Tampere University

Sebastian Siegel

# DETECTION OF SHOCKABLE HEART RHYTHMS WITH CONVOLUTIONAL NEURAL NETWORKS

Based on ECG spectrograms

Master thesis
Faculty of Medicine and Health Technology
Antti Vehkaoja
Milla Juutinen
April 2023

# ABSTRACT

Sebastian Siegel: Detection of Shockable Heart Rhythms with Convolutional Neural Networks
Master Thesis
Tampere University
Biomedical Science and Engineering: Health Informatics
April 2023

**Purpose**

Automated feature extraction combined with deep learning has had and continues to have a strong impact on the improvement and implementation of pattern recognition driven by machine learning. Systems without prior expertise about a problem but with the ability to iteratively learn strategies to solve problems, tend to outperform concepts of manual feature engineering in various fields. In ECG data analysis as well as in other medical domains, models based on manual feature extraction are tedious to develop, require scientific expertise, and are oftentimes not easily adaptive to variations of the problem to be solved. This work aims to examine automated feature extraction and classification of ECG data, specifically of shockable heart rhythms, with convolutional neural networks and residual neural networks. The precise and rapid determination of shockable cardiac conditions is a decisive step to improve the chances of survival for patients having a sudden cardiac arrest. Conventional, commercially available automated external defibrillators (AEDs) deploy algorithms based on manual feature extraction. Approximately 1 out of 10 shockable conditions is not recognized by the AED. Consequently, strategies for improvement need to be explored.

**Methods**

125 ECG recordings from four annotated cardiac arrhythmia databases (American Heart Association Database, Creighton University Tachyarrhythmia Database, MIT-BIH Arrhythmia Database, MIT-BIH Malignant Ventricular Arrhythmia Database) with a duration of 30 mins or 8 mins (Creighton University Tachyarrhythmia Database) per recording were processed. Shockable conditions are identified as ventricular tachycardia, ventricular fibrillation, and ventricular flutter. The 1 channel ECG recordings (modified limb lead II) were normalized to 250 Hz sampling frequency, high-pass filtered (1 Hz cutoff and 0.85 filter steepness), second order Butterworth low-pass filtered (30 Hz cutoff), and notch filtered at 50 Hz. Consistent wavelet transformation with 5 octaves, 20 voices per octave, and a time bandwidth product parameter of 50 was applied to generate greyscale spectrogram representations of the ECG data (pixel value range from 0 to 255). The recordings were segmented into 3 s segments. Data augmentation around the borders of shockable episodes and along shockable episodes was carried out to create balanced datasets consisting of 60340 samples. 45% of samples in the balanced dataset contain shockable rhythms with more than 60% temporal prevalence within each sample. Conventional convolutional neural networks and residual neural networks with varying architectures and hyperparameter settings were trained and evaluated on balanced datasets (train/val/test: 70/15/15). The approach focused on examining a broader range of parameter settings and model architectures rather than optimizing a specific configuration. The best performing model was evaluated in a 5-fold cross-validation. Exemplarily, a leave-one-subject-out cross-validation was deployed with 3 randomly chosen recordings, with the constraints that each subject must come from a different database and contain a different shockable condition.

**Results and Conclusion**

The best performing model was a residual neural network with 96 residual blocks. The 5-fold cross-validation results on average in an accuracy of 0.987, a sensitivity of 0.992 on shockable rhythms, and a specificity of 0.984 for non-shockable rhythms on the test sets. The ROC AUC score is 0.998 on average. The 3-fold leave-one-subject-out cross-validation reaches on average an accuracy of 0.984, a sensitivity of 0.984, and a specificity of 0.980. The ROC AUC score reaches 0.997 on average. The analysis of misclassified segments reveals that the classifier performs less accurately on border segments containing a shockable and at least one non-shockable rhythm. While the test set contains 4.73% border segments, the set of misclassified samples includes 11.29% border segments. The label distributions of the test set and the set of misclassified samples show that segments annotated as “not defined” (ND) and “ventricular fibrillation or flutter” (VF-VFL) are significantly more prevalent in the set of misclassified samples. Histogram analysis, referring to the mean pixel intensity of the spectrograms, indicates that the classifier works less accurately on spectrograms with mean pixel values below 2 (practically flatline signals or signals with very small amplitude).

The results indicate that it is possible to improve the analysis of ECG data by deploying automated feature detection combined with artificial neural networks. The methods presented in this work are not restricted to the detection of shockable cardiac arrhythmias, they likewise emphasize the potential of machine learning in the domain of biosignal analysis and correlated medical data. In the next step, the approach needs to be verified on a broader database. The technology can even help create more comprehensive databases of clinical ECG data by supporting automated annotation.

Keywords: automated external defibrillator, AED, cardiac arrhythmia, cardiac arrest, convolutional neural network, CNN, defibrillation, ECG data analysis, feature extraction, deep learning, residual neural network, ResNet, shockable, shockable heart rhythms, spectrogram, wavelet transform

# PREFACE

This work is inspired by the aspiration to benefit society by connecting medical research and the fascination for the capability of machines to reveal complex patterns in large amounts of data in the blink of an eye. Like always, we are only at the beginning. The more experience we gain, the more there is to learn. With research and medical needs evolves technology. Perhaps it is our purpose or even obligation to discover the potentials of technologies and to ensure that they’re used reasonably, in a way that lets us walk towards altruism harmonizing individual needs with the greater good. I want to thank Tampere University and the Finnish society for leading and encouraging me to walk on this path. Special thanks go to my mentors Milla Juutinen, Antti Vehkaoja, and Christian Haverkamp for always supporting me and encounter my flaws with empathy and tolerance. I experience them as people who believe in the potential of their students and colleagues, and who are passionate about unfolding these potentials. Thanks to Niku Oksala for providing financial support and  arranging required licences to medical databases! Thank you, Thomas Otto, for your support and sparking the idea in me to go to Tampere! I want to say thank you to Sabrina Schmid, to my family, and to all the soulmates I was blessed to meet so far, without whom I couldn’t be writing this.

Tampere, 24 March 2023

Sebastian Siegel

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| Adam | Adaptive moment estimation |
| AED | Automated external defibrillator |
| AHADB | American Heart Association Database |
| AUC | Area under the receiver operating characteristic curve |
| AV node | Atrioventricular node |
| CNN | Convolutional neural network |
| CPU | Central processing unit |
| CUDB | Creighton University Tachyarrhythmia Database |
| ECG | Electrocardiogram |
| FN | False Negatives |
| FP | False Positives |
| GPU | Graphics processing unit |
| ICD | Implantable cardioverter defibrillator |
| LBB | Left bundle branch |
| MITDB | MIT-BIH Arrhythmia Database |
| MLII | Modified limb lead II |
| NShr | non-shockable |
| PNG | Portable Network Graphics |
| RBB | Right bundle branch |
| ReLU | Rectified Linear Unit function |
| ResNet | Residual neural network |
| RMSProp | Root mean squared propagation |
| ROC | receiver operating characteristic |
| SA node | Sinoatrial node |
| Shr | Shockable |
| SVM | Support vector machine |
| TCI | threshold crossing intervals |
| TN | True Negatives |
| TP | True Positives |
| VF | Ventricular Fibrillation |
| VFDB | MIT-BIH Malignant Ventricular Arrhythmia Database |
| VFL | Ventricular Flutter |
| VT | Ventricular Tachycardia |
| WFDB | Wave Form Data Base |

# 1. INTRODUCTION

Sudden cardiac arrest, the abrupt loss of heart function, accounts for 5-15% of the total death rate in developed countries [1]. In a public situation where a person is found unconscious or unresponsive, it might be likely that this person suffers from a shockable cardiac condition that requires the immediate deployment of defibrillation. On the other hand, if strong electrical shocks to the heart are applied when medically not indicated, the patient can suffer severe complications and even die. If the same situation happens in a hospital, medical expertise and medical devices are present to assess the problem and provide proper treatment. Therefore, survival chances are significantly higher for cardiac arrests in a in hospital situation (see Section 2.2.1). In public places, it is reasonable to develop and deploy devices that can substitute for medical expertise and automatically apply the right treatment. One such device is the automated external defibrillator (AED). For people with chronic heart diseases, it might be indicated to use an implantable version, the implantable cardioverter defibrillator (ICD). Both concepts are explained in more detail in Sections 2.2.2 and 2.2.3. AED and ICD are both based on electrocardiogram (ECG) analysis.

For decades, research has been carried out in ECG data analysis to manually find distinct features and characteristic patterns that are associated with certain medical or, more specifically, cardiac conditions. Medical devices such as the AED and ICD are derived from this research. The feature extraction is correlated with the assessment of shockable cardiac conditions. The features are logically linked in a decisive algorithm that determines if defibrillation should be applied or not. Commercially available AEDs approximately reach sensitivities of 90% for shockable conditions and specificities of 95% for non-shockable conditions. Consequently, 1 out of 10 shockable conditions is not recognized by the AED and remains untreated [2] [p. 23] [3–5].

Manual feature extraction has some weak spots. It is often tedious, and it might be too insensitive to catch the relevant patterns in detail. Since 2012, with the implementation of GPU support, Convolutional Neural Networks (CNNs) have been on the rise and have contributed heavily to the approach of automated feature extraction. In many domains, automated feature extraction with deep learning methods tends to outperform handmade feature extraction and has brought machine learning applications to the next level. Algorithms that learn relevant features from scratch don’t require to be configured and trained by domain-specific experts. Instead, similar CNN architectures can be utilized to solve a variety of problems across different domains. This advantage is boosting machine learning applications in almost all parts of society.

This thesis is dedicated to examining automated feature extraction and the classification of shockable heart rhythms in the context of ECG analysis. The main question raised in this work is whether and how CNNs can be utilized to solve the classification problem of shockable cardiac conditions without the implementation of expert driven manual feature extraction. This is realized by converting 1 channel ECG recordings into spectrogram segments with a length of 3s. These ECG image representations are calculated by applying wavelet transformation. The annotated ECG data are derived from cardiac arrhythmia databases, namely the MIT-BIH Arrhythmia database (MITDB), the Creighton University Ventricular Tachycardia database (CUDB), the MIT-BIH Ventricular Arrhythmia database (VFDB), and the American Heart Association database (AHADB) (see Section 3.1). Different CNN architectures, such as conventional CNNs and residual neural networks with varying configurations, are trained and evaluated with annotated ECG spectrograms.

The thesis starts by providing basic theoretical background on cardiac physiology, ECG analysis, and shockable cardiac arrhythmia (see Section 2.1). Then the defibrillation technology and its relevance are discussed in Section 2.2, followed by an overview of algorithmic strategies to assess shockable conditions. This begins with algorithms implemented in commercially available defibrillation devices. Further, research into solving the problem by implementing machine learning approaches is discussed. Hybrids combining manual feature selection with machine learning classifiers are followed by 1D and 2D CNNs deploying automated feature selection (see Section 2.3). To round up the theoretical background, CNNs are explained in more detail: How do CNNs process data? How do they extract features? What are their components? How are these components structured? And how do CNNs learn (see Section 2.4)?

The work continues by discussing the applied methods in Chapter 3. The ECG source data are described, and the pre-processing steps applied to the data are discussed, including querying, filtering, wavelet transformation, and annotation. The pre-processing is concluded by implementing augmentation strategies and generating balanced datasets, ready to be fed into CNN models. Further, the applied CNN models as well as their configuration options and hyperparameter settings are introduced. Finally, the applied evaluation metrics are described.

Chapter 4 summarizes the results obtained from the evaluation of the applied CNN models. The model configurations are not a comprehensive approach to maximizing the detection performance of shockable heart rhythms. Instead of examining narrow adjustments of model parameters, the approach rather focuses on deploying 2 different concepts of CNN architecture: conventional CNNs and residual neural networks. Most of the time, the individual models differ by implementing and adjusting several structural changes and hyperparameters at once. Therefore, a wider range of settings and their impact on detection performance could be captured. At first, conventional CNNs and residual neural networks are trained and evaluated on balanced datasets derived from the mentioned databases. Further, the best performing model is evaluated in a 5-fold cross-validation. Then, again on the best-performing model, a leave-one-subject-out cross-validation is exemplarily carried out with 3 entire recordings from different patients with a focus on different shockable conditions. Eventually, a closer look at misclassified data is taken to draw some assumptions about why the model failed on those samples.

The thesis concludes by emphasizing the medical relevance of accurate detection algorithms for cardiac arrhythmias. Weaknesses, especially regarding the available annotated ECG data sources, are discussed, and an outlook is drawn for a potential solution and the general potential of automated feature detection in ECG data analysis.

# 2. THEORETICAL BACKGROUND

## 2.1 Physiology of the heart and Electrocardiogram (ECG)

The following Section gives a brief overview of the physiological phenomena that are related to shockable conditions of the heart. Shockable conditions are a form of cardiac arrhythmia that, without intervention, are likely to lead to cardiac arrest and the death of the patient. Shockable conditions indicate intervention by defibrillation, which utilizes electrical shocks to depolarize the heart muscle. Defibrillation is an attempt to terminate the shockable arrhythmia and restore coordinated contractions of the heart.

Further, this Section provides basic information about ECG signals while focusing on parameters that are expedient for the interpretation of ECG signals.

### 2.1.1 Physiology and anatomic aspects of the heart

In many cases, evolution utilized diffusion to transport substances within organisms. Diffusion works sufficiently on the cellular level. But the larger and more complex the organism, the higher the demand for substances to be transported over long distances. Vital substances like oxygen and nutrients require an efficient delivery system. At its center is the heart, an engine that generates the flow in every blood vessel and ensures the delivery of substances and the removal of waste at the right velocity and at the right time to supply every other organ in the system.

The heart is part of the closed circulatory system in the human body. As a muscular pump, it powers the circulation of blood by elevating its pressure in the arteries. The elevation is produced by contraction of the heart muscle, which is induced by an orchestrated rise of action potential throughout the organ. The human heart consists of four chambers: two atria, which receive blood returning to the heart, and two ventricles, which pump the blood out of the heart [6] [pp. 272-277]. The heart’s size is comparable to a fist, and it is located behind and slightly to the left of the breastbone. Figure 1 gives an overview of the components of the heart and the direction of blood flow.

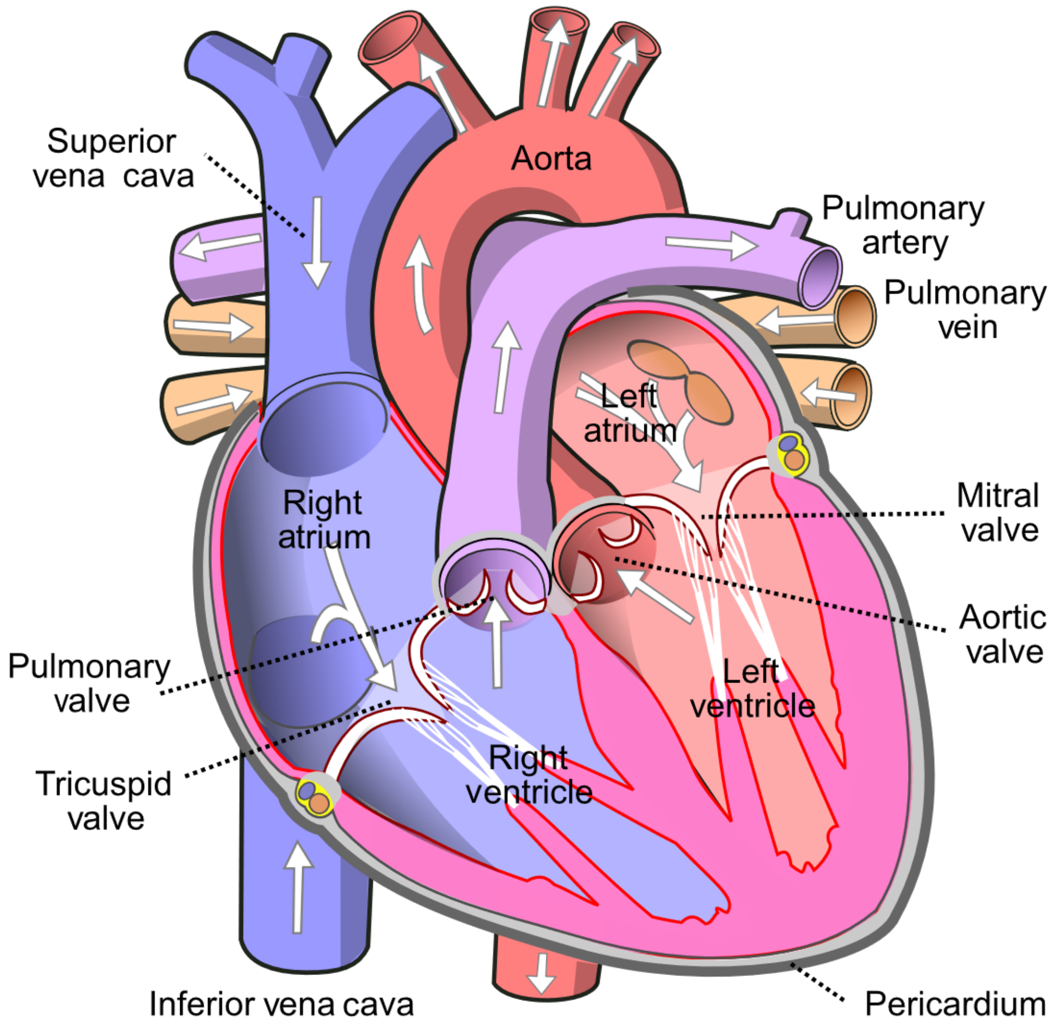


***Figure 1:** Anatomy of the heart **[7]***

The contraction of cardiac muscle cells is driven by the action potential. It is caused by the inflow of sodium ions into the myocytes. The change in electric gradient in the cell causes depolarization with an amplitude of around 100 mV. The repolarization is initiated by the outflow of potassium ions. The action impulse lasts about 300 ms and causes contraction of the cell. The activation in cardiac muscle tissue can propagate in any direction from cell to cell and forms orchestrated wavefronts of the cardiac electrical activity [8] [p. 185].

**Ventricles**

The heart contains two large chambers at its bottom, the ventricles. The ventricles are responsible for collecting blood from the atrium and expelling it towards the peripheral vascular system. The right ventricle pumps blood, which is low in oxygen, through the pulmonary arteries towards the lungs. The left ventricle pumps oxygen-rich blood through the aorta towards the systemic capillaries in order to provide the body tissues with oxygen.

**The electrical conduction system of the heart**

The electrical conduction system, as shown in Figure 2, consists of specialized cells that are responsible for the transmission of electrical impulses throughout the heart tissue as well as for pacemaker functions. The pacemaker sets the rate at which the heart's pumping cycle is repeated. Each component of the conduction system can set the pace, though there is a hierarchy following the cells with the fastest pace, the sinoatrial node (SA node), down to the cells with the slowest pace (Purkinje cells), which is likewise the direction of the electrical impulse throughout the heart tissue.

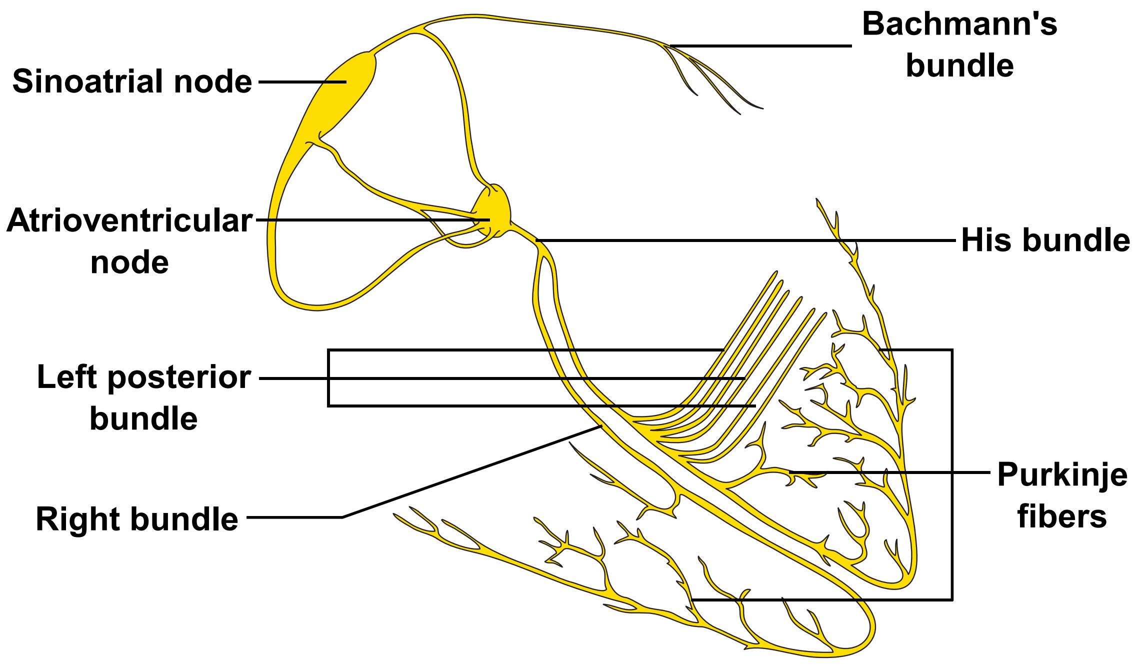


***Figure 2:** The electrical conduction system of the heart **[9]***

In case the SA node, the main pacemaker, fails to fire, there are several backups in place to ensure contraction of the heart muscle. The electrical impulse starts at the SA node, which is located in the right atrium, and it propagates through the atrioventricular node (AV node), which slows down the impulse conduction from the atrium to the ventricles to ensure a separate atrial contraction. Eventually, the impulse is conducted to the left and right bundle branches (LBB and RBB), which innervate the ventricles [10] [pp. 43-52].

### 2.1.2 Electrocardiogram (ECG)

The electrical activity of the heart can be detected by electrodes applied to the body surface. The change in the electric potential can be measured as a change in voltage between two electrodes. The measurement of cardioelectrical activity is called electrocardiography. The most common approach to measure the heart's electrical activity is the standard 12-lead ECG, which allows a time dependent representation of the strength and direction of electric impulses emerging from the heart. The 12-lead ECG consists of 3 bipolar limb leads (I, II, III), 3 augmented unipolar leads (aVR, aVL, aVF), and 6 unipolar chest leads (V1-V6) [10] [pp. 71-72]. As the leads, or electrodes, pick up the electrical activity of the impulse vectors, each of the leads represents the heart's activity from a different angle. Therefore, the more leads used, the higher the resolution of the representation, and the better certain aspects and processes can be localized.

In practice, shockable conditions of the heart (2.1.3) can be determined with a single-lead ECG, where only 2 electrodes are placed on the body to assess the ECG signal [11]. The 2 electrodes are usually placed in the anterior-lateral scheme, which can be used for ECG heart rhythm analysis as well as for defibrillation. The anterior electrode is placed below the right calvarium. The lateral electrode is placed on the left side of the patient below and towards the left end of the chest muscle [12].

The regular heartbeat can be described as a cycle of coordinated, consecutive electrical events, which are represented as characteristic waveforms in the ECG signal. These characteristic waveforms, which are repetitive and synchronized with each heartbeat, are defined as the ECG complex. The ECG complex can be segmented into single and several consecutive waves, that represent basic components of the heart rhythm (Figure 3).

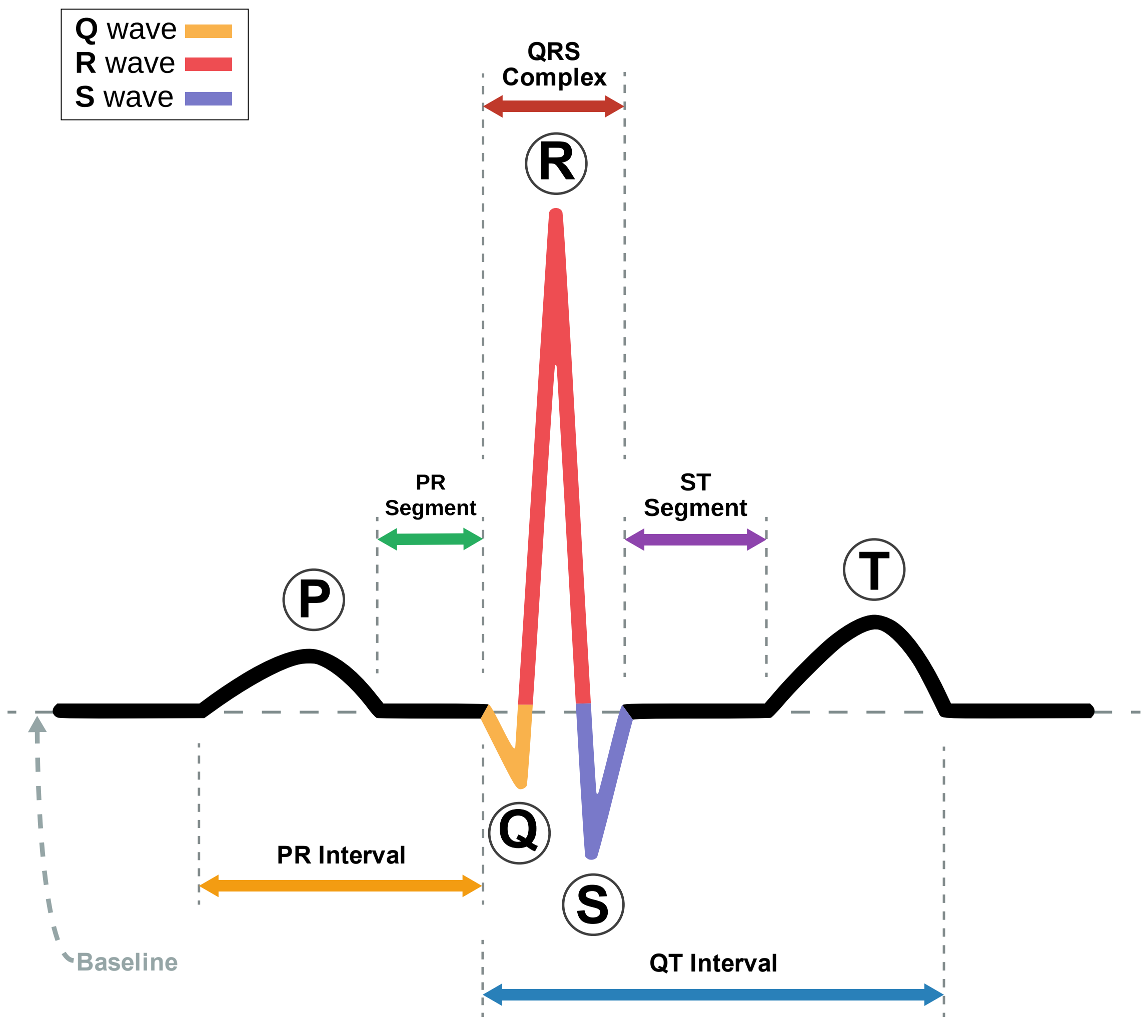


***Figure 3:** Basic components of the ECG complex **[13]***

The illustrated segment between the T and P waves in Figure 3 indicates the baseline on which the characteristic waves are deflected.

The P wave represents the depolarization of both atria, which provokes the atrial contraction or atrial systole. The normal duration of the P wave lies between 0.08 and 0.11 s.

The PR wave runs along the baseline. If the PR wave is shifted below normal, it can indicate pathological conditions. Within the PR wave, the electrical depolarization wave is transmitted through the AV node down to the bundle branches.

The PR interval consists of the P wave and the PR wave. PR intervals shorter than 0.11 s are considered shortened, while PR intervals longer than 0.20 s are first-degree AV blocks. AV blocks are a disease in which the conduction of electrical impulses from the SA node towards the ventricles is slowed down.

The QRS complex describes the ventricular depolarization with a normal duration of 0.06 to 0.11 s. If the Q wave is longer than 0.03 s or if its height is greater than one-third the height of the R wave, it indicates myocardial infarction.

The ST wave represents an electrically neutral period between ventricular depolarization and repolarization, during which blood is pushed out of the ventricles.

The T wave stands for ventricular repolarization [10] [pp. 100-112].

### 2.1.3 Shockable Cardiac Arrhythmia

Arrhythmias of the heart associated with dysfunctional ventricles are related to conditions that are treated by electrical shocks induced in the myocardium (defibrillation). There are 3 cardiac arrhythmias to be classified as shockable conditions: ventricular tachycardia (VT), ventricular fibrillation (VF), and ventricular flutter (VFL) [14,15].

**Ventricular Tachycardia**

Ventricular tachycardia (VT) is a medical condition that is associated with a fast, regular beating of the ventricles, which is usually dissociated from an underlying atrial rate [10] [p.159]. VT is defined as four or more ventricular ectopic beats in rapid succession (Figure 4) [14] [p. 77]. The ventricular ectopic beats occur at a rate of at least 100 bpm, while the QRS complex lasts more than 0.12 s. VT is commonly caused by myocardial infarction and can occur with or without the presence of heart disease, though VT is more likely to occur when associated with chronic coronary artery disease, congenital heart disease, and cardiomyopathy [15] [p. 280]. The two main types of VT are monomorphic and polymorphic. Monomorphic VT is characterized by a rapid succession of ventricular ectopic beats with rates from 120 to 250 bpm and similar occurrences. Polymorphic VT is associated with repeated, progressive changes in the direction and amplitude of ventricular complexes and prolonged QT intervals [14] [p. 77].

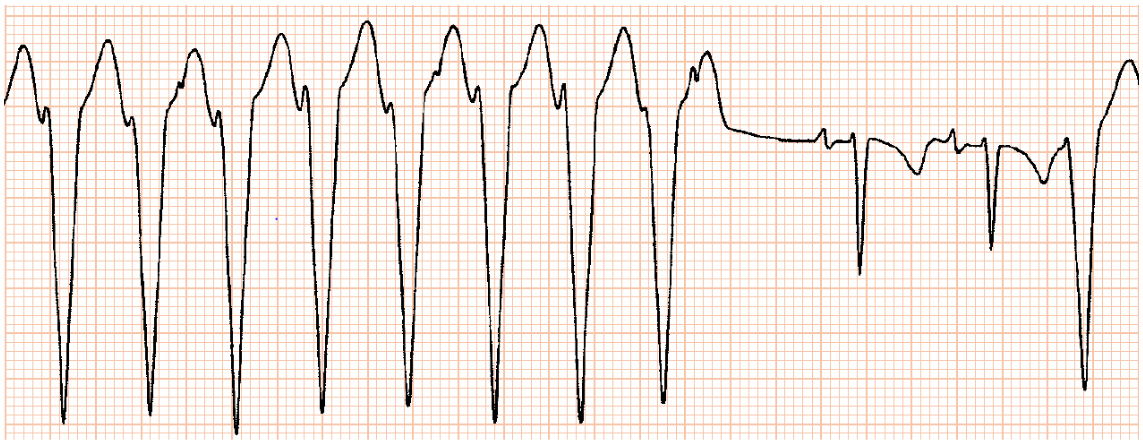


***Figure 4:** Monomorphic VT (sequence of ectopic beats) followed by two sinus beats and a single ventricular ectopic beat **[14]** [p. 78]*

In consequence, the heart loses its ability to pump blood sufficiently into the cardiovascular system and fails to provide the body with oxygenated blood. When VT lasts longer than a few seconds, it causes falling blood pressure and shortness of breath, which can lead to dizziness and fainting. VT can further lead to severe hypotension, ventricular fibrillation (VF), and cardiac arrest [16]. In general, severe symptoms are more likely with longer episodes of VT (> 30 s) and faster rates over 150 bpm [15] [pp. 280-281].

In acute situations, VT is treated by defibrillation. In the long term, VT is treated by surgical removal of the heart cells that cause VT and by medication [16]. VT at rates between 130-170 bpm can often be terminated by antitachycardia pacing. Antitachycardia pacing refers to painless shocking techniques applied by implantable cardioverter defibrillators (ICDs), intending to stimulate the heart's pace [17]. Fast VT rates are likely to cause a collapse. In cases where the antitachycardia pacing is failing after several attempts, the condition is treated with defibrillating shocks. If the VT doesn't show hemodynamic impairment, the initial shocking energy is between 50-100 J. In case of intolerable or pulseless VT, the initial shock is given at 200 J. If necessary, repeating shocks are initiated at increasing energy steps of 300 J and 360 J [18] (pp. 236). Energy levels for internal shocks (from implantable devices) are approximately one tenth of energy levels for external defibrillation [19].

### Ventricular Fibrillation

Ventricular fibrillation (VF) is characterized by uncoordinated quivering of the ventricles due to rapid, irregular electrical impulses ranging between 350-450 bpm. VF is the most prevalent arrhythmia related to cardiac arrest (a pulseless situation). VF causes circulatory arrest, and unconsciousness develops within 10-20 s [14] [p. 106].

If VF continues for more than 1-2 minutes, it often leads to death. The ECG signal is characterized by a wandering baseline and strongly varying QRS complexes in morphology and height (Figure 5). The QRS complex appears indistinguishable from ST segments, and P and T waves are not recognizable. VF is occasionally difficult to separate from polymorphic VT [15] [p. 306]. VF is commonly initiated by ventricular ectopic beats and can arise from ventricular tachycardia.

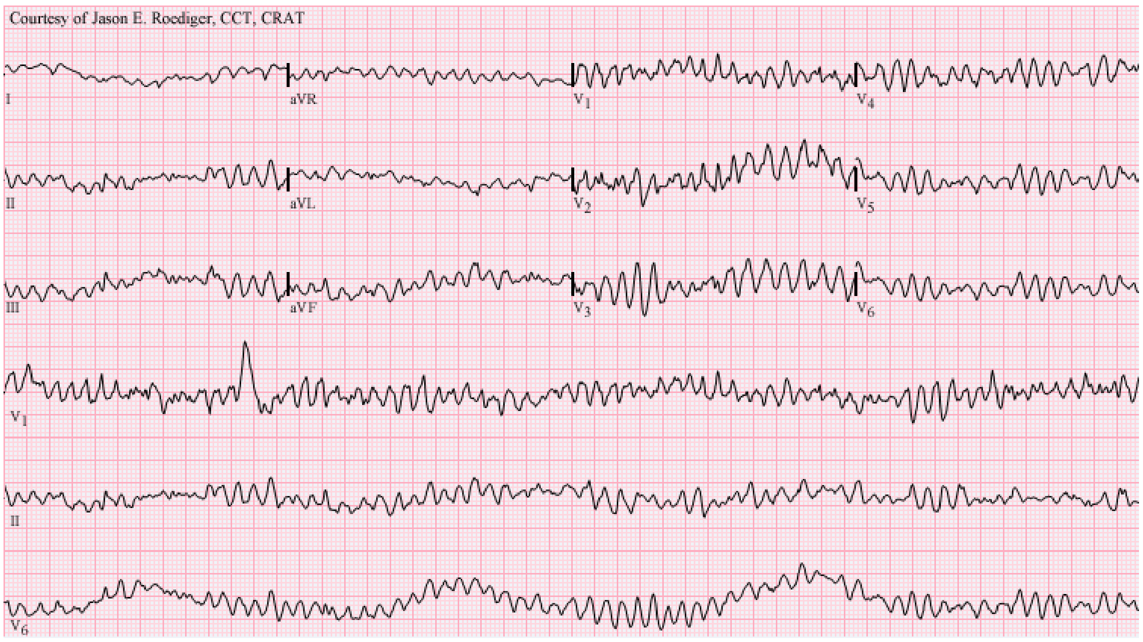


***Figure 5:*** *12-lead ECG recording of ventricular fibrillation* ***[20]***

Ventricular fibrillation is treated by prompt defibrillation with high energy shocks, which successfully defibrillate 90% of cases. Initial shocks have an energy level between 150-200 J, if unsuccessful, further shocks between 200-360 J are induced (Bennett 2012) (pp. 106, 144, 173). The energies mentioned refer to an external defibrillator. Energy levels for internal shocks are approximately one tenth of energy levels for external defibrillation [19].

**Ventricular Flutter**

Ventricular Flutter (VFL) is a rapid form of VT with rates over 280 bpm [18] [p. 236]. The ECG signal is characterized by a regular, almost sinusoidal pattern without distinguishable components, as shown in Figure 6 [10] [p. 163].

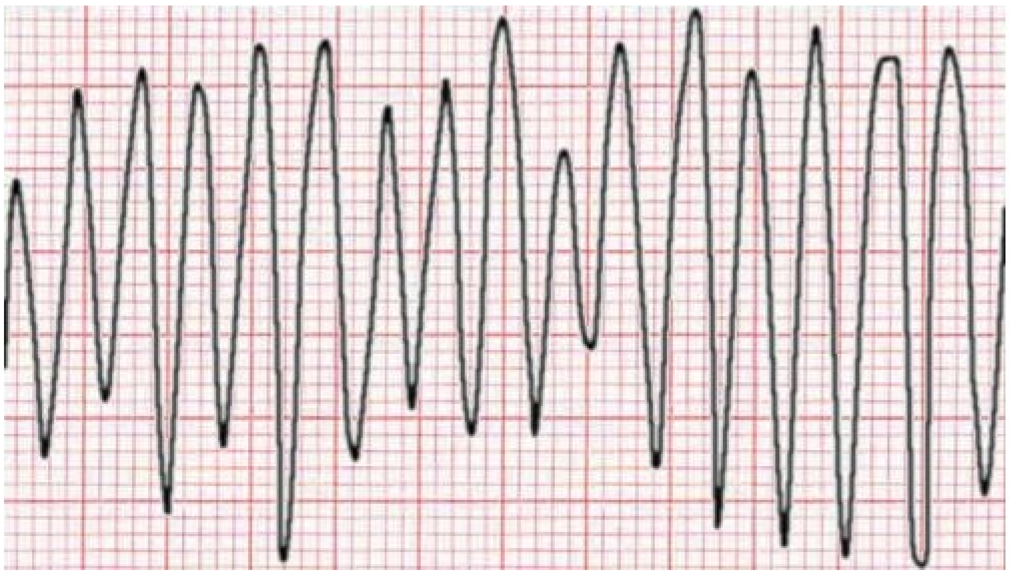


***Figure 6:*** *ECG lead of Ventricular Flutter* ***[21]***

QRS complexes and T waves become indistinguishable [15] [p. 306]. VFL is treated with emergent electrical shocks as described earlier for VT.

## 2.2 Defibrillation

Defibrillation is the application of electrical shocks into the heart with the purpose of terminating a shockable, non-perfusing cardiac arrhythmia (Section 2.1.3). The electrical shock is induced to depolarize the myocardium and restore coordinated contractions [22] [Ch 23]. The depolarization of the heart muscle enables the SA node to reimplement its natural pace throughout the electrical conduction system of the heart. In defibrillation, the electrical shock to the heart is delivered at a random moment, whereas in cardioversion, electrical impulses are applied at specific moments with regard to the cardiac cycle [23].

At the end of the 19th century, researchers found that moderate electrical shocks could provoke ventricular fibrillation in dogs, while strong electrical shocks could terminate the arrhythmia and restore the normal heart beat [24]. In 1947, the first successful human defibrillation was performed on a patient suffering from VF [25]. Today's standard are automated external defibrillators (AEDs) and implantable defibrillators that have the ability to automatically assess if a shockable condition is present and therefore electrical shocking is required. Next to in-hospital use, the automated detection of shockable rhythms enabled the defibrillation technology to be used in out-of-hospital care and therefore to save more lives. Manual defibrillation is still performed in the intensive care unit since the patients are constantly monitored by medical professionals and the decision if and how to perform defibrillation remains the responsibility of the medical experts. Though, computer-based alarm systems are used to inform the personnel about critical situations.

### 2.2.1 Relevance of defibrillation technology

Sudden cardiac arrest is the abrupt loss of heart function, resulting in the cessation of blood flow. In developed countries, sudden cardiac arrest accounts for 5-15% of the total death rate [1]. In Europe, approximately 38 out of every 100000 persons per year suffer from an out-of-hospital sudden cardiac arrest, which is treated by emergency medical services. The survival rate is 10.7% [26]. In the USA, 55 out of every 100000 persons per year are affected, and the survival rate lays at 8% [27]. Approximately 80% of cardiac arrests are out-of-hospital incidents [28].

Around 70% of out-of-hospital cardiac arrests and 80% of in-hospital cardiac arrests are caused by conditions with pulseless electrical activity and asystole, which are typically not treatable by electrical shock [29,30]. For shockable conditions that occur outside of hospitals and are treated by emergency medical services, the survival rate lays between 17 and 21% [26]. In contrast, for shockable conditions that occur and are treated in hospitals, the survival rate increases to 45-60% [29–31]. The main reason for this deviation is based on immediate access to appropriate treatment in a clinical environment. When the heart loses its ability to provide perfusion of the body, the risk of irreversible organ damage and death rises quickly without treatment. The survival rate decreases approximately by 5-10% with each minute from collapse to defibrillation [32]. In the case of a shockable condition, early defibrillation is a key measure to enhance the chance of survival [29]. On the other hand, the inappropriate use of defibrillation can cause severe damage, even leading to the death of the patient and the person who is applying the measure [11]. Manual defibrillators require the expertise of healthcare professionals. With automated detection of shockable arrhythmias, defibrillation becomes a medical aid technology that is not restricted to being used by medical professionals only. Consequently, automatic defibrillation technology became beneficial for public and outpatient usage. Automated external defibrillators (AEDs) and implantable cardioverter defibrillators ease and save the lives of many people [2] [p. 7]. Observational studies show that early out-of-hospital defibrillation of shockable conditions with AEDs, which is applied by first aid responders (mainly people with no or little medical training), reaches a median survival rate of 53.0%. The survival rate (generated from the same studies) for people treated by emergency medical services lies at 28.6% [33].

### 2.2.2 Automated External Defibrillator (AED)

AEDs aim to automatically assess shockable arrhythmias of the heart accurately and apply the appropriate therapy by inducing electrical shocks through electrodes placed on the patient's body. The electrodes are usually attached following the anterior-lateral scheme as illustrated in Figure 7 (Section 2.1.2). AEDs are utilized for public access defibrillation. Therefor, it is required that the AED perform robustly and accurately even under unfavorable conditions. Further, the AED needs to be widely accessible in public places, safe, and easy to use to enable quick and correct usage by untrained persons without medical expertise. When the electrodes are attached, the AED determines if defibrillation is indicated. If so, the device advises the shock, and the rescuer is informed to stay clear of the patient. When the capacitors are charged, the rescuer is informed to press the shock button on the device, and the shock is delivered [2].

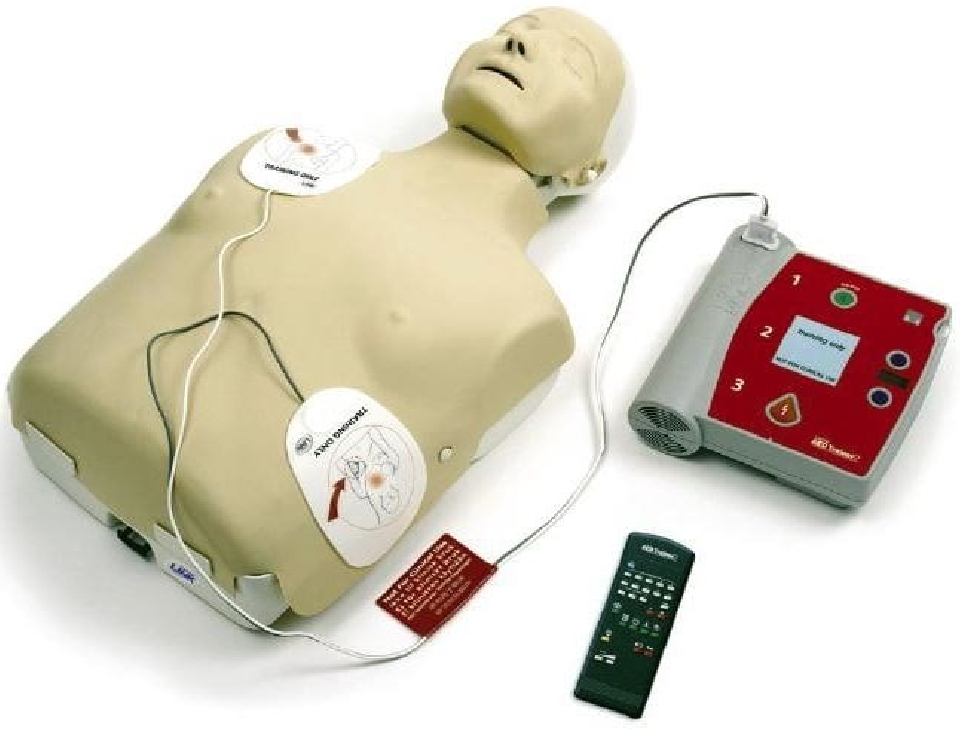


***Figure 7:** Automatic external defibrillator **[34]***

Only arrhythmias such as ventricular tachycardia, ventricular fibrillation, and ventricular flutter can be treated by defibrillation. Strong electrical shocks in the presence of another rhythmical condition of the heart can cause serious complications or death to the patient. It is crucial that the AED classifies the patient's condition accurately as shockable or non-shockable. A shockable rhythm refers to a lethal condition that terminates in the patient's death unless defibrillation is applied quickly [11].

The accuracy of commercially available AEDs focuses on maximizing the specificity for shockable rhythms at the cost of sensitivity for detecting shockable rhythms. This is due to potential legal problems in certain countries in which the person providing first aid is held responsible for harm caused to the patient by a falsely applied electrical shock [3]. According to the IEC 60601-2-4 international standard, the requirements of the American Heart Association, and several performance studies, commercially available AEDs reach a sensitivity of approximately 90% and a specificity of around 95% [2,4,5,35]. It means that around 1 out of 10 shockable conditions is not detected by the device, and if a shockable condition is detected, the AED is accurate in 95% of the cases. The actual performance values might vary slightly depending on the study, the kind of devices, and the test data. It might be beneficial for the comparability of device performances if all devices are tested on a common and officially verified ECG database [11].

### 2.2.3 Implantable Cardioverter Defibrillator (ICD)

Implantable cardioverter defibrillators (ICDs) are subcutaneously implanted devices with leads either positioned within the heart or subcutaneously. The main purpose of ICDs is to detect and treat shockable arrhythmias. Additional to defibrillation, ICDs can deliver electrical impulses to the heart at specific moments of the cardiac cycle (cardioversion) and more recently, pacing of the heart [36]. The implantation of ICDs is indicated as a form of secondary prevention when patients have already suffered from life-threatening cardiac events. Another indication is primary prevention, when patients haven’t had a life threatening cardiac event but are at high risk for one to occur [35].

In contrast to AEDs, the algorithms of ICDs are designed to maximize sensitivity in the detection of shockable rhythms at the cost of specificity. This is due to the assumption that missing shockable conditions results in a higher fatality risk than falsely delivered shocks from ICDs. Some devices perform at up to 99% sensitivity, while specificity is decreased below 90% [37]. Occasionally, this results in the delivery of shocks without the presence of a shockable condition. Studies show that wrongfully detected ventricular arrhythmias lead to inappropriate shocks in approximately 20% of patients with ICDs. With common detection algorithms, wrongfully delivered shocks are mainly caused by false discrimination of supraventricular tachycardia and oversensing of other physiological signals and interferences [38] [p. 66]. Falsely delivered shocks diminish the life quality of the patient and increase mortality [39]. International efforts have been put forth by the industry, the research community, and governments to address the problem and work on solutions that improve the performance of detection models for shockable conditions [38] [p. 87].

### 2.2.4 Detection algorithms in commercial defibrillation devices

The most established methods for detecting shockable cardiac rhythms by defibrillation devices are based on count algorithms, which analyze the ECG data that is collected from the patient. The detection algorithms utilize the heart rate to roughly classify the present heart rhythm. Further techniques that analyze the ECG morphology and the behavior of the heart cycle intervals are applied to make the classification more distinct, as, for example, it is crucial to correctly distinguish between VT and supraventricular tachycardia [40]. In contrast to VT, supraventricular tachycardia is not a shockable condition and requires a different treatment. False classification of supraventricular tachycardia still remains a weakness of defibrillation devices, as described in Section 2.2.3.

In 1990, the threshold crossing intervals (TCI) algorithm introduced a time-domain method for the detection of VF and VT and laid the groundwork for many subsequent developments and improvements [40,41]. Another relevant time-domain procedure for VF detection is the auto-correlation function [42]. The VF-filter by Kuo and Dillman and the spectrum analysis methods are popular VF detectors in the frequency domain [43].

**Threshold crossing interval algorithm (TCI)** [44]

Though different approaches are mentioned, the principle of heartrate-based detection algorithms for shockable rhythms is explained with the TCI algorithm by Nitish Thakor and Yi-Sheng Zhu as follows [44]: The pre-processing part starts with digitizing the ECG signal and filtering to clean up the signal. A low-pass filter removes muscle noise, a high-pass filter removes baseline drift, and a notch filter is applied to suppress power line interference. Then the signal is segmented into 1 s segments and transformed into a binary signal. A threshold set at 20% of the peak value in each segment serves as the boundary for the binary signal generation, everything above the threshold is considered 1 and everything below as 0. Further, for each segment, the average interval between threshold crossings is calculated as described in Figure 8:

$$\mathrm{TCI} = \frac{1000}{(N-1) + \frac{t_2}{t_1 + t_2} + \frac{t_3}{t_3 + t_4}} \; (\mathrm{ms})$$

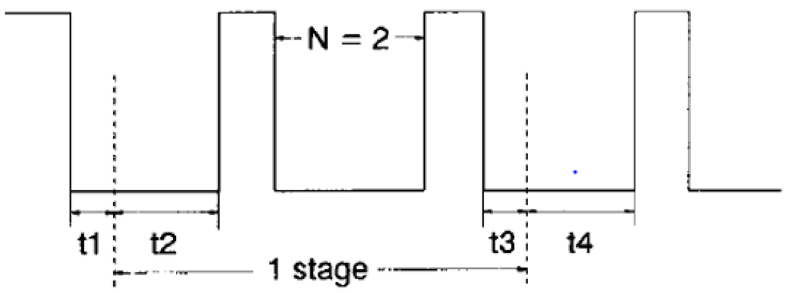


***Figure 8:*** *Threshold crossing interval (TCI)* ***[44]***

In order to say if the calculated TCI of a segment indicates a shockable condition, a probability distribution with labeled ECG data was created (Figure 9). In the original TCI research paper, 170 clinical recordings of VF and VT were used.

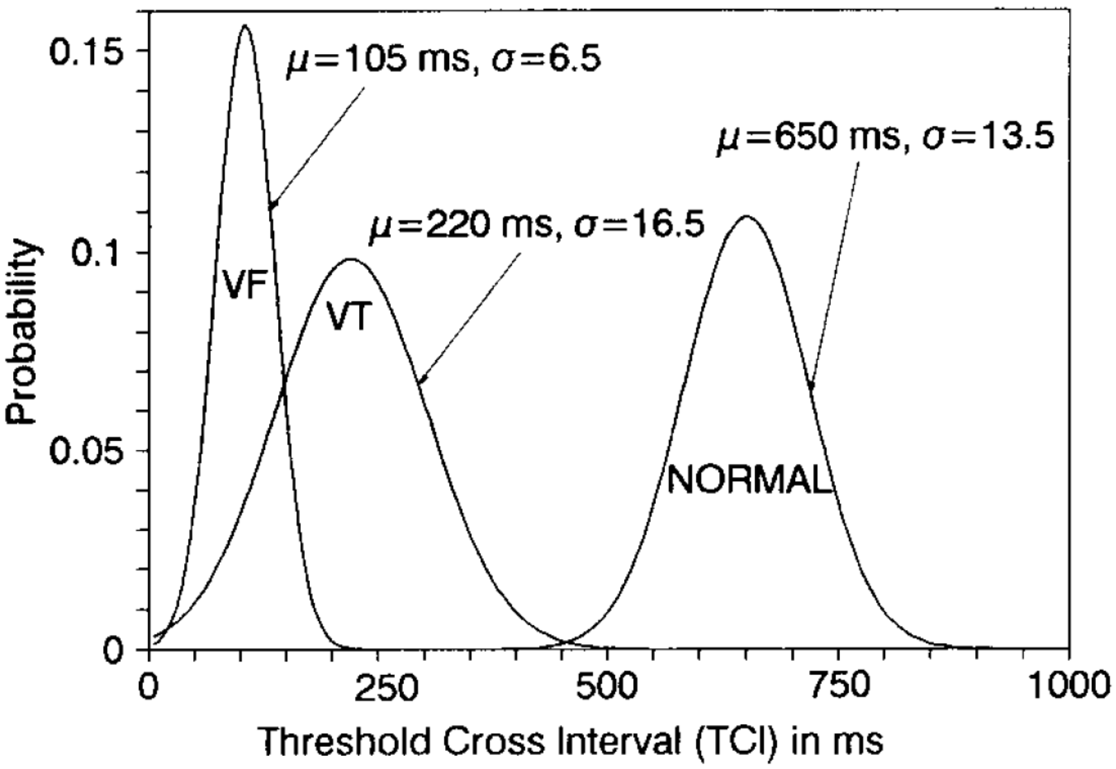


***Figure 9:*** *Probability distribution of TCIs* ***[44]***

As shown in Figure 9, the TCI distributions for normal sine rhythm and shockable rhythms don’t overlap, which means a decision can be easily obtained from the TCI value of a given segment. Since the distributions for VF and VT overlap, further testing is necessary to discriminate between the two. The paper suggests a sequential probability ratio test in which no decision is made if the sample TCI lies in the overlapping zone between 2 preset probability thresholds. The test is repeated with consecutive TCI samples until the resulting probability is no longer within the probability threshold window and the segment can be assigned to either VF or VT.

**Philips SMART Analysis AED algorithm** [45]

Since the publication of the TCI algorithm, detection methods for shockable rhythms have been constantly developed. Philips SMART analysis AED algorithm gives an example of present decision-making in defibrillation devices. The device-algorithm systems by Philips meet the requirements of IEC 60601-2-4, which requires a sensitivity of >90% and a specificity of >95% in the detection of shockable rhythms [46]. Before processing the ECG signal further, the patient's impedance is measured to ensure that the electrode pads are properly attached and the ECG signal can be read efficiently. Likewise, transthoracic impedance, common mode current, and electrical potentials are sensed by the electrode pads. If these signals correlate with the ECG signal and are disturbingly high in their amplitudes, the algorithm classifies the artifact and provides appropriate measures. For example, if pacemaker artifacts are classified, the algorithm removes the pacemaker spikes from the signal and proceeds.

The rhythm detection part of the algorithm is applied on 4.5 s segments and considers 4 parameters in the decision-making for shocking: the heart rate, the shape of the QRS complex, the regularity of the waveform pattern, and the amplitudes of the signal. The higher the heart rate the stronger the vote of the heart rate parameter for shocking. But regardless of the heartrate, if the QRS complex, or rather the R wave, is sufficiently narrow, the algorithm decides against shocking. 135 bpm is the lowest rate at which the algorithm can decide to give a shock, depending on the evaluation of the other parameters. A shock decision at 135 bpm is only granted if the rhythm is strongly disorganized and the QRS complex is sufficiently wide. Eventually, the peak-to-peak median amplitude within the segment is checked. If it's below a certain threshold, the segment is considered asystole, and the shock decision is rejected.

The heart rate is a key feature in the assessment of a shockable condition, but in contrast to the TCI algorithm (Section 2.2.4) this feature can be overruled, for example, if the shape and stability of the QRS-complex are within normal values.

## 2.3 Detection of shockable rhythms based on Machine Learning approaches

The exponential growth in computational capacity per spatial unit and the development of self-learning algorithms opened the path for new, promising approaches in almost every field of society, including the field of detecting and classifying cardiac rhythms. When applied, it is most likely that new, machine-learning-driven solutions will outperform the standard, as more and more scientific studies show promising results. And since the requirements for shockable rhythm detection devices formulated in international standards are still at >90% for sensitivity and >95% for specificity, there is room for improvement [46]. Classic algorithms, which are solely based on the combination of signal processing steps and medical expertise, are prone to overlook subtleties, which can add up, especially the more complex the system is and the more ramified the underlying patterns are. In contrast, supervised learning techniques are capable of optimizing the detection or classification model automatically. Machine learning algorithms can make slight adjustments to certain or even random parameters within the model and compare the model output to the ground truth. Automatically means in this context that a great number of iterations in which the model parameters are tweaked can be performed in a short time compared to manual approaches. After each iteration, the model output is compared to the ground truth, which is a set of annotated samples. The learning algorithms are typically designed to discard parameter adjustments which decrease the model performance, while adjustments that improve performance are further examined.

The first observable trend focuses on the combination of ECG features with machine learning classifiers like Support Vector Machine (SVM), Logistic Regression, Random Forest, and Neural Networks [47–50]. Well-known features, such as the TCI metric (Section 2.2.4) are evaluated for their suitability in discriminating between shockable and non-shockable ECG signals. Then a selection of suitable features is implemented in a classifier that is trained by labeled ECG data.

The performance metrics of detection algorithms for shockable rhythms highly depend on the data on which the models are trained and tested. Since there is currently no official standard database to evaluate the models, most of the publications, which are referred to in this work, use public ECG databases with a focus on cardiac arrhythmias, such as the MIT-BIH Arrhythmia Database (MITDB), the Creighton University Ventricular Tachyarrhythmia Database (CUDB), the Malignant Ventricular Arrhythmia Database (VFDB), and the American Heart Association Database (AHADB). Based on data from the mentioned databases, the models reached sensitivity values above 95% and specificity values up to 98% for shockable rhythms [47–50].

The second, more recent trend in the classification of cardiac rhythms is to automate feature detection and selection by utilizing convolutional neural networks (CNNs). The CNN models reached overall accuracy values of 98.8%, with sensitivities of up to 95% and specificities of up to 99% [5,51].

### 2.3.1 Feature selection combined with machine learning classifiers

During the last decades, the research community provided numerous algorithms that proved to be suitable for emphasizing characteristic patterns of shockable cardiac arrhythmias. These characteristic features can be grouped into three categories: time domain features, frequency domain features, and complexity features.

Time domain features focus on heart rate, signal amplitude, slope, or sample distribution [48]. Examples are TCI (Section 2.2.4), threshold crossing sample count (TCSC), standard exponential, and mean absolute value. TCSC counts the samples that cross a certain threshold within 3 s segments [41]. The standard exponential is another count algorithm that sets a decreasing exponential curve on the point where the maximum amplitude within a segment occurs. Then the algorithm counts the crossing points of the ECG signal with the exponential curve within the segment [52]. The mean absolute value refers to the absolute mean of a 2 s segment [53].

Frequency domain features examine spectral concentrations, power contents in different frequency bands, or normalized spectral moments [48]. Examples are the VF filter and the M, A1, and A2 parameters. The VF filter applies a narrowband elimination filter that is centered at the mean signal frequency of the ECG segment [54] [pp. 347-349]. M, A1, and A2 parameters measure the energy content in different frequency bands using the Fourier transform.

The complexity features of the ECG signal are represented by the complexity measure, covariance, frequency, and others [48].

Several studies have examined the combination of the described characteristic features with machine learning classifiers such as logistic regression, SVM and Random Forests [[49], [48], [47]]. To determine a selection of the most relevant features, which generate the most decisive output of the classifier and likewise minimize the computational effort, certain methods were introduced to rank the features regarding their ability to emphasize distinctions between shockable and not shockable rhythms.

### Genetic algorithm in feature selection

One method to find the optimal combination of features uses genetic algorithms, which are adapted from Darwin's evolution theory and mimic the processes of natural selection, reproduction, and mutation. Genetic algorithms select the fittest individuals from a population (the best performing sets of feature combinations) to be reproduced and tested in the next generation [55] [Ch 1]. Therefore, a binary vector with a length equal to the number of all available features is created. 1 in the vector means that the corresponding feature is selected, and 0 means that it's not selected to be used in the classifier. As an analogy, the binary vector can be called a chromosome. The genetic algorithm starts with a population of randomly generated chromosomes, each consisting of a specific combination of selected features. Then each chromosome is tested on the classifier. The chromosomes that provide the best results for the classifier are selected for reproduction, which is determined by the fitness function [56]. As with parental individuals, respectively, 2 of the selected chromosomes are recombined in a crossover process, where a crossover point defines which parent contributes which feature settings to the offspring. To prevent premature convergence, it rarely occurs that some of the feature selections within the offspring can switch from off to on or from on to off. This is called a mutation [56]. When the algorithm doesn't improve the performance of the offspring anymore, the optimization can be terminated.

In a publication from Qiao Li et al., a genetic algorithm was combined with SVM to detect VF and VT rhythms [47]. The features were ranked by the number of times they were selected during the optimization with the genetic algorithm. The most frequently chosen features were then used for classification with SVM. Training and testing were performed on public ECG databases: the Creighton University Tachyarrhythmia Database (CUDB), the MIT-BIH Arrhythmia Database (MITDB), the MIT-BIH Malignant Ventricular Arrhythmia Database (VFDB), and the American Heart Association Database (AHADB). The classifier reached a validation accuracy of 96.3% on distinguishing VF and VT from other rhythms, obtained from fivefold cross-validation.

### Sequential Forward Feature Selection (SFFS)

SFFS defines a basic method to select an optimized set from the entire feature set. First, each feature is evaluated by a common classifier (e.g. k-nearest neighbor algorithm). Then, the features are ranked by performance. Gradually, the feature with the highest score is removed from the entire feature set and added to the optimal feature subset. If the model's performance improves, the last added feature remains in the subset, otherwise, it is removed. The procedure continues until a stopping criterion is met (e.g. maximum number of features in the subset). Hai et al. used SFFS and k-nearest neighbor on data from CUDB and VFDB with rhythms annotated as VF, VT, non-VF, ventricular flutter, and normal sinus rhythm. The model reached 96.7% sensitivity and 99.7% specificity [57].

Another approach to feature selection utilizes the intrinsic feature selection capability of certain classification algorithms, such as regularized logistic regression. The feature ranks are expressed by the magnitude of the regression coefficients after the training phase. More relevant features are related to larger values of the regression coefficient, which gives those features a higher influence on the decision-making within the classifier. The least important features were then iteratively eliminated. The classification performance for each feature subset was then evaluated, and the best subset was chosen. By focusing on VF detection and utilizing public data from CUDB, VFDB, and a subset of 10 recordings from AHADB, the method reached a sensitivity of 96.6% and a specificity of 98.8% [48].

### 2.3.2 Classification of ECG data with Convolutional Neural Networks

Before the approach to explain convolutional neural networks (CNNs) in more detail (Section 2.4) and why they might be the right choice for the classification of shockable conditions from ECG data, a few works applying the method should be mentioned. The works have in common that they utilize labeled data containing shockable conditions from different ECG databases, which are introduced in Section 3.1. The ECG recordings are split into labeled segments, which are used to train CNNs.

The works can be divided into 2 groups regarding the dimensionality of the network and the input data fed into it. The first group refers to 1D CNNs. Here, the input format for the net is a 1-dimensional vector representing the amplitude of an ECG channel over time. Acharya et al use 2 s segments derived from MITDB, CUDB, and VFDB. The data are used for training 11-layer 1D CNNs. The models are evaluated in 10-fold cross-validation and reach an accuracy of 0.932 with a sensitivity for shockable rhythms of 0.953 and a specificity for non-shockable rhythms of 0.910 [58]. Another work by Nguyen et al. deployed 8 s segments from CUDB and VFDB on 13-layer 1D CNNs [59]. The 5-fold cross-validation reaches an accuracy of 0.993 with a sensitivity of 0.971 and a specificity of 0.994. The validation metrics are explained in Section 3.5.

Lai et al., on the other hand, used 2D CNNs. By applying wavelet transform to one-channel ECG recordings, spectrograms are generated and segmented to serve as 2-

dimensional input data for the CNN (see Section 3.2.3). In a 10-fold cross-validation with 3 s segments from AHADB, CUDB, MITDB, and VFDB, the accuracy is 0.988 with a sensitivity of 0.951 and a specificity of 0.994 [51].

## 2.4 Convolutional Neural Networks (CNNs)

Convolutional neural networks were first introduced in 1989 by LeCun [60]. The work applied backpropagation and automated feature detection with a neural network to classify handwritten zip code digits. With the increase in computational capacities and the extensive availability of training data, CNNs are nowadays often the most effective choice for tasks such as image recognition, computer vision, audio, and natural language processing. The secret behind the success of CNNs is based on their flexibility and freedom of choice when it comes to adapting to specific problems.

As described in Section 2.2.4 traditional detection algorithms incorporate static rules and features derived from analytical research. Throughout scientific studies, certain properties of objects are derived that allow a qualitative distinction between the observed objects. These qualities or characteristics are then shaped into mathematical constructs that can be applied to similar objects in order to classify them algorithmically. This handmade feature extraction is often tedious, and it might miss relevant aspects of objects that haven't been included in the scientific study but that might occur in use cases. When we humans identify an object by looking at it, we don't apply the same analytical approach. We wouldn't count certain lines or calculate the orientation of lines in a conscious manner. We use a system that is rather dynamic and adjustable, by design it is capable of learning advanced skills from scratch. For the example of visual recognition, we use our visual cortex, which inspired the development of CNNs [61]. Through our experience of the world, we define the matrix of neural connections within the visual cortex. Depending on the learned configuration of each neuron, a signal is either transmitted further or blocked. The neurons in the, so to speak, input layer of the visual cortex are only locally receptive regarding the visual field that comes through the eye. It means that the input neurons are specialized on a specific region of the incoming image, and they only react to signals coming from that specific region. Furthermore, the first layers in the visual cortex become specialized in reacting to simple, basal structures such as line orientations. Some neurons react only on horizontal lines, others on vertical lines, and so on. The layers that come next react to certain combinations of the lower-level neurons and are able to respond to more complex patterns. The deeper the layers within the net, the more complex the recognized structures are. It starts by recognizing simple lines and develops further by combining them to form components and eventually complete objects [62,63].

As shown in Figure 10, a CNN consists of two main parts. The first part contains convolutional and pooling layers to learn relevant features from the input. The second part is a fully connected neural network that processes the features and concludes with a classification as its output. In the fully connected part, the features from the convolutional part are combined through several layers of neurons, where various combinations of features result in certain votes. The strongest votes eventually determine the class decision of the net at its output layer.

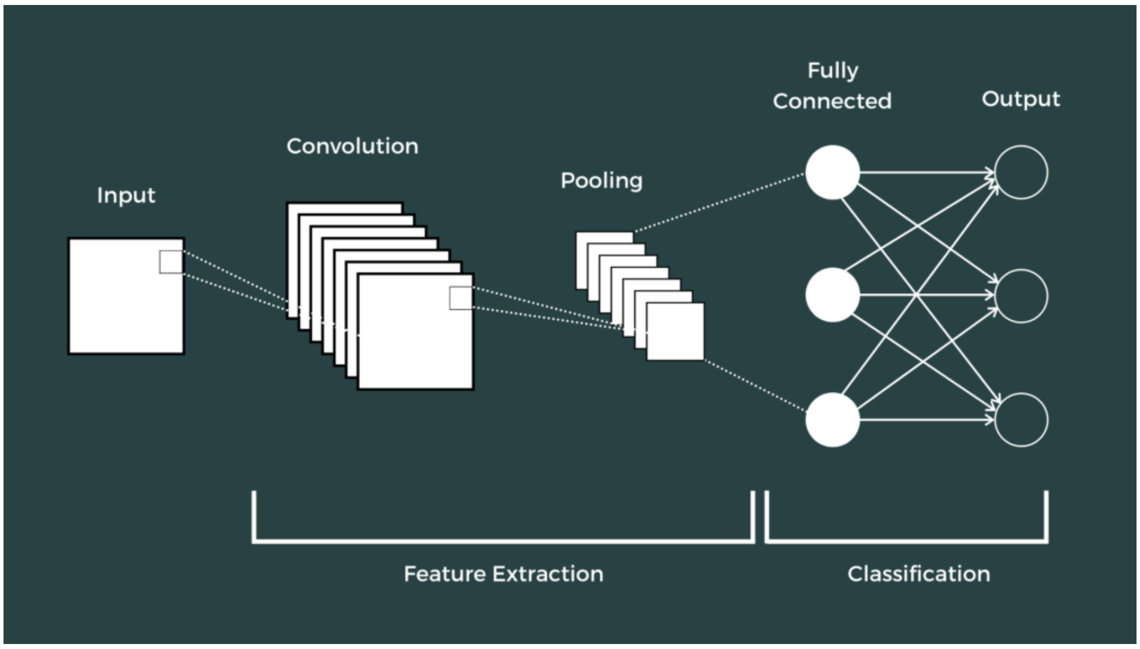


***Figure 10:*** *Structure of* a Convolutional Neural Network ***[64]***

### 2.4.1 Feature extraction in CNNs

**Convolution**

The convolutional block of a CNN is designed to extract the characteristic pieces of an object as features. As an example, a CNN should classify handwritten X-letters on images. What makes the task complicated for a computer is that each handwritten X-letter can vary in its shape, scale, position, rotation, shifting, and line thickness. Therefore, the digital pixel representations of the X-letter images differ as well. To simplify things, the input images have a 9x9 pixel dimension, and the images are binary, meaning 1 for white pixels and -1 for black pixels (Figure 11).

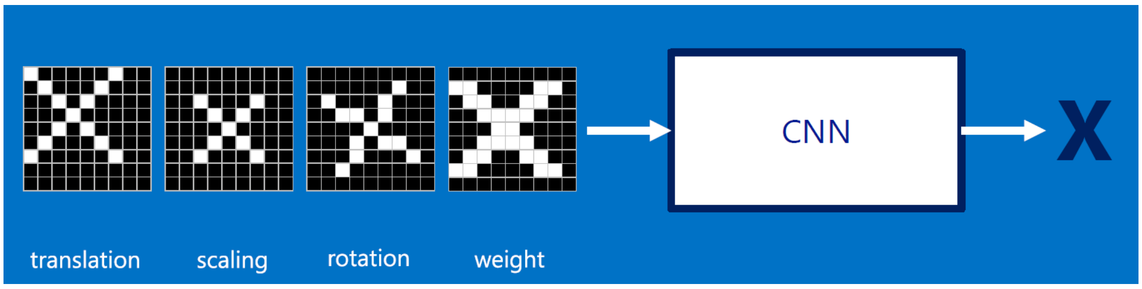


***Figure 11:** Classification of handwritten X-letter representations with a CNN **[65]***

Though all images show the same symbol, their pixel representations are rather varied. A literal match between them, which means to map each pixel value from a reference image to its corresponding pixel value in the sample image, can only result in an acceptable classifier if the sample is compared to a very large number of possible cases in how the X-letter could be written. This is not a very sophisticated way of teaching a computer to classify objects, and considering larger amounts of information, it would require enormous computational effort. Similar to the visual cortex, the CNN approach focuses on extracting simple patterns or basal features such as line orientations within the first layers of the net. This is realized by convolution, as shown in Figure 12. In this example, a small filter kernel of 3x3 pixels is piecewise mapped on the X-letter image. The kernel has a diagonal pattern, with ones from the top left to the bottom right. It is meant to detect diagonal lines with the same or similar orientation in the input image. The mapping starts at the top left corner of the input image and goes incrementally to the right, then line by line down until the image is completely convoluted.

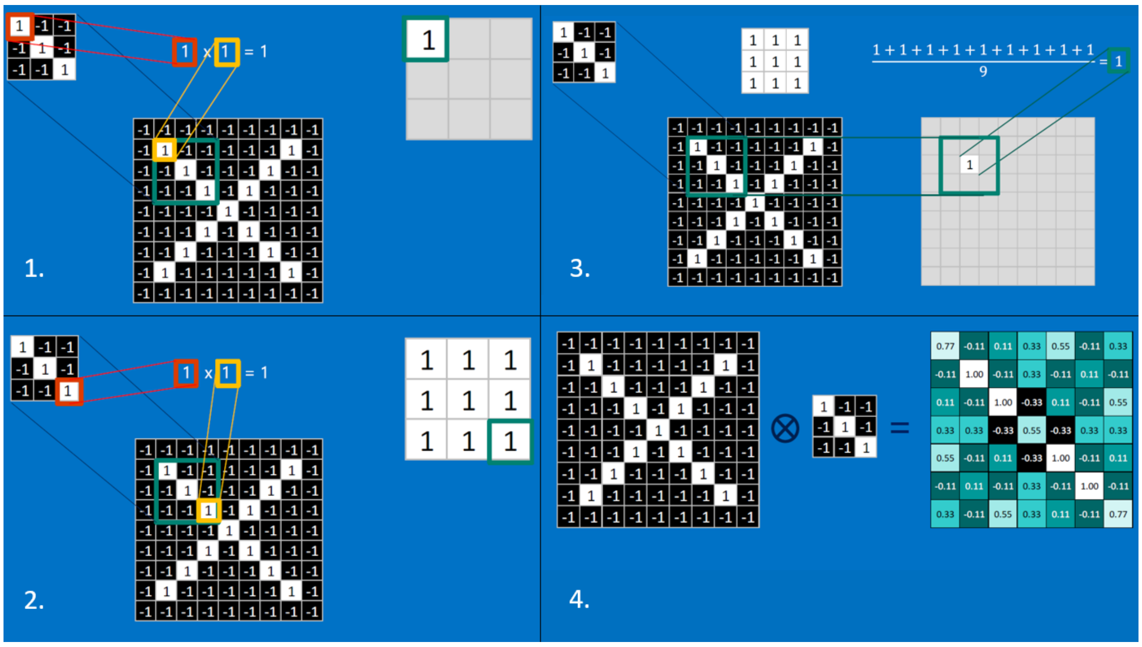


***Figure 12:** Convolution with diagonal line filter kernel **[65]***

When the filter kernel is mapped onto the input image, each pixel value of the image is multiplied by its corresponding pixel value in the filter kernel, as illustrated in steps 1 and 2 in Figure 12. Then in step 3, the products from each pixel are summed up and divided by the total number of pixels in the filter kernel. In this example, the filter kernel and the 3x3 patch on the input image are a perfect match. This equals 1 as the result of the convolution, and it is likewise the maximum number that can be achieved. In image sections where the filter kernel doesn't find good matches, the convolution results in lower values. Step 4 assembles the results of each convolutional step as a filtered map of the input image. In this case, the diagonal lines from left to right are emphasized, while the rest of the original image becomes blurry.

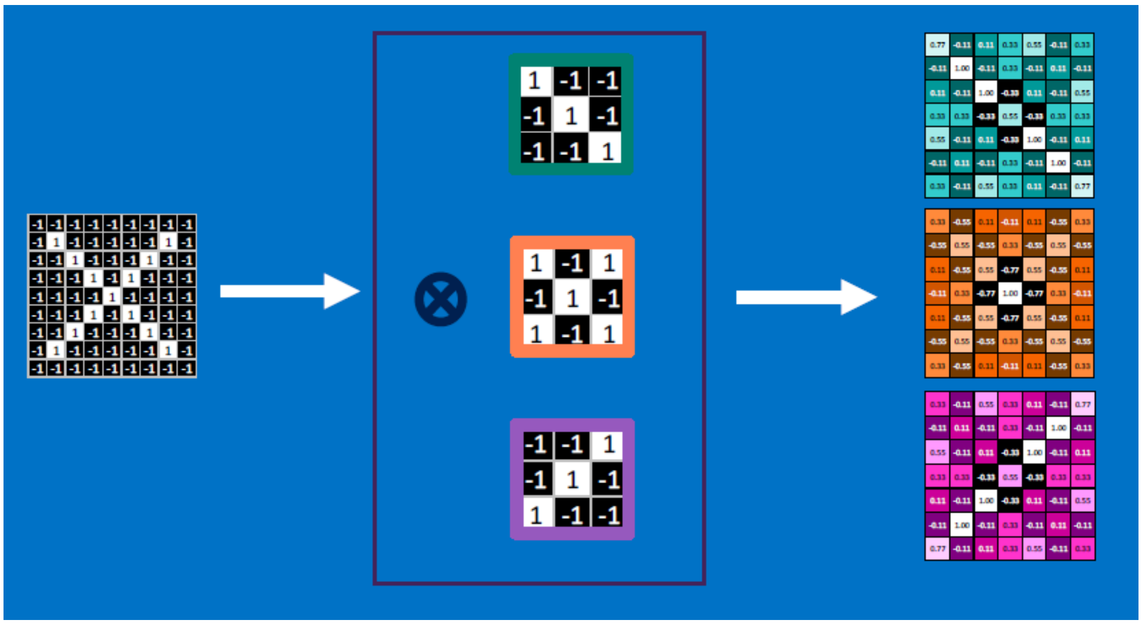


***Figure 13:*** *Convolutional mapping with different filter kernels* ***[65]***

Depending on the nature of the filter kernel, different aspects of the input image can be emphasized (Figure 13). A convolutional layer in the CNN applies a variety of filters to the input image, resulting in a stack of filtered images as the output of the convolutional layer. Each convoluted output image can emphasize different structures of the original input image, such as horizontal or vertical edges.

As explained earlier, the success of CNNs is due to their ability to fit and solve a variety of real problems with high accuracy and in a relatively short time. The key features needed to solve the problem (e.g., to make accurate object classification) are not implemented based on previous knowledge. The network learns the relevant features by itself by working with large amounts of labeled training data. All filter kernel values in each convolutional layer are adjustable parameters. Before training, the kernel values are randomly initialized. During the training process, the network optimizes the kernel values iteratively until no better solution for the given problem can be found.

**Activation**

In the next step, the Rectified Linear Unit Function (ReLU) is applied elementwise on the convoluted pixels. ReLU simply outputs 0 if its input is 0 or lower, otherwise, the input value is preserved. The ReLU layers enable the network to build nonlinear models, which are often required when creating abstractions and predictions of a world with nonlinear relations. If the filter kernel contains negative values or parameters, it might result in a negative value after the convolution. No matter what the negative result might be, ReLU ensures that every negative value is processed as a 0 in the upcoming layers. As an activation function that introduces nonlinearity in the network, ReLU is one of several options, such as the logistic function (Sigmoid) and hyperbolic tangent (Tanh). In 2012, a CNN project called AlexNet implemented the ReLU activation (among other novelties) and outperformed all previous approaches on the ImageNet dataset. Since then, ReLU has become the dominant activation function used after the convolution. ReLU is computationally cheaper. Unlike the other functions, it doesn't require exponential calculation [66]. Another advantage of ReLU is that it passes real 0 values instead of approximations, which reduces noise and simplifies the model [67] [p. 507]. For positive input values, ReLU simply passes them and doesn't show convergence for higher input values, which reduces vanishing gradients [68].

**Subsampling**

Subsampling aims to shrink the input dimensions for computational efficiency while preserving the relevant information regarding the extracted features. Originally, this was done by computing the average of every pixel block of a certain dimension (e.g., 2x2 pixels). Currently, max pooling has become more common [67] [p. 335]. In this operation, the filtered images serve as input. From a block of fixed size, for example 2x2 pixels, only the maximum value is preserved in the output of this layer. While average pooling preserves features in a smoother representation, max pooling retains the most prominent values from the feature map. The pooling window slides with a fixed step size along the input image, resulting in a feature map of decreased size (Figure 14). The shrunk image preserves the features detected by the convolutional layer and discards secondary information. In this way, the computational demand for following layers is reduced, and secondly, the feature detection becomes less dependent on the feature position in the original image. By reducing the number of parameters with pooling layers, the net becomes more robust against overfitting on training data [69] [p. 601].

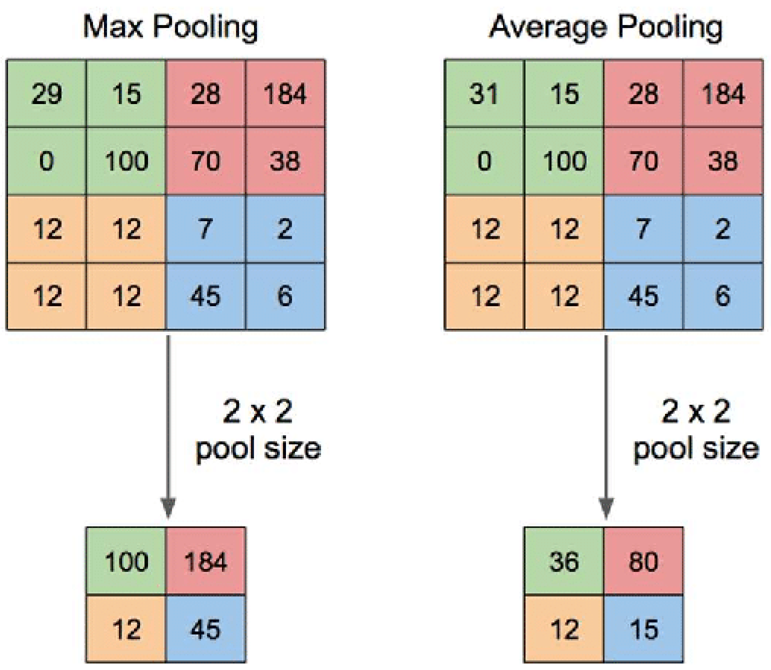


***Figure 14:*** *Max and Average pooling operation in CNNs* ***[65]***

The arrangement of convolutional layers, ReLU layers, and pooling layers can be repeated several times. The deeper the layers within the net, the more complex the learned features become. As in the visual cortex, the first layers in the CNN start by extracting simple features such as edges and bright spots. The following convolutional layers combine low level features with textures. While the convolutional filters search for larger and more sophisticated patterns, which further leads to parts and eventually objects that are abstracted in the deepest convolutional layers of the net [70].

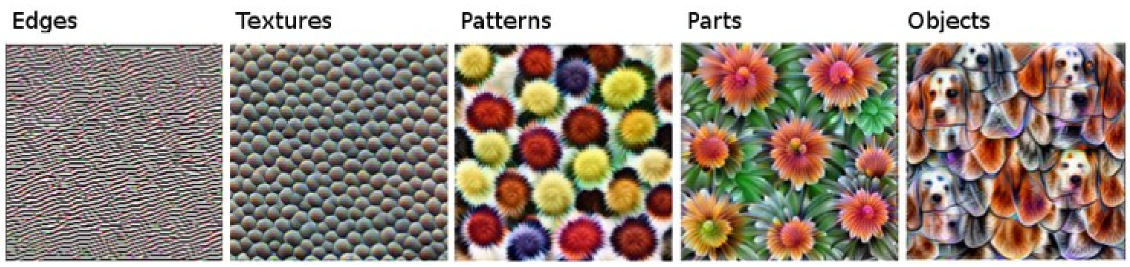


***Figure 15:*** *Feature visualization of convolutional layers (from left - first layers to right - deeper layers) in GoogLeNet (CNN) trained on the ImageNet dataset* ***[71]***

To better understand CNNs and make them easier to interpret, several approaches are focusing on feature visualization [72]. Figure 15 shows feature visualizations of certain channels (feature maps) in different convolutional layers of GoogLeNet CNN, from low-level features on the left side to high-level features on the right side. GoogLeNet was trained on the ImageNet dataset. It has 22 layers and can classify 1000 object categories. To visualize the features at a certain channel in a particular convolutional layer, an image of random pixel values is fed into the network. The activation of the neurons at the location of interest is monitored. A neuron in the convolutional part of the CNN refers to a unit that processes a single convolution, with a receptive field equal to the size of the filter kernel. Then, by applying gradient ascent backwards through the net, the pixel values of the input image are gradually adjusted in a way that maximizes the activation of the neurons of interest. During a loop of up to 2048 steps, the random noise image becomes a visual representation of the neurons behavior as illustrated in Figure 16 [71].

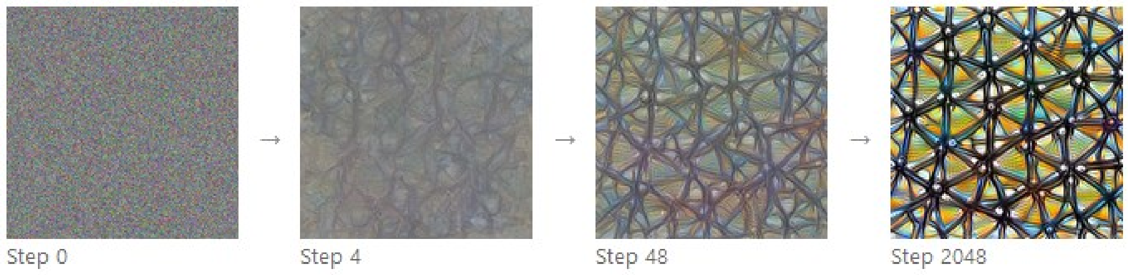


***Figure 16:*** *Feature visualization in CNN from random noise to a representation of maximizing particular neuron activation* ***[71]***

**Flatten**

The final step in the convolutional part of the CNN is to flatten the stacked feature maps from the last pooling layer into a single list, which is referred to as the feature vector. This one-dimensional feature representation serves as the input layer for the fully connected part of the CNN.

### 2.4.2 Fully connected layers in CNNs – Multilayer perceptrons

A multilayer perceptron is an artificial neural network where each neuron is connected to all other neurons in the previous and in the next layer. The artificial neuron (in context with artificial intelligence, also called just neuron) is the building block of an artificial neural network. The components of a neuron are represented in Figure 17. Artificial neural networks are organized in layers, where each layer consists of a certain number of neurons and one bias neuron. The input of a neuron are the incoming signals described as numerical values. In a fully connected network, each neuron's output from the previous layer provides one of the input signals $x_n$ to the current neuron. Each input signal is multiplied by a weight parameter, which can be a positive or negative number, or 0. Weights determine the impact of an input signal on the behavior of the neuron.

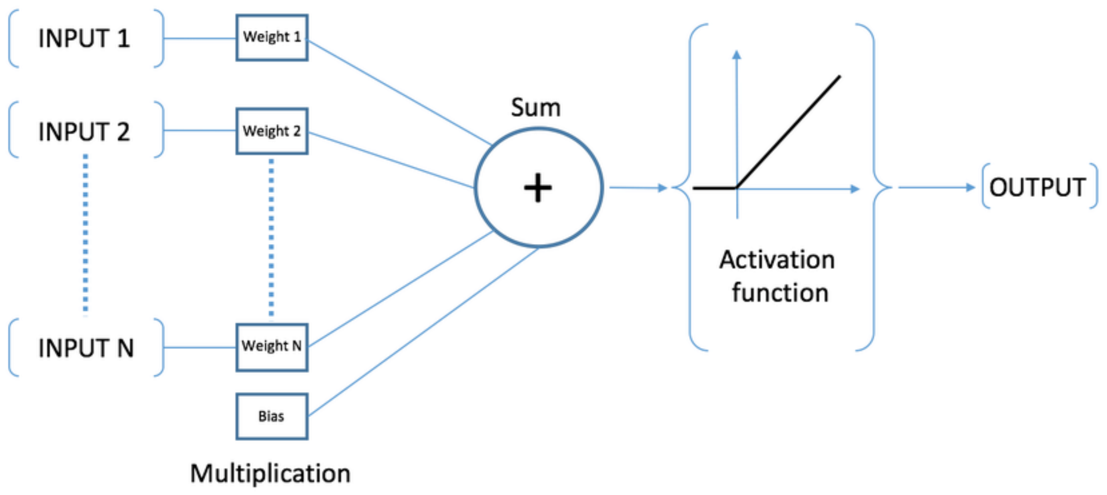


***Figure 17:*** *Components of an artificial neuron* ***[73]***

The multiplications of input signals and weights are summed up to a single value. Further, a bias term is added. The bias term is simply a constant of 1 multiplied by another weight parameter. The bias term is likewise independent from input signals, and by adding it to the summation of weighted input signals, it provides the neuron with the ability to shift its output, similar to a constant added to a linear function that lifts the restriction of the function to be bond to the origin (0, 0). The shifting ability of the neuron makes the neural network more flexible to fit various data [74].

The sum of all multiplications (weights times input signal) plus the bias term is then passed to the activation function of the neuron. Higher weights mean that the corresponding input signals have a stronger positive influence on the neuron's decision to get activated and fire to the next layer. On the other hand, negative weights have an inhibiting effect on the neuron. Weights close to 0 silence the influence of the input signal on the neuron's activation.

**Activation functions in artificial neural networks**

After the weighted sum of the neurons input signals is computed, the resulting value is forwarded to the activation function. The activation function determines if the neuron is going to pass the signal further to the next layer and, if so, with what value it is going to fire. There are other important aspects that need to be considered when implementing activation functions in a neural network: The activation function should emphasize the model's ability to adjust to a variety of problems and enhance classification performance. Further, the activation function needs to be designed in a way that enables the network to learn so that an optimization algorithm can optimize the weights by gradually tweaking them over several iterations.

In 2 dimensions, a problem is linearly solvable if the object classes are separable by straight lines in the 2-dimensional space. But many problems require curved lines to separate the classes effectively. To solve problems such as object classification, where the classes are not linearly separable, it is necessary to implement nonlinear behavior in the model to achieve good performance [75] [p. 72]. This can be realized through non-linear activation functions. Figure 18 illustrates the behaviors of the most common activation functions used in CNN architectures, excluding the step function [69] [p 380]. The x-axis represents the weighted sum of the inputs, whereas the y-axis refers to the neurons output. The step function was one of the first activation functions used in the perceptron by Frank Rosenblatt in 1957 [76]. It gives a binary output of 1 or -1, depending on whether the input value is positive or negative. But when it comes to defining strategies that enable the network to learn the weights, the step function becomes insufficient. When optimizing weights to increase performance, it is desirable to adjust them in small steps and evaluate the impact on the network output iteratively. This becomes problematic with an activation function that is binary and doesn’t have gradients. Another problem is that the step function is not differentiable at 0, and the derivative elsewhere is 0. Updating the weights using backpropagation with gradient descent doesn’t work with the step function.

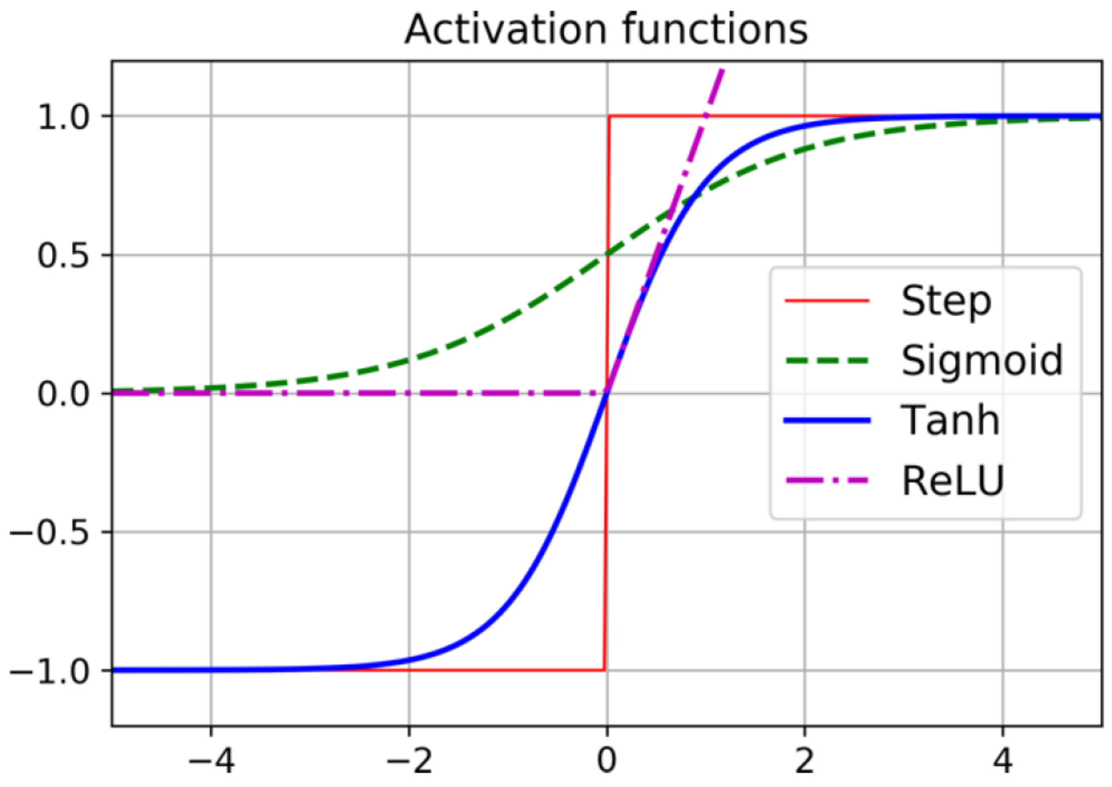


***Figure 18:** Nonlinear activation functions used in artificial neural networks **[69]** [p 380]*

The most common activation functions in deep learning and CNNs are the Rectified Linear Unit Function (ReLU), the Sigmoid function, and the Tanh function. Except for the output layer, it is common to use the same activation function for all neurons within a block of a neural network. As mentioned in Section 2.4.1, the Sigmoid function and the Tanh function have the disadvantage of requiring exponential calculation, which makes the model computationally more expensive [66]. Another issue with Sigmoid and Tanh is their convergent behavior towards higher positive and negative values, which diminishes the gradients for those values. A significant change in the input value might result in an unsignificant change in the output of the activation function. When the learning approach of the neural network is based on gradients, this can slow down or even inhibit the learning process since the weights are adjusted by the magnitude of the gradients. This is also called the vanishing gradient problem  [77]. The fixed ranges for Sigmoid [0, 1] and Tanh [-1, 1] allow the neuron to output a decisive probability, which is a desirable property for classification problems. When using ReLU, this problem can be solved by implementing a normalized exponential function, such as Softmax, in the final output layer of the network. Softmax is a logistic function such as Sigmoid and usually has Euler’s number as the exponential base, resulting in an S-shaped curve around the y-axis.

### Architecture of multilayer perceptrons and CNNs

Figure 19 illustrates the structure of multilayer perceptrons. The feature vector, derived from the convolutional block of the net, serves as the input layer for the multilayer perceptron. From the input layer, each feature value is directly forwarded to each neuron in the first hidden layer. That means that each neuron in the hidden layer receives every feature value from the input layer plus the bias. Each feature value and the bias are multiplied by a weight, specifically for each neuron. The products are summed up and passed to the activation function (e.g., ReLU). Consequently, the output of each neuron in the first hidden layer is forwarded to every neuron in the second hidden layer, and so on. The last layer is responsible for the final prediction and is defined as the output layer. For a classification problem, the output layer has as many output neurons as the given problem has classes. For a binary classification problem, like shockable or non-shockable heart rhythms, the output layer has 2 neurons. With a different activation function, such as Softmax, each output neuron gives a probability for its allocated class. Softmax also ensures that all probabilities from the output layer sum up to 1. The neuron with the strongest vote decides the class prediction of the model.

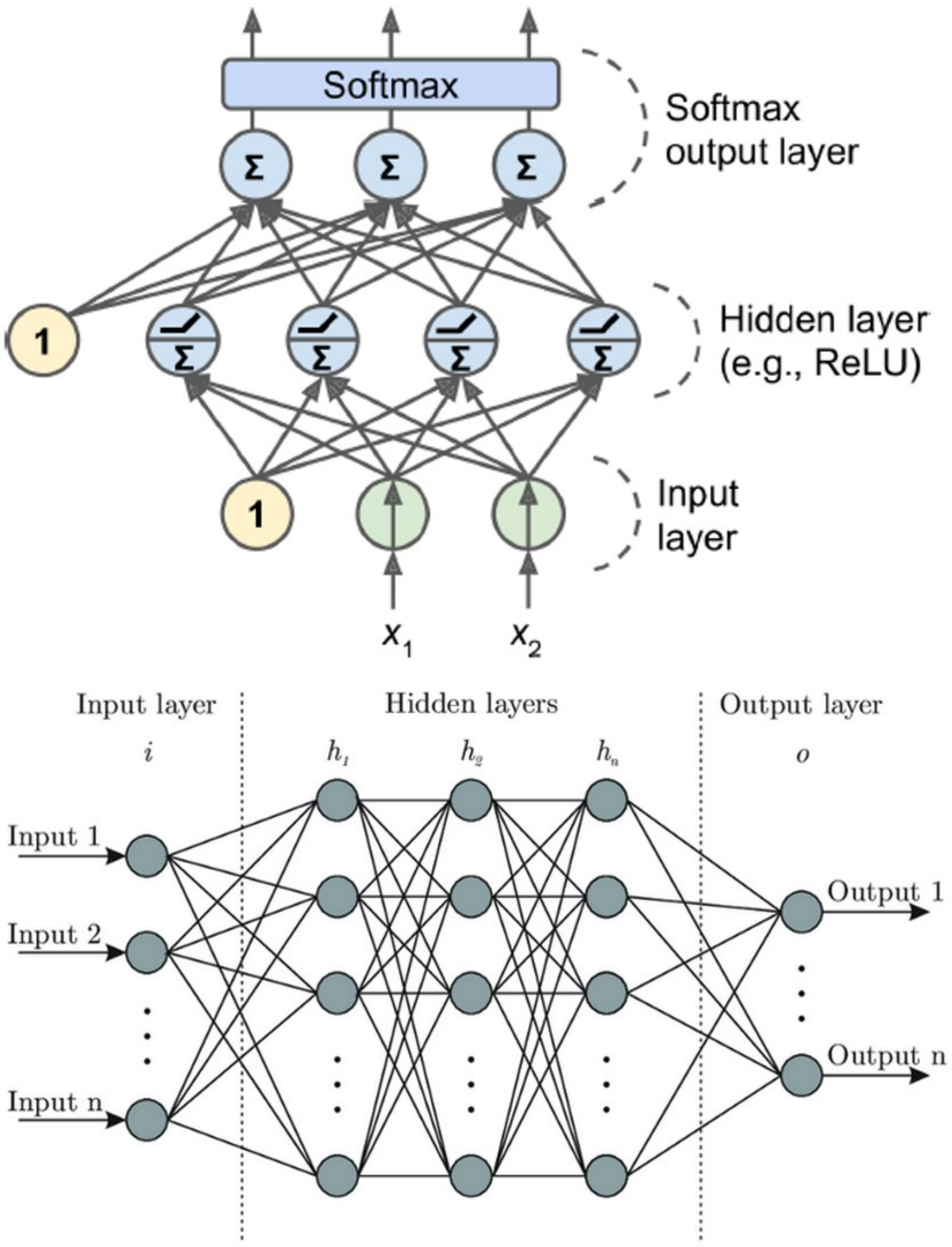


***Figure 19:*** *Architecture of a multilayer perceptrons* ***[69,78]***

Choosing an appropriate size of the CNN (including the convolutional block and the multilayer perceptron part) depends on the given problem and its complexity. For now, there is no analytical approach to calculating the architecture of a CNN, regarding the number of layers and the number of neurons per layer, to achieve good performance on a certain problem [79]. The continuous increase of computational resources allows us to build and train more and more complex networks. History shows that deeper and more complex models tend to perform better on a variety of problems [67] [p. 197].

Since 2010, the ImageNet Large Scale Visual Recognition Challenge has been an annual contest where various models compete in the object detection of images with 1000 classes. The dataset used in the competition contains 1.5 million images with an average of 1000 manually annotated images per class [80]. Between 2011 and 2022, the prediction accuracy has climbed from 50.9% to 91%. The top-five accuracy, meaning that the correct class is under the top-five probabilities predicted by the model, has improved from 73.8% to 99%. As a comparison, humans reach a top-five accuracy of 94.9% on the ImageNet dataset [81]. The novelties in the architecture that helped boost the performance and that endured in later, even better performing models can be considered when designing CNNs. The number of parameters (weights) increased from a few millions to several billions [81]. CNNs tended to become deeper. The likelihood of overfitting increased accordingly. Overfitting was counteracted with regularization, dropouts, and augmented input data. The filter kernel sizes in the convolutional layers became smaller, often to sizes such as 3x3 pixels within the first convolutional layers.

The increasing depth of CNNs caused problems with vanishing or exploding gradients, causing CNNs to lose accuracy after reaching a certain level of depth. The mechanism is explained in more detail in the following Sections about backpropagation and vanishing gradients (see Section 2.4.3). The problem was countered by the introduction of residual CNNs.

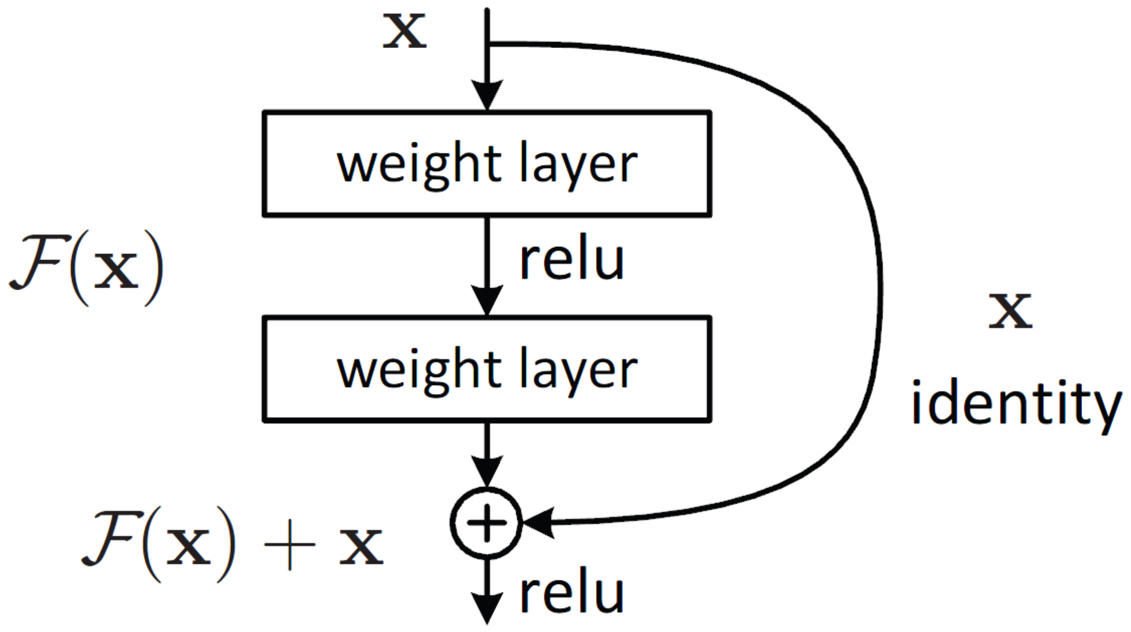


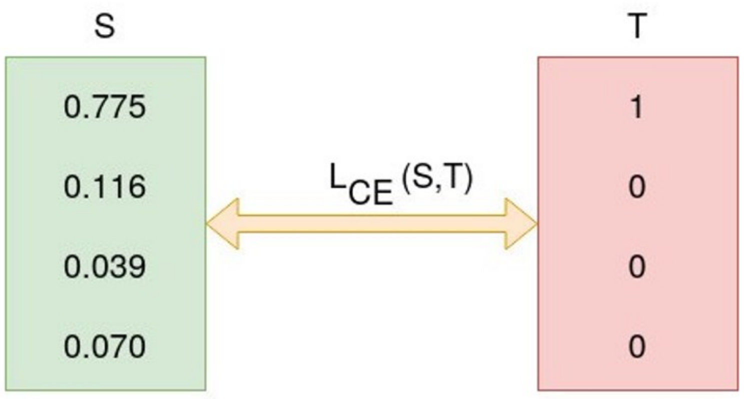
***Figure 20:*** *Residual block in neural networks* ***[82]***

The residual block is implemented as a shortcut, skipping one or several layers in the CNN. The input of a certain layer is passed around this layer and added to the output before reaching the activation function (Figure 20). The skip connection allowed to efficiently train deep CNNs with up to several hundreds of layers, which enabled the CNNs to identify more complex structures and increase performance further [82].

### 2.4.3 How CNNs learn – Gradient descent with backpropagation

**Cost function**

The learning process in a CNN can be described as an optimization problem. Before the training starts, all weights and kernel parameters are initialized with random values within a given margin. When the CNN is fed with training data, e.g., labeled objects on images, it makes a prediction of the input image according to the initial weight setting. Depending on the number of classes to be considered and on the logistic activation function in the output layer, the CNN will give probabilities for each output neuron or class. The neuron with the highest value defines the prediction. After predicting a series of input images, an error rate of the network's performance can be evaluated. The error is based on the cost function, which compares the true label of the input image to its predicted label. This is achieved by calculating a metric distance between the predicted probabilities and the true labels. In that sense, the cost or loss function depends on all weights in the model, so each weight adjustment can reflect on the output of the cost function. The task for the learning algorithm is to adjust the weights so the cost function gets minimized. The most common cost function used for classification problems in CNNs is the cross-entropy loss [67] [p 175]. In the cross-entropy loss, the true label is multiplied with the logarithm of the predicted label probability. Eventually, the multiplications are summed up over all output neurons, resulting in the loss value as illustrated in the example in Figure 21.



$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1\log_2(0.775) + 0\log_2(0.126) + 0\log_2(0.039) + 0\log_2(0.070)\right] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$

***Figure 21:*** *Cross-entropy loss example with 4 output neurons, where T are the true labels and S the predicted probabilities* ***[83]***

After a batch of training samples is forwarded through the CNN, the average of the cross-entropy loss function is computed. The average loss serves as a starting point for adjusting the weights using the learning algorithm.

**Gradient descent**

The backpropagation training algorithm is based on gradient descent, which is a model-fitting approach that iteratively optimizes the weights for the training dataset by minimizing the cost function. Updating the weights is realized by the error gradient, which defines the direction and magnitude of the weight adjustment. The gradient is calculated by the partial derivative of the loss function, which gives the slope of the function for the specific input setting. In a 2-dimensional space, if the slope is negative, the weight should be increased to move towards the minimum to the right. If the slope is positive, the weight should be diminished. This goes incrementally until the gradient converges at the minimum (Figure 22).

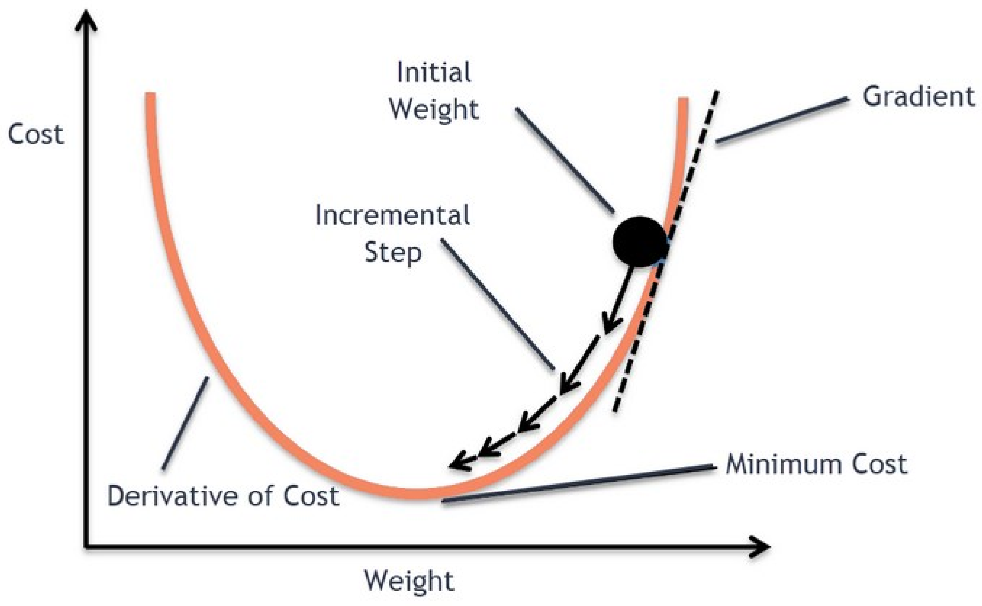


***Figure 22:*** *Gradient descent* ***[84]***

The actual number of dimensions equals the number of weights in the model. The learning rate hyperparameter determines the size of the incremental step. If the learning rate is too small, the optimization might get stuck in a local minimum, plus the number of iterations is increased, which results in a longer computation. On the other hand, if the learning rate is too high, the algorithm might diverge and fail to settle at a minimum. One approach to this problem is to start with a rather large step size for the learning rate. This ensures that the algorithm has good chances of jumping out of local minima. For further iterations, the learning rate is then gradually reduced, which allows the algorithm to converge [69] [pp. 175-185].

**Backpropagation**

Since the output of the cost function is dependent on all weights and biases within the model, which for a CNN can easily reach millions or even billions of parameters, it needs an algorithm to efficiently tweak those parameters to minimize the output of the cost function. The forward pass of a CNN is the regular way for the model to compute its predictions, starting from the input layer all the way down to the output layer. The backpropagation algorithm starts at the output layer by doing a backward pass. Considering an example where only one output neuron is present, backpropagation looks at the cost function (or the difference between the true label and the predicted probability) and aims to determine how strong the influences of each activation from the previous layer are on the prediction of the output layer. In other words, the "brighter" a neuron fires in the previous layer, the more influential is a weight adjustment for this neuron towards the next layer. Further backwards, the influence or gradient of this neuron in the previous layer depends again on the activations in the layer before, and so on. The backpropagation algorithm determines the gradients for every connection in the network by applying the chain rule. Updating a specific weight *w* in the model is realized by multiplying the gradient at this weight with the learning rate Epsilon and then subtracting the term from the weight *w*:

$$w := w - \epsilon \frac{\partial C}{\partial w}$$

***1***

The gradient is defined as the partial derivative of the cost function C over the weight of interest [85]. All the weights and biases between the cost function (output layer) and the weight of interest at a certain layer in the net need to be considered when computing the gradient. This is computed by applying the chain rule, which will be further explained with *a* simplified example as illustrated in Figure 23. The variables *w* and *b* stand for weight and bias. *z* is the computed neuron value before entering the activation function. *a* is the activation value after passing the activation function. z is calculated as follows:

$$z = (\sum_{i=1}^{n} w_i \times a_i) + b$$

***2***

*w* are the weights and *a* are the activation values from the previous layer.

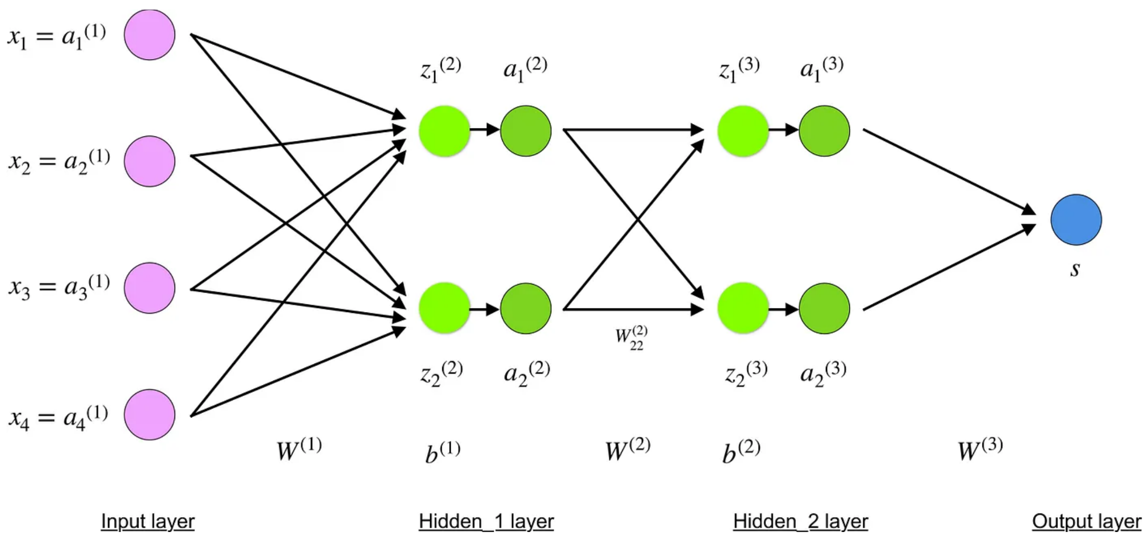
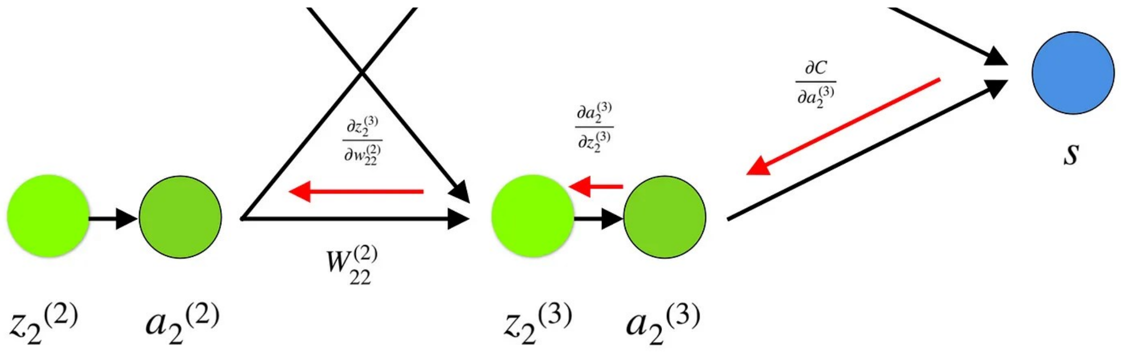


***Figure 23:*** *Simple 4-layer neural network with 4 inputs, 2 hidden layers, and 1 output* ***[86]***

To get the gradient for weight *w₂₂*, the algorithm needs to backpropagate from the cost function backwards to *w₂₂* by applying the chain rule to unnest the partial derivative of the cost function over *w₂₂*, as visualized in Figure 24:



$\frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$
$\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}$
$\frac{\partial C}{\partial a_2^{(3)}}$
$s$
$W_{22}^{(2)}$
$z_2^{(2)}$
$a_2^{(2)}$
$z_2^{(3)}$
$a_2^{(3)}$

***Figure 24:*** *Visual representation of the backpropagation algorithm in a neural network* ***[86]***

The superscripted numbers indicate the corresponding layer. With the chain rule, the gradient for *w₂₂* is calculated as the derivative of the cost function as follows:

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \, . \, a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \, . \, f'(z_2^{(3)}) \, . \, a_2^{(2)}$$

***3***

The above description of backpropagation applies only to a single input sample. In the real use case, a whole batch of input samples is considered before adjusting the weights. After a sample is forwarded through the network, the cost function is calculated. Then the backpropagation algorithm calculates a gradient for each parameter (weights and biases) in the model. This represents a suggestion on how the parameters should change to make a slightly better prediction for the current input sample. The extent to which the parameters change depends on the learning rate. The batch size of input samples defines how many gradients are computed before the weights are adjusted. If the batch size is, for example, 32, then the algorithm will calculate 32 suggestions (gradients) for each parameter in the network. Eventually, the 32 gradients are averaged, resulting in the final gradient that defines the adjustment of the weight. Following the development of the cost function, the learning algorithm continues until no further improvement in minimizing the cost function is registered for a certain number of batch iterations.

Considering that the gradients need to be calculated for every parameter and for each training sample when millions or even billions of parameters are present and hundreds of thousands of training samples, then backpropagation becomes a computationally expensive endeavor. The historical jump of halving the error rate in object detection on images, which was achieved by AlexNet in 2012, could be realized because the backpropagation was computed by GPUs (graphics processing units) [66]. GPUs have chips that are optimized for calculating rather simple tasks compared to a CPU (central processing unit). But the main advantage of the GPU is that it has a lot of those chips computing in parallel.

### Vanishing/Exploding gradients problem

As mentioned in Section 2.4.2, an increase in layer depth and the use of backpropagation can lead to vanishing gradients in the front layers when training neural networks. Small gradients close to 0 imply that the regarding weights don't really change when they're updated. This is caused by the chain rule that multiplies partial derivatives and activations backwards from the output layer. Several small multipliers in the equation are likely to cause a result close to 0, which prevents the weights from actually updating their values and prevents the model from learning. The effect significantly increases with the degree of deepness and when using saturating activation functions such as the Sigmoid function. The Sigmoid function squeezes its input value space into a much smaller value space between 0 and 1. It also saturates on both ends, which causes the derivatives there to get close to 0. The problem can be countered by using non-saturating activation functions such as ReLU and by implementing residual blocks in the network [77]. On the other hand, if several large derivatives appear in the backpropagation chain, they can cause very large gradients in the frontal layers, which is referred to as exploding gradients. Exploding gradients lead to an unstable training process and prevent the model from learning effectively. Gradient clipping introduces a solution against exploding gradients by checking and limiting the gradient values during the backpropagation [87].

### Normalization and Batch Normalization

Normalization of input data increases learning speed and model performance by balancing the value range of the input data, which makes the model more robust against varying input data and improves generalization. For CNNs, the pixel values of the input data are commonly normalized between 0 and 1. Another approach is standardization scaling. It is implemented by subtracting the mean value of the data from the input value and dividing the result by the standard deviation of the data [75] [p. 260].

Batch normalization follows a similar approach, but instead of normalizing the input data, it aims to normalize the values in the hidden layers of the network. In other words, batch normalization reduces the amount by which hidden layer units shift around when confronted with alternating distributions of input values from previous layers. This provides an advantage to the next hidden layer in that the distribution of input data on which it has to learn is reduced. Overall, deeper layers become more robust against shifting input data and provide them with a more stable foundation for their learning process. Batch normalization is applied to the current batch of training data and can be implemented by applying standardization scaling.

# 3. METHODS AND MATERIALS

## 3.1 ECG Databases

Various detection algorithms for shockable rhythms have been developed utilizing public databases for cardiac arrhythmias. The most prominent databases used in this field of research are the MIT-BIH Arrhythmia Database (MITDB), the Creighton University Ventricular Tachycardia Database (CUDB), the MIT-BIH Ventricular Arrhythmia Database (VFDB), and the American Heart Association Database (AHADB) [41–44,47–53,58,88–91]. These four databases are also utilized for this work. ECG signals have been obtained from cardiac monitoring, with 2 channels for AHADB, MITDB, and VFDB, and 1 channel for CUDB [92]. The databases were implemented to provide developers of detection algorithms with standard arrhythmia datasets to evaluate, compare, and reproduce algorithms from different research groups [93,94]. Due to the expensive and tedious process of manually annotating ECG waveforms and the exclusion of recordings with cardiopulmonary resuscitation artifacts, the databases contain less than 200 recordings in total [95]. Even so, the datasets are valued as reliable sources for algorithm development. The data were selected to represent the spectrum of variability in ECG rhythms and waveform morphology and also include uncommon but clinically important arrhythmias. Further, the databases consider a wide range of patient age groups and include inpatients as well as outpatients [93,94].

***Table 1.*** *content overview of the accessed data from 4 arrhythmia databases*

| Database | Recordings | Sampling frequency (Hz) | Duration (min) | Shockable conditions |
|---|---|---|---|---|
| MITDB | 48 | 360 | 30 | VT, VFL |
| CUDB | 35 | 250 | 8 | VF, VT |
| VFDB | 22 | 250 | 30 | VF, VT, VFL |
| AHADB | 20 | 250 | 30 | VF-VFL |

Table 1 gives an overview of the acquired data from the medical databases. In summary, 125 recordings have been processed. Originally, the AHADB contained 80 recordings, of which only the sets including shockable conditions were utilized in this work.

For the AHADB, only the last 30 mins of the 3-hour recordings were considered because only those intervals are properly annotated for the supervised learning approach. The last column in Table 1 contains the shockable conditions that were found in each database. AHADB doesn't make a distinction between VF and VFL conditions, instead, the conditions are annotated as VF-VFL. Therefore, the class of shockable conditions contains 4 subclasses: VT, VF, VFL, and VF-VFL.

## 3.2 Pre-Processing

The pre-processing includes all measures for preparing the ECG data to be processed by CNNs. Figure 25 illustrates the consecutive steps that are applied to the raw ECG data. The final dataset management and the conversion to image format are realized with Python. All other steps are implemented in MATLAB. The program code is provided in the appendix (Section 6). The main script of the pre-processing steps can be followed on Figure 44-Figure 45.

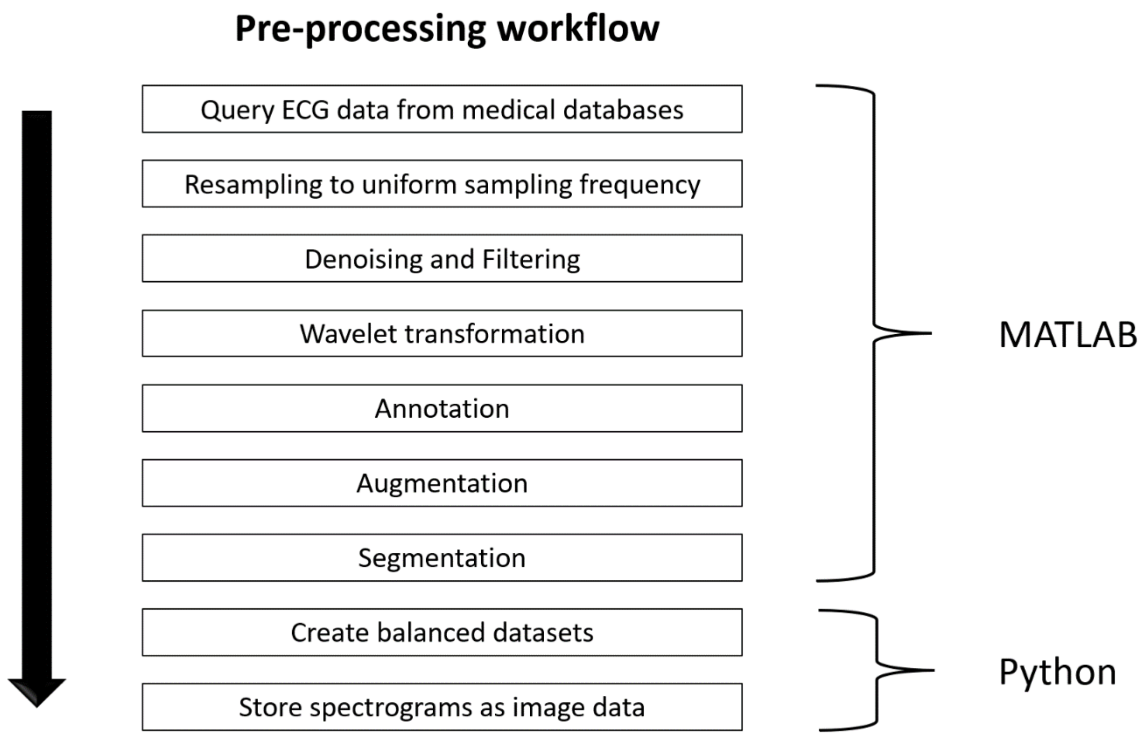


***Figure 25:*** *Pre-processing workflow from ECG raw data to trainable spectrogram representations for CNNs*

The quality and quantity of training data are crucial for the learning success of CNNs. The best model is going to perform poorly when it's trained with insufficient, low-quality data. Therefore, gathering and preparing data is important and often the bottleneck in machine learning applications. In general, it can be said that the more comprehensive the training data is regarding the real-world situation, the better the chances are for the model to perform well in real-world use cases. For the model to learn all the relevant patterns that represent the properties of objects belonging to one class and not another, it is only possible if the training data covers those properties.

For classification, it is important that each class has enough prevalence in the training data to have a sufficient impact on adjusting the weights in the network during the learning process. Ideally, the classes are balanced. This means that in a binary classification, the data would consist of 50% samples for each class. Often, this is not the case. Regarding the detection of shockable heart rhythms, it is to be assumed that the dataset will be unbalanced. Shockable conditions occur very rarely, and that they are recorded is even rarer, which is partially compensated by the fact that the databases utilized in this work are specialized on arrhythmias. It is not only important to aim for balanced classes, but it’s likewise crucial to provide enough samples that capture the variety within a class. The different shockable conditions, namely VT, VF, and VFL, should be aimed at having a balanced distribution within the class of shockable rhythms. Balanced distributions are not always within reach when preparing the data, but one goal of the pre-processing is to avoid that the gaps between the classes and between different object categories within a class become too big, which would have negative effects on the learning process of the model. This problem is tackled by thorough dataset management and the augmentation of certain parts of the data.

Another aspect of the pre-processing is to clean the data from noise and artifacts to support the model in focusing on relevant features of the data. CNNs are designed to work properly on digital image representations. Therefore, the conversion from ECG waveforms to spectrograms is realized by applying wavelet transformation. Further, the spectrograms are segmented into consecutive samples of 3 s length. The segment labeling is organized in a way that not only provides a label for supervised learning but additionally provides a data frame that breaks down all the annotations that are contained in a segment.

### 3.2.1 Query ECG data from medical databases

3 of the 4 databases, namely MITDB, CUDB, and VFDB, are accessible via HTTP on Physionet servers. The WaveForm DataBase (WFDB) toolbox for MATLAB provides functions to automatically query and download the ECG data as well as attached meta data [96]. The rdsamp() function fetches the ECG data as a waveform converted into a data frame with a column for the time stamp in s and 1 or 2 columns for voltage in mV. The CUDB contains single-channel recordings. For the other databases, the second column corresponds to the upper ECG channel, whereas the third column belongs to the

lower ECG channel. The upper channels consistently refer to a modified limb lead II (MLII), where the electrodes are placed on the chest as in standard ambulatory ECG recording. The lower channels vary between V1, V2, V3, V4, and V5 [93]. Since the upper channels provide more consistency in electrode positioning and the positioning is relatively close to an AED use case, the lower channels were discarded from further processing. Additionally, previous works regarding the automated detection of shockable rhythms have used the upper channels as well, which supports the comparability of the different approaches [49,51,58,97].

AHADB access is restricted and not accessible with the WFDB toolbox. The license for the AHADB was obtained, and the data was provided in CSV files, representing the ECG signal in waveform with the first column as the upper ECG channel in mV, the second column as the lower ECG channel in mV, and the third column for the annotation label in between quotes.

### 3.2.2 Resampling, denoising and filtering

As pointed out in Table 1 (Section 3.1), CUDB, VFDB, and AHADB provide data with a sampling frequency of 250 Hz, whereas MITDB data is sampled at 360 Hz. For further processing, the ECG time series and the annotations are resampled to a uniform sampling frequency of 250 Hz.

The filtering of the raw signals is based on previous research that carried out the detection of shockable rhythms obtained from ECG signals [47,90,98]. The filtering is exemplarily illustrated on a 3 s ECG segment in the waveform in Figure 26.

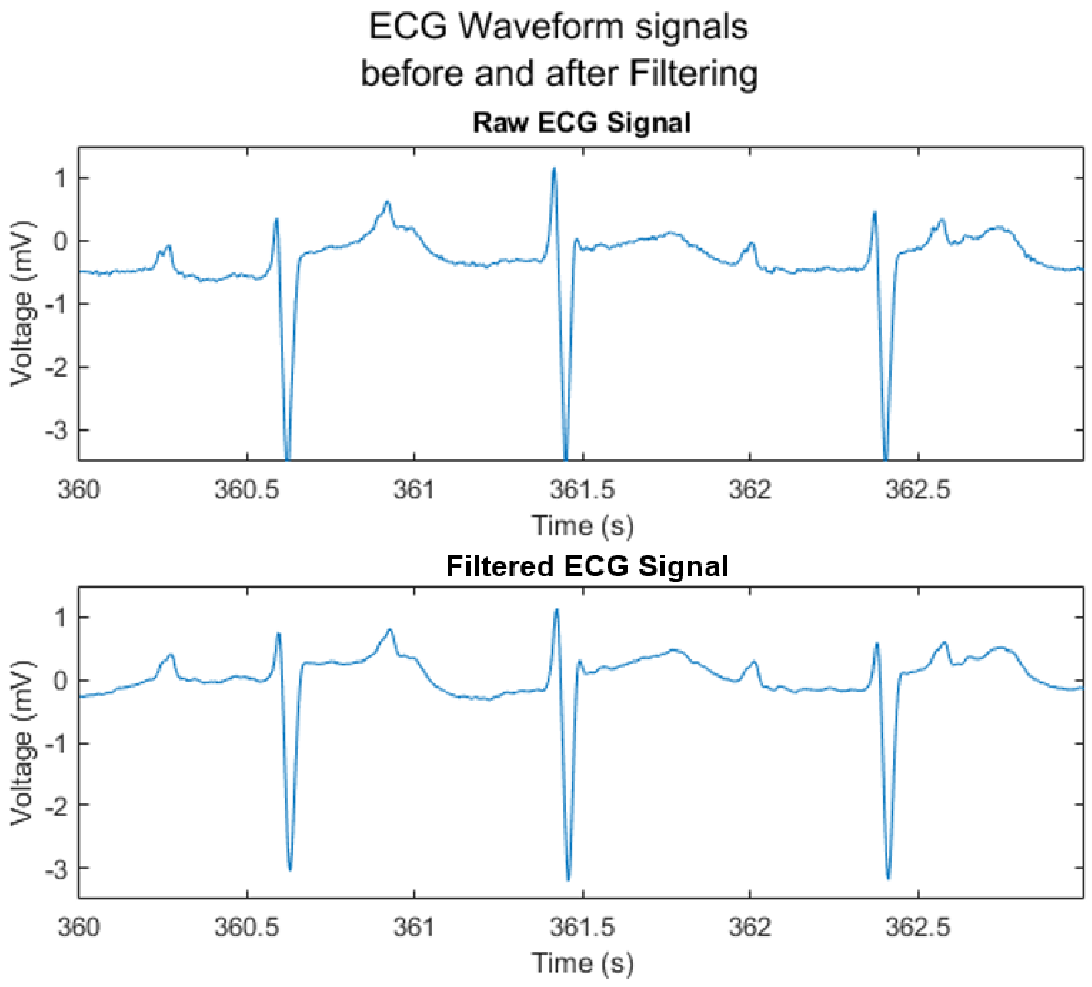


***Figure 26:*** *3 s ECG segment in waveform before and after filtering; source: VFDB recording 614 from sample 90001 to 90750*

Baseline drifting refers to low-frequency artifacts caused by breathing, movement, and charged electrodes [99]. The residual baseline drifting is suppressed by applying MATLAB's default high-pass filter from the signal processing toolbox with a 1 Hz cutoff frequency. The value for filter steepness is 0.85, which sets the transition width of the filter to 15% of the passband frequency. Muscle noise is reduced by a second-order Butterworth low-pass filter at a 30 Hz cutoff. The normalized cutoff frequency is calculated by dividing the cutoff frequency by half the sampling frequency. The transfer function coefficients are computed by MATLAB's butter() function from the signal processing toolbox. The filtering is realized by applying the filter() function with the transfer function coefficients and the output signal from the high-pass filter as its arguments. Since the amplitude response of the second-order Butterworth filter is relatively smooth, an additional notch filter is implemented at 50 Hz to cancel powerline interferences. The code for querying the data, resampling, and filtering can be found in Figure 46 in Section 6.

### 3.2.3 ECG as spectrogram representation by Wavelet Transform

The wavelet transform is an algorithm that is utilized to decompose signals, which are not stationary, into a spectrum of frequencies over short time intervals. So far, the ECG segments are represented as a discretized signal of voltage amplitude over time (as visualized in Figure 26). The wavelet transform allows one to visualize signal features in frequency and time domains parallelly. Therefore, the segments can be converted into an image representation that presents the signal information in an appropriate way for the CNN to learn the characteristic patterns of specific heart rhythms.

The wavelet transformation generates a spectrogram representation of the ECG signal. The information units of the spectrogram are defined as small boxes containing 3 dimensions. The horizontal dimension represents the time resolution, whereas the vertical dimension stands for the frequency resolution. The pixel intensity represents the amplitude of a certain frequency range in a certain time interval. The spectrogram arranges the information units in 2-dimensional space with time on the horizontal axis and frequency on the vertical axis. Low frequencies appear at the bottom and high frequencies at the top. The time domain proceeds from left to right. Figure 27 to Figure 29 show examples of transformed ECG segments for normal heart rhythm, VT, and VFL.

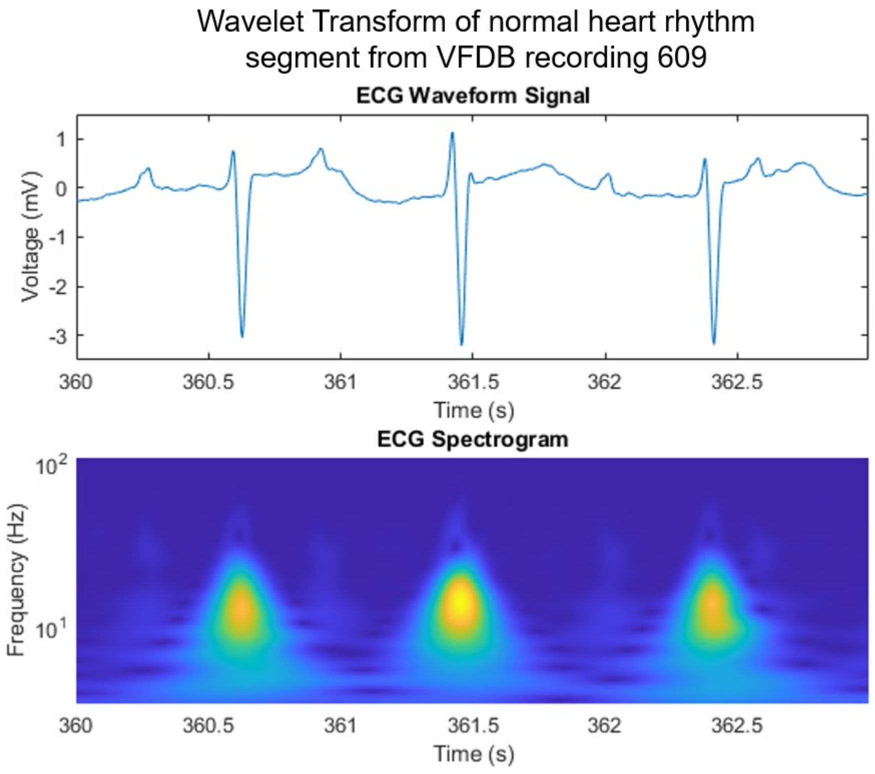


***Figure 27:*** *ECG signal and its Spectrogram representation from a normal heart rhythm episode in VFDB recording 614 from sample 90001 to 90750*

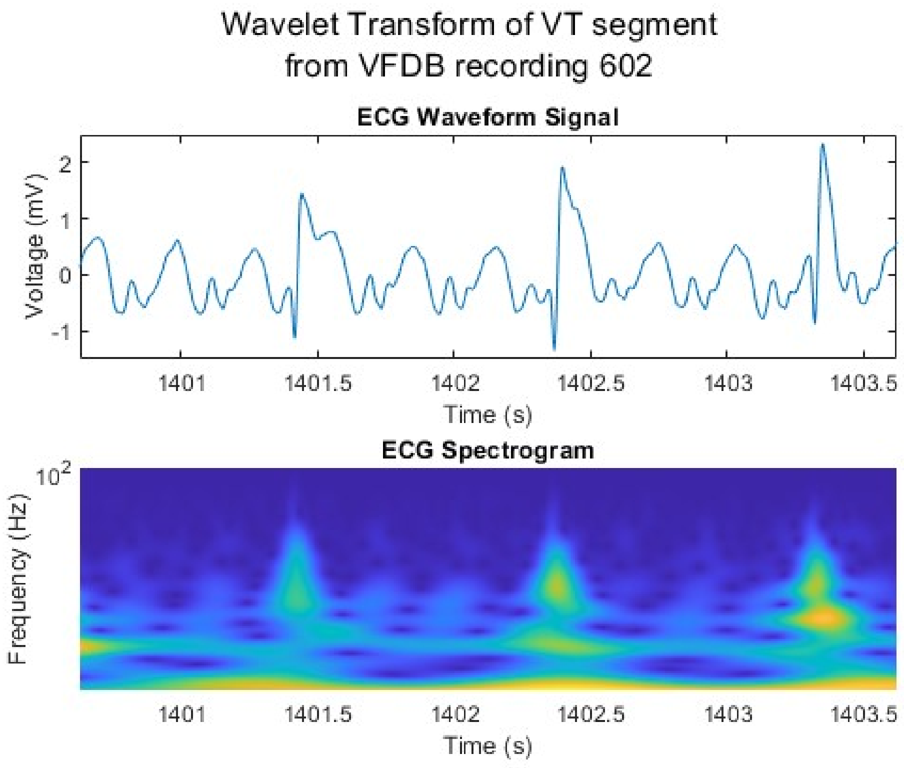


***Figure 28:*** *ECG signal and its Spectrogram representation from a VT episode in VFDB recording 602 from sample 350158 to 350907*

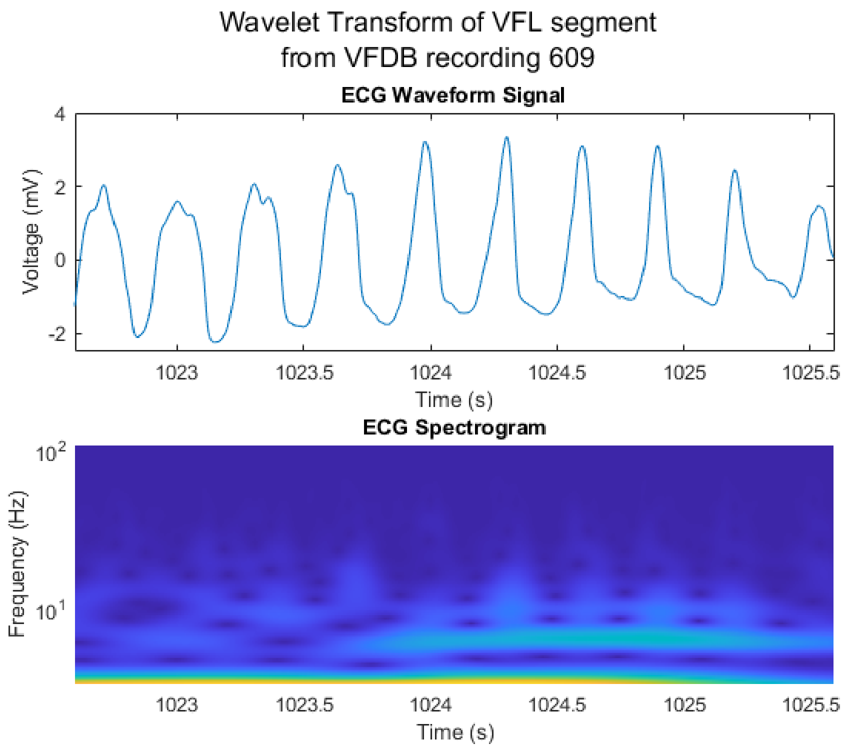


***Figure 29:*** *ECG signal and its Spectrogram representation from a VFL episode in VFDB recording 609 from sample 255650 to 256399*

The wavelet transform is based on the Fourier transform, which decomposes functions into a series of cosine and sine functions by applying Euler´s formula. The standard Fourier transform can determine the frequency components in the signal, but it doesn’t provide information on the time at which a particular frequency occurred. To analyze non-stationary signals, the short-time Fourier transform was developed [100]. A sliding window function extracts the frequencies of the signal over a defined time interval with a fixed time-frequency resolution. The short-time Fourier transform doesn’t consider Heisenberg’s uncertainty principle, which states that a particle's position and momentum are not entirely determinable. This means that for windows with shorter time intervals, the time resolution increases, but likewise the frequency resolution decreases. For windows with wider time intervals, the frequency resolution increases, but the certainty of when a certain frequency occurred diminishes. The wavelet transform encounters this problem by defining a mother wavelet from which different window functions are dilated or compressed. In that manner, high frequencies are captured with shrunken wavelets to achieve high time resolution. Respectively, slowly changing low frequencies are resolved by expanded wavelets with regard to the mother wavelet.

The wavelet transform is applied by utilizing MATLAB’s continuous 1-D wavelet transform. The chosen mother wavelet is Morse. With the property of being exactly analytic, Morse wavelets avoid interference and artifacts in the time-frequency plane, which increases the accuracy of amplitude and phase estimates. Morse wavelets have been proven to be quite useful for time-varying spectrum estimation on biosignals. In order to train classifiers to distinguish cardiac conditions based on localized events of short duration, the Morse wavelet allows to generate suitable signal representations and is adjustable in refining localization in frequency and time [101].

The number of octaves is set to 5. The number of voices per octave is set to 20 and defines the number of wavelet filters applied per octave. The number of voices per octave and the number of octaves affect the scaling of the wavelet. The parameters are proportional to the time domain window and inversely proportional to the frequency domain window, and therefore are set to a balanced tradeoff between the two domains. The time bandwidth product is set to 50. The wavelet parameter settings are derived from Lai et al. and have been proven to generate spectrograms suitable for ECG data analysis with 2D CNNs [51]. When having a working CNN model, it might be interesting to experiment with different wavelets and parameter settings and evaluate their effect on the model’s performance. This is not carried out since it exceeds the scope of this work. It defines the time bandwidth of the Morse wavelet and can range between 3 and 120. Higher values generate wavelets with larger spreads in time, whereas lower values generate wavelets with larger spreads in frequency.

The spectrograms are segmented into samples of 3 s. Eventually, the segments are stored as grayscale images (pixel values ranging from 0 to 255) in portable network graphics (PNG) format with a size of 100x100 pixels. The file conversion is realized with the cv2 library in Python.

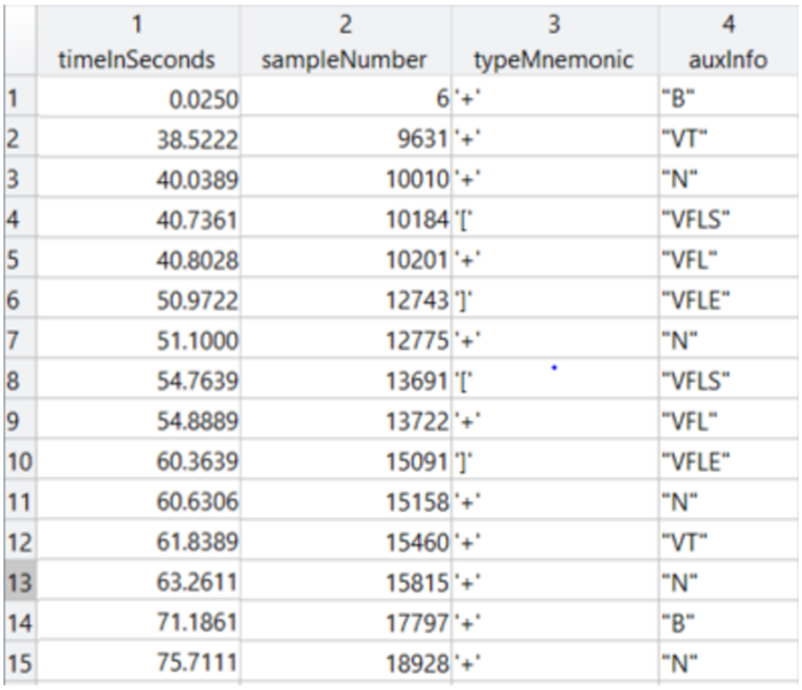
### 3.2.4 Annotations

This part describes how the medical annotations for the ECG data are extracted and processed to be mapped on the spectrogram segments. Table 2 gives an overview of all annotations and their medical explanations occurring in the processed ECG data. The shockable conditions are listed in the lower part of the table. During the pre-processing, the labels VF and VFIB are combined into VF since they represent the same condition. The MATLAB code can be found on Figure 47 in the appendix.

***Table 2.*** *List of cardiac annotations corresponding to the ECG data extracted from MITDB, CUDB, VFDB, and AHADB*

| Class | Annotation | Explanation |
|---|---|---|
| Not Shockable | AB | Atrial bigemy |
| | AF | Atrial fibrillation |
| | AFIB | Atrial fibrillation |
| | AFL | Atrial flutter |
| | ASYS | asystole |
| | B | Ventricular bigeminy |
| | BI | first degree heart block |
| | BII | 2° heart block |
| | HGEA | high grade ventricular activity |
| | IVR | Idioventricular rhythm |
| | N | Normal beat |
| | ND | not defined (AHADB) |
| | NOD | Nodal (A-V junctional) rhythm |
| | NOISE | noise |
| | NSR | Normal sinus rhythm |
| | P | Paced rhythm |
| | PM | pacemaker (paced rhythm) |
| | PREX | Pre-excitation (WPW) |
| | SBR | Sinus bradycardia |
| | SVTA | Supraventricular tachyarrhythmia |
| | T | Ventricular trigeminy |
| | VER | Ventricular escape rhythm |
| Shockable | VF | Ventricular fibrillation |
| | VFIB | Ventricular fibrillation |
| | VFL | Ventricular flutter |
| | VF-VFL | Ventricular fibrillation or flutter |
| | VT | Ventricular tachycardia |

For MITDB, CUDB, and VFDB, the rdann() function from the WFDB toolbox is used to query the annotations, which are then transformed into tables. The AHADB provides the annotations as a column next to the ECG waveform data, where each sample of the recording is already labeled. The annotation tables for the other databases include time stamps and sample numbers of starting or changing events. The following examples are derived from MITDB's recording 207. Figure 30 shows an extract from the annotation table. Annotations marked as "[]" are replaced by VFLS for a beginning ventricular flutter and VFLE for an ending ventricular flutter episode. The sample numbers are resampled from 360 Hz to 250 Hz in order to match the resampled ECG data.



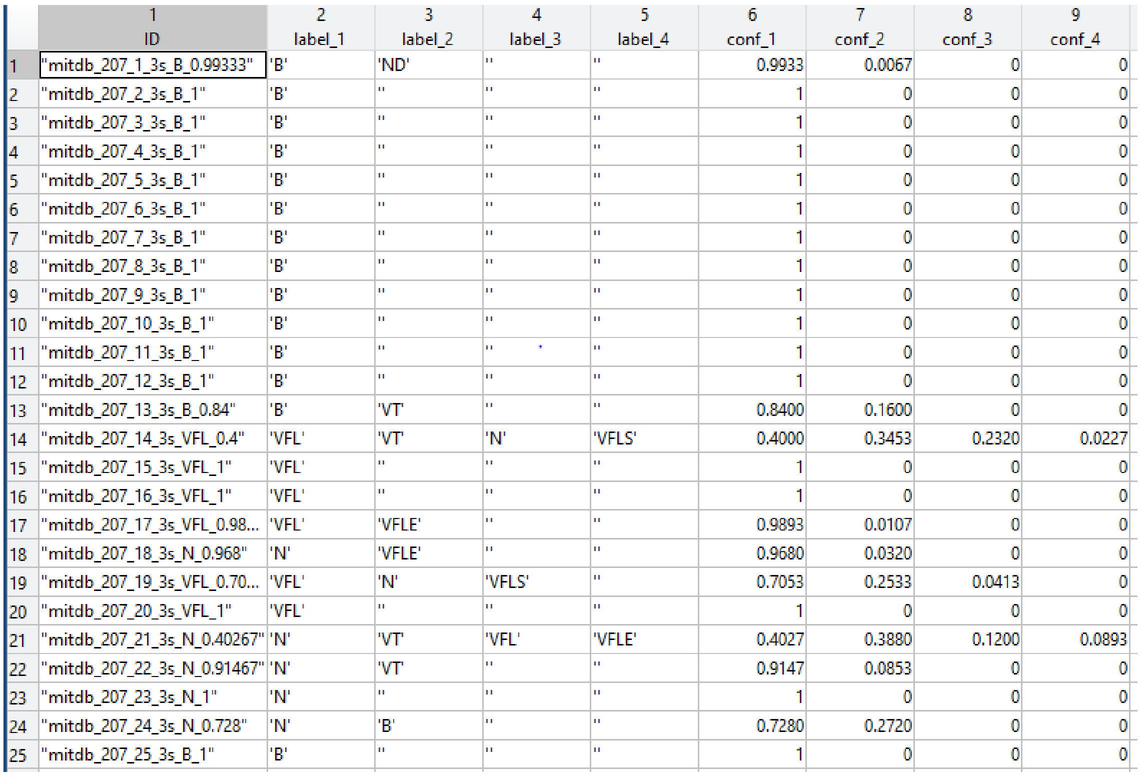
| | 1 timeInSeconds | 2 sampleNumber | 3 typeMnemonic | 4 auxInfo |
|---|---|---|---|---|
| 1 | 0.0250 | 6 | '+' | "B" |
| 2 | 38.5222 | 9631 | '+' | "VT" |
| 3 | 40.0389 | 10010 | '+' | "N" |
| 4 | 40.7361 | 10184 | '[' | "VFLS" |
| 5 | 40.8028 | 10201 | '+' | "VFL" |
| 6 | 50.9722 | 12743 | ']' | "VFLE" |
| 7 | 51.1000 | 12775 | '+' | "N" |
| 8 | 54.7639 | 13691 | '[' | "VFLS" |
| 9 | 54.8889 | 13722 | '+' | "VFL" |
| 10 | 60.3639 | 15091 | ']' | "VFLE" |
| 11 | 60.6306 | 15158 | '+' | "N" |
| 12 | 61.8389 | 15460 | '+' | "VT" |
| 13 | 63.2611 | 15815 | '+' | "N" |
| 14 | 71.1861 | 17797 | '+' | "B" |
| 15 | 75.7111 | 18928 | '+' | "N" |

***Figure 30:*** *Cleaned and resampled extract from MITDBs recording 207 annotation table*

In the next step, a sample label vector is created, which maps the episode annotations to each of the 450000 ECG samples per recording. The "sampleNumber" column in the annotation table serves as an index range to define the label ranges for event episodes in the sample label vector. Undefined samples are marked as "ND".

A previous study on the detection of shockable rhythms with CNNs experimented with segment lengths of 3, 5, 8, and 10 seconds. The approach with 3 s segments achieved the most promising results.[51] The pre-processing revealed that even shorter segments of 3 s can contain several subsegments with different annotations. Since the CNNs are trained based on the most dominant annotation within a segment, this work favors the use of 3 s segments to support the learning process of the models. In a real-world use case, the detection algorithm can be implemented in such a way that it needs to detect a certain number of consecutive shockable events before it advises defibrillation.

According to the segment length of 3 s, the samples and their corresponding labels are aggregated. One segment consists of 750 samples (3 times the sampling frequency) and can contain several annotations. To support further analysis after testing the CNN models, the content of each segment should remain traceable. Therefore, the occurrences of all annotations within a segment are counted, and their proportion is calculated. The segment is assigned the label that gets the majority vote. Consequently, a segment table is created that includes the segment ID followed by the 4 labels with the highest prevalence in the segment and their proportions marked as confidence values. The labels and confidence values within the segment are sorted in descending order. The segment ID contains information about the database, the file and segment numbers, the segment length, the majority label, and the majority label confidence. Shows an extract from the segment table.



| | 1<br>ID | 2<br>label_1 | 3<br>label_2 | 4<br>label_3 | 5<br>label_4 | 6<br>conf_1 | 7<br>conf_2 | 8<br>conf_3 | 9<br>conf_4 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | "mitdb_207_1_3s_B_0.99333" | 'B' | 'ND' | '' | '' | 0.9933 | 0.0067 | 0 | 0 |
| 2 | "mitdb_207_2_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 3 | "mitdb_207_3_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 4 | "mitdb_207_4_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 5 | "mitdb_207_5_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 6 | "mitdb_207_6_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 7 | "mitdb_207_7_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 8 | "mitdb_207_8_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 9 | "mitdb_207_9_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 10 | "mitdb_207_10_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 11 | "mitdb_207_11_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 12 | "mitdb_207_12_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 13 | "mitdb_207_13_3s_B_0.84" | 'B' | 'VT' | '' | '' | 0.8400 | 0.1600 | 0 | 0 |
| 14 | "mitdb_207_14_3s_VFL_0.4" | 'VFL' | 'VT' | 'N' | 'VFLS' | 0.4000 | 0.3453 | 0.2320 | 0.0227 |
| 15 | "mitdb_207_15_3s_VFL_1" | 'VFL' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 16 | "mitdb_207_16_3s_VFL_1" | 'VFL' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 17 | "mitdb_207_17_3s_VFL_0.98... | 'VFL' | 'VFLE' | '' | '' | 0.9893 | 0.0107 | 0 | 0 |
| 18 | "mitdb_207_18_3s_N_0.968" | 'N' | 'VFLE' | '' | '' | 0.9680 | 0.0320 | 0 | 0 |
| 19 | "mitdb_207_19_3s_VFL_0.70... | 'VFL' | 'N' | 'VFLS' | '' | 0.7053 | 0.2533 | 0.0413 | 0 |
| 20 | "mitdb_207_20_3s_VFL_1" | 'VFL' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 21 | "mitdb_207_21_3s_N_0.40267" | 'N' | 'VT' | 'VFL' | 'VFLE' | 0.4027 | 0.3880 | 0.1200 | 0.0893 |
| 22 | "mitdb_207_22_3s_N_0.91467" | 'N' | 'VT' | '' | '' | 0.9147 | 0.0853 | 0 | 0 |
| 23 | "mitdb_207_23_3s_N_1" | 'N' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 24 | "mitdb_207_24_3s_N_0.728" | 'N' | 'B' | '' | '' | 0.7280 | 0.2720 | 0 | 0 |
| 25 | "mitdb_207_25_3s_B_1" | 'B' | '' | '' | '' | 1 | 0 | 0 | 0 |

***Figure 31:*** *Extract of the segment table derived from MITDBs recording 207*

### 3.2.5 Augmentation

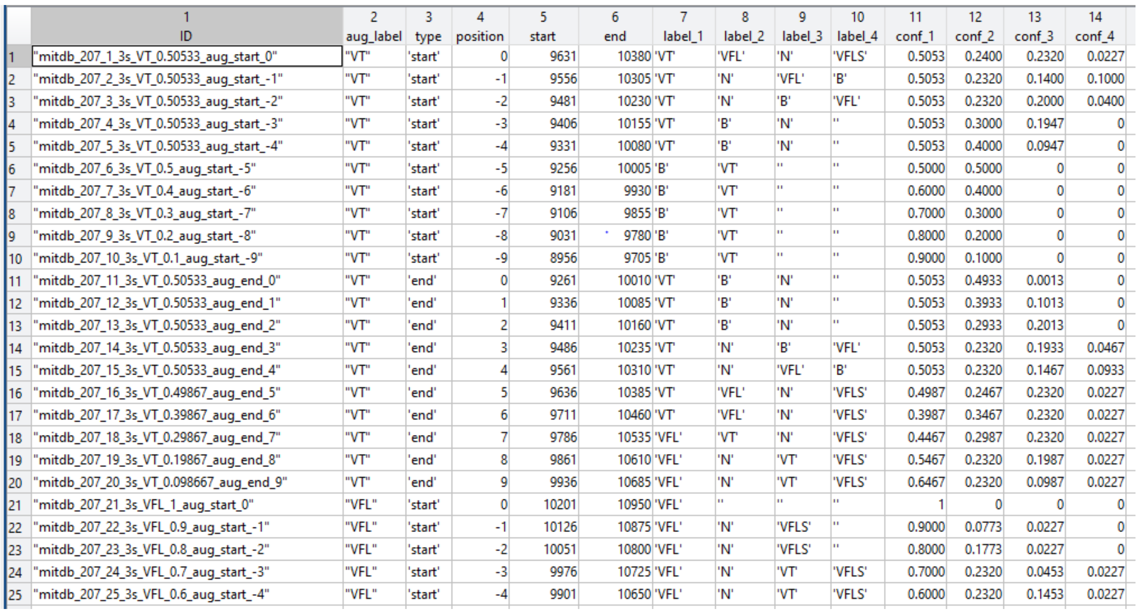
Table 3 summarizes the number of segments that could be obtained from processing the annotations. To support the learning process, a threshold for the majority label is implemented. Segments with a majority label below a 60% proportion of the segment are excluded from further processing. Some segments from VFDB are labeled NOISE and ASYS. These samples, representing distorted ECG signals and flatline signals, are removed as well. Eventually, the dataset provides 59469 segments of 3 s length. 86.86% of the segments are labeled as non-shockable (NShr). The subsets in the shockable conditions show that VFL is represented by only 0.42% of the data, VT by 3.03%, VF by 3.42%, and VF-VFL by 6.32%. The annotation process reveals that the numbers of shockable and non-shockable samples are quite imbalanced.

***Table 3.*** *Summary of segments derived from arrhythmia databases with NShr for non-shockable and Shr for shockable conditions; if indicated with Thr>=0.6, all samples with a vote for the majority label smaller than 0.6 are discarded; *: in VFDB, segments with a majority vote for NOISE and ASYS are discarded*

| Database | Segments | NShr | Shr | | | | |
|---|---|---|---|---|---|---|---|
| | | | VF | VT | VFL | VF-VFL | Total |
| MITDB | 28848 | | | | | | |
| MITDB (Thr>=0.6) | 28594 | 28497 | | 51 | 46 | | 97 |
| AHADB | 12000 | | | | | | |
| AHADB (Thr>=0.6) | 11993 | 8234 | | | | 3759 | 3759 |
| CUDB | 5915 | | | | | | |
| CUDB (Thr>=0.6) | 5882 | 4632 | 1241 | 9 | | | 1250 |
| VFDB | 15400 | | | | | | |
| VFDB (Thr>=0.6)* | 13000 | 10289 | 769 | 1741 | 201 | | 2711 |
| Total | 62163 59469 | **51652** | 2010 | 1801 | 247 | 3759 | **7817** |
| Proportion | **100.00%** | **86.86%** | 3.38% | 3.03% | 0.42% | 6.32% | **13.14%** |

As described earlier, it is important to aim for a relatively balanced dataset to support the training process of the CNNs. Therefore, two strategies to enrich the subsets of shockable conditions are applied as a form of dataset augmentation. The first strategy focuses on deriving segments around the borders of shockable episodes. In addition to generating shockable labeled samples, the CNNs will receive more training data around episode borders. In a way, the models are more challenged to learn features that are relevant around episode borders to detect the right condition. The second strategy derives segments along shockable episodes.

The code for the first augmentation strategy can be found in Figure 48 in the appendix. First, a table is created that contains the sample numbers for the start and end of each shockable rhythm episode. Then, a shifting segment window with a 1/10 step size of the segment length is implemented. When reaching the starting point of a shockable episode, the shifting segment window moves backward 10 times with the set step size of 75 samples. When reaching the endpoint of a shockable episode, the shifting segment window starts 1 segment length before the endpoint and moves 10 times forward from there, again with the same step size of 75 samples. Consequently, for each border region of a shockable condition, the algorithm creates 10 augmented border segments. The augmented border segments are stored in the border augmentation table (Figure 32), which includes the segment ID (containing database, file number, segment number, segment length, label, label confidence, aug, start/end, Position), the type, the position, starting and ending sample number, and the 4 most prevalent labels of the segment and their confidence values. The type indicates if the segment was derived from the beginning or end of a shockable episode. The position is given by an integer between -9 and +9, with negative numbers indicating going backwards and positive numbers indicating going forward.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ID | aug_label | type | position | start | end | label_1 | label_2 | label_3 | label_4 | conf_1 | conf_2 | conf_3 | conf_4 |
| 1 | "mitdb_207_1_3s_VT_0.50533_aug_start_0" | "VT" | 'start' | 0 | 9631 | 10380 | 'VT' | 'VFL' | 'N' | 'VFLS' | 0.5053 | 0.2400 | 0.2320 | 0.0227 |
| 2 | "mitdb_207_2_3s_VT_0.50533_aug_start_-1" | "VT" | 'start' | -1 | 9556 | 10305 | 'VT' | 'N' | 'VFL' | 'B' | 0.5053 | 0.2320 | 0.1400 | 0.1000 |
| 3 | "mitdb_207_3_3s_VT_0.50533_aug_start_-2" | "VT" | 'start' | -2 | 9481 | 10230 | 'VT' | 'N' | 'B' | 'VFL' | 0.5053 | 0.2320 | 0.2000 | 0.0400 |
| 4 | "mitdb_207_4_3s_VT_0.50533_aug_start_-3" | "VT" | 'start' | -3 | 9406 | 10155 | 'VT' | 'B' | 'N' | '' | 0.5053 | 0.3000 | 0.1947 | 0 |
| 5 | "mitdb_207_5_3s_VT_0.50533_aug_start_-4" | "VT" | 'start' | -4 | 9331 | 10080 | 'VT' | 'B' | 'N' | '' | 0.5053 | 0.4000 | 0.0947 | 0 |
| 6 | "mitdb_207_6_3s_VT_0.5_aug_start_-5" | "VT" | 'start' | -5 | 9256 | 10005 | 'B' | 'VT' | '' | '' | 0.5000 | 0.5000 | 0 | 0 |
| 7 | "mitdb_207_7_3s_VT_0.4_aug_start_-6" | "VT" | 'start' | -6 | 9181 | 9930 | 'B' | 'VT' | '' | '' | 0.6000 | 0.4000 | 0 | 0 |
| 8 | "mitdb_207_8_3s_VT_0.3_aug_start_-7" | "VT" | 'start' | -7 | 9106 | 9855 | 'B' | 'VT' | '' | '' | 0.7000 | 0.3000 | 0 | 0 |
| 9 | "mitdb_207_9_3s_VT_0.2_aug_start_-8" | "VT" | 'start' | -8 | 9031 | 9780 | 'B' | 'VT' | '' | '' | 0.8000 | 0.2000 | 0 | 0 |
| 10 | "mitdb_207_10_3s_VT_0.1_aug_start_-9" | "VT" | 'start' | -9 | 8956 | 9705 | 'B' | 'VT' | '' | '' | 0.9000 | 0.1000 | 0 | 0 |
| 11 | "mitdb_207_11_3s_VT_0.50533_aug_end_0" | "VT" | 'end' | 0 | 9261 | 10010 | 'VT' | 'B' | 'N' | '' | 0.5053 | 0.4933 | 0.0013 | 0 |
| 12 | "mitdb_207_12_3s_VT_0.50533_aug_end_1" | "VT" | 'end' | 1 | 9336 | 10085 | 'VT' | 'B' | 'N' | '' | 0.5053 | 0.3933 | 0.1013 | 0 |
| 13 | "mitdb_207_13_3s_VT_0.50533_aug_end_2" | "VT" | 'end' | 2 | 9411 | 10160 | 'VT' | 'B' | 'N' | '' | 0.5053 | 0.2933 | 0.2013 | 0 |
| 14 | "mitdb_207_14_3s_VT_0.50533_aug_end_3" | "VT" | 'end' | 3 | 9486 | 10235 | 'VT' | 'N' | 'B' | 'VFL' | 0.5053 | 0.2320 | 0.1933 | 0.0467 |
| 15 | "mitdb_207_15_3s_VT_0.50533_aug_end_4" | "VT" | 'end' | 4 | 9561 | 10310 | 'VT' | 'N' | 'VFL' | 'B' | 0.5053 | 0.2320 | 0.1467 | 0.0933 |
| 16 | "mitdb_207_16_3s_VT_0.49867_aug_end_5" | "VT" | 'end' | 5 | 9636 | 10385 | 'VT' | 'VFL' | 'N' | 'VFLS' | 0.4987 | 0.2467 | 0.2320 | 0.0227 |
| 17 | "mitdb_207_17_3s_VT_0.39867_aug_end_6" | "VT" | 'end' | 6 | 9711 | 10460 | 'VT' | 'VFL' | 'N' | 'VFLS' | 0.3987 | 0.3467 | 0.2320 | 0.0227 |
| 18 | "mitdb_207_18_3s_VT_0.29867_aug_end_7" | "VT" | 'end' | 7 | 9786 | 10535 | 'VFL' | 'VT' | 'N' | 'VFLS' | 0.4467 | 0.2987 | 0.2320 | 0.0227 |
| 19 | "mitdb_207_19_3s_VT_0.19867_aug_end_8" | "VT" | 'end' | 8 | 9861 | 10610 | 'VFL' | 'N' | 'VT' | 'VFLS' | 0.5467 | 0.2320 | 0.1987 | 0.0227 |
| 20 | "mitdb_207_20_3s_VT_0.098667_aug_end_9" | "VT" | 'end' | 9 | 9936 | 10685 | 'VFL' | 'N' | 'VT' | 'VFLS' | 0.6467 | 0.2320 | 0.0987 | 0.0227 |
| 21 | "mitdb_207_21_3s_VFL_1_aug_start_0" | "VFL" | 'start' | 0 | 10201 | 10950 | 'VFL' | '' | '' | '' | 1 | 0 | 0 | 0 |
| 22 | "mitdb_207_22_3s_VFL_0.9_aug_start_-1" | "VFL" | 'start' | -1 | 10126 | 10875 | 'VFL' | 'N' | 'VFLS' | '' | 0.9000 | 0.0773 | 0.0227 | 0 |
| 23 | "mitdb_207_23_3s_VFL_0.8_aug_start_-2" | "VFL" | 'start' | -2 | 10051 | 10800 | 'VFL' | 'N' | 'VFLS' | '' | 0.8000 | 0.1773 | 0.0227 | 0 |
| 24 | "mitdb_207_24_3s_VFL_0.7_aug_start_-3" | "VFL" | 'start' | -3 | 9976 | 10725 | 'VFL' | 'N' | 'VT' | 'VFLS' | 0.7000 | 0.2320 | 0.0453 | 0.0227 |
| 25 | "mitdb_207_25_3s_VFL_0.6_aug_start_-4" | "VFL" | 'start' | -4 | 9901 | 10650 | 'VFL' | 'N' | 'VT' | 'VFLS' | 0.6000 | 0.2320 | 0.1453 | 0.0227 |

**Figure 32:** *Extract of the border augmentation table from MITDBs recording 207*

The second augmentation approach looks for starting points of shockable episodes and measures the length of the episode. If the shockable condition exceeds 1 segment length, a shifting segment window is applied at the starting point of the episode. The shifting window moves forward with a step size of 1/10 of the 3 s segment length (75 samples) while capturing segments along the shockable episode. When the window reaches the end of the episode, the process is terminated. An episode augmentation table is generated similar to the border augmentation table as illustrated in Figure 32. The code for the augmentation along shockable episodes can be followed on Figure 49 in the appendix.

Eventually, two selection criteria are applied. First, a threshold (Thr>=0.6) is applied that excludes augmented segments with a majority label proportion of less than 60%, ensuring that a segment fed to and evaluated by the CNN primarily belongs to either the shockable or the not shockable class. Second, all augmented segments that overlap below 75 samples with other augmented or original segments are excluded from further processing. This means that the final dataset includes only segments with non-overlapping areas of at least 10%.

Table 4 summarizes the number of augmented segments for each of the arrhythmia databases as well as the aggregation of augmented segments. Throughout the augmentation process a total number of 2527 augmented segments around shockable episode borders and 74036 augmented segments along shockable episodes are generated. Without augmentation only 7817 shockable segments are present in the dataset. In the next Section, the segments, originals, and augmentations, are used to generate balanced datasets for training, validation, and testing of the CNN models.

***Table 4.*** *Summary of shockable segments derived from arrhythmia databases including augmented segments around shockable episode borders and along shockable episodes; Thr>=0.6, all samples with a vote for the majority label smaller than 0.6 are discarded*

| Database | Shr segments | | | | |
|---|---|---|---|---|---|
| | VF | VT | VFL | VF-VFL | Total |
| **MITDB (Thr>=0.6)** | | 51 | 46 | | 97 |
| Border augmentation | | 278 | 54 | | |
| Episode augmentation | | 319 | 410 | | |
| **AHADB (Thr>=0.6)** | | | | 3759 | 3759 |
| Border augmentation | | | | 124 | |
| Episode augmentation | | | | 37412 | |
| **CUDB (Thr>=0.6)** | 1241 | 9 | | | 1250 |
| Border augmentation | 358 | 54 | | | |
| Episode augmentation | 12154 | 49 | | | |
| **VFDB (Thr>=0.6)** | 769 | 1741 | 201 | | 2711 |
| Border augmentation | 77 | 978 | 604 | | |
| Episode augmentation | 5536 | 16593 | 1563 | | |
| Total without augmented | 2010 | 1801 | 247 | 3759 | **7817** |
| Total augmented border | 435 | 1310 | 658 | 124 | **2527** |
| Total augmented episode | 17690 | 16961 | 1973 | 37412 | **74036** |

### 3.2.6 Dataset management and storing

The final step of the pre-processing in MATLAB is to create spectrogram segments derived from the annotation table, the border augmentation table, and the episode augmentation table. The tables include the start and end points of each segment, which are used to index the cutting points for the spectrogram segments. All the described segments are cropped from the spectrogram representations for each of the ECG recordings and stored as matrices in mat file format. The MATLAB code can be followed on Figure 50 in the appendix.

Creating balanced datasets for training, validation, and testing is realized with Python. The code can be followed on Figure 51. In total, 60340 segments are randomly selected from the subsets of shockable and non-shockable segments as listed in Table 5.

***Table 5.*** *Balanced dataset (bottom part) derived from dataset with augmented segments (top part as in **Table 4**); Thr>=0.6, all samples with a vote for the majority label smaller than 0.6 are discarded*

| Database | NShr | Shr segments | | | | Total |
|---|---|---|---|---|---|---|
| | | VF | VT | VFL | VF-VFL | |
| **MITDB (Thr>=0.6)** | 28497 | | 51 | 46 | | 28594 |
| Border augmentation | | | 278 | 54 | | 332 |
| Episode augmentation | | | 319 | 410 | | 729 |
| **AHADB (Thr>=0.6)** | 8234 | | | | 3759 | 11993 |
| Border augmentation | | | | | 124 | 124 |
| Episode augmentation | | | | | 37412 | 37412 |
| **CUDB (Thr>=0.6)** | 4632 | 1241 | 9 | | | 5882 |
| Border augmentation | | 358 | 54 | | | 412 |
| Episode augmentation | | 12154 | 49 | | | 12203 |
| **VFDB (Thr>=0.6)** | 10289 | 769 | 1741 | 201 | | 13000 |
| Border augmentation | | 77 | 978 | 604 | | 1659 |
| Episode augmentation | | 5536 | 16593 | 1563 | | 23692 |
| Total | **51652** | **20135** | **20072** | **2878** | **41295** | **136032** |
| **Balanced dataset** | | | | | | |
| **MITDB (Thr>=0.6)** | 10000 * | | 51 | 46 | | 10097 |
| Border augmentation | | | 278 | 54 | | 332 |
| Episode augmentation | | | 319 | 410 | | 729 |
| **AHADB (Thr>=0.6)** | 8234 | | | | 3759 | 11993 |
| Border augmentation | | | | | 124 | 124 |
| Episode augmentation | | | | | 4000 * | 4000 |
| **CUDB (Thr>=0.6)** | 4632 | 1241 | 9 | | | 5882 |
| Border augmentation | | 358 | 54 | | | 412 |
| Episode augmentation | | 3000 * | 49 | | | 3049 |
| **VFDB (Thr>=0.6)** | 10289 | 769 | 1741 | 201 | | 13000 |
| Border augmentation | | 77 | 978 | 604 | | 1659 |
| Episode augmentation | | 2500 * | 5000 * | 1563 | | 9063 |
| Total | **33155** | **7945** | **8479** | **2878** | **7883** | **60340** |
| Proportion | **54.95%** | **13.17%** | **14.05%** | **4.77%** | **13.06%** | **100.00%** |
| Train (70%) | 23209 | 5562 | 5935 | 2015 | 5518 | 42239 |
| Validation/Test (15% each) | 4973 | 1192 | 1272 | 432 | 1182 | 9051 |

* Reduced number of segments randomly chosen from regarding subsets of NShr segments and augmented Shr segments

From the 28497 non-shockable segments in the MITDB dataset, 10000 are randomly selected for the final dataset. All other non-shockable segments from the other databases are kept entirely, which sums up to 33155 non-shockable segments in total and accounts for 54.95% of the final dataset. For shockable segments, several subsets of augmentations along shockable episodes are narrowed down. Regarding VF conditions, the 12154 segments from CUDB are reduced to 3000, and from the 5536 segments from VFDB, 2500 remain. In total, the VF segments make up 13.17% of the final dataset. The VT segments from VFDB are reduced from 16593 to 5000 segments, which equates to 14.05% of VT segments in the final set. All VFL segments, including augmented ones, are kept, which accounts for 4.77% of the total. The VF-VFL segments, which are only present in the AHADB, are narrowed down as well. From 37412 augmentation segments along shockable episodes, 4000 are randomly selected. Therefore, VF-VFL segments account for 13.06% of the final dataset. Combined, the shockable segments in the balanced dataset reach 45.05% of all segments.

The balanced subsets are chosen randomly but equally distributed among the training, validation, and test sets. The training set consists of 42239 segments, which equals 70% of the balanced dataset. Test- and validation set contain 9051 segments each, which contributes 15% for each set to the balanced dataset. The algorithm that creates the final datasets implements a random seed number that remains the same for the segment selection throughout the whole procedure. The random seed is changed when generating several balanced datasets for cross-validation purposes.

## 3.3 CNN Models

Two main model architectures are considered and examined for this work. The conventional approach, where the convolutional block is followed by a block of dense layers, and the residual CNN (both concepts are explained in Section 2.4). The aim is to maximize the model's accuracy in distinguishing between shockable and non-shockable heart rhythms, regardless of the extent of model complexity and computational demand. The approach follows and examines the assumption that deeper CNNs provide a higher degree of flexibility in adapting to the problem and therefore are more competitive in solving it.

4 conventional CNN architectures are implemented, starting at relatively low-level model complexity, which is incrementally enhanced by adding convolutional layers, increasing the number of convolutional filters, and adding dense layers.

The residual CNN models (ResNets) are introduced to increase the depth of the net and likewise avoid the problem of vanishing gradients by allowing input data to bypass layers. In that sense, the learning ability of the network is preserved by residual blocks, which can lead to a better performance in solving the problem. The ResNets consist of 3 stages, each including several residual blocks. The depth of each stage is defined by the number of residual blocks, which vary from 6 to 96. The structure of the residual block is illustrated in Figure 33. It is derived from Kaiming He’s second publication on ResNets, which introduced improvements to the residual block [82].

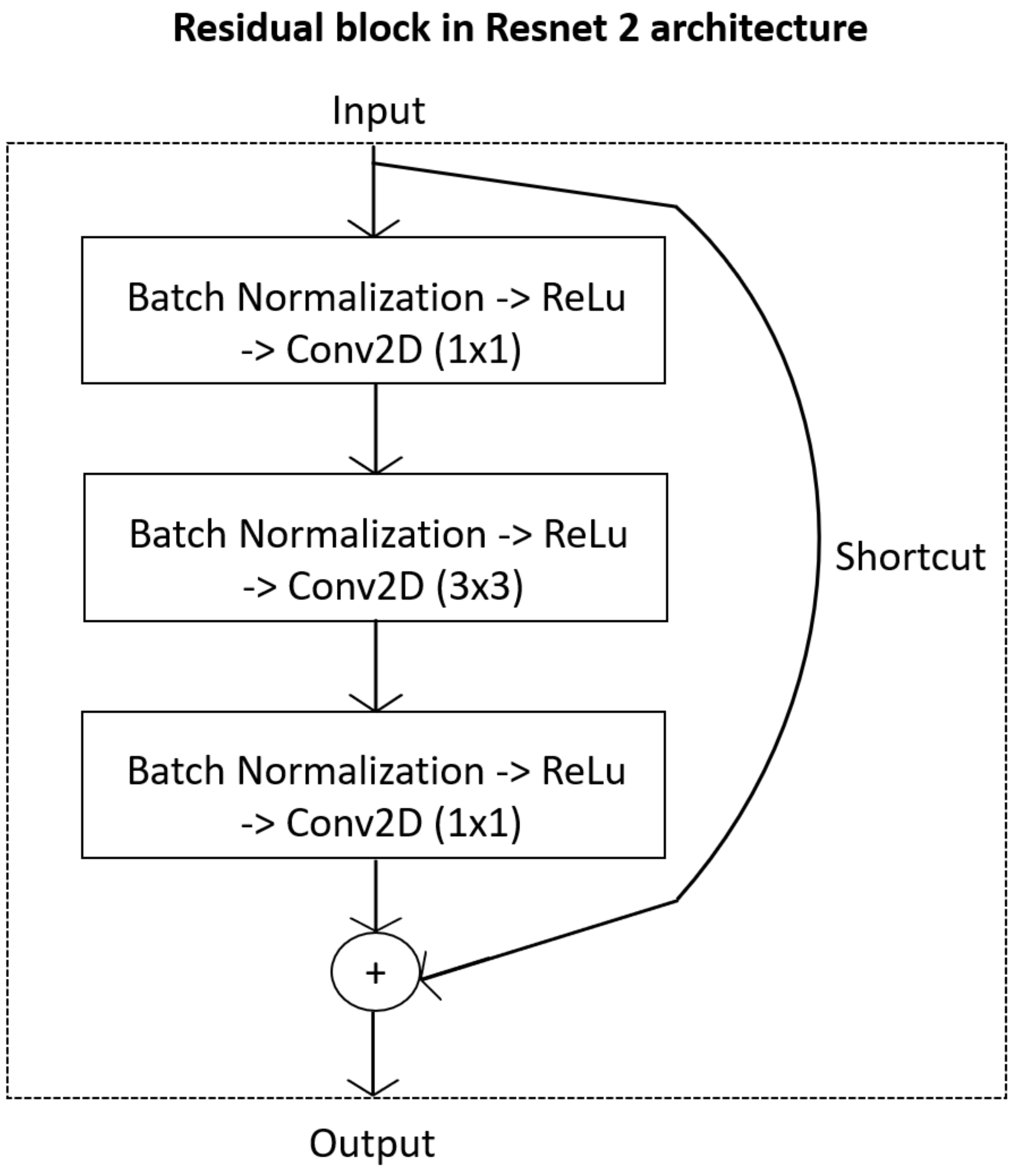

Residual block in Resnet 2 architecture
Input
Batch Normalization -> ReLu -> Conv2D (1x1)
Batch Normalization -> ReLu -> Conv2D (3x3)
Batch Normalization -> ReLu -> Conv2D (1x1)
Shortcut
+
Output

***Figure 33:*** *Residual block in ResNet 2 architecture with layers containing batch normalization, ReLu activation and convolution*

The residual block consists of 3 convolutional layers. Each layer begins with batch normalization followed by activation with ReLu. The shortcut connection bypasses the 3 convolutional layers and adds the input of the residual block to its output. The first and third convolutional layers of the block have a kernel size of 1x1 and are implemented to control the dimensionality of the output of the residual block. This regards the number of feature maps, which is controlled by the number of convolutional filters and by the stride parameter, which influences the size of the feature map. A stride parameter set to 2 downsamples the input to half its size and then works as a pooling layer. Since the 1x1 kernel contains a weight that is multiplied with the layer input, the 1x1 convolutional layer serves as a feature refinement before the data enters the next layer. The middle layer of the residual block has a 3x3 filter kernel and therefore actually performs the convolution in the block.

Figure 34 sketches a blueprint of the ResNet architecture. The number of residual blocks within a stage is variable and indicated from 1-n.

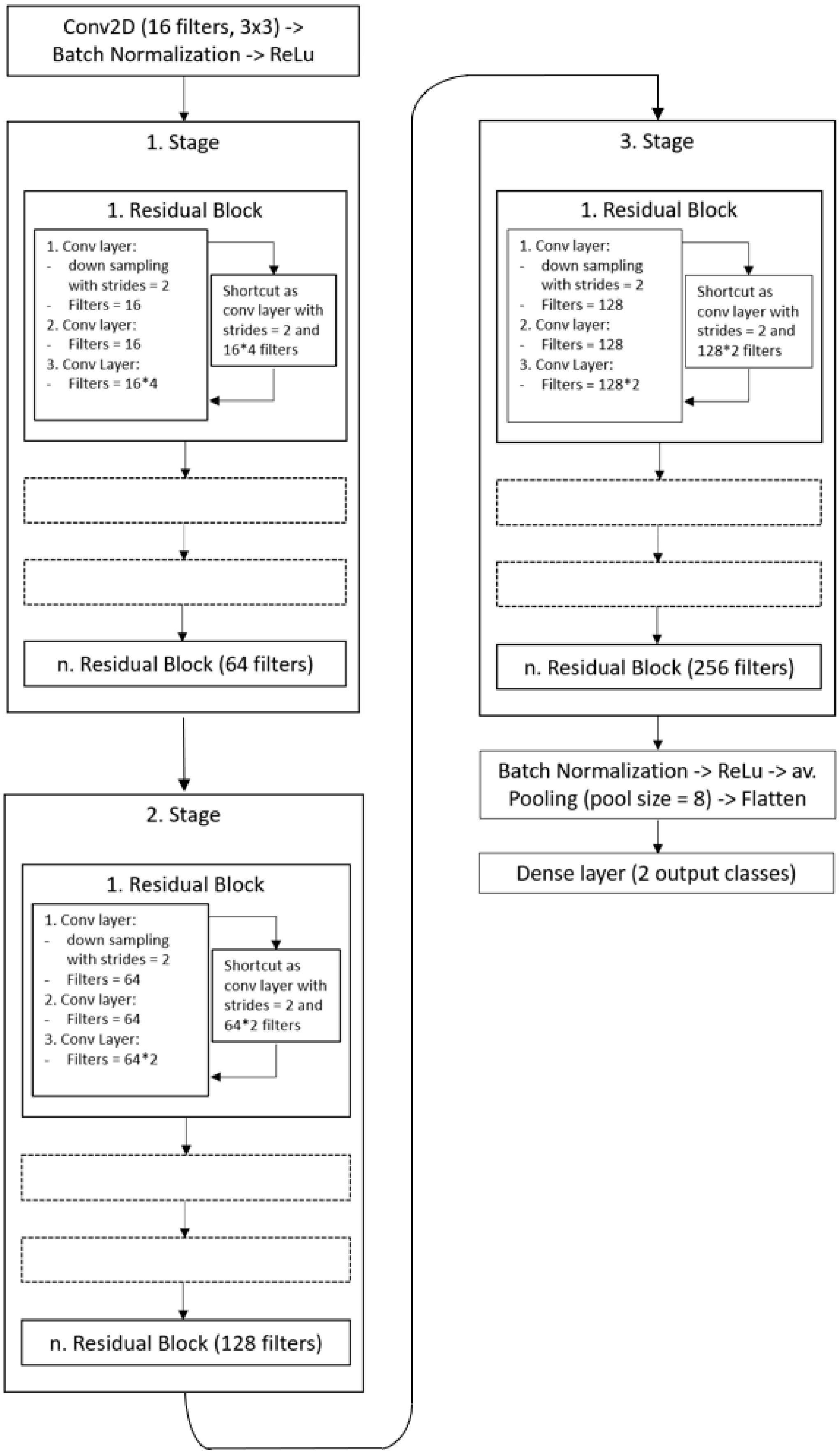

Conv2D (16 filters, 3x3) -> Batch Normalization -> ReLu
1. Stage
1. Residual Block
1. Conv layer:
- down sampling with strides = 2
- Filters = 16
2. Conv layer:
- Filters = 16
3. Conv Layer:
- Filters = 16*4
Shortcut as conv layer with strides = 2 and 16*4 filters
n. Residual Block (64 filters)
2. Stage
1. Residual Block
1. Conv layer:
- down sampling with strides = 2
- Filters = 64
2. Conv layer:
- Filters = 64
3. Conv Layer:
- Filters = 64*2
Shortcut as conv layer with strides = 2 and 64*2 filters
n. Residual Block (128 filters)
3. Stage
1. Residual Block
1. Conv layer:
- down sampling with strides = 2
- Filters = 128
2. Conv layer:
- Filters = 128
3. Conv Layer:
- Filters = 128*2
Shortcut as conv layer with strides = 2 and 128*2 filters
n. Residual Block (256 filters)
Batch Normalization -> ReLu -> av. Pooling (pool size = 8) -> Flatten
Dense layer (2 output classes)

***Figure 34:*** *ResNet architecture with 3 stages and n residual blocks at each stage*

Before the input enters the first stage of the ResNet, the data is convoluted in a 3x3 convolutional layer with 16 filters applied. In contradiction to the residual block, batch normalization and ReLu activation are performed after the convolution. When the data is forwarded to the first residual block of each stage, the feature maps are down sampled to half size by setting stride equal to 2. Before exiting the first residual block, the 3. convolutional layer (1x1) doubles the number of feature maps. In the first stage, the feature maps are quadrupled. To match the changing dimensionality within the first residual blocks, the input data entering the shortcut connection goes through a 1x1 convolutional layer that adjusts the number and size of the feature maps. All the following residual blocks within each stage preserve the dimensionality of the feature maps. By initializing the filter size at 16, the first stage produces 64 feature maps, the second, 128 maps, and the third, 256 maps.

All models are realized using the TensorFlow Keras library in Python. Regardless of the applied model, the greyscale spectrogram segments are rescaled to a range between 0 and 1 as described in Section 2.4.3, which is realized by applying Keras rescaling layer.

Likewise applied in all models is ReLu as the activation function of choice. The reasons for this are described in Section 2.4.2. Excepted are the classifying output layers. The activation functions in the output layer are identity for the conventional CNNs and sigmoid for the ResNets. Since ReLu is used, the weights in all models are initialized with the He initialization. With this method, the weights are randomly initialized from a Gaussian distribution. Then, regarding the current layer in the network, the weights are scaled with respect to the number of inputs from the previous layer. This is realized by multiplying the random value by the square root of 2 over the number of inputs from the previous layer. The specialty of He initialization is the value 2 within the square root, which is used instead of 1 to calculate the standard deviation. Taking the number of inputs from the previous layer into account when initializing weights proved to result in better-performing models with a faster learning process [69] [p 437].

To avoid frequent shifting between Chapters, the specific model configurations are described along with the model performances in the results and discussion part (see Section 4).

## 3.4 Hyperparameters in CNNs

Depending on the model architecture and the chosen optimization algorithms, the performance of a model depends on the setting of adjustable hyperparameters. This includes learning rate, momentum, regularization, batch size, number of epochs, and weight initialization. The individual hyperparameter setting for each model is presented in the results part (see Section 4.1) a long with the model performances.

### 3.4.1 Optimization algorithm and learning rate

The optimization algorithm used for the CNN models is Adam, which is derived from adaptive moment estimation. Adam is based on stochastic gradient descent and is widely used in deep learning applications. It combines the concepts of momentum optimization and RMSProp (root mean squared propagation), which both provide Adam with an adaptive learning rate. The adaptive learning rate comes from taking the behavior of weights during previous iterations into account.

Regarding regular gradient descent (see Section 2.4.3), updating weights depends on the learning rate and the current gradient. For small local gradients, the process goes slowly. As for the inertia moment in physics, the current state of an object in motion and how this state can be changed depend on its momentum, or the amount of kinetic energy it has accumulated. Analogously, the momentum optimization considers the size of previous gradients, which, so to speak, adjust the velocity of the system and define the momentum vector m. The momentum vector is also influenced by the learning rate and the hyperparameter β, which is called the momentum. Momentum optimization can be described as follows:

$$m := \beta m - \epsilon \frac{\partial C}{\partial w} \tag{4}$$

$$w := w + m \tag{5}$$

First, the momentum vector is updated by multiplying β and subtracting the current gradient multiplied by the learning rate ϵ. Eventually, the momentum vector is added to the previous weights, defining the new weight values. The hyperparameter β can be set between 0 and 1. It regulates how strong previous gradients influences are. If β is set to 0 the method is the same as for regular gradient descent.

For all CNN models, β is set to 0.9, which is widely used as default for most CNN applications. Compared to regular gradient descent, momentum optimization allows faster computation and efficiently escaping local optima [69] [p. 461].

RMSProp adapts the learning by implementing an exponential decay of the influence of previous gradients on the learning rate. The weightage of previous gradients on updating the weights falls exponentially the farther the gradients are from the current step. In contrast to the original RMSProp algorithm, Adam computes an exponentially decaying average of previous gradients rather than a sum. The hyperparameter $\beta_2$ is set to 0.999, which is the default value that serves well for most applications [69] [p. 467].

Since the ability of the Adam algorithm to adjust the step size of updating weights based on previous gradients, the learning rate for all conventional CNN models is set to 0.001.

For the deeper ResNet approaches, the learning rate is either fixed, with values between 0.0001 and 0.00001, or scheduled. The scheduled setting starts with a relatively large learning rate of 0.01 and incrementally decreases to 0.0005 with respect to the number of training epochs. Higher learning rates aim to accelerate the training process in the beginning and should prevent the model from getting stuck in local minima. During later epochs, the learning rate is decreased to fine-tune the weights.

### 3.4.2 Batch size

The batch size defines the number of training samples that are used to calculate the average gradient during backpropagation for each weight and bias in the network before updating the weights. The content of the training samples within the batch has a direct influence on the weight adjustment. For example, if one label (object type) is too dominant in the batch, the optimization drives towards this label without considering other labels when adjusting the weights. Therefore, the training data should be shuffled toward a uniform distribution when considering balanced label sets).

The batch size influences the speed and stability of the learning process. The more training samples are used to update the weights, or so to speak, the larger the batch size, the more accurate the estimation will be, and therefore the weight adjustment will be more likely to improve the model's performance. On the other hand, the larger the batch size, the greater the computational effort required to train the network, as more predictions must be made before calculating the estimate and updating the weights. Smaller batch sizes tend to produce noisy estimates, meaning that many updates with quite different error gradients are produced. This provides a regularizing effect and lowers the generalization error, which can lead to more robustness in the model and a faster learning rate [67] [p. 278]. Further, smaller batch sizes ensure that the batch data fits into memory, especially when using a GPU for training. Batch sizes are usually chosen between 1 and a few hundreds. Across a wide range of experiments with deep neural networks, it has been confirmed that batch sizes of 32 or smaller achieve the best training stability and generalization performance for a given computational cost [102].

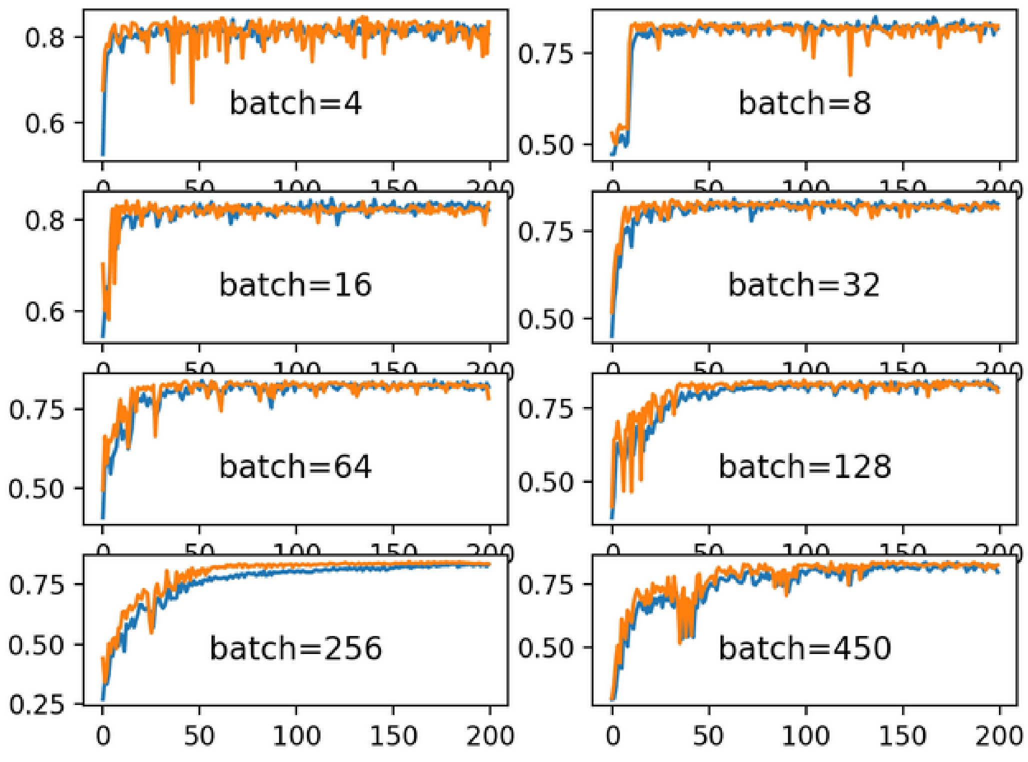


***Figure 35:*** *learning curves of different batch sizes of a MLP with 1 hidden layer and 50 neurons on sklearn.make_blobs() dataset (blue lines = training data, orange lines = test data* ***[103]***

Figure 35 illustrates the influence of the batch size by plotting learning curves based on different batch sizes. Learning curves show classification accuracy on training and test data over a certain number of epochs. Learning curves reveal insights on the performance of the learning process in terms of learning speed, accuracy, and the noise of weight updates. Train and test data are generated with scikit-learn's make_blobs() function to randomly create 1000 points of 3 classes in 2D space. The dataset is equally divided into 500 training samples and 500 test samples. The neural network consists of one hidden layer with 50 neurons. The neurons are activated with ReLU, and the weights are initialized randomly with the Keras He uniform function [103]. The learning curves indicate that the network learns faster with smaller batch sizes, but the learning process tends to imply higher variances in classification accuracy. The larger batch sizes result in slower learning progress but lead to a stable convergence of classification accuracy later on in the training process. As mentioned earlier, a batch size of 32 seems to be a good trade-off between stability, convergence, and computational workload, and is likewise the dominant batch size setting for the models in this work.

### 3.4.3 Regularization

Regularization is implemented to counter the process of overfitting the model function to the training data, which might result in poor model performance when confronted with unseen test data. In other words, regularization aims for good generalization performance on real-world test data. Figure 36 illustrates the problem of a function that aims to correctly classify every outlier in the training set.

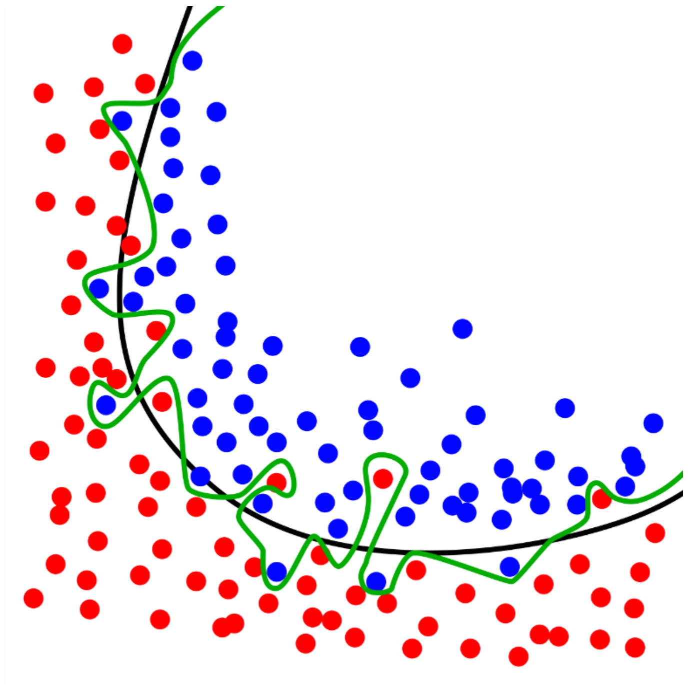


***Figure 36:*** *2 Functions separating datapoints: green line resembles overfitting model, while black line resembles regularized model* ***[104]***

The applied regularization methods are L2 regularization and dropout for the conventional CNN models. Additionally, gradient value clipping is introduced for the ResNet models.

**L2 regularization** or Ridge regression adds a penalty term to the cost function (introduced in Section 2.4.3). The penalty term consists of the sum of squared weights, which is scaled by the hyperparameter λ. In that sense, higher values of λ mean a stronger influence of the penalty term on the cost function and therefore on the process of updating the weights with gradient descent.

L2 regularization is implemented in all ResNet models, with λ set to 0.0001. Regarding the conventional CNNs, l2 regularization occurs in model CNN4 with λ set to 0.001.

Another regularization method which is applied to the conventional CNN models is **dropout**. A defined share of the neurons is randomly and temporarily excluded from the training. The hyperparameter *p* determines the probability at which a neuron is dropped. *p* can take values between 0 and 1. Temporary exclusion means that a dropout state lasts only for one training step, which is defined by the batch size. For the following batch, a new configuration of dropped-out neurons is randomly calculated, and so on. When the training is completed and the model is validated, all neurons are active.

Dropout is used in models CNN2, CNN3, CNN3BN, and CNN4 with values of *p* between 0.2 and 0.28. On which layers dropout is applied is discussed in the results part (Section 4.1).

**Max-norm regularization** is used in CNN3, CNN3BN, and CNN4. It reduces overfitting by constraining the l2 norm of the weight matrix to be smaller or equal to a certain value, which is defined as the max-norm hyperparameter. Regarding this rule, the weights are rescaled if needed. Smaller values of the max-norm hyperparameter increase the effect of regularization on the model [69] [p. 486].

For deeper nets, such as the ResNet models, **gradient value clipping** is introduced. The hyperparameter clipvalue defines a threshold at which gradients are truncated. Every gradient exceeding the threshold is clipped to the value of the threshold. This regularizing method stabilizes the model and counters the problem of exploding gradients. Clipvalue is applied to the ResNet models and is set to values between 0.5 and 0.6.

### 3.4.4 Epochs

The number of epochs determines the number of iterations in which the whole training set is looped through the model. By implementing EarlyStopping from tensorflow.keras.callbacks, the training process stops when no sufficient improvement in the accuracy metric is recognized over a certain number of iterations. The two parameters to be considered are min_delta and patience. Min_delta is set to 0.0001 and patience is set to 15. The training stops when the accuracy doesn't improve by 0.0001 points within 15 epochs.

## 3.5 Model evaluation

The model's performances are evaluated on test data sets by the metrics accuracy, specificity, sensitivity or recall, precision, and AUC score (area under the receiver operating characteristic curve). These metrics are based on 4 fundamental classification scales:

True Positives (TP) – is the number of correctly classified shockable segments.

False Positives (FP) – refers to the number of segments that are non-shockable but falsely predicted as shockable.

True Negatives (TN) – is the number of correctly classified non-shockable segments.

False Negatives (FN) – refers to the number of segments that are shockable but wrongly classified as non-shockable.

**Accuracy** is defined as the ratio of all correctly predicted segments over all predicted values of the test set.

**Specificity** is the ratio of the correctly predicted non-shockable segments over all non-shockable segments:

$$Specificity = \frac{TN}{TN + FP}$$

***6***

**Sensitivity** or Recall is the ratio of correctly predicted shockable segments over all shockable segments:

$$Sensitivity = \frac{TP}{TP + FN}$$

***7***

**Precision** is the ratio of correctly predicted shockable segments over all as shockable predicted segments:

$$Precision = \frac{TP}{TP + FP}$$

***8***

The **AUC score** is based on the ROC curve (receiver operating characteristic curve), which plots the false positive rate or the probability of false alarm over the sensitivity or probability of detection at varying classification thresholds. Regarding a binary classification, the classification threshold determines at what probability (a value between 0 and 1) the model classifies samples as positive. Additionally, the model classifies probabilities below the threshold as negative. As the threshold is decreased, the model gets more sensitive for positive samples (e.g., shockable rhythms), but likewise, it becomes more prone to raise false alarm. The ROC curve captures the variance between sensitivity and false alarms while changing the classification threshold (Figure 37).

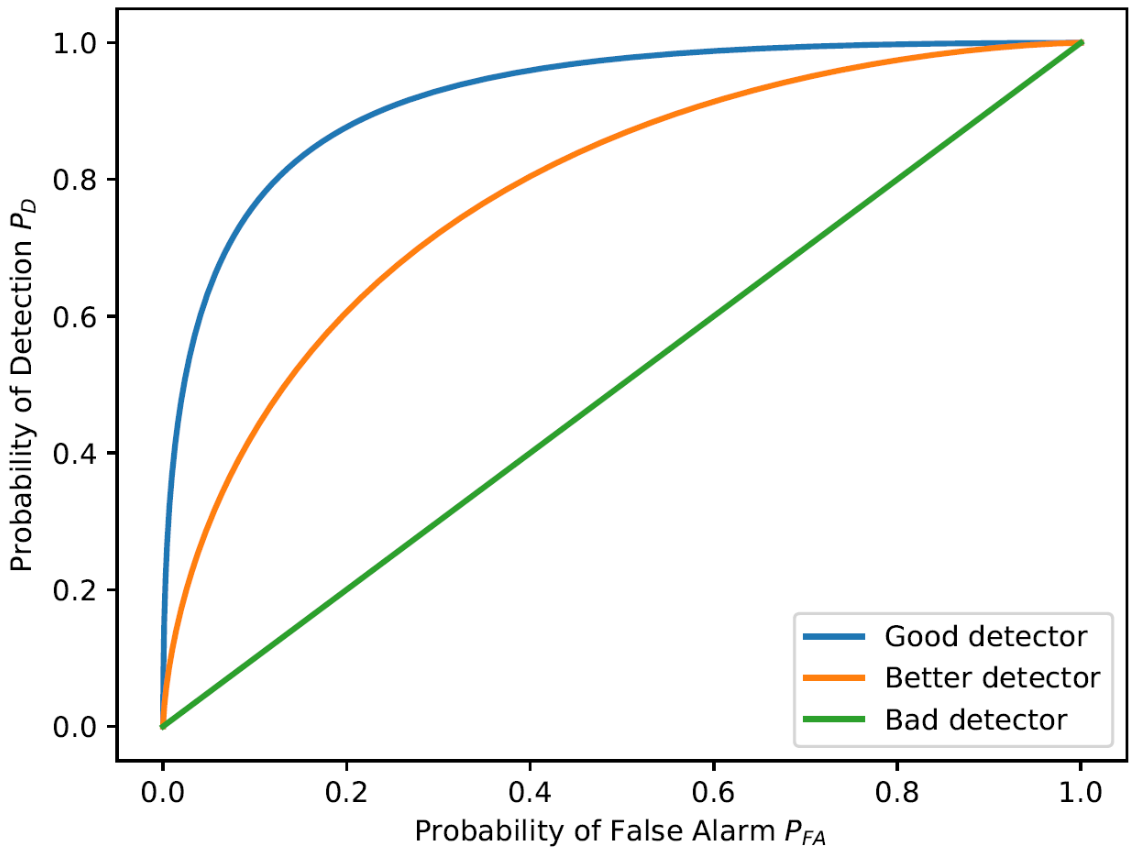


***Figure 37:*** *Receiver operating characteristic curve as a measure of classification performance*

The number of thresholds is set to 200, and likewise the default value of the tensorflow.keras.metrics.AUC function. The integral under the ROC curve represents the AUC score. The higher the ROC curve, and therefore the closer the AUC score is to 1, the better the performance of the classifier.

The trained models were saved utilizing the save attribute of the Model module from tensorflow.keras.models. The model histories were stored as dictionaries in JSON format. All cross-validation models were additionally saved in h5 format. Unfortunately, only models saved in h5 format could be utilized to calculate other evaluation metrics than accuracy. Setting up fresh virtual environments with updated TensorFlow versions didn’t solve the problem.

# 4. RESULTS AND DISCUSSION

This chapter discusses the following aspects of the implemented CNNs and the contexts between those aspects: model configurations, training processes, validation performances, and input dataset quality. The study contains conventional CNNs and residual neural networks, both with varying complexity and different hyperparameter settings. At first, the conventional CNNs are reviewed, followed by the residual neural networks. The balanced datasets described in Section 3.2.6 are applied to both approaches. Subsequently, the best-performing model is deployed in 5-fold cross-validation. Therefore, the algorithm that generates the balanced datasets is applied with 5 different random seeds to generate 5 balanced datasets for cross-validation.

The goal of the chosen CNN configurations and hyperparameter settings is not to find the optimal solution for the given dataset in a comprehensive, methodological way. This work rather aims to experiment exemplarily with the in previous chapters discussed methods in order to apply a broader range of different techniques considering limited work resources. To determine the underlying effects of varying combinations of hyperparameters, it would be a more thorough approach to change only one parameter at a time and reach for a more comprehensive approach. It might also be reasonable to utilize optimization algorithms to find the best-fitting CNN architectures and hyperparameter settings for the given datasets.

Due to the data augmentation described in Section 3.2.5 the datasets contain segments with up to 90% overlap. The dataset algorithm distributes the data, regularized and randomly, into training and validation sets. This means that the validation set might contain segments that overlap with segments in the training set. To examine if the best-performing model reaches similar accuracies with non-overlapping and strictly unknown test data, a leave-one-subject-out cross-validation is carried out exemplarily with 3 recordings from the databases as test sets. The recordings are chosen randomly, with the only constraints being that they must be from different databases and contain different shockable conditions.

Conclusively, misclassified data from the best-performing model is analyzed and compared with the entire test set in order to find significant distinctions.

## 4.1 Conventional CNN models for the detection of shockable heart rhythms

The first models to be tested on predicting shockable cardiac rhythms are conventional CNNs, as described in Section 2.4. All conventional models share a range of equally set hyperparameters, which are summarized in Table 6.

***Table 6.*** *Common hyperparameter setting for CNN models from* ***Table 7***

| Hyperparameter | Choice |
|---|---|
| Activation | ReLU |
| Activation Output layer | Linear |
| Pooling | Max pooling |
| Batch size | 32 |
| Learning rate | 0.001 |
| Optimizer | Adam |
| Loss function | Sparse Categorical Cross Entropy |

The pixel values of the input data are normalized to values between 0 and 1 by applying tensorflow.keras.layers.experimental.preprocessing.Rescaling. All models use ReLu as the activation function, except the classification layer, which passes the output linearly. The reasons for this are discussed in Section 2.4.2. To reduce dimensionality after convolutional layers, max pooling is applied with a pooling window of 2x2 and stride set to 2. The learning rate remains fixed at 0.001, which is compensated by deploying Adam as an optimization algorithm, which provides learning rate adaptation as described in Section 3.4.1. The utilized loss function for all models is sparse categorical cross-entropy, as pointed out in Section 2.4.3. Sparse is chosen since the classes are encoded as integers. As indicated in Section 3.4.2, the batch size is set to 32 for all models.

Overall, 4 different architectures, named CNN1 through CNN4, are implemented with rising model complexity. The layer structures, varying hyperparameters, and accuracy metrics are shown in Table 7.

***Table 7.*** *Summary of best performing CNN models with common hyperparameters from* ***Table 6*** *with nf = number of filters, ks = kernel size, ps = pooling size, str = strides*

| | CNN1 | CNN2 | CNN3 | CNN3BN | CNN4 |
|---|---|---|---|---|---|
| Conv2D (nf,ks) | 32, 3 | 32, 3 | 32,3 | 32, 3 | 32,3 |
| Maxpooling (ps,str) | 2, 2 | 2, 2 | 2, 2 | 2, 2 | 2, 2 |
| Conv2D | 32, 3 | 32, 3 | 32, 3 | 32, 3 | 64, 3 |
| Maxpooling | 2, 2 | 2, 2 | 2, 2 | 2, 2 | 2, 2 |
| Conv2D | 32, 3 | 64, 5 | 64, 5 | 64, 5 | 64, 5 |
| Dropout (D) / Max-norm (M) | - | - | D 0.28 | M 4 | D 0.2, M 4 |
| Maxpooling | 2, 2 | 2, 2 | 2, 2 | 2, 2 | 2, 2 |
| Conv2D | - | 64, 5 | 128, 5 | 128, 5 | 128, 5 |
| Dropout (D) / Max-norm (M) | - | - | D 0.28 | M 4 | D 0.2, M 4 |
| Maxpooling | - | 2, 2 | 2, 2 | 2, 2 | 2, 2 |
| Dense (outputs) | 128 | 128 | 256 | 256 | 512 |
| Dropout (D) / Max-norm (M) | - | D 0.2 | D 0.28 | M 4 | D 0.2, M 4 |
| Dense | 2 | 64 | 128 | 128 | 256 |
| Dropout (D) / Max-norm (M) | - | D 0.2 | D 0.28 | M 4 | D 0.2, M 4 |
| Dense | - | 2 | 2 | 2 | 128 |
| Dropout (D) / Max-norm (M) | - | - | - | - | D 0.2, M 4 |
| Dense | - | - | - | - | 2 |
| Parameters | 428802 | 204578 | 430242 | 432802 | 753346 |
| L2 regularization | 0 | 0 | 0 | 0 | 0.001 |
| Acc. Test set | **0.974** | **0.972** | **0.976** | **0.977** | **0.965** |

The CNN1 approach consists of 3 convolutional layers with constant filter numbers of 32 and kernel sizes of 3x3. One fully connected hidden layer with 128 output neurons and a binary output layer with 2 neurons for classification are implemented. CNN1 doesn't contain regularization methods. It reaches an accuracy of 0.974 on the test set. No errors are assumed, the first approach suggests that the combination of machine learning methods and data preprocessing works quite well for the detection of shockable rhythms in the utilized databases.

CNN2 consists of 4 convolutional layers, whereby the third and fourth convolutional layers have 64 filters with a kernel size of 5x5. The net continues with 2 hidden dense layers of 128 and 64 output neurons. A dropout of 0.2 is deployed on the hidden dense layers. The model reaches an accuracy of 0.972 on the test set, which is slightly worse than CNN1.

CNN3 and CNN3B share the same model architecture, with differences regarding regularization and batch normalization. The models have 4 convolutional layers, starting with 32 filters and a kernel size of 3x3 on the first and second layer, respectively. The third convolutional layer has 64 filters and a kernel size of 5x5, followed by the fourth convolutional layer with 128 filters and a kernel size of 5x5. Two hidden dense layers are attached with 256 and 128 output neurons. CNN3 applies a dropout of 0.28 from the third convolutional layer up to the last hidden dense layer. In contrast, CNN3BN applies a max-norm regularization of 4 on the equivalent layers. Additionally, CNN3BN implements batch-normalization after each layer except for the output layer. Regarding the convolutional layers, batch normalization is applied after the pooling layers. CNN3BN achieved an accuracy of 0.977, which is slightly better than CNN3s 0.976.

The learning curves for accuracy and loss are exemplarily shown for CNN3BN in Figure 38 and Figure 39.

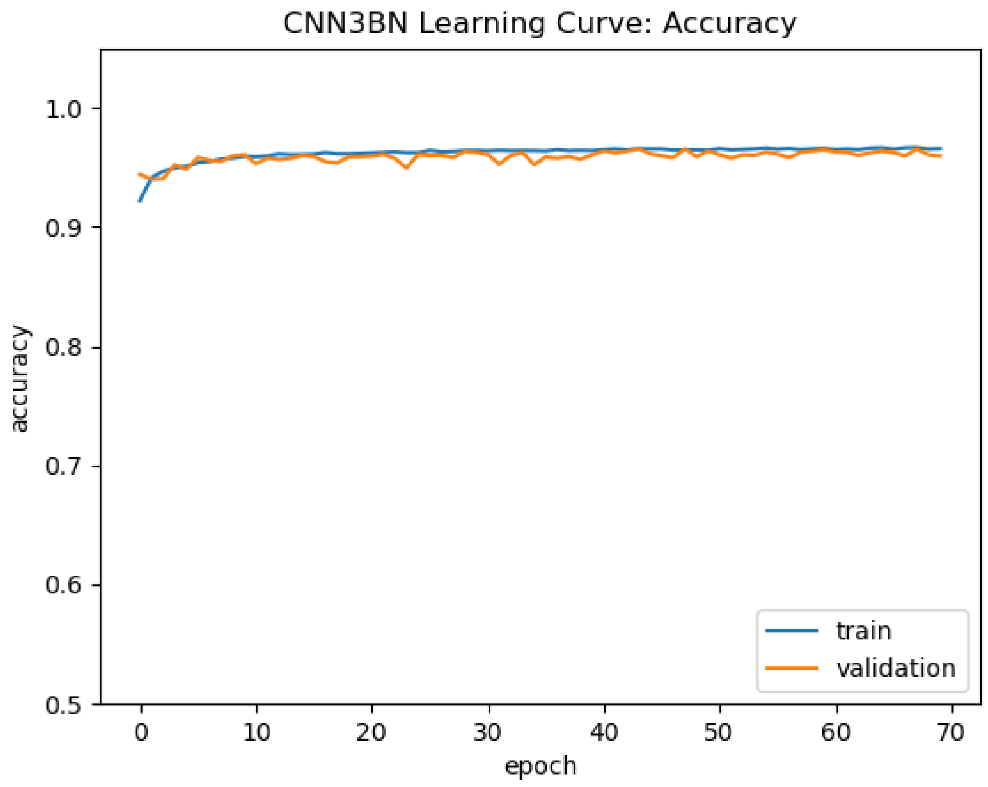


***Figure 38:*** *CNN3BN Learning curves for accuracy metric on training and validation sets*

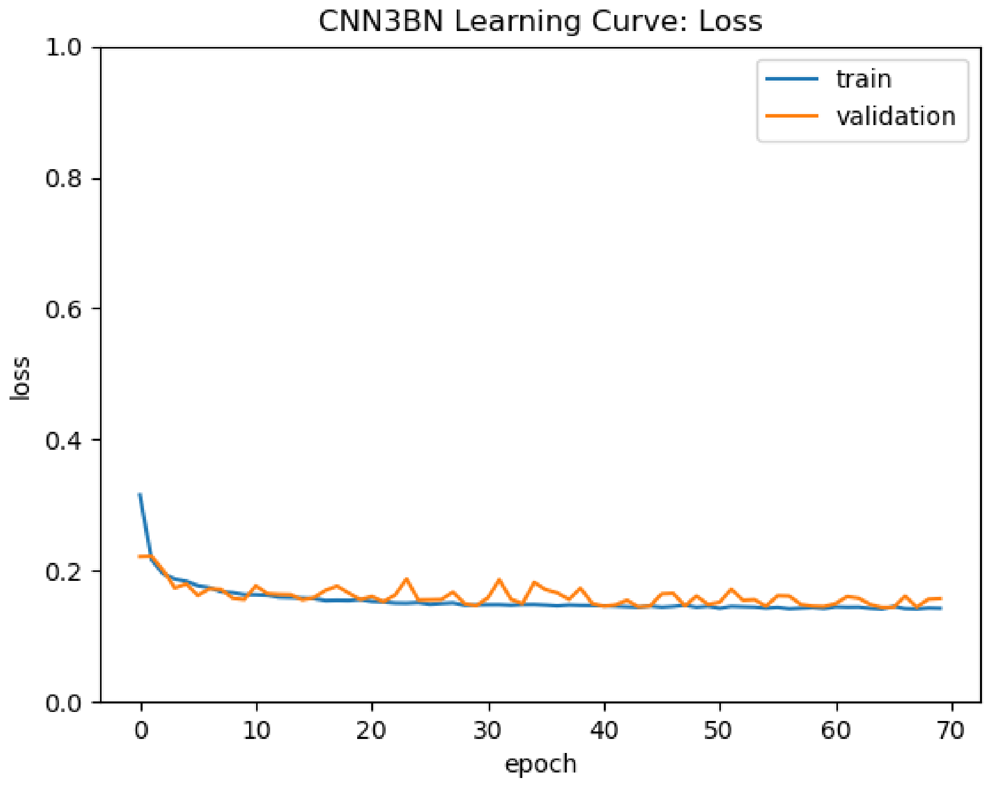


***Figure 39:*** *CNN3BN Learning curves for loss metric on training and validation sets*

It can be observed that the model fits well to training and validation data by reaching accuracy values above 0.9 already after the first epochs. This indicates that the significant patterns, which enable the network to distinguish between shockable and non-shockable segments, are rather prevalent and repetitive over the entire dataset. Therefore, it can be concluded that the given problem is quite suitable for convolutional neural networks. Further, the learning curves for training and validation sets stay tightly together over the training process. This indicates that the training set is well balanced and represents enough variety to prepare the net for the validation set. Additionally, from the learning curves, it can be suggested that the hyperparameter settings are balanced and prevent the model from over- or underfitting the data.

CNN4 contains 4 convolutional layers, starting with 32 filters and a kernel size of 3x3 in the first layer. The second and third layers have 64 filters, whereby the second layer has a kernel size of 3x3 and the third layer of 5x5. The fourth convolutional layer has 128 filters and a kernel size of 5x5. Further, the model includes 3 hidden dense layers with 512, 256, and 128 output neurons. From all the conventional CNNs shown, CNN4 applies the highest grade of regularization. It combines l2 regularization, dropout, and max-norm regularization. L2 regularization of 0.001 is deployed on each layer (except the output layer). Dropout and max-norm regularization start at the third convolutional layer and continue up to the last hidden dense layer. Dropout is set to 0.2 and max-norm regularization to 4. CNN4 reached an accuracy of 0.965 on the test set.

Though CNN4 comes with increased model complexity, it couldn't improve the results of CNN3B. It might be that the combination of the 3 regularization methods prevented the model from reaching a better fit. CNN3B remains the best-performing model for the conventional approach. As mentioned earlier, it seems quite probable that a more comprehensive determination of parameters can lead to better fitting models, as shown by an accuracy of 0.988 in the detection of shockable rhythms with conventional CNNs by Lai et al. [51]. Instead, this work continues to introduce residual neural networks for solving the problem.

## 4.2 Residual Neural Networks

The residual neural networks are implemented as described in Section 3.3. In contrast to the conventional CNNs, the ResNet models allow for an increase in the number of layers and likewise suppress the problem of vanishing gradients. In total, 6 ResNet models of varying complexity are trained and tested. Table 8 summarizes the common hyperparameter settings for all ResNet approaches.

***Table 8.*** *Common hyperparameter setting for ResNet models from* ***Table 9***

| Hyperparameter | Choice |
|---|---|
| Activation | ReLU |
| Activation Output layer | Sigmoid |
| Kernel initializer | He normal |
| Pooling | average (pool size = 8) |
| Batch size | 32 |
| L2 regularization | 0.0001 |
| Optimizer | Adam |
| Loss function | Sparse Categorical Cross Entropy |

Again, the input data are normalized to values between 0 and 1 by applying tensorflow.keras.layers.experimental.preprocessing.Rescaling. The applied activation function is ReLU except for the output layer, which applies Sigmoid. Each convolutional layer deploys L2 regularization set to 0.0001. Adam optimization is utilized to minimize the models cost functions, which are defined with sparse categorical cross entropy. The batch size is set to 32, and the weights are initialized with He initialization as described in Section 3.3. Before flattening and entering the output layer, the dimensionality is reduced by average pooling with a pool size of 8x8.

The number of implemented residual blocks in each model along with the accuracy metrics for the test set are shown in Table 9.

***Table 9.*** *Summary of best performing ResNet models with common hyperparameters from **Table 8**; *spectrogram segments with mean pixel values below 2 are excluded from the dataset*

| | **ResNet1** | **ResNet2** | **ResNet3** | **ResNet4** | **ResNet5** | **ResNet6** |
|---|---|---|---|---|---|---|
| Depth | 56 | 110 | 110 | 218 | 866 | 866 |
| Residual blocks | 6 | 12 | 12 | 24 | 96 | 96 |
| Conv layers | 18 | 36 | 36 | 72 | 288 | 288 |
| Learning rate | scheduled | scheduled | 0.0001 | 0.0001 | 0.00001 | 0.00005 |
| Clip value | 0 | 0 | 0.5 | 0.5 | 0.6 | 0.5 |
| Acc Test set | **0.966** | **0.971** | **0.982** | **0.983** | **0.981** | **0.986** |
| Acc excluding low pixel segments * | | | | | | **0.989** |

ResNet1 consists of 6 residual blocks, which results in 18 convolutional layers and a depth of 56. The learning rate incrementally decreases as described in Table 10. ResNet1 reaches an accuracy of 0.966 on the test set.

***Table 10.*** *Learning rate schedule for ResNet1 and ResNet2 in **Table 9***

| **Learning rate schedule** | |
|---|---|
| initial | 0.01 |
| > 1 epoch | 0.01 |
| > 10 epochs | 0.001 |
| > 40 epochs | 0.0005 |

ResNet2 uses the same learning rate scheme as ResNet1, but increases complexity by doubling the residual blocks to 12. Therefore, the number of convolutional layers goes up to 36 and the total number of layers to 110. The accuracy on the test set jumps to 0.971.

The following approaches apply a fixed learning rate and value clipping to stabilize the learning process (see Section 3.4.3). ResNet3 has the same architecture as ResNet2. In contrast, ResNet3 implements a steady learning rate of 0.0001 and applies a clip value of 0.5. The accuracy on the test set can be improved to 0.982.

ResNet4 keeps the learning rate and clip value settings from ResNet3, but doubles the number of residual blocks to 24, which results in a depth of 218 with 72 convolutional layers. The accuracy reaches 0.983, which is a slight improvement over ResNet3.

ResNet5 comes with 96 residual blocks, which is the deepest approach implemented in this series. ResNet5 has a total of 866 layers, including 288 convolutional layers. The learning rate is decreased to 0.00001, and the clip value is set to 0.6. The model reaches an accuracy of 0.981 and is therefore no improvement over the previous approaches.

The final model ResNet6 uses the same complexity as ResNet5, but increases the learning rate to 0.00005 and sets the clip value back to 0.5. ResNet6 reaches an accuracy of 0.986 on the test set and is therefore the best-performing model.

When analyzing the misclassified data, it appears that segments with very low mean pixel intensities are significantly more prevalent. These segments presumably contain very little pattern information for the network to deal with, since they seem to be technically flatline segments but are annotated differently. More insights on this are given in Section 4.5. When excluding segments with mean pixel values below 2 from the test set, the accuracy of ResNet6 improves to 0.989.

## 4.3 5-fold cross-validation

The architecture and hyperparameter settings from the best-performing model, ResNet6 are now utilized to perform a 5-fold cross-validation. The datasets are generated with the script for balanced datasets, which is described in Section 3.2.6. 5 datasets, containing subsets for training, validation, and testing, are determined by implementing 5 random seeds from 1 to 5 within the algorithm. Each dataset is applied to train and test a ResNet model, whereas each approach is named from k1 to k5. The in Section 3.5 described metrics accuracy, sensitivity, specificity, precision, and ROC AUC are calculated for each model and summarized in Table 11.

***Table 11.*** *5-fold cross-validation based on ResNet 6 model architecture*

| ResNet 6 | k1 | k2 | k3 | k4 | k5 | Average |
|---|---|---|---|---|---|---|
| Accuracy | 0.989 | 0.986 | 0.986 | 0.988 | 0.987 | **0.987** |
| Sensitivity | 0.996 | 0.99 | 0.988 | 0.997 | 0.989 | **0.992** |
| Specificity | 0.982 | 0.988 | 0.984 | 0.984 | 0.984 | **0.984** |
| Precision | 0.980 | 0.982 | 0.981 | 0.979 | 0.983 | **0.981** |
| ROC AUC | 0.997 | 0.998 | 0.998 | 0.998 | 0.997 | **0.998** |

On average, the model reaches an accuracy of 0.987 on the test sets. The sensitivity for shockable rhythms stands at 0.992 with a precision of 0.981. Non-shockable segments are detected with a specificity of 0.984. The AUC score reaches 0.998.

Table 12 compares some performance metrics of the cross-validation with other works that utilize CNNs to detect shockable heart rhythms. The previous approaches are introduced in Section 2.3.2.

***Table 12.*** *Comparison of CNN based methods for the detection of shockable cardiac rhythms (Acc.=accuracy, Se.=sensitivity, Sp.=specificity)*

| Reference | Databases | Input di-mension-ality | seg-ment length in s | k-fold cross-vali-dation | Acc. | Se. | Sp. |
|---|---|---|---|---|---|---|---|
| Acharya et al 2017 [58] | CUDB, MITDB, VFDB | 1D | 2 | k = 10 | 0.932 | 0.953 | 0.910 |
| Nguyen et al 2018 [59] | CUDB, VFDB | 1D | 8 | k = 5 | 0.993 | 0.971 | 0.994 |
| Lai et al 2020 [51] | AHADB, CUDB, MITDB, VFDB | 2D | 3 | k = 10 | 0.988 | 0.951 | 0.994 |
| **This work** | **AHADB, CUDB, MITDB, VFDB** | **2D** | **3** | **k = 5** | **0.987** | **0.992** | **0.984** |

With their 1D CNN model, Nguyen et al. reach an accuracy of 0.993 on 8 s segments derived from CUDB and VFDB. The work of Nguyen et al. discarded parts of the data by stating: “The artifacts, noise, asystole, transition rhythms, slow VT of intermediate rhythms with rate under 150 beats per minute, and peak-to-peak amplitude under 200 µV of VF rhythms are removed”.[59] Lai et al. and this work extend the considered databases by adding AHADB and MITDB and reach similar accuracies of 0.988 and 0.987. Both works process 3 s segments. Whereas this work experiments with conventional CNNs and ResNets, Lai et al. focus on conventional CNNs [51]. The sensitivity for shockable segments stands at 0.992 in this work, compared to 0.951 for Lai et al. This might be caused by the preprocessing steps focusing on generating balanced datasets for training the models (see Section 3.2). Due to data augmentation, the training data are enriched with shockable segments, especially around episode borders, where the annotation changes. Another difference might be that this work discarded segments where the majority class (shockable or not shockable) made up less than 60% of the segment. Lai et al. left out segments with “extreme artifacts” or with peak-to-peak amplitudes below 150 µV [51]. The augmentation process in this work produces segments with overlaps in the datasets, which stands in contrast to the other works mentioned in Table 12. Therefore, a leave-one-subject-out cross-validation is carried out to evaluate the impact of overlapping segments on the model’s performance.

## 4.4 Leave-one-subject-out cross-validation

To evaluate if the model architecture achieves similar results when it is ensured that no overlapping segments cross training and test sets, whole recordings are excluded from training and preserved for testing. The leave-one-subject-out cross-validation is only carried out exemplarily by applying 3 folds. A more comprehensive approach would be desirable but would exceeds the scope of this work. The left-out recordings are chosen randomly, with the constraints that each subject must come from a different database and contain a different shockable condition.

Recording 207 from MITDB contains VFL episodes. Recording 420 from VFDB includes VT episodes. Since CUDB recordings are only 8 mins long instead of 30 mins, recordings 1, 3, and 4 from CUDB are combined into one validation set. Therefore, the number of segments in training and validation sets maintains roughly the same. Table 13 summarizes the validation metrics of the 3 trained ResNet6 models in leave-one-subject-out cross-validation.

***Table 13.*** *3-fold leave-one-subject-out cross-validation on ResNet6 model architecture*

| Test Set | MITDB 207 | VFDB 420 | CUDB 1&3&4 | Average |
|---|---|---|---|---|
| Accuracy | 0.984 | 0.992 | 0.977 | **0.984** |
| Sensitivity | 0.970 | 0.984 | 0.997 | **0.984** |
| Specificity | 0.991 | 1.000 | 0.948 | **0.980** |
| Precision | 0.994 | 1.000 | 0.959 | **0.984** |
| ROC AUC | 0.999 | 1.000 | 0.993 | **0.997** |

With an average accuracy of 0.984 and a sensitivity for shockable rhythms of 0.984, the metrics are similar to the 5-fold cross-validation. To profoundly validate the results, leave-one-subject-out cross-validation should be performed with more than 3 subjects in a future approach. Nevertheless, the results indicate that the problem of overlapping segments within the balanced datasets doesn't cause the models to produce false or unrealistic validation metrics.

## 4.5 Misclassified data analysis

In the following, the analysis aims to formulate hypotheses that try to give explanations why parts of the test data were misclassified by the trained models. The analysis is carried out on the test set used for the ResNet6 approach in Section 4.2, where 1.37% of the test data were falsely classified. Common problems that lead to misclassification of test data are overfitting the model with training data or discrepancies between training sets and test sets. Those incompatibilities can be caused by imbalanced data sets or by a training set that is simply too limited to prepare the net properly for unknown test data. The last point is countered by generating a balanced dataset.

**Label distribution and border segments**

In the context of this work an episode within an ECG recording is defined as a section with a single annotation or label. Segments around episode borders are likely to contain several labels. The proportion of segments containing at least 2 labels is 4.73% within the test set. This is already elevated due to the border augmentation during the creation of the balanced dataset. One reason to enrich border segments in the datasets is to provide the model with more samples containing several cardiac rhythms during the training process in order to produce a more robust classifier. Even though border segments are more prevalent during training, it seems challenging for the model to classify all border segments correctly. Figure 40 reveals that 11.29% of the falsely classified segments are indeed border segments. Therefore, the set of misclassified segments contains more than double the share of border segments compared to the test set.

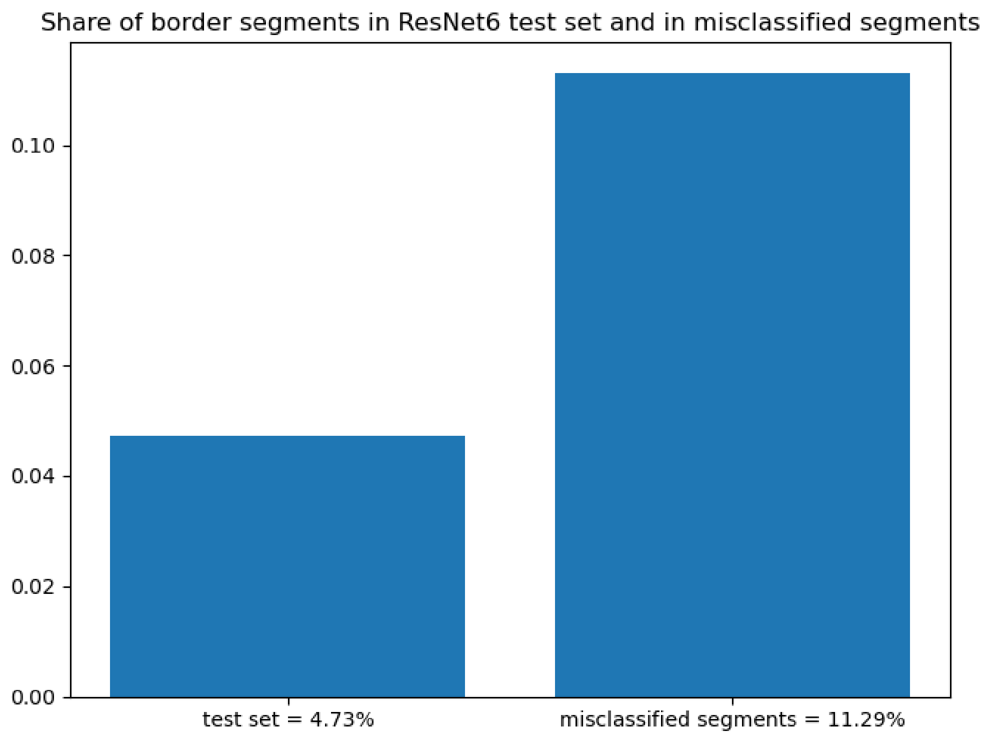


***Figure 40:*** *Comparison of the proportions of border segments within ResNet6 test set and in by ResNet6 misclassified segments from the test set*

Another possibility to analyze the model’s performance is to count the appearances of all annotations throughout the test set and compare the distribution to the set of misclassified segments. The label distributions for the test set and the set of misclassified labels are visualized in Figure 41. Table 2 in Section 3.2.4 gives an overview of all cardiac annotations used in the databases.

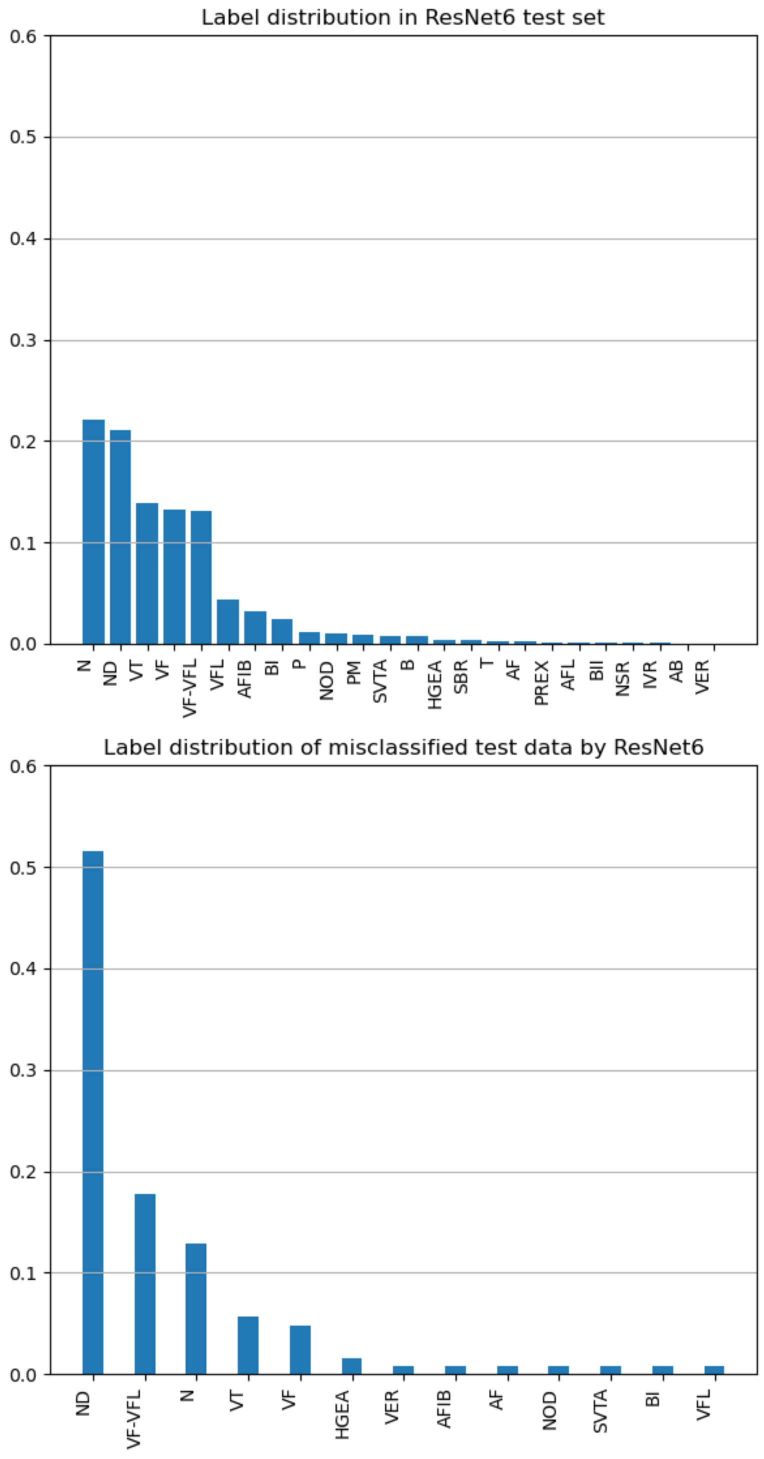


***Figure 41:*** *Comparison of label distributions of Resnet6 test set and its misclassified segments*

Since the set of misclassified segments is rather small (124 segments compared to the test set containing almost 9000 segments), analyzing the label distributions would be more robust if multiple sets of misclassified segments derived from several test sets were considered. This extends the scope of this work, but it can be done in future analyses. Nevertheless, two aspects of the label distributions are worth mentioning. The majority of misclassified segments are with 51.6% annotated as “not defined” (ND). This is a quite elevated proportion compared to the 21.1% in the label distribution of the test set. The second noticeable aspect refers to a shockable condition. 17.7% of misclassified segments are labeled as VF-VFL. The share of VF-VFL in the test set is only 13.2%. The difference might not occur as crucially, but the shares regarding other shockable segments are all significantly lower in the set of misclassified segments than in the test set.

## Mean value analysis

In order to find more hints as to why some of the data were misclassified, the mean and median pixel values for all spectrogram segments were calculated and distributed as illustrated in Figure 42.

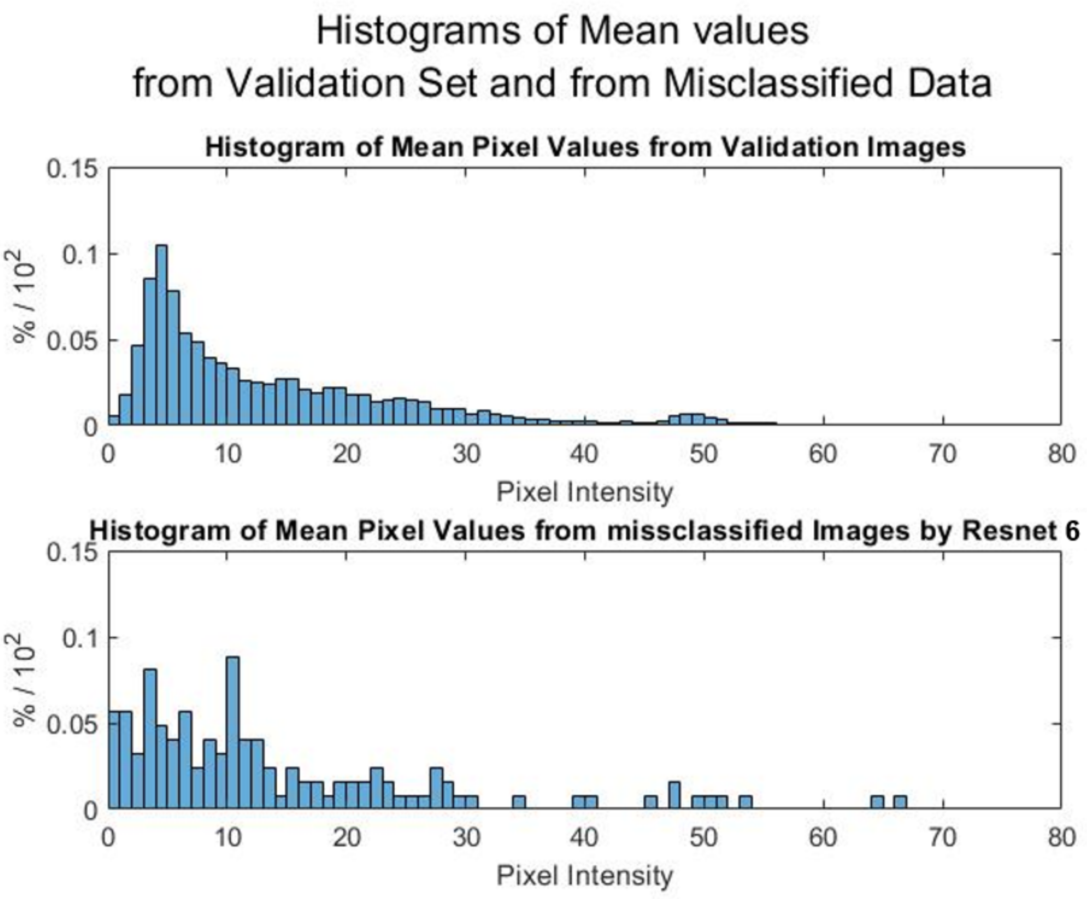


***Figure 42:*** *Histograms of mean pixel values for validation set and for misclassified spectrograms*

By comparing the histograms of mean pixel values, it appears that low pixel intensities between 0 and 2 appear more frequently in the set of misclassified segments:

validation data set:

p(mean_val [0 1]) = 0.005601

p(mean_val [1 2]) = 0.01748

p(mean_val[0 2]) = 0.02308

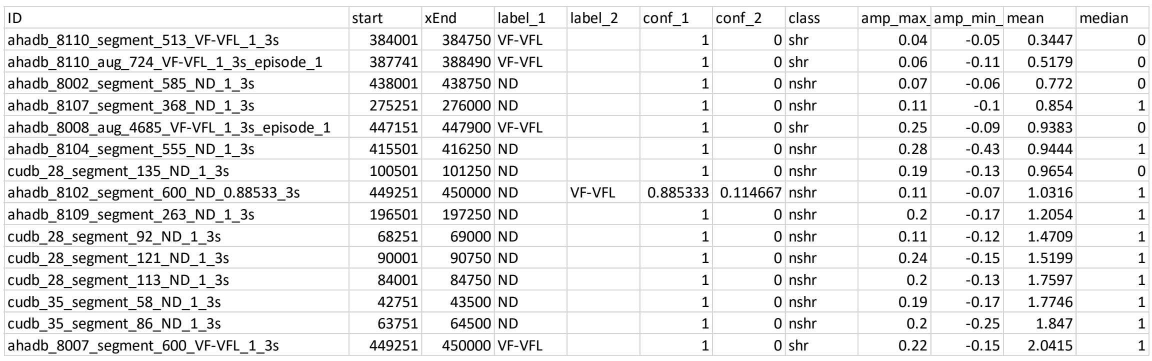
misclassified segments:

p(mean_f[0 1]) = 0.05645

p(mean_f[1 2]) = 0.05645

p(mean_f[0 2]) = 0.1129

When looking at the ECG signals for these low mean value segments, it is noticeable that the amplitude ranges (amp_max_mV and amp_min_mV) are quite narrow around 0, as shown in the extract of Figure 43. It seems reasonable to assume that there is a correlation between low pixel intensity mean values, low amplitude ranges in the ECG signal, and misclassification by the net. If there is almost nothing to be seen in the signal or the spectrogram, the net can't find characteristic features and patterns for the true label, though the segment might originate from an episode that is annotated as having a particular rhythm, including shockable ones. By excluding segments with mean pixel values below 2, the accuracy of ResNet6 improves from 0.986 to 0.989.



| ID | start | xEnd | label_1 | label_2 | conf_1 | conf_2 | class | amp_max_ | amp_min_ | mean | median |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ahadb_8110_segment_513_VF-VFL_1_3s | 384001 | 384750 | VF-VFL | | 1 | 0 | shr | 0.04 | -0.05 | 0.3447 | 0 |
| ahadb_8110_aug_724_VF-VFL_1_3s_episode_1 | 387741 | 388490 | VF-VFL | | 1 | 0 | shr | 0.06 | -0.11 | 0.5179 | 0 |
| ahadb_8002_segment_585_ND_1_3s | 438001 | 438750 | ND | | 1 | 0 | nshr | 0.07 | -0.06 | 0.772 | 0 |
| ahadb_8107_segment_368_ND_1_3s | 275251 | 276000 | ND | | 1 | 0 | nshr | 0.11 | -0.1 | 0.854 | 1 |
| ahadb_8008_aug_4685_VF-VFL_1_3s_episode_1 | 447151 | 447900 | VF-VFL | | 1 | 0 | shr | 0.25 | -0.09 | 0.9383 | 0 |
| ahadb_8104_segment_555_ND_1_3s | 415501 | 416250 | ND | | 1 | 0 | nshr | 0.28 | -0.43 | 0.9444 | 1 |
| cudb_28_segment_135_ND_1_3s | 100501 | 101250 | ND | | 1 | 0 | nshr | 0.19 | -0.13 | 0.9654 | 0 |
| ahadb_8102_segment_600_ND_0.88533_3s | 449251 | 450000 | ND | VF-VFL | 0.885333 | 0.114667 | nshr | 0.11 | -0.07 | 1.0316 | 1 |
| ahadb_8109_segment_263_ND_1_3s | 196501 | 197250 | ND | | 1 | 0 | nshr | 0.2 | -0.17 | 1.2054 | 1 |
| cudb_28_segment_92_ND_1_3s | 68251 | 69000 | ND | | 1 | 0 | nshr | 0.11 | -0.12 | 1.4709 | 1 |
| cudb_28_segment_121_ND_1_3s | 90001 | 90750 | ND | | 1 | 0 | nshr | 0.24 | -0.15 | 1.5199 | 1 |
| cudb_28_segment_113_ND_1_3s | 84001 | 84750 | ND | | 1 | 0 | nshr | 0.2 | -0.13 | 1.7597 | 1 |
| cudb_35_segment_58_ND_1_3s | 42751 | 43500 | ND | | 1 | 0 | nshr | 0.19 | -0.17 | 1.7746 | 1 |
| cudb_35_segment_86_ND_1_3s | 63751 | 64500 | ND | | 1 | 0 | nshr | 0.2 | -0.25 | 1.847 | 1 |
| ahadb_8007_segment_600_VF-VFL_1_3s | 449251 | 450000 | VF-VFL | | 1 | 0 | shr | 0.22 | -0.15 | 2.0415 | 1 |

***Figure 43:*** *draft -> extract of misclassified segments sorted by mean pixel values in ascending order*

# 5. CONCLUSION

This work tries to sensitize readers to the relevance of the assessment of cardiac arrhythmias, especially shockable cardiac arrhythmias. Regarding the application of defibrillation, it is shown that the rapid and precise determination of shockable cardiac conditions is crucial in saving lives. Detection algorithms for shockable conditions in commercially available defibrillation devices reach sensitivities of approximately 90% and specificities of around 95% (see Section 2.2.4). Since there is room to improve and due to the rise of automated feature detection in machine learning, this work experiments with the deployment of convolutional neural networks for the classification of shockable cardiac conditions. Four databases (AHADB, CUDB, MITDB, and VFDB) are exploited and processed to optimize CNN training with balanced datasets of ECG spectrogram segments. The wavelet transform is applied to convert 1-channel ECG recordings to 2-dimensional spectrogram representations.

Conventional CNN models experimenting with various hyperparameter settings demonstrate that the detection of shockable conditions works well with the chosen approach by reaching accuracies between 97% and 98% on the given data. The approach with residual neural networks reaches 98.7% on average in the cross-validation. Besides training and testing models with balanced datasets leave-one-subject-out cross-validation shows that the approach works likewise well when ensured that entire ECG recordings make up the test sets without any interference with training sets. The leave-one-subject-out cross-validation, which was exemplarily carried out on 3 recordings, reaches an accuracy of 98.4% and indicates a similar performance level of the network as for validation with the balanced datasets.

It is worth mentioning that the model settings are not optimized for maximizing performance but rather applied exemplarily to draw a more comprehensive picture of different strategies. Even so, the models already achieve remarkable results. Future examinations might focus more on optimizing certain hyperparameter settings to maximize the model's performance. Another aspect of future approaches is balancing model performance with the computational demand for the device. The deeper and more complex the neural network is, the higher its demand for computational power. When it comes to commercial deployment, it needs to be ensured that the chosen model works well with the device’s hardware and resource limitations. An interesting approach could be to implement the model to TensorFlow Lite, which is a platform that is optimized for deploying models on mobile devices. The effects on model performance and computational demand could be evaluated and compared to this work. When developing a balanced model between classification performance and resource demands, it might be appropriate to continue the leave-one-subject-out cross-validation over the entire set of 125 ECG recordings to obtain a more robust picture of the model's performance.

There are other limitations that should be examined in future work. It should be experimented with different wavelets and parameter settings throughout the wavelet transform as well as with different image representations of the spectrograms regarding size, compression, grayscale, and color. Other preprocessing steps, such as filtering, might be altered too. One or more CNN models can serve as references to evaluate the effects of adjustments in the preprocessing.

The threshold that a segment needs to consist of more than 60% of the majority class could be removed. On the contrary, to enhance comparability with AED performance testing, the models could be tested also on segments that contain only one heart rhythm [11]. It is probable and could be shown in future work that the models perform better on segments containing only one rhythm.

It might also be beneficial to analyze the datasets more thoroughly for events such as artifacts and flatlines, which then gives the opportunity to examine misclassified data in the context of these events.

Though many research approaches to developing detection algorithms for cardiac conditions use the same databases as the ones being processed in this work, either partially or entirely, on the one hand, it makes the research easier to compare with each other. On the other hand, it seems that a data pool of ECG recordings derived from only 125 individuals must heavily underrepresent the entirety of real-world situations with billions of potential patients. Therefore, it seems that the field of automated ECG analysis would massively profit from larger databases that would provide a more comprehensive picture of reality.

Besides aiming for the improvement of commercial defibrillation devices, this work should be seen as an inspiration for the potential of machine learning in deploying automated feature detection in ECG analysis. The methods explored in this work and beyond are not restricted to the detection of shockable cardiac conditions. The technique is rather flexible and can be applied to other areas regarding the classification of ECG data. It is also thinkable to utilize automated feature detection over various fields in healthcare and to discover cross-links between patterns in ECG data and other health-related fields than cardiology, such as neurology, psychiatry, epigenetics, or something else.

Automated classification methods for ECG data, such as those in this work, can also be used to generate a more comprehensive clinical database to boost research in the field. When it comes to certain cardiac conditions that can be classified by algorithms with an equal degree of accuracy as medical professionals, these algorithms can be used to automatically annotate ECG data by deriving soft labels from unsupervised learning. In an iterative process, potential soft labels are cross-checked by medical experts and might become hard labels for automated annotation. In this way, resources can be saved. Further, the generated database can be utilized to improve the classification algorithms, which again boosts the automated annotation.

.

# REFERENCES

[1] Benjamin EJ, Virani SS, Callaway CW, Chamberlain AM, Chang AR, Cheng S, et al. Heart Disease and Stroke Statistics-2018 Update: A Report From the American Heart Association. Circulation 2018;137:e67–492. https://doi.org/10.1161/CIR.0000000000000558.

[2] KIHT. Defibrillator Technical Compendium. Kalam Institute Of Health Technology; 2018.

[3] Petersen KF. Legal Implications of Lay Use of Automatic External Defibrillators in Non-Hospital Settings. Journalof ContemporaryHealth Law and Policy 2000;17:47.

[4] Nishiyama T, Nishiyama A, Negishi M, Kashimura S, Katsumata Y, Kimura T, et al. Diagnostic Accuracy of Commercially Available Automated External Defibrillators. J Am Heart Assoc 2015;4:e002465. https://doi.org/10.1161/JAHA.115.002465.

[5] Jaureguibeitia X, Zubia G, Irusta U, Aramendi E, Chicote B, Alonso D, et al. Shock Decision Algorithms for Automated External Defibrillators Based on Convolutional Networks. IEEE Access 2020;8:154746–58. https://doi.org/10.1109/ACCESS.2020.3018704.

[6] Reece JB, Jackson RB, Urry LA, Cain ML, Minorsky PV, Wasserman SA. Campbell Biology. Pearson College Division; 2012.

[7] Wapcaplet. Diagram of the human heart, created by Wapcaplet in Sodipodi. Cropped by Yaddah to remove white space (this cropping is not the same as Wapcaplet's original crop). 2006.

[8] Malmivuo J, Plonsey R. Bioelectromagnetism: Principles and Applications of Bioelectric and Biomagnetic Fields. New York: Oxford University Press; 1995. https://doi.org/10.1093/acprof:oso/9780195058239.001.0001.

[9] Cardiac conduction system. Wikipedia 2023.

[10] Garcia TB, Holtz NE. 12 Lead ECG: The Art of Interpretation. Jones & Bartlett Learning; 2001.

[11] Kerber RE, Becker LB, Bourland JD, Cummins RO, Hallstrom AP, Michos MB, et al. Automatic External Defibrillators for Public Access Defibrillation: Recommendations for Specifying and Reporting Arrhythmia Analysis Algorithm Performance, Incorporating New Waveforms, and Enhancing Safety. Circulation 1997;95:1677–82. https://doi.org/10.1161/01.CIR.95.6.1677.

[12] Jacobs I, Sunde K, Deakin CD, Hazinski MF, Kerber RE, Koster RW, et al. Part 6: Defibrillation. Circulation 2010;122:S325–37. https://doi.org/10.1161/CIRCULATIONAHA.110.971010.

[13] Electrocardiography. Wikipedia 2023.

[14] Bennett DH. Bennett's Cardiac Arrhythmias: Practical Notes on Interpretation and Treatment. John Wiley & Sons; 2012.

[15] Stroobandt RX, Barold SS, Sinnaeve AF. ECG from Basics to Essentials: Step by Step. John Wiley & Sons; 2016.

[16] Ventricular Tachycardia n.d. https://www.hopkinsmedicine.org/health/conditions-and-diseases/ventricular-tachycardia (accessed January 10, 2022).

[17] Sweeney MO. Antitachycardia Pacing for Ventricular Tachycardia. Medscape n.d. http://www.medscape.com/viewarticle/490543 (accessed January 10, 2022).

[18] Olshansky B, Chung MK, Pogwizd SM, Goldschlager N. Arrhythmia Essentials E-Book. Elsevier Health Sciences; 2016.

[19] Mirowski M, Mower MM, Reid PR. The automatic implantable defibrillator. Am Heart J 1980;100:1089–92. https://doi.org/10.1016/0002-8703(80)90218-5.

[20] Ventricular fibrillation. Wikipedia 2022.

[21] Ventricular flutter. Wikipedia 2021.

[22] Ong MEH, Lim SH, Venkataraman A. Defibrillation and Cardioversion. In: Tintinalli JE, Stapczynski JS, Ma OJ, Yealy DM, Meckler GD, Cline DM, editors. Tintinalli's Emergency Medicine: A Comprehensive Study Guide. 8th ed., New York, NY: McGraw-Hill Education; 2016.

[23] Marino PL. The ICU Book. Lippincott Williams & Wilkins; 2007.

[24] PREVOST J. La mort par les courants electriques-courants alternatifs a haute tension. J Physiol Path Gen 1899;1:427.

[25] Ball CM, Featherstone PJ. Early history of defibrillation. Anaesth Intensive Care 2019;47:112–5. https://doi.org/10.1177/0310057X19838914.

[26] Atwood C, Eisenberg MS, Herlitz J, Rea TD. Incidence of EMS-treated out-of-hospital cardiac arrest in Europe. Resuscitation 2005;67:75–80. https://doi.org/10.1016/j.resuscitation.2005.03.021.
[27] Rea TD, Eisenberg MS, Sinibaldi G, White RD. Incidence of EMS-treated out-of-hospital cardiac arrest in the United States. Resuscitation 2004;63:17–24. https://doi.org/10.1016/j.resuscitation.2004.03.025.
[28] McNally B, Stokes A, Crouch A, Kellermann AL, CARES Surveillance Group. CARES: Cardiac Arrest Registry to Enhance Survival. Ann Emerg Med 2009;54:674-683.e2. https://doi.org/10.1016/j.annemergmed.2009.03.018.
[29] Committee on the Treatment of Cardiac Arrest: Current Status and Future Directions, Board on Health Sciences Policy, Institute of Medicine. Strategies to Improve Cardiac Arrest Survival: A Time to Act. Washington (DC): National Academies Press (US); 2015.
[30] Andersen LW, Holmberg MJ, Berg KM, Donnino MW, Granfeldt A. In-Hospital Cardiac Arrest. JAMA 2019;321:1200–10. https://doi.org/10.1001/jama.2019.1696.
[31] Fuchs A, Käser D, Theiler L, Greif R, Knapp J, Berger-Estilita J. Survival and long-term outcomes following in-hospital cardiac arrest in a Swiss university hospital: a prospective observational study. Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine 2021;29:115. https://doi.org/10.1186/s13049-021-00931-0.
[32] Valenzuela TD, Roe DJ, Nichol G, Clark LL, Spaite DW, Hardman RG. Outcomes of rapid defibrillation by security officers after cardiac arrest in casinos. N Engl J Med 2000;343:1206–9. https://doi.org/10.1056/NEJM200010263431701.
[33] Bækgaard JS, Viereck S, Møller TP, Ersbøll AK, Lippert F, Folke F. The Effects of Public Access Defibrillation on Survival After Out-of-Hospital Cardiac Arrest: A Systematic Review of Observational Studies. Circulation 2017;136:954–65. https://doi.org/10.1161/CIRCULATIONAHA.117.029067.
[34] California S of. AED | EMSA n.d. https://emsa.ca.gov/aed/ (accessed January 14, 2022).
[35] Ghzally Y, Mahajan K. Implantable Defibrillator. StatPearls, Treasure Island (FL): StatPearls Publishing; 2022.
[36] DiMarco JP. Implantable Cardioverter–Defibrillators. N Engl j Med 2003:12.
[37] Kouakam C, Kacet S, Hazard J-R, Ferraci A, Mansour H, Defaye P, et al. Performance of a dual-chamber implantable defibrillator algorithm for discrimination of ventricular from supraventricular tachycardia. EP Europace 2004;6:32–42. https://doi.org/10.1016/j.eupc.2003.09.007.
[38] Frontera A. ICD Algorithms in the management of arrhythmias: Pitfalls and advancements. PhD Thesis. Bordeaux, 2019.
[39] Daubert JP, Zareba W, Cannom DS, McNitt S, Rosero SZ, Wang P, et al. Inappropriate implantable cardioverter-defibrillator shocks in MADIT II: frequency, mechanisms, predictors, and survival impact. J Am Coll Cardiol 2008;51:1357–65. https://doi.org/10.1016/j.jacc.2007.09.073.
[40] Kaup HJ, Hexamer M, Werner J. Morphological detection algorithms for the automatic implantable cardioverter/defibrillator (AICD). Biomed Tech (Berl) 2004;49:306–10. https://doi.org/10.1515/BMT.2004.057.
[41] Arafat MA, Chowdhury AW, Hasan MdK. A simple time domain algorithm for the detection of ventricular fibrillation in electrocardiogram. SIViP 2011;5:1–10. https://doi.org/10.1007/s11760-009-0136-1.
[42] Chen S, Thakor NV, Mower MM. Ventricular fibrillation detection by a regression test on the autocorrelation function. Med Biol Eng Comput 1987;25:241–9. https://doi.org/10.1007/BF02447420.
[43] Jekova I. Comparison of five algorithms for the detection of ventricular fibrillation from the surface ECG. Physiol Meas 2000;21:429–39. https://doi.org/10.1088/0967-3334/21/4/301.
[44] Thakor NV, Zhu YS, Pan KY. Ventricular tachycardia and fibrillation detection by a sequential hypothesis testing algorithm. IEEE Trans Biomed Eng 1990;37:837–43. https://doi.org/10.1109/10.58594.
[45] Koninklijke Philips N.V. Philips SMART Analysis AED algorithm 2019.
[46] SIST EN 60601-2-4:2011 - Medical electrical equipment - Part 2-4: Particular requirements for basic safety and essential performance of cardiac defibrillators. ITeh Standards Store n.d. https://standards.iteh.ai/catalog/standards/sist/adc8ebce-67a4-4f49-bd65-9f3c8c73eb78/sist-en-60601-2-4-2011 (accessed January 20, 2022).

[47] Li Q, Rajagopalan C, Clifford GD. Ventricular Fibrillation and Tachycardia Classification Using a Machine Learning Approach. IEEE Transactions on Biomedical Engineering 2014;61:1607–13. https://doi.org/10.1109/TBME.2013.2275000.

[48] Figuera C, Irusta U, Morgado E, Aramendi E, Ayala U, Wik L, et al. Machine Learning Techniques for the Detection of Shockable Rhythms in Automated External Defibrillators. PLoS One 2016;11:e0159654. https://doi.org/10.1371/journal.pone.0159654.

[49] Alonso-Atienza F, Morgado E, Fernández-Martínez L, García-Alberola A, Rojo-Álvarez JL. Detection of Life-Threatening Arrhythmias Using Feature Selection and Support Vector Machines. IEEE Transactions on Biomedical Engineering 2014;61:832–40. https://doi.org/10.1109/TBME.2013.2290800.

[50] Ibtehaz N, Rahman MS, Rahman MS. VFPred: A fusion of signal processing and machine learning techniques in detecting ventricular fibrillation from ECG signals. Biomedical Signal Processing and Control 2019;49:349–59. https://doi.org/10.1016/j.bspc.2018.12.016.

[51] Lai D, Fan X, Zhang Y, Chen W. Intelligent and Efficient Detection of Life-Threatening Ventricular Arrhythmias in Short Segments of Surface ECG Signals. IEEE Sensors Journal 2020:1–1. https://doi.org/10.1109/JSEN.2020.3031597.

[52] Amann A, Tratnig R, Unterkofler K. Reliability of old and new ventricular fibrillation detection algorithms for automated external defibrillators. Biomed Eng Online 2005;4:60. https://doi.org/10.1186/1475-925X-4-60.

[53] Anas EMA, Lee SY, Hasan MK. Sequential algorithm for life threatening cardiac pathologies detection based on mean signal strength and EMD functions. BioMedical Engineering OnLine 2010;9:43. https://doi.org/10.1186/1475-925X-9-43.

[54] Computers in Cardiology: September 1978, Stanford University, Calif. IEEE Computer Society; 1978.

[55] Goldberg DE, Goldberg DE, Goldberg G David Edward, Goldberg VAP of HDE. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company; 1989.

[56] Mallawaarachchi V. Introduction to Genetic Algorithms — Including Example Code. Medium 2020. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (accessed February 10, 2022).

[57] Hai DT, Tuan NM, Thu-Hang NT, Le CH. An Efficient Shock Advice Algorithm based on K-Nearest Neighbors for Automated External Defibrillators. 2021 International Conference on Advanced Technologies for Communications (ATC), 2021, p. 57–61. https://doi.org/10.1109/ATC52653.2021.9598311.

[58] Acharya UR, Fujita H, Oh SL, U R, Tan JH, Adam M, et al. Automated identification of shockable and non-shockable life-threatening ventricular arrhythmias using convolutional neural network. Future Generation Computer Systems 2017;79. https://doi.org/10.1016/j.future.2017.08.039.

[59] Nguyen MT, Nguyen BV, Kim K. Deep Feature Learning for Sudden Cardiac Arrest Detection in Automated External Defibrillators. Sci Rep 2018;8. https://doi.org/10.1038/s41598-018-33424-9.

[60] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, et al. Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation 1989;1:541–51. https://doi.org/10.1162/neco.1989.1.4.541.

[61] Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybernetics 1980;36:193–202. https://doi.org/10.1007/BF00344251.

[62] Hubel DH. Single unit activity in striate cortex of unrestrained cats. J Physiol 1959;147:226-238.2.

[63] Receptive fields and functional architecture of monkey striate cortex - PubMed n.d. https://pubmed.ncbi.nlm.nih.gov/4966457/ (accessed April 8, 2022).

[64] Kharpann Enterprises. Building A Convolutional Neural Network - The Click Reader n.d. https://www.theclickreader.com/building-a-convolutional-neural-network/ (accessed April 11, 2022).

[65] Rohrer B. How do Convolutional Neural Networks work? 2016. https://e2eml.school/how_convolutional_neural_networks_work.html (accessed April 12, 2022).

[66] Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc.; 2012.

[67] Goodfellow I, Bengio Y, Courville A. Deep Learning. Illustrated edition. Cambridge, Massachusetts: The MIT Press; 2016.
[68] Glorot X, Bordes A, Bengio Y. Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings; 2011, p. 315–23.
[69] Géron A. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed Edition. Beijing China ; Sebastopol, CA: O'Reilly UK Ltd.; 2019.
[70] Multi-Temporal Remote Sensing Image Registration Using Deep Convolutional Features | IEEE Journals & Magazine | IEEE Xplore n.d. https://ieeexplore.ieee.org/document/8404075 (accessed April 15, 2022).
[71] Olah C, Mordvintsev A, Schubert L. Feature Visualization. Distill 2017;2:e7. https://doi.org/10.23915/distill.00007.
[72] Yosinski J, Clune J, Nguyen A, Fuchs T, Lipson H. Understanding Neural Networks Through Deep Visualization. ArXiv:150606579 [Cs] 2015.
[73] Decaro C, Montanari G, Molinariz R, Gilberti A, Bagnoli D, Bianconi M, et al. Machine Learning Approach for Prediction of Hematic Parameters in Hemodialysis Patients. IEEE Journal of Translational Engineering in Health and Medicine 2019;PP:1–1. https://doi.org/10.1109/JTEHM.2019.2938951.
[74] Gebel Ł. Why We Need Bias in Neural Networks. Medium 2022. https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98 (accessed May 3, 2022).
[75] Chollet F. Deep Learning with Python. 1st edition. Shelter Island, New York: Manning; 2017.
[76] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 1958;65:386–408. https://doi.org/10.1037/h0042519.
[77] Verma Y. Addressing The Vanishing Gradient Problem: A Guide For Beginners. Analytics India Magazine 2021. https://analyticsindiamag.com/addressing-the-vanishing-gradient-problem-a-guide-for-beginners/ (accessed May 6, 2022).
[78] Shukla L. Designing Your Neural Networks. Medium 2019. https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed (accessed May 6, 2022).
[79] Brownlee J. How to Configure the Number of Layers and Nodes in a Neural Network. Machine Learning Mastery 2018. https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/ (accessed May 7, 2022).
[80] Sharma P. 7 Popular Image Classification Models in ImageNet Challenge (ILSVRC) Competition History. MLK - Machine Learning Knowledge 2020. https://machinelearningknowledge.ai/popular-image-classification-models-in-imagenet-challenge-ilsvrc-competition-history/ (accessed May 8, 2022).
[81] Stojnic R. Papers with Code - ImageNet Benchmark (Image Classification) n.d. https://paperswithcode.com/sota/image-classification-on-imagenet (accessed May 12, 2022).
[82] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. ArXiv:151203385 [Cs] 2015.
[83] Koech KE. Cross-Entropy Loss Function. Medium 2021. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e (accessed May 12, 2022).
[84] Dawar H. Stochastic Gradient Descent. Analytics Vidhya 2020. https://medium.com/analytics-vidhya/stochastic-gradient-descent-1ab661fabf89 (accessed May 12, 2022).
[85] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986;323:533–6. https://doi.org/10.1038/323533a0.
[86] Kostadinov S. Understanding Backpropagation Algorithm. Medium 2019. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd (accessed May 13, 2022).
[87] Pykes K. The Vanishing/Exploding Gradient Problem in Deep Neural Networks. Medium 2020. https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11 (accessed May 15, 2022).
[88] Clayton RH, Murray A. Comparison of techniques for time–frequency analysis of the ECG during human ventricular fibrillation. IEE Proceedings - Science, Measurement and Technology 1998;145:301–6. https://doi.org/10.1049/ip-smt:19982322.
[89] Clayton RH, Murray A, Campbell RWF. Comparison of four techniques for recognition of ventricular fibrillation from the surface ECG. Med Biol Eng Comput 1993;31:111–7. https://doi.org/10.1007/BF02446668.

[90] Nguyen MT, Shahzad A, Nguyen B, Kim K. Diagnosis of Shockable Rhythms for Automated External Defibrillators Using a Reliable Support Vector Machine Classifier. Biomedical Signal Processing and Control 2018;44. https://doi.org/10.1016/j.bspc.2018.03.014.

[91] Nguyen MT, Nguyen T-HT, Le H-C. A review of progress and an advanced method for shock advice algorithms in automated external defibrillators. Biomed Eng Online 2022;21:22. https://doi.org/10.1186/s12938-022-00993-w.

[92] Jeong J-W, Lee IB, Song Y, Jang Y, Noh HW, Lee S. Sequential algorithm for the detection of the shockable rhythms in electrocardiogram. Annu Int Conf IEEE Eng Med Biol Soc 2012;2012:5082–5. https://doi.org/10.1109/EMBC.2012.6347136.

[93] Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database. IEEE Engineering in Medicine and Biology Magazine 2001;20:45–50. https://doi.org/10.1109/51.932724.

[94] Goldberger AL, Amaral LA, Glass L, Hausdorff JM, Ivanov PC, Mark RG, et al. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. Circulation 2000;101:E215-220. https://doi.org/10.1161/01.cir.101.23.e215.

[95] Musa N, Gital AY, Aljojo N, Chiroma H, Adewole KS, Mojeed HA, et al. A systematic review and Meta-data analysis on the applications of Deep Learning in Electrocardiogram. J Ambient Intell Humaniz Comput 2022:1–74. https://doi.org/10.1007/s12652-022-03868-z.

[96] Silva I, Moody G. An Open-source Toolbox for Analysing and Processing PhysioNet Databases in MATLAB and Octave. Journal of Open Research Software 2014;2:e27. https://doi.org/10.5334/jors.bi.

[97] Xu Y, Wang D, Zhang W, Ping P, Feng L. Detection of ventricular tachycardia and fibrillation using adaptive variational mode decomposition and boosted-CART classifier. Biomedical Signal Processing and Control 2018;39:219–29. https://doi.org/10.1016/j.bspc.2017.07.031.

[98] Jekova I, Krasteva V. Real time detection of ventricular fibrillation and tachycardia. Physiol Meas 2004;25:1167–78. https://doi.org/10.1088/0967-3334/25/5/007.

[99] Lenis G, Pilia N, Loewe A, Schulze WHW, Dössel O. Comparison of Baseline Wander Removal Techniques considering the Preservation of ST Changes in the Ischemic ECG: A Simulation Study. Computational and Mathematical Methods in Medicine 2017;2017:e9295029. https://doi.org/10.1155/2017/9295029.

[100] Gabor D. Theory of communication. Part 1: The analysis of information. Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering 1946;93:429–41. https://doi.org/10.1049/ji-3-2.1946.0074.

[101] Wachowiak MP, Wachowiak-Smolíková R, Johnson MJ, Hay DC, Power KE, Williams-Bell FM. Quantitative feature analysis of continuous analytic wavelet transforms of electrocardiography and electromyography. Philos Trans A Math Phys Eng Sci 2018;376:20170250. https://doi.org/10.1098/rsta.2017.0250.

[102] Masters D, Luschi C. Revisiting Small Batch Training for Deep Neural Networks. ArXiv:180407612 [Cs, Stat] 2018.

[103] Brownlee J. How to Control the Stability of Training Neural Networks With the Batch Size. Machine Learning Mastery 2019. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (accessed May 15, 2022).

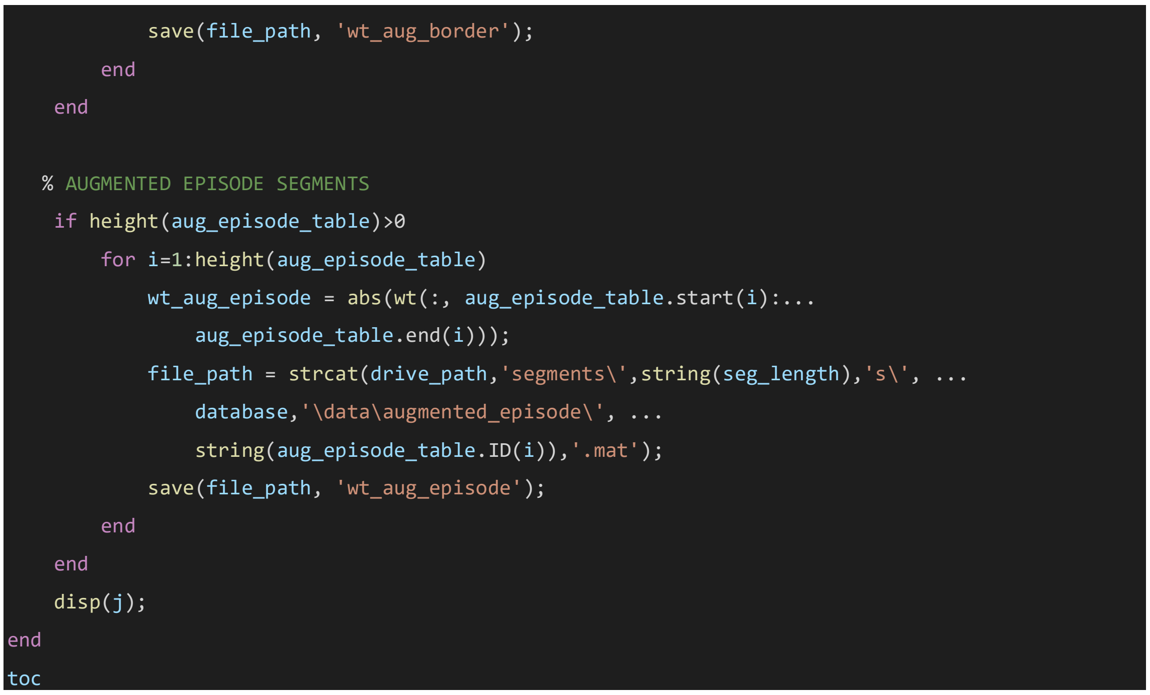
[104] Overfitting. Wikipedia 2022.

# 6. APPENDIX

```
clc;
clear all;
close all;

%% PREPROCESSING

%Annotation classes:
%+ = rhythm change
%" = comment annotation
%A = Atrial premature beat
%F = Fusion of ventricular and normal beat
%V = Premature ventricular contraction
%N = Normal beat
%~ = Change in signal quality

%Shockable rhythm: VT (ventr. tachycardia),
%VF ventr. fibrill.), VFL (ventric. flutter)
% GENERAL PARAMETERS
wavelet = 'morse';% wavelet type
seg_length = 3;%segemnt length in s
fs = 250;% uniform sampling frequency

drive_path = 'E:\shr_detection\';
%database = 'cudb';
database = 'mitdb';
%database = 'vfdb';
%database = 'ahadb';

if strcmp(database, 'mitdb')
   % MITDB
   %Dataset 1: MIT-BIH arrhythmia
   %mitdb/100 - 124, 200-234
    f = 360;% Sampling frequency from database
    db_id = 1000;% database index for metadata
    file_names = readmatrix('mitdb_files.csv');
end


if strcmp(database, 'cudb')
   % CUDB
   % Creighton university ventricular
   % tachyarrhythmia db (CUDB)
    f = 250;% Sampling frequency from database
    db_id = 2000;% database index for metadata
    file_names = readmatrix('cudb_files.csv');
end

if strcmp(database, 'vfdb')
   % MIT-BIH malignant ventricular arrhythmia (VFDB)
   % vfdb/418-430, 602-615 (some ids missing from between)
    f = 250;% Sampling frequency from database
    db_id = 3000;% database index for metadata
    file_names = readmatrix('vfdb_filenames.csv');
end

if strcmp(database, 'ahadb')
   % AHA Database (not public)
    aha_path = 'E:\shr_detection\AHA_database\AHA Text Data\';
    f = 250;
    db_id=40000;
    file_names = readmatrix('ahadb_filenames.csv');
end
```

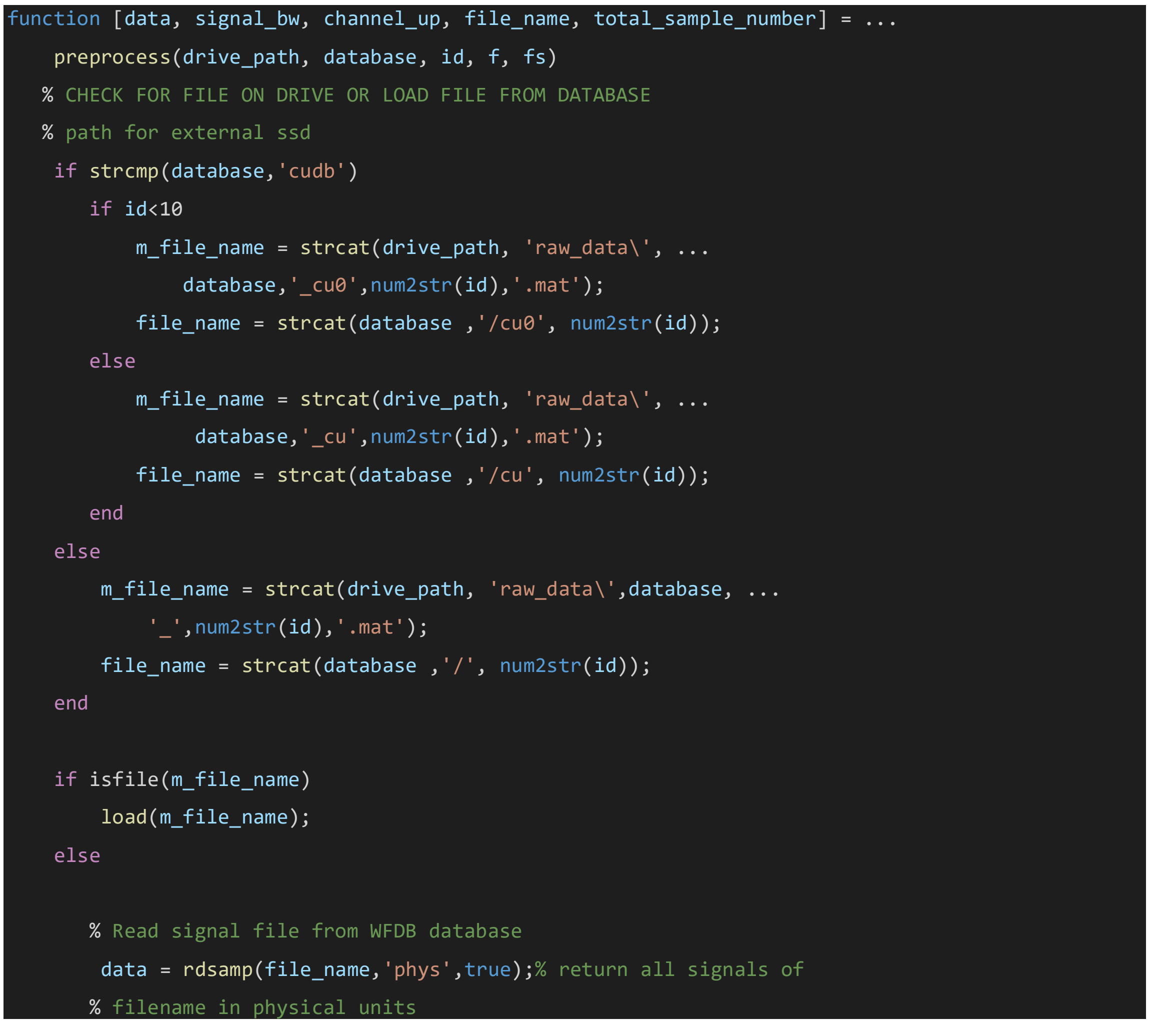
***Figure 44:*** *Pre-processing main script in MATLAB part 1*

```
% Initialize tables
[segment_tbl_all, aug_border_tbl_all, aug_episode_tbl_all, ...
    episode_tbl_all, episode_VFLS_tbl_all] = deal(table());

tic
for j=1:length(file_names)
    id = file_names(j);
   % PREPROCESSING
    if strcmp(database, 'ahadb')
        [data, signal_bw, channel_up, file_name, total_sample_number] = ...
            preprocess_aha(aha_path, database, id, f, fs);
    else
        [data, signal_bw, channel_up, file_name, total_sample_number] = ...
            preprocess(drive_path, database, id, f, fs);
    end

   % WAVELET TRANSFORM
    wav_name = "morse";
    [wt,frequencies] = cwt(signal_bw, wav_name, fs, 'VoicesPerOctave', 20, 'NumOctaves', 5,
'TimeBandwidth',50);


   % ANNOTATION AND AUGMENTATION
   %Find events/labels in Annotation frame
   % typeMnemonic (non-beat annotations
   % "[" Start of ventricular flutter/fibrillation
   % "]" End of ventricular flutter/fibrillation
   % "+" rhythm change
   % auxinfo (Beat annotation)
   % Shockable rhythms:
   % "(VFL" Ventricular flutter
   % "(VT" Ventricular tachycardia
   % "VFLS" Start of VFL/VF
   % "VFLE" End of VFL/VF
    if strcmp(database, 'ahadb')
        [ann_table, segment_table, ann_table_events, sample_labels, file_length] = ...
            annotate_aha(data, fs, seg_length, database, id);

    else
        [ann_table, segment_table, ann_table_events, sample_labels, file_length] = ...
            annotate(file_name, f, fs, channel_up, seg_length, database, id);
    end

   % AUGMENTATION AROUND EPISODE BORDERS
    [aug_border_table, episode_table, episode_VFLS_table] = ...
        augment_border(ann_table_events, ...
        database, id, seg_length, fs, sample_labels, file_length);

   % AUGMENTATION ALONG EPISODES
    aug_episode_table = augment_episodes(episode_table, ...
        database, id, seg_length, fs);


   % CONCATENATE SEGMENT TABLES AND AUGMENTATION TABLES
    segment_tbl_all = vertcat(segment_tbl_all, segment_table);
    aug_border_tbl_all = vertcat(aug_border_tbl_all, aug_border_table);
    aug_episode_tbl_all = vertcat(aug_episode_tbl_all, aug_episode_table);
    episode_tbl_all = vertcat(episode_tbl_all, episode_table);
    episode_VFLS_tbl_all = vertcat(episode_VFLS_tbl_all, episode_VFLS_table);

   % SEGMENTATION
   % STANDARD SEGMENTS
    for i=1:height(segment_table)
       wt_segment = abs(wt(:, segment_table.start(i):segment_table.end(i)));
       file_path = strcat(drive_path,'segments\',string(seg_length),'s\', ...
           database,'\data\standard\',string(segment_table.ID(i)),'.mat');
       save(file_path, 'wt_segment');
    end

   % AUGMENTED BORDER SEGMENTS
    if height(aug_border_table)>0
        for i=1:height(aug_border_table)
            wt_aug_border = abs(wt(:, aug_border_table.start(i):...
                aug_border_table.end(i)));
            file_path = strcat(drive_path,'segments\',string(seg_length),'s\', ...
                database,'\data\augmented_border\', ...
                string(aug_border_table.ID(i)),'.mat');
```



```
            save(file_path, 'wt_aug_border');
        end
    end

   % AUGMENTED EPISODE SEGMENTS
    if height(aug_episode_table)>0
        for i=1:height(aug_episode_table)
            wt_aug_episode = abs(wt(:, aug_episode_table.start(i):...
                aug_episode_table.end(i)));
            file_path = strcat(drive_path,'segments\',string(seg_length),'s\', ...
                database,'\data\augmented_episode\', ...
                string(aug_episode_table.ID(i)),'.mat');
            save(file_path, 'wt_aug_episode');
        end
    end
    disp(j);
end
toc
```

***Figure 45:*** *Pre-processing main script in MATLAB part 2*



```
function [data, signal_bw, channel_up, file_name, total_sample_number] = ...
    preprocess(drive_path, database, id, f, fs)
   % CHECK FOR FILE ON DRIVE OR LOAD FILE FROM DATABASE
   % path for external ssd
    if strcmp(database,'cudb')
       if id<10
           m_file_name = strcat(drive_path, 'raw_data\', ...
               database,'_cu0',num2str(id),'.mat');
           file_name = strcat(database ,'/cu0', num2str(id));
       else
           m_file_name = strcat(drive_path, 'raw_data\', ...
                database,'_cu',num2str(id),'.mat');
           file_name = strcat(database ,'/cu', num2str(id));
       end
    else
        m_file_name = strcat(drive_path, 'raw_data\',database, ...
            '_',num2str(id),'.mat');
        file_name = strcat(database ,'/', num2str(id));
    end

    if isfile(m_file_name)
        load(m_file_name);
    else

       % Read signal file from WFDB database
        data = rdsamp(file_name,'phys',true);% return all signals of
       % filename in physical units
       % write data matrix to m file
        save(m_file_name, 'data');
    end

   % RESAMPLE TO 250 Hz
    channel_up = resample(data(:,2), fs, f);
    total_sample_number = length(channel_up);
   %channel_low = resample(data(:,3), fs, f);
    time = (1:length(channel_up)).' / fs;

   % LINEAR INTERPOLATION OF MISSING SIGNAL VALUES
    channel_up = fillmissing(channel_up, 'linear');
   %channel_low = fillmissing(channel_low, 'linear');

   % DENOISING
   % VF and Tachycardia Classification using a machine learning approach
   % (Li, Rajagopalan, and Clifford 2014)
   % - 250 Hz sampling frequency
   % - Highpass 1 Hz cut-off
   % - Second order 30 Hz Butterworth low-pass filter
   % - 50 Hz notch filter

    fc = 30;
   % - Highpass 1 Hz cut-off
    signal_hp = highpass(channel_up, 1, fs);

   % - Second order 30 Hz Butterworth low-pass filter
    [a,b] = butter(2, fc/(fs/2));
    signal_bw = filter(a,b, signal_hp);

   % title_1 = 'Filtered signal';
   % title_2 = 'raw signal';
   % testplot(signal_bw, channel_up, time, start, end_, title_1,
   % title_2, 3);
end
```

***Figure 46:*** *MATLAB function for querying databases, resampling, and filtering*

```
function [ann_table, segment_table, ann_table_events, sample_labels, file_length] = ...
    annotate(file_name, f, fs, channel_up, seg_length, database, id)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %% ANNOTATION
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    annotation = rdann(file_name,'atr');
    ann_table_compl = struct2table(annotation);%Format into table
    ann_table = ann_table_compl(:,[1,2,3,7]);%reduce to relevant columns
```

```
  %Find events/labels in Annotation frame
  % typeMnemonic (non-beat annotations
  % "[" Start of ventricular flutter/fibrillation
  % "]" End of ventricular flutter/fibrillation
  % "+" rhythm change
   bool_change = ann_table.typeMnemonic == "+";
   bool_vfl_start = ann_table.typeMnemonic == "[";
   bool_vfl_end = ann_table.typeMnemonic == "]";
   bool_lab = bool_change + bool_vfl_start + bool_vfl_end;
   ann_event_idx = find(bool_lab);
   ann_table_events = ann_table(ann_event_idx,:);

  % CLEAN ANNOTATION LABEL NAMES

  % auxinfo (Beat annotation)
  % Shockable rhythms:
  % "(VFL" Ventricular flutter
  % "(VT" Ventricular tachycardia
  % "VFLS" Start of VFL
  % "VFLE" End of VFL

  % CUDB only considers VF episodes in typeMnemonic column with:
  % "[" Start of ventricular flutter/fibrillation
  % "]" End of ventricular flutter/fibrillation
   if strcmp(database,'cudb')
      % replace "[" with "VF"
       ann_table_events.auxInfo(ismember(ann_table_events.typeMnemonic, "[")) = cellstr("VF");
      % replace "]" with "VF"
       ann_table_events.auxInfo(ismember(ann_table_events.typeMnemonic, "]")) = cellstr("VF");
   else
      % replace "[" with "VFLS" for start of VFL
       ann_table_events.auxInfo(ismember(ann_table_events.typeMnemonic, "[")) = cell-
str("VFLS");
      % replace "]" with "VLSE" for end of VFL
       ann_table_events.auxInfo(ismember(ann_table_events.typeMnemonic, "]")) = cell-
str("VFLE");
   end
  % remove "(" from Annotation labels
   ann_labels = string(ann_table_events.auxInfo);
   ann_labels = replace(ann_labels, "(", "");
   ann_table_events.auxInfo = ann_labels;

  % RESAMPLE ANNOTATION TABLE TO UNIFORM SAMPLING FREQUENCY fs=250
  % Unified total sample number = 450000
   q = fs/f;% resampling factor
  % Resample sampling number:
   ann_table_events.sampleNumber = round(ann_table_events.sampleNumber * q);
  % Resample time:
   ann_table_events.timeInSeconds = ann_table_events.timeInSeconds * q;

  % CREATE SAMPLE LABEL VECTOR
   file_length = floor(length(channel_up)/fs) * fs;
   sample_numbers = [1:file_length]';
  % mark undefined samples as ND
   sample_labels = repmat({'ND'},file_length,1);

   if strcmp(database,'cudb')
       for i=1:height(ann_table_events)
           if ~strcmp(ann_table_events.typeMnemonic(i), ']')
               if i == height(ann_table_events)
                  episode_start = ann_table_events.sampleNumber(i);
                  episode_end = file_length;
                  sample_labels(episode_start:episode_end) = cellstr(ann_table_events.aux-
Info(i));
               else
                   episode_start = ann_table_events.sampleNumber(i);
                   episode_end = ann_table_events.sampleNumber(i+1)-1;
                   sample_labels(episode_start:episode_end) = cellstr(ann_table_events.aux-
Info(i));
               end
           end
       end
   else
       for i=1:height(ann_table_events)
           if i == height(ann_table_events)
              episode_start = ann_table_events.sampleNumber(i);
              episode_end = file_length;
              sample_labels(episode_start:episode_end) = cellstr(ann_table_events.auxInfo(i));
           else
               episode_start = ann_table_events.sampleNumber(i);
               episode_end = ann_table_events.sampleNumber(i+1)-1;
               sample_labels(episode_start:episode_end) = cellstr(ann_table_events.auxInfo(i));
           end
       end
   end

  % CREATE SEGMENT TABLE WITH LABEL AND LABEL CONFIDENCE
   segment_sample_numbers = seg_length * fs;% Number of samples per segment
   nrows = floor(file_length / segment_sample_numbers);% Number of rows in segment table

  % Initialize Segment labels and their confidence values
   segment_first_labels = repmat({''},nrows,1);
   first_label_confidence = zeros(nrows,1);

   segment_second_labels = segment_first_labels;
   second_label_confidence = first_label_confidence;
   segment_third_labels = segment_first_labels;
   third_label_confidence = first_label_confidence;

   segment_fourth_labels = segment_first_labels;
   fourth_label_confidence = first_label_confidence;

  % Initialize segment IDs, start, and end
   ID = repmat({''},nrows,1);
   segment_end = 0;
   [start, end_] = deal(zeros(nrows,1));

   for i=0:(nrows-1)
       if file_length - segment_end >= 750
           segment_start = segment_sample_numbers * i + 1;
           segment_end = segment_sample_numbers * (i+1);
           segment = sample_labels(segment_start : segment_end);
           start(i+1) = segment_start;
           end_(i+1) = segment_end;
          % count unique strings
          % unique_strings as a vector of unique elements in segment and idx_c as an
          % index vector referring to the index in unique_strings for each sample in segment
           [label, idx_a, idx_c] = unique(segment);
          % count repeating indices in idx_c and create a Count table in descending
          % order
           counts = accumarray(idx_c, 1);
           count_table = table(label, counts);
           count_table = sortrows(count_table,'counts', 'descend');
          % retreive the label with majority vote from count_table and calculate
          % label confidence
           segment_first_labels(i+1) = count_table.label(1);
           first_label_confidence(i+1) = count_table.counts(1) / segment_sample_numbers;

          % Check for second label in segment
           if length(count_table.label)>=2
               segment_second_labels(i+1) = count_table.label(2);
               second_label_confidence(i+1) = count_table.counts(2) / segment_sample_numbers;
           end
          % Check for third label in segment
           if length(count_table.label)>=3
               segment_third_labels(i+1) = count_table.label(3);
               third_label_confidence(i+1) = count_table.counts(3) / segment_sample_numbers;
           end
          % Check for fourth label in segment
           if length(count_table.label)>=4
               segment_fourth_labels(i+1) = count_table.label(4);
               fourth_label_confidence(i+1) = count_table.counts(4) / segment_sample_numbers;
           end

          % Define segment ID containing database, file number, segment number,
          % segment length, label, label confidence

           segment_ID = strcat(database ,'_', num2str(id),'_segment_', num2str(i+1),'_',...
               string(count_table.label(1)), '_', string(first_label_confidence(i+1)),...
               '_', num2str(seg_length),'s');
           ID(i+1) = {segment_ID};
       end
   end
   segment_table = table(ID, start, end_, ...
                   segment_first_labels, segment_second_labels, ...
                   segment_third_labels, segment_fourth_labels, ...
                   first_label_confidence, second_label_confidence, ...
                   third_label_confidence, fourth_label_confidence);
   segment_table.Properties.VariableNames = {'ID' 'start' 'end' ...
           'label_1' 'label_2' 'label_3' 'label_4' 'conf_1' ...
           'conf_2' 'conf_3' 'conf_4'};

end
```

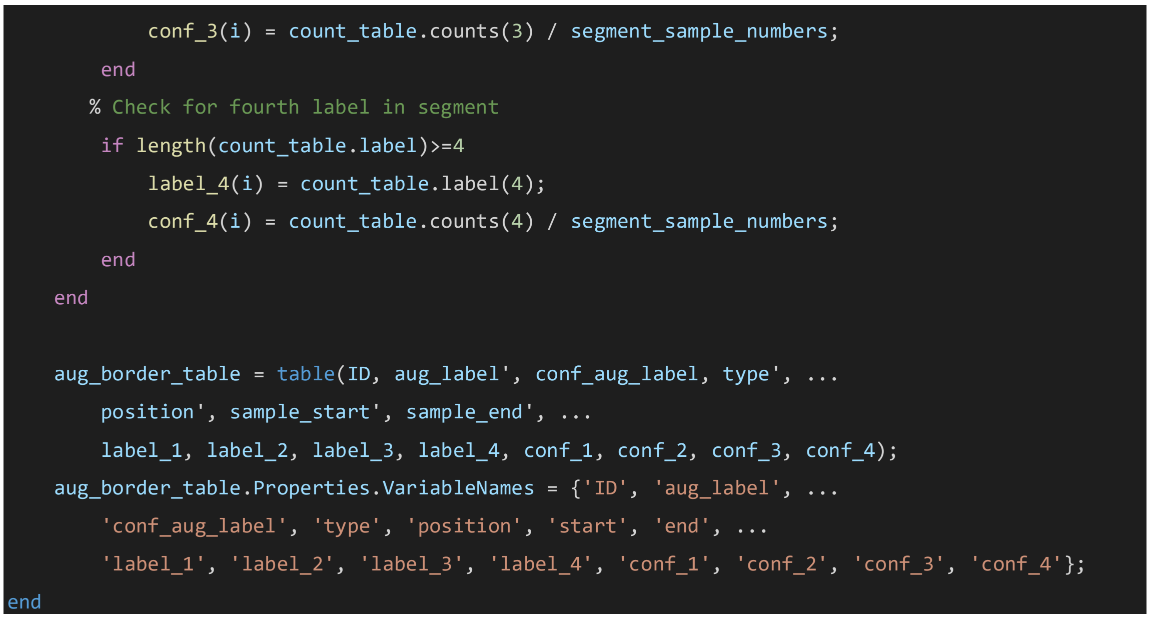
***Figure 47:*** *MATLAB annotation function*

```
function [aug_border_table, episode_table, episode_VFLS_table] = ...
    augment_border(ann_table_events, ...
    database, id, seg_length, fs, sample_labels, file_length)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %% AUGMENTATION AROUND EPISODE BORDERS
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(database, 'ahadb')
       % CREATE EPISODE TABLE
        [aug_label, episode_id] = deal({});
        [ep_start, ep_end] = deal([]);
        k=1;

        for i=1:height(ann_table_events)
            if string(ann_table_events.label(i)) == '['...
                    && ann_table_events.sample_number(i) < file_length
                ep_start(k) = ann_table_events.sample_number(i);
                if i<height(ann_table_events)
                    ep_end(k) = ann_table_events.sample_number(i+1);
                else
                    ep_end(k) = file_length;
                end
                aug_label(k) = {'VF-VFL'};
                episode_id(k) = {strcat(database,'_', string(id),'_ep_',num2str(k))};
                k = k+1;
            end
        end
        episode_table = table(episode_id', aug_label', ep_start', ep_end');
        episode_table.Properties.VariableNames = {'episode_id', 'aug_label', ...
            'ep_start', 'ep_end'};

        episode_VFLS_table = episode_table;
    else
       % Filter ann_table_events by VT and VFL
       % assuming that VFLS and VFLE are always short and therefore not considered
       % to be determined later in Python
        table_filter = {'VT' 'VFL' 'VF'}';
        table_filter_idx = ismember(ann_table_events.auxInfo, table_filter);
        ann_table_shr = ann_table_events(table_filter_idx,:);

        [aug_label, episode_id] = deal({});
       %ep_start = zeros(height(ann_table_vfls),1);
       %ep_end = zeros(height(ann_table_vfls),1);
        [ep_start, ep_end] = deal([]);
        k=1;
        if strcmp(database, 'cudb')
            for i=1:height(ann_table_events)
                if (ann_table_events.auxInfo(i) == 'VT' || ann_table_events.auxInfo(i) == 'VFL'
...
                        || string(ann_table_events.typeMnemonic(i)) == '[')...
                        && ann_table_events.sampleNumber(i) < file_length
                    ep_start(k) = ann_table_events.sampleNumber(i);
                    if i<height(ann_table_events)
                        ep_end(k) = ann_table_events.sampleNumber(i+1);
                    else
                        ep_end(k) = file_length;
                    end
                    aug_label(k) = {ann_table_events.auxInfo(i)};
                    episode_id(k) = {strcat(database,'_', string(id),'_ep_',num2str(k))};
                    k = k+1;
                end
            end
        else
            for i=1:height(ann_table_events)
                if (ann_table_events.auxInfo(i) == 'VT' || ann_table_events.auxInfo(i) == 'VFL'
...
                        || ann_table_events.auxInfo(i) == 'VF')...
                        && ann_table_events.sampleNumber(i) < file_length
                    ep_start(k) = ann_table_events.sampleNumber(i);
                    if i<height(ann_table_events)
                        ep_end(k) = ann_table_events.sampleNumber(i+1);
                    else
                        ep_end(k) = file_length;
                    end
                    aug_label(k) = {ann_table_events.auxInfo(i)};
                    episode_id(k) = {strcat(database,'_', string(id),'_ep_',num2str(k))};
```

```
             k = k+1;
         end
     end
 end
 episode_table = table(episode_id', aug_label', ep_start', ep_end');
 episode_table.Properties.VariableNames = {'episode_id', 'aug_label', ...
     'ep_start', 'ep_end'};

%%
% CREATE EPISODE TABLE INCLUDING VFLS AND VFLE
 table_filter = {'VT' 'VFL' 'VF' 'VFLS' 'VFLE'}';
 table_filter_idx = ismember(ann_table_events.auxInfo, table_filter);
 ann_table_vfls = ann_table_events(table_filter_idx,:);

%aug_label = repmat({''},height(ann_table_vfls),1);
%episode_id = repmat({''},height(ann_table_vfls),1);
 [aug_label, episode_id] = deal({});
%ep_start = zeros(height(ann_table_vfls),1);
%ep_end = zeros(height(ann_table_vfls),1);
 [ep_start, ep_end] = deal([]);
 k=1;
 if strcmp(database, 'cudb')
     for i=1:height(ann_table_events)
         if (ann_table_events.auxInfo(i) == 'VT' || ann_table_events.auxInfo(i) == 'VFL'
...
                 || string(ann_table_events.typeMnemonic(i)) == '[' ...
                 || ann_table_events.auxInfo(i) == 'VFLS' ...
                 || ann_table_events.auxInfo(i) == 'VFLE')...
                 && ann_table_events.sampleNumber(i) < file_length
             ep_start(k) = ann_table_events.sampleNumber(i);
             if i<height(ann_table_events)
                 ep_end(k) = ann_table_events.sampleNumber(i+1);
             else
                 ep_end(k) = file_length;
             end
             aug_label(k) = {ann_table_events.auxInfo(i)};
             episode_id(k) = {strcat(database,'_', string(id),'_ep_',num2str(k))};
             k = k+1;
         end
     end
 else
     for i=1:height(ann_table_events)
         if (ann_table_events.auxInfo(i) == 'VT' || ann_table_events.auxInfo(i) == 'VFL'
...
                 || ann_table_events.auxInfo(i) == 'VF' ...
                 || ann_table_events.auxInfo(i) == 'VFLS' ...
                 || ann_table_events.auxInfo(i) == 'VFLE')...
                 && ann_table_events.sampleNumber(i) < file_length
             ep_start(k) = ann_table_events.sampleNumber(i);
```

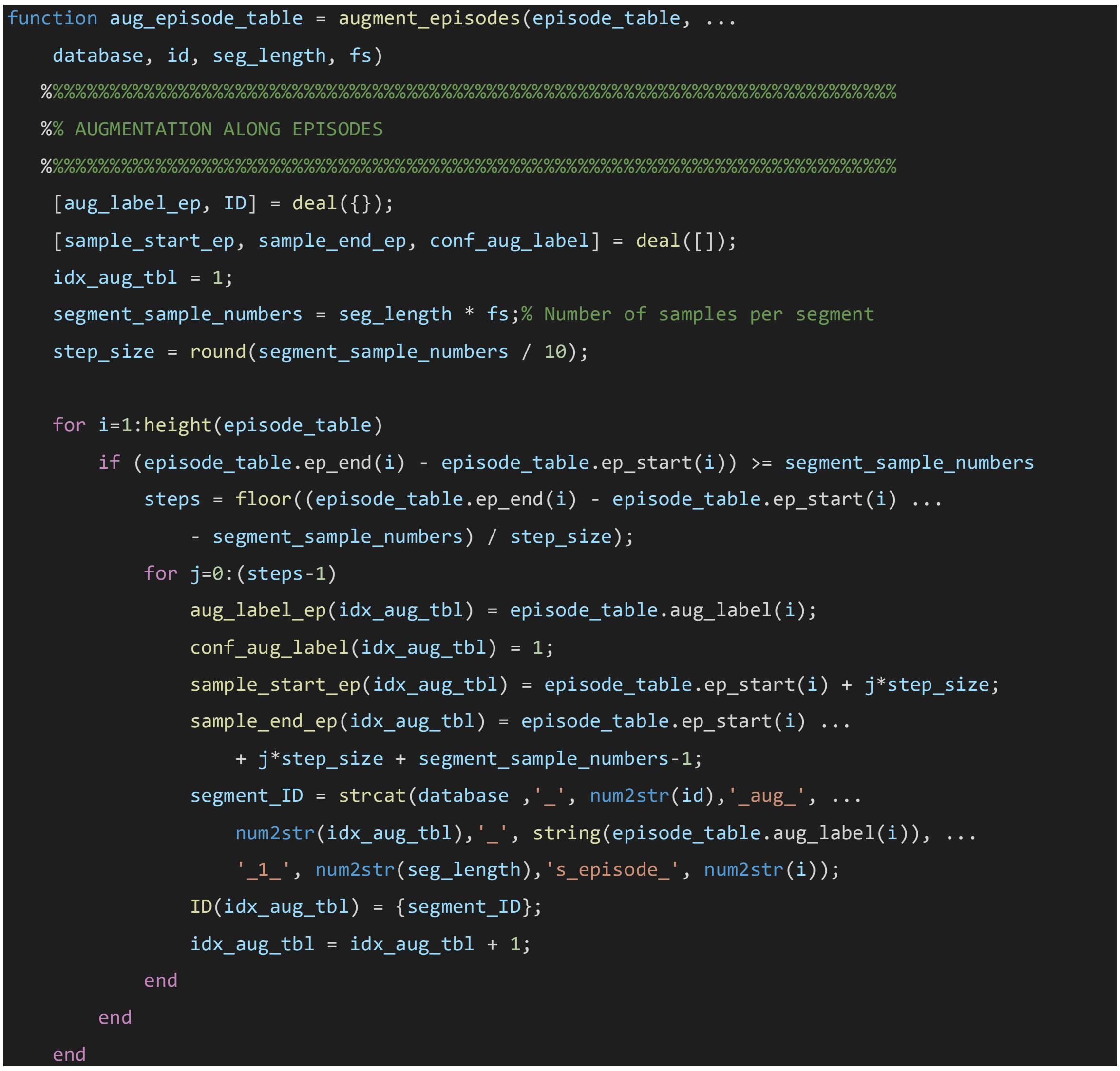
```
                    if i<height(ann_table_events)
                        ep_end(k) = ann_table_events.sampleNumber(i+1);
                    else
                        ep_end(k) = file_length;
                    end
                    aug_label(k) = {ann_table_events.auxInfo(i)};
                    episode_id(k) = {strcat(database,'_', string(id),'_ep_',num2str(k))};
                    k = k+1;
                end
            end
        end
        episode_VFLS_table = table(episode_id', aug_label', ep_start', ep_end');
        episode_VFLS_table.Properties.VariableNames = {'episode_id', 'aug_label', ...
            'ep_start', 'ep_end'};
    end
   %%
   % CREATE AUGMENTATION TABLE
    segment_sample_numbers = seg_length * fs;% Number of samples per segment
    step_size = round(segment_sample_numbers / 10);
    [type, aug_label] = deal({});
    [position, sample_start, sample_end] = deal([]);
    idx_aug_tbl = 1;

    for i=1:height(episode_table)
        if episode_table.ep_start(i) >= 2*segment_sample_numbers
            for j=0:9
                aug_label(idx_aug_tbl) = episode_table.aug_label(i);
                type(idx_aug_tbl) = {'start'};
                position(idx_aug_tbl) = -j;
                sample_start(idx_aug_tbl) = episode_table.ep_start(i) - j*step_size;
                sample_end(idx_aug_tbl) = episode_table.ep_start(i) - j*step_size + segment_sam-
ple_numbers - 1;
                idx_aug_tbl = idx_aug_tbl + 1;
            end
        end

        if episode_table.ep_end(i) >= 2*segment_sample_numbers && ...
                episode_table.ep_end(i) + segment_sample_numbers <= ...
                file_length
            for j=0:9
                aug_label(idx_aug_tbl) = episode_table.aug_label(i);
                type(idx_aug_tbl) = {'end'};
                position(idx_aug_tbl) = j;
                sample_start(idx_aug_tbl) = episode_table.ep_end(i) + j*step_size - segment_sam-
ple_numbers + 1;
                sample_end(idx_aug_tbl) = episode_table.ep_end(i) + j*step_size;
                idx_aug_tbl = idx_aug_tbl + 1;
            end
        end
```

```
 end

 aug_table = table(aug_label', type', position', sample_start', sample_end');
 aug_table.Properties.VariableNames = {'aug_label', 'type', 'position', 'start', 'end'};

%%
% Add ID, labels, and label confidences to augmentation table
 [ID, label_1, label_2, label_3, label_4] = deal(repmat({''},height(aug_table),1));
 [conf_aug_label, conf_1, conf_2, conf_3, conf_4] = deal(zeros(height(aug_table),1));

 for i=1:height(aug_table)
     segment_start = aug_table.start(i);
     segment_end = aug_table.end(i);
     segment = sample_labels(segment_start : segment_end);
    % count unique strings
    % unique_strings as a vector of unique elements in segment and idx_c as an
    % index vector referring to the index in unique_strings for each sample in segment
     [label, idx_a, idx_c] = unique(segment);
    % count repeating indices in idx_c and create a Count table in descending
    % order
     counts = accumarray(idx_c, 1);
     count_table = table(label, counts);
     count_table = sortrows(count_table,'counts', 'descend');

    % Calculate confidence value for the segment label
     idx_conf = ismember(count_table.label, string(aug_label(i)));
     conf_label = count_table.counts(idx_conf) / segment_sample_numbers;
     conf_aug_label(i) = conf_label;

    % Create ID containing database, file number, segment number,
    % segment length, label, label confidence, aug, start/end, Position
     segment_ID = strcat(database ,'_', num2str(id), '_augborder_', num2str(i),'_',...
         string(aug_label(i)), '_', string(conf_label), '_',...
         num2str(seg_length),'s_',string(type(i)), '_', string(position(i)));
     ID(i) = {segment_ID};

    % retreive the label with majority vote from count_table and calculate
    % label confidence
     label_1(i) = count_table.label(1);
     conf_1(i) = count_table.counts(1) / segment_sample_numbers;

    % Check for second label in segment
     if length(count_table.label)>=2
         label_2(i) = count_table.label(2);
         conf_2(i) = count_table.counts(2) / segment_sample_numbers;
     end
    % Check for third label in segment
     if length(count_table.label)>=3
         label_3(i) = count_table.label(3);
```



```
            conf_3(i) = count_table.counts(3) / segment_sample_numbers;
        end
       % Check for fourth label in segment
        if length(count_table.label)>=4
            label_4(i) = count_table.label(4);
            conf_4(i) = count_table.counts(4) / segment_sample_numbers;
        end
    end

    aug_border_table = table(ID, aug_label', conf_aug_label, type', ...
        position', sample_start', sample_end', ...
        label_1, label_2, label_3, label_4, conf_1, conf_2, conf_3, conf_4);
    aug_border_table.Properties.VariableNames = {'ID', 'aug_label', ...
        'conf_aug_label', 'type', 'position', 'start', 'end', ...
        'label_1', 'label_2', 'label_3', 'label_4', 'conf_1', 'conf_2', 'conf_3', 'conf_4'};
end
```

***Figure 48:*** *MATLAB augmentation function around episode borders with shockable conditions*



```
function aug_episode_table = augment_episodes(episode_table, ...
    database, id, seg_length, fs)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %% AUGMENTATION ALONG EPISODES
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [aug_label_ep, ID] = deal({});
    [sample_start_ep, sample_end_ep, conf_aug_label] = deal([]);
    idx_aug_tbl = 1;
    segment_sample_numbers = seg_length * fs;% Number of samples per segment
    step_size = round(segment_sample_numbers / 10);

    for i=1:height(episode_table)
        if (episode_table.ep_end(i) - episode_table.ep_start(i)) >= segment_sample_numbers
            steps = floor((episode_table.ep_end(i) - episode_table.ep_start(i) ...
                - segment_sample_numbers) / step_size);
            for j=0:(steps-1)
                aug_label_ep(idx_aug_tbl) = episode_table.aug_label(i);
                conf_aug_label(idx_aug_tbl) = 1;
                sample_start_ep(idx_aug_tbl) = episode_table.ep_start(i) + j*step_size;
                sample_end_ep(idx_aug_tbl) = episode_table.ep_start(i) ...
                    + j*step_size + segment_sample_numbers-1;
                segment_ID = strcat(database ,'_', num2str(id),'_aug_', ...
                    num2str(idx_aug_tbl),'_', string(episode_table.aug_label(i)), ...
                    '_1_', num2str(seg_length),'s_episode_', num2str(i));
                ID(idx_aug_tbl) = {segment_ID};
                idx_aug_tbl = idx_aug_tbl + 1;
            end
        end
    end
    aug_episode_table = table(ID', aug_label_ep', conf_aug_label', ...
        sample_start_ep', sample_end_ep');
    aug_episode_table.Properties.VariableNames = {'ID', 'aug_label', ...
        'conf_aug_label', 'start', 'end'};
end
```

***Figure 49:*** *MATLAB augmentation function along shockable episodes with 300 ms step size*

```
%% SEGMENTATION

seg_ind = 1:(seg_length*250):length(wt);%250 samples per second
label_vector=[];
fignames={};

for i=1:(length(seg_ind)-1)
%for i=1:2

    segment = abs(wt(:,seg_ind(i):(seg_ind(i+1)-1)));
   %Save the block as image
    figname = strcat(img_path,'/',database,'_',num2str(id),'_',num2str(i),'.jpeg');
    fignames{i}=figname;
    imwrite(segment,figname)
    label_vector(i)=sum(labels(seg_ind(i):(seg_ind(i+1)-1)))/(f);
end

for i=1:(length(seg_ind)-1)
    label_vector(i)=sum(labels(seg_ind(i):(seg_ind(i+1)-1)))/(f);
end

labeled_data=label_vector';
```

***Figure 50:*** *MATLAB segmentation and storing of spectrograms*

```
import pandas as pd
import os

###############################################################################
#%%LOAD CSV LABEL FILES INTO DFS
###############################################################################

databases = ['mitdb', 'cudb', 'vfdb', 'ahadb']
seg_length_str = '3s'
seg_length = 3
df_list = ['_episode_df', '_episode_VFLS_df', '_segment_df',
           '_aug_border_df', '_aug_episode_df']
fs = 250 # sampling frequency
```

```
for database in databases:

    #table_folder = f'E:\shr_detection\segments\{seg_length_str!s}\{database!s}\labels'
    wd = os.getcwd()
    table_folder = f'{wd!s}\{seg_length_str!s}\{database!s}\labels'
    file_list = [f'\{database!s}_episode_tbl.csv', f'\{database!s}_episode_VFLS_tbl.csv',
                 f'\{database!s}_segment_tbl.csv', f'\{database!s}_aug_border_tbl.csv',
                 f'\{database!s}_aug_episode_tbl.csv']
    for file, df in zip(file_list, df_list):

        df_name = f'{database!s}{df!s}'
        file_name = f'{table_folder!s}{file!s}'
        globals()[df_name] = pd.read_csv(file_name)

###############################################################################
#%% CREATE LABEL DFS
###############################################################################
thr = 0.6 #threshold
raw_df_list = ['_segment_df', '_aug_border_df', '_aug_episode_df']

for db in databases:
    df_seg = globals()[f'{db!s}_segment_df'][['ID', 'label_1',
                                                  'conf_1', 'start', 'end']]
    df_seg.rename(columns={'label_1':'label', 'conf_1':'conf'}, inplace=True)
    df_seg = df_seg[df_seg['conf'] >= thr]
    globals()[f'{db!s}_seg'] = df_seg

    df_border = globals()[f'{db!s}_aug_border_df'][['ID', 'aug_label', 'conf_aug_label',
                                    'start', 'end']]
    df_border.rename(columns={'aug_label':'label', 'conf_aug_label':'conf'},
                        inplace=True)
    df_border = df_border[df_border['conf'] >= thr]
    globals()[f'{db!s}_border'] = df_border

    df_ep = globals()[f'{db!s}_aug_episode_df']
    df_ep.rename(columns={'aug_label':'label', 'conf_aug_label':'conf'},
                    inplace=True)
    df_ep = df_ep[df_ep['conf'] >= thr]
    globals()[f'{db!s}_ep'] = df_ep
#%% OVERWRITE VFIB LABELS IN VFDB WITH VF (208 in total)
vfdb_seg.loc[vfdb_seg['label'] == 'VFIB', 'label'] = 'VF'

#%% DISCARD NOISE AND ASYSTOLIC SEGMENTS FROM VFDB
vfdb_seg = vfdb_seg[vfdb_seg['label'] != 'NOISE']
vfdb_seg = vfdb_seg[vfdb_seg['label'] != 'ASYS']
#test_counts = test['label'].value_counts()
#%%
un_lab_mitdb_seg = mitdb_seg['label'].value_counts()
un_lab_mitdb_border = mitdb_border['label'].value_counts()
```

```
un_lab_mitdb_ep = mitdb_ep['label'].value_counts()

un_lab_ahadb_seg = ahadb_seg['label'].value_counts()
un_lab_ahadb_border = ahadb_border['label'].value_counts()
un_lab_ahadb_ep = ahadb_ep['label'].value_counts()

un_lab_cudb_seg = cudb_seg['label'].value_counts()
un_lab_cudb_border = cudb_border['label'].value_counts()
un_lab_cudb_ep = cudb_ep['label'].value_counts()

un_lab_vfdb_seg = vfdb_seg['label'].value_counts()
un_lab_vfdb_border = vfdb_border['label'].value_counts()
un_lab_vfdb_ep = vfdb_ep['label'].value_counts()

#%%
# ORGANIZE MITDB SEGMENTS
# take all shr mitdb segments, reduce nshr from 28594 to 10000
# seperate shr from nshr
mitdb_seg_shr = mitdb_seg[(mitdb_seg['label'] == 'VT') |
                          (mitdb_seg['label'] == 'VFL')]
mitdb_seg_nshr = mitdb_seg[(mitdb_seg['label'] != 'VT') &
                          (mitdb_seg['label'] != 'VFL')]
# randomly choose 10000 nshr segments
mitdb_seg_nshr_r = mitdb_seg_nshr.sample(n=10000, random_state=1)

mitdb_conc = pd.concat([mitdb_seg_shr, mitdb_border, mitdb_ep,
                        mitdb_seg_nshr_r])
# ORGANIZE AHADB SEGMENTS
# in segments 3759 shr and 8234 nshr -> use all
# 124 border segments -> use all
# augmented episodes 37412 -> all of the same class VF-VFL
# -> reduce augmented shr episodes to 4000
ahadb_ep_r = ahadb_ep.sample(n=4000, random_state=1)
ahadb_conc = pd.concat([ahadb_ep_r, ahadb_border, ahadb_seg])

# ORGANIZE CUDB SEGMENTS
# in segments 1250 shr and 4632 nshr -> use all
# 412 border segments -> use all
# augmented episodes 12203 -> 12154 VF and 54 VT -> keep all VT and 3000 VF
# randomly chosen segments
cudb_ep_VT = cudb_ep[cudb_ep['label'] == 'VT']
cudb_ep_VF = cudb_ep[cudb_ep['label'] == 'VF']
cudb_ep_r = cudb_ep_VF.sample(n=3000, random_state=1)
cudb_conc = pd.concat([cudb_ep_VT, cudb_ep_r, cudb_border, cudb_seg])

# ORGANIZE VFDB SEGMENTS
# in segments 2503 shr and 12763 nshr -> use all
# 1659 border segments -> use all
# augmented episodes 23692 -> 5536 VF, 16593 VT and 1563 VFL
# -> keep 2500 VF, 5000 VT and all VFL
vfdb_ep_VF = vfdb_ep[vfdb_ep['label'] == 'VF']
vfdb_ep_VT = vfdb_ep[vfdb_ep['label'] == 'VT']
vfdb_ep_VFL = vfdb_ep[vfdb_ep['label'] == 'VFL']
vfdb_ep_VF_r = vfdb_ep_VF.sample(n=2500, random_state=1)
vfdb_ep_VT_r = vfdb_ep_VT.sample(n=5000, random_state=1)
vfdb_conc = pd.concat([vfdb_ep_VF_r, vfdb_ep_VT_r, vfdb_ep_VFL,
                       vfdb_border, vfdb_seg])

#%% CREATE CLASS MATRICES FOR WHOLE DATASET
#MITDB
s_mitdb_VT = mitdb_seg_shr[mitdb_seg_shr['label'] == 'VT']
s_mitdb_VFL = mitdb_seg_shr[mitdb_seg_shr['label'] == 'VFL']
s_mitdb_nshr = mitdb_seg_nshr_r.sample(n=10000, random_state=1)
s_mitdb_border_VT = mitdb_border[mitdb_border['label'] == 'VT']
s_mitdb_border_VFL = mitdb_border[mitdb_border['label'] == 'VFL']
s_mitdb_ep_VT = mitdb_ep[mitdb_ep['label'] == 'VT']
s_mitdb_ep_VFL = mitdb_ep[mitdb_ep['label'] == 'VFL']
s_mitdb = pd.concat([s_mitdb_VT, s_mitdb_VFL, s_mitdb_nshr, s_mitdb_border_VT,
                     s_mitdb_border_VFL, s_mitdb_ep_VT, s_mitdb_ep_VFL])
#AHADB
s_ahadb_seg_shr = ahadb_seg[ahadb_seg['label'] == 'VF-VFL']
s_ahadb_border_shr = ahadb_border[ahadb_border['label'] == 'VF-VFL']
s_ahadb_ep_shr = ahadb_ep[ahadb_ep['label'] == 'VF-VFL'].sample(n=4000,
                                                                random_state=1)
s_ahadb_seg_nshr = ahadb_seg[ahadb_seg['label'] != 'VF-VFL']
s_ahadb = pd.concat([s_ahadb_seg_shr, s_ahadb_border_shr, s_ahadb_ep_shr,
                     s_ahadb_seg_nshr])
#CUDB
s_cudb_seg_VF = cudb_seg[cudb_seg['label'] == 'VF']
s_cudb_seg_VT = cudb_seg[cudb_seg['label'] == 'VT']
s_cudb_nshr = cudb_seg[(cudb_seg['label']!='VF') & (cudb_seg['label']!='VT')]
s_cudb_border_VF = cudb_border[cudb_border['label'] == 'VF']
s_cudb_border_VT = cudb_border[cudb_border['label'] == 'VT']
s_cudb_ep_VF = cudb_ep[cudb_ep['label'] == 'VF'].sample(n=3000,
                                                         random_state=1)
s_cudb_ep_VT = cudb_ep[cudb_ep['label'] == 'VT']
s_cudb = pd.concat([s_cudb_seg_VF, s_cudb_seg_VT, s_cudb_border_VF,
                    s_cudb_border_VT, s_cudb_ep_VF, s_cudb_ep_VT,
                    s_cudb_nshr])
#VFDB
s_vfdb_seg_VF = vfdb_seg[vfdb_seg['label'] == 'VF']
s_vfdb_seg_VT = vfdb_seg[vfdb_seg['label'] == 'VT']
s_vfdb_seg_VFL = vfdb_seg[vfdb_seg['label'] == 'VFL']
s_vfdb_nshr = vfdb_seg[(vfdb_seg['label'] != 'VF') & (vfdb_seg['label'] != 'VT') &
                       (vfdb_seg['label'] != 'VFL')]
s_vfdb_border_VF = vfdb_border[vfdb_border['label'] == 'VF']
s_vfdb_border_VT = vfdb_border[vfdb_border['label'] == 'VT']
```

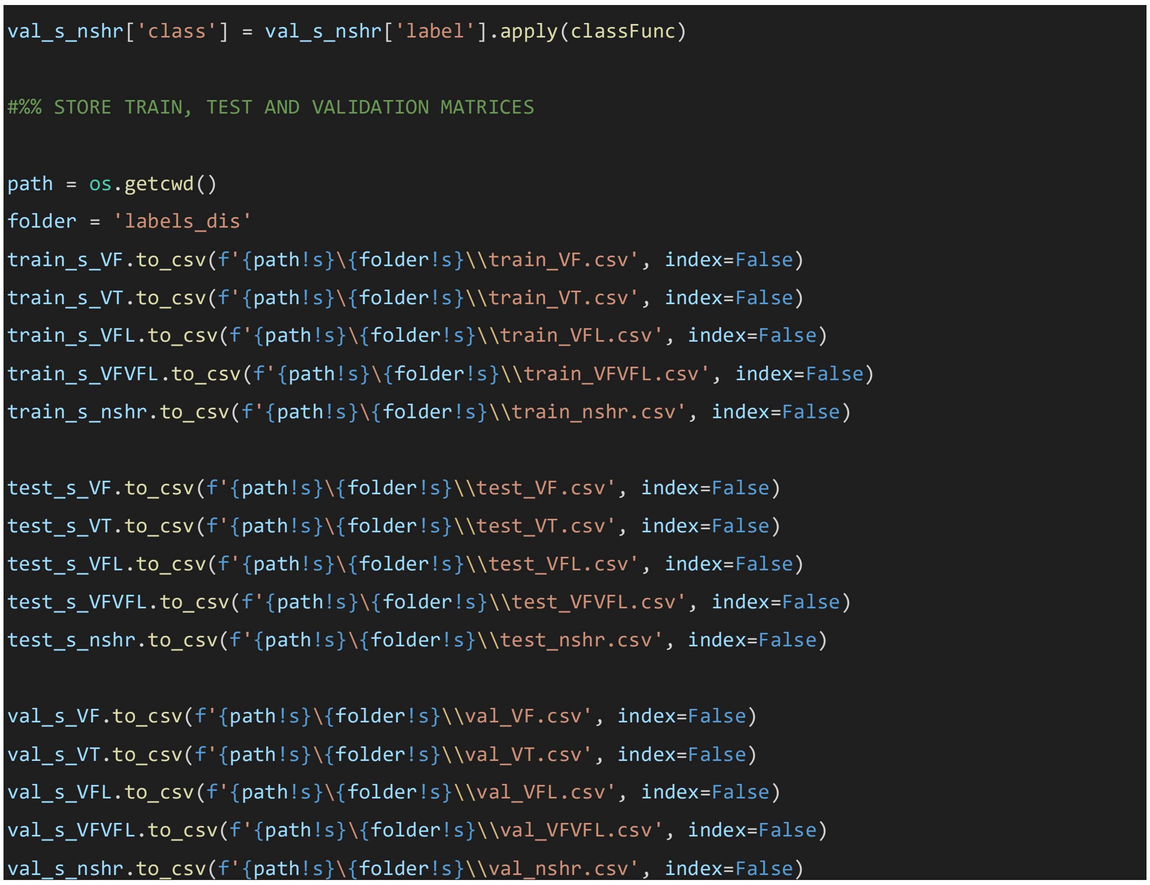
```
s_vfdb_border_VFL = vfdb_border[vfdb_border['label'] ==
'VFL']
s_vfdb_ep_VF = vfdb_ep_VF.sample(n=2500, random_state=1)
s_vfdb_ep_VT = vfdb_ep_VT.sample(n=5000, random_state=1)
s_vfdb_ep_VFL = vfdb_ep_VFL
s_vfdb = pd.concat([s_vfdb_seg_VF, s_vfdb_seg_VT, s_vfdb_seg_VFL,
                   s_vfdb_nshr, s_vfdb_border_VF, s_vfdb_border_VT,
                   s_vfdb_border_VFL, s_vfdb_ep_VF, s_vfdb_ep_VT,
                   s_vfdb_ep_VFL])
s_df = pd.concat([s_mitdb, s_ahadb, s_cudb, s_vfdb])
s_df = s_df.sample(frac=1, random_state=1)

#%%
# add classes: nshr = 0, VF = 1, VT = 2, VFL = 3, VF-VFL = 4
s_df['class'] = 0
def classFunc(lab):
    if lab == 'VF':
        return 1
    elif lab == 'VT':
        return 2
    elif lab == 'VFL':
        return 3
    elif lab == 'VF-VFL':
        return 4
    else:
        return 0

s_df['class'] = s_df['label'].apply(classFunc)
#%% STORE TO CSV
path = os.getcwd()
file = 'labels_discard.csv'
s_df.to_csv(f'{path!s}\{file!s}', index=False)

#%% CREATE LABEL MATRICES SEPERATELY FOR TRAIN AND FOR TEST SETS
frac = 0.15
val = 15/85
rnd_st = 1
#MITDB
test_s_mitdb_nshr = s_mitdb_nshr.sample(frac=frac, random_state=rnd_st)
train_s_mitdb_nshr = s_mitdb_nshr[~s_mitdb_nshr['ID'].isin(test_s_mitdb_nshr['ID'])]
val_s_mitdb_nshr = train_s_mitdb_nshr.sample(frac=val, random_state=rnd_st)
train_s_mitdb_nshr = train_s_mitdb_nshr[~train_s_mitdb_nshr['ID'].isin(val_s_mitdb_nshr['ID'])]

test_s_mitdb_VT = s_mitdb_VT.sample(frac=frac, random_state=rnd_st)
train_s_mitdb_VT = s_mitdb_VT[~s_mitdb_VT['ID'].isin(test_s_mitdb_VT['ID'])]
val_s_mitdb_VT = train_s_mitdb_VT.sample(frac=val, random_state=rnd_st)
train_s_mitdb_VT = train_s_mitdb_VT[~train_s_mitdb_VT['ID'].isin(val_s_mitdb_VT['ID'])]

test_s_mitdb_VFL = s_mitdb_VFL.sample(frac=frac, random_state=rnd_st)
```

```
train_s_mitdb_VFL = s_mitdb_VFL[~s_mitdb_VFL['ID'].isin(test_s_mitdb_VFL['ID'])]
val_s_mitdb_VFL = train_s_mitdb_VFL.sample(frac=val, random_state=rnd_st)
train_s_mitdb_VFL = train_s_mitdb_VFL[~train_s_mitdb_VFL['ID'].isin(val_s_mitdb_VFL['ID'])]

test_s_mitdb_border_VT = s_mitdb_border_VT.sample(frac=frac, random_state=rnd_st)
train_s_mitdb_border_VT = s_mitdb_border_VT[~s_mitdb_border_VT['ID'].isin(test_s_mitdb_bor-
der_VT['ID'])]
val_s_mitdb_border_VT = train_s_mitdb_border_VT.sample(frac=val, random_state=rnd_st)
train_s_mitdb_border_VT = train_s_mitdb_border_VT[~train_s_mitdb_bor-
der_VT['ID'].isin(val_s_mitdb_border_VT['ID'])]

test_s_mitdb_border_VFL = s_mitdb_border_VFL.sample(frac=frac, random_state=rnd_st)
train_s_mitdb_border_VFL = s_mitdb_border_VFL[~s_mitdb_border_VFL['ID'].isin(test_s_mitdb_bor-
der_VFL['ID'])]
val_s_mitdb_border_VFL = train_s_mitdb_border_VFL.sample(frac=val, random_state=rnd_st)
train_s_mitdb_border_VFL = train_s_mitdb_border_VFL[~train_s_mitdb_bor-
der_VFL['ID'].isin(val_s_mitdb_border_VFL['ID'])]

test_s_mitdb_ep_VT = s_mitdb_ep_VT.sample(frac=frac, random_state=rnd_st)
train_s_mitdb_ep_VT = s_mitdb_ep_VT[~s_mitdb_ep_VT['ID'].isin(test_s_mitdb_ep_VT['ID'])]
val_s_mitdb_ep_VT = train_s_mitdb_ep_VT.sample(frac=val, random_state=rnd_st)
train_s_mitdb_ep_VT =
train_s_mitdb_ep_VT[~train_s_mitdb_ep_VT['ID'].isin(val_s_mitdb_ep_VT['ID'])]

test_s_mitdb_ep_VFL = s_mitdb_ep_VFL.sample(frac=frac, random_state=rnd_st)
train_s_mitdb_ep_VFL = s_mitdb_ep_VFL[~s_mitdb_ep_VFL['ID'].isin(test_s_mitdb_ep_VFL['ID'])]
val_s_mitdb_ep_VFL = train_s_mitdb_ep_VFL.sample(frac=val, random_state=rnd_st)
train_s_mitdb_ep_VFL =
train_s_mitdb_ep_VFL[~train_s_mitdb_ep_VFL['ID'].isin(val_s_mitdb_ep_VFL['ID'])]

#AHADB
test_s_ahadb_nshr = s_ahadb_seg_nshr.sample(frac=frac, random_state=rnd_st)
train_s_ahadb_nshr = s_ahadb_seg_nshr[~s_ahadb_seg_nshr['ID'].isin(test_s_ahadb_nshr['ID'])]
val_s_ahadb_nshr = train_s_ahadb_nshr.sample(frac=val, random_state=rnd_st)
train_s_ahadb_nshr = train_s_ahadb_nshr[~train_s_ahadb_nshr['ID'].isin(val_s_ahadb_nshr['ID'])]

test_s_ahadb_seg_shr = s_ahadb_seg_shr.sample(frac=frac, random_state=rnd_st)
train_s_ahadb_seg_shr = s_ahadb_seg_shr[~s_ahadb_seg_shr['ID'].isin(test_s_ahadb_seg_shr['ID'])]
val_s_ahadb_seg_shr = train_s_ahadb_seg_shr.sample(frac=val, random_state=rnd_st)
train_s_ahadb_seg_shr =
train_s_ahadb_seg_shr[~train_s_ahadb_seg_shr['ID'].isin(val_s_ahadb_seg_shr['ID'])]

test_s_ahadb_ep_shr = s_ahadb_ep_shr.sample(frac=frac, random_state=rnd_st)
train_s_ahadb_ep_shr = s_ahadb_ep_shr[~s_ahadb_ep_shr['ID'].isin(test_s_ahadb_ep_shr['ID'])]
val_s_ahadb_ep_shr = train_s_ahadb_ep_shr.sample(frac=val, random_state=rnd_st)
train_s_ahadb_ep_shr =
train_s_ahadb_ep_shr[~train_s_ahadb_ep_shr['ID'].isin(val_s_ahadb_ep_shr['ID'])]

test_s_ahadb_border_shr = s_ahadb_border_shr.sample(frac=frac, random_state=rnd_st)
```

```
train_s_ahadb_border_shr = s_ahadb_border_shr[~s_ahadb_border_shr['ID'].isin(test_s_ahadb_bor-
der_shr['ID'])]
val_s_ahadb_border_shr = train_s_ahadb_border_shr.sample(frac=val, random_state=rnd_st)
train_s_ahadb_border_shr = train_s_ahadb_border_shr[~train_s_ahadb_bor-
der_shr['ID'].isin(val_s_ahadb_border_shr['ID'])]

#CUDB
test_s_cudb_nshr = s_cudb_nshr.sample(frac=frac, random_state=rnd_st)
train_s_cudb_nshr = s_cudb_nshr[~s_cudb_nshr['ID'].isin(test_s_cudb_nshr['ID'])]
val_s_cudb_nshr = train_s_cudb_nshr.sample(frac=val, random_state=rnd_st)
train_s_cudb_nshr = train_s_cudb_nshr[~train_s_cudb_nshr['ID'].isin(val_s_cudb_nshr['ID'])]

test_s_cudb_seg_VT = s_cudb_seg_VT.sample(frac=frac, random_state=rnd_st)
train_s_cudb_seg_VT = s_cudb_seg_VT[~s_cudb_seg_VT['ID'].isin(test_s_cudb_seg_VT['ID'])]
val_s_cudb_seg_VT = train_s_cudb_seg_VT.sample(frac=val, random_state=rnd_st)
train_s_cudb_seg_VT =
train_s_cudb_seg_VT[~train_s_cudb_seg_VT['ID'].isin(val_s_cudb_seg_VT['ID'])]

test_s_cudb_seg_VF = s_cudb_seg_VF.sample(frac=frac, random_state=rnd_st)
train_s_cudb_seg_VF = s_cudb_seg_VF[~s_cudb_seg_VF['ID'].isin(test_s_cudb_seg_VF['ID'])]
val_s_cudb_seg_VF = train_s_cudb_seg_VF.sample(frac=val, random_state=rnd_st)
train_s_cudb_seg_VF =
train_s_cudb_seg_VF[~train_s_cudb_seg_VF['ID'].isin(val_s_cudb_seg_VF['ID'])]

test_s_cudb_border_VT = s_cudb_border_VT.sample(frac=frac, random_state=rnd_st)
train_s_cudb_border_VT = s_cudb_border_VT[~s_cudb_border_VT['ID'].isin(test_s_cudb_bor-
der_VT['ID'])]
val_s_cudb_border_VT = train_s_cudb_border_VT.sample(frac=val, random_state=rnd_st)
train_s_cudb_border_VT = train_s_cudb_border_VT[~train_s_cudb_bor-
der_VT['ID'].isin(val_s_cudb_border_VT['ID'])]

test_s_cudb_border_VF = s_cudb_border_VF.sample(frac=frac, random_state=rnd_st)
train_s_cudb_border_VF = s_cudb_border_VF[~s_cudb_border_VF['ID'].isin(test_s_cudb_bor-
der_VF['ID'])]
val_s_cudb_border_VF = train_s_cudb_border_VF.sample(frac=val, random_state=rnd_st)
train_s_cudb_border_VF = train_s_cudb_border_VF[~train_s_cudb_bor-
der_VF['ID'].isin(val_s_cudb_border_VF['ID'])]

test_s_cudb_ep_VT = s_cudb_ep_VT.sample(frac=frac, random_state=rnd_st)
train_s_cudb_ep_VT = s_cudb_ep_VT[~s_cudb_ep_VT['ID'].isin(test_s_cudb_ep_VT['ID'])]
val_s_cudb_ep_VT = train_s_cudb_ep_VT.sample(frac=val, random_state=rnd_st)
train_s_cudb_ep_VT = train_s_cudb_ep_VT[~train_s_cudb_ep_VT['ID'].isin(val_s_cudb_ep_VT['ID'])]

test_s_cudb_ep_VF = s_cudb_ep_VF.sample(frac=frac, random_state=rnd_st)
train_s_cudb_ep_VF = s_cudb_ep_VF[~s_cudb_ep_VF['ID'].isin(test_s_cudb_ep_VF['ID'])]
val_s_cudb_ep_VF = train_s_cudb_ep_VF.sample(frac=val, random_state=rnd_st)
train_s_cudb_ep_VF = train_s_cudb_ep_VF[~train_s_cudb_ep_VF['ID'].isin(val_s_cudb_ep_VF['ID'])]

#VFDB
```

```
test_s_vfdb_nshr = s_vfdb_nshr.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_nshr = s_vfdb_nshr[~s_vfdb_nshr['ID'].isin(test_s_vfdb_nshr['ID'])]
val_s_vfdb_nshr = train_s_vfdb_nshr.sample(frac=val, random_state=rnd_st)
train_s_vfdb_nshr = train_s_vfdb_nshr[~train_s_vfdb_nshr['ID'].isin(val_s_vfdb_nshr['ID'])]

test_s_vfdb_seg_VT = s_vfdb_seg_VT.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_seg_VT = s_vfdb_seg_VT[~s_vfdb_seg_VT['ID'].isin(test_s_vfdb_seg_VT['ID'])]
val_s_vfdb_seg_VT = train_s_vfdb_seg_VT.sample(frac=val, random_state=rnd_st)
train_s_vfdb_seg_VT =
train_s_vfdb_seg_VT[~train_s_vfdb_seg_VT['ID'].isin(val_s_vfdb_seg_VT['ID'])]

test_s_vfdb_seg_VF = s_vfdb_seg_VF.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_seg_VF = s_vfdb_seg_VF[~s_vfdb_seg_VF['ID'].isin(test_s_vfdb_seg_VF['ID'])]
val_s_vfdb_seg_VF = train_s_vfdb_seg_VF.sample(frac=val, random_state=rnd_st)
train_s_vfdb_seg_VF =
train_s_vfdb_seg_VF[~train_s_vfdb_seg_VF['ID'].isin(val_s_vfdb_seg_VF['ID'])]

test_s_vfdb_seg_VFL = s_vfdb_seg_VFL.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_seg_VFL = s_vfdb_seg_VFL[~s_vfdb_seg_VFL['ID'].isin(test_s_vfdb_seg_VFL['ID'])]
val_s_vfdb_seg_VFL = train_s_vfdb_seg_VFL.sample(frac=val, random_state=rnd_st)
train_s_vfdb_seg_VFL =
train_s_vfdb_seg_VFL[~train_s_vfdb_seg_VFL['ID'].isin(val_s_vfdb_seg_VFL['ID'])]

test_s_vfdb_border_VT = s_vfdb_border_VT.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_border_VT = s_vfdb_border_VT[~s_vfdb_border_VT['ID'].isin(test_s_vfdb_bor-
der_VT['ID'])]
val_s_vfdb_border_VT = train_s_vfdb_border_VT.sample(frac=val, random_state=rnd_st)
train_s_vfdb_border_VT = train_s_vfdb_border_VT[~train_s_vfdb_bor-
der_VT['ID'].isin(val_s_vfdb_border_VT['ID'])]

test_s_vfdb_border_VF = s_vfdb_border_VF.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_border_VF = s_vfdb_border_VF[~s_vfdb_border_VF['ID'].isin(test_s_vfdb_bor-
der_VF['ID'])]
val_s_vfdb_border_VF = train_s_vfdb_border_VF.sample(frac=val, random_state=rnd_st)
train_s_vfdb_border_VF = train_s_vfdb_border_VF[~train_s_vfdb_bor-
der_VF['ID'].isin(val_s_vfdb_border_VF['ID'])]

test_s_vfdb_border_VFL = s_vfdb_border_VFL.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_border_VFL = s_vfdb_border_VFL[~s_vfdb_border_VFL['ID'].isin(test_s_vfdb_bor-
der_VFL['ID'])]
val_s_vfdb_border_VFL = train_s_vfdb_border_VFL.sample(frac=val, random_state=rnd_st)
train_s_vfdb_border_VFL = train_s_vfdb_border_VFL[~train_s_vfdb_bor-
der_VFL['ID'].isin(val_s_vfdb_border_VFL['ID'])]

test_s_vfdb_ep_VT = s_vfdb_ep_VT.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_ep_VT = s_vfdb_ep_VT[~s_vfdb_ep_VT['ID'].isin(test_s_vfdb_ep_VT['ID'])]
val_s_vfdb_ep_VT = train_s_vfdb_ep_VT.sample(frac=val, random_state=rnd_st)
train_s_vfdb_ep_VT = train_s_vfdb_ep_VT[~train_s_vfdb_ep_VT['ID'].isin(val_s_vfdb_ep_VT['ID'])]
```

```
test_s_vfdb_ep_VF = s_vfdb_ep_VF.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_ep_VF = s_vfdb_ep_VF[~s_vfdb_ep_VF['ID'].isin(test_s_vfdb_ep_VF['ID'])]
val_s_vfdb_ep_VF = train_s_vfdb_ep_VF.sample(frac=val, random_state=rnd_st)
train_s_vfdb_ep_VF = train_s_vfdb_ep_VF[~train_s_vfdb_ep_VF['ID'].isin(val_s_vfdb_ep_VF['ID'])]

test_s_vfdb_ep_VFL = s_vfdb_ep_VFL.sample(frac=frac, random_state=rnd_st)
train_s_vfdb_ep_VFL = s_vfdb_ep_VFL[~s_vfdb_ep_VFL['ID'].isin(test_s_vfdb_ep_VFL['ID'])]
val_s_vfdb_ep_VFL = train_s_vfdb_ep_VFL.sample(frac=val, random_state=rnd_st)
train_s_vfdb_ep_VFL =
train_s_vfdb_ep_VFL[~train_s_vfdb_ep_VFL['ID'].isin(val_s_vfdb_ep_VFL['ID'])]

#CONCAT
train_s_VF = pd.concat([train_s_cudb_border_VF, train_s_cudb_ep_VF,
                        train_s_cudb_seg_VF,
                        train_s_vfdb_border_VF, train_s_vfdb_ep_VF,
                        train_s_vfdb_seg_VF])
train_s_VF['class'] = train_s_VF['label'].apply(classFunc)

test_s_VF = pd.concat([test_s_cudb_border_VF, test_s_cudb_ep_VF,
                        test_s_cudb_seg_VF,
                        test_s_vfdb_border_VF, test_s_vfdb_ep_VF,
                        test_s_vfdb_seg_VF])
test_s_VF['class'] = test_s_VF['label'].apply(classFunc)

val_s_VF = pd.concat([val_s_cudb_border_VF, val_s_cudb_ep_VF,
                        val_s_cudb_seg_VF,
                        val_s_vfdb_border_VF, val_s_vfdb_ep_VF,
                        val_s_vfdb_seg_VF])
val_s_VF['class'] = val_s_VF['label'].apply(classFunc)


train_s_VT = pd.concat([train_s_cudb_border_VT, train_s_cudb_ep_VT,
                        train_s_cudb_seg_VT,
                        train_s_mitdb_VT, train_s_mitdb_border_VT,
                        train_s_mitdb_ep_VT,
                        train_s_vfdb_border_VT, train_s_vfdb_ep_VT,
                        train_s_vfdb_seg_VT])
train_s_VT['class'] = train_s_VT['label'].apply(classFunc)

test_s_VT = pd.concat([test_s_cudb_border_VT, test_s_cudb_ep_VT,
                        test_s_cudb_seg_VT,
                        test_s_mitdb_VT, test_s_mitdb_border_VT,
                        test_s_mitdb_ep_VT,
                        test_s_vfdb_border_VT, test_s_vfdb_ep_VT,
                        test_s_vfdb_seg_VT])
test_s_VT['class'] = test_s_VT['label'].apply(classFunc)

val_s_VT = pd.concat([val_s_cudb_border_VT, val_s_cudb_ep_VT,
                        val_s_cudb_seg_VT,
                        val_s_mitdb_VT, val_s_mitdb_border_VT,
                        val_s_mitdb_ep_VT,
                        val_s_vfdb_border_VT, val_s_vfdb_ep_VT,
                        val_s_vfdb_seg_VT])
val_s_VT['class'] = val_s_VT['label'].apply(classFunc)


train_s_VFL = pd.concat([train_s_mitdb_VFL, train_s_mitdb_border_VFL,
                         train_s_mitdb_ep_VFL,
                         train_s_vfdb_border_VFL, train_s_vfdb_ep_VFL,
                         train_s_vfdb_seg_VFL])
train_s_VFL['class'] = train_s_VFL['label'].apply(classFunc)

test_s_VFL = pd.concat([test_s_mitdb_VFL, test_s_mitdb_border_VFL,
                         test_s_mitdb_ep_VFL,
                         test_s_vfdb_border_VFL, test_s_vfdb_ep_VFL,
                         test_s_vfdb_seg_VFL])
test_s_VFL['class'] = test_s_VFL['label'].apply(classFunc)

val_s_VFL = pd.concat([val_s_mitdb_VFL, val_s_mitdb_border_VFL,
                         val_s_mitdb_ep_VFL,
                         val_s_vfdb_border_VFL, val_s_vfdb_ep_VFL,
                         val_s_vfdb_seg_VFL])
val_s_VFL['class'] = val_s_VFL['label'].apply(classFunc)


train_s_VFVFL = pd.concat([train_s_ahadb_border_shr, train_s_ahadb_ep_shr,
                            train_s_ahadb_seg_shr])
train_s_VFVFL['class'] = train_s_VFVFL['label'].apply(classFunc)

test_s_VFVFL = pd.concat([test_s_ahadb_border_shr, test_s_ahadb_ep_shr,
                            test_s_ahadb_seg_shr])
test_s_VFVFL['class'] = test_s_VFVFL['label'].apply(classFunc)

val_s_VFVFL = pd.concat([val_s_ahadb_border_shr, val_s_ahadb_ep_shr,
                            val_s_ahadb_seg_shr])
val_s_VFVFL['class'] = val_s_VFVFL['label'].apply(classFunc)


train_s_nshr = pd.concat([train_s_ahadb_nshr, train_s_cudb_nshr,
                          train_s_mitdb_nshr, train_s_vfdb_nshr])
train_s_nshr['class'] = train_s_nshr['label'].apply(classFunc)

test_s_nshr = pd.concat([test_s_ahadb_nshr, test_s_cudb_nshr,
                          test_s_mitdb_nshr, test_s_vfdb_nshr])
test_s_nshr['class'] = test_s_nshr['label'].apply(classFunc)

val_s_nshr = pd.concat([val_s_ahadb_nshr, val_s_cudb_nshr,
                          val_s_mitdb_nshr, val_s_vfdb_nshr])
```



```
val_s_nshr['class'] = val_s_nshr['label'].apply(classFunc)

#%% STORE TRAIN, TEST AND VALIDATION MATRICES

path = os.getcwd()
folder = 'labels_dis'
train_s_VF.to_csv(f'{path!s}\{folder!s}\\train_VF.csv', index=False)
train_s_VT.to_csv(f'{path!s}\{folder!s}\\train_VT.csv', index=False)
train_s_VFL.to_csv(f'{path!s}\{folder!s}\\train_VFL.csv', index=False)
train_s_VFVFL.to_csv(f'{path!s}\{folder!s}\\train_VFVFL.csv', index=False)
train_s_nshr.to_csv(f'{path!s}\{folder!s}\\train_nshr.csv', index=False)

test_s_VF.to_csv(f'{path!s}\{folder!s}\\test_VF.csv', index=False)
test_s_VT.to_csv(f'{path!s}\{folder!s}\\test_VT.csv', index=False)
test_s_VFL.to_csv(f'{path!s}\{folder!s}\\test_VFL.csv', index=False)
test_s_VFVFL.to_csv(f'{path!s}\{folder!s}\\test_VFVFL.csv', index=False)
test_s_nshr.to_csv(f'{path!s}\{folder!s}\\test_nshr.csv', index=False)

val_s_VF.to_csv(f'{path!s}\{folder!s}\\val_VF.csv', index=False)
val_s_VT.to_csv(f'{path!s}\{folder!s}\\val_VT.csv', index=False)
val_s_VFL.to_csv(f'{path!s}\{folder!s}\\val_VFL.csv', index=False)
val_s_VFVFL.to_csv(f'{path!s}\{folder!s}\\val_VFVFL.csv', index=False)
val_s_nshr.to_csv(f'{path!s}\{folder!s}\\val_nshr.csv', index=False)
```

***Figure 51:*** *Python script for dataset management*