VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI - 590018

"Spoken Language Recognition with deep learning"

Thesis submitted in partial fulfillment of the curriculum prescribed for the award of the degree of Bachelor of Engineering in Computer Science & Engineering by

1CR14CS021 Aparna Vinod

Under the Guidance of

Mrs. Priya L
Assistant Professor
Department of CSE, CMRIT, Bengaluru

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
#132, AECS LAYOUT, IT PARK ROAD, BENGALURU - 560037

2017-18

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI - 590018

CERTIFICATE

This is to certify that the project entitled **"Spoken Language Recignition with Deep Learning"** is a bonafide work carried out by **Aparna Vinod** in partial fulfillment of the award of the degree of Bachelor of Engineering in Computer Science & Engineering of Visvesvaraya Technological University, Belgaum, during the year 2017-18. It is certified that all corrections / suggestions indicated during reviews have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

-------------------------	-------------------------	-------------------------
Signature of Guide	Signature of HoD	Signature of Principal
Mrs. Priya L	**Dr. Jhansi Rani P**	**Dr. Sanjay Jain**
Assistant Professor	Professor & Head	Principal
Department of CSE	Department of CSE	CMRIT,
CMRIT, Bengaluru - 37	CMRIT, Bengaluru - 37	Bengaluru - 37

External Viva

	Name of the Examiners	Institution	Signature with Date
1.	-------------------------	----------------------------	----------------------------
2.	-------------------------	----------------------------	----------------------------

ACKNOWLEDGEMENT

I take this opportunity to thank all of those who have generously helped me to give a proper shape to my work and complete my BE project successfully. A successful project is fruitful culmination efforts by many people, some directly involved and some others indirectly, by providing support and encouragement.

I would like to thank **Dr. SANJAY JAIN** , Principal , CMRIT , for providing excellent academic environment in the college.

I would like to express our gratitude towards **Dr. JHANSI RANI** , Professor & HOD , Dept of CSE , CMRIT , who provided guidance and gave valuable suggestions regarding the project.

I consider it a privilege and honour to express my sincere gratitude to my Internal Guide **MRS. PRIYA .L** , Asst. Professor , Department of Computer Science & Engineering , CMRIT , for her valuable guidance throughout the tenure of this project work.

Aparna Vinod

Table of Contents

Table of Contents **ii**

List of Figures **iv**

List of Tables **v**

Abstract **vi**

1 Introduction **1**

1.1 Preamble 2

1.2 Purpose of the project 2

1.3 Scope of the project 3

1.4 Definitions, acronyms and abbreviations 3

2 Literature Survey **4**

2.1 Tensorflow 5

2.2 PyCharm 5

3 Theoretical Background **6**

3.1 Deep Learning 7

3.2 Multilayer Perceptron 9

3.3 Convolutional Neural Network 10

3.4 Mel Spectral Frequency Cepstral Coefficient 12

3.5 Tensorflow 13

4 Software requirements specifications **14**

4.1 Operating Environment 15

4.2 Functional requirements 15

4.3 Non-functional Requirements 16

4.4 User characteristics 16

4.5 Applications of the system 16

5 System Design **18**

5.1 System Development Methodology . . . 19
5.2 Design using UML . . . 20
5.3 Data Flow Diagram . . . 21
5.4 Class Diagram . . . 22
5.5 Use Case Diagram . . . 22
5.6 Activity Diagram . . . 22
5.7 Sequence Diagram . . . 23

6 Implementation **27**
6.1 Introduction . . . 28
6.2 Training Code . . . 28
6.3 Testing Code . . . 32
6.4 Feature Extraction Code . . . 36
6.5 Process . . . 38
6.6 Important Libraries used . . . 38

7 Testing and Results **42**
7.1 Introduction . . . 43
7.2 Testing Methodologies . . . 43
7.3 Running and Testing of codes . . . 45

8 Conclusion **49**
8.1 Future Scope . . . 50
8.2 Building a complete system . . . 50

References **51**

List of Figures

3.1 Multilayer Perceptron . . . 9
3.2 Mel Frequency Cepstral Coefficient . . . 12

5.1 Waterfall Model . . . 21
5.2 Data Flow Diagram . . . 22
5.3 Class Diagram . . . 23
5.4 Use Case Diagram . . . 24
5.5 Activity Diagram . . . 25
5.6 Sequence Diagram . . . 26

List of Tables

7.1 Unit Test Cases 45

Abstract

This study proposes a Deep Neural Network based approach towards identification of a language that is spoken by a user. The languages that are being processed are English and Hindi.

Often in certain Networking Companies, it is observed that language barrier problems arise. This can be overcome by the use of identification of the spoken language, following which correspondence will occur with the appropriate person.

Various concepts of deep learning and neural networking will be used along with the main concept of Mel Frequency Cepstral Coefficients (MFCC). In sound processing, 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.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design, where they have produced results comparable to and in some cases superior to human experts.

Chapter 1

Introduction

1.1 Preamble

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design, where they have produced results comparable to and in some cases superior to human experts.

Deep learning models are vaguely inspired by information processing and communication patterns in biological nervous systems yet have various differences from the structural and functional properties of biological brains, which make them incompatible with neuroscience evidences. In our case of spoken language recognition, the main aim is to identify a language that is being spoken by a user.

Various concepts of deep learning and neural networking will be used along with the main concept of Mel Frequency Cepstral Coefficients (MFCC). In sound processing, 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. 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.

1.2 Purpose of the project

The purpose of this project is to design and implement a software system which will automate the task of identifying a language that is being spoken by a user.

In telecommunication service provider companies, when a call is made by the customer, an automatically generated voice asks the user to enter the number according the language they want to communicate in.

This process can be avoided if, when the user speaks in any language, the call will automatically get transferred to the service provider employee that is aware of that language.

In this project, only the back-end procedure is created in order to identify two different languages, Hindi and English. Thus, this report encapsulates a brief overview of the product, the requirements necessary for its development, the design features and considerations, the process of implementing it as well as its testing details.

This report has been prepared to Outline the main features of the software system introduced and the details of the development procedure followed. Act as a basis for any future enhancement that might be implemented with respect to this project.

1.3 Scope of the project

In telecommunication service provider companies, when a call is made by the customer, an automatically generated voice asks the user to enter the number according the language they want to communicate in.

This process can be avoided if, when the user speaks in any language, the call will automatically get transferred to the service provider employee that is aware of that language.

There a considerate amount of time waste and energy that could be avoided by the use of this process.

1.4 Definitions, acronyms and abbreviations

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.

MFCC: Mel Frequency Cepstral Coefficients are coefficients that collectively make up an MFC.

WAV: Waveform Audio File Format is a Microsoft and IBM audio file format standard for storing an audio bitstream on PCs.

CSV: Comma Separated Values (excel sheet format)

Chapter 2

Literature Survey

Literature survey is the documentation of a comprehensive review of the published and unpublished work from secondary sources data in the areas of specifc interest to the researcher. The library is a rich storage base for secondary data and researchers used to spend several weeks and sometimes months going through books, journals, newspapers, magazines, conference proceedings, doctoral dissertations, master's theses, government publications and financial reports to understand information on their research topic. Reviewing the literature on the topic area at this time helps the researcher to focus further interviews more meaningfully on certain aspects found to be important is the published studies even if these had not surfaced during the earlier questioning .So the literature survey is important for gathering the secondary data for the research which might be proved very helpful in the research. The literature survey can be conducted for several reasons. The literature review can be in any area of the business.

2.1 Tensorflow

2.1.1 Determine which tensorflow to install

- TensorFlow with CPU support only. If the system does not have a NVIDIA GPU, install this version. Note that this version of TensorFlow is typically much easier to install (typically, in 5 or 10 minutes), so even if you have an NVIDIA GPU, installing this version first is recommended. Prebuilt binaries will use AVX instructions.
- TensorFlow with GPU support. TensorFlow programs typically run significantly faster on a GPU than on a CPU. Therefore, if your system has a NVIDIA GPU meeting the prerequisites shown below and you need to run performance-critical applications, you should ultimately install this version.

2.2 PyCharm

- Install Anaconda (Python) on your operating system. You can either download anaconda from the official site and install on your own or you can follow these anaconda installation tutorials below.
- Download the community edition of Pycharm for your operating system.

Chapter 3

Theoretical Background

3.1 Deep Learning

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised. Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design,] where they have produced results comparable to and in some cases superior to human experts. Deep learning models are vaguely inspired by information processing and communication patterns in biological nervous systems yet have various differences from the structural and functional properties of biological brains, which make them incompatible with neuroscience evidences.

3.1.1 Definition

Deep learning is a class of machine learning algorithms that:[10](pp199200)

- use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

3.1.2 Types

3.1.2.1 Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.[1] It infers a function from labeled training data consisting of a set of training examples.[2] In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (see inductive bias). The parallel task in human and animal psychology is often referred to as concept learning. In order to solve a given problem of supervised learning, one has to perform the following steps:

- Determine the type of training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
- Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
- Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.
- Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or decision trees.
- Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation.
- Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

3.1.2.2 Unsupervised Learning

Unsupervised machine learning is the machine learning task of inferring a function that describes the structure of "unlabeled" data (i.e. data that has not been classified or categorized). Since the examples given to the learning algorithm are unlabeled, there is no straightforward way to evaluate the accuracy of the structure that is produced by the algorithmone feature that distinguishes unsupervised learning from supervised learning and reinforcement learning. A central application of unsupervised learning is in the field of density estimation in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining various key features of data.

The classical example of unsupervised learning in the study of both natural and artificial neural networks is subsumed by Donald Hebb's principle, that is, neurons that fire together wire together. In Hebbian learning, the connection is reinforced irrespective of an error, but is exclusively a function of the coincidence between action potentials between the two neurons. A similar version that modifies synaptic weights takes into account the time between the action potentials (spike-timing-dependent plasticity or STDP). Hebbian Learning has been hypothesized to underlie a range of cognitive functions, such as pattern recognition and experiential learning.

3.2 Multilayer Perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes. 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. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer. **Acti-**

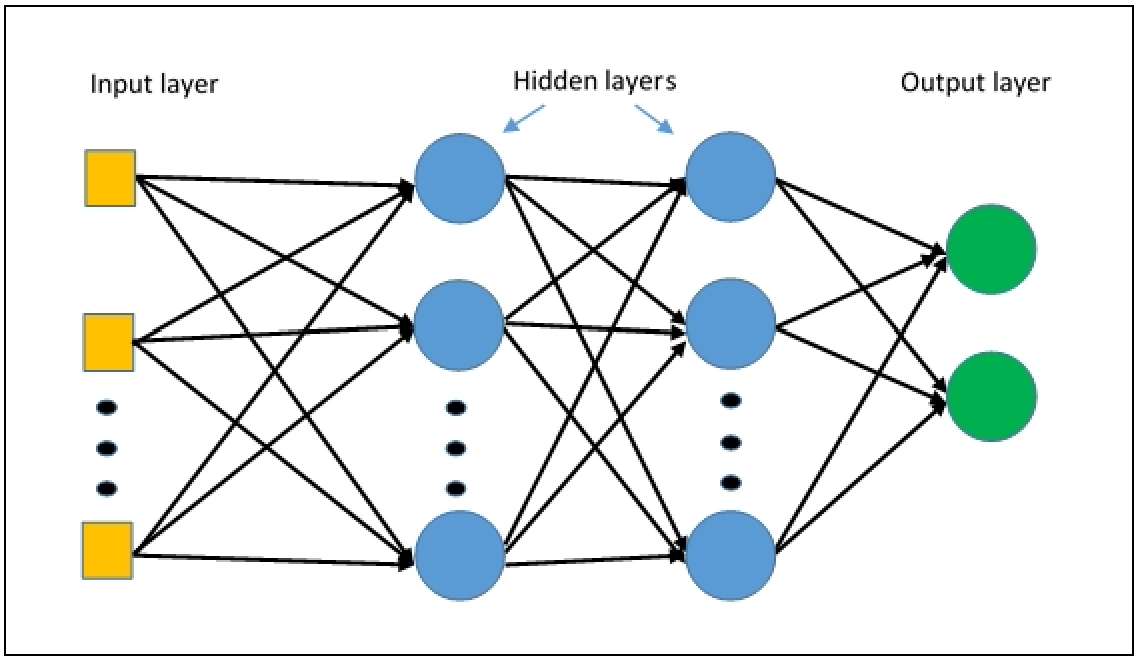


Figure 3.1: Multilayer Perceptron

vation 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 common activation functions are both sigmoids, and are described by y(vi) = tanh(vi) and y(vi)=(1+e to the power vi)power -1 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. Here yi is the output of the ith node (neuron) and vi is the weighted sum of the input connections. 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 making it a deep neural network. Since MLPs are fully connected, each node in one layer connects with a certain weight wij 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.

