AUDIO CLASSIFICATION USING DEEP LEARNING

INTRODUCTION

One of the most widely used applications in Deep Learning is Audio classification, in which the model learns to classify sounds based on audio features. When given the input, it will predict the label for that audio feature. These can be used in different industrial applications like classifying short utterances of the speakers, music genre classification, etc. Here, we will walk through a simple demonstration of audio classification on the UrbanSound8K dataset. We will write certain audio preprocessing functions, we will extract Mel Spectrogram from audio, then pass the features to a basic convolutional neural network and start the training, specifically focusing on the classification of particular urban sounds.
Our goal is to give an audio file as the input, then our model should determine whether the audio features contain one of the target labels.

DATASET

UrbanSound8K dataset contains 8732 labeled urban sounds from 10 classes, that is air_conditioner, car_horn, children_playing, dog_bark, drilling, enginge_idling, gun_shot, jackhammer, siren, and street_music. It also contained a CSV file UrbanSound8k.csv which contains meta-data information about every audio file in the dataset.

Download dataset from :-
https://urbansounddataset.weebly.com/download-urbansound8k.html

PRACTICAL IMPLEMENTATAION

Libraries Installation

The very important and great library that supports audio and music analysis is **Librosa**. Simply use the Pip command to install the library. It provides building blocks that are required to construct an information retrieval model from music. Another great library we will use is for deep learning modeling purposes is TensorFlow.

```
[1]: !pip install Librosa
     !pip install tensorflow
```

EXPLORATORY DATA ANALYSIS OF AUDIO

We have 10 different folders under the urban dataset folder. Before applying any preprocessing, we will see how to load audio files and how to visualize them in form of the waveform.And also to load the audio file and listen to it,for thiswe will use the IPython library and directly give it an audio file path. We have taken the first audio file of dog_bark.

[2]:
```
import IPython.display as ipd
filename=('C:/UrbanSound8K/UrbanSound8K/100032-3-0-0.wav')
ipd.Audio(filename)
```

[2]:

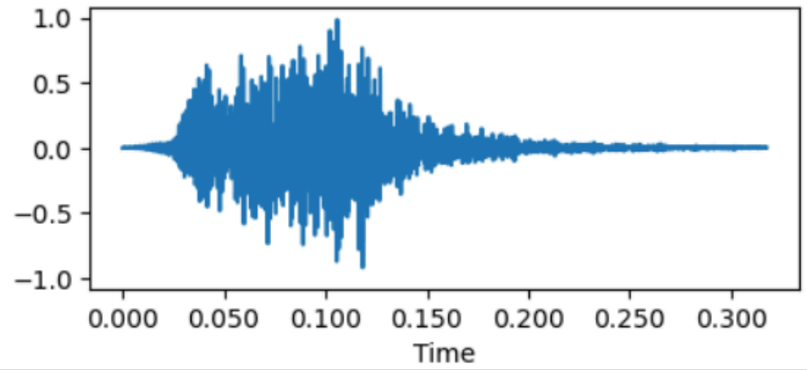
EXPLORATORY DATA ANALYSIS OF AUDIO

Now we will use Librosa to load audio data. So when we load any audio file with Librosa, it gives us 2 things. One is **sample rate**, and the other is a **two-dimensional array.** Let us load the above audio file with Librosa and plot the waveform using Librosa.

- **Sample rate** – It represents how many samples are recorded per second. The default sampling rate with which librosa reads the file is 22050. The sample rate differs by the library you choose.
- **2-D Array** – The first axis represents recorded samples of amplitude. And the second axis represents the number of channels. There are different types of channels – Monophonic(audio that has one channel) and stereo(audio that has two channels).

```
import matplotlib.pyplot as plt
%matplotlib inline
import librosa
import librosa.display
data,sample_rate=librosa.core.load('C:/UrbanSound8K/UrbanSound8K/100032-3-0-0.wav')
plt.figure(figsize=(5,2))
librosa.display.waveshow(data,sr=sample_rate)
```

```
<librosa.display.AdaptiveWaveplot at 0x160291ca990>
```



