Machine Learning Engineer Nanodegree

Urban Sounds Classification using Deep Learning

Capstone Project

Valiveti Aditya Pramith Krishna

September 1st, 2019

UDACITY

1 Definition

1.1 Project Overview

Sound is all around us. Either directly or indirectly, people are always in contact with sounds. Sounds outline the context of our daily activities, ranging from the sound of a breath when we are alone to the music we dance to when we are in a party, and the other environmental sounds that we hear in every day such as a dog barking, the sound of the rain and the sound of traffic for an urban person. The human brain continuously processes and understands this sound subconsciously, giving us information about the environment around us without much of a thought.

Automatic environmental sound classification is a growing area of research with numerous real-world applications. Whilst there is a large body of research in related audio fields such as speech and music, but the work on the classification of environmental sounds is comparatively scarce. The problem here is the applicability of the Neural Network techniques in the domains, such as sound classification, as many of the discrete sounds happen all the time.

The goal of this capstone project is to apply Deep Learning techniques for the classification of environmental sounds, specifically focusing on the identification of particular urban sounds.

There are a large number of real-world applications where audio classification can be used such as:

- Noise-cancelling while audio recording to remove any background noises.
- Security Purposes where we can call emergency services in case of a disaster based on the sounds.
- Automotive where recognizing sounds both inside and outside of the vehicle can improve the safety and comfort of the travelers.
- Identifying different kinds of sounds to assist people with disabilities.

1.2 Problem Statement

The main objective of this project will be to use Deep Learning techniques to classify urban sounds. When given an audio sample in a computer readable format which in my case is WAV of a few seconds duration, we want to be able to determine if it contains one of the target urban sounds with a corresponding likelihood score. Conversely, if none of the target sounds were detected, we will be presented with an unknown score.

1.3 Metrics

The evaluation metric for this problem will be the 'Classification Accuracy' which is defined as the percentage of correct predictions.

Accuracy = correct classifications / number of classifications

Classification Accuracy was deemed to be the optimal choice metric as it is presumed that the dataset will be relatively symmetrical (as we will explore in the next section) with this being a multi-class classifier whereby the target data classes will be generally uniform in size.

Other metrics such as Precision, Recall (or combined as the F1 score) were ruled out as they are more applicable to classification challenges that contain a relatively tiny target class in an unbalanced data set.

2 Analysis

2.1 Data Exploration and Visualization

2.1.1 Datasets and Inputs

For this project, we will use a dataset called Urbansound8K [1]. The dataset contains 8732 sound extracts (<=4s) of urban sounds from 10 classes, which are as follows:

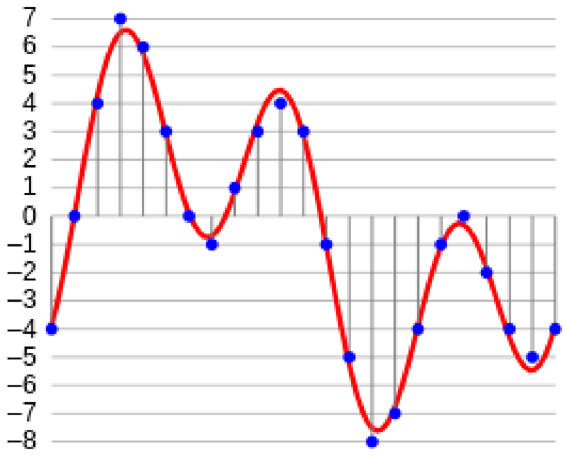
1. air_conditioner
2. car_horn
3. children_playing
4. dog_bark
5. drilling
6. engine_idling
7. gun_shot
8. jackhammer
9. siren
10. street_music.

The metadata accompanying the dataset contains a unique ID for each sound extract along with its given class name.

2.1.2 Audio sample file data overview

These sound extracts are digital audio files in .wav format. Sound waves are digitized by sampling them at discrete intervals known as the sampling rate of 44,100 times per second or 44.1 kHz typically. Each sample is the amplitude of the wave at a particular time interval, where the bit depth determines how detailed the sample will be also known as the dynamic range of the signal where typically 16bit which means a sample can range from 65,536 amplitude values. Therefore, the data we will be analyzing for each sound extracts is essentially a one dimensional array or vector of amplitude values.

This can be represented using the following image:



The data set is unbalanced as there are 1000 samples for all the classes and for siren we have 929 which not a huge difference when considered in a data set of 8732 samples but for car_horn and gun_shot classes there are considerably fewer samples at 429 and 347 respectively.

2.1.3 Analysing audio data

For audio analysis, we will be using the following libraries:

1 **Librosa** is a Python package for music and audio processing by Brian McFee and will allow us to load audio in our notebook as a numpy array for analysis and manipulation.
2 **Matplotlib** is used of ploting the feature extracts of the audio files

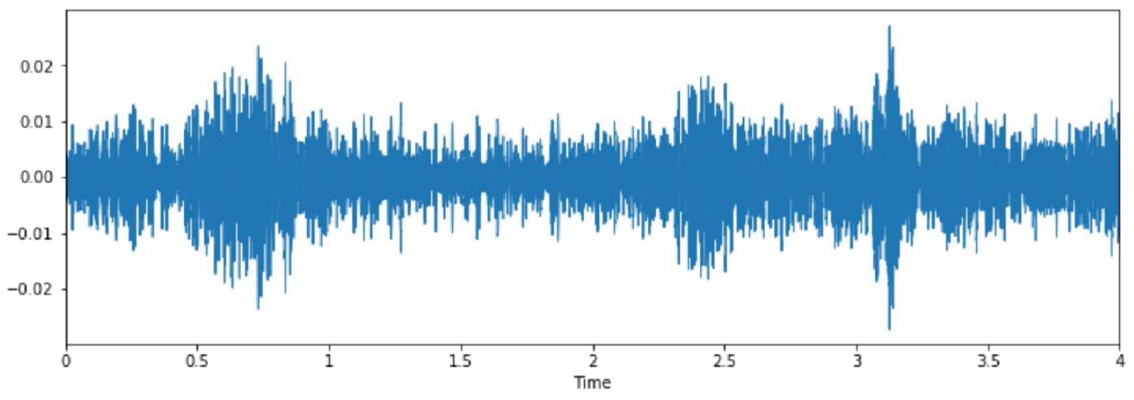
2.1.4 Audio Inspection and Visualization

