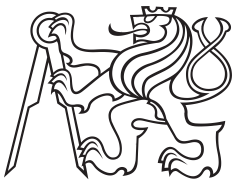
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Circut Theory

Software aplication for real time analysis of ECG signal on AI chip

Jan Lubojacký

Supervisor: doc. Ing. Patrik Kutilek, M.Sc., Ph.D.
Field of study: Medicinal Electronics and Bioinformatics
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Lubojacký Jan** Personal ID number: **492102**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Circuit Theory**

Study program: **Medical Electronics and Bioinformatics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Software application for real time analysis of ECG signal on AI chip

Bachelor’s thesis title in Czech:

Softwarová aplikace pro real time analýzu EKG signál využitím AI čipu

Guidelines:

Student will propose and implement methods for processing ECG signal and identification of high cognitive load from the ECG signal with the usage of AI implemented on an economically priced microcontroller. Proposed methods will be based on methods that are already used in clinical practice, such methods will be preferred. Student will use data that have been collected for research in how the ECG of a person changes under different levels of cognitive load.
Main tasks:
1) Use an affordable microcontroller and test it for applica tion in analysis of biological signals in real time, use of digital filtration and deployment of neural networks.
2) Test the chip for the possibility of QRS complex detection from a simulated signal.
3) Propose and implement at least one method suitable for detection of high cognitive load from ECG in real time with the usage of neural networks.
4) Propose and implement methods for representation of used data and computed parameters. Also implement methods for clear visualisation of the data and the computed parameters on PC.
5) Proposed methods will be implemented in Python or MATLAB.
6) Proposed methods implement on the AI chip and test and statistically analyze classification accuracy .
7) Compare proposed methods with available research.

Bibliography / sources:

[1] YLEN, L., WALLISCH, P. Neural Data Science: A Primer with MATLAB and Python. Academic Press; 1 edition (April 4, 2017). 368 pages. ISBN-13: 978-0128040430
[2] MOHYLOVÁ J., KRAJČA V. Zpracování signálů v lékařství. Žilinská universita 2005. ISBN 80-8070-341-8. Skriptum na CD.
[3] MADEIRO, J., Paulo, CORTEZ, P., Cesar, et all. Developments and Applications for ECG Signal Processing: Modeling, Segmentation, and Pattern Recognition. Academic Press (5 December 2018). 210 pages. ISBN-13: 978-0128140352

Name and workplace of bachelor’s thesis supervisor:

doc. Ing. Patrik Kutílek, MSc., Ph.D. Department of Health Care Disciplines and Population Protection FBME

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **03.02.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

doc. Ing. Patrik Kutílek, MSc., Ph.D. Supervisor’s signature	doc. Ing. Radoslav Bortel, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

I would like to thank
doc. Ing. Patrik Kutilek, M.Sc., Ph.D.,
Ing. et Ing. Jan Hejda, PhD. and
Bc. Ján Hýbl for their guidance and support. I would also like to thank my family for their continuous support during my studies.

Declaration

I hereby declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 18. May, 2022

. . . . . .

Abstract

This thesis proposes an application for real-time cognitive load detection from the ECG signal using convolutional neural networks and a microcontroller. Two network architectures are proposed and statistically evaluated. The neural networks were proposed in Python with TensorFlow, and the application was developed for the STM32F303RE microcontroller in C language with development tools from STMicroelectronics. The application also contains a method for detecting R-peaks and computing several parameters from the ECG signal. A PC application that receives and visualizes the results sent from the microcontroller was also developed. The proposed solution is evaluated and compared to existing research.

Keywords: cognitive load detection, convolutional neural networks, ECG, embedded machine learning, microcontrollers

Supervisor: doc. Ing. Patrik Kutilek, M.Sc., Ph.D.
Kladno, náměstí Sítná 3105

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace pro mikrokontrolér pro detekci kognitivní zátěže na základě analýzy EKG signálu v reálném čase. Pro analýzu EKG signálu byly využity konvoluční neuronové sítě. Na základě experimentů byly vybrány dvě architektury neuronových sítí. Neuronové sítě byly navrženy v Pythonu, ale samotná aplikace pro mikrokontrolér STM32F303RE je vytvořená v programovacím jazyce C za pomoci nástrojů od STMicroelectronics. Aplikace obsahuje také metodu pro detekci R kmitů a výpočet několika parametrů z EKG signálu. Součástí práce je také aplikace na počítači, která přijímá a graficky zobrazuje výsledky z mikrokontroléru. Přesnost navrženého řešení je statisticky vyhodnocena a porovnána s existujícím výzkumem v této oblasti.

Klíčová slova: detekce kongitivní zátěže, konvoluční neuronové sítě, EKG, strojové učení na mikrokontrolérech, mikrokontroléry

Překlad názvu: Softwarová aplikace pro real time analýzu EKG signálu s využitím AI čipu

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

If there is something that every one of us does every day, it is processing information. Thus, if we could measure the effectiveness of that, it would be most beneficial in many applications. There is a paper from 1988 [2], which describes a parameter, called the cognitive load, that relates to our ability to process information. Being able to measure the cognitive load in real-time reliably is applicable in various fields ranging from education and tutoring to smart medical and healthcare diagnostics or human-machine interactions.

1.2 Problem formulation

The idea behind the cognitive load theory is quite simple. There is working memory and long-term memory. When we are overwhelmed with too many perceptions, working memory cannot process all the information, the cognitive load is high, and the effectiveness of processing information is reduced.

Several physiological signals and parameters can be used to determine the state of cognitive load. In this work, we will use the ECG signal for this purpose, as research suggests that the cognitive load can be quite accurately determined from it. [6] [9]

Now how to compute the cognitive load. There are several machine learning methods suitable for this task. However, this work aims to classify cognitive load on a microcontroller, so we will be using neural networks, as there are

software libraries suitable for compressing them and implementing them on embedded devices.

In this paper, [14] data is collected by a wrist-worn sensor and sent to the cloud, where a powerful machine learning model determines the state of the cognitive load. While this works, it would be far more practical to perform on-site classification directly on the embedded device and thus eliminate the need to send the data elsewhere, saving time and energy needed in the process.

This is something that an emerging branch of machine learning called *tiny machine learning* could help solve. *Tiny machine learning* is trying to port complex machine learning models to mobile and embedded devices, where they can run with minimal power and yet be able to compute various difficult problems.

Using a microcontroller means that the final solution will be lightweight and portable, as microchips are small and can be integrated into a device that fits in a pocket. However, this also creates several difficulties that need to be dealt with. One downside is that the big and powerful machine learning models usually require much computational power. However, our model must be fast and light because microcontrollers have limited computational power and memory. In recent years, there has been much research into model architectures that have performance similar to the large machine learning models. Yet, they are small enough, so it is feasible to use them in mobile and embedded applications. [44][45]

Now, we will set goals for the practical part of this thesis. Firstly, we need to prepare the collected ECG data for training and testing. Secondly, it will be necessary to experiment with various designs to find a light, fast, and precise architecture we can employ on a microcontroller. Thirdly, we need to create a communication pipeline so that the microcontroller can correctly process a continuous stream of data and output the computed results. Furthermore, we need to analyze the results statistically to determine whether our proposed method is effective.

1.3 Images credits

Unless otherwise stated all of the images and graphs used in this work were created by the author of this thesis, either drawn by hand or with the help of the free open source software diagrams.net [55] or with the matplotlib.pyplot [53] library.

Chapter 2

Cognitive load

2.1 Cognitive load theory

Cognitive load theory comes from the late 1980s [2]. It puts the success in learning and problem solving into context with the amount of effort and resources invested in the process. Resources in this context refer to attention, decisions, working memory load, or task-related knowledge.

Long-term memory, much like in computers, in human cognition, there is a structure for storing the large amounts of information we come in contact with. This structure is called long-term memory.

Working memory is a memory segment we have for processing new information. Compared to long-term memory, it is quite limited both in capacity and time. [4]

The basis for it is that we primarily learn by acquiring schema, and in the future, we know what actions to choose because we have the schema for it. Cognitive load theory describes the correlation between the effectiveness of schema acquisition and current cognitive load. Imposing high levels of cognitive load is not very effective for learning, the reason being that human working memory is limited, and if we overload it with too many perceptions, there are not enough resources left for schema acquisition. [2]

When we are close to the limit of our working memory capacity, we enter what we would call a state of high cognitive load. This state is typically associated with stimuli that require a lot of mental efforts, such as calculations,

presentations, and memorization. On the other hand, we have a low cognitive load state. This state is associated with tasks that require minimal mental effort. For example, resting or tasks that are highly automatized.

Several attributes contribute to the generation of cognitive load. The difficulty of the materials generates an intrinsic load, and extraneous load is generated by the design of the instruction and materials. And lastly, the germane load is the amount of mental effort invested. [[3] p.1] Apart from these, another critical attribute is prior knowledge of the subject. These knowledge patterns are referred to as schemas. Possessing relevant schemas decreases the intrinsic load. For example, a math professor will have a lower cognitive load than your average first-year university student when doing calculus. [1]

2.2 Measuring cognitive load

The traditional approach uses self-rated subjective measures such as questionnaires. [5] These methods provide insight into individual perceptions and the mental state of the subject. However, they are not suitable for the real-time classification of cognitive load.

There is considerable evidence that the changes in mental state are reflected in the state of the whole body [12]. So a more recent approach is to measure various physiological signals to try and classify their cognitive load with the help of some machine learning or statistical techniques. Although we can use a range of signals to detect cognitive load, not all are suitable for real-time detection in embedded devices. We need the signals we want to use for classification in embedded devices to have several properties. Ideally, they must be easily and reliably measurable with a cheap sensor at skin contact. This thesis aims to predict the cognitive load state from the ECG signal, so we will mostly focus on cardiovascular measures. These are discussed in more detail in 4.1.3. Besides the ECG, some other signals we could use for this purpose are the GSR (galvanic skin response), eye activity, and skin temperature. [12]

State-of-the-art methods reach around 97% in classification accuracy. [12][13] Highly accurate methods usually use multiple signals, which increases the information collected, and it is more reliable than using only one

measure. [13] However, using various signals also increases the complexity of the application, and some, like the EEG, are hard to measure with wearable devices. The best method we could find that used only ECG reached 90% accuracy. [13]

2.3 Cognitive load influence

Increased cognitive load has been linked to decreased performance in learning and memorization [2][1]. It also influences decision-making. According to [42] it leads to poorer performance in arithmetic tasks, increases the reluctance to take risks, and the anchoring effect. These findings suggest that increased cognitive load hinders the ability to make decisions.

Chapter 3

Microcontroller

The embedded device used for this work was the STM32F303RET6 Nucleo-64 board. This board was lent to me for this thesis as a part of the assignment.

For our application to work correctly, we need to run it on a board that can handle it. This creates several requirements for choosing the board. We need a large enough Flash memory to fit the code and all the libraries. We also need a big enough RAM for storing all the buffers used for the ECG signal and the parameters for the neural network. Also, since we are using the X-Cube-AI package [20], we need a compatible processor. The chosen board satisfies all of these while still being a mid-range board and thus not overly expensive.

STM32 is a series of 32-bit microcontrollers from STMicroelectronics. Each board consists of several parts. The main components are the processor core, SRAM, flash memory, debugging interface, and various peripherals such as UART, ADC, etc. This chapter will cover the parts of the board that our application uses so that the reader can understand the necessary microcontroller features to grasp how the software application works.

3.1 Nucleo STM32F303RE

STM32F303RE is a mid-range, affordable board. Its main features are the Arm Cortex-M4 core processor running at 72 MHz, 512 Kbytes of Flash memory, and 80 Kbytes of SRAM, which is divided into two segments, one of which is 64 Kbytes of SRAM and the other is 16 Kbytes of CCM SRAM

(core coupled memory ram). The purpose of this is that the code can be placed in the CCM SRAM, which is faster compared to when the code is executed from flash memory. Data are then placed into the standard SRAM. This configuration can lead to a performance increase in some cases. [18][19]

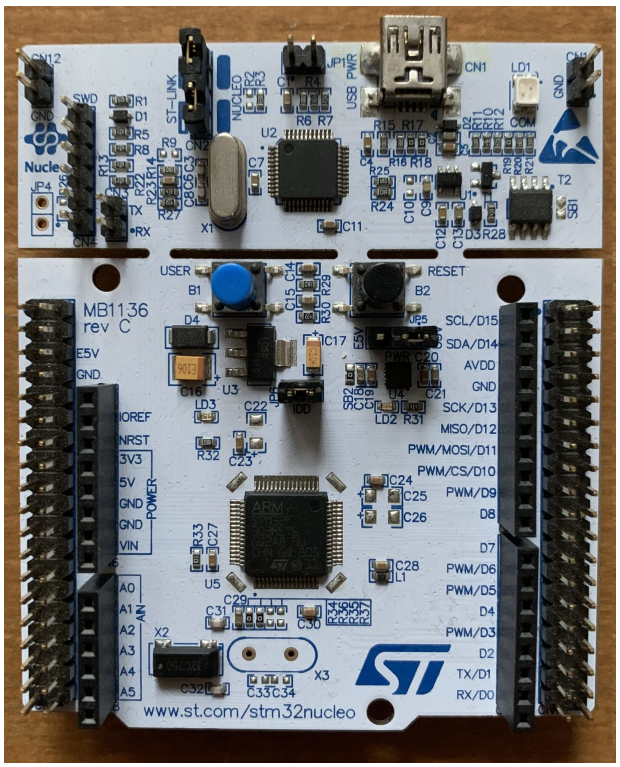


Figure 3.1: STM32F303RET6 Nucleo-64 board

3.1.1 Universal Synchronous/Asynchronous Receiver/Transmitter

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is one of the peripherals integrated on the Nucleo board. It is used for serial communication over a wire. It is composed of a TX, a wire used for sending, RX, which is used for receiving, and common ground. Synchronous communication requires one endpoint of the communication to provide a clock signal. Asynchronous communication does not require the clock signal, but both endpoints must use the same baud rate (transfer speed). STM32 USART / UART peripherals can be configured to operate in both synchronous and asynchronous modes. The application for this thesis uses the asynchronous one.

The default value for the UART communication line is HIGH. The start of a message is indicated by a start bit, which makes the signal go LOW. After the start-bit, there are 8 bits with information and, depending on the setting, also 1 parity bit, which serves as a control mechanism to check whether the message is received correctly. At the end of the message, there is a stop-bit whose value is HIGH. Only 8-bit values can be sent over UART. That means that in order to send a number, each digit has to be sent as a text character and has to be correctly interpreted and reconstructed by the receiver. [17]

3.1.2 Direct memory access

Direct memory access (DMA) is a digital logic element in the hardware architecture, and its purpose is to handle memory transfers. Without DMA, if we want to write or read some data into or from memory, it is necessary to run the instructions for it by the processor. And when the data are received continuously, a large number of interrupts are fired from the peripherals. This puts a load on the processor that slows it down, and it can even make it miss some instructions. Using DMA for this purpose significantly increases the application's execution speed. Because while DMA handles data transfer, the processor can execute other instructions. In our case, the DMA fills the input buffer with the data from the USART while the processor runs the neural network in parallel.

