUSZ epilepsy database. The input EEG data is processed by removing artifacts, then applying a short-term Fourier transform (STFT) and normalizing and quantizing this to 1-byte values. The neural network is optimized by reducing the input space, this is done by reducing the number of channels, time, and frequencies used. The CNN itself is reduced by quantizing the neural network and reducing its size. Based on these results two ANNs are selected and implemented on an FPGA, one is optimized for accuracy and one for network size.

The resulting networks have a sensitivity of 72.26% and 70.00% and an MCC-score of 0.711 and 0.591. The first FPGA implementations consume 0.296 W and 0.562 W at 50000 detections per second.

0.2. Acknowledgments

It must be an unwritten rule: writing the preface very deep in the night on the day of the deadline. With this last burst of energy, I would like to sincerely thank my supervisors Can Akgün and Wouter Serdijn. During our regular meetings, Can supplied me with tons of useful feedback. In almost all fields Can proved to be an expert, he provided me with excellent information with often steered me in the right direction. Can was almost always available, no matter the time of the day or his physical location in the world. Wouter gave my very first lecture when I started studying way back in 2013, between now and then he became a full professor. During my time at the TU Delft I have regularly relied on his wisdom, even though he was often busy as a professor, he always made time for students, something I truly admire. I find it fitting he is present at the start and end of my student career.

I also would like to thank my family for supporting me throughout my student career. Their trust in my capabilities made it possible for me to achieve my dream of becoming an engineer.

Furthermore, I would like to thank the Electrotechnische Vereeniging, these people really were like family to me. The ETV is central to a lot of my keenest memories.

Lastly, I would like to thank my friends for helping me through these sometimes difficult times. Without them, my time at this university would be way less fun. The countless coffee (or something stronger) breaks I enjoyed with them. Listening to me complaining about life the universe and everything helped me in more than they could possibly know.

Joey Maarten Simon van Rijn
Delft, August 2021

Contents

1 Introduction

Artificial Intelligence (AI) and Machine learning (ML) more specifically has led to breakthroughs in a lot of fields in recent years. Typical Machine Learning problems are hard to classify in a conditional fashion, typical examples are image classification or voice recognition. It is hard to develop an algorithm that recognizes a voice-based on traditional tools such as if statements, loops and basic signal processing such as Fourier transforms [21]. Voice just is an array of values and it is hard to define what describes a specific voice, epilepsy detection using EEG signals is very similar to voice. The EEG signals can be compared to multi-track audio recording and just like audio, it is hard to make sense of it using conditional logic.

Most machine learning networks focus on Best-of-class detection, this means that as much computational power as available is thrown at a problem, resulting in big and resource heavy solutions. This can lead to square root solutions, which means that throwing multiple(s) of the number of resources to a problem only leads to marginal improvements. More recently, interest has risen in the usage of Artificial Neural Networks (ANN) for low-power and high throughput use cases. A neural network is in essence a bunch of matrix multiplications that can be implemented on hardware such as an FPGA or ASIC. Thus, a network is trained 'online' and then synthesized to an 'offline' hardware solution. This means that if an ANN is trained and synthesized it can not be changed, but it will be energy efficient.

The path from EEG data to a synthesized circuit has three steps. These are data pre-processing, network optimization, and network synthesizing. Each step will be implemented to showcase the feasibility of low-power epilepsy detection. In essence, this work is a feasibility study of this approach.

The data processing steps has two main components. First, it should harmonize the EEG data, even in the best datasets there is a huge internal difference in quality. Sample rates differ, EEG channels have different names, extra information such as heart rate are included and faulty signals are present. The type of neural network suitable for low-level implementations has fixed input sizes and thus all data needs to have the same input space. Next, the input space should be as small as possible, as a small input space lead to a smaller network and ultimately to low hardware requirements. Short Term Fourier Transform (STFT) and a normalization step are chosen to reduce the input size from the original time series. These transformations have been chosen as they are relatively easy to implement on hardware, thus reducing complexity and power consumption.

ANNs can be seen as energy-efficient filter topologies. In a hypothetical closed-loop treatment system the detection method consumes a big part of the energy. Attempts will be made to reduce the network to find the sweet spot where the accuracy starts to rapidly decline. Lots of parameters can be tweaked to reduce the size of the network while not decreasing performance too much.

The last phase is to implement this network on a field-programmable field array (FPGA). Experimental packages exist that, with some modifications, can be used to synthesize those networks. This will result in a network that can stream seizures and showcase the feasibility of the overall approach. This work will implement the network on an FPGA which makes it hard to make accurate power predictions, however, the result of different implementations can be compared.

The existence of low-power, small-hardware solutions to detect seizures could unlock new applications. Lower power, high-quality applications have a certain value on their own. For example wearable detection devices or as part of closed-loop stimulation systems.

1.1. Problem definition

Based on the introduction in the main question of this work are defined as:

Is it possible to design a low-power epilepsy detection system with reduced input data?.

To answer this question the following sub-questions need to be answered:

- **Are datasets available that are suitable to this task of general-purpose epilepsy detection?** Multiple datasets exist, however till recently these typically contained only a few patients making it hard to generate general detection methods.
- **How to design a pre-process pipeline to clean the source data?** Datasets generally have some oddities which can reduce the accuracy of the resulting detection algorithm. An attempt will be made to filters those oddities out, reduce unnecessary data, and chose methods that are practical in hardware.
- **Which Neural Network structure is suitable for the task of predicting?** Machine learning and more specifically Neural Networks come in all forms and sizes. Typical networks used in computer science may not be the most natural match for this prediction scheme.
- **Which trade-offs are made in sizing the networks and how does this reduce accuracy?** Sizing a network is an important factor in the final size, by reducing its size goes down. Multiple ways of achieving this exist, methods such as reducing inputs and reducing network size. The goal is to map out what the effect of these techniques is on prediction accuracy.
- **How to implement a hardware solution and which trade-offs are made there?** Until recently it was very hard to implement a neural network in hardware, a lot of legwork in VHDL and Verilog had to be performed. Luckily, frameworks start to appear now which are able to handle this. However, they are not necessarily ideal, typical neural network building blocks may not be implemented and strange extra boundary conditions may appear.

1.2. Organization of this work

This thesis is organized into three chapters that explore the main topics of this thesis: data gathering and pre-processing, neural network design, and hardware implementation. Before diving into this, a bit of background is needed to understand why certain decisions are made.

Chapter 2.1 will discuss the background. First, the neurological disorder epilepsy is discussed. Next, the most common way of measuring epilepsy, the EEG in explained. Afterward, a high-level discussion of neural networks is given, this is by no means a thorough discussion but can be used to bring the reader without any prior knowledge up to speed. The chapter concludes with some miscellaneous concepts such a scoring measures, data formats used in the biomedical time series world, and existing devices that may be used for detection.

Chapter 3 begins with looking at the different datasets and selecting a suitable one. This data will then be processed to be used as the training data of the neural network. This process has two steps, in Section,3.2 the data will be cleaned, and in section 3.3 data reduction will be explained.

Chapter 4 discusses the design of the neural network. First, the architecture of the neural network is discussed, then a multitude of experiments are performed to minimize the input requirements of this network. The chapter concludes by selecting two network designs, one optimized for size and one for accuracy.

Chapter 5, implements these selected networks on an FPGA. Chapter 6, tests the selected networks. Finally, chapter 7 summarizes the work done in this work, answers the questions from the problem definition, and makes some recommendations.

2 Background

This background chapter has as goal to touch on subjects that are relevant in the zoomed-out overview of this thesis. This chapter will first touch in section 2.1 on epilepsy, the goal is to familiarize the reader with the disease, explain its origins, and what it means for a human when left untreated.

Section 2.2 will introduce EEG recordings. These are used to record brain activity, and when an EEG recording and an epileptic seizure coincide, this results in an epileptic recording. EEGs are by no means the only way of recordings seizures but as will be explained the most suitable to use in this work. The section will explain how neurologist uses this tool, what the (electrical)-challenges are, and what is actually measured.

Neural networks are a key to this work as they ultimately will act as the detection method. Due to the hype, a massive amount of media coverage, and interchangeable use of machine learning, neural networks, and artificial intelligence it sometimes feels like science fiction. However, the types used in this work is quite boring and logical for this work. A high-level overview will include the important types of networks (ANNs and CNNs) and a feel for the underlying mathematical concepts. In an imperfect analogy one could say that epilepsy is the reason, EEGs are the understanding and neural networks are the tool with which can be used to solve the problem.

This chapter also contains some concepts which didn't fit anywhere else in a natural way such as similar devices, scoring methods, and data formats.

2.1. Epilepsy

Worldwide 50 million people are diagnosed with epilepsy [16], this is likely an underreporting. In the US, where reliable data exists, around 1.2% or 3.4 million people have a form of epilepsy [30]. It can be highly interrupting and dangerous in people's daily life, people with epilepsy have a 2 to 3-times higher chance to die prematurely [16]. For some patients, a seizure can happen suddenly, which can be a risk in activities that are considered normal such as driving or using sharp items and thus limiting people's freedom. Epilepsy is a neurological group of disorders, it is caused by abnormal firing of neurons which can lead to loss of conscience. There is no known cure for epilepsy, however, multiple treatments do exist which depend on the severity and type of epilepsy. Most cases that need treatment are treated by drugs, however, up to 30% of patients don't get their seizures under sufficient control using drug treatment [14]. Due to the sheer size of epilepsy patients, this group represents a ”blockbuster treatment”; a big financial opportunity. In the most extreme cases, a stimulator can be installed during surgery and this apparatus can stimulate tissue to depress seizures.

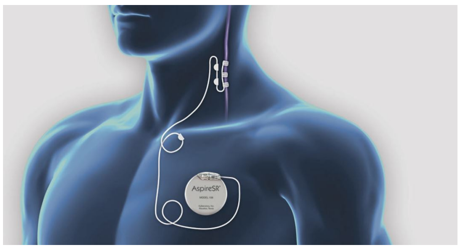
2.1.1. The disease

Epilepsy is a brain disorder, which means it is a disorder that affects the central nervous system. A seizure originates from abnormal cerebral discharges. The cause of epilepsy is mostly unknown but can sometimes be attributed to events like a (traffic)-accident, bacterial infections, or secondary immune responses [29]. How it manifests itself varies wildly, some people experience blackouts, others experience spastic reactions in one or multiple limbs and even others may experience nothing at all. Some can feel a seizure coming but for others, it can happen any moment. When such an event is observed once, it is called a seizure, when it happens multiple times epilepsy is diagnosed.

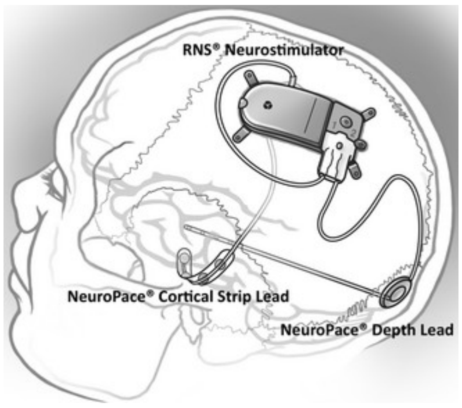
Epilepsy is often described as short-circuiting in the brain, as a seizure is in essence a (group of) neurons that start firing in an uncontrolled fashion. However, not every seizure is created equal. Different types can be classified, typically this is done using EEG recordings and the symptoms which can be visually observed such as motor reactions (spasms). Some types such as Tonic-clonic (explained later) are only observed on a subset of the recorded EEG channels, which means that they occur locally in the brain. Depending on the function of the local brain part this can give a hint in how seizures manifest themselves in the behavior of the patient.

Diagnosis is done using a combination of EEG data and direct observation/or video recordings of the patient in a clinical environment. Seizures are described in two ways, the first part is based on the origin of the seizure and how they spread through the brain, the second is on the linked impairment. Generalized seizures are thought to originate at one point and then quickly spread through the brain while focal seizures are only ”visible” in a part of the brain. The second factor describes the impairment, a lot of different annotations exist, a tonic-clonic for example points to muse stiffening and spasms while myoclonic points to muscle twitching [28].

Based on the symptoms, seizures are categorized into two categories, partial and generalized. Partial or focal seizures, cause only a part of the cerebral hemisphere to be affected. There are two types of partial seizures: simple- and complex-partial, in a simple-partial, the patient does not lose consciousness but cannot talk freely, in a complex-partial a patient gets confused and starts to behave abnormally. In generalized seizures, all regions of the brain suffer [23].



(a) A vagus nerve stimulator looks like a pacemaker



(b) The NeuroPace uses stimulating depth lead and an ECoG path

Figure 2.1: Two modern stimulators, both are installed during surgery. Once installed the devices can detect epileptic activity and treat the patient accordingly using electrical pulses

2.1.2. Impact and treatments

Epilepsy can have an enormous effect on people’s Quality of Live (QoL). People who have refractory epilepsy (unpredictable) are especially vulnerable as they can be prohibited to drive and control dangerous machinery. It is known from studies that people with epilepsy generally have a lower income, lower life span, and a lower quality of life. It has been estimated that the societal cost in the US alone is more than 42 billion per year [3].

As epilepsy is a disorder that manifests itself differently in each patient, treating the disease is hard. Most epileptic patients are treated using antiepileptic drugs (AED), around 30 different such drugs are known [11]. Finding the right one is often a process of trial and error, using different doses and managing side effects with even more drugs. Sometimes, this result in refractory epilepsy, this means that seizures are not well-controlled by medications. The National Health Services (NHS, the health system of England) thinks that AEDs work sufficiently for 7 out of 10 patients.

As the group of epilepsy patients is quite big even this relatively small percentages of patients dealing with refractory epilepsy means that there is a sufficient market for alternative treatments. There are three big categories: brain surgery, where a part of the brain is removed, can work when only a small part of the brain is affected. Then there are two stimulation techniques, there is deep brain stimulation (DBS), and vagus nerve stimulation (VNS).

The vagus nerve is an important connection between the brain and the rest of the body. Electrically stimulating this interconnect has been proven to suppress seizures [4]. An installed VNS is shown in figure 2.1a, the device looks like a pacemaker. It works by giving timely electrical pulses to the vagus nerve, the most modern versions also monitor the heartbeat, which can be an indication of seizure and can adjust the stimulation modality based on that information. This creates a partly closed-loop system. Regulators have found VNS useful in practice, the NHS reimburses it when certain conditions are met [4].

Deep brain stimulation is a method whereby a stimulator is implemented inside the brain. This is done by brain surgery, part of the scalp is removed and deep stimulation leads are inserted. Like VNS, pulses are used to suppress seizures however from inside the brain. The approach works by stimulating the (pre-detected) source of the seizures and thus is only suitable for a subset of patients with tonic-clonic seizures. The most novel technique is using this in a closed-loop fashion, detecting brain waves and stimulating when a seizure is detected. An example of such a closed-loop simulator is the NeuroPace [18], a device that has been FDA approved and has a publicized long-term efficiency study [12]. However, it should be noted that to the best of the author's knowledge no public health care system currently reimburses the use of DBS systems for epilepsy.

2.2. Electroencephalography

Electroencephalography (EEG) is the most used tool to analyze brainwaves analysis, the method is (relatively) cheap and ex vivo. Pads (typically between 16 and 32) are connected to the scalp, the pads are then used to record electrical activity. While EEGs are not a very granular approach to recording activity, they give a sense of what is happening in brain activity over a long(er) time. Doctors use these recordings for a wide variety of applications such as detecting brain damage after strokes, epilepsy diagnosis, and sleep research. Big datasets of well-labeled epileptic EEG recordings are publicized, making them the best biomarker.

2.2.1. History

The first epileptic seizure spike measurement was made in 1934 by Fischer and Lowenbach [5]. Before this, the only way of detecting epileptic behavior was by physically observing a patient's behavior. Fischer and Lowenbach used the then-novel technique of Electroencephalography (EEG). The EEG measures the electrical activity which can be recorded by putting pads on the outside of the scalp, these form a conductor for the electrical activity inside the brain and, thus, can be recorded. An early version of this was not small and easy to handle though, figure 2.2a shows the hardware needed to make such an old recording. Lots has changed since the 40s. More use cases for brain recordings (EEGs) have been found, devices have become cheaper, smaller, and more capable. Despite this, most EEGs are still done in clinical settings, the results are interpreted by (expensive) medical personal [15]. This is not only an expensive and labor-intensive job, but it also limits the ways potential of EEGs.

(a) A man being wired up to an EEG machine...

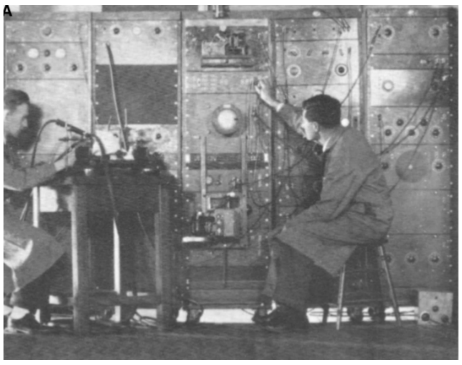

(b) ... and the hardware needed to make the recording [24]

Figure 2.2: In modern times an EEG device is a simple and small electrical device. In the early days of EEG recordings, quite a bit of hardware was needed for a recording this can be seen in these pictures

(a) A modern EEG path...

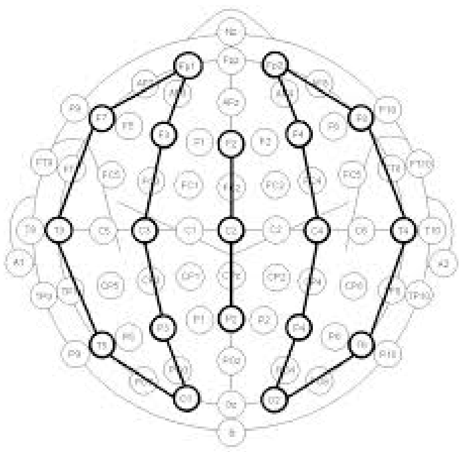

(b) ...uses a standardized grid

Figure 2.3: Figure 2.3b shows the standardized EEG grid. The locations are spaced in set intervals, the blacks lines select a subset of all those locations. Figure 2.3b, shows a modern solution to place the pads on the right location

2.2.2. Properties and handling

The quality of an EEG depends on the quality of the recording. This is especially important in neural network applications where the comparable quality of different EEG recordings is of importance. To get high-quality recordings two factors are important. First the quality of the recording device, noise suppression techniques, and amplifier quality matter. Luckily, a whole host of recordings devices exists and the requirements are not extremely hard to reach (measuring millivolts differences up to 500Hz) with modern electronics. For the pads that are connected to the scalp, important factors are the capacitance and resistance between the pad and the measurement system, this can be improved by adding gel, the lower this property the better the recording.

An EEG measures the group activity of neurons. When a neuron fires it creates a spike of electrical activity. It is impossible to measure each neuron separately as there are over 1 billion of them in a human brain [8]. Neuron firing and reversing their potential tends to happen quite randomly. However, when large chunks of neurons fire, this can be detected by the EEG "probes" which make it possible to record brain activity.

2.2.3. Montages

Figure 2.3a shows a standardized pad that can be used to place EEG pads/probes on a person's head. The pad contains standardized names which correspond to places on the scalp. This grid is standardized to make recordings comparable and reproducible. Generally speaking, subsets of these pads are selected to make an EEG recording. The most wildly used placement system is the 10-20 system, this means that the longitudinal and transverse position of pads on the brain is 10% and 20% spaced [1]. The amount of channelsvaries wildly, sub-hairline ones ("behind the ear") use 3 or 4 channels [6]. In this work, the focus will be on the most common 16 to 20 signals as big datasets of those are available.

The black dots in figure 2.3b show the 19 channels selected in a recording. These nodes are selected as they give good spatial coverage of the brain. Most systems record per channel, montages can later be created by simple mathematical operations on a computer. Unpaired (channel to ground) provide the cleanest signal, but the recorded signals can suffer from DC offset, a relatively big voltage range (between different channels), and 50/60Hz distortion. To overcome this problem montages are used they relate channels to each other. A bipolar montage takes two channels and subtracts those from each other (ie Fp1 - F7 and F7 - T4), referential montages uses a common signal to subtract, for instance, use the relative stable ear channels as reference or the common average where the reference is the average of all channels. However, when naively implemented these montages can lead to considerable voltage distortion. An engineering solution to this is group montages such as the "double banana"-montage shown in figure 2.3b. It uses a circular bipolar design, (Fp1-F7, F7-T5, and T5-T3), as the channels are relatively close to each other it can be assumed that the two channels have a relatively high correlation which results in less distortion when subtracted.

2.2.4. Interesting EEG data

EEGs are only interesting when they can be connected to real events. Typically this information is in the frequencies beneath 50Hz. In theory, this means that EEG recording machines could function at a $f_s = 100Hz$ sampling frequency. Neurologists often look at FFTs of EEGs and classify these using Delta (1-3Hz), Theta (4-7Hz), Alpha (8-12Hz), Beta (13-38Hz), and Gamma (39-42Hz) "waves" [20]. Peaks in these frequency bins are supposed to correspond to typical patterns during certain activities. Though, different sources quote different values for the boundaries between frequency regimes. The interpretation is mainly a way to practically order signal information.

2.3. Machine learning

This section will aim to give a relevant introduction to Machine Learning and more specifically Artificial Neural Networks (ANN). The aim of this section is not to give a complete explanation of the inner workings of neural networks, the field is simply too big so no justice could be done. However, a (very) high-level explanation will be given, and (more importantly) some specifically relevant concepts will be highlighted.

2.3.1. A short history

For most of the existence of computers, code was written as conditional logic. If statements that are embedded in for loops in functions that are inside classes. In the last decade, things have been changing, not all problems are easily solved using conditional logic, think of how one would try to recognize words from audio recordings using conditional logic. The approach to solving this is not new, already in 1950, Alex Turing, also known for breaking the Enigma Machine during the Second World War, proposed machine learning (ML) [25]. Machine learning tries to answer programmatic problems not in a conditional fashion but in a stochastic fashion. This typically results in answers to classification problems that don't give exact answers, but answers like "prediction is a two with 65% confidence". For a very long time, this method was not really feasible in practice, thus entering the so-called "AI Winter". Around the turn of the century, two important things changed. The first was a focus shift to data-driven approaches, where representing data was used to show examples which computers could ”learn from”. Learning needs a lot of computing power, due to the minimization of chip feature sizes and the related improvements in computing power (as described by Moore’s law) this came available.