To inspect the Audio we and use IPython.display.Audio to listen to the audio files.

```
import IPython.display.Audio as ipd
```

To Visualize the data as Waveforms we can use Librosa.display

We get the following wave plot of the audio



Since waveforms are similar for some of the repetitive sounds like Air conditioner, Drilling, Continuous Gunfire, and Engine Idling it is hard to differentiate using waveforms. We will look into different features for training our model. As discussed in the Proposal we are going to use the Spectrograms of Mel Coefficients and Chroma

2.1.3 Dataset and Metadata

Here we will load the metadata that was provided along with the dataset. To import the metadata from CSV to data frame we use Pandas library.

We display the head of the data set and it is as follows:

slice_file_name	fsID	start	end	salience	fold	classID	class
100032-3-0-0.wav	100032	0	0.317551	1	5	3	dog_bark
100263-2-0-117.wav	100263	58.5	62.5	1	5	2	children_playing
100263-2-0-121.wav	100263	60.5	64.5	1	5	2	children_playing
100263-2-0-126.wav	100263	63	67	1	5	2	children_playing
100263-2-0-137.wav	100263	68.5	72.5	1	5	2	children_playing

2.1.4 Class distributions

We check the counts of data from different classes

Class Names	Counts
children_playing	1000
dog_bark	1000
street_music	1000
jackhammer	1000
engine_idling	1000
air_conditioner	1000
drilling	1000
siren	929
car_horn	429
gun_shot	374

2.1.5 Observations

Here we can see the Class labels are unbalanced. Although 7 out of the 10 classes all have exactly 1000 samples, and siren is not far off with 929, the remaining two (car_horn, gun_shot) have significantly fewer samples at 43% and 37% respectively. So the data is unbalanced.

2.1.6 Folder distributions

Here we check the counts in each fold as mentioned in the dataset documentation we train data based on folds instead of Train_test_split.

Fold	**Counts**
4	990
5	936
3	925
2	888
1	873
7	838
10	837
6	823
9	816
8	806

2.1.7 Algorithms and Techniques

The proposed solution is to apply Deep Learning techniques that have proved very successful in the field of image classification in sound or audio classification.

First, we will extract Mel-Frequency Cepstral Coefficients (MFCC) [2] from the audio samples on a per-frame basis with a window size of a few milliseconds. MFCCs are coefficients that collectively make up Mel-frequency Cepstrum (MFC).MFCC summarizes the frequency distribution across the window size, so it is possible to analyze both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification.

The next step will be to train a Deep Neural Network with these data sets and make predictions. I believe that this will be very effective at finding patterns within the MFCC's much like they are effective at finding patterns within images.

Multi-layer perceptron's (MLP) is classed as a type of Deep Neural Network as they are composed of more than one layer of perceptron's and use non-linear activation which distinguishes them from linear perceptron's. Their architecture consists of an input layer, an output layer that ultimately makes a prediction about the input, and in-between the two layers there is an arbitrary number of hidden layers.

These hidden layers have no direct connection with the outside world and perform the model computations. The network is fed a labelled dataset (this being a form of supervised learning) of input-output pairs and is then trained to learn a correlation between those inputs and outputs. The training process involves adjusting the weights and biases within the perceptrons in the hidden layers in order to minimise the error.

The algorithm for training an MLP is known as Backpropagation. Starting with all weights in the network being randomly assigned, the inputs do a forward pass through the network and the decision of the output layer is measured against the ground truth of the labels you want to predict. Then the weights and biases are backpropagated back though the network where an optimisation method, typically Stochastic Gradient descent is used to adjust the weights so they will move one step closer to the error minimum on the next pass. The training phase will keep on performing this cycle on the network until the error can go no lower which is known as convergence.

Initially have trained the model in MLP only on features extracted from MFCC's I was not able to attain best results achieved by the bench mark. I have trained it on different features from chromagram and mel-scaled spectrogram.

Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave. Since, in music, notes exactly one octave apart are perceived as particularly similar, knowing the distribution of chroma even without the absolute frequency (i.e. the original octave) can give useful musical information about the audio and may even reveal perceived musical similarity that is not apparent in the original spectra.

The main routine Chromagram operates much like a spectrogram, taking an audio input and generating a sequence of short-time chroma frames (as columns of the resulting matrix). To obtain high-resolution Chroma profiles. We use implementations chromagram_E and chromagram_P, which use the simpler procedure of mapping each STFT bin directly to chroma, after selecting only spectral peaks in chromagram_P.Even when trained on MLP with more features not able to achieve the top bench mark scores.So we have trained on CNN.

Convolutional Neural Networks (CNNs) are built upon the architecture of MLPs but with a number of important changes. Firstly, the layers are organised into three dimensions, width, height and depth. Secondly, the nodes in one layer do not necessarily connect to all nodes in the subsequent layer, but often just a sub region of it.

This allows CNN to perform two important stages. The first being the feature extraction phase. Here a filter window slides over the input and extracts a sum of the convolution at each location which is then stored in the feature map. A pooling process is often included between CNN layers where typically the max value in each window is taken which decreases the feature map size but retains the significant data. This is important as it reduces the dimensionality of the network meaning it reduces both the training time and likelihood of overfitting. Then lastly we have the classification phase. This is where the 3D data within the network is flattened into a 1D vector to be output.