3.2 Development enviroment & programming the board

The development environment used to program the microcontroller was STM32CubeIDE. It contains a variety of tools in one IDE and all the necessary software needed to create applications for various STM boards and processors.

If we look at the board 3.1, we can see two regions. The right part is the nucleo itself, and the left part contains the ST-LINK. ST-LINK is a debugger and programmer for STM8 and STM32 microcontrollers. Using STM32CUBEIDE and ST-LINK, we can easily flash the program into the microcontroller. It also provides access to the onboard debugger, so it is possible to debug the application in real-time. Some handy debugging tools are the possibility to see the contents of individual buffers and variables or

the amount of flash memory and RAM used. It is also connected to the board peripherals, so one USB-mini cable is used for the communication between the board and the PC and for programming the board.

There is also the CubeMX, which is a graphical tool that allows the user to configure peripherals with a graphical user interface. It also generates the initialization code for the peripherals.

3.2.1 Hardware abstraction layer

Microcontrollers are controlled by writing bits into their registers. While this can be done manually and allows precise control of the microcontroller, writing such code is also tedious and slow. The HAL library provides functions that can perform complex tasks, such as sending data through the USART or blinking a LED, with a few function calls. It also makes the code portable between different microcontrollers. The disadvantage of this is that the code can be suboptimal in performance at the cost of portability.

3.2.2 STM X-Cube-AI

STM X-Cube-AI [20] is a package integrated into STM32CubeMX that allows easy implementation of machine learning models on microcontrollers. It contains a graphical user interface that allows the user to load the neural network into the microcontroller and analyze it. The analysis shows the architecture and all the buffers of the neural network. It also shows the RAM and FLASH memory that the model requires. It is also possible to run the model with random numbers or input provided to check if it is executed safely. This evaluation can be run on both the PC and the board.

It provides support for models exported from a variety of deep learning frameworks, such as TensorFlow Lite, PyTorch, or MATLAB. There are two runtimes available for the TensorFlow Lite models. One is the open-source TensorFlow Lite for Microcontrollers runtime, and the other is the proprietary STM32Cube.AI runtime. Although using proprietary software can sometimes create problems with compatibility, the STM32Cube.AI runtime seems to have better performance and more straightforward implementation, so that is what I used in the end. It also offers a tool for model compression. However, our models were already quantized, so we did not make any use of this.

Chapter 4

Data and preprocessing

4.1 Electrocardiogram (ECG)

4.1.1 Heart anatomy

The heart is an organ that pumps blood through the body. It is composed of four chambers, two atriums, and two ventricles.

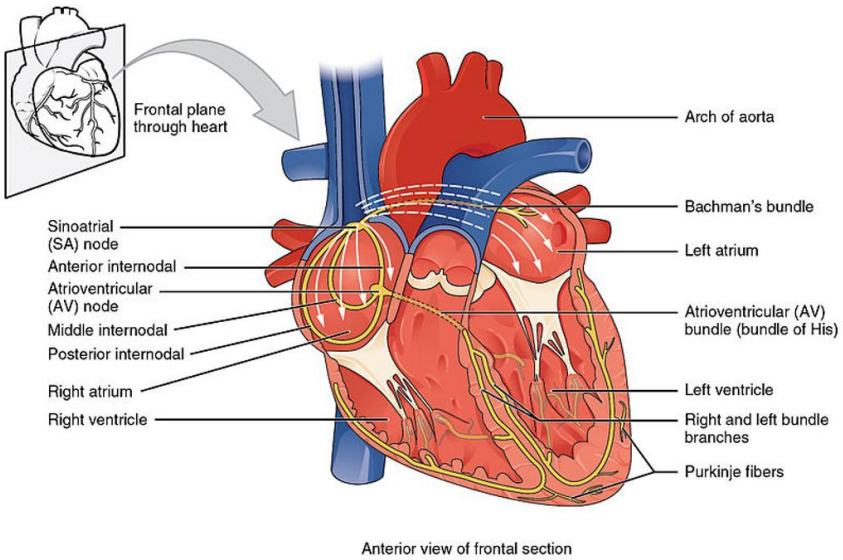


Figure 4.1: Schema of hearts anatomy, source [33]

The unoxygenated blood from the body flows from the inferior and superior vena cava, then through the right atrium and right ventricle. Finally, it is

pumped into the pulmonary arteries and the lungs. It flows through the pulmonary veins back into the heart, enters the left atrium and the left ventricle, and then is pumped into the body through the aorta.

4.1.2 ECG generation

The heart is composed of cardiac muscle cells, and like all muscle cells, they are controlled by a system of nerves. ECG is an electrical signal that describes the activity of these nerves. The signal begins in the sinoatrial (SA) node, where it causes the right atrium to contract. This denotes the P wave on the ECG graph. Then the signal passes from the atria into the ventricles through the atrioventricular (AV) node, and the ventricles fill with blood. This corresponds to the flat line between the P wave and the QRS complex. Then the signal passes through the bundle of His and into the bundle branches. Which causes the contraction of the ventricles, and the contraction of the ventricles creates the QRS complex. After that, the repolarization of the ventricles occurs, shown as the T wave. Then the atria are filled with blood, and the cycle begins again.

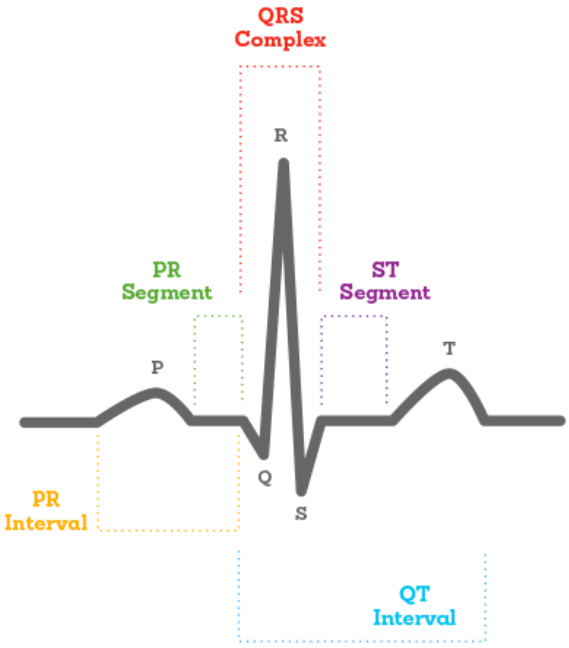


Figure 4.2: ECG, source: [32]

4.1.3 Cognitive load influence on ECG

In order to estimate whether ECG can be used to classify the state of cognitive load, it is meaningful first to analyze the signal and determine whether cognitive load influences some statistical parameters. If it does, then there is a strong possibility that our machine learning models can learn some patterns and correctly classify the signal.

Heart rate is a measure that has been proven to be quite sensitive to cognitive load. [12] Another metric that seems sensitive is the absolute mean deviation of the ECG, which is similar to the standard deviation but is more resistant to outliers. [41] This research used RR intervals and statistical features extracted from them to classify low or high cognitive load with an accuracy between 0.80-0.85. [9] Heart rate variability has also been linked to the identification of cognitive load. [8] [11]. Another ECG segment positively correlated with cognitive load is the QT interval. [11]

This shows that changes in ECG are highly correlated with cognitive load, and thus ECG analysis is a suitable method to determine the state of cognitive load.

4.1.4 Sampling frequency

If one term connects all aspects of this work, it is optimization. Thus, exploring the minimal valid sampling frequency that still carries enough information for classification makes sense. The lower the amount of data fed into the network, the lighter the network, the faster the inference. But too little information may lead to underfitting, meaning that the model will be unable to learn any patterns in the data.

From the basics of signal processing, the Nyquist-Shannon theorem states that we need to sample with at least two times the frequency of its fastest component $f_s > 2 \cdot B$ to record a signal without aliasing. With this in mind, the next question is what are the critical frequencies in the ECG signal that carry the information our model will need for its predictions.

Kohler [31] found that the QRS complex is composed of frequencies between 10 and 25 Hz. However, for this application, other frequencies are also of interest. The main components of the ECG signal, including T and P waves,

are composed of frequencies that range between 1 and 50 Hz.[30] This is also evident from the spectrum graph. 4.7

This research [29] states that 250 Hz is suitable for both time and frequency analysis, 100 Hz is acceptable for time-domain analysis, and 50 Hz is not suitable for either. Contrary to this, another research cites 50 Hz as a reasonable sampling rate for some purposes [10]. With this in mind, we ran experiments for 250, 100, and 50 Hz. Higher sampling frequencies are also used in practice. Using them for this work is not desired as 10 seconds of ECG signal sampled at 500 Hz would amount to $10 \cdot 500 = 5000$ samples. And 5000 floating-point numbers require $5000 \cdot 4 = 20$ kilobytes out of the 64 kB of available RAM for the input tensor.

4.1.5 ECG window length

Another input signal parameter that we can set is the window length. The hypothesis was that longer signal windows provide more features the model can learn, and thus, the classification accuracy should be better. 6.3

However, using a longer signal window than needed is impractical in embedded devices, as it consumes a significant amount of memory. A 30-second long ECG signal sampled at 100 Hz totals to $30 \cdot 100 = 3000$ numbers which in float format take $3000 \cdot 4 = 12$ kilobytes of memory, so 18.75 % of RAM is consumed just for the input buffer. A larger input buffer also results in a more complex neural network that takes more memory and is slower. 6.3 A real-time detection application using longer windows also means that we must compute on a moving window with overlap to get predictions more often, which adds yet more complexity.

So we need to find a good compromise between a too short signal window that does not carry enough information and an unnecessarily long one. Another parameter that puts requirements on the window length is the computation of statistical parameters. The window must be long enough to capture several beats to provide a valid estimation of heart rate and other statistical measures. Based on available research [27], 10 seconds seemed like an excellent primary candidate for the window length, so the experiments were based around that.

4.2 Preprocessing techniques

Scaling

The scale of real-world data can vary. This can cause several adverse effects during the training process of our model.[26] To give an example from this work, look at the following figure, which depicts two different ECG signals from 2 datasets.

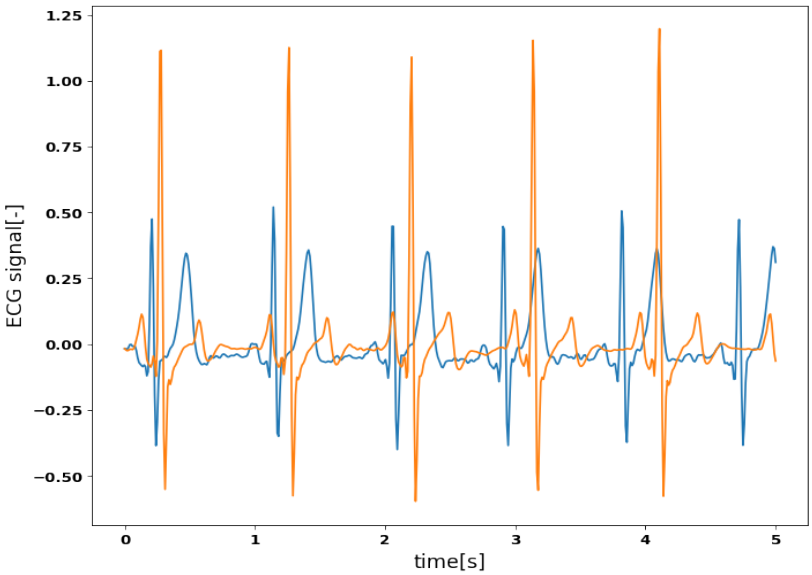


Figure 4.3: Not scaled

We can see that the data are on different scales. This can lead to poor generalization when feeding new data to the model, which is on a different scale. Another problem that occurs during the training process is that input with large values will shift the model weights a lot, but input with very small values will shift them very little [36]. We also found that scaling the data led to lower quantization errors during compression. The Light CNN model experienced a drop of 4 % without normalized data but 0 % with normalized data, the same holds for the MobileNet3 model. Also, the validation accuracy tended to oscillate a lot during training without normalization.

According to this formula, Min-max scales the input data to a range from 0 to 1. [37]

$$x_{scaled} = \frac{x - min(x)}{max(x) - min(x)} \tag{4.1}$$

Now let us look at the same two signals, but this time with min-max scaling.

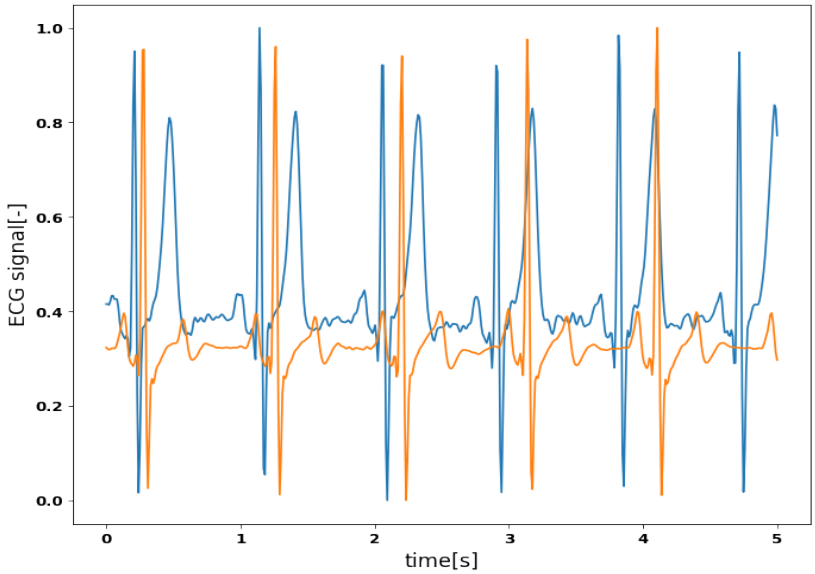


Figure 4.4: Scaled with min-max scaling

Both signals are now scaled to the same range and thus more suitable for training the model, as having all the input data on the same scale is more suitable for the gradient descent algorithm. [36]

Min-max scaling is suitable for use here because it is computationally cheap, as searching the array for min and max values is of linear complexity, and in the constrained environment of an embedded device, it is good to save computation time when possible.

Standardization

Another common technique for scaling data is to standardize them. This technique modifies the data so that the resulting distribution has a mean of zero and standard deviation of one. This again helps with training the neural network. [36] It is computed with this equation, where μ is the mean of the vector x and $sigma$ is the standard derivation of vector x.

$$x_{standard} = \frac{x - \mu}{\sigma} \tag{4.2}$$

4.2.1 Common noises in ECG

Filtering is needed because some unwanted noise almost always corrupts real-world data. Noise in signals is expected, the sensor picks up some other signal that gets mixed up with the signal we want to measure, and thus, we end up with a noisy signal. If we can determine the frequency of the noise, it can be removed with a frequency filter that suppresses the noise but allows the valuable parts of the signal to pass. Here we describe several frequent causes of noise in the ECG signal and how to remove them. As a motivation for this section, let us look at a comparison of a raw ECG signal.

