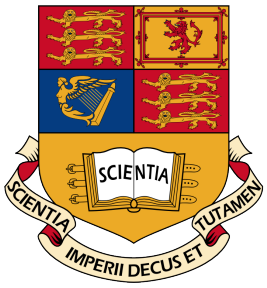
Imperial College London

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

The motivation of this project is the early prediction and prevention of hypertension, or high blood pressure, using 24-hour ambulatory monitoring methods. The currently recognised methods are split into two categories: invasive and cuffed non-invasive. Both methods are not feasible for ambulatory monitoring as they either require to be measured in Intensive Care Units (ICUs) or are too uncomfortable to be measured for long periods of time respectively. Hence, this provides the motivation to develop a non-invasive, cuffless method to estimate ambulatory blood pressure.

For this project, Systolic and Diastolic blood pressure values are estimated from photoplethysmography (PPG) signals using a Transformer Encoder Artificial Neural Network. The performance of this architecture is compared alongside four other architectures which have been previously used for cuffless estimation of blood pressure values.

This report includes a literature survey of all existing estimation methods. In addition, the analysis, design, testing and results of the proposed method are discussed. My proposed model achieves a mean absolute error of 12.740 mmHg for Systolic BP and 5.4763 mmHg for Diastolic BP respectively. Overall, the proposed method obtains an acceptable accuracy which can be improved in the future with further testing.

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, 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].

Ambulatory blood pressure monitoring is seen as a promising method for detecting early symptoms of hypertension [3]. There is a lot of existing research to predict ambulatory blood pressure using methods which are cuffless, continuous and non-invasive [4]. Recent developments in technology have made wearable sensors, such as Electrocardiogram (ECG) and Photoplethysmography (PPG) sensors significantly more feasible to be used for research into the long-term monitoring of blood pressure. These sensors cause minimal harm to patients compared to existing cuff-based methods [5] [6]. These sensors can also provide real-time 24 hour monitoring of the human body and have shown to be correlated with the behaviour of how the heart pumps blood around the body [6]. Hence there is great potential in using these sensors in wearable devices to diagnose medical conditions, such as hypertension, in real-time, thus helping to save lives [7].

1.2 High-level problem statement

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 objectives that will be assessed at the end of this report:

- Conduct a literature review to assess what are the best performing methods for estimating cuffless blood pressure 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 [8].

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, PPG derivative features)

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 [9]. 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 [10]. 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% [6] (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,

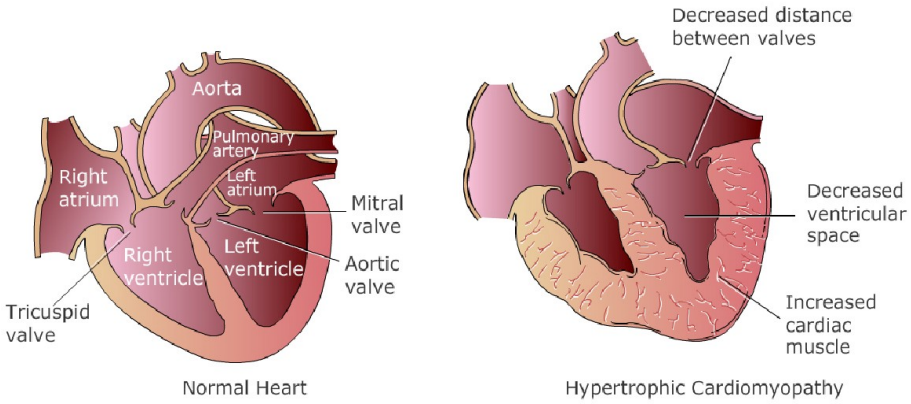


Figure 1: The effects of hypertension on the heart [11]

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

Table 1: Categories of blood pressure in adults [10] [12]

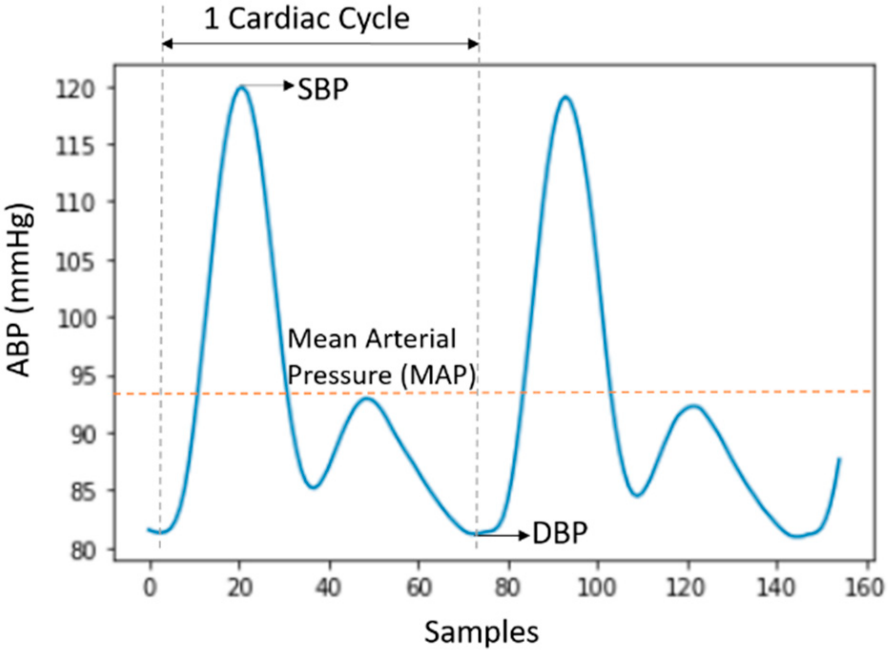
Blood pressure classification	**Blood Pressure (mmHg)**	
	Systolic	**Diastolic**
Hypotension	≤ 90	Or ≤ 60
Normal	90-119	And 60-79
Prehypertension	120-139	Or 80-89
Stage 1 hypertension	140-159	Or 90-99
Stage 2 hypertension	≥ 160	Or ≥ 100
Isolated Systolic hypertension	≥ 140	And < 90
Hypertensive crisis	≥ 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 [13]. It has been classed as the gold standard for detecting and diagnosing hypertension and also assessing BP values over a 24 hour period [3]. ABPM provides data on several important and unique parameters [3]. This data can explain how changes in your BP may correlate with your daily activities and sleep patterns [13]. Conventionally, ABP is monitored by using a cuff attached to a portable device which is worn on the patient's waist [3].

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) [7]. 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 [12] [14]. An example of this structure is provided in Figure 2.



(b)

Figure 2: Structure of an Arterial Blood Pressure signal [15]

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.

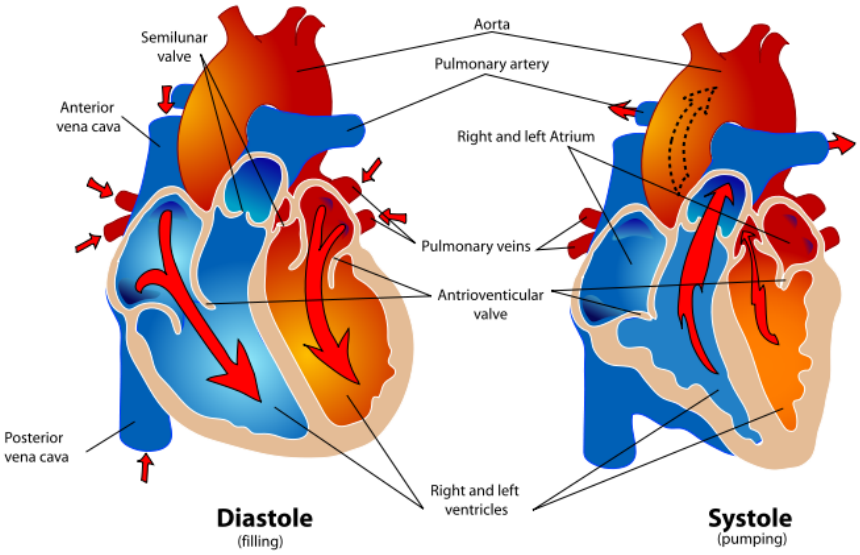


Figure 3: Visualisation for SBP and DBP [16]

Blood pressure values can also be assessed mathematically using the following formulae [17],

$$\text{Normal-to-normal (NN) interval} = \frac{\text{Largest difference between systolic peaks}}{f_s} \tag{1}$$

$$\text{Heart rate (HR)} = \frac{60}{\text{Mean (NN interval)}} \tag{2}$$

where f_s is the sampling frequency of the signal.

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 [4]. 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] [18]. However, this method is usually restricted to hospitals, as medical supervision is required [14]. 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 [4][18]. The main form of non-invasive BP measurements are through upper arm cuffed monitors. These forms of measurements are more beneficial than the invasive gold standard method, because they do not cause any major health risks [18]. However, if the cuff is worn on the arm for extended periods of time, this will cause a lot of discomfort to the patients [9]. In addition, it has been found that over three in ten home BP monitoring cuffs have produced inaccurate results [19].

As a result, the existing invasive and non-invasive BP measurement techniques are not feasible for an implementation involving continuous ambulatory BP monitoring [18]. 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 [12]. 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) [20]. The signals are recorded by measuring the electric potential difference by placing electrodes across the heart of an individual [9] [12]. 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 [9].

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 [20].

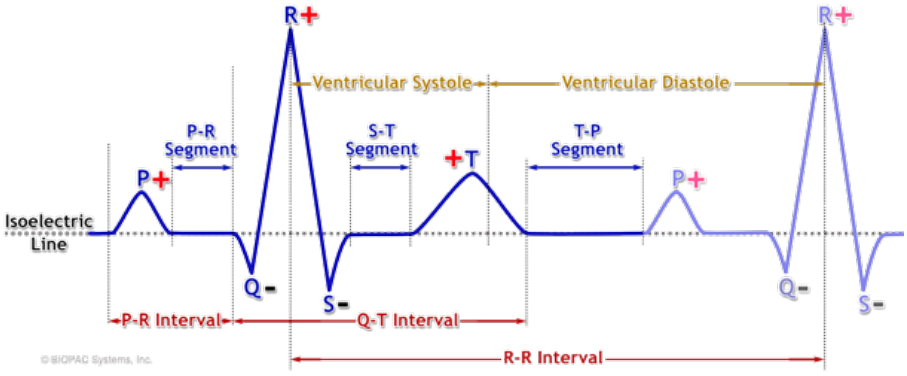


Figure 4: Structure of an ECG signal [21] [22]

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 [18]. Physically, the PPG signal is acquired by measuring the optical signal transmitted through or reflected from the subject's tissue [9]. 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 [18] [20].

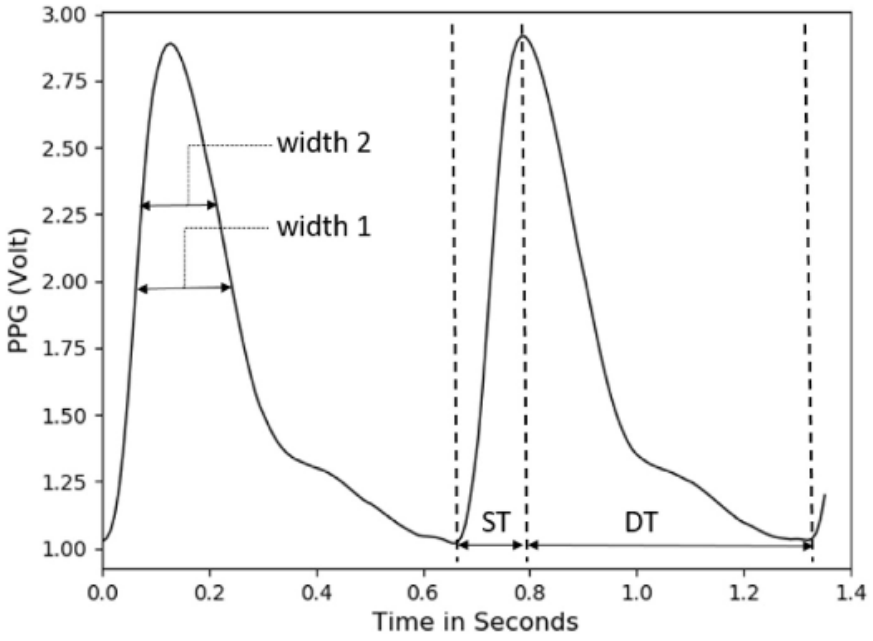


Figure 5: Structure of a PPG Signal [18]

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) [18]. PPG waveforms have a wide range of temporal features [18]. These features have been utilised in several experimentations, creating models to estimate blood pressure [14].

2.2 Cuff-less methods for deriving blood pressure (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 [23]. 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 [24]. 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 [9] [10] [18], as indicated in Figure 6.

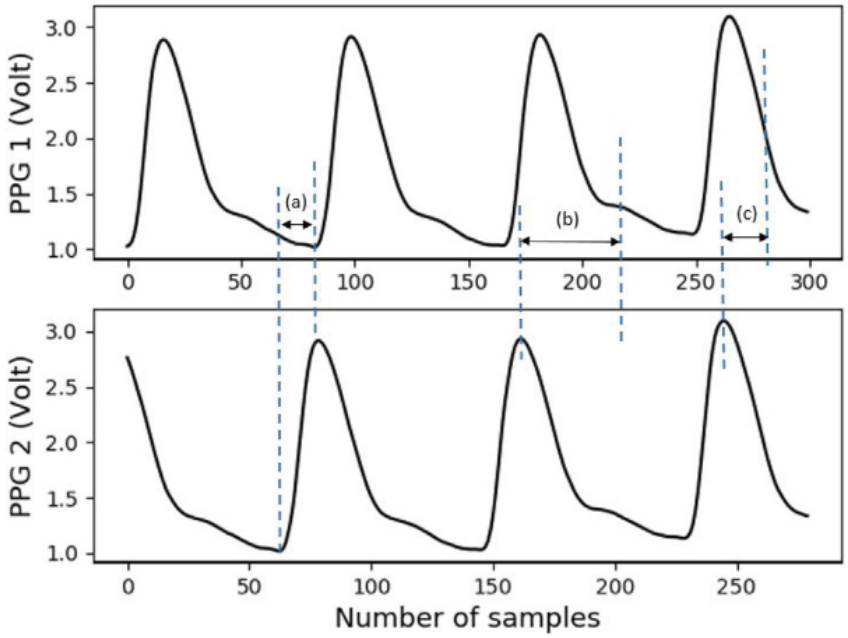


Figure 6: Pulse Transit Time (PTT) visualisation [18]

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 [18]. As a proof of concept, increasing blood pressure leads to an increase in the tension along the arterial wall tension, which therefore reduces the PTT. Hence, the opposite also applies [20].

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 [18]

[5]. 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 [25]. PAT is measured using two sensors, an ECG and a PPG sensor [18].

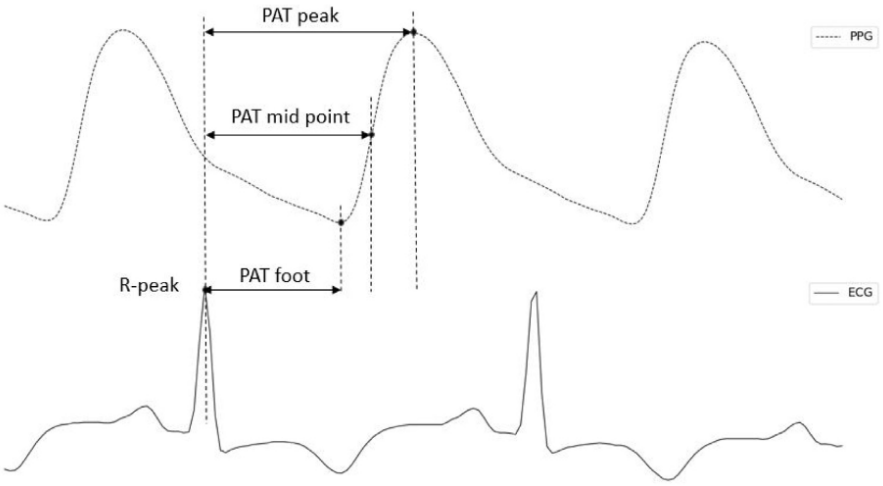


Figure 7: Pulse Arrival Time (PAT) visualisation [18]

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,

$$PAT = PTT + PEP \tag{3}$$

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 [14] [18]. The relation between PTT and PWV can be expressed as

$$PWV = \frac{d}{PTT} \tag{4}$$

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,

$$PWV = \sqrt{\frac{Eh}{\rho d}} \tag{5}$$

where

$$PWV = \text{ Velocity of the pulse wave (m/s)} \tag{6}$$
$$E = \text{ Young's modulus of vessel wall (Pa)} \tag{7}$$
$$h = \text{ vessel thickness (m)} \tag{8}$$
$$\rho = \text{ blood density (kg/}m^3\text{)} \tag{9}$$
$$d = \text{ arterial diameter (m)} \tag{10}$$

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

$$E = E_0 e^{\lambda P} \tag{11}$$

where E_0 and λ depend on the thoracic and abdominal aortas and P is the vessel blood pressure (mmHg) [2] [9] [26].

By equating and solving Equations 4 and 5, the final equation for estimated blood pressure is expressed as,

$$P = \frac{1}{\lambda} \ln \left(2r\rho \frac{\Delta X^2}{E_0 h}\right) - \frac{2}{\lambda} \ln \left(PTT\right) \tag{12}$$

where

$$r = \frac{d}{2} = \text{ arterial radius} \tag{13}$$

$$\Delta X = \text{ distance from heart to vessel} \tag{14}$$

2.2.4 Limitations

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 [25]. In addition, there is a very likely possibility that these sensor devices will have different real-time sampling rates. In order to be able to continuously measure BP, constant calibration of the methods is required [18]. This is due to individual patients having different physiological parameters [25]. 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 [18].

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 estimation using these parameters. As a result, the use of these parameters is not recommended for investigations which involve signal data over a long period of time, due to data processing limitations. 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. Finally, all other parameters important to defining neural network architectures will be defined.

