Wrocław University of Science and Technology
Faculty of Information and Communication Technology

Field of study: **Sztuczna Inteligencja (SZT)**

MASTER THESIS

The influence of medical signals representation on machine learning models results

Tomasz Hawro

Supervisor
prof. dr hab. inż. Urszula Markowska-Kaczmar

Keywords: Medical signals, Machine Learning, Artificial Intelligence, Feature Extraction, Representations

WROCŁAW, 2023

Contents

Chapter 1

Introduction

1.1. Motivation

Medical signal processing and analysis is an extremely important area of science. This is mainly because such data is linked to health issues. Conclusions going from that analysis often contribute to increasing the effectiveness of medical diagnoses and improving the quality of treatment. Artificial Intelligence (AI) has long been used for tasks related to medical data. Areas where AI finds applications include image segmentation and classification, anomaly detection, and time series regression and classification. The area of interest in this work is the latter, namely time series in the medical domain, further understood as medical signals. Many research works deal with specific types of medical signals, such as electrocardiography (ECG) signals [51], [12], [71], [50], photoplethysmography (PPG) [20], [54], [4], [61], [29], [2], [55], [39] or electroencephalography (EEG) [38], [41], [33], [60]. These works usually discuss specialized approaches, such as feature extraction [39], or the use of raw signal [2]. The former often includes only analysis of specific groups of features and the latter operates only on signals understood as samples over time, without considering expert features. However, there is a lack of work in which there is a comparison of different representations of medical signals along with the use of different machine learning (ML) models. The main motivation of this work is to fill this gap. An analysis of this type could show whether there is a dependence between the quality of the results obtained and the representation of medical signals for different ML models and tasks. Conclusions from such an analysis would facilitate the selection of the type of medical signals representation depending on the models for which the signals will be used.

1.2. Purpose

The purpose of this study is to analyze the impact of medical signal representation on the quality of the results of ML models. Task-dependent quality measures (classification or regression) will be evaluated. The model's computational complexity and disk space needs will also be included in the comparison.

1.3. Scope

Achieving the stated goal was made possible by following several smaller steps:

1. literature review, which includes an analysis of research papers related to medical signals, namely feature extraction and the use of ML models for related tasks,
2. finding relevant datasets so that the analyzed approaches can be compared,
3. implementation of a python package for signals analysis, representations extraction, and models training and evaluation,
4. extraction of representations and training data preparation,
5. hyper-parameter optimization,
6. ML models training and evaluation,
7. analysis of the obtained results,
8. formulation of conclusions.

1.4. Guide

Chapter 2 contains a description of medical signals and their processing methods. Both processing and feature extraction of the analyzed medical signals are described there. Medical signals representations description is placed in chapter 3. Chapter 4 contains the theoretical basis of the ML models used in the work. Chapter 5 presents a literature review presenting related works. The next three chapters present a description of the problem under analysis (chapter 6), the research procedure that was used to solve it (chapter 7), and datasets, which served as a source of data for experiments (chapter 8). Experiments, which allowed to make final conclusions are described in the chapter 9. The last chapter (10) contains a summary, which presents the final conclusions and opportunities to develop the work.

Chapter 2

Medical signals

Medical signals are the source of information about a patient's health and physical or physiological condition. There is a variety of medical signals, e.g. the ones used to monitor vital signs, neurophysiological signals, or chemical signals. In this work, the main focus is put on the following medical signals:

- electrocardiogram (ECG) - records electrical activity of the heart,
- photopletysmogram (PPG) - records changes in blood volume in the blood vessels,
- electroencephalogram (EEG) - records electrical activity of the brain,
- electrooculogram (EOG) - records electrical activity of the eye.

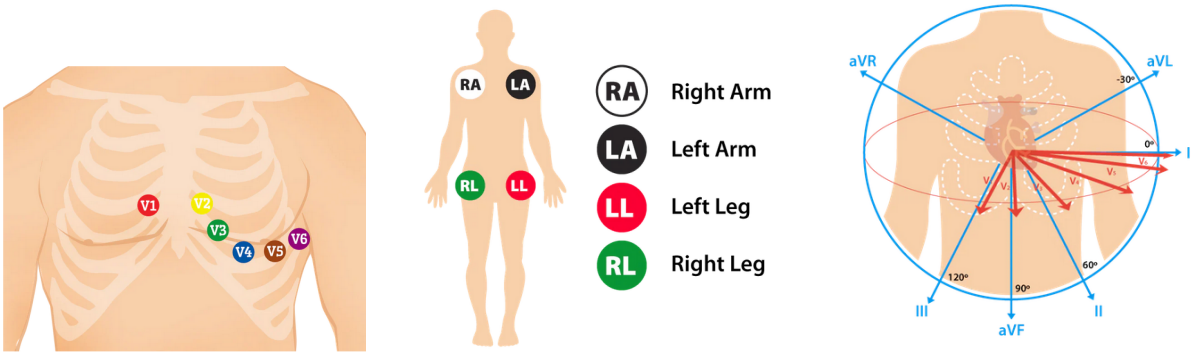
The choice of signals is due to the diversity of medical signal types and their characteristics. The ECG and PPG signals are periodic in the sense that they depend on the heart rate, thanks to which they can be segmented into single episodes occurring at (almost) constant intervals. A single episode is a segment of a signal that repeats every (almost) constant period (resulting, for example, from the heart rhythm) and its successive occurrences share common morphological characteristics. Another reason behind the choice of these signals was the availability of datasets for each of them. Recording of medical signals can be done with a variety of sensors. The data collected from these sensors are processed and analyzed to extract meaningful information about the patient's condition. One of the forms of that kind of information is features that can be extracted from the medical signal using specialized algorithms. In Section 2.1, each of the medical signals used in this work is described in terms of the signals source and its processing. Section 2.2 describes the tabular features that can be extracted from these signals.

2.1. The medical signals in detail

Medical signals differ in variety of aspects. Each of the signals has its specific properties, which will be described in the following sections. Sections 2.1.1 and 2.1.2 describe ECG and PPG signals. In the next Section (2.1.3), there is the EEG signal description, and finally in Section 2.1.4, the EOG signal is described.

2.1.1. ECG

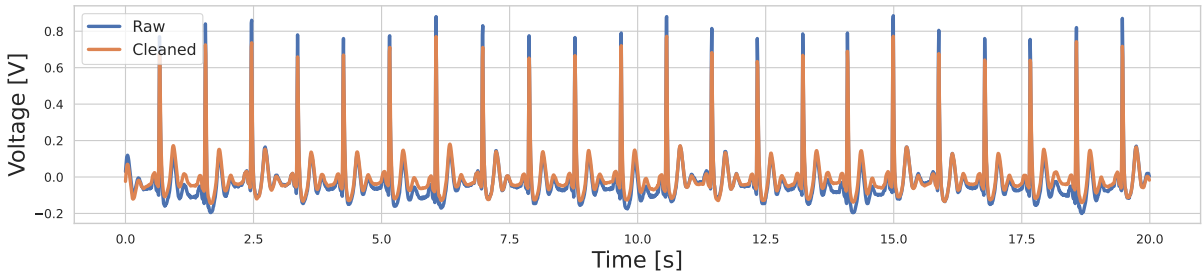
An electrocardiography (ECG) signal shows the electrical activity of the heart. That kind of information is useful for heart-related conditions diagnosis, such as arrhythmias. The source of the ECG signal is electrodes attached to the skin on the chest, arms, and legs (Fig. 2.1).



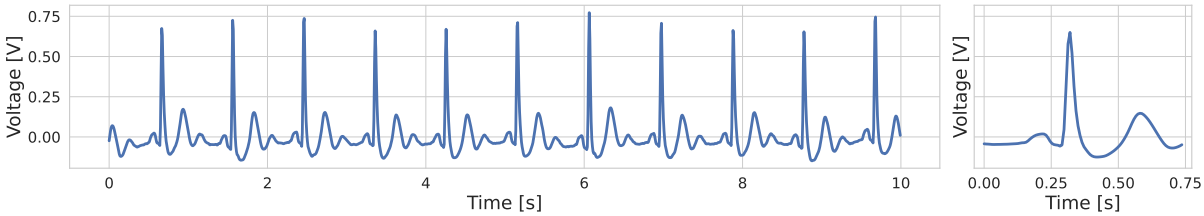
(a) Chest electrodes placement (b) Limbs electrodes placement (c) 12 ECG leads

Figure 2.1: ECG electrodes placement and leads (source: [7])

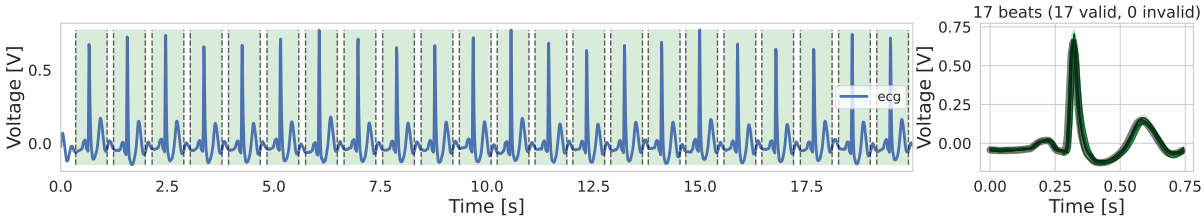
These electrodes detect the electrical signals produced by the heart. The ECG signal is composed of many repeating episodes (Fig. 2.2b). The single ECG episode (single beat) is composed of several segments, each of which represents a different phase of the heart's electrical activity. The ECG signal provides information about the heart's rate (HR) and rhythm. Abnormalities occurring in the ECG signal may indicate problems with the heart's electrical activity, such as arrhythmias, coronary artery disease, or heart attacks. ECG signal provides useful information used to diagnose and manage heart conditions [17]. Fig. 2.2b (blue) shows ECG signal waveform. However, the raw signal is often noisy. It is caused by the noise which comes from the power supply and nearby muscles (other than the heart). This brings the need for signal cleaning (filtering). Fig. 2.2a shows an example of ECG signal before (blue) and after (orange) the filtering process. Since ECG is a periodic (in terms of heart beats) signal, it is possible to extract ECG episodes and aggregate those episodes into single ECG beat (Fig. 2.2c). The episodes aggregation process involves resampling them to a fixed number of samples and averaging over time for each region and channel separately. Thus, for an example data sample consisting of R regions and C channels, divided into B individual episodes, resampled to 100 samples each, aggregation will consist of reducing the dimensionality of the representation from [B, 100, R, C] to [100, R, C].



(a) Single ECG signal cleaning process. The cleaning process includes signal detrending and denoising.



(b) Single ECG cleaned signal (left) and its aggregated single beat (right). Aggregated beat is the result of averaging all beats extracted from the clean signal.



(c) Single ECG signal beats segmentation (left) and aggregation (right). Aggregated beat is the result of averaging all beats extracted from the clean signal

Figure 2.2: Single ECG signal processing

2.1.2. PPG

The photoplethysmography (PPG) signal comes from a PPG sensor that is made of a light-emitting diode (LED) and a photodetector (PD). The sensor is used to measure changes in the amount of light absorbed by the tissue (shown in Fig. 2.3), which reflects changes in blood volume over time. The light source illuminates the tissue with light of a specified spectrum and the photodetector measures how much of that light was absorbed by the tissue. Blood inside the tissue is consistently flowing, so it changes the amount of absorbed light. The PPG signal is the result of these changes.

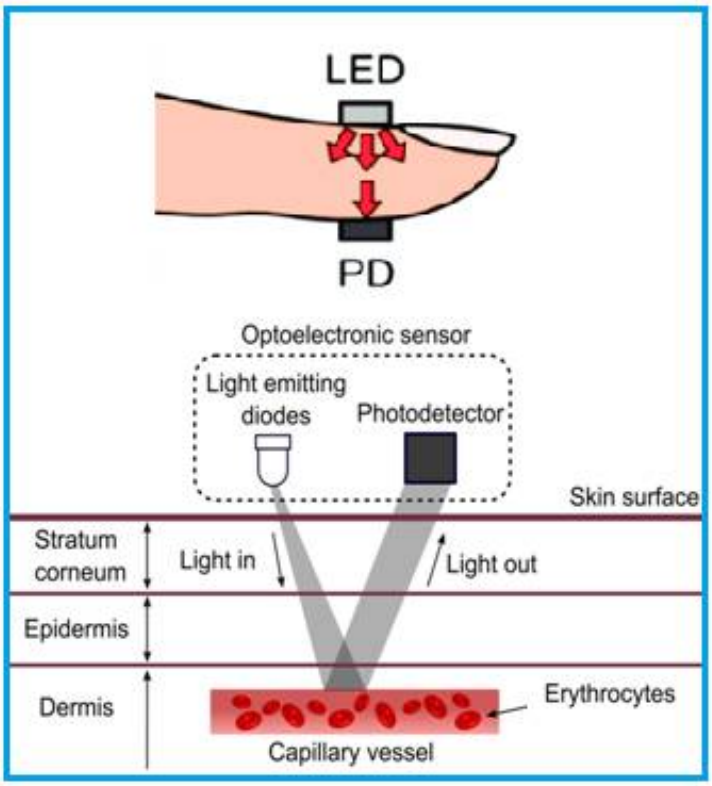
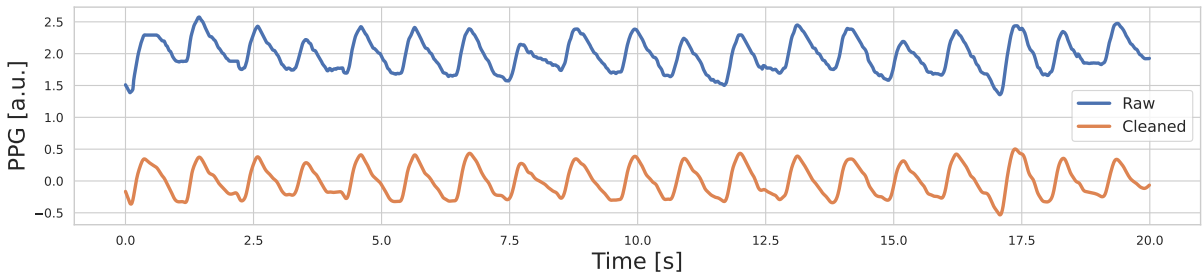
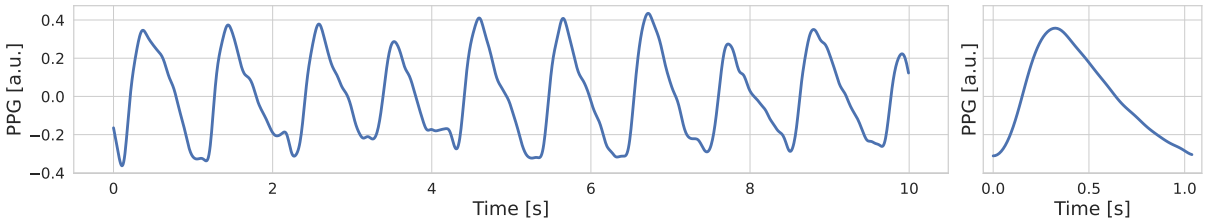


Figure 2.3: PPG sensor and how it works (source: [69])

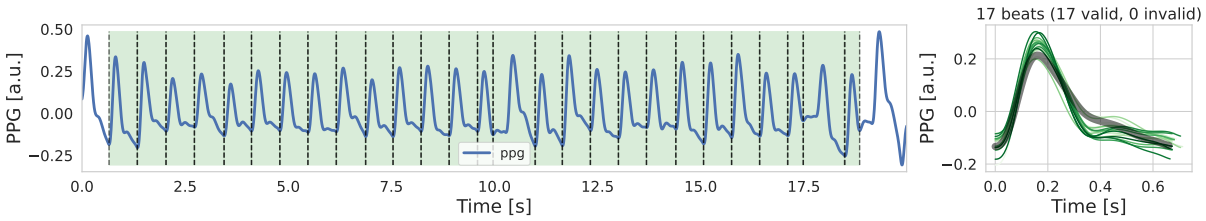
The PPG signal is composed of many repeating episodes (Fig. 2.4b). The single PPG episode typically consists of a peak and a trough. The amount of blood in the tissue is reflected by the height of the peak. The PPG signal provides useful information about many physiological parameters, including heart rate, blood pressure, and oxygen saturation [34]. Fig. 2.4b shows PPG signal waveform. However, the raw signal is often noisy. It is caused by the noise which comes from movement artifacts. This brings the need for signal filtering. Fig. 2.4a shows an example of ECG signal before and after the filtering process. Since PPG is a periodic (in terms of heartbeats) signal, it is possible to extract single PPG episodes (Fig. 2.4c).



(a) Single PPG signal cleaning process. The cleaning process includes signal detrending and denoising.



(b) Single PPG cleaned signal (left) and its aggregated single beat (right)



(c) Single PPG signal beats segmentation (left) and aggregation (right)

Figure 2.4: Single PPG signal processing

2.1.3. EEG

An electroencephalography (EEG) signal shows the electrical activity of the brain. That kind of information is useful for brain-related conditions diagnosis, such as epilepsy or sleep disorders. The source of the EEG signal is electrodes attached to the scalp (Fig. 2.5).

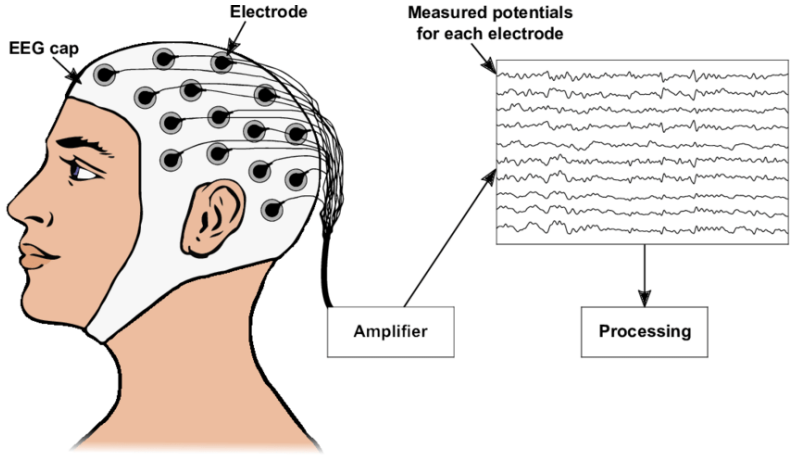


Figure 2.5: EEG electrodes placement and out coming signals (source: [43])

These electrodes detect the electrical signals produced by the brain. The EEG signal consists of many frequencies, i.e. delta, theta, alpha, and beta. Delta waves are the slowest (4Hz and below), have the highest amplitude, and are usually associated with deep sleep. Theta waves occur between 4 and 7.5 Hz frequencies and are typically associated with relaxation. Alpha waves occur between 8 and 13 Hz and are typically associated with calmness (eyes closed). Beta waves represent the fastest (of previously mentioned) activity, which is between 14 and 26 Hz, and are typically associated with alertness and concentration [53]. Fig. 2.6 shows EEG signal waveform.

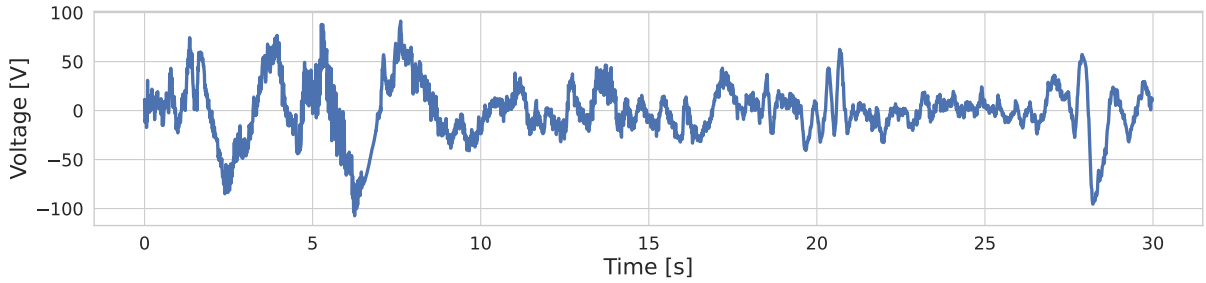


Figure 2.6: Single EEG signal

2.1.4. EOG

An electrooculography (EOG) signal shows the electrical activity of the eye. That kind of information is useful for eye movement-related conditions diagnosis. The source of the EOG signal is electrodes attached to the skin around the eyes (Fig. 2.7).

These electrodes captures electrical potential caused by eyes movement [25]. Fig. 2.8a shows EOG signal waveform. However, the raw signal is often noisy. It is caused by the noise which comes from nearby muscles (other than the eye muscles). This brings the need for signal cleaning (filtering). Fig. 2.8b shows an example of the EOG signal before and after the filtering process.

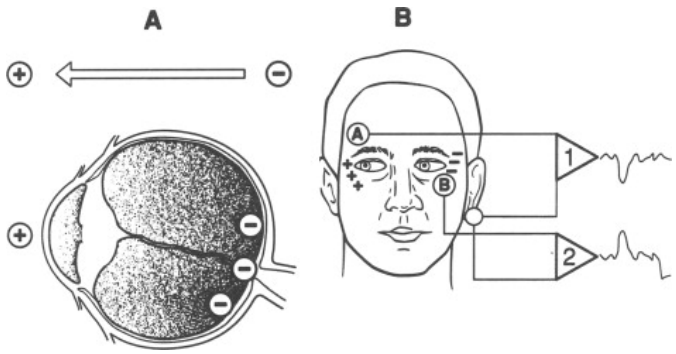
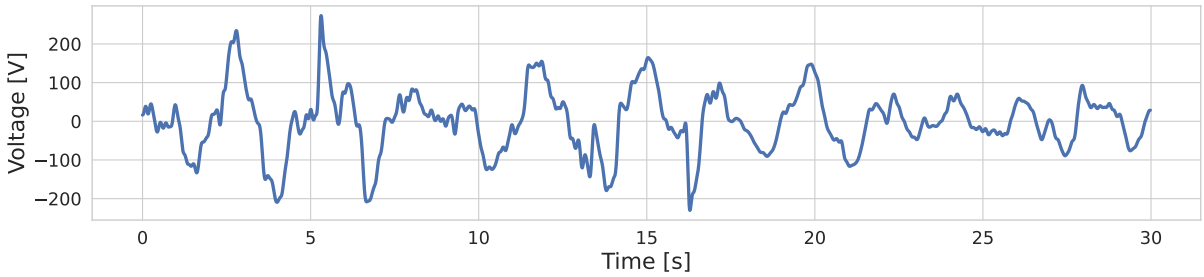
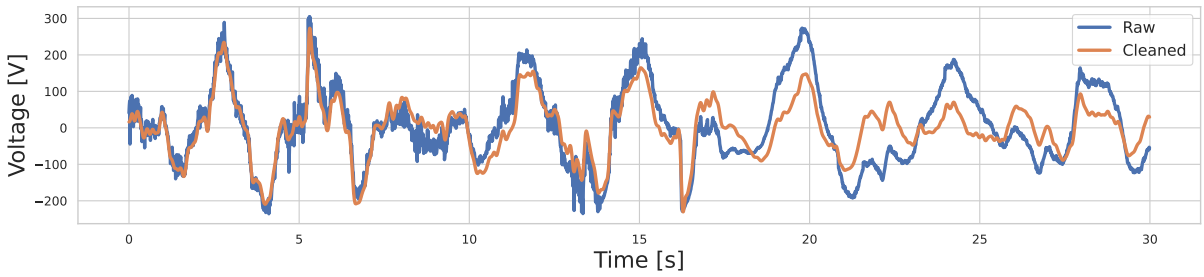


Figure 2.7: EOG electrodes placement (source: [66])



(a) Single EOG signal



(b) Single EOG signal cleaning process

Figure 2.8: Single EOG signal processing

2.2. Features extraction

There is a different type of features that can be extracted from each of the previously mentioned signals (domain-specific features), but there are also many features that can be extracted for all of the signals. The latter, that is the common features are listed below.

