FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Exploring Modifications of Fourier Transform and their Impact on Accuracy of Machine Learning Techniques for ECG Classification
Student:	Bc. Bogdan Buliakov
Supervisor:	Ing. Miroslav Čepek, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2024/2025

Instructions

This thesis aims to explore machine learning techniques and options to process time signals with focus on ECG. The thesis explores impact of modifications of Fourier tranform on performace of machine learning models and will compare the accuracy of models in frequency domain with individual modifications to models working in time domain.

Individual steps:

1) Review machine learning models for time signal classification (partial and whole) and techniques for preprocessing time signals.
2) Describe and explore suitable dataset. For example CORE-15 for ECG Classification.
3) Review Fourier transform, it's modifications and propose your modifications.
4) Create a baseline model and experiment with modified models and document the impact of modifications on accuracy of the model.
5) Compare the results with techniques directly using the time domain data.

Literature:

Carlos Mateo, Juan Antonio Talavera. Short-time Fourier transform with the window size fixed in the frequency domain. Digital Signal Processing. Volume 77. 2018. ISSN

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 16 February 2024 in Prague.

1051-2004. https://doi.org/10.1016/j.dsp.2017.11.003.
Ribeiro, Antônio H., et al. "Automatic diagnosis of the 12-lead ECG using a deep neural network." Nature communications 11.1 (2020): 1760.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 16 February 2024 in Prague.

Master's thesis

EXPLORING MODIFICATIONS OF FOURIER TRANSFORM AND THEIR IMPACT ON ACCURACY OF MACHINE LEARNING TECHNIQUES FOR ECG CLASSIFICATION

Bc. Bogdan Buliakov

Faculty of Information Technology
Department of Knowledge Engineering
Supervisor: Ing. Miroslav Čepek Ph.D.
May 9, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Buliakov Bogdan. *Exploring Modifications of Fourier Transform and their Impact on Accuracy of Machine Learning Techniques for ECG Classification.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

List of code listings

First of all, I would like to thank my supervisor Ing. Miroslav Čepek Ph.D. for his valuable time, his advices and his willingness to adapt to my needs. I thank FIT CVUT IT department for opportunity to use GPU cluster for calculations - this allowed me to conduct much more experiments than would have been available to me at my own computing power. I would also like to say a huge thank you to my wife for her support during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 9, 2024

Abstract

This thesis aims to explore machine learning techniques and options to process time signals with focus on ECG. The thesis explores impact of modifications of Fourier tranform on performace of machine learning models and will compare the accuracy of models in frequency domain with individual modifications to models working in time domain.

Individual steps:

1) Review machine learning models for time signal classification (partial and whole) and techniques for preprocessing time signals. 2) Describe and explore suitable dataset. For example CODE-15 for ECG Classification. 3) Review Fourier transform, it's modifications and propose your modifications. 4) Create a baseline model and experiment with modified models and document the impact of modifications on accuracy of the model. 5) Compare the results with techniques directly using the time domain data.

Literature: Carlos Mateo, Juan Antonio Talavera. Short-time Fourier transform with the window size fixed in the frequency domain. Digital Signal Processing. Volume 77. 2018. ISSN 1051-2004. https://doi.org/10.1016/j.dsp.2017.11.003. Ribeiro, Antônio H., et al. "Automatic diagnosis of the 12-lead ECG using a deep neural network." Nature communications 11.1 (2020): 1760.

Keywords ECG, FFT, CNN, DNN, Preprocessing

Abstrakt

Tato práce si klade za cíl prozkoumat techniky strojového učení a možnosti zpracování časových signálů se zaměřením na EKG. Práce zkoumá vliv modifikací Fourierovy transformace na výkonnost modelů strojového učení a porovná přesnost modelů ve frekvenční oblasti s jednotlivými modifikacemi modelů pracujících v časové oblasti.

Jednotlivé kroky:

1) Přezkoumat modely strojového učení pro klasifikaci časových signálů (částečné a celé) a techniky pro předzpracování časových signálů. 2) Popsat a prozkoumat vhodný soubor dat. Například CODE-15 pro klasifikaci EKG. 3) Projít si Fourierovu transformaci, její úpravy a navrhnout své úpravy. 4) Vytvořit základní model a experimentovat s upravenými modely a zdokumentovat dopad úprav na přesnost modelu. 5) Porovnovat výsledky s technikami přímo pomocí dat v časové oblasti.

Klíčová slova ECG, FFT, CNN, DNN, Předzpracování

List of abbreviations

ECG	electrocardiogram
NN	neural network
RF	random forest
CNN	convolutional neural networks
LSTM	long short-term memory
DNN	deep Neural Networks
Res-Net	residual neural networks
1dAVb	1st degree AV block
RBBB	right bundle branch block
LBBB	left bundle branch block
SB	sinus bradycardia
AF	atrial fibrillation
ST	sinus tachycardia

Introduction

Here and there we are faced with data. And one of complex types of data is time series. It is a track of a certain variable over time. It may be the temperature outside, exchange rate, musical composition or GPS trajectory - there is a lot of information that can be stored in this form. And from this information, like from any other, we want to gain knowledge. There are plenty of ways to process time series. One may want to classify data, predict its future, and even generate some new data if we're talking about music.

This thesis will be limited to considering such specific time series as Electrocardiogram (ECG). Cardiovascular diseases are the leading cause of death worldwide [1]. And ECG is the major tool for their diagnoses. In Chapter 1 I will introduce you to what an ECG is in general terms. Then I will overview other methods for studying ECG in Chapter 2. For the purpose of this work was used the part of large dataset CODE, which will be described in more details in Chapter 3. The thesis will focus on combinations of signal transformations and classifiers of different architecture, which will be described in Chapter 4, including a new method whose authorship I attribute to myself. In the Chapter 5 there will be described implementation of all experiments. Finally, Chapter 6 will contain the comparison of different transformations and classifiers efficiency.

Chapter 1

Introduction in ECG

In this chapter I want to bring you up to date on what an ECG is.

Electrocardiogram is a result of electrocardiography - a technique for recording and studying electric fields generated during the work of the heart. Using electrodes on the limbs and chest, information is collected about changes in the potential difference between different points of the human body.

Table 1.1 ECG leads and its positions

Group	Leads			Position	Heart view
1	6-Limb	3-Bipolar/ Standard	Lead-I	Left arm & right arm	Left side
			Lead-II	Left leg & right arm	Inferior left view
			Lead-III	Left leg & left arm	Inferior right view
		3-Unipolar/ Augmented	aVR	Right arm (+), Left arm (-), Left leg (-)	Upper right side
			aVL	Left arm (+), Right arm (-), Left leg (-)	Upper left side
			aVF	Left leg (+), Right arm (-), Left arm (-)	Inferior wall of heart
2		6-Chest	V1	2-Septum wall	2-Oriented to right ventricle
			V2		
			V3	2-Anterior wall	2-Face inter ventricular septum
			V4		
			V5	2-Lateral wall	2-Face left ventricle anterolaterally
			V6		

Spatial information about the electrical activity of heart can be captured by standard 12 conventional leads (equally divided into limb- and pre-cordial/chest-leads) placed in three orthogonal directions (i.e., right to left, superior to inferior, and anterior to posterior) on human skin/body. Limb- and chest-leads help in recording the potential difference across the frontal- and horizontal-plane, respectively [2]. The six limb leads are equally divided into bipolar/standard (such as: lead-I, lead-II, and lead-III) leads and uni-polar/augmented (aVR, aVL, and aVF) leads. The six uni-polar pre-cordial/chest leads are V1, V2, V3, V4, V5, and V6. The detail description, position, and purpose of the 12-leads are represented precisely in Table 1.1.

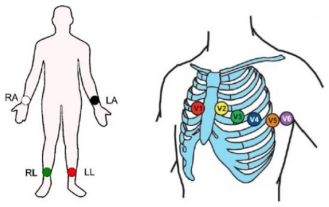


Figure 1.1 Placement of electrodes on the body for a 12-lead ECG [2]

This provides the relation of ECG with orientation of heart.

This is a rather indirect method of obtaining information about the work of the heart, but the ECG carries enough information to determine various disorders even by eye.

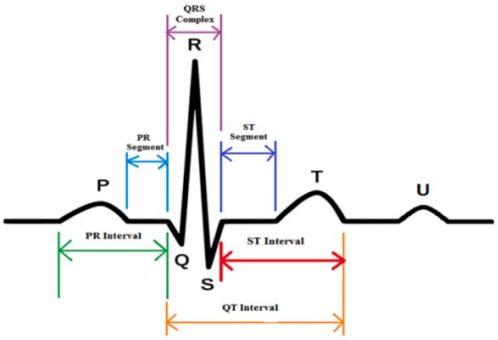


Figure 1.2 ECG morphology: different segments of ECG signal for normal person [3]

Activity of heart and its condition can be known from the morphology (peaks and duration of various peaks) of ECG signal [4]. The main components are P-wave, QRS complex, T-wave, and the duration between them. P-wave occurs due to the sequential activation (i.e., depolarization) of the right and left atria. QRS complex is appeared by right and left ventricular depolarization (i.e., ventricles are activated simultaneously). T-wave arises by ventricular re-polarization, whereas U-wave is occurred after depolarization in the ventricles. Interval between different wave carries diagnostic information about the heart diseases. PR interval produced by time delay from onset of atrial depolarization (P-wave) to onset of ventricular depolarization (QRS complex). QRS period depends on ventricular muscle depolarization, whereas QT interval rely on the duration of ventricular depolarization and re-polarization. The RR- and PP-interval depend on ventricular-cycle and atrial-cycle duration (or rate), respectively.

The morphological specification of an ECG signal of a normal person and the duration of different waveform are shown in Fig. 1.2 and Table 1.2 summarizes the duration, amplitude and the cause of various waveform in normal ECG signal.

Although the ECG is obviously a time series, due to the nature of the signal, its prediction or generation makes no practical sense. On the contrary, classifying signals - or rather patients - is an important task that saves thousands of lives every day. That is why many studies have been conducted on the topic of various methods for automatic classification of the electrocardiogram for the purpose of diagnosis.

Table 1.2 Specification of morphological features in normal ECG.

Waveform	Duration (seconds)	Amplitude (mV)	Remarks
P-wave	0.08–0.12	0.25	Depolarization of LA & RA
Q-wave	0.03	0.2–0.4	Initial ventricular depolarization
R-wave	-	1.60	Depolarization of the ventricles
S-wave	-	1.8–3.0	Final ventricular depolarization
T-wave	0.1–0.25	0.1–0.5	Ventricular repolarization
W-wave	-	0.1–0.33	Purkinje fibers repolarization
PR interval	0.12–0.20	120	Atrial & ventricular depolarization
QR duration	0.06–0.12	2.5–3.0	Depolarization of ventricles
Q interval	0.35–0.44	-	Reflect ventricular repolarization
R interval	0.6–1.2	-	Measures heart rate variability
PP interval	0.60–1.04	-	Interval between two P-waves
ST segment	0.08	0.1–0.2	Early re-polarization

Chapter 2

Overview of methods for ECG analysis

2.1 Classification methods

In this chapter I will introduce you some works related with ECG processing and diagnosing.

There are many ways to process the ECG signal. To overview them I have read the article **"Analysis of various techniques for ECG signal in healthcare, past, present, and future"** [5]. All approaches they divide into three categories:

- conventional/traditional
- machine learning (ML)
- deep learning (DL)

Traditional methods involve extraction and analysis of various ECG features. It may be heart rate, distances between different peaks, entropy, means and variances of any value, correlation coefficients between them. Time, frequency and wavelet based approaches also refers here.

Machine learning methods use hidden Markov model (HMM), support vector machine (SVM), principal component analysis (PCA), linear discriminant analysis (LDA), independent component analysis (ICA) , k-nearest neighbor classifier (KNN), vector quantization, random forest (RF) and other similar techniques.

To Deep learning methods refer Convolutional neural networks (CNN), Long short-term memory (LSTM), Generative adversarial networks (GAN), Deep auto-encoders (DAE), Transformers and Residual neural networks (Res-Net).

A separate category of methods can be considered getting rid of misleading signals and noises. They don't solve the task themselves, but significantly improve the result. Mainly, ECG signals are affected by two forms of noise (high and low frequency). Baseline wander (BW) is coming under low frequency noise. Whereas, power-line interference (PLI), muscle artifact (MA), Gaussian noise, and electromagnetic interference (EMI) etc., lie in high frequency group.

I have also read few articles about interesting methods.

First of all, of course, it was the article **"Automatic diagnosis of the 12-lead ECG using a deep neural network"** [6], that was for me first source of inspiration, the dataset explored and used in the thesis and state-of-the-art Residual Convolution Neural Network architecture. I will describe them in the relevant sections of the work.

I have also found the article **"A Wide and Deep Transformer Neural Network for 12-Lead ECG Classification"** [7], where they used ensemble of Transformer and Dense Network on ECG features to diagnose 27 diseases as part of 2020 PhysioNet/CinC challenge (Fig. 2.1).

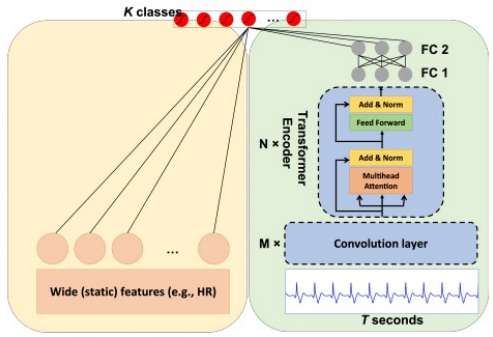


Figure 2.1 Wise and Deep Transformer NN

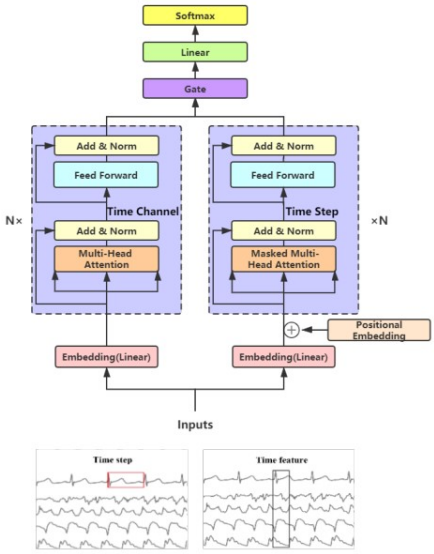


Figure 2.2 Gated Transformer Networks

The ensemble of CNN and LSTM have used authors of the article **"Automated Detection of Abnormalities in ECG signals using Deep Neural Network"** [9]. They used as data for NN different features from peaks - intervals between P and R peaks, widths of them and other similar properties. (Fig. 2.3)

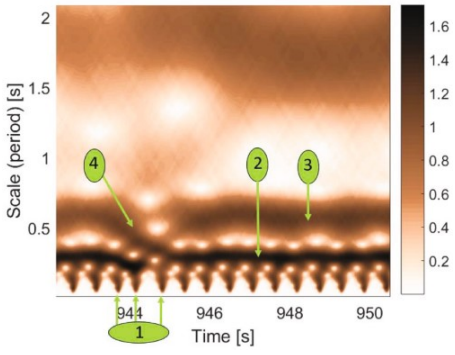


Figure 2.4 STFT-FD of first ECG channel for 15 minute time slot

Another interesting article about transformer usage was **"Gated Transformer Networks for Multivariate Time Series Classification"** [8]. They used there ensemble of two Transformers, first of which processed data along the channels, while the second one processed data of certain moments of time (Fig. 2.2).

