Vehicle Sound Classification Using Deep Learning

ADVANCED AUDIO DATASETS DEEP LEARNING GUIDE

This article was published as a part of the Data Science Blogathon.

In this article, we will gain knowledge of working with audio data. We will use audio data to solve a very critical and real-world problem. Before going in-depth, let's list what will cover in this article.

Table of Contents

Let's begin…

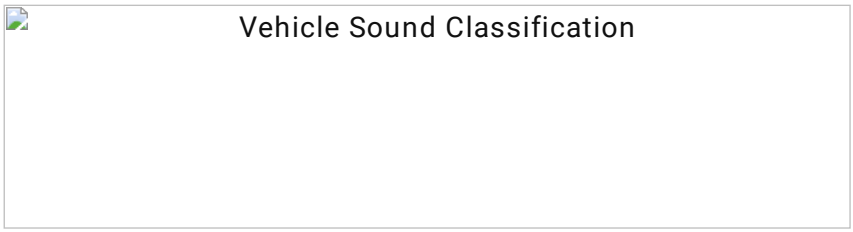
What is an Audio Signal?

Any object that vibrates produces a sound wave.

For example:- A tuning fork produces a characteristic sound when it vibrates. These vibrations have sound, so when an object vibrates, the air molecules oscillate to and fro from their resting position and translate

their energy to the neighbouring molecules. This results in energy transmission from one molecule to another, producing a sound wave.

So, an audio signal is nothing but the representation of a soundwave, and an audio signal has a few critical parameters. It is vital to have a basic understanding of these parameters to help us build our model and solve the problem.

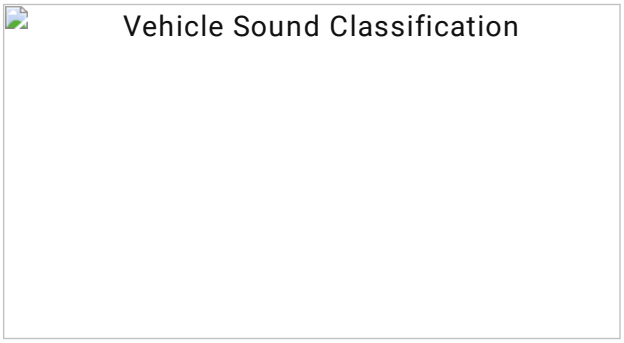


Source:-https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373581-introduction-to-audio-data

Parameters of an Audio Signal

1) Amplitude:-

One of the most critical parameters of the audio signal is amplitude. Amplitude can be defined as the maximum displacement to amend the rest position, and Sometimes the rest position is also known as the central position, as you can see here in this diagram.

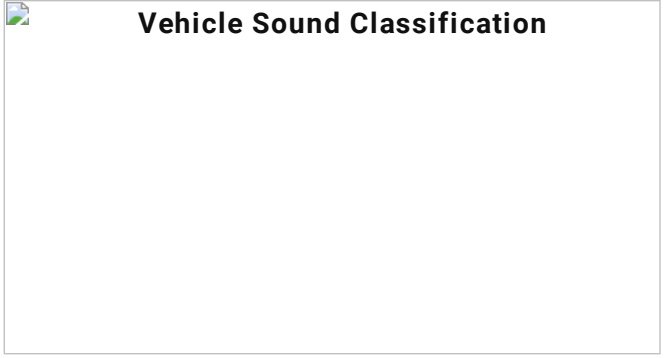


Source:- https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373581-introduction-to-audio-data

The dotted line represents the rest position in the above diagram, and the top division of a particle from its rest position is known as its amplitude.

2) Cycle:-

Audio waves travel in cycles. A cycle is an upward and downward wave movement concerning its main position.



Source:- https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373581-introduction-to-audio-data

3) Frequency:-

This refers to how fast a signal is changing over some time.

In other words, it can be defined as the number of cycles, per second as shown in this figure.

The first signal is an example of high frequency, whereas is the second signal is an example of low frequency.

Source: https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373581-introduction-to-audio-data

Now let's discuss different types of audio signals.

Types of Audio Signals

There are broadly two different types of signals, in our day to day life,

- Digital
- Analog.

1) Digital Signal:-

A digital signal is a discrete representation of a signal over some time. Here a finite number of values exist between any two consecutive time instances.

Source:- Author

For example, in this image, there is only one value that this signal can have at the moment, either zero or one. You can also say that it is a discrete wave that carries information in binary form.

