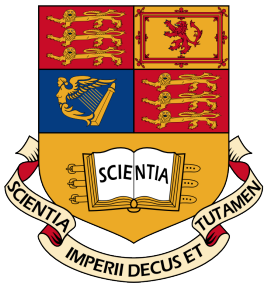
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Final Report 2022



Imperial College London

Project Title:	**Cuffless Estimation of Blood Pressure from Photoplethysmography Signals using Transformers**
Student:	**Arijit Bhattacharyya**
CID:	**01496199**
Course:	**MEng Electrical and Electronic Engineering**
Project Supervisor:	**Professor Esther Rodriguez Villegas**
Co Supervisor:	**Dr Zaibaa Patel**
Second Marker:	**Dr Christos Bouganis**

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

Abstract (CHANGE AT END)

Abstract should consist of motivation, methods, results, conclusion. All concise & capturing the reader within 200 words. Your abstract needs working and finalising once you have your results and concluding statement.

The prevention, evaluation, and treatment of hypertension have attracted increasing attention in recent years. The advancement of wearable technology has resulted in increasing importance into the monitoring of non-invasive ambulatory blood pressure, compared to the traditional invasive blood-pressure monitoring methods. As photoplethysmography (PPG) technology has been widely applied to wearable sensors, the noninvasive estimation of blood pressure (BP) using the PPG method has received considerable interest. For this project, systolic and diastolic BPs are estimated using PPG signals. A Recurrent Neural Network (RNN) is used for estimation. Due to their being several alternative existing methods for estimating blood pressure, it was necessary to perform a comparison between the best performing Deep Learning based methods. Overall, the proposed method obtains better accuracy. The model achieves a mean absolute error of mmHg for systolic BP and mmHg for diastolic BP.

Acknowledgements

Firstly, I would like to thank Dr Zaibaa Patel for her patience and guidance in the development of this project. I would also like to thank Professor Esther Rodriguez Villegas for her valuable feedback and for allowing me the opportunity to present my work.

Next, I want to thank all of my friends at Imperial for supporting me throughout my 4 years here and for helping me to grow as both an engineer and a person.

Finally, I want to thank my Mum and Dad for all of their support throughout my life. Without them, I would never have had the opportunity to study at such an amazing institution.

Contents

1 Introduction

1.1 Motivation

Cardiovascular disease is one of the main causes of death around the world. High blood pressure (BP), which is also known as hypertension, is a common condition which can be a cause of cardiovascular disease [1]. According to the World Health Organization (WHO), the mortality rate due to hypertension is 9.4 million per year and it causes 55.3% of total deaths in cardiovascular patients [2]. If hypertension is detected early and prevented, this will greatly lower the number of deaths associated with cardiovascular diseases [2].

Recent developments in technology have made wearable sensors, such as Electrocardiogram (ECG) and Photoplethysmography (PPG) sensors significantly more popular in today's world. These sensors provide real-time 24 hour monitoring of the human bodily function. Hence there is great potential in using these sensors to diagnose medical conditions, such as hypertension, in real-time, thus helping to save lives [3]. Ambulatory BP monitoring is seen as a promising method for detecting early symptoms of hypertension [4]. There is a lot of existing research to predict ambulatory BP using methods which are cuff-less, continuous and non-invasive [5]. Hence wearables are seen as a viable option for this. ECG and PPG sensors have been discovered to be a potential estimator of blood pressure that cause minimal harm to patients compared to existing cuff-based methods [6] [7].

1.2 High-level problem statement

Needs rewriting. Aim: Your intention/what you hope to achieve.

Objectives: Statements of measurable outcomes/What will you be doing to achieve the aim/desire outcome. (this is the work you're going to do) The aim is not to implement and evaluate different techniques. The aim is to estimate cuffless BP using PPG for wearable technology purpose. I advise you to have a clear aim and clear objectives. Objectives can be listed as bullets or numerated. These objectives will reappear in your conclusion where you state if you completed them or not.

Based on the provided motivation, the main aim of this Final Year Project (FYP) is to estimate cuff-less blood pressure values using ECG and/or PPG signals, so that this estimation process can be integrated onto future wearable technology devices. Hence, these are the following objectives for this Final Year Project (FYP):

- Conduct a literature review to assess what are the most popular methods for estimating cuff-less BP values and to decide on the most feasible implementation for this FYP
- Develop a novel algorithm in the Python programming language to estimate cuffless blood pressure
- Assess the performance of this algorithm against existing methods
- Conclude whether this method is feasible for future wearable technology products

1.3 Overview of work

This section provides a chronological overview of the work that will be done in this FYP. The comprehensive overview of the work is provided in the Gantt Chart in the Appendix.

1.3.1 Autumn Term 2021

The main tasks for this term were classified under the theme of **Research and Understanding**. The main tasks were as follows:

- Searching for all background knowledge required to understand the motivation and objectives of this project
- Investigation into existing experimentations conducted with wearable technologies for estimating blood pressure
- Research into signal denoising techniques and machine learning based methods for estimating BP
- Familiarisation with the PhysioNet MIMIC database
- Conducting a literature survey in order to assess what is the best method for the cuffless estimation of blood pressure

1.3.2 Spring Term 2022

The main tasks for this term were classified under the theme of **Implementation of chosen methods**. The main tasks were as follows:

- The implementation of Convolutional Neural Network (CNN) architectures
- The implementation of a Recurrent Neural Network - Long Short Term Memory (RNN - LSTM) architecture
- The implementation of a Transformer Encoder architecture
- Testing of the architectures on the MIMIC database using different parameters (window length, ...?)

1.3.3 Summer Term 2022

The main tasks for this term were classified under the theme of **Performance Testing**. The main tasks were as follows:

- Finetuning of the Transformer Encoder architecture
- Graphical comparisons of the different architectures using error metrics
- Interpretation of results and conclusions

2 Background

In this chapter, the aim is to provide sufficient background information, in order to understand how the cuff-less estimation of blood pressure (BP) values can be achieved. An overview on all necessary fields will first be given. These fields are as follows:

- Medical Background
- Cuffless methods for deriving blood pressure
- Neural Networks
- Overview on cuffless blood pressure device options

Finally, a literature review will be conducted to assess what is the most feasible implementation for blood pressure cuffless estimation for this FYP.

2.1 Medical background

In this chapter, all of the medical knowledge required to understand the basis of this project will be discussed.

2.1.1 Hypertension

The heart can suffer from a variety of diseases and pathologies. Low blood pressure, or hypotension, has the potential to cause a lack of oxygen flowing to the brain and other organs, causing shock [8]. Whilst hypotension is a serious issue, hypertension has been identified by the World Health Organization (WHO) as the most significant risk factor for cardiovascular diseases [9]. According to the 2017 American Heart Association guidelines for hypertension, the risk of developing stage two hypertension, $\geq$ 140 mmHg systolic or $\geq$ 90mmHg diastolic is almost 90% [7] (see Table 1). Over 20% of adults have hypertension and its complications cause a major number of diseases, including heart attacks, strokes and heart failure. If hypertension is not diagnosed and properly treated it can even cause death [2].

Hypertension or high blood pressure (BP) is where blood continues to exert more and more pressure on the arterial walls. One particular disease linked to hypertension is hypertrophic cardiomyopathy, as indicated in Figure 1,

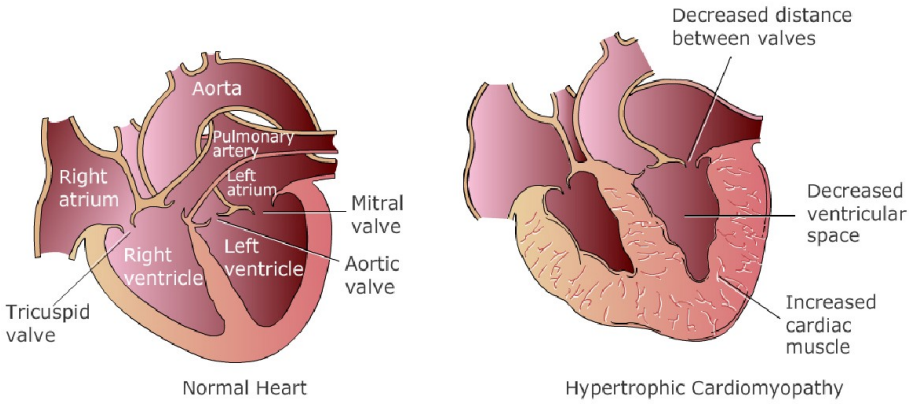


Figure 1: The effects of hypertension on the heart [10]

Hence it is clear that hypertension is one of the largest motivating factors for this project.

Table 1: Categories of blood pressure in adults [9] [11]

Blood pressure classification	**Blood Pressure (mmHg)**	
	Systolic	**Diastolic**
Hypotension	≤ 90	Or ≤ 60
Normal	90-119	And 60-79
Prehypertension	120-139	Or 80-89
Stage 1 hypertension	140-159	Or 90-99
Stage 2 hypertension	≥ 160	Or ≥ 100
Isolated Systolic hypertension	≥ 140	And < 90
Hypertensive crisis	≥ 180	Or ≥ 110

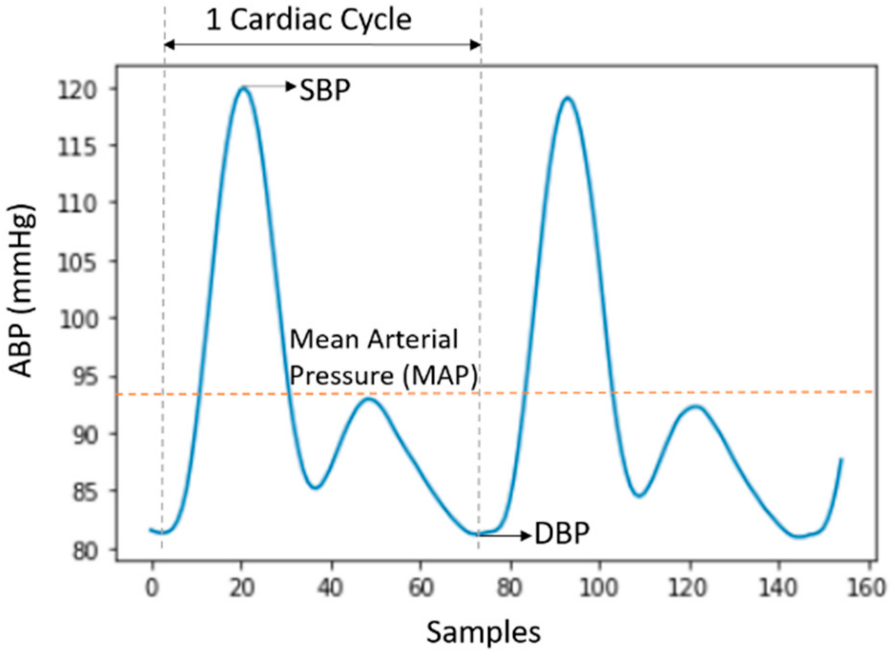
2.1.2 Ambulatory Blood Pressure (ABP)

ABP monitoring (ABPM) is when BP is measured as the patient moves around, and it allows patients to still live their normal daily lives [12]. It has been classed as the gold standard for detecting and diagnosing hypertension and also assessing BP values over a 24 hour period [4]. ABPM provides data on several important and unique parameters [4]. This data can explain how changes in your BP may correlate with your daily activities and sleep patterns [12]. Conventionally, ABP is monitored by using a cuff attached to a portable device which is worn on the patient's waist [4]. In the data provided for this project, the blood pressure signals have been recorded from ICU patients. As a result, these waveforms are not ABP waveforms but are instead Arterial Blood Pressure waveforms. DO YOU WANT TO KEEP THIS LAST SENTENCE IN?

2.1.3 Blood Pressure measurements

Blood pressure (BP) is the force of the blood pushing against the arterial walls as the heart pumps blood. It is measured in millimeters of mercury (mmHg) [3]. BP is measured in terms of systolic blood pressure (SBP) and diastolic blood pressure (DBP). These values are the

maximum and minimum pressure values of an Arterial Blood Pressure waveform during a cardiac cycle respectively [11] [13]. An example of this structure is provided in Figure 2.



(b)

Figure 2: Structure of an Arterial Blood Pressure signal [14]

As shown in Figure 2, the Systolic and Diastolic Blood Pressure values of the waveform are defined by the maximum and minimum ABP values within the provided sampling window.

BP has oscillations or pulses that mirror the oscillatory nature of the heart. The blood is propelled during systole, also known as heart contraction, and the blood is rested during diastole, known as heart relaxation, as illustrated in Figure 3.

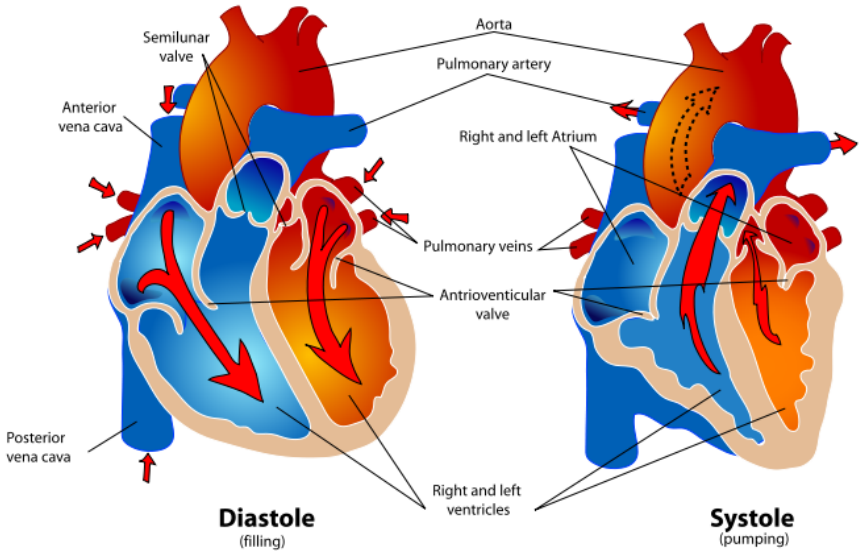


Figure 3: Visualisation for SBP and DBP [15]

There are two conventional methods for measuring blood pressure (BP). These are invasive and non-invasive methods.

The most popular form of invasive BP measurement is catheterization [5]. Invasive BP measurements are continuous and the most accurate from heartbeat to heartbeat. As a result these measurements are recognised as the gold standard internationally [1] [16]. However, this method is usually restricted to hospitals, as medical supervision is required [13]. In addition, this method poses several health risks, including bleeding and infection. As a result, invasive measurements are only utilised for critically ill patients in intensive care units and for use during surgery [5][16]. The main form of non-invasive BP measurements are through upper arm cuffed monitors. Cuff-based methods provide BP measurements without any major side effects as opposed to BP measured invasively [16]. However, patients will feel uncomfortable with long term monitoring due to the painful cuff inflation which interrupts the regular blood flow [8]. In addition, these methods can only measure BP intermittently with intervals between measurements greater than at least two minutes. These devices are too cumbersome to wear during measurements. Also, it has been found that over three in ten home BP monitoring cuffs have produced inaccurate results [17].

As a result, the existing invasive and non-invasive BP measurement techniques are not feasible for an implementation involving continuous ambulatory BP monitoring [16]. Hence, after having assessed the viability of all aforementioned methods, it is clear that it is difficult for these methods to be integrated with wearable technologies, which continue to gain popularity in commercial sectors and clinical practice [1]. This provides the motivation in using other heart signals which do not require invasive or cuff-based methods to be measured accurately.

2.1.4 Electrocardiogram (ECG) signals

ECG signals provide an overview of the electrical impulses occurring in the heart [11]. Electrical changes in the heart conduct through the body and are received at skin level. The record of these electrical fluctuations during the cardiac cycle is called the Electrocardiogram (ECG) [18]. The signals are recorded by measuring the electric potential difference by placing electrodes across the heart of an individual [8] [11]. These electrodes are connected to the ECG machine with recordings from 12 different places on the body, which is known as the 12-lead ECG. The standard ECG leads are I, II, III, aVF, aVR, aVL, V1, V2, V3, V4, V5, V6. Leads I, II, III, aVR, aVL, aVF are classed as the limb leads and the others are precordial leads [8].

