A

Dissertation Report on

Comparing Traditional Machine Learning Algorithms and Deep Learning Networks for ECG-Based Disease Classification: A Study Using the MIT-BIH Arrhythmia Dataset

Submitted

in partial fulfilment of the requirements for the degree of

Bachelor of Science

in

Physics (Hons.)

by

Mr. Archisman Chakraborti

Roll No. 167

Registration No. A01-20-300-4-06-0167

Under the Supervision of

Dr. Indranath Chaudhuri

Department of Physics

St. Xavier's College (Autonomous), Kolkata

Calcutta University

2022-2023

St. Xavier's College(Autonomous), Kolkata

(University of Calcutta)

CERTIFICATE

This is to certify that, Mr. Archisman Chakraborti (Roll No-167) has successfully completed the dissertation work and submitted dissertation report on ("Title: Comparing Traditional Machine Learning Algorithms and Deep Learning Networks for ECG-Based Disease Classification: A Study Using the MIT-BIH Arrhythmia Dataset") for the partial fulfillment of the requirement for the degree of Bachelor of Science in Physics (Hons.) from the Department of Physics, as per the rules and regulations of St. Xavier's College, Kolkata.

Date: 5th April, 2023

Place: St. Xavier's College(Autonomous), Kolkata

Dr. Indranath Chaudhuri

Name and Sign of Supervisor

DECLARATION

I declare that this report reflects my thoughts about the subject in my own words. I have sufficiently cited and referenced the original sources, referred or considered in this work. I have not misrepresented or fabricated or falsified any idea/data/fact/source in this my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute.

Place: St. Xavier's College (Autonomous), Kolkata

Name of Student: Mr. Archisman Chakraborti

Signature of Student :

Date: 5th April, 2023

Roll No: 167

ACKNOWLEDGEMENTS

I must mention several individuals and organizations that were of enormous help in the development of this work. Professor Dr. Indranath Chaudhuri, my supervisor, philosopher and personality with a midas touch encouraged me to carry this work. His continuous invaluable knowledgably guidance throughout the course of this study helped me to complete the work up to this stage and hope will continue in further research.

I am very grateful to Ms. Rama Mishra for her positive cooperation and kind help during the period of work with her.

In addition, the energetic and competitive atmosphere of the Department had much to do with this work. I acknowledge with thanks to faculty, teaching and non-teaching staff of the department, Central library and Colleagues.

Last but not the least, I am indebted to my father, Mr. Kousik Chakraborti and my mother, Mrs. Papiya Chakraborti, for their constant support for this work in all aspects

Place: St. Xavier's College(Autonomous), Kolkata

Name of Student: Mr. Archisman Chakraborti

Date: 5th April, 2020

Roll No: 167

ABSTRACT

This dissertation explores the application of artificial intelligence for electrocardiogram (ECG) processing and disease classification. The study focuses on comparing traditional machine learning algorithms and modern deep learning networks, specifically convolutional neural networks (CNNs), for their efficiency and accuracy in classifying diseases based on ECG signals. The research uses the MIT-BIH Arrhythmia dataset and applies traditional filtering methods, including convolution filtering, band-pass filters, and Gustafson's edge correction methods, to preprocess the data. Mathematical features like mean, kurtosis, and skewness, as well as biological features like R-R interval, PR Interval, ST Interval, and Q-S Wave height, are extracted to implement traditional machine learning models. For the deep learning models, both CNNs and artificial neural networks (ANNs) with various inner structures and layers are used to make predictions. Cross Entropy loss is employed as the loss function, and Adaptive Moment Estimation is used as the optimisation function for the neural networks. For the machine learning models, a simple Jaccard Index is used as this is a classification problem. The study compares the efficiency and accuracy of all these methods to develop a robust method for ECG-based disease classification. Most of the work is done using Python as the programming language. However, some work is being done using JavaScript and HTML to design illustrations. To validate the model's effectiveness, the study tests it on a live subject and gets the results validated by practising physicians. The study aims to contribute to the development of a more accurate and efficient method for ECG-based disease classification, which can ultimately lead to improved patient care and diagnosis.

Keywords: Deep Learning, machine Learning, Convolution Neural Networks(CNNs), Artificial Neural Networks (ANNs), Electrocardiogram(ECG)

Contents

List of Figures

NOMENCLATURE

α, β, γ Represent angles or other geometric measurements.

δ Often used to represent change, such as in reaction rates or temperature differences.

μ, σ Represent mean and standard deviation in statistical analyses.

N The symbol for the number of particles or molecules in a system, often used in thermodynamics.

x, y, z Cartesian coordinates of space

ABBREVIATIONS

ECG	Electrocardiogram
TP	True Positive
TN	True Negative
FP	False positive
FN	False Negative
MIT	Massachusetts Institute of Technology
BIH	Beth Israel Hospital
CNN	Convolution Neural Network
ANN	Artificial Nural Network

Chapter 1

The Dataset

The MIT-BIH Arrhythmia dataset [3] is a widely used public dataset consisting of electrocardiogram (ECG) recordings of 48 different patients, each with a duration of approximately 30 minutes.

1.1 Origin

The dataset was created by the Massachusetts Institute of Technology (MIT) and the Beth Israel Hospital in Boston and released in 1980. The dataset was obtained in various parts from https://physionet.org [4] and https://www.kaggle.com

1.2 What it has

The ECG recordings in the dataset were sampled at a frequency of 360 Hz and include annotations indicating the presence and type of arrhythmia present in the recording. The dataset includes a total of 109,446 individual heartbeats, with 25 different arrhythmia types represented.

Twenty-three recordings were chosen at random from a set of 4000 24-hour ambulatory ECG recordings collected from a mixed population of inpatients (about 60%) and outpatients (about 40%) at Boston's Beth Israel Hospital; the remaining 25 recordings were selected from the same set to include less common but clinically significant arrhythmias that would not be well-represented in a small random sample.

1.3 Usage

The MIT-BIH Arrhythmia dataset has been used extensively in the development and evaluation of algorithms for arrhythmia detection and classification. It has also been used for research in signal

processing, machine learning, and other related fields. Many open-source software tools and libraries have been developed to facilitate access and analysis of this dataset.

Overall, the MIT-BIH Arrhythmia dataset has played a crucial role in advancing the field of arrhythmia detection and classification. Its availability and well-established annotation make it an essential resource for researchers and developers working in this area.

Chapter 2

Visualizing the Dataset

In this chapter, we will visualize the various mathematical features of our dataset and try to get a better understanding of how to dataset is distributed among the various diseases.

2.1 Percentage distribution of diseases in the dataset

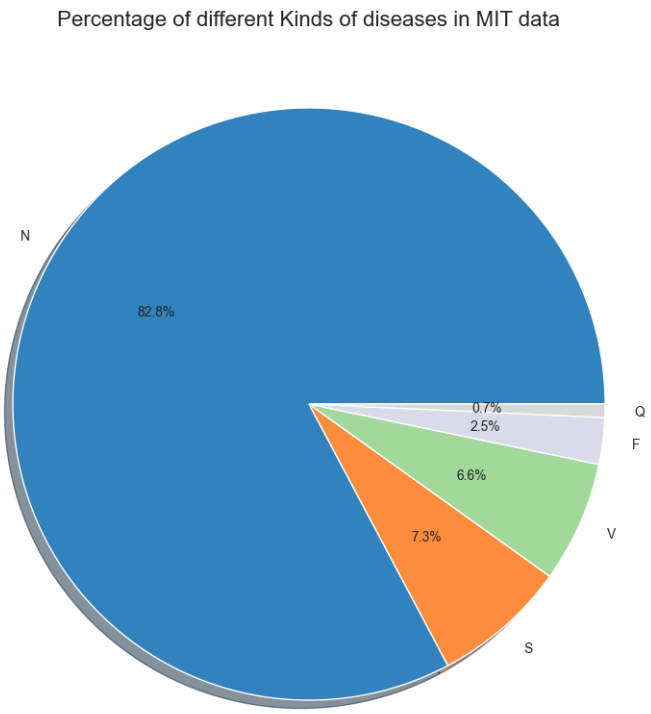


Figure 2.1: Pie Chart showing the distribution of various diseases in the dataset

2.2 Distribution of Mean and Standard Deviation

- We see that the mean values of the ECGs for each disease are not very different from each other.

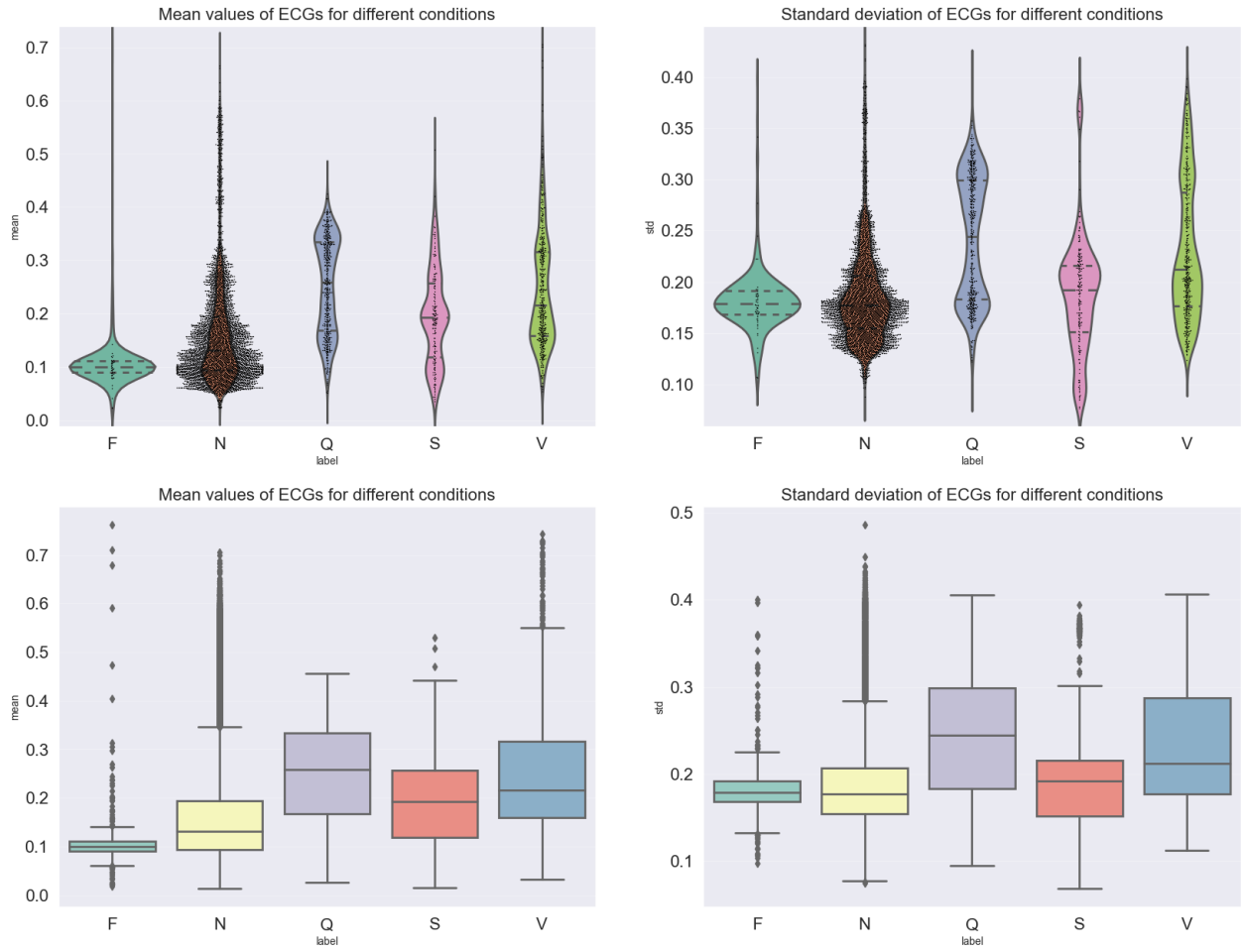


Figure 2.2: Box plots and Violin plots showing the overall mean and standard deviation of various diseases in the dataset

- The shape of the boxplot and violin plot is very unique for "F" in the mean value plots. It has a very low deviation in its mean value for individual ECGs.
- The shape of the boxplot and violin plot is very unique for "F" in the standard deviation plot as well. There is very less deviation in the values of standard deviation for individual ECGs having "F" as the disease.
- There are a lot of outliers in the standard deviation plot for "N". This may be due to the fact that there are a lot of "N" samples in the data and the disease is not very rare or they may pose some problem later in the model.
- F also has some outliers but not too much and mostly on the higher side.
- The standard deviation of most of the data is quite low overall hovering in the 0.2-0.3 rang which is a good sign ans shows uniformity in the ECG samples.
- It may have been useful to plot a swarm plot on the whole dataset as well but there are too many samples and it would have been very difficult to see the swarm plot. So, we have used a subset of

the data for the swarm plot having 8 % of the data.

