AUDIO RECOGNITION SYSTEM

USING DEEP LEARNING

PHASE-II PROJECT REPORT

Submitted by

GURU PRASATH S R (16P110)

JEEVANANDHAM K (16P115)

KOUSHIKARAMAKRISHNAN S (16P120)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY :: CHENNAI 600025

MARCH 2020

KARPAGAM
COLLEGE OF ENGINEERING
Rediscover | Refine | Redefine
Coimbatore - 641 032
Autonomous | Affiliated to Anna University, Chennai
Accredited by NAAC with 'A' Grade | NBA (ECE, EEE, CSE, MECH & IT)

BONAFIDE CERTIFICATE

Certified that this project report "AUDIO RECOGNITION SYSTEM USING DEEP LEARNING" is the bonafide work of "GURU PRASATH S R (16P110), JEEVANANDHAM K (16P115), KOUSHIKARAMAKRISHNAN S (16P120)" who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported here in does not form part of any other dissertation on the basis of which degree or award was conferred on an earlier occasion this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. S. ANGEL LATHA MARY, M.E, Ph.D	**Dr. S. KANNIMUTHU, M.E, Ph.D**
HEAD OF THE DEPARTMENT	**SUPERVISOR**

Certified that the candidate was examined in the viva-voce examination held on _____

…….....	…….....
(Internal Examiner)	(External Examiner)

ACKNOWLEDGEMENT

We express our sincere thanks to Karpagam educational and charitable trust for providing necessary facilities to bring out the project successfully. We felt greatness to record our thanks to the chairman **Dr.R.VASANTHAKUMAR, B.E.,(Hons), D.Sc** for all his support and ray of strengthening hope extended.

It is the moment of immense pride for us to reveal our profound thanks to our respected principal **Dr.P.VIJAYAKUMAR, M.E., Ph.D.** who happens to be striving force in all our endeavours.

We express our sincere thanks to **Dr.S. ANGEL LATHA MARY, M.E, Ph.D.** Head of the Department of Computer Science and Engineering for providing an opportunity to work on this project. Her valuable suggestions helped us a lot to do this project.

A word of thanks would not be sufficient for the work of our project guide **Dr.S.KANNIMUTHU, M.E, Ph.D.** Department of Computer Science and Engineering whose efforts and inspiration lead us through every trying circumstance.

We would also like to recollect the courage and enthusiasm that was inculcated in us by our project co-ordinator **Dr.K.BHUVANESHWARI, M.E,Ph.D.** Department of Computer Science and Engineering for valuable guidance and support through the tenure of our project.

We deeply express our gratitude to all the members of the faculty of the department of Computer Science and Engineering for the encouragement, which we received throughout the semester.

ABSTRACT

Audio Recognition has number of applications in different areas of engineering and day-to-day aspects. Nowadays it is being used in health care, telephony military and people with disabilities.

The main objective of the system is to provide support to military professionals, security from unwanted intruders within military camps. The objective is achieved through processing the voice data within the military camps and recognizing the unwanted intruders.

The voice is processed using convolutional neural networks, which is applied on the spectrogram of the audio sample. If an unauthorized voice sample is found, the concerned officials are intimated.

At the initial effort is made to provide the basic operations as discussed above, the model is trained with multilayer perceptron. And to enhance the model, convolutional neural network (CNN) is used.

The speech waves of particular voice form the basis of identification of speaker. The voice features which are extracted are compared with already saved voices in the database for matching. This paper provides review of various voice and speaker recognition systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ZCR	Zero Crossing Rate
RBF	Radial Basis Function
SVM	Support Vector Machines
LPC	Linear Predictive Coding
PLP	Perpetual Linear Prediction
MLP	Multi Layer Perceptron
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
PyPI	Python Package Installer

CHAPTER 1

INTRODUCTION

1.1 DEFINITION

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, listening music and all environmental sounds such as sound of the car, rain, hammer etc.., The human brain is continuously processing and understanding the audio data 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. There may be a lot of research works done on other acoustic sounds but environmental classification is a unique area. Likewise, observing the recent advancements in the field of image classification where convolutional neural networks are used 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.
There are a whole lot of real world applications available:

• 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 • Industrial uses such as predictive maintenance.

The audio signal is usually broken down into smaller segments with a particular time window and features are extracted for each segment. Mel-Frequency Cepstral Coefficients (MFCCs),0 Zero Crossing Rate (ZCR), Spectral Centroid are some of the most widely used features for audio analysis. Many previous efforts have also been made in classifying audio signals using other features such asMPEG-7 descriptors, Linear Prediction coefficients, features derived from statistics of spectrogram image of an audio and Log–Gabor Filter.

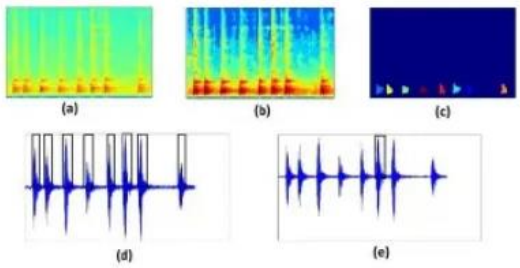


Fig-1: Sound Segmentation of dog bark. (a) Spectrogram of a dog bark; (b) k-means clustering of 1(a) spectrogram with $\epsilon = 10$; (c)Connected components of the labels retained from 1(b); (d) Dog bark segments using the segments obtained by applying k-means for spectrogram image; (e) Selecting the longest segment from 1(d).

Fig1.1: Sound segmentation of Dog Bark

The speech or voice of any human being is his/her unique personal characteristics. No two have almost identical voice there are some features which must be present in one voice and missing from another voice. In order to identify this uniqueness a robust and efficient technique is required so that we can accurately identify the genuine voice from the bunch of fake voices. The voice recognition system is broadly categorized in two ways A. Speaker Identification B. Speaker Verification The process of identifying a voice of a given speech from the group of given speakers is called speaker identification. The speaker whose maximum voice characteristics are matches with the stored voice is identified & the speaker whose voice characteristics are not matched is eligible for new entry in the database.

