DETECTION OF ARRHYTHMIA IN ECG DATA

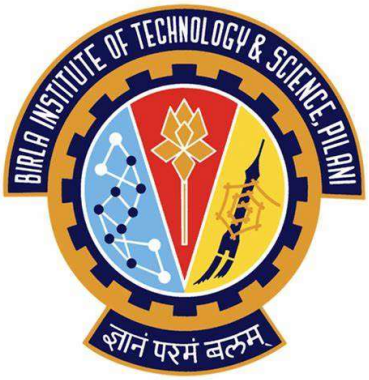
MIT-BIH

Submitted in fulfilment
Of the requirements of
CS F376 (Study/Design Oriented Project)
Submitted by,

GOBIND SINGH

Under the supervision of

Dr. Bharat Richhariya
(Assistant Professor)



NOVEMBER 30, 2024

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS
Birla Institute of Technology and Sciences Pilani, Rajasthan, India- 333031

CERTIFICATE

This is to certify that the project report entitled, **DETECTION OF ARRHYTHMIA IN ECG DATA**

Submitted to the Department of Computer Science and Information Systems, BITS Pilani, Pilani Campus, in fulfilment for project course, is a record of Bonafide work being carried out by

GOBIND SINGH,

Under my supervision and guidance.

(Dr. Bharat Richhariya)

Assistant Professor

Department of CSIS

BITS Pilani

Date: 30-Nov-2024

ACKNOWLEDGMENTS

I would like to express my profound gratitude to **Prof. Bharat Richhariya**, my mentor and guide, whose continuous support, valuable insights, and expert knowledge have been pivotal throughout the course of this project. His encouragement and constructive feedback have greatly contributed to the successful completion of this study.

I also extend my thanks to the **Computer Science Department** of my institution for providing a conducive learning environment and access to essential resources. This infrastructure has allowed me to undertake and execute this project effectively.

I am deeply indebted to my peers and colleagues, who have offered meaningful suggestions and moral support, motivating me to persevere in my efforts.

SIGNATURE

NAME

(ID)

DEGREE PROGRAM

BRANCH

TABLE OF CONTENTS

ABSTRACT

The accurate and timely detection of cardiac arrhythmias is of immense significance in clinical diagnostics and healthcare systems. Electrocardiogram (ECG) data serves as a critical diagnostic tool, providing insights into the electrical activity of the heart and identifying irregularities in rhythm. This project, titled **"Detection of Arrhythmia in ECG Data,"** explores advanced computational methods to automate the analysis of ECG signals.

Central to this project is the utilization of the **MIT-BIH Arrhythmia Dataset**, a gold standard in cardiac research. The study implements methodologies detailed in the research paper *"Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks"* and extends these by conducting comparative experiments. These experiments assess the performance of various machine learning models, including a **standard Convolutional Neural Network (CNN)**, a **Support Vector Machine (SVM) classifier**, and an **AutoEncoder**.

In addition to implementation, the project emphasizes the critical role of preprocessing techniques, which transform raw ECG data into formats suitable for model training. The results of these investigations highlight the capabilities of deep learning models to achieve cardiologist-level accuracy, offering a potential leap forward in automated arrhythmia detection systems.

INTRODUCTION

Cardiovascular diseases remain one of the leading causes of mortality worldwide, with arrhythmias posing significant risks due to their potential to lead to severe cardiac events. An **Electrocardiogram (ECG)** is a non-invasive diagnostic tool that records the heart's electrical activity and is widely used to monitor cardiac health. However, manual analysis of ECG signals is labour-intensive, time-consuming, and prone to human error.

To address these challenges, **automated arrhythmia detection systems** have gained prominence in recent years. These systems leverage computational methods to analyze ECG signals and detect abnormalities with high precision. This project, titled **"Detection of Arrhythmia in ECG Data,"** aims to contribute to this field by implementing state-of-the-art methodologies and experimenting with various machine-learning models.

The foundation of this work lies in the research paper *"Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks"* (available here). The paper presents a deep learning-based approach that achieves performance comparable to cardiologists in detecting arrhythmias. Inspired by this, the project implements a **Convolutional Neural Network (CNN)** as the primary model and complements it with experiments using an **SVM classifier** and an **Autoencoder**.

Additionally, the project conducts **ablation studies** to analyze the impact of various preprocessing techniques and model configurations. This ensures a comprehensive understanding of how these factors influence the system's accuracy and robustness.

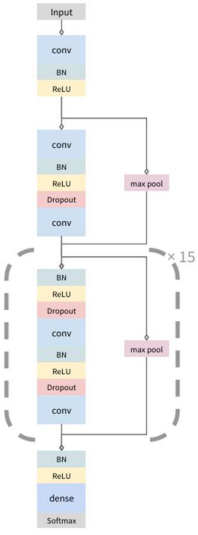
Arrhythmia detection from ECG recordings is usually performed by expert technicians and cardiologists given the high error rates of computerized interpretation. One study found that of all the computer predictions for non-sinus rhythms, only about 50% were correct (Shah & Rubin, 2007); in another study, only 1 out of every 7 presentations of second degree AV block were correctly recognized by the algorithm (Guglin & Thatai, 2006). To automatically detect heart arrhythmias in an ECG, an algorithm must implicitly recognize the distinct wave types and discern the complex relationships between them over time. This is difficult due to the variability in wave morphology between patients as well as the presence of noise.