The QRS complex of an ECG signal is detailed in Figure 4. This complex is first created through the generation of the electrical impulses from the heart. These signals then move along the electrical highway and as a result cause the ventricles to contract and pump oxygenated blood into the arteries. Physically, this whole describes the QRS complex [18].

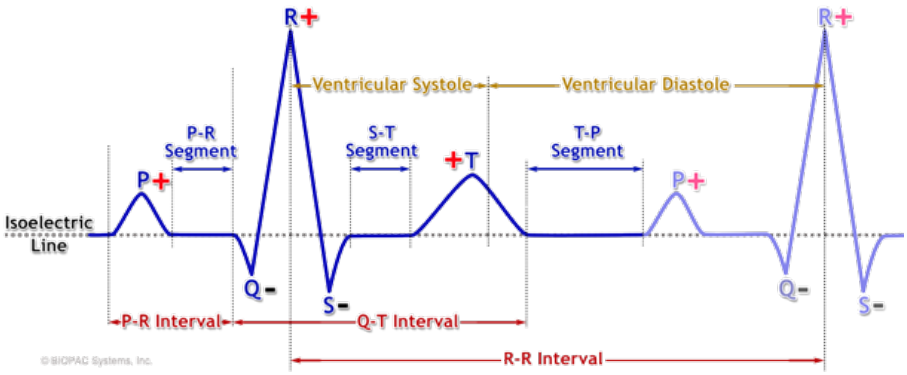


Figure 4: Structure of an ECG signal [19] [20]

2.1.5 Photoplethysmography (PPG) signals

Photoplethysmography (PPG) measures the blood volume changes per pulse. It is an optical and non-invasive technique that can determine a wide range of medical values, including an estimate for BP [16]. Physically, the PPG signal is acquired by measuring the optical signal transmitted through or reflected from the subject's tissue [8]. The PPG sensor consists of two components. The first component is an Light Emitting Diode (LED) to light up the surface of the skin. The second component is a photodetector, which is utilised for measuring the changes in light absorption over a period of time [16] [18].

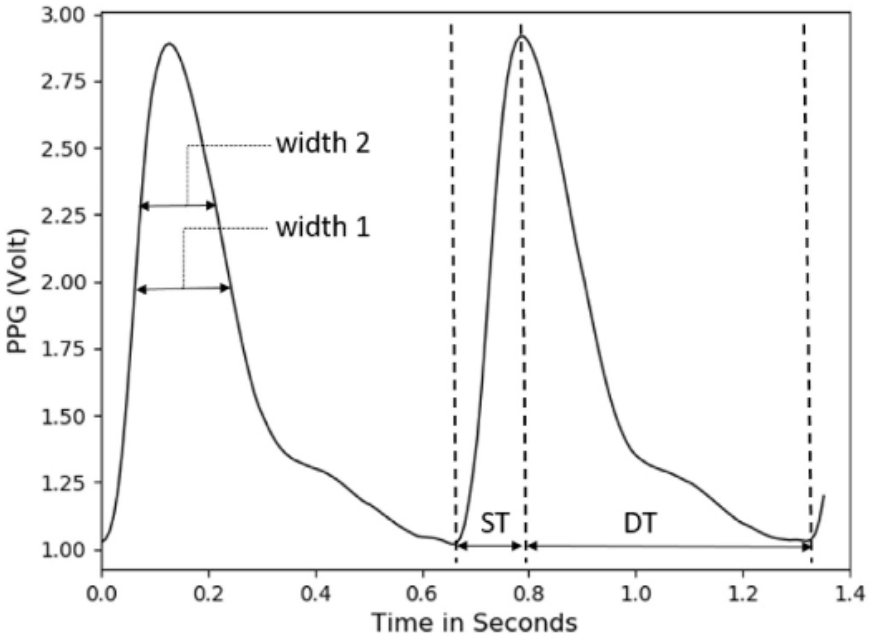


Figure 5: Structure of a PPG Signal [16]

In Figure 5, the four features are the Systolic upstroke Time (ST), Diastolic Time (DT), width at $\frac{1}{2}$ amplitude (width 1) and width at $\frac{2}{3}$ amplitude (width 2) [16]. PPG waveforms have a wide range of temporal features [16]. These features have been utilised in several experimentations, creating models to estimate blood pressure [13].

2.2 Cuff-less methods for deriving BP

Cuff-less methods have great potential in being used to estimate BP. This is because they provide continuous measurements, they cause minimal harm to the patients and they produce BP values over a long period of time [21]. There are three fundamental cuff-less methods which will now be discussed which can be used for deriving BP. These three methods rely on Pulse Transit Time (PTT), Pulse Arrival Time (PAT) and Pulse Wave Velocity (PWV) respectively [22]. These will each now be discussed in more detail.

2.2.1 Pulse Transit Time (PTT)

PTT is the time required for the arterial pressure wave to travel from the left ventricle to a distal arterial site. PTT holds an inverse relationship to blood pressure and as a result it is dependent on arterial compliance, arterial wall thickness, arterial radius, and blood density. PTT is conventionally found with the use of two PPG sensors [8] [9] [16], as indicated in Figure 6.

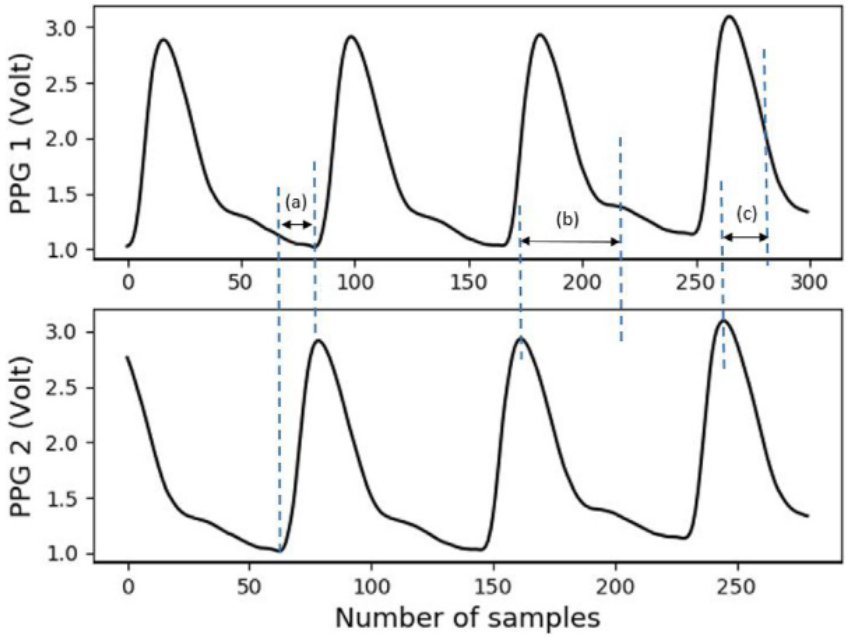


Figure 6: Pulse Transit Time (PTT) visualisation [16]

It is important to note for Figure 6 that the PTT can be measured at different points along the PPG waveforms. (a) represents a foot-to-foot time delay, (b) is a peak-to-dicrotic notch time delay and (c) is a peak to mid-point of the falling edge time delay [16]. As a proof of concept, increasing BP leads to an increase in the tension along the arterial wall tension, which therefore reduces the PTT. Hence, the opposite also applies [18].

2.2.2 Pulse Arrival Time (PAT)

The PAT is the difference in time between the R-peak of the ECG signal and the systolic peak of the PPG signal when measured during the same cardiac cycle, as indicated in Figure 7 [16] [6]. Physically, PAT is the interval in time between the activation of electrical impulses at the heart and the arrival of the pulse wave at a location on the body, such as the finger [23]. PAT is measured using two sensors, an ECG and a PPG sensor [16].

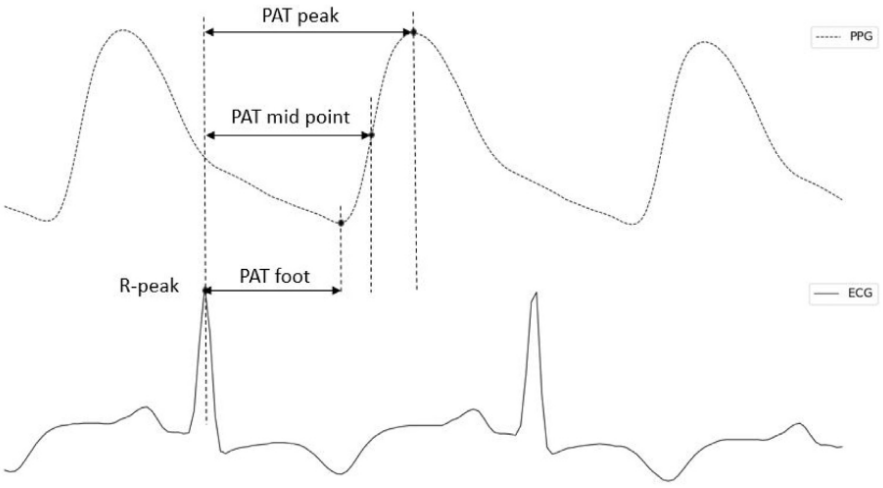


Figure 7: Pulse Arrival Time (PAT) visualisation [16]

The Pre-ejection Period (PEP) delay can also be briefly discussed. PEP is the time needed to convert the electrical signal into a mechanical pumping force and isovolumetric contraction to open the aortic valves,

$$PAT = PTT + PEP \tag{1}$$

2.2.3 Pulse Wave Velocity (PWV)

The PWV calculates the velocity of the pulse wave using two PPG sensors located on the same arterial branch at a known distance apart [13] [16]. The relation between PTT and PWV can be expressed as

$$PWV = \frac{d}{PTT} \tag{2}$$

where d is the arterial distance travelled by the pressure wave. PWV is related to the Young's modulus of the vessel wall by the Moens-Kortweg equation,

$$PWV = \sqrt{\frac{Eh}{\rho d}} \tag{3}$$

where

$$PWV = \text{ Velocity of the pulse wave (m/s)} \tag{4}$$
$$E = \text{ Young's modulus of vessel wall (Pa)} \tag{5}$$
$$h = \text{ vessel thickness (m)} \tag{6}$$
$$\rho = \text{ blood density (kg/}m^3\text{)} \tag{7}$$
$$d = \text{ arterial diameter (m)} \tag{8}$$

The Young's modulus of the vessel wall is then related to the arterial pressure by the Bramwell-Hills equation,

$$E = E_0 e^{\lambda P} \tag{9}$$

where E_0 and λ depend on the thoracic and abdominal aortas and P is the vessel blood pressure (mmHg) [2] [8] [24].

By equating and solving Equations 2 and 3, the final equation for estimated blood pressure is expressed as,

$$P = \frac{1}{\lambda} \ln \left(2r\rho \frac{\Delta X^2}{E_0 h}\right) - \frac{2}{\lambda} \ln \left(PTT\right) \tag{10}$$

where

$$r = \frac{d}{2} = \text{ arterial radius} \tag{11}$$

$$\Delta X = \text{ distance from heart to vessel} \tag{12}$$

2.2.4 Limitations

The blood pressure (BP) can be derived through mathematical models as soon as estimates have been calculated for PTT, PAT and PWV. Although these models are common approaches for BP monitoring in an environment that is non-invasive and cuff-less, there are many challenges to these implementations. As a result, none of these techniques by themselves have been established as a reliable indicator for the estimation of BP.

Firstly, all three of the aforementioned methods require two separate measurements from two synchronised sensor devices. This can be a very inconvenient process for patients who are uncomfortable with this method [23]. In addition, there is a very likely possibility that these sensor devices will have different real-time sampling rates. Their operability depends on rather complicated arterial wave propagation models [16]. In order to be able to continuously measure BP, constant calibration of the methods is required. This is due to individual patients having different physiological parameters [23]. Finally, even with per-person calibration, these models can only provide BP estimation for a short period of time. As a result, this makes the models unreliable for the estimation of BP with every heartbeat [16].

To conclude this chapter, there is a lot of potential in the use of the three above parameters in the estimation of Ambulatory BP. However, there are still overarching limitations which currently hinder the progress of these parameters. As a result, the use of these parameters is not recommended for investigations which involve a large amount of input data. This provides the motivation to use data-driven techniques, as discussed in the next section.

2.3 Neural Networks

In this section the aim is to describe neural networks in a number of stages. Firstly, an overview of why they form part of a field of data-driven techniques is given and how the complexity of the

calculations are affected. Secondly, more information will be provided on the most important neural network architectures relevant to this FYP, and what components make them unique to each other. Finally, all other parameters important to defining neural network architectures will be defined.

2.3.1 Artificial neural networks

Data-driven techniques allow the system to learn and to train from the data rather than to create and solve a system of equations, as seen in the previous section. Due to advancements in technology related to machine learning, there has been a lot of research into neural networks algorithms that can offer continuous BP measurements that are non-invasive and also cuffless [13]. However, in this case BP estimation is motivated by how much data is available to the algorithm [16].

Artificial neural networks (ANNs) are a machine learning method that can be used to estimate blood pressure [13]. ANNs are based on the neural networks found in the human body and aim to replicate their behaviour [24]. The structure of the ANN consists of multiple individual units called neurons. A single neuron is shown in Figure 8.

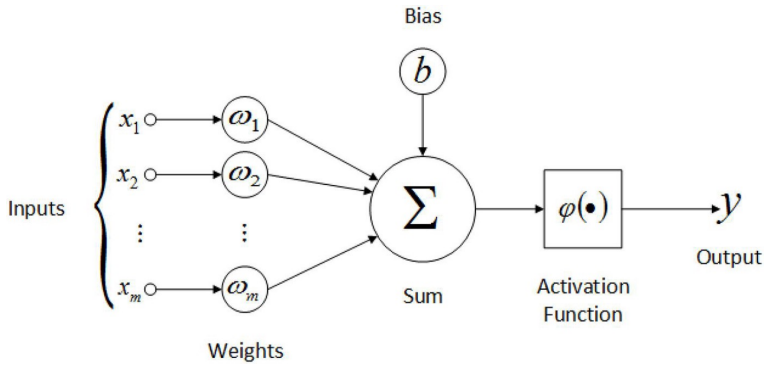


Figure 8: Structure of a neuron [25]

Each neuron has five main building blocks. These are inputs ($\boldsymbol{x}$), weights ($\boldsymbol{\omega}$), transfer function (Σ), activation function ($\phi(\cdot)$) and bias (b) [26]. In order to mathematically express the neuron unit, it is important to first understand its goal. The neuron applies a linear transformation to an m-dimensional input feature vector $\mathbf{x}$ by applying a dot product with the weights $\boldsymbol{\omega}$ and adding a scalar bias b to this dot product. After this, a non-linear activation function $\phi(\cdot)$ is applied to the linear mapping, allowing the neuron to model non-linear relationships. This is expressed mathematically as follows,

$$y = \phi(\sum_i \omega_i x_i + b) = \phi(\boldsymbol{\omega}^T \mathbf{x} + b) \tag{13}$$

where $\boldsymbol{\omega} \in \mathbb{R}^m$ and y, b are scalars. When each of these neurons are connected together with several other neurons across several layers, this forms an ANN, as shown in Figure 9, where each of the neurons are represented by a grey unit. ANNs enable the modelling of more complex relationships than just a single neuron. In addition, the output of the neural network can have as many units as needed depending on the task at hand. In the case of estimating blood pressure

values, this machine learning problem would be treated as a regression problem, which is where the aim is to predict a real and continuous value. Ideally, the network will have two regression values at the output, one for the Systolic Blood Pressure (SBP) and one for the Diastolic Blood Pressure (DBP). The network of a single fully-connected layer is mathematically expressed using Equation 14,

$$\mathbf{y} = \phi(\mathbf{\Omega x} + \mathbf{b}) \tag{14}$$

where $\mathbf{\Omega} \in \mathbb{R}^{N \times M}$ is the weights matrix, $\mathbf{b} \in \mathbb{R}^{N}$ is the bias vector, $\mathbf{y} \in \mathbb{R}^{N}$ is the output vector and $\phi(\cdot)$ performs element-wise non-linear transformations.