Modes

- Training Mode
  Data is provided to the network and the neurons goes through it, learning and classifying automatically.
- Testing Mode In this mode, an input is provided on which the neuron has been trained on. It will detect the input and provide the output.

3.3 Convolutional Neural Network

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery.

CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems and natural language processing.

Design

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers[citation needed]. Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

Convolutional Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. 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 10000 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. 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, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of

neurons at the prior layer. Another example is average pooling, which 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).
Weights CNNs share weights in convolutional layers, which means that the same filter (weights bank) is used for each receptive field in the layer; this reduces memory footprint and improves performance.

3.4 Mel Spectral Frequency Cepstral Coefficient

In order to identify which language is being spoken, the system must be able to differentiate between them. This can be done using MFCC (Mel Frequency Cepstral Coefficient) extraction. It is a feature extraction technique. In sound processing,

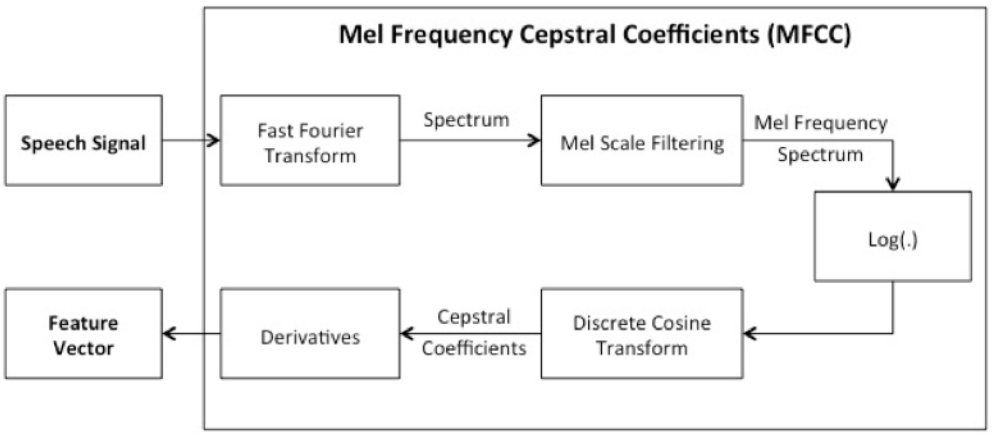


Figure 3.2: Mel Frequency Cepstral Coefficient

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. 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. This frequency warping can allow for better representation of sound, for example, in audio compression.

MFCCs are commonly derived as follows:

- Take the Fourier transform of (a windowed excerpt of) a signal.
- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
- Take the logs of the powers at each of the mel frequencies.
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
- The MFCCs are the amplitudes of the resulting spectrum.

There can be variations on this process, for example: differences in the shape or spacing of the windows used to map the scale, or addition of dynamics features such as ”delta” and ”delta-delta” (first- and second-order frame-to-frame difference) coefficients.
The European Telecommunications Standards Institute in the early 2000s defined a standardised MFCC algorithm to be used in mobile phones.

3.5 Tensorflow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, often replacing its closed-source predecessor, DistBelief.
TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015.

Chapter 4

Software requirements specifications

The Software Requirements Specifications (SRS) that follows describes the behaviour expected from the system to be developed. It includes a set of functional and non-functional requirements and the operating environment (hardware requirements and software requirements) to be used. It also explains the characteristics of the intended user along with valid reasons for building the proposed system by enlisting its applications and advantages. The purpose of this section is to specify the various requirements for the successful running of this project.

4.1 Operating Environment

4.1.1 Hardware requirements

Processor: Intel Pentium 4/ AMD 64 and above
RAM: A size of at least 512 MB
Hard Disk: Space of a least 1 GB

4.1.1.1 Software requirements

Operating System Platform: Windows XP/ Windows 7/ Windows 10
IDE: PyCharm/Jupyter Notebook
Framework: Tensorflow
Language: Python
sub section content goes here

4.2 Functional requirements

They state the services provided by the system, behaviour of the system and reaction of the system to particular inputs. It may also specify what the system should not do.

- Audio is taken as an input.
- Features from audio recordings are extracted.
- Features are stored in an excel sheet and inputted into code in a CSV format.
- The system is trained on the features that have been extracted so that identification of the language is possible.
- Code implements the classification of audio signals into Hindi and English.
- For checking if the system has been trained correctly, a testing code is implemented.

- Last step is validation of the output.

4.3 Non-functional Requirements

They are requirements which impose constraints on the design, implementation (such as performance engineering requirements, quality standards or design constraints) and services or functions offered by the system. They apply to the system as a whole.

- Platform Independence: The software should run on any hardware or software platform (PC, Mac, SunSparc, etc.) or software platform (Linux, Unix, MacOS, Windows etc.)
- Accuracy: The audio recordings must be classified into their true classes. There should be minimum misclassification error.
- Efficiency: Software should not waste system resources such as memory and processor cycles.
- Reliability: Probability of failure-free software operation for a specific period of time in a specified environment should be high.
- Speed: Classification speed must be as fast as neural networking system can attain.
- Friendly user interface

4.4 User characteristics

The users of this application would be:

- People who know only a certain language can use it to translate an unknown language into a known language.
- Natural Language Processing users and developers

4.5 Applications of the system

Applications of deep learning are:

Automatic speech recognition Large-scale automatic speech recognition is the first and most convincing successful case of deep learning. LSTM RNNs can learn "Very Deep Learning" tasks that involve multi-second intervals containing speech events separated by thousands of discrete time steps, where one time step corresponds to about

10 ms. LSTM with forget gates is competitive with traditional speech recognizers on certain tasks.

Image recognition A common evaluation set for image classification is the MNIST database data set. MNIST is composed of handwritten digits and includes 60,000 training examples and 10,000 test examples. As with TIMIT, its small size lets users test multiple configurations.

Visual art processing Closely related to the progress that has been made in image recognition is the increasing application of deep learning techniques to various visual art tasks. DNNs have proven themselves capable, for example, of a) identifying the style period of a given painting, b) "capturing" the style of a given painting and applying it in a visually pleasing manner to an arbitrary photograph, and c) generating striking imagery based on random visual input fields.

Natural language processinG Neural networks have been used for implementing language models since the early 2000s. LSTM helped to improve machine translation and language modeling.

Chapter 5

System Design

Design is a meaningful engineering representation of something that is to be built.

It is the most crucial phase in the developments of a system. Software design is a process through which the requirements are translated into a representation of software. Design is a place where design is fostered in software Engineering. Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. Design is the perfect way to accurately translate a customers requirement in the finished software product. Design creates a representation or model, provides details about software data structure, architecture, interfaces and components that are necessary to implement a system.

The logical system design arrived at as a result of systems analysis is converted into physical system design.

5.1 System Development Methodology

System development method is a process through which a product will get completed or a product gets rid from any problem. Software development process is described as a number of phases, procedures and steps that gives the complete software. It follows series of steps which is used for product progress. The development method followed in this project is waterfall model.

5.1.1 Model Phases

The waterfall model is a sequential software development process, in which progress is seen as owing steadily downwards (like a waterfall) through the phases of Requirement initiation, Analysis, Design, Implementation, Testing and maintenance.

Requirement Analysis: This phase is concerned about collection of requirement of the system. This process involves generating document and requirement review.

System Design: Keeping the requirements in mind the system specifications are translated in to a software representation. In this phase the designer emphasizes on algorithm, data structure, software architecture etc.

Coding: In this phase programmer starts his coding in order to give a full sketch of product. In other words system specifications are only converted in to machine readable compute code.

Implementation: The implementation phase involves the actual coding or programming of the software. The output of this phase is typically the library,executables, user manuals and additional software documentation

Testing: In this phase all programs (models) are integrated and tested to ensure that the complete system meets the software requirements. The testing is concerned with verification and validation.

Maintenance: The maintenance phase is the longest phase in which the software is updated to fulfil the changing customer need, adapt to accommodate change in the external environment, correct errors and oversights previously undetected in the testing phase, enhance the efficiency of the software.

5.1.2 Reason for choosing waterfall model as development model

- Clear project objectives.
- Stable project requirements.
- Progress of system is measurable.
- Strict sign-off requirements.
- Helps you to be perfect.
- Logic of software development is clearly understood.
- Production of a formal specification
- Better resource allocation.
- Improves quality. The emphasis on requirements and design before writing a single line of code ensures minimal wastage of time and effort and reduces the risk of schedule slippage.
- Less human resources required as once one phase is finished those people can start working on to the next phase.

5.2 Design using UML

Designing UML diagram specifies, how the process within the system communicates along with how the objects with in the process collaborate using both static as well as dynamic UML diagrams since in this ever-changing world of Object Oriented application development, it has been getting harder and harder to develop and manage high quality applications in reasonable amount of time. As a result of this challenge and the need for a universal object modeling language every one could use, the Unified Modeling Language (UML) is the Information industries version of blue print. It is a method for describing the systems architecture in detail. Easier to build or maintains system, and to ensure that the system will hold up to the requirement changes.

