Università degli Studi di Milano

Data Science and Economics (DSE)

EXPERIMENTAL PROJECT

Urban Sound Classification with Neural Networks

A Project Presented to the module: **Machine Learning**

Author: Fatima Otegen

fatima.otegen@studenti.unimi.it

971106

ABSTRACT

Our brain and auditory system are continuously recognizing each sound that it hears, in their own unique way, since we live in a world surrounded by many types of sound from various sources. We can naturally recognize any sounds, however how cool it would be if computer systems could also identify various sounds and classify them into categories. One of the most extensively utilized applications in Audio Deep Learning is Sound Classification. It entails learning to classify sounds and predict which group they belong to. This sort of task may be utilized in a variety of circumstances, such as categorizing music clips to determine the genre of music or classifying brief utterances by a group of speakers to identify the speaker based on the voice. The main goal of this experimental project is using Deep Learning algorithms to teach a computer how to identify environmental sounds and by what criteria they should be divided into categories.

Neural nets are a means of doing machine learning, in which a computer learns to perform some task by analyzing training examples. It consists of an input layer**,** one or more hidden layers, and an output layer *(MIT's definition)*. Using various Deep Learning Algorithms, they can identify these hidden patterns and correlations in raw data, cluster and classify it. Just like this I will apply Neural Networks techniques to detecting particular Urban Sounds.

DECLARATION

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

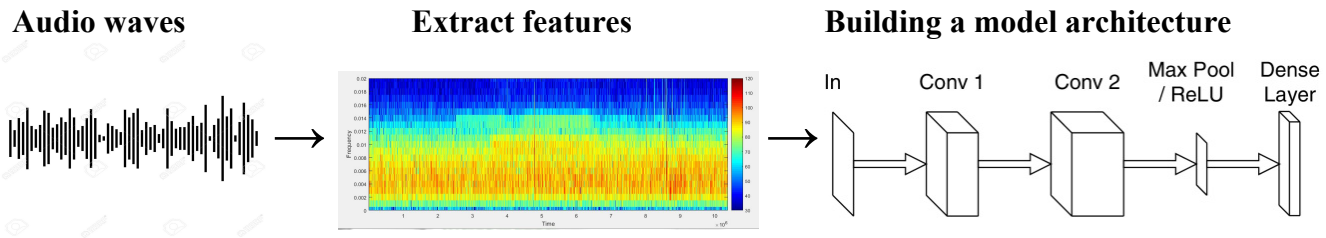
Problem Statement

For this experimental project, I am going to use the *UrbanSound8K* dataset that consists of a corpus of ordinary sounds recorded from day-to-day city life. The given sounds are collected from 10 classes such as:

1. Children playing - 1000
2. Dog bark - 1000
3. Street music - 1000
4. Jackhammer - 1000
5. Engine Idling - 1000
6. Air Conditioner - 1000
7. Drilling - 1000
8. Siren - 929
9. Car Horn - 429
10. Gun Shot - 374

Each sound sample is labeled with the class to which it belongs. The dataset contains 8732 sounds and the max length of each sound is 4 seconds. After downloading the dataset, we see that it consists of two parts:

- Audio folder that contains audio files : It has 10 sub-folders from ‘fold1’ to ‘fold10’. Each sub-folder contains a number of ‘.wav’ audio samples , for example ‘fold5/50104–5–10–0.wav
- Metadata in the ‘metadata’ folder: It has a file ‘UrbanSound8K.csv’ that contains information about each audio sample in the dataset such as its filename, its class label, the ‘fold’ sub-folder location, and so on. The class label is a numeric Class ID from 0–9 for each of the 10 classes. For instance, the number 3 means children playing, 9 for street music, and etc.

