A mobile application for signal comparison using Siamese networks

Match EGM

First Examiner:
Prof Dr Stephan Dörfel

Second Examiner:
Dr. Med. Evgeny Lyan

Michael Polonskiy

Abstract

Treating a special type of heart arrythmia, requires a procedure called cardiac ablation. During this procedure, operators utilize patient's electrograms, provided by an implanted pacemaker, determining where culprit tissue is located and whether the clinical arrythmia observed, corresponds to the pattern searched for. Analyzing electrograms during pace mapping, provides an indication, which areas should be investigated further and more thoroughly. Comparison between patient's recorded electrograms, containing originally occurred arrythmia and clinical ones, has to be done visually via image comparison, because technical restrictions by the manufacturer deny a digital export of patient signals from the implanted pacemaker devices. Since this method is inefficient and imprecise, this thesis wants to create an application capable of processing patient signal data, which is represented as image pairs, thereby calculating a similarity score for any image signal pair given. In addition, this work wants to investigate how well a prediction algorithm for signal comparison can be pretrained, using surrogate data, given no true electrogram correlation values available, due to technical export restrictions.

To achieve this goal, two surrogate datasets with different data variance have been created, containing labelled signal image pairs. A third dataset acquired from patient's electrocardiogram signals has been used for model validation and evaluation, because we believe this data to approximate patient's electrograms closest. Based on current research in the area of signal processing and neural network development, two different prediction models (one deterministic and one non-deterministic) have been developed and implemented. Using up-to-date model explanation techniques, we provide an integral prediction explanation technique, verifying and describing the prediction results achieved. Providing a flexible and secure prediction solution, we have developed a web application, enabling electrophysiologists to use prediction models with mobile devices, laptops and desktop pcs alike.

Our experiments show, that using surrogate data an effective prediction model can be established, with increasing variance in training data favoring the overall prediction quality. Both approaches (deterministic and non-deterministic) succeed at providing useful similarity scores, with the neural network surpassing the deterministic algorithm. Moreover, our model explanation provides a simple and intuitive way for understanding and verifying prediction results. With regards to the program required, we have successfully implemented a web application, which can be run securely on internal laboratory hardware infrastructure, complying with general data protection standards, as well as operated by electrophysiologists in an easy manner.

Acknowledgement

I would like to thank the Gerhard C. Starck Foundation who supported me both ideally and financially during my course of studies. The grant made it possible for me to focus on my academic career and thanks to the ideal support I was able to expand upon my personal knowledge horizon.

List of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial
CNN	Convolutional Neural Network
CT	Computerized Tomography
ECG	Electrocardiogram
EGM	Electrogram
EPU	Electrophysiological Laboratory
GAN	Generative Adversarial Network
GDPR	General Data Protection Regulation
GPU	Graphical Processing Unit
KDD	Knowledge Discovery Process
MAE	Mean Average Error
MRI	Magnetic Resonance Imaging
MSE	Mean Squared Error
RAM	Read Only Memory
TPE	Tree Parzen Estimators
UKSH	Universitätsklinikum Schleswig-Holstein

1 Introduction

Medical image processing is part of many advanced medical diagnostics and modern pathological analysis utilizes information obtained from various imaging procedures. Popular examples of these technologies which many people will encounter throughout their life time are X-ray imaging, magnetic resonance imaging (MRI) and computerized tomography (CT). Next to these very common examples, medical imaging also takes up a key role in electrophysiology, a medical discipline which revolves around the electrical properties of biological cells and tissues [1]. Although the term itself covers all type of cells which transfer electric current, in the context of medicine, electrophysiology usually denotes the subject area that concerns itself with matters of the human heart, namely the heart rhythm. Therefore, in medical domain this term represents a sub division of cardiology, the main "branch of medicine that deals with disorders of the heart and the cardiovascular system" [2]. Clinical laboratories which deal with heart arrhythmias are often called electrophysiological laboratories (EPU labs).

The electrophysiology laboratory of the University Hospital Schleswig-Holstein (UKSH) focuses on particularly challenging heart operations and utilizes cutting edge methods and research to heal and improve patients' lives. One area of research is cardiac ablation a procedure developed to treat heart arrhythmia (for more information refer to [3]). The main task is to find the critical site of rhythm disorder and involves placing a catheter into a blood vessel and guiding it into the patient's heart. There, malfunctioning cells can be destroyed by cauterizing them. That way, electric signals can no longer be transmitted through these cells and the main trigger for an irregular heartbeat can be cured [4]. However, finding the right spot for cauterization is not an easy task. First of all, not only one or two cells but a complete path needs to be blocked. Finding the right area and building a complete path which will block further erroneous electrical stimuli is challenging. If the path is not completely sealed, thus providing some current leakage, or the wrong path is blocked, the procedure has to be repeated. Therefore, these types of operations usually take up several hours and require very modern and costly medical equipment, as well as highly skilled staff. In order to find the correct area for operation, a voltage map of the heart is created. This 3-D map shows the sequence in which the patient's heart is activated. Figure 1 below displays part of a human heart, namely the atrium cordis, with its electro-anatomical map depicting measured voltages in corresponding colors: red corresponds to low/erroneous voltage while purple corresponds to high/normal voltage. The big red tags along the left and right parts of the heart called pulmonary veins, are connected by a grey line in the image and represent the path that was blocked using ablation in order to stop a malfunctioning heart rhythm from circulating[1].

[1] This type of ablation is very common and is called pulmonary vein isolation (PVI), for more information see [6]

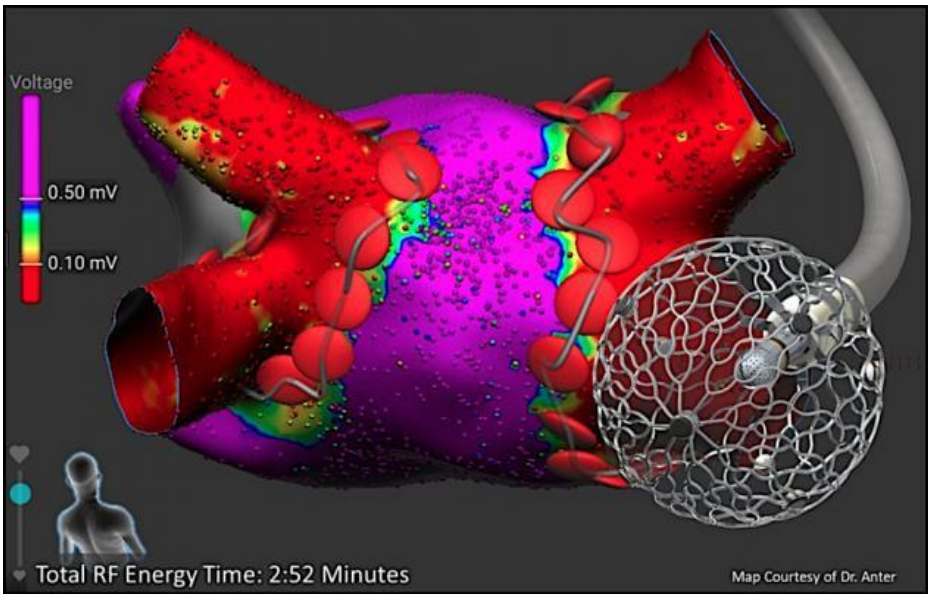


Figure 1 Image of a voltage map (red areas indicate injured tissue) [5].

The preferred condition for the creation of this map is while the patient is experiencing the arrythmia, since that way the exact trace of the signal can be plotted. However, in some cases inducing the arrythmia is not possible or even dangerous since it can lead to heart failure. In such cases plotting the map yields only limited information. In other cases, a further difficulty is finding the right arrythmia. Some patients suffer from several irregularities simultaneously, but not all are harmful / need to be treated in the same way. When trying to identify and cure the observed problem a different signal can be picked up upon and traced. This will result in the root of the problem not being cured and the arrythmia occurring again after the operation.

1.1 Match EGM

One effective tool mitigating the named challenges is using the recorded electrical activity of the patients heart, called an electrogram (EGM), from when the arrythmia has occurred. Most patients suffering from these kinds of diseases have implanted pacemakers that monitor and record their heart rate via electrograms. Recordings can be extracted by proprietary software while the patient is in the laboratory and displayed on a monitor. During operation the signals can provide information on whether the same arrhythmia is present, as well as help to locate the right area when arrythmia cannot be induced. Still, restrictions by the manufacturer prohibit doctors from exporting these recordings in any other format other than a video file. Therefore, if such a scenario occurs, electric signals have to be compared visually while operating. The signal measured by the device during the operation is captured as an image for a desired timeframe and put up against a template, cut out from the saved recordings. Figure 2 displays a screenshot taken from a video which was extracted from a patient's pacemaker. If we consider the signal denoted with the symbol FF we can now cut out a timeframe and compare it against a measurement taken live while operating.

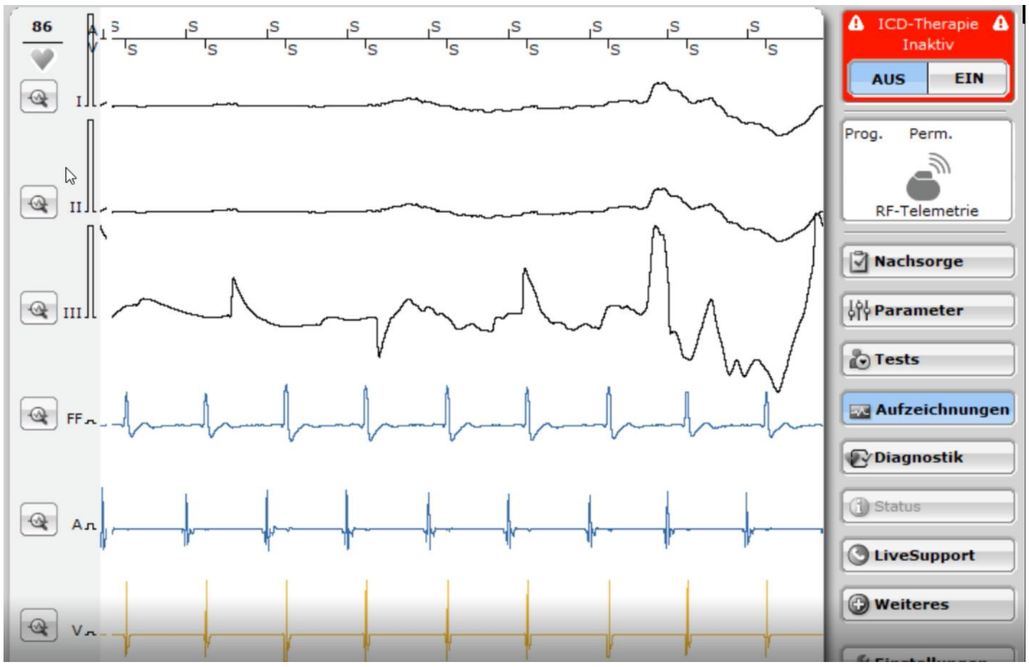


Figure 2 Screenshot from a pacemaker recording.

Determining the percentage of similarity between two EGM images visually, based on operator's perception, is quite challenging and not precise. Therefore, we want to develop an application which can take two images extract the underlying signals and calculate the corresponding similarity score. This score can then be used to establish whether the operator is on track for ablating culprit tissue and whether the clinical arrythmia (the right one) is being observed or not. Moreover, the result can be used in the process of pace mapping, when trying to narrow down the correct area. Low correlation values indicate being far off from the arrhythmia origin, while high values correspond to being close to origin (reminiscent of a hot and cold search).

Summing up our task, we start off with a pair of images from the implanted device: one clinical template and one image depicting the signal currently captured during electrical stimulation at the current position of our mapping catheter. Given this input data, our goal is to provide information on whether the current catheter position is ideal (high correlation value) or whether a better position should be searched for, yielding a new image with a new comparison and a higher correlation value.

1.2 KDD Process

Such type of problem can be solved by utilizing the Knowledge Discovery in Databases process (KDD process), which was first published in 1996 by Fayyad et al. [7]. Although this process focusses on data mining in order to extract meaningful patterns from mostly large databases, we can adapt this concept for our purposes. Figure 3 depicts an extract from the original KDD process, which we will explain and adapt in the following section:

1. **Selecting target data**
   This first step takes care of selecting the right type of data based on an identified goal.
2. **Date Preprocessing**
   The second step revolves around cleaning and processing. Common tasks are handling outliers and missing data points as well as noise reduction and normalization.
3. **Data Transformation**
   Third, useful features need to be extracted from the data. Usually several features might be combined into a new metric, with the benefit of dimensionality reduction.

4. **Data Mining / Machine Learning**
   Fourth, useful algorithms for exploiting data patterns are selected and applied to the data.
5. **Interpretation and Evaluation**
   Finally, mined patterns are interpreted and evaluated.

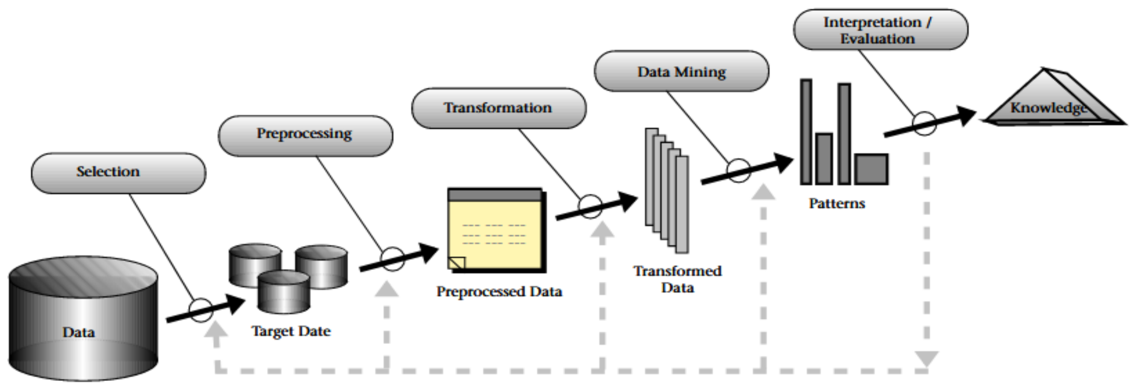


Figure 3 Original KDD process [7].

Looking at this methodology we can easily observe, that our task contains some major differences, thus adaptation is needed. We do not have any database to utilize, instead our data input consists of image pairs, where for each operation one image is static (chosen as template for the patient) while the other one changes in real time with change of catheter position. Moreover, knowledge acquired is very sensitive to the patient and template image chosen. A good catheter position (indicated by a high correlation value) can be meaningless with a different patient , e.g. a different template used. Furthermore, since EGM images cannot provide any correlation label, surrogate labels are required for performance evaluation. Thus, we need to create a generally applicable set of data, consisting of image pairs with an already determined correlation value in order to devise and evaluate prediction procedures. In addition, we do not need to perform traditional data mining or pattern discovery, since our pattern is clearly defined by the signal. Instead, an application is required which can extract the pattern (signal) from the image and provide a correlation value. Figure 4 shows the adapted KDD process which will be the basis for our work.

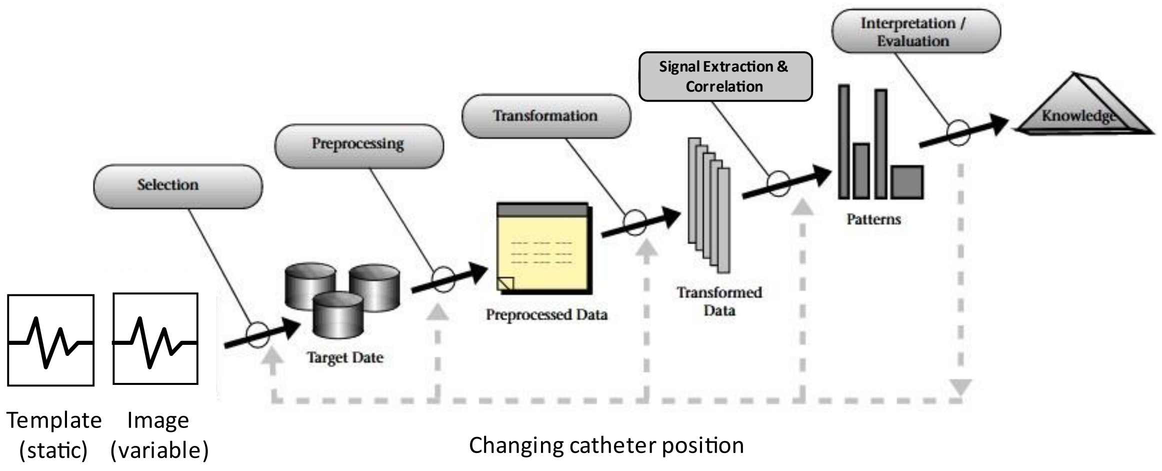


Figure 4 Adapted KDD process.

1.3 Main Structure

Utilizing the adapted KDD process our work consists of 10 chapters, which we want to present in this section:

Chapter 1 Introduction: In this first chapter we introduced the reader to the application domain: Electrophysiology. We explain the goal of this work and provide its global structure which builds on top of the KDD process. Finally the scientific contribution of this academic work is highlighted.

Chapter 2 Related Work: This chapter refers to previous research done in this area. We want to touch upon the main models and structures used as well as the results which have been achieved.

Chapter 3 Data Generation and Acquisition: As mentioned in the previous section we need to create a generally applicable dataset, which provides us with correlation labels we can use for model evaluation. To generate useful surrogate data we will use the beginning of this chapter to elaborate further on basic electrophysiology concepts establishing meaningful data generation. Thereafter, we want to experiment with several different data generation approaches. The first approach will focus on artificial image generation based on tabular ECG data while the second approach will build on top of ECG images collected in the electrophysiology laboratory.

Chapter 4 Data Processing: This chapter will introduce several preprocessing techniques applied to any image pair (surrogate or EGM) run through our adapted KDD process. Main aspects of this chapter focus on image resizing and removing possible extraction disturbances.

Chapter 5 Data Transformation: We want to use this chapter to introduce a general signal transformation technique which has been developed for a similar task. The key notion of this chapter is presenting an approach on how to treat pixel representations of any given signal.

Chapter 6 Signal Extraction & Correlation Prediction: Next, we want to develop and implement different methods for extracting signals from any image pair given and computing a corresponding correlation value. The main content of this chapter focusses on two different approaches, comparing deterministic prediction and signal extraction via algorithm to none deterministic one using a neural network. Thus, the main architecture of the network used will be discussed and presented.

Chapter 7 Experiments Setup: This chapter will explain the general experiments setup. Since we utilize several different datasets and prediction methods, we want to illustrate how we are going to compare the different possible combinations. Moreover, we want to define the possible grid of hyperparameters for training our neural network and introduce the algorithm used for hyperparameter optimization.

Chapter 8 Experiment Results & Discussion: After setting up our investigation, all experiments are caried out accordingly. This chapter aims at presenting and discussing the results in their chronological order. In addition, we want to define which procedure shall be used for correlation prediction during operations in the EPU laboratory. Finally, a method taking care of the explainability of predictions made by the neural network will be introduced.

Chapter 9 Deploying Match EGM: Considering a successfully established extraction and prediction procedure, we want to develop an application, including all necessary preprocessing and transformation steps, which can be operated by electrophysiologists in the EPU laboratory.

Chapter 10: Conclusion and Outlook: This chapter will reevaluate our results with regards to our goal and scientific contribution defined. Finally, we will present further improvements and research areas worth examining.

1.4 Scientific Contribution

The main contribution of this work consists of two components. The first task, as already mentioned above, focusses on developing a sophisticated application, which electrophysiologists can utilize to make informed decisions during operation procedures. This application needs to be reliable, understandable und easily manageable. Moreover it should be implemented such, that medical staff who are less informed in terms of computer science, are enabled to use it and benefit from its predictions.

The second task of this work is to investigate which techniques can be used to extract and compare time series data (in our case provided as ECG signals) from images and how neural networks can utilize artificially generated data to approximate a task which lacks quantifiable supervised information.

2 Related Work

This thesis wants to build on top of previous research in this area. However, most scientific contributions regarding image comparison are based upon classification tasks, which deal with the question if two images are identical or not [8]. Only very few authors examine the task of comparison as a regression problem. Nevertheless we believe, that for our case it is essential to treat this problem as a regression task, as explained in the introduction of this thesis. The first paper we want to utilize is "ReSimNet: drug response similarity prediction using Siamese neural networks", published by Jeon et al [9]. Although the authors do not handle images, the core of this work deals with similarity prediction of two compound pairs. In drug discovery it is believed that efficient medicine can be created by identifying a target for a disease and deriving a compound that binds to this target. Since the structure of the compound is essential, compounds with similar structure are said to have similar binding effects on a target. Thus, given a known compound for a target, the authors want to investigate structure similarity scores to other targets, analyzing how similar two compounds are. That way potential drug surrogates or alternative healing methods should be deduced. Therefore, two compounds are used as an input pair for a neural network, which creates two corresponding embedding vectors and calculates the cosine similarity between these two vectors. This measure is than used as the compound structure similarity score and compared against a label provided by the ZINC15 database, "a curated collection of commercially available chemical compounds prepared especially for virtual screening" [10]. For the architecture of the neural network a Siamese Network is used, which we will utilize for our network architecture as well (for details on Siamese networks see chapter 6.2.1). We want to adapt the general notion of taking in a pair of two data objects, transforming them into two corresponding embedding vectors and calculating a similarity score based off these two embedding vectors.

Contributions published on the popular webpages "Towards Data Science" and "Medium" utilize the same approach, albeit with images. Here images are transferred into embedding vectors and compared using cosine similarity. The authors describe in their blog posts how to utilize this concept in order to create an image recommender system, returning the n most similar images for a target image given ([11], [12]). However the images used, contain much more structure and texture since complex objects like cars, dogs or humans are depicted. Therefore, the network architectures used in these examples cannot be implemented in the same manner for our task. Moreover, while image similarity between dogs or cats is relatively vague, the correlation labels needed for our goal should be as precise as possible, with clearly distinguishable signals, which have a distinct correlation value. Therefore, we believe that the general setup of ReSimNet is more suitable for our task, since chemical compounds also represent clearly distinguishable and comparable data structures.

Based upon ReSimNet, the main contribution this work wants to build upon, was part of the application project of the University of Applied Sciences Kiel. The paper called HazelNet has investigated the development of an application for correlation prediction of signals depicted as images [13]. In this work, artificial signal images were created based on tabular ECG data, representing measured heartbeat signals. Since the source signals have been available, correlation was computed based on time series data and used as a label for validating various image processing techniques. First off, simple methods like taking image means or hash calculation had been applied, however no promising results were found. Therefore, more sophisticated procedures were used, resulting in a neural network for image recognition. The network, called HazelNet (analogous), built on top of a Siamese network architecture as well, utilizing the concepts of convolutional networks and a transfer learning methodology with resnet18, an 18 layer deep architecture developed by Microsoft Research, published in 2015 [14]. The network was trained, validated and tested on 17.348 image pairs in total, following a 70,20,10 splitting rule. Figure 5 illustrates the final setup, which used a transfernet layer with 11.170.240 parameters, followed by a linear layer with 12.721.280 parameters, resulting in a 1x128 vector for each of the two input images. In order to compute a correlation score, cosine similarity distance (adapted from the ReSimNet paper mentioned above) as metric was applied. As evaluation criterion mean squared error (MSE) has been used.

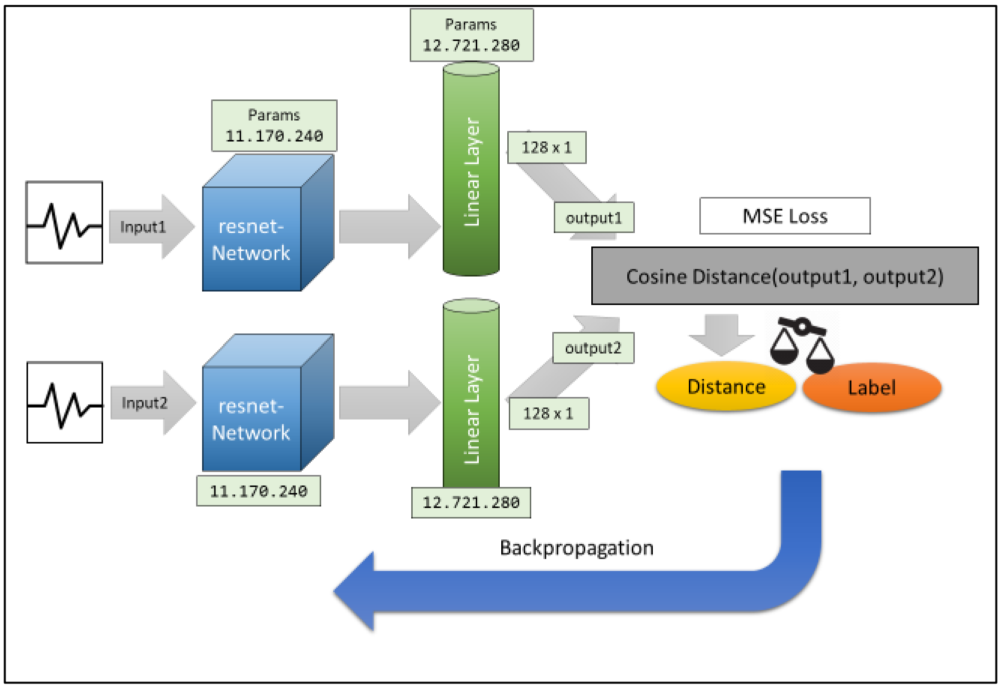


Figure 5 Architecture HazelNet.

After training for 100 epochs with the setup depicted above, final evaluation on the test data set yielded a MSE score of 0.01 and a R2 score of 93.57%. However, no experiments on images from the laboratory have been conducted, thus no information about real world application is present. Therefore, we want to build on top of this approach improving the overall structure as well as the data source used, applying it to ECG images from the laboratory. Moreover, we want to reduce network size, since we believe that the number of hyperparameters is oversized for the complexity of this problem, resulting in severe overfitting.

