HEART SOUND CLASSIFICATION

NLP MINI PROJECT REPORT SUBMITTED BY

Disha Ganesh Bhat
4NM17CS059
VIII Semester, B section

Glenita Shareen Furtado
4NM17CS066
VIII Semester, B section

Joshni Princia Saldanha
4NM17CS072
VIII Semester, B section

Mallika
4NM17CS098
VIII Semester, B section

UNDER THE GUIDANCE OF

Ms. Keerthana B C
Associate Professor Gd. II

Department of Computer Science and Engineering

In partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering in Computer Science & Engineering

From

Visvesvaraya Technological University, Belagavi

NITTE EDUCATION TRUST

N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India

(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
:08258 - 281039 – 281263, Fax: 08258 – 281265

B.E.CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

June 2021

NITTE EDUCATION TRUST | **N.M.A.M. INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)

Nitte – 574 110, Karnataka, India

(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC

☎: 08258 - 281039 – 281263, Fax: 08258 – 281265

Department of Computer Science and Engineering

B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

CERTIFICATE

"Heart Sound Classification" *is a bonafide work carried out by* ***Disha Ganesh Bhat (4NM17CS059), Glenita Shareen Furtado (4NM17CS066), Joshni Princia Saldanha (4NM17CS072)*** and ***Mallika (4NM17CS098)*** *in partial fulfilment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering prescribed by Visvesvaraya Technological University, Belagavi during the year 2020-2021.*

It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The mini project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

Signature of Guide **Signature of HOD**

ACKNOWLEDGEMENT

We believe that our project will be complete only after we thank the people who have contributed to make this project successful.

First and foremost, our sincere thanks to our beloved principal, **Dr. Niranjan N. Chiplunkar** for giving us an opportunity to carry out our project work at our college and providing us with all the needed facilities.

Our thanks to our beloved HOD, **Dr. Jyothi Shetty**, for extending support in carrying out this project in the department and providing us with all necessary facilities.

We express our deep sense of gratitude and indebtedness to our guide **Ms. Keerthana B C**, Associate Professor Gd. II, Department of Computer Science and Engineering, for their inspiring guidance, constant encouragement, support and suggestions for improvement during the course for our project.

We also thank all those who have supported us throughout the entire duration of our project.

Finally, we thank the staff members of the Department of Computer Science and Engineering and all our friends for their honest opinions and suggestions throughout the course of our project.

Disha Ganesh Bhat (4NM17CS059)

Glenita Shareen Furtado (4NM17CS066)

Joshni Princia Saldanha (4NM17CS072)

Mallika (4NM17CS098)

ABSTRACT

Cardiovascular diseases have become one of the most prevalent threats to human health throughout the world. As a non-invasive assistant diagnostic tool, the heart sound classification techniques play an important role in the prediction of cardiovascular diseases. An effective and automatic diagnostic method is highly attractive since it can help discover potential threat at the early stage. The main theme behind the project is to study the effectiveness of using convolutional neural networks (CNNs) and build a model to automatically detect the heart sounds and classify them into different classes. This model can predict whether a heart sound recording is normal or not. We collected a data set containing normal and abnormal heart sounds. We have performed Heart sound classification by producing a method that can classify real heart sound into one of three categories: normal, murmur and extra-heart Sound. We have given the raw audio waveforms to the CNN model. The results indicate that CNN achieved an average classification accuracy of 85%.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Heart disease is the leading cause of death globally, resulting in more people dying every year due to cardiovascular diseases (CVD’s) compared to any other cause of death. Rural mortality rates have surpassed those of urban areas as 75% of rural primary-care is handled by unqualified practitioners owing to an acute shortage of doctors.

Currently, there are two effective ways to monitor one's heart condition: electrocardiogram (ECG) and echocardiogram. However, both methods are relatively expensive for mass inspection and require technical expertise in using them. Our goal is to develop a reliable, fast and low-cost system that can be used by untrained frontline health workers or anyone with internet access, to help determine whether an individual should be referred for expert diagnosis, particularly in areas where access to clinicians and medical care is limited. This will also help in early diagnosis of CVD’s which will drastically decrease the potential risk factors of these deaths.

In this work, we take stethoscope sounds and even waveforms recorded using the microphone of a mobile phone as input and apply deep learning to the task of automated cardiac auscultation, i.e., recognizing abnormalities in heart sounds. A 1D convolutional neural network (CNN) model which directly classifies heart sound signals into normal, murmur and extra heart sound independently of ECG is proposed. High classification accuracy and F-score for heart sound classification are provided by the 1D CNN. The presented model is applicable to any heart sound signal collected by an electronic stethoscope and exhibits favorable robustness.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SYSTEM

The existing work proposed a two-dimensional convolutional neural network (CNN) model, which divides heart sound signals into two categories namely normal and abnormal. The deep features of heart sounds were extracted by using feature extraction. MFCC was given as the input feature of 1D CNN. This model obtained an accuracy of 70% on the test set.

2.2 PROPOSED SYSTEM

We proposed a 1D convolutional neural network (CNN) model that can classify real heart audio (also known as “beat classification”) into one of the three categories namely Normal, Murmur and Extra Heart Sound. We passed whole raw audio in time domain representation by allowing the machine to learn the features by its own and make the classification decision. The performance of CNN model was evaluated in terms of accuracy. Our model obtained an accuracy of 85% on the test set.

CHAPTER 3

DESIGN

3.1 HEARTBEAT CLASSIFICATION SYSTEM

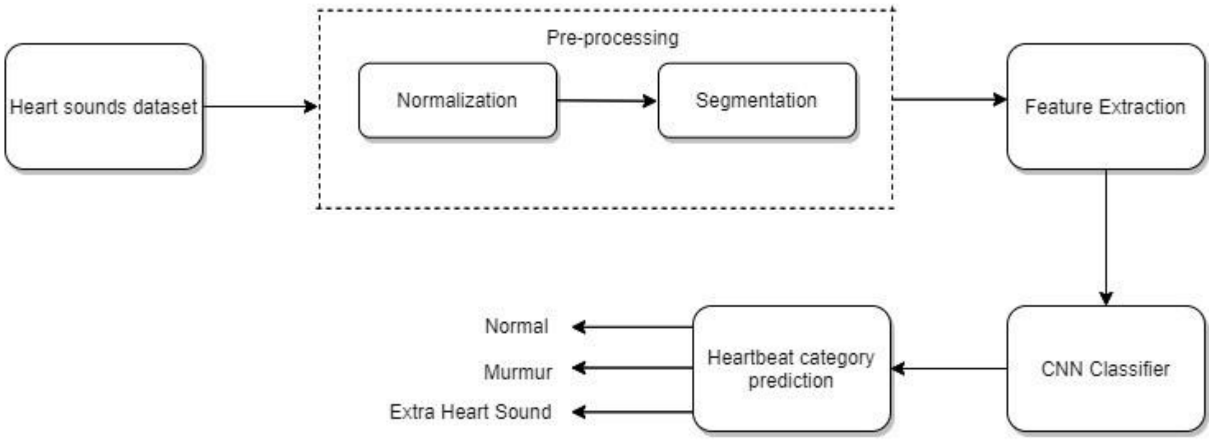


Figure 3.1: Block diagram of the proposed model

Figure 3.1 depicts the overall proposed approach for heart sound classification using CNN. The heart sound is first prepossessed and segmented into smaller chunks. Features extracted from these segments are given to the CNN for classification. The property of CNN architecture is that it automatically emphasizes and extracts the most suitable features from the input data by applying convolution operation. It keeps more concentration on the local features and its location among the other features. The classifier then makes prediction on the heartbeat sounds into three categories – Normal, Murmur and Extra Heart Sound.

CHAPTER 4

IMPLEMENTATION

4.1 Data Collection

The dataset was originally used in a Machine Learning challenge for the classification of heartbeat sounds by Mr. Peter Bentley. The dataset is divided into 2 sets depending on the sources from where it was collected. Dataset A was collected from the general public via the iStethoscope Pro iPhone app and Dataset B was collected from a clinical trial in hospitals using the digital stethoscope DigiScope.

In these datasets, there are 4 classes of heartbeat sounds:

1. Normal: Healthy heart sounds
2. Murmur: Extra sounds that occur when there is turbulence in blood flow that causes the extra vibrations that can be heard
3. Extra heart sound: Heartbeats with an additional sound
4. Extrasystoles: Additional heartbeats that occur outside the physiological heart rhythm and can cause unpleasant symptoms

Content:

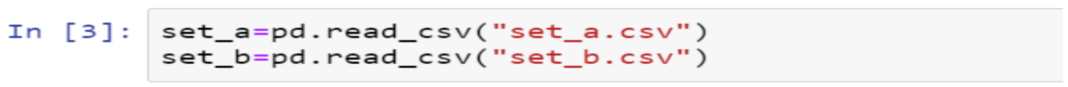
The dataset is split into two sources, A and B:

- set_a.csv - Labels and metadata for heart beats collected from the general public via an iPhone app.
- set_b.csv - Labels and metadata for heart beats collected from a clinical trial in hospitals using a digital stethoscope.
- audio files - Varying lengths, between 1 second and 30 seconds. (Some have been clipped to reduce excessive noise and provide the salient fragment of the sound)

4.2 Data Pre-processing

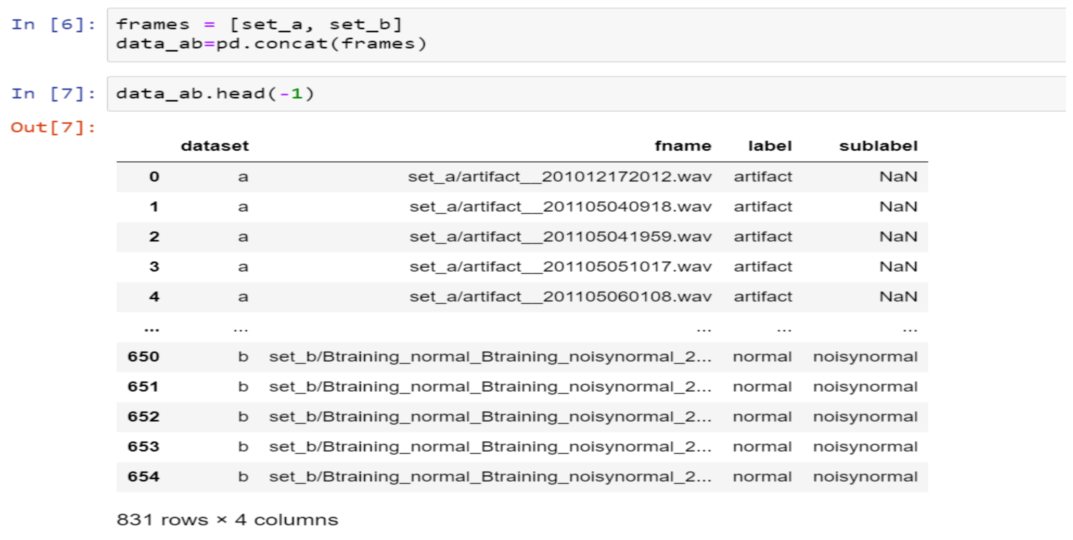
The heart sounds recorded by digital stethoscope and the mobile phone microphone often has background noise. The preprocessing of heart sounds is an essential and crucial step for automatic analysis of heartbeat recordings.

