Udacity Machine Learning Nanodegree 2019

Capstone Project

Classification of Urban sounds using Deep Learning Models

DEEPARTH GUPTA

1 PROJECT DEFINITION

1.1 PROJECT OVERVIEW

An average person's experience of life is filled with sound. Whether engaged in a conversation or going for a walk, our ears are always picking up sound and our brains are always processing them. It is an important adaptation too. Sound carries a lot of information about the environment that cannot be gleaned with just vision. Allowing any creature capable of hearing to be aware of their surroundings beyond their line of sight.

Automatic environmental sound classification is a growing area of research with many real-world applications. It is still nascent compared to other areas related to speech and music, hence, literature on it is relatively scarce. It could be useful to use techniques used in other domains in this one. In fact, there are examples of this happening.

One such example is the usage of Convolutional Neural Networks in sound processing and classification. Due to their ability to glean spatial relationships between features in an image, they are a great tool to analyze images and the spatial relationship of elements in them. One can train a CNN to analyze spectrographs of various sounds and classify them. CNNs are also able to classify sounds at very high accuracies even if there are multiple sounds mixed together.

1.2 PROBLEM STATEMENT

The main objective of this project is to classify common urban sounds using a deep learning model.
When input in the form of a short duration audio file is received, the project should be able to classify it into one of the target sounds.

There are a few ways to do this. My method will be to create spectrograms and use CNNs (Convolutional Neural Networks) to turn this into an image classification task. I was familiar with this concept before I started the project. So, there wasn't much need for me to explore.

1.3 METRICS

The evaluation metric for this problem will be plain old accuracy. Accuracy is defined as simply the number of correct predictions from the incorrect ones.

The dataset was presumed to be relatively balanced and easy to work with. Hence, accuracy was deemed to be good enough. Other metrics like precision and recall or their combined F-beta score are also valid.

Accuracy = number of correct predictions/total number of predictions

2 ANALYSIS

2.1 DATA EXPLORATION

2.1.1 The UrbanSound8K dataset

The UrbanSound8k dataset is one of the results of research in the Taxonomy of urban environments. This dataset came out of the paper "A Dataset and Taxonomy for Urban Sound Research" by J. Salamon, C. Jacoby and J.P. Bello at the 2nd ACM International Conference on Multimedia, Orlando USA in Nov. 2014.

This dataset contains 8732 sound clips of <=4 second duration. These sounds belong to 10 classes. Namely:

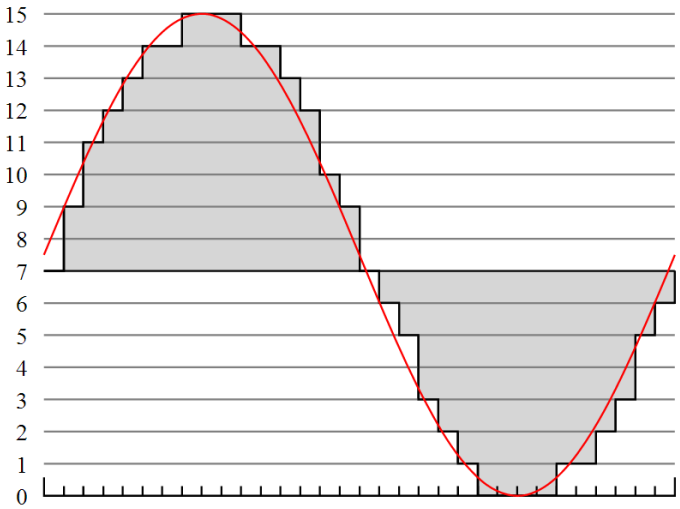
- Air Conditioner
- Car horn
- Children Player
- Dog barking
- Drilling
- Engine Idling
- Gunshot
- Jackhammer
- Siren
- Street music

The dataset also contains metadata with a unique ID for each of the classes along with other information not required for the project.

2.1.2 Audio File

Each audio file is a .wav file. Generally, sound is recorded and stored at **a sample rate** (the rate at which the amplitude of sound is recorded) of 44.1KHz with a bit-depth of 16(the number of bits used to represent each sample).

Here sample rate is the number of datapoints stored or read within a second. Sound waves are smooth and have infinite resolution. Since computers are digital in nature, they cannot store something with infinite resolution.



Here, an analog wave is overlaid on its digital representation. Individual points have been mapped to the wave giving a very jagged approximation. Every corner represents a point on the analog wave. The more points there are per unit time, the higher the accuracy of the digital wave is. This is, in simple terms, called **bit depth**. Higher bit depths can represent more points.

2.1.3 Audio Analysis Tools

Two audio manipulation tools tools will we employed in this project:

- According to the description on their page, **libROSA** is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.
- **IPython.display.Audio** is used to show audio playback controls in an IPython notebook.

2.1.4 Auditory Inspection

IPython.display.Audio(file) allows the insertion of music playback elements into a Jupyter notebook. One of the first thing noticed was that some sounds were very similar. For example, it requires domain knowledge to be able to accurately differentiate between a drill and a

Jackhammer. I you've never heard a jackhammer before you might think it's a high-powered drill or a combustion motor like a loud car engine.

Auditory Inspection

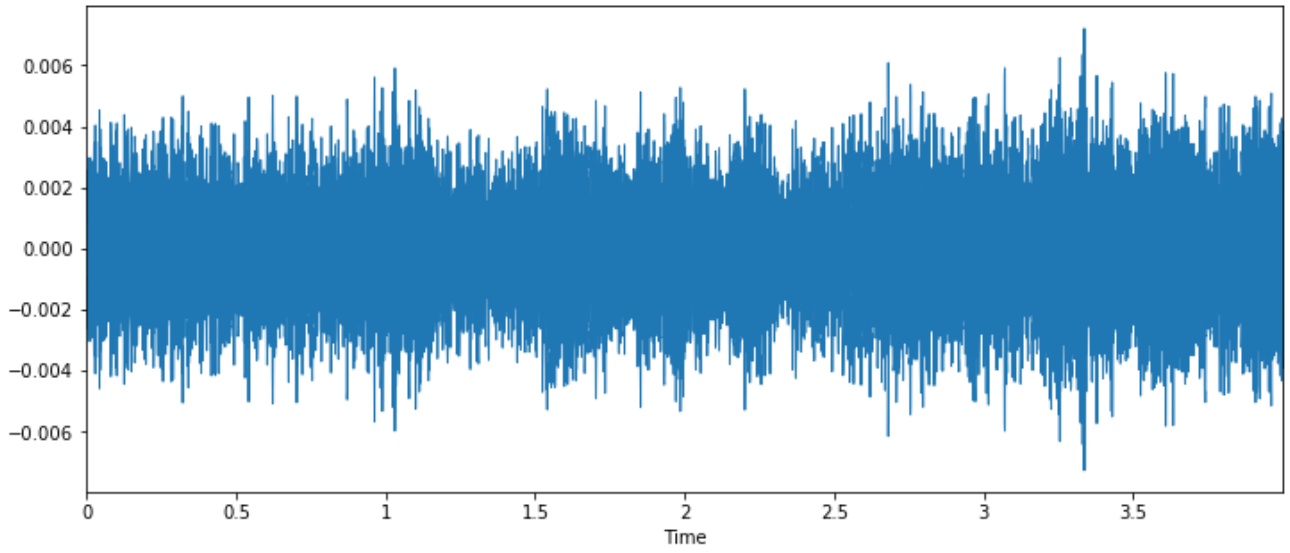
```
1  import IPython.display as ip
2
3  AUDIO_DIR = 'data/urbansound8k/audio'
4  ip.Audio(AUDIO_DIR+'/Fold6/4912-3-2-0.wav')
```