- As expected, "N" has some outliers and a lot of data.
- However, "S" also has some outliers as well.

2.3 All ECGs together

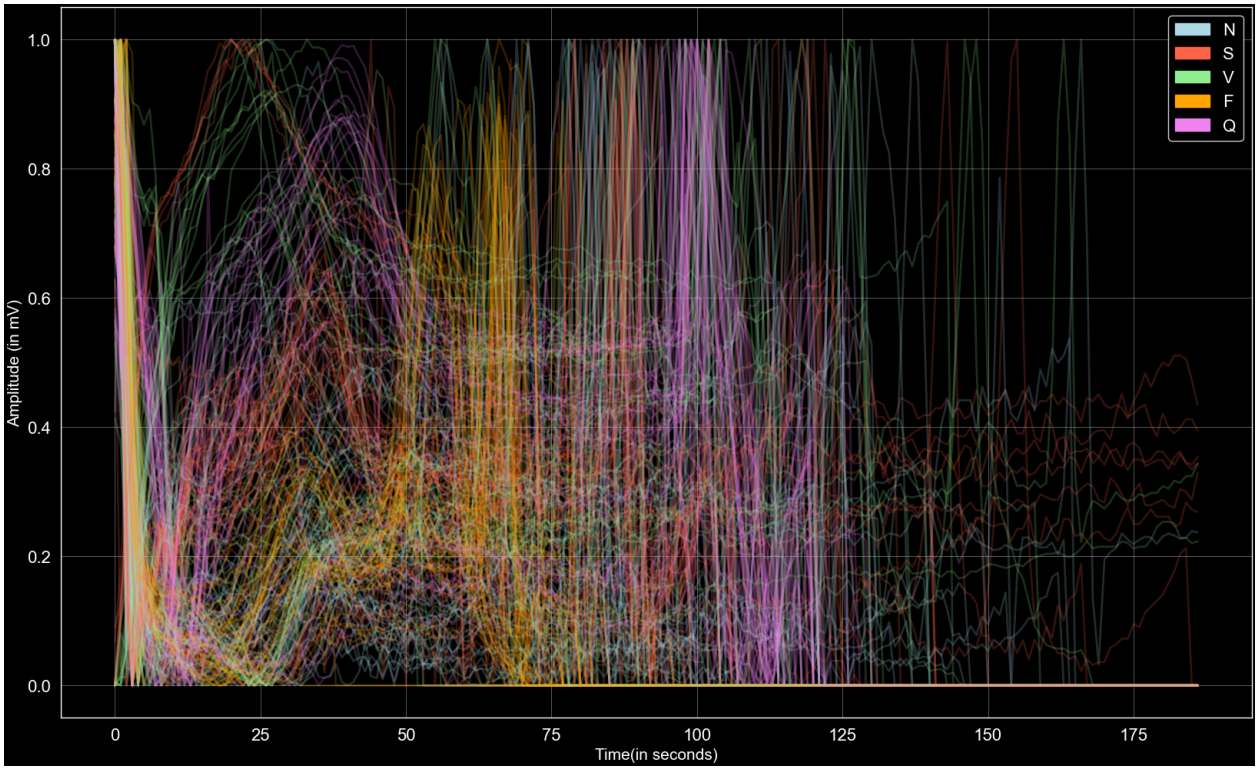


Figure 2.3: All the ECGs shown together

- The above plot may be very crowded(though, we have just plotted the first 50 ECGs for each disease).
- However, it shows us a somewhat of a difference in the distributions of the ECGs for different conditions.
- It may be useful to further analyse all the ECGs as a whole for each disease and see how different features of the ECGs are different for the different diseases.

Chapter 3

Data Filtering

This section covers the methodologies used to reduce noise in the data and ultimately create sharp peaks in the data which will be useful to extract features in the data.

3.1 Convolution filtering

Convolution filtering [5] is a powerful technique used in image processing and computer vision to enhance and transform digital images. It involves the use of a mathematical operation known as convolution to modify an image by applying a filter, or kernel, to each pixel in the image. This process allows for a variety of effects, such as smoothing, sharpening, edge detection, and noise reduction.

The convolution operation involves sliding a small window of values, known as a kernel, over the image. At each location, the values in the kernel are multiplied by the corresponding pixel values in the image, and the resulting products are summed to produce a single output value. This output value is then assigned to the corresponding pixel location in the output image. The size and shape of the kernel determine the specific type of filtering that is performed.

3.1.1 Mathematical basis

Convolution filtering involves the use of a mathematical operation known as convolution [6], which is defined as follows:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

where ‘f‘ and ‘g‘ are functions, and ‘*‘ denotes convolution. This formula represents the process of sliding the function g over the function f, multiplying the values at each overlap, and integrating the result.

In image processing, we apply convolution filtering to 2D images. We use a small matrix of values, known as a kernel, to modify the image. The convolution operation for images can be defined as follows:

$$(I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

where I is the input image, K is the kernel, and (i,j) represents the pixel location in the output image. The summation is over all values of m and n that result in a valid index for the input image.

3.1.2 The Hanning Window

A Hanning window [7], also known as a Hann window or a raised cosine window, is a type of window function that is commonly used in signal processing and digital signal analysis. The Hanning window is designed to reduce the spectral leakage that can occur when a finite-length signal is transformed from the time domain to the frequency domain using the Fourier transform. Spectral leakage is a phenomenon where the energy of a signal leaks into adjacent frequency bins, leading to distortion and inaccuracies in the frequency spectrum.

The function is defined as follows:

$$w(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N - 1}\right)$$

where **n** is the index of the sample, and **N** is the length of the window. The window is applied to the signal by multiplying each sample of the signal by the corresponding sample of the window.

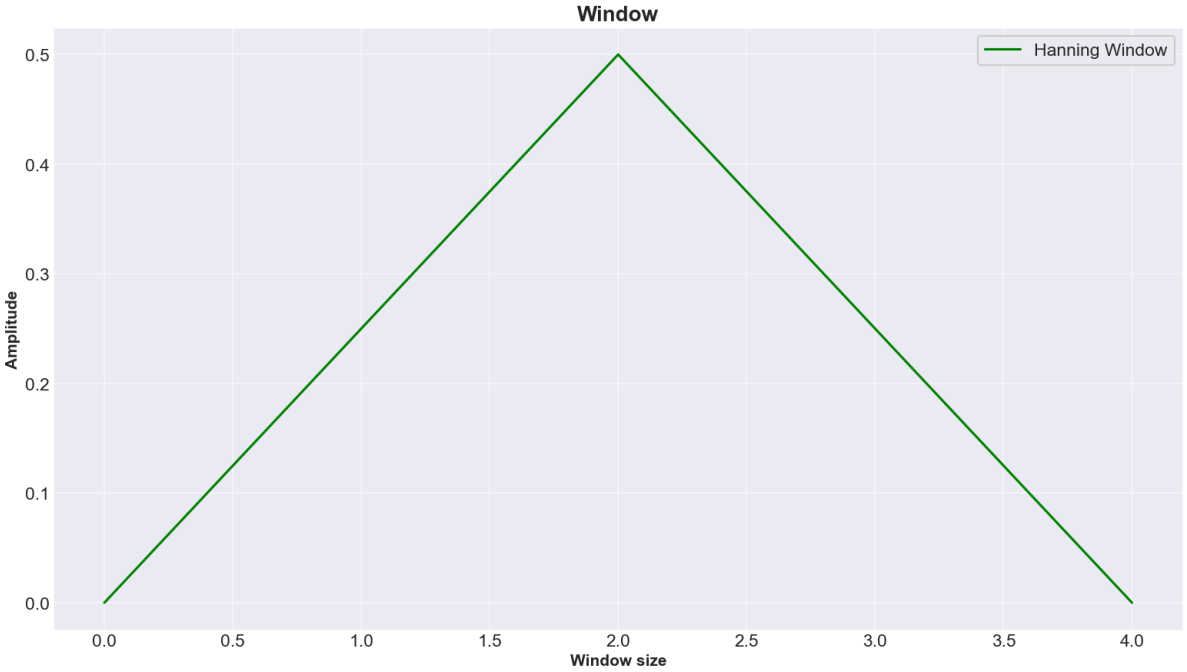


Figure 3.1: Hanning Window. The Amplitude is in millivolts(mV)

Fig 3.2 shows the Hanning Window used in the project. It is a Hanning window of size 5.

Choosing of Window size

The window size for the hanning window was chosen by a **hit and trial method**. Windows of various sizes were selected and the one which best fitted the data in terms of

1. Fitting to the amplitude of the original signal
2. Sharpness of peaks in the signal

was selected and was used for the final Convolution filtering algorithm.

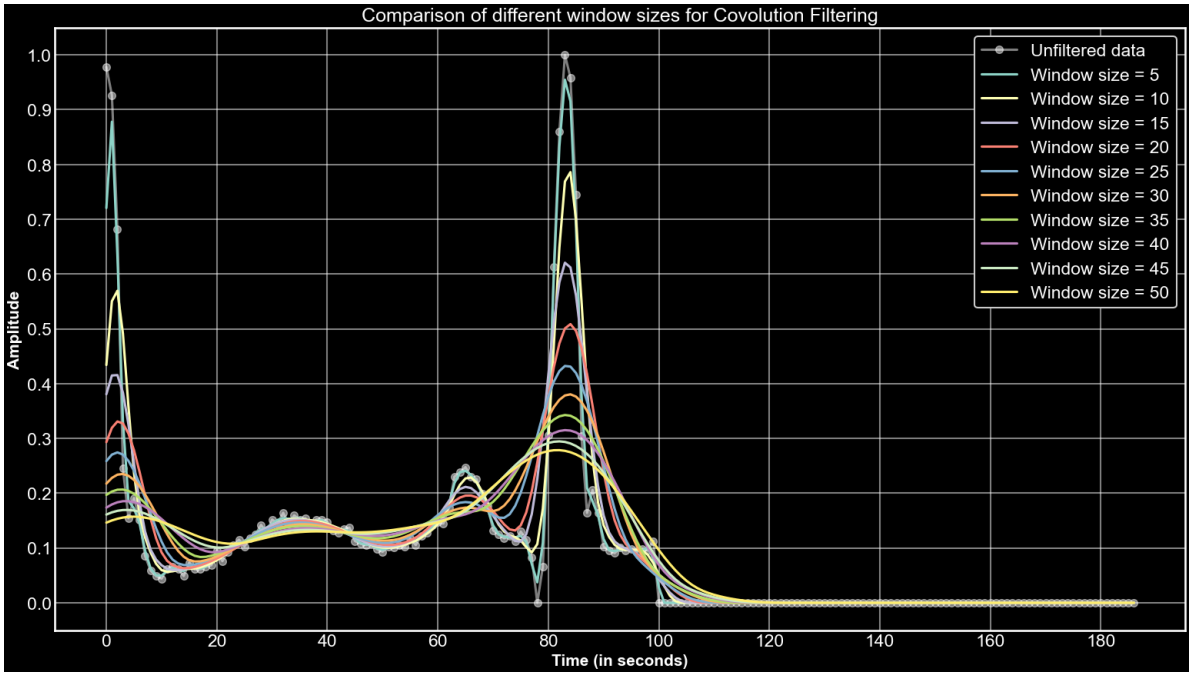


Figure 3.2: Different Window Sizes. The Amplitude is in millivolts(mV)

As can be seen in Fig 3.2, a window size of 5 fits the data best. Increasing the window size, progressively decreases the amplitude of the fitted data.

3.2 Gustaffson's method

Gustafsson's edge filtering method [8] is a popular technique used for image processing and computer vision applications. The technique is designed to reduce image noise while preserving edges, making it useful for applications such as image enhancement, object detection, and segmentation.

The method involves convolving the image with a filter kernel that consists of two parts: a low-pass filter and a high-pass filter. The low-pass filter is used to smooth the image and reduce noise, while the high-pass filter is used to detect and enhance edges.

The low-pass filter is typically a Gaussian filter, which is designed to smooth the image while preserving the overall structure of the image. The high-pass filter, on the other hand, is designed to detect edges by subtracting the smoothed image from the original image. This results in an image that highlights the edges while suppressing noise.

