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 them. Just like this, I will apply Neural Networks techniques to detect 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 offenses 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

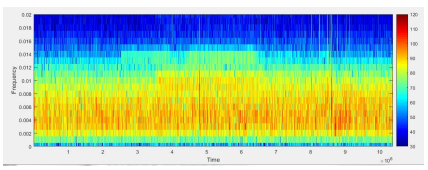
For this experimental project, I am going to use the *UrbanSound8K* dataset that consists of a corpus of ordinary sounds recorded from day-to-day city life. The given sounds are collected from 10 classes such as:

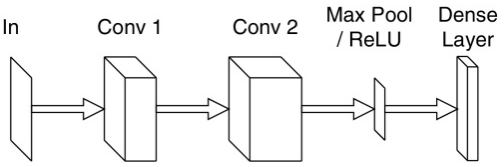
1. Children playing - 1000
2. Dog bark - 1000
3. Street music - 1000
4. Jackhammer - 1000
5. Engine Idling - 1000
6. Air Conditioner - 1000
7. Drilling - 1000
8. Siren - 929
9. Car Horn - 429
10. Gun Shot - 374

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. We can observe that the dataset is divided into two pieces after downloading it:

- The 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
- The 'metadata' folder contains the following metadata: It has a file called 'UrbanSound8K.csv' that includes details about each sound sample within the dataset, including the filename, class label, and location of the 'fold' sub-folder. For each of the ten classes, the class label is an integer Class ID ranging from 0 to 9. For instance, the number 3 means children playing, 9 for street music, and etc.

**Audio waves** → **Extract features** → **Building a model architecture**



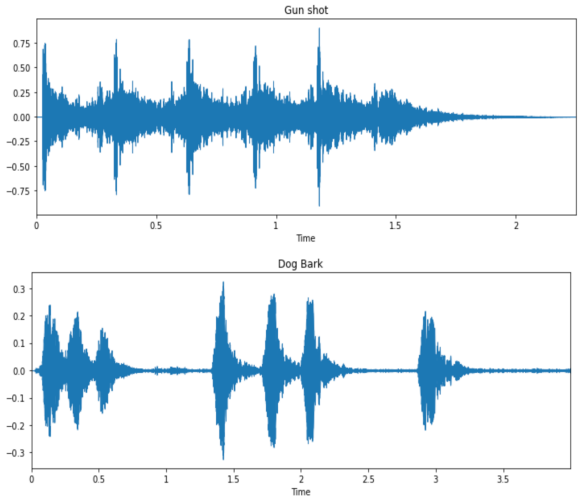


As illustrated in the picture above, I will start with sound files, convert them into spectrograms, input them into a Neural Network model, and produce predictions about the class to which the sound belongs.

## 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. The spectrogram shows different frequencies playing at a particular time along with its amplitude. Amplitude and frequency are essential sound properties that are specific to each audio sample. librosa.display.waveplot plots an amplitude vs. time waveform with the first axis being amplitude and the second axis being 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 depths.** 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 data frame.

| | 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 is represented at its most basic level by a stream of samples, each of which specifies the amplitude of the audio waveform as measured for a specific slice of the overall waveform of the audio signal. Individual samples within an audio file can be saved in a variety of formats. 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.

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

## Feature Extraction

Models constructed using Deep learning techniques rarely take raw audio datasets 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 graphic image of the frequency spectrum of sound changes over time. It's a visual representation of how music signals' frequencies change over time.

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 sound. The MFCC summarizes 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 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 them 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 the 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. 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 the 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.

This research paper's first model is an artificial neural network (ANN). The collected features are utilized to construct a data-frame, which will be used to create train and test sets. The following layers are included in the model:

• **Input Layer:** The input layer is the initial layer, and it contains the spectrogram pictures that the model processes. Because n mfcc was 40 during feature extraction, the first layer is of node 512, with an input shape of (40). The dropout is 0.5 and the activation function is 'ReLu.' This will disregard a few neurons picked at random during the training period.

• **Two Hidden Layers:** In hidden layers, there are two sub-layers. Node 256 is the initial, hidden layer. 'ReLu' is the activation function, and the dropout is 0.5. Node 128 is the second buried layer. The activation functions 'ReLu' are utilized to help solve the vanishing gradient problem. The 'ReLu' activation function will delete the negative component of the argument.

f ReLU(x)=max(0, x)

• **Output Layer:** For the final output layer we have used 'SoftMax' activation function. The output layer will be used to generate the output based on the number of labels.

