UDACITY

UDACITY MACHINE LEARNING NANODEGREE

2019

# CAPSTONE PROJECT

## Classifying Urban sounds using Deep Learning

Ferhat AKAR

# 1 Definition
## 1.1 Project Overview

People are always in contact with audio data. The human brain is continuously processing and understanding this audio data, either consciously or subconsciously, giving us information about the environment around us. Environmental sound classification is a growing area of research with numerous real world applications. I would like to focus on solving automations problems while learning environmental sound classification. The machines running around us have a variety of sounds during their work. For example, when the press machine produces faulty parts during operation, the sound coming voice is different. Also, The sound of the strap from the hood while the car is running. In this project, I will investigate this problem. But I no avilable a enought data for my problem.
The goal of this capstone project, is to apply Deep Learning techniques to the classification of environmental sounds, specifically focusing on the identification of particular urban sounds.
There is a plethora of real world applications for this research, such as:
- Can be used for hearing impaired people.
- Smart home use
- Automotive where recognising sounds both inside and outside of the car can improve safety
- Industrial uses such as predictive maintenance
- Used for quality controls.

## 1.2 Problem Statement

The main objective of this project will be to use Deep Learning techniques to classify urban sounds. When given an voice sample in a computer readable format (.wav file) of a few seconds duration, I want to be able to determine if it contains one of the target urban sounds with a corresponding likelihood score. Conversely, if don't detect of the target sounds, 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 Visualisation
### 2.1.1 UrbanSound dataset

I will use a dataset called Urbansound8K for this project. The dataset contains 8732 sound excerpts (<=4s) of urban sounds in10 classes, which are:

- Air Conditioner
- Car Horn
- Children Playing
- Dog bark
- Drilling
- Engine Idling
- Gun Shot
- Jackhammer
- Siren
- Street Music

This metadata contains a unique ID for each sound excerpt along with it's given class name.

A sample of this dataset is included with the accompanying git repo and the full dataset can be access from here.

#### 2.1.2 Audio sample file data overview

Used sound are digital audio files in .wav format.

Sound waves are digitised by sampling them at discrete intervals known as the sampling rate.

The bit depth determines how detailed the sample will be also known as the dynamic range of the signal (typically 16bit which means a sample can range from 65,536 amplitude values).

Therefore, the data we will be analysing for each sound excerpts is essentially a one dimensional array or vector of amplitude values.

### 2.1.3 Analysing audio data

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

***1. IPython.display.Audio***

This allows us to play audio directly in the Jupyter Notebook.

***2. Librosa***

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.

You may need to install librosa using pip as follows:

```
pip install librosa
```

### 2.1.4 **Auditory inspection**

I will use `IPython.display.Audio` to play the audio files so we can inspect aurally.

In [1]:

```
import IPython.display as ipd

ipd.Audio('../UrbanSound Dataset sample/audio/100032-3-0-0.wav')
```

## Visual inspection

### 2.1.5 Visual inspection

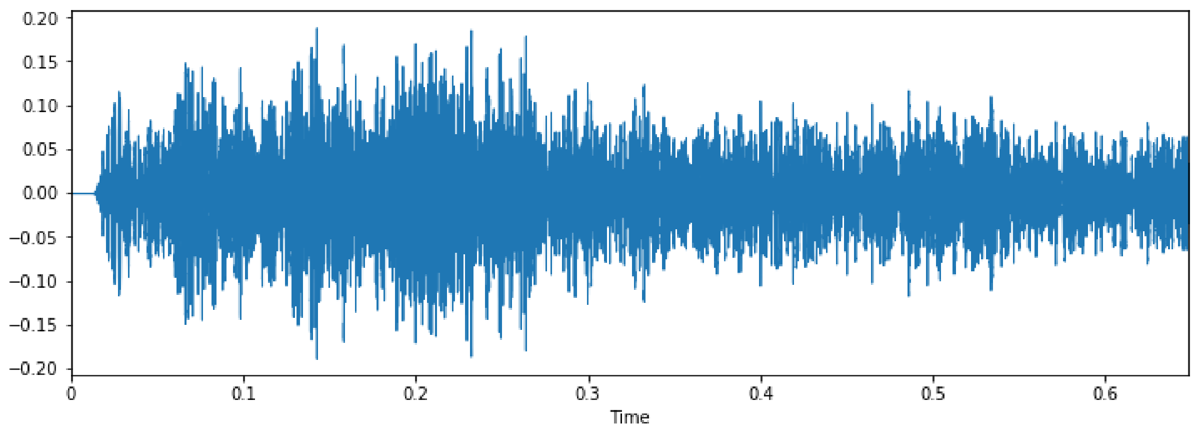
I will load a sample from each class and visually inspect the data for any patterns. I will use librosa to load the audio file into an array then librosa.display and matplotlib to display the waveform.

In [2]:

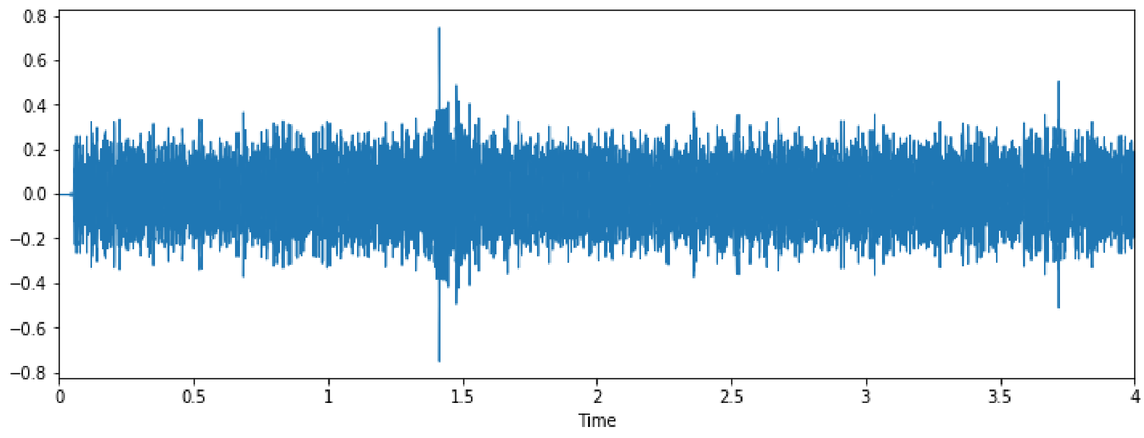
```
# Load imports

import IPython.display as ipd
import librosa
import librosa.display
import matplotlib.pyplot as plt
```

In [3]:

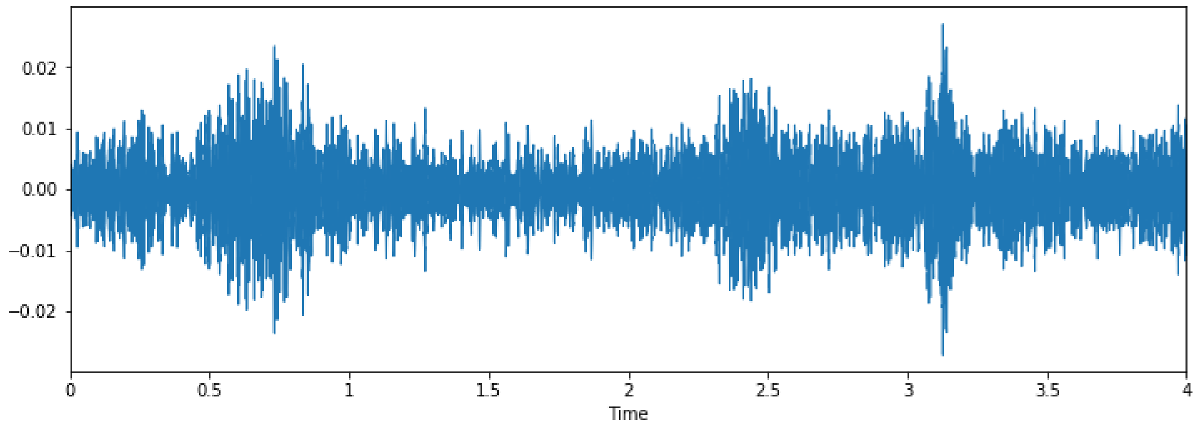
```
# Class: Car horn

filename = '../UrbanSound Dataset sample/audio/100648-1-0-0.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```



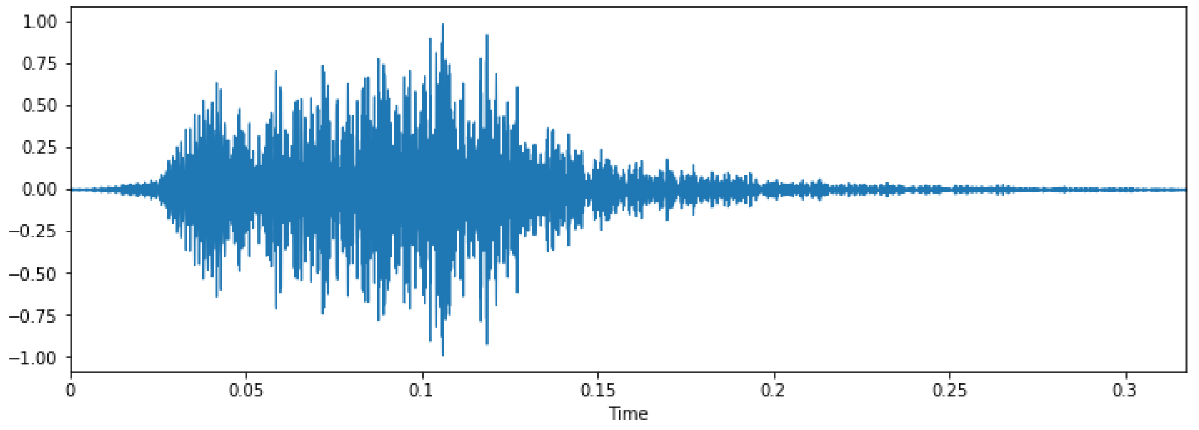
```
# Class: Air Conditioner
```



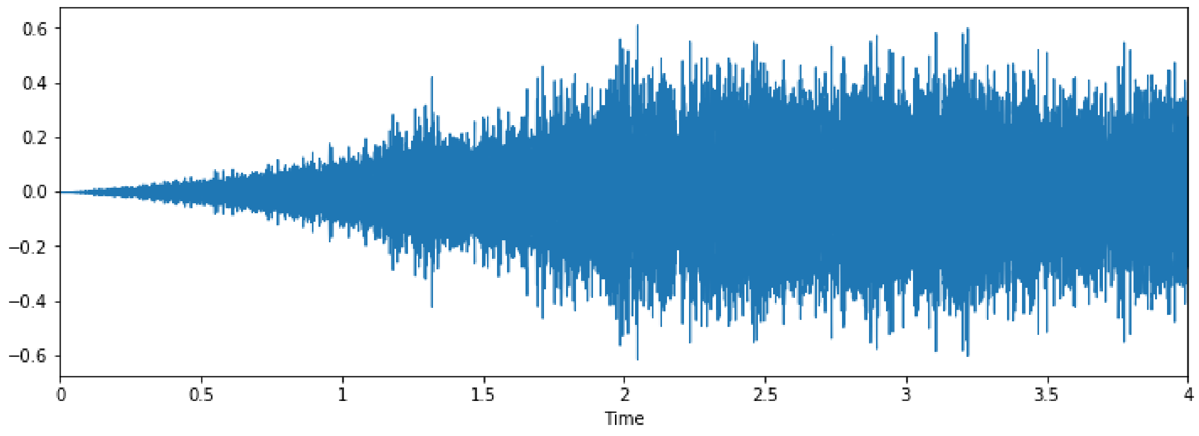
# Class: Children playing



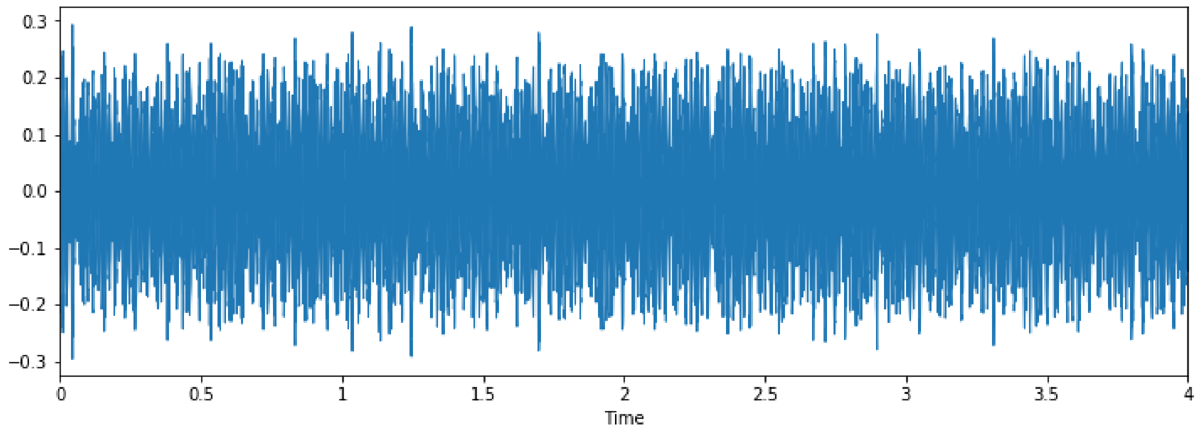
# Class: Dog bark



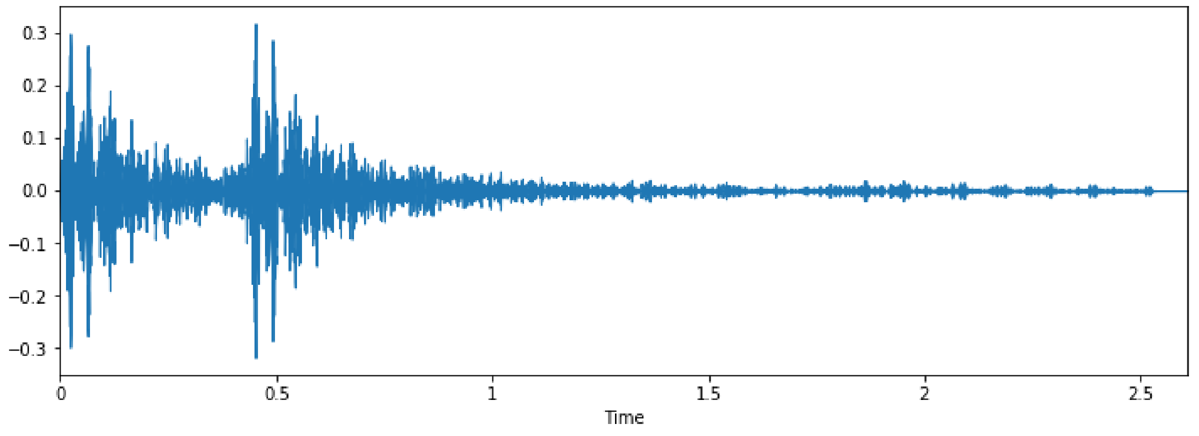
# Class: Drilling



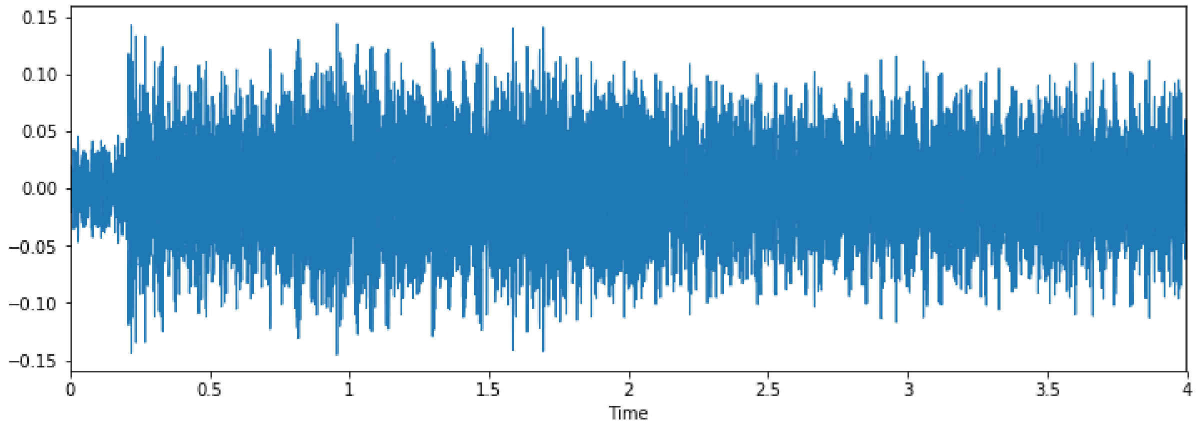
# Class: Engine Idling



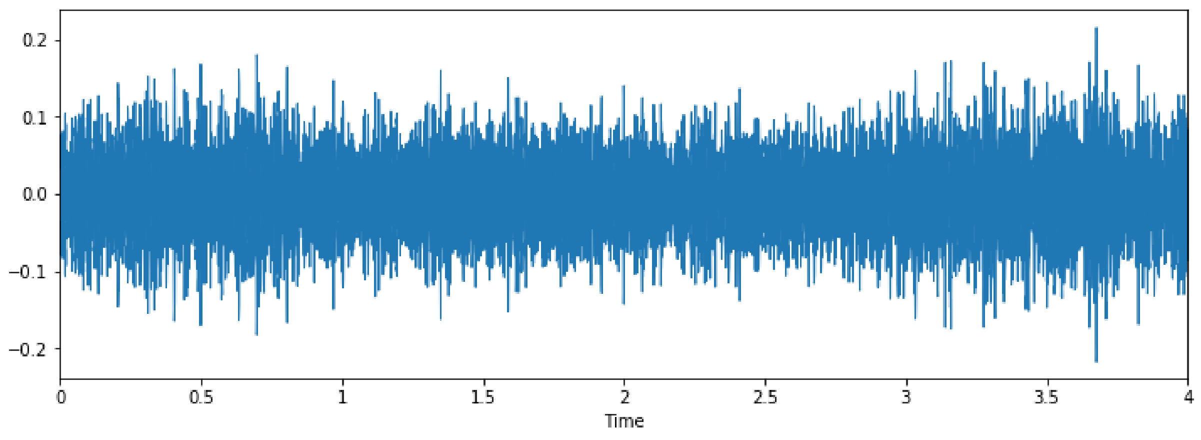
# Class: Gunshot



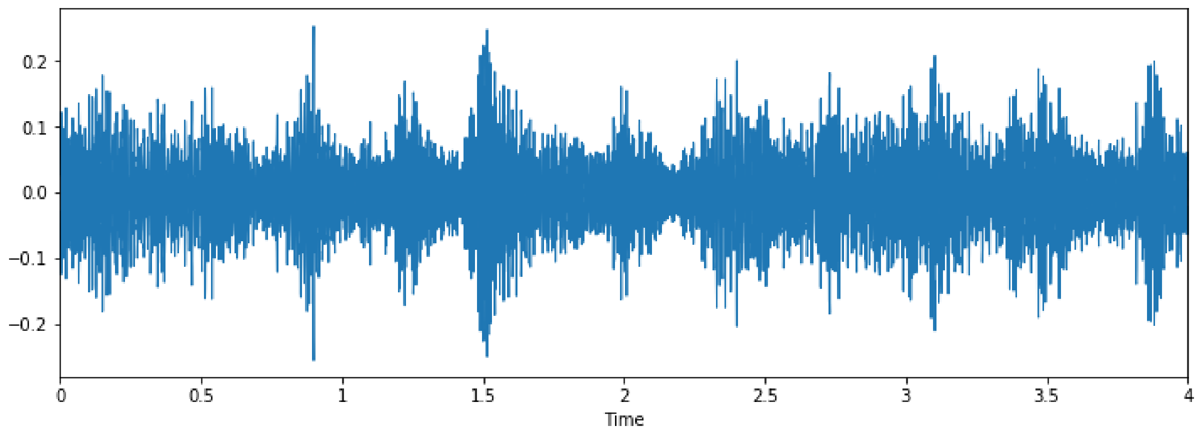
# Class: Jackhammer



# Class: Siren



# Class: Street music



### 2.1.6 Observations

From a visual inspection we can see that it is tricky to visualise the difference between some of the classes.

Particularly, the waveforms for reptitive sounds for air conditioner, drilling, engine idling and jackhammer are similar in shape.

Likewise the peak in the dog barking sample is simmilar in shape to the gun shot sample (albeit the samples differ in that there are two peaks for two gunshots compared to the one peak for one dog bark). Also, the car horn is similar too.

We show to similarities between the children playing and street music.

The human ear can naturally detect the difference between the harmonics, it will be interesting to see how well a deep learning model will be able to extract the necessary features to distinguish between these classes.

However, it is easy to differentiate from the waveform shape, the difference between certain classes such as dog barking and engine idling.

### 2.1.7 Dataset Metadata

Here we will load the UrbanSound metadata .csv file into a Panda dataframe.

In [13]:

```
import pandas as pd
metadata = pd.read_csv('../UrbanSound Dataset sample/metadata/UrbanSound8K.
csv')
metadata.head()
```

| | slice_file_name | fsID | start | end | salience | fold | classID | class_name |
|---|---|---|---|---|---|---|---|---|
| **0** | 100032-3-0-0.wav | 100032 | 0.0 | 0.317551 | 1 | 5 | 3 | dog_bark |
| **1** | 100263-2-0-117.wav | 100263 | 58.5 | 62.500000 | 1 | 5 | 2 | children_playing |
| **2** | 100263-2-0-121.wav | 100263 | 60.5 | 64.500000 | 1 | 5 | 2 | children_playing |
| **3** | 100263-2-0-126.wav | 100263 | 63.0 | 67.000000 | 1 | 5 | 2 | children_playing |
| **4** | 100263-2-0-137.wav | 100263 | 68.5 | 72.500000 | 1 | 5 | 2 | children_playing |

### 2.1.8 Class distributions

```
print(metadata.class_name.value_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
Name: class_name, dtype: int64
```

### 2.1.9 Observations

Here I 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 less samples at 43% and 37% respectively.

This will be a concern and something we may need to address later on.

### 2.1.10 Audio sample file properties

Next I will iterate through each of the audio sample files and extract, number of audio channels, sample rate and bit-depth.

In [15]:

```
# Load various imports
import pandas as pd
import os
import librosa
import librosa.display

from helpers.wavfilehelper import WavFileHelper
wavfilehelper = WavFileHelper()

audiodata = []
for index, row in metadata.iterrows():

    file_name = os.path.join(os.path.abspath('/Volumes/Untitled/ML_Data/Urban Sound/UrbanSound8K/audio/'),'fold'+str(row["fold"])+'/',str(row["slice_file_name"]))
    data = wavfilehelper.read_file_properties(file_name)
    audiodata.append(data)

# Convert into a Panda dataframe
audiodf = pd.DataFrame(audiodata, columns=['num_channels','sample_rate','bit_depth'])
```

### 2.1.11 Audio channels

Most of the samples have two audio channels (meaning stereo) with a few with just the one channel (mono).

The easiest option here to make them uniform will be to merge the two channels in the stero samples into one by averaging the values of the two channels.

In [19]:

```
# num of channels

print(audiodf.num_channels.value_counts(normalize=True))
```

```
2    0.915369
1    0.084631
Name: num_channels, dtype: float64
```

### 2.1.12 Sample rate

There is a wide range of Sample rates that have been used across all the samples which is a concern (ranging from 96k to 8k).

This likley means that we will have to apply a sample-rate conversion technique (either up-conversion or down-conversion) so we can see an agnostic representation of their waveform which will allow us to do a fair comparison.

In [21]:

```
# sample rates

print(audiodf.sample_rate.value_counts(normalize=True))
```

```
44100     0.614979
48000     0.286532
```

```
96000     0.069858
24000     0.009391
16000     0.005153
22050     0.005039
11025     0.004466
192000    0.001947
8000      0.001374
11024     0.000802
32000     0.000458
Name: sample_rate, dtype: float64
```

### 2.1.13 Bit-depth

There is also a wide range of bit-depths. It's likely that we may need to normalise them by taking the maximum and minimum amplitude values for a given bit-depth.

In [22]:

```
# bit depth

print(audiodf.bit_depth.value_counts(normalize=True))
```

```
16    0.659414
24    0.315277
32    0.019354
8     0.004924
4     0.001031
Name: bit_depth, dtype: float64
```

### 2.1.14 Other audio properties to consider

I may also need to consider normalising the volume levels (wave amplitude value) if this is seen to vary greatly, by either looking at the peak volume or the RMS volume.

## 2.2 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) [2] from the 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 architecture, such as Multi-Layer Perceptron before experimenting with more complex architectures such as Convolutional Neural Networks.

Multi-layer perceptron's (MLP) are classed as a type of Deep Neural Network as they are composed of more than one layer of perceptrons and use non-linear activation which distinguish them from linear perceptrons. Their architecture consists of an input layer, an output layer that ultimately make 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 it the error can go no lower which is known as convergence.

Convolutional Neural Networks (CNNs) build 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 the 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.

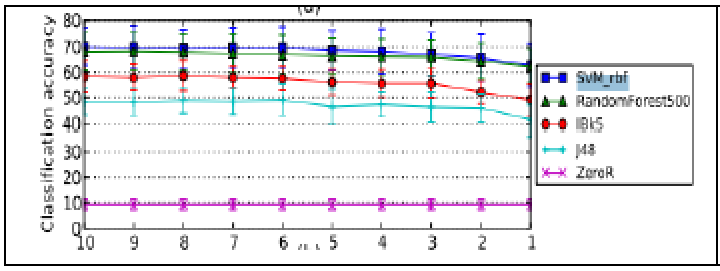
For the reasons discussed, both MLPs and CNN's typically make good classifiers, where CNN's in particular perform very well with image classification tasks due to their feature extraction and classification parts.

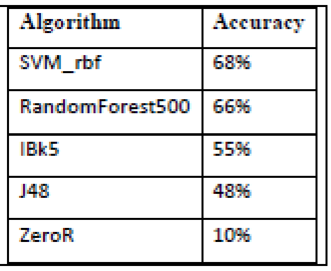
I believe that this will be very effective at finding patterns within the MFCC's much like they are effective at finding patterns within images.

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.

### 2.3 Benchmark Model

For the benchmark model, I 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 a audio slice maximum duration of 4 seconds.





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

## 3 Methodology

### 3.1 Data Preprocessing and Data Splitting

#### 3.1.1 Audio properties that will require normalising

Following on from the previous notebook, we identifed the following audio properties that need preprocessing to ensure consistency across the whole dataset:

- Audio Channels
- Sample rate
- Bit-depth

We will continue to use Librosa which will be useful for the pre-processing and feature extraction.

#### 3.1.2 Preprocessing stage

For much of the preprocessing we will be able to use Librosa's load() function.

We will compare the outputs from Librosa against the default outputs of scipy's wavfile library using a chosen file from the dataset.

***Sample rate conversion***

Librosa’s load function converts by defaults the sampling rate to 22.05 KHz which we can use as our comparison level.

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

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.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
```

### ***Bit-depth***

Librosa’s load function will normalise the data so it's values range between -1 and 1. This removes the complication of the dataset having a wide range of bit-depths.

```
print('Original audio file min~max range:', np.min(scipy_audio), 'to', np.m
ax(scipy_audio))
print('Librosa audio file min~max range:', np.min(librosa_audio), 'to', np.
max(librosa_audio))
```

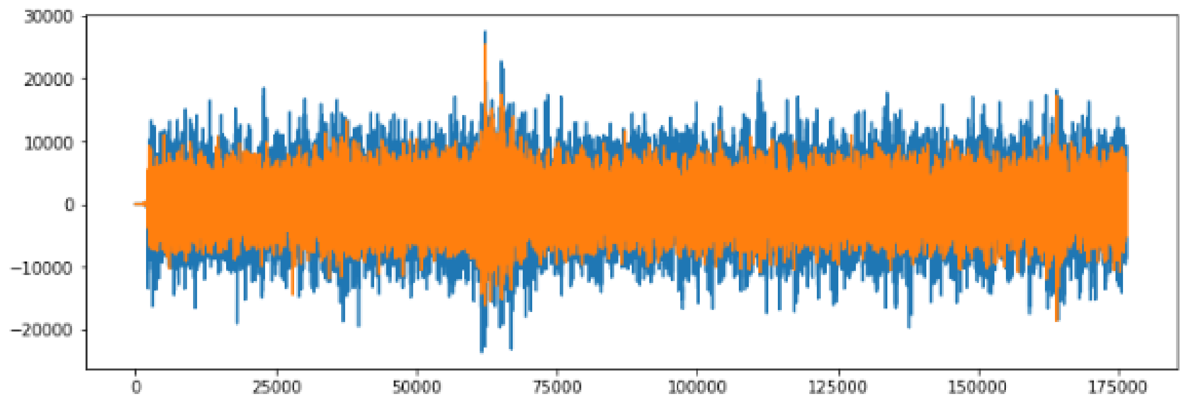
```
Original audio file min~max range: -23628 to 27507
Librosa audio file min~max range: -0.50266445 to 0.74983937
```

### ***Merge audio channels***

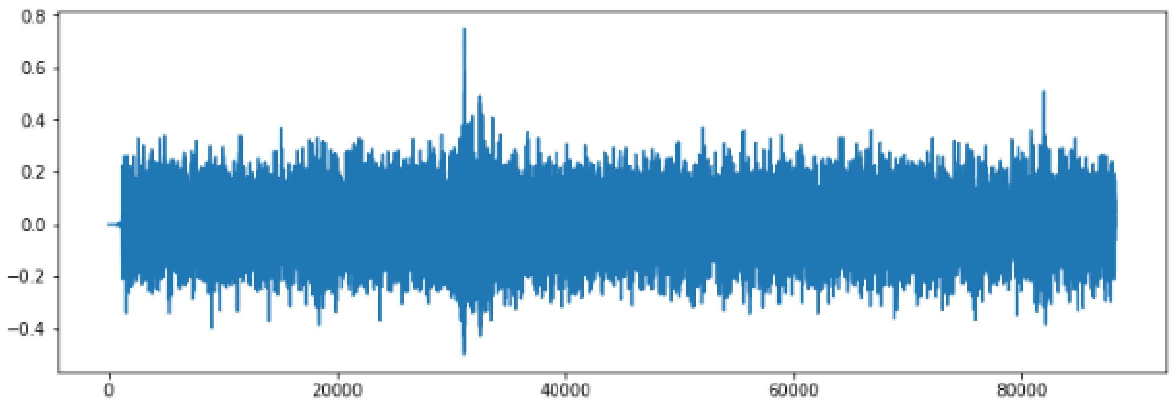
Librosa will also convert the signal to mono,it's meaning the number of channels will always be 1.

In [5]:

```
import matplotlib.pyplot as plt

# Original audio with 2 channels
plt.figure(figsize=(12, 4))
plt.plot(scipy_audio)
```



```
# Librosa audio with channels merged
plt.figure(figsize=(12, 4))
plt.plot(librosa_audio)
```



***Other audio properties to consider***

At this stage it is not yet clear whether other factors may also need to be taken into account, such as sample duration length and volume levels.

We will proceed as is for the meantime and come back to address these later if it's perceived to be effecting the validity of our target metrics.

### 3.1.3 **Extract Features**

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

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.

***Extracting a MFCC***

For this we will use Librosa's mfcc() function which generates an MFCC from time series audio data.

In [7]:

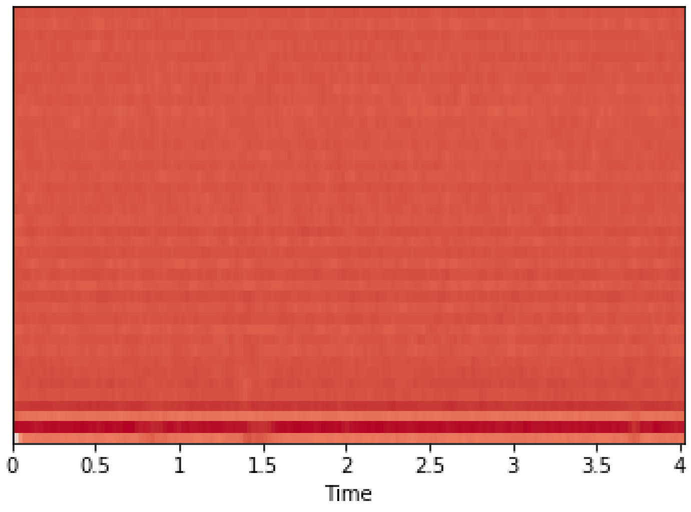
```
mfccs = librosa.feature.mfcc(y=librosa_audio, sr=librosa_sample_rate, n_mfc
c=40)
print(mfccs.shape)
```

```
(40, 173)
```

This shows librosa calculated a series of 40 MFCCs over 173 frames.

In [8]:

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



***Extracting MFCC's for every file***

We will now extract an MFCC for each audio file in the dataset and store it in a Panda Dataframe along with it's classification label.

In [14]:

```
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)
        mfccsscaled = np.mean(mfccs.T,axis=0)

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

    return mfccsscaled
```

In [19]:

```
# Load various imports
import pandas as pd
import os
import librosa

# Set the path to the full UrbanSound dataset
fulldatasetpath = '/Volumes/Untitled/ML_Data/Urban Sound/UrbanSound8K/audio
/'

metadata = pd.read_csv('../UrbanSound Dataset sample/metadata/UrbanSound8K.
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),'fold'+str(ro
w["fold"])+'/',str(row["slice_file_name"]))

    class_label = row["class_name"]
    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  8732  files
```

### 3.1.4 Convert the data and labels

We will use `sklearn.preprocessing.LabelEncoder` to encode the categorical text data into model-understandable numerical data.

In [16]:

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

# Convert features and corresponding classification labels into numpy array
s
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))
```

### 3.1.5 Split the dataset

Here we will use `sklearn.model_selection.train_test_split` to split the dataset into training and testing sets. The testing set size will be 20% and we will set a random state.

In [17]:

```
# 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=0.2, r
andom_state = 42)
```

## 3.2 Implementation
### 3.2.1 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 (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`, or `Rectified Linear Activation`. This activation function has been proven 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 generalisation and is less likely to overfit the training data.

Our output layer will have 10 nodes (num_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.

In [2]:

```
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 = 2

# Construct model
model = Sequential()

model.add(Dense(256, input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))

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

### 3.2.2 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'], optimi
zer='adam')
```

In [4]:

```
# 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 #
=================================================================
dense_1 (Dense)              (None, 256)               10496
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 78,858
Trainable params: 78,858
Non-trainable params: 0
_________________________________________________________________
Pre-training accuracy: 11.5627%
```

### 3.2.3 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.

In [5]:

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

num_epochs = 100
num_batch_size = 32

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

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


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

```
Train on 6985 samples, validate on 1747 samples
Epoch 1/100
6985/6985 [==============================] - 4s 583us/step - loss: 11.1227
- acc: 0.2165 - val_loss: 6.8835 - val_acc: 0.3526

Epoch 00001: val_loss improved from inf to 6.88353, saving model to saved_m
odels/weights.best.basic_mlp.hdf5
Epoch 2/100
6985/6985 [==============================] - 2s 352us/step - loss: 4.8051 -
acc: 0.3087 - val_loss: 1.8028 - val_acc: 0.4139

Epoch 00002: val_loss improved from 6.88353 to 1.80277, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 3/100
6985/6985 [==============================] - 2s 336us/step - loss: 1.9797 -
acc: 0.3576 - val_loss: 1.6131 - val_acc: 0.5243

Epoch 00003: val_loss improved from 1.80277 to 1.61305, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 4/100
6985/6985 [==============================] - 2s 335us/step - loss: 1.6974 -
acc: 0.4283 - val_loss: 1.3788 - val_acc: 0.5736
```

```
Epoch 00004: val_loss improved from 1.61305 to 1.37876, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 5/100
6985/6985 [==============================] - 2s 339us/step - loss: 1.5283 -
acc: 0.4769 - val_loss: 1.2597 - val_acc: 0.6033