The known sets of voices are cataloged into two parameters called Open set mode and the Close set mode. With Open set mode the speaker need not to be part of some known speakers. This is used in case of some criminal act where out of multiple suspects identify of main criminal is identified. In close set parameter mode the speaker is part of some known voices already available in database. This method is used for authentication purpose also called biometric security to identify the authorized person out of multiple claimed persons. On the other hand, Speaker Verification is the process of accepting or rejecting the identity claim of a speaker. It is used for the verification of the person claiming for authentication.

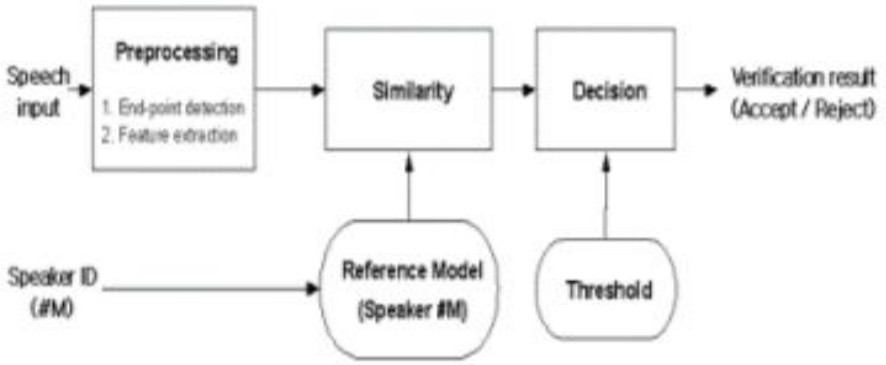


Fig1.2: Workflow of the model

1.2 SPECTOGRAM

Spectrograms represent the frequency content in the audio as colors in an image. Frequency content of milliseconds chunks is stringed together as colored vertical bars. Spectrograms are basically two-dimensional graphs, with a third dimension represented by colors.

1. **Time** runs from left (oldest) to right (youngest) along the horizontal axis.
2. The vertical axis represents **frequency**, with the lowest frequencies at the bottom and the highest frequencies at the top.
3. The amplitude (or energy or "loudness") of a particular frequency at a particular time is represented by the third dimension, **color**, with dark blues corresponding to low amplitudes and brighter colors up through red corresponding to progressively stronger (or louder) amplitudes.

1.3 OBJECTIVE

The following will demonstrate how to apply Deep Learning techniques to the classification of environmental sounds, specifically focusing on the identification of particular 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 **Classification Accuracy score**.

1.4 METRICS

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

.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

where TP is True Positive, TN is True Negative, FP is False Positive, FN is False Negative.

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.

CHAPTER 2

SYSTEM OVERVIEW

2.1 RELATED WORKS

It is referenced from Salamon[1] who proposed a novel which describes five different algorithms with the following accuracies for a audio slice maximum duration of 4 seconds using the same Urban Sound dataset.

Algorithm	Classification Accuracy
SVM_rbf	68%
RandomForest500	66%
IBk5	55%
J48	48%
ZeroR	10%

Table2.1: Related Works

T.Lambrou, P.Kudumakis, R.Speller, M.Sandler and A.Linney on the paper "Classification of Audio Signals using Statistical features on time and wavelet transform domains." This paper presents a study on musical signals classification using wavelet transform analysis in conjunction with statistical pattern recognition techniques.

In [2], To use the deep learning networks for classifying the environmental sounds based on the generated spectrograms of these sounds. It used the spectrogram images of environmental sounds to train the convolutional neural network (CNN) and the tensor deep stacking network (TDSN).

In [3], It suggests a new way of combining JFA and SVMs for speaker verification. It consists in directly using the speaker factors estimated with JFA as input to the SVM. The best results were obtained using the cosine kernel when within-class covariance normalization (WCCN) is also used to compensate for residual channel effects in the speaker factor space.

In [4], It proposes the popular Mel-frequency cepstral coefficients (MFCCs) which describe the audio spectral shape. Environmental sounds, such as chirpings of insects and sounds of rain which are typically noise-like with a broad flat spectrum, may include strong temporal domain signatures. However, only few temporal-domain features have been developed to characterize such diverse audio signals previously. Here, we perform an empirical feature analysis for audio environment characterization and propose to use the matching pursuit (MP) algorithm to obtain

effective time-frequency features. The MP-based method utilizes a dictionary of atoms for feature selection, resulting in a flexible, intuitive and physically interpretable set of features.

In [7], This paper propose a new algorithm that exploits higher order frequency dependencies of source signals in order to separate them when they are mixed. In the frequency domain, this formulation assumes that dependencies exist between frequency bins instead of defining independence for each frequency bin. In this manner, we can avoid the well-known frequency permutation problem. To derive the learning algorithm, we define a cost function, which is an extension of mutual information between multivariate random variables. By introducing a source prior that models the inherent frequency dependencies, we obtain a simple form of a multivariate score function.

CHAPTER 3

REQUIREMENTS SPECIFICATION

3.1 HARDWARE SPECIFICATION

Operating System	:	Windows 10
Tools	:	Python, Jupyter Notebook
Hard Disk Capacity	:	40 GB