Gustafsson's edge filtering method has several advantages over other edge detection techniques. For example, it is computationally efficient and can be easily implemented using standard image processing libraries. Additionally, it is relatively insensitive to noise and can be used with a wide range of image types, including grayscale and color images.

Let $I(x, y)$ be the input image and $G(x, y)$ be the Gaussian filter with standard deviation σ. The smoothed image $\bar{I}(x, y)$ can be obtained by convolving $I(x, y)$ with $G(x, y)$:

$$\bar{I}(x, y) = I(x, y) * G(x, y) = \iint_{-\infty}^{\infty} I(u, v)G(x - u, y - v)du\,dv \tag{3.1}$$

The high-pass filter $H(x, y)$ can be obtained by subtracting $\bar{I}(x, y)$ from $I(x, y)$:

$$H(x, y) = I(x, y) - \bar{I}(x, y) \tag{3.2}$$

To enhance the edges in the image, we can add $H(x, y)$ back to $\bar{I}(x, y)$ using a scale factor k:

$$E(x, y) = \bar{I}(x, y) + kH(x, y) \tag{3.3}$$

The value of k can be adjusted to control the strength of the edge enhancement. A typical value for k is 1.

The overall edge filtering process can be summarized as follows:

1. Compute the Gaussian filter $G(x, y)$ with standard deviation σ.
2. Convolve the input image $I(x, y)$ with $G(x, y)$ to obtain $\bar{I}(x, y)$.
3. Subtract $\bar{I}(x, y)$ from $I(x, y)$ to obtain $H(x, y)$.
4. Add $kH(x, y)$ back to $\bar{I}(x, y)$ to obtain the enhanced image $E(x, y)$.

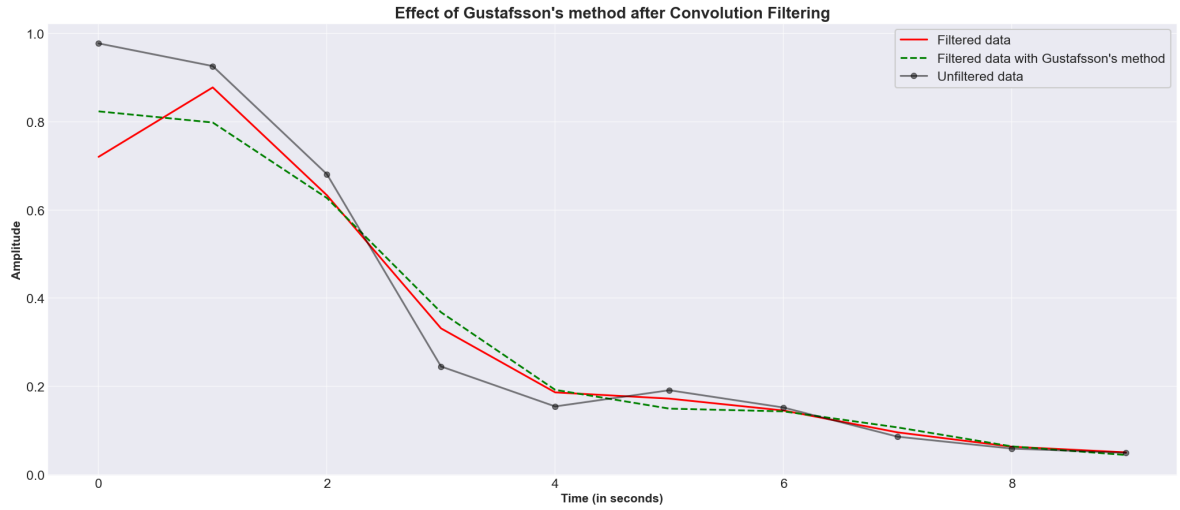


Figure 3.3: Gustaffson's method compared with padding. The amplitude is in millivolts(mV)

Fig 3.4 shows how one of the data record's edges look after filtering with convolution filtering and correcting the edges using the Gustaffson's edge filtering.

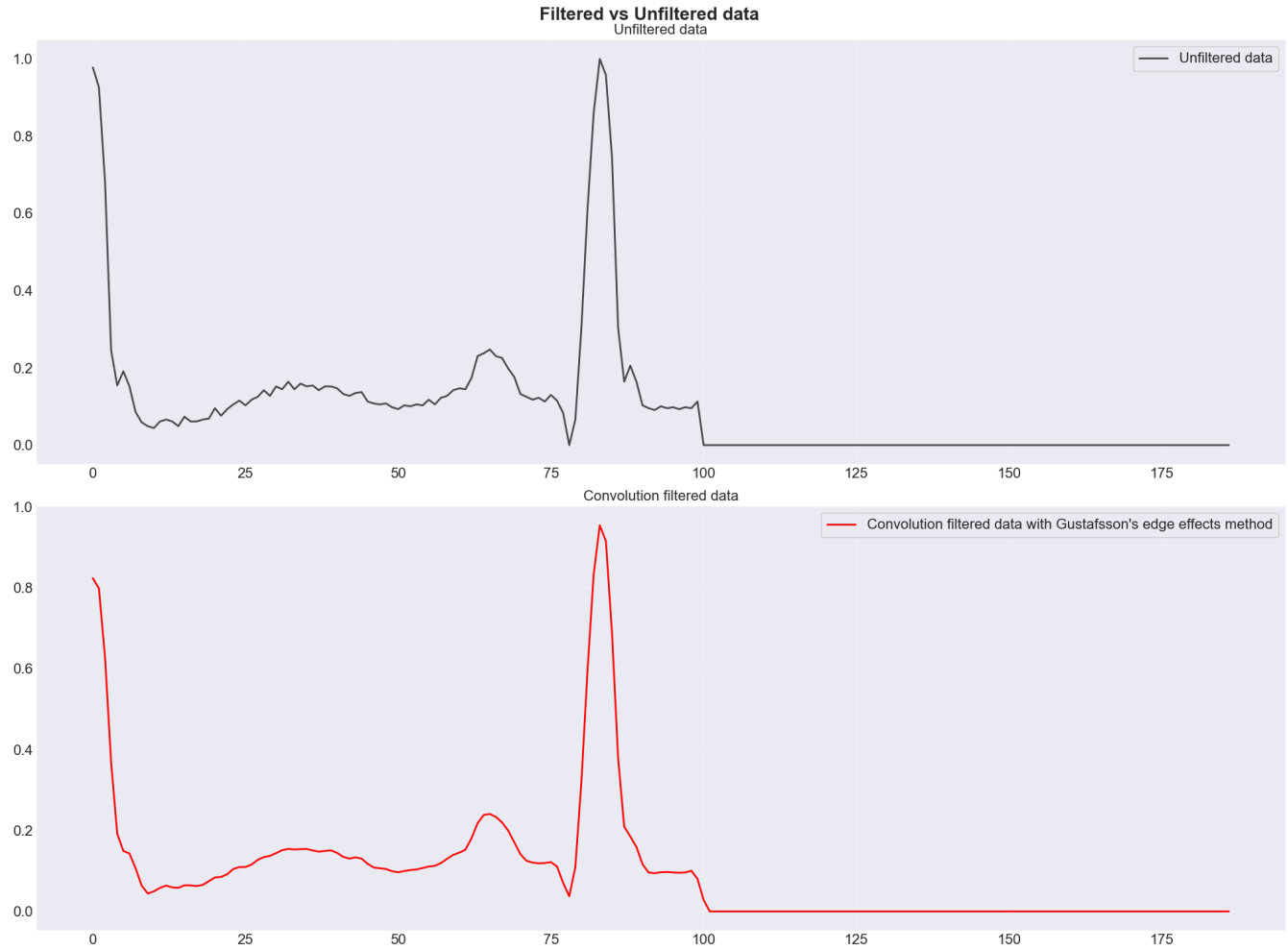


Figure 3.4: Filtered data after convolution filtering and Gustaffson's edge correction vs Unfiltered data

Chapter 4

Feature extraction

Feature extraction [9] is the process of extracting relevant information or features from raw data in order to facilitate machine learning and pattern recognition. In the context of computer vision, feature extraction is often used to identify and extract key characteristics or attributes from an image, such as edges, corners, or color histograms, that can be used for subsequent processing and analysis. The success of many machine learning and computer vision tasks depends heavily on the quality and relevance of the features extracted.

4.1 Types of Features

We have extracted 3 kinds of features from the data.

1. Mathematical Features which do not require a sampling frequency
2. Mathematical features which require a sampling frequency
3. Purely Biological Features

4.1.1 Mathematical features without sampling frequency

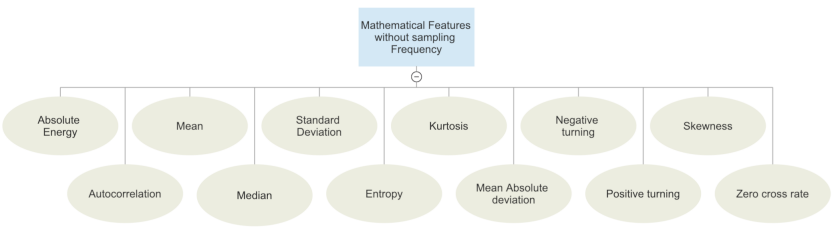


Figure 4.1: Mathematical features which don't require sampling frequency.

1. Absolute energy: The absolute energy E of a signal $x(t)$ can be represented mathematically as:

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

2. Autocorrelation: The autocorrelation function of a signal $x(t)$ can be defined as:

$$R_x(\tau) = \int_{-\infty}^{\infty} x(t)x(t+\tau)dt$$

3. Mean: the mean of a signal is:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

In this formula, $\bar{x}$ represents the mean of the signal, N represents the number of samples in the signal, and x_i represents the $i-th$ sample in the signal.

4. Median: The median of a signal x is a value that divides the signal into two halves of equal length or nearly equal length. If the length of the signal is odd, the median is the middle value of the sorted signal. If the length of the signal is even, the median is the average of the two middle values of the sorted signal.

5. Standard deviation:

$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$

where N is the number of samples in the signal, x_i is the ith sample, and μ is the mean of the signal.

6. Entropy:

$H = -\sum_{i=1}^{n} p_i \log_2 p_i$

where n is the number of unique values in the signal, p_i is the probability of the ith value occurring, and $\log_2$ is the base-2 logarithm.

7. Kurtosis:

$K = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i-\mu}{\sigma}\right)^4 - 3$

where N is the number of samples in the signal, x_i is the ith sample, μ is the mean of the signal, and σ is the standard deviation of the signal.

8. Mean absolute deviation:

$MAD = \frac{1}{N}\sum_{i=1}^{N}|x_i - \mu|$

where N is the number of samples in the signal, x_i is the ith sample, and μ is the mean of the signal.

9. Negative turning:

   $NT = \sum_{i=1}^{N-1} \max(0, x_i - x_{i+1})$

   where N is the number of samples in the signal, and x_i is the ith sample.

10. Positive turning:

    $PT = \sum_{i=1}^{N-1} \max(0, x_{i+1} - x_i)$

    where N is the number of samples in the signal, and x_i is the ith sample.

11. Skewness:

    $S = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^3}{\sigma^3}$

    where N is the number of samples in the signal, x_i is the ith sample, μ is the mean of the signal, and σ is the standard deviation of the signal.

12. Zero-cross rate:

    $ZCR = \frac{1}{2N}\sum_{i=1}^{N-1} |\text{sgn}(x_i) - \text{sgn}(x_{i+1})|$

    where N is the number of samples in the signal, x_i is the ith sample, and $\text{sgn}(x)$ is the sign of x.

4.1.2 Mathematical features with sampling frequency

Sampling frequency = 360Hz

1. Area under the curve:

$$A = \int_{-\infty}^{\infty} |x(t)|dt$$

   where $x(t)$ is the signal.

2. Centroid:

$$C = \frac{\int_{-\infty}^{\infty} f|X(f)|df}{\int_{-\infty}^{\infty} |X(f)|df}$$

   where $X(f)$ is the Fourier transform of the signal and f is the frequency.

3. Fundamental frequency: The fundamental frequency can be estimated using various techniques, such as the autocorrelation method or the HPS (harmonic product spectrum) method.

4. Human range energy:

$$HRE = \int_{20}^{20000} |X(f)|^2 df$$

where $X(f)$ is the Fourier transform of the signal and the limits of integration are chosen to include frequencies in the range of human hearing.

5. Maximum frequency:

$$f_{\max} = \text{argmax}|X(f)|$$

where $X(f)$ is the Fourier transform of the signal and argmax returns the frequency at which $|X(f)|$ is maximum.