Epoch 00005: val_loss improved from 1.37876 to 1.25972, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 6/100
6985/6985 [==============================] - 2s 340us/step - loss: 1.4293 -
acc: 0.5215 - val_loss: 1.1548 - val_acc: 0.6485

Epoch 00006: val_loss improved from 1.25972 to 1.15475, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 7/100
6985/6985 [==============================] - 2s 350us/step - loss: 1.3435 -
acc: 0.5509 - val_loss: 1.1064 - val_acc: 0.6611

Epoch 00007: val_loss improved from 1.15475 to 1.10643, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 8/100
6985/6985 [==============================] - 3s 362us/step - loss: 1.2510 -
acc: 0.5781 - val_loss: 1.0391 - val_acc: 0.6823

Epoch 00008: val_loss improved from 1.10643 to 1.03908, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 9/100
6985/6985 [==============================] - 3s 374us/step - loss: 1.2365 -
acc: 0.5778 - val_loss: 1.0108 - val_acc: 0.6754

Epoch 00009: val_loss improved from 1.03908 to 1.01082, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 10/100
6985/6985 [==============================] - 3s 361us/step - loss: 1.1524 -
acc: 0.6030 - val_loss: 0.9651 - val_acc: 0.6886

Epoch 00010: val_loss improved from 1.01082 to 0.96510, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 11/100
6985/6985 [==============================] - 2s 354us/step - loss: 1.1274 -
acc: 0.6156 - val_loss: 0.9360 - val_acc: 0.6903

Epoch 00011: val_loss improved from 0.96510 to 0.93600, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 12/100
6985/6985 [==============================] - 2s 344us/step - loss: 1.0735 -
acc: 0.6265 - val_loss: 0.8927 - val_acc: 0.7281
```

```
Epoch 00012: val_loss improved from 0.93600 to 0.89269, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 13/100
6985/6985 [==============================] - 2s 350us/step - loss: 1.0516 -
acc: 0.6389 - val_loss: 0.8353 - val_acc: 0.7230

Epoch 00013: val_loss improved from 0.89269 to 0.83531, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 14/100
6985/6985 [==============================] - 2s 353us/step - loss: 1.0339 -
acc: 0.6465 - val_loss: 0.8109 - val_acc: 0.7321

Epoch 00014: val_loss improved from 0.83531 to 0.81090, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 15/100
6985/6985 [==============================] - 2s 355us/step - loss: 0.9959 -
acc: 0.6547 - val_loss: 0.8110 - val_acc: 0.7476

Epoch 00015: val_loss did not improve from 0.81090
Epoch 16/100
6985/6985 [==============================] - 2s 345us/step - loss: 0.9613 -
acc: 0.6687 - val_loss: 0.7552 - val_acc: 0.7590

Epoch 00016: val_loss improved from 0.81090 to 0.75516, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 17/100
6985/6985 [==============================] - 2s 349us/step - loss: 0.9541 -
acc: 0.6797 - val_loss: 0.7335 - val_acc: 0.7676

Epoch 00017: val_loss improved from 0.75516 to 0.73350, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 18/100
6985/6985 [==============================] - 2s 354us/step - loss: 0.9186 -
acc: 0.6902 - val_loss: 0.7308 - val_acc: 0.7607

Epoch 00018: val_loss improved from 0.73350 to 0.73077, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 19/100
6985/6985 [==============================] - 2s 353us/step - loss: 0.8931 -
acc: 0.6959 - val_loss: 0.7040 - val_acc: 0.7813

Epoch 00019: val_loss improved from 0.73077 to 0.70403, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 20/100
6985/6985 [==============================] - 3s 361us/step - loss: 0.8665 -
acc: 0.7078 - val_loss: 0.6615 - val_acc: 0.8002
```

```
Epoch 00020: val_loss improved from 0.70403 to 0.66146, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 21/100
6985/6985 [==============================] - 3s 358us/step - loss: 0.8527 -
acc: 0.7114 - val_loss: 0.6488 - val_acc: 0.8048

Epoch 00021: val_loss improved from 0.66146 to 0.64877, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 22/100
6985/6985 [==============================] - 2s 348us/step - loss: 0.8578 -
acc: 0.7148 - val_loss: 0.6610 - val_acc: 0.7956

Epoch 00022: val_loss did not improve from 0.64877
Epoch 23/100
6985/6985 [==============================] - 2s 350us/step - loss: 0.8290 -
acc: 0.7137 - val_loss: 0.6494 - val_acc: 0.8077

Epoch 00023: val_loss did not improve from 0.64877
Epoch 24/100
6985/6985 [==============================] - 3s 369us/step - loss: 0.8144 -
acc: 0.7236 - val_loss: 0.6361 - val_acc: 0.8031

Epoch 00024: val_loss improved from 0.64877 to 0.63609, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 25/100
6985/6985 [==============================] - 3s 364us/step - loss: 0.8250 -
acc: 0.7168 - val_loss: 0.6198 - val_acc: 0.8122

Epoch 00025: val_loss improved from 0.63609 to 0.61980, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 26/100
6985/6985 [==============================] - 3s 372us/step - loss: 0.8066 -
acc: 0.7251 - val_loss: 0.6066 - val_acc: 0.8191

Epoch 00026: val_loss improved from 0.61980 to 0.60664, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 27/100
6985/6985 [==============================] - 2s 356us/step - loss: 0.7804 -
acc: 0.7403 - val_loss: 0.5878 - val_acc: 0.8128

Epoch 00027: val_loss improved from 0.60664 to 0.58775, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 28/100
6985/6985 [==============================] - 3s 367us/step - loss: 0.7838 -
acc: 0.7307 - val_loss: 0.5887 - val_acc: 0.8197

Epoch 00028: val_loss did not improve from 0.58775
Epoch 29/100
```

```
6985/6985 [==============================] - 3s 371us/step - loss: 0.7706 -
acc: 0.7373 - val_loss: 0.5608 - val_acc: 0.8237

Epoch 00029: val_loss improved from 0.58775 to 0.56081, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 30/100
6985/6985 [==============================] - 2s 342us/step - loss: 0.7481 -
acc: 0.7396 - val_loss: 0.5763 - val_acc: 0.8220

Epoch 00030: val_loss did not improve from 0.56081
Epoch 31/100
6985/6985 [==============================] - 2s 358us/step - loss: 0.7472 -
acc: 0.7459 - val_loss: 0.5722 - val_acc: 0.8197

Epoch 00031: val_loss did not improve from 0.56081
Epoch 32/100
6985/6985 [==============================] - 3s 367us/step - loss: 0.7365 -
acc: 0.7476 - val_loss: 0.5668 - val_acc: 0.8180

Epoch 00032: val_loss did not improve from 0.56081
Epoch 33/100
6985/6985 [==============================] - 3s 361us/step - loss: 0.7410 -
acc: 0.7492 - val_loss: 0.5643 - val_acc: 0.8191

Epoch 00033: val_loss did not improve from 0.56081
Epoch 34/100
6985/6985 [==============================] - 2s 349us/step - loss: 0.7344 -
acc: 0.7472 - val_loss: 0.5609 - val_acc: 0.8288

Epoch 00034: val_loss did not improve from 0.56081
Epoch 35/100
6985/6985 [==============================] - 3s 368us/step - loss: 0.7187 -
acc: 0.7563 - val_loss: 0.5607 - val_acc: 0.8208

Epoch 00035: val_loss improved from 0.56081 to 0.56067, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 36/100
6985/6985 [==============================] - 2s 343us/step - loss: 0.7177 -
acc: 0.7556 - val_loss: 0.5344 - val_acc: 0.8414

Epoch 00036: val_loss improved from 0.56067 to 0.53436, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 37/100
6985/6985 [==============================] - 2s 329us/step - loss: 0.6918 -
acc: 0.7609 - val_loss: 0.5245 - val_acc: 0.8380

Epoch 00037: val_loss improved from 0.53436 to 0.52452, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
```

```
Epoch 38/100
6985/6985 [==============================] - 2s 330us/step - loss: 0.7010 -
acc: 0.7543 - val_loss: 0.5409 - val_acc: 0.8323

Epoch 00038: val_loss did not improve from 0.52452
Epoch 39/100
6985/6985 [==============================] - 2s 349us/step - loss: 0.6888 -
acc: 0.7611 - val_loss: 0.5411 - val_acc: 0.8271

Epoch 00039: val_loss did not improve from 0.52452
Epoch 40/100
6985/6985 [==============================] - 2s 352us/step - loss: 0.6761 -
acc: 0.7678 - val_loss: 0.5462 - val_acc: 0.8220

Epoch 00040: val_loss did not improve from 0.52452
Epoch 41/100
6985/6985 [==============================] - 2s 344us/step - loss: 0.6940 -
acc: 0.7641 - val_loss: 0.5248 - val_acc: 0.8260

Epoch 00041: val_loss did not improve from 0.52452
Epoch 42/100
6985/6985 [==============================] - 2s 344us/step - loss: 0.7008 -
acc: 0.7644 - val_loss: 0.5202 - val_acc: 0.8397

Epoch 00042: val_loss improved from 0.52452 to 0.52018, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 43/100
6985/6985 [==============================] - 2s 340us/step - loss: 0.6817 -
acc: 0.7655 - val_loss: 0.5340 - val_acc: 0.8357

Epoch 00043: val_loss did not improve from 0.52018
Epoch 44/100
6985/6985 [==============================] - 2s 350us/step - loss: 0.6772 -
acc: 0.7694 - val_loss: 0.5314 - val_acc: 0.8397

Epoch 00044: val_loss did not improve from 0.52018
Epoch 45/100
6985/6985 [==============================] - 2s 339us/step - loss: 0.6768 -
acc: 0.7712 - val_loss: 0.5035 - val_acc: 0.8523

Epoch 00045: val_loss improved from 0.52018 to 0.50350, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 46/100
6985/6985 [==============================] - 2s 336us/step - loss: 0.6785 -
acc: 0.7682 - val_loss: 0.5208 - val_acc: 0.8340

Epoch 00046: val_loss did not improve from 0.50350
Epoch 47/100
```

```
6985/6985 [==============================] - 2s 330us/step - loss: 0.6712 -
acc: 0.7754 - val_loss: 0.4978 - val_acc: 0.8477

Epoch 00047: val_loss improved from 0.50350 to 0.49784, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 48/100
6985/6985 [==============================] - 2s 332us/step - loss: 0.6736 -
acc: 0.7701 - val_loss: 0.4905 - val_acc: 0.8426

Epoch 00048: val_loss improved from 0.49784 to 0.49050, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 49/100
6985/6985 [==============================] - 2s 335us/step - loss: 0.6382 -
acc: 0.7787 - val_loss: 0.4851 - val_acc: 0.8477

Epoch 00049: val_loss improved from 0.49050 to 0.48515, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 50/100
6985/6985 [==============================] - 2s 331us/step - loss: 0.6423 -
acc: 0.7853 - val_loss: 0.5047 - val_acc: 0.8294

Epoch 00050: val_loss did not improve from 0.48515
Epoch 51/100
6985/6985 [==============================] - 2s 339us/step - loss: 0.6426 -
acc: 0.7847 - val_loss: 0.4914 - val_acc: 0.8454

Epoch 00051: val_loss did not improve from 0.48515
Epoch 52/100
6985/6985 [==============================] - 2s 338us/step - loss: 0.6233 -
acc: 0.7847 - val_loss: 0.4822 - val_acc: 0.8546

Epoch 00052: val_loss improved from 0.48515 to 0.48218, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 53/100
6985/6985 [==============================] - 2s 345us/step - loss: 0.6257 -
acc: 0.7851 - val_loss: 0.4939 - val_acc: 0.8443

Epoch 00053: val_loss did not improve from 0.48218
Epoch 54/100
6985/6985 [==============================] - 2s 346us/step - loss: 0.6243 -
acc: 0.7884 - val_loss: 0.4835 - val_acc: 0.8506

Epoch 00054: val_loss did not improve from 0.48218
Epoch 55/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.6167 -
acc: 0.7871 - val_loss: 0.4682 - val_acc: 0.8552
```