2.3.1 Artificial neural networks

Artificial Neural Networks (ANNs) are a family of data-driven techniques. These techniques are successfully able to automatically extract features from the input data, instead of creating and solving 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 [14]. However, in this case BP estimation is motivated by how much data is available to the algorithm [18].

Artificial neural networks (ANNs) are a machine learning method that can be used to estimate blood pressure [14]. ANNs are based on the neural networks found in the human body and aim to replicate their behaviour [26]. The structure of the ANN consists of multiple individual units called neurons. A single neuron is shown in Figure 8.

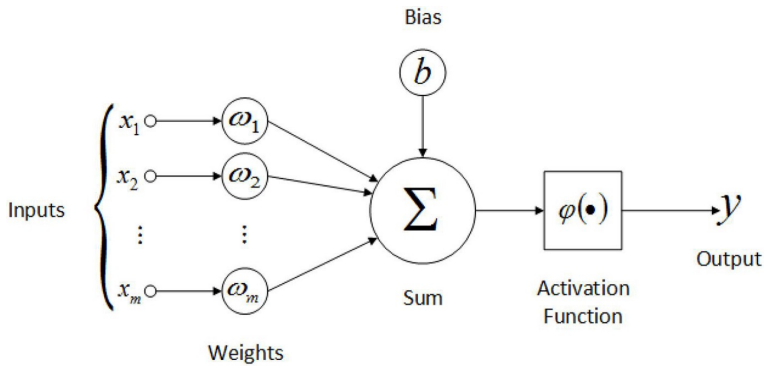


Figure 8: Structure of a neuron [27]

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) [28]. In order to mathematically express the neuron unit, it is important to first understand its goal. The neuron applies a dot product with the weights $\boldsymbol{\omega}$ and adds a scalar bias b. After this, a non-linear activation function $\phi(\cdot)$ is applied, which enables non-linear relationships, such as those seen in cuffless estimation of blood pressure, to be modelled. This is expressed mathematically as follows,

$$y = \phi(\sum_i \omega_i x_i + b) = \phi(\boldsymbol{\omega}^T \mathbf{x} + b) \tag{15}$$

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 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 16,

$$\mathbf{y} = \phi(\mathbf{\Omega x} + \mathbf{b}) \tag{16}$$

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.

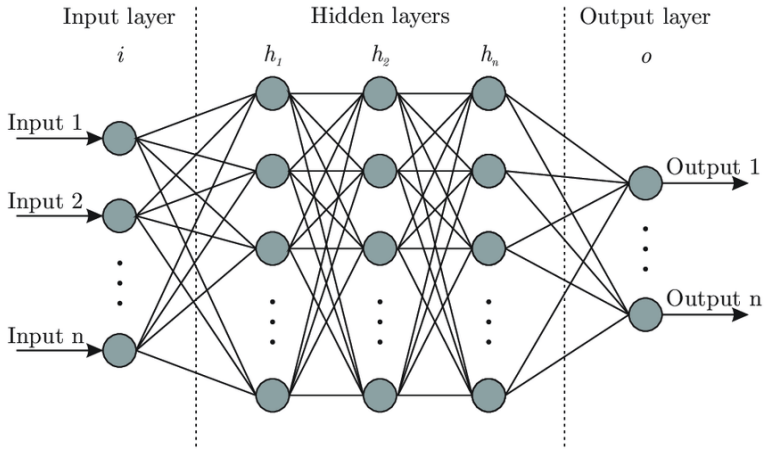


Figure 9: Structure of a multi-layer ANN [28]

Convolutional Neural Networks (CNNs) are a specific class of ANNs. Two-dimensional CNNs were originally developed for two-dimensional image classification problems. However, for this FYP the CNNs accept one-dimensional data inputs. CNNs and all other neural network architectures can either be given specific of the data as input or can be given the whole dataset to learn from. In the second case, the neural network itself extracts the strongest features from the data in order to make accurate estimations. Three high performing one-dimensional CNN models are the AlexNet [29], ResNet [30] and ResNet-LOSO (Leave One Subject Out) [31] architectures.

2.3.2 Recurrent Neural Networks (RNNs)

Although CNNs have the potential to extract features from time-series data, they face issues as the input sizes increase. Larger sizes result in a much longer time in training on the data, leading to large increases in computational complexity. A Recurrent Neural Network (RNN) is a specific type of architecture that is widely used to deal with time-varying data [32]. 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 [32]. 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 [28]. Figure 10 shows the simplest version of an RNN, which can be easily derived from a simple feedforward architecture by adding a single loop:

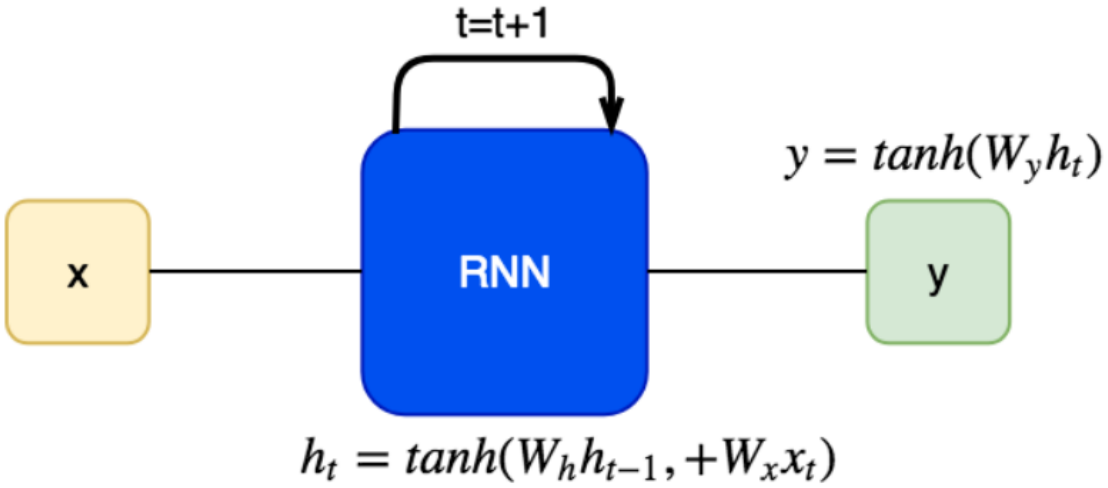


Figure 10: Simplification of a RNN [28]

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 [33]. This is known as the vanishing gradient problem. To overcome this limitation, in practical applications the Long Short Term Memory (LSTM) unit is adopted. This is a special RNNs architecture composed of multiple interacting layers.

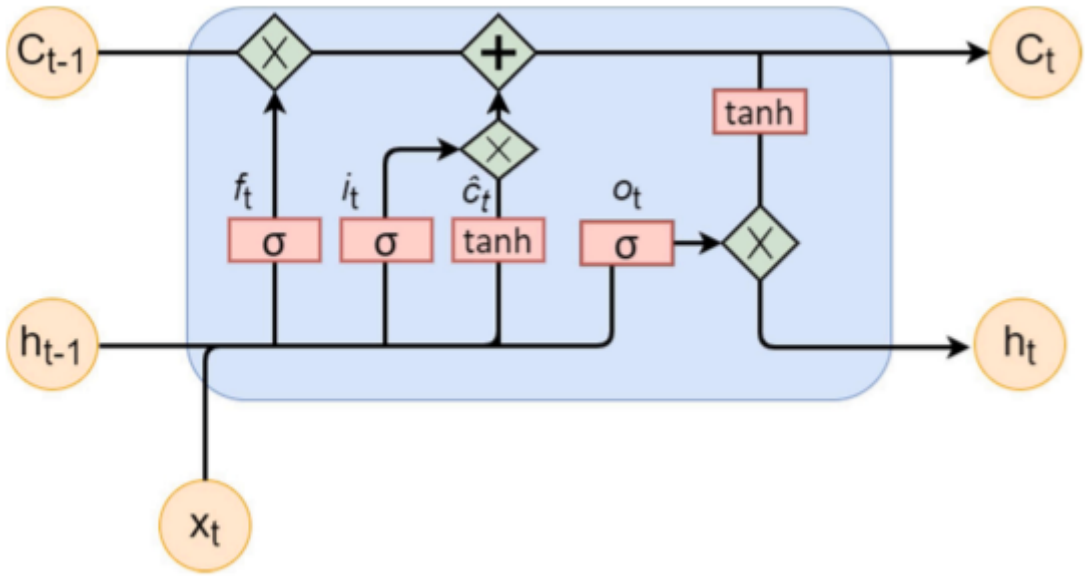


Figure 11: Structure of an LSTM cell [34]

As previously noted, LSTMs are more efficient than the standard RNNs in processing sequential data, as they are able to learn long-term dependencies in the data, which effectively removes the vanishing gradient problem seen in RNNs. There are two other main variants of LSTMs. Firstly the Gated Recurrent Unit (GRU), which uses fewer parameters and does not have an output gate in comparison with the LSTM, and the Bi-directional LSTM (Bi-LSTM), which is able to process the sequential input data in both the forwards and backwards direction.

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 [35]. Transformers were originally created for solving problems in Natural Language Processing (NLP) but can also be extended to time-series sequential data. Unlike RNNs and their variants, transformers do not use any recurrent connections and hence have no memory of previous data during training. However, transformers solve this problem by processing whole segments of sequential data at a time. As a result of this, the depth of the neural network is reduced, leading to a decrease in computational complexity. For the purposes of this FYP, the focus will be on the transformer encoder architecture. The main building block of a transformer encoder is the Encoder block, which is represented as a block diagram in Figure 12.

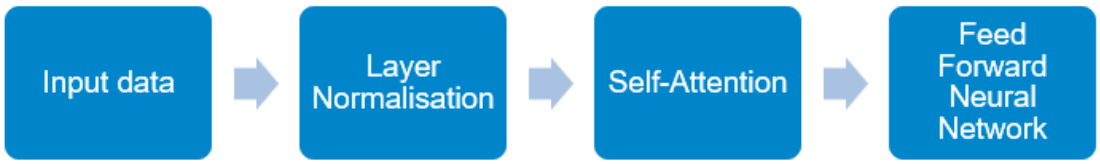


Figure 12: Block diagram to illustrate the Encoder block [35]

The following stages are described as follows:

- **Layer Normalisation**: applies a normlisation operation across the whole layer of data inputs
- **Self-Head Attention**: performs 3 main tasks; firstly it performs a dot product similarity to find alignment scores, then performs normalization of the scores to get the weights and finally reweighs the original embeddings using the weights
- **Feed Forward Neural Network**: 1-dimensional CNN that processes the data from the output of the attention block

It is important to note that the main limitation of transformers is that they require a lot of data to learn effectively [35]. Having provided an overview of three families of neural networks, it is now necessary to introduce a few additional concepts for the purposes of this FYP.

2.3.4 Activation Functions

Activation functions are able to model non-linear functions by applying a non-linear mathematical operation to the input. For this project, the estimation of BP is treated as a regression problem. Hence, the rectified linear unit (ReLU) activation function is ideally used during the training process, and is mathematically described as follows,

$$f(x) = \max(0, x) \tag{17}$$

where f is the output of the ReLU activation function and x is the input.

2.3.5 Loss Functions

During the training process of neural networks, their aim is to minimise their loss functions, so that they can achieve the best possible estimation accuracy. For this project, the loss function of concern is the Mean Squared Error (MSE) loss function, which is defined in Equation 18.

$$l_{\text{MSE}}\left(\mathbf{y}, \hat{\mathbf{y}}\right) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - \hat{\mathbf{y}}_\mathbf{i})^2 \tag{18}$$

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 loss function was chosen, as it has been proven to be more robust to outliers in the predictions by applying larger penalty weights through the squaring operation. The training process is finishing after finding the optimal weights (per layer) of the model as expressed in Equation 19, where $\hat{\mathbf{y}}$ is the estimate of the true label $\mathbf{y}$ and l is the loss function.

$$\mathbf{\Omega}_{opt} = \arg\min_{\mathbf{\Omega}}\{\mathbb{E}(l(\mathbf{y}, \hat{\mathbf{y}}))\} \tag{19}$$

Overfitting occurs when the neural network model performs well on the training input data used but performs poorly on the unseen test data. Overfitting is clearly an indicator of poor performance and can be avoided by introducing a validation dataset, which is just a subset of the training data which is not used by the neural network until the very end of the training process. After training the model on the training data, the learned model can then be evaluated on the validation dataset to give the user an insight as to how well the model performs on a set of unseen data.

2.3.6 Complexity of neural networks

There are two main factors associated with the complexity of neural networks:

- **Input data**: The larger the size of the input data, the greater the number of neurons are required in the neural network architecture. As a result, this leads to a increase in computational time and hence how long it takes to train the neural network model. Specifically for this project, if a large number of features are extracted from the PPG signals, then the computational complexity will significantly increase

- **Depth of the model**: The depth refers to how many hidden layers are present in the neural network architecture. As this Chapter on neural networks has progressed, several neural network architectures have been discussed, such as CNNs, RNNs, LSTMs

and Transformer Encoders. The depths of the networks are greater for the deep neural networks, such as the CNNs and RNNs. Increasing depth leads to a significant increase in the computational complexity of calcuations. As a result, this leads to increases in both the training and inference times. The inference time is the time taken for the model to make a prediction on the test dataset

2.3.7 Conclusion

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

This chapter provides a detailed account of the literature review conducted for this FYP. The literature search equation used will first be discussed. This is then 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* [18] and *Continuous Blood Pressure Estimation From Electrocardiogram and Photoplethysmogram During Arrhythmias* [23]. 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 [36]

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 [37]. The process is illustrated in Figure 13.

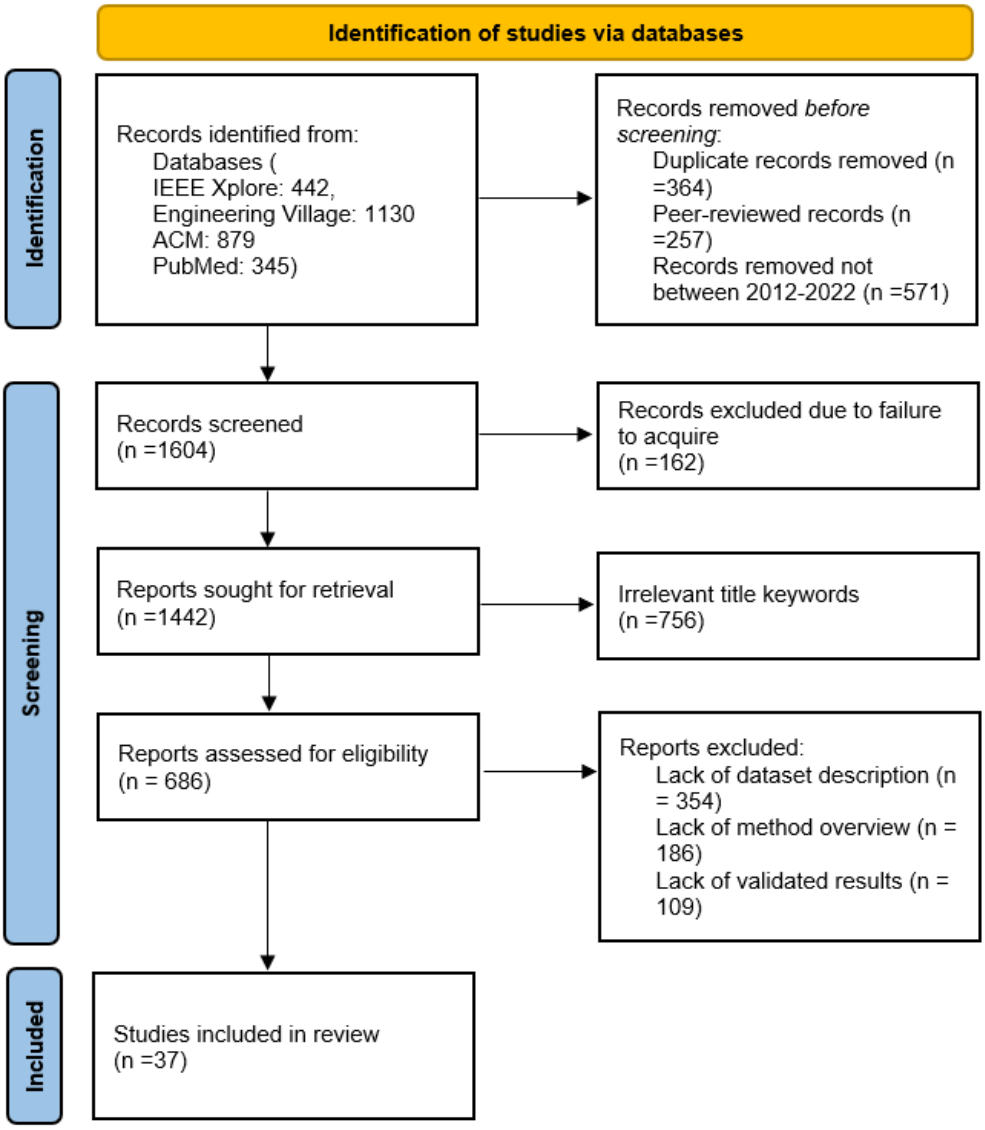


Figure 13: PRISMA checklist flow diagram (correct to 2020 guidelines) [37]

2.4.3 Literature survey table

As a reference, the original literature survey matrix can be found on the Github repository [38]. Firstly, in Table 3, a literature survey matrix 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 et al. (Year)	**Signal**	**No. Subjects**	**Age**	**Method**	**MAE SBP**
Ahmad (2012) [39]	ECG, PTT	10	24-63	Numerical solution	± 5.93
Chen (2013) [40]	ECG	5	N/A	Analytical solution	9 ± 5.6
Daimiwal (2014) [41]	PPG	16	18-48	Frequency analysis	0.8 ± 7
Yamanaka (2016) [42]	PTT	127	N/A	Wavelet transforms	± 7.63
Ding (2016) [43]	PTT, PPG	27	21-29	Analytical solution	-0.37 ± 5.21
Chatterjee (2019) [44]	PPG	22	22-45	Cubic regression	N/A
Lubin (2020) [45]	PPG	6	N/A	Analytical solution	N/A
Pielmuş (2019) [46]	PPG	10	26 ± 3.3	Polynomial regression	N/A

In addition, Table 4 provides an overview of the best performing Machine Learning based methods for cuffless estimation of blood pressure values.

