Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Project Title

Computer Aided Diagnosis (CAD) For Heart Diseases using Deep Learning Approach

By

Team Members:

No	Name	Department
1	Wael Mohamed Abd ElMoaty	Scientific Computing
2	Mahmoud Hamed Mohamed	Scientific Computing
3	Asmaa Hamed Mohamed	Scientific Computing
4	Asmaa Sobhy Mostafa	Scientific Computing
5	Gehad Khaled Sayed	Scientific Computing

Under Supervision of

[*Manal Mohsen Tantawi*]
[Associate Professor],
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

[Abdelrahman Mohamed Shaker]
[Teaching Assistant],
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

August 2020

Acknowledgements

All praise and thanks to ALLAH, who provided me the ability to complete this work. I hope to accept this work from me.

I am grateful of *my parents* and *my family* who are always providing help and support throughout the whole years of study. I hope I can give that back to them.

I also offer my sincerest gratitude to my supervisors,
Associate *Prof. Dr.Manal Mohsen Tantawi, and T.A. Abelrahman Mohamed Shaker* who have supported me throughout my thesis with their patience, knowledge and experience.

Finally, I would thank my friends and all people who gave me support and encouragement.

Abstract

Heart diseases classification based on electrocardiogram (ECG) signal has become a priority topic in the diagnosis of heart diseases because it can be obtained with a simple diagnostic tool of low cost. Since early detection of heart disease can enable us to ease the treatment as well as save people’s lives, accurate detection of heart disease using ECG is very important.

In this project, convolutional neural networks are proposed to classify sixteen different ECG beat types in the MIT-BIH arrhythmia database.

We aimed to implement a computer-aided diagnosis system in intensive care to help doctors instead of keeping an eye on the patient 24 hours, the system automatically classifies heartbeats and alarm the doctor.

We have tried Long short-term memory (LSTM), Visual Geometry Group (VGG16), Convolution Neural Network (CNN) and Inception. We have two architectures for classification, one stage and two stages. We tried more than one trail and we got best result from CNN.

One Stage CNN has 91.17 % as average accuracy
And 99.08 % as overall accuracy.
Two stages CNN has 91.98 % as average accuracy
And 98.74 % as overall accuracy.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AE: Atrial Escape
AP: Aberrated Atrial Premature
APC: Atrial Premature Contraction
BAP: Blocked Atrial Premature
CNN: Convolution Neural Network
Dynseg: Dynamic Segmentation
FBN: Fusion of Paced and Normal
FVN: Fusion of Ventricular and Norm
LBBB: Left Bundle Branch Block
LSTM: Long Short-Term Memory
NE: Nodal (Junctional) Escape
NN: Neural Network
NOR: Normal
NP: Nodal (Junctional) Premature
PCA: Principal Component Analysis
PVC: Premature Ventricular Contract
RBBB: Right Bundle Branch Block
SVM: Support Vector Machine
VE: Ventricular Escape
VF: Ventricular Flutter Wave
VGG: Visual Geometry Group

Chapter One:

Introduction

1-Introduction

1.1 Problem Definition

1.1.1 History

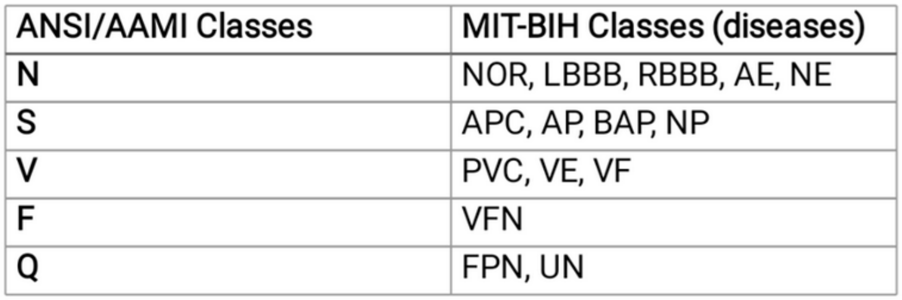
The electrocardiogram (ECG) signal is one of the most important and well-known biological signals used for diagnosing people's health.

The ECG trace shows three successive prominent waves named P, QRS and T waves. The Atria contractions (i.e. both right and left) are shown as the P wave, while the ventricular contractions (both right and left) are shown as QRS complex.
Finally, the T wave reflects the electrical activity produced when the ventricles are recharging for the next contraction.

Detection of QRS complex is one of the most important parts carried out in the ECG signal analysis. QRS detection, especially detection of R wave in heart signal, is easier than other portions of ECG signal due to its structural form and high amplitude.

Hence, cardiac arrhythmias can be indicated by detecting changes that occur in the shape of any of the three waves (P, QRS and T waves). Cardiac arrhythmias mean that there is an abnormal activity in the heart depending upon certain groups
of conditions. Arrhythmias have two main categories: The first one can trigger cardiac arrest and sudden death, such as tachycardia and ventricular fibrillation . On the other hand,
the second category (our interest in this project) needs attention but is not critically as life threatening as the first category.

the non-life-threatening arrhythmias (second category) can be divided into 5 main classes namely: non-ectopic (N), supraventricular ectopic (S), ventricular
ectopic (V), fusion (F), and unknown (Q), each class of them consist of number of diseases mention below in table



ANSI/AAMI Classes	MIT-BIH Classes (diseases)
N	NOR, LBBB, RBBB, AE, NE
S	APC, AP, BAP, NP
V	PVC, VE, VF
F	VFN
Q	FPN, UN

Table 1.1 The five main categories according to the ANSI/AAMI EC57: 1998 standard.

1.1.2 Applications

- **Apple Watch app**

How the ECG app works

The ECG app on Apple Watch Series 4 or later generates an ECG that is similar to a single-lead (or Lead I) ECG. In a doctor’s office, a standard 12-lead ECG is usually taken. This 12-lead ECG records electrical signals from different angles in the heart to produce twelve different waveforms. The ECG app on Apple Watch measures a waveform similar to one of those twelve waveforms. A single-lead ECG is able to provide information about heart rate and heart rhythm and enables classification of AFib. However, a single-lead ECG cannot be used to identify some other conditions, like heart attacks. Single-lead ECGs are often prescribed by doctors for people to wear at home or within the hospital so that the doctor can get a better look at the underlying rate and rhythm of the heart. However, the ECG app on Apple Watch Series 4 or later allows you to generate an ECG similar to a single-lead ECG without a prescription from your doctor.

In studies comparing the ECG app on Apple Watch to a standard 12-lead ECG taken at the same time, there was agreement between the ECG app classification of the rhythm as sinus or AFib compared to the standard 12-lead ECG.

The ability of the ECG app to accurately classify an ECG recording into A Fib and sinus rhythm was tested in a clinical trial of approximately 600 subjects, and demonstrated 99.6% specificity with respect to sinus rhythm classification and 98.3% sensitivity for A Fib classification for the classifiable results.

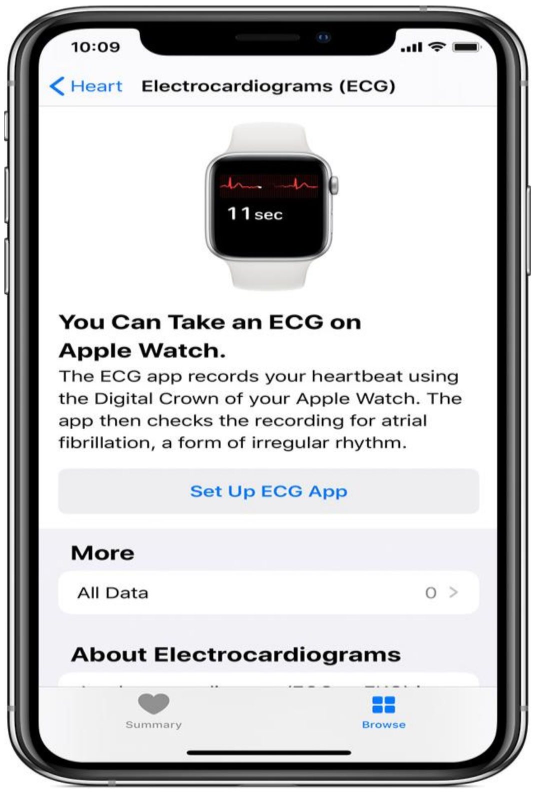
The clinical validation results reflect use in a controlled environment. Real world use of the ECG app may result in a greater number of strips being deemed inconclusive and not classifiable.

How to use the ECG app

The ECG app can record your heartbeat and rhythm using the electrical heart sensor on Apple Watch Series 4 or later and then check the recording for atrial fibrillation (AFib), a form of irregular rhythm.

The ECG app records an electrocardiogram which represents the electrical pulses that make your heartbeat. The ECG app checks these pulses to get your heart rate and see if the upper and lower chambers of your heart are in rhythm. If they're out of rhythm, that could be AFib.

The ECG app is currently available only in certain countries and regions.


10:09

Heart Electrocardiograms (ECG)

11 sec

You Can Take an ECG on Apple Watch.

The ECG app records your heartbeat using the Digital Crown of your Apple Watch. The app then checks the recording for atrial fibrillation, a form of irregular rhythm.

Set Up ECG App

More

All Data 0 >

About Electrocardiograms

Summary Browse

Figure 1.1 Apple watch app

Take an ECG

You can take an ECG at any time, when you're feeling symptoms such as a rapid or skipped heartbeat, when you have other general concerns about your heart health, or when you receive an irregular rhythm notification.

1. Make sure that your Apple Watch is snug and on the wrist that you selected in the Apple Watch app. To check, open the Apple Watch app, tap the My Watch tab, then go to General > Watch Orientation.
2. Open the ECG app on your Apple Watch.
3. Rest your arms on a table or in your lap.
4. With the hand opposite your watch, hold your finger on the Digital Crown. You don't need to press the Digital Crown during the session.
5. Wait. The recording takes 30 seconds. At the end of the recording, you will receive a classification, then you can tap Add Symptoms and choose your symptoms.
6. Tap Save to note any symptoms, then tap Done.

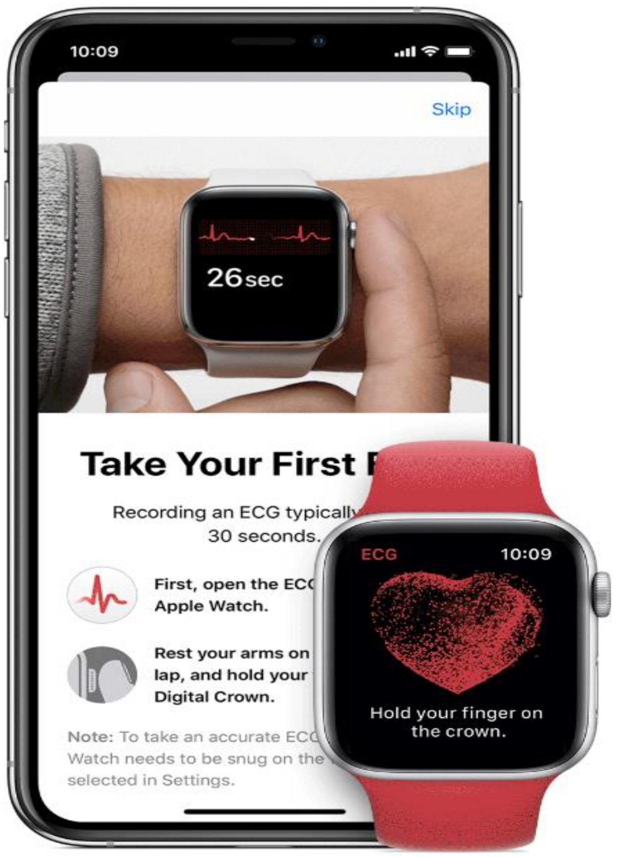
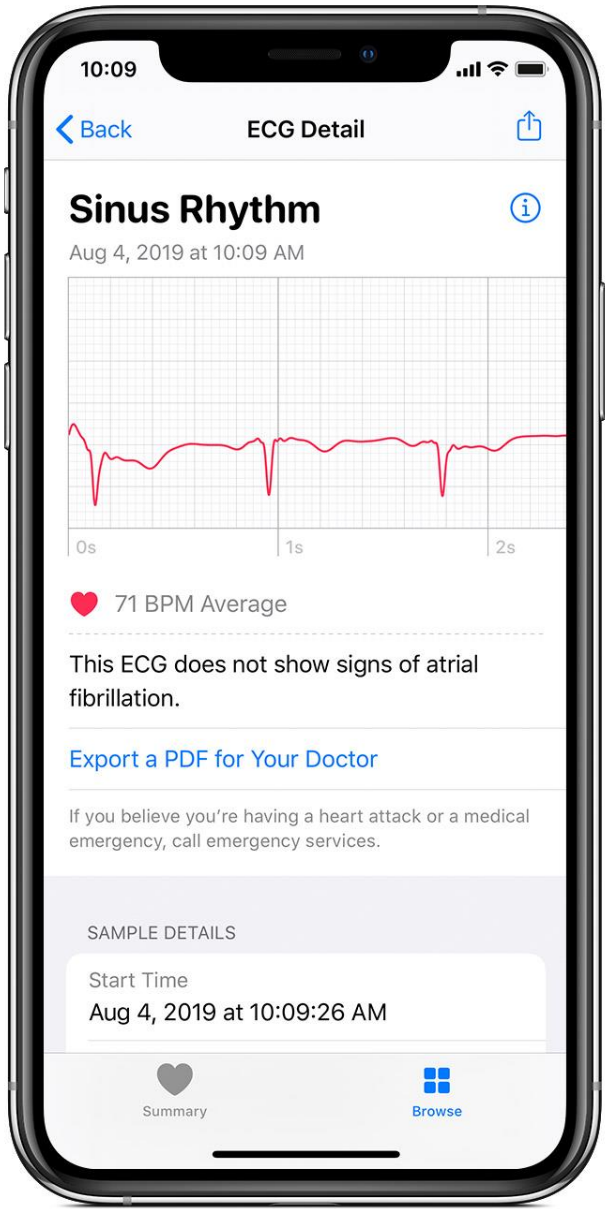


Figure 1.2 Apple watch

View and share your Health information

The ECG waveform, its associated classifications, and any noted symptoms will be saved in the Health app on your iPhone. You can also share a PDF with your doctor.

1. Open the Health app.
2. Tap the Browse tab, then tap Heart > Electrocardiograms (ECG).
3. Tap the chart for your ECG result.
4. Tap Export a PDF for Your Doctor.
5. Tap the Share button to print or share the PDF.


10:09

Back

ECG Detail

Sinus Rhythm

Aug 4, 2019 at 10:09 AM

71 BPM Average

This ECG does not show signs of atrial fibrillation.

Export a PDF for Your Doctor

If you believe you're having a heart attack or a medical emergency, call emergency services.

SAMPLE DETAILS

Start Time

Aug 4, 2019 at 10:09:26 AM

Figure 1.3 View and share your Health information *using Apple watch*

1.2 Motivation

Monitoring ECG in intensive care room for 24 hours is a very exhaustive task. Hence automating the diagnosis process through computer aided diagnosis (CAD) can provide great help to clinicians and more care to patients with minimum effort.

1.3 Objectives

The goal of this project is Developing a reliable system using deep learning to diagnose the input ECG heart beats to one of the cardiac arrhythmias.

1.4 Time plan

- **Take Background of Signal processing and deep learning.**
  Start: [October] End: [November]

- **Apply Pre-processing for ECG Signals Dataset.**
  Start: [November] End: [December]

- **Construct the deep learning approach.**
  Start: [December] End: [February]

- **Test deep learning approach and modify on it to get better accuracy.**

  Start: [February] End: [April]

- **Create the application.**

  Start: [April] End: [June]

1.5 Documentation Outline

Documentation Outline will be as follow:

Chapter 2: Background about project, including survey results and summary of published papers.

Chapter 3: Including System architecture and detailed explanation for each part in the architecture like system flow, preprocessing steps, deep learning models and so on.

Chapter 4: Including Implementation for each part in chapter 3.

Chapter 5: Including Results, models evaluation and Interface screenshots.

Chapter6: Conclusion and future work.

Background

2- Background

This project aims to target the **medical field**, especially **Heart Department**, this used in intensive care room for 24 hours is a very exhaustive task. Hence automating the diagnosis process through computer aided diagnosis (CAD) can provide great help to clinicians and more care to patients with minimum effort.

We aim to develop a reliable system using deep learning to diagnose the input ECG heart beats to one of the cardiac arrhythmias.

2.1. Related Work

In this section, a brief survey of some key published studies presented below.

2.1.1 Using Traditional methods

Tapanhavar et al [2] utilized Wavelet transform morphologic to extract features, SVM for classification and different datasets to classify 16 classes. 99% is the best accuracy achieved using random forest classification algorithm and MIT-BIH dataset. It has been applied to both lead 1 and lead 2 separately

Kumar Das el.at et al. [3] proposed utilized mixture of ST and Wavelet Transform to extract feature. then classified using Multilayer perceptron neural network (MLPNN) to classify 5 classes, resulting in an overall accuracy of 97.14 % using MIT-BIH as a validation database

Martis et al. [4] have considered five different Arrhythmia classes (Normal, Right Bundle Branch, Left Bundle Branch, Atrial

Premature Contraction and Premature Ventricular Contraction) and they achieved overall accuracy 98.11% after comparing different approaches for feature extraction which based on PCA, DWT and linear prediction model.

Yazdanian et al. [5] have considered the same five classes and they have achieved overall accuracy 96.67% using SVM, time-domain and wavelet features.

Sahoo et al [6] using SVM and (MLPNN) to achieve 96.9% using MIT-BIH as a validation database to classify 5 classes.

EL saadawy et al [7] utilized Wavelet transform and PCA to extract features, SVM to classify 15 classes mapped to five main categories using MIT-BIH as a validation database .98% is the best accuracy achieved for first stage (Category type) and 94.9% for second stage (Class type) it has been applied to both lead 1 and lead 2 separately

Features from DWT and PCA have been utilized by Elhaj *et al* [8] to classify 5 classes. Using SVM-RBF a 98.91% overall accuracy has been achieved.

Gnecchi *et al.* [9] utilized Wavelet transform Operation to extract features, probabilistic NN for classification and different datasets to classify 8 classes. 98.7% is the best accuracy achieved.

2.1.2 Using Deep learning methods

Zhang *et al.* [10] considered only five classes. Important points of each segmented heartbeat have been derived and the classified using multi layers 1D CNN. An overall accuracy of 97.5% has been achieved using MIT-BIH as a validation database.

Features from 1D-CNN have been utilized by Yildirim *et al* [11] to classify 17 classes. a 91.3% overall accuracy has been achieved using MIT-BIH as a validation database.

Acharya et al. [12] utilized CNN & LSTM to classify 5 classes. 98.1% is the best accuracy achieved.

Paper name	preprocessing	Feature Extraction	Classification	Accuracy	One leads or Two?	Classes
Topannavar et al. [1] 2014	-Filter using a wavelets-based -Segmentation using R-peak	Wavelets transform-Morphological	SVM classifier	99% in "class--oriented" -86% in the "subject-oriented	Tow leads	16 classes
Kumar Das et.al. [2] 2014	normalize the amplitude of ECG signals -the bandpass filter	mixture of ST and WT	multilayer perceptron neural network (MLPNN)	97.14%	One leads	5 classes
Martis et al. [3]	(DWT) discrete wavelet transform	principal components of segmented ECG	Fully connected feed forward neural Network - Least Square Support Vector Machine (LSSVM).	average accuracy of 98.11%.	One lead	5 Classes
Yazdanian et al. [4] 2013	combination of moving average filter, Butterworth filter and DWT	DWT (Di Transform)	Support Vector Machines	96.97%	One leads	5 classes
Sahoo et al. [5] 2017	multiresolution wavelet transform Numerous algorithms		neural network (NN) MLP-BP and support vector machines (SVM)	96.67% and 98.39% in NN and SVM	one Leads	4 categories
El-Saadawy et al. [6] 2018	Discrete Wavelet Transform	using DWT and RR features. Principle Component Analysis	(SVM) Support Vector Machine	-first stage is 98.40% -second stage 94.94%	Two Leads	15 classes mapped to 5 categories
Elhaj et al. [7] 2016	Using a filter bank. (DWT), QRS-detection (Pan-Tompkins), ECG-segmentation.	discrete wavelets transform (DWT) -principal component analysis (PCA)	SVM-RBF and NN	98.91%	One Leads	5 Classes
Gnecchi et al. [8] 2017	bandpass filter	Wavelets transform operations	Probabilistic Neural Network	92.75%	One Leads	8 Classes
Zhang et al. [9] 2017	- wavelet threshold method and wavelet decomposition - segmentation using R peaks	1D-CNN method	1D-CNN method	97.5%	One leads	5 classes

Yıldırım et al. [10] 2018	Using signal rescaling to the range [-1,1]	1D-CNN	1D-CNN	overall classification accuracy of 91.33%	One leads	17 classes
Acharya et al. [11] 2017		CNN and long short-term memory (LSTM)	CNN and long short-term memory (LSTM)	98.10%, sensitivity of 97.50% and specificity of 98.70%	One leads	5 classes

Table2.1 Related work survey

2.2 Observations

Note on Dataset:

- Most of the papers used **MIT BIH database**
- Data don't balance, data for normal classes is significantly more than other classes
- Almost of the papers takes one class from each category especially N, LBBB, RBBB, PVC, APC

Preprocessing:

- Filtering
- segmentation: to do it, we need to detect R-peaks

Feature & classification: we found two way to do these step

- Traditional methods: mainly wavelets & support vector machine (SVM).
- Deep learning using convolutional neural network "CNN".

Accuracy:
It is acceptable and High because almost of people work on 3 or 5 classes only.

We found most of papers classify only 3- 5 classes, few papers considered 15 classes.

Only one paper considers **16 Classes** Which Will Be Our Goal in This Project.

Chapter Three:

System Architecture

3.1 System Architecture

In General Architecture we first load our Dataset (The MIT-BIH dataset), Then we made preprocessing on all data, preprocessing containing four steps:

1-filtering.
2-QRS Detection.
3-Dynamic Segmentation.
4-Normalization and Resampling.

