CHAPTER 1

INTRODUCTION

1.1 Motivation

1.1.1 Removal of Noise from the sound

Nowadays there are increasing applications of audio analysis and classification in every field. Voice assistants also use audio analysis to understand their commands. Sometimes there is a need to separate background noises from the primary noises in an audio sample to make sense of the main part of the audio.
For example: If there is an audio sample of a conversation between two people in an open area so there is a need to remove the environmental sounds to get the contents of the real conversation so there is a requirement to classify parts of sounds as environmental noises so that we can remove them.

1.1.2 Identifying the Source of the sound

Another application motivates us to classify sound samples which is that we can identify the source(device) or the performer of the audio or the song that you are listening to. This application is different from identifying the sound or song track from their acoustic fingerprint as the acoustic fingerprint of a particular soundtrack is the same and remains the

same for recorded versions of the soundtrack. We just need to create and compare the fingerprints.

Our application requires an analysis of features of sound that represents the characteristics of the source(human or device) of that sound.

1.2 Key Challenges

- With large datasets, the training of the model was a time-consuming task. It took hours to train the whole data. To avoid this we shifted to Google Colab to run our model.

- Initially, we tried using Tensorflow and found it complex and the error resolving part was difficult. So we tried Keras which is an abstract layer above TensorFlow and makes the code rather compact.

- Other challenges here include the complexity and processing time it requires to extract features and train them.

1.3 Problem Addressed

The project addresses the sound classification in simple words. In the urban society, there are various types of noise and they had to be classified in certain scenarios where the noise turns out to be evil and had the potential of destroying the entire integrity and aspects of certain projects which require a certain type of sound for their functioning, There is a need to separate background noises from the primary noises in an audio sample to make sense of the main part of the audio. Voice assistants also use audio analysis to understand their commands. Another common aspect regarding the same could say, if there is an audio

sample of a conversation between two people in an open area so there is a need to remove the environmental sounds to get the contents of the real conversation so there is a requirement to classify parts of sounds as environmental noises so that we can remove them and the overall dimensions to which the project could help is limitless with uncountable possibilities in time to come.

1.4 Approach to the Problem

1.4.1 Data Analysis and Exploration

The dataset that we have used in this project is the urbansound8k dataset which consists of 8732 audio samples of 4s categorized into 10 classes namely: air conditioner, car horn, children playing, dog park, drilling, engine idling, gunshot, jackhammer, siren, and street music.

The objective here is to convert each audio sample into such a domain so that we can easily explore the various characteristics of the samples present in different categories and try to obtain a clear difference between the categories.

1.4.2 Feature Extraction from the Data

Based on the analysis of the data our objective here is to obtain features for the dataset which can be used to train our neural network architecture.

Another objective in this part is to only consider those features that are based on how humans perceive sound information

1.4.3 Training a CNN architecture

Using the features extracted we want to train a convolutional neural network and try and refine the architecture and the parameters such as the number of neurons in the convolutional layer, filter size, optimizer used, learning rate, etc.

The basic flow of steps in the implementation of the project is depicted below.

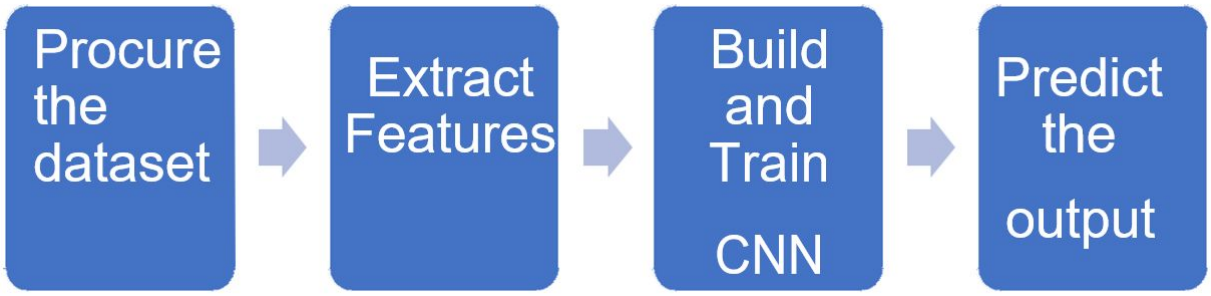


Figure 1: Steps involved in the project

CHAPTER 2

METHODOLOGY OF THE WORK

2.1 Data Analysis and Exploration

The UrbanSound8K dataset consists of 8732 audio clips belonging to 10 different classes. The clips are labeled and the metadata(named metadata) is stored in a different file.
We use the librosa library to convert the clips to a mono(single) channel. The sounds were sampled and the sampling rate was found to be 22050 samples/sec. The samples are now represented as a 1D array containing float values.

A few of the signals were decomposed into harmonic and percussive parts. The harmonic part of the signal is generated by non-pitched instruments and Percussive is attributed to non-pitched instruments.

When a spectrogram is plotted for a sound signal, harmonic components appear as horizontal lines on the spectrogram while percussive components are visible as vertical lines.

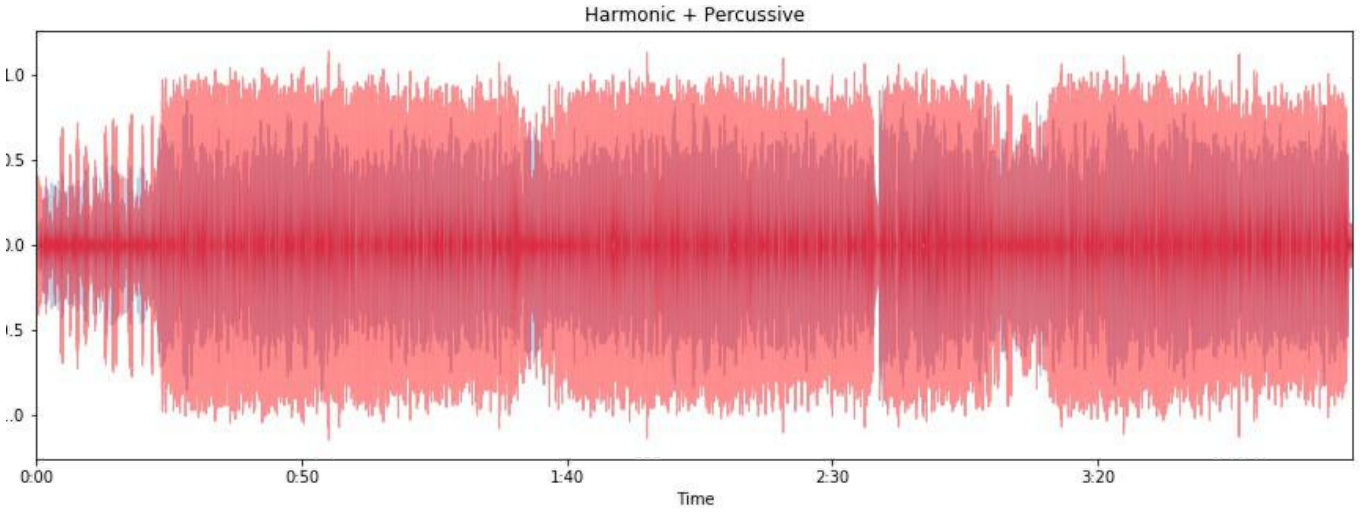


Figure 2.1: Spectrogram of harmonic and percussive components

2.2 Data preprocessing and Feature Extraction

Since the sampling rate is 22050 samples/sec and if we take each float value in 1D sampled array as a feature to be input to the CNN, then even for a 1-sec clip the number of features is 22050 and as the standard length of each clip in the UrbanSound8K dataset is 4 secs, the number of feature will be 88200, which is a huge number and will require a lot of computational resources and time.

Since it is infeasible to use the clips as they are as input to the CNN, we needed to reduce the number of features/inputs and bring down the number significantly. Some kind of preprocessing and useful metrics to classify the sounds are needed. These identified metrics will be used as features for the sound and used as input to the Convolutional Neural Network(CNN).

As the dataset is predominantly clean with no unnecessary noises or missing values and all clips are neatly labeled, so cleaning of the data is not needed, hence data cleaning steps were not needed.

The major work in this section was that of extracting and identifying the metrics using which the data will be classified into various classes.

The major features used in the CNN are :

1. MFCC: Mel Frequency Cepstral Coefficients are a set of features that can be used to model a sound digitally to considerable accuracy. This can be done because MFCC(although small in number) can describe the overall shape of the spectral envelope. **We are extracting 30 MFCC and then taking their mean and standard deviation over all the time-frames as the features.** The process of calculating MFCCs from an audio clip is depicted in the image below.