2.1.5 Visual Inspection

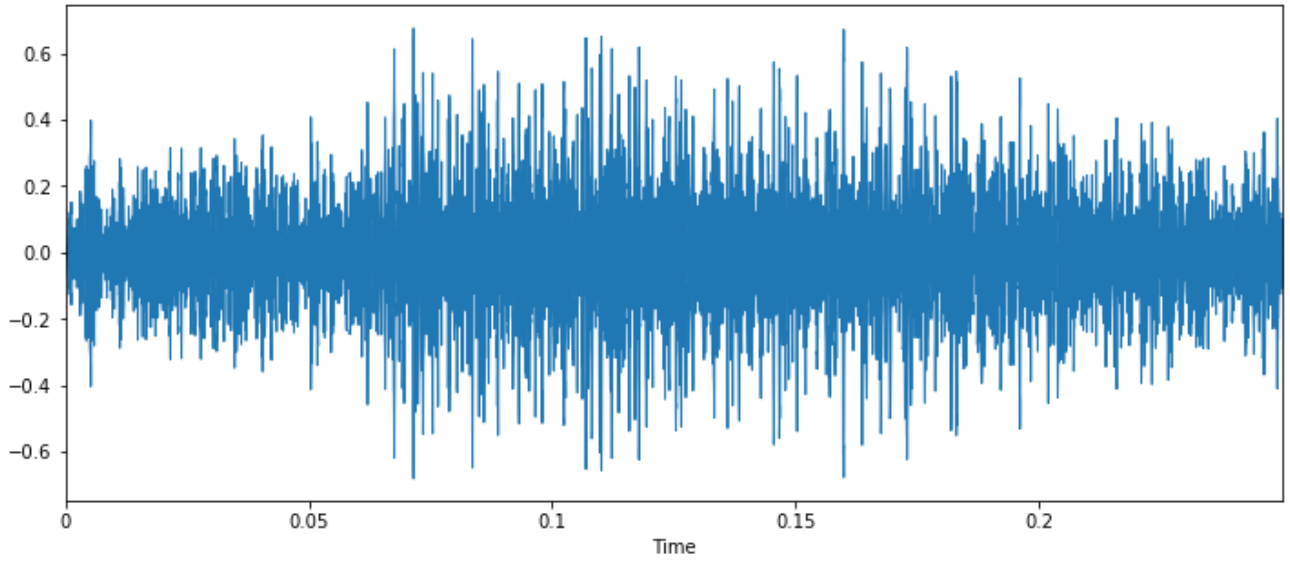
Let us visualize a sample of each class. This was made possible by libROSA:

```
def show_waveform(filepath):
    plt.figure(figsize=(12,5))
    data, sample_rate = librosa.load(filepath,sr=22050)
    _ = librosa.display.waveplot(data,sr=sample_rate)
```