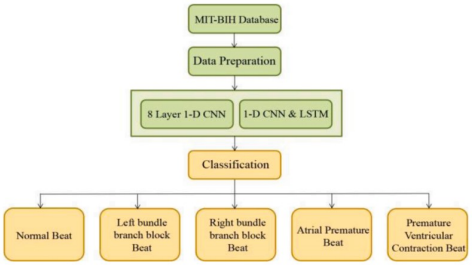


Figure 2.3 Deep NN for Abnormalities detection

In the article **"Analysis of atrial and ventricular premature contractions using the Short Time Fourier Transform with the window size fixed in the frequency domain"** [10] authors use their own Fourier Transform variation **Short-Time Fourier Transform with a Window Size Fixed in the Frequency Domain (STFT-FD)** to diagnose arrhythmia (Fig. 2.4). Their idea is to use for each frequency the count of data, that contains some certain count of cycles. So, different frequencies use different samples count in summation in the Fourier transform formula.

2.2 Anomaly detection

The purpose of the anomaly detection is the same - to identify problems in the patient. However, unlike the previous type of tasks, here it is necessary to process a much longer signal and determine the presence of anomalous areas.

In [11] authors divide the task into rhythm and heartbeat classifications. Rhythm classification determines the type of the rhythm anomalies such as Normal Sinus Rhythm (NSR), Atrial Fibrillation (AF), Ventricular Flutter (VF), and so on. The ECG segments for rhythm classification are usually contained in several heartbeats, which could be normal or abnormal. Heartbeat classification takes an ECG signal that only includes one heartbeat as the input and outputs the heartbeat type such as normal heartbeat, Left/Right bundle branch block beats, premature beats, etc.

In addition to the methods from the previous chapter, signal processing tools such as autoregressive integrated moving average (ARIMA), cumulative sum statistics (CUSUM), exponentially weighted moving average (EWMA) are also used. But modern DNN models as LSTM and Transformers show better efficiency. There also exist methods of comparing the signal with the predicted one. For this purpose, certain generative models and metrics for assessing discrepancies in signals are used. For example in **"Unsupervised Transformer-Based Anomaly Detection in ECG Signals"** [12] they use Transformer to predict ECG next cardiac cycle in real time and calculate its difference with real signal. (Fig. 2.5)

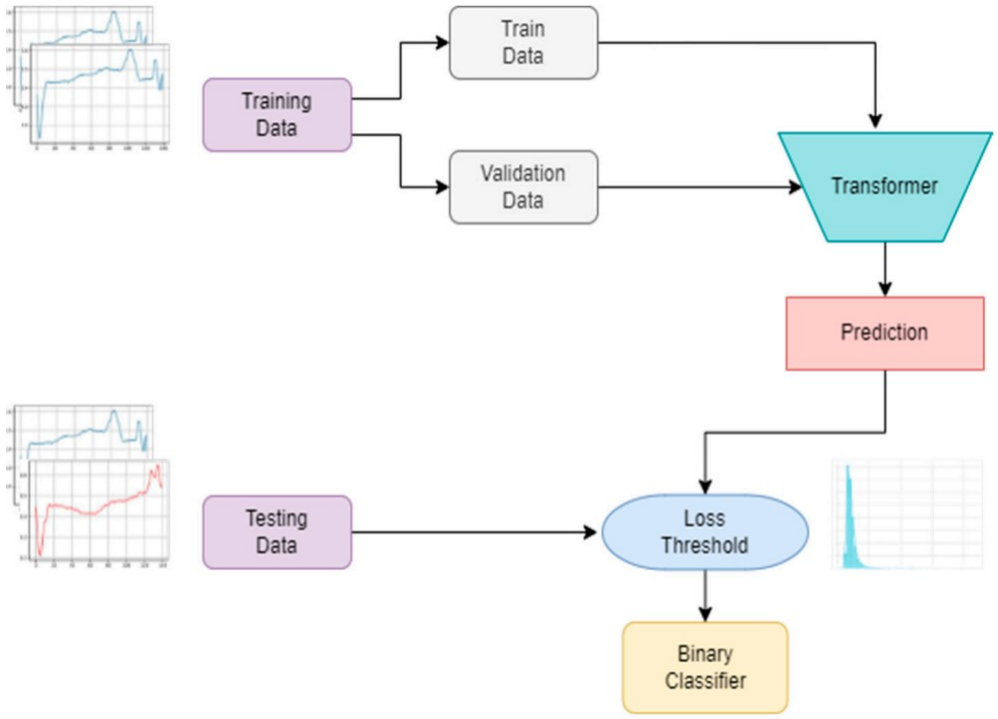


Figure 2.5 Architecture of anomaly detection model

Chapter 3

Overview of dataset CODE-15

In this chapter I introduce you to the CODE dataset, that I have used in this work for experiments.

As a basis for this thesis I took the work [6]. I will call it "CODE article" in this and next chapters. In the article they created their own dataset CODE consisting of 2,322,513 ECG records from 1,676,384 different patients of 811 counties in the state of Minas Gerais/Brazil from the Telehealth Network of Minas Gerais (TNMG). This dataset they have used for training and validation. To evaluate the models, they used a special data set that they call in files for the article **Gold standard**. I will use this name for this dataset in the thesis.

Article authors have used different approaches for labeling these two datasets. For CODE dataset they used 3 source of labels: i) the Uni-G statements and Minnesota codes obtained by the Uni-G automatic analysis (automatic diagnosis); ; ii) automatic measurements provided by the Uni-G software; and, iii) the text labels extracted from the expert text reports of diagnosis using the semi-supervised text processing methodology Lazy Associative Classifier (LAC) [13] (medical diagnosis). They combined these 3 sources to compensate errors of automatic classification, of the practicing expert cardiologists and the labeling methodology. For Gold standard test dataset labeling they collected the opinions of several experts who specifically looked at these ECGs and filled out their findings in an easy-to-process form - Yes/No marks for each disease, rather than a text conclusion. Therefore, it is assumed that the labeling accuracy for this dataset is even higher than for CODE.

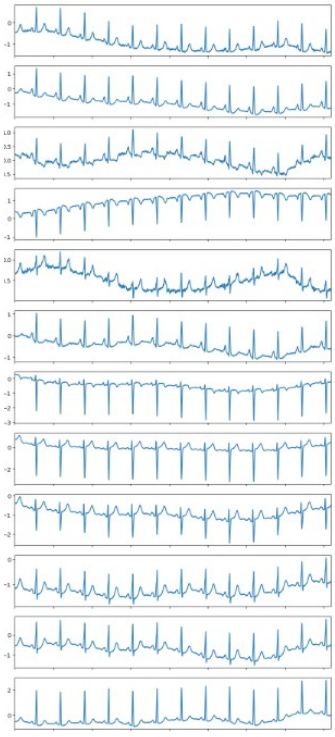


Figure 3.1 ECG example from dataset

As a source they used the short-duration, standard, 12-lead ECG. That signal contains 12 channels of data. All ECG recordings were re-sampled by authors to a 400 Hz sampling rate. The ECG recordings, which have between 7 and 10 seconds, are zero-padded resulting in a signal with 4096 samples for each lead. So, all instances had shape (4096, 12).

Unfortunately dataset CODE is not available for free use and what was more important it is huge. These are the reasons I have used public dataset CODE-15, which is random selection of 15% of patients from CODE. It contains 345,779 instances, and after removing several instances due to failed Fourier transform I got 345,777 instances. I have divided them

to Train, Val and Test datasets by randomly dividing patients into groups in a 70/20/10 ratio. Then I balanced dataset Train by dropping part of healthy instances, so their percent in "Train balanced" dataset became 50% (Table 3.1).

Table 3.1 (Dataset summary) Patient abnormalities prevalence, n (%)

Abnormality	Train (n = 242,015)	Train balanced (n = 52,934)	Val (n = 69,181)	Test (n = 34,581)	Gold standard (n = 827)
1dAVb	3,958 (1.6%)	3,958 (7.5%)	1,180 (1.7%)	578 (1.7%)	28 (3.4%)
RBBB	6,760 (2.8%)	6,760 (12.8%)	1,932 (2.8%)	980 (2.8%)	34 (4.1%)
LBBB	4,246 (1.8%)	4,246 (8.0%)	1,190 (1.7%)	590 (1.7%)	30 (3.6%)
SB	3,960 (1.6%)	3,960 (7.5%)	1,116 (1.6%)	529 (1.5%)	16 (1.9%)
AF	5,316 (2.2%)	5,316 (10.0%)	1,520 (2.2%)	748 (2.2%)	13 (1.6%)
ST	4,884 (2.0%)	4,884 (9.2%)	1,403 (2.0%)	746 (2.2%)	37 (4.5%)
Healthy	215,548 (89.1%)	26,467 (50.0%)	61,650 (89.1%)	30,804 (89.1%)	681 (82.4%)

Authors of CODE article trained a DNN to detect: 1st degree AV block (1dAVb), right bundle branch block (RBBB), left bundle branch block (LBBB), sinus bradycardia (SB), atrial fibrillation (AF) and sinus tachycardia (ST) (Fig. 3.2). My research concerns the same diagnoses.

3.1 Files of dataset

Dataset CODE-15 may be downloaded from WEB page [14] as 18 **.hdf5** files with ECG data and one **.csv** file with labels and patient ID. Total data size in this format is near 47 Gb. The connection between the data and the label was carried out using column *exam_id*, which was not very convenient.

I have processed these files before use. I deleted few instances, which I couldn't process with Fourier transformation. Then I have split all *patients* to train-val-test groups in a ratio of 70-20-10 and have written their ECG instances to different **.hdf5** files - 18 files in every of 3 different directory "train", "val" and "test". After that I used this division in all experiments.

I also have created different files for labels, for I could just ZIP them with data without linking with *exam_id*.

In subsection 5.1.1 I describe, how I work with these files.

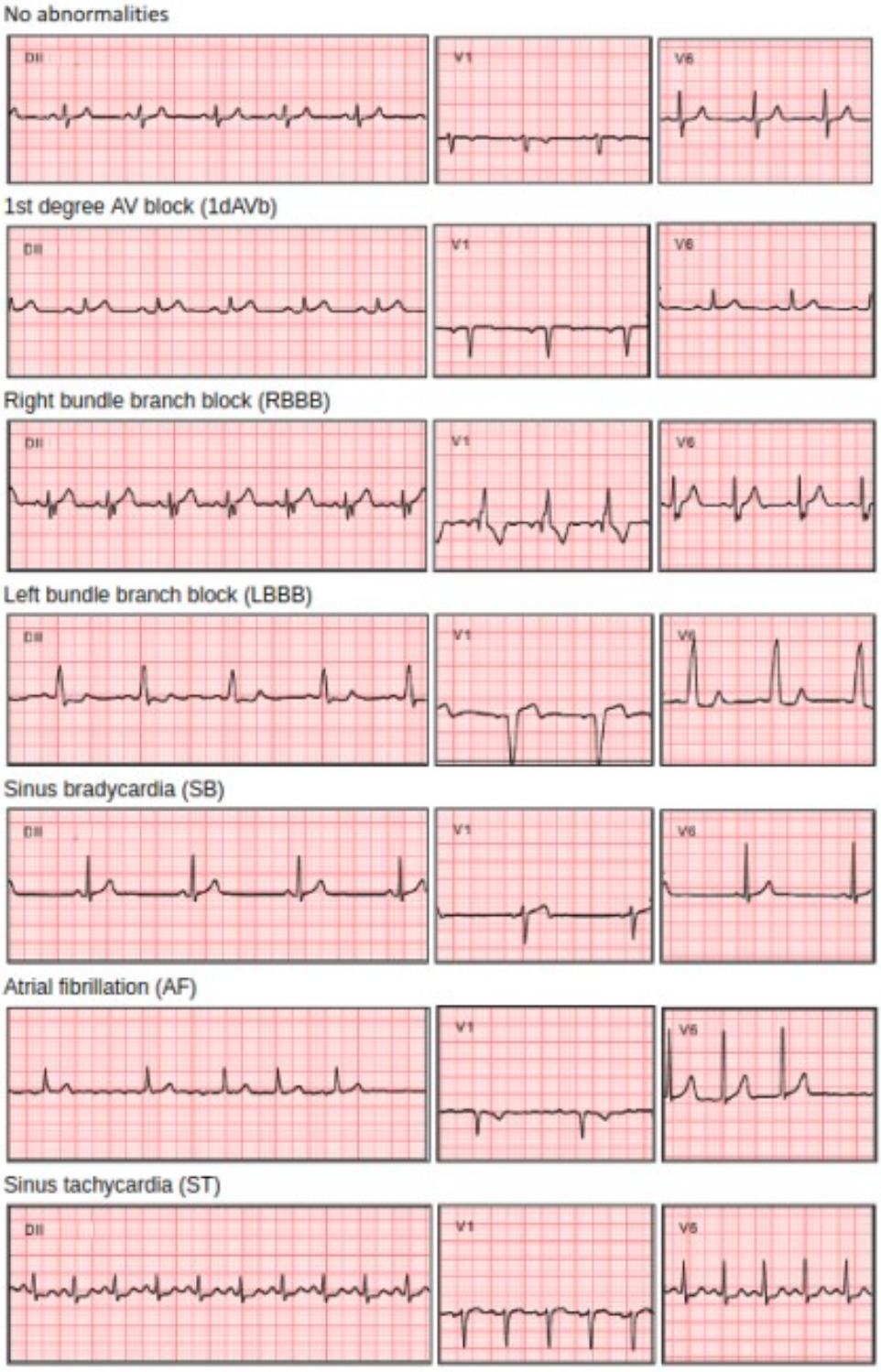


Figure 3.2 (Abnormalities examples)
A list of all the abnormalities the model classifies. Here is 3 representative leads only (DII, V1 and V6).

Chapter 4

Overview of transformations and classifiers in this article

In this chapter I describe few data preprocessing methods and few different classifiers I have used inn this work. I will pay special attention to the transformation that I came up with myself.

4.1 My own FFT transformation (FFTwDW)

This is the transformation variation I have not found information in other articles about. So I think I can call this idea my own.

First of all, what was the inspiration for this idea. It was the fact, that for human cardiologists the scale of ECG graph is not very important. Sure, for some features there are windows of acceptable values in millimeters, but other features (like the form of ECG for healthy patient) they know as template, independent on RR-interval. So I have thought about way of signal "normalization" to get rid of the influence of the heart rate on the ECG shape.

The way to implement this I have found in Fourier Transform.

Since ECG in our task is periodic function, defined on limited interval, it is more convenient for us to process it with Fourier series then with Fourier transform. Let's look at Fourier series formulas:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{i2\pi \frac{n}{P} x} \tag{4.1}$$

$$c_n = \frac{1}{P} \int_{-\frac{P}{2}}^{\frac{P}{2}} f(x) e^{-i2\pi \frac{n}{P} x} dx \tag{4.2}$$

where P - length of interval, where our function f(x) is known.

Let's say we have another function $f^*(x) = f(x \cdot 1.02)$. For ECG it may be the ECG of same patient, when he has a little faster heart rate. Its Fourier series will look like:

$$f^*(x) = f(x \cdot 1.02) = \sum_{n=-\infty}^{\infty} c_n^* e^{i2\pi \frac{n}{P} x} \tag{4.3}$$

But it also may be written as

$$f^*(x) = f(x \cdot 1.02) = \sum_{n=-\infty}^{\infty} c_n e^{i2\pi \frac{n}{P} x \cdot 1.02} = \sum_{n=-\infty}^{\infty} c_n e^{i2\pi \frac{n}{P/1.02} x} \tag{4.4}$$

So, you can see, that we can move from 4.1 to 4.4 with preservation of coefficients by just changing interval size P.

$$c_n = \frac{1}{P}\int_{-\frac{P}{2}}^{\frac{P}{2}} f(x)e^{-i2\pi\frac{n}{P}x}dx \tag{4.5}$$

With $y = {}^{x}/_{1.02}$

$$c_n = \frac{1}{P}\int_{-\frac{P/1.02}{2}}^{\frac{P/1.02}{2}} f(y\cdot 1.02)e^{-i2\pi\frac{n}{P}y\cdot 1.02}1.02\cdot dy = \frac{1}{P^*}\int_{-\frac{P^*}{2}}^{\frac{P^*}{2}} f^*(y)e^{-i2\pi\frac{n}{P^*}y}dy \tag{4.6}$$

where $P^* = {}^{P}/_{1.02}$.