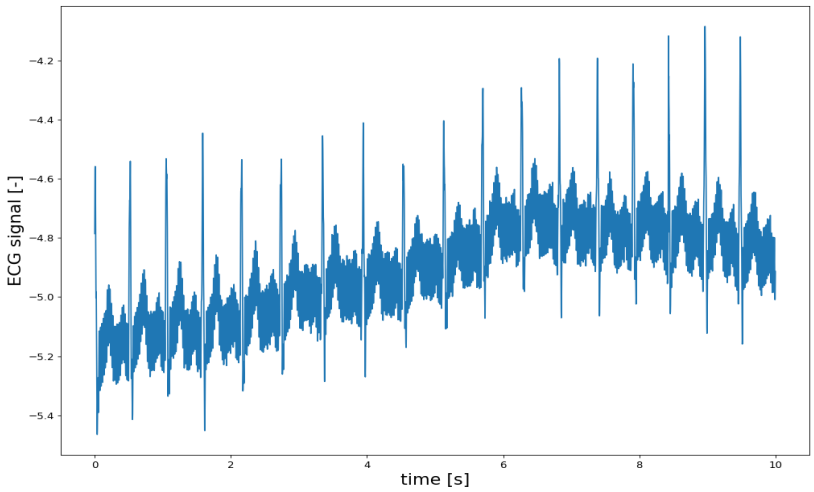


Figure 4.5: Raw ECG signal

And a filtered one.

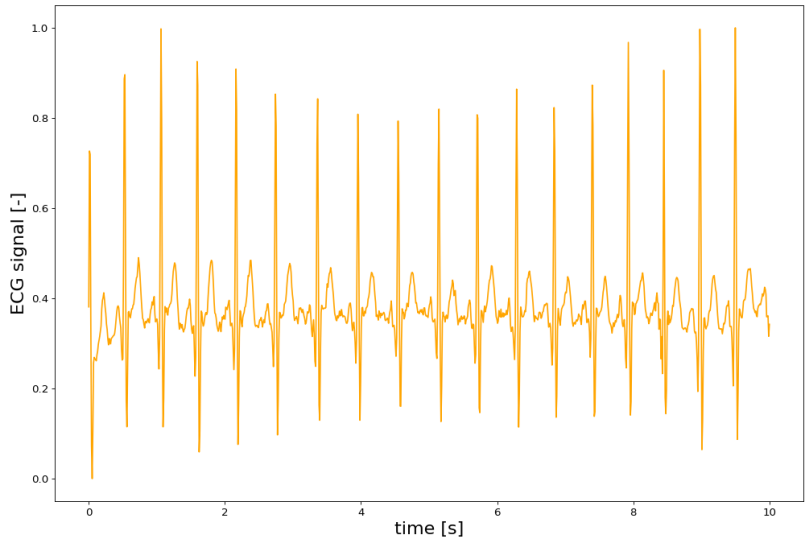


Figure 4.6: Filtered ECG signal

To identify the frequencies present in the signal, we can use the Fourier transformation, which transforms the signal from the time domain into the frequency domain. Here is a plot for our two samples.

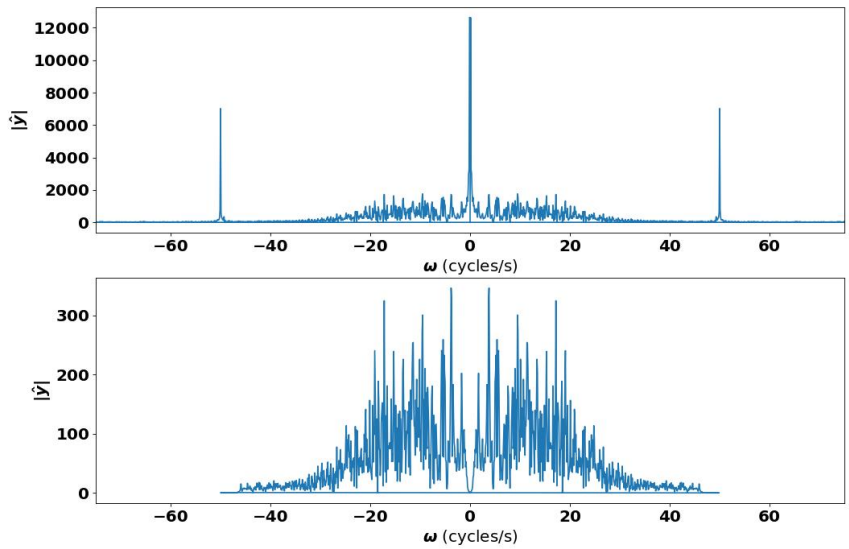


Figure 4.7: ECG spectrum, upper: raw, lower: filtered

We notice two significant sections in the unfiltered signal, present as peaks on the upper spectrum graph. The large one at zero is the DC component, and the small frequencies around it are the baseline wander/drift. This refers to a slow-moving component in the ECG that results in the signal periodically drifting up and down. It is most commonly caused by breathing, slow movements of the patient, or electro-chemical reactions where the electrodes touch the skin. The frequency of these components is usually between 0.05 and 0.5 Hz [40], which is much lower than the frequencies at which we find the essential parts of the signal, so it is possible to filter these components out with a high-pass filter without losing critical information. A common choice for the cutoff frequency is 0.5-1 Hz. (The one used for the graph was a fifth-order Butterworth high-pass with a cutoff frequency of 0.5 Hz.)

The second prominent noise is the mains hum. These are the two peaks at 50 Hz on the spectrum graph. This noise originates from the electrical grid, and its frequency will be 50 or 60 Hz, depending on the country. It can be removed with a band-stop or low-pass filter of corresponding frequency. (The one used for the graph was an eighth-order Chebyshev low-pass with epsilon equal to one.) With band-stop, it is necessary to filter the correct frequency, or we can use a low-pass with a cutoff around 40-45 Hz and filter them both.

Besides these, we can also have other noises, for example, from the contraction of other muscles. The frequencies in these types of noise overlap with the ones in the ECG signal, and filtering them is more complicated; however, they are also less common, so filtering them is not strictly necessary.

4.3 ECG filtering

Now that we know at what frequencies noises occur in the ECG signal, we need to remove them. For this purpose, we use filters. These can be analog ones implemented in the sensor or digital ones implemented in code on the microcontroller. We will use IIR (infinite impulse response) digital filters, which we will describe now.

A general IIR filter is described with the following equation [60]:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k] \tag{4.3}$$

Where the output $y[n]$ depends both on the value of the input and on the

scaled and shifted values of the previous outputs $y[n-k]$. If we apply z-transform and rearrange the equation, we can express the transfer function of the IIR filter as the ratio of 2 polynomials. [60]

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \frac{Y(z)}{X(z)} \tag{4.4}$$

And now we will look at the characteristics of two famous filter types.

Butterworth

This is the transfer function of a Butterworth filter, where Ω_c represents the cutoff frequency and N is the filter order.

$$|H(\Omega)| = \frac{1}{\sqrt{1 + (\Omega/\Omega_c)^{2N}}} \tag{4.5}$$

Butterworth filter has the smallest gain out of all the common IIR filter designs but has one big advantage. Its response is perfectly flat in the passband. [60]

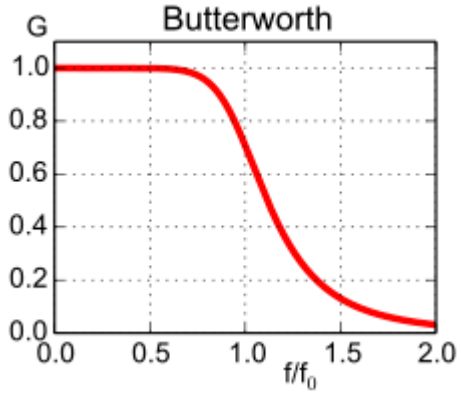


Figure 4.8: Gain of a Butterworth filter [54]

Chebyshev type I

The transfer function of a type I Chebyshev filter looks like this.

$$|H(\Omega)| = \frac{1}{\sqrt{1 + \varepsilon_p T_N^2(\Omega/\Omega_c)}} \tag{4.6}$$

$$T_N(x) = cos(N \cdot cos^{-1}(x)) \tag{4.7}$$

Chebyshev filter has a ripple in the passband, which can damage the signal; however, it also has a bigger gain, which results in a smaller transfer band and better attenuation of higher frequencies. [60]

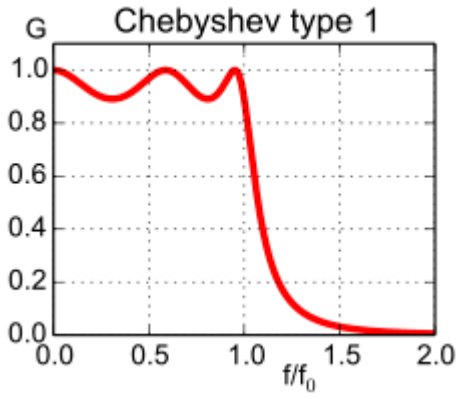


Figure 4.9: Gain of a Chebyshev filter [54]

Zero phase filtering

One disadvantage of IIR filters is that they do not have a linear phase response. This can be fixed with zero-phase filtering with the following algorithm. We filter the signal, flip it in time, filter again, and flip it back. Now we have a filter with a linear phase; however, with twice the computational complexity.

4.4 Datasets

The first step in creating a machine learning model is to collect the data we want to train the model on. This section will describe three datasets I used for training models for this thesis.

4.4.1 WESAD

WESAD (WEarable Stress and Affect Detection) is a publicly available dataset consisting of data from 15 subjects. The exclusion criteria for this study were pregnancy, heavy smoking, mental disorders, and chronic and cardiovascular diseases, and the mean age was 27.5 +/- 2.4 years, with 12 subjects being male and 3 female.

It contains data for blood volume pulse, electrocardiogram, electrodermal activity, electromyogram, respiration, body temperature, and three-axis acceleration. Data were measured using a chest sensor and also a wrist-worn sensor. The ECG signal, which is of interest for this thesis, was measured with a RespiBAN Professional chest device with a sample rate of 700 Hz with a standard three-point ECG. Although this dataset is primarily meant for stress detection, it is also suitable for cognitive load detection.

The signals were recorded for three classes, baseline, stress, and amusement. Out of which, we used the baseline and stress class for this work. The baseline condition consisted of subjects relaxing while sitting or reading a magazine. Stress conditions consisted of public speaking and mental arithmetic tasks, which are tasks known to reliably cause cognitive load. [15] To help determine the labels of the data, subjective self-reports were also collected after each measurement. [16]

4.4.2 CLAS

CLAS (A Database for Cognitive Load, Affect and Stress Recognition) is a multimodal data set consisting of electrocardiography (ECG), plethysmography (PPG), electrodermal activity (EDA), and accelerometer data. There are data from 62 subjects in this dataset, 45 were men and 17 were women. Fifty-eight of these subjects were between 20 and 27 years old, and the remaining subjects were older.

This data set aims to improve human-machine interaction and offers data recorded during various tasks for the automatic classification of emotional and mental states. ECG data were recorded with the Shimmer3 ECG Unit with a sample rate of 256 Hz and a resolution of 16 bits per sample. Data were recorded during several different tasks. The data we used for the cognitive load class were recorded under three different stimuli, math problems, Stroop test, and logic problems. We used the baseline data for the negative class, which were recorded when there were no stimuli. The dataset paper achieved a precision of 78.2 % on high/low concentration classification with an SVM classifier using the combination of ECG and GSR signals. [25]

4.4.3 BIOMECH

We call this dataset BIOMECH because it was kindly provided to me by the Laboratory of Biomechanics and assistive technologies at the Faculty of Biomedical Engineering of the Czech Technical University. The BIOMECH dataset is the smallest of the three datasets, with only five subjects. It contains data for two classes, the baseline and the high cognitive load. The high cognitive load was induced with the Stroop test. The signal was sampled at 500 Hz.

Because this dataset is much smaller than the other two datasets, it also functions as a control dataset. A good model should have similar performance across all three datasets. If the model performs well on the bigger dataset but fails on the small one, it indicates that the features learned by the model are not generally applicable to different data.

Chapter 5

Designing a neural network

5.1 Neural network fundamentals

Artificial neural networks have received a lot of attention in the last 25 years. They represent a complex function that receives an input and outputs an output. The strengths of neural networks are that they are capable of parallel distributed processing, which speeds up the computation, they are capable of nonlinear mapping, and they learn through training, which makes them able to adapt to many different problems.

Single neuron

The fundamental building block of neural networks is a single neuron. This function attempts to mimic biological neurons. It has many inputs and one output. A single neuron is composed of a perceptron which is a linear, binary classifier that multiplies each input x_i by weight w_i and scales the product by a bias b (representing the activation threshold). The other part of a single neuron is a non-linear activation function that maps the perceptron activation to an output. The equation for the output comes out as. [34]

$$y(t) = f\left(\sum_{i=1}^{\infty} w_i x_i - b\right) \tag{5.1}$$

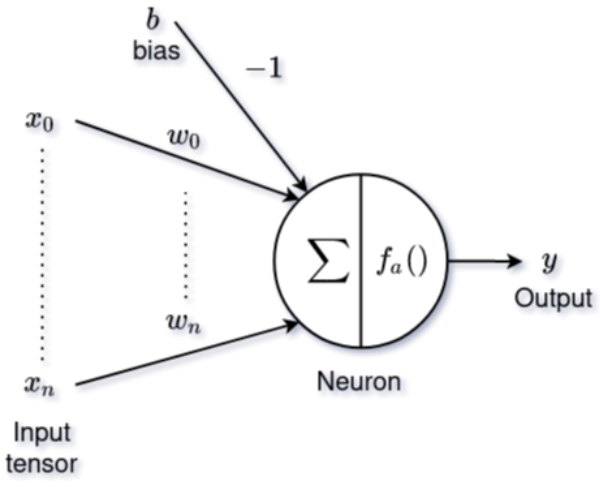


Figure 5.1: Schema for one neuron [34]

Deep neural networks

If a simple neuron mimics the function of a human neuron, then deep neural networks aim to simulate the brain's nervous system. Deep neural networks are made up of layers. The most straightforward layers are formed of neurons like 5.1. Such layers are called dense layers.

