Detecting Heart Anomalies using Heartbeat Sound

A Project / Dissertation as a Course requirement for

MSc Data Science and Computing

Majeti VSSS Durgesh

19230

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

(Deemed to be University)

Department of Mathematics and Computer Science

Muddenahalli Campus

April, 2021

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING
(Deemed to be University)

Dept. of **Mathematics & Computer Science**
Muddenahalli Campus

CERTIFICATE

This is to certify that this Project / Dissertation titled **Detecting Heart Anomalies using Heartbeat Sound** submitted by **Majeti VSSS Durgesh**, 19230, Department of Mathematics and Computer Science, Muddenahalli Campus is a bonafide record of the original work done under my supervision as a Course requirement for the Degree of MSc Data Science and Computing.

………………………………………..
Dr. Sampath Lonka
Project / Dissertation Supervisor

Countersigned by

Place: ………………………………...

Date: Head of the Department

DECLARATION

The Project / Dissertation titled **Detecting Heart Anomalies using Heartbeat Sound** was carried out by me under the supervision of Dr. Sampath Lonka, Department of Mathematics and Computer Science, Muddenahalli Campus as a course requirement for the Degree of MSc Data Science and Computing and has not formed the basis for the award of any degree, diploma or any other such title by this or any other University.

M. Durgesh

…………………………..

Place : Visakhapatnam

Date : 19th April, 2021

Majeti VSSS Durgesh
19230
II MDSC
Muddenahalli Campus

Acknowledgements

I would like to dedicate this page to all the people who dedicated their time to assist me in this project. Many people played a significant role in this project. I cannot rank their efforts but I would like to explicitly mention a few people who played vital roles throughout the lifetime of the project.

I thank our project guide **Dr. Sampath Lonka** whose assistance we always received whenever we asked for it.

I thank our campus **Deputy Director** and our **Associate Head of the Department** for planning the project in such a way that we get enough time and resources for completing this project.

I thank my partner, Sri **Ashwin Prakash** (19227), who worked along with me and shared all the troubles and brainstorming sessions all through the year for completing the project.

I thank all my **classmates** who are always ready to help me in my project.

I thank my **parents** for it is their consistent motivation that charged me whenever I was struck somewhere.

I would be failing in my task if I do not thank all the people across the globe, whose assistance I received in the form of articles that they explained and the techniques they shared in social media and learning platforms.

I am strongly convinced that it is the unseen hand of **Bhagawan Sri Sathya Sai Baba** that conducts the puppet show of this world. It is indeed the embodiment of love within everyone that actually helped me in completing the project.

Contents

1. INTRODUCTION

Cardiovascular diseases are one of the major causes of human death across the globe. The number of deaths due to CVD (Cardiovascular Diseases) is constantly rising and is expected to rise at a faster pace. We are in a constant need of advanced techniques to detect and treat such CVDs. This could revolutionize the way of detecting and identifying heart diseases and can impact many lives as well as the health care sector. The aim of this project is to develop a reliable and efficient method where in the first level of screening can be performed at hospitals by doctors using digital stethoscopes and also by patients at their home using a mobile device and an app to record the heartbeat sounds.

This project illustrates different techniques used to process audio data and model machine learning and deep learning models. The data that is collected is not void of environment noises. The dicrimination between various classes of audio is not trivial and it calls for advanced techniques in machine learning and deep learning areas to deal with the discriminators.

Heart is a very important organ that pumps blood to every part of the body. Good heart is very important for the overall health of the human body. Also, deaths due to CVD is the most common reason as stated by the World Health Organization and Center for Disease Control and Prevention. In Spite of its significance in the medical sector, this is a relatively unexplored area in the field of machine learning and deep learning.

Ashwin Prakash (19227) and myself worked on various machine learning and deep learning models to classify heart diseases and to accurately predict the disease that a person with a specific heartbeat pattern is facing.

The code for the project can be found at my GitHub page [1] .

2. Problem Statement

This problem statement was initially launched as a competition [2] in the year 2012. The problem statement involves classification of audio sample data, where distinguishing between classes of interest is non-trivial. Data is collected from two different types of devices. They vary in the accuracy of their recording and may contain unwanted noises. The differences between heart sounds corresponding to different heart symptoms can also be extremely subtle and challenging to separate. Success in classifying this form of data requires extremely robust classifiers. Despite its medical importance, this field is explored only a little using the tools of deep learning and machine learning. With this project, we try to build a robust and reliable model that will be of help to the health care sector and of people in general. We will also integrate the model and deploy it in cloud so that it is available for all to use.

3. Literature Review

3.1 Technology

The main technology used here is Python Language. All the code is written in python language. This is because python provides many standard libraries that can be used across various stages in a data science project pipeline. All the code is run on an Acer laptop with intel i5 9th Gen processor equipped with 8GB RAM. A dedicated Nvidia GTX 1650 GPU graphics card is used for developing deep learning models. All the code is written using jupyter notebooks or in vs-code editors.

3.2 Libraries and Modules

The following are the various python libraries and modules that are used in the development of this project.

3.2.1 Librosa

Librosa [6] is a very famous python package used for analysing audio files. This library is used to extract low level audio features like zero crossings and spectral centroid as well as high level features like MFCCs. Version 0.8.0 is used in this project.

3.2.2 Scikit Learn

Scikit Learn [5] package is famous for the varieties of machine learning models that it provides. It contains many machine learning algorithms as well as other tools that are used across the machine learning pipeline. Version 0.23.2 of this library is used in the project.

3.2.3 JSON

The JSON module is used for reading and writing JSON files. Version 2.0.9 is used in this project.

3.2.4 Numpy

Python Numpy [7] package makes it easy to work with multi dimensional vectors and vector operations. Numpy arrays are almost 50 times faster than regular lists in python. So, numpy objects are used whenever possible. Version 1.18.5 is used in this project.

3.2.5 Pandas

Pandas is a python library used for working with datasets. It has well defined methods that are useful in exploring, analysing and cleaning data. Version 1.1.4 is used in this project.

3.2.6 OS

OS module provides methods to interact with the operating system. This module, here, is used for reading files across folders.

3.2.7 TensorFlow

TensorFlow is another famous python package used for building machine learning and deep learning pipelines. In this project TensorFlow is used to build deep learning architectures like Convolution Neural Networks, Recurrent Neural Networks and Auto Encoders. Version 2.3.2 is used in this project.

3.3 Data

This data is originally collected for a machine learning challenge for classifying heart beats. The data contains audio files of varying lengths collected from two different sources. The data contains five different types of heartbeat sounds. They are:

- Normal:
  In this category, there are normal, healthy heart sounds. Since, the data is collected across age groups, we can expect to have heart beat rate from 50 to 140 beats per minute.
- Murmur:
  Heart murmur may sound as though there is a whooshing noise between “lub” and “dub” of heart beat. They can be symptoms of some very serious heart diseases.
- Extra Heartbeat:
  Extra heart beat is an additional “lub” or “dub” in heart beat. They need not be particularly fatal but may prove to be very serious in some situations.Detecting this type of heart beat can significantly help the patient and doctor.
- Artifact:
  In this category, there could be a wide variety of sounds that are captured while recording the heartbeats. They could range from the echos to music and noise while recording the heartbeat sound. Identifying this correctly will tell the practitioner to collect the heartbeat once again.
- Extrasystole:
  These types of sounds may occur occasionally and the heart beat skips the rhythm. An extra or a skipped “lub” or “dub” could be possible. It need not be a sign of a disease as it is a common phenomenon in adults and children alike. But some extra systole could be the reason for a heart disease.

This dataset link is kept in the reference section [3]

3.4 Evaluation Metrics

The evaluation metrics are metrics that are used to quantify the accuracy and reliability of the model. The model built is a classification model. Accuracy score is a very commonly used metric for classification problems. We use other metrics as well.

Model hyper parameter tuning is performed based on accuracy score of the model. The tuned models are then evaluated on other metrics like Recall, Precision and F1 score.

3.4.1 Confusion matrix

Confusion matrix is a visualization matrix used in classification problems. The value in each element of the matrix corresponds to the number of predictions that fall into a specific category.

Each element in that matrix is represented with a pair of numbers (i,j) where i is the row number and j is the column number. Number i corresponds to the actual class value and number j corresponds to the predicted class value.

3.4.2 Accuracy

Accuracy is defined as the ratio of correctly predicted observation by total number of observations.

$$Accuracy = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

3.4.3 Precision

Precision score of a class ‘A’ is the ratio of correctly predicted observations to all the observations that are predicted as class ‘A’. From the confusion matrix, the precision score of ith class is the ratio of ith element of ith column with the sum of all the elements of ith column.

3.4.4 Recall

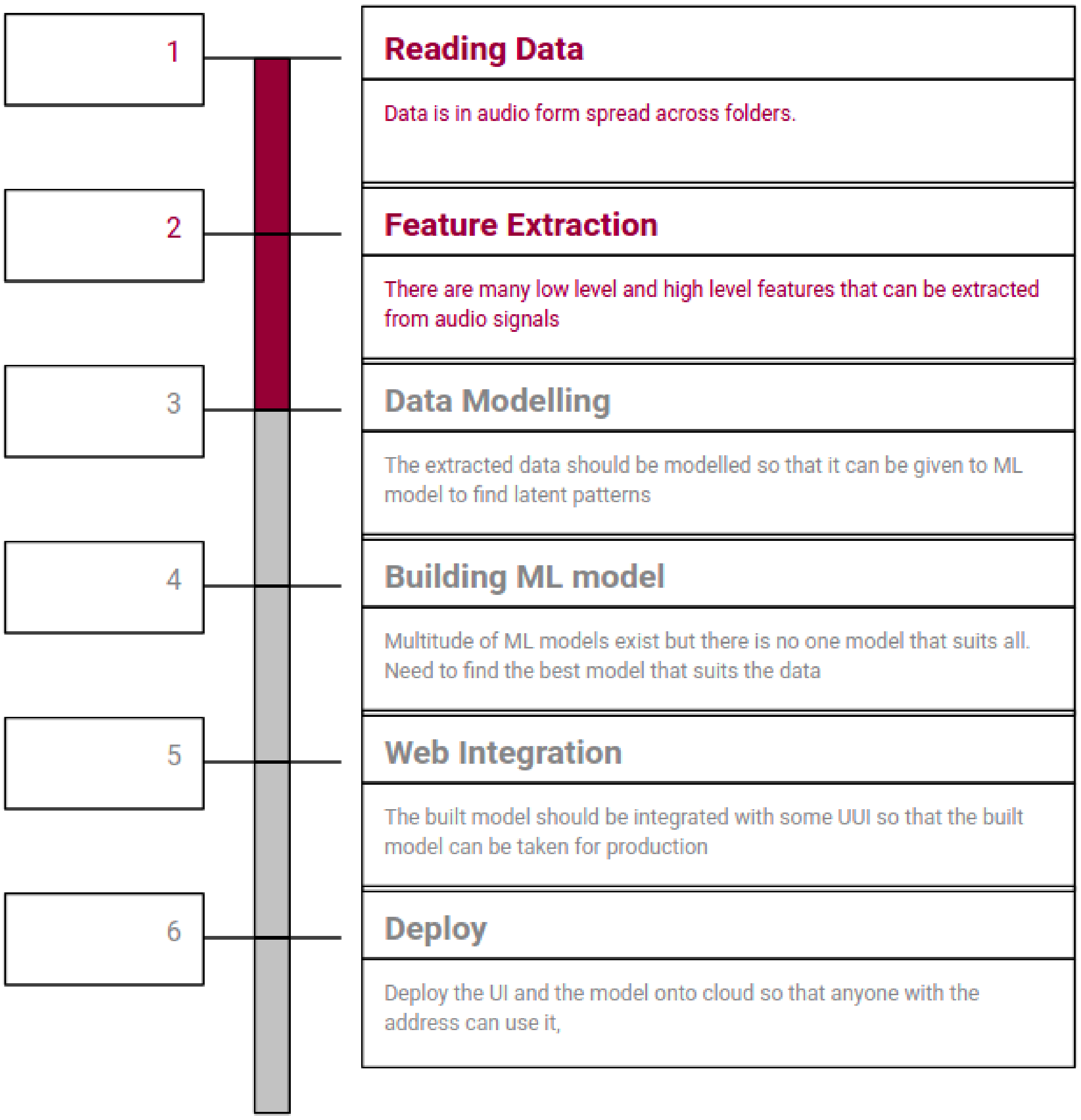
Recall score of a class ‘A’ is the ratio of correctly predicted observations to all the observations that belong to class ‘A’. From the confusion matrix, the recall score of ith class is the ratio of ith element of ith row with the sum of all the elements of ith row.