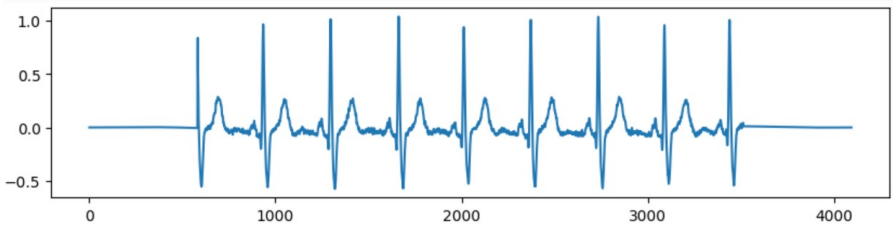
So, if we know the difference between f and f^*, we can transform them to the same Fourier components through choice of correct interval P. This transformation with calculated interval I have called **Fourier Transform with Dynamic Window**.

Now we should remember, that we should use discrete transform. It will look like

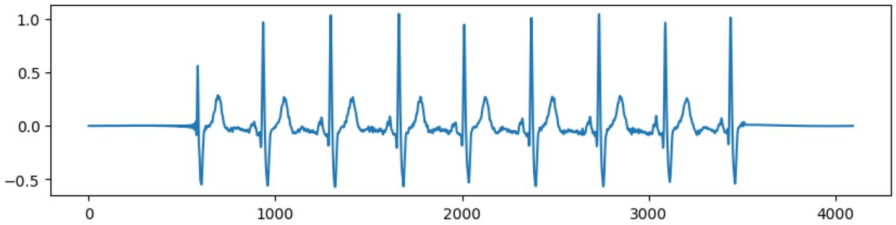
$$X_k = \sum_{n=0}^{N^*-1} x_n \cdot e^{-i2\pi\frac{k}{N^*}n} \tag{4.7}$$

, where N^* depends on heart rate.

The number of components in the Fourier expansion will also vary depending on N^*. This is not suitable for further signal processing, so I took only first 500 Fourier components. On the Fig. 4.1 I'm demonstrating, that 500 first Fourier coefficients are enough to re-composite signal in quality sufficient for diagnosis by eyes.



(a) Here is the initial signal - first channel of some random ECG



(b) Here is restored same signal - after rFFT I reset to 0 all components except the first 500 and then make inverse rFFT

Figure 4.1 Estimation of the significance of the first 500 components of the frequency domain

To the eye, the reconstructed signal hardly differs from the original one. It can be assumed that the first 500 components in frequency domain carry almost all the information necessary for diagnosis.

So I have limited myself to these 500 components. Here is the code I used for this transformation:

```
def fn_I_Fourie(instance, a, b, first_N, fs):
    # calculate heart_rate
    heart_rate = get_heart_rate(instance)
    # calculate necessary N*
    new_N = math.ceil(2000 * 120 / heart_rate)
    # calculate phase of some peak
    tick_of_peak = np.argmax(np.sum(np.abs(instance), axis = 1))
    tick_between_peaks = fs / (heart_rate / 60)
    tick_phase = (tick_of_peak - N/2 + tick_between_peaks/2)%tick_between_peaks
                        - tick_between_peaks/2

    # calculate actual signal start and end
    if new_N/2 <= N/2+tick_phase:
        A = 0
        C = math.floor(N/2-new_N/2+tick_phase)
    else:
        A = math.floor(new_N/2-N/2-tick_phase)
        C = 0

    if new_N/2 <= N/2-tick_phase:
        B = new_N
        D = math.floor(N/2+new_N/2-tick_phase)
    else:
        B = math.floor(new_N/2+N/2-tick_phase)
        D = N

    if B-A < D-C:
        D = C + B - A
    if B-A > D-C:
        B = A + D - C

    # Change signal size
    new_inst[A:B, :] = inst[C:D, :]
    # rFFT
    y_new = scipy.fft.rfft(new_inst, n=new_N, axis=0)

    # Get first 500 components
    return magnitude_new[:first_N]
```

I made here one more improvement - I have moved peak of ECG to the middle of new signal window. This should affected the phase of the signal so that the resulting Fourier coefficients for different signals were even more similar, but in the end I did not notice a significant effect of phase on classification quality. In final version of transformation I use only magnitudes of coefficients.

As a function $f(x)$ i took here ECG of heart rate 120 and on interval 2000 frames. All instances in CODE dataset have 4000 frames, but often they have only 70% of interval filled, so I chose such parameters, that on heart rate near 80 I use about 3000 of frames.

To illustrate the result of my transformations, I will present several signals restored after the operations described above using an inverse Fourier transformation (Fig. 4.2).

On the left there is channel 0 from initial signal with window, that I use for Fourier transform. On the right you can see result of inverse Fourier transform after **FFTwDW**. I accented two central cardiac cycles - their comparability was the main goal.

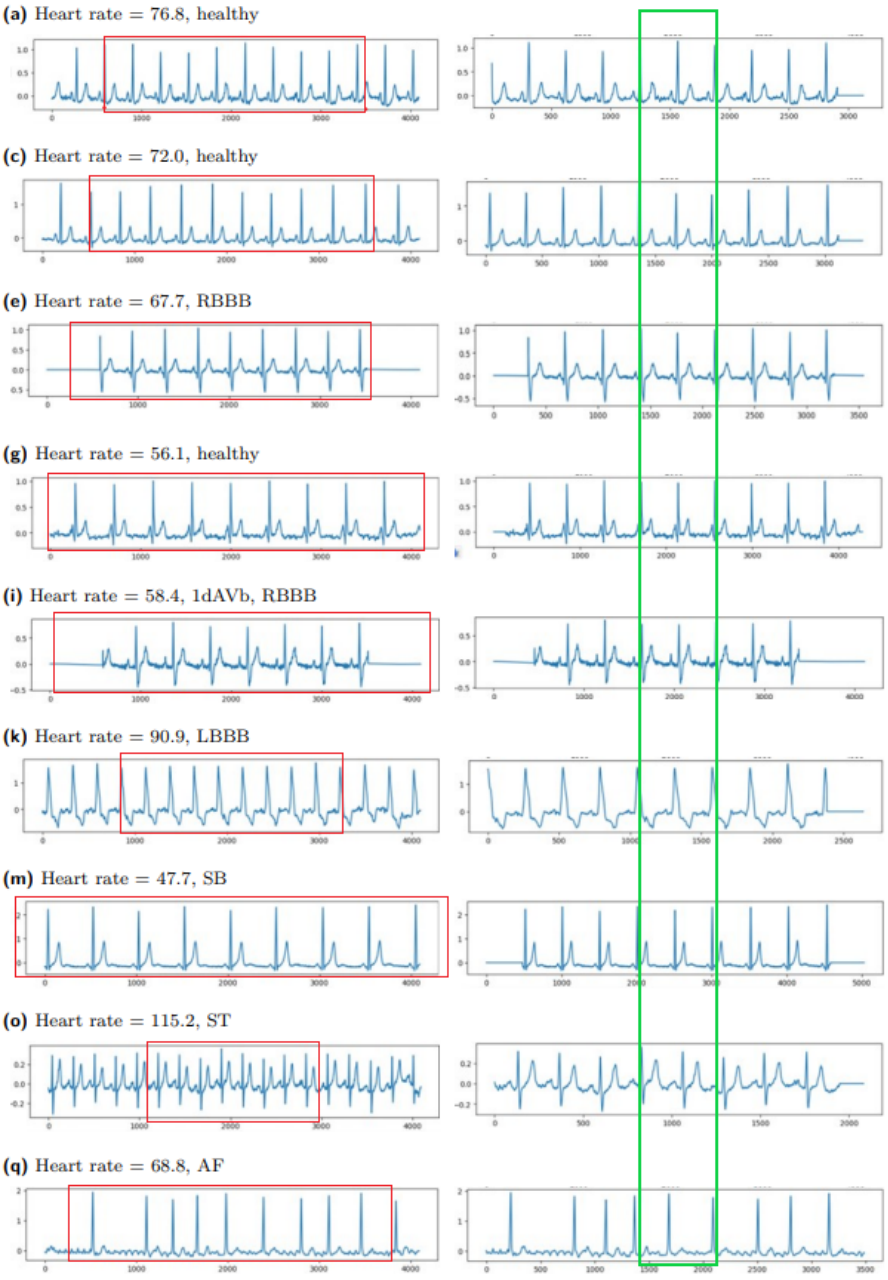


Figure 4.2 Examples of effect of Fourier Transform with Dynamic Window.

4.2 Transformations for comparison

In this section I will describe methods of data transformation I have used to prepare data for classification. Most of then are known, and I use some standard packages for them, but one is my own idea.

4.2.1 Time domain

This is the initial dataset. In the CODE article [6] there was no information about any noise removal or normalization. So I did not do it neither in this dataset. The only thing I have done was deleting several signals for which the Fourier transform did not work (they had a horizontal line in one or more channels - apparently, some sensor had moved away from the skin). So all datasets I have used had the same instances set.

4.2.2 Fast Fourier Transform for real values (rFFT)

The formula for FT is

$$f(x) = \int_{-\infty}^{\infty} F(k) e^{2\pi i k x} dk \tag{4.8}$$

where

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} dx \tag{4.9}$$

are exactly the components from frequency domain.

As is known, $e^{ix} = \cos(x) + i\sin(x)$ (Euler's formula). So dealing with real values functions it makes sense to get only real part of function:

$$F^c(k) = \int_{-\infty}^{\infty} \cos(2\pi k x) f(x) dx \tag{4.10}$$

This transform is often called as **Fourier Cosine Transform**.

If we try to apply these formulas in practice, we are faced with the fact that instead of a continuous signal we have a sequence of values at different points in time. This forces us to make certain important changes to the formulas:

- We have to replace the integral with the *sum*.
- Now we can calculate only as many components in the frequency domain as we have signal values.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi \frac{k}{N} n} \tag{4.11}$$

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \frac{k}{N} n} \tag{4.12}$$

where $n \subset \{0, \ldots, N-1\}$. This type of transformation is called **Discrete Fourier transform (DFT)**.

This algorithm has a complexity $O(n^2)$, which we always want to improve. And it is possible. There are few algorithms of DFT with complexity $O(n \log(n))$, so they are called **Fast Fourier Transform (FFT)**.

In my exploration I uses function from package *scipy*: scipy.fft.rfft. I use it on the whole signal length, which differs from the classical use of Fourier transform on small pieces of a signal to study the dynamics of signal changes over time. I don't try to study the dynamics, I want to collect the most accurate information about the signal spectrum.

```
1 def fn_Fourie(instance, first_N):
2     # Number of sample points
3     N = instance.shape[0]
4     y_new = scipy.fft.rfft(instance, n=N, axis=0)
```

```
    # Get magnitude
    magnitude_new = np.abs(y_new)

    if magnitude_new.shape[0] < first_N:
        magnitude_new_new = np.zeros((500, magnitude_new.shape[1]))
        magnitude_new_new[:magnitude_new.shape[0], :] = magnitude_new
        return magnitude_new_new
    return magnitude_new[:first_N]
```

You can see here, that I limit transformation result by first *first_N* components. It makes more sense in **FFTwDW** transformation, so I decided to copy the approach for better comparison.

4.2.3 Modified discrete cosine transform (MDCT)

Two next transformations I decided to add for comparison with other popular variants. **Modified discrete cosine transform (MDCT)** [15] is the popular transformation, used in most modern audio coding standards, including MP3, Dolby Digital (AC-3), Vorbis (Ogg), Windows Media Audio (WMA), ATRAC, Cook, Advanced Audio Coding (AAC), High-Definition Coding (HDC), LDAC, Dolby AC-4, and MPEG-H 3D Audio.

Classically FFT is applied after dividing the signal into equal consecutive non-overlapping intervals. MDCT applies FCT 4.10 to consecutive **overlapping** intervals, when two adjacent intervals intersect by half the length. It makes the MDCT especially attractive for signal compression applications, since it helps to avoid artifacts stemming from the block boundaries.

I have used the package *mdct* in default modification. I have not used all components from this transformation again, because I saw, that there is no visible information in components higher then 128 and I wanted to save space on hard disk.

```
import mdct
def mdct_reshaped(x):
    # MDCT
    MDCT = np.nan_to_num(mdct.mdst(x), 0)
    # Reshape from 3-D to 2-D, which is necessary for all my classifiers
    MDCT = MDCT[:128, :, :].reshape(128, -1)
    # normalization
    mx = np.max(np.abs(MDCT))
    if mx != 0:
        MDCT = MDCT / mx
    return MDCT
```

4.2.4 Wavelet transform (WT)

A wavelet is a wave-like oscillation with an amplitude that begins at zero, increases or decreases, and then returns to zero one or more times. On Fig. 4.3 you can see some examples of wavelets sufficient for Wavelet transform.

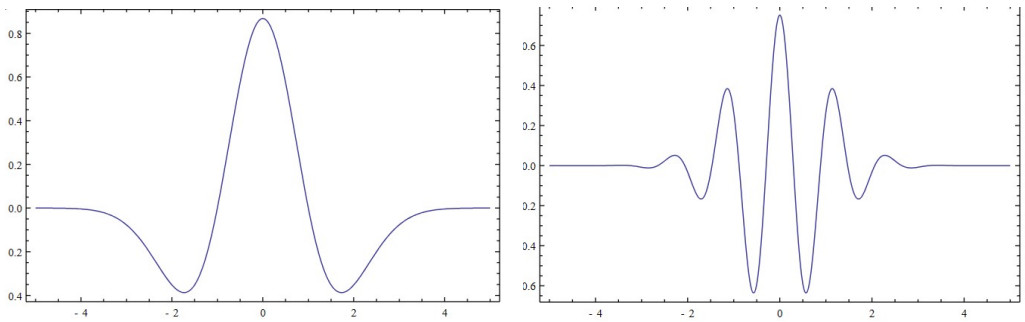
For **Wavelet transform (WT)** we create a subspace with coordinates a and b, where a is a scale, and b is a shift:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\Big(\frac{t-b}{a}\Big) \tag{4.13}$$

where ψ is a base wavelet function.

In the next step we calculate correlation between every of this function with the function $x(t)$ we want to transform:

$$WT_{\psi}\{x\}(a,b) = \langle x, \psi_{a,b}\rangle = \int_{\mathbb{R}} x(t)\psi_{a,b}(t)dt \tag{4.14}$$



(a) Mexican hat Wavelet **(b)** Morlet Wavelet

Figure 4.3 Examples of different Wavelet functions

Since wavelet is very locally defined function, wavelet coefficients reflect the behavior of function $x(t)$ in some area near point with shift b. Axis b plays here the role of a time axis. While the axis a defines wavelet scale. And since the scale changes the frequencies on wavelets Fourier transform, so the axis a is related to the frequency spectrum of the function $x(t)$. That is the reason, why Wavelet transform is considered forms of *time-frequency representation* of continuous-time signals 4.4 (source - [16]).