MFCCs are obtained by applying a double fast Fourier transform on the audio signal broken into overlapping frames and a Mel filter used in between.

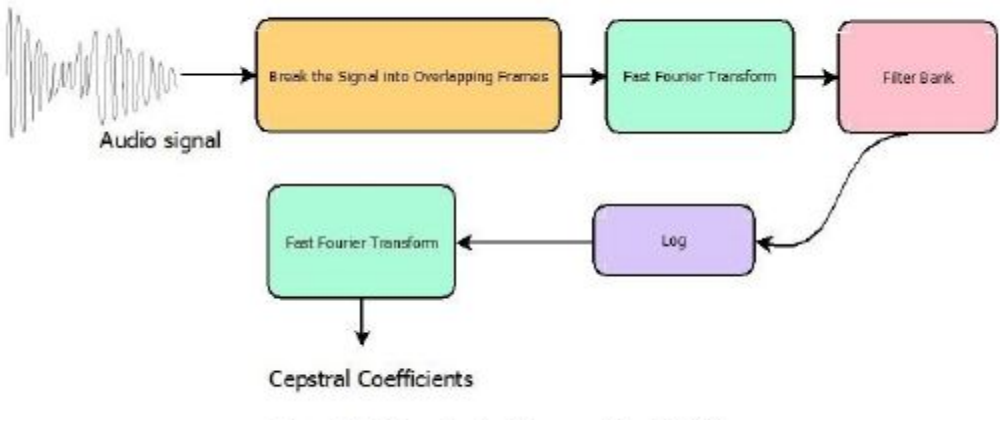


Figure 2.2: Steps involved in computing MFC

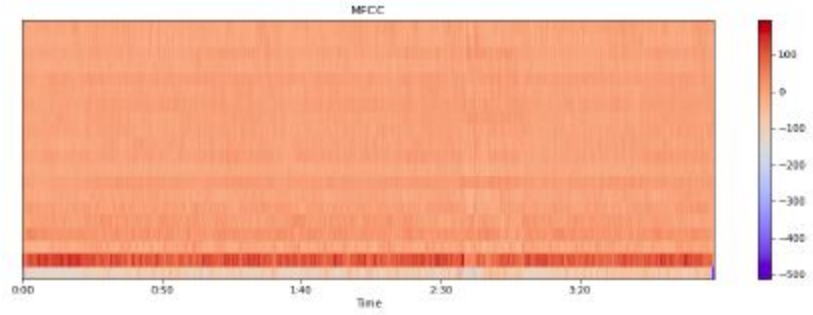


Figure 2.3: Spectrogram of MFCCs of Dog's bark

2. Mel-Spectrogram: The MelSpectrogram is a spectrogram in which the y-axis is the Mel scale and the x-axis is time.

The main purpose of the Mel scale is that sounds at equal distance on the Mel scale, "sound" to humans as such they are equal in distance from each other.

We take the mean and standard deviation of the float values returned at all frames and use them as features. Mel scale uses decibels.

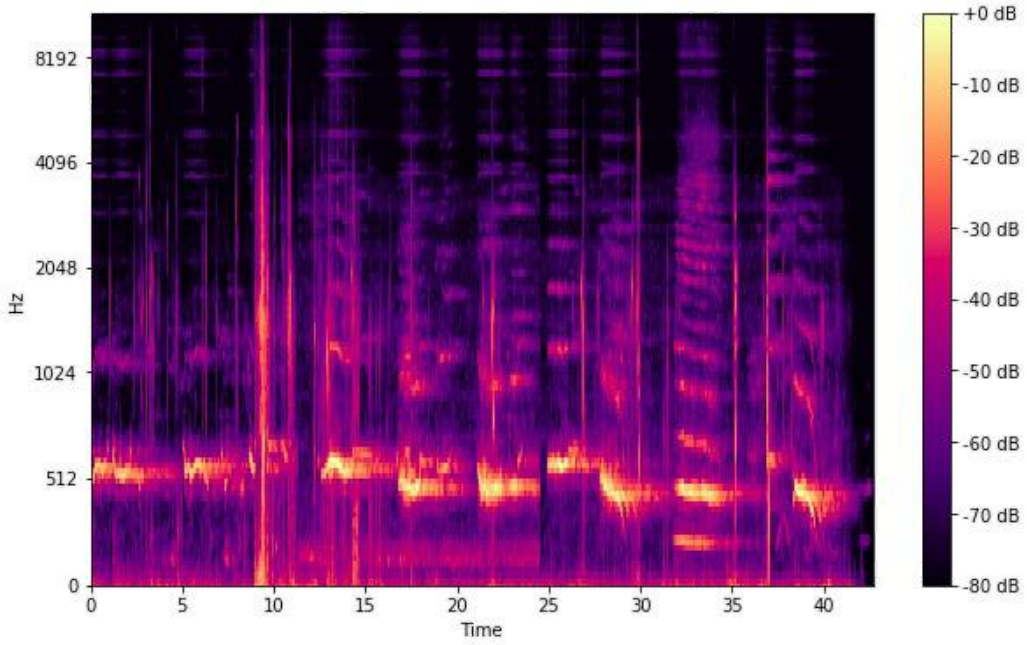


Figure 2.4 Mel Spectrogram of Dog's bark

3. Chroma-stft: STFT stands for short term Fourier transform. The STFT represents a signal in the time-frequency domain by computing discrete Fourier Transforms. It is an example of time-frequency distribution.

Stft is used to obtain time localized frequency information for audio clips/signals where the frequency varies over time. The normal Fourier transform provides frequency information over the entire signal while stft provides information over short intervals.

Although the Short Term Fourier Transforms(stft) (narrow-width) window provides better results in the time domain but it also has adverse results in the frequency domain. So it should be used on short time windows or better results.

STFT is computed as a series of Fast Fourier Transforms on windows of data where we slide over the windows similar to the sliding window paradigm in algorithms.
It can be viewed as producing a Fourier Transform centered around each time frame. As the time frames advance, a sequence of spectral transforms is obtained.

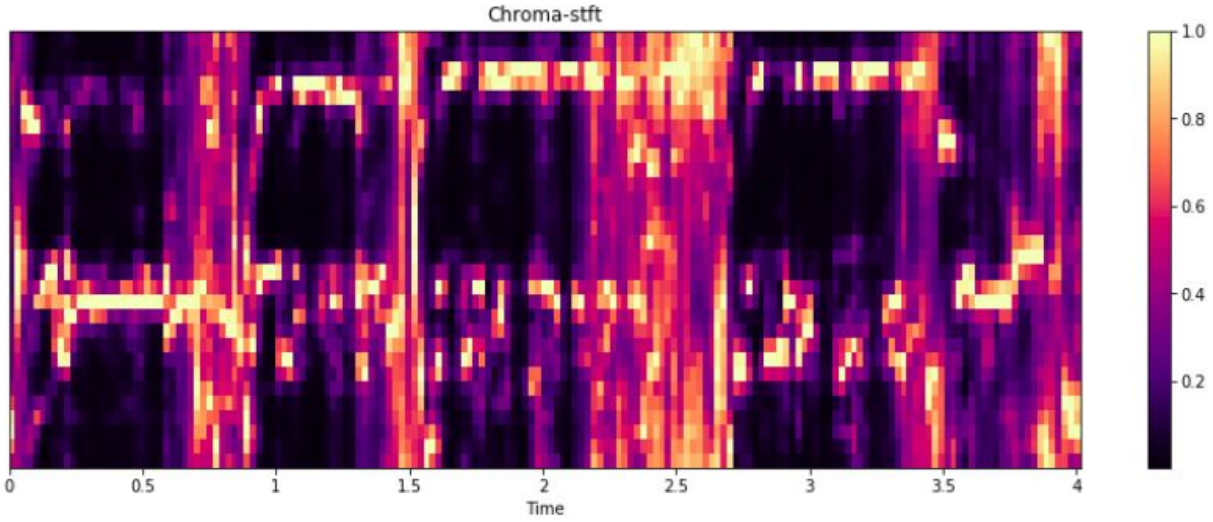


Figure 2.5: Spectrogram of Chroma stft Dog's bark

4. Chroma CQT: Constant Q Transform is similar to the Fourier transform. It is also a set of filters just like Fourier transform. The major difference between the two is that Constant

Q Transform has geometrically spaced center frequencies fk = f0 · 2 k b where b decides the number of filters per octave.