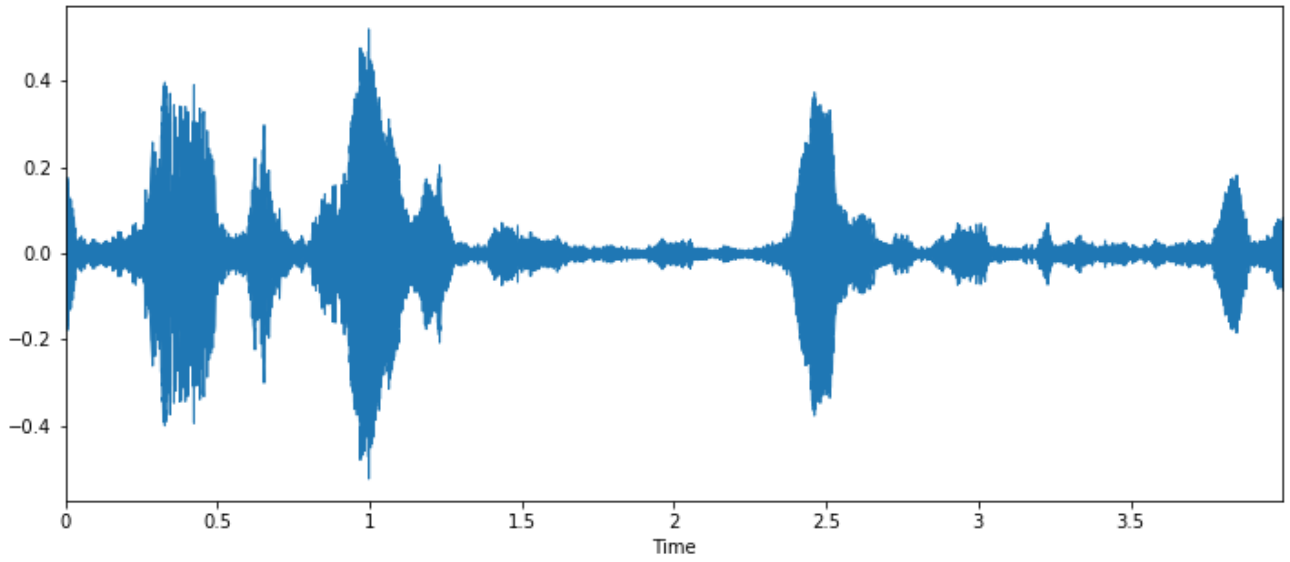
2.1.5.1 Air Conditioner



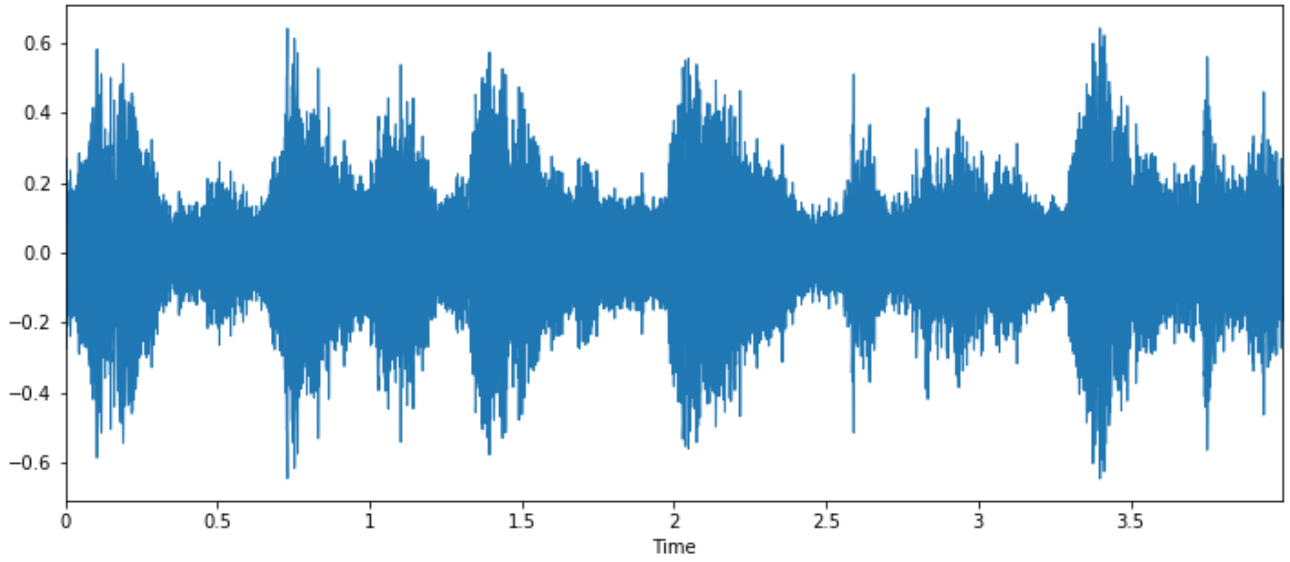
2.1.5.2 Car Horn



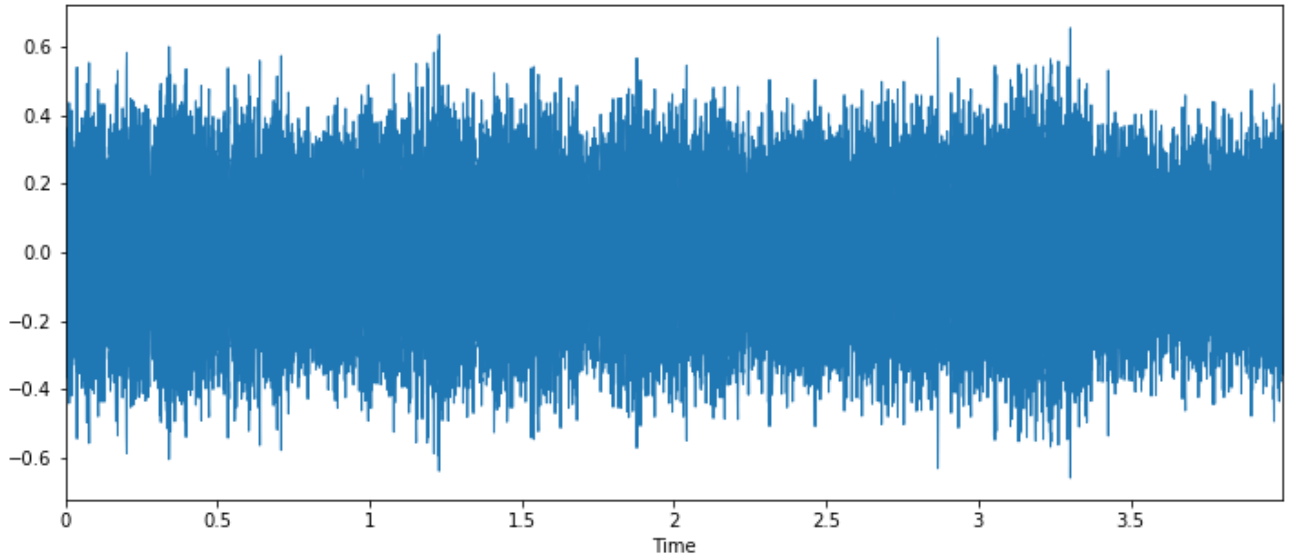
2.1.5.3 Children playing



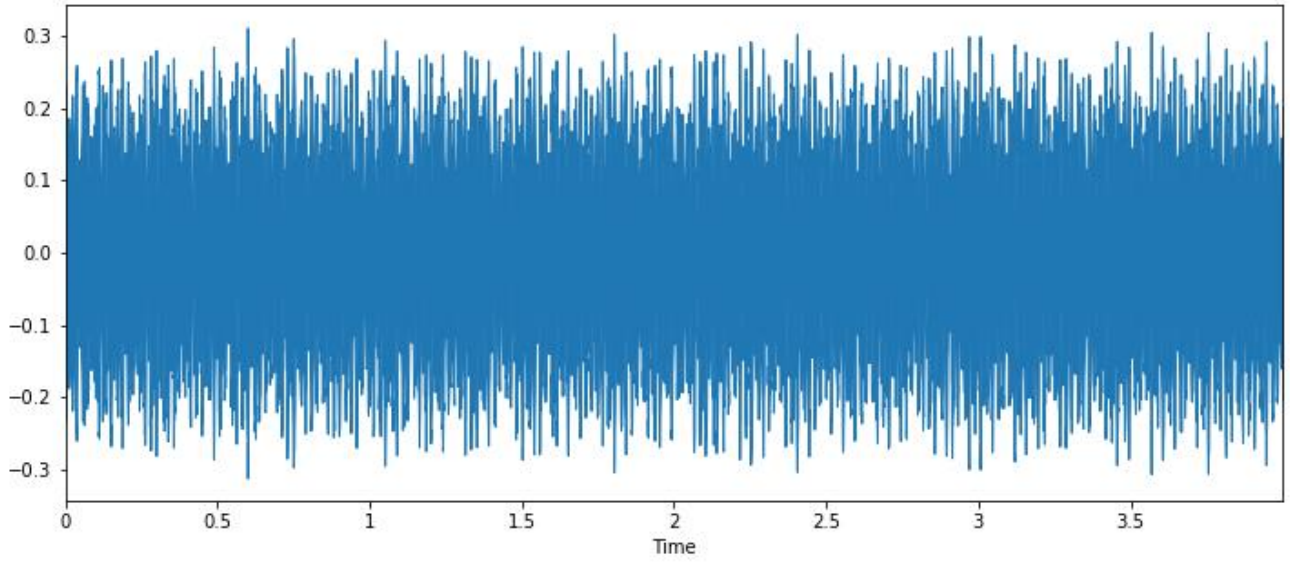
2.1.5.4 Dog barking



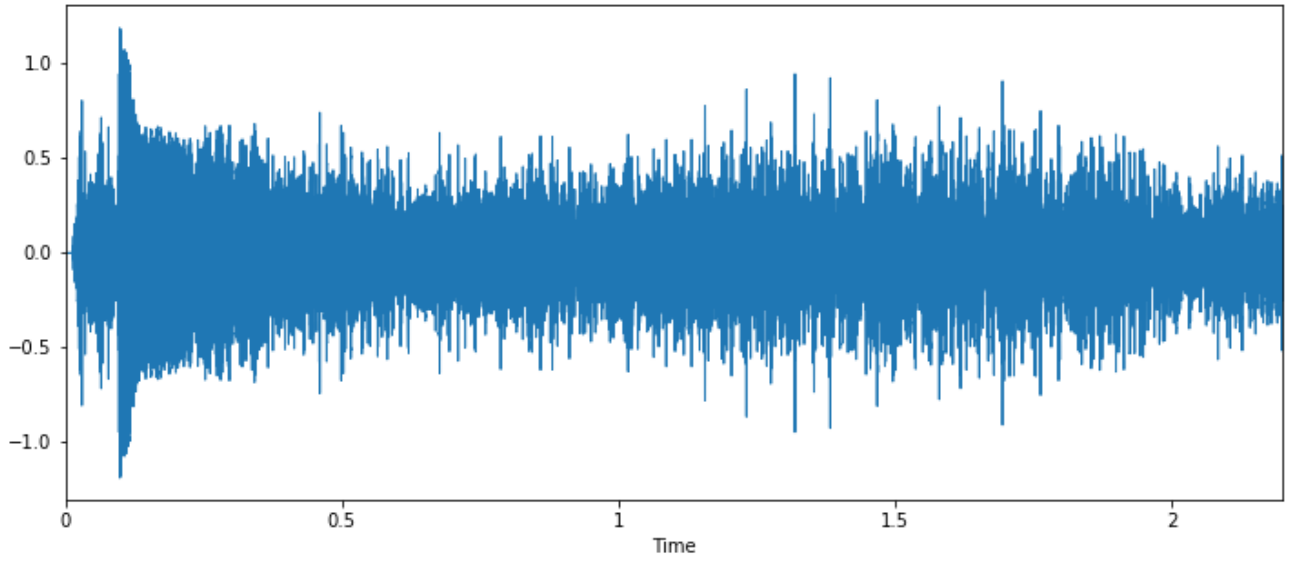
2.1.5.5 Drilling



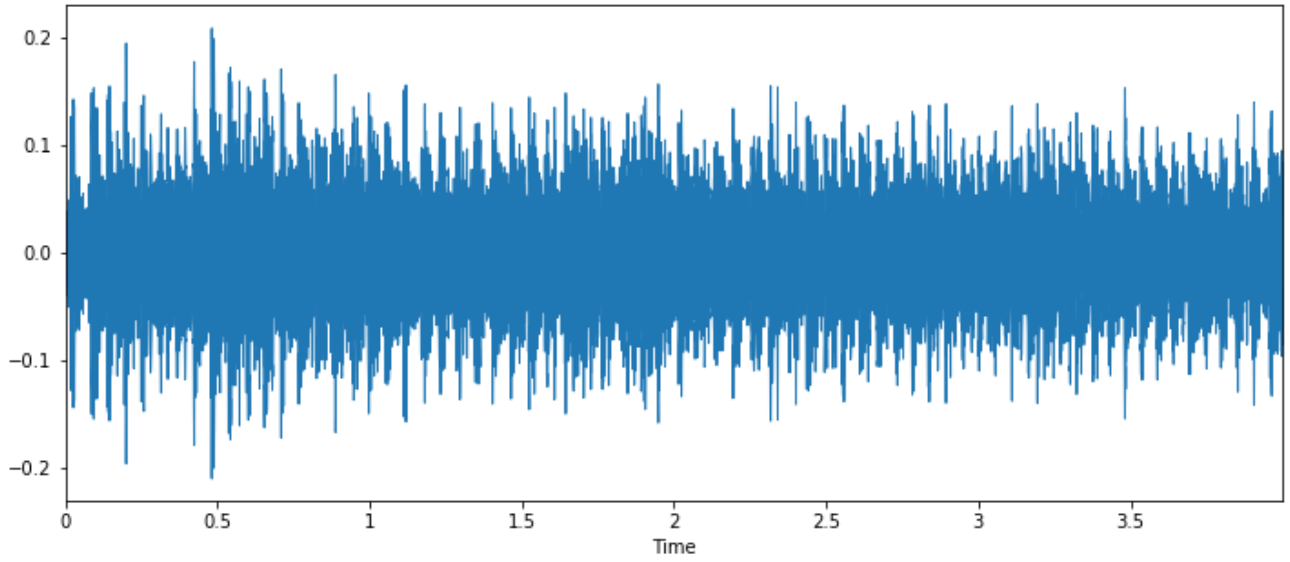
2.1.5.6 Idling engine



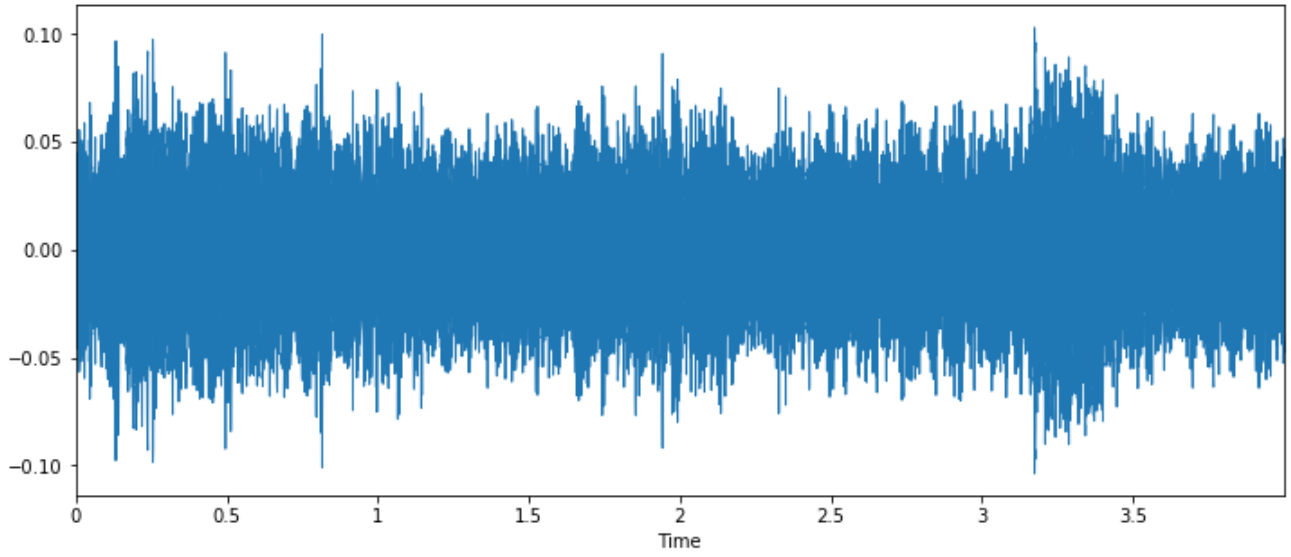
2.1.5.7 Gunshot



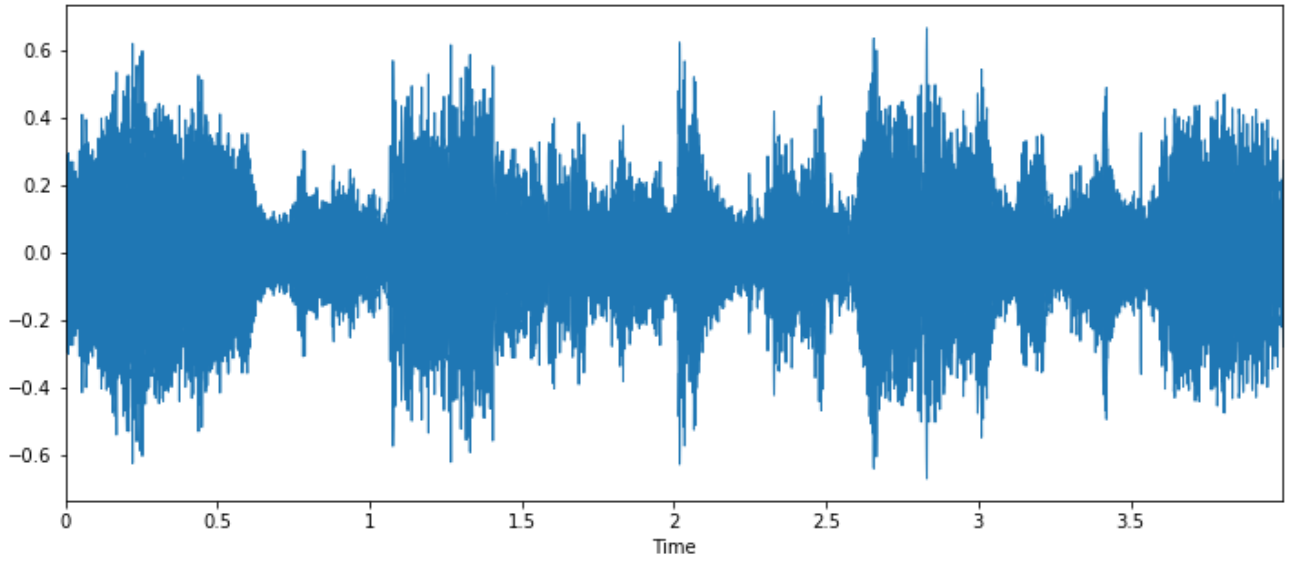
2.1.5.8 Jackhammer



2.1.5.9 Siren



2.1.5.10 Street Music



2.1.6 Observations

Visual inspection of waveforms is very uncertain in nature. The above representation does not carry information about the frequencies. Regardless, one thing that I did notice is that things like music, voices and dog barks had a very distinctive waveform containing beats.. This can be seen in the way that waveforms for those sounds have distinctive and relatively smooth rise and fall of amplitudes. Other things not designed to communicate have continuous and very noisy waveforms. That's something a CNN can detect with ease.

It can also be seen that the drill and engine have similar looking waveforms. The siren and the drill also have very similar looking waveforms. The human ear can still differentiate between the harmonics

2.1.7 Dataset Metadata

The metadata for each file is contained in the UrbanSound8K.csv file. It is imported as a Pandas Dataframe via the read_csv() function. It contains the slice_file_name which is unique to each file. FSID, start, end, salience and fold are not important for this project. ClassID is the numerical ID for the class mentioned in the class column.

```
1 META_PATH = 'data/urbansound8k/metadata/UrbanSound8K.csv'
2 metadata = pd.read_csv(META_PATH)
3 metadata.head()
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
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 Audio file properties

The most important properties for this project are **sample rate**, **bit depth** and **number of channels**.