2) Analog Signal:-

An analog signal is a continuous wave that changes over time. In an analog signal, an infinite number of samples exist between any two consecutive time instances. To be able to work on and process analog signals, we have to convert them to digital signals. Let's see how it is done.

Source:- Author

Analog to Digital Conversion

This can be done with the help of sampling.

1) Sampling

Source:- https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373581-introduction-to-audio-data

In this image, the blue wave is an analog signal, and these red circles are type points sampled from this curve.

In sampling, after a fixed time interval the value of the analog signal is measured and using a threshold value converted to Digital Signal. This sampled signal can later be used to reconstruct the original continuous signal.

2) Sampling Rate

The average number of samples captured in one second is called sampling rate, the higher the sampling rate better the sound quality.

Source: https://musictech.com/guides/essential-guide/science-of-signal-sampling/

However, a high sampling rate comes at the cost of more memory space.

Just for your information on the telephone, the sampling rate is 8000 samples per second or 8000 hertz.

Now let's move on to the audio signal representation.

Ways to Represent Audio Signal

1) Time Domain

In the time domain, the audio signal is represented by the amplitude as a function of time. It means that the amplitude is recorded at different intervals of time.

In simple words, it is a 2-D plot between amplitude and time. The X-axis represents the time and the Y-axis represents the amplitude. The raw audio is already available in the time domain, so we can process it directly, and start working on it.

2) Spectrogram

The second type of representation of audio data is the spectrogram. It is a 2-D plot between time and frequency. Each point in the plot also represents the amplitude of a particular frequency at a particular time in terms of intensity colors. In simple terms, the spectrogram is a spectrum of frequencies as it varies with time. The X-axis represents the time whereas the Y-axis represents the frequency and the value at a particular point represents the amplitude of that frequency at that particular time. The low intensity represents lower amplitude and vice versa. So we will convert the audio data to spectrogram first, and pass the spectrogram as an input to the model.

Source: https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373581-introduction-to-audio-data

In the next section, we will understand the problem and how it can be solved using deep learning.

Understanding the Audio classification problem

Problem Statement:- According to the national crime records bureau, nearly 24000 people die each day, due to a delay in getting medical assistance. Any accident victim waits for help at the site and a delay in reaching the destination costs them their lives. One of the biggest reasons for this delay is getting stuck in a traffic jam.

Solution Approach:

Creating an automatic emergency vehicle detection system, that detects the emergency vehicles, from a good enough distance and changes the traffic signals accordingly.

Objective:-

From the data science perspective, it can be solved as an audio classification problem. We will build a model, that will be able to distinguish between the sound of the sirens, the emergency vehicles, and the sound generated by the other vehicles.

Source:https://courses.analyticsvidhya.com/courses/take/fundamentals-of-deep-learning/lessons/11373593-understanding-the-audio-classification-problem

In this next section, we will go through code and explore and preprocess the audio data.

Data Preparation

In the previous section, we discussed the problem of detecting emergency records by identifying the sound made by this siren. Now in this section, we will use a dataset, which is a collection of sound clips of emergency vehicle siren and non-emergency vehicle sound. We will first read the audio data, then we will explore and preprocess the raw audio data. Next, we will make their training and validation sets, and finally, we will get started with model building. We will be using the libraries **Librosa** and **spicy** in this problem to preprocess the audio data.

Librosa is an open-source library in python that is used for audio processing and analysis.

Scipy is a well-known python library, it is used for signal processing, image processing, linear algebra etc.

```
#Audio Processing Libraries import librosa from scipy import signal #For Playing Audio import IPython.display as ipd #Array Processing import numpy as np #Data Visualization import matplotlib.pyplot as plt
```

Here in this section, we will use the above-mentioned libraries to create audio features. We will use the Ipython display to play a few sample audio clips. e will use **numpy** for array processing and **matplotlib** for data visualization. Now coming onto the data science, I will fetch the dataset from file audio.zip, which is the data that we will be working with. You can download it from the link given below this link.

