ECG Interpretation with Machine Learning

Submitted by: Diya Patel
Student ID: 20115612

Supervisor: Andrew Lowe

School of Engineering, Computer and Mathematical Science
A final year project report presented to the Auckland University of Technology in partial fulfilment of the requirements of the degree of Bachelor of Engineering
2023

Acknowledgements

I want to thank my supervisor, Professor Andrew Lowe, for the constant guidance, encouragement, and support that you have provided me throughout this project.

I want to acknowledge my friends for the help and encouragement you provided me while working on this project.

Lastly, I would like to thank my family for supporting me through the process of this project and my degree.

Statement of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Diya Patel

......01/08/2024...... ....

Date Diya Patel

Abstract

Electrocardiograms (ECGs) measure the electrical activity of the heart, presenting a distinct pattern of waves and pulses analyzed by medical experts to assess cardiac health. Recent advancements in healthcare, driven by technological innovation, have enabled the automation and enhancement of ECG monitoring, analysis, and interpretation. To contribute to this progress, extensive literature reviews were conducted to understand state-of-the-art techniques in ECG analysis. This research covered various machine learning and deep learning approaches for automating ECG signal interpretation and anomaly detection. Building on these insights, a Convolutional Neural Network (CNN) model was developed to gain further understanding. This work highlights the potential of deep learning to revolutionize ECG analysis, assisting medical professionals in making more accurate and timely diagnoses, thereby improving patient outcomes and easing the burden on healthcare systems.

Keywords: Electrocardiograms (ECGs), Convolutional Neural Network (CNN), cardiac health, deep learning, ECG signal analysis

Acronyms

Afib	Atrial Fibrillation
ECG	Electrocardiogram
CNN	Convolution Neural Network
SAN	Sinoatrial Node
AVN	Atrioventricular Node

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Introduction

Electrocardiogram (ECG) interpretation holds significant importance in the field of cardiac healthcare. It serves as a fundamental diagnostic tool used by medical professionals to assess the electrical activity of the heart. The patterns and intervals observed in ECG waveforms provide valuable insights into various cardiac conditions, ranging from arrhythmias to myocardial infarction. However, the process of manual ECG interpretation is not without its challenges, including interobserver variability, human error, and the need for specialized expertise.

In recent years, the intersection of healthcare and technology has led to remarkable advancements, particularly in the realm of machine learning. By leveraging algorithms capable of learning from data, machine learning techniques have the potential to revolutionize ECG analysis and interpretation. This fusion of medical expertise with computational intelligence promises to automate and enhance the diagnostic process, thereby improving patient outcomes and streamlining healthcare delivery.

This project aims to explore the integration of machine learning with ECG interpretation, offering a comprehensive investigation into examining the feasibility and efficacy of such an approach. By delving into the underlying theory, reviewing existing literature, and potentially developing novel algorithms, this research endeavours to contribute to the ongoing evolution of cardiac healthcare. This exploration is compelling as it bridges the gap between traditional medical practices and cutting-edge technological innovations, providing insights into the future of ECG analysis and its impact on patient care.

1.2 Scope and Limitations

This project explores various aspects of ECG monitoring, analysis, and interpretation. However, not all these aspects are directly connected. To keep the project focused, it will concentrate on the following three specific aims:

1. Feature Extraction: The project will commence with a focus on feature extraction from ECG signals. This process aims to identify and extract relevant features crucial for subsequent classification tasks. Emphasis will be placed on discerning clinically significant features to aid in ECG interpretation.

2. Denoising Techniques: Recognizing the significance of data quality, the project will explore techniques for denoising ECG signals. These methods will aim to effectively remove noise artifacts while preserving the integrity of the underlying ECG waveform. Improved data quality ensures more accurate analysis and classification outcomes.

3. Transformation of ECG Time Series into Image Representations: An additional essential aspect involves transforming ECG time series data into image representations before feeding into machine learning models. This step enhances the algorithm's ability to discern patterns and features within the data, potentially improving classification accuracy and interpretability. Integration of techniques for this transformation should be considered to optimize classification performance and enhance clinical utility.

4. Machine Learning Algorithm Implementation: CNN machine learning algorithms will be implemented using Python. These algorithms play a pivotal role in achieving robust ECG classification. The selection and fine-tuning of these algorithms will be crucial for optimizing classification performance. Algorithms will be carefully selected and fine-tuned to maximize classification accuracy and effectiveness.

Although the field of "ECG Interpretation" is vast, it is unrealistic to cover all the ground necessary within the scope of this project. The project is set to cover the most basic concepts and provide a bird's eye-view of what can be studied and researched. In-depth exploration of specialized topics such as real-time monitoring, multi-class classification, or domain adaptation may fall beyond the scope of this project. Additionally, while efforts will be made to ensure accurate feature extraction and denoising, the efficacy of machine learning algorithms may be influenced by factors such as data quality, sample size, and model complexity, which may present limitations to the classification performance.

Chapter 2 Theory

2.1.1 The Heart and Conductive System

2.1.1.1 Introduction to ECG (Electrocardiography):

[1] Electrocardiography (ECG) is a non-invasive diagnostic tool used to assess the electrical activity of the heart. It involves recording the electrical impulses generated by the heart as it contracts and relaxes. ECG is crucial in clinical practice for diagnosing various cardiac abnormalities, including arrhythmias, ischemia, myocardial infarction, and conduction disorders. By analysing the patterns and intervals of the ECG waveform, healthcare professionals can gain valuable insights into the heart's health and function.

2.1.1.2 Cardiac Conduction

[2] The cardiac conduction system is a network of specialized cells that coordinates the electrical impulses responsible for regulating the heartbeat. It ensures the synchronized contraction and relaxation of the heart chambers, facilitating the efficient pumping of blood throughout the body. The key components of the cardiac conduction system include:

1. The Sinoatrial Node (SAN), located in the right atrium, functions as the heart's natural pacemaker by spontaneously generating electrical impulses to initiate atrial contraction.
2. Positioned in the atrioventricular septum, the Atrioventricular Node (AVN) receives impulses from the atria and transmits them to the ventricles, facilitating coordinated contraction.
3. The Bundle of His, comprised of specialized muscle cells, conducts impulses from the AVN to the ventricles through bundle branches, ensuring efficient transmission of electrical signals.
4. Purkinje Fibers, found as subendocardial fibres in the ventricles, rapidly transmit electrical signals, leading to synchronized ventricular contraction and effective blood pumping.

2.1.1.3 ECG Waves

[3] In terms of clinical significance and diagnostic value, as shown in figure 1, the most important waves in an electrocardiogram (ECG) are typically considered to be:

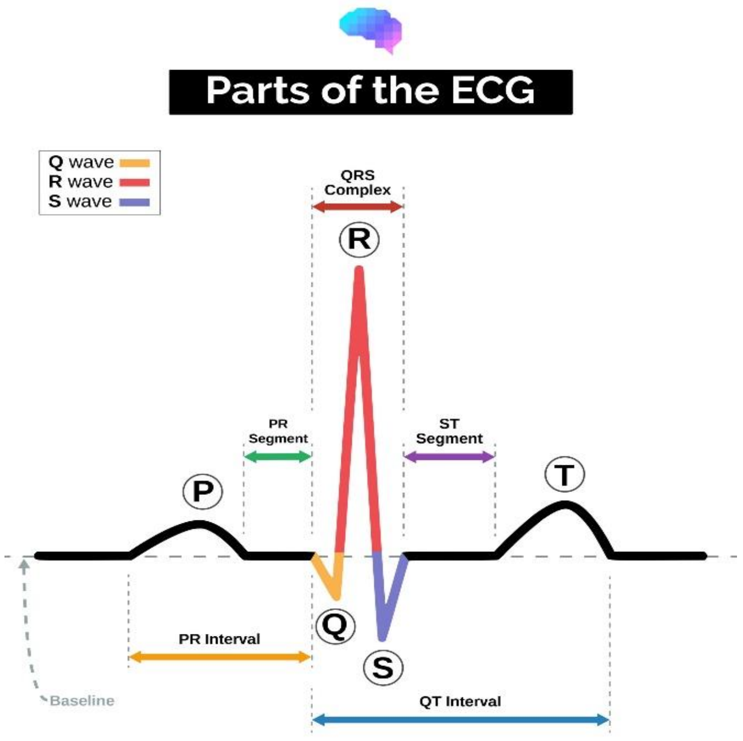


Figure 1 Parts of the ECG

1. The P wave, the first waveform in an ECG, depicts the initial electrical pulse from the SA node propagating across the atria to depolarize myocardial cells for contraction and blood influx. Although smaller in amplitude compared to the QRS complex and T wave, abnormalities in the P wave can indicate atrial enlargement, atrial fibrillation, or other atrial arrhythmias. The end of the P wave marks the point where the pulse traverses through the Atrioventricular Node (AV Node), signifying the beginning of the diastole phase.

2. QRS Complex: The QRS complex represents ventricular depolarization, initiating ventricular contraction and serving as a vital indicator for diagnosing abnormalities in ventricular conduction, such as bundle branch blocks, ventricular hypertrophy, and myocardial infarction (heart attack). This waveform starts after the PR segment, signifying the departure of the pulse from the AVN and the commencement of ventricular depolarization through the His-Purkinje system. With greater magnitude on ECGs due to the larger size of the ventricles compared to the atria, the QRS complex marks the systole phase, essential for the effective pumping of blood out of the heart.

3. The ST segment and T wave together signify the repolarization phase of both the ventricles and atria in the cardiac cycle. The ST segment is crucial for identifying myocardial ischemia or injury, with elevation or depression indicating conditions like myocardial infarction, ischemia, or pericarditis. Similarly, the T wave represents ventricular repolarization, and changes in its morphology or amplitude can signal electrolyte imbalances, myocardial ischemia, infarction, or other cardiac abnormalities. Together, these components provide critical information for diagnosing and managing various cardiac conditions.

2.1.1.4 Arrhythmia

The combination of all these waves makes up one heartbeat and repeats in a consistent fashion to make what is called Normal Sinus Rhythm. Arrhythmias are heartbeat rhythms that are different to the normal sinus rhythm we expect to see, as shown in figure 2. There are several different arrhythmias categorized by their cause, place of origin and phenomena, not all of which are medically concerning.

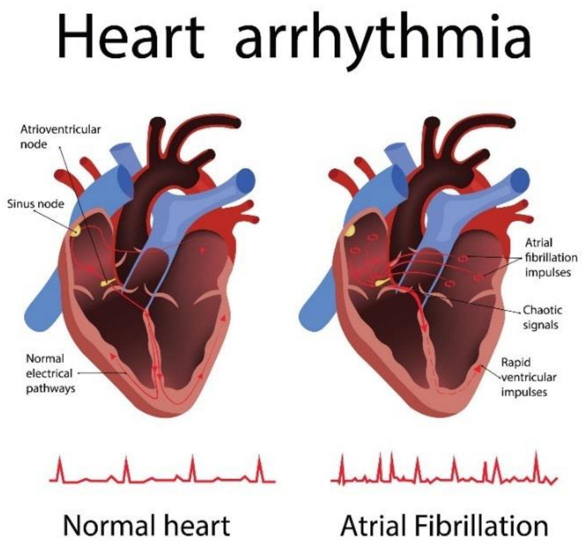


Figure 2 Arrhythmia

Places of Origin:

- SA Node
- Atria
- AV Junction
- Ventricles

Phenomena:

- Tachycardia (>100 bpm)
- Bradycardia (<60 bpm)
- Premature Contractions
- Flutter
- Fibrillation

Atrial Fibrillation (Afib):

- Common type of arrhythmia characterized by asynchronous and erratic atrial contractions.
- Often caused by reentry mechanism.
- ECG shows undiscernible P waves (F-waves) and irregular R-R intervals.
- Associated with risks such as blood clots leading to strokes, ischemic heart disease, hypertension, pulmonary disease, and rheumatic heart disease.
- Symptoms include heart palpitations, shortness of breath, weakness, but can also be asymptomatic.
- Early detection is crucial due to associated risks.

2.1.2 ECG Instrumentation

2.1.2.1 ECG instruments

Electrodes: Contacts made between the skin and the electrical readout circuit.
Wet Electrodes: Wet electrodes are the most common kind of electrodes used in hospitals. These gel patches are applied to the skin and contain a tiny metal contact covered with electrolytic paste. To achieve galvanic coupling, this forms an electrolyte-electrode contact that transforms ionic current on the skin's surface into electrical current. After that, the patches are connected to conductive copper wires leading to the rest of the circuit. Its key benefits are its low electrode impedance and easier disposal and use for hygienic and financial reasons.
Dry Electrodes: Instead of using a gel, dry electrodes rely on capacitive coupling to isolate the metal contact from the skin. This is achieved by coating the metal contact with a thin layer of insulating oxide or a film with a high dielectric constant. Since there is no need for skin preparation, this is more practical for a personal biopotential collecting device that is portable.

Leads: The location of the electrodes on the body gives different signals.
12-Leads: From 10 electrodes, we can derive 12 signals characterizing the heart's activity. On the left arm, left leg, and right arm, three electrodes are positioned. Lead I, II, and III are the three bipolar leads, and the three unipolar leads (lead aVR, aVL, and aVF) that result from this create six limb leads. The remaining six leads (V1–V6) are unipolar and come from six electrodes that are positioned on the chest.
Single-Lead: Two electrodes, one on the left and one on the right arm, can be used to generate a single lead, lead I in a 12-lead configuration. For convenience, handheld ECG monitors frequently employ this configuration.

Amplifier: Used to amplify the biopotential signal from the electrodes.
Instrumentation Amplifiers: The most used type of amplifier for testing and measurement is an instrumentation amplifier. The gain, which is typically controlled by a single external resistor, is the amount by which the signal is amplified. High gain has the drawback of magnifying even noise, but it can provide us a larger picture to work with. The Common-Mode Rejection Ratio, which reduces Power Line Interference, is another crucial feature.

ADC: Digitizes the analogue signal from the amplifier for processing.
Microcontrollers: Due to their on-board ADC and transmitter for transmitting the digital signal to a computer, where software may be used to process and graphically display the ECG signal, microcontrollers/processors are frequently utilized. The quality of the conversion and the Voltage Reference are determined by the resolution of an ADC. To achieve maximum accuracy and detail in the digitized signal, higher resolution at lower voltage reference (as near to the amplifier's range) is used.

2.1.2.2 Challenges in Manual ECG Interpretation

Manual ECG interpretation has several drawbacks and difficulties, such as:

- Interobserver variability: Different medical professionals may interpret an identical ECG in various ways, which could result in conflicting diagnoses.
- Human error: Exhaustion, distraction, or lack of experience might lead to misinterpretation or overlooking tiny irregularities.
- Requirement for specialized knowledge: Not all healthcare facilities may have access to the required training and experience needed for accurate ECG interpretation.