There are two main types of ANN. The first one is the feedforward network. Here, the network is composed of layers. Each layer receives an input tensor and maps it into an output tensor which is passed to the next layer until the end of the network is reached. All models used in this work are of this type. The second is a recursive or feedback network. In these, the outputs can flow backward in the network, becoming inputs for neurons in the same or previous layers. [34]

5.2 Learning of neural networks

At first, the weights are initialized randomly. Then an input is passed to the network, which maps to some output. The loss function then calculates how different the network output is from the desired value, returning small values for correct and large values for wrong predictions. The learning process is then formulated as an optimization problem that minimizes the loss function across all training samples by updating the weights and biases of the network.

The output $\mathbf{y}$ of a network with input $\mathbf{x}$ can be described as:

$$\hat{y} = a^L(\mathbf{W}^L \cdot a^{L-1}(\mathbf{W}^{L-1} \dots a^1(\mathbf{W}^1 \cdot \mathbf{x}))) \text{ [34]} \tag{5.2}$$

Where $\mathbf{W}^l$ denotes the weights and biases in each layer, a denotes the activations and the index. The backpropagation algorithm calculates the derivatives of weights and activations layer by layer with the chain rule.

$$\frac{\partial L(y-\hat{y})}{\partial \mathbf{W}^l} = \frac{\partial L(y-\hat{y})}{\partial \hat{y}} \frac{\partial y}{\partial y^{L-1}} \frac{\partial y^{L-1}}{\partial y^{L-2}} \cdots \frac{\partial y^{l+1}}{\partial y^l} \frac{\partial y^l}{\partial \mathbf{W}^l} \text{ [34]} \tag{5.3}$$

And the network is trained with a gradient descent algorithm that updates the weights and biases like this, where ν represents the learning rate.

$$\mathbf{W}^l = \mathbf{W}^l - \nu \frac{\partial L(y-\hat{y})}{\partial \mathbf{W}^l} \text{ [34]} \tag{5.4}$$

5.3 Convolutional neural networks

Convolutional neural networks are inspired by biological structures of the visual system. They are popular in computer vision and image recognition because they perform exceptionally well with image data. This ability comes from the convolutional and pooling layers that extract patterns from the data. This section aims to give an overview of the commonly used layers in convolutional neural networks.

Convolution

Let us look at the structure of the most significant block in a CNN, and that is the convolutional layer. There is an input signal, often an image or other tensor data. In our case, it is a one-dimensional tensor with an ECG signal. This input is then convoluted with the learned convolutional filter, and the output of this operation is called a feature map.

This layer has several parameters that we can set, including the size of the convolutional kernel, the stride, which is how many samples we move the

filter in each step, and the number of filters. On the image, we have a 9x1x1 input tensor, 3x1x2 filters, and a 7x1x2 feature map. The filter size is three, there are two filters, and the stride is one.

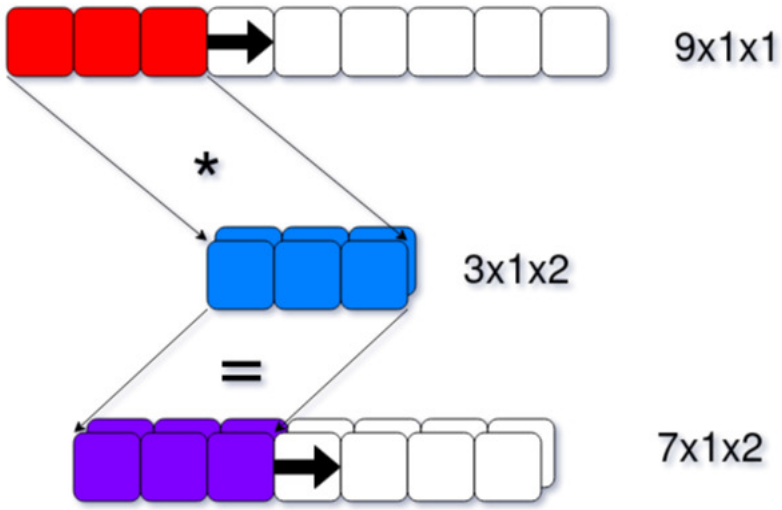


Figure 5.2: Convolution

Separable convolution

Separable convolution is a particular convolutional layer designed to compute the result with fewer operations, thus speeding up the computation. This makes them suitable for use in environments with limited resources; for example, they are used in the MobileNets models [44].

The concept comes from matrix factorization. From linear algebra, we know that if we can, for example, factor a matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$ into $\mathbf{B} \in \mathbb{R}^{3\times1}$ and $\mathbf{C} \in \mathbb{R}^{1\times3}$ and then calculate the dot product as $\mathbf{X} \cdot \mathbf{B} \cdot \mathbf{C}$, the result will be the same as if we calculated it as $\mathbf{X} \cdot \mathbf{A}$. To multiply $\mathbf{X}$ and $\mathbf{A}$, nine multiplications are needed. However, for $\mathbf{X} \cdot \mathbf{B} \cdot \mathbf{C}$, it is only six.

However, not all matrices can be factorized. For this reason, another approach is used, called depthwise separable convolution. This method consists of two steps. The first is the depthwise convolution, where the convolution filter acts separately on all the input channels. The second is the pointwise convolution, which uses 1×1 filters to compute the final feature map by convoluting the channels. [39]

Normal convolution takes an input tensor $L_i \in \mathbb{R}^{h_i \times w_i \times d_i}$ and applies

a convolutional filter $K \in \mathbb{R}^{k \times k \times d_i \times d_j}$ to produce an output tensor $L_j \in \mathbb{R}^{h_i \times w_i \times d_j}$. This operation has a computational cost of $hi \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k$. With depthwise separable convolution, the cost is only: $h_i \cdot w_i \cdot d_i(k_2 + d_j)$ with minimal reduction in accuracy. This provides a speed-up of 8 to 9 times[44].

Pooling

Besides convolution, other crucial parts of CNN are the pooling layers. These are placed after feature maps, and they downsample the feature map and extract useful information from it. This condenses the information and makes the networks smaller and faster. Like the convolutional layer, a moving filter computes the output. We specify the size, stride, and type of the pooling filter. There exists max-pooling, which takes the largest value from the filtering window, and average-polling, which takes the arithmetic average of these values.

Activations

Activations are placed after convolutional or dense layers. Their purpose is to map the layer activation with a nonlinear function, and the function decides whether the neuron is "activated" or not. An activation function must be continuously differentiable for the backpropagation algorithm to work. There are many activation functions, but ReLU (rectified linear unit) is the most widely used (it is not differentiable at 0, but this does not create problems.) It looks like this

$$f(x) = max(0, x) \tag{5.5}$$

performs well in many cases and is currently very popular. It is also used as a building block for more advanced activation functions. For example, the hard-swish (HS) non-linearity

$$f(x) = x \cdot \frac{ReLU_6(x+3)}{6} \tag{5.6}$$

Which is used in the MobileNetV3 [45]. $ReLU_6$ is similar to normal ReLU, but it is capped at 6.

$$ReLU_6 = min(max(x, 0), 6) \tag{5.7}$$

Another common activation functions is the sigmoid activation function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5.8}$$

Sigmoid works well for binary classification because it is bounded to an interval $[0, 1]$. For this reason, it can be used as the final activation in binary classification, outputting 0 for a negative and 1 for a positive class. [34]

Batch norm

The importance of normalizing input data before we feed it to the network has already been discussed in the preprocessing section. Batch norm is added to the network as another layer, and it is used to normalize data between layers. Like normalizing the input data, batch normalization helps the convergence of gradient descent by normalizing each feature map. There is some discussion on whether it is better to place it before or after the activation layer. We tested both on the Light CNN model, and it performed better when placed before activation. [36]

5.4 Compression

Model compression techniques are an essential part of this work. As a motivation, here is an uncompressed neural network (Light CNN model, 100 Hz, 10-second window, this analysis was done by the STM32 X-Cube-AI tool.)

Complexity	575371 MACC
Flash size	46.19 kB (out of 512 kB present)
RAM	51.91 kB (out of 64 kB present)

Table 5.1: Uncompressed model size

And the same neural network but now fully compressed.

Complexity	559980 MACC
Flash size	12.07 kB (out of 512 kB present)
RAM	16.66 kB (out of 64 kB present)

Table 5.2: Compressed model size

We can observe a 3.8-times reduction in flash size and a 4.3-times reduction in RAM usage from these figures. For this model, this makes a difference between an unusable model, which takes 81 % of RAM, not leaving enough memory for other parts of the application, and a compressed model, which takes only 25 % of RAM. So, in this case, compression is not only helpful but also necessary.

The goal of compression is to reduce the size and increase the speed of the model with minimal effects on its performance. Here, we will describe two techniques used for this purpose.

5.4.1 Weight pruning

Neural networks are usually very large with a large number of weights. The idea behind pruning is that some of these weights learned are of small values and, as a result, do not influence the network's decision as much. We can remove these weights without significantly damaging the performance of the network. The pruned network with fewer weights is not only smaller but also faster because, with fewer weights, the number of operations needed for inference is reduced. The TensorFlow documentation [43] states that this can compress the model size up to 6x with minimal losses of accuracy. A recommended approach is to first pre-train the whole model first and then use pruning, and after that, fine-tune the pruned model with normal training. [35]

5.4.2 Quantization

Quantization is a well-known compression technique. One use for it is image compression, where it is used to reduce the number of bits used for color representation, decreasing size but also resolution. It has a similar use for the compression of neural networks. Where it is used to reduce the size of the weights and computations from 32 bits to 8 bits, this provides several

benefits. First, the most obvious one, 8 bits, is four times less memory than 32 bits, so quantized networks take less space. Another one is speed. With less memory access and simpler computations, the inference time decreases. Easier computations also imply a reduction in energy consumption, which means prolonged battery life in embedded devices.

The idea of 8-bit quantization is to map the 32-bit floating-point tensors into tensors composed of 8-bit values multiplied by a scale factor.

$$\hat{x} = s_x \cdot x_{8-bit} \approx x_{32-bit} \text{ [24]} \tag{5.9}$$

This approximation, of course, introduces some errors in the calculations. This error is called quantization noise. Two primary causes of quantization noise are rounding and clipping errors, and they depend on the q_{min} and q_{max} values. The rounding error arises from the weights being rounded further from their original value and increases with the difference between q_{min} and q_{max}. The clipping error comes from weights with values outside the quantization bounds being quantized to the q_{max} or q_{min} value. [24] In our case, we found the error to range from 0 to 5 % loss of accuracy, which is very much acceptable compared to the benefits it brings.

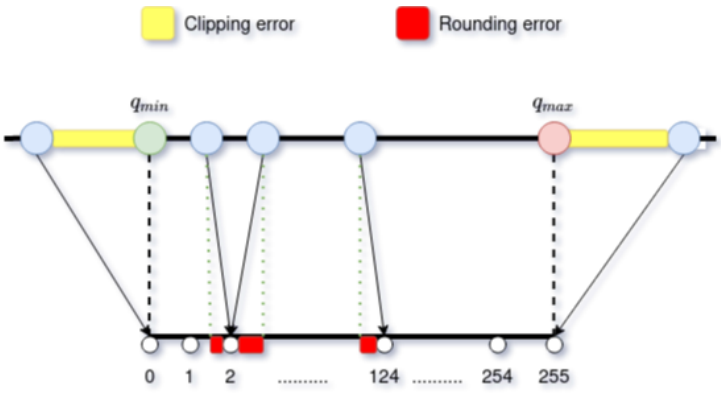


Figure 5.3: Quantization errors

$$\epsilon_{total} = \sum_{data} \epsilon_{clip} + \epsilon_{round} \tag{5.10}$$

There are two options for using quantization. One is the post-training

quantization, which takes place after the model is trained. This is faster and easier to code. However, it hinders the model accuracy more. The other method is quantization-aware training, which happens during training and is better for model accuracy. We opted for post-training quantization, as it is simpler. Also, our models did not experience any significant drop in accuracy from quantization, so we decided that there was not much benefit to gain from using quantization-aware training. Our implementation of post-training quantization is adapted from the official TensorFlow tutorial and the quantization documentation.[23]

5.5 Evaluation metrics

To evaluate the performance of our model and understand its behavior, we employ several statistical metrics. First, we test the model on data reserved for testing and count all the possible combinations between the ground truths and predictions.

TP: true positives, samples correctly classified as positive

TN: true negatives, samples correctly classified as negative

FP: false positives, negative samples misclassified as positive

FN: false negatives, positive samples misclassified as negative

These values are often displayed in a confusion matrix for visualization purposes, and we can also compute several other parameters from them.

Accuracy

How good the model is at predicting the correct classes.

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{5.11}$$

Sensitivity (Recall)

How good the model is at identifying positive samples. The lower it is, the bigger the chance the model will mistake a positive sample for a negative one.

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.12}$$

Specificity

How good the model is at identifying negative samples. Analogous to sensitivity.

$$Specificity = \frac{TN}{TN + FP} \tag{5.13}$$

Precision

The rate of true positive prediction to all positive predictions can be seen as a measure of the quality of positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{5.14}$$

F-score

F score is another measure of test accuracy calculated from the precision and sensitivity.

$$Fscore = 2 \cdot \frac{Precision \cdot Sensitivity}{Precision + Sensitivity} \tag{5.15}$$

Chapter 6

Experiments and evaluation

6.1 Proposed architectures

Light CNN

The first model tested was a custom convolutional neural network. We spent a significant amount of time tuning the optimal number of layers, the number and size of convolutional kernels, and various other parameters. The resulting network is this one.

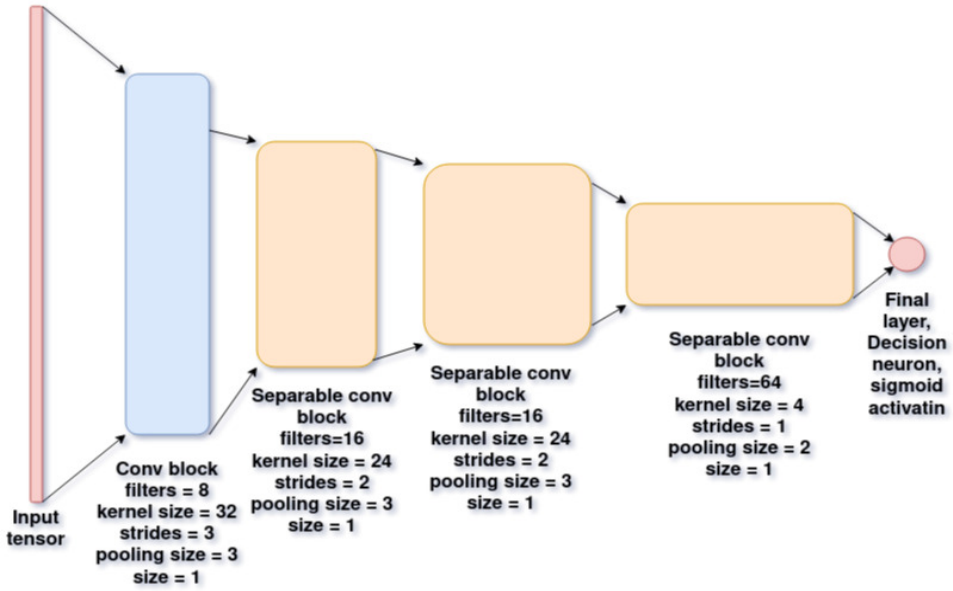


Figure 6.1: Light CNN

It contains two types of blocks, the convolutional and the separable convolutional. The convolutional block is composed of 4 layers in the following order, 1D convolution (with the number of filters, the size of filters, and the filter stride), batch normalization, the ReLU activation function, and 1D MaxPooling (with pool size and stride). The separable convolution blocks are similar, but instead of regular convolution, they use the separable convolution, and instead of max pooling, they use average pooling. Also, after the last separable convolution block, there is a 0.3 dropout, then the feature map is flattened and connected to the final neuron with sigmoid activation.