Dataset link:- https://drive.google.com/file/d/14ZCIdME9M1CN7Bu7MAbZ5qY9X0HWr0nC/view

It contains two files, one the emergency vehicle sound, and the other non-emergency vehicle sound. these two files contain multiple audio clips, recorded at different places. We will load these audio clips, with the help of the load function from **librosa**.

```
path='audio/emergency.wav' emergency,sample_rate = librosa.load(path, sr = 16000) path='audio/non emergency.wav' non_emergency,sample_rate= librosa.load(path, sr =16000)
```

This parameter **sr** is the sampling rate, which is the number of samples per second. We have used a sampling rate of 16000 to read these two audio clips and the duration of these audio clips is around 23 minutes and 27 minutes. Basically, we are preparing a data set of audio chunks to train a deep learning model to perform audio Classification.

```
duration1 = librosa.get_duration(emergency,sr=16000) duration2 = librosa.get_duration(non_emergency,sr=16000) print("Duration of an emergency and Non Emergency (in min):",duration1/60,duration2/60)
```

```
Duration of an emergency and Non Emergency (in min): 22.920290625 27.160834375
```

Preparing Data

Let me show you how we are going to prepare these chunks. Let's assume that we are working on this audio clip. Each cell here is one second of audio, this audio clip has seven cells, so it is a seven-second cell. So it is a seven-second of audio data.

Source: Author

Now we will break down this audio clip into audio chunks of two seconds each. This is the first audio chunk. To extract the other chunk we will slide the window to the second timestamp one step, like this.

Source: Author

So from a seven-second audio sequence, we get as many as six sequences of length two seconds each.

Source: Author

Now to perform the same task, we have a few lines of code to prepare data.

It will split the audio clips into audio chunks of two seconds or 32000 samples. So, you can see here this function gives him three arguments. The audio data which is an array is the number of samples which is set to 32000 and the sampling rate which is set to 16000.

The below code prepares the required audio chunks as mentioned above. At every iteration, we use offset to set the starting and ending positions of the chunk.

```
def prepare_data(samples, num_of_samples=32000, num_of_common=16000): data=[] for offset in range(0, len(samples), num_of_common): start = offset end = offset + num_of_samples chunk = samples[start:end] if(len(chunk)==32000): data.append(chunk) return data
```

Finally, this list named **data** would contain all the extracted samples. We will use the above function to prepare separate sets of audio chanks one for emergency vehicles and the other for non-emergency vehicles.

```
emergency = prepare_data(emergency) non_emergency = prepare_data(non_emergency)
```

So as you can see here, we have got over 1300 emergency vehicle audio chunks, and around 1600 non-emergency vehicle audio chunks.

```
print("No. of Chunks of Emergency and Non Emergency:",len(emergency),len(non_emergency))
```

Let's visualize a few of these chunks.

Visualizing the Audio Data

Here we are using **matplotlib** to plot audio chunk one for each of the emergency vehicles and non-emergency vehicles.

```
plt.figure(figsize=(14,4)) plt.plot(np.linspace(0, 2, num=32000),emergency[103]) plt.title('Emergency') plt.xlabel('Time') plt.ylabel('Amplitude') plt.figure(figsize=(14,4)) plt.plot(np.linspace(0, 2, num=32000),non_emergency[102]) plt.title('Non Emergency') plt.xlabel('Time') plt.ylabel('Amplitude')
```

Source: Author

The first plot is for emergency vehicles and the second plot is for non-emergency vehicles. Now from these audio chunks, we will prepare the training and validation data. We will treat the emergency vehicle audio chunks as one class and the non-emergency vehicle audio chunks as the other class. So it will become a binary classification problem.

The first step here is to combine them into two categories of audio chunks and assign labels of these chunks zero for emergency vehicle chunks and one for the non-emergency vehicle chunks.

```
audio = np.concatenate([emergency,non_emergency]) labels1 = np.zeros(len(emergency)) labels2 = np.ones(len(non_emergency)) labels = np.concatenate([labels1,labels2]) print(audio.shape)
```

The shape of the combined data is 3002 chunks, with 32000 dimensions.

Now, we will split the data into training and validation sets.

```
from sklearn.model_selection import train_test_split x_tr, x_val, y_tr, y_val = train_test_split(np.array(audio),np.array(labels), stratify=labels,test_size = 0.1, random_state=777,shuffle=True)
```

These two arrays, x_tr, and x_val are two-dimensional arrays. The first dimension is the number of chunks and the second dimension is the number of samples which is 32000.