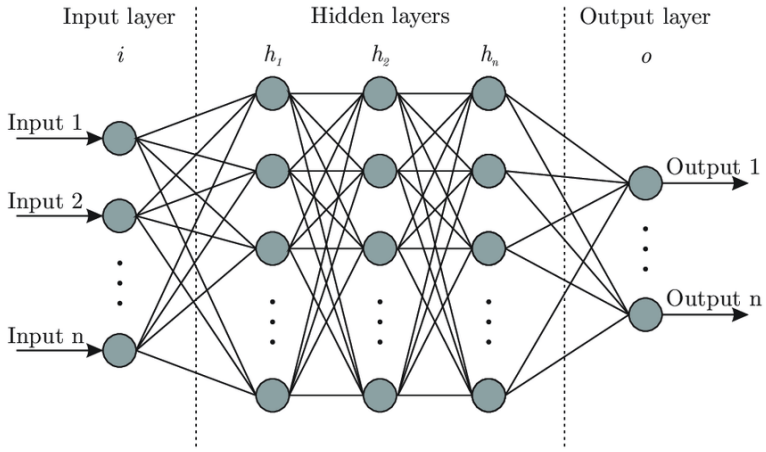


Figure 9: Structure of a multi-layer ANN

Convolutional Neural Networks (CNNs) are a specific class of ANNs. Two-dimensional CNNs were originally developed for image classification problems, where the model learns an internal representation of a two-dimensional input, in a process referred to as feature learning. However, for this FYP the CNNs accept one-dimensional data inputs. The model learns to extract features from sequences of observations and how to map the internal features to different activity types. The benefit of using CNNs for sequence classification is that they can learn from the raw time series data directly, and in turn do not require domain expertise to manually engineer input features. The model can learn an internal representation of the time series data and ideally achieve comparable performance to models fit on a version of the dataset with engineered features. Three high performing one-dimensional CNN models are the AlexNet, ResNet and ResNet-LOSO (Leave One Subject Out) architectures. These architectures will be discussed in more detail in Chapter 3.

2.3.2 Recurrent Neural Networks (RNNs)

Traditional neural networks have found success in many fields, However it has been demonstrated that they cannot capture temporal dependencies in the data, making it unsuitable for signal processing applications. A Recurrent Neural Network (RNN) is a specific type of architecture that is widely used to deal with time-varying data [27]. RNNs contain additional memory states that retain and process information from previous time steps.

RNNs are called recurrent since they apply the same operation to each of the input sequences, with the output of an individual element being dependent on the previous one. Theoretically,

RNNs establish a connection between the actual input and all the previous ones [27]. Although this is assumed, in the practice, RNNs have proven to only remember a limited number of inputs. In other words, RNNs have a memory that allows them to remember previous elements and use their information to deal with the current input [26]. Figure 10 shows the simplest version of an RNN, which can be easily derived from a simple feedforward architecture by adding a single loop:

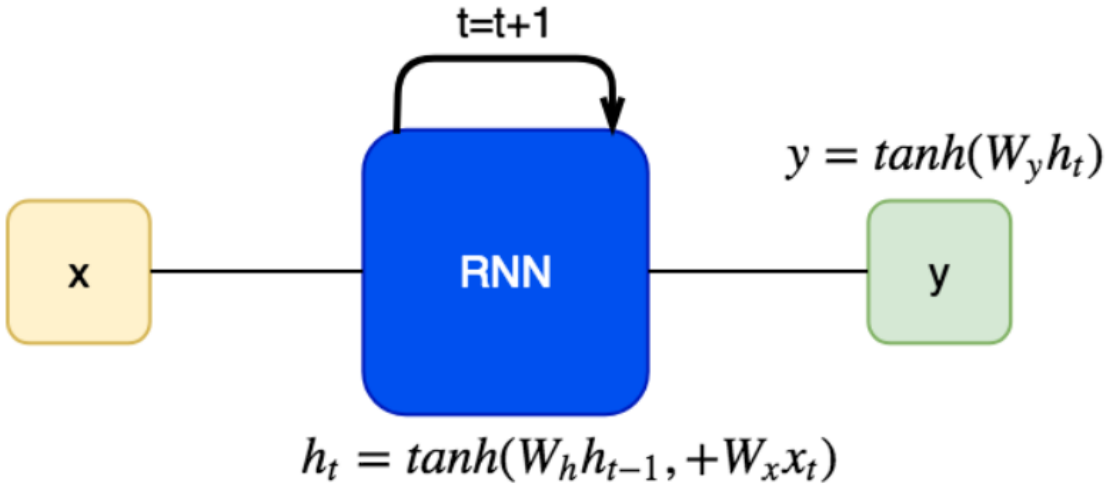


Figure 10: Simplification of a RNN

During training, the hidden state h is iteratively updated based on the input value x and the learned weights W_h and W_x. The final output y is estimated from the current state h_t and the matrix W_y. Although RNN can assure short-term dependencies within the network, simple RNNs become unable to learn to connect information as the gap between past and present information grows [28]. To overcome this limitation, in practical applications LSTM unit is adopted, that is a special RNNs architecture composed of multiple interacting layers.

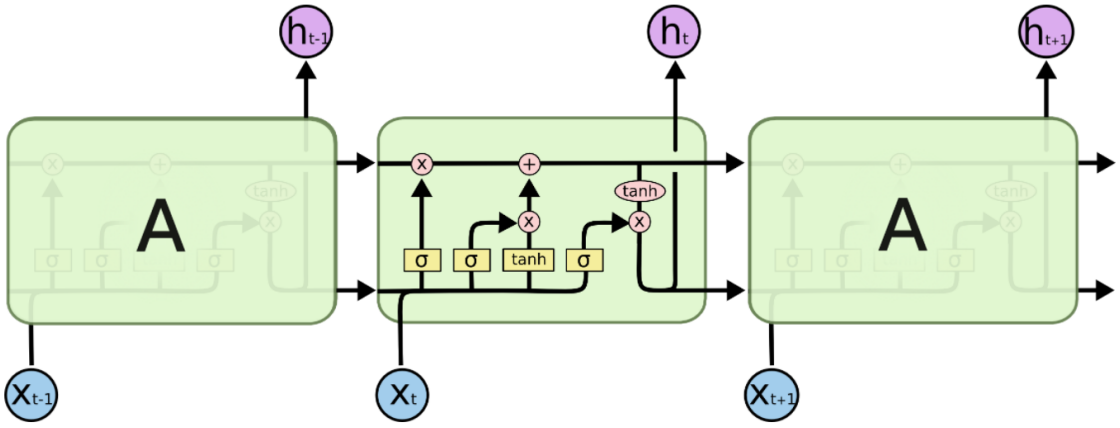


Figure 11: LSTM network

2.3.3 Transformers

Transformers are another class of ANN that provide state-of-the-art solutions for many of the problems previously assigned to RNNs [29]. Sequences can form both the input and the output

of a neural network, examples of such configurations include:

- Vector to Sequence - Image captioning
- Sequence to Vector - Sentiment analysis
- Sequence to Sequence - Language translation

Sequence-to-sequence allows an input sequence to produce an output sequence based on an input sequence. Transformers focus primarily on this sequence-to-sequence configuration.

RNNs and traditional CNNs are outperformed by transformers for several reasons. It was challenging to deal with long-range dependencies between words that were spread far apart in a long sentence. They process the input sequence sequentially one word at a time, which means that it cannot do the computation for time-step t until it has completed the computation for time-step $t|1$. This slows down training and inference. As an aside, with CNNs, all of the outputs can be computed in parallel, which makes convolutions much faster. However, they also have limitations in dealing with long-range dependencies:

In a convolutional layer, only parts of the image (or words if applied to text data) that are close enough to fit within the kernel size can interact with each other. For items that are further apart, you need a much deeper network with many layers. The Transformer architecture addresses both of these limitations. It got rid of RNNs altogether and relied exclusively on the benefits of Attention.

They process all the words in the sequence in parallel, thus greatly speeding up computation.

2.3.4 Activation Functions

Activation functions transform the output of a neural network unit element-wise, allowing it to model non-linear functions. For this project, the estimation of BP is treated as a regression problem. Hence,

2.3.5 Loss Functions

Loss functions are objective functions that the neural network aims to minimize when being trained. For this project, the loss function of concern is the Mean Squared Error (MSE) loss function, which is defined in Equation 15.

$$l_{\text{MSE}}\left(\mathbf{y}, \hat{\mathbf{y}}\right) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - \hat{\mathbf{y}}_\mathbf{i})^2 \tag{15}$$

where N is the number of training examples, $\mathbf{y}_i$ is the target output vector, and $\mathbf{y}_{\hat{y}_i}$ is the predicted output vector. The MSE is effective for ensuring that the trained model has no outlier predictions with huge errors, since the MSE puts larger weight on theses errors due to the squaring operation of the function.

2.3.6 Neural Network Training

In the same manner for any other machine learning algorithm, neural networks aim to learn an underlying pattern present in the data by minimizing an error measure defined by a loss

function given some sample data (a process known as training). Formally, this problem is done by finding the optimal weights of the model per layer as defined in Equation 16, where $\hat{\mathbf{y}}$ is the estimate of the true label $\mathbf{y}$ and l is the loss function.

$$\mathbf{\Omega}_{opt} = \arg\min_{\mathbf{\Omega}}\{\mathbb{E}(l(\mathbf{y}, \hat{\mathbf{y}}))\} \tag{16}$$

TALK ABOUT THE FOLLOWING THINGS:

- Overfitting
- BatchNorm
- Pooling
- Dropout
- Complexity

To conclude this chapter, an overview has been given on the underlying theory behind the most popular neural network architectures. The main tradeoffs between these architectures is the decision between less complexity, with CNN and RNN architectures, and with maximal estimation accuracy, with the Transformer encoder architecture. As a result, it is necessary to assess all existing methods for cuffless blood pressure estimation, so that it is possible to decide which is the best in achieving the objectives of the FYP.

2.4 Literature Review

- what comments do you have on the results you've tabulated? - You mention factors were considered.. why? and when you considered them, what about them? Why is it important? - What should the reader be left with?

What are the advantages and disadvantages?

Currently, the literature review is below average. You need to work on this.

This chapter provides a detailed account of the literature review conducted for this FYP. The literature search equation used will first be discussed, followed by an explanation of the PRISMA flow diagram and how it was used to benefit this literature review. To help the reader, a table of the scientific papers used in this project is provided. Finally, a critical analysis will be given on the literature review and what can be concluded as a result. The aim is to be able to justify which is the most feasible method for estimating cuffless blood pressure values.

2.4.1 Survey Equation

At the beginning of the FYP, the only information provided was the FYP mission statement (see Appendix Item 10.1) and two published papers, *A review of machine learning techniques in photoplethysmography for the non-invasive cuff-less measurement of blood pressure* [16] and *Continuous Blood Pressure Estimation From Electrocardiogram and Photoplethysmogram During Arrhythmias* [21]. These resources pserved as an introduction to both the medical background and machine learning knowledge for myself. After reviewing this information, the next step was to perform an informal search of literature databases using keywords extracted from the FYP brief. These keywords can be divided into two fields:

Background knowledge

- "Ambulatory"
- "Blood Pressure"
- "Electrocardiogram"
- "Photoplethysmography"
- "Wearable technology"

Implementation strategy

- "Accuracy"
- "Algorithm"
- "Computational complexity"

These keywords were entered into 3 official literature databases, as shown in Table 2.

Table 2: Official online databases used to conduct the literature review [30]

Literature database	Description
ACM Digital Library	The digital library of the Association for Computing Machinery
Engineering Village	Database platform for Physics, Electrical Engineering, Electronics and Computing
IEEE Xplore	Digital library containing full text of IEEE journals, conference/meeting papers and standards
National Library of Medicine (Pubmed)	Biomedical and life sciences literature

The ACM Digital Library was chosen to help deepen my understanding of the existing algorithms and programming methods available for estimating cuffless blood pressure. Engineering Village provides a wide variety of both signal processing and machine learning based methods for cuffless estimation, whereas the IEEE Xplore library gives a detailed insight into machine learning based methods for cuffless estimation. Finally the Pubmed database was chosen to help deepen my understanding of the medical knowledge required to perform the literature review.

This informal search enabled a clearer understanding of how the relevant published literature phrased their titles. Based on the findings of the informal search, the following literature survey search equation was used to identify the literature that best fits the needs of the FYP requirements. The equation chosen was:

- (Extraction OR Estimation OR Review) AND (Blood OR Arterial OR Ambulatory OR Cuffless) AND (Pressure) AND (ECG OR PPG) AND (Machine Learning OR Signal Processing).

Hence, this equation was entered into the four databases displayed in Table 2.

2.4.2 PRISMA checklist

After applying the chosen equation to the four databases in Table 2, the next step was to systematically eliminate all papers that were not relevant to the aims of the FYP. The Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) checklist was used to identify the motivations, methods and findings of all published articles relevant to this FYP [31]. The process is illustrated in Figure 12.

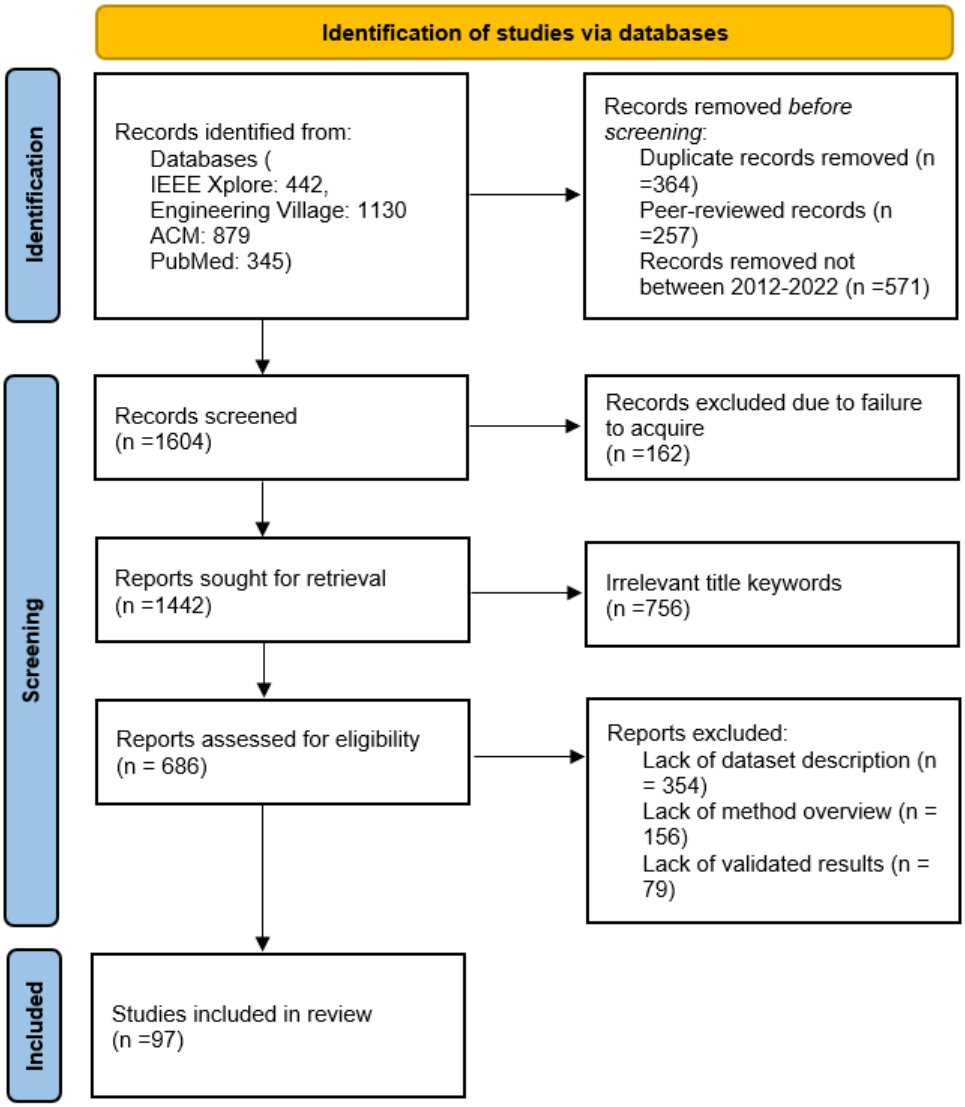


Figure 12: PRISMA checklist flow diagram (correct to 2020 guidelines) (NEED TO SORT OUT NUMBERS)

2.4.3 Literature survey table

As a reference, the original literature survey matrix can be found on the Github repository [32]. Firstly, in Table 3, a simplified literature survey has been detailed out for the best performing methods which do not employ machine learning methods.