Even though it is generally less accurate than the more advanced mobile net model, it holds one advantage over it. It takes less RAM.

MobileNet-Tiny

MobileNet has 3 versions, V1, V2 and V3. Out of these, the V3 version is the newest one with the best performance. The V3 paper [45] presents two models, MobileNetV3-Large and MobileNetV3-Small. Because they are aimed at mobile devices, these models are still too large for the microcontroller. In this work, we scaled down the smaller V3 model, and the resulting architecture is dubbed the MobileNetV3-Tiny.

Compared to the MobileNetV3-Small, the MobileNetV3-Tiny has a lower number of bottleneck layers and a lower number of filters in each layer.

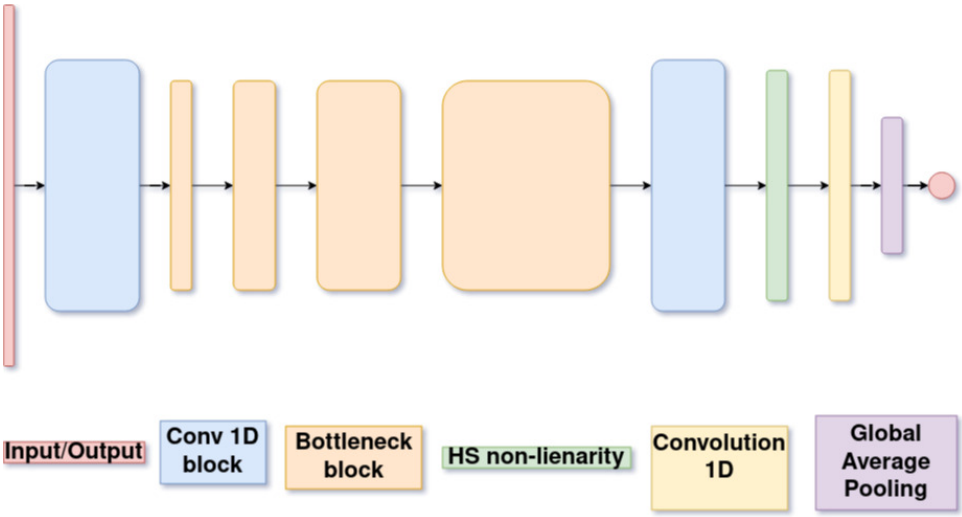


Figure 6.2: MobileNetV3-Tiny architecture

MobileNetV3-Tiny layers
Conv 1D block: filters 8, kernel sz 3, stride 3, pooling sz 3, nl HS
Bottleneck: filters 8, kernel sz 3, width 8, stride 2, squeeze True, nl HS
Bottleneck: filters 10, kernel sz 3, width 16, stride 2, squeeze False, nl HS
Bottleneck: filters 10, kernel sz 3, width 32, stride 1, squeeze False, nl HS
Bottleneck: filters 12, kernel sz 5, width 64, stride 1, squeeze True, nl HS
Conv 1D block: filters 5, kernel sz 1, stride 1, pooling sz 3, nl HS
HS non-linearity
Convolution 1D: filters 10, kernel sz 1
Global Average Pooling 1D
Output layer (one neuron with sigmoid activation)

Table 6.1: MobileNetV3-Tiny architecture

There are two main structures in the MobileNetV3 diagram, the Conv 1D block, and the Bottleneck block. These are the same blocks as in the original paper. Conv 1D block consists of a convolutional layer, batch norm layer, and a nonlinearity, either ReLU (RE) or hard swish (HS). Bottleneck blocks are the core structures introduced in the paper. The V3 version [45] combines the inverted residual blocks from MobileNetV2 [44] with a squeeze and excitation layers.

Feature model

This model is inspired by this paper. [28] They did not use the entire ECG signal as an input for the network but only features extracted from it. For this purpose, they proposed a straightforward dense network. This network is elementary and does not take much memory, but it depends on the successful detection of R-peaks. Thus it must be paired with a reliable peak detection algorithm.

Four of the computed features 6.5 were used as the input data. The network was a simple dense, deep neural network with 4 input neurons, 4 hidden layers with 20 neurons and ReLU activation, and an output neuron with sigmoid activation, totaling 1,381 parameters.

	2.5	5 s	10 s	20 s
Treshold R peaks [%]	71	68	68	69
Neurokit R peaks [%]	69	70	70	71

Table 6.2: Classification accuracies for Feature model

Although the accuracy of this model is significantly worse than the other two, it shows a few interesting things. Firstly, since similar accuracy was reached with both detection methods, the simple R peak detection method is not that bad even when compared to a more advanced, computationally significantly more complex method. Secondly, it is much less computationally expensive because it only uses computed features and not the whole signal. This means that this solution could be used even in low-end microcontrollers with only a few kilobytes of RAM.

6.2 How to train a network

Preparing the data

The first part of training a neural network is to prepare the data. The network itself can be perfectly designed, but if we present it with bad data, it might behave entirely differently than expected. Therefore, selecting which data we train the network on is essential because it will decide what features the network will learn.

Because this work worked with three datasets, we could create more complex experiments and create models that could generalize as well as possible. The WESAD dataset, which contains classes for baseline and stress, seems to be especially easy for the networks. Even really simple architectures achieved high accuracy rates on this dataset. The BIOMECH dataset contains only two classes, one for baseline and one for cognitive load, recorded during the Stroop test. BIOMECH is small compared to the other two datasets, so it functions well as a control dataset. During the experiments, if a model reached more than 90 % on WESAD yet under 60 % accuracy on BIOMECH, we can conclude that the model learned some patterns that are perhaps not generally applicable. The problem turned out to be the CLAS datasets. Initially, neutral and baseline recordings were used for the negative class, and Stroop test, IQ test, and math test recordings were used for the positive class. When we tried training the models separately on each dataset, it turned out that the accuracy was significantly poorer on the CLAS dataset than on the other two.

What helped was to use only the baseline for the negative class. It turned out that the problem was mainly caused by the neutral class, which was

recorded during 30s breaks between the high cognitive load tasks. As a result, these samples were often classified as positive class and disrupted the accuracy of the models. It is debatable whether larger, more advanced networks could learn to classify these samples correctly or if such subtle changes in such short intervals can even be determined only from the ECG signal.

After this modification, the models achieved significantly better results when trained on all the datasets combined while still preserving high accuracy on each dataset individually. This improvement leads us to believe that training on this modified dataset creates models that can generalize better.

The majority of the preparation was done with the help of a python script. The script allows the user to set the desired sample rate, window length, and whether the signal should be preprocessed.

Equalizing classes

Equalizing classes was needed because the number of samples from different classes was not balanced. Classes not being balanced is a common problem in machine learning, and it causes the model to lean towards the more common class heavily and classify nearly all the samples as one class. Dropping some samples from the more common class means that the model can not use the prior probability of classes for learning, and thus, it helps with generalization.

Training environment

The experiments were run with the TensorFlow [59] library, version 2.7. The models were trained with Google Colab [58], a free web-based IDE for Jupyter notebooks. In the form of Jupyter notebooks, it provides access to virtual machines with powerful GPUs for training the models and running all the experiments.

The program for training the models is written in a python notebook. It contains several commented cells which go through the training process. The datasets are loaded and preprocessed. Then the model that will be trained is selected. There is a method for estimating the optimal learning rate, which is then passed to the Adam optimizer. [57] Two methods for training are present, a normal one and a one used for training-aware pruning. There is

a method for evaluating the trained network on the test dataset, and these scores are displayed. Then the trained model is converted both with and without the 8-bit quantization, and it is re-evaluated to see whether the quantization hindered the model's performance.

Training

We use a binary cross-entropy loss for the loss function, a common choice for binary classification problems. [47]

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{6.1}$$

We trained all the models with a batch size of 64. An often-used value [38].

We also used early stopping. With early stopping, we select a target accuracy we want the model to reach and stop the training once it does. This helps against overfitting and ensures we end up with a model that has high accuracy on the validation set. [52]

Learning rate

Learning rate is an influential hyperparameter in training a machine learning model. When the model trains with gradient descent, it calculates the direction to some local optimum, and the learning rate specifies the size of steps it takes in that direction. Too small a learning rate might make the model train too long and not make any significant progress. On the other hand, if the learning rate is too big, it might make the model take very long steps and struggle to find the optima. The optimal training rate lies between these two extremes and lets the model find a good local optimum fast. However, it varies depending on the model used. Now let us look at a simple method used to estimate the optimal learning rate for each model. It is a simple method that takes only a few lines of code and a couple of plots.

First, we train the model for 100 epochs. In each epoch, the learning rate increases starting from $1 \cdot 10^{-4}$ (which would be too small for most models) up to 0.2 (which, in most cases, would be too large). Then we plot the loss for each epoch, and the results are these graphs with a bowl shape where the optimal learning rate lies at the global minimum of the lower graph. In this

case, the model reaches optimum around epoch 30, and then it decreases. In this case, this happens around epoch 26, which corresponds to the learning rate of $6.8 \cdot 10^{-4}$ and that is what I will use to train this model. [21][22]

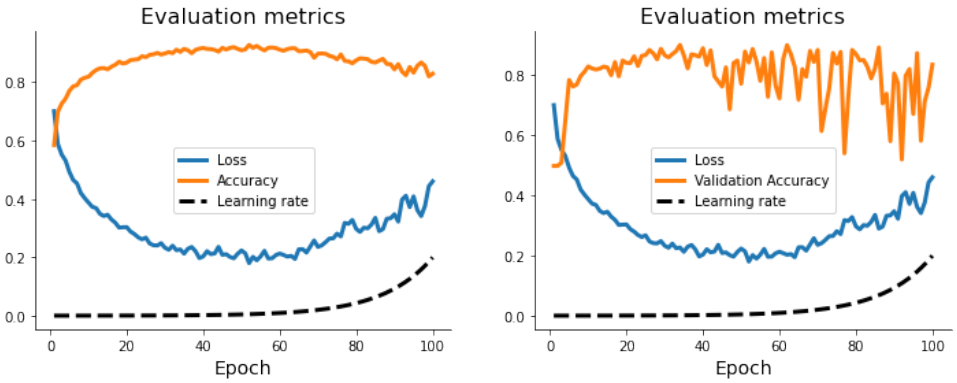


Figure 6.3: Graphs for determining learning rate

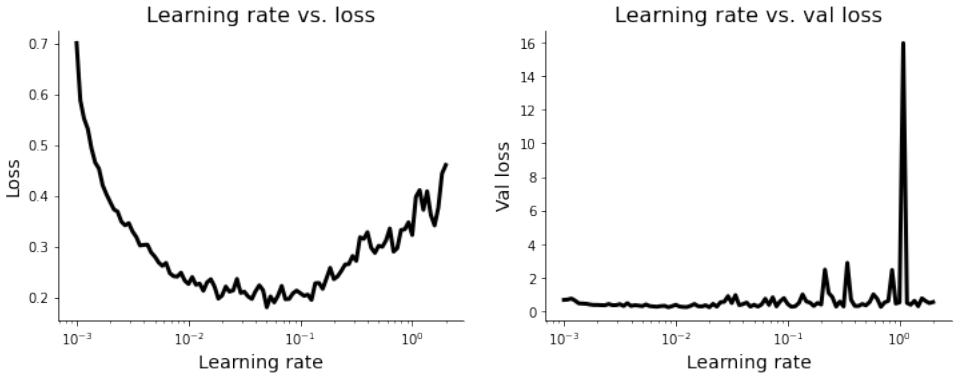


Figure 6.4: Graphs for determining learning rate

It is also interesting to plot similar plots for validation accuracy and validation loss. If the validation loss and accuracy oscillate a lot in some regions (in this case, anything after episode 40), we found that the corresponding learning rates will not be very stable for training, and the validation accuracy will tend to oscillate between epochs.

This estimated learning rate is then passed to the Adam optimizer as a starting point. Adam is a popular algorithm that improves the standard stochastic gradient algorithm. [57]

6.3 Experiments

Because the models used in this work are small and training them takes 10 minutes, it was possible to use cross-validation to test many different configurations and hyperparameters. The evaluation was done on the testing part of the combined dataset.

Memory requirements and speed of models

In almost all of the experiments, there are mentions of smaller input tensors resulting in smaller and quicker models, so here is a direct comparison between the input size (in numbers) and the model memory requirements, complexity, and speed. These were measured with the STM32-X-CUBE-AI tool [20], and the timing was done on the microcontroller with a timer.

	500	1000	2500
Ligth CNN flash [kB]	6.88	15.34	27.69
Light CNN RAM [kB]	6.93	16.66	45.96
LightCNN complexity [MACC]	216,356	559,980	1,594,980
Light CNN inference time [ms]	33.8	79.6	NaN
MbNetV3 flash [kB]	5.34	5.34	5.34
MbNetV3 RAM [kB]	11.4	21.1	50.46
MbNetV3 complexity [MACC]	273,237	542,333	1,356,237
MbNetV3 inference time [ms]	41.3	74.9	NaN

Table 6.3: Model memory requirements and complexity vs input tensor size, NaN - application did not fit into memory

Different windows

Here are the results of experiments with different windows. This experiment was done with the filtered dataset and a 100 Hz sample rate.

	2.5 s	5 s	10 s	20 s
Ligth CNN accuracy [%]	93	**94**	87	83
MobileNetV3 accuracy [%]	93	**94**	**94**	93

Table 6.4: Model accuracies for window length

From the table, we can see that the Light CNN model performs better with shorter windows, and the MobileNetV3-Tiny does well with all the windows.

These results disprove the initial hypothesis. Nevertheless, they are good because using a shorter window is preferable to a longer one, as it saves memory, increases inference speed, and produces predictions more often.

The tests for longer sampling windows were not ideal because the datasets contained many short recordings. As a result, in the 20-second, most positive samples were from one dataset, and most of the negative were from another. This disbalance creates worries about the model learning some non-general dataset-specific features. For this reason, longer sampling windows were not explored.

So without more data available, the 5-second window for both the MobileNet and the Light CNN seem like reasonable results to continue with. Using a shorter window also means more training data will be available, and the more data we have available for the model, the better the model can learn its features.