Another good thing about Constant Q Transform is its increasing time resolution towards higher frequencies, which is similar to the situation in our auditory system.

The major feature that makes Constant Q Transform useful is that if f0(the minimal center frequency) and b are chosen appropriately, the center frequencies correspond directly to the musical notes.

We take 30 CQT at all time frames and use their mean and standard deviation as features.

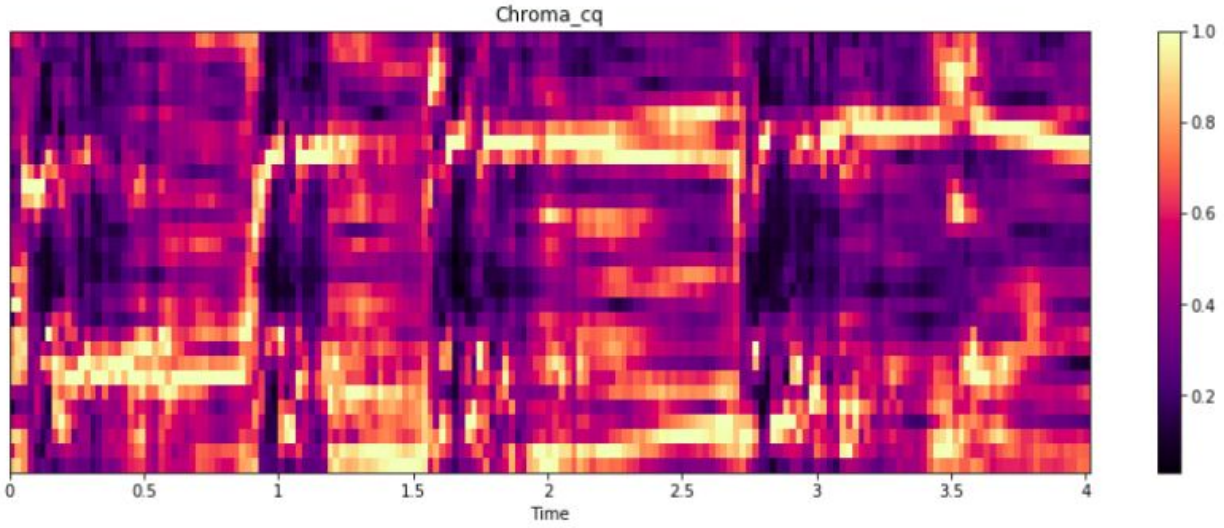


Figure 2.6: Spectrogram of Chroma CQT of Dog's bark

5. Chroma_cens: Chroma Energy Normalized Statistics take statistics over large windows to smooth the deviations in tempo, articulation, and musical instruments.

CENS features are resistant to changes over short windows, timbre, articulation as these are averaged statistics over large windows so they can be used for audio matching and identification.

Just like other normalization techniques CENS also balances out the differences in dynamics across and within the audio clips. The metric across which the chroma vector is normalized is the euclidean norm.

After normalization, quantization and smoothening techniques are applied over the normalized chroma vector. The resulting features of the normalized, quantized, and smoothened vector are called Chroma Energy Normalized Statistics(CENS).

We take 30 CENS coefficients at all time frames and use their mean and standard deviation as features.

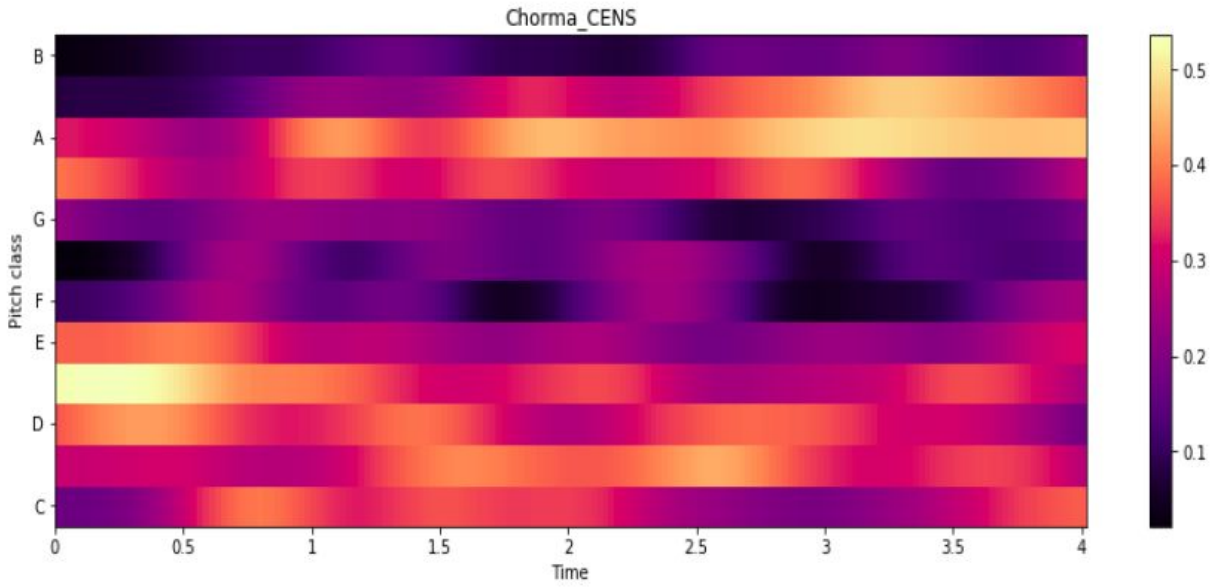


Figure 2.7: Spectrogram of Chroma Cens of Dog's bark

2.3 CNN IMPLEMENTATION

2.3.1 Basics

CNN is a deep learning technique that is used for the classification of 2d data. It appoints weights to various aspects of the data and helps classify the data into various categories.
A CNN architecture consists of the layer that tries to find out patterns out of 2-dimensional Data.
The various layers present in a CNN are :

Convolutional Layer: The method of convolution of 2d data with a filter is the summation of all the values in the dot product of the filter with the data. Applying the same filter throughout the whole image results in a feature map.
A CNN layer consists of many such filters and each of them is convolved with the image and a feature map is obtained.

Pooling Layer: This layer is present to reduce the size of the feature map so that the computational power required to process the data decreases. There are two kinds of pooling layers, one is average pooling and the other one is max-pooling. Average pooling takes the average of the values of the portion of the data on which the filter is applied and Max pooling takes the max.

Dense Layer: This is just a regular layer consisting of neurons. Every neuron in this layer receives the input from each neuron in the previous layer thus is called fully connected or dense. This layer calculates the total input and then applies an activation function to it.

2.3.2 Architecture

1. First Architecture :

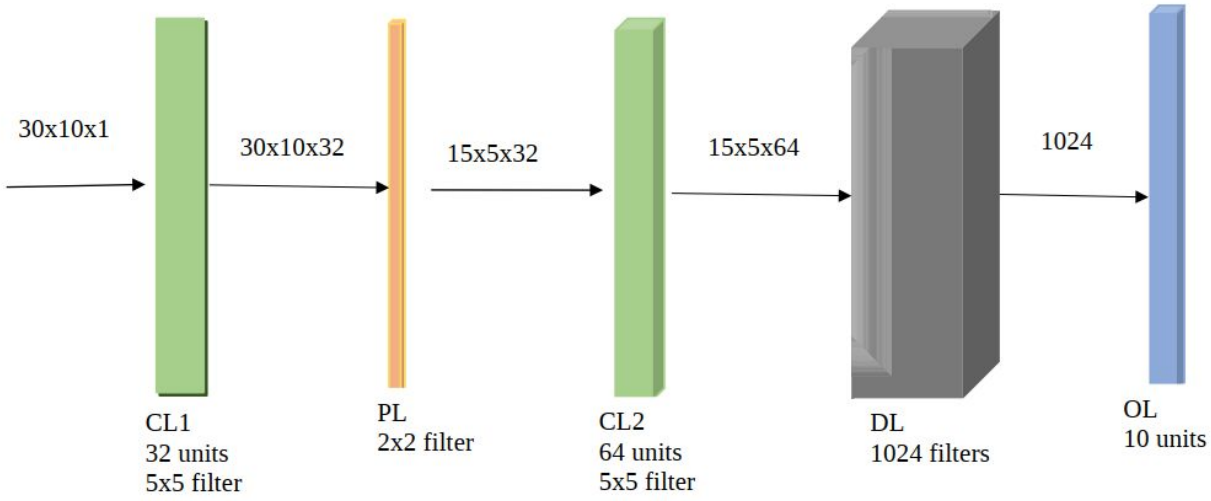


Figure 2.8:Architecture I

CL 1 - Convolutional Layer 1
PL - Partition Layer
CL 2 - Convolutional Layer 2
DL – Dense Layer
OL – Output Layer

After many rounds of exploration and experiments, we found out that a new architecture with the 2 convolutional layers, 2 max-pooling layer, 3 dense layer was optimal for this project.

