MIMIC HUMAN SPEECH IN BAHASA INDONESIA USING SPEECH RECOGNITION AND SPEECH SYNTHESIS

By

Valens Prabagita Ivan Susilo

001201500075

A Thesis
Submitted to the Faculty of Computing
President University
in Partial Fulfilment of the Requirements
for the Degree of Bachelor of Science
in Information Technology

Cikarang, Bekasi, Indonesia

February 2019

MIMIC HUMAN SPEECH IN BAHASA INDONESIA USING

SPEECH RECOGNITION AND SPEECH SYNTHESIS

By

Valens Prabagita Ivan Susilo

Approved:

________________________________	________________________________
Dr. Tjong Wan Sen, S.T., M.T. Thesis Advisor	Drs. Nur Hadisukmana, M.Sc. Program Head of Information Technology

____

Ir. Rila Mandala, M.Eng., Ph.D.
Dean of Faculty of Computing

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other University or Institution.

I have read the Thesis Regulations and I am aware of the potential consequences of any breach of them.

Cikarang, February 2019

Valens Prabagita Ivan Susilo

ABSTRACT

People use speech recognition and speech synthesis to help, support, and boost their daily activities. With just one of the speech technologies, developer can produce various software. Combine both speech technologies, developer could produce more various software. One of the combinations is mimic human speech. This research will discuss about Speech Recognition that use Convolutional Neural Network as machine learning model and Speech Synthesis that use Concatenative Synthesis with syllables as speech unit. Different with the recent related works, this research has simpler approach to mimic speech in Bahasa Indonesia. The purpose of this research is to develop application to collect, train, and mimic speech in Bahasa Indonesia. User can participate to record their speech. Those speeches are collected to be trained for recognizing speech in the application later. With the trained model, now user is able to make the computer mimic their speech. First, user must identify their speech to be recognized by the application. This step is necessary to create the user digital speech. After that, based on registered syllables, which the speech has been identified by the application, user is able to generate speech by making sentences from those syllables. The applications to collect and mimic speech are developed as website application and the application to train is developed as command prompt.

Keywords*: Mimic Speech, Speech Recognition, Speech Synthesis, Convolutional Neural Network, Concatenative Synthesis, Syllables, Bahasa Indonesia*

ACKNOWLEDGMENTS

The author gratefully acknowledges the amazing supports, guidance and advices from:

1. Bebe, Annisa Feryannie Widya.
2. My beloved family, Ayah, Ibu, Mas Risky and Dek Lia.
3. B35TFR13NDZ0NE, Nisa, Marsel, Bora, Fira, Rai, Pindy, Oscar, and Michel.
4. Sangar Team, Mas Andreas, Mba Diana, Nisa, Alfian, and Aci
5. DV 14, Vovo, Gojal, Ariyel, Rino, Willy, Azmi, Damara, Atisah, Risda, Sonya, Dini, and Arizha.
6. My cool thesis lecturer, Mr. Wan Sen.
7. My second family, Van Lith XXII.
8. And you, yes you.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

LIST OF EQUATIONS

FIGURE Page

CHAPTER I

INTRODUCTION

1.1 Background

“Ok Google, play some music”. “Siri, what should I eat for lunch?”. People use their artificial assistance to boost their activities. People tend to use it because they only talked to their device and then in seconds, the wish is granted. The truth is, speech recognition takes big role with the help of machine learning. Google Assistance, Apple Siri, Microsoft Cortana, Amazon Alexa, and others have thousands of speech data to be analysed using the machine learning and they easily add data by collecting user speech from the assistance with permission.

If speech recognition is the process to get data by analysing speech, the opposite of speech recognition is speech synthesis, the process to produce artificial speech. Therefore, speech recognition is known as speech-to-text and speech synthesis is known as text-to-speech. “Hey Cortana, read my email” command make artificial assistance generate speech from the email text. With just one of the speech technologies, developer can produce various software. Combine both speech technologies, developer could produce more various software. One of the combinations is mimic human speech. The most known usage of mimic human speech is creating a digital speech that will be used as the artificial assistance’s speech vocal. Making the artificial assistance more private or personal to the user.

1.2 Problem Statement

This research aims to answer how to implement speech recognition and speech synthesis to build mimic speech system.

1.3 Research Objective

This research aims to develop application which can be used and able to:

1. Collect speech data.
2. Train machine learning model with the collected data.
3. Mimic speech in Bahasa Indonesia.
4. Recognize most of times recorded speech with trained machine learning model.
5. Generate speech based on inputted text.

The Collect and Mimic application are developed as website application, and Train application is developed as command prompt application.

1.4 Scope and Limitation

This research focuses on developing an application which will be able to:

1. Perform collecting data.
2. Perform train model.
3. Perform speech recognition.
4. Perform speech synthesis.

The limitations of this application are as following:

1. There are 9 selected syllables to be used in the application, a, i, na, ma, mu, di, ri, ku, and kan. The syllables are used as speech unit.

2. The duration of the recorded speech in 1 second, with sample rate 16000 and mono sound.
3. Speech recognition is used to recognize speech when user want to create digital speech in identifying speech process.
4. Speech synthesis is used to generate speech based on selected digital speech and inputted text in generating speech process.