We have used csv files for cross reference check. CSV files contain the filename of the wav files and categories.



```
In [3]: set_a=pd.read_csv("set_a.csv")
        set_b=pd.read_csv("set_b.csv")
```

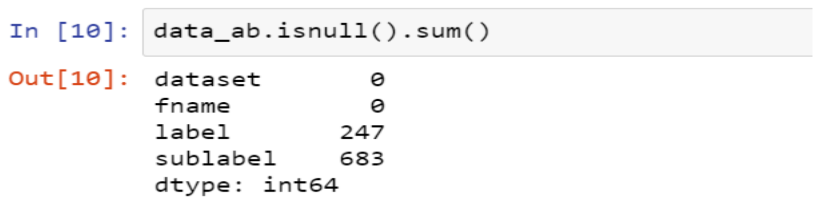
Combine the set_a and set_b csv files.



```
In [6]: frames = [set_a, set_b]
        data_ab=pd.concat(frames)

In [7]: data_ab.head(-1)
Out[7]:
```

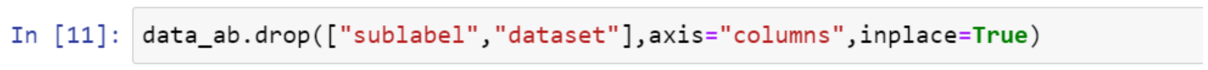
	dataset	fname	label	sublabel
0	a	set_a/artifact__201012172012.wav	artifact	NaN
1	a	set_a/artifact__201105040918.wav	artifact	NaN
2	a	set_a/artifact__201105041959.wav	artifact	NaN
3	a	set_a/artifact__201105051017.wav	artifact	NaN
4	a	set_a/artifact__201105060108.wav	artifact	NaN
...	...	...	...	...
650	b	set_b/Btraining_normal_Btraining_noisynormal_2...	normal	noisynormal
651	b	set_b/Btraining_normal_Btraining_noisynormal_2...	normal	noisynormal
652	b	set_b/Btraining_normal_Btraining_noisynormal_2...	normal	noisynormal
653	b	set_b/Btraining_normal_Btraining_noisynormal_2...	normal	noisynormal
654	b	set_b/Btraining_normal_Btraining_noisynormal_2...	normal	noisynormal

831 rows × 4 columns



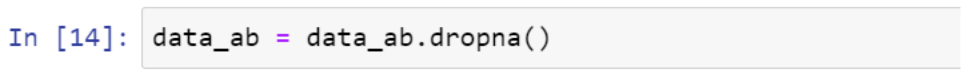
```
In [10]: data_ab.isnull().sum()

Out[10]: dataset       0
         fname         0
         label       247
         sublabel    683
         dtype: int64
```

We will drop sublabel column as it contains many nan values. In addition, we'll also drop dataset column as it will not be used.



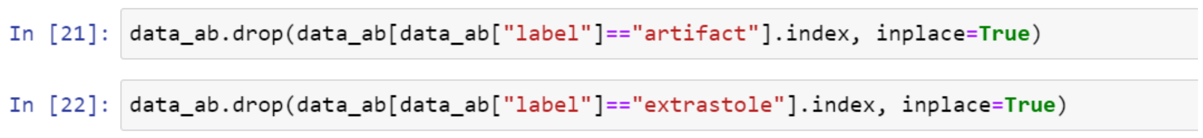
```
In [11]: data_ab.drop(["sublabel","dataset"],axis="columns",inplace=True)
```

We will also drop the unlabeled category in label column.



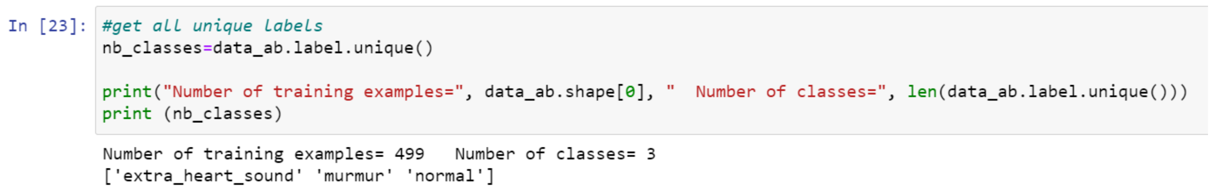
```
In [14]: data_ab = data_ab.dropna()
```

We will drop artifact and extrasystole categories from label column.



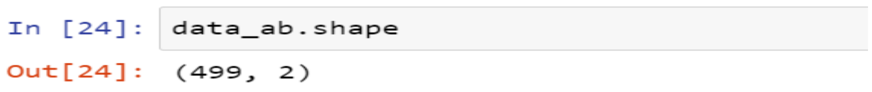
```
In [21]: data_ab.drop(data_ab[data_ab["label"]=="artifact"].index, inplace=True)

In [22]: data_ab.drop(data_ab[data_ab["label"]=="extrastole"].index, inplace=True)
```

We will take three categories of heartbeat sounds – Extra heart sound, Murmur and normal.



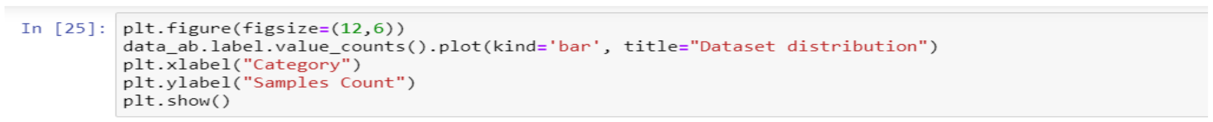
```
In [23]: #get all unique labels
         nb_classes=data_ab.label.unique()

         print("Number of training examples=", data_ab.shape[0], "  Number of classes=", len(data_ab.label.unique()))
         print (nb_classes)

         Number of training examples= 499   Number of classes= 3
         ['extra_heart_sound' 'murmur' 'normal']
```

There are 499 rows and 2 columns namely fname and label in our dataset.

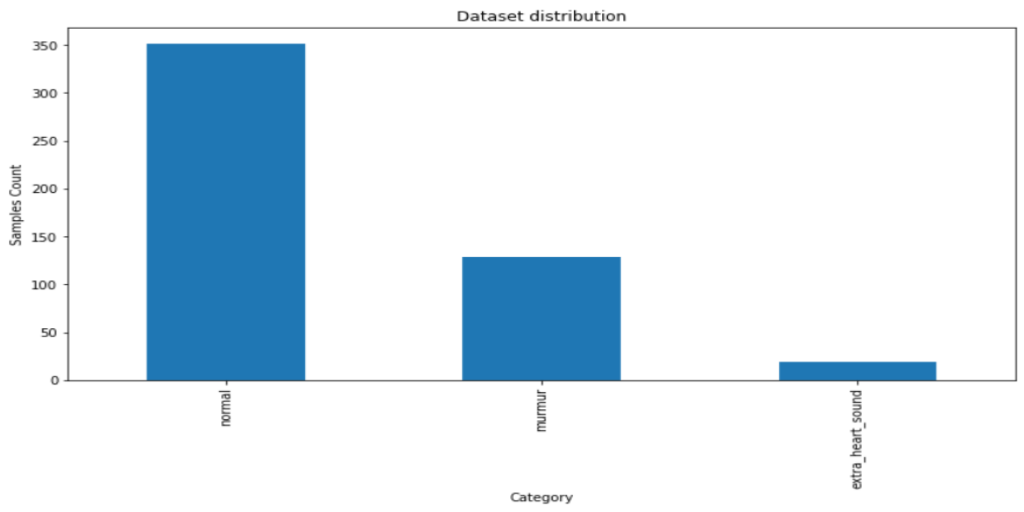


```
In [24]: data_ab.shape
Out[24]: (499, 2)
```

Distribution of categories:



```
In [25]: plt.figure(figsize=(12,6))
         data_ab.label.value_counts().plot(kind='bar', title="Dataset distribution")
         plt.xlabel("Category")
         plt.ylabel("Samples Count")
         plt.show()
```



Let's look at each category,

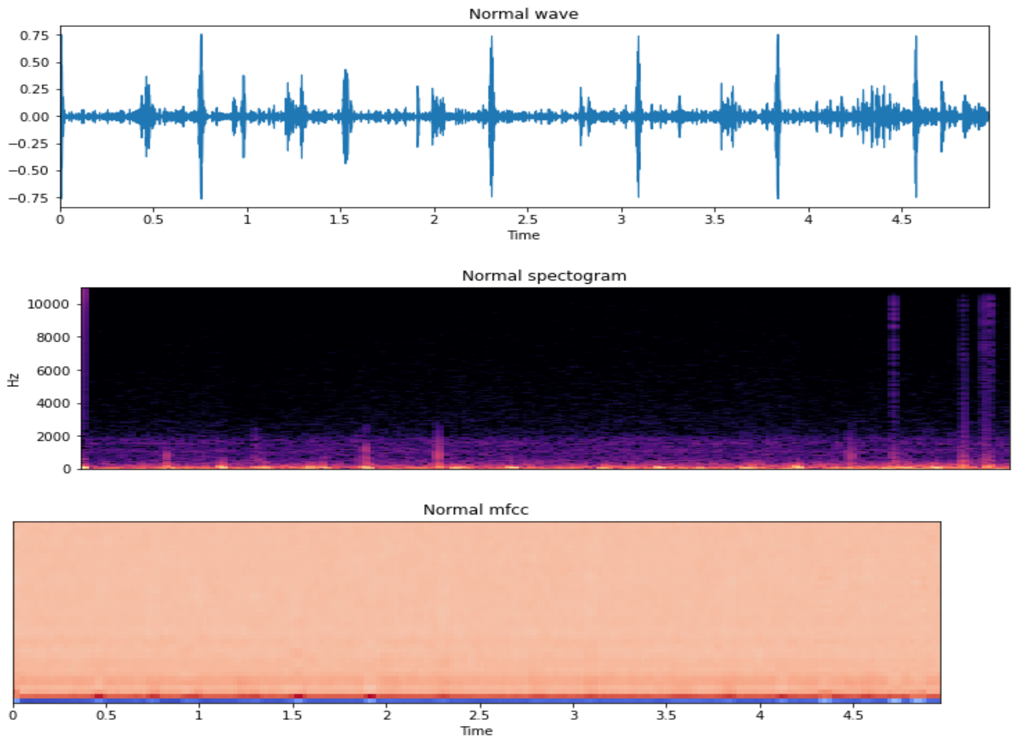
1) Normal

   In the Normal category there are normal, healthy heart sounds. These may contain noise in the final second of the recording as the device is removed from the body. They may contain a variety of background noises (from traffic to radios). They may also contain occasional random noise corresponding to breathing, or brushing the microphone against clothing or skin. A normal heart sound has a clear "lub dub, lub dub" pattern, with the time from "lub" to "dub" shorter than the time from "dub" to the next "lub" (when the heart rate is less than 140 beats per minute).

```
In [27]: normal_file="set_a/normal__201106111136.wav"
```

```
In [28]: # Load using Librosa
         y, sr = librosa.load(normal_file, duration=5)   #default sampling rate is 22 HZ
         dur=librosa.get_duration(y)
         print ("duration:", dur)
         print(y.shape, sr)

         duration: 4.963809523809524
         (109452,) 22050
```

```
In [30]: # Librosa plot
         import librosa.display

         plt.figure(figsize=(12, 3))
         librosa.display.waveplot(y, sr=sr)
         plt.title("Normal wave")

         plt.figure(figsize=(12, 3))
         D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
         librosa.display.specshow(D, y_axis='linear')
         plt.title("Normal spectogram")

         plt.figure(figsize=(12, 3))
         mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
         librosa.display.specshow(mfccs, x_axis='time')
         plt.title("Normal mfcc")

Out[30]: Text(0.5, 1.0, 'Normal mfcc')
```

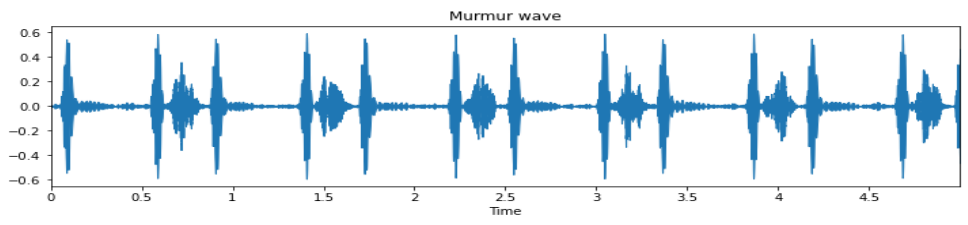


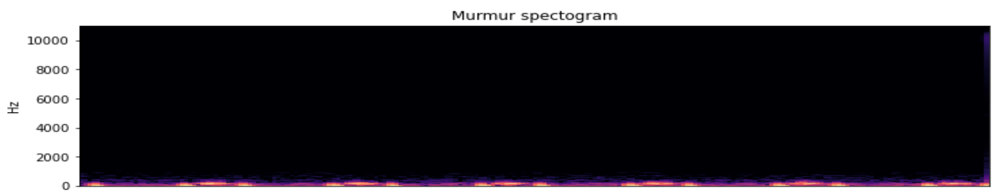
We see that there are cycles of heartbeat, with a higher intensity sound followed by a lower intensity sound.

