**AUDIO SCENE CLASSIFIACATION**

***A Thesis Submitted***

***In Partial Fulfilment of the Requirements for the Degree of***

**Batchelor of Technology**

**In**

**Electronic and Communication Engineering**

***Submitted by***

**Kum. Mohini Dharamdas Hurre**

(20EC8083)

***Under the Supervision of***

**Dr. SUMAN SAMUI**

**Assistant Professor**



**ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT**

राष्ट्रीय प्रौद्योगिकी संस्थान दुर्गापुर

**National Institute of Technology Durgapur**

**West Bengal – 713209**

**India**

**APRIL, 2024**



**Department of Electronics and Communication**

**राष्ट्रीय प्रोद्योगिकी संस्थान दुर्गापुर**

**National Institute of Technology Durgapur**

**Mahatma Gandhi Avenue, Durgapur-713209**

**West Bengal**

**India**

**Bona-fide Certificate**

This is to certify that , Kumari Mohini Dharamdas Hurre bearing Roll No. 20EC8083, has successfully completed her project on the thesis entitled, “**Audio Scene Classification**” under my supervision for submission to National Institute of Technology Durgapur towards partial fulfillment for the award of the Degree of Bachelor of Technology in Electronics and Communication Engineering and this thesis is an authentic record of his own work carried out under my supervision.

…………………………………….

**Thesis Supervisor**

Designation,

Electronics and Communication Engineering Department,

National Institute of Technology Durgapur



**Department of Electronics and Communication**

**राष्ट्रीय प्रोद्योगिकी संस्थान दुर्गापुर**

**National Institute of Technology Durgapur**

**Mahatma Gandhi Avenue, Durgapur-713209**

**West Bengal**

**India**

**Certificate of Approval**

The foregoing thesis entitled **“Audio Scene Classification”** is hereby approved as a study of a technology subject carried out in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made, opinion expressed, or conclusion drawn thereon. The thesis is approved only for the purpose for which it is submitted.

………………………………

Examiner(s)

**DECLARATION**

I, Kumari Mohini Dharamdas Hurre bearing 20EC8083 of B. Tech in Electronics and Communication Engineering hereby declare that my B. Tech. Project / Thesis titled “Audio Scene Classification" has been carried out independently by me. I also declare that I have not indulged in any form of plagiarism to carry out the project and also while writing this report. The work or ideas of other authors which are utilized in this report has been properly acknowledged and mentioned in the Bibliography. The work presented in this dissertation or any part thereof has not been submitted anywhere else for the award of any degree in another university. I undertake total responsibility if any traces of plagiarism is found at any later stage. In such a case my degree will be cancelled automatically.

Date:15 April,2024

Place: NIT, Durgapur

(Kumari Mohini Dharamdas Hurre)

Roll Number: 20EC8083

1

### Abstract

Sound class can be used to monitor and classify environmental sounds. This is useful in applications that include wildlife monitoring, noise pollution, acoustic occasion detection, and surveillance systems. with the aid of mechanically classifying sounds such as animal calls, car sounds, alarms, or gunshots, sound class aids in detecting anomalies, identifying unique activities, and alerting government or customers to capacity threats or disturbances.

In recent years, the research on Audio scene classification (ASC), which is dedicated mainly to identify specific sound events, such as identifying dog barking, gunshots, and air conditioning sounds, has received increasing attention. The study result has been used in many practical applications, including robotic hearing, smart home, audio monitoring system, soundscape assessment and so on. Compared with regular and structured sounds such as speech and music, the environmental sound has neither static time patterns like melodies or rhythms nor semantic sequences like phonemes.

Hence, it is difficult to find universal features that can represent various temporal patterns. Besides, the environmental sound contains a lot of noise and some sounds unrelated to the sound event, which lead to complicated composition structure with variability, diversity, and unstructured characteristics. To deal with the above problems, various signal processing methods and machine learning techniques have been used for ASC tasks.

### Nomenclature

|  |  |
| --- | --- |
| **Abbreviation** | **Full form** |
| CNN | Convolutional Neural Networks |
| MLP | Multi-Layer Perceptron |

**Contents**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Title Page** Certificate of Recommendation **Bona-fide Certificate** | | | | | | | | | I  ii  iii |
| **Certificate of Approval** | | | | | | | | | iv |
| **Declaration** | | | | | | | | | v |
| **Acknowledgements**  **Abstract** | | | | | | | | | vi  vii |
| **Nomenclature** | | | | | | | | | viii |
| **Table of Contents** | | | | | | | | | xii |
| Chapter 1 | **Introduction** | | | | | | | | **1** |
|  | 1.1 | Project Overview | | | | | | | 1 |
|  |  | 1.1.1 | |  | | | | | - |
|  |  | 1.1.2 | |  | | | | | - |
|  | 1.  2. | Problem Statement | | | | | | | - |
|  |  | | | | | - | |
|  |  | | | | | -- | |
|  | -- | |
|  | -- | |
| Chapter 2 |  | | | | | | | | **-** |
|  | 2. | Analysis | | | | | | | -- |
|  | | | | | - | |
|  | | | | | - | |
|  | | | | | - | |
| Chapter 3 |  | | | | | | | | **-** |
|  | 3. | Methodology | | | | | | | - |
|  |  |  | | | | | | | - |
| Chapter 4 |  | | | | | | | | **-** |
|  | 4. | Conclusion | | | | | | | - |
|  | 1. | Free-Form Visualisation | | | | | | | - |
|  | 2. | Summary | | | | | | | - |
| Chapter 5 | **Concluding Remarks and Scope for Future Work** | | | | | | | | **-** |
|  | 5.1 | Concluding Remarks | | | | | | | - |
|  | 5.2 | Scope for Future Work | | | | | | | - |
| **References** | | | | | | | | | - |
| **Appendix A:** | | | | | | | | | - |
| **Appendix B:** | | | | | | | | | - |

**1 Definition**

* 1. **Project Overview**

Sounds are all around us. Whether directly or indirectly, we are always in contact with audio data. Sounds outline the context of our daily activities, ranging from the conversations we have when interacting with people, the music we listen to, and all the other environmental sounds that we hear on a daily basis such as a car driving past, the pattern of rain, or any other kind of background noise.