3.4.5 F1 Score

F1 score is the weighted average of Precision and Recall values. F1 score gives unbiased scores when the classes are not balanced.

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

4. Project structure and timeline


1
2
3
4
5
6
Reading Data
Data is in audio form spread across folders.
Feature Extraction
There are many low level and high level features that can be extracted from audio signals
Data Modelling
The extracted data should be modelled so that it can be given to ML model to find latent patterns
Building ML model
Multitude of ML models exist but there is no one model that suits all. Need to find the best model that suits the data
Web Integration
The built model should be integrated with some UUI so that the built model can be taken for production
Deploy
Deploy the UI and the model onto cloud so that anyone with the address can use it,

The first stage of the project starts with reading the audio files. The audio files are then processed to extract some important features. We also try to reduce the feature space using some advanced techniques like auto encoders. Once the features are extracted, we model the data so that it suits the machine learning and deep learning models that we use. In the fourth stage, we pass the data to some predefined machine algorithms. We also create our own Convolution Neural Network and Long Short Term Memory models. We then choose the best algorithm based on various metrics that we defined earlier. In the next stage, we integrate the model with the web tool using some web integration frameworks. In the final stage, we deploy it in the Heroku free tier so that the application is accessible from anywhere.

5. Reading data and Feature Extraction

This project deals with classifying five different types of heartbeat sounds. An audio file is identified to be one of the five types by the name of the audio file. For example, Normal heart beat audible file has the word “normal” in it. Similarly, murmur heartbeat has the word “murmur” in it’s file name. Different files are identified in this way.

We use os modules in python to access various audio files across different folders. We then use the librosa package to extract audio features from the audio clip. The features that are extracted are:

- MFCC:

  MFCC stands for Mel Frequency Cepstral Coefficient. It is a scaling feature that scales frequencies that humans can perceive more clearly. Humans can identify the change of pitch at lower frequencies than at higher frequencies. This MFCC uses Discrete Cosine Transformation and log transformation in frequency domain so that it scales down to what humans can clearly identify the differences of the sound. For this data set, we extracted 30 MFCC values for each window of audio data.
- Spectral Centroid:

  It is a value that is similar to the weighted average of different frequencies in frequency domain. It tells the center of energy of the audio signal.
- Zero Crossings:

  Zero crossings is the number of times the sound wave touched the zero line. It is the number of times the sound oscillated between positive and negative values.
- Spectral Rolloff:

  It gives a threshold frequency value. It is a frequency value under which most of the energy lies.
- Chroma STFT:

  Chroma STFT is another high level feature that is used mostly in identifying frequency scales. It uses Short Term Fourier Transform to transform audio from time domain to frequency domain. The frequency domain data is then processed to chroma bins.

We used two methods of processing the audio files.

- Unclipped audio file:

  Audio clips range from 1 second to several seconds. All the above mentioned features are extracted for the whole audio file without clipping it into specific duration. The features are then stored in a csv file.
- Clipped audio file:

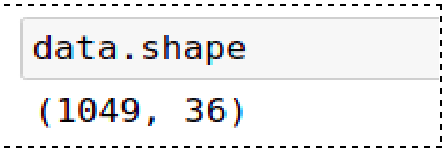
  Audio file is cut into several 3 seconds audio clips. The left over audio clip is removed. For example, an audio clip of length 14 seconds is clipped into four 3 second clips and the last 2 seconds is dropped. The above mentioned audio features are extracted for each 3 second audio clip. Each 3 second audio clip becomes a separate audio file and has a separate row in the csv file. This is done because deep learning models do not accept variable length input. Hence, audio has to be clipped to fixed length to use the deep learning models.

Two files are created to store the features of each type of processing (clipping the audio file and unclipped)

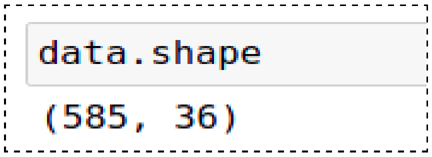
- Multiple features:
  All the mentioned audio features are extracted for each audio clip (clipped and unclipped). Mean values are taken on the temporal axis for features that have temporal axis. This is done to ensure that different duration audio clips are all having the same number of features. These features are stored in a csv file.
- MFCCs:
  MFCCs are two dimensional matrices with width of the matrix being the temporal axis. These MFCCs are stored in a JSON file. These JSON files are used in deep learning models.

At the end of this stage, we have the following files.

- Features_clipped.csv:
  This file contains features that are extracted for the 3 second clipped audio file.
- MFCC_clipped.json:
  This file contains the MFCC matrix of each audio file. All the matrices correspond to the three seconds audio files only.
- Features_unclipped.csv:
  This file contains features that are extracted for the complete audio file.



```
data.shape
(1049, 36)
```



```
data.shape
(585, 36)
```

The first image is the shape of the Features_clipped.csv file and the second image is the shape of the Features_unclipped.csv file.

6. Reducing Feature space of MFCC

MFCCs are a very important feature of audio data. But they are so huge that a simple machine learning algorithm cannot handle that. So, we can use Auto encoders to reduce the feature space of the recorded MFCC.

The MFCCs that are extracted are having the dimension of 130 X 30. It makes it 3900 features. This is a very huge number when compared to the number of samples collected. So, it is very important for us to reduce the feature space using some techniques.

Auto encoders [14, 15, 16] proved to be very useful in reducing feature space of very large dimensions. In a situation where we have 3900 features and only ~1000 samples, we cannot use simple unsupervised techniques like clustering or PCA. It calls for more advanced techniques like auto encoders.

Auto encoders are a kind of architecture that tries to recreate the same input using much smaller data. Auto encoders have two parts attached to it, encoder and decoder. Encoder reduces the dimension of the input and decoder tries to recreate the input using the reduced feature space. This makes it a very flexible unsupervised model. User has the freedom to choose the number of dimensions he/she wants to reduce the input to. This architecture works well for image and audio data as well. But the user needs to identify and choose the latent feature space dimensions wisely. Only then, the user can expect good results from the model.

Auto encoders need to be built from scratch. There are no predefined classes for autoencoders in tensorflow. In this project, a CNN based auto encoder is built using 2D Conv and Dense objects in tensorflow.

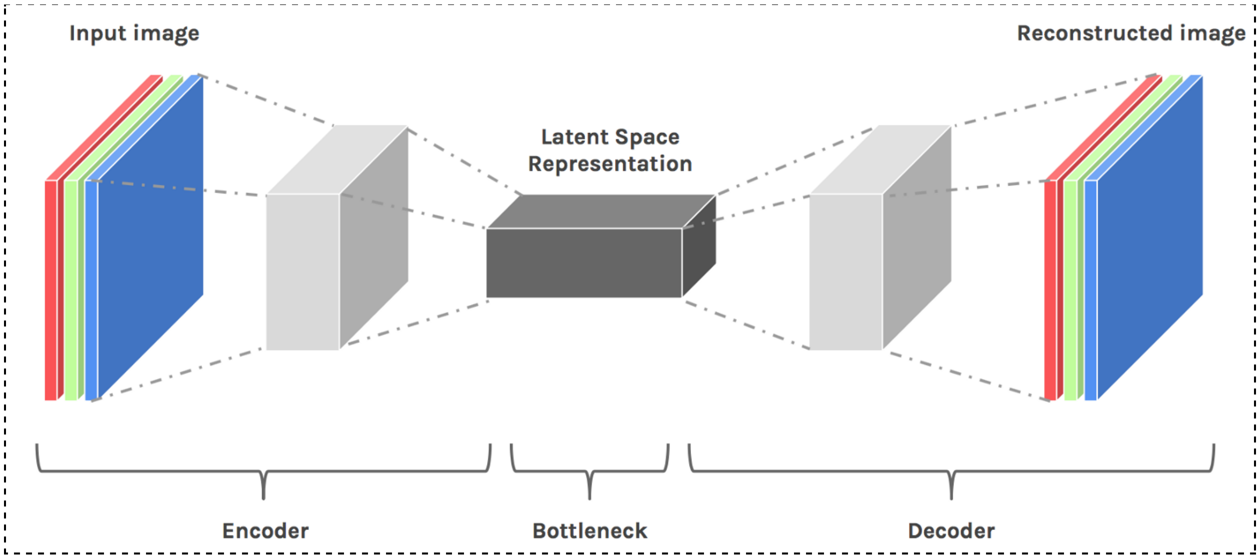
As we can see, the auto encoder has two models in it, encoder model and decoder model. Encoder is giving an output of 60 neurons. It means that the encoder is condensing 3900 dimension feature space into 60 dimension feature space. This is a great reduction in the number of dimensions. We are also avoiding the curse of dimensionality. This autoencoder has a total of 8,322,045 parameters. Number of dimensions has been reduced by 65 times but this reduction also has an error associated with it. This autoencoder is trained using mean squared error. We can only identify the loss when we actually build a machine learning model using this latent space.

We will now take a deeper look into the encoder and decoder part.

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder_input (InputLayer)   [(None, 130, 30, 1)]      0         
_________________________________________________________________
encoder_conv_layer_1 (Conv2D (None, 130, 30, 32)       320       
_________________________________________________________________
encoder_relu_1 (ReLU)        (None, 130, 30, 32)       0         
_________________________________________________________________
encoder_bn_1 (BatchNormaliza (None, 130, 30, 32)       128       
_________________________________________________________________
encoder_conv_layer_2 (Conv2D (None, 65, 15, 64)        51264     
_________________________________________________________________
encoder_relu_2 (ReLU)        (None, 65, 15, 64)        0         
_________________________________________________________________
encoder_bn_2 (BatchNormaliza (None, 65, 15, 64)        256       
_________________________________________________________________
encoder_conv_layer_3 (Conv2D (None, 65, 15, 64)        102464    
_________________________________________________________________
encoder_relu_3 (ReLU)        (None, 65, 15, 64)        0         
_________________________________________________________________
encoder_bn_3 (BatchNormaliza (None, 65, 15, 64)        256       
_________________________________________________________________
encoder_conv_layer_4 (Conv2D (None, 65, 15, 64)        102464    
_________________________________________________________________
encoder_relu_4 (ReLU)        (None, 65, 15, 64)        0         
_________________________________________________________________
encoder_bn_4 (BatchNormaliza (None, 65, 15, 64)        256       
_________________________________________________________________
encoder_conv_layer_5 (Conv2D (None, 65, 15, 64)        102464    
_________________________________________________________________
encoder_relu_5 (ReLU)        (None, 65, 15, 64)        0         
_________________________________________________________________
encoder_bn_5 (BatchNormaliza (None, 65, 15, 64)        256       
_________________________________________________________________
flatten (Flatten)            (None, 62400)             0         
_________________________________________________________________
encoder_output (Dense)       (None, 60)                3744060   
=================================================================
Total params: 4,104,188
Trainable params: 4,103,612
Non-trainable params: 576
```