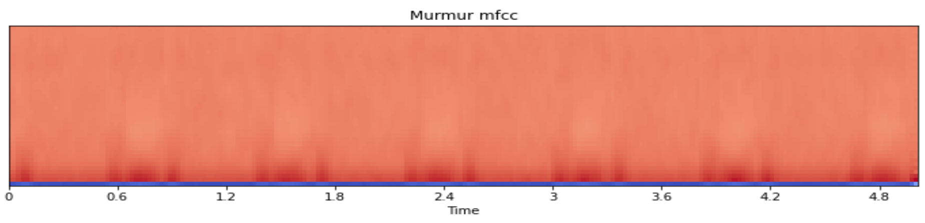
2) Murmur

Heart murmurs sound as though there is a “whooshing, roaring, rumbling, or turbulent fluid” noise in one of two temporal locations: (1) between “lub” and “dub”, or (2) between “dub” and “lub”. They can be a symptom of many heart disorders, some serious. There will still be a “lub” and a “dub”. One of the things that confuses non-medically trained people is that murmurs happen between lub and dub or between dub and lub; not on lub and not on dub.

```
In [31]: # murmur case
         murmur_file="set_a/murmur__201108222231.wav"

In [32]: y2, sr2 = librosa.load(murmur_file,duration=5)
         dur=librosa.get_duration(y2)
         print ("duration:", dur)
         print(y2.shape,sr2)

         duration: 5.0
         (110250,) 22050

In [34]: # librosa plot
         import librosa.display

         plt.figure(figsize=(12, 3))
         librosa.display.waveplot(y2, sr=sr2)
         plt.title("Murmur wave")

         plt.figure(figsize=(12, 3))
         D = librosa.amplitude_to_db(np.abs(librosa.stft(y2)), ref=np.max)
         librosa.display.specshow(D, y_axis='linear')
         plt.title("Murmur spectogram")

         plt.figure(figsize=(12, 3))
         mfccs = librosa.feature.mfcc(y=y2, sr=sr2, n_mfcc=40)
         librosa.display.specshow(mfccs, x_axis='time')
         plt.title("Murmur mfcc")

Out[34]: Text(0.5, 1.0, 'Murmur mfcc')
```







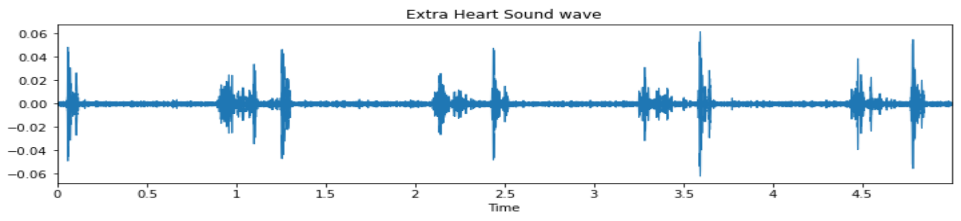
3) Extra Heart Sound

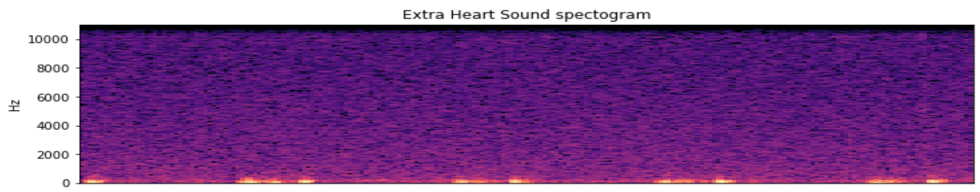
Extra heart sounds can be identified because there is an additional sound, e.g., a “lub-lub dub” or a “lub dub-dub”. An extra heart sound may not be a sign of disease. However, in some situations it is an important sign of disease, which if detected early could help a person. The extra heart sound is important to be able to detect as it cannot be detected by ultrasound very well.

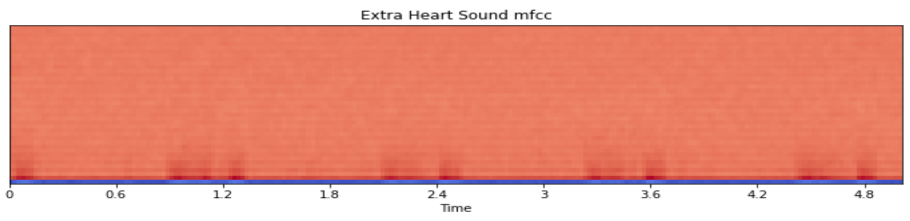
```
In [35]: # sample file
         extra_heart_sound_file="set_a/extrahls__201101070953.wav"
         y3, sr3 = librosa.load(extra_heart_sound_file, duration=5)
         dur=librosa.get_duration(y3)
         print ("duration:", dur)
         print(y3.shape,sr3)

         duration: 5.0
         (110250,) 22050
```

```
In [37]: # librosa plot
         import librosa.display

         plt.figure(figsize=(12, 3))
         librosa.display.waveplot(y3, sr=sr3)
         plt.title("Extra Heart Sound wave")

         plt.figure(figsize=(12, 3))
         D = librosa.amplitude_to_db(np.abs(librosa.stft(y3)), ref=np.max)
         librosa.display.specshow(D, y_axis='linear')
         plt.title("Extra Heart Sound spectogram")

         plt.figure(figsize=(12, 3))
         mfccs = librosa.feature.mfcc(y=y3, sr=sr3, n_mfcc=40)
         librosa.display.specshow(mfccs, x_axis='time')
         plt.title("Extra Heart Sound mfcc")
```

```
Out[37]: Text(0.5, 1.0, 'Extra Heart Sound mfcc')
```

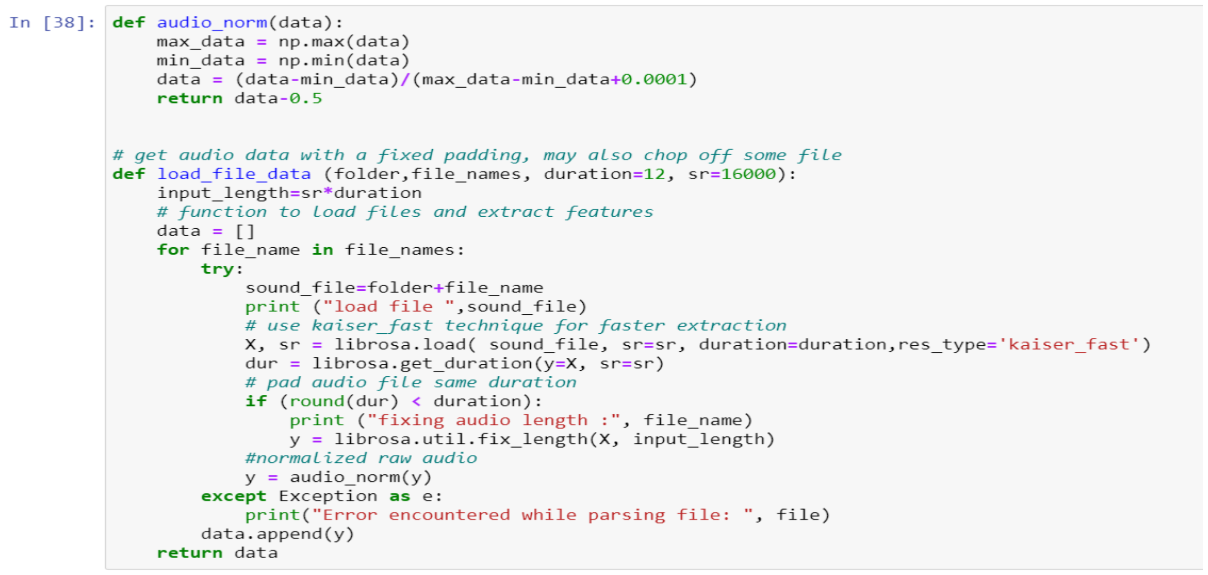






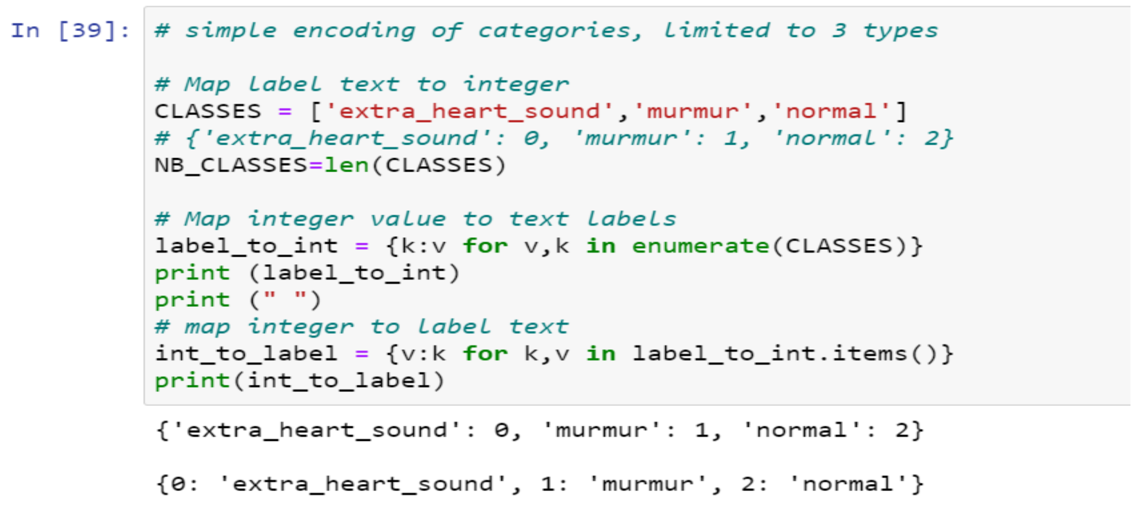
When we create this data, a bit of pre-processing is required. First is to make all the extracted samples of same shape, second is to normalize the data and third is to create appropriate X and Y for our deep learning model.