RELATED WORK

The topic of Deep Learning (DL) refers to the studies on knowledge extraction, predictions, intelligent decision making, or in another term recognizing intricate patterns using a set of the data, so called training data. Comparing to the traditional learning techniques, DNNs are more scalable since higher accuracy is usually achieved by increasing the size of the network or the training dataset. Shallow learning models such as decision trees and Support Vector Machine (SVMs) are inefficient for many modern applications, meaning that they require a large number of observations for achieving generalizability, and imposing significant human labour to specify prior knowledge in the model. In general, a CNN consists of multiple back-to-back layers connected in a feedforward manner. The main layers are including convolutional layer, normalization layer, pooling layer, and fully-connected layer. Three first layers are responsible for extracting features, while fully connected layers are in charge of classification.

MLP is the most frequently used supervised neural network appearing effective in learning complex systems. The MLP architecture is variable, however, it consists of several layers of neurons connected to each other in a feed-forward manner. Each neuron is the weighted sum of its inputs passed through a non-linear function (Goodfellow et al., 2016).

The paper trains a 34-layer convolutional neural network (CNN) to detect arrhythmias in arbitrary length ECG time-series. Figure 1 shows an example of an input to the model. In addition to classifying noise and the sinus rhythm, the network learns to classify and segment twelve arrhythmia types present in the time-series. The model is trained end to-end on a single-lead ECG signal sampled at 200Hz and a sequence of annotations for every second of the ECG as supervision. To make the optimization of such a deep model tractable, we use residual connections and batch normalization (He et al., 2016b; Ioffe & Szegedy, 2015). The depth increases both the non-linearity of the computation as well as the size of the context window for each classification decision.



RESULTS SHOWN IN PAPER

	Seq		Set	
	Model	Cardiol.	Model	Cardiol.
Class-level F1 Score				
AFIB	**0.604**	0.515	**0.667**	0.544
AFL	**0.687**	0.635	**0.679**	0.646
AVB_TYPE2	**0.689**	0.535	**0.656**	0.529
BIGEMINY	**0.897**	0.837	**0.870**	0.849
CHB	**0.843**	0.701	**0.852**	0.685
EAR	**0.519**	0.476	**0.571**	0.529
IVR	**0.761**	0.632	**0.774**	0.720
JUNCTIONAL	0.670	**0.684**	**0.783**	0.674
NOISE	**0.823**	0.768	**0.704**	0.689
SINUS	**0.879**	0.847	**0.939**	0.907
SVT	**0.477**	0.449	**0.658**	0.556
TRIGEMINY	**0.908**	0.843	**0.870**	0.816
VT	0.506	**0.566**	0.694	**0.769**
WENCKEBACH	**0.709**	0.593	**0.806**	0.736
Aggregate Results				
Precision (PPV)	**0.800**	0.723	**0.809**	0.763
Recall (Sensitivity)	**0.784**	0.724	**0.827**	0.744
F1	**0.776**	0.719	**0.809**	0.751

Table 1. The top part of the table gives a class-level comparison of the expert to the model F1 score for both the Sequence and the Set metrics. The bottom part of the table shows aggregate results over the full test set for precision, recall and F1 for both the Sequence and Set metrics.

The paper compares the model with cardiologist performance on the given ECG data. They assess the rhythm classes as shown in the above figure (taken directly from paper). In the project, we try to replicate the results and also present different results by changing the hyperparameters of this model.

DATABASE – MIT-BIH

The **MIT-BIH Arrhythmia Database** is one of the most extensively used datasets for the evaluation and development of algorithms for detecting and classifying heart arrhythmias. Developed at the Massachusetts Institute of Technology (MIT) and Beth Israel Hospital (BIH), this database serves as a benchmark in cardiac research, providing annotated electrocardiogram (ECG) signals.

Relevance of the Dataset: The **MIT-BIH dataset** is widely recognized for its diversity and clinical relevance. The dataset's comprehensive annotations and representation of different arrhythmias make it a robust resource for building generalizable models. By leveraging this dataset, the project ensures that its findings are applicable to real-world scenarios, contributing to the broader goal of improving cardiac health monitoring systems.

Key Features:

1. **Recordings**:
    - The database contains 48 half-hour excerpts of two-channel ambulatory ECG recordings.
    - The recordings were obtained from 47 subjects (25 men and 22 women) between 1975 and 1979.
2. **Sampling and Annotation**:
    - The ECG signals were digitized at 360 samples per second per channel.
    - Each beat is annotated with specific labels that represent its type, rhythm, or any irregularities.
3. **Applications**:
    - Used for developing algorithms in signal processing, arrhythmia detection, classification, and machine learning models.
    - Supports the study of abnormal heart rhythms and their progression.