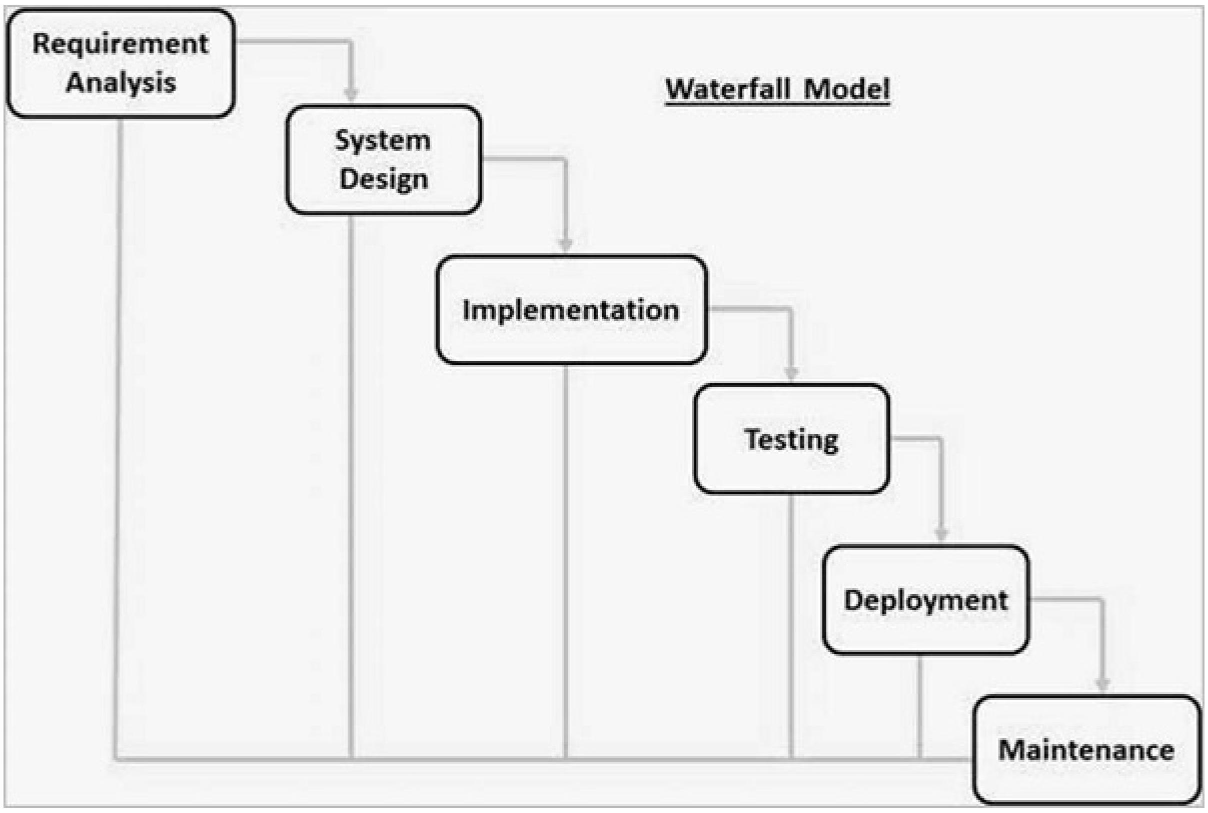
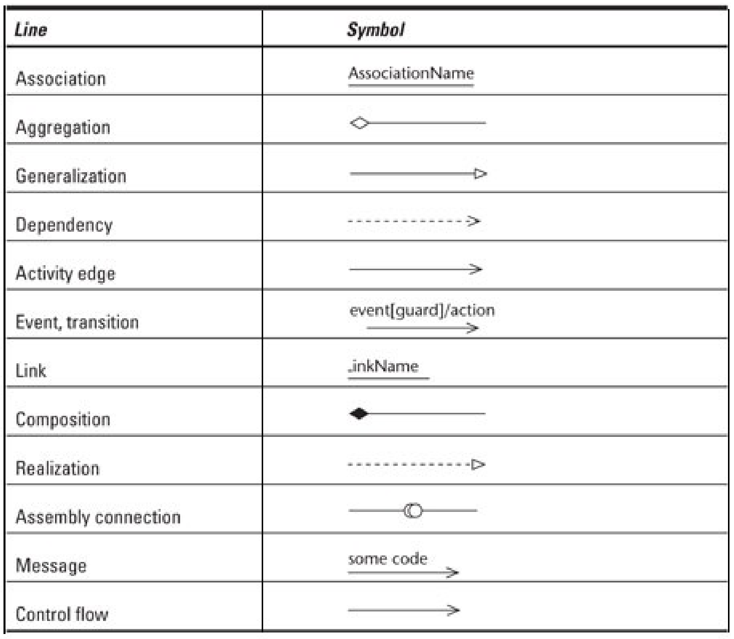


Figure 5.1: Waterfall Model

5.3 Data Flow Diagram

A data flow diagram (DFD) is graphic representation of the ”flow” of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. DFDs show the flow of data from external entities into the system, how the data moves from one process to another, as well as its logical storage. There are only four symbols: 1. Squares representing external entities, which are sources and destinations of information entering and leaving the system. 2. Rounded rectangles representing processes, in other methodologies, may be called ’Activities’, ’Actions’, ’Procedures’, ’Subsystems’ etc. which take data as input, do processing to it, and output it. 3. Arrows representing the data flows, which can either, be electronic data or physical items. It is impossible for data to flow from data store to data store except via a process, and external entities are not allowed to access data stores directly. 4. The three-sided rectangle is representing data stores should both receive information for storing and provide it for further processing.



Line	*Symbol*
Association	AssociationName
Aggregation	◇———
Generalization	———▷
Dependency	- - - - - - - ->
Activity edge	———>
Event, transition	event[guard]/action ——>
Link	LinkName
Composition	◆———
Realization	- - - - - - - -▷
Assembly connection	——◯——
Message	some code ——>
Control flow	———>

Figure 5.2: Data Flow Diagram

5.4 Class Diagram

UML class diagram shows the static structure of the model. The class diagram is a collection of static modeling elements, such as classes and their relationships, connected as a graph to each other and to their contents. The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed.

5.5 Use Case Diagram

A use case defines a goal-oriented set of interactions between external entities and the system under consideration. The external entities which interact with the system are its actors. A set of use cases describe the complete functionality of the system at a particular level of detail and it can be graphically denoted by the use case diagram.

5.6 Activity Diagram

An activity diagram shows the sequence of steps that make up a complex process. An activity is shown as a round box containing the name of the operation. An outgoing solid arrow attached to the end of the activity symbol indicates a transition triggered

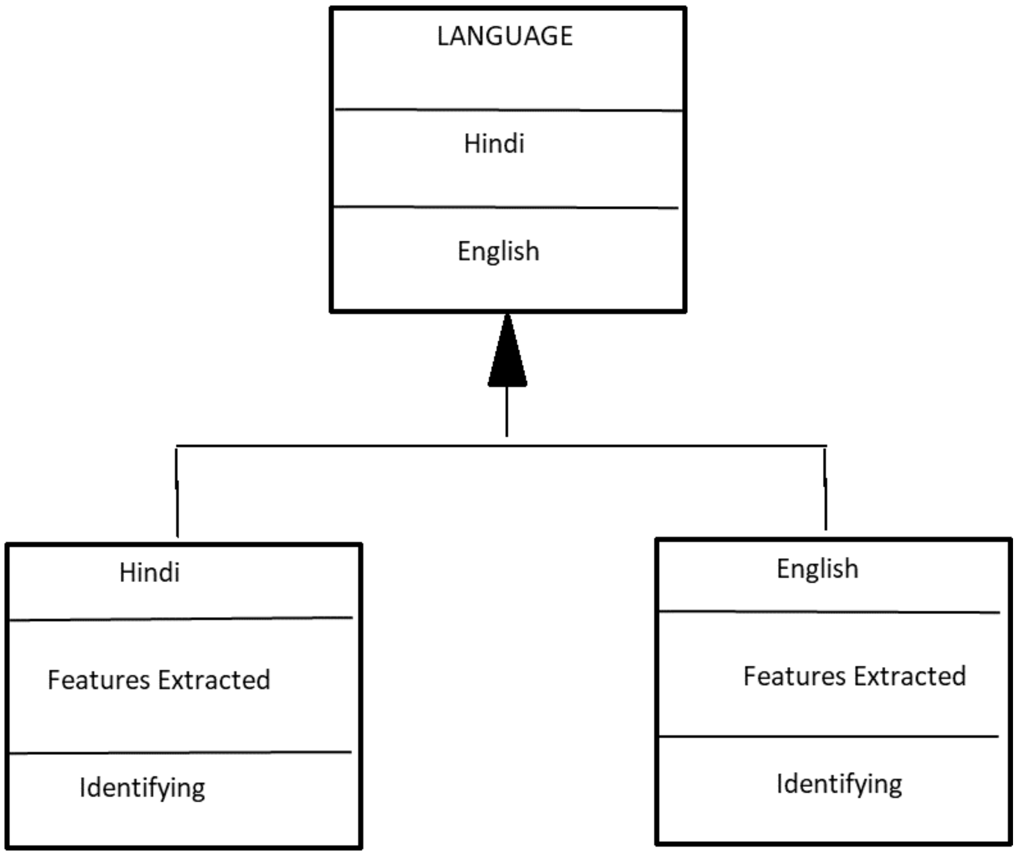


Figure 5.3: Class Diagram

by the completion. Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

5.7 Sequence Diagram

Sequence diagram are an easy and intuitive way of describing the behaviour of a system by viewing the interaction between the system and the environment. A sequence diagram shows an interaction arranged in a time sequence. A sequence diagram has two dimensions: vertical dimension represents time, the horizontal dimension

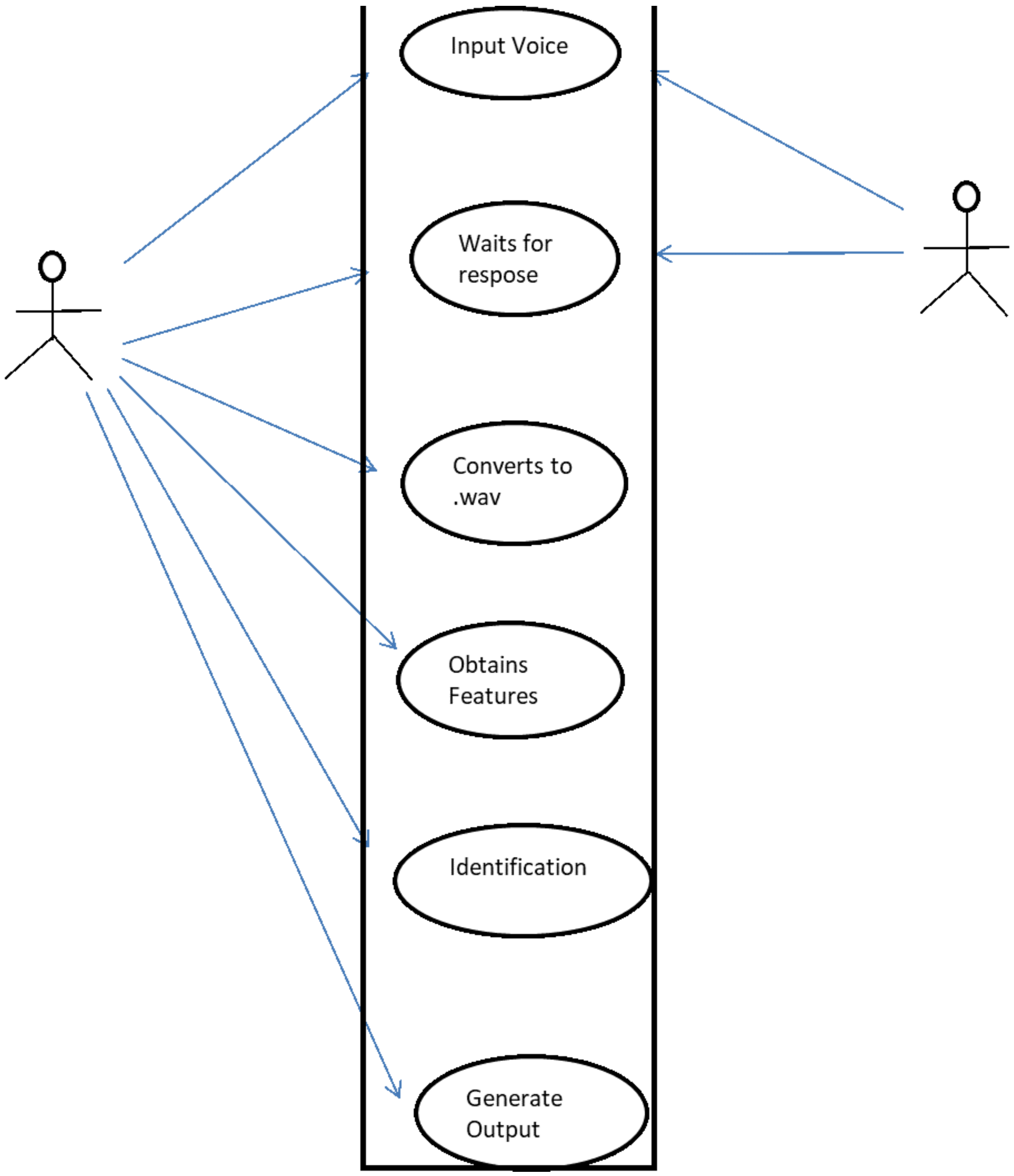


Figure 5.4: Use Case Diagram

represents the objects existence during the interaction. Basic elements:

- Vertical rectangle: Represent the object is active (method is being performed).
- Vertical dashed line: Represent the life of the object.
- X: represent the life end of an object. (Being destroyed from memory)
- Horizontal line with arrows: Messages from one object to another.

