Name: Chitipolu Sri Sudheera

Tool: CNN,keras

Urban Sound Classification

The dataset contains 8732 sound excerpts (<=4s) of urban sounds from 10 classes, namely:

- air conditioner,
- car horn,
- children playing,
- dog bark,
- drilling,
- engine idling,
- gun shot,
- jackhammer,
- siren, and
- street music

```
import IPython.display as ipd


ipd.Audio('../data/Train/2022.wav')
```

Now let us load this audio in our notebook as a numpy array. For this, we will use librosa library in python. To install librosa, just type this in command line

```
pip install librosa
```

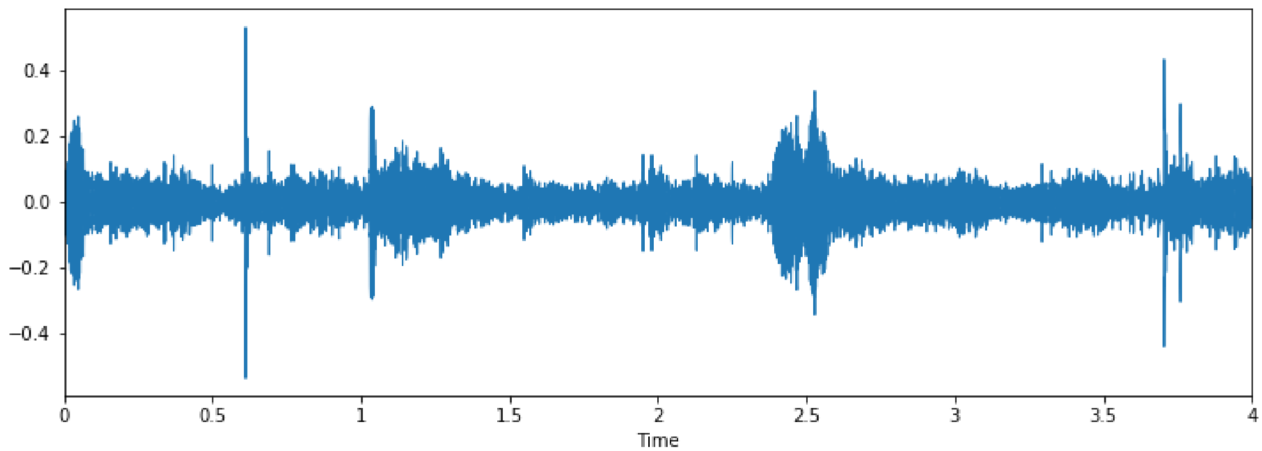
Now we can run the following code to load the data

```
data, sampling_rate = librosa.load('../data/Train/2022.wav')
```

When you load the data, it gives you two objects; a numpy array of an audio file and the corresponding sampling rate by which it was extracted. Now to represent this as a waveform (which it originally is), use the following code

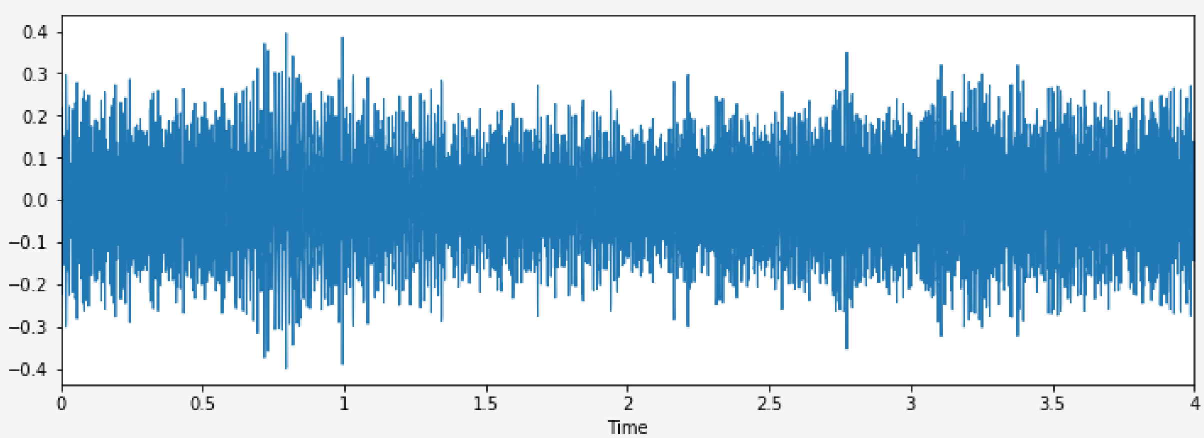
```
% pylab inline

import os

import pandas as pd

import librosa

import glob



plt.figure(figsize=(12, 4))

librosa.display.waveplot(data, sr=sampling_rate)
```