The accuracy still holds even with a very short 2.5 seconds window. While using shorter windows provides already mentioned advantages in terms of speed and memory requirements, it is not very good for R-peak detection. To calculate heart rate and other parameters, we need to catch at minimum 2 peaks. Now let us say that there is an R-peak exactly in the middle of the window. That would mean that the interval between beats must be shorter than 1.25 seconds which means that the system could fail to catch heart rate below $\frac{60}{1.25} = 48$, which is a low, but perfectly possible heart rate., for this reason, the 5-second window is preferred, where the minimum is $\frac{60}{2.5} = 24$, which is way below the healthy range. [46]

Different sampling rates

Here are the results for different sampling rates. These different frequencies were tested with filtered data and a 10-second window.

	50 Hz	100 Hz	250 Hz
Light CNN accuracy [%]	84	87	**88**
MobileNetV3 accuracy [%]	90	**94**	93

Table 6.5: Model accuracies for window length

The optimal sampling rate seems to be 100 Hz, confirming the hypothesis based on available research that a sampling rate of 100 Hz should be a good

compromise between memory requirements and the model's accuracy.

Moreover, we can conclude that with these models, a further increase in sampling rate to 250 Hz does not provide benefits that would justify the increased memory requirements and slower computation speeds on the larger input signal. Also, with the microcontroller available for this work, models of this size do not leave enough space for other parts of the application.

Furthermore, when the sampling rate was reduced to 50 Hz, the model's accuracy dropped, but not drastically. This indicates that lower sampling rates could be used for smaller-sized models in environments with even stricter memory requirements.

Scaling vs. normalization

There seems to be no difference between using either normalization or scaling. However, scaling is preferred for deployment in embedded devices because it uses fewer operations. These experiments were performed on the unfiltered dataset with the Light CNN model.

	Normalized	Scaled
Light CNN [%]	70	70

Table 6.6: Accuracy on unfilterd data with standardization and scaling

Preprocess or not?

Why preprocessing and cleaning the data makes sense has already been explained. Now let us quantify the advantages it provides in the context of our experiments. These experiments were done with a 100 Hz sampling rate and a 10-second window.

We thought it sensible to scale it with min-max scaling even when testing on raw data. This operation is not costly, and it prevents the model from learning some dataset-specific features, such as DC offset or min-max range, which can be different for each recording. If this reduces accuracy, it most likely means that the model previously used some of these features, and thus, it does not hurt the model but helps with generalization.

	Not scaled	Scaled
Light CNN [%]	**73**	70
MBV3-Tiny [%]	**87**	83

Table 6.7: Accuracy on unfilterd data

When training on only the WESAD and BIOMECH datasets filtering the data essentially did not matter because the data was already without much noise. However, with the CLAS dataset added to the training data, preprocessing the data improved the accuracy of the models by a significant margin. The cause for this is most likely the CLAS dataset containing a lot more noise, mainly from the mains hum and baseline drift. Filtering removes these noises, and with cleaner data, the models can better learn the correct patterns needed for classification.

	Unfiltered	Filtered
Light CNN [%]	70	**88**
MBV3-Tiny [%]	83	**94**

Table 6.8: Accuracy on unfilterd vs filtered data

6.4 Evaluation on microcontroller

Based on these experiments, two model configurations were chosen as the most promising. These were further evaluated directly on the microcontroller. These are the Light CNN on 5-second signal windows with 100 Hz sampling rate and the MobileNet3-Tiny with 5-second signal windows and 100 Hz sampling rate, both trained on filtered data.

The filtering was at first done with the NeuroKit2 library, which was suitable for prototyping. However, the accuracy of these models dropped to about 77 % when implemented with preprocessing done on the microcontroller. It seems that our neural networks are sensitive to the data they were trained on and to minor artifacts in the signal introduced by the filtering method. So we had to retrain the models on data filtered with the method implemented on the microcontroller. The following table shows the performance of our two models when tested on the same data they were trained on.

	LCNN μC	LCNN NK2	MB3Tiny μC	MB3Tiny NK2
acc.[%]	91	94	94	94
sens.[%]	93	94	95	94
spec.[%]	89	92	92	94
prec.[%]	90	94	92	94
f1 score	0.92	0.94	0.94	0.94

Table 6.9: Comparison between different types of preprocessing
μC - preprocessing done on microcontroller
NK2 - preprocessed with the NeuroKit2 library [56],
acc.-accuracy, spec.-specificity, sens.-sensitivity,prec.-precision

Notice that the accuracy slightly drops when we use the proposed filtering method. We could likely gain a few percent in classification accuracy with a better preprocessing method. However, this is not necessarily significant.

6.5 QRS complex detection

There are various algorithms for identifying R peaks in the ECG signal. The oldest but still used Pan-Tompkins algorithm [49] (1985) makes use of filtering and adaptive thresholding. Recent approaches also use more advanced methods, such as neural networks, wavelet transforms, or heuristic methods. [48]

So how do we choose a suitable algorithm when so many algorithms are available? In the constrained environment of a mid-range microcontroller, the algorithm has to have several properties. It must be fast and computationally cheap. This excludes more complex algorithms, which are usually composed of computationally expensive operations. Another constraint is that the algorithm must not consume much RAM, especially when the goal is to run it alongside the neural network, which requires considerable resources. Therefore, transformations or other methods that need to allocate additional arrays are also not suitable.

Proposed method: Thresholding

This method aims to be as minimalistic as possible. As a result, it has its limits. However, it requires much fewer resources than other more advanced methods. It uses the preprocessing already utilized in the application. With

the data scaled between 0 and 1, the R peaks will have similar values across all ECG signals, which allows the use of thresholding for R peak detection. The optimal threshold was found with cross-validation, and it is 0.713.

We also use a 400 ms repolarization restraint to prevent detecting prominent T waves as R-peaks. This value was also selected by cross-validation to give the best performance on the training data. One disadvantage is that this limits the maximum heart rate the algorithm can detect to 150 bpm. However, such high heart rates were not present in the dataset we used.

Method evaluation

Because the datasets are not annotated for R peaks, and the author of this thesis values his time too much to annotate thousands of samples by hand, we do the next best thing. We take a more advanced R peak detection algorithm from the NeuroKit2 library [56] and compare it with the proposed method.

Then we implement the proposed low-complexity thresholding method and compare the peaks detected. Each method outputs the number of peaks detected and the position at which each peak was detected. If the number of detected peaks is equal and the absolute difference between the detected positions is five samples at maximum, we assume the detection to be successful. Furthermore, because sometimes the more advanced method tends to miss beats near the edges of the window, we also consider the detection correct if the positions match except for one beat near the edges. Additionally, there is also the possibility that two beats are near both edges, which is also considered. If neither of these cases occurs, the detection is considered to have failed. The methods were tested on 4621 5-second windows.

exact matches: 2227 samples
exact match except for one beat near the edges: 2199 samples
exact match except for two beats near the edges: 176 samples
total failures: 99 samples

Apart from the positions of detected R peaks, we can also compare the heart rates estimated by each method. With a maximum difference of ± 1 bpm, the predicted heart rates match in 77.6 % of cases (3589/4621 samples). With a slightly larger range of ± 3 bpm, we get a match in 91.1 % of samples (4209/4621 samples), and with an even larger range of ± 5 bpm, it is 94.7 %.

And a plot for a successful R-peak detection,

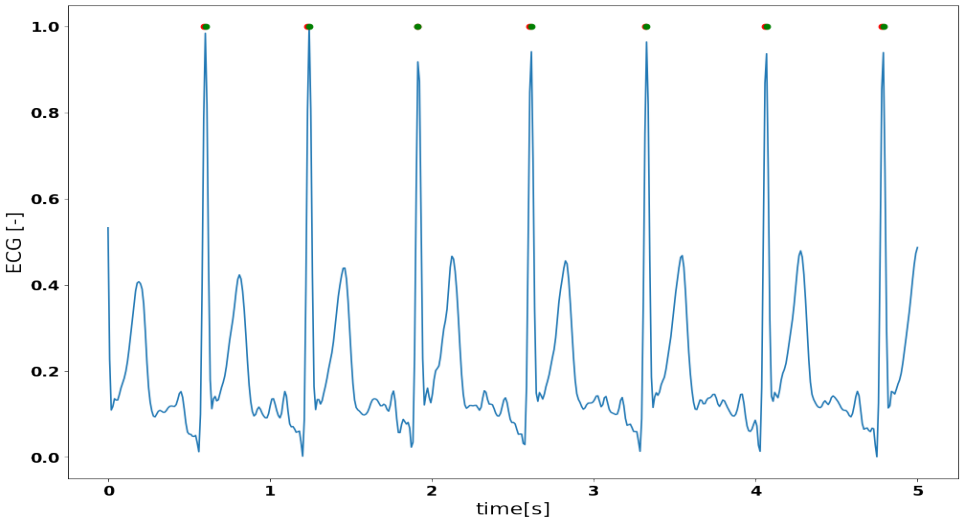


Figure 6.5: Succesful R peak detection, R peaks predicted by tresholding - red, NeuroKit [56] R peak detection - green

and a failed one.

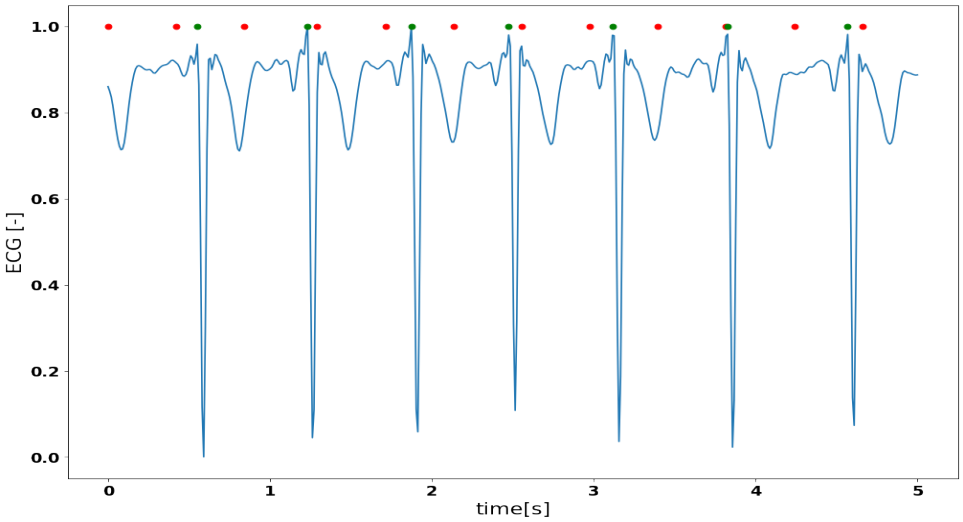


Figure 6.6: Detection failed, R peaks predicted by tresholding - red, NeuroKit [56] R peak detection - green

In terms of speed, in Google Colab [58] virtual machine, which uses a powerful Intel Xeon CPU, the more advanced method from the NeuroKit2 library [56] takes about 1 minute and 20 seconds to evaluate all 4621 samples. The proposed method takes approximately 1 second.

Based on these results, we can conclude that the proposed method significantly reduces computational load while being very accurate on more than 90 % of samples. As the portion of samples where the method totally failed is relatively tiny, the method is considered suitable for use.

Chapter 7

Application implementation

This flowchart describes how the full application works.

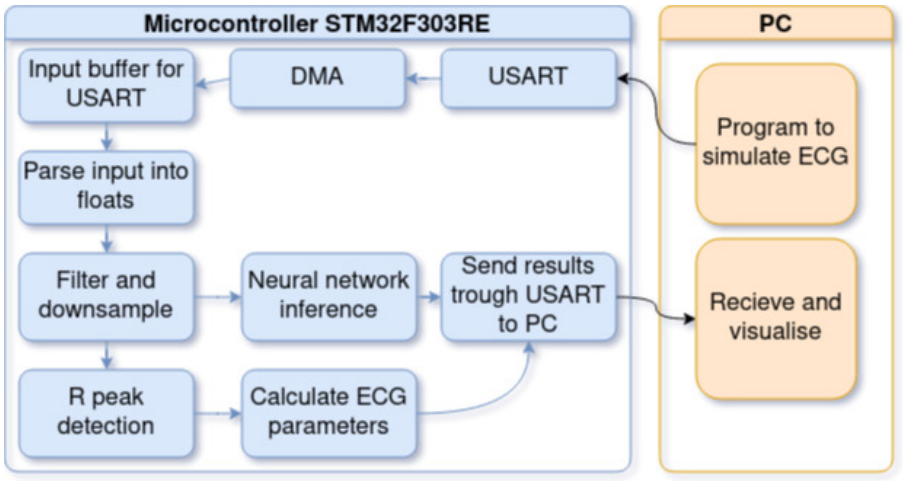


Figure 7.1: App workflow

And this chapter will describe all of the parts included in the application.

7.1 ECG simulator

This part of the application simulates the ECG machine. It loads the test data and transforms each number into a string. A separating character needs to be appended at the end of the string so the decoder knows where each number ends. Each character in the string is represented by an 8-bit value which is then sent to the microcontroller through USART. It is implemented

in the send_data.py script and accepts a command-line argument to specify the number of window samples to be sent in the form of –samples=NUM.

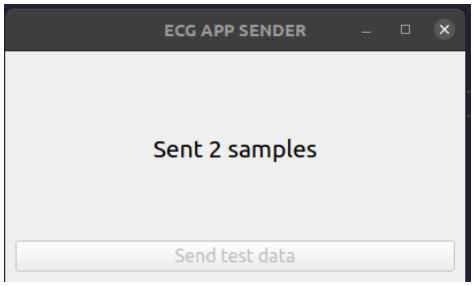


Figure 7.2: Sender application window

The sending application sends the data to the microcontroller trough a serial port. The port can be set in the code or with an optional argument in the --port=PORT (example for Linux --port=/dev/ttyACM0 and for Windows --port=COM9.) An error message is displayed if nothing is connected to the port the application tried to open.

7.2 Microcontroller application - receive input

Receive data

The first part of the application has to correctly receive the data and transform them from the char format in which they are transferred to the floating-point format. For this purpose, there exists a simple communication protocol. Each number is composed of numbers and a floating-point. Trailing zeros are added to the message to ensure a fixed message length. A program that can work with variable-length messages could also be created. However, I decided that the solution used is sufficient for testing purposes. At the end of each message, there is a separating character 'x', which is then used by the receiver to separate messages and parse them back into floats.

This method requires the ECG simulator to ensure that each message has the same length. This can be easily implemented into the simulated ECG machine. It would be possible to code a solution that would deal with variable-length messages. However, this would be pretty bothersome with the DMA, which does not provide any method to deal with input data besides the half-full and full callbacks. Moreover, all the methods that we are aware

of would defeat the whole purpose of DMA, which is to receive data without the CPU's intervention.