1.5 Thesis Methodology

Rapid Application Development (RAD) methodology will be used in the development of this application. RAD method is a term originally used to describe a software development process introduced by James Martin in 1991 [1]. RAD method is a methodology to develop software that requires minimum planning for rapid prototyping. As James Martin says, RAD is a lifecycle used for development of software which provides faster development and also gives high quality software then, by using traditional software development lifecycle. In short, RAD is the process which accelerates the cycle of development of an application [2]. The RAD diagram is depicted in Figure 1.1.

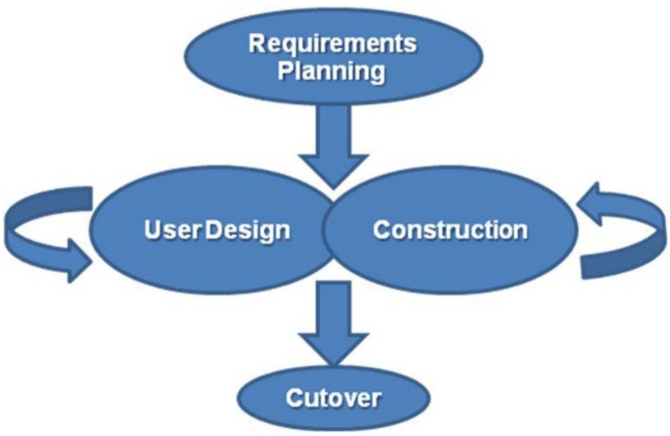


Figure 1.1 RAD Diagram [3].

The RAD model implemented in this thesis will consists of four major phases:

1. Requirement Planning Phase

This phase combines elements of the system planning and systems analysis phases of the Systems Development Life Cycle (SDLC) [3].

2. User Design Phase

During this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. This phase is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs [3].

3. Construction Phase

This phase focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing [3].

4. Cut Over Phase

This final phase resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner [3].

1.6 Thesis Outline

The thesis consists of seven chapters, which are as follow:

1. Chapter I: Introduction

   This chapter introduce the research background, problem, and objective. It also explains the research scope and limitation, method to achieve the objective.

2. Chapter II: Literature Study

   This chapter contain the literature study that related to the research background.

3. Chapter III: System Analysis

   This chapter explains the analysis of the application – both in its function and behaviour, in order to fulfil the prescribed requirements.

4. Chapter IV: System Design

   This chapter explains the system design of interfaces and class diagram based on the previous chapter that will be used in the next chapter.

5. Chapter V: System Development

   This chapter explains the system development of interfaces and code details on the application.

6. Chapter VI: System Testing

This chapter ensures the application system runs well by evaluating all the features, and making sure the system fulfils its function requirements.

7. Chapter VII: Conclusion and Future Work

This chapter sums up this research and also suggestion for future research work.

CHAPTER II

LITERATURE STUDY

2.1 Speech Synthesis

Speech synthesis is the process with the goal of building a system that can start with text and produce speech automatically [4]. There are many approaches to do speech synthesis, such as articulatory speech synthesizer, formant synthesis, and concatenative synthesis [5].

Articulatory speech synthesizer is production of speech sounds using vocal tract as a model, which directly or indirectly simulates the movements of the speech articulators. A properly constructed articulatory synthesizer is capable of reproducing all the naturally relevant effects for the generation of fricatives and plosives, modelling coarticulation transitions as well as source-tract interaction in a manner that resembles the physical process that occurs in real speech production [5].

Formant synthesis is based on a set of rules used to determine the parameters that necessary to synthesize a desired utterance. There are so many parameters that can be used, such as, voicing fundamental frequency and intensity of low and high frequency region [5].

Concatenative synthesis connecting pre-recorded natural utterances is probably the easiest way to produce intelligible and natural sounding synthetic speech. One of the most important aspects in concatenative synthesis is to find correct unit length. The selection is usually a trade-off between longer and shorter units. With longer units, high naturalness, less concatenation points and good control of coarticulation are achieved, but the number of required units and memory is increased. With shorter units, less memory is needed, but the sample collecting and labelling procedures become more difficult and complex.in present systems units used are usually words, syllables, demisyllables, phonemes, diphones, and sometimes even triphones [5].

As in mimic speech, the speech is taken from recognized speech, concatenative synthesis can be the best approach other than the other approach. Besides it is the easiest way rather than the others, it also quick to develop. Articulatory and Formant synthesis are too complex because in need a lot of parameter to develop the vocal tract or set of rules that can fit to many speakers.

Unfortunately, it is hard to find research regarding to exact amount of Bahasa Indonesia demisyllables, phonemes or smaller. Then, syllables are the best options for the speech unit as word need much more memory and less flexibility to generate speech in form of sentence.

2.2 Syllable

Syllable is a unit of pronunciation having one vowel sound, with or without surrounding consonants, forming the whole or a part of a word; for example, there are two syllables in water and three in inferno [6]. As in Bahasa Indonesia there are many rules to decoding a word to get the syllable. It constructed from 5 *vokal* (vowel), 21 *konsonan* (consonants), 4 *diftong* (diphthong), 4 *gabungan huruf konsonan* (cluster) [7].

There are 9 syllables that is used in this research, a, i, na, ma, mu, di, ri, ku, and kan. The syllables are randomly selected because it is hard to found research about syllable or even words that often be used in the daily life. But with those 9 syllables able to produce a lot of words. For examples, akan, ikan, nama, dimana, makan, dia, iri, kuakui, aku, diri, and so on.