Table 3: Overview of performance of the best non-invasive non-ML cuff-less methods for measuring BP

Study	**Source**	**No. Subjects**	**Age**	**Implementation**	**MAE SBP**
[33]	ECG, PTT-CP	10	24-63	Numerical solution	± 5.93
[34]	ECG	5	N/A	Analytical solution	9 ± 5.6
[35]	PPG	16	18-48	Frequency analysis	0.8 ± 7
[36]	ECG, PPG, PTT	N/A	N/A	Analytical solution	7.49 ± 8.8
[37]	PTT	127	N/A	Wavelet transforms	± 7.63
[38]	PTT, PPG	27	21-29	Analytical solution	-0.37 ± 5.21

Table doesn't fit on the page. Consider either turning it landscape or redesigning the table to fit.

Redesigning could be making a key for different sources and/or the method. E.g ECG = filled square, PPG = filled circle.

If you are abbreviating or using symbols, state clearly in your table captions.

Table 4: Overview of performance of the best non-invasive ML cuff-less methods for measuring BP

Study	**Source**	**No. Subjects**	**Age**	**Method**	**MAE SBP**
[24]	ECG, PPG	14 males	17-43	ANN	7.99 ± 10.34
[39]	PPG	65	22-65	Wavelet, SVM	5.1 ± 4.34
[40]	PPG	MIMIC II	Adults	Linear Reg., ANN, SVM	13.84 ± 17.56
[3]	ECG	51	16-83	Complexity analysis + ML	7.72 ± 10.22
[9]	PPG	72	N/A	ANN (MLP)	4.02 ± 2.79
[13]	ECG, PPG	MIMIC II	Adults, neonatal	ANN (150 neurons)	5.76 ± 6.39
[8]	ECG, PPG	39	20-100	ANN-LSTM	1.10
[41]	PTT, ECG, PPG	MIMIC I	N/A	SVM, Lin Reg.	3.27 ± 5.52
[42]	PTT	250	MIMIC I	ANN-RBM	3.70

In addition to the information displayed in the tables, the following comments can be made in reference to the complete Literature Survey table [32]:

- **Feasibility for usage in wearable devices** is an additional column, which was included to see if any papers discussed integrating their proposed methods onto wearable devices. In general, the most feasible solutions came from the non-ML based methods

2.4.4 Critical analysis of literature survey table

In this section, the aim is to provide a detailed analysis of all the papers provided in the literature review table.

Firstly, it is clear that the dominant method for cuffless blood pressure measurement is through Machine Learning - based techniques. This is indicated by the fact that the majority of the papers employ some form of neural network architecture, such as ANNs, RNNs and LSTMs. In addition, the Mean Absolute Error (MAE) values are shown to be significantly lower for these machine learning methods over the traditional mathematical and signal processing based methods.

Secondly, it is clear that the most recent Machine Learning techniques only utilise the PPG signal. The justification for this design choice is that the PPG signal has sufficient physiological features in both the time and frequency domains, such that the features of other signals, such as the ECG, are not required for accurate estimation of blood pressure.

- Talk about the databases used
- Most prominent ML research is in Transformers, but not a clear frontrunner, need to analyse multiple neural network methods on the same dataset

2.4.5 Conclusions of literature survey

In this chapter, the literature survey process has been detailed and as a result, the next stage is to design an implementation for cuffless blood pressure estimation using only PPG signals. In order to achieve the best accuracy in estimation, the review has shown that neural network based methods are the best choice. The issue of the complexity of neural networks has been discussed and will be a relevant factor in the design process of the proposed implementation.

3 Analysis and Design

In this chapter, the aims are to:

- Provide a high-level overview of the design of the system to estimate blood pressure values from PPG signals
- Describe the specifications of each stage of the system based on the findings of the literature review in Chapter 2
- Discuss any changes or justifications for design choices made for the system blocks where appropriate

3.1 High Level overview of proposed design

Figure 13 illustrates the high-level design specification for the cuffless estimation of blood pressure from PPG signals. The diagram describes each discrete stage that is required.

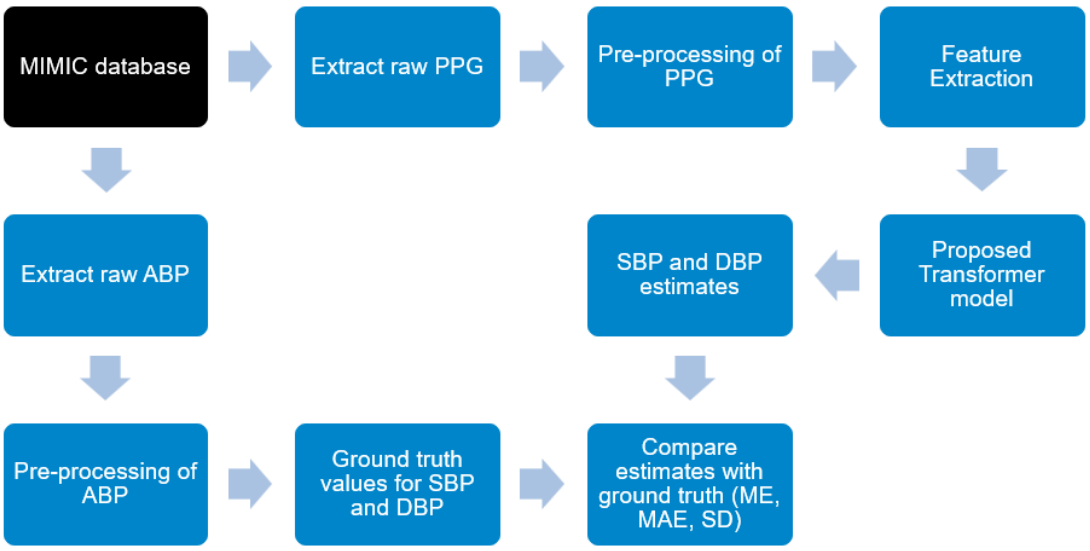


Figure 13: High-level block diagram of blood pressure cuffless estimation from PPG

The following subchapters will explain each stage in more detail.

3.2 Choice of database

As previously discussed in the literature review in Chapter 2, the MIMIC (Multi-parameter Intelligent Monitoring for Intensive Care) database has been chosen for the implementation. The data has been recorded from patient monitors in the medical, surgical, and cardiac intensive care units of Boston's Beth Israel Hospital. The MIMIC Database has records for 72 ICU patients. Referring back to the motivation for this FYP, the early detection and prevention of hypertension and other Cardiovascular diseases (CVDs) is crucial. Therefore, testing was only performed on patients with CVDs or heart-related issues, which was a data subset of 12

patients. The age range of the patients is 52 to 85 years. In addition, the data is from 8 males and 4 females. The data obtained from the bedside monitors are divided into files each containing 10 minutes of recorded signals, which can then be assembled without gaps to form a continuous recording [43]. Both the PPG and ABP signals in the MIMIC database are sampled at a frequency of 125 Hz. Each patient record contains a minimum of 73 and a maximum of 412 individual files. The patients contain one of the following heart-related diseases:

- **Congestive Heart Failure** (CHF). This refers to patients who suffer a chronic progressive condition that affects the pumping power of your heart muscle
- **Postoperative Coronary Artery Bypass Graft** (CABG). This refers to patients recovering from a surgical procedure to restore normal blood flow to an obstructed coronary artery
- **Myocardial Infarction** (MI) / **Cardiogenic shock**. This refers to patients who have suffered heart attacks or cardiac shock

NEED REFERENCES HERE!

3.3 Choice of programming language

Python is used as the sole programming language for this project. Python has a wide variety of easy to use and powerful libraries [44]. The scientific libraries from Python that are used for this project are `numpy` [45] and `pandas` [46]. In addition, the machine learning libraries used are `tensorflow` [47], `keras` [48] and `scikit-learn` [49]. In addition the `heartpy` [50] and `wfdb-python` [51] packages were installed, which are libraries of tools for reading, writing, and processing Waveform-Database (WFDB) signals and annotations.

MATLAB was also considered as a potential programming language to use, due to it having a wide range of signal processing and machine learning add-on toolboxes. However Python has been shown to offer a wider set of choices in graphics packages and toolsets, such as through `matplotlib` [52], and it also produces more compact and readable code. For this FYP, Python will be used in the Google Colaboratory environment in the form of a Jupyter notebook, as there is access to to high-performance Graphics Processing Units (GPUs) on Google Cloud for training using machine learning.

3.4 Extraction of raw signal data

For this implementation, it is necessary to extract the raw signal data for two signals, the ABP and PPG. Both signals are first extracted from the Physionet website using the `wfdb` package.

- Use heartpy library (hp.process)
- Despite being measured using gold-standard invasive, still needs preprocessing. TALK ABOUT ALL STEPS!

3.5 Pre-processing of PPG

- hp.process
- Butterworth filter
- Z-normalisation
- Additional sanity checks using custom SBP and DBP min and max values

The Z-score normalisation equation is applied to the PPG signal,

$$Z_i = \frac{(PPG_i - \mu_{PPG_i})}{\sigma_{PPG_i}} \tag{17}$$

where Z_i is the Z-score normalised PPG signal for a particular window i, PPG_i is the raw PPG signal, μ is the mean of the PPG signal and σ is the standard deviation.

3.6 Feature extraction

- Based on Literature review, machine learning methods are favouring automated feature extraction
- Hence, this method will segment the PPG signal into discrete windows and feed this into the network
- However, there is still prominence among the handcrafted feature methods
- Provide an additional exploration to add the first and second derivatives of the PPG to see if this affects the accuracy for any model

3.7 Machine Learning models

- Function based for each ML model
- Using tensorflow and keras

3.8 Error metrics

The two considered error calculations used in this experimentation are the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). They are defined by the following equations,

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|a_{i_M} - b_{i_M}| \tag{18}$$

$$RMSE = \frac{1}{N}\sqrt{\sum_{i=1}^{N}|a_{i_M} - b_{i_M}|^2} \tag{19}$$

In the context of BP estimation, b_{i_M} and a_{i_M} represent the true value and BP estimate respectively for the Mth element of the time sequence.

4 Implementation

In this chapter, the aim is to:

- Provide a detailed description of the Python code used to estimate cuffless blood pressure from PPG signals
- Discuss any changes or justifications made to the code

4.1 Stage 1: Filtering the dataset

As discussed in Chapter 3, the MIMIC Database includes data recorded for 72 ICU patients, ranging from patient indexes '037' to '485'. Firstly, it was necessary to check which of these patients contained signal data for the PPG and ABP channels. In Python, this was performed by inspecting the `wfdb.rdrecord.sig_name` string array for each patient record, and checking to see if the ABP and PPG channels were present (indicated by the `ABP` and `PLETH` keywords). As a result, the following patients were excluded from the dataset:

- '037'
- '208'
- '209'
- '210'
- '222'
- '262'
- '291'
- '405'
- '413'
- '415'
- '450'

Hence, there was now data available from 61 ICU patients.

4.2 Stage 2: Decision on the window length

The next stage is to split up the PPG signal into discrete intervals of the same time interval. In order to justify the window length used, it was necessary to analysis the waveform structure of the PPG for particular patients.

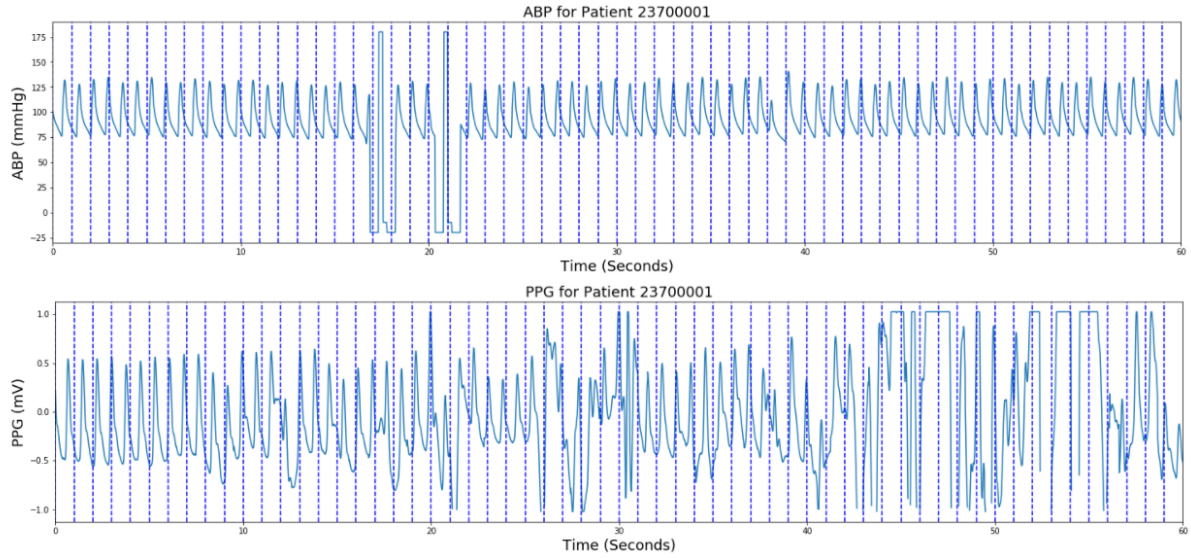


Figure 14: Decision to window the signal

4.3 Stage 3: Choosing the patients

Why a subset? Why not all? If you're doing a split, explain this. Are you only using 8 patients out of the 90?

Based on the decision to use a window length of 1 second, it was necessary to take a subset of the dataset. WHY? This was because a small window length means that a lot of datapoints are available for training already, so there is no need to acquire data from all 61 patients. By considering that the motivation of the FYP is to investigate the detection and prevention of hypertension, the decision was to examine the records of patients suffering with Cardiovascular diseases or other heart-related illnesses. Hence, a subset of the MIMIC I database was chosen, as illustrated in Table 5.

Table 5: Characteristics of the chosen 12 patients from the MIMIC-I database

Patient record number	Age	Gender	Health condition
418	52	M	CHF/pulmonary edema
480	52	M	Post-op CABG
237	63	F	MI/cardiogenic shock
477	67	M	Post-op CABG
466	70	M	CHF/pulmonary edema
476	72	F	Post-op CABG
225	73	M	CHF/pulmonary edema
230	75	F	CHF/pulmonary edema
213	82	F	CHF/pulmonary edema
212	84	M	CHF/pulmonary edema
456	84	M	Post-op CABG
417	85	M	CHF/pulmonary edema

Table 6: Number of samples available after initial preprocessing from the MIMIC database subset

Patient record number	10-minute Samples
418	0
480	30
237	17
477	3
466	139
476	67
225	17
230	38
213	42
212	143
456	83
417	0

4.4 Stage 4: Extracting the ground truth blood pressure values

The ground truth Systolic and Diastolic blood pressure values are calculated by taking the respective maxima and minima of the arterial blood pressure signal within each window of the signal.

THEN TALK ABOUT PREPROCESSING APPLIED TO ABP

4.5 Stage 5: Preprocessing the PPG signal

4.6 Stage 6: Processing of data before training

4.7 Stage 7: Overview of Neural Network models used

4.7.1 CNN AlexNet model

What type of information should the reader gather from this figure? It needs explaining and annotations/labelling.

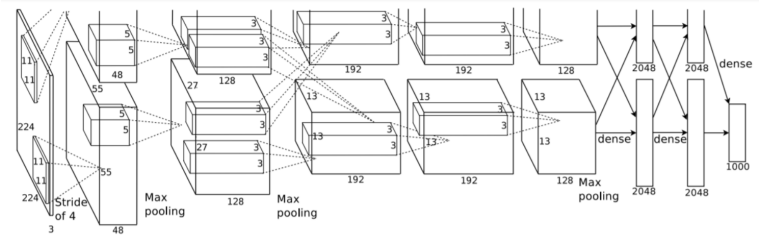


Figure 15: Overview of the AlexNet architecture

4.7.2 CNN ResNet model

4.7.3 ResNet with LOSO model

4.7.4 LSTM model

4.7.5 Proposed Transformer Encoder model

4.8 Stage 8: Training the model

PARAMETERS used

4.9 Stage 9: How the error is evaluated

5 Testing Plan

-

6 Overview of Results

In this chapter, the aim is to:

- Apply the objectives set out in the Testing Plan in Chapter 5
- Compare the performance of the five neural network architectures for cuffless estimation of blood pressure using PPG
- Assess whether the performance of the neural networks is estimate blood pressure effectively

1. Ensure all results have units (if it doesn't have units, it's A.U for arbitrary units)
2. If you have sub figures or diagram, label them a, b, c and include a short description in the caption.