Since the core data of our research concerns signals which are represented as images, we want to utilize further work, which deals with the perception and representation of time series data. The paper "Visual Time Series Forecasting: An Image-driven Approach" published in 2021 by Sood et al. [15], focuses on developing a method for time series forecasting using only images. The authors investigate how well time series prediction based on computer vision can perform when compared to traditional time series forecasting based on numeric time series values. The paper compares traditional time series forecasting algorithms, ai based time series forecasting and ai visual time series forecasting. As data source, different datasets are used, however we want to emphasize on experiments and corresponding results retrieved from the MIT-BIH Normal Sinus Rhythm Database. This database contains ECG data measured from 17 different people [16]. Although the authors do not execute any kind of similarity comparison but are rather investigating forecasting prediction quality, the results of this approach are very promising and we believe that we can adapt this structure for our purposes (for details on adaptation see chapter 5).

3 Data Generation and Acquisition

As mentioned in the introduction of this work, the next step after defining our goal, is selecting appropriate target data. Original EGM images do not provide any means of labelled correlation value, therefore it is impossible to use these images for application building or supervised training of a neural network. Hence, domain experts will have to use these images in the end, to evaluate based on domain knowledge, if our predictions are helpful. In order to obtain labelled target data, we need to utilize surrogates, which we will describe in the following section. The main goal of these surrogates is to mimic real case EGM images as closely as possible, providing a good approximation of the actual target data. In this section we want to introduce two different techniques, one utilizing data generation, while the other focusses on data acquisition. The first approach will build on top of data collection already used in HazelNet. We will adapt and improve this technique introducing new insights. Images obtained from this procedure shall be called artificial images (AI images). Moreover, we want to experiment with a second type of artificial data, which will be generated in a similar manner, albeit with more variation of hyperparameters. Hence, we will distinguish between uniform artificial data (ai uniform) and random artificial data (ai random). The second approach wants to target images which we do not generate ourselves via a plotting library, but rather focus on images acquired from the laboratory software LABSYSTEM PRO[2]. These images are still based on ECG data, however we perceive their nature to be more similar to our final data type, since electrophysiologists use these images in everyday work for correlation computation and informed decision making. Data obtained from this procedure shall be called real data (or ECG data), mimicking the actual area of application as close as possible. Figure 6 illustrates the different data types used in this thesis:

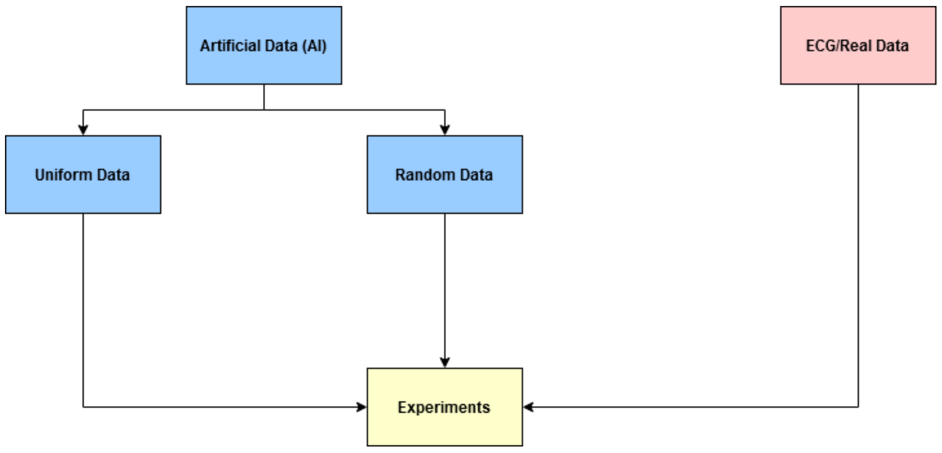


Figure 6 Target Data Types.

For the basis of correlation value computation we start off with the procedure used in previous research, which applied Spearman's R coefficient for determining label values. In the beginning of this work we were told that the algorithm used by LABSYSTEM PRO was Spearman's R correlation coefficient as well. However, during the first runs of our experiments we could observe that our results were significantly better when using Pearson's R coefficient instead of Spearman's, as suggested. This resulted in a thorough investigation of the software LABSYSTEM PRO. After passing our question to Boston Scientific, we received a reply, confirming our results[3]. The algorithm used is indeed Pearson's

[2] For more information on LABSYSTEM PRO see [17]

[3] The original answer by the manufacturer can be seen in Appendix I

R coefficient and not Spearman's as originally suggested. Therefore, we will focus on this new knowledge, abandoning the old computation method.

3.1 Understanding the application domain

After giving an introduction to the application domain of this thesis, we want to take a closer look at the signals which are measured and have to be compared. Therefore, we want to examine the most relevant part of electrophysiology needed for our experiments: the human heartbeat. When measuring and plotting the human heart beat, the depicted curve is called an electrocardiogram (ECG), analogous to the device used for measurements. It is separated into several different parts which together form a repetitive pattern, known as the sinus rhythm. For healthy individuals this pattern repeats between 60 to 100 times in a minute, known as the heart frequency [18]. Heartrates below 60 beats per minute are denoted as sinus bradycardia [19], while rhythms above 100 beats per minute are known as sinus tachycardia [20]. Both tachycardia and bradycardia are considered a form of heart arrythmia [12]. In addition to a slow or fast heartbeat the shape of a single beat can vary as well. Each heartbeat is made up of 4 parts[4] [22]:

- P wave: This wave represents the beginning of each heartbeat. It is the propagation of excitation in the atria[5] of the heart.
- QRS complex: A sharply serrated complex, which is the depolarization of both heart chambers.
- T wave: It is caused by the regression of excitation in the ventricles[6] of the heart.
- U wave: An inconstantly occurring elevation after the T-wave.

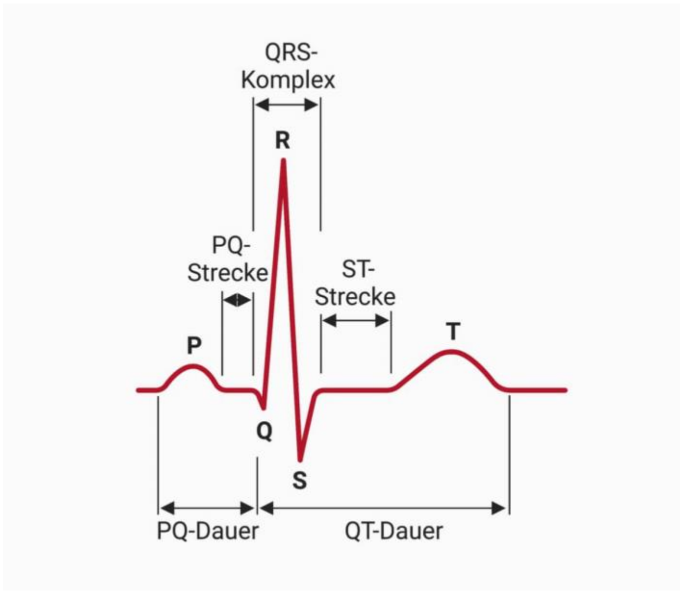


Figure 7 Human Heartbeat [22].

Figure 7 shows three of the four parts introduced, omitting the U wave, because it is irrelevant for our research purposes. When talking about signal similarity and matching, we want to focus on the QRS complex, since it is widely used to analyze heart arrythmias. Thus, all correlations calculated will be mainly based upon the comparison between two QRS complexes. The negative deflection in the beginning of a complex is called Q wave. It is followed by a large positive deflection, the R wave.

[4] For a detailed description of the anatomy of the human heart see: [21]
[5] Atria represents the front part of the human heart. Each heart consists of a right and a left atrium. For more information see: [23]
[6] Ventricle and heart chamber are synonyms

Afterwards, a second negative deflection occurs, named the s wave. Together these three waves from the QRS complex. In adults the normal interval of this complex (from q wave to s wave) lasts up to 100 milliseconds.

3.2 Generating Artificial Data

The original HazelNet has been trained on images created from 12-lead ECG and endocardial bipolar and unipolar signals, which had been provided as tabular csv files. In total 11 files have been used for artificial data generation. The data was extracted from electrophysiological mapping system where to each anatomical point there are 2500 ms recordings of several cardiac signals, resulting in 72 different columns per file. Table 1 depicts an exemplary extract of the first eight signals:

ECG_Export_4.0
Raw ECG to MV (gain) = 0.003000
Unipolar Mapping Channel=M1 Bipolar Mapping Channel=M1-M2
Reference Channel=CS1-CS2

M1(1)	M2(2)	M3(3)	M4(4)	CS1(11)	CS2(12)	CS3(13)	CS4(14)
123	82	80	77	37	32	45	54
118	77	75	73	33	29	42	51
114	72	71	69	28	25	39	48
108	67	66	66	23	21	36	45
102	63	62	63	19	19	34	43
97	58	60	61	17	18	33	43
93	55	58	60	16	17	33	42

Table 1 Extract signal columns from one file.

Each row represents a measured value in millivolt (mv) with the row index corresponding to the time t in milliseconds (ms). Thus column M1(1) shows an initial value of -140 mv measured at $t=0$ ms in the first row, while the second row shows a measured value of -135 mv measured at $t=1$ ms. These digital signals were plotted as curves using the python library matplotlib. Afterwards, random samples have been created by pairing up two different images and labelling them based on their corresponding digital signals, using Spearman’s R correlation coefficient (which we now will change to Pearson’s R). For creation of this data, several hyperparameters and assumptions like signal length of the QRS complex, zoom factor, image resolution etc. were made, which we want to improve on in this work. As mentioned, we want to compare artificial data generated with uniform hyperparameters to artificial data generated with varying hyperparameters, investigating how well a neural network can learn to predict correlation scores on our real data, given training data variant to some degree. Our hypothesis is, that we expect artificial data with more variance to represent real data more accurately.

In contrast to the method used in HazelNet we want to refine the creation of artificial data, using further background knowledge from the laboratory. This means that all assumptions and values used, are based upon recommendations or best practices from domain experts in electrophysiology. Although we did discuss and adjust some of these assumptions, we will use the knowledge provided.

3.2.1 Uniform Artificial Data

For the source of our data we will use the same origin as before, albeit now choosing 20 ECG files collected. More data is not needed, because each column represents a signal with the length of 2500 datapoints, corresponding to 2500 ms. Since a healthy QRS complex is usually around 100 ms long, we want to limit the depicted measurement length of our signals to 140 milliseconds, instead of the 200 ms used originally, thereby leaving enough window size for prolonged erroneous signals without the risk of capturing two complexes at once. This means that each column can yield at most 17 separate signals. Since each file contains 72 individual columns we are left with an upper bound of $20*72*17 = 24.480$ signals. Considering each signal can be paired up with every other signal the maximum number of signal pairs is: $24480*24480= 599.270.400$.

3.2.1.1 Peak Detection

The main reference point of each signal shall be its r wave (peak). Therefore, we want to determine the peaks in any given column, using them as the center of all extracted images. For finding the peaks we use the function “find_peaks” from the scipy library, analogous to the approach taken in HazelNet [24]. This library specializes on signal processing and also works with signals from electrograms. However, in contrast to earlier approaches we consider negative and positive peaks alike since signal measurement can be inverted depending on the measurement axis used. Therefore, each signal is squared. This magnifies large peaks more than smaller ones. To adjust for this change, we alter the prominence of a peak to 20% of the maximum of the squared signal. The prominence of a peak is defined as a measure of “how much a peak stands out from the surrounding baseline of the signal and is defined as the vertical distance between the peak and its lowest contour line” [25]. This value was suggested as a common procedure in electrophysiology. Figure 8 illustrates how a lowest contour line is established for different peaks in one signal curve (corresponding to one column in our tabular data). For the minimal height of a peak we use 10, which means that all values below 10 mv will not be considered.

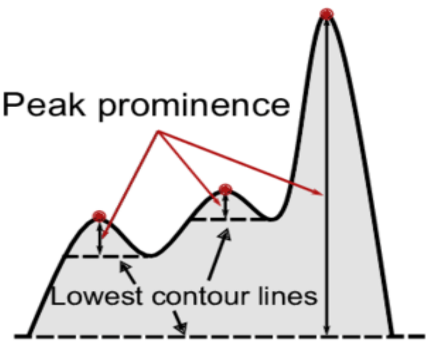


Figure 8 Peak Prominence [17].

After finding all peaks of a signal, a noisiness threshold (in our case 17) is applied, discarding signals with more than 17 peaks, since we expect each signal to contain at most 17 (2500 / 140 = 17.86). With the indexes of found peaks at hand, we can extract the data from the original, unsquared signal. We use the pre-defined distance parameter (set to a length of 70ms) to cut out signal parts with the detected peak in the middle and respectively many values to the left and right (+/- distance). Figure 9 shows detected peaks in a curve, marked with a little x at the top of the peak. We can observe that our r wave is correctly identified, while the subsequent t wave is ignored.

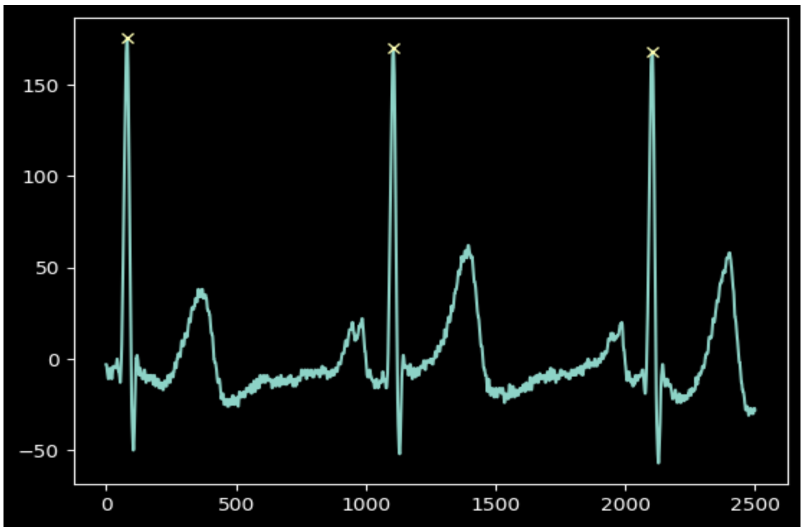


Figure 9 Detected Peaks of a signal.

3.2.1.2 Signal Amplitude

Now that all detected peaks with their corresponding signals are extracted, we want to analyze the amplitude distribution of our signals. Figure 10 (on the left hand side) contains a boxplot showing the overall amplitude distribution in our data. We can see that the amplitude values vary widely. Ignoring the outliers with values of up to 5000 mv, we have a difference of roughly 2000 mv between the last and the first quantile. If we drew all signals on a unified scale, some signals would be up to 200 times smaller than the largest possible depiction (assuming a min height of 10 mv and a max amplitude of 2000 mv). In practice, this would lead to some signals being not recognizable, showing as white straight lines because of the scale.

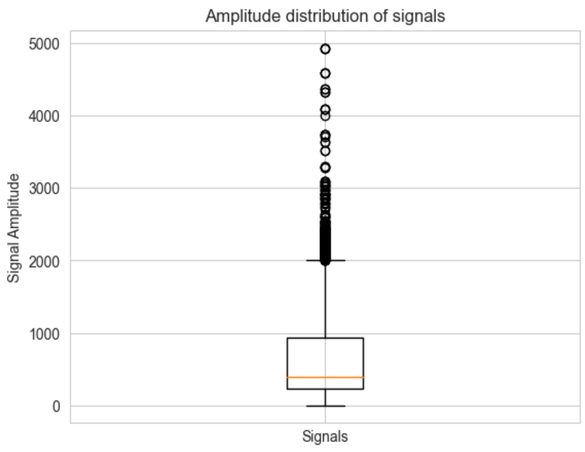


Figure 10 Amplitude distribution before reduction.

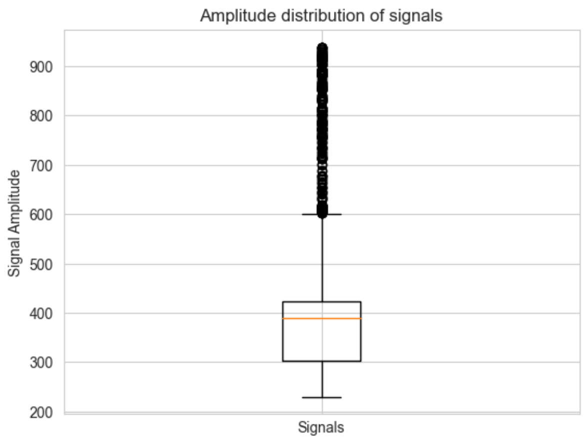


Figure 11 Amplitude distribution after reduction.

We do not want to scale the entire time series, since this would affect the Pearson coefficient, which is sensitive to absolute change in value. Instead, we will omit very large and very small amplitudes, only keeping signals between the 25-50 percentile. This results in the plot depicted in figure 11 (on the right hand side) with most signals ranging between 250 mv and 600 mv amplitude and 50 percent of the data being roughly between 300-420 mv. However, we still encounter many outliers, denoted as black points above the upper whisker of our plot. Thus, we choose to remove this data in a second preprocessing step, leaving only signals whose amplitudes are in between the boxplot whiskers (lower to upper quantile). This results in an overall signal amplitude difference of three, which means that the largest signal amplitude is about three times as large as the smallest one in our dataset. After aligning the signals amplitude, we end up with 3729 signals. We believe this amount to be sufficient, yielding $3729*3729 = 13.905.441$ possible signal pairs, with reasonable variance in amplitude data. Figure 12 shows our final amplitude distribution via boxplot.

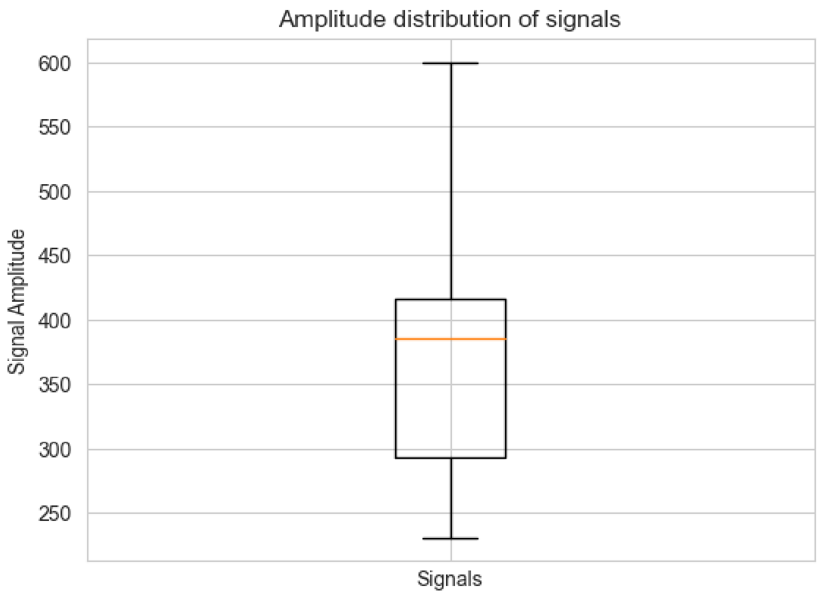


Figure 12 Final amplitude distribution.

3.2.1.3 Generating Images, Pairs and Labels

Once all signals have been created we now commence image generation by iterating over each signal using the libraries seaborn and matplotlib to draw the signal. We choose to draw all signals as a white line on a black background and will set the settings accordingly. The main advantage of this setup is that pixels which do not contribute to our signal are automatically set to 0 (pixel value black), while pixels that depict the curve get the highest possible value 255 (pixel value white). This means, that all images used, need to be adjusted for this setup (using greyscaling and inverting if needed). We set the y-axis limit to the lowest and highest y value available in the dataset, reflecting the highest possible amplitude, thereby scaling all depicted signals to equal size. Considering usual line plots, each plot starts off with a border around the figure and two axis with corresponding ticks, providing better oversight. However, since we want to look at images depicting QRS signals, the real case scenario will not contain such information. Hence, we decided to remove any black padding as well as plot axis, such that each image immediately starts off with the white line representing the signal. Finally, we save each image as a png file, using the dimensions provided by the parameter size, which we set to 224x224 (because these are the dimensions used by most large neural networks, like resnet, alexnet etc.). Each image is saved in the folder *trainingimgs* under the name *img**id**.png* where id corresponds to the signal number it is based upon, e.g. (img0.png for the first signal). Analogous to the number of signals we end up with 3729 images.

Next we proceed with creating our pairs and labels. We begin by picking two signals at random from our signals list. Afterwards, we compute the correlation label (Pearson's R coefficient) using the corresponding time series values, saving the calculated label, as well as the indices (position in the signals list) of the two signals picked. We execute this algorithm until all possible signal combinations have been paired. We end up with a data structure containing the indices of all paired up signals as tuples and a list of all labels calculated. This means that at position 10 in our list we might find the ids (100,500) corresponding to the signals at index 100 and 500 in the signals list. The associated label can be extracted from the labels list at position 10, while the images can be found by loading img100.png and img500.png from our images folder. We perform a permutation with repetition ending up with $3729*3729 = 13.905.441$ pairs and labels.

3.2.1.4 Label Balancing

Looking at figure 13, we can observe that our artificial dataset is imbalanced, with significantly more high correlation values (up to 2.500.000) than lower ones (e.g. roughly 20.000 for correlation value 0). Therefore, we want to apply a balancing technique, providing a well-proportioned dataset.

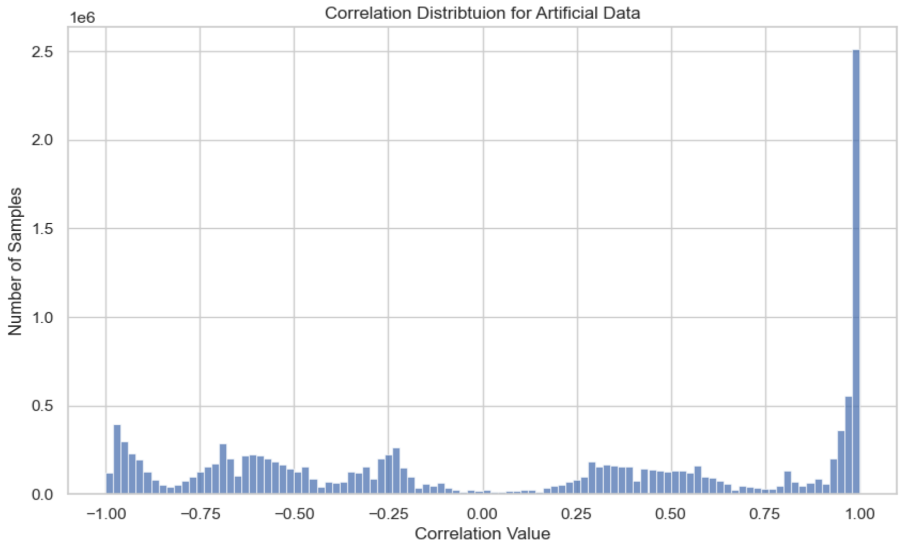


Figure 13 Correlation Distribution of Artificial Data.

Since continuous data is hard to balance, we need to create some form of data discretization. We choose to apply binning, thus creating equally sized bins holding labels within a certain range. All labels within one bin will be considered equal with regards to class imbalance. We pick a bin size of 1%, hence each bin will be representing a 1% window (e.g. 0.99-1.0). After the binning, the maximum threshold is determined by finding the group with the lowest members. All other groups will be downsampled to resemble the same number of data. Executing this procedure we can observe that our smallest bin yields 6027 samples. Based on computational limitations we choose to reduce this threshold further, down to 50 samples per bin. This results in a uniform distribution with regards to our chosen bin size, providing 10.000 samples (200 bins * 50 samples per bin) in total.

Once data generation is completed, we can save the list of image pairs and the list of correlation labels as two separate numpy arrays called *imageLabelsPearson.npy* and *imagePairsPearson.npy* in the folder *trainingData*. The main advantage of using this data structure instead of the one applied in HazelNet, which saved each image for each pair individually, is a significantly faster processing time. Since we have much more pairs than images, saving and loading each image only once, reusing it multiple times by index reference from the pairs array, saves computing power on this expensive operation.

3.2.2 Random Artificial Data

In addition to the procedure described above, we will alter the creation of our artificial data to account for more randomness. First, we will change the length of a signal by randomly selecting the distance taken from a given peak between 50 and 100 data points. This means that the overall signal length will vary between 100 ms and 200 ms with 51 different lengths present (100, 102, 104 … 200). Next, we change the size of an image created, choosing a random value for the width and the height of the image (between 50 and 200 pixels). In addition we will adjust the y-axis limit, thereby scaling the depicted signal between 100 and 250% (zooming effect). Note that in-between any image pair given, we still want to keep each template and match image similar, having the same underlying signal length, size and scaling factor, since we believe that changing these dimensions in-between an image pair could proof a task too difficult to solve due to lack of a reference frame.

In order to achieve this we start by ensuring equal underlying signal length of any image pair given. All created signals are binned by the new randomized signal length. Next, bins which yield more than 100 different signals are kept, while the rest is discarded. In addition all remaining bins are filtered to contain exactly 100 signals chosen at random, which results in a uniform distribution. To ensure that we have enough data present to provide 100 signals per binned signal length, we repeat the process described above with 20 different seeds, thereby altering how our randomized parameters are picked per signal. Looking at the histogram depicted in figure 14, we can see that not every chosen bin length had sufficient data generated. Out of 51 signal lengths, 49 can be used. However, this result is sufficient for our training purposes, therefore we can proceed safely.

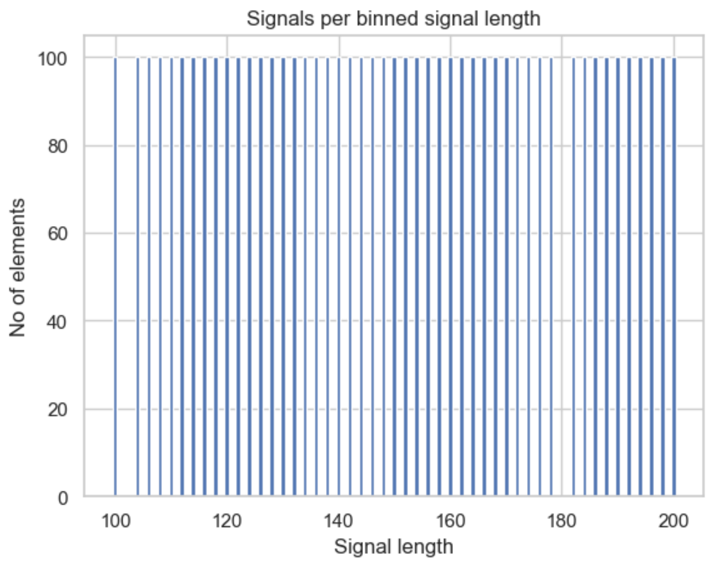


Figure 14 Signals per binned signal length.