2.3 Speech Recognition

Speech recognition is the process of converting a speech signal to a sequence of words, by means of an algorithm implemented as a computer program [8]. There are so many techniques and approaches [8, 9] to do speech recognition, it depends on the problem that going to be solved in optimum way. In mimic speech, as a speech is constructed by word and word constructed by syllables, convolutional neural network is one of optimum way to word spotting [10]. For small-footprint keyword, convolutional neural network has 27% improvement in performance other than deep neural network [10].

The way that computer understand the speech or signal is by sound sampling. Sound sampling is taking a reading thousands of times a second and recording a number representing the height of the sound wave at that point in time [11]. Nyquist sampling theorem provides a prescription for the nominal sampling interval required to avoid aliasing. The sampling frequency should be at least twice the highest frequency contained in the signal [12].

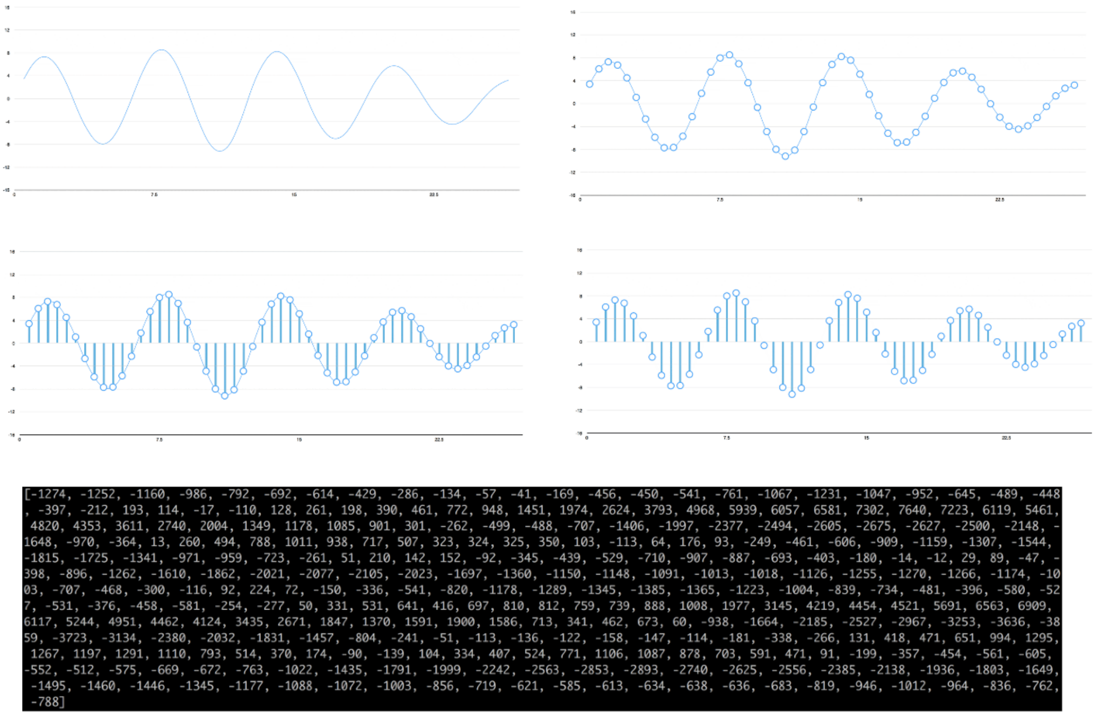


Figure 2.1 Sound sampling process [11].

Three extraction technique to get feature from a speech is Mel Frequency Cepstral Coefficients (MFCC), Linear Prediction Coefficients (LPC), and Perceptually Based Linear Predictive Analysis (PLP) [9].

Mel Frequency Cepstral Coefficients or MFCC is one of the most commonly used feature extraction method in speech recognition introduced by Davis and Mermelstein in the 1980s [13]. The technique is called FFT based which means that feature vectors are extracted from the frequency spectra of the windowed speech frames [9]. It aims to mimic the non-linear human ear perception of sound [14]. Linear

Prediction Coefficients or LPC analyses the speech signal by estimating the formants, removing their effects from the speech signal, and estimating the intensity and frequency of the remaining buzz. Perceptually Based Linear Predictive Analysis or PLP is analysis that is purposed by H. Hermansky, B. A. Hanson, and H. Wakita which models perceptually motivated auditory spectrum by a low all pole function, using the autocorrelation LP technique. It involves two major steps which is obtaining auditory spectrum and approximating the auditory spectrum by an all pole model. PLP analysis results demonstrated that speech representation is more consistent than the standard LP method [9].

In this research MFCC extraction is used. MFCC has advantage over LPC. It able to mimic human auditory system well. Although PLP also able to mimic human auditory system, MFCC is still be used due to its most common feature extraction. The technique is widely spread so that it easier to develop and debug.

2.4 Mel Frequency Cepstral Coefficients

The common steps to do MFCC are framing and windowing, Discrete Fourier Transform (DFT) and power spectrum, mel filterbank, logarithm, and finally Discrete Cosine Transform (DCT) before feeding to convolutional neural network. [9, 13, 14, 15]. Yet there can be variations can be made in process [15].