```
_________________________________________________________________
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   [(None, 60)]              0
_________________________________________________________________
decoder_dense (Dense)        (None, 62400)             3806400
_________________________________________________________________
reshape (Reshape)            (None, 65, 15, 64)        0
_________________________________________________________________
decoder_conv_transpose_layer (None, 65, 15, 64)        102464
_________________________________________________________________
decoder_relu_1 (ReLU)        (None, 65, 15, 64)        0
_________________________________________________________________
decoder_bn_1 (BatchNormaliza (None, 65, 15, 64)        256
_________________________________________________________________
decoder_conv_transpose_layer (None, 65, 15, 64)        102464
_________________________________________________________________
decoder_relu_2 (ReLU)        (None, 65, 15, 64)        0
_________________________________________________________________
decoder_bn_2 (BatchNormaliza (None, 65, 15, 64)        256
_________________________________________________________________
decoder_conv_transpose_layer (None, 65, 15, 64)        102464
_________________________________________________________________
decoder_relu_3 (ReLU)        (None, 65, 15, 64)        0
_________________________________________________________________
decoder_bn_3 (BatchNormaliza (None, 65, 15, 64)        256
_________________________________________________________________
decoder_conv_transpose_layer (None, 130, 30, 64)       102464
_________________________________________________________________
decoder_relu_4 (ReLU)        (None, 130, 30, 64)       0
_________________________________________________________________
decoder_bn_4 (BatchNormaliza (None, 130, 30, 64)       256
_________________________________________________________________
decoder_conv_transpose_layer (None, 130, 30, 1)        577
_________________________________________________________________
sigmoid_layer (Activation)   (None, 130, 30, 1)        0
=================================================================
Total params: 4,217,857
Trainable params: 4,217,345
Non-trainable params: 512
```

```
_________________________________________________________________
Model: "autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   [(None, 130, 30, 1)]      0
_________________________________________________________________
encoder (Functional)         (None, 60)                4104188
_________________________________________________________________
decoder (Functional)         (None, 130, 30, 1)        4217857
=================================================================
Total params: 8,322,045
Trainable params: 8,320,957
Non-trainable params: 1,088
_________________________________________________________________
```



Img src: https://miro.medium.com/max/875/1*LSYNW5m3TN7xRX61BZhoZA.png

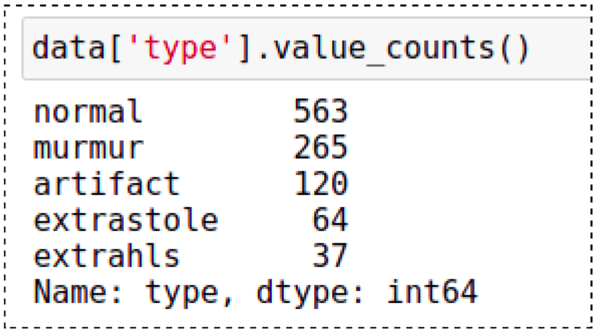
As we can see from the image, a high dimensional image is compressed into a lower dimensional latent space. The image is reconstructed using this lower dimensional latent space. The decoder is an exact mirror image of the encoder model. In our model, we constructed an encoder with five convolution layers and then a flatten layer which is connected to the bottleneck dimension. The mirror image of it is done in the decoder. Tensorflow provides inverse operation of convolution by name convolution transpose. That object is used to mirror the convolution operation.

The model is trained for 100 epochs with mean squared loss function.

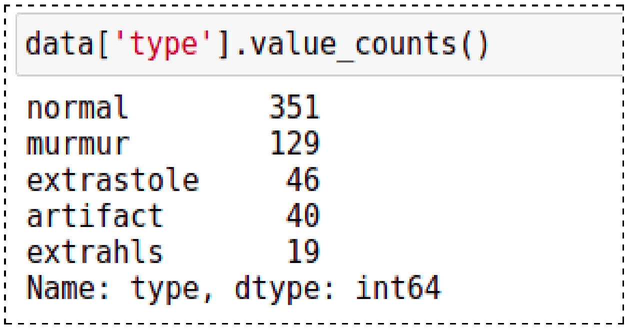
7. Data Modelling

Once the feature extraction is performed, we need to perform certain preprocessing for the csv and JSON files so that they are acceptable to the models we use.

Before modelling the data, let us see the class distribution of the audio files.



```
data['type'].value_counts()

normal        563
murmur        265
artifact      120
extrastole     64
extrahls       37
Name: type, dtype: int64
```



```
data['type'].value_counts()

normal        351
murmur        129
extrastole     46
artifact       40
extrahls       19
Name: type, dtype: int64
```

The first image is the distribution of classes extracted from the clipped audio files and the second image is the distribution of classes extracted from the complete audio file. We can see that there is a lot of class imbalance across classes. Some algorithms like boosting algorithms need not have a balanced class to perform well. But, data class imbalance has to be addressed for most of the other algorithms. We will see in the coming sections on how we dealt with the issue of class imbalance.

7.1 Data Scaling

Data can be directly passed to tree based algorithms. Tree based algorithms do not need the data to be scaled. So, the data is passed directly to the tree based algorithms.

Algorithms like SVM and Naive Bayes require the data to be standardized. Else, the loss function may be affected. The data is scaled using a standard scalar function in the sklearn package. Standard scalar subtracts the mean and standardizes the variance of the data. This standardized data is passed as input to the models like Gaussian Naive Bayes and support vector classifier algorithms.

The MFCCs are stored in a JSON file. This JSON file is converted to a two dimensional numpy array. This numpy array is given as input to our recurrent neural network model.

But the convolution neural network asks each input to be a three dimensional sample. Hence, we need to add a new dimension to the matrix. We can convert the two dimensional numpy array to a three dimensional numpy array by adding another axis to the matrix. This three dimensional matrix is passed as input for the convolution neural network.

8. Building Models

We have three different datasets. A JSON file and two csv files. Each csv file corresponds to clipped and unclipped audio files. We processed our datasets so that they can be passed as inputs to the model.

For testing purposes, we need to keep some data aside. For this, we will split the available dataset into two parts, train and test. We will use the train dataset to build the models. We will use the test dataset to check how good our model is. All the hyper parameter tuning is performed using the train dataset and not the test dataset.

For all the machine learning models, 25% of the data is set aside for testing purposes and 75% of the extracted samples are used for training the models. For deep learning models, 20% is set for testing and another 20% is set aside for validation purposes. We use the validation set to monitor the development of the model at different epochs.

This splitting of data is performed in a stratified way. This is to ensure that all the distribution of classes is not changed.

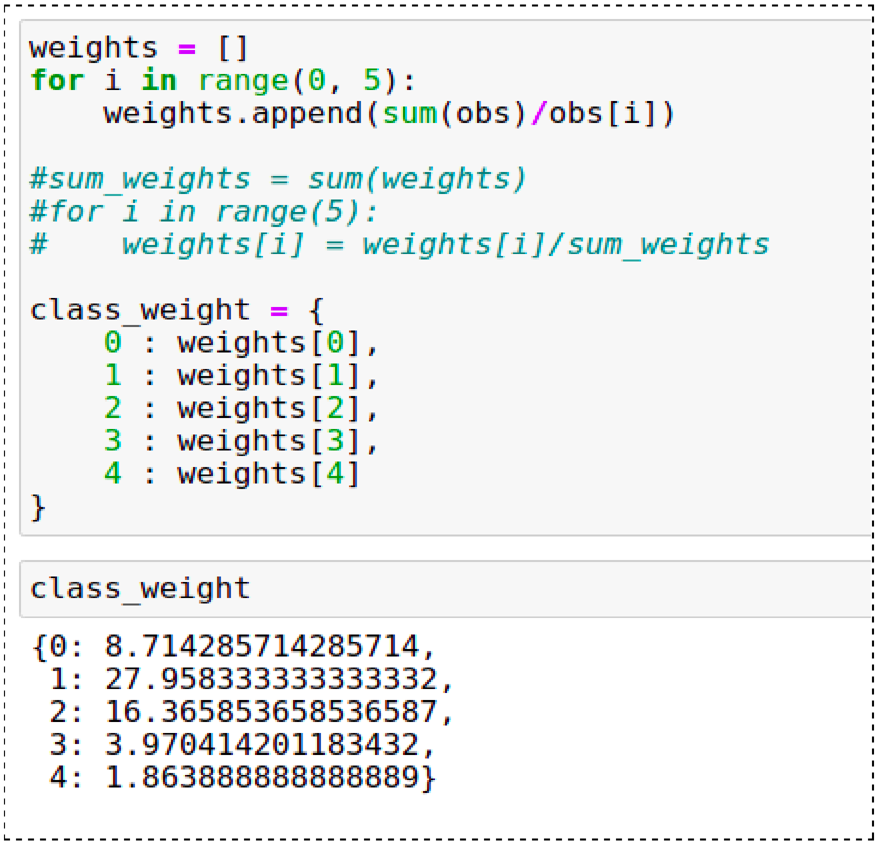
We will also deal with class imbalance in this stage.

8.1 Dealing with class imbalance

Class imbalance [18] plays an important role in determining the accuracy of the models. It is very much necessary to deal with class imbalance because it can affect the accuracy of the model. There are different ways to deal with class imbalance. But, for this project, we are going to use one of the very simple methods of assigning the weights to the samples.

In this technique, we will assign specific weights to each sample. Samples of the same class are assigned the same weight. The weights of the samples are calculated based on the frequency of the class in the dataset.

We calculated the class weight dictionary. Using this class weight, we will assign weights to each sample. These weights will be used while calculating the loss function.



```
weights = []
for i in range(0, 5):
    weights.append(sum(obs)/obs[i])

#sum_weights = sum(weights)
#for i in range(5):
#    weights[i] = weights[i]/sum_weights

class_weight = {
    0 : weights[0],
    1 : weights[1],
    2 : weights[2],
    3 : weights[3],
    4 : weights[4]
}
```

```
class_weight
```

```
{0: 8.714285714285714,
 1: 27.958333333333332,
 2: 16.365853658536587,
 3: 3.970414201183432,
 4: 1.8638888888888889}
```

We will now see summaries of all the models that are built using the three different datasets. We will iterate through the datasets and compare the models.

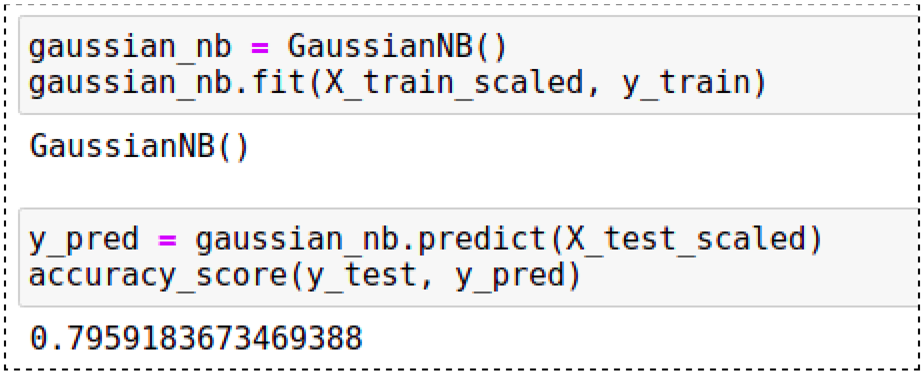
8.2 Features_unclipped.csv

This dataset contains the features extracted from the audio file as a whole. These models do have an advantage when they are kept in production. A bit of preprocessing step can be skipped for using these models. This is because the audio files need not be split to pass them to the model. Six different machine learning models are built using this dataset. We will iterate through each model.

8.2.1 Gaussian Naive Bayes

Gaussian Naive Bayes is a variation of Naive Bayes. Naive Bayes is a probability based classification model. It is based on conditional probability theorem and Bayes theorem. In Naive bayes, we find the probability of a sample falling into a category based on the independent features of each sample.

In Gaussian Naive Bayes, the probability of the sample will follow a Gaussian distribution. This is an advancement of Naive Bayes and it showed better accuracy than Naive Bayes on many occations.

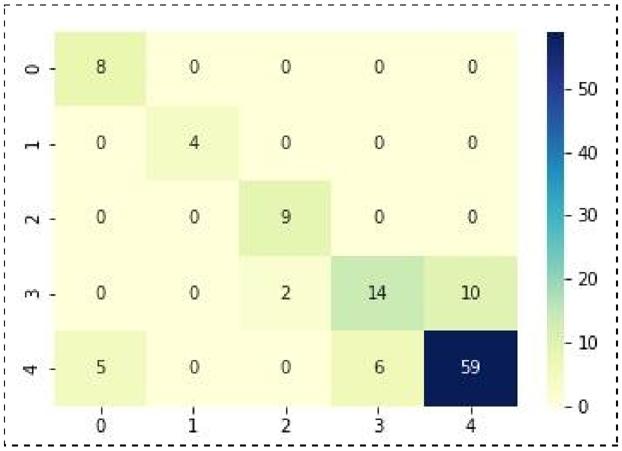