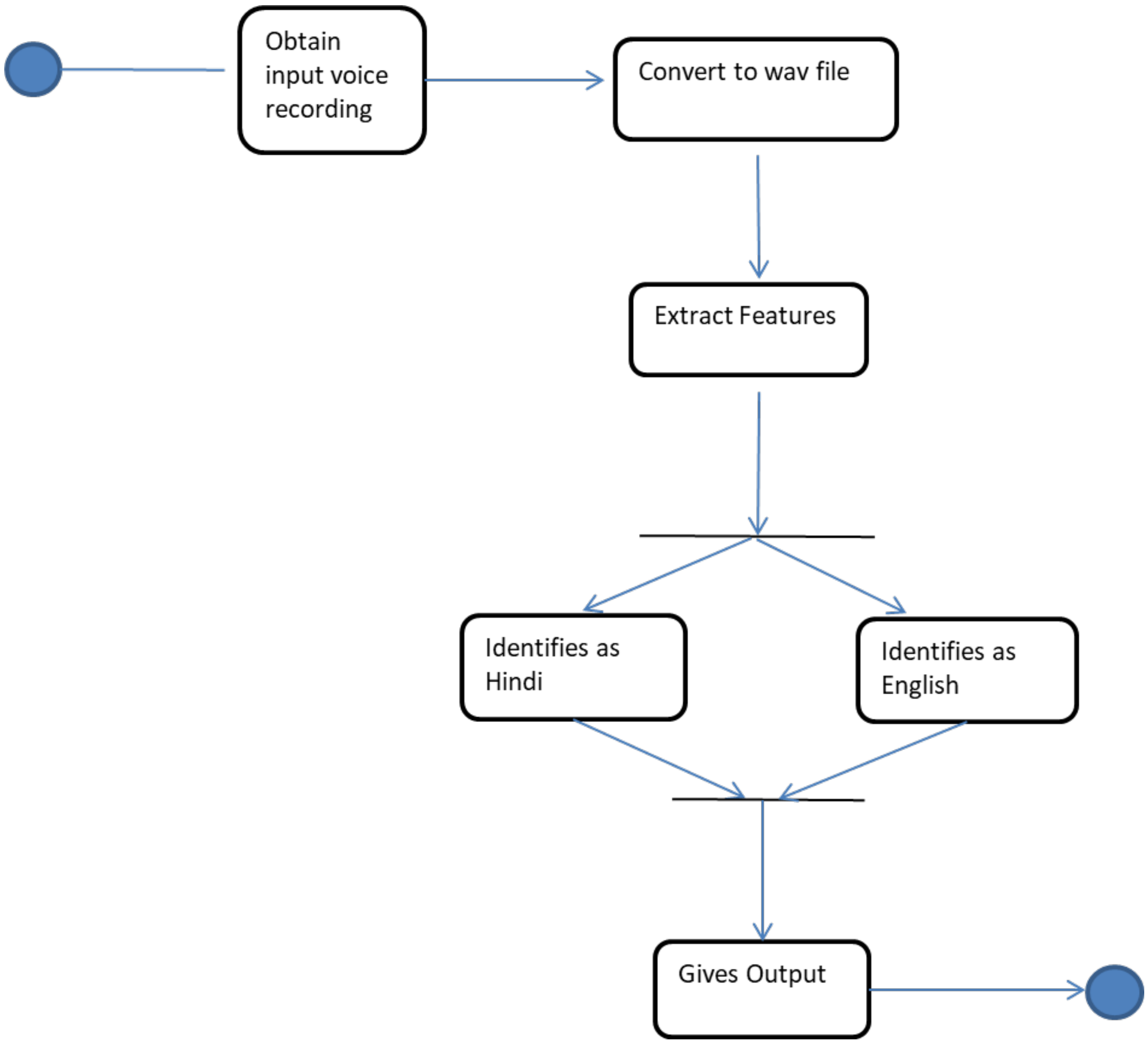


Figure 5.5: Activity Diagram

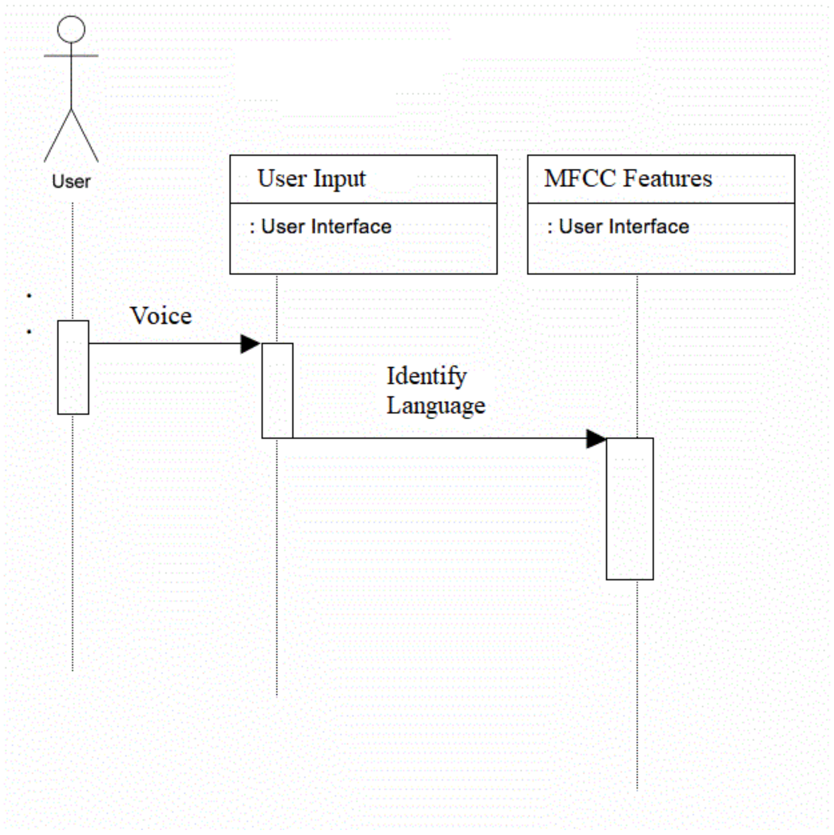


Figure 5.6: Sequence Diagram

Chapter 6

Implementation

6.1 Introduction

The implementation phase of the project is where the detailed design is actually transformed into working code. Aim of the phase is to translate the design into a best possible solution in a suitable programming language. This chapter covers the implementation aspects of the project, giving details of the programming language and development environment used. It also gives an overview of the core modules of the project with their step by step flow. The implementation stage requires the following tasks:

Careful planning.
Investigation of system and constraints.
Design of methods to achieve the changeover.
Evaluation of the changeover method.
Correct decisions regarding selection of the platform.
Appropriate selection of the language for application development.

6.2 Training Code

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split



# Reading the dataset
def read_dataset():
    df = pd.read_csv("C:\\Users\\Aparna_Vinod\\PycharmProjects\\Language
    #print(len(df.columns))
    X = df[df.columns[0:5603]].values
    y = df[df.columns[5603]]

    # Encode the dependent variable
    encoder = LabelEncoder()
    encoder.fit(y)
    y = encoder.transform(y)
    Y = one_hot_encode(y)
    print(X.shape)
    return (X, Y)



# Define the encoder function.
def one_hot_encode(labels):
    n_labels = len(labels)
    n_unique_labels = len(np.unique(labels))
    one_hot_encode = np.zeros((n_labels, n_unique_labels))
    one_hot_encode[np.arange(n_labels), labels] = 1
    return one_hot_encode



# Read the dataset
X, Y = read_dataset()

# Shuffle the dataset to mix up the rows.
X, Y = shuffle(X, Y, random_state=1)

# Convert the dataset into train and test part
train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size=0.20

# Inpect the shape of the training and testing.
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)

# Define the important parameters and variable to work with the tensors
learning_rate = 0.3
training_epochs = 1000
cost_history = np.empty(shape=[1], dtype=float)
n_dim = X.shape[1]
print("n_dim", n_dim)
n_class = 2
model_path = "C:\\Users\\Aparna_Vinod\\PycharmProjects\\Language_Recogni

# Define the number of hidden layers and number of neurons for each laye
n_hidden_1 = 60
n_hidden_2 = 60
```

```
n_hidden_3 = 60
n_hidden_4 = 60

x = tf.placeholder(tf.float32, [None, n_dim])
W= tf.Variable(tf.zeros([n_dim, n_class]))
b = tf.Variable(tf.zeros([n_class]))
y_ = tf.placeholder(tf.float32, [None, n_class])


# Define the model
def multilayer_perceptron(x, weights, biases):

    # Hidden layer with RELU activationsd
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.sigmoid(layer_1)

    # Hidden layer with sigmoid activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.sigmoid(layer_2)

    # Hidden layer with sigmoid activation
    layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
    layer_3 = tf.nn.sigmoid(layer_3)

    # Hidden layer with RELU activation
    layer_4 = tf.add(tf.matmul(layer_3, weights['h4']), biases['b4'])
    layer_4 = tf.nn.relu(layer_4)

    # Output layer with linear activation
    out_layer = tf.matmul(layer_4, weights['out']) + biases['out']
    return out_layer


# Define the weights and the biases for each layer

weights = {
    'h1': tf.Variable(tf.truncated_normal([n_dim, n_hidden_1])),
    'h2': tf.Variable(tf.truncated_normal([n_hidden_1, n_hidden_2])),
    'h3': tf.Variable(tf.truncated_normal([n_hidden_2, n_hidden_3])),
    'h4': tf.Variable(tf.truncated_normal([n_hidden_3, n_hidden_4])),
    'out': tf.Variable(tf.truncated_normal([n_hidden_4, n_class]))
}
biases = {
    'b1': tf.Variable(tf.truncated_normal([n_hidden_1])),
    'b2': tf.Variable(tf.truncated_normal([n_hidden_2])),
    'b3': tf.Variable(tf.truncated_normal([n_hidden_3])),
    'b4': tf.Variable(tf.truncated_normal([n_hidden_4])),
    'out': tf.Variable(tf.truncated_normal([n_class]))
}

# Initialize all the variables

init = tf.global_variables_initializer()

saver = tf.train.Saver()

# Call your model defined
y = multilayer_perceptron(x, weights, biases)

# Define the cost function and optimizer
cost_function = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(l
training_step = tf.train.GradientDescentOptimizer(learning_rate).minimiz

sess = tf.Session()
sess.run(init)