1. basic features - the simplest kind of features which can be divided into two groups:
   - statistic features - statistic measures of the data, i.e. mean, standard deviation (std), kurtosis, skewness, median, quantiles, entropy, energy [10], [3],
   - zero crossing features - number of times when the signal crosses zero and standard deviation of crossing intervals,
2. peaks signal features - basic features extracted for the signal of the peaks, which are calculated as maxima of the signal (in the general case) or as systolic peaks for PPG signal or R peaks for ECG signal,

3. troughs signal features - basic features extracted for the signal of the troughs, which are calculated as a minimum value between peaks,
4. amplitudes signal features - basic features extracted for the signal of the amplitudes, which are calculated as the differences between successive peaks and troughs,
5. discrete wavelet transform (DWT) features - extracted from discrete wavelet decomposition, which is a method for analyzing and decomposing a signal into its frequency components using wavelets (example of the decomposition is shown in Fig. 2.10). It involves representing the signal at different scales or resolutions and analyzing the frequency content of the signal at each scale, which is very useful for a signal with a mix of different frequency components. The DWT features consist of basic features extracted for different levels of discrete wavelet decomposition [68], [1], [45], [42]. The number of extracted DWT features depends on the length of the signal from which these features are extracted. The basic features are extracted for detail coefficients at all levels and the approximation coefficient from the last level of decomposition.

2.2.1. ECG

All the common features previously mentioned can also be extracted from an ECG signal. The example of ECG peaks, troughs, and amplitudes signals used for peaks, troughs, and amplitudes feature extraction are shown in Fig. 2.9 (left). The example of aggregated beat ECG and its features is shown in Fig. 2.9 (right). The example of ECG signal discrete wavelet decomposition is shown in Fig. 2.10.

The following features can be extracted from an ECG signal:

1. common features – described in the beginning of Section 2.2,
2. aggregated beat features – features extracted for aggregated ECG cycle [51] (shown in Fig. 2.9 right):
    1. basic features - extracted the same way as basic features group from common features,
    2. critical points features - locations and values for P, Q, R, S, and T critical points,
    3. area features - areas of P segment, QRS complex, and T segment,
    4. slope features - onset and offset slopes of P, R, and T peaks,
    5. energy features - energy of P segment, QRS complex, and T segment,
    6. intervals features - intervals between critical points
3. heart rate variability (HRV) features [15], [13].

2.2.2. PPG

Every common feature listed at the beginning of section 2.2 can also be extracted from a PPG signal. The example of PPG peaks, troughs, and amplitudes signals used for peaks, troughs, and amplitudes feature extraction are shown in Fig. 2.11 (top). The example of aggregated beat PPG and its features is shown in Fig. 2.11 (bottom). The example of PPG signal discrete wavelet decomposition is shown in Fig. 2.12.

The following features can be extracted from a PPG signal:

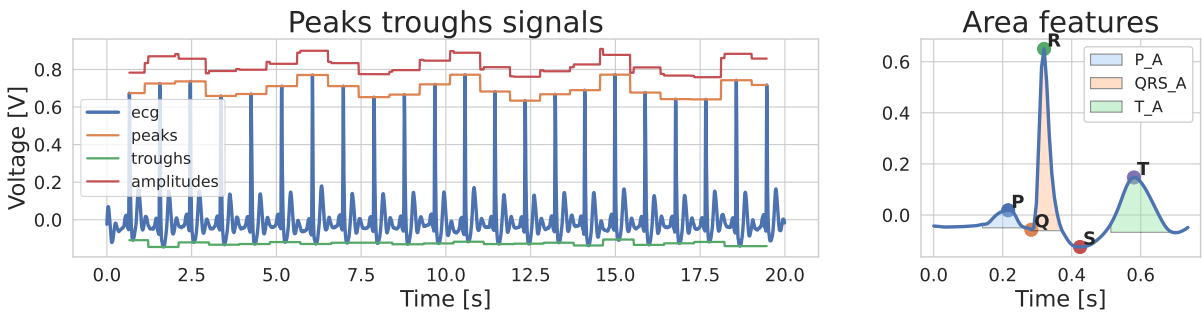


Figure 2.9: Single ECG signal features visualisation plot. Peaks, troughs and amplitudes signals (left) and area features (right).

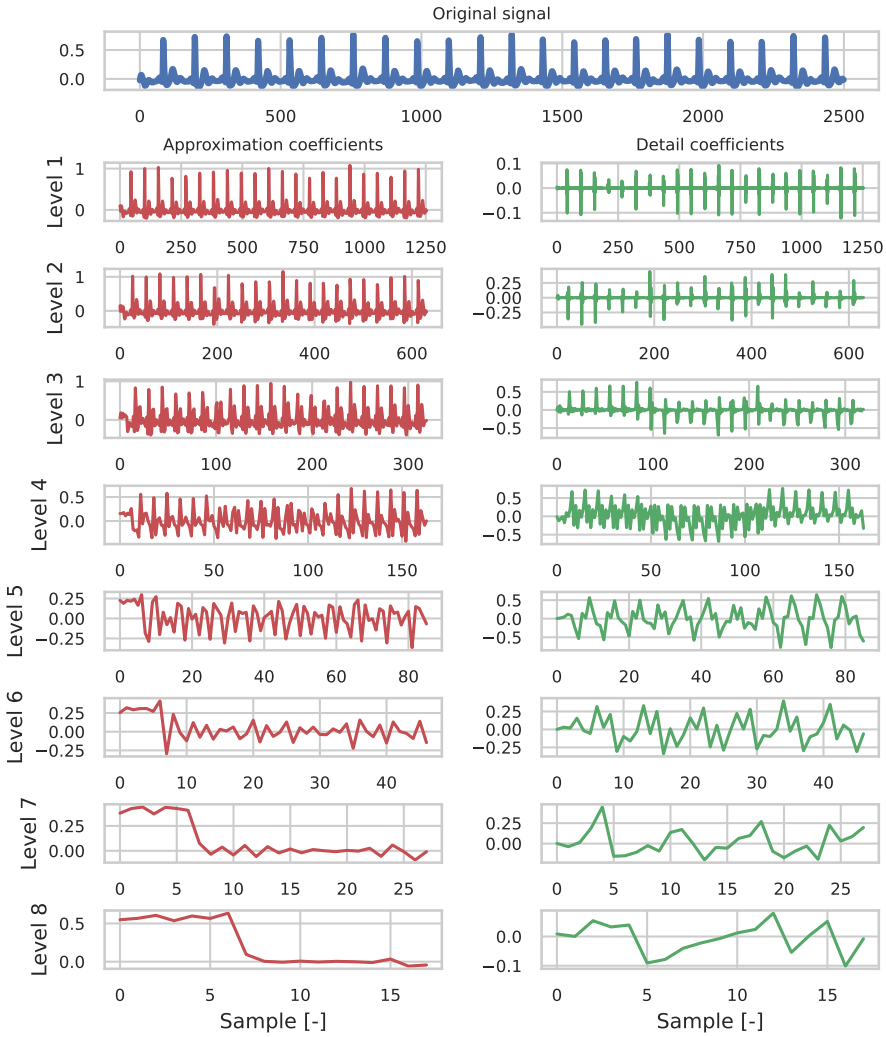


Figure 2.10: Single ECG signal discrete wavelet decomposition

1. common features – described in the beginning of Section 2.2,
2. aggregated beat features - features extracted for aggregated PPG cycle [51] (shown on Fig. 2.11 bottom):
    1. basic features - extracted the same way as basic features group from common features,
    2. critical points features - location and value for systolic peak critical point,
    3. area features - area before and after systolic peak,

4. pulse width features - intervals on different height percentages of the beat (10, 25, 33, 50, 66, 75 percentages) [39] (shown on Fig. 2.11 bottom center)
5. pulse height features - pulse values on different width percentages of the beat (10, 25, 33, 50, 66, 75 percentages) (Fig. 2.11 bottom right). This group of features was based on [39].
6. slope features - onset and offset slope of the beat,
7. energy features - energy before and after systolic peak,

3. heart rate variability (HRV) features [15], [13].

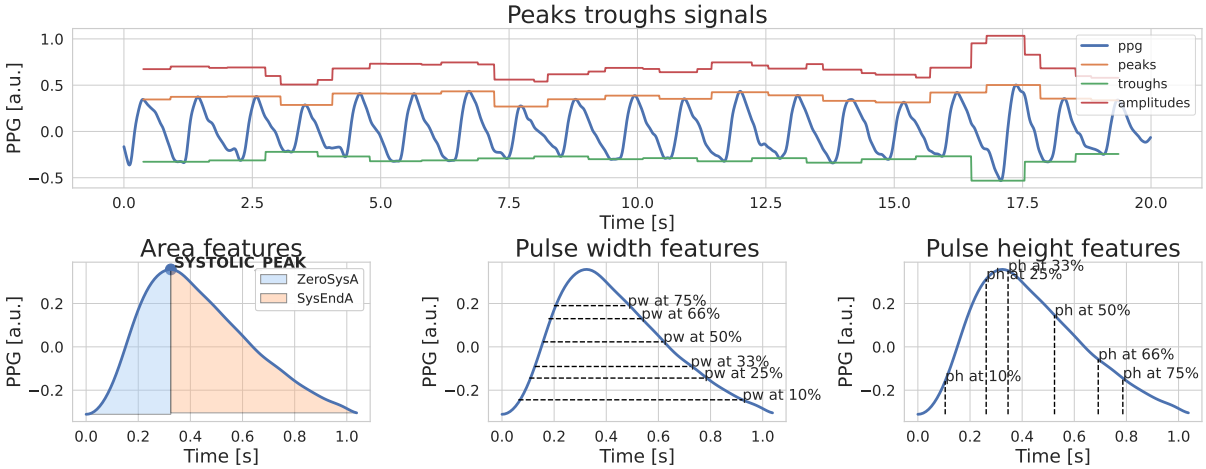


Figure 2.11: Single PPG signal features visualisation plot. Peaks, troughs and amplitudes signals (top) and aggregates beat features (bottom).

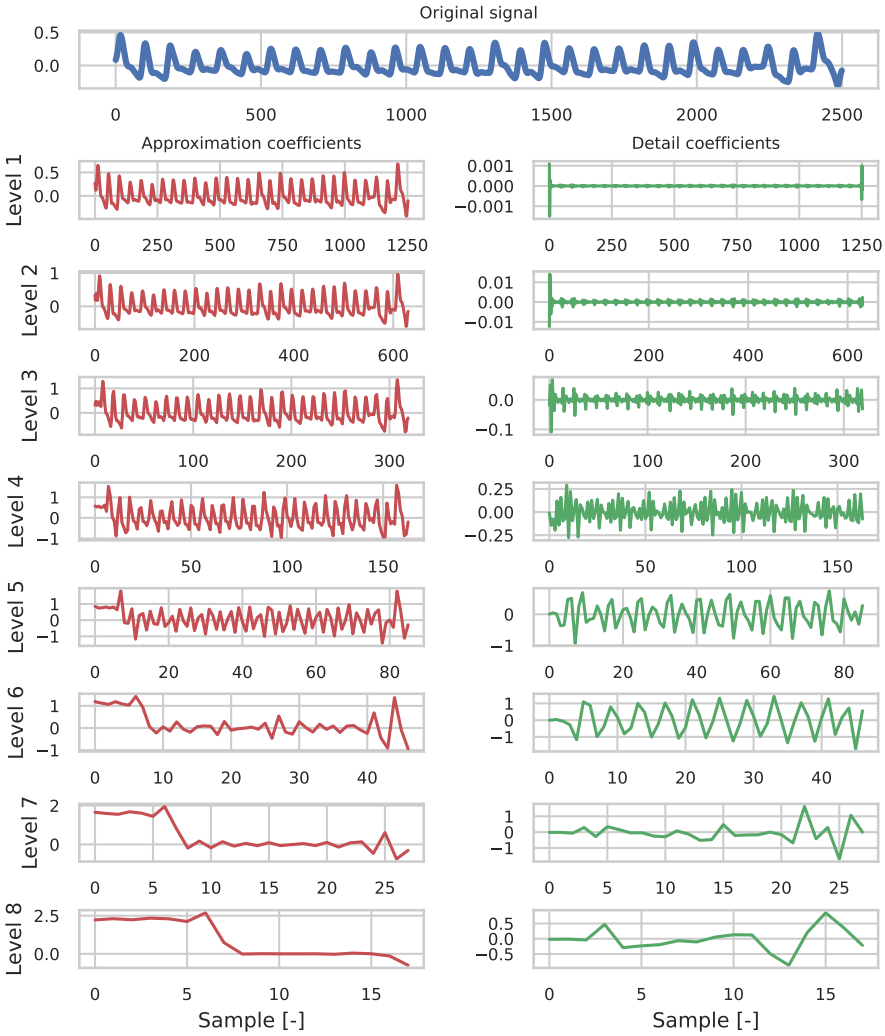


Figure 2.12: Single PPG signal discrete wavelet decomposition

2.2.3. EEG

All the common features previously mentioned can also be extracted from an EEG signal. The example of EEG peaks, troughs, and amplitudes signals used for peaks, troughs, and amplitudes feature extraction are shown in Fig. 2.13 (left). The example of EEG signal discrete wavelet decomposition is shown in Fig. 2.14. The following features are extracted from an EEG signal:

1. common features – described in the beginning of Section 2.2,
2. frequency features - mean of power spectral density for specific frequency ranges, i.e. delta, theta, alpha, sigma, and beta [57]. The example of EEG power spectral density (PSD) which is the source for this feature group is shown in Fig. 2.13 (right).

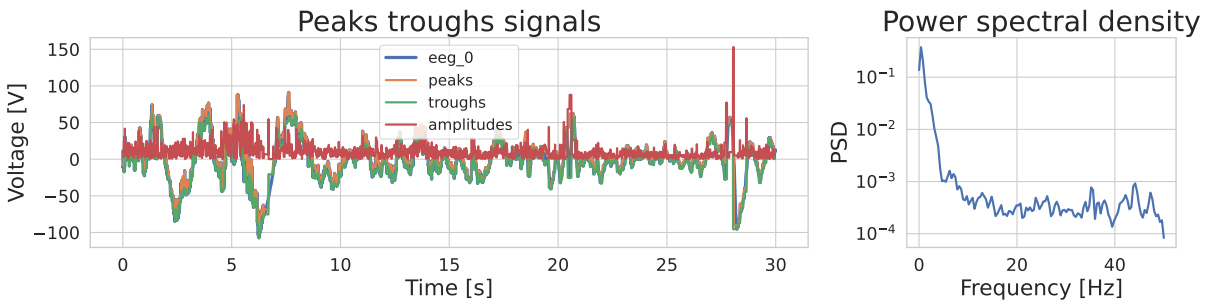


Figure 2.13: Single EEG signal features visualisation plot. Peaks, troughs and amplitudes signals (left) and power spectral density (used for frequency features) (right). The Power spectral density (PSD) describes the distribution of power over the frequency spectrum of a signal. It shows how the power of a signal is distributed across different frequencies. The PSD can be used to analyze the frequency content of the signal.

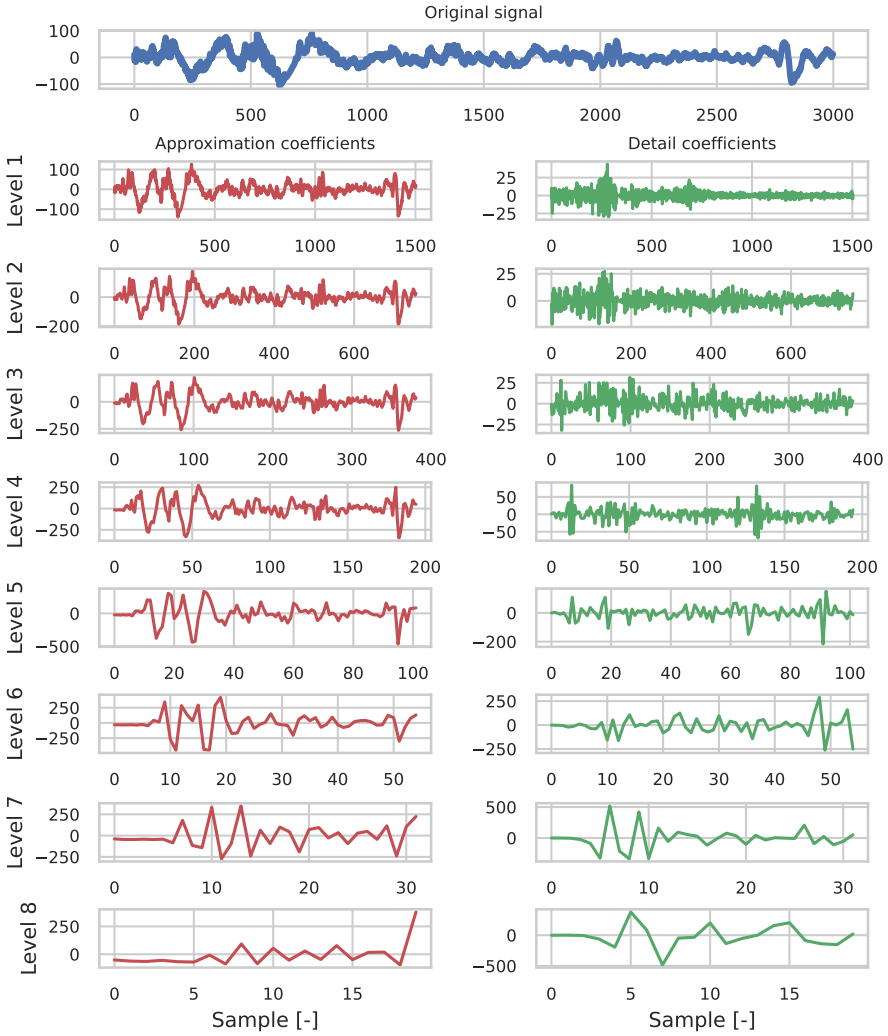


Figure 2.14: Single EEG signal discrete wavelet decomposition

2.2.4. EOG

Each of the common features presented at the beginning of section 2.2 can also be extracted from an EOG signal. The example of EOG peaks, troughs, and amplitudes signals used for peaks,

troughs, and amplitudes feature extraction are shown in Fig. 2.15. The example of EOG signal discrete wavelet decomposition is shown in Fig. 2.16. The following features can be extracted from an EOG signal:

1. common features – described at the beginning of Section 2.2,
2. eye blink features - frequency of detected eye blinks and mean and standard deviation of intervals between the eye blinks.

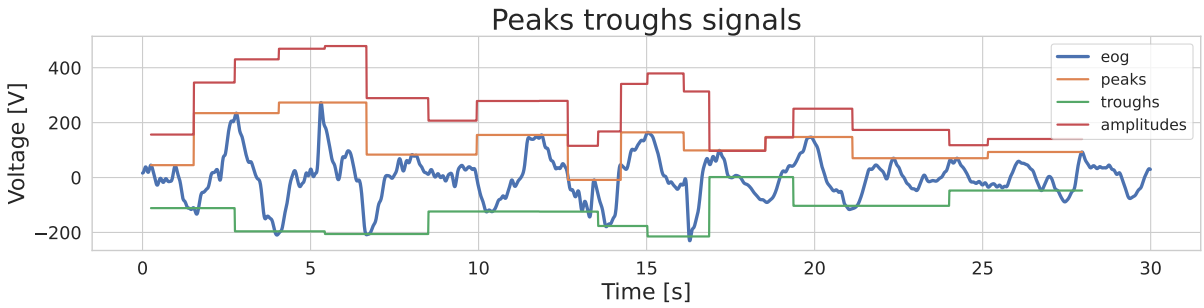


Figure 2.15: Single EOG signal features visualisation plot. Peaks, troughs and amplitudes signals.