Another and more straightforward method of receiving the input would be to bring the voltage from the ECG machine to the analog-digital converter of the microcontroller. The DMA can then sample the voltage and place the samples in a buffer for further processing. This solution would be easier to implement and more robust. It would also eliminate the need for an input buffer for the USART, which would free up memory that other parts of the application could use. However, it is not suitable for testing purposes, as it requires a subject from which the ECG is measured, and also, we did not have access to an ECG sensor.

Parse input - double buffer

We want the application to be able to deal with a continuous stream of data. Our solution is to use a double buffer. The HAL library provides two callbacks for working with DMA. The first is executed when the first half of the buffer is reached, and the second is executed when the end of the buffer is reached. In step one, the input parser processes the lower half while DMA writes into the upper half. In step two, the input parser processes the upper half while DMA writes in the lower half. This way, information can be received and processed in parallel.

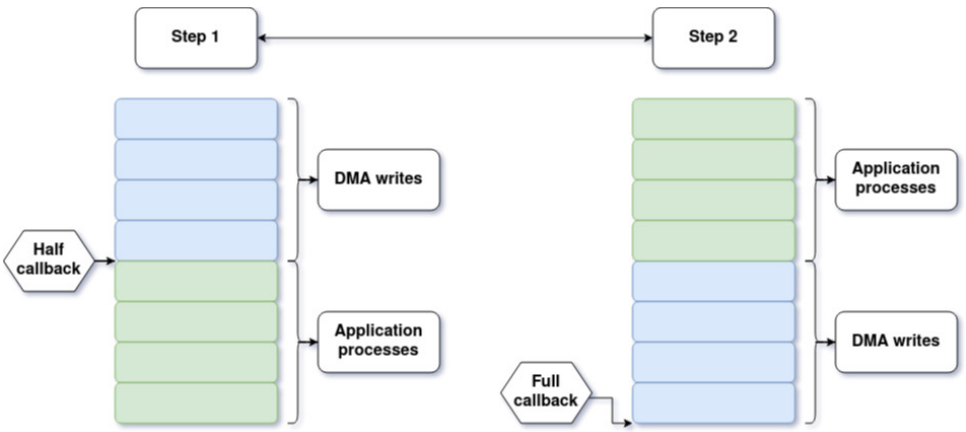


Figure 7.3: Double buffer schema

We have to guarantee that the application can process the data before it is overwritten for this to work. However, ECG is usually sampled at rates below

1 kHz, which is relatively slow, and therefore it leaves plenty of time for the application to parse the received data. And to the limits of my testing, the application works with even faster sampling speeds.

The DMA can be set to a circular mode, which means that when it reaches the end of the buffer, it will continue writing into the first byte of the buffer again. This configuration saves us from the need to manually restart the communication each time the end of the buffer is reached.

7.3 Microcontroller application - process input

Preprocessing the data

This stage differs based on the quality of the signal provided. If the signal is already preprocessed with analog filters implemented in the measuring device, we may not need to implement additional filtering. However, if this is not the case and noise is present, we need to filter the signal digitally.

In the process of preprocessing a raw ECG signal, we normalize the signal, effectively eliminating the DC component. A high-pass filter also has this property. However, the filter does not start filtering immediately at time zero, which results in the signal "swinging" from its previous DC value to zero.

Before downsampling to our target frequency, we need to limit the signal's bandwidth to prevent aliasing and filter the mains hum. For this purpose, we use a cascade of two fourth-order Chebyshev low-pass filters with epsilon 0.5 and a cutoff frequency of 40 Hz. Chebyshev filter has a ripple in the passband. However, at 40 Hz, we do not find many useful ECG components that would suffer from this, so this is acceptable. [30] The advantage of the Chebyshev filter is that it has a bigger gain, so it can better attenuate higher frequencies.

We use two second-order Butterworth filters with a cutoff of 0.5 Hz to filter the baseline drift. Butterworth has a flat response in the passband, which is wanted, as a ripple in low frequencies would distort the signal. After filtering, we scale the signal to the range [0,1] with Min-Max scaling, and now the signal is ready for analysis.

Statistical parameters

Detection of R peaks 6.5 allows the calculation of several statistical parameters. These are helpful in the detection of cognitive load and provide additional information about the input signal. The ones included in this work are mentioned in several papers as statistically significant. [28] [50] These are the mean RR interval,

$$\overline{RR} = \frac{\sum_{i=1}^{n} RR_i}{n} \tag{7.1}$$

the SDNN (standard deviation of RR intervals),

$$SDNN = \sqrt{\frac{\sum_{i=1}^{n}(RR_i - \overline{RR})^2}{n}} \tag{7.2}$$

and the RMSSD (root mean square of successive differences.)

$$RMSSD = \sqrt{\frac{\sum_{i=1}^{n-1}(RR_{i+1} - RR_i)^2}{n-1}} \tag{7.3}$$

In addition to these, the detection of R peaks also allows the estimation of heart rate. We considered two methods to calculate the heart rate. We can either use the number of peaks detected and the window length.

$$HR = 60 \cdot \frac{peaks\ detected}{window\ length} \tag{7.4}$$

Or the mean RR interval and the sampling frequency, which is considered better because it relies on the $\overline{RR}$, which is much more resistant to peaks not being detected. Also, the method works on any window length.

$$HR = 60 \cdot \frac{sampling\ frequency}{\overline{RR}} \tag{7.5}$$

These metrics are all calculated and displayed in milliseconds. An exception is the heart rate, which is standardly expressed in beats per minute.

Neural network

Implementing the neural network is quite simple with the X-Cube-AI [20] module API. It provides a graphical interface in which we can load the compressed model and analyze its memory requirements and computational complexity. An "X-CUBE-AI/App" folder is added to the project. Inside, we find modelname.h, which contains information about the version and API used, modelname_data.h/c contains the weights and activations, and modelname.h/c contains the functions used to interact with the network and information for allocating the tensors for the network. The API functions for interacting with the network were then used to implement three functions, *ai_init()*, *ai_run()* and *perform_inference()*. These functions are then used for initialization and inference with the network.

7.4 Application memory requirements and execution speed

Here we compare the memory requirements and processing speed of the entire application. This includes filtering and scaling the signal, detecting the R peaks, computing the parameters, and neural network model inference. The time is measured from the moment the input buffer is filled until all the results are sent to the PC. The execution speed is timed without plotting the signal and with plotting the signal, which requires sending the processed ECG signal to the PC.

	Flash [kB]	RAM [kB]	execution speed [s]
LightCNN 5s 100Hz	94.31	32.27	0.23
MB3-tiny 5s 100Hz	113.45	45.66	0.24
LightCNN 10s 100Hz	114.22	41.95	0.47
MB3-Tiny 10s 100Hz	129.16	55.34	0.46

Table 7.1: Memory requirements and execution speed

7.5 PC visualisation

The application displays the prediction and parameters for the last 5-second window and keeps a graph for the last 20 samples (100 seconds).

The receiving application reads from the serial port to which the microcontroller is connected. This can be set in the code or with an optional argument in the form --port=PORT (example for Linux --port=/dev/ttyACM0 and for Windows --port=COM9.) An error message is displayed if nothing is connected to the port the application tried to open.

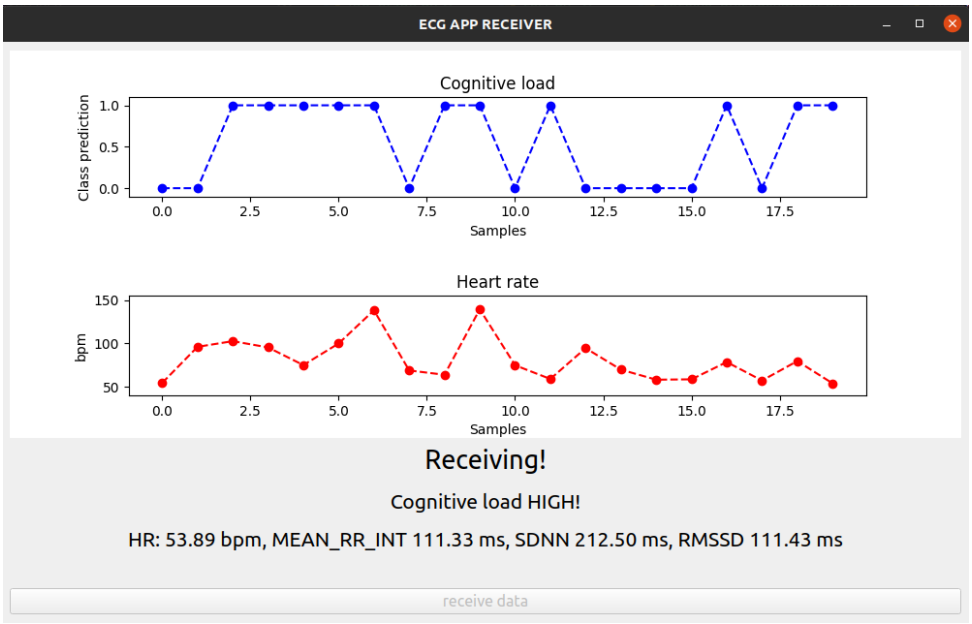


Figure 7.4: Receiver application window

PC application for windows

The Windows operating system does not natively support two programs accessing one serial port simultaneously. For this reason, another version of the GUI application exists, which integrates the functionalities of the sender and the receiver applications into one application.

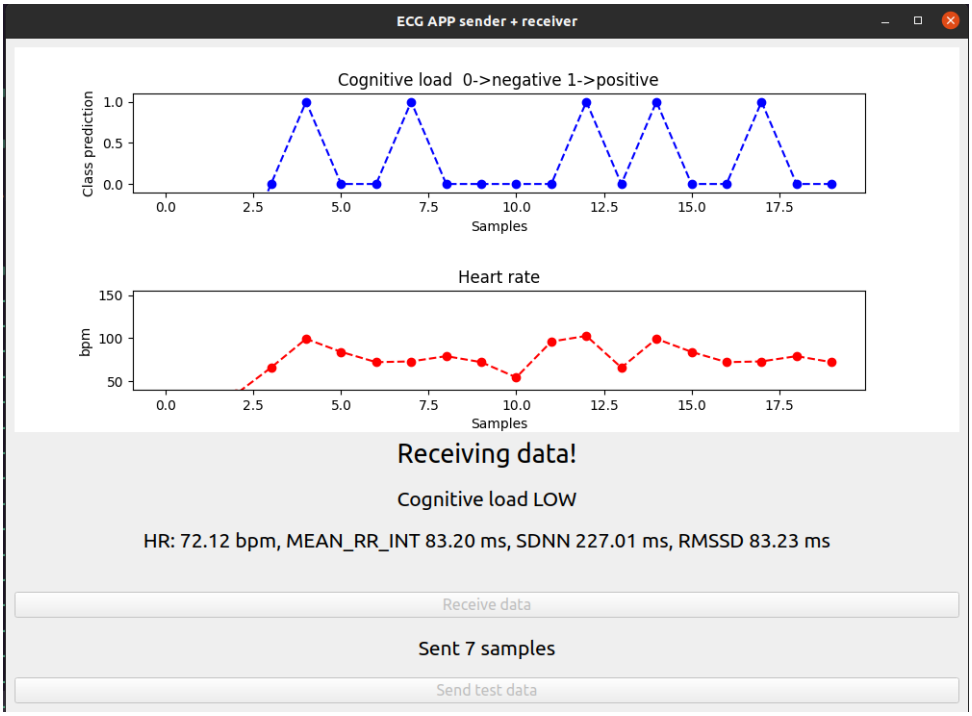


Figure 7.5: Receiver + sender application window

Chapter 8

Discussion and Conclusions

8.1 Comparison against other methods

A common approach is to extract features from the time and frequency domains of ECG signal and train classifiers such as Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), or K-Nearest Neighbors (KNN) on them. [51][41][13] However, we use a different approach in this thesis. We feed the signal directly into a convolutional neural network. This approach provides a fast and accurate prediction of cognitive load without complicated preprocessing and computation of several statistical parameters and allows us to reach state-of-the-art classification accuracy with only the time series of the ECG signal.

However, it also has its disadvantages. A neural network is rather sensitive to the data on which it was trained and will only perform well on data preprocessed the same way as the training data.

8.2 Limitations & Recommendations for further continuation

Even though the models in this work reached high accuracies, they are still limited by the available data. We use a dataset with 87 subjects, which is slightly larger than in other research, where the number is usually around 30 subjects [13][41][28]. However, the dataset should still be much larger, as it is suggested that the cognitive load can manifest differently in each individual

[51].

Furthermore, if we could train on data recorded for a bigger range of tasks, it could lead to better generalization. It seems that the successful detection of cognitive load depends on the stimuli. For example, detecting a high cognitive load during stressful driving [28][50] seems like an easier task than detecting a high cognitive load during arithmetic tasks. [50]

Using more physiological signals to estimate cognitive load should also be beneficial, as the influence of cognitive load on physiological signals can vary between individuals [51]. Devices that use multiple signals to estimate the cognitive load also seem to reach higher accuracy.

Exploring different model architectures could also lead to further improvements in classification accuracy and general network performance.

Another possible continuation of this work would be to create a device that can measure ECG and classify cognitive load from it. In this form the application would have to receive samples from an ECG sensor, so the input method would probably need to be changed. We could use I^2C to communicate with the sensor digitally or take samples with an A/D converter, both of which could be implemented into the existing code without much trouble.

8.3 Conclusions

In this work, we proposed, implemented, and tested a microcontroller application for the real-time detection of high and low cognitive load from the ECG signal. The application receives a continuous stream of ECG data, applies preprocessing to it, runs a trained convolutional neural network, and computes the heart rate, mean RR interval, SDNN, and RMSSD. It sends the network prediction and the calculated parameters over a serial port to a PC application that plots them in a graphical window.

We tested two architectures and many parameters and selected two configurations that produced the best results. These are the Light CNN with a 5-second window and a 100 Hz sampling rate and the MobileNet-Tiny with a 5-second window and a 100 Hz sampling rate. Both reached up to 94 % in classification accuracy. However, generally, the MobileNet model seems to

be slightly better. We trained the models on a dataset combined out of the WESAD [16], CLAS [25], and BIOMECH datasets. This combined dataset contains nearly 11 hours of ECG recordings from 87 subjects.

The application uses filtering and scaling to preprocess the ECG signal and is capable of detecting R peaks with a custom low-complexity algorithm that is significantly faster than more advanced methods and matches the accuracy of an R peak algorithm from the NeuroKit2 library [56] in around 95 % of cases.

These results prove that it is possible to detect a person's cognitive load on an embedded device in real-time and contribute to the goal of creating systems that can intelligently recognize the state of the user or patient.