3.2 SOFTWARE SPECIFICATION

Processor	:	Intel i3
Processor Speed	:	1.5 GHz
Main Storage	:	4 GB

CHAPTER 4

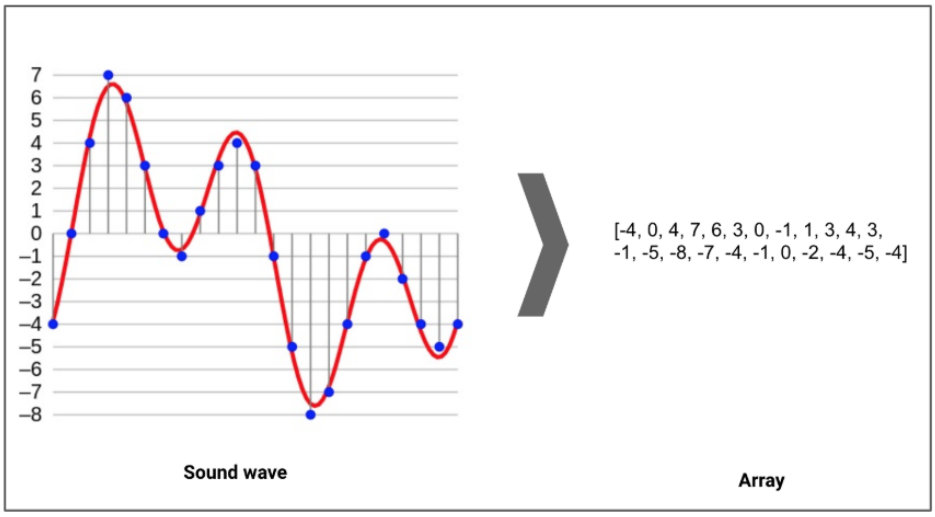
SYSTEM DESIGN

4.1 AUDIO PROPERTIES

We identified the following audio properties for preprocessing to ensure consistency across the whole dataset:

- Audio Channels
- Sample rate
- Bit-depth

Librosa library will be useful for the pre-processing and feature selection.



A sound wave, in red, represented digitally, in blue (after sampling and 4-bit quantisation), with the resulting array shown on the right. Original © Aquegg | Wikimedia Commons

Fig4.1: Sound wave to numpy array display

4.2 PROPOSED APPROACH

The MFCC (Mel-Frequency Cepstral Co-efficients) 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 representation will allow us to identify features for classification.

For extracting a MFCC, we will use Librosa's mfcc() function which generates an MFCC from time series audio data.

Feature Extraction and Audio Properties:

Feature extraction. is used in determining a subset of the initial features is called feature selection. The selected features are expected to contain the relevant information

from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data.

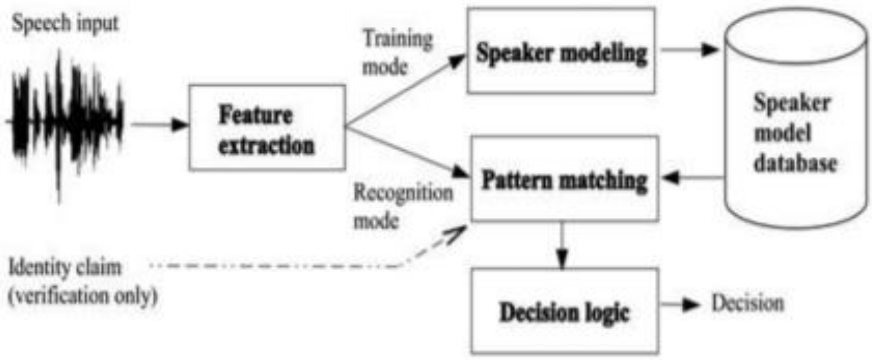


Fig4.2: Feature Extraction

Feature extraction is used to find a set of properties that are stable and acoustically correlated to each other. So it is a type of parameterization of speech signal. Such parameters can form the observation vectors. The goal of feature extractor is to identify relevant information for purpose of accurate classification.

EXISTING FEATURE EXTRACTION TECHNIQUE:

A. Linear Predictive Coding (LPC):

Linear predictive coding (LPC) is a method used mostly in audio signal processing and speech processing for representing the spectral envelope of a digital signal of speech in compressed form, using the information of a linear predictive model. It is one of the most powerful speech analysis techniques, and one of the most useful methods for encoding good quality speech at a low bit rate and provides highly accurate estimates of speech parameters.

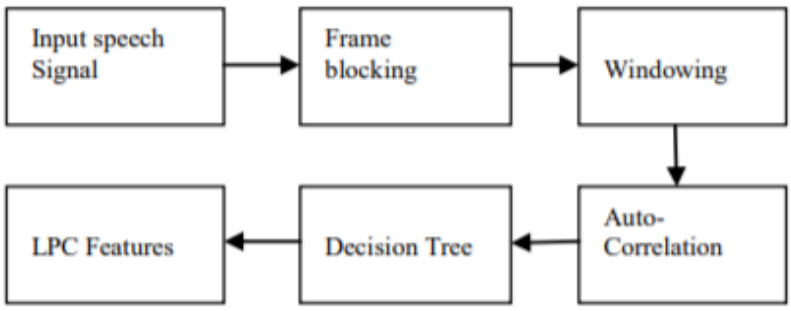


Fig. 3. Steps to extract LPC Features

Fig4.3: Steps to extract LPC Features

B. Perceptual Linear Prediction (PLP)

Perceptual Linear prediction (PLP) demonstrated an improvement over the LPC due to its three principal characteristics derived from the psycho-acoustic properties of human hearing like spectral resolution of critical band, equal loudness curve and intensity loudness power law.

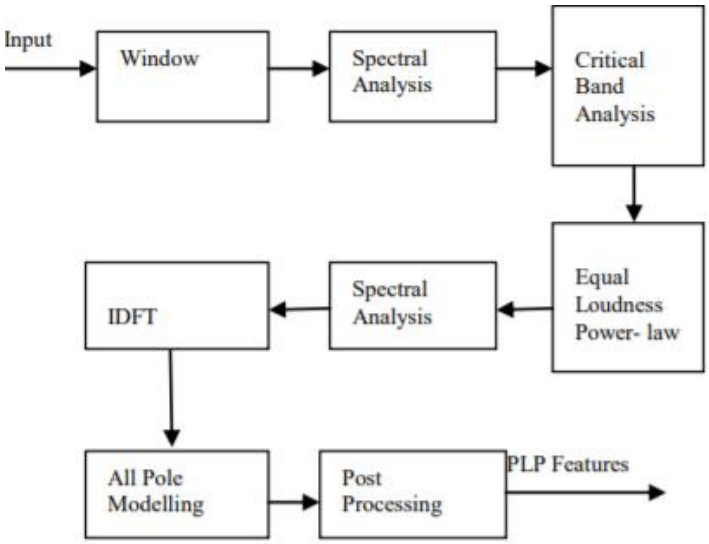


Fig. 5. Block diagram of PLP feature extraction approach

Fig4.4: Block diagram of PLP feature extraction approach

PROPOSED FEATURE EXTRACTION TECHNIQUE:

MFCC(Mel-Frequency Cepstral Co-efficients):

The mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC.[1] They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum.