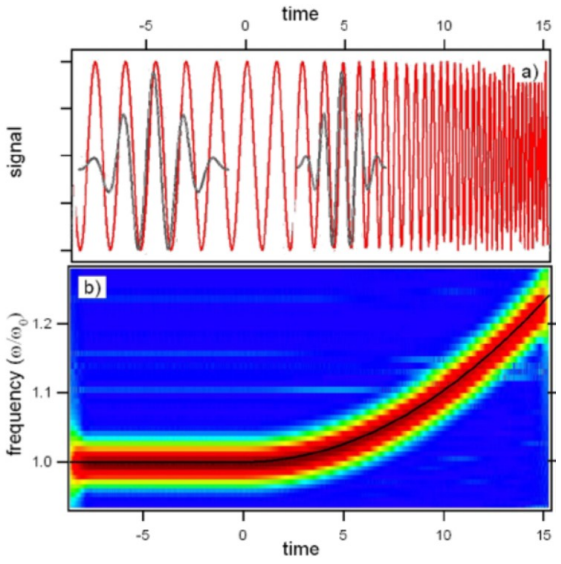


Figure 4.4 Nice
Wavelet demonstration from *Tip-sample interactions on graphite studied using the wavelet transform*

Not every function with limited definition area is suitable for WT.

1. The wavelet must have finite energy:

$$E = \int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty$$

2. The wavelet must have a mean equal to zero
3. For complex wavelets the Fourier transform must be both real and must decrease for negative frequencies

4. Localization: The wavelet must be continuous, integrable, have a compact domain, and be localized in both time (space) and frequency

There are a significant number of different suitable functions. In my experiment I used Coiflets "coif17" fro package PyWavelets (look at Fig. 4.5)

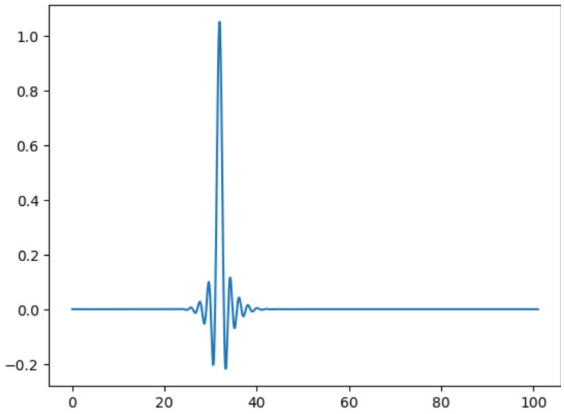


Figure 4.5 Wavelet "coif17"

My code I have used as preprocessing for WT dataset in final experiment:

```
import pywt
def wavelet_transformation(x):
    DWT = []
    # WT transform for every channel
    for i in range(x.shape[-1]):
        DWT.append(np.vstack(pywt.dwt(x[:, i], 'coif17')))
    # stack them to get 2-D tensor
    DWT = np.vstack(DWT)
    # normalization
    mx = np.max(np.abs(DWT))
    if mx != 0:
        DWT = DWT / mx
    return DWT.T
```

4.2.5 Short-Time Fourier Transform with the Window Size Fixed in the Frequency Domain (STFT-FD)

I have tried to apply this interesting transform from [10], but its calculation took two minutes per instance. Having near 300,000 instances, it was not possible to process them all. So I refused to include this transformation in the experiments. But in Appendix A.3 there are results of this transform. It probably would by as useful as **Wavelet transform**.

4.3 Classifiers

In this section there will be description of classifiers. One of them is from CODE article [6], which is fine-tuned for CODE dataset. The second one is my convolution neural network I have fine-tuned by using simulated annealing on dataset with **FFTwDW** preprocessing. And the others are different models representing various other neural network architectures and classifiers.

4.3.1 CODE ResNet

As I have mentioned before, this is the deep neural network from original article with CODE dataset [6]. The network consists of a convolutional layer (Conv) followed by 4 residual blocks with two convolutional layers per block. The output of the last block is fed into a fully connected layer (Dense) with a sigmoid activation function, σ, which was used because the classes are not mutually exclusive (i.e. two or more classes may occur in the same exam). The output of each convolutional layer is rescaled using batch normalization, (BN), and fed into a rectified linear activation unit (ReLU). Dropout is applied after the nonlinearity. (Fig. 4.6)

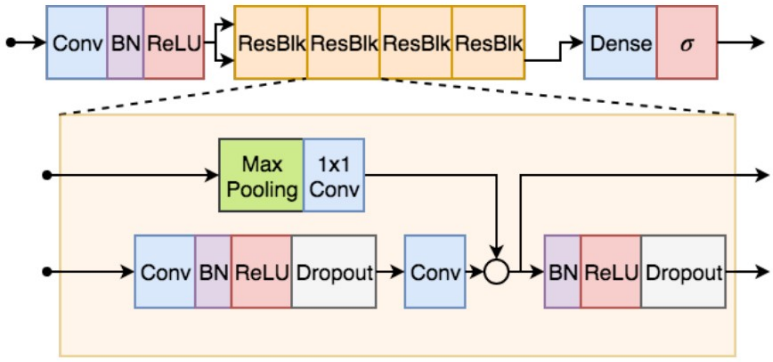


Figure 4.6 Architecture of residual DNN from the CODE article [6]

They made fine-tuning of the model in the following hyperparameter space: residual neural networks with 2, 4, 8, 16 residual blocks, kernel size 8, 16, 32, batch size 16, 32, 64, initial learning rate 0.01, 0.001, 0.0001, optimization algorithms SGD, ADAM, activation functions ReLU, ELU, dropout rate 0, 0.5, 0.8, number of epochs without improvement in plateus between 5 and 10, that would result in a reduction in the learning rate between 0.1 and 0.5.

The final parameters are: the convolutional layers have filter length 16, starting with 4096 samples and 64 filters for the first layer and residual block and increasing the number of filters by 64 every second residual block and subsampling by a factor of 4 every residual block. Max Pooling and convolutional layers with filter length 1 (1x1 Conv) are included in the skip connections to make the dimensions match those from the signals in the main branch.

4.3.2 4-layers CNN

From the beginning I had hope to create much simplier neural network then one from Section 4.3.1, to compensate for the complexity of the neural network with a high-quality preprocessing algorithm and obtain comparable results.

I have tried few 1-layer neural networks like Dense, Convolutional and LocallyConnected1D layers, and they were too weak. So I have stopped at 4-layers Convolution Neural Network. After every Convolutional layer I have MaxPooling1D (Fig. 4.7).

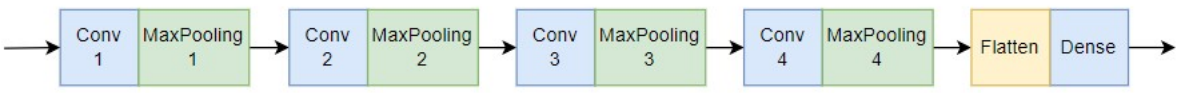


Figure 4.7 The structure of 4-layer CNN I fine-tuned

I have fine-tuned this NN using Simulated Annealing [17]. As objective function I have

used F1 score on Validation dataset, and for training I have used one by one files with data of CODE-15 dataset. More details you can read in Section 5.4.

The best one solution had vector of hyperparameters (1,0,9,0,2,3,5,1,1,0,9,0,3,2,7,1). Which gives us model on Fig. 4.8:

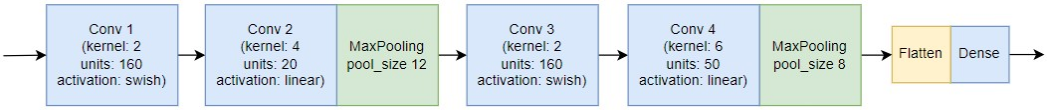


Figure 4.8 The structure of 4-layer CNN after fine-tuninig

4.3.3 Transformer

Initially Transformers are language models [18]. To apply them for ECG we need other embedding approach. Fortunately, this task was already solved in Visual Transformers (ViT) [19], and I took code for TensorFlow 2 from this article [20] without any change.

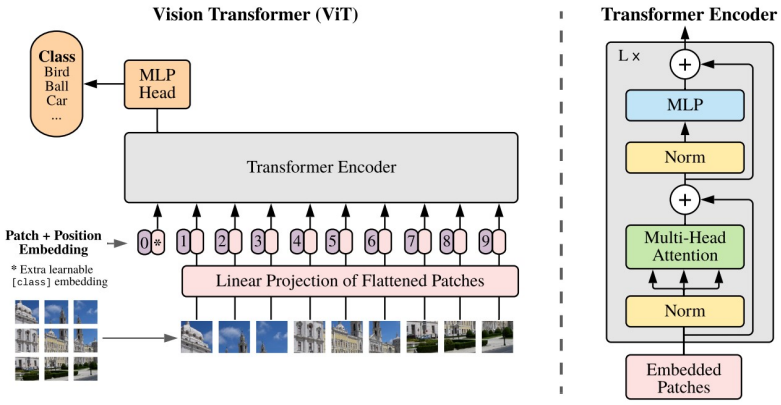


Figure 4.9 ViT model overview.

In ViT we split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by [18]. (Fig. 4.9)

Code of Transformer classifier is in attachment.

4.3.4 LSTM, LSTM+CNN

I have tried few simple variants of NN with LSTM layer, and came to the conclusion that LSTM works satisfactorily only in combination with the layer Bidirectional (Fig. 4.10).

Also there is interesting parameter in Tensorflow implementation of LSTM layer - *return_sequences*. $return\_sequences = False$ means, that after LSTM we get only output of *the last* LSTM cell. On the contrary, $return\_sequences = True$ let us work with outputs of whole LSTM layer. To deal with it I have added CNN layer after LSTM.

I couldn't decide, what solution will show better result, so implemented both of them. Its code is not too big, so I write it right here:

Code listing 4.1 Code for simple LSTM model

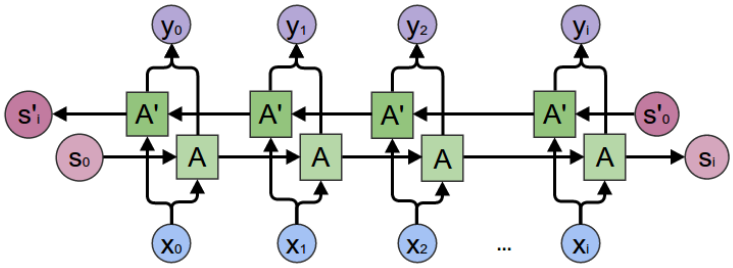


Figure 4.10 Scheme of bidirectional approach

```
def get_LSTM_model(input_shape, n_classes):
    input = tf.keras.Input(shape=input_shape, name='CNN_input_X')
    LSTM = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(12*6, dropout=0.2))(input)
    output = tf.keras.layers.Dense(n_classes, activation='sigmoid')(LSTM)

    model = tf.keras.models.Model(inputs=input
                                  , outputs=output)
    return model
```

Code listing 4.2 Code for LSTM model with Convoltuion layer

```
def get_LSTM_CNN_model(input_shape, n_classes):
    input = tf.keras.Input(shape=input_shape, name='CNN_input_X')
    biLSTM = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(12*6, dropout=0.2, return_sequenc
    CNN = tf.keras.layers.Conv1D(6, 3)(biLSTM)
    Flat = tf.keras.layers.Flatten()(CNN)
    output = tf.keras.layers.Dense(n_classes, activation='sigmoid')(Flat)

    model = tf.keras.models.Model(inputs=input
                                  , outputs=output)
    return model
```

I understand that there was room for fine-tuning, but Simulated Annealing takes an extremely long time even on more lightweight architectures. I decided to limit myself to one more or less working option, so you can assume that the parameters were chosen randomly.

4.3.5 Multi-layer Perceptrone (MLP)

Here I used **MLPClassifier** from package *scikit-learn*.

NN with 4 hidden Dense layers with 50, 40, 30, 20 neurons respectively. All other parameters are default. I took it as example of simpliest classification methods.

4.3.6 Random forest

For the same reasons I have added to final experiment **RandomForestClassifier** classifier from package *scikit-learn*.

I have chosen this one, because it took many times less time to train then **AdaBoostClassifier**, **DecisionTreeClassifier** or **GaussianProcessClassifier**, which I initially wanted to try too. But these methods took hours for every try, so I gave up on them.

I have taken these parameters values for this classifier: max_depth=15, n_estimators=50, max_features=1.

4.3.7 Other classifiers from *scikit-learn*

I wanted to try other classifiers from *scikit-learn*, like **AdaBoost**, **Decision Tree**, **Gaussian Process** and **RBF SVM**, but each of them took more then 8 hours to process one run, and I needed to calculate dozens of them, so I excluded them from experiments.

4.3.8 RNN, RNN+CNN

I wanted also to explore efficiency of **RNN** architecture, but had to decide not to include them into experiments. It took too long to train and what is worse my models did not converge at all. In Appendix A.3.1 you can take a look at code of my **RNN**-models, but in the experiments it is absent.

Chapter 5

Methodology of experiments

In this chapter I describe technical issues I have met and solved and the implementation of experiments.

5.1 General tools

5.1.1 Work with datasets

The modern convenient format for storing data is **.hdf5**. This is the initial format of dataset **CODE-15**. Package *h5py* allows download data from **.hdf5** file into *numpy* array, what is appropriate for any classifier on *TensorFlow* or from *scikit-learn*.

But if data are in few files, you need some tool, that can merge data to one array. For this I have implemented function **read_and_random_undersampling_dataset**. It can process one file with some number or all files in folder and combine them into one array. It can apply random under-sampling with changeable ratio. It allows to add pre-processing both for data and for labels. And it can return *tf.data.Dataset* object or two *numpy* arrays with data and labels separately.

However, in the end it turned out that the entire dataset did not fit into RAM. So I have developed another tool - *class* **hdf5_file_dataset** based on *tf.keras.utils.Sequence*. It works only with *TensorFlow*, but allows use data for training or evaluation reading them right from hard disk. Like previous one tool it can read data from one or few files or from all available matching files in folder. It allows to set batch_size, shuffling, under-sampling, pre-processing for data and labels, or choose only instances, limited to maximum disease count (I needed it for one of experiments).

The idea of this dataset is to build sort of map with information, in which file you should look for data with certain IDs. Files *.hdf5* allow to read data from any row or row range you want like it is *numpy* array. And additionally we create array with all IDs, matching conditions we need. Since it is very fast to process only files with labels, arrays of IDs for under-sampling or other labels-dependent filters is fast and cheap. It is a very flexible approach.

All these tools you can see in the application, file **load_dataset.py**.

5.1.2 Models evaluation - datasets

For last experiments I have prepared callback, that calculate and save to file estimation of model in 4 ways:

- Test dataset "Random test" from random 10% of patients

- With a common threshold that optimizes F1 score for all diseases together
- With a special threshold for each disease that optimizes F1 score for this particular disease

- Test dataset "Gold standard" from the CODE article [6]
  - With a common threshold that optimizes F1 score for all diseases together
  - With a special threshold for each disease that optimizes F1 score for this particular disease

In the end I decided, that the best tool to compare different models will be combination "Random test"+"common threshold", but I have all 4 variants of estimation.

5.1.3 Models evaluation - statistics and scores

In all experiments I collect to statistic files following information:

- Metrics
  - ⊕ Count of rows with **True positive** result, **TP**
  - ⊕ Count of rows with **True negative** result, **TN**
  - ⊕ Count of rows with **False positive** result, **FP**
  - ⊕ Count of rows with **False negative** result, **FN**
  - ⊕ Train duration
- Dimensions
  - ÷ Task name (depends on experiment)
  - ÷ Date-time of training start (defines different runs)
  - ÷ Disease number (I called it Task ID, because nature of labels was not important for me)
  - ÷ . . . any other useful features

The feature of these metrics (**TP**, **TN**, **FP**, **FN**) is that they can be aggregated across different runs and even across different experiment configurations without greatly deformation the meaning. The same cannot be said about indicators whose formulas contain division - **Precision**, **Recall** and **F1** in particular.

$$\overline{F1} \neq \frac{1}{N} \sum F1$$

At least choice of this formula for averaged **F1** is not robust to extreme metric values on small instances. From my experience I prefer to use initial formula for **F1** with aggregated components:

$$\overline{F1} = \frac{\sum TP}{\sum TP + 0.5 * (\sum FP + \sum FN)}$$

F1 score aggregated for group of experiments in this way will be closer to most part of experiments and will suppress some falling out events. In addition, experiments with a large amount of information (like total instances/patients count) will influence such a metric more - and this is good, because the influence of the law of large numbers on them is stronger.