# Calculate the cost and the accuracy for each epoch

mse_history = []
accuracy_history = []

for epoch in range(training_epochs):
    sess.run(training_step, feed_dict={x: train_x, y_: train_y})
    cost = sess.run(cost_function, feed_dict={x: train_x, y_: train_y})
    cost_history = np.append(cost_history, cost)
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # print("Accuracy:␣", (sess.run(accuracy, feed_dict={x: test_x, y_:
```

```
    pred_y = sess.run(y, feed_dict={x: test_x})
    mse = tf.reduce_mean(tf.square(pred_y - test_y))
    mse_ = sess.run(mse)
    mse_history.append(mse_)
    accuracy = (sess.run(accuracy, feed_dict={x: train_x, y_: train_y}))
    accuracy_history.append(accuracy)

    print('epoch␣:␣', epoch, '␣-␣', 'cost:␣', cost, "␣-␣MSE:␣", mse_, "-

save_path = saver.save(sess, model_path)
print("Model␣saved␣in␣file:␣%s" % save_path)

# Plot mse and accuracy graph

plt.plot(mse_history, 'r')
plt.show()

plt.plot(accuracy_history)
plt.show()

# Print the final accuracy

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Test␣Accuracy:␣", (sess.run(accuracy, feed_dict={x: test_x, y_: t

# Print the final mean square error

pred_y = sess.run(y, feed_dict={x: test_x})
mse = tf.reduce_mean(tf.square(pred_y - test_y))
print("MSE:␣%.4f" % sess.run(mse))
```

6.3 Testing Code

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import random


# Reading the dataset
def read_dataset():
    df = pd.read_csv("C:\\Users\\Aparna_Vinod\\PycharmProjects\\Language
    X = df[df.columns[0:5603]].values
    y1 = df[df.columns[5603]]
    encoder = LabelEncoder()
    encoder.fit(y1)
    y = encoder.transform(y1)
    Y = one_hot_encode(y)

    # Return
    return (X, Y, y1)


# Define the encoder function.
def one_hot_encode(labels):
    n_labels = len(labels)
    n_unique_labels = len(np.unique(labels))
    one_hot_encode = np.zeros((n_labels, n_unique_labels))
    one_hot_encode[np.arange(n_labels), labels] = 1
    return one_hot_encode


X, Y, y1 = read_dataset()

model_path = "C:\\Users\\Aparna_Vinod\\PycharmProjects\\Language_Recogni
learning_rate = 0.3
training_epochs = 1000
cost_history = np.empty(shape=[1], dtype=float)
n_dim = 5603
n_class = 2

# Define the number of hidden layers and number of neurons for each laye
n_hidden_1 = 60
```

```
n_hidden_2 = 60
n_hidden_3 = 60
n_hidden_4 = 60

x = tf.placeholder(tf.float32, [None, n_dim])
W= tf.Variable(tf.zeros([n_dim, n_class]))
b = tf.Variable(tf.zeros([n_class]))
y_ = tf.placeholder(tf.float32, [None, n_class])


# Define the model
def multilayer_perceptron(x, weights, biases):

    # Hidden layer with RELU activationsd
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.sigmoid(layer_1)

    # Hidden layer with RELU activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.sigmoid(layer_2)

    # Hidden layer with RELU activation
    layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
    layer_3 = tf.nn.sigmoid(layer_3)

    # Hidden layer with RELU activation
    layer_4 = tf.add(tf.matmul(layer_3, weights['h4']), biases['b4'])
    layer_4 = tf.nn.relu(layer_4)

    # Output layer with linear activation
    out_layer = tf.matmul(layer_4, weights['out']) + biases['out']
    return out_layer


# Define the weights and the biases for each layer
weights = {
    'h1': tf.Variable(tf.truncated_normal([n_dim, n_hidden_1])),
    'h2': tf.Variable(tf.truncated_normal([n_hidden_1, n_hidden_2])),
    'h3': tf.Variable(tf.truncated_normal([n_hidden_2, n_hidden_3])),
    'h4': tf.Variable(tf.truncated_normal([n_hidden_3, n_hidden_4])),
    'out': tf.Variable(tf.truncated_normal([n_hidden_4, n_class]))
}
biases = {
    'b1': tf.Variable(tf.truncated_normal([n_hidden_1])),
    'b2': tf.Variable(tf.truncated_normal([n_hidden_2])),
    'b3': tf.Variable(tf.truncated_normal([n_hidden_3])),
    'b4': tf.Variable(tf.truncated_normal([n_hidden_4])),
    'out': tf.Variable(tf.truncated_normal([n_class]))
}

# Initialize all the variables

init = tf.global_variables_initializer()

saver = tf.train.Saver()

# Call your model defined
y = multilayer_perceptron(x, weights, biases)

# Define the cost function and optimizer
cost_function = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(l
training_step = tf.train.GradientDescentOptimizer(learning_rate).minimiz

init = tf.global_variables_initializer()
saver = tf.train.Saver()
sess = tf.Session()
sess.run(init)
saver.restore(sess, model_path)

prediction = tf.argmax(y, 1)
correct_prediction = tf.equal(prediction, tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# print (accuracy_run)
print('*****************************************************')
print(" 0 Stands for E i.e. English & 1 Stands for H i.e. Hindi")
print('*****************************************************')
for i in range(480,520):
    prediction_run = sess.run(prediction, feed_dict={x: X[i].reshape(1,
    accuracy_run = sess.run(accuracy, feed_dict={x: X[i].reshape(1, 5603
    print("Original Class : ", y1[i], " Predicted Values : ", prediction


# print(sess.run(prediction, feed_dict={x: x_test}))
# print(sess.run(accuracy,  feed_dict={x: x_test, y_: y_test}))
```

6.4 Feature Extraction Code

```
import librosa
import csv
import numpy
import os
from pathlib import Path

#Librosa is used for MFCC feature extraction
#csv module is used to store the array as a spreadsheet



# 1. Get the file path to the included audio example



audio_file_eng=os.listdir("C:\\Python35\\LangPred\\Audio\\English_wav")
audio_file_hin=os.listdir("C:\\Python35\\LangPred\\Audio\\Hindi_wav")

csvfile = "C:\\Python35\\LangPred\\MFCC_FEATURES.csv"
if os.path.exists(csvfile):
    os.remove("C:\\Python35\\LangPred\\MFCC_FEATURES.csv")

with open(csvfile, "a") as output:
    writer = csv.writer(output, lineterminator='\n')

    for audio_file in audio_file_eng:
        print (audio_file)
        # 2. Load the audio as a waveform `y`
        #    Store the sampling rate as `sr`
        y, sr = librosa.load("C:\\Python35\\LangPred\\Audio\\English_wav
        # 3. Run the default beat tracker
        #tempo, beat_frames = librosa.beat.beat_track(y=y, sr=sr)

        #print('Estimated␣tempo:␣{:.2f}␣beats␣per␣minute'.format(tempo))

        # 4. Convert the frame indices of beat events into timestamps
        #beat_times = librosa.frames_to_time(beat_frames, sr=sr)
        #print("y␣=",y,"\nsr␣=",sr)
        #print('Saving␣output␣to␣beat_times.csv')
        #librosa.output.times_csv('beat_times.csv', beat_times)
        mfccs=librosa.feature.mfcc(y, sr, n_mfcc=13)
        mfccs2=numpy.concatenate(mfccs)
        mfccs2=numpy.append(mfccs,["E"])
        #print("Features␣extracted␣:␣\n",mfccs,"\nShape␣of␣List␣:␣",mfcc
        print ("*"*50)
        writer.writerow(mfccs2)
    print ("*"*50)
        #writer.writerows("\n\n\n")
    for audio_file in audio_file_hin:
        print (audio_file)
        # 2. Load the audio as a waveform `y`
        #    Store the sampling rate as `sr`
        y, sr = librosa.load("C:\\Python35\\LangPred\\Audio\\Hindi_wav\\

        # 3. Run the default beat tracker
        #tempo, beat_frames = librosa.beat.beat_track(y=y, sr=sr)

        #print('Estimated␣tempo:␣{:.2f}␣beats␣per␣minute'.format(tempo))

        # 4. Convert the frame indices of beat events into timestamps
        #beat_times = librosa.frames_to_time(beat_frames, sr=sr)
        #print("y␣=",y,"\nsr␣=",sr)
        #print('Saving␣output␣to␣beat_times.csv')
        #librosa.output.times_csv('beat_times.csv', beat_times)
        mfccs=librosa.feature.mfcc(y, sr, n_mfcc=13)
        mfccs2=numpy.concatenate(mfccs)
        mfccs2=numpy.append(mfccs,["H"])
        #print("Features␣extracted␣:␣\n",mfccs,"\nShape␣of␣List␣:␣",mfcc
        print ("*"*50)
```

```
            writer.writerow(mfccs2)
output.close()
```

6.5 Process

- A number of recordings (In this case, 500 English recordings and 288 Hindi recordings) have been taken.
- Each of these recordings is in mp3 format. They need to be .wav format and can easily be converted using an online converter (see future scope)
- The feature extraction code also trims every recording into ten seconds.
- Once the recordings are converted, features need to extracted from them in order to classify into Hindi and English.
- This can be done using the Librosa function that is contained in the feature extraction code.
- What the code will do, is extract MFCC features from each recording and classify them into Hindi or English. Once this is complete, the features will get stored into an excel sheet, with a class H or E at the end of each row.
- After all 788 recordings features have been extracted, a matrix in the excel sheet of size 5603*788 is formed, i.e., 5603 rows and 788 columns.
- This needs to be stored in CSV format.
- Once the sheet is filled, it will be inputed into the training code and training of the machine will begin.

6.6 Important Libraries used

6.6.1 Librosa

LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create audio information retrieval systems.

- librosa.beat Functions for estimating tempo and detecting beat events.
- librosa.core Core functionality includes functions to load audio from disk, compute various spectrogram representations, and a variety of commonly used tools for music analysis. For convenience, all functionality in this submodule is directly accessible from the top-level librosa.* namespace.

- librosa.decompose Functions for harmonic-percussive source separation (HPSS) and generic spectrogram decomposition using matrix decomposition methods implemented in scikit-learn.
- librosa.display Visualization and display routines using matplotlib.
- librosa.effects Time-domain audio processing, such as pitch shifting and time stretching. This submodule also provides time-domain wrappers for the decompose submodule.
- librosa.feature Feature extraction and manipulation. This includes low-level feature extraction, such as chromagrams, pseudo-constant-Q (log-frequency) transforms, Mel spectrogram, MFCC, and tuning estimation. Also provided are feature manipulation methods, such as delta features, memory embedding, and event-synchronous feature alignment.
- librosa.filters Filter-bank generation (chroma, pseudo-CQT, CQT, etc.). These are primarily internal functions used by other parts of librosa.
- librosa.onset Onset detection and onset strength computation.
- librosa.output Text- and wav-file output.
- librosa.segment Functions useful for structural segmentation, such as recurrence matrix construction, time-lag representation, and sequentially constrained clustering.
- librosa.sequence Functions for sequential modeling. Various forms of Viterbi decoding, and helper functions for constructing transition matrices.
- librosa.util Helper utilities (normalization, padding, centering, etc.)

All of these functions are not required, but functions like librosa.beat and librosa.feature have been used.