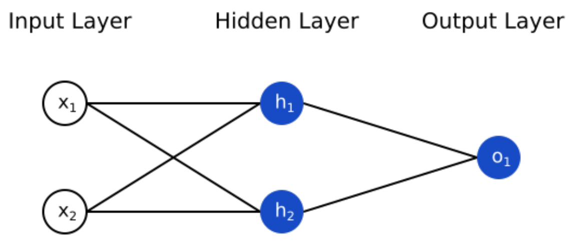


Figure 2.4: An illustration of a very simple linear neural network. The network has three phases: the input layer, which is connected to the source data, a hidden layer, which adds learnable parameters to the network, and an output layer that labels all possible outcomes)

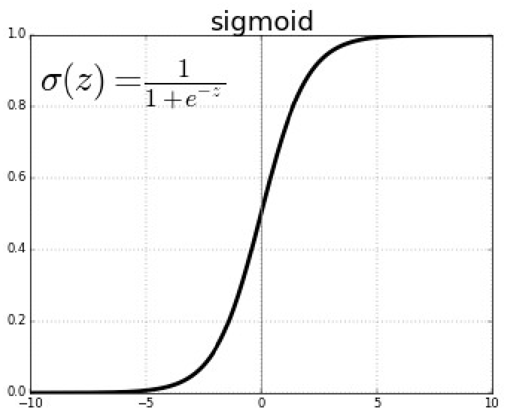


Figure 2.5: A sigmoid function translates numbers in a very wide range to a number between zero and one

2.3.2. Basic neural networks

The method which gained the most popularity is a class called artificial neural networks (ANNs). The idea is to use a bunch of algorithms that try to recognize the relationships in data through a process of repeatedly showing labeled examples. Figure 2.4 shows a very simplified version of an ANN. There are three distinct types of layers in there. First, the input layers which are as big as the input, an example image with 10 by 10 will be 100 values wide. Then there is the output layer, which corresponds with the number of values one wants to classify, thus if those images would be numbers from 0 to 9 this would be 10 values. The epilepsy detection in this work is a binary classification problem as only the difference between background signals and seizure signals are relevant. In between these values can be layers of nodes, the Figure shows only one hidden layer, but there can be multiple. These can be either wider or denser than the input layer.

The graphical explanation does represent a mathematical description between them. The nodes are connected using simple logic which leads to simple math, each node and line has a weight. Which leads to formulas like equations 2.1 and 2.2.

$$h_1 = w_1 * x_1^1 + w_2 * x_2^1 \tag{2.1}$$

$$o_1 = h_1 + h_2 \tag{2.2}$$

This can be translated to matrices relatively easily. For instance, Equation 2.1 would be translated into equation 2.3. Computers excel in solving these kinds of equations (though this doesn't matter much once implemented in hardware). Practical solutions use matrices that are much bigger, multiple hidden layers are typically used, resulting in multiple matrix multiplications. These types of neural networks are in essence just matrix multiplications, multiplying a set of numbers at the input with those matrices should result in a correct prediction.

$$\begin{bmatrix} o_1 \\ o_2 \end{bmatrix} = \begin{bmatrix} x_1 & h_1 \\ x_2 & h_2 \end{bmatrix} * \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} \tag{2.3}$$

To get the correct answer the program needs to tweak the values in the matrix such that it will produce correct predictions. This is in essence recognition of patterns of the applied input data. The way to do this is by showing examples and determining if the predicted value is correct or wrong. If the network predicts it wrong the network should be ”punished”. During the process, a network makes a prediction and the values are adjusted based on how wrong this prediction is.

The values in the network can quickly escalate to quite high numbers, this results in quite high numbers at the output of the network. It is difficult to train networks correctly if the values of the output have a very wide range, to help to mitigate this problem, layers are introduced between (some) layers which have a goal to narrow this range. Figure 2.5 shows such a function which translates numbers in a very wide range to a number between 0 and 1. In the training phase dropout layers are often used, these delete a certain random fraction of connections in the network (or zero them in the matrices) for each training batch. This makes sure the network is detecting patterns and not trains to just recognize a specific input. It is not uncommon to delete 50% of learnable parameters when using a dropout. Dropouts are only used in the training process, there are not needed when the network is used to classify inputs (in production mode).

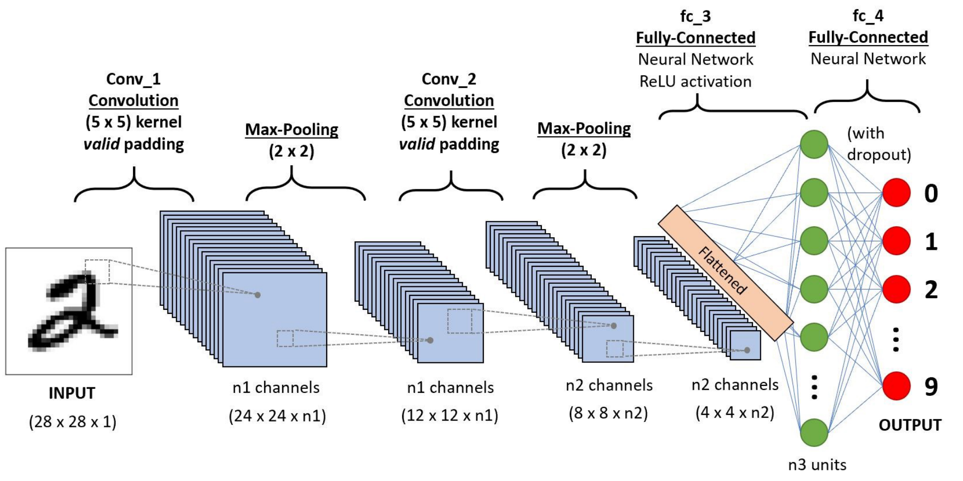


Figure 2.6: A typical architecture for a Convolutional Neural Network with convolutional layers and a conventional part after

2.3.3. Convolutional Neural Networks

This work uses convolutional neural networks (CNNs), they are a logical extension to ANN's. A typical CNN usage is to classify images, CNN keeps spatial information separated and can result in smaller networks. Figure 2.6 shows such an example, it shows a typical example of image recognition, the input has a Width and Length of 28 by 28 pixels, it has one channel (the image is black and white), if it was a color image there would be 3 channels (RGB). The equivalent of this in EEG would be the amount of EEG channels used. The mathematical description of a (2D)-convolution layer is given in equation 2.4.

$$out(N_i, C_{out}) = \sum_{k=0}^{C_{in}-1} weight(C_{out_j,k}) * input(N_i, k) \tag{2.4}$$

In this work a Short Term Fourier Transform will be used, the result of this which can be interpreted as picture (X and Y axis are time and frequency) and with the connected channels as colors (but 18 of them).

CNN's have a frame ("kernel") which it connects to a value in the next frame. Figure 2.6 shows such a frame, on the left a kernel of 5x5 is selected, which then translates to 1x1 pixel in the next layer. The bigger the kernel size the smaller the following layer (Original-size - kernel-size + 1). This can result in much smaller networks compared to the classical linear approach. The number of channels can be freely varied, in the example, the input layer only had one layer (black-white value), which is extended to 11 in the second layer. The CNN layers is great as it can reduce the size of the network while keeping spatial information. It is typical for such a network to end with a linear network, all resulting values in the last CNN layer are simply put next to each other (flatten) and function as the input of the linear network.

2.3.4. Software

Software for neural networks is readily available as machine learning applications are used in both industry and academia. The biggest users of these techniques have released their software as open-source software. This work is almost entirely written in Python as these packages are (primarily) released in Python and Python is a very good catch-all programming language. The two most popular packages are PyTorch and TensorFlow, one by Google one by Facebook. The difference between the two is quite small for relatively standard use cases such as this work. Only in the state-of-the-art sometimes there are differences in implementation. PyTorch is a little more popular in the scientific world while TensorFlow is more used in production. PyTorch was chosen for this work.

Brevitas

Brevitas [17] is the framework used to quantize networks. PyTorch only works with float32 and float64 data types. In "vanilla" PyTorch it is possible to convert networks to int8 or int16 networks and this is focused CPU applications. For software solutions, it doesn't make sense to go lower than int8 as CPUs are not optimized for these kinds of instructions. This is the goal of Brevitas, it implements a layer upon PyTorch to make quantized neural networks (QNN's) possible. It is possible to quantize numbers that are originally float32 to 1, 2, 3, or any number of bits. Brevitas implements a hole setup function to deal with this such as translation layers, native implementations, and helper functions. It also works together with another experimental package called FINN which can be used to transfer a Brevitas-Pytorch network to an FPGA. Both Brevitas and FINN are created by the Xilinx research lab and are both still experimental.

2.4. Miscellaneous

This section will explain some concepts not big enough for a section but important enough to give a short introduction. Existing devices are shortly discussed including a back-of-the-envelope energy consumption calculation. Then the ways to benchmark the quality of the system are explained. Lastly, some miscellaneous even in miscellaneous chapter concepts are touched upon.

2.4.1. Existing devices

A logical and, at least for this work, existential question is: "Why Would We Want Low-Power Epilepsy Detection?". A lot of different answers are possible to that question, smaller batteries and/or more lifetime or less heat production and thus safer to implement in the brain. To get a better understanding of power consumption it is good to look at integrated systems which detect epileptic seizures. Multiple devices for Deep Brain Stimulation (DBS) purposes have been approved in recent years. Medtronic Inc has long produced such stimulators to treat Parkinson's, recently its Percept system was approved in Europe and the US. The percept system is a "general purpose" DBS system which means that it can be used to treat all kinds of illnesses. Similar devices are the Boston Scientific Vercise and the Abott Infinity, like the Medtronic devices the primary target seems to be Parkison's disease. These devices typically consist of a controller implemented in the chest with wire through the neck to the brain where it is connected to a DBS stimulator.

NeuroPace

The NeuroPace is a device that is specifically build to treat epilepsy. It is a closed-loop system, different then the competitors, it doesn't put the control device in the chest but instead cuts out a part of the scalp which is replaced with the necessary device. The company published the result of a phase 4 study with 9 years of data [13]. In traditional open-loop solutions, commonly the stimulator consumed the most energy. They stimulate onset times during their lifetimes, this typically means multiple times per hour or even minute. The combined energy consumption used by stimulating typically counts for the biggest chunk of energy usage. This is not the case for closed-loop systems, these only stimulate short chunks of time. Breaking such a device down into two parts, a stimulating system, and a detecting system it would become clear that the bulk of the energy is used by the detection part which is continuously on. The way the detection algorithm works is part of trade secrets, but it is probably some kind of peak detection based on ECoG. An approximation of the energy consumption can be made by looking at the mean stimulation numbers as published in the long-term study.

The NeuroPace on average stimulates 1028 times per day, the battery of the NeuroPace is $705mAh$. The most common stimulation pattern is 2 bursts of stimulation between 100Hz to 200Hz (take 150Hz) with a pulse width of 160 microseconds and 100 milliseconds burst duration. Typically the stimulation happens at 6mA at which it survives for 3.5 years.

$$t_{stim-time-in-3.5y} = N_{bursts} * t_{burst-length} * \frac{1}{f_{stim-freq}} * t_{pulse-width} * t_{days-in-years} * t_{avg-battery} \tag{2.5}$$

$$t_{stim-time-in-3.5y} = 2 * 100ms * \frac{1}{150} * 160\mu s * 365 * 3.5 * 1024 = 6303.69s \tag{2.6}$$

Equation 2.6 shows that this works to about 6304 seconds or 1.75 hours of pure stimulation over the lifetime of the device. Considering this happens at 6mA this means $6mA * 1.75 = 10.5mAh$ is used for stimulating purposes. Thus as Equation 2.7 the device uses 67 times more energy for other things than stimulating.

$$N_{times_{m}ore_{e}nergy_{d}etecting} = \frac{battery-size}{stimulating-use} = 705/10.5 = 67 \tag{2.7}$$

This is obviously not a perfect comparison, but it shows that the detection system uses at least an order of magnitude more energy than the stimulation system. It is thus worth it to optimize the detection system as even modest improvements will result in big energy savings.

2.4.2. Interpreting results

Seizure detection is a binary classification problem, this means that there are only two possible outcomes. The outcome can be either a seizure or background. When comparing these predicted outcomes with the labels four outcomes are possible. Table 2.1 shows the possible outcomes.

When comparing network performance people, are typically interested in the accuracy, which can be derived using the values from table 2.1 using equation 2.8.

Value	Label	Prediction
True Positive (TP)	Seizure	Seizure
False Positive (FP)	Background	Seizure
False Negative (FN)	Seizure	Background
True Negative (TN)	Background	Background

Table 2.1: All possible scenarios in binary prediction. When TP, FP, FN, and TN are put in a matrix form they are known as the confusion matrix

$$accuracy(\%) = \frac{TP + TN}{TP + FP + FN + TN} * 100 \tag{2.8}$$

Accuracy is great to measure of success when the categories of the binary classification are equal. This is not typically the case in epilepsy datasets. For most patients, a seizure is an exception, thus, most data is background. A 95% background signal and 5% seizure are typical. This makes accuracy not a good measure, if in such a 95:5 dataset all would be classified as background this would still give an accuracy of 95%. This is why sensitivity and specificity are wildly used to benchmark networks. These measures are also known as the True Positive Rate and True Negative Rate and are shown in equations 2.9 and 2.10. These measure the relative correctness of the background and seizure signals.

$$Sensitivity/Recall/True - Positive - Rate(TPR) = \frac{TP}{TP + FN} * 100 \tag{2.9}$$

$$Specificity/True - Negative - Rate(TNR) = \frac{TN}{TN + FP} * 100 \tag{2.10}$$

Equation 2.11 shows the precision, this gives a measure of how exact the network is. It compares the correct seizure prediction with the wrong predictions in the background. As the amount of background data is around 19 times bigger than the seizure parts, it is thus very hard to score high in this metric as not a lot of mistakes are allowed.

$$Precision = \frac{TP}{TP + FP} \tag{2.11}$$

Multiple measures are hard to compare. This is why the F1 and MCC scores are used, they try to measure the quality of a network in one number. The F1 is the most used single score and is shown in equation 2.12. Results are bounded between zero and one. As can be seen, the true negatives are not included in this score, this is why some don't like this score.

$$F_{score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} = 2 * \frac{precision * recall}{precision + recall} \tag{2.12}$$

A second scoring is introduced, the Matthews correlation coefficient (MCC) in equation 2.13. This score does contain all elements of the confusion matrix (see table 2.1). The scores are measured between -1 and 1, where 1 is best, 0 is as expected when guessing randomly. In practice, both scoring algorithms give pretty similar results, however as MCC is considered best and F1 is most widely used both are given in this work.

$$MCC = \frac{(TP * TN)(FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.13}$$

Specific to epilepsy detection there are two more important metrics. The first is wrong predictions per hour because seizures (hopefully) don't happen often (orders of magnitude less than background time) therefore it is very hard to develop algorithms (albeit classical approaches or newer data-driven approaches) that never accidentally flag background signals. This describes the number of wrong seizures (FP) are detected during detection. Equation 2.14 describes dividing the total time of FP in one hour by the total time of correct predictions.

$$Wrong - predictions - per - hour = \frac{t_{false-positive}}{t_{typical-recording}} \tag{2.14}$$

Another important metric is the time to prediction. An Analog-to-Digital Converter (ADC) is used to sample the EEG signals, this makes time discrete. These discrete points are then bundled in bigger chunks which typically span multiple seconds which are then fed to the prediction network. As a prediction can't be made before all data points are recorded, prediction can only happen after the fact, the time between the actual event and prediction (assuming the prediction is correct). It is obviously more difficult to create a prediction algorithm that can predict using less data point (time). This value we call the time-to-prediction.

2.4.3. EDF data format

The European Data Format (EDF) is a standard file format used to save medical time series. It is used to save EEGs but uses also includes electrocardiography (ECGs) and electromyography (EMG). Besides the time signals, it also dictates how the metadata of the recorded time signal should be saved, such as sampling frequencies, min and max values, patient data, and labels. When people talk about EDF, they are mostly referring to EDF+ which supports bigger data types such as float32. The format was standardized in March 1990 in the city of Leiden. Today EDF, at least in the EEG field is the main way of sharing recordings, commercial recording sets often include an option to save in EDF or recordings are converted to EDF before being shared.

A word on predictions and classifications

Prediction and classification are used in two different contexts in this work which may result in some confusion. One is in the context of neural networks, prediction here means how the network classifies the given data. Thus, given a piece of data does, it ”think” that it was a seizure or a background signal. The second is in the context of EEG signals/annotations. Here, prediction means detecting a seizure is going to happen based on EEG data and classification is the act of defining if something is, or isn't a seizure. It is extremely hard to predict seizures, and thus, these works try to classify seizures when they happen. As there is some time between recording and letting the neural network predict, the prediction of the neural networks always occurs after the fact.

3

Datasets and pre-processing

This chapter will first explain the process in which the dataset was selected and what the properties of this dataset are. In the second part, the pre-processing steps will be explained, first, the shortcomings of the dataset will be pointed out, and then possible fixes for these problems. The last part will explain ways to reduce the size of the network. The chapter will conclude by summarizing the whole pre-process pipeline.

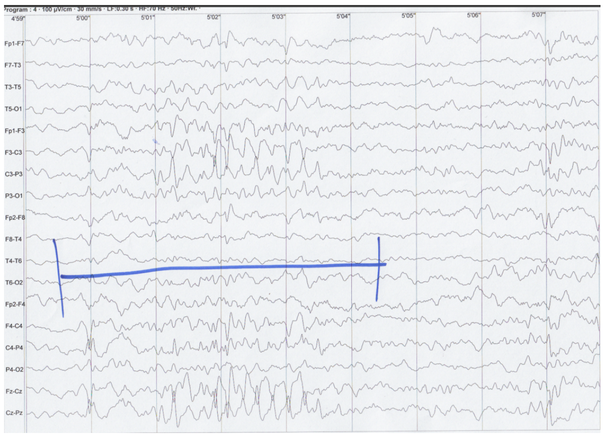


Figure 3.1: Doctors are not engineers when it comes to working with exact data

3.1. Datasets

3.1.1. Typical problems with datasets

A typical neural network tries to find patterns in a sea of data. The type of network used in this work does achieve this by modeling itself steered by the labels provided by the dataset. This in effect means that the performance of this kind of network can never be better than the quality of the annotation of the dataset. This may sound like an obvious statement, however, it plays a big role in datasets containing biological signals. Interpretations are done by doctors which creates a gray zone. To put this in perspective, figure 3.1 shows a real annotated EEG recording by a doctor, little differences in digitizing (setting the start and endpoint) give a degree of noise to a dataset which ultimately results in lower accuracy.

To make matters even worse, *what* doctors will annotate will differ depending on the situation. When figuring out where seizures originate from, the level of detail is different than when doctors analyze long-term data. A point of concern is the inter-ictal period (the time between seizures), when this period is small two seizures are combined into one. What is small is however up for debate and depends on the length of recordings, the average length of seizures, and, quite frankly, the mood of the doctor. This results in a gray zone. Publicly available datasets are typically made of historical data of (university) hospitals. This historical data is checked and made anonymous, in this process some of the original contexts get lost. Extra channels can be used to excite a seizure or the recording is made to check how the patient reacts to a new type of epileptic drug.

For these reasons, it is quite unreasonable to expect 100% prediction accuracy when training on a dataset. However, the ultimate test of the quality of a dataset is performance relative to another dataset. In the future, it would probably be better to let AI annotate and only correct mistakes, this may lead to higher-quality datasets and less ambiguity between recordings.

3.1.2. Available datasets

Multiple datasets exist, the most important factors for such datasets are size and quality. However, a lot of additional factors play a role, such as the number of patients and metadata about recordings (which kind of rooms, setups, kind of epileptic seizure).

The **Bern Barcelona dataset** [2] is a dataset that consists of long-term data of 5 patients. Focal and non-focal seizures are included, as well as recordings during typical activities such as the opening and closing of eyes and lifting objects. The database is available as is, no additions are the be expected. It consists of around 2 hours of seizure data and 2 hours of background-ish signals (activities). The dataset used to be pretty popular to showcase patient-specific detection as the quality of the 5 recordings is excellent and specially made for the purpose of machine learning.

The **Epilepsiae dataset** [9] is a combination of European University hospital databases. A European grand was used to combine these databases and harmonize the recordings and create ”the biggest epilepsy database in the world”. The website commercially advertises the EEG data of the first 30 patients. Presumably, this means that the other 250 patients are not yet available. The website states that currently all databases are being harmonized, however, there seems to be no activity and the websites used for the project have not been updated for a few years. Ultimately the project claims to have data from 250 patients, 2500 seizures, and 45000 hours of EEG recordings. The project seems to be not actively managed and the website hasn’t changed during this project.

The **CHB-MIT** [22] is a dataset published around 2010. The dataset is not actively managed and contains 24 different patients. Till recently it was the biggest freely dataset and quite popular, as it was the dataset used in most papers until recently. No active work is done and the total set consists of a few hours of EEG recording and around 900 seconds of seizures.

The **TUH (Temple University Hospital) EEG Seizure Corpus (TUSZ)** [7] dataset is a reaction to the lack of big high-quality datasets. The project specifically mentions the popularity of the CHB-MIT database as motivation for the project: the importance of epilepsy detection combined with the research interest measured in research papers made it worthwhile to create a big curated dataset. The aim of the Temple University project is to make the biggest available EEG database, currently, 692 different patients with 293 hours of seizures and 1074 hours of total time are included. Documentation of the dataset is excellent and provides meta-information about the patient and the context in which it was recorded. The project is very open about the data collected and sorted, further updates in the future are to be expected.

Comparison

Combining all databases into one big database would result in the biggest possible database, however, this is not common for two reasons: comparability and minor differences. People like comparing their work, if these are based on the same dataset comparison becomes a lot easier. Sizes, data quality, and amount of patients are similar. Minor differences when combining datasets can also play a role in degrading the quality of the net results. Table 3.1 gives an overview of all information found in these datasets.

	Dataset			
Name	Patients	Seizure [h]	Normal [h]	Active
Bern-Barcelona	4	4		No
CHB-MIT	24	?		No
EPILEPSIAE	250	? (at least 2500 recordings)	Yes	
TUSZ	280	74	293	Yes

Table 3.1: This table tries to summarize the properties of the four datasets
*Exact numbers are not known, these numbers are an indication

For this work, the Temple University database is selected. Their database is actively managed (more updates are expected in the near future). It is clearly the biggest and designed especially for the lack of big high-quality databases. It is quite simply worlds apart in all measures and thus the only logical option.

