Springboard Data Science Career Track Capstone Project

Classifying Urban sounds *using* Deep Learning

Final Report

Data Exploration and Visualisation link:
https://github.com/xwu0223/Urban-Sound-Classification-using-Deep-Learning/blob/master/Data_Exploration_visualization_preprocessing.ipynb

MLP Model Training and Evaluation:
https://github.com/xwu0223/Urban-Sound-Classification-using-Deep-Learning/blob/master/Model_Training_Evaluation.ipynb

CNN Model Training and Evaluation:
https://github.com/xwu0223/Urban-Sound-Classification-using-Deep-Learning/blob/master/CNN_Model.ipynb

Presentation:
https://github.com/xwu0223/Urban-Sound-Classification-using-Deep-Learning/blob/master/Urban_Sound_Classification.pptm

Xingkai Wu

Oct 09 2019

Problem Statement:

The objective of this project will be to use Deep Learning techniques to classify urban sounds.

When given an audio sample in a wav format 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.

To achieve this, we plan on using different neural network architectures such as Multilayer Per- ceptrons (MLPs) and Convolutional Neural Networks (CNNs).

Analysis:

Data Exploration and Visualisation

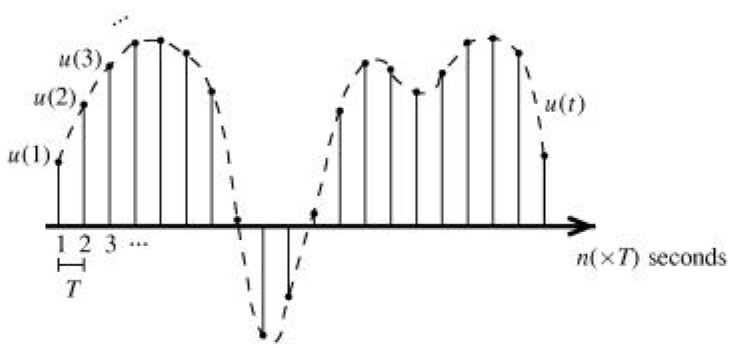
The Dataset:

This dataset contains 8732 labeled sound excerpts (<=4s) of urban sounds from 10 classes:

air_conditioner	car_horn	Children_playing	dog_bark	drilling
engine_idling	gun_shot	jackhammer	Siren	street_music

Wav file is stored in a way such that the sampling rate is 44.1kHz(44100 times per second).
Each sample is the amplitude of the wave at 1/44100 second=22.67us of interval, where the bit depth determines how detailed the sample will be also known as the dynamic range of the signal. For example, 16bit range means a sample’s has 2^16=65536 amplitude values with equal amplitude interval.

The wav sound signal is discrete time signal, the following image is an example of discrete time signal

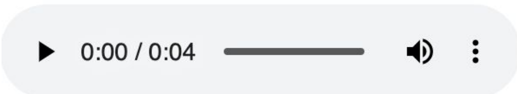


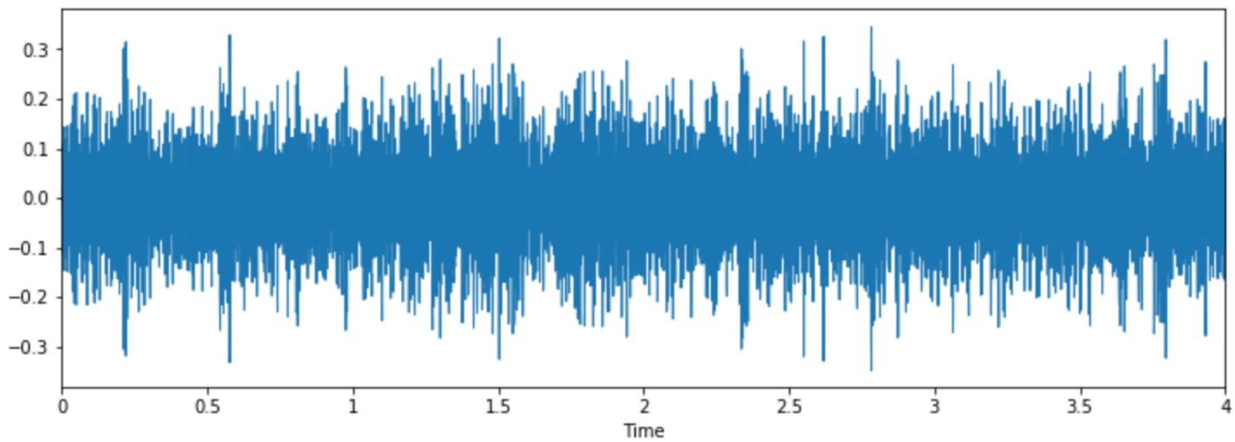
Therefore, the sound signal can be analyzed in one- dimensional array or vector of amplitude value.

The following libraries are used to explore the audio signal in python:
Ipython.display.Audio: This package allows us to play audio directly in the Jupyter Notebook.
Librosa: This package allows us to load audio to notebook as a numpy array for analysis and manipulation.

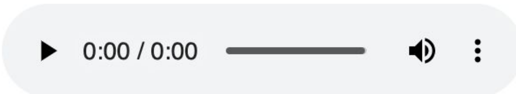
The following images are the 10 different classes in the file directory.