Then we split Dataset into training and testing data , then we need to made feature extraction before classification but we follow Deep Learning Models That doing feature extraction and classification in the same network don't need to made feature extraction first , because first layers in network made feature extraction and last layers made classification .
We used four deep learning models
1-CNN.
2-VGG16.
3-LSTM.
4-INCEPTION.
Each model we tried it using Five Trials of data to select best trial and best model to using it.

Our project follows two stages of classification, **first stage** makes class classification, classify between 16 class(disease).
Second Stage make category classification
Classify between five categories (N, S, V, Q, F) then make class classification of each category.

This table show classification of two stage.

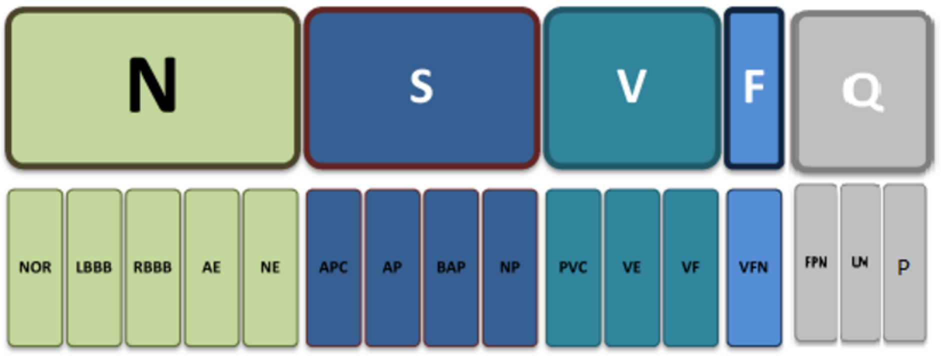


Figure 3.1 shows the classes associated with each category

Therefore, we have two architectures, first architecture for one stage (class classification), second architecture for two Stage (category classification the make class classification of each class).

→ this figure show architecture for one stage

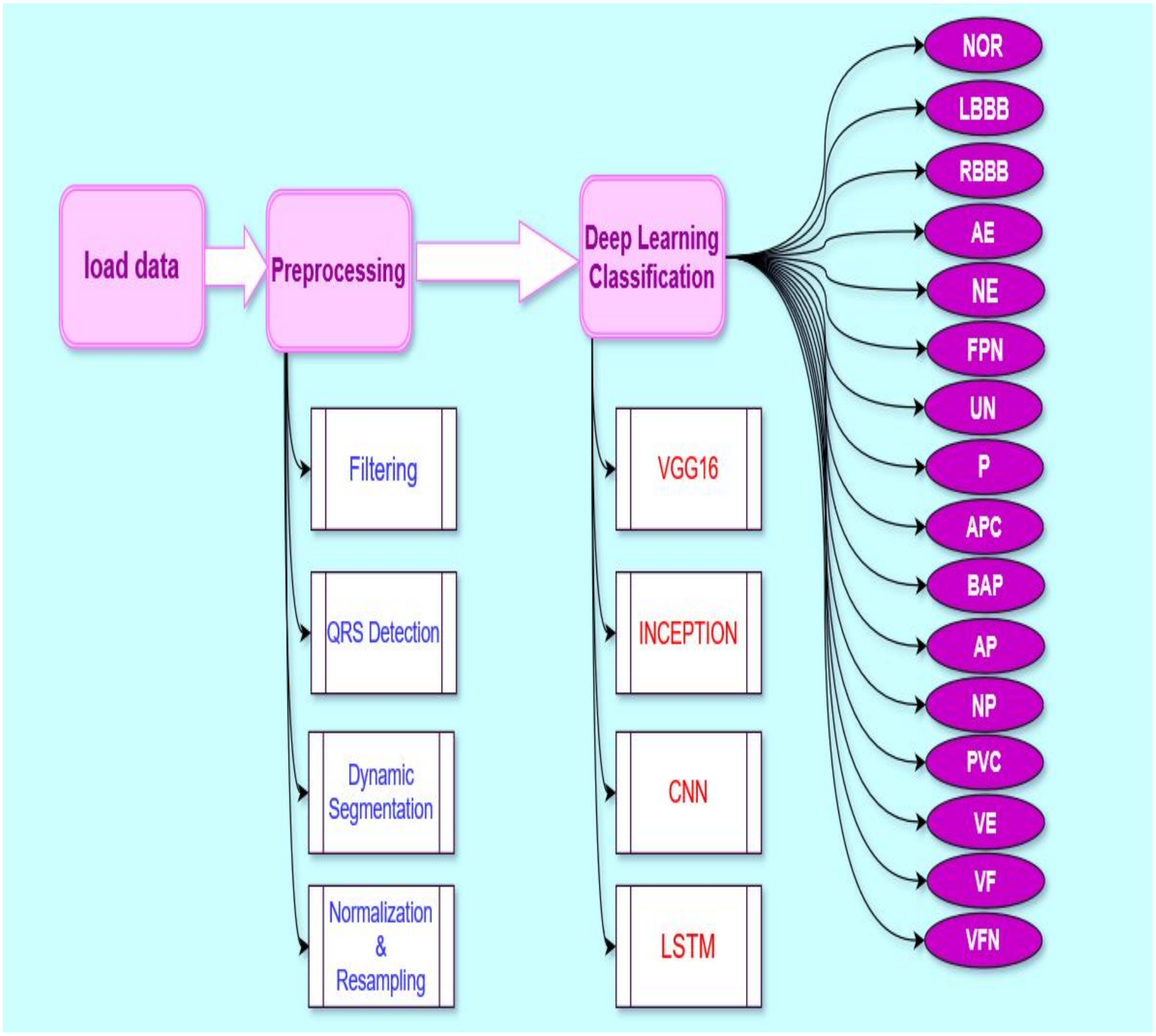


Figure 3.2 System Architecture for one stage

→ **this figure show architecture for Two stage.**

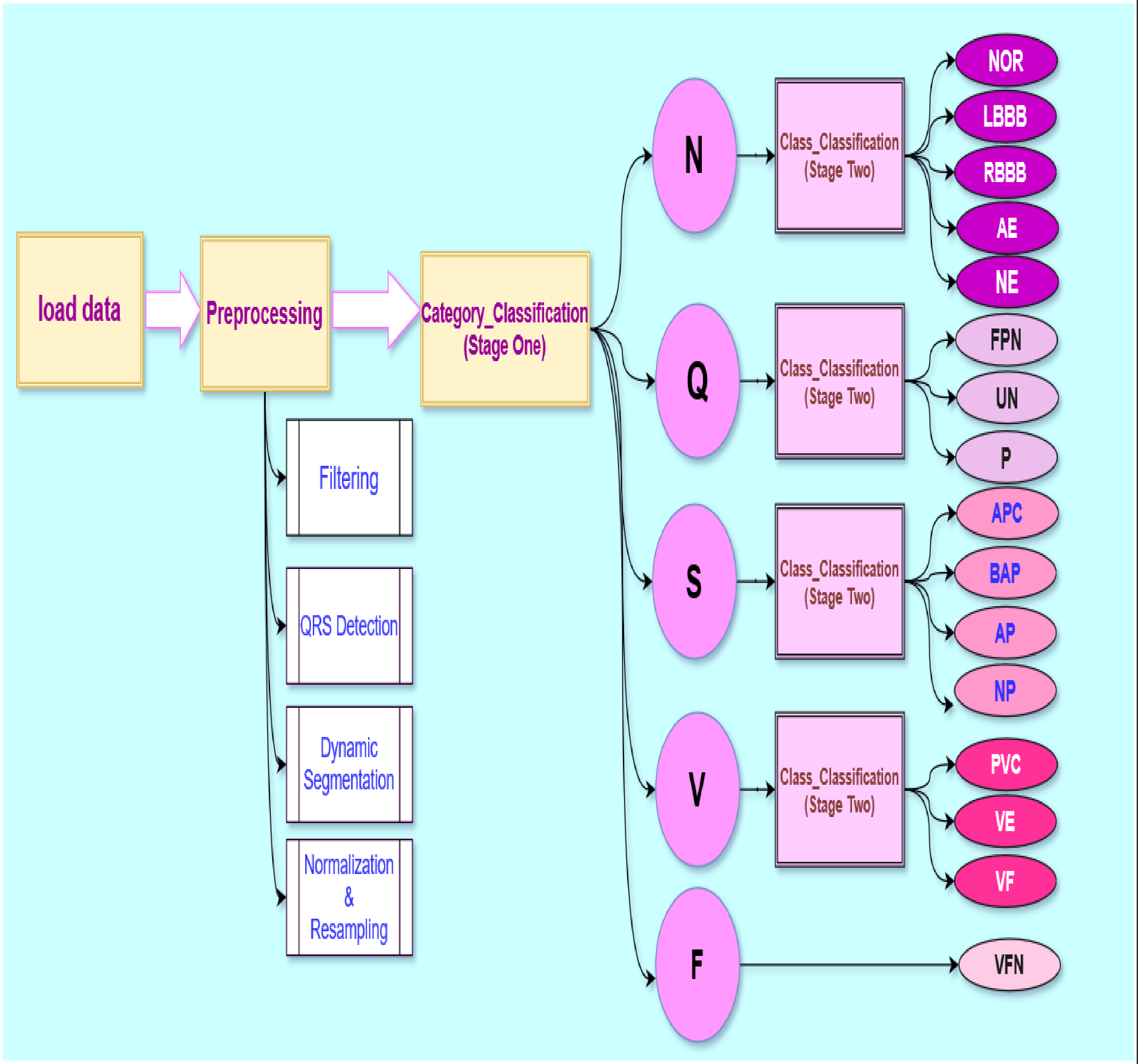

load data
Preprocessing
Category_Classification (Stage One)
Filtering
QRS Detection
Dynamic Segmentation
Normalization & Resampling
N
Q
S
V
F
Class_Classification (Stage Two)
Class_Classification (Stage Two)
Class_Classification (Stage Two)
Class_Classification (Stage Two)
NOR
LBBB
RBBB
AE
NE
FPN
UN
P
APC
BAP
AP
NP
PVC
VE
VF
VFN

Figure 3.3 System Architecture for Two stage

Now we explain each step in two architectures.

3.1.1- Datasets

The MIT-BIH database is the most popular database utilized by the existing studies for training and testing purposes. The MIT-BIH database consists of 48 records, each of them is a 30 min long sample with sampling frequency of 360 Hz but according to the ANSI/AAMI, only 44 records can be utilized as there are four paced records. Moreover, attached with each record a file contains the annotations of the beats and the locations of the R-peaks. The total number of samples for each class is not equally distributed which is why the percentage of the training and testing portions are not the same in all the classes.

3.1.2 Preprocessing

3.1.2.1 filtering:

The aim of this step is to improve the signal-to-noise ratio and extract the heartbeats. Noise has been reduced by removing both high and low frequencies out of the ECG spectra. Hence, Butterworth bandpass filter with a range 0.5–40 Hz is applied.

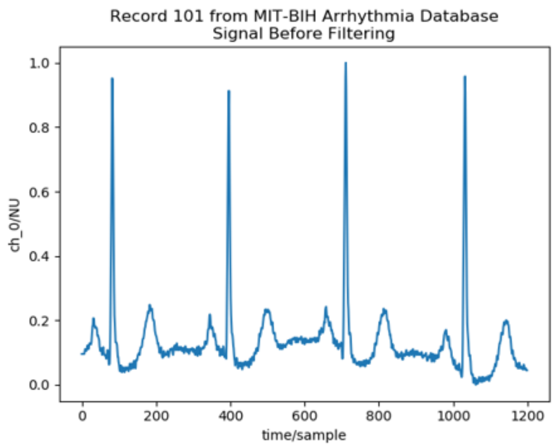


Figure 3.4 Before filter

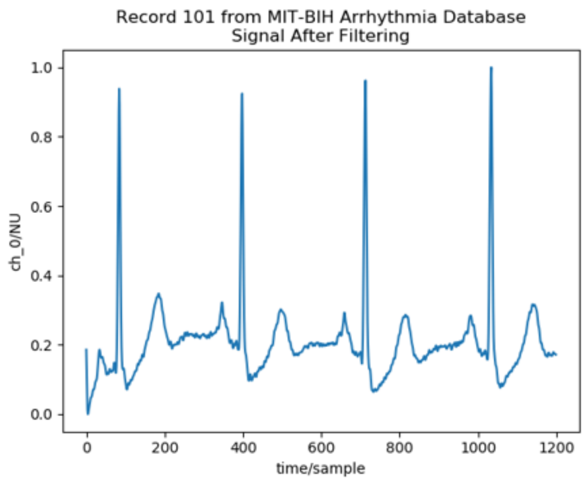


Figure 3.5 After filter

3.1.2.2 QRS Detection:

QRS complex based on the dual criteria of the amplitude and duration of QRS complex. It consists of simple operations, such as a finite impulse response filter, differentiation. The QRS detection performance is evaluated by using an MIT-BIH arrhythmia database In this step we used python libraries to get it more accurate.

figure[3.6] shown an example for R-peak Detection

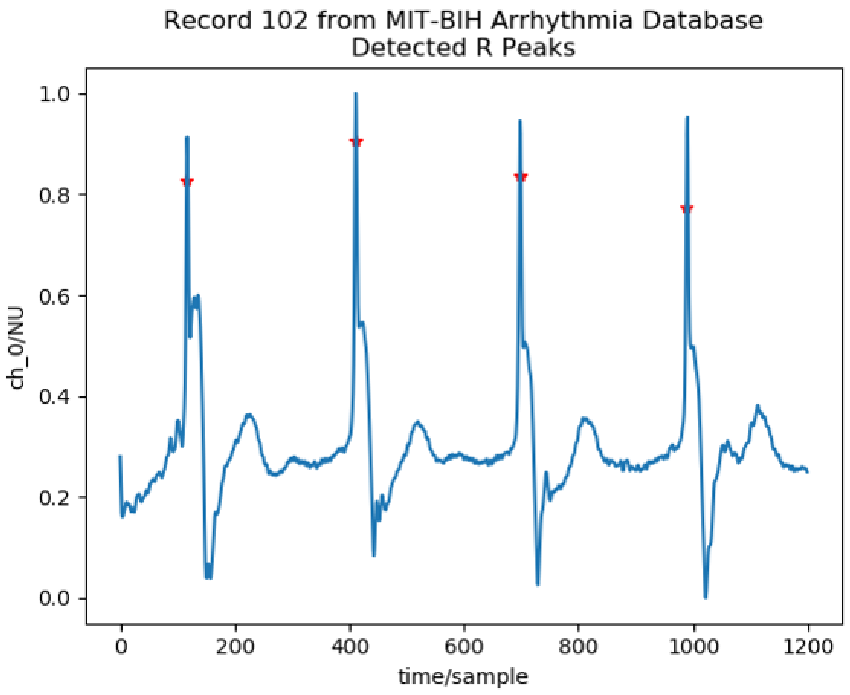


Figure 3.6 Example for R peak Detection

The red stars is the detected R peaks using QRS detection algorithm

3.1.2.3 Dynamic Segmentation:

a dynamic segmentation strategy invariant to heart rate variability. The number of samples considered before and after the R peak is derived according to equation (1)&(2).Fig. 9 shows an example for a segmented heartbeat where the previous RR is the number of samples between the current R peak and the previous one. On the other hand, the next RR is the number of samples
between the current R peak and the next one

Before R peak =

$$1/3 * \text{Max}\,(\text{RR previous, RR next}) \qquad (1)$$

After R peak =

$$2/3 * \text{Max}\,(\text{RR previous, RR next}) \qquad (2)$$

Finally, all the heartbeats are then adjusted to have the same length of 300 samples

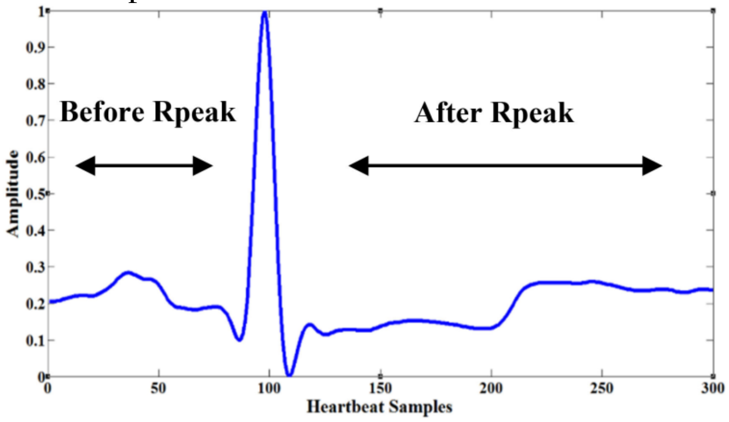


Figure 3.7 Segmented heartbeat using dynamic segmentation strategy

3.1.2.4 Resampling and normalization

- Normalize the raw signal in range (0-1).
- Resampling all segmented beats to have the same sampling rate (300 sample/sec).

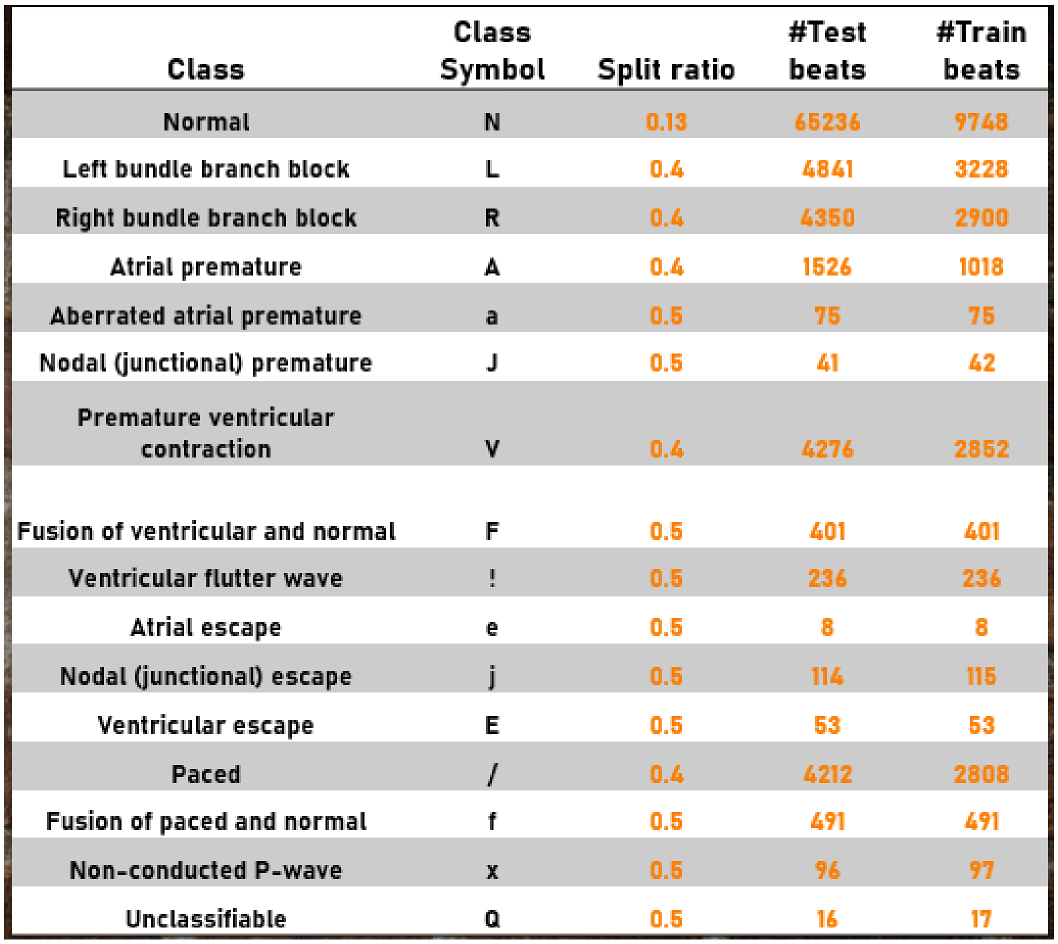
$$[Normalized\ Signal] = \frac{[Signal] - min\ (Signal)}{max(Signal) - min\ (Signal)}$$

Figure 3.8 Normalized signal rule

3.1.3 Testing and training split

3.1.3.1 Beats Type split

In this way, we split data based on beat type. So, we take all records and collect the beats from same class together, after this step we have dataset as Collection of beats and split it to train and test.
In this type from split we follow the method used in El-Saadawy et al. [7] 2018, this split data as the following table



Class	Class Symbol	Split ratio	#Test beats	#Train beats
Normal	N	0.13	65236	9748
Left bundle branch block	L	0.4	4841	3228
Right bundle branch block	R	0.4	4350	2900
Atrial premature	A	0.4	1526	1018
Aberrated atrial premature	a	0.5	75	75
Nodal (junctional) premature	J	0.5	41	42
Premature ventricular contraction	V	0.4	4276	2852
Fusion of ventricular and normal	F	0.5	401	401
Ventricular flutter wave	!	0.5	236	236
Atrial escape	e	0.5	8	8
Nodal (junctional) escape	j	0.5	114	115
Ventricular escape	E	0.5	53	53
Paced	/	0.4	4212	2808
Fusion of paced and normal	f	0.5	491	491
Non-conducted P-wave	x	0.5	96	97
Unclassifiable	Q	0.5	16	17

Table 3.1 Training and testing percentages &Number of beats used

3.1.3.2 Subject orientation record split

Split data in this type based on record,
In this type from split we follow the method used in [13] 2004, this split data as 50% tarin and test.

3.1.4 Classification model for one stage

From the survey we did before, determine we will use deep learning model to classify Heartbeat. For that We did more tries using different deep learning models and Recurrent neural network (RNN). **In one stage we used this model to know beat belong to which class**

3.1.4.1 VGG model

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford.

They proposed total 5 configurations, named as A-E. But VGG16 and VGG19 are famous. The main concept is stacking of convolutional layers to create deep neural networks.