```
gaussian_nb = GaussianNB()
gaussian_nb.fit(X_train_scaled, y_train)
```

```
GaussianNB()
```

```
y_pred = gaussian_nb.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

```
0.7959183673469388
```

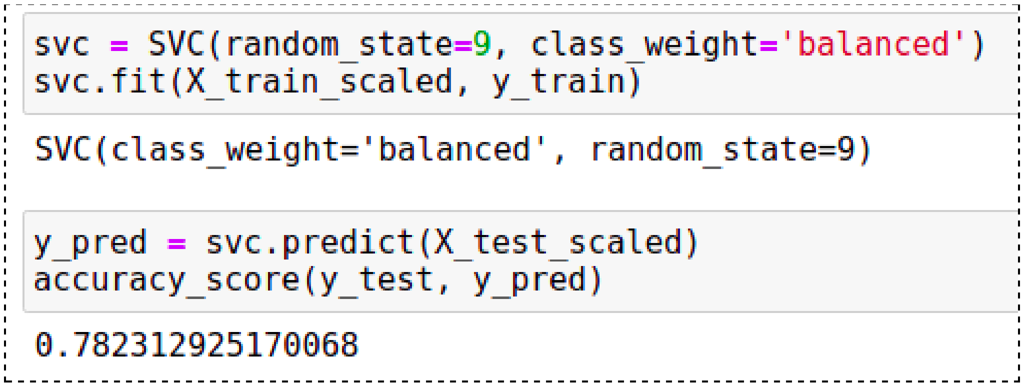


	0	1	2	3	4
0	8	0	0	0	0
1	0	4	0	0	0
2	0	0	9	0	0
3	0	0	2	14	10
4	5	0	0	6	59

Gaussian Naive Bayes gave an accuracy of 79.5%. But when we see the confusion matrix, we can see that it was not able to classify the classes 3 and 4 correctly. So, the model works good for classes 0, 1 and 2 but is unable to classify properly for classes 3 and 4.

8.2.2 Support Vector Classifier

Support Vector Classifier [17] is basically a classification algorithm that gives the best discriminator vector based on maximum margin. They work well with high dimensional data but become slow with increase in sample size. For this dataset, it is ideal because the number of features are high and the sample size is relatively not too large.

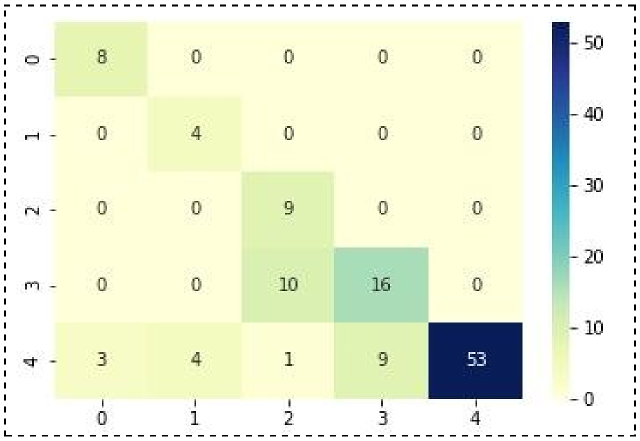


```
svc = SVC(random_state=9, class_weight='balanced')
svc.fit(X_train_scaled, y_train)
```

```
SVC(class_weight='balanced', random_state=9)
```

```
y_pred = svc.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

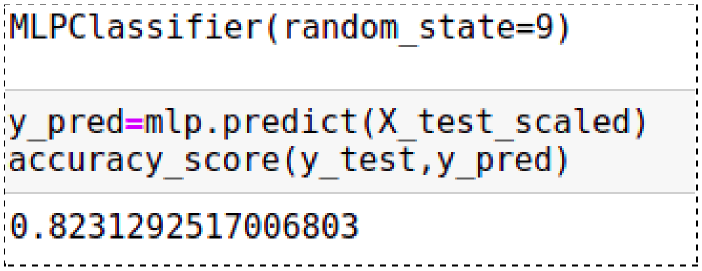
```
0.782312925170068
```



Support Vector Classifier also worked well with most of the classes. This model is also unable to classify the classes 3 and 4 correctly.

8.2.3 Multi Layer Perceptron

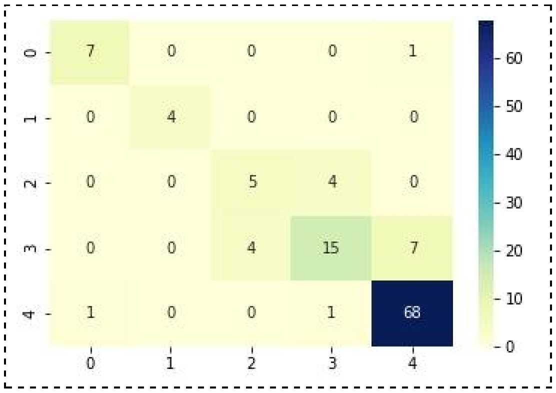
Multi layer perceptron is a fully connected neural network model. It learns non linear relations between the input variables and the output variable. Here, we used a classifier architecture because we are dealing with classification problems. A simple neural network is used here. This architecture contains a single hidden layer with 100 neurons in it. It uses lbfgs optimizer. Lbfgs is an optimizer that belongs to the quasi Newton family of optimizers. This architecture is trained for 100 epochs.



```
MLPClassifier(random_state=9)
```

```
y_pred=mlp.predict(X_test_scaled)
accuracy_score(y_test,y_pred)
```

```
0.8231292517006803
```



This algorithm worked better than the previous two algorithms. It is able to classify all the classes fairly well. This model is able to perform better than other two models with respect to class 4. But it is unable to properly classify the classes 2 and 3.

8.2.4 Random Forest Classifier

Random forest classifier [12] is a kind of ensemble technique called bagging. This algorithm constructs several decision trees by taking a part of the data and a part of feature space. This method ensures that the model does not overfit and can generalize to unseen data.

```
forest = RandomForestClassifier(random_state=31, class_weight='balanced', oob_score=True)
forest.fit(X_train, y_train)
```

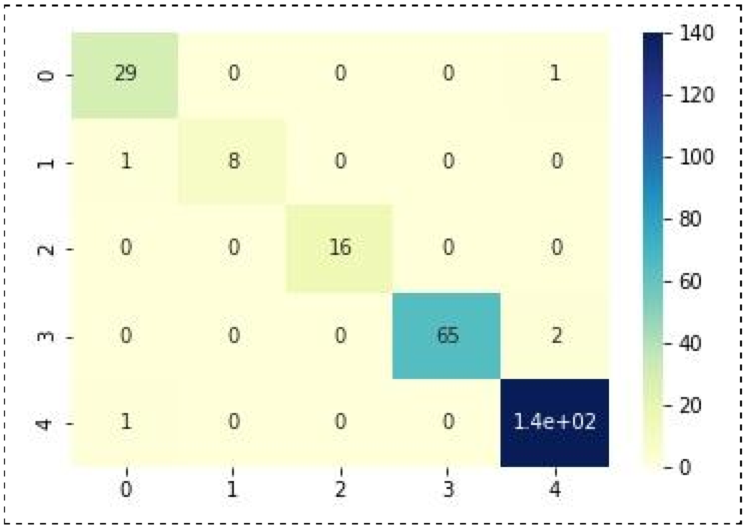
```
RandomForestClassifier(class_weight='balanced', oob_score=True, random_state=31)
```

```
y_pred = forest.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.9455782312925171
```

```
forest.oob_score_
```

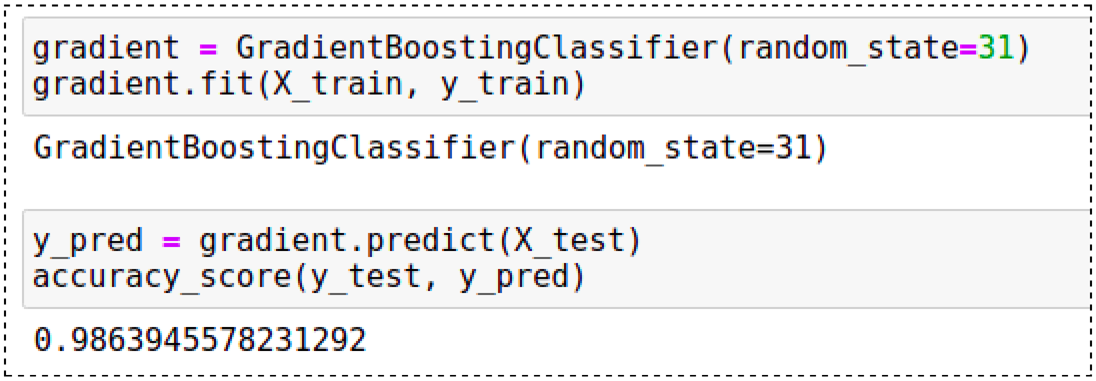
```
0.9200913242009132
```



This is the best model we saw till now. It is giving an accuracy of around 94.5%. Only a few samples are classified incorrectly. This model did fairly well on unseen data also.

8.2.5 Gradient Boosting

Gradient boosting [13] is another type of ensemble method called boosting. This algorithm works sequentially. It learns the misclassified samples more faster than the properly classified samples. This way, it boosts its accuracy by learning all the samples. It has high chances of overfitting. But that is controlled by using decision stumps as a base learner.

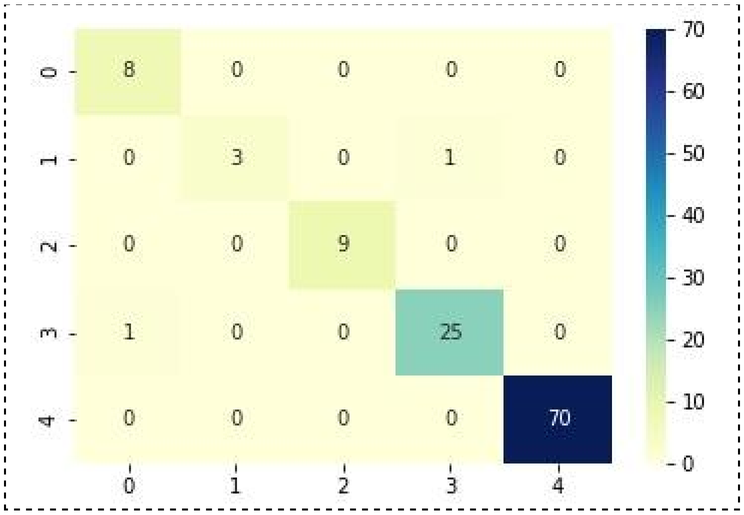


```
gradient = GradientBoostingClassifier(random_state=31)
gradient.fit(X_train, y_train)
```