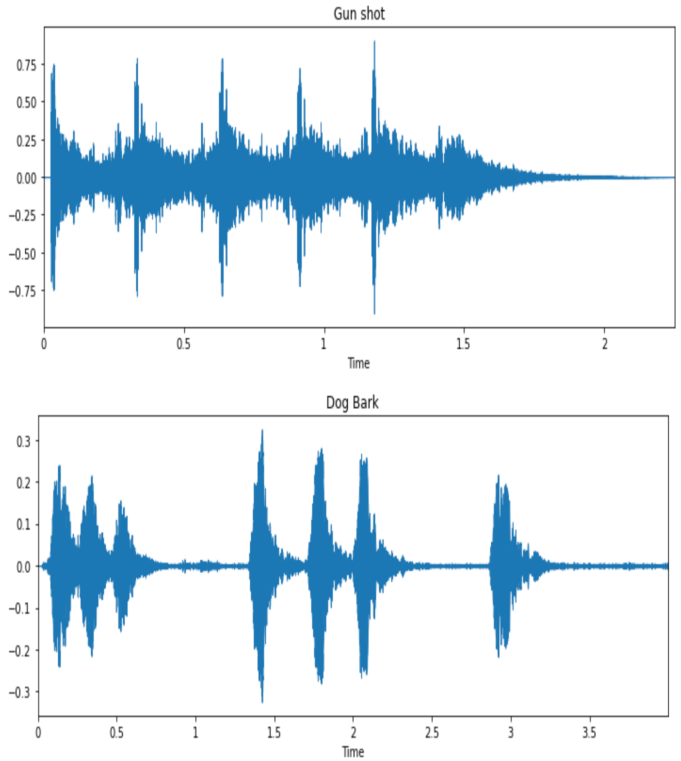


As you can see in the picture above, I will start with sound files, convert them into spectrograms, input them into a Neural Network model, and produce predictions about the class to which the sound belongs.

Data preparations

Audio data for deep learning models will usually start out as digital audio files. From listening to sound recordings and music, we all know that these files are stored in a variety of formats based on how the sound is compressed. Examples of these formats are .wav, .mp3, .wma, .aac, .flac and many more. Python has some great libraries for audio processing. **Librosa** is one of the most popular and has an extensive set of features. (*Ketan Doshi, 2020)*

librosa.display is used to display the audio files in different formats such as wave plot, spectrogram, or colormap. Wave Plots let us know the loudness of the audio at a given time. Spectrogram shows different frequencies playing at a particular time along with its amplitude. Amplitude and frequency are important parameters of the sound and are unique for each audio. librosa.display.waveplot is used to plot waveform of amplitude vs time where the first axis is an amplitude and second axis is time.



To display the waveplot I have used the Librosa package that helps me to load audio in a notebook as a numpy array then to visualize the sound wave. Here we can see that the dataset has a range of varying audio properties that will need standardizing before we can use it to train our model.

We need to iterate through each of the audio sample files and extract, **number of audio channels, sample rate and bit-depth.** For the preprocessing we will be able to use Librosa's load() function.

Since the dataset has a metadata file that contains all information about each audio file, we can use that directly. The below code shows that I have loaded the UrbanSound metadata .csv file into a Panda dataframe.

	slice_file_name	fsID	start	end	salience	fold	classID	class_name
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

Audio data is obtained by sampling the sound wave at regular time intervals and measuring the intensity or amplitude of the wave at each sample. The metadata for that audio tells us the sampling rate which is the number of samples per second. When that audio is saved in a file it is in a compressed format. When the file is loaded, it is decompressed and converted into a Numpy array.

At the most basic level, audio is represented by a stream of samples, each specifying the amplitude of the audio waveform as measured for a given slice of the overall waveform of the audio signal. There are several formats used for the individual samples within an audio file. A one-second piece of audio, for example, would have 44200 numbers if the sampling rate was 44200. The data only contains the amplitude numbers, not the time values, because the measurements are conducted at regular intervals of time. With the given sample rate, we can figure out at what time instant each amplitude number measurement was taken. By default, Librosa's load function converts the sampling rate to 22.05 KHz which we can use as our comparison level.