The human brain is continuously processing and understanding this audio data, either consciously or subconsciously, giving us information about the environment around us. Automatic environmental sound classification is a growing area of research with numerous

real world applications. Whilst there is a large body of research in related audio fields such as speech and music, work on the classification of environmental sounds is comparatively scarce. Likewise, observing the recent advancements in the field of image classification where convolutional neural networks are used to to classify images with high accuracy and at scale, it begs the question of the applicability of these techniques in other domains, such as sound classification, where discrete sounds happen over time.

The goal of this capstone project, is to apply Deep Learning techniques to the classification of environmental sounds, specifically focusing on the identification of particular urban sounds. There is a plethora of realworld applications for this research, such as:

• Content-based multimedia indexing and retrieval

• Assisting deaf individuals in their daily activities

• Smart home use cases such as 360-degree safety and security capabilities

• Automotive where recognising sounds both inside and outside of the car can improve safety

• Industrial uses such as predictive maintenance.

My personal motivation for working on sound classification is my background in DSP. I am keen to apply my machine learning knowledge to this domain.

**Problem Statement**

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).

**1 1.3 Metrics**

The evaluation metric for this problem will be the ‘Classification Accuracy’ which is defined as the percentage of correct predictions.

**Accuracy = correct classifications / number of classifications**

Classification Accuracy was deemed to be the optimal choice metric as it is presumed that the dataset will be relatively symmetrical (as we will explore in the next section) with this being a multi-class classifier whereby the target data classes will be generally uniform in size. Other metrics such as Precision, Recall (or combined as the F1 score) were ruled out as they are more applicable to classification challenges that contain a relatively tiny target class in an unbalanced data set.

**2 Analysis**

**2.1 Data Exploration and Visualisation**

**2.1.1 UrbanSound dataset**

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

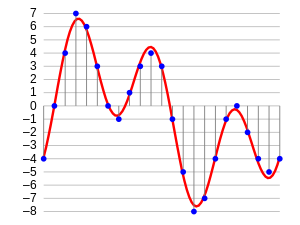
• Street Music

The accompanying metadata contains a unique ID for each sound excerpt along with it’s given class name. A sample of this dataset is included with the accompanying git repo and the full dataset can be downloaded.

**2.1.2 Audio sample file data overview**

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).

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).

This can be represented with the following image:

Therefore, the data we will be analysing for each sound excerpts is essentially a one dimensional array or vector of amplitude values.

### Analysing audio data

For audio analysis, we will be using the following libraries:

1. **IPython.display.Audio** This allows us to play audio directly in the Jupyter Notebook.
2. **Librosa** 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.

You may need to install librosa using pip as follows:

pip install librosa

### Auditory inspection

We will use IPython.display.Audio to play the audio files so we can inspect aurally.

import IPython.display as ipd

ipd.Audio('../UrbanSound Dataset sample/audio/100032-3-0-0.wav')

**2.1.3 Visual inspection**

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.

*# Load imports*

import IPython.display as ipd import librosa

import librosa.display

import matplotlib.pyplot as plt

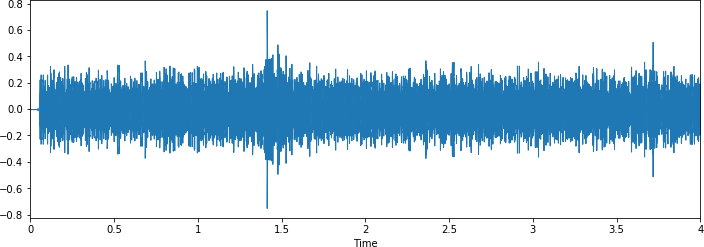
*# Class: Air Conditioner*

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav' plt.figure(figsize=(12,4))

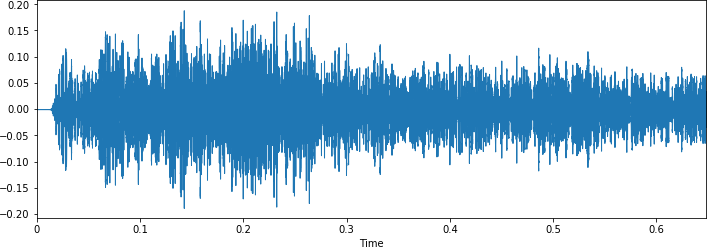
data,sample\_rate = librosa.load(filename)

plot= librosa.display.waveplot(data,sr=sample\_rate)

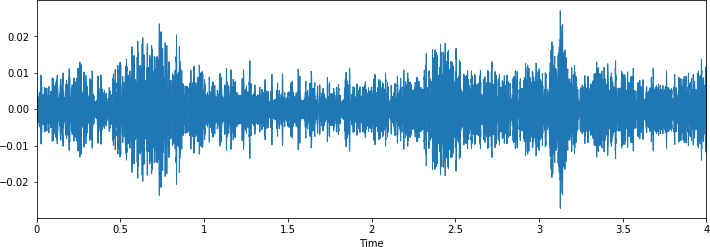
ipd.Audio(filename)



