CAPSTONE PROJECT: URBAN SOUND CLASSIFICATION

Objective:

To train a machine learning model to classify the category of 10 urban sounds correctly from the sound file.

Introduction:

Classifying sounds is a growing area of research. Speech processing applications like Amazon Alexa, Google Home, Siri etc. are vital in assisting us in daily life activities. Other applications of sound classification include surveillance via sounds such as gunshot detection can help law-enforcing authorities to dispatch help as soon as such activity is detected.

In this project, a machine learning model is designed and trained to classify ten different categories of sounds that are common in an urban environment.

Data:

The data used for this project is Kaggle's Urban Sound Classification dataset uploaded by Pavan Sanagapati, which includes 8732 sound excerpts of less than or equal to 4 seconds in .wav format. These files have sounds of eight categories:

a) Air Conditioner.
b) Car Horn.
c) Children Playing.
d) Dog bark.
e) Drilling.
f) Engine Idling.
g) Gun Shot.
h) Jackhammer.
i) Siren.
j) Street Music.

The dataset is divided into Train and Test folder and comes with 2 .CSV file that contains two columns for Train dataset:

1) ID: A distinct ID of each sound file.
2) Class: Classification of sound file

The Test dataset .CSV file contains only the ID column, representing distinct ID of sound files in Test folder, the Class is to be predicted by the algorithm.

Feature Extraction:

Feature extraction is the most important part for designing a machine learning model. Choosing appropriate features helps the model work well. For classifying sound, we convert the time-domain signal to mel-frequency scale which is commonly used for speech processing. Here is a short intro of the terms used for sound processing.

Cepstrum:

A cepstrum is the result of taking the inverse Fourier transform (IFT) of the logarithm of the estimated spectrum of a signal.

Mel-frequency cepstrum (MFC):

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Mel-Frequency Cepstral Coefficient (MFCC):

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC[1]. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum").

The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

Melspectrogram:

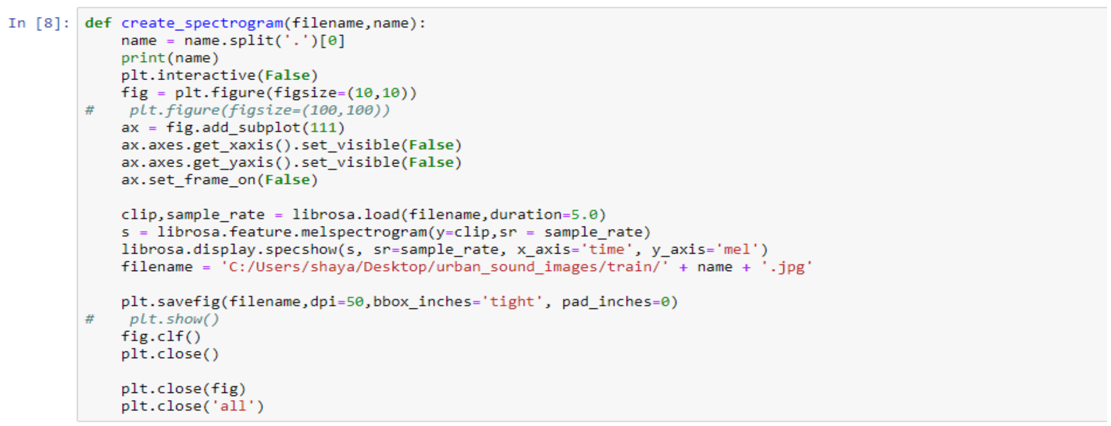
An object of type MelSpectrogram represents an acoustic time-frequency representation of a sound: the power spectral density P(f, t).It is sampled into a number of points around equally spaced times ti and frequencies fj (on a Mel frequency scale).

The mel frequency scale is defined as:

$$mel = 2595 * \log10 (1 + hertz / 700)$$

Methodology:

For sound classification, we use Python-based library called “librosa” to generate the images of melspectrogram of each sound excerpt and use 2-D Convolutional Neural Network to make a sound classifier. First, we read the sound files and use the following function to create spectrogram images and save them in a folder.



```
In [8]: def create_spectrogram(filename,name):
            name = name.split('.')[0]
            print(name)
            plt.interactive(False)
            fig = plt.figure(figsize=(10,10))
        #    plt.figure(figsize=(100,100))
            ax = fig.add_subplot(111)
            ax.axes.get_xaxis().set_visible(False)
            ax.axes.get_yaxis().set_visible(False)
            ax.set_frame_on(False)

            clip,sample_rate = librosa.load(filename,duration=5.0)
            s = librosa.feature.melspectrogram(y=clip,sr = sample_rate)
            librosa.display.specshow(s, sr=sample_rate, x_axis='time', y_axis='mel')
            filename = 'C:/Users/shaya/Desktop/urban_sound_images/train/' + name + '.jpg'

            plt.savefig(filename,dpi=50,bbox_inches='tight', pad_inches=0)
        #    plt.show()
            fig.clf()
            plt.close()

            plt.close(fig)
            plt.close('all')
```

The melspectrogram of sound of siren is shown in Figure 1.1. It shows how the frequency components that make up the sound vary over time.

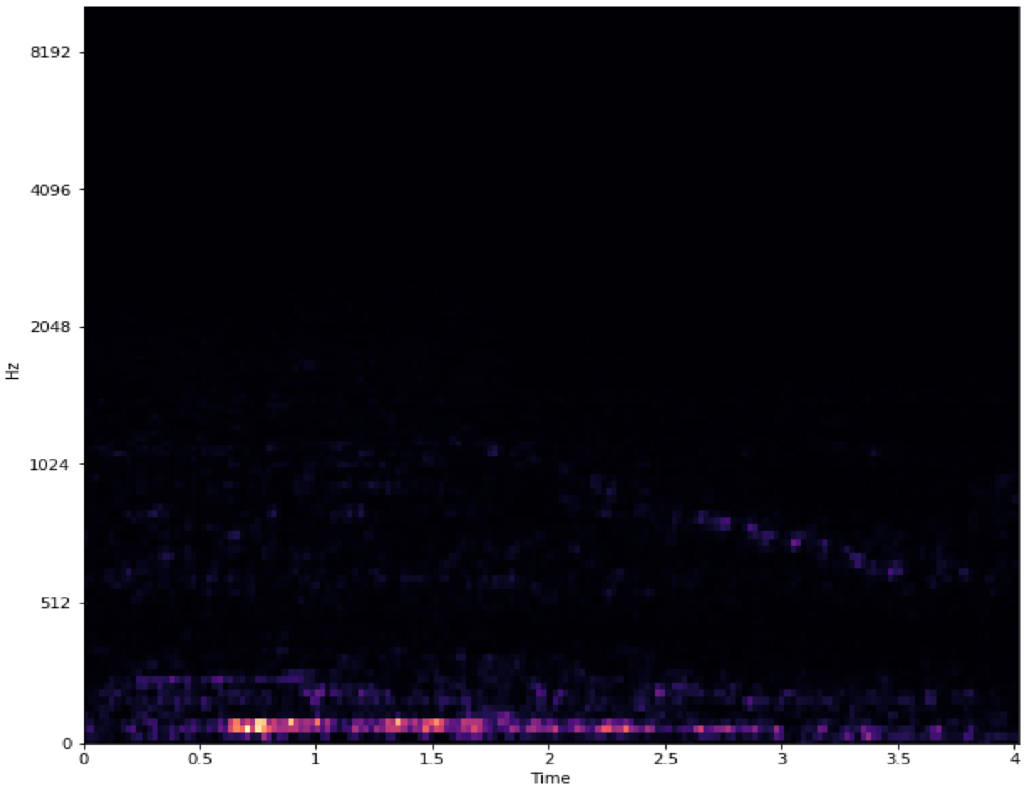
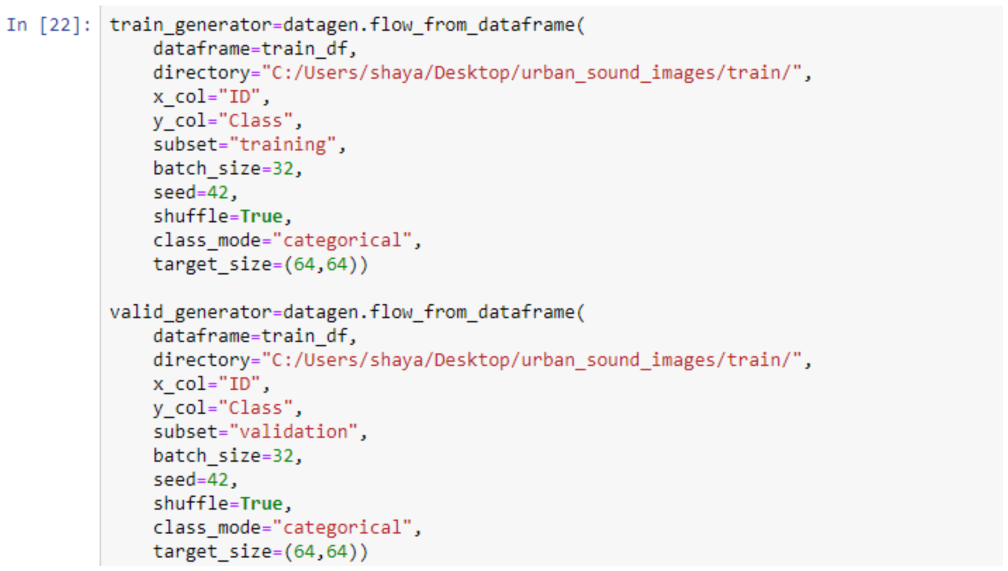