2.4.1 Framing and Windowing

A step can be done before framing and windowing is to apply a pre-emphasis filter on the signal to amplify the high frequencies. A pre-emphasis filter is useful in several ways such as, balance the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower frequencies, avoid numerical problems during the Fourier transform operation, and may also improve the Signal-to-Noise Ratio [14]. The pre-emphasis filter can be applied to a signal x using the first order filter in the following equation where typical values for the filter coefficient (α) are 0.95 or 0.97 [14]:

$$y(t) = x(t) - \alpha x(t-1) \tag{2.1}$$

Framing is done because of an audio signal is constantly changing, so to simplify things, assuming that on short time scales the audio signal does not change. Typically, signal is framing into 20-40ms frames (25ms is standard). If the frame is much shorter, it does not have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame [13]. Frame stripe typically is 10ms, which allows some overlap to the frames. If the speech file does not divide into an even number of frames, pad it with zeros so that it does [13]. This step is similar to pre-processing image in the convolutional neural network in the previous section.

After slicing the signal into frames, apply a window function such as the Hamming window to each frame can be done. Several reasons apply a window function to the frames, notably to counteract the assumption made by the FFT that the data is infinite and to reduce spectral leakage. Hamming window has the following form where, $0 \leq n \leq N-1$, N is the window length [14]:

$$w[n] = 0.54 - 0.46\, cos\left(\frac{2\pi n}{N-1}\right) \tag{2.2}$$

2.4.2 Discrete Fourier Transform and Power Spectrum

Compute each window with Discrete Fourier Transform (DFT) or Fast Fourier Transform (FFT) can be followed with compute power spectrum (periodogram). Motivated by the human cochlea which vibrates at different spots depending on the frequency of the incoming sounds. The periodogram estimate performs a similar job, identifying which frequencies are present in the frame [13]. Periodogram use the following equation where, x_i is the i^{th} frame of signal x and N is FFT size as a power of two greater than or equal to the number of samples in a single window length [14]:

$$P = \frac{|FFT(x_i)|^2}{N} \tag{2.3}$$

2.4.3 Mel Filterbank

The periodogram spectral estimate still contains a lot of information not required for speech recognition. In particular the cochlea cannot discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason, take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by mel filterbank [13].

Computing mel filterbank is applying triangular filters, typically 20 – 40 (26 or 40 is standard) filters, on a mel-scale to the power spectrum to extract frequency bands. Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminative at lower frequencies and less discriminative at higher frequencies. The formula to convert between Hertz (f) and Mel (m) using the following equations [14]:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.4}$$

$$f = 700\left(10^{\frac{m}{2595}} - 1\right) \tag{2.5}$$

Each filter in the filterbank is triangular having a response of 1 at the center frequency and decrease linearly towards 0 till it reaches the center frequencies of the two adjacent filters where the response is 0 [14]. Good values are to start filter from 300Hz for the lower and up to 8000Hz for the upper frequency [13]. The filterbank can be modelled by the following equation [14]:

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)}, & f(m-1) \leq k < f(m) \\ 1, & k = f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)}, & f(m) < k \leq f(m+1) \\ 0, & k > f(m+1) \end{cases} \tag{2.6}$$

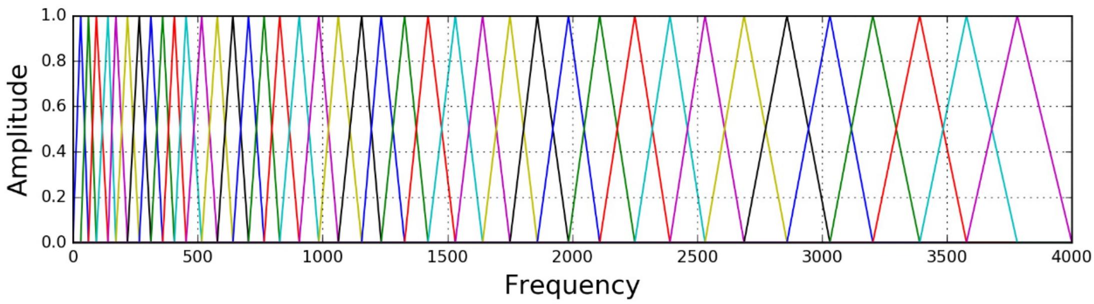


Figure 2.2 40 Filterbank from 0 Hz to 4000 [14].

2.4.4 Logarithm

Once compute the mel filterbank, the next is simply take the logarithm of them. This is also motivated by human hearing, as humans do not hear loudness on a linear scale. Generally, to double the perceived volume of a sound it needs to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression operation makes the features match more closely what humans actually hear [14].

2.4.5 Discrete Cosine Transform

The final step is to compute the Discrete Cosine Transform (DCT). There are 2 main reasons this is performed. It because the filterbanks are all overlapping and the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal covariance matrices can be used to model the features. But only 12 of the DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade speech recognition performance, so dropping them will get a small improvement [13].

2.5 Machine Learning

Machine learning is a form of AI that enables a system to learn from data rather than through explicit programming [16]. Others state that machine learning usually refers to the changes in systems that perform tasks associated with artificial intelligence (AI) [17].

In general, there are 2 types of machine learning, Supervised Learning and Unsupervised Learning [16, 17, 18].

2.5.1 Supervised Learning

In supervised learning, the algorithms are trained using pre-processed examples, and at this point, the performance of the algorithms is evaluated with test data. Occasionally, patterns that are identified in a subset of the data cannot be detected in the larger population of data. If the model is fit to only represent the patterns that exist in the training subset, it creates a problem called overfitting [16].

Overfitting means that the machine learning model is precisely tuned for the training data but may not be applicable for large sets of unknown data. To protect against overfitting, testing needs to be done against unforeseen or unknown labelled data. Using unforeseen data for the test set can help evaluate the accuracy of the model in predicting outcomes and results [16].