Afterwards, labels and image pairs are generated as described by the uniform procedure above, albeit only using signals from the same bin. This results in 10.000 pairs per bin (100x100) which yields 490.000 signal pairs in total, since we have 49 bins (for distances from 50 to 100). After leveling out our new data with respect to a uniform correlation distribution, using once again a threshold of 50 signals per bin (1 % bins) we end up with 10.000 (200 bins*50 signals per bin) image pairs. Note, that this setup results in a random distribution of signal lengths in the final dataset. Now, that we have created our signal pairs and their corresponding labels, images can be generated. One drawback of this method is that we need to physically create each image, since although two images might reference the same signal, their size and scaling parameters might and should differ. Therefore, we end up with 20.000 images, which we numerate in an ascending manner. Every odd and even subsequent image pair represents a template / match combination. Thus our image pair entries represent simple ascending numbers from 0 to 19.999. We store our randomized images in a separate folder called *trainingimgs_random*, while pairs and labels are stored in numpy arrays called *imagesPairsRandom.npy* and *imageLabelsRandom.npy* respectively.

We want to point out that in earlier versions of this thesis different correlation coefficients, like Spearman's R and comparing the area under the curve of both signals (area match) have been researched. While Spearman's R coefficient can safely be disregarded due to information provided from the manufacturer, the general usefulness and application of area match should be discussed and might be topic of future investigation. Figure 15 depicts some sampled signal images with the respective correlation value provided:

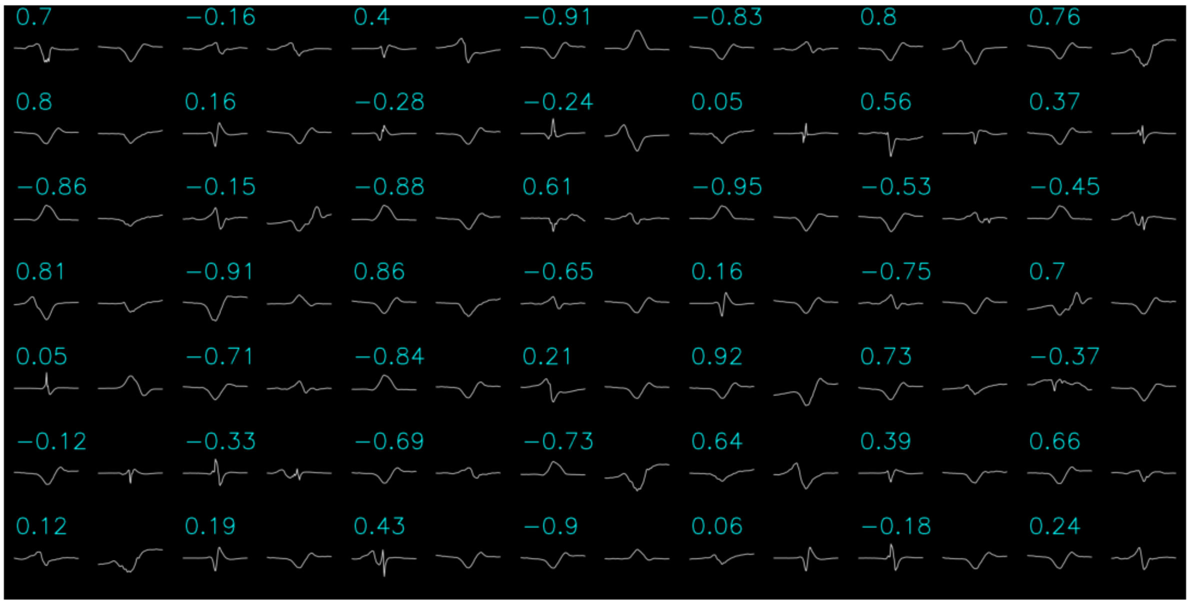


Figure 15 Sampled Artificial Signal Pairs.

3.3 Collecting data from the laboratory

After finishing artificial data preprocessing, we now want to adapt the techniques used, to our ECG target dataset. As mentioned the main source for lab images will be "LABSYSTEM PRO" a system made by Boston Scientific [17]. This software uses the ECG put on during operations to record and analyze signals. One of its several features is providing the opportunity to measure, save and reload patients ECG data, representing it via a dedicated screen. These signals can then be processed as a running timeline, where an arbitrary spot can be selected in order to determine a template and calculate a similarity score with another signal sequence. This procedure is used by doctors to determine signal similarity during operations. However, one of its major drawbacks is not being able to compare recorded arrythmias via pacemaker EGMs with lab measurements since the measurement axis of the ECG is different due to its location in the heart. Therefore, a new ECG template needs to be created on site which yields the same problems as described during the introduction of this work[7]. Thus, it is not always possible to acquire a correct template of the patients arrythmia to make valid comparisons. Nevertheless, we can use this data to approximate images obtained from EGM devices with the large benefit of having a label provided by the system.

Images are acquired by loading a patients ECG recording from the LABSYSTEM PRO archive. Each recording represents a timeline which can be scrolled through, showing how the signal has changed during the recording session. Figure 16 shows such an imported file. In order to create a template, an arbitrary area of the time line can be selected. This part is marked in red by the system and will be displayed on the left hand side of the screen. Next, a matching area is chosen from the ECG track and displayed on the left hand side. Based on the length of the chosen template window, the area with the highest overlap within the marked frame will be chosen for correlation computation (marked with a

[7] Finding correct arrhythmia under many / not being able to induce arrythmia due to complications

yellow line). This means that when choosing a window for comparison with an already predefined template, we can choose window sizes larger than the one of the template (template window e.g. 169 ms), since within this larger area an equal sized window with the highest possible correlation will be found and marked (e.g. best 169ms within marked 300ms). The system supports selecting several matching areas, thus one template can be compared with several time series sections simultaneously.

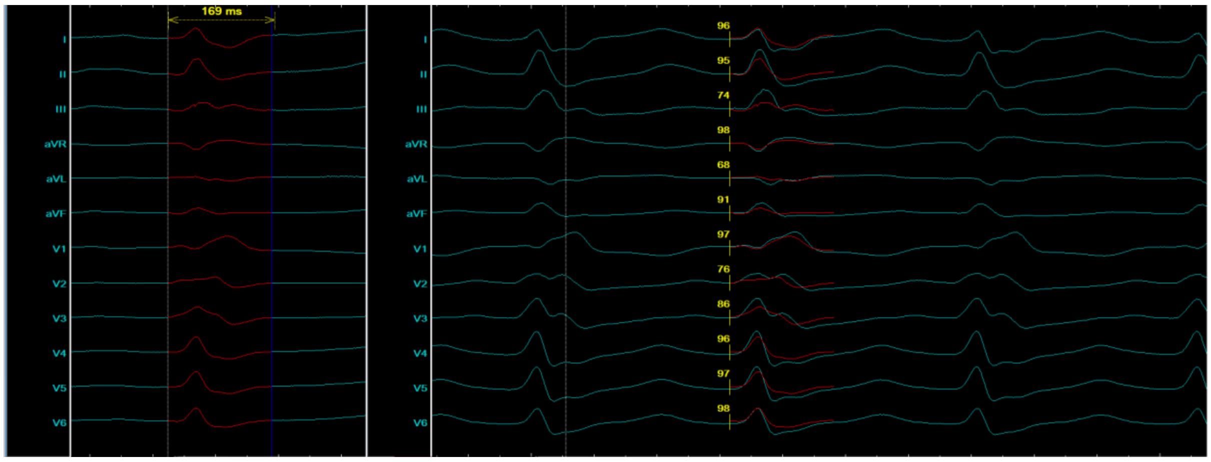


Figure 16 ECG Screenshot.

Every matching section chosen, provides 12 new signal pairs, since we are using a 12-pole ECG. Each signal is represented via its own line, which can be individually scaled to provide a better resolution. Therefore, it is important to pay attention to signal scaling between template and marked window of one channel. Scaling's can vary between different channels but should be checked for the same channel. Otherwise corrupt data is acquired. The system always uses the underlying timeseries data, which means that scaling does not affect the label computation, however it may lead to template and match inconsistency, if one is scaled larger than the other, which will either disrupt the image (overlap of signals) or disrupt the application (template and match visually not comparable anymore). Figure 17 illustrates how unfavorable scaling settings can disrupt the collection process since no clear signal can be extracted.

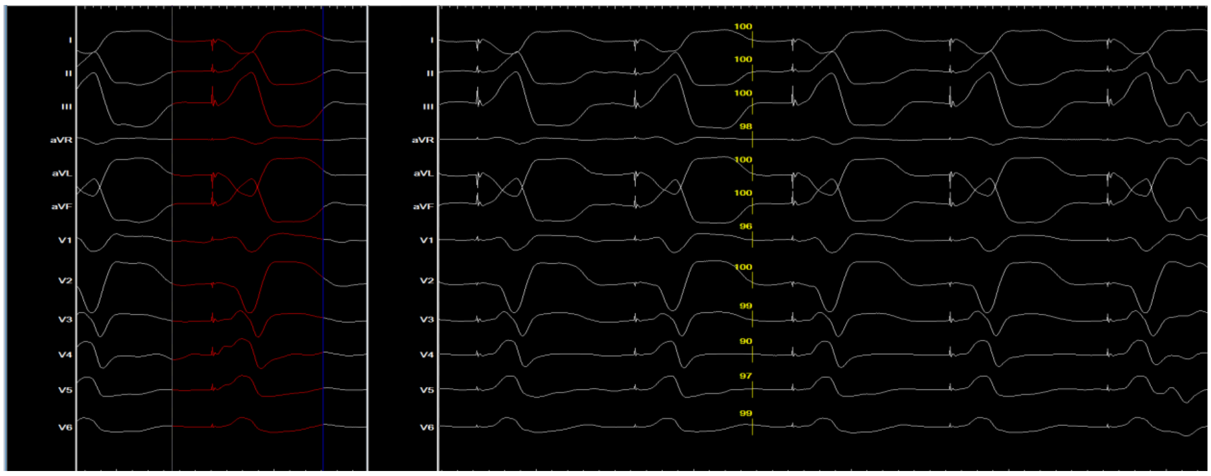


Figure 17 Unfavorable Signal Scaling.

In order to extract and process the ECG recording, we capture each segment with template and match section(s) selected by taking a screenshot on the system. To transfer each screenshot into 12 separate pairs we will utilize two different data processing techniques, one involving more manual steps, the other applying a more automated procedure.

3.3.1 Manual Image Extraction

The first approach uses image editing software, e.g. adobe photoshop. Figure 18 illustrates the preprocessing procedure used. We start by cutting out the two main parts containing the template and match signals. This is achieved by defining a cut out window based on the area marked by the red curve. Next we move this window such that it starts off at the yellow marker with the correlation coefficient. Thus we get two cut outs which are identical in width and height. Afterwards, an automatic grid is defined, separating our two cut out sections into three vertically (one template, one match and one for space in between) and twelve horizontally divided sections representing the different signals. We align the horizontal grid such that each signal fits entirely into the grid lines which will result in some cut outs being larger than others. Once this process is finished we can export the grid resulting in 24 pictures 12 template and 12 match images. These files are saved into a folder and enriched by a csv file containing the yellow marked correlation labels as inserted values. This has to be done manually for each file. Using this method we ensure that each image pair has equal width and height while width and height in general will vary between different image pairs.

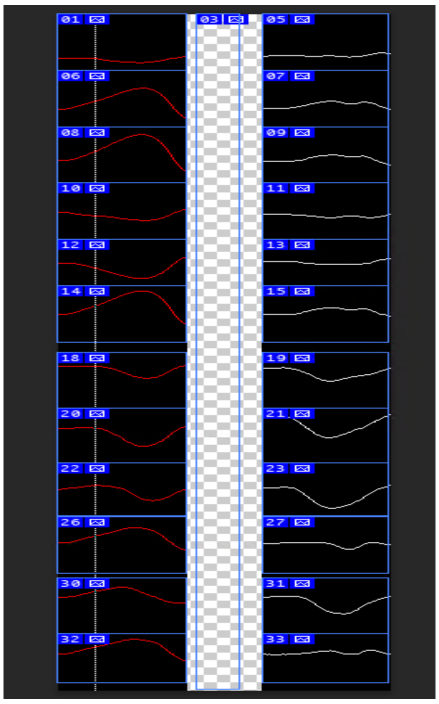


Figure 18 Manual Signal extraction.

3.3.2 Semi-Automatic Image Extraction

Since the usage of image manipulation software requires some knowledge and is time consuming we have experimented with a second data acquisition approach. In this scenario only two cut outs are generated (one for the template one for the match). This task can be executed with every image editing software at hand. Afterwards we use an algorithm which will grey scale, combine and resize each cut out pair , yielding a new preprocessed "screenshot" with the dimensions 400 x 1200. Thus, each screenshot has 100 pixels in height per signal pair (1200 / 12) and 400 pixels in width. Template and match section will have each a width of 200 pixels (400 / 2) accounting for 200ms template and 200 ms match signal. Although we have defined 140 ms as the desired timeframe for identifying QRS complexes, in practice, due to operational inexactness, some signals measured are between 140 and 200 ms long. Note, in an ideal environment no signals longer than 140ms should be considered. However, since this threshold seems to pose a challenge in data collection, we decide to allow for higher values up to 200ms as already used in previous work. This choice corresponds with the maximum signal length threshold used in the randomized data generation (chapter 3.2.2). Next, we iterate over each new screenshot row by row, analyzing the sum of pixel values. Sums larger 0 indicate the start of a signal (its peak) which we use as our threshold. From this position we subtract 25 pixels (up) to get our start index and add 75 pixels (down) to get our end index. We choose this division since a signal might have some downward peaks which could lead to a wider range towards the end of our selected window. This process is repeated until the end of the screenshot is reached. Finally each detected section gets split in the middle (column wise), returning again separate images for template and match. Thus, we can generate our grid in an automatic fashion. However, the major drawback of this process is that we cannot adjust the height of an individual window, which leads to some signals being cut too early or picking up on signals from the next channel. Moreover, the threshold of values larger 0 for identifying the start of a signal can be erroneous, especially if shades of black are involved in an image. This might lead to falsely classifying black area as the start of a signal. Hence, some images need to be discarded and redone manually. Labels are entered into a csv file by hand. Note, this procedure ensures once again that width and height between each template and match pair are identical.

Overall, for some sections the signal quality was insufficient, e.g. depicting overlapping signals as shown in the figure 17. In these cases less than 12 pairs could be extracted from the screenshot since some pairs could not be processed. After finishing both collection processes, we end up with 556 image pairs in total.

3.3.3 Balancing Real Data

Acquiring pictures from the laboratory is a fairly difficult task, because we are restricted to specific time windows which are after the regular operation hours, but before the cleaning teams come in. These windows usually cover a timespan of less than two hours which sometimes might be significantly shorter due to overtime work in the operating room. This results in a significantly smaller dataset, when compared to the artificially generated data. In addition, it is not easy to collect data which is evenly distributed since the system described above always picks matches within a time window with the highest correlation. This leads to more images with high correlation values than low or negative ones and results in class imbalance. Due to the smaller dataset size we need to adapt our balancing procedure, since using a 1% bin width would not be feasible with some bins containing only a single image pair. After consulting domain experts, we choose to alter the algorithm, using a bin width of 10%. Moreover, bins are shifted in favor of smaller values, by defining the first bin with values ranging from -99 to -89 since no data with a negative correlation of -100 exists in our samples. However, if future research should replicate our experiments, labels with this value would be perceived as belonging to the bin -99 to –89. In order to retain an equal spacing between bins, the largest bin group is shortened, including only values from 91 to 101. Because values above 100 are impossible, this group will be smaller in width than the other ones, however we believe this adjustment to be reasonable since much more data with high correlation (100 or 99) exists. Note: In order to keep artificial data similar to real data preprocessing we choose to omit the bin -100 for artificial data generation as well. Figure 19 depicts the distribution of the binned image pairs acquired:

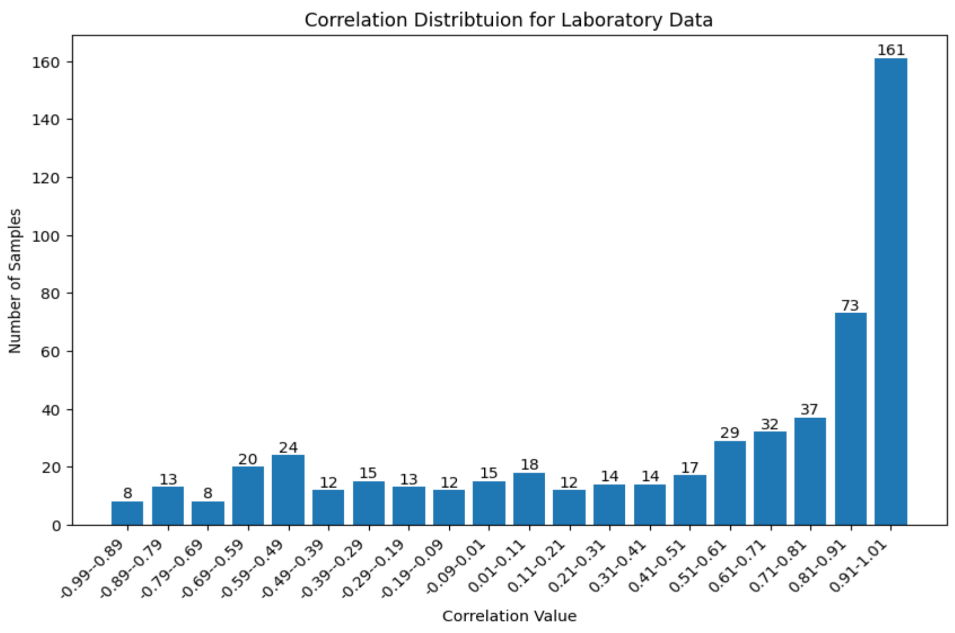


Figure 19 Correlation Distribution of Laboratory Data.

After creating the bins we can observe, that the smallest bin size contains only 8 image pairs, while our largest bin accounts for 161 data points. Therefore, instead of utilizing downsampling as shown in the preprocessing of artificial data, we will apply oversampling to balance out our dataset. We use our largest group as our threshold and duplicate samples from all other smaller groups until equal size is reached. Thus, our smallest group with only 8 pairs will receive 153 duplicates. This procedure yields a uniform label distribution, albeit with many duplicates in particular bins. Note that oversampling is only applied to training data, validation or test data will not be oversampled.

Summing up our preparation process, we end up with a structure as follows: We have one main folder called *ECGcutouts*, which contains three subfolders. The first folder *manualimgs* contains all source images for manual processing as well as their corresponding image software files. The second folder *automaticimgs* contains all source images for semi-automatic processing as described in chapter 3.3.2. The folder *realimgs* contains all processed images from both methods, with as many subdirectories as source images processed. Each subdirectory is simply represented by an integer value and contains itself up to 24 “png” images, which use the prefix of the folder and an independent counter starting at $i=0$. Every two subsequent images represent an image pair of template and match. E.g. when looking at the folder 10, we have images 10_01, 10_02, 10_03, 10_04 etc. with (10_01,10_02) forming the first pair (10_03,10_04) forming the second pair and so on. In addition each folder contains a csv file “labels.csv” which contains the true correlation labels, as comma separated integer values ranging from -100 to 100.

4 Data Preprocessing

Now that we have established the types of target data that we want to use and generated the corresponding surrogate datasets, we want to take a closer look at the general preprocessing steps necessary for any input image pair. This chapter aims at establishing universally valid preprocessing methods, which will be included in the final application, applied to every image pair processed regardless of its source (surrogate artificial, surrogate ECG or live during operation).

4.1 Handling Signal Disturbances

Taking a closer look at ECG data, we can observe that some images depict a white dotted line which is placed vertically across the signal. Figure 20 shows such an example:

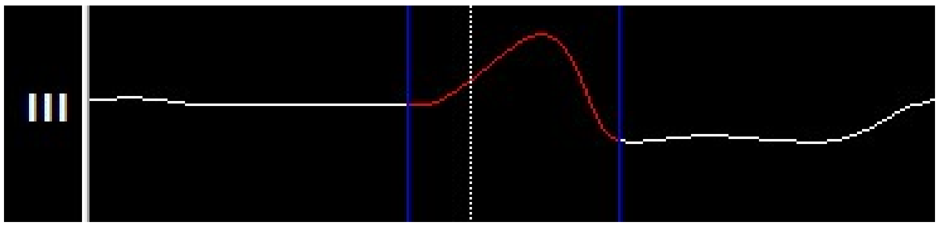


Figure 20 Real data with vertical dotted line.

Observations from the laboratory showed these types of disturbances occurring not exclusively on ECG data, but on EGM pacemaker images as well. Therefore, we have to assume that our system will face such disturbances in go live depending on the EGM device used. In order to avoid a negative influence on the prediction quality of our application, a method is needed for removing such interferences beforehand. One possible technique for removing unwanted structural elements from an image utilizes a concept called morphological transformations. In the following sections we want to provide a short digression into this transformation technique and explain how we can apply its concepts to mitigate this issue. Before introducing morphological transformations in general, we would like to present a binarization procedure, which separates every image into two separate pixel intensity classes, black pixels (0) and white pixels (255). This step is necessary, since all morphological transformations are designed to operate on binary images.

4.1.1 Thresholding

A common procedure used for such binarization tasks is thresholding. In thresholding a single threshold value is picked, all pixel intensities below the threshold will be regarded as noise and set to 0, while all pixel intensities above (or equal to) the threshold will be set to 255 (being the highest possible pixel value). In our first approach, we chose 127 as our threshold value since it is right between 0 and 255. However, this setting did not yield satisfying results on our laboratory data, leading to most of the images losing their information, only showing a blank black screen. Taking a look at the distribution of pixel intensity means for our real target data, depicted in figure 21, we can see, that on average our images have a pixel intensity mean of 2.96. This is a rather low value, which might be based upon monitor settings when taking the screenshots from the display. Taking into account this rather low intensity distribution, it is clear why our first approach did not yield successful results. Using 127 as our threshold value resulted in all pixels being set to 0 in each image.

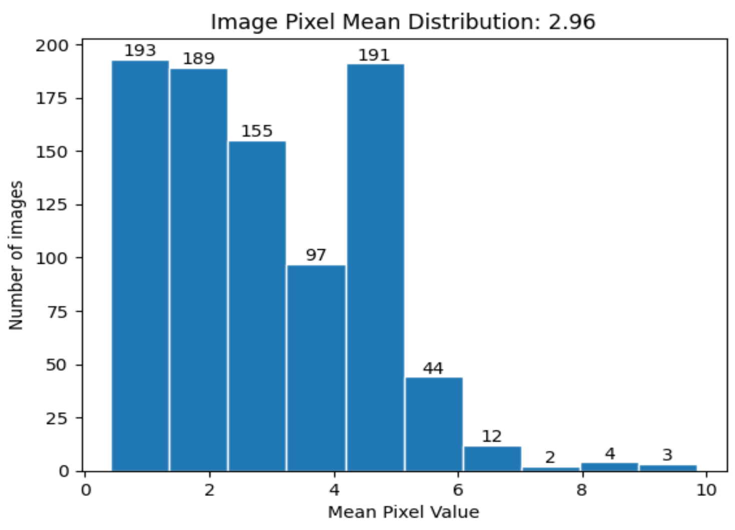


Figure 21 Distribution of pixel intensity means.

One possible adjustment of our threshold procedure could therefore use a lower threshold of 2 or even 1. However, such an approach would be very sensitive to changes in display settings or screenshot software. Moreover, we want to keep the possibility of using smartphone cameras for image capturing in the future. Therefore a different, more adaptable threshold method is needed. Looking at state of the art binarization techniques, we chose to implement "Otsu's method" [26]. An algorithm for determining an adaptive threshold based on the histogram distribution of pixel intensities for each image.

4.1.2 Otsu's Method

The base principle of this algorithm is very similar to a k-means algorithm with 2 clusters. The goal of this procedure is to separate pixels into two classes such that we maximize between-class variance[8]: $\sigma_B^2(t)$ [27]. One class thereby represents the background pixels (class 0) while the other class represents the foreground pixels (class 1) of an image. Using the formulation from the original paper Otsu's method can be expressed as:

$$\sigma_B^2(t) = \omega_0(t)\,\omega_1(t)\,[\mu_0(t) - \mu_1(t)]^2$$

In this formula, $\omega_0(t), \omega_1(t)$ are the class probabilities (denoting the probability of a given pixel belonging to the respective class), t represents the chosen threshold pixel intensity value and $\mu_0(t), \mu_1(t)$ denote the mean pixel intensity of the two classes. The key idea of this algorithm starts out by building a histogram of all available pixel intensities (0-255) in an image given. Afterwards, each possible threshold value t (0-255) will be picked in an iterative procedure, dividing the histogram constructed into values smaller and larger (or equal to) than the current threshold. For each possible subdivision, the algorithm calculates the corresponding between-class variance $\sigma_B^2(t)$. Finally, the largest value $\sigma_B^2(t)$ gets chosen as the best intensity threshold for the image.

The example below (using figure 22) illustrates how for a 6x6 image $\sigma_B^2(2)$ is calculated[9]. First the related histogram is created, by counting the number of pixels with each intensity value. Because the pixel value 0 occurs 9 times in the image depicted to the left of the figure, the corresponding histogram bar on the right will have a height of 9. All other bars are created accordingly. For this simple example

[8] Since we only care about two classes, maximizing between-class variance is equivalent to minimizing inter-class variance.