6.1 Overview of testing parameters

- KEEP ADDING MORE STUFF
- For all testing performed in Google Colaboratory, the GPU being used to compute inference times was the Tesla T4 Persistence-M GPU
- Learning rate $= 0.005$
- The Adam optimizer was used for all experiments
- Loss function?

6.2 Training and Validation MAE for the deep learning architectures

6.2.1 AlexNet

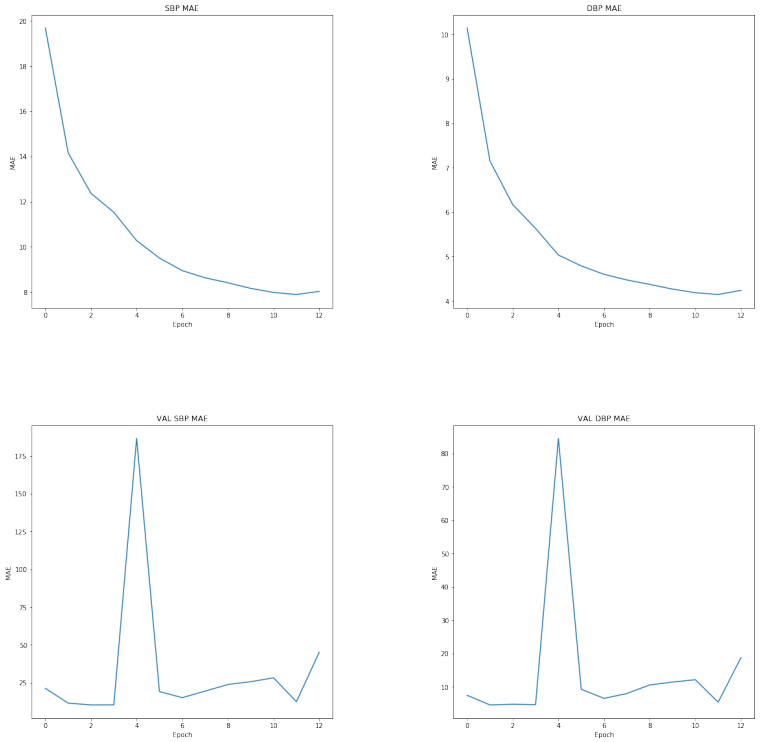


Figure 16: MAE of SBP and DBP for AlexNet architecture

6.2.2 ResNet

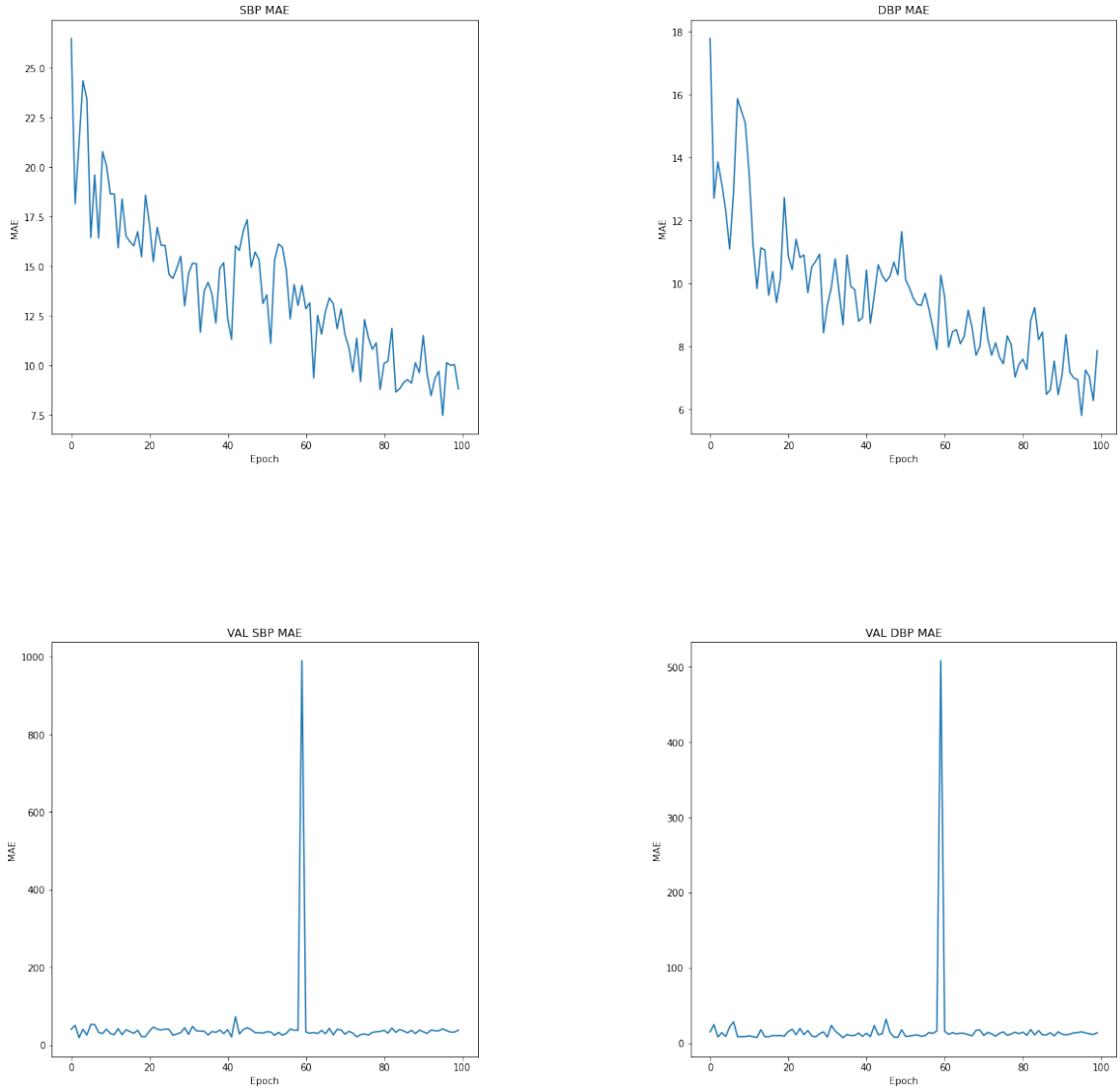


Figure 17: MAE of SBP and DBP for ResNet architecture

6.2.3 ResNet with LOSO

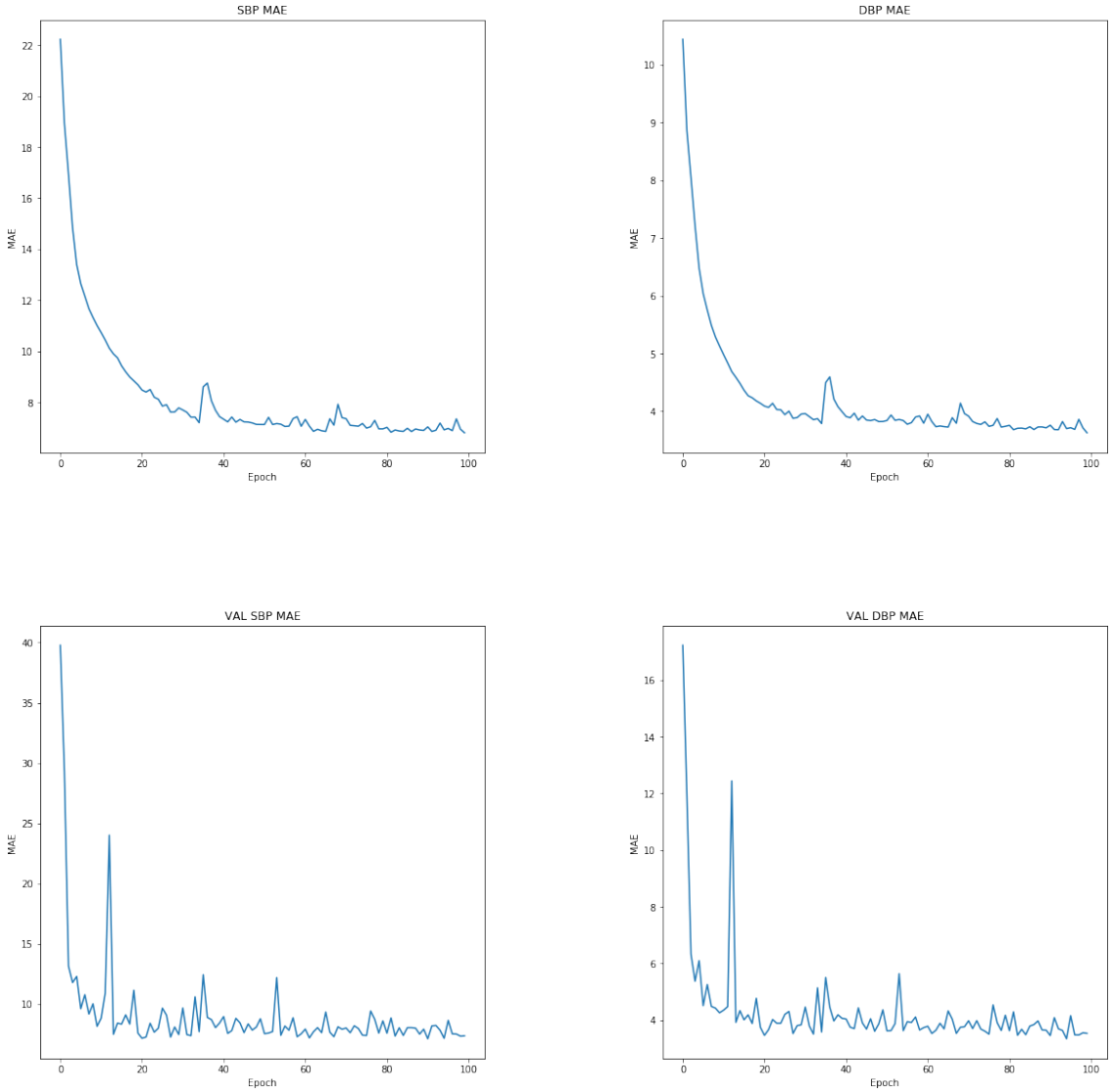


Figure 18: MAE of SBP and DBP for ResNet-LOSO architecture

6.2.4 LSTM

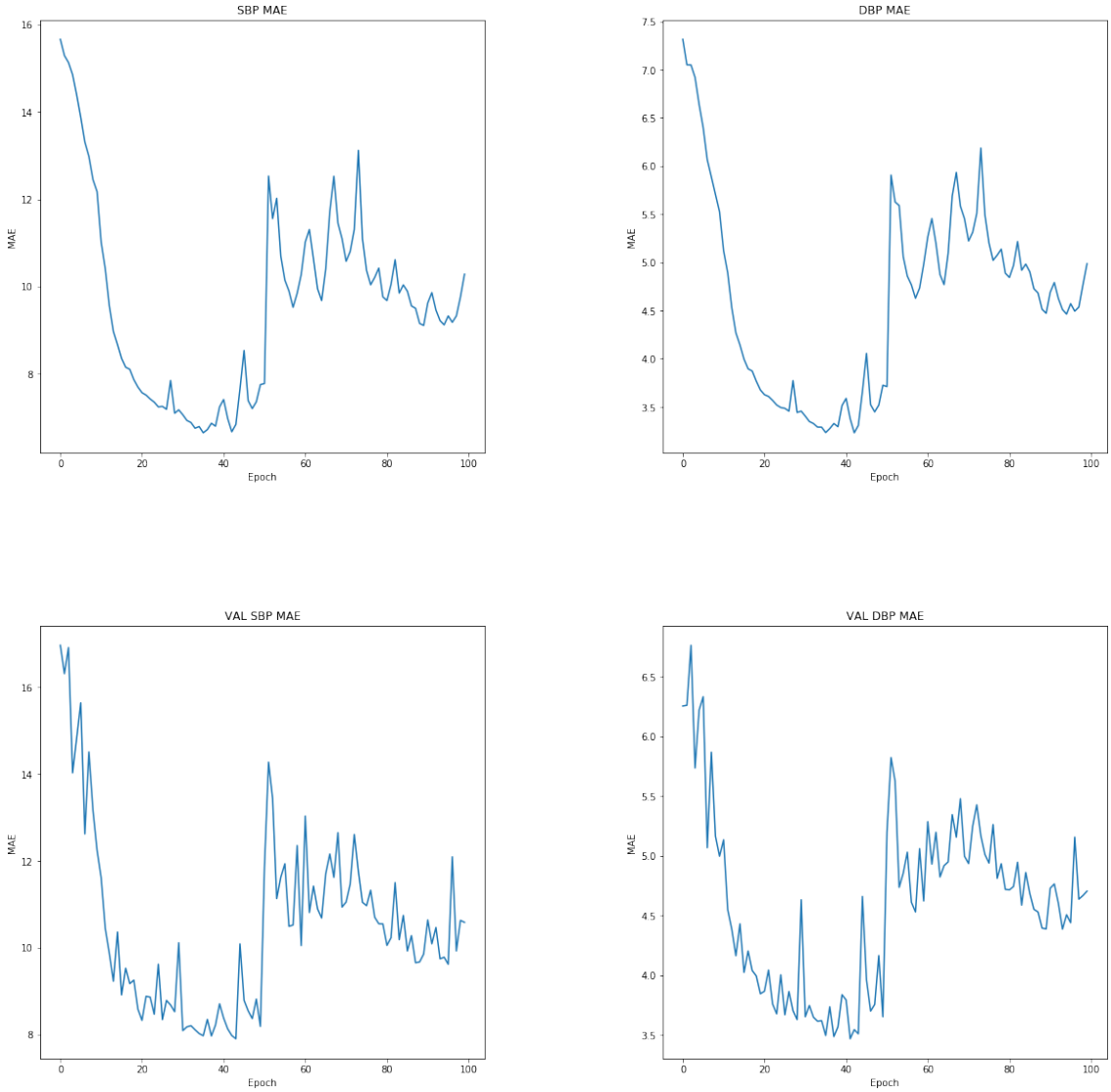


Figure 19: MAE of SBP and DBP for LSTM architecture

6.2.5 Transformer Encoder

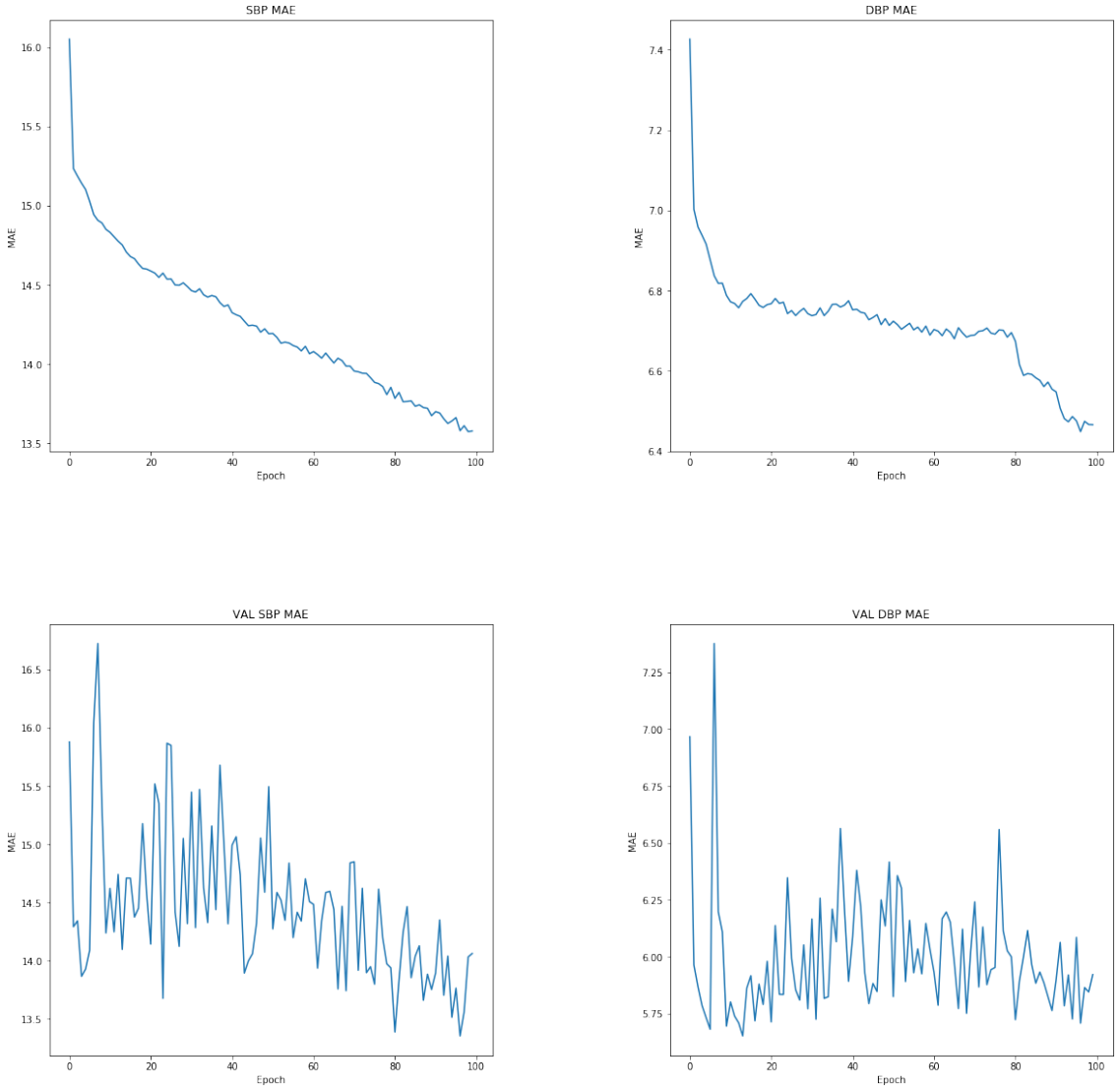


Figure 20: MAE of SBP and DBP for the Transformer encoder architecture

7 Evaluation of results

- All models except AlexNet have acceptable validation errors
- RNN/LSTM perform best on this data, not Transformer as expected (for no derivatives)
- Derivative?