- It is a very good architecture for benchmarking on a particular task.
- pre-trained networks for VGG are available freely on the internet, so it is commonly used out of the box for various applications.
- It has so many weight parameters, the models are very heavy, 550 MB + of weight size.
- Which also means long inference time

We try this by this architecture in fig [3.9-3.10]

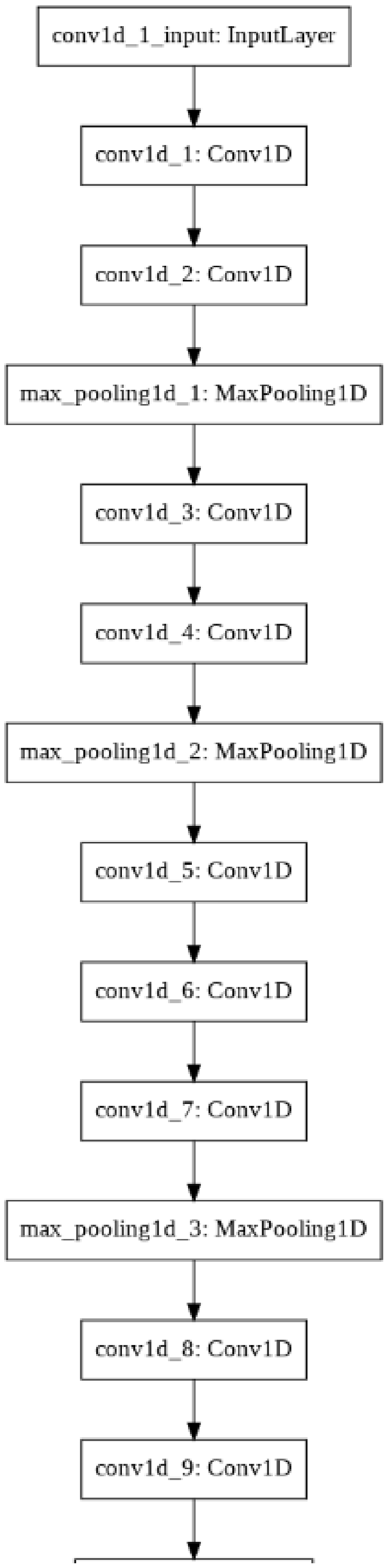


Figure 3.9 VGG16 Architecture(part1)

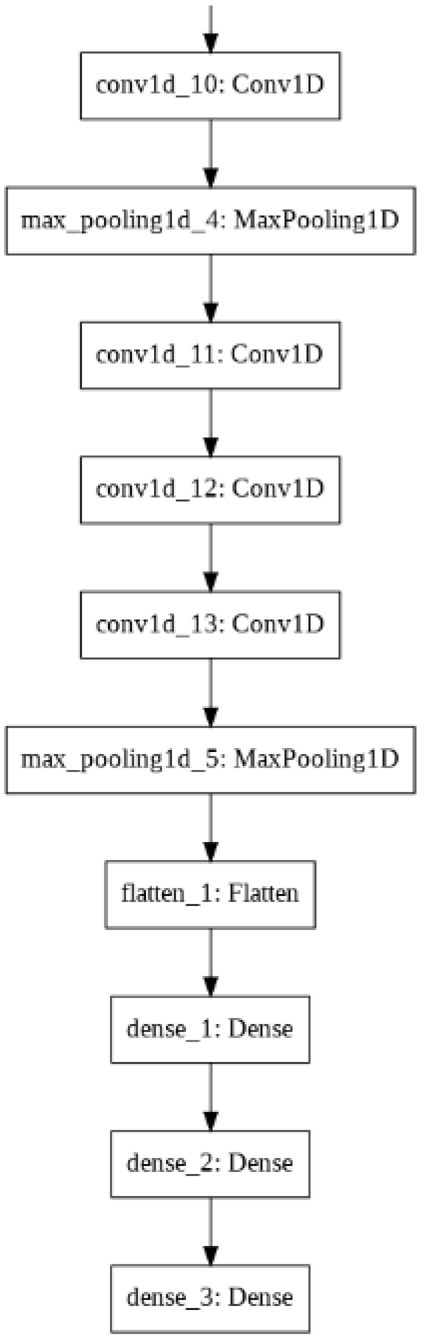


Figure 3.10 VGG16 Architecture(part2)

3.1.4.2 LSTM model

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way [14]

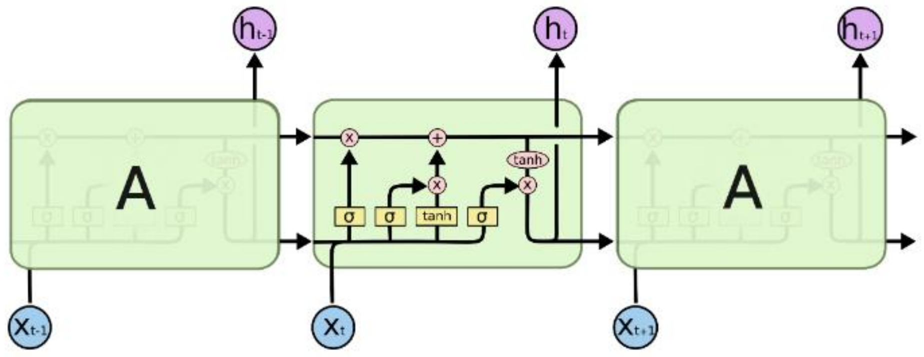


Figure 3.11. The repeating module in an LSTM contains four interacting layers

We used in our architecture several lstm with different units to save more information as shown in [13]

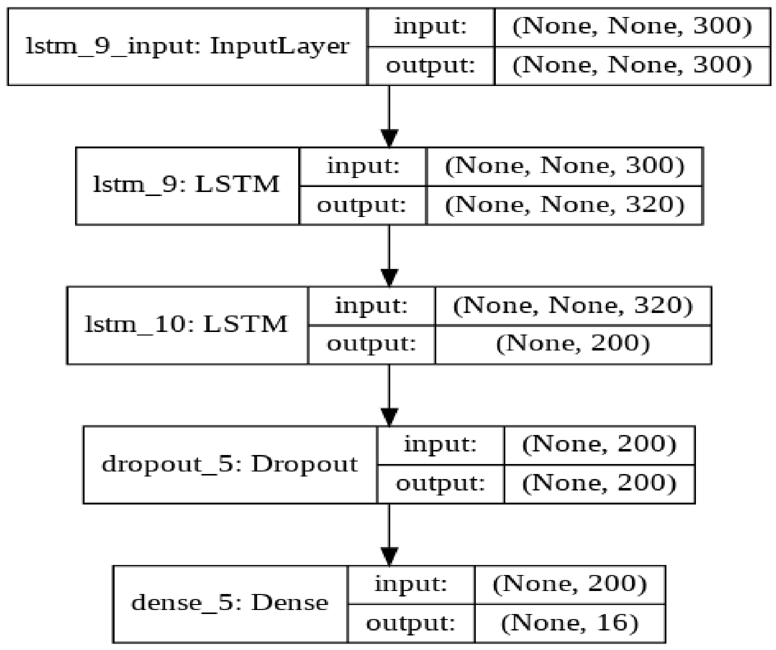


Figure 3.12. our LSTM Architecture

3.1.4.3 CNN model

A CNN is composed of layers that filters(convolve) the inputs to get useful information. These have two kinds of layers: convolution layers and pooling layers. The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.

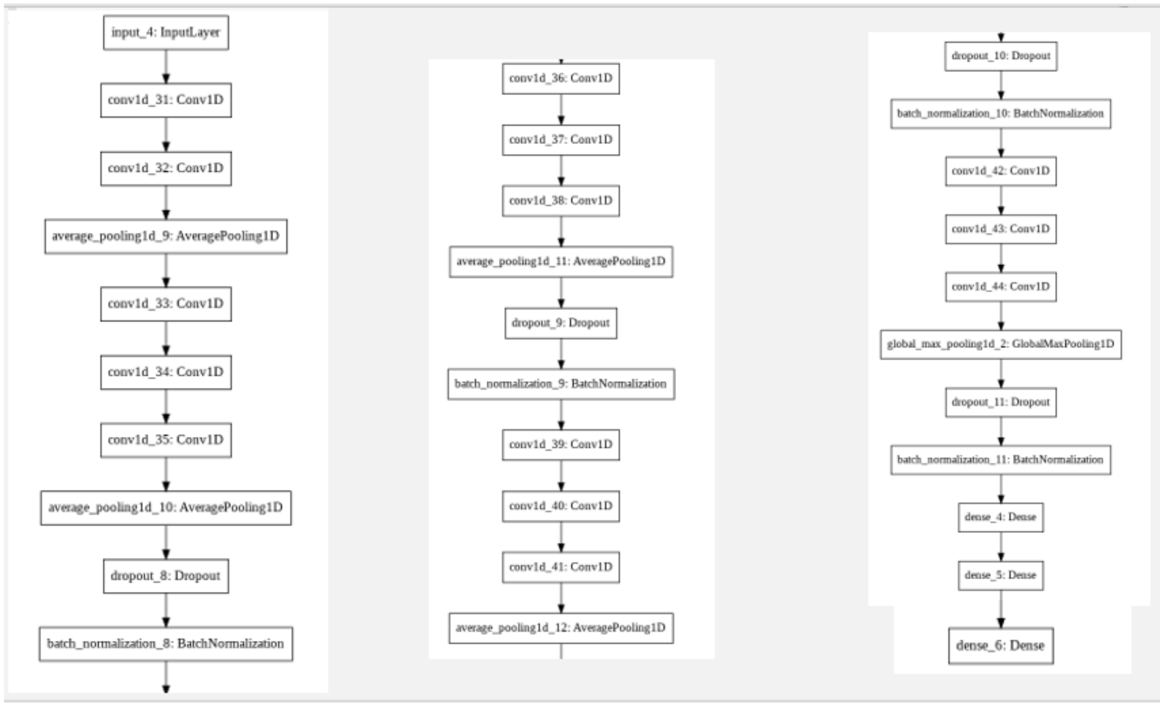


Figure 3.13. our CNN Architecture

We try more than one CNN architecture on data.

Figure.15 shows an architecture that give us best accuracies.

3.1.4.4 inception model

- Inception Modules are used in CNN to allow for more efficient computation and deeper Networks through a dimensionality reduction with stacked 1×1 convolution.
- The main difference between the Inception models and regular CNNs are the inception blocks. These involve convolving the same input tensor with multiple filters and concatenating their result

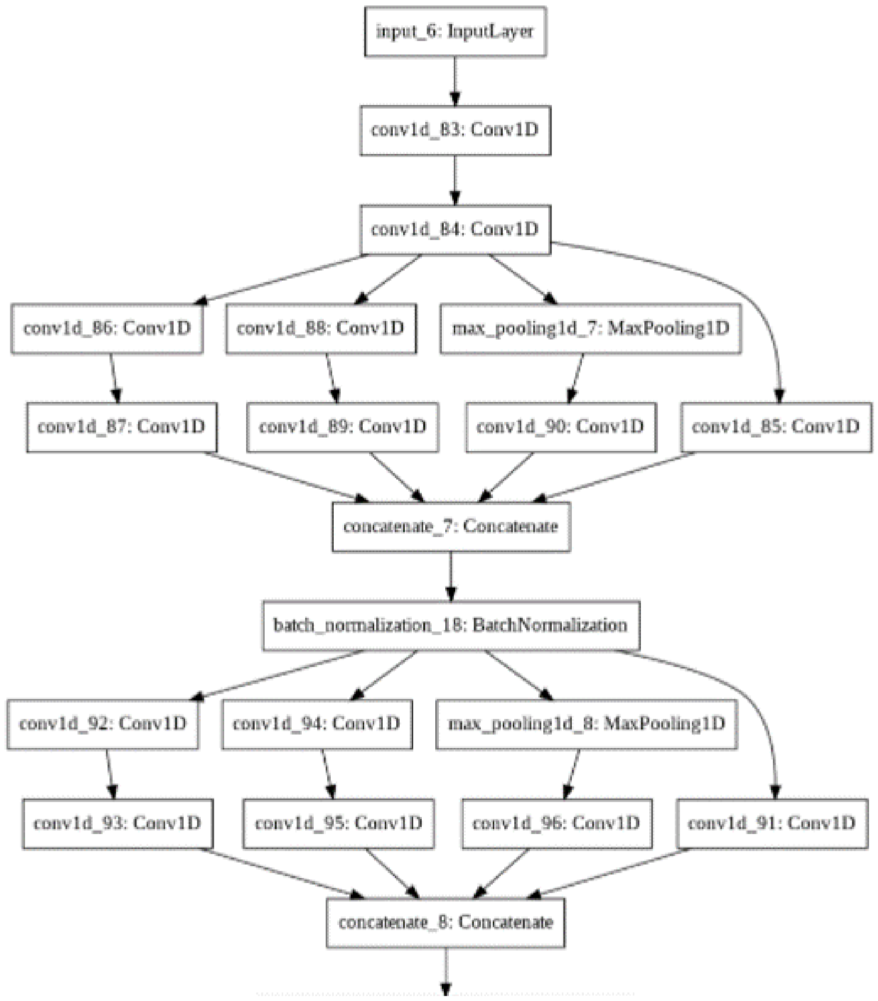


Figure 3.14. custom inception Architecture (part 1)

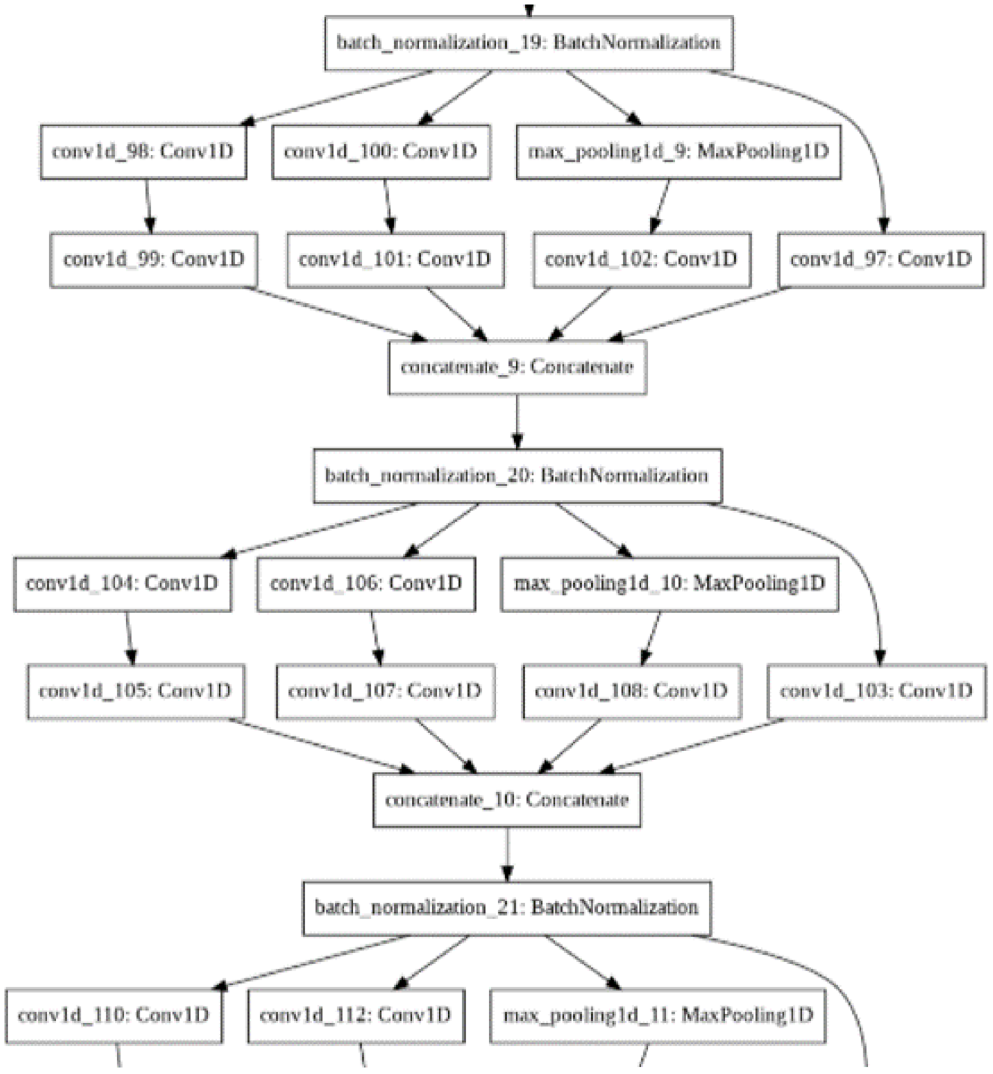

batch_normalization_19: BatchNormalization
conv1d_98: Conv1D
conv1d_100: Conv1D
max_pooling1d_9: MaxPooling1D
conv1d_99: Conv1D
conv1d_101: Conv1D
conv1d_102: Conv1D
conv1d_97: Conv1D
concatenate_9: Concatenate
batch_normalization_20: BatchNormalization
conv1d_104: Conv1D
conv1d_106: Conv1D
max_pooling1d_10: MaxPooling1D
conv1d_105: Conv1D
conv1d_107: Conv1D
conv1d_108: Conv1D
conv1d_103: Conv1D
concatenate_10: Concatenate
batch_normalization_21: BatchNormalization
conv1d_110: Conv1D
conv1d_112: Conv1D
max_pooling1d_11: MaxPooling1D

Figure 3.15. custom inception Architecture (part 2)

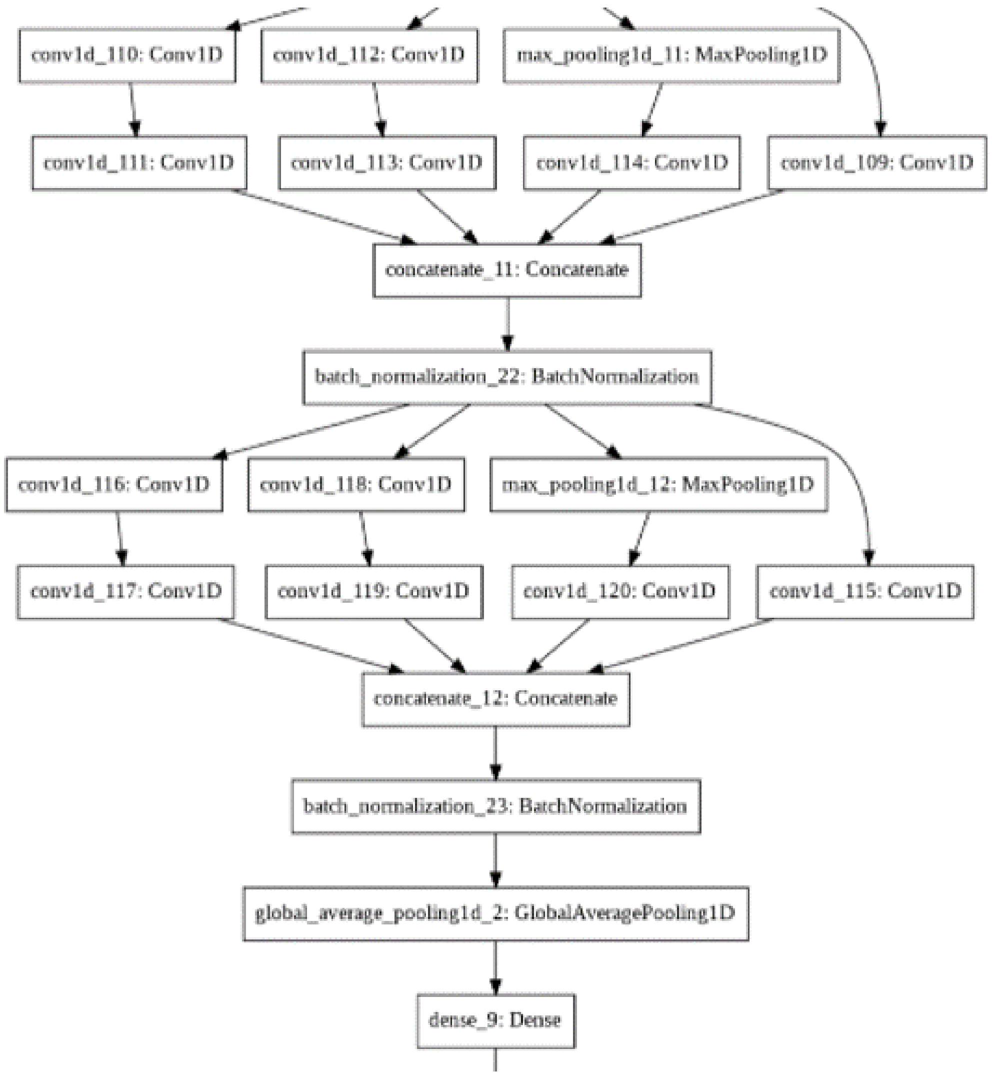

conv1d_110: Conv1D
conv1d_112: Conv1D
max_pooling1d_11: MaxPooling1D
conv1d_111: Conv1D
conv1d_113: Conv1D
conv1d_114: Conv1D
conv1d_109: Conv1D
concatenate_11: Concatenate
batch_normalization_22: BatchNormalization
conv1d_116: Conv1D
conv1d_118: Conv1D
max_pooling1d_12: MaxPooling1D
conv1d_117: Conv1D
conv1d_119: Conv1D
conv1d_120: Conv1D
conv1d_115: Conv1D
concatenate_12: Concatenate
batch_normalization_23: BatchNormalization
global_average_pooling1d_2: GlobalAveragePooling1D
dense_9: Dense

Figure 3.16. custom inception Architecture (part 3)

We follow inception way by make our custom inception after many tries to get best result from this model

In the next chapter, we will show Our Architecture models in detail.

3.1.5 classification for Two stage

A two-stage hierarchical classification process is introduced. In the first stage, the heartbeat is assigned to one of the five main categories as mentioned in Table [1]. Subsequently, the heartbeat is further classified into one of the classes that belong to its category known from the previous stage as shown in Figure [4].
For doing this type of classification we did the same steps and used the previous explained models has been used in first stage for classification.

Chapter Four: Implementation

4.1 Reading Dataset

We read each record in the dataset using **wfdb** python package and we can read record annotations (R Peaks indices and Beats Types) using the same package

wfdb Package A library of tools for reading, writing, and processing WFDB signals and annotations.