2. Second Architecture :

Figure 2.9: Architecture II

CL	-	*Convolutional Layer*
MP	-	*Max-pooling Layer*
DROP	-	*Dropout Layer*
DENSE	-	*Dense Layer*
FLAT	-	*Flattening Layer*

2.3.2 Training

Initialization of Weights:

The first step in the training of CNN is the initialization of weights and biases with random values using normal distribution.

This is the model initialization according to the first architecture:

```
def build_model():

    model = Sequential()

    f_size = 3
    model.add(Convolution2D(24, f_size, f_size, border_mode='same', input_shape=(bands, frames, num_channels)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Activation('relu'))
    model.add(Convolution2D(48, f_size, f_size, border_mode='same'))
    model.add(Activation('relu'))
    model.add(Convolution2D(48, f_size, f_size, border_mode='valid'))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(64, W_regularizer=l2(0.001)))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_labels, W_regularizer=l2(0.001)))
    model.add(Dropout(0.5))
    model.add(Activation('sigmoid'))
    sgd = SGD(lr=0.001, momentum=0.9, decay=0.0, nesterov=True)
    model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=sgd)

    return model
```

Figure 2.10: Code snippet for model initialization(Architecture I)

This is the model initialization according to the second architecture:

```
#adding layers and forming the model
model.add(Conv2D(64,kernel_size=5,strides=1,padding="Same",activation="relu",input_shape=(40,5,1)))
model.add(MaxPooling2D(padding="same"))

model.add(Conv2D(128,kernel_size=5,strides=1,padding="same",activation="relu"))
model.add(MaxPooling2D(padding="same"))
model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(256,activation="relu"))
model.add(Dropout(0.3))

model.add(Dense(512,activation="relu"))
model.add(Dropout(0.3))

model.add(Dense(10,activation="softmax"))
```

Figure 2.11: Code snippet for model initialization(Architecture II)

There is the use of regularization(L2) and also changes in the number of layers which prevents the model from overfitting on the training data and hence improves the results.

Optimization using optimizer:

There are various kinds of optimizers available for the optimization of weights in CNN. Many of these implementations are available in Keras and TensorFlow for our use.

The basic concept used by all of these optimizers is to update weights by using gradient descent and propagating the error backward into previous layers(Back Propagation)

The basic approach is explained in the figure given below:

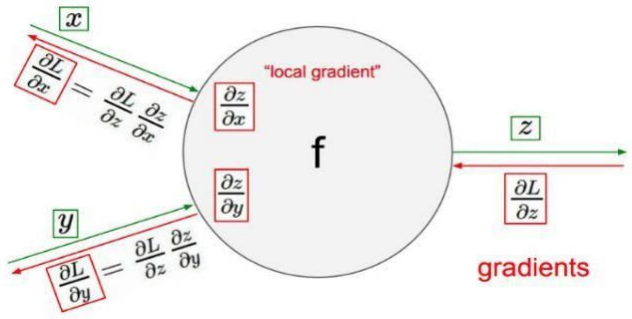


Figure 2.12: flow of gradient

One Epoch:

What one run of an optimizer does is goes through the training data, predict the output based on the current weights, and then calculates the error for each data point and after updates the weights using gradient descent after batch_size number of data points have been iterated through.

The number of epochs is decided by experiment. We try different values and study the loss function vs epoch_number plot to find out the point where to stop.

Learning Rate:

The learning rate is another factor that determines how fast the loss function will converge to minima. A very large value can result in overshooting and may result in divergence. A

very small value will result in a long time to converge which increases the number of epochs. Thus we have to experiment with this value and find out the best that suits the dataset and the architecture.

2.4 CNN REFINEMENT

The following parameters are fine-tuned for the CNN architecture :

1. **Number of conventional Layers**: We initially started with 3 convolutional layers having respectively with filter size 3 * 3. The work of a convolutional layer is to create a feature map. The number of units and the number of is found out by exploration. We found the best results for 2 convolutional layers with 2 max-pooling layers and 3 dense layers.

2. **Number of dense Layers**: We initially started with 2 dense layers**.** We found that 3 dense layers give us greater accuracy.

3. **Filter size**: The filter size determines whether we focus on minute details or try to figure out a larger pattern. We started with a filter size of 3*3 and then adjusted it to 5*5.

4. **Max pooling size**: The max-pooling layer helps in reducing the processing time by reducing the feature map size. Also, the pooling size has to be compatible with the feature map sizes.

5. **Dropout Layer**: Dropout can help a model generalize by randomly setting the output for a given neuron to 0. In setting the output to 0, the cost function becomes more sensitive to neighboring neurons changing the way the weights will be updated during the process of backpropagation.

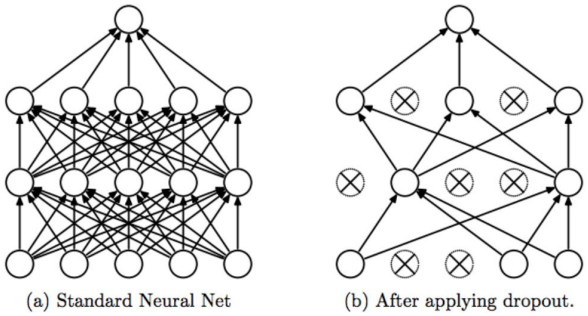


Figure 2.13: Dropout mechanism in Neural Net

6. **Momentum factor**: While using gradient descent we might get stuck at local minima. To avoid this we use the momentum factor which helps us move beyond a local minima.

7. **Early Stopping and Checkpointing in Keras:** An excessive number of epochs can prompt overfitting of the training dataset, while too few may bring about an underfit model. Early stopping is a strategy that permits you to determine a discretionary large number of training epochs and stop training once the model performance stops improving on a hold-out validation dataset.

Early stopping of training is supported by Keras via a callback called EarlyStopping. This callback permits you to specify the performance measure to observe the trigger, and once triggered, it will halt the training process. When instantiated via arguments, the EarlyStopping callback is configured

The EarlyStopping callback will stop training once set off, but the model at the end of training may not be the model with the best performance on the validation dataset.

An additional callback is needed that will save the best model ascertained throughout training for later use. This is the ModelCheckpoint callback.

The ModelCheckpoint callback is versatile in the way it is used, however in this case we are going to utilize it solely to save the best model ascertained during training as characterized by a chosen performance measure on the validation dataset.

“Overall with the addition of above callbacks accuracy improved by 0.6%”.

CHAPTER 3

RESULT CONCLUSIONS AND FUTURE WORK

3.1 RESULT

As mentioned above the dataset we have has the entire dataset divided into 10 folds. So we perform 10 fold validation here.

Also, two architectures are being used here so we present two sets of scores.

3.1.1 Architecture 1

The training and testing for this architecture is done using 10 fold cross-validation the results for various folds are shown below:

Testing Accuracy

FOLD NO.	ACCURACY (100 Epochs)	ACCURACY (150 Epochs)
1	75.10	75.54
2	72.54	74.38
3	64.57	64.39
4	60.97	63.01
5	65.13	63.66
6	68.58	69.91
7	60.51	61.53

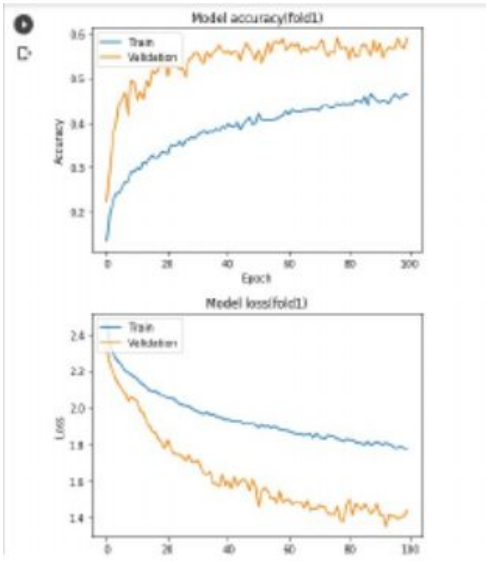
8	57.83	60.80
9	63.40	62.50
10	60.13	60.15