Type	Number	Total time (s)	Mean time (s)	Min (s)	Max (s)	std
TCSZ	48	5548.09	115.59	11.54	2343.39	331.42
FNSZ	1836	121139.22	65.98	1.85	2379.65	143.59
CPSZ	367	36321.03	98.97	4.96	1303.0	112.55
GNSZ	583	59716.69	102.43	2.57	2448.0	265.41
SPSZ	52	2145.82	41.27	20.14	97.06	20.01
MYSZ	3	1311.99	437.33	19.11	686.89	364.44
TNSZ	62	1204.01	19.42	4.74	62.34	9.37
ABSZ	99	851.99	8.61	1.40	26.43	5.37
Total	3050	228238.85	74.83	1.40	2448.0	172.65

Table 3.2: Multiple types of seizures are present in the dataset, some in bigger numbers than others. TCSZ: tonic seizure (patient aware, small stiffness, complete brain), FNSZ: focal non-specific seizure (local, no specific effects), CPSZ: complex partial seizure (local, patient unconsciousness), GNSZ: generalized non-specific (general, but not specific), SPSZ: simple partial seizure (local, patient consciousness), MYSZ: myoclonic seizure (jerks of limbs), TNSZ: tonic seizure (all body, stiffening of the body)

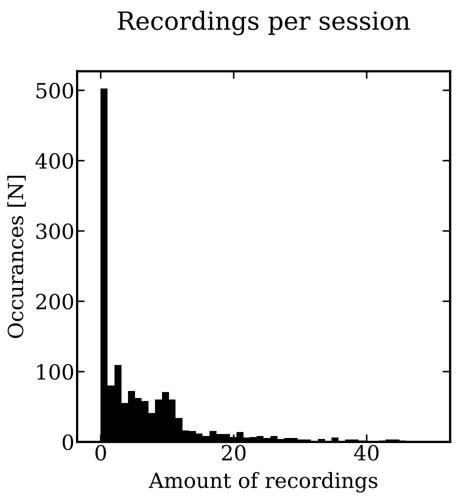
Sample frequency (f_s)	Occurrences (N)
250Hz	1046
256Hz	3751
400Hz	637
512Hz	127
1000Hz	51

Table 3.3: Multiple types of seizures are present in the dataset, some in bigger numbers than others.

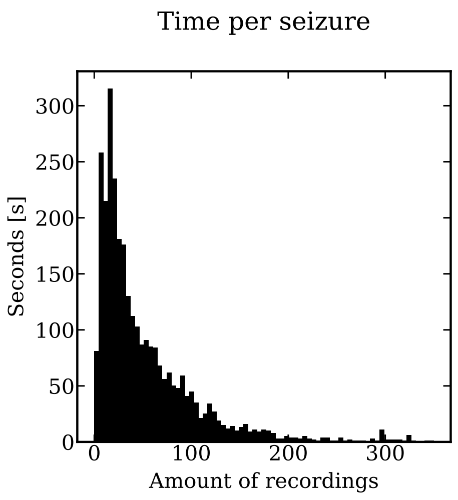
3.1.3. Temple University Dataset

The selected dataset, the Temple University Seizure Corpus (TUSZ) is part of the bigger project, the TUH EEG Corpus dataset. Consisting of 30,000 hours of EEG recordings made in the Temple University Hospital between 2002 and now. Data has been recovered from CD-Roms and old hard disks and converted to the EDF standard, then checked and made anonymous. This data is used to make multiple datasets such as the TUH EEG Artifact Corpus (TUEV) which contains EEG data labeled with labels such as eye movement, chewing, and shivering among others.

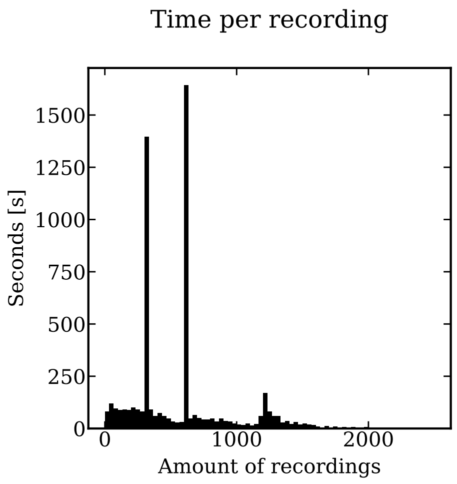
The data in the TUSZ dataset is thus a subset of the total Temple University-Corpus. The raw EEG recordings are accompanied by clinical report files which contain information about the recording added by the neurologist after analyzing the EEG. In the (complete) TUH EEG Corpus dataset less than 0.1%



(a) The histogram shows the number of recordings in patient sessions



(b) This histogram shows the length of seizures in recordings



(c) This histogram shows the length of recordings

Figure 3.2: Three ways to look at the dataset, (a) how many recordings does each patient have (b) how long are seizures and (c) how long are complete recordings.

of recordings contain actual seizure events. A pre-selection by searching through the reports for words mentioning seizures or epileptic behavior was used. The dataset comes with a reference document, where for each recording, metadata such as the seizure type, recording location (ICU, clinical), recording type (Long term or routine), and such are mentioned. Recordings are organized in a standardized way, in [patient]_[session]_[recording] format.

Patients may have multiple sessions, this may happen when patients are recorded multiple times (on different days). A session can also have multiple recordings as sometimes recording devices only allow for 10-20 minutes, or channels are modified. The total dataset has **5612** different recordings, which come from **1423** different sessions. These sessions originate from **637** different patients, this works out to 2.2 sessions per patient on average. The 5612 recordings have a total length of 923 hours, averaging 592 seconds, the shortest is 11 seconds, and the longest 3598 seconds (almost an hour). Table 3.2 gives more information about the types of seizures present and the length of them. Tonic-clonic seizures (TCSZ) is the most common subtype with a significant amount of generalized non-specific (GNSZ) and complex partial seizures (CPSZ) also present. Other categories are too small to analyze separately in the end results. Of the 5612 recordings, 4463 don't have any seizures in them. Figure 3.2a shows how many recordings patient sessions typically have, one by far the most common number. This means that the recording only exists as a background recording, quite some patient sessions consist of 1-15 recordings, more becomes increasingly rare. Figure 3.2b show the histogram of the seizure lengths in recordings, most recordings are between 10 and 200 seconds with a peak around 20 seconds. Figure 3.2c shows a histogram of the length of all recordings (a recording can contain zero, one, or multiple seizures). Two big peaks are observed at 300 and 600 seconds, corresponding to 5 and 10 minutes which seems to be a typical length for recordings. Presumably, this is because older machines were only able to record short time frames.

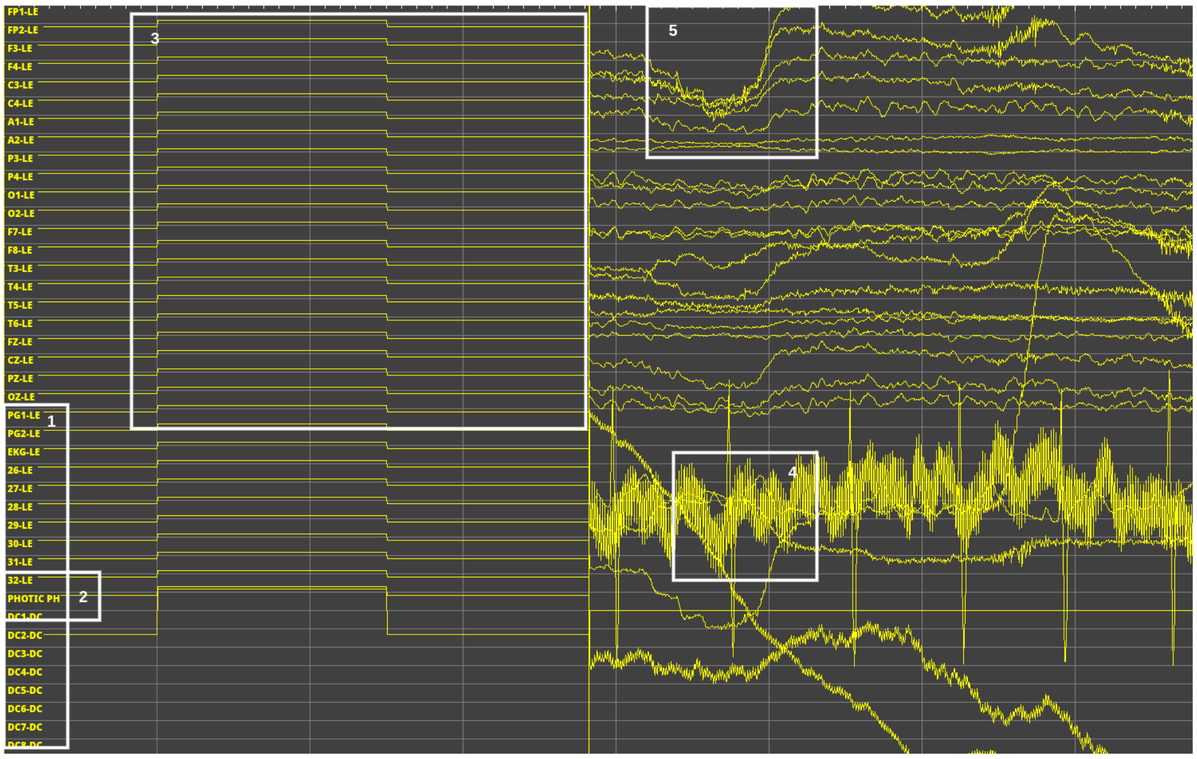


Figure 3.3: A typical EEG recording from the TUSZ dataset, 1 $second/div$ and 100 $\mu V/div$, recording: $675 - s002 - t001$. Showing: (1) Too many channels (2) Stimulus signals (3) faulty signals at the start (4) Low-frequency distortion and (5) high-frequency distortion

3.2. Data pre-processing

The selected TUSZ dataset is not directly usable, certain tweaks need to be performed to make it suitable. Some recordings or patients may be filtered, tweaked, or excluded. This is perfectly alright as long as it happens programmatically and is reproducible. In this work, special thought is given on how to transfer these pre-process steps to a workable hardware solution. Especially important is that pre-processing is relatively easy and computationally cheap to perform on hardware. The amount of steps involved in transforming the TUSZ is so big that it can't happen "live" during training but needs to be implemented in a separate data processing step. Figure 3.3 shows the start of a typical recording in the TUSZ database, to numbers in the list that follows correspondents to those numbers. The image shows the problems which have to be dealt with beforehand:

- (1) This specific example contains way more channels than needed. For the ANN only the smallest common subset of channels is usable as the data applied to the network should be the same for each data point.
- (2) Additional channels such as Photic PH and EKG channels are included. These measure/or apply different biomarkers and should thus be excluded
- (3) This specific recording starts with block signals, this sometimes happens before the recording

started. It doesn't happen in each recording and if it happens the shape (It can be a line at zero or a at the max or minimum value) and length can vary. It should be filtered out before feeding it to the ANN as it is not real data and will "confuse" the ANN.

- (See table 3.3) The sampling frequency of recordings varies among recordings, neural networks are fed with a fixed amount of samples. This means that if the input of the neural network had a width of 50 samples this would correspond to 0.2 seconds if the sample rate is 250Hz and only 0.125 seconds when sampled at 400Hz. Ideally, all recordings would be in the same sample frequency, however, alternatively, all data can be sampled in the lowest common sample frequency.

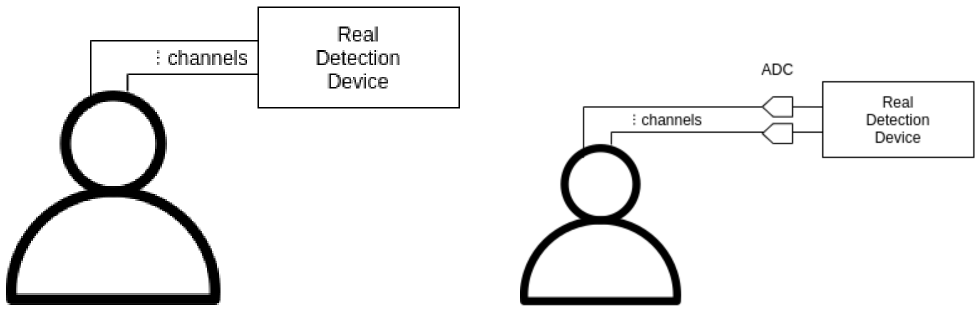
- (4 and 5) Some signals contain very profound DC signals and/or high-frequency components. As explained in section 2.2.4, there isn't too much information above 50Hz, so it's relatively easy to filter this information out.

- EEG recordings are like music, time series of bits, just as the next song on Spotify can suddenly sound much louder this problem also exists in EEG recordings on a multilevel scale.

  – (No visible) In one recording different EEG channels may differ a bit in input impedance due to how well the pad was placed on the patient's head, the effect of this can be a small (relative) difference in recorded values. This is the equivalent of a mistake between left and rights channels in the music

  – (Not visible) Even as all paths are connected perfectly similar different EEG machinery recordings are used to make recordings and for these recordings, medical personal can make tweaks in the settings of the machine. The result of this is differences in the relative "volume" between recordings.

These problems fall into two categories. The first three (1, 2, and 3 in the list) can be fixed by connecting a device in a standardized way, action is needed to pre-process the dataset correctly but it is easy to design a hypothetical design around these. The other category of problems (the rest) need real solutions in hardware but also need to be filtered from the dataset.



(a) The device has as many inputs as the number of channels it uses to detect seizures

(b) Adding an ADC to control the sampling frequency/digitization

Figure 3.4: Figures show a human connected to a hypothetical detection device, in the right picture an analog to digital converter (ADC) has been added.

3.2.1. Additional channels

The first category is problems that can be fixed relatively easily. Just connect only the necessary channels, the only inputs on the real device will be the actual channels used. Every channel should spatially be connected to the correct place on the scalp. Figure 3.4a shows such as setup. To prepare the dataset all channels were indexed, the result of this can be found in Table 3.4. From this it can be noted that 19 channels are always present, these are selected.

Channel name	Number	Channel name	Number	Channel name	Number
EEG FP1-REF	5076	EEG FP2-REF	5076	EEG F3-REF	5076
EEG F4-REF	5076	EEG C3-REF	5076	EEG C4-REF	5076
EEG P3-REF	5076	EEG P4-REF	5076	EEG O1-REF	5076
EEG O2-REF	5076	EEG F7-REF	5076	EEG F8-REF	5076
EEG T3-REF	5076	EEG T4-REF	5076	EEG T5-REF	5076
EEG T6-REF	5076	EEG FZ-REF	5076	EEG CZ-REF	5076
EEG PZ-REF	5076	EEG T1-REF	4537	EEG T2-REF	4537
IBI	4439	BURSTS	4439	SUPPR	4439
EEG EKG1-REF	4424	EEG A1-REF	4075	EEG A2-REF	4075
EEG 31-REF	2927	EEG 32-REF	2927	EEG C3P-REF	2233
EEG C4P-REF	2233	EEG SP1-REF	2172	EEG SP2-REF	2083
EMG-REF	1584	EEG 29-REF	1249	EEG 30-REF	1249
EEG ROC-REF	1058	EEG LOC-REF	1058	PHOTIC-REF	893
EEG 26-REF	795	EEG 27-REF	795	EEG 28-REF	795
EEG FP1-LE	536	EEG FP2-LE	536	EEG F3-LE	536
EEG F4-LE	536	EEG C3-LE	536	EEG C4-LE	536
EEG A1-LE	536	EEG A2-LE	536	EEG P3-LE	536
EEG P4-LE	536	EEG O1-LE	536	EEG O2-LE	536
EEG F7-LE	536	EEG F8-LE	536	EEG T3-LE	536
EEG T4-LE	536	EEG T5-LE	536	EEG T6-LE	536
EEG FZ-LE	536	EEG CZ-LE	536	EEG PZ-LE	536
EEG OZ-LE	536	EEG EKG-LE	536	EEG 30-LE	536
EEG 21-REF	527	EEG 22-REF	527	EEG 25-REF	527
EEG PG1-LE	522	EEG PG2-LE	522	PHOTIC PH	522
EEG 20-REF	522	EEG 23-REF	522	EEG 24-REF	522
EEG 28-LE	464	EEG 29-LE	464	EEG 26-LE	350
EEG 27-LE	350	EEG 31-LE	336	EEG 32-LE	336
DC1-DC	315	DC2-DC	315	DC3-DC	315
DC4-DC	315	DC5-DC	315	DC6-DC	315
DC7-DC	315	DC8-DC	315	EEG T1-LE	200
EEG T2-LE	200	EEG SP2-LE	186	EEG SP1-LE	186
EEG EKG-REF	115	EEG LUC-REF	110	EEG RLC-REF	110
EEG RESP1-REF	110	EEG RESP2-REF	110		

Table 3.4: Channels, as found in the TUSZ dataset, only channels with more than 100 occurrences, are shown.

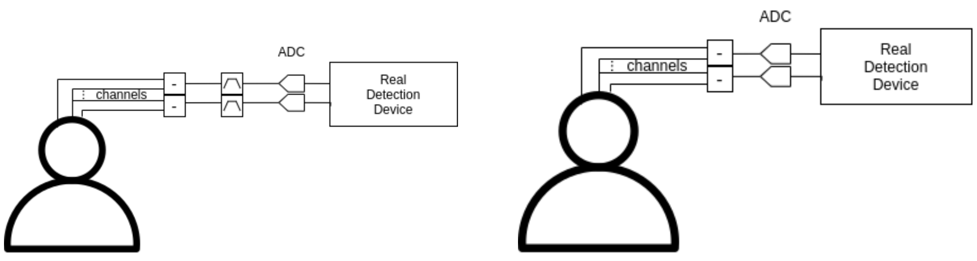
3.2.2. Sample frequency

Not all recordings are recorded with the same sampling frequency (f_s) Table 3.3 shows that all recordings are sampled at 250Hz or higher. As explained in section 2.2.4 not much information is above 40Hz. The solution is simple, downsample all recordings to the lowest common value, this is $f_s = 250Hz$. For a real device, this simply means that the input should be sampled at a set frequency, if this wouldn't be the case the prediction would simply be incorrect. Adding these results in figure 3.4b.

3.2.3. Montages

Montages were explained in section 2.2.3, the neural network will use the double banana montage. Figure 2.3b shows this standard, channels are circularly subtracted from each other, two circular loops are visible. This has an advantage: 19 channels become 18 as the three pads in the middle (Fz, Cz, and Pz) become only two channels (Fz - Cz and Pz - Oz).

To prepare the dataset for the double banana EEG, simple subtractions have to be performed as the EEG channels are just two-time series. In a real device, there are multiple ways to subtract these numbers. The raw signals from the EEG pads are typically amplified by an op-amp, this could be



(a) Adding a subtractor to create a montage (b) Adding a filter to the network

Figure 3.5: The left picture ads a subtractor to the device to create montages and possibly gain, the right picture also adds a (simple) filter in the analog domain so it can be implemented with analog components

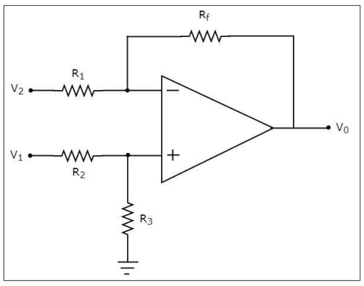


Figure 3.6: A subtractor circuit when all resistor values are the same a gain of 1 is achieved, if $R_1 = R_2$ and $R_f = R_3$ the formula works out to $V_{out} = \frac{R_3}{R_1}(V_2 - V_1)$, by playing with the resistor values the gain can be achieved.

leveraged by implementing this as a subtractor circuit such as the one shown in figure 3.6. When this is implemented, the resulting circuit looks like figure 3.5a.

3.2.4. Filter

A DC offset and high-frequency noise may be present in the EEG data. To filter this, a bandpass filter is introduced. This is done using a second-order Butterworth with $f_{low} = 0.5Hz$ and $f_{high} = 120Hz$. The upper frequency is just under the Nyquist frequency of 125Hz. A possible circuit solution is shown in figure 3.5b.

3.2.5. Normalization

Recordings can have different ”loudness” depending on how the recording was originally made. The quality of the connections and the settings of the recording device all play a role. Normalizing is hard as there is no reference point to normalize it to, the brains don’t have a ground, and using the whole body has too many electrical noises to act as a stable base point compared to the millivolts measured by the EEG. The data is not normally distributed as can be seen in figure 3.11b. Normalization can happen at multiple points. It can be performed in the time domain or in the frequency domain, which in practical terms means, before or after the STFT step.

Normalization is done in the frequency domain, which means after the STFT block. The normalization scheme used is a Min-Max feature scaling, described in equation 3.1, other methods such as standard scores or standardized momentum require estimators which are not easy to implement in hardware. An STFT is a windowed FFT, after conversion parts of the signal are in the imaginary domain, to overcome this the absolute value is calculated, which has an effect that no negative values are present in the data. This has as effect that the normalization scheme has zero as value for X_{min}.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} |_{X_{min}=0} = \frac{X}{X_{max}} \tag{3.1}$$

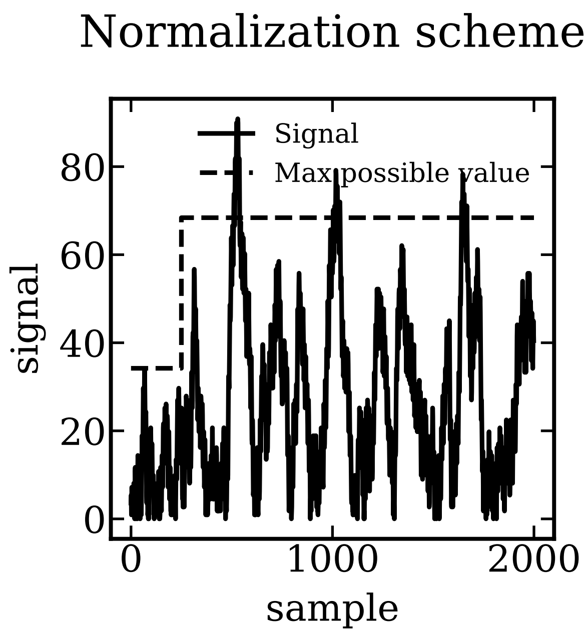


Figure 3.7: A visualization of the normalization scheme. First, the equivalent time of 250 samples are recorded, the maximum value is multiplied by a certain number (2 in this example), from that point onwards the recordings are normalized with this number as X_{max}

In the dataset is it possible to look at a recording and find the maximum number and normalize relative to this. This has two problems: first, this leads to a non-causal problem. In a live recording on a real device, it is not possible to see the EEG in the future so one can't know the maximum value of the complete recording. Secondly, as figure 3.7 shows, recordings vary in length, normalizing on the maximum value of a very short recording may not be entirely representative as not much information is included.

An engineering approach to this problem has been implemented and is schematically shown in figure 3.7. At the start-up of a device a few seconds of data are recorded, the maximum value of this is calculated and this value is multiplied by a certain factor (explained later). After initializing process, the rest of the recording is normalized with this number. As the figure shows it can happen that some samples are higher than the maximum value, the signal is then saturated. The maximum value is calculated for each channel in the recording. Alternatively, the device can be programmed to do this periodically to account for changes in brain activity over longer periods of time.

3.3. Data reduction

The amount of data used at the input of a network is proportional to the size of a neural network and in the end the power consumption. The data is first converted using the Short Term Fourier Transform (STFT), this leaves four dimensions in which the input data can be reduced: channels, time, frequency, and bitrate. First, the current data size will be explored, then the choice and the implementation of the STFT will be explained. After this, the dimensions of possible data reduction will be discussed.