6. Maximum power spectrum:

$$P_{\max} = \max|X(f)|^2$$

where $X(f)$ is the Fourier transform of the signal.

7. Median frequency:

$$M = \frac{\int_{-\infty}^{\infty} |X(f)|df}{\int_{-\infty}^{f_m} |X(f)|df}$$

where $X(f)$ is the Fourier transform of the signal, f_m is the frequency at which half of the energy of the signal is contained below f_m.

8. Power bandwidth:

$$BW = \int_{-\infty}^{\infty} |X(f)|^2 df$$

where $X(f)$ is the Fourier transform of the signal.

9. Spectral centroid:

$$C_s = \frac{\sum_{k=1}^{N} f_k |X(k)|^2}{\sum_{k=1}^{N} |X(k)|^2}$$

where $X(k)$ is the discrete Fourier transform of the signal, f_k is the frequency corresponding to the kth bin, and N is the number of bins.

10. Spectral decrease:

$$SD = \frac{1}{N-1} \sum_{k=1}^{N-1} (|X(k+1)| - |X(k)|)$$

where $X(k)$ is the discrete Fourier transform of the signal and N is the number of bins.

11. Spectral distance:

$$SDist = \sqrt{\sum_{k=1}^{N} (f_k - f_0)^2 |X(k)|^2}$$

where $X(k)$ is the discrete Fourier transform of the signal, f_k is the frequency corresponding to the kth bin, and f_0

12. Spectral entropy:
$$H(f) = -\int_{-\infty}^{\infty} P(f) \log P(f) df$$
where $P(f)$ is the power spectral density function.

13. Spectral skewness:
$$\gamma_1(f) = \frac{\int_{-\infty}^{\infty}(f - \mu(f))^3 P(f) df}{[\int_{-\infty}^{\infty}(f - \mu(f))^2 P(f) df]^{3/2}}$$
where $\mu(f)$ is the mean frequency.

14. Spectral kurtosis:
$$\gamma_2(f) = \frac{\int_{-\infty}^{\infty}(f - \mu(f))^4 P(f) df}{[\int_{-\infty}^{\infty}(f - \mu(f))^2 P(f) df]^2} - 3$$

15. Spectral variation:
$$SV = \frac{\sqrt{\int_{-\infty}^{\infty}(P(f) - \bar{P})^2 df}}{\bar{P}}$$
where $\bar{P}$ is the average power spectral density.

16. Total energy of a signal with sampling frequency $f_s = 360$ Hz:
$$E = \sum_{n=0}^{N-1} |x[n]|^2$$
where $x[n]$ is the discrete-time signal and N is the number of samples, given by $N = T \cdot f_s$, where T is the duration of the signal in seconds.

4.1.3 Biological features

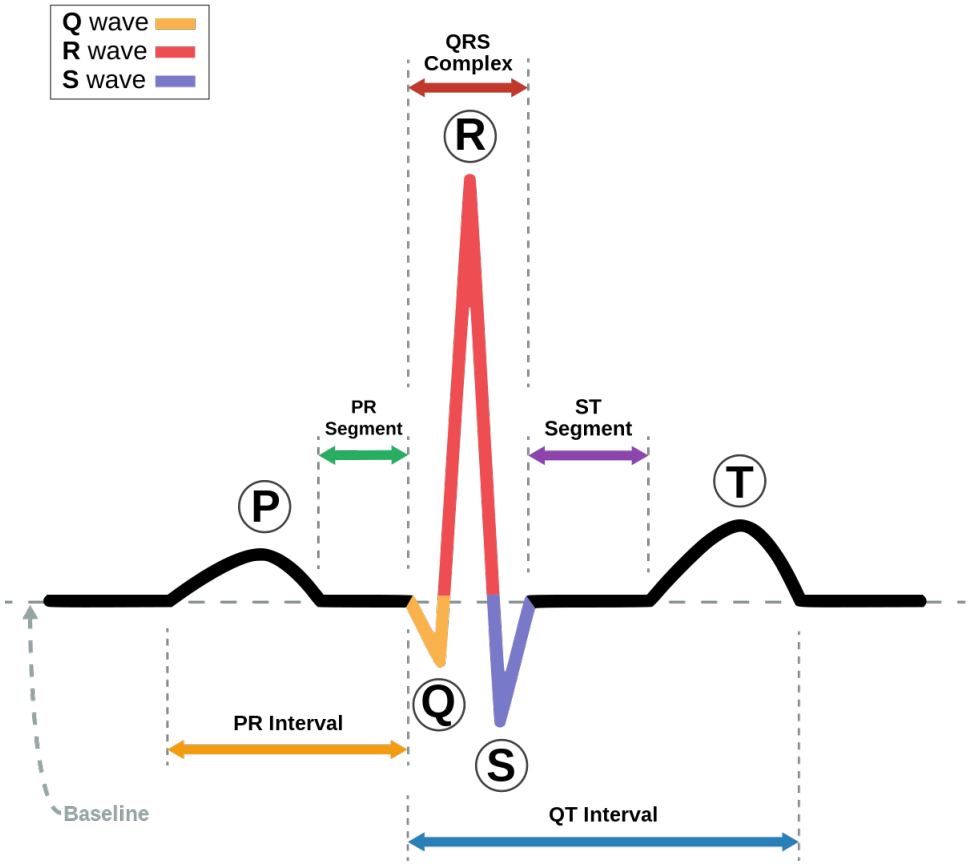


Figure 4.2: Biological Features [2]

1. QRS complex:

$$QRS = Q + R + S$$

where Q, R, and S are the amplitudes of the Q, R, and S waves, respectively.

2. PR segment:

$$PR\ segment = TP - P\ wave$$

where TP is the end of the T wave and P wave is the beginning of the P wave.

3. ST segment:

$$ST\ segment = S\ wave\ end - T\ wave\ beginning$$

where S wave end is the end of the S wave and T wave beginning is the beginning of the T wave.

4. QT interval:

$$QT\ interval = Q\ wave\ beginning - T\ wave\ end$$

where Q wave beginning is the beginning of the Q wave and T wave end is the end of the T wave.

5. PR interval:

$$PR\ interval = P\ wave\ beginning - Q\ wave\ beginning$$

where P wave beginning is the beginning of the P wave and Q wave beginning is the beginning of the Q wave

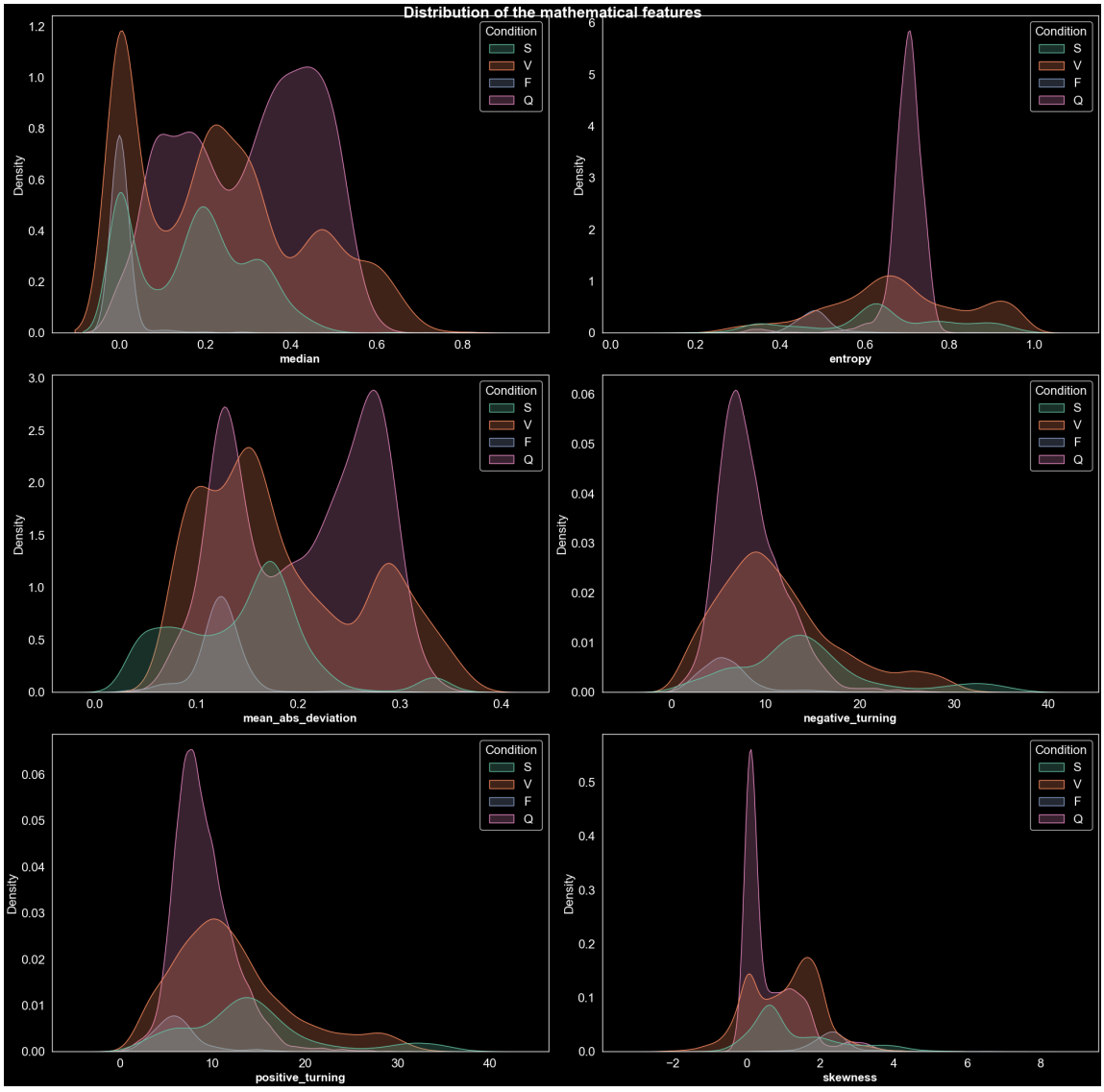


Figure 4.3: Mathematical features without sampling frequency

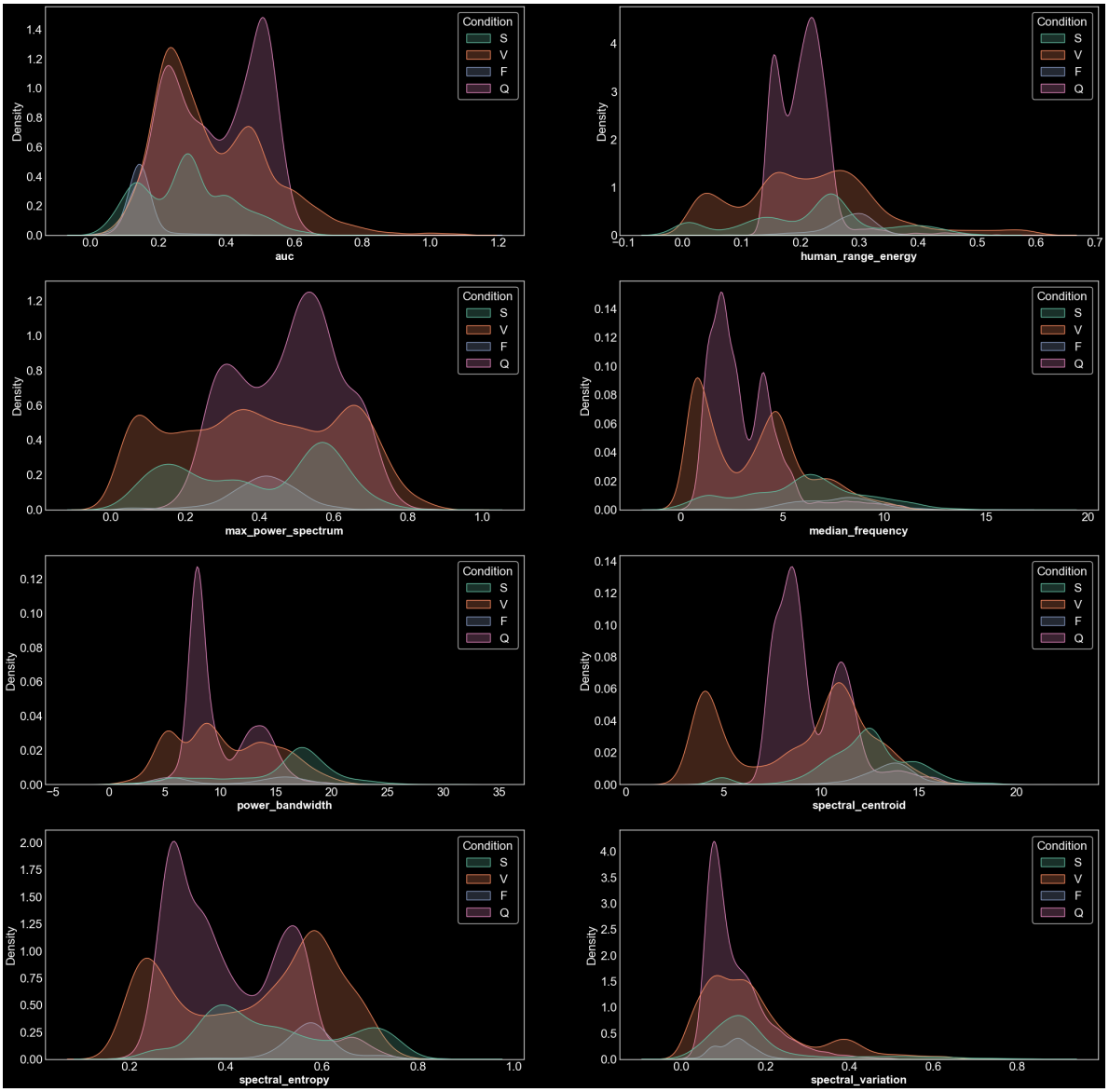


Figure 4.4: Mathematical features with sampling frequency

4.1.4 Comparing Features

To compare features for a machine learning model using the Pearson correlation coefficient [10], we calculate the correlation matrix for all pairs of features using the following formula:

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.1}$$

where $r_{X,Y}$ is the Pearson correlation coefficient between features X and Y, x_i and y_i are the values of features X and Y at observation i, and $\bar{x}$ and $\bar{y}$ are the means of features X and Y, respectively. By examining the correlation matrix, we can identify pairs of features that are highly correlated and potentially redundant, which can be removed to simplify the model and improve its performance.

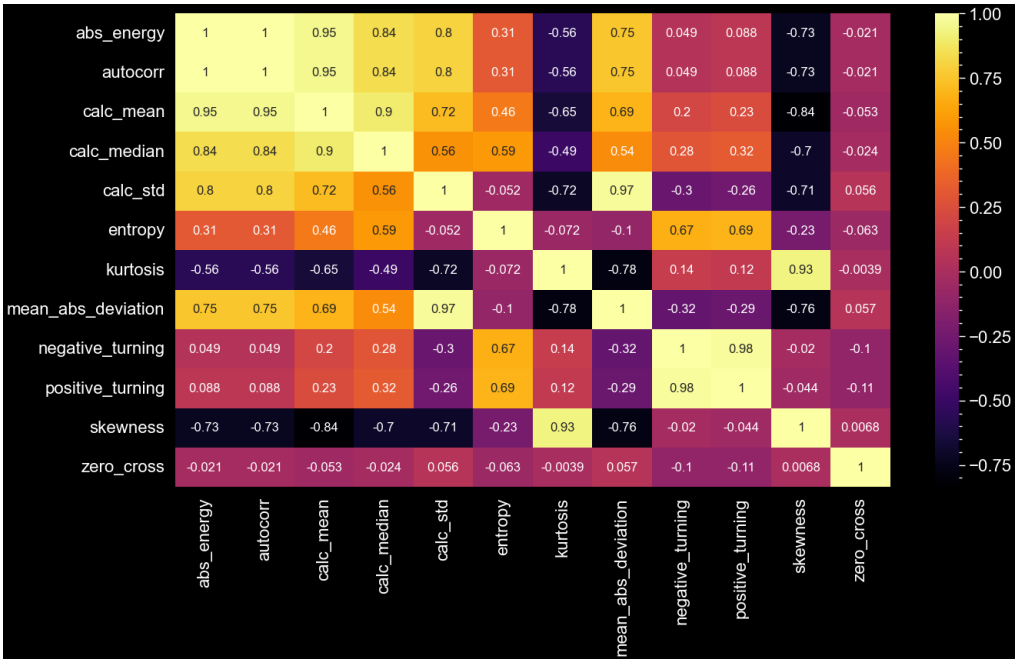


Figure 4.5: Correlation among mathematical features without sampling frequency

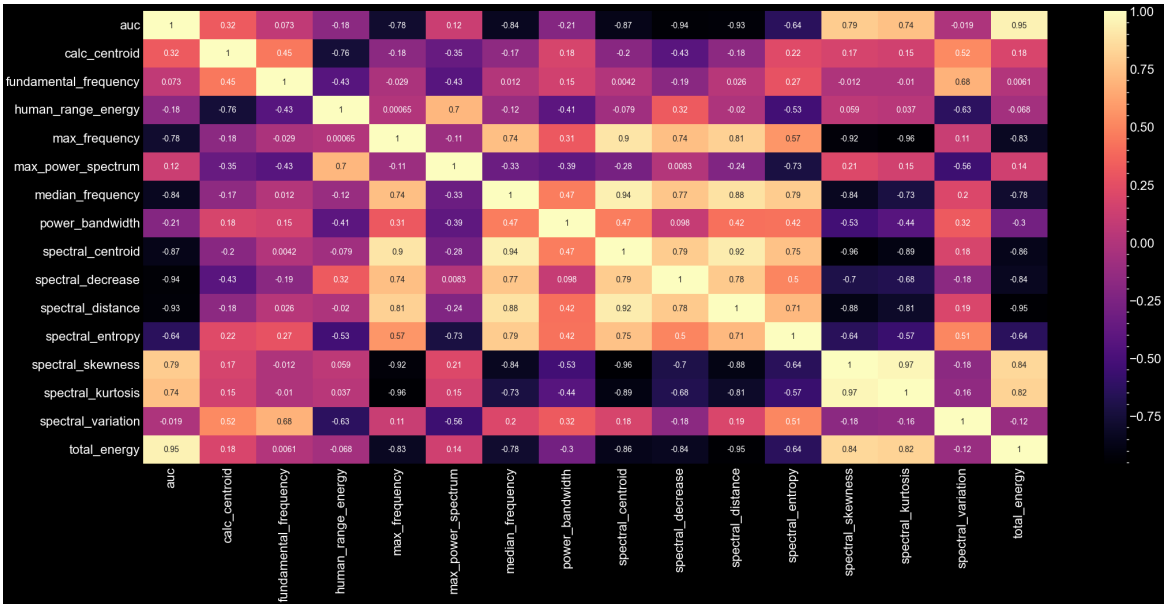


Figure 4.6: Mathematical features with sampling frequency

When the correlation between pairs of features is greater than 0.8, it indicates that they may contain similar information. To avoid redundancy, one of the features was discarded to simplify the model and prevent overfitting.

Chapter 5

Machinery Used

Primary Computer

1. Model : Macbook Air M1
2. CPU : M1 Apple Silicon Chip
3. GPU : MacOS MPS (Metal Performance Shader) 8 cores

Secondary Computer

1. CPU : Intel i7 2020 version
2. GPU : Tesla T4, CUDA Version 10.1

Chapter 6

Machine Learning

6.1 Model Pipeline

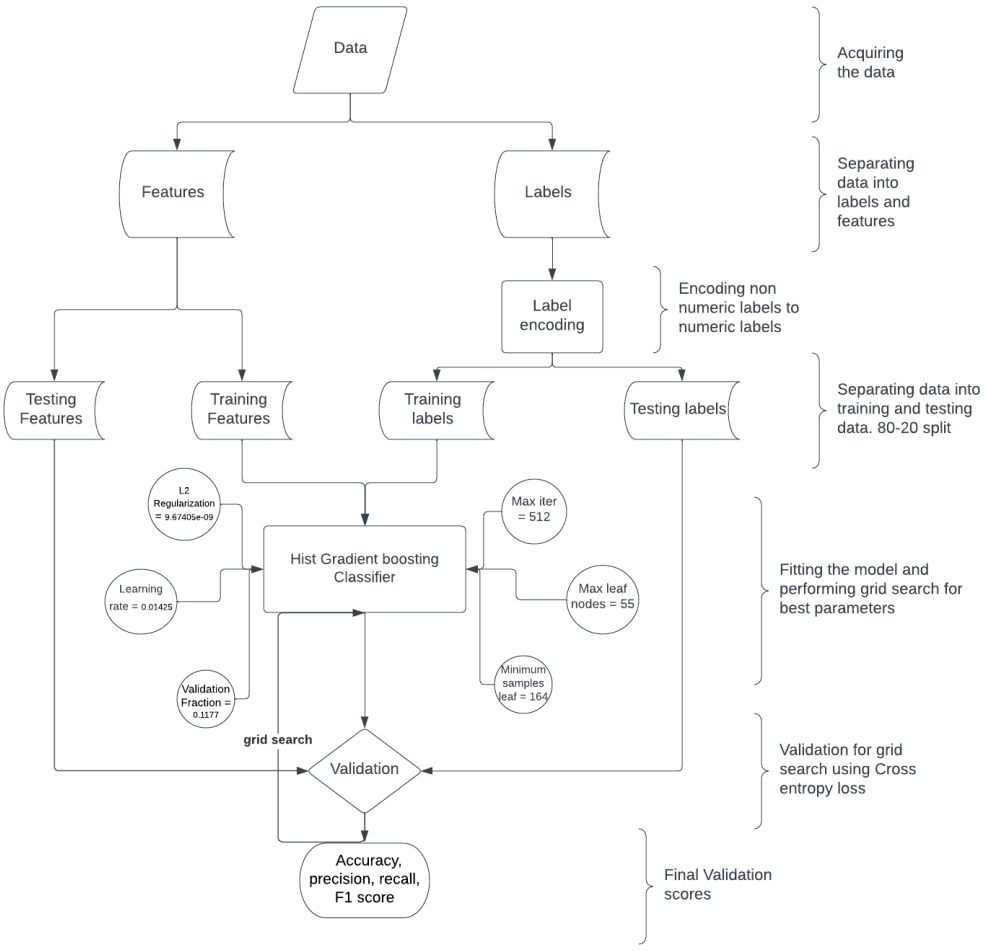


Figure 6.1: Pipeline of Machine Learning Process

6.2 Steps involved

1. **Reading the data**: The features and labels were read together into a single **csv** file.
2. **Separating the data**: The data was separated into labels and features.
3. **Label Encoding**: Label encoding is a technique used in machine learning to convert categorical variables into numerical format. In this technique, each unique category is assigned a numerical value. This is useful for machine learning algorithms that require numerical input, as categorical variables cannot be processed directly. The labels (5 different classes) were properly encoded as (0, 1, 2, 3, 4) respectively.
4. **Splitting data**: The data was separated into training and testing data. 80% of the data was kept for training the model while 20 % was kept for testing and cross validation.
5. **Fitting the data**: The model was fitted using the training data.
6. **Validation**: Accuracy, cross validation score, precision, recall and F1 scores were calculated from the confusion matrix.

6.3 The Model

Histogram Gradient Boosting [1] Classifier is a variant of gradient boosting that is designed specifically for handling histogram-based data. This classifier is often used for classification problems that involve large datasets, as it is computationally efficient and can handle a large number of features.

The basic idea behind Histogram Gradient Boosting Classifier is to iteratively train a sequence of decision trees on the data, where each tree is built to correct the errors of the previous trees. Unlike traditional gradient boosting methods that split the data based on feature values, Histogram Gradient Boosting Classifier partitions the feature space into histograms, and uses the histograms to create a set of candidate splits.

The algorithm works by minimizing a cost function that is defined in terms of the difference between the predicted values and the true labels of the data. The cost function is minimized using a gradient descent algorithm, where each iteration involves training a new decision tree on the data and updating the predicted values based on the output of the tree.

The mathematics of Histogram Gradient Boosting Classifier can be expressed as follows:

Let D be the training dataset, where each sample is represented by a set of n features $x_1, x_2, ..., x_n$ and a label y. Let $f(x)$ be the predicted value of the classifier for a given input x. The goal of the algorithm is to find a function $f(x)$ that minimizes the following cost function:

$$\min_f \sum_{i=1}^{N} L(y_i, f(x_i))$$

where L is a loss function that measures the difference between the predicted values and the true labels. The algorithm then iteratively trains a sequence of decision trees $T_1, T_2, ..., T_m$ to minimize the cost function. The predicted value of the classifier for a given input x is then given by:

$$f(x) = \sum_{i=1}^{m} \gamma_i T_i(x)$$

where γ_i is the learning rate for the i^{th} tree, and $T_i(x)$ is the output of the i^{th} tree for the input x.