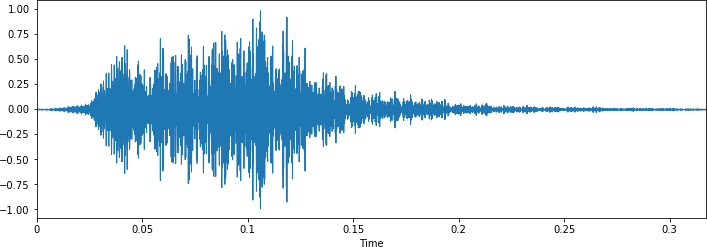
*# Class: Car horn*



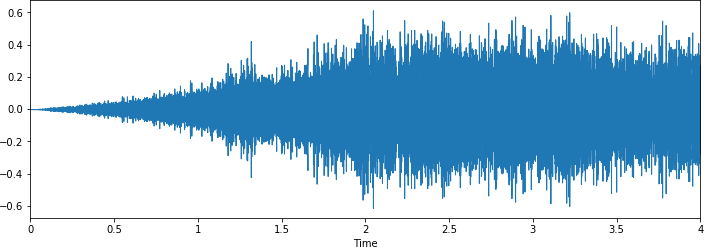
*# Class: Children playing*



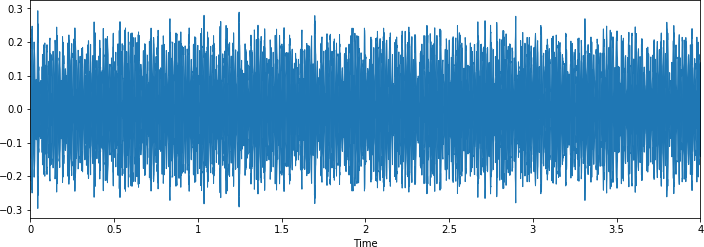
*# Class: Dog bark*



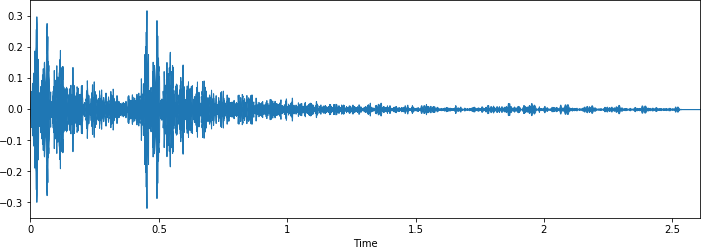
*# Class: Drilling*



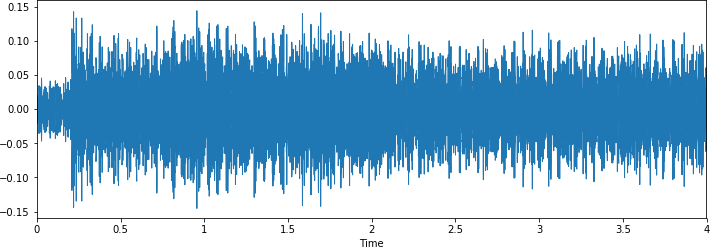
*# Class: Engine Idling*



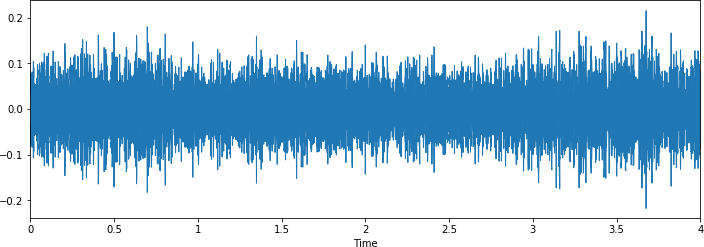
*# Class: Gunshot*



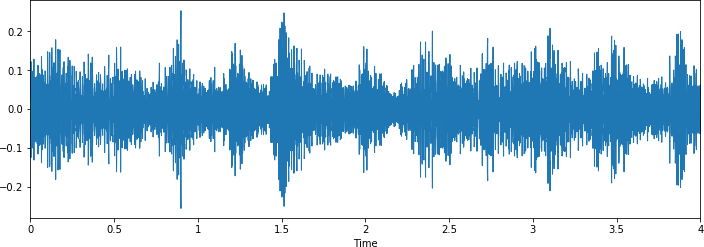
*# Class: Jackhammer*



*# Class: Siren*



*# Class: Street music*



### 2.1.4 Observations

From a visual inspection we can see that it is tricky to visualise the difference between some of the classes.

Particularly, the waveforms for repetitive sounds for air conditioner, drilling, engine idling and jackhammer are similar in shape.

Likewise the peak in the dog barking sample is similar in shape to the gun shot sample (albeit the samples differ in that there are two peaks for two gunshots compared to the one peak for one dog bark). Also, the car horn is similar too. There are also similarities between the children playing and street music.

The human ear can naturally detect the difference between the harmonics, it will be interesting to see how well a deep learning model will be able to extract the necessary features to distinguish between these classes.

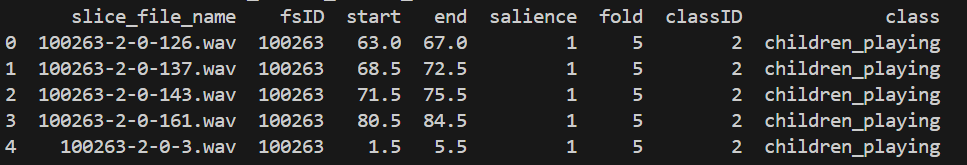
However, it is easy to differentiate from the waveform shape, the difference between certain classes such as dog barking and jackhammer.

**2.1.4 Dataset Metadata**

Here we will load the UrbanSound metadata .csv file into a Panda dataframe.

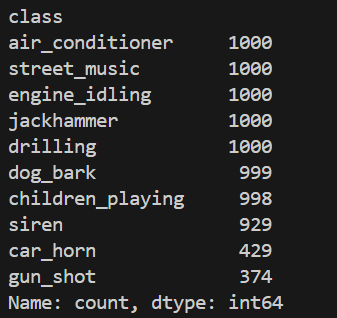
import pandas as pd

metadata = pd.read\_csv('../UrbanSound Dataset sample/metadata/UrbanSound8K.csv') metadata.head()



**2.1.5 Class distributions**

print(metadata.[‘class’].value\_counts())



### Observations

Here we can see the Class labels are unbalanced. Although 7 out of the 10 classes all have exactly 1000 samples, and siren is not far off with 929, the remaining two (car\_horn, gun\_shot) have significantly less samples at 43% and 37% respectively.This will be a concern and something we may need to address later on.