```
GradientBoostingClassifier(random_state=31)
```

```
y_pred = gradient.predict(X_test)
accuracy_score(y_test, y_pred)
```

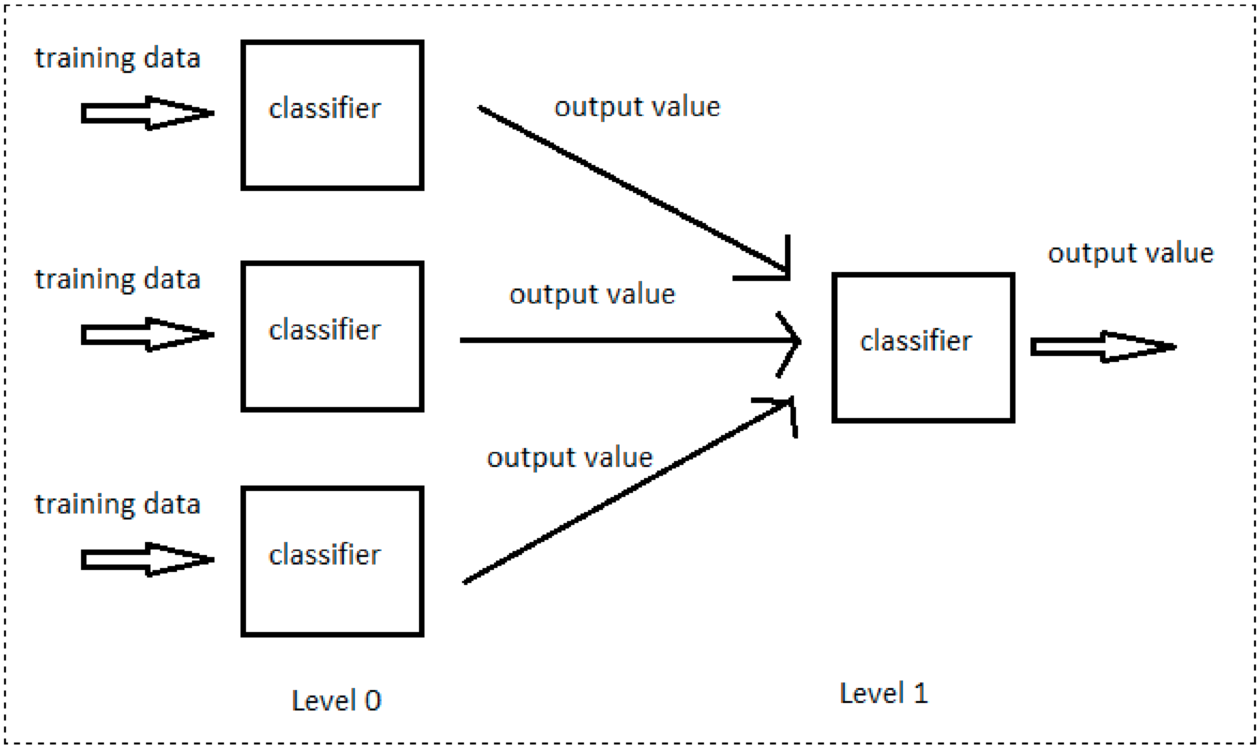
```
0.9863945578231292
```

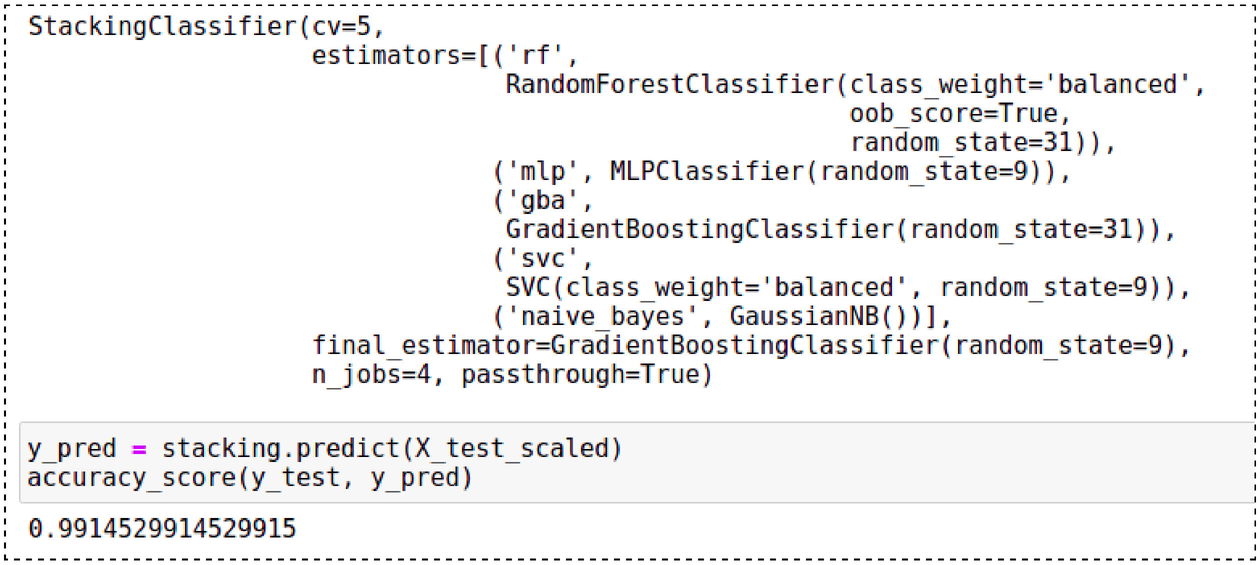


By far, this is the best classifier. It is giving test accuracy of around 98.6%. Very few samples are misclassified.

8.2.6 Stacking Classifier

We observed that few algorithms are able to classify few classes correctly while unable to learn other classes. This calls for another ensemble technique called Stacking Classifier. Using this technique, we can combine different algorithms and create a meta classifier that learns well on all the classes. Stacking Classifier is known to give very good results even when the level 0 classifiers are not very good.



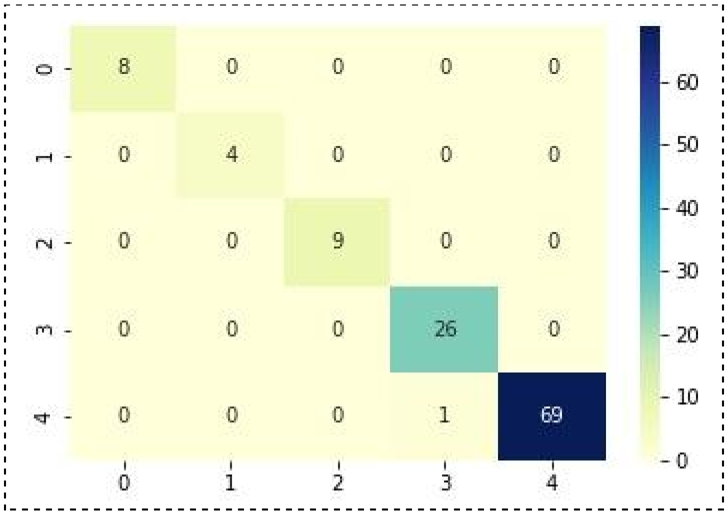


```
StackingClassifier(cv=5,
                   estimators=[('rf',
                                RandomForestClassifier(class_weight='balanced',
                                                       oob_score=True,
                                                       random_state=31)),
                               ('mlp', MLPClassifier(random_state=9)),
                               ('gba',
                                GradientBoostingClassifier(random_state=31)),
                               ('svc',
                                SVC(class_weight='balanced', random_state=9)),
                               ('naive_bayes', GaussianNB())],
                   final_estimator=GradientBoostingClassifier(random_state=9),
                   n_jobs=4, passthrough=True)
```

```
y_pred = stacking.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

```
0.9914529914529915
```

This stacking classifier model is having 5 classifiers at level 0. We used the earlier mentioned classifiers in level 0. They are the random forest classifier, multi layer perceptron classifier, gradient boosting classifier, support vector classifier and Gaussian naive Bayes classifier. We used another gradient boosting classifier as the meta classifier.



This is the best model we got with the accuracy of more than 99% There is only one misclassified sample. This makes it a very good classifier.

8.2.7 Summary

Let us now look at the summary of all the models that are built using the dataset Features_unclipped.csv.

Model Name	Accuracy	Precision	Recall	F1 Score
Gaussian Naive Bayes	0.795	0.79	0.80	0.79
Support Vector Classifier	0.782	0.85	0.78	0.79
Multi Layer Perceptron	0.823	0.82	0.82	0.82
Random Forest Classifier	0.945	0.95	0.95	0.94
Gradient Boosting Classifier	0.986	0.99	0.99	0.99
Stacking Classifier	0.991	0.99	0.99	0.99

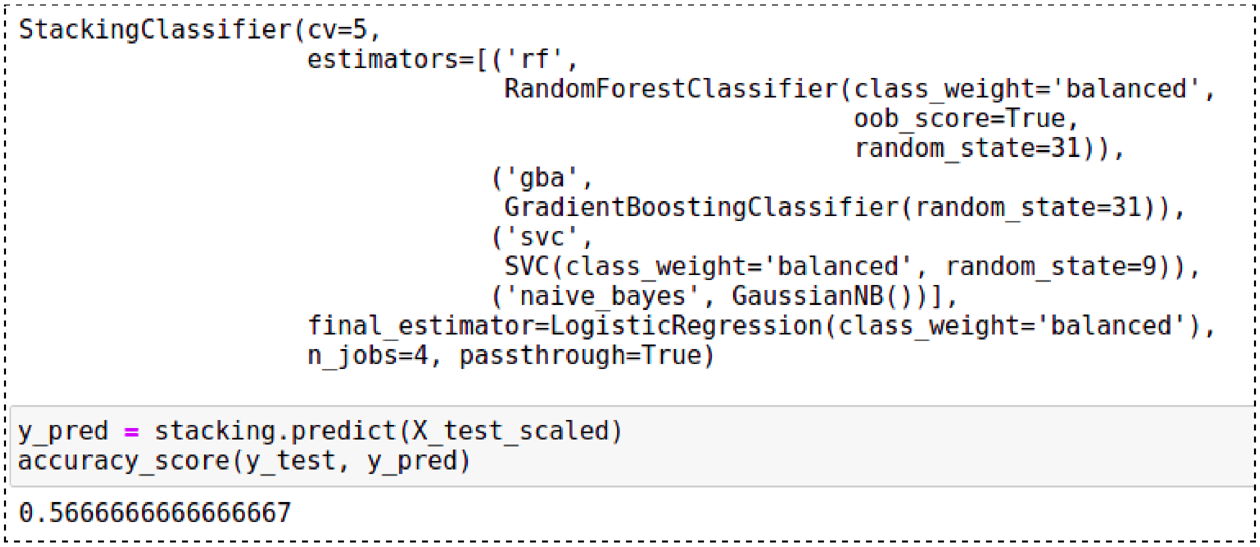
After comparing all the algorithms, we can see that Stacking Classifier gave the highest accuracy with only one sample in test data being incorrectly classified.

8.3 Features_clipped.csv

The dataset - Features_clipped.csv - contains features extracted from three second clips of heartbeat. By clipping the data into three second clips we were able to use the latent space that we created using autoencoders along with other features that we extracted.

8.3.1 Use of Auto Encoders

Autoencoder is trained on JSON file that contain the MFCCs. Using that model, we are able to reduce the MFCC dimensions from 3900 dimensions to just 60 dimensions. We used this condensed feature space along with other features and built a stacking classifier.



```
StackingClassifier(cv=5,
                   estimators=[('rf',
                                RandomForestClassifier(class_weight='balanced',
                                                       oob_score=True,
                                                       random_state=31)),
                               ('gba',
                                GradientBoostingClassifier(random_state=31)),
                               ('svc',
                                SVC(class_weight='balanced', random_state=9)),
                               ('naive_bayes', GaussianNB())],
                   final_estimator=LogisticRegression(class_weight='balanced'),
                   n_jobs=4, passthrough=True)
```

```
y_pred = stacking.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