The lengths of the audio files in the dataset varies from 1 to 30 seconds long. For training purpose, we use first 12 seconds of the audio and pad missing length for file smaller than 12 seconds. We pad the audio files because some of the audio files have duration less than 5 seconds in which they do not contain enough data points to accurately classify heartbeats. The recordings have to be converted to some fixed length prior to training. We slice the heart sounds into fixed-length segments of length 12 sec. We are cutting about half a second from the start and end of all the audio files because the noise is due to the contact of the microphone with the body.



```
In [38]: def audio_norm(data):
             max_data = np.max(data)
             min_data = np.min(data)
             data = (data-min_data)/(max_data-min_data+0.0001)
             return data-0.5


         # get audio data with a fixed padding, may also chop off some file
         def load_file_data (folder,file_names, duration=12, sr=16000):
             input_length=sr*duration
             # function to load files and extract features
             data = []
             for file_name in file_names:
                 try:
                     sound_file=folder+file_name
                     print ("load file ",sound_file)
                     # use kaiser_fast technique for faster extraction
                     X, sr = librosa.load( sound_file, sr=sr, duration=duration,res_type='kaiser_fast')
                     dur = librosa.get_duration(y=X, sr=sr)
                     # pad audio file same duration
                     if (round(dur) < duration):
                         print ("fixing audio length :", file_name)
                         y = librosa.util.fix_length(X, input_length)
                     #normalized raw audio
                     y = audio_norm(y)
                 except Exception as e:
                     print("Error encountered while parsing file: ", file)
                 data.append(y)
             return data
```

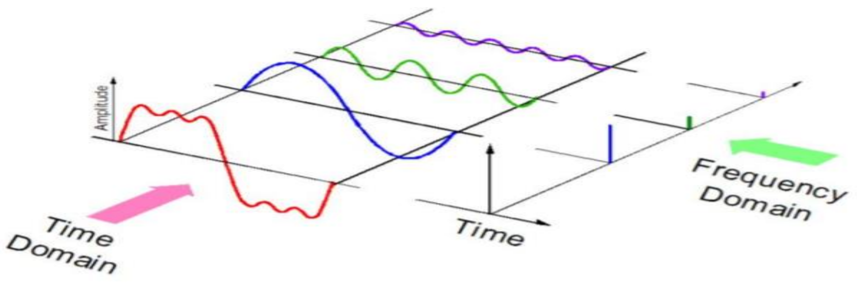
We will map three labels which are in text format to the integer values.



```
In [39]: # simple encoding of categories, limited to 3 types

         # Map label text to integer
         CLASSES = ['extra_heart_sound','murmur','normal']
         # {'extra_heart_sound': 0, 'murmur': 1, 'normal': 2}
         NB_CLASSES=len(CLASSES)

         # Map integer value to text labels
         label_to_int = {k:v for v,k in enumerate(CLASSES)}
         print (label_to_int)
         print (" ")
         # map integer to label text
         int_to_label = {v:k for k,v in label_to_int.items()}
         print(int_to_label)

         {'extra_heart_sound': 0, 'murmur': 1, 'normal': 2}

         {0: 'extra_heart_sound', 1: 'murmur', 2: 'normal'}
```

4.3 Feature Extraction

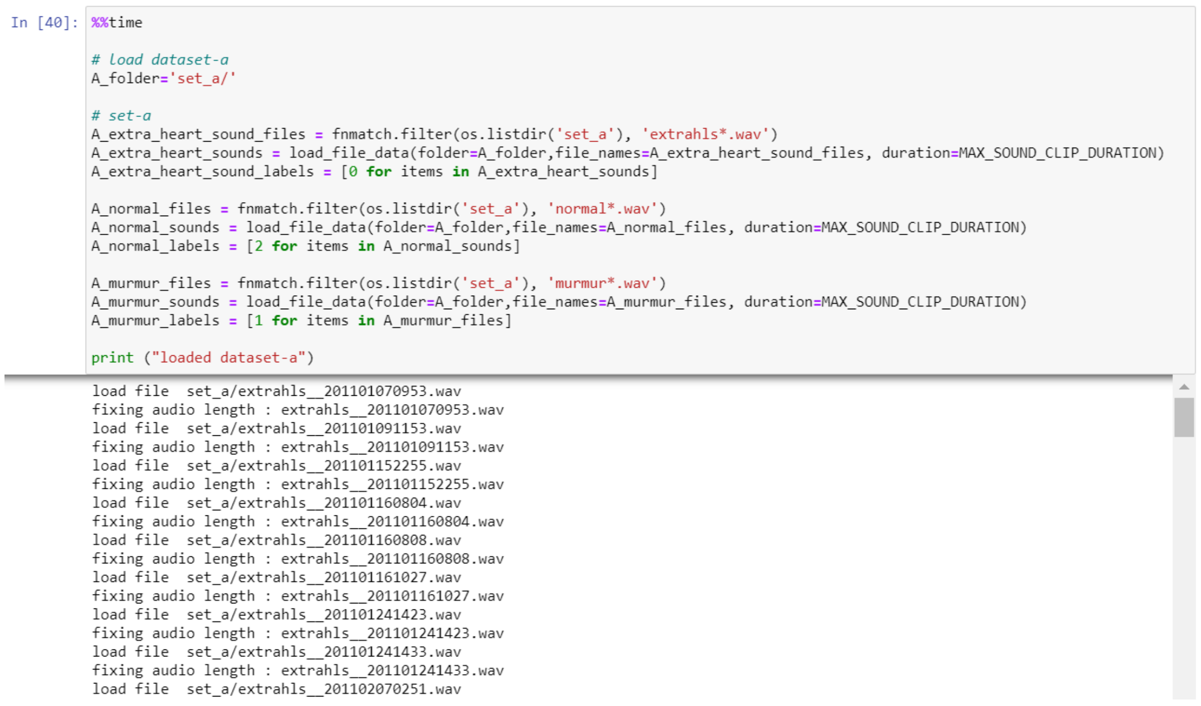
The raw audio data used is .wav type (Waveform Audio File Format) is in the amplitude vs time form in the time domain. The data was collected using different instruments (i.e., digital stethoscopes and mobile phone microphone) which results in varying amplitude ranges. We didn't use any feature extraction methods instead we passed the raw audio in which is in time domain representation.



Loading Data:

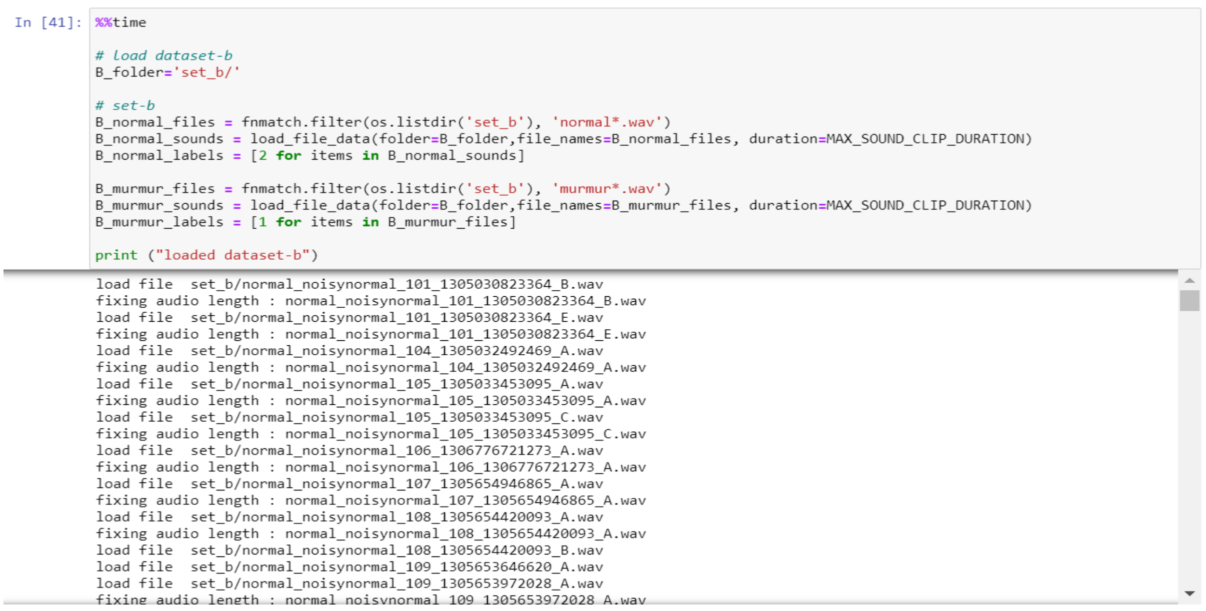
Loading of the audio data file will be based on content from directory since each filename is associated with the category type. Hence, we can use csv file for cross reference check.

We have to create training data by going through all the raw files, and extracting a part of the audio along with its respective label.



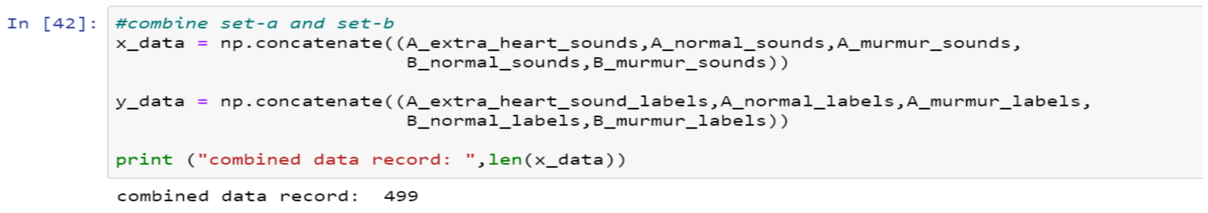
```
In [40]: %%time

         # load dataset-a
         A_folder='set_a/'

         # set-a
         A_extra_heart_sound_files = fnmatch.filter(os.listdir('set_a'), 'extrahls*.wav')
         A_extra_heart_sounds = load_file_data(folder=A_folder,file_names=A_extra_heart_sound_files, duration=MAX_SOUND_CLIP_DURATION)
         A_extra_heart_sound_labels = [0 for items in A_extra_heart_sounds]

         A_normal_files = fnmatch.filter(os.listdir('set_a'), 'normal*.wav')
         A_normal_sounds = load_file_data(folder=A_folder,file_names=A_normal_files, duration=MAX_SOUND_CLIP_DURATION)
         A_normal_labels = [2 for items in A_normal_sounds]

         A_murmur_files = fnmatch.filter(os.listdir('set_a'), 'murmur*.wav')
         A_murmur_sounds = load_file_data(folder=A_folder,file_names=A_murmur_files, duration=MAX_SOUND_CLIP_DURATION)
         A_murmur_labels = [1 for items in A_murmur_files]

         print ("loaded dataset-a")
```

```
load file  set_a/extrahls__201101070953.wav
fixing audio length : extrahls__201101070953.wav
load file  set_a/extrahls__201101091153.wav
fixing audio length : extrahls__201101091153.wav
load file  set_a/extrahls__201101152255.wav
fixing audio length : extrahls__201101152255.wav
load file  set_a/extrahls__201101160804.wav
fixing audio length : extrahls__201101160804.wav
load file  set_a/extrahls__201101160808.wav
fixing audio length : extrahls__201101160808.wav
load file  set_a/extrahls__201101161027.wav
fixing audio length : extrahls__201101161027.wav
load file  set_a/extrahls__201101241423.wav
fixing audio length : extrahls__201101241423.wav
load file  set_a/extrahls__201101241433.wav
fixing audio length : extrahls__201101241433.wav
load file  set_a/extrahls__201102070251.wav
```



```
In [41]: %%time

         # load dataset-b
         B_folder='set_b/'

         # set-b
         B_normal_files = fnmatch.filter(os.listdir('set_b'), 'normal*.wav')
         B_normal_sounds = load_file_data(folder=B_folder,file_names=B_normal_files, duration=MAX_SOUND_CLIP_DURATION)
         B_normal_labels = [2 for items in B_normal_sounds]

         B_murmur_files = fnmatch.filter(os.listdir('set_b'), 'murmur*.wav')
         B_murmur_sounds = load_file_data(folder=B_folder,file_names=B_murmur_files, duration=MAX_SOUND_CLIP_DURATION)
         B_murmur_labels = [1 for items in B_murmur_files]

         print ("loaded dataset-b")
```