2.1.3 Machine Learning

2.1.3.1 Introduction to Machine Learning:

Within the field of artificial intelligence, machine learning allows computers to learn from data and make judgments or predictions without the need for explicit programming. It covers a range of methods, such as reinforcement learning (where models learn by feedback and trial and error), supervised learning (where models learn from labelled data), and unsupervised learning (where models find patterns in unlabelled data). Applications for machine learning are numerous and span a variety of industries, including banking, healthcare, and natural language processing in addition to picture recognition.

2.1.3.2 Machine Learning in Healthcare:

Machine learning has tremendous potential to improve patient outcomes, diagnosis accuracy, and healthcare delivery. Machine learning algorithms can detect patterns, forecast the likelihood of a disease, and help clinicians make evidence-based decisions by analysing vast amounts of medical data, including genetic data, imaging investigations, and patient records. Machine learning has several uses in the medical field, including personalized medicine, disease detection, therapy optimization, and health monitoring.

2.1.3.3 Deep Learning with Neural Networks

Deep learning, a subset of machine learning, has emerged as a powerful tool in ECG analysis, owing to its capacity to autonomously discern intricate patterns and features from raw data. At the heart of deep learning lie neural networks, computational constructs inspired by the structure and functionality of the human brain. Artificial neurons are arranged in these networks in interconnected layers, with each layer processing and transforming incoming data to produce useful outputs. CNNs, or convolutional neural networks, have proven to be incredibly effective in analysing spatial patterns found in ECG signals. Convolutional layers enable CNNs to automatically extract relevant features from unprocessed ECG data, enabling accurate classification of arrhythmias and other cardiac abnormalities. In a similar vein, Recurrent Neural Networks (RNNs) are especially well-suited for applications like ECG signal processing because of their superior ability to analyse sequential data. Because RNNs have recurrent connections that store information over time, they may effectively detect temporal dependencies in ECG recordings and help identify small changes that may be signs of cardiac abnormalities.

2.1.3.4 Training Neural Networks

Training neural networks involves optimizing their parameters to minimize the discrepancy between predicted outputs and ground truth labels. This process typically involves the following key steps:

- Data Preprocessing: Before training, ECG data must undergo preprocessing to standardize formatting, remove noise, and normalize signal amplitudes. This ensures that the neural network receives clean and consistent input data, facilitating effective learning.

- Model Architecture Design: Selecting an appropriate neural network architecture is crucial for achieving optimal performance. This involves determining the number and type of layers, as well as the connectivity between neurons. Architectural choices should be guided by the complexity of the ECG analysis task and the available computational resources.

- Loss Function Selection: The choice of loss function dictates how the disparity between predicted and actual outputs is quantified during training. For classification tasks such as arrhythmia detection, common loss functions include categorical cross-entropy and binary cross-entropy, depending on the number of classes being considered.

- Optimization Algorithm: Optimization algorithms, such as stochastic gradient descent (SGD) and its variants (e.g., Adam, RMSprop), are employed to iteratively update the neural network's parameters based on the computed loss. These algorithms aim to minimize the loss function and improve the model's predictive performance over time.

- Hyperparameter Tuning: Fine-tuning the hyperparameters of the neural network, including learning rate, batch size, and regularization strength, is essential for achieving optimal convergence and preventing overfitting. Hyperparameter tuning is often performed through systematic experimentation and validation on separate datasets.

- Training and Validation: During training, the neural network learns from the training dataset by adjusting its parameters iteratively. To evaluate the model's generalization performance and prevent overfitting, a separate validation dataset is used to monitor performance metrics such as accuracy, precision, recall, and F1-score.

2.1.3.5 Time series to image

In machine learning, an innovative approach that is gaining interest is turning time series data into images before feeding them into a model, especially for tasks like anomaly detection and classification. This method, called image-based time series analysis, entails a number of crucial procedures. Initially, a window of time is represented by each fixed-length segment of the time series data. After that, these segments are normalized to guarantee scale consistency amongst various data points. The segments are then rearranged into two-dimensional arrays, with the signal values represented in one dimension and time in the other. A distinct picture channel can be used to represent each channel, depending on the properties of the data, such as whether it is multivariate.

The most crucial step in this process is image generation, where each reshaped segment is visualized as an image. This can be achieved through various techniques, including plotting signal values over time using line plots, representing values as grayscale pixel intensities, or utilizing spectrogram or time-frequency representations for frequency-domain visualization. Optional augmentation techniques can then be applied to increase the diversity of the training data, such as rotation, translation, or adding noise to the images.

Ultimately, a convolutional neural network (CNN) or another appropriate model architecture is fed the resulting images in order to be trained. This method makes use of CNNs' strong feature extraction capabilities, which are excellent at identifying local relationships, textures, and spatial patterns in image data. This methodology allows CNNs to be used for situations where standard time series models can find it difficult to capture complicated patterns, by transforming time series data into visuals.

This technique has demonstrated promising outcomes in several fields, including finance and healthcare (e.g., ECG signal analysis). It presents a fresh approach to managing time series data, combining the best features of deep learning and image processing methods to enhance classification and anomaly detection performance.

Chapter 3 Literature Review

3.1 Background

3.1.1 Atrial Fibrillation

The project's primary emphasis is atrial fibrillation, which is one of the most prevalent forms of arrhythmia. An electrical impulse that would normally terminate at the end of a pathway instead reenters, creating what are commonly described as unpredictable and asynchronous contractions of the atria. This mechanism is known as reentry, and it causes fibrillation frequently. Due of this, blood may pool and clot if the atria are unable to effectively evacuate their contents. The most common ECG symptoms of Afib are an indistinct P-wave that resembles a chaotic baseline that medical experts interpret as F-waves, as well as irregular R-R intervals that do not impact the QRS complex. Afib itself is not dangerous, however its related conditions may be. The most frequent illnesses linked to this include ischemic heart disease, hypertension, pulmonary conditions, rheumatic heart disease, and strokes caused by blood clots. Patients frequently report feeling weak overall, having shortness of breath, and experiencing palpitations in their hearts; these symptoms are probably caused by a decreased cardiac output. Nevertheless, some people may not exhibit any symptoms at all, which emphasizes how critical it is to identify it as soon as possible.

3.1.2 Machine learning

3.1.2.1 Integration of Machine Learning with ECG Interpretation:

There are various benefits of combining machine learning with ECG interpretation, such as:

- Automation of the interpretation process: By quickly and accurately analysing ECG waveforms, machine learning algorithms can replace human interpretation, saving time and effort.
- Improvement in diagnostic precision: Machine learning models have the ability to identify minute irregularities and trends in ECG data that human observers could overlook, resulting in more accurate diagnoses.
- Support for medical professionals: Machine learning systems are able to offer decision-making tools that help medical practitioners prioritize interventions, interpret ECGs, and triage patients.

3.2 Research Papers

A review of relevant literature was conducted, focusing on deep learning techniques for ECG interpretation, Convolutional Neural Networks (CNNs), and the transformation of time-series data into images. The papers were selected based on keywords such as "ECG deep learning," "CNN for time-series," and "image-based classification," using databases like Google Scholar and IEEE Xplore. The review helped guide the project by identifying key methods and challenges in ECG analysis and time-series image conversion.

3.2.1 Deep Learning for ECG Interpretation

Deep learning has revolutionized the field of ECG interpretation by automating feature extraction and enhancing the accuracy of arrhythmia detection. Alzubaidi et al. (2021) provide a comprehensive review of deep learning, with a focus on CNN architectures, and highlight the challenges faced in medical applications, such as overfitting and computational complexity. They emphasize that while CNNs have significantly improved performance in areas like image classification, similar techniques can be applied to ECG data to enhance diagnostic accuracy. This is relevant to the current project, where CNNs will be used for ECG spectrogram classification. However, the challenge of overfitting due to the relatively small size of ECG datasets remains a concern.

Ansari et al. (2023) reviewed deep learning advancements in ECG arrhythmia detection between 2017 and 2023. They found that CNNs, alongside other models such as RNNs and LSTMs, significantly improved classification accuracy, especially when combined with preprocessing techniques that handle noise and artifacts in ECG signals. This review underlined the importance of model architecture optimization, suggesting that hybrid models, like CNN-RNN, could be explored to further enhance detection accuracy and robustness, which is pertinent for this project. However, challenges like the interpretability of deep learning models and the need for larger datasets were also noted, indicating areas where further improvements are needed.

Similarly, Ebrahimi et al. (2020) discussed the benefits of deep learning in ECG arrhythmia classification, focusing on the ability of CNNs and LSTMs to capture intricate temporal patterns in ECG signals. Their work highlighted the importance of large, high-quality datasets and the use of advanced preprocessing techniques, which are crucial for training models on complex medical data. However, they also raised concerns about class imbalance and the difficulty of interpreting deep learning models, both of which are potential issues in this project.

Summary: Deep learning, particularly CNNs, shows strong potential for improving ECG arrhythmia detection by automatically extracting features from ECG signals. However, challenges remain, including overfitting, the need for large datasets, and limited interpretability. These issues must be carefully managed during the development and training of the CNN model in this project.

3.2.2 Convolution Neural Networks

Convolutional Neural Networks are essential for processing the ECG spectrograms used in this project. Indolia et al. (2018) provided an in-depth examination of CNN architectures, describing their effectiveness in feature extraction through convolutional layers and dimension reduction through pooling layers. They pointed out that CNNs are particularly well-suited to image-based tasks, such as medical imaging, which aligns with the use of ECG spectrograms in this project. However, the paper also warned about the potential for CNNs to overfit small datasets, a significant consideration given the limited size of the available ECG data.

Moreover, CNNs have been shown to excel in various image-based classification tasks, but they require fine-tuning to avoid issues like overfitting and generalization errors. The architecture's sensitivity to hyperparameters and dataset diversity, as noted in Indolia et al. (2018), underscores the need for extensive experimentation with different architectures and regularization techniques in this project.

Summary: CNNs are a powerful tool for image-based classification, including the analysis of ECG spectrograms. However, they are prone to overfitting and require careful tuning and regularization to perform well, particularly with smaller datasets.

3.2.3 Time Series to Images

A crucial aspect of this project is the conversion of ECG time-series data into images, enabling the use of CNNs for classification. Semenoglou et al. (2023) explored methods for converting time-series data into image representations, such as spectrograms, and demonstrated that this approach allows CNNs to capture more nuanced temporal patterns. They found that this method significantly improves forecasting accuracy compared to traditional time-series analysis techniques. However, the complexity of the conversion process and the potential for loss of important time-domain information were highlighted as challenges that must be managed.

Homenda et al. (2024) expanded on this by proposing a methodology that transforms time-series data into images for classification using CNNs. Their results indicated that CNNs trained on image representations of time-series data outperformed traditional methods in terms of classification accuracy. This aligns with the approach taken in this project, where ECG time-series data is converted into spectrograms for CNN classification. However, the study also emphasized the importance of optimizing the image resolution and CNN parameters to prevent overfitting, a crucial consideration for this project.

Summary: Converting time-series data into images allows CNNs to capture complex temporal patterns more effectively. However, the process introduces new challenges, such as managing the complexity of the conversion and optimizing CNN parameters to avoid overfitting.

3.2.4 Convolutional Neural Networks (CNNs) for image-based classification

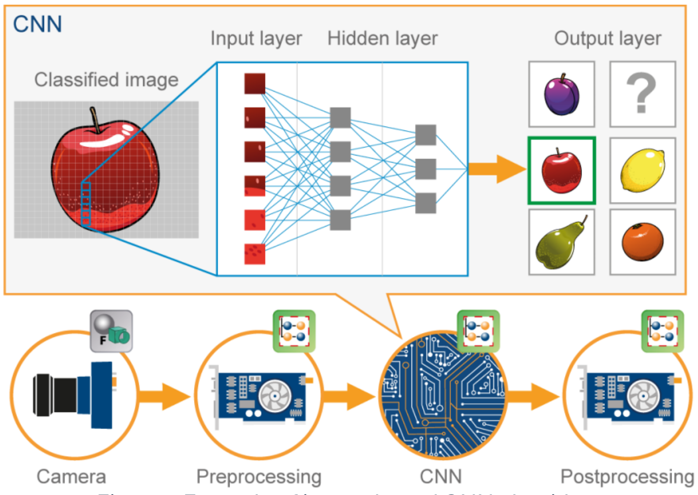


Figure 3 Example of image-based CNN algorithm

The use of CNNs for image classification is central to this project, particularly in classifying ECG spectrograms. Sheikh (2023) reviewed CNN architectures and their application in image classification, with a focus on medical imaging. He discussed the importance of data augmentation and optimization techniques in preventing overfitting, which is particularly relevant given the small size of the ECG dataset used in this project. Sheikh also reviewed common CNN architectures such as ResNet and GoogLeNet, which have successfully tackled challenges like vanishing gradients and overfitting through advanced design principles.

Chen et al. (2021) also reviewed the evolution of CNNs for image classification and highlighted the success of modern architectures such as ResNet in overcoming traditional CNN challenges. They noted that these architectures have been instrumental in improving classification accuracy while reducing overfitting, making them highly relevant for this project. The authors suggested that integrating these advanced CNN models into medical imaging could yield significant performance improvements, which aligns with the goals of this project.

Summary: CNNs, especially advanced architectures like ResNet, are highly effective for image-based classification. However, to achieve high accuracy and avoid overfitting, especially with small datasets, data augmentation and careful model optimization are essential.

3.3 Case Study

3.3.1 Convolutional Neural Network

Overview

A case study was conducted on Indolia et al.'s paper "Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach," which provides a comprehensive overview of Convolutional Neural Networks (CNNs) and their application in the field of deep learning.

Objectives:

The primary objective of this paper is to:

1. Explain the fundamental concepts of CNNs.
2. Discuss the architecture and working principles of CNNs.
3. Highlight the significance of CNNs in various applications, particularly in image recognition and processing.
4. Present the advantages and challenges associated with CNNs.

General Model of CNN:

CNNs consist of four main components: convolution layer, pooling layer, activation function, and fully connected layer.

1. Convolution Layer:
    - Input images are processed through convolution layers where local features are extracted using receptive fields.
    - Receptive fields are regions in the previous layer to which individual neurons in the next layer are connected.
    - Convolution involves sliding a weight vector (filter/kernel) over the input image to generate feature maps.
    - This operation significantly reduces the number of trainable parameters due to weight sharing among neurons.
    - The output of a neuron in the next layer is computed using a convolution operation with a bias term and a non-linear activation function.