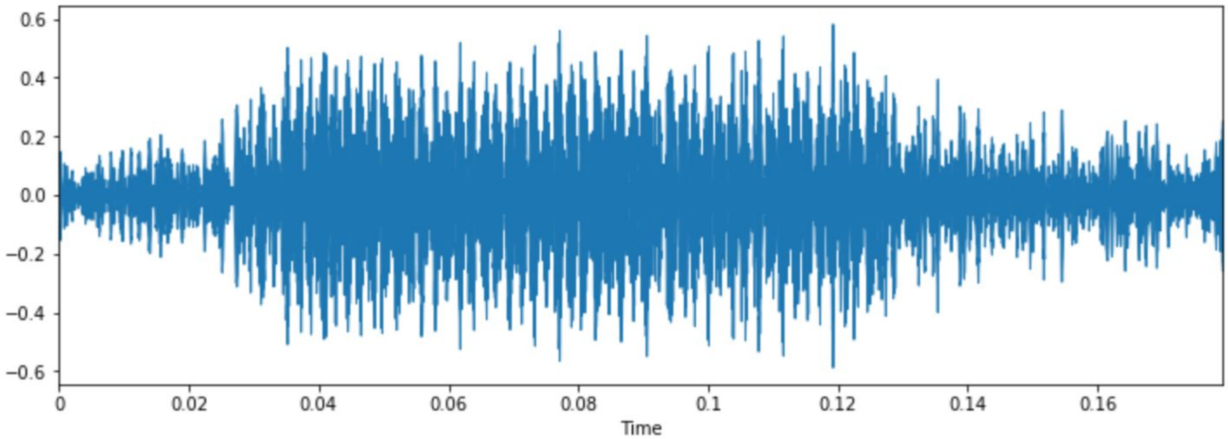
```
# Class: Air Conditioner

filename = '../Capstone_Project_2/train/Train/24.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```



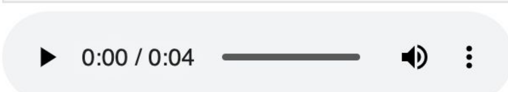


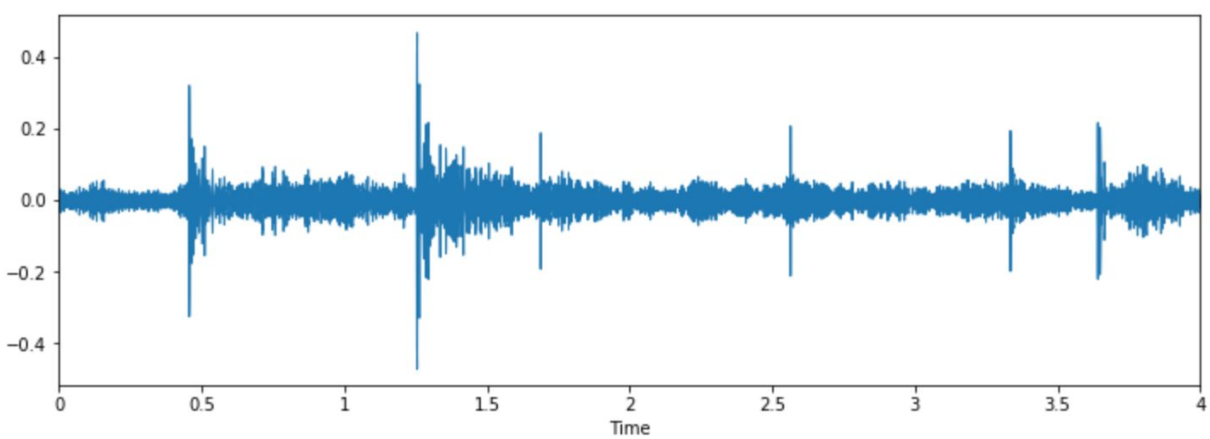
```
# Class: Car Horn

filename = '../Capstone_Project_2/train/Train/48.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```



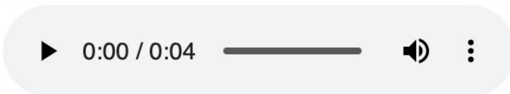


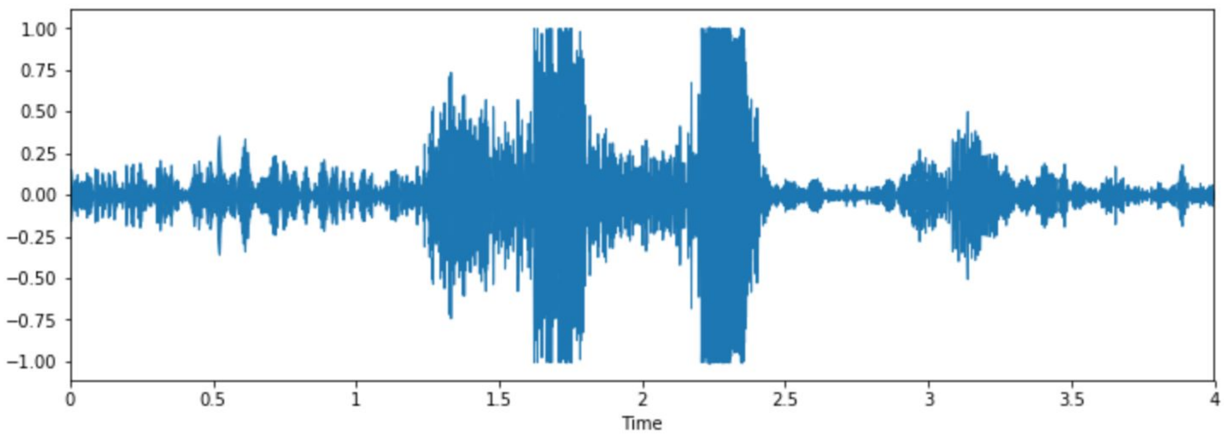

```
# Class:Children Playing

filename = '../Capstone_Project_2/train/Train/6.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```



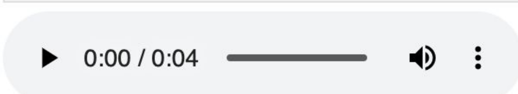


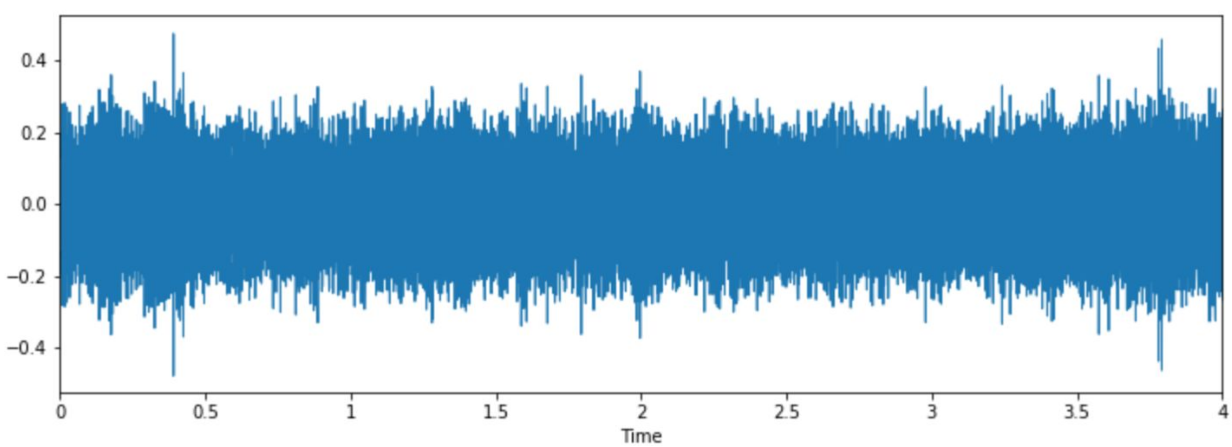
```python
# Class: Dog Bark