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

Study et al. (Year)	**Signal**	**No. Subjects**	**Age**	**Method**	**MAE SBP**
Yang (2020)[26]	ECG, PPG	14 males	17-43	ANN	7.99 ± 10.34
Gao (2016)[47]	PPG	65	22-65	Wavelet, SVM	5.1 ± 4.34
Kachuee (2015)[48]	PPG	MIMIC II	Adults	Linear Reg., ANN, SVM	13.84 ± 17.56
Simjanoska (2018)[7]	ECG	51	16-83	Complexity analysis + ML	7.72 ± 10.22
Wang (2018)[10]	PPG	72	N/A	ANN (MLP)	4.02 ± 2.79
Pradenas (2020) [14]	ECG, PPG	MIMIC II	Adults, neonatal	ANN (150 neurons)	5.76 ± 6.39
Tanveer (2018) [9]	ECG, PPG	39	20-100	ANN-LSTM	1.10
Chen (2019) [49]	PTT, ECG, PPG	MIMIC I	N/A	SVM, Lin Reg.	N/A
Ripoll (2019) [50]	PTT	250	MIMIC I	ANN-RBM	3.70
Şentürk (2018) [51]	ECG, PPG	N/A (MIMIC II)	N/A	ANN	N/A
Nath (2018) [52]	PPG	20 (MIMIC I)	N/A	Random Forest classifier	N/A
Maqsood (2021) [53]	PPG	219	21-86	Bi-LSTM	3.87 ± 4.79
Li (2021) [54]	PPG	8000	N/A	PAT features	4.881 ± 5.537
Shimazaki (2018) [55]	PPG	1363	38-75	Autoencoder	N/A
Şentürk (2018) [56]	PPG, ECG	N/A	N/A	Bi-LSTM	N/A
Xie (2018) [57]	PPG	11492	N/A	Random Forest	4.21 ± 7.59
Li (2020) [58]	PPG, ECG	12000	N/A	SVM	7.44 ± 7.37
Wang (2022) [59]	PPG	348	N/A	AlexNet (Transfer Learning)	0.00 ± 8.46
Kachuee (2017) [60]	ECG,PPG	3663	N/A (MIMIC-II)	AdaBoost	8.21 ± 5.45
Singla (2020) [61]	ECG, PPG	33	50-70	SVM-Linear Regression	0.2
Sertac (2020) [62]	PPG	216	21-86	SVM	13.57 ± 3.23
Jung (2019) [63]	ECG, PPG, PTT	9	20-29	SVM-Linear Regression	N/A
El Hajj (2020) [64]	PPG	500	N/A (MIMIC-II)	LSTM-GRU	1.43 ± 1.77
Shimazaki (2019) [65]	PPG	78	N/A	CNN	N/A
Yan (2019) [66]	ECG, PPG	604	N/A (MIMIC-II)	CNN	3.09 ± 2.76
Slapničar (2019)[31]	PPG	51	N/A (MIMIC-III)	ResNet-LOSO	9.43
Baek (2019) [67]	PPG, ECG	N/A (MIMIC-II)	N/A	CNN	5.32 ± 3.38
El Hajj (2021) [34]	PPG	500	N/A (MIMIC-II)	Transformer (Attention)	2.58 ± 3.35
Eom (2020) [68]	ECG, PPG	15	23-29	Transformer (Attention)	4.46 ± 4.06

In addition to the information displayed in the tables, the following comments can be made in reference to the complete Literature Survey table [38]:

- **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 literature search yielded very minimal discussion on the feasibility of long-term wearable implementations. Several of the non-ML based methods utilise the previously discussed features related to PPG and ECG signals, such as PTT, PAT [43] [39] [42]. However, as previously discussed these features are powerful enough for ambulatory monitoring, where recordings are taken for at least 24 hours
- **Preprocessing**, as several methods applied filtering-based methods to clean the signal and prepare it for further analysis, so that accurate estimations can be found
- **Feature extraction**, as several methods utilise the time-domain and frequency-domain features of both the ECG and PPG signals. 4 feasible time-domain PPG features were previously discussed in Chapter 2. There is a clear split between how the features are extracted, with there being methods which perform personalised checks to train particular features and other methods which rely on the network to automatically extract features

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 reviewed papers employ some form of neural network architecture, such as ANNs, LSTMs or Transformers with the attention mechanism. In addition, the Mean Absolute Error (MAE) values for SBP and DBP 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 some of the most recent Machine Learning techniques utilise both the ECG and PPG signals. However, several studies solely use the PPG signal for cuffless estimation and argue that the PPG signal itself has sufficient time-domain and frequency-domain physiological features to enable accurate cuffless estimation [31] [34] [59]. Therefore, my utilising solely the PPG signal for cuffless estimation, the complexity of the system input is significantly reduced without any significant reduction in accuracy.

Thirdly, the main data sources used are either physical recordings or online databases, in particular the MIMIC I, II and III. As this is a software-based FYP, the intention is to use one of these databases for further analysis. After further research, it was decided to use the MIMIC database, as it contains a wide variety of patients with cardiovascular diseases and is the easiest to integrate into code.

For the next point, regarding the specific neural network architecture to use for this FYP it is still unclear as to which method is best, as the methods have been tested on a variety of datasets and with different signal inputs. CNNs, LSTMs and Transformers all have excellent

performance (in terms of MAE SBP) but all have their own limitations. As a result, implementations will be made for all three families of neural network and their performances will be compared in the same testing conditions.

Finally, the literature review also yielded a wide variety of feature extraction techniques. The main two were determined to be handpicked, where the user themself picks the features to feed into the input, and automated feature extraction, where the model is able to decide which features are best for cuffless estimation. As a result, an attempt will be made in this FYP to see if there is any effect in using either of the two feature extraction methods.

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. Experiments will now be performed on several architectures of neural networks and there will be additional tests to see if there is any effect by using handpicked PPG features over automated feature extraction from the model. 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 14 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.

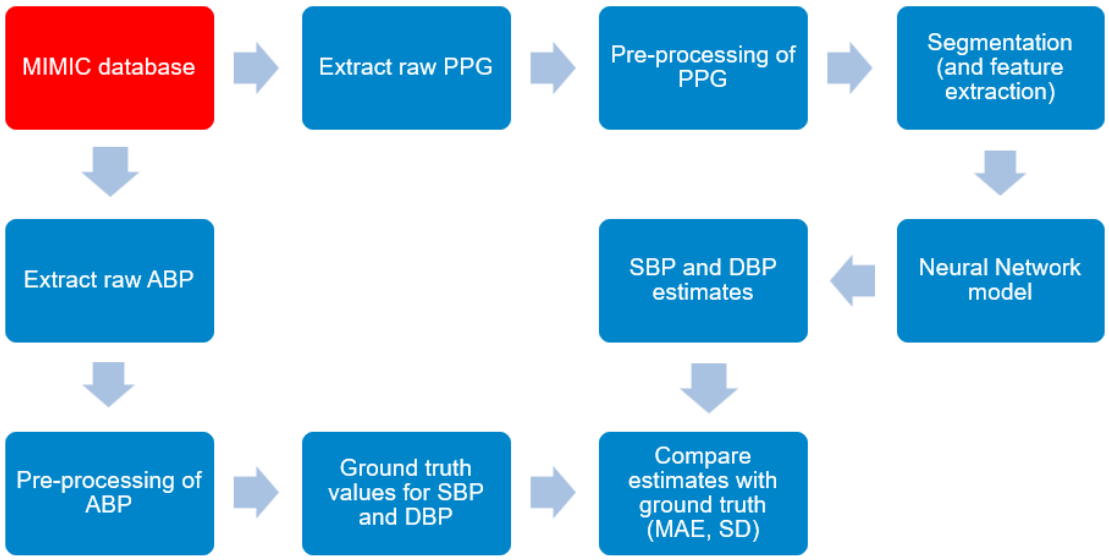


Figure 14: High-level block diagram of blood pressure cuffless estimation from PPG

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 [69]. 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 [70]
- **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 [71]
- **Myocardial Infarction** (MI) / **Cardiogenic shock**. This refers to patients who have suffered heart attacks or cardiac shock [72]

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 [73]. The scientific libraries from Python that are used for this project are `numpy` [74] and `pandas` [75]. In addition, the machine learning libraries used are `tensorflow` [76], `keras` [77] and `scikit-learn` [78]. In addition the `heartpy` [79] and `wfdb-python` [80] packages were installed, which are libraries of tools for reading, writing, and processing Waveform-Database (WFDB) signals and annotations. 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.

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` [81], and it also produces more compact and readable code.

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` Python package. In order to extract the ground truth values for Systolic and Diastolic blood pressure, the following steps are applied to the ABP signal (more detail is provided in Chapter 4):

- Use the Python `heartpy` library to perform sanity checks on the ABP signal (and the PPG) to check whether data can be accurately acquired
- Use a peak detection algorithm to find the Systolic and Diastolic BP and to perform sanity checks on these values

These steps are described in more detail in Chapter 4.

3.5 Pre-processing of PPG

The PPG signals from the Physionet database need to first be cleaned in order to be analysed accurately. The following steps describe how the PPG signal is pre-processed.

- Apply a 4th order Butterworth bandpass filter
- Apply Z-score normalisation in order to standardise the data
- Additional sanity checks using SBP and DBP minimum and maximum threshold values

3.6 Segmentation (and feature extraction)

Based on the findings of the literature review, the most recent machine learning methods are favouring automated feature extraction from signal data. As a result, the PPG signal itself will be split up, or segmented, into windows of 1 second, and each of these data windows will act as input to the neural network model. In addition, the literature review also suggests that there is still high performance among the machine learning methods which use handcrafted features. Therefore, as the aim is to assess the performance of the transformer encoder for cuffless blood pressure estimation, the following features will also be extracted from the PPG signal and fed into the neural network model (alongside the PPG window segment):

- First order derivative of the PPG signal window (with respect to time)
- Second order derivative of the PPG signal window (with respect to time)

The justification for adding these two features to the PPG window segment is that the transformer encoders require a large amount of data to perform well. Therefore, this suggests that purely training the neural network model on the features would be detrimental to performance.

3.7 Neural Network models

The following Neural Network models will be used to estimate cuffless blood pressure values from PPG signals:

- AlexNet (Convolutional Neural Network)
- ResNet (Convolutional Neural Network)
- ResNet with Leave One Subject Out (LOSO) (Convolutional Neural Network)
- Bi-directional Long Short Term Memory (Recurrent Neural Network)
- Transformer Encoder

All models above have been programmed using functions from the `tensorflow` and `keras` libraries and all code has been provided in the Appendix and on the Github repository. Further details on each of these architectures is provided in Chapter 4.

3.8 Error metrics

The error calculation used in this experimentation is the Mean Absolute Error (MAE) for both the Systolic and Diastolic blood pressure values. The MAE is considered so that the results can be compared with the official AAMI/ESH regulations for cuffless blood pressure estimation. MAE is defined by the following equation,

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|a_{i_M} - b_{i_M}| \tag{20}$$

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.

To conclude this chapter, a high-level specification of the method used to estimate cuffless blood pressure values from PPG signals has been detailed. Where appropriate, decisions made for the high-level specification have been based on the findings of the literature review in the previous chapter. Hence, now that the method has been discussed at a high-level, it is now necessary to discuss this implementation in more detail.

3.9 Evaluation of error metrics

In order to assess how well each of the neural network architectures perform in the cuffless estimation of blood pressure values, the MAE value produced will be be assessed against the standards set by the Advancement of Medical Instrumentation (AAMI) [82]: a MAE blood pressure difference of $\leq$ 5 mmHg with a standard deviation of $\leq$ 8 mmHg against the gold standard reference measurement.

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 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 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, as shown in Figure 15.

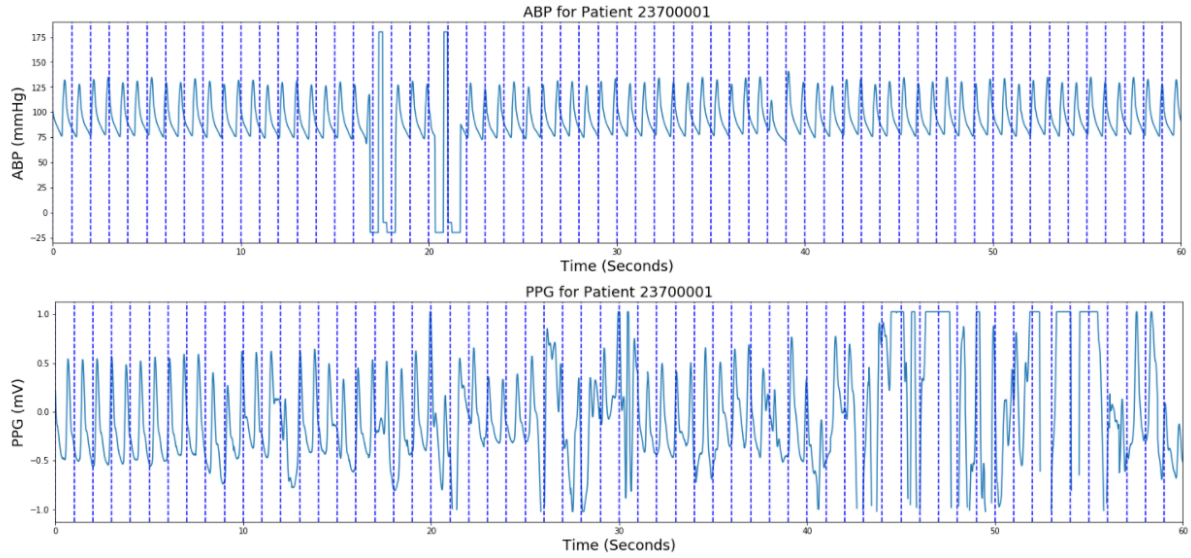


Figure 15: ABP and PPG signals for patient record 23700001

Based on the windowing segments in Figure 15, it was ascertained that sufficient physiological information about the PPG signal could be picked up within each window, as there is access to both a maximum and minimum. Hence, there would be sufficient PPG features for the neural network models to learn from within each window.

4.3 Choosing the patients

Based on the decision to use a window length of 1 second, it was necessary to take a subset of the dataset. This was because a small window length implies that a lot of datapoints are available for training already. For example, if there are 1000 seconds worth of data and a window of 10 seconds is applied, then there will be 100 discrete datapoints available for training. Therefore, it is not necessary to acquire data from all 61 patients in the MIMIC database. 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

After processing the data and checking for NaN (not a number) values in the patient recordings, Table 6 shows the number of samples available for each patient.

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 Extracting the ground truth blood pressure values

Before the ground truth blood pressure values are extracted from the ABP signals, the following sanity checks are applied:

- Check that the calculated Systolic peaks are within the range of $60 - 210 mmHg$. An additional threshold of $30 mmHg$ to the most extreme blood pressure values in Table 1 has been applied to account for the most serious cardiac patients
- Check that the calculated Diastolic peaks are within the range $30 - 140 mmHg$. The same threshold has been applied here as well

- The Normal-to-Normal (NN) interval and Heart Rate (HR) are calculated for each particular window.
- If the NN interval is not a reasonable value (within the range of 0.3 to 1.4 seconds) then the value is discarded
- If the heart rate is not a reasonable value (within the range of 50 to 140 Beats per minute), then the value is discarded
- Both the Systolic and Diastolic peaks are checked for NaNs (not a number) values
- Finally, within each signal window, the ground truth blood pressure value is taken as the median of all blood pressure values found in that particular window

4.5 Preprocessing the PPG signal

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

$$Z_i = \frac{(PPG_i - \mu_{PPG_i})}{\sigma_{PPG_i}} \tag{21}$$

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. Z-score normalisation was used instead of minimum-maximum (min-max) normalisation, as this method is less affected by outliers in the data. In addition, a 4th order Butterworth bandpass filter with cutoff frequencies of 0.5 and 40 Hz respectively are applied to the PPG. This is applied based on what was discussed in a previous implementation discussed in the literature review [52]. The filter is used to remove both low frequency noise and baseline wander from the signal. These operations are performed using the following Python code,

```
1  import numpy as np
2  from scipy.signal import butter, filtfilt
3  # 4th order butterworth filter for PPG preprocessing
4  b,a = butter(4,[0.5, 40], 'bandpass', fs=fs)
5
6  for i in range(0, NumberWindows):
7      ppg_win = filtfilt(b,a, ppg_win)
8      ppg_win = ppg_win - np.mean(ppg_win)
9      ppg_win = ppg_win/np.std(ppg_win)
10     ppg_record[i, :] = ppg_win
```

Figure 16: Python code for the preprocessing of the PPG signal

4.6 Processing of data before training

After the PPG signals of the MIMIC database subset have been preprocessed and segmented with 1-second windows, there are now 337998 PPG samples available in total for training with the neural network models. The data is split into training, testing and validation sets in the ratio of 50:25:25. It is also ensured in the Python code created that training is only performed on

the data of 1 subject at any time. This is extremely important because as discussed previously, the subjects are not the same in gender, age and have different cardiovascular diseases. Hence, the data is now ready to be learned on by the proposed neural network models. Hence, the distributions of the ground truth Systolic and Diastolic blood pressure values for the three datasets can be displayed for the training set in Figures 17 and 18.