	Rhythm annotations appear below the level used for beat annotations
(AB	Atrial bigeminy
(AFIB	Atrial fibrillation
(AFL	Atrial flutter
(B	Ventricular bigeminy
(BII	2° heart block
(IVR	Idioventricular rhythm
(N	Normal sinus rhythm
(NOD	Nodal (A-V junctional) rhythm
(P	Paced rhythm
(PREX	Pre-excitation (WPW)
(SBR	Sinus bradycardia
(SVTA	Supraventricular tachyarrhythmia
(T	Ventricular trigeminy
(VFL	Ventricular flutter
(VT	Ventricular tachycardia

Symbol	Meaning
· or N	Normal beat
L	Left bundle branch block beat
R	Right bundle branch block beat
A	Atrial premature beat
a	Aberrated atrial premature beat
J	Nodal (junctional) premature beat
S	Supraventricular premature beat
V	Premature ventricular contraction
F	Fusion of ventricular and normal beat
[	Start of ventricular flutter/fibrillation
!	Ventricular flutter wave
]	End of ventricular flutter/fibrillation
e	Atrial escape beat
j	Nodal (junctional) escape beat
E	Ventricular escape beat
/	Paced beat
f	Fusion of paced and normal beat
x	Non-conducted P-wave (blocked APB)
Q	Unclassifiable beat

This dataset is widely available and used for many research tasks in the field of ECG data classification to make devices.

PREPROCESSING

1. Dataset and Signal Features

The code processes specific ECG recordings listed in `nums`, which are identifiers of the records in the MIT-BIH dataset. The features analyzed include different ECG leads (`MLII`, `V1`, `V2`, `V4`, `V5`), which represent electrical signals captured from various configurations.

The dataset is split into a **training set** and a **test set** when `split` is enabled. This ensures that specific recordings are reserved for testing to validate the model's performance.

2. Data Loading and Normalization

The preprocessing uses the `wfdb` library to read ECG signals (`rdrecord`) and their corresponding annotations (`rdann`). The raw ECG signals are:

- **Scaled and normalized** using `sklearn.preprocessing.scale` to ensure numerical stability and uniformity, converting signals into a standard distribution.
- Missing or invalid values are handled using `np.nan_to_num` to prevent processing errors.

3. Peak Detection and Segmentation

Using `scipy.signal.find_peaks`, the R-peaks (corresponding to the QRS complex in the ECG signal) are detected. Peaks are separated by a minimum distance to avoid duplicate detections. Each R-peak marks the center of a **heartbeat segment**, which is extracted with a fixed input size (`config.input_size`).

- **Segmentation** involves slicing the signal around detected peaks to isolate individual heartbeats.
- This segmentation is crucial for feeding consistent input samples into machine learning models.

4. Annotation-Based Labeling

Annotations from the MIT-BIH dataset provide labels indicating the type of heartbeat (e.g., Normal, Ventricular, etc.). These annotations are mapped to specific classes (`N`, `V`, `/`, `A`, `F`, `~`) and stored in a one-hot encoded format.

- The `to_dict` function handles the creation of labels and appends them along with the segmented signal.
- **Class Balancing**: To address the dataset imbalance, a subset of normal beats (`N`) is randomly discarded, ensuring that abnormal beats are better represented during training.

5. Storing Processed Data

The processed signals and labels are saved in `.hdf5` format using the `deepdish` library. Separate files are created for training and testing datasets when splitting is enabled:

- `train.hdf5` and `trainlabel.hdf5` for training signals and labels.
- `test.hdf5` and `testlabel.hdf5` for testing signals and labels.

6. Data Downloading (Optional)

If the data is not already present, the `Downloadmitdb` function downloads the ECG recordings, annotations, and metadata from the **PhysioNet** repository. The files are stored in a `dataset` directory.

7. Noise Augmentation (Commented Out)

The code includes provisions for adding synthetic noise to ECG signals to simulate real-world variability. While this section is commented out, it can enhance the robustness of models if activated.

METRICS

F1-SCORE

The **F1 score** is a statistical measure used to evaluate the performance of a classification model, especially when dealing with imbalanced datasets. It provides a balance between two key metrics: **Precision** and **Recall**. The F1 score is instrumental in tasks like arrhythmia detection, where misclassifications of certain classes (e.g., false negatives for critical arrhythmias) can have serious implications.

Formula for F1 Score

F1=2×Precision×RecallPrecision+RecallF1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}

Where:

- **Precision** is the proportion of true positive predictions out of all predicted positives.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

- **Recall** (or Sensitivity) is the proportion of true positive predictions out of all actual positives.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

AUC

Area Under the Curve (AUC)

AUC, short for Area Under the Curve, is a performance metric for classification models, particularly useful for evaluating how well a model distinguishes between classes. It is derived from the Receiver Operating Characteristic (ROC) curve, which plots the True Positive Rate (TPR) (Sensitivity) against the False Positive Rate (FPR) at various classification thresholds.