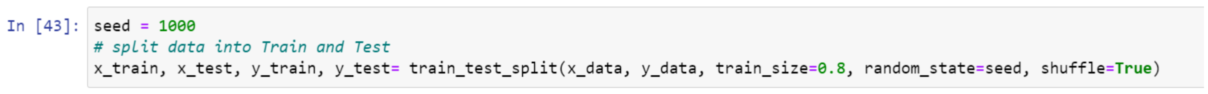
```
load file  set_b/normal_noisynormal_101_1305030823364_B.wav
fixing audio length : normal_noisynormal_101_1305030823364_B.wav
load file  set_b/normal_noisynormal_101_1305030823364_E.wav
fixing audio length : normal_noisynormal_101_1305030823364_E.wav
load file  set_b/normal_noisynormal_104_1305032492469_A.wav
fixing audio length : normal_noisynormal_104_1305032492469_A.wav
load file  set_b/normal_noisynormal_105_1305033453095_A.wav
fixing audio length : normal_noisynormal_105_1305033453095_A.wav
load file  set_b/normal_noisynormal_105_1305033453095_C.wav
fixing audio length : normal_noisynormal_105_1305033453095_C.wav
load file  set_b/normal_noisynormal_106_1306776721273_A.wav
fixing audio length : normal_noisynormal_106_1306776721273_A.wav
load file  set_b/normal_noisynormal_107_1305654946865_A.wav
fixing audio length : normal_noisynormal_107_1305654946865_A.wav
load file  set_b/normal_noisynormal_108_1305654420093_A.wav
fixing audio length : normal_noisynormal_108_1305654420093_A.wav
load file  set_b/normal_noisynormal_108_1305654420093_B.wav
load file  set_b/normal_noisynormal_109_1305653646620_A.wav
load file  set_b/normal_noisynormal_109_1305653972028_A.wav
fixing audio length : normal noisynormal 109 1305653972028 A.wav
```



```
In [42]: #combine set-a and set-b
         x_data = np.concatenate((A_extra_heart_sounds,A_normal_sounds,A_murmur_sounds,
                                  B_normal_sounds,B_murmur_sounds))

         y_data = np.concatenate((A_extra_heart_sound_labels,A_normal_labels,A_murmur_labels,
                                  B_normal_labels,B_murmur_labels))

         print ("combined data record: ",len(x_data))

         combined data record:  499
```

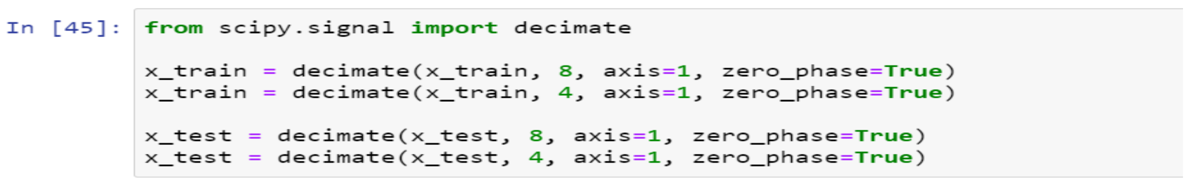
Train-Test Split:

We split our dataset into 80% training data and 20% testing data.

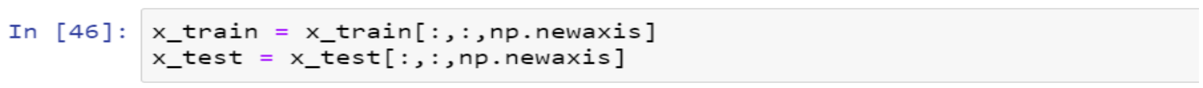


```
In [43]: seed = 1000
         # split data into Train and Test
         x_train, x_test, y_train, y_test= train_test_split(x_data, y_data, train_size=0.8, random_state=seed, shuffle=True)
```

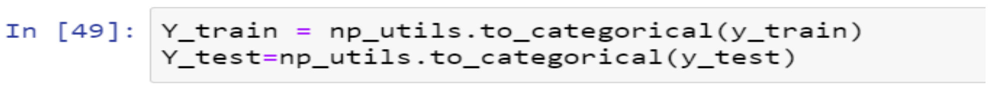
We now downsample the data with what is in effect a very aggressive low pass filter. This is not needed for computational time, but it seems to improve generalization on this dataset. With more data, we should remove or reduce this step and instead add 2-5 extra convolution layers. The reason this works is probably that what you hear in the stethoscope is almost exclusively low frequency sounds, especially murmurs.



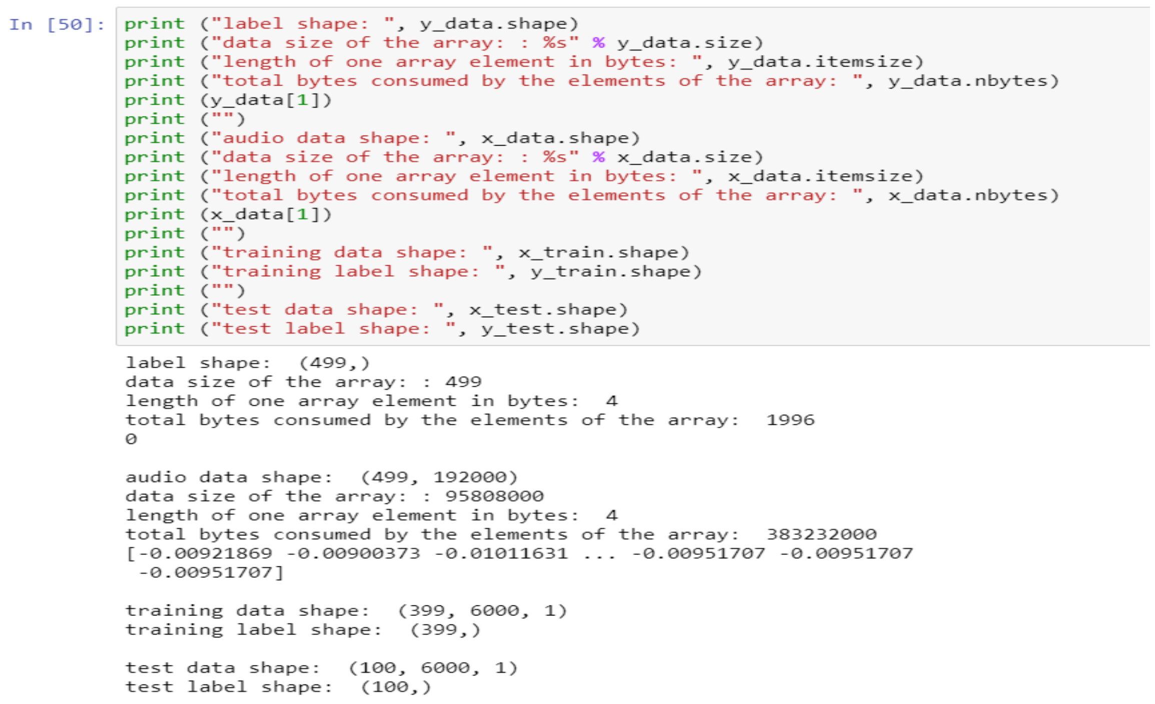
```
In [45]: from scipy.signal import decimate

         x_train = decimate(x_train, 8, axis=1, zero_phase=True)
         x_train = decimate(x_train, 4, axis=1, zero_phase=True)

         x_test = decimate(x_test, 8, axis=1, zero_phase=True)
         x_test = decimate(x_test, 4, axis=1, zero_phase=True)
```



```
In [46]: x_train = x_train[:,:,np.newaxis]
         x_test = x_test[:,:,np.newaxis]
```



```
In [49]: Y_train = np_utils.to_categorical(y_train)
         Y_test=np_utils.to_categorical(y_test)
```



```
In [50]: print ("label shape: ", y_data.shape)
         print ("data size of the array: : %s" % y_data.size)
         print ("length of one array element in bytes: ", y_data.itemsize)
         print ("total bytes consumed by the elements of the array: ", y_data.nbytes)
         print (y_data[1])
         print ("")
         print ("audio data shape: ", x_data.shape)
         print ("data size of the array: : %s" % x_data.size)
         print ("length of one array element in bytes: ", x_data.itemsize)
         print ("total bytes consumed by the elements of the array: ", x_data.nbytes)
         print (x_data[1])
         print ("")
         print ("training data shape: ", x_train.shape)
         print ("training label shape: ", y_train.shape)
         print ("")
         print ("test data shape: ", x_test.shape)
         print ("test label shape: ", y_test.shape)
```

```
label shape:  (499,)
data size of the array: : 499
length of one array element in bytes:  4
total bytes consumed by the elements of the array:  1996
0

audio data shape:  (499, 192000)
data size of the array: : 95808000
length of one array element in bytes:  4
total bytes consumed by the elements of the array:  383232000
[-0.00921869 -0.00900373 -0.01011631 ... -0.00951707 -0.00951707
 -0.00951707]

training data shape:  (399, 6000, 1)
training label shape:  (399,)

test data shape:  (100, 6000, 1)
test label shape:  (100,)
```

4.4 Model Building

We will build the one-dimensional Convolutional Neural Network model (CNN).

The Convolutional Neural Network (CNN) is one of the neural network architectures specifically used for image classification. Just like other neural network methods, CNN is also inspired by human brain tissue. Convolution neural network is mainly composed of two parts, feature extraction, and classification.

The network architecture of a convolutional neural network that accepts as input a single channel and outputs the classification, predicting whether the input segment represents a normal, murmur or extra heart sound.

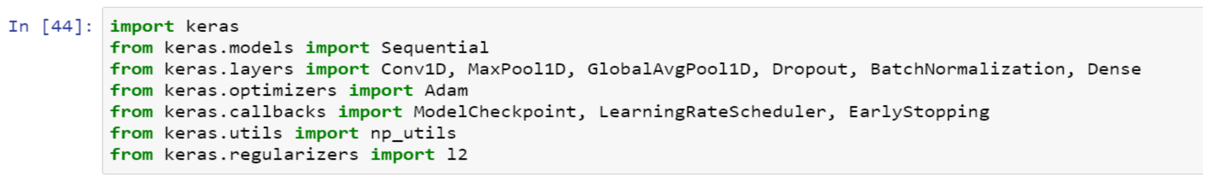
Convolutional Neural Network in this study uses 6 convolution layers, 4 max-pooling layers, 3 dropout layers, 6 batch normalization layers, 1 global average pooling layer and finally a dense layer. Due to the class imbalance, Dropout and kernel regularizers are employed selectively to prevent overfitting. A GAP layer was introduced into the designed CNN in order to obtain global information about the feature maps to improve the classification performance. This was also shown to reduce the model’s complexity and avoid overfitting.

The activation function in convolution layers uses Rectifier Linear Unit (ReLU) algorithm. The ReLU algorithm has advantages in time efficiency for training and testing.