```
Epoch 00055: val_loss improved from 0.48218 to 0.46823, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 56/100
6985/6985 [==============================] - 2s 350us/step - loss: 0.6204 -
acc: 0.7875 - val_loss: 0.4837 - val_acc: 0.8558

Epoch 00056: val_loss did not improve from 0.46823
Epoch 57/100
6985/6985 [==============================] - 2s 353us/step - loss: 0.6257 -
acc: 0.7790 - val_loss: 0.4628 - val_acc: 0.8592

Epoch 00057: val_loss improved from 0.46823 to 0.46283, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 58/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.6344 -
acc: 0.7797 - val_loss: 0.4570 - val_acc: 0.8626

Epoch 00058: val_loss improved from 0.46283 to 0.45701, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 59/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.6024 -
acc: 0.7931 - val_loss: 0.4806 - val_acc: 0.8489

Epoch 00059: val_loss did not improve from 0.45701
Epoch 60/100
6985/6985 [==============================] - 2s 352us/step - loss: 0.6021 -
acc: 0.7970 - val_loss: 0.4691 - val_acc: 0.8701

Epoch 00060: val_loss did not improve from 0.45701
Epoch 61/100
6985/6985 [==============================] - 2s 350us/step - loss: 0.5985 -
acc: 0.7936 - val_loss: 0.4859 - val_acc: 0.8506

Epoch 00061: val_loss did not improve from 0.45701
Epoch 62/100
6985/6985 [==============================] - 2s 347us/step - loss: 0.5855 -
acc: 0.7974 - val_loss: 0.4725 - val_acc: 0.8580

Epoch 00062: val_loss did not improve from 0.45701
Epoch 63/100
6985/6985 [==============================] - 3s 365us/step - loss: 0.6021 -
acc: 0.7885 - val_loss: 0.4705 - val_acc: 0.8512

Epoch 00063: val_loss did not improve from 0.45701
Epoch 64/100
6985/6985 [==============================] - 2s 352us/step - loss: 0.5974 -
acc: 0.7960 - val_loss: 0.4710 - val_acc: 0.8535
```

```
Epoch 00064: val_loss did not improve from 0.45701
Epoch 65/100
6985/6985 [==============================] - 2s 350us/step - loss: 0.6117 -
acc: 0.7914 - val_loss: 0.4814 - val_acc: 0.8558

Epoch 00065: val_loss did not improve from 0.45701
Epoch 66/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.5910 -
acc: 0.8013 - val_loss: 0.4649 - val_acc: 0.8592

Epoch 00066: val_loss did not improve from 0.45701
Epoch 67/100
6985/6985 [==============================] - 3s 367us/step - loss: 0.6112 -
acc: 0.7976 - val_loss: 0.4594 - val_acc: 0.8598

Epoch 00067: val_loss did not improve from 0.45701
Epoch 68/100
6985/6985 [==============================] - 3s 371us/step - loss: 0.5852 -
acc: 0.8016 - val_loss: 0.4869 - val_acc: 0.8443

Epoch 00068: val_loss did not improve from 0.45701
Epoch 69/100
6985/6985 [==============================] - 3s 370us/step - loss: 0.5918 -
acc: 0.7999 - val_loss: 0.4709 - val_acc: 0.8529

Epoch 00069: val_loss did not improve from 0.45701
Epoch 70/100
6985/6985 [==============================] - 2s 354us/step - loss: 0.5636 -
acc: 0.8133 - val_loss: 0.4491 - val_acc: 0.8649

Epoch 00070: val_loss improved from 0.45701 to 0.44906, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 71/100
6985/6985 [==============================] - 2s 348us/step - loss: 0.6037 -
acc: 0.8010 - val_loss: 0.4600 - val_acc: 0.8638

Epoch 00071: val_loss did not improve from 0.44906
Epoch 72/100
6985/6985 [==============================] - 3s 367us/step - loss: 0.5755 -
acc: 0.8040 - val_loss: 0.4379 - val_acc: 0.8695

Epoch 00072: val_loss improved from 0.44906 to 0.43790, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 73/100
6985/6985 [==============================] - 3s 364us/step - loss: 0.5781 -
acc: 0.8020 - val_loss: 0.4415 - val_acc: 0.8575

Epoch 00073: val_loss did not improve from 0.43790
Epoch 74/100
6985/6985 [==============================] - 3s 364us/step - loss: 0.5757 -
acc: 0.8064 - val_loss: 0.4409 - val_acc: 0.8649

Epoch 00074: val_loss did not improve from 0.43790
Epoch 75/100
6985/6985 [==============================] - 2s 353us/step - loss: 0.5687 -
acc: 0.8145 - val_loss: 0.4505 - val_acc: 0.8741

Epoch 00075: val_loss did not improve from 0.43790
Epoch 76/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.5500 -
acc: 0.8143 - val_loss: 0.4285 - val_acc: 0.8672

Epoch 00076: val_loss improved from 0.43790 to 0.42855, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 77/100
6985/6985 [==============================] - 3s 418us/step - loss: 0.5870 -
acc: 0.7994 - val_loss: 0.4398 - val_acc: 0.8712

Epoch 00077: val_loss did not improve from 0.42855
Epoch 78/100
6985/6985 [==============================] - 3s 395us/step - loss: 0.5718 -
acc: 0.8056 - val_loss: 0.4423 - val_acc: 0.8706

Epoch 00078: val_loss did not improve from 0.42855
Epoch 79/100
6985/6985 [==============================] - 3s 359us/step - loss: 0.5768 -
acc: 0.8057 - val_loss: 0.4479 - val_acc: 0.8706

Epoch 00079: val_loss did not improve from 0.42855
Epoch 80/100
6985/6985 [==============================] - 2s 338us/step - loss: 0.5719 -
acc: 0.8070 - val_loss: 0.4340 - val_acc: 0.8701

Epoch 00080: val_loss did not improve from 0.42855
Epoch 81/100
6985/6985 [==============================] - 2s 347us/step - loss: 0.5600 -
acc: 0.8122 - val_loss: 0.4380 - val_acc: 0.8632

Epoch 00081: val_loss did not improve from 0.42855
Epoch 82/100
6985/6985 [==============================] - 3s 366us/step - loss: 0.5541 -
acc: 0.8149 - val_loss: 0.4205 - val_acc: 0.8729

Epoch 00082: val_loss improved from 0.42855 to 0.42049, saving model to sav
ed_models/weights.best.basic_mlp.hdf5
Epoch 83/100
```

```
6985/6985 [==============================] - 3s 386us/step - loss: 0.5528 -
acc: 0.8107 - val_loss: 0.4254 - val_acc: 0.8792

Epoch 00083: val_loss did not improve from 0.42049
Epoch 84/100
6985/6985 [==============================] - 3s 480us/step - loss: 0.5465 -
acc: 0.8135 - val_loss: 0.4517 - val_acc: 0.8632

Epoch 00084: val_loss did not improve from 0.42049
Epoch 85/100
6985/6985 [==============================] - 3s 443us/step - loss: 0.5402 -
acc: 0.8147 - val_loss: 0.4400 - val_acc: 0.8724

Epoch 00085: val_loss did not improve from 0.42049
Epoch 86/100
6985/6985 [==============================] - 3s 478us/step - loss: 0.5569 -
acc: 0.8097 - val_loss: 0.4488 - val_acc: 0.8626

Epoch 00086: val_loss did not improve from 0.42049
Epoch 87/100
6985/6985 [==============================] - 3s 413us/step - loss: 0.5529 -
acc: 0.8127 - val_loss: 0.4429 - val_acc: 0.8620

Epoch 00087: val_loss did not improve from 0.42049
Epoch 88/100
6985/6985 [==============================] - 3s 396us/step - loss: 0.5717 -
acc: 0.8125 - val_loss: 0.4551 - val_acc: 0.8649

Epoch 00088: val_loss did not improve from 0.42049
Epoch 89/100
6985/6985 [==============================] - 3s 383us/step - loss: 0.5520 -
acc: 0.8188 - val_loss: 0.4413 - val_acc: 0.8769

Epoch 00089: val_loss did not improve from 0.42049
Epoch 90/100
6985/6985 [==============================] - 3s 365us/step - loss: 0.5566 -
acc: 0.8106 - val_loss: 0.4512 - val_acc: 0.8626

Epoch 00090: val_loss did not improve from 0.42049
Epoch 91/100
6985/6985 [==============================] - 3s 361us/step - loss: 0.5526 -
acc: 0.8125 - val_loss: 0.4453 - val_acc: 0.8638

Epoch 00091: val_loss did not improve from 0.42049
Epoch 92/100
6985/6985 [==============================] - 3s 384us/step - loss: 0.5487 -
acc: 0.8110 - val_loss: 0.4290 - val_acc: 0.8735
```

```
Epoch 00092: val_loss did not improve from 0.42049
Epoch 93/100
6985/6985 [==============================] - 3s 365us/step - loss: 0.5279 -
acc: 0.8245 - val_loss: 0.4223 - val_acc: 0.8724

Epoch 00093: val_loss did not improve from 0.42049
Epoch 94/100
6985/6985 [==============================] - 3s 365us/step - loss: 0.5412 -
acc: 0.8219 - val_loss: 0.4317 - val_acc: 0.8804

Epoch 00094: val_loss did not improve from 0.42049
Epoch 95/100
6985/6985 [==============================] - 3s 376us/step - loss: 0.5393 -
acc: 0.8210 - val_loss: 0.4422 - val_acc: 0.8643

Epoch 00095: val_loss did not improve from 0.42049
Epoch 96/100
6985/6985 [==============================] - 3s 376us/step - loss: 0.5327 -
acc: 0.8203 - val_loss: 0.4266 - val_acc: 0.8689

Epoch 00096: val_loss did not improve from 0.42049
Epoch 97/100
6985/6985 [==============================] - 2s 334us/step - loss: 0.5525 -
acc: 0.8125 - val_loss: 0.4370 - val_acc: 0.8626

Epoch 00097: val_loss did not improve from 0.42049
Epoch 98/100
6985/6985 [==============================] - 2s 329us/step - loss: 0.5246 -
acc: 0.8241 - val_loss: 0.4338 - val_acc: 0.8620

Epoch 00098: val_loss did not improve from 0.42049
Epoch 99/100
6985/6985 [==============================] - 2s 347us/step - loss: 0.5346 -
acc: 0.8169 - val_loss: 0.4457 - val_acc: 0.8586

Epoch 00099: val_loss did not improve from 0.42049
Epoch 100/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.5413 -
acc: 0.8153 - val_loss: 0.4306 - val_acc: 0.8764

Epoch 00100: val_loss did not improve from 0.42049
Training completed in time:  0:04:15.582298
```

### 3.2.4 Test the model

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

In [6]:

```
# 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:  0.9252684323550465
Testing Accuracy:  0.8763594734511787
```

The initial Training and Testing accuracy scores are quite high. As there is not a great difference between the Training and Test scores (~5%) this suggests that the model has not suffered from overfitting.

### 3.2.5 Predictions

Here we will build a method which will allow us to test the models predictions on a specified audio .wav file.

In [7]:

```
import librosa
import numpy as np

def extract_feature(file_name):

    try:
        audio_data, sample_rate = librosa.load(file_name, res_type='kaiser_
fast')
        mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=4
0)
        mfccsscaled = np.mean(mfccs.T,axis=0)

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

    return np.array([mfccsscaled])
```

In [8]:

```
def print_prediction(file_name):
    prediction_feature = extract_feature(file_name)

    predicted_vector = model.predict_classes(prediction_feature)
    predicted_class = le.inverse_transform(predicted_vector)
    print("The predicted class is:", predicted_class[0], '\n')

    predicted_proba_vector = model.predict_proba(prediction_feature)
    predicted_proba = predicted_proba_vector[0]
    for i in range(len(predicted_proba)):
        category = le.inverse_transform(np.array([i]))
        print(category[0], "\t\t : ", format(predicted_proba[i], '.32f') )
```

### 3.2.6 Validation

**Test with sample data** Initial sanity check to verify the predictions using a subsection of the sample audio files we explored in the first notebook. We expect the bulk of these to be classified correctly.