The bit-depth shows us how many possible values those amplitude measurements for each sample can take. For example, 8 bit audio's amplitude number can be between 0 and 255 (2^8- 1). The bit-depth influences the resolution of the audio measurement — the higher the

bit-depth, the better the audio fidelity. It's likely that we may need to normalise them by taking the maximum and minimum amplitude values for a given bit-depth. The code fragment below shows that almost 66% of our dataset has 16 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.

```
print(audio.bit_depth.value_counts)
16 0.659414
24 0.315277
32 0.019354
8 0.004924
4 0.001031
```

About audio channels: the position of each audio source within the audio signal is called a channel. Each channel contains a sample indicating the amplitude of the audio being produced by that source at a given moment in time. For instance, in stereo sound, there are two audio sources: one speaker on the left, and one on the right. Each of these is represented by one channel (from https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Audio_concepts) . As we can see in the code below, most of our audios are stereo channels. Librosa will also convert the signal to mono, meaning the number of channels will always be 1.

```
print(audio.channels.value_counts)
2 0.915369
1 0.084631
```

Feature Extraction

Models constructed using Deep learning techniques rarely take raw audio dataset directly as input. Therefore the common practice is to convert the audio into a spectrogram. After that we will work with this image format. We'll need to train our model to extract the features. To do so, we'll generate a visual representation of each of the audio samples, which will allow us to find features for classification, using the same techniques used to classify images with high accuracy.

A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It's a representation of frequencies changing with respect to

time for given music signals.

Spectrograms are a handy tool for visualizing a sound's spectrum of frequencies and how they change over time. We will be using a similar technique known as Mel-Frequency Cepstral Coefficients (MFCC). The main difference is that a spectrogram uses a linear spaced frequency scale, whereas an MFCC uses a quasi-logarithmic spaced frequency scale, which is more similar to how the human auditory system processes sounds. 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. The digitized speech samples are processed using MFCC to produce voice features. For this we will use *Librosa's mfcc()* function which generates an MFCC from time series audio data. We are going to iterate through each sound file in the dataset and extract the features and convert into a Panda Dataframe to store it there.

.mfcc is used to calculate mfccs of a signal. By printing the shape of mfccs you get how many mfccs are calculated on how many frames. The first value represents the number of mfccs calculated and another value represents a number of frames available. In our case the number of MFCC and the number of frames are 40 and 173 respectively.

A MFCC cepstral is a matrix, the problem with this approach is that if constant window spacing is used, the lengths of the input and stored sequences are unlikely to be the same. To finish feature extraction from 8732 files will take time.

```
def mfcc_extract(file_name):
    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mfcc = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=50)
        mfccsscaled = np.mean(mfcc.T,axis=0)
    except Exception as e:
        print("Error encountered while parsing file: ", file)
        return None
    return mfccsscaled
```

Note: When using the default sample rate, the *'kaiser_fast'* noticeably reduced the load time

After feature extraction, we need to convert features and corresponding classification labels into numpy arrays. To do so I have used *sklearn.preprocessing.LabelEncoder* to encode the categorical text data into model - understandable numerical data. In addition, we will use this *sklearn* package for splitting our UrbanSound8K dataset into test and training sets. To train

the model on folds: 1, 2, 3, 4, 6, and test it on folds: 5, 7, 8, 9, 10 we have used *sklearn.model_selection.train_test_split,* therefore the testing set size will be 50%.

Building a model: ANN vs CNN

Multilayer Perceptron and CNN are two fundamental concepts in Machine Learning. When we apply activations to Multilayer perceptrons, we get the Artificial Neural Network (ANN) which is one of the earliest ML models. CNN can later be an improvement to the limitations of ANN/ Multilayer perceptrons.

We will start with constructing a Multilayer Perceptron (MLP) Neural Network using Keras and a Tensorflow.