* + 1. **Audio sample file properties**

Next we will iterate through each of the audio sample files and extract, number of audio channels, sample rate and bit-depth.

*# Load various imports* import pandas as pd import os

import librosa

import librosa.display

from helpers.wavfilehelper import WavFileHelper wavfilehelper = WavFileHelper()

audiodata = []

for index, row in metadata.iterrows():

file\_name = os.path.join(os.path.abspath('/Volumes/Untitled/ML\_Data/Urban Sound/UrbanSound8K data = wavfilehelper.read\_file\_properties(file\_name)

audiodata.append(data)

*# Convert into a Panda dataframe*

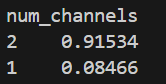
audiodf = pd.DataFrame(audiodata, columns=['num\_channels','sample\_rate','bit\_depth'])

### Audio channels

Most of the samples have two audio channels (meaning stereo) with a few with just the one channel (mono). The easiest option here to make them uniform will be to merge the two channels in the stereo samples into one by averaging the values of the two channels.

*# num of channels*

print(audiodf.num\_channels.value\_counts(normalize=True))



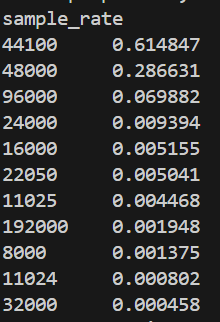
### Sample rate

There is a wide range of Sample rates that have been used across all the samples which is a concern (ranging from 96k to 8k).

This likely means that we will have to apply a sample-rate conversion technique (either up- conversion or down-conversion) so we can see an agnostic representation of their waveform which will allow us to do a fair comparison.

*# sample rates*

print(audiodf.sample\_rate.value\_counts(normalize=True))

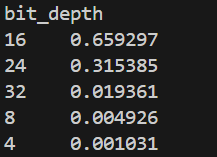


### Bit-depth

There is also a wide range of bit-depths. It’s likely that we may need to normalise them by taking the maximum and minimum amplitude values for a given bit-depth.

*# bit depth*

print(audiodf.bit\_depth.value\_counts(normalize=True))



### Other audio properties to consider

We may also need to consider normalising the volume levels (wave amplitude value) if this is seen to vary greatly, by either looking at the peak volume or the RMS volume.

## Algorithms and Techniques

The proposed solution to this problem is to apply Deep Learning techniques that have proved to be highly successful in the field of image classification.

First we will extract Mel-Spectogram Coefficients from the the audio samples on a per-frame basis with a window size of a few milliseconds. The Mel-Spec applies a frequency-domain filter bank to audio signals that are windowed in time, 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 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.

Multi-layer perceptron’s (MLP) are classed as a type of Deep Neural Network as they are composed of more than one layer of perceptrons and use non-linear activation which distinguish them from linear perceptrons. Their architecture consists of an input layer, an output layer that ultimately make a prediction about the input, and in-between the two layers there is an arbitrary number of hidden layers.

These hidden layers have no direct connection with the outside world and perform the model computations. The network is fed a labelled dataset (this being a form of supervised learning) of input-output pairs and is then trained to learn a correlation between those inputs and outputs. The training process involves adjusting the weights and biases within the perceptrons in the hidden layers in order to minimise the error.

The algorithm for training an MLP is known as Backpropagation. Starting with all weights in the network being randomly assigned, the inputs do a forward pass through the network and the decision of the output layer is measured against the ground truth of the labels you want to predict. Then the weights and biases are backpropagated back though the network where an optimisation method, typically Stochastic Gradient descent is used to adjust the weights so they will move one step closer to the error minimum on the next pass. The training phase will keep on performing this cycle on the network until it the error can go no lower which is known as convergence.

Convolutional Neural Networks (CNNs) build upon the architecture of MLPs but with a number of important changes. Firstly, the layers are organised into three dimensions, width, height and depth. Secondly, the nodes in one layer do not necessarily connect to all nodes in the subsequent layer, but often just a sub region of it.

This allows the CNN to perform two important stages. The first being the feature extraction phase. Here a filter window slides over the input and extracts a sum of the convolution at each location which is then stored in the feature map. A pooling process is often included between CNN layers where typically the max value in each window is taken which decreases the feature map size but retains the significant data. This is important as it reduces the dimensionality of the network meaning it reduces both the training time and likelihood of overfitting. Then lastly we have the classification phase. This is where the 3D data within the network is flattened into a 1D vector to be output.

For the reasons discussed, both MLPs and CNN’s typically make good classifiers, where CNN’s in particular perform very well with image classification tasks due to their feature extraction and classification parts. I believe that this will be very effective at finding patterns within the Mel spectogram much like they are effective at finding patterns within images.

We will use the evaluation metrics described in earlier sections to compare the performance of these solutions against the benchmark models in the next section.

# 3. Methodology

## 3.1 Data Preprocessing and Data Splitting

### Audio properties that will require normalising

Following on from the previous section, we identified the following audio properties that need preprocessing to ensure consistency across the whole dataset:

* + - * Audio Channels
      * Sample rate
      * Bit-depth

We will continue to use Librosa which will be useful for the pre-processing and feature extraction.