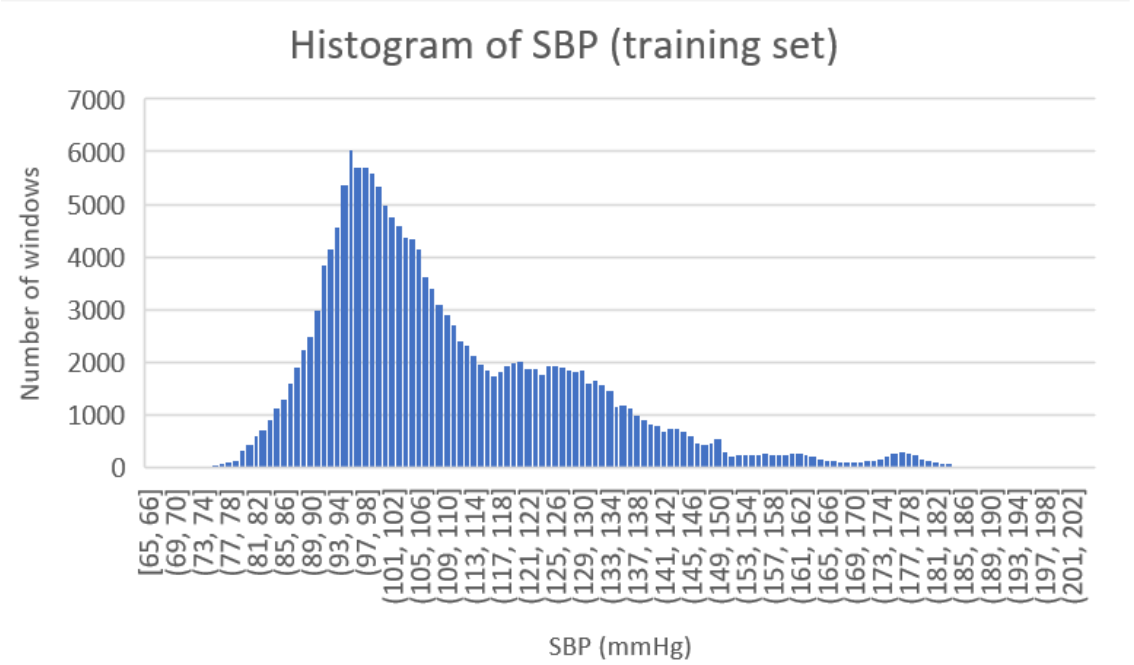


Figure 17: Histogram of the SBP ground truth values in the training set

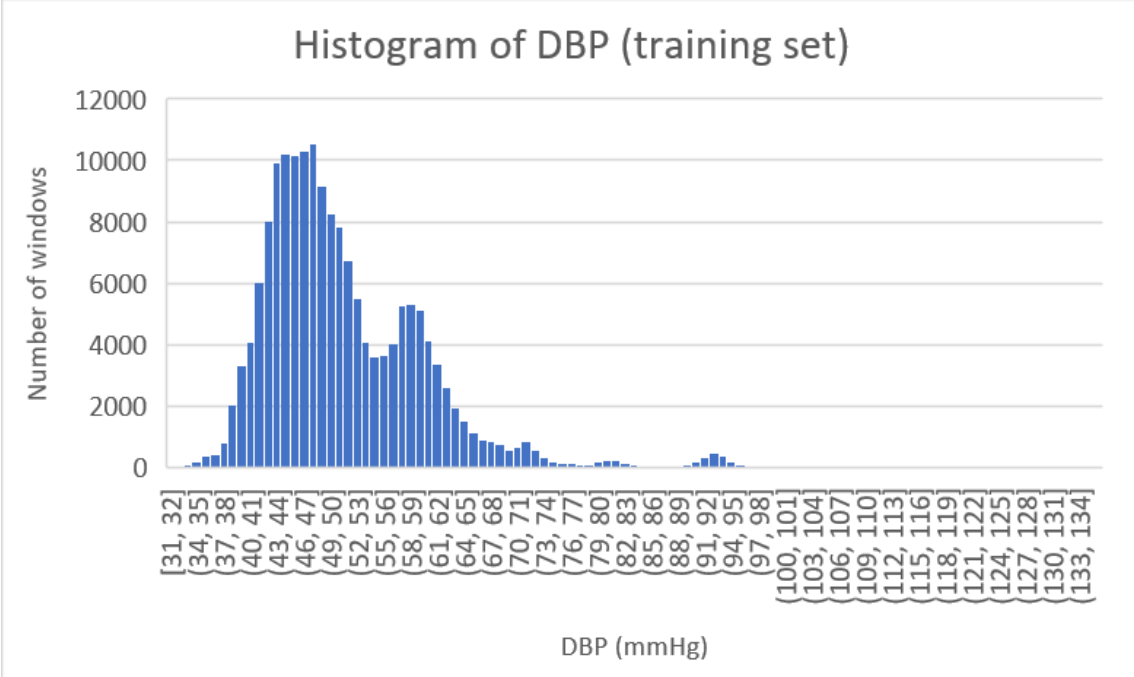


Figure 18: Histogram of the DBP ground truth values in the training set

The histograms of the SBP and DBP for the testing and validation sets also have a similar distribution to the training set, and hence are not included in this report.

4.7 Overview of Neural Network models used

In this section, a breakdown of the five proposed neural network models for experimentation, stated previously in Chapter 3, are discussed.

4.7.1 CNN AlexNet model

An overview of the AlexNet architecture [29] used for this FYP is demonstrated in Figure 19 (using the terminology from the Python Keras documentation).

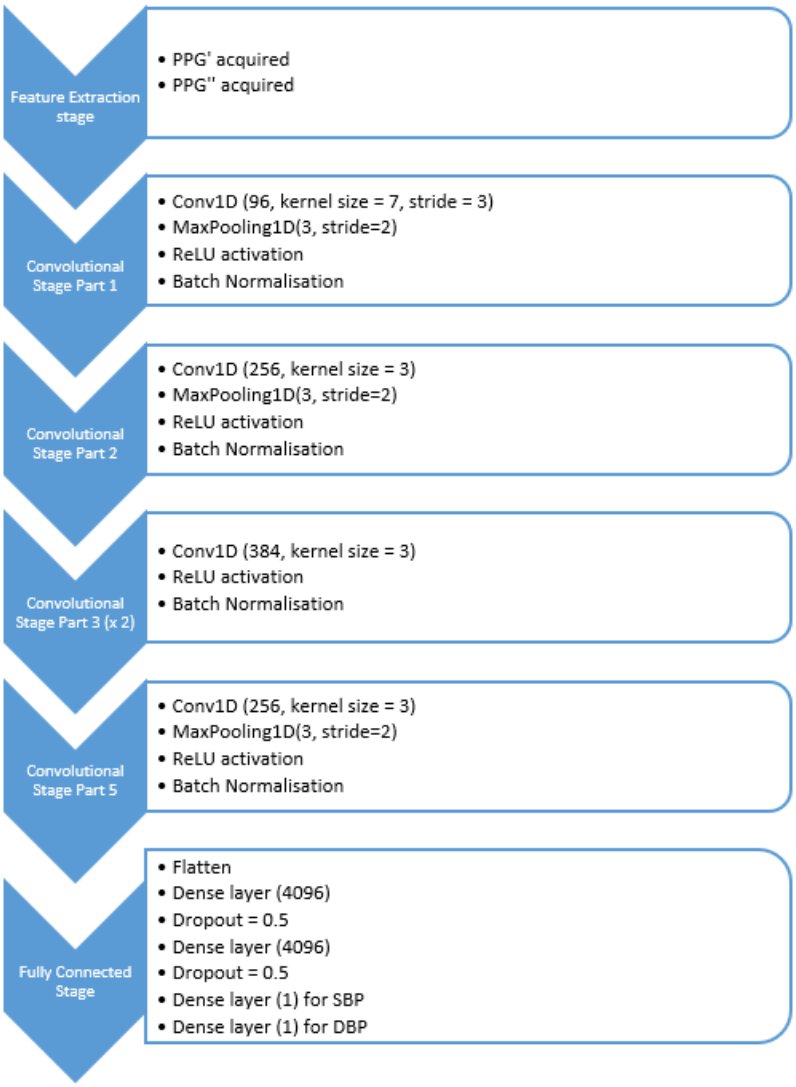


Figure 19: Overview of the AlexNet architecture

The following comments are made about the components of this architecture:

- PPG' and PPG" refer to the first and second derivatives of the PPG signal with respect to time. This is implemented using the following Python code,

```
import tensorflow as tf
# data_in_shape = size of windowed PPG segment
# Input function is the input layer to the neural network
                                        model
X_input = Input(shape=data_in_shape)
# sampling frequency
fs = 125
# first derivative calculation
dt1 = (X_input[:,1:] - X_input[:,:-1])*fs
# second derivative calculation
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)
```

Figure 20: Feature extraction of PPG first and second temporal derivatives

- In Figure 20, the first derivative is found by first calculating the difference between the input data and the input data with a delay of 1 sample. This difference is then multiplied by the PPG sampling frequency, $f_s = 125Hz$, as this can be effectively expressed as the rate of how many samples are processed in 1 second. This is mathematically expressed as,

$$PPG' = \Big(x_{PPG}(t+1) - x_{PPG}(t)\Big) \times f_s = \frac{d}{dt}\Big(x_{PPG}(t+1) - x_{PPG}(t)\Big) \tag{22}$$

$$\therefore PPG' = PPG' \times f_s = \frac{d}{dt}(PPG') \tag{23}$$

  In addition, a further multiplication of the first derivative by the sampling frequency produces a result for the second derivative, PPG'', as illustrated above. This code is applied directly to the four other proposed neural network architectures

- Conv1D(96, kernel size = 7, stride = 3) is a 1-dimensional convolutional layer with 96 filters. Kernel size specifies the size of the 1-dimensional convolutional window applied to the data. A stride of 3 means that the convolutional window will move 3 units at a time

- MaxPooling1D(3, stride = 3) is a 1-dimensional maximum pooling layer with a window size of 3. This layer takes the maximum value within each set window

- The ReLU activation function has been discussed previously in Chapter 2

- Batch Normalisation transforms the input data such that the average output is approximately 0 and the standard deviation is approximately 1

- The Flatten layer flattens the input data into a 1-dimensional vector

- The Dense layer (4096) is a fully connected layer which produces an output vector of size 4096
- The Dropout layer is used to prevent overfitting. The Dropout layer randomly sets a fraction of the input units to 0 (in this case 0.5)
- The Systolic and Diastolic blood pressure estimates are returned through a Dense layer with an output vector dimensionality of 1

4.7.2 CNN ResNet model

The architecture of the ResNet model is directly based on the 34-layer residual architecture described in the paper *Deep Residual Learning for Image Recognition* [30], and is illustrated in Figure 21.

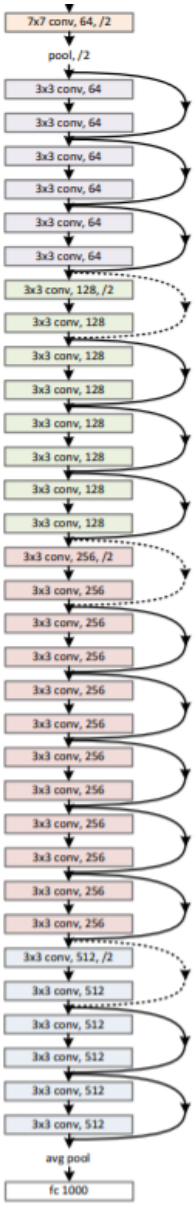


Figure 21: Overview of the ResNet architecture

The input is also modified to include a feature extraction stage for PPG' and PPG'' as discussed previously.

4.7.3 ResNet with Leave One Subject Out (LOSO) model

The architecture of the ResNet with LOSO model was proposed in the paper *Blood Pressure Estimation from Photoplethysmogram Using a Spectro-Temporal Deep Neural Network* [31]. This architecture is classified as a spectro-temporal neural network, as it utilises features of the signal in both the time and frequency domains.

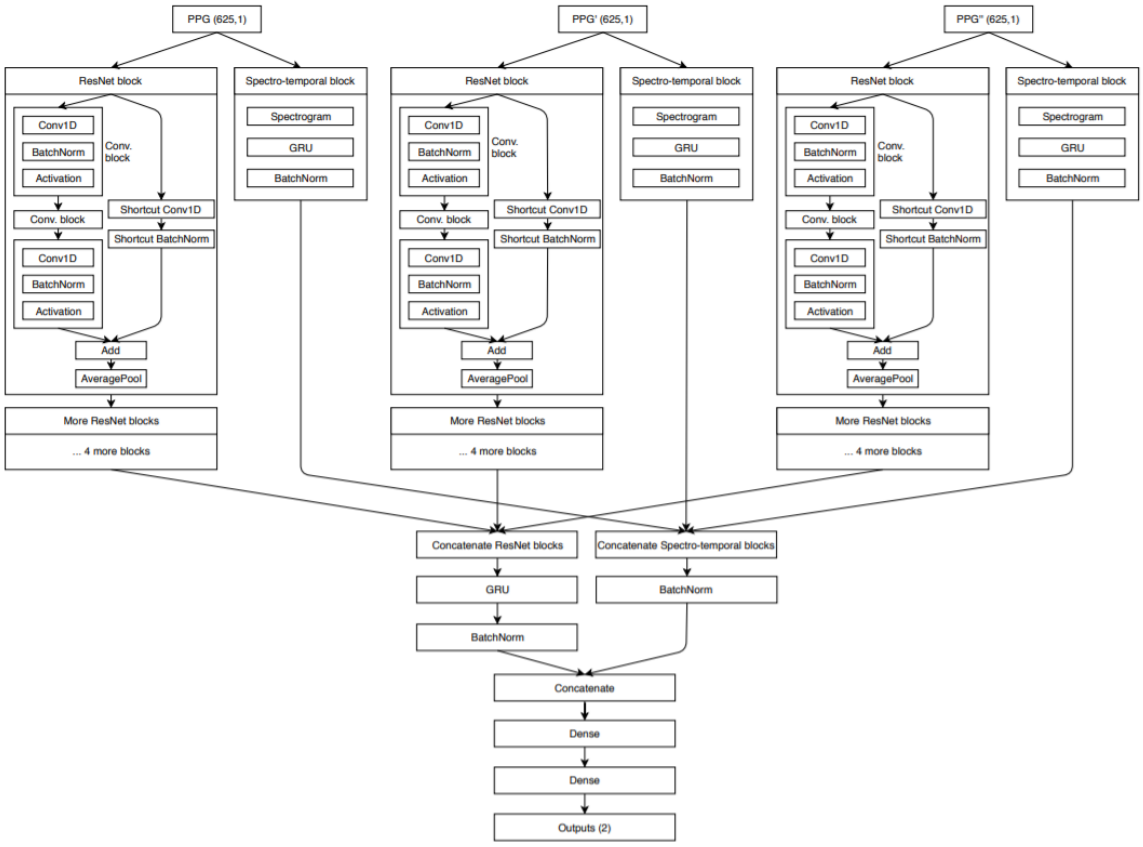


Figure 22: Overview of the ResNet-LOSO architecture [31]

The following comments are made about the components of this architecture:

- The term LOSO refers to how the experiment was conducted in the original paper. LOSO is a cross-validation technique where the dataset is split according to the number of subjects available. In addition to this, one subject is randomly selected for testing the model and the other subjects are used to train the model. This is repeated until all the subjects have been used for the test dataset.
- As indicated in Figure 22, the architecture consists of three distinct blocks, with the difference being what the blocks take as input. The three blocks take the windowed PPG, the first derivative and the second derivative of the PPG signal with respect to time

4.7.4 LSTM model

The Bi-directional LSTM model architecture used for experimentation in this FYP is illustrated in Figure 23.

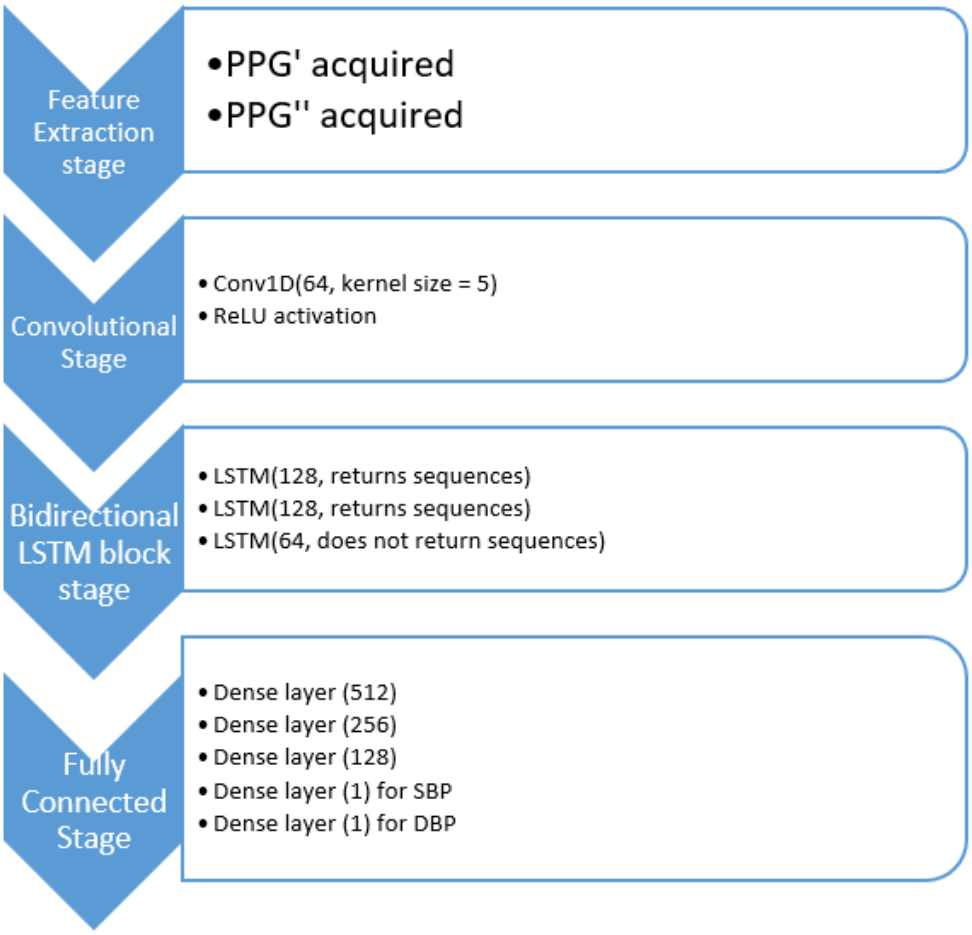


Figure 23: Overview of the Bi-directional LSTM architecture

The following comments are made about the components of this architecture:

- LSTM(128) creates an LSTM layer with 128 units
- When the LSTM layer returns sequences, the layer outputs the full sequence of hidden states, $[h_1, h_2, \ldots, h_n]$. Otherwise, only the last hidden state, h_n is returned as the output of the layer

4.7.5 Proposed Transformer Encoder model

The Transformer Encoder model architecture used for experimentation in this FYP is illustrated in Figure 23.