Figure 1.1: Melspectrogram of sound of siren.

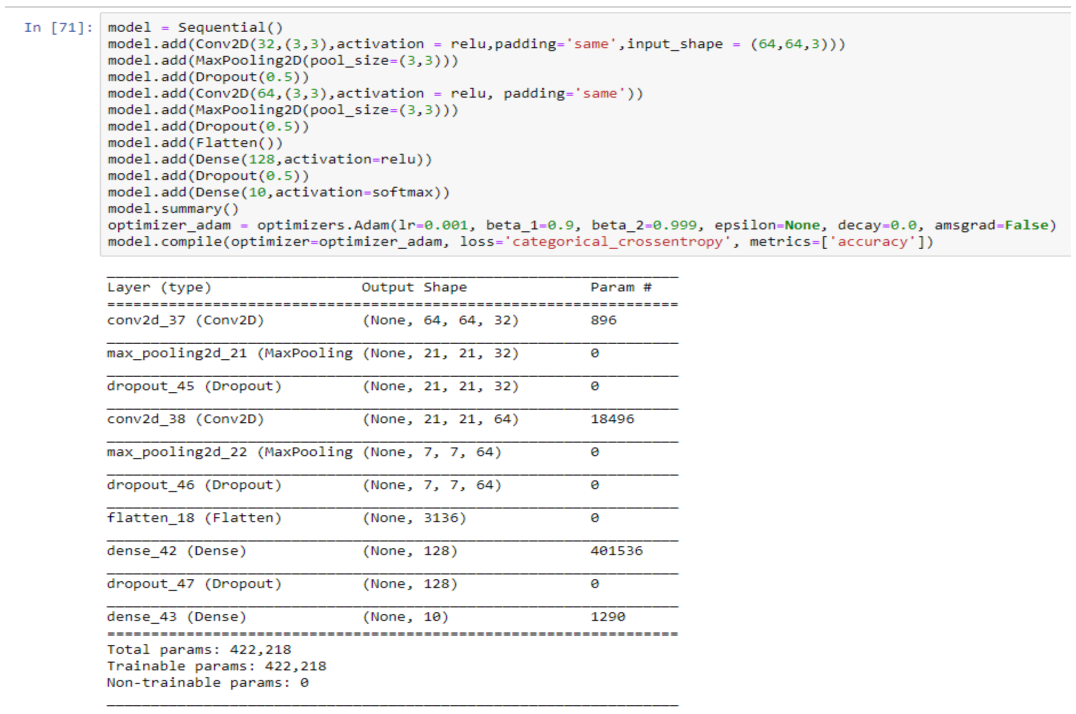
For training our model, we load the saved melspectrogram images and convert them to 64x64 pixels before feeding in batches of 32 images to our neural network. This is shown in Python code below:



```
In [22]: train_generator=datagen.flow_from_dataframe(
             dataframe=train_df,
             directory="C:/Users/shaya/Desktop/urban_sound_images/train/",
             x_col="ID",
             y_col="Class",
             subset="training",
             batch_size=32,
             seed=42,
             shuffle=True,
             class_mode="categorical",
             target_size=(64,64))

         valid_generator=datagen.flow_from_dataframe(
             dataframe=train_df,
             directory="C:/Users/shaya/Desktop/urban_sound_images/train/",
             x_col="ID",
             y_col="Class",
             subset="validation",
             batch_size=32,
             seed=42,
             shuffle=True,
             class_mode="categorical",
             target_size=(64,64))
```