[9] The following example is taken from [28]

only intensities up to 5 are present, which results in the histogram only depicting 6 bars (0-5). Next, class separation is performed. Since we chose 2 as our threshold, all pixel values below 2 are considered as background (coloring the corresponding histogram bars in red) while all values above and including two are considered as foreground (coloring the histogram bars blue). The class probabilities $\omega_0(2), \omega_1(2)$ can be calculated by dividing the total number of pixels in each class by the total number of pixels in the image, e.g.

$$\omega_0(2) = \frac{9+6}{36} = 0.42 \qquad \omega_1(2) = \frac{4+5+8+4}{36} = 0.58$$

The mean intensity of a class can be determined by multiplying the amount of pixels with the corresponding intensity and dividing the result by the total number of pixels in this class, which yields:

$$\mu_0(2) = \frac{(9*0)+(6*1)}{9+6} = 0.4 \qquad \mu_1(2) = \frac{(4*2)+(5*3)+(8*4)+(4*4)}{4+5+8+4} = 3.57$$

Thus our between-class variance for t=2 is: $\sigma_B^2(2) = \omega_0(2)\,\omega_1(2)\,[\mu_0(2) - \mu_1(2)]^2 = 2.44$

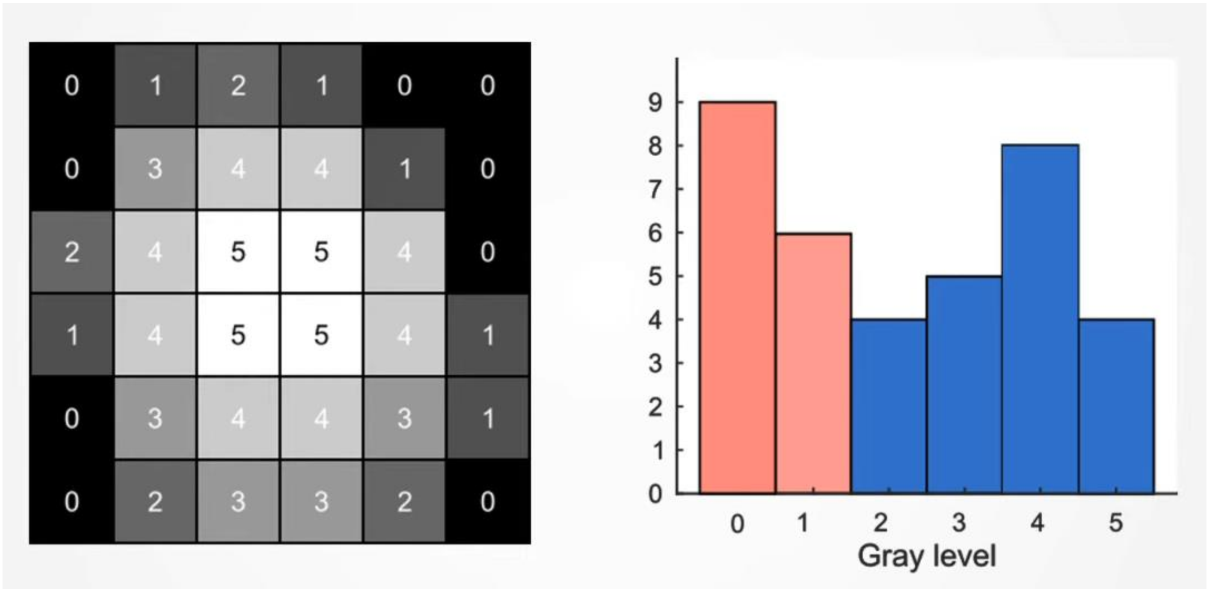


Figure 22 Example of Otsu's method, Image taken from [28].

Note, this method is still a global thresholding method, since one threshold for the entire image is calculated. Other approaches use a dynamic thresholding approach, whereby local thresholds are calculated for separate segments of the image[10]. By applying this method beforehand to an image given, we obtain a binarized version of this image, which can be used in further steps for applying morphological transformations in order to remove the visual disturbances. Note: The implementation used by cv2 computes thresholding for all pixel values smaller (or equal to) the threshold chosen, instead of only smaller. Thus, the equality condition switches sides, which results in 0 being the lowest possible threshold, instead of 1, as mentioned earlier in this chapter [29].

4.1.3 Morphological Transformations

Morphological transformations are simple operations to alter an image based on its shape. All morphological transformations rely on a kernel, which very similar to convolutional operations, slides over the image, applying the operation defined. As mentioned, these types of operations usually are designed to perform on binary images where the foreground is white and the background is black,

[10] For more information refer to [29]

which is why we will apply Otsu's method in advance. In this subchapter we want to provide a brief introduction of common morphological transformations:

The first basic operation is Erosion (figure 24). When using Erosion the resulting pixel of the transformed image will be white, if all pixels under the kernel in the original image were white. If only one pixel was zero the resulting value will be zero. Therefore Erosion is used to erode the boundaries of an image, making it slimer by transforming pixels at the border into black ones.

Dilation on the other hand represents the inverse of Erosion, transforming each resulting pixel into a white one if at least one pixel under the kernel in the original image has been white (figure 25). This yields wider images with extended borders, increasing the number of white pixels.

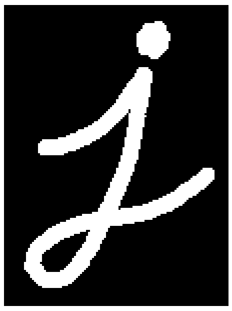


Figure 23 Original Image [30].

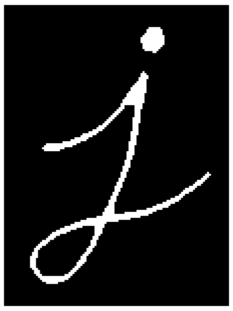


Figure 24 Erosion [30].

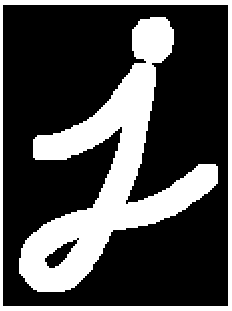


Figure 25 Dilation [30].

Applying Erosion first, followed by Dilation is called Opening (figure 26). This procedure is often used to eliminate noise in an image (Erosion) albeit retaining the original graphic by enlarging the cleaned image again (Dilation).

Using Dilation before Erosion is named Closing, this operation is utilized to close small holes inside the image, eliminating small black points inside the graphic (as depicted in figure 27).

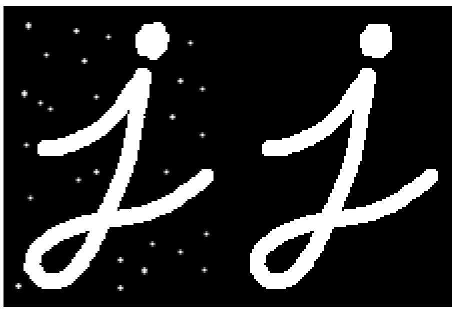


Figure 26 Opening [30].

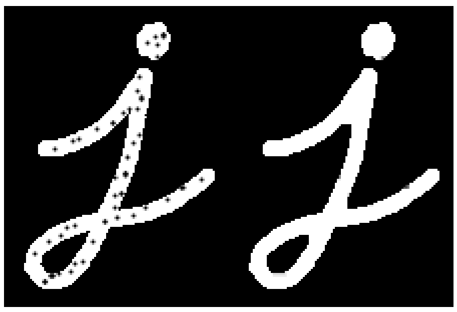


Figure 27 Closing [30].

Further operations include the Morphological Gradient (difference between dilation and erosion of an image), the Top Hat (difference between original image and opening of an image) as well as Black Hat (difference between original image and closing of an image)[11]. The shape and size of a kernel determines how a morphological transformation will be applied in the end and which parts of an image will be affected.

4.1.4 Applying Morphological Transformations

Now that we have introduced common techniques used in morphological transformations as well as state of the art image binarization, we can utilize these concepts to omit disturbances. We start off with dividing each image from our pair into white and black pixels, using Otsu's method as described above. The vertical lines in our dataset are dotted and not continuous, therefore we need to perform

[11] For further information see: [30]

a Closing operation. We choose to use a fixed kernel of width 3 and height 10, adapting the dimensions of the kernel such, that only the dotted lines are altered, transforming them into continuous ones. Afterwards, we perform an Opening operation, using a vertical kernel with width 1 and height corresponding to 80% of image height. That way only white lines which take up most of the image height are detected, thus avoiding accidental deletion of signal parts which might contain some vertical component. Note, that we target only vertical lines, omitting horizontal ones, since no horizontal disturbances are present in the dataset provided. Moreover, detecting and deleting horizontal lines would prove a task more difficult to solve, since some signals can contain prolonged horizontal components which might be detected by the algorithm. Nevertheless, if necessary, a similar procedure could be utilized to handle such disturbances. All parts identified by our Opening operation will be set to 0, creating some empty columns in the image, which need to be deleted afterwards in order to maintain a continuous signal depiction. However, we want to preserve the original dimensions of each image prior to the transformation, thus a resizing operation is needed.

4.2 Data Resizing

Depending on the type of procedure we want to use for correlation prediction, all images need to be of the same shape and size. Although our data has been generated in such a way that between two pairs no difference in size can occur, the intra shape difference between different pairs needs to be handled. Moreover, depending on the operator using our application small differences in image size can be expected due to operational errors, e.g. taking a screenshot of the EGM image which is slightly larger or smaller in size than the screenshot from the recorded template. Thus, we need to provide a possibility to adjust size for any image pair given (either to a globally defined size or resizing to the smallest common dimension in a pair). If we require a fixed overall image size some images will be below this shape while other images will be above. The genuine notion dictates, that downscaling should be preferred to upscaling since the first method compresses originally available information into a smaller format, while the latter needs to create new artificial pixels which have not been present before. Therefore, if images are above our global size we will downscale them accordingly. However, if images are below this size, we want to experiment with two different resizing approaches. The first approach will utilize the same procedure which we want to use for downscaling. We choose to apply a nearest neighbor interpolation, implemented by the python library cv2 (for implementational details see [31]). The second approach will utilize a padding method, thus simply adding black pixels around the borders of an image until desired image size is reached. We want to investigate which procedure performs best, comparing both results when implementing our prediction algorithms.

5 Data Transformation

After finishing all data preprocessing steps, this chapter wants to examine feature engineering and data projection. The main goal of this section is devising a procedure which will be capable of extracting the underlying signal from an image. Thus, some form of meaningful image representation or embedding is required, which can be applied to an image pair such that the resulting representations can be used for calculating a correlation value. We want to start with the notion that each signal is a simple two dimensional plot, involving a x- and a y-axis. Therefore, each image can be viewed as a representation of this plot, with each pixel representing a coordinate in a two dimensional grid. The width of the image can be used as x-coordinate and the height as y-coordinate. To illustrate this concept let us consider a simple depiction of a signal:

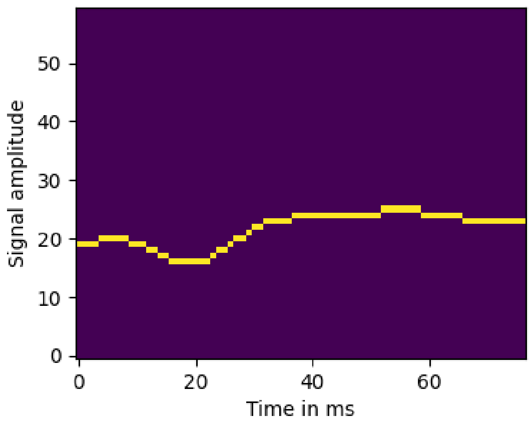


Figure 28 Image representation of a signal.

As shown in figure 28, the x-coordinate of the signal can be described by looking at each column individually. The first column corresponds to a measurement at time $t=0$ ms, the second column corresponds to $t=1$ ms etc. However, extracting the y-coordinate represents a more challenging task, because several y values for one x-value exist. Since all our images are greyscaled during preprocessing, we can safely assume to be left with pixel values ranging from 0 to 255, with 0 being the black pixel, which does not contain any valuable information. Nevertheless, it is not clear how to process different pixel intensities above 0. Therefore we require a procedure, attributing meaning to different pixel intensities in a single column of any image given.

We intend to utilize the data transformation step used in the paper "Visual Time Series Forecasting: An Image-driven Approach". The main idea is to determine the y-value of each column by perceiving all pixels as part of a discrete probability distribution. Thus the values of all pixels in a given column x will always sum up to 1. In order to achieve such a representation, all pixels in an image are normalized column wise by dividing every pixel in a given column by the sum of all pixel values from this column. Figure 29 illustrates the method used by the authors in the original paper:

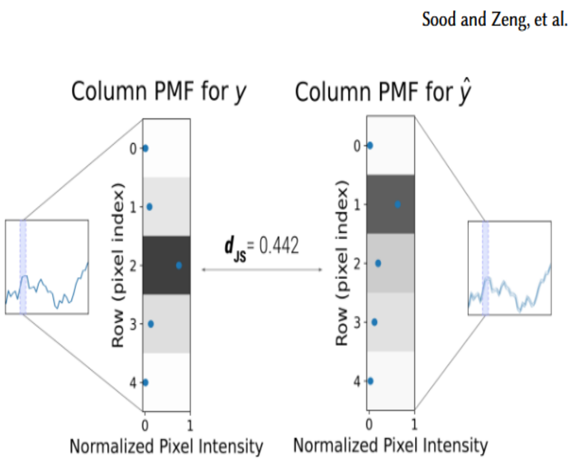


Figure 29 Normalized Pixel Intensity Distribution [15].

Considering a single column from the signal shown on the left, we can zoom in into the pixel distribution of this column. The brightest pixel can be observed at row 2, while slightly weaker pixel intensities can be observed for rows 1 and 3. The smallest value is recorded for rows 0 and 4. Thus, the pixel distribution resembles a gaussian bell curve. On the right hand side the prediction made by the autoencoder network used by the authors $\hat{y}$ is displayed. Here, the pixel distribution of the same column is slightly shifted, with the brightest pixel occurring at row 1, while the second brightest pixel is shown at row 2 and 3 consecutively. Rows 0 and 4 once again contain no valuable information. The difference between original and predicted pixel distribution is calculated using Jensen-Shannon Divergence, a loss function which calculates the distance between an approximate distribution Q to a true distribution P (for more information see [32]). Note, that figure 29 displays a white background with relevant pixel intensities being displayed as blue values. However, in our scenario black background is preferred since black pixels carry no information. Thus, the background needs to be inverted while the graph itself changes from blue to white. Figure 30 illustrates the adapted normalization technique for our images. For better readability the white signal line is depicted with yellow intensity points. In addition, no distance metric is applied, since our loss function will be calculated based on the embedding vectors. Nevertheless the general logic of this procedure remains the same. Depending on the distribution of pixel intensities, our normalization procedure yields a single dominant pixel per column or several equally bright pixels, which based on the prediction procedure used, needs to be handled accordingly.

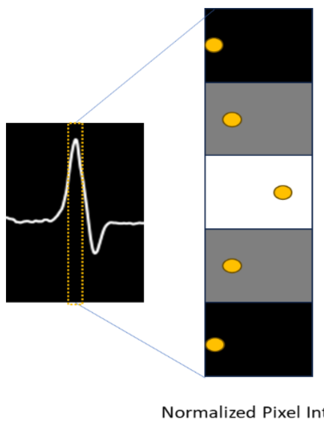


Figure 30 Normalized Pixel Intensity Adaptation.

6 Feature Extraction and Correlation Prediction

We can observe that after applying the transformation described above, we are still left with multiple values for one x-coordinate. Thus, our next step needs to devise a procedure which can use the transformed signal depiction and derive some sort of embedding, which can extract the signal from the image and use it for correlation computation. In this chapter we want to introduce two different approaches for image comparison. On the one hand we will rely on a deterministic algorithm, which will calculate one single y-value per column given. On the other hand we want to utilize a non-deterministic approach using a neural network, similar to the research described in HazelNet, which will consider several pixels for each y-value.

6.1 Match EGM Algorithm

While building a deterministic algorithm we choose to investigate two separate approaches, yielding a single row value per column. The first approach picks the row with the highest intensity value in each column. If more than one row should exist, the average is formed. The second approach evaluates the weighted average of all white pixels by multiplying their intensity value with the corresponding row number. Thus, if for a chosen column (say column 10) our transformed rows (say rows 160,161,162,163 have white pixels with the following intensities (0.15, 0.5, 0.15, 0.2) the function value at $x = 10$ for the first approach will be 161, while the second approach will yield $y = 161.4$ $(160*0.15+161*0.5+162*0.15+163*0.2)$. Note that both approaches can yield row values which are non-integers. Nevertheless, we can use these values without further processing, because correlation computation does not require integer values. Since images which belong to a single pair have the same size no resizing procedure needs to be applied for this technique[12]. Figure 31 illustrates the extracted signals of an image pair using our deterministic method, by plotting the time series values below the original image. We can observe that the extracted curve differs from the original picture, since subtle changes in the signal depicted cause a much stronger deviation in the plot constructed. Nevertheless, the general notion of the original image is captured. We call this method: Match EGM Algorithm

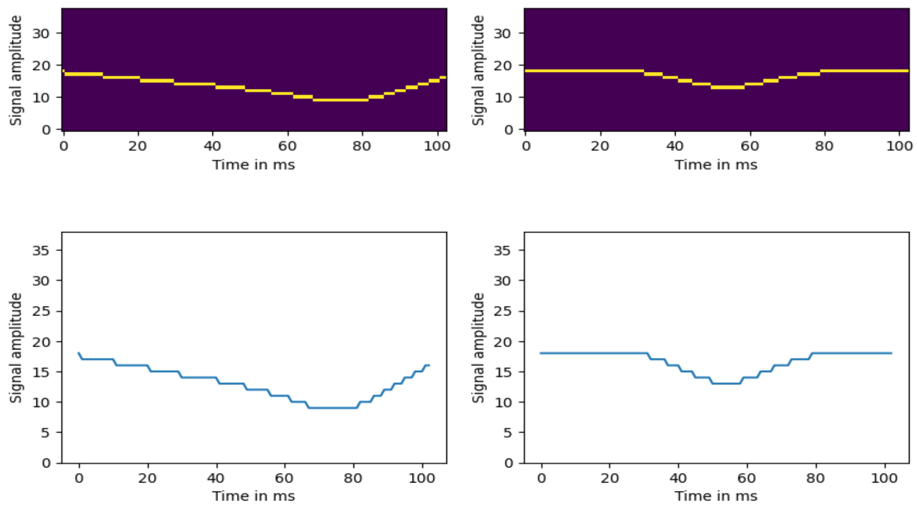


Figure 31 Algorithmic image to plot conversion.

[12] However, preliminary experiments showed, that small changes in size (only several pixels per dimension) can be mitigated using resizing procedures. This can be particularly useful, if template and match image are not exactly of the same shape.

6.2 Neural Network

Next, we want to take a closer look at a non-deterministic extraction approach, utilizing a neural network. Considering previous work, the setup used has been a Siamese network, related to the results achieved in HazelNet. In order to expand upon this setup, we want to provide a quick introduction on Siamese networks, the core of our non-deterministic setup. Afterwards we will introduce the improvements to the original architecture made by this work.

6.2.1 Siamese Networks

Siamese Networks have first been introduced in 1994 [33], with the novel idea of sharing weights between two identical networks for feature extraction. Since then, this architecture has been successfully used in many different use cases, the most prominent one being face recognition [34]. Main advantages of this setup are a robustness to transformations (e.g. change in illumination, size, rotation etc.), computational efficiency (since we only train / use one physical network) and a holistic training approach (feature extraction and distance calculation are combined in training). However, some drawbacks usually recognized are its complexity (finding the right hyperparameter setup) and its weakness for overfitting [35] . Figure 32 illustrates the setup of a typical Siamese network:

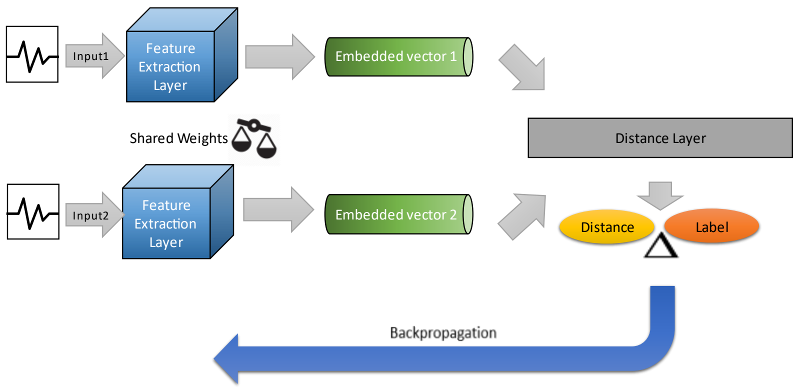


Figure 32 Typical Architecture of a Siamese network.

Every Siamese setup is made up off a feature extraction layer (depicted in blue) and a distance layer (depicted in grey) which are combined to calculate the similarity based on two inputs. This means that for each input pair (img1,img2) both elements are passed individually through the same feature extraction layer, resulting in two embedded vectors (emb1, emb2) shown as a green cylinder in the figure . Afterwards, a distance metric is applied to calculate the distance between these two vectors in their n-dimensional feature space. The result is than used for classification or regression, comparing it against a provided label, hence Siamese networks are usually used as part of supervised learning methodology.

6.2.2 Match EGM Network Architecture

In order to establish our non-deterministic prediction solution, we want to combine the Siamese network architecture introduced, with the solution devised by Sood et al. [15] called VisualAE. The authors utilize an undercomplete autoencoder architecture, depicted in figure 33, to extend the visual representation of a time series given, by continuing the time series in the decoded output image. The autoencoder receives a greyscaled image which is encoded by three consecutive convolutional layers, halving the initial image dimensions with each layer, resulting in an embedding vector of the dimensions 1x512. Afterwards, the embedded vector is decoded by the same number of convolutional layers with equal dimensions as the encoding section, yielding a new image of equal size, containing the prolonged time series. Since we do not want to predict time series development, the second part of the architecture can safely be discarded. Instead, we want to take the first half of the autoencoder as feature extraction layer, creating the corresponding embedding vector for the distance layer of the Siamese network. Therefore we want to reuse and adapt the encoding layers of the architecture shown in figure 33 (first four sections depicted), to build the one dimensional embedding vector (depicted in the middle of the figure).

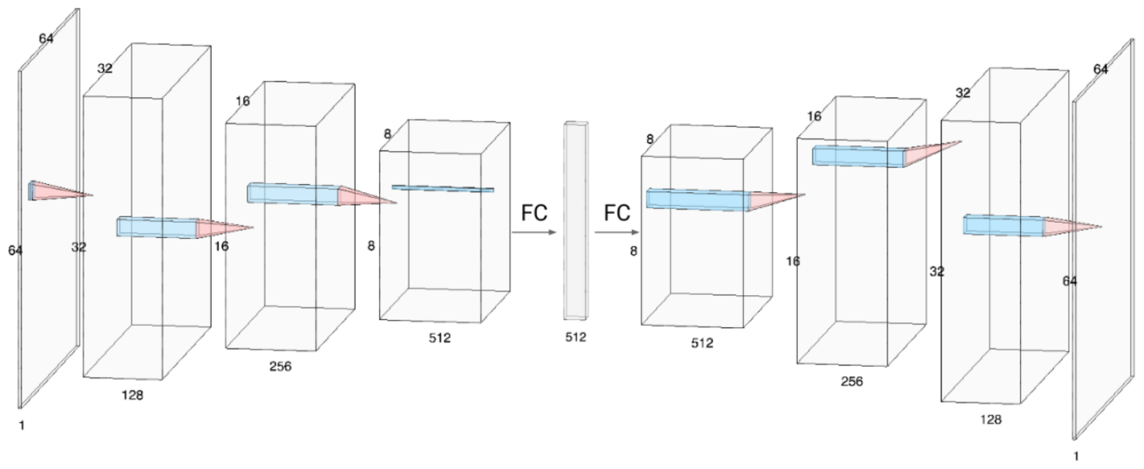


Figure 33 Encoder / Decoder architecture used by Sood et al [15].

After applying the feature extraction to both images, the resulting embedding vectors will be compared in the distance layer of our Siamese network, similar to the structure in HazelNet. However, implementing this adaptation, will yield a much smaller feature extraction layer, with significantly fewer parameters (weights). This corresponds to our assumption that a much smaller feature extraction layer will suffice for our task, since most pixels in our image do not contribute to the signal depicted. We choose to alter the number of neurons in the linear layer from 512 to 256, following the general notion that our images depict signals between 100-200 milliseconds, thus we perceive 512 embedding neurons too large for this signal length. In addition, the input layer of our feature extraction shall have the same dimensions as the embedding layer. Therefore, all image pairs will be resized to fit the determined embedding layer size of 256, having the dimensions 256x256. The number of convolutions used, kernel size, stride and padding will be subject to hyperparameter optimization. However, the setup used by the authors will be included (CNN blocks = 3, kernel size = 5x5, stride = 2, padding = 2). Figure 34 illustrates the new general architecture of our neural network which we call Match EGM Network:

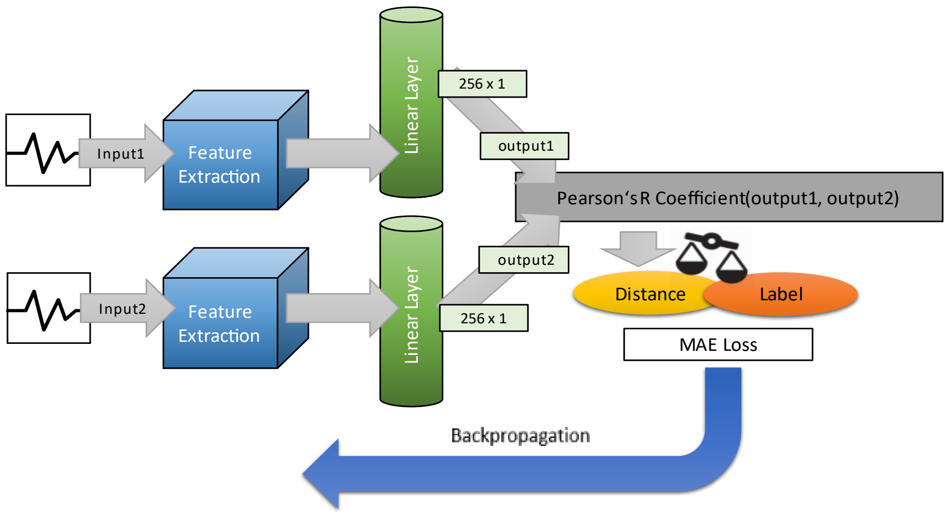


Figure 34 Architecture of Match EGM Network.