Table 3.1: Testing Accuracy(Architecture I)

Average accuracy with 100 epochs = 64.487%
Average accuracy with 150 epochs = 65.587%

Various plots of loss function and training and validation accuracy for different folds :

FOLD 1 :



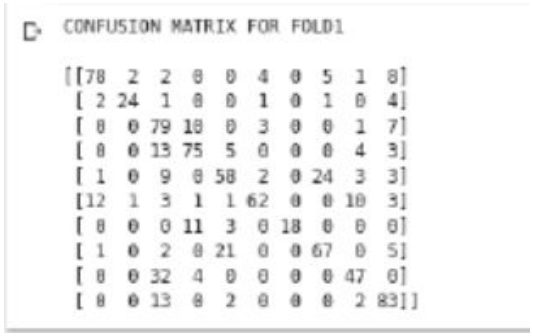
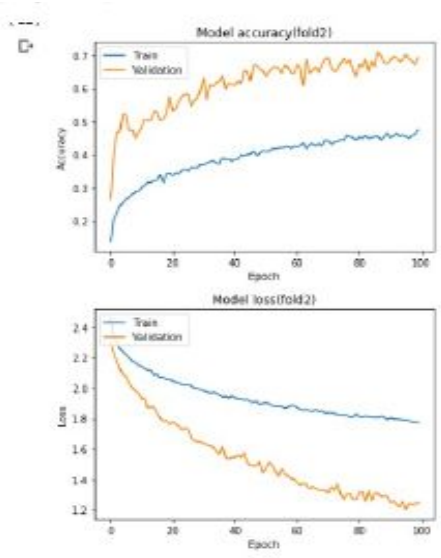


Figure 3.1: loss and accuracy plots for FOLD1 and confusion matrix for FOLD1

FOLD 2:

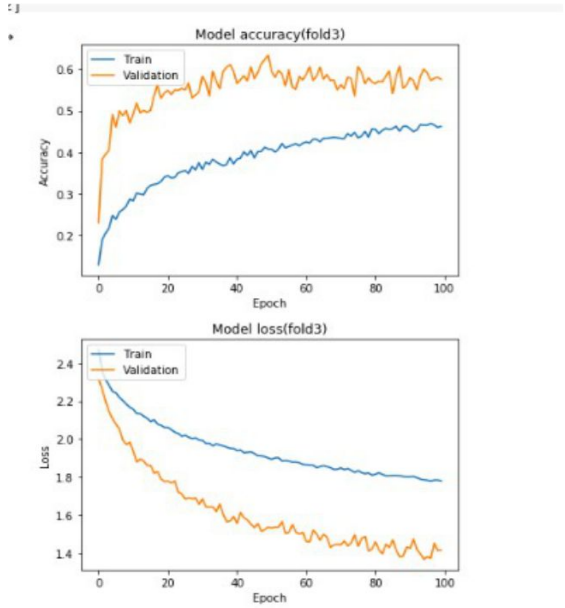



```
CONFUSION MATRIX FOR FOLD2

[[42  0  0  2 27  0  0  1  0 28]
 [ 1 22  0  0  0  6  0  1  0  2]
 [ 0  0 85  6  0  2  0  1  4  2]
 [ 8  0  5 74  1  5  0  3  1  3]
 [ 0  6  0  0 55  5  1 32  0  1]
 [ 2  0  2  0  0 78  0  0  0  7]
 [ 0  0  0  9  1  0 21  0  0  0]
 [ 0  0  3  0  7  4  0 68  0  0]
 [ 0  0  0  5  0  0  0  0 75  2]
 [ 2  0 21  2  1  0  0  1  1 72]]
```

Figure 3.2: loss and accuracy plots for FOLD2 and confusion matrix for FOLD2

FOLD 3:

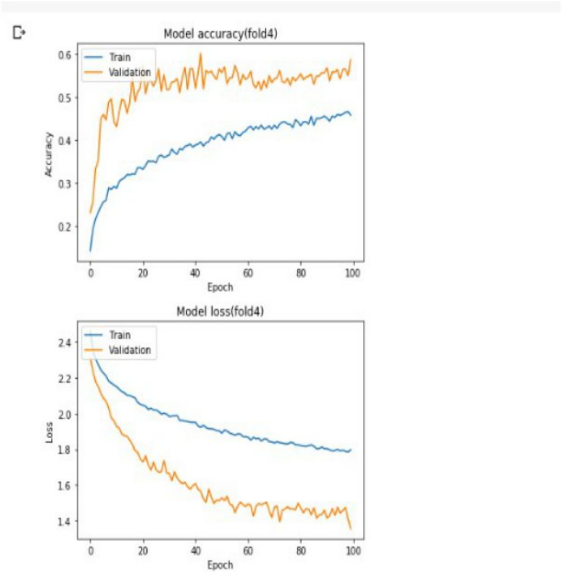



```
CONFUSION MATRIX FOR FOLD3

[[34  0  0  0 12 18  0  1  5 30]
 [ 0 24  1  0  1  2  0  0  0  2]
 [ 3  0 61  4  0 15  0  0  1 16]
 [ 4  3 11 65  1  3  0  0 11  2]
 [ 0  0  0  4 78  0  2 13  2  1]
 [50  0  0  0  0 30  0  1  4  3]
 [ 0  0  0  0  1  0 29  0  0  0]
 [ 0  0  0  0 22  0  0 55  0  1]
 [ 1  0  4  0  0  1  0  0 74  0]
 [ 2  1  4  1  2  6  4  3  7 70]]
```

Figure 3.3: loss and accuracy plots for FOLD3 and confusion matrix for FOLD3

FOLD 4:

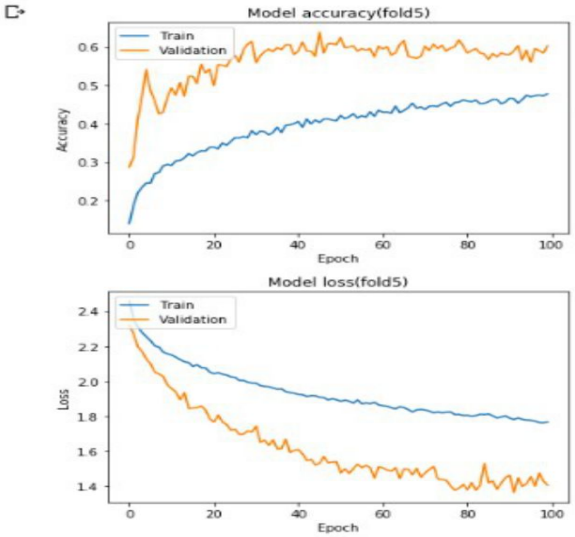



```
CONFUSION MATRIX FOR FOLD4

[[30  0  0  0  0 41  0 29  0  0]
 [ 2 12  4  1  2  2  0  3  0  2]
 [10  0 58  1  2  4  0  2  0 23]
 [ 0  0  8 85  0  1  0  0  3  3]
 [ 0  2  2  1 53 11  0 12  1 18]
 [15  1  3  2  0 59  0 24  1  1]
 [ 0  0  1  2  5  1 42  0  0  0]
 [ 1  0  0  0 16  0  1 55  0  3]
 [ 0  0  5  4  1 13  0  2 48  4]
 [ 0  0 25  4  0  0  0  0  2 69]]
```

Figure 3.4: loss and accuracy plots for FOLD4 and confusion matrix for FOLD4

FOLD 5:

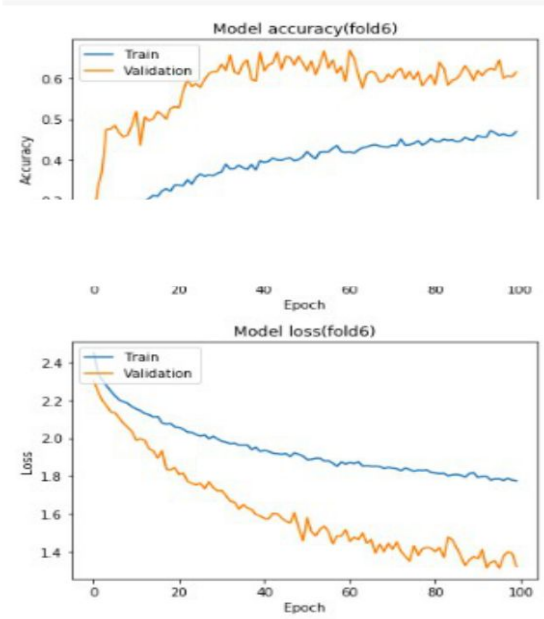



```
CONFUSION MATRIX FOR FOLD5

[[40  0  5  1  1 41  0  0  3  9]
 [ 1 16  0  0  3  6  0  0  1  1]
 [ 0  0 63  4  3  1  0  2  1 26]
 [ 2  3  8 67  6  2  2  3  2  5]
 [ 0  1  0 12 68  0  5 14  0  0]
 [17  8  0  0  1 74  0  0  2  5]
 [ 0  0  0  1  4  0 41  0  0  0]
 [ 4  0  0  0  7 11  0 44  0  2]
 [ 6  0  5  1  2  5  1  0 54  0]
 [ 4  3 15  1  2  6  0  0  0 69]]
```