2.5.2 Unsupervised Learning

In unsupervised learning, the algorithms segment data into groups of examples (clusters) or groups of features. The unlabelled data creates the parameter values and classification of the data. In essence, this process adds labels to the data so that it becomes supervised [16].

Unsupervised learning can determine the outcome when there is a massive amount of data. Unsupervised can be used as the first step before passing the data to a supervised learning process if one does not know the context of the data being analysed, and labelling is not possible at the stage [16].

2.6 Neural Network

A neural network is an approach to machine learning in which small computational units are connected in a way that is inspired by connections in the brain [18]. Non-linear elements have as their inputs a weighted sum of the outputs of other elements, much like networks of biological neurons do [17].

Every neural network model is basically a three-layered system, which are Input layer, Hidden Layer and Output Layer [16, 18]. Input layer, is designed to receive information from the outside model. While the other side of the network, an output layer, communicates a decision about the data that has been received. Between these, other layers communicate information about elements of the input to each other, which contribute to the output [18].

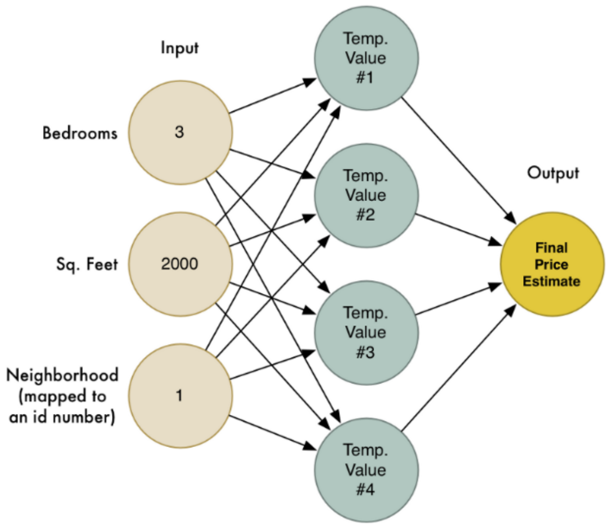


Figure 2.3 Neural network to find estimated price from specific input [19].

2.7 Convolutional Neural Network

Convolutional neural network (CNN or ConvNet) is one of known variants neural network model to recognized image [20]. As an image is just an image is really just a grid of numbers that represent how dark each pixel to a computer [19].

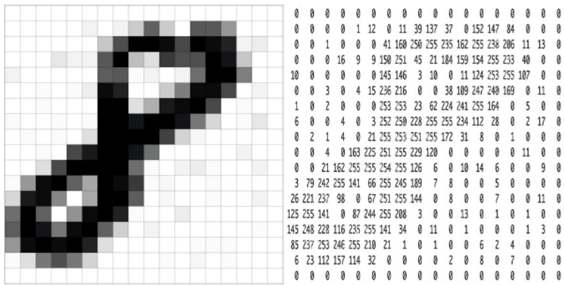


Figure 2.4 Computer read an image [19].

The model is designed to recognize an object no matter what surface the object is on. The model does not have to re-learn the idea of child for every possible surface it could appear on [19].

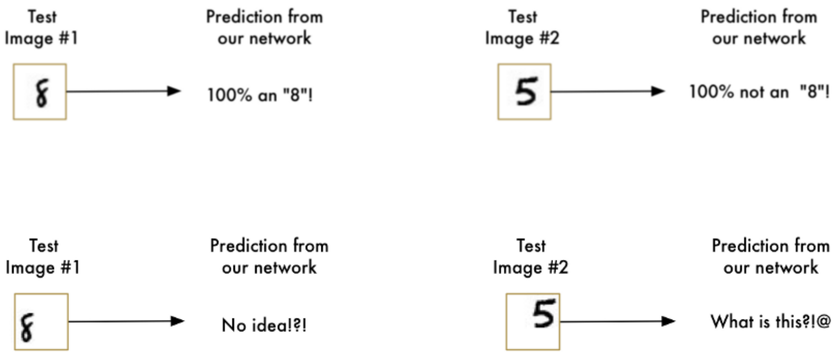


Figure 2.5 The basic neural network where it only recognizes the object on the center (top), but does not on another surface (bottom) [19].

The steps to applied convolutional neural network is the combination of convolution, max-pooling, and fully-connected layers. And yet, it might need a lot of experimentation and testing. Training a lot of model before find the optimal structure and parameters to solve the problem [19].

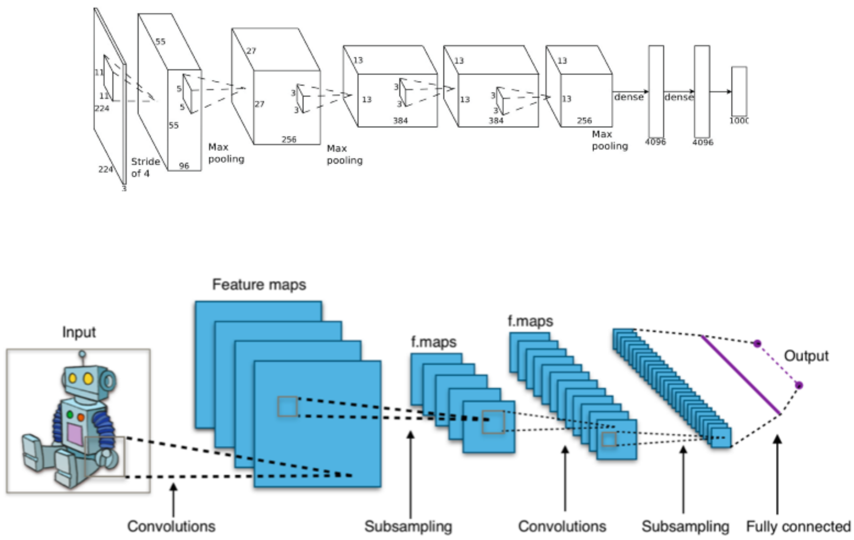


Figure 2.6 Two example of CNN for its own problem that trying to be solved [19, 20].