The process of training the decision trees involves partitioning the feature space into histograms, and selecting the best split for each histogram based on the gain in the cost function. The histograms are constructed by grouping the feature values into bins, and computing the frequency of the label for each bin. The gain in the cost function is computed as:

$$\Delta L = \frac{1}{2}[\frac{\sum_{i\in I_L} w_i y_i}{\sum_{i\in I_L} w_i + \lambda} + \frac{\sum_{i\in I_R} w_i y_i}{\sum_{i\in I_R} w_i + \lambda} - \frac{\sum_{i\in I} w_i y_i}{\sum_{i\in I} w_i + \lambda}] - \gamma$$

where I is the index set of the samples in the histogram, I_L and I_R are the index sets of the samples in the left and right histograms, w_i is the weight of the i^{th} sample, λ is a regularization parameter, and γ is the learning rate. The algorithm then selects the split with the highest gain, and continues recursively until a stopping criterion is met.

6.3.1 Model Parameters [1]

1. *early stopping*=True
2. *l2 regularization*= $9.674948183980905 \times 10^{-9}$
3. *learning rate*= 0.014247987845444413
4. loss='auto'
5. *max iter*=512
6. *max leaf nodes*=55
7. *min samples leaf* = 164
8. n_iter no change=1
9. *random state*=1

10. *validation fraction*=0.11770489601182355

11. *warm start* = True

6.4 Validation

6.4.1 Confusion Matrix

A confusion matrix [11] is a table that is often used to evaluate the performance of a classification model. It displays the number of correct and incorrect predictions made by the model on a set of test data, compared to the actual outcomes. The matrix consists of four categories: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). These categories are used to calculate different metrics such as accuracy, precision, recall, and F1 score, which can provide insights into the model's performance and help optimize it.

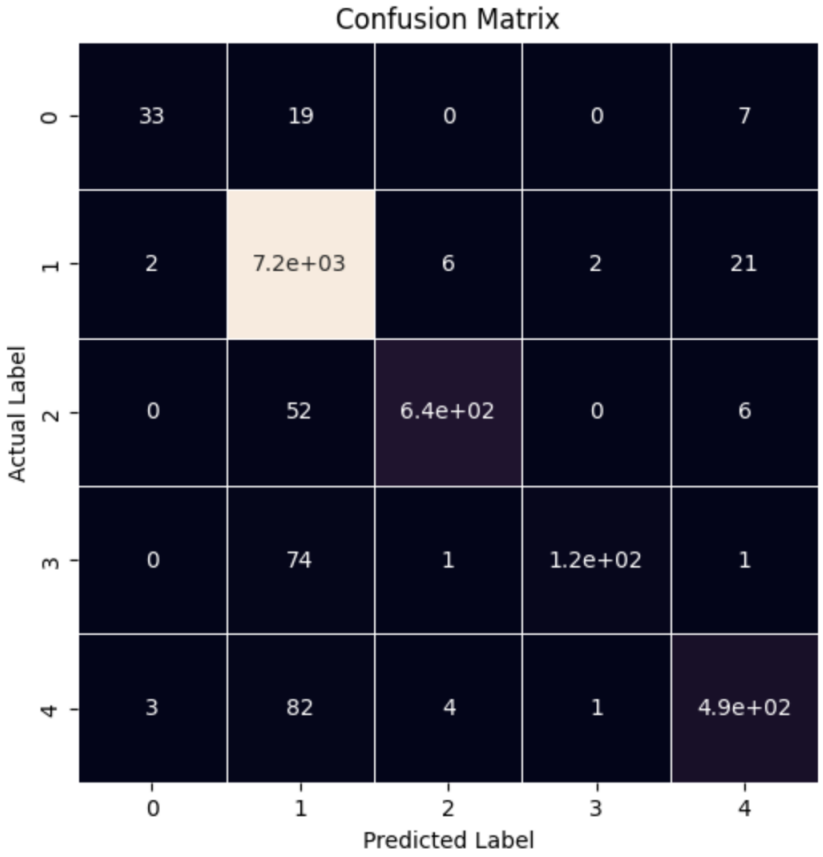


Figure 6.2: Confusion matrix

6.4.2 Accuracy, Precision, Recall, F1 Score

Accuracy, precision, recall, and F1 score [12] are common metrics used to evaluate the performance of a classification model.

Accuracy

Accuracy measures the overall correctness of the predictions made by the model. It is calculated as the ratio of the number of correct predictions to the total number of predictions made by the model, expressed as a percentage. Mathematically, accuracy can be defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP denotes true positives, TN denotes true negatives, FP denotes false positives, and FN denotes false negatives.

Precision

Precision measures the proportion of true positives among all the positive predictions made by the model. It is calculated as the ratio of the number of true positives to the sum of true positives and false positives. Mathematically, precision can be defined as:

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

F1 Score

The F1 score is the harmonic mean of precision and recall. It is a metric that takes both precision and recall into account and is particularly useful when dealing with imbalanced datasets. It is calculated as:

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Final accuracy of model = 0.9679077204202833

Label	True positive	False positive	False Negative	Precision	Recall
0	33	5	26	0.868421052631579	0.559322033898305
1	7193	227	31	0.969407008086253	0.995708748615725
2	642	11	58	0.983154670750383	0.917142857142857
3	117	3	76	0.975	0.606217616580311
4	490	35	90	0.933333333333333	0.844827586206897

Figure 6.3: Precision, Recall and F1 Score

6.5 Disclaimer

The model presented in this context was trained for a total of 30 minutes, and the grid search was performed for another 30 minutes. While the results and metrics presented here are accurate based on

the data and resources used at the time of training, it is important to note that the accuracy and the model itself may be subject to change in the future. Specifically, when the algorithm and grid search are given more time, the results may be different, and the model may become more accurate. Thus, the information provided should be considered as a snapshot of the model's performance at a specific point in time, and it should not be relied upon as a definitive representation of the model's capabilities.

Chapter 7

Deep Learning

7.1 Splitting the data

Training data, testing data, and validation data are all crucial components in machine learning workflows.

1. Training data: This is the data set that is used to train the machine learning model. The training data is labeled with the correct output, and the model learns to predict the output based on the input features. The quality and size of the training data have a significant impact on the model's performance. The testing data was kept as 80% of the total data.

2. Testing data: This is a separate data set that is used to evaluate the performance of the trained model. The testing data is used to simulate real-world scenarios and to check how well the model can generalize to new data. It is important to use testing data that is not used during training to avoid overfitting. The testing data was kept as 20% of the total data

3. Validation data: This is a subset of the training data that is used to tune the model's hyperparameters. The validation data is used to evaluate the model's performance during training and to make adjustments to improve the model's accuracy. It is important to use a validation data set that is separate from the testing data to avoid any bias in evaluating the model's performance. The validation data was kept as 10% of the training data.

7.2 Loss function

Categorical cross entropy [13] loss is a commonly used loss function in machine learning, particularly for multi-class classification problems. It measures the difference between the predicted probability distribution and the true probability distribution of the output classes. The formula for categorical

cross entropy loss can be expressed as:

$$\mathcal{L}_{\text{CE}}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where y is the true probability distribution of the output classes, $\hat{y}$ is the predicted probability distribution of the output classes, and C is the total number of classes. The summation is taken over all the classes.

In the above equation, y_i is a binary indicator (0 or 1) of whether class i is the correct classification for the input. If class i is the correct classification, then y_i is equal to 1; otherwise, y_i is equal to 0. The predicted probability distribution $\hat{y}$ is the output of the model, which is a probability distribution over all the classes.

Using categorical cross entropy loss as the loss function encourages the model to output high probabilities for the correct class and low probabilities for the incorrect classes. The gradient of this loss function with respect to the model's parameters can be used to update the parameters during the training process.

7.3 Optimizer

Adam [14] (Adaptive Moment Estimation) is an optimization algorithm used in training machine learning models. It is an extension of the stochastic gradient descent (SGD) optimizer and combines the advantages of both the Adagrad and RMSprop optimizers. Adam maintains a running estimate of the first and second moments of the gradients, which are used to update the parameters in the model.

The update rule for Adam can be expressed as follows:

$$t = t + 1 \tag{7.1}$$

$$lr_t = learning\_rate \times \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \tag{7.2}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{7.3}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7.4}$$

$$\theta = \theta - lr_t \times \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{7.5}$$

where:

1. t is the current iteration number
2. lr_t is the learning rate at iteration t

3. β_1 and β_2 are the decay rates for the first and second moments of the gradients, respectively
4. m_t is the first moment (mean) of the gradients
5. v_t is the second moment (variance) of the gradients
6. g_t is the gradient of the loss function with respect to the model parameters
7. ϵ is a small value added to the denominator to prevent division by zero

Adam adaptively scales the learning rate for each parameter based on the first and second moments of the gradients. This means that the learning rate is adjusted automatically during training, making it more efficient than fixed learning rate methods.

7.4 Activation Functions

Activation functions are used to introduce non-linearity in neural networks, which is important to learn complex patterns in the data. The activation functions used in the current model are:

1. Softmax : The Softmax activation function [15] is used to convert the output of the neural network into a probability distribution over the classes. It is commonly used in multi-class classification problems. The Softmax function is defined as:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

   where z_i is the output of the $i-th$ neuron, and k is the number of classes.

2. ReLU: The ReLU (Rectified Linear Unit) [16] activation function is a non-linear function that outputs the input value if it is positive, and zero otherwise. It is computationally efficient and has been shown to be effective in deep neural networks. The ReLU function is defined as:

$$ReLU(x) = max(0, x)$$

3. Linear: The Linear activation function simply outputs the input value without any non-linearity. It is commonly used in the output layer of regression problems, where the model is trying to predict a continuous value. The Linear function is defined as:

$$Linear(x) = x$$

7.5 Model Components

7.5.1 Convolution 1D layer

A 1D convolutional layer [17] is a fundamental building block of convolutional neural networks (CNNs) used for processing sequential data such as time-series signals or natural language text. It applies a set of learned filters (also known as kernels) to the input data, sliding them along the time axis to extract local patterns and features.

The output of a 1D convolutional layer is obtained by computing the convolution of the input signal with the set of filters, followed by a non-linear activation function. The size of the output tensor is determined by the number of filters, their length, and the padding and stride values used in the convolution operation.

The mathematical expression for the 1D convolution operation can be written as:

$$y_i = \sigma\Big(\sum_{j=0}^{k-1} w_j x_{i+j} + b\Big) \tag{7.6}$$

where y_i is the i-th element of the output tensor, σ is the activation function, w_j are the filter weights, x_{i+j} are the input values that the filter is applied to, b is the bias term, and k is the length of the filter.

In practice, 1D convolutional layers are often stacked together with pooling layers and other types of layers to form more complex neural network architectures that can learn hierarchical representations of the input data.

7.5.2 MaxPooling 1D layer

Max pooling [18] is a common operation used in convolutional neural networks to downsample the feature maps produced by convolutional layers. In 1D max pooling, the input sequence is divided into non-overlapping windows, and the maximum value within each window is selected to form the output sequence.

The main purpose of max pooling is to reduce the spatial dimensionality of the feature maps while retaining the most salient features. This can help to prevent overfitting and increase the computational efficiency of the network.

The mathematical expression for 1D max pooling can be written as:

$$y_i = \max_{j=i\times s}^{(i+1)\times s-1} x_j \tag{7.7}$$

where y_i is the i-th element of the output sequence, x_j is the j-th element of the input sequence, and s is the size of the pooling window.

In practice, max pooling is often used in conjunction with 1D convolutional layers to create a feature extraction pipeline that can learn hierarchical representations of the input data.

7.5.3 Batch Normalization Layer

Batch normalization is a technique [19] used in deep learning to improve the training stability and speed of neural networks. It normalizes the activations of a given layer by subtracting the batch mean and dividing by the batch standard deviation, thus reducing the internal covariate shift that can occur during training.

The main benefits of batch normalization are improved gradient flow, reduced overfitting, and increased learning rates. In addition, batch normalization can act as a regularizer by introducing small amounts of noise to the activations.

The mathematical expression for batch normalization can be written as:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{7.8}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{7.9}$$

where x_i is the i-th element of the input batch, μ_B and σ_B are the mean and standard deviation of the batch, ϵ is a small constant to avoid division by zero, $\hat{x}_i$ is the normalized value of x_i, γ and β are learnable scale and shift parameters, and y_i is the output value.

7.6 Activation Function layers

Activation functions are used in neural networks to introduce non-linearity to the output of a layer. Without an activation function, a neural network would simply be a linear regression model, which is limited in its ability to represent complex non-linear relationships in the data.