8 Conclusions and Further Work

8.1 Summary of project achievements

- Design choices: Aim was to minimize complexity of the neural network model in order to effectively monitor BP
- What was most difficult: 2 parts to this: Firstly the choice of how to preprocess the signal for effective analysis. This was solved by surveying the relevant literature and assessing the most feasible set of steps. Secondly, designing the ideal neural network architectures
- What I learned: Despite what was expected from the theory, the CNN Resnet performs better on the PPG signal data than the LSTM networks
-

8.2 Future work

- Discuss the transformer papers and the attention mechanism. The main reason the transformer model was not explored for this paper is due to its high model complexity. However, this could be tested on in the future and see if the results are any/much better than the Resnet/LSTM models
-

References

[1] Manuja Sharma et al. "Cuff-Less and Continuous Blood Pressure Monitoring: A Methodological Review". In: *Technologies* 5 (2 May 2017), p. 21. ISSN: 2227-7080. DOI: 10.3390/technologies5020021 (cit. on pp. 1, 6).

[2] G. Janjua et al. "Wireless chest wearable vital sign monitoring platform for hypertension". In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* (Sept. 2017), pp. 821–824. ISSN: 1557170X. DOI: 10.1109/EMBC.2017.8036950 (cit. on pp. 1, 3, 11).

[3] Monika Simjanoska et al. "Non-invasive blood pressure estimation from ECG using machine learning techniques". In: *Sensors (Switzerland)* 18 (4 Apr. 2018). ISSN: 14248220. DOI: 10.3390/s18041160 (cit. on pp. 1, 4, 20, 42).

[4] Kazuomi Kario et al. "Guidance on ambulatory blood pressure monitoring: A statement from the HOPE Asia Network". In: *Journal of Clinical Hypertension* 23 (3 Mar. 2021), pp. 411–421. ISSN: 17517176. DOI: 10.1111/jch.14128 (cit. on pp. 1, 4).

[5] Wan SW Zaki et al. "Blood pressure estimation from photoplethysmogram and electrocardiogram signals using machine learning". In: *University of Nottingham* (2018) (cit. on pp. 1, 6).

[6] Malikeh Pour Ebrahim et al. "Blood Pressure Estimation Using On-body Continuous Wave Radar and Photoplethysmogram in Various Posture and Exercise Conditions". In: *Scientific Reports* 9 (1 Dec. 2019). ISSN: 20452322. DOI: 10.1038/s41598-019-52710-8 (cit. on pp. 1, 9).

[7] Dylan M. Bard, Jeffrey I. Joseph, and Noud van Helmond. "Cuff-Less Methods for Blood Pressure Telemonitoring". In: *Frontiers in Cardiovascular Medicine* 6 (Apr. 2019). ISSN: 2297055X. DOI: 10.3389/fcvm.2019.00040 (cit. on pp. 1, 3, 42–44).

[8] Md. Sayed Tanveer and Md. Kamrul Hasan. "Cuffless Blood Pressure Estimation from Electrocardiogram and Photoplethysmogram Using Waveform Based ANN-LSTM Network". In: *Bangladesh University of Engineering and Technology* (Nov. 2018). URL: http://arxiv.org/abs/1811.02214 (cit. on pp. 3, 6–8, 11, 20).

[9] Ludi Wang et al. "A novel neural network model for blood pressure estimation using photoplethesmography without electrocardiogram". In: *Journal of Healthcare Engineering* 2018 (2018). ISSN: 20402309. DOI: 10.1155/2018/7804243 (cit. on pp. 3, 4, 8, 20).

[10] Wayne N. Dillon. *High Blood Pressure and Hypertensive Heart Disease*. 2015 (cit. on p. 4).

[11] Monika Simjanoska et al. "ECG-derived blood pressure classification using complexity analysis-based machine learning". In: *HEALTHINF 2018 - 11th International Conference on Health Informatics, Proceedings; Part of 11th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2018* 5 (2018), pp. 282–292. DOI: 10.5220/0006538202820292 (cit. on pp. 4, 5, 7).

[12] Qi Fang Huang et al. "Ambulatory Blood Pressure Monitoring to Diagnose and Manage Hypertension". In: *Hypertension* (2021), pp. 254–264. ISSN: 15244563. DOI: 10.1161/HYPERTENSIONAHA.120.14591 (cit. on p. 4).

[13] Lorena Pradenas. "A Novel Non-Invasive Estimation of Arterial Blood Pressure from Electrocardiography and Photoplethysmography Signals using Machine Learning". In: *Biomedical Journal of Scientific & Technical Research* 30 (1 Sept. 2020). DOI: 10.26717/bjstr.2020.30.004883 (cit. on pp. 5, 6, 8, 10, 12, 20).

[14] Tasbiraha Athaya and Sunwoong Choi. "An Estimation Method of Continuous Non-Invasive Arterial Blood Pressure Waveform Using Photoplethysmography: A U-Net Architecture-Based Approach". In: *Sensors* 21.5 (5 Mar. 2021), pp. 1–18. ISSN: 1424-8220. DOI: 10.3390/s21051867. URL: https://www.mdpi.com/1424-8220/21/5/1867 (cit. on p. 5).

[15] Difference Between. *Difference Between Systolic and Diastolic*. Jan. 2021 (cit. on p. 6).

[16] C. El-Hajj and P. A. Kyriacou. "A review of machine learning techniques in photoplethysmography for the non-invasive cuff-less measurement of blood pressure". In: *Biomedical Signal Processing and Control* 58 (Apr. 2020). ISSN: 17468108. DOI: 10.1016/j.bspc.2020.101870 (cit. on pp. 6–12, 16, 42).

[17] Alexander A. Leung et al. "Hypertension Canada's 2016 Canadian Hypertension Education Program Guidelines for Blood Pressure Measurement, Diagnosis, Assessment of Risk, Prevention, and Treatment of Hypertension". In: *Canadian Journal of Cardiology* 32 (5 May 2016), pp. 569–588. ISSN: 0828282X. DOI: 10.1016/j.cjca.2016.02.066 (cit. on p. 6).

[18] Sarvesh Kumar and Shahanaz Ayub. "Estimation of blood pressure by using electrocardiogram (ECG) and photo-plethysmogram (PPG)". In: *Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015* (Sept. 2015), pp. 521–524. DOI: 10.1109/CSNT.2015.99 (cit. on pp. 7, 9).

[19] Wikipedia. *Electrocardiography*. June 2022 (cit. on p. 7).

[20] Wikipedia. *QRS Complex*. June 2022 (cit. on p. 7).

[21] Zeng Ding Liu et al. "Continuous Blood Pressure Estimation From Electrocardiogram and Photoplethysmogram During Arrhythmias". In: *Frontiers in Physiology* 11 (Sept. 2020). ISSN: 1664042X. DOI: 10.3389/fphys.2020.575407 (cit. on pp. 8, 16).

[22] Ross Nye, Zhe Zhang, and Qiang Fang. "Continuous non-invasive blood pressure monitoring using photoplethysmography: A review". In: *4th International Symposium on Bioelectronics and Bioinformatics, ISBB 2015* (Dec. 2015), pp. 176–179. DOI: 10.1109/ISBB.2015.7344952 (cit. on p. 8).

[23] Da Un Jeong and Ki Moo Lim. "Combined deep CNN-LSTM network-based multitasking learning architecture for noninvasive continuous blood pressure estimation using difference in ECG-PPG features". In: *Scientific Reports* 11 (1 Dec. 2021). ISSN: 20452322. DOI: 10.1038/s41598-021-92997-0 (cit. on pp. 9, 11).

[24] Sen Yang et al. *Blood pressure estimation with complexity features from electrocardiogram and photoplethysmogram signals*. URL: http://creativecommons.org/licenses/by/4.0 (cit. on pp. 11, 12, 20).

[25] Hussein Almusawi and Abbas Burhan. "Evaluation of the Productivity of Ready Mixed Concrete Batch Plant Using Artificial Intelligence Techniques". In: *IOP Conference Series: Materials Science and Engineering* 901 (Sept. 2020), p. 012020. DOI: 10.1088/1757-899X/901/1/012020 (cit. on p. 12).

[26] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. 2017 (cit. on pp. 12, 14).

[27] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. May 2015 (cit. on pp. 13, 14).

[28] Colah. *Understanding LSTM Networks*. Aug. 2015 (cit. on p. 14).

[29] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762 (cit. on p. 14).

[30] Imperial College London. *Electrical and Electronic Engineering search tools*. June 2022 (cit. on p. 17).

[31] Matthew J Page et al. "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews". In: *BMJ* 372 (2021). DOI: 10.1136/bmj.n71. eprint: https://www.bmj.com/content/372/bmj.n71.full.pdf. URL: https://www.bmj.com/content/372/bmj.n71 (cit. on p. 18).

[32] Arijit Bhattacharyya. *Final Year Project Literature Survey matrix*. Jan. 2022 (cit. on pp. 18, 21).

[33] Saif Ahmad et al. "Electrocardiogram-assisted blood pressure estimation". In: *IEEE Transactions on Biomedical Engineering* 59 (3 Mar. 2012), pp. 608–618. ISSN: 00189294. DOI: 10.1109/TBME.2011.2180019 (cit. on p. 19).

[34] Zhihao Chen et al. "Noninvasive monitoring of blood pressure using optical Ballistocardiography and Photoplethysmograph approaches". In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* (2013), pp. 2425–2428. ISSN: 1557170X. DOI: 10.1109/EMBC.2013.6610029 (cit. on p. 19).

[35] Nivedita Daimiwal, M. Sundhararajan, and Revati Shriram. "Respiratory rate, heart rate and continuous measurement of BP using PPG". In: *International Conference on Communication and Signal Processing, ICCSP 2014 - Proceedings* (Nov. 2014), pp. 999–1002. DOI: 10.1109/ICCSP.2014.6949996 (cit. on p. 19).

[36] K. W. Chan, K. Hung, and Y. T. Zhang. "Noninvasive and cuffless measurements of blood pressure for telemedicine". In: *Annual Reports of the Research Reactor Institute, Kyoto University* 4 (2001), pp. 3592–3593. ISSN: 04549244. DOI: 10.1109/iembs.2001.1019611 (cit. on p. 19).

[37] Zeng Ding Liu et al. "The Wavelet Transform of Pulse Wave and Electrocardiogram Improves Accuracy of Blood Pressure Estimation in Cuffless Blood Pressure Measurement". In: 11 (Mar. 2018) (cit. on p. 19).

[38] Xiao Rong Ding et al. "Continuous Cuffless Blood Pressure Estimation Using Pulse Transit Time and Photoplethysmogram Intensity Ratio". In: *IEEE Transactions on Biomedical Engineering* 63 (5 May 2016), pp. 964–972. ISSN: 15582531. DOI: 10.1109/TBME.2015.2480679 (cit. on p. 19).

[39] Shi Chao Gao et al. "Data-driven estimation of blood pressure using photoplethysmographic signals". In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* 2016-October (Oct. 2016), pp. 766–769. ISSN: 1557170X. DOI: 10.1109/EMBC.2016.7590814 (cit. on p. 20).

[40] Mohamad Kachuee et al. “Cuff-less high-accuracy calibration-free blood pressure estimation using pulse transit time”. In: *Proceedings - IEEE International Symposium on Circuits and Systems* 2015-July (July 2015), pp. 1006–1009. ISSN: 02714310. DOI: 10.1109/ISCAS.2015.7168806 (cit. on p. 20).

[41] Shuo Chen et al. “A non-invasive continuous blood pressure estimation approach based on machine learning”. In: *Sensors (Switzerland)* 19 (11 June 2019). ISSN: 14248220. DOI: 10.3390/s19112585 (cit. on p. 20).

[42] Vicent Ripoll and Alfredo Vellido. “Blood Pressure Assessment with Differential Pulse Transit Time and Deep Learning: A Proof of Concept”. In: *Kidney Diseases* 5 (1 2019), pp. 23–27. ISSN: 2296-9381. DOI: 10.1159/000493478 (cit. on p. 20).

[43] George B. Moody and Roger G. Mark. “A database to support development and evaluation of intelligent intensive care monitoring”. In: *Computers in Cardiology* 0 (0 1996), pp. 657–660. ISSN: 02766574. DOI: 10.1109/cic.1996.542622 (cit. on p. 23).

[44] Python. *History and License for Python*. June 2022 (cit. on p. 23).

[45] Python. *Numpy documentation*. June 2022 (cit. on p. 23).

[46] Python. *Pandas documentation*. June 2022 (cit. on p. 23).

[47] Python. *Tensorflow documentation*. June 2022 (cit. on p. 23).

[48] Python. *Keras documentation*. June 2022 (cit. on p. 23).

[49] Python. *Scikit-Learn documentation*. June 2022 (cit. on p. 23).

[50] Python. *Heartpy documentation*. June 2022 (cit. on p. 23).

[51] Python. *WFDB-Python documentation*. June 2022 (cit. on p. 23).

[52] Python. *Matplotlib documentation*. June 2022 (cit. on p. 23).

Appendix

FYP Mission Statement (correct to June 2022)

'*Ambulatory blood pressure monitoring has become increasingly relevant due to the advancement of wearable technology. Modalities such as Electrocardiogram (ECG) and Photoplethysmography (PPG) have provided an indirect method of blood pressure estimations compared to a traditional blood-pressure monitor. Although algorithms exist for calculating blood pressure with ECG and PPG, it is vital that their computational complexity is minimal, whilst maintaining accuracy, due to the power limitations of wearable technology. The goal of this project is to establish the tradeoffs between using PPG and ECG to quantify blood pressure, in the context of wearable technology, where power must be kept to a minimum. This project is ideal for students interested in signal processing, who have excellent programming skills in Matlab.*'

Health standards requirements for blood pressure estimation

- The Advancement of Medical Instrumentation (AAMI) standard requires a mean BP difference of ≤ 5 mmHg with a standard deviation of ≤ 8 mmHg against auscultatory reference measurement
- Significant variation in BP measurements(> 12 mmHg systolic or > 8 mmHg diastolic) from the validated reference device is an exclusion criterion in the AAMI protocol [7]
- The European Society of Hypertension (ESH) protocol requires that the majority of subjects have investigational BP readings within ≤ 5 mmHg of the reference measurement.