Similar to the autoencoder architecture, each CNN block of our feature extraction layer will be superseded by a batch norm layer and completed by a ReLU activation function. In addition, a dropout layer will be inserted before the batch norm layer, dropping out 50% of neurons during training. We believe this alteration to be useful, helping in generalizing on the different types of images (based on different data sources) and mitigating the general drawback of Siamese networks mentioned earlier: overfitting. For the distance layer, Pearson's R coefficient will be used instead of the cosine similarity score utilized by HazelNet. Since we know that our target labels are computed based on Pearson's correlation coefficient, we believe to achieve the best results by using the same function in our distance layer. The loss function will be changed to mean average error (MAE) instead of mean squared error (MSE), to benefit from better interpretability of the loss metric[13]. Thus, an MAE of 0.10 can be interpreted as: On average our prediction is 0.1 lower or 0.1 higher than the actual target label, e.g. prediction: 0.8, label: 0.7.

[13] Note, the loss function used by Sood et al. cannot be adapted for our work since the loss metric used for time series forecasting is entirely different focusing on the likelihood of pixel intensity in a particular location of an image. For more information see chapter 4.2 of the corresponding paper [15].

7 Experiments Setup

Rounding off the feature extraction and correlation prediction section of this work, we want to introduce the experiments setup used for determining the best algorithm. As explained in the beginning of this thesis, we want to investigate three different datasets: Artificial-Uniform (AI Uniform), Artificial-Random (AI Random) and the ECG-pictures from the laboratory (ECG/Real). The main goal is to analyze and find the best setup for predicting the correlation of two images provided by the ECG-dataset, since we consider it to be closest to our real case scenario. Therefore, all experiments will always use ECG data (not an artificial validation / test hold out) for calculating validation and test loss. Artificial validation and test hold outs are used solemnly for review purposes, not for evaluation. Since setting up the non-deterministic approach poses a much more comprehensive task, we want to focus in this chapter on the setup used for investigating the neural network. Nevertheless, we will introduce a preliminary experiment based on ECG data, evaluating the two different pixel evaluation approaches for our algorithm as described earlier (averaging pixel intensities vs brightest pixel). Figure 35 depicts the general setup of our experiments:

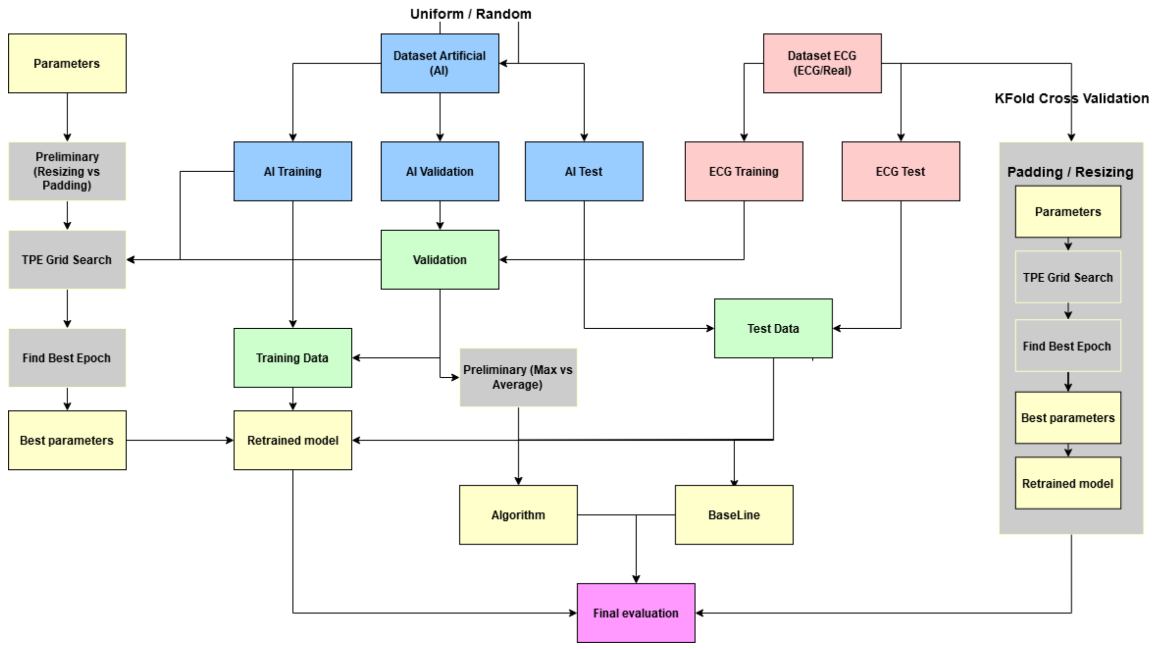


Figure 35 Experiments Setup.

On the left hand side of figure 35, experiments on artificial data are illustrated. First a preliminary experiment is conducted, researching which data preprocessing method yields potentially better results (resizing vs padding). Next, we want to analyze how stronger randomization in data generation affects evaluation results, thus comparing AI Uniform and AI Random datasets. After determining which dataset is best for training, we will investigate how retraining our best possible setup with ECG data affects final performance. Therefore, we want to compare a network trained purely on artificial data to one subsequently retrained with ECG data. On the right hand side of the figure experiments exclusively on ECG data are depicted. This part of our setup wants to yield the best possible results achievable if no artificial data is involved. Hence, we want to observe if a network trained solemnly on ECG data is better or comparable to the other two approaches. Finally, considering our best setups (artificial, retrained and purely ECG) we want to compare these results to our deterministic extraction algorithm devised above, as well as a simple baseline during final evaluation. For the baseline a simple randomized prediction will be made for each image pair, picking a value between -100 and 100 and dividing it by 100 to get a scaled label value accordingly.

In the final assessment, all setups except for the one trained purely on ECG data, will be evaluated on a test hold out of the ECG data. We choose to use Cross Validation with K Folding for the latter, because the ECG dataset is much smaller than the AI datasets (558 vs 10.000 image pairs). We apply inner and outer cross validation with k being equal to 5 for both cross validation loops. In contrast, experiments on artificial data will utilize a traditional train-val-test split with 70% train, 15% validation and 15% test data. The ECG data used for validation, retraining and evaluation in these experiments will be split into 60% train and 40% test data. All splitting will utilize stratification, ensuring equally distributed labels, based on the binning algorithm described earlier. To ensure a globally uniform, independent test data set, needed for meaningful method comparison, we control which samples are split into which fold by setting a global seed parameter. This makes sure that we always compare all methods on the same test dataset for final prediction quality evaluation. For data augmentation a vertical line will be added at random to an image. We choose 10% as our chance of augmentation based on roughly 6% of observed samples with vertical lines in our ECG data. In addition, the thickness (1-2 pixels), intensity (1-255), horizontal position (0-image width) and length of each dash (1-3 pixels) will be altered for every augmentation applied.

All experiments regarding neural network training require some sort of hyperparameter optimization, which we want to explain in the following. For hyperparameter optimization we choose to use Optuna, a state of the art optimization framework [36]. Each optimization iteration is denoted as an Optuna trial, while all trials combined form an Optuna study object. Every trial carried out will be logged in the study object and can be accessed via the appropriate function. Overall, we choose to conduct three separate studies for each type of dataset, each containing 42 trials. Since hyperparameter optimization is time consuming, we want to undertake several measures to reduce the number of possible hyperparameters involved. First, we want to conduct a preliminary experiment, investigating performance of standard resizing vs the padding procedure as mentioned above. Second, we fix the number of epochs used during optimization to reduce calculation time to 30 epochs per trial. Once we have established the best hyperparameter setup the optimal number of epochs will be determined, by using a patience setup with the number of patience epochs set to 5 and a threshold of 0.005. This means that training will be stopped if for 5 consecutive epochs difference in validation loss to prior epoch is below 0.005. Third, we want to utilize an early stopping procedure, which after waiting 10 startup trials will prune a trial after the 10th epoch if median real loss is larger than the median of all previous trials at this epoch. This check will be performed every 5 epochs.

7.1 Grid Search

However, the most time consuming operation during hyperparameter optimization is the grid search. Therefore we want to take a closer look, at different grid search algorithms.

7.1.1 Grid Search Algorithms

The first possibility is manual hyperparameter tuning. This method relies on domain expertise and experience, whereby we manually adjust the hyperparameters after observing a result with the hyperparameters chosen before. However, this process is very time consuming since it requires a constant monitoring and adjusting of the setup. The second possibility considers an automated approach, by executing an exhaustive grid search, trying out all possible combinations of hyperparameters involved. Although this approach does not require much human interaction, it is very costly in terms of computational power, since the number of possible combinations increases exponentially with the number of hyperparameters used. A different approach uses randomized grid search, limiting the number of trials to a certain threshold, randomly selecting hyperparameter values for each run and returning the best possible combination after the number of trials has been exceeded. Although better in terms of computational power, this process might not lead to good results, since favorable combinations might be missed. In order to mitigate this behavior more trials can be used, which once again comes at the cost of computational power. In order to improve this search strategy the successive halving algorithm can be utilized, whereby n random setups are chosen at the same time and executed for a threshold number of epochs k. After k epochs only the best 50 % setups are kept, while the other half is discarded. This process repeats itself every k epochs until only one model is left [37]. Using successive halving speeds up computation significantly, but comes at the cost of parallelization, which requires strong hardware resources, especially GPU RAM, since images are involved.

However, one main disadvantage of the approaches described above is not reusing or considering previous results from already executed runs when starting the next one. Thus, prior knowledge is lost and each trial starts out from zero. Bayesian optimization mitigates this problem, by reusing the results from previous experiments in order to improve on the current trial. The main idea of these types of algorithms is to use a surrogate function trying to mimic the real objective function f, which given a set of hyperparameters x yields a certain loss y, where y is the value of our chosen loss function (e.g. MAE). Since we do not know the distribution of this function, several different surrogate approaches exist. Common choices are Gaussian Processes [38], Random Forest Regressions [39] and Tree Parzen Estimators (TPE) [40]. All these approaches belong to the group of Sequential Model-Based Optimizations [41], since they use a surrogate model optimizing it in a sequential manner, thus executing one trial after the other, updating the model after each iteration. This methodology works well with RAM intensive data, since parallelization is not required. In this work we want to focus on the last of these approaches, namely TPE.

7.1.2 Tree Parzen Estimators

TPE has first been proposed by Bergstra et al in 2013 in the corresponding paper "Algorithms for Hyper-Parameter Optimization" [42]. Given a real objective function f, the set of hyperparameters x and loss y, a selection function EI(x) is constructed, in order to decide how the next set of hyperparameters x shall be retrieved. EI(x) represents the expected improvement of our loss y with respect to a threshold loss y*, given the set of hyperparameters x. It is defined as:

$$EI(x) = \int_{-\infty}^{y*} (y^* - y) * P(y|x)$$

$P(y|x)$ represents the surrogate function, the probability of observing a loss value y given a hyperparameter x. The key idea of TPE is using Bayes rule to express $P(y|x)$ as:

$$P(y|x) = \frac{P(x|y) * P(y)}{P(x)}$$

$P(x|y)$ means that we want to model the probability of observing a certain hyperparameter x given a loss value y. The next step is dividing this function into two sub models. The first model represents the probability distribution of observing x given loss values y which are larger than some threshold y*, while the second model represents the probability distribution of observing x given loss values y which are smaller than the given threshold y*. In other words, we keep track of two separate surrogates, one which models undesirable loss values $P(x|y > y^*)$ and one which models favorable loss values $P(x|y < y^*)$. The threshold y* is itself another hyperparameter which needs to be determined. The work of Bergstra et al shows, that the ratio of $\frac{P(x|y<y^*)}{P(x|y>y^*)}$ is proportional to the expected improvement EI(x). This means, that we want to choose a new hyperparameter x such that the probability of observing x given $y < y*$ is as large as possible while the probability of observing x given $y > y*$ is as small as possible. We can see how this procedure plays out in practice by looking at the example below (figure 36):

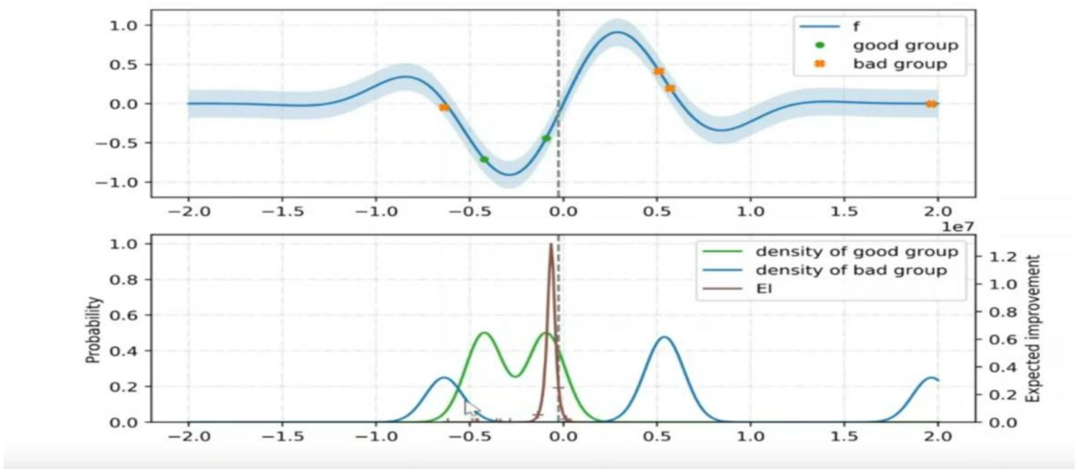


Figure 36 Objective Function with surrogates (TPE) [43].

The first plot shows our original objective function f. The graph beneath depicts the two surrogate functions, the green curve showing the probability distribution of $\mathrm{y} < \mathrm{y}*$ while the blue curve shows the distribution of $\mathrm{y} > \mathrm{y}*$. Given these two surrogate functions, the final probability distribution $\mathrm{EI}(\mathrm{x})$ (shown in red) can be determined by combining them. The point where $\mathrm{EI}(\mathrm{x})$ is largest can be used as the new hyperparameter setup x'. We determine the actual loss value for x' and repeat the process again, until we exceed the amount of trials or our function converges to a global or local minimum (shown by the hills / valleys in the objective function above). For the exact implementation used in this thesis and the available hyperparameters of TPE, see the following documentation by Optuna [44]. One of the major drawbacks of this method is, that it does not consider the interactions between different hyperparameters but rather treats each hyperparameter individually. It is therefore part of an independent sampling strategy. This means that the processed described above is executed for each hyperparameter introduced to the grid. Thus, with growing amount of hyperparameters this method also weakens, since determining the maximum of $\mathrm{EI}(\mathrm{x})$ for each x adds time to each trial. An alternative to include dependencies between hyperparameters is shown by Gaussian processes [38]. However, we will not pursue this altered approach further since for the number of hyperparameters present TPE can safely be applied.

7.2 Defining the Search Space

Next we want to define the search space, hence the set of hyperparameters we want to investigate. Since TPE is not able to evaluate combinations of different parameters, but rather considers each parameter individually, we will conduct three separate optimization experiments, varying the dataset used (analogue to an exhaustive grid search scenario). This way we can ensure that we analyze all combinations individually and do not discard one in favor of the other too early. In addition, the preliminary experiment will determine which data preprocessing method shall be used. Thus, our final grid will already build on top of the results from this preliminary experiment. Our search grid looks as follows:

Variable Parameters:

- Optimizers: Adam, RMSprop, SGD
- Learning Rate: 1 * 10^{-5} - 1* 10^{-1}
- Random weight initialization seed: 1-100
- Network Architecture:
  - Number of CNN blocks: 3,4,5 (since we always want to end up with 256 feature channels we build the layer definitions top to bottom, halving the feature channels accordingly, resulting in 32,16,8 feature-channels at the first layer)
  - Stride and Padding of each layer: 2,4,6 (we always use an equal value for each dimension)
  - Kernel Size: 3,5,7 (we always use an equal value for each dimension)

Note that for our dynamic architecture search some combinations of hyperparameters are invalid. This behavior can occur if stride, padding, kernel size and number of blocks are chosen such, that resulting image dimensions are below 1 pixel per feature channel . To avoid such a scenario, we determine the final image width beforehand and skip a hyperparameter setup if it is invalid. Out of 81 possible combinations for the hyperparameters named above (stride, padding, kernel size and number of convolutional blocks), 9 are invalid, which yields a skip rate of 11.11 %.

Fixed Parameters:

- Number of neurons in the linear Layer: 256
- Number of in-channels in the first CNN layer: 256
- Activation function: ReLU
- Each CNN block will be finished by a BatchNorm and a Dropout Layer (dropping 50%)
- Color of the images: All images will be greyscaled
- Distance function: Pearson's R coefficient
- Loss Function: MAE
- Number of epochs for each trial: 30
- Number of trials per study: 42
- Number of maximum epochs for epoch optimization: 2000
- Resizing Method: Nearest Neighbor Interpolation / Padding (Determined by preliminary experiment)
- Evaluation Metrics: MAE, MSE, R2

Exhaustive Grid Search Parameters (represented by 3 separate studies):

- Artificial Data:
  - Uniform: Images generated with uniform amplitude, time series length, image size
  - Random: Images generated with varying amplitude, time series length and image size
- ECG Data

8 Interpretation & Evaluation[14]

After introducing the general setup of our experiments, we want to present the results of our research. This chapter will start with the results of our preliminary experiments, which we will utilize for the main analysis afterwards. Next, all research based on artificial data will be presented, starting out with the grid search results, moving over to epoch optimization and finally complete retraining and evaluation. Afterwards, experiments solemnly on ECG data will be presented. In the end, the last sub chapter will compare all different setups, determining which one should be used for the app utilized live during operations.

8.1 Preliminary Experiments

We start our evaluation with the two preliminary experiments mentioned earlier. The first one analyzes the resizing methods required for image transformation, while the second one will focus on the deterministic solution.

8.1.1 Network (Resizing vs Padding)

In order to determine which resizing method should be preferred, we will execute 30 trials per resizing method using TPE grid search on the uniform AI dataset with 20 epochs per trial. The hyperparameters used during grid search will be identical to the ones describe above. We want to compare the best result evaluated on ECG data.

Trans method	MAE	Model Seed	Learnig rate	Optimizer	Blocks	Padding	Stride	Kernel size
padding	0.2660	73	0.0339	RMSprop	3	2	4	5
resize	0.3107	35	0.0009	Adam	3	6	6	7

Table 2 Preliminary Results Network.

Looking at the results of our preliminary experiment depicted in table 2, we can clearly see, that using padding as part of data preprocessing yields significantly better results when compared to regular resizing. Prediction performance improves by 14.39% from 0.3107 MAE standard resizing to 0.2660 MAE with padding. Thus all further experiments will use padding as part of data preprocessing.

8.1.2 Algorithm (Maxsize vs Average)

As mentioned in the previous chapter, we want to conduct a small analysis regarding the best data transformation for our deterministic solution, comparing the two transformation methods presented. Thus, we executed the algorithm on validation hold outs from uniform and random artificial datasets, comparing the evaluation metrics MSE, MAE and R2 to the result achieved on the ECG training hold out. Table 3 below depicts the experiment results:

Data	Transform Method	MAE	MSE	R2
ai_uniform	maxintensity	0.0037	2.7E-05	0.9999
	average	0.0049	6.3E-05	0.9998
ai_random	maxintensity	0.0077	3.5E-04	0.9990
	average	0.0146	0.0008	0.9975
real	average	0.1410	0.0700	0.7893
	maxintensity	0.1411	0.0702	0.7898

Table 3 Preliminary Results Algorithm.

[14] All values presented will be rounded to maximum 4 decimal places for a better display

Looking at the MAE values, we can observe, that the maxintensity approach fairs best on both artificial datasets with scores of 0.0037 (uniform) and 0.0077 (random) MAE respectively. However, on ECG data maxintensity scores slightly worse than the averaging method with 0.1410 vs 0.1411. We conclude, that it is not quite clear which method is more useful for our real world application, since maxintensity seems to perform better on artificial data while averaging is slightly better on ECG data. However, since we consider ECG data to be more realistic, we decide to use averaging as the method for our algorithm.

8.2 Experiments on Artificial Data

After finishing the two preliminary investigations, we want to turn to the main analysis of this chapter: Developing a neural network which is trained on artificial images and evaluated on ECG ones.

8.2.1 Grid Search Results:

We start with the grid search mentioned in the previous chapter. Table 4 below shows the results of our grid search executed on artificial target data. We can observe, that all hyperparameters regarding the architecture of the transformation layer are identical for both datasets. Both setups yield a large kernel size of 7x7 pixels, a small stride with 2 pixels and moderate padding of 4 pixels. This means, that best results are achieved if we consider large portions of the image for each convolutional operation, but move only slowly from one area to the next. Moderate padding sizes can be explained by the fact that all images start immediately with a white pixel depicting the signal, thus padding is needed to not lose information when performing convolutional operations[15]. The main differences between our datasets arise with the choice of optimizer, learning rate and weight initialization. Here, SGD with a learning rate of 0.0573 is chosen for uniform data, while Adam with a learning rate roughly three times smaller (0.0193) for random data is preferred. Weight initialization for the first dataset is performed with a seed of 78, while for the second 84 is picked. We can also observe, that overall learning rate is quite high, especially when comparing these settings to common values used in established networks like resnet18, which usually use rates of about 0.0001 with Adam. Overall, we can see that introducing more randomness to our artificial data, approximates ECG data significantly better, yielding a mean average error of 0.1440 for randomized data vs 0.2819 for uniform, which yields a 48.91% performance improvement. Considering the uniform surrogate data used in previous experiments conducted in HazelNet, we believe, that our new randomized approach, should surpass the old model prediction quality. Therefore, we will focus on randomized data for training our neural network.

Datatype	**MAE**	**Model Seed**	**Learning rate**	**Optimizer**	**Blocks**	**Padding**	**Stride**	**Kernel Size**
uniform	0.2819	78	0.0573	SGD	4	4	2	7
random	0.1440	84	0.0193	Adam	4	4	2	7

Table 4 Grid Search Results.

8.2.2 Choosing Number of Epochs

Next we want to determine the maximum number of epochs required for training our setup using the patience method described above. Table 5 shows the original MAE before epoch optimization and the best metrics (MAE,MSE,R2) found during longer execution for ECG data, as well as for the artificial validation holdout.

[15] For a summary on CNN operations and padding see: [45]

Grid Search MAE	MAE Best Epoch	R2 Best Epoch	MSE Best Epoch	MAE VAL AI	R2 VAL AI	MSE VAL AI	Best Epoch
0.1440	0.1361	0.8374	0.0563	0.0306	0.9921	0.0026	74

Table 5 Best Epoch AI Data.

We can see, that optimizing the number of epochs yields a performance increase of about 5.5 %, from 0.1440 to 0.1361 MAE. In addition, we calculate the R2 coefficient with 0.8374 and MSE with 0.0563. Both metrics are interesting, since utilizing the MSE we can gain knowledge about outlier distribution, a point we will analyze further down in this chapter, while the R2 coefficient shows, that about 83.74% of overall variance can be explained by our model. Looking at the artificial hold out, we can observe much lower MAE / higher R2 respectively, indicating overfitting on AI data.

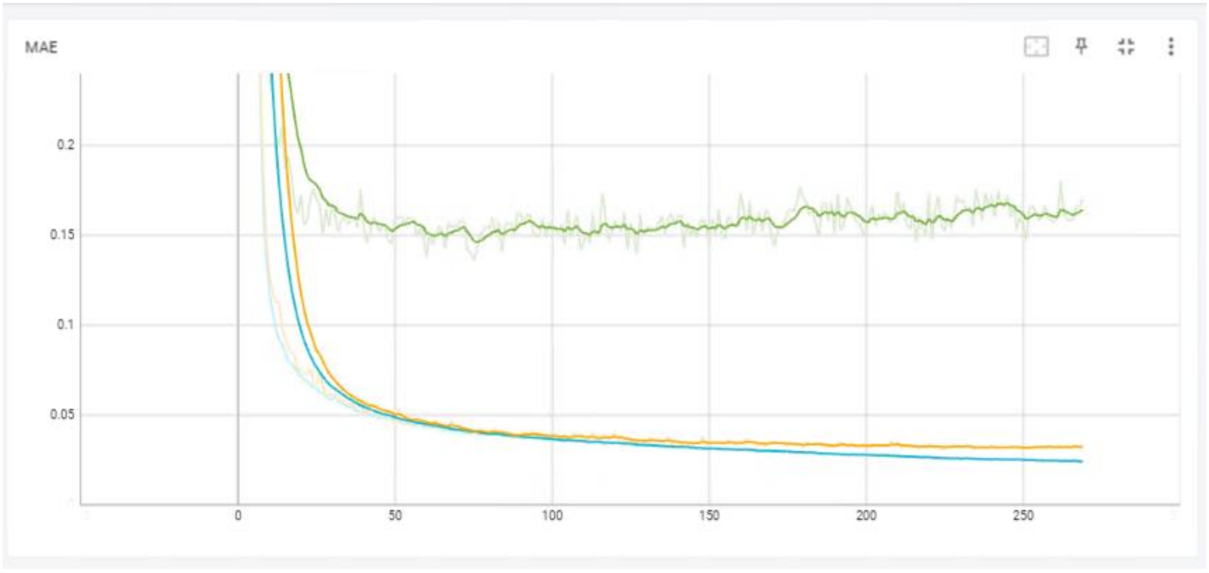


Figure 37 Loss plot finding best epoch.