```
import struct

def get_info(file):

        wave = open(file,"rb")
        riff = wave.read(12)
        fmt = wave.read(36)

        num_channels_string = fmt[10:12]
        num_channels = struct.unpack('<H', num_channels_string)[0]

        sample_rate_string = fmt[12:16]
        sample_rate = struct.unpack("<I",sample_rate_string)[0]

        bit_depth_string = fmt[22:24]
        bit_depth = struct.unpack("<H",bit_depth_string)[0]

        return (num_channels, sample_rate, bit_depth)

audiodata = []
for index, row in metadata.iterrows():
    filename = os.path.join(os.path.abspath('data/UrbanSound8K/audio/'),
                            'fold'+str(row["fold"])+'/',
                            str(row["slice_file_name"]))
    data = get_info(filename)
    audiodata.append(data)

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

2.1.8.1 Audio Channels

Most samples are stereo (2 channels) but some are mono (1 channel). The easiest way to handle this is to convert all samples to mono.

```
1  # Channel count
2  audio_prop['num_channels'].value_counts(normalize=True)
```

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

2.1.8.2 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 means they files will all have to be resampled at a common sampling rate to make the learning algorithm agnostic to sample rate.

```
# Sample rates
audio_prop['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
```

2.1.8.3 Bit Depth

There are a range of bit depths between samples. These must be normalized to make the learning algorithm agnostic to differences in bit depth.

```
audio_prop.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
```

2.2 Algorithms and Techniques

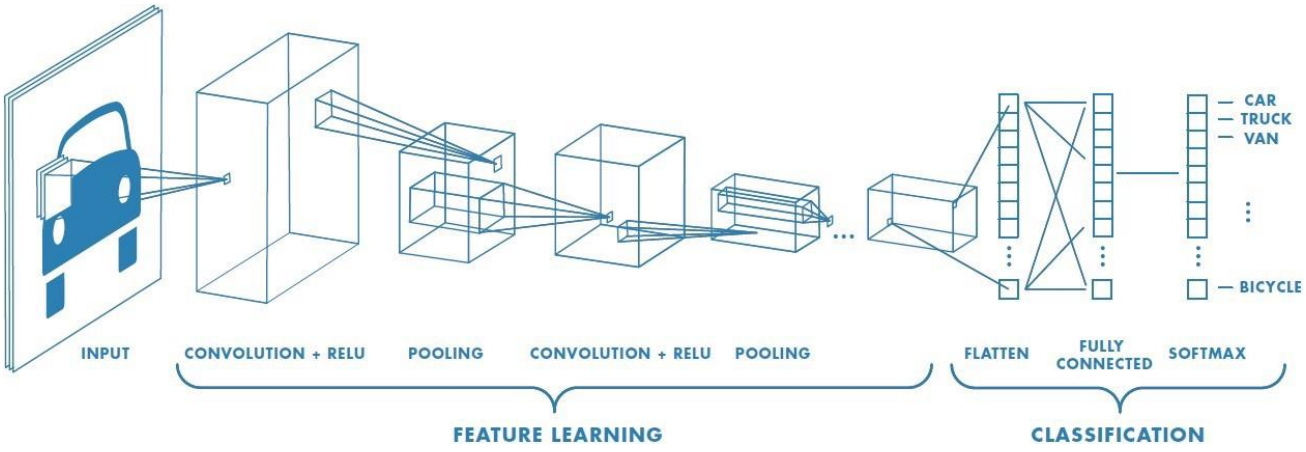
The solution, as mentioned in the proposal, is to use a deep neural network as they have proved to be very successful in noisy, real world applications.

The first step is to extract Mel-Frequency Cepstral Coefficients (MFCCs) from the audio samples on a per frame basis with a window size in the order of milliseconds. MFCCs collectively make up the Mel Frequency Cestrum (MFC) of a sound clip. The MFC of a sound is defined as a representation of the short-term power spectrum of sound based on linear cosine transform of a log power spectrum on a non-linear Mel scale of frequency. Here, the nonlinear scale describes human hearing.

The next step is to derive the MFCs of all the sounds and train a Convolutional Neural Network on them. MFCs are 2 dimensional which means that they can be converted to images, turning this task into an image classification problem.

Next, a Convolutional Neural Network (CNN) must be trained on the generated dataset. CNNs are one of if not the best architecture for image processing and classification. They act as feature extractors, producing a 1-dimensional representation of the CNN input. These representations or feature maps are input for a Multilayer perceptron that learns to classify them.

The CNN works on the principle of kernel convolutions. The kernel is a small, weighted sliding window that slides over the pixels of the image and performs matrix operations with the weights corresponding to each pixel's value. This allows it to generate increasingly complex feature maps that are then flattened before being fed to the MLP classifier.



As shown in the image above, the image is input to the first layer. This layer's size is equivalent to the input dimensions. Kernel convolutions are performed, and the results are input to the rectified linear unit activation function.

The **filter** parameter specifies the number of nodes in each layer, 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 activation parameter specifies the activation function. It is almost always 'relu' for image data.

The outputs of an activation function can be passed to the next convolution layer or a pooling layer. The pooling layer reduces the dimensionality of the model by reducing the number of parameters through max or average pooling. This also shortens the training time. The global average pooling layer performs average pooling on the entire input at once resulting in a 1-dimensional vector of features. This can be passed to a MLP for classification.

2.3 Benchmark Model

The model will be compared to algorithms outlined in the paper corresponding to the dataset, "A Dataset and Taxonomy for Urban Sound Research (Salamon, 2014)". This paper describes 4 different algorithms and their accuracies on the dataset.

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

3 METHODOLOGY

3.1 DATA PREPROCESSING

3.1.1 Normalizing audio

As mentioned in the previous section, the following audio properties must be normalized:

- Audio channels
- Sample rate
- Bit-depth

This will be done using the librosa library.

3.1.2 Preprocessing

Much of the processing will be done by default by the load() function. We shall compare librosa's load function to scipy since scipy does not perform any processing on the audio on load.

3.1.2.1 Sample rate conversion

Librosa converts all audio to 22.05KHz by default

```
#sample rate conversion
libro_audio, libro_sr = librosa.load(file)
scipy_sr,scipy_audio = wav.read(file)
print('Original sample rate:',scipy_sr)
print('Resampled rate:',libro_sr)
```

```
Original sample rate: 44100
Resampled rate: 22050
```

3.1.2.2 Bit Depth

Librosa normalizes data to values between -1 and 1. This elimintates the problem of audio having a range of bit-depths.