Core components of this package are based on the original WFDB specifications. This package does not contain the exact same functionality as the original WFDB package. It aims to implement as many of its core features as possible, with user-friendly APIs. Additional useful physiological signal-processing tools are added over time.

4.2 Preprocessing steps

These steps are applied to signal before training and also applied to any test signal

- Filtering Signal
- Normalization
- Detecting R peaks Indices Algorithm
- Dynamic Segmentation
- Resampling

Will mention Below in Details

4.2.1 Filtering Signal

Filtering signal to remove noise from the signal
Filtering using butter worth band pass filter to save ECG band (0-40)

The **Butterworth filter** is a type of signal processing **filter** designed to have a frequency response as flat as possible in the the **passband**. It is also referred to as a maximally flat magnitude **filter**.

Butter band pass filter included from **scipy** library

SciPy is a library that uses **NumPy** for more mathematical functions. **SciPy** uses **NumPy** arrays as the basic data structure, and comes with modules for various commonly used tasks in scientific programming

4.2.2 Normalization

Normalize signal values from (0-1) to make all input signal in the same range of amplitude.

Using Normalization Formula:

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Where min (x) is minimum Value of Signal and max (x)
Is maximum Value of Signal

4.2.3 Detecting R Peaks indices Algorithm

Using wfdb.processing python package.
1- load the signal and configuration of parameters.
2- bandpass filter the signal between 5 and 20 hz, to get the filtered signal.
3- apply moving wave integration (mwi) with a ricker (mexican hat) wavelet onto the filtered signal, and save the square of the integrated signal.
4- conduct learning if specified, to initialize running parameters of noise and qrs amplitudes, the qrs detection threshold, and recent rr intervals. If learning is unspecified or fails, use default parameters.
5- run the main detection. Iterate through the local maxima of the mwi signal. For each local maximum
6- check if it is a qrs complex. To be classified as a qrs.
7- if not a qrs, classify it as a noise peak and update running parameters.
8- before continuing to the next local maxima, if no qrs was detected within 1.66 times the recent rr interval, perform back search qrs detection.

4.2.4 Dynamic Segmentation

Segment ECG signal into heartbeats using R Peaks indices
Dynamic not static as segmentation is based on RR interval, throw RR interval we have three complex waves of heartbeat (P QRS T) which guarantees that beat will be full-featured.

4.2.5 Resampling

Take each segmented heartbeat and resample it to 300 sample/sec
Using scipy Library

4.3 Calculating Accuracy of (Detecting R Peaks Algorithm)

- Iterate on each record in the dataset
    * Read annotations (R Peaks indices).
    * Calculate R Peaks indices using Algorithm mentioned above
    * Check difference in R Peaks count in all record
    * Calculate accuracy for count
      (total beats with algorithm)/actual total beats * 100
    * Calculate shift Average
      (avg difference between actual index and index calculated by the algorithm)

And we got 98.86 % count accuracy and total shift average = 1.51

4.4 Prepare Training Data

4.4.1 Beat Type Split

This step is done by reading all records from MITBIH dataset and all annotations,
Then apply preprocessing steps on each signal record
- Segment each record to beats using R peaks indices loaded from annotation file.
- Cluster each beat type together then write text file includes all beats for each class.
- Then read all text files and split each class to train and test.
- Then concatenate train data and test data together.
- We have followed train test split ratio like [**7**].

In this type from split we follow the method used in El-Saadawy et al. [7] 2018, this split data as shown in table[3.1]

Training and Testing Data Shapes for one stage and two stages

All x train (24081, 300, 1) y train (24081, 16)
All x test (85980, 300, 1) y test (85980, 16)

N category x train (15996, 300, 1) y train (15996, 5)
N category x test (74552, 300, 1) y test (74552, 5)

S category x train (1229, 300, 1), y train (1229, 4)
S category x test (1741, 300, 1), y test (1741, 4)

V category x train (3140, 300, 1), y train (3140, 3)
V category x test (4566, 300, 1), y test (4566, 3)

F category x train (401, 300, 1), x test (401, 300 ,1)

Q category x train (3315, 300, 1), y train (3315, 3)
Q category x test (4720, 300, 1), y test (4720, 3)

4.4.2 Data Augmentation

Augmentation Done using SMOTE (Synthetic Minority Oversampling Technique) – Oversampling

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem.
It aims to balance class distribution by randomly increasing minority class examples by replicating them.
SMOTE synthesizes new minority instances between existing minority instances. It generates the **virtual training records by linear interpolation** for the minority class. These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class. After the oversampling process, the data is reconstructed, and several classification models can be applied for the processed data [15]

Training Data Shapes Before Generate sample and after

- **All Data**
  Before : X train (24081, 300, 1) Y Train (24081, 16)
  After : X Train (155952, 300, 1) Y Train (155952, 16)
- **Two Stage Data**

  First stage
  Before: X train (24081, 300, 1) Y Train (24081, 16)
  After: X Train (79980, 300, 1) Y Train (79980, 16)

  Category N
  Before: X train (15996, 300, 1) Y Train (15996, 16)
  After: X Train (48735, 300, 1) Y Train (48735, 16)

  Category S
  Before: X train (1229, 300, 1) Y Train (1229, 16)
  After: X Train (4068, 300, 1) Y Train (4068, 16)

  Category V
  Before: X train (3140, 300, 1) Y Train (3140, 16)
  After: X Train (8553, 300, 1) Y Train (8553, 16)

  Category Q
  Before: X train (3315, 300, 1) Y Train (3315, 16)
  After: X Train (8424, 300, 1) Y Train (8424, 16)

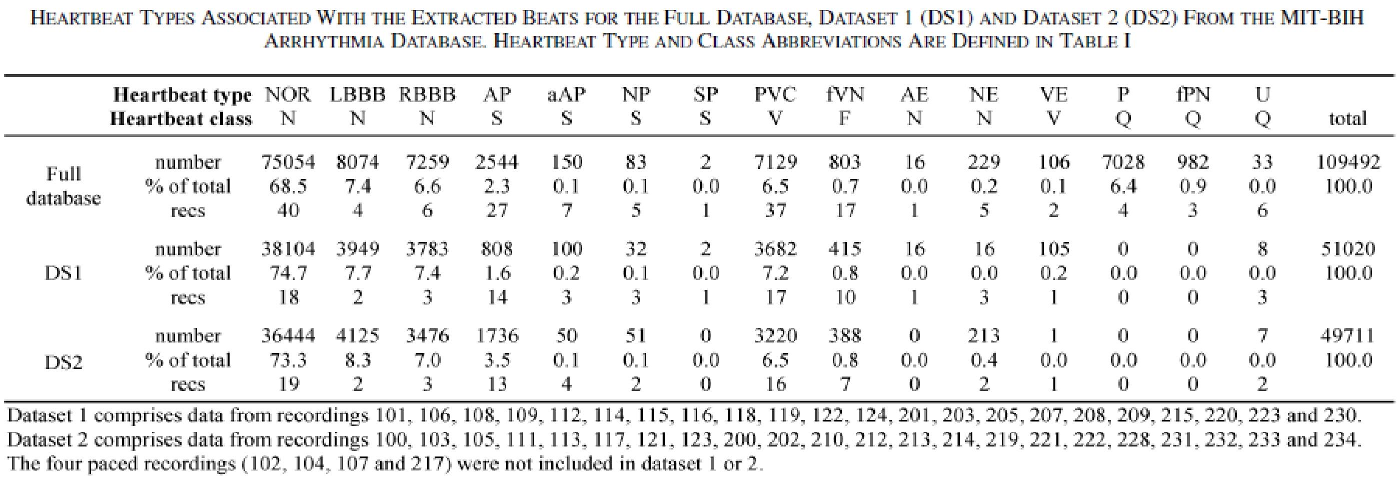
4.4.3 Subject Oriented Record Split

This is done by considering train records and test record
subject oriented means that we train the model with record and test with other records.
For each train and test record

- Read all records
- Segment each record into beats based on R Peaks indices annotations
- Cluster beats based on their classes.
- Write each class to text file

- Read text file for each class the concatenate all data together.

In this type from split we follow the method used in [13] 2004 this split data as 50% train and test



HEARTBEAT TYPES ASSOCIATED WITH THE EXTRACTED BEATS FOR THE FULL DATABASE, DATASET 1 (DS1) AND DATASET 2 (DS2) FROM THE MIT-BIH ARRHYTHMIA DATABASE. HEARTBEAT TYPE AND CLASS ABBREVIATIONS ARE DEFINED IN TABLE I

	Heartbeat type	NOR	LBBB	RBBB	AP	aAP	NP	SP	PVC	fVN	AE	NE	VE	P	fPN	U	
	Heartbeat class	N	N	N	S	S	S	S	V	F	N	N	V	Q	Q	Q	total
Full database	number	75054	8074	7259	2544	150	83	2	7129	803	16	229	106	7028	982	33	109492
	% of total	68.5	7.4	6.6	2.3	0.1	0.1	0.0	6.5	0.7	0.0	0.2	0.1	6.4	0.9	0.0	100.0
	recs	40	4	6	27	7	5	1	37	17	1	5	2	4	3	6	
DS1	number	38104	3949	3783	808	100	32	2	3682	415	16	16	105	0	0	8	51020
	% of total	74.7	7.7	7.4	1.6	0.2	0.1	0.0	7.2	0.8	0.0	0.0	0.2	0.0	0.0	0.0	100.0
	recs	18	2	3	14	3	3	1	17	10	1	3	1	0	0	3	
DS2	number	36444	4125	3476	1736	50	51	0	3220	388	0	213	1	0	0	7	49711
	% of total	73.3	8.3	7.0	3.5	0.1	0.1	0.0	6.5	0.8	0.0	0.4	0.0	0.0	0.0	0.0	100.0
	recs	19	2	3	13	4	2	0	16	7	0	2	1	0	0	2	

Dataset 1 comprises data from recordings 101, 106, 108, 109, 112, 114, 115, 116, 118, 119, 122, 124, 201, 203, 205, 207, 208, 209, 215, 220, 223 and 230.
Dataset 2 comprises data from recordings 100, 103, 105, 111, 113, 117, 121, 123, 200, 202, 210, 212, 213, 214, 219, 221, 222, 228, 231, 232, 233 and 234.
The four paced recordings (102, 104, 107 and 217) were not included in dataset 1 or 2.

Figure 4.1 Subject oriented record split

Dataset 1 is training set.
Dataset 2 is testing set.

4.5 Deep Learning Models

To test efficiency for each model we test it on **5** trails as minimum

Evaluate model depend on its results in **overall accuracy** and **Average Accuracy** for all classes and all Trails we will improve some Graphs to express this results after each model

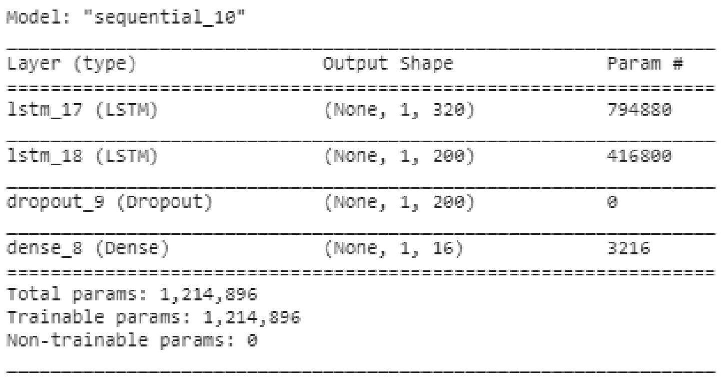
We use keras deep learning Framework to include all models

Keras is an open-source neural-network library written in Python. It can run on top of **TensorFlow**, Microsoft Cognitive Toolkit, R, **Theano**, or **PlaidML**. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

4.5.1 LSTM

- **Long Short-Term Memory networks** – usually just called "**LSTMs**" – are a special kind of **RNN**, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems and are now widely used. [14]
- **LSTMs** are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

  All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way [14]
- Our model summary using **LSTM** from **Keras** framework



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_17 (LSTM)               (None, 1, 320)            794880
_________________________________________________________________
lstm_18 (LSTM)               (None, 1, 200)            416800
_________________________________________________________________
dropout_9 (Dropout)          (None, 1, 200)            0
_________________________________________________________________
dense_8 (Dense)              (None, 1, 16)             3216
=================================================================
Total params: 1,214,896
Trainable params: 1,214,896
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.2 LSTM Summary

- Using after LSTMs one dropout layer with Fraction of the input units to drop = 0.1
- Then Traditional Dense Layer With Activation Function Softmax to assigns decimal probabilities to each class from 16 class in our multi-class problem , Those decimal probabilities must add up to 1.0. This

additional constraint helps training converge more quickly than it otherwise would.

Parameters:

- Total Parameters: 1,214,896 parameters
- Trainable parameters: 1,214,896 parameters
- Non trainable parameters: 0 parameter

4.5.2 VGG16 1D

General approach:

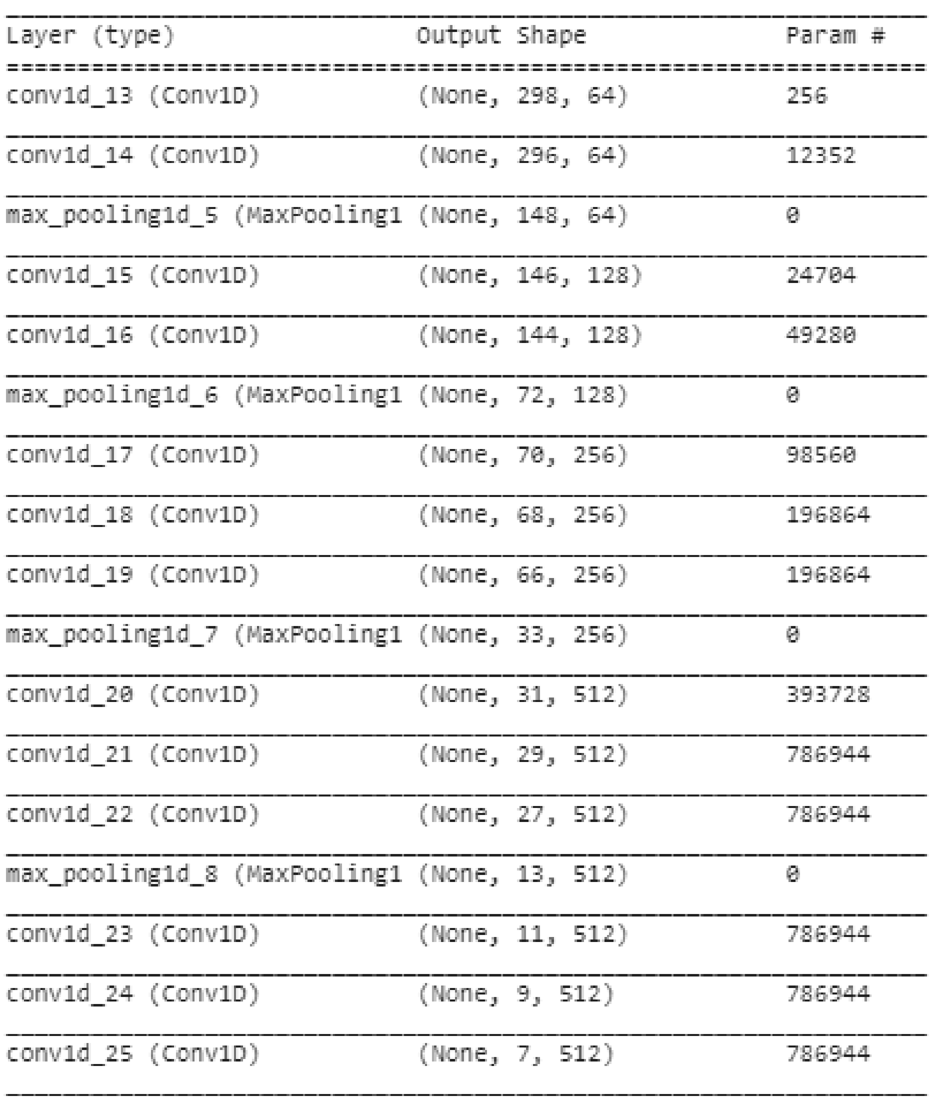
VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's [16]

For Signals

This model proposed for images but we need to try it on Signals this is Done by follow same Architecture by using one dimension Convolution Neural network instead of Two dimension Convolution Neural Network and using one dimension Max polling layer instead of Two Dimension Max Polling

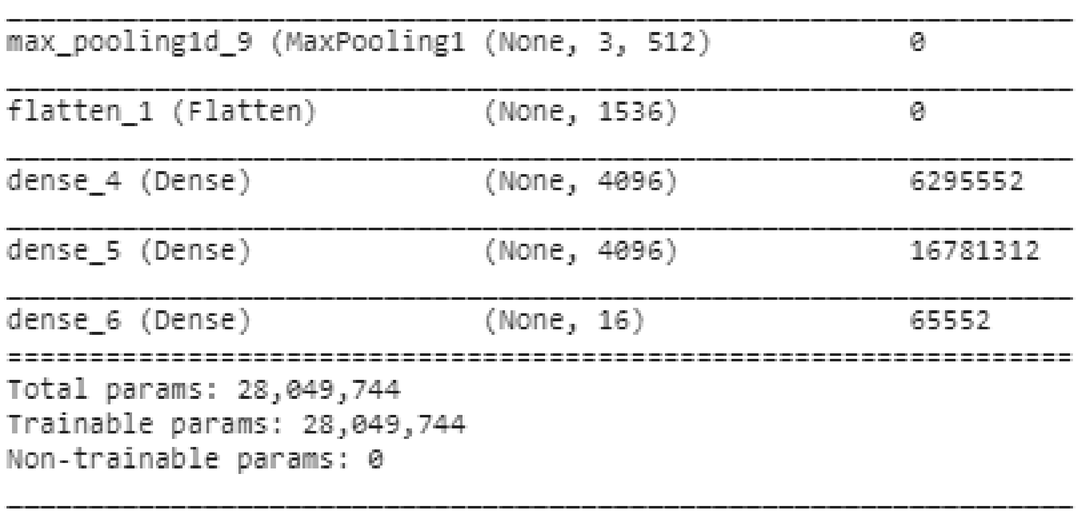
So this Model done without using built in model in keras frame work like **LSTM** just build sequential model and follow the Same layers of **VGG16**

- **Model Summary**



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_13 (Conv1D)           (None, 298, 64)           256       
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 296, 64)           12352     
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 148, 64)           0         
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 146, 128)          24704     
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 144, 128)          49280     
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 72, 128)           0         
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 70, 256)           98560     
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 68, 256)           196864    
_________________________________________________________________
conv1d_19 (Conv1D)           (None, 66, 256)           196864    
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 33, 256)           0         
_________________________________________________________________
conv1d_20 (Conv1D)           (None, 31, 512)           393728    
_________________________________________________________________
conv1d_21 (Conv1D)           (None, 29, 512)           786944    
_________________________________________________________________
conv1d_22 (Conv1D)           (None, 27, 512)           786944    
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 13, 512)           0         
_________________________________________________________________
conv1d_23 (Conv1D)           (None, 11, 512)           786944    
_________________________________________________________________
conv1d_24 (Conv1D)           (None, 9, 512)            786944    
_________________________________________________________________
conv1d_25 (Conv1D)           (None, 7, 512)            786944    
_________________________________________________________________
```

Figure 4.3 VGG Summary (part 1)



```
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 3, 512)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1536)              0
_________________________________________________________________
dense_4 (Dense)              (None, 4096)              6295552
_________________________________________________________________
dense_5 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_6 (Dense)              (None, 16)                65552
=================================================================
Total params: 28,049,744
Trainable params: 28,049,744
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.5 VGG16 Summary (part 1)

- After all, one-dimension convolution Neural network and one dimension max polling Layers comes the **flatten layer** collapses the spatial dimensions of the input into the channel dimension to make it ready for next Dense Layers
- Then Two traditional Dense Layers with 4096 neurons for each one

- Finally, One Dense Layer with Activation Function SoftMax to assigns decimal probabilities to each class from 16 class in our multi-class problem

Parameters:

- Total Parameters: 28,049,744 parameters
- Trainable parameters: 28,049,744 parameters
- Non trainable parameters: 0 parameter

4.5.3 Inception 1D

The main idea is to make each layer in the model an **inception Module** and the main difference between the Inception models and regular CNNs are the inception blocks

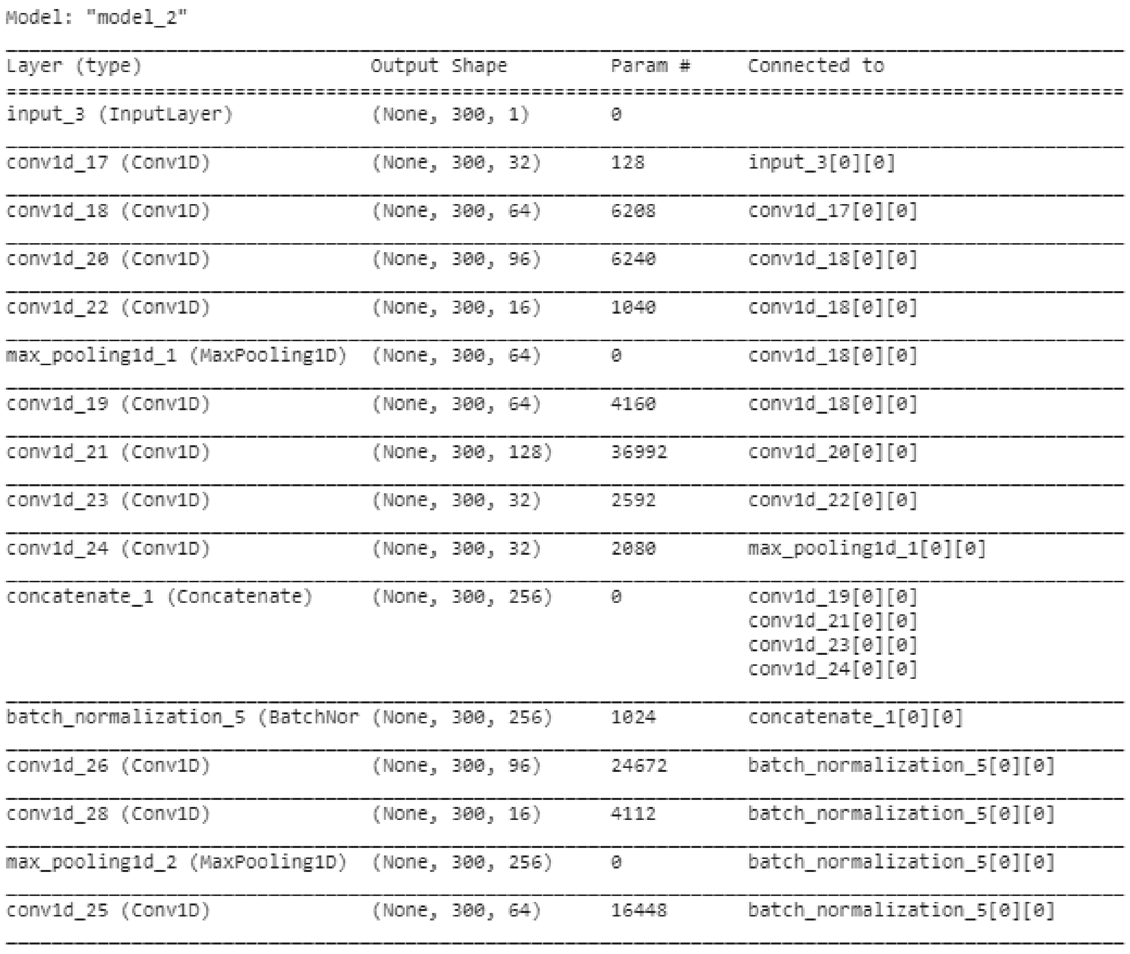
Inception Modules or inception blocks are used in Convolutional Neural Networks to allow for more efficient computation and deeper Networks through a dimensionality reduction with stacked 1×1 convolutions. The modules were designed to solve the problem of computational expense, as well as overfitting, among other issues. The solution, in short, is to take multiple kernel filter sizes within the CNN, and rather than stacking them sequentially, ordering them to operate on the same level. [17]

For Signals

We need to follow same approach with maintaining the appropriate dimensions of the signals.