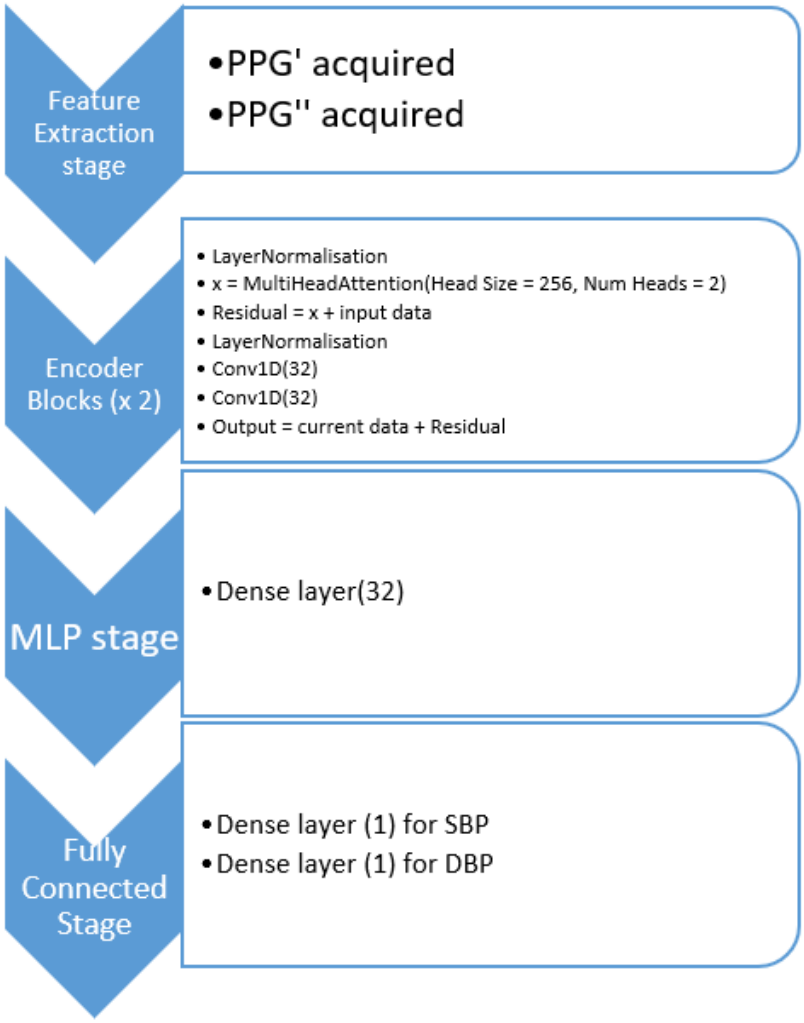


Figure 24: Overview of the Transformer Encoder architecture

The following comments are made about the components of this architecture:

- The architecture has the following parameters available that can be changed:
  - The number of encoder blocks
  - The head size in the encoder block
  - The number of heads in the encoder block
  - The number of filters in the Conv1D layers in the encoder block
  - The number of Multi Layer Perceptron (MLP) units
- For this FYP, these parameters have been chosen in order to address the tradeoff between minimizing the training time of the model whilst aiming to maximising the accuracy of the cuffless estimation of blood pressure values

To conclude this chapter, an overview has been given of the five neural network architectures used for the cuffless estimation of blood pressure values. The specific layers and parameters used in the Python code have also been highlighted for further understanding as to how the architectures differ from each other. Hence, it is now suitable to proceed with the testing plan in the next chapter.

5 Testing Plan

In this chapter, the aims are to:

- Highlight what neural network architectures are being tested for this FYP
- State what parameters were used for all experiments
- Explain how the results are being calculated
- Highlight any other measures that are calculated

The aim of this project is to assess the best performing neural network for the cuffless estimation of blood pressure values from PPG signals. As a result of the findings of the Literature review, it has been made clear that there are three distinct groups of neural networks that have been able to produce accurate estimates, according to the AAMI/ESH standards. These three groups are:

- **Convolutional Neural Networks (CNNs)**: AlexNet, ResNet and ResNet- Leave One Subject Out (LOSO)
- **Recurrent Neural Networks (RNNs)**: Bi-directional Long Short Term Memory (Bi-LSTM)
- **Transformer ANNs**: Transformer Encoders

The following parameters are used during the training, validation and testing processes of the models on the MIMIC I database subset:

- The training:test:validation dataset split is 50:25:25
- Loss function: Mean Squared Error
- Optimiser: Adam
- Learning rate: 0.005 arbitary units (AU)
- Batch size: 128 samples
- Window length: 1 second
- Number of epochs: 100
- Metrics: Mean Absolute Error (MAE) of the Systolic and Diastolic blood pressure values, measured in millimetres of Mercury (mmHg)
- For all testing performed in Google Colaboratory, the GPU being used to compute inference times was the Tesla T4 Persistence-M GPU
- In order to produce reproducible results, each of the five models were run 3 times, and the average of the minimum MAE for SBP and DBP is presented in the tables shown in the next chapter

To conclude this chapter, the overview of the testing strategy has now been clearly defined. Hence, it is now suitable to proceed with the collection of results.

6 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 in estimating cuffless blood pressure meets the required standards set in the previous chapter

6.1 Introduction

This overview is split into five sections, one for each neural network architecture. Each section will contain the following:

- Mean Absolute Error (MAE) of the Systolic and Diastolic blood pressure against Epochs curves for the training set
- MAE of SBP and DBP against Epochs curves for the validation set
- The above two sets of curves with the addition of the first and second derivatives of the PPG signal with respect to time
- MAE table of results for all architectures
- Training and inference times

6.1.1 Introductory disclaimer

The 4×4 subplots for the MAE are arranged in the following order:

- Top-left curve: Mean Absolute Error (MAE) of Systolic blood pressure estimation using the training dataset, in mmHg, against the number of Epochs, in Arbitrary Units (AU)
- Top-right curve: MAE of Diastolic blood pressure estimation using the training dataset, in mmHg, against the number of Epochs
- Bottom-left curve: MAE of SBP estimation using the validation dataset against Epochs
- Bottom-right curve: MAE of DBP estimation using the validation dataset against Epochs

6.2 AlexNet

Figure 25 shows the performance of the AlexNet architecture using only the PPG segmented windows. As illustrated below, the two training curves show a very significant drop in MAE after approximately 5 Epochs in training, indicating that the model has learned the patterns of the training data well. Regarding the two validation curves, despite there being a significant low MAE for approximately 50 Epochs of testing, there are noticeable spikes in MAE occuring during the remaining 50 Epochs.

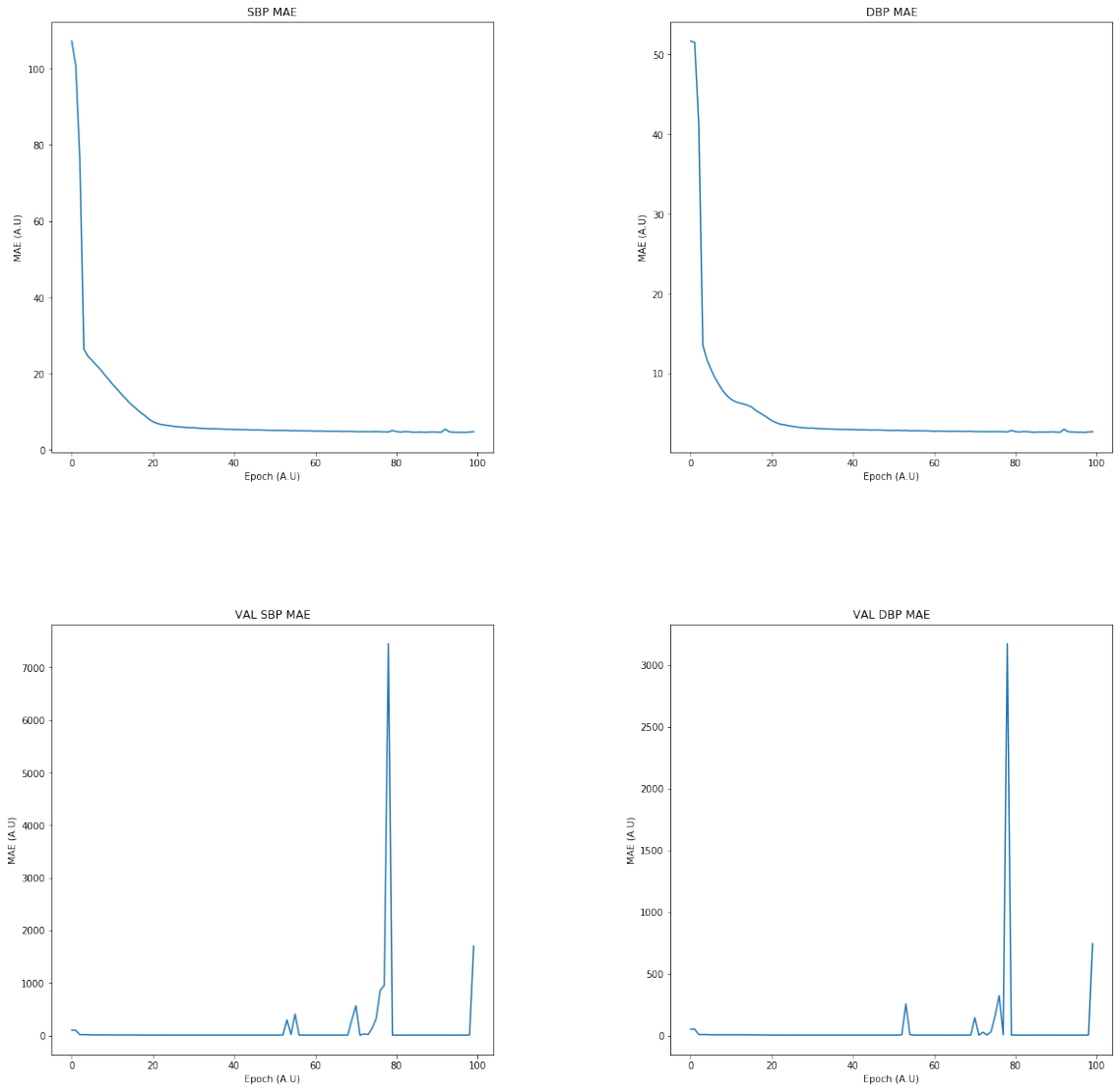


Figure 25: MAE of SBP and DBP for the AlexNet architecture

Figure 26 illustrates the same training and validation curves with the addition of the first and second derivatives of the PPG signal. This model also performs well on the training data, but also appears to have a better quality of estimation on the validation dataset, due to the lack of noticeable upward spikes in the MAE.

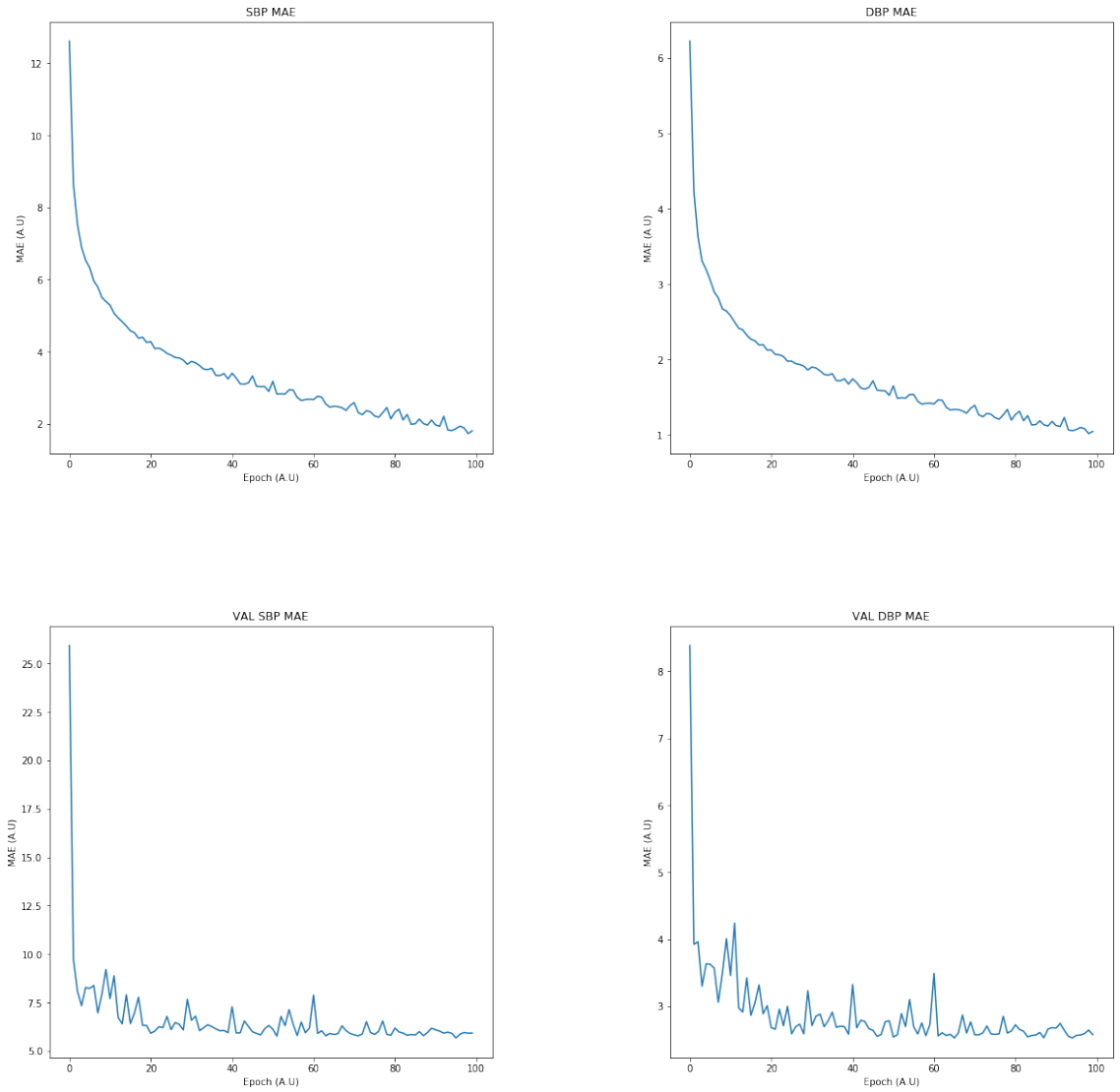


Figure 26: MAE of SBP and DBP for the AlexNet architecture with the PPG derivative features

6.3 ResNet

Figure 27 shows the performance of the ResNet architecture using only the PPG segmented windows. As illustrated below, the two training curves show a steady drop in MAE throughout the training process, indicating that the model has learned the patterns of the training data well. There are similar noticeable upward spikes in the validation curves, suggesting that this is a linking factor between the performance of CNN architectures on this particular dataset.

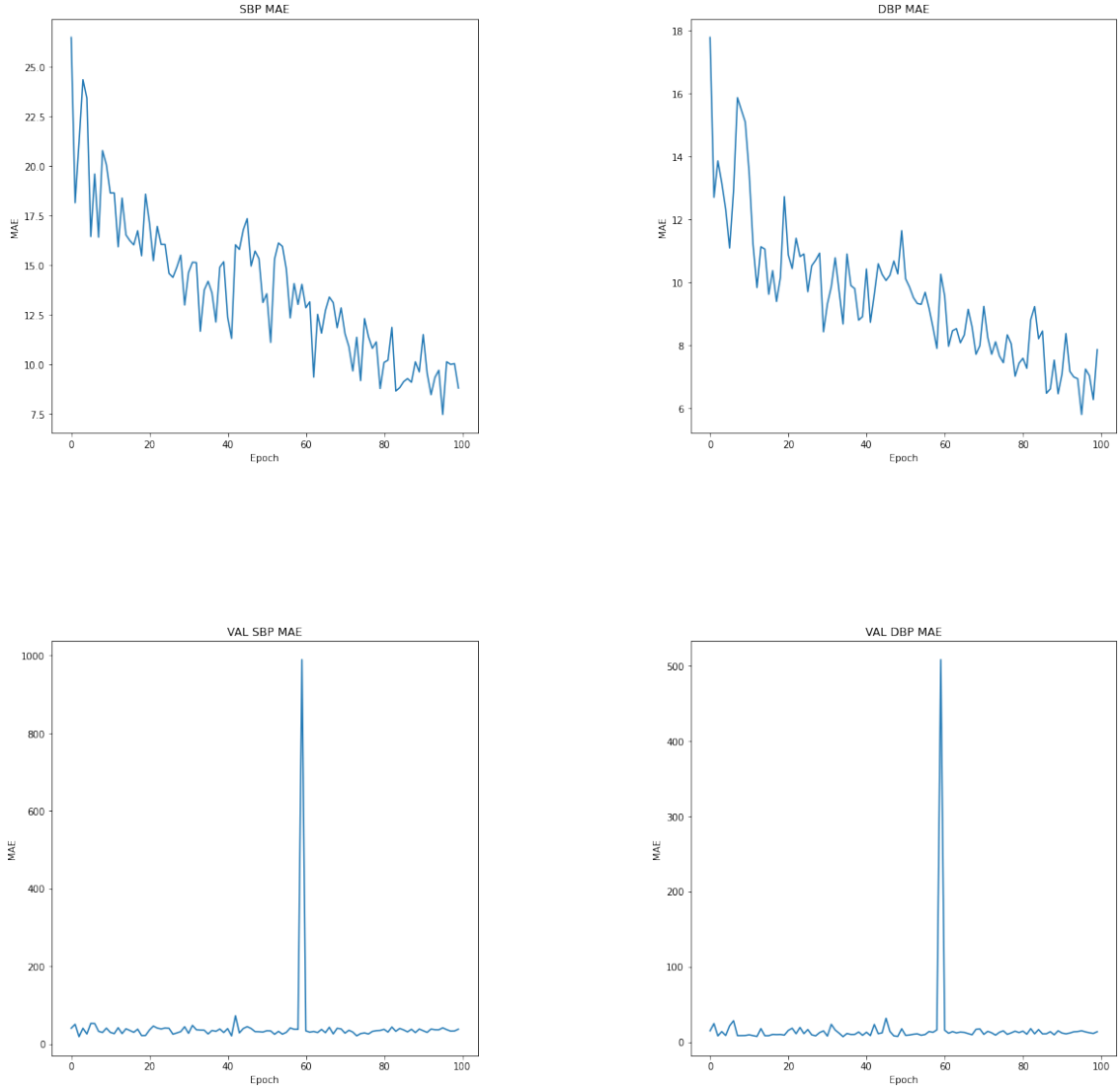


Figure 27: MAE of SBP and DBP for the ResNet architecture

Figure 28 illustrates the same training and validation curves with the addition of the first and second derivatives of the PPG signal. This model appears to perform better on the training data than the previous model, as there is less variation in the MAE curves for increasing Epochs. well on the training data. There is also a better quality of estimation on the validation dataset, due to both the sudden drop in MAE after approximately 4 Epochs and the lack of noticeable upward spikes in the MAE.

