Sound Classification using Deep Learning

Methodology: The objective of this project will be to use Deep Learning techniques to classify urban sounds. When given an audio sample in a computer readable format (such as a .wav file) of a few seconds duration, we want to be able to determine if it contains one of the target urban sounds with a corresponding likelihood score. Conversely, if none of the target sounds were detected, we will be presented with an unknown score. To achieve this, we plan on using different neural network architectures such as Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs).

First we will extract Mel-Frequency Cepstral Coefficients (MFCC) from the audio samples on a per-frame basis with a window size of a few milliseconds. The MFCC summarises the frequency distribution across the window size, so it is possible to analyze both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification.The next step will be to train a Deep Neural Network with these data sets and make predictions. We will begin by using a simple neural network architecture, such as Multi-Layer Perceptron before experimenting with more complex architectures such as Convolutional Neural Networks.

- For this project we will use a dataset called Urbansound8K. The dataset contains 8732 sound excerpts (<=4s) of urban sounds from 10 classes, which are, Air Conditioner ,Car Horn , Children Playing , Dog bark , Drilling , Engine Idling , Gun Shot , Jackhammer , Siren , Street Music.
- These sound excerpts are digital audio files in .wav format. Sound waves are digitised by sampling them at discrete intervals known as the sampling rate (typically 44.1kHz for CD quality audio meaning samples are taken 44,100 times per second).
- For audio analysis, we will be using the following libraries:
  1. IPython.display.Audio
  2. Librosa

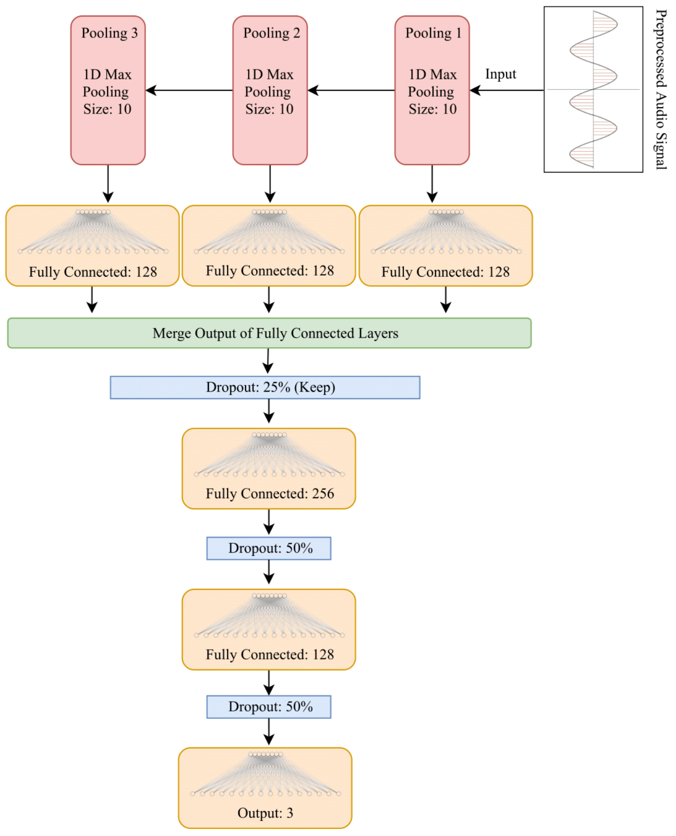
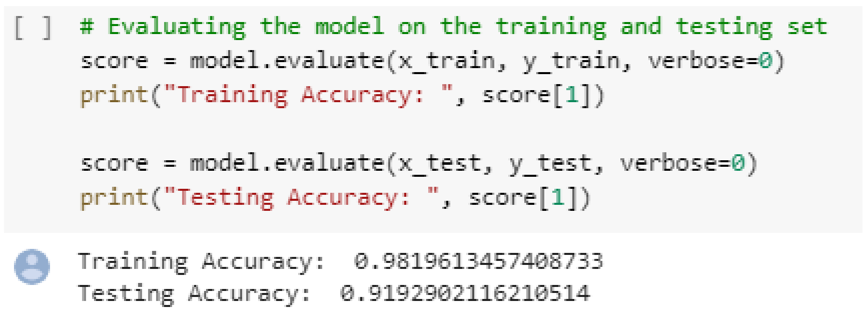


Figure 1 Flow Diagram of CNN-MLP

Results :



```
[ ]  # 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:  0.9819613457408733
Testing Accuracy:  0.9192902116210514
```

Figure 2: Accuracy results of testing data

Accuracy of 92% can be achieved from the testing data.