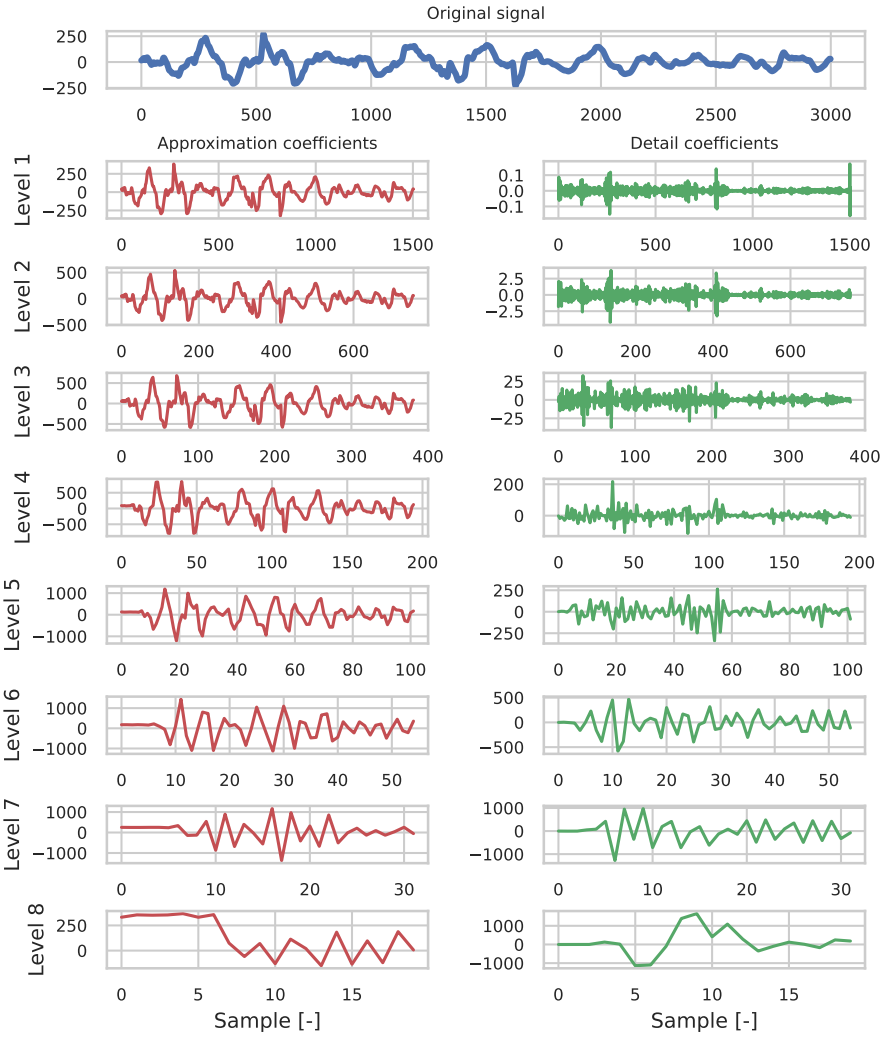


Figure 2.16: Single EOG signal discrete wavelet decomposition

Chapter 3

Medical signals representations

Medical signal in its raw form is a sequence of samples for successive time points, often sampled with constant sampling frequency (fs). Although the simple waveform provides a lot of useful information, it is often not enough. By using a raw signal form, it is possible to extract many other types of representations, such as windowed waveforms, tabular features, spectrogram or wavelet transforms. In this work, the main focus is put on waveforms and tabular feature representations, which are briefly described below. The type of representation used is often dictated by computational power and expertise knowledge trade-off. Some representations require less computational power (f.e. tabular features) but need broad expertise for a proper feature extraction process implementation. On the other hand, pure waveform representations require less domain knowledge, but much more computational power to fit the ML models. In the following sections, the representation's dimensionality shapes are described with N, R, C, B, W, and F letters: N - number of signal samples, R - number of signal regions, C - number of signal channels, B - number of beats/episodes the signal was cut into, W - number of windows the signal was cut into, F - number of features extracted from the signals.

3.1. Waveforms

The raw signal waveform is the simplest form of representation since it is the original (raw) signal derived from the medical sensors. It consists of signal samples taken for successive time points (Fig. 3.1), so it preserves temporal dependencies. That type of representation often doesn't require any processing and can be used directly as input to the ML model. Medical signals are obtained using the sensors, which are located at specific body regions (Fig. 2.1), often forming a spatial map (Fig. 2.5). The division of signals into different regions of their acquisition will be referred to as division into regions and its dimensionality will be represented by R. The regions may also refer to different medical sensors used to acquire the signals. An example of region dimensionality may be 12 different leads of ECG signals, which are reflected as 12 regions. It is possible to easily extend the waveform representation by calculating the derivatives of the signal and stacking it onto existing raw signals. The derivatives will be referred to as channels and their dimensionality will be represented by C. The dimensionality of raw signal waveform representation is $[N, R, C]$, where C may be equal to 1 if no derivatives are used.

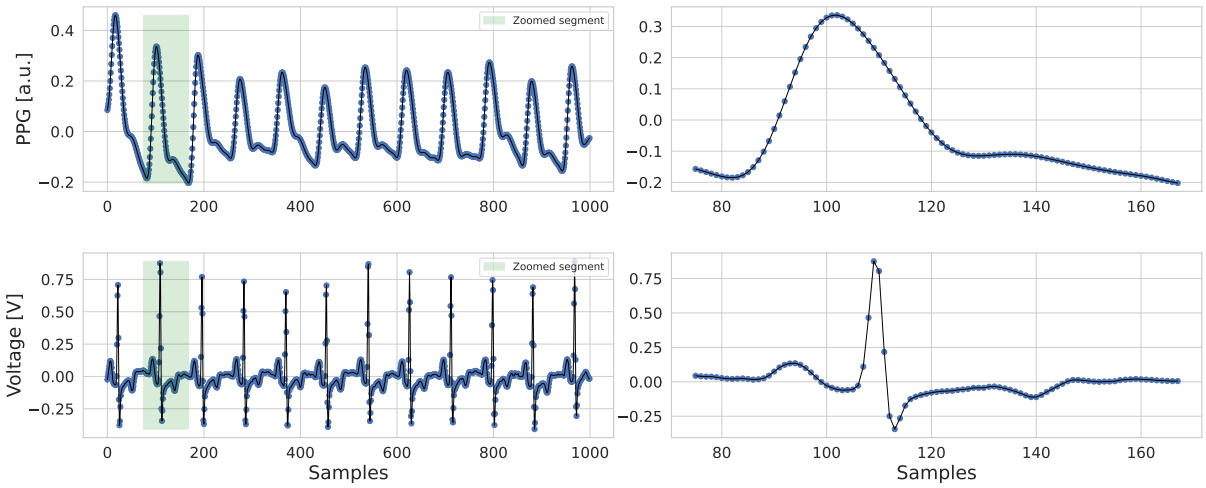


Figure 3.1: Waveform showing PPG (top) and ECG (bottom) signals with samples taken for successive time points (left) and zoomed segments (right). The figure shows an example of a measurement consisting of 1000 samples length, two regions (PPG and ECG signals) and one channel (no additional derivatives). Following the $[N, R, C]$ convention, the representation dimensionality in that case would be $[1000, 2, 1]$.

3.1.1. Whole signal waveforms

The whole signals waveforms representation consists of all available signal samples for all available regions. Due to the lack of need for expertise and external models in creating this type of representation, it is the simplest in terms of obtaining the form of the medical signal analyzed in the work. Its dimensionality is $[N, R, C]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. (optional) calculate D derivatives and stack them onto current data (shape $[N, R, 1 + D]$ or $[N, R, 1]$ if no derivatives are used),
4. (optional) apply flattening if using tabular features model (shape $[N \cdot R \cdot C, 1]$).

3.1.2. Windows waveforms

Windows waveforms representation consists of signal samples taken for windowed signal for all available regions (Fig. 3.2). Its dimensionality is $[W, N, R, C]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. (optional) calculate D derivatives and stack them onto current data (shape $[N, R, 1 + D]$ or $[N, R, 1]$ if no derivatives are used),
4. cut signal into windows using sliding window approach. Each window has win_len samples and is taken every $step$ samples (shape $[W, N, R, C]$),
5. (optional) apply flattening if using tabular features model (shape $[W \cdot N \cdot R \cdot C, 1]$).

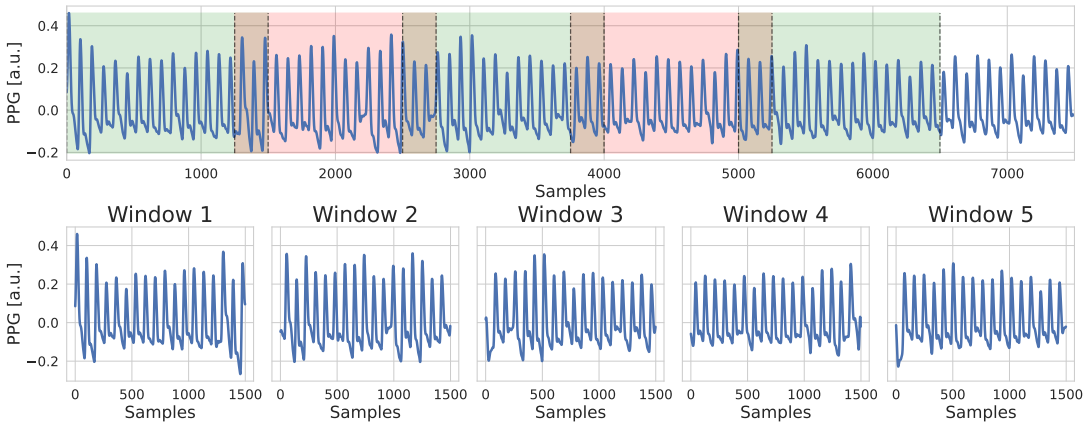


Figure 3.2: Windows waveforms representation. Waveform showing signal with windows bounds (top) and segmented windows (bottom). A measurement consists of 7500 samples length, one region (PPG signal) and one channel. The measurement was cut into windows, with 1250 samples step and length of each window is 1500 samples. Following the $[W, N, R, C]$ convention, the representation dimensionality in that case would be $[5, 1500, 1, 1]$.

3.1.3. Beats waveforms

Beats waveforms representation is possible to obtain only for periodic medical signals. It consists of signal samples taken for beats of the signal for all available regions (Fig. 3.3). Its dimensionality is $[B, N, R, C]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. find beats intervals (start and stop locations) using a signal specific algorithm,
4. (optional) calculate D derivatives and stack them onto current data (shape $[N, R, 1 + D]$ or $[N, R, 1]$ if no derivatives are used),
5. cut signal into beats using the found beats intervals (shape $[B, N, R, C]$),
6. (optional) apply flattening if using tabular features model (shape $[B \cdot N \cdot R \cdot C, 1]$).

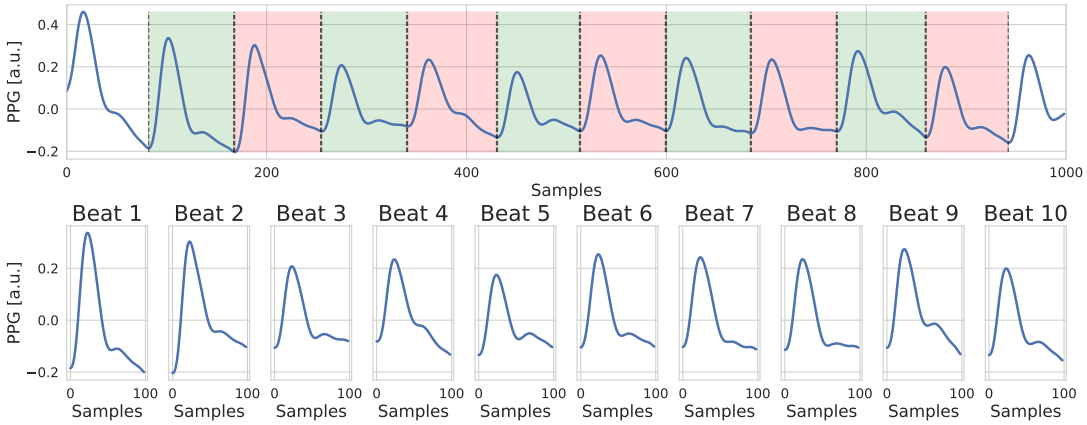


Figure 3.3: Beats waveforms representation. Waveform showing signal with beats bounds (top) and segmented beats (bottom). Measurement consists of 1000 samples length, one region (PPG signal) and one channel. The measurement was cut into 10 episodes (beats) and each of those was resampled to 100 samples. Following the $[B, N, R, C]$ convention, the representation dimensionality in that case would be $[10, 100, 1, 1]$.

3.1.4. Aggregated beat waveforms

Aggregated beat waveform representation is possible to obtain only for periodic medical signals. It consists of signal samples taken for aggregated beats of the signal for all available regions (Fig. 2.2c right, black line). It is worth noting that, at this point, no references to this type of representation have been found in the literature. Therefore, this is an approach worth investigating. Its dimensionality is $[N, R, C]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. find beats intervals (start and stop locations) using a signal specific algorithm,
4. (optional) calculate D derivatives and stack them onto current data (shape $[N, R, 1 + D]$ or $[N, R, 1]$ if no derivatives are used),
5. cut signal into beats using the found beats intervals (shape $[B, N, R, C]$),
6. aggregate (average) beats for specific region and channel into single beat (shape $[N, R, C]$),
7. (optional) apply flattening if using tabular features model (shape $[N \cdot R \cdot C, 1]$).

3.2. Features

Features representation consists of features extracted from medical signals of different waveform representations. In contrast to waveform representation, features are not always temporally dependent among themselves. Due to the high need for expertise in creating this representation, it is the most difficult in terms of obtaining form of the medical signal analyzed in this work.

3.2.1. Whole signal features

Whole signals features representation consists of features extracted for all available signals for all available regions. Fig. 2.11 (top) shows an example of features source for the whole signal and section 2.2 explains how the features may be extracted. Its dimensionality is $[F, 1]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. extract features for each region and concatenate the features (shape $[F, 1]$).

3.2.2. Windows features

Windows features representation consists of features extracted for signal windows for all available regions. The feature extraction process for windows is the same as for whole signals, the only difference is that the signals are shorter. Its dimensionality is $[W, F]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$)
3. cut signal into windows using sliding window approach. Each window has win_len_s length and is taken every step_s seconds (shape $[W, N, R]$),

4. extract region-specific features for all windows separately (shape $[W, F]$),
5. (optional) apply flattening if using tabular features model (shape $[W \cdot F]$).

3.2.3. Beats features

Only periodic medical signals can be the source of this representation type. It consists of features extracted for beats of the signal for all available regions. Fig. 2.11 (bottom) shows an example of features extracted for a single beat and Fig. 2. explains how the features may be extracted. Its dimensionality is $[B, F]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. find beats intervals (start and stop locations) using a signal specific algorithm,
4. cut signal into beats using the found beats intervals (shape $[B, N, R]$),
5. extract region-specific features for all beats separately (shape $[B, F]$),
6. (optional) apply flattening if using tabular features model (shape $[B \cdot F]$).

3.2.4. Aggregated beat features

Aggregated beat features representation is possible to obtain only for periodic medical signals. The feature extraction process for aggregated beat is the same as for beats, the only difference is that the signal is the result of averaging over beats, which often works as noise reduction. It consists of features extracted for aggregated beats of the signal for all available regions. It is worth noting that, at this point, no references to this type of representation have been found in the literature. Therefore, this is an approach worth investigating. Its dimensionality is $[F, 1]$ and the steps to obtain this type of representation are:

1. take one measurement from the dataset (shape $[N, R]$),
2. (optional) apply processing appropriate for the type of the signal (shape $[N, R]$),
3. find beats intervals (start and stop locations) using a signal specific algorithm,
4. cut signal into beats using the found beats intervals (shape $[B, N, R]$),
5. aggregate (average) beats for specific region and channel into single beat (shape $[N, R]$),
6. extract features for each region and concatenate the features (shape $[F, 1]$).

3.3. Continuous wavelet transform image

The Continuous Wavelet Transform (CWT) is a method used to analyze signals in the time-frequency domain. It involves the convolution of a signal with a set of wavelets, which are described by a scale and translation parameters. The wavelets are localized in both time and frequency and can be adapted to the specific characteristics of the analyzed signal. The CWT is represented by the following equation:

$$CWT(u, s) = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} x(t) \psi^* \left(\frac{t - u}{s} \right) dt$$

Where s is the scale parameter, u is the translation parameter, $x(t)$ is the signal to be transformed, $\psi(t)$ is the wavelet function and $\psi^*(t)$ is the complex conjugate of the wavelet function. The

scale parameter determines the frequency resolution of the transform, where larger s corresponds to higher frequency resolution and smaller s to lower frequency resolution. The translation parameter determines the temporal resolution of the transform, where larger u corresponds to higher temporal resolution and smaller u to lower temporal resolution [58].

Applying CWT to a medical signal for different s and u values generates an image showing different frequency components of the signal at different time points, known as a CWT image. The CWT image can then be used as an input for ML model training. An example of a CWT image is shown in Fig. 3.4.

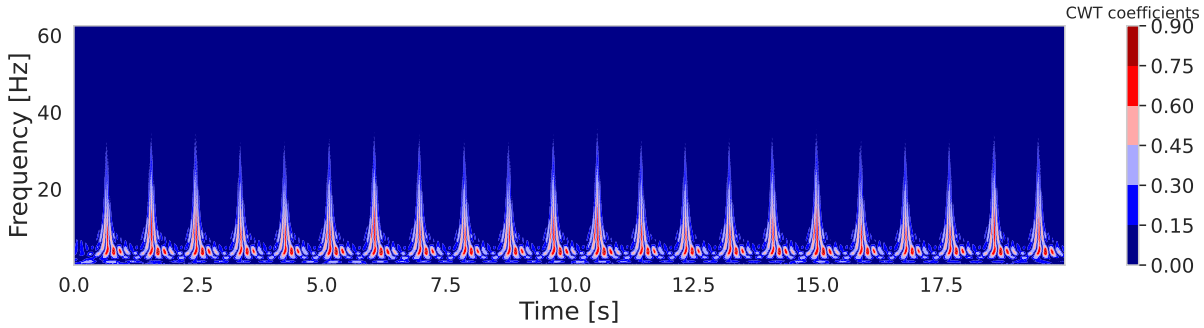


Figure 3.4: CWT image which is a result of applying CWT to ECG signal for different scale and translation values

Chapter 4

Models

A variety of ML models is trained during the research. The theoretical basis of each model is described in the following sections.

4.1. Logistic Regression (*LogR*)

Logistic regression is a model that performs well in the case where classes are linearly separable. It works well for both binary and multi-class cases. Logistic regression consists of two basic equations [49]:

$$z = w^{\mathrm{T}}x = w_0x_0 + w_1x_1 + \ldots + w_nx_n \tag{4.1}$$

$$\phi(z) = P(y = 1|z) = \frac{1}{1 + e^{-z}} \tag{4.2}$$

Where z is the product of the input x and weights w, $x_1, x_2, ...x_2$ are independent variables, $w_0, w_1, w_2, ...w_n$ are the learned weights ($x_0 = 1$), and ϕ is the sigmoid function. The loss function used in Logistic Regression is the cross-entropy loss:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j}) \tag{4.3}$$

where N is the number of samples, M is the number of classes, $y_{i,j}$ is 1 if the i-th sample belongs to the j-th class, and 0 otherwise and $p_{i,j}$ is the predicted probability that the i-th sample belongs to the j-th class.

The Logistic Regression model used in this study is imported from the *scikit-learn* [47] framework, which enables additional hyper-parameters used to control the training process. The objective function used in *scikit-learn* implementation is defined in eq. 4.4:

$$\min_{W} -C\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j}) + r(W) \tag{4.4}$$

where C is the inverse of regularization strength, $r(W)$ is the weights regularization term (later set to L_2) and W is the matrix of weights, where each row corresponds to a specific class. The other LogR hyper-parameters include $fit\_intercept$ which decides whether to add the bias

term to the decision function and the $class\_weight$ hyper-parameter which decides if classes are weighted during the loss calculation (set to "balanced" so the classes are weighted according to their frequencies in the input data). Also, scikit-learn's implementation allows one to choose the algorithm for the optimization problem. The $solver$ hyper-parameter controls which solver to use and max_iter controls the maximum iterations number, that the solver takes to converge. Logistic Regression default hyper-parameter values used in this study are present in table 7.2.

4.2. Linear Regression (*LinR*)

Linear Regression is an algorithm used to model the relationship between a dependent variable and one or more independent variables. It consists of one basic equation

$$y = w^{\mathrm{T}}x = w_0x_0 + w_1x_1 + \ldots + w_nx_n \tag{4.5}$$

where y is the dependent variable, $x_1, x_2, ...x_2$ are independent variables ($x_0 = 1$), and $w_0, w_1, w_2, ...w_n$ are the model's parameters, which are learned from the data.

The Linear Regression algorithm aims to find the values of w to obtain the best fit to the data that it is trained on. The quality of a fit is measured by a cost function, such as L_2 loss:

$$L_2 = \sum_{i=1}^{N}(w^{\mathrm{T}}x_i - y_i)^2 \tag{4.6}$$

where $w^{\mathrm{T}}x_i$ is an output of regression model for sample x_i and learned weights w and y_i is the true value belonging to that sample [49].

The Linear Regression model used in this study is imported from the *scikit-learn* [47] framework, which enables additional hyper-parameters used to control the training process. The objective function used in *scikit-learn* implementation is defined in eq. 4.7:

$$\min_W \|Xw - y\|_2^2 \tag{4.7}$$

where X is the matrix with independent variables and y is the vector with target values. LinR hyper-parameters include $fit\_intercept$ which decides whether to add the bias term to the decision function and the $positive$ hyper-parameter, which determines whether the weights should be positive. Linear Regression default hyper-parameter values used in this study are present in table 7.3.

4.3. Decision Tree (*DT*)

A Decision Tree is a model, that is capable of being trained for both classification and regression tasks. It is a tree-based model that recursively splits the data into smaller sets based on the input feature values. The goal is to create subsets (or leaves) that are as pure as possible with respect to the target variable. The input to the decision tree can be both categorical and numerical data.

For classification tasks, the goal is to create subsets (or leaves) that contain as many observations from a single class as possible. For regression tasks, the goal is to create subsets that have similar target variable values.

The decision tree starts from the root, from which the data is split taking into account the feature that gives the largest information gain (IG). The splitting process is repeated until all the data in the split set belongs to the same class. The use of this strategy can result in decision trees with multiple levels, leading to the over-fitting of the model. To prevent this, a technique called pruning is used [49]. Pruning involves eliminating a node whose removal will improve the loss of the test set. We trim the tree as long as the loss on the test set improves.