The optimizer function used for this model is "Adam". Adam optimizing algorithm is an extension of stochastic gradient descent, which is used to update the weights of the network in the training data. In Stochastic gradient descent, a single learning rate is used for all weights, whereas Adam will maintain a different learning rate for each network weight. The training process used 10 epochs. In the end, the mean of 10 fold cross-validation score will be computed to achieve the final result. Ultimately. **accuracy for 10 fold cross validation: 93.1 %**

(Figure 1) shows the plot which contains loss of train set and test sets. (Figure 2) shows the plot which contains accuracy of train and test sets. The observed accuracy is less and loss is more for test sets. This can be improved by changing layers and models.

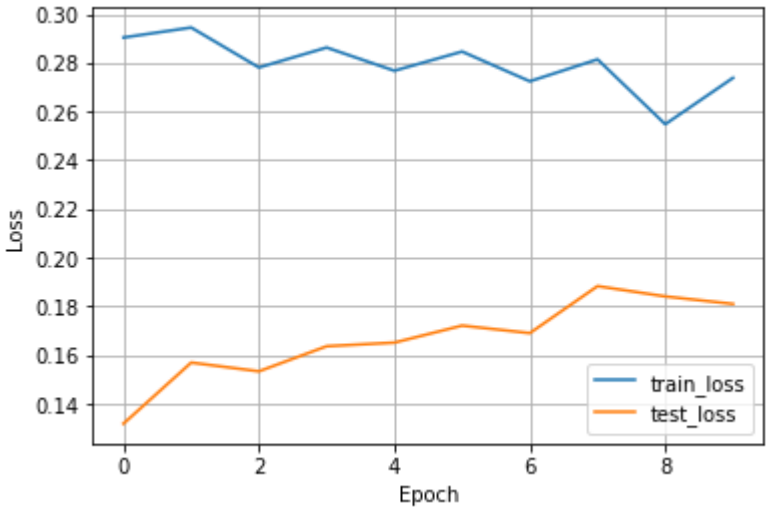



Figure 1: ANN model loss graph

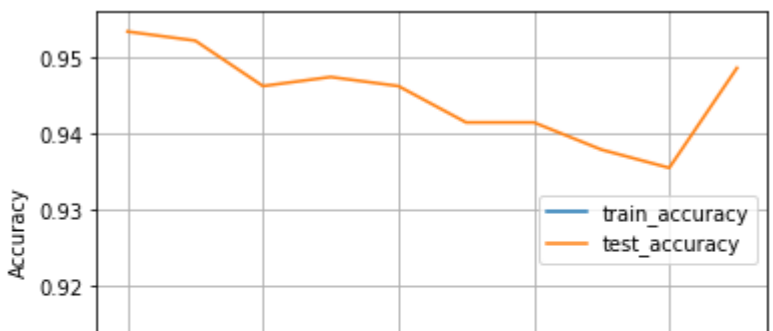


Figure 2: ANN model accuracy graph

**Convolutional Neural Network (CNN 2D) - ReLu.** The second model is Convolutional Neural Network (CNN). This is a two-dimensional convolutional neural network model using the 'ReLu' activation function.(Shamsaldin et al. 2019) After feature extraction, data-frame has been created. This data-frame has been split into a train and validation set using sklearn's train test split, the same as in ANN model. Added layers are as follows:

- **Input Layer:** The first layer is an input layer, which has a filter = 16, the size of the kernel is 2, and the activation 'ReLu'(Rectified Linear Unit). MaxPooling2D has been added with a pool size of (2,2) as well as dropout has been set as 0.2. The negative part of the argument will be removed by using the 'ReLu' activation function by tackling the vanishing gradient problem.
- **Three Hidden Layers:** Hidden layers contain a total of three layers of filters 32, 64, and 128. All have the same activation function 'ReLu' with dropout as 0.2 and MaxPooling2D as (2,2). MaxPooling2D will reduce the dimension of the image by retaining important information. GlobalAveragePooling2D will be used as the next sub-layer, this will be used to apply pooling on a spatial dimension, which is average pooling. A flatten sub-layer is used after the last hidden layer, which will convert the image into the one-dimension image, which will be used as input for the next layer
- **Output Layer:** The final layer is the output layer. The activation function used is 'SoftMax'. The output layer will be used to generate the output based on the number of labels, i.e., 10. 'SoftMax' will give the probability of each class that will sum up equal to 1.

The optimizer used was 'Adam'. Adam optimizing algorithm is an extension of stochastic gradient descent, which is used to update the weights of the network in the training data. Loss has been selected as "categorical_crossentropy" and metrics used is 'accuracy'. The batch size used is 256 and epochs is 100 at the time of fitting the model. CNN 2D model provided very good results when compared to the ANN model.

**Training Accuracy is: 97.25 %**

**Testing Accuracy is: 91.4 %**

Figure 3 shows the loss of CNN 2D ReLu model and it can be observed that loss of both test and train sets are decreasing after each epoch. Figure 4 shows the accuracy of CNN 2D ReLu model and it can be observed that accuracy of both test and train sets are increasing after each epoch.