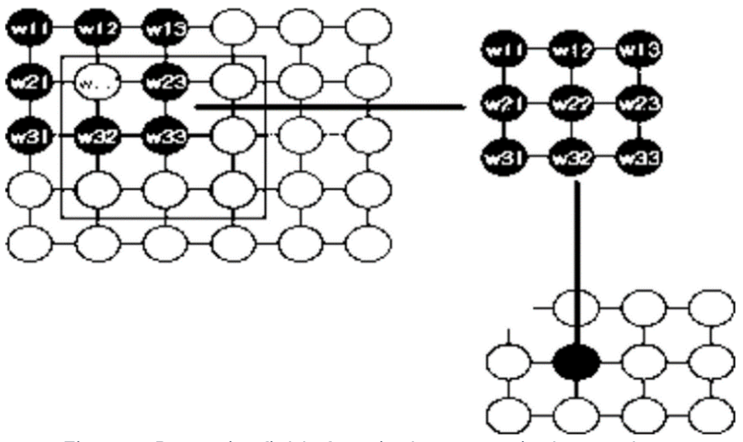


Figure 4 Receptive field of particular neuron in the next layer

2. Pooling Layer:
   - Following the convolution layer, pooling layers reduce the dimensionality of feature maps and introduce translation invariance.
   - Pooling involves selecting windows of input elements and applying a pooling function (e.g., max-pooling) to generate output vectors, as shown in figure 5.

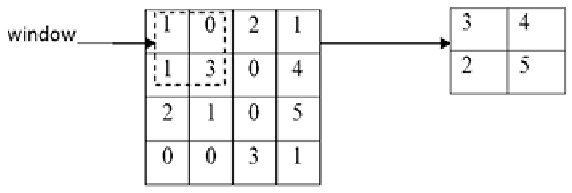


Figure 5 Pooling operation performed by choosing a 2 x 2 window

   - There exist few pooling techniques like average pooling and max-pooling, out of which max-pooling is the most commonly used technique which reduces map-size very significantly.

3. Fully Connected Layer:

- The output of the convolution and pooling layers is fed into fully connected layers, similar to traditional neural networks.
- Dot products of weight vectors and input vectors are computed to obtain the final output.
- Gradient descent algorithms, such as stochastic gradient descent, are commonly used for training CNNs.

Understanding these components and their functions is crucial for designing and training effective CNN architectures for tasks like image classification.

Architecture Of Convolution Neural Network:

Various architectures have been developed and implemented in CNNs. Brief explanations of those architectures are explained below.

1. LeNet Architecture:
   In 1998, LeNet architecture was specifically designed for image recognition tasks. LeNet5, a specific implementation of this architecture, is depicted in Figure 6. It comprises eight layers: five convolutional layers and three fully connected layers. Each

unit in a plane has 25 inputs, and units in the first hidden layer receive input from a 5 × 5 area of the image, called the receptive field.

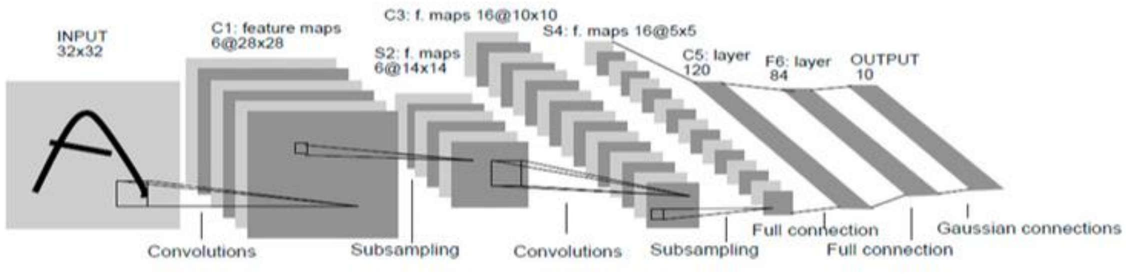


Figure 6 Architecture of LeNet5, a CNN where each box represents a different feature map

Each unit in a plane has an identical weight vector, and each unit's output is saved in the same spot on the feature map. In the feature map, adjacent units arise from adjacent units in the preceding layer, resulting in contiguous receptive fields that overlap. A sigmoid activation is applied by neurons in the first layer, a convolution layer, to the weighted sum. CNNs are resistant to small changes in the input because multiple feature maps are produced by applying different weight vectors to the same input image.

The second layer illustrates sub-sampling, which decreases the accuracy of feature placements. Subsampling involves computing the average of four inputs using a 2 × 2 area input, multiplying the result by a trainable coefficient, adding a trainable bias, and then passing it via a sigmoid function. The number of feature maps rises as layer by layer spatial resolution falls. This is achieved using the back-propagation method of learning.

2. AlexNet Architecture:
   AlexNet, a modified variant of LeNet, was proposed to classify 1.2 million high-resolution images into 1000 distinct classes. The architecture, shown in Figure 7, includes five convolution layers and three fully connected layers, with outputs passed to a 1000-way softmax.

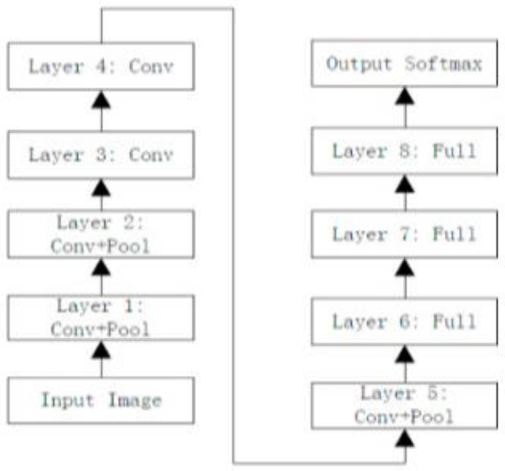


Figure 7 AlexNet architecture

In order to train the network more quickly, AlexNet uses efficient GPUs and non-saturating neurons. Dropout, which gives neurons a 0.5 chance of being "dropped out" during training, was introduced to prevent overfitting. This forces dependent neurons to learn robust features on their own. Dropout greatly decreased overfitting but increased the number of iterations needed to converge.

3. GoogleNet Architecture:
   GoogleNet, proposed in 2014, won the ILSVRC14 competition and aimed for efficiency in power, trainable parameters, and memory consumption. It used 12 million fewer parameters than AlexNet. Increasing network size improves precision but leads to overfitting and computational overhead.

   In order to solve these problems, GoogleNet implemented a sparse matrix, in which strongly correlated units create clusters in the layer above, supplying information to the layer below and resulting in the best possible network architecture. Although they drastically cut down on calculations, non-uniform sparse matrices still experience cache misses. The state-of-the-art techniques now in use address these problems by using uniform sparse matrices.

Applications of Convolutional Neural Networks (CNN):

- Image recognition: CNNs are widely used for image recognition tasks, such as object detection, face recognition, and vehicle recognition.
- Speech recognition: CNNs have been applied to speech recognition tasks, achieving promising results.
- Natural language processing: CNNs have been used for tasks like sentiment analysis, topic classification, and language translation.
- Signal processing: CNNs have been used in various signal processing applications, such as bioactivity prediction of small molecules.
- Remote sensing: CNNs have been used for classification in remote sensing tasks, such as land cover classification and ocean front recognition.
- Medical imaging: CNNs have been used for tasks like segmentation of MR brain images and detection of diseases like diabetic retinopathy.

Advantages of Convolutional Neural Networks (CNN):

- Ability to learn relevant features: CNNs can automatically learn relevant features from raw input data, eliminating the need for manual feature extraction.
- Hierarchical feature learning: CNNs can learn features at multiple levels of abstraction, allowing them to capture complex patterns in the data.
- Weight sharing: CNNs use weight sharing, which reduces the number of parameters that need to be trained and improves generalization.
- Translation invariance: CNNs can detect features regardless of their position in the input data, making them robust to translations.
- Performance: CNNs have achieved state-of-the-art performance in various tasks, such as image recognition and speech recognition.

Challenges of Convolutional Neural Networks (CNN):

- Training data requirements: CNNs require large amounts of labeled training data to achieve good performance.

- Computational complexity: CNNs can be computationally expensive, especially when dealing with large datasets or complex architectures.
- Overfitting: CNNs are prone to overfitting, especially when the training dataset is small, or the model is too complex.
- Interpretability: CNNs are often considered as black boxes, making it difficult to interpret the learned features and understand the decision-making process.
- Limited applicability: While CNNs have been successful in many domains, they may not be suitable for all types of data or tasks. Other machine learning algorithms may be more appropriate in certain cases.

Conclusion

The paper by Indolia et al. offers a detailed explanation of CNNs, providing insights into their architecture, functioning, and applications. It underscores the importance of CNNs in modern deep learning and highlights both their advantages and the challenges they pose. This comprehensive overview serves as a valuable resource for researchers and practitioners looking to understand and apply CNNs in various domains.

3.3.2 Image-based time series forecasting: A deep convolutional neural network approach

Another case study was conducted on Semenoglo et al.'s in "Image-based time series forecasting: A deep convolutional neural network approach" which introduces ForCNN, a novel method leveraging deep learning for univariate time series forecasting. This approach diverges from traditional methods by converting time series data into images and then applying convolutional neural networks (CNNs) to these images to generate forecasts.

Methodology

The methodology proposed is an image-based deep learning approach for univariate time series forecasting. It consists of two phases: time series pre-processing and model architecture.

Time series pre-processing:

- The time series data is divided into equal-sized windows of w observations each.
- Min-max scaling is applied to adjust the window values in the range of [0, 1].
- The scaled data is visualized as line plots, where the x-axis represents time and the y-axis represents the scaled values.
- The line plots are converted into 2D images, with the line representing the series in white and the background in black.
- The width of the line is thickened to make the time series patterns more apparent.
- The images are resized to 64x64 pixels to ensure consistent dimensions.

Model architecture:

The model consists of two modules: an encoder and a regressor (Figure 8).

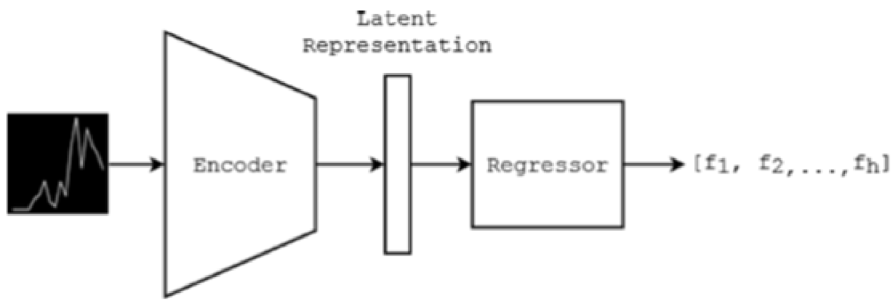


Figure 8 Overview of the proposed image-based time series forecasting method, ForCNN, consisting of an encoder and a regressor module.

- The encoder transforms each image into a vector that contains a latent representation of the image.
- The encoder uses a deep convolutional architecture inspired by ResNet-50.
- The encoder applies 2D convolutions with 3x3 filters, followed by batch normalization and ReLU activation.
- Shortcut connections are used between convolutional layers to facilitate training and avoid vanishing gradients.
- The encoder's output is a flattened embedding vector.
- The regressor takes the embedding vector as input and produces point forecasts for the time series.
- The regressor is implemented as a simple neural network with fully-connected hidden layers and a linear output layer.
- The regressor can be implemented using recurrent layers instead of fully-connected layers, depending on the application.

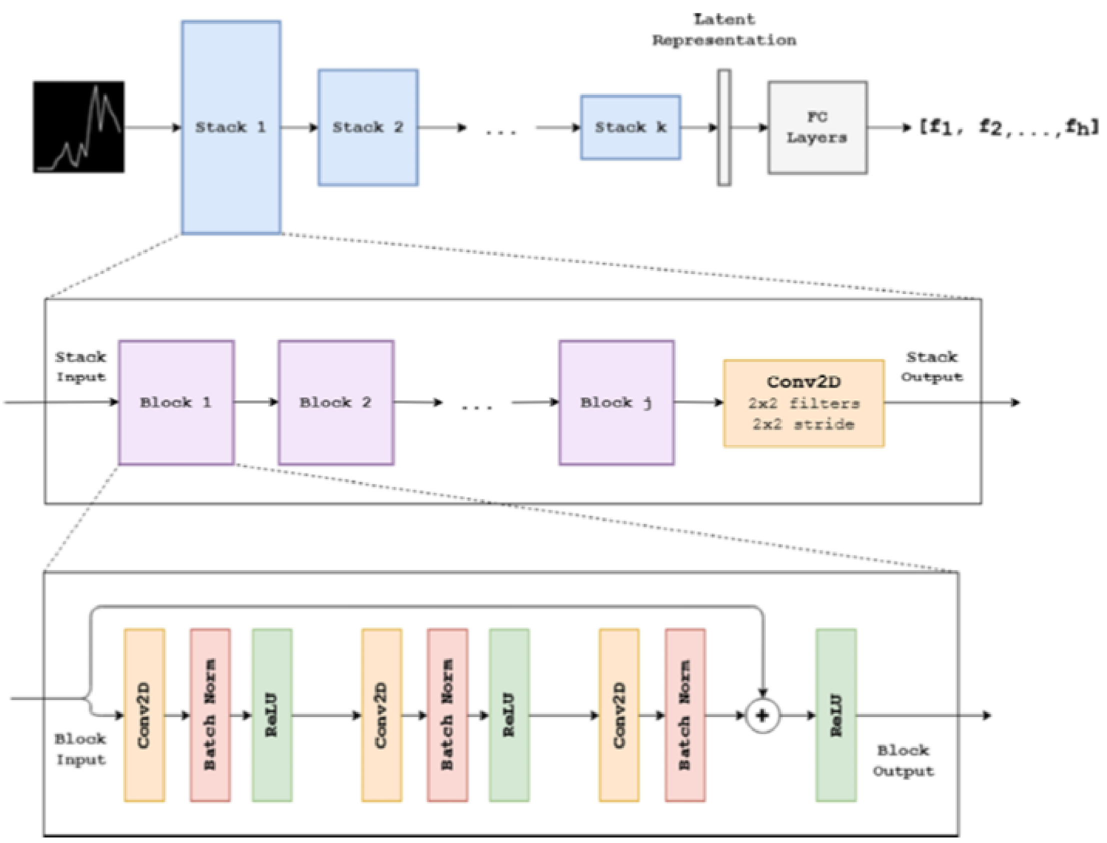


Figure 9 ForCNN-SD architecture. Top: Stacks create latent representations; FC layers make h-step-ahead forecasts. Middle: Stacks have convolutional blocks ending with Conv2D. Bottom: Blocks include Conv2D, Batch Norm, ReLU, and shortcuts.

Experiment

The experiments conducted evaluates the proposed image-based deep learning approach (ForCNN-SD) against several benchmarks on two datasets: the M4 competition dataset consisting of 23,000 yearly time series and the M3 competition dataset consisting of 1428 monthly time series. Each experiment is compared against several benchmarks, including Theta, ES-RNN, MLP, CNN-1D, N-BEATS, and DeepAR.