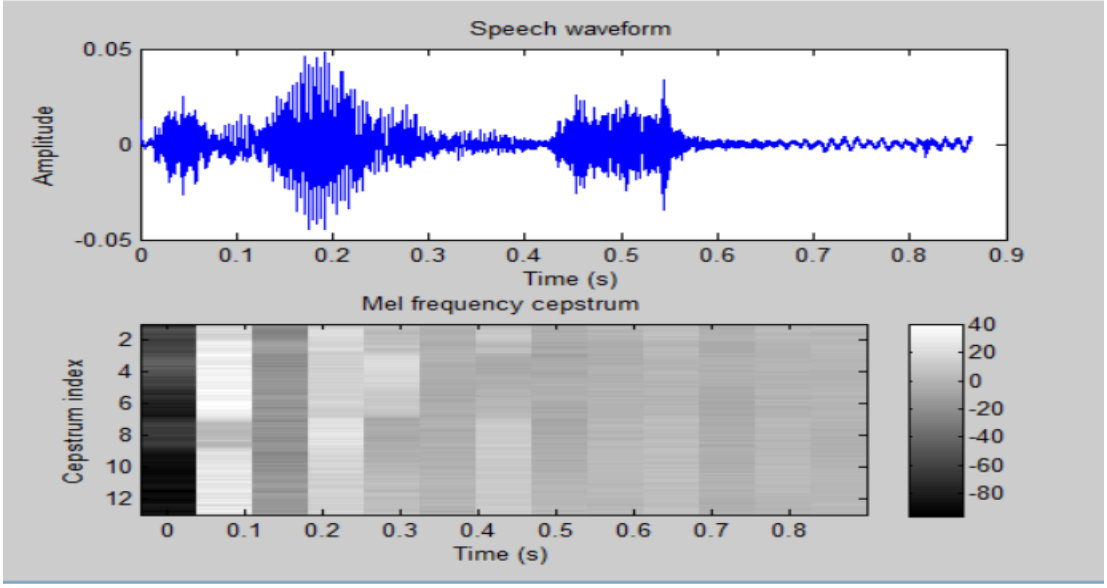


Fig4.5: Speech waveform and Mel Frequency cepstrum

DATA COLLECTION AND EXPLORATION:

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.

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).

2.1.3 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.

2.1.4 Auditory inspection

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

2.1.5 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.

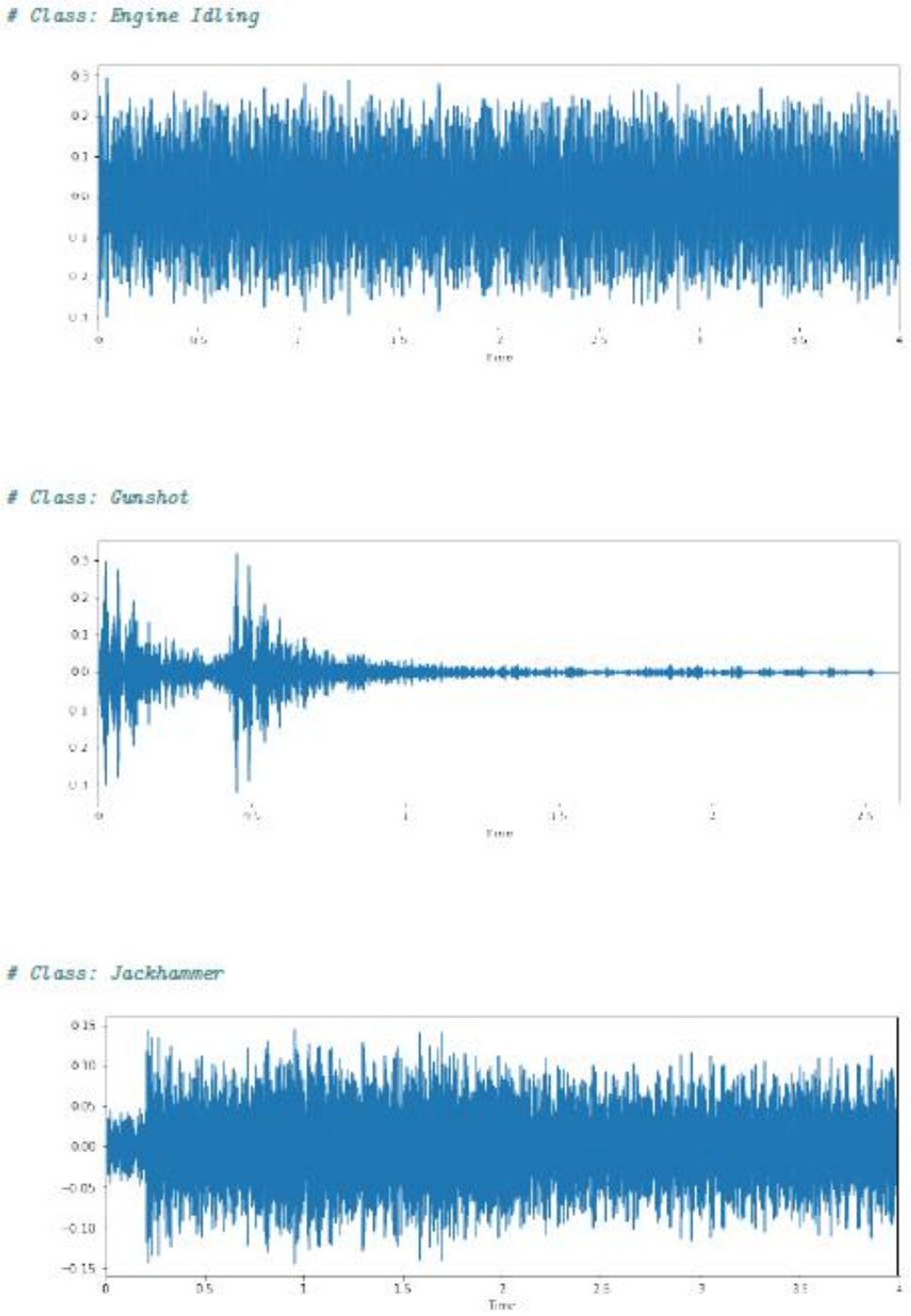


Fig4.6: Some sample waveforms of different sounds

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 dogbark). 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.