filename = '../Capstone_Project_2/train/Train/4.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```





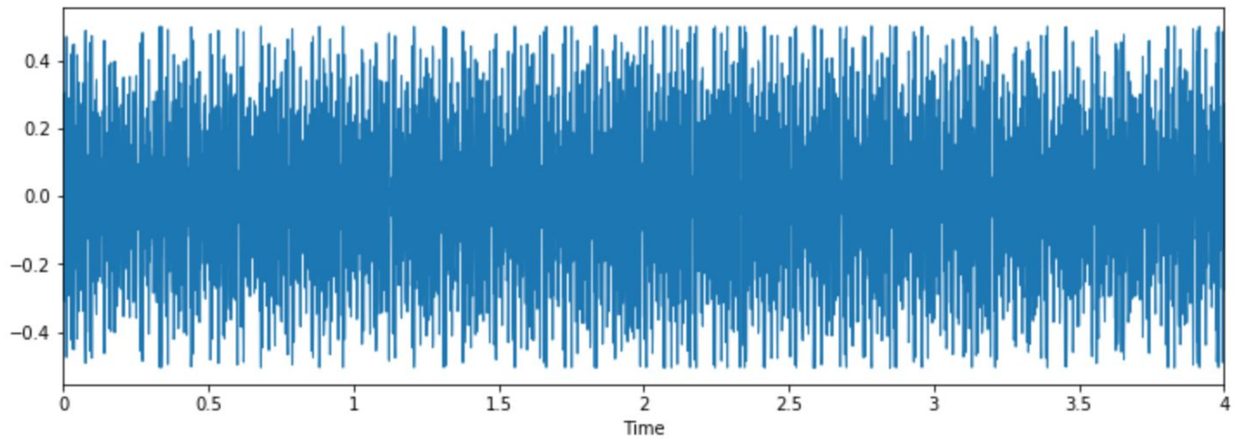

```python
# Class: Drilling

filename = '../Capstone_Project_2/train/Train/11.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```





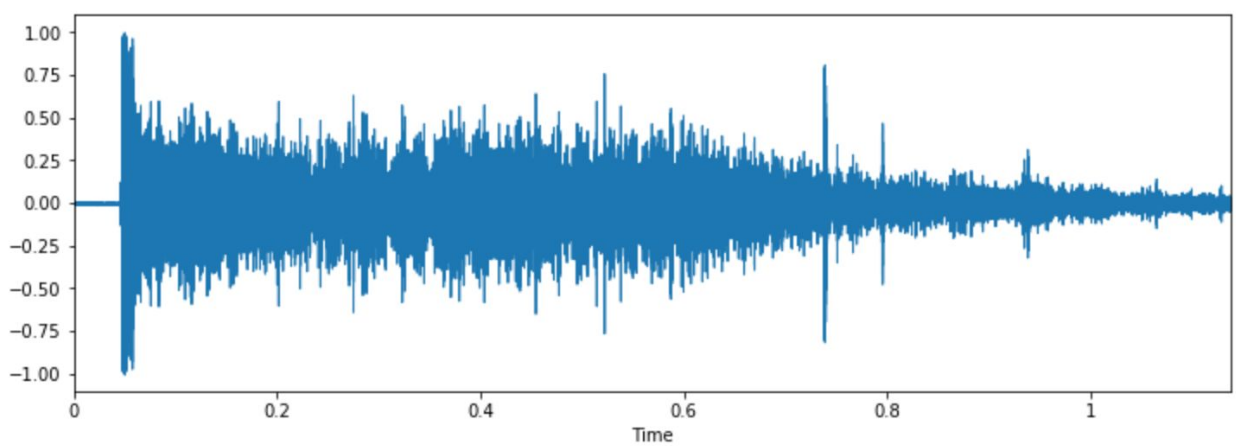
```
# Class: Engine Idling

filename = '../Capstone_Project_2/train/Train/17.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