```
0.5666666666666667
```

The stacking classifier did not perform good on the dataset. One reason could be that the latent space itself is not accurate enough. We can say that a lot of information was lost while compressing the feature space.

8.3.2 Summary

Different models are built on this dataset and here we show a summary of all the models. We will compare each model and get the best model for this dataset.

Model Name	Accuracy	Precision	Recall	F1 Score
Gaussian Naive Bayes	0.688	0.72	0.69	0.69
Support Vector Classifier	0.863	0.90	0.86	0.87
Multi Layer Perceptron	0.951	0.95	0.95	0.95
Random Forest Classifier	0.981	0.98	0.98	0.98
Gradient Boosting Classifier	0.984	0.98	0.98	0.98
Stacking Classifier	0.988	0.99	0.99	0.99
Stacking Classifier using AE	0.566	0.67	0.57	0.58

The score of several algorithms are comparatively higher than the models that are built using the previous dataset. But the best model on this dataset is still inferior to the best model of the previous dataset. Unexpectedly, the stacking classifier built using an autoencoder is the worst performing model.

8.4 MFCC.json

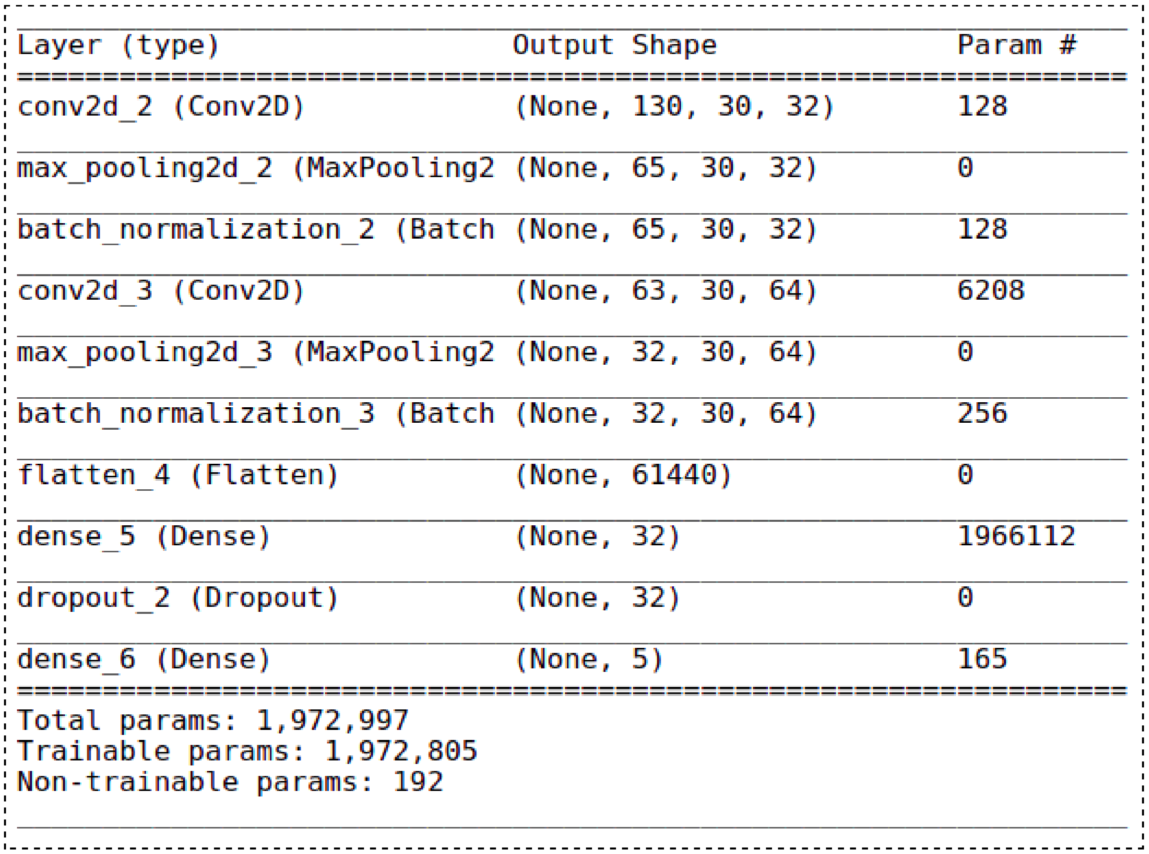
MFCC.json contains an array of matrices. These matrices are the MFCCs that are extracted from the three second audio clips. These matrices have the dimension of 130 X 30. They are also temporal matrices. The 130 rows are sequentially extracted from the audio clip. The 30 columns are independent of each other. But the 130 rows are sequential in the temporal axis.

This data is not balanced. It suffers from class imbalance. So we created a dictionary containing the appropriate weights of each class. We need to pass this dictionary to the models while training the models. This will ensure that models do not bias towards classes that contain high samples.

Advanced Deep Learning architectures like CNN and RNN can make use of such temporal properties of the matrices. We will see how CNN and RNN kind of architectures are implemented to classify such matrices.

8.4.1 Convolution Neural Network

Convolution Neural Network [10] is a type of neural network that uses mathematical operation convolution. Convolution neural network is a locality sensitive architecture. This can be used here because our matrices contain temporal data. This temporal data is highly dependent on nearby (in time domain) features.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 130, 30, 32)       128       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 65, 30, 32)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 65, 30, 32)        128       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 63, 30, 64)        6208      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 30, 64)        0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 30, 64)        256       
_________________________________________________________________
flatten_4 (Flatten)          (None, 61440)             0         
_________________________________________________________________
dense_5 (Dense)              (None, 32)                1966112   
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 165       
=================================================================
Total params: 1,972,997
Trainable params: 1,972,805
Non-trainable params: 192
_________________________________________________________________
```

We implemented an architecture that contains two convolution layers, two max pooling layers, two dense layers, two batch normalization layers and a dropout layer.

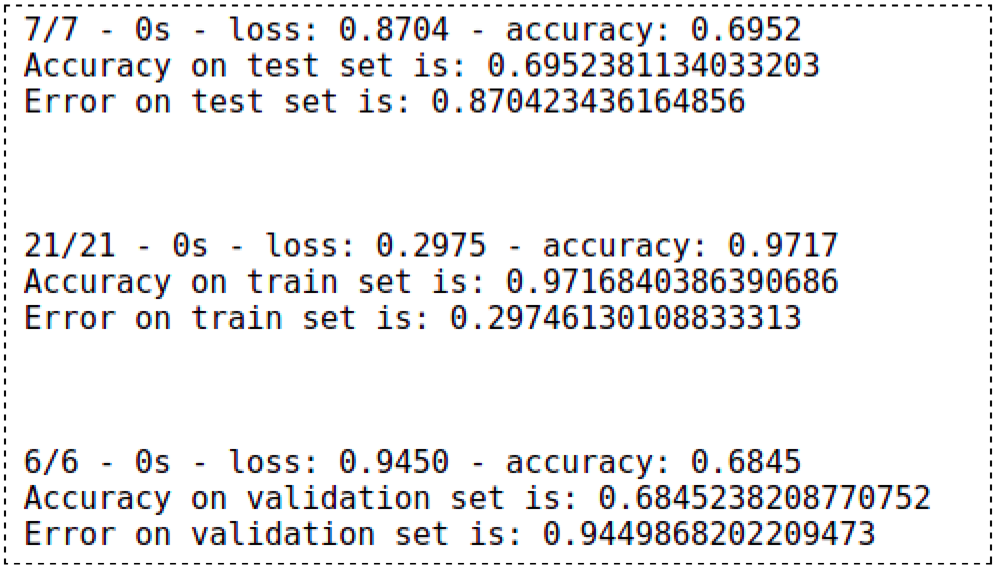
The first layer is a Convolution layer that contains 32 convolution kernels of size 3X1. This is because we know that each column in that sample matrix corresponds to a single MFCC. Each MFCC is independent of the other. Hence, kernels are not spanned across columns.

We then applied a 2D max pooling layer. Here again, the window size is 2X1 because of the reason stated earlier. This is followed by a Batch Normalization layer.

We now again apply a convolution layer with 64 kernels of each size 3X1. Then, again, we apply a max pooling layer and a batch normalization layer.

We now have a matrix of size 32X30X64. We now flatten the matrix so that we can connect it to the dense layers. We connect it to a dense layer of 32 neurons followed by a dropout layer and final output layer of 5 neurons. Each of these 5 neurons represent a class of heartbeat. All the layers of the architecture have ReLU activation functions except for the final layer that has softmax activation function.

We used Adam optimizer with sparse categorical cross entropy as loss function. We trained the model for 60 epochs with batch size of 64.



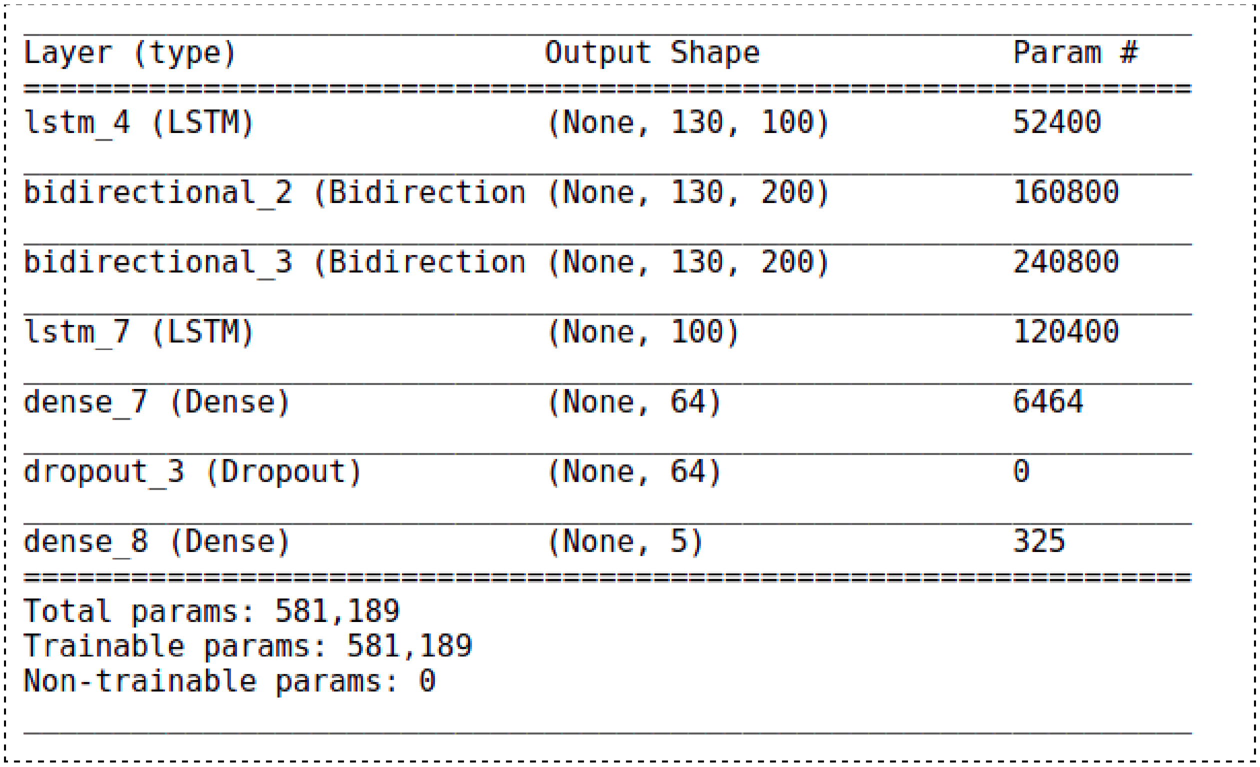
```
7/7 - 0s - loss: 0.8704 - accuracy: 0.6952
Accuracy on test set is: 0.6952381134033203
Error on test set is: 0.870423436164856



21/21 - 0s - loss: 0.2975 - accuracy: 0.9717
Accuracy on train set is: 0.9716840386390686
Error on train set is: 0.29746130108833313



6/6 - 0s - loss: 0.9450 - accuracy: 0.6845
Accuracy on validation set is: 0.6845238208770752
Error on validation set is: 0.9449868202209473
```

From this output, we can see that the architecture learnt well on train data but it is unable to generalize on the unseen data. This is a problem of overfitting.

8.4.2 Recurrent Neural Network

Recurrent Neural Network [11] is a kind of architecture that sustains information from previous data. So, it is best suited for temporal data. Here, we have temporal audio data. So, we can use RNN based architectures.

Here, we are going to use a mixture of LSTM and Bi Directional LSTM architecture. LSTM models are known to possess the power of memorising over long time gaps. Bi Directional LSTM not only uses previous memory but also future memory to influence the current weight of the parameter. Since, we are dealing with temporal data, these architectures come handy while classifying the data.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_4 (LSTM)                (None, 130, 100)          52400     
_________________________________________________________________
bidirectional_2 (Bidirection (None, 130, 200)          160800    
_________________________________________________________________
bidirectional_3 (Bidirection (None, 130, 200)          240800    
_________________________________________________________________
lstm_7 (LSTM)                (None, 100)               120400    
_________________________________________________________________
dense_7 (Dense)              (None, 64)                6464      
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_8 (Dense)              (None, 5)                 325       
=================================================================
Total params: 581,189
Trainable params: 581,189
Non-trainable params: 0
_________________________________________________________________
```

Our architecture contains two LSTM layers, two bi directional layers, two dense layers and a dropout layer.

The first layer is an LSTM layer. This LSTM layer tries to remember from previous 100 values. In the temporal axis. Then we have a bi directional LSTM which learns 100 values from the past and 100 values from the future. This is followed by another bi directional layer. We then have another LSTM layer.

We connected the last LSTM layer with a Dense layer with 64 neurons. We then introduced a Dropout layer with dropout percentage fixed at 30. It is finally connected to the output layer that contains 5 neurons, each representing a class of output.

There are a total of 581,189 parameters that this network is learning.

It uses Adam optimizer with learning rate of 0.0001 and loss as sparse categorical cross entropy. It is trained for 100 epochs with a batch size of 32 samples.