The most important thing in creating a decision tree is to maximize information gain:

$$\arg\max_f \; IG(D_p, f) = \arg\max_f \; I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j), \tag{4.8}$$

where f is the feature against which we perform the partitioning, D_p and D_j are the data sets of the parent node and child node, I is the measure of information impurity, N_p is the total amount of data of the parent node, and N_j is the number of data of the child node. Based on the formula 4.8, it can be seen that the information gain (IG) is the difference between the impurity of the parent node and the sum of the impurity of the child nodes. This leads to the conclusion that the lower the impurity of the children's nodes, the higher the information gain. Due to the possible complexity of some decision trees, most libraries use binary division of nodes. After taking this into account, the formula for information gain is as follows:

$$IG(D_p, f) = I(D_p) - \frac{N_{\text{left}}}{N_p} I(D_{\text{left}}) - \frac{N_{\text{right}}}{N_p} I(D_{\text{right}}), \tag{4.9}$$

where *left* and *right* are the labels of the left and right nodes, respectively. To evaluate the impurity or criteria for splitting the nodes of binary decision trees, different measures are used, the choice of which depends on the type of task (classification or regression). In the case of classification, measures such as Gini index (I_G), or entropy (I_H) are used [49]. The choice of entropy is equivalent to the cross-entropy loss. In the case of regression, one uses the L_2 loss (eq. 4.6), mean squared error (MSE), mean absolute error (MAE), or Half Poisson deviance.

The Decision Tree model used in this study is imported from the *scikit-learn* [47] framework, which enables additional hyper-parameters used to control the training process. The objective function (later called $criterion$) used in *scikit-learn* implementation is set to "squared_error" (which is equivalent to L_2 loss, eq. 4.6) for regression task and to "gini" (which measures the Gini impurity) for classification tasks. Tree growth is controlled by max_depth and max_leaf_nodes parameters, the former sets the maximum depth of the tree and the latter the maximum number of leaf nodes (both help to control the overfitting). Each node split is controlled by $splitter$, $min\_samples\_split$, $max\_features$, and $min\_impurity\_decrease$ hyper-parameters. The $max\_features$ hyper-parameter controls the number of features taken into account while performing the split and the $splitter$ decides which splitting strategy to use, i.e. to choose the best split or the best random split. The split is performed if the $min\_samples\_split$ samples meet the splitting condition and if the impurity decreases by at least $min\_impurity\_decrease$ value. The other important hyper-parameters include the $min\_samples\_leaf$, which defines the minimum number of samples, that is required for a node to become a leaf, and the $class\_weight$ hyper-parameter (used for classification only) which decides if classes are weighted during the loss calculation (set to "balanced" so the classes are weighted according to their frequencies in

the input data). Decision Tree default hyper-parameter values used in this study are present in table 7.4.

4.4. Light Gradient Boosting Machine (*LGBM*)

LightGBM is a framework that uses different gradient-boosting methods to train an ensemble of decision trees. LGBM model is using the boosting technique to train the weak learners (decision trees) using gradient optimization. The learning process involves fitting weak learners sequentially, where each next decision tree aims to fix errors from the previous one. The larger the error, the more weight it gets during loss calculation, thus there is more effort put into decreasing that error. The final model is obtained by calculating the weighted average of all models. The one big difference between LGBM and other boosting algorithms is that it uses the leaf-wise tree growth algorithm [31]. LGBM model may be trained for both classification and regression tasks. In the case of the former is used cross-entropy loss (eq. 4.3) and for the latter, it uses L_2 loss (eq. 4.6).

The LGBM model used in this study is imported from the *LightGBM* [31] framework, which enables additional hyper-parameters used to control the training process. The objective function used in *LightGBM* implementation is set to L_2 loss (eq. 4.6) for the regression task and Cross-Entropy loss for the classification tasks. The boosting algorithms hyper-parameters include the $boosting\_type$ which decides which boosting algorithm to use and the $learning\_rate$ hyper-parameter which defines the learning rate used in boosting. The number of the trained trees (the weak learners) is controlled by the $n\_estimators$ hyper-parameter. Each tree is controlled by a set of hyper-parameters that are similar to the ones from the Decision Tree model. Tree growth is controlled by max_depth and num_leaves parameters, the former sets the maximum depth of the single tree and the latter the maximum number of leaf nodes (both help to control the overfitting). Each node split is controlled by $colsample\_bytree$, $subsample$, and min_split_gain hyper-parameters. The $colsample\_bytree$ and $subsample$ hyper-parameters control the number of features (the former) and data (the latter) randomly selected for tree building. The split is performed if the loss is reduced by at least min_split_gain value. The other important hyper-parameters include the $min\_child\_samples$, which defines the minimum number of samples, that is required for a node to become a leaf, and the $class\_weight$ hyper-parameter (used for classification only) which decides if classes are weighted during the loss calculation (set to "balanced" so the classes are weighted according to their frequencies in the input data). The regularization may also be included in the training process by setting the reg_alpha and reg_lambda hyper-parameters. The former defines the L_1 and the latter the L_2 regularization term on weights. Light Gradient Boosting Machine default hyper-parameter values used in this study are present in table 7.5.

4.5. Multilayer Perceptron (*MLP*)

A simple multilayer perceptron is a feedforward neural network built using multiple layers of connected perceptrons. The MLP considered in this study is the deep neural network with at least 2 hidden layers. Each layer consist of a specific number of neurons (perceptrons), which learn

how to process the data to produce the best predictions. A single perceptron accepts $x_0, x_1, ..., x_n$ inputs, multiplies those inputs with their weights $w_0, w_1, ..., w_n$ and adds bias term b. Then, the activation function a is applied to the sum of those multiplications to produce output y (scalar):

$$y = f(x; w, b) = a(w^{\mathrm{T}}x + b) \tag{4.10}$$

A single perceptron allows fitting only a simple function to the data, which is often not enough for most of the tasks. Many perceptrons connected to the same input (but not among each other) create a layer, which works similarly to the single perceptron, except that it has a set of weights for each neuron and it produces a vector instead of a scalar. The output of l-th layer would be:

$$y^{(l)} = f^{(l)}(x^{(l-1)}; w^{(l)}, b^{(l)}) = a^{(l)}(w^{(l)\mathrm{T}}x^{(l-1)} + b^{(l)}) \tag{4.11}$$

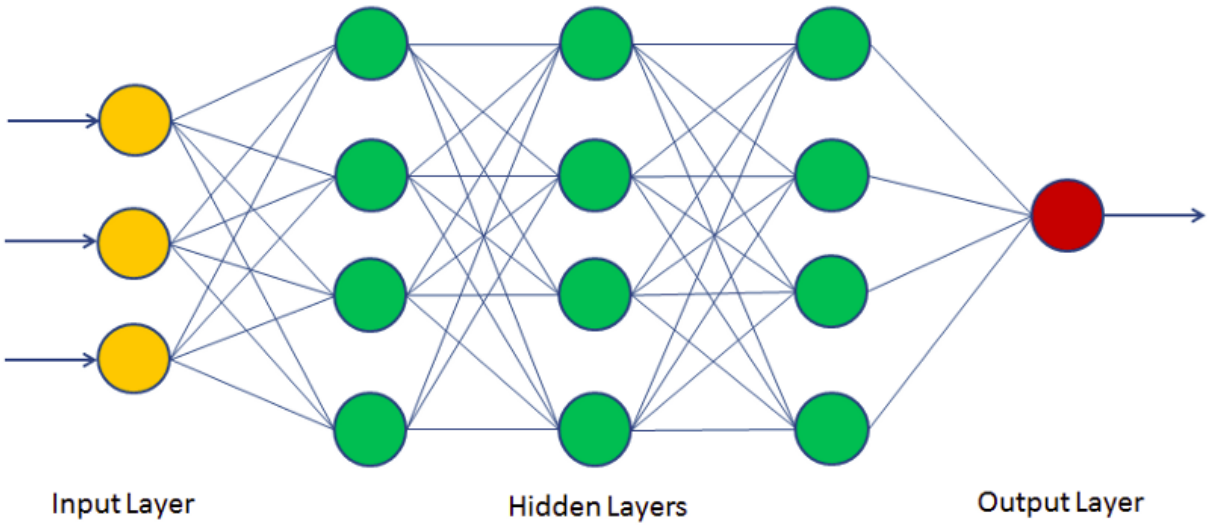


Figure 4.1: An example of MLP neural network with three hidden layers (source: [44])

Stacking many such layers creates a Multilayer perceptron (MLP), which thanks to non-linear activation functions between the layers is capable of producing more complex functions. An MLP is usually constructed of 3 layers at least (including input and output layers). An example of MLP architecture is shown in Fig. 4.1. The first layer is called an input layer, the last layer is called an output layer, and all layers between the first and last are called the hidden layers. An equation for the output of a 3-layer MLP neural network is the following (weights and biases are omitted for simplicity) [19]:

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))) \tag{4.12}$$

Formula 4.12 is called a forward pass. After the forward pass, a loss for the predictions and the true value is calculated. Loss tells us how far from the true value was the predicted value. Neural network training aims to reduce the differences between predicted values and true values, which is equivalent to reducing the loss value. The loss value is calculated using the loss function, which depends on the task. Examples of loss functions include MSE loss (eq. 4.6) for regression tasks or Cross-Entropy loss (eq. 4.3) for classification tasks. MLP reduces the loss value by using the gradient descent optimization algorithm. Gradient descent starts with a backward pass, i.e. calculating the partial derivatives of the loss with respect to all neural network parameters. After

derivatives calculation, neural network weights are updated according to their gradients. The parameter which controls the intensity of those updates is called the learning rate. Let θ be the neural network learned weights, J be the loss function, ∇ be the gradient operator and η be the learning rate. Then weights θ are updated to follow the gradient of the loss with respect to the weights ($\nabla_\theta J(\theta)$):

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{4.13}$$

MLP in its raw form may suffer from overfitting and slow and unstable learning. The former may be reduced by adding a dropout layer [56] to each MLP layer, which randomly turns off the neurons during the training process. The latter may be resolved by adding a Batch Normalization layer before the activation function of each MLP layer. The batch Normalization layer performs normalization of layer outputs, which results in faster and better (in terms of loss) convergence [27].

The MLP model used in this study is implemented using the *PyTorch* framework [46]. The implementation enables additional hyper-parameters used to control the training process. The objective function is set to L_2 loss (eq. 4.6) for the regression task and Cross-Entropy loss (eq. 4.3) for the classification tasks. Fig. 4.2 shows the MLP architecture that is used in this study. The architecture is parameterized by four hyper-parameters. The first one, $hidden\_dims$ controls the number of hidden layers and the dimensions of each Linear layer inside. All other hyper-parameters ($batch\_norm$, $activation$, and $dropout$) are used to define how the layers are constructed. The $batch\_norm$ decides whether to apply Batch Normalization (BN) layer after the Linear layer. The $activation$ hyper-parameter defines which activation function to use (set to ReLU by default) in Activation Layer which is placed after the (optional) BN layer. The $dropout$ value defines the probability used for dropout layers placed after the Activation layer. Multilayer Perceptron default hyper-parameter values used in this study are present in table 7.6.

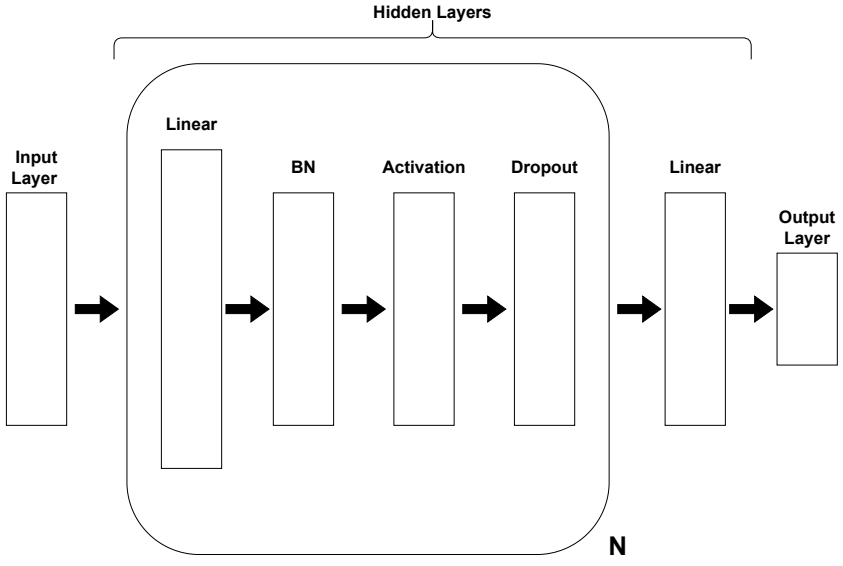


Figure 4.2: Multilayer Perceptron architecture used in this study. N stands for the number of layers defined by $hidden\_dims$ hyper-parameter. The last linear layer is used to map the features to the outputs, which in case of classification is wrapped with softmax activation function.

4.6. Convolutional Neural Network (*CNN*)

Convolutional Neural Networks are similar to Multilayer Perceptron in terms of being composed of units with learnable weights and biases. The core building blocks of CNN are the Convolutional layer, the Pooling layer, and the Fully-Connected (FC) layer.

The first and most important is the Convolutional layer composed of filters with learnable weights and bias. The filters usually have small spatial dimensions and large depth dimensions. It is important to note, that filter weights (parameters) are shared across a single depth slice, which helps to reduce the total number of parameters. During the training process filters learn to extract spatial and temporal features from the input data. The hyper-parameters of the classical Convolutional layer include the filter size (*kernel_size*), filter step (*stride*), number of input channels (*in_channels*), number of out channels (*out_channels*) and amount of padding (*padding*) for the input. The next block is a pooling layer, which is used to reduce the spatial dimensionality of the input passing through the neural network. That kind of reduction reduces the number of learnable parameters and thus it helps to control the overfitting. The pooling layer is also composed of filters, but in contrast to the Convolutional layer, pooling filters have no learnable parameters. The pooling filters are usually performing simple operations like taking the maximum value (max pooling) or averaging all values (average pooling). Pooling is usually applied using filters of size 2x2 (or 2 for 1-D CNN) with stride 2. The Convolutional Neural Network ends with a Fully-Connected layer which is used to map the features from convolutional blocks to the predicted value. Construction of a Convolutional neural network consists of stacking blocks of convolutional, activation, and pooling layers on top of each other and adding Fully-Connected layer as last in the network [30], [19]. The CNN training process, same as with MLP, involves using a gradient descent algorithm to minimize the loss function by updating the learnable weights.

An example of CNN architecture is shown in Fig. 4.3.

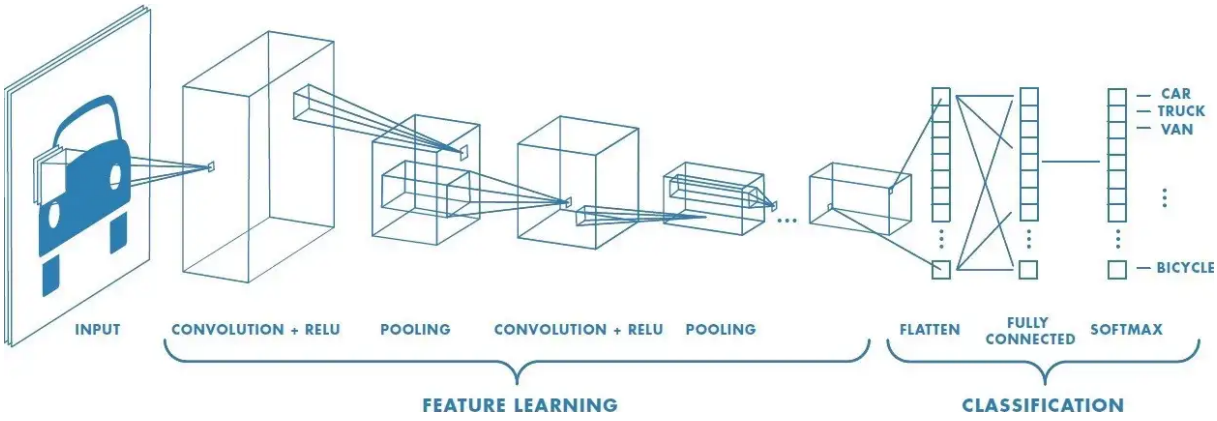


Figure 4.3: An example of a Convolutional Neural Network composed of Convolutional, Pooling, Activation, and Fully-Connected Layers. That kind of CNN could be used for image classification tasks (source: [52]).

Stacking many convolutional blocks leads to the vanishing gradient problem. The convolutional architecture called ResNet [22] (shown on Fig. 4.4) helps with the vanishing gradient problem by utilizing the residual connections, which allow the gradient to pass easily through the network.

The CNN, the same as MLP allows for additional Dropout and Batch Normalization layers. The addition of those layers usually helps to stabilize the training process and prevent overfitting.

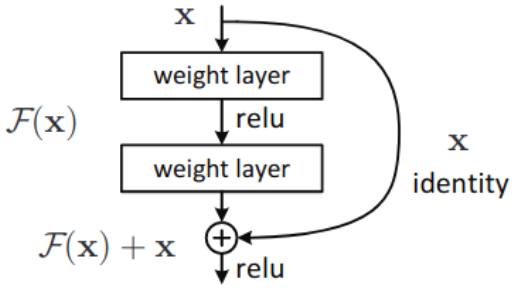


Figure 4.4: Single block of ResNet architecture (source: [22])

The CNN model used in this study is implemented using the *PyTorch* framework [46]. The implementation is based on 1-D ResNet by [67] and enables additional hyper-parameters used to control the training process. The objective function is set to L_2 loss (eq. 4.6) for the regression task and Cross-Entropy loss (eq. 4.3) for the classification tasks.

The architecture includes the CNN features extractor followed by a Feed Forward layers. The CNN feature extractor consists of stem convolution and residual layers. The stem convolution is parameterized by three hyper-parameters, the size of the filters $conv\_0\_kernel\_size$, filter's step $stride$, and $conv\_0\_channels$ which defines the number of output channels. The number of residual blocks in each residual layer is defined by the $layers$ hyper-parameter. The $layer$ hyper-parameter is a list of integer values, where the i-th value defines the number of residual blocks in the i-th residual layer. Each of the residual blocks consists of two convolutional blocks and one downsample layer. The $block\_kernel\_1$ defines the filter size of the first convolutional block and $block\_kernel\_2$ defines the filter size of the second convolutional block. The stride of both convolutional blocks is controlled by the $block\_stride$ hyper-parameter. The Feed Forward (FF) layers that come after the convolutional extractor are similar to the MLP layers described in the previous section. The hidden layers are controlled by the ff_hidden_dims hyper-parameter, the Batch Normalization layer and ReLU activation are used by default, and the dropout is controlled by the $ff\_dropout$ hyper-parameter. Convolutional Neural Network default hyper-parameter values used in this study are present in table 7.7.

Chapter 5

Literature review

Every year, more and more publications are related to the application of artificial intelligence to medical signal tasks. This shows that this topic is crucial in terms of the development of science and the use of artificial intelligence in detecting and treating various types of diseases. The collection, processing, and analysis of medical signals is a very complex topic, resulting in the existence of a large number of methods for this purpose. Differences between methods appear at every stage of projects in which medical signals play a central role. Beginning with differences resulting from cleaning (filtering) the signals through different ways of representing them and ending with ML models learned from them.

The preprocessing of medical signals is crucial in feature-based approaches and is often needed only to clean the signals from the noise. Researchers from [21], [20], [2] articles utilized Butterworth bandpass filters to filter out the noise outside of the specified range. Time-frequency domain-based methods were also applied, [26] and [29] used discrete wavelet decomposition to eliminate very low and very high-frequency coefficients and [11] and [61] filtered the signals with Fourier-based methods. The raw waveforms, without any preprocessing applied, were primarily used for deep learning-based approaches [23].

After being preprocessed, medical signals are often either put into deep learning models, which learn the features or are used for a hand-crafted feature extraction process. Due to differences arising from the characteristics of medical signals, the feature extraction process differs between signal types, however, some feature groups are common for all signal types. That is, the statistical features, which were extracted by authors of the papers [10], [3], [11], [39], the wavelet transform based features extracted by [68], [1], [45], [42], [8] or the frequency features utilized in [39]. There also exist feature groups extracted for specific signal types, like the HRV features for ECG and PPG signals [11], [24], [13], [15] and morphological features extracted for PPG beats [39], [24], [35] or ECG beats [51], [11] signals. In contrast to feature extraction, the other way to deal with signals after (optional) processing is to represent the signals as a 2-D spectrogram, which [62] did or as Fourier frequency spectrum [4]. Authors from [33] have shown that it is also possible to train a deep neural network to learn the signal feature embeddings.

The preprocessed signals and extracted tabular features are input for training ML models. Many types of ML models are used for the regression and classification of medical signals. Classical ML algorithms, like AdaBoost or Multi-Linear Regression, were used in feature-based training in [11] and [20]. The vast majority of research groups focus on deep models for this type

of task [48], [33], [2], [59]. Among the deep models, the most commonly implemented are those based on convolutional networks [50], [54], [60], [4], [62]. The analysis from [4] includes using several representations of PPG and ECG signals to predict the corresponding blood pressure values. Signal waveforms, the Fourier frequency spectrum, and the two derivatives were jointly used as the input to fully convolutional networks. It is also appropriate to use recurrent networks for this type of task, which [39], [21], [21], [35] did. Researchers from [35] have shown an interesting approach, where the PPG and ECG features extracted from the successive beat episodes were used as input to the bidirectional LSTM model. The combination of the previous two model types (CNN + RNN) is also gaining popularity in terms of medical signals classification, and regression tasks [9], [23], [41], [61], [60], [38]. Authors of [60] introduced a new architecture that uses two convolutional neural networks, a bidirectional LSTM, and a fully connected layer. The CNN modules are composed of different filter sizes at the first layers, a smaller one to capture the temporal information and the larger one to capture the frequency information better. There is also a comparison of classical ML models trained with tabular features and deep learning models trained on raw data [38]. Authors have shown that for the ECG sleep stage classification task, there exists a superiority of CNN + LSTM neural network trained on raw data over the feature-based models. Another research group compared feature-based classical ML models (Linear Regression, Random Forest, AdaBoost, and Support Vector Machine) against the deep learning models (GRU and bi-LSTM) for blood pressure regression task [39]. They have also shown the superiority of deep learning models (GRU and bi-LSTM) over other tested approaches. Deep neural networks are also trained using tabular features, [24] successfully trained the MLP neural network using PPG features for the blood pressure estimation task.

Authors from [59] performed analyses on the PTB-XL dataset. The MIMIC dataset was utilised in papers [26], [54], [21], [20], [61], [29], [2], [35], [4], [24], [39]. The use of the Sleep-EDF dataset is present in [41], [60] papers.

Chapter 6

Problem description

This study aims to investigate the impact of medical signal representations on the quality of the results of ML models. The quality of the results is mainly understood as the prediction quality, but the computational complexity, and disk space needed to store the models will also be investigated. Another element to be assessed is domain knowledge needed for representation extraction, which is crucial for AI projects implementing intelligent systems inferring from medical signals.