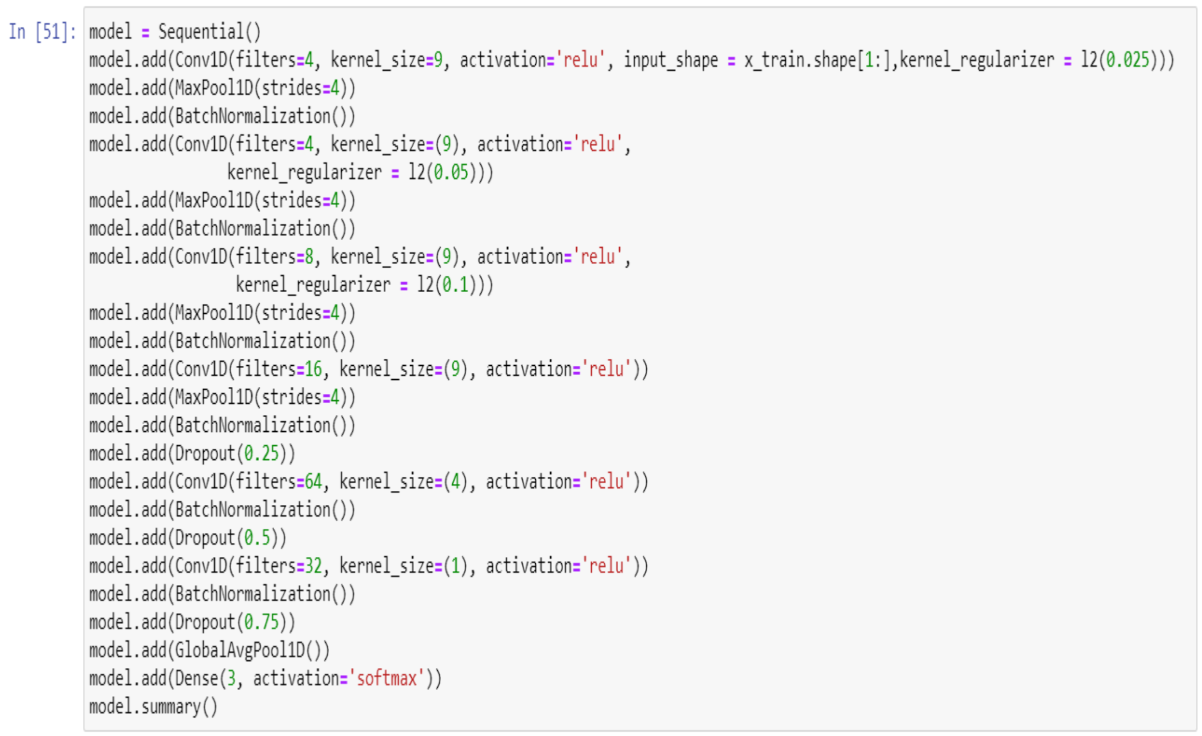
The dropout layer: The term "dropout" refers to dropping out units (both hidden and visible) in a neural network. It is a very efficient way of performing model averaging with neural networks. Model averaging is a natural response to model uncertainty. The dropout layer allows for regularization by randomly setting some neurons in previous layers to zero during training.

Max Pooling: The objective of Max pooling is to down-sample an input representation. It helps in reducing the dimensionality and alleviate feature extraction. It reduces the computational cost-reducing the number of parameters to learned.

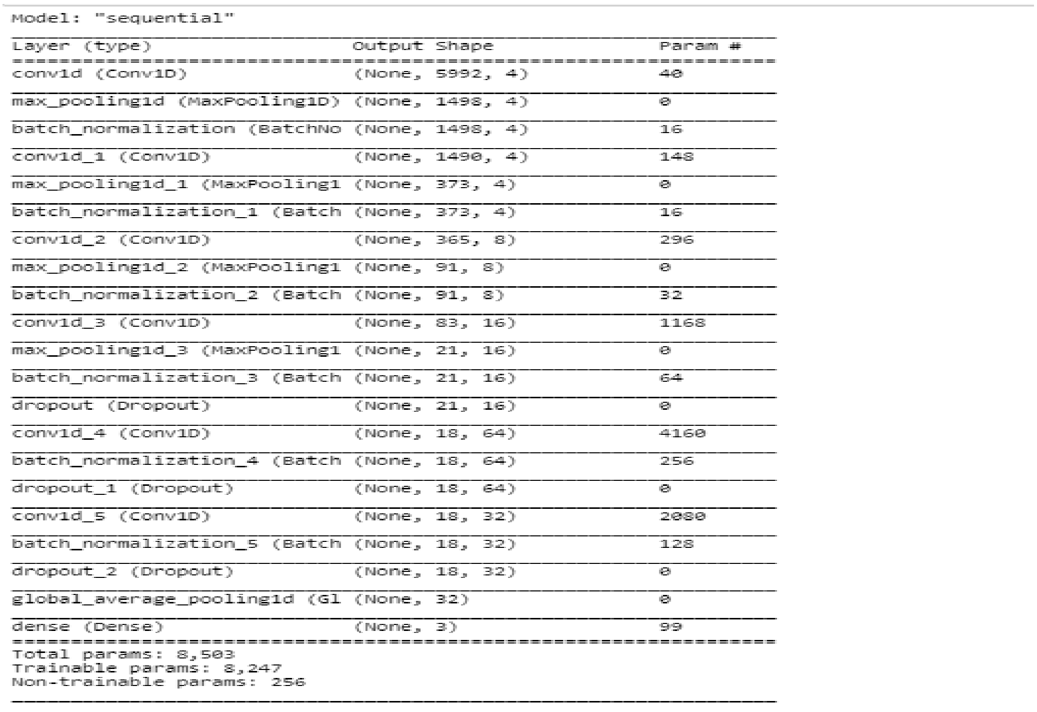
Dense layer: Here every input is connected to every output by weight. And we are using softmax as the non-linear activation function after this layer.



```
In [44]: import keras
         from keras.models import Sequential
         from keras.layers import Conv1D, MaxPool1D, GlobalAvgPool1D, Dropout, BatchNormalization, Dense
         from keras.optimizers import Adam
         from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping
         from keras.utils import np_utils
         from keras.regularizers import l2
```



```
In [51]: model = Sequential()
         model.add(Conv1D(filters=4, kernel_size=9, activation='relu', input_shape = x_train.shape[1:],kernel_regularizer = l2(0.025)))
         model.add(MaxPool1D(strides=4))
         model.add(BatchNormalization())
         model.add(Conv1D(filters=4, kernel_size=(9), activation='relu',
                          kernel_regularizer = l2(0.05)))
         model.add(MaxPool1D(strides=4))
         model.add(BatchNormalization())
         model.add(Conv1D(filters=8, kernel_size=(9), activation='relu',
                           kernel_regularizer = l2(0.1)))
         model.add(MaxPool1D(strides=4))
         model.add(BatchNormalization())
         model.add(Conv1D(filters=16, kernel_size=(9), activation='relu'))
         model.add(MaxPool1D(strides=4))
         model.add(BatchNormalization())
         model.add(Dropout(0.25))
         model.add(Conv1D(filters=64, kernel_size=(4), activation='relu'))
         model.add(BatchNormalization())
         model.add(Dropout(0.5))
         model.add(Conv1D(filters=32, kernel_size=(1), activation='relu'))
         model.add(BatchNormalization())
         model.add(Dropout(0.75))
         model.add(GlobalAvgPool1D())
         model.add(Dense(3, activation='softmax'))
         model.summary()
```

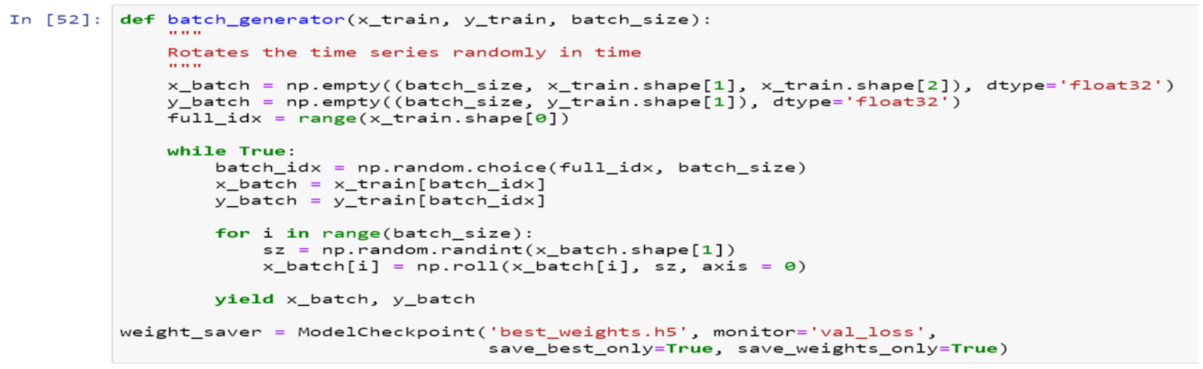


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 5992, 4)           40
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 1498, 4)           0
_________________________________________________________________
batch_normalization (BatchNo (None, 1498, 4)           16
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1490, 4)           148
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 373, 4)            0
_________________________________________________________________
batch_normalization_1 (Batch (None, 373, 4)            16
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 365, 8)            296
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 91, 8)             0
_________________________________________________________________
batch_normalization_2 (Batch (None, 91, 8)             32
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 83, 16)            1168
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 21, 16)            0
_________________________________________________________________
batch_normalization_3 (Batch (None, 21, 16)            64
_________________________________________________________________
dropout (Dropout)            (None, 21, 16)            0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 18, 64)            4160
_________________________________________________________________
batch_normalization_4 (Batch (None, 18, 64)            256
_________________________________________________________________
dropout_1 (Dropout)          (None, 18, 64)            0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 18, 32)            2080
_________________________________________________________________
batch_normalization_5 (Batch (None, 18, 32)            128
_________________________________________________________________
dropout_2 (Dropout)          (None, 18, 32)            0
_________________________________________________________________
global_average_pooling1d (Gl (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 3)                 99
=================================================================
Total params: 8,503
Trainable params: 8,247
Non-trainable params: 256
_________________________________________________________________
```

4.5 MODEL EVALUATION

We have used accuracy as our evaluation metric that calculates the percentage of heartbeat sound categories that were correctly predicted.

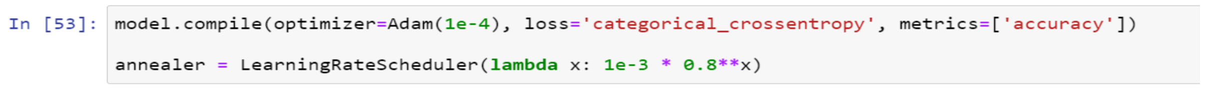
Class Weights:



```
In [52]: def batch_generator(x_train, y_train, batch_size):
             """
             Rotates the time series randomly in time
             """
             x_batch = np.empty((batch_size, x_train.shape[1], x_train.shape[2]), dtype='float32')
             y_batch = np.empty((batch_size, y_train.shape[1]), dtype='float32')
             full_idx = range(x_train.shape[0])

             while True:
                 batch_idx = np.random.choice(full_idx, batch_size)
                 x_batch = x_train[batch_idx]
                 y_batch = y_train[batch_idx]

                 for i in range(batch_size):
                     sz = np.random.randint(x_batch.shape[1])
                     x_batch[i] = np.roll(x_batch[i], sz, axis = 0)