5.2 Experiments implementation

5.2.1 Research on the relationship between disease and channels

I did this research before I fine-tuned my 4-layer CNN and before I implemented convenient datasets. So I created some 4-layer dense NN with the simplest structure:

```
input_F_X = tf.keras.Input(shape=(500,12,), name='CNN_input_F_X')
Get_elem = Lambda(lambda x: x[:, :, channel])(input_F_X)
# Flat = Flatten(name='Flat')(input_F_X)
D_1 = Dense(120, activation = 'linear')(Get_elem)
D_2 = Dense(90 , activation = 'linear')(D_1)
D_3 = Dense(60 , activation = 'linear')(D_2)
D_4 = Dense(30 , activation = 'linear')(D_3)
outputs = Dense(6, activation='sigmoid')(D_4)
model = tf.keras.models.Model(inputs=input_F_X
                              , outputs=outputs)
```

As dataset for the experiment I took only file 0 with under-sampling with all instances with disease and 10% of healthy instances. Table 3.1 you can see, that it make ratio healthy/sick close to 1/1. As pre-processing I used **FFTwDW** (section 4.1), since this is the innovation I tried to research. Validation dataset I also took from file 0 (Let me remind you that I divided all files to train-val-test parts).

During the experiment I have compared F1 score on Validation dataset after training on every ECG channel separately, on all channels (label "All") and on 9 channels without 3 the worst (label "Few").

Results of this experiment you can see in section 6.1.

5.2.2 Research of the usefulness of phase, real and imaginary parts

For this experiment I have written generator of 4-layer CNN from hyper-parameters vector. I used for this purpose hyper-parameters space 5.1, same as for simulated annealing. I randomly generated 10 different CNN, and I used for every experiment one random file with data of dataset CODE-15.

Every experiment I have repeated (in the same configuration CNN and file of dataset) 3 times, so F1 score is already averaged.

Results of this experiment you can see in section 6.1.1.

5.2.3 Training on patients with up to one disease and check of under-sampling

For this experiment I have used 2 configurations:

- **CODE ResNet** 4.3.1 + **Time domain** 4.2.1
- **4l CNN** 4.3.2 + **FFTwDW** 4.1

For both of them I trained classifier on 4 variation of dataset:

- Without Under-sampling (False), all patients (All)

- Without Under-sampling (False), only patients with 1 disease and healthy (1)
- With Under-sampling (True), all patients (All)
- With Under-sampling (True), only patients with 1 disease and healthy (1)

Under-sampling here is random the under-sampling with the ratio healthy-sick as 1:1.
Statistics collected is estimation on test datasets.
Results of this experiment you can see in section 6.1.2.

5.2.4 Main experiment: collection of statistics on transformations and classifiers

Idea of this experiment was to aggregate statistics about efficiency of different classifiers with different pre-processing techniques.

As train dataset I have used under-sampled train dataset **train_US_ratio_1** (data distribution look in Table 3.1), because there was problem with disk space on server, where I made calculations, and full **train** dataset does not fit to RAM of my PC.

As validation dataset I used full "val" dataset (20% of **CODE-15**). For model estimation I have collected all 4 variants from section 5.1.2.

I have used 5 options of **pre-processing**:

- **Time domain** - Initial data without any transformation. Section 4.2.1.
- **rFFT** - Fast Fourier Transformation for real values. Section 4.2.2.
- **FFTwDW** - Fourier Transform with Dynamic Window. Section 4.1.
- **MDCT** - Modified Discrete Cosine Transform. Section 4.2.3.
- **WT** - Wavelet transform. Section 4.2.4.

I used 7 different **classifiers**:

- **CODE ResNet** - It is the residual NN from the CODE article [6]. Section 4.3.1.
- **4L CNN** - It is 4-layer CNN, that I have fine-tuned on dataset with FFTwDW pre-processing. Section 4.3.2.
- **Transformer** - It is some example of Visual Transformer from Internet. Section 4.3.3.
- **LSTM** - There is the simplest LSTM NN with minimum layers. Section 4.3.4, code 4.1.
- **LSTM+CNN** - Here is LSTM NN in combination with CNN layer. Section 4.3.4, code 4.2.
- **MLP** - Standard Multi Layer Perceptrone classifier from *scikit-learn* package. Section 4.3.5.
- **Random Forest** - Random Forest classifier from *scikit-learn* package. Section 4.3.6.

When executing this experiment, I encountered the following difficulties.

First, classifiers from *scikit-learn* do not work with GPUs and with my dataset, that reads data from hard disk. These classifiers need *numpy* array to work with. While other classifiers do both these things. For this reason I created two different programs - one for my PC, which processed classifiers **MLP** and **Random Forest**, and second one for the computing server *gpu0102-prod.in.fit.cvut.cz*, where I have processed all other classifiers.

Second problem was, that there are two GPUs on that computing server, but TensorFlow by default use only one of them. I failed to set up the usage of both GPUs as a cluster, but I

succeed to run the program on certain GPU. So I divided my task pool to two parts by diseases (1, 2, 3 in first part, 0, 4, 5 in the second one). Them I ran simultaneously on different GPUs.

I tried to use the package *concurrent.futures* for parallel executions, but it did not use power of server effectively and did not speed up processing, so I prefer not to add this complication.

Final programs for computing server are rather straightforward. In nested loops by domains (pre-processing methods), classifiers and diseases I ran the sequence "prepare all datasets"→"fit model"→"evaluate model on test datasets "→"calculate TP/TN/FP/FN scores"→"write to statistics file" 3 times (for greater statistical reliability). There was complication with MDCT and WT domains - I have implemented them initially as preprocessing over Time domain data to calculate them right during training. It worked, but was very slow. So I have prepared these datasets for train and validation purposes as separate files for certain step of experiment only (there was problem with space on computing server).

On my PC I had also problems with computing resources. All *train*, *test* and *Gold standard* datasets could not fit to its RAM at the same time. Fortunately, I did not need validation dataset, cause classifiers **MLP** and **Random forest** don't use validation. So the sequence here was "prepare train dataset"→"fit model"→"delete train dataset"→"prepare test datasets"→"evaluate model on test datasets "→"calculate TP/TN/FP/FN scores"→"write to statistics file"→"delete test datasets". Which, of course, increased the running time of the program.

In all versions of the program I implemented a control of already processed combinations, since I had to restart the programs regularly, and the whole experiment lasted about a month.

Results of this experiment you can see in section 6.1.3.

5.3 Problem with dataset Gold standard

The CODE article [6] also provided residual model, that contains more then 40 layers (12 of which are Convolutional). I call this model CODE ResNet. It will be described in more detail in the chapter 4.

In the process of studying various models and data transformations, I noticed a stable unpleasant trend. In Table 5.1 we see a typical picture of the model's performance results on these two datasets. Regardless of the chosen classifier and ECG signal transformation, we see this behavior: evaluation on Test dataset shows more or less close results for all diseases, but on Gold standard we see terrible accuracy on diseases AF (4) and ST (5). We can see here, that two datasets have different distribution.

Table 5.1 F1 score for some classifier.

TestType	1dAVb 0	RBBB 1	LBBB 2	SB 3	AF 4	ST 5
Gold standard	0.79	0.93	0.99	0.84	0.04	0.10
Test (random)	0.56	0.79	0.77	0.62	0.71	0.79

Authors of the CODE article [6] had also noticed this. Here in the Table 5.2 are their results for split 90%-5%-5% under different rules (ordered: randomly; by date; stratified by patients). And we can see, that here they had little better results, then I have, but the distribution of which diagnoses are easier and which are more dif-

Table 5.2 F1 score for the same model

Type		1dAVb 0	RBBB 1	LBBB 2	SB 3	AF 4	ST 5
Test (random)	My	0.564	0.793	0.769	0.620	0.714	0.794
	Article	0.61	0.84	0.83	0.67	0.81	0.76
Gold standard	My	0.790	0.925	0.994	0.841	0.044	0.101
	Article	0.897	0.944	1.000	0.882	0.870	0.960
Gold standard training	GS only	0.831	0.900	0.925	0.798	0.832	0.950
	With Train	0.721	0.897	0.958	0.863	0.222	0.433

ficult to learn is the same. I show here in the table metric F1 for the same model on Time domain from my experiments. I think, that the reason of lower quality on random Test dataset is less data for training, but distribution looks same.

On the contrary, F1 score for Gold standard looks different. The only explanation that came to my mind is that there are no patients in the dataset CODE-15, that carry critical information to classify these 50 instances with AF and ST in Gold standard. To check this I have tried to train model on Gold standard dataset - I know, that there is over-fitting, but at least it shows, that the model is suitable for this task if the data is of high quality. In the rows "Gold standard training" there are my tries to train the same model on pure "Gold standard" dataset and on combination of "Gold standard" and Train datasets. Addition of extra 50 instances with these diseases significantly improved the results (in Train dataset there is 10,000 instances with these two diseases total).

5.4 Simulated annealing

Simulated annealing (SA) is a metaheuristic optimization technique based on the annealing process used in metallurgy, where a metal is heated to a high temperature quickly and then gradually cooled. At high temperatures, the atoms move fast, and when the temperature is reduced, their kinetic energy decreases as well. At the end of the annealing process, the atoms fall into a more ordered state, and the material is more ductile and easier to work with.

Similarly, in SA, a search process starts with a high-energy state (an initial solution) and gradually lowers the temperature (a control parameter) until it reaches a state of minimum energy (the optimal solution).

I have not found module in Python for SA in discrete space of hyper-parameters, so I have written my own code. It will be in Attachments.

Again, the structure of base CNN is on Fig. 5.1.

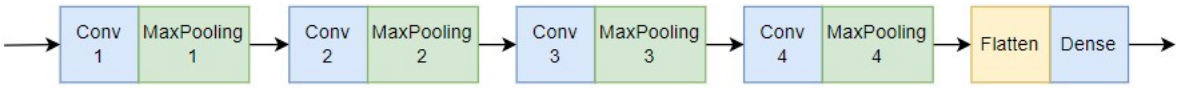


Figure 5.1 The structure of 4-layer CNN I fine-tuned

As hyper-parameter space I used the following set of properties:

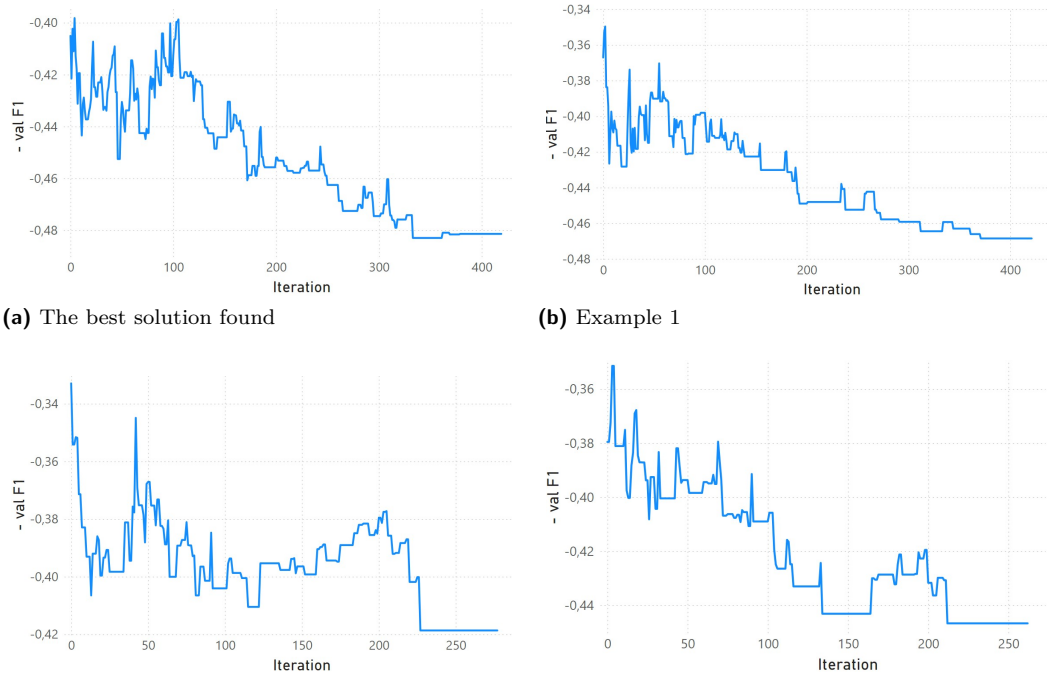
Code listing 5.1 Hyper-parameters space for 4-layer CNN generating

```
{ 'kernel_0' : (1, 2, 4, 6),      # kernel
  'max_pooling_0': (1, 4, 8, 12, 16),      # max_pooling
  'units_0': (2, 4, 8, 12, 16, 20, 30, 50, 80, 160), # units/filters
  'activation_0': ('swish', 'linear'), # activation
 'kernel_1' : (1, 2, 4, 6),      # kernel
  'max_pooling_1': (1, 4, 8, 12, 16),      # max_pooling
  'units_1': (2, 4, 8, 12, 16, 20, 30, 50, 80, 160), # units/filters
  'activation_1': ('swish', 'linear'), # activation
 'kernel_2' : (1, 2, 4, 6),      # kernel
  'max_pooling_2': (1, 4, 8, 12, 16),      # max_pooling
  'units_2': (2, 4, 8, 12, 16, 20, 30, 50, 80, 160), # units/filters
  'activation_2': ('swish', 'linear'), # activation
 'kernel_3' : (1, 2, 4, 6),      # kernel
  'max_pooling_3': (1, 4, 8, 12, 16),      # max_pooling
```

```
        'units_3': (2, 4, 8, 12, 16, 20, 30, 50, 80, 160), # units/filters
        'activation_3': ('swish', 'linear') # activation
    }
```

As objective function I have used F1 score on Validation dataset, and for training I have used random files with data of CODE-15 dataset.

I have ran the SA about 20 times, and even so it lasted for about 2 weeks, since every step of iterative annealing process is fitting of NN and most of them needed few hundred of steps to finish.



(a) The best solution found

(b) Example 1

(c) Example 2

(d) Example 3

Figure 5.2 Few examples of Simulated Annealing process

To demonstrate that it made sense, I will show you comparison of 2 set of train-evaluate cycles - one on the best found model and the other - on randomly generated in hyper-parameter space 5.1. In Table 5.3 you can see comparison of F1 score ranges for the-best-after-annealing model and set of random models.

Obviously SA allowed me to find the better CNN configuration then most of other random configurations, although this is not necessarily the best network globally.

Table 5.3 F1 score range

	F1 score range
Best	0,456 ± 0,023
Random	0,355 ± 0,075

Chapter 6

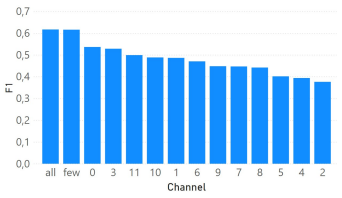
Experimental results

This chapter contains the results of all experiments I describe in this work.

6.1 Research on the relationship between disease and channels

Implementation of this experiment was described in section 5.2.1.

First result I got was the importance of different channels for total F1 score.