Initial Model Architecture-MLP

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to refer to *any* feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training.[2][3] Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

In the Multilayer perceptron, there can more than one linear layer (combinations of neurons). If we take the simple example the three-layer network, first layer will be the *input layer* and last will be *output layer* and middle layer will be called *hidden layer*. We feed our input data into the input layer and take the output from the output layer. We can increase the number of the hidden layer as much as we want, to make the model more complex according to our task.

Feed Forward Network, is the most typical neural network model. Its goal is to approximate some function f (). Given, for example, a classifier $y = f*(x)$ that maps an input x to an output class y, the MLP find the best approximation to that classifier by defining a mapping, $y = f(x; \theta)$ and learning the best parameters θ for it. The MLP networks are

composed of many functions that are chained together. A network with three functions or layers would form f(x) = f (3)(f (2)(f (1)(x))). Each of these layers is composed of units that perform an affine transformation of a linear sum of inputs. Each layer is represented as y = f(WxT + b). Where f is the activation function (covered below), W is the set of parameter, or weights, in the layer, x is the input vector, which can also be the output of the previous layer, and b is the bias vector. The layers of an MLP consists of several fully connected layers because each unit in a layer is connected to all the units in the previous layer. In a fully connected layer, the parameters of each unit are independent of the rest of the units in the layer, that means each unit possess a unique set of weights.

In a supervised classification system, each input vector is associated with a label, or ground truth, defining its class or class label is given with the data. The output of the network gives a class score, or prediction, for each input. To measure the performance of the classifier, the loss function is defined. The loss will be high if the predicted class does not correspond to the true class, it will be low otherwise. Sometimes the problem of overfitting and underfitting occurs at the time of training the model. In this case, Our model performs very well on training data but not on testing data. In order to train the network, an optimization procedure is required for this we need loss function and an optimizer. This procedure will find the values for the set of weights, W that minimizes the loss function.

Features of MLP:

Activation function

If a multilayer perceptron has a linear activation function in all neurons, that is, a linear function that maps the weighted inputs to the output of each neuron, then linear algebra shows that any number of layers can be reduced to a two-layer input-output model. In MLPs some neurons use a *nonlinear* activation function that was developed to model the frequency of action potentials, or firing, of biological neurons.

The two historically common activation functions are both sigmoids, and are described by

$$y(v_i) = \tanh(v_i) \text{ and } y(v_i) = (1 + e^{-v_i})^{-1}.$$

Activation function

In recent developments of deep learning the rectifier linear unit (ReLU) is more frequently used as one of the possible ways to overcome the numerical problems related to the sigmoids.

The first is a hyperbolic tangent that ranges from -1 to 1, while the other is the logistic function, which is similar in shape but ranges from 0 to 1. Alternative activation functions have been proposed, including the rectifier and softplus functions. More specialized activation functions include radial basis functions (used in radial basis networks, another class of supervised neural network models).

Layers

The MLP consists of three or more layers (an input and an output layer with one or more *hidden layers*) of nonlinearly-activating nodes. Since MLPs are fully connected, each node in one layer connects with a certain weight to every node in the following layer.

Learning

Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron.

We can represent the degree of error in an output node in the th data point (training example) by , where is the target value and is the value produced by the perceptron. The node weights can then be adjusted based on corrections that minimize the error in the entire output, given by

$$\mathcal{E}(n) = \frac{1}{2} \sum_j e_j^2(n).$$

Degree of error

Using gradient descent, the change in each weight is

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

Change in each weight

where is the output of the previous neuron and is the *learning rate*, which is \selected to ensure that the weights quickly converge to a response, without oscillations.

Working - MLP

We will start with constructing a MultiLayer Perceptron using Keras. Starting with a sequential model so new can build the model layer by layer.

We will begin with a simple architecture , consisting of three layers, a input layer,a hidden layer,and an output layer. The first layer will receive the input shape . Each input sample contains 40 MFCC'S which means we start with a shape of 1X 40. The first two layers will have 256 nodes.The activation function used is Rectified Linear Activation. The ReLU is chosed as it is proven for good record with working on neural networks.The output layer will have the count of diffenterne classes used in the training data.The activation used for output layer is Softmax as it reduces the values to probabalitites.

The class is derived or classified with the probability values, , i.e, The higher the probability the more the likeliness of being that class.

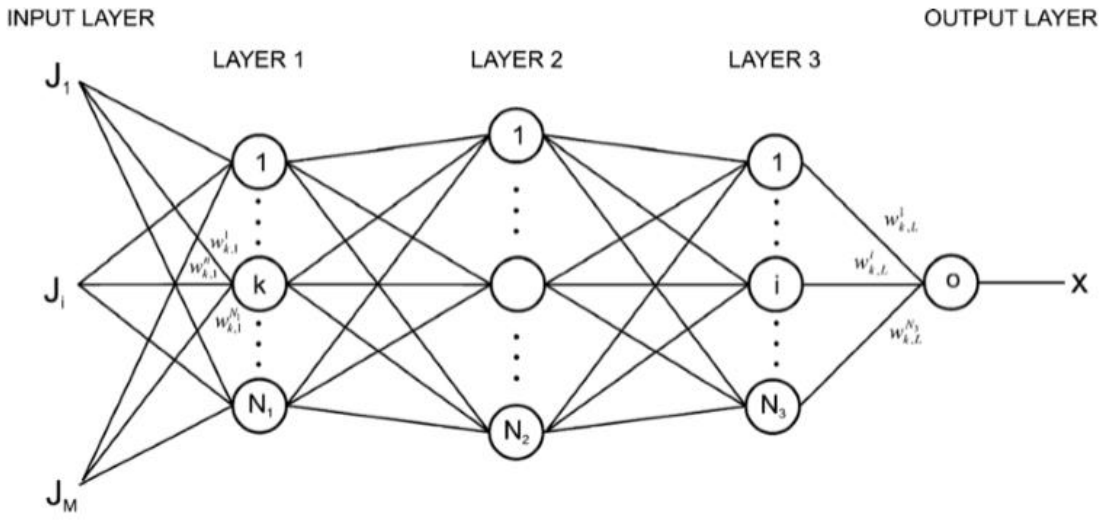


Fig4.7: Layers of Multi Layer Perceptron (MLP)

Final Model Architecture - CNN:

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation.