                 yield x_batch, y_batch

         weight_saver = ModelCheckpoint('best_weights.h5', monitor='val_loss',
                                        save_best_only=True, save_weights_only=True)
```

Compiling Model:

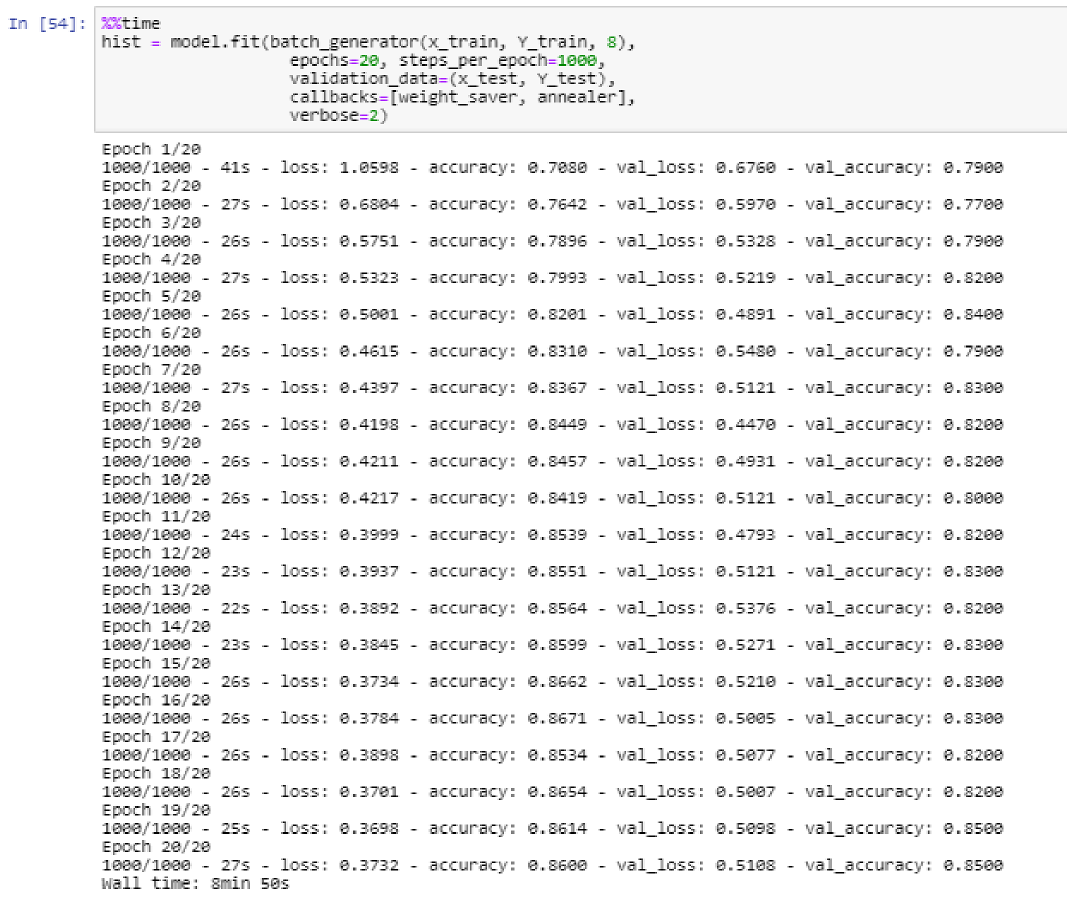
Adam method is used for the optimization process to update the weight on the Convolutional Neural Network. This method has efficient computation (memory and time), invariant to gradient scaling and suitable when applied to large data or parameters.



```
In [53]: model.compile(optimizer=Adam(1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

         annealer = LearningRateScheduler(lambda x: 1e-3 * 0.8**x)
```

Fitting Model:

We have given 20 epochs on batch size of 8.

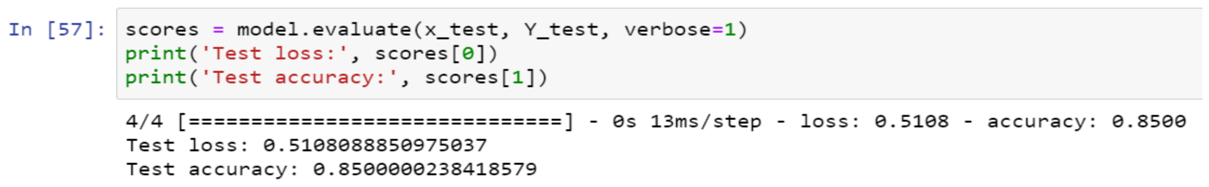


```
In [54]: %%time
         hist = model.fit(batch_generator(x_train, Y_train, 8),
                          epochs=20, steps_per_epoch=1000,
                          validation_data=(x_test, Y_test),
                          callbacks=[weight_saver, annealer],
                          verbose=2)
```

```
Epoch 1/20
1000/1000 - 41s - loss: 1.0598 - accuracy: 0.7080 - val_loss: 0.6760 - val_accuracy: 0.7900
Epoch 2/20
1000/1000 - 27s - loss: 0.6804 - accuracy: 0.7642 - val_loss: 0.5970 - val_accuracy: 0.7700
Epoch 3/20
1000/1000 - 26s - loss: 0.5751 - accuracy: 0.7896 - val_loss: 0.5328 - val_accuracy: 0.7900
Epoch 4/20
1000/1000 - 27s - loss: 0.5323 - accuracy: 0.7993 - val_loss: 0.5219 - val_accuracy: 0.8200
Epoch 5/20
1000/1000 - 26s - loss: 0.5001 - accuracy: 0.8201 - val_loss: 0.4891 - val_accuracy: 0.8400
Epoch 6/20
1000/1000 - 26s - loss: 0.4615 - accuracy: 0.8310 - val_loss: 0.5480 - val_accuracy: 0.7900
Epoch 7/20
1000/1000 - 27s - loss: 0.4397 - accuracy: 0.8367 - val_loss: 0.5121 - val_accuracy: 0.8300
Epoch 8/20
1000/1000 - 26s - loss: 0.4198 - accuracy: 0.8449 - val_loss: 0.4470 - val_accuracy: 0.8200
Epoch 9/20
1000/1000 - 26s - loss: 0.4211 - accuracy: 0.8457 - val_loss: 0.4931 - val_accuracy: 0.8200
Epoch 10/20
1000/1000 - 26s - loss: 0.4217 - accuracy: 0.8419 - val_loss: 0.5121 - val_accuracy: 0.8000
Epoch 11/20
1000/1000 - 24s - loss: 0.3999 - accuracy: 0.8539 - val_loss: 0.4793 - val_accuracy: 0.8200
Epoch 12/20
1000/1000 - 23s - loss: 0.3937 - accuracy: 0.8551 - val_loss: 0.5121 - val_accuracy: 0.8300
Epoch 13/20
1000/1000 - 22s - loss: 0.3892 - accuracy: 0.8564 - val_loss: 0.5376 - val_accuracy: 0.8200
Epoch 14/20
1000/1000 - 23s - loss: 0.3845 - accuracy: 0.8599 - val_loss: 0.5271 - val_accuracy: 0.8300
Epoch 15/20
1000/1000 - 26s - loss: 0.3734 - accuracy: 0.8662 - val_loss: 0.5210 - val_accuracy: 0.8300
Epoch 16/20
1000/1000 - 26s - loss: 0.3784 - accuracy: 0.8671 - val_loss: 0.5005 - val_accuracy: 0.8300
Epoch 17/20
1000/1000 - 26s - loss: 0.3898 - accuracy: 0.8534 - val_loss: 0.5077 - val_accuracy: 0.8200
Epoch 18/20
1000/1000 - 26s - loss: 0.3701 - accuracy: 0.8654 - val_loss: 0.5007 - val_accuracy: 0.8200
Epoch 19/20
1000/1000 - 25s - loss: 0.3698 - accuracy: 0.8614 - val_loss: 0.5098 - val_accuracy: 0.8500
Epoch 20/20
1000/1000 - 27s - loss: 0.3732 - accuracy: 0.8600 - val_loss: 0.5108 - val_accuracy: 0.8500
Wall time: 8min 50s
```

Evaluating Model:

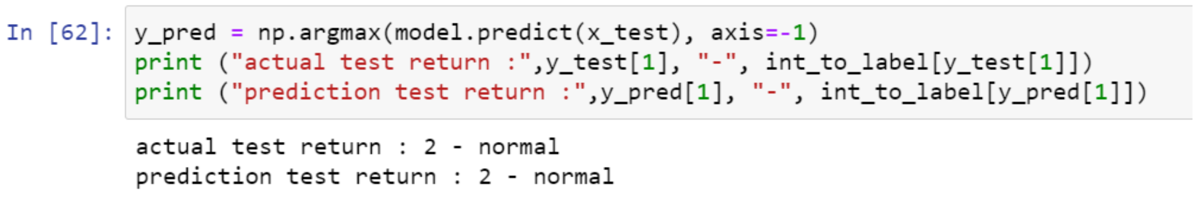
We got the test loss of 51% and test accuracy of 85%.



```
In [57]: scores = model.evaluate(x_test, Y_test, verbose=1)
         print('Test loss:', scores[0])
         print('Test accuracy:', scores[1])
```

```
4/4 [==============================] - 0s 13ms/step - loss: 0.5108 - accuracy: 0.8500
Test loss: 0.5108088850975037
Test accuracy: 0.8500000238418579
```

Classifying Heartbeat:



```
In [62]: y_pred = np.argmax(model.predict(x_test), axis=-1)
         print ("actual test return :",y_test[1], "-", int_to_label[y_test[1]])
         print ("prediction test return :",y_pred[1], "-", int_to_label[y_pred[1]])
```

```
actual test return : 2 - normal
prediction test return : 2 - normal
```

CHAPTER 5

RESULTS

5.1 Training and Validation Accuracy Curves

Figure 5.1 shows the plot of Training and Validation Accuracy. We achieved Training Accuracy of 86% and Validation Accuracy of 85%.

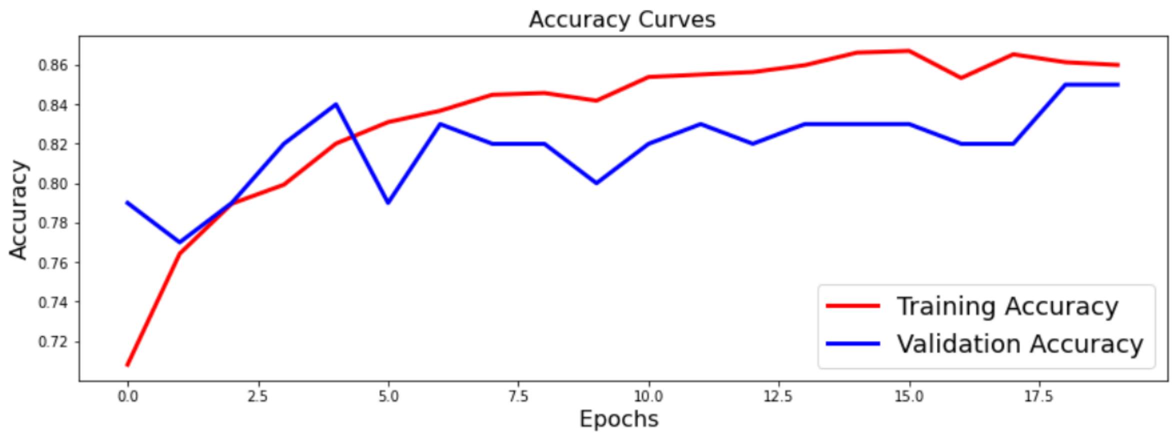


Figure 5.1 Training and Validation Accuracy Curves

5.2 Training and Validation Loss Curves

Figure 5.2 shows the plot of Training and Validation Loss. We achieved Training Loss of 37.32% and Validation Loss of 51.08%.

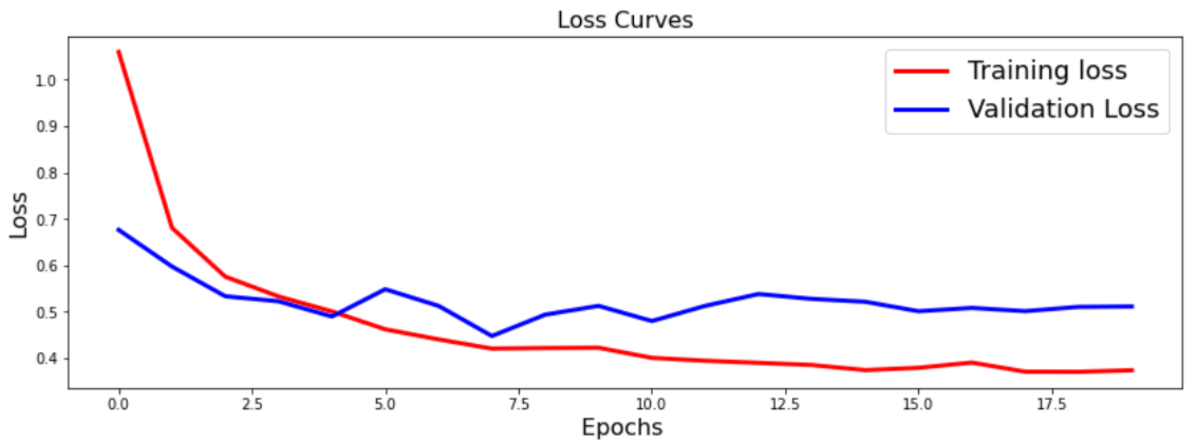
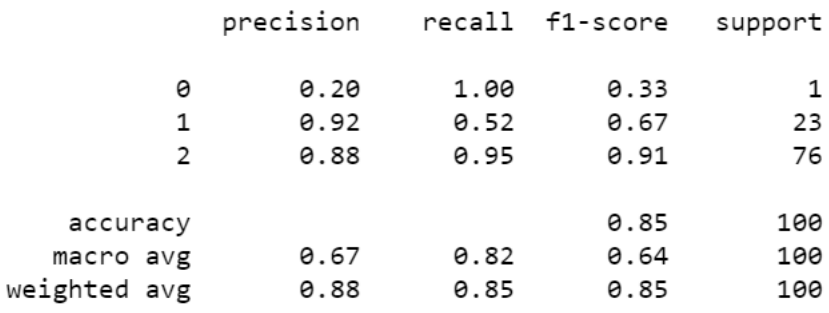


Figure 5.2 Training and Validation Loss Curves

As the graph says that both "training and validation" errors reduces as number of epochs to the training model increases. From above plot we can also infer that the number of suitable epochs is around 20 as the accuracy of test data remains constant after 20 epochs.

5.3 Classification Report