### Preprocessing stage

For much of the preprocessing we will be able to use [Librosa’s load() function.](https://librosa.github.io/librosa/generated/librosa.core.load.html)

We will compare the outputs from Librosa against the default outputs of [scipy’s wavfile library](https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.io.wavfile.read.html) 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.

import librosa

from scipy.io import wavfile as wav import numpy as np

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav'

librosa\_audio, librosa\_sample\_rate = librosa.load(filename) scipy\_sample\_rate, scipy\_audio = wav.read(filename)

print('Original sample rate:', scipy\_sample\_rate) print('Librosa sample rate:', librosa\_sample\_rate)

Original sample rate: 44100 Librosa sample rate: 22050

**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('Original audio file min~max range:', np.min(scipy\_audio), 'to', np.max(scipy\_audio)) print('Librosa audio file min~max range:', np.min(librosa\_audio), 'to', np.max(librosa\_audio))

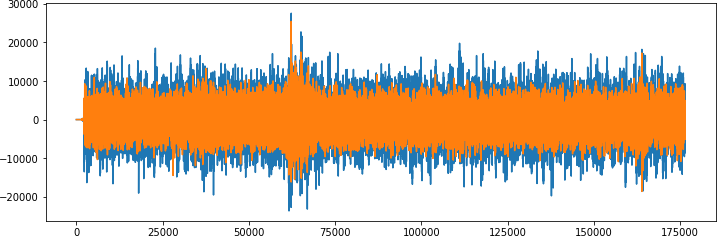
Original audio file min~max range: -23628 to 27507

Librosa audio file min~max range: -0.50266445 to 0.74983937

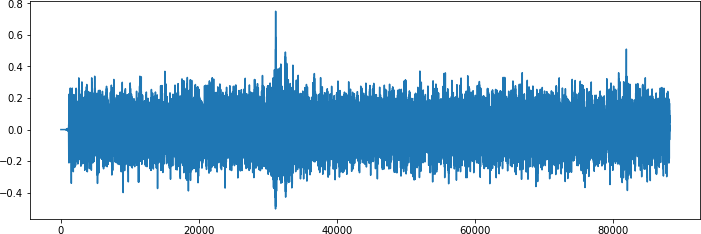
**Merge audio channels** Librosa will also convert the signal to mono, meaning the number of channels will always be 1.

import matplotlib.pyplot as plt

*# Original audio with 2 channels* plt.figure(figsize=(12, 4)) plt.plot(scipy\_audio)



*# Librosa audio with channels merged* plt.figure(figsize=(12, 4)) plt.plot(librosa\_audio)



**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.

We will proceed as is for the meantime and come back to address these later if it’s perceived to be effecting the validity of our target metrics.

### Extract Features

As outlined in the proposal, we will extract [Mel-Spectogram feautres](https://en.wikipedia.org/wiki/Mel-frequency_cepstrum) from the the audio samples.

It 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 Mel Spectogram** For this we will use [Librosa’s melspectogram() function](https://librosa.github.io/librosa/generated/librosa.feature.mfcc.html) which generates mel-features from time series audio data.

 mel\_features = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels=128, fmax=8000)

import librosa.display

librosa.display.specshow(mel\_features, sr=librosa\_sample\_rate, x\_axis='time')

**Extracting Mel spectrogram for every file:** We will now extract an mel features for each audio file in the dataset and store it in a Panda Dataframe along with it’s classification label.

def features\_extractor(file\_name):

    try:

        audio, sample\_rate = librosa.load(file\_name, res\_type='kaiser\_fast')

        mel\_features = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels=128, fmax=8000)

        mel\_scaled\_features = np.mean(mel\_features.T, axis=0)

        except Exception as e:

        print("Error encountered while parsing file: ", file\_name)

        # print(mel\_scaled\_features)

        return None

    return mel\_scaled\_features

*# Load various imports* import pandas as pd import os

import librosa

*# Set the path to the full UrbanSound dataset*

fulldatasetpath = 'C:\\ASC\_24\\Test\\audio'

metadata = pd.read\_csv('C:\\ASC\_24\\Test\\metadata\\UrbanSound8K.csv')

extracted\_features=[]

for index\_num,row in tqdm(metadata.iterrows()):

    file\_name = os.path.join(os.path.abspath(fulldatasetpath),'fold'+str(row["fold"])+'\\',str(row["slice\_file\_name"]))

    final\_class\_labels=row['class']

    data=features\_extractor(file\_name)

    extracted\_features.append([data,final\_class\_labels])

# Convert into a Panda dataframe

featuresdf = pd.DataFrame(extracted\_features, columns=['feature','final\_class\_labels'])

Finished feature extraction from 8732 files

### Convert the data and labels

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

from sklearn.preprocessing import LabelEncoder from keras.utils import to\_categorical

*# Convert features and corresponding classification labels into numpy arrays*

X = np.array(featuresdf.feature.tolist())

y = np.array(featuresdf[‘final\_class\_labels’].tolist())

*# Encode the classification labels*

le = LabelEncoder()

yy = to\_categorical(le.fit\_transform(y))

### 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.

*# split the dataset*

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, yy, test\_size=0.2, random\_state = 42)

## 3.2 Implementation

### 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 Mel features (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.

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten from keras.layers import Convolution2D, MaxPooling2D

from keras.optimizers import Adam from keras.utils import np\_utils from sklearn import metrics

num\_labels = yy.shape[1] filter\_size = 2

*# Construct model*

model = Sequential()

model.add(Dense(256, input\_shape=(40,))) model.add(Activation('relu')) model.add(Dropout(0.5))

model.add(Dense(256)) model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_labels)) model.add(Activation('softmax'))

### Compiling the model

For compiling our model, we will use the following three parameters:

* + - * Loss function - we will use categorical\_crossentropy. This is the most common choice for classification. A lower score indicates that the model is performing better.
      * Metrics - we will use the accuracy metric which will allow us to view the accuracy score on the validation data when we train the model.
      * Optimizer - here we will use adam which is a generally good optimizer for many use cases.