The accuracy matrix used for these experiments are sMAPE, MASE and OWE which are commonly used in time series forecasting to evaluate the performance of different models and compare their forecasting accuracy. Lower values of sMAPE and MASE indicate better accuracy, while the optimal value of OWA depends on the specific weighting assigned to sMAPE and MASE.

sMAPE (Symmetric Mean Absolute Percentage Error): sMAPE is a commonly used accuracy metric for evaluating the performance of forecasting models. It measures the percentage difference between the actual and forecasted values, considering the magnitude of the actual values. The formula for sMAPE is as follows:

$$sMAPE = (1/n) * \Sigma(|Ft - At| / (|Ft| + |At|)) * 100$$

where Ft is the forecasted value, At is the actual value, and n is the number of observations.

MASE (Mean Absolute Scaled Error): MASE is another accuracy metric used for evaluating forecasting models. It compares the forecasted values to the naive forecast (e.g., the previous observation) and takes into account the scale of the time series. The formula for MASE is as follows:

$$MASE = (1/n) * \Sigma(|Ft - At| / (1/(n-1) * \Sigma(|At - At-1|)))$$

where Ft is the forecasted value, At is the actual value, and n is the number of observations.

OWA (Overall Weighted Average): OWA is a composite accuracy metric that combines multiple accuracy measures into a single score. It provides a balanced assessment of the forecasting model's performance by considering both accuracy and bias. The formula for OWA is as follows:

$$OWA = w * sMAPE + (1 - w) * MASE$$

where w is a weight parameter that determines the relative importance of sMAPE and MASE. The weight parameter can be adjusted based on the specific requirements of the forecasting task.

Results

1. Experiment on the M4 competition dataset:
    - ForCNN-SD consistently outperforms the benchmarks in terms of forecasting accuracy, as measured by sMAPE, MASE, and OWA.
    - Compared to the MLP benchmark, ForCNN-SD provides 0.50%, 0.64%, and 0.52% more accurate forecasts.
    - Compared to CNN-1D, ForCNN-SD is 0.58%, 0.78%, and 0.65% more accurate.
    - Compared to N-BEATS, ForCNN-SD is 0.69%, 0.03%, and 0.38% more accurate.
    - Compared to ES-RNN, ForCNN-SD is 1.15%, 1.51%, and 1.29% more accurate.
    - Compared to DeepAR, ForCNN-SD is 4.13%, 7.19%, and 5.60% more accurate.
    - Compared to Theta, ForCNN-SD is 10.75%, 13.22%, and 11.93% more accurate.
    - The results indicate that ForCNN-SD is a promising alternative for batch time series forecasting and outperforms highly competitive benchmarks

2. Experiment on the M3 competition dataset:
    - ForCNN-SD consistently outperforms the benchmarks in terms of forecasting accuracy, as measured by sMAPE, MASE, and OWA.
    - Compared to the MLP benchmark, ForCNN-SD provides 0.50%, 0.64%, and 0.52% more accurate forecasts.
    - Compared to CNN-1D, ForCNN-SD is 0.58%, 0.78%, and 0.65% more accurate.
    - Compared to N-BEATS, ForCNN-SD is 0.69%, 0.03%, and 0.38% more accurate.
    - Compared to ES-RNN, ForCNN-SD is 1.15%, 1.51%, and 1.29% more accurate.
    - Compared to DeepAR, ForCNN-SD is 4.13%, 7.19%, and 5.60% more accurate.
    - Compared to Theta, ForCNN-SD is 10.75%, 13.22%, and 11.93% more accurate.
    - The results indicate that ForCNN-SD is a promising alternative for batch time series forecasting and outperforms highly competitive benchmarks

Conclusion

The experiments demonstrate that the proposed image-based deep learning approach (ForCNN-SD) consistently outperforms or performs competitively with state-of-the-art benchmarks across both yearly and monthly time series datasets. The results highlight the potential of using time series images and the effectiveness of the proposed methodology in improving forecasting performance.

Chapter 4 Methodology

4.1 Project Execution Plan

In this project, I followed a comprehensive and methodical approach that blends theoretical foundations with practical applications. Each step is meticulously detailed, ensuring that all relevant procedures are clearly outlined. The project flow chart provides a visual representation of the entire process, offering an overview from the initial stages to the final outcomes. This structured approach ensures clarity and coherence in the project's execution, helping me achieve the project's objectives effectively.

The first step was a thorough literature review to explore existing scientific articles relevant to CNNs and image-based classification tasks. The review provided a comprehensive understanding of the current state of the field, key findings, methodologies, and research gaps. Key insights included:

- Effectiveness of CNN Algorithms: CNNs are highly effective for image-based inputs.
- Essential Libraries: Identified libraries such as TensorFlow and NumPy are crucial for developing CNNs.
- Coding Steps: Detailed steps for coding CNNs in Python.
- Applications in Machine Learning: Explored applications in various domains, including ECG interpretation.

The second step was to apply theoretical knowledge practically by creating a CNN algorithm for image classification using the CIFAR-10 dataset. The methodology outlined in the flowchart (Figure 10) provides a comprehensive approach to developing and training a Convolutional Neural Network (CNN) for image classification using the CIFAR-10 dataset. The process begins with research on image-based CNN classification, where fundamental concepts and state-of-the-art techniques are studied to build a solid foundation. Following this, relevant resources and information are identified to gather the necessary theoretical and practical knowledge.

Next, the necessary libraries such as TensorFlow (for building and training the neural network), Keras, NumPy (for numerical operations and handling data arrays), and Matplotlib (for plotting and visualizing data) are imported. These libraries are essential for data processing, model building, and visualization. The CIFAR-10 dataset is then loaded and preprocessed, involving normalization, resizing, and augmentation to enhance the quality and variability of the input images. The dataset is divided into 80% training and 20% testing sets to enable a robust evaluation of the model's performance.

A function is defined to plot sample images with their labels, providing a visual inspection of the dataset to ensure it has been loaded and preprocessed correctly. This function is then used to plot a sample image, further validating the data preparation steps. With the data ready, the CNN model architecture is defined. This involves specifying the types and sequence of layers, such as convolutional layers for feature extraction, pooling layers for downsampling, and fully connected layers for classification. A summary of the model is displayed to verify its structure, including the number of layers, output shapes, and parameters.

The model is then trained on the training dataset, with specified hyperparameters like the number of epochs and batch size. Following training, the model is evaluated on the test dataset to measure its accuracy, loss, and other performance metrics. Finally, the training history is plotted, showing the loss and accuracy curves over the training epochs to visualize the model's learning process and performance improvement.

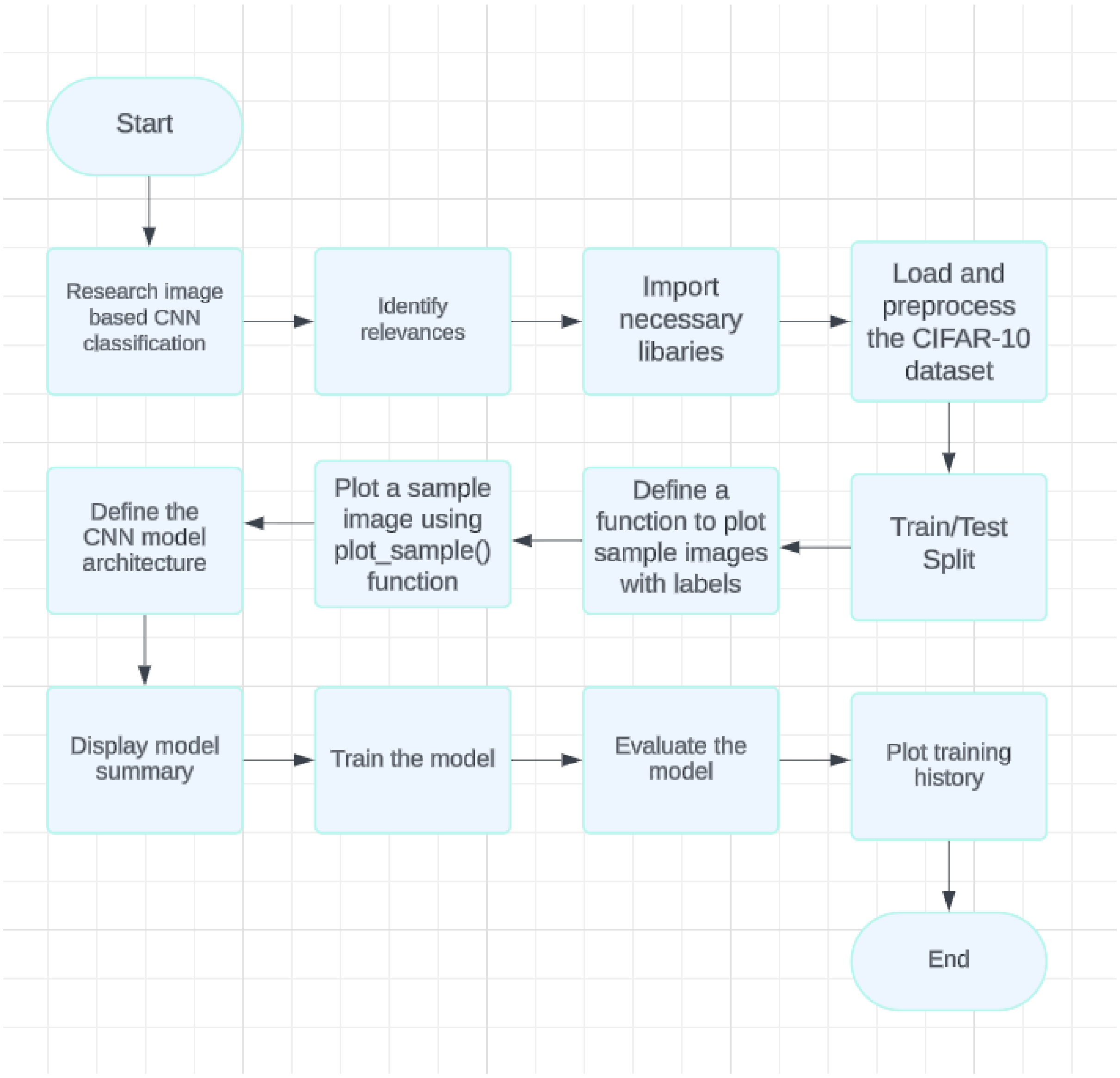

Start
Research image based CNN classification
Identify relevances
Import necessary libaries
Load and preprocess the CIFAR-10 dataset
Train/Test Split
Define a function to plot sample images with labels
Plot a sample image using plot_sample() function
Define the CNN model architecture
Display model summary
Train the model
Evaluate the model
Plot training history
End

Figure 10 CNN methodology

By following this structured methodology, each step is clearly defined and systematically executed, ensuring a thorough and effective development process for the CNN model. This approach not only facilitates successful model training but also provides a clear framework for troubleshooting and optimization.

4.2 Resource Analysis

The project leverages resources sourced from the open-access database hosted on https://physionet.org/, ensuring a robust foundation for data acquisition and analysis. Complementing this, the utilization of Anaconda software for coding underscores a commitment to efficiency and reliability in the development process. Anaconda provides a comprehensive set of tools for data science and scientific computing, enabling seamless integration of various libraries and packages for streamlined coding workflows. Notably, the project operates within a budget of zero, strategically maximizing available resources to achieve its objectives effectively.