The analysis includes 4 types of representations: whole signal waveforms (WSW), whole signal features (WSF), aggregates beat waveforms (ABW), aggregated beat features (ABF), and 5 types of ML models: Regression (Reg), Decision Tree (DT), Light Gradient Boosting Machine (LGBM), Multilayer Perceptron (MLP), Convolutional Neural Network (CNN). The study was conducted for 3 datasets: PTB-XL, Sleep-EDF, and MIMIC. The task for PTB-XL and Sleep-EDF datasets was multiclass classification and the task for the MIMIC dataset was a regression.

The study was diverse in many aspects to ensure the generalization of the results. First, the representations are obtained from different types of medical signals and for each signal class, a specific set of representations was extracted. Also, different datasets are used, with varying tasks, namely classification, and regression. Finally, for each task and representation type, different types of ML models are trained and evaluated. That kind of evaluation will allow investigation of the impact of each representation type on each ML model type.

Each dataset will be analyzed separately, and the conclusions of each multi-level analysis will be compared and summarized. The first step will be to compare the results obtained for specific representation types and different kinds of ML models. This comparison will allow to assess which ML model is the most suitable for a given representation type. Next, the results obtained for specific ML model types and different medical signals representations will be compared with each other. This analysis will show which type of representation is most suitable for the given ML model. Combinations of representation types and models that achieved the best results in the context of different evaluation options will also be identified.

The final step will be to check the repeatability of the results of previous analyses between different datasets. All previous observations will be summarized, and conclusions will be formulated.

As a result of this research, it is expected to demonstrate the most efficient combination of the type of ML model along with the type of medical signal representation on which the model is

supposed to be trained. For a given datasets D:

$$D = \{\text{PTB-XL}, \text{Sleep-EDF}, \text{MIMIC}\} \tag{6.1}$$

ML model types M:

$$M = \{\text{Reg}, \text{DT}, \text{LGBM}, \text{MLP}, \text{CNN}\} \tag{6.2}$$

and medical signals representation types R (which depends on a chosen dataset d):

$$R = \begin{cases} \{\text{WSW}, \text{WSF}, \text{ABW}, \text{ABF}\}, & \text{if } d \in \{\text{PTB-XL}, \text{MIMIC}\} \\ \{\text{WSW}, \text{WSF}\}, & \text{otherwise} \end{cases} \tag{6.3}$$

the best combination of model type m and representation type r for dataset d is the one that obtained the highest task-specific evaluation score E (eq. 6.4):

$$m_d, r_d = \underset{m \in M, r \in R}{\arg\max}\, E(m, r, d) \tag{6.4}$$

For each ML model $m \in M$ analyzed, the most appropriate type of medical signal representation will be determined (eq. 6.5):

$$r_{m,d} = \underset{r \in R}{\arg\max}\, E(m, r, d) \tag{6.5}$$

Also, for each representation type $r \in R$ analyzed, the most appropriate type of ML model will be determined (eq. 6.6):

$$m_{r,d} = \underset{m \in M}{\arg\max}\, E(m, r, d) \tag{6.6}$$

Chapter 7

Research procedure

The purpose of this study is to investigate the effect of medical signal representations on the quality of the results of ML models. To achieve this goal, an analysis of the results obtained for many different types of representations, for many different data sets, and for different ML models that will be trained using these representations is required. Each analyzed dataset consists of different medical signals and each medical signal requires different processing and has different feature extraction strategies. This chapter covers the experiment's setup, that is, medical signals processing, feature extraction, signals representation types, and models used to evaluate tasks associated with each dataset. Section 7.1 describes the processing for each signal and section 7.2 covers the features extraction. Then, in section 7.3 there is a description of representation types used for further research. The models used in the research are described in section 7.4. Each of the models was subjected to a hyperparameter optimization process, as described in the 7.5 section. Finally, the evaluation process description is placed in section 7.6. The datasets used and related ML models tasks are described in the next chapter (chapter 8).

7.1. Medical signals processing

Each medical signal requires different processing for signal cleaning or beats segmentation. The processing approaches for all analyzed medical signals are described in this section.

ECG The ECG processing consists of cleaning the signal, locating beats intervals, and aggregating beats into a single ECG episode. The *NeuroKit2* [37] library with its algorithm was used to filter the ECG signals. The beat segmentation process is done with the use of the *NeuroKit2* [37] library (which uses *biopeaks* package [6]) and self-written heart rate-based algorithms. The beats are then aligned (to R peak location) and averaged.

PPG The PPG processing consists of cleaning the signal, locating beats intervals, and aggregating beats into a single PPG episode. The *NeuroKit2* [37] library with algorithm from [14] was used to filter the PPG signals. After the cleaning process, it is possible to segment the signal into single episodes (beats) and extract the features. The beat segmentation process is done with the algorithm based on [14]. The complete set of PPG features is listed below.

EEG It is possible to extract the features straight from the raw signal.

EOG The EOG processing consists only of cleaning the signal. The *NeuroKit2* [37] library with its algorithm was used to filter the EOG signals.

7.2. Feature extraction

The feature extraction process is different for each medical signal analyzed in this work. Those differences are described in this section.

ECG All features from section 2.2.1 are extracted from ECG signals and used for further analysis. Whole signal features representation consists of all the features available for ECG signal and aggregated beat features representation consists only of aggregated beat features. The mother wavelet used for DWT features extraction is Daubechies 8 wavelet (db8).

PPG All features from section 2.2.2 are extracted from PPG signals and used for further analysis. Whole signal features representation consists of all the features available for PPG signal and aggregated beat features representation consists only of aggregated beat features. The mother wavelet used for DWT features extraction is Daubechies 8 wavelet (db8).

EEG All features from section 2.2.3 are extracted from EEG signals and used for further analysis. The whole signal features representation consists of all the features available for the EEG signal. The mother wavelet used for DWT features extraction is Daubechies 8 wavelet (db8).

EOG All features from section 2.2.4 are extracted from EOG signals and used for further analysis. The whole signal features representation consists of all the features available for the EOG signal. The mother wavelet used for DWT features extraction is Daubechies 8 wavelet (db8).

7.3. Representation types

The research was carried out for four different types of representations:

1. Whole signal waveforms – time series representation for whole signal (raw form),
2. Whole signal features – basic and expert features extracted from whole signal waveforms,
3. Aggregated beat waveforms – time series representation for an aggregated single beat,
4. Aggregated beat features – basic and expert features extracted from the aggregated beat waveform.

This choice was due to their diversity in terms of the nature of representation (waveforms and tabular features), representation size, and the amount of expert knowledge needed to extract the representation. The CWT image representation was not used for further research because its use exceeds the memory capacity of the computing unit on which the research was carried out.

The dimensionality of each representation differs between signals used for model training and is presented in the tab. 7.1.

Table 7.1: Number of samples and features for each signal for different representation types. The ECG signal is present in both PTB-XL and MIMIC datasets (with different number of data points in each), so it occurs twice in the table.

Dataset	**Periodic**	**Representations dimensionality**			
		Whole signal waveform	Whole signal features	Aggregated beat waveform	Aggregated beat features
ECG PTB-XL	Yes	1000	303	100	78
ECG MIMIC	Yes	7500	363	100	78
PPG	Yes	7500	329	100	44
EEG	No	3000	266	-	-
EOG	No	3000	265	-	-

7.4. Models

Depending on the type of task, it is necessary to use different ML models (classification or regression variants). The choice of models is guided by their diversity in terms of the algorithms on which they are based, the types of data they are designed for, and their computational and memory complexity.

The research is conducted for the Regression (Reg), Decision Tree (DT), Light Gradient Boosting Machine (LGBM), Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) models, which are described in this section.

Regression (Reg) Logistic Regression (LogR) for classification tasks and Linear Regression (LinR) for regression task. Both models are imported from the *scikit-learn* framework [47]. Hyper-parameters of LogR model are described in more detail in section 4.1 and in short in table 7.2. Hyper-parameters of LinR model are described in more detail in section 4.2 and in short in table 7.3.

Table 7.2: Logistic Regression hyper-parameters description and default values. The names of the hyper-parameters that underwent the optimization process are in bold.

Hyper-parameter	**Description**	**Default**
C	inverse of a regularization strength	0.1
fit_intercept	whether to add the bias term to decision function	True
class_weight	whether to apply class weighting	balanced
solver	optimization algorithm to use	lbfgs
max_iter	maximum iterations number	500

Table 7.3: Linear Regression hyper-parameters description and default values. The names of the hyper-parameters that underwent the optimization process are in bold.

Hyper-parameter	**Description**	**Default**
fit_intercept	whether to calculate the bias term (intercept)	True
positive	whether to force the coefficients to be positive	False

Decision Tree (DT) DT implementation is imported from the *scikit-learn* framework [47], its hyper-parameters are extensively explained in section 4.3 and briefly described in table 7.4.

Table 7.4: Decision Tree model hyper-parameters description and default values. The names of the hyper-parameters that underwent the optimization process are in bold.

Hyper-parameter	**Description**	**Default**
splitter	the splitting strategy, i.e. whether to choose the best split or a best random split	best
criterion	the choice of impurity metric	squared_error (regression) gini (classification)
max_depth	maximum depth of the tree	-1
min_samples_split	minimum number of samples required to perform a node split	2
min_samples_leaf	minimum number of samples required for a node to become a leaf	1
max_features	the ratio of features that are taken into account when performing a split	1.0
max_leaf_nodes	maximum number of leaf nodes	None
min_impurity_decrease	a value that defines a minimal required decrease in impurity to perform a split	0
class_weight	whether to apply class weighting	balanced

Light Gradient Boosting Machine (LGBM) The *LightGBM* framework [31] is used for LGBM model implementation. The hyper-parameters used to control the training process are explained in section 4.4 and briefly described in table 7.5.

Table 7.5: LGBM model hyper-parameters description and default values. The names of the hyper-parameters that underwent the optimization process are in bold.

Hyper-parameter	Description	Default
boosting_type	which boosting algorithm to use	dart
num_leaves	maximum number of leaves for weak learners	31
max_depth	maximum depth of weak learners	-1
learning_rate	learning rate used for boosting	0.1
n_estimators	number of weak learners to train	600
class_weight	whether to apply class weighting	balanced
min_split_gain	a minimal loss reduction to perform a split	0
min_child_samples	minimum number of samples required for a node to become a leaf	20
subsample	Amount of data (ratio) randomly selected during training	1.0
colsample_bytree	Number of features (ratio) to randomly select for tree building	1.0
reg_alpha	L_1 regularization	0
reg_lambda	L_2 regularization	0

Multilayer Perceptron (MLP) The MLP model implementation is done using the *PyTorch* framework [46]. Its hyper-parameters are explained in detail in section 4.5 and briefly described in table 7.6.

Table 7.6: MLP model hyper-parameters description and default values. The names of the hyper-parameters that underwent the optimization process are in bold.

Hyper-parameter	Description	Default
hidden_dims	dimensionality of hidden layers	[256, 512, 256]
batch_norm	whether to apply batch normalization in each layer	True
dropout	probability used for dropout layers	0.2
activation	type of activation used for each layer	ReLU
learning_rate	the initial value of optimizer's step size	0.01
weight_decay	L_2 regularization on neural network's weights	0.001
max_epochs	maximum number of training epochs	100
patience	number of epochs with no improvement before stopping the training process	15
lr_scheduler_patience	number of epochs with no improvement before reducing the learning_rate value	7
lr_scheduler_factor	how much to reduce the learning_rate	0.7

Convolutional Neural Network (CNN) CNN implementation was based on [67] and implemented using *PyTorch* framework [46]. Its hyper-parameters are extensively explained in section 4.6 and briefly described in table 7.7.

Table 7.7: CNN model hyper-parameters description and default values. The names of the hyper-parameters that underwent the optimization process are in bold.

Hyper-parameter	Description	Default
conv0_kernel_size	kernel size for primary the convolution	7
conv0_stride	stride for the primary convolution	3
conv0_channels	number of channels to which the depth is extended	128
layers	number of residual blocks in each layer	[1, 1, 1]
block_kernel_1	kernel size for first convolutional layer in residual block	5
block_kernel_2	kernel size for second convolutional layer in residual block	3
block_stride	defines stride for downsampling layer	2
ff_hidden_dims	hidden layers dimensionality in FC layer	[128, 128]
ff_dropout	probability used for FC dropout layers	0.2
learning_rate	the initial value of optimizer's step size	0.01
weight_decay	L_2 regularization on neural network's weights	0.001
max_epochs	maximum number of training epochs	100
patience	number of epochs with no improvement before stopping the training process	15
lr_scheduler_patience	number of epochs with no improvement before reducing the learning_rate value	7
lr_scheduler_factor	how much to reduce the learning_rate	0.7

Both deep learning models (MLP and CNN) are subjected to the following training procedure. AdamW optimizer [36] is used along with the learning rate scheduler. The learning_rate and weight_decay optimizer's hyper-parameters are carefully chosen during the optimization process. The learning_rate decreases by a factor of 0.7 if the model's validation loss did not improve for 7 epochs. The Early Stopping is also used, i.e. the training process is terminated if the model did not improve on validation loss for 15 epochs and the best model is picked. The models are trained for a maximum of 100 epochs.

7.5. Hyper-parameter optimization

The proper research on assessing the impact of medical signals representations on the quality of ML models results should be preceded by the models' hyper-parameters optimization for each type of representation. Considering that, each ML model is subjected to a hyperparameter optimization process. The hyper-parameters of the models are optimized separately for each type of medical signal representation. The optimization process is carried out only on the PTB-XL dataset, and the sets of hyper-parameters selected from it will be also used for the MIMIC dataset. Sleep-EDF experiments are conducted with the use of default hyper-parameters values. Limiting the optimization of hyper-parameters to the PTB-XL dataset is due to the excessive time requirement necessary to optimize the models for each dataset separately. Also, the use of the same hyper-parameters for PTB-XL and MIMIC datasets is guided by the fact that in both datasets there are signals with similar characteristics (both ECG and PPG signals are HR-based).

The choice of hyper-parameters is guided by the value of the evaluation metric (F_1) on the validation set. The optimization process is implemented according to the pseudocode 1. *FindBestHyperparameters* function is used to find the best hyper-parameters for a given model type (*Model*), representation type (*Representation*) and hyper-parameters settings (*search_space*, *default_hparams*). An example of hyper-parameters setting definition is shown in lines 1 – 12 of the pseudocode. *N* stands for the number of optimized hyper-parameters and *M* for the total number of model's hyper-parameters.

Getting the best optimization results would be possible by using the Grid Search approach, which involves checking all possible combinations of hyper-parameters. However, the evaluation of all possible combinations is highly time-consuming, making it necessary to use another approach (presented in the pseudocode 1). This approach involves checking each optimized hyper-parameter (setting the other hyper-parameters to their default values) and updating its default value when the score (F_1) exceeds the best score so far. Using this approach significantly reduces the number of hyper-parameter combinations, however, it is prone to choosing the order of the optimized hyper-parameters. Therefore, the order is carefully determined taking into account the characteristics of each model.

7.6. Evaluation

This section describes how the evaluation is conducted. The metrics for each task are described in section 7.6.1, and the evaluation procedure in section 7.6.2.

7.6.1. Metrics

Evaluation metric depends on the type of task (classification or regression). The following metrics are used:

Algorithm 1 Algorithm used to find best hyper-parameters for specific model type (*Model*), representation type (*Representation*) and hyper-parameters settings (*search_space*, *default_hparams*). Lines 1 – 5 and 7 – 12 show an example of hyper-parameters settings definition. The former defines hyper-parameters search space and the latter the default hyper-parameters values. *N* is the number of optimized hyper-parameters and *M* is the number of all model's hyper-parameters.

```
1:  search_space ← {}                                   ▷ Define hyper-parameters search space
2:  search_space[hparam_1_name] ← hparam_1_search_space
3:  search_space[hparam_2_name] ← hparam_2_search_space
4:  ...
5:  search_space[hparam_N_name] ← hparam_N_search_space
6:
7:  default_hparams ← {}                                ▷ Define default hyper-parameters values
8:  default_hparams[hparam_1_name] ← hparam_1_default_value
9:  ...
10: default_hparams[hparam_N_name] ← hparam_N_default_value
11: ...
12: default_hparams[hparam_M_name] ← hparam_M_default_value
13:
14: function FindBestHyperparameters(Model, Representation, search_space, default_hparams)
15:     best_hparams ← default_hparams                  ▷ Copy default
16:     best_score ← 0
17:     for hparam_i_name, hparam_i_search_space in search_space do
18:         for hparam_i_value_j in hparam_i_search_space do
19:             model_params ← best_hparams             ▷ Copy best
20:             model_params[hparam_i_name] ← hparam_i_value_j   ▷ Update current
21:             model ← Model(model_params)             ▷ Initialize model
22:             score ← evaluate(model, Representation) ▷ Evaluate model
23:             if score > best_score then
24:                 best_hparams[hparam_i_name] ← hparam_i_value_j   ▷ Update best
25:                 best_score ← score
26:             end if
27:         end for
28:     end for
29:     return best_hparams
30: end function
```

- classification – f1-score, F_1 – the harmonic mean of precision and recall. That metric was chosen because it works well on imbalanced data [16],

$$precision = \frac{TP}{TP + FP} \tag{7.1}$$

$$recall = \frac{TP}{TP + FN} \tag{7.2}$$

$$F_1 = \frac{2 * precision * recall}{precision + recall} \tag{7.3}$$

- regression – mean absolute error, MAE – the average of all absolute errors. This metric was chosen due to its use in other articles related to the MIMIC dataset.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i| \tag{7.4}$$

7.6.2. Evaluation procedure

All analyzed combinations of signal representations and ML models for each dataset are shown in table 7.8. Each of the 50 experiments followed the same procedure. The only differences are due to the different data sets, representations, and ML models used. The procedure for a single experiment is as follows:

1. dataset selection,
2. representation type selection,
3. extraction of the selected representation for all classes of signals belonging to a given dataset,
4. model initialization with hyper-parameters taken from the optimization process,
5. model training using the data from the selected representation type
6. model evaluation on the test set.

Table 7.8: Experiments setup for all datasets. The "Total" column refers to the total number of experiments for a dataset, which is calculated as total number of models and representation types combinations.

Dataset	**Representation**	**Models**	**Total**
PTB-XL	WSW WSF ABW ABF	LogR DT LGBM MLP CNN	20
MIMIC	WSW WSF ABW ABF	LinR DT LGBM MLP CNN	20
Sleep-EDF	WSW WSF	LogR DR LGBM MLP CNN	10
Total number of experiments:			50

The results for each model and representation variant will be compared using the prediction quality, i.e. F_1 for classification tasks and MAE for regression task. Also, the expertise needed to obtain the representation, the disk space needed to store the model and model's inference time will be included in the conclusions.

7.7. Environment

All the code needed to download the datasets, extract the representations and run the experiments is written with *python 3.8.10* programming language [63] configured with *Hydra* framework [70] and stored in a *github* repository [18]. The *Weights & Biases* framework is used to track and visualize all experiments [5]. The hardware setup used to train the ML models consists of NVIDIA GeForce RTX 3070 Laptop GPU for MLP and CNN models training and 16 CPUs for Regression, DT and LGBM models training.

Chapter 8

Datasets

The study was conducted for three different datasets the selection of which was guided by three factors. The first is the presence of medical signals selected for analysis. The second factor is the diversity of the datasets in terms of the machine learning task (classification or regression) associated with them. The last factor is the free access to the data. The MIMI-III, PTB-XL, and Sleep-EDF datasets were chosen. The datasets summary is presented in the tab. 8.1 and representations dimensionality in tab. 8.2. A description of each dataset is presented in the following sections.

Table 8.1: Datasets statistics

Dataset	**Input signals**	**Signal duration**	**Sampling frequency**	**Target**	**Data samples**
PTB-XL	12-lead ECG	10s	100	diagnostic class (classification)	16 272
MIMIC	ECG, PPG	60s	125	SBP (regression)	20842
Sleep-EDF	2-lead EEG, EOG	30s	100	sleep phase (classification)	196350

Table 8.2: Representations dimensionality for each dataset, where N - number of signal samples, R - number of signal regions, C - number of signal channels, F - number of features extracted from the signals

Dataset	**Periodic**	**Representations dimensionality**			
		Whole signal waveforms (shape $[N, R, C]$)	Whole signal features (shape $[F, 1]$)	Aggregated beat waveforms (shape $[N, R, C]$)	Aggregated beat features (shape $[F, 1]$)
PTB-XL	Yes	[1000, 12, 1]	[3636, 1]	[100, 12, 1]	[936, 1]
MIMIC	Yes	[7500, 2, 1]	[692, 1]	[100, 2, 1]	[122, 1]
Sleep-EDF	No	[3000, 3, 1]	[797, 1]	-	-

8.1. PTB-XL

PTB-XL dataset consists of 12-leads ECG signals with annotated diagnostics. The dataset includes 21837 recordings of 12-lead ECG signals from 18885 patients. All recordings are 10 seconds in length and include metadata containing information about the subject's age, sex, weight, and height. The raw data were annotated by up to two cardiologists ([65], [64]). The example of a single data sample taken from the processed dataset is shown in Fig. 8.1. The task associated with this dataset is to classify the diagnostic statement using the 12-lead ECG signal. The diagnostic statement consists of 5 classes (counts in the brackets): *NORM* - normal ECG (9083), *MI* - Myocardial Infarction (2538), *STTC* - ST/T Change (2406), *CD* - Conduction Disturbance (1709) and *HYP* - Hypertrophy (536).

Samples chosen for further model training must follow two rules:

- diagnostic class statement is available (not null),
- only one diagnostic class statement is present (no multiple labels).

After filtering the dataset by those two rules, 16272 data samples were left. Samples were then split into the train, val and test splits according to folds proposed by the authors. The class distribution of each split is shown in Fig. 8.2. Class distributions seen on Fig. 8.2 indicate class imbalance occurring in the PTB-XL dataset.