There are several activation functions commonly used in deep learning, including the sigmoid function, linear activation function, and rectified linear unit (ReLU) function.

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{7.10}$$

The hyperbolic tangent function is defined as:

The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{7.11}$$

The linear activation function, also known as the identity function, is a simple activation function that performs a linear transformation on the input, leaving it unchanged. In Tensorflow, the linear activation function is implemented as the default activation function for the output layer of many neural network models.

The mathematical expression for the linear activation function is:

$$f(x) = x \tag{7.12}$$

This function is linear because it simply maps the input to the output without any non-linear transformation. While it may not be as powerful as some other activation functions in terms of capturing non-linear relationships, the linear activation function is often useful for regression tasks where the output is a continuous value.

Activation functions are typically applied element-wise to the output of a layer, so that each neuron in the layer has a non-linear activation. The choice of activation function can have a significant impact on the performance of the network, and different functions may be more appropriate for different types of problems.

7.6.1 Concantenate Layer

The concatenate layer is a type of layer in neural networks that combines the output of multiple layers into a single tensor along a specified axis. This is often used to merge the output of two or more parallel branches of a network, allowing the network to learn from multiple sources of information simultaneously.

The concatenate operation can be represented mathematically as:

$$y = [x_1, x_2, \ldots, x_n] \tag{7.13}$$

where y is the concatenated output tensor, and $x_1, x_2, \ldots, x_n$ are the input tensors to be concatenated.

7.6.2 Add layer

The add layer is a type of layer in neural networks that adds the output of multiple layers element-wise, producing a single tensor output. This is often used to merge the output of two or more parallel branches of a network, allowing the network to learn from multiple sources of information simultaneously.

The add operation can be represented mathematically as:

$$y = x_1 + x_2 + \cdots + x_n \tag{7.14}$$

where y is the output tensor, and $x_1, x_2, \ldots, x_n$ are the input tensors to be added.

7.6.3 Global Average Pooling Layer

The global average pooling layer is a type of layer in neural networks that downsamples the spatial dimensions of a tensor to produce a single feature map. It does this by computing the average value of each feature map across all spatial locations, resulting in a single value per feature map. This is often used as a form of regularization, as it encourages the network to learn features that are globally relevant to the input rather than being sensitive to local variations.

The global average pooling operation can be represented mathematically as:

$$y_i = \frac{1}{H \times W} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{i,h,w} \tag{7.15}$$

where y_i is the output value for the i-th feature map, $x_{i,h,w}$ is the input value at spatial location (h, w) in the i-th feature map, and H and W are the height and width of the feature maps, respectively.

7.6.4 Dense Layer

The dense layer [20], also known as a fully connected layer, is a type of layer in neural networks where each neuron in the layer is connected to every neuron in the previous layer. This allows the layer to learn complex, non-linear relationships between the input and output. The output of a dense layer is computed by taking a weighted sum of the inputs, followed by an activation function.

The operation of a dense layer can be represented mathematically as:

$$y = \sigma(Wx + b) \tag{7.16}$$

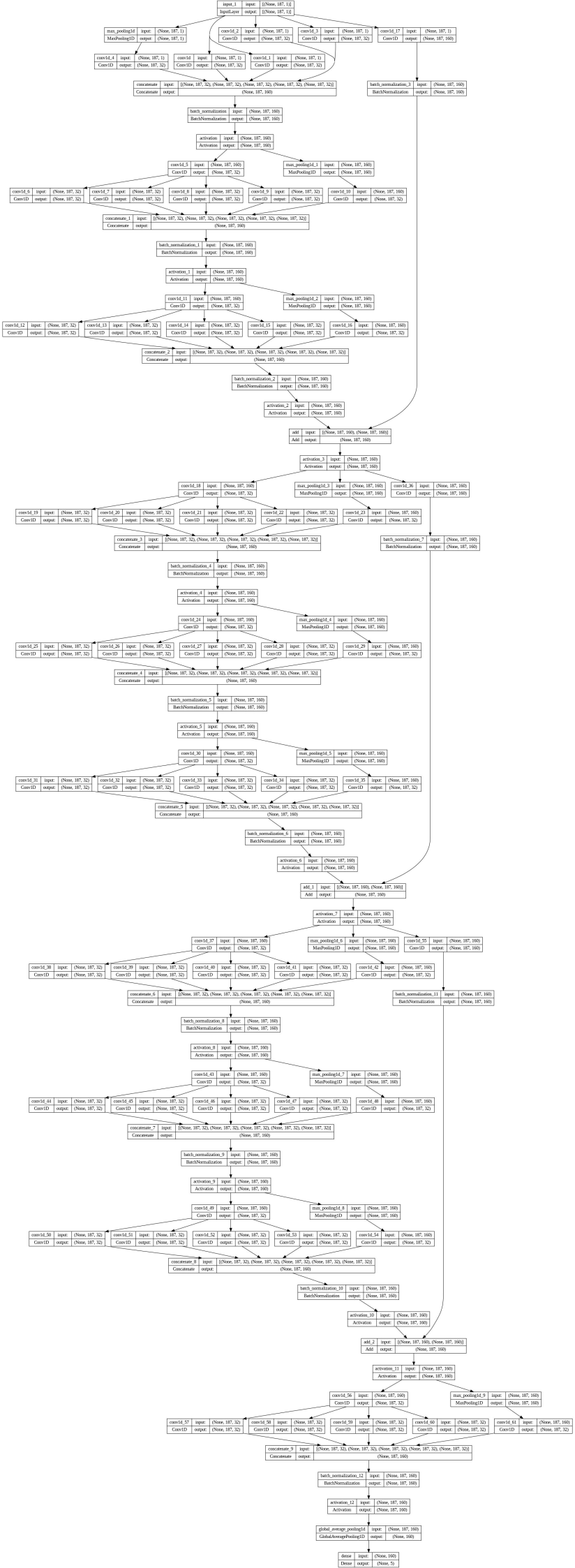
where y is the output of the dense layer, x is the input to the layer, W is the weight matrix, b is the bias vector, and $\sigma(\cdot)$ is the activation function.

7.7 Model Architecture

The architecture of the model used in this study is quite complex and detailed, and includes multiple layers of different types of neurons, as well as various hyperparameters and optimization strategies. Due to space limitations, we are unable to provide a detailed description of the model architecture in the main text of this document. However, a comprehensive summary of the model's structure is provided in the Appendix for those who are interested in learning more about the specifics of the model's design.

1. Total params: 846,277
2. Trainable params: 842,117
3. Non-trainable params: 4,160

Please refer to the next page for a schematic representation of the model.


input_1 | input: [(None, 187, 1)] | InputLayer | output: [(None, 187, 1)]
max_pooling1d | input: (None, 187, 1) | MaxPooling1D | output: (None, 187, 1)
conv1d_2 | input: (None, 187, 1) | Conv1D | output: (None, 187, 32)
conv1d_3 | input: (None, 187, 1) | Conv1D | output: (None, 187, 32)
conv1d_17 | input: (None, 187, 1) | Conv1D | output: (None, 187, 160)
conv1d_4 | input: (None, 187, 1) | Conv1D | output: (None, 187, 32)
conv1d | input: (None, 187, 1) | Conv1D | output: (None, 187, 32)
conv1d_1 | input: (None, 187, 1) | Conv1D | output: (None, 187, 32)
concatenate | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_3 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
batch_normalization | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_5 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_1 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_6 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_7 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_8 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_9 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_10 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_1 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_1 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_1 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_11 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_2 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_12 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_13 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_14 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_15 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_16 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_2 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_2 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_2 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
add | input: [(None, 187, 160), (None, 187, 160)] | Add | output: (None, 187, 160)
activation_3 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_18 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_3 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_36 | input: (None, 187, 160) | Conv1D | output: (None, 187, 160)
conv1d_19 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_20 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_21 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_22 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_23 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_3 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_7 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
batch_normalization_4 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_4 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_24 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_4 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_25 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_26 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_27 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_28 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_29 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_4 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_5 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_5 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_30 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_5 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_31 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_32 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_33 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_34 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_35 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_5 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_6 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_6 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
add_1 | input: [(None, 187, 160), (None, 187, 160)] | Add | output: (None, 187, 160)
activation_7 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_37 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_6 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_55 | input: (None, 187, 160) | Conv1D | output: (None, 187, 160)
conv1d_38 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_39 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_40 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_41 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_42 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_6 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_11 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
batch_normalization_8 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_8 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_43 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_7 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_44 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_45 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_46 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_47 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_48 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_7 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_9 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_9 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_49 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_8 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_50 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_51 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_52 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_53 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_54 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_8 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_10 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_10 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
add_2 | input: [(None, 187, 160), (None, 187, 160)] | Add | output: (None, 187, 160)
activation_11 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
conv1d_56 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
max_pooling1d_9 | input: (None, 187, 160) | MaxPooling1D | output: (None, 187, 160)
conv1d_57 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_58 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_59 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_60 | input: (None, 187, 32) | Conv1D | output: (None, 187, 32)
conv1d_61 | input: (None, 187, 160) | Conv1D | output: (None, 187, 32)
concatenate_9 | input: [(None, 187, 32), (None, 187, 32), (None, 187, 32), (None, 187, 32)] | Concatenate | output: (None, 187, 160)
batch_normalization_12 | input: (None, 187, 160) | BatchNormalization | output: (None, 187, 160)
activation_12 | input: (None, 187, 160) | Activation | output: (None, 187, 160)
global_average_pooling1d | input: (None, 187, 160) | GlobalAveragePooling1D | output: (None, 160)
dense | input: (None, 160) | Dense | output: (None, 5)

7.8 Model Training

The model was trained on a GPU for 60 minutes, allowing for efficient computation and faster training. During this time, the model underwent 100 epochs of training, which is a common benchmark for deep learning models.

Throughout the training process, several metrics were used to evaluate the model's performance. These metrics include loss, precision, recall, accuracy, and F1 score. Here is a breakdown of these metrics and how they evolved during the training process:

1. Loss: The loss metric measures the difference between the predicted output and the actual output. It represents how well the model is fitting the training data. During training, the loss gradually decreased, indicating that the model was learning and improving.

2. Precision: Precision measures the proportion of true positives among all predicted positives. It tells us how often the model correctly predicted positive examples. Precision improved steadily over the course of training, indicating that the model was becoming more precise in its predictions.

3. Recall: Recall measures the proportion of true positives among all actual positives. It tells us how well the model is able to identify positive examples. Recall also improved over time, suggesting that the model was becoming more sensitive to positive examples.

4. Accuracy: Accuracy measures the proportion of correct predictions among all predictions. It gives us an overall idea of how well the model is performing. Accuracy increased over time as the model improved.

5. F1 score: The F1 score is the harmonic mean of precision and recall. It is a good measure of overall performance, taking into account both precision and recall. The F1 score increased during training as the model became more accurate and precise.

6. Learning rate: The learning rate determines how quickly the model adjusts its parameters based on the error. During training, the learning rate was adjusted at various points to ensure that the model was learning optimally. It was gradually decreased over time to allow the model to converge on a good solution. The learning rate was monitored for both the training and validation sets to ensure that the model was not overfitting or underfitting.

7. Support : In validation metrics, "support" refers to the number of samples or instances that belong to a particular class in the validation set. It is a key metric in evaluating the performance of a machine learning model, particularly in multi-class classification problems.