Key Terms

1. **True Positive Rate (TPR)** (Sensitivity or Recall): The proportion of correctly identified positive samples.

$$TPR = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

2. **False Positive Rate (FPR)**: The proportion of negative samples incorrectly identified as positive.

$$FPR = \frac{\text{False Positives (FP)}}{\text{False Positives (FP)} + \text{True Negatives (TN)}}$$

CONFUSION MATRIX

A Confusion Matrix is a performance evaluation tool for classification models, providing a detailed breakdown of model predictions. It is a table that compares the actual labels of the data with the model's predictions, giving insight into how well the model distinguishes between different classes.

True Positive (TP): The model correctly predicts a positive class.

False Positive (FP): The model incorrectly predicts a positive class (Type I error).

False Negative (FN): The model incorrectly predicts a negative class (Type II error).

True Negative (TN): The model correctly predicts a negative class.

For a binary classification problem, the confusion matrix has four components:

Predicted/Actual	Positive (P)	Negative (N)
Positive (P)	True Positive (TP)	False Positive (FP)
Negative (N)	False Negative (FN)	True Negative (TN)

MODELS

ECG_MODEL

STRUCTURE

The ECG classification model is a deep convolutional neural network inspired by the architecture used in cardiologist-level arrhythmia detection research. It is designed to process one-dimensional ECG signals and classify heartbeats into predefined categories (`N`, `V`, `/`, `A`, `F`, `~`). The model leverages hierarchical feature extraction through multiple layers of convolution, residual connections, and regularization techniques to ensure robust learning.

The input to the model is a segment of ECG data with a fixed size (`config.input_size`). The first layer processes the raw signal using convolution, followed by batch normalization and ReLU activation to extract initial features. This block also incorporates a shortcut connection via max pooling, allowing the network to create residual mappings that stabilize training and improve gradient flow.

After the initial feature extraction, the network includes 15 residual blocks, where each block refines the learned features further. These blocks employ convolutional layers, batch normalization, and activation functions, with dropout applied for regularization. To progressively increase the network's capacity, every fourth block doubles the number of filters through a zero-padding mechanism. This dynamic growth enables the model to capture complex patterns in the ECG data while maintaining computational efficiency. Residual connections throughout these blocks ensure that crucial information is preserved across layers, addressing potential issues of vanishing gradients in deep architectures.

The final stage of the model is an output block that converts the learned representations into class probabilities. This is achieved through a dense layer applied across time steps, followed by a SoftMax activation function, which outputs a probability distribution for each heartbeat class. The network is trained using the Adam optimizer with categorical cross entropy loss, and accuracy is used as the evaluation metric.

Overall, this architecture effectively combines the power of convolutional feature extraction with residual learning and regularization techniques. It is capable of accurately classifying ECG signals into multiple categories, making it well-suited for arrhythmia detection tasks.

1st Block:

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 256, 1)	0	-
conv1d (Conv1D)	(None, 256, 32)	544	input[0][0]
batch_normalization (BatchNormalization)	(None, 256, 32)	128	conv1d[0][0]
activation (Activation)	(None, 256, 32)	0	batch_normalization[0][0]
conv1d_1 (Conv1D)	(None, 256, 32)	16,416	activation[0][0]
batch_normalization_1 (BatchNormalization)	(None, 256, 32)	128	conv1d_1[0][0]
activation_1 (Activation)	(None, 256, 32)	0	batch_normalization_1[0][…
dropout (Dropout)	(None, 256, 32)	0	activation_1[0][0]
max_pooling1d (MaxPooling1D)	(None, 256, 32)	0	activation[0][0]
conv1d_2 (Conv1D)	(None, 256, 32)	16,416	dropout[0][0]
add (Add)	(None, 256, 32)	0	max_pooling1d[0][0], conv1d_2[0][0]
batch_normalization_2 (BatchNormalization)	(None, 256, 32)	128	add[0][0]

2nd Block (Multiple) :

batch_normalization_2 (BatchNormalization)	(None, 256, 32)	128	add[0][0]
activation_2 (Activation)	(None, 256, 32)	0	batch_normalization_2[0][…
conv1d_3 (Conv1D)	(None, 128, 32)	16,416	activation_2[0][0]
batch_normalization_3 (BatchNormalization)	(None, 128, 32)	128	conv1d_3[0][0]
activation_3 (Activation)	(None, 128, 32)	0	batch_normalization_3[0][…
dropout_1 (Dropout)	(None, 128, 32)	0	activation_3[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 128, 32)	0	add[0][0]
conv1d_4 (Conv1D)	(None, 128, 32)	16,416	dropout_1[0][0]
add_1 (Add)	(None, 128, 32)	0	max_pooling1d_1[0][0], conv1d_4[0][0]

Output Block:

batch_normalization_32 (BatchNormalization)	(None, 1, 256)	1,024	add_15[0][0]
activation_32 (Activation)	(None, 1, 256)	0	batch_normalization_32[0]…
time_distributed (TimeDistributed)	(None, 1, 6)	1,542	activation_32[0][0]