3.3.1. Data format

The EDF data format is used to save the ”raw” files from the dataset. A channel consists, after resampling, of 250 samples per second. A data frame is typically between 1 and 5 seconds of data using 18 channels in the double banana montage. This means there are between 4500 and 22500 data points at the input of the neural network, the resulting network would be big. Furthermore, EDFs use 32-bit floating points numbers, as these become physical hardware, this means a lot of data lines. Besides

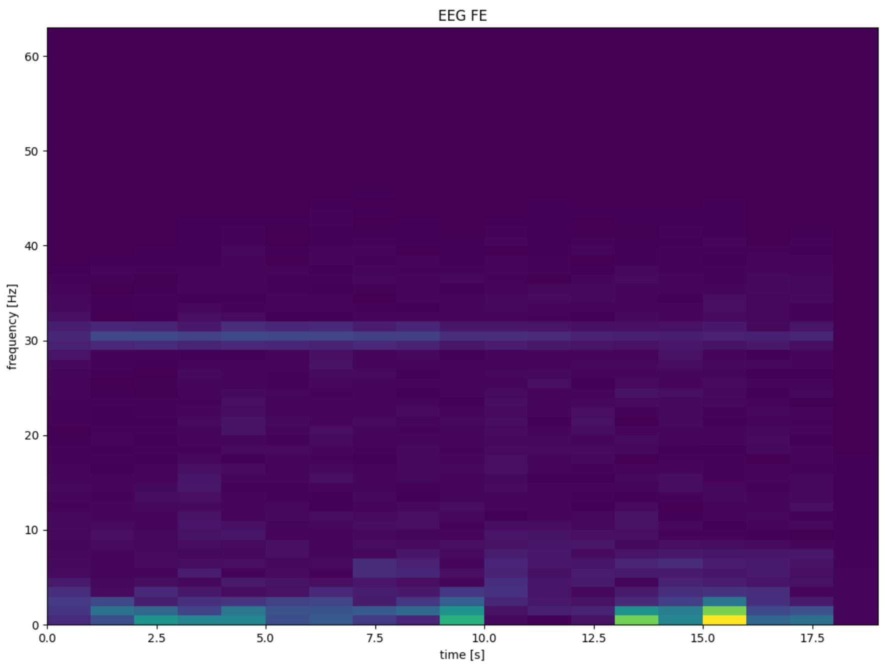


Figure 3.8: An STFT of one channel in a recording. The STFT is in essence a windowed FFT, each column represents (in this example) 1 second.

this, the logic needed for floating-point numbers is typically bigger than logic used for integer arithmetic.

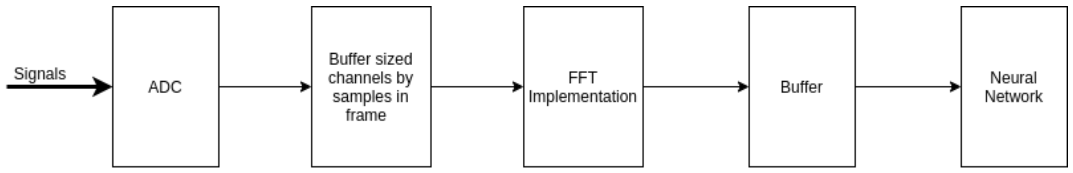


Figure 3.9: Dataflow in an STFT implementation using FFTs. Recorded signals enter from the left side, an ADC converts them to digital values. They are then buffered in memory till enough time is recorded for one STFT frame. The FFT is performed on this data chunk and the output of the STFT is once again saved in memory waiting to be used by the neural network.

3.3.2. Short Term Fourier Transform

To reduce the input size, the Short Term Fourier Transform is used. An STFT is in essence a rolling FFT. An example is shown in figure 3.8, on chunks of 1 second the FFT is performed. This creates an image with time at its X-axis and frequency at its Y-axis, EEG recordings contain 18 channels. This image-like quality makes EEGs perfect for CNNs. STFTs of 1 second are relatively ”rough”, in this specific example 1 bin in the Y-axis represents 1 Hz, this is a trade-off between more granular time bins or more frequency bins.

An STFT can be implemented in hardware in a relatively simple manner using a pre-existing FFT implementation. Figure 3.9 shows the schema of such an implementation, channels are digitized and an appropriate amount of samples are collected (example: if the period is 1 second this makes for 250 data points). The input data format will be channels by select data samples, typically this is something in the order of 18 (channels) x 100 (samples) elements. This is then ”released” through an FFT implementation. Which generates one row of the STFT as seen in Figure 3.8. The connected neural network accepts a certain input width so the resulting parts of the FFT signal are buffered again.

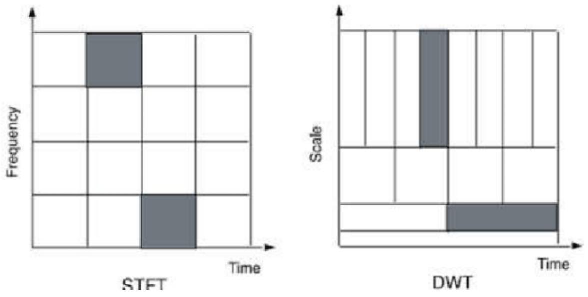


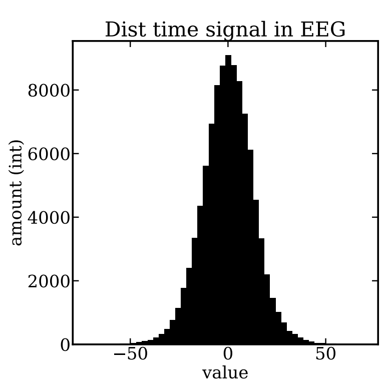
Figure 3.10: (Left) the schematic output of an STFT and (right) the schematic output of a Wavelet.

In literature, the closest neighbor to an STFT is the wavelet function. When one looks in the literature to similar work, quite a few chose the wavelet function [10]. Figure 3.10 shows schematically the difference between the STFT and a Wavelet. Where the STFT has square blocks which means that each block represents the same amount of time and frequencies. The Wavelet function makes it possible to vary this. This is handy as lower frequencies may not fit in one block. The wavelet function results in difficult hardware as FFT blocks cannot be reused and timing can become a bit opaque in the process. As [10] mentions, STFT is more applicable for real-time processing and thus the better option in this work. The sought-after signals mostly live between 1 Hz and 30 Hz, and as the chosen timeframe will be between 0.25s and 0.5s, this will not be a big problem.

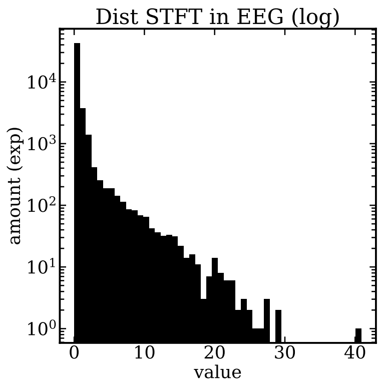
3.3.3. Reduction techniques

To reduce the footprint of the network multiple vectors can be reduced. The best way of reducing the network complexity is by reducing the size of the input. Converting the signal to an STFT doesn't directly reduce the size but gives more dimensions to reduce it. A typical example consists of 2.5 seconds of data. There are 18x625 data samples but after the STFT, this would be 18x63x10 (ch x freq x t), a little bigger as highest frequency bins contains less only 1 Hz of data. This transformation gives two distinct advantages, it generates an extra dimension that can be used to reduce the network size. Secondly, it changes it from a time series to an image which makes it suitable for CNNs which can be used to reduce the network size. It has been found that the 3 dimensions (channels, frequency, and time) have a (relative) orthogonal relationship. Thus, they can be optimized more or less separately.

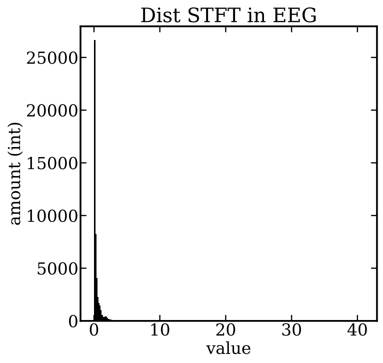
- **Time**, The more time is used for a the slower the prediction will be as predictions only happen after recording. Obviously, less data makes it harder to make a correct prediction, there are simply fewer data points to find patterns in. The length of the time blocks is set by the STFT, so the reduction is relatively easy. Fewer time bins means more data frames, half the time means double the data points. The size of the dataset doesn't decrease by using less time but the amount of datapoints to train the network on will increase and the time to prediction will decrease.
- **Frequency**, blocks are increased by applying the STFT. As discussed in chapter 2.2.4, not too much information can be expected above 40Hz. The 63 frequency bins included, each contain roughly 2 Hz of information. Lots of these can be discarded (the same could be achieved by simply re-sampling at a lower frequency). It may be possible to use even fewer frequency bins. The meta signal which this induces (the one which can be measured with EEG probes) most probably measure a relatively slow peak. Thus maybe not too much information will be lost while reducing the number of frequency bins beneath the 30Hz. The data not used are simply thrown out, thus reducing the size of the dataset.
- **Input quantization** As explained in 2.4.3, EDFs are saved as float32 numbers. In computer arithmetic, there are two main groups of mathematical instructions. The integer and floating-point operations, the first group are mainly used to represent integers while the second can represent numbers relatively precise up to $2-2^{-23}*2^{127}$ including fractional numbers. The hardware needed however is much bigger, which is no problem in modern computers. However, in the context of hardware for EEG detection, this represents a big drawback. The needed hardware will be bigger



(a) The histogram shows the distributions of numbers in one channel of a typical EEG recording



(b) The histogram shows the distributions of numbers in one channel of a typical EEG recording after an STFT has been performed in log format



(c) The histogram shows the distributions of numbers in one channel of a typical EEG recording after an STFT has been performed.

Figure 3.11: Distribution of typical data for a 295-second recording of patient 5943 (session 4, recording 1). (a) shows the time data is more or less normally distributed, (b) the distribution of numbers after the STFT is performed in log scale, as (c) shows why this is shown in log scale.

because as integer logic directly represents numbers while floating numbers consist of a fraction and exponent (and sign value) to represent a number. Hence the latter approximation uses more logic. Secondly, the standard width of a float is 32bit which is a lot. Reducing this size by two would already result in half as many wires. Data of saved in the EDF files are typically capped at 10,000 different values. The data distribution of a normal signal is shown in figure 3.11a and the STFT variant in figure 3.11b, it can be seen that most data points fall in the lower parts of the distributions. The higher values are either because of very high brain activity or clipping signals.

- **Channels** The double banana method as explained in chapter 2.2.3 is primarily used for human interpretations by neurologists. However computer logic of computers may be different. There are a few different ways to walk this road, one can look at the spatial distribution of EEG channels or look at the correlation between channels in historical recordings. The method is chosen in this work select a few logical combinations in the EEG path as shown in Figure 2.3b.

3.4. Conclusions

This chapter has dealt with selecting a dataset and with creating a method to effectively pre-process this. The steps have been combined in a Python script that processes all data and outputs a single file that contains all processed data. The data is saved as HDF (Hierarchical Data Format) file, this is a standard way to save multidimensional arrays of scientific data. Algorithm 1 shows the steps, roughly speaking four things happen. 1) All files are indexed, 2) Per session these files are pre-processed by using a bandpass filter, resampling, and creating a montage, 3) the first few of the first recordings are just to find values for normalization, and 4) Data is saved as int8 values (only 1 byte) and chunks of 10 seconds of data.

Algorithm 1: The pre-process script starts by indexing all data. This is then processed per session, the first recording 2 seconds of a recording are used to calculate normalization

```
Result: Data pre-process steps
Index all recordings in dataset and group them per session;
for all sessions do
    select first 2s of first recording of specific patient session;
    clean data by sampling at 250Hz, filter with bandpass, select correct channels and create
      montage(); calculate STFT of this first 2s and keep the highest values per channel;
    for recordings in patient session do
        load recording and index seizures();
        pre-process file with bandpass filter and creating the montage();
        calculate the STFT();
        normalize using kept normalization values for each channel, result will be between 0
          and 1();
        quantize these values to 256 values (uint8);
        save per patient-session in a ”nrm” and ”szr” folder, the files are split in chunks of 10
          seconds
    end
end
```

The network contains after processing **328314** chunks of data which are all 10 seconds long. Of those **306761** chunks are background EEG recordings and only **21553** or 6.56% are seizure recordings (all types), this percentage seizures is similar to the original dataset.

4 Artificial Neural Network

This chapter will describe the design of the neural network, specific challenges, and optimizations. ANNs are hard to design in a (traditional) mathematical fashion, and they are typically compared based upon results. There is no one formula to optimize to get the most optimum results. Finding the optimum design often means trial and error, benchmarking the result while tweaking a parameter, and making a design decision of which point to select. This also means that it is often not clear if a parameter is orthogonal (possible to design this parameter without affecting the ideal setting of other parameters) to the rest of the design. However, during the design of such a system, one often gets a feeling for the orthogonality of the design.

This chapter tries to give insights into how the ”best” network(s) is/are selected. These decisions begin by selecting a design or set of designs. The architecture and design of the training architecture are explained, special attention is given to oddities such as sampling schemes and ”stop ahead of time”-functions. Both the design and input space will be reduced to reach a minimal viable design. This often involves testing different design parameters such as the used time or frequency bins to find the ”off the cliff” point. This is the point from where accuracy starts to decline. The chapter will end with selecting a few possible trained network options to be used in the next chapter where this will be transformed into an FPGA solution.

4.1. Design

This section will first discuss the steps training program, after this, the training algorithm followed by the design of the ANN itself. The aim of this chapter is not to explain the code on a line-by-line basis but merely to explain the most important concepts. The code itself is available in a Github project.

4.1.1. Main functionality

The main loop contains all functionality to train a neural network. This network is trained offline thus this can happen on a computer with enough resources. The main loop’s most important function is to transfer the pre-processed data to a trained ANN. Other functions include handling the post-process phase, where the performance is checked, and initializing the data loaders. The algorithm 2 shows very roughly the outline of the main loop.

The mentioned main algorithm needs a few arguments to run. These are the **files location**, the location where the pre-processed data (typically a few GBs) are stored, **location of meta data**, a CSV file which corresponds to the data blocks in the files and contains information such as recording location, and to which recording they correspond. The **Network structure** is defined, this gives information about the specific sizes of the layers, the **out folder** defines the location where the results are saved. Also

Algorithm 2: The main loop of the training algorithm, only showing very roughly the functions. First, the data is loaded and split. The training data is oversampled to create a 50:50 ratio of background and seizure data. This is all used to train the network.

```
Result: A trained network
initialization phase;
index all files in given dataset;
shuffle the index;
split index in 80% validation data and 20% training data;
Create training data loader which over samples seizure data to a 50/50 ratio;
Create validation data loader with ratio of the dataset;
Initialize neural network;
Train neural network;
Save neural network;
```

the **learning rate** (a measure of how ”fast” the network will learn), the **batch size** and the **epochs** are defined here.

When benchmarking results, multiple runs of this algorithm are performed. This accuracy will vary a little bit because the accuracy of the ANN is stochastic as it is the function of random initial values and randomly shuffled training data. Multiple runs, typically between 3-10, will be used to calculate a mean accuracy and a confidence interval around it.

Dataloader

The first five steps of the main loop (see algorithm 2) are the functionalities of the data loader. Before training an ANN, a data loader is needed, a data loader is a class that handles the loading and indexing of the data elements. When the training loops need new data, the data loader can provide it with more data batches. As the amount of data is very big it doesn’t fit in memory (64GB would be required). Thus, a custom memory handler is implemented in the data loader. This custom data loader can keep a maximum amount of data elements in its memory, when this limit is reached it will automatically delete the oldest elements. The data loader also matches the data with the metadata, this is done so afterward, it’s possible to see if certain types of data are harder to classify. The data load process has two main steps: indexing and loading. Initializing means that both the meta CSV file and datafile are opened.

In the metafile information about the length and amount of data is saved. Chunks of data have a standard length of 10 seconds or 20 datapoints (each data point corresponds to 0.5 seconds of data). A patient typically contains multiple of these chunks which are categorized as either seizure or background. Each data chunk can be identified using the file path (ie patient-session-type-number => 258-2-szr-4). The input of the ANN varies, it can use 5 seconds of data or 4 depending on the configuration. The data loader can mix and match the data chunks to accommodate these needs, for very long recordings (>10 seconds) it can combine chunks from the same patient session, and for shorter, it can create multiple data points out of one chunk (ie 10 seconds can be four 2.5 second points). This index object is already 300MB in memory. The elements in this list correspond to the data elements used to train the network but don’t include the data.

After the data loader is initialized, the list is shuffled to randomize the elements and then split the dataset into 80% training data and 20% validation data. The top 80% of the randomized list will become the training set and the bottom 20% the validation set. The training data will be used to optimize the network and the validation will be used to make sure the network actually trains to detects patterns and not just the training data. The validation part is not used to train the network it can be used to verify the network trains. Sometimes the accuracy of the training data gets higher than the corresponding validation data, this is called overfitting and this may mean that too much training is done. Reducing the number of epochs or increasing the batch sizes may help to fix this problem.

The data needs to be balanced as the vast amount of data points in the epilepsy dataset are from background recordings. In the original dataset around 94% of the data is background EEG. Thus classifying all elements as background would achieve a 95% correctness. However, this is obviously not an acceptable result. Two approaches exist to tackle this problem, oversampling and undersampling. In under-sampling, fewer samples are taken from the majority group (the background signals) to match the minority group (the seizure signals). As the minority group is only 6% of the total dataset this would result in only using 12% of the source data. Oversampling is when the minority group signals are used multiple times to match the majority group, in the 94:6 ratio this means that for a 50:50 ratio minority data is used over 15 times. An alternative to this approach would be two create more source data by, for instance, adding versions with noise and use windowing. The training data is over-sampled to achieve a 50:50 ratio while the validation data is not as we are ultimately interested in the accuracy in a normal dataset.

Training the network

After the data loader is initialized, the ANN layout will be used to scale the template ANN. The design of the template ANN will be discussed in Subsection 4.1.1. This process mostly takes place behind the scenes in PyTorch. A network object is created, which contains the layers as specified by the template and fills it with (random) numbers. This results in a network object which can be used to train.

Now the real training phase is beginning. Algorithm 3 shows the training loop. It needs a few arguments to work, obviously the network itself and the two data loaders (the training and validation loader). But also an optimizer (strategy to optimize the network) and a loss function (how much the network should be punished for a wrong result) and the epochs (how many times all training elements will be used upon the network).

Algorithm 3: The train loop roughly explains the steps in the training process of a neural network

```
Result: Train loop
transfer network to GPU ;
for epochs do
    for batch in train loader do
        load batch and labels;
        output network = network(batch);
        loss = loss function(output, labels);
        loss.backward();
        optimize.step();
        keep generated output and associated labels;
    end
    profile train output(output, labels);
    for batch in valid loader do
        output network = network(batch);
        keep generated out and associated label;
    end
    profile validation output(output, labels);
end
```

First, the network is transferred to the GPU if available. Training times are much lower on a GPU as machine learning is heavily parallelizable. A for loop will loop over all data as many times as specified by the epoch numbers (an integer), typically this is 25 times. After this, the real training process starts by loading a batch (64 elements) of training data with corresponding labels. The data is first loaded from the hard disk to the memory and then transferred to the GPU memory for batch training. The training data is then put through the network in training mode and after the loss function compares the generated network predictions with the labels. The function used here is a Binary Cross-Entropy loss (BCEloss), it first puts the output through a sigmoid function which bounds the prediction between 0 and 1 and compares the outputs of the batch with the labels, which are either 0 or 1. The more these numbers are off the more the network should be punished, this number is averaged over the batch.

This function is also known as the cost function as the cost is higher as more results are wrong. This loss value is then backwardly calculated for all the values in the neural network, this process calculates the dloss/dx for each node. This is called backpropagation and it is the basis of how neural networks learn. The optimizer is then used to update the values in all those nodes, how aggressive this is done depends on the setting of the learning rate. This training is easily the most computationally intensive part of the whole algorithm and cannot be easily implemented in an efficient fashion.

The generated output and labels are kept and after all, batches are processed to profile the outputs. The confusion matrix (true positive, true negative, and the likes) is calculated, and sensitivity, correctness, specificity, and precision are based on these numbers. This is however based on the training data, which means that is not a real measure of the quality of the network. This has two reasons 1) an over-sampler is used, so samples can end up multiple times in these measures, and 2) a network cannot be trained and validated with the same data as it could be trained to detect just the training elements and not the general patterns. It is however still important to calculate the metrics of the training data as they give an insight into how far the training process is.

After the training loop, a validation loop is implemented, this loop is used to keep track of the quality of the network. Just like the training loop, the output is generated, however, the network is not optimized based upon those results. The labels together with the predictions (outputs) are used to calculate the confusion matrix and the measures of quality. These measures of quality can then be used to calculate the MCC and F1 scores, these are the is scores that try to measure the quality in one number.

Most neural networks are run to the pre-defined amount of epochs and then saved. However, due to the applied oversampling resulting from the 95:5 ratio of seizures and background signals and overfitting, the output tends to vary quite a bit. For this reason, the network with the highest MCC score is saved. This ensures that the best network is selected.

Neural network

The design which is chosen needs to be adaptable for lots of different situations and input sizes. The chosen architecture can be seen in figure 4.1. This is a very typical structure, it consists of two parts, a 2D Convolutional part followed by a linear part. As can be seen in the figure the convolutional part has an input layer that matches the size of the input, next are similar convolutional layers, these can happen multiple times, but in practice, this will never be more than three. As it has been explained in section 2.3, a convolutional layer has the kernel size and the output channels. Every CNN layer is followed by a 2D Batch Normalization layer but this isn't shown, the size of this is the same as the number of output channels of the CNN layer.

It is typical for these kinds of networks to include max pool layers, however, the framework which translates this hardware has a very buggy MaxPool implementation. Because of this MaxPool layers are not used in the design. The network is quantized using the Brevitas package, this means the network is quantized which in a later phase can be used to minimize the network size.

Algorithm 4: The neural network from 4.1 in words

```
Result: Boilerplate neural network design
for cnn-layers do
    2D-CNN(kernel, amount of input channels, amount of output channels);
    BatchNorm2D(amount of output channels);
end
flatten(size network) dropout(0.5) -> only used in trainings mode;
out-size = linear(size of flattened network);
batchNorm1D(out-size);
 dropout(0.5) -> only used in trainings mode;
end size (1) = linear(out size);
```

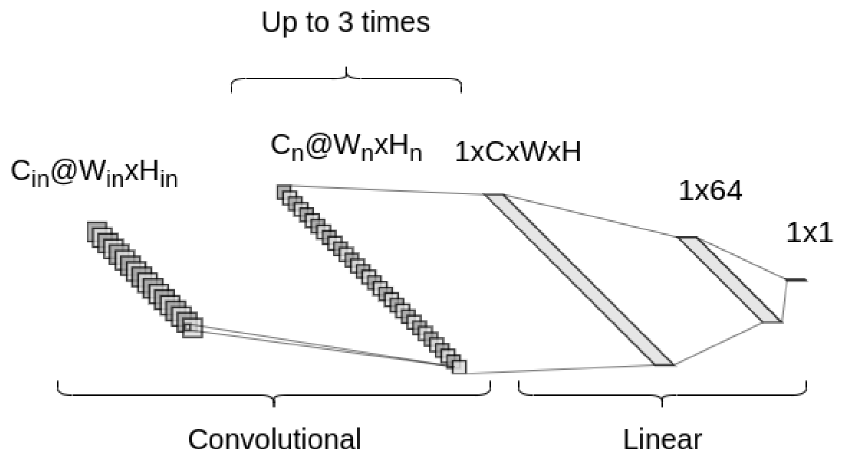


Figure 4.1: Boilerplate Neural Network Design. The input layer is followed by up to three convolutional layers, the signal is then flattened and two linear layers follow. The convolutional layers are followed by a 2DBatchNorm for normalization. The first linear layer is followed by a 1DBatchNorm. After the last linear layer, where size is reduced to 1, a sigmoid is applied to force the number between 0 and 1, this number is rounded.