0:00 / 0:04

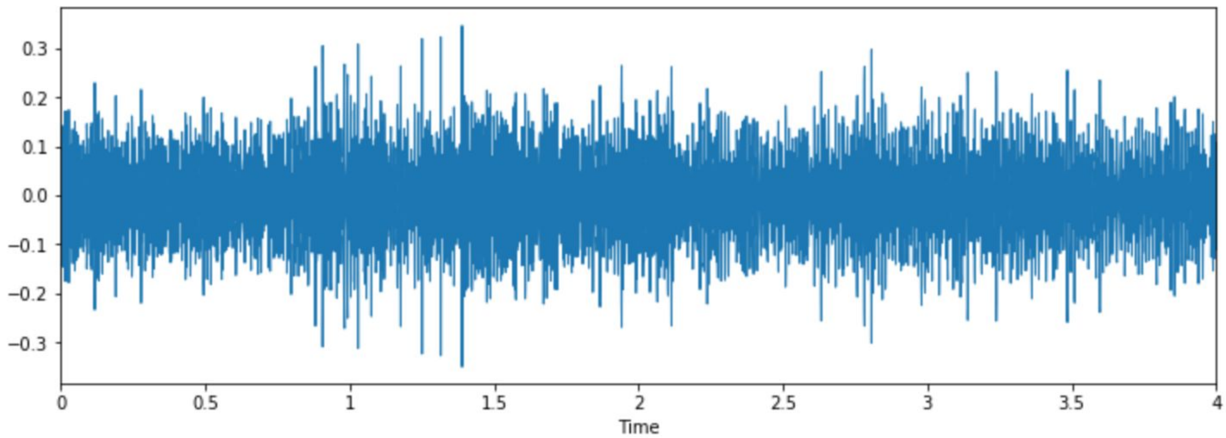



```
# Class: Gun Shot

filename = '../Capstone_Project_2/train/Train/12.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

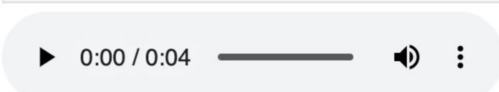
0:00 / 0:01

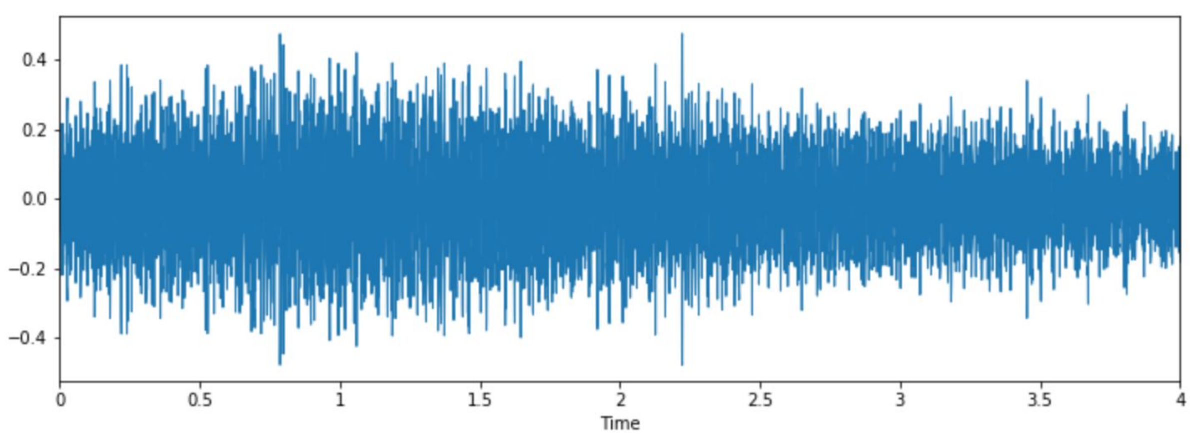


```python
# Class: Jackhammer

filename = '../Capstone_Project_2/train/Train/33.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

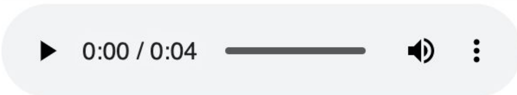


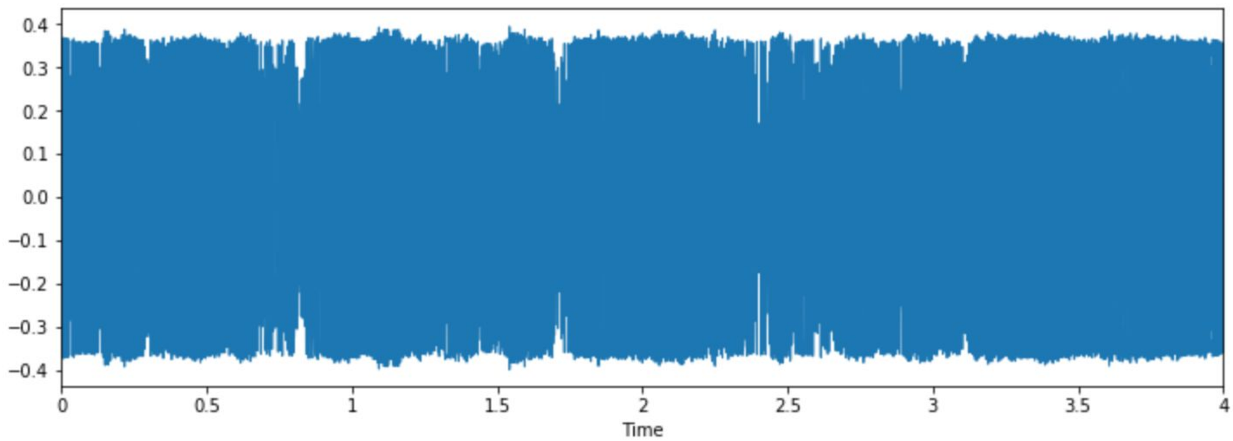

```python
# Class: Siren

filename = '../Capstone_Project_2/train/Train/0.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```





```python
# Class: Street Music

filename = '../Capstone_Project_2/train/Train/1.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```





Visually, it is very trick to tell the differences between the classes. Especially the air conditioner, drilling, engine idling and jackhammer are similar in shape.
Likewise, the dog bark and gunshot are similar as they both have have peaks. Children playing and street music are also very close in shape.