The output comes out as follows

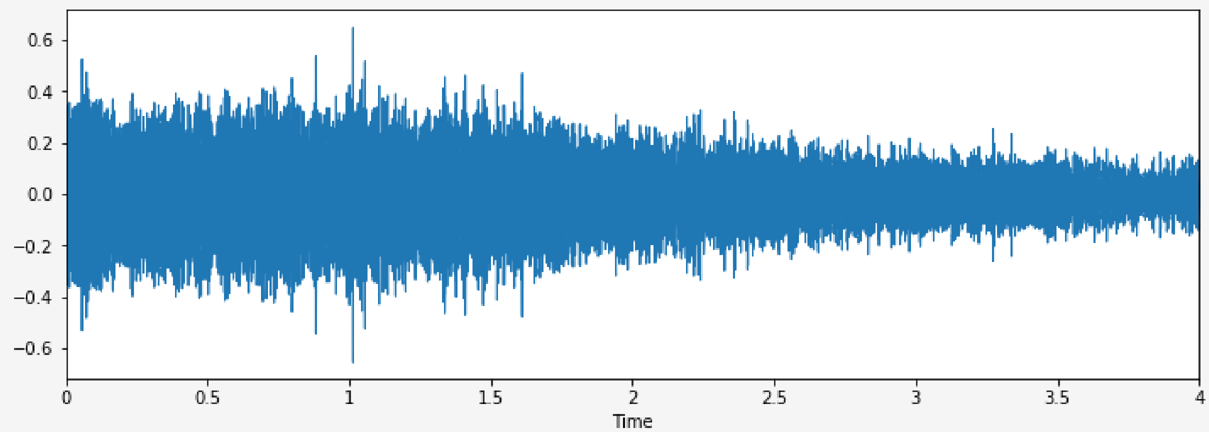


Let us now visually inspect our data and see if we can find patterns in the data

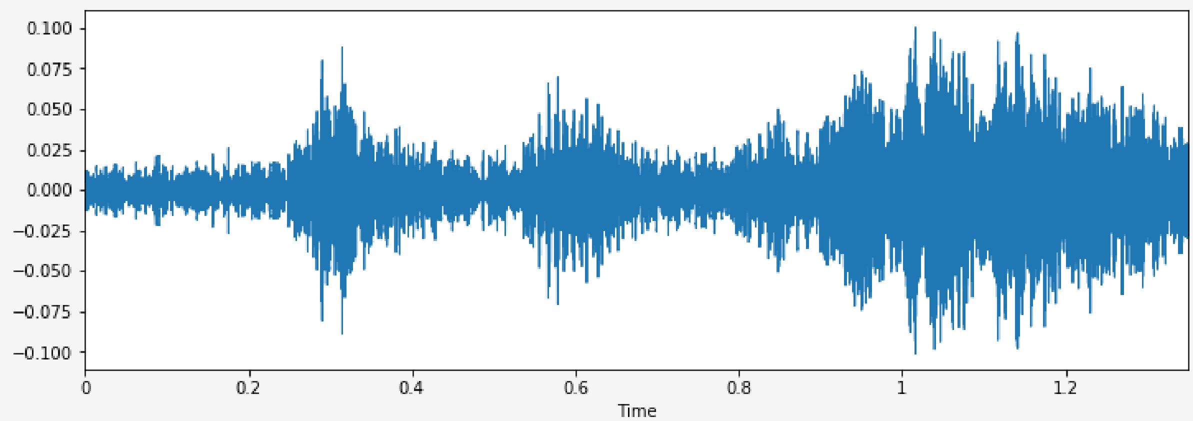
```
Class:  jackhammer
```




```
Class: drilling
```




```
Class: dog_barking
```