6.6.2 Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Pythons built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of todays scientific/mathematical Python-based software, just knowing how to use Pythons built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

6.6.3 Matplolib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

6.6.4 Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with relational or labeled data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

Chapter 7

Testing and Results

7.1 Introduction

Testing is an important phase in the development life cycle of the product this was the phase where the error remaining from all the phases was detected. Hence testing performs a very critical role for quality assurance and ensuring the reliability of the software. Once the implementation is done, a test plan should be developed and run on a given set of test data. Each test has a different purpose, all work to verify that all the system elements have been properly integrated and perform allocated functions. The testing process is actually carried out to make sure that the product exactly does the same thing what is supposed to do. Testing is the final verification and validation activity within the organization itself. In the testing stage following goals are tried to achieve:-

- To affirm the quality of the project.
- To find and eliminate any residual errors from previous stages.
- To validate the software as the solution to the original problem.
- To provide operational reliability of the system.

During testing the major activities are concentrated on the examination and modification of the source code. The test cases executed for this project are listed below. Description of the test case, steps to be followed; expected result, status and screenshots are explained with each of the test cases.

7.2 Testing Methodologies

There are many different types of testing methods or techniques used as part of the software testing methodology. Some of the important types of testing are:

7.2.1 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level. Using white box testing we can derive test cases that:

- Guarantee that all independent paths within a module have been exercised at least once.
- Exercise all logical decisions on their true and false sides.

- Execute all loops at their boundaries and within their operational bounds.
- Execute internal data structure to assure their validity.

7.2.2 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box you cannot see into it. The test provides inputs and responds to outputs without considering how the software works. It uncovers a different class of errors in the following categories:

- Incorrect or missing function.
- Interface errors.
- Performance errors.
- Initialization and termination errors.
- Errors in objects.

Advantages:

- The test is unbiased as the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages.
- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.

7.2.3 Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases. Test strategy and approach Field testing will be performed manually and functional tests will be written in detail.
Test objectives:

- All Components must work properly.
- Correct Features of audio clips must be obtained.
- Correct distinguishing of Hindi and English must be made.

Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs
Postconditions

Table 7.1: Unit Test Cases

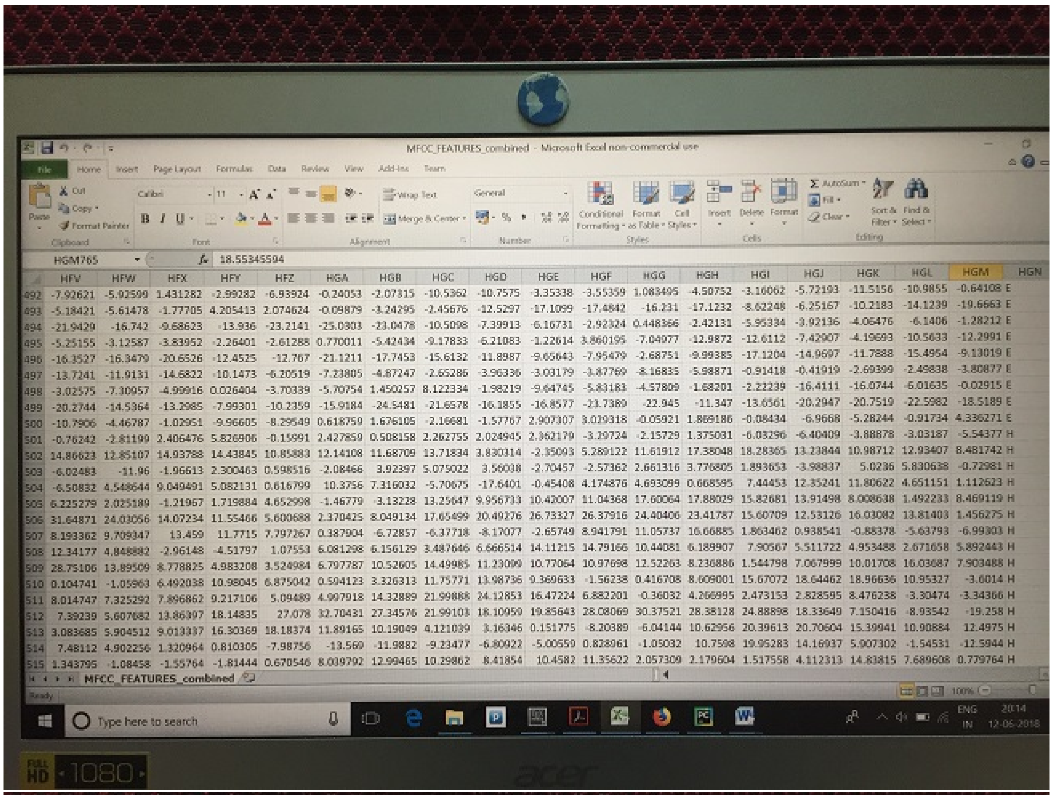
7.3 Running and Testing of codes

7.3.1 Execution of Feature Extraction Code

The code is run in PyCharm, which is a cross platform IDE for Python. One execution of the code, the excel sheet in csv format is generated, containing the features extracted from the 788 recordings. This CSV file will now be inputted into the training and testing codes.

7.3.2 Execution of Training code

The code is run in PyCharm, which is a cross platform IDE for Python. The training of the features of the recordings will run for a 1000 iterations (This can be varied).After the iterations are complete, the Mean Square Error (MSE) is found. MSE is a network performance function. It measures the network's performance according to the mean of squared errors. Difference between predicted value and actual value in test data is the MSE.mse(E,X,PP) takes from one to three arguments,
E - Matrix or cell array of error vector(s).

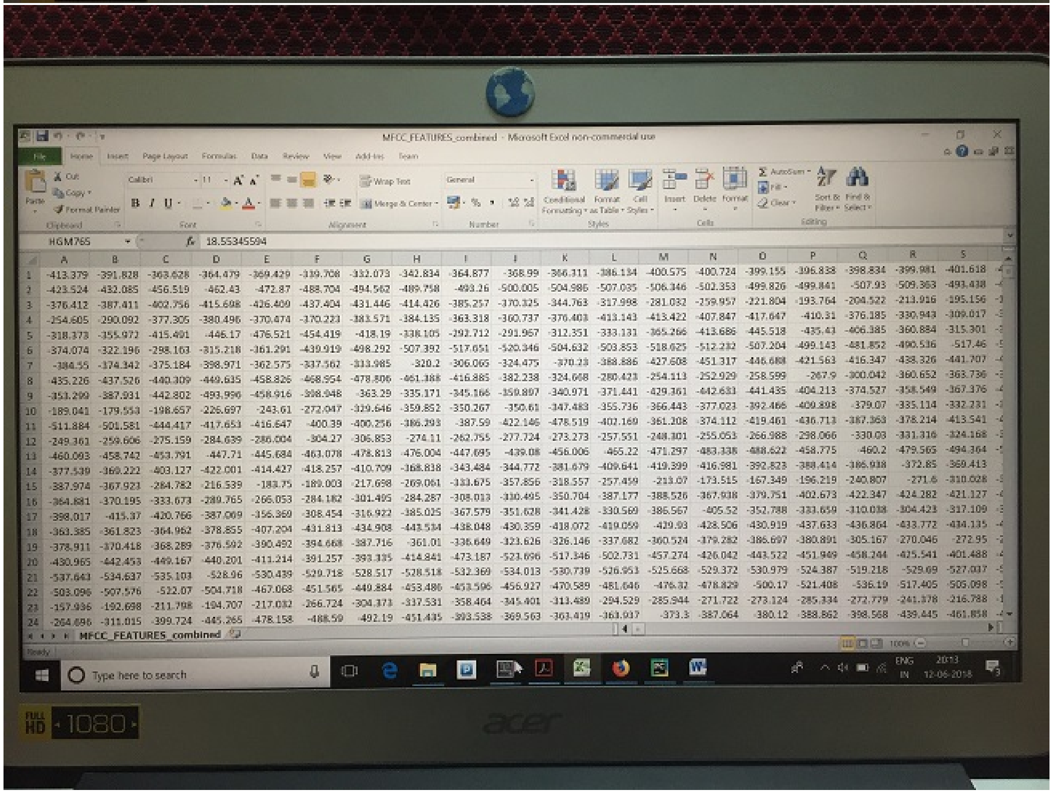


MFCC_FEATURES_combined - Microsoft Excel non-commercial use

HGM765 18.55345594

	HFV	HFW	HFX	HFY	HFZ	HGA	HGB	HGC	HGD	HGE	HGF	HGG	HGH	HGI	HGJ	HGK	HGL	HGM	HGN
492	-7.92621	-5.92599	1.431282	-2.99282	-6.93924	-0.24053	-2.07315	-10.5362	-10.7575	-3.35338	-3.55359	1.083495	-4.50752	-3.16062	-5.72193	-11.5156	-10.9855	-0.64108	E
493	-5.18421	-5.61478	-1.77705	4.205413	2.074624	-0.09879	-3.24295	-2.45676	-12.5297	-17.1099	-17.4842	-16.231	-17.1232	-8.62248	-6.25167	-10.2183	-14.1239	-19.6663	E
494	-21.9429	-16.742	-9.68623	-13.936	-23.2141	-25.0303	-23.0478	-10.5098	-7.39913	-6.16731	-2.92324	0.448366	-2.42131	-5.95334	-3.92136	-4.06476	-6.1406	-1.28212	E
495	-5.25155	-3.12587	-3.83952	-2.26401	-2.61288	0.770011	-5.42434	-9.17833	-6.21083	-1.22614	3.860195	-7.04977	-12.9872	-12.6112	-7.42907	-4.19693	-10.5633	-12.2991	E
496	-16.3527	-16.3479	-20.6526	-12.4525	-12.767	-21.1211	-17.7453	-15.6132	-11.8987	-9.65643	-7.95479	-2.68751	-9.99385	-17.1204	-14.9697	-11.7888	-15.4954	-9.13019	E
497	-13.7241	-11.9131	-14.6822	-10.1473	-6.20519	-7.23805	-4.87247	-2.65286	-3.96336	-3.03179	-3.87769	-8.16835	-5.98871	-0.91418	-0.41919	-2.69399	-2.49838	-3.80877	E
498	-3.02575	-7.30957	-4.99916	0.026404	-3.70339	-5.70754	1.450257	8.122334	-1.98219	-9.64745	-5.83183	-4.57809	-1.68201	-2.22239	-16.4111	-16.0744	-6.01635	-0.02915	E
499	-20.2744	-14.5364	-13.2985	-7.99301	-10.2359	-15.9184	-24.5481	-21.6578	-16.1855	-16.8577	-23.7389	-22.945	-11.347	-13.6561	-20.2947	-20.7519	-22.5982	-18.5189	E
500	-10.7906	-4.46787	-1.02951	-9.96605	-8.29549	0.618759	1.676105	-2.16681	-1.57767	2.907307	3.029318	-0.05921	1.869186	-0.08434	-6.9668	-5.28244	-0.91734	4.336271	E
501	-0.76242	-2.81199	2.406476	5.826906	-0.15991	2.427859	0.508158	2.262755	2.024945	2.362179	-3.29724	-2.15729	1.375031	-6.03296	-6.40409	-3.88878	-3.03187	-5.54377	H
502	14.86623	12.85107	14.93788	14.43845	10.85883	12.14108	11.68709	13.71834	3.830314	-2.35093	5.289122	11.61912	17.38048	18.28365	13.23844	10.98712	12.93407	8.481742	H
503	-6.02483	-11.96	-1.96613	2.300463	0.598516	-2.08466	3.92397	5.075022	3.56038	-2.70457	-2.57362	2.661316	3.776805	1.893653	-3.98837	5.0236	5.830638	-0.72981	H
504	-6.50832	4.548644	9.049491	5.082131	0.616799	10.3756	7.316032	-5.70675	-17.6401	-0.45408	4.174876	4.693099	0.668595	7.44453	12.35241	11.80622	4.651151	1.112623	H
505	6.225279	2.025189	-1.21967	1.719884	4.652998	-1.46779	-3.13228	13.25647	9.956733	10.42007	11.04368	17.60064	17.88029	15.82681	13.91498	8.008638	1.492233	8.469119	H
506	31.64871	24.03056	14.07234	11.55466	5.600688	2.370425	8.049134	17.65499	20.49276	26.73327	26.37916	24.40406	23.41787	15.60709	12.53126	16.03082	13.81403	1.456275	H
507	8.193362	9.709347	13.459	11.7715	7.797267	0.387904	-6.72857	-6.37718	-8.17077	-2.65749	8.941791	11.05737	16.66885	1.863462	0.938541	-0.88378	-5.63793	-6.99303	H
508	12.34177	4.848882	-2.96148	-4.51797	1.07553	6.081298	6.156129	3.487646	6.666514	14.11215	14.79166	10.44081	6.189907	7.90567	5.511722	4.953488	2.671658	5.892443	H
509	28.75106	13.89509	8.778825	4.983208	3.524984	6.797787	10.52605	14.49985	11.23099	10.77064	10.97698	12.52263	8.236886	1.544798	7.067999	10.01708	16.03687	7.903488	H
510	0.104741	-1.05963	6.492038	10.98045	6.875042	0.594123	3.326313	11.75771	13.98736	9.369633	-1.56238	0.416708	8.609001	15.67072	18.64462	18.96636	10.95327	-3.6014	H
511	8.014747	7.325292	7.896862	9.217106	5.09489	4.997918	14.32889	21.99888	24.12853	16.47224	6.882201	-0.36032	4.266995	2.473153	2.828595	8.476238	-3.30474	-3.34366	H
512	7.39239	5.607682	13.86397	18.14835	27.078	32.70431	27.34576	21.99103	18.10959	19.85643	28.08069	30.37521	28.38128	24.88898	18.33649	7.150416	-8.93542	-19.258	H
513	3.083685	5.904512	9.013337	16.30369	18.18374	11.89165	10.19049	4.121039	3.16346	0.151775	-8.20389	-6.04144	10.62956	20.39613	20.70604	15.39941	10.90884	12.4975	H
514	7.48112	4.902256	1.320964	0.810305	-7.98756	-13.569	-11.9882	-9.23477	-6.80922	-5.00559	0.828961	-1.05032	10.7598	19.95283	14.16937	5.907302	-1.54531	-12.5944	H
515	1.343795	-1.08458	-1.55764	-1.81444	0.670546	8.039792	12.99465	10.29862	8.41854	10.4582	11.35622	2.057309	2.179604	1.517558	4.112313	14.83815	7.689608	0.779764	H