Bibliography

[1] Sweller, J. Cognitive load theory. In J. P. Mestre & B. H. Ross (Eds.), The psychology of learning and motivation: Cognition in education (pp. 37–76). (2011). Elsevier Academic Press. https://doi.org/10.1016/B978-0-12-387691-1.00002-8

[2] Sweller, John (April 1988). *"Cognitive Load During Problem Solving: Effects on Learning"*. Cognitive Science. 12 (2): 257–285.

[3] *Cognitive Load Theory* BRUNKEN N, R.; MORENO, R.; PLASS, J. 2010 Press, 2010. ISBN 9780521860239. Accessed on: 22 Feb. 2022.

[4] Peterson, L., and Peterson, M. J. (1959). *Short-term retention of individual verbal items. Journal of Experimental Psychology,* 58(3), 193–198. https://doi.org/10.1037/h0049234

[5] Leppink, J., Paas, F., Van der Vleuten, C. P. M., Van Gog, T. & Van Merriënboer, J. J. G. *Development of an instrument for measuring different types of cognitive load.* Behavior Research Methods 45, 1058–1072, https://doi.org/10.3758/s13428-013-0334-1 (2013).

[6] Hughes, A. M. et al. (2019) *'Cardiac Measures of Cognitive Workload: A Meta-Analysis'*, Human Factors, 61(3), pp. 393–414. doi:https://doi.org/10.1177/0018720819830553.

[7] *TinyML* Warden, P. and Situnayake, D., 2019. Publisher O'Reilly Media, Inc.

[8] J. Heard, C. E. Harriott, and J. A. Adams, *A survey of workload assessment algorithms* IEEE Transactions on Human-Machine Systems, (2018.)

[9] Yu, J., Liu, G.Y., Wen, W.H. and Chen, C.W. (2020), *Evaluating cognitive task result through heart rate pattern analysis.* Healthc. Technol. Lett., 7: 41-44. https://doi.org/10.1049/htl.2018.5068

[10] Mahdiani S, Jeyhani V, Peltokangas M, Vehkaoja A. *Is 50 Hz high enough ECG sampling frequency for accurate HRV analysis?*, Annu Int Conf IEEE Eng Med Biol Soc. 2015;2015:5948-51. doi: https://doi.org/10.1109/EMBC.2015.7319746. PMID: 26737646.

[11] Solhjoo, S., Haigney, M. C., McBee, E., van Merrienboer, J., Schuwirth, L., Artino, A. R., Jr, Battista, A., Ratcliffe, T. A., Lee, H. D., & Durning, S. J. (2019). *Heart Rate and Heart Rate Variability Correlate with Clinical Reasoning Performance and Self-Reported Measures of Cognitive Load.* Scientific reports, 9(1), 14668. DOI: https://doi.org/10.1038/s41598-019-50280-3

[12] Ayres P, Lee JY, Paas F and van Merriënboer JJG (2021) *The Validity of Physiological Measures to Identify Differences in Intrinsic Cognitive Load.* Front. Psychol. 12:702538. doi: https://doi.org/10.3389/fpsyg.2021.702538

[13] Xiong, Ronglong et al. *"Pattern Recognition of Cognitive Load Using EEG and ECG Signals.* Sensors (Basel, Switzerland) vol. 20,18 5122. 8 Sep. 2020, doi: https://doi.org/10.3390/s20185122

[14] Romine, W. L., Schroeder, N. L., Graft, J., Yang, F., Sadeghi, R., Zabihimayvan, M., Banerjee, T. *Using machine learning to train a wearable device for measuring students' cognitive load during problem-solving activities based on electrodermal activity, body temperature, and heart rate: Development of a cognitive load tracker for both personal and classroom use.* (2020). Sensors, 20(17), 4833. doi:http://dx.doi.org/10.3390/s20174833

[15] K. Plarre, A. Raij, and M. Scott. 2011. *Continuous inference of psychological stress from sensory measurements collected in the natural environment.* In 10th International Conference on Information Processing in Sensor Networks (IPSN). 97–108

[16] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, and Kristof Van Laerhoven. 2018. *Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection.* In Proceedings of the 20th ACM International Conference on Multimodal Interaction (ICMI '18).

Association for Computing Machinery, New York, NY, USA, 400–408. DOI:https://doi.org/10.1145/3242969.3242985

[17] Gay W. (2018) *USART. In: Beginning STM32.* Apress, Berkeley, CA. https://doi-org.ezproxy.techlib.cz/10.1007/978-1-4842-3624-6_6

[18] *Documentation for the STM32 Nucleo f303re by STMicroelectronics*, Accesed on: 27. March 2022, Available on: https://www.st.com/en/evaluation-tools/nucleo-f303re.html?ecmp=tt9470_gl_link_feb2019&rt=db&id=DB2196#documentation

[19] *Documentation for the STM32F3/STM32G4 CCM SRAM by STMicroelectronics*, Accesed on: 27. March 2022, Available on: https://www.st.com/resource/en/application_note/an4296-use-stm32f3stm32g4-ccm-sram-with-iar-embedded-workbench-keil-mdkarm-stmicroelectronics-stm32cubeide-and-other-gnubased-toolchains-stmicroelectronics.pdf

[20] *Documentation for the STM X-Cube-AI package*, Accessed on: 15.5.2022, Available on: https://www.st.com/en/embedded-software/x-cube-ai.html#documentation

[21] *Cyclical Learning Rates for Training Neural Networks Smith*, Leslie N., publisher arXiv, 2015, https://doi.org/10.48550/arxiv.1506.01186,

[22] Pavel Surmenok *Estimating an Optimal Learning Rate For a Deep Neural Network* Towards Data Science, Created on: 13.Nov. 2017, Acessed on: 29. Mar. 2022 https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0

[23] *Post-training integer quantization* Tensor-Flow, Acessed on: 30.Mar. 2022 Available on: https://www.tensorflow.org/lite/performance/post_training_integer_quant

[24] Nagel, Markus and Fournarakis, Marios and Amjad, Rana Ali and Bondarenko, Yelysei and van Baalen, Mart and Blankevoort, Tijmen, *A White Paper on Neural Network Quantization*, publisher arXiv, 2021, https://doi.org/10.48550/arxiv.2106.08295

[25] V. Markova, T. Ganchev and K. Kalinkov, *"CLAS: A Database for Cognitive Load, Affect and Stress Recognition,"* 2019 International Conference

on Biomedical Innovations and Applications (BIA), 2019, pp. 1-4, doi: https://doi.org/10.1109/BIA48344.2019.8967457.

[26] J. Sola and J. Sevilla, *"Importance of input data normalization for the application of neural networks to complex industrial problems,"* in IEEE Transactions on Nuclear Science, vol. 44, no. 3, pp. 1464-1468, June 1997, doi: https://doi.org/10.1109/23.589532

[27] Tzevelekakis, K.; Stefanidi, Z.; Margetis, G. *Real-Time Stress Level Feedback from Raw Ecg Signals for Personalised, Context-Aware Applications Using Lightweight Convolutional Neural Network Architectures.* Sensors 2021, 21, 7802. https://doi.org/10.3390/s21237802

[28] Tjolleng, Amir & Jung, Kihyo. *Development of a Real-time System for Detecting Driver's Cognitive Load using Multi-Layer Artificial Neural Network on ECG Signals.* Journal of the Ergonomics Society of Korea. (2020). 39. 625-636. https://doi.org/10.5143/JESK.2020.39.6.625

[29] Kwon O, Jeong J, Kim HB, Kwon IH, Park SY, Kim JE, Choi Y. *Electrocardiogram Sampling Frequency Range Acceptable for Heart Rate Variability Analysis.* Healthc Inform Res. 2018 Jul;24(3):198-206. doi: https://doi.org/10.4258/hir.2018.24.3.198 Epub 2018 Jul 31. PMID: 30109153; PMCID: PMC6085204.

[30] E. Ajdaraga and M. Gusev, "Analysis of sampling frequency and resolution in ECG signals," 2017 25th Telecommunication Forum (TELFOR), 2017, pp. 1-4, doi: https://doi.org/10.1109/TELFOR.2017.8249438.

[31] B.-U. Kohler, C. Hennig and R. Orglmeister, *The principles of software QRS detection* IEEE Engineering in Medicine and Biology Magazine, vol. 21, no. 1, pp. 42-57, 2002.

[32] *What is an ECG?* alivecor.com, accessed on 2.4.2022, Available online on: https://www.alivecor.com/education/ecg.html

[33] Knipe, H., Hacking, C. *Innervation of the heart.* Reference article, Radiopaedia.org. (accessed on 06 Apr 2022) https://doi.org/10.53347/rID-26402

[34] Zixing Cai, Lijue Liu, Baifan Chen, Yong Wang *Artificial intelligence from beginning to date*, Publisher Tsinghua University Press and World Scientific, ISBN:9789811223716, Year: 2021, Pages used:175-202

[35] Yuanming Shi, Kai Yang, Zhanpeng Yang, Yong Zhou *Mobile Edge Artificial Intelligene, Opprtunities and Challenges* Publisher: Elsevier, Academic press, ISBN: 9780128238172, Year: 2022, Pages used: 7-35 and 71-80

[36] Ketan Doshi *Batch Norm Explained Visually, How it works, and why neural networks need it* Accessed on 10.4.2022, Available online on: `https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739`

[37] Wikipedia, the free encyclopedia *Feature scaling* Accessed on 15.5.2022, Last edited on: 25 April 2022, Available on: `https://en.wikipedia.org/wiki/Feature_scaling`

[38] Ibrahem Kandel, Mauro Castelli, *The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset*, ICT Express, Volume 6, Issue 4, 2020, Pages 312-315, ISSN 2405-9595, DOI: `https://doi.org/10.1016/j.icte.2020.04.010`

[39] Chi-Feng Wang *A Basic Introduction to Separable Convolutions* Acessed on 11.4.2022 Available on: `https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728`

[40] Yurong Luo, Rosalyn H. Hargraves, Ashwin Belle, Ou Bai, Xuguang Qi, Kevin R. Ward, Michael Paul Pfaffenberger, Kayvan Najarian, *"A Hierarchical Method for Removal of Baseline Drift from Biomedical Signals: Application in ECG Analysis"*, The Scientific World Journal, vol. 2013, Article ID 896056, 10 pages, 2013. `https://doi.org/10.1155/2013/896056`

[41] Eija Haapalainen, SeungJun Kim, Jodi F. Forlizzi, and Anind K. Dey. 2010. *Psycho-physiological measures for assessing cognitive load.* In Proceedings of the 12th ACM international conference on Ubiquitous computing (UbiComp '10). Association for Computing Machinery, New York, NY, USA, 301–310. DOI: `https://doi.org/10.1145/1864349.1864395`

[42] Cary Deck, Salar Jahedi, *The effect of cognitive load on economic decision making: A survey and new experiments,* European Economic Review, Volume 78, 2015, Pages 97-119, ISSN 0014-2921, DOI: `https://doi.org/10.1016/j.euroecorev.2015.05.004`

[43] *Weigth pruning guide*, accessed on 16.4.2022, `https://www.tensorflow.org/model_optimization/guide/pruning`

[44] Sandler, Mark and Howard, Andrew and Zhu, Menglong and Zhmoginov, Andrey and Chen, Liang-Chieh *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, publisher arXiv, year 2018, copyright arXiv.org perpetual, non-exclusive license, DOI: https://doi.org/10.48550/arxiv.1801.04381

[45] Howard, Andrew and Sandler, Mark and Chu, Grace and Chen, Liang-Chieh and Chen, Bo and Tan, Mingxing and Wang, Weijun and Zhu, Yukun and Pang, Ruoming and Vasudevan, Vijay and Le, Quoc V. and Adam, Hartwig, *Searching for MobileNetV3*, publisher arXiv, year 2019, copyright arXiv.org perpetual, non-exclusive license, DOI: https://doi.org/10.48550/arxiv.1905.02244

[46] *Heart rate* Wikipedia, the free encyclopedia, last edited on: April 8th 2022, visited on: April 18th 2022, https://en.wikipedia.org/wiki/Heart_rate

[47] Daniel Godoy *Understanding binary cross-entropy / log loss: a visual explanation* Towards Data Science, Created on: 21.Nov. 2018, Acessed on: 28.4. April. 2022 https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[48] Di Nunzio, Luca & Silvestri, Francesca & Cardarilli, Gian Carlo & Fazzolari, Rocco & re, Marco. *Comparison of Low-Complexity Algorithms for Real-Time QRS Detection using Standard ECG Database.* (2018). International Journal on Advanced Science, Engineering and Information Technology. 8. DOI: https://doi.org/10.18517/ijaseit.8.2.4956.

[49] J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," in IEEE Transactions on Biomedical Engineering, vol. BME-32, no. 3, pp. 230-236, March 1985, doi: https://doi.org/10.1109/TBME.1985.325532.

[50] Pritam Sarkar and Kyle Ross and Aaron J. Ruberto and Dirk Rodenbura and Paul Hungler and Ali Etemad *Classification of Cognitive Load and Expertise for Adaptive Simulation using Deep Multitask Learning*, Publisher IEEE, 8th International Conference on Affective Computing and Intelligent Interaction (ACII), 2019, DOI: https://doi.org/10.1109%2Facii.2019.8925507

[51] E. Ferreira et al., *Assessing real-time cognitive load based on psycho-physiological measures for younger and older adults*, 2014 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2014, pp. 39-48, doi: https://doi.org/10.1109/CCMB.2014.7020692

[52] Prechelt, L. (2012). *Early Stopping — But When?*. In: Montavon, G., Orr, G.B., Müller, KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_5

[53] Hunter, J. D., Matplotlib: A 2D graphics environment, Computing in Science & Engineering, Volume 9, Number 3, Pages 90–95, publisher IEEE COMPUTER SOC, doi https://doi.org/10.1109/MCSE.2007.55, 2007

[54] © 2022 Geek3 / GNU-FDL, accessed on 7.5.2022, available from: https://upload.wikimedia.org/wikipedia/commons/b/bd/Filters_order5.svg

[55] Diagrams.net: https://www.diagrams.net, Source Code: https://github.com/jgraph/www.diagrams.net-source, developed by jgraph

[56] Makowski, D., Pham, T., Lau, Z. J., Brammer, J. C., Lespinasse, F., Pham, H., Schölzel, C., & Chen, S. H. A. (2021). NeuroKit2: A Python toolbox for neurophysiological signal processing. Behavior Research Methods, 53(4), 1689–1696. https://doi.org/10.3758/s13428-020-01516-y

[57] Kingma, Diederik P. and Ba, Jimmy, *Adam: A Method for Stochastic Optimization*, publisher arXiv, 2014, copyright arXiv.org perpetual, non-exclusive license, doi https://doi.org/10.48550/arxiv.1412.6980

[58] Google Colaboratory, https://research.google.com/colaboratory/faq.html

[59] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg,

Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from `https://tensorflow.org`.

[60] Holton Thomas, *Digital signals processing: principles and applications*, published 2021, publisher Cambridge University Press, ISBN 978108418447