Layer	In Size	Out	Kernel	Bias
CNN_{in} + 2D batch norm	18@10x10	32@6x6	5	No
CNN_1 + 2D batch norm	32@6x6	64@3x3	4	No
CNN_2 + 2D batch norm	64@3x3	128@1x1	3	No
$Linear_1$ + 1D batch norm	128	128		No
$Linear_{out}$ + sigmoid and rounding	128	1		No

Table 4.1: The sizes of the reference network can be projected in figure 4.1, this network has three convolutional layers followed by two linear layers. The input is a time by frequency of 10 by 10. Brevitas is used but in a 16-bit configuration. The resulting network can be converted to an FPGA implementation.

4.2. Network optimization

This section will explain the decisions made to optimize the network. A lot of tests will be performed by training the network in slightly different ways to find out which works best. Based on this information two designs will be chosen to be converted to hardware implementation on an FPGA in chapter 5, one for the most accurate results and one optimized for size. In this section multiple experiments will be performed, all experiments follow the same structure: 1) First the experiment is explained, 2) the results are shown, and 3) the results are interpreted.

4.2.1. Reference circuit

This chapter is about finding the best circuit given the circumstances. Normally the goal is quite simply to find the best performing circuit, however in this work, the goal is quite complicated; size, accuracy, input width, and other boundary conditions all play a role. Thus, comparisons are often done to a reference circuit. This is a circuit that was developed during countless hours of experimentation and it has shown to give a good performance. This circuit is used to check if changes made to the neural network result in improvements to the accuracy, or as a base to start making changes in an experiment where different normalization schemes are tested. The description of this reference neural network is in table 4.1. The accuracy of this reference network is shown in table 4.2. The network consists of three convolutional layers and two linear layers.

This network will be the basis, based on the needs of the specific experiment, differences will be mentioned which are relative to this reference. When fewer convolutional layers are used, the highest kernels sizes are used. Three layers mean the kernel sizes are 5, 4, and 3, two use 5 and 4, and when only one layer is used this is 5.

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
Reference	95.97 (0.48)	77.68 (4.56)	97.25 (0.82)	66.58 (4.96)	0.70 (0)	0.72 (0.01)

Table 4.2: The result of the reference network, between parenthesis is the 95% confidence interval. The MCC-score is on a scale of -1 to 1 and the F1-score 0 to 1

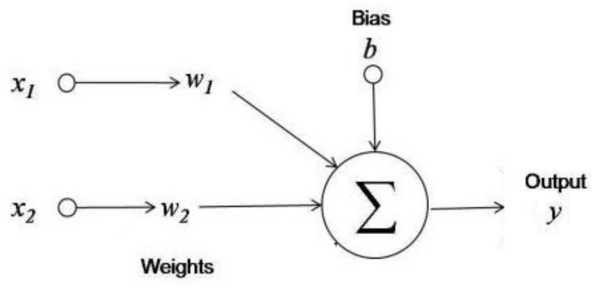


Figure 4.2: In this simple neuron, the inputs are multiplied by the weight and then summed by the neuron. When a bias is enabled the neuron has an extra trainable element; the bias.

4.2.2. Biased network

An ANN consists of trainable elements, these are called neurons. Neurons are elements with are optimized to make predictions network given an output. During training, the values inside neurons are constantly updated to better represent the patterns in the source data. Figure 4.2 shows such a neuron, many thousands of them are in typical networks. The inputs of the neuron are multiplied by the weights and then added together. An extra trainable variable called the bias can be added. This is added to the ”result” of the neuron as a single number. Each neuron can have a bias, the reference network has a little over two thousand possible biases (summing the layers: 32*6*6 + 64*3*3 + 128*1*1 + 128 + 128 + 1), accounting for roughly 2% of the total number of trainable variables.

Experiment The reference network has no biases applied. To check if this is the correct design decision, a network with bias enabled has been trained.

Results and interpretation The result of this experiment is shown in table 4.3. Multiple runs have been performed, the results are each time a little different as training an ANN is a stochastic process. Compared to the reference network from table 4.2 not much has changed, all values are within the confidence interval. The probable reason for this is that missing 2% of trainable parameters isn't much, accuracy tends to fall off a cliff on a certain point, before this point adding or deleting trainable parameters has relatively little effect.

4.2.3. Reducing the network

Reducing the network size is one of the most efficient ways of reducing the size of the hardware implementation of the ANN. A smaller size leads to less trainable parameters, which leads to fewer calculations and finally to a lower power consumption per prediction. However, fewer parameters also mean fewer parameters to train, so a very rough link between trainable parameters and accuracy should be expected.

Experiment Both the convolutional and linear layers of the network are resized in order to assess the effect on the accuracy. This creates networks in a wide range of trainable parameters. The structure of the network is changed relative to the reference circuit described in table 4.2. The first two columns of table 4.4 shows the parameters of the resized networks. The CNN column shows the number of channels at the output of each convolutional layer, the input of the first convolutional always 18 (channels) with a 10x10 input frame. The kernel sizes for three layers convolutional are 5, 4, and 3, for a two-layer setup 5 and 4, and if only one layer is used: 5. There are always two linear layers, the last

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
With bias	95.97 (0.58)	76.34 (6.16)	97.35 (1.04)	67.29 (7.26)	0.69 (0.02)	0.71 (0.02)

Table 4.3: Result of reference with bias enabled, between parenthesis is the 95% confidence interval

CNN C1/C2/C3	NN L1/L2	Elements N	Correctness %	Sensitivity %	Specificity %	Precision %	MCC
64/128/256	256/128	488834	96.66	75.83	98.12	74.08	0.73
32/64/128	128/64	129730	95.67	79.24	96.82	63.81	0.69
16/32/64	64/32	36194	94.89	72.15	96.47	58.74	0.62
32/64	576/64	84418	95.66	75.31	97.10	65.34	0.68
16/32	288/64	34114	94.02	72.83	95.50	53.71	0.59
16/32	288/32	24802	93.32	76.76	94.48	49.39	0.58
64	2304/64	176578	95.56	71.39	97.25	64.55	0.65
32	1152/64	88386	94.07	69.24	95.81	53.82	0.58
16	576/64	44290	92.09	70.92	93.58	45.81	0.53
16	576/32	25762	92.11	73.08	93.44	44.91	0.53
16	576/16	16498	91.67	73.62	92.94	42.40	0.52
8	288/16	8274	90.88	69.33	92.38	38.89	0.48
4	144/16	4162	88.91	70.35	90.21	33.47	0.43

Table 4.4: The first (CNN) and second (NN) columns set the layout of the neural network. The CNN number corresponds to the amount of channels at the output of a layer. The input size of the network is always 10x10 (frequency bins by time bins). The kernel sizes for three-layer convolutional networks are 5, 4, and 3, for two layers 5 and 4 and if only one layer is the kernel is 5. There are always two linear layers, the last layer of the CNN part is flattened and this number is the input of the first linear layer (L1). The output of this layer is the L2 value, the second layer has L2 as input and has 1 value at the output. The elements columns show how many trainable elements are in the specific configuration of the network, this is regardless if these elements are in the convolutional or linear part of the network. All networks have been trained multiple times, the quoted numbers are the mean value.

convolutional layer flattened (i.e. 128x2x2 means L1 is 512), this number is the input of the first linear layer (L1), this value thus cannot be freely chosen: it depends on the size of the last convolutional layer. The output of this first linear layer is the L2 value, the second layer has L2 as input, the output of the second linear layer is one number.

Results and interpretation The result is in table 4.4, and plotted in figure 4.3. Please note the plots are in a log scale, as the range of trainable parameters is roughly 100 between the smallest and the biggest size. On a linear scale, the scores and correctness plots rise fast and then stabilizes. A few observations can be made:

- There is indeed a rough link between the trainable parameters and quality. The scores and the correctness plots show a line that increases roughly linear (in a log plot) to increased network size.
- There is a very distinct tooth saw a pattern in the results, the reason for this is the ratio of trainable parameters between the convolutional and linear parts. Looking at the score plot the 'vertical' sides of the tooth saw(s) the higher scores have a bigger linear part compared to the lower score on the vertical side. The conclusion is that parameters from the linear layers improve the accuracy more.
- The biggest network, which contains almost 500,000 parameters has the highest score. However, when so many parameters are available the network can "hardcode" data. Around 40,000 seizures recordings are in the dataset, the most difficult of them can be recognized and hardcoded in the network. This is not real pattern recognition and is thus bad. A hint that this phenomenon is happening is in the train v validation plot, train correctness is higher than the validation correctness for very big networks. Typically the correctness of the validation is higher than the training

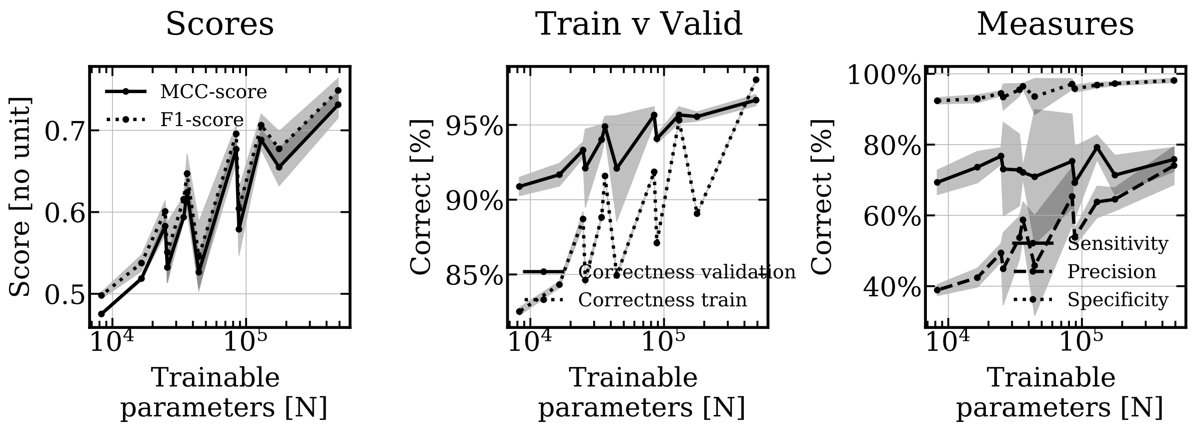


Figure 4.3: The images are all relative to the network size expressed in trainable parameters. (Left) shows to scores, which try to catch the quality of networks in one number. The middle shows the correctness of the training set vs the validation set. The right one shows measures, which were explained in section 2.4.2. The X-axis of these graphs has been changed to a log scale for readability. The spread in trainable numbers is between 4162 and 488834: spanning a range of 100. On a linear scale, the scores and correctness rise very fast and then stabilize at a high level. The plots are created from the results in table 4.4, gray areas show the 95% confidence interval which is generated by training the network multiple times, the dark line is the mean of these runs.

(see plots of other experiments). A conclusion of this is that networks that are "too big" discourage learning patterns. It should be noted that the network couldn't be enlarged even more as there simply wasn't enough GPU memory for even bigger networks.

- Due to the log scale the improvements in score seems a bit more linear than it really is, somewhere between 100,000 and 150,000 elements seems to be the optimum.

Bins (freq)	CNN C1/C2/C3	NN L1/L2	Elements N	Correctness %	Sensitivity %	Specificity %	Precision %	MCC
63 (all)	16/32/64	3456/32	144738	97.03	82.97	98.03	74.99	0.77
30	16/32/64	1344/64	120258	95.09	78.80	96.23	59.40	0.66
20	32/64	2496/32	127330	95.95	78.56	97.17	66.41	0.70
15	32/64	768/32	146018	95.85	84.29	96.66	64.11	0.71
10 (ref)	32/64/128	128/64	129730	95.98	76.72	97.32	66.83	0.69
8	32	768/128	113154	93.35	75.15	94.62	49.49	0.58
6	64	768/128	127618	93.32	77.89	94.41	49.54	0.59
4	64@3	3456/32	141954	91.54	75.82	92.64	41.86	0.52
3	128@2	3456/32	152450	89.34	78.28	90.11	35.55	0.48

Table 4.5: The second (CNN) and third (NN) columns set the layout of the neural network. This works the same as in table 4.4, however as input sizes become small the kernel sizes can be too big for the input (for instance 3 frequency bins don't work with a kernel size of 5 as the kernel is bigger than the input) the kernel size is then reduced, this is signaled by the @ symbol in the CNN column. The width and height of the input are not square in this experiment they range from 10 by 63 to 10 by 3.

4.2.4. Frequency

The fewer frequency bins are used, the smaller the input space is. The signals at the output of the STFT contain 63 frequency bins, up to 125 Hz of data, the last bin contains 1 Hz and all others 2 Hz. It is assumed that most information is in the lower bins as explained in section 2.2. This is why the reference network only uses the lowest 10 bins representing 0-20 Hz. The time signals are filtered with a bandpass filter but some DC signals may be left in the lowest frequency bin.

Experiment Two experiments have been performed: a check for residual DC signals and an experiment with the range of frequencies applied to the network.

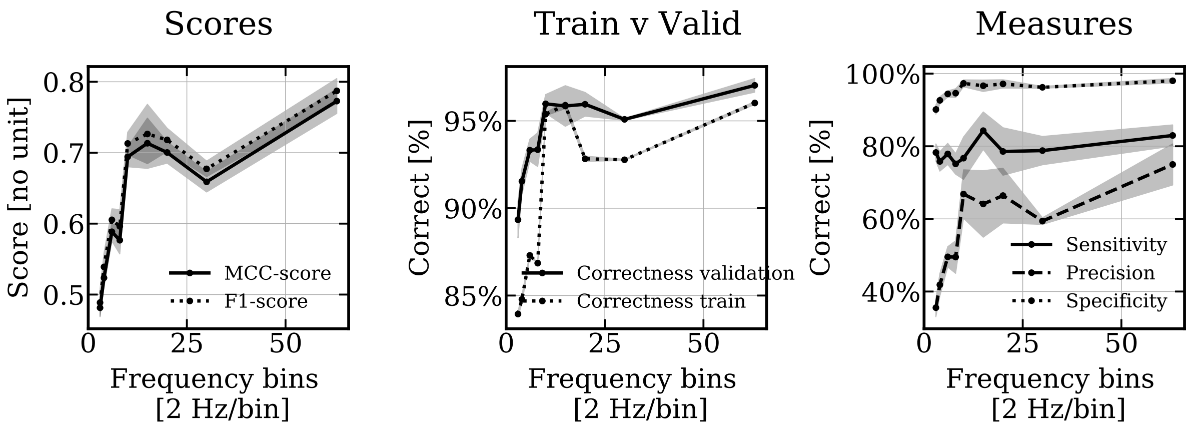


Figure 4.4: Plotted results of table 4.5, gray areas show the 95% confidence interval. The plots show the same measures as in the network sizes plots 4.3. The x-axis shows the amount frequency bins used in the prediction. The STFT gives 63 such bins corresponding to the 125 Hz of data, the highest bin (number 63) contains 1 Hz, all other bins 2 Hz. It is assumed that the higher frequencies contain less information compared to the lower ones.

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
No DC	91.55 (0.8)	83.22 (2.20)	92.14 (1.02)	42.83 (3.12)	0.56 (0.02)	0.57 (0.02)

Table 4.6: Result using no DC component by applying frequency bin 2 to 11. Between the parenthesis is the 95% confidence interval. Observed MCC/F1 scores are much lower than the scores of the reference network.

Residual DC signal: The reference network as described in table 4.2 is used, however not the lowest 10 frequency bins are used but the numbers 2 to 11. The lowest bin containing the frequency range 0-2 Hz has been excluded. With this experiment, it can be seen if the residual DC signal has an effect on the prediction accuracy. Lower accuracy is expected as it is assumed that quite a lot of useful information is present in this lowest frequency bin.

Modifying the applied frequencies: The applied frequency bins are varied, thus a smaller or a bigger proportion of the frequency range is used to learn. As the input size is modified the size of the network will also change and with this the trainable elements in the network. This has an effect on the accuracy, networks are resized to have roughly the same number of trainable elements as the reference network. To aim is to stay within 10% (105,000 - 155,000) of the reference network.

Results and interpretation The result of the DC experiment is shown in table 4.6. The result of the different frequencies ranges experiment is in table 4.5, and plotted in figure 4.4. Some interpretations:

- The no DC experiment shows worse results. If there would be disturbance due to DC disturbances a higher accuracy would be expected. Based on this experiment it cannot be concluded that no DC disturbance is happening, this experiment is not sufficient to prove this. It can be concluded that there is useful information is in the lowest frequency bin.

- The scores and correctness rise fast till 10 frequency bins (20 Hz of data). After this, the accuracy stabilizes signaling that most information is indeed in the lower bins.

- The drop at 20 bins can be explained by the same phenomenon as the tooth saws of plot 4.3. Although care has been given to keep the ratio between convolutional bins and linear bins relatively stable, the boundary condition of keeping network sizes roughly equal with different input sizes makes this hard. Luckily, the ”tooth saw” drop is much less extreme, this is because the ratio varies much less wildly.

Bins time	CNN C1/C2/C3	NN L1/L2	Elements Elements	Correctness %	Sensitivity %	Specificity %	Precision %	MCC
20	32/64	2496/32	127330	96.29	80.41	97.40	68.46	0.72
15	32/64	768/32	146018	96.87	85.89	97.64	72.41	0.77
10	32/64/128	128/64	129730	96.22	73.32	97.82	70.31	0.70
8	32	768/128	113154	92.97	77.08	94.07	47.61	0.57
6	64	768/128	127618	93.04	76.00	94.24	48.41	0.57
4	64@3	3456/32	141954	92.21	69.86	93.77	45.36	0.52

Table 4.7: The second (CNN) and third (NN) columns set the layout of the neural network. This works the same as in table 4.4, however as input sizes become small the kernel sizes can to big for the input (for instance 3 frequency bins don't work with a kernel size of 5 as the kernel is bigger than the input), the kernel size is then reduced, this is signaled by the @ symbol in the CNN column. The width and height of the input is not square in this experiment, they range from 2 by 10 to 20 by 10.

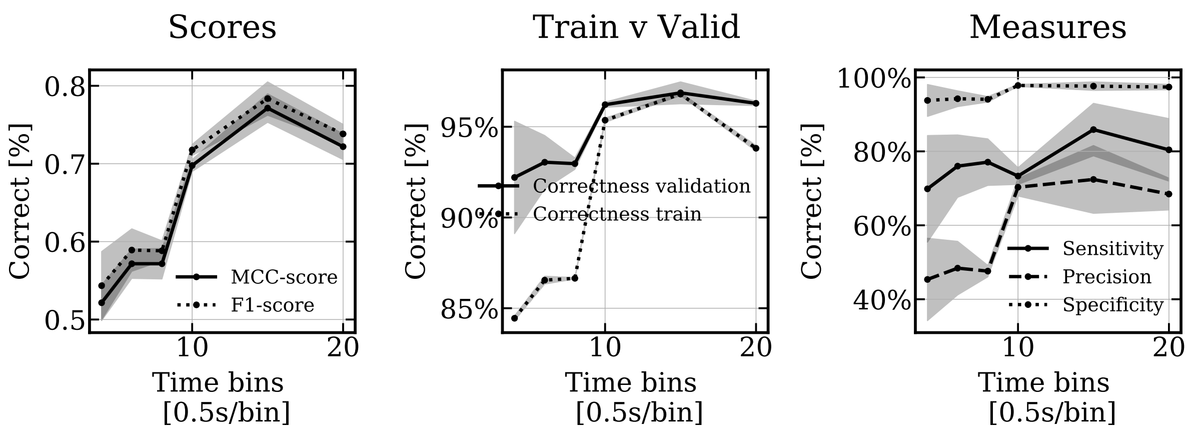


Figure 4.5: Plotted results of table 4.7, gray areas show the 95% confidence interval. The plots show the same measures as in the network sizes plots 4.3. The x-axis shows the number of times bins used in the prediction. The STFT gives 2 times bins per second of data. The lowest time bin contains 2 seconds the highest 10 seconds.

4.2.5. Time

The reference network uses 10 times bins or 5 seconds as each bin represents 0.5 seconds of data. Reducing this has two advantages. First, the input width of the network is reduced, secondly, the time to detection is reduced. However, applying less time means that the network has to make predictions based on fewer data points.

Experiment Like changing the number of frequency bins in section 4.2.4 changing the time bins results in changing the network size and the number of trainable elements. This time the amount of frequency bins is set at 10 and the time bins are varied, the range is 4 by 10 to 20 by 10, or 2 seconds to 10 seconds of data. The network layouts which are used are the same as the ones used for the frequency. Like the frequency bin chapter this will be kept in between +/- 10% of the reference network.

Results and interpretation The result of the time experiment are in table 4.7, and plotted in figure 4.5. Some interpretations:

- The general shape of the graphs looks a lot like the ones from the frequency experiment. Compare to the first 20 frequency bins in the score plot in figure 4.4, the shape is almost equal. The same ”dip” in performance can be observed for 15-time bins which are probably due to a non-optimal ratio in trainable variables between linear and convolutional layers.
- The optimum seems to be reached around 10-time bins (or 5 seconds), this is great as it means that the optimum is at a square input size (10 by 10).