2.7.1 Pre-processing

Pre-processing is the preparation process by breaking the image into overlapping image tiles so that it can be feed into the model. Sliding window over the entire image will break the image into overlapping image tiles resulting equally sized tiny image tiles [19].

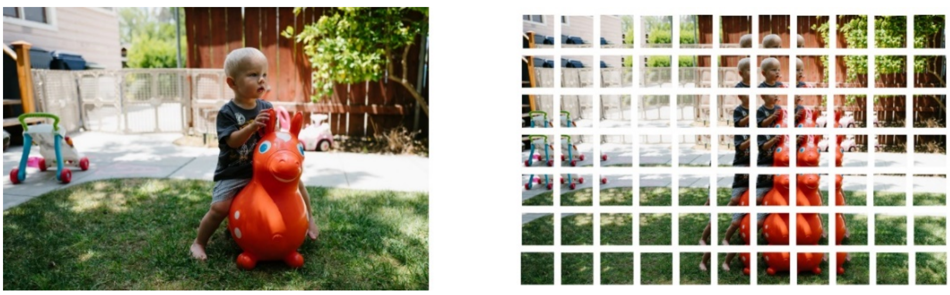


Figure 2.7 Before (left) and after (right) pre-processing image [19].

2.7.2 *Convolution Layer*

Convolution layer is the layer to feed the pre-processing image or another output into small neural network. The small neural network treating every image or output equally. It will mark if something interesting appears as the model learning [19].

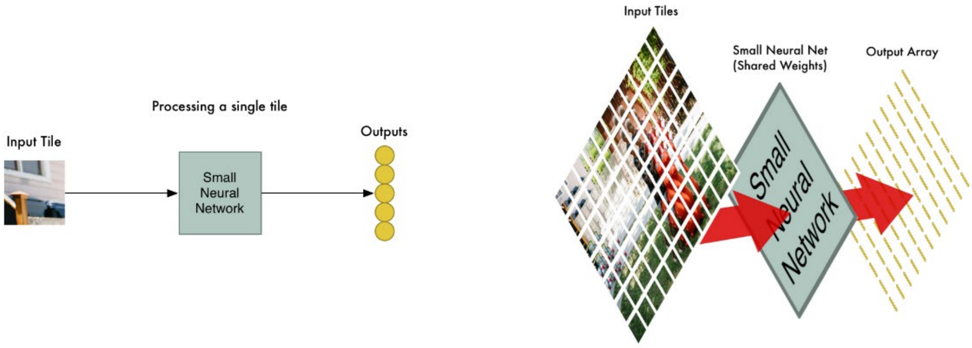


Figure 2.8 Convolution process on single tile (left) and overview all tiles (right) [19].

2.7.3 *Max-pooling Layer*

Max-pooling or down sampling is the layer to reducing the output by finding maximum value in the output. The process is similar like pre-processing in the previous section. The output broke down into equal pool size and stride or slide into entire output. Then, each pool is found the maximum value [19].

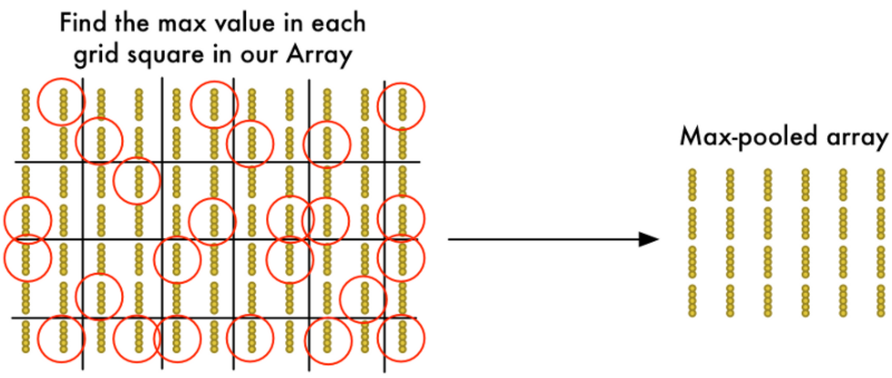


Figure 2.9 Max-pooling layer with 2x2 pool size and 2 strides [19].

2.7.4 Fully-connected Layer

Fully-connected is the layer to high-level reasoning in the dense neural network to recognize the image [20].