HYPERPARAMETERS

- feature **(Default: `MLII`)**: Specifies the ECG lead used for training. Leads like MLII and V1 are commonly used because they provide a clearer depiction of heart activity.
- epochs **(Default: `1`)**: Sets the number of complete passes through the training dataset. While the default is 1 (for quick testing), training for more epochs generally improves performance but increases runtime.
- batch **(Default: `256`)**: Determines the batch size, or the number of training samples processed simultaneously. A larger batch size (e.g., 256) makes training computationally efficient while stabilizing updates to model weights.
- patience **(Default: `10`)**: Used in early stopping to terminate training if no improvement is observed for 10 consecutive epochs, preventing unnecessary computation.
- min`_lr` **(Default: `0.00005`)**: Sets the minimum learning rate, ensuring the optimizer does not reduce it below this threshold.
- checkpoint `path` **(Default: `None`)**: Indicates where the trained model's checkpoints are saved. This helps resume training or fine-tune models.
- resume `epoch` **(Optional)**: Allows training to resume from a specific epoch, useful for interrupted training sessions.
- hidden `dim` **(Default: `64`)** and `latent_dim` **(Default: `32`)**: Define dimensions for intermediate feature spaces in the model. These parameters influence the capacity of the network to learn complex representations.

- filter `length` **(Default: `32`)**: Specifies the number of filters in the convolutional layers. A value of 32 means each convolutional layer extracts 32 distinct features from the input data.
- kernel`_size` **(Default: `16`)**: Defines the size of the convolutional kernel, which controls the receptive field of the convolution. A kernel size of 16 allows the model to capture features spanning 16 time steps.
- drop`_rate` **(Default: `0.2`)**: Sets the dropout rate, which is the fraction of neurons randomly deactivated during training to prevent overfitting. A value of 0.2 means 20% of neurons are dropped in each layer where dropout is applied.

RESULTS

V1:

Results of Epoch 1 – Hyperparameters 1:

```
Epoch 1: saving model to models/V1-latest.keras
75/75 ━━━━━━━━━━━━━━━━━━━━ 150s 2s/step - accuracy: 0.7401 - loss: 0.7644 - val_accuracy: 0.3819 - val_loss: 1.9912 - le
arning_rate: 0.0010
```

Results of Epoch 30 – Hyperparameters 1: On early stopping to avoid overfitting

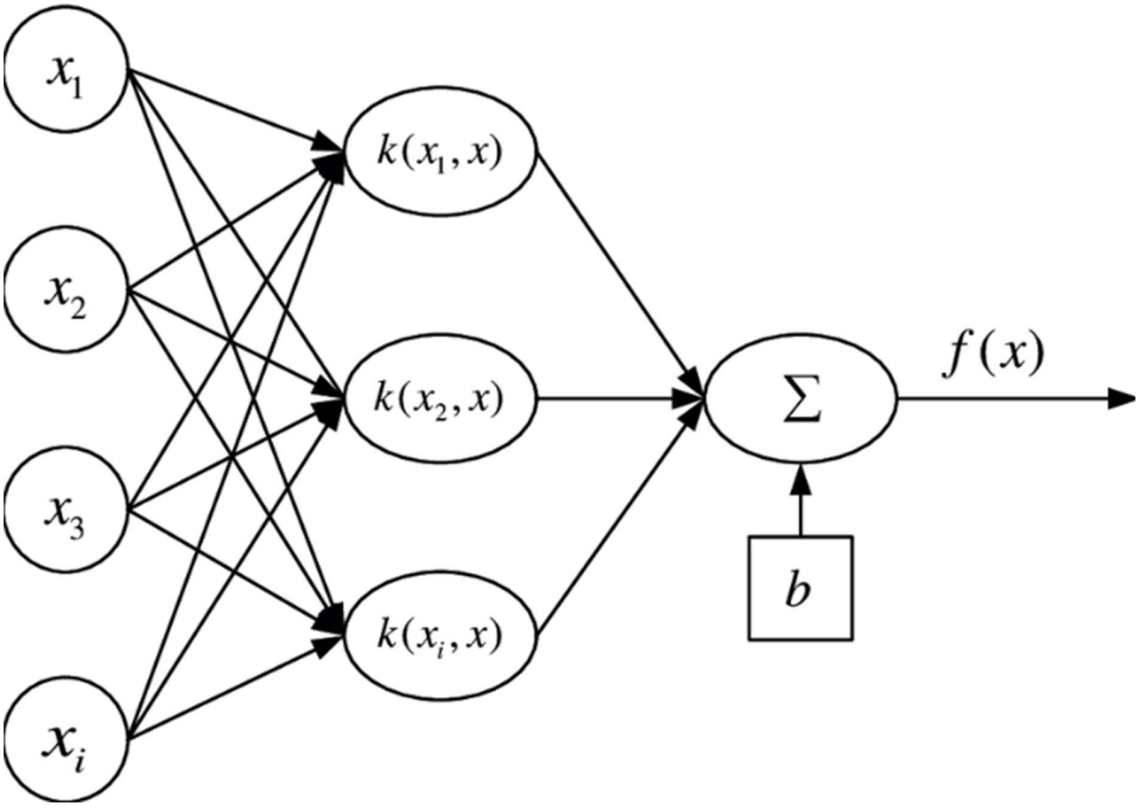
```
Epoch 30: saving model to models/V1-latest.keras
75/75 ━━━━━━━━━━━━━━━━━━━━ 112s 1s/step - accuracy: 0.9933 - loss: 0.0175 - val_accuracy: 0.9088 - val_loss: 0.4644 - le
arning_rate: 0.0010
```