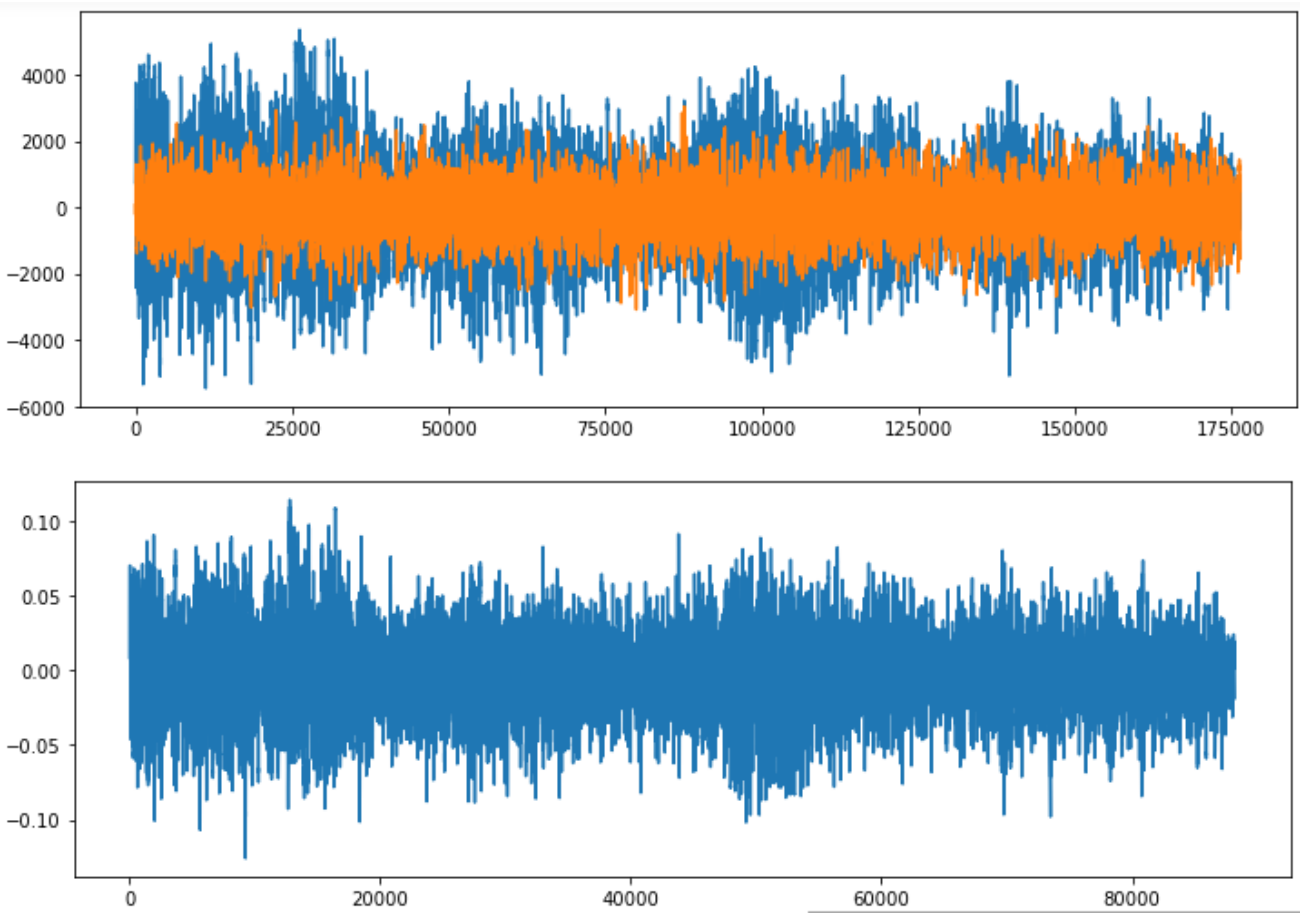
```
#bit depth converstion
print('Original file min/max range: ',np.min(scipy_audio),' to ',np.max(scipy_audio))
print('Resampled file min/max range: ',np.min(libro_audio),' to ',np.max(libro_audio))
```

```
Original file min/max range:  -5462 to 5355
Resampled file min/max range:  -0.1260349 to 0.114439055
```

3.1.2.3 Merging audio channels

Librosa will convert any n-channel audio to mono.

```
#Combining audio channels
plt.figure(figsize=(12, 4))
plt.plot(scipy_audio)
print('')
plt.figure(figsize=(12, 4))
plt.plot(libro_audio)
plt.show()
```



3.1.3 Feature extraction

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

MFCCs summarize frequency distribution in a time interval along the Mel scale. They can characterize both frequency and time characteristics of audio. These representations allow for the identification of features in audio.

3.1.3.1 Extracting MFCC

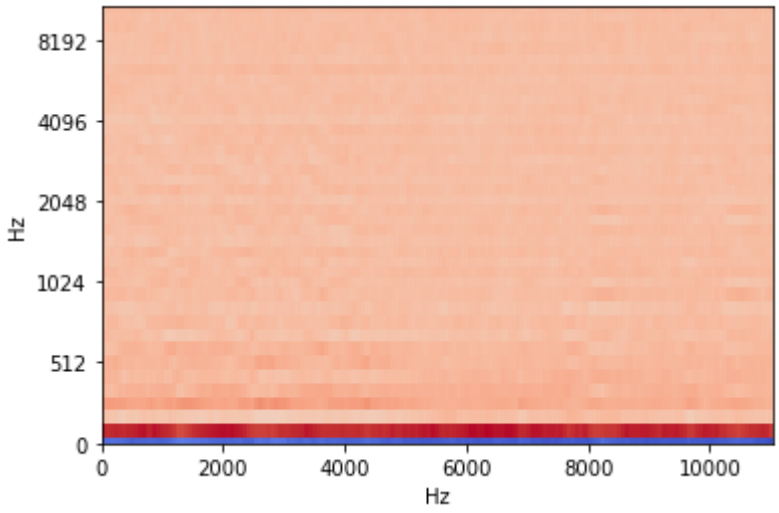
Librosa mfcc() function provides a convenient way to extract MFCCs over the intervals of an audio sample.

```
mfccs = librosa.feature.mfcc(y=libro_audio, sr=libro_sr, n_mfcc=40)
mfccs = librosa.amplitude_to_db(mfccs)
print(mfccs.shape)
```

```
(40,173)
```

This shows that 173 MFCCs were calculated with either corresponding to a 40 millisecond inte rval.

```
import librosa.display
librosa.display.specshow(mfccs, sr=libro_sr, x_axis='hz',y_axis='mel')
```



3.1.3.2 Extracting MFCCs for every file

The extract_mfcc() function will be used to extract MFCCs of all audio samples which will then be saved in a Pandas dataframe along with their class names. This function will be run on multiple threads to speed up processing.

```
import librosa
import sys
from random import randrange
import os

def extract_mfcc(index,row):
    file = os.path.join(os.path.abspath('data/urbansound8k/audio'),'fold'+str(row["fold"])+'/',str(row["slice_file_name"]))
    label =  row['class']
    max_padding = 175
    try:
        audio,sample_rate = librosa.load(file,res_type='kaiser_fast')
        #audio = add_noise(audio,noise_factor=0.2)

    except Exception as e:
        print('Error on line {}'.format(sys.exc_info()[-1].tb_lineno), type(e).__name__, e)
        return None

    mfccs = librosa.feature.mfcc(y=audio, n_mfcc=40)
    pad_width = max_padding - mfccs.shape[1]
    mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')

    return [mfccs,label]
```

```
%%time

import pandas as pd
import concurrent.futures
from tqdm.notebook import tqdm
from random import seed,randint

#n_samples = 20000

seed(2836)
metadata = pd.read_csv('data/urbansound8k/metadata/UrbanSound8k.csv')
#metadata_extended = pd.DataFrame([metadata.iloc[randint(0,len(metadata.index)-1)] for i in range(n_samples)],colum
data =[]

#multithreaded run of the function
with concurrent.futures.ThreadPoolExecutor() as exe:
    #lambda function was used to pass multiple args to the function
    data = list(tqdm(exe.map(lambda p:extract_mfcc(*p),metadata.iterrows()),total=len(metadata.index),leave=None))
```

```
Wall time: 3min 5s
```

```
#Slicing metadata and ordering the slice according to class IDs
classes_unique = metadata[['classID','class']].drop_duplicates('classID')
classes_unique.sort_values(by='classID',inplace=True)
classes = np.array(classes_unique['class'])

#Putting the data into pandas a dataframe
dataframe = pd.DataFrame(data,columns=['features', 'class'])
```

3.1.3.3 Conversion of categorical labels

Categorical text data in the labels is encoded into numerical format because most machine learning models only work on numerical data. The module used here is the sklearn.preprocessing.LabelEncoder. The labels are converted to categorical using keras.utils.to_categorical. This ensures that the model does not find any mathematical features in the label.

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

X = np.array(dataframe['features'].tolist())
y = np.array(dataframe['class'].tolist())

encoder = LabelEncoder()
cat_y = to_categorical(encoder.fit_transform(y))
```

3.1.3.4 Saving the dataset.

The dataset is saved to a .npz file for future use.