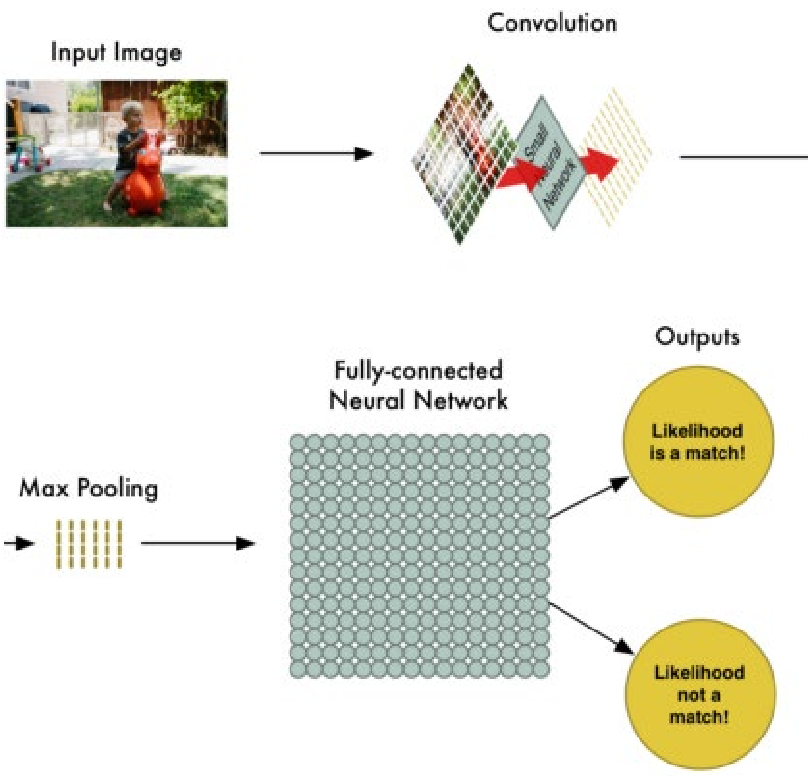


Figure 2.10 Fully-connected layer in CNN [19].

2.8 Related Work

The following are most related work to the research. Have relation to mimic speech, both speech recognition and speech synthesis.

2.8.1 Lyrebird

Lyrebird is website application, *https://lyrebird.ai*, that has 3 products: Custom Voice, Vocal Avatar, and Vocal Avatar API. Custom voice is a product to create speech based on artificial sound, it can control the intonation, expression, and the emotion of the speech. Vocal avatar is a product to create own digital speech by read some English sentences, and then generate any sentences with own digital speech. Vocal avatar API is a product to provide API to use user's own vocal avatar [21].

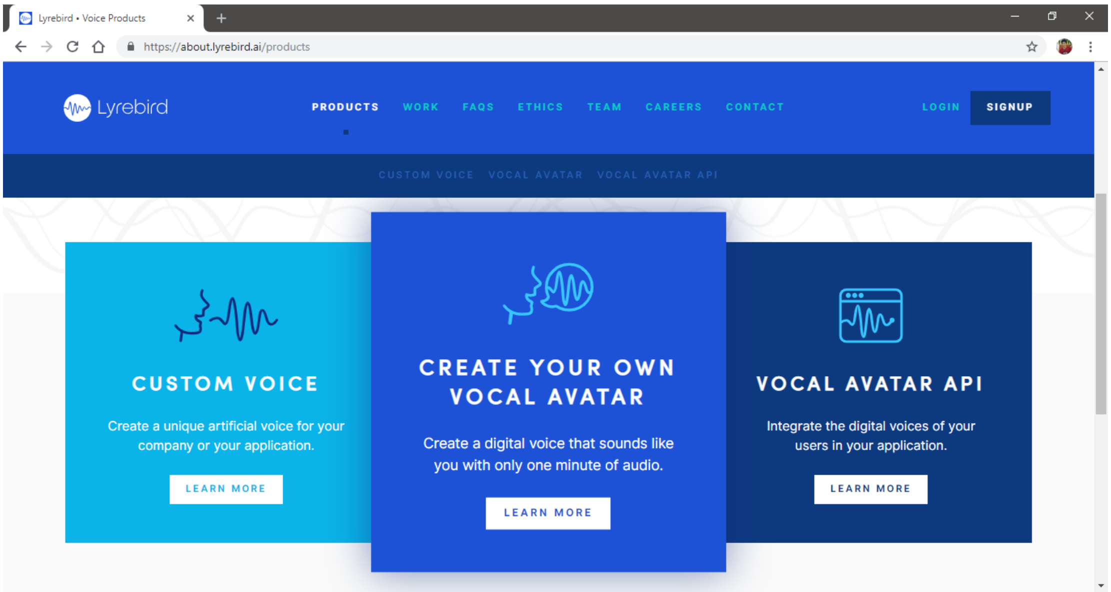


Figure 2.5 Screenshot of Lyrebird in the website [21].

2.8.2 Google Translate

Google Translate is one of Google products that is an application to translate languages. Google Translate can be access freely through *https://translate.google.com/*. It has many features and one of them is Talk feature. Talk has function to input text from speech and generate speech from text in any languages [22].