As we know, a sequence model has three dimensions, that are the number of examples or chunks, the number of time steps, and the third dimension being the length of the features. This third dimension is not present in our array as of now. So in this code cell, we are reshaping the two-dimensional arrays into three-dimensional arrays and in the third dimension, we have set it to one.

```
x_tr_features = x_tr.reshape(len(x_tr),-1,1) x_val_features = x_val.reshape(len(x_val),-1,1) print("Reshaped Array Size",x_tr_features.shape)
```

And now if you check the shape of the resultant array, it has three dimensions.

So in this section, we completed a few steps such as:-

1. How to load audio data,

2. How to visualize it.

3. How to break it into chunks

4. And finally how to prepare training and validation sets.

So, now we are ready with our data and we can use it to train a deep learning model.

Let's see how it is done, in the next section.

https://drive.google.com/file/d/14ZCIdME9M1CN7Bu7MAbZ5qY9X0HWr0nC/view

Audio Classification using Time-Domain Features

In this previous section, we loaded two audio clips for emergency and non-emergency records and then we split them into audio chunks. Now we will use these audio chunks to train deep learning of the classification models.

Let's look at the architecture that we are going to use. Note that we are going to use the functional API of Keras here. In this architecture, we are using two sets of convolutional and max-pool layers.

Model Building using CNN

Note that we are using one-dimensional convolutional layers or Conv1D because we want the filters to move across one-dimensional only. The number of filters in the first Conv1D layer is 8, and the height of the filters is 13. the number of filters in the second Conv1D layer is 16 and the height of the filter is 11. then comes down the GLobal max-pool layer, which will convert the two-dimensional array output of the previous layer. The one-dimensional output will be fed to one dense layer as you can see here. Finally, the final layer makes the prediction.

This entire architecture is contained in a function so that you can use this architecture multiple times without having to define it again and again, and this function returns two items, the model itself and the saved model weights.

```
from keras.layers import * from keras.models import * from keras.callbacks import * from keras import backend as K def cnn(x_tr): K.clear_session() inputs = Input(shape=(x_tr.shape[1],x_tr.shape[2])) #First Conv1D layer conv = Conv1D(8, 13, padding='same', activation='relu')(inputs) conv = Dropout(0.3)(conv) conv = MaxPooling1D(2)(conv) #Second Conv1D layer conv = Conv1D(16, 11, padding='same', activation='relu')(conv) conv = Dropout(0.3)(conv) conv = MaxPooling1D(2)(conv) #MaxPooling 1D conv = GlobalMaxPool1D()(conv) #Dense Layer conv = Dense(16, activation='relu')(conv) outputs = Dense(1,activation='sigmoid')(conv) model = Model(inputs, outputs) model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['acc']) model_checkpoint = ModelCheckpoint('best_model.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max') return model, model_checkpoint model, model_checkpoint = cnn(x_tr_features)
```

If you are interested in finding out the shape of the outputs at every layer of our model, then you can see it here in the summary.

```
model.summary()
```

And then we train the model. Note that we are using accuracy here as the evaluation metric since both the classes have more or less the same proportion.

```
history=model.fit(x_tr_features, y_tr ,epochs=10, callbacks=[model_checkpoint], batch_size=32, validation_data=(x_val_features,y_val))
```

```
model.load_weights('best_model.hdf5')
```

Here, we have a plot of model performance in terms of loss reduction. s you see here, the validation loss kept on decreasing till the epoch.

#summarize history for loss

```
plt.plot(history.history['loss'])      plt.plot(history.history['val_loss'])      plt.title('Model      loss')
plt.xlabel('Time') plt.ylabel('epoch') plt.legend(['train','validation'],loc = 'upper left') plt.show()
```

Source: Author

And the overall accuracy of this model on the validation data is over 88%.

```
_, acc = model.evaluate(x_val_features,y_val) print("Validation Accuracy:",acc)
```

Now let's see how we can use this model to classify incoming audio as an emergency record sound or a non-emergency record sound. o here is an audio input to the model, and the model prediction is non-emergency, which is the correct prediction.

```
ind=35 test_audio = x_val[ind] ipd.Audio(test_audio,rate=16000) feature = x_val_features[ind] prob = model.predict(feature.reshape(1,-1,1)) if (prob[0][0] < 0.5 ): pred='emergency' else: pred='non emergency' print("Prediction:",pred)
```

However since the accuracy on the validation set is just over 88%, we can do better than this. Let's try out another model. We will use LSTM instead of the CNN model.

Model Building using LSTM

Let's once again, look at the shape of the input sequences. ere, the number of samples is 32000 and if you are using an LSTM layer, it means that there will be 32000 steps to deal with for every channel. In that case, the model training would be extremely slow and the vanishing gradient might also come into play.