For the reasons discussed, both MLPs and CNN's typically make good classifiers, where CNN's, in particular, perform very well with image classification tasks due to their feature extraction and classification parts. I believe that this will be very effective at finding patterns within the MFCC's much like they are effective at finding patterns within images.

We will use the evaluation metrics described in earlier sections to compare the performance of these solutions against the benchmark models in the next section.

2.2 Benchmark Model

For the benchmark models, we will use the algorithms outlined in the paper "*A Dataset and Taxonomy for Urban Sound Research*" (Salamon, 2014) [3]. The paper describes five different algorithms with the following accuracies for an audio slice maximum at a duration of 4 seconds.

Algorithm	**Accuracy**
SVM_rbf	**68%**
RandomForest500	**66%**
IBk5	**55%**
J48	**48%**
ZeroR	**10%**

2.3 Evaluation Metrics

The evaluation metric for this problem is the Accuracy Score. Even though the data is unbalanced it should be fine as we will probably be fine as 90% of the data is balanced in 8 different classes as I have mentioned previously car_horn and gun_shot are the only classes which are around 43% and 35% when compared to other class's samples. This might create an accuracy paradox but even if we wrongly classifying 2 audio classes we can still have the good model as no single class does dominate the data as I have mentioned 90% of data is balanced among 8 classes other than car_horn and gun_shot.

3 Methodology

3.1 Data Preprocessing

3.1.1 Features

MFCC

As outlined in the proposal, we will extract Mel-Frequency Cepstral Coefficients (MFCC) from the audio samples.

The MFCC summarizes the frequency distribution across the window size, so it is possible to analyze both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification.

To get the feature we use the following function from Librosa:

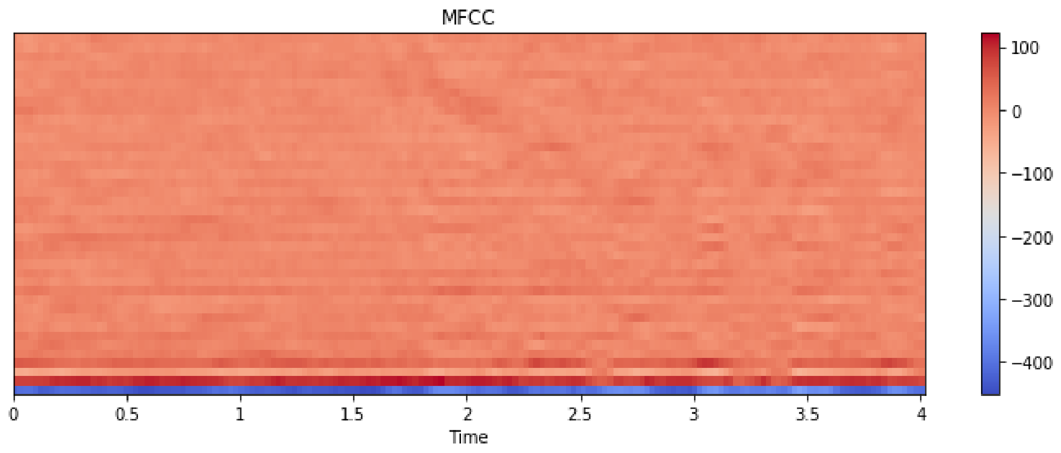
librosa.feature.mfcc(y, sr, n_mfcc=40.

The MFCC gives the features in the shape of (40, 173).

We use the display function to visualize MFCC:

librosa.display.specshow(feature_name, x_axis='time')

It can be visualized as follows using matplotlib



Mel spectrogram

Its uses the same Mel frequency in a scaled factor it is also called as Mel-Scaled Spectrogram

To get the feature we use the following function from Librosa:

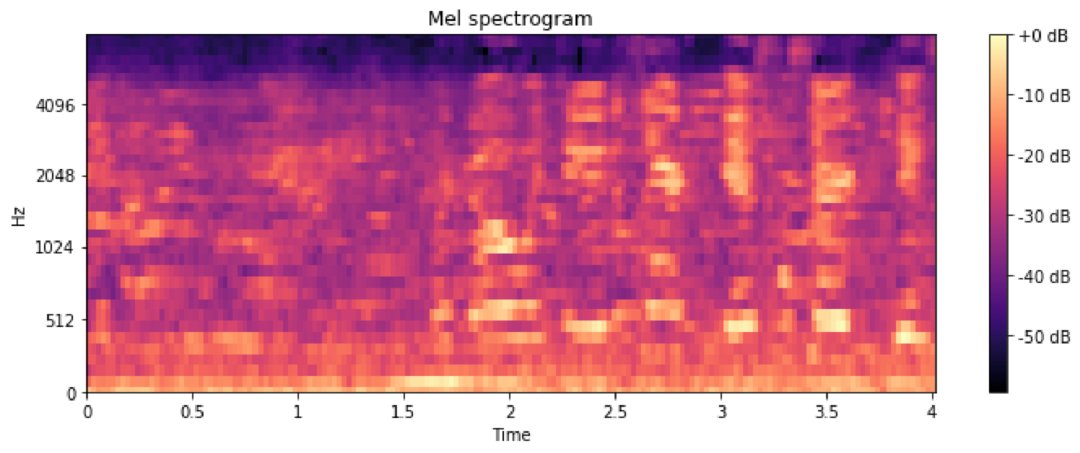
```
librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40,fmax=8000)
```

The melspectrogram gives the features in the shape of (40, 173).