MFCC_FEATURES_combined



MFCC_FEATURES_combined - Microsoft Excel non-commercial use

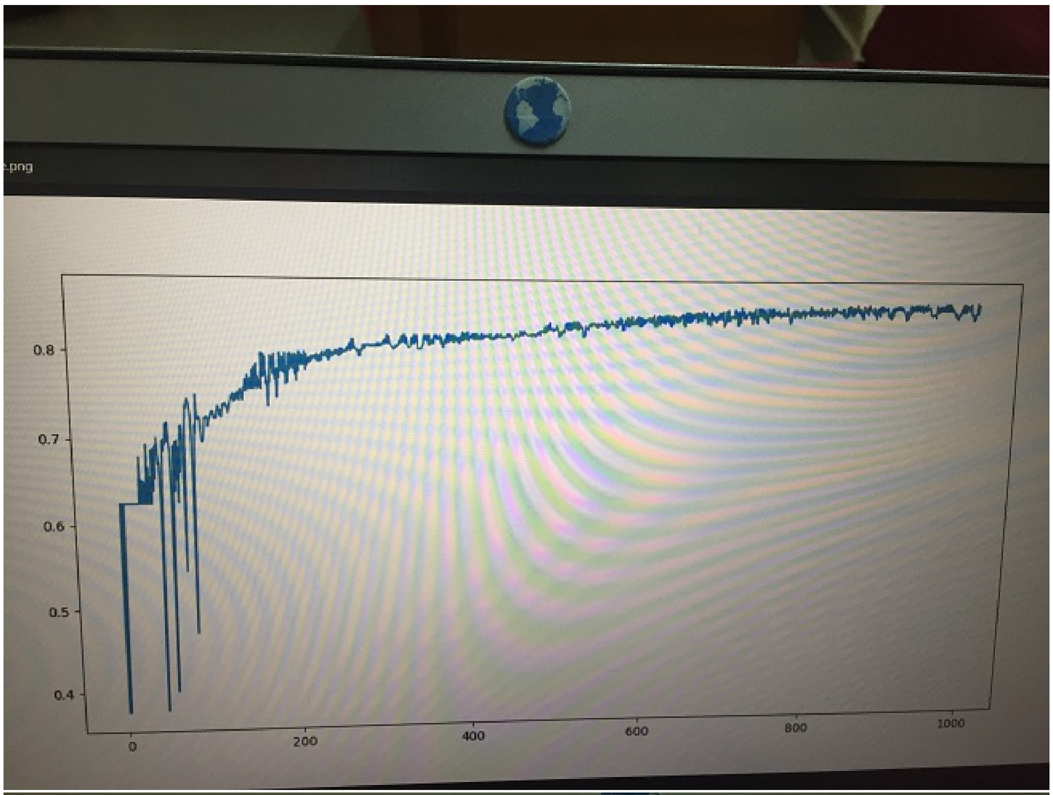
HGM765 18.55345594

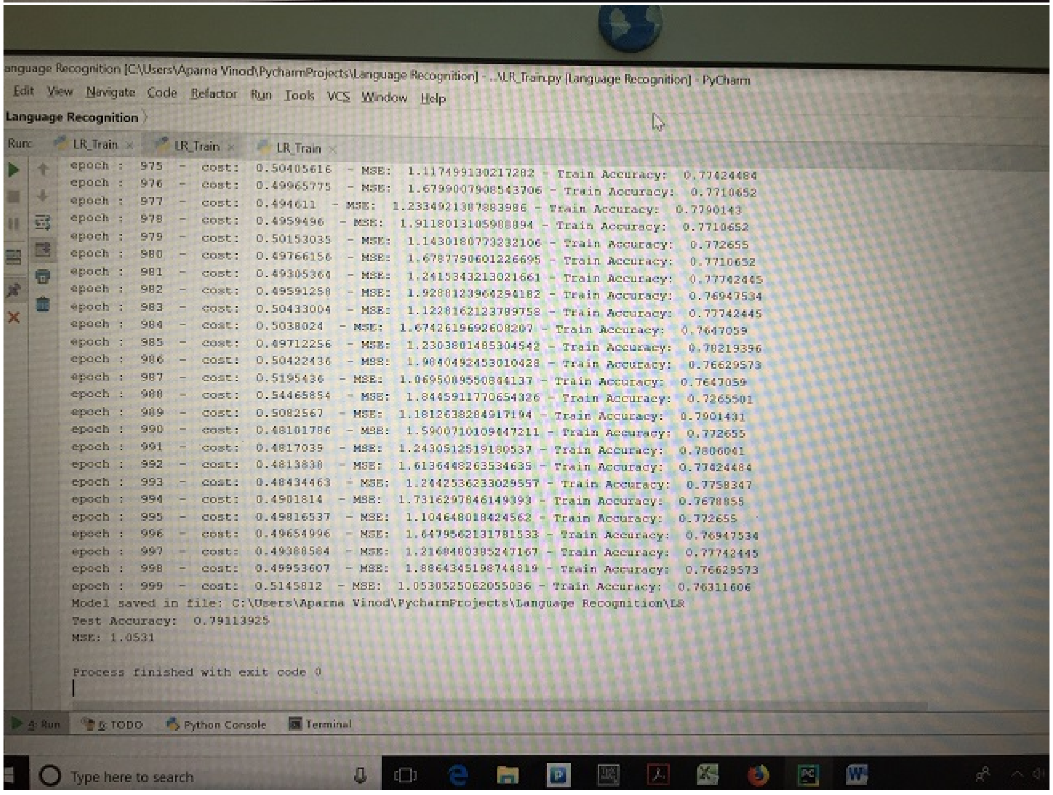
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	-413.379	-391.828	-363.628	-364.479	-369.429	-339.708	-332.073	-342.834	-364.877	-368.99	-366.311	-386.134	-400.575	-400.724	-399.155	-396.838	-398.834	-399.981	-401.618
2	-423.524	-432.085	-456.519	-462.43	-472.87	-488.704	-494.562	-489.758	-493.26	-500.005	-504.986	-507.035	-506.346	-502.353	-499.826	-499.841	-507.93	-509.363	-493.438
3	-376.412	-387.411	-402.756	-415.698	-426.409	-437.404	-431.446	-414.426	-385.257	-370.325	-344.763	-317.998	-281.032	-259.957	-221.804	-193.764	-204.522	-213.916	-195.156
4	-254.605	-290.092	-377.305	-380.496	-370.474	-370.223	-383.571	-384.135	-363.318	-360.737	-376.403	-413.143	-413.422	-407.847	-417.647	-410.31	-376.185	-330.943	-309.017
5	-318.373	-355.972	-415.491	-446.17	-476.521	-454.419	-418.19	-338.105	-292.712	-291.967	-312.351	-333.131	-365.266	-413.686	-445.518	-435.43	-406.385	-360.884	-315.301
6	-374.074	-322.196	-298.163	-315.218	-361.291	-439.919	-498.292	-507.392	-517.651	-520.346	-504.632	-503.853	-518.625	-512.232	-507.204	-499.143	-481.852	-490.536	-517.46
7	-384.55	-374.342	-375.184	-398.971	-362.575	-337.562	-313.985	-320.2	-306.065	-324.475	-370.23	-388.886	-427.608	-451.317	-446.688	-421.563	-416.347	-438.326	-441.707
8	-435.226	-437.526	-440.309	-449.635	-458.826	-468.954	-478.806	-461.388	-416.885	-382.238	-324.668	-280.423	-254.113	-252.929	-258.599	-267.9	-300.042	-360.652	-363.736
9	-353.299	-387.931	-442.802	-493.996	-458.916	-398.948	-363.29	-335.171	-345.166	-350.897	-340.971	-371.441	-429.361	-442.631	-441.435	-404.213	-374.527	-358.549	-367.376
10	-189.041	-179.553	-198.657	-226.697	-243.61	-272.047	-329.646	-359.852	-350.267	-350.61	-347.483	-355.736	-366.443	-377.023	-392.466	-409.898	-379.07	-335.114	-332.231
11	-511.884	-501.581	-444.417	-417.653	-416.647	-400.39	-400.256	-386.393	-387.59	-422.146	-478.519	-402.169	-361.208	-374.112	-419.461	-436.713	-387.363	-378.214	-413.541
12	-249.361	-259.606	-275.159	-284.639	-286.004	-304.27	-306.853	-274.11	-262.755	-277.724	-273.273	-257.551	-248.301	-255.053	-266.988	-298.066	-330.03	-331.316	-324.168
13	-460.093	-458.742	-453.791	-447.71	-445.684	-463.078	-478.813	-476.004	-447.695	-439.08	-456.006	-465.22	-471.297	-483.338	-488.622	-458.775	-460.2	-479.565	-494.364
14	-377.539	-369.222	-403.127	-422.001	-414.427	-418.257	-410.709	-368.838	-343.484	-344.772	-381.679	-409.641	-419.399	-416.981	-392.823	-388.414	-386.938	-372.85	-369.413
15	-387.974	-367.923	-284.782	-216.539	-183.75	-189.003	-217.698	-269.061	-333.675	-357.856	-318.557	-257.459	-213.07	-173.515	-167.349	-196.219	-240.807	-271.6	-310.028
16	-364.881	-370.195	-333.673	-289.765	-266.053	-284.182	-301.495	-284.287	-308.013	-330.495	-350.704	-387.177	-388.526	-367.938	-379.751	-402.673	-422.347	-424.282	-421.127
17	-398.017	-415.37	-420.766	-387.069	-356.369	-308.454	-316.922	-385.025	-367.579	-351.628	-341.428	-330.569	-386.567	-405.52	-352.788	-333.659	-310.038	-304.423	-317.109
18	-363.385	-361.823	-364.962	-378.855	-407.204	-431.813	-434.908	-443.534	-438.048	-430.359	-418.072	-419.059	-419.93	-428.506	-430.919	-437.633	-436.804	-433.772	-434.135
19	-378.911	-370.418	-368.289	-376.592	-390.492	-394.668	-387.716	-361.01	-336.649	-323.626	-326.146	-337.682	-360.524	-379.282	-386.697	-380.891	-305.167	-270.046	-272.95
20	-430.965	-442.453	-449.167	-440.201	-411.214	-391.257	-393.335	-414.841	-473.187	-523.696	-517.346	-502.731	-457.274	-426.042	-443.522	-451.949	-458.244	-425.541	-401.488
21	-537.643	-534.637	-535.103	-528.96	-530.439	-529.718	-528.517	-528.518	-532.369	-534.013	-530.739	-526.953	-525.668	-529.372	-530.979	-524.387	-519.218	-529.69	-527.037
22	-503.096	-507.576	-522.07	-504.718	-467.068	-451.565	-449.884	-453.486	-453.596	-456.927	-470.589	-481.646	-476.32	-478.829	-500.17	-521.408	-536.19	-517.405	-505.098
23	-157.936	-192.698	-211.798	-194.707	-217.032	-266.724	-304.373	-337.531	-358.464	-345.401	-313.489	-294.529	-285.944	-271.722	-273.124	-285.334	-272.779	-241.378	-216.788
24	-264.696	-311.015	-399.724	-445.265	-478.158	-488.59	-492.19	-451.435	-393.538	-369.563	-363.419	-363.937	-373.3	-387.064	-380.12	-388.862	-398.568	-439.445	-461.858