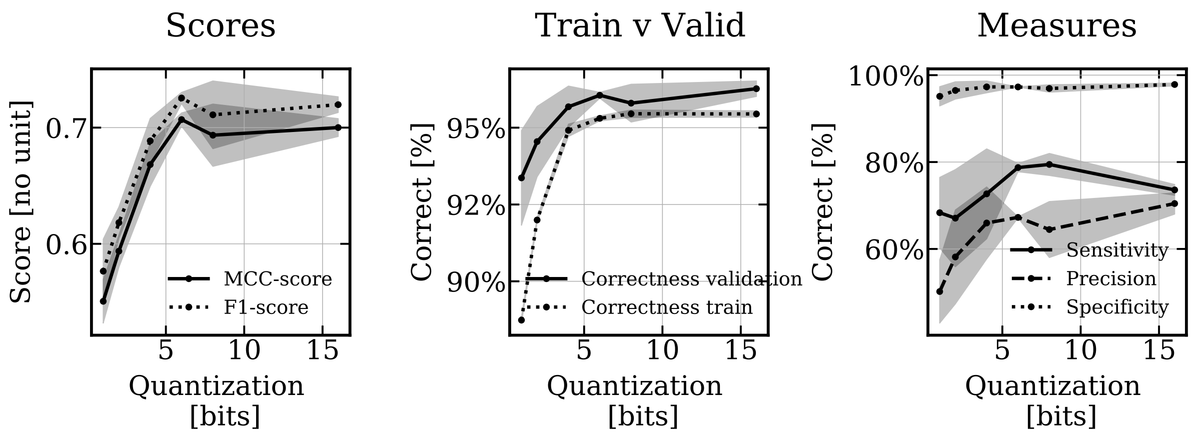


Figure 4.6: Plotted result of table 4.8. The amount of quantization bits determines how many values the weights in the neurons can have. A one bit QNN can represent only 0 and 1, a 2 bit version can only represent 4 values. Gray areas show the 95% confidence interval over multiple runs

Quantization N of bits	Correctness %	Sensitivity %	Specificity %	Precision %	MCC
1	93.37	68.35	95.13	50.20	0.55
2	94.55	67.09	96.48	58.16	0.59
4	95.68	72.68	97.29	65.97	0.67
6	96.06	78.73	97.28	67.27	0.71
8	95.80	79.45	96.94	64.47	0.69
16	96.27	73.59	97.85	70.45	0.70
32	95.86	76.11	97.24	66.09	0.69

Table 4.8: Quantized networks between 1 and 32 bits have been tested based on the reference network. All layers in the network are quantized using the same quantization factor.

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
No quantization	95.90 (0.24)	78.26 (5.66)	97.15 (0.44)	66.14 (2.12)	0.70 (0.01)	0.72 (0.02)

Table 4.9: The reference network is quantized, the result in this table show a version where the network was quantized. The values are similar to the reference proving that quantizing is possible for this dataset. Between parenthesis is the 95% confidence interval.

4.2.6. Network quantization

A standard neural network trained by PyTorch uses floating points numbers as for the weights for the neurons. This means that learnable parameters can have a very wide range, but this may not be needed to effectively detect patterns in the input data. Each (half)-float takes 4 bytes in the memory, a method to reduce this is by quantizing the weights in the network. This reduces the number of values a neuron can represent but also reduces the memory needed to represent the network. When a neuron is quantized to one bit each neuron can only represent 0 or 1, with 2 bits only 4 values are possible. To achieve this Brevitas was implemented, Brevitas is a package that works as an extra layer on PyTorch, can quantize neural networks, transforming the network into a quantized neural network (QNN).

Experiments Two experiments are performed to check if quantization is possible with epilepsy data and the effect of different quantization levels.

Does quantization work: First, it should be proven that the neural network can actually learn patterns with a quantized network. The number of possible numbers is greatly reduced and it may not be possible to internalize difficult patterns once quantized. The reference network is already quantized, so to check, a version of the network without quantization has been trained. As the neurons can represent numbers more accuracy would be expected.

Effect of quantization values on the accuracy: The most extreme version binarizes the network (also known as BNNs) as neurons can only represent 0 or 1. While this is the most memory-efficient it may not yield good results. An experiment has been performed whereby the amount of bins is steadily increased to find the point where more quantization bits don't add any extra accuracy. It is possible to quantize each layer in the ANN with a different quantization level, in this experiment all layers are quantized using the same number of bits.

Results and interpretation Table 4.9 shows the performance of a network with no quantization. Table 4.8 shows what happens for different quantization factors these are plotted in figure 4.6. Some interpretations:

- The not quantized network performs almost exactly like the quantized version. This signals that quantization can effectively be used to reduce the size of the network.
- The score very rapidly increases till 6 bits, after that it stays flat and not much is to be gained by adding more quantization levels. Six bits mean 64 different levels and are granular enough to train an accurate network.
- As is the case in the training v validation plot in figure 4.6, the validation correctness is often higher than the training correctness. This is because the train network tends to predict more often a data point as negative (background), as most (95 to 5) samples are background this yields a higher correctness. The correctness of the training data is calculated over a over sampled set (50:50) while the validation data is a set which has the original 95:5 ratio.

4.2.7. Normalization and dataset quantization

Normalization has been introduced in chapter 3.2.5. First, it is proved that normalization actually improves the network, and then the and experiment is explained which finds the optimum value for nor-

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
No normalization	91.55 (0.8)	83.22 (2.20)	92.14 (1.02)	42.83 (3.12)	0.56 (0.02)	0.57 (0.02)

Table 4.10: A version of the network where no normalization is used. The STFT output is directly used as input to the neural network. Between parenthesis is the 95% confidence interval

Max norm int	Correctness %	Sensitivity %	Specificity %	Precision %	MCC
8	96.12	76.69	97.48	68.41	0.70
6	96.14	75.09	97.61	68.87	0.70
4	95.93	75.73	97.34	66.95	0.69
2	95.81	75.41	97.23	65.99	0.68
1	94.68	75.15	96.05	57.26	0.63

Table 4.11: Normalization term, conform the algorithm laid out in chapter 3.2.5. Increasing this normalization factor, will decrease the amount of clipping signals but also decrease precision the data as the data is quantized to 256 values in the dataset.

malization. Normalization is an action that is performed on the dataset, but the optimal value can only be found by applying this slightly different data on a network

Experiment 1 - No normalization: To prove that normalization improves the quality of the data and as result the accuracy of the network, a network is trained with data that has not been normalized. It is expected that this network performs worse as compared to the reference network which has been trained using normalized data.

Experiment 2 - optimized normalization: the min-max normalization scheme is an optimization problem. After the maximum value of the first few seconds is determined, this is multiplied by a factor. This is set as the maximum allowed value in the rest of the recording, higher values will clip to this value. The maximum value is set to 1, all other numbers are between 0 and 1 relative to this maximum value. This is then multiplied by 256 and rounded to save as 1-byte ints (effectively quantizing the input to 256 values). By choosing a high number as a normalization factor, the number of values which ”clip” will decrease, but also decrease the precision. Each doubling of the normalization factor means that a number in the int8 describes double the amount of range.

Results and interpretation Results of the not normalized version are shown in table 4.10. The results of the normalization experiment are shown in table 4.11 and plotted in figure 4.7. Some observations:

- A ”normalization factor” of 2 already gives quite decent results. A little surprisingly no decrease in accuracy for a high value was observed. The highest applied normalization factor was 8, this probably is not high enough to demonstrate this expected decrease (although the correctness and other measures are marginally lower for the 6x version, but well within the confidence interval).
- The reference model uses a 16-bit quantized neural network to be at the safe side. In section 4.2.6 it was demonstrated that quantized networks of 6-bits result in a good performance. In the first step of the QNN, the input layer is ”re-quantized” to the quantization level of the network. Thus, in the case of a 6-bit neural network, this means that the 256 values of the input are quantized to 64 values. The networks is thus able to learn using only a few bits, the input is quantized as to 8-bit (uint8), which is a hints means that the input has more range then the network. This hints that there is enough dynamic range in function with different max normalization levels: there is enough ”room” to adequately describe the patterns. This is why the network trains yields the same results when less dynamic range is available.
- The input quantization has no direct effect on the size of the network. The usage of input normalization improves the accuracy but as the result shown it doesn’t really matter which factor in

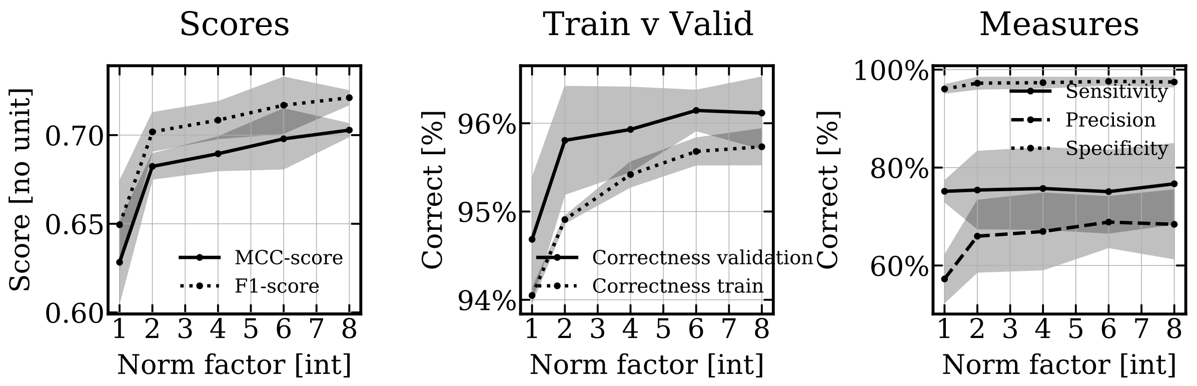


Figure 4.7: Plotted results of table 4.11. The x-axis shows normalization factor, the score rapidly rises and then stabilizes. After a normalization factor of 2 the improvements quickly stabilizes, the number only slowly rises.

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
No montage	94.75 (1.26)	74.01 (10.54)	96.22 (2.08)	58.84 (9.38)	0.63 (0.02)	0.65 (0.02)

Table 4.12: Results of the experiment where no montage was used, this means that the ”dots” from figure 4.8 are directly used as input to the network. This means 19 channels are present as channel 9 and 10 are created from 3 different paths. As always the quoted number is a average of multiple tests and between bracket the 95% confidence interval is shown.

the range 2-8 is used. This hints towards more room for optimization, values could be quantized even more to 4 or 6 bits, however, no standard data types (such as the current int8 type) to save these. This would be a further optimization

4.2.8. Montages

The network standard uses the 18 channels in the double banana configuration (see figure 4.8). However, as explained in the chapter 2.2 montages have there pro's and cons. It is debatable if montages are really needed when letting a neural network interpret recordings. It is also interesting to see what the effect is on accuracy when less channels are used, each channel which is ditched reduces the input space. Two experiments have been performed:

Experiment 1 - Are montages useful: To test if montages are useful at all, a version of the network has been trained without a montage. This version has 19 channels (and thus even adds more channels to the network) as channel 9 and 10 use 3 pads.

Experiment 2 - Use less channels: Using less channels limits the input the ANN. The selected channels are hand picked based on the spatial distributions over the head. The smallest version uses 4 channels, channel 1, 4, 15 and 18. The 6 channel versions adds 9 and 10, and the 10 channel version also adds 5, 8, 11 and 14.

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
10-channels	94.79 (0.92)	75.90 (10.92)	96.12 (1.72)	58.46 (9.90)	0.64 (0.02)	0.66 (0.02)
6-channels	93.95 (0.44)	71.68 (2.52)	95.51 (0.6)	52.95 (2.76)	0.58 (0.02)	0.61 (0.02)
4-channels	91.75 (3.18)	69.01 (12.36)	93.34 (4.26)	43.41 (11.20)	0.50 (0.04)	0.53 (0.44)

Table 4.13: The result of the experiment where less channels are used. The aim is to keep the cover the hole head but less granular. The four channel network uses channel 1, 4, 15 and 18 (see figure 4.8, the 6 channel version adds 9 and 10. The 10-channel version adds more ”coverage” by adding channel 5, 8, 11 and 14. As would be expected the accuracy decreases when less channels are used, however, the 10 channel montage yields quite decent results.

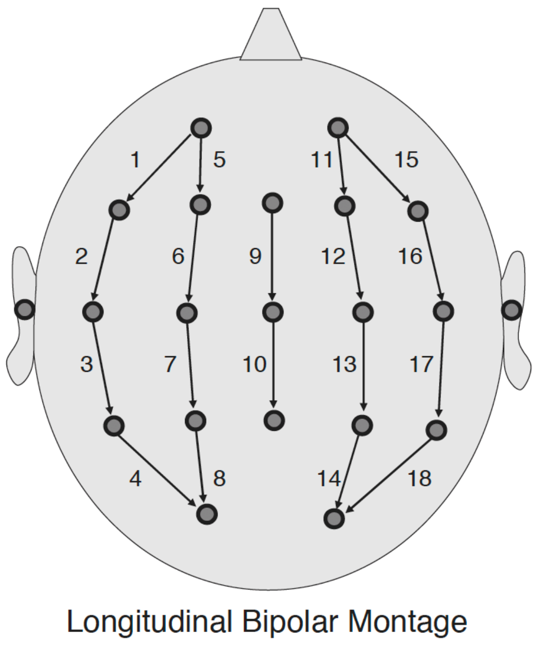


Figure 4.8: A image of the double banana montage (also known as the longitudinal bipolar montage. The dots represent the placements of the EEG pads. Lines are between two pads and represent a channel which consists of two pads. Two ears also have a dot, sometimes these are also included.

Results and interpretation: The accuracy of the no montage experiment is shown in table 4.12, the accuracy when using less channels in shown in table 4.13. Some observations:

- Using no montage is a bad idea, it uses a channel more, but also decreases the accuracy. The montage seems to better organize the data and supress noisy behaviour.
- Using less channels is can be a good idea but don't go to extreme. The version which uses 10 channels actually results in a decent performance. The version with 4 channels less good results, but may be still be interesting as the amount of wires and pads is way lower.

Type	Full	Optimized
Network size	32/64/128 128/64 (129730 elements)	32/64 64/64 (46050)
Montage	Double banana	(Reduced to 10 channels) double banana
Time bins	10	8
Frequency bins	10	8
DC	No	No
Bias	No	No
Quantization (using Brevitas)	6	4

Table 4.14: Two networks are selected and optimized a full version. They are summarized in this table.

4.3. Conclusions

This chapter has explored the effects of tweaking design parameters on the accuracy of the neural network. Based on these experiments two networks are selected which are transformed to FPGA hardware implementations in the next chapter. The first is referred to as ”full”, it tries to be as accurate as possible. The other is ”optimized”, which offers some accuracy in exchange for a smaller implementation. In these decisions, some limitations of the package which transforms the ANN to FPGA

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
Full	96.56	72.26	98.38	75.34	0.711	0.729
Optimized	94.35	70.00	96.04	54.98	0.591	0.616

Table 4.15: The result of the optimized network. This is the best-trained network and thus not an average of multiple runs as in the other results. This is because this trained network is converted to FPGA hardware and not an averaged version.

hardware are taken into account, these limitations will be explained in chapter 5.1.2. Network structures are limited (not all types of layers can be used) and the time and frequency bins need to be equal as this package can only synthesize such networks.

Based on the results of section 4.2.5 and 4.2.4, combined with the knowledge that the output values need to be square (bins time equal bins frequency) a 10x10 input size for the full version and a 8x8 window size for the optimized version has been selected. This is right on the edge where results start to drop rapidly. The input data is quantized at 4 times the max of the first few seconds, this puts it well into the ”save”-zone from section 4.2.7, this is done for both the full and the ”size optimized version”, the network itself is quantized using 6 bits style for the full and 4 bits for the optimized network. As was concluded in section 4.2.8, montages improve results, however, reducing channels also reduces results. For the full version all channels are used, for the optimized version, a 10 channel montage is a good alternative. No biases are used, as section 4.2.2 shows this leads to a worse result and the bandpass filter should take care of DC disturbances. It is not possible to directly select a network from section 4.2.3 for the 8x8 optimized version. However, the lessons are applied by creating a network whom is ”heavy on linear” trainable parameters.

All these factors are summarized in table 4.14. The result of the full and optimized networks can be found in table 4.15. The result is not the result of multiple tests, but just the best-achieved version because the best version will be converted to hardware.

5 Hardware implementation

This chapter will describe the last step in the process, the transformation of the ANN to an (FPGA) hardware design. This is done with the help of an experimental codebase by the Xilinx Research lab called FINN [26]. FINN consists of two parts: a project which aims to research architectures to generate Quantized Neural networks (QNNs) and a compiler architecture that can generate such structures on an FPGA. This chapter will aim to introduce the main concepts of this dataflow architecture and explain how the compiler can be used to generate FPGA data files. FINN uses Vivado to do the heavy lifting in synthesizing hardware, an Ultra 96V2 development board by Avnet is used to run the synthesized network. The first section of this chapter will be about FINN and its limitations, the second will explain the practicalities, and the last part is dedicated to the generated circuit.

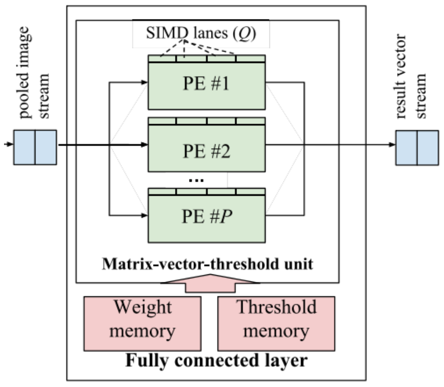
5.1. FINN

As FINN is still an experimental package, no real documentation has been written. Instead, the to-go way is adapting pre-written examples, this is a practical way of leading users around unknown bugs and show them what is possible other than what is not. In order for FINN to be used, the neural network must be trained using the Brevitas package (which the chosen implementations are). This package is a layer upon vanilla PyTorch which makes QNNs possible. The network must be saved in the ONNX (Open Neural Network Exchange) standard, this is an open standard that makes neural network structures interchangeable between different packages such as PyTorch, Keras, and others. More importantly, it allows the addition of extra information fields which is important to describe the specific properties of the QNN. Together with this network, a ”golden standard” should be supplied, this is one element and one prediction pair from the original network, it will be used to check if the network still works correctly during the transformation to FPGA hardware.

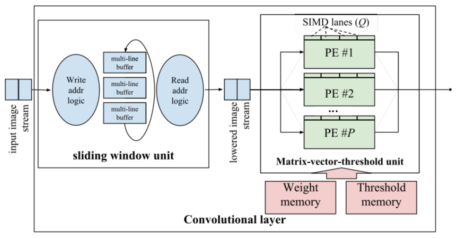
FINN is designed to work on devices that support Pynq (Python Productivity for Zynq), Pynq makes it possible to use the programmable logic together with an FPGA. In practice this means that is it possible to run Python scripts. Using build-in functions of Python, data can be transferred from and to the FPGA ANN. These extra software layers make it much easier to check, debug and generally work with the FPGA logic. The FINN compiler doesn’t only generate a bitstream for the FPGA, it also generates a bunch of Python scripts. These scripts can transfer the data files to bit files and transfer them to the hardware blocks.

5.1.1. Implementation of layers

FINN converts all layers of the neural network separately to hardware. Such an implementation of an layer consists of two FIFO’s. One contains the input signals of the layer and the other the values of the trained layer which are combined with the values of the activation layers such as Batch norms and



(a) A visual representation of a linear layer. The input is cut in pieces and streamed in parts through the network. Each cycle weight of neurons is loaded in the processing elements (PE's) and corresponding inputs are "streamed" through multiple lanes. This way the linear layer is implemented in multiple cycles



(b) The convolutional version adds a sliding window before the layer. In a convolutional network, a kernel is used which shifts over the input values, input values are used multiple times, these slices are created by the sliding window unit.

Figure 5.1: The two main building blocks of FINN, (left) a schematic representation of a linear layer and (right) the version with a sliding window unit that can be used to implement convolutional layers.

sigmoids. Figure 5.1a shows a schematic implementation of a linear layer, a FIFO before the layer sends parts of the total input to the processing element (PE), it also loads the relevant weights from memory, the PE calculates the outputs, and then saves them in the FIFO at the output. A convolutional layer works in a very similar way, the schematic is shown in figure 5.1b, in a convolutional network a kernel is used which shifts over the input values, values are used multiple times. To accommodate this, a "sliding window unit" is used which shifts through windows.

Layers differ in size, and thus the number of loops needed to calculate the layer. Ideally, each layer takes the same amount of time, FINN can optimize for this by varying the FIFO depths and the number of PE's. It is possible to tweak each layer separately, this can give greater control over the types of memory which will be used.

5.1.2. Limitations

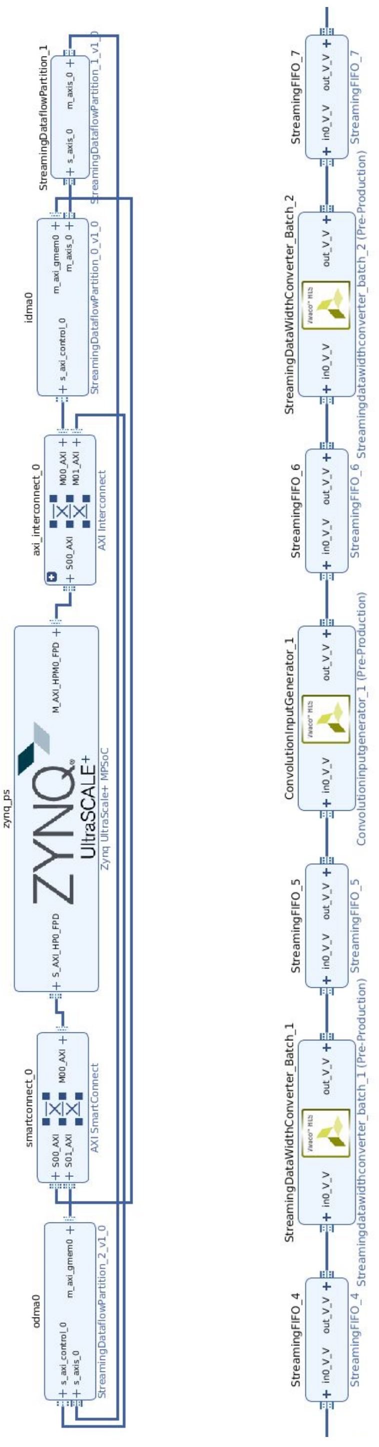
FINN is still a very "young" package and changes from month to month. The version used in this work was was version 0.5b. The most important limitations were:

- Only square inputs are allowed. This means that the time and frequency bins should be equal, 10x10 is alright but a version with 10x8 will not work.
- The last layer of the convolutional part of the neural network can only have channels. A 2D-convolutional layer is described by the number of channels, width, and height (in the first layers this there are 18 channels, 10 frequency bins, and 10-time bins). The kernel sizes modify the weight and height of each layer. The last convolutional layer, the width and height should be 1, so this layer should look like Nx1x1.

5.2. Design

5.2.1. Steps of compiler

FINN tries to optimize the network based on inputs such as throughput and speed, as explained in section 5.1.1. The FINN compiler has several steps, each step will be discussed shortly and high-level


odma0
StreamingDataflowPartition_2_v1_0
smartconnect_0
AXI SmartConnect
zynq_ps
ZYNQ UltraSCALE+
Zynq UltraScale+ MPSoC
S_AXI_HP0_FPD
M_AXI_HPM0_FPD
axi_interconnect_0
AXI Interconnect
idma0
StreamingDataflowPartition_0_v1_0
StreamingDataflowPartition_1
StreamingDataflowPartition_1_v1_0
StreamingFIFO_4
StreamingDataWidthConverter_Batch_1
StreamingFIFO_5
ConvolutionInputGenerator_1
StreamingFIFO_6
StreamingDataWidthConverter_Batch_2
StreamingFIFO_7

(a) The top level-view of the synthesized accelerator system.