```
              precision    recall  f1-score   support

           0       0.20      1.00      0.33         1
           1       0.92      0.52      0.67        23
           2       0.88      0.95      0.91        76

    accuracy                           0.85       100
   macro avg       0.67      0.82      0.64       100
weighted avg       0.88      0.85      0.85       100
```

Figure 5.3 Classification Report

5.4 Confusion Matrix

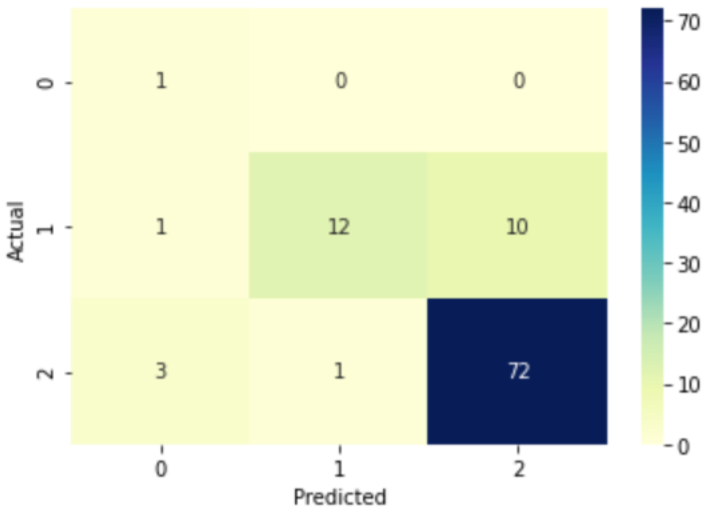


Figure 5.4 Confusion Matrix

Figure 5.4 shows the Confusion Matrix. Here 0 represents Extra Heart Sound,1 represents Murmur, 2 represents Normal. Y axis consists of Actual heartbeat categories and X axis consists of Predicted heartbeat categories. There was 1 extra heart sound in the test set but our model predicted 5 of them. There were 23 actual murmur heart sounds but our model predicted 13 correctly. There were 76 actual normal heartbeats but our model predicted 82 normal heartbeats.

5.5 Evaluation plots for each category

- Extra Heart Sound

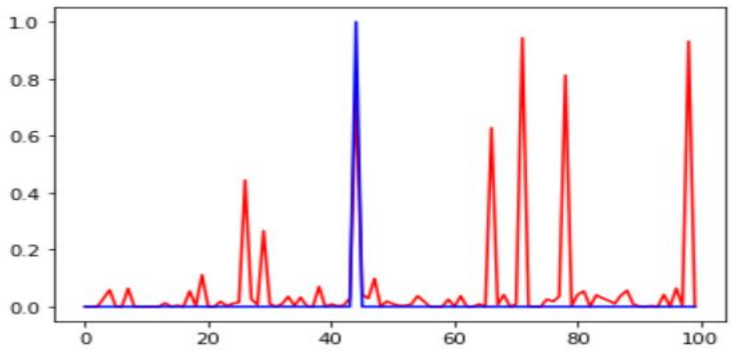


Figure 5.5.1: Evaluation Plot of Extra Heart Sound Category

- Murmur

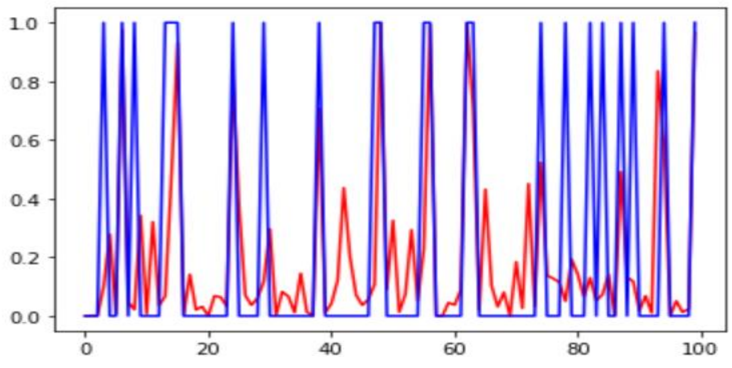


Figure 5.5.2: Evaluation Plot of Murmur Category

- Normal

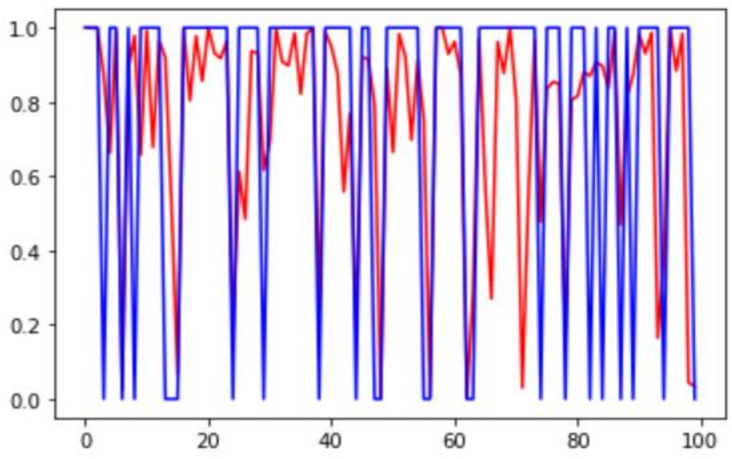


Figure 5.5.3: Evaluation Plot of Normal Category

5.6 Misclassified Category

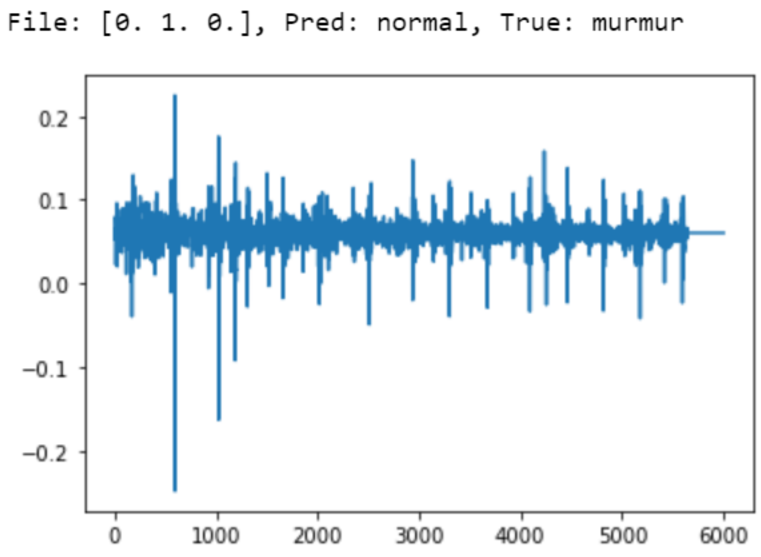


Figure 5.6: Misclassified Category

5.7 Correctly Classified Category

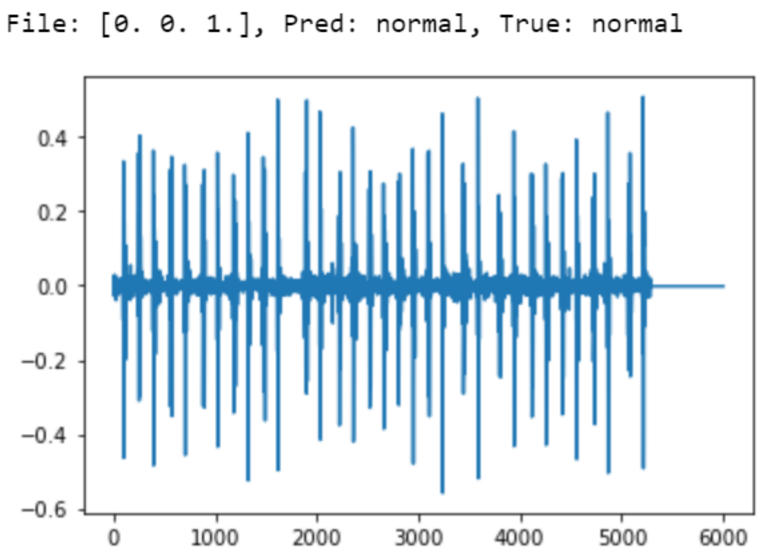


Figure 5.7: Correctly Classified Category

CHAPTER 6

CONCLUSION

In this project, we proposed a method for classifying normal, murmur and extra heart sounds. Classification of heartbeat sounds was conducted using a Convolutional Neural Network. We have made the machine to learn the features by its own by passing whole raw audio in time domain representation and make the classification decision. A GAP layer was introduced into the designed CNN in order to obtain global information about the feature maps to improve the classification performance. This was also shown to reduce the model's complexity and avoid overfitting. Class weights were set during the training process to solve the problem of class imbalance, which is an issue that commonly arises during the processing of medical images and signals. The best results were obtained with 20 epochs on batch size of 8. The training accuracy is 86, while the testing accuracy rate is 85. The results obtained in this project have demonstrated the efficacy of the proposed method. This result is crucial for the further realization of automatic diagnosis of heart disease.

REFERENCES

[1] [Z.-J. Yang, J. Liu, J.-P. Ge, L. Chen, Z.-G. Zhao, and W.-Y. Yang, 2011] “Prevalence of cardiovascular disease risk factor in the Chinese population: the 2007–2008 china national diabetes and metabolic disorders study,” European heart journal, vol. 33, no. 2, pp. 213–220, 2011.

[2] [Bentley P. and Nordehn G. et al.,2011] The Classifying Heart Sounds Challenge 2011 http://www.peterjbentley.com/heartchallenge/index.html"

[3] [Davis and Mermelstein, 1980] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE transactions on acoustics, speech, and signal processing, 28(4):357–366, 1980.

[4] [Luigi Bungaro, 2018] How to detect anomalies in Audio Signal Processing of the heart with the sound coming from mobile phone https://medium.com/@luigi.bungaro/how-to-detect-anomalies-in-audio-signal-processing -of-the-heart-with-sound-coming-from-mobile-e034e8fd709b

[5] [Jonathan Rubin, et al. 2017] Recognizing Abnormal Heart Sounds Using Deep Learning arXiv:1707.04642v2

[6] [Gaurang Jungare et al, 2018] Heart Anomaly Detection using Deep Learning approach based on PCG signal analysis. IRAJ International Conference, 2018, Pune, India

[7] [Siddique Lati et al, 2018] Phonocardiographic Sensing using Deep Learning for Abnormal Heartbeat Detection arXiv:1801.08322v3, IEEE