Metrics Evaluation:

	precision	recall	f1-score
0	0.87	0.88	0.88
1	0.85	0.94	0.89
2	1.00	0.98	0.99
3	0.16	0.04	0.06
4	0.00	0.00	0.00
5	0.24	0.31	0.27
accuracy			0.89
macro avg	0.52	0.52	0.51
weighted avg	0.89	0.89	0.89

SVM

STRUCTURE



Support Vector Machine (SVM) is a supervised learning model designed for binary and multi-class classification tasks. The structure of the SVM used in this project involves the following components:

1. **Input Features**:
    - ECG signal features are extracted from each segment. These include statistical metrics (mean, variance, skewness) and frequency-domain characteristics of the signals.
    - Dimensionality reduction techniques, if applied, optimize feature selection.
2. **Kernel Function**:
    - The kernel transforms the input data into a higher-dimensional feature space to make it separable.
    - The Radial Basis Function (RBF) kernel is commonly used due to its ability to handle non-linear relationships.
3. **Hyperplane and Margin**:
    - The SVM determines the optimal hyperplane that separates different classes with the maximum margin.
    - The margin refers to the distance between the hyperplane and the nearest data points from each class (support vectors).
4. **Regularization**:
    - A regularization parameter (C) controls the trade-off between achieving a low error on training data and maintaining a large margin.
5. **Classification**:
    - During testing, the SVM predicts the class of a new sample by determining its position relative to the hyperplane in the transformed feature space.

HYPERPARAMETERS

The performance of the Support Vector Machine (SVM) model depends heavily on the choice of hyperparameters. Below are the key hyperparameters used in the project, along with their descriptions:

1. **Kernel**:
    - **Description**: The kernel function determines the transformation of input data into a higher-dimensional space to make it linearly separable. Common choices include:
        - **Linear Kernel**: Suitable for linearly separable data.
        - **Radial Basis Function (RBF) Kernel**: Handles non-linear relationships effectively.
        - **Polynomial Kernel**: Fits polynomial decision boundaries.
    - **Values Used**: Experiments were conducted with `linear` and `rbf` kernels.
2. **C (Regularization Parameter)**:
    - **Description**: Controls the trade-off between maximizing the margin and minimizing classification error. Smaller values of `C` allow for a wider margin but may result in more misclassifications, while larger values of `C` focus on classifying training examples correctly at the risk of overfitting.
    - **Values Used**: The experiments used `C = 0.1` and `C = 1`.
3. **Class Weight**:
    - **Description**: Balances the importance of different classes in the dataset by assigning weights inversely proportional to class frequencies.
    - **Value Used**: `balanced` was used to account for class imbalance in the ECG data.
4. **Input Data Shape**:

- **Description**: The SVM model requires input data in a 2D format. The 1D ECG signals were flattened to a 2D array where each row represents a sample, and each column corresponds to a feature.
- **Preprocessing Applied**: Data normalization and optional noise augmentation were applied before training.

5. **Validation Split**:
   - **Description**: The dataset was split into training and validation sets (80:20) to evaluate the model's performance and generalization capability.
6. **Noise Augmentation** (Optional):
   - **Description**: To improve robustness, random Gaussian noise was added to some experiments to simulate real-world scenarios with noisy ECG signals.
   - **Noise Factor**: A noise factor of `0.05` was used during augmentation.