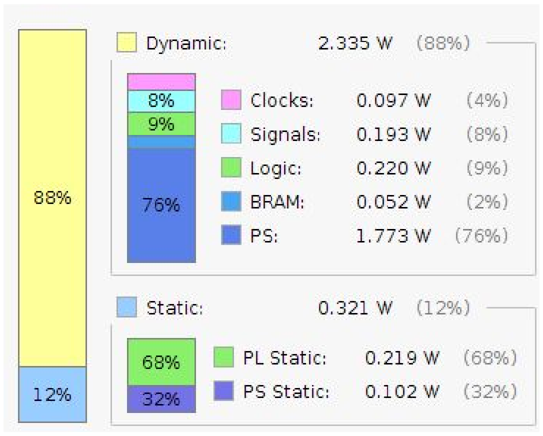
(b) A part of the accelerator.

Figure 5.2: (a) shows the top view of the synthesized network. The Zynq UltraScale+ processor runs the PYNQ image which loads the input data and sends it through an AXI-bus and DMA to the synthesized neural network, this is called an accelerator. The accelerator is is the ”StreamingDatatFlowPartition” element. (b) Shows (a part) of what’s inside this accelerator, ”simple” blocks with stream data through the networks. This is the implementation of figure 5.1.

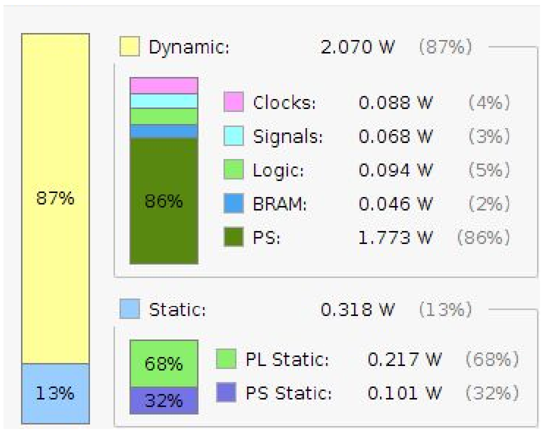
as they are performed in the package.

- **Step 1:** the ONNX model is loaded, some clean-up work is done (renaming) and layers are combined (i.e., linear and batch norm), standard arithmetic operations such as constant number additions or divisions are converted to multiplication elements (MULs). Most of this is just housekeeping.
- **Step 2:** Based on inputs such as the throughput and demanded prediction speed, the FIFO-loops are determined, this generation is based on a model of the output
- **Step 3:** All layers are separately converted to a combination of VHDL and behavioral descriptions.
- **Step 4:** These files are converted to hardware elements and tested
- **Step 5:** Together with the generated Python code, the bitfile is packaged in a folder that is ready to be deployed on the FPGA.

Figure 5.2a shows a schematic of the Vivado stitched design. Every project generated with FINN will look like this. The Zynq UltraScale+ is a separate processor integrated into the die of the FPGA, it is an ARM processor which runs Pynq, on a Linux variant, which runs Python. The Python module can open the datafiles which it then sends through an AXI bus to a Direct Memory Access (DMA) block. The created network lives inside the "StreamingDataflowPartition" block. Figure 5.2b shows a part of this block (the actual one is 50 blocks long), it consists of a long stream of elements consisting of FIFO, conventional, and batch layers. After this, the result is written to a DMC block which then transfers the data through an AXI bus back to the Pynq process.



(a) Power as quoted by Vivado for the full implementation of the network

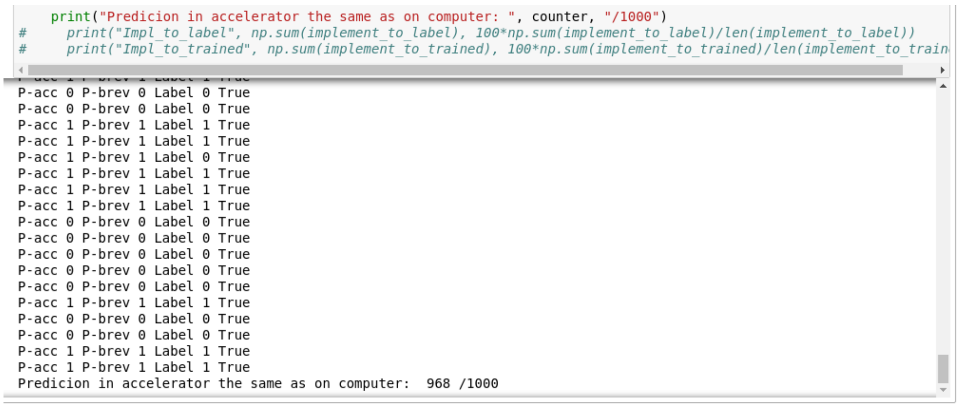


(b) Power as quoted by Vivado for the full implementation of the network

Figure 5.3: Power figures are generated by Vivado. It's hard to make real comparisons based on these numbers. Most energy is consummated by the processor (PS) and static consumption of the FPGA (PL). The logic

Resource	Full	Optimised	Available
LUT	49084 (69.56%)	40823 (57.86%)	70560
LUTRAM	10569 (36.7%)	3191 (11.08%)	28800
FF	47924 (33.96%)	51315 (36.36%)	141120
BRAM	32.5 (15.05%)	40 (18.52%)	216
BUFG	9 (4.59%)	9 (4.59%)	196

Table 5.1: Two networks are selected and optimized a full version. They are summarized in this table. The optimized version uses less LUTRAM than the full version but more BRAM. LUT=LookUp Table, FF=Flip-flop, BUFG=clock buffers



```
    print("Predicion in accelerator the same as on computer: ", counter, "/1000")
#     print("Impl_to_label", np.sum(implement_to_label), 100*np.sum(implement_to_label)/len(implement_to_label))
#     print("Impl_to_trained", np.sum(implement_to_trained), 100*np.sum(implement_to_trained)/len(implement_to_train
```

```
P-acc 0 P-brev 0 Label 0 True
P-acc 0 P-brev 0 Label 0 True
P-acc 1 P-brev 1 Label 1 True
P-acc 1 P-brev 1 Label 1 True
P-acc 1 P-brev 1 Label 0 True
P-acc 1 P-brev 1 Label 1 True
P-acc 1 P-brev 1 Label 1 True
P-acc 1 P-brev 1 Label 1 True
P-acc 0 P-brev 0 Label 0 True
P-acc 0 P-brev 0 Label 0 True
P-acc 0 P-brev 0 Label 0 True
P-acc 0 P-brev 0 Label 0 True
P-acc 0 P-brev 0 Label 0 True
P-acc 1 P-brev 1 Label 1 True
P-acc 0 P-brev 0 Label 0 True
P-acc 0 P-brev 0 Label 0 True
P-acc 1 P-brev 1 Label 1 True
P-acc 1 P-brev 1 Label 1 True
Predicion in accelerator the same as on computer:  968 /1000
```

Figure 5.4: 1000 input examples are thrown at the synthesized accelerator running at the Ultra96V2 FPGA.

5.2.2. Implementation

FPGA implementations are able to detect with an extremely high speed. Multiples of 10-thousands per second are no exception, however, the faster the demanded speed the more PEs and faster memory are needed. Live detection only generates two data bins per second, does only need a few detections per second. The chosen network can be quite big as it contains 18 (or 10) input channels. FINN allows deep control in how the network is synthesized, by specifying memory types, the amount of PEs, and more on a per-layer level. However, as figure 5.2b shows, data ”flow” from left to right in a pipeline. Ideally, all elements of the pipeline take an equal amount of time. Luckily, FINN has optimization scripts for this which only need a few hyperparameters. These hyperparameters specify things like clock speed and frames per second (FPS, the number of recordings that can be classified by the accelerator per second). On these hyperparameters, FINN then runs an optimization script that calculates the memory modes and PE's for all layers in the network which can then be used to generate hardware.

To be fair, this makes it sound simpler than it in reality is. Most networks fail with strange and badly explained mistakes, so when a relatively slow network was finishing the process we're already happy. The used hyperparameters in this work were:

- **target_fps=5000**: The network will be able to predict up to 5000 EEG recordings per second. This is quite high, but lower values tended to crash the synthesizes.
- **synth_clk_period_ns=5.0**: The clock frequency used on the FPGA side which is 200MHz.
- **mvau_wwidth_max=5000** sets the maximum allowed FIFO depth. This number is so high that the FIFO depth is in practice, not an important boundary condition.

Result Both selected networks, full and optimized, have been synthesized using the same hyperparameters which were discussed in section 5.2.2. With these parameters both networks fitted on the programmable logic (PL) of the FPGA, in some other configurations, this was not possible. A common problem in these unsuccessful attempts was memory. Two types are available on the FPGA, BRAM, and LUTRAM, LUTRAM uses logic elements like RAM, this has the advantage of being faster (BRMA needs one clock cycle) and being distributed (closer to the logic which needs it). but uses more energy. Very fast implementations (target_fps in the order of hundred thousand) need a lot of LUTRAM, which simply isn't available on the board and thus fail. Table 5.1 show the resource allocation after synthesis. Only slightly fewer LUTs (Lookup Tables; the main logical element in an FPGA) are used in the optimised network, while the network has way less trainable parameters. The RAM usage has shifted big time from the faster LUTRAM to the more efficient BRAM. Together with the increase in flip flop (FF) use, who are typically used to hold data in between clock cycles, this suggests that more PE's are used while the FIFO depths are shallower.

Figure 5.3 shows the power consumption estimated by Vivado, this is tricky as only the power consumption of the accelerator is of real interest. However, most energy consumption is consumed by the peripherals of the network implementation such as the Zynq processor (PS and PS Static). Most of the logic and en RAM is actually used to implement the neural network, however it is hard to draw too much conclusions from this. It is clear though that by using less LUTRAM the overal consumption energy of the logic is much reduced. From figure 5.3, the clock, signals, logic and BRAM power consumption can be associated with the consumption of the accelerator. For the full version this is 0.562 W and for the optimized version 0.296 W, a factor of 1.9x. This shows that a smaller network does indeed use less energy.

6 Results

This chapter will discuss the performance of the two selected networks. First, accuracy for different seizure types is discussed. Then the performance per patient is visualized using histograms. After the trained network is hooked up to stream EEG data directly from the source through the network, this tests if the accuracy is equal to the validation data. Finally, the robustness of the network is tested by adding noise to the input data.

Type	Correctness %
CPSZ (Full)	69.19
CPSZ (Optimized)	76.84
FNSZ (F)	63.38
FNSZ (O)	63.30
TCSZ (F)	70.21
TCSZ (O)	68.02
ABSZ (F)	91.01
ABSZ (O)	88.23
BCKG (F)	98.45
BCKG (O)	96.04

Table 6.1: The accuracy of the selected networks for different seizure types and the background signals. This gives insight into how simple/hard it is to detect certain types of seizures. Only the correctness is shown for the validation data. Table 3.2 shows all seizure types, only the types with more then 1000 data frames are included

6.1. Accuracy

6.1.1. Accuracy per seizure type

During the training phase of the neural network, all seizure types were categorized as one type: a seizure/positive. However, not every seizure type is equally difficult/easy to detect. To analyze this the seizures were tested per type. Table 6.1 shows the result of this experiment. Only the correctness is shown in this table as true positive and false negative are zero per definition for seizures (they are either a correct predicted seizure (TP), or an incorrect predicted as background (FN)), in this case, correctness is the same as sensitivity (or true positive). For the background signals, the correctness is the same as the specificity as TP and FN are zero by definition. Only seizure types with more than 1000 data points (this is different than the *number* of recordings show in table 3.2) are used, otherwise, results are not deemed relevant as the sample is too small. Some observations:

- Generally speaking the full network is a little better in predicting than the optimized network, which makes sense as this QNN can use more resources.
- ABSZ (absence seizures) are easiest to detect (91% and 88%). This makes sense as absence seizures result in a complete loss of consciousness and are thus a pretty extreme event. This should result in big spikes which are easy to detect.
- FNSZ (focal non-specific seizure) is the worst performer (only 63% for both networks). This makes sense as FNSZ only are in a part of the brain, and thus more difficult to detect.
- The same is true for CPSZ (complex partial seizure), these are typically only detected in a part of the brain.

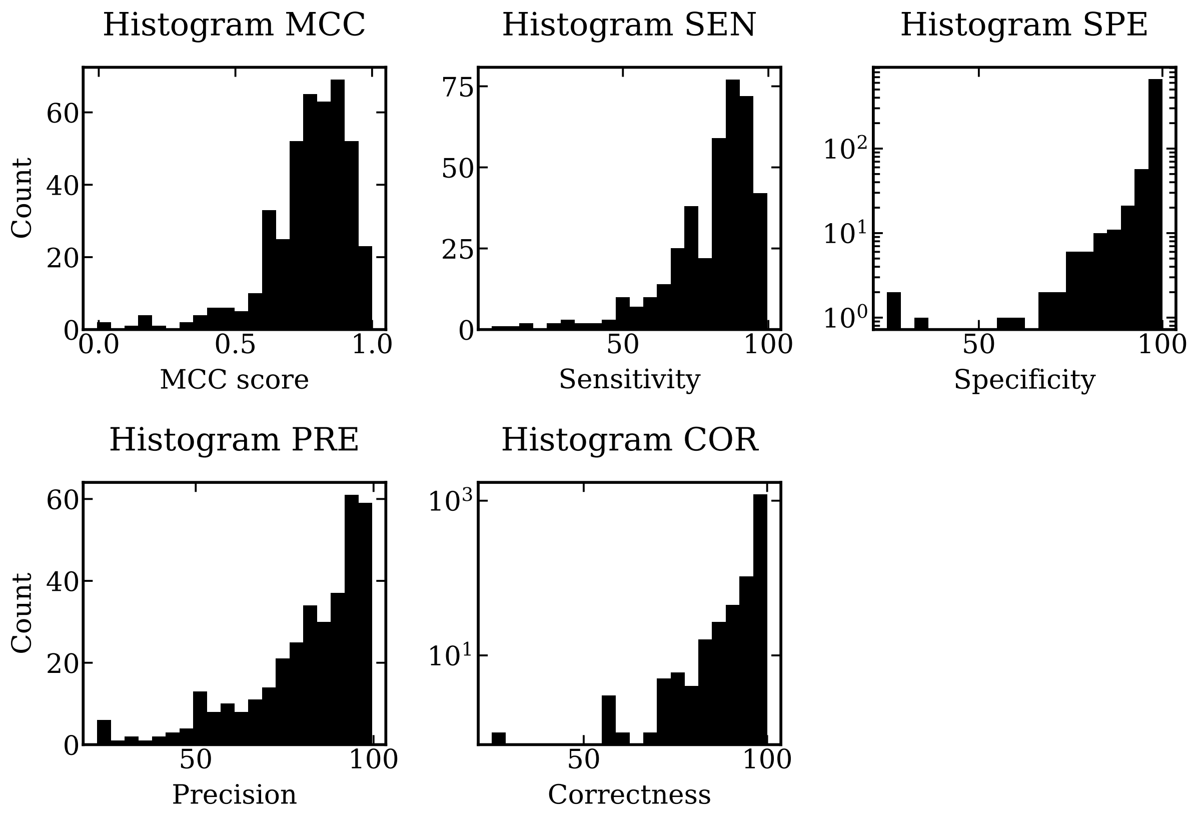


Figure 6.1: The per-session results of the full network. All data histograms use bin sizes of 5% (or for the MCC score just 0.05). Please note that the correctness and specificity are plotted on a logarithmic scale.

6.1.2. Accuracy per patient session

The data used to train the network is sourced from a lot of different patients. It is interesting to look at the results on patient level. It may be that the QNN has been optimized for patients where a lot of data was available. The dataset has 1423 recording sessions, the quality parameters were calculated for all of them separately. The results were then plotted as histograms, these are shown in figure 6.1 for the full network, and 6.2 for the optimized network. Some observations:

- Both correctness and specificity are correctness are log scale. A big chunk of the recordings are just background signals, the neural network is very good at recognizing this. This results in a completely correct detection, which means a 100% score for specificity and correctness.
- The results of the full network really feels like somebody pushed the result from the optimized version to the right. This is good, it means that the performance of the full network is improved over the whole range.

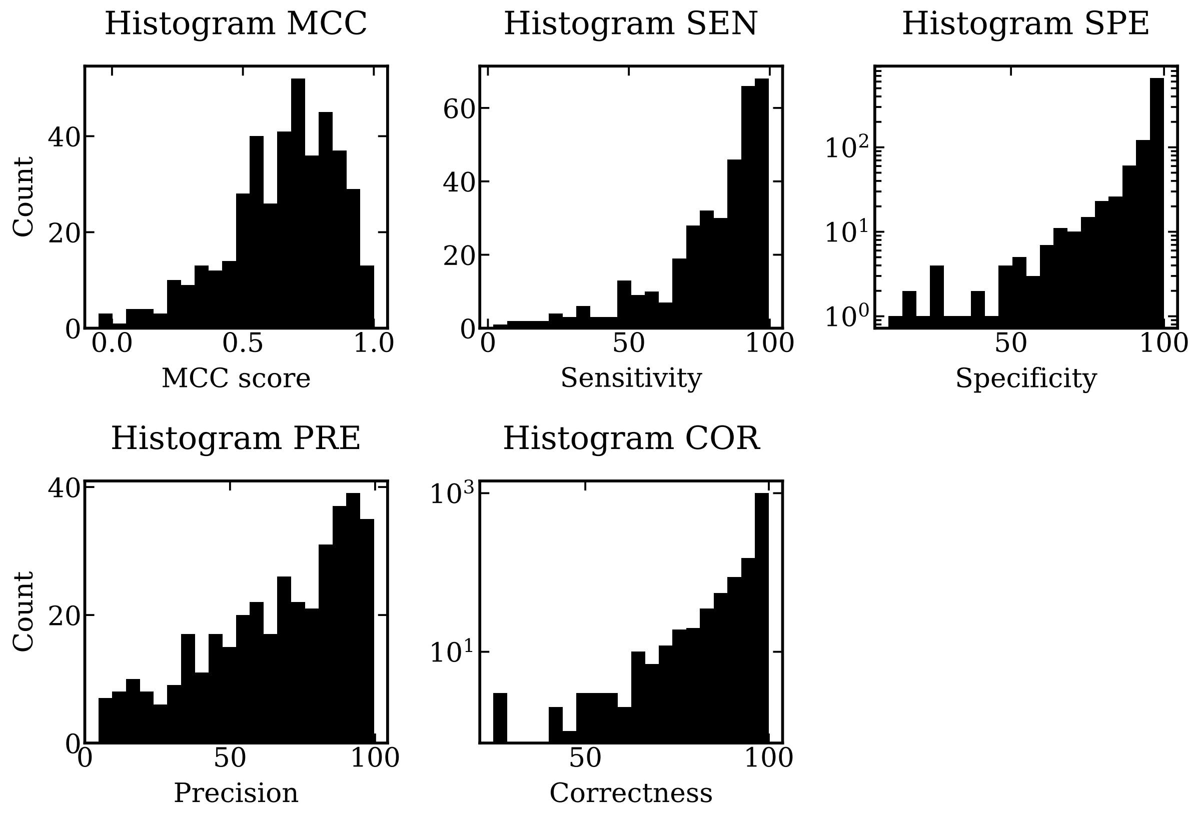


Figure 6.2: The per-session results of the optimized network. All data is histograms use bin sizes of 5% (or for the MCC score just 0.05). Please note that the correctness and specificity are plotted on a logarithmic scale.

- The MCC score can only be calculated for recordings that contain both seizures and non-seizures (see formula 2.13, the denominator becomes zero), thus all those 100% only background recordings are not included in the MCC histogram. This results in an almost normal-like distribution with a ”fat” tail to the left (the lower MCC values). The mean value of the full version is clearly higher than the peak of the optimized version.

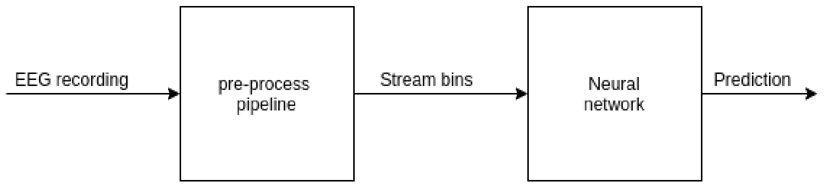
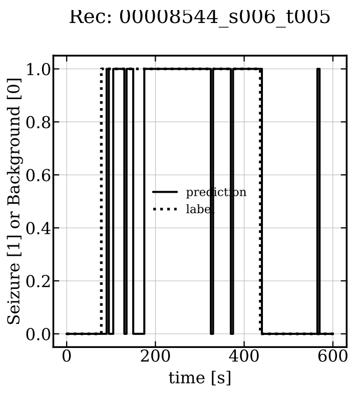


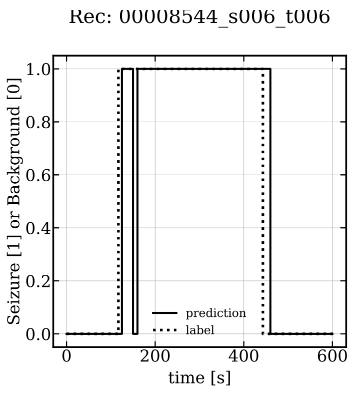
Figure 6.3: Data is loaded from the original dataset as EDF, then pre-processed and then moved through the neural network

6.2. Streaming real EEG data

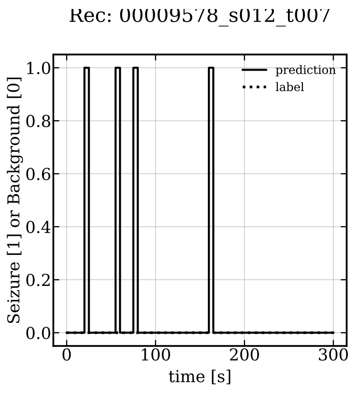
The data frames on which the neural network is trained are not 100 percent compared to real detection data. When training random data point out of a big dataset are selected, the consecutive data points have almost nothing in common. When real data is applied, the two consecutive frames are from the same recording/patient. There is probably a high covariance between these two elements. When the dataset was created, seizures and background signals are separated and saved at a different location. The network never trains on the border regions between the background (no data frames have an overlap in the background and the seizure part). In a real situation, data is streamed to a neural network, this leads to a delay of 5 (full) or 4 (optimized) seconds in prediction. First, a data frame has to be completely recorded before it can be used by the neural network. This introduces a lag in



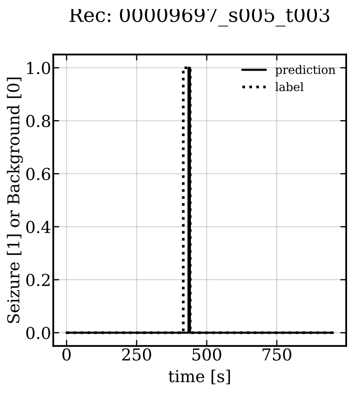
(a) A long seizure between 100 and 425 seconds.