The Metadata Set:

```
import pandas as pd
metadata = pd.read_csv('../Capstone_Project_2/train/train.csv')
metadata.head()
```

	ID	Class
0	0	siren
1	1	street_music
2	2	drilling
3	3	siren
4	4	dog_bark

The following table shows the count for each class in the dataset. The dataset is **not** balanced.

```
print(metadata['Class'].value_counts())
```

```
jackhammer          668
engine_idling       624
siren               607
drilling            600
air_conditioner     600
children_playing    600
dog_bark            600
street_music        600
car_horn            306
gun_shot            230
Name: Class, dtype: int64
```

It's obvious to tell that the ID number is the prefix of the audio file name, let's merge the the audio file name and the metadata set.

```
df_merge.head()
```

	File_Name	ID	Class
0	4666.wav	4666	engine_idling
1	2217.wav	2217	engine_idling
2	7409.wav	7409	drilling
3	1078.wav	1078	car_horn
4	6717.wav	6717	children_playing

Audio file properties:

All the wav file has following 3 properties

1. Sample rate
2. Number of channels
3. Bit depth

```
from pathlib import Path
import soundfile as sf
data_folder = Path("../Capstone_Project_2/train/Train/")
sample_rate = []
num_channel = []
bit_depth = []
for name in df_merge['File_Name']:
    ob = sf.SoundFile(data_folder / name)
    sample_rate.append(ob.samplerate)
    num_channel.append(ob.channels)
    bit_depth.append(ob.subtype)
```

```
sound_file = pd.DataFrame(
    {'File_Name': file,
     'sample_rate': sample_rate,
     'num_channel': num_channel,
     'bit_depth':bit_depth
    })
```

```
combined = df_merge.merge(sound_file,on='File_Name')
```

```
combined.head()
```

	File_Name	ID	Class	sample_rate	num_channel	bit_depth
0	4666.wav	4666	engine_idling	44100	2	PCM_16
1	2217.wav	2217	engine_idling	48000	1	PCM_24
2	7409.wav	7409	drilling	44100	2	PCM_16
3	1078.wav	1078	car_horn	44100	2	PCM_16
4	6717.wav	6717	children_playing	44100	2	PCM_16

```
print(combined['bit_depth'].value_counts(normalize=True))
```

```
PCM_16       0.667341
PCM_24       0.302668
FLOAT        0.021711
PCM_U8       0.007912
MS_ADPCM     0.000184
IMA_ADPCM    0.000184
Name: bit_depth, dtype: float64
```

```
print(combined['sample_rate'].value_counts(normalize=True))
```

```
44100    0.598896
48000    0.305980
96000    0.066053
24000    0.009752
16000    0.007176
22050    0.006624
11025    0.003680
8000     0.001104
32000    0.000736
Name: sample_rate, dtype: float64
```

Algorithms and Techniques:
The proposed solution to this problem is to apply Deep Learning techniques that have proved to be highly successful in the field of image classification.

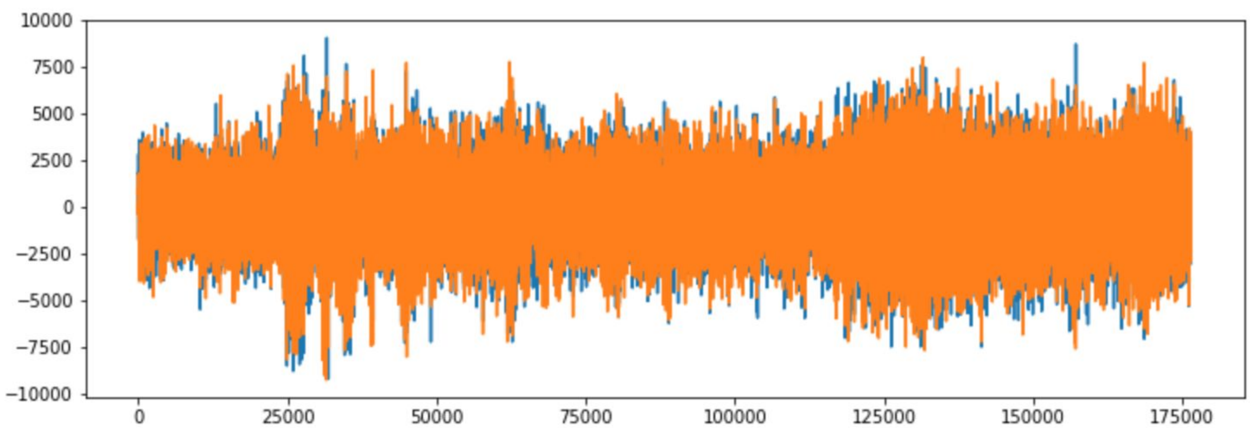
First we will extract Mel-Frequency Cepstral Coefficients (MFCC) from the audio samples on a per-frame basis with a window size of a few milliseconds. The MFCC summarises the frequency distribution across the window size, so it is possible to analyse both the frequency and time characteristics of the 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. We will begin by using a simple neural network architectures, such as Multi-Layer Perceptron before experimenting with more complex architectures such as Convolutional Neural Networks.

```
import librosa
from scipy.io import wavfile as wav
import numpy as np

filename = '../Capstone_Project_2/train/Train/4666.wav'

librosa_audio, librosa_sample_rate = librosa.load(filename)
scipy_sample_rate, scipy_audio = wav.read(filename)

print('Original sample rate:', scipy_sample_rate)
print('Librosa sample rate:', librosa_sample_rate)
```

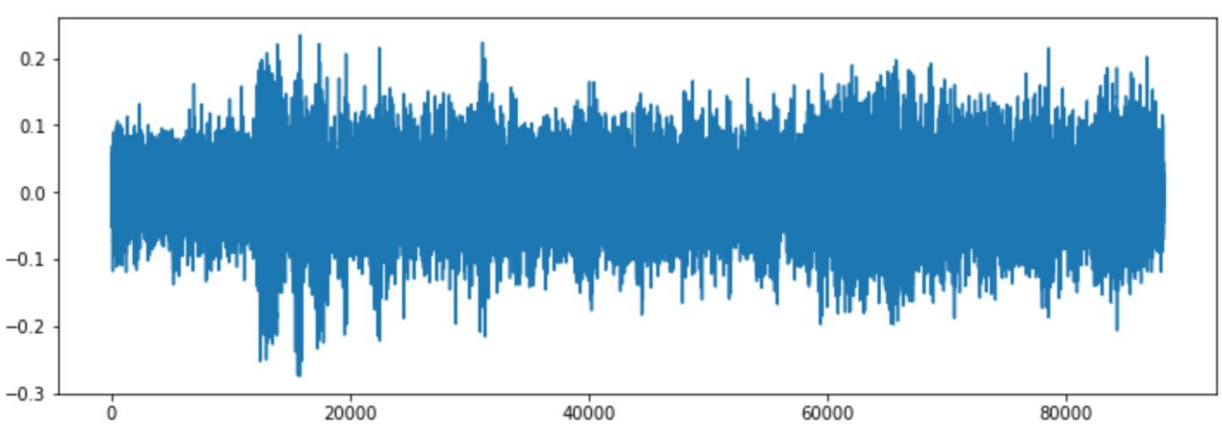
```
Original sample rate: 44100
Librosa sample rate: 22050
```

```
The original audio clip 4666.wav has 2 channel, the first plot illustrate the 2 seperate channels, the second plot
illustrated the merged channel using librosa.
Librosa converts the audio signal to mono, meaning the number of channels will always be 1.
```




```
plt.figure(figsize=(12, 4))
plt.plot(librosa_audio)
```

```
[<matplotlib.lines.Line2D at 0x1c3c2cce48>]
```