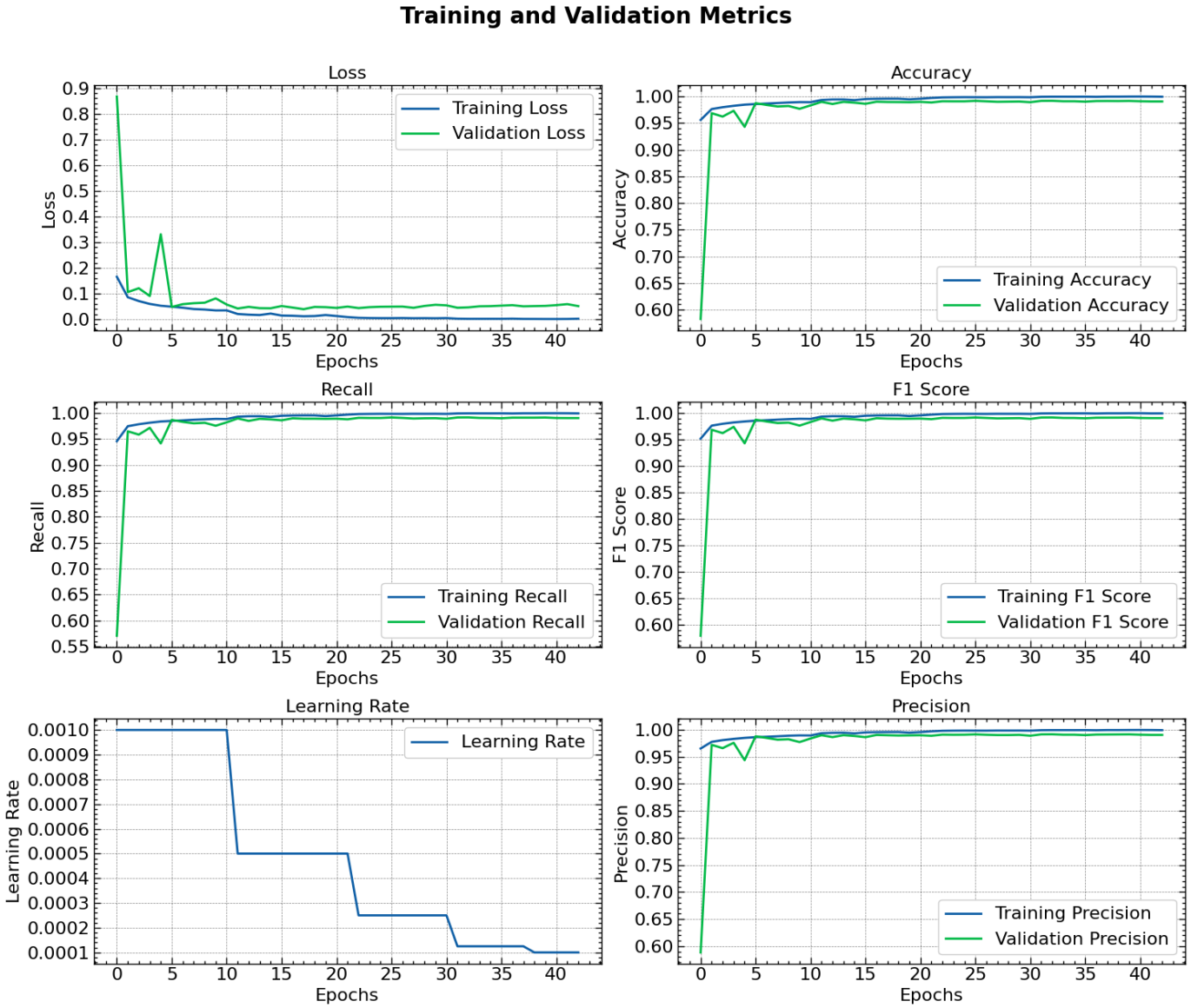


Figure 7.1: Model Metrics during Training

7.9 Model Performance

All data is based on final testing dataset.

Confusion Matrix

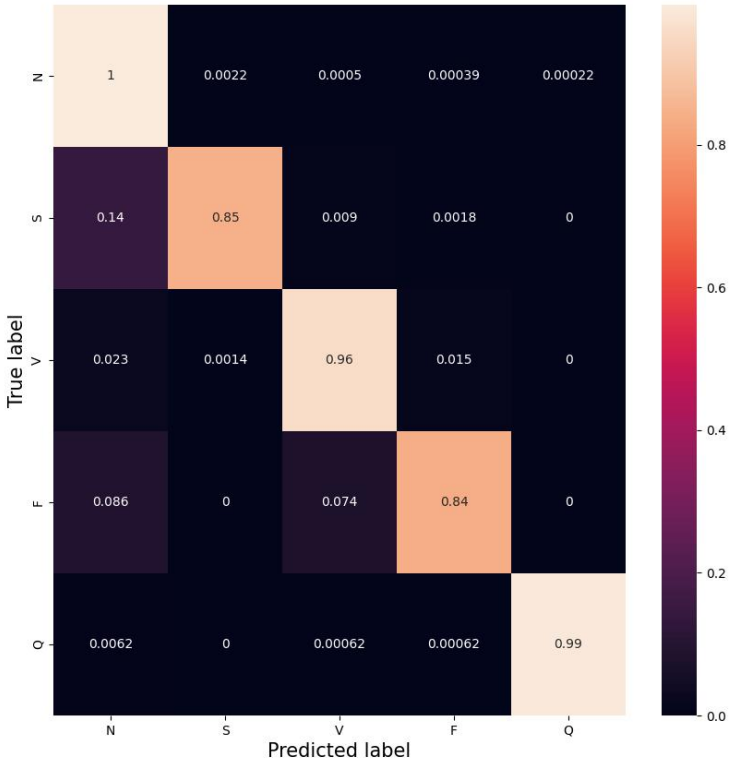


Figure 7.2: Confusion matrix generated from testing data. Percentages of the total testing data are indicated in place of actual values to decrease clutter

Classification Report

	Precision	Recall	F1 score	Support	
Label					
N	0.99	1.00	0.99	18118	
S	0.92	0.85	0.88	556	
V	0.98	0.96	0.97	1448	
F	0.81	0.84	0.83	162	
Q	1.00	0.99	1.00	1608	

Figure 7.3: Classification report based on testing data.

Final Average Metrics (based on testing data)

1. Accuracy: 0.9889457336013155
2. Precision: 0.9889457336013155
3. Recall: 0.9889457336013155
4. F1 Score: 0.9889457336013155

7.10 Disclaimer

It is important to note that the deep learning model used in this project was trained for a total of 100 epochs and tested for one hour on GPU. While the model achieved an impressive testing accuracy of almost 99%, it is possible that further improvements could be made by tweaking the model architecture or training it for a longer period of time. Therefore, the reported accuracy should be viewed as a preliminary result, and further experimentation is necessary to fully understand the capabilities of the model.

CONCLUSION

In conclusion, this project aimed to compare the performance of machine learning and deep learning models. While we were able to achieve a testing accuracy of about 96.7% using various algorithms and techniques for feature extraction in machine learning, our deep learning model yielded an impressive testing accuracy of almost 99% without any explicit feature engineering. These results suggest that deep learning is a more efficient and effective approach than traditional machine learning methods.

The success of deep learning raises a preliminary question about the role of human intelligence in the future of science and technology. With deep neural networks being able to automatically learn and extract complex features, it seems that the need for human input and intervention may become obsolete in certain fields. However, it is important to note that while deep learning has shown remarkable success in certain domains, it still has limitations and is not a complete replacement for human intelligence.

In conclusion, this project highlights the potential of deep learning in solving complex problems and achieving high accuracy rates. While the future of science may involve a greater reliance on artificial intelligence, it is important to consider the ethical implications of such advancements and continue to ensure that human intelligence remains relevant and valued in our society.

REFERENCES

[1] S. learn team, "Histogram Boosting Classifier." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html, 2008. [Online; accessed 01-April-2023].

[2] A. Atkielski, "Features of ECG." https://en.wikipedia.org/wiki/Electrocardiography#/media/File:SinusRhythmLabels.svg, 2007. [Online; accessed 22-March-2023].

[3] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *IEEE ENGINEERING IN MEDICINE AND BIOLOGY*, vol. 20, pp. 45–50, 6 2001. (PMID: 11446209).

[4] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[5] W.-K. Jeong, H. Pfister, and M. Fatica, "Chapter 46 - medical image processing using gpu-accelerated itk image filters," in *GPU Computing Gems Emerald Edition* (W. mei W. Hwu, ed.), Applications of GPU Computing Series, pp. 737–749, Boston: Morgan Kaufmann, 2011.

[6] J. E. E. Jack Xin, "Filtering and Convolutions." https://www.math.uci.edu/icamp/courses/math77a/lecture_10f/filtering.pdf. [Online; accessed 26-March-2023].

[7] N. Contributors, "numpy.hanning." https://numpy.org/doc/stable/reference/generated/numpy.hanning.html. [Online; accessed 16-March-2023].

[8] F. Gustafsson, "Determining the initial states in forward-backward filtering," *IEEE Transactions on Signal Processing*, vol. 44, no. 4, pp. 988–992, 1996.

[9] S. learn Developers, "Feature Extraction." https://scikit-learn.org/stable/modules/feature_extraction.html, 2007-2022. [Online; accessed 22-March-2023].

[10] "Pearson Correlation Coefficient." https://en.wikipedia.org/wiki/Pearson_correlation_coefficient, 2022. [Online; accessed 26-March-2023].

[11] S. Narkhede, "Understanding Confusion Matrix." https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62, 2018. [Online; accessed 26-March-2023].

[12] K. P. Shung, "Accuracy, Precision, Recall or F1?." https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, 2018. [Online; accessed 02-April-2023].

[13] K. E. Koech, "Cross-Entropy Loss Function." https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e, 2020. [Online; accessed 26-March-2023].

[14] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning." https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, 2017. [Online; accessed 28-March-2023].

[15] H. Mahmood, "The Softmax Function, Simplified ." https://towardsdatascience.com/softmax-function-simplified-714068bf8156, 2018. [Online; accessed 27-March-2023].

[16] J. Brownlee, "A Gentle Introduction to the Rectified Linear Unit (ReLU)." https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neura 2019. [Online; accessed 28-March-2023].

[17] K. contributors, "Keras convolution1d layer documentation." https://keras.io/api/layers/convolution_layers/convolution1d/, Accessed 2023-04-04.

[18] K. contributors, "Keras maxpooling1d layer documentation." https://keras.io/api/layers/pooling_layers/max_pooling1d/, Accessed 2023-04-04.

[19] S. Singla, "Why is Batch Normalization useful in Deep Neural Networks?." https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf, 2020. [Online; accessed 30-March-2023].

[20] A. N. V L Helen Josephine and V. L. Alluri, "Impact of Hidden Dense Layers in Convolutional Neural Network to enhance Performance of Classification Model," *IOP Conference Series: Materials Science and Engineering*, vol. 1131, no. 1, p. 012007, 2021.

LIST OF PUBLICATIONS ON PRESENT WORK

[1] Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database. IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001). (PMID: 11446209)

[2] Bender, Theresa, et al. "Benchmarking the Impact of Noise on Deep Learning-based Classification of Atrial Fibrillation in 12-Lead ECG." arXiv preprint arXiv:2303.13915 (2023).

[3] Pham, Huy, et al. "Machine learning-based detection of cardiovascular disease using ECG signals: performance vs. complexity." arXiv preprint arXiv:2303.11429 (2023).

[4] Odugoudar, Aryan, and Jaskaran Singh Walia. "ECG Classification System for Arrhythmia Detection Using Convolutional Neural Networks." arXiv preprint arXiv:2303.03660 (2023).

[5] Mishra S, Khatwani G, Patil R, Sapariya D, Shah V, Parmar D, Dinesh S, Daphal P, Mehendale N. ECG Paper Record Digitization and Diagnosis Using Deep Learning. J Med Biol Eng. 2021;41(4):422-432. doi: 10.1007/s40846-021-00632-0. Epub 2021 Jun 15. PMID: 34149335; PMCID: PMC8204064.

[6] Wu, H., Patel, K.H.K., Li, X. et al. A fully-automated paper ECG digitisation algorithm using deep learning. Sci Rep 12, 20963 (2022). https://doi.org/10.1038/s41598-022-25284-1

Appendix A

Model Structure Summary

Due to the complexity of the model structure and the fact that it is typically printed in a text file, it can be challenging to include it in a Portable Document Format (PDF) document. Therefore, we are providing a link to the uploaded model structure file on GitHub for easy access and reference. The link to the model structure file can be found here: https://github.com/ScientificArchisman/Dissertaton/blob/main/model_summary.txt.

Appendix B

Code for Data preprocessing and Filtering and Feature Extraction

---

The code for data preprocessing, filtering, and feature extraction is often extensive and may be challenging to include in a Portable Document Format (PDF). To make this code accessible and available for further reference, we are providing links to the GitHub repositories where the code can be found. These links can be found here:

https://github.com/ScientificArchisman/Dissertaton/blob/main/Preprocessing_Feature_Extraction_Visualisation.ipynb We hope that these links will allow for easy access to the data preprocessing, filtering, and feature extraction code for interested readers and researchers.

Appendix C

Code For Models

The code for both the deep learning and machine learning models used in this project can be quite large, making it difficult to include in a Portable Document Format (PDF). To make the code accessible and available for further reference, we are providing links to both the deep learning model and the machine learning model. These links can be found here:

1. Deep learning model https://github.com/ScientificArchisman/Dissertaton/blob/main/convoluti ipynb
2. Machine Learning Model https://github.com/ScientificArchisman/Dissertaton/blob/main/ machine_learning.ipynb

[Insert link to machine learning model code repository] We hope that these links will allow for easy access to the model code for interested readers and researchers.