Complete FYP Gantt chart

Overview on Cuff-Less BP device options

Wearable technology provides an opportunity for real-time monitoring of human vital signs, thus enabling the possibility for preventive, timely notification and real-time diagnosis. Unlike commonly-used BP sensors, which demand a specific measurement procedure, modern wearable bio-sensors monitor vital signals online and all day long, presenting no additional burden other than wearing the device. A wide range of wearable devices achieve very positive results, even those wearables which are cheaper in price [3]. Some of the systems developed for the purpose of non-invasive BP monitoring will be discussed in more detail now. In addition, they will be critically analysed against two other viable alternatives.

The motivation behind this project is to replace the current cuff-based BP devices. Cuff-based devices often require the supervision of an expert to work correctly and do not provide continuous measurements for BP. In addition they can cause irritation and inconvenience for patients due to cuff inflation and deflation. As a result, current clinical cuff-based BP devices are not suitable for providing continuous BP monitoring which could play a significant role in the early detection of diseases which affect the heart [16].

Smart Watches

Smart watches are becoming an increasingly popular form of wearable technology [7]. Most of the existing smart watch produces currently measure BP through Pulse Transit Time (PTT) from a pulse wave measurement at the wrist. The Heartisans BP smartwatch uses ECG and PPG signals to measure PTT and estimate BP. The watch requires a motionless 20 second scan with the device held at heart level for measurements and provides systolic and diastolic BP readings. Calibration with a validated cuff-type BP device is required prior to standalone use. Despite its availability on the market, the Heartisans Watch has not undergone a formal validation study [7].

Another wearable method is through arterial tonometry. The BPro device, developed by the London company HealthSTATS Technologies, is one such device.

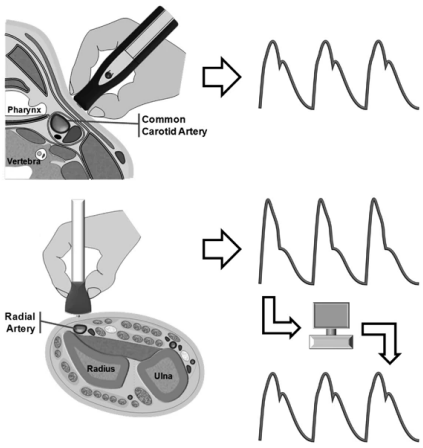


Figure 21: Arterial Tonometry

Figure 21 shows two different methods for using arterial tonometry to measure blood pressure. The first image shows that the recording is taking at the carotid artery level in order to estimate the central blood pressure waveform. In the second image, the recording is taken at the radial artery level to estimate the arterial pulse wave. The central waveform is then reconstructed from this pulse wave using software.

The BPro allows for continuous 24 hour waveform analysis of BP. Calibration to the brachial artery BP is required using a validated upper arm cuff device prior to use. The BPro has been validated in a non-ambulatory study following the Association for the Advancement of Medical Instrumentation (AAMI) standards and the European Society of Hypertension (ESH) protocol.

Smartphone Applications

Smartphones offer a great potential to expand the continuous recording of BP if their sensors can be correctly utilised [7]. A formal validation study on one particular iOS application, the Instant Blood Pressure app, revealed poor accuracy of BP measurements. Mean absolute differences of 12.4 mmHg for systolic BP and 10.1 mmHg for diastolic BP were found between the Instant Blood Pressure application and a reference device. This resulted in approximately four out of five hypertensive individuals being falsely classified as normotensive [7]. The My BP

Lab application measures BP through measurements of Pulse Transit Time (PTT). However, there is currently a lack of experimental data to justify it's dominance in the market.

Medical Tricorders

A medical tricorder is a handheld portable device used by consumers to self-diagnose medical conditions and take basic vital signs measurements. The BodiMetrics Performance Monitor uses ECG and PPG signal data to estimate BP through PTT. The Bodimetrics tricorder obtains measurements with a 20 second scan of the user's fingertip at heart level but only provides systolic BP data. This tricorder has a large spread in absolute bias against an automated sphygmomanometer, hence it is unlikely to meet formal accuracy and precision standards [7].

The FreeScan Personal Cardiovascular Monitor, which is developed by the Taiwanese company Maisense Inc., also estimates BP from PTT. However the device uses a force sensor to capture the systolic arterial waveform rather than PPG. This requires the user to physically apply the force sensor directly to the radial artery for around 10 seconds for measurements. The Freescan device has been verified according to the AAMI protocol [7].

The SOMNOtouch NIBP is a non-traditional medical tricorder that utilizes PTT data collected in a similar fashion to the Bodimetrics Performance Monitor. The device has met the ESH standards, however it begins to lose accuracy for higher SBP and DBP values [7].

Conclusion

After having discussed three viable devices for cuff-less measurements of BP, it has been finalised that smart watches are the most viable option. Whilst existing smart watch devices do suffer from inaccuracies in BP estimation due to motion, it is clear that they produce the most acceptable results in line with the AAMI and ESH standards. This chapter can be seen as a forward looking overview of how the estimation methods discussed in this report can be used to benefit future products. With regards to this project, this chapter can be treated as a supplementary overview.

Python code for the neural network architectures

AlexNet

```
1    from tensorflow.keras.layers import Softmax, Permute, Input, Add,
                                         Conv1D, MaxPooling1D, Dense,
                                         Activation, ZeroPadding2D,
                                         BatchNormalization, Flatten,
                                         Conv2D, AveragePooling1D,
                                         MaxPooling2D,
                                         GlobalMaxPooling2D, LeakyReLU,
                                         GlobalAveragePooling2D, ReLU,
                                         Dropout
2    from tensorflow.keras.initializers import glorot_uniform
3    from tensorflow.keras.models import Model
4    import tensorflow as tf
```

```
5       import scipy
6
7       def AlexNet_1D(data_in_shape, num_output=2, dil=1, kernel_size=3,
                                        fs = 125, useMaxPooling=True,
                                        UseDerivative=False):
8
9           # Define the input as a tensor with shape input_shape
10          X_input = Input(shape=data_in_shape)
11
12          if UseDerivative:
13              dt1 = (X_input[:,1:] - X_input[:,:-1])*fs
14              dt2 = (dt1[:,1:] - dt1[:,:-1])*fs
15              dt1 = tf.pad(dt1, tf.constant([[0,0],[0,1],[0,0]]))
16              dt2 = tf.pad(dt2, tf.constant([[0,0],[0,2],[0,0]]))
17              X = tf.concat([X_input, dt1, dt2], axis=2)
18          else:
19              X=X_input
20
21
22          # convolutional stage
23          X = Conv1D(96, 7, strides=3, name='conv1', kernel_initializer=
                                        glorot_uniform(seed=0),
                                        padding="same")(X)
24          if useMaxPooling:
25              X = MaxPooling1D(3, strides=2, name="MaxPool1")(X)
26          X = Activation(ReLU())(X)
27          X = BatchNormalization(axis=-1, name='BatchNorm1')(X)
28
29          X = Conv1D(256, kernel_size=kernel_size, strides=1,
                                        dilation_rate=dil, name='
                                        conv2', kernel_initializer=
                                        glorot_uniform(seed=0),
                                        padding="same")(X)
30          if useMaxPooling:
31              X = MaxPooling1D(3, strides=2, name="MaxPool2")(X)
32          X = Activation(ReLU())(X)
33          X = BatchNormalization(axis=-1, name='BatchNorm2')(X)
34
35          X = Conv1D(384, kernel_size=kernel_size, strides=1,
                                        dilation_rate=dil, name='
                                        conv3', kernel_initializer=
                                        glorot_uniform(seed=0),
                                        padding="same")(X)
36          X = Activation(ReLU())(X)
37          X = BatchNormalization(axis=-1, name='BatchNorm3')(X)
38
39          X = Conv1D(384, kernel_size=kernel_size, strides=1,
                                        dilation_rate=dil, name='
                                        conv4', kernel_initializer=
```

```
                                        glorot_uniform(seed=0),
                                        padding="same")(X)
        X = Activation(ReLU())(X)
        X = BatchNormalization(axis=-1, name='BatchNorm4')(X)

        X = Conv1D(256, kernel_size=kernel_size, strides=1,
                                        dilation_rate=dil, name='
                                        conv5', kernel_initializer=
                                        glorot_uniform(seed=0),
                                        padding="same")(X)
        if useMaxPooling:
            X = MaxPooling1D(3, strides=2, name="MaxPool5")(X)
        X = Activation(ReLU())(X)
        X = BatchNormalization(axis=-1, name='BatchNorm5')(X)

        # Fully connected stage
        X = Flatten()(X)
        X = Dense(4096, activation='relu', name='dense1',
                                        kernel_initializer=
                                        glorot_uniform(seed=0))(X)
        X = Dropout(rate=0.5)(X)
        X = Dense(4096, activation='relu', name='dense2',
                                        kernel_initializer=
                                        glorot_uniform(seed=0))(X)
        X = Dropout(rate=0.5)(X)

        # Create model
        if num_output == 1:
            X_out = Dense(3, activation='softmax', name='out',
                                            kernel_initializer=
                                            glorot_uniform(seed=0))
                                            (X)
            model = Model(inputs=X_input, outputs=X_out, name='
                                            AlexNet_1D')
        else:
            # output stage
            X_SBP = Dense(1, activation='relu', name='SBP',
                                            kernel_initializer=
                                            glorot_uniform(seed=0))
                                            (X)
            X_DBP = Dense(1, activation='relu', name='DBP',
                                            kernel_initializer=
                                            glorot_uniform(seed=0))
                                            (X)
            model = Model(inputs=X_input, outputs=[X_SBP, X_DBP], name
                                            ='AlexNet_1D')

        return model
```

ResNet

```
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Add, Dense, Activation,
                                    ZeroPadding1D, BatchNormalization,
                                    Flatten, Conv1D, AveragePooling1D,
                                    MaxPooling1D, GlobalMaxPooling2D,
                                    LeakyReLU, GlobalAveragePooling2D,
                                    ReLU, concatenate
from tensorflow.keras.initializers import glorot_uniform, constant
from tensorflow.keras.models import Model
import tensorflow as tf


def identity_block(X, f, filters, stage, block, dil=1):

    # Implementation of the identity block as defined in Figure 3

    # Arguments:
    # X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
    # f -- integer, specifying the shape of the middle CONV's window
                                        for the main path
    # filters -- python list of integers, defining the number of
                                        filters in the CONV layers of
                                        the main path
    # stage -- integer, used to name the layers, depending on their
                                        position in the network
    # block -- string/character, used to name the layers, depending on
                                         their position in the network

    #  Returns:
    # X -- output of the identity block, tensor of shape (n_H, n_W,
                                        n_C)

    # defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value. You'll need this later to add back to the
                                        main path.
    X_shortcut = X

    # First component of main path
    X = Conv1D(filters = F1, kernel_size = 1, strides = 1,
                                        dilation_rate=dil, padding = '
                                        valid', name = conv_name_base +
```

```
                                          '2a', kernel_initializer =
                                          glorot_uniform(seed=0))(X)
34      X = BatchNormalization(momentum = 0.9, name = bn_name_base + '2a')
                                          (X)
35      X = Activation(ReLU())(X)
36
37      # Second component of main path
38      X = Conv1D(filters = F2, kernel_size = f, strides = 1,
                                          dilation_rate=dil, padding = '
                                          same', name = conv_name_base +
                                          '2b', kernel_initializer =
                                          glorot_uniform(seed=0))(X)
39      X = BatchNormalization(momentum=0.9, name=bn_name_base + '2b')(X)
40      X = Activation(ReLU())(X)
41
42      # Third component of main path
43      X = Conv1D(filters = F3, kernel_size = 1, strides = 1,
                                          dilation_rate=dil, padding = '
                                          valid', name = conv_name_base +
                                          '2c', kernel_initializer =
                                          glorot_uniform(seed=0))(X)
44      X = BatchNormalization(momentum = 0.9, name = bn_name_base + '2c')
                                          (X)
45
46      # Final step: Add shortcut value to main path, and pass it through
                                          a RELU activation
47      X = Add()([X, X_shortcut])
48      X = Activation(ReLU())(X)
49      # X = BatchNormalization(momentum = 0.9, name = bn_name_base + '2c
                                          ')(X)
50
51      return X
52
53  def convolutional_block(X, f, filters, stage, block, s = 2, dil = 1):
54
55      # Implementation of the convolutional block as defined in Figure 4
56
57      # Arguments:
58      # X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
59      # f -- integer, specifying the shape of the middle CONV's window
                                          for the main path
60      # filters -- python list of integers, defining the number of
                                          filters in the CONV layers of
                                          the main path
61      # stage -- integer, used to name the layers, depending on their
                                          position in the network
62      # block -- string/character, used to name the layers, depending on
                                          their position in the network
63      # s -- Integer, specifying the stride to be used
```

```
64
65      # Returns:
66      # X -- output of the convolutional block, tensor of shape (n_H,
                                                  n_W, n_C)
67
68      # defining name basis
69      conv_name_base = 'res' + str(stage) + block + '_branch'
70      bn_name_base = 'bn' + str(stage) + block + '_branch'
71
72      # Retrieve Filters
73      F1, F2, F3 = filters
74
75      # Save the input value
76      X_shortcut = X
77
78
79      ##### MAIN PATH #####
80      # First component of main path
81      X = Conv1D(F1, 1, strides = s, name = conv_name_base + '2a',
                                          kernel_initializer =
                                          glorot_uniform(seed=0))(X)
82      X = BatchNormalization(momentum=0.9, name=bn_name_base + '2a')(X)
83      X = Activation(ReLU())(X)
84
85      # Second component of main path
86      X = Conv1D(filters = F2, kernel_size = f, strides = 1,
                                          dilation_rate=dil, padding = '
                                          same', name = conv_name_base +
                                          '2b', kernel_initializer =
                                          glorot_uniform(seed=0))(X)
87      X = BatchNormalization(momentum=0.9, name=bn_name_base + '2b')(X)
88      X = Activation(ReLU())(X)
89
90      # Third component of main path
91      X = Conv1D(filters = F3, kernel_size = 1, dilation_rate=dil,
                                          strides = 1, padding = 'valid',
                                           name = conv_name_base + '2c',
                                          kernel_initializer =
                                          glorot_uniform(seed=0))(X)
92      X = BatchNormalization(momentum = 0.9, name = bn_name_base + '2c')
                                          (X)
93
94      ##### SHORTCUT PATH ####
95      X_shortcut = Conv1D(filters = F3, kernel_size = 1, strides = s,
                                          padding = 'valid', name =
                                          conv_name_base + '1',
96                          kernel_initializer = glorot_uniform(seed=0))(
                                                              X_shortcut)
```

```
97      X_shortcut = BatchNormalization( momentum = 0.9, name =
                                        bn_name_base + '1')(X_shortcut)
98
99      # Final step: Add shortcut value to main path, and pass it through
                                         a RELU activation
100     X = Add()([X, X_shortcut])
101     X = Activation(ReLU())(X)
102     # X = BatchNormalization(momentum = 0.9, name = bn_name_base + '2c
                                        ')(X)
103
104     return X
105
106 def ResNet50_1D(data_in_shape, num_output=2, fs=125, UseDerivative=
                                     False):
107
108     # Implementation of the popular ResNet50 the following
                                         architecture:
109     # CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVBLOCK -> IDBLOCK*2
                                          -> CONVBLOCK -> IDBLOCK*3
110     # -> CONVBLOCK -> IDBLOCK*5 -> CONVBLOCK -> IDBLOCK*2 -> AVGPOOL
                                         -> TOPLAYER
111
112     # Arguments:
113     # input_shape -- shape of the images of the dataset
114     # classes -- integer, number of classes
115     # Returns:
116     # model -- a Model() instance in Keras
117
118     # Define the input as a tensor with shape input_shape
119     X_input = Input(shape=data_in_shape)
120
121     if UseDerivative:
122         dt1 = (X_input[:,1:] - X_input[:,:-1])*fs
123         dt2 = (dt1[:,1:] - dt1[:,:-1])*fs
124
125         dt1 = tf.pad(dt1, tf.constant([[0,0],[0,1],[0,0]]))
126         dt2 = tf.pad(dt2, tf.constant([[0,0],[0,2],[0,0]]))
127
128         X = tf.concat([X_input, dt1, dt2], axis=2)
129     else:
130         X=X_input
131
132     # Zero-Padding
133     X = ZeroPadding1D(3)(X_input)
134
135     # Stage 1
136     X = Conv1D(64, 7, strides=2, name='conv1', kernel_initializer=
                                        glorot_uniform(seed=0))(X)
137     X = BatchNormalization(axis=2, name='bn_conv1')(X)
```

```
    X = Activation('relu')(X)
    X = MaxPooling1D(3, strides=3)(X)

    # Stage 2
    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2,
                                        block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')

    # Stage 3
    X = convolutional_block(X, f = 3, filters = [128, 128, 512], stage
                                         = 3, block='a', s = 2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    # Stage 4
    X = convolutional_block(X, f = 3, filters = [256, 256, 1024],
                                        stage = 4, block='a', s = 2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    # Stage 5
    X = convolutional_block(X, f = 3, filters = [512, 512, 2048],
                                        stage = 5, block='a', s = 2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    X = AveragePooling1D(2, name="avg_pool")(X)

    # output layer
    X = Flatten()(X)
    X_SBP = Dense(1, activation='linear', name='SBP',
                                        kernel_initializer =
                                        glorot_uniform(seed=0))(X)
    X_DBP = Dense(1, activation='linear', name='DBP',
                                        kernel_initializer =
                                        glorot_uniform(seed=0))(X)
    HR = Dense(1, activation='linear', name='HR', kernel_initializer =
                                         glorot_uniform(seed=0))(X)

    # Create model
    if num_output==3:
        model = Model(inputs=X_input, outputs=[X_SBP, X_DBP, HR], name
                                            ='ResNet50_1D')
    else:
```

```
        model = Model(inputs = X_input, outputs = [X_SBP, X_DBP], name
                                                ='ResNet50_1D')

    return model
```