Figure 3.5: loss and accuracy plots for FOLD5 and confusion matrix for FOLD5

FOLD 6:

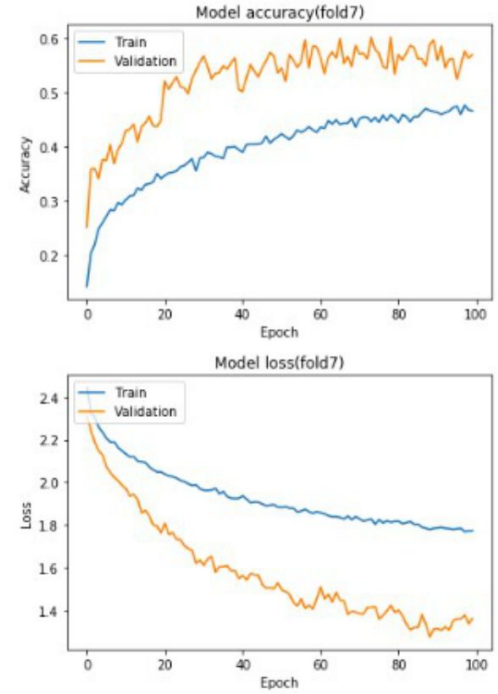



```
CONFUSION MATRIX FOR FOLD6

[[51  0  6  4  0 25  0 14  0  0]
 [ 1 66  3  2  6 10  0  4  2  4]
 [ 0  0 75  5  9  0  0  0  4  7]
 [ 0  0 12 65  2  5  3  0 10  3]
 [ 1  0  1  9 54  5  0 22  8  0]
 [23  2  8  0  3 70  0  0  0  1]
 [ 0  0  0  2  1  1 36  0  0  0]
 [ 2  0  0  0 18 11  0 83  1  5]
 [ 0  2  1  0  0  0  0  0 68  0]
 [ 2  0 13  6  1  1  0  0  3 74]]
```

Figure 3.6: loss and accuracy plots for FOLD6 and confusion matrix for FOLD6

FOLD 7:

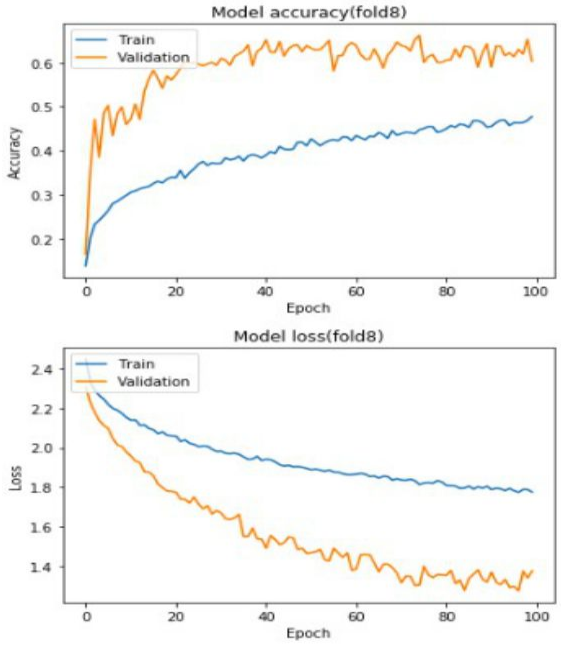



```
CONFUSION MATRIX FOR FOLD7

[[ 22   0   0   6   9  10   0  51   1   1]
 [  7  24   0   2   4   2   0   9   2   9]
 [  0   0  55  21  10   0   3   1   0  10]
 [  2   0   5  87   2   0   4   0   0   0]
 [  1   4   0   2  69   2   0  16   2   4]
 [  2   2   0   7  36  32   0  15   1  12]
 [  0   0   0   7   2   0  29   0   0   0]
 [ 19   0   0   0  68   3   0  25   0   5]
 [  0   0  19   5   0   6   0   0 133   3]
 [  6   0  21   1   4   3   2   7   6  50]]
```

Figure 3.7: loss and accuracy plots for FOLD7 and confusion matrix for FOLD7

FOLD 8:

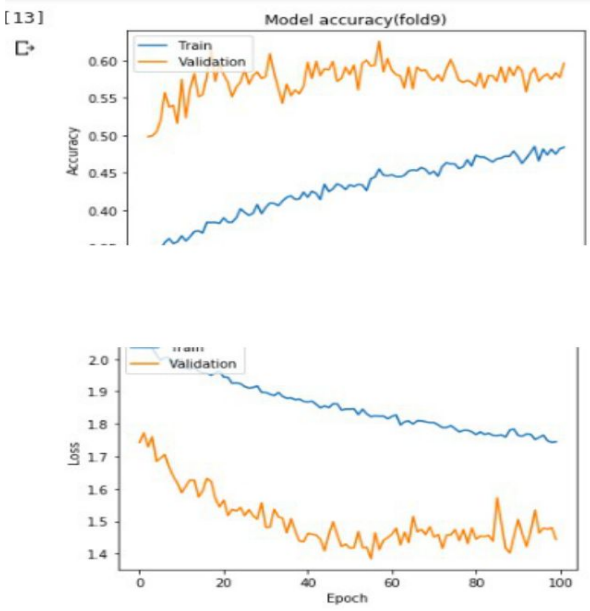



```
CONFUSION MATRIX FOR FOLD8

[[12  3  9 24  0  6  0 25 15  6]
 [ 1 35  0  0  2  0  0  3  2  0]
 [ 3  1 67 16  0  0  2  0  0 11]
 [ 2  0 14 67  1  1  0  0  5 10]
 [ 9  3 12  5 40 11  3  0  7 10]
 [ 2  2  5  2  0 37  1 53  1  4]
 [ 0  0  4  2  0  0 30  0  0  0]
 [17  0  2  0  4  5  0 92  0  0]
 [ 4  0  0  5  0  2  0  2 99  7]
 [ 1  3 37  0  1  1  0  0  1 56]]
```

Figure 3.8: loss and accuracy plots for FOLD8 and confusion matrix for FOLD8

FOLD 9:

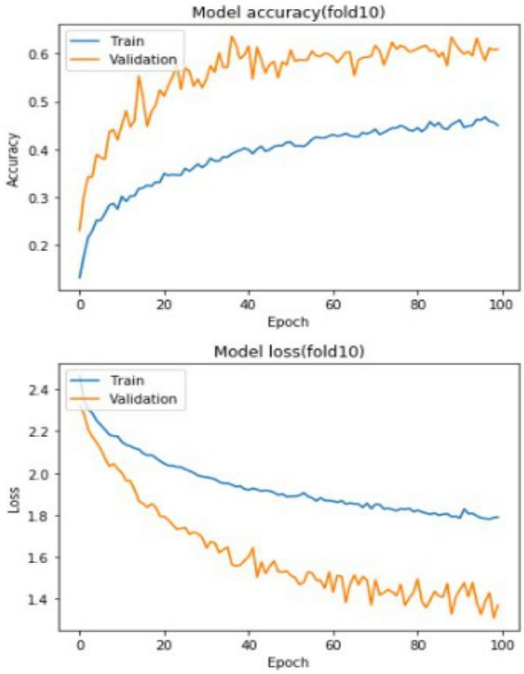



```
CONFUSION MATRIX FOR FOLD9

[[68  0  0  0  3 23  0  6  0  0]
 [ 6 18  0  0 12  1  0  0  4  1]
 [ 2  0 74  0  0  0  0  0  2 22]
 [ 2  5  5 83  0  0  0  0  1  4]
 [ 3  0  1  5 62  0 12 17  0  0]
 [17  0  8  2  0 54  0  0 17  2]
 [ 0  0  0 11  1  0 22  0  0  1]
 [ 0  5  0  0  9 71  0 33  0  2]
 [ 5  0  1  0  0 14  0  0 71  0]
 [ 2  0 18  0  0  1  0  1  0 78]]
```