Figure 37 above shows the loss calculated per epoch, all curves are smoothed with a factor of 0.8. The green curve represents MAE calculated on ECG data, while the orange curve shows the same metric evaluated on AI validation data and the blue curve on AI training data. We can observe, that training was cancelled after epoch 269. However, as shown by table 5 above, best results on ECG data were achieved in epoch 74 indicated by the green downward spike. Overall train and validation (AI) loss show a very similar development, both curves being almost identical up until epoch 150. Afterwards, overfitting can be observed with validation loss deviating from training loss, which continues to decrease. The validation curve flattens out around epoch 200 with MAE 0.0479, while training loss flattens our around epoch 255 with MAE 0.0337. Nevertheless, the largest decrease of loss occurs for all curves during the first 50 epochs. Here the elbow of the graphs can be observed around epoch 35. The green curve fluctuates much more than the other two with several up and downward spikes. Moreover it shows clear signs of overfitting roughly after epoch 130, increasing again in value after from the minimum reached in epoch 74 with 0.1361 MAE up to 0.1644 at the last epoch. Because training cancelation with the patience method is based on ECG data, fluctuations in loss value (shown as green up and downward spikes in the figure) lead to a much prolonged training period until stopping criterion is reached.

8.2.3 Retraining on all Data

After finding the best possible model, we want to investigate if by retraining on ECG data our model will perform better on the test holdout when compared to the same model solemnly trained on artificial data. However, retraining on ECG data exclusively is not recommended, since this might lead to an effect called catastrophic forgetting [46]. This term describes the loss of generalization ability of a model due to overfitting on the retraining set. Thus, a model which performed well on the first set of data might completely fail on the same type of data once retrained with the new dataset. Although we believe that ECG data is more similar to our final target data, we do not want to lose possible generalization ability acquired by training on artificial data, because we have much more artificial data samples, covering a much wider range of possible signal pairs. In literature, several different approaches exist, mitigating catastrophic forgetting. Possible solutions include freezing partial layers during retraining, adding new layers dedicated to new type of data, changing hyperparameters during retraining (like learning rate, epoch, batch size) etc. [47]. Since mitigating catastrophic forgetting and retraining neural networks on new data or for new tasks represents a separata line of research, we do not want to expand on it further, because it would leave the scope of this thesis. Instead, we chose to utilize a simple, yet effective method for mitigating this effect, by pairing up data used in original training with new data suited for retraining. In literature this process is called "Rehearsal" [48]. In our particular case we adapt this general notion, by combining artificial data used for validation with ECG data used for validation. Thus we retrain our network on a mix of data it has not yet been trained on before, thereby adding new information which can improve predictions. Table 6 illustrates the amount of samples available for retraining:

AI Samples	Real Samples	Real Train Split	AI Train Split	Real Val Split	AI Val Split
1600	352	1880	1280	425	320

Table 6 Sample Distribution AI/Real + Folding.

We can observe, that the artificial part of the combined dataset, outweighs the ECG part by a factor of roughly 4.5, with 1600 AI samples vs 352 ECG samples. This disproportion would lead to a significantly larger influence of AI data, which we want to mitigate by oversampling the ECG dataset. Moreover, using the combined dataset, the optimal number of epochs established in the previous experiment needs to be reevaluated, since this value might no longer be valid for the new training data. Therefore, a separate analysis for finding the best retraining epoch needs to be conducted. Since the ECG dataset is not very large, we choose to apply K-Fold-Cross-Validation, with k equal to 5, thereby ensuring that all labels present in the data are considered. Because the optimal number of retraining epochs shall be determined based off combined data, we prefer for each fold combination (train and validation) to have a similar proportion of ECG and AI data. In order to avoid data leakage, oversampling will take place after splitting data into corresponding folds, but will be executed on training and validation sets alike. This means, that in each iteration first ECG folds will be selected for training and validation, next the selected folds will be oversampled and finally a combined training and validation set out of oversampled ECG data und not oversampled AI data will be created. After applying K-Folding our training data will consist of 3160 samples in total. 1880 real samples (4 Folds of ECG data yielding 282 samples in total, which are oversampled) and 1280 artificial samples (80% of artificial validation data). The validation fold on the other hand will be made up of 745 samples, with 425 ECG samples (1 fold of ECG data yielding 70 samples, which are oversampled) and 320 artificial ones (20% of artificial validation data). Hence, in each iteration ECG data will be slightly overrepresented in both data sets. However, this setup enables us to have a roughly equal sized relation of artificial data to real data.

SPLIT (FOLD)	MAE Best Epoch	R2 Best Epoch	MSE Best Epoch	Best Epoch
1	0.1159	0.8622	0.0457	14
2	0.1073	0.8896	0.0367	4
3	0.0761	0.9582	0.0121	48
4	0.0826	0.9488	0.0163	8
5	0.0784	0.9522	0.0149	71
Average	0.0921	0.9222	0.02514	29

Table 7 K-Folding Best Epoch Eval.

Table 7 above, depicts the resulting scores from all 5 folds as well as the averaged metrics. First of all we can observe that our averaged best epoch (rounded to 29) is much lower than the value previously determined based on purely artificial data. This corresponds with the convention in literature, that retraining should usually take up less epochs than initial learning. Note that overall much better loss and evaluation metric results can be seen when comparing with results before retraining. However, these values are more of an informative character, since we have observed, that our network performs significantly better on artificial data than on real one. Thus, combining these two datatypes for performance evaluation is unfavorable, pushing overall results on ECG data. Using the determined epoch we now proceed retraining our model with the entire combined data. Therefore, we use 3905 samples in total, 1600 AI samples and 325 ECG samples oversampled to 2305 for retraining. This yields a real data / artificial data ratio of 1.44, hence we have roughly 44% more ECG samples than artificial ones. The final result of retraining will be presented in the chapter Final Evaluation, comparing it with our other approaches.

8.3 Experiments on ECG Data

After finishing research regarding artificial data we want to analyze results obtained from experiments purely on ECG data. Because cross validation is combined with TPE Grid Search, simple result averaging cannot be applied, since we cannot ensure, that each hyperparameter setup tested by the first TPE Search (e.g. for inner fold 1) is also tested for every subsequent fold. Thus, we choose to utilize a thresholding method, discarding setups that have been tested in less than 80% of cases (4/5 folds). In addition, the parameters learning rate and model seed are averaged as well, because they are chosen from a much larger target space (real numbers for learning rate and model seeds between 1-100). Thus, repetition is very unlikely. This means, that for each outer fold ($i=1$), 5 separate TPE grid searches are conducted, each containing 42 trials with 30 epochs per trial. Next, all results that contain invalid combinations are discarded. Afterwards, hyperparameters are grouped over each inner fold, averaging the resulting MAE, learning rate and model seed[16]. If a setup occurs at least 80% of the time it is considered for final evaluation. From this remaining number of setups the one with lowest MAE value is chosen and used for the outer fold i. In order to determine the optimal number of epochs, the determined best setup is evaluated once again in the inner cross validation, utilizing a maximum epoch threshold of 2000 and early stopping. For all 5 folds the best epoch found is recorded and again averaged. Finally, the optimal setup with the averaged epoch number is executed in the outer fold loop and evaluated on the test fold. All 5 outer test fold results are averaged and used as the final evaluation metric. The corresponding table in appendix II, illustrates which results from the inner cross validation grid search were chosen for each fold. For better readability we will not present the table in this chapter. However, all setups chosen were present in all 5 inner folds. The parameters picked occur around the 5th place when ranking results according to MAE values. Thus, some potentially better

[16] Note, since model seed needs to be an integer, the mean value is rounded down

setups are discarded, however all discarded setups fair only best in 1 or 2 out of 5 folds. Therefore, we choose to utilize slightly worse scoring results, which seem to generalize well on all folds. Next, we want to take a look at results of the inner cross validation grid search for the 5 outer folds:

Fold	MAE (AVG)	Model Seed	Learning Rate	Optimizer	Blocks	Padding	Stride	Kernel Size
1	0.1657	81	0.0369	SGD	4	4	2	7
2	0.1943	73	0.0392	SGD	4	4	2	7
3	0.1843	74	0.0347	SGD	4	4	2	7
4	0.1771	69	0.0327	SGD	4	4	2	7
5	0.1929	73	0.0320	SGD	4	4	2	7

Table 8 ECG K-Folding Grid Search Results.

Table 8 shows, that similar to our results based on artificial data, all hyperparameters regarding transfer layer architecture are identical (in between the folds and when compared to the best setup found on AI data). Thus, this setup seems to yield the best results, independent of training data used. However, results differ when turning towards optimizer and learning rate. In this case, SGD seems to be the best choice across all folds, moreover learning rate varies slightly from 0.0320 up to 0.0392, yielding an average of 0.0351 and a standard deviation of 0.0027. Weight initialization differs as well, with suitable model seeds being between 69 to 81. We can observe that both values show little standard deviation with values being close together, which implies stable results. Table 9 illustrates results after epoch optimization, the number of epochs chosen and final evaluation metrics MAE, MSE and R2 for the outer fold.

Fold	MAE Outer Fold	MSE Outer Fold	R2 Outer Fold	MAE (AVG) Best Epoch	MAE (AVG) Grid Search	Number Epochs
0	0.1880	0.0982	0.6981	0.1504	0.1657	72
1	0.1359	0.0524	0.7608	0.1760	0.1943	74
2	0.1723	0.0745	0.7623	0.1611	0.1843	68
3	0.1298	0.0681	0.8071	0.1683	0.1771	65
4	0.1333	0.0470	0.8642	0.1713	0.1929	55
Average	**0.1519**	**0.0680**	**0.7785**	**0.1654**	**0.1829**	**66.8**

Table 9 ECG Data Best Epoch Results.

The number of epochs for each fold varies between 55 to 74, thus having a standard deviation of 6.68, which considering the overall scale of the parameter is wider than the deviations from the inner folds for learning rate and model seed. In general, the number of epochs for training is slightly larger than the one determined for artificial data (67 vs 74). Comparing the average MAE values before and after epoch optimization we can observe, that epoch optimization gives a performance improvement of 9.57 % from 0.1829 to 0.1654. Turning to the evaluation results on the test fold, further improvement of 8.16% from 0.1654 to 0.1519 can be observed. However, looking at the separate fold results, we can see, that only three out of 5 folds show better results, while the other two increase in MAE value. This implies that training the model on a larger training data set (since outer validation uses more training data) improves performance. Nevertheless, some splits seem to fail at capturing the entirety of the dataset. This might be explained by heavy imbalance in the data, with some correlation bins containing only few samples. Although stratification is used, some models might fail at predicting these labels correctly, leading to a decrease in MAE, after these images are present. Thus, more ECG data could improve this approach further. Overall, MSE is 0.0680 while R2 is

77.85%. These scores suggest, that larger mispredictions are still occurring, and a quarter of all variance is not explained by this approach.

8.4 Final Evaluation

Now we want to compare all methods described in our initial experiments setup, using the test hold out of ECG data (except for the K-Folding setup, here the average will be displayed) as well as the test hold out based on randomized artificial data.

8.4.1 Artificial Holdout

We start with evaluating the artificial data hold out, as seen in table 10 (the table is sorted by MAE accordingly):

Model	**MAE**	**MSE**	**R2**
Algorithm	0.0160	0.0010	0.9971
Network Pretrained	0.0531	0.0060	0.9817
BaseLine	0.6542	0.6547	-0.9806

Table 10 Final Results AI Holdout.

We can observe that our deterministic approach fairs best, with an MAE value of 0.0160. This means that on average a correlation value predicted deviates only by 0.0160 from the label. The pretrained network comes in second with an MAE roughly three times as large of 0.0531. The baseline fairs worst, with a value of 0.6542. Thus, we can safely say that both approaches devised, outperform a random baseline, yielding meaningful results. Looking at the MSE values, we can observe that the proportions shift further in favor of the algorithm, with an MSE of 0.001, hence six times smaller when compared to the pretrained network with MSE 0.006. This indicates, that while our algorithm performs best, the pretrained network shows larger errors, with overall larger prediction deviations. For the baseline the MSE stays roughly the same. Turning to R2 values, we can observe, that deterministic and non-deterministic solutions are even more alike, with an R2 value of 0.9971 for the algorithm and 0.9817 for the network. This shows, that both methods explain almost all variance present in our data. For our baseline the high negative R2 coefficient indicates, that no meaningful explanation takes place. The negative value show, that the null hypothesis used during coefficient calculation (mean of label values) fairs better than our baseline[17].

Ignoring our baseline, figure 38 depicts the prediction errors based on our artificial test hold out. Predictions made by the neural network are colored in blue, while predictions made by the algorithm are orange. The bisector colored in black indicates a perfect fit, with target values on the x-axis and predictions on the y-axis.

[17] For more explanation on negative R2 see [49]

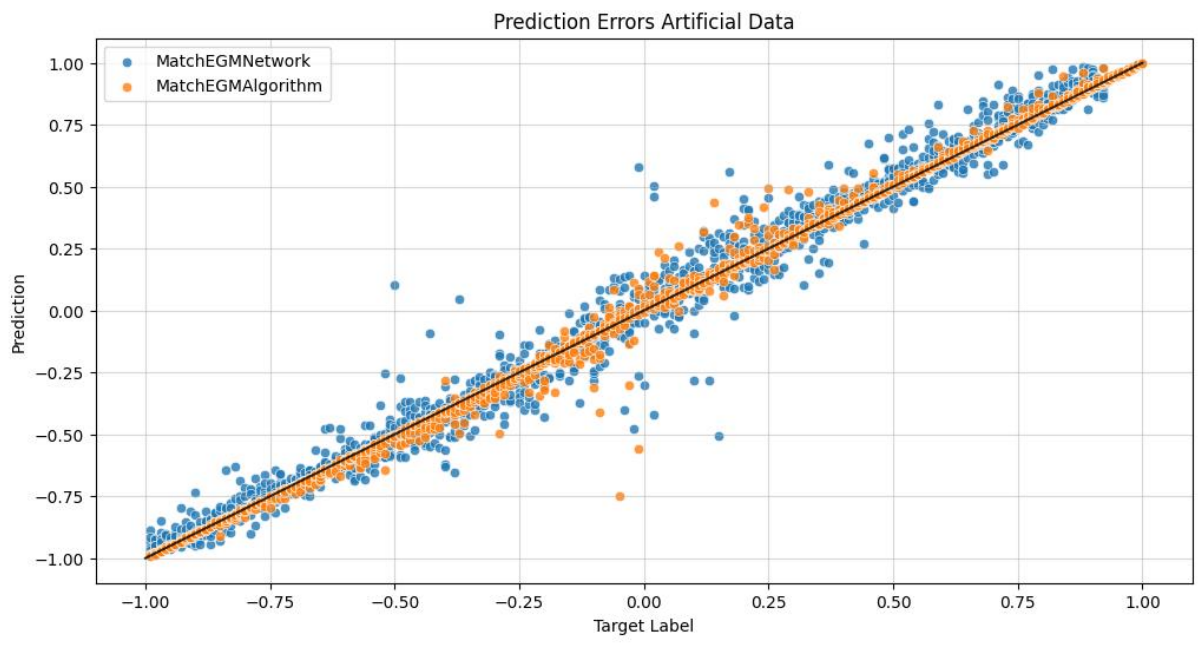


Figure 38 Prediction Errors Artificial Holdout.

Looking at the plot we can observe both distributions following the regression line with very little deviation. However, blue data points show a stronger divergence than orange ones, accounting for higher MAE and MSE values. Some more outliers exist for the pretrained network, explaining the higher difference(ratio) in MSE values between both methods. Predictions falling short of the bisector seem to be analogous for both methods, while only blue points overshooting exist.

8.4.2 ECG Holdout

Turning to the ECG holdout, table 11 depicts the results achieved respectively, albeit adding evaluation metrics scored by the network trained solemnly on real data (averaged over the folds).

Model	**MAE**	**MSE**	**R2**
Network Pretrained	0.0793	0.0188	0.9500
Algorithm	0.1024	0.0524	0.8503
Network K-Fold[18]	0.1519	0.0680	0.7785
BaseLine	0.7875	0.9245	-1.5901

Table 11 Final Results ECG holdout.

Starting our analysis from top to bottom, we can see, that the pretrained network fairs best, with an MAE of 0.0792, followed by the algorithm with a score of 0.1024. The network trained exclusively on ECG data comes in third with an MAE of 0.1519. As expected, our baseline shows the worst evaluation scores with 0.7875, thus we can conclude that all methods devised outperform the baseline significantly. Moreover we can observe, that although we use the same network setup (save for the optimizer), pretraining the network with artificial data improves performance by roughly 47.86% (from 0.1519 to 0.0793). While the pretrained network performs roughly 24% better than the deterministic solution by MAE, this difference increases for MSE to about 63% (0.0524 MSE algorithm vs 0.0188 MSE pretrained network), indicating that the network has significantly fewer large prediction errors. The

[18] Metrics of this row are not evaluated on the test holdout but averaged over 5 folds

difference between algorithm and K-Fold network stays roughly the same with 31.4% improvement in MAE and 26% in MSE. Turning to R2 correlation, we can say that the pretrained solution explains significantly more of the overall variance (95%) compared to 85.58% algorithm, 77.85% K-Fold network and -159.01% baseline. Except for the baseline, which similar to results on artificial data shows that no meaningful explanation takes place, the difference between the other three approaches is roughly 0.1 in descending order. Thus, the deterministic approach explains 10% more of total variance than the non-deterministic ECG one, while the pretrained network explains another 10% more than the algorithm. Since our pretrained approach is certainly superior to the ECG one, we choose to discard the latter during further investigation, as well as the baseline for better oversight. Figure 39 shows once again the prediction errors for our top two approaches, albeit on the ECG holdout.

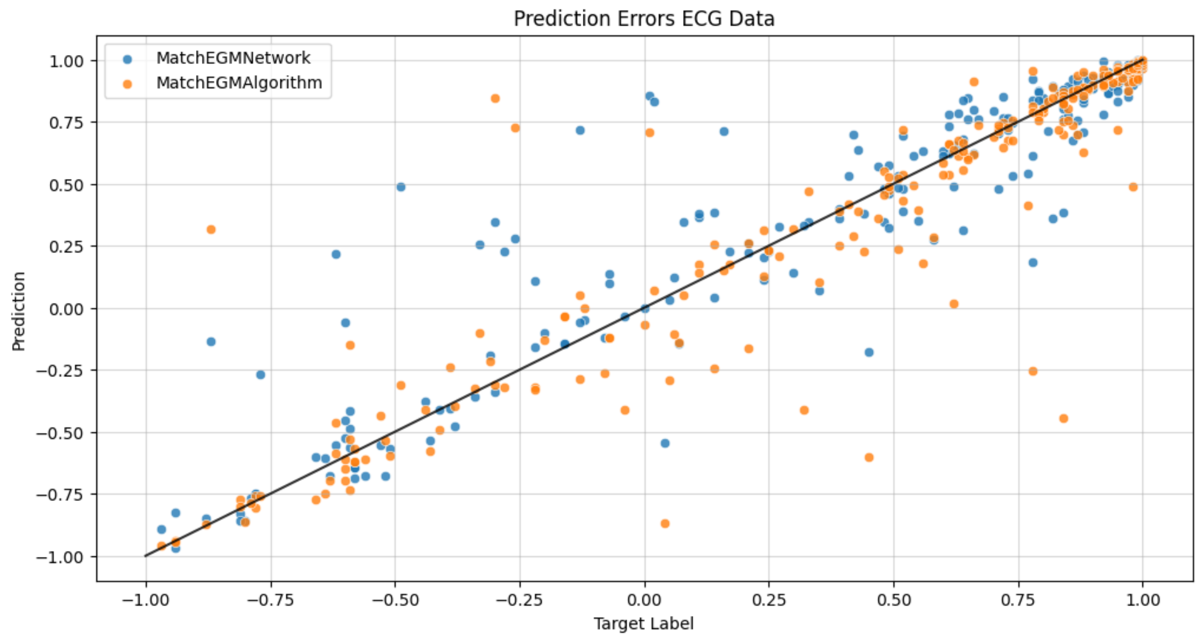


Figure 39 Prediction Errors ECG Data.

First off, we can observe that both groups show a similar distribution around the bisector, albeit orange data points being more below the curve, while blue points are more above it. This indicates, that our pretrained network tends to overshoot predictions while the algorithmic predictions fall short of the label. Most deviation can be noticed in the mid-range, while points around low and high values tend to be quite close to the regression line. Thus, both methods handle high and low correlations significantly better than midrange predictions.

Comparing our results with predictions made on artificial data, we can say, that both setups fair significantly better on artificial data than on ECG one. In general, data points appear much more scattered on the second prediction plot than on the first, supported by much higher MAE and MSE values. The non-deterministic approach outperforms the algorithm on this second data source by 22.56% (0.0793 network to 0.1024 algorithm) whereas the algorithm beats the pretrained network on artificial data by 69.87% (0.0531 network to 0.0160 algorithm). Thus, we decide to keep both approaches, since they seem to differ in their strengths (overshooting vs undershooting on ECG data). Since predictions made by the algorithm can easily be analyzed and understood, by looking at the extracted signal, comparing it with the original image, no further work on explanation is required. However, explaining predictions made by our pretrained network, is much more difficult. Thus, the following chapter wants to introduce an explanation technique.

8.5 Model explanation

In order to understand how our model predictions are made we want to investigate the process of correlation prediction using a modern explanation technique: Integrated Gradients.

8.5.1 Integrated Gradients

Integrated Gradients is a method published in 2017 in the corresponding paper "Axiomatic Attribution for Deep Networks" by Sundararajan et al. [50]. Its main goal is to explain the prediction of a neural network with regards to its input features. For this purpose, the gradients of the network are used. The method is based upon two axioms, which we want to introduce in the following: The first axiom is Sensitivity: „An attribution method satisfies Sensitivity if for every input and baseline that differ in one feature but have different predictions, then the differing feature should be given a non-zero attribution. If the function implemented by the deep network does not depend (mathematically) on some variable, then the attribution to that variable is always zero."[50, p. 2]. Using normal gradients can violate this axiom, since the „[…] prediction function may flatten at the input and thus have zero gradient despite the function value at the input being different from that at the baseline"[50, p. 3]. The authors illustrate this, by using a simple one variable, one ReLU network, modelled by the function $f(x) = 1 - \mathrm{ReLU}(1-x)$. Figure 40 depicts the function values for $0 <= x <= 2$. Sundararajan et al. argue that by choosing x=0 as baseline and x=2 as input value, the y-value changes between baseline and input from 0 to 1. However, this change is not reflected by the gradient as would be required by the axiom, since the function flattens for all inputs greater 1. In practice this effect can lead to gradients focusing on irrelevant features as is shown in the paper.

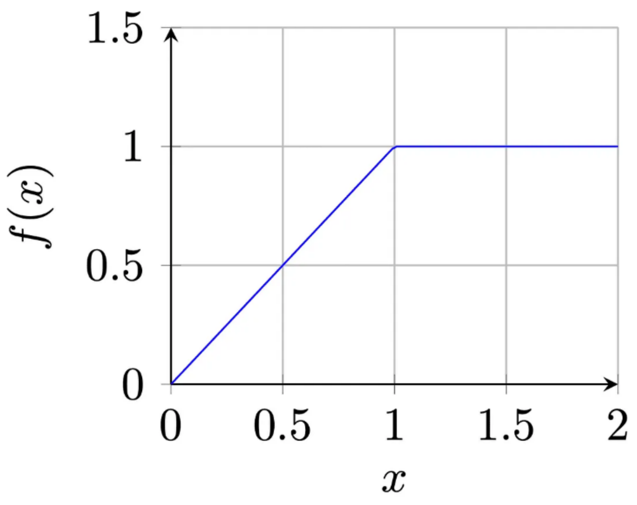


Figure 40 Breaking Sensitivity with ReLU [50].

The second axiom required is Implementation Invariance: "Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. Attribution methods should satisfy Implementation Invariance, i.e., the attributions are always identical for two functionally equivalent networks." [50, p. 2]. Gradients satisfy this axiom in general, as is proven by the authors using the chain-rule for gradients:

$$\frac{\partial f}{\partial g} = \frac{\partial f}{\partial h} * \frac{\partial h}{\partial g}$$

If g and f are viewed as the input and output of a system, with h being some implementation detail, the gradient of output f to input g can either be computed directly using $\frac{\partial f}{\partial g}$ or via the chain rule using the implementational detail h. Thus, implementational invariance is preserved. In order to overcome the violation of Sensitivity, Integrated Gradients is introduced. The main idea of this technique starts off with a function $\mathrm{F} : \mathrm{Rn} \rightarrow [0, 1]$ representing a neural network. Next an arbitrary input $x \in R^n$ is chosen and compared against the baseline input $x' \in R^n$. The authors consider a straight line in R^n between x and x', computing the gradients at all points along the path. The integrated gradient of this path can be obtained by cummulating all gradients calculated. Thus, the integrated gradient along the i^{th} dimension can be modelled using the following formula:

$$IntegratedGrads_i(x) ::= (x_i - x_i') * \int_{\alpha=0}^{1} \frac{\partial F\big(x' + \alpha * (x - x')\big)}{\partial x_i} \, d\alpha$$

The method proposed satisfies a third Axiom called Completeness, which states that "the attributions add up to the difference between the output of F at the input x and the baseline x' "[50, p. 3]. The authors demonstrate that by satisfying Completeness, Sensitivity is satisfied as well. Because of the integral the original definition of Integrated Gradients is incalculable, therefore a computational approximation is introduced using m interpolation steps with Riemann trapezoids[19]. The number of interpolation steps m is a hyperparameter. Common values range between 20 and 300 interpolation steps. Applying this approximation yields the following formula which is used across various implementations:

$$IntegratedGrads_i^{approx} ::= (x_i - x_i') * \sum_{k=1}^{m} \frac{dF\,(x' + \frac{k}{m} * (x - x'))}{d\,x_i} * \frac{1}{m}$$

Recapturing Integrated Gradients from a more practical point of view, the main idea of this concept is to look at gradient magnitude for each feature (in our case for each pixel) with respect to a prediction. Since the gradient alone does not provide sufficient information about feature importance, due to saturation during training, a cumulative approach is used. Thus, Integrated Gradients adds up the feature importance for each step, yielding a final value indicating the magnitude of influence each pixel has on the final prediction result. Large gradients indicate high feature importance with large positive values showing strong prediction support, while large negative values indicate key differences. Applying this methodology to a classification task the actual value of a pixel indicates if it is characteristic for identifying (large positive) or discarding (large negative) the predicted target class. Figure 41 illustrates the computational approximation of Integrated Gradients using a black image as base line and $m=5$ interpolation steps. With each step, the alpha value of the original input image is increased, ranging from 0.2 to 1.0. This means that with every image the intensity of pixel values is rising until the final image is displayed. Gradient values for each pixel can be accumulated across all images (omitting the baseline), resulting in a final value which corresponds to the notion of feature importance per pixel.