So to overcome this problem, we will use a simple hack, so as you can see here in the shape, instead of keeping the third dimensions as one, we can set it to a higher number, such as 160, so that will, in turn, bring down the time steps from 3200 to just 200, and then we can easily use

these reshaped features or sequences in our model.

```
x_tr_features = x_tr.reshape(len(x_tr),-1,160) x_val_features = x_val.reshape(len(x_val),-1,160) print("Reshaped Array Size",x_tr_features.shape)
```

Now let's look at the model architecture. s of now, there is only one LSTM layer in this architecture, however, you can add more layers if you want. You can call this model the same way we called the CNN-based model.

```
def lstm(x_tr): K.clear_session() inputs = Input(shape=(x_tr.shape[1],x_tr.shape[2])) #lstm x = LSTM(128)(inputs) x = Dropout(0.3)(x) #dense x= Dense(64,activation='relu')(x) x= Dense(1,activation='sigmoid')(x) model = Model(inputs, x) model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['acc']) return model model = lstm(x_tr_features) model.summary()
```

```
mc = ModelCheckpoint('best_model.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

And next, we train this model for 10 epochs, feel free to increase the number of the epochs.

It is just a hyper pyramid.

```
history=model.fit(x_tr_features, y_tr, epochs=10, callbacks=[mc], batch_size=32, validation_data=(x_val_features,y_val))
```

```
model.load_weights('best_model.hdf5')
```

In this epoch versus loss plot, the moment of loss is pretty uneven for both the validation and the train set.

```
#summarize history for loss plt.plot(history.history['loss']) plt.plot(history.history['val_loss']) plt.title('Model loss') plt.xlabel('Time') plt.ylabel('epoch') plt.legend(['train','validation'],loc = 'upper left') plt.show()
```

Source: Author

If you look at the **performance** on the validation set, it is just **0.89**, which is slightly better than what we got with the previous model.

```
_,acc = model.evaluate(x_val_features,y_val) print("Accuracy:",acc)
```

After observing the model-loss for CNN and LSTM I would still go with the CNN-based model instead of LSTM even though LSTM's performance is better than CNN because there the loss decrease gradually and smoothly in CNN and here it is unpredictable.

So in this section, we built a sound classification model using two different kinds of deep learning models, one was CNN-based and the other was LSTM based. We also reshaped the input arrays, keeping in mind the limitations of the LSTM. The features that we have used here in both models are the time-domain features.

The next section will work with the spectrogram feature and see if that brings any improvement in the performance or not.

Audio Classification using Time Spectrogram Features

Let us define a function that computes the spectrogram. Before that, we need to understand how the spectrogram is computed.

Spectrogram computed

Spectrogram accepts the raw audio wave and then breaks it into chunks or windows and then applies fast Fourier transformation(FFT) on each window to compute the frequencies.

Coming to the parameters for computing spectrogram:

- **nperseg** is the Size of the window i.e. no. of samples in each chunk
- **noverlap** is No. of overlapping samples between each window

In the previous section, we built two classification models, one was CNN based and the other was LSTM based.

We used the time-domain feature in both the models, and we got the best accuracy in the range of 88 to 89%.

Now we will work with spectrograms of audio and we will see how they are used as features to train deep learning models.

In this section we will use the **signal** module from **scipy**, to compute a spectrogram for a given audio chunk.

This function **scipy. ignal.spectrogram** returns three arrays. The first array is the array frequencies and the second array of segment times. These two arrays are not of much use for us as of now, but this third array 'spec' is important to us. This represents the spectrogram.

Note that we are returning the log of the spectrogram, because the values in the spectrogram, which are amplitude values, could be extremely small on the scale of 10 to the power minus eight, and it's difficult to distinguish such small values. Therefore, we take the log of the spectrogram.

```
def log_specgram(audio, sample_rate, eps=1e-10): nperseg = 320 noverlap = 160 freqs, times, spec = signal.spectrogram(audio,fs=sample_rate, nperseg=nperseg,noverlap=noverlap,detrend=False) return freqs, times, np.log(spec.T.astype(np.float32) + eps)
```

Visualizing the Spectrogram

You can also use a Librosa library to prepare spectrogram features.

Let's see what these spectrograms look like. his is the function that we will use to visualize the spectrograms. his is function takes in two arguments. The first argument is the 'SPECTROGRAM', and the second argument is the 'label', which is either 'emergency' or 'non-emergency'. n this spectrogram of emergency vehicles, we can see some distinctive lines. Maybe these are due to the sound of the siren, whereas there is no such pattern in the spectrogram of a non-emergency vehicle.