Following code shows the neural network configuration for our sound classifier. We used Adam optimizer and categorical crossentropy loss as this is a multi-class classification problem.



```
In [71]: model = Sequential()
         model.add(Conv2D(32,(3,3),activation = relu,padding='same',input_shape = (64,64,3)))
         model.add(MaxPooling2D(pool_size=(3,3)))
         model.add(Dropout(0.5))
         model.add(Conv2D(64,(3,3),activation = relu, padding='same'))
         model.add(MaxPooling2D(pool_size=(3,3)))
         model.add(Dropout(0.5))
         model.add(Flatten())
         model.add(Dense(128,activation=relu))
         model.add(Dropout(0.5))
         model.add(Dense(10,activation=softmax))
         model.summary()
         optimizer_adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
         model.compile(optimizer=optimizer_adam, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_37 (Conv2D)           (None, 64, 64, 32)        896       
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 21, 21, 32)        0         
_________________________________________________________________
dropout_45 (Dropout)         (None, 21, 21, 32)        0         
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 21, 21, 64)        18496     
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 7, 7, 64)          0         
_________________________________________________________________
dropout_46 (Dropout)         (None, 7, 7, 64)          0         
_________________________________________________________________
flatten_18 (Flatten)         (None, 3136)              0         
_________________________________________________________________
dense_42 (Dense)             (None, 128)               401536    
_________________________________________________________________
dropout_47 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_43 (Dense)             (None, 10)                1290      
=================================================================
Total params: 422,218
Trainable params: 422,218
Non-trainable params: 0
_________________________________________________________________
```

Results:

Training the model for 120 EPOCHS took >20 minutes on a GTX1060 6GB RAM graphic card and i7-8750H laptop gave the following training and validation loss and accuracy curves:

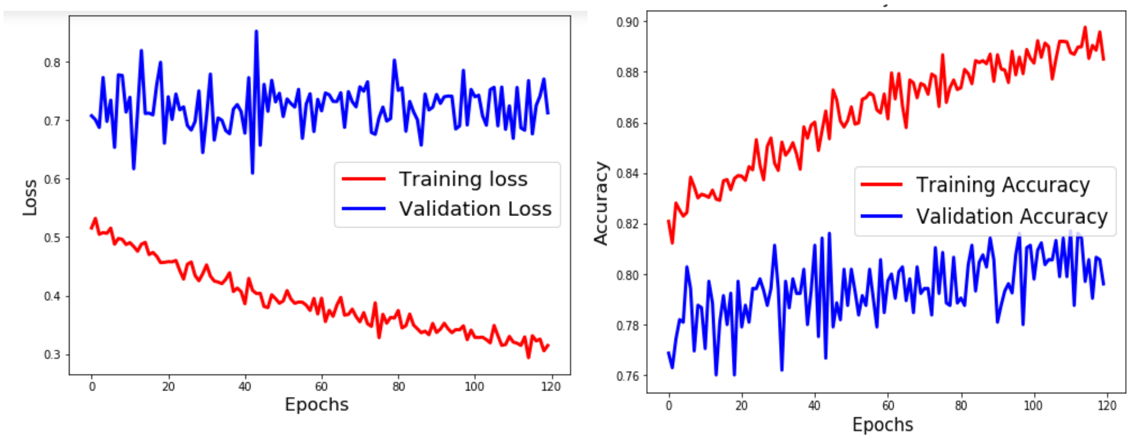


Figure 1.2: Training and validation Loss (left) and accuracy (right) curves of model

Hence, at the end of 120 EPOCHS, the following parameters are observed:

Training loss: 0.3144, Training Accuracy: 88.51%, Validation Loss: 0.7127 – Validation Accuracy: 79.62%.

Even though the accuracy for training and accuracy can be improved further, testing the unseen sound excerpts on the model, gave right classifications for the new sound files.

Conclusion:

Sound classification is important and can be utilized for many applications that can make life easier. This project helps understand the use of Convolutional Neural Network to classify sounds of 10 different categories and performs very well in categorizing the sound files.

Future work on it can include sounds of people arguing, which can lead to fight and/or sound of gunshot or explosion. Installing such systems to alert authorities can help law-enforcing agencies despatch quick help in case of disruption of law and order. Other applications of this project include music genre classification, detecting species of different animals for wildlife preservation etc. Hence, without engaging humans to classify sounds, a machine learning algorithm can do this job perfectly and help make our lives easier.