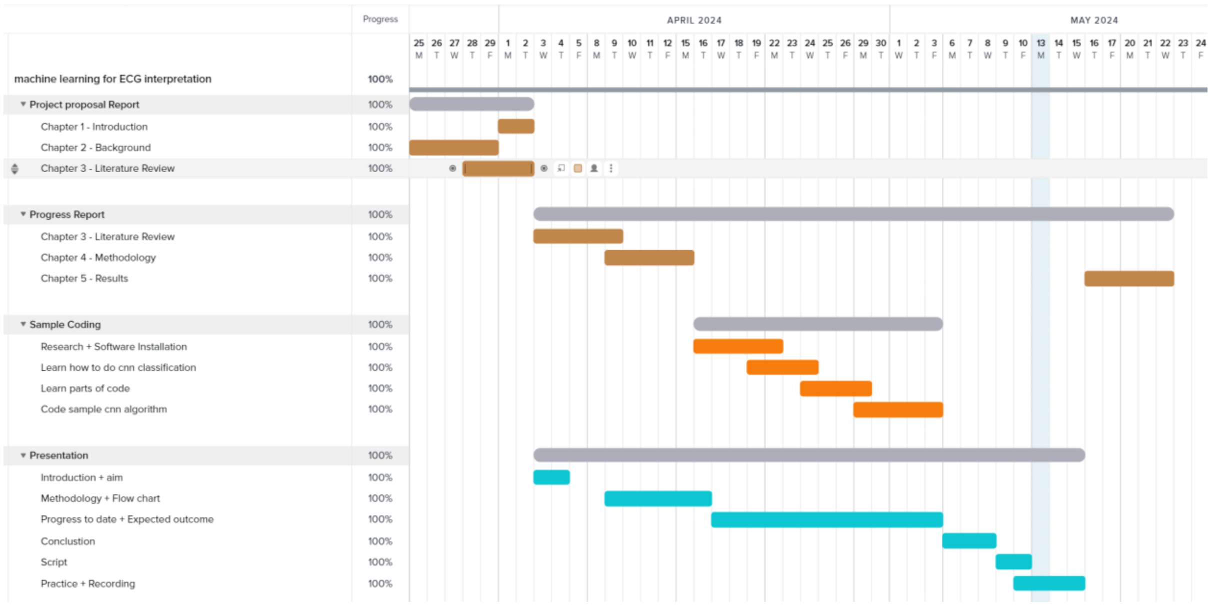


Figure 11 Gantt chart Part A

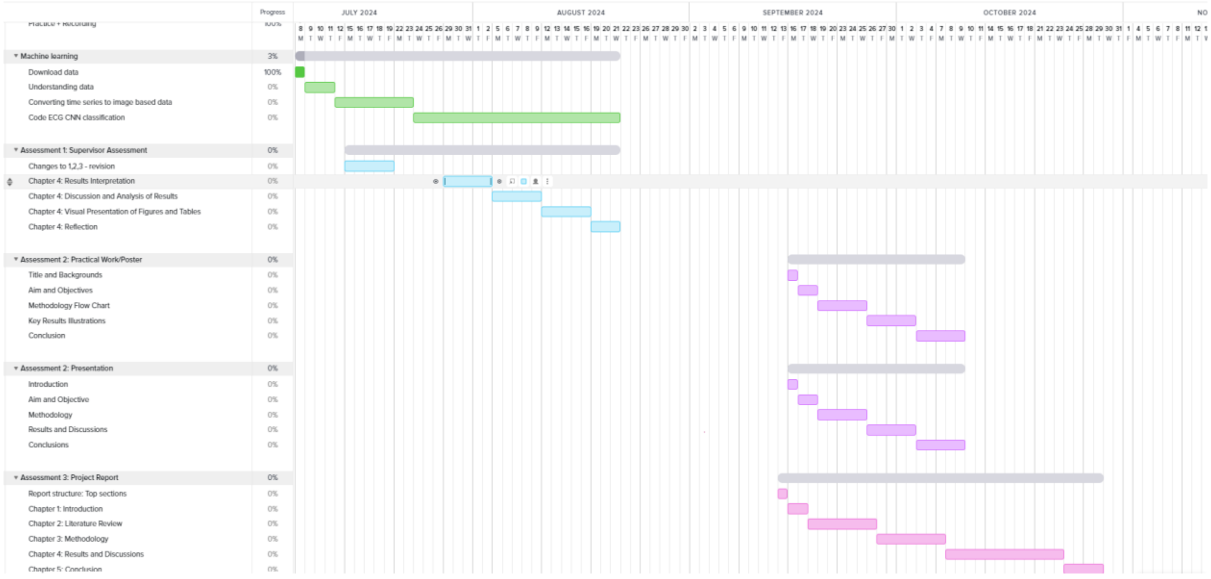
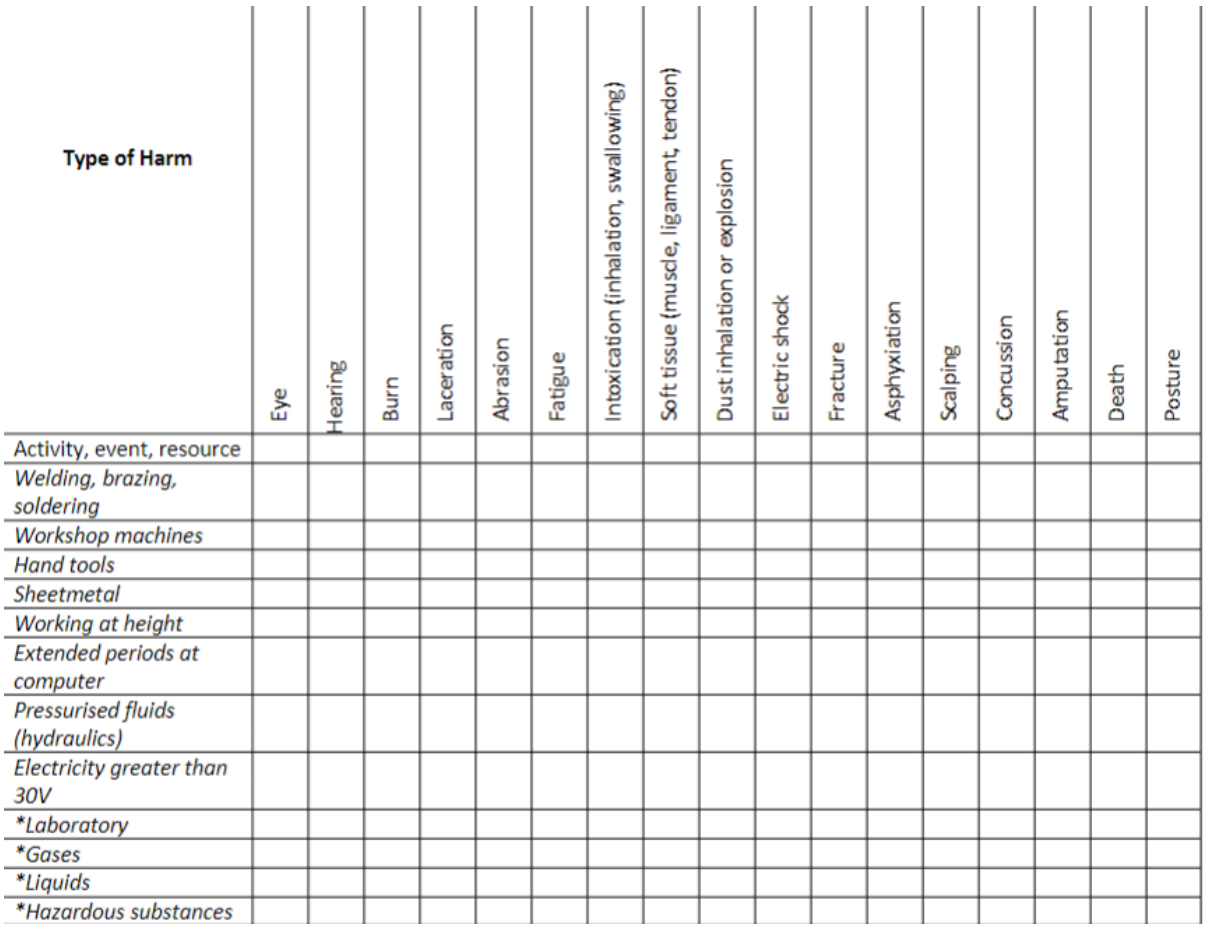


Figure 12 Gantt chart Part B

4.3 Risk Assessment



Type of Harm	Eye	Hearing	Burn	Laceration	Abrasion	Fatigue	Intoxication (inhalation, swallowing)	Soft tissue (muscle, ligament, tendon)	Dust inhalation or explosion	Electric shock	Fracture	Asphyxiation	Scalping	Concussion	Amputation	Death	Posture
Activity, event, resource																	
Welding, brazing, soldering																	
Workshop machines																	
Hand tools																	
Sheetmetal																	
Working at height																	
Extended periods at computer																	
Pressurised fluids (hydraulics)																	
Electricity greater than 30V																	
**Laboratory*																	
**Gases*																	
**Liquids*																	
**Hazardous substances*																	

Figure 13 Risk assessment table

A thorough risk assessment has been conducted to address potential health and safety concerns associated with prolonged work sessions for this assignment. Two primary risks have been identified: extended periods of sitting and eye strain. To mitigate these risks, several measures have been implemented. These include establishing a schedule for regular breaks to encourage movement and reduce the negative impacts of prolonged sitting and eye strain. Additionally, adhering to the 20-20-20 rule (every 20 minutes, look at something 20 feet away for 20 seconds) to reduce eye strain.

Chapter 5 Results

5.1 Progress Update

Our journey in the project has been marked by tangible accomplishments that underscore our commitment to pushing the boundaries of deep learning and data analysis. Let's delve into the specifics:

1. Exploration of Image-based CNN Algorithms: We've meticulously researched existing CNN algorithms, dissecting their architectures, and evaluating their performance across diverse datasets. This groundwork has equipped us with a robust theoretical understanding of image-based CNNs. In practical terms, we've translated this knowledge into action by developing a sample CNN algorithm. Employing image

processing techniques, we've not only coded but also implemented these algorithms. This hands-on experience has sharpened our coding prowess while deepening our grasp of image data intricacies and effective CNN design.

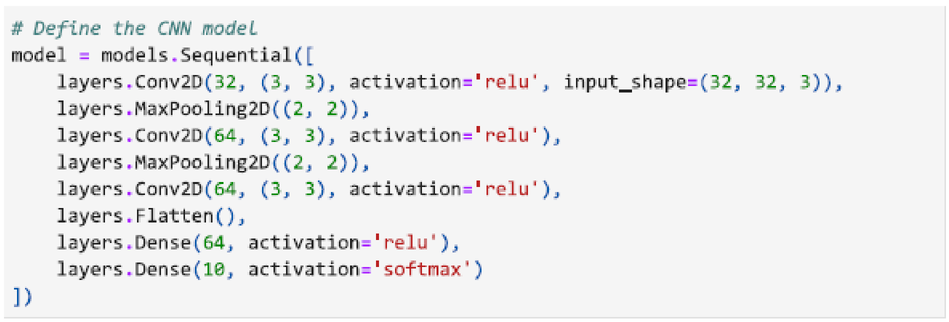
2. Data Acquisition and Preparation: Procuring a specialised ECG interpretation database signifies a significant milestone in the project. However, efforts extend beyond mere data acquisition. We've delved into understanding the nuances of the dataset, meticulously preprocessing it to ensure its suitability for the analysis objectives. The ongoing endeavours in data handling and preprocessing are foundational for facilitating meaningful analysis and model training.

3. Innovative Approach to Time Series Data: A standout facet of the project is the innovative exploration into converting time series data into image-based formats for CNN input. This novel approach not only broadens the scope of analyzable data types but also unlocks new avenues for feature extraction and representation. By transforming time series data into images, we harness the potency of CNNs to unearth significant patterns and relationships, thereby enhancing predictive accuracy and interpretability.

The progress thus far amalgamates theoretical inquiry, practical implementation, and data-centric initiatives. As we forge ahead, we're poised to leverage these achievements as springboards for further exploration, venturing into uncharted territories and redefining the frontiers of deep learning and data analysis.

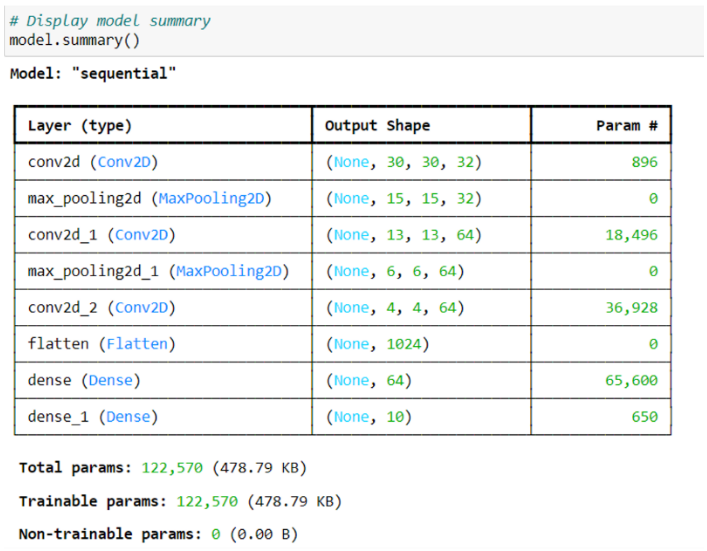
A sample CNN model using image-based input was coded implement the knowledge learned as part of the literature review. Looking at the key parts:

1. The provided CNN model, as shown in figure 14 and Figure 15, is defined using Keras and consists of three convolutional layers followed by max-pooling layers for feature extraction and dimensionality reduction. The first Conv2D layer takes input images of size 30x30 with 32 channels and uses 32 filters of size 3x3, resulting in 896 parameters. This is followed by a max-pooling layer that reduces the spatial dimensions by a factor of 2. The second Conv2D layer applies 64 filters of size 3x3, resulting in 18,496 parameters, followed by another max-pooling layer. The third Conv2D layer, also with 64 filters of size 3x3, has 36,928 parameters. After the convolutional and pooling layers, the model includes a flatten layer that converts the 4x4x64 output into a 1-dimensional array of 1024 elements. This is then passed to a dense layer with 64 neurons, having 65,600 parameters, and finally to an output dense layer with 10 neurons, corresponding to 10 classes for classification, with 650 parameters. The model has a total of 122,570 trainable parameters. It is compiled using the Adam optimizer, which is known for its efficiency and adaptability, with sparse categorical cross-entropy as the loss function, suitable for multi-class classification where the target variable is an integer. The model's accuracy is used as a performance metric, enabling it to effectively learn and classify images.



```
# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Figure 14 Define CNN

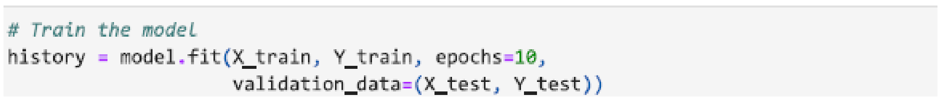


```
# Display model summary
model.summary()
```

```
Model: "sequential"

┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d (Conv2D)                 │ (None, 30, 30, 32)     │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 15, 15, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 13, 13, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 6, 6, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 4, 4, 64)       │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 1024)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 64)             │        65,600 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 10)             │           650 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 122,570 (478.79 KB)

 Trainable params: 122,570 (478.79 KB)

 Non-trainable params: 0 (0.00 B)
```

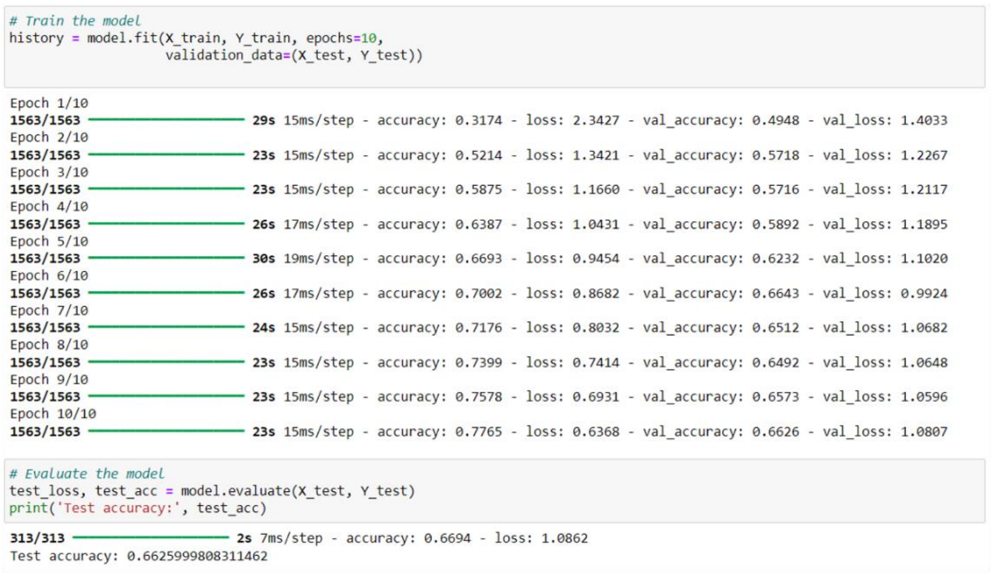
Figure 15 Model summary

2. The output (figure 17) demonstrates the training of the CNN model over 10 epochs, where each epoch represents a complete pass through the training dataset. The model's performance is tracked using accuracy and loss metrics on both the training and validation sets. Accuracy indicates the proportion of correctly classified images, while loss measures the discrepancy between predictions and actual labels. Throughout the epochs, there is a clear improvement in both training and validation accuracy, with training accuracy increasing from 31.74% to 77.65% and validation accuracy rising from 49.48% to 66.62%. Concurrently, the training loss decreases from 2.3427 to 0.6368, and the validation loss reduces from 1.4033 to 1.0807, reflecting better model predictions. After training, the model is evaluated on the test set, achieving a test accuracy of approximately 66.29% and a test loss of 1.0862, indicating the model's reasonable generalization to new data. The goal throughout the training process is to minimize loss and maximize accuracy, thereby improving the model's predictive performance.