```
# Class: Air Conditioner

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav'
print_prediction(filename)
The predicted class is: air_conditioner

air_conditioner                 :  0.99989426136016845703125000000000
car_horn                :  0.00000715439318810240365564823151
children_playing                :  0.00001791530303307808935642242432
dog_bark                :  0.00000256554130828590132296085 36
drilling                :  0.00000283992426375334616750478745
engine_idling           :  0.00005887898078071884810924530029
gun_shot                :  0.00000001620782441591472888831049
jackhammer              :  0.00000035662964137372910045087337
siren           :  0.00000004348472515403045690618455
street_music            :  0.00001830780820455402135848999023




# Class: Drilling

filename = '../UrbanSound Dataset sample/audio/103199-4-0-0.wav'
print_prediction(filename)
The predicted class is: drilling

air_conditioner                 :  0.00000000003295356001964400149973
car_horn                :  0.00000000308959258177310402970761
children_playing                :  0.00002208830665040295571088790894
dog_bark                :  0.00000067401481373963179066777229
drilling                :  0.99972504377365112304687500000000
engine_idling           :  0.00000000002312424904338250541969
gun_shot                :  0.00000014346949228638550266623497
jackhammer              :  0.00000000029780389265710027757450
siren           :  0.00000000156893398273183493074612
street_music            :  0.00025209947489202022552490234375
```

In [11]:

```
# Class: Street music
```

```
filename = '../UrbanSound Dataset sample/audio/101848-9-0-0.wav'
print_prediction(filename)
```

```
The predicted class is: street_music

air_conditioner                 :  0.09999072551727294921875000000000
car_horn                :  0.00305506144650280475616455078125
children_playing                :  0.09950152784585952758789062500000
dog_bark                :  0.02582867257297039031982421875000
drilling                :  0.00509325042366981506347656250000
engine_idling           :  0.00916280318051576614379882812500
gun_shot                :  0.00549275847151875495910644531250
jackhammer              :  0.03270008042454719543457031250000
siren           :  0.00361734302714467048645019531250
street_music            :  0.71555775403976440429687500000000
```

In [12]:

```
# Class: Car Horn

filename = '../UrbanSound Dataset sample/audio/100648-1-0-0.wav'
print_prediction(filename)
```

```
The predicted class is: car_horn

air_conditioner                 :  0.00188611494377255439758300781250
car_horn                :  0.68632853031158447265625000000000
children_playing                :  0.01224335655570030212402343750000
dog_bark                :  0.16461659967899322509765625000000
drilling                :  0.05645351111888885498046875000000
engine_idling           :  0.00212736334651708602905273437500
gun_shot                :  0.00211420282721519470214843750000
jackhammer              :  0.00372551172040402889251708984375
siren           :  0.00587591761723160743713378906250
street_music            :  0.06462877988815307617187500000000
```

**Observations** From this brief sanity check the model seems to predict well. One error was observed whereby a car horn was incorrectly classified as a dog bark.
We can see from the per class confidence that this was quite a low score (43%). This allows follows our early observation that a dog bark and car horn are similar in spectral shape.

## 3.2.7 Other audio

Here we will use a sample of various copyright free sounds that we not part of either our test or training data to further validate our model.

In [13]:

```
filename = '../Evaluation audio/dog_bark_1.wav'
print_prediction(filename)
```

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

```
air_conditioner                 :  0.00038618501275777816772460937500
car_horn                :  0.00915508810430765151977539062500
children_playing                :  0.06478454917669296264648437500000
dog_bark                :  0.71007812023162841796875000000000
drilling                :  0.02283692173659801483154296875000
engine_idling           :  0.00240809586830437183380126953125
gun_shot                :  0.10433794558048248291015625000000
jackhammer              :  0.00001514166433480568230152130127
siren           :  0.01288078445941209793090820312500
street_music            :  0.07311715185642242431640625000000
```

In [14]:

```python
filename = '../Evaluation audio/drilling_1.wav'

print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                 :  0.32110649347305297851562500000000
car_horn                :  0.00000022923920539597020251676440
children_playing                :  0.00010408438356535043567419052112
dog_bark                :  0.00000026054382828988309483975172
drilling                :  0.66649377346038818359375000000000
engine_idling           :  0.00000000133662025891823077472509
gun_shot                :  0.00000043437574959170888178050518
jackhammer              :  0.01238841470330953598022460937500
siren           :  0.00000000000289116074830841895959 6
street_music            :  0.00000000052894209012777082534739 7
```

In [15]:

```python
filename = '../Evaluation audio/gun_shot_1.wav'

print_prediction(filename)

# sample data weighted towards gun shot - peak in the dog barking sample is
simmilar in shape to the gun shot sample
```

```
The predicted class is: dog_bark

air_conditioner                 :  0.02008811198174953460693359375000
car_horn                :  0.00047429648111574351787567138672
children_playing                :  0.00094942341092973947525024414062
dog_bark                :  0.53654015064239501953125000000000
drilling                :  0.00093174201902002096176147460938
engine_idling           :  0.03123776055872440338134765625000
gun_shot                :  0.00091215252177789807319641113281
jackhammer              :  0.00002015420614043250679969787598
siren           :  0.00055970775429159402847290039062
street_music            :  0.40828645229339599609375000000000
```

In [16]:

```python
filename = '../Evaluation audio/siren_1.wav'
```

```
print_prediction(filename)

The predicted class is: siren

air_conditioner                 :  0.00000732402349967742338776588440
car_horn                :  0.00057092373026534914970397949219
children_playing                :  0.00199068244546651840209960937500
dog_bark                :  0.02090488374233245849609375000000
drilling                :  0.00046552356798201799392700195312
engine_idling           :  0.14164580404758453369140625000000
gun_shot                :  0.00050196843221783638000488281250
jackhammer              :  0.00276053301058709621429443359375
siren           :  0.81527197360992431640625000000000
street_music            :  0.01588040776550769805908203125000
```

***Observations***

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

## 3.3 Refinement

In our inital attempt, we were able to achieve a Classification Accuracy score of:

- Training data Accuracy: 92.3%
- Testing data Accuracy: 87%

We will now see if we can improve upon that score using a Convolutional Neural Network (CNN).

***Feature Extraction refinement***

In the prevous feature extraction stage, the MFCC vectors would vary in size for the different audio files (depending on the samples duration).

However, CNNs require a fixed size for all inputs. To overcome this we will zero pad the output vectors to make them all the same size.

```
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='con
stant')

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

    return mfccs
```

In [7]:

```
# Load various imports
import pandas as pd
import os
import librosa

# Set the path to the full UrbanSound dataset
fulldatasetpath = '/Volumes/Untitled/ML_Data/Urban Sound/UrbanSound8K/audio
/'

metadata = pd.read_csv('../UrbanSound Dataset sample/metadata/UrbanSound8K.
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),'fold'+str(ro
w["fold"])+'/',str(row["slice_file_name"]))

    class_label = row["class_name"]
    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  8732  files
```

In [62]:

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

# Convert features and corresponding classification labels into numpy array
s
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))

# 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=0.2, r
andom_state = 42)
```

### 3.3.1 Convolutional Neural Network (CNN) model architecture

We will modify our model to be a Convolutional Neural Network (CNN) again using Keras and a Tensorflow backend.

Again we will use a `sequential` model, starting with a simple model architecture, consisting of four `Conv2D` convolution layers, with our final output layer being a `dense` layer.

The convolution layers are designed for feature detection. It works by sliding a filter window over the input and performing a matrix multiplication and storing the result in a feature map. This operation is known as a convolution.

The `filter` parameter specifies the number of nodes in each layer. Each layer will increase in size from 16, 32, 64 to 128, while the `kernel_size` parameter specifies the size of the kernel window which in this case is 2 resulting in a 2x2 filter matrix.

The first layer will receive the input shape of (40, 174, 1) where 40 is the number of MFCC's 174 is the number of frames taking padding into account and the 1 signifying that the audio is mono.

The activation function we will be using for our convolutional layers is `ReLU` which is the same as our previous model. We will use a smaller `Dropout` value of 20% on our convolutional layers.

Each convolutional layer has an associated pooling layer of `MaxPooling2D` type with the final convolutional layer having a `GlobalAveragePooling2D` type. The pooling layer is do reduce the dimensionality of the model (by reducing the parameters and subsquent computation requirements) which serves to shorten the training time and reduce overfitting. The Max Pooling type takes the maximum size for each window and the Global Average Pooling type takes the average which is suitable for feeding into our `dense` output layer.

Our output layer will have 10 nodes (num_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.

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, Conv2D, MaxPooling2D, GlobalAve
ragePooling2D
from keras.optimizers import Adam
from keras.utils import np_utils
from sklearn import metrics

num_rows = 40
num_columns = 174
num_channels = 1

x_train = x_train.reshape(x_train.shape[0], num_rows, num_columns, num_
channels)
x_test = x_test.reshape(x_test.shape[0], num_rows, num_columns, num_cha
nnels)

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'))
```

### 3.3.2 Compiling the model

For compiling our model, we will use the same three parameters as the previous model:

In [44]:

```
# Compile the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimi
zer='adam')
```

In [45]:

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

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_11 (Conv2D)           (None, 39, 173, 16)       80        
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 19, 86, 16)        0         
_________________________________________________________________
dropout_17 (Dropout)         (None, 19, 86, 16)        0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 18, 85, 32)        2080      
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 9, 42, 32)         0         
_________________________________________________________________
dropout_18 (Dropout)         (None, 9, 42, 32)         0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 41, 64)         8256      
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 4, 20, 64)         0         
_________________________________________________________________
dropout_19 (Dropout)         (None, 4, 20, 64)         0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 3, 19, 128)        32896     
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 1, 9, 128)         0         
_________________________________________________________________
dropout_20 (Dropout)         (None, 1, 9, 128)         0         
_________________________________________________________________
global_average_pooling2d_1 ( (None, 128)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290      
=================================================================
Total params: 44,602
Trainable params: 44,602
Non-trainable params: 0
_________________________________________________________________
1747/1747 [==============================] - 9s 5ms/step
Pre-training accuracy: 12.0206%
```

### 3.3.3 Training

Here we will train the model. As training a CNN can take a sigificant amount of time, we will start with a low number of epochs and a low batch size. If we can see from the output that the model is converging, we will increase both numbers.

In [60]:

```
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='saved_models/weights.best.basic_cn
n.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

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


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

```
Train on 6985 samples, validate on 1747 samples
Epoch 1/72
6985/6985 [==============================] - 76s 11ms/step - loss: 0.2628 -
acc: 0.9085 - val_loss: 0.3790 - val_acc: 0.8775

Epoch 00001: val_loss improved from inf to 0.37902, saving model to saved_m
odels/weights.best.basic_cnn.hdf5
Epoch 2/72
6985/6985 [==============================] - 73s 10ms/step - loss: 0.2660 -
acc: 0.9059 - val_loss: 0.3559 - val_acc: 0.8878

Epoch 00002: val_loss improved from 0.37902 to 0.35589, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 3/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2410 -
acc: 0.9184 - val_loss: 0.3456 - val_acc: 0.8930

Epoch 00003: val_loss improved from 0.35589 to 0.34559, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 4/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2464 -
acc: 0.9101 - val_loss: 0.3377 - val_acc: 0.8941

Epoch 00004: val_loss improved from 0.34559 to 0.33769, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 5/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2316 -
acc: 0.9152 - val_loss: 0.3458 - val_acc: 0.8930

Epoch 00005: val_loss did not improve from 0.33769
Epoch 6/72
```

```
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2376 -
acc: 0.9144 - val_loss: 0.3508 - val_acc: 0.8844

Epoch 00006: val_loss did not improve from 0.33769
Epoch 7/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2501 -
acc: 0.9145 - val_loss: 0.3896 - val_acc: 0.8735

Epoch 00007: val_loss did not improve from 0.33769
Epoch 8/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2270 -
acc: 0.9158 - val_loss: 0.3549 - val_acc: 0.8878

Epoch 00008: val_loss did not improve from 0.33769
Epoch 9/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2331 -
acc: 0.9211 - val_loss: 0.3394 - val_acc: 0.8890

Epoch 00009: val_loss did not improve from 0.33769
Epoch 10/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2351 -
acc: 0.9167 - val_loss: 0.3328 - val_acc: 0.8912