```
7/7 - 0s - loss: 1.0731 - accuracy: 0.7429
Accuracy on test set is: 0.7428571581840515
Error on test set is: 1.0730655193328857



21/21 - 0s - loss: 0.1449 - accuracy: 0.9568
Accuracy on train set is: 0.9567809104919434
Error on train set is: 0.14486941695213318



6/6 - 0s - loss: 1.1397 - accuracy: 0.6905
Accuracy on validation set is: 0.6904761791229248
Error on validation set is: 1.13969087600708
```

From the output, we can see that the accuracy of LSTM is not very encouraging. There could be many reasons for this. One reason could be that there is not much information that can be learnt from the temporal property of the matrix.

8.4.3 Summary

We now compare the two models.

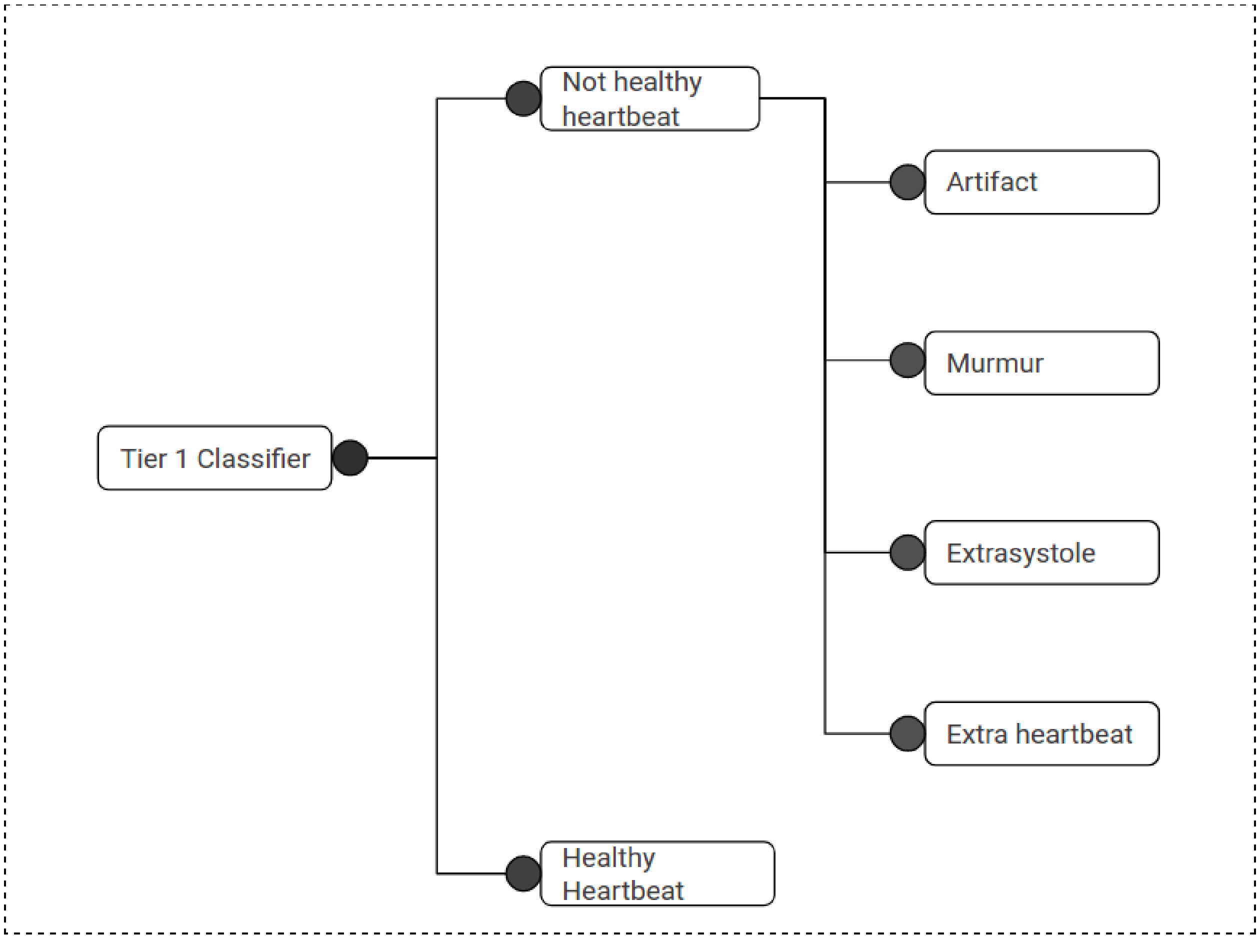
	CNN based Model	RNN based Model
Number of parameters	1,972,997	581,189
Number of Epochs ran	60	100
Train Accuracy	0.971	95.6%
Validate Accuracy	0.684	69.04%
Test Accuracy	0.695	74.28%

9. A Novel Approach

In this section, we are going to propose a new method of solving this classification problem. We divide the problem statement into two different stages. The first stage deals with identifying if the heartbeat is normal heartbeat or diseased heartbeat. In this stage we train a binary classification model.

The second stage deals with identifying the type of heart disease. We train models in this stage using only the four subcategories of the data. We do not train on the normal heartbeat data. Hence, we can pass data to this model only if the model in the previous stage does not classify a healthy heartbeat as unhealthy.

The data that is passed to tier two are the samples that are predicted as abnormal. Hence, there is a dependence between the tier 1 and the tier 2 model.



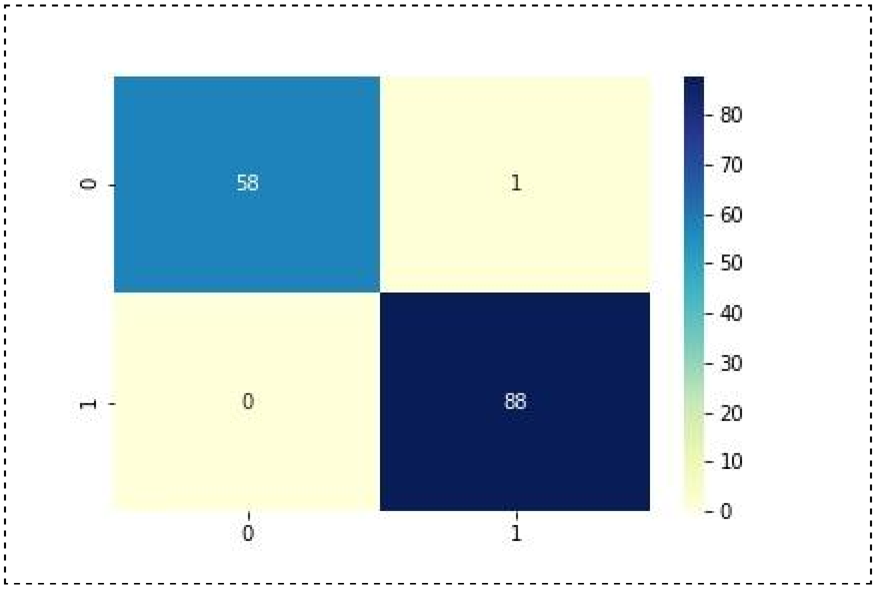
Different algorithms like random forest and gradient boosting are implemented at both the levels.

9.1 Tier 1

Tier 1 is a binary classification problem. It differentiates between healthy heartbeat and unhealthy heartbeat. The below table gives us the accuracies of some models that are used in tier 1

	Accuracy	Precision	Recall	F1 score
Support Vector Classifier	0.877	0.9	0.88	0.88
Random Forest Classifier	0.959	0.96	0.96	0.96
Gradient Boosting Classifier	0.993	0.99	0.99	0.99

Let us take a deeper look at the best model we got in tier 1.



The above confusion matrix corresponds to the Gradient boosting classifier. It has the accuracy of 99.31% and the only misclassified data tells that some unhealthy heartbeat is classified as a healthy heartbeat.

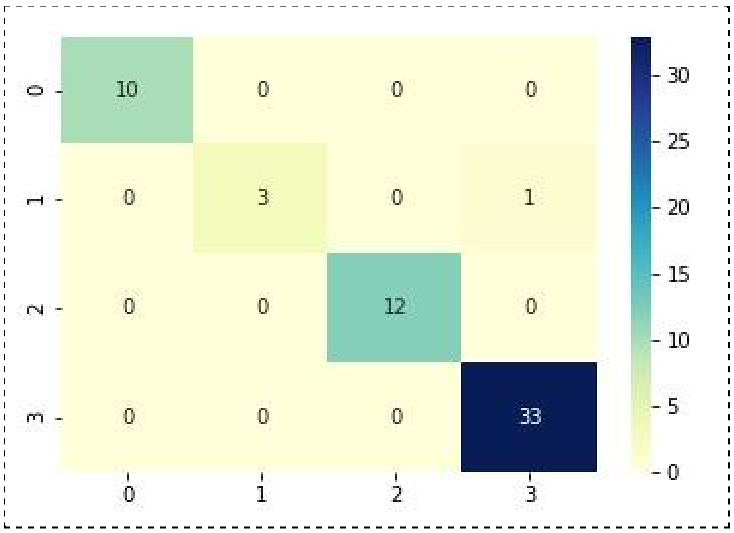
9.2 Tier 2

Tier 2 is a multi class classification problem that focuses on identifying the type of non healthy heartbeat. It categories the heartbeat in one of the 4 different categories. All models in this tier are built using only the four categories.

The below table gives accuracies and other classification metrics of the models that are built in the second tier.

	Accuracy	Precision	Recall	F1 score
Random Forest Classifier	0.949	0.95	0.95	0.95
Gradient Boosting Classifier	0.983	0.98	0.98	0.98

From the table, we can see that the gradient boosting classifier performed better than the random forest classifier. Here is the confusion matrix of gradient boosting classifier of tier 2.



The confusion matrix says that the gradient boosting classifier model is able to correctly classify all the classes except for one sample.

9.3 Summary

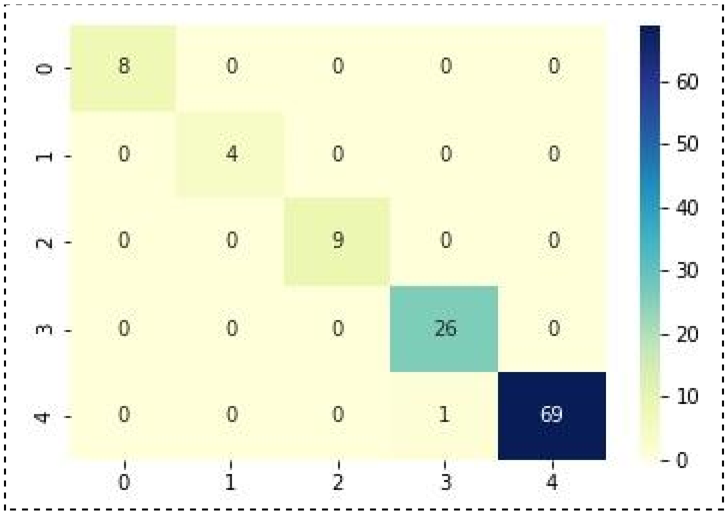
In conclusion, we built a two tier model that first classifies a heartbeat as healthy and unhealthy. It then classifies the heartbeat that is predicted as unhealthy into different types of heartbeats. We trained the models on the features extracted from the complete audio file.

We used gradient boosting classifiers in both the tiers.

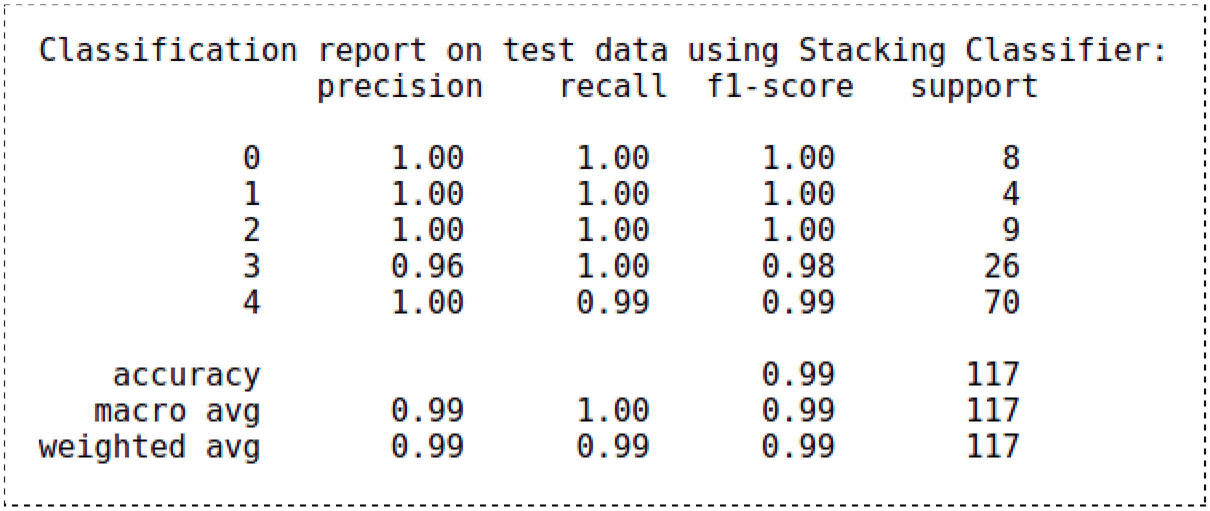
10. Results

We have chosen the two best models from many models that we trained and experimented on. We will now compare both these models.

One model is the stacking classifier model built on Features_unclipped.csv file. That model gave an accuracy of 99.1%



This above confusion matrix is the confusion matrix of the stacking classifier. The confusion matrix shows a healthy classification. The train data gave perfect accuracy and only one sample is misclassified in test data. From the matrix we can say that one sample of class 4 was actually classified as class 3.