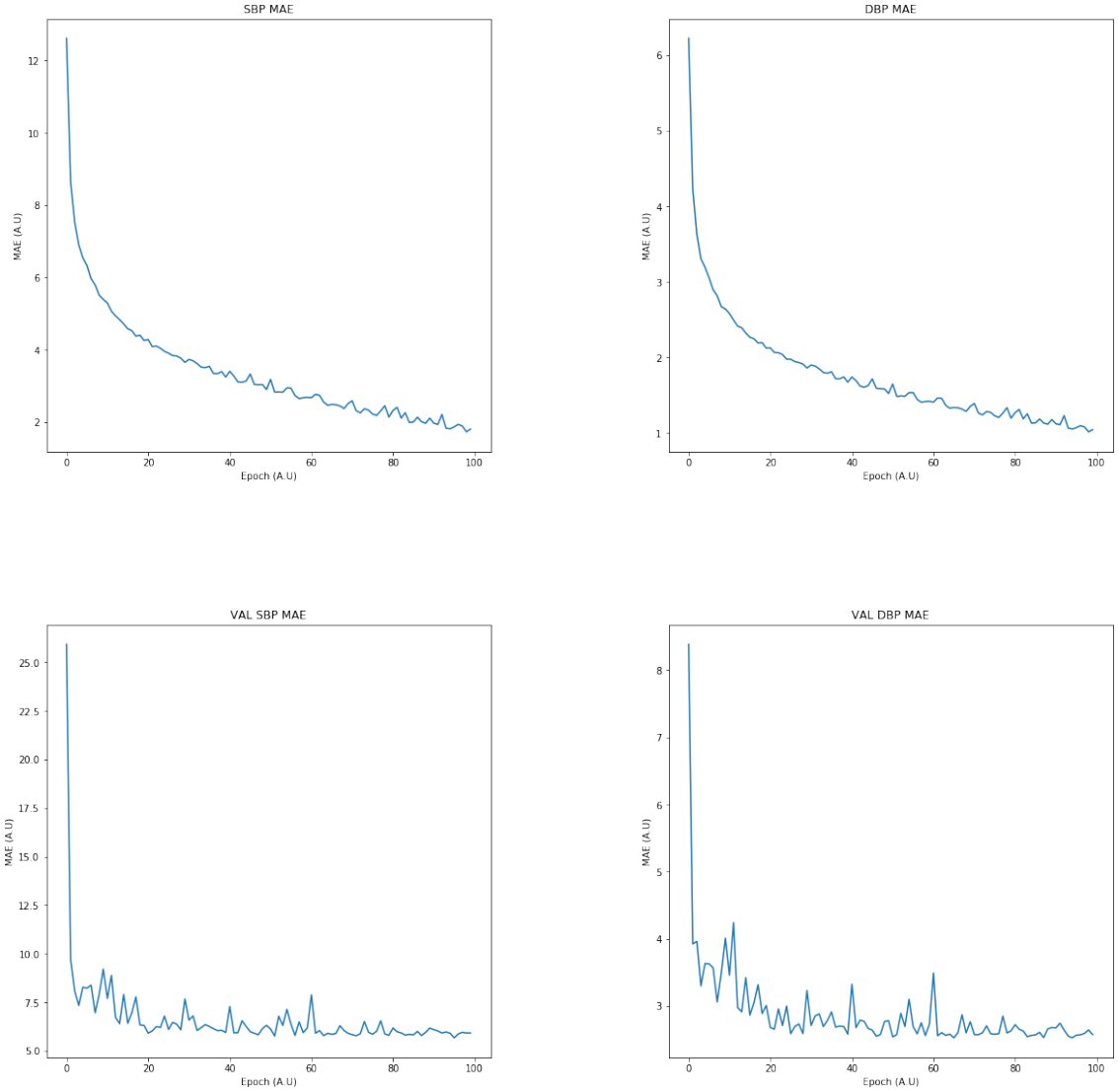


Figure 28: MAE of SBP and DBP for the ResNet architecture with the PPG derivative features

6.4 ResNet with LOSO

Figure 29 shows the performance of the ResNet-LOSO architecture using only the PPG segmented windows. As illustrated below, the two training curves show a steady drop in MAE throughout the training process, indicating that the model has learned the patterns of the training data well. There is now a reduction in the upward spikes in the validation curves, indicating that the implementation of an architecture that is both a function of frequency and time (or spectro-temporal architecture) is beneficial, as there are now more features available for extraction during each PPG window segment.

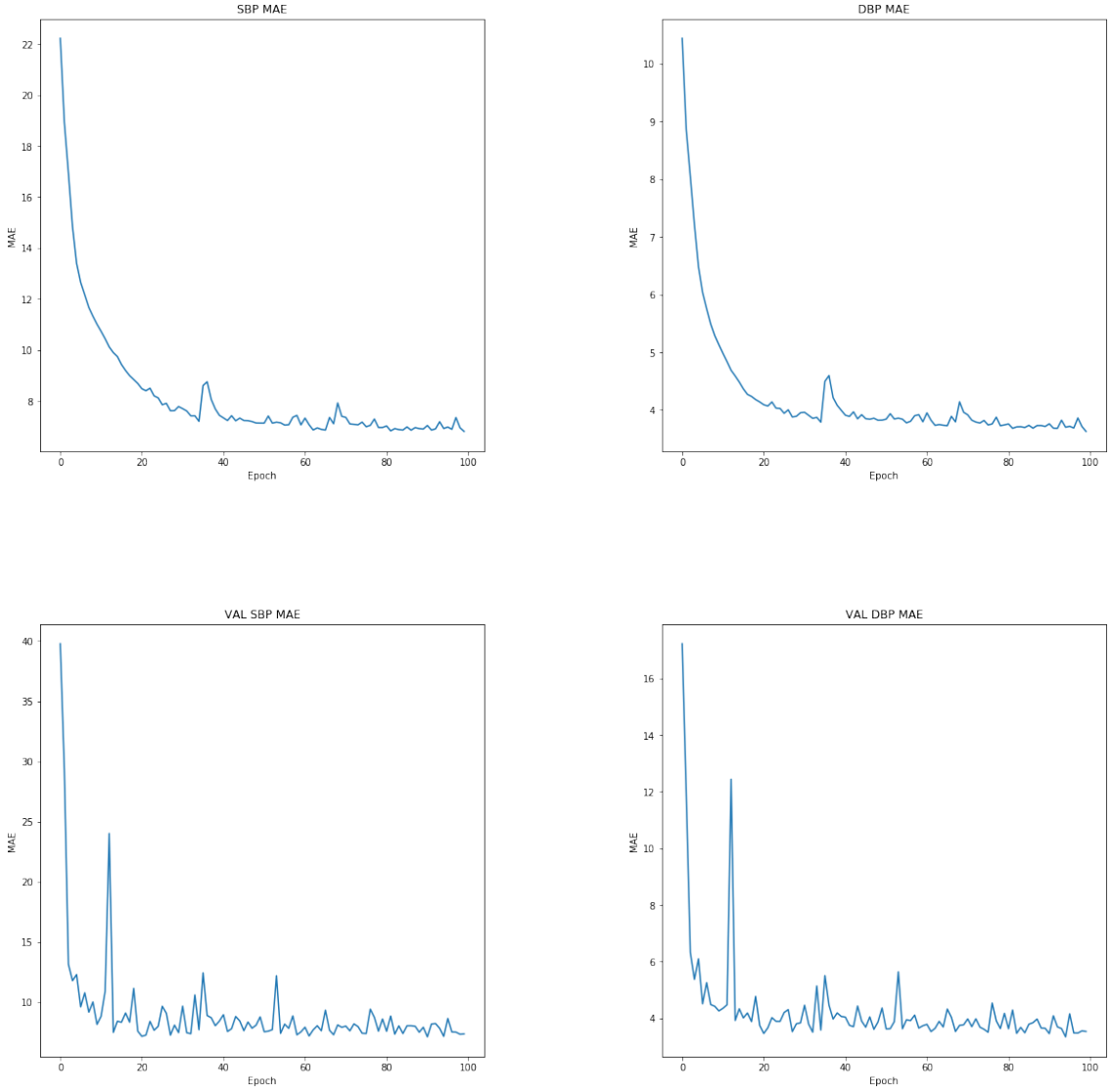


Figure 29: MAE of SBP and DBP for the ResNet-LOSO architecture

Figure 30 indicates that the performances of all curves are in fact very similar to the implementation without the additional PPG features. This suggests that the base ResNet-LOSO architecture has sufficient capabilities to automatically extract PPG features during each segmented window.

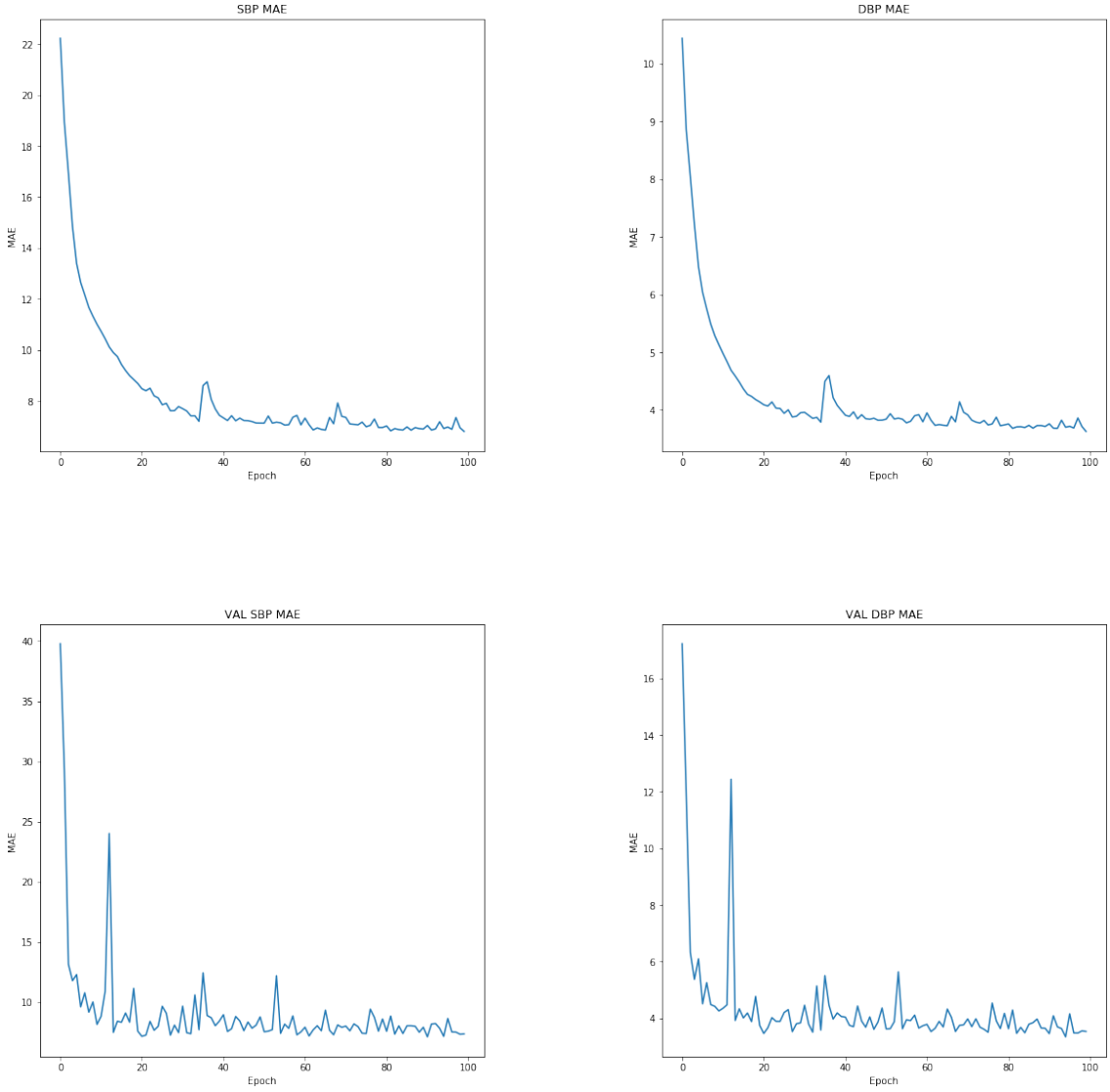


Figure 30: MAE of SBP and DBP for the ResNet-LOSO architecture with the PPG derivative features

6.5 Bi-directional LSTM

Figure 31 shows the performance of the Bi-directional LSTM architecture using only the PPG segmented windows. It is clear that the training curves do not perform as well on the training and validation data compared to the 3 previous CNN architectures, due to the volatile nature of all four curves.

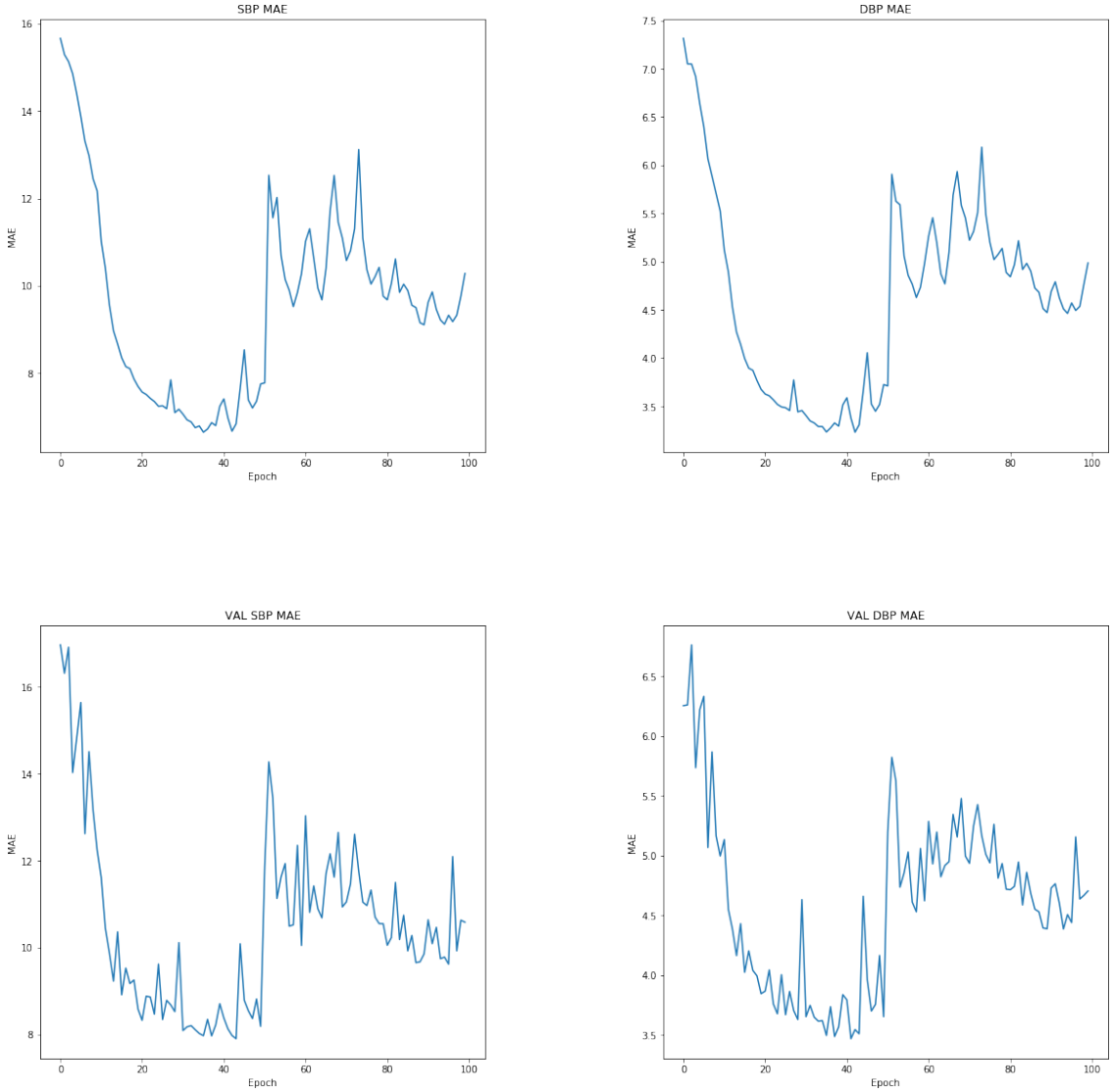


Figure 31: MAE of SBP and DBP for the LSTM architecture

Figure 32 shows that the addition of the PPG features does appear to smoothen the training curves, however it still does not have any significant effect on the validation curves.