EXPLORATORY DATA ANALYSIS OF AUDIO

IMPORTING METADATA AND CHECKING DATA IS IMBALANCED OR NOT

```
#importing metadata csv file
import pandas as pd
metadata=pd.read_csv('C:/UrbanSound8K/UrbanSound8K/metadata/UrbanSound8K.csv')
metadata.head(10)
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
1	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
2	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
3	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
4	100263-2-0-143.wav	100263	71.500000	75.500000	1	5	2	children_playing

```
### Check whether the dataset is imbalanced
metadata['class'].value_counts()
```

```
class
children_playing    1000
air_conditioner     1000
street_music        1000
engine_idling       1000
jackhammer          1000
drilling            1000
dog_bark             999
siren                929
car_horn             429
gun_shot             374
Name: count, dtype: int64
```

DATA PREPROCESSING

SOME AUDIOS ARE GETTING RECORDED AT A DIFFERENT RATE-LIKE 44KHZ OR 22KHZ. USING LIBROSA, IT WILL BE AT 22KHZ, AND THEN, WE CAN SEE THE DATA IN A NORMALIZED PATTERN. NOW, OUR TASK IS TO EXTRACT SOME IMPORTANT INFORMATION, AND KEEP OUR DATA IN THE FORM OF INDEPENDENT(EXTRACTED FEATURES FROM THE AUDIO SIGNAL) AND DEPENDENT FEATURES(CLASS LABELS). WE WILL USE MEL FREQUENCY CEPSTRAL COEFFICIENTS TO EXTRACT INDEPENDENT FEATURES FROM AUDIO SIGNALS

MFCCS – THE MFCC SUMMARIZES THE FREQUENCY DISTRIBUTION ACROSS THE WINDOW SIZE. SO, IT IS POSSIBLE TO ANALYZE BOTH THE FREQUENCY AND TIME CHARACTERISTICS OF THE SOUND. THIS AUDIO REPRESENTATION WILL ALLOW US TO IDENTIFY FEATURES FOR CLASSIFICATION. SO, IT WILL TRY TO CONVERT AUDIO INTO SOME KIND OF FEATURES BASED ON TIME AND FREQUENCY CHARACTERISTICS THAT WILL HELP US TO DO CLASSIFICATION.

```
[6]: mfccs = librosa.feature.mfcc(y=data, sr=sample_rate, n_mfcc=40)
     print(mfccs.shape)

     (40, 14)

[7]: mfccs
```

DATA PREPROCESSING

NOW, WE HAVE TO EXTRACT FEATURES FROM ALL THE AUDIO FILES AND PREPARE THE DATAFRAME. SO, WE WILL CREATE A FUNCTION THAT TAKES THE FILENAME(FILE PATH WHERE IT IS PRESENT). IT LOADS THE FILE USING LIBROSA, WHERE WE GET 2 INFORMATION. FIRST, WE'LL FIND MFCC FOR THE AUDIO DATA, AND TO FIND OUT SCALED FEATURES, WE'LL FIND THE MEAN OF THE TRANSPOSE OF AN ARRAY.

[8]:
```
#### Extracting MFCC's For every audio file
import pandas as pd
import os
import librosa

audio_dataset_path='C:/UrbanSound8K/UrbanSound8K/audio'
metadata=pd.read_csv('C:/UrbanSound8K/UrbanSound8K/metadata/UrbanSound8K.csv')
metadata.head()
```

[9]:
```
def features_extractor(file):
    data, sample_rate = librosa.load(file_name)
    mfccs_features = librosa.feature.mfcc(y=data, sr=sample_rate, n_mfcc=40)
    mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