```
%%time

out_path = 'data/dataset.npz'
try:
    np.savez(out_path,features=X,labels=cat_y,classes=classes)
except Exception as e:
    print(e)
else:
    print('Done!')
```

```
Done!
Wall time: 584 ms
```

3.2 IMPLEMENTATION

This part was done on Google Colaboratory and, so, may look different. The dataset was also saved onto Google Drive for better data access speeds.

3.2.1 Data import and splitting.

The data is on Google Drive and needs to be retrieved into the Colaboratory notebook. It is then split using Sklearn's train_test_split() function.

```
# Retrieve contents of the data file.
import numpy as np
from tqdm.notebook import tqdm

BASE_PATH = '/content/drive/My Drive/mlnd_data'
file_path = BASE_PATH+'/dataset.npz'
contents = np.load(file_path, allow_pickle=True)

X = contents['features']
y = contents['labels']
classes = contents['classes']
```

```
%tensorflow_version 2.x

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.1,shuffle=True)
```

3.2.2 Convolutional Neural Networks

CNNs are some of the best tools for image classification due to their ability to quantify spatial relationships. Our architecture will be a relatively small Sequential model consisting of 3 Conv2D convolution layers each followed by a MaxPooling2D pooling layer. Finally, a GlobalAveragePooling2D layer converts the data in to a 1-dimensional vector which is passed to a Dense layer that outputs to 10 nodes. Each corresponding to one of the classes.

```
%tensorflow_version 2.x

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Conv2D, MaxPooling2D, GlobalAveragePooling2D, BatchNormalization

width = 40
height = 175
channels = 1
batch_size = 256

X_train = X_train.reshape(X_train.shape[0],width,height,channels)
X_test = X_test.reshape(X_test.shape[0],width,height,channels)

label_count = y.shape[1]

model = Sequential([
    Conv2D(filters=16,kernel_size=2,input_shape=(width,height,channels),activation='relu'),
    Dropout(0.2),
    MaxPooling2D(pool_size=2),

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

    Conv2D(filters=64,kernel_size=2,activation='relu'),
    Dropout(0.2),
    GlobalAveragePooling2D(),

    Dense(units=128,activation='tanh'),
    Dropout(0.1),
    Dense(units=label_count,activation='softmax')
])
```

3.2.2.1 Compiling the model

The model uses categorical crossentropy as it's loss and Adam as its optimizer. This is standard for multiclass classification.

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

The model is described as the following.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_67 (Conv2D)           (None, 39, 174, 16)       80
_________________________________________________________________
dropout_62 (Dropout)         (None, 39, 174, 16)       0
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 19, 87, 16)        0
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 18, 86, 32)        2080
_________________________________________________________________
dropout_63 (Dropout)         (None, 18, 86, 32)        0
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 9, 43, 32)         0
_________________________________________________________________
conv2d_69 (Conv2D)           (None, 8, 42, 64)         8256
_________________________________________________________________
dropout_64 (Dropout)         (None, 8, 42, 64)         0
_________________________________________________________________
global_average_pooling2d_21  (None, 64)                0
_________________________________________________________________
dense_43 (Dense)             (None, 128)               8320
_________________________________________________________________
dropout_65 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_44 (Dense)             (None, 10)                1290
=================================================================
Total params: 20,026
Trainable params: 20,026
Non-trainable params: 0
_________________________________________________________________
```

3.2.2.2 Model training.

The model is trained for 75 epochs with a batch size of 256 at an initial learning rate of 0.01. Some callbacks were also introduced. Reduce on plateau reduces the learning rate if the loss in the monitor parameter stops changing for a defined number of epochs which was 2 in this case. Another callback included is the ModelCheckPoint. This callback allows one to create checkpoints when a condition is met. In this case, the checkpointer saves the current epoch as a model if the validation loss reduces. I've also included a TQDMNotebook callback for nicer looking progress bars.

```
%%timeit

from keras_tqdm import TQDMNotebookCallback
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, History,TensorBoard
from tensorflow.keras.optimizers import Adam
from datetime import datetime
import os

now = datetime.now()

#learning rate reduction
MAX_PATIENCE = 2

best_filepath = BASE_PATH+'/best_model.hdf5'

#callbacks
callback = [ReduceLROnPlateau(patience = MAX_PATIENCE, verbose = 0),
            ModelCheckpoint(filepath=best_filepath,monitor='val_loss',verbose=0,save_best_only=True),
            TQDMNotebookCallback(leave_outer=None,leave_inner=None)]

#compile
model.compile(optimizer=Adam(learning_rate=0.01),loss='categorical_crossentropy',metrics=['accuracy'])

#train
model.fit(x=X_train,
          y=y_train,
          epochs=75,
          batch_size=batch_size,
          verbose=0,
          validation_split = 0.3,
          shuffle=True,
          callbacks=callback)
```

3.2.2.3 Model testing

As can be seen during model splitting, 10% of the dataset was set aside as a testing set.

```
# 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:  0.93026215
Testing Accuracy:  0.88787186
```

The model performs well, though the difference between the training and testing scores can indicate overfitting. This can be mitigated by augmenting the audio data before extraction of MFCCs by adding noise, distortion, pitch shifts and other random manipulation of sound.

The model still had a training accuracy of 93% and a testing accuracy of 89%.

3.2.2.4 Predicting with the model

We shall make predictions on data sourced from the outside world or the internet. The sound samples have to be converted to MFCs before being input to the model since the model expects 2-dimensional arrays.

```
import librosa
from sklearn.preprocessing import LabelEncoder

def get_mfccs(file_path,offset):
    max_pad_len=175

    try:
        audio, sample_rate = librosa.load(file_path, res_type='kaiser_fast',offset=offset,duration=4.0)
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
        pad_width = max_pad_len - mfccs.shape[1]
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')

    except Exception as e:
        print(e,file_path)
        return None

    return mfccs

def print_prediction(file_path,offset):
    prediction_feature = get_mfccs(file_path,offset)
    prediction_feature = prediction_feature.reshape(1, width, height, channels)

    prediction_vector = model.predict(prediction_feature)
    predicted_class = np.argmax(prediction_vector)
    print("Prediction: ",classes[predicted_class])
```

3.2.2.4.1 Air Conditioner

```
%%time
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'air-conditioner-on-a-camping-place.wav'
print_prediction(file_path,45)
```

```
Prediction:  engine_idling
CPU times: user 228 ms, sys: 252 ms, total: 481 ms
Wall time: 557 ms
```

3.2.2.4.2 Drilling

```
%%time
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'road-drilling-and-noise.wav'
print_prediction(file_path,0)
```

```
Prediction:  drilling
CPU times: user 105 ms, sys: 159 ms, total: 264 ms
Wall time: 421 ms
```

3.2.2.4.3 Street Music

```
%%time
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'street-music.wav'
print_prediction(file_path,2)
```

```
Prediction:  street_music
CPU times: user 118 ms, sys: 144 ms, total: 262 ms
Wall time: 708 ms
```

3.2.2.4.4 Gunshot

```
%%time
#InspectorJ (www.jshaw.co.uk) of Freesound.org
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'gunshot-distant-a.wav'
print_prediction(file_path,0)
```

```
Prediction:  dog_bark
CPU times: user 107 ms, sys: 150 ms, total: 256 ms
Wall time: 415 ms
```

3.2.2.4.5 Car Horn

```
%%time
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'car-very-long-mini-countryman.wav'
print_prediction(file_path,0)
```

```
Prediction:  drilling
CPU times: user 102 ms, sys: 156 ms, total: 258 ms
Wall time: 402 ms
```

3.2.2.4.6 Dog Barking

```
%%time
#InspectorJ (www.jshaw.co.uk) of Freesound.org
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'dog-barking-single-a.wav'
print_prediction(file_path,1)
```

```
Prediction:  dog_bark
CPU times: user 114 ms, sys: 135 ms, total: 249 ms
Wall time: 403 ms
```

3.2.2.4.7 Jackhammer

```
%%time
audio_dir = BASE_PATH+'/sample_sounds/'
file_path = audio_dir+'jackhammer.wav'
print_prediction(file_path,1)
```

```
Prediction:  jackhammer
CPU times: user 110 ms, sys: 177 ms, total: 287 ms
Wall time: 517 ms
```

3.3 Observations

The model performed well on real world examples recorded with different types of hardware. I was pleasantly surprised when it was able to distinguish between drilling and a jackhammer.