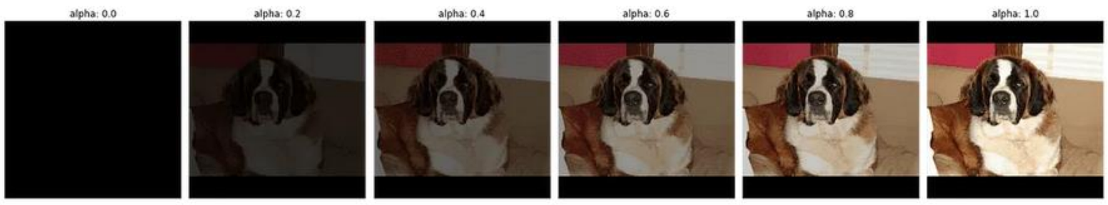


Figure 41 Alpha Interpolation and Base Line using Integrated Gradients [52].

[19] For more information on integral calculation and Riemann sums see: [51]

Note that as of today there is an ongoing discussion about choosing a suitable base line for this procedure. The key point of this debate concerns images where black pixels have some importance for calculating the prediction value, e.g. black pixels that are not simply noise. In these cases a black image cannot be used as a base line and other approaches need to be utilized. However, since we are only concerned with images where a white signal is depicted on a black background, no further adjustment of the baseline image is necessary.

8.5.2 Analyzing Prediction Results

Applying the method described on our test holdouts, we choose to calculate the gradients exclusively on the second image pair. Thus, we take the first image as a template given, analyzing how different parts of the second image contribute to the correlation value determined. Therefore, we will display positive and negative contributions marked with green and red color accordingly. To obtain a better overview of the mechanism, we will plot the template in white, with the calculated gradients from the second image as overlay. Hence, we can see both signals mapped on top of each other. Green areas indicate pixels contributing to a large positive value (a match), red areas on the other hand illustrate pixels contributing to a large negative value (a mismatch). The intensity of the color indicates the magnitude of the corresponding pixel. We start off by executing our method on the artificial test holdout. Some sample results are depicted below in figure 42:

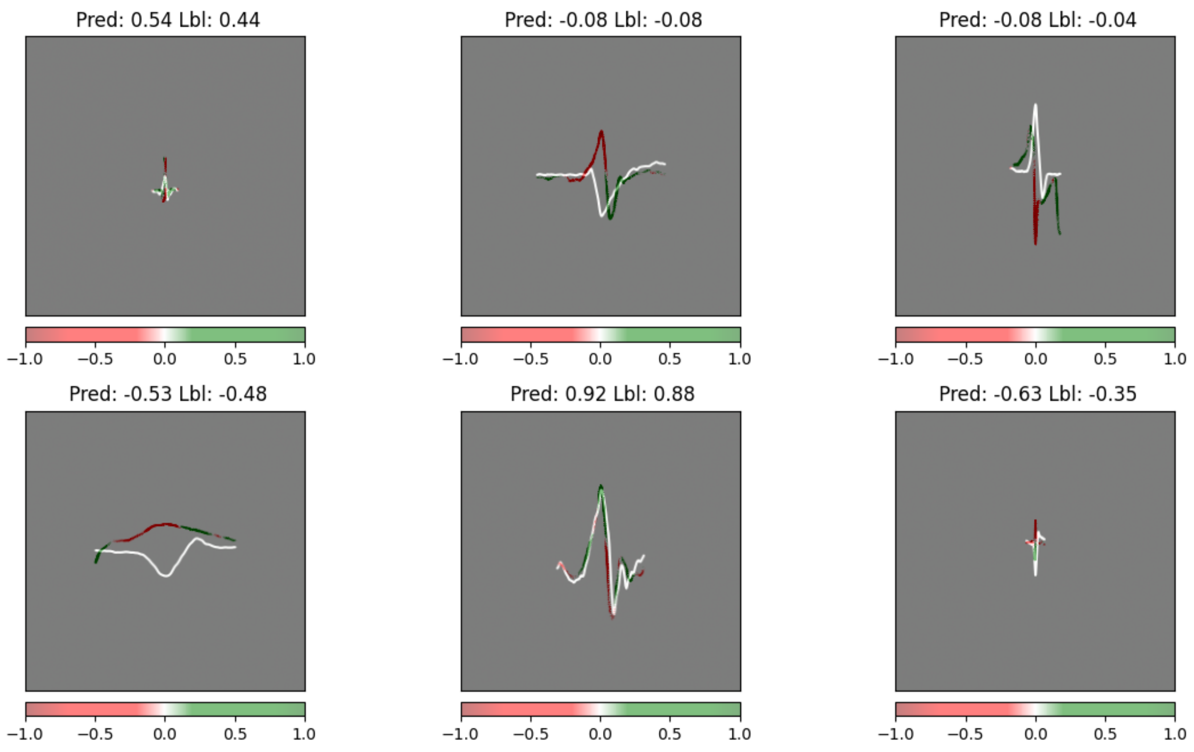


Figure 42 Integrated Gradients on Artificial Test Holdout.

Looking at the image in the middle of the first row, we can observe how the red peak of the matching signal clearly indicates the largest mismatch detected between the two signals. However, since the matching signal also shows a downward peak, which is slightly shifted, the network recognizes some similarity with the template, showing itself by the green color. Summing up both peaks we end up with a predicted correlation value of -0.08, which corresponds to the target label. Thus, in this case the network correctly identifes both major components which oppose each other yielding a small correlation value. Moreover, it is able to extract a slightly larger influence by the upward peak resulting in the negative correlation value. Taking a look at the image in the middle of the second row, we can observe a correct attribution of the upweard peak, colored in green. However, the downward peak is colored slightly red, showing a negative contribution to the correlation value. A possible explaination might be a slight shift (x-wise) between the two signals, resulting in the downward peak of the matching image occuring before the downward peak of the template. Nevertheless, we obtain a high correlation prediciton of 0.92 which is higher than the label attributed of 0.88. Thus, some inaccuracy in signal extraction can be noticed. Next the same procedure is executed for the ECG test hold out, with some of the results depicted in figure 43:

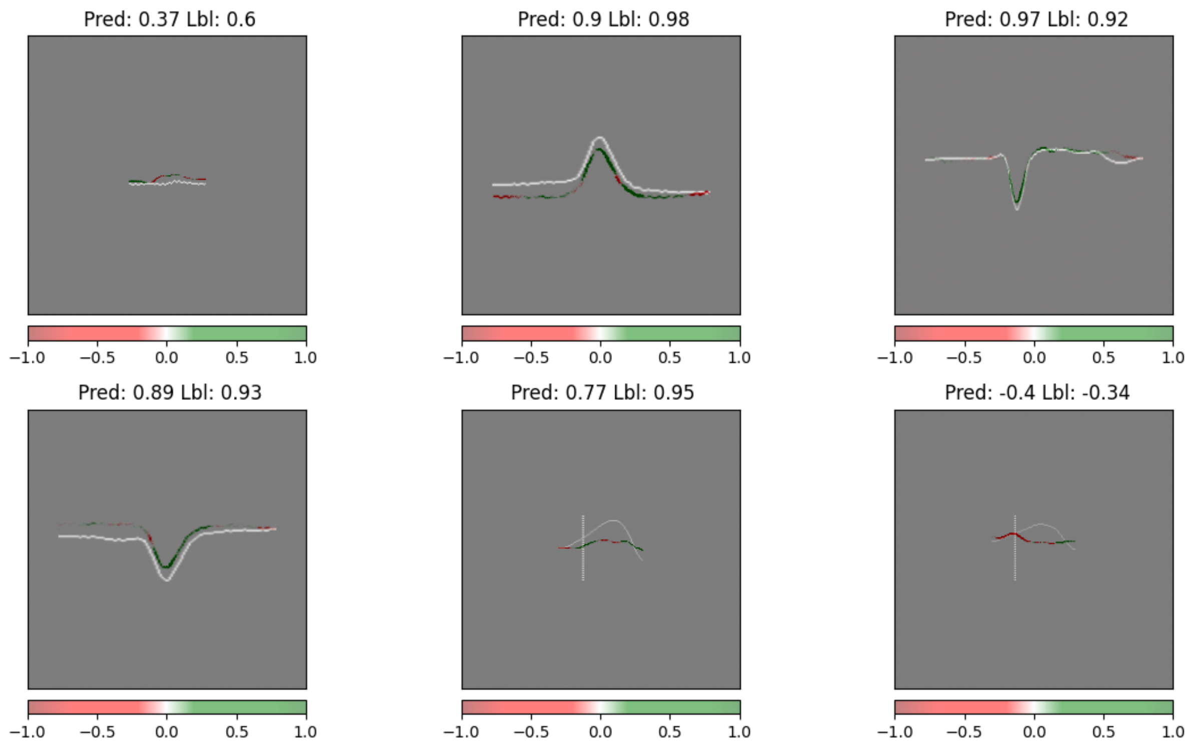


Figure 43 Integrated Gradients on ECG Holdout.

Once again, we can observe successful peak attribution by looking at the image in the middle of the first row, where the upward peaks match. Similar results can be observed for the image on the left of the second row, where the downward peaks correspond. Nevertheless, both predictions fall slightly short of the label value (0.9 vs 0.98 and 0.89 vs 0.93), which might be caused by the upward/downward shift of the matching image, appearing slightly above/below the template. The last two images in the second row depict image pairs containing a vertical disturbance. We can see, that the dotted line does not disrupt our matching procedure, since we remove it beforehand during preprocessing. However, looking at the area around the disturbance, we can notice that in the first image positive correlation in the beginning of both signals is identified correctly, while in the second image the initial signal match is inaccurate. Still, the prediction value of the first example is significantly more off (0.77 predicted vs 0.95 label) than in the second one (-0.4 predicted vs -0.34 label). A possible conclusion from this observation could be, that although correlating parts are identified correctly in the first example, their overall contribution is unbalanced, which leads to an overproportionate influence of negatively contributing values. This notion corresponds with the results observed on the artificial test hold out, from the section above (figure 42). Thus, potential model improvement could focus on the distance layer of the network, which is responsible for processing the embedding vectors. Overall, this method provides an interesting approach on analyzing and understanding prediction results. Therefore, further research and optimization attempts could utilize this procedure.

9 Deploying Match EGM

After finishing the chapter on interpretability of prediction results from the neural network approach, we now want to focus on creating the application for use in the laboratory. The main goal of Match EGM is enabling doctors to compare two signals at any time on any device and in any place. Therefore, we choose to deploy the software as a web application. This enables us to provide a device independent solution, that can be used with any operating system. Moreover, we do not have to worry about security compliance. Taking images of a patient's heart rhythm has to follow the General Data Protection Regulation (GDPR)[53], since this data is not only personal but also belongs to a special category of personal data according to "Article 9 GDPR" [54]. Thus, special security and protection measures apply. Running the application on a mobile or a personal device ensuring these rather high standards could pose a challenge. By choosing to use a web application, we mitigate these risks, since the web server can be hosted using the UKSH internal IT infrastructure. Therefore, we can rely on already managed and secure hardware and transmit our images only via the internal UKSH Wireless Lan (UKSH secure). Figure 44 depicts the main architecture of Match EGM:

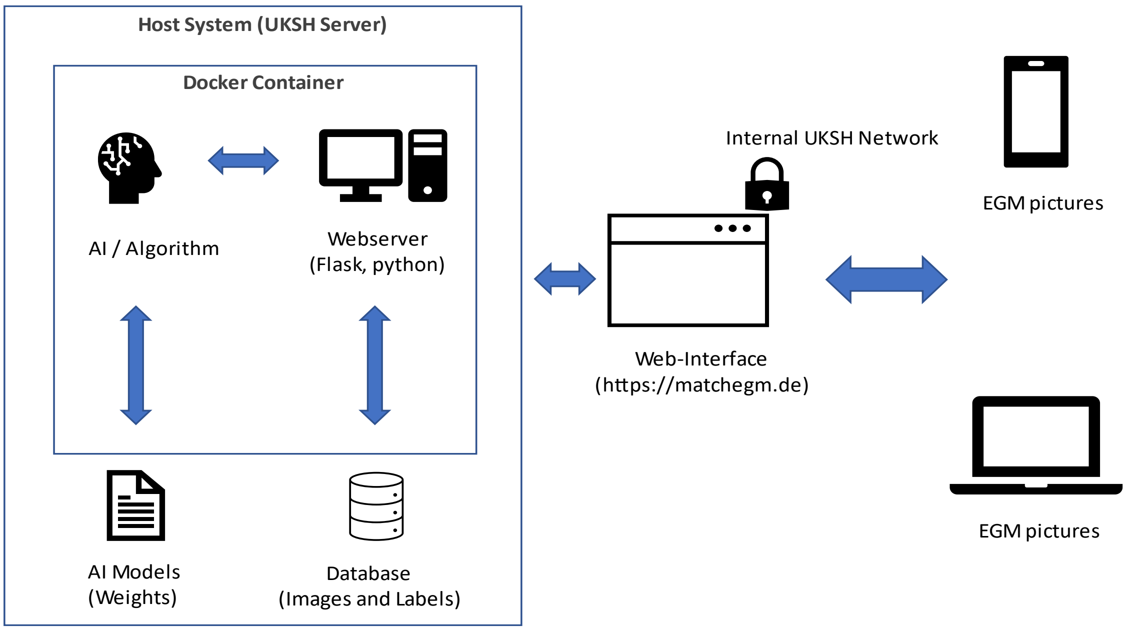

Host System (UKSH Server)
Docker Container
AI / Algorithm
Webserver (Flask, python)
AI Models (Weights)
Database (Images and Labels)
Internal UKSH Network
Web-Interface (https://matchegm.de)
EGM pictures
EGM pictures

Figure 44 Main Architecture Match EGM.

The core of our system will be built with the micro web framework[20] Flask [58]. Its main characteristics are a lightweight and simplistic approach, which enables users to build web applications. Since Flask is written in python, we can integrate it seamless with our already developed prediction software, by importing the python scripts with the preprocessing blocks, models and algorithms included. Flask utilizes Jinja as a web template system[21] with which we can provide dynamic web templates that generate custom web pages (HTML code) with data passed from the server [60]. Together with Bootstrap[22] [62] and JavaScript, a simple web application with a uniform look and feel can be created.

[20] Web application frameworks are software frameworks [55] that "provide a standard way to build and deploy web applications"[56]. Microframeworks are a subcategory of web application frameworks. The main difference is lack of core functionalities (user authentication, data base abstraction etc. [57].

[21] Web template systems speed up web development by providing the opportunity to create reusable dynamic templates which can be individually shaped depending on the use case. For more information see [59].

[22] A sophisticated CSS framework [61] which makes all HTML elements automatically resize and align to the host display size.

The front end of Match EGM consists of three HTML files: *base.html*, *index.html* and *results.html*. The first file is responsible for the main application layout as well as some simple error handling. Via Jinja it is added to all subsequent web pages by simply extending it in the beginning of the HTML code. The second file, *index.html* represents the main page of our web app. It provides the user interface for uploading new images, either directly via camera or via file upload. In addition a preview of the uploaded data (using JavaScript) is shown. After clicking the submit button, all data is send to the server via POST request.

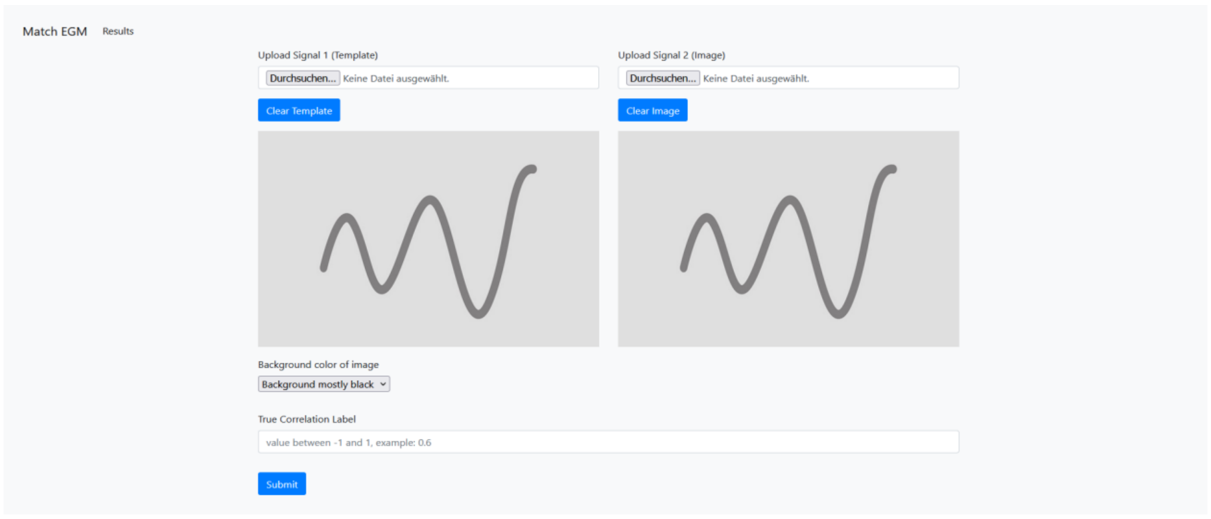
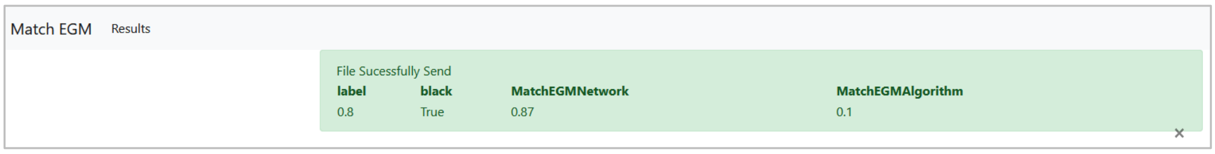

Match EGM
Results
Upload Signal 1 (Template)
Durchsuchen... Keine Datei ausgewählt.
Clear Template
Upload Signal 2 (Image)
Durchsuchen... Keine Datei ausgewählt.
Clear Image
Background color of image
Background mostly black
True Correlation Label
value between -1 and 1, example: 0.6
Submit

Figure 45 Main Webpage.

The input form itself (figure 45) allows for two additional parameters: “Background color of image” and “True Correlation Label”. The first parameter is used for data preprocessing, since both prediction methods expect images to be on a white background. If a different coloring is present, an internal invert is applied. The second field is optional. It is designed for improving the prediction methods. If the pictures taken can be labelled, e.g. by taking further images from LABSYSTEM PRO or utilizing another data source, these labels will be stored together with the uploaded image pairs. The stored information can be extracted und used for retraining the neural network or improving preprocessing methods in general. If an incorrect picture has been uploaded, it can be removed via the “Clear Template” / “Clear Image” button. Once the data is submitted using the “Submit” button, it is send to the webserver, which internally preprocesses the images and feeds them to both prediction methods. Afterwards, the prediction result is send back to the user and pops up in a green banner on top of the screen (figure 46).

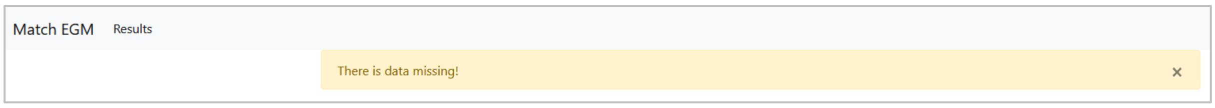


File Sucessfully Send

label	black	MatchEGMNetwork	MatchEGMAlgorithm
0.8	True	0.87	0.1

Figure 46 Successful data transfer

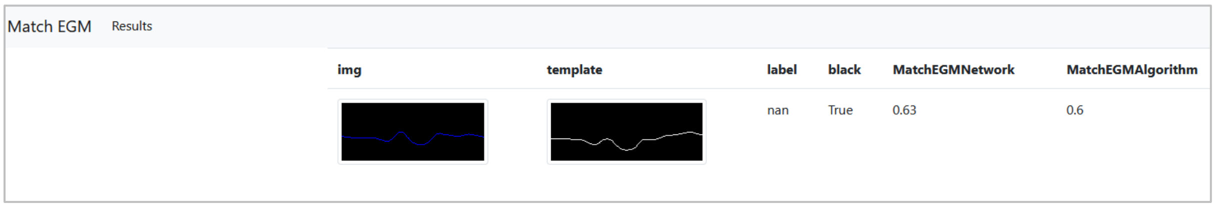
If data is missing or an incorrect label is provided, a corresponding error message will be displayed (figure 47).


Match EGM Results

There is data missing! ×

Figure 47 Error Message.

All images taken can be observed in the “Results” section, which can be accessed through the navigation bar. Internally, this page is controlled by the file *results.html*. It provides a table containing all prediction results, as well as corresponding additional information and the original input data (figure 48).


Match EGM Results

img	template	label	black	MatchEGMNetwork	MatchEGMAlgorithm
		nan	True	0.63	0.6

Figure 48 Results Page.

On the backend side the complete web server is managed by the file *app.py* which contains the two endpoints “/” and “/results”. The first connection represents the main route, providing a GET and a POST method. This endpoint is responsible for the main page which operators use for uploading their images. It passes the incoming data to the evaluation models, returns the prediction results to the user and saves all information provided in a pandas DataFrame. The second endpoint corresponds to the results section, offering only a GET method, which is used to load and display the DataFrame, containing all image paths, coloring information and the prediction results. Looking a bit deeper into the implementation, clicking the submit button invokes the function “evalimgs” in the backend by passing the request method POST to the routing endpoint “/”. This in turn calls the function “importimg” for loading both images just saved on the server and executes the prediction for both models stored. While the architecture of the neural network used is defined by the file *MainNetDefinitions.py* the general setup of the deterministic algorithm is managed by the file *AlgorithmDefinition.py*. All functions necessary for loading the models, executing preprocessing steps and loading the images are stored in the script *MainFunctions.py*. This way we provide two separate model classes, which can be instantiated for prediction execution, while at the same time keeping flexibility, sharing analogous preprocessing steps used by both models. All images are saved in the dedicated folder *images* which separates them into templates and images, using the according prefix /temp and /img). In addition the file *currentint.txt* stores a running integer number, which will be increased each time an image pair is inserted. The file *logging.csv* represents the DataFrame containing the image paths, predictions, labels and parameter settings. The MatchEGMNetwork model with the pretrained weights is stored in the folder *neuralnets*. Combined with the csv file *netparams.csv* we can use the architecture definitions stored in *MainNetDefinitions.py* to reconstruct any model previously designed and trained. All data mentioned is stored in a global folder named *static*, which represents static information, globally accessible during runtime of the web application. This gives us the flexibility to improve and exchange model types during the life cycle of Match EGM. We can extract all images with their corresponding labels, retrain the network and exchange the weights file stored in the corresponding subfolder. In addition if required, the architecture of the model can be changed to some extent as well, modifying settings in the *netparams.csv* file (e.g. number of CNN layers etc.). However, a server restart is required, since model weights and structure are loaded initially once during startup. Note, that for a production environment this setup should be changed to use a database, e.g. SQLite storing the csv and txt files. Nevertheless, all images should be stored as is, since we choose to avoid storing them in a database, thereby omitting image conversion (e.g. into a BLOB file) which is costly und time consuming [63]. However, while still in refinement, utilizing csv files yields more flexibility resulting in faster adaptation cycles.

In order to make the web server run independent of the used host system we use docker, virtualizing the flask server environment [64]. We wrap our server in a docker container and create a docker file, which can be used on any other host system to reproduce and start the application. The main file for this task is the *Dockerfile* located in the main *Flask-App* directory. This file contains instructions for building up the environment needed for our server. The file *requirements.txt* contains all libraries and frameworks used to run our application, this way our virtual environment will install and provide these dependencies, e.g. flask, numpy, pandas etc. For the core of our docker image, we build on top of *python:3.10.4-slim-buster* [65], a python image containing the version 3.10.4 of python. This image builds on top of the *Debian-Slim* image, which means that we are running a simple Linux based distribution in our docker container. Figure 49 shows the dockerized Setup of Match EGM:

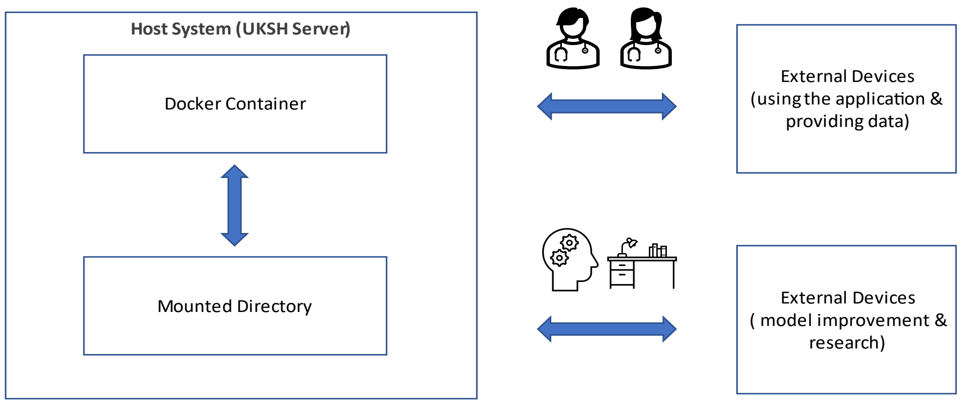

Host System (UKSH Server)
Docker Container
Mounted Directory
External Devices (using the application & providing data)
External Devices (model improvement & research)

Figure 49 Dockerized Setup of Match EGM.