*# Compile the model*

model.compile(loss='categorical\_crossentropy', metrics=['accuracy'], optimizer='adam')

*# Display model architecture summary*

model.summary()

*# Calculate pre-training accuracy*

score = model.evaluate(x\_test, y\_test, verbose=0) accuracy = 100\*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |
| --- | --- | --- | --- |
| dense\_1 (Dense) | (None, | 256) | 10496 |
| activation\_1 (Activation) | (None, | 256) | 0 |
| dropout\_1 (Dropout) | (None, | 256) | 0 |
| dense\_2 (Dense) | (None, | 256) | 65792 |
| activation\_2 (Activation) | (None, | 256) | 0 |
| dropout\_2 (Dropout) | (None, | 256) | 0 |
| dense\_3 (Dense) | (None, | 10) | 2570 |
| activation\_3 (Activation) | (None, | 10) | 0 |

=================================================================

Total params: 78,858

Trainable params: 78,858

Non-trainable params: 0

Pre-training accuracy: 9.5627%

### Training

Here we will train the model.

We will start with 100 epochs which is the number of times the model will cycle through the data. The model will improve on each cycle until it reaches a certain point.

We will also start with a low batch size, as having a large batch size can reduce the generalisation ability of the model.

from keras.callbacks import ModelCheckpoint from datetime import datetime

num\_epochs = 100

num\_batch\_size = 32

start = datetime.now()

model.fit(x\_train, y\_train, batch\_size=num\_batch\_size, epochs=num\_epochs, validation\_data=(x\_tes

duration = datetime.now() - start print("Training completed in time: ", duration)

Train on 6985 samples, validate on 1747 samples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch  Epoch | 00097: val\_loss did not improve from  98/100 | 0.42049 | | | | | |
| 6985/6985 [==============================] | | - 2s 329us/step | - loss: | 0.5246 | - acc: | 0.8241 | - val\_lo |
| Epoch 00098: val\_loss did not improve from Epoch 99/100  6985/6985 [==============================] | | 0.42049  - 2s 347us/step | - loss: | 0.5346 | - acc: | 0.8169 | - val\_lo |
| Epoch 00099: val\_loss did not improve from Epoch 100/100  6985/6985 [==============================] | | 0.42049  - 2s 351us/step | - loss: | 0.5413 | - acc: | 0.8153 | - val\_lo |
| Epoch 00100: val\_loss did not improve from Training completed in time: 0:04:15.582298 | | 0.42049 |  |  |  |  |  |

### Test the model

Here we will review the accuracy of the model on both the training and test data sets.

*# 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: 0.9252684323550465

Testing Accuracy: 0.8763594734511787

The initial Training and Testing accuracy scores are quite high. As there is not a great difference between the Training and Test scores (~5%) this suggests that the model has not suffered from overfitting.

### Predictions

Here we will build a method which will allow us to test the models predictions on a specified audio .wav file.

import librosa import numpy as np

def features\_extractor(file\_name):

    try:

        audio, sample\_rate = librosa.load(file\_name, res\_type='kaiser\_fast')

        mel\_features = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels=128, fmax=8000)

        mel\_scaled\_features = np.mean(mel\_features.T, axis=0)

        except Exception as e:

        print("Error encountered while parsing file: ", file\_name)

        # print(mel\_scaled\_features)

        return None

    return np.array([mel\_scaled\_features])

def print\_prediction(file\_name):

prediction\_feature = extract\_feature(file\_name)

predicted\_vector = model.predict\_classes(prediction\_feature) predicted\_class = le.inverse\_transform(predicted\_vector) print("The predicted class is:", predicted\_class[0], '\n')

predicted\_proba\_vector = model.predict\_proba(prediction\_feature) predicted\_proba = predicted\_proba\_vector[0]

for i in range(len(predicted\_proba)):

category = le.inverse\_transform(np.array([i]))

print(category[0], "\t\t : ", format(predicted\_proba[i], '.32f') )

### Validation

**Test with sample data** Initial sanity check to verify the predictions using a subsection of the sample audio files we explored in the first notebook. We expect the bulk of these to be classified correctly.

*# Class: Air Conditioner*

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav' print\_prediction(filename)

The predicted class is: air\_conditioner

*# Class: Drilling*

filename = '../UrbanSound Dataset sample/audio/103199-4-0-0.wav' print\_prediction(filename)

The predicted class is: drilling

*# Class: Street music*

filename = '../UrbanSound Dataset sample/audio/101848-9-0-0.wav' print\_prediction(filename)

The predicted class is: street\_music.

*# Class: Car Horn*

filename = '../UrbanSound Dataset sample/audio/100648-1-0-0.wav' print\_prediction(filename)

The predicted class is: car\_horn

**Observations** From this brief sanity check the model seems to predict well. One error was observed whereby a car horn was incorrectly classified as a dog bark.

We can see from the per class confidence that this was quite a low score (43%). This allows follows our early observation that a dog bark and car horn are similar in spectral shape.

## Refinement

**Feature Extraction refinement** In the previous feature extraction stage, the Mel features vectors would vary in size for the different audio files (depending on the samples duration).

However, CNNs require a fixed size for all inputs which means we will have to revisit the feature extraction code that we previously wrote. To overcome this we will zero pad the output vectors to make them all the same size.

import numpy as np max\_pad\_len = 174

def features\_extractor(file\_name):

    try:

        audio, sample\_rate = librosa.load(file\_name, res\_type='kaiser\_fast')

        mel\_features = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_mels=128, fmax=8000)

        pad\_width = max\_pad\_len - mel\_features.shape[1]

        mel\_scaled\_features = np.pad(mel\_features, pad\_width=((0, 0), (0, pad\_width)), mode='constant')

    except Exception as e:

        print("Error encountered while parsing file: ", file\_name)

        # print(mel\_scaled\_features)

        return None

    return mel\_scaled\_features

*# Load various imports* import pandas as pd import os

import librosa

*# Set the path to the full UrbanSound dataset*

fulldatasetpath = '/Volumes/Untitled/ML\_Data/Urban Sound/UrbanSound8K/audio/' metadata = pd.read\_csv('../UrbanSound Dataset sample/metadata/UrbanSound8K.csv') features = []

*# Iterate through each sound file and extract the features*

for index, row in metadata.iterrows():

file\_name = os.path.join(os.path.abspath(fulldatasetpath),'fold'+str(row["fold"])+'/',str(ro

class\_label = row["class\_name"] data = extract\_features(file\_name)

features.append([data, class\_label])