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

When programming a CNN, the input is a tensor with shape (number of images) x (image width) x (image height) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map width) x (feature map height) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

- Convolutional kernels defined by a width and height (hyper-parameters).
- The number of input channels and output channels (hyper-parameter).
- The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus.[11] Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for *each* neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters.[12] For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

Pooling

Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. *Max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. *Average pooling* uses the average value from each of a cluster of neurons at the prior layer.

Fully connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Receptive field

In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from *every* element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its *receptive field*. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.

Weights

Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

The vector of weights and the bias are called *filters* and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.

Convolutional layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-

dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.[nb 1]

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Local connectivity

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume.

The extent of this connectivity is a hyper parameter called the receptive field of the neuron. The connections are local in space (along width and height), but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

Pooling layer

Fig4.8 : Pooling Layer

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which *max pooling* is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks. The pooling layer serves to progressively reduce the spatial size of the representation, to

reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control over fitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture.[*citation needed*] The pooling operation provides another form of translation invariance.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 down samples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations:

In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.

In addition to max pooling, pooling units can use other functions, such as average pooling or ℓ_2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to max pooling, which performs better in practice.[56]

Due to the aggressive reduction in the size of the representation,[*which?*] there is a recent trend towards using smaller filters or discarding pooling layers altogether.[58]

Fig4.9: RoI Pooling

"Region of Interest" pooling (also known as RoI pooling) is a variant of max pooling, in which output size is fixed and input rectangle is a parameter.[59]

Pooling is an important component of convolutional neural networks for object detection based on Fast R-CNN[60] architecture.

ReLU layer

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function .[51] It effectively removes negative values from an activation map by setting them to zero.[61] It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent , and the sigmoid function . ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

Fully connected layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

Loss layer

The "loss layer" specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions appropriate for different tasks may be used.

Working - CNN

We will modify our model to be a Convolutional Neural Network (CNN) again using Keras and a Tensorflow backend. 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. 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 which serves to shorten the training time and reduce overfitting. The output layer will have 10 nodes . 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.

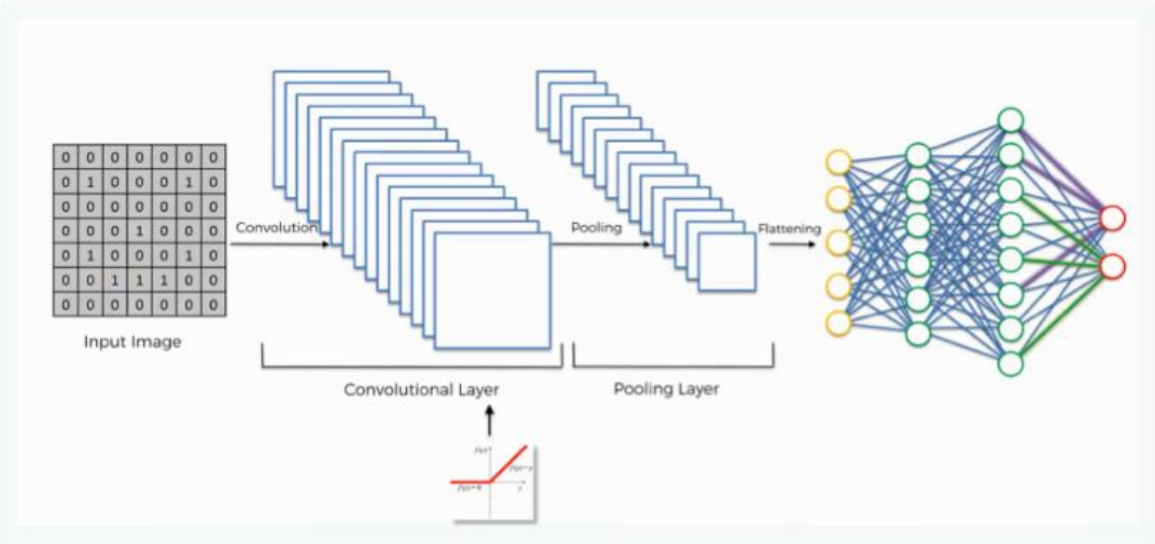


Fig4.10: Pooling Layers and Convolutional Layers

CHAPTER 5

SYSTEM IMPLEMENTATION

Data Exploration and Visualisation

```
import IPython.display as ipd
ipd.Audio('../UrbanSound Dataset sample/audio/100032-3-0-0.wav')

import IPython.display as ipd
import librosa
import librosa.display
import matplotlib.pyplot as plt

# visualizing  audio sample

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav'
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Audio sample file properties

```
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/audio/'),'fold'+str(row["fold"])+'/',str(row["slice_file_name"]))
    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'])
```

Data Preprocessing and Data Splitting

```
#Sample  rate conversion
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)
# Extracting features

mfccs = librosa.feature.mfcc(y=librosa_audio, sr=librosa_sample_rate, n_mfcc=40)
print(mfccs.shape)

import librosa.display
librosa.display.specshow(mfccs, sr=librosa_sample_rate, x_axis='time')
def extract_features(file_name):

    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
        mfccsscaled = np.mean(mfccs.T,axis=0)
    except Exception as e:
        print("Error encountered while parsing file: ", file)
```

```
        return None

    return mfccsscaled
# 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(row["slice_file_na
me"]))

    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')



# Converting data to labels and splitting the data
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

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

le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))
```

```
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)
```

Initial model architecture - MLP

```
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'))

#Model Compilation
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

 model.summary()
```

```
score = model.evaluate(x_test, y_test, verbose=0)
accuracy = 100*score[1]