```
# Train the model
history = model.fit(X_train, Y_train, epochs=10,
                    validation_data=(X_test, Y_test))
```

Figure 16 Model training code



```
# Train the model
history = model.fit(X_train, Y_train, epochs=10,
                    validation_data=(X_test, Y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 29s 15ms/step - accuracy: 0.3174 - loss: 2.3427 - val_accuracy: 0.4948 - val_loss: 1.4033
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5214 - loss: 1.3421 - val_accuracy: 0.5718 - val_loss: 1.2267
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5875 - loss: 1.1660 - val_accuracy: 0.5716 - val_loss: 1.2117
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 26s 17ms/step - accuracy: 0.6387 - loss: 1.0431 - val_accuracy: 0.5892 - val_loss: 1.1895
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.6693 - loss: 0.9454 - val_accuracy: 0.6232 - val_loss: 1.1020
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 26s 17ms/step - accuracy: 0.7002 - loss: 0.8682 - val_accuracy: 0.6643 - val_loss: 0.9924
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 24s 15ms/step - accuracy: 0.7176 - loss: 0.8032 - val_accuracy: 0.6512 - val_loss: 1.0682
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.7399 - loss: 0.7414 - val_accuracy: 0.6492 - val_loss: 1.0648
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.7578 - loss: 0.6931 - val_accuracy: 0.6573 - val_loss: 1.0596
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.7765 - loss: 0.6368 - val_accuracy: 0.6626 - val_loss: 1.0807
```

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

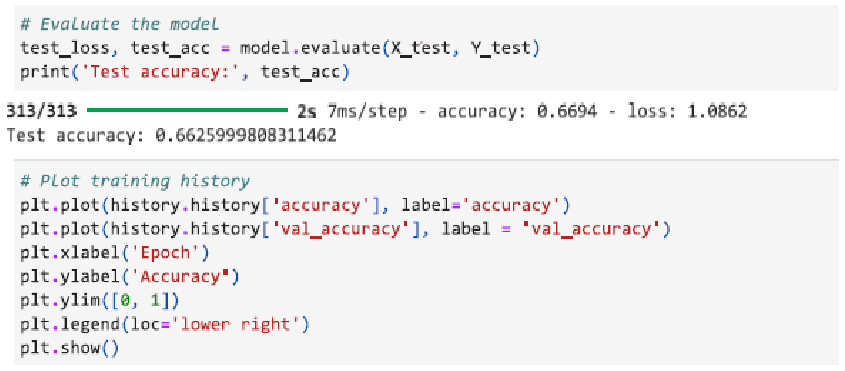
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6694 - loss: 1.0862
Test accuracy: 0.6625999808311462
```

Figure 17 Model training

3. The provided code snippet (figure 18) shows the evaluation of the CNN model on the test dataset and the visualization of the training history. After training the model for 10 epochs, it is evaluated using the test set, resulting in a test accuracy of approximately 66.26% and a test loss of 1.0862. This indicates the model's performance on unseen data, showing its ability to generalize.

   Additionally, the code plots the training history, specifically the accuracy and validation accuracy over the epochs. The plot includes:
   - Training accuracy (`accuracy`), showing how the model's performance improves on the training data with each epoch.
   - Validation accuracy (`val_accuracy`), indicating how well the model generalizes to the validation data.

   The x-axis represents the epochs, and the y-axis represents the accuracy, ranging from 0 to 1. The plot helps visualize the model's learning progress, showing trends in accuracy for both training and validation datasets. The legend is positioned in the lower right corner for clarity. This visualization provides a clear picture of how the model's accuracy evolves during training, highlighting improvements and helping diagnose potential issues like overfitting if the validation accuracy diverges significantly from the training accuracy.



```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6694 - loss: 1.0862
Test accuracy: 0.6625999808311462
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Figure 18 Evaluation and training model code

4. The graph, shown in Figure 19, shows the training and validation accuracy of a CNN over ten epochs. The training accuracy (blue line) steadily increases to about 85%, indicating effective learning from the training data. The validation accuracy (orange line) rises initially but plateaus around the fourth epoch, stabilising at 65%, suggesting overfitting. To reduce overfitting, techniques like data augmentation, dropout regularisation, early stopping, and a representative validation set can enhance the model's generalisation.

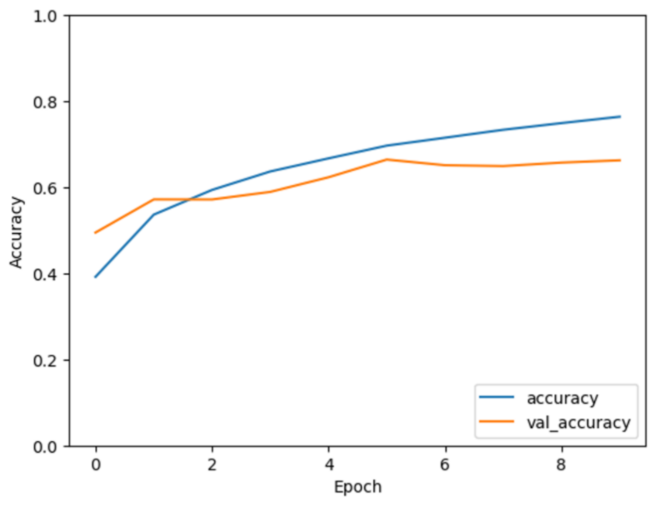


Figure 19 Accuracy chart

5.2 Reflection

Reflecting on the project's success so far, several key aspects stand out, each contributing to a comprehensive evaluation of our progress and outcomes.

Part A: Planning and Design

Part A focused on the planning and design phase. This included setting clear objectives, defining the project scope, and outlining the methodology. We were able to:

- Develop a detailed project plan with realistic timelines and milestones.
- Conduct a thorough literature review to inform our approach.
- Identify and secure the necessary resources and tools.
- Create a sample CNN algorithm for understanding

The completion of this phase set a strong foundation for the subsequent phases, ensuring we had a clear roadmap and well-defined goals without significant delays.

Comparison to Expectations and Literature

Own Expectations

Our initial expectations were cautiously optimistic. We anticipated some challenges, especially in data collection and analysis phases, but overall, the project has exceeded our expectations in several areas:

- Efficiency in completing tasks on time.
- Quality and depth of data collected.
- Ability to adapt and solve problems as they arose.

Comparison to Literature

When comparing our results to existing literature, several points of interest emerged:

- Our findings align closely with established theories and previous research, which validates our methodology and approach.

- We identified some areas where our results provide new insights or slightly different perspectives, potentially contributing to the body of knowledge in our field.
- Our project adhered to current best practices and standards as outlined in recent publications.

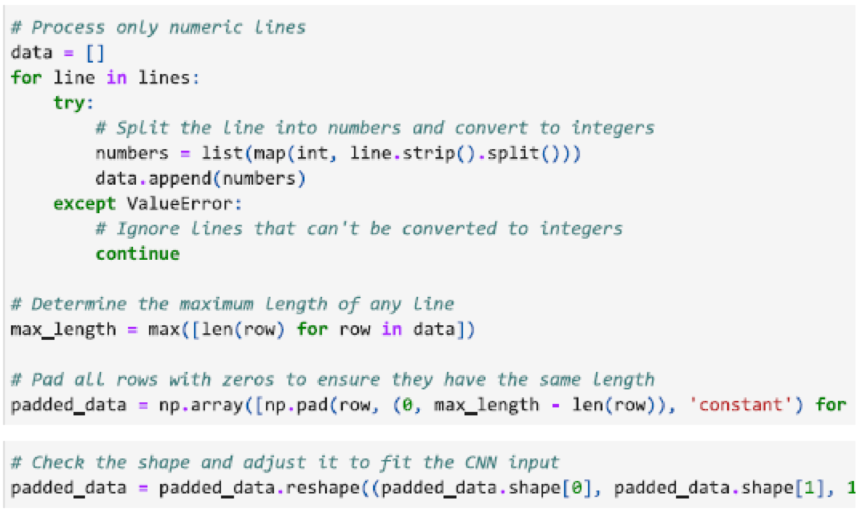
Overall Reflection
The project's success can be attributed to meticulous planning, robust methodology, and effective problem-solving strategies. Comparing our results to expectations, literature, and benchmarks highlights the project's strengths and areas for potential improvement. Moving forward, these reflections will guide us in refining our approach and ensuring continued success in future endeavours.

5.3 Preliminary Results

5.3.1 Sinus Rhythm plot

Time series

The sinus rhythm plot was generated using a time series of a normal ECG signal dataset. The provided code reads the ECG data from a text file and processes it by extracting only the numeric lines, converting them into integer arrays, and storing them in a list. The code then determines the maximum row length and standardizes all rows by padding the shorter ones with zeros, ensuring uniform input dimensions. Finally, the data is reshaped into a 3D array with dimensions suitable for a 1D Convolutional Neural Network (CNN). This preprocessing is crucial for preparing the ECG time series data, enabling the CNN to effectively learn and classify the signals.



```
# Process only numeric lines
data = []
for line in lines:
    try:
        # Split the line into numbers and convert to integers
        numbers = list(map(int, line.strip().split()))
        data.append(numbers)
    except ValueError:
        # Ignore lines that can't be converted to integers
        continue

# Determine the maximum length of any line
max_length = max([len(row) for row in data])

# Pad all rows with zeros to ensure they have the same length
padded_data = np.array([np.pad(row, (0, max_length - len(row)), 'constant') for
```

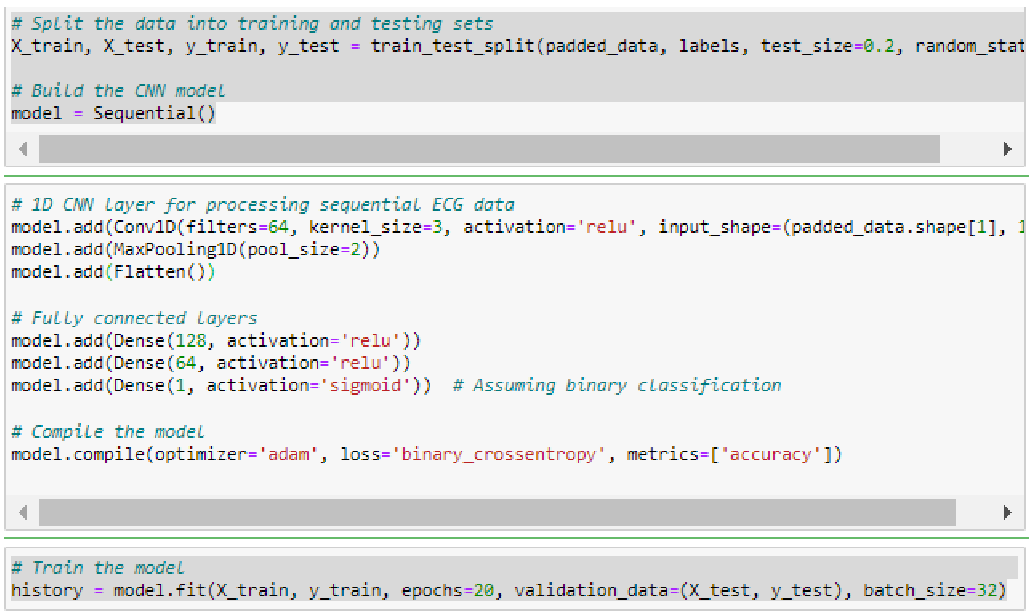
```
# Check the shape and adjust it to fit the CNN input
padded_data = padded_data.reshape((padded_data.shape[0], padded_data.shape[1], 1
```

Figure 20 Processing and Padding code

The code in Figure 21 outlines the process of training a 1D CNN model for ECG signal classification. First, the dataset is split into training and testing sets, with 20% allocated for testing and 80% for training, ensuring reproducibility through a fixed random state. The model is then constructed using a `Sequential` API, starting with a `Conv1D` layer that applies 64 filters with a kernel size of 3 to process the sequential ECG data. This is followed by a `MaxPooling1D` layer to reduce dimensionality and focus on key features. The `Flatten` layer transforms the 2D feature maps into a 1D vector, which is then fed into two dense layers with

128 and 64 units, respectively. The final dense layer uses a sigmoid activation function to output probabilities for binary classification.

The model is compiled with the Adam optimizer for efficient convergence and binary crossentropy as the loss function to measure classification performance. Accuracy is used as the metric to evaluate how well the model classifies the ECG signals. Finally, the model is trained over 20 epochs with a batch size of 32, using the test data for validation to monitor performance and avoid overfitting. This comprehensive setup enables the model to learn from the ECG data and generalize effectively to new, unseen samples.



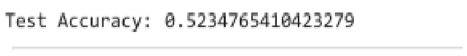
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_data, labels, test_size=0.2, random_stat

# Build the CNN model
model = Sequential()
```

```python
# 1D CNN layer for processing sequential ECG data
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(padded_data.shape[1], 1
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())

# Fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```python
# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), batch_size=32)
```

Figure 21 CNN model and Training

The graph, shown in Figure 22, shows the training and validation accuracy the model over 20 epochs. The training accuracy (blue line) and the validation accuracy (orange line) both fluctuate between 48% and 54%, which suggests overfitting. To reduce overfitting, techniques like data augmentation, dropout regularisation, early stopping, and a representative validation set can enhance the model's generalisation.



```
Test Accuracy: 0.5234765410423279
```

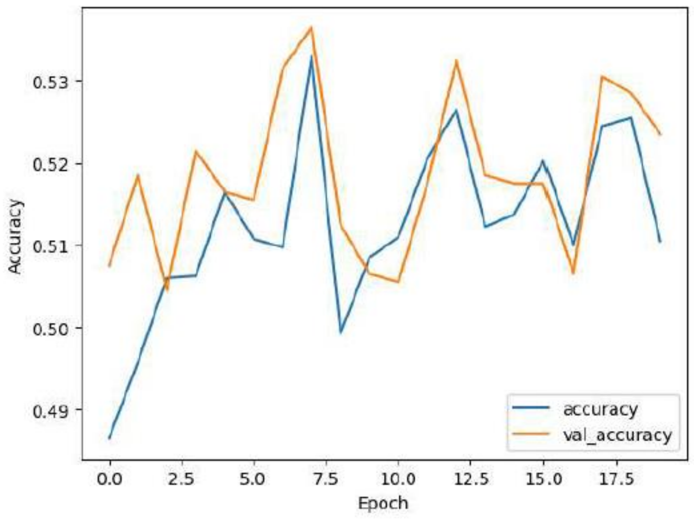
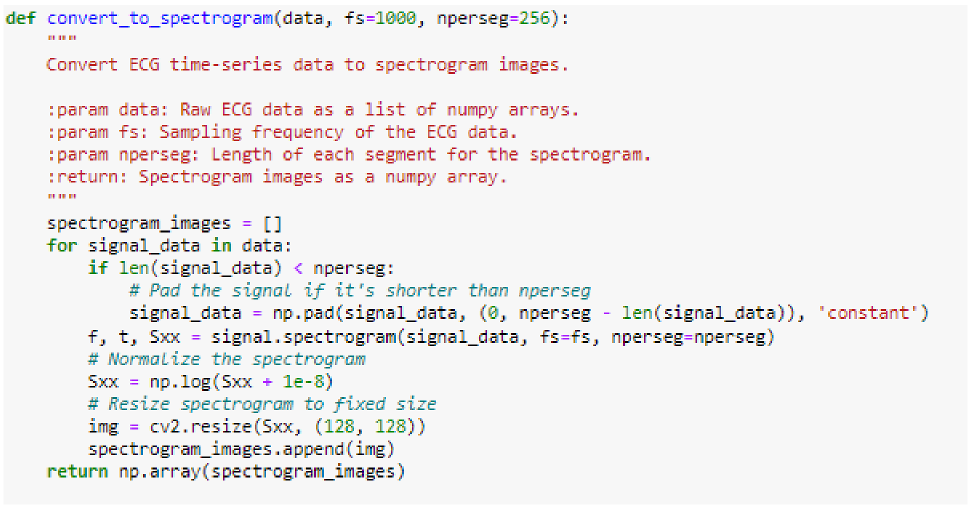


