Deep Learning PROJECT

Classifying Urban sounds using Deep Learning

Sboui Nihed

December 6- 2019

1. Problem Statement

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 **Classification Accuracy score**.

2 Data Exploration and Visualization

2.1 Urban Sound dataset

For this we will use a dataset called Urbansound8K. The dataset contains 8732 sound excerpts (<=4s) of urban sounds from 10 classes, which are:

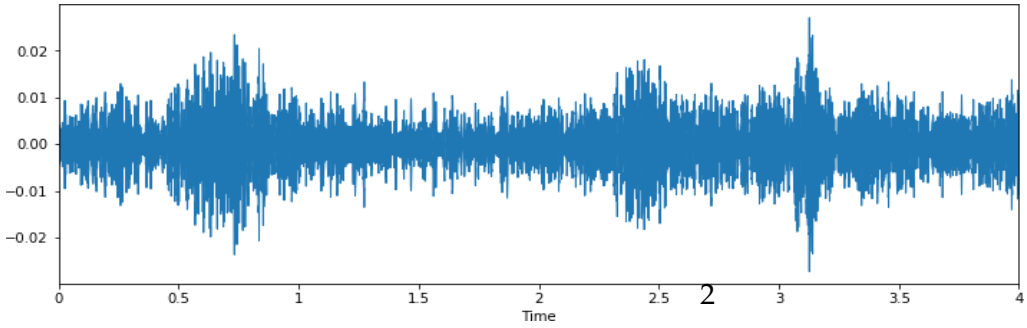
•Air Conditioner
• Car Horn
• Children Playing
• Dog bark
• Drilling
• Engine Idling
• Gun Shot
• Jackhammer
• Siren
• Street Music

These sound excerpts are digital audio files in **.wav format**. Sound waves are digitized 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).

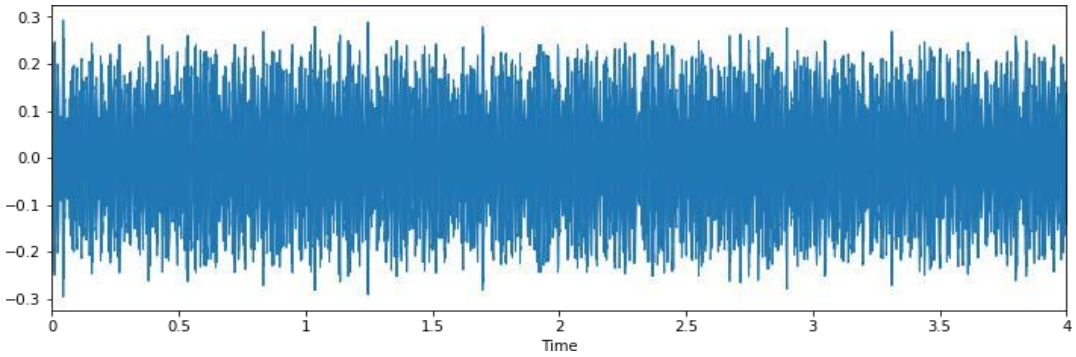
Each sample is the amplitude of the wave at a particular time interval, where the bit depth determines how detailed the sample will be also known as the dynamic range of the signal (typically 16bit which means a sample can range from 65,536 amplitude values).

2.2 Analyzing audio data

From a visual inspection we can see that it is tricky to visualize the difference between some of the classes. For audio analysis, we will be using the following libraries:
IPython.display.Audio This allows us to play audio directly in the Jupyter Notebook.
Librosa is a Python package for music and audio processing by Brian McFee and will allow us to load audio in our notebook as a numpy array for analysis and manipulation.
We will load a sample from each class and visually inspect the data for any patterns.
We will use librosa to load the audio file into an array then librosa.display and matplotlib to display the waveform.



Class: Children playing



Class: engine idling

3. Methodology

3.1. Data Preprocessing and Data Splitting

3.1.1 Preprocessing stage

For much of the preprocessing we will be able to use Librosa's load() function.
We will compare the outputs from Librosa against the default outputs of scipy's wavfile library using a chosen file from the dataset.

Sample rate conversion by default, Librosa's load function converts the sampling rate to 22.05 KHz which we can use as our comparison level.