*# Convert into a Panda dataframe*

featuresdf = pd.DataFrame(features, columns=['feature','class\_label'])

print('Finished feature extraction from ', len(featuresdf), ' files')

Finished feature extraction from 8732 files

from sklearn.preprocessing import LabelEncoder

from keras.utils import to\_categorical

*# Convert features and corresponding classification labels into numpy arrays*

X = np.array(featuresdf.feature.tolist())

y = np.array(featuresdf.class\_label.tolist())

*# Encode the classification labels*

le = LabelEncoder()

yy = to\_categorical(le.fit\_transform(y))

*# split the dataset*

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, yy, test\_size=0.2, random\_state = 42)

### 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 parameter specifies the number of 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 mel features 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 require- ments) 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 classifications. The activation is for our output layer is softmax. Softmax makes the output sum up to so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Convolution2D, Conv2D, MaxPooling2D, GlobalAveragePooling2D from keras.optimizers import Adam

from keras.utils import np\_utils from sklearn import metrics

num\_rows = 40

num\_columns = 174

num\_channels = 1

x\_train = x\_train.reshape(x\_train.shape[0], num\_rows, num\_columns, num\_channels) x\_test = x\_test.reshape(x\_test.shape[0], num\_rows, num\_columns, num\_channels)

num\_labels = yy.shape[1] filter\_size = 2

*# Construct model*

model = Sequential()

model.add(Conv2D(filters=16, kernel\_size=2, input\_shape=(num\_rows, num\_columns, num\_channels), model.add(MaxPooling2D(pool\_size=2))

model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel\_size=2, activation='relu')) model.add(MaxPooling2D(pool\_size=2))

model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel\_size=2, activation='relu')) model.add(MaxPooling2D(pool\_size=2))

model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel\_size=2, activation='relu')) model.add(MaxPooling2D(pool\_size=2))

model.add(Dropout(0.2)) model.add(GlobalAveragePooling2D())

model.add(Dense(num\_labels, activation='softmax'))

### Compiling the model

For compiling our model, we will use the same three parameters as the previous model:

*# Compile the model*

model.compile(loss='categorical\_crossentropy', metrics=['accuracy'], optimizer='adam')

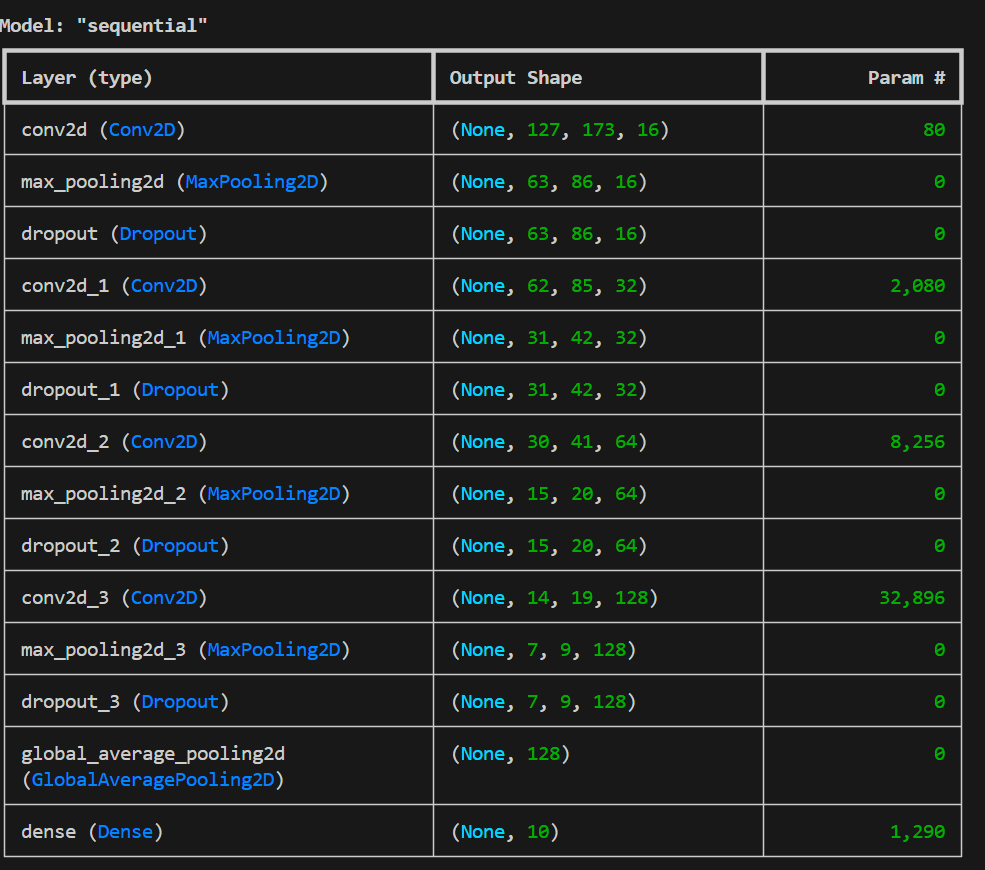
*# Display model architecture summary*

model.summary()

*# Calculate pre-training accuracy*

score = model.evaluate(x\_test, y\_test, verbose=1) accuracy = 100\*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)



Total params: 44,602

Trainable params: 44,602

Non-trainable params: 0

1747/1747 [==============================] - 9s 5ms/step

Pre-training accuracy: 12.0206%

### Training

Here we will train the model. As training a CNN can take a sigificant amount of time, we will start with a low number of epochs and a low batch size. If we can see from the output that the model is converging, we will increase both numbers.

from keras.callbacks import ModelCheckpoint from datetime import datetime

*#num\_epochs = 12*

*#num\_batch\_size = 128*

num\_epochs = 72

num\_batch\_size = 256

checkpointer = ModelCheckpoint(filepath='saved\_models/weights.best.basic\_cnn.hdf5',