We use the display function to visualize Mel Spectogram:

```
librosa.display.specshow(librosa.power_to_db(melspectrogram,ref=np.max),y_axis='mel',
fmax=8000,x_axis='time')
```

It can be visualized as follows using matplotlib



Chroma:

As discussed in the algorithms and Techniques, we will use features of Chroma.

Chroma STFT

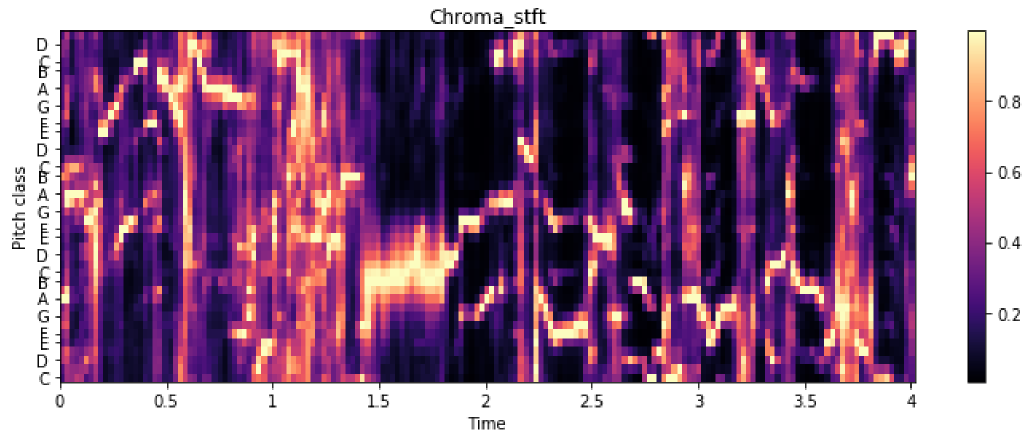
To get the feature we use the following function from Librosa:

chroma_stft=librosa.feature.chroma_stft(y=y, sr=sr,n_chroma=40)

The chroma_stft gives the features in shape of (40, 173).

We use the display function to visualize Chroma stft:

librosa.display.specshow(chroma_stft, y_axis='chroma', x_axis='time')



Chroma CQT:

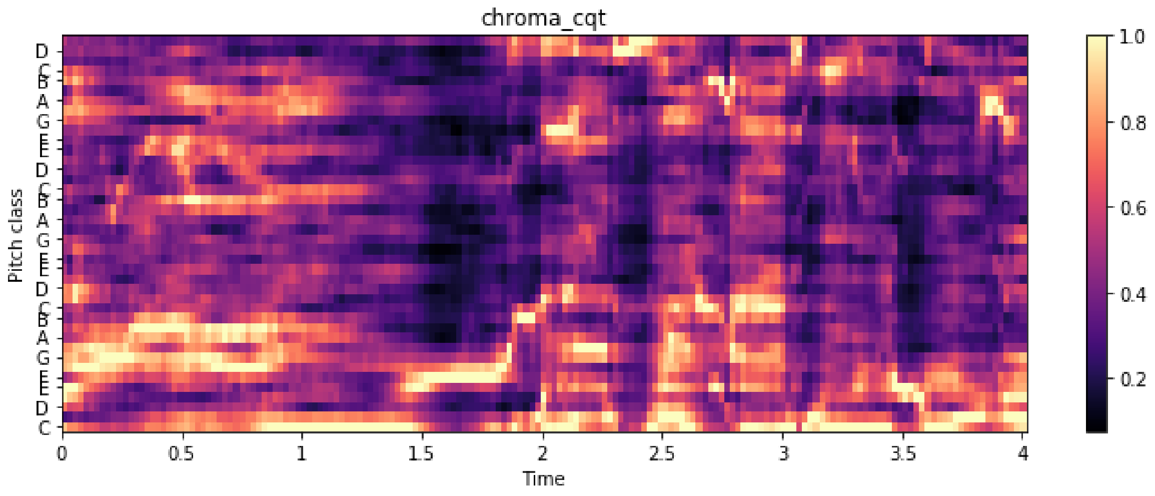
To get the feature we use the following function from Librosa:

chroma_cq =librosa.feature.chroma_cqt(y=y, sr=sr,n_chroma=40)

The chroma_cq gives the features in shape of (40, 173).

We use the display function to visualize Chroma CQT:

librosa.display.specshow(chroma_cq, y_axis='chroma', x_axis='time')



Chroma CENS:

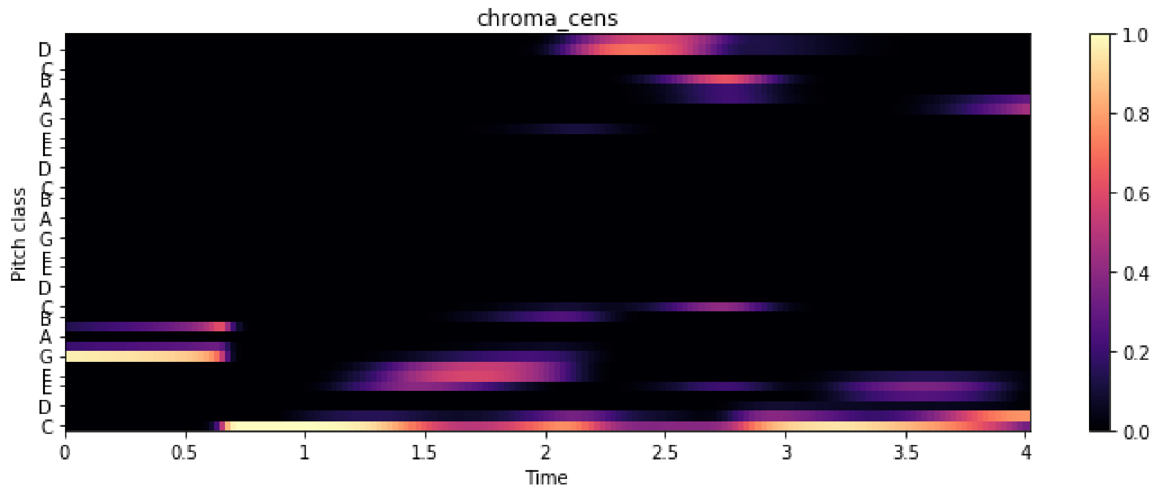
To get the feature we use the following function from Librosa:

chroma_cens =librosa.feature.chroma_cens(y=y, sr=sr,n_chroma=40)

The chroma_cq gives the features in shape of (40, 173).