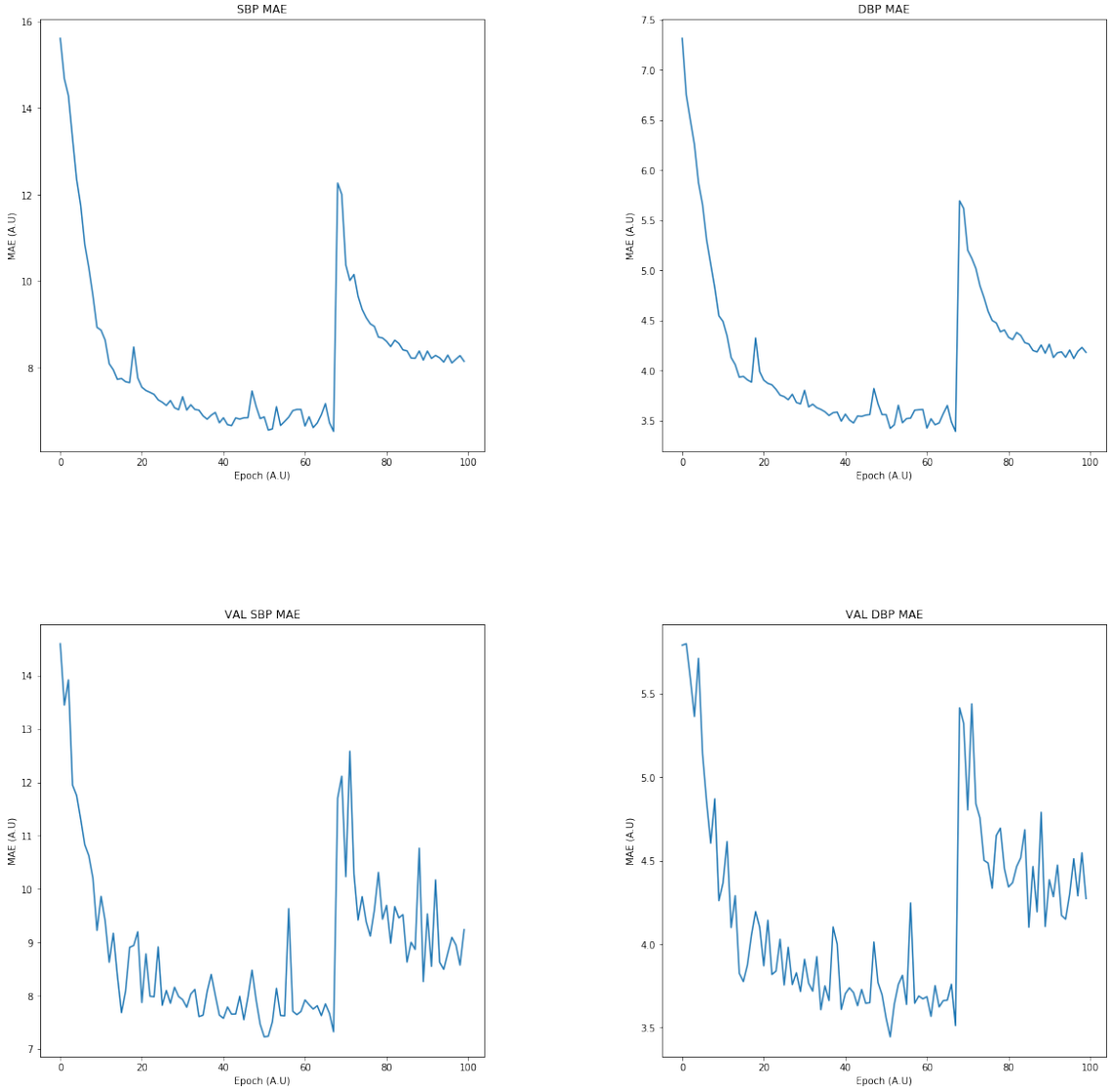


Figure 32: MAE of SBP and DBP for the LSTM architecture with the PPG derivative features

6.6 Transformer Encoder

Figure 33 shows the performance of the Transformer encoder architecture using only the PPG segmented windows. It is clear that the model has acceptable performance on the training dataset, due to the steady reduction in the MAE curves for increasing Epochs. However, for the validation dataset, although there is a gradual decrease in the MAE estimation, there is a lot of variance in the MAE for increasing Epochs.

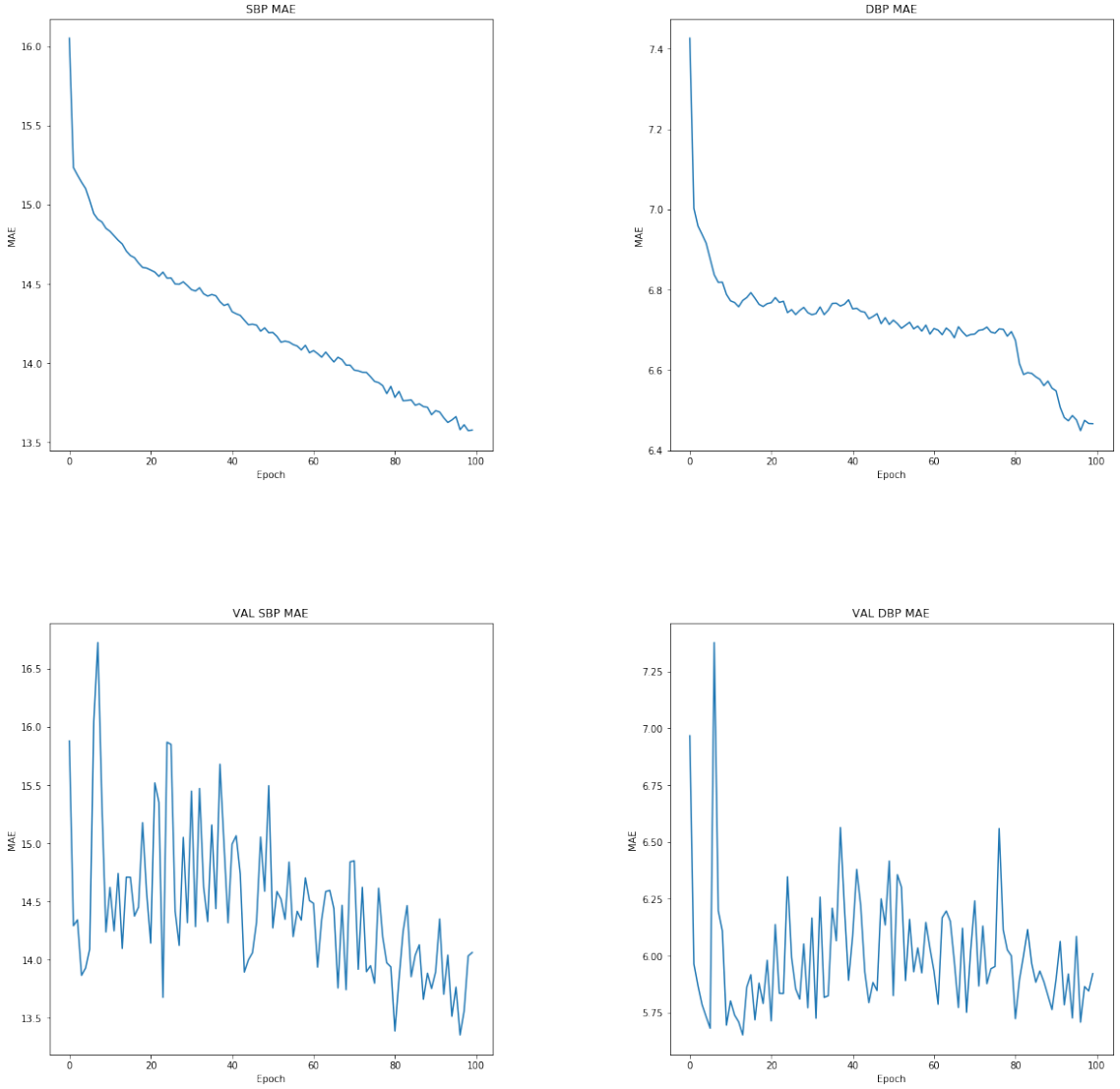


Figure 33: MAE of SBP and DBP for the Transformer encoder architecture

Figure 34 does indicate an improvement in the training curves, suggesting that it is beneficial to add the PPG derivative features to the model's input. However, there is still very minimal change in the validation curves.

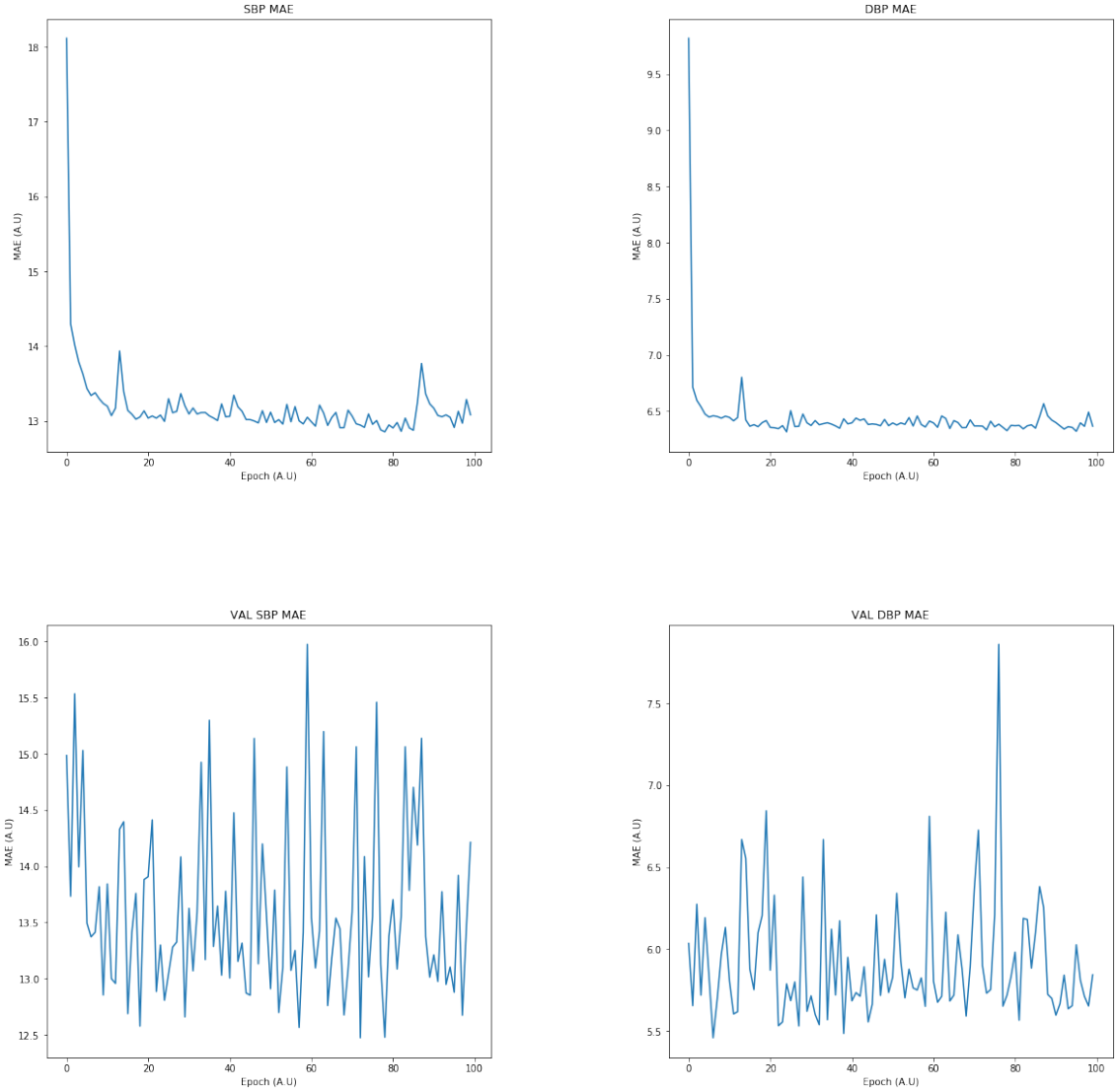


Figure 34: MAE of SBP and DBP for the Transformer encoder architecture with the PPG derivative features

6.7 Table of MAE results

Based on Figures 25 to 34, Table 7 summarises the performances of each of the five neural network architectures and illustrates the effect of the addition of the PPG derivative features.

Table 7: Minimum MAE of the five neural network architectures (average after running each configuration 3 times)

Neural network architecture	SBP MAE (mmHg)		DBP MAE (mmHg)	
	Training	Validation	Training	Validation
AlexNet	4.5867	4.6724	3.5892	3.7895
AlexNet with derivative	2.0134	6.7865	1.3424	2.9451
ResNet	7.7123	9.8753	6.0123	10.297
ResNet with derivative	1.9845	6.9732	1.2765	2.2145
ResNet-LOSO	7.5329	9.4234	4.2154	4.3785
ResNet-LOSO with derivative	7.4923	9.5246	3.7894	3.9720
LSTM	5.1370	8.2793	3.2159	3.4196
LSTM with derivative	5.1299	8.2432	3.7913	3.5217
Transformer Encoder	13.214	13.359	6.3412	5.6129
Transformer Encoder with derivative	12.992	12.740	6.4387	5.4763

6.8 Training and Inference times

As discussed previously in Chapter 2, the inference time is the time taken for any of the five neural network models to perform a prediction on the unseen test data. This time measurement is also a good indicator of how much depth, or how much complexity, a neural network architecture has. The inference time calculation was implemented using the following Python code,

```
import time
# firstly iterate through each test sample
for i in range(int(Ntest//batch_size)):
    # acquire the unseen test data sample with the .next()
                                        function
    ppg_test, BP_true = test_dataset.next()
    start_time = time.time()
    # apply the neural network model to the unseen data sample
    BP_est = model.predict(ppg_test)
    time_elapsed = time.time() - start_time
    print('Average inference time per test sample: {:.4f} (ms)'.
                                        format(1000*time_elapsed/
                                        len(ppg_test)))
```

Figure 35: Python code for finding the inference time

In addition, the training times are displayed through the Python Tensorflow console interface and an average value can be calculated as a result. Hence, these times are displayed in Table 8.

Table 8: Average Training Times and Inference Times for the five neural network architectures (average after running each configuration 3 times)

Neural network architecture	**Average Training time (seconds per epoch)**		**Average Inference Time (milliseconds per test sample)**	
	Without derivative features	**With derivative features**	**Without derivative features**	**With derivative features**
AlexNet	26.29	27.12	0.5204	
ResNet	130.49	130.12	0.5756	
ResNet-LOSO	59.04	126.27	0.5891	
LSTM	86.47	87.89	0.6043	
Transformer Encoder	49.35	52.90	0.3590	

7 Evaluation of results

In this chapter, the aims are to:

- Critically evaluate the results of the experiments in the previous chapter
- Assess whether the original objectives in Chapter 1 have been fulfilled

The ResNet with derivative architecture is shown to perform best on this dataset, as it produces the smallest Validation SBP and DBP MAE values, as shown in Table 7. In addition, both the AlexNet and ResNet-LOSO architectures perform well on the dataset, which suggests that the data is more suitable to CNN architectures, especially when extra signal features are applied as input to the neural network. This is supported by the fact that the Bi-LSTM and Transformer Encoder architectures perform worse on the dataset. The Transformer encoder architecture is shown to have the largest SBP MAE (Training and Validation) and However, this architecture also has the lowest inference time as shown in Table 8, indicating that it is the least complex model with regards to analysing the test dataset.

As a general point, it is clear that the DBP MAE in Table 7 are all noticeably lower than the SBP MAE values for all architectures. This is supported by Figures 17 and 18, as there is smaller variance in the DBP values compared to the SBP values. As a result, this means that it is easier for the architectures to learn patterns based on the DBP ground truth values. Unfortunately the AAMI standards cannot be used as a benchmark of comparison between the five architectures, as the standards requires experiments to contain at least 85 subjects [82].

In this final paragraph, it will be assessed whether the original objectives have been fulfilled. Firstly, a literature review has been successfully carried out to assess what is the most feasible implementation for cuffless blood pressure estimation, however the outcome was unexpected. As there is no clear frontrunner in terms of performance, it was necessary to test a range of feasible neural network architectures on a personalised dataset in order to see which network performs best. Secondly, a novel transformer encoder has been created in Python and although its performance is acceptable, as shown in Table 7, it is clear that further tuning of the custom parameters is required to allow for better performance. In essence, the motivation for using the transformer was that its network architecture is less complex than the other four architectures, due to its ability to process layers of input data at a time. In this regard, this is a still promising future implementation that could be integrated into future wearable devices, however further validation is required to assess whether this is the case. Therefore, the method can still be seen as feasible for future wearable technology products.

8 Conclusions and Further Work

8.1 Summary of project achievements

This project has proposed and analysed a novel Transformer Encoder neural network architecture in order to perform the cuffless estimation of blood pressure values from PPG signals. In addition, it has also compared this novel architecture against four existing implementations for cuffless BP estimation. The results have demonstrated that the transformer does have acceptable performance on the MIMIC database subset, only being slightly outperformed by the other four architectures. However, it is important to note that the transformer encoder has lower computational complexity, due to how it processes the data. As a result, there is a lot of potential to improve this novel architecture further so that it can match and even outperform the existing CNN and RNN architectures.