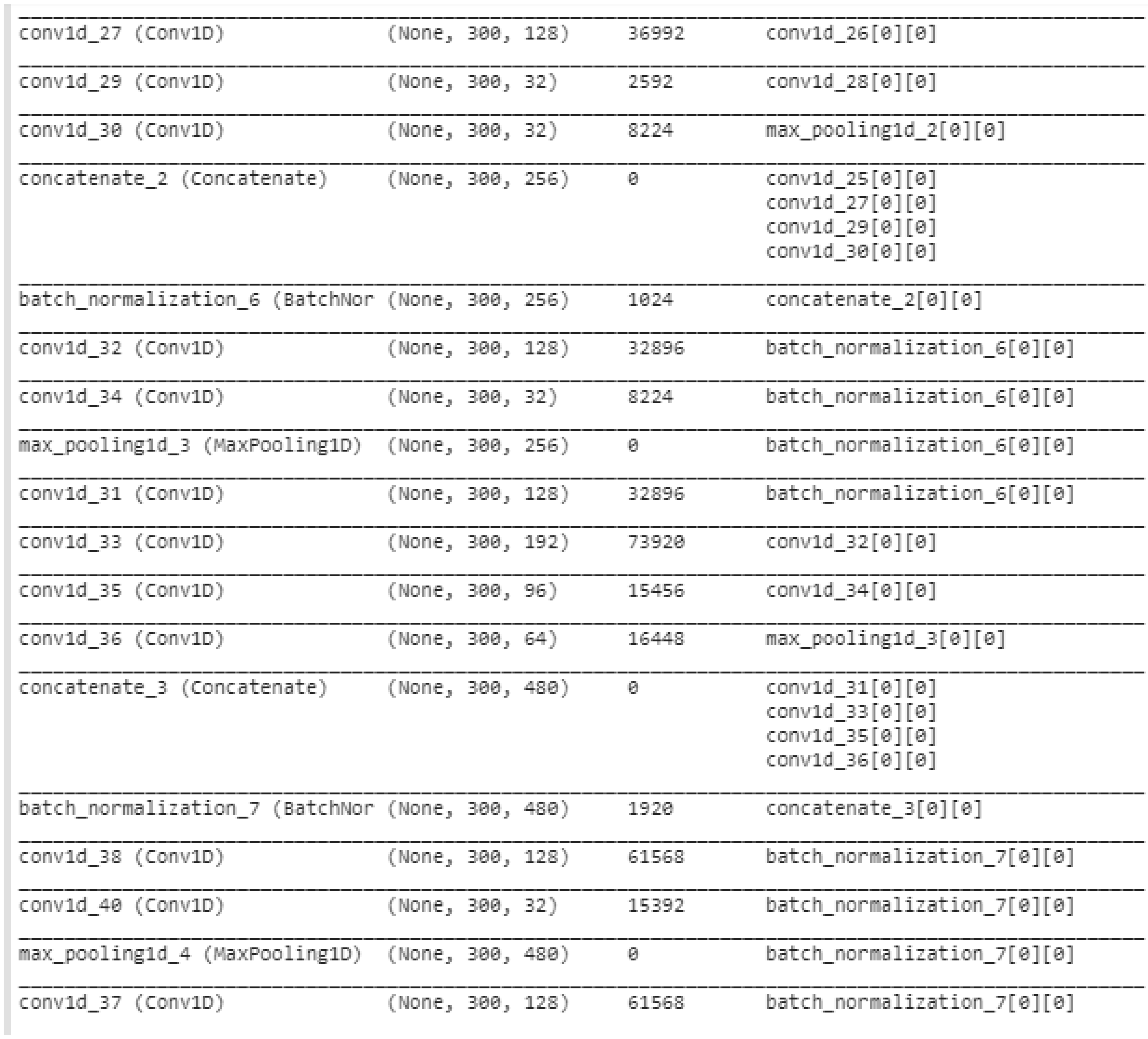
After many and many experiments with different number of blocks and the architecture of the model this is our proposed model summary using inception modules

Model Summary



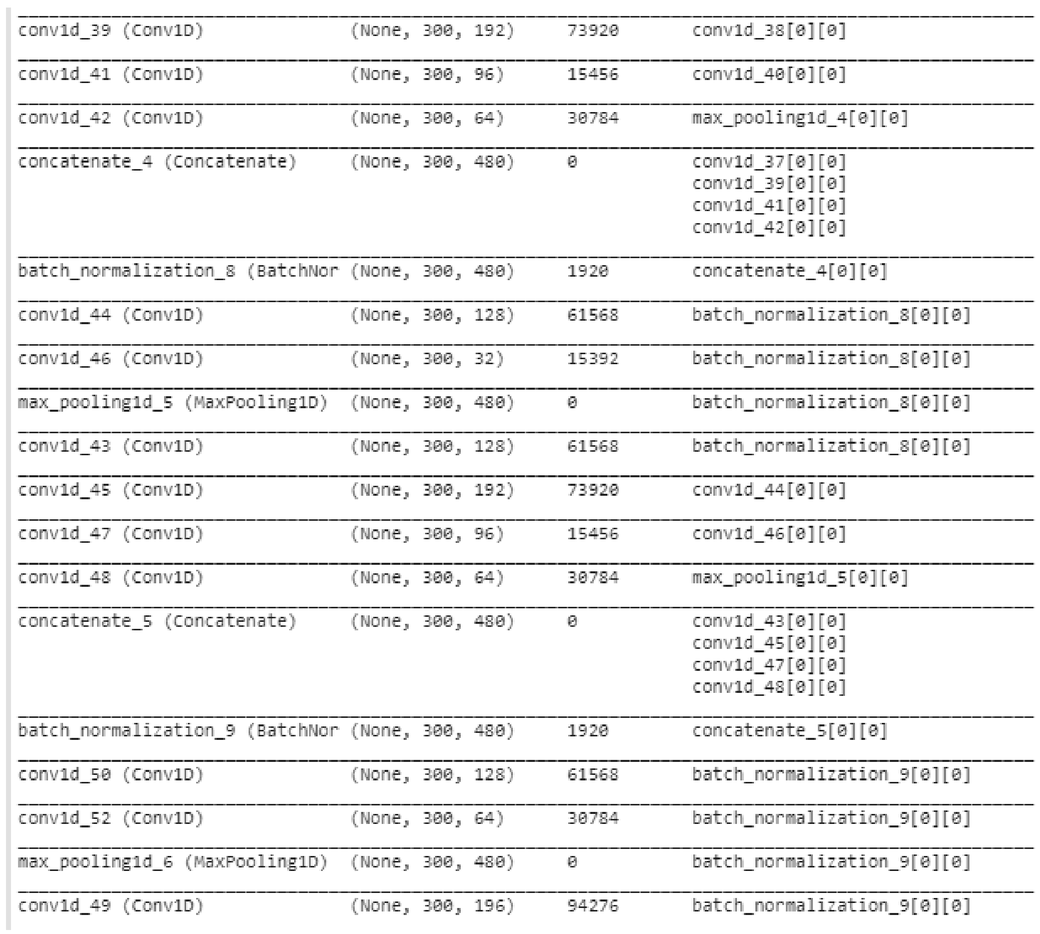
```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            (None, 300, 1)       0
__________________________________________________________________________________________________
conv1d_17 (Conv1D)              (None, 300, 32)      128         input_3[0][0]
__________________________________________________________________________________________________
conv1d_18 (Conv1D)              (None, 300, 64)      6208        conv1d_17[0][0]
__________________________________________________________________________________________________
conv1d_20 (Conv1D)              (None, 300, 96)      6240        conv1d_18[0][0]
__________________________________________________________________________________________________
conv1d_22 (Conv1D)              (None, 300, 16)      1040        conv1d_18[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 300, 64)      0           conv1d_18[0][0]
__________________________________________________________________________________________________
conv1d_19 (Conv1D)              (None, 300, 64)      4160        conv1d_18[0][0]
__________________________________________________________________________________________________
conv1d_21 (Conv1D)              (None, 300, 128)     36992       conv1d_20[0][0]
__________________________________________________________________________________________________
conv1d_23 (Conv1D)              (None, 300, 32)      2592        conv1d_22[0][0]
__________________________________________________________________________________________________
conv1d_24 (Conv1D)              (None, 300, 32)      2080        max_pooling1d_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 300, 256)     0           conv1d_19[0][0]
                                                                 conv1d_21[0][0]
                                                                 conv1d_23[0][0]
                                                                 conv1d_24[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 300, 256)     1024        concatenate_1[0][0]
__________________________________________________________________________________________________
conv1d_26 (Conv1D)              (None, 300, 96)      24672       batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv1d_28 (Conv1D)              (None, 300, 16)      4112        batch_normalization_5[0][0]
__________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 300, 256)     0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv1d_25 (Conv1D)              (None, 300, 64)      16448       batch_normalization_5[0][0]
__________________________________________________________________________________________________
```

Figure 4.6 inception Summary (part 1)

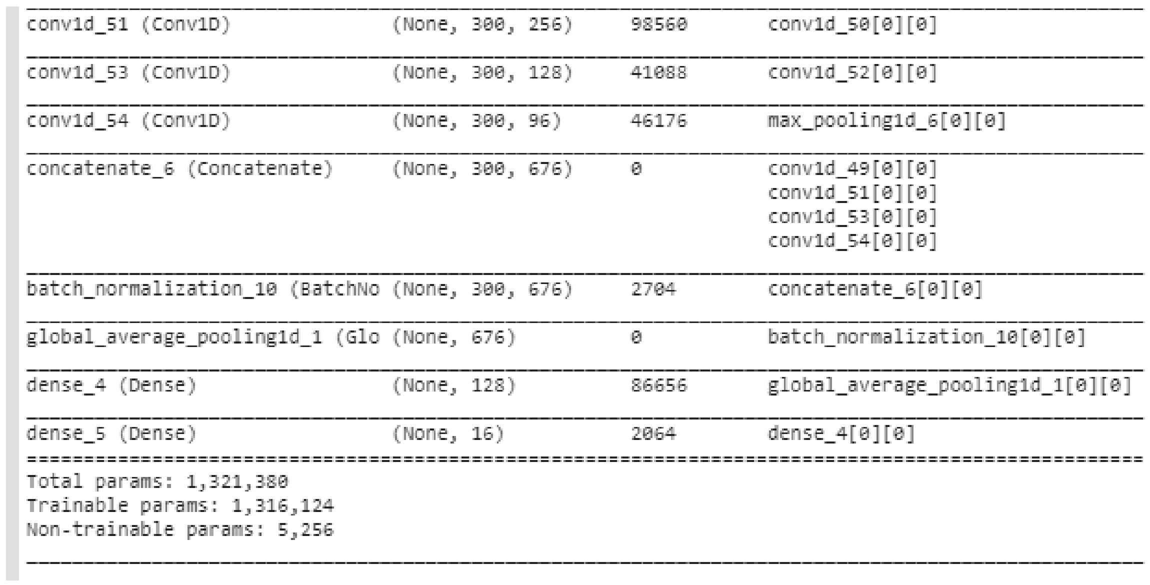


```
__________________________________________________________________________________________________
conv1d_27 (Conv1D)              (None, 300, 128)     36992       conv1d_26[0][0]
__________________________________________________________________________________________________
conv1d_29 (Conv1D)              (None, 300, 32)      2592        conv1d_28[0][0]
__________________________________________________________________________________________________
conv1d_30 (Conv1D)              (None, 300, 32)      8224        max_pooling1d_2[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 300, 256)     0           conv1d_25[0][0]
                                                                 conv1d_27[0][0]
                                                                 conv1d_29[0][0]
                                                                 conv1d_30[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 300, 256)     1024        concatenate_2[0][0]
__________________________________________________________________________________________________
conv1d_32 (Conv1D)              (None, 300, 128)     32896       batch_normalization_6[0][0]
__________________________________________________________________________________________________
conv1d_34 (Conv1D)              (None, 300, 32)      8224        batch_normalization_6[0][0]
__________________________________________________________________________________________________
max_pooling1d_3 (MaxPooling1D)  (None, 300, 256)     0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
conv1d_31 (Conv1D)              (None, 300, 128)     32896       batch_normalization_6[0][0]
__________________________________________________________________________________________________
conv1d_33 (Conv1D)              (None, 300, 192)     73920       conv1d_32[0][0]
__________________________________________________________________________________________________
conv1d_35 (Conv1D)              (None, 300, 96)      15456       conv1d_34[0][0]
__________________________________________________________________________________________________
conv1d_36 (Conv1D)              (None, 300, 64)      16448       max_pooling1d_3[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 300, 480)     0           conv1d_31[0][0]
                                                                 conv1d_33[0][0]
                                                                 conv1d_35[0][0]
                                                                 conv1d_36[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 300, 480)     1920        concatenate_3[0][0]
__________________________________________________________________________________________________
conv1d_38 (Conv1D)              (None, 300, 128)     61568       batch_normalization_7[0][0]
__________________________________________________________________________________________________
conv1d_40 (Conv1D)              (None, 300, 32)      15392       batch_normalization_7[0][0]
__________________________________________________________________________________________________
max_pooling1d_4 (MaxPooling1D)  (None, 300, 480)     0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
conv1d_37 (Conv1D)              (None, 300, 128)     61568       batch_normalization_7[0][0]
```

Figure 4.7 inception Summary (part 2)



```
____________________________________________________________________________________________________
conv1d_39 (Conv1D)              (None, 300, 192)     73920       conv1d_38[0][0]
____________________________________________________________________________________________________
conv1d_41 (Conv1D)              (None, 300, 96)      15456       conv1d_40[0][0]
____________________________________________________________________________________________________
conv1d_42 (Conv1D)              (None, 300, 64)      30784       max_pooling1d_4[0][0]
____________________________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 300, 480)     0           conv1d_37[0][0]
                                                                 conv1d_39[0][0]
                                                                 conv1d_41[0][0]
                                                                 conv1d_42[0][0]
____________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 300, 480)     1920        concatenate_4[0][0]
____________________________________________________________________________________________________
conv1d_44 (Conv1D)              (None, 300, 128)     61568       batch_normalization_8[0][0]
____________________________________________________________________________________________________
conv1d_46 (Conv1D)              (None, 300, 32)      15392       batch_normalization_8[0][0]
____________________________________________________________________________________________________
max_pooling1d_5 (MaxPooling1D)  (None, 300, 480)     0           batch_normalization_8[0][0]
____________________________________________________________________________________________________
conv1d_43 (Conv1D)              (None, 300, 128)     61568       batch_normalization_8[0][0]
____________________________________________________________________________________________________
conv1d_45 (Conv1D)              (None, 300, 192)     73920       conv1d_44[0][0]
____________________________________________________________________________________________________
conv1d_47 (Conv1D)              (None, 300, 96)      15456       conv1d_46[0][0]
____________________________________________________________________________________________________
conv1d_48 (Conv1D)              (None, 300, 64)      30784       max_pooling1d_5[0][0]
____________________________________________________________________________________________________
concatenate_5 (Concatenate)     (None, 300, 480)     0           conv1d_43[0][0]
                                                                 conv1d_45[0][0]
                                                                 conv1d_47[0][0]
                                                                 conv1d_48[0][0]
____________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 300, 480)     1920        concatenate_5[0][0]
____________________________________________________________________________________________________
conv1d_50 (Conv1D)              (None, 300, 128)     61568       batch_normalization_9[0][0]
____________________________________________________________________________________________________
conv1d_52 (Conv1D)              (None, 300, 64)      30784       batch_normalization_9[0][0]
____________________________________________________________________________________________________
max_pooling1d_6 (MaxPooling1D)  (None, 300, 480)     0           batch_normalization_9[0][0]
____________________________________________________________________________________________________
conv1d_49 (Conv1D)              (None, 300, 196)     94276       batch_normalization_9[0][0]
```



```
____________________________________________________________________________________________________
conv1d_51 (Conv1D)              (None, 300, 256)     98560       conv1d_50[0][0]
____________________________________________________________________________________________________
conv1d_53 (Conv1D)              (None, 300, 128)     41088       conv1d_52[0][0]
____________________________________________________________________________________________________
conv1d_54 (Conv1D)              (None, 300, 96)      46176       max_pooling1d_6[0][0]
____________________________________________________________________________________________________
concatenate_6 (Concatenate)     (None, 300, 676)     0           conv1d_49[0][0]
                                                                 conv1d_51[0][0]
                                                                 conv1d_53[0][0]
                                                                 conv1d_54[0][0]
____________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 300, 676)     2704        concatenate_6[0][0]
____________________________________________________________________________________________________
global_average_pooling1d_1 (Glo (None, 676)          0           batch_normalization_10[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                 (None, 128)          86656       global_average_pooling1d_1[0][0]
____________________________________________________________________________________________________
dense_5 (Dense)                 (None, 16)           2064        dense_4[0][0]
====================================================================================================
Total params: 1,321,380
Trainable params: 1,316,124
Non-trainable params: 5,256
____________________________________________________________________________________________________
```

Figure 4.8 inception Summary (part 3)

We have found that when the inputs into the inception modules was Features it achieved better result

So, At the beginning of the model before inception module there are two convolution neural networks to extract some features

- Then six inception modules with different number of parameters
- Between each inception module there is **Batch Normalization** layer

We normalize the input layer by adjusting and scaling the activations. For example, when we have features from 0 to 1 and some from 1 to 1000, we should normalize them to speed up learning. If the input layer is benefiting from it, why not do the same thing also for the values in the hidden inception modules layers , that are changing all the time, and get 10 times or more improvement in the training speed.[18]

- **Batch normalization** reduces the amount by what the hidden unit values shift around (covariance shift).[18]

- **Batch normalization** solve gradient vanishing and exploding problem in our deeper network

so, we use Batch Normalization after every inception Module

- Then using **Global Average Pooling** (GAP) layer to minimize over-fitting and reducing the total number of parameters in the model.

- Then traditional Dense Layer with 128 neurons

- Finally, Dense Layer with Activation Function Soft max to assigns decimal probabilities to each class in our multi-class problem

Parameters:

- Total Parameters: 1,321,380 parameters
- Trainable parameters: 1,316,124 parameters
- Non trainable parameters: 5,256 parameters

4.5.4 CNN 1D

- A CNN is composed of layers that filters (convolve) the inputs to get useful information. These have two kinds of layers: convolution layers and pooling layers. The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image or signal with a filter

- a simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. We use three main types of layers to build CNN architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full CNN architecture.