We use the display function to visualize Chroma CENS:

librosa.display.specshow(chroma_cens, y_axis='chroma', x_axis='time')



3.1.2 Extracting features and data splitting

Extracting Features

To extract the features of all the data first we need to load the metadata into data frames using Pandas.

```
import pandas as pd
data=pd.read_csv("UrbanSound8K/metadata/UrbanSound8K.csv")
```

We extract the features mentioned above:

Initially, I have started with only one feature MFCC.
To get the features I have used this function

```
path="UrbanSound8K/audio/fold"
for i in tqdm(range(len(data))):
 fold_no=str(data.iloc[i]["fold"])
 file=data.iloc[i]["slice_file_name"]
 label=data.iloc[i]["classID"]
 filename=path+fold_no+"/"+file
 y,sr=librosa.load(filename)
 mfccs = np.mean(librosa.feature.mfcc(y, sr, n_mfcc=40).T,axis=0)
```

I wasn’t able to achieve the desired accuracy with the above model so I have decided to use all the features mentioned above.
So I have changed the function as follows.

```
path="UrbanSound8K/audio/fold"
for i in tqdm(range(len(data))):
   fold_no=str(data.iloc[i]["fold"])
   file=data.iloc[i]["slice_file_name"]
   label=data.iloc[i]["classID"]
   filename=path+fold_no+"/"+file
   y,sr=librosa.load(filename)
   mfccs = np.mean(librosa.feature.mfcc(y, sr, n_mfcc=40).T,axis=0)
   melspectrogram = np.mean(librosa.feature.melspectrogram(y=y, sr=sr,
n_mels=40,fmax=8000).T,axis=0)
   chroma_stft=np.mean(librosa.feature.chroma_stft(y=y, sr=sr,n_chroma=40).T,axis=0)
   chroma_cq = np.mean(librosa.feature.chroma_cqt(y=y, sr=sr,n_chroma=40).T,axis=0)
   chroma_cens = np.mean(librosa.feature.chroma_cens(y=y, sr=sr,n_chroma=40).T,axis=0)
   features=np.reshape(np.vstack((mfccs,melspectrogram,chroma_stft,chroma_cq,chroma_cens)),(40,5))
```

I have reshaped them and stacked them

Data Splitting

We split the data based on the folder I have chosen to use the data in folder 10 as the Training set.
To Split the data into training and testing set I have used an if Condition that add the features to x_test and adds the labels to y_test if the fold_no is not 10 and to x_train and y_train if fold_no is 10.

```
if(fold_no!='10'):
 x_train.append(features)
 y_train.append(label)
else:
 x_test.append(features)
 y_test.append(label)
```

3.2 Implemetation

3.2.1 Initial model Single Feature – MLP

We will start by constructing a Multilayer Perceptron (MLP) Neural Network using Keras and a Tensorflow backend.

Starting with a sequential model so we can build the model layer by layer.

We will begin with a simple model architecture, consisting of three layers, an input layer, a hidden layer, and an output layer. All three layers will be of the dense layer type which is a standard layer type that is used in many cases for neural networks.

The first layer will receive the input shape. As each sample contains 40 MFCCs (or columns) we have a shape of (1x40) this means we will start with an input shape of 40.

The first two layers will have 256 nodes. The activation function we will be using is the ReLU or Rectified Linear Activation. This activation function has been proved to work well in neural networks.

We will also apply a Dropout value of 50% on our first two layers. This will randomly exclude nodes from each update cycle, which in turn results in a network that is capable of better generalization and is less likely to over fit the training data.

Our output layer will have 10 nodes, which matches the number of possible classifications. The activation is for our output layer is softmax. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

```
from keras import Sequential
from keras.layers import Dense,Dropout,Activationmetrics
#forming model
model=Sequential()
#building the model
model.add(Dense(units=256,activation='relu',input_dim=40))
model.add(Dropout(0.5))
model.add(Dense(units=256,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=256,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=10,activation='softmax'))
```

3.2.2 Compiling the Single Feature MLP and Training

For compiling our model, we will use the following three parameters:

- Loss function - we will use categorical_crossentropy. This is the most common choice for classification. A lower score indicates that the model is performing better.
- Metrics - we will use the accuracy metric which will allow us to view the accuracy score on the validation data when we train the model.
- Optimizer - here we will use adam which is a generally good optimizer for many use cases.

```
# Compile the model
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
from keras.callbacks import ModelCheckpoint
model.fit(x_train,y_train,epochs=50,validation_data=(x_test,y_test),batch_size=50,callbacks=[ModelCheckpoint(filepath='Saved_models/basic_mlp.hdf5', verbose=1, save_best_only=True)],
verbose=1)
```

```
Epoch 00006: val_loss did not improve from 1.29163
Epoch 7/50
7895/7895 [==============================] - ETA: 1s - loss: 0.5714 - acc: 0.820 - ETA: 0s - loss: 0.7178 -
acc: 0.762 - ETA: 0s - loss: 0.7417 - acc: 0.747 - ETA: 0s - loss: 0.7389 - acc: 0.743 - ETA: 0s - loss: 0.7252 - acc:
0.754 - ETA: 0s - loss: 0.7410 - acc: 0.756 - ETA: 0s - loss: 0.7466 - acc: 0.753 - ETA: 0s - loss: 0.7343 - acc: 0.75
7 - ETA: 0s - loss: 0.7331 - acc: 0.758 - ETA: 0s - loss: 0.7301 - acc: 0.757 - ETA: 0s - loss: 0.7461 - acc: 0.751 - E
TA: 0s - loss: 0.7478 - acc: 0.752 - ETA: 0s - loss: 0.7462 - acc: 0.752 - ETA: 0s - loss: 0.7465 - acc: 0.753 - ETA: 0
s - loss: 0.7472 - acc: 0.752 - ETA: 0s - loss: 0.7408 - acc: 0.754 - ETA: 0s - loss: 0.7381 - acc: 0.755 - ETA: 0s - lo
ss: 0.7386 - acc: 0.757 - ETA: 0s - loss: 0.7394 - acc: 0.757 - ETA: 0s - loss: 0.7407 - acc: 0.757 - ETA: 0s - loss: 0.
7422 - acc: 0.757 - ETA: 0s - loss: 0.7401 - acc: 0.757 - ETA: 0s - loss: 0.7401 - acc: 0.757 - 1s 163us/step - loss:
0.7419 - acc: 0.7567 - val_loss: 1.2697 - val_acc: 0.5998

Epoch 00007: val_loss improved from 1.29163 to 1.26970, saving model to Saved_models/basic_mlp.hdf5
Epoch 8/50
7895/7895 [==============================] - ETA: 1s - loss: 0.6268 - acc: 0.760 - ETA: 1s - loss: 0.6287 -
acc: 0.760 - ETA: 1s - loss: 0.6546 - acc: 0.763 - ETA: 1s - loss: 0.6242 - acc: 0.774 - ETA: 1s - loss: 0.6414 - acc:
0.767 - ETA: 0s - loss: 0.6623 - acc: 0.762 - ETA: 0s - loss: 0.6804 - acc: 0.759 - ETA: 0s - loss: 0.6717 - acc: 0.76
4 - ETA: 0s - loss: 0.6843 - acc: 0.765 - ETA: 0s - loss: 0.6925 - acc: 0.763 - ETA: 0s - loss: 0.6914 - acc: 0.765 - E
TA: 0s - loss: 0.6904 - acc: 0.766 - ETA: 0s - loss: 0.6916 - acc: 0.766 - ETA: 0s - loss: 0.7031 - acc: 0.765 - ETA: 0
s - loss: 0.7125 - acc: 0.762 - ETA: 0s - loss: 0.7148 - acc: 0.760 - ETA: 0s - loss: 0.7114 - acc: 0.762 - ETA: 0s - lo
ss: 0.7084 - acc: 0.763 - ETA: 0s - loss: 0.7138 - acc: 0.762 - ETA: 0s - loss: 0.7152 - acc: 0.761 - ETA: 0s - loss: 0.
7165 - acc: 0.761 - ETA: 0s - loss: 0.7150 - acc: 0.762 - 1s 149us/step - loss: 0.7165 - acc: 0.7624 - val_loss: 1.2
696 - val_acc: 0.5842
```

```
# Display model architecture summary
model.summary()
```

```
Layer (type)              Output Shape            Param #
=================================================================
dense_9 (Dense)             (None, 256)             10496
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)             0
_________________________________________________________________
dense_10 (Dense)             (None, 256)             65792
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)             0
_________________________________________________________________
dense_11 (Dense)             (None, 256)             65792
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)             0
_________________________________________________________________
dense_12 (Dense)             (None, 10)              2570
=================================================================
Total params: 144,650
Trainable params: 144,650
Non-trainable params: 0
_________________________________________________________________
7895/7895 [==============================] - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 0s 46us/s
tep
837/837 [==============================] - ETA:  - 0s 40us/step
```

3.2.3 Testing the model and Calculating Accuracy

Here we verify the Accuracty of the model over the test data set

```
# Calculate training accuracy
score_train = model.evaluate(x_train, y_train, verbose=1)
score_test = model.evaluate(x_test, y_test, verbose=1)
accuracy_train = 100*score_train[1]
accuracy_test = 100*score_test[1]

print("Training accuracy: %.4f%%" % accuracy_train)
print("Test accuracy: %.4f%%" % accuracy_test)
```

```
Training accuracy: 91.6909%
Test accuracy: 59.7372%
```

3.2.3 Prediction and Validation

To Extract the features we use the function extract_features where we get the features

```
def extract_features(file_path):
   audio,sample_rate=librosa.load(file_path)
   mfccs = np.mean(librosa.feature.mfcc(audio, sample_rate, n_mfcc=40).T,axis=0)
   mfccs=np.reshape(mfccs,(1,mfccs.shape[0]))
   return mfccs
```

In Print_prediction function we predict the class and display the class that has been predicted

```
#Importing the necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
from librosa import display
import librosa
class_lable=["air_conditioner","car_horn","children_playing","dog_bark","drilling","engine_idling","gun_shot
","jackhammer","siren","street_music"]
def print_prediction(file_path):
   features = extract_features(file_path)
   predicted_vector = model.predict_classes(features)
   predicted_class = class_lable[predicted_vector[0]]
   print("The predicted class is:", predicted_class, '\n')
```

I have verified the data on some of the test data from fold_no 10

```
 file_path='UrbanSound8K/audio/fold5/100032-3-0-0.wav'
 print_prediction(file_path)
```

```
The predicted class is: dog_bark
```

```
 file_path='UrbanSound8K/audio/fold7/101848-9-0-0.wav'
 print_prediction(file_path)
```

```
The predicted class is: street_music
```

```
 file_path='UrbanSound8K/audio/fold10/200460-6-1-0.wav'
 print_prediction(file_path)
```

```
The predicted class is: dog_bark
```