    return mfccs_scaled_features
```

DATA PREPROCESSING

NOW, TO EXTRACT ALL THE FEATURES FOR EACH AUDIO FILE, WE HAVE TO USE A LOOP OVER EACH ROW IN THE DATAFRAME. WE ALSO USE THE TQDM PYTHON LIBRARY TO TRACK THE PROGRESS. INSIDE THE LOOP, WE'LL PREPARE A CUSTOMIZED FILE PATH FOR EACH FILE AND CALL THE FUNCTION TO EXTRACT MFCC FEATURES AND APPEND FEATURES AND CORRESPONDING LABELS IN A NEWLY FORMED DATAFRAME.

[10]:
```
import numpy as np
from tqdm import tqdm
### Now we iterate through every audio file and extract features
### using Mel-Frequency Cepstral Coefficients
extracted_features=[]
for index_num,row in tqdm(metadata.iterrows()):
    file_name = os.path.join(os.path.abspath(audio_dataset_path),'fold'+str(row["fold"])+'/',str(row["slice_file_name"]))
    final_class_labels=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data,final_class_labels])
```

DATA PREPROCESSING

NOW, TO EXTRACT ALL THE FEATURES FOR EACH AUDIO FILE, WE HAVE TO USE A LOOP OVER EACH ROW IN THE DATAFRAME. WE ALSO USE THE TQDM PYTHON LIBRARY TO TRACK THE PROGRESS. INSIDE THE LOOP, WE'LL PREPARE A CUSTOMIZED FILE PATH FOR EACH FILE AND CALL THE FUNCTION TO EXTRACT MFCC FEATURES AND APPEND FEATURES AND CORRESPONDING LABELS IN A NEWLY FORMED DATAFRAME.

[10]:
```
import numpy as np
from tqdm import tqdm
### Now we iterate through every audio file and extract features
### using Mel-Frequency Cepstral Coefficients
extracted_features=[]
for index_num,row in tqdm(metadata.iterrows()):
    file_name = os.path.join(os.path.abspath(audio_dataset_path),'fold'+str(row["fold"])+'/',str(row["slice_file_name"]))
    final_class_labels=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data,final_class_labels])
```

[11]:
```
#Converting extracted features to Pandas dataframe
extracted_features_df=pd.DataFrame(extracted_features,columns=(['feature','Class']))
extracted_features_df.head(-5)
```

TRAIN TEST SPLIT

- FIRST, WE SPLIT THE DEPENDENT AND INDEPENDENT FEATURES.
- AFTER THAT, WE HAVE 10 CLASSES, SO WE USE LABEL ENCODING(INTEGER LABEL ENCODING) FROM NUMBER 1 TO 10 AND CONVERT IT INTO CATEGORIES.
- AFTER THAT, WE SPLIT THE DATA INTO TRAIN AND TEST SETS IN AN 80-20 RATIO.

[12]:
```
#Split dataset into dependent and independent features
X=np.array(extracted_features_df['feature'].tolist())
y=np.array(extracted_features_df['Class'].tolist())
```

[13]:
```
##Label Encoding
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical(labelencoder.fit_transform(y))
```

[14]:
```
#Train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.22,random_state=0)
```

AUDIO CLASSIFICATION MODEL CREATION

WE HAVE EXTRACTED FEATURES FROM THE AUDIO SAMPLE AND SPLITTER IN THE TRAIN AND TEST SET. NOW WE WILL IMPLEMENT AN ANN MODEL USING KERAS SEQUENTIAL API.
THE NUMBER OF CLASSES IS 10, WHICH IS OUR OUTPUT SHAPE(NUMBER OF CLASSES), AND WE WILL CREATE ANN WITH 1 DENSE LAYERS (BECAUSE WE HAVE TRIED MORE THAN ONE DENSE LAYER BUT MODEL WITH SINGLE LAYER GIVES GOOD ACCURACY COMPARATIVELY WITH OTHERS) AND ARCHITECTURE IS EXPLAINED BELOW.

THE FIRST LAYER HAS 200 NEURONS. INPUT SHAPE IS 40 ACCORDING TO THE NUMBER OF FEATURES WITH ACTIVATION FUNCTION AS RELU, AND TO AVOID ANY OVERFITTING, WE'LL USE THE DROPOUT LAYER AT A RATE OF 0.5.

THE SECOND LAYER HAS 300 NEURONS WITH ACTIVATION FUNCTION AS RELU AND THE DROP OUT AT A RATE OF 0.5.

THE THIRD LAYER AGAIN HAS 100 NEURONS WITH ACTIVATION AS RELU AND THE DROP OUT AT A RATE OF 0.5.

THE FINAL LAYER HAS DENSE OF NUM_LABELS & SOFTMAX AS A ACTIVATION FUNCTION BECAUSE IT IS MULTI-CLASS CLASSIFICATION PROBLEM.

THEN CAN SEE MODEL SUMMARY USING **MODEL.SUMMARY** IN PYTHON

AUDIO CLASSIFICATION MODEL CREATION

[15]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten
from tensorflow.keras.optimizers import Adam
from sklearn import metrics
```