MFCC_FEATURES_combined

X - Vector of all weight and bias values (ignored).
PP - Performance parameters (ignored).
and returns the mean squared error.
Next, the accuracy of the Training data is calculated. This will vary constantly

depending on the number of voice recordings that are inputted. The more the recordings, the higher the accuracy of predicting the correct output.In this case, the accuracy varies from 70-80%.The shape of the training and testing model is also printed.
A graph of Accuracy on the Y-axis and Number of Epochs on the X-axis is then plotted.



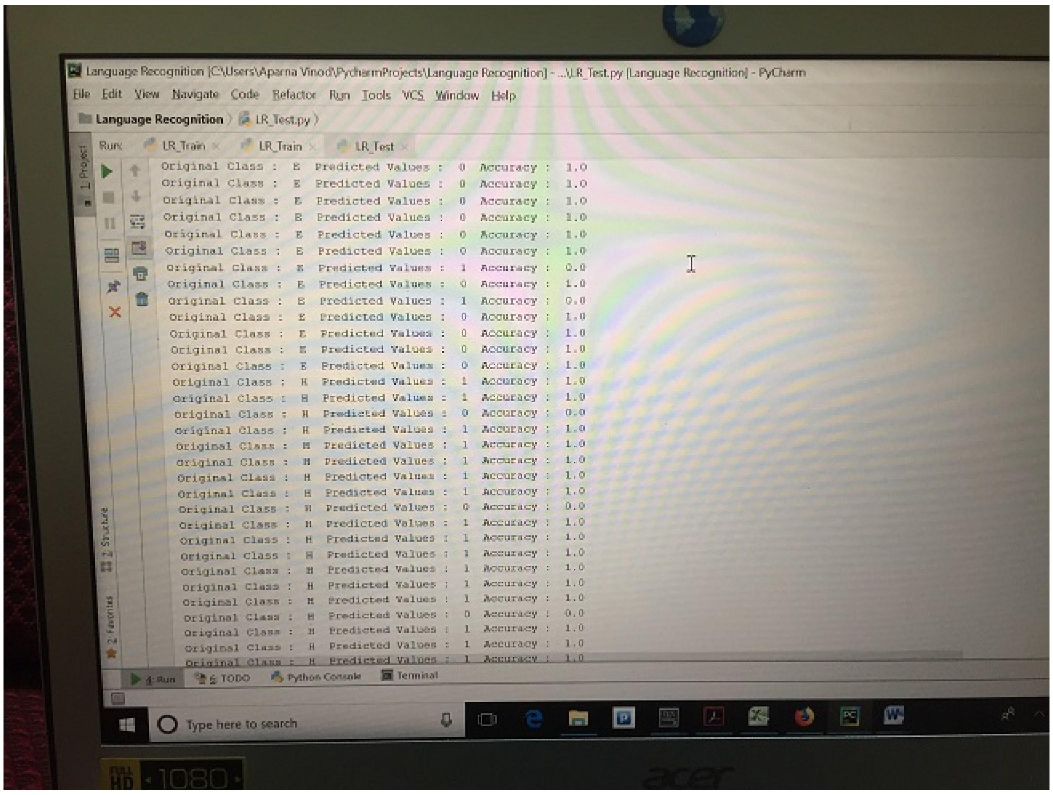

anguage Recognition [C:\Users\Aparna Vinod\PycharmProjects\Language Recognition] - ...\LR_Train.py [Language Recognition] - PyCharm
Edit View Navigate Code Refactor Run Tools VCS Window Help
Language Recognition
Run: LR_Train LR_Train LR_Train
4: Run 6: TODO Python Console Terminal
Type here to search

```
epoch :  975  -  cost:  0.50405616  - MSE:  1.117499130217282 - Train Accuracy:  0.77424484
epoch :  976  -  cost:  0.49965775  - MSE:  1.6799007908543706 - Train Accuracy:  0.7710652
epoch :  977  -  cost:  0.494611  - MSE:  1.2334921387883986 - Train Accuracy:  0.7790143
epoch :  978  -  cost:  0.4959496  - MSE:  1.9118013105988894 - Train Accuracy:  0.7710652
epoch :  979  -  cost:  0.50153035  - MSE:  1.1430180773232106 - Train Accuracy:  0.772655
epoch :  980  -  cost:  0.49766156  - MSE:  1.6787790601226695 - Train Accuracy:  0.7710652
epoch :  981  -  cost:  0.49305364  - MSE:  1.2415343213021661 - Train Accuracy:  0.77742445
epoch :  982  -  cost:  0.49591258  - MSE:  1.9288123964294182 - Train Accuracy:  0.76947534
epoch :  983  -  cost:  0.50433004  - MSE:  1.1228162123789758 - Train Accuracy:  0.77742445
epoch :  984  -  cost:  0.5038024  - MSE:  1.6742619692608207 - Train Accuracy:  0.7647059
epoch :  985  -  cost:  0.49712256  - MSE:  1.2303801485304542 - Train Accuracy:  0.78219396
epoch :  986  -  cost:  0.50422436  - MSE:  1.9040492453010428 - Train Accuracy:  0.76629573
epoch :  987  -  cost:  0.5195436  - MSE:  1.0695089550844137 - Train Accuracy:  0.7647059
epoch :  988  -  cost:  0.54465854  - MSE:  1.8445911770654326 - Train Accuracy:  0.7265501
epoch :  989  -  cost:  0.5082567  - MSE:  1.1812638284917194 - Train Accuracy:  0.7901431
epoch :  990  -  cost:  0.48101786  - MSE:  1.5900710109447211 - Train Accuracy:  0.772655
epoch :  991  -  cost:  0.4817039  - MSE:  1.2430512519180537 - Train Accuracy:  0.7806041
epoch :  992  -  cost:  0.4813838  - MSE:  1.6136448263534635 - Train Accuracy:  0.77424484
epoch :  993  -  cost:  0.48434463  - MSE:  1.2442536233029557 - Train Accuracy:  0.7758347
epoch :  994  -  cost:  0.4901814  - MSE:  1.7316297846149393 - Train Accuracy:  0.7678855
epoch :  995  -  cost:  0.49816537  - MSE:  1.104648018424562 - Train Accuracy:  0.772655
epoch :  996  -  cost:  0.49654996  - MSE:  1.6479562131781533 - Train Accuracy:  0.76947534
epoch :  997  -  cost:  0.49388584  - MSE:  1.2168480385247167 - Train Accuracy:  0.77742445
epoch :  998  -  cost:  0.49953607  - MSE:  1.8864345198744819 - Train Accuracy:  0.76629573
epoch :  999  -  cost:  0.5145812  - MSE:  1.0530525062055036 - Train Accuracy:  0.76311606
Model saved in file: C:\Users\Aparna Vinod\PycharmProjects\Language Recognition\LR
Test Accuracy:  0.79113925
MSE: 1.0531

Process finished with exit code 0
```

7.3.3 Execution of Testing Code

For testing, 20% of the recordings have been separated already by the code.When the testing code is run, the output will be shown, identifying whether the recordings are in Hindi or in English.


Language Recognition [C:\Users\Aparna Vinod\PycharmProjects\Language Recognition] - ...\LR_Test.py [Language Recognition] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Language Recognition
LR_Test.py
Run: LR_Train LR_Train LR_Test
4: Run
6: TODO
Python Console
Terminal
Type here to search

```
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  1  Accuracy :  0.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  1  Accuracy :  0.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  E  Predicted Values :  0  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  0  Accuracy :  0.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  0  Accuracy :  0.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  0  Accuracy :  0.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
Original Class :  H  Predicted Values :  1  Accuracy :  1.0
```

Chapter 8

Conclusion

A project like language identification using mfcc extracted feature values has never been attempted before. Right from the start it was identified that a 100This project has been a terrific journey where lots of knew concepts and studies were learnt, that have been summarized in this report.

8.1 Future Scope

The options for future scope are vast. A user interface for the system needs to be made, wherein a user will speak and the backend process will take place, giving the language spoken as an output.

This could be in the form of an application or implemented into a hardware device.

All processes of converting the mp3 audio into .wav format, placing the recording into the csv file, clipping of the audio etc. can be hidden from the user by creating a good front end.

8.2 Building a complete system

The aim of this project was to implement language recognising features to a telephone call that is being made by a customer to a service provider.

Based on the language, the call must get transferred to a provider who knows that particular language.

Thus a complete system would be identification of over 15 languages. For that thousands of recordings for each language would need to be provided.

References

[1] Deep Learning, by Ian Goodfellow, Yoshua Bengio, Aaron Courville 2016

[2] Tensorflow tutorials, `www.tensorflow.org`

[3] Deep Learning using Tensorflow, tensorflow tutorial on python by Edureka

[4] Speech & Language Processing, by Dan Jurafsky

[5] About deep learning on Wikipedia `https://en.wikipedia.org/wiki/Deep_learning`

[6] About MFCC on `https://www.sciencedirect.com/science/article/pii/S1877705812023053`