Channel	All	Few	0	3	11	10	1
F1 score	0.617	0.616	0.536	0.528	0.499	0.488	0.486

Channel	6	9	7	8	5	4	2
F1 score	0.470	0.448	0.447	0.442	0.401	0.394	0.376

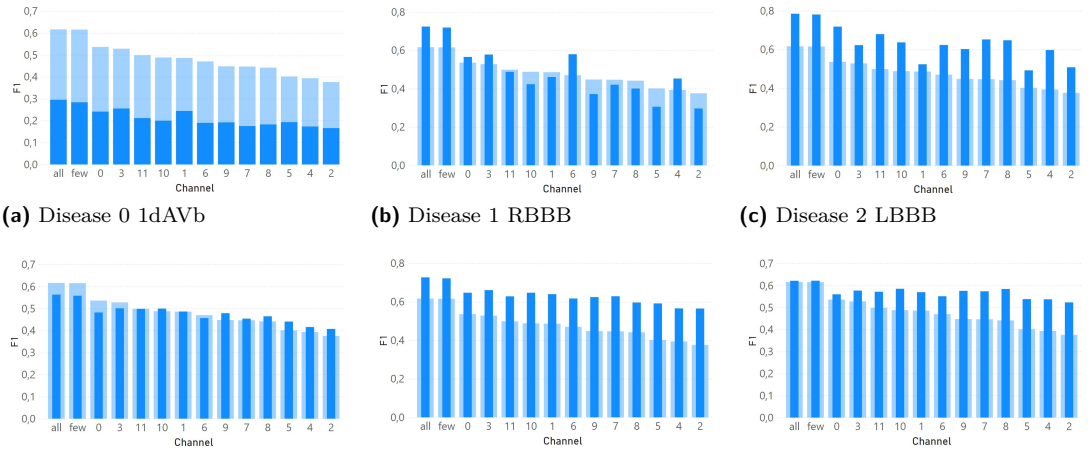
Figure 6.1 F1 score depending on channel

We can see here, that channels 2, 4 and 5 give less information about diseases, then others. Moreover, if we exclude them from training process, the effect of result seems not significant (it is the column *"Few"*). But for greater confidence let's take a look at the same table for all diseases:

From Fig. 6.3 we can see, that there are some difference in usefullness of channels for different diseases. But what I have found more important, Fig. 6.2 shows, that effect of 3 worst channels

Disease ID	0	1	2	3	4	5	6	7	8	9	10	11	few	all
0	0,241	0,244	0,166	0,255	0,173	0,193	0,190	0,175	0,183	0,192	0,200	0,212	0,284	0,295
1	0,566	0,461	0,297	0,578	0,453	0,306	0,580	0,421	0,401	0,372	0,423	0,488	0,719	0,724
2	0,718	0,524	0,508	0,623	0,598	0,493	0,624	0,652	0,648	0,603	0,637	0,680	0,781	0,785
3	0,483	0,487	0,408	0,501	0,416	0,441	0,457	0,454	0,465	0,479	0,500	0,499	0,558	0,564
4	0,647	0,640	0,565	0,661	0,566	0,592	0,617	0,629	0,596	0,625	0,647	0,628	0,722	0,727
5	0,560	0,570	0,524	0,577	0,538	0,538	0,552	0,574	0,585	0,576	0,585	0,572	0,622	0,622
Total	**0,536**	**0,486**	**0,376**	**0,528**	**0,394**	**0,401**	**0,470**	**0,447**	**0,442**	**0,448**	**0,488**	**0,499**	**0,616**	**0,617**

Figure 6.2 F1 score depending on channel, all diseases



(a) Disease 0 1dAVb **(b)** Disease 1 RBBB **(c)** Disease 2 LBBB

(d) Disease 3 SB **(e)** Disease 4 AF **(f)** Disease 5 ST

Figure 6.3 Effect of different channels for each disease separately

may be more significant for some diseases (look at *"Few"* and *"All"* for disease 0 **1dAVb**). So I decided not to ignore any of the channels.

6.1.1 Research of the usefulness of phase, real and imaginary parts

Implementation of this experiment was described in section 5.2.2.

I have compared F1 score for validation dataset after training of different datasets - Fourier Transform (section 4.2.2) and FT with Dynamic window (section 4.1) in tree variations each:

- Only magnitude
- Magnitude + Phase
- Magnitude + Real part + Imaginary part

Results are in Table 6.1. Columns *Phase* consist data with phase of Fourier Transform added to magnitude. Columns *Complex* consist data with Fourier Transform without additional transformations added to magnitude.

We can see, that addition of phase allowed to better results in more than half of cases. But average F1 score change is negative for **FFTwDW** and close to 0 for **rFFT**. For *Complex* colunm both indicators are even worse.

Moreover, this will require 2 and 3 times more disk space, respectively. So I decided to use only magnitude of Fourier transform coefficients.

6.1.2 Training on patients with up to one disease and check of under-sampling

Implementation of this experiment was described in section 5.2.3.

Idea of this experiment was to check, if patients with few diseases make the the training more difficult. Let's take a look at the table with experiment results Table 6.2.

First, we can see, that balancing under-sampling is not always useful for these NN. Moreover, for **CODE ResNet** there are better results without under-sampling. For **4l CNN** there is no

Dataset	Experiment	Variation	(F1 score)		Comparison		Difference	
	Nº	Magnitude	+Phase	+Complex	Phase	Complex	Phase	Complex
FFTwDW	0	0.352	0.441	0.4	1 (better)	1 (better)	0.09	0.049
	1	0.375	0.284	0.314	0 (worse)	0 (worse)	−0.09	−0.06
	2	0.367	0.455	0.354	1 (better)	0 (worse)	0.088	−0.013
	3	0.365	0.315	0.254	0 (worse)	0 (worse)	−0.05	−0.112
	4	0.27	0.311	0.271	1 (better)	1 (better)	0.041	0.002
	5	0.33	0.35	0.458	1 (better)	1 (better)	0.02	0.128
	6	0.389	0.324	0.204	0 (worse)	0 (worse)	−0.065	−0.185
	7	0.331	0.381	0.331	1 (better)	0 (worse)	0.05	−0.001
	8	0.419	0.398	0.426	0 (worse)	1 (better)	−0.021	0.007
	9	0.434	0.196	0.26	0 (worse)	0 (worse)	−0.238	−0.174
	10	0.399	0.43	0.298	1 (better)	0 (worse)	0.031	−0.1
	11	0.455	0.35	0.355	0 (worse)	0 (worse)	−0.104	−0.1
	12	0.329	0.366	0.396	1 (better)	1 (better)	0.037	0.067
	13	0.154	0.169	0.051	1 (better)	0 (worse)	0.015	−0.103
rFFT	0	0.057	0.057	0.054	1 (better)	0 (worse)	0.001	−0.003
	1	0.052	0.053	0.053	1 (better)	1 (better)	0.001	0.001
	2	0.048	0.108	0.049	1 (better)	1 (better)	0.06	0.001
	3	0.093	0.096	0.086	1 (better)	0 (worse)	0.004	−0.007
	4	0.054	0.111	0.054	1 (better)	0 (worse)	0.057	0
	5	0.111	0.063	0.06	0 (worse)	0 (worse)	−0.049	−0.051
	6	0.035	0.097	0.112	1 (better)	1 (better)	0.063	0.077
	7	0.07	0.051	0.051	0 (worse)	0 (worse)	−0.019	−0.019
	8	0.055	0.055	0.051	1 (better)	0 (worse)	0.001	−0.003
	9	0.051	0.054	0.052	1 (better)	1 (better)	0.002	0.001
	10	0.111	0.091	0.088	0 (worse)	0 (worse)	−0.02	−0.023
	11	0.049	0.045	0.049	0 (worse)	1 (better)	−0.003	0
	12	0.064	0.05	0.049	0 (worse)	0 (worse)	−0.013	−0.015
	13	0.108	0.037	0.036	0 (worse)	0 (worse)	−0.071	−0.071
FFTwDW	Average				0.57	0.36	−0.014	−0.043
rFFT	Average				0.57	0.36	0.001	−0.008

Table 6.1 Effect of addition phase and complex parts to dataset.

Configuration	Max disease count limitation	Under-sampling	1dAVb 0	RBBB 1	LBBB 2	SB 3	AF 4	ST 5
CODE ResNet + Time domain	All	True	0,538	0,795	0,756	0,600	0,713	0,783
		False	0,561	0,800	0,775	0,630	0,721	0,779
	Max 1 disease	True	0,500	0,786	0,727	0,594	0,696	0,748
		False	0,519	0,793	0,758	0,621	0,721	0,781
4l CNN + FFTwDW	All	True	0,215	0,705	0,714	0,357	0,634	0,538
		False	0,248	0,701	0,711	0,388	0,622	0,577
	Max 1 disease	True	0,249	0,690	0,700	0,365	0,636	0,525
		False	0,236	0,710	0,697	0,364	0,627	0,543

Table 6.2 Maximum disease count and under-sampling research results

significant difference between them. So I decided to use under-sampling if I will need to save disk space.

Second, if we estimate effect of limitation to patients with maximum 1 disease we can see, that for **4l CNN** effect is not defined, and for **CODE ResNet** it seems, that full dataset is little more useful. So I decided not to complicate the logic of other experiments and in them I left patients with any count of diseases.

6.1.3 Main experiment: collection of statistics on transformations and classifiers

Implementation of this experiment was described in section 5.2.4.

Here was a huge experiment, so there is plenty of results and their combinations.

6.1.3.1 General quality estimations

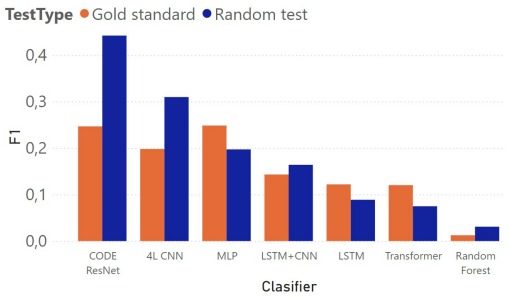


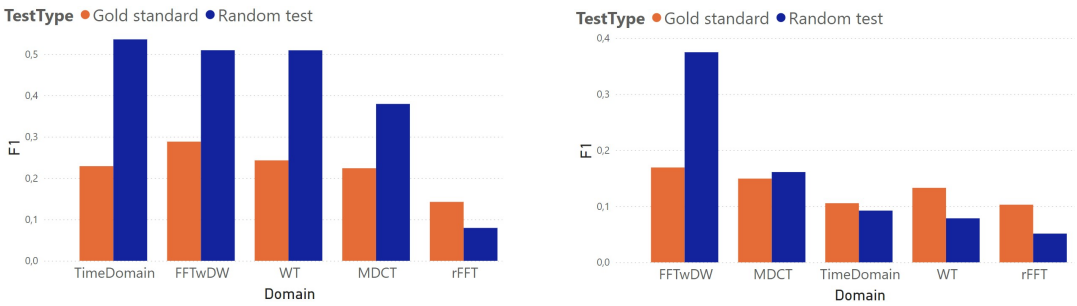
Figure 6.4 All classifiers - global estimation

Here of Fig. 6.4 we can see, that our classifiers have a wide range of quality. Don't be confused by such a big difference between the test datasets. I have mentioned it and offered my explanation for this in 5.3. I will explore classifiers in more details, but now I want to define two groups among them - the group of strong classifiers and the group of outsiders. We will see further, that different domains show different quality distribution in these groups.

Strong: CODE ResNet, 4l CNN, MLP

Weak: LSTM+CNN, LSTM, Transformer

I will show below, that Random forest is just too weak, and fails almost always.



(a) Strong group

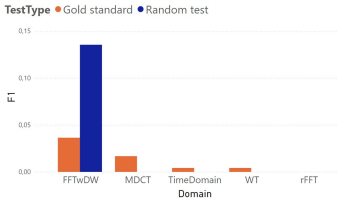
(b) Weak group

Figure 6.5 Estimation of domains on the strong and weak classifier groups

On the Fig. 6.5 you can see, that in case of strong classifiers preprocessings **FFTwDW**, **WT** and raw **Time domain** show almost equal and high efficiency. Every one of classifiers will show little different situation, but these 3 preprocessing methods show better results for more complex and large models.

On the contrary, for weak models preprocessing **FFTwDW** shows the best results. We will see below, that here also exists difference for particular classifiers, but **WT** and **Time domain** are too difficult domains for such simple models, and **FFTwDW** helps them a lot.

6.1.3.2 Look at Random forest



	1dAVb 0	RBBB 1	LBBB 2	SB 3	AF 4	ST 5
Random test	0,000	0,000	0,040	0,200	0,390	0,000
Gold standard	0,000	0,000	0,000	0,261	0,000	0,000

Figure 6.6 Results of Random forest, F1 score. Histogram for all domains, table only for **FFTwDW**

Random forest classifier from *scikit-learn* package was very fast in training, but almost useless. You can see here, that the only one preprocessing technique, that allowed Random forest to take some practical info from ECG, was **FFTwDW**. Even there the efficiency is low. On the Fig. 6.6 you can see both histogram of F1 score for all domains and table with F1 score for all disease only for **FFTwDW**.

Such a low results are the reason, why I excluded this classifier from experiment that explores diseases difficulties.

6.1.3.3 Diseases difficulties

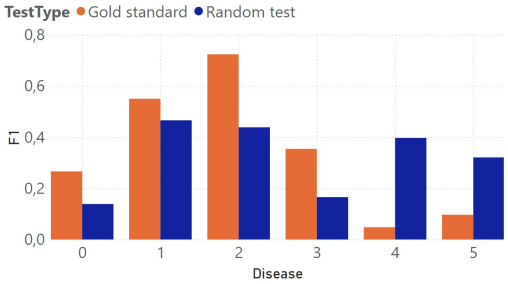


Figure 6.7 All diseases estimation on classifier group "Strong", F1 score

Here is F1 score separated by disease. I took here only "Strong" classifiers, because disease groups I want to define are more obvious here. But on the other classifiers except Random forest the situation is more or less similar.

We have two diseases, which are processed good both in **Gold standard** and **Random test** datasets. It is diseases 1 and 2 (**RBBB** and **LBBB**). And in **Gold standard** we see better score.

Next group is diseases 0 and 3 (**1dAVb** and **SB**) - we have the worst results for them on **Random test**, and **Gold standard** shows significantly worse score, then previous group.

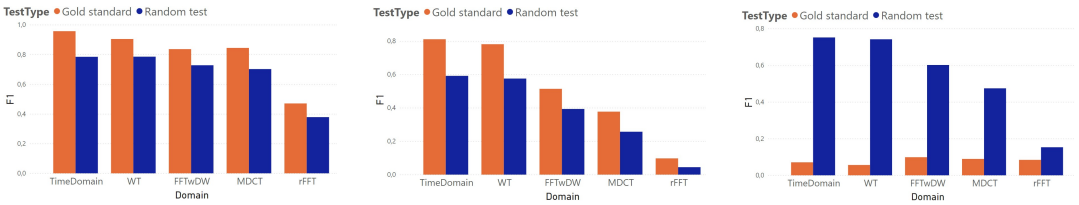
And the last one group is diseases 4 and 5 (**AF** and **ST**). On **Gold standard** it looks like classifiers doesn't work at all, but on **Random test** we see rather good results.

We can see here, that for first 4 diseases dataset **Gold standard** shows better results, then **Random test**. This correlates with the observations of authors of CODE article [6]. While on last 2 disease **Gold standard** looks very bad. I have researched this in section 5.3.

We will see this behavior almost on every particular classifier.