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

from keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 100
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.basic_mlp.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_test, y_test), callbacks=[checkpointer], verbose=1)



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

#Model Evaluation
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])
```

Final model architecture – CNN

```
#Model Compilation
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), activation='relu'))
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'))

model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

model.summary()

score = model.evaluate(x_test, y_test, verbose=1)
accuracy = 100*score[1]

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



#Model Training

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_test, y_test), callbacks=[checkpointer], verbose=1)

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

#Model Evaluation
```

```
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])
```

Audio prediction

```
#Feature Extraction
import numpy as np
max_pad_len = 174

def extract_features(file_name):

    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
        pad_width = max_pad_len - mfccs.shape[1]
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')

    except Exception as e:
        print("Error encountered while parsing file: ", file_name)
        return None

    return mfccs


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(row["slice_file_na
me"]))

    class_label = row["class_name"]
    data = extract_features(file_name)

    features.append([data, class_label])

featuresdf = pd.DataFrame(features, columns=['feature','class_label'])

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

# Prediction
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_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') )



filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav'
print_prediction(filename)
```

CHAPTER 6

SAMPLE OUTPUT

```
In [19]: # num of channels

         print(audiodf.num_channels.value_counts(normalize=True))
```

```
2    0.915369
1    0.084631
Name: num_channels, dtype: float64
```

Fig6.1: Output of number of channels

```
In [21]: # sample rates

         print(audiodf.sample_rate.value_counts(normalize=True))
```

```
44100     0.614979
48000     0.286532
96000     0.069858
24000     0.009391
16000     0.005153
22050     0.005039
11025     0.004466
192000    0.001947
8000      0.001374
11024     0.000802
32000     0.000458
Name: sample_rate, dtype: float64
```

Fig6.2: Output of sample rates

```
In [22]: # bit depth

         print(audiodf.bit_depth.value_counts(normalize=True))
```

```
16    0.659414
24    0.315277
32    0.019354
8     0.004924
4     0.001031
Name: bit_depth, dtype: float64
```

Fig6.3: Output of bit depth 1

```
In [22]: # bit depth

         print(audiodf.bit_depth.value_counts(normalize=True))
```

```
16    0.659414
24    0.315277
32    0.019354
8     0.004924
4     0.001031
Name: bit_depth, dtype: float64
```

Fig6.4: Output of bit depth 2

```
Epoch 00096: val_loss did not improve from 0.42049
Epoch 97/100
6985/6985 [==============================] - 2s 334us/step - loss: 0.5525 - acc: 0.8125 - val_loss: 0.4370 - val_acc: 0.8626

Epoch 00097: val_loss did not improve from 0.42049
Epoch 98/100
6985/6985 [==============================] - 2s 329us/step - loss: 0.5246 - acc: 0.8241 - val_loss: 0.4338 - val_acc: 0.8620

Epoch 00098: val_loss did not improve from 0.42049
Epoch 99/100
6985/6985 [==============================] - 2s 347us/step - loss: 0.5346 - acc: 0.8169 - val_loss: 0.4457 - val_acc: 0.8586

Epoch 00099: val_loss did not improve from 0.42049
Epoch 100/100
6985/6985 [==============================] - 2s 351us/step - loss: 0.5413 - acc: 0.8153 - val_loss: 0.4306 - val_acc: 0.8764

Epoch 00100: val_loss did not improve from 0.42049
Training completed in time:  0:04:15.582298
```

In [6]:
```
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
```

Fig6.5: Epochs and accuracy of model in MLP

In [9]:
```
# Class: Air Conditioner

filename = '../UrbanSound Dataset sample/audio/100852-0-0-0.wav'
print_prediction(filename)
```

```
The predicted class is: air_conditioner

air_conditioner                 :  0.9998942613601684570312500000000
car_horn                :  0.000007154393188102403655648231 51
children_playing                :  0.000017915303033078089356422424 32
dog_bark                :  0.000000256554130828590132296085 36
drilling                :  0.000002839924263753346167504787 45
engine_idling           :  0.000058878980780718848109245300 29
gun_shot                :  0.000000016207824415914728888310 49
jackhammer              :  0.000000356629641373729100450873 37
siren           :  0.000000043484725154030456906184 55
street_music            :  0.000018307808204554021358489990 23
```

In [10]:
```
# Class: Drilling

filename = '../UrbanSound Dataset sample/audio/103199-4-0-0.wav'
print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                 :  0.0000000000329535600196440014 9973
car_horn                :  0.000000003089592581773104029707 61
children_playing                :  0.000022088306650402955710887908 94
dog_bark                :  0.000000674014813739631790667772 29
drilling                :  0.999725043773651123046875000000 00
engine_idling           :  0.000000000023124249043382505419 69
gun_shot                :  0.000000143469492286385502666234 97
jackhammer              :  0.000000000297803892657100277574 50
siren           :  0.000000001568933982731834930746 12
street_music            :  0.000252099474892020225524902343 75
```

Fig6.6: Predictions of some audios in MLP (1)

```
In [15]: filename = '../Evaluation audio/gun_shot_1.wav'

         print_prediction(filename)
```

```
The predicted class is: dog_bark

air_conditioner                 :  0.020088111981749534606933593750000
car_horn                :  0.000474296481115743517875671386 72
children_playing                :  0.000949423410929739475250244140 62
dog_bark                :  0.536540150642395019531250000000000
drilling                :  0.000931742019020020961761474609 38
engine_idling           :  0.031237760558724403381346756250 00
gun_shot                :  0.000912152521777898073196411132 81
jackhammer              :  0.000020154206140432506799697875 98
siren           :  0.000559707754291594028472900390 62
street_music            :  0.408286452293395996093750000000000
```