[16]:
```
##No. of classes
num_labels=y.shape[1]
```

[17]:
```
###layers of ANN
model=Sequential()
###first layer
model.add(Dense(200,input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

###second layer
model.add(Dense(300))
model.add(Activation('relu'))
model.add(Dropout(0.5))

###Final layer
model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

COMPILE & TRAIN THE MODEL

To compile the model we need to define loss function which is categorical cross-entropy, accuracy metrics which is accuracy score, and an optimizer which is Adam.

[19]:
```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'],optimizer='adam')
```

We will **train the model** and save the model in HDF5 format. We will train a model for 100 epochs and batch size as 32. We'll use callback, which is a checkpoint to know how much time it took to train over data

[20]:
```
## Trianing my model
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 200
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification.keras',
                               verbose=1, save_best_only=True)
start = datetime.now()

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


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

CHECK THE TEST ACCURACY

Now we will evaluate the model on test data. we got near about 88 percent accuracy on the training dataset and 89 percent on test data.

```
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
test_accuracy[1]
```

```
0.894864393615723
```

In TensorFlow version below 2.6, then we can use predict classes function to predict the corresponding class for each audio file. But, if using 2.6 and above, then can use predict and argument maximum function.

```
##Model predicting class
predict_x=model.predict(X_test)
classes_x=np.argmax(predict_x,axis=1)
print(classes_x)
```

TESTING SOME TEST AUDIO SAMPLE

Now we test some random audio samples. Whenever we'll get new audio, we will perform three steps again to get the predicted label and class.

1. First, preprocess the audio file (load it using Librosa and extract MFCC features)
2. Predict the label to which audio belongs.
3. An inverse transforms the predicted label to get the respective class name to which it belongs.

```
filename="C:/UrbanSound8K/UrbanSound8K/14387-9-0-15.wav"
##Preprocessing audiofile
audio, sample_rate = librosa.core.load(filename)
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)
print(mfccs_scaled_features)

##reshape MFCC feature to 2-D array
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)

##predicted_label = model.predict(mfccs_scaled_features)

x_predict=model.predict(mfccs_scaled_features)
x_label=np.argmax(x_predict,axis=1)
print(x_label)
prediction_class=labelencoder.inverse_transform(x_label)
prediction_class
```

OUTPUT

Which is correct

```
[-317.80743     158.84431      -7.8308206    39.62621      -3.0769029
   -0.39469874    4.628633     11.348094      2.6572006    -2.3324416
   17.067583      7.080102      6.4992504     1.98673       1.3234887
    8.727966      5.2164516     5.9689026    -5.084938     -2.6112342
   -1.8349239     3.1345532     2.7855914     4.000883      1.1125592
   -3.1095183    -3.8233416     5.63856      -0.5054775    -2.9793928
    1.5528826     9.117689      9.156756      1.3443911     2.0258229
    0.3984364    -9.457513     -4.8549147     4.111599      3.1215594 ]
[[-317.80743     158.84431      -7.8308206    39.62621      -3.0769029
    -0.39469874    4.628633     11.348094      2.6572006    -2.3324416
    17.067583      7.080102      6.4992504     1.98673       1.3234887
     8.727966      5.2164516     5.9689026    -5.084938     -2.6112342
    -1.8349239     3.1345532     2.7855914     4.000883      1.1125592
    -3.1095183    -3.8233416     5.63856      -0.5054775    -2.9793928
     1.5528826     9.117689      9.156756      1.3443911     2.0258229
     0.3984364    -9.457513     -4.8549147     4.111599      3.1215594 ]]
(1, 40)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
[9]
array(['street_music'], dtype='<U16')
```

CONCLUSION

- Audio classification is a technique to classify sounds into different categories.
- We can visualize any audio in the form of a waveform.
- MFCC method is used to extract important features from audio files.
- Scaling the audio samples to a common scale is important before feeding data to the model to understand it better.

THANK YOU

TRY THIS BACKGROUND FOR ONLINE CLASS.

*Please delete this section before downloading.

PRESS THESE KEYS WHILE ON PRESENT MODE!

B FOR BLUR

C FOR CONFETTI

D FOR A DRUMROLL

M FOR MIC DROP

O FOR BUBBLES

Q FOR QUIET

U FOR UNVEIL

ANY NUMBER FROM 0-9 FOR A TIMER