We can’t tell the difference much in term of shape of the signal of 2 channels and combined into 1.

MFCC(Mel- frequency Cepstrum Coefficient) is a function to extract the coefficients of Mel- Frequency Cepstrum which is a representation of a short term power spectrum of a sound based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

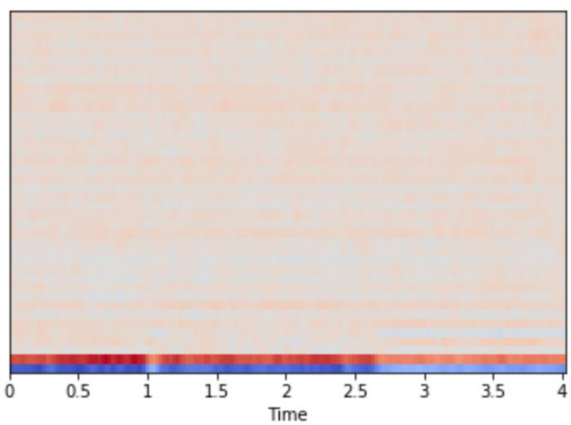
```
mfccs = librosa.feature.mfcc(y=librosa_audio, sr=librosa_sample_rate, n_mfcc=40)
print(mfccs.shape)
```

```
(40, 173)
```

```
This shows the merged channel clip of 4666.wav calculated a series of 40 MFCCs over 173 frames.
```

```
import librosa.display
librosa.display.specshow(mfccs, sr=librosa_sample_rate, x_axis='time')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c3c23bac8>
```




```
This plot illustrates the sectrum of the merged channel clip 4666.wav
```

Let us extract the 40 feature coefficients(MFCC) of each audio signal:

```
# Set the path to the full UrbanSound dataset
fulldatasetpath = '../Capstone_Project_2/train/Train/'

metadata = pd.read_csv('../Capstone_Project_2/train_detail.csv')

features = []

# Iterate through each sound file and extract the features
for index, row in metadata.iterrows():

    file_name = os.path.join(os.path.abspath(fulldatasetpath)+'/',str(row["File_Name"]))

    class_label = row["Class"]
    data = extract_features(file_name)

    features.append([data, class_label])

# Convert into a Panda dataframe
featuresdf = pd.DataFrame(features, columns=['feature','class_label'])

print('Finished feature extraction from ', len(featuresdf), ' files')
```

```
Finished feature extraction from  5435  files
```

```
featuresdf.head()
```

	feature	class_label
0	[-148.29195, 125.17131, -18.48167, 13.752467, ...	engine_idling
1	[-216.45773, 175.77246, 0.244275, 65.77827, 6....	engine_idling
2	[-193.82822, 105.66298, -39.473175, 32.525784,...	drilling
3	[-143.99443, 111.079796, -33.822388, 46.725525...	car_horn
4	[-261.05377, 119.62229, -55.061405, 31.013195,...	children_playing

Now, encode the categorical class label into one-hot numerical code by using sklearn preprocessing LabelEncoder so it can be understood when analysing.

```
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

# Convert features and corresponding classification labels into numpy arrays
X = np.array(featuresdf.feature.tolist())
y = np.array(featuresdf.class_label.tolist())

# Encode the classification labels
le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))
```

Then we will use sklearn.model_selection.train_test_split to split the dataset into training and testing sets. The testing set size will be 30% and we will set a random state.

```
# split the dataset
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, yy, test_size=0.3, random_state = 42)
```

Initial model architecture - MLP

We will start with 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 (Mel Frequency Cepstral Coefficients)(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 for our first 2 layers is the ReLU(Rectified Linear Activation). This activation function has been proven to work well in neural networks.

We will also apply a **Dropout value** of 20% 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 generalisation and is less likely to overfit the training data.

Our output layer will have 10 nodes (number of labels) 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.

```python
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import Adam
from keras.utils import np_utils
from sklearn import metrics

num_labels = yy.shape[1]
filter_size = 4

# Construct model
model = Sequential()

model.add(Dense(256, input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

Compiling the model

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(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

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

# Calculate pre-training accuracy
score = model.evaluate(x_test, y_test, verbose=0)
accuracy = 100*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)
```

Training

Here we will train the model. We will start with 100 epochs which is the number of times the model will cycle through the data. The model will improve on each cycle until it reaches a certain point. We will also start with a low batch size, as having a large batch size can reduce the generalisation ability of the model.

```
from keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 100
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='../Capstone_Project_2/weights.best.basic_mlp.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(x_test, y_test), callbacks=[che


duration = datetime.now() - start
print("Training completed in time: ", duration)
```

Test the model

Here we will review the accuracy of the model on both the training and test data sets.

```
# Evaluating the model on the training and testing set
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])
```

```
Training Accuracy:  0.9978969693183899
Testing Accuracy:  0.9270386099815369
```

The testing accuracy is 7% lower than the training accuracy, and 92.7% of testing accuracy is high enough to make the prediction. But this model has overfitting issue, as the accuracy difference between training and testing is more than 5%