6.1.3.4 Preprocessing methods efficiency by classifier

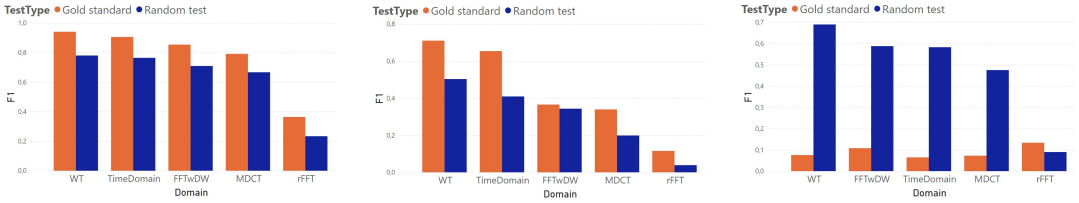
Here I will finally describe the results of the experiment in minimal acceptable detail to make it easier to understand. Full files with statistics will be in attachment and on GitHub. I prefer to use here histograms, but all numeric values are available in PowerBI file *Models analysis*. In Appendix A.1 you can take a look at my report page appearance. And in Appendix A.2 there is report page "Matrix", where it is possible to check **confusion matrix** for any configuration, to check, where F1 score came from.



(a) Diseases **RBBB** and **LBBB** **(b)** Diseases **1dAVb** and **SB** **(c)** Diseases **AF** and **ST**

Figure 6.8 Estimation of domains for disease groups for classifier **CODE ResNet**

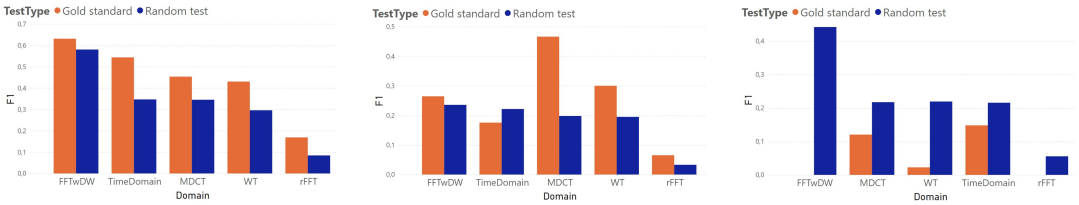
For **CODE ResNet** we can see, that **Time domain** gives the best results (Fig. 6.8). Nothing surprising, it was fine-tuned for this domain. But we can see, that Wavelets (**WT**) also works good. And for some diseases **MDCT** and **FFTwDW** work fine.



(a) Diseases **RBBB** and **LBBB** **(b)** Diseases **1dAVb** and **SB** **(c)** Diseases **AF** and **ST**

Figure 6.9 Estimation of domains for disease groups for classifier **4-layer CNN**

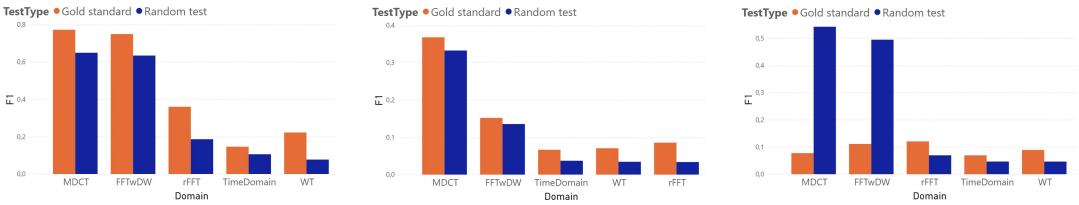
My **4-layer CNN** took more surprises (Fig. 6.9). Although **FFTwDW** also is functional here, better results we see on **WT** preprocessing, and even **Time domain** without preprocessing at all. Well, probably I could spend more time for fine-tuning, but even so it is more effective then most other random NN with similar architecture (see section 5.4).



(a) Diseases **RBBB** and **LBBB** **(b)** Diseases **1dAVb** and **SB** **(c)** Diseases **AF** and **ST**

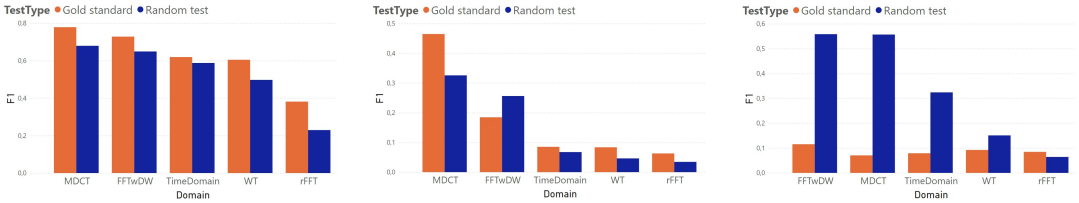
Figure 6.10 Estimation of domains for disease groups for classifier **MLP**

For **MLP** usefulness of **FFTwDW** is undoubted (Fig. 6.10). Only for disease 3 **SB** we see strangely high efficiency of **MDCT** algorithm on test dataset *Gold standard*, for which I don't see any prerequisites. So probably it is luck in finding some successful combination of factors.



(a) Diseases **RBBB** and **LBBB** **(b)** Diseases **1dAVb** and **SB** **(c)** Diseases **AF** and **ST**

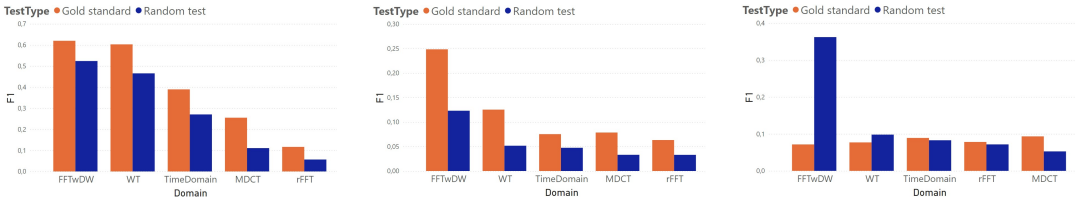
Figure 6.11 Estimation of domains for disease groups for classifier **LSTM**



(a) Diseases **RBBB** and **LBBB** **(b)** Diseases **1dAVb** and **SB** **(c)** Diseases **AF** and **ST**

Figure 6.12 Estimation of domains for disease groups for classifier **LSTM+CNN**

Models **LSTM** and **LSTM+CNN** took the main surprise in whole this experiment (Fig. 6.11 and 6.12). Although **FFTwDW** shows significant improvement compared with **Time domain**, the preprocessing method **MDCT** shows even better results. However, this still falls short of the best results of **CODE ResNet** and **4l CNN**, but I have not done any fine-tuning of **LSTM**, so probably this combination has high potential.



(a) Diseases **RBBB** and **LBBB** **(b)** Diseases **1dAVb** and **SB** **(c)** Diseases **AF** and **ST**

Figure 6.13 Estimation of domains for disease groups for classifier **Transformer**

FFTwDW turned out to be the best preprocessing methon for classifier **Transformer** (Fig. 6.13). Although compared to the best models, the results are not impressive at all, but it is clear that the **FFTwDW** is most suitable for this architecture.

Chapter 7
Conclusion

This thesis touched upon an important and quite developed area of signal research. In the process of working on it I have read few classic and modern articles about ECG signals processing and classification, and about different Neural networks architectures as **Transformers** or **Retentive neural networks**, which I did not find a way to use in this work.

I have studied one fresh and quite high quality dataset CODE and discovered some internal patterns in it. Here in the thesis I have showed different levels of difficulty for diseases, presented in the dataset. I also have described the problem with test dataset **Gold Standard** used in the CODE article [6] and tried to find its reasons.

I have studied different methods of signal transformation, like **Fourier Transform**, **Wavelet transform**, **Modified Cosine Transform**, which were included into this work, but also I have studied and explored **Short-Time Fourier Transform with the Window Size Fixed in the Frequency Domain** (STFT-FD [10]), which was too slow for usage (see App. A.3). In the process of studying, an idea of **Fast Fourier Transformation with Dynamic window (FFTwDW)** came to my mind, and I had not found it anywhere else. So I have explored it and compared with other methods, and have found areas, where it has an advantage over others.

I have implemented and explored few different NN architectures, including **Transformers**, **LSTM**, **CNN**, **Residual NN**. Additionally I have explored **RNN**, but it was excluded from experiment due to extremely slow speed of training and lack of satisfactory results on those configurations where I managed to try it. Due to the same reasons few more classifiers from *Scikit-learn* package, like **AdaBoost**, **Decision Tree**, **Gaussian Process** and **RBF SVM**, were not presented here. I chose only two the fastest classifiers from this packege - **MLP** and **Random forest**.

Finally I have collected structured statistics about different architectures and preprocessing methods and made the convenient tool to explore this data. This required my extensive reporting development experience from job, and in the process I also have built ROC curves and Recall-Precision graphs in PowerBI, which was not useful in the end.

There are enough things I had not time to complete. It may be useful to check **FFTwDW** preprocessing method on other popular ECG datasets like **PTB** or **IKEM**. I am sure that by spending more effort on fine-tuning the models, I could get results comparable to the state-of-the-art **CODE ResNet**. Probably, other combinations like **LSTM+MDCT** has much more higher potential then we have seen here.

Appendix A

Appendix

A.1 Report page "Tables"

Here is the first page of the report I prepared to analyze data of main experiment (Fig. A.1). I can choose here any set on classifiers, domains, diseases, test datasets and even certain run.

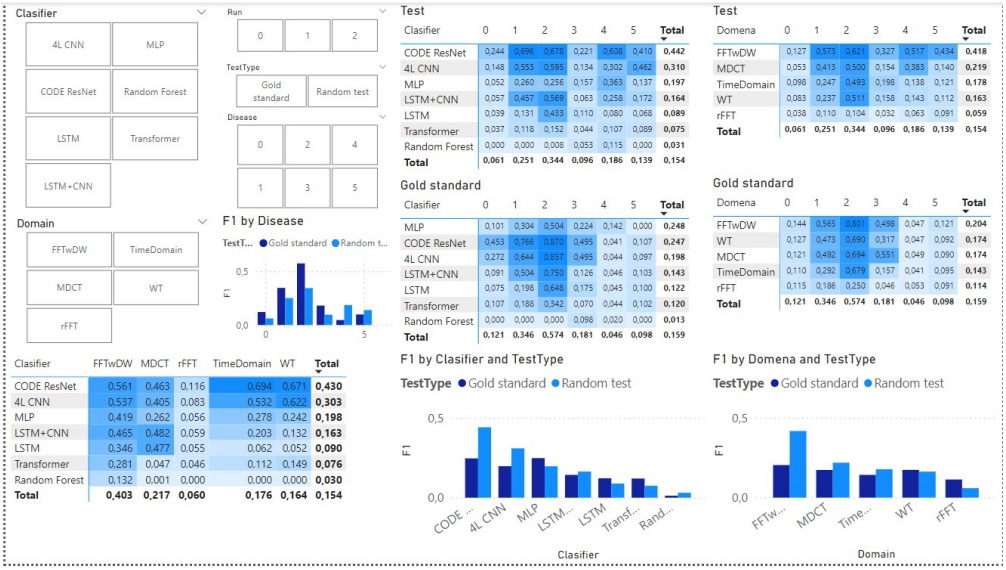


Figure A.1 PowerBI, list Tables

A.2 Report page "Matrix"

Here is the second page of the report I prepared to analyze data of main experiment. It was convenient to understand, why I have certain F1 score for some combination of tools. It also helped me to understand, if some model better process positive or negative cases, for example. Below I will add few examples of looking of the certain numbers.

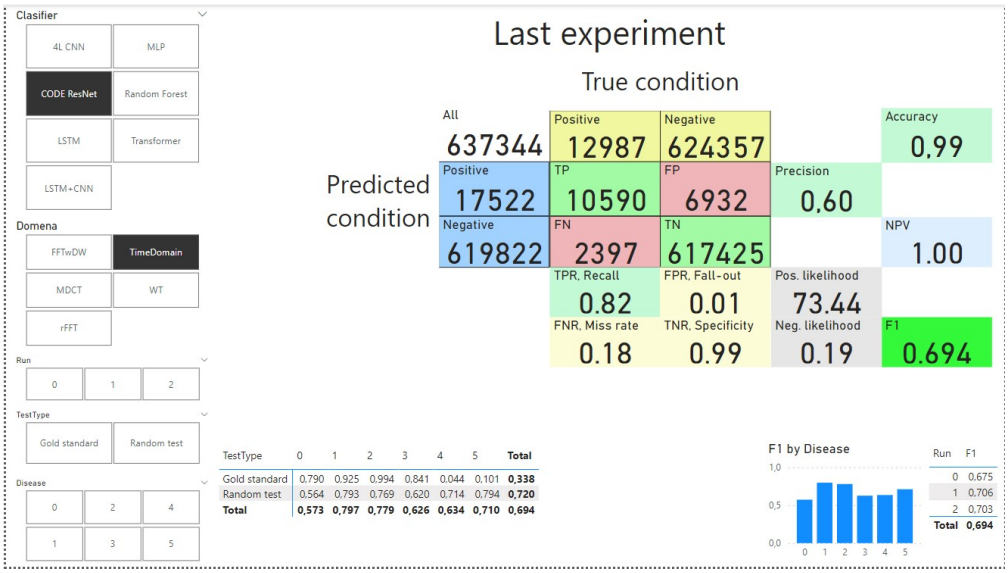
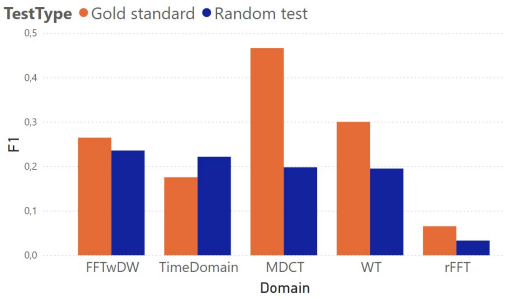


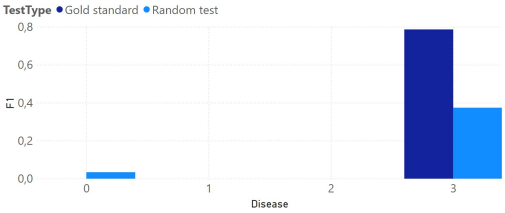
Figure A.2 PowerBI, list Matrix

A.2.1 Examples of Confusion matrix

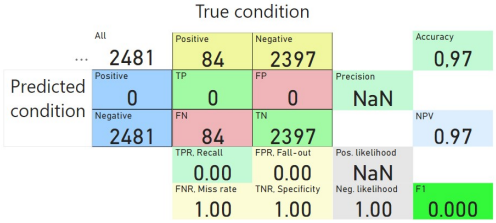
Strange peak of **MDCT** for **MLP** classifier on Fig. 6.10. We can see here, that this combination could not cope with disease **1dAVb** at all. But on **SB** it perfectly diagnosed every sickened instance, and had quite few false positives.