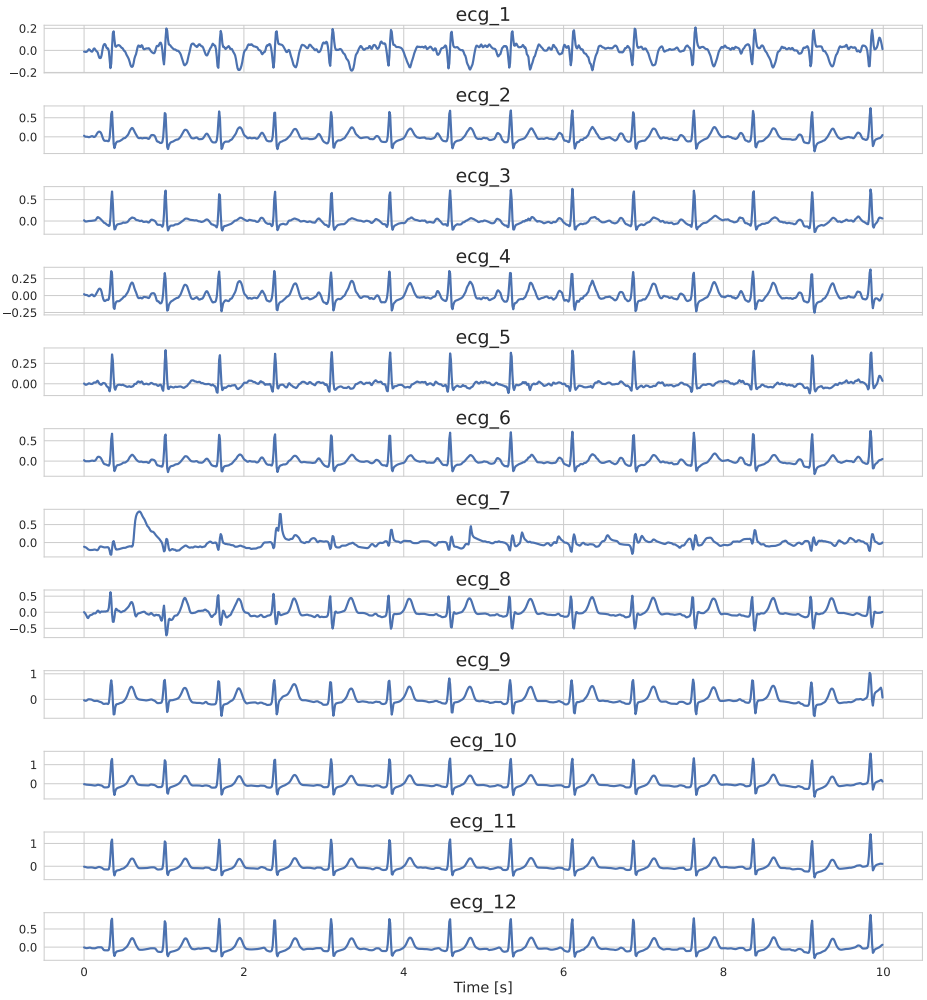


Figure 8.1: The 12-lead ECG signals from PTB-XL dataset.

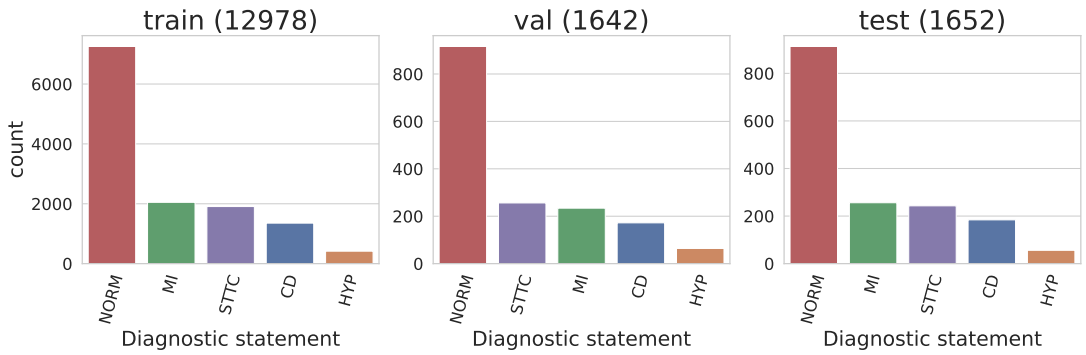


Figure 8.2: PTB-XL dataset classes statistics. The number in the brackets indicates the size of each split.

8.2. MIMIC-III Waveform Database

MIMIC-III consists of physiological signals and time series of vital parameters collected from bedside monitors of patients in adult and neonatal intensive care units. The dataset includes 67830 recordings from about 30000 patients. The length of the recordings is not strictly defined, with some reaching lengths of up to several weeks. These recordings contain data on blood pressure (ABP), electrocardiogram (ECG), respiration, photoplethysmogram (PPG), and many other medical signals. A subset of this database was used in this work (*MIMIC-III Waveform Database Matched Subset*), which additionally underwent a selection and processing process [40], [28]. The example of a single data sample taken from the processed dataset is shown in Fig. 8.3. Many samples from the MIMIC database are of low quality, which brings the need to filter them out. Samples chosen for further model training proceeded with the following filtering pipeline (according to [2]):

1. filter out measurements that don't have any of ABP, PPG, or ECG (lead II) signals (all three signals must be available),
2. find valid segments, i.e.:
    1. filter out measurements with a number of samples smaller than 30000 (fs=125Hz, so it is equivalent to 240s),
    2. check all three signals (ABP, PPG, ECG) for the following conditions and filter out those which do not meet any:
        - ABP: maximum number of repeated samples is less than 10,
        - ECG: maximum number of repeated samples is less than 10,
        - PPG: maximum number of repeated samples is less than 10,
3. cut valid segments into 30000 samples length measurements (240s),
4. trim each measurement to 7500 samples (60s),
5. for each subject randomly select a maximum of 8 processed measurements.

After the filtering pipeline, 20842 data samples were left. Samples were then split into the train, val, and test splits (ratio 0.7/0.15/0.15) using grouped split. The use of grouped split ensured that all samples belonging to one patient are in the same split. Blood pressure targets distribution

of each split is shown in Fig. 8.4. The task associated with this dataset is to predict averaged systolic blood pressure values (SBP) using the PPG and ECG signals.

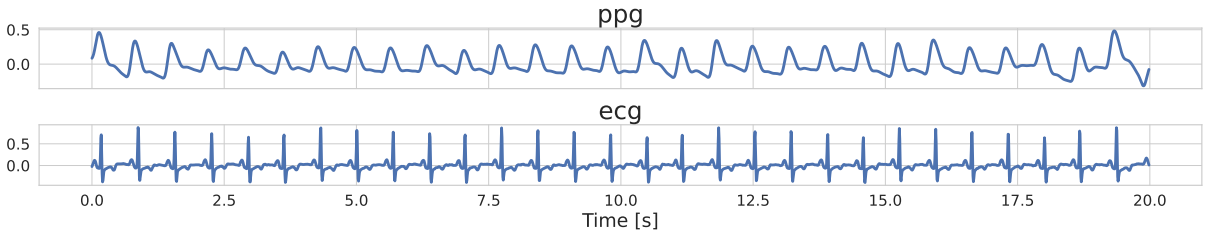


Figure 8.3: The PPG and ECG signals from MIMIC dataset

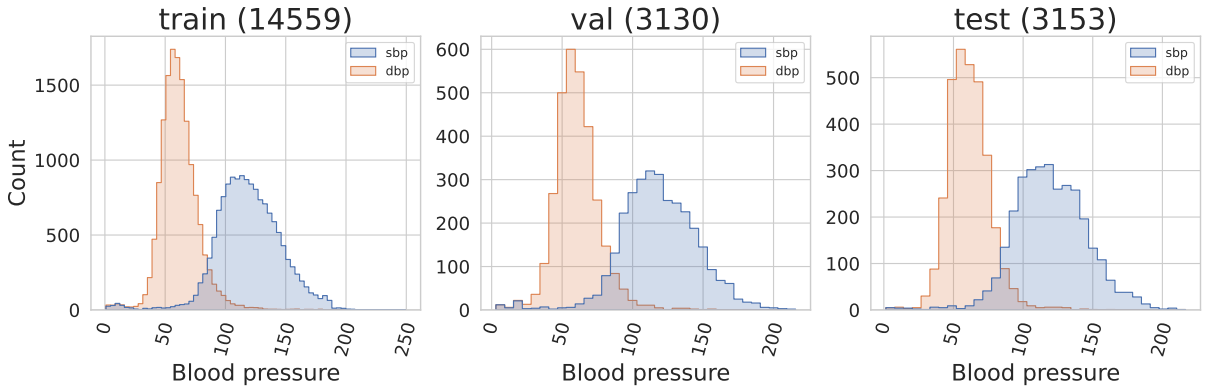


Figure 8.4: MIMIC dataset blood pressure targets statistics (sbp – systolic blood pressure, dbp – diastolic blood pressure). The number in the brackets indicates the size of each split.

8.3. Sleep-EDF

The sleep-EDF dataset consists of physiological signals with annotations of sleep phases. The dataset includes 197 all-night sleep recordings. These recordings include EEG, EOG, and EMG signals and are manually annotated by trained technicians [32]. The example of a single data sample taken from the processed dataset is shown in Fig. 8.5. Each all-night recording was split into 30-second epochs, which was treated as single data sample (according to [41], [33] and [38]). Samples were then split into the train, val, and test splits (ratios 0.7/0.15/0.15) using grouped split. The use of grouped split ensured that all samples belonging to one patient are in the same split. The class distribution of each split is shown in Fig. 8.6. As seen in the Fig. 8.6, Sleep-EDF classes are imbalanced. The task associated with this dataset is to classify the sleep stage using one EOG and two EEG signals. The possible sleep stages classes are (counts in the brackets): *Sleep stage W* (66822), *Sleep stage R* (25835), *Sleep stage 1* (21522), *Sleep stage 2* (69132), *Sleep stage 3* (8793) and *Sleep stage 4* (4246).

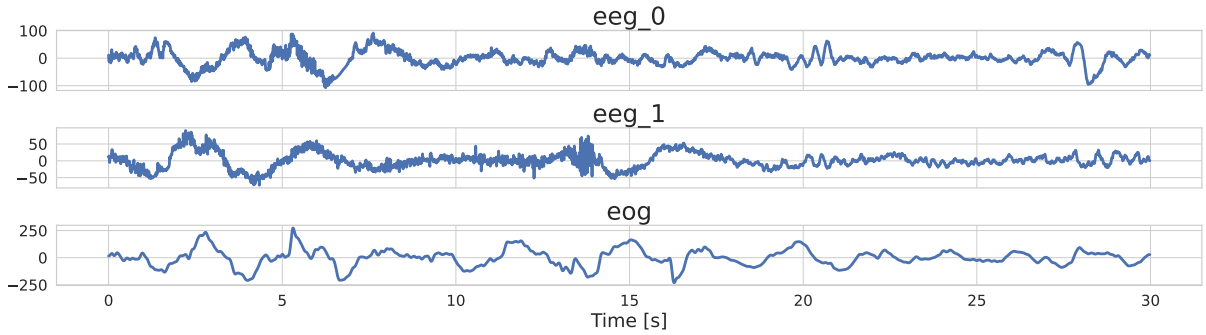


Figure 8.5: The EOG and 2-lead EEG signals from Sleep-EDF dataset

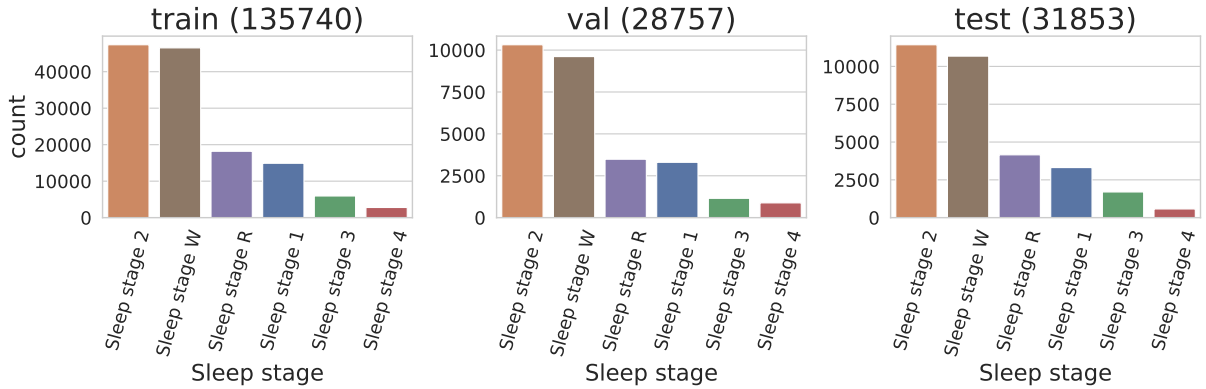


Figure 8.6: Sleep-EDF dataset classes statistics. The number in the brackets indicates the size of each split.

Chapter 9

Experiments

First, each model was subjected to the hyper-parameters optimization process for different representation types (section 9.1) and then using the carefully chosen sets of hyper-parameters, the final experiments were conducted (section 9.2). For simplicity, the following abbreviations have been adopted for the representation types and models, WSW – whole signal waveforms, WSF – whole signal features, ABW – aggregated beat waveforms, ABF – aggregated beat features, Reg – Regression, LogR – Logistic Regression, LinR – Linear Regression, DT – Decision Tree, LGBM – Light Gradient Boosting Machine, MLP – Multilayer Perceptron, CNN – Convolutional Neural Network.

9.1. Experiment 1: Hyper-parameters optimization

The purpose of this experiment is to find the best hyper-parameter settings for each model depending on the medical signal representation type used. Hyper-parameters optimization process follows the procedure presented in section 7.5. In the following sections, there is presented a hyper-parameters optimization order along with the possible values of each of the optimized hyper-parameters. Tables 9.2 (LogR), 9.1 (DT), 9.3 (LGBM), 9.4 (MLP), 9.5 (CNN) shows the results of the hyper-parameters optimization process, i.e. the hyper-parameters settings for each model type and representation type. The linear Regression model (LinR) was not subjected to an optimization process due to the lack of hyper-parameters requiring optimization. A broad description of hyper-parameters optimization process is presented for DT & ABF variant (Decision Tree model trained with Aggregated Beat Features representation) in section 9.1.1. All other models & representations variants were subjected to an equivalent procedure.

9.1.1. Decision Tree

The order of optimization of the hyper-parameters of the Decision Tree model, along with their possible values, is as follows:

1. max_features $\in \{0.1, 0.3, 0.5, 0.8, 1.0\}$
2. min_samples_split $\in \{2, 4, 8, 16\}$
3. max_depth $\in \{4, 16, 32, 64, -1\}$

4. min_samples_leaf $\in \{1, 2, 4, 8, 16\}$

In the beginning, all hyper-parameters are set to default values, i.e. $max\_features = 1.0$, $min\_samples\_split = 2$, $max\_depth = -1$ and $min\_samples\_leaf = 1$. Next, the first hyper-parameter ($max\_features$) is optimized, while the rest is fixed. The best $max\_features$ parameter value (defined by the highest F_1) is picked and its default value is updated. After the default value update, the other hyper-parameters are optimized using the same procedure. Fig. 9.1 shows an example of a hyper-parameters optimization process conducted for the DT model and the ABF representation. Plots from the left to the right demonstrate the order of hyper-parameters optimization. The x-axis represents the hyper-parameter's tested values. The y-axis represents the score (F_1) that the model (initialized with specific hyper-parameters) obtained on the validation set. The red star indicates, that the new best score (F_1) is found in the current iteration and the hyper-parameter default value is updated. The "OPT" beside the hyper-parameter name in the subplot titles indicates that this hyper-parameter is currently optimized, while other parameters are set to fixed values defined next to their names.

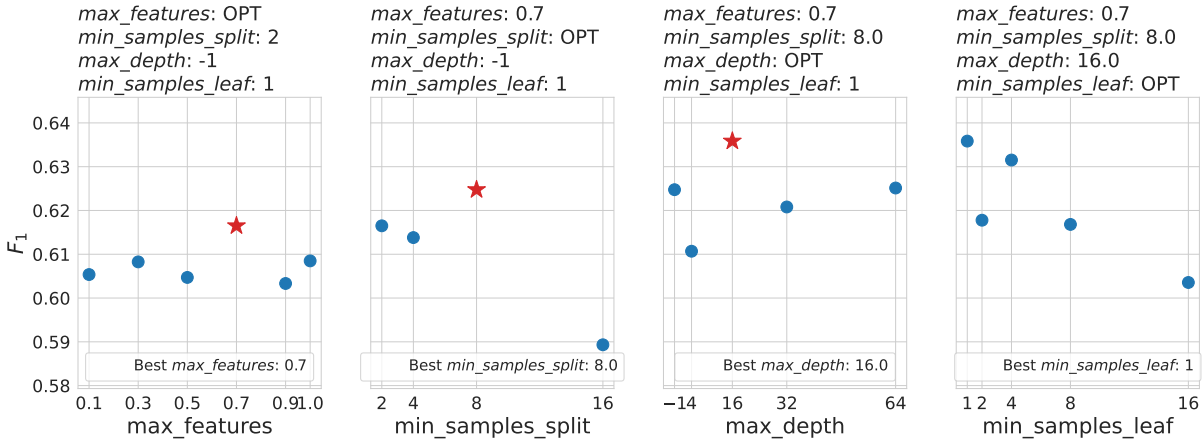


Figure 9.1: Hyper-parameters optimization process for DT & ABF variant. The red star indicates, that the new best score (F_1) is found in the current iteration and the hyper-parameter default value is updated. The "OPT" beside hyper-parameter name in the subplots titles indicates that this hyper-parameter is currently optimized, while other parameters are set to fixed values defined next to their names. The annotation at bottom part of the plots tells which hyper-parameter values are selected as the best.

DT model hyper-parameter optimization process determined the final set of hyper-parameters for each representation type, that is presented in table 9.1.

Table 9.1: Hyper-parameters used for Decision Tree model for different variants of medical signals representation types. Parameters that are not shown here and have been mentioned in the 7.4 table are set to the default values.

Hyper-parameter	**WSW**	**WSF**	**ABW**	**ABF**
max_features	0.7	0.5	0.1	0.7
min_samples_split	2	2	2	8
max_depth	64	16	16	16
min_samples_leaf	1	4	1	1

9.1.2. Logistic Regression

The order of optimization of the hyper-parameters of the Logistic Regression model, along with their possible values, is as follows:

1. C $\in \{0.001, 0.01, 0.1, 1, 10\}$
2. solver $\in \{lbfgs, newton-cg, sag, saga\}$
3. max_iter $\in \{100, 500, 2000, 5000\}$

LogR model hyper-parameter optimization process determined the final set of hyper-parameters for each representation type, that is presented in table 9.2.

Table 9.2: Hyper-parameters used for Logistic Regression model for different variants of medical signals representation types. Hyper-parameters that are not shown here and have been mentioned in the 7.3 table are set to the default values.

Hyper-parameter	**WSW**	**WSF**	**ABW**	**ABF**
C	10.0	0.001	0.1	0.1
solver	lbfgs	newton-cg	sag	newton-cg
max_iter	2000	2000	2000	2000

9.1.3. Light Gradient Boosting Machine

The order of optimization of the hyper-parameters of the Light Gradient Boosting Machine model, along with their possible values, is as follows:

1. colsample_bytree $\in \{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$
2. max_depth $\in \{4, 16, 32, 64, -1\}$
3. num_leaves $\in \{7, 15, 31, 63\}$
4. n_estimators $\in \{100, 250, 600, 1000\}$
5. learning_rate $\in \{0.001, 0.005, 0.01, 0.1\}$

LGBM model hyper-parameter optimization process determined the final set of hyper-parameters for each representation, that is presented in table 9.3.

Table 9.3: Hyper-parameters used for Light Gradient Boosting Machine (LGBM) model for different variants of medical signals representation types. Parameters that are not shown here and have been mentioned in the 7.5 table are set to the default values.

Hyper-parameter	**WSW**	**WSF**	**ABW**	**ABF**
colsample_bytree	0.1	0.1	0.9	0.9
max_depth	4	64	32	32
num_leaves	31	31	31	31
n_estimators	600	600	600	600
learning_rate	0.1	0.1	0.1	0.1

9.1.4. Multilayer Perceptron

The order of optimization of the hyper-parameters of the Multilayer Perceptron model, along with their possible values, is as follows:

1. learning_rate $\in \{0.001, 0.01, 0.05, 0.1\}$
2. hidden_dims $\in \{[512], [256, 512], [512, 256], [256, 512, 256], [128, 256, 512, 256, 128]\}$
3. dropout $\in \{0, 0.1, 0.2, 0.3, 0.4\}$
4. weight_decay $\in \{0.0001, 0.001, 0.01, 0.1\}$

The final set of MLP hyper-parameters for each representation type is presented in table 9.4.

Table 9.4: Hyper-parameters used for Multilayer Perceptron (MLP) model for different variants of medical signals representation types. Parameters that are not shown here and have been mentioned in the 7.6 table are set to the default values.

Hyper-parameter	**WSW**	**WSF**	**ABW**	**ABF**
learning_rate	0.001	0.001	0.01	0.001
hidden_dims	[256,512,256]	[256,512,256]	[512]	[256,512]
dropout	0.2	0.2	0.2	0.2
weight_decay	0.001	0.001	0.0001	0.001

9.1.5. Convolutional Neural Network

The order of optimization of the hyper-parameters of the Convolutional Neural Network model, along with their possible values, is as follows:

1. learning_rate $\in \{0.001, 0.01, 0.1\}$
2. conv0_kernel_size $\in \{5, 7, 9\}$
3. conv0_channels $\in \{64, 128, 256\}$
4. layers $\in \{[1], [1, 1], [1, 1, 1], [1, 1, 1, 1]\}$
5. ff_hidden_dims $\in \{[64], [128], [128, 128], [128, 64]\}$
6. ff_dropout $\in \{0, 0.1, 0.2, 0.3\}$
7. weight_decay $\in \{0.0001, 0.001, 0.01\}$

The CNN model final set of hyper-parameters for each representation type defined by the hyper-parameters optimization process is presented in table 9.5.

Table 9.5: Hyper-parameters used for Convolutional Neural Network (CNN) model for different variants of representation type. Parameters that are not shown here and have been mentioned in the 7.7 table are set to the default values.