Now, let us test some random wav file in the test dataset.

```
# Drilling
filename = '../Capstone_Project_2/test/Test/4060.wav'
print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                   :  0.00000000000000000000000000000027
car_horn                :  0.00000000000000002903503873479124
children_playing                  :  0.00000000000000000000115494052993
dog_bark                :  0.00000000000001462043410871503163
drilling                :  0.99821376800537109375000000000000
engine_idling           :  0.00000000000000000000000000000002
gun_shot                :  0.00000000000000000000092428992010
jackhammer              :  0.00178629334550350904464721679688
siren          :  0.00000000000000000000000000000324
street_music            :  0.00000000000000003960605623863920
```

```
# Children Playing
filename = '../Capstone_Project_2/test/Test/7128.wav'
print_prediction(filename)
```

```
The predicted class is: children_playing

air_conditioner                   :  0.00000000000003283306244071526602
car_horn                :  0.00000015141870335355633869767189
children_playing                  :  0.98311859369277954101562500000000
dog_bark                :  0.00007684771117055788636207580566
drilling                :  0.00000007332023699291312368586659
engine_idling           :  0.00000000000007794520508631191946
gun_shot                :  0.01669478975236415863037109375000
jackhammer              :  0.00000000000000237326074557211910
siren          :  0.00000000000065398576311326284127
street_music            :  0.00010941340588033199310302734375
```

```python
# Street Music
filename = '../Capstone_Project_2/test/Test/3335.wav'
print_prediction(filename)
```

```
The predicted class is: street_music

air_conditioner                  :  0.00770732527598738670349121093750
car_horn                 :  0.00000438925690104952082037925720
children_playing                 :  0.00181827484630048274993896484375
dog_bark                 :  0.00021549727534875273704528808594
drilling                 :  0.23018267750740051269531250000000
engine_idling            :  0.00080198544310405850410461425781
gun_shot                 :  0.00001067862194759072735905647278
jackhammer               :  0.00843258295208215713500976562500
siren            :  0.00002845075505319982767105102539
street_music             :  0.75079816579818725585937500000000
```

```python
# Dog bark
filename = '../Capstone_Project_2/test/Test/281.wav'
print_prediction(filename)
```

```
The predicted class is: dog_bark

air_conditioner                  :  0.00000000000000000000000000000000
car_horn                 :  0.00000000000000000254738789596014
children_playing                 :  0.00000000000000000005841554009493
dog_bark                 :  1.00000000000000000000000000000000
drilling                 :  0.00000000000000000017729706531400
engine_idling            :  0.00000000000000000000000000000000
gun_shot                 :  0.000000000000044505501714531270352
jackhammer               :  0.00000000000000000000000000000000
siren            :  0.00000000000000000002195405523493
street_music             :  0.00000000000000000000000000000002
```

```
# Gun shot
filename = '../Capstone_Project_2/test/Test/7117.wav'
print_prediction(filename)
```

```
The predicted class is: gun_shot

air_conditioner                  :  0.00000000000000007833223053353841
car_horn                 :  0.00000000000000000905931515590526
children_playing                 :  0.00001230783709615934640169143677
dog_bark                 :  0.05733761936426162719726562500000
drilling                 :  0.00000017433109178455197252333164
engine_idling            :  0.00000000000000000000000000216941
gun_shot                 :  0.94264984130859375000000000000000
jackhammer               :  0.00000000000000000000018524611116
siren            :  0.00000000002287933384415019588687
street_music             :  0.00000000005439900266357433622488
```

```
#air conditioner
filename = '../Capstone_Project_2/test/Test/1127.wav'
print_prediction(filename)
```

```
The predicted class is: air_conditioner

air_conditioner                  :  0.41991049051284790039062500000000
car_horn                 :  0.00000048313881961803417652845383
children_playing                 :  0.24721004068851470947265625000000
dog_bark                 :  0.10683305561542510986328125000000
drilling                 :  0.00000125928022498555947095155716
engine_idling            :  0.01999645680189132690429687500000
gun_shot                 :  0.00118373206350952386856079101562
jackhammer               :  0.00002046230110863689333200454712
siren            :  0.00000950130413457372924312949180
street_music             :  0.20484319329261779785156250000000
```

```python
#car horn
filename = '../Capstone_Project_2/test/Test/1102.wav'
print_prediction(filename)
```

```
The predicted class is: car_horn

air_conditioner                  :  0.00000000000000000000000000000001
car_horn                 :  1.00000000000000000000000000000000
children_playing                 :  0.00000000000000000000000000000000
dog_bark                 :  0.00000000000000000000000000000000
drilling                 :  0.00000000000000000000000000000000
engine_idling            :  0.00000000000000000000000000000000
gun_shot                 :  0.00000000000000000000000000000000
jackhammer               :  0.00000000000000000000000000000000
siren            :  0.00000000000000000000000000000000
street_music             :  0.00000000000000000000000000000000
```

```python
#engine idling
filename = '../Capstone_Project_2/test/Test/107.wav'
print_prediction(filename)
```

```
The predicted class is: engine_idling

air_conditioner                  :  0.00023651641095057129859924316406
car_horn                 :  0.00000987872226687613874673843384
children_playing                 :  0.00071573175955563783645629882812
dog_bark                 :  0.00007891857967479154467582702637
drilling                 :  0.00000005278201697933582181576639
engine_idling            :  0.98152655363082885742187500000000
gun_shot                 :  0.00001562062607263214886188507080
jackhammer               :  0.00000114825729724543634802103043
siren            :  0.00003644346361397765576839447021
street_music             :  0.01737919263541698455810546875000
```

```
#siren
filename = '../Capstone_Project_2/test/Test/106.wav'
print_prediction(filename)
```

```
The predicted class is: siren

air_conditioner                  :  0.00000000000002614255552757949880
car_horn                :  0.00000000000004463886671326328103 9
children_playing                 :  0.00000000130105382023515403489000
dog_bark                :  0.00026945961872115731239318847656
drilling                :  0.00000000001841969979321511630133
engine_idling           :  0.00000000000063600188483781128213
gun_shot                :  0.00000000128683586009259443017072
jackhammer              :  0.00000000000066077492869714982149
siren          :  0.99973052740097045898437500000000
street_music            :  0.00000010344954226582103729015 22
```

```
#jackhammer
filename = '../Capstone_Project_2/test/Test/1099.wav'
print_prediction(filename)
```

```
The predicted class is: jackhammer

air_conditioner                  :  0.00000929705265662050805985927 58
car_horn                :  0.00000000000003573262359287711354
children_playing                 :  0.00000000005342382439210702216315
dog_bark                :  0.00000000000010555459636777189680
drilling                :  0.00000000000000000582313272310129
engine_idling           :  0.00000000000003284541565392131 75
gun_shot                :  0.00000000000102864331635865724479
jackhammer              :  0.99999046325683593750000000000000
siren          :  0.00000000000015835591301791018815
street_music            :  0.00000000000242116613216603049352
```