The model has predicted the gun_shot as a dog bark

I have tested the data with some more audio from https://freesound.org/, which was mentioned in the Dataset document

```
file_path='Sample Auido/Car_horn.wav'
print_prediction(file_path)
```

The predicted class is: street_music

The model has predicted Car horn as music

```
file_path='Sample Auido/dog_bark.wav'
print_prediction(file_path)
```

The predicted class is: dog_bark

```
file_path='Sample Auido/drilling.wav'
print_prediction(file_path)
```

The predicted class is: drilling

```
file_path='Sample Auido/siren.wav'
print_prediction(file_path)
```

The predicted class is: siren

```
file_path='Sample Auido/street_music.wav'
print_prediction(file_path)
```

The predicted class is: street_music

```
file_path='Sample Auido/engine.wav'
print_prediction(file_path)
```

The predicted class is: engine_idling

3.3 Model Refinement:

Since the Accuracy was not upto the Benchmark I have chosen to use all the features I have discussed using MLP. In the Report, I will call it as Multi Feature MLP.

We have trained the model in a similar way as the Single Feature MLP but with multiple features.

We have seen a better accuracy:

Training accuracy: 94.9335%
Test accuracy: 64.8746%

```
def extract_features(file_path):
    y,sr=librosa.load(file_path)
    mfccs = np.mean(librosa.feature.mfcc(y, sr, n_mfcc=40).T,axis=0)
    melspectrogram = np.mean(librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40,fmax=8000).T,axis=0)
    chroma_stft=np.mean(librosa.feature.chroma_stft(y=y, sr=sr,n_chroma=40).T,axis=0)
    chroma_cq = np.mean(librosa.feature.chroma_cqt(y=y, sr=sr,n_chroma=40).T,axis=0)
    chroma_cens = np.mean(librosa.feature.chroma_cens(y=y, sr=sr,n_chroma=40).T,axis=0)
    features=np.reshape(np.vstack((mfccs,melspectrogram,chroma_stft,chroma_cq,chroma_cens)),(1,200))
    #print(features.shape)
    return features]

class_lable=["air_conditioner","car_horn","children_playing","dog_bark","drilling","engine_idling","gun_shot","jac
khammer","siren","street_music"]
def print_prediction(file_path):
    features = extract_features(file_path)
    predicted_vector = model.predict_classes(features)
    predicted_class = class_lable[predicted_vector[0]]
    print("The predicted class is:", predicted_class, '\n')
```

We have tested and validated it on the samples that we have used above:

```
file_path='UrbanSound8K/audio/fold5/100032-3-0-0.wav'
print_prediction(file_path)
(1, 200)

The predicted class is: dog_bark

file_path='UrbanSound8K/audio/fold7/101848-9-0-0.wav'
print_prediction(file_path)
The predicted class is: street_music

file_path='UrbanSound8K/audio/fold10/200460-6-1-0.wav'
print_prediction(file_path)
The predicted class is: gun_shot
```

We were able to predict the gun_shot accurately in this model

```
file_path='Sample Auido/Car_horn.wav'
print_prediction(file_path)
The predicted class is: street_music

file_path='Sample Auido/dog_bark.wav'
print_prediction(file_path)
The predicted class is: dog_bark
```

```
file_path='Sample Auido/drilling.wav'
print_prediction(file_path)
The predicted class is: drilling

file_path='Sample Auido/engine.wav'
print_prediction(file_path)
The predicted class is: engine_idling
```

Multi Feature CNN:

Now, further refining model, we will check if using Convenutional Neural Networks we will be able to get better Results.
We reshape the features that we extracted so that they can be used by the CNN model.

As we get 200 features 40 from each of 5 different spectrograms we have the shape of (40,5) when the features are stacked and Reshaped.

We reshape the Features so that they can be used by the

```
#reshaping to shape required by CNN
x_train=np.reshape(x_train,(x_train.shape[0], 40,5,1))
x_test=np.reshape(x_test,(x_test.shape[0], 40,5,1))
x_train.shape, x_test.shape
((7895, 40, 5, 1), (837, 40, 5, 1))
```

I have Trained the model on two convolutional 2D layers with 64 filters and 128 filters respectively both of the layers used the relu activation function

I have used two Max Pooling 2D layers after each convolutional layer

Then I have added dropout of 30% before adding the 2 more dense layers with relu activation functions and 256 and 512 nodes respectively

Then I have added one mode Dropout layer of 30%

Finally for the Output layer with 10 nodes and a softmax activation function.

```
from keras import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,Dropout
#forming model
model=Sequential()
#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"))
```

Compiling the multi Feature MLP and Training it.

```
#compiling
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"])

from keras.callbacks import ModelCheckpoint
#fitting
model.fit(x_train,y_train,epochs=50,validation_data=(x_test,y_test),batch_size=50,callbacks=[ModelCheckpoint(filepath='Saved_models/multi_CNN.hdf5', verbose=1, save_best_only=True)], verbose=1)
```

```
Epoch 00034: val_loss did not improve from 0.85690
Epoch 35/50
7895/7895 [==============================] - 16s 2ms/step - loss: 0.2146 -
 acc: 0.9322 - val_loss: 1.2673 - val_acc: 0.6965

Epoch 00035: val_loss did not improve from 0.85690
Epoch 36/50
7895/7895 [==============================] - 16s 2ms/step - loss: 0.2415 -
 acc: 0.9246 - val_loss: 1.2265 - val_acc: 0.6989

Epoch 00036: val_loss did not improve from 0.85690
Epoch 37/50
7895/7895 [==============================] - 16s 2ms/step - loss: 0.2454 -
 acc: 0.9217 - val_loss: 1.2453 - val_acc: 0.6822
```

model.summary()

```
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 40, 5, 64)         1664      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 20, 3, 64)         0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 3, 128)        204928    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 2, 128)        0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 10, 2, 128)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 2560)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 256)               655616    
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 512)               131584    
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130      
=================================================================
Total params: 998,922
Trainable params: 998,922
Non-trainable params: 0
_________________________________________________________________
7895/7895 [==============================] - 3s 366us/step
837/837 [==============================] - 0s 341us/step
```