Epoch 00010: val_loss improved from 0.33769 to 0.33276, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 11/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2269 -
acc: 0.9187 - val_loss: 0.3289 - val_acc: 0.9021

Epoch 00011: val_loss improved from 0.33276 to 0.32894, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 12/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2144 -
acc: 0.9264 - val_loss: 0.3220 - val_acc: 0.9015

Epoch 00012: val_loss improved from 0.32894 to 0.32195, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 13/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2214 -
acc: 0.9237 - val_loss: 0.3807 - val_acc: 0.8867

Epoch 00013: val_loss did not improve from 0.32195
Epoch 14/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.2173 -
acc: 0.9246 - val_loss: 0.3123 - val_acc: 0.9033

Epoch 00014: val_loss improved from 0.32195 to 0.31233, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
```

```
Epoch 15/72
6985/6985 [==============================] - 73s 10ms/step - loss: 0.2070 -
acc: 0.9270 - val_loss: 0.3089 - val_acc: 0.9027

Epoch 00015: val_loss improved from 0.31233 to 0.30894, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 16/72
6985/6985 [==============================] - 79s 11ms/step - loss: 0.2148 -
acc: 0.9254 - val_loss: 0.3139 - val_acc: 0.9033

Epoch 00016: val_loss did not improve from 0.30894
Epoch 17/72
6985/6985 [==============================] - 73s 10ms/step - loss: 0.2046 -
acc: 0.9251 - val_loss: 0.3173 - val_acc: 0.9015

Epoch 00017: val_loss did not improve from 0.30894
Epoch 18/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2045 -
acc: 0.9250 - val_loss: 0.3344 - val_acc: 0.8998

Epoch 00018: val_loss did not improve from 0.30894
Epoch 19/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1988 -
acc: 0.9316 - val_loss: 0.3075 - val_acc: 0.9107

Epoch 00019: val_loss improved from 0.30894 to 0.30752, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 20/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2026 -
acc: 0.9286 - val_loss: 0.3416 - val_acc: 0.9056

Epoch 00020: val_loss did not improve from 0.30752
Epoch 21/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.2062 -
acc: 0.9266 - val_loss: 0.3123 - val_acc: 0.9073

Epoch 00021: val_loss did not improve from 0.30752
Epoch 22/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1956 -
acc: 0.9320 - val_loss: 0.2969 - val_acc: 0.9107

Epoch 00022: val_loss improved from 0.30752 to 0.29689, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 23/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1992 -
acc: 0.9297 - val_loss: 0.3282 - val_acc: 0.9061

Epoch 00023: val_loss did not improve from 0.29689
Epoch 24/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1990 -
acc: 0.9298 - val_loss: 0.3500 - val_acc: 0.8981

Epoch 00024: val_loss did not improve from 0.29689
Epoch 25/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1906 -
acc: 0.9359 - val_loss: 0.3025 - val_acc: 0.9113

Epoch 00025: val_loss did not improve from 0.29689
Epoch 26/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1808 -
acc: 0.9369 - val_loss: 0.3063 - val_acc: 0.9067

Epoch 00026: val_loss did not improve from 0.29689
Epoch 27/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1890 -
acc: 0.9369 - val_loss: 0.3290 - val_acc: 0.9033

Epoch 00027: val_loss did not improve from 0.29689
Epoch 28/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1803 -
acc: 0.9360 - val_loss: 0.2824 - val_acc: 0.9107

Epoch 00028: val_loss improved from 0.29689 to 0.28240, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 29/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1795 -
acc: 0.9374 - val_loss: 0.4025 - val_acc: 0.8792

Epoch 00029: val_loss did not improve from 0.28240
Epoch 30/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1828 -
acc: 0.9372 - val_loss: 0.3079 - val_acc: 0.9084

Epoch 00030: val_loss did not improve from 0.28240
Epoch 31/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1858 -
acc: 0.9363 - val_loss: 0.3268 - val_acc: 0.8987

Epoch 00031: val_loss did not improve from 0.28240
Epoch 32/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1787 -
acc: 0.9390 - val_loss: 0.3239 - val_acc: 0.9004

Epoch 00032: val_loss did not improve from 0.28240
Epoch 33/72
```

```
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1694 -
acc: 0.9413 - val_loss: 0.3237 - val_acc: 0.9078

Epoch 00033: val_loss did not improve from 0.28240
Epoch 34/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1689 -
acc: 0.9380 - val_loss: 0.3053 - val_acc: 0.9056

Epoch 00034: val_loss did not improve from 0.28240
Epoch 35/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1621 -
acc: 0.9466 - val_loss: 0.3185 - val_acc: 0.9090

Epoch 00035: val_loss did not improve from 0.28240
Epoch 36/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1570 -
acc: 0.9449 - val_loss: 0.2865 - val_acc: 0.9113

Epoch 00036: val_loss did not improve from 0.28240
Epoch 37/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1571 -
acc: 0.9469 - val_loss: 0.3076 - val_acc: 0.9084

Epoch 00037: val_loss did not improve from 0.28240
Epoch 38/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1523 -
acc: 0.9472 - val_loss: 0.3855 - val_acc: 0.8958

Epoch 00038: val_loss did not improve from 0.28240
Epoch 39/72
6985/6985 [==============================] - 72s 10ms/step - loss: 0.1620 -
acc: 0.9453 - val_loss: 0.3272 - val_acc: 0.8952

Epoch 00039: val_loss did not improve from 0.28240
Epoch 40/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1613 -
acc: 0.9449 - val_loss: 0.3224 - val_acc: 0.9067

Epoch 00040: val_loss did not improve from 0.28240
Epoch 41/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1641 -
acc: 0.9417 - val_loss: 0.2917 - val_acc: 0.9199

Epoch 00041: val_loss did not improve from 0.28240
Epoch 42/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1606 -
acc: 0.9440 - val_loss: 0.2793 - val_acc: 0.9187
```

```
Epoch 00042: val_loss improved from 0.28240 to 0.27932, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 43/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1505 -
acc: 0.9476 - val_loss: 0.2989 - val_acc: 0.9084

Epoch 00043: val_loss did not improve from 0.27932
Epoch 44/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1577 -
acc: 0.9432 - val_loss: 0.3483 - val_acc: 0.9056

Epoch 00044: val_loss did not improve from 0.27932
Epoch 45/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1540 -
acc: 0.9475 - val_loss: 0.2879 - val_acc: 0.9170

Epoch 00045: val_loss did not improve from 0.27932
Epoch 46/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1461 -
acc: 0.9490 - val_loss: 0.3188 - val_acc: 0.9113

Epoch 00046: val_loss did not improve from 0.27932
Epoch 47/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1479 -
acc: 0.9472 - val_loss: 0.3203 - val_acc: 0.9073

Epoch 00047: val_loss did not improve from 0.27932
Epoch 48/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1637 -
acc: 0.9450 - val_loss: 0.3045 - val_acc: 0.9050

Epoch 00048: val_loss did not improve from 0.27932
Epoch 49/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1511 -
acc: 0.9473 - val_loss: 0.2745 - val_acc: 0.9256

Epoch 00049: val_loss improved from 0.27932 to 0.27453, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 50/72
6985/6985 [==============================] - 71s 10ms/step - loss: 0.1376 -
acc: 0.9513 - val_loss: 0.3237 - val_acc: 0.9141

Epoch 00050: val_loss did not improve from 0.27453
Epoch 51/72
6985/6985 [==============================] - 70s 10ms/step - loss: 0.1385 -
acc: 0.9515 - val_loss: 0.3292 - val_acc: 0.9067

Epoch 00051: val_loss did not improve from 0.27453
Epoch 52/72
6985/6985 [==============================] - 10691s 2s/step - loss: 0.1424
- acc: 0.9485 - val_loss: 0.2846 - val_acc: 0.9181

Epoch 00052: val_loss did not improve from 0.27453
Epoch 53/72
6985/6985 [==============================] - 176s 25ms/step - loss: 0.1337
- acc: 0.9533 - val_loss: 0.2813 - val_acc: 0.9164

Epoch 00053: val_loss did not improve from 0.27453
Epoch 54/72
6985/6985 [==============================] - 176s 25ms/step - loss: 0.1529
- acc: 0.9466 - val_loss: 0.3030 - val_acc: 0.9141

Epoch 00054: val_loss did not improve from 0.27453
Epoch 55/72
6985/6985 [==============================] - 177s 25ms/step - loss: 0.1392
- acc: 0.9523 - val_loss: 0.2990 - val_acc: 0.9130

Epoch 00055: val_loss did not improve from 0.27453
Epoch 56/72
6985/6985 [==============================] - 181s 26ms/step - loss: 0.1461
- acc: 0.9509 - val_loss: 0.3298 - val_acc: 0.9010

Epoch 00056: val_loss did not improve from 0.27453
Epoch 57/72
6985/6985 [==============================] - 179s 26ms/step - loss: 0.1330
- acc: 0.9542 - val_loss: 0.3374 - val_acc: 0.9084

Epoch 00057: val_loss did not improve from 0.27453
Epoch 58/72
6985/6985 [==============================] - 182s 26ms/step - loss: 0.1321
- acc: 0.9529 - val_loss: 0.3465 - val_acc: 0.9061

Epoch 00058: val_loss did not improve from 0.27453
Epoch 59/72
6985/6985 [==============================] - 182s 26ms/step - loss: 0.1281
- acc: 0.9553 - val_loss: 0.2965 - val_acc: 0.9187

Epoch 00059: val_loss did not improve from 0.27453
Epoch 60/72
6985/6985 [==============================] - 182s 26ms/step - loss: 0.1393
- acc: 0.9503 - val_loss: 0.3104 - val_acc: 0.9136

Epoch 00060: val_loss did not improve from 0.27453
Epoch 61/72
6985/6985 [==============================] - 183s 26ms/step - loss: 0.1287
- acc: 0.9530 - val_loss: 0.2871 - val_acc: 0.9147
```

```
Epoch 00061: val_loss did not improve from 0.27453
Epoch 62/72
6985/6985 [==============================] - 185s 26ms/step - loss: 0.1410
- acc: 0.9508 - val_loss: 0.3286 - val_acc: 0.9124

Epoch 00062: val_loss did not improve from 0.27453
Epoch 63/72
6985/6985 [==============================] - 184s 26ms/step - loss: 0.1303
- acc: 0.9526 - val_loss: 0.3357 - val_acc: 0.9084

Epoch 00063: val_loss did not improve from 0.27453
Epoch 64/72
6985/6985 [==============================] - 186s 27ms/step - loss: 0.1350
- acc: 0.9503 - val_loss: 0.3213 - val_acc: 0.9147

Epoch 00064: val_loss did not improve from 0.27453
Epoch 65/72
6985/6985 [==============================] - 183s 26ms/step - loss: 0.1210
- acc: 0.9556 - val_loss: 0.2724 - val_acc: 0.9256

Epoch 00065: val_loss improved from 0.27453 to 0.27239, saving model to sav
ed_models/weights.best.basic_cnn.hdf5
Epoch 66/72
6985/6985 [==============================] - 184s 26ms/step - loss: 0.1216
- acc: 0.9596 - val_loss: 0.3336 - val_acc: 0.9101

Epoch 00066: val_loss did not improve from 0.27239
Epoch 67/72
6985/6985 [==============================] - 198s 28ms/step - loss: 0.1135
- acc: 0.9583 - val_loss: 0.2843 - val_acc: 0.9227

Epoch 00067: val_loss did not improve from 0.27239
Epoch 68/72
6985/6985 [==============================] - 265s 38ms/step - loss: 0.1245
- acc: 0.9572 - val_loss: 0.2972 - val_acc: 0.9147

Epoch 00068: val_loss did not improve from 0.27239
Epoch 69/72
6985/6985 [==============================] - 266s 38ms/step - loss: 0.1172
- acc: 0.9599 - val_loss: 0.3116 - val_acc: 0.9204

Epoch 00069: val_loss did not improve from 0.27239
Epoch 70/72
6985/6985 [==============================] - 265s 38ms/step - loss: 0.1213
- acc: 0.9568 - val_loss: 0.2978 - val_acc: 0.9164