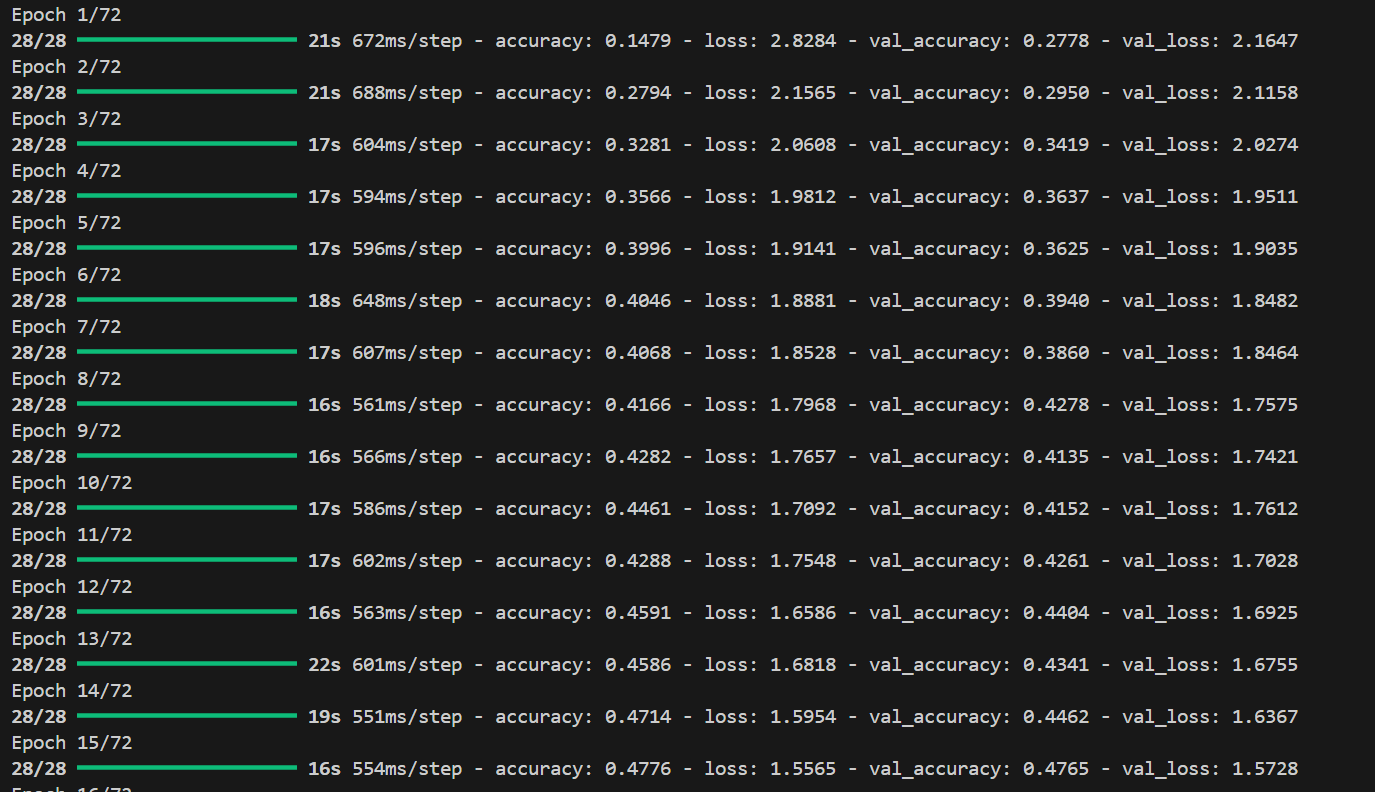
verbose=1, save\_best\_only=True)

start = datetime.now()

model.fit(x\_train, y\_train, batch\_size=num\_batch\_size, epochs=num\_epochs, validation\_data=(x\_tes

duration = datetime.now() - start print("Training completed in time: ", duration)

Train on 6985 samples, validate on 1747 samples



### Test the model

Here we will review the accuracy of the model on both the training and test data sets.

*# 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

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.

### Predictions

Here we will modify our previous method for testing the models predictions on a specified audio

.wav file.

def print\_prediction(file\_name):

prediction\_feature = extract\_features(file\_name)

prediction\_feature = prediction\_feature.reshape(1, num\_rows, num\_columns, num\_channels)

predicted\_vector = model.predict\_class(prediction\_feature) predicted\_class = le.inverse\_transform(predicted\_vector) print("The predicted class is:", predicted\_class[0], '\n')

predicted\_proba\_vector = model.predict\_proba(prediction\_feature) predicted\_proba = predicted\_proba\_vector[0]

for i in range(len(predicted\_proba)):

category = le.inverse\_transform(np.array([i]))

print(category[0], "\t\t : ", format(predicted\_proba[i], '.32f') )

# 4.Conclusion

## Free-Form Visualisation

It was previously noted in our data exploration, that it is difficult to visualise the difference be- tween some of the classes. In particular, the following sub-groups are similar in shape:

* + - Repetitive sounds for air conditioner, drilling, engine idling and jackhammer.
    - Sharp peaks for dog barking and gun shot.
    - Similar pattern for children playing and street music.

## Summary

The process used for this project can be summarised with the following steps:

1. The initial problem was defined and relevant public dataset was located.
2. The data was explored and analysed.
3. Data was preprocessed and features were extracted.
4. An initial model was trained and evaluated.
5. A further model was trained and refined.
6. The final model was evaluated.

From the initial exploration of the data in step 2, I envisaged that the preprocessing work in step 3 would be incredibly time consuming. However, this was actually relatively easy thanks to the Python tool Librosa. I also thought that the feature extraction would be a lot trickier but again Librosa shortened the effort required immensely.

Mel Spectogram we extracted in step 3 perform much better than I had expected.