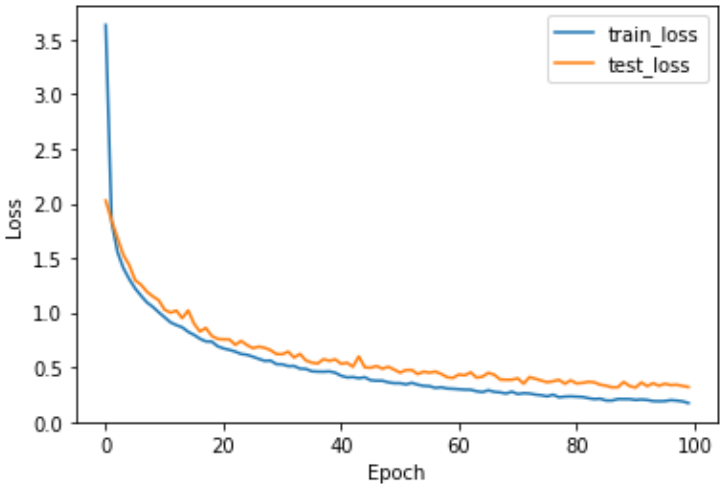



Figure 3: CNN 2D ReLu model loss graph

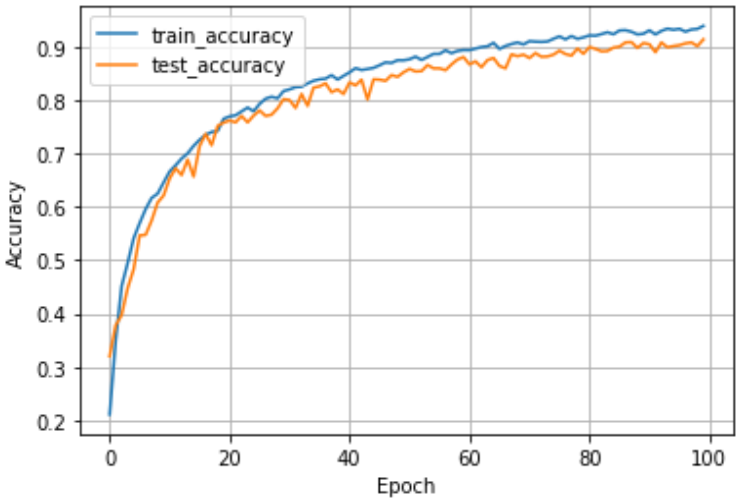



Figure 4: CNN 2D ReLu model accuracy graph

## Conclusion

To summarize, deep learning models are unable to directly analyze audio recordings. The solution to this challenge was to extract features from all of the audio files and as shown the features were extracted using the Librosa library, and a new data-frame was produced with the extracted features. Four deep learning models used this data frame. On this data set, ANN did perform well, with accuracy ranging from 90 to 93 percent. Next, splitted sets are used by CNN 2D - ReLu model. As illustrated in graphs above CNN 2D - ReLu model produced a better result with 97 percent accuracy on the training set and 91 percent accuracy on the test set, which is 2 - 4 % more than ANN model.

Therefore overall, we can surely say that CNN 2D model performs best for the audio dataset, resulting in higher accuracy and lower error rates. In future, multiple layers can be implemented with different parameters and to find the high, medium, and low loudness levels. In addition, the LSTM model and CNN 2D ELU model can be used for this particular dataset and it would be great to clearly see the difference in accuracy.

## References


1. Shamsaldin, Ahmed et al. (Dec. 2019). "The Study of The Convolutional Neural Networks Applications". In: UKH Journal of Science and Engineering 3, pp. 31–40. DOI: 10.25079/ukhjse.v3n2y2019.pp31-40.
2. Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi (2017). "Understanding of a convolutional neural network". In: 2017 International Conference on Engineering and Technology (ICET), pp. 1–6. DOI: 10.1109/ICEngTechnol. 2017.8308186.
3. Huang, Zilong et al. (2020). "Urban sound classification based on 2-order dense convolutional network using dual features". In: Applied Acoustics 164, p. 107243. ISSN: 0003- 682X. DOI: https://doi.org/10.1016/j.apacoust.2020.107243
4. Scarpiniti, Michele et al. (2021). "Deep Belief Network based audio classification for construction sites monitoring". In: Expert Systems with Applications 177, p. 114839. ISSN: 0957-4174.https://www.sciencedirect.com/science/article/pii/S0957417421002803?via%3Dihub
5. Sang, Jonghee, Soomyung Park, and Junwoo Lee (2018). "Convolutional Recurrent Neural Networks for Urban Sound Classification Using Raw Waveforms". In: 2018 26th European Signal Processing Conference (EUSIPCO), pp. 2444–2448. DOI: https://ieeexplore.ieee.org/document/8553247