Epoch 00070: val_loss did not improve from 0.27239
Epoch 71/72
6985/6985 [==============================] - 14289s 2s/step - loss: 0.1203
- acc: 0.9581 - val_loss: 0.2878 - val_acc: 0.9164

Epoch 00071: val_loss did not improve from 0.27239
Epoch 72/72
6985/6985 [==============================] - 92s 13ms/step - loss: 0.1147 -
acc: 0.9596 - val_loss: 0.3005 - val_acc: 0.9193

Epoch 00072: val_loss did not improve from 0.27239
Training completed in time:  8:57:38.203486
```

### 3.3.4 Test the model

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

In [61]:

```
# 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:  0.9819613457408733
Testing Accuracy:  0.9192902116210514
```

The Training and Testing accuracy scores are both high and an increase on our initial model. Training accuracy has increased by ~6% and Testing accuracy has increased by ~4%.

There is a marginal increase in the difference between the Training and Test scores (~6% compared to ~5% previously) though the difference remains low so the model has not suffered from overfitting.

### 3.3.5 Predictions

Here we will modify our previous method for testing the models predictions on a specified audio .wav file.

In [50]:

```
def print_prediction(file_name):
    prediction_feature = extract_features(file_name)
    prediction_feature = prediction_feature.reshape(1, num_rows, num_column
s, num_channels)

    predicted_vector = model.predict_classes(prediction_feature)
    predicted_class = le.inverse_transform(predicted_vector)
    print("The predicted class is:", predicted_class[0], '\n')

    predicted_proba_vector = model.predict_proba(prediction_feature)
    predicted_proba = predicted_proba_vector[0]
    for i in range(len(predicted_proba)):
        category = le.inverse_transform(np.array([i]))
        print(category[0], "\t\t : ", format(predicted_proba[i], '.32f') )
```

### 3.3.6 Validation

***Test with sample data***

As before we will verify the predictions using a subsection of the sample audio files we explored in the first notebook. We expect the bulk of these to be classified correctly.

In [51]:

```
# Class: Air Conditioner

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav'
print_prediction(filename)
The predicted class is: air_conditioner

air_conditioner                  :  0.90663295984268188476562500000000
car_horn                :  0.00003793123823925270698964595579
children_playing                 :  0.00372877437621355056762695312500
dog_bark                :  0.00003181818829034455120563507080
drilling                :  0.00387497572228312492370605468750
engine_idling           :  0.00299200275912880897521972656250
gun_shot                :  0.00765613839030265808105468750000
jackhammer              :  0.07329261302947998046875000000000
siren           :  0.00018024632299784570932388305664
street_music            :  0.00160682143177837133407592773438
```

In [52]:

```
# Class: Drilling

filename = '../UrbanSound Dataset sample/audio/103199-4-0-0.wav'
print_prediction(filename)
The predicted class is: drilling

air_conditioner                  :  0.00070991273969411849975585937500
car_horn                :  0.00000001777174851724794280016795
children_playing                 :  0.00001405069633619859814643859863
dog_bark                :  0.00000047111242906794359441846609
drilling                :  0.99598699808120727539062500000000
engine_idling           :  0.00000354658413925790227949619293
gun_shot                :  0.00000003223207656333215709310025
jackhammer              :  0.00052903906907886266708374023438
siren           :  0.00000098340262866258854046463966
street_music            :  0.00275487988255918025970458984375
```

In [53]:

```
# Class: Street music
```

```
filename = '../UrbanSound Dataset sample/audio/101848-9-0-0.wav'
print_prediction(filename)
The predicted class is: street_music

air_conditioner                 :  0.00011496015213197097182273864746
car_horn                :  0.00079288281267508864402770996094
children_playing                :  0.01791538484394550323486328125000
dog_bark                :  0.00257923710159957408905029296875
drilling                :  0.00007904539961600676178932189941
engine_idling           :  0.00006061193562345579266548156738
gun_shot                :  0.00000000007482268277181347571059
jackhammer              :  0.00000457825990451965481042861938
siren           :  0.00922307930886745452880859375000
street_music            :  0.96923023462295532226562500000000
```

In [64]:

```
# Class: Car Horn

filename = '../UrbanSound Dataset sample/audio/100648-1-0-0.wav'
print_prediction(filename)
The predicted class is: drilling

air_conditioner                 :  0.00059866637457162141799926757812
car_horn                :  0.26391193270683288574218750000000
children_playing                :  0.00126012135297060012817382812500
dog_bark                :  0.27843952178955078125000000000000
drilling                :  0.34817233681678771972656250000000
engine_idling           :  0.00339049054309725761413574218750
gun_shot                :  0.05176293104887008666992187500000
jackhammer              :  0.03859317675232887268066406250000
siren           :  0.01271206419914960861206054687500
street_music            :  0.00115874561015516519546508789062
```

***Observations***

We can see that the model performs well.

Interestingly, car horn was again incorrectly classifed but this time as drilling - though the per class confidence shows it was a close decision between car horn with 26% confidence and drilling at 34% confidence.

### 3.3.7 **Other audio**

Again we will further validate our model using a sample of various copyright free sounds that we not part of either our test or training data.

In [55]:

```
filename = '../Evaluation audio/dog_bark_1.wav'
```

```
print_prediction(filename)
The predicted class is: dog_bark

air_conditioner                 :  0.00053069164277985692024230957031
car_horn                :  0.01807974837720394134521484375000
children_playing                :  0.00958889070898294448852539062500
dog_bark                :  0.84292083978652954101562500000000
drilling                :  0.02251568622887134552001953125000
engine_idling           :  0.00286057707853615283966064453125
gun_shot                :  0.09233076870441436767578125000000
jackhammer              :  0.00147349410690367221832275390625
siren           :  0.00702858529984951019287109375000
street_music            :  0.00267084036022424697875976562500
```

In [66]:

```
filename = '../Evaluation audio/drilling_1.wav'

print_prediction(filename)
The predicted class is: jackhammer

air_conditioner                 :  0.07861315459012985229492187500000
car_horn                :  0.00000012394852433317282702773809
children_playing                :  0.00008794507266429718583822250370
dog_bark                :  0.00001840709501266246661543846130
drilling                :  0.00033784923289204016327857971190
engine_idling           :  0.06372328102588653564453125000000
gun_shot                :  0.00000011736039340348725090734661
jackhammer              :  0.85761523246765136718750000000000
siren           :  0.00003615087280195439234375953670
street_music            :  0.00000013487000671830173814669251
```

In [65]:

```
filename = '../Evaluation audio/gun_shot_1.wav'

print_prediction(filename)
The predicted class is: gun_shot

air_conditioner                 :  0.00000001711038777330031734891236
car_horn                :  0.00000002828730849557814508443698
children_playing                :  0.00001153892753791296854615211487
dog_bark                :  0.00006763751298421993851661682129
drilling                :  0.00002225582647952251136302947998
engine_idling           :  0.00000385214798370725475251674652
gun_shot                :  0.99988853931427001953125000000000
jackhammer              :  0.00000000060133342749679741245927
siren           :  0.00006033371391822583973407745360
street_music            :  0.00000002041979207945132657187060
```

# 4 Results

## 4.1 Model Evaluation and Validation

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. This architecture is described in detail in section 3.

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 classifier performs well with new data.
- Misclassification does occur but seems to be between classes that are relatively similar such as Drilling and Jackhammer.

## 4.2 Justification

The final model achieved a classification accuracy of 92% on the testing data which exceeded my expectations given the benchmark was 68%.

| **Model** | **Classification Accuracy** |
|---|---|
| CNN | 92% |
| MLP | 88% |
| Benchmark SVM_rbfc | 68% |

The final solution performs well when presented with a .wav file with a duration of a few seconds and returns a reliable classification.

However, we do not know how the model would perform on Real-time audio. We do not know

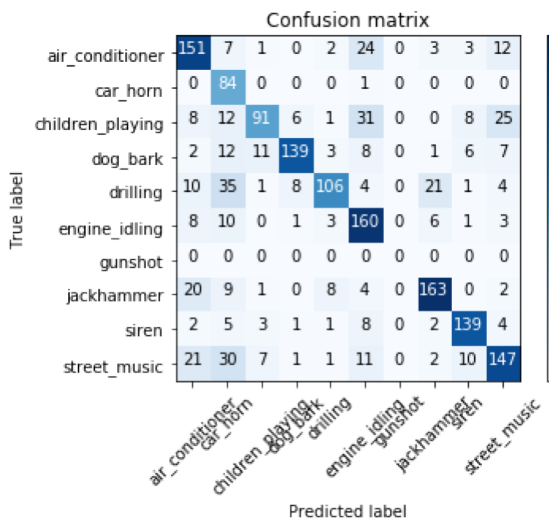
# 5 Conclusion

## 5.1 Free-Form Visualisation

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.

Using a confusion matrix we will examine if the final model also struggled to differentiate between these classes.



| | air_conditioner | car_horn | children_playing | dog_bark | drilling | engine_idling | gunshot | jackhammer | siren | street_music |
|---|---|---|---|---|---|---|---|---|---|---|
| air_conditioner | 151 | 7 | 1 | 0 | 2 | 24 | 0 | 3 | 3 | 12 |
| car_horn | 0 | 84 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| children_playing | 8 | 12 | 91 | 6 | 1 | 31 | 0 | 0 | 8 | 25 |
| dog_bark | 2 | 12 | 11 | 139 | 3 | 8 | 0 | 1 | 6 | 7 |
| drilling | 10 | 35 | 1 | 8 | 106 | 4 | 0 | 21 | 1 | 4 |
| engine_idling | 8 | 10 | 0 | 1 | 3 | 160 | 0 | 6 | 1 | 3 |
| gunshot | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| jackhammer | 20 | 9 | 1 | 0 | 8 | 4 | 0 | 163 | 0 | 2 |
| siren | 2 | 5 | 3 | 1 | 1 | 8 | 0 | 2 | 139 | 4 |
| street_music | 21 | 30 | 7 | 1 | 1 | 11 | 0 | 2 | 10 | 147 |

The Confusion Matrix tells a different story. Here we can see that our model struggles the most with the following sub-groups:

- air conditioner, jackhammer and street music.
- car horn, drilling, and street music.
- air conditioner, children playing and engine idling.
- jackhammer and drilling.
- air conditioner, car horn, children playing and street music.

This shows us that the problem is more nuanced than our initial assessment and gives some insights into the features that the CNN is extracting to make it's classifications. For example, street music is one of the commonly classified classes and could be to a wide variety of different samples within the class.

## 5.2 Reflection

The process used for this project can be summarised with the 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. An initial model was trained and evaluated.
5. A further model was trained and refined.
6. The final model was evaluated.

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 thanks to the Python tool Librosa. 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. However, we had to revisit the extraction process when we transitioned to using a CNN as our model. I did consider revisiting our MLP model to see how it performed with the updated feature extraction technique, but unfortunately there was not enough time for this.

Overall, the model performed better than planned. One observation we made during step 2 is that the dataset is slightly unbalanced with 2 out of the 10 classes having roughly a 40% sample size of the other 8. However, it is unclear whether this is significant enough to have caused any issues.

## 5.3 Improvement

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

- As previously mentioned, test the models performance with Real-time audio.
- Train the model for real world data. This would likely involve augmenting the training data

in various ways such as:

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

- Experiment to see if per-class accuracy is affected by using training data of different durations.
- Experiment with other techniques for feature extraction such as different forms of Spectrograms.

## References

[1] Justin Salamon, Christopher Jacoby and Juan Pablo Bello. Urban Sound Datasets. 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] Manik Soni AI which classifies Sounds:Code:Python. https://hackernoon.com/ai-which-classifies-sounds-code-python-6a07a2043810

[5] Manash Kumar Mandal Building a Dead Simple Speech Recognition
Engine using ConvNet in Keras. https://blog.manash.me/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b

[6] Eijaz Allibhai Building a Convolutional Neural Network (CNN) in Keras. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5

[7] Daphne Cornelisse An intuitive guide to Convolutional Neural Networks. https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

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

[9] wavsource.com THE Source for Free Sound Files and Reviews. http://www.wavsource.com/

[10] soundbible.com. http://soundbible.com//