```
Classification report on test data using Stacking Classifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         9
           3       0.96      1.00      0.98        26
           4       1.00      0.99      0.99        70

    accuracy                           0.99       117
   macro avg       0.99      1.00      0.99       117
weighted avg       0.99      0.99      0.99       117
```

The classification report also says the same. The recall, precision and f1 scores are very high.

The novel model that is designed gave the second best performance by just misclassifying only two audio files and rest all being classified correctly.

11. Comparison with published papers

Some work has been done by researchers on this dataset. We will now compare the results that we achieved with the results of other published papers.

11.1 Heartbeat sound classification using Deep Learning

The authors of the paper[9] implemented Multi Layer Perceptron and Recurrent Neural Network architectures. They split the audio files with the duration of 12.5 seconds and 27.8 seconds. They were able to achieve highest accuracy with a 12.5 second audio clip dataset. RNN (LSTM) gave the highest accuracy of 80.8%.

Their preprocessing included downsampling the sample rate to reduce the feature space. They only considered classifying heartbeat into three different classes, Normal, Murmur and Extra-Systole. No features are extracted. Data is passed as is to the model.

In our project, we used LSTM to classify audio clips into 5 different classes and we got test accuracy of 74.28%. We used feature extraction to extract MFCC from the audio file and we processed on that MFCC matrix.

11.2 Human Heart sounds classification using Ensemble methods

The authors of the paper [13] implemented ensemble techniques to classify the audio clips. They classified audio clips into four categories, Normal, Murmur, Extra heart sound and Artifact.

They extracted a total of 27 features which include 13 MFCCs, 10 Linear Predictive Coding (LPC) and 4 non spectral features. The non spectral features include fraction of low energy window, root mean square of frames, zero crossings and relative difference function. These four features are extracted using statistical techniques that include finding mean and standard deviation. Our features include 30 MFCCs and 4 other audio features. One feature belongs to the time domain and three other features belong to the frequency domain.

Their maximum accuracy reached 98.78% using an ensemble technique. Their technique consists of using random forest as base classifier and Ada Boost as the ensemble classifier. The best model in our project gave an accuracy of 99.1% using a Gradient Boosting Classifier that uses Decision stumps as base classifiers. Our model is built for classifying 5 class classification problem whereas this paper implemented a 4 class classification problem.

11.3 Summary

Let us compare the accuracies of our model with the models deployed in the other two papers.

11.3.1 Comparison with Paper-1

	Paper 1	Our model
Model type	LSTM	Bi directional LSTM
Accuracy	**80.8%**	74.5%
Input type	Down sampled audio file	MFCCs extracted from audio
Problem type	Three class classification	**Five class classification**

11.3.2 Comparison with Paper-2

	Paper 2	Our model
Best model	AdaBoost with random forest as base classifier	Stacking classifier with gradient boosting as meta learner
Best Accuracy	98.78%	**99.1%**
Number of features used	27	**34**
Problem type	Four class classification	**Five class classification**

12. Model Deployment

The main purpose of any model is to be used by people. It is of no use if the best model is not kept to use. Hence, we developed a web interface and integrated the best model with the interface. We then used the free tier cloud service of Heroku Cloud to publish our model. This model can be used from anywhere in the world and using any device.

The user needs to just upload the audio file to the web application. The uploaded audio file then goes through the process of feature extraction and prediction using the best model. The resultant output is then displayed to the user using the web interface.

12.1 Web Interface

The web interface is developed using a python framework called flask. Flask is easy to use and required very little code to build a web interface and make it running. Flask is very pythonic as to say because of its development process. It resembles python in the way the code is written. Hence it has become a very popular python web framework.

The front end will be developed using HTML, CSS and Vanilla Javascript. The Flask Framework controls the requests that users make.

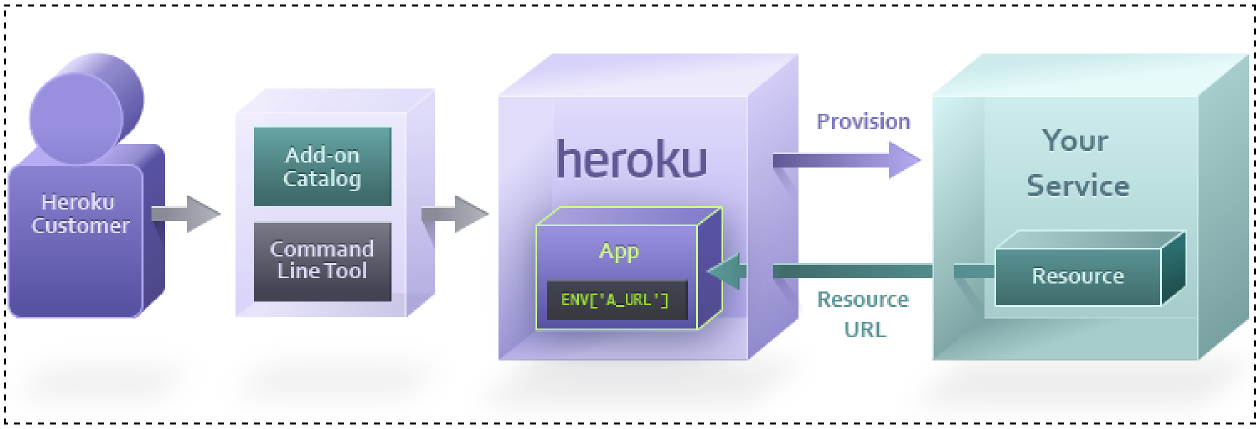
The Flask framework sends the requests to the python server.

The python server takes in the audio file and processes it. It passes the processed audio file to the model and gives a prediction on the audio file.

The server then sends the response to the Flask framework. The Flask framework then sends the response to the user through the front end UI.

12.2 Cloud Platform

The web framework and the model is built. They should be deployed using a cloud service so that they can be used from anywhere in the world. We chose Heroku Cloud Platform because of the advantages it provides over other cloud platforms for a python developer.



Img src: https://devcenter1.assets.heroku.com/article-images/1726-imported-1443570487-1726-imported-1443554965-how-provision1.png

Few of the advantages include:

- Allows the developer to focus on code instead of infrastructure
- Support form Modern Open Source Languages
- Beginner and startup-friendly

Heroku also provides free tier services that are very helpful for the projects that are in the development stage.

13. Scope and Conclusion

This project is just a spark in the ever evolving health care sector. The model is built on a static dataset. Model is built and then deployed. The model is built with ~500 audio samples. But, we can keep improving the model by making the model learn constantly. This is possible with the expertise of doctors. Doctors can continuously upload the audio file along with the category of the heartbeat. The model can then run on cloud platforms and continue to give better results.

The models used hare are discriminative models. The models learn the discriminator vector and not the class properties. We can build a generative model wherein it learns the distribution of each class. That would give more light on identifying each class instead of classifying the classes.

The model is currently hosted on cloud platform and can be accessed through any device. But, we need to pass the audio file to the web application. We can integrate this model with IoT devices so that it can monitor an individual's heart beat continuously. This model can be integrated with FitBit or similar devices. These devices read and record the heartbeat sounds. These sounds can be passed to the model integrated within the device itself. The user can track the health of his heart whenever the user wishes to.

14. References

1. Durgesh Majeti - Detecting Heart Anomalies using Heartbeat sound - https://bit.ly/3dusyyN
2. Classifying Heart Sounds Challenge - http://www.peterjbentley.com/heartchallenge/#taskoverview
3. Heartbeat Sounds dataset
4. Valerio Velardo - YouTube series on Audio data classification - https://www.youtube.com/playlist?list=PL-wATfeyAMNrtbkCNsLcpoAyBBRJZVlnf
5. Scikit learn documentation - https://scikit-learn.org/0.23/user_guide.html
6. Librosa documentation - https://librosa.org/doc/latest/index.html
7. Numpy documentation - https://numpy.org/doc/1.18/
8. Corey Schafer - YouTube series on Python Flask - https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH
9. Raza, Ali & Mehmood, Arif & Ullah, Dr. Saleem & Ahmad, Maqsood & Choi, Gyu Sang & On, Byung-Won. (2019). Heartbeat Sound Signal Classification Using Deep Learning. Sensors. 19. 10.3390/s19214819.
10. Choo, Keng & Low, Jia. (2018). Automatic Classification of Periodic Heart Sounds Using Convolutional Neural Network.
11. Rubin, Jonathan & Abreu, Rui & Ganguli, Anurag & Nelaturi, Saigopal & Matei, Ion & Sricharan, Kumar. (2017). Recognizing Abnormal Heart Sounds Using Deep Learning.
12. Pretorius, Arnu & Bierman, Surette & Steel, Sarel. (2016). A meta-analysis of research in random forests for classification. 1-6. 10.1109/RoboMech.2016.7813171.
13. Ali, Sajjad & Adnan, SM & Nawaz, T. & Obaidullah, M. & Aziz, Sumair. (2017). Human Heart Sounds Classification using Ensemble Methods.
14. Zhang, Y. (2018). A Better Autoencoder for Image: Convolutional Autoencoder.
15. Alqahtani, Ali & Xie, X. & Deng, J. & Jones, Mark. (2018). A Deep Convolutional Auto-Encoder with Embedded Clustering. 4058-4062. 10.1109/ICIP.2018.8451506.
16. Marco Maggipinto, Chiara Masiero, Alessandro Beghi, Gian Antonio Susto. (2018). A Convolutional Autoencoder Approach for Feature Extraction in Virtual Metrology. Procedia Manufacturing. Volume 17 Pages 126-133. ISSN 2351-9789.
17. Tian, Yingjie & Shi, Yong & Liu, Xiaohui. (2012). Recent advances on support vector machines research. Technological and Economic Development of Economy. 18. 10.3846/20294913.2012.661205.
18. Ms. Monica. Ochani , Dr. S. D. Sawarkar , Mrs. Swati Narwane, 2019, A Novel Approach to Handle Class Imbalance in Machine Learning, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 08, Issue 11 (November 2019)