Observations

The performance of our initial model is satisfactory and has generalised well, seeming to predict well when tested against testing audio data.

Advanced Mode

Convolutional Neural Network(CNN)

CNN is typically make good classifiers and perform particular well with image classification task due to their feature extraction and classification parts. I believe that this will be very effective in finding patterns within MFCC, much like they are effective at finding patterns within images.

With sequential model, starting with a simple model architecture, consisting of 4 Conv2D convolution layers, with our final output layer being a dense layer. Our output layer will have 10 nodes which matches the number of possible classifications.

As CNN typically use the same- size input, and our audio samples have different audio lengths, so we will pad zero when needed in MFCCs.

The following the new feature extracting function.

```python
import numpy as np
max_pad_len = 174

def extract_features(file_name):

    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
        pad_width = max_pad_len - mfccs.shape[1]
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')

    except Exception as e:
        print("Error encountered while parsing file: ", file_name)
        return None

    return mfccs
```

Now we have a 4D training dataset with dimension of

174* 3804* 40* 1

Building the Model:

```
num_rows = 40
num_columns = 174
num_channels = 1

x_train = x_train.reshape(x_train.shape[1], num_rows, num_columns, num_channels)
x_test = x_test.reshape(x_test.shape[1], num_rows, num_columns, num_channels)

num_labels = yy.shape[1]
filter_size = 2

# Construct model
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns, num_channels), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(GlobalAveragePooling2D())

model.add(Dense(num_labels, activation='softmax'))
```

Compiling the model:

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

# Calculate pre-training accuracy
score = model.evaluate(x_test, y_test, verbose=0)
accuracy = 100*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 39, 173, 16)       80
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 19, 86, 16)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 19, 86, 16)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 18, 85, 32)        2080
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 9, 42, 32)         0
_________________________________________________________________
dropout_18 (Dropout)         (None, 9, 42, 32)         0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 41, 64)         8256
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 4, 20, 64)         0
_________________________________________________________________
dropout_19 (Dropout)         (None, 4, 20, 64)         0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 3, 19, 128)        32896
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 1, 9, 128)         0
_________________________________________________________________
dropout_20 (Dropout)         (None, 1, 9, 128)         0
_________________________________________________________________
global_average_pooling2d_5 ( (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 44,602
Trainable params: 44,602
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/ke
recated. Please use tf.compat.v1.global_variables instead.

Pre-training accuracy: 9.1355%
```

Training the model:

In this process, CNN can take a significant amount of time (19 minutes) with 72 epochs and 256 batches.

```
from keras.callbacks import ModelCheckpoint
from datetime import datetime

#num_epochs = 12
#num_batch_size = 128

num_epochs = 72
num_batch_size = 256

checkpointer = ModelCheckpoint(filepath='../Capstone_Project_2/weights.best.basic_mlp.hdf6',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs,
          validation_data=(x_test, y_test), callbacks=[checkpointer], verbose=1)


duration = datetime.now() - start
print("Training completed in time: ", duration)
```

Evaluating the CNN model:

From the result shown below, the overfitting issue has been resolved. Even though the accuracy is much lower than that in MLP, but in classification problem, we should take overfitting issue prior the accuracy.

```
# Evaluating the model on the training and testing set
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])
```

```
Training Accuracy:  0.9048370122909546
Testing Accuracy:  0.847946047782898
```

Result:
The CNN model has lower accuracy compared to MLP model, but 85% of testing accuracy with 90% of training accuracy is still considerably high accuracy in classification problem. And MLP has much higher accuracy, in both training and testing(99% and 92%) . This leaves us the tradeoff between accuracy and overfitting. In classification problem, we typically **refer lower overfitting** than higher accuracy when accuracy for both models are considerably high enough.
So in this case, **CNN model** is preferred in this Urban Sound Classification problem.

Conclusion

It was previously noted in our data exploration, that it is difficult to visualise 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.

• Similar pattern for children playing and street music.

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

1. The initial problem was defined and relevant public dataset was located.
2. The data was explored and analysed
3. Data was preprocessed and features were extracted.
4. MLP model was trained and evaluated(99% training accuracy and 92% testing accuracy)
5. CNN model was trained and evaluated(90% training accuracy with 85% testing accuracy)
6. Both models have considerably high accuracy, therefore we prefer the one with less overfitting, i.e. CNN model.

From the initial exploration of the data in step 2, I envisaged that the preprocessing work in step 3 would be incredibly time consuming. However, this was actually relatively easy with Librosa package. I also thought that the feature extraction would be a lot trickier but again Librosa shortened the effort required immensely.

MFCC's we extracted in step 3 perform much better than I had expected.

Improvement

If I was to continue with this project there are a number of additional areas that could be explored:

1. Test the models performance with Real-time audio.

2. Train the model for real world data. This would likely involve augmenting the training data in various ways such as:

   a. **Adding a variety of different background sounds.**
   b. **Adjusting the volume levels of the target sound or adding echoes.**
   c. **Changing the starting position of the recording sample, e.g. the shape of a dog bark.**

3. Experiment to see if per-class accuracy is affected by using training data of different durations.