I am not surprised at the fact that the model could not differentiate between an idling vehicle and an air conditioner or a gunshot and barking dogs. Those two were hard to differentiate due to similar sounds. I am confused about the car horn being confused as drilling noise. I have not been able to rectify any of these errors. I could be argued, however, that these real-world noises may need more processing before being fed to the model or that the training data needs to be augmented.
I had experimented with adding random noise to the sounds before conversion to MFCCs and it resulted in a poorly performing model, so I didn't take that further.

I also observed that making a convolutional layer too big resulted in a poorly performing model with accuracy scores as low as 0.1.

4 Results

4.1 Model Evaluation and Validation

During model training, 30% of the training data was used as the validation set. The final model and hyperparameters were chosen because they seemed to perform the best and were fast to train.

By using external sounds sourced from the internet, the robustness of the model against real-world, unprocessed recordings was tested. The model seems to classify most samples correctly but some similar sounding samples may get misclassified. This indicates that the training data may need to be further processed. Increasing the size of the neural network resulted in poor performance.

4.2 Justification

The final model has an accuracy of 88% against new data which is better than the best benchmark model , SVM-rbf with 68% accuracy.

Algorithm	**Classification Accuracy**
CNN	88%
SVM_bf	68%
RandomForest500	66%
IBk5	55%
J48	48%
ZeroR	10%

We must keep in mind, though, that the model has not been tested on a real-time audio stream. My guess is that it will perform poorly because, even if we implement ways to slice the audio stream into equal fragments, the model does not know what other sounds are or what to do with unknown classes. A city has more classes of sounds than the 10 classes in the dataset. Another issue is inference speed. I expect it to be fast due to its small size, but I don't have the ability to test that.

There are other factors like echo, pitch shift or muffled audio that the model may not account for. Again, the model in the paper mentioned above may be able to tackle these issues while remaining fast.

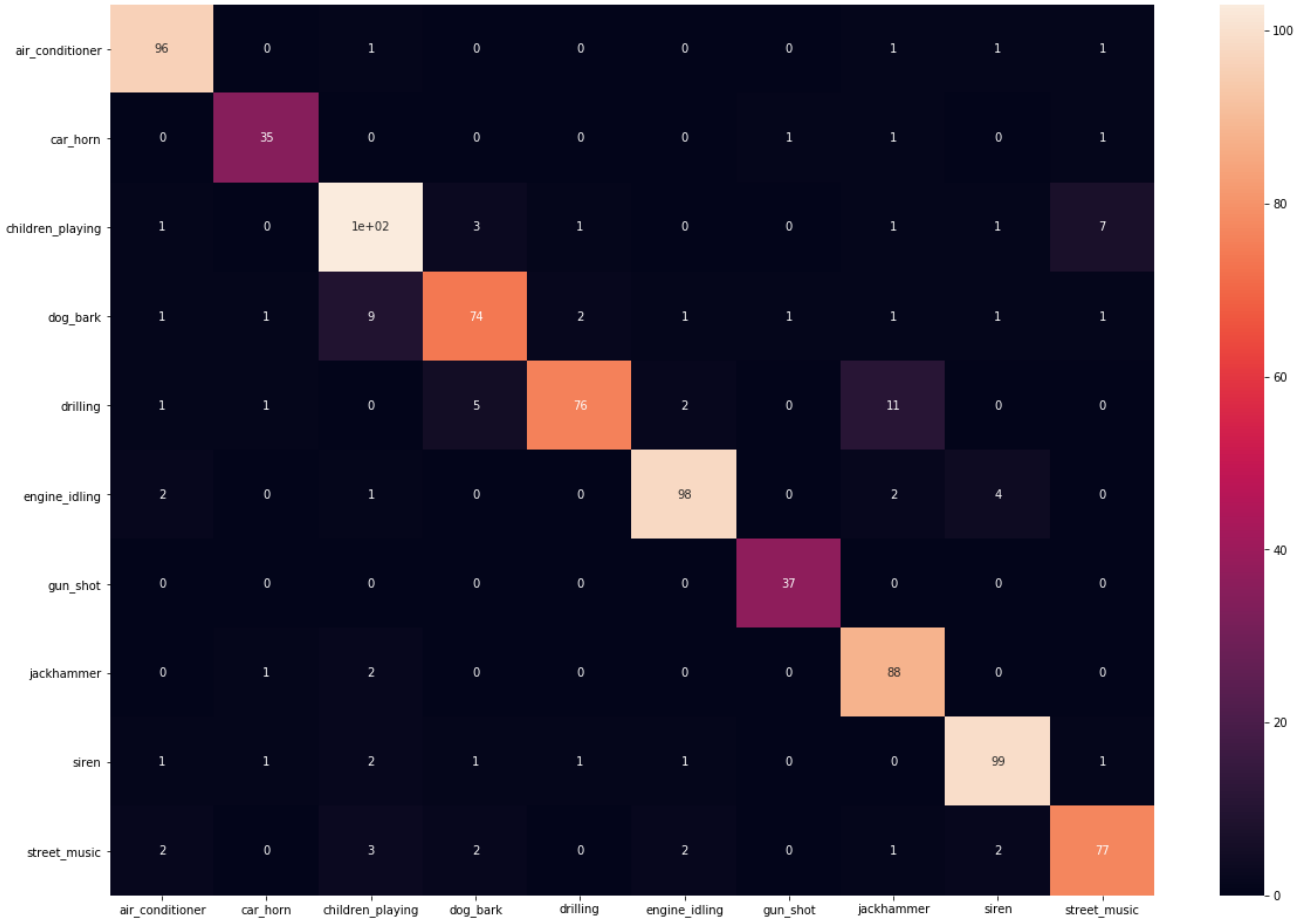
5 CONCLUSION

5.1 CONFUSION MATRIX

It was stated during data exploration that it is difficult to find differences between some classes. Especially those with similar waveforms.

- Sharp, distinctive peaks in sounds like dog barking or a gun firing.
- Street music can have patterns similar to children playing since it can include vocal chops or other characteristics depending on genre of music.
- Repetitive sounds like drilling, air-conditioning and idling cars.

A confusion matrix can be used to determine how similar the model thinks the sounds are.



The confusion matrix reveals other similarities not previously considered.

- Dogs barking and children playing.
- Drilling and dogs barking.

- Children playing and street music
- Engine idling and siren

This shows that there are other nuances that the model does not account for. To do so may require further data preprocessing and/or a more complex model.

5.2 Future improvements

- While working on the project I learned of the paper End-to-End Environmental Sound Classification using a 1D Convolutional Neural Network that uses a 1 dimensional CNN. It seems to achieve much better performance without the need for audio preprocessing.
- I have tried experimenting with adding noise to the audio samples before converting them to MFCCs but that resulted in a very poor model.
- One problem with this dataset is that it is too small. A data generator like the ImageDataGenerator class in Keras could be built to augment (once that's understood) and generate samples at runtime alleviating the problem of unbalanced classes and too small a dataset.

6 References

1. Justin Salamon, Christopher Jacoby and Juan Pablo Bello. Urban Sound Datasets
2. Sci-kit learn how to print labels for confusion matrix?
3. Wikipedia article on Mel-Frequency cepstrum
4. Urban Sound Classification using Convolutional Neural Networks with Keras: Theory and Implementation
5. A Dataset and Taxonomy for Urban Sound Research
6. Urban Sound Classification — Part 2: sample rate conversion, Librosa