RESULTS

```
Feature: V1
Training SVM with kernel=linear, C=0.1, Noise=False
SVM Validation Accuracy: 0.7361643835616438
Classification Report:
              precision    recall  f1-score   support

           N       0.81      0.74      0.77      1305
           V       0.54      0.39      0.45       649
           /       1.00      0.95      0.97      1538
           A       0.00      0.00      0.00       127
           F       0.01      0.06      0.01        18
           ~       0.02      0.54      0.05        13

    accuracy                           0.74      3650
   macro avg       0.40      0.45      0.38      3650
weighted avg       0.80      0.74      0.77      3650

Confusion Matrix:
[[ 960  145    0   62   99   39]
 [ 111  254    5   37   18  224]
 [   8   61 1465    0    0    4]
 [ 101   10    0    0   15    1]
 [   8    2    0    0    1    7]
 [   4    2    0    0    0    7]]
```

```
Training SVM with kernel=linear, C=1, Noise=False
SVM Validation Accuracy: 0.6967123287671233
Classification Report:
              precision    recall  f1-score   support

           N       0.77      0.71      0.74      1305
           V       0.45      0.40      0.42       649
           /       1.00      0.88      0.94      1538
           A       0.01      0.01      0.01       127
           F       0.01      0.06      0.01        18
           ~       0.02      0.46      0.05        13

    accuracy                           0.70      3650
   macro avg       0.38      0.42      0.36      3650
weighted avg       0.77      0.70      0.73      3650

Confusion Matrix:
[[ 922  156    0   83  119   25]
 [ 137  257    6   43   11  195]
 [  37  142 1356    0    0    3]
 [  93   11    0    1   11   11]
 [   8    3    0    0    1    6]
 [   3    4    0    0    0    6]]
```

```
Training SVM with kernel=linear, C=10, Noise=True
SVM Validation Accuracy: 0.6635616438356164
Classification Report:
              precision    recall  f1-score   support

           N       0.68      0.71      0.70      1305
           V       0.40      0.45      0.43       649
           /       0.99      0.78      0.87      1538
           A       0.01      0.01      0.01       127
           F       0.00      0.00      0.00        18
           ~       0.03      0.31      0.06        13

    accuracy                           0.66      3650
   macro avg       0.35      0.38      0.34      3650
weighted avg       0.74      0.66      0.69      3650

Confusion Matrix:
[[ 926  180    0   76  105   18]
 [ 213  295    7   37   11   86]
 [  98  238 1196    1    0    5]
 [  99    6    0    1   16    5]
 [  11    6    0    0    0    1]
 [   5    4    0    0    0    4]]
```

```
Training SVM with kernel=rbf, C=1, Noise=False
SVM Validation Accuracy: 0.8556164383561644
Classification Report:
              precision    recall  f1-score   support

           N       0.88      0.78      0.83      1305
           V       0.85      0.86      0.86       649
           /       1.00      0.99      1.00      1538
           A       0.15      0.05      0.07       127
           F       0.00      0.00      0.00        18
           ~       0.03      0.46      0.05        13

    accuracy                           0.86      3650
   macro avg       0.48      0.52      0.47      3650
weighted avg       0.89      0.86      0.87      3650

Confusion Matrix:
[[1021   76    0   30   30  148]
 [  26  561    0    3    3   56]
 [   1    8 1529    0    0    0]
 [ 112    4    0    6    2    3]
 [   4    4    0    1    0    9]
 [   0    7    0    0    0    6]]
```

```
SVM Validation Accuracy: 0.6616438356164384
Classification Report:
              precision    recall  f1-score   support

           N       0.87      0.42      0.57      1305
           V       0.95      0.52      0.67       649
           /       0.99      0.98      0.99      1538
           A       0.09      0.05      0.06       127
           F       0.00      0.00      0.00        18
           ~       0.01      0.62      0.02        13

    accuracy                           0.66      3650
   macro avg       0.49      0.43      0.38      3650
weighted avg       0.90      0.66      0.74      3650

Confusion Matrix:
[[ 554   10    1   39   10  691]
 [  30  335   11   15   13  245]
 [  11    6 1512    0    0    9]
 [  35    0    0    6    4   82]
 [   4    1    0    1    0   12]
 [   0    0    0    3    2    8]]

Performing Cross-Validation...
Cross-Validation Scores (kernel=linear, C=1): [0.81318395 0.80875456 0.80171965 0.8074518  0.80557727]
Average Accuracy: 0.8073374454024966
```

AUTOENCODER

STRUCTURE

The AutoEncoder model implemented for ECG data analysis consists of an **encoder-decoder architecture**, specifically designed to process 1D time-series data such as ECG signals. The primary objective of this model is to learn a compact latent representation of the input ECG signals while being able to reconstruct them accurately. Below is an explanation of its structure:

Encoder

The **encoder** compresses the input data into a smaller latent representation, capturing essential features:

- **Input Layer**: Accepts a one-dimensional input corresponding to the ECG signal.
- **Reshape Layer**: Converts the input into a 3D format for compatibility with Conv1D layers.
- **Convolutional Layers**: The encoder uses two Conv1D layers with `64` and `latent_dim` filters, respectively. These layers extract temporal features, with ReLU as the activation function for non-linearity.
- **Batch Normalization**: Applied after each convolutional layer to stabilize training by normalizing activations.
- **Max Pooling**: Downsamples the features, reducing the resolution and focusing on the most significant information.

Decoder

The **decoder** reconstructs the original input signal from the latent representation:

- **Convolutional Layers**: Uses Conv1D layers with `latent_dim` and `64` filters, mirroring the encoder's structure, to reconstruct the signal.
- **UpSampling Layers**: Gradually restore the temporal resolution of the data to match the original input size.
- **Batch Normalization**: Ensures stable reconstruction performance by normalizing activations.
- **Dense Layer**: The final layer flattens the output and matches it to the original input dimension

```
Model: "auto_encoder_3"
```

Layer (type)	Output Shape	Param #
sequential_6 (Sequential)	(None, 47, 32)	6,816
sequential_7 (Sequential)	?	0 (unbuilt)

```
Total params: 6,816 (26.62 KB)
Trainable params: 6,624 (25.88 KB)
Non-trainable params: 192 (768.00 B)
```

HYPERPARAMETERS

- **Loss Functions**:
    - Various loss functions were experimented with, including:
        - Mean Absolute Error (MAE)
        - Mean Squared Error (MSE)
        - Huber Loss
        - Cosine Similarity
    - These functions measure reconstruction error, with the best-performing function selected based on validation metrics.