We can see that it may be difficult to differentiate between jackhammer and drilling, but it is still easy to discern between dog_barking and drilling. To see more such examples, you can use this code

```
i = random.choice(train.index)




audio_name = train.ID[i]

path = os.path.join(data_dir, 'Train', str(audio_name) + '.wav')
```

```
print('Class: ', train.Class[i])

x, sr = librosa.load('../data/Train/' + str(train.ID[i]) + '.wav')


plt.figure(figsize=(12, 4))

librosa.display.waveplot(x, sr=sr)
```

Intermission: Our first submission

to see the class distributions and just predict the max occurrence of all test cases as that class.

Let us see the distributions for this problem.

```
train.Class.value_counts()

Out[10]:


jackhammer 0.122907

engine_idling 0.114811
```

```
siren 0.111684

dog_bark 0.110396

air_conditioner 0.110396

children_playing 0.110396

street_music 0.110396

drilling 0.110396

car_horn 0.056302

gun_shot 0.042318
```

We see that jackhammer class has more values than any other class. So let us create our first submission with this idea.

```
test = pd.read_csv('../data/test.csv')

test['Class'] = 'jackhammer'

test.to_csv('sub01.csv', index=False)
```

This seems like a good idea as a benchmark for any challenge, but for this problem, it seems a bit unfair. This is so because the dataset is not much imbalanced.

Let’s solve the challenge! Part 2: Building better models

Now let us see how we can leverage the concepts we learned above to solve the problem. We will follow these steps to solve the problem.

Step 1: Load audio files
Step 2: Extract features from audio
Step 3: Convert the data to pass it in our deep learning model
Step 4: Run a deep learning model and get results

Below is a code of how I implemented these steps

Step 1 and 2 combined: Load audio files and extract features

```
def parser(row):

   # function to load files and extract features

   file_name = os.path.join(os.path.abspath(data_dir), 'Train', str(row.ID) + '.wav')



   # handle exception to check if there isn't a file which is corrupted

   try:

      # here kaiser_fast is a technique used for faster extraction

      X, sample_rate = librosa.load(file_name, res_type='kaiser_fast')

      # we extract mfcc feature from data

      mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
```

```
    except Exception as e:

        print("Error encountered while parsing file: ", file)

        return None, None



    feature = mfccs

    label = row.Class



    return [feature, label]



temp = train.apply(parser, axis=1)

temp.columns = ['feature', 'label']
```

Step 3: Convert the data to pass it in our deep learning model

```
from sklearn.preprocessing import LabelEncoder
```

```
X = np.array(temp.feature.tolist())

y = np.array(temp.label.tolist())



lb = LabelEncoder()



y = np_utils.to_categorical(lb.fit_transform(y))
```

Step 4: Run a deep learning model and get results

```
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 = y.shape[1]

filter_size = 2



# build 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'))



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

Now let us train our model

```
model.fit(X, y, batch_size=32, epochs=5, validation_data=(val_x, val_y))
```

This is the result I got on training for 5 epochs

```
Train on 5435 samples, validate on 1359 samples

Epoch 1/10

5435/5435 [==============================] - 2s - loss: 12.0145 - acc: 0.1799 - val_l
oss: 8.3553 - val_acc: 0.2958

Epoch 2/10

5435/5435 [==============================] - 0s - loss: 7.6847 - acc: 0.2925 - val_lo
ss: 2.1265 - val_acc: 0.5026

Epoch 3/10

5435/5435 [==============================] - 0s - loss: 2.5338 - acc: 0.3553 - val_lo
ss: 1.7296 - val_acc: 0.5033
```

```
Epoch 4/10

5435/5435 [==============================] - 0s - loss: 1.8101 - acc: 0.4039 - val_lo
ss: 1.4127 - val_acc: 0.6144

Epoch 5/10

5435/5435 [==============================] - 0s - loss: 1.5522 - acc: 0.4822 - val_lo
ss: 1.2489 - val_acc: 0.6637
```

Seems ok