Bit-depth Librosa's load function will also normalise the data so it's values range between -1 and 1. This removes the complication of the dataset having a wide range of bit-depths.

Merge audio channels Librosa will also convert the signal to mono, meaning the number of channels will always be 1.

Other audio properties to consider At this stage it is not yet clear whether other factors may also need to be taken into account, such as sample duration length and volume levels.

3.1.2 Extract Features

As outlined in the proposal, we will extract Mel-Frequency Cepstral Coefficients (MFCC) from the the audio samples.

The MFCC summarises the frequency distribution across the window size, so it is possible to analyse both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification.

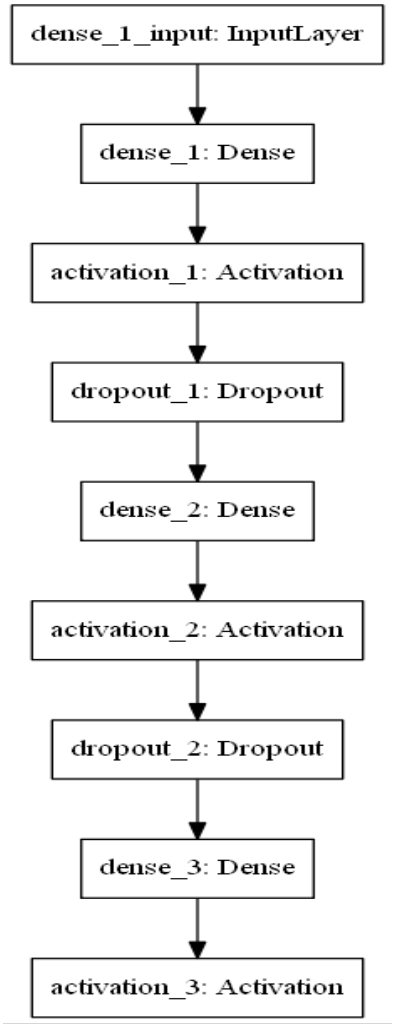
Extracting a MFCC For this we will use Librosa's mfcc() function which generates an MFCC from time series audio data.
Extracting MFCC's for every file
We will now extract an MFCC for each audio file in the dataset and store it in a Panda Dataframe along with its classification label.

3.1.3 Convert the data and labels

We will use sklearn.preprocessing.LabelEncoder to encode the categorical text data into model-understandable numerical data.

3.1.4 Split the dataset

Here we will use sklearn.model_selection.train_test_split to split the dataset into training and testing sets. The testing set size will be 20% and we will set a random state.



4. Implementation

4.1 Initial model architecture - MLP

We will start with constructing a Multilayer Perceptron (MLP) Neural Network using Keras and a Tensorflow backend.
Starting with a sequential model so we can build the model layer by layer.

We will begin with a simple model architecture, consisting of three layers, an input layer, a hidden layer and an output layer. All three layers will be of the dense layer type which is a standard layer type that is used in many cases for neural networks.

The first layer will receive the input shape. As each sample contains 40 MFCCs (or columns) we have a shape of (1x40) this means we will start with an input shape of 40.

The first two layers will have 256 nodes. The activation function we will be using for our first 2 layers is the ReLU, or Rectified Linear Activation. This activation function has been proven to work well in neural networks.

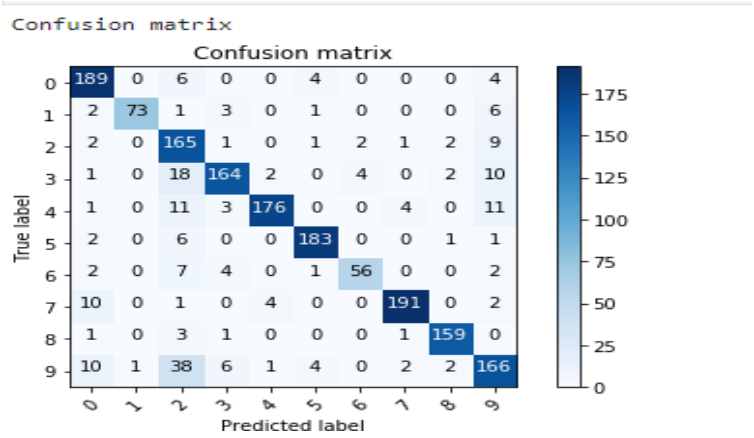
We will also apply a Dropout value of 50% on our first two layers. This will randomly exclude nodes from each update cycle which in turn results in a network that is capable of better generali- sation and is less likely to overfit the training data.