ResNet with Leave One Subject Out (LOSO)

```
from __future__ import division, print_function
import numpy as np
np.random.seed(3)
import os

from scipy.signal import butter, lfilter, lfilter_zi, filtfilt,
                                    savgol_filter
from sklearn.preprocessing import MinMaxScaler
import sklearn as sk
import random
random.seed(3)

import scipy.io as sio
import matplotlib.pyplot as plt
import natsort as natsort
from scipy import signal
import math

import tensorflow as tf
# from tensorflow.keras.utils import multi_gpu_model

from tensorflow.keras.models import Sequential
from tensorflow.keras.backend import squeeze
from kapre import STFT, Magnitude, MagnitudeToDecibel
# from kapre.utils import Normalization2D
from tensorflow.keras.layers import Input, BatchNormalization, Lambda,
                                    AveragePooling2D, Flatten, Dense,
                                    Conv1D, Activation, add,
                                    AveragePooling1D, Dropout, Permute,
                                    concatenate, MaxPooling1D, LSTM,
                                    Reshape, GRU
from tensorflow.keras.regularizers import l2
from tensorflow.keras import Model
from tensorflow.keras import optimizers
#from tensorflow.keras.utils.vis_utils import plot_model

from tensorflow.keras.layers import Conv2D, MaxPooling2D

def diff(input, fs):
    dt = (input[:, 1:] - input[:, :-1]) * fs
    dt = tf.pad(dt, tf.constant([[0, 0], [0, 1], [0, 0]]))
```

```
36
37      return dt
38
39  def mid_spectrogram_layer(input_x):
40      l2_lambda = .001
41      n_dft = 128
42      n_hop = 64
43      fmin = 0.0
44      fmax = 50 / 2
45
46      x = Permute((2, 1))(input_x)
47      # x = input_x
48      x = STFT(n_fft=n_dft, hop_length=n_hop, output_data_format='
                                              channels_last')(x)
49      x = Magnitude()(x)
50      #x = MagnitudeToDecibel()(x)
51      #x = BatchNormalization()(x)
52      # x = Normalization2D(str_axis='batch')(x)
53      x = Flatten()(x)
54      x = Dense(32, activation="relu", kernel_regularizer=l2(l2_lambda))
                                              (x)
55      x = BatchNormalization()(x)
56
57      return x
58
59
60  def mid_spectrogram_LSTM_layer(input_x):
61      l2_lambda = .001
62      n_dft = 64
63
64      n_hop = 64
65      fmin = 0.0
66      fmax = 50 / 2
67
68      #x = Permute((2, 1))(input_x)
69      x = input_x
70      x = STFT(n_fft=n_dft, hop_length=n_hop, output_data_format='
                                              channels_last')(x)
71      x = Magnitude()(x)
72      x = MagnitudeToDecibel()(x)
73     #x = BatchNormalization()(x)
74      # x = Normalization2D(str_axis='batch')(x)
75      # print(np.array(x).shape)
76      # x = Reshape((2, 64))(x)
77      # x = GRU(64)(x)
78      x = Flatten()(x)
79      x = Dense(32, activation="relu", kernel_regularizer=l2(l2_lambda))
                                              (x)
80      x = BatchNormalization()(x)
```

```
81
82      return x
83
84
85  def single_channel_resnet(my_input, num_filters=64, num_res_blocks=4,
                                  cnn_per_res=3,
86                            kernel_sizes=[8, 5, 3], max_filters=128,
                                                          pool_size
                                                          =3,
                                                          pool_stride_size
                                                          =2):
87
88      #my_input = Input(shape=input_shape)
89      # my_input = input_shape
90      # my_input = ks.expand_dims(my_input, axis=2)
91
92      for i in np.arange(num_res_blocks):
93          if (i == 0):
94              block_input = my_input
95              x = BatchNormalization()(block_input)
96          else:
97              block_input = x
98
99          for j in np.arange(cnn_per_res):
100             x = Conv1D(num_filters, kernel_sizes[j], padding='same')(x
                                                    )
101             x = BatchNormalization()(x)
102             if (j < cnn_per_res - 1):
103                 x = Activation('relu')(x)
104
105         is_expand_channels = not (my_input.shape[0] == num_filters)
106
107         if is_expand_channels:
108             res_conn = Conv1D(num_filters, 1, padding='same')(
                                                    block_input)
109             res_conn = BatchNormalization()(res_conn)
110         else:
111             res_conn = BatchNormalization()(block_input)
112
113         x = add([res_conn, x])
114         x = Activation('relu')(x)
115
116         if (i < 5):
117             x = AveragePooling1D(pool_size=pool_size, strides=
                                                    pool_stride_size)(x)
118
119         num_filters = 2 * num_filters
120         if max_filters < num_filters:
121             num_filters = max_filters
```

```
122
123     return my_input, x
124
125
126 def raw_signals_deep_ResNet(input, UseDerivative=False):
127     fs=125
128
129     inputs = []
130     l2_lambda = .001
131     channel_outputs = []
132     num_filters = 32
133
134     X_input = Input(shape=input)
135
136     if UseDerivative:
137         # fs = tf.constant(fs, dtype=float)
138         X_dt1 = Lambda(diff, arguments={'fs': fs})(X_input)
139         X_dt2 = Lambda(diff, arguments={'fs': fs})(X_dt1)
140         X = [X_input, X_dt1, X_dt2]
141     else:
142         X = [X_input]
143
144     num_channels = len(X)
145
146     for i in np.arange(num_channels):
147         channel_resnet_input, channel_resnet_out =
                                            single_channel_resnet(X[i],
                                             num_filters=num_filters,
148         num_res_blocks=4, cnn_per_res=3, kernel_sizes=[8, 5, 5, 3],
                                            max_filters=64, pool_size=2
                                            , pool_stride_size=1)
149         channel_outputs.append(channel_resnet_out)
150         inputs.append(channel_resnet_input)
151
152     spectral_outputs = []
153     num_filters = 32
154     for x in inputs:
155         spectro_x = mid_spectrogram_LSTM_layer(x)
156         spectral_outputs.append(spectro_x)
157
158     # concatenate the channel specific residual layers
159
160     # print("Num Channels: ", num_channels)
161
162     if num_channels > 1:
163         x = concatenate(channel_outputs, axis=-1)
164     else:
165         x = channel_outputs[0]
166
```

```
167     x = BatchNormalization()(x)
168     x = GRU(65)(x)
169     # x = Flatten()(x)
170     x = BatchNormalization()(x)
171
172     # join time-domain and frequnecy domain fully-conencted layers
173     if num_channels > 1:
174         s = concatenate(spectral_outputs, axis=-1)
175     else:
176         s = spectral_outputs[0]
177
178     # s = Flatten()(s)
179     #     x = Dense(128,activation="relu",kernel_regularizer=l2(
                                            l2_lambda)) (x)
180     s = BatchNormalization()(s)
181     # LETS DO OVERFIT
182     x = concatenate([s, x])
183     x = Dense(32, activation="relu", kernel_regularizer=l2(l2_lambda))
                                            (x)
184     x = Dropout(0.25)(x)
185     x = Dense(32, activation="relu", kernel_regularizer=l2(l2_lambda))
                                            (x)
186     x = Dropout(0.25)(x)
187     #output = Dense(2, activation="relu")(x)
188     x = Flatten()(x)
189     X_SBP = Dense(1, activation='linear', name='SBP')(x)
190     X_DBP = Dense(1, activation='linear', name='DBP')(x)
191
192     model = Model(inputs=X_input, outputs=[X_SBP, X_DBP], name="
                                            Slapnicar_Model")
193     # model = multi_gpu_model(model, gpus=2)
194     # optimizer = optimizers.Adadelta()
195     # loss = ks.keras.losses.mean_absolute_error
196     # model.compile(optimizer=optimizer, loss=loss, metrics=['mae', '
                                            mae'])
197     print(model.summary())
198     # plot_model(model=model, to_file='lstm_model.png', show_shapes=
                                            True, show_layer_names=True)
199     return model
200
201
202 def one_chennel_resnet(input_shape,num_filters=16,num_res_blocks = 5,
                                    cnn_per_res = 3,
203                         kernel_sizes = [3,3,3], max_filters = 64,
                                                        pool_size =
                                                         3,
204                         pool_stride_size = 2,num_classes=8):
205     my_input  = Input(shape=(input_shape))
206     for i in np.arange(num_res_blocks):
```

```
207         if(i==0):
208             block_input = my_input
209             x = BatchNormalization()(block_input)
210         else:
211             block_input = x
212         for j in np.arange(cnn_per_res):
213             x = Conv1D(num_filters, kernel_sizes[j], padding='same')(x
                                                    )
214             x = BatchNormalization()(x)
215             if(j<cnn_per_res-1):
216                 x = Activation('relu')(x)
217         is_expand_channels = not (input_shape[0] == num_filters)
218         if is_expand_channels:
219             res_conn = Conv1D(num_filters, 1,padding='same')(
                                                    block_input)
220             res_conn = BatchNormalization()(res_conn)
221         else:
222             res_conn = BatchNormalization()(block_input)
223         x = add([res_conn, x])
224         x = Activation('relu')(x)
225         if(i<5):
226             x = MaxPooling1D(pool_size=pool_size,strides =
                                                    pool_stride_size)(x)
227         num_filters = 2*num_filters
228         if max_filters<num_filters:
229             num_filters = max_filters
230     return my_input,x
231
232
233 def one_chennel_resnet_2D(input_shape, input_layer, num_filters=16,
                                    num_res_blocks = 5,cnn_per_res = 3,
234                         kernel_sizes = [8, 5, 3], max_filters = 64,
                                                        pool_size =
                                                         (3,3),
235                         pool_stride_size = 2, num_classes=8):
236     kernel_sizes = [(8, 1), (5, 1), (3, 1)]
237     my_input = input_layer
238     for i in np.arange(num_res_blocks):
239         if(i==0):
240             block_input = my_input
241             x = BatchNormalization()(block_input)
242         else:
243             block_input = x
244         for j in np.arange(cnn_per_res):
245             x = Conv2D(num_filters, kernel_sizes[j], padding='same')(x
                                                    )
246             x = BatchNormalization()(x)
247             if(j<cnn_per_res-1):
248                 x = Activation('relu')(x)
```

```
249         is_expand_channels = not (input_shape[0] == num_filters)
250         if is_expand_channels:
251             res_conn = Conv2D(num_filters, (1,1), padding='same')(
                                                    block_input)
252             res_conn = BatchNormalization()(res_conn)
253         else:
254             res_conn = BatchNormalization()(block_input)
255         x = add([res_conn, x])
256         x = Activation('relu')(x)
257         if(i<5):
258             x = MaxPooling2D(pool_size=pool_size,strides =
                                                    pool_stride_size)(x)
259         num_filters = 2*num_filters
260         if max_filters<num_filters:
261             num_filters = max_filters
262     return my_input,x
263
264
265 def spectro_layer_mid(input_x,sampling_rate, ndft=0, num_classes=8):
266     l2_lambda = .001
267     if(ndft == 0):
268         n_dft= 128
269     else:
270         n_dft = ndft
271     # n_dft = 64
272     n_hop = 64
273     fmin=0.0
274     fmax=sampling_rate//2
275
276     x = Permute((2,1))(input_x)
277     x = STFT(n_fft=n_dft, hop_length=n_hop, output_data_format='
                                        channels_last')(x)
278     x = Magnitude()(x)
279     #x = MagnitudeToDecibel()(x)
280     #x = BatchNormalization()(x)     # x = Normalization2D(str_axis='
                                        batch')(x)
281     channel_resnet_input,channel_resnet_out= one_chennel_resnet_2D((
                                        625, 1), x, num_filters=64,
282                     num_res_blocks = 6,cnn_per_res = 3,kernel_sizes =
                                                    [3,3,3,3],
283                     max_filters = 32, pool_size = 1,
284                     pool_stride_size =1,num_classes=8)
285     channel_resnet_out = BatchNormalization()(channel_resnet_out)
286
287     # x = Reshape((10, 65))(x)
288     # x = GRU(65)(x)
289
290     return channel_resnet_out
```

LSTM

```
    from tensorflow.keras.layers import Input, LSTM, Dense,
                                        Bidirectional, Conv1D, ReLU

from tensorflow.keras import Model

def define_LSTM(data_in_shape, UseDerivative=False):
    X_input = Input(shape=data_in_shape)

    fs = 125

    if UseDerivative:
        dt1 = (X_input[:,1:] - X_input[:,:-1])*fs
        dt2 = (dt1[:,1:] - dt1[:,:-1])*fs

        dt1 = tf.pad(dt1, tf.constant([[0,0],[0,1],[0,0]]))
        dt2 = tf.pad(dt2, tf.constant([[0,0],[0,2],[0,0]]))

        X = tf.concat([X_input, dt1, dt2], axis=2)
    else:
        X=X_input


    X = Conv1D(filters=64, kernel_size=5, strides=1, padding='causal',
                                         activation='relu')(X)
    X = Bidirectional(LSTM(128, return_sequences=True))(X)
    X = Bidirectional(LSTM(128, return_sequences=True))(X)
    X = Bidirectional(LSTM(64, return_sequences=False))(X)
    X = Dense(512, activation='relu')(X)
    X = Dense(256, activation='relu')(X)
    X = Dense(128, activation='relu')(X)

    X_SBP = Dense(1, name='SBP')(X)
    X_DBP = Dense(1, name='DBP')(X)

    model = Model(inputs=X_input, outputs=[X_SBP, X_DBP], name='LSTM')

    return model
```

Transformer encoder

```
from tensorflow import keras
from tensorflow.keras import layers

def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=
                                        0):
    # Normalization and Attention
```

```
    x = layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    # Feed Forward Part
    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res


def define_encoder(
    input_shape,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout=0,
    mlp_dropout=0,
    UseDerivative=False
):
    inputs = keras.Input(shape=input_shape)
    X_input = inputs
    fs = 125

    if UseDerivative:
        dt1 = (X_input[:,1:] - X_input[:,:-1])*fs
        dt2 = (dt1[:,1:] - dt1[:,:-1])*fs

        dt1 = tf.pad(dt1, tf.constant([[0,0],[0,1],[0,0]]))
        dt2 = tf.pad(dt2, tf.constant([[0,0],[0,2],[0,0]]))

        x = tf.concat([X_input, dt1, dt2], axis=2)
    else:
        x=X_input

    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="relu")(x)
```

```
        x = layers.Dropout(mlp_dropout)(x)

    # outputs = layers.Dense(1)(x)

    X_SBP = layers.Dense(1, name='SBP')(x)
    X_DBP = layers.Dense(1, name='DBP')(x)

    model = Model(inputs=inputs, outputs=[X_SBP, X_DBP], name='Encoder')
    # return keras.Model(inputs, outputs)
    return model
```