Hyper-parameter	WSW	WSF	ABW	ABF
learning_rate	0.001	0.001	0.001	0.001
conv0_kernel_size	9	7	5	9
conv0_channels	256	64	128	256
layers	[1, 1, 1]	[1, 1, 1]	[1, 1, 1]	[1, 1, 1, 1]
ff_hidden_dims	[128, 128]	[128, 128]	[128, 128]	[128, 64]
ff_dropout	0.2	0.2	0.0	0.2
weight_decay	0.001	0.001	0.01	0.001

9.2. Experiment 2: Assessing the impact of representation types on the quality of ML models results

The purpose of the following experiments is to analyze the dependence between the quality of the results and the used medical signal representation type for each ML model. Section 9.2.1 shows how the results changed when the ML model type was fixed and different representation types were used. Section 9.2.2 shows how the results changed when the representation type was fixed and different ML models were used. Each model was trained using the best hyper-parameter settings obtained from the optimization process. The precise experiments results obtained for each dataset are presented in A Appendix in tables A.1 (PTB-XL), A.2 (Sleep-EDF) and A.3 (MIMIC).

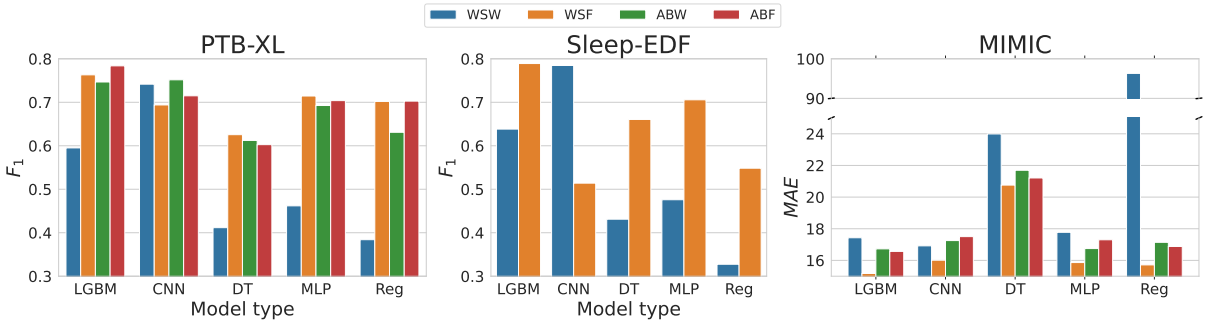
9.2.1. Models perspective

The analysis of results quality from the models perspective, that is when ML model types were fixed and different representation types were used. The experiment results are presented in Fig. 9.2. For each dataset, the LGBM, DT, MLP, and Reg models earned much better prediction quality results for features-based representations than for waveform-based representations. All of these models obtained the worst results for the WSW representation. For two out of three datasets, the CNN model's prediction quality results were better for waveforms-based representations than for features-based. It is worth noting, that in all cases, the ABW representation obtained similar prediction quality results to the features-based representations.

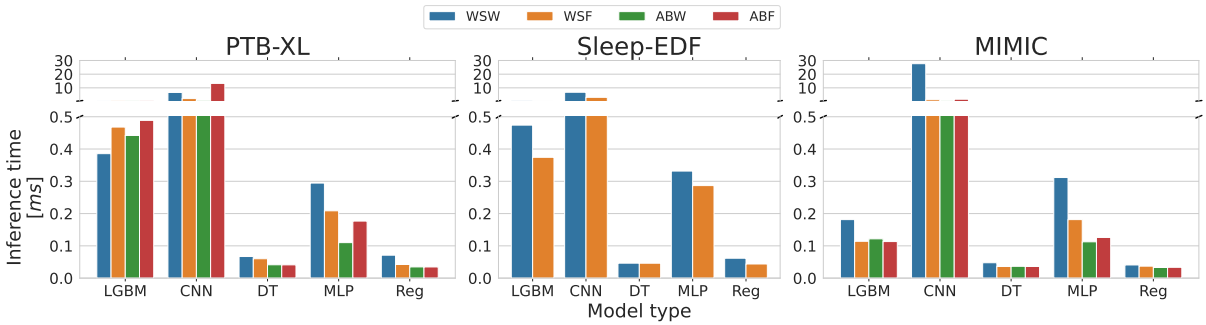
The inference time mainly depended on the type of ML model used. However, a subtle effect of representation type on the inference time of ML models was observed, in particular for models that use a 1D vector representation (LGBM, DT, MLP, Reg) as input. The inference time for the WSW representation was the longest in almost all cases and the shortest for the ABW representation.

The amount of disk space needed is dependent only on the model type and hyper-parameters chosen. The heaviest are the MLP and CNN models and the lightest are the DT and Reg models.

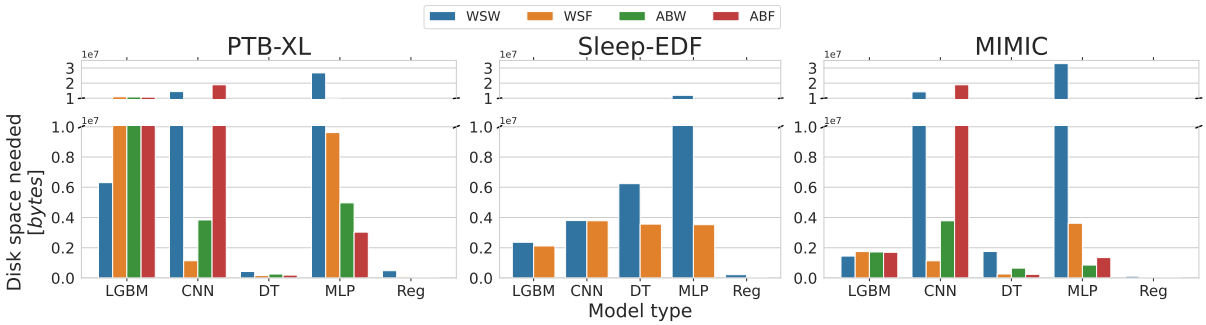
The representations that obtained the best prediction quality results for specific models and datasets are shown in table 9.6. For each dataset, the LGBM, DT, MLP, and Reg models earned the best prediction quality results for features-based representations (WSF mostly). For the CNN model, the best representation types varied depending on the dataset, however, it was mostly the waveforms-based representations.



(a) Predictions quality evaluated on the test set



(b) Models inference time



(c) The amount of disk space needed to store the models

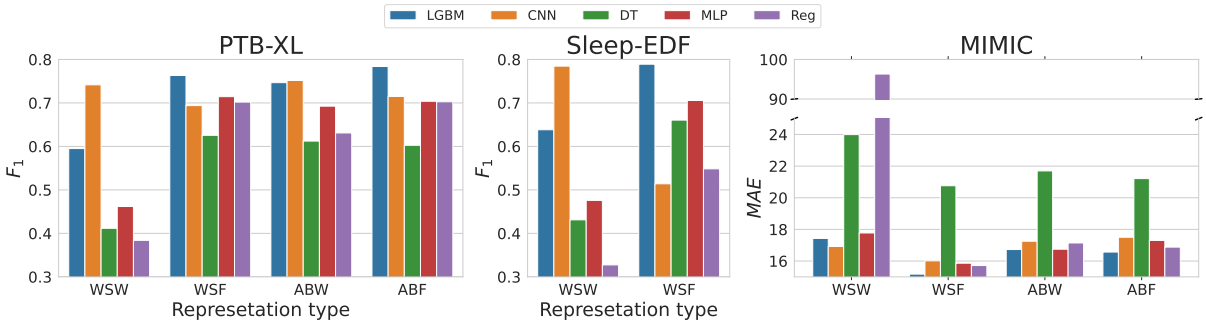
Figure 9.2: Experiments results, models perspective. Each color is defined for a specific representation type: blue for Whole Signal Waveforms (WSW), orange for Whole Signal Features (WSF), green for Aggregated Beat Waveforms (ABW) and red for Aggregated Bead Features (ABF).

Table 9.6: Medical signals representation types with the highest evaluation score for each ML model type and dataset

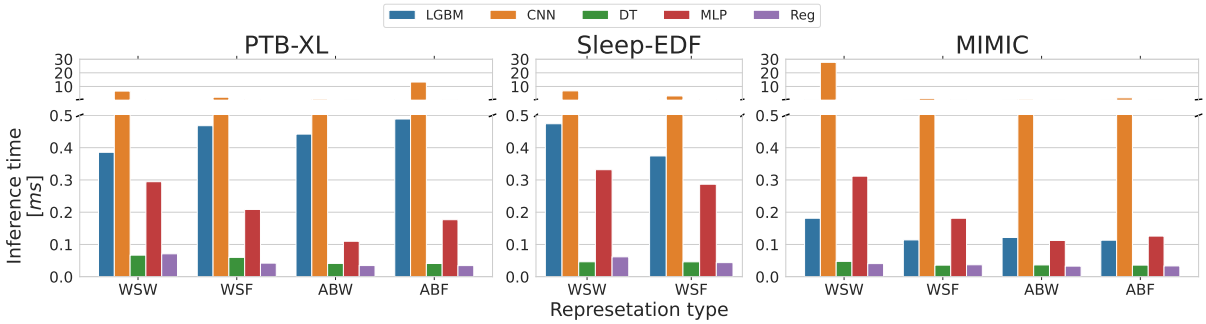
Model	Dataset		
	PTB-XL	Sleep-EDF	MIMIC
Reg	ABF	WSF	WSF
DT	WSF	WSF	WSF
LGBM	ABF	WSF	WSF
MLP	WSF	WSF	WSF
CNN	ABW	WSW	WSF

9.2.2. Representations perspective

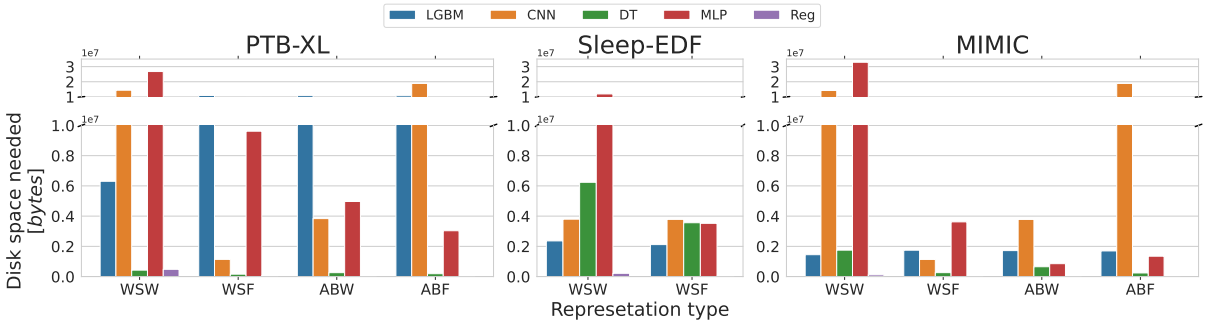
The analysis of results quality from the representations perspective, that is when medical signals representation types were fixed and different ML model types were used.



(a) Predictions quality evaluated on the test set



(b) Models inference time



(c) The amount of disk space needed to store the models

Figure 9.3: Experiments results, representations perspective.

For each dataset, the WSW representation earned the best results when the CNN model was used. It is also worth noting, that the results obtained by other models for this type of representation are much worse than the CNN model's results. For each dataset, the WSF representation obtained the best prediction quality results when the LGBM model was used. Also, the DT and Reg models obtained the worst results for this representation type. The ABW representation earned the best results for the CNN and LGBM models and the worst results for the DT model. The results obtained for the ABF representation are the best when the LGBM model is used and the worst for the DT model.

For each dataset and representation type, the inference time was the longest when the CNN model was used. The second worst was (in almost all cases) the LGBM model. DT and Reg models got the best inference time for every representation type considered.

There is no clear dependence on the amount of disk space needed to store the model and the representation type used. This confirms the observations about disk space needed from the model's perspective.

The models that obtained the best prediction quality results for specific representations and datasets are shown in table 9.7. For each dataset, the WSW representation obtained the best prediction quality results when the CNN model was used. In the case of the ABW representation, the CNN model earned the best results for the PTB-XL dataset, and the LGBM model earned the best results for the MIMIC dataset. In all other cases, the LGBM model obtained the best results in terms of prediction quality.

Table 9.7: ML model types with the highest evaluation score for each medical signals representation type and dataset

	Dataset		
Representation	PTB-XL	Sleep-EDF	MIMIC
WSW	CNN	CNN	CNN
WSF	LGBM	LGBM	LGBM
ABW	CNN	-	LGBM
ABF	LGBM	-	LGBM

9.2.3. Best variants

Out of all the combinations of model types and representation types analyzed, the top-5 variants were selected for each dataset (table 9.8). The LGBM & WSF combination yields the best prediction quality results in most cases. Also, the CNN & WSW and MLP & WSF combinations have repeated as among the best ones.

Table 9.8: Top-5 best ML model types (*Model*) and medical signals representation types (*Rep*) variants for each dataset.

	PTB-XL			**Sleep-EDF**			**MIMIC**		
#	Model	Rep	F_1	Model	Rep	F_1	Model	Rep	MAE
1	LGBM	ABF	0.78	LGBM	WSF	0.79	LGBM	WSF	15.17
2	LGBM	WSF	0.76	CNN	WSW	0.78	Reg	WSF	15.71
3	CNN	ABW	0.75	MLP	WSF	0.71	MLP	WSF	15.86
4	LGBM	ABW	0.75	DT	WSF	0.66	CNN	WSF	16.01
5	CNN	WSW	0.74	LGBM	WSW	0.64	LGBM	ABF	16.57

Chapter 10

Summary

The purpose of this study was to analyze the impact of medical signal representation on the quality of the results of ML models. To achieve this goal, experiments were conducted to evaluate the quality of ML models trained with different variants of medical signals representations. Among all the medical signals representation types and ML model types analyzed, combinations of the Light Gradient Boosting Machine (LGBM) model with the Whole Signal Features (WSF) and Aggregated Beat Features (ABF) representations yield the best results in terms of prediction quality. Also, both inference time and the amount of disk space needed to store the model file for the LGBM model are sufficiently good. The LGBM & WSF was top-1 for two out of three datasets, therefore it is the most recommended approach. The WSF representation consists of features extracted from medical signals, so it is an easy representation to interpret. Also, the LGBM model is tree-based, so its inference is also simple to interpret. These are very big advantages considering that interpretability is a very important (often decisive) factor for AI applications in medicine.

The Convolutional Neural Network (CNN) trained on Whole Signal Waveforms (WSW) scored very well against the rest of the models (top-5 PTB-XL and top-2 Sleep-EDF). This variant is noteworthy since the WSW representation does not require any preprocessing, and the CNN model is a neural network, making it the easiest variant to implement on the end device. The analysis of this variant shows that it is possible to obtain very good results without the need for much expertise in medical signals. Another big advantage of this approach is that it is possible to transfer the knowledge of the CNN model to another task of this type through a finetuning process.

From the model's perspective, the most recommended representation types for the Regression (Reg), Decision Tree (DT), LGBM, and Multilayer Perceptron (MLP) models are the features-based representations, especially the WSF. Furthermore, for all of these models, the Aggregated Beat Waveforms (ABW) yielded very good results as well. It is caused by the fact, that the beats were aligned to the same time location before the aggregation process. Thanks to that, some of the values from waveform representation are strongly related to some of the features extracted from the beat (extended in appendix B). It also shows that it is possible to get very good results without the need for a broad feature extraction process. In the case of the CNN model, the most recommended is to use the Whole Signal Waveforms (WSW) representation. The worst results were obtained for the DT and Reg models, so it is not recommended to use them for any

of the representation types analyzed. It appears that these types of models are unable to learn the complex patterns that occur in medical signals.

It is also worth mentioning that the ABW and ABF representations proposed in this work have repeatedly appeared in the top-5 best variants, making them worthy of consideration for future research. They also have the added advantage of being very small in size (as shown in table 8.2), which allows models using them to be lightweight and fast.

From the perspective of the representation, the most recommended model type for the WSW representation is the CNN model, and for the WSF, ABW, and ABF representations is the LGBM model. One can also achieve good results with the use of the CNN model trained using the ABW representation. The superiority of the CNN model for the raw waveform-based representation could be due to the CNN's ability to capture temporal dependencies through the use of convolution filters. Gradient Boosting models like LGBM perform very well on tabular features, which explains the high quality results obtained by the LGBM model for the feature-based representations.

In conclusion, if interpretability and the best possible prediction quality are most critical, and the cost of feature extraction is negligible, LGBM & WSF is the most recommended approach. On the other hand, if the simplicity of implementation and the ability to use the solution on the end device are important, at the cost of slightly inferior performance, the CNN & WSW approach is recommended. The goal of the study was achieved - the dependence between the quality of ML model results and medical signals representation types was analyzed and conclusions have been made. Also, the combinations of models and representations that are recommended to use in the specific cases were selected.

Due to limited time, the study was conducted for a limited number of representations and ML model types. Further research related to this topic could consider expanding the set of possible models and representations. Representations that would also be worth exploring include beats and windows waveforms, beats and windows features, and CWT image. Models that would also be worth exploring include LSTM neural networks or a combination of CNN + LSTM.

A side-effect of this work is a framework created to analyze, extract features, and obtain different representations from medical signals. That kind of framework may be essential for future research related to the analysis of medical signal representations. The current implementation supports medical signals such as ECG, PPG, EEG, and EOG.

The code needed to run all experiments is stored in the github repository. It is possible to analyze each experiment along with additional plots (feature importance, ROC curve, confusion matrix, etc.) by entering *wandb* public projects associated with this research. Hyper-parameter optimization process is logged in hyper-parameter optimization project and the final experiments are logged in experiments_1 and experiments_2 projects.

Appendix A

Evaluation results

Tables A.1, A.2 and A.3 show results of all experiments.

Table A.1: PTB-XL experiments results

Model type	**Representation type**	F_1	**Inference time** [ms]	**Disk space needed** [$bytes$]
CNN	ABF	0.715	13.242	18M
	ABW	0.752	0.831	3M
	WSF	0.694	2.135	1M
	WSW	0.742	6.622	14M
DT	ABF	0.603	0.041	190K
	ABW	0.612	0.041	260K
	WSF	0.625	0.060	160K
	WSW	0.412	0.067	427K
LGBM	ABF	0.784	0.489	10M
	ABW	0.747	0.442	10M
	WSF	0.763	0.468	10M
	WSW	0.595	0.386	6M
MLP	ABF	0.704	0.177	3M
	ABW	0.693	0.110	4M
	WSF	0.714	0.209	9M
	WSW	0.462	0.294	26M
Reg	ABF	0.703	0.035	19K
	ABW	0.631	0.035	24K
	WSF	0.702	0.042	73K
	WSW	0.384	0.071	480K

Table A.2: Sleep-EDF experiments results

Model type	**Representation type**	F_1	**Inference time** [ms]	**Disk space needed** [$bytes$]
CNN	WSF	0.514	3.051	3M
	WSW	0.785	6.768	3M
DT	WSF	0.661	0.046	3M
	WSW	0.431	0.046	6M
LGBM	WSF	0.789	0.374	2M
	WSW	0.638	0.474	2M
MLP	WSF	0.706	0.287	3M
	WSW	0.476	0.332	11M
Reg	WSF	0.548	0.044	19K
	WSW	0.327	0.062	216K

Table A.3: MIMIC experiments results

Model type	**Representation type**	MAE	**Inference time** [ms]	**Disk space needed** [$bytes$]
CNN	ABF	17.502	1.662	18M
	ABW	17.252	0.795	3M
	WSF	16.012	1.274	1M
	WSW	16.918	27.652	14M
DT	ABF	21.209	0.036	240K
	ABW	21.696	0.037	651K
	WSF	20.758	0.036	261K
	WSW	23.990	0.048	1M
LGBM	ABF	16.568	0.113	1M
	ABW	16.727	0.122	1M
	WSF	15.172	0.114	1M
	WSW	17.429	0.181	1M
MLP	ABF	17.301	0.126	1M
	ABW	16.747	0.112	854K
	WSF	15.863	0.181	3M
	WSW	17.774	0.312	32M
Reg	ABF	16.880	0.033	1K
	ABW	17.141	0.033	2K
	WSF	15.715	0.037	6K
	WSW	96.310	0.041	115K

Appendix B

Feature importance

This section compares Aggregated Beat Waveforms (ABW) and Aggregated Beat Features (ABF) representations, namely the feature importances of LGBM models trained using these representations. The selected models are from experiments on the PTB-XL dataset. Fig B.1 shows the feature importances of LGBM models for ABF (left) and ABW (right) representations. Full feature names information is available in github repository. Fig. B.2a shows an example of aggregated beat which is the result of averaging whole aggregated beat waveform dataset. Also, the critical points of ECG beat are plotted on the same figure. Also, the ABW feature importances heatmap is plotted (Fig. B.2b) along with ABW data for each ECG lead. The feature importances are normalized for each ECG lead separately to highlight the regions of the signals which were the most important for model predictions. The 1 – 12 ECG leads are numbered 0 – 11 on the figure.