- CNN Model much faster than Inception model and VGG16 Model

-This model is clearly based on our own experiences in this problem

Model Summary

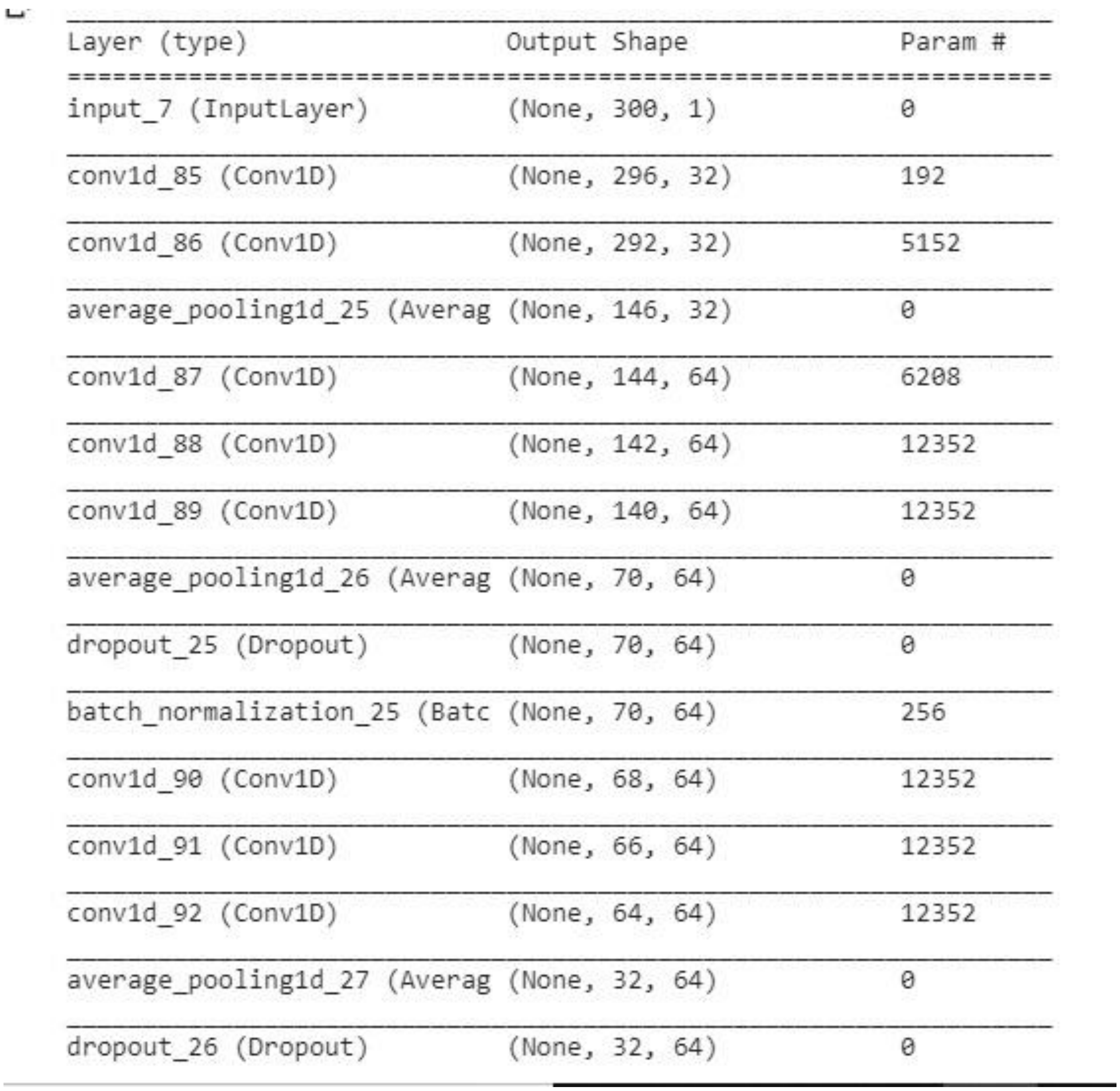
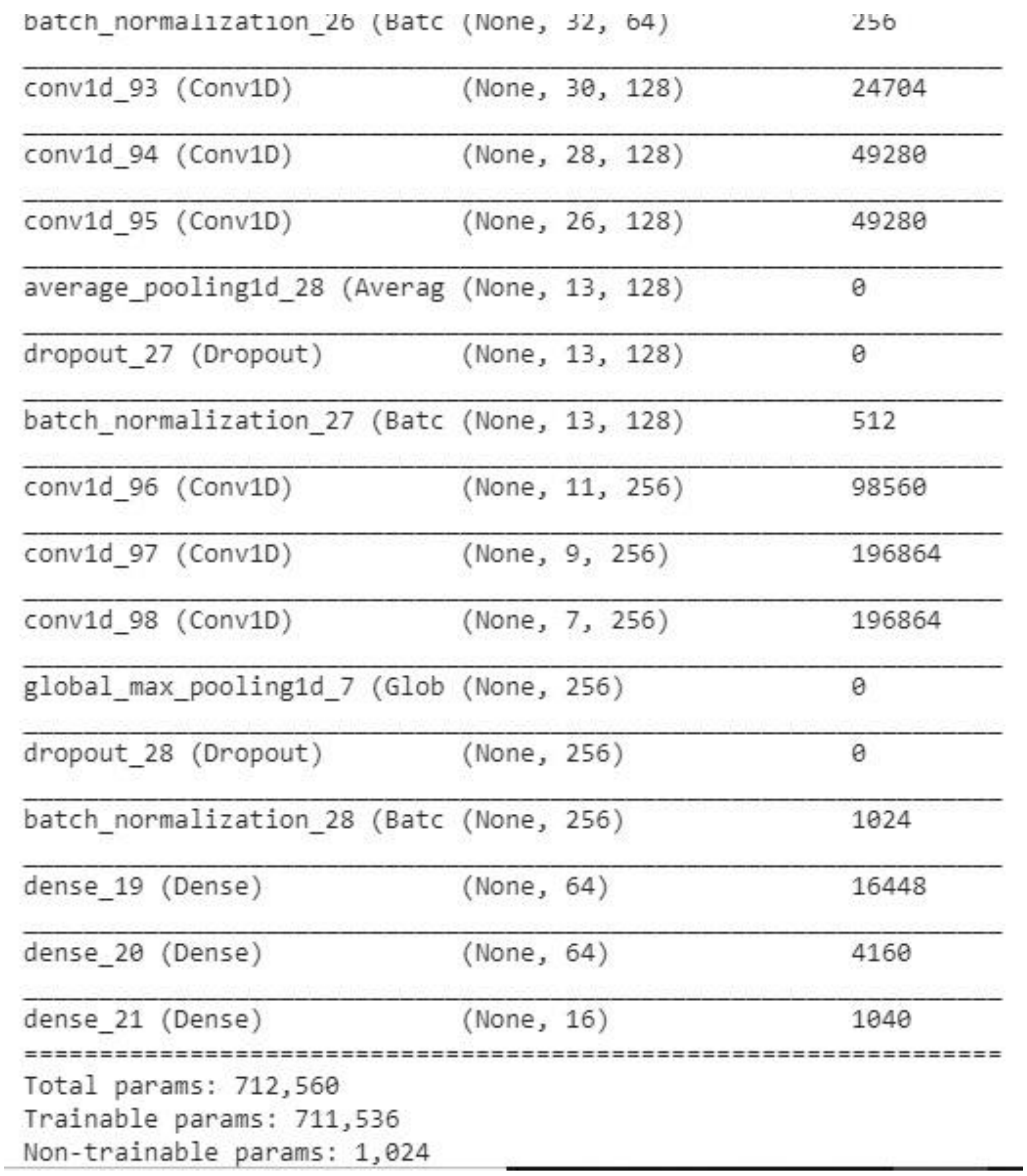

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 300, 1)            0
_________________________________________________________________
conv1d_85 (Conv1D)           (None, 296, 32)           192
_________________________________________________________________
conv1d_86 (Conv1D)           (None, 292, 32)           5152
_________________________________________________________________
average_pooling1d_25 (Averag (None, 146, 32)           0
_________________________________________________________________
conv1d_87 (Conv1D)           (None, 144, 64)           6208
_________________________________________________________________
conv1d_88 (Conv1D)           (None, 142, 64)           12352
_________________________________________________________________
conv1d_89 (Conv1D)           (None, 140, 64)           12352
_________________________________________________________________
average_pooling1d_26 (Averag (None, 70, 64)            0
_________________________________________________________________
dropout_25 (Dropout)         (None, 70, 64)            0
_________________________________________________________________
batch_normalization_25 (Batc (None, 70, 64)            256
_________________________________________________________________
conv1d_90 (Conv1D)           (None, 68, 64)            12352
_________________________________________________________________
conv1d_91 (Conv1D)           (None, 66, 64)            12352
_________________________________________________________________
conv1d_92 (Conv1D)           (None, 64, 64)            12352
_________________________________________________________________
average_pooling1d_27 (Averag (None, 32, 64)            0
_________________________________________________________________
dropout_26 (Dropout)         (None, 32, 64)            0
```

Figure 4.9 CNN Summary (part 1)