```
In [16]: filename = '../Evaluation audio/siren_1.wav'

         print_prediction(filename)
```

```
The predicted class is: siren

air_conditioner                 :  0.000007324023499677423387765884 40
car_horn                :  0.000570923730265349149703979492 19
children_playing                :  0.001990682445466518402099609375 00
dog_bark                :  0.020904883742332458496093750000000
drilling                :  0.000465523567982017993927001953 12
engine_idling           :  0.141645804047584533691406250000000
gun_shot                :  0.000501968432217836380004882812 50
jackhammer              :  0.002760533010587096214294433593 75
siren           :  0.815271973609924316406250000000000
street_music            :  0.015880407765507698059082031250000
```

Fig6.7: Predictions of some audios in MLP (2)

```
Epoch 69/72
6985/6985 [==============================] - 266s 38ms/step - loss: 0.1172 - acc: 0.9599 - val_loss: 0.3116 - val_acc: 0.920
4

Epoch 00069: val_loss did not improve from 0.27239
Epoch 70/72
6985/6985 [==============================] - 265s 38ms/step - loss: 0.1213 - acc: 0.9568 - val_loss: 0.2978 - val_acc: 0.916
4

Epoch 00070: val_loss did not improve from 0.27239
Epoch 71/72
6985/6985 [==============================] - 14289s 2s/step - loss: 0.1203 - acc: 0.9581 - val_loss: 0.2878 - val_acc: 0.916
4

Epoch 00071: val_loss did not improve from 0.27239
Epoch 72/72
6985/6985 [==============================] - 92s 13ms/step - loss: 0.1147 - acc: 0.9596 - val_loss: 0.3005 - val_acc: 0.9193

Epoch 00072: val_loss did not improve from 0.27239
Training completed in time:  8:57:38.203486
```

```
In [61]: 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
```

Fig6.8: Epochs and accuracy of model in CNN

In [53]:
```
# Class: Street music

filename = '../UrbanSound Dataset sample/audio/101848-9-0-0.wav'
print_prediction(filename)
```

```
The predicted class is: street_music

air_conditioner                 :  0.000114960152131970971822738647 46
car_horn                :  0.000792882812675088644027709960 94
children_playing                :  0.017915384843945503234863281250 00
dog_bark                :  0.002579237101599574089050292968 75
drilling                :  0.000079045399616006761789321899 41
engine_idling           :  0.000060611935623455792665481567 38
gun_shot                :  0.000000000074822682771813475710 59
jackhammer              :  0.000004578259904519654810428619 38
siren           :  0.009223079308867454528808593750 00
street_music            :  0.969230234622955322656250000000 00
```

In [64]:
```
# Class: Car Horn

filename = '../UrbanSound Dataset sample/audio/100648-1-0-0.wav'
print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                 :  0.000598666374571621417999269578 12
car_horn                :  0.263911932706832885742187500000 00
children_playing                :  0.001260121352970600128173828125 00
dog_bark                :  0.278439521789550781250000000000 00
drilling                :  0.348172336816787719726562500000 00
engine_idling           :  0.003390490543097257614135742187 50
gun_shot                :  0.051762931048870086669921875000 00
jackhammer              :  0.038593176752328872680664062500 00
siren           :  0.012712064199149608612060546875 00
street_music            :  0.001158745610155165195465087890 62
```

Fig6.9: Predictions of some audios in CNN (1)

In [66]:
```
filename = '../Evaluation audio/drilling_1.wav'

print_prediction(filename)
```

```
The predicted class is: jackhammer

air_conditioner                 :  0.078613154590129852294921875000 00
car_horn                :  0.000000123948524333172827027738 09
children_playing                :  0.000008794507266429718583822250 37
dog_bark                :  0.000001840709501266246661543846 13
drilling                :  0.000033784923289204016327857971 19
engine_idling           :  0.063723281025886535644531250000 00
gun_shot                :  0.000000117360393403487250907346 61
jackhammer              :  0.857615232467651367187500000000 00
siren           :  0.000003615087280195439234375953 67
street_music            :  0.000000134870006718301738146692 51
```

In [65]:
```
filename = '../Evaluation audio/gun_shot_1.wav'

print_prediction(filename)
```

```
The predicted class is: gun_shot

air_conditioner                 :  0.000000017110387773300317348912 36
car_horn                :  0.000000028287308495578145084436 98
children_playing                :  0.000011538927537912968546152114 87
dog_bark                :  0.000067637512984219938516616821 29
drilling                :  0.000022255826479522511363029479 98
engine_idling           :  0.000003852147983707254752516746 52
gun_shot                :  0.999888539314270019531250000000 00
jackhammer              :  0.000000000601333427496797412459 27
siren           :  0.000006033371391822583973407745 36
street_music            :  0.000000020419792079451326571870 60
```

Fig6.10: Predictions of some audios in CNN (2)

CHAPTER 8

CONCLUSION

The audio recognition methodology used here would produce higher degree of accuracy than in the previous models. The use of MFCC over traditional spectrograms paves way to a lot of new features to be described on the audio samples. These new features provides a lot of clarity to the models to be used. This project when fed with a noise less audio sample provides almost near perfect classifications. The project when trained with a Multi Layer Perceptron- MLP provides an accuracy of ~92% on the training data and ~ 87 % on the test data. The model is further enhanced when using a convolutional neural network. i.e, The training accuracy in CNN is ~98% which is huge increase over the MLP accuracy and also significant rise of 5% is shown in the test data ~92%. For Further fine tuning and for the usage in real world applications the audio samples must at first be cleared of noises, the noises can be cleared using pre-existing machine learning algorithms.

REFERENCES

[1] Justin Salamon, Christopher Jacoby and Juan Pablo Bello. Urban Sound Datasets. https://urbansounddataset.weebly.com/urbansound8k.html

[2] Mel-frequency spectrum Wikipedia page.

[3] Justin Salamon, Christopher Jacoby and Juan Pablo Bello. A dataset and taxonomy of urban sound search. http://www.justinsalamon.com.uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf

[4] Manik Soni AI which classifies sounds: code: Python. https://hackernoon,com/ai-which-classifies-sounds-code-python

[5] Urban Sound Classification – Part 2: Sample rate conversion, Librosa. https://towardsdatascience.com

[6] Audio Source: http://soundbible.com

[7] Speech Recognition Using Deep Neural Networks: A Systematic Review https://ieeexplore.ieee.org/document/8632885/footnotes#footnotes