Accuracy:

```
Training accuracy: 98.9614%
Test accuracy: 72.2820%
```

```
import librosa
def extract_features(file_path):
    y,sr=librosa.load(file_path)
    mfccs = np.mean(librosa.feature.mfcc(y, sr, n_mfcc=40).T,axis=0)
    melspectrogram = np.mean(librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40,fmax=8000).T,axis=0)
    chroma_stft=np.mean(librosa.feature.chroma_stft(y=y, sr=sr,n_chroma=40).T,axis=0)
    chroma_cq = np.mean(librosa.feature.chroma_cqt(y=y, sr=sr,n_chroma=40).T,axis=0)
    chroma_cens = np.mean(librosa.feature.chroma_cens(y=y, sr=sr,n_chroma=40).T,axis=0)
    features=np.reshape(np.vstack((mfccs,melspectrogram,chroma_stft,chroma_cq,chroma_cens)),(1,40,5,1))
    #print(features.shape)
    return features
```

```
class_lable=["air_conditioner","car_horn","children_playing","dog_bark","drilling","engine_idling","gun_shot","jac
khammer","siren","street_music"]
def print_prediction(file_path):
    features = extract_features(file_path)
    predicted_vector = model.predict_classes(features)
    predicted_class = class_lable[predicted_vector[0]]
    print("The predicted class is:", predicted_class, '\n')

file_path='UrbanSound8K/audio/fold10/200460-6-1-0.wav'
print_prediction(file_path)
The predicted class is: gun_shot

file_path='Sample Auido/Car_horn.wav'
print_prediction(file_path)
The predicted class is: street_music

file_path='Sample Auido/dog_bark.wav'
print_prediction(file_path)
The predicted class is: dog_bark

file_path='Sample Auido/drilling.wav'
print_prediction(file_path)
The predicted class is: drilling

file_path='Sample Auido/street_music.wav'
print_prediction(file_path)
The predicted class is: street_music

file_path='Sample Auido/engine.wav'
print_prediction(file_path)
The predicted class is: engine_idling
```

4 Results

4.1 Model Evaluation

During the model development phase, the validation data was used to evaluate the model. The final model architecture and hyperparameters were chosen because they performed the best among the tried combinations. As we can see from the validation work in the previous section, to verify the robustness of the final model, a test was conducted using copyright free sounds from sourced from the internet. The following observations are based on the results of the test:

- The model sometimes misclassifies Drilling with Jackhammer and Car_horn with Street music.
- The Model performs well with New Data.

4.2 Justification

As you can see in the below table the Final model has a better accuracy of 72% than the Benchmark Models provided which had the highest accuracy of 68%.

Model	Classification Accuracy
Multi Feature CNN	72%
Benchmark SVM_rbf	68%
Multi Feature MLP	64%
Single Feature MLP	59%

5 Conclusion

5.1 Free-Form Visualization

I observed that it is difficult to visualize the difference between some of the classes. In particular, the following sub-groups are similar in shape:

- Repetitive sounds for air conditioner, drilling, engine idling, and jackhammer.
- Sharp peaks for dog barking and gun shot.
- A similar pattern for children playing and street music.

But during the Validation I was able to observe that:

- The model sometimes misclassifies Drilling with Jackhammer
- It misclassifies Car horn with Street music sometimes.
- Its misclassifies air_conditioner with Drilling sometimes

5.2 Reflection

The process used for this project can be summarized with the following steps:

1. The initial problem has been defined and relevant public dataset was located.
2. The data is analyzed and explored.
3. The features were extracted from the data using Librosa.
4. An initial model was trained and evaluated on MLP with only MFCC.
5. A further model was trained and refined with more features from Librosa using MLP.
6. The final model was built and evaluated on with features from Librosa using CNN's.

5.3 Improvement

- We need to make it work with Real-time audio
- We can improve the model and make it work as Voice recognition
- We can make enhancements to that model so that it can clear the background noise
  So that we can get a better model with great accuracy

References

[1] Justin Salamon, Christopher Jacoby and Juan Pablo Bello, "Urban Sound Datasets", "UrbanSound8K" https://urbansounddataset.weebly.com/urbansound8k.html

[2] Mel-frequency cepstrum Wikipedia page https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

[3] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research" http://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf

[4] S. Chaudhuri and B. Raj. Unsupervised hierarchical structure induction for deeper semantic analysis of audio. In IEEE ICASSP, 2013.

[5] Sounds from Free audio samples https://freesound.org/

[6] Audio Classification using FastAI and On-the-Fly Frequency Transforms https://towardsdatascience.com/audio-classification-using-fastai-and-on-the-fly-frequency-transforms-4dbe1b540f89

[7] Librosa Tutorial and Features https://librosa.github.io/librosa/feature.html

[8] Urban Sound Classification — Part 1: sound wave, digital audio signal https://towardsdatascience.com/urban-sound-classification-part-1-99137c6335f9

[9] Urban Sound Classification — Part 2: sample rate conversion, Librosa https://towardsdatascience.com/urban-sound-classification-part-2-sample-rate-conversion-librosa-ba7bc88f209a