```
def plot(spectrogram,label): fig = plt.figure(figsize=(14, 8)) ax = fig.add_subplot(211) ax.imshow(spectrogram.T, aspect='auto', origin='lower',extent=[times.min(), times.max(), freqs.min(), freqs.max()]) ax.set_title('Spectrogram of '+label) ax.set_ylabel('Freqs in Hz') ax.set_xlabel('Seconds')
```

Calling the function to compute and visualize the spectrogram:

```
freqs, times, spectrogram = log_specgram(emergency[300], sample_rate) plot(spectrogram,"emergency") freqs, times, spectrogram = log_specgram(non_emergency[300], sample_rate) plot(spectrogram,"non emergency")
```

Source: Author

The shape of the spectrogram:

```
spectrogram.shape
```

Extract the spectrogram features

Now, the below function will be used to extract spectrogram features from the same audio chunks that we prepared when we were working with the time-domain features.

```
def extract_spectrogram_features(x_tr): features=[] for i in x_tr: _, _, spectrogram = log_specgram(i, sample_rate) mean = np.mean(spectrogram, axis=0) std = np.std(spectrogram, axis=0) spectrogram = (spectrogram - mean) / std features.append(spectrogram) return np.array(features)
```

We will call this function here, to get the features for both the training set and validation set.

```
x_tr_features = extract_spectrogram_features(x_tr) x_val_features = extract_spectrogram_features(x_val)
```

Model Building using LSTM

Now coming onto the model, we are again using the same LSTM architecture, and we don't have to define the architecture here again.

We can simply call the same function 'LSTM_model' and that is it.

```
model, mc = lstm(x_tr_features) model.summary()
```

Then we train the model with the spectrogram features, and if you look at the last moment, it is still uneven.

```
history=model.fit(x_tr_features, y_tr, epochs=10, callbacks=[mc], batch_size=32, validation_data=(x_val_features,y_val))
```

But not as much as it was with the time-domain features. nd the model's performance on the validation set is outstanding.

```
model.load_weights('best_model.hdf5') _,acc = model.evaluate(x_val_features,y_val) print("Accuracy:",ACC)
```

It has an accuracy of over 94%. o searching through spectrogram features does bring in some significant performance boosts.

Model Building using CNN

Let's see how these features perform with a CNN-based model. e will simply use the same CNN-based model architecture and we will use the function 'conv_model' and we are good to go.

```
model, mc = conv1d(x_tr_features) model.summary()
```

It is also trained for 10 epochs. We are keeping all the hyperparameters the same for all models so that the model comparison remains fair. And if you look at the last moment it is pretty consistent and it is gradually decreasing.

```
history=model.fit(x_tr_features, y_tr, epochs=10, callbacks=[mc], batch_size=32, validation_data=(x_val_features,y_val))
```

```
model.load_weights('best_model.hdf5') _,acc = model.evaluate(x_val_features,y_val) print("Accuracy:",acc)
```

On the validation set, it has not outperformed the previous LSTM model, but it has done a great job by getting an accuracy of over 96%.

The spectrogram features are quite instrumental in solving this audio classification task and trying to improve the model's performance.

There are many things that you can do. You can try out different model architectures with the help of the function API of Keras. You can use different kinds of features, such as frequency domain features, and you can also try to change the input sequence length, which might give a performance boost.

Conclusion

Understanding the theory and fundamentals of audio data is critical for solving the business challenge and developing the necessary model. When it comes to working with audio data, the most difficult task is figuring out how to break audio samples into pieces that can be supplied to the model.

I hope the articles helped you understand how to deal with audio data, how to represent it in both time domain and spectrogram form, and how to design a deep learning model while working with audio data.

About the Author

Hi, I am Kajal Kumari. have completed my Master's from IIT(ISM) Dhanbad in Computer Science & Engineering. s of now, I am working as Machine Learning Engineer in Hyderabad. You can also check out a few other blogs that I have written here.

Article Url - https://www.analyticsvidhya.com/blog/2022/01/vehicle-sound-classification-using-deep-learning/

Kajal Kumari