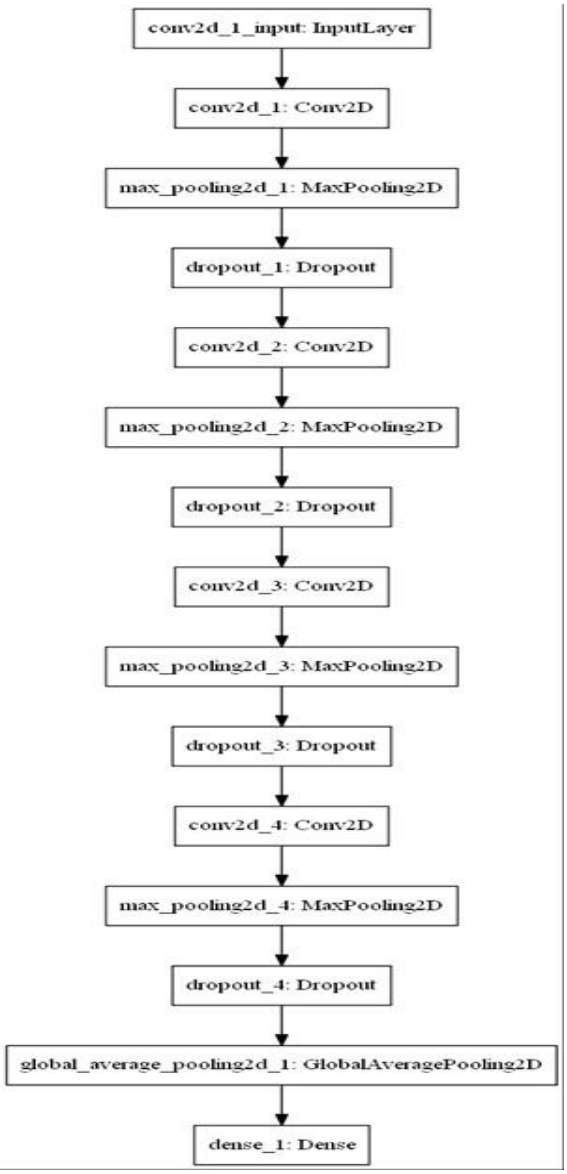
Our output layer will have 10 nodes (num_labels) which matches the number of possible classifi- cations. The activation is for our output layer is softmax.
Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability

Performances

In our initial attempt, we were able to achieve a Classification Accuracy score of:

Training data Accuracy: 0.9302791953086853
Testing data Accuracy: **0.8712077736854553**
Confusion Matrix





We will now see if we can improve upon that score using a Convolutional Neural Network (CNN)

4.2 Convolutional Neural Network (CNN) model architecture

We will modify our model to be a Convolutional Neural Network (CNN) again using Keras and a Tensorflow backend.

Again, we will use a sequential model, starting with a simple model architecture, consisting of four Conv2D convolution layers, with our final output layer being a dense layer.

The convolution layers are designed for feature detection. It works by sliding a filter window over the input and performing a matrix multiplication and storing the result in a feature map. This operation is known as a convolution.

The filter parameterspecifiesthe numberof nodes in each layer. Each layer will increase in size from 16, 32, 64 to 128, while the kernel_size parameter specifies the size of the kernel window which in this case is 2 resulting in a 2x2 filter matrix.

The first layer will receive the input shape of (40, 174, 1) where 40 is the number of MFCC's 174 is the number of frames taking padding into account and the 1 signifying that the audio is mono.

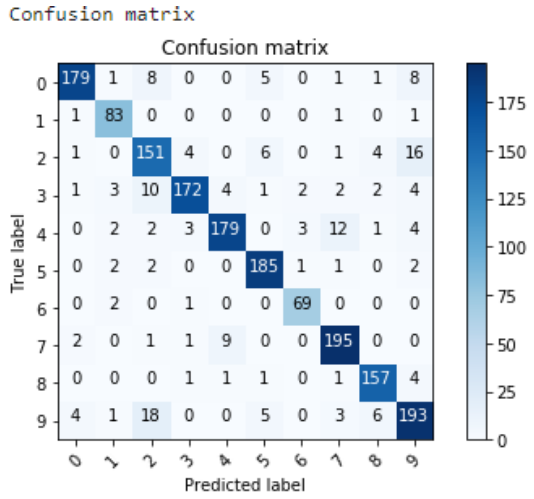
The activation function we will be using for our convolutional layers is ReLU which is the same as our previous model. We will use a smaller Dropout value of 20% on our convolutional layers.