```
batch_normalization_26 (Batc (None, 32, 64)            256
_________________________________________________________________
conv1d_93 (Conv1D)           (None, 30, 128)           24704
_________________________________________________________________
conv1d_94 (Conv1D)           (None, 28, 128)           49280
_________________________________________________________________
conv1d_95 (Conv1D)           (None, 26, 128)           49280
_________________________________________________________________
average_pooling1d_28 (Averag (None, 13, 128)           0
_________________________________________________________________
dropout_27 (Dropout)         (None, 13, 128)           0
_________________________________________________________________
batch_normalization_27 (Batc (None, 13, 128)           512
_________________________________________________________________
conv1d_96 (Conv1D)           (None, 11, 256)           98560
_________________________________________________________________
conv1d_97 (Conv1D)           (None, 9, 256)            196864
_________________________________________________________________
conv1d_98 (Conv1D)           (None, 7, 256)            196864
_________________________________________________________________
global_max_pooling1d_7 (Glob (None, 256)               0
_________________________________________________________________
dropout_28 (Dropout)         (None, 256)               0
_________________________________________________________________
batch_normalization_28 (Batc (None, 256)               1024
_________________________________________________________________
dense_19 (Dense)             (None, 64)                16448
_________________________________________________________________
dense_20 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_21 (Dense)             (None, 16)                1040
=================================================================
Total params: 712,560
Trainable params: 711,536
Non-trainable params: 1,024
```

Figure 4.10 CNN Summary (part 2)

- Before last Dense layers use **Global Average Pooling** (GAP) layer to minimize over-fitting and reducing the total number of parameters in the model.

- Batch Normalization between CNN layers to solve gradient vanishing and exploding problem and reduces the amount by what the hidden unit values shift around (covariance shift) and to speed up learning

- Then traditional two Dense Layer with 64 neurons for each one

- Finally, Dense Layer with Activation Function Soft max to assigns decimal probabilities to each class in our multi-class problem

Parameters:

- Total Parameter: 712,560 parameters
- Trainable parameters: 711,536 parameters
- Non trainable parameters: 1,024 parameters

Chapter Five:

System Testing

5.1 Evaluation

We evaluate our models by calculating confusion matrix that clarify right and wrong classifications in each class.
Then we use confusion matrix to calculate overall accuracy and calculate accuracy for each class then calculate avg accuracy of all classes.

Overall Accuracy: number of correctly predicted beats/total of beats to predict.

Average Accuracy: it is the average of each accuracy per class (sum of accuracy for each class predicted/number of class)

In unbalanced datasets, **overall accuracy** may be not accurate to test a model.

Average accuracy is a good way to test a model to know if it has learnt all classes almost the same.

i. Build Confusion Matrix

- Given actual classes and predictions output from model
- Define 2D array of zeros of size classes×classes
- Iterate on actual and predictions
- Row index = actual[i]
- Column index = predictions[i]
- Increment confusion_matrix[Row index, Column index]

ii. Calculate Accuracy

- Given Confusion Matrix
- Overall Accuracy = sum(diagonal)/sum (confusion matrix)) *100
- Accuracy for each class = (confusion matrix [i,i]/sum(row i))*100
- Average Accuracy = avg (classes accuracies).

iii. Predict and Evaluate

- Given trained model, x test and y test
- Predict x test using trained model.
- pass predictions and actual

to method that calls Build Confusion Matrix and Calculate Accuracy

iv. Two Stages Predict

- Given first stage model and second stage models.
- Predict each x in x test using the first stage model
- Based on result of prediction, the model of predicted category predicts x again.
- Output prediction from category model is the final prediction.
- Collect all final predictions and it is the output of predicting using two stages model.
- Then Evaluate the model using the same Evaluation methods.

5.2 Results

5.2.1 Results of different models

LSTM:

LSTM model has good overall accuracy, but it is biased to some classes which leads to unacceptable average accuracy.

Overall Accuracy over 5 trails

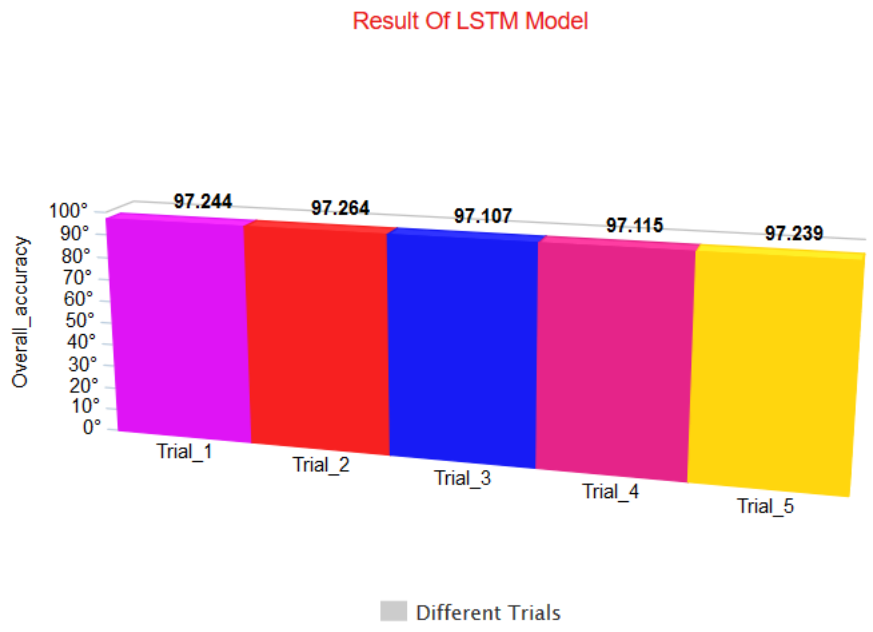


Figure 5.1 Overall accuracy in 5 trails based on LSTM

- **Average Accuracy over 5 trails**

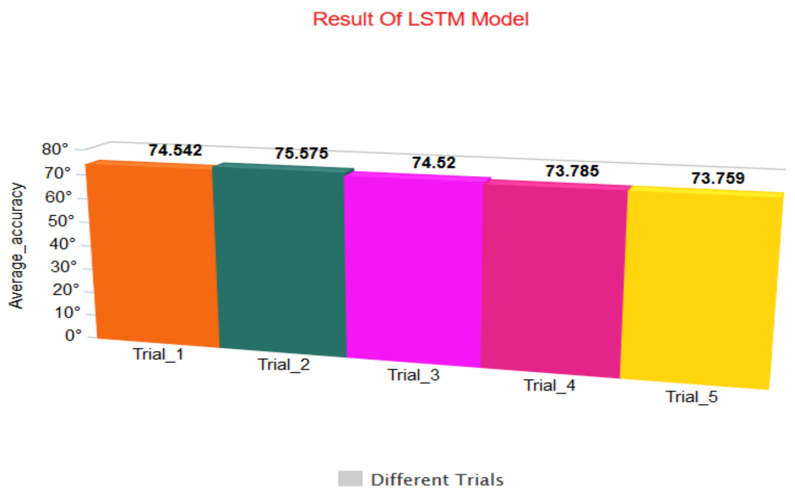


Figure 5.2 Average accuracy in 5 trails based on LSTM

VGG16:

VGG16 model has good overall accuracy but moderate average accuracy.

- **Overall Accuracy over 5 trails**

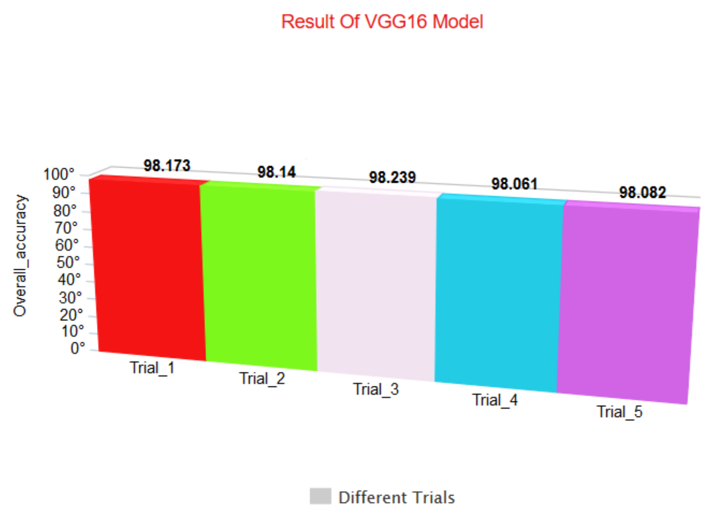


Figure 5.3 Overall accuracy in 5 trails based on VGG16

- **Average Accuracy over 5 trails**

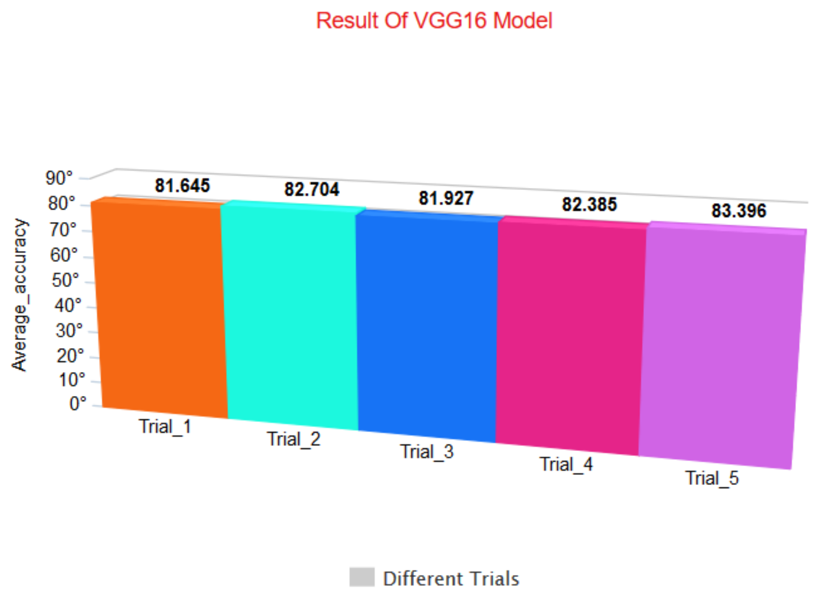


Figure 5.4 Average accuracy in 5 trails based on VGG16

Inception:

Inception model has high overall accuracy and good average accuracy.

- **Overall Accuracy over 5 trails**

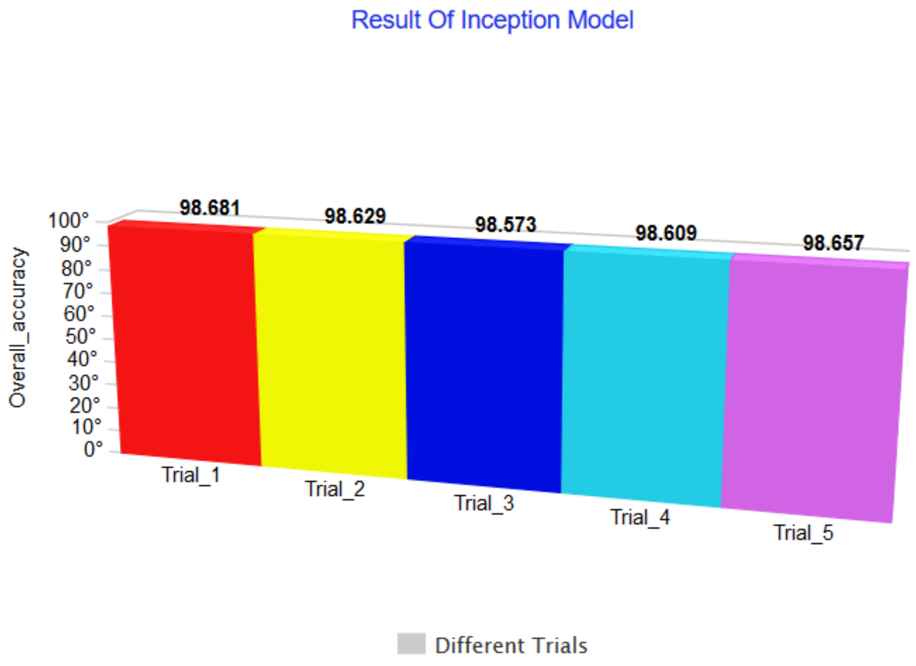


Figure 5.5 Overall accuracy in 5 trails based on inception

- **Average Accuracy over 5 trails**

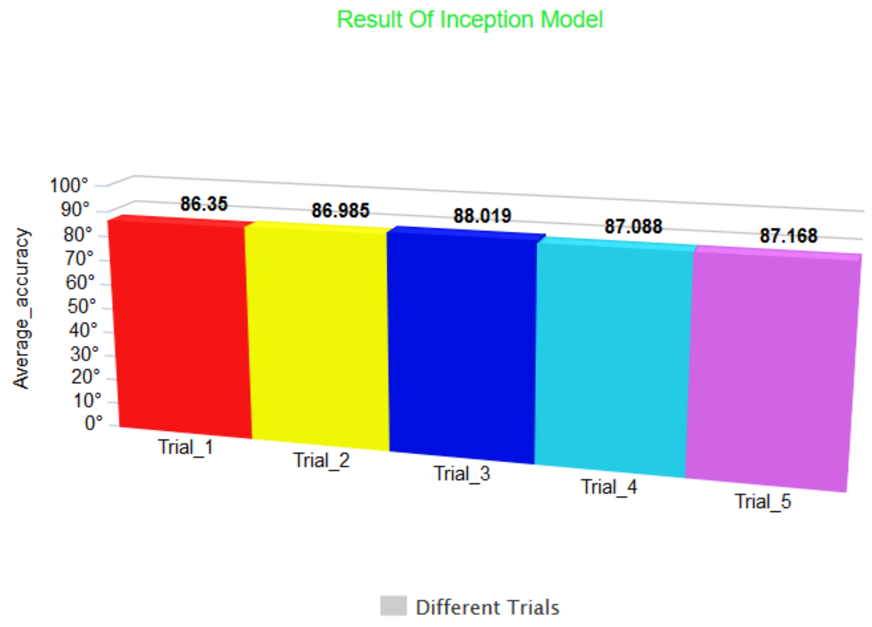


Figure 5.6 Average accuracy in 5 trails based on inception

CNN:

CNN model has high overall accuracy and good average accuracy.

- **Overall Accuracy over 5 trails**

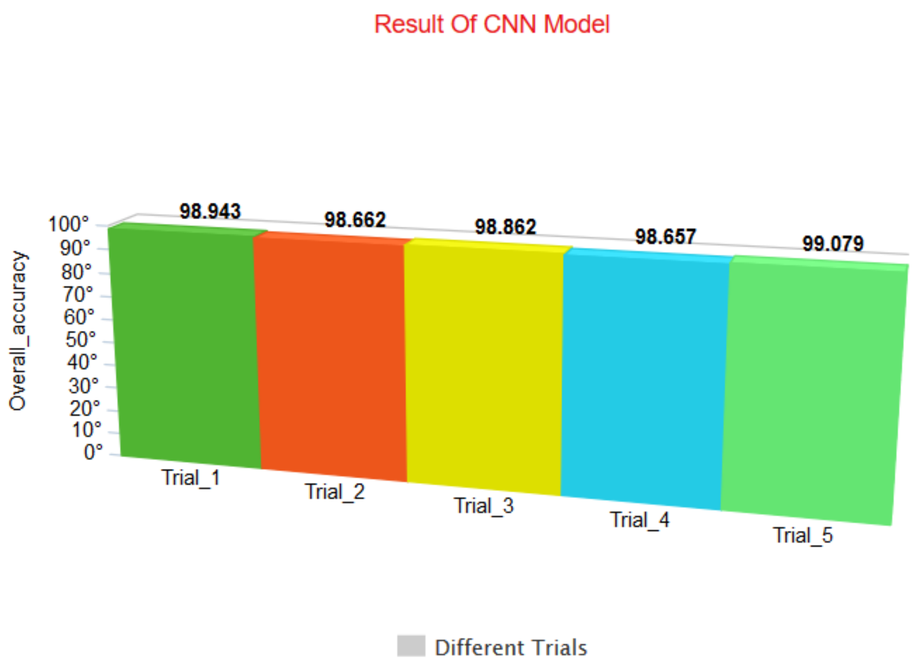


Figure 5.7 Overall accuracy in 5 trails based on CNN

- **Average Accuracy over 5 trails**

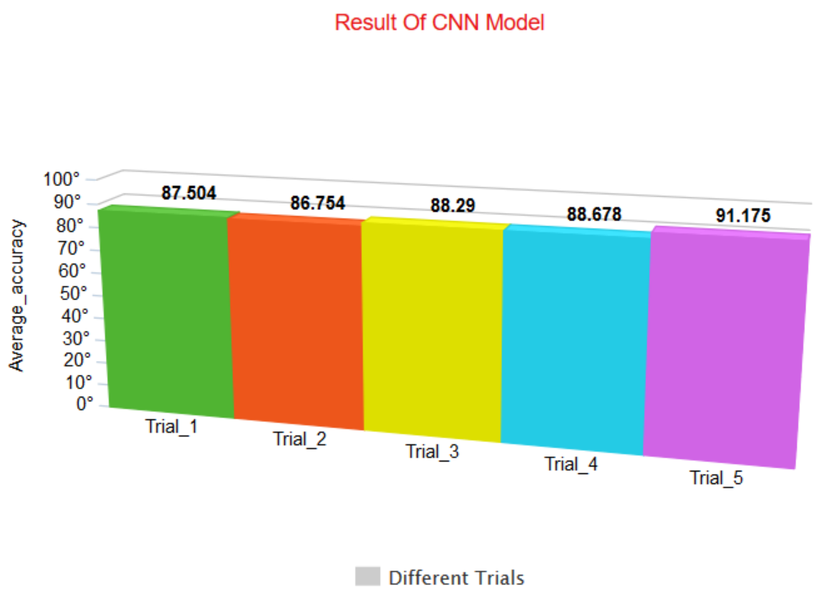


Figure 5.8 Average accuracy in 5 trails based on CNN

5.2.2 Comparison between models

Model	**LSTM**	**VGG16**	**CNN**	**Inception**
# Params	1,214,896	28,049,744	712,560	1,321,380
Overall Accuracy	97.19 %	98.14 %	98.84 %	98.63 %
Average Accuracy	74.44 %	82.41 %	88.48 %	87.31 %

Table 5.1 Comparison between all used models

Average of accuracies across all trials.

In one stage CNN and Inception are similar.

5.2.3 Comparison between One Stage and Two Stages model

Average of accuracies across all trials for CNN.

	One Stage CNN	**Two Stages CNN**
Overall Accuracy	98.84 %	**98.61 %**
Average Accuracy	88.48 %	**90.12 %**

Table 5.2 Comparison between One Stage and Two Stages in CNN

Using Multistage technique have improved average accuracy in CNN.

Average of accuracies across all trials for Inception.

	One Stage Inception	Two Stages Inception
Overall Accuracy	98.63 %	**98.29 %**
Average Accuracy	87.31 %	**87.39 %**

Table 5.3 Comparison between One Stage and Two Stages in Inception

In Inception it is almost the same.

5.2.4 Result of Data augmentation

We have applied data augmentation on training data as we explained before and trained a CNN model on new augmented data

- **Overall Accuracy over 5 trails**

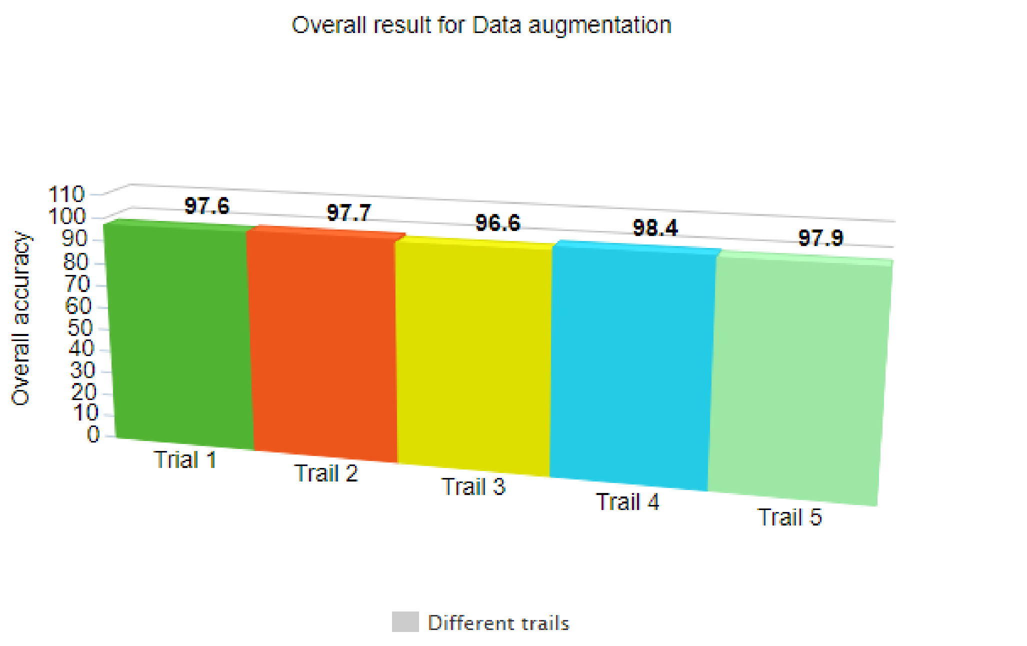


Figure 5.9 Overall accuracy for data augmentation

5.2.5 Result for subject oriented

This type of classification use data has been split as we mention in previous chapter, using inception model an 81.16% overall accuracy and 50.1% average accuracy have been achieved.

5.2.6 Best Result

After trying different models and different architectures we have found that CNN is the best model fitting the data.
In trail 5:
One Stage CNN has 91.17 % as average accuracy
And 99.08 % as overall accuracy.

Two stages CNN has 91.98 % as average accuracy
And 98.74 % as overall accuracy.

5.3 Interface

Computer Aided Diagnosis for Heart diseases Application

1-The lunch Form Contains (Test, Our Results) Buttons

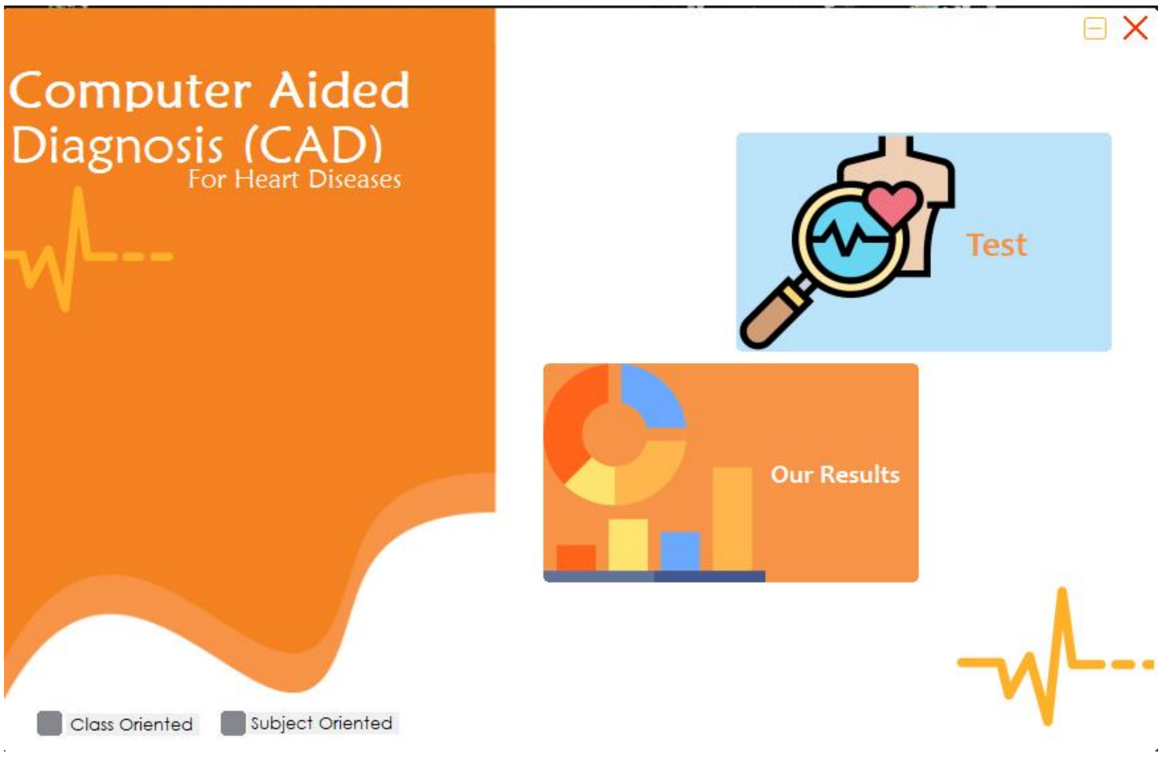


Figure 5.10 Starting lunch project

When it’s lunch we have Two scenario

First scenario If **Test Button** has been chosen, he should Choose from Check boxes between Two types of classification (Subject or Class) Oriented.