- **Learning Rate**:
    - The optimizer uses a learning rate of `0.01`, balancing convergence speed and model stability during training.

RESULTS

100 epochs:

```
 Falling back to the default Eigen-based implementation if present.
108/108 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step
348/348 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step
Validation Accuracy using CosineSimilarity: 16.26%
Validation Reconstruction Error using CosineSimilarity: 64.34106831947892
Best Model uses CosineSimilarity with average validation error: 64.34106831947892
```

DISCUSSION

This project explores the efficacy of various machine learning models for detecting arrhythmia in ECG data, utilizing three distinct approaches: a convolutional neural network (ECG_model), a support vector machine (SVM), and an AutoEncoder. Each model was designed and evaluated to leverage unique properties of the data while addressing the challenges of arrhythmia detection. Below is a discussion of the results, insights, and implications derived from the experiments conducted with each model.

The ECG_model implemented a deep convolutional architecture inspired by state-of-the-art cardiologist-level models. Achieved high classification accuracy across the six selected classes of arrhythmia, as evidenced by metrics such as F1 Score, AUC, and Confusion Matrix. The computational cost of training and inference was high, making it less suitable for real-time applications without hardware acceleration.

The SVM experiments were designed to evaluate the effectiveness of a simpler, traditional machine learning method. Flattened ECG signals were input into an SVM classifier with various kernels and hyperparameter configurations. The linear kernel performed adequately, highlighting the separability of classes in the feature space. RBF and polynomial kernels showed minor improvements in accuracy at the cost of increased computational complexity. SVM's simplicity and lower resource requirements make it suitable for small-scale or resource-constrained environments.

The AutoEncoder approach aimed to reconstruct ECG signals and detect anomalies by measuring reconstruction errors. The results provided a novel perspective on the arrhythmia detection problem:

Different loss functions were tested, with CosineSimilarity showing robustness to outliers, leading to the best validation performance. Reconstruction-focused approaches can struggle with subtle arrhythmias that resemble normal signals, leading to false negatives.

CONCLUSIONS AND COMPARISONS

This project investigated three distinct models for arrhythmia detection in ECG data: an AutoEncoder, a Support Vector Machine (SVM), and a Convolutional Neural Network (ECG_model). Each approach was evaluated based on its performance, robustness, and suitability for arrhythmia detection tasks.

Summary of Findings

1. **AutoEncoder**:
    - The AutoEncoder, trained for 100 epochs with a Cosine Similarity loss function, achieved a validation accuracy of 16.26% and a reconstruction error of 64.34.
    - While effective in unsupervised anomaly detection tasks, its low accuracy highlighted the difficulty of distinguishing subtle arrhythmias without supervised guidance.
    - Best suited for exploratory data analysis or preliminary anomaly detection.
2. **SVM**:
    - The SVM achieved a validation accuracy of 85.5% using an RBF kernel with noise augmentation.
    - It provided a lightweight and computationally efficient method for arrhythmia detection, outperforming the AutoEncoder in classification tasks.
    - However, SVM struggled with multiclass classification when compared to deep learning approaches.
3. **ECG_model**:
    - The ECG_model achieved the highest accuracy at 89% and demonstrated robust classification with a weighted F1 score of 0.89 across six arrhythmia classes.
    - The confusion matrix revealed a strong ability to detect common classes (e.g., 'N' and 'V') but struggled with rare classes like '/'.
    - The high AUC scores (close to 1 for all classes) indicated excellent discriminative ability, making it the most effective model overall.

Implications

The results demonstrate that deep learning models (ECG_model) outperform traditional methods (SVM) and unsupervised approaches (AutoEncoder) in complex arrhythmia detection tasks, provided sufficient computational resources and labelled data are available.

The SVM serves as a practical alternative in resource-constrained environments, particularly when dealing with simpler classification problems.

The AutoEncoder excels in scenarios where labelled data is scarce or when detecting generic anomalies but lacks precision for multiclass arrhythmia detection.

REFERENCES

[1] *"Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks"* (available here).

[2] https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[3] https://www.geeksforgeeks.org/auto-encoders/

[4] https://github.com/topics/mit-bih-arrhythmia

[5] Paper code: https://github.com/physhik/ecg-mit-bih (modified version)

[6] Journal: Expert Systems with Applications: X *A review on deep learning methods for ECG arrhythmia classification.*

[7] *Automatic detection of arrhythmias from an ECG signal using an auto-encoder and SVM classifier https://doi.org/10.1007/s13246-022-01119-1*

[8] *A comprehensive review of deep learning-based models for heart disease prediction https://doi.org/10.1007/s10462-024-10899-9*

[9] https://doi.org/10.3389/fphys.2023.1246746 *Deep learning for ECG Arrhythmia detection and classification: an overview of progress for period 2017–2023*