Throughout the course of this FYP, several important design choices were made in order to implement this novel architecture. A large amount of these decisions were explained and verified through the findings of the literature review. These choices included the database used, only using PPG signals, the preprocessing stages applied to the PPG signals, the segmentation window length, the decision to investigate the differences between automated feature extraction and handpicked features and crucially the decision to propose the novel transformer encoder architecture.

The main difficulties arose in the analysis of the MAE curves produced in the Results chapter. The choice was made in the Implementation chapter to only extract the patients suffering with cardiovascular diseases and other heart-related illnesses. This decision was made to address the tradeoff between minimising the training time of the transformer whilst also aiming to maximise the estimation accuracy. In addition, the initial intention was to use the Association for the Advancement of Medical Instrumentation (AAMI) standards as a marker of performance. However these standards can only be verified if the study uses 85 patients or more, which would not even be possible with the full MIMIC-I database. Regardless of this, it is clear that further data is now required to train the transformer encoder model. Ideally, the whole Physionet MIMIC database should now be used, as this includes patients suffering from a wider range of diseases, including respiratory failure and sepsis, but in order to be able to check performance against the AAMI standards, more time should be spent in choosing another online database with sufficient data for PPG and ABP signals.

8.2 Future work

Although this novel transformer encoder architecture appears to have acceptable performance on the MIMIC subset database, it is clear that there are several opportunities for improvement in future work.

Firstly, there are several hyperparameters required to set up the transformer encoder architecture. For the purposes of this project, the chosen hyperparameters were chosen to minimise training data whilst still achieving an acceptable performance in cuffless blood pressure estimation. In order to achieve optimal performance, it is essential that further testing is carried

out to assess the tradeoff between the hyperparameter values, training time and estimation performance.

Secondly, due to limitations in the quality of the signal recordings, only 2 physiological features of the PPG signal were extracted and used as input to the neural network architectures (alongside the PPG windowed segment). These were the first and second order derivatives of the PPG signal with respect to time. It has been evidently shown in the Results chapter that the addition of these features did not have any significant effect on the estimation performance of the transformer encoder. However, it has been displayed in the literature review in Chapter 2 that there are a wide variety of features that have been previously used in both implementations involving CNNs and RNNs. These include other physiological features, such as the cardiac period, systolic and diastolic widths at different amplitude heights. However, other arbitrary features relating to the patient, such as age and heart rate, have also been used in other previous experiments.

Finally, it is clear that a lot of the findings displayed in this report do not provide any suggestions as to how the transformer encoder solution can be implemented into wearable technologies. The main reason for this was discovered during the Literature Review in Chapter 2, as there was very little discussion about how neural network based implementations are feasible. There were only comments made on the complexity of the models used, and how this may affect the computational power requirements in future wearable devices. As a result, the aim was to implement the Transformer Encoder, a model that was motivated by the paper *Attention Is All You Need* [35], a class of neural network which aims to maximise estimation accuracy whilst minimising the computational complexity of the calculations involved. However, during the Literature Review in Chapter 2, a paper, titled *Cuff-Less Blood Pressure Estimation From Photoplethysmography via Visibility Graph and Transfer Learning* [59], was found to implement a completely novel technique for cuffless blood pressure estimation using PPG signals through Visibility Graphs and Transfer Learning, which is demonstrated to minimise computational complexity in the neural network architecture but to also implement a solution that is potentially feasible for future wearable technologies. As stated previously, this method was not considered feasible for this FYP, due to the lack of papers supporting this method.

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, 10).

[3] 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).

[4] 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).

[5] 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).

[6] 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).

[7] 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).

[8] Arijit Bhattacharyya. *Final Year Project Gantt Chart*. https://github.com/ab10918/FinalYearProject---BP-from-ECG-PPG/blob/main/Gantt_FYP.pdf. Jan. 2022 (cit. on p. 1).

[9] 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, 10, 20).

[10] 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).

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

[12] 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, 6).

[13] 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).

[14] 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. 4, 6, 7, 9, 11, 20).

[15] 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).

[16] Difference Between. *Difference Between Systolic and Diastolic*. Jan. 2021 (cit. on p. 5).

[17] Fred Shaffer and J. P. Ginsberg. “An Overview of Heart Rate Variability Metrics and Norms”. In: *Frontiers in Public Health* 5 (Sept. 2017). ISSN: 22962565. DOI: 10.3389/fpubh.2017.00258 (cit. on p. 6).

[18] 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. 6–11, 16).

[19] 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).

[20] 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. 6–8).

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

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

[23] 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).

[24] 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).

[25] 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, 10).

[26] 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. 10, 11, 20).

[27] 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. 11).

[28] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. 2017 (cit. on pp. 11–13).

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. URL: http://code.google.com/p/cuda-convnet/ (cit. on pp. 12, 32).

[30] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 12, 34).

[31] Luštrek M. Slapničar G Mlakar N. "Blood Pressure Estimation from Photoplethysmogram Using a Spectro-Temporal Deep Neural Network". In: (2019). DOI: 10.3390/s19153420 (cit. on pp. 12, 20, 21, 35, 36).

[32] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. May 2015 (cit. on p. 12).

[33] Colah. *Understanding LSTM Networks*. Aug. 2015 (cit. on p. 13).

[34] C. El-Hajj and P. A. Kyriacou. "Deep learning models for cuffless blood pressure monitoring from PPG signals using attention mechanism". In: *Biomedical Signal Processing and Control* 65 (Mar. 2021). ISSN: 17468108. DOI: 10.1016/j.bspc.2020.102301 (cit. on pp. 13, 20, 21).

[35] 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 pp. 14, 56).

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

[37] 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 pp. 17, 18).

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

[39] 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 pp. 19, 21).

[40] 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).

[41] 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).

[42] 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 pp. 19, 21).

[43] 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 pp. 19, 21).

[44] Ananda Chatterjee, Madhuchhanda Mitra, and Saurabh Pal. “Cuffless Systolic Blood Pressure Estimation Using Photoplethysmography Signal”. In: (2019), pp. 424–427 (cit. on p. 19).

[45] M. Lubin, D. Vray, and S. Bonnet. “Blood pressure measurement by coupling an external pressure and photo-plethysmographic signals”. In: (2020), pp. 4996–4999. DOI: 10.1109/EMBC44109.2020.9176730 (cit. on p. 19).

[46] A. G. Pielmuş et al. “Spectral Parametrization of PPG, IPG and pAT Pulse Waves for Continuous Noninvasive Blood Pressure Estimation”. In: (2019), pp. 4673–4676. DOI: 10.1109/EMBC.2019.8857697 (cit. on p. 19).

[47] 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).

[48] 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).

[49] 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).

[50] 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).

[51] Ümit Şentürk, İbrahim Yücedağ, and Kemal Polat. “Cuff-less continuous blood pressure estimation from Electrocardiogram(ECG) and Photoplethysmography (PPG) signals with artificial neural network”. In: (2018), pp. 1–4. DOI: 10.1109/SIU.2018.8404255 (cit. on p. 20).

[52] Rajdeep Kumar Nath, Himanshu Thapliyal, and Allison Caban-Holt. "Towards Photoplethysmogram Based Non-Invasive Blood Pressure Classification". In: (2018), pp. 37–39. DOI: 10.1109/iSES.2018.00018 (cit. on pp. 20, 30).

[53] Sumbal Maqsood et al. "A Benchmark Study of Machine Learning for Analysis of Signal Feature Extraction Techniques for Blood Pressure Estimation Using Photoplethysmography (PPG)". In: *IEEE Access* 9 (2021), pp. 138817–138833. DOI: 10.1109/ACCESS.2021.3117969 (cit. on p. 20).

[54] Peihao Li and Taous-Meriem Laleg-Kirati. "Central Blood Pressure Estimation From Distal PPG Measurement Using Semiclassical Signal Analysis Features". In: *IEEE Access* 9 (2021), pp. 44963–44973. DOI: 10.1109/ACCESS.2021.3065576 (cit. on p. 20).

[55] Shota Shimazaki et al. "Features Extraction for Cuffless Blood Pressure Estimation by Autoencoder from Photoplethysmography". In: (2018), pp. 2857–2860. DOI: 10.1109/EMBC.2018.8512829 (cit. on p. 20).

[56] Ümit Şentürk, Ibrahim Yücedağ, and Kemal Polat. "Repetitive neural network (RNN) based blood pressure estimation using PPG and ECG signals". In: (2018), pp. 1–4. DOI: 10.1109/ISMSIT.2018.8567071 (cit. on p. 20).

[57] Qingsong Xie et al. "Machine Learning Methods for Real-Time Blood Pressure Measurement Based on Photoplethysmography". In: (2018), pp. 1–5. DOI: 10.1109/ICDSP.2018.8631690 (cit. on p. 20).

[58] Peihao Li and Taous-Meriem Laleg-Kirati. "Schrödinger Spectrum Based PPG Features for the Estimation of the Arterial Blood Pressure". In: (2020), pp. 2683–2686. DOI: 10.1109/EMBC44109.2020.9176849 (cit. on p. 20).

[59] Weinan Wang et al. "Cuff-Less Blood Pressure Estimation From Photoplethysmography via Visibility Graph and Transfer Learning". In: *IEEE Journal of Biomedical and Health Informatics* 26.5 (2022), pp. 2075–2085. DOI: 10.1109/JBHI.2021.3128383 (cit. on pp. 20, 21, 56).

[60] Mohammad Kachuee et al. "Cuffless Blood Pressure Estimation Algorithms for Continuous Health-Care Monitoring". In: *IEEE Transactions on Biomedical Engineering* 64.4 (2017), pp. 859–869. DOI: 10.1109/TBME.2016.2580904 (cit. on p. 20).

[61] Muskan Singla, Syed Azeemuddin, and Prasad Sistla. "Learning-Based Model for Central Blood Pressure Estimation using Feature Extracted from ECG and PPG signals". In: (2020), pp. 855–858. DOI: 10.1109/EMBC44109.2020.9176593 (cit. on p. 20).

[62] Sertaç Kılıçkaya, Aytuğ Güner, and Barış Dal. "Comparison of Different Machine Learning Techniques for the Cuffless Estimation of Blood Pressure using PPG Signals". In: (2020), pp. 1–6. DOI: 10.1109/HORA49412.2020.9152602 (cit. on p. 20).

[63] Jaehyo Jung et al. "Development of Semi-Supervised Learning-Based Continuous Blood Pressure Estimation System". In: (2019), pp. 920–923. DOI: 10.1109/CSCI49370.2019.00175 (cit. on p. 20).

[64] Chadi El Hajj and Panayiotis A. Kyriacou. "Cuffless and Continuous Blood Pressure Estimation From PPG Signals Using Recurrent Neural Networks". In: (2020), pp. 4269–4272. DOI: 10.1109/EMBC44109.2020.9175699 (cit. on p. 20).

[65] Shota Shimazaki et al. "Cuffless Blood Pressure Estimation from only the Waveform of Photoplethysmography using CNN". In: (2019), pp. 5042–5045. DOI: 10.1109/EMBC.2019.8856706 (cit. on p. 20).

[66] Cong Yan et al. "Novel Deep Convolutional Neural Network for Cuff-less Blood Pressure Measurement Using ECG and PPG Signals". In: (2019), pp. 1917–1920. DOI: 10.1109/EMBC.2019.8857108 (cit. on p. 20).

[67] Sanghyun Baek, Jiyong Jang, and Sungroh Yoon. "End-to-end blood pressure prediction via fully convolutional networks". In: *IEEE Access* 7 (2019), pp. 185458–185468. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2960844 (cit. on p. 20).

[68] Heesang Eom et al. "End-to-end deep learning architecture for continuous blood pressure estimation using attention mechanism". In: *Sensors (Switzerland)* 20 (8 Apr. 2020). ISSN: 14248220. DOI: 10.3390/s20082338 (cit. on p. 20).

[69] 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. 24).

[70] Ahmad Malik; Daniel Brito; Sarosh Vaqar; Lovely Chhabra. "Congestive Heart Failure". In: *National Library of Medicine* (2022). URL: https://www.ncbi.nlm.nih.gov/books/NBK430873/ (cit. on p. 24).

[71] Long B Montrief T Koyfman A. "Coronary artery bypass graft surgery complications: A review for emergency clinicians." In: *National Library of Medicine* (2018). URL: https://www.ncbi.nlm.nih.gov/books/NBK430873/ (cit. on p. 24).

[72] Hai O Kosaraju A Pendela VS. "Cardiogenic Shock". In: *National Library of Medicine* (2022). URL: https://www.ncbi.nlm.nih.gov/books/NBK482255/ (cit. on p. 24).

[73] Python. *History and License for Python*. June 2022 (cit. on p. 24).

[74] Python. *Numpy documentation*. June 2022 (cit. on p. 24).

[75] Python. *Pandas documentation*. June 2022 (cit. on p. 24).

[76] Python. *Tensorflow documentation*. June 2022 (cit. on p. 24).

[77] Python. *Keras documentation*. June 2022 (cit. on p. 24).

[78] Python. *Scikit-Learn documentation*. June 2022 (cit. on p. 24).

[79] Python. *Heartpy documentation*. June 2022 (cit. on p. 24).

[80] Python. *WFDB-Python documentation*. June 2022 (cit. on p. 24).

[81] Python. *Matplotlib documentation*. June 2022 (cit. on p. 24).

[82] Stergiou G S. "A Universal Standard for the Validation of Blood Pressure Measuring Devices: Association for the Advancement of Medical Instrumentation/European Society of Hypertension/International Organization for Standardization (AAMI/ESH/ISO) Collaboration Statement". In: *National Library of Medicine* (2018) (cit. on pp. 26, 54).

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.*'

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)
40          X = Activation(ReLU())(X)
41          X = BatchNormalization(axis=-1, name='BatchNorm4')(X)
42
43          X = Conv1D(256, kernel_size=kernel_size, strides=1,
                                        dilation_rate=dil, name='
                                        conv5', kernel_initializer=
                                        glorot_uniform(seed=0),
                                        padding="same")(X)
44          if useMaxPooling:
45              X = MaxPooling1D(3, strides=2, name="MaxPool5")(X)
46          X = Activation(ReLU())(X)
47          X = BatchNormalization(axis=-1, name='BatchNorm5')(X)
48
49          # Fully connected stage
50          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):

```

```
10      # Implementation of the identity block as defined in Figure 3
11
12      # Arguments:
13      # X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
14      # f -- integer, specifying the shape of the middle CONV's window
                                          for the main path
15      # filters -- python list of integers, defining the number of
                                          filters in the CONV layers of
                                          the main path
16      # stage -- integer, used to name the layers, depending on their
                                          position in the network
17      # block -- string/character, used to name the layers, depending on
                                           their position in the network
18
19      #  Returns:
20      # X -- output of the identity block, tensor of shape (n_H, n_W,
                                          n_C)
21
22      # defining name basis
23      conv_name_base = 'res' + str(stage) + block + '_branch'
24      bn_name_base = 'bn' + str(stage) + block + '_branch'
25
26      # Retrieve Filters
27      F1, F2, F3 = filters
28
29      # Save the input value. You'll need this later to add back to the
                                          main path.
30      X_shortcut = X
31
32      # First component of main path
33      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)
138     X = Activation('relu')(X)
139     X = MaxPooling1D(3, strides=3)(X)
140
141     # Stage 2
142     X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2,
                                            block='a', s=1)
143     X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
144     X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')
145
146     # Stage 3
147     X = convolutional_block(X, f = 3, filters = [128, 128, 512], stage
                                             = 3, block='a', s = 2)
148     X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
149     X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
150     X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')
151
152     # 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]]))

    return dt

def mid_spectrogram_layer(input_x):
    l2_lambda = .001
    n_dft = 128
    n_hop = 64
    fmin = 0.0
    fmax = 50 / 2

    x = Permute((2, 1))(input_x)
    # x = input_x
    x = STFT(n_fft=n_dft, hop_length=n_hop, output_data_format='
                                        channels_last')(x)
    x = Magnitude()(x)
    #x = MagnitudeToDecibel()(x)
    #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)
```

```
        X_dt2 = Lambda(diff, arguments={'fs': fs})(X_dt1)
        X = [X_input, X_dt1, X_dt2]
    else:
        X = [X_input]

    num_channels = len(X)

    for i in np.arange(num_channels):
        channel_resnet_input, channel_resnet_out =
                                            single_channel_resnet(X[i],
                                             num_filters=num_filters,
        num_res_blocks=4, cnn_per_res=3, kernel_sizes=[8, 5, 5, 3],
                                            max_filters=64, pool_size=2
                                            , pool_stride_size=1)
        channel_outputs.append(channel_resnet_out)
        inputs.append(channel_resnet_input)

    spectral_outputs = []
    num_filters = 32
    for x in inputs:
        spectro_x = mid_spectrogram_LSTM_layer(x)
        spectral_outputs.append(spectro_x)

    # concatenate the channel specific residual layers

    # print("Num Channels: ", num_channels)

    if num_channels > 1:
        x = concatenate(channel_outputs, axis=-1)
    else:
        x = channel_outputs[0]

    x = BatchNormalization()(x)
    x = GRU(65)(x)
    # x = Flatten()(x)
    x = BatchNormalization()(x)

    # join time-domain and frequnecy domain fully-conencted layers
    if num_channels > 1:
        s = concatenate(spectral_outputs, axis=-1)
    else:
        s = spectral_outputs[0]

    # s = Flatten()(s)
    #     x = Dense(128,activation="relu",kernel_regularizer=l2(
                                        l2_lambda)) (x)
    s = BatchNormalization()(s)
    # LETS DO OVERFIT
    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

```
1     from tensorflow.keras.layers import Input, LSTM, Dense,
                                        Bidirectional, Conv1D, ReLU
2
3 from tensorflow.keras import Model
4
5 def define_LSTM(data_in_shape, UseDerivative=False):
6     X_input = Input(shape=data_in_shape)
7
8     fs = 125
9
10    if UseDerivative:
11        dt1 = (X_input[:,1:] - X_input[:,:-1])*fs
12        dt2 = (dt1[:,1:] - dt1[:,:-1])*fs
13
```

```
        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
```