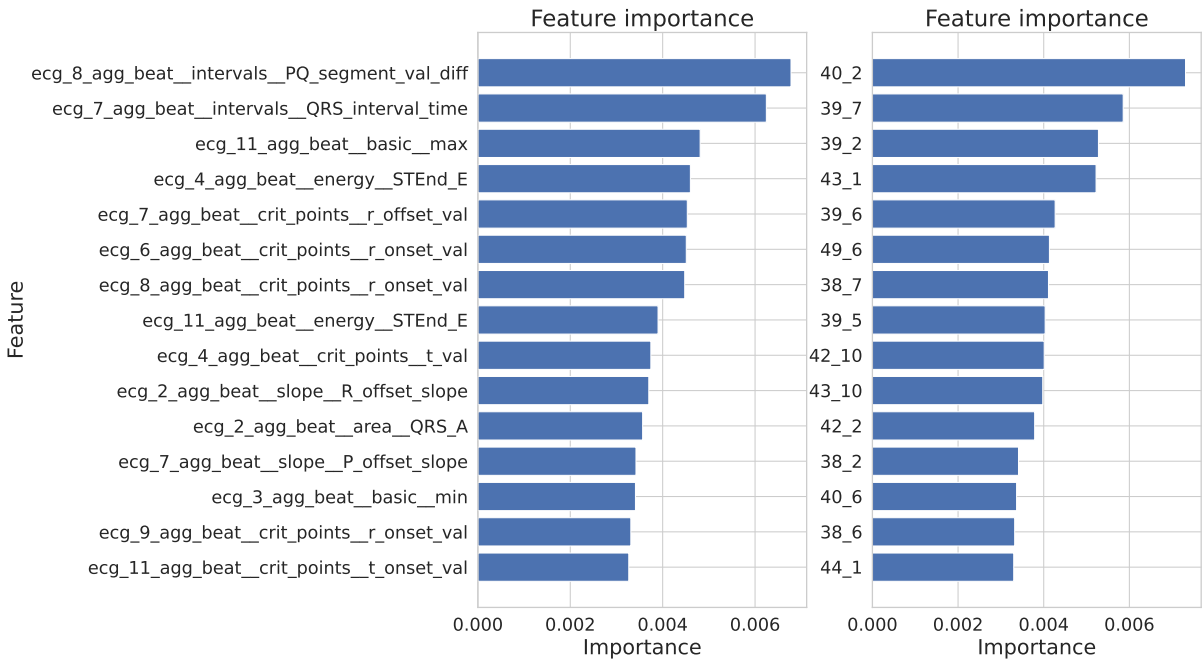
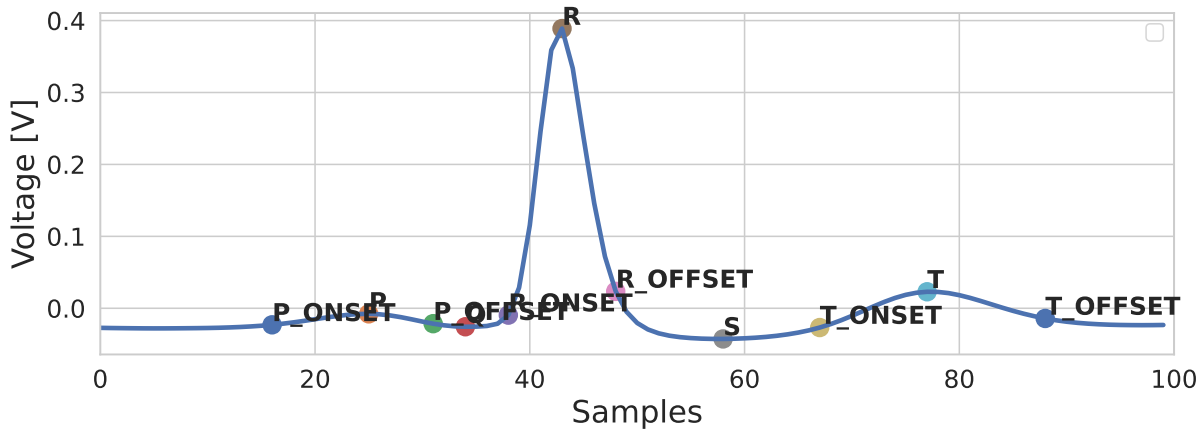
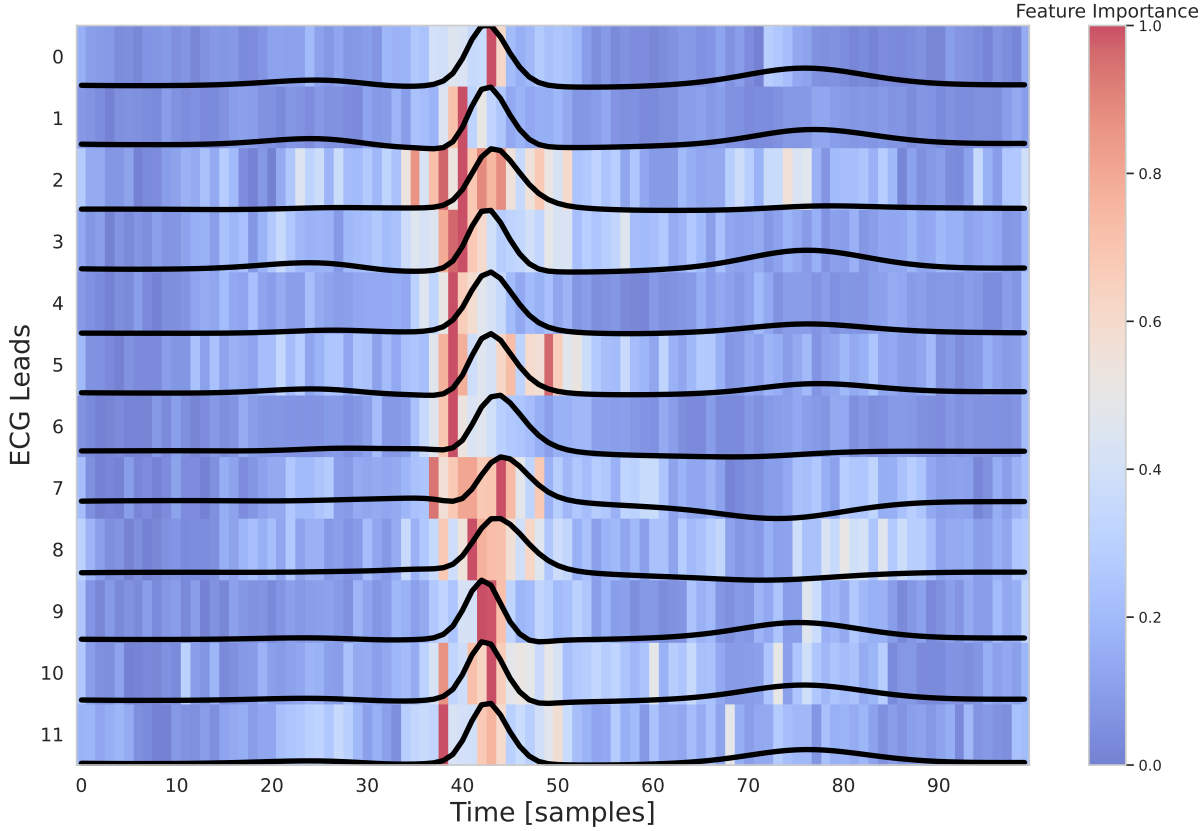


Figure B.1: Feature importances for LGBM model trained on ABF (left) and ABW (right) representations



(a) ECG aggregated beat averaged across ABW dataset



(b) LGBM model feature importances for each ECG lead from ABW representation

Figure B.2: ECG aggregated beats and feature importances for ABW representation

Analysis of all these visualizations shows that although the ABW representation is in the form of a time series, it is very much related to the ABF representation. Fig. B.2b shows the signal locations that were most significant in terms of model prediction, and at the same time signal values for those locations can very easily be interpreted as features (approximately). These are mainly features associated with the QRS complex, for example:

- *R_onset* – high importance for leads: 2, 4, 5, 6, 7, 11, 12
- *R_offset* – high importance for leads: 6
- *R* – high importance for leads: 1, 3, 8, 10

Bibliography

[1] Hafeez Ullah Amin et al. “Classification of EEG Signals Based on Pattern Recognition Approach”. In: *Frontiers in Computational Neuroscience* 11 (2017). ISSN: 1662-5188. DOI: 10.3389/fncom.2017.00103. URL: https://www.frontiersin.org/articles/10.3389/fncom.2017.00103.

[2] Paviglianiti Annunziata et al. “A Comparison of Deep Learning Techniques for Arterial Blood Pressure Prediction.” In: *PubMed* 14 (2022), pp. 1689–1710. DOI: 10.1007/s12559-021-09910-0.

[3] Siriwadee Aungsakul et al. “Evaluating Feature Extraction Methods of Electrooculography (EOG) Signal for Human-Computer Interface”. In: *Procedia Engineering* 32 (Dec. 2012), pp. 246–252. DOI: 10.1016/j.proeng.2012.01.1264.

[4] 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. DOI: 10.1109/ACCESS.2019.2960844.

[5] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[6] Jan C. Brammer. “biopeaks: a graphical user interface for feature extraction from heart- and breathing biosignals”. In: *Journal of Open Source Software* 5.54 (2020), p. 2621. DOI: 10.21105/joss.02621. URL: https://doi.org/10.21105/joss.02621.

[7] cablesandsensors. *12-Lead ECG Placement Guide with Illustrations*. 2023. URL: https://www.cablesandsensors.eu/pages/12-lead-ecg-placement-guide-with-illustrations.

[8] P. de Chazal, B.G. Celler, and R.B. Reilly. “Using wavelet coefficients for the classification of the electrocardiogram”. In: *Proceedings of the 22nd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (Cat. No.00CH37143)*. Vol. 1. 2000, 64–67 vol.1. DOI: 10.1109/IEMBS.2000.900669.

[9] Tsai-Min Chen et al. “Detection and Classification of Cardiac Arrhythmias by a Challenge-Best Deep Learning Neural Network Model”. In: *iScience* 23.3 (2020), p. 100886. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2020.100886. URL: https://www.sciencedirect.com/science/article/pii/S2589004220300705.

[10] Dang Chuangyin, Mala S., and Latha K. "Feature Selection in Classification of Eye Movements Using Electrooculography for Activity Recognition". In: *Computational and Mathematical Methods in Medicine* (2014). DOI: 10.1155/2014/713818. URL: https://doi.org/10.1155/2014/713818.

[11] Shreyasi Datta et al. "Identifying normal, AF and other abnormal ECG rhythms using a cascaded binary classifier". In: *2017 Computing in Cardiology (CinC)*. 2017, pp. 1–4. DOI: 10.22489/CinC.2017.173-154.

[12] Aykut Diker et al. "A novel ECG signal classification method using DEA-ELM". In: *Medical Hypotheses* 136 (2020), p. 109515. ISSN: 0306-9877. DOI: https://doi.org/10.1016/j.mehy.2019.109515. URL: https://www.sciencedirect.com/science/article/pii/S0306987719312381.

[13] Yasser Djawad et al. "Essential Feature Extraction of Photoplethysmography Signal of Men and Women in Their 20s". In: *Engineering Journal* 21 (July 2017), pp. 259–272. DOI: 10.4186/ej.2017.21.4.259.

[14] Mohamed Elgendi et al. "Systolic Peak Detection in Acceleration Photoplethysmograms Measured from Emergency Responders in Tropical Conditions". In: *PLOS ONE* 8.10 (Oct. 2013), pp. 1–11. DOI: 10.1371/journal.pone.0076585. URL: https://doi.org/10.1371/journal.pone.0076585.

[15] Shaffer F and Ginsberg JP. "An Overview of Heart Rate Variability Metrics and Norms." In: *Front Public Health* (2017). DOI: 10.3389/fpubh.2017.00258. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5624990/.

[16] George Forman and Martin Scholz. "Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement". In: *SIGKDD Explor. Newsl.* 12.1 (Nov. 2010), pp. 49–57. ISSN: 1931-0145. DOI: 10.1145/1882471.1882479. URL: https://doi.org/10.1145/1882471.1882479.

[17] Adam Gacek and Witold Pedrycz. *ECG Signal Processing, Classification and Interpretation*. Springer London, 2011. DOI: https://doi.org/10.1007/978-0-85729-868-3. URL: https://link.springer.com/book/10.1007/978-0-85729-868-3.

[18] github. *GitHub*. 2020. URL: https://github.com/.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[20] Serj Haddad, Assim Boukhayma, and Antonino Caizzone. "Continuous PPG-Based Blood Pressure Monitoring Using Multi-Linear Regression". In: *IEEE Journal of Biomedical and Health Informatics* 26.5 (May 2022), pp. 2096–2105. DOI: 10.1109/jbhi.2021.3128229. URL: https://doi.org/10.1109%2Fjbhi.2021.3128229.

[21] Latifa Nabila Harfiya, Ching-Chun Chang, and Yung-Hui Li. "Continuous Blood Pressure Estimation Using Exclusively Photopletysmography by LSTM-Based Signal-to-Signal Translation". In: *Sensors* 21.9 (2021). ISSN: 1424-8220. DOI: 10.3390/s21092952. URL: https://www.mdpi.com/1424-8220/21/9/2952.

[22] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[23] Runnan He et al. "Automatic Cardiac Arrhythmia Classification Using Combination of Deep Residual Network and Bidirectional LSTM". In: *IEEE Access* 7 (2019), pp. 102119–102135. DOI: 10.1109/ACCESS.2019.2931500.

[24] Yan-Cheng Hsu et al. "Generalized Deep Neural Network Model for Cuffless Blood Pressure Estimation with Photoplethysmogram Signal Only". In: *Sensors* 20.19 (2020). ISSN: 1424-8220. DOI: 10.3390/s20195668. URL: https://www.mdpi.com/1424-8220/20/19/5668.

[25] Fatimah Ibrahim et al. *3rd Kuala Lumpur International Conference on Biomedical Engineering 2006*. Springer Berlin, Heidelberg, 2006. ISBN: 978-3-540-68017-8. DOI: https://doi.org/10.1007/978-3-540-68017-8. URL: https://link.springer.com/book/10.1007/978-3-540-68017-8#bibliographic-information.

[26] Nabil Ibtehaz et al. *PPG2ABP: Translating Photoplethysmogram (PPG) Signals to Arterial Blood Pressure (ABP) Waveforms using Fully Convolutional Neural Networks*. 2020. DOI: 10.48550/ARXIV.2005.01669. URL: https://arxiv.org/abs/2005.01669.

[27] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://arxiv.org/abs/1502.03167.

[28] Alistair E.W. Johnson et al. "MIMIC-III, a freely accessible critical care database". In: *Scientific Data* 3 (2016). DOI: 10.1038/sdata.2016.35. URL: https://doi.org/10.1038/sdata.2016.35.

[29] Rishi Vardhan K et al. *BP-Net: Efficient Deep Learning for Continuous Arterial Blood Pressure Estimation using Photoplethysmogram*. 2021. DOI: 10.48550/ARXIV.2111.14558. URL: https://arxiv.org/abs/2111.14558.

[30] Andrej Karpathy. *Convolutional Neural Networks (CNNs / ConvNets)*. 2022. URL: https://cs231n.github.io/convolutional-networks/.

[31] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[32] B. Kemp et al. "Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the EEG". In: *IEEE Transactions on Biomedical Engineering* 47.9 (2000), pp. 1185–1194. DOI: 10.1109/10.867928.

[33] Patrick Krauss et al. "Analysis and visualization of sleep stages based on deep neural networks". In: *Neurobiology of Sleep and Circadian Rhythms* 10 (2021), p. 100064. ISSN: 2451-9944. DOI: https://doi.org/10.1016/j.nbscr.2021.100064. URL: https://www.sciencedirect.com/science/article/pii/S2451994421000055.

[34] Panicos Kyriacou and John Allen. *Photoplethysmography: Technology, Signal Analysis and Applications*. Elsevier, 2021. URL: https://www.elsevier.com/books/photoplethysmography/kyriacou/978-0-12-823374-0.

[35] Dongseok Lee et al. "Beat-to-Beat Continuous Blood Pressure Estimation Using Bidirectional Long Short-Term Memory Network". In: *Sensors* 21.1 (2021). ISSN: 1424-8220. DOI: 10.3390/s21010096. URL: https://www.mdpi.com/1424-8220/21/1/96.

[36] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101.

[37] Dominique Makowski et al. "NeuroKit2: A Python toolbox for neurophysiological signal processing". In: *Behavior Research Methods* 53.4 (2021), pp. 1689–1696. DOI: 10.3758/s13428-020-01516-y. URL: https://neuropsychology.github.io/NeuroKit/index.html.

[38] Alexander Malafeev et al. "Automatic Human Sleep Stage Scoring Using Deep Neural Networks". In: *Frontiers in Neuroscience* 12 (2018). ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00781. URL: https://www.frontiersin.org/articles/10.3389/fnins.2018.00781.

[39] 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.

[40] Benjamin Moody et al. *MIMIC-III Waveform Database*. 2020. URL: https://doi.org/10.13026/c2607m.

[41] Sajad Mousavi, Fatemeh Afghah, and U. Rajendra Acharya. "SleepEEGNet: Automated sleep stage scoring with sequence to sequence deep learning approach". In: *PLOS ONE* 14.5 (May 2019), e0216456. DOI: 10.1371/journal.pone.0216456. URL: https://doi.org/10.1371%2Fjournal.pone.0216456.

[42] Ji Na et al. "EEG Signals Feature Extraction Based on DWT and EMD Combined with Approximate Entropy". In: (2019). DOI: 10.3390/brainsci9080201.

[43] Sebastian Nagel. "Towards a home-use BCI: fast asynchronous control and robust non-control state detection". PhD thesis. Dec. 2019. DOI: 10.15496/publikation-37739.

[44] Avinash Navlani. *Multi-Layer Perceptron Neural Network using Python*. 2021. URL: https://machinelearninggeek.com/multi-layer-perceptron-neural-network-using-python/.

[45] M. S. Oliveira et al. "Methods of EEG Signal Features Extraction Using Linear Analysis in Frequency and Time-Frequency Domains". In: (2014). DOI: 10.1155/2014/730218. URL: https://doi.org/10.1155/2014/730218.

[46] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[47] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[48] Pranav Rajpurkar et al. *Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks*. 2017. DOI: 10.48550/ARXIV.1707.01836. URL: https://arxiv.org/abs/1707.01836.

[49] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning*. Birmingham, UK: Packt Publishing, 2019. URL: https://www.packtpub.com/product/python-machine-learning-third-edition/9781789955750.

[50] Antônio H. Ribeiro et al. “Automatic diagnosis of the 12-lead ECG using a deep neural network”. In: *nature* (2020). DOI: https://doi.org/10.1038/s41467-020-15432-4. URL: https://www.nature.com/articles/s41467-020-15432-4#citeas.

[51] Eduardo José da S. Luz et al. “ECG-based heartbeat classification for arrhythmia detection: A survey”. In: *Computer Methods and Programs in Biomedicine* 127 (2016), pp. 144–164. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2015.12.008. URL: https://www.sciencedirect.com/science/article/pii/S0169260715003314.

[52] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[53] Saeid Sanei and J.A. Chambers. *EEG Signal Processing*. John Wiley Sons, Ltd, 2007. Chap. 1, pp. 1–34. ISBN: 9780470511923. DOI: https://doi.org/10.1002/9780470511923.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470511923.ch1.

[54] Fabian Schrumpf et al. *Assessment of deep learning based blood pressure prediction from PPG and rPPG signals*. 2021. DOI: 10.48550/ARXIV.2104.09313. URL: https://arxiv.org/abs/2104.09313.

[55] Fabian Schrumpf et al. “Assessment of Non-Invasive Blood Pressure Prediction from PPG and rPPG Signals Using Deep Learning”. In: *Sensors* 21.18 (2021). ISSN: 1424-8220. DOI: 10.3390/s21186022. URL: https://www.mdpi.com/1424-8220/21/18/6022.

[56] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[57] Igor Stancin, Mario Cifrek, and Alan Jovic. “A Review of EEG Signal Features and Their Application in Driver Drowsiness Detection Systems”. In: *Sensors* 21.11 (2021). ISSN: 1424-8220. DOI: 10.3390/s21113786. URL: https://www.mdpi.com/1424-8220/21/11/3786.

[58] Mallat Stéphane. “CHAPTER 1 - Sparse Representations”. In: *A Wavelet Tour of Signal Processing (Third Edition)*. Ed. by Mallat Stéphane. Third Edition. Boston: Academic Press, 2009, pp. 1–31. ISBN: 978-0-12-374370-1. DOI: https://doi.org/10.1016/B978-0-12-374370-1.00005-7. URL: https://www.sciencedirect.com/science/article/pii/B9780123743701000057.

[59] Nils Strodthoff et al. “Deep Learning for ECG Analysis: Benchmarks and Insights from PTB-XL”. In: *IEEE Journal of Biomedical and Health Informatics* 25.5 (2021), pp. 1519–1528. DOI: 10.1109/JBHI.2020.3022989.

[60] Akara Supratak et al. “DeepSleepNet: A Model for Automatic Sleep Stage Scoring Based on Raw Single-Channel EEG”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.11 (Nov. 2017), pp. 1998–2008. DOI: 10.1109/tnsre.2017.2721116. URL: https://doi.org/10.1109%2Ftnsre.2017.2721116.

[61] Ali Tazarv and Marco Levorato. *A Deep Learning Approach to Predict Blood Pressure from PPG Signals*. 2021. DOI: 10.48550/ARXIV.2108.00099. URL: https://arxiv.org/abs/2108.00099.

[62] Amin Ullah et al. “Classification of Arrhythmia by Using Deep Learning with 2-D ECG Spectral Image Representation”. In: *Remote Sensing* 12.10 (May 2020), p. 1685. DOI: 10.3390/rs12101685. URL: https://doi.org/10.3390%2Frs12101685.

[63] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[64] Patrick Wagner et al. “PTB-XL, a large publicly available electrocardiography dataset”. In: *Scientific Data* 7 (2020). DOI: 10.1038/s41597-020-0495-6. URL: https://doi.org/10.1038/s41597-020-0495-6.

[65] Patrick Wagner et al. *PTB-XL, a large publicly available electrocardiography dataset*. 2022. URL: https://doi.org/10.13026/kfzx-aw45.

[66] Thaddeus Walczak and Sudhansu Chokroverty. “7 - Electroencephalography, Electromyography and Electrooculography: General Principles and Basic Technology”. In: *Sleep Disorders Medicine*. Ed. by Sudhansu Chokroverty. Butterworth-Heinemann, 1994, pp. 95–117. ISBN: 978-0-7506-9002-7. DOI: https://doi.org/10.1016/B978-0-7506-9002-7.50012-4. URL: https://www.sciencedirect.com/science/article/pii/B9780750690027500124.

[67] Zhiguang Wang, Weizhong Yan, and Tim Oates. *Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline*. 2016. DOI: 10.48550/ARXIV.1611.06455. URL: https://arxiv.org/abs/1611.06455.

[68] Chen Wei et al. “Study on conditioning and feature extraction algorithm of photoplethysmography signal for physiological parameters detection”. In: 4 (2011), pp. 2194–2197. DOI: 10.1109/CISP.2011.6100581.

[69] RF Wireless World. *ECG sensor vs PPG sensor | Difference between ECG sensor and PPG sensor*. 2023. URL: https://www.rfwireless-world.com/Terminology/Difference-between-ECG-sensor-and-PPG-sensor.html.

[70] Omry Yadan. *Hydra - A framework for elegantly configuring complex applications*. 2019. URL: https://github.com/facebookresearch/hydra.

[71] Dongdong Zhang et al. "Interpretable deep learning for automatic diagnosis of 12-lead electrocardiogram". In: *iScience* 24.4 (2021), p. 102373. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2021.102373. URL: https://www.sciencedirect.com/science/article/pii/S2589004221003412.

List of Figures

List of Tables

List of Algorithms