Despite virtualizing our application we still want to be able to access all files uploaded and predictions produced. This way we can extract the data and use it for further research and optimization purposes as already mentioned. Because accessing data inside a docker container is not trivial, we use a concept called “Binded mounts” [66]. Binded mounts create a directory on the host machine which uses the hosts filesystem. Therefore it is fully managed and accessible by the host. This directory is than mounted into the docker container, thus making it available to our virtualized application. By performing a binded mount with the *static* directory, mentioned earlier in this chapter, we can extract data and exchange the models used as illustrated above. As mentioned, model weights and structure are loaded only once during startup off the application, thus restarting the docker container is necessary (analogous to the web server restart). The major benefit of this setup is fast adaptation and improvement of the prediction model used without the need of new deployment. We provide our docker image in two different versions, one compiled for ARM (arm64) the other for x86 (amd64) based CPU architectures in order to use MatchEGM on windows and apple based hardware alike. This is particularly useful during testing phase, because several electrophysiologists can run individual web servers on their private laptops. Nevertheless, all results can be extracted from the static folder and aggregated globally on a shared drive, thereby providing the opportunity to improve upon all results collected. Appendix III illustrates how to run the corresponding container, given the docker image.

10 Conclusion and Outlook

Summarizing the results of this work, we can conclude, that both goals formulated, have been successfully achieved. On the one hand, we have developed a web application called MatchEGM, which can predict the correlation coefficient of two signals given as an image pair. Moreover, our application is capable of receiving continuous new information via new match images, comparing it with a predefined template, which can be exchanged if necessary and delivering the correlation scores in real time. Images itself can be taken with any device chosen, e.g. smartphone cameras, laptop and desktop screenshots and the simple interface, encourages medical staff to make use of its predictions. From the backend side, the web server can safely be administrated from within the UKSH IT infrastructure, providing maximum data security and compliance. In addition, the application provides the possibility of a continuous learning cycle, by extracting collected images (match and template), with the optionally provided correlation label, retraining the prediction models used and updating the logic implemented.

On the other hand, we have investigated several techniques, for extracting time series data from an image given, resulting in two successfully implemented prediction models. The first model is based on a deterministic approach, which extracts the underling signal plot from each image given, utilizing the pixel transformation method presented by Sood et al.[15]. The second model combines this transformation technique with the prediction capabilities of a Siamese neural network. Reviewing the architecture constructed in this thesis, we can observe, that a leaner and simpler network structure has been devised, yielding satisfying results on the ECG dataset acquired as well as on the artificial test hold out. Thus, we believe that we have improved upon the original non-deterministic prediction model developed, by utilizing relevant research and constructing new surrogate datasets, which capture the surrogate target data (ECG data) more accurately, than would have been possible with the old approach[23]. Comparing the two prediction models on our ECG test hold-out, we observe, that using MatchEGMNetwork yields a 24% performance improvement, when compared to the MatchEGMAlgorithm. Analyzing the interpretability of the results produced, the Integrated Gradients method presented, provides a very promising visual explanation of matching signal parts detected by the network. Using this approach, prediction results gain a significant amount of credibility, because individual correlation scores can be investigated and questioned. Taking a closer look at the artificial training data, we conclude, that higher data variance with regards to domain specific parameters: signal length, image size and y-axis limit (image zooming), yields significantly better prediction results, with an overall performance improvement of 48.91% over uniform AI data (measured on the ECG dataset). In addition, we have demonstrated, that in general, image signal pairs can be approximated, using an artificially created surrogate dataset, showing a 47.86% prediction error improvement, when compared with the same model architecture trained purely on ECG data. These results give us some confidence, that the approach taken, will also yield successful results on the EGM data targeted.

However, since no labelled EGM images exist, the true usefulness of our prediction algorithms needs to be evaluated by domain experts. Thus, the final verdict is still outstanding. Considering the two different prediction methods applied, we are able to predict the correlation score with an MAE of 0.0793, when using the neural network and 0.1024, when using the deterministic algorithm (based on results from the ECG dataset). Although these values provide sufficient prediction strength, overall, better results should be possible, as can be seen by the severe overfitting occurring with regards to

[23] Note: Although no ECG data has been used in the HazelNet approach, we believe that due to uniform artificial surrogate training data, results would have been worse, if compared to the predictions made by MatchEGMNetwork.

ECG data. As already mentioned earlier, we believe, that a larger ECG dataset size, would benefit the experiments significantly. Because data acquisition is not easy, one possible solution could include, constructing a Generative Adversarial Networks (GANs) [67] for creating additional artificially generated ECG images, which can then be used by our neural network for result improvement. Further training data improvement could include collecting images which have been taken by smartphone camera, since so far, all training images have been taken via screenshots (in this case, a similar artificial creation methodology with GANs could be applied as well, given a large enough initial dataset). In addition, further experiments could include image pairs which are not completely identical in terms of image dimensions. This constraint has been used during this work, in order to focus on the general feasibility of this task. Nevertheless, we can expect images to vary slightly in size, when taken by smartphone cameras. Thus, it remains to be seen, if such data can be processed by the prediction algorithms provided. Using the knowledge acquired from the Integrated Gradients method, further improvement could focus on the distance layer, because first results indicate, that calculation errors between the matching and none-matching parts of two signals and the resulting correlation value exist. Thus, even if the signal embedding vectors are correct, some calculation error might persist. One possible alteration could focus on changing the computation of the correlation coefficient, using the area under the curve as a similarity measure or applying a binning procedure, the latter altering the type of problem to a classification task. Both approaches could provide a benefit, because they might be more robust and precise in terms of processing the embedding vectors. Thus, a tradeoff between modelling the problem as precise as possible and pursuing a surrogate problem, which might yield a more robust solution, might be considered. Moreover, we suggest to add a dedicated explanation page, to the MatchEGM application developed, providing the possibility, to execute the Integrated Gradients method individually, to any image pair inspected by the operator. Finally, as encountered quite often in the domain of neural network training, additional computational capacity, would encourage significantly larger experiments, with more hyperparameters investigated and larger artificial surrogate datasets used.

List of references

[1] “Electrophysiology,” *Wikipedia*. May 17, 2023. Accessed: May 25, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Electrophysiology&oldid=1155293057

[2] “Cardiology,” *Wikipedia*. May 19, 2023. Accessed: May 25, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cardiology&oldid=1155715861

[3] “Atrial Fibrillation | cdc.gov,” *Centers for Disease Control and Prevention*, Oct. 14, 2022. https://www.cdc.gov/heartdisease/atrial_fibrillation.htm (accessed May 25, 2023).

[4] S. Bernstein, “Cardiac Ablation,” *WebMD*. https://www.webmd.com/heart-disease/atrial-fibrillation/what-is-cardiac-ablation (accessed Apr. 17, 2023).

[5] dfornell, “Pulsed Field Ablation Successfully Treats Atrial Fibrillation,” *DAIC*, May 08, 2020. http://www.dicardiology.com/article/pulsed-field-ablation-successfully-treats-atrial-fibrillation (accessed Apr. 17, 2023).

[6] M. bei DocCheck, “Pulmonalvenenisolation,” *DocCheck Flexikon*. https://flexikon.doccheck.com/de/Pulmonalvenenisolation (accessed May 25, 2023).

[7] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “From Data Mining to Knowledge Discovery in Databases,” *AI Mag.*, vol. 17, no. 3, Art. no. 3, Mar. 1996, doi: 10.1609/aimag.v17i3.1230.

[8] S. Benhur, “A friendly Introduction to Siamese Networks,” *Medium*, Jun. 05, 2022. https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942 (accessed Jul. 10, 2023).

[9] M. Jeon *et al.*, “ReSimNet: drug response similarity prediction using Siamese neural networks,” *Bioinformatics*, vol. 35, no. 24, pp. 5249–5256, Dec. 2019, doi: 10.1093/bioinformatics/btz411.

[10] “ZINC database,” *Wikipedia*. Jul. 23, 2022. Accessed: Jul. 10, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ZINC_database&oldid=1099871275

[11] M. D. Korzec, “Effortlessly Recommending Similar Images,” *Medium*, Oct. 31, 2020. https://towardsdatascience.com/effortlessly-recommending-similar-images-b65aff6aabfb (accessed Jul. 10, 2023).

[12] fareid, “Image similarity using feature embeddings,” *Medium*, Apr. 03, 2023. https://medium.com/@f.a.reid/image-similarity-using-feature-embeddings-357dc01514f8 (accessed Jul. 10, 2023).

[13] P. Schulz, Jan Michael, “HazelNet (Application Project am UKSH).” Unpublished (FH Kiel Fachrichtung Datascience), 2023.

[14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[15] S. Sood, Z. Zeng, N. Cohen, T. Balch, and M. Veloso, “Visual time series forecasting: an image-driven approach,” in *Proceedings of the Second ACM International Conference on AI in Finance*, Virtual Event: ACM, Nov. 2021, pp. 1–9. doi: 10.1145/3490354.3494387.

[16] T. A. L. The Beth Israel Deaconess Medical Center, “The MIT-BIH Normal Sinus Rhythm Database.” physionet.org, 1990. doi: 10.13026/C2NK5R.

[17] “Electrophysiology (EP) Recording System – LABSYSTEM PRO,” *www.bostonscientific.com*. https://www.bostonscientific.com/en-EU/medical-specialties/electrophysiology/arrhythmias/cardiac-mapping-system/electrophysiology-recording-system.html (accessed Apr. 18, 2023).

[18] “Sinusrhythmus & Sinusknoten | Ratgeber Herzinsuffizienz.” https://www.ratgeber-herzinsuffizienz.de/herzinsuffizienz/herzfunktion/sinusrhythmus (accessed Jul. 11, 2023).

[19] M. bei DocCheck, “Sinusbradykardie,” *DocCheck Flexikon*. https://flexikon.doccheck.com/de/Sinusbradykardie (accessed May 30, 2023).

[20] M. bei DocCheck, “Sinustachykardie,” *DocCheck Flexikon*. https://flexikon.doccheck.com/de/Sinustachykardie (accessed May 30, 2023).

[21] “Anatomie und Aufbau | Herzstiftung.” https://herzstiftung.de/ihre-herzgesundheit/das-herz/anatomie-und-aufbau (accessed May 30, 2023).

[22] M. bei DocCheck, “Elektrokardiogramm,” *DocCheck Flexikon*. https://flexikon.doccheck.com/de/Elektrokardiogramm (accessed May 30, 2023).

[23] M. bei DocCheck, “Herzvorhof,” *DocCheck Flexikon*. https://flexikon.doccheck.com/de/Herzvorhof (accessed May 30, 2023).

[24] “scipy.signal.find_peaks — SciPy v1.10.1 Manual.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html#scipy.signal.find_peaks (accessed May 02, 2023).

[25] “scipy.signal.peak_prominences — SciPy v1.10.1 Manual.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.peak_prominences.html (accessed May 02, 2023).

[26] N. Otsu, “A Threshold Selection Method from Gray-Level Histograms,” *IEEE Trans. Syst. Man Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.

[27] “Otsu’s method,” *Wikipedia*. Apr. 27, 2023. Accessed: May 01, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Otsu%27s_method&oldid=1151941947

[28] *Otsu’s Method*, (Sep. 21, 2020). Accessed: May 12, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=jUUkMaNuHP8

[29] “OpenCV: Image Thresholding.” https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html (accessed May 12, 2023).

[30] “OpenCV: Morphological Transformations.” https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html (accessed May 17, 2023).

[31] “Tech-Algorithm.com ~ Nearest Neighbor Image Scaling.” https://tech-algorithm.com/articles/nearest-neighbor-image-scaling/ (accessed Jun. 01, 2023).

[32] A. Dhinakaran, “How to Understand and Use the Jensen-Shannon Divergence,” *Medium*, Mar. 22, 2023. https://towardsdatascience.com/how-to-understand-and-use-jensen-shannon-divergence-b10e11b03fd6 (accessed Jul. 10, 2023).

[33] D. Chicco, “Siamese Neural Networks: An Overview,” in *Artificial Neural Networks*, H. Cartwright, Ed., in Methods in Molecular Biology. New York, NY: Springer US, 2021, pp. 73–94. doi: 10.1007/978-1-0716-0826-5_3.

[34] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA: IEEE, Jun. 2014, pp. 1701–1708. doi: 10.1109/CVPR.2014.220.

[35] G. Nanos, “How Do Siamese Networks Work in Image Recognition? | Baeldung on Computer Science,” Mar. 15, 2023. https://www.baeldung.com/cs/siamese-networks (accessed May 03, 2023).

[36] “Optuna: A hyperparameter optimization framework — Optuna 3.2.0 documentation.” https://optuna.readthedocs.io/en/stable/ (accessed Jun. 04, 2023).

[37] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization,” *J. Mach. Learn. Res. 18*, 2018.

[38] “Gaussian process,” *Wikipedia*. Apr. 05, 2023. Accessed: May 03, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Gaussian_process&oldid=1148277340

[39] F. Hutter, H. Hoos, and K. Leyton-Brown, “An evaluation of sequential model-based optimization for expensive blackbox functions,” in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, Amsterdam The Netherlands: ACM, Jul. 2013, pp. 1209–1216. doi: 10.1145/2464576.2501592.

[40] W. Koehrsen, “A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning,” *Medium*, Jul. 02, 2018. https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (accessed May 03, 2023).

[41] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for Hyper-Parameter Optimization,” *NIPS11 Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, Dec. 2011.

[42] J. Bergstra, D. Yamins, and D. D. Cox, “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures,” JMLR, 2013. Accessed: Jul. 11, 2023. [Online]. Available: https://dash.harvard.edu/handle/1/12561000

[43] “Hyper-parameter optimization algorithms: a short review | by Aloïs Bissuel | Criteo R&D Blog | Medium.” https://medium.com/criteo-engineering/hyper-parameter-optimization-algorithms-2fe447525903 (accessed May 03, 2023).

[44] “optuna.samplers.TPESampler — Optuna 2.0.0 documentation.” https://optuna.readthedocs.io/en/v2.0.0/reference/generated/optuna.samplers.TPESampler.html (accessed May 03, 2023).

[45] A. Perera, “What is Padding in Convolutional Neural Network’s(CNN’s) padding,” *Medium*, Feb. 19, 2019. https://ayeshmanthaperera.medium.com/what-is-padding-in-cnns-71b21fb0dd7 (accessed Jun. 15, 2023).

[46] P. Kaushik, A. Kortylewski, A. Gain, and A. Yuille, “Understanding Catastrophic Forgetting and Remembering in Continual Learning with Optimal Relevance Mapping,” *NeurIPS 2021 Workshop Met. Poster*, Oct. 2021.

[47] M. Wei, “Forgetting in Deep Learning,” *Medium*, Dec. 18, 2020. https://towardsdatascience.com/forgetting-in-deep-learning-4672e8843a7f (accessed Jun. 04, 2023).

[48] A. ROBINS, “Catastrophic Forgetting, Rehearsal and Pseudorehearsal,” *Connect. Sci.*, vol. 7, no. 2, pp. 123–146, Jun. 1995, doi: 10.1080/09540099550039318.

[49] “Coefficient of determination,” *Wikipedia*. May 31, 2023. Accessed: Jun. 06, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Coefficient_of_determination&oldid=1157897397

[50] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic Attribution for Deep Networks,” *JMLRorg CML17 Proc. 34th Int. Conf. Mach. Learn.*, vol. Volume 70, Jun. 2017.

[51] “Riemann sum - Wikipedia.” https://en.wikipedia.org/wiki/Riemann_sum (accessed Jun. 15, 2023).

[52] K. Erdem (burnpiro), “XAI Methods — Integrated Gradients,” *Medium*, Apr. 21, 2022. https://medium.com/@kemalpiro/xai-methods-integrated-gradients-6ee1fe4120d8 (accessed May 24, 2023).

[53] “General Data Protection Regulation (GDPR) – Official Legal Text,” *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/ (accessed Apr. 26, 2023).

[54] “Art. 9 GDPR – Processing of special categories of personal data,” *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/art-9-gdpr/ (accessed Apr. 26, 2023).

[55] “Software framework,” *Wikipedia*. Apr. 06, 2023. Accessed: Apr. 26, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Software_framework&oldid=1148548565

[56] “Web framework,” *Wikipedia*. Mar. 23, 2023. Accessed: Apr. 26, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Web_framework&oldid=1146176403

[57] “Microframework,” *Wikipedia*. Jan. 25, 2023. Accessed: Apr. 26, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Microframework&oldid=1135581572

[58] “Welcome to Flask — Flask Documentation (2.3.x).” https://flask.palletsprojects.com/en/2.3.x/ (accessed Apr. 26, 2023).

[59] “Web template system - Wikipedia.” https://en.wikipedia.org/wiki/Web_template_system (accessed Apr. 26, 2023).

[60] “Jinja,” *Pallets*. https://palletsprojects.com/p/jinja/ (accessed Apr. 26, 2023).

[61] “CSS framework,” *Wikipedia*. Jan. 17, 2023. Accessed: Apr. 26, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=CSS_framework&oldid=1134143218

[62] M. O. contributors Jacob Thornton, and Bootstrap, “Get started with Bootstrap.” https://getbootstrap.com/docs/5.3/getting-started/introduction/ (accessed Apr. 26, 2023).

[63] A. Singh, “Storing images in Blob vs File System,” *Medium*, Jun. 19, 2020. https://medium.com/@anilsingh.jsr/storing-images-in-blob-vs-file-system-3d704988e44e (accessed Apr. 27, 2023).

[64] “Docker: Accelerated, Containerized Application Development,” May 10, 2022. https://www.docker.com/ (accessed Apr. 27, 2023).

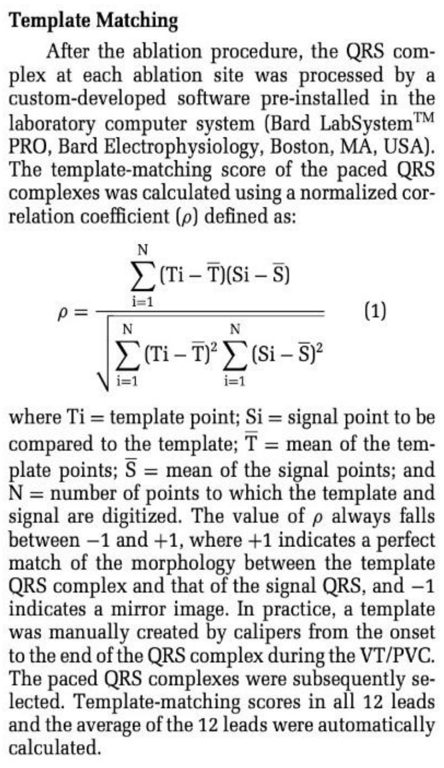
[65] “https://github.com/docker-library/python.” Docker Official Images, Apr. 25, 2023. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/docker-library/python/blob/f7bb6f4ee29f0cdc8c1dbb55b17aa4e0f38497e0/3.10/slim-buster/Dockerfile

[66] “Bind mounts,” *Docker Documentation*, Apr. 26, 2023. https://docs.docker.com/storage/bind-mounts/ (accessed Apr. 27, 2023).

[67] O. G. Yalçın, “Image Generation in 10 Minutes with Generative Adversarial Networks,” *Medium*, Jan. 12, 2023. https://towardsdatascience.com/image-generation-in-10-minutes-with-generative-adversarial-networks-c2afc56bfa3b (accessed Jul. 11, 2023).

Appendix

Appendix I. Reply from Boston Scientific (template matching method)



Template Matching

After the ablation procedure, the QRS complex at each ablation site was processed by a custom-developed software pre-installed in the laboratory computer system (Bard LabSystem^{TM} PRO, Bard Electrophysiology, Boston, MA, USA). The template-matching score of the paced QRS complexes was calculated using a normalized correlation coefficient (ρ) defined as:

$$\rho = \frac{\sum_{i=1}^{N} (Ti - \overline{T})(Si - \overline{S})}{\sqrt{\sum_{i=1}^{N} (Ti - \overline{T})^2 \sum_{i=1}^{N} (Si - \overline{S})^2}} \quad (1)$$

where Ti = template point; Si = signal point to be compared to the template; $\overline{T}$ = mean of the template points; $\overline{S}$ = mean of the signal points; and N = number of points to which the template and signal are digitized. The value of ρ always falls between −1 and +1, where +1 indicates a perfect match of the morphology between the template QRS complex and that of the signal QRS, and −1 indicates a mirror image. In practice, a template was manually created by calipers from the onset to the end of the QRS complex during the VT/PVC. The paced QRS complexes were subsequently selected. Template-matching scores in all 12 leads and the average of the 12 leads were automatically calculated.

Figure 50 Reply from Boston Scientific.

Appendix II. Hyperparameter setups per fold (inner cross validation)

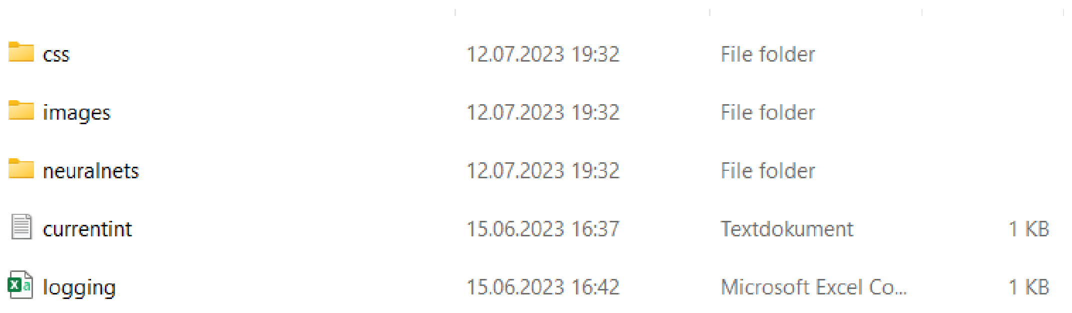
Outer Fold	**Blocks**	**Kernel Size**	**Optimizer**	**Padding**	**Stride**	**Count**	**Mae mean**
0	4	7	SGD	2	2	1	0.14496
	5	7	SGD	2	2	1	0.156406
	4	7	RMSprop	2	2	1	0.160228
	5	7	SGD	6	2	1	0.163125
	3	5	RMSprop	4	2	3	0.165409
	4	7	SGD	4	2	5	0.165664
		3	SGD	2	2	1	0.168792
1	4	7	RMSprop	2	2	1	0.158587
			SGD	2	2	2	0.175799
	5	7	SGD	2	2	1	0.177823
	3	3	SGD	2	2	1	0.183142
		7	Adam	4	2	1	0.183179

	5	7	SGD	6	2	1	0.188422
	4	7	SGD	4	2	5	0.194274
2	5	7	SGD	6	2	2	0.157601
	4	7	Adam	4	2	1	0.157964
			SGD	6	2	1	0.170791
	5	7	SGD	2	2	3	0.177816
	4	7	SGD	4	2	5	0.184383
	5	7	SGD	4	2	5	0.186059
		3	SGD	4	2	1	0.192923
3	5	7	SGD	2	2	2	0.154149
				6	2	1	0.162147
		3	SGD	2	2	1	0.164251
	4	7	SGD	4	2	5	0.177051
	3	5	RMSprop	2	2	1	0.177173
	5	7	SGD	4	2	5	0.183706
	3	7	SGD	4	2	2	0.187445
4	4	7	SGD	2	2	1	0.177314
				6	2	1	0.18534
			Adam	2	2	1	0.188898
	5	7	SGD	2	2	1	0.190922
	4	7	SGD	4	2	5	0.192933
	5	7	SGD	4	2	5	0.199212
	3	3	SGD	2	2	1	0.199219

Table 12 Hyperparameter Setups per Fold.

Appendix III. Running the Docker Container

Both docker images created, amd64 and arm64 architecture, are provided publicly via dockerhub in the repository *mpolonskiy/masterthesis.com.* Thus, pulling images can be done by executing the docker command: docker pull *mpolonskiy/masterthesis.com*. Note, that the image tag provided enables to distinguish the two architectures using it behind the general pull command, e.g. *docker pull mpolonskiy/masterthesis.com:flaskmac* (for arm64) and *docker pull polonskiy/masterthesis.com:flaskwin (*for amd64). The following command can be executed to pull the image (if not already done beforehand) and create the corresponding container: *docker run -v ${PWD}/static:/Flask-App/static --name MatchEGM -d -p 5000:5000 mpolonskiy/masterthesis.com:flaskwin*. This instruction creates a docker container which is mapped to the host port 5000, thus, the application can be accessed from the host network (e.g. via smartphone), using the assigned IP address of the host and adding port 5000 as destination. If needed, host port can be altered, however the dockerized web server will always run on port 5000, therefore this port needs to be mapped to the desired host port. The name of the application in this demo command is MatchEGM, which can be changed to any name wanted. The image chosen is flaskwin, the other option would be calling mpolonskiy/masterthesis.com:flaskmac. As explained in chapter 9, we want to access and exchange data between the host system the container is running on and the web application itself, running inside the container. Therefore, a binded mount is created, connecting the host folder ${PWD}/static to the application folder /Flask-App/static. While the latter cannot be altered, changing the binded location on the host device is possible. The variable ${PWD} provides the absolute path on the host system to the level currently accessed via terminal and can be changed to any absolute path wanted. Note, that it is still necessary to provide a folder called "static" at the path location chosen. This folder needs to contain all exchangeable static information used by the application, thus the following data needs to be present:



css	12.07.2023 19:32	File folder	
images	12.07.2023 19:32	File folder	
neuralnets	12.07.2023 19:32	File folder	
currentint	15.06.2023 16:37	Textdokument	1 KB
logging	15.06.2023 16:42	Microsoft Excel Co...	1 KB

Figure 51 Data inside static folder.

The folder *css* contains at least one single file called style responsible for the application style. The location *Images* can be empty, since here all uploaded images will be stored. The folder *neuralnets* contains the model definition currently used by the MatchEGMNetwork, thus a file with model weights called *MatchEGMNetwork* and a *netparams.csv* file need to be present, to create a model architecture and load the trained model. The file *currentint.txt* contains the current running integer value used for image identification. Finally, the file *logging.csv* contains logged image information such as path to images, background parameter set, optional correlation label and prediction scores of both models. This file can be omitted, because it will be created with first data upload, if not present. The files necessary can be accessed via GitHub using the following url: *https://github.com/mpolonskiy1995/MatchEGM.git*.

Formal Declaration

Ich versichere, dass ich die Masterarbeit **Match EGM: A mobile application for signal comparison using Siamese networks** selbständig und ohne unzulässige fremde Hilfe angefertigt habe und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet und die entsprechenden Quellen angegeben habe. Diese Arbeit hat noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift