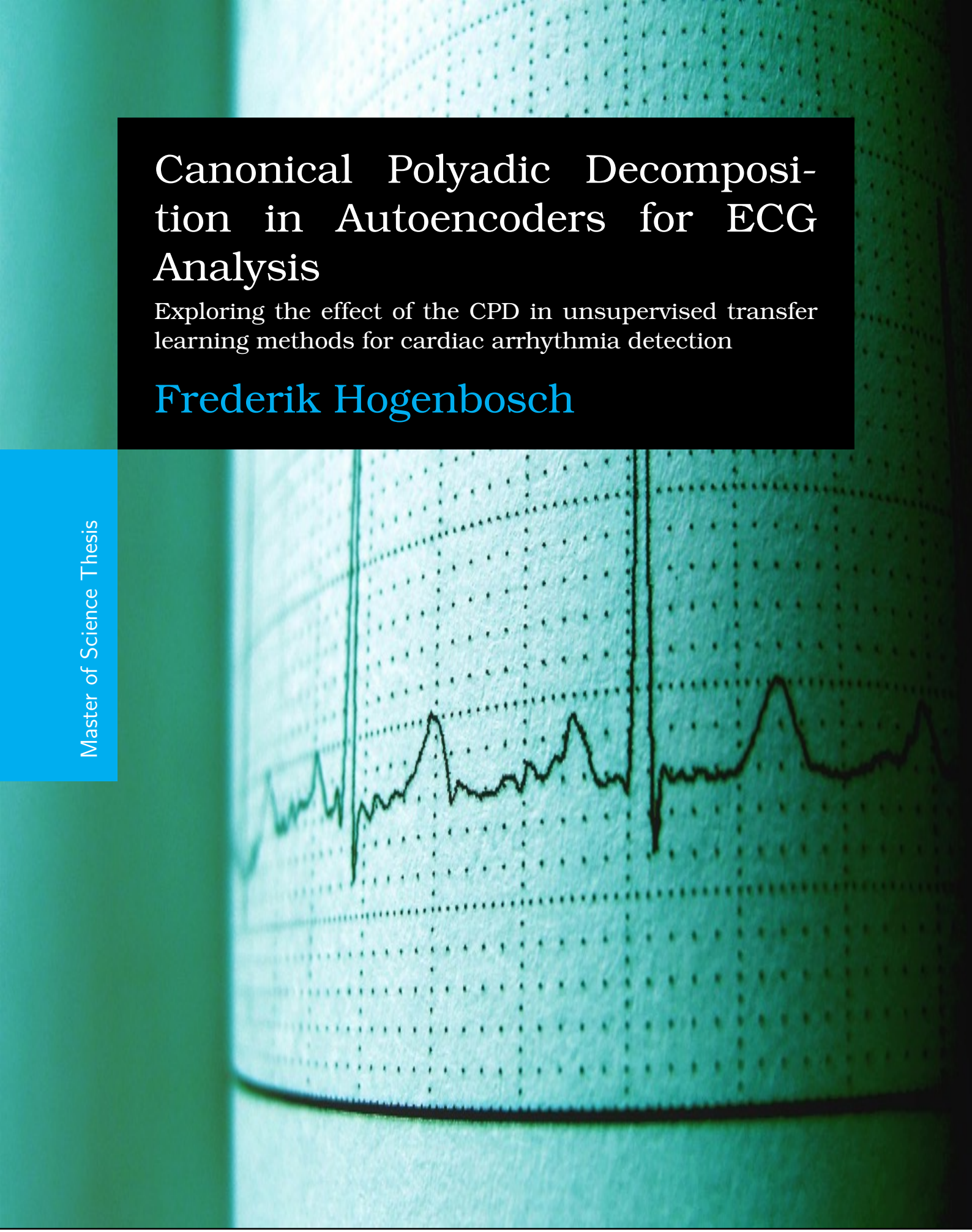
Canonical Polyadic Decomposition in Autoencoders for ECG Analysis

Exploring the effect of the CPD in unsupervised transfer learning methods for cardiac arrhythmia detection

Frederik Hogenbosch

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Canonical Polyadic Decomposition in Autoencoders for ECG Analysis
Exploring the effect of the CPD in unsupervised transfer learning methods for cardiac arrhythmia detection

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Frederik Hogenbosch

June 21, 2024

Faculty of Mechanical Engineering (ME) · Delft University of Technology

Abstract

This thesis studies the application of the Canonical Polyadic Decomposition (CPD) in unsupervised transfer learning methods for cardiac arrhythmia detection. Unsupervised learning methods have become more prevalent in the healthcare sector due to the abundance of unlabeled data. Labeling of medical data is often non-trivial as it is labor-intensive and requires expert knowledge. Transfer learning can utilize the large number of unlabeled data by extracting relevant features, which can in turn be used for a smaller supervised learning part. Furthermore, in the medical field, AI models are often deployed on embedded systems, requiring efficient model architectures while maintaining high diagnostic performance.

The unsupervised transfer learning method was designed with an autoencoder for the unsupervised part and a linear network for the supervised part. The first experiment explored four models to select the most optimal architecture to apply the CPD to. These models include an autoencoder adaptation of the ResNet and ConvNeXt models, the U-Net autoencoder and a basic implementation of a convolutional autoencoder. In the second experiment, the autoencoder model is decomposed using the CPD and evaluated at various compression ratios on its reconstruction capabilities, classification accuracy and computational performance. The CPD implementation is also tested on its convergence speed and data efficiency as compared to its uncompressed counterpart.

The first experiment found that the basic implementation of a convolutional autoencoder performed best overall. The U-Net model had high reconstruction quality, however lacked the predictive accuracy. The ResNet model was found to have slightly worse reconstruction and prediction capabilities while having a larger parameter count. The ConvNeXt model failed to accurately reconstruct the images.

The second experiment showed the CP-decomposed model approached the uncompressed model in terms of predictive capabilities, while having lower reconstruction qualities. This is likely due to the regularization effect of the CPD, suggesting significant redundancy in the uncompressed model. Despite the reduction in forward pass FLOPs for the CP-decomposed model, it was found that both the computational complexity of the backpropagation process was higher than the uncompressed model at the lower compression ratios and that the memory allocation suffered a significant increase. This resulted in longer and less efficient training of the CP-decomposed models. It was furthermore found that the CP-decomposed models converged faster and had higher data efficiency as compared to the uncompressed model.

Table of Contents

Preface & Acknowledgements **xi**

1 Introduction **1**
1-1 Problem formulation . . . 2
1-2 Thesis outline . . . 3
1-3 Implementation . . . 4

2 Background **5**
2-1 Electrocardiography . . . 5
2-1-1 Cardiovascular system . . . 5
2-1-2 Cardiac cycle . . . 6
2-1-3 Electrocardiography . . . 7
2-1-4 Arrhythmia . . . 8
2-2 Tensors and the Canonical Polyadic Decompostion . . . 10
2-2-1 Preliminaries . . . 10
2-2-2 Tensor Properties . . . 11
2-2-3 Canonical Polyadic Decomposition . . . 12
2-3 Convolutional Autoencoders . . . 14
2-3-1 Autoencoders . . . 14
2-3-2 Convolutional neural networks . . . 15
2-3-3 Convolutional Layer . . . 17
2-3-4 Convolutional Autoencoder Network . . . 18
2-3-5 The Canonical Polyadic Decomposition in CNNs . . . 19
2-3-6 Optimization methods . . . 21

3 Methodology **23**
3-1 Data . . . 23
3-1-1 Datasets . . . 24
3-1-2 Preprocessing . . . 26
3-2 Experiment 1: Autoencoder model comparison . . . 30
3-2-1 Model introductions . . . 30

3-3 Experiment 2: Tensor decomposed autoencoder . . . 37
3-3-1 Experiment 2.1: Accuracy comparison over various compression ratios . . . 38
3-3-2 Experiment 2.2: Convergence speed comparison . . . 38
3-3-3 Experiment 2.3: Data efficiency comparison . . . 38
3-3-4 Evaluation Metrics . . . 39
3-4 Experimental setup . . . 39

4 Results and discussion **43**
4-1 Experiment 1: Autoencoder model comparison . . . 43
4-1-1 Error comparison . . . 43
4-1-2 Image reconstruction comparison . . . 43
4-1-3 Performance comparison . . . 45
4-1-4 Classification comparison . . . 46
4-1-5 Model recommendations . . . 46
4-2 Experiment 2: CP-decomposed autoencoder . . . 48
4-2-1 Experiment 2.1: Accuracy comparison over various compression ratios . . . 48
4-2-2 Experiment 2.2: Convergence speed comparison . . . 51
4-2-3 Experiment 2.3: Data efficiency comparison . . . 52

5 Conclusion **55**
5-1 Recommendations for future research . . . 57

A Training Settings **59**
A-1 Experiment 1 . . . 59
A-1-1 Autoencoder settings . . . 59
A-1-2 Classifier settings . . . 60
A-2 Experiment 2 . . . 60

B Backpropagation Complexity **61**
B-1 Convolution . . . 61
B-2 CP-decomposed convolution . . . 63

Glossary **71**

List of Figures

2-1 Overview of the heart indicating its most essential components. Located at the top of the heart are the left and right atria. On the bottom are the left and right ventricles. Obtained from Herring and Paterson [1]. 6

2-2 Overview of the cardiac conduction system. The signal originates at the top of the right atria in the SA node, after which it follows the AV node towards the bundles of His, dissipating via the bundle branches. Obtained from Herring and Paterson [1]. . . 7

2-3 On the left the placement of the three limb and six chest electrodes is shown. On the right the resulting angles and ECG examples are given. Obtained from Herring and Paterson [1]. 8

2-4 Electrical activity of one cardiac cycle showing the P- and T-wave and the QRS complex. Obtained from Herring and Paterson [1]. 8

2-5 Graphical representation of the fibers of a tensor over mode-1 in (a), mode-2 in (b) and mode-3 in (c). Obtained from Kolda and Bader [2]. 10

2-6 Graphical representation of the horizontal (a), lateral (b) and frontal(c) slices of a tensor. Obtained from Kolda and Bader [2]. 11

2-7 Rank one tensor visualization. Tensor $\underline{\boldsymbol{X}}$ is decomposed into three vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. Modified image with original from Kolda and Bader [2]. 12

2-8 Block diagram of a CPD of a third order tensor. The original tensor $\underline{\boldsymbol{X}}$ on the left is decomposed in a sum of rank R outer product of vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. On the right the factor matrix representation is given. Obtained from Cichocki et al. [3]. 13

2-9 Example of MNIST autoencoder architecture. The original input image is compressed by the encoder to a lower dimensional latent space representation, after which it is reconstructed back to its original image by the decoder. Obtained from Bank et al. [4]. 14

2-10 Example of a 2D convolution operation. The feature map on the left is multiplied by the upper left quarter of the input, resulting in the top left value in the right most matrix. 16

2-11 GELU activation function. For values below 0 the GELU function approaches 0 with a slight drop before the value of 0. For values higher than 0 the function returns the input. This introduces a nonlinearity in the model. 18

2-12 Example diagram of a simple implementation of a CNN. The input image of size 128×128 is compressed by three convolutional layers to a latent space representation $\mathbf{z}$. The decoder reconstructs the lower dimensional representation through a sequence of upsampling and convolutional layers back to its original size. 19

2-13 Example of a 2×2 max pooling operation. The maximum value of the upper left quarter is selected for the downsampled feature map. 19

2-14 Visualization of a convolution operation with a kernel size of $S \times T \times d \times d$. 19

2-15 Tensor decomposed convolutional operation into four subsequent steps. The input channels dimension S is firstly mapped to R, which is later mapped to the number of output channels T. The spatial dimensions are mapped separately instead of combined. In the illustration d is used to indicate the kernel size, instead of D used in the thesis. 20

3-1 Fourier transform of random selected input signal before filtering. The bandpass filter will use a lower cutoff of 0.1Hz and a high cutoff of 100Hz. Frequencies over 100Hz contribute less to the signal due to their have low magnitudes. 26

3-2 Bandpass filtered signal overlayed on unfiltered signal. The effect of the highpass filter can be seen by smaller deviations from the mean. The lowpass filter reduces the noise. 27

3-3 Peak indication after the detection algorithm. 28

3-4 Normalized segment with a cutoff of 100 samples to the left of the R-peak and 200 samples to the right. 28

3-5 Random sample of input images. Each sample has varying amounts of noise and defects. 29

3-6 Basic model architecture consisting of three layers for both the encoder and decoder. Each layer consists of two convolutions and a down- or upsampling operation. 31

3-7 Overview of the Residual Block from the ResNet architecture containing two convolutions. These are followed by a skip connection and an activation layer. 32

3-8 ResNet architecture featuring three layers per encoder and decoder, where the second and third layer contain two ResNet Blocks each. 32

3-9 Overview of the ConvNeXt Residual Block. The input is first normalized over its layers, after which it is subjected to a high kernel size convolution, followed by an activation layer. An expanding pointwise convolutions maps the intermediate image to 4 times the number of input channels. The GELU activation function is applied before it is convoluted back to its original number of input channels. Finally, the skip connection adds the original input image to the output. 33

3-10 ConvNext autoencoder architecture. The model consists of ConvNeXt block in each layer of the encoder and decoder. Instead of max pooling in the encoder, the architecture used downsampling layers to reduce the spatial dimensions. 34

3-11 The UNet architecture uses three convolutions per layer instead of the regular two. It furthermore employs skip connections between the respective layers in the encoder and decoder. the concatenated feature maps are indicated by the white boxes. . . . 35

3-12 Classifier architecture with latent space input. The network consists of an input layer with dimension $16 \times 16 \times T$, a single hidden layer with dimension 256 and output layer with the five classes: F, N, Q, S and V. Example shows only a few connections for readability, but each neuron is connected. 35

3-13 Loss function for both the SGD and Adam optimizer. The SGD loss has a smoother optimization landscape, but converges to a much higher MSE. The Adam optimizer has a much steeper decline, but is less smooth. 40

4-1 Three run averaged validation loss for the basic, ResNet, ConvNeXt and U-Net models with error bars. The U-Net model has the lowest convergence point. The ConvNeXt and ResNet model both convergence quickly. 44

4-2 Input image 45

4-3 Reconstructed images for the basic, U-Net, ResNet and ConvNeXt model. The basic and U-Net model are able to reconstruct the high frequencies of the singal. The ResNet model has more trouble with the higher frequencies. The ConvNeXt model is only able to reconstruct the general contours of the signal. 45

4-4 Input image and uncompressed reconstruction. 49

4-5 Reconstructed images at various compression ratios. Higher compression ratios have difficulty with reconstructing the high dimensional functions. This gets increasingly better as the compression ratio is lowered. 49

4-6 Triple run average accuracy of tensor decomposed autoencoder models at various compression ratio compared to the baseline uncompressed autoencoder model. . . . 50

4-7 Three run average of the uncompressed autoencoder validation loss with error bars. The CP-decomposed model convergences after 15 epochs, whereas the uncompressed model fails to converge within the 25 epochs. 51

4-9 MSE of test set for both the uncompressed model and CP-decomposed model at reduction rates of 0.025, 0.05, 0.1 and 0.2. The full training set test set MSEs for both models are indicated by the dashed lines. 52

4-8 Validation loss during training for both the uncompressed (left) and CP-decomposed (right) models at various compression ratios of 0.025, 0.05, 0.1, and 0.2. The full training set validation loss is given in the orange plots. 53

B-1 Unfolding process the of $\mathbf{X}$ in a convolution operation with a 2×2 kernel. The blue shaded elements indicate the unfolding of that specific step. 62

List of Tables

3-1 Hypotheses for the second experiment. 23
3-2 Number of beats of selected datasets. 24
3-3 Heartbeat classes 25
3-4 MIT-BIH dataset inter-patient division. 25
3-5 Data distribution for the labeled MIT-BIH and INCART datasets. 25

4-1 Averaged MSE on the test set of each of the models. 44
4-2 Number of parameters, epoch duration and FLOPs for the basic, ResNet, ConvNeXt and U-Net models. 45
4-3 Classification accuracy for the basic, ResNet, ConvNeXt and U-Net models. 46
4-4 Confusion matrix for basic model. 46
4-5 CP-rank, number of parameters, epoch duration and forward pass FLOPs for the uncompressed and compressed models. 50

A-1 Basic model training settings 59
A-2 ResNet training settings 59
A-3 ConvNeXT training settings 59
A-4 U-Net training settings 59
A-5 Classifier training settings 60
A-6 CP-decomposed autoencoder training settings 60
A-7 Classifier training settings 60

B-1 Dimension for the unfolded matrices. 63
B-2 Matrix dimension for the R to T convolution step in the CP-decoposed convolution operation. 63
B-3 Matrix dimension for the depthwise convolution steps in the CP-decoposed convolution operation. 64
B-4 Matrix dimension for the S to R convolution step in the CP-decoposed convolution operation. 64

Preface & Acknowledgements

This thesis came into being through a project set out by dr. de Jong, who was interested in developing an ECG classification algorithm for use in the healthcare sector. The research began with selecting the state-of-the-art algorithms and what their corresponding challenges are when applied in clinical settings. From that point on the goal was to improve on those limitations and hopefully be able to increase the performance of the state-of-the-art algorithms.

I would like to thank both dr. ir. Batselier and dr. de Jong for their guidance throughout these last nine months. This thesis has been highly educational for me and that was greatly due to your expertise.

"Turn and face the strange."
— *David Bowie*

Chapter 1

Introduction

Artificial Intelligence (AI) has rapidly evolved from a futuristic concept to a technology reshaping countless industries. One notable area that has experienced a transformation due to AI in recent years is the healthcare sector. The advantage of AI in the medical world lies in its ability to process large amounts of data quickly and accurately, making it a valuable tool for improving diagnostic accuracy.

The advancement of AI in medical diagnosis has been achieved through the application of deep learning methods. These algorithms analyze large datasets to recognize patterns and make predictions accordingly. Deep learning algorithms can analyze medical data, such as medical imaging, genetic information and health records, to detect diseases with greater precision than traditional methods. For example, AI has demonstrated success in the field of radiology, where it assists in interpreting imaging studies like X-rays, CT scans, and MRIs.[5] The detection of abnormalities such as tumors or fractures in these medical images is achieved with high accuracy, sometimes even surpassing medical experts.[6]

Another area where AI shows potential is in electrocardiography (ECG) analysis, which is essential for diagnosing various cardiac arrhythmia. ECGs are among the most widely used methods in diagnosing heart diseases due to their low usage costs and limited impact on the patient.[7] ECGs monitor the amplitude and direction of the electrical signal during a contraction cyclus of the heart, which contains valuable information about the functioning of the heart. A valid classification is often crucial in determining the next course of action, whether that is sending the patient to the correct hospital or to do further research in a specific area.[8]

Despite the promising developments, the implementation of AI in ECG analysis and other medical fields faces several challenges. One major issue is the abundance of unlabeled data in the healthcare sector due to the labeling process being labor-intensive and requiring expert knowledge. This makes it a time-consuming and costly process. There is however a necessity of labeled data for the training of AI.[9]

Additionally, for AI models to be effective in medical environments, they must be able to generalize well to new patients who were not included in the training set. For this purpose, training data is often split inter-patiently, meaning data is sorted based on the patient.[10]

To address these challenges, transfer learning methods are being explored, which split models up into the subsequent optimization of two different deep learning models. For the purpose of ECG classification it would be beneficial to extract features from the ECG using only unlabeled data in the first part, which can in turn be used by the second part to do the classification using a smaller set of labeled data.

1-1 Problem formulation

For the first part of the transfer learning method, an autoencoder will be used to extract features from the ECG images. These features will then be subsequently used as the input to a conventional neural network for the classification part.
Autoencoders are unsupervised neural networks that consist of an encoder and a decoder part. The encoder takes an input $\mathbf{x}$ and compresses the data to a lower dimensional latent space $\mathbf{z}$. The decoder then attempts to reconstruct the original input vector, denoted by $\mathbf{x}'$. The model is optimized by backpropagation where the loss function is based on the difference between the reconstructed and original input vector.[11] This approach enables us to extract meaningful features from large sets of unlabeled data.

The goal of the autoencoder is to optimize both networks over a loss function,

$$\arg\min_{\theta,\phi} L(\theta, \phi)$$

where the loss function is defined as,

$$L(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left(\Delta \left(x_i, D_\theta \left(E_\phi(x_i)\right)\right)\right)^2 .$$

Here Δ is the difference operator and θ and ϕ represent the parameters of the encoder and decoder network respectively.[4]
For both the encoder and decoder, sequences of convolutions will be implemented to obtain the relevant filters. Convolutions are operations that involve applying a kernel (or filter) to an input image to obtain a feature map. This kernel can learn to extract the most relevant spatial features and patterns within the input image for it to be reconstructed. There are various well-known models that utilize convolutions for the purpose of image classification, achieving substantial results, such as VGG[12], ResNet[13] and ConvNext[14]. However, these models require a large number of parameters and significant computational resources to obtain their results. For example, ConvNext requires $101 \cdot 10^9$ Floating point operations (FLOPs) for a single image forward pass.

Model training and inference can thus be costly and time-consuming, especially given the need for large datasets and the required computational resources for the training process. Furthermore, most deployed electrocardiographs are not digitized and print the ECG recording on a piece of paper. A tool that can be run on a mobile device or workstation allows cardiologists to implement the latest models without the need of acquiring the newest hardware. However, mobile devices and workstations are often limited in their computational power and storage capacity. Both these constraints require the development and deployment of the model to be efficient, without sacrificing accuracy.

One promising approach to improve the efficiency of AI models are tensor decompositions. By breaking down large, complex tensors into simpler components, tensor decompositions can

significantly reduce the computational load, making the models more suitable for deployment on embedded devices with limited computational resources. Besides making the models more efficient, this technique can also maintain or potentially increase model accuracy by reducing overfitting and improving generalization to new data.

The primary objective of this thesis is to answer the following:

Can the Canonical Polyadic Decompositoon reduce the computational complexity of ECG classification models while maintaining diagnostic performance?

This research question will be answered by testing the following three hypotheses:

- CP-decomposed autoencoders intended for ECG classification can maintain a similar level of accuracy compared to uncompressed autoencoders.
- CP-decomposed autoencoder networks will lead to faster convergence compared to uncompressed autoencoder networks.
- Superior results can be achieved with CP-decomposed autoencoders when limiting the number of training samples compared to uncompressed autoencoder networks.

The paper of Wang et al. [15] presents a one dimensional unsupervised pre-training method using the ConvNeXt architecture for the autoencoder. This results of this thesis will be compared to theirs, where they achieve an accuracy of 94.39%. The same datasets are used to provide a fair comparison.

1-2 Thesis outline

Chapter 2 introduces the topics of electrocardiography, tensors, the CPD and convolutional autoencoders. The first section provides an explanation of the cardiovascular system, cardiac cycle and the electrocardiography process. The tensor section introduces the concept of tensors, their basic operations and the CPD, which will be the tensor decomposition applied throughout this thesis. In the last section the convolutional autoencoder will be described, what a convolution is and how the CPD can be applied to the autoencoder.

The methodology will be presented in Chapter 3. The methodology includes the data collection process, the various preprocessing steps and its distribution. Furthermore, it gives an overview of the various autoencoder models tested in the first experiment, the application of tensor decompositions and the evaluation metrics used to compare performance. This chapter details the step-by-step process followed to perform the experiments.

In Chapter 4 the results of the conducted experiments are presented. It includes a comparison of the different autoencoder models, the impact of tensor decompositions on model performance and an analysis of convergence speed and data efficiency. For each experiment the results are discussed in the context of the research hypotheses.

Finally, in Chapter 5 the findings of this thesis will be summarized along with recommendations for future work.

The glossary with a list of acronyms and a list of symbols can be found at the end of the thesis.

1-3 Implementation

All model design, data generation, training and evaluation was done in Python. The machine learning functionality was obtained from Pytorch[16]. The Tensorly[17] library was furthermore used to implement the various Tensor Decompositions throughout this thesis.

All of the experiments were run on a server provided by the TU Delft. The server consisted of 4 NVIDIA RTX A5000 each with 24GB of VRAM and a 12-core CPU with 128GB of RAM.

The full implementation of the model can be found on Github at: `https://github.com/frederikhogenbosch98/MScThesis`. These scripts are designed to be deployed on NVIDIA GPU's.

Chapter 2

Background

2-1 Electrocardiography

First discovered by the Dutch researcher Willem Einthoven, ECGs have been a fundamental tool in medical treatment since the early 19th century. Einthoven's research laid the groundwork for the current extensive use of ECG in monitoring cardiac health.[18] An ECG is a recording of the electrical activity of the heart over time as it goes through its cardiac cycles. The electrical activity is captured by placing various electrodes over the body and measuring the potential between these nodes. The standard in modern medicine practices is a 12-lead ECG, where electrodes are placed on the limbs and around the heart of the patient.
The core function of the cardiac muscle lays in its contraction, triggered by electric signals. These signals generate a complex electrical pattern that travels through the heart, captured as a potential movement over time by the ECG. The data collected from these various measurement points are critical as they can reveal various cardiac issues, such as arrhythmia. References for this chapter include Herring and Paterson[1] and Garcia[19].

2-1-1 Cardiovascular system

The cardiovascular system is one of the most complex organs in the human physiology. Its main function is to provide oxygen, nutrients and water to the other organs in the body. The heart also distributes hormones to tissues, regulates the body temperature and provides the mechanism for reproduction. It does this by circulating blood through the cardiovascular system. Figure 2-1 gives an overview of the anatomy of the heart.
The heart consists of two sides, left and right, each with an atrium and ventricle. The right side receives deoxygenated blood from the body and pumps it through the lungs where it releases carbondioxide and receives oxygen. The left side receives the oxygenated blood from the lungs and pumps it to the rest of the body. The ventricles are located at the bottom of the heart. Its function is to fill up with blood after which it contracts and pumps the blood into the arteries. The atria are located at the top of the heart. They ensure the ventricles are filled completely before contraction.
The aortic arch provides the blood to the rest of the organs, whereas the vena cava receives the deoxygenated blood. The left pulmonary artery distributes the blood to the lungs, after which they are returned to the left atrium via the pulmonary veins.

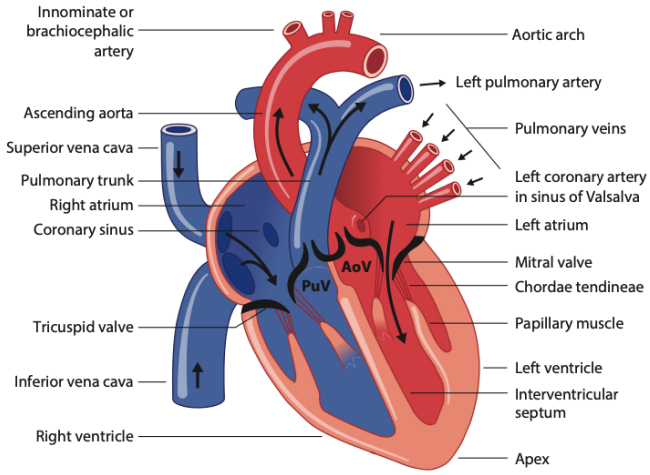


Figure 2-1: Overview of the heart indicating its most essential components. Located at the top of the heart are the left and right atria. On the bottom are the left and right ventricles. Obtained from Herring and Paterson [1].

2-1-2 Cardiac cycle

The cardiac cycle consists of two periods. When the atria and ventricles are filling up with blood it is called the diastole. The atria push blood into the ventricles for the part that it is not able to fill up passively. The other period is called systole and represents the contraction. This cycle is guided by an electrical signal travelling through the heart. It is initiated by the pacemaker, which is the 'spark plug' of the heart. This sets off an action potential that follows a complex system throughout the heart where various parts of the cardiac muscle contract.

Cardiac conduction system

To pump the blood out of the atria and ventricles, the cardiac muscles must contract. The contraction relies on the distribution of ions within the cardiac muscles, myocytes. These myocytes have a varying potential within their cell through the distribution of ions.
The electrical signal starts at the Sinoatrial (SA) node. This node is located at the top of the right atrium. Its cells can spontaneously change potential, setting of an action potential to neighbouring cells. This change in potential is called the depolarization of a cell. This signal first travels to the Atrioventricular (AV) node. This node adds a delay to the signal before it continues into the Bundle of His. From the Bundle of His the signal splits into the right and left bundle branch which move all the way around both ventricles, depolarizing this area. As the potential moves through the myocytes, they contract and thereby control the size of the four chambers.
Figure 2-2 shows the location of all relevant parts in this process. The signal thus starts at the top of the right atrium where it moves down towards the gap between the atria and ventricles. It then moves around the ventricles and makes its way up again. The cycle ends with the areas polarizing again. The cycle takes about 0.9s for an average adult, which is equivalent to 67 BPM.

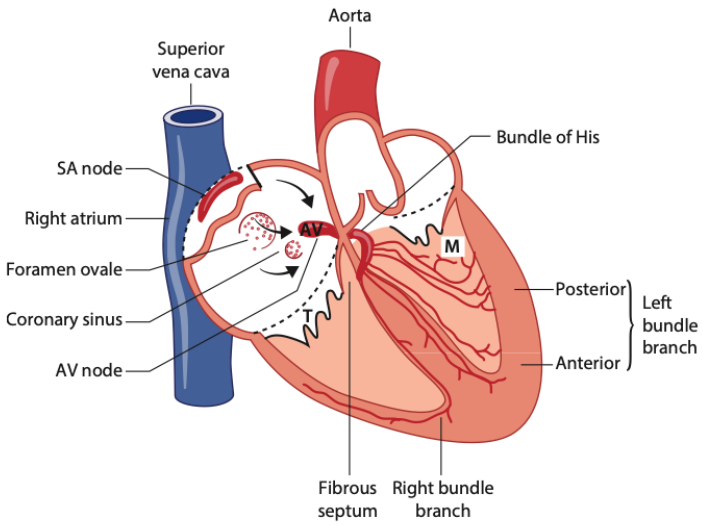


Figure 2-2: Overview of the cardiac conduction system. The signal originates at the top of the right atria in the SA node, after which it follows the AV node towards the bundles of His, dissipating via the bundle branches. Obtained from Herring and Paterson [1].

2-1-3 Electrocardiography

Standard ECG leads

Electrocardiography is the recording of the electrical activity over a period of time by the use of electrodes placed over the body. These electrodes are placed such that various angles of the heart can be inspected. Most modern ECG recordings use 10 electrodes placed over the body of which 12 angles or leads are then inspected. Figure 2-3 shows the placement of the 10 electrodes and the corresponding axis on which the activity is recorded.
One lead is the sum of the action potentials between two electrodes. This can be visualized by looking from the positive measuring electrode to the negative measuring electrode and observing the sum of the vector field of action potentials over time. The leads can be separated into two distinct categories, 6 originate from the limb electrodes and 6 originate from the chest electrodes. Leads I, II and III are combinations of axes between the positive and negative poles of the limb electrodes. Furthermore, you have various augmented leads, aVR, aVL and aVF, which point in the direction of the right arm, left arm and foot respectively. These are a combination of the the three limb electrodes to create extra perspectives of the heart.
The chest electrodes result in a close up view of the heart from various perspectives. These leads are denoted by V1, V2, V3, V4, V5 and V6. V1 is close to the right ventricle and V6 is rotated about 90 degrees towards the side of the left ventricle as is visualized in Figure 2-3. They are all pointed at the center of the heart.

Excitation sequence

The typical heart cycle contains three deflections; the P-wave, the QRS complex and the T-wave. Figure 2-4 shows one cardiac cycle with all three wave types from the lead I angle. Please refer to Figure 2-1 for an overview of the heart anatomy.
The P-wave comes from the electrical activity of the firing of the SA node that creates a mean vector towards the AV node. The AV node acts as a delay on the signal, which can be seen in the short interval after the P-wave. This interval is called the PQ-interval and is measured as the time between the beginning of the P-wave to the beginning of the Q-wave.

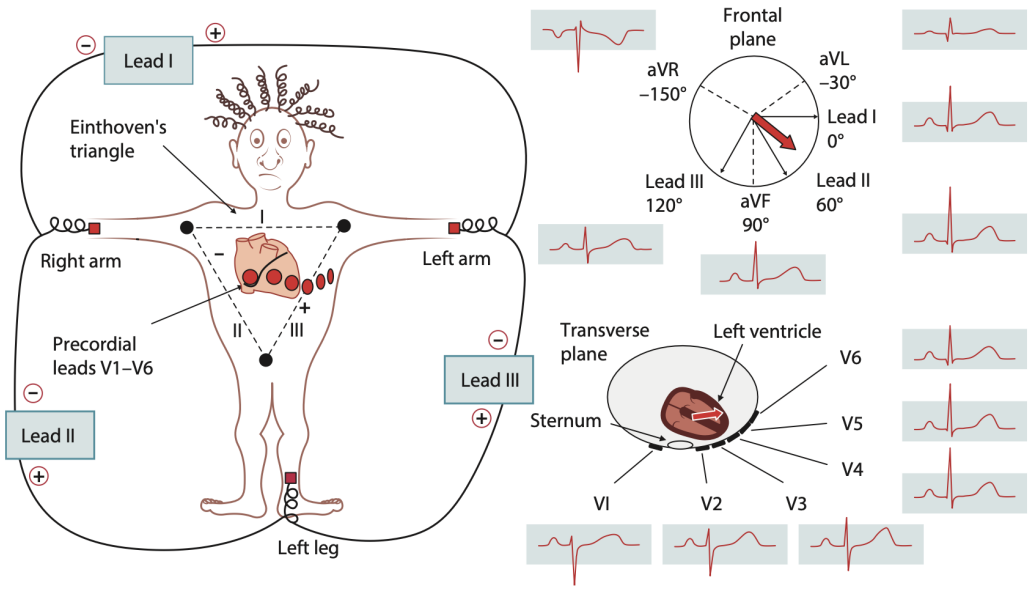


Figure 2-3: On the left the placement of the three limb and six chest electrodes is shown. On the right the resulting angles and ECG examples are given. Obtained from Herring and Paterson [1].

The interval is followed by the QRS complex, which corresponds to the ventricular depolarization. The QRS complex is intialized by the depolarization of the area between the left and right bundle branch. Here the action potential moves from the left bundle branch towards the right bundle branch as the electrical activity moves down the bundle branches. The mean vector points from the left bundle branch towards the right ventricle. The R-wave is the result of a more complex composed mean vector as the posterior and anterior depolarize outward. The vector points towards the positive electrode as the left ventricle is larger than the right ventricle and thus contributes more charge to the mean. Finally, the S-wave is the electrical signal moving to the end of the bundles branches to the top of the heart. After the QRS complex there is a final T-wave, which is the repolarization of the ventricles, where the left ventricle again contributes the most to the mean vector. The corresponding interval commonly used is the QT-interval.

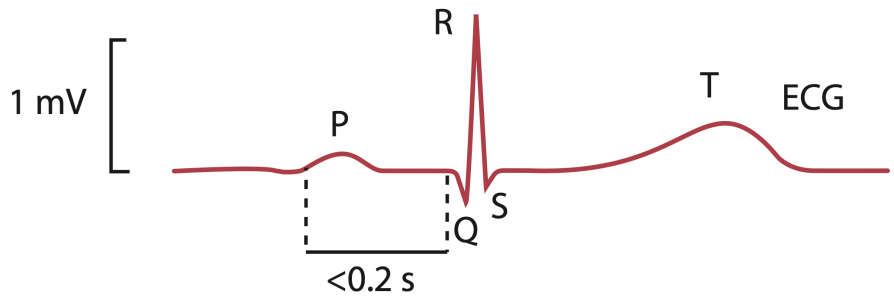


Figure 2-4: Electrical activity of one cardiac cycle showing the P- and T-wave and the QRS complex. Obtained from Herring and Paterson [1].

2-1-4 Arrhythmia

Arrhythmia are irregularities in the heart's rhythm that can arise from various physiological abnormalities. Although there are many classes of arrhythmia, this thesis will focus only on the

Supraventricular Ectopic Beat (SVEB), the Ventricular Ectopic Beat (VEB), and the Fusion of Ventricular and Normal Beat (F). The arrhythmia descriptions in this section were obtained from ECGpedia [20].

Supraventricular Ectopic Beat (SVEB)

Ectopic beats are the result of spontaneous depolarization from cells that are not in the sinus node. The sinus node generally dominates because it has the fastest pacing rate. However, any healthy cardiomyocyte can act as an ectopic pacemaker. These ectopic depolarizations can arise from the atria (60-80 bpm), AV node (40-60 bpm) or ventricles (20-40 bpm). In the case of Supraventricular Ectopic Beats, the depolarization happens in the atria or AV node.
Ectopic beats are premature and disrupt the regular rhythm of the heart. They can be characterized in the ECG by:

- **P-wave**: Often abnormal in shape or direction when compared to previous P-waves.
- **QRS complex**: Generally follows the normal conduction pathway. However, if the premature beat is very early, the AV node is not able to conduct. This results in the absence of a QRS complex.
- **Timing**: Occurs earlier than expected if following the normal rhythm, causing a premature beat.

Ventricular Ectopic Beat (VEB)

A ventricular ectopic beat, also known as a ventricular premature beat (VPB), originates from the ventricles. VEBs are conducted through the specialized conduction system of the heart, resulting in broad QRS complexes. The VEB can be recognized from the width of the QRS complex which is at least > 0.12s, but often in the range of 0.16s to 0.2s. VEBs can be caused by a variety of factors, including ischemia, hypoxia, old scar tissue or may occur idiopathically.

Cardiac conditions such as structural heart disease, ischemia, congenital arrhythmia, and pulmonary disease are often linked with VEBs. However, they may also occur in individuals without known heart disease.

Fusion of Ventricular and Normal Beat (F)

A fusion beat occurs when a normal beat from the atria coincides with a ventricular ectopic beat, resulting in a hybrid complex. This beat contains features of both normal and ectopic beats. They can be characterized in the ECG by:

- **QRS complex**: Intermediate in width and shape between normal and ectopic QRS complexes, as it represents a combination of both pathways of depolarization.
- **Timing**: Occurs at a point where the atrial and ventricular beats overlap, leading to a fusion of electrical activity.

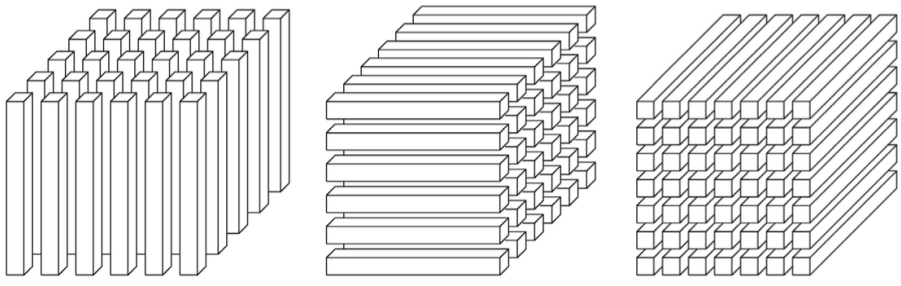
2-2 Tensors and the Canonical Polyadic Decompostion

The objective of this section is to first introduce the concept of tensors, giving their definition, basic properties and their usefulness in representing multidimensional data. In the following part their various operations will be introduced, such as the outer product and rank. Finally, the Canonical Polyadic Decomposition (CPD) will be explored to determine how this method obtains more efficient approximations of tensors.
For a more in-depth introduction on tensors it is recommended to read Cichocki et al. [3].

Tensors provide a framework for representing and manipulating multi-dimensional data. Familiar forms of tensors are the zero-dimensional scalar, the one-dimensional vector and the two-dimensional matrix. In this thesis the number of dimensions of a tensor is denoted by N. A simple example of a three-dimensional tensor is the description of the spatial dimensions, x, y and z. This can be extended to four dimensions when the time dimension is considered as well. Each point in space at a specific timestep is represented by an element in the four-dimensional tensor.

2-2-1 Preliminaries

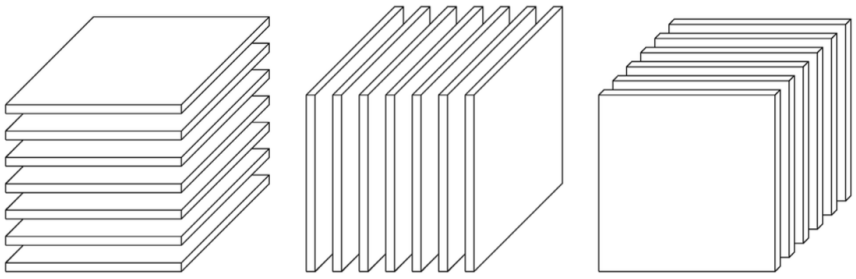
To distinguish between the common data types, a tensor is denoted by a bold capital letter with an underline, $\underline{\boldsymbol{X}}$, a matrix as a capital bold letter, $\boldsymbol{X}$, a vector is denoted as a lowercase bold symbol $\boldsymbol{x}$ and a scalar as a lower- or uppercase letter, a or A. A tensor with N dimensions is defined as $\underline{\boldsymbol{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Here I_i is the size of the i-th dimension. An element of a tensor $\underline{\boldsymbol{X}}$ at index $i_1, i_2, ..., i_N$ is expressed as $x_{i_1,i_2,...,i_N}$. For a vector element this is denoted as $\boldsymbol{x}_i$, for a matrix element as $\boldsymbol{X}_{i,j}$. Indices typically range from 1 to I_n as, $i_n = 1, 2, ..., I_n$.
It can be useful to distinguish between the *fibers* of a tensor, which are analogous to columns and rows of a matrix. A visual representation of the fibers of a third-order tensor can be found in Figure 2-5. For such a third-order tensor the columns, rows and tubes are denoted as $\boldsymbol{x}_{:,j,k}$, $\boldsymbol{x}_{i,:,k}$ and $\boldsymbol{x}_{i,j,:}$ respectively. [2]



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$ (b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$ (c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$

Figure 2-5: Graphical representation of the fibers of a tensor over mode-1 in (a), mode-2 in (b) and mode-3 in (c). Obtained from Kolda and Bader [2].

Another useful method of describing tensors is by its *slices*. Compared to fibers, here you fix all but two indices, as opposed to one. Figure 2-6 shows the graphical representations of sllices within a third-order tensor. The slices are denoted as, $\mathbf{X}_{i,:,:}$, $\mathbf{X}_{:,j,:}$, $\mathbf{X}_{:,:,k}$.



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

Figure 2-6: Graphical representation of the horizontal (a), lateral (b) and frontal(c) slices of a tensor. Obtained from Kolda and Bader[2].

2-2-2 Tensor Properties

Outer product

The outer product is a fundamental operation in tensor decomposition methods as it is often used to express complex tensor structures into simpler vector representations. The outer product of two vectors, $\mathbf{u}$ and $\mathbf{v}$ is defined as,

$$\mathbf{u} \circ \mathbf{v} = \begin{bmatrix} u_1v_1 & u_1v_2 & \dots & u_1v_n \\ u_2v_1 & u_2v_2 & \dots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \dots & u_mv_n \end{bmatrix}. \tag{2-1}$$

The result is a matrix where each element (i, j) is the product of u_i and v_j.
The outer product can be expanded to higher dimensions. Consider tensors $\underline{\boldsymbol{X}} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ and $\underline{\boldsymbol{Y}} \in \mathbb{R}^{J_1 \times \dots \times J_M}$. The outer product is then defined as,

$$\underline{\mathbf{Z}} = \underline{\mathbf{X}} \circ \underline{\mathbf{Y}}. \tag{2-2}$$

Each element of the resulting tensor $\underline{\boldsymbol{Z}}$, denoted by $z_{i_1,\dots,i_N,j_1,\dots,j_M}$, is derived by multiplying the corresponding elements from $\underline{\boldsymbol{X}}$ and $\underline{\boldsymbol{Y}}$. This forms a new tensor whose dimensionality is the sum of the dimensions of the original tensors,

$$z_{i_1,\dots,i_N,j_1,\dots,j_M} = \sum_{r=1}^{1} x_{i_1,\dots,i_N,r} \cdot y_{r,j_1,\dots,j_M}. \tag{2-3}$$

Tensor Multiplication

For this research only the multiplication of a tensor with a matrix will be considered. This is called the mode-n product of a tensor with a matrix. For a tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ and a matrix $\boldsymbol{A} \in \mathbb{R}^{J \times I_n}$, the mode-$n$ product is defined as[21],

$$(\underline{\boldsymbol{X}} \times_n \mathbf{A})_{i_1,\dots i_{n-1},j,i_{n+1}\dots i_N} = \sum_{i_n=1}^{I_n} x_{i_1,i_2,\dots i_N} a_{j,i_n}. \tag{2-4}$$

Rank-one Tensors

An important property of a tensor is its rank. For matrices this is simply defined as the maximum number of linearly independent columns (or row) vectors. For tensors the definition is more complex as it represents the minimum number of rank-one tensors needed to generate the original tensor as their sum. A tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ of dimension N, is considered to be rank-one if it can be expressed as the outer product of N vectors,

$$\underline{\boldsymbol{X}} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(N)} \tag{2-5}$$

where $\mathbf{a}^{(i)}$ represents a first-order tensor.
Figure 2-7 shows a third-order rank-one tensor $\underline{\boldsymbol{X}} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$.

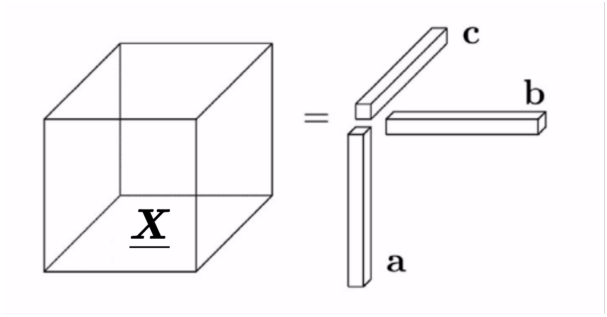


Figure 2-7: Rank one tensor visualization. Tensor $\underline{\boldsymbol{X}}$ is decomposed into three vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. Modified image with original from Kolda and Bader[2].

Rank-one tensors are an important concept in tensor decompositions as they represent the most fundamental form of a tensor from which more complex tensors can be constructed. If a complex data tensor can be reduced to a sum of rank-one tensors, its most important structures are captured within the rank-one tensors.

2-2-3 Canonical Polyadic Decomposition

Just as matrices can be broken down into products of smaller matrices through methods like Singular Value Decomposition, tensors can similarly be expressed through decomposition into smaller tensors. Decomposition methods allow data to be mapped to smaller representations while maintaining the most important information. This can be useful when working with very large tensors, which are becoming more prevalent in fields such as machine learning.[22]

The concept of tensor decompositions was introduced by Hitchcock in 1927, where he showed a tensor can be expressed in a polyadic form.[23] This polyadic form is a sum of the outer product of a finite number of rank-one tensors. The concept of tensor decompositions was not developed much further until the work of Tucker[24] in the 1960s. Since then it has gained much attention in a wide range of fields such as quantum mechanics, data science and machine learning.[2]

Tensor decompositions are generally applied in either of two ways. It can reduce tensor storage by decomposing the structure to keep only the most vital information. However, this thesis will focus on the other application, which is to decompose a structure with undefined elements to use in an optimization problem. In for example machine learning, a data structure containing random weights is optimized according to some loss function. Tensor decompostions could reduce the training and inference computational complexity when applied to the weights tensor.

The Canonical Polyadic Decomposition (CPD) stands as one of the most straightforward and commonly used methods for decomposing tensors. This technique is also known under the names PARAFAC[25] and CANDECOMP[26]. The fundamental concept behind CPD is to express the original tensor as a sum of R rank-one tensors. In this context, R is the only parameter of the CPD and is referred to as the CP-rank.

To decompose a tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ using CPD, it can be represented as in equation 2-6.

$$\underline{\mathbf{X}} \cong \sum_{r=1}^{R} \lambda_r \mathbf{b}_r^{(1)} \circ \mathbf{b}_r^{(2)} \circ \cdots \circ \mathbf{b}_r^{(N)} = \underline{\mathbf{\Lambda}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \ldots \times_N \mathbf{B}^{(N)} \tag{2-6}$$

The first part consists of a sum of rank-one tensors, each denoted as $\mathbf{b}_r^{(i)}$, which are multiplied by a corresponding scalar λ_r. These scalars, λ_r, are the non-zero diagonal entries of the core tensor $\mathbf{\Lambda}$, which has dimensions $\mathbb{R}^{R \times R \times \cdots \times R}$. This core tensor essentially scales each rank-one component, influencing the contribution of each component to the overall tensor structure.
The latter part uses factor matrices to express the CPD. Here, $\mathbf{B}^{(i)} = \left[\mathbf{b}_1^{(i)}, \mathbf{b}_2^{(i)}, \ldots, \mathbf{b}_R^{(i)}\right]$ are termed as factor matrices, which organize the vectors of each mode into columns of a matrix. The entire decomposition can also be neatly expressed using the notation $[\![\underline{\mathbf{\Lambda}}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \ldots, \mathbf{B}^{(N)}]\!]$.

Figure 2-8 visually illustrates the block diagram of the CPD for a third-order tensor. In this diagram, the tensor $\underline{\mathbf{X}}$ is approximated by the sum of the outer products of vectors $\boldsymbol{a}_r$, $\boldsymbol{b}_r$, and $\boldsymbol{c}_r$, each scaled by λ_r.

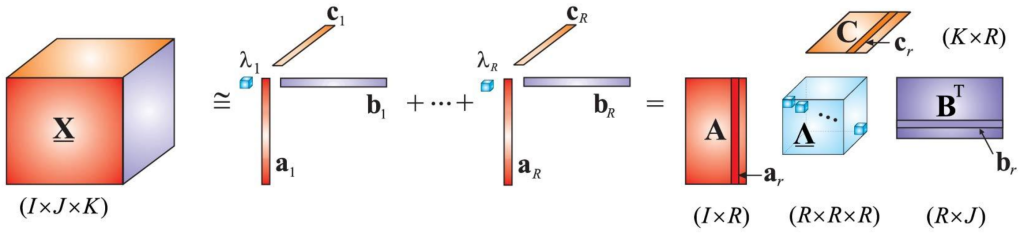


Figure 2-8: Block diagram of a CPD of a third order tensor. The original tensor $\underline{\boldsymbol{X}}$ on the left is decomposed in a sum of rank R outer product of vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. On the right the factor matrix representation is given. Obtained from Cichocki et al.[3].

Furthermore, the uniqueness of the CPD under certain conditions is an important characteristic that differentiates it from other tensor decompositions, like the Tucker decomposition. Its uniqueness implies that the tensor can be decomposed in a way that the factor matrices involved are determined uniquely, except for some trivial variations such as the scaling and permutation of their columns.[27]

There are various algorithms to compute the CPD, where the Alternating Least Squares (ALS)[3] method is among the most frequently used. The method works by iteratively fixing all but one factor matrix at a time, and then solving the least squares problem to update the unfixed factor matrix. This process repeats, alternating between fixing the factor matrix until a stopping criterion is met or when it reaches the maximum number of iterations.

Limitations of the CPD can be determining the optimal number of components (CP-rank) which can be non-trivial, but is essential to achieve an optimal decomposition. Furthermore, algorithms to obtain the CPD, such as ALS, can converge to local minima, resulting in a non-optimzed decomposition.

2-3 Convolutional Autoencoders

This section introduces the fundamental aspects of autoencoders, their integration with convolutional networks and the application of the CPD in convolutional networks.

2-3-1 Autoencoders

An autoencoder is a type of artificial neural network used for unsupervised learning of efficient encodings. The objective of an autoencoder is to learn a compressed representation of input data, which can then be reconstructed back to its original input. The architecture of an autoencoder consists of two main components, an encoder network and a decoder network.

The encoder network is defined as $E : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The goal of the encoder is to compress the input data of dimension n into a lower-dimensional latent space of dimension m. This process involves mapping the input data $\mathbf{x}$ to a latent representation $\mathbf{z}$. The purpose of this compression is to capture the most relevant features of the input data while discarding noise.

The decoder network then decompresses the latent space $\mathbf{z}$ to a reconstruction $\mathbf{x}'$. This can be denoted by $D : \mathbb{R}^m \rightarrow \mathbb{R}^n$. The objective here is to minimize the error between the input image $\mathbf{x}$ and the reconstruction $\mathbf{x}'$. The network is optimized by backpropagation using a loss function that measures the difference between the input and the reconstruction. This is formulated as,

$$\arg\min_{\theta,\phi} L(\theta, \phi) \tag{2-7}$$

where the loss function used is the Mean Squared Error (MSE) given by,

$$L(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left(\Delta \left(x_i, D_\theta \left(E_\phi(x_i)\right)\right)\right)^2. \tag{2-8}$$

The parameters of the encoder and decoder are denoted by ϕ and θ respectively.[4]
Figure 2-9 shows an example of the reconstruction of an MNIST[28] number using an autoencoder network.

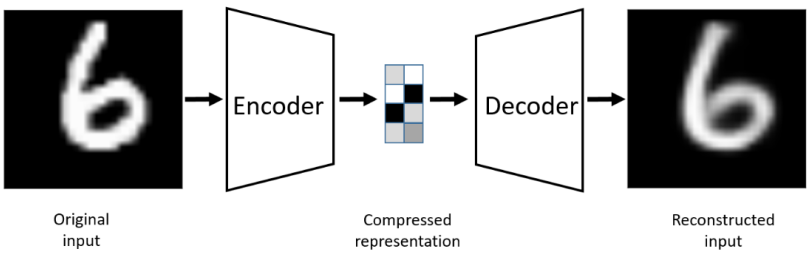


Figure 2-9: Example of MNIST autoencoder architecture. The original input image is compressed by the encoder to a lower dimensional latent space representation, after which it is reconstructed back to its original image by the decoder. Obtained from Bank et al.[4].

Autoencoders have a wide array of applications in the fields of image processsing. One application is data denoising, where autoencoders are trained to remove noise from images or signals. By introducing noise to the input data and training the autoencoder to reconstruct the original noise-free data, the model learns to filter out noise.[29]

Variational Autoencoders (VAEs) are another variation of the autoencoder, where the network learns to generate new data samples similar to the training data. VAEs are mainly used in generative tasks such as image synthesis and data augmentation.[30]

In the context of transfer learning, which will be the application of autoencoders in this thesis, autoencoders are used to pre-train networks. The encoder part of a pre-trained autoencoder can be used to initialize other models, such as classifiers, making use of the learned features as an input. This approach is particularly useful in scenarios with limited labeled data, where pre-training of an unsupervised task can provide an increase in performance for supervised learning tasks.

2-3-2 Convolutional neural networks

Introduced in 1989 by LeCun et al.[31], convolutional neural networks are a type of neural network characterized by their use of kernels that convolve over the input data. Due to their shared weights, the number of parameters is reduced in comparison to for example a fully connected layer. This makes CNNs more efficient to train, while minimizing the impact on its performance.

CNNs are composed of a series of convolutional layers, which are the fundamental building blocks of the network. Each convolutional layer typically consists of three main components: a convolution operation, a normalization operation and an activation function. The normalization layer, often a batch normalization layer, standardizes the output of the convolution operation. This layer helps to stabilize the weights by reducing internal covariate shift. The normalization ensures that the distribution of inputs to each layer remains consistent.[32] The activation layer introduces non-linearity into the model, which is necessary for the network to learn more complex patterns.[33]

Discrete Convolution

The convolution operations takes two discrete fucntions as input and outputs another function. The operation is defined as[11],

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]. \tag{2-9}$$

This can be interpreted as the function g moving itself over function f step by step. At each step n, all values of f are multiplied with the corresponding values in g. The sum of these multiplications is the value of the output function at index n.

The focus for this thesis will be on discrete convolutions as the input data is going to be discrete.

1D case

For the one dimensional case the filter is defined as a vector of weights,

$$\mathbf{w} = [w_0, w_1, w_2, \ldots], \tag{2-10}$$

and the input is a time series vector x,

$$\mathbf{x} = [x_0, x_1, x_2, \ldots] \tag{2-11}$$

then the convolution for the 1-dimensional case is calculated as,

$$\mathbf{y}_j = \sum_{k=-\infty}^{\infty} \mathbf{x}_{j+k} \mathbf{w}_k. \tag{2-12}$$

Here, $\mathbf{y}_j$ represents the output of the convolution at position j. This operation slides the filter $\mathbf{w}$ over the input $\mathbf{x}$, calculating the dot product at each step and producing a new sequence that extracts features of the input data based on the filter used.

2D case

The 1D case can be extended to two dimensions by considering an input signal of $\mathbf{X} = [x_{i,j}]$, which can represent an image or any other grid-like data structure. In this case, the convolution operation involves a 2D filter or kernel, which is also a grid of weights.

$$\mathbf{Y}_{i,j} = \sum_{k=-\infty}^{\infty} \sum_{r=-\infty}^{\infty} \mathbf{X}_{i+k,j+r} \mathbf{W}_{k,r} \tag{2-13}$$

This can be interpreted as a grid structure of weights moving over the image. This grid structure of weights is often referred to as the kernel or filter. Each position of the kernel on the image produces a single output value, which is the sum of element-wise multiplications between the kernel and the corresponding input values. This process produces an output feature map that highlights certain patterns or features present in the input image.

An example of the convolution of a 2×2 kernel and a 3×3 input matrix is given in Figure 2-10. In the example the filter on the left of the first arrow is element-wise multiplied with the top left four elements in the input matrix. The result is shown to the right of the outer most arrow.

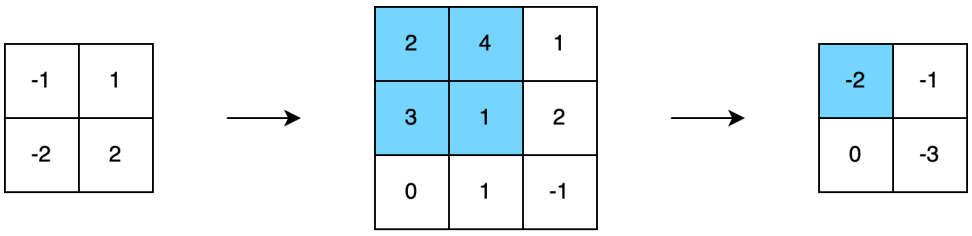


Figure 2-10: Example of a 2D convolution operation. The feature map on the left is multiplied by the upper left quarter of the input, resulting in the top left value in the right most matrix.

Oftentimes channels are considered within each convolution. Channels are used to include more feature maps per layer, which allows the network to extract more features per layer. For example, RGB images consist of three input channels: red, green, and blue. These are often extended to 64 channels after the first convolution to capture more features per input channel. To include channels, the convolution operation equation of a single feature map multiplication becomes (in tensor format),

$$\underline{\boldsymbol{Y}}_{t,w',h'} = \sum_{s=1}^{S} \sum_{i=1}^{D} \sum_{j=1}^{D} \underline{\boldsymbol{K}}_{t,s,j,i} \underline{\boldsymbol{X}}_{s,w_j,h_i}. \tag{2-14}$$

In this equation, $\underline{\boldsymbol{Y}}_{t,w',h'}$ represents the output value at position (w', h') and output channel t, reflecting the result of the convolution operation for a specific output feature map. The term S denotes the total number of input channels. The input value $\underline{\boldsymbol{X}}_{s,w_j,h_i}$ corresponds to the element at position (w_j, h_i) and input channel s. The weight $\underline{\boldsymbol{K}}_{t,s,j,i}$ refers to the kernel weight at position (j, i), input channel s, and output channel t, representing the filter weight applied to the input element for a particular input and output channel.

In addition to the basic convolution operation, two important concepts are stride and padding. Stride refers to the number of pixels the kernel moves over the input image after each application, which can affect the size of the output feature map. Padding adds extra pixels around the border of the input image, such that the kernel can cover edge areas more effectively. Padding is also used to control the spatial dimensions of the output.

This is implemented as,

$$w_j = (w' - 1)\,\Delta + j - p \text{ and } h_i = (h' - 1)\,\Delta + i - p,$$

where Δ is the stride and p is the padding size.

2-3-3 Convolutional Layer

The convolutional layer usually consists of a convolution operation, normalization step and activation function. This section introduces the concepts of batch normalization and the GELU, which are the chosen normalization method and activation function for this thesis.

Batch normalization

Normalizing the output of the convolution layer is necessary to stabilize the training process. This happens by scaling the weights based on the statistics a training batch. This helps to maintain the output in a stable range, which reduces the risk of vanishing or exploding gradients. A common method of normalization in CNNs, which will be used in this thesis, is batch normalization. The batch normalization process is defined by,

$$\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} \tag{2-15}$$

where $\mathrm{E}[x]$ is the mean, $\mathrm{Var}[x]$ is the variance of the input batch, and ϵ is a constant to prevent division by zero.[34]

GELU (Gaussian Error Linear Unit)

Following normalization, an activation function is applied to introduce non-linearity into the model. The GELU activation function is defined as,[35]

$$\mathrm{GELU}(x) = x \cdot P(X \leq x) = x \cdot \frac{1}{2}[1 + \mathrm{erf}(\frac{x}{\sqrt{2}})] \tag{2-16}$$

where erf is the Gauss error function. GELU combines properties of both ReLU and sigmoid functions, which results in a smooth non-linear curve. The GELU is plotted in Figure 2-11.

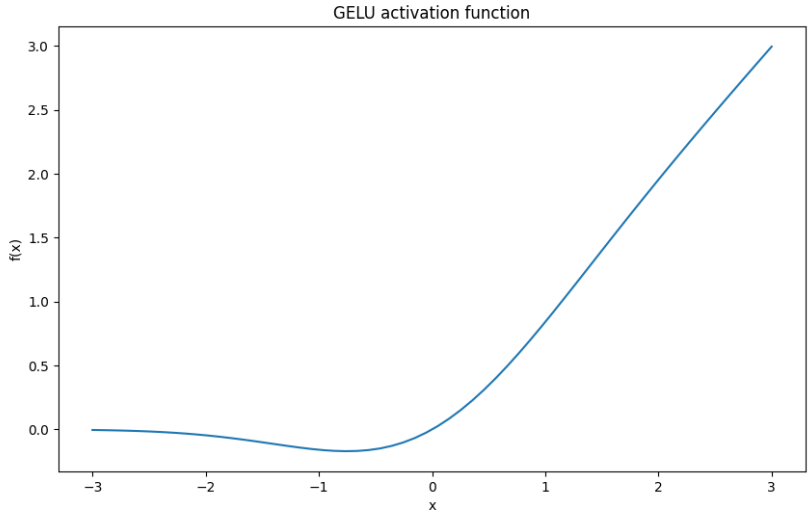


Figure 2-11: GELU activation function. For values below 0 the GELU function approaches 0 with a slight drop before the value of 0. For values higher than 0 the function returns the input. This introduces a nonlinearity in the model.

2-3-4 Convolutional Autoencoder Network

A typical CNN consists of multiple convolutional layers, with max pooling and upsampling operations in between. The spatial dimensions of the image are often reduced, while the number of channels increases. The diagram of an example of a simple convolutional autoencoder is given in Figure 2-12. This diagram indicates the feature extraction in the encoder $E : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and the image reconstruction in the decoder $D : \mathbb{R}^m \rightarrow \mathbb{R}^n$ with the intermediate latent space $\mathbf{z}$. The numbers on top of the convolution objects are the number of channels produced by the convolution. The number of input channels can be deducted from the previous objects.

This simple example compresses an input image of 128×128 in the spatial dimension, to a 32×32 representation, after which it gets reconstructed. The encoder contains max pooling layers, which extracts the maximum value from a subset of the input feature map. This reduces the dimensionality while retaining the most important features. The decoder contains upsampling layers which use bilinear interpolation to increase the spatial dimension of the intermediate image.

Max pooling

The pooling step is implemented to reduce the complexity of the training process. The most used pooling operation is max pooling[36], where only the maximum value of a $\alpha \times \alpha$ matrix within the feature map is extracted. An example of this operation is given in Figure 2-13.

Bilinear upsampling

Bilinear upsampling is a technique used to increase the spatial resolution of an image by applying linear interpolation on all surrounding pixels to estimate new pixel values. This provides a smoother result than for example nearest-neighbours upsampling, where the value of the nearest pixel is selected for the new pixel.

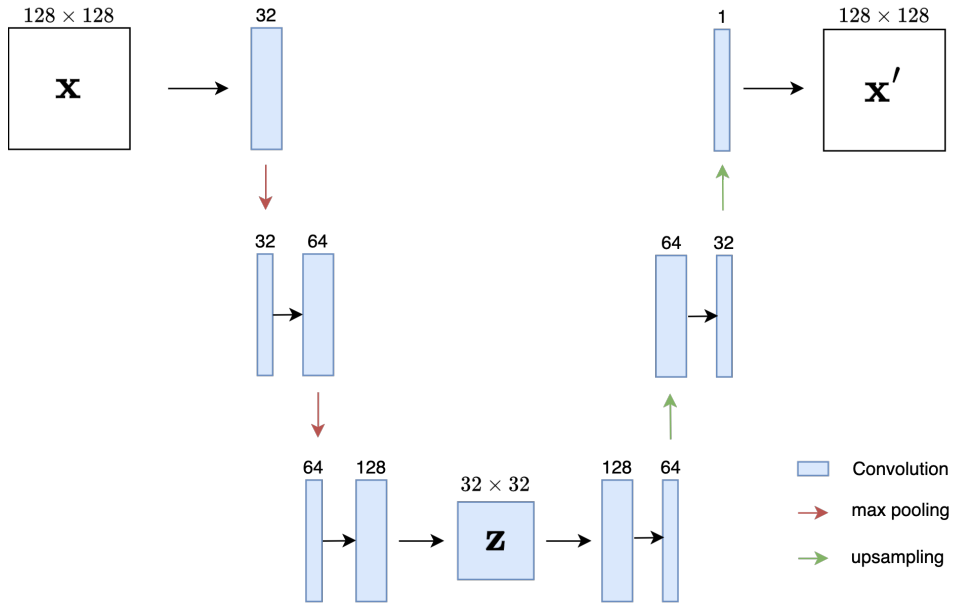


Figure 2-12: Example diagram of a simple implementation of a CNN. The input image of size 128×128 is compressed by three convolutional layers to a latent space representation $\mathbf{z}$. The decoder reconstructs the lower dimensional representation through a sequence of upsampling and convolutional layers back to its original size.

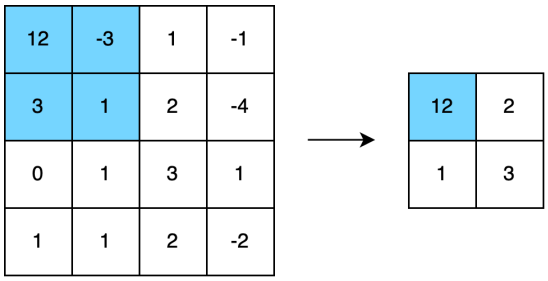


Figure 2-13: Example of a 2×2 max pooling operation. The maximum value of the upper left quarter is selected for the downsampled feature map.

2-3-5 The Canonical Polyadic Decomposition in CNNs

When tensor decomposition are applied to a convolutional layer, they decompose the kernel weights such that the convolution executes in four subsequent steps. Figure 2-14 gives a visualization of the convolution operation, whereas Figure 2-15 gives the decomposed operation. The convolution happens over one of the four dimension in each step.

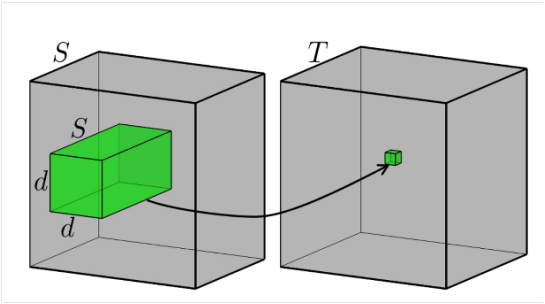


Figure 2-14: Visualization of a convolution operation with a kernel size of $S \times T \times d \times d$.

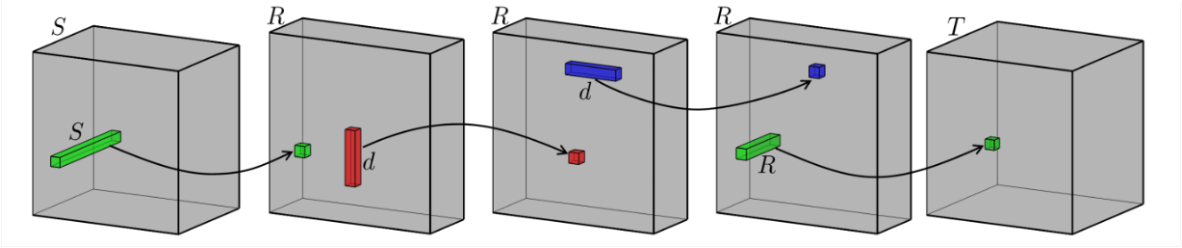


Figure 2-15: Tensor decomposed convolutional operation into four subsequent steps. The input channels dimension S is firstly mapped to R, which is later mapped to the number of output channels T. The spatial dimensions are mapped separately instead of combined. In the illustration d is used to indicate the kernel size, instead of D used in the thesis.

The convolution operation equation of a single feature map multiplication was defined in equation 2-14. It is repeated here for readability of the derivation,

$$\underline{\boldsymbol{Y}}_{t,w',h'} = \sum_{s=1}^{S}\sum_{i=1}^{D}\sum_{j=1}^{D} \underline{\boldsymbol{K}}_{t,s,j,i}\underline{\boldsymbol{X}}_{s,w_j,h_i}.$$

In CP-decomposition the kernel tensor is approximated by a sum of R rank-one tensors. This can be represented as[37],

$$\underline{\boldsymbol{K}}_{t,s,j,i} = \sum_{r=1}^{R} K^t_{t,r}K^h_{i,r}K^w_{j,r}K^s_{s,r}. \tag{2-17}$$

where K^a are factor matrices with $a \in [t,h,w,s]$. This is an element wise multiplication as the kernel element $\underline{\boldsymbol{K}}_{t,s,j,i}$ is approximated by a sum of R element-wise products. Substituting equation (2-17) into equation (2-14) results in the following convolution operation,

$$\underline{\boldsymbol{Y}}_{t,w',h'} = \sum_{r=1}^{R} K^t_{t,r}\left(\sum_{i=1}^{D} K^h_{r,h_i}\left(\sum_{j=1}^{D} K^w_{r,w_j}\left(\sum_{s=1}^{S} K^s_{r,s}\underline{\boldsymbol{X}}_{s,w_j,h_i}\right)\right)\right), \tag{2-18}$$

from which four separate kernel steps can be established,

$$\underline{\boldsymbol{X}}^{(1)}_{r,w_j,h_i} = \sum_{s=1}^{S} K^s_{r,s}\underline{\boldsymbol{X}}_{s,w_j,h_i}, \tag{2-19}$$

$$\underline{\boldsymbol{X}}^{(2)}_{r,w',h_i} = \sum_{j=1}^{D} K^w_{r,w_j}\underline{\boldsymbol{X}}^{(1)}_{r,w_j,h_i}, \tag{2-20}$$

$$\underline{\boldsymbol{X}}^{(3)}_{r,w',h'} = \sum_{i=1}^{D} K^h_{r,h_i}\underline{\boldsymbol{X}}^{(2)}_{r,w',h_i}, \tag{2-21}$$

and

$$\underline{\boldsymbol{Y}}_{t,w',h'} = \sum_{r=1}^{R} K^t_{t,r}\underline{\boldsymbol{X}}^{(3)}_{r,w',h'}. \tag{2-22}$$

Here $\underline{\boldsymbol{X}}^{(1)}_{r,w_j,h_i}$, $\underline{\boldsymbol{X}}^{(2)}_{r,w',h_i}$ and $\underline{\boldsymbol{X}}^{(3)}_{r,w',h'}$ are the intermediate filter tensors.

This decomposition alters the storage complexity from TSD^2 number of parameters to $R\left(2D+T+S\right)$, where R is the CP-rank.[38] The number of FLOPs for the convolution operation changes from $2TSD^2W'H'$ for the uncompressed convolution to $2R\left(SWH + D^2W'H' + TW'H'\right)$ for the CP-decomposed convolution.

2-3-6 Optimization methods

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimization algorithm which calculates the gradient based on a small batch instead of the entire dataset. This makes the algorithm faster and more scalable for large datasets. The algorithm computes the gradient of the loss function with respect to the parameters, giving the direction in which the parameters should be adjusted to reduce the error. The learning rate, denoted by α, controls the size of the steps taken in this direction.

The algorithm takes the learning rate α, the loss function $L(\theta, \phi)$, and the initial parameter vectors θ and ϕ as inputs. At each iteration, a randomly selected mini-batch is used to compute the gradient. The parameters are then updated by moving them in the direction of the negative gradient, scaled by the learning rate. This process is repeated until the algorithm converges to a minimum of the loss function or a predefined number of iterations is reached.

Algorithm 2-3.1 gives each of the steps for the parameter optimization process using SGD. The parameters of the encoder and decoder are both considered in θ.

Algorithm 2-3.1 SGD: Stochastic Gradient Descent

Require: α, $L(\theta)$, θ_0
1: $t \leftarrow 0$ (Initialize timestep)
2: **while** θ_t not converged **do**
3: $t \leftarrow t + 1$
4: $g_t \leftarrow \nabla_\theta L_t(\theta_{t-1})$ (Compute gradients w.r.t. loss function at timestep t)
5: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t$ (Parameter update)
6: **end while**
7: **return** θ_t (Resulting parameters)

Adam

The Adam optimizer[39] is a widely used optimization algorithm in the field of machine learning. The name Adam is derived from adaptive moment estimation, referring to the variable learning rate it uses. The Adam method combines two variations of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The algorithm computes adaptive learning rates for each parameter by maintaining first and second moment estimates of the gradients. The first second moment representing the mean and variance of the gradients, respectively. Using the first and second moments, Adam calculates the learning rate for each individual weight.

The algorithm build from the SGD, using the hyperparameters, learning rate α, loss function $L(\theta, \phi)$ and initial parameter vectors θ and ϕ as inputs, while adding two new hyperparameters $\beta_1, \beta_2 \in [0, 1)$. The variables β_1 and β_2 introduced by the Adam optimizer, determine the influence of gradients from previous iterations on the current update step. Typically they are set to values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Additionally, there is a constant of ϵ to safeguard against division by zero.

Algorithm 2-3.2 gives each of the steps for the parameter optimization process using Adam. The parameters of the encoder and decoder are both considered in θ.

Algorithm 2-3.2 Adam: A Method for Stochastic Optimization

Require: α, $\beta_1, \beta_2 \in [0, 1)$, $L(\theta)$, θ_0
1: $m_0 \leftarrow 0$ (Initialize first moment vector)
2: $v_0 \leftarrow 0$ (Initialize second moment vector)
3: $t \leftarrow 0$ (Initialize timestep)
4: **while** θ_t not converged **do**
5: $t \leftarrow t + 1$
6: $g_t \leftarrow \nabla_\theta L_t(\theta_{t-1})$ (Calculate gradients w.r.t. loss function at timestep t)
7: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (First moment update)
8: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Second moment update)
9: $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)
10: $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)
11: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ (Update parameters)
12: **end while**
13: **return** θ_t

Chapter 3

Methodology

This chapter introduces the data used in this thesis, defines two experiments and gives the experimental setup. The data section gives an overview of the various datasets used, what the preprocessing steps are and how the data is distributed among the various datasets. Two experiments are defined: the first focuses on selecting the most suited model for the classification of ECG images. The second experiment implements the CPD in the selected model to see how well the network performs at various compression ratios. Finally, a general experimental setup is given.

The first experiment focuses on selecting the most optimal uncompressed model for the purpose of ECG classiciation. The second experiment will implement the CPD and seek to test the hypothesis given in Table 3-1.

Hypothesis 1	CP-decomposed autoencoders intended for ECG classification can maintain a similar level of accuracy compared to uncompressed autoencoders.
Hypothesis 2	CP-decomposed autoencoder networks will lead to faster convergence compared to uncompressed autoencoder networks.
Hypothesis 3	Superior results can be achieved with CP-decomposed autoencoders when limiting the number of training samples compared to uncompressed autoencoder networks.

Table 3-1: Hypotheses for the second experiment.

3-1 Data

This section introduces the datasets used in this thesis, what preprocessing steps were taken and gives an insight into the distribution of the data. In order to be able to compare the results from this thesis with the results of Wang et al.[15], the same data sources were used for the training, validation and testing steps.

3-1-1 Datasets

The datasets used in this thesis were obtained from the PhysioNet platform [40], specifically from the Computing in Cardiology Challenge 2020. [41] An overview of these datasets is provided in Table 3-2. Due to its relatively small number of samples, the PTB dataset was excluded.

Given the two-step structure of the model, the datasets were categorized into unlabeled and labeled sets accordingly. The unlabeled datasets were used to train the autoencoder, whereas the labeled sets were used to train the classifier. Specifically, the PTB-XL, CPSC, and G12EC datasets were used for the unsupervised training phase, whereas the INCART and MIT-BIH datasets were used for the supervised training phase.

Table 3-2: Number of beats of selected datasets.

Dataset	Number of beats
PTB-XL	130,794
CPSC	41,262
G12EC	60,021
INCART	175,580
MIT-BIH	100,604

Here is a brief overview of the datasets used in this thesis:

- **PTB-XL**: The PTB-XL ECG dataset consists of 21,799 clinical 12-lead ECG 10 second recordings from 18,869 patients. The dataset is annotated by two cardiologists and includes 71 different ECG annotations. [42]
- **CPSC**: The 4th China Physiological Signal Challenge 2021 (CPSC 2021) was introduced to promote the development of algorithms for detecting paroxysmal atrial fibrillation (PAF). The dateset includes ECG recordings with Artial Fibriliation (AF) rhythms, non-AF rhythms, and their transitions. [43]
- **G12EC**: The Georgia ECG Database provides a dataset from the Southeastern United States, including 10,344 12-lead ECG recordings, each with a duration of 10 seconds. [40]
- **INCART**: The INCART Database contains 75 annotated recordings from 32 sessions, each lasting 30 minutes with 12 standard ECG leads. The data was collected from patients tested for coronary artery disease, with a focus on ischemia and conduction abnormalities. [40]
- **MIT-BIH**: The MIT-BIH Arrhythmia Database consists of 48 half-hour two-channel ECG recordings from 48 subjects. The database includes both common and rare arrhythmias, annotated by multiple cardiologists to ensure its accuracy. [44]

For the annotated datasets INCART and MIT-BIH the beats were categorized in a set of 5 classes: Normal beat (N), Supraventricular Ectopic Beat (S), Ventricular Ectopic Beat (V), Fusion beat (F) and Unknown beat (Q). This division of heartbeats is compliant with the ANSI/AAMI EC57:1998 standard. [45] Both datasets contain annotations for 15 different arrhythmia, of which the allocated classes can be found in Table 3-3.

To be able to test the model's capacity to generalize to new patients, the training and test set was split inter-patiently for the MIT-BIH dataset. The split in patients is equal to the one found in de Chazal et al. [45], where a comparable division of arrhythmias was maintained

Table 3-3: Heartbeat classes

Class	Symbol	Members
Normal	N	N Normal L Left bundle branch block R Right bundle branch block e Atrial escape beat j Nodal (junctional) escape beat
Supraventricular Ectopic Beat	S	A Atrial premature beat a Aberrated atrial premature beat J Nodal (junctional) premature beat S Supraventricular premature or ectopic beat (atrial or nodal)
Ventricular Ectopic Beat	V	V Premature ventricular contraction E Ventricular escape beat
Fusion beat	F	F Fusion of ventricular and normal beat
Unknown	Q	/ Paced beat f Fusion of paced and normal beat Q Unclassifiable

between the training and test set. For the validation set four patients were chosen randomly from the training set.
The division of patients over the training, validation and test set can be found in Table 3-4. The four patients with pacemakers, 102, 104, 107 and 217, were excluded from the data.

Table 3-4: MIT-BIH dataset inter-patient division.

Database		Patient numbers
DS1	DS11 (training set)	101, 106, 108, 109, 112, 114, 115, 116, 119, 122, 124, 203, 205, 208, 209, 215, 220, 223
	DS12 (validation set)	118, 201, 207, 230
DS2 (test set)		100, 103, 105, 111, 113, 117, 121, 123, 200, 202, 210, 212, 213, 214, 219, 221, 222, 228, 231, 232, 233, 234

It was not necessary to split the INCART database inter-patiently as it was only used as a training set. For this purpose it was concatenated with the DS11 training set.
Table 3-5 shows the data distribution of the labeled datasets.

Table 3-5: Data distribution for the labeled MIT-BIH and INCART datasets.

Dataset	N	S	V	F	Q
DS11 (training)	38,217	613	3,362	412	8
DS12 (val)	7,591	328	425	2	0
DS2 (test)	44,198	1,835	3,218	388	7
INCART (training)	153,478	1,957	19,994	219	6

The arrhythmia distribution of the unlabeled dataset can found in Figure 1 of Alday et al. [46]. Important to note is the abundance of normal samples including normal sinus rhythm, sinus bradycardia and sinus tachycardiam, which all result in the same isolated beat ECG signal.

3-1-2 Preprocessing

Before data can be used to train and evaluate a model, it must be subjected to some preliminary processing steps. The pipeline of data processing can be divided into four subsequent steps: filtering, peak detection, segment extraction and normalization. Since not all datasets have 12-leads, only lead I will be considered for consistency. This section uses data from the PTB-XL dataset to illustrate the preprocessing steps, which are applied uniformly across all datasets to generate the input images.

Filtering

It is common to filter ECG data with a bandpass filter that has a cutoff frequency at the low end of 0.1 Hz and a high cutoff frequency of 100 Hz. [15] [45] Figure 3-1 shows the Fourier transform of a randomly selected signal, showing no significant peaks at frequencies higher than 100 Hz, indicating that little information is present in that range. Additionally, frequencies lower than 0.1 Hz generally do not provide relevant information about the heart's electrical activity as the high intensity in this range often correlates with patient movements, such as breathing.

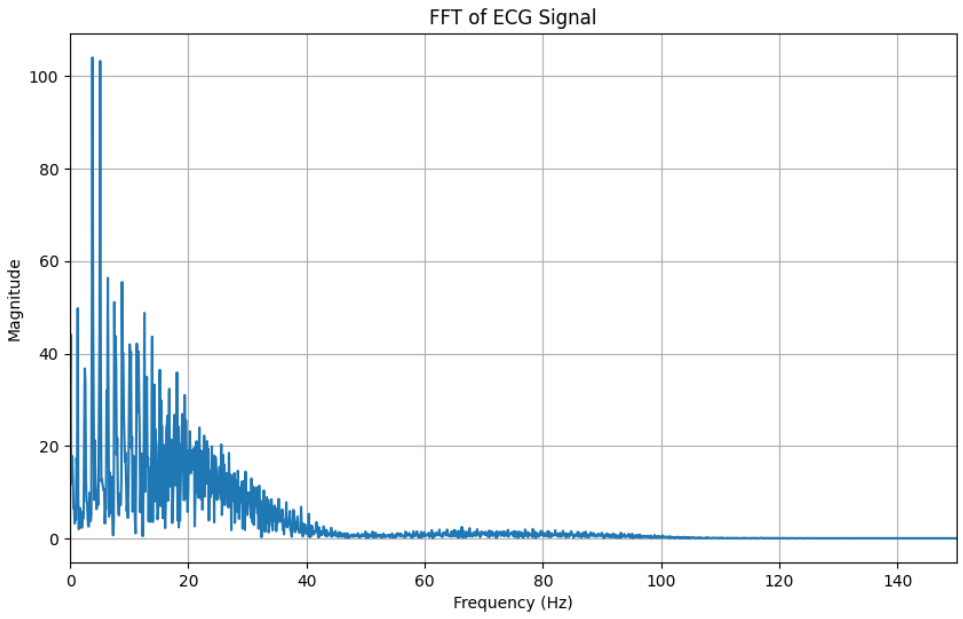


Figure 3-1: Fourier transform of random selected input signal before filtering. The bandpass filter will use a lower cutoff of 0.1Hz and a high cutoff of 100Hz. Frequencies over 100Hz contribute less to the signal due to their have low magnitudes.

Figure 3-2 shows the overlap of the filtered signal over the unfiltered signal. Resampling was not necessary in this case as the signals are interpolated when converted to images.

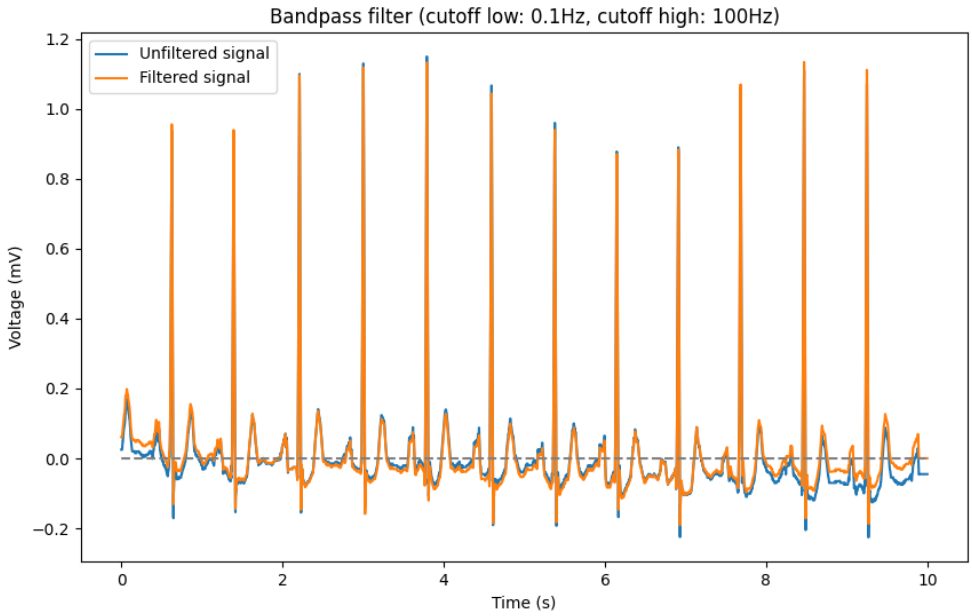


Figure 3-2: Bandpass filtered signal overlayed on unfiltered signal. The effect of the highpass filter can be seen by smaller deviations from the mean. The lowpass filter reduces the noise.

Peak detection and segment extraction

As the input to the model are isolated beat images, the heartbeat segments will have to be extracted from the signal. For this purpose the peaks of the R-waves are detected to then isolate an area around the R-wave, which will be considered as one beat. In Figure 3-3 the detected peaks locations are plotted over the processed signal.
After the peaks are detected from a signal, the center most beat is located. From this point four beats to the left and four beats to the right are selected and cut at 200 samples after and 150 samples before their respective R waves. The selection of the 8 center most beats ensures no artefacts that are commonly found at the beginning and end of a recording, are used in the training process. If two subsequent beats fall within the cut off range of 200 samples, the cutoff range is updated to ignore the samples towards the next beat. This is to ensure extracted segments contain no double beats.

Normalization

After beat extraction, the signal is normalized to ensure consistency across all data samples. This normalization process is necessary such that each beat is comparable and that the model can learn effectively from the patterns present in the data without being influenced by amplitude variations. Normalization involves scaling the amplitude of the ECG signal so that it fits within a standard range, typically between 0 and 1, which is required to transform the signal to an image. Figure 3-4 shows a normalized and isolated beat. This signal will be converted to a 128×128 image to function as the input to the model. Figure 3-5 shows four randomly selected input images.

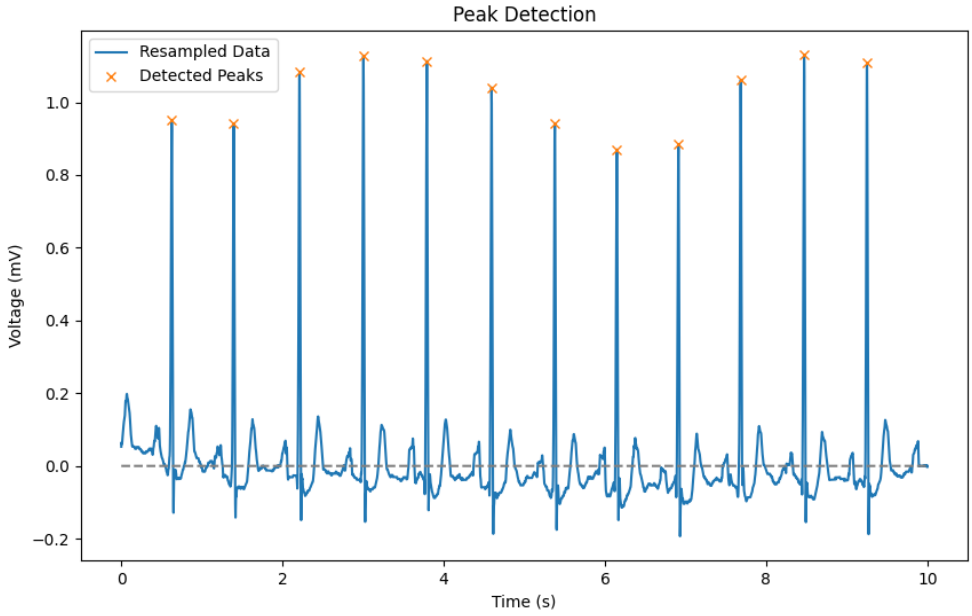


Figure 3-3: Peak indication after the detection algorithm.

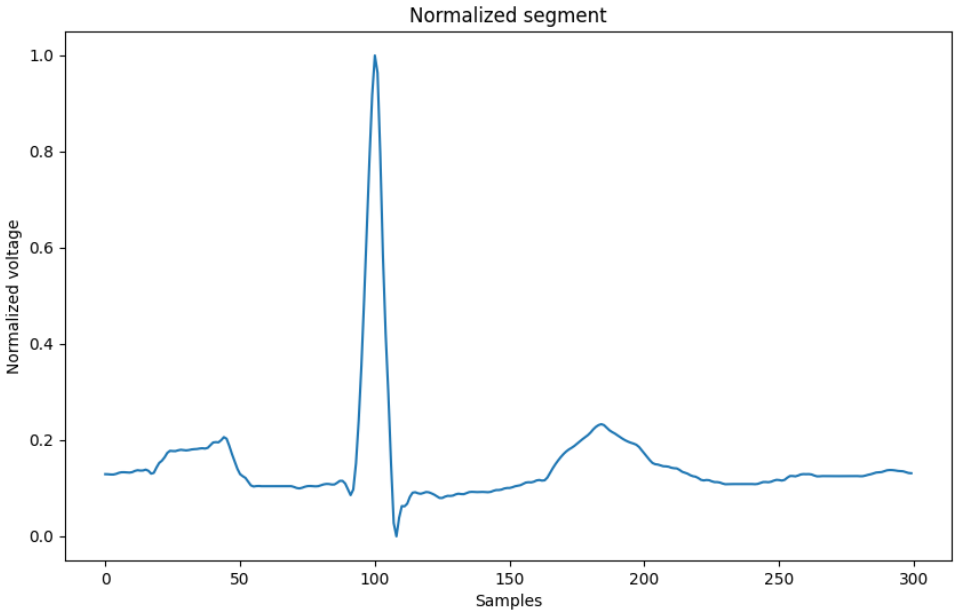
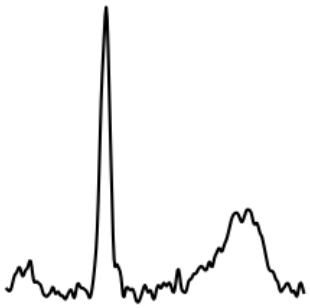
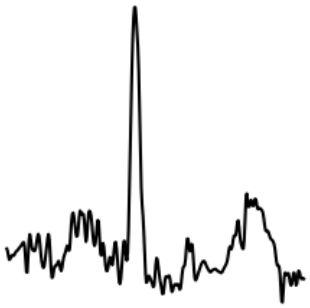


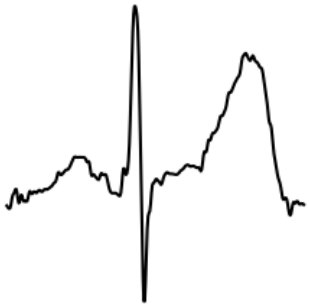
Figure 3-4: Normalized segment with a cutoff of 100 samples to the left of the R-peak and 200 samples to the right.



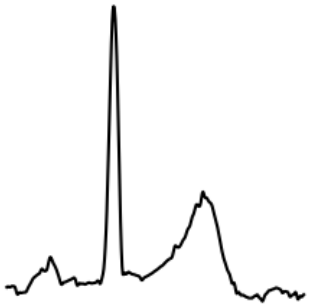
(a) Input image 1.



(b) Input image 2.



(c) Input image 3.



(d) Input image 4.

Figure 3-5: Random sample of input images. Each sample has varying amounts of noise and defects.

3-2 Experiment 1: Autoencoder model comparison

The primary objective of the first experiment in this thesis is to find the most suitable model for ECG feature extraction. Four different models will be evaluated: a basic implementation of an autoencoder, ResNet, ConvNeXt, and U-Net. As ResNet and ConvNeXt were originally designed to only be an encoder, the architectures were adapted to autoencoder format.

The basic CNN autoencoder architecture functions as a benchmark for comparison to a simple convolutional layer structure. ResNet introduces residual connections, which are particularly useful in handling vanishing or exploding gradients that often occur in deep networks. The ConvNeXt architecture takes some techniques from Vision Transformers (ViTs) and combines it with conventional structures of CNNs. Finally, U-Net is evaluated, which is known for its architecture to maintain high-resolution features throughout the model.

The performance of each model will be evaluated based on the mean squared error (MSE) between the original and reconstructed images. This metric is a measure of reconstruction accuracy, with lower MSE values indicating better performance. To ensure a fair comparison, training parameters such as learning rate, batch size, and the number of epochs were kept as consistent as possible across all models.

The aim of this experiment is to determine which architecture is most effective in the extraction of the most optimal features for the purpose of ECG classification. This experiment does not include the optimization of the model's accuracy on ECG classification, it is merely to choose an optimal architecture to use in subsequent experiments where tensor decompositions will be applied.

3-2-1 Model introductions

The models that are compared in this thesis were chosen each based on their performance, typical applications and complexity. The depth and dimensions of the models were chosen to be as similar as possible such that the comparison was fair. However, it must be taken into account that some models have different requirements to perform well.

The following sections give brief introductions on the models that were compared for this thesis, including diagrams of the corresponding architectures. In these diagrams the blue rectangles indicate a convolutional layer. This layer consists of a convolution operation, a batch normalization and a GELU activation layer, unless indicated otherwise. The number on top of the blue rectangles indicates the number of output channels of the convolution operation, the number of input channels can be derived from the operation prior. Furthermore, the red arrows indicate a max pooling layer and a green arrow indicates an upsampling layer. Both these operations scale with a factor of 2 unless specified otherwise.

Basic CNN

The basic CNN model illustrated in Figure 3-6 features three dimensions for both the encoder and decoder, each with two convolutional layers. The channels are increased in steps from 1 in the input image to 128 in the latent space. The three max pooling layers result in a compression of the 128×128 input image to 16×16 feature maps. The decoder then increases the spatial dimension via upsampling operations and uses the extracted features to reconstruct the image.

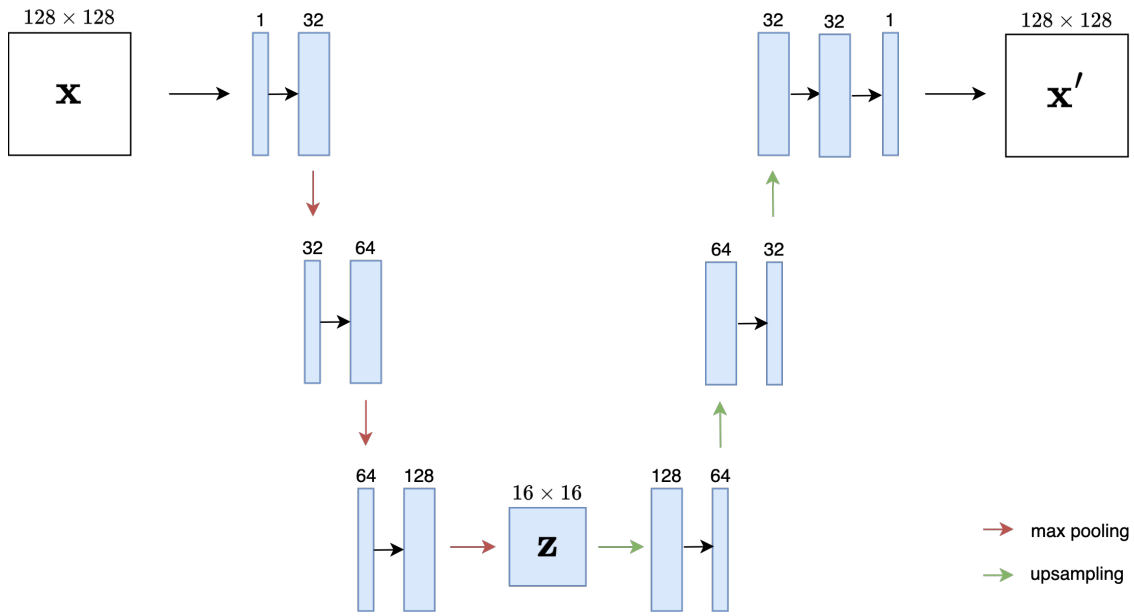


Figure 3-6: Basic model architecture consisting of three layers for both the encoder and decoder. Each layer consists of two convolutions and a down- or upsampling operation.

ResNet

Originally, ResNet only consists of an encoder part that is directly attached to a fully connected layer. For this thesis the ResNet architecture was converted to also include a decoder such that the model could be used in the reconstruction of the images. ResNet was primarily chosen to test the impact of residual layers on the reconstruction, while maintaining a relatively simple architecture.[47]

The architecture is comparable to the basic CNN introduced earlier in this section. However, instead of conventional convolutional layers, this includes residual blocks. ResNet's residual blocks are designed to address the vanishing and exploding gradient problem. Each residual block includes shortcut connections that skip one or more layers. Figure 3-7 gives the architecture of such a residual block.

Figure 3-8 shows how these blocks are placed throughout the autoencoder network. The max-pool operations are performed with a kernel size of 3, a stride of 2 and a padding size of 1. Furthermore, the first convolution in the encoder increases the spatial dimensions by a factor of 2, whereas the last convolution in the decoder reduces the spatial dimensions by a factor of 2.

ConvNeXt

The ConvNeXt model was designed to improve on the performance of CNNs by using certain aspects of new architectures like Vision Transformers (ViTs)[48]. The techniques applied from ViTs include larger kernel sizes, layer normalization, depthwise separable convolutions and inverted bottlenecks.[14]

The architecture of ConvNeXt starts with a ResNet-like structure, but gradually implements several changes. The ConvNeXt block is designed to mimic the multi-head self-attention mechanism of transformers but implemented entirely with convolutions. Figure 3-9 gives an overview

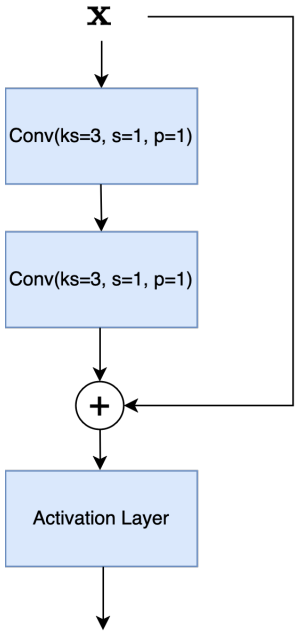


Figure 3-7: Overview of the Residual Block from the ResNet architecture containing two convolutions. These are followed by a skip connection and an activation layer.

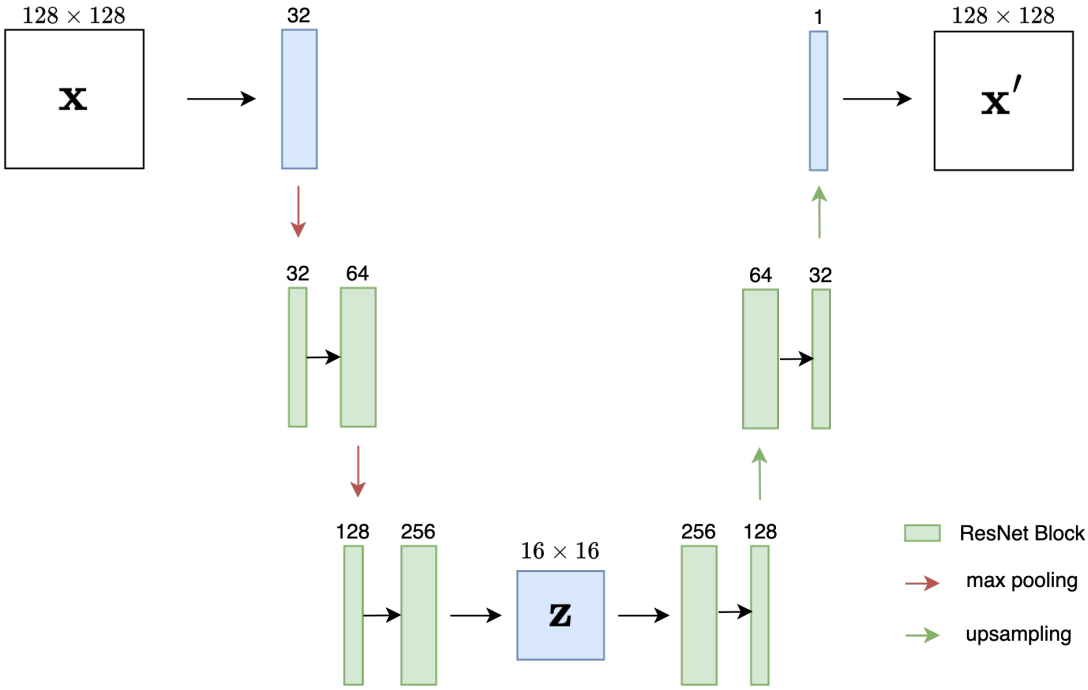


Figure 3-8: ResNet architecture featuring three layers per encoder and decoder, where the second and third layer contain two ResNet Blocks each.

of the ConvNeXt block, which features depthwise convolutions and a series of 1×1 convolutions that mix spatial and channel information.

Figure 3-10 shows the architecture of the autoencoder adaption of the ConvNeXt network. The depth ratio for the ConvNeXt model is commonly chosen in the ratio of $1:1:3:1$. To keep the

ConvNeXt model comparable to the others, the minimal depth was chosen to be $[1, 1, 3, 1]$. As the ConvNeXt model already uses a high number of parameters, this minimal depth maintains the accuracy of the comparison. This is indicated in the figure by a 3x for the third layer.

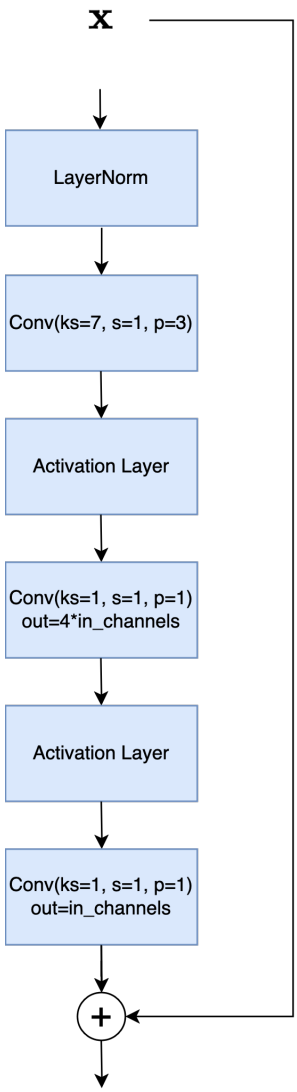


Figure 3-9: Overview of the ConvNeXt Residual Block. The input is first normalized over its layers, after which it is subjected to a high kernel size convolution, followed by an activation layer. An expanding pointwise convolutions maps the intermediate image to 4 times the number of input channels. The GELU activation function is applied before it is convoluted back to its original number of input channels. Finally, the skip connection adds the original input image to the output.

U-Net

The U-Net architecture was designed with the need for accurate biomedical image segmentation in mind, where annotated training samples are limited. U-Net is an autoencoder by original design, which includes skip connections between corresponding layers in the encoder and decoder. These skip connections transfer feature maps from the encoder to the decoder, such that it retains the high level spatial features in the reconstruction part. The U-Net has shown promising results in various challenges, such as neuronal structure segmentation in electron microscopy

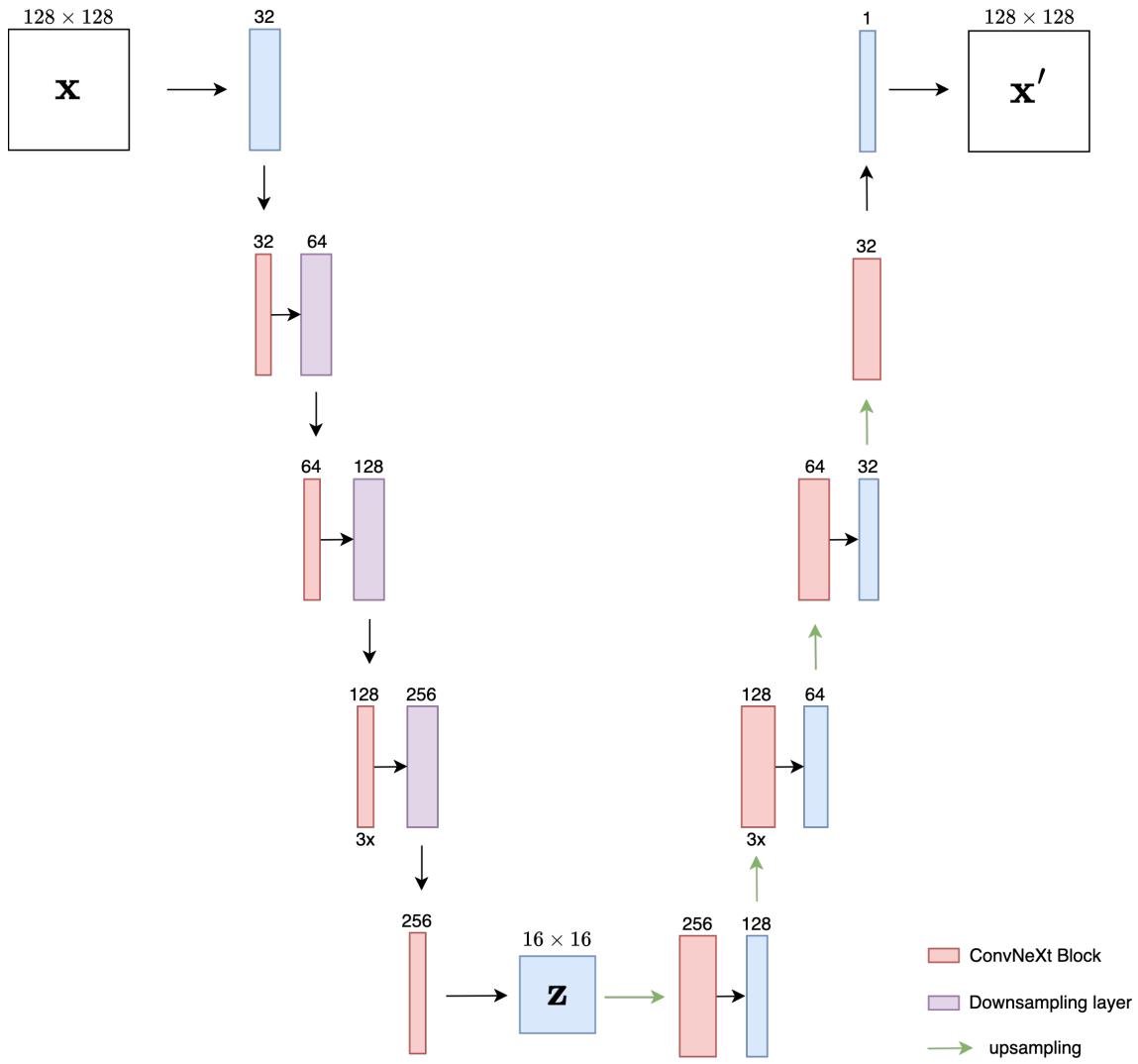


Figure 3-10: ConvNext autoencoder architecture. The model consists of ConvNeXt block in each layer of the encoder and decoder. Instead of max pooling in the encoder, the architecture used downsampling layers to reduce the spatial dimensions.

images, significantly outperforming previous methods in both speed and accuracy.[49]

Figure 3-11 illustrates the U-Net architecture. Each layer in the encoder includes concatenation with the corresponding layer in the decoder, indicated by the white boxes attached to the blue convolutions, with the corresponding feature map from the encoder. The U-Net encoder/decoder layers include three convolutional layers compared to the typical two layers found in ResNet and ConvNeXt.

Classifier model

The classifier model is a simple multi-layer perceptron model, consisting of an input layer, hidden layer and output layer. The first layer is followed by a one dimensional batch norm and a GELU activation function. The input to the classifier model is the output of the encoder. During the training and inference process the encoder is prepended to the classifier. Figure 3-12 shows the setup of the encoder classifier combination.

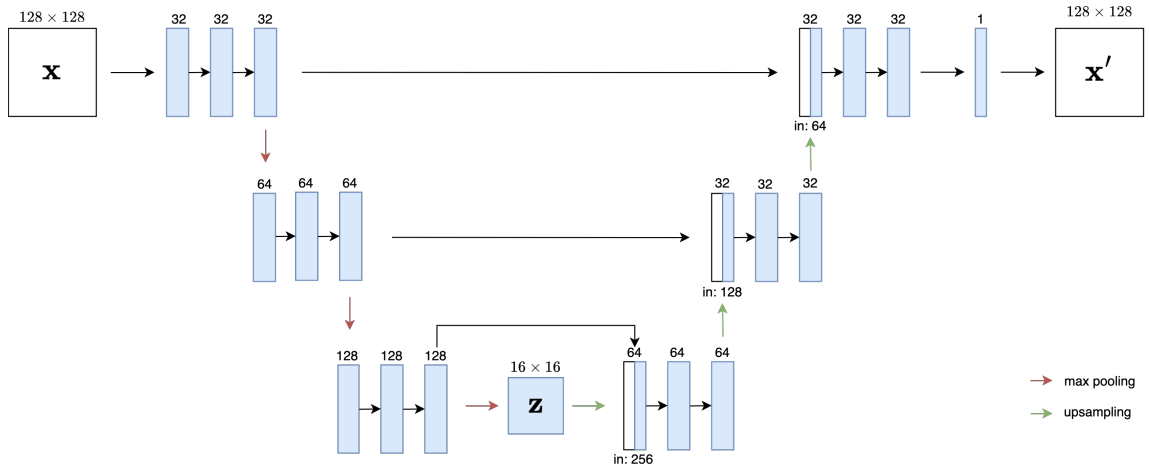


Figure 3-11: The UNet architecture uses three convolutions per layer instead of the regular two. It furthermore employs skip connections between the respective layers in the encoder and decoder. the concatenated feature maps are indicated by the white boxes.

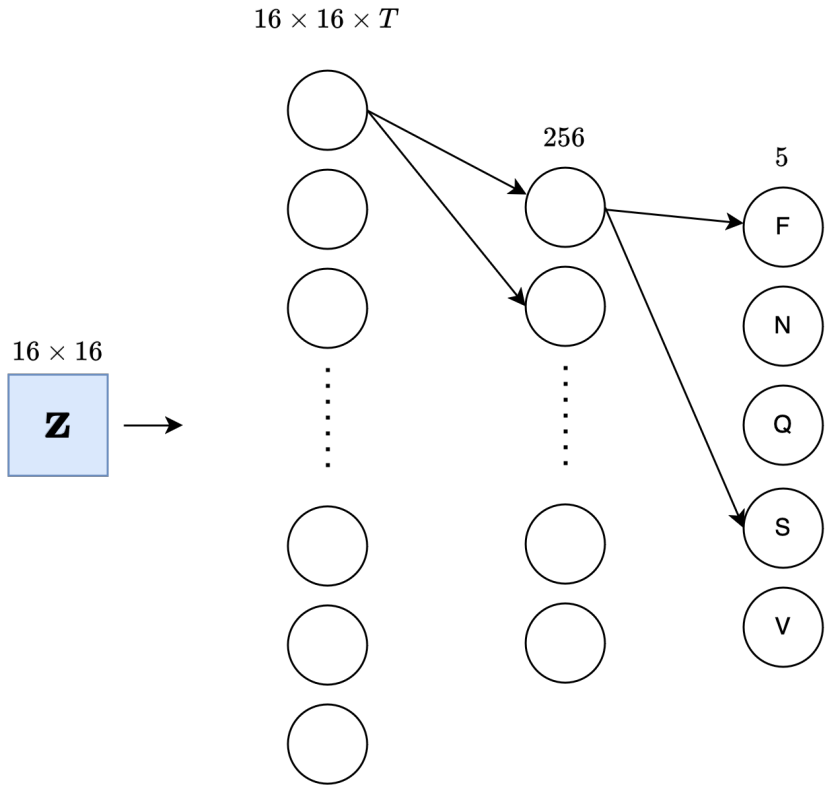


Figure 3-12: Classifier architecture with latent space input. The network consists of an input layer with dimension $16 \times 16 \times T$, a single hidden layer with dimension 256 and output layer with the five classes: F, N, Q, S and V. Example shows only a few connections for readability, but each neuron is connected.

Evaluation metric

Mean Squared Error (MSE) was chosen as the evaluation metric for the performance of the autoencoders in this thesis. MSE measures the average squared difference between the input

values and the reconstructed values,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \hat{\mathbf{x}}_i)^2 \tag{3-1}$$

where $\mathbf{x}_i$ represents the original input values, $\hat{\mathbf{x}}_i$ denotes the reconstructed values, and n is the number of data points.
Besides the MSE, the reconstructed images will also be compared visually with the input images.

3-3 Experiment 2: Tensor decomposed autoencoder

This experiment is designed to evaluate the change in model performance after the application of the CPD in the autoencoder layers. The decomposition will be applied to the convolution operations within each of the layers. Various compression ratios will be tested to determine their effect on model accuracy and computational efficiency. The objective is to find an optimal balance between model complexity and performance.

Tensor decompositions add a range of benefits to the optimization process of training neural nets, for example reduced computational load, faster training times and regularization. This experiment will highlight the effects of regularization in particular. Regularization is important to address in training deep neural nets as it prevents overfitting, which occurs when a model learns noise and non-extendable features from the training data. This can especially be of importance when handling datasets with a disbalanced class distribution, due to a tendency to overfit on the abundant classes.

Familiar forms of regularization are L1 and L2, which add a penalty term to the loss function, or Dropout layers, which randomly set weights to zero in the backpropagation process.[50][51] The application of tensor decompositions can also be seen as a form of regularization, where a number of parameters are eliminated from the kernel tensors. The main difference between the use of tensor decompositions as regularization and the conventional techniques, is that L1, L2 and Dropout do not reduce the total number of parameters in the model and can even increase the computational complexity. Besides regularization, tensor decompositions can also improve convergence speed due to a reduction in the total number of parameters. Faster convergence often implies more stable updates during the training process, resulting in a smoother optimization landscape. This stability can prevent the model from oscillating and overfitting to specific patterns in the training data, which in turn can be seen as a form of regularization.

The number of parameters in a kernel for a CP model of rank-R is given by $R(2D + S + T)$, where D represents the dimension of the input, S the dimension of the output, and T the size of the tensor. For uncompressed kernels the dimensions are represented as $T \times S \times D \times D$. The compression ratio per convolutional layer is then calculated as,

$$\text{compression ratio} = \frac{TSD^2}{R(2D + S + T)}. \tag{3-2}$$

The kernels in the uncompressed model from experiment 1 all have a kernel size of 3×3, resulting in a maximum kernel dimension in the final layer of $512 \cdot 256 \cdot 3 \cdot 3 = 1,179,648$. The decomposed kernel would then exist of $R(2 \cdot 3 + 256 + 512) = 774R$ elements.
For the entire network the compression ratio is calculated as the ratio between the number of parameters in the uncompressed network and the number of parameters in the tensor decomposed network. For this model configuration, a rank of $R = 5$ would result in a compression ratio of 210×.

The weights of the kernels were initialized randomly after the decompositions were applied. This reduces the computational load as the original random convolution weights were not approximated.

This experiment is divided into three parts: an accuracy comparison per compression ratio, a training convergence comparison with the uncompressed model and an evaluation on the data efficiency.

3-3-1 Experiment 2.1: Accuracy comparison over various compression ratios

The first part of this experiment focuses on the effect the CPD has on the accuracy of the arrhythmia classification. The following hypothesis will be tested: CP-decomposed autoencoders intended for ECG classification can maintain a similar level of accuracy compared to uncompressed autoencoders.

The network is evaluated for various compression ratios: 210.5×, 118.8×, 82.7×, 63.5×, 51.5×, 37.4×, 26.5×, 17.8×, 13.4×, 10.7× and 8.5×, which are a result from evenly distributed rank values. Each ratio will be trained three times and averaged to account for the randomness of the model optimization process. This ensures the results are not based on a local minimum reached by some random set of initial weights. The average epoch duration, number of FLOPs for a forward pass, average accuracy, average MSE and various performance metrics will be collected.

The secondary goal of this experiment is to find an optimal compression ratio that balances between predictive accuracy and computational complexity. This compression ratio will then be used in the following experiments.

3-3-2 Experiment 2.2: Convergence speed comparison

Due to the relatively large set of normal beats in the datasets, there are expected to be a lot of redundancies in the network. The regularization introduced by tensor decompositions may lead to a faster convergence of tensor decomposed networks as opposed to uncompressed networks by relieving the model of these redundancies.

The primary objective of this experiment is to test the hypothesis: CP-decomposed autoencoder networks will lead to faster convergence compared to uncompressed autoencoder networks. This will be tested by comparing the validation loss progress over a three run average. The model is said to be converged after no significant improvements on the validation loss are made and the standard deviation is low.

3-3-3 Experiment 2.3: Data efficiency comparison

The workings of the regularization effect of tensor decompositions that was explained in the introduction of this section, could also result in an increase in data efficiency. Data efficiency learning increases the capability of a model to understand complex relations without the need of a high number of training samples. The regularization effect is expected to increase the model's ability to generalize to new data. By implementing a lower-rank representation, CPD can potentially prevent overfitting, which allows the model to achieve better performance with fewer training samples.

The primary objective of this third sub-experiment is to test the hypothesis: Superior results can be achieved with CP-decomposed autoencoders when limiting the number of training samples compared to uncompressed autoencoder networks.

Given the large number of data samples available in the datasets, the reduction in parameters has to be significant to see an effect. This experiment was done at various percentages of

the original number of data samples, at [2.5%, 5%, 10%, 20%]. The uncompressed and CP-decomposed models are optimized three times and averaged to obtain its results. To test the hypothesis, the performance decrease is considered with regards to the optimization of the complete dataset for each respective model. The performance is defined by the MSE on the test set and the validation loss decline.

3-3-4 Evaluation Metrics

To assess the performance of transfer learning model in classifying ECG images, several evaluation metrics are used:

Accuracy

Accuracy is the most straightforward metric, representing the proportion of correctly identified classes. It gives a general measure of how well the model is performing overall.

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{number of samples}}$$

Confusion Matrix

A confusion matrix maps the true values to the predicted values, allowing to observe the distribution of prediction per class. The structure of the matrix is as follows,

	Class 1	Class 2	$\cdots$	Class n
Class 1	TP_1	$FP_{1,2}$	$\cdots$	$FP_{1,n}$
Class 2	$FN_{2,1}$	TP_2	$\cdots$	$FP_{2,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
Class n	$FN_{n,1}$	$FN_{n,2}$	$\cdots$	TP_n

(3-3)

where TP_i are the number of true positives for class i, $FP_{i,j}$ denote the false positives for class i identified as class j and $FN_{j,i}$ denote the false negatives for class j identified as class i.
The confusion matrix gives a better understanding of the underlying distribution of mistakes for misclassification.

3-4 Experimental setup

This section gives an overview of the general setup for the experiments implemented with PyTorch. The overview includes data preparation, model training, validation, and testing.

Data preparation

The first step in the data preparation process is resizing and transforming the ECG images. Each image is transformed into a tensor suitable for model input, ensuring that the inputs are uniform across all datasets.

Following the transformations, the datasets are split into training, validation, and test sets. The unsupervised datasets from PTB-XL, G12EC, and CPSC are combined and then divided into training, validation, and test subsets, following a ratio of roughly 80 : 10 : 10 for the

training, test and validation set respectively. The labeled datasets, MIT-BIH and INCART, are prepared separately for supervised training. The split ratio for the labeled sets is described at the beginning of this chapter in the datasets section.

Autoencoder training

The models are trained on four CUDA devices, which requires a slightly different setup than CPU training. To speed up the training process for example, half precision (float16) is chosen over single precision (float32). For CUDA devices the automated mixed precision package[52] has to be incorporated in the model in order to use half precision. Furthermore, the model has to be wrapped in a parallel programming object to ensure the GPUs can perform optimally during training.

The training criterion used is the Mean Squared Error (MSE) for the autoencoder and Cross Entropy Loss for the classifier. Both are standard methods for evaluating autoencoders and classifiers. Futerhmore, the Adam optimizer is chosen due to its efficiency and effectiveness in handling sparse gradients, making it suitable for this application. Stochastic Gradient Descent (SGD) was also tested as an optimizer. However, the Adam optimizer converged to a much lower MSE than the SGD optimizer was able to do. Figure 3-13 shows a comparison between the two optimizers for the same set of model parameters.

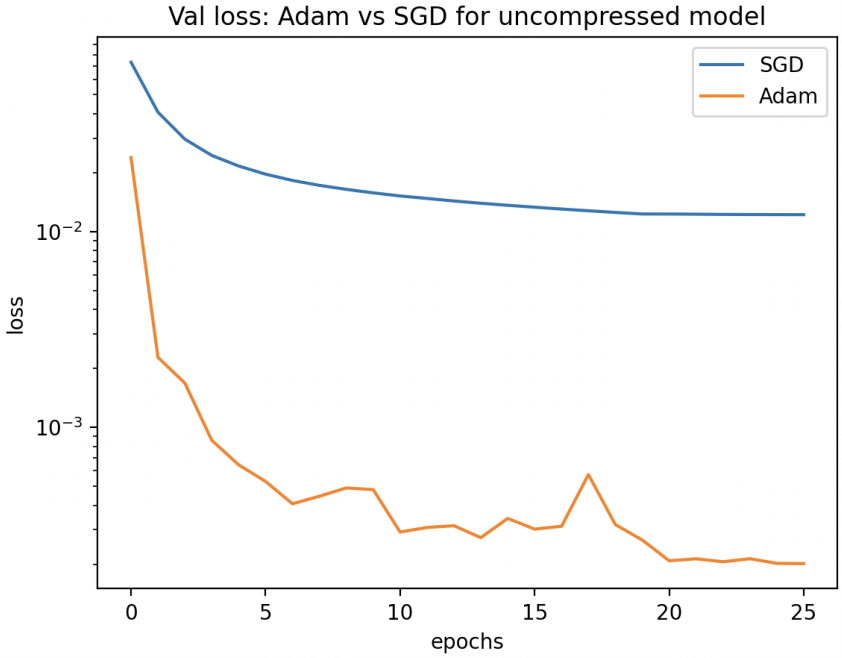


Figure 3-13: Loss function for both the SGD and Adam optimizer. The SGD loss has a smoother optimization landscape, but converges to a much higher MSE. The Adam optimizer has a much steeper decline, but is less smooth.

To dynamically adjust the learning rate during training, a StepLR scheduler is implemented. This scheduler reduces the learning rate at predefined intervals, helping to avoid overfitting.

During training, the model processes the data in batches. For each batch, the model's predictions are compared against the actual data using the loss function. The gradients of the

loss with respect to the model's parameters are computed using autograd, and the optimizer updates the parameters to minimize the loss. This process is repeated for a specified number of epochs, with the learning rate being adjusted periodically by the scheduler.

To prevent potential data loss, the model's state is saved every 5 epochs. This periodic saving allows for possible recovery and evaluation of the model at intermediate training steps.

Validation of the model is conducted in a manner similar to training but without computing gradients. The validation loss is tracked to monitor the model's progress and to ensure it is not overfitting on the training data.

Testing the model involves evaluating its performance on the test set, also without gradient computation.

Classifier training

Post-training, the encoder part of the autoencoder, which has learned to compress and reconstruct ECG images, is used as a feature extractor for a classifier. The encoder is prepended to a classifier layer, with its parameters frozen to retain the learned features without further updates.

The classifier follows a similar training process to the autoencoder, consisting of a training, validation and testing phase. For the classifier the same learning rate scheduler is used, which is a step function. Furthermore, the accuracy is calculated by applying a maximum function on the output of the classifier.

Chapter 4

Results and discussion

4-1 Experiment 1: Autoencoder model comparison

The results for experiment 1 can be divided into four comparisons: an error comparison, a visual comparison, a performance comparison and a predictive accuracy comparison between the uncompressed network and the CP-decomposed networks. The error comparison will look at the training and validation loss progression over the epochs, while also looking at the averaged MSE and accuracy on the test set for each of the models over each of its three runs. The visual comparison is done to ensure the model is able to reconstruct the input images well enough for the classification step. Finally, the performance is taken into account to compare metrics such as epoch duration and total number of FLOPs.

In Appendix A an overview of all training settings for the first experiment can be found.

4-1-1 Error comparison

Figure 4-1 shows the validation loss of the basic, ResNet, ConvNeXt and U-Net models over a 25 epoch run on a logarithmic scale. Additionally, the MSE on the test set is given in Table 4-1.
The U-Net model clearly outperforms the rest of the models with a 10× improvement on its validation and test set MSE. The validation loss converges slightly slower compared to the other models, however it converges to a much lower final MSE value. The validation loss of the basic model converges slightly worse than the U-Net model. It is however more stable in its decline. The ResNet model converges relatively quickly to a higher MSE than the U-Net and basic models. It is however important to note that the MSE on the test set of the ResNet model gives a comparable result to the basic model. Finally, the ConvNext model has the highest MSE on the test set and validation set of all four models.

4-1-2 Image reconstruction comparison

Figure 4-2 shows the input image that is to be reconstructed by each of the trained models. The image contains a normal ECG signal with some high frequenciy oscillations around the P-wave.

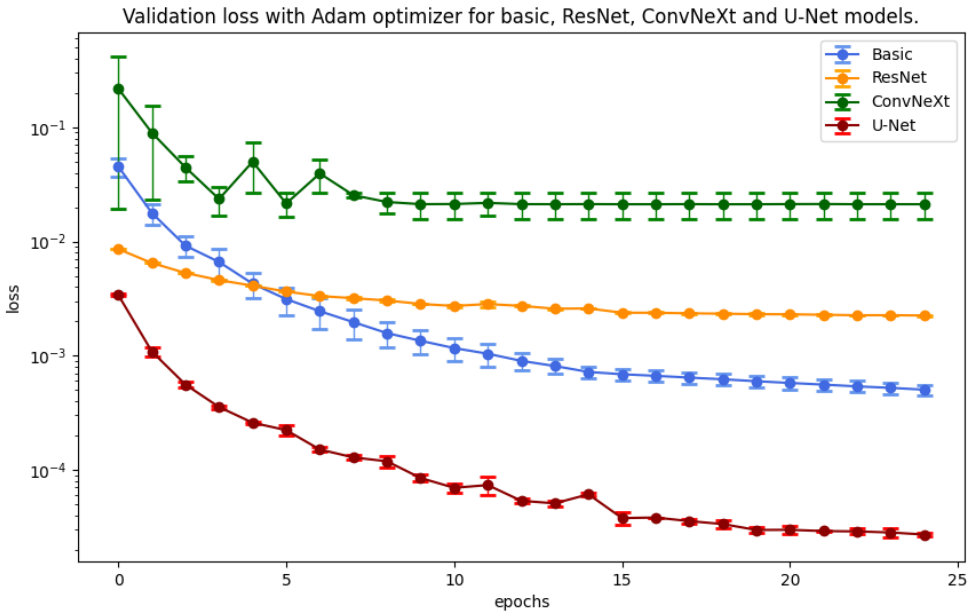


Figure 4-1: Three run averaged validation loss for the basic, ResNet, ConvNeXt and U-Net models with error bars. The U-Net model has the lowest convergence point. The ConvNeXt and ResNet model both convergence quickly.

Table 4-1: Averaged MSE on the test set of each of the models.

Model	MSE
Basic	2.2e−4
ResNet	2.4e−4
ConvNeXt	3.6e−3
U-Net	3.0e−5

Figure 4-3 displays the reconstructed images for the basic model (a), the U-Net model (b), the ResNet model (c) and the ConvNeXt model (d).

The ConvNeXt model is only able to reconstruct the general direction of the ECG signal, capturing the lowest frequencies. It is furthermore unable to reconstruct the white background. As any higher frequencies are missing in the reconstructed area of the image, it is most likely not suitable to use for any classification purposes.

The ResNet model captures the signal relatively well, the P-wave, QRS-complex and T wave are all clearly present. The higher frequency activity in the input image in for example the P-wave is not reconstructed. This may be an issue for the classification step as it could be the abnormalities can be identified by only lower frequencies.

Both the basic and U-Net model are able to reconstruct the input image well, capturing both the higher and lower frequencies of the signal. U-Net performs slightly better in the detailed reconstruction, however this will likely not be a significant difference for the arrhythmia detection.

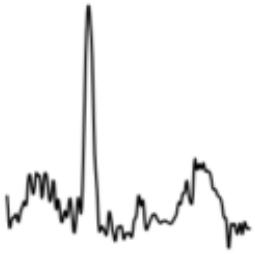
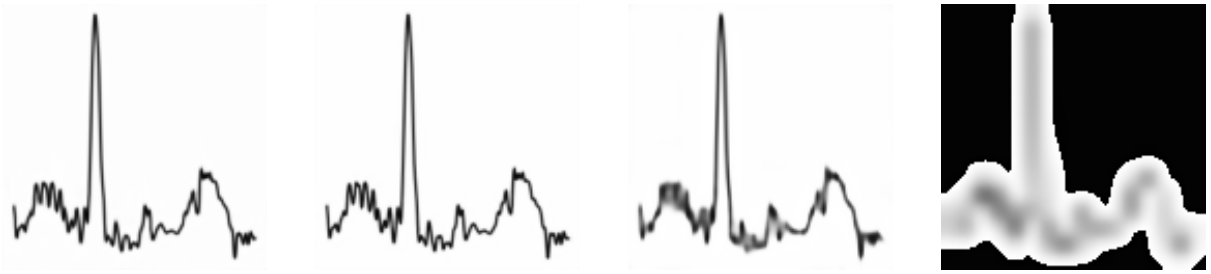


Figure 4-2: Input image



(a) Basic **(b)** U-Net **(c)** ResNet **(d)** ConvNeXt

Figure 4-3: Reconstructed images for the basic, U-Net, ResNet and ConvNeXt model. The basic and U-Net model are able to reconstruct the high frequencies of the singal. The ResNet model has more trouble with the higher frequencies. The ConvNeXt model is only able to reconstruct the general contours of the signal.

4-1-3 Performance comparison

Table 4-2 gives the total number of parameters, duration of an epoch and the number of FLOPs for a single image forward pass. The number of parameters between the Basic, U-Net and ConvNeXt models are similar, where ResNet has about double the number of parameters. The duration of an epoch and number of FLOPs are considerably higher for the ConvNeXt model compared to the others. This is likely due to the use of expanding pointwise convolutions, meaning a kernel size of 1. This small kernel size will not increase the number of parameters by much, however it does increase the number of calculations.

The inconsistency in the relation of FLOPs and epoch duration is due to FLOPs containing only the forward pass operations, where epoch duration also includes the backpropagation process. The reason ResNet has the largest number of parameters and the lowest number of FLOPs can be explained by an early reduction of spatial dimensions leading to a reduction in FLOPs. The residual blocks contain however a large number of filters, resulting in more parameters.

Table 4-2: Number of parameters, epoch duration and FLOPs for the basic, ResNet, ConvNeXt and U-Net models.

Model	Parameters	Epoch duration (s)	GFLOPs
Basic	$2.36 \cdot 10^6$	251	1.84
ResNet	$4.22 \cdot 10^6$	251	1.27
ConvNeXt	$2.01 \cdot 10^6$	645	4.11
U-Net	$3.12 \cdot 10^6$	242	1.56

4-1-4 Classification comparison

Table 4-3 presents the classification accuracies of each of the models. First of all, the differences between the accuracies are relatively small. This is probably due to the relatively low amount of data samples for the S, F and Q classes. Table 3-5 shows the S class contains only 2570 samples, the F class only 631 and the Q class 14 samples in the training set. Table 4-4 gives the confusion matrix of the basic model. The beats that are classified well by the model are the N and V classes. The S class can be identified by a defect in the RR-interval of subsequent beats. Due to the data being a single beat image, this is difficult for the model to learn. The S class can also be identified by a small defect in the P-wave. This defect is however harder to detect due to its subtleness. Both these issues complicate the identification process for the network. The confusion matrix also shows the F and Q classes being classified almost always as the N class.

The conclusion from these findings is the increasing difficulty of classification after the N and V classes have converged. To increase the accuracy, the model must mostly learn to classify the S class, which is difficult given the aforementioned observations.

ConvNeXt has the lowest classification performance. This can be explained by the results encountered earlier in this section where it became clear the reconstructed image does not contain enough detail to make accurate predictions. Despite U-Net's low MSE, its accuracy on classification is not performing as expected considering its ability to reconstruct the input images well. ResNet still has a considerably high accuracy given the absence of the higher frequencies in the reconstructed image and relatively high MSE. The basic model performed best with an accuracy of 94.39%.

Table 4-3: Classification accuracy for the basic, ResNet, ConvNeXt and U-Net models.

Model	Accuracy
Basic	94.39%
ResNet	93.98%
ConvNeXt	92.9%
U-Net	93.57%

Table 4-4: Confusion matrix for basic model.

	F	N	Q	S	V
F	0.0077	0.91	0	0	0.08
N	0.0007	0.98	0	0.006	0.0093
Q	0	0.43	0	0	0.57
S	0.006	0.8	0	0.11	0.09
V	0.0047	0.062	0	0.011	0.92

4-1-5 Model recommendations

The ConvNext model showed lowest predictive accuracy and image reconstruction capabilities while being most computationally heavy. In other classification tasks such as the ImageNet problem, ConvNeXt is able to achieve high accuracies. This is however only for an encoder classifier combination. Its low performance for this particular task could be due to the type of data it is applied to, but it is most likely due to the conversion to an autoencoder network. The

fundamental architecture of the ConvNeXt model may not have been transferable for image reconstruction from a compressed representation.

The U-Net model achieved great results in image reconstruction, however failed to predict the corresponding classes accurately. This is probably due to the use of skip connections, which work particularly well for image reconstruction, but does not result in the most optimal latent space for classification. Skip connections transfer feature maps from the encoder to the corresponding layer in the decoder. With the use of skip connections the model is not forced to learn the most important features for reconstruction as hierarchical information is also passed between the encoder and decoder, skipping the lower layers.

In both image reconstruction and classification, the ResNet model did not achieve high performance results, despite having the highest number of parameters of all models. For the image reconstruction a lot of high frequencies were missing, which probably resulted in the slightly lower predictive capabilities in the classification step. Like ConvNeXt, the ResNet model originally consisted of just an encoder. For this thesis its architecture was adapted to create an autoencoder network. This may have caused issues as its architecture was not designed for reconstruction purposes.

Finally, the basic model achieved the highest accuracy in the classification process, while having comparable performance results to ResNet and U-Net. The image reconstruction was on par with that of U-Net, but without the use of skip connections throughout the network. This likely resulted in a better feature extractor network and thus the higher accuracy. Furthermore, the basic model has comparable results to the baseline paper where this thesis was based on. Wang et al.[15] achieved a similar accuracy of 94.39%. Given these results, the basic model will be used in the second experiment.

4-2 Experiment 2: CP-decomposed autoencoder

Experiment 2 was divided into three sub-experiments: a classification performance comparison for various compression ratios, a convergence speed assessment between an uncompressed network and a decomposed network and a comparison of the performance drop when reducing the number of training samples for an uncompressed network and a decomposed network.

The configuration of the basic model chosen in experiment 1 was optimized to increase its performance. The number of channels was doubled from [32, 64, 128, 256] to [64, 128, 256, 512] and an extra layer was added in both the encoder and decoder. The latent space spatial dimension then becomes 8×8 instead of 16×16. These changes will not have resulted in a different comparison result for experiment 1 as the fundamental structures of the model remains unchanged. The configuration change resulted in an accuracy increase from 94.39% to 94.51%.

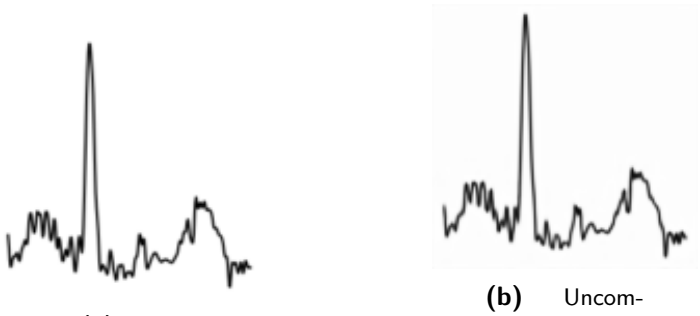
4-2-1 Experiment 2.1: Accuracy comparison over various compression ratios

Figure 4-4 shows the original image in **(a)** and the uncompressed reconstructed image in image **(b)**. Figure 4-5 then gives the reconstructed images for the tensor decomposed model at various compression ratios. For higher compression ratios the high frequencies are more difficult to reconstruct. In for example image 4-5**(b)**, the general contours of the P- and T-waves and QRS-complex are present in the reconstruction, however any defects can not be identified from this level of detail. The limitations for the tensor decomposed networks come from the lower dimensional vector representations, resulting in an inability to capture higher dimensional functions in the model. As the compression ratio decreases, the parameter space gets more higher dimensional and the model is able to reconstruct the images with more precision.

Figure 4-6 shows the averaged accuracies of the tensor decomposed networks in comparison with the uncompressed network. An increase in accuracy can be seen over the compression ratio scale. For high compression ratios, the accuracy drops below the uncompressed baseline value. Looking at the reconstructions at these higher compression ratios, it is expected to drop in accuracy due to the absence of certain ECG features. However, as the compression ratio decreases, the accuracy improves to the baseline of the uncompressed network, indicating significant redundancy in the model. This redundancy is likely due to overfitting on the N and V classes, which is the consequence of the large amount of samples for these classes. The model will be biased towards predicting the N and V classes as it has seen many more examples of those in the training process.

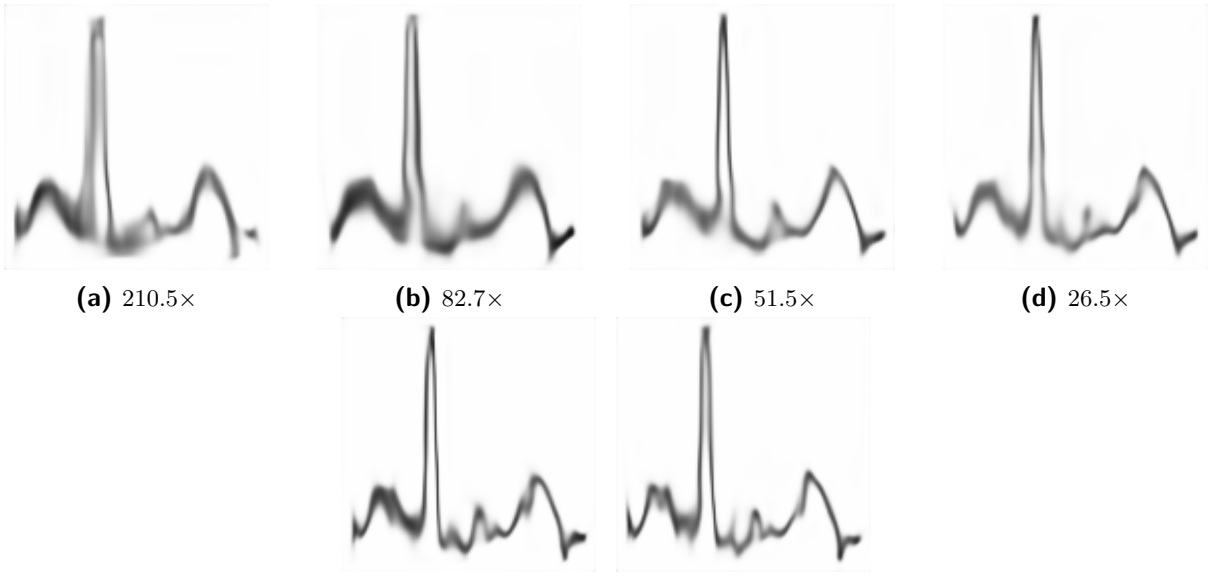
Tensor decompositions can reduce redundancies by limiting the amount of overfitting. The decomposition will not be applied to the supervised learning step directly. However, given the abundance of normal samples in the unlabeled dataset, the autoencoder will also suffer from generalization to normal beats.

When applying CPD to the model, it constraints the network to learn a function within the space spanned by the lower-dimensional vectors. This means it cannot represent all possible functions that the original, higher-dimensional tensor could. It is therefore restricted to a subspace of the original parameter space. This limitation in the complexity is a form of regularization that reduces the amount of overfitting on the training data. The reduction in overfitting will lead to more balanced feature extraction, shifting the focus from normal data reconstruction.



(a) Input **(b)** Uncompressed

Figure 4-4: Input image and uncompressed reconstruction.



(a) 210.5× **(b)** 82.7× **(c)** 51.5× **(d)** 26.5×

(e) 17.8× **(f)** 13.4×

Figure 4-5: Reconstructed images at various compression ratios. Higher compression ratios have difficulty with reconstructing the high dimensional functions. This gets increasingly better as the compression ratio is lowered.

The CP-rank, number of parameters, epoch duration and number of FLOPs for the uncompressed and compressed model are given in Table 4-5. The number of FLOPs is significantly reduced for the compressed models, however not by the same ratio as the number of parameters. This is due to the move from a single convolution operation to four subsequent smaller convolution operations, which requires more calculations. The added rank dimension of R in the decomposed convolution adds to the total number of FLOPs, which is absent in the uncompressed convolution.

Notable is the increase in epoch duration for compression ratios lower than 63.5×, which was expected to decrease, especially given the reduction in total FLOPs. This indicates that the additional time originates from the backpropagation step. The derivation of the number of FLOPs for the backpropagation of the CP-decomposed convolution is presented in Appendix B. The ratio of FLOPs between the CP-decomposed model and the uncompressed model for the backpropagation step is given by,

$$\text{ratio} = \frac{RS + 2DR^2 + RT}{TSD^2}. \tag{4-1}$$

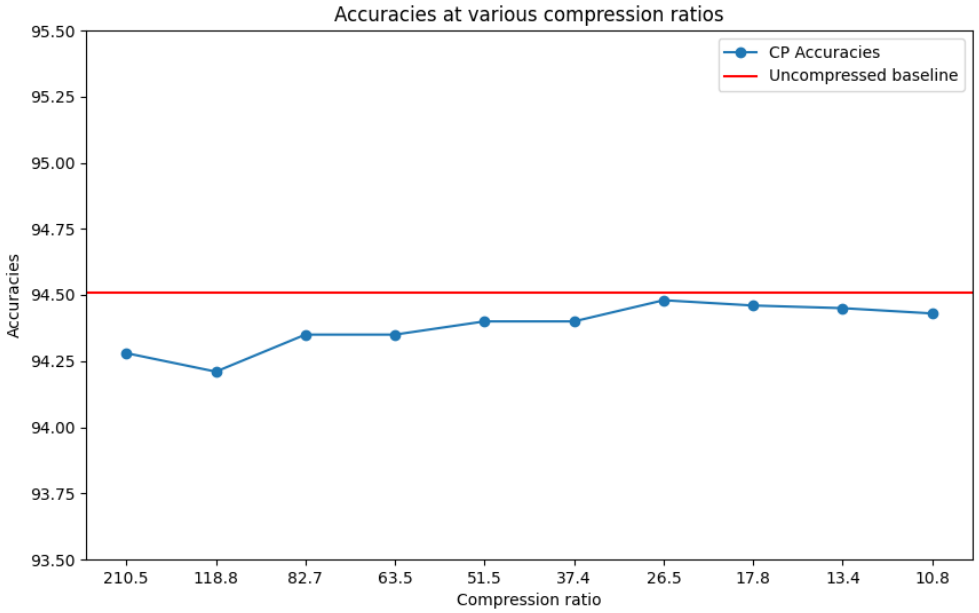


Figure 4-6: Triple run average accuracy of tensor decomposed autoencoder models at various compression ratio compared to the baseline uncompressed autoencoder model.

Using this ratio the number of FLOPs was calculated at various values of R, showing the expected quadratic increase of the number of FLOPs. It was found the number of FLOPs for the CP-decomposed backpropagation step is equal to the uncompressed model for a value of $R = 101$. This does however not account for the full increase in epoch duration. Upon inspection of the GPU's memory usage, it was found that for a CP-decomposed model with a rank value of $R = 100$, the process uses 14GB of RAM per GPU to 6GB of RAM for the uncompressed model, suggesting a significant increase in memory allocation operations.

Table 4-5: CP-rank, number of parameters, epoch duration and forward pass FLOPs for the uncompressed and compressed models.

Model	CP-rank	Parameters	Epoch duration (s)	GFLOPs
Uncompressed		$94.1 \cdot 10^5$	308	7.31
210.5×	5	$0.45 \cdot 10^5$	294	0.12
118.8×	10	$0.79 \cdot 10^5$	294	0.20
82.7×	15	$1.14 \cdot 10^5$	296	0.28
63.5×	20	$1.48 \cdot 10^5$	307	0.35
51.5×	25	$1.82 \cdot 10^5$	314	0.43
37.4×	35	$2.52 \cdot 10^5$	325	0.59
26.5×	50	$3.55 \cdot 10^5$	345	0.82
17.8×	75	$5.28 \cdot 10^5$	378	1.20
13.4×	100	$7.01 \cdot 10^5$	413	1.59
10.8×	125	$8.73 \cdot 10^5$	454	1.98

4-2-2 Experiment 2.2: Convergence speed comparison

The compression ratio chosen for this experiment was 13.4× as it provided an optimal balance between classification accuracy and compression ratio.

Figure 4-7 gives the validation loss average of three runs of the uncompressed and CP-decomposed model. For the first 8 epochs in the uncompressed model, the loss and its standard deviation steadily decline to a local minimum. The Adam optimizer bumps the objective function out of a local minimum to further decline until epoch 15 after which the rate of decline slows down. It has not reached a convergence point at epoch 15 as the validation loss has not settled yet and is still improving its parameters. The standard deviations are also relatively large. As the model converges it would be expected the standard deviation to become smaller.

Similarly, the CP-decomposed model has a steady decline in its validation loss, however it converges faster than the uncompressed model. After 15 epochs there is no more improvement in the validation loss. The standard deviation is smaller compared to the uncompressed model as well. The earlier convergence point for the CPD model could be explained by the reduced parameter space. This minimizes overfitting and speeds up the training process as there are fewer weights to update and optimize during each iteration. However, the model may not be complex enough to capture all relevant features for reconstruction. Experiment 2.1 showed that this does not necessarily lead to a reduction in diagnostic performance when choosing an appropriate compression ratio.

The hypothesis for this experiment was formulated as: CP-decomposed autoencoder networks will lead to faster convergence compared to uncompressed autoencoder networks. This was verified by these results. The faster converges is thought to originate from the reduced parameter space and regularization effect.

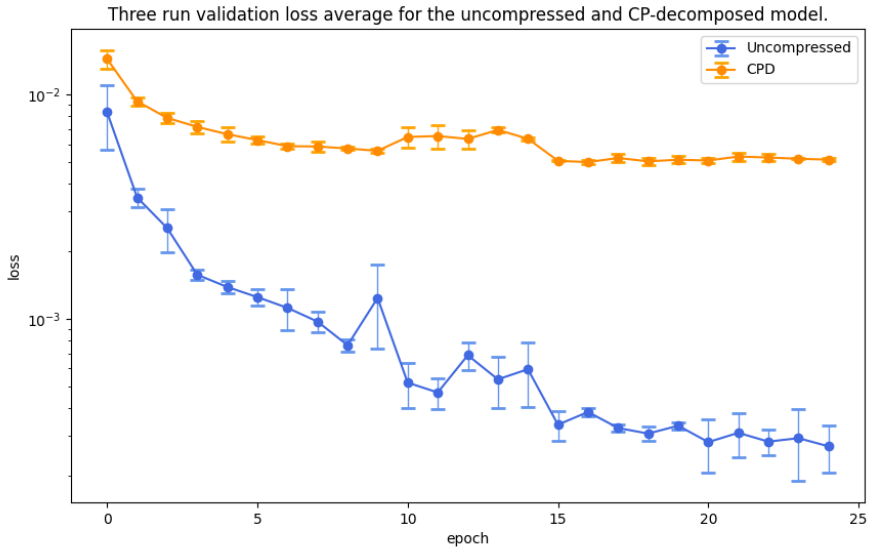


Figure 4-7: Three run average of the uncompressed autoencoder validation loss with error bars. The CP-decomposed model convergences after 15 epochs, whereas the uncompressed model fails to converge within the 25 epochs.

4-2-3 Experiment 2.3: Data efficiency comparison

In this experiment a compression ratio of 13.4× was selected, as it offered an optimal balance between compression ratio and predictive performance.

Figure 4-8 shows the averaged validation loss during training of both the uncompressed and CP-decomposed models for ratios of 2.5%, 5%, 10%, and 20% of the complete dataset size. The scale is logarithmic and equal for all subfigures. At a reduction ratio of 2.5%, the uncompressed model converges earlier than the complete dataset run. Compared to the CP-decomposed model, it converges with a larger gap to the complete dataset run as well. For a 5% reduction ratio both models show an improvement on the gap, however the CP-decomposed model approaches the complete dataset run better than the uncompressed model. This trend continues at 10% and 20% reduction ratios, where the CP-decomposed model shows better improvements in the performance gap.

Figure 4-9 shows the MSE of the test set at each reduction ratio for both the uncompressed model and the CP-decomposed model, along with dashed lines indicating the test set MSE for the complete dataset run in red for the uncompressed model and in green for the CP-decomposed model. At each reduction ratio the CP-decomposed model has a smaller performance gap to the full training set run than the uncompressed model.

It was hypothesized that the implementation of CPD would force a low-rank constraint on the model, which would lead to better generalization from fewer data samples. The results suggest that CP-decomposed models are more data-efficient, indicating better generalization for these models. The enhanced generalization could be the result of the regularization effect of CP decomposition. This experiment thus validates the hypothesis that CP-decomposed autoencoders achieve superior results compared to uncompressed autoencoders when training samples are limited.

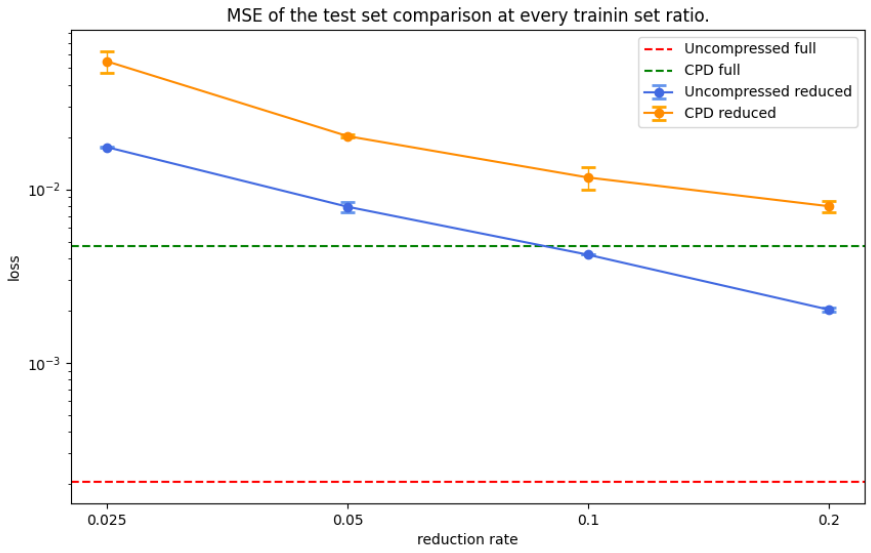


Figure 4-9: MSE of test set for both the uncompressed model and CP-decomposed model at reduction rates of 0.025, 0.05, 0.1 and 0.2. The full training set test set MSEs for both models are indicated by the dashed lines.

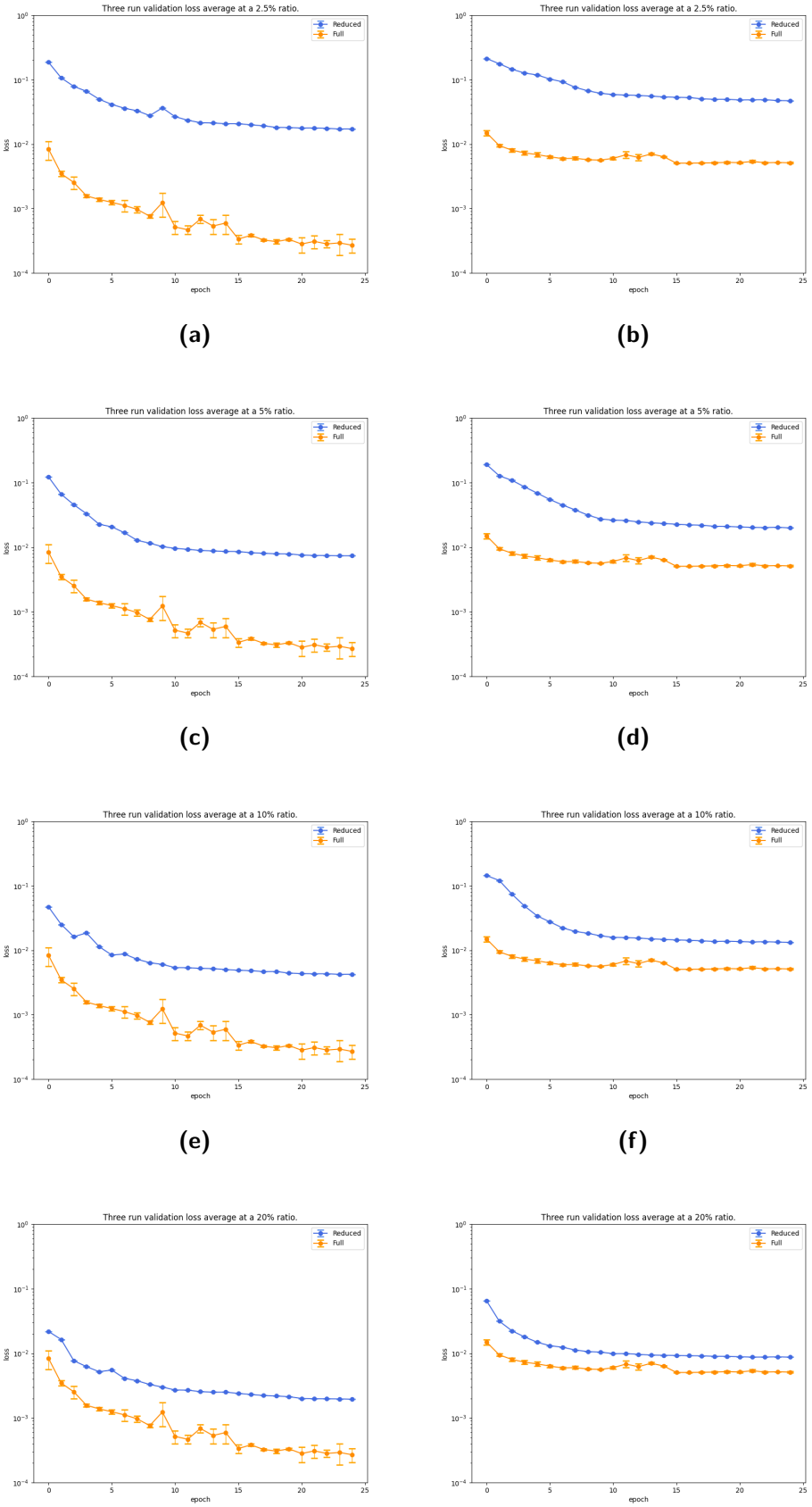


Figure 4-8: Validation loss during training for both the uncompressed (left) and CP-decomposed (right) models at various compression ratios of 0.025, 0.05, 0.1, and 0.2. The full training set validation loss is given in the orange plots.

Chapter 5

Conclusion

This thesis investigated whether transfer learning ECG classification models could benefit from the application of tensor decompositions. In particular, the Canonical Polyadic Decomposition (CPD) was applied to the convolutional layers of the autoencoder part of the model. The first part of the research focused on selecting the most optimal model to use for the ECG classification task, before implementing the tensor decompositions. For this purpose four different models were tested: a basic implementation of a convolutional autoencoder, an autoencoder adaption of both the ResNet and ConvNeXt architectures and a U-Net model. These model were evaluated based on their reconstruction ability, predictive accuracy and various performance metrics such as total number of FLOPs and epoch duration, after which a recommendation could be made.

It was found that the basic implementation of a convolutional autoencoder performed best overall on the evaluation metrics. The U-Net model had high reconstruction capabilities, however showed worse results on the predictive accuracy. The ResNet model had worse reconstruction and prediction capabilities, while having the largest parameters count. The ConvNeXt model failed to accurately reconstruct the images.

The basic model was subsequently used to address the main goal of this thesis:

Can the Canonical Polyadic Decomposition reduce the computational complexity of ECG classification models while maintaining diagnostic performance?

To answer this research question three hypotheses were formulated:

- CP-decomposed autoencoders intended for ECG classification can maintain a similar level of accuracy compared to uncompressed autoencoders.
- CP-decomposed autoencoder networks will lead to faster convergence compared to uncompressed autoencoder networks.
- Superior results can be achieved with CP-decomposed autoencoders when limiting the number of training samples compared to uncompressed autoencoder networks.

These hypotheses were tested in the second part of the thesis by conducting three separate sub-experiments.

To test the first hypothesis, decompositions were applied to the convolution operations within each layer. The decomposed models were then optimized and tested for various compression ratios to determine their effect on model accuracy and computational efficiency. The objective was to find an optimal balance between model complexity and performance, including reduced computational load, faster training times, and prevention of overfitting. This was particularly important for training on datasets with imbalanced class distributions.

Higher compression ratios made high frequencies harder to reconstruct due to limitations in the lower-dimensional vector representations, which reduce the ability to capture higher-dimensional functions. For lower compression ratios, the parameter space expands which allows for more efficient reconstructing. Accuracy initially dropped at high compression ratios due to limited feature extarction, but improved to the uncompressed baseline at lower ratios, indicating significant redundancy in the data. Tensor decompositions reduced this redundancy by limiting overfitting, introducing a form of regularization that increased accuracy. The number of FLOPs was significantly reduced for compressed models, although epoch duration unexpectedly increased. This was found to be due by the increase in computational expense of backpropagation through multiple smaller convolutions and an increase in GPU memory allocation.

Due to the increase in computational expense of the backpropagation step, the training process did not benefit from the implementation of the CPD in the autoencoder network. The total FLOPs of the forward pass did however improve significantly when compared to the uncompressed model. The implementation of tensor decomposition thus eases the use of such large classification models on embedded systems at inference time.

The primary objective of the second experiment was to test the hypothesis that CP-decomposed autoencoder networks will lead to faster convergence compared to uncompressed autoencoder networks. This was tested by comparing the average validation loss decline over three runs.

Results showed that the CP-decomposed model converged faster than the uncompressed model. The uncompressed model's validation loss failed to convergence within the 25 epochs. The CP-decomposed model however achieved convergence by 15 epochs. This faster convergence could be attributed to the reduction in parameters, resulting in a simpler parameter space to be explored and minimized overfitting.

The third experiment sought to test the hypothesis: Superior results can be achieved with CP-decomposed autoencoders when limiting the number of training samples compared to uncompressed autoencoder networks. The results of this experiment indicate that at a 2.5% reduction ratio, the uncompressed model converges faster but maintains a larger gap to the complete dataset run compared to the CP-decomposed model. This pattern of convergence improves with higher reduction ratios [5%, 10%, and 20%], where the CP-decomposed model closes the performance gap more effectively than the uncompressed model.

This pattern was also visible on the MSE of the test set, where the gap at each ratio was smaller for the CP-decomposed model than that of the uncompresssed model. This pattern is likely due to the CPD that imposes a low-rank constraint, improving the model's generalization ability with fewer data samples.

To answer the main research question of this thesis: the CPD can not reduce the computational complexity of ECG classification models while maintaining diagnostic performance. For lower compression ratios the accuracy approached the baseline, however the backpropagation process

of the CPD convolution increased the computational complexity. At higher compression ratios, the overall computational complexity was lower, but the classification accuracy was not on par with the uncompressed model. Using the CP-decomposed models it was however possible to outperform the accuracy of the paper of Wang et al. [15], which was used as a baseline metric, while maintaining a smaller storage complexity.

5-1 Recommendations for future research

Data

The data quality caused several issues throughout this thesis. First of all, the lack of data samples for some classes might have caused overfitting on particular classes. It can be expected of medical data to contain a high number of normal samples. For this dataset however, three of the four other classes made up only 2.1% of the samples in the labeled datasets. A more balanced data distribution would reduce overfitting and in turn better classification results.

For future work, using more leads should be explored to provide a more comprehensive view of the heart. This could help with classifying more complex arrhythmia. These leads could be presented on a single image, this will however make the reconstruction problem more difficult. The second option is to use each lead as an input channel to the model. This will increase kernel sizes by a factor 12^2, increasing the number of parameters significantly.

Tensor decomposition methods

As the Canoncial Polyadic Decomposition is only one method of tensor decomposition, other methods could provide improvements on its limitations. One method that could be implemented is the Tucker decomposition. The Tucker Decomposition gives more flexibility in choosing the ranks, as each dimension has its own rank value. The separate ranks could then be adjusted to allow for more higher dimension in for example the spatial dimensions as opposed to the channel dimensions.

Optimizer

There could be some issues with using the Adam optimizer in combination with a CP-decomposed network. The Adam optimizer takes into account both the mean of the gradient and its variance to adaptively update the learning rate parameter during training. When CPD is applied to a network, it reduces the number of parameters to introduce regularization and remove redundancies. However, the backpropagation process will have a smaller effect on larger network structures due to the abundance of parameters it can divide the updates over. Smaller networks have to process these updates with less parameters, resulting in higher variances. As the Adam optimizer punishes high variance parameter updates, it may fail to reach a global minimum.

Furthermore, the structure of the CP-decomposed convolution into four subsequent convolution might lead to too complex gradient interactions. The Adam optimizer may fail to recognize these gradient interactions properly and penalize them wrongfully.
Future work should explore different optimizers and test their optimization performance when applied to CP-decomposed networks.

Medical use

To further develop this research into something that can be employed in the medical setting, a few changes would have to be made. First of all, as the images would be taken of ECG prints, introducing significant noise to the input images. The performance of the model should be tested under these conditions. Furthermore, the diagnostic accuracy should be improved. With a top accuracy of 94.5%, the CP-decomposed models are not ready to be used in a clinical setting. This is largely due to the inability of the model to process inter-heartbeat relations and the above-mentioned lack of data samples for some classes.

Moreover, as only five general classes were taken into account for this thesis, the number of classified arrhythmia should be increased to improve the diagnostic help of such models.

Appendix A

Training Settings

A-1 Experiment 1

A-1-1 Autoencoder settings

config	value
optimizer	Adam
base learning rate	5e−5
weight decay	0.05
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
batch size	256
learning rate scheduler	StepLR
scheduler step size/gamma	15/0.1
training epochs	25

Table A-1: Basic model training settings

config	value
optimizer	Adam
base learning rate	1e−4
weight decay	0.05
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$
batch size	256
learning rate scheduler	StepLR
scheduler step size/gamma	15/0.1
training epochs	25

Table A-2: ResNet training settings

config	value
optimizer	Adam
base learning rate	1e−4
weight decay	0.05
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
batch size	256
learning rate scheduler	StepLR
scheduler step size/gamma	15/0.1
training epochs	25

Table A-3: ConvNeXT training settings

config	value
optimizer	Adam
base learning rate	1e−4
weight decay	0.05
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
batch size	256
learning rate scheduler	StepLR
scheduler step size/gamma	15/0.1
training epochs	25

Table A-4: U-Net training settings

A-1-2 Classifier settings

Table A-5: Classifier training settings

Config	Value
Optimizer	Adam
Base learning rate	1e−4
Weight decay	0.05
Batch size	256
Training epochs	10

A-2 Experiment 2

Table A-6: CP-decomposed autoencoder training settings

config	value
optimizer	Adam
base learning rate	8e−5
weight decay	0.05
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
batch size	256
learning rate scheduler	StepLR
scheduler step size/gamma	15/0.1
training epochs	25

Table A-7: Classifier training settings

Config	Value
Optimizer	Adam
Base learning rate	1e−4
Weight decay	0.05
Batch size	256
Training epochs	10

Appendix B

Backpropagation Complexity

This chapter derives the total number of FLOPs for the gradient calculation in the backpropagation process for both the uncompressed and CP-decomposed convolution. The convolution in this chapter remains its spatial dimensions, so $H_{in} = H_{out}$ and $W_{in} = W_{out}$.

B-1 Convolution

In order to optimize the weight parameters during training, the gradients of these weights with respect to the loss function have to be calculated. The forward pass equation of a single convolution can be given in tensor format by,

$$\underline{\boldsymbol{Y}} = \underline{\boldsymbol{K}} * \underline{\boldsymbol{X}} \tag{B-1}$$

with output $\underline{\boldsymbol{Y}}$, input $\underline{\boldsymbol{X}}$ and weights $\underline{\boldsymbol{K}}$.

The gradient of the weight parameters with respect to the loss function can then be obtained using the chain rule[11],

$$\frac{\partial L}{\partial \underline{\boldsymbol{K}}} = \frac{\partial L}{\partial \underline{\boldsymbol{Y}}} \cdot \frac{\partial \underline{\boldsymbol{Y}}}{\partial \underline{\boldsymbol{K}}}, \tag{B-2}$$

and with respect to the inputs with,

$$\frac{\partial L}{\partial \underline{\boldsymbol{X}}} = \frac{\partial L}{\partial \underline{\boldsymbol{Y}}} \cdot \frac{\partial \underline{\boldsymbol{Y}}}{\partial \underline{\boldsymbol{X}}}. \tag{B-3}$$

When calculating convolutions, tensors are often transformed to matrices using the *unfold* operation, which allows the convolution calculations to be obtained with a single matrix multiplication.[17] These conversion will be denoted by $\mathbf{X}_{col}$, $\mathbf{Y}_{col}$ and $\mathbf{K}_{row}$. Figure B-1 gives an example of the unfolding function of $\mathbf{X}$ within a convolution operation. After the unfolding the convolution operation in equation B-1 is simplified to a matrix multiplication,

$$\mathbf{Y}_{col} = \mathbf{K}_{row} \cdot \mathbf{X}_{col} \tag{B-4}$$

The backpropagation step for a single convolution starts by unfolding tensor into patches of dimension $S \cdot D \cdot D$ for its rows and repeated for each kernel multiplication in its columns with

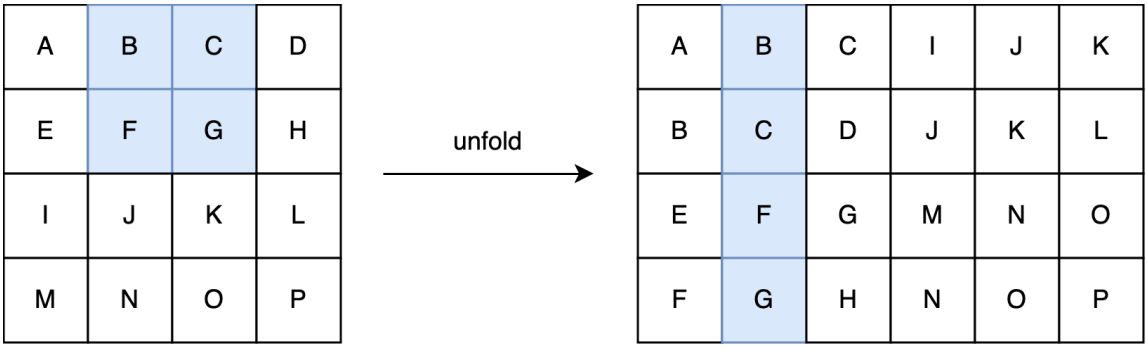


Figure B-1: Unfolding process the of $\mathbf{X}$ in a convolution operation with a 2×2 kernel. The blue shaded elements indicate the unfolding of that specific step.

dimension $H \cdot W$, resulting in a dimension of $(S \cdot D \cdot D \times H \cdot W)$. The dimension of the kernel becomes $(T \times S \cdot D \cdot D)$ and for the output, $(C_{out} \times H \cdot W)$.

After unfolding the two gradient equations become,

$$\frac{\partial L}{\partial \mathbf{K}_{\text{row}}} = \frac{\partial L}{\partial \mathbf{Y}_{col}} \cdot \frac{\partial \mathbf{Y}_{col}}{\partial \mathbf{K}_{\text{row}}}, \tag{B-5}$$

and,

$$\frac{\partial L}{\partial \mathbf{X}_{col}} = \frac{\partial L}{\partial \mathbf{Y}_{col}} \cdot \frac{\partial \mathbf{Y}_{col}}{\partial \mathbf{X}_{col}}. \tag{B-6}$$

To simplify these expressions, the derivation in Deisenroth et al. [53] is used to obtain,

$$\frac{\partial \mathbf{Y}_{col}}{\partial \mathbf{K}_{\text{row}}} = \mathbf{X}_{col}^T, \tag{B-7}$$

and

$$\frac{\partial \mathbf{Y}_{col}}{\partial \mathbf{X}_{col}} = \mathbf{K}_{\text{row}}^T \, . \tag{B-8}$$

The gradient expressions are then updated as,

$$\frac{\partial L}{\partial \mathbf{K}_{row}} = \frac{\partial L}{\partial \mathbf{Y}_{col}} \cdot \mathbf{X}_{col}^T, \tag{B-9}$$

and

$$\frac{\partial L}{\partial \mathbf{X}_{col}} = \mathbf{K}_{row}^T \cdot \frac{\partial L}{\partial \mathbf{Y}_{col}}. \tag{B-10}$$

Table B-1 gives the dimension of $\mathbf{X}_{col}$, $\mathbf{K}_{row}$ and $\mathbf{Y}_{col}$.
The goal here is to calculate the number of FLOPs for the backpropagation process for a single convolution. The FLOPs for the multiplication of two matrices with dimension $(m \times p)$ and $(n \times p)$ is calculated as [54],

$$\text{FLOPs} = 2mnp. \tag{B-11}$$

The FLOPs calculation for equation B-9 is then given by,

$$\text{FLOPs} = 2HWTSD^2, \tag{B-12}$$

and for equation B-10 by,

$$\text{FLOPs} = 2HWTSD^2. \tag{B-13}$$

Note that these expression are equal. The *unfold* operation does not contribute to the total number of FLOPs as it is a relocation of memory.

Table B-1: Dimension for the unfolded matrices.

Matrix	Dimension
$\mathbf{X}_{col}$	$S \cdot D \cdot D \times H \cdot W$
$\mathbf{K}_{row}$	$T \times S \cdot D \cdot D$
$\mathbf{Y}_{col}$	$T \times H \cdot W$

B-2 CP-decomposed convolution

The CP-decomposed convolution is separated into four subsequent smaller convolution. First the input dimension S is mapped to R via a pointwise convolution, then two depthwise convolutions for the horizontal and vertical spatial dimension, and finally the R dimension is mapped back to T.[38] The tensors are unfolded to matrices in the same manner as the previous section. To calculate the gradient of this process the following set of equations can be followed,

$$\frac{\partial L}{\partial \mathbf{K}_{rt}} = \left(\frac{\partial L}{\partial \mathbf{Y}}\right) \cdot \mathbf{X}_h^T, \tag{B-14}$$

$$\frac{\partial L}{\partial \mathbf{K}_{h}} = \left(\frac{\partial L}{\partial \mathbf{X}_h}\right) \cdot \mathbf{X}_w^T, \tag{B-15}$$

$$\frac{\partial L}{\partial \mathbf{K}_{w}} = \left(\frac{\partial L}{\partial \mathbf{X}_w}\right) \cdot \mathbf{X}_{sr}^T, \tag{B-16}$$

$$\frac{\partial L}{\partial \mathbf{K}_{sr}} = \left(\frac{\partial L}{\partial \mathbf{X}_{sr}}\right) \cdot \mathbf{X}^T. \tag{B-17}$$

As the gradients are calcualted from the last weights to the first, the order of calculation is reversed.

For the intermediate layers, the output of the convolution is the input for the next. The gradients of the loss function with respect to the intermediate input matrices are obtained with,

$$\frac{\partial L}{\partial \mathbf{X}_{h}} = \left(\frac{\partial L}{\partial \mathbf{Y}}\right) \cdot \mathbf{K}_{rt}^T, \tag{B-18}$$

$$\frac{\partial L}{\partial \mathbf{X}_{w}} = \left(\frac{\partial L}{\partial \mathbf{X}_h}\right) \cdot \mathbf{K}_{h}^T, \tag{B-19}$$

$$\frac{\partial L}{\partial \mathbf{X}_{sr}} = \left(\frac{\partial L}{\partial \mathbf{X}_w}\right) \cdot \mathbf{K}_{w}^T. \tag{B-20}$$

The dimensions of the S to R convolution step is given in Table B-2, for the depthwise convolutions in Table B-3 and for the R to T convolution step in Table B-4.

Table B-2: Matrix dimension for the R to T convolution step in the CP-decoposed convolution operation.

Matrix	Dimension
$\mathbf{X}_{h,col}$	$R \times H \cdot W$
$\mathbf{K}_{rt,row}$	$T \times R$
$\mathbf{Y}_{col}$	$T \times H \cdot W$

Table B-3: Matrix dimension for the depthwise convolution steps in the CP-decoposed convolution operation.

Matrix	Dimension
$\mathbf{Y}_{w,col}/\mathbf{Y}_{sr,col}$	$R \times H \cdot W$
$\mathbf{K}_{h,row}/\mathbf{K}_{w,row}$	$R \times R \cdot D$
$\mathbf{Y}_{h,col}/\mathbf{Y}_{w,col}$	$R \times H \cdot W$

Table B-4: Matrix dimension for the S to R convolution step in the CP-decoposed convolution operation.

Matrix	Dimension
$\mathbf{X}_{col}$	$S \times H \cdot W$
$\mathbf{K}_{sr,row}$	$R \times S$
$\mathbf{Y}_{sr,col}$	$R \times H \cdot W$

In the previous section it was found that the number of FLOPs for $\frac{\partial L}{\partial \mathbf{K}_{\text{row}}}$ was equal to the number for $\frac{\partial L}{\partial \mathbf{X}_{\text{col}}}$. For ease of notation, the number of FLOPs is only written down for the weights and then doubled to arrive at the total number of FLOPs for the backward pass, which also works in this case. Using equation B-11 the number of FLOPs for each step in the decomposed convolution is calculated for the S to R step as,

$$\text{FLOPs}_{sr} = 4RSHW, \tag{B-21}$$

for the horizontal depthwise convolution as,

$$\text{FLOPs}_{w} = 4DHWR^2, \tag{B-22}$$

for the vertical depthwise convolution as,

$$\text{FLOPs}_{h} = 4DHWR^2, \tag{B-23}$$

and finally for the mapping of R to T as,

$$\text{FLOPs}_{rt} = 4RTHW. \tag{B-24}$$

Bibliography

[1] Neil Herring and David J. Paterson. *Levick's Introduction to Cardiovascular Physiology, Sixth Edition*. ISBN 9780815363613.

[2] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51:455–500, 2009. ISSN 00361445. doi: 10.1137/07070111X.

[3] A. Cichocki, N. Lee, I. V. Oseledets, A. H. Phan, Q. Zhao, and D. Mandic. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1. 9 2016. doi: 10.1561/2200000059. URL http://arxiv.org/abs/1609.00893http://dx.doi.org/10.1561/2200000059.

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. 3 2020. URL http://arxiv.org/abs/2003.05991.

[5] Wenji Zuo. Deep learning for enhancing cancer identification and diagnosis. *Applied and Computational Engineering*, 2023. doi: 10.54254/2755-2721/27/20230093.

[6] J. Viswanathan, N. Saranya, and Abinaya Inbamani. Deep learning applications in medical imaging. *Deep Learning Applications in Medical Imaging*, 2021. doi: 10.4018/978-1-7998-5071-7.ch007.

[7] Eduardo José da S. Luz, William Robson Schwartz, Guillermo Cámara Chávez, and David Menotti. Ecg-based heartbeat classification for arrhythmia detection: A survey. *Computer methods and programs in biomedicine*, 127:144–164, 2016. doi: 10.1016/j.cmpb.2015.12.008.

[8] R. H. Hongo and N. Goldschlager. Overreliance on computerized algorithms to interpret electrocardiograms. *The American Journal of Medicine*, 117(9):706–708, 2004. doi: 10.1016/j.amjmed.2004.08.006.

[9] Md. Moklesur Rahman, M. Rivolta, F. Badilini, and R. Sassi. A systematic survey of data augmentation of ecg signals for ai applications. *Sensors (Basel, Switzerland)*, 23, 2023. doi: 10.3390/s23115237.

[10] Jing Zhang, Aiping Liu, Deng Liang, Xun Chen, and Min Gao. Interpatient ecg heartbeat classification with an adversarial convolutional neural network. *Journal of Healthcare Engineering*, 2021, 2021. doi: 10.1155/2021/9946596.

[11] Ovidiu Calin. *Deep Learning Architectures: A Mathematical Approach*. Springer Series in the Data Sciences. Springer, Cham, Switzerland, 1 edition, 2020. ISBN 978-3-030-36721-3. doi: 10.1007/978-3-030-36721-3. URL https://link.springer.com/book/10.1007/978-3-030-36721-3.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2015. URL https://arxiv.org/abs/1409.1556.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2016. URL https://arxiv.org/abs/1512.03385.

[14] Zhuang Liu, Hanzi Mao, Chaozheng Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11966–11976. IEEE, 2022. URL https://arxiv.org/abs/2201.03545.

[15] Guoxin Wang, Qingyuan Wang, Ganesh Neelakanta Iyer, Avishek Nag, and Deepu John. Unsupervised pre-training using masked autoencoders for ecg analysis. 10 2023. URL http://arxiv.org/abs/2310.11153.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martín Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://pytorch.org/.

[17] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20:1–6, 2019. URL https://tensorly.org/.

[18] S. Serge Barold. Willem einthoven and the birth of clinical electrocardiography a hundred years ago. *Cardiac Electrophysiology Review*, 7(1):99–104, 2003. URL https://doi.org/10.1023/A:1023667812925.

[19] Tomas B. Garcia. *12-Lead ECG: The Art of Interpretation*. ISBN 9780763773519.

[20] ECGpedia. Atrial premature complexes — ecgpedia,, 2013. URL https://en.ecgpedia.org/index.php?title=Atrial_Premature_Complexes&oldid=16671. [Online; accessed 28-May-2024].

[21] Andrzej Cichocki and Lieven De Lathauwer. Tensor decompositions for signal processing applications. 2015.

[22] Pablo Villalobos, J. Sevilla, T. Besiroglu, Lennart Heim, A. Ho, and Marius Hobbhahn. Machine learning model sizes and the parameter gap. *ArXiv*, abs/2207.02852, 2022. doi: 10.48550/arXiv.2207.02852.

[23] Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927. doi: 10.1002/sapm192761164.

[24] Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966. doi: 10.1007/BF02289464.

[25] Richard A. Harshman. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. Technical report, UCLA Working Papers in Phonetics, 1970. Available from the author at University of Western Ontario.

[26] J. Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35 (3):283–319, 1970. doi: 10.1007/BF02310791.

[27] N.D. Sidiropoulos and Rasmus Bro. On the uniqueness of multilinear decomposition of n-way arrays. *Journal of Chemometrics - J CHEMOMETR*, 14:229–239, 05 2000. doi: 10.1002/1099-128X(200005/06)14:33.0.CO;2-N.

[28] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[29] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2020. URL https://ar5iv.org/abs/2003.05991.

[30] Carl Doersch. Tutorial on variational autoencoders. *ArXiv*, abs/1606.05908, 2016. doi: 10.48550/arXiv.1606.05908.

[31] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[32] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[33] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, October 1969. doi: 10.1109/TSSC.1969.300225.

[34] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015. URL https://arxiv.org/abs/1502.03167.

[35] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *ArXiv*, abs/1606.08415, 2016. URL https://arxiv.org/abs/1606.08415.

[36] Benjamin Graham. Fractional max-pooling. *ArXiv*, abs/1412.6071, 2014.

[37] V. Lebedev, Yaroslav Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *CoRR*, abs/1412.6553, 2014.

[38] Marcella Astrid and Seung-Ik Lee. Cp-decomposition with tensor power method for convolutional neural networks compression. 1 2017. URL http://arxiv.org/abs/1701.07148.

[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

[40] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. C. Ivanov, R. Mark, ..., and H. E. Stanley. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation [Online]*, 101(23):e215–e220, 2000. URL https://doi.org/10.1161/01.CIR.101.23.e215.

[41] Erick A. Perez Alday, Annie Gu, Amit J. Shah, Chad Robichaux, An-Kwok Ian Wong, Chengyu Liu, Feifei Liu, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, Qiao Li, Ashish Sharma, Gari D. Clifford, and Matthew A. Reyna. Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020. *Physiological Measurement*, 41(12): 124003, 2020. doi: 10.1088/1361-6579/abc960.

[42] Patrick Wagner, Nils Strodthoff, R. Bousseljot, Wojciech Samek, and Tobias Schaeffter. Ptb-xl, a large publicly available electrocardiography dataset (version 1.0.3), 2022. URL https://doi.org/10.13026/kfzx-aw45.

[43] X. Wang, C. Ma, X. Zhang, H. Gao, G. D. Clifford, and C. Liu. Paroxysmal atrial fibrillation events detection from dynamic ecg recordings: The 4th china physiological signal challenge 2021 (version 1.0.0), 2021. URL https://doi.org/10.13026/ksya-qw89. PhysioNet.

[44] George B. Moody and Roger G. Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, May-June 2001. doi: 10.1109/51.932724.

[45] Philip de Chazal, Maria O'Dwyer, and Richard B. Reilly. Automatic classification of heartbeats using ecg morphology and heartbeat interval features. *IEEE Transactions on Biomedical Engineering*, 51(7):1196–1206, 2004. doi: 10.1109/TBME.2004.827359.

[46] Erick A. Perez Alday, Annie Gu, Amit Shah, Chad Robichaux, An-Kwok Ian Wong, Chengyu Liu, Feifei Liu, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, Qiao Li, Ashish Sharma, Gari D. Clifford, and Matthew A. Reyna. Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020. *medRxiv*, 2020. doi: 10.1101/2020.08.11.20172601. URL https://doi.org/10.1101/2020.08.11.20172601.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 12 2015. URL http://arxiv.org/abs/1512.03385.

[48] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2021. URL https://arxiv.org/abs/2010.11929.

[49] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, 2015. doi: 10.1007/978-3-319-24574-4_28.

[50] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004. doi: 10.1145/1015330.1015435.

[51] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[52] PyTorch. *Automatic Mixed Precision*, 2023. URL https://pytorch.org/docs/stable/amp.html. Accessed: 2024-05-24.

[53] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, Cambridge, 2020. URL https://course.ccs.neu.edu/ds4420sp20/readings/mml-book.pdf.

[54] Ryan Tibshirani. Convex optimization: Fall 2019, lecture 19, 2019. URL http://www.stat.cmu.edu/~ryantibs/convexopt-F19/lectures.html. Course Lecture, Machine Learning 10-725, Carnegie Mellon University.

Glossary

List of Acronyms

N Normal beat

S Supraventricular Ectopic Beat

V Ventricular Ectopic Beat

F Fusion beat

Q Unknown beat

FLOPs Floating point operations

ECG electrocardiography

List of Symbols

$*$ Convolution

$\boldsymbol{v}$ Vector

$\circ$ Outer product

$\mathbf{M}$ Matrix

$\mathbf{X}_{col}$ Unfolded column matrix

$\mathbf{X}_{row}$ Unfolded row matrix

$\underline{\boldsymbol{X}}$ Tensor

a, A Scalar