Figures from 5. [10-15] Show test Step:
1-When Clicking on Load Test Signal Button [5.10]
2- Open File Dialog to Choose Test Signal 5. [11,12]

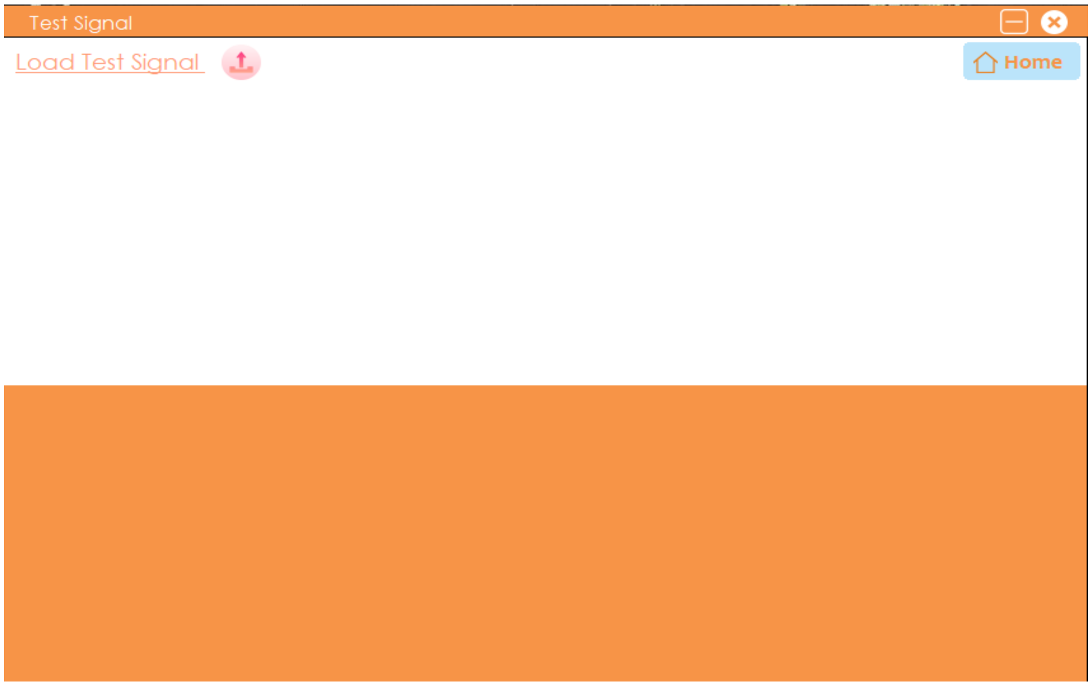
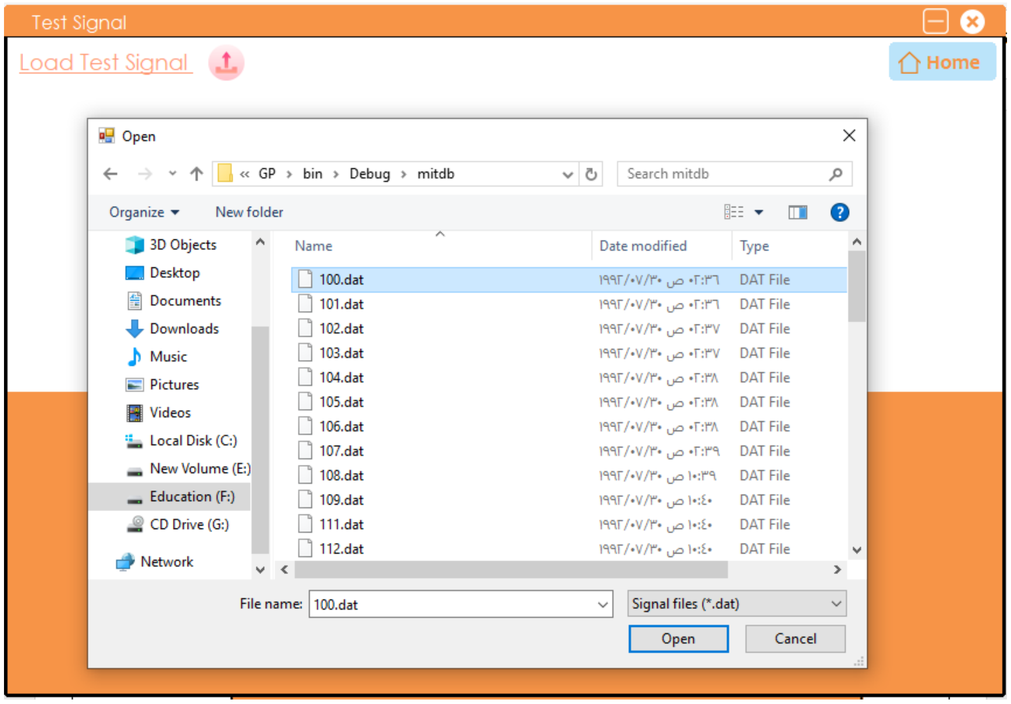


Figure5.10. Choose signal for test



Name	Date modified	Type
100.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٦	DAT File
101.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٦	DAT File
102.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٧	DAT File
103.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٧	DAT File
104.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٨	DAT File
105.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٨	DAT File
106.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٨	DAT File
107.dat	١٩٩٢/٠٧/٣٠ ص ٠٢:٣٩	DAT File
108.dat	١٩٩٢/٠٧/٣٠ ص ١٠:٣٩	DAT File
109.dat	١٩٩٢/٠٧/٣٠ ص ١٠:٤٠	DAT File
111.dat	١٩٩٢/٠٧/٣٠ ص ١٠:٤٠	DAT File
112.dat	١٩٩٢/٠٧/٣٠ ص ١٠:٤٠	DAT File

File name: 100.dat

Signal files (*.dat)

Figure 5.11 Select test signal

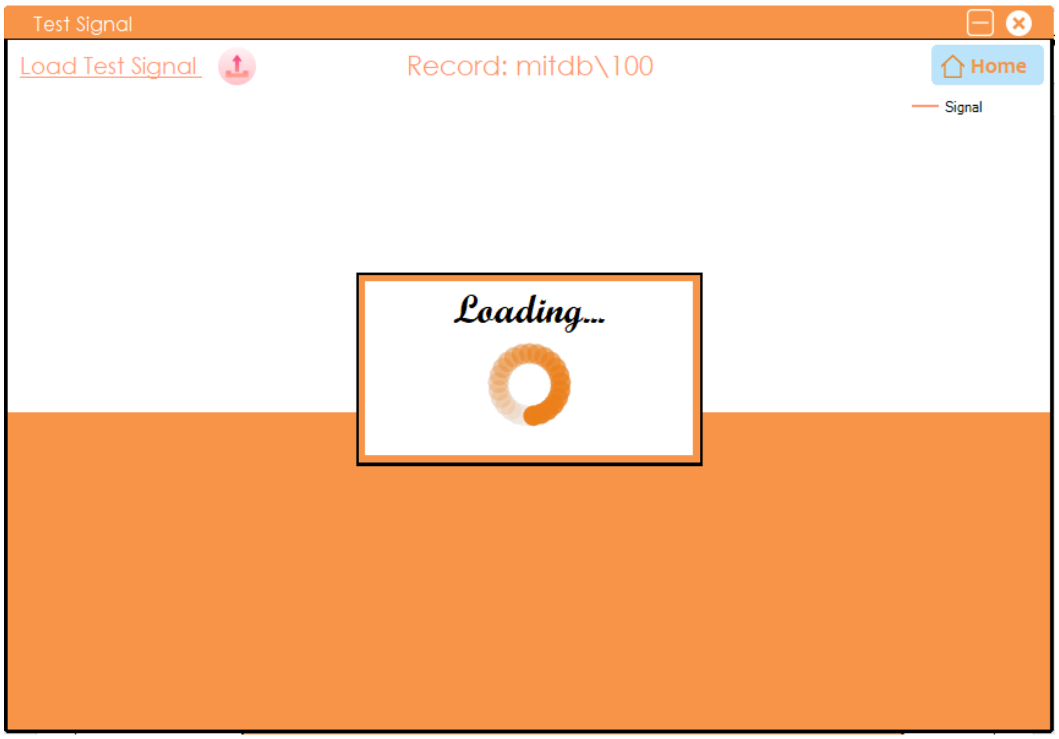


Figure 5.13. Wait signal to load

3- Start Drawing Chosen Signal as Figure [5.13].

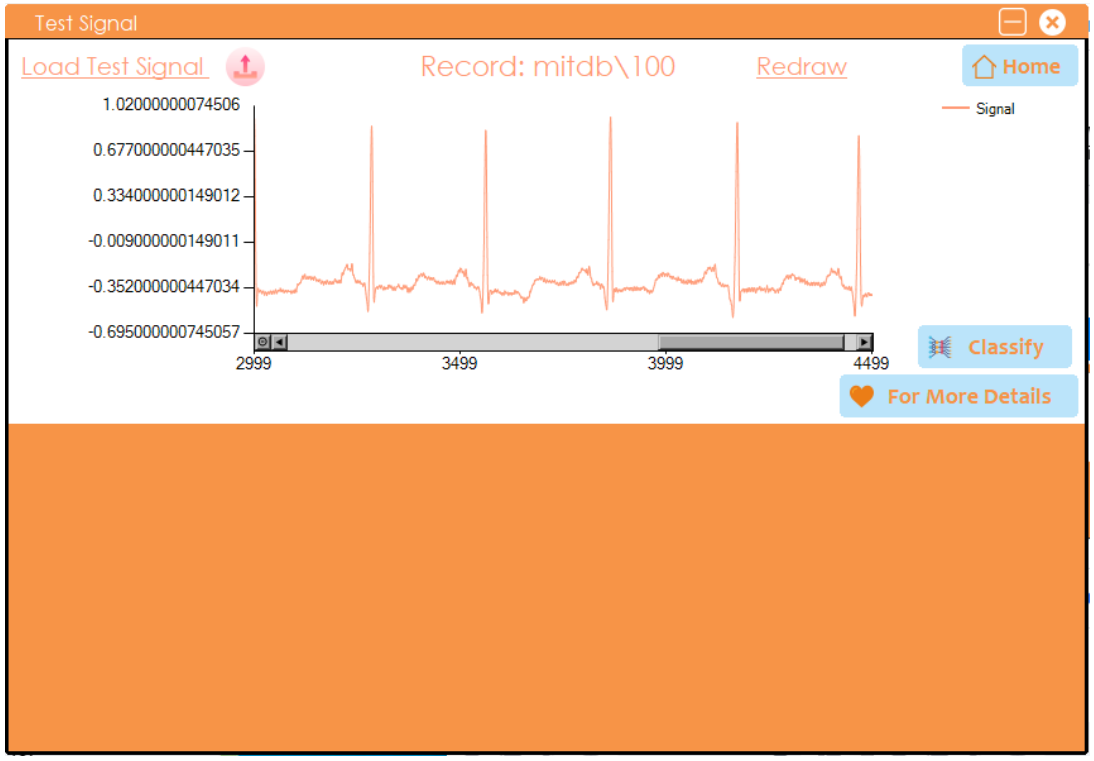
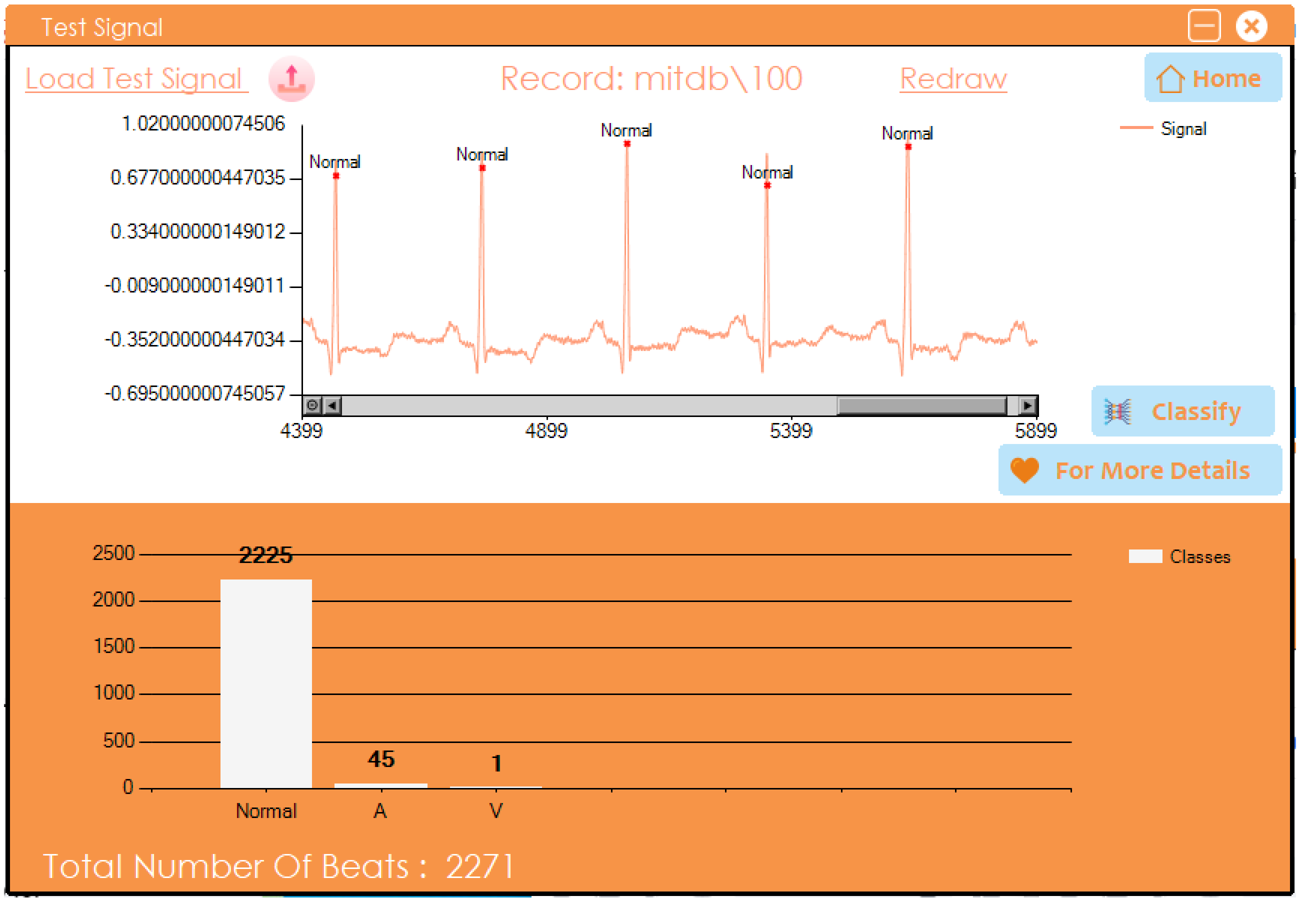


Figure 5.14 Draw chosen signal

There are **two probabilistic** to deal with selected signal

First is click on Classify Button To show total numbers of beat for each class and type of each beat



Total Number Of Beats : 2271

Figure5. 15.Classify Result

Second is click on For More Details Button, it shows our project steps to classify each beat in signal
(Filtration and Normalize Signal, Detect R Peaks of Signal, Segmentation, Classification) As Shown in Figure 5. [14-15].

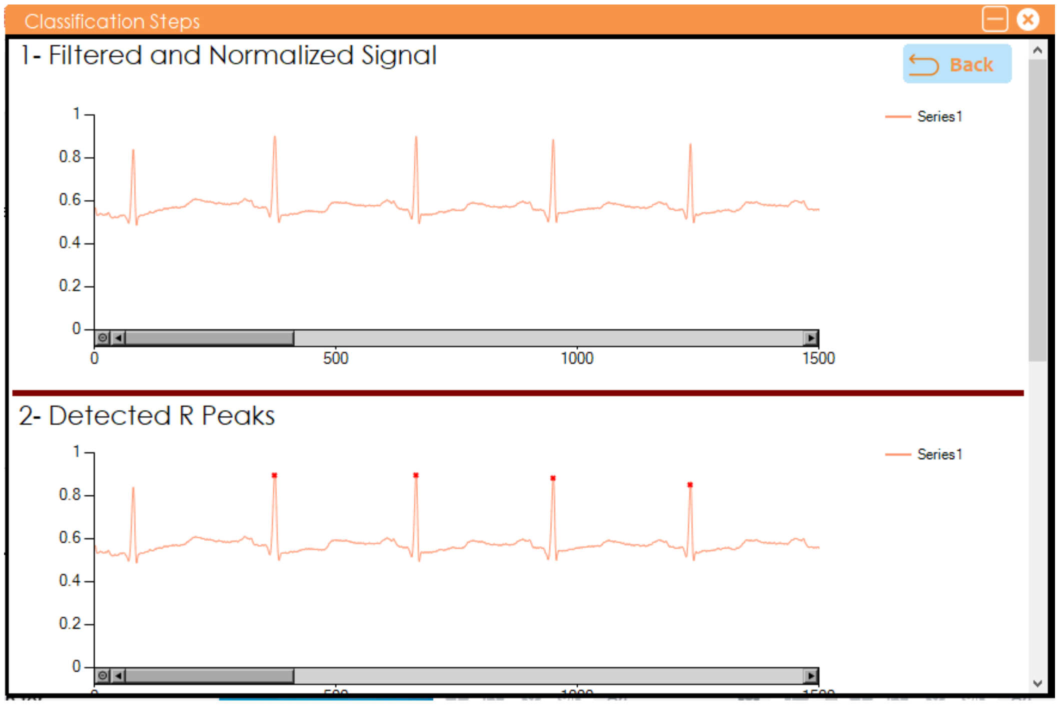


Figure 5.16 filtration and detection steps

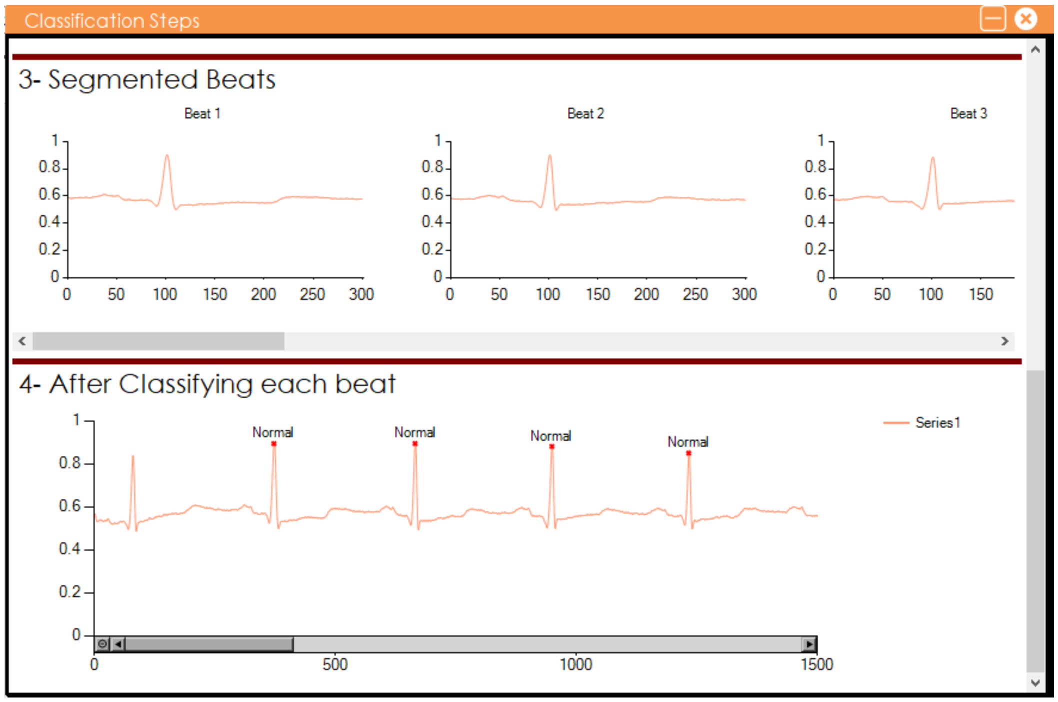


Figure 5.17 segmentation and classification steps

Second scenario If **Our Results Button** has been chosen, Result of all what we did, will appear based on what is selected from combo boxes

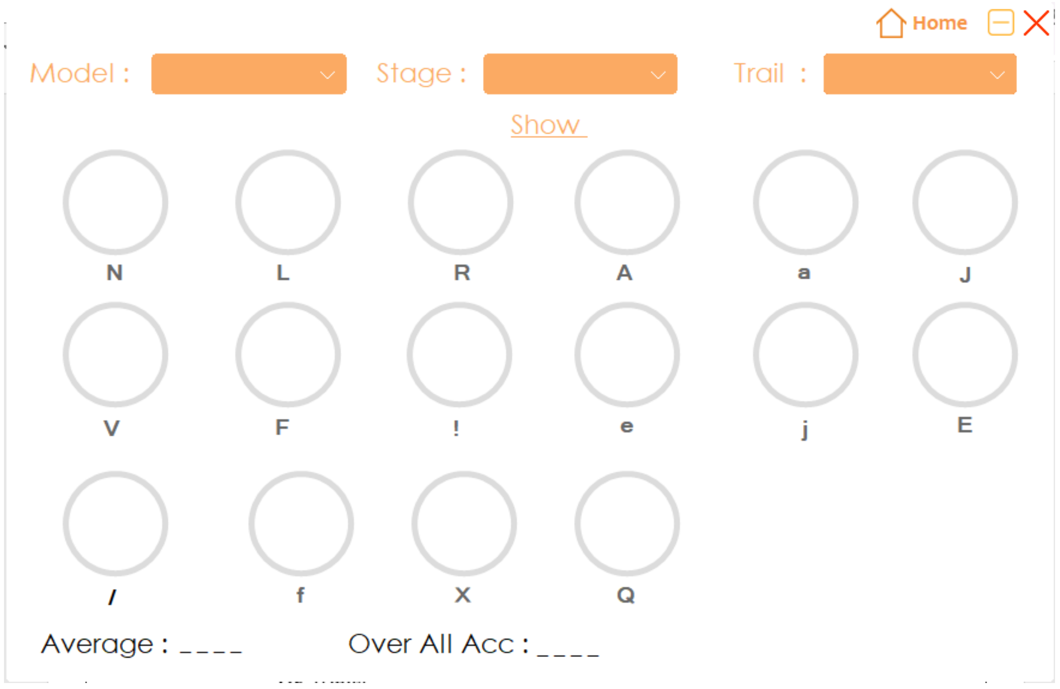
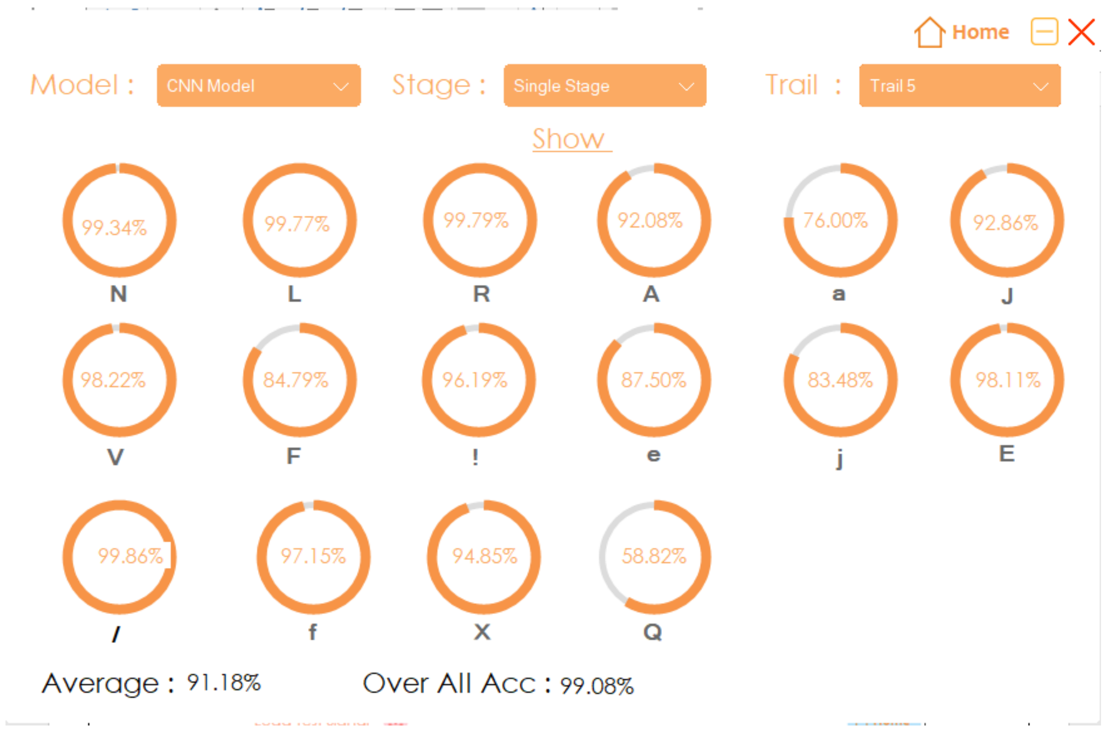


Figure 5.18. Main result form

In figure [5.17,5.18] show examples for second scenario

Example 1 when user choose **CNN model** and **One Stage** to see result of this model in trail 5



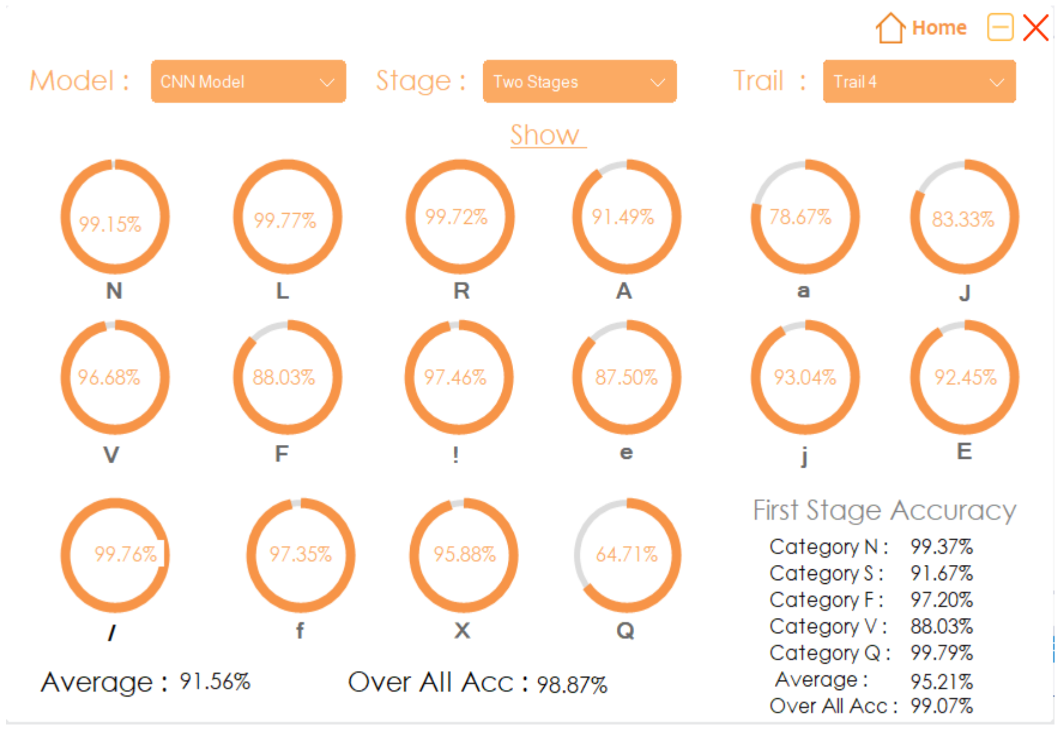
Model : CNN Model Stage : Single Stage Trail : Trail 5

99.34% N
99.77% L
99.79% R
92.08% A
76.00% a
92.86% J
98.22% V
84.79% F
96.19% !
87.50% e
83.48% j
98.11% E
99.86% /
97.15% f
94.85% X
58.82% Q

Average : 91.18% Over All Acc : 99.08%

Figure 5.19 first stage on trial 5 based on CNN model

Example 2 when user choose **CNN model** and **Two Stage** to see result of this model in **trail 4**



Model : CNN Model Stage : Two Stages Trail : Trail 4

99.15% N
99.77% L
99.72% R
91.49% A
78.67% a
83.33% J
96.68% V
88.03% F
97.46% !
87.50% e
93.04% j
92.45% E
99.76% /
97.35% f
95.88% X
64.71% Q

Average : 91.56% Over All Acc : 98.87%

First Stage Accuracy

Category N :	99.37%
Category S :	91.67%
Category F :	97.20%
Category V :	88.03%
Category Q :	99.79%
Average :	95.21%
Over All Acc :	99.07%

Figure 5.20 second stage on trial 4 based on CNN model

Chapter Six:

Conclusion and future work

6.1 Conclusion

To sum up, an automatic hybrid hierarchy method of two stages has been developed. The first stage classifies a heartbeat into one of five main categories (N, Q, V, S, F) Thereafter, the heartbeat moves to the second stage to know which class in this category it belongs.
This progress was based on two main steps. The first one is preprocessing that filtering, normalizing and segment signals based on R-peak.
The second one is a classification models that is based on deep learning techniques, we have two architectures for classification, one stage and two stages. And we implemented some models and they are LSTM, VGG16, Inception and CNN Model
dataset (MIT-BIH) which contains 16 classes mapped to five main categories and 48 records
Finally, our method has achieved in One Stage CNN has 91.17% as average accuracy and 99.08% as overall accuracy.
And in Two stages CNN has first stage the best average accuracy of "95.16". and the best result of" 99.01" overall accuracy Regarding the second stage "91.98 %" average accuracy and "98.74 %" overall accuracy has been achieved

6.2 Future Work

We will make our project work on real time to be **Real Time System,**
by taking signals live from ECG device and out results without need to save or load any data.

Tools

- Python
- C#
- Google Colab
- Visual studio
- PyCharm
- Python packages
  - NumPy
  - Pandas
  - Sklearn
  - TensorFlow
  - Keras framework
  - Wfdp
  - SciPy
  - matplotlib

References

- 1- apple watch app
  https://support.apple.com/en-us/HT208955

- 2-**1N P JOSHI,2P S TOPANNAVAR, SUPPORT VECTOR MACHINE BASED HEARTBEAT CLASSIFICATION, International Journal of Advances in Science Engineering and Technology, ISSN: 2321-9009, Volume- 2, Issue-3, July-2014**

- **3- Manab Kumar Das and Samit Ari, ECG Beats Classification Using Mixture of Features, Hindawi Publishing Corporation ,International Scholarly Research Notices ,Volume (2014),Article ID 178436**

- **4- Roshan Joy Martis , U. Rajendra Acharya , K.M. Mandana c, A.K. Ray , Chandan Chakraborty , Application of principal component analysis to ECG signals for automated diagnosis of cardiac health , Expert Systems with Applications Volume 39,issue 14, 2012, Pages 11792-11800**

- **5-Hassan Yazdanian , Ashkan Nomani and Mohammad Reza Yazdchi , Autonomous Detection of Heartbeats and Categorizing them by using Support Vector Machines, Proceedings of 20th Iranian Conference on Biomedical Engineering (ICBME 2013), University of Tehran, Tehran, Iran, December (2013).**

- **6- S. Sahoo, B. Kanungo, S. Behera, S. Sabut, Multiresolution wavelet transform based feature extraction and ECG classification to detect cardiac abnormalities, Measurement (2017)**

- **7- Hadeer El-Saadawy, Manal Tantawi, Howida A. Shedeed, Mohamed F. Tolba, Hybrid hierarchical method for electrocardiogram heartbeat classification , The Institution of Engineering and Technology ,Signal Processing ,(March 2018)**

- **8- Fatin A. Elhaj , Naomie Salima, Arief R. Harris,Tan Tian Swee , Taqwa Ahmeda, Arrhythmia recognition and classification using combined linear and nonlinear features of ECG signals, Computer Methods and Programs in Biomedicine , Volume 127, April 2016, Pages 52-63**

- **9- Jose Antonio Gutiérrez- Gnecchi, Rodrigo Morfin -Magana , Daniel Lorias -Espinoza , Adriana del Carmen Tellez- Anguianoa , Enrique Reyes- Archundia , Arturo Méndez-Patino , Rodrigo Castaneda-Miranda , DSP-based arrhythmia classification using wavelet transform and probabilistic neural network , Biomedical Signal Processing and Control 32 (2017) .**

- **10-Dan Li, Jianxin Zhang*, Qiang Zhang*, Xiaopeng Wei, Classification of ECG Signals Based on 1D Convolution Neural Network, Dalian University, Dalian, P. R. China,IEEE 19th International Conference on e-Health Networking, Applications and Services (2017)**

- **11- Ö. Yıldırım, Paweł. Pławiak, R.-S. Tan, U.R. Acharya, Arrhythmia detection using deep convolutional neural network with long duration ECG signals, Computers in Biology and Medicine (2018).**

- **12- U.R. Acharya, S.L. Oh, Y. Hagiwara, J.H. Tan, M. Adam, A. Gertych, T.R. San, A deep convolutional neural network model to classify heartbeats, Computers in Biology and Medicine**

- **13- Philip de Chazal, Member, IEEE, Maria O'Dwyer, and Richard B. Reilly Senior Member, Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features, (2004)**

- **14- https://colah.github.io/posts/2015-08-Understanding-LSTMs/**

- **15 - https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-miss-algorithm-in-python/#:~:text=SMOTE%20(synthetic%20minority%20oversampling%20technique)%20is%20one%20of%20the%20most,instances%20between%20existing%20minority%20instances.**

- **16 - https://neurohive.io/en/popular-networks/vgg16/**

- **17-https://deepai.org/machine-learning-glossary-and-terms/inception-module**

- **18 - https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c**