Each convolutional layer has an associated pooling layer of MaxPooling2D type with the final convolutional layer having a GlobalAveragePooling2D type. The pooling layer is do reduce the dimensionality of the model (by reducing the parameters and subsquent computation requirements) which serves to shorten the training time and reduce overfitting. The Max Pooling type takes the maximum size for each window and the Global Average Pooling type takes the average which is suitable for feeding into our dense output layer. Our output layer will have 10 nodes (num_labels) which matches the number of possible classifi- cations. The activation is for our output layer is softmax. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

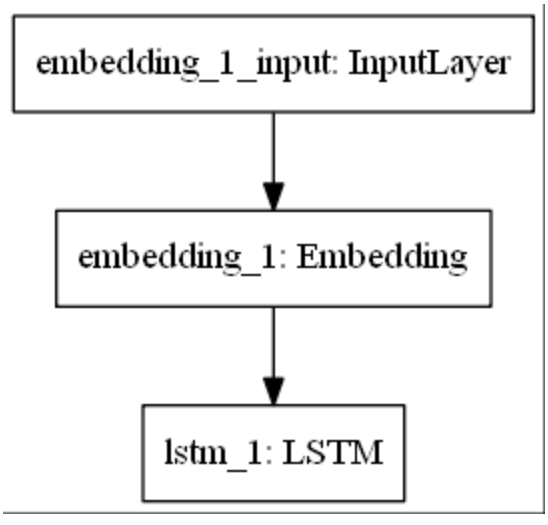
Performances

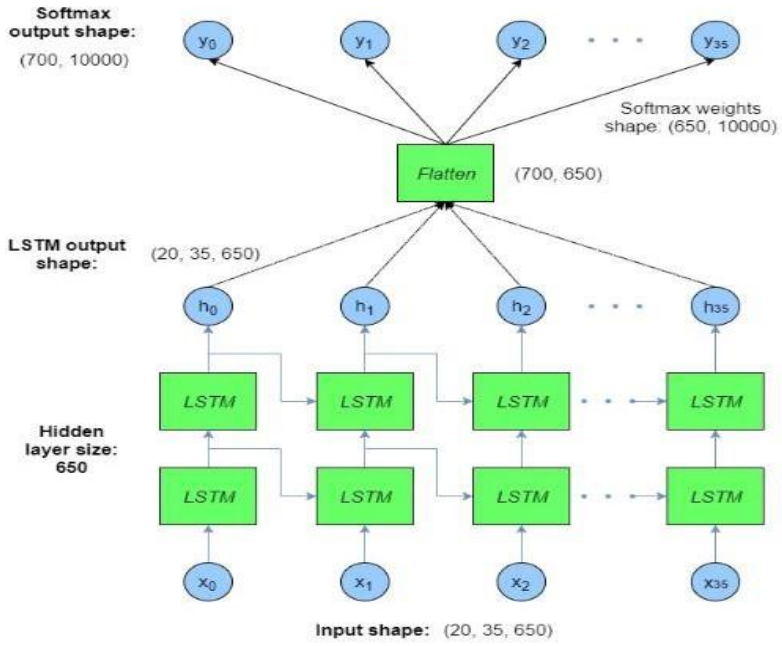
Training Accuracy: 0.9494631290435791
Testing Accuracy: 0.8946765661239624
The Training and Testing accuracy scores are both high and an increase on our initial model. Training accuracy has increased by ~6% and Testing accuracy has increased by ~4%.
There is a marginal increase in the difference between the Training and Test scores (~6% com- pared to ~5% previously) though the difference remains low so the model has not suffered from overfitting.)

Confusion Matrix



4.3 LSTM Recurrent neural network model architecture





Architecture LSTM

Performances

Training Accuracy: 0.9494631290435791

Testing Accuracy: 0.9246765661239624