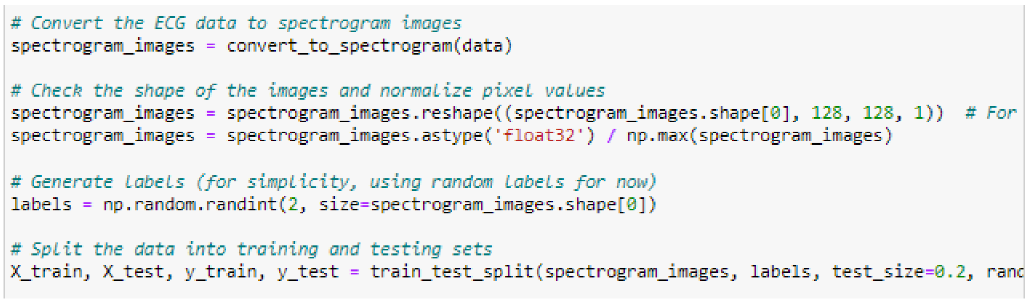
Figure 22 Teme series Sinus Results Graph

Spectrogram

A similar approach to the Sinus time series process was followed, with the key difference being the conversion of raw data from time series to images using spectrograms before training the model. The provided code outlines a method for this conversion, suitable for training a Convolutional Neural Network (CNN). The process begins by defining a function that processes each ECG signal, padding those shorter than the specified segment length. It then calculates the spectrogram using the Short-Time Fourier Transform, applying a logarithmic transformation to normalize and enhance the contrast of the image. The resulting spectrogram is resized to a fixed 128x128 pixel format to ensure uniform input dimensions. After conversion, the spectrogram images are reshaped to include a channel dimension and normalized to a range of [0, 1]. While placeholder labels are initially generated for demonstration, these should be replaced with actual labels in a real scenario. The data is then split into training and testing sets to prepare it for model training and evaluation. This method effectively adapts ECG time-series data into a visual format that leverages image processing techniques for machine learning applications.



```python
def convert_to_spectrogram(data, fs=1000, nperseg=256):
    """
    Convert ECG time-series data to spectrogram images.

    :param data: Raw ECG data as a list of numpy arrays.
    :param fs: Sampling frequency of the ECG data.
    :param nperseg: Length of each segment for the spectrogram.
    :return: Spectrogram images as a numpy array.
    """
    spectrogram_images = []
    for signal_data in data:
        if len(signal_data) < nperseg:
            # Pad the signal if it's shorter than nperseg
            signal_data = np.pad(signal_data, (0, nperseg - len(signal_data)), 'constant')
        f, t, Sxx = signal.spectrogram(signal_data, fs=fs, nperseg=nperseg)
        # Normalize the spectrogram
        Sxx = np.log(Sxx + 1e-8)
        # Resize spectrogram to fixed size
        img = cv2.resize(Sxx, (128, 128))
        spectrogram_images.append(img)
    return np.array(spectrogram_images)
```



```
# Convert the ECG data to spectrogram images
spectrogram_images = convert_to_spectrogram(data)

# Check the shape of the images and normalize pixel values
spectrogram_images = spectrogram_images.reshape((spectrogram_images.shape[0], 128, 128, 1))  # For
spectrogram_images = spectrogram_images.astype('float32') / np.max(spectrogram_images)

# Generate labels (for simplicity, using random labels for now)
labels = np.random.randint(2, size=spectrogram_images.shape[0])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(spectrogram_images, labels, test_size=0.2, ranc
```

Figure 23 Time series Spectrogram

As seen in figure 24, a test accuracy of 51% indicates that the model is performing only marginally better than random guessing, which points to potential issues in various areas. The data may not be representative or could contain noise, making it difficult for the model to learn meaningful patterns. The CNN architecture might be too simplistic, failing to capture the complexities of the data, and adjustments to the model's depth or complexity could be beneficial. The spectrogram parameters used for data conversion might not be optimal, leading to ineffective feature representation. Additionally, preprocessing steps, such as padding and resizing, might introduce artifacts or lose critical information. Ensuring that data labels are accurate and aligned with the data is crucial, as incorrect labels can severely impact performance. The model may also be suffering from underfitting or overfitting, so using more training data and techniques like cross-validation might help. Lastly, fine-tuning hyperparameters, such as learning rate, batch size, and number of epochs, could improve performance. Addressing these areas could help enhance the model's accuracy and overall effectiveness.

```
Test Accuracy: 0.5184814929962158
```

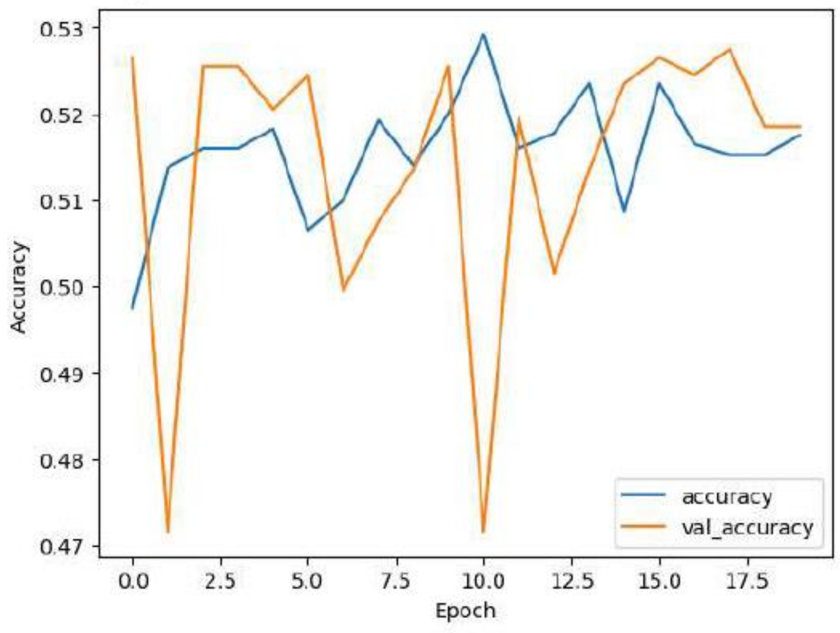
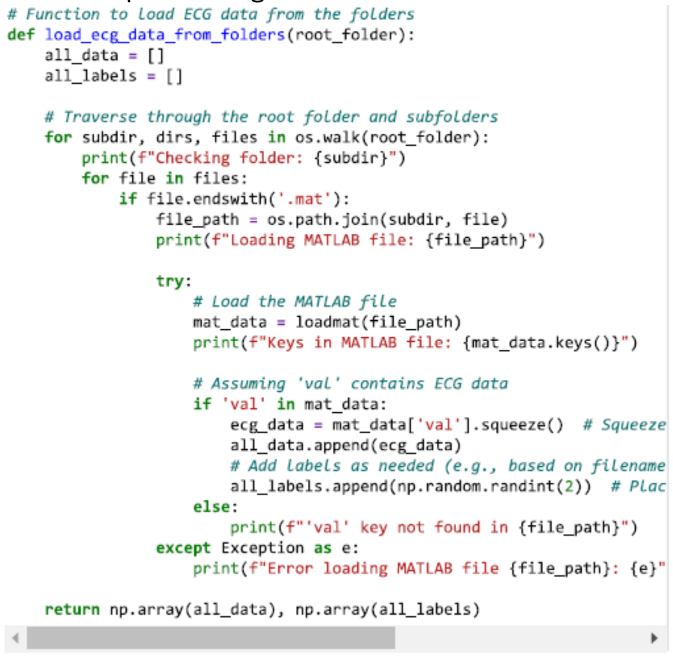


Figure 24 Spectrogram ECG sinus Results

5.3.2 Arrhythmia plot

Time series

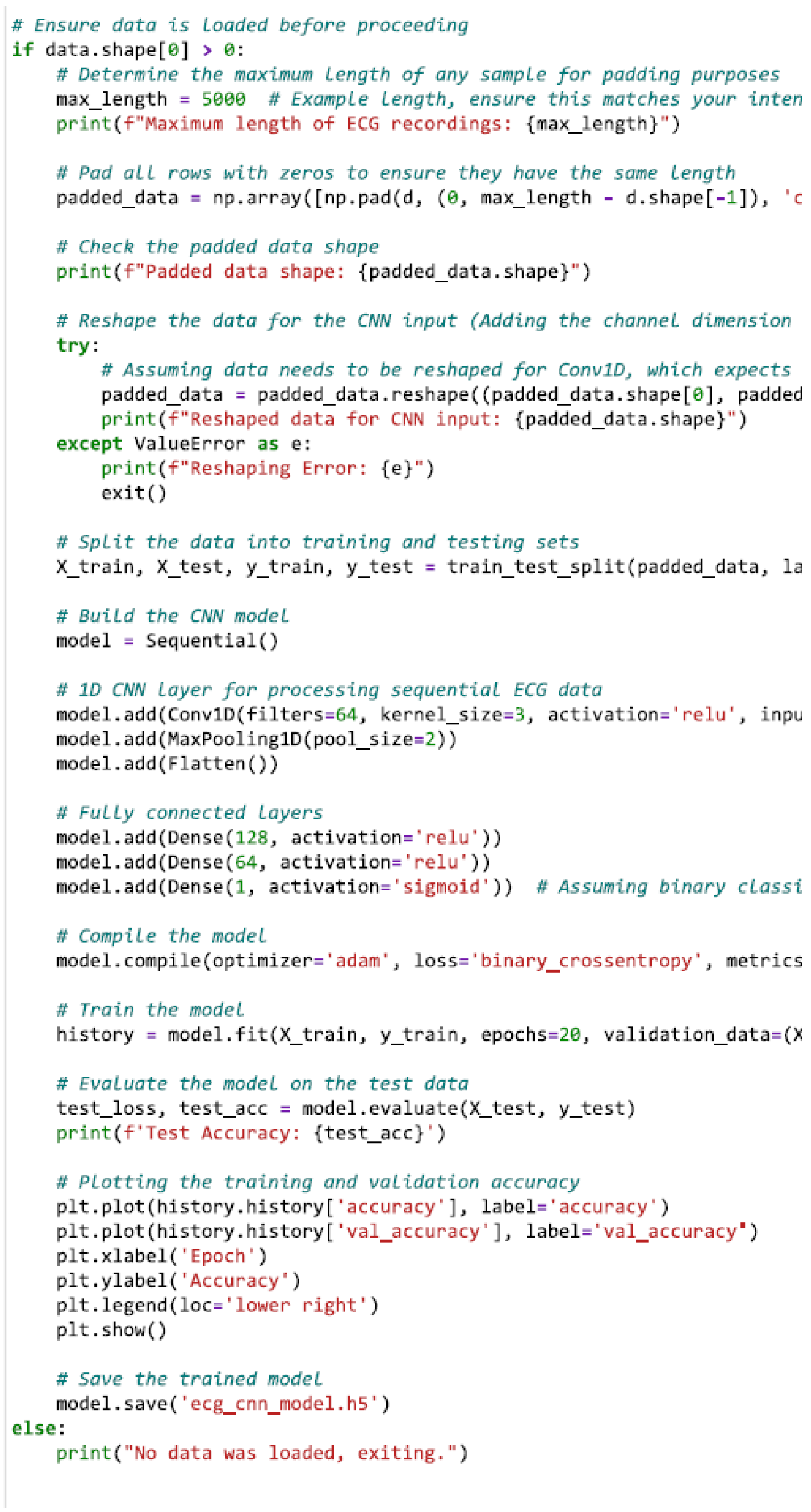
The Arrhythmia rhythm plot was generated using a time series from an ECG signal dataset specifically for arrhythmia. The provided function, `load_ecg_data_from_folders`, is crafted to facilitate the loading of ECG data stored in MATLAB files within a structured folder hierarchy. It takes the path to a root folder as input and systematically explores all nested directories and files. For each `.mat` file it encounters, the function tries to load the file and extract the ECG data, which is presumed to be under the `'val'` key. If this key exists, the ECG data is retrieved and any extra dimensions are removed using `squeeze()`, and the data is then appended to a list. A placeholder label (currently a random binary value) is also appended to a separate list. The function is designed to handle errors during the loading process and will alert the user if the expected data key is not found. Once all files are processed, the function converts the accumulated lists of data and labels into NumPy arrays, making them ready for subsequent analysis or machine learning tasks. This approach is particularly useful for organizing and preparing ECG data for further processing.



```
# Function to load ECG data from the folders
def load_ecg_data_from_folders(root_folder):
    all_data = []
    all_labels = []

    # Traverse through the root folder and subfolders
    for subdir, dirs, files in os.walk(root_folder):
        print(f"Checking folder: {subdir}")
        for file in files:
            if file.endswith('.mat'):
                file_path = os.path.join(subdir, file)
                print(f"Loading MATLAB file: {file_path}")

                try:
                    # Load the MATLAB file
                    mat_data = loadmat(file_path)
                    print(f"Keys in MATLAB file: {mat_data.keys()}")

                    # Assuming 'val' contains ECG data
                    if 'val' in mat_data:
                        ecg_data = mat_data['val'].squeeze()  # Squeeze
                        all_data.append(ecg_data)
                        # Add labels as needed (e.g., based on filename
                        all_labels.append(np.random.randint(2))  # Plac
                    else:
                        print(f"'val' key not found in {file_path}")
                except Exception as e:
                    print(f"Error loading MATLAB file {file_path}: {e}"

    return np.array(all_data), np.array(all_labels)
```