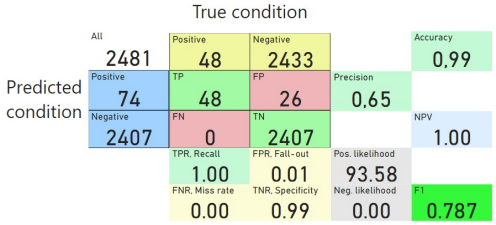
(a) Diseases **1dAVb** and **SB**



(b) Distributiom between diseases



(c) Disease **AF** confusion matrix

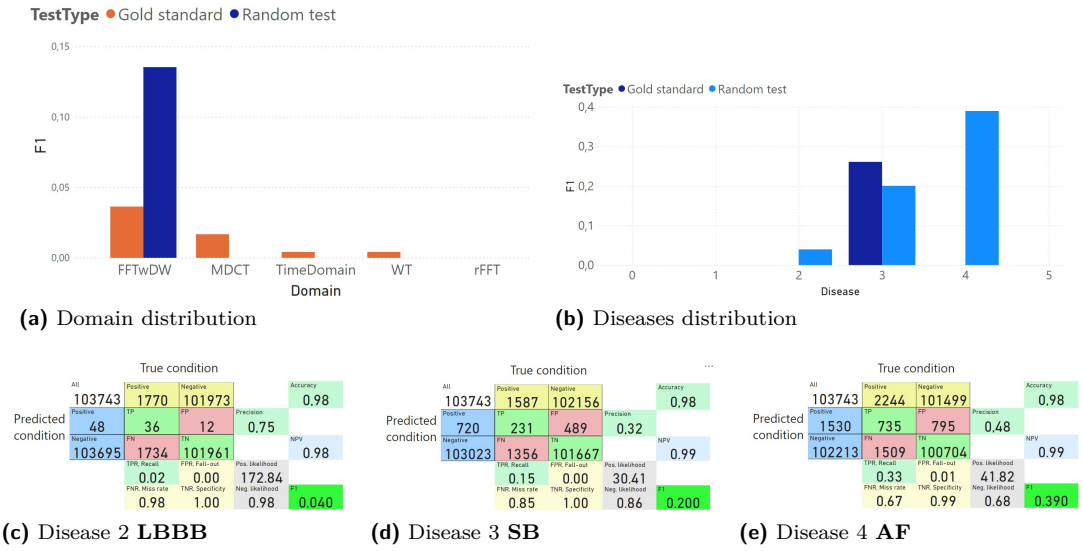


(d) Disease **ST** confusion matrix

Figure A.3
Estimation of domains for disease group **1dAVb** and **SB** for classifier **MLP** and test **Gold standard**

Random forest showed some efficiency with **FFTwDW** preprocessing. Let's take a look, how good it really was (Fig. A.4). Well, in the best case, **AF**, it could diagnose hald of sick

instances. Not so good. But with other preprocessing methods it couldn't diagnose nothing at all. Knowing how Random forest works, I would guess that **FFTwDW** succeeded to find some really important frequency combination for this disease.



(a) Domain distribution **(b)** Diseases distribution

(c) Disease 2 **LBBB** **(d)** Disease 3 **SB** **(e)** Disease 4 **AF**

Figure A.4 Results of Random forest, F1 score. All domains, All diseases for **FFTwDW** only, and confusion matrices for diseases 2, 3, 4.

A.3 Short-Time Fourier Transform with the Window Size Fixed in the Frequency Domain (STFT-FD) examples

As I have mentioned in 2, the idea of this transform is to use for each frequency the count of data, that contains some certain count of cycles. So, different frequencies use different samples count in summation in the Fourier transform formula.

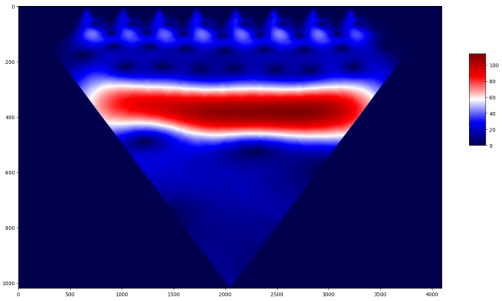
Here is Fig. A.5 is the result of **STFT-FD** on one channel of ECG. It is calculated from top to the bottom, using count of frames for window in integral from 2 to $\frac{\langle Count\ of\ frames \rangle}{\langle Count\ of\ cycles \rangle}$. Each such window is used for Fourier coefficient calculation for one certain frequency with corresponding period. These windows are overlapping here, and we have different count of them for different frequencies. You can see here the straight version of picture and one with exponential Y-axis. This one is more similar to what they show in their article, and there is more useful information for comparing different instances by eye.

And finally here is the concatenated result for all 12 channels, which I'd use for experiments. Unfortunately, as I mentioned, it takes too long - about 2 minutes for each of 300,000 instances. It would take more then year for me only to process the whole dataset. So I did not use this transformation.

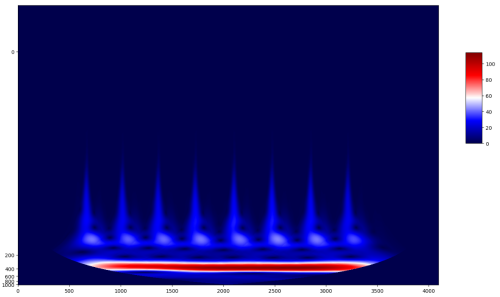
A.3.1 RNN, RNN+CNN

Same as with **LSTM**, I used **RNN** in combination with the layer Bidirectional (Fig. **??**).

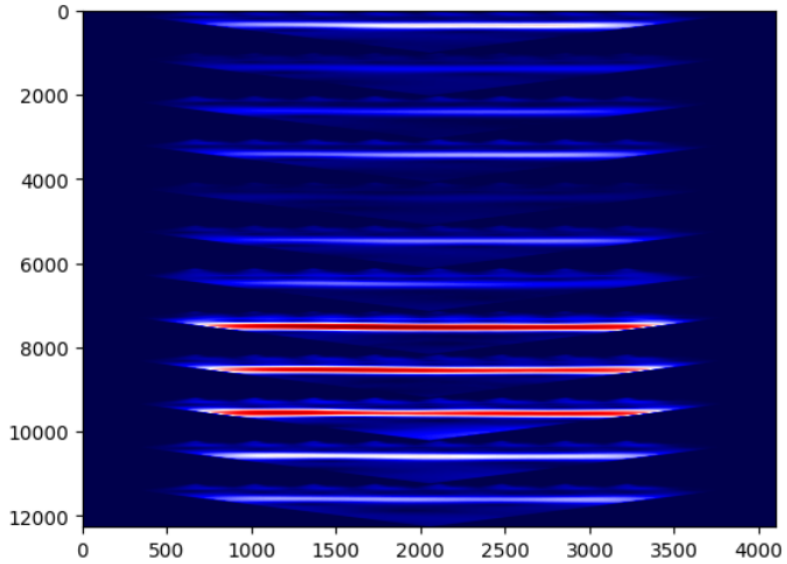
And Tensorflow implementation of RNN layer also has parameter *return_sequences*. *return_sequences* = *False* means, that after RNN we get only output of *the last* RNN cell. On the contrary,



(a) STFT-FD on one channel with straight Y-axis



(b) STFT-FD on one channel with exponential Y-axis



(c) STFT-FD on 12 channels

Figure A.5 Results of **STFT-FD**

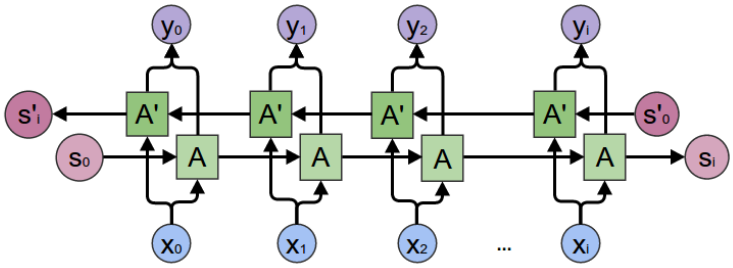


Figure A.6 Scheme of bidirectional approach

return_sequences = *True* let us work with outputs of whole RNN layer. To deal with it I have added CNN layer after RNN.

I couldn't decide, what solution will show better result, so implemented both of them. Its code is not too big, so I write it right here:

Code listing A.1 Code for simple RNN model

```
def get_RNN_model(input_shape, n_classes):
    input = tf.keras.Input(shape=input_shape, name='CNN_input_X')
    RNN = tf.keras.layers.Bidirectional(tf.keras.layers.RNN(MinimalRNNCell(12*6)))(input)
    output = tf.keras.layers.Dense(n_classes, activation='sigmoid')(RNN)

    model = tf.keras.models.Model(inputs=input
                                  , outputs=output)
    return model
```

Code listing A.2 Code for RNN model with Convoltuion layer

```
def get_RNN_CNN_model(input_shape, n_classes):
    input = tf.keras.Input(shape=input_shape, name='CNN_input_X')
    biRNN = tf.keras.layers.Bidirectional(tf.keras.layers.RNN(MinimalRNNCell(12*6), return_sequen
    CNN = tf.keras.layers.Conv1D(6, 3)(biRNN)
    Flat = tf.keras.layers.Flatten()(CNN)
    output = tf.keras.layers.Dense(n_classes, activation='sigmoid')(Flat)

    model = tf.keras.models.Model(inputs=input
                                  , outputs=output)
    return model
```

Unfortunately **RNN** took more then 6 hours to train and in those few configuration I tried them they both didn't converge. So I excluded it from experiments.

Bibliography

1. AL., G. A. Roth et. Global, regional, and national age-sex-specific mortality for 282 causes of death in 195 countries and territories, 1980–2017: A systematic analysis for the Global Burden of Disease Study 2017. *The Lancet*. 2018, vol. 392, no. 1, pp. 1736–1788. Available from DOI: 10.1016/S0140-6736(18)32203-7.
2. *AIMCARDIO.COM. 12 Lead Placement Guide with Diagram.* [N.d.]. Available also from: https://aimcardio.com/blog/12-lead-placement-guide-with-diagram/.
3. SINHA, R. *An Approach for Classifying ECG Arrhythmia Based on Features Extracted from EMD and Wavelet Packet Domains*. 2012.
4. CHAZAL, P. de; O'DWYER, M.; REILLY, R. Automatic classification of heartbeats using ECG morphology and heartbeat interval features. *IEEE Trans. Biomed. Eng.* 2004, pp. 1196–1206. Available from DOI: 10.1109/TBME.2004.827359.
5. ANBALAGAN, T.; NATH, M. Kumar; VIJAYALAKSHMI, D.; ANBALAGAN, A. Analysis of various techniques for ECG signal in healthcare, past, present, and future. *Biomedical Engineering Advances*. 2023. Available from DOI: https://doi.org/10.1016/j.bea.2023.100089.
6. RIBEIRO, Antônio H.; RIBEIRO, Manoel Horta; PAIXÃO, Gabriela M. M.; OLIVEIRA, Derick M.; GOMES, Paulo R.; CANAZART, Jéssica A.; FERREIRA, Milton P. S.; ANDERSSON, Carl R.; MACFARLANE, Peter W.; MEIRA JR., Wagner; SCHÖN, Thomas B.; RIBEIRO, Antonio Luiz P. Automatic diagnosis of the 12-lead ECG using a deep neural network. *Nature Communications*. 2020, vol. 11, no. 2, p. 1760. ISSN 2041-1723. Available from DOI: 10.1038/s41467-020-15432-4.
7. NATARAJAN, Annamalai; CHANG, Yale; MARIANI, Sara; RAHMAN, Asif; BOVERMAN, Gregory; VIJ, Shruti; RUBIN, Jonathan. A Wide and Deep Transformer Neural Network for 12-Lead ECG Classification. In: *2020 Computing in Cardiology*. 2020, pp. 1–4. Available from DOI: 10.22489/CinC.2020.107.
8. LIU, Minghao; REN, Shengqi; MA, Siyuan; JIAO, Jiahui; CHEN, Yizhou; WANG, Zhiguang; SONG, Wei. *Gated Transformer Networks for Multivariate Time Series Classification*. 2021. Available from arXiv: 2103.14438 [cs.LG].
9. BEGUM, S; PRIYADARSHI, Esha; PRATAP, Sharath; KULSHRESTHA, Sharmistha; SINGH, Vipula. Automated Detection of Abnormalities in ECG signals using Deep Neural Network. *Biomedical Engineering Advances*. 2022, vol. 5, p. 100066. Available from DOI: 10.1016/j.bea.2022.100066.
10. MATEO, Carlos; TALAVERA, Juan. Short-Time Fourier Transform with the Window Size Fixed in the Frequency Domain. *Digital Signal Processing*. 2017, vol. 77. Available from DOI: 10.1016/j.dsp.2017.11.003.

11. LI, Hongzu; BOULANGER, Pierre. Structural Anomalies Detection from Electrocardiogram (ECG) with Spectrogram and Handcrafted Features. *Sensors*. 2022. Available from DOI: 10.3390/s22072467.

12. ALAMR, A.; ARTOLI, A. Unsupervised Transformer-Based Anomaly Detection in ECG Signals. *Algorithms*. 2023. Available from DOI: 10.3390/a16030152.

13. VELOSO, Adriano; MEIRA JR, Wagner; ZAKI, Mohammed. Lazy Associative Classification. In: 2006, pp. 645–654. Available from DOI: 10.1109/ICDM.2006.96.

14. *Page to download dataset CODE-15*. [N.d.]. Available also from: https://zenodo.org/records/4916206.

15. PRINCEN, J.; JOHNSON, A.; BRADLEY, A. Subband/Transform coding using filter bank designs based on time domain aliasing cancellation. In: *ICASSP '87. IEEE International Conference on Acoustics, Speech, and Signal Processing*. 1987, vol. 12, pp. 2161–2164. Available from DOI: 10.1109/ICASSP.1987.1169405.

16. MALEGORI, Giovanna; FERRINI, Gabriele. Tip-sample interactions on graphite studied using the wavelet transform. *Beilstein journal of nanotechnology*. 2010, vol. 1, pp. 172–81. Available from DOI: 10.3762/bjnano.1.21.

17. LAARHOVEN, Peter J. M.; AARTS, Emile H. L. *Simulated Annealing: Theory and Applications*. Springer Dordrecht, 1987. Available from DOI: 10.1007/978-94-015-7744-1.

18. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need*. 2017. Available from DOI: 10.48550/arXiv.1706.03762.

19. DOSOVITSKIY, Alexey; BEYER, Lucas; KOLESNIKOV, Alexander; WEISSENBORN, Dirk; ZHAI, Xiaohua; UNTERTHINER, Thomas; DEHGHANI, Mostafa; MINDERER, Matthias; HEIGOLD, Georg; GELLY, Sylvain; USZKOREIT, Jakob; HOULSBY, Neil. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. Available from DOI: 10.48550/arXiv.2010.11929.

20. GOTTESMAN, Yoni. *Interpretable ECG Classification With 1D Vision Transformer*. [N.d.]. Available also from: https://yonigottesman.github.io/ecg/vit/deep-learning/2023/01/20/ecg-vit.html.

Contents of attachments

These files are also posted on Github: `https://github.com/Laeryid/DiplomaCVUT`

- `readme.md` ..... the work annotation
- `experiments` ..... directory with scripts for experiments
  - `exp1_relations.ipynb` ..... experiment 1 6.1
  - `exp2_phase|_complex.ipynb` ..... experiment 2 6.1.1
  - `exp3_patients_one_disease_and_US_4lCNN.py` ..... experiment 3 on 4l_CNN 6.1.2
  - `exp3_patients_one_disease_and_US_CODE.py` ..... experiment 3 on CODE RenNet 6.1.2
  - `exp_main_scikit_learn.ipynb` ..... main experiment, Scikit-learn classifiers 6.1.3
  - `exp_main_tensorflow.py` ..... main experiment, TensorFlow classifiers 6.1.3
- `packages` ..... directory with scripts for technical issues
  - `load_dataset.py` ..... work with datasets
  - `anneal_simulation.py` ..... Simulated Annealing implementation
  - `callback_save_files.py` ..... callbacks implementation
  - `model_Article_CODE.py` ..... code of ResNet
  - `CNN_creation.py` ..... code of generator CNN from hyper-parameters
  - `RNN_LSTM_Transformer.py` ..... implementation of RNN, LSTM and Transformer
- `src` ..... code of thesis in LaTeX
  - `FITthesis-LaTeX-master.zip` ..... source code of the thesis
- `result` ..... thesis text and PowerBI report with data
  - `thesis.pdf` ..... thesis text in PDF format
  - `Models analysis.pbix` ..... report for main experiment statistics, PowerBI