(b) A 10-minute recording which a very long seizure during



(c) This is a recording of 5 minutes (300 seconds) of background signal, sometimes a false positive spike happens.



(d) A little spike which detected by the trained network

Figure 6.4: Four (selected) examples of prediction results, the dotted are labels (from the dataset), the black line shows what the neural network has predicted. A high signal means a seizure is happening or predicted. These plots were made using the full version of the network.

prediction (the so-called time to prediction).

The real challenge for the developed neural network is to detect seizures in a "real setting". To simulate this the raw/original data is streamed through the network. This method is shown in figure 6.3, all data is processed and put through the network. Figure 6.4 shows four (selected) examples of this method, plotted together with its labels. The lag can be seen as predictions often rise after the label goes high. By using overlapping recordings for the data frames this number can be reduced to half the length of the data frame, on average 2.5 seconds for the full version and 2 for the optimized version.

Table 6.2 shows the result of the experiment where all data is streamed through the network. Some observations:

- Generally there is quite a big drop in quality compared to the table 6.1. Especially the sensitivity has quite a steep drop. A few possible explanations for this were given in the text above: "edge" problems and lagging detection. Especially the second factor can play a big role. The mean time of a seizure is 75 seconds (see table 3.2), if the first 5 seconds are wrong this has a big effect on specificity.
- The ΔMCC -0.061 for full and -0.131 for the optimized network. The optimized network is impacted more by the streaming data.

Type	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC	F1
Full	95.16	62.25	97.80	66.46	0.62	0.65
Optimized	93.31	56.90	95.49	43.18	0.46	0.48

Table 6.2: The numbers for all recordings when tested in a streaming fashion. The numbers should be compared to table 4.15, especially the sensitivity is notably lower. The performance of the optimized network is reduced more than the full version. The ΔMCC's are -0.061 for full and -0.131 for the optimized network.

Noise factor	Accuracy %	Sensitivity %	Specificity %	Precision %	MCC
0	98.51	57.14	99.41	68.08	0.61
0.1	98.51	58.92	99.37	67.34	0.62
0.2	98.29	55.35	99.22	60.78	0.57
0.3	98.44	55.35	99.37	65.95	0.59
0.4	98.13	53.57	99.10	56.60	0.54
0.5	97.45	48.21	98.52	41.53	0.43

Table 6.3: The noise experiment selected 250 recordings which were applied to the full network. First, the accuracy without noise was calculated, then the noise level was steadily increased. When more noise is applied the accuracy of the network decreased as expected.

6.3. Noise performance

Noise performance is important for the robustness of the system, a neural network that can still detect when a (lot of) noise is added is more capable in the real world. The noise is calculated with formula 6.1, normally distributed numbers are generated based on the mean and std of the clean signal. This array of normalized numbers is then multiplied by a ”noise-factor” and added to the original signal, when this noise factor is zero no noise is applied. The noise factor can be varied to see the effect of different noise levels on the source data.

$$Signal_{noise} = signal_{original} + noise_{normal-dist}(mean(signal_{clean}), std(signal_{clean})) * factor \quad (6.1)$$

Figure 6.3 shows the result of this method on a typical recording that contains only a background signal. The noise factor was increased from 0 to 2, the higher this number the more false positives were recorded.

Experiment: Adding noise has been tested on more data. From the 1423 original recordings, 250 were randomly selected, these were applied to the full network. First, a version without any noise was tested, then the noise was added in a range from 0 to 0.5. This was not tested on all data as it is quite computationally heavy. The result of this is shown in table 6.3.

A few observations:

- The 250 recordings were streamed from the original files, the result of the zero-noise version is in line with table 6.3..
- As would be expected accuracy decreases as the noise level increases.

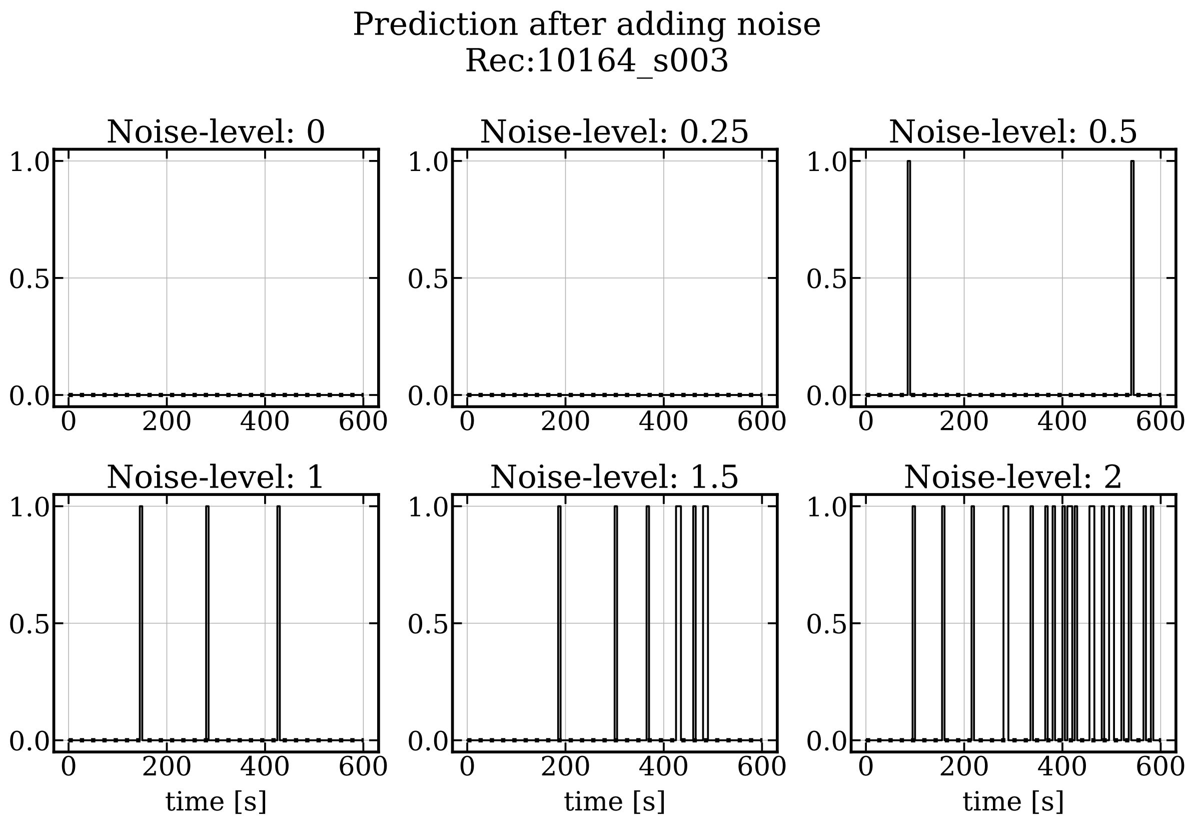


Figure 6.5: Adding noise to the network distorts the signal. These figures show an example of what happens in a recording when big noise signals are added. The original signal (left top) was predicted correctly, when noise was added more and more false positives emerged.

7 Conclusions

7.1. Summary of results

This thesis researched the viability of a (near)-real-time lower power epileptic using quantized neural networks. A big dataset was analyzed and processed to make it suitable for neural networks. This neural network was quantized and optimized by experimenting with different design parameters. Two suitable networks were selected out of these experiments and converted to FPGA hardware.

In chapter 3 the TUSZ dataset was selected and its properties discussed. The set was robustly cleaned and a process set up to accommodate a transformation to STFT. The output was then normalized using a modified min-max process. After this step, the data was quantized to fit in just one byte, which greatly reduced the size of the dataset after processing.

In chapter 4, a neural network architecture was selected and the general training procedure was explained. A method to deal with a 95:5 ratio between background and seizures was introduced. This network is used to use experiments to optimize the neural network in size. A whole range of experiments has been performed. The effects of adding or removing data by changing how many time bins, frequency bins, or channels were explored. The effects of sizing the network from very small to very big and applying a bias were explored. Brevitas was introduced to experiment with quantizing the network and test for the ideal level of network quantization. The min-max quantization/normalization scheme was also fine-tuned using the accuracy of the neural network. All this information was then compiled into two possible implementations, one optimized for accuracy and one for size.

In chapter 5, these selected networks were used to create an FPGA hardware implementation using FINN, which creates an HDL description using the neural network and uses Vivado to create an FPGA implementation. This showed the feasibility of the detection scheme.

In chapter 6 the results of the selected networks were tested by looking at the result per seizure type. The neural network was projected on the original dataset by looking at the accuracy of the original recording. The robustness of the network was tested by adding noise to the input.

In section 1.1 the question of this thesis was defined. These will now be answered.

- **Are datasets available that are suitable to this task of general-purpose epilepsy detection?** A few datasets were available, most were either too small or of too low quality. Only the TUSZ was sufficiently big and of high quality. The TUSZ is currently the only serious freely available epilepsy dataset.

- **How to design a pre-process pipeline to clean the source data?** EEG time signals are big, luckily the size can be reduced. Normalization turned out to be a great trick in reducing data size. By smartly designing a process pipeline it was possible to greatly reduce the footprint of the original dataset. The original dataset was +50GB, the final dataset was only 1.25GB.

- **Which trade-offs are made in sizing the networks and how does this reduce accuracy?** Multiple ”vectors of attack” were identified and analyzed such as time, frequency, quantization (both input and the network), network size, and channels. The philosophy was to reduce as much as makes sense. This will always have an effect on accuracy but the author believes smart choices were made.
- **How to implement a hardware solution and which trade-offs are made there?** The neural network is implemented on hardware using FINN and Vivado. The main trade-off here is the prediction vs hardware costs. Slower prediction requirements result in less power-consuming hardware used. The biggest trade-off is already made in the network design: a smaller QNN equals a smaller implementation.

The main question was defined as:

Is it possible to design a low-power epilepsy detection system with reduced input data? As this work shows this is possible. By sufficiently processing incoming EEG data and minimizing the neural network design. The resulting networks have a sensitivity of 72.26% and 70.00%, specificity of 98.38% and 96.04%, and an MCC score of 0.711 and 0.591 (F1 0.729 and 0.616). Comparing this work based on different datasets is difficult as most work is done on datasets that are orders of magnitude smaller. Work done by IBM [19] on the TUSZ dataset quotes an F1 score of 0.722 for their CNN implementation, which is very much in line with the score in this work, however, their work wasn't minimized. [27] claims a maximum sensitivity of 71.61%, specificity of 83.7 %, and an F1 of 0.51, this works performance notably better on all these measures while being minimized.

The first FPGA implementations consume 0.296 W and 0.562 W at 50000 detections per second. This is only a first step in developing a hardware implementation, there is a lot of room for optimization and improvement. It shows two things: it is possible to minimize a network to fit on an FPGA and reducing its size does reduce its power consumption.

7.2. Future work

This work touched on a broad range of subjects. It is thus that it was not possible to go in every ally as deep as one wants. Below some suggestions are made, sorted into categories.

Dataset and processing

- Input data is saved as int8, it might be possible to use even smaller datatypes. It is quite possible that 4, 6, or 8 bits quantization will work just fine. Computers don't like this datatype, 1 byte is used for the smallest datatypes thus some workarounds may be needed. Input quantization should probably be done with more or equal bits than the network quantization.
- The current dataset tries to include all data which was in the original dataset. However, 5 out of 6 recordings only contain background signals. It may be useful to exclude those, or at least partly and the mixed recording (recordings containing both seizures and background) signals are probably harder to detect.
- All seizure types are used in the current implementation. Depending on the application it may not be needed to detect all seizures, DBS systems for example are only used to treat certain types of seizures. It may be interesting to play a little with this requirement.

Neural network

- An optimization which was not used in this work is purging the network. This means deleting a certain fraction of the neural network connection. FINN will not synthesize any hardware for these connections, it is not uncommon to delete more than 50% of parameters with only a minimal drop inaccuracy.

- A standard loss function was used, this loss function only considers the labels and the predictions which in essence optimizes for correctness. It may be possible to implement custom loss functions that can optimize for the MCC-score or the F1 score.
- The data input to the ANN were sequential, no overlapping was allowed. Implementing overlapping can decrease detection time, waste fewer data samples, and reduce the need for oversampling by generating more seizure data.

Hardware

- While this work constantly talks about small network = lower power consumption, it is almost impossible to make great claims about this using data from the FPGA. Xilinx only gives highly contextual information. To make realistic predictions the model should be transferred to an ASIC. However, much has to be changed to make this work:
  - The normalization scheme and STFT need to be implemented on the programmable logic. This is can be implemented as a separate accelerator. The processor first uses an accelerator to process raw input data. This is then fed to the neural network accelerator.
  - Next the two accelerators should be merged to one, the input and output connected to the IO of the FPGA. The microprocessor is not needed anymore.
  - All layers of the neural networks and FIFO's are available in ”plain” VHDL. This can be the starting point of an ASIC implementation.
- The generated neural network implementation should pipeline the data like a processor. The neural networks flow through phases, so multiple detections can happen a the same time. To make this happen the HDL files should be altered, but as the phases already take roughly equal time this should be doable.

Bibliography

[1] Jayant Acharya and Vinita Acharya. Overview of eeg montages and principles of localization. *Journal of clinical neurophysiology : official publication of the American Electroencephalographic Society*, 36:325–329, 09 2019. doi: 10.1097/WNP.0000000000000538.

[2] Ralph Andrzejak, Kaspar Schindler, and Christian Rummel. Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 86:046206, 10 2012.

[3] Senior Art Director, Art Director, Sales Director, Circulation Director, Chief Financial Officer, Business Intelligence, and Corporate Branding. Examining the economic impact and implications of epilepsy. 2020.

[4] UK Epilepsy Society. Vagus nerve stimulation, 2020. URL https://epilepsysociety.org.uk/about-epilepsy/treatment/vagus-nerve-stimulation.

[5] Max Heinrich Fischer and Hans Löwenbach. Aktionsströme des zentralnervensystems unter der einwirkung von krampfgiften. *Naunyn-Schmiedebergs Archiv für experimentelle Pathologie und Pharmakologie*, 174(3-4):357–382, 1933.

[6] Ying Gu, Evy Cleeren, Jonathan Dan, Kasper Claes, Wim Van Paesschen, Sabine Van Huffel, and Borbála Hunyadi. Comparison between scalp eeg and behind-the-ear eeg for development of a wearable seizure detection system for patients with focal epilepsy. *Sensors*, 18(1), 2018. ISSN 1424-8220. doi: 10.3390/s18010029. URL https://www.mdpi.com/1424-8220/18/1/29.

[7] A. Harati, S. López, I. Obeid, J. Picone, M. P. Jacobson, and S. Tobochnik. The tuh eeg corpus: A big data resource for automated eeg interpretation. In *2014 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, pages 1–5, 2014. doi: 10.1109/SPMB.2014.7002953.

[8] Suzana Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proceedings of the National Academy of Sciences*, 109 (Supplement 1):10661–10668, 2012. ISSN 0027-8424. doi: 10.1073/pnas.1201895109. URL https://www.pnas.org/content/109/Supplement_1/10661.

[9] Matthias Ihle, Hinnerk Feldwisch-Drentrup, César A. Teixeira, Adrien Witon, Björn Schelter, Jens Timmer, and Andreas Schulze-Bonhage. Epilepsiae - a european epilepsy database. *Comput. Methods Prog. Biomed.*, 106(3):127–138, June 2012. ISSN 0169-2607. doi: 10.1016/j.cmpb.2010.08.011. URL https://doi.org/10.1016/j.cmpb.2010.08.011.

[10] M.Kemal Kıymık, İnan Güler, Alper Dizibüyük, and Mehmet Akın. Comparison of stft and wavelet transform methods in determining epileptic seizure activity in eeg signals for real-time application. *Computers in Biology and Medicine*, 35(7):603–616, 2005. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2004.05.001. URL https://www.sciencedirect.com/science/article/pii/S0010482504000691.

[11] The Royal Children's Hospital Melbourne. Antiepileptic medications, 2020. URL https://www.rch.org.au/neurology/patient_information/antiepileptic_medications/.

[12] Dileep R. Nair, Kenneth D. Laxer, Peter B. Weber, Anthony M. Murro, Yong D. Park, Gregory L. Barkley, Brien J. Smith, Ryder P. Gwinn, Michael J. Doherty, Katherine H. Noe, Richard S. Zimmerman, Gregory K. Bergey, William S. Anderson, Christianne Heck, Charles Y. Liu, Ricky W. Lee,

Toni Sadler, Robert B. Duckrow, Lawrence J. Hirsch, Robert E. Wharen, William Tatum, Shraddha Srinivasan, Guy M. McKhann, Mark A. Agostini, Andreas V. Alexopoulos, Barbara C. Jobst, David W. Roberts, Vicenta Salanova, Thomas C. Witt, Sydney S. Cash, Andrew J. Cole, Gregory A. Worrell, Brian N. Lundstrom, Jonathan C. Edwards, Jonathan J. Halford, David C. Spencer, Lia Ernst, Christopher T. Skidmore, Michael R. Sperling, Ian Miller, Eric B. Geller, Michel J. Berg, A. James Fessler, Paul Rutecki, Alica M. Goldman, Eli M. Mizrahi, Robert E. Gross, Donald C. Shields, Theodore H. Schwartz, Douglas R. Labar, Nathan B. Fountain, W. Jeff Elias, Piotr W. Olejniczak, Nicole R. Villemarette-Pittman, Stephan Eisenschenk, Steven N. Roper, Jane G. Boggs, Tracy A. Courtney, Felice T. Sun, Cairn G. Seale, Kathy L. Miller, Tara L. Skarpaas, and Martha J. and Morrell. Nine-year prospective efficacy and safety of brain-responsive neurostimulation for focal epilepsy. *Neurology*, 95(9):e1244–e1256, 2020. ISSN 0028-3878. doi: 10.1212/WNL.0000000000010154. URL https://n.neurology.org/content/95/9/e1244.

[13] Dileep R. Nair, Kenneth D. Laxer, Peter B. Weber, Anthony M. Murro, Yong D. Park, Gregory L. Barkley, Brien J. Smith, Ryder P. Gwinn, Michael J. Doherty, Katherine H. Noe, Richard S. Zimmerman, Gregory K. Bergey, William S. Anderson, Christianne Heck, Charles Y. Liu, Ricky W. Lee, Toni Sadler, Robert B. Duckrow, Lawrence J. Hirsch, Robert E. Wharen, William Tatum, Shraddha Srinivasan, Guy M. McKhann, Mark A. Agostini, Andreas V. Alexopoulos, Barbara C. Jobst, David W. Roberts, Vicenta Salanova, Thomas C. Witt, Sydney S. Cash, Andrew J. Cole, Gregory A. Worrell, Brian N. Lundstrom, Jonathan C. Edwards, Jonathan J. Halford, David C. Spencer, Lia Ernst, Christopher T. Skidmore, Michael R. Sperling, Ian Miller, Eric B. Geller, Michel J. Berg, A. James Fessler, Paul Rutecki, Alica M. Goldman, Eli M. Mizrahi, Robert E. Gross, Donald C. Shields, Theodore H. Schwartz, Douglas R. Labar, Nathan B. Fountain, W. Jeff Elias, Piotr W. Olejniczak, Nicole R. Villemarette-Pittman, Stephan Eisenschenk, Steven N. Roper, Jane G. Boggs, Tracy A. Courtney, Felice T. Sun, Cairn G. Seale, Kathy L. Miller, Tara L. Skarpaas, and Martha J. Morrell. Nine-year prospective efficacy and safety of brain-responsive neurostimulation for focal epilepsy. *Neurology*, 95(9):e1244–e1256, 2020. ISSN 0028-3878. doi: 10.1212/WNL.0000000000010154. URL https://n.neurology.org/content/95/9/e1244.

[14] NHS. Epilepsy treatment, 2020. URL https://www.nhs.uk/conditions/epilepsy/treatment/.

[15] Nina Omejc, Bojan Rojc, Piero Battaglini, and Uros Marusic. Review of the therapeutic neurofeedback method using electroencephalography: Eeg neurofeedback. *Bosnian Journal of Basic Medical Sciences*, 19, 11 2018. doi: 10.17305/bjbms.2018.3785.

[16] World Health Organization. *Epilepsy: a public health imperative*. World Health Organization, 2019.

[17] Alessandro Pappalardo. Xilinx/brevitas. URL https://doi.org/10.5281/zenodo.3333552.

[18] Benjamin D. Pless Stephen T. Archer Craig M. Baysinger Barbara GibbSuresh K. Gurunathan Bruce Kirk patrickThomas K. Tcheng. Seizure sensing and detection using an implantable device, US6810285B2, June. 2002.

[19] Subhrajit Roy, Umar Asif, Jianbin Tang, and Stefan Harrer. Seizure type classification using eeg signals and machine learning: Setting a benchmark, 2020.

[20] Joni Saby and Peter Marshall. The utility of eeg band power analysis in the study of infancy and early childhood. *Developmental neuropsychology*, 37:253–73, 04 2012. doi: 10.1080/87565641.2011.614663.

[21] Iqbal Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2, 05 2021. doi: 10.1007/s42979-021-00592-x.

[22] Ali Shoeb and John Guttag. Application of machine learning to epileptic seizure detection. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 975–982, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[23] Mohammad Khubeb Siddiqui, R. Morales-Menéndez, Xiaodi Huang, and Nasir Hussain. A review of epileptic seizure detection using machine learning classifiers. *Brain Informatics*, 7, 2020.

[24] James Stone and John Hughes. Early history of electroencephalography and establishment of the american clinical neurophysiology society. *Journal of clinical neurophysiology : official publication of the American Electroencephalographic Society*, 30:28–44, 02 2013. doi: 10.1097/WNP.0b013e31827edb2d.

[25] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL https://doi.org/10.1093/mind/LIX.236.433.

[26] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 65–74. ACM, 2017.

[27] Paul Vanabelle, Pierre De Handschutter, Riëm Tahry, Mohammed Benjelloun, and Mohamed Boukhebouze. Epileptic seizure detection using eeg signals and extreme gradient boosting. 04 2019.

[28] Linnea Vaurio, Stella Karantzoulis, and William B. Barr. *The Impact of Epilepsy on Quality of Life*, pages 167–187. Springer New York, New York, NY, 2017. ISBN 978-0-387-98188-8. doi: 10.1007/978-0-387-98188-8_8. URL https://doi.org/10.1007/978-0-387-98188-8_8.

[29] WHO. Mental health: neurological disorders, 2016. URL https://www.who.int/news-room/q-a-detail/mental-health-neurological-disorders.

[30] Matthew Zack and Rosemarie Kobau. National and state estimates of the numbers of adults and children with active epilepsy — united states, 2015. *MMWR. Morbidity and Mortality Weekly Report*, 66:821–825, 08 2017. doi: 10.15585/mmwr.mm6631a1.