Figure 25 Load ECG data

Next, to build on this process, you would typically focus on evaluating and fine-tuning the model based on its performance. This involves analyzing the accuracy and loss metrics from the training and validation phases to determine if the model is overfitting or underfitting. If necessary, you might adjust hyperparameters, such as the number of epochs, learning rate, or batch size, to improve performance. Additionally, techniques like cross-validation could be employed to ensure the model's robustness. Once the model is optimized, further steps might include applying it to new ECG data for predictions, comparing its performance with other models or methods, and documenting the results. This iterative approach helps refine the model and enhances its effectiveness in real-world applications.



```python
# Ensure data is loaded before proceeding
if data.shape[0] > 0:
    # Determine the maximum length of any sample for padding purposes
    max_length = 5000  # Example length, ensure this matches your inten
    print(f"Maximum length of ECG recordings: {max_length}")

    # Pad all rows with zeros to ensure they have the same length
    padded_data = np.array([np.pad(d, (0, max_length - d.shape[-1]), 'c

    # Check the padded data shape
    print(f"Padded data shape: {padded_data.shape}")

    # Reshape the data for the CNN input (Adding the channel dimension
    try:
        # Assuming data needs to be reshaped for Conv1D, which expects
        padded_data = padded_data.reshape((padded_data.shape[0], padded
        print(f"Reshaped data for CNN input: {padded_data.shape}")
    except ValueError as e:
        print(f"Reshaping Error: {e}")
        exit()

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(padded_data, la

    # Build the CNN model
    model = Sequential()

    # 1D CNN layer for processing sequential ECG data
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', inpu
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))  # Assuming binary classi

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics

    # Train the model
    history = model.fit(X_train, y_train, epochs=20, validation_data=(X

    # Evaluate the model on the test data
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Test Accuracy: {test_acc}')

    # Plotting the training and validation accuracy
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')
    plt.show()

    # Save the trained model
    model.save('ecg_cnn_model.h5')
else:
    print("No data was loaded, exiting.")
```

Figure 26 Process, train and display results

The graph, shown in Figure 25, shows the training and validation accuracy the model over 20 epochs. The training accuracy (blue line) and the validation accuracy (orange line) both fluctuate between 49.4% and 50.5%, which suggests overfitting.

```
Test Accuracy: 0.5065883994102478
```

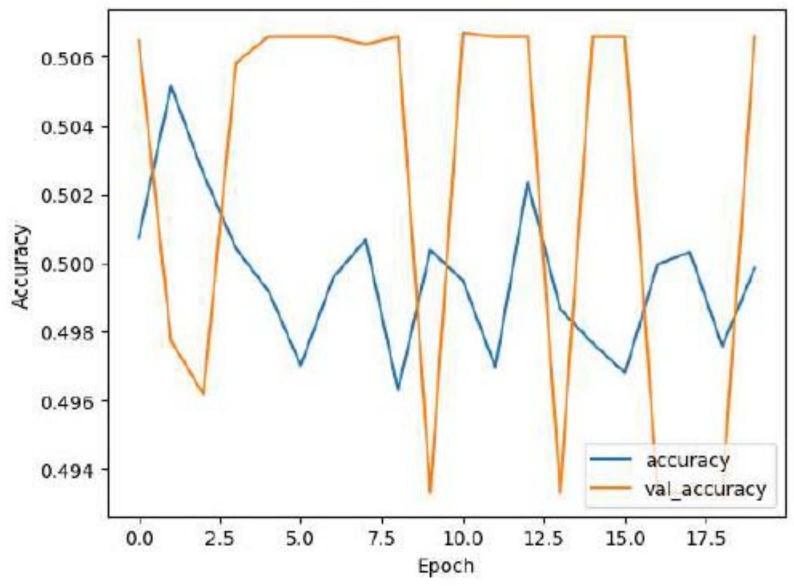
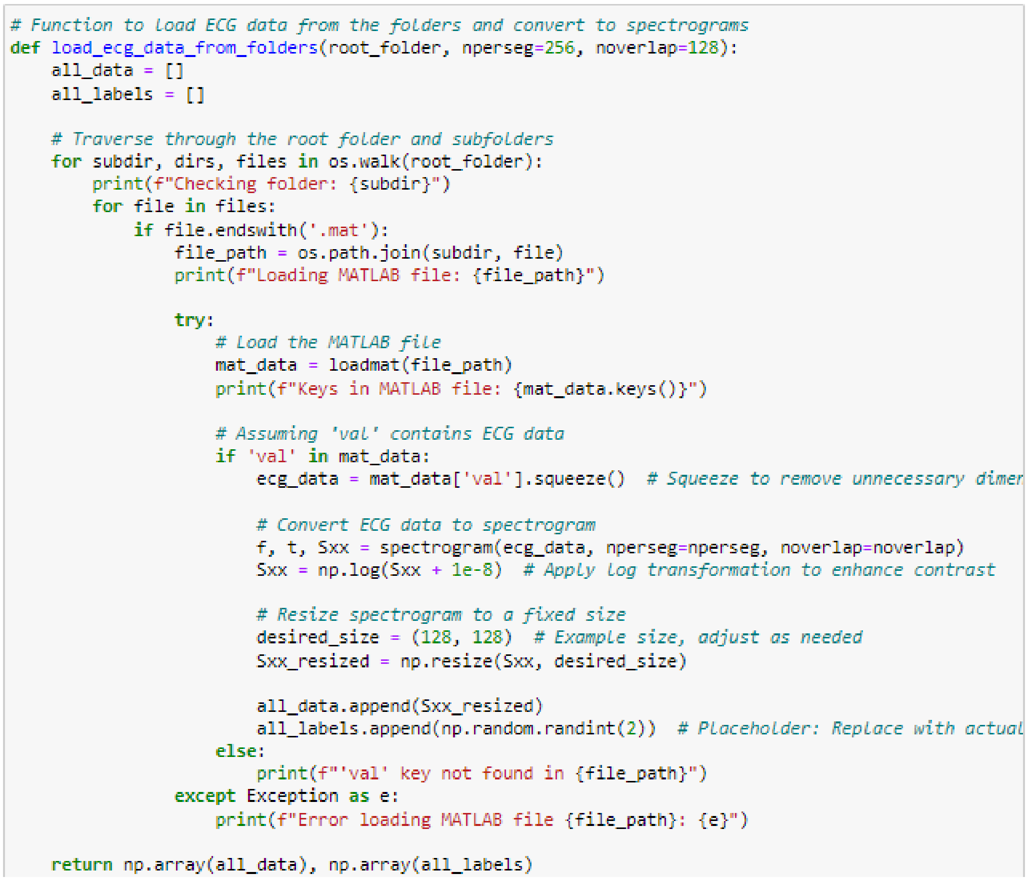


Figure 27 Time series Arrhythmia Results Graph

Spectrogram

For the Spectrogram process, a similar process to the time series Arrhythmia is followed with an additional step of first converting the time series data into images then processing and training the data. The spectrogram conversion process begins with loading ECG data from MATLAB files, specifically extracting the time series data from the `'val'` key. This data is then transformed into a spectrogram using the Short-Time Fourier Transform (STFT), which captures how the frequency content of the ECG signal changes over time. The `spectrogram` function is used with parameters like `nperseg` and `noverlap` to define segment length and overlap. The resulting spectrogram is enhanced by applying a logarithmic transformation to improve contrast and make variations more discernible. To ensure uniform input dimensions for further analysis, the spectrogram is resized to a fixed size, such as 128x128 pixels. Finally, the resized spectrograms, along with placeholder labels, are stored in arrays ready for use in machine learning models. This approach enables the use of image-based techniques for analyzing ECG signals by converting them into a visual format that captures both temporal and frequency information.



```python
# Function to load ECG data from the folders and convert to spectrograms
def load_ecg_data_from_folders(root_folder, nperseg=256, noverlap=128):
    all_data = []
    all_labels = []

    # Traverse through the root folder and subfolders
    for subdir, dirs, files in os.walk(root_folder):
        print(f"Checking folder: {subdir}")
        for file in files:
            if file.endswith('.mat'):
                file_path = os.path.join(subdir, file)
                print(f"Loading MATLAB file: {file_path}")

                try:
                    # Load the MATLAB file
                    mat_data = loadmat(file_path)
                    print(f"Keys in MATLAB file: {mat_data.keys()}")

                    # Assuming 'val' contains ECG data
                    if 'val' in mat_data:
                        ecg_data = mat_data['val'].squeeze()  # Squeeze to remove unnecessary dimer

                        # Convert ECG data to spectrogram
                        f, t, Sxx = spectrogram(ecg_data, nperseg=nperseg, noverlap=noverlap)
                        Sxx = np.log(Sxx + 1e-8)  # Apply log transformation to enhance contrast

                        # Resize spectrogram to a fixed size
                        desired_size = (128, 128)  # Example size, adjust as needed
                        Sxx_resized = np.resize(Sxx, desired_size)

                        all_data.append(Sxx_resized)
                        all_labels.append(np.random.randint(2))  # Placeholder: Replace with actual
                    else:
                        print(f"'val' key not found in {file_path}")
                except Exception as e:
                    print(f"Error loading MATLAB file {file_path}: {e}")

    return np.array(all_data), np.array(all_labels)
```

Figure 28 Spectrogram conversion Arrhythmia

As seen in Figure 28, a test accuracy of approximately 0.498 indicates that the model is performing almost as if it were randomly guessing, which could stem from several issues. The model might be too simplistic or not well-suited to capture the complexities of the ECG data, possibly due to an inadequate architecture. If the training data is noisy, improperly labeled, or not representative, the model will struggle to learn meaningful patterns. Additionally, if placeholder labels were used instead of accurate ones, the model would not be learning from the correct data. Issues in data preprocessing, such as incorrect padding or resizing, and suboptimal hyperparameters could also contribute to poor performance. It's crucial to verify that the spectrogram parameters and preprocessing steps are appropriate for the data. To improve performance, consider using accurate labels, experimenting with more complex models, tuning hyperparameters, and ensuring that your data preprocessing effectively captures relevant features.

```
Test Accuracy: 0.4979515075683594
```

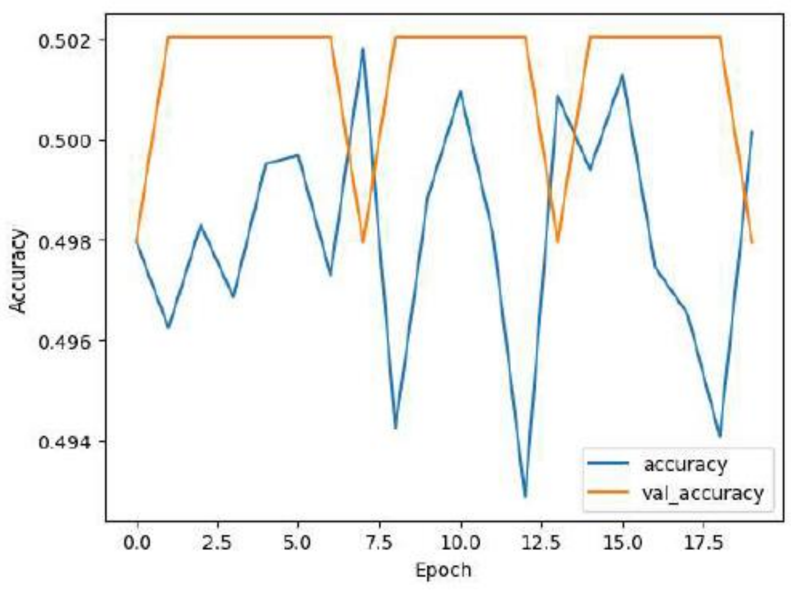


Figure 29 Spectrogram results Arrhythmia

5.3.3 Discussion of Errors and Limitations

Modelling Limitations: Several limitations were identified in these preliminary results. For example, the time series plots provide clear visual insights but may miss subtle variations that are better captured in the frequency domain. Spectrograms, while useful, may also have limitations due to their resolution, potentially missing higher-frequency components critical for certain types of arrhythmias.

Critical Analysis: The preliminary results suggest that while the current approach effectively distinguishes between normal and arrhythmic signals, there are opportunities to improve accuracy. Enhancing data preprocessing techniques, such as noise filtering and baseline correction, could reduce errors and improve model reliability. Additionally, exploring alternative time-frequency representations might yield better insights into the underlying patterns of ECG signals.

References

Electrocardiogram (ECG or EKG) - Mayo Clinic. (2024, April 2). https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983

Aboulhosn, R. (2020, May 10). The Heart's Conduction System. Geeky Medics. https://geekymedics.com/the-hearts-conduction-system/

Rawshani, A., MD PhD, & Rawshani, A., MD PhD. (2023, June 25). ECG interpretation: Characteristics of the normal ECG (P-wave, QRS complex, ST segment, T-wave). Cardiovascular Education. https://ecgwaves.com/topic/ecg-normal-p-wave-qrs-complex-st-segment-t-wave-j-point/

Heart all a-flutter? (2024, April 2). Heart Foundation NZ. https://www.heartfoundation.org.nz/your-heart/heart-conditions/arrhythmia

Klein , E. (2022, March 28). *Everything You Want to Know About Arrhythmia*. Health Line. https://healthline.com/health/arrhythmia

What is atrial fibrillation? (2024, April 2). Heart Foundation NZ. https://www.heartfoundation.org.nz/your-heart/heart-conditions/atrial-fibrillation/

Potter, L. (2024, April 16). Understanding an ECG | ECG interpretation | Geeky Medics. Geeky Medics. https://geekymedics.com/understanding-an-ecg/

R. B. Northrop, Non-Invasive Instrumentation and Measurement in Medical Diagnosis, Second Edition. Taylor & Francis Group, 2017.

Ebrahimi, Z., Loni, M., Daneshtalab, M., & Gharehbaghi, A. (2020). A review on deep learning methods for ECG arrhythmia classification. *Expert Systems With Applications: X*, *7*, 100033. https://doi.org/10.1016/j.eswax.2020.100033

Ansari, Y, Mourad, O, Qaraqe, K, & Serpedin E, (2023) Deep learning for ECG Arrhythmia detection and classification: an overview of progress for period 2017–2023. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10542398/

Semenoglou, A., Spiliotis, E., & Assimakopoulos, V. (2023b). Image-based time series forecasting: A deep convolutional neural network approach. *Neural Networks*, *157*, 39–53. https://doi.org/10.1016/j.neunet.2022.10.006

Homenda, W., Jastrzębska, A., Pedrycz, W., & Wrzesień, M. (2024) Time series classification with their image representation. https://www.sciencedirect.com/science/article/pii/S0925231223013371

Alzubaidi, L., Zhang, J., Humaidi, A. J., Duan, Y., Santamaría, J., Fadhel, M. A., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data, 8(1), 1-74. https://doi.org/10.1186/s40537-021-00444-8

Indolia, S., Goswami, A. K., Mishra, S., & Asopa, P. (2018). Conceptual understanding of Convolutional Neural Network- a deep learning approach. Procedia Computer Science, 132, 679–688. https://doi.org/10.1016/j.procs.2018.05.069

Sheikh, A. (2023, June 25). *Understanding Convolutional Neural networks (CNNs): best for image classification*. https://www.linkedin.com/pulse/understanding-convolutional-neural-networks-cnns-best-al-ameen

Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., & Miao, Y. (2021). Review of image classification algorithms based on convolutional neural networks. Remote Sensing, 13(22), 4712. https://doi.org/10.3390/rs13224712

Sharma, N., Jain, V., & Mishra, A. (2018). An analysis of convolutional neural networks for image classification. Procedia Computer Science, 132, 377–384. https://doi.org/10.1016/j.procs.2018.05.198