Figure 3.9: loss and accuracy plots for FOLD9 and confusion matrix for FOLD9

FOLD 10:




```
[15] CONFUSION MATRIX FOR FOLD10

[[61  0  0  0  0 28  0  3  0  8]
 [ 1 27  0  1  0  3  0  3  1  0]
 [ 1  0 70  0  2  0  0  2  1 24]
 [ 4  1 10 69 10  0  4  0  1  1]
 [12  0  8  1 54  2  1 13  0  9]
 [ 1  0  0  0 31 61  0  0  0  3]
 [ 0  0  0  1  8  0 26  0  0  0]
 [31  0  0  0 58  0  0 31  0  0]
 [ 0  0  7  3  2  0  5  0 67  2]
 [ 6  3 14  1  7  0  2  3  5 59]]
```

Figure 3.10: loss, accuracy plots for FOLD10 and confusion matrix for FOLD10

3.1.2 Architecture 2

A Convolutional Neural Network with dropouts was used to avoid overfitting.
The following results are obtained by training on folders 1-9 and testing on folder 10.

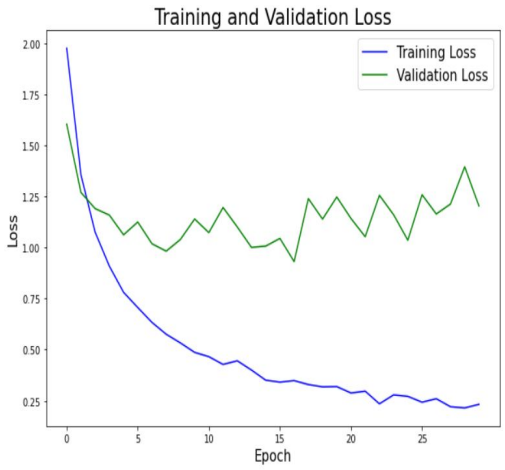
Testing Accuracy

FOLD NO.	ACCURACY (150 Epochs)	ACCURACY (150 Epochs)
1	98.7	770.2
2	98.6	69.9
3	98.9	66.0
4	98.9	68.5
5	98.6	74.6
6	98.8	70.9

7	98.5	60.2
8	98.9	71.1
9	98.4	68.0
10	99.0	73.5

Table 3.2: Testing Accuracy(Architecture II)

Train accuracy: 95.90%
Test accuracy: 73.11%



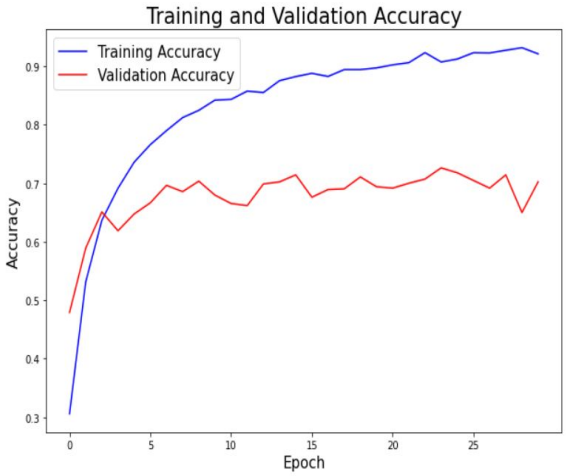


Figure 3.11: Training and Validation loss (Architecture-II)

Figure 3.12: Training and Validation Accuracy

FOLD 1:

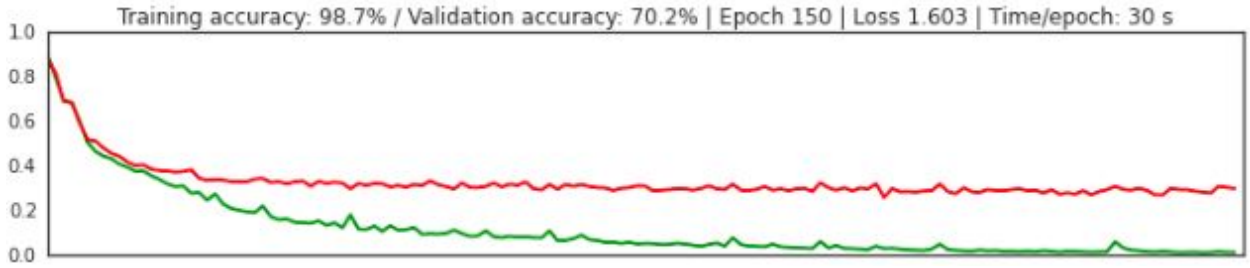


Figure 3.13: Training & Validation Loss plot for Fold 1

FOLD 2:

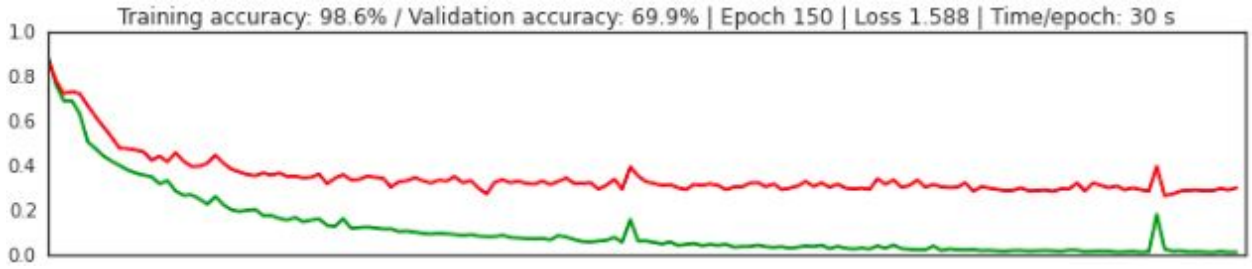


Figure 3.14:Training & Validation Loss plot for Fold 2

FOLD 3:

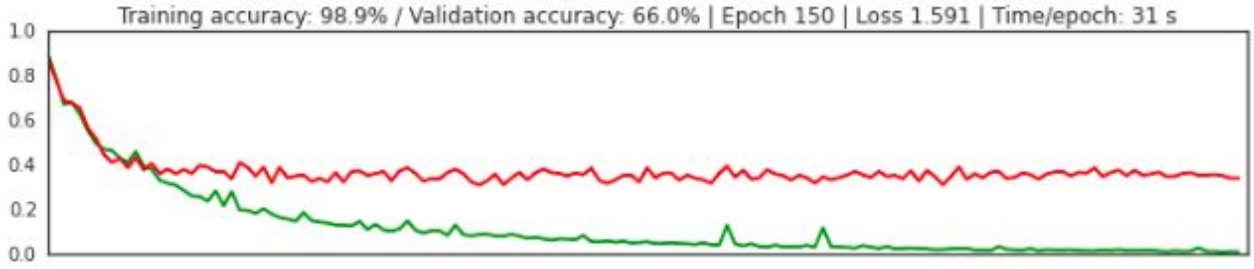


Figure 3.15:Training & Validation Loss plot for Fold 3

FOLD 4:

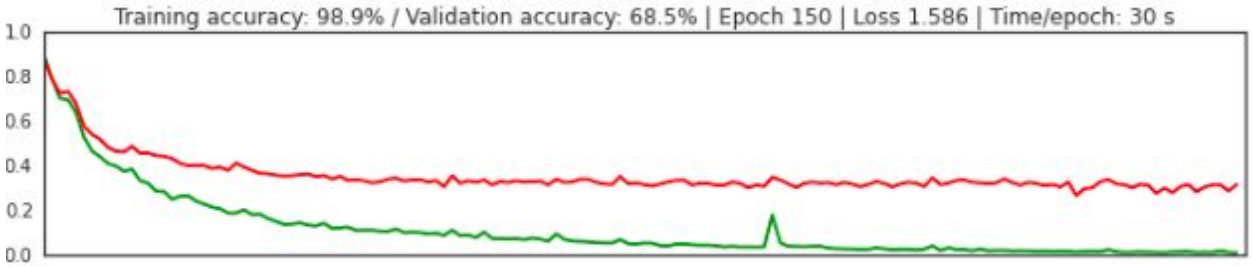


Figure 3.16: Training & Validation Loss plot for Fold 4

FOLD 5:

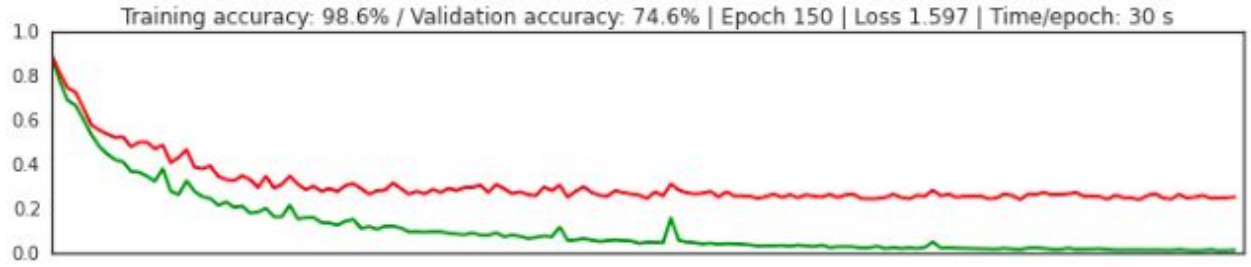


Figure 3.17:Training & Validation Loss plot for Fold 5

FOLD 6:

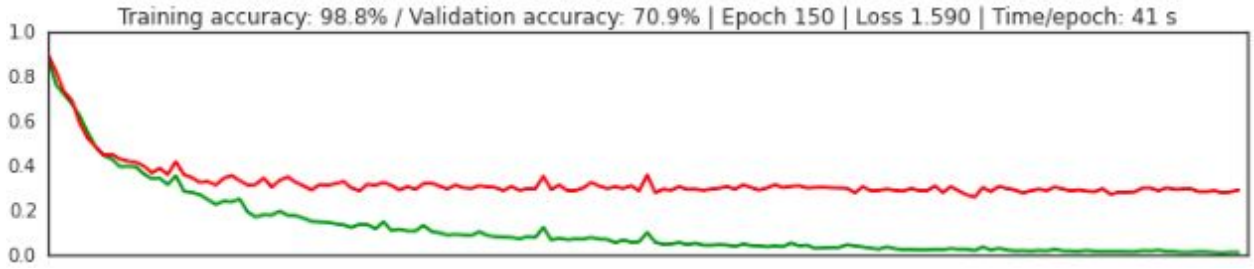


Figure 3.18:Training & Validation Loss plot for Fold 6

FOLD 7:

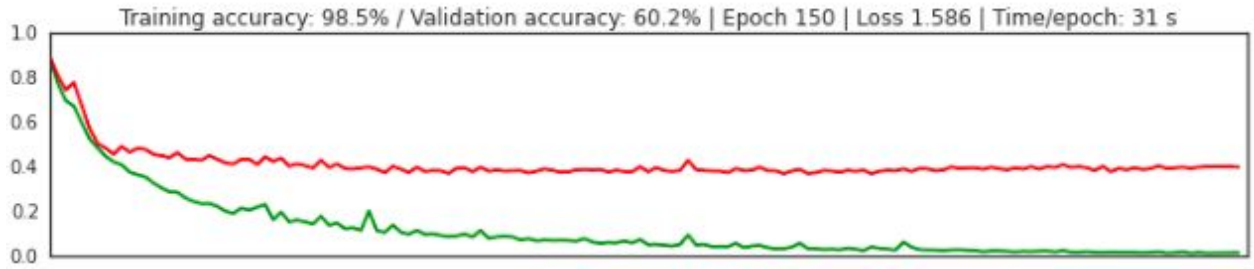


Figure 3.19:Training & Validation Loss plot for Fold 7

FOLD 8:

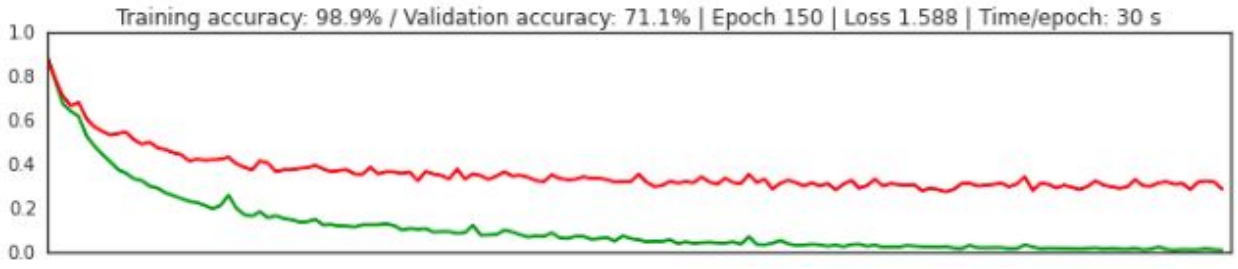


Figure 3.20:Training & Validation Loss plot for Fold 8

FOLD 9:

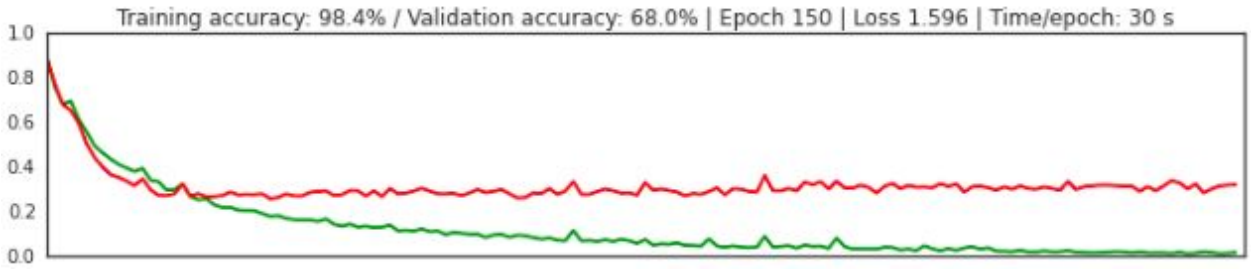


Figure 3.21:Training & Validation Loss plot for Fold 9

FOLD 10:

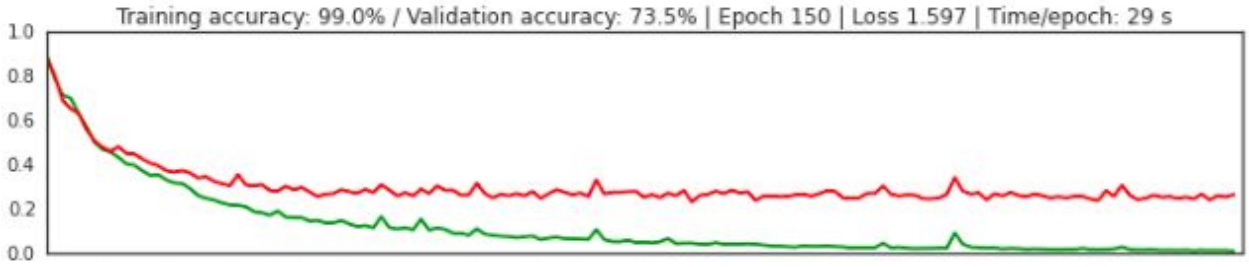


Figure 3.22:Training & Validation Loss plot for Fold 10

3.2 TASK, ACHIEVEMENT, and POSSIBLE BENEFICIARIES

Sound plays an important role in everyday human life and carries with it huge amounts of information, it can also enable automatic multimedia labeling. These models can be used for designing security systems. There are solutions available in the market but their efficiency is not up to the mark.

Deep learning architecture provides a better solution to this problem. In this paper, we use spectrograms obtained using librosa library and used small size filters to identify spectro-temporal patterns that indicate the class of the sound even if it's masked.

These models can be applied to fields such as context-aware computing, noise mitigation, and for surveillance purposes.

3.3 FUTURE SCOPE

The deep neural networks proposed in this report are dependent on the availability of larger training sets to learn a nonlinear function and generalize, scarcity of labeled data for audio classification poses a major problem. With the improvement in the datasets, the accuracy of the model can be increased.

A possible solution is data augmentation, which allows us to create more training samples by applying deformations on the dataset, by training on the deformed data we can expect the model to generalize better on unseen data. Few possible augmentation techniques that can be applied are time-shifting, pitch shifting, and time stretching.

We can use these models for outdoor environmental sound recognition. This can be used to detect “danger ” situations(using sound in a video surveillance system)
They also allow us to detect the type of environment, location in which the audio is recorded

The use of pre-trained models like VGG, ResNet-50 can reduce the model building time.