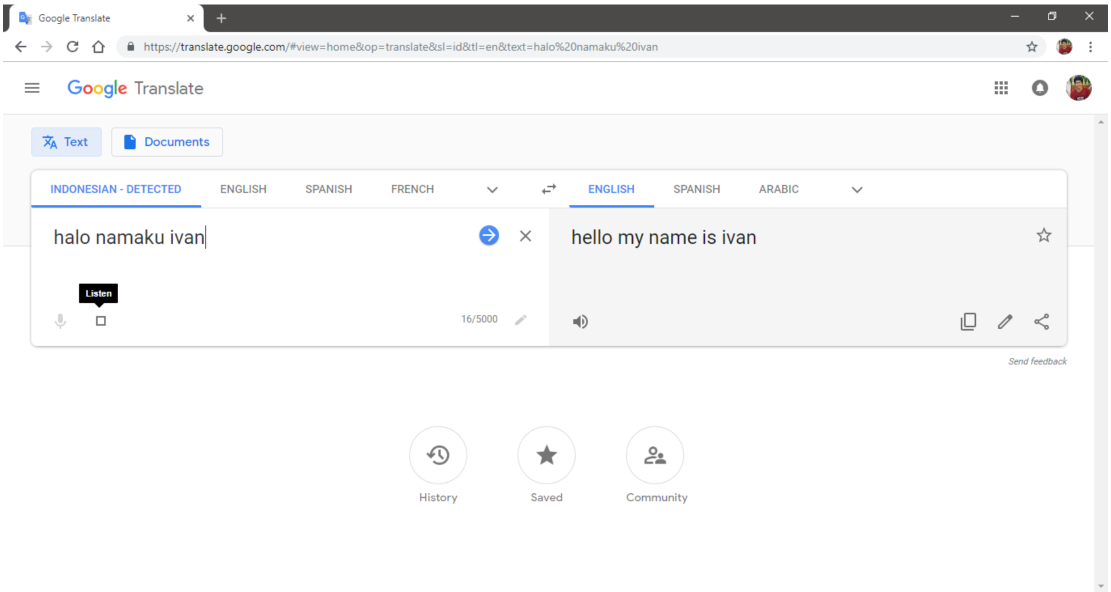


Figure 2.12 Screenshot of Google Translate in website [22].

From related work, it can be concluded that Lyrebird can mimic speech with its vocal avatar, but the speech is in English. In the other hand, Google Translate could mimic speech into any languages, but the speech vocal is from the Google Translate itself.

In this research, the application is developed more similar to Lyrebird's vocal avatar but the speech is in Bahasa Indonesia. That means the application will be able to recognize and generate speech with different speech vocal unlike Google Translate that the speech vocal is provided and cannot be change. Rather than read some English sentences like in vocal avatar, user create its own digital speech by record speech like Google Translate. But the speech based on available syllable, so that it can be saved. Then, by combining each syllables user can generate its speech based on inputted text by using saved speech. The following table sums up the difference between the application in this research with related work.

Table 2.1 Comparison Application.

Application	**Ability**					
	Recognize Speech	**Generate Speech**			**Speech Vocal**	
		In English	**In Bahasa Indonesia**	**Others**	**Artificial**	**Own Digital Speech**
Lyrebird	V	V	X	X	V	V
Google Translate	V	V	V	V	V	X
Mimic Speech (This Research)	V	X	V	X	X	V

Although this research has more constraints than the related works. This research is still conducted because it uses different language which is Bahasa Indonesia and it has simpler approach to do mimic speech rather than Lyrebird. This research also has different mechanism and purpose, that use recognize speech to make digital speech and generate speech to mimic speech from the digital speech, while Google Translate use recognize speech as input for translation process and generate speech to spell the texts.

CHAPTER III

SYSTEM ANALYSIS

3.1 System Overview

Based on the previous section, this research is intended to implement speech recognition and speech synthesis to produce mimic speech. This research consists of 3 application, Collect, Train, and Mimic application.

Collect application is used to collect speech data which are 9 syllables and random unknown sounds. User able to use it to participate their speech that later will be trained. Collect application develops as website application.

Train application train machine learning model with collected data for speech recognition. Convolutional neural network used as machine learning model for speech recognition and MFCC used as the feature extraction. The model later will be used to recognize user speech in the process to create its speech ID or digital speech. User just able to run the training command and wait until the training is complete. Train application develops as command prompt application.

Mimic application is used to mimic the speech. It identifies what user said with are trained model. Saves the speech data if the model recognizes. Then, it generates speech with saved speech data. Concatenative synthesis used as the speech synthesis technique. User able to create its speech ID by recognizing its speech based on available syllables and then generate its speech based on inputted text. Mimic application develops as websites application.

3.2 Functional Analysis

There are several functions for each application listed in the Table 3.1 up to 3.3.

Table 3.1 Collect application functionality table.

No	Function Description
1	Allow user to access Collect application.
2	Allow user to record user's speech.

Table 3.2 Train application functionality table.

No	Function Description
1	Allow user to train model with collected data.

Table 3.3 Mimic application functionality table.

No	Function Description
1	Allow user to access Mimic application
2	Allow user to identify user's speech.
3	Allow user to generate speech.

3.3 Software and System Requirements

This research and application development should be supported by the following list requirement in order to write the research, build and run the application well.

1. Laptop / Personal Computer

   Laptop or Personal Computer is used as the tool where operating system is run. In this research, ASUS A455LN is used with Windows 10 as the OS.

2. Browser

Browser is used as development tool when developing website application. In this research, Chrome v70.0.3538.110 is used.

3. Microsoft Office Word

Microsoft Office Word is used to write the research documentation. In this research, Microsoft Office Word 2016 is used.

4. Node.js, JavaScript Run-Time Environment.

Node.js is an open source server environment – Node.js is free – Node.js runs on various platform (Windows, Linux, Unix, Mac OS X, etc) – Node.js uses JavaScript on the server [23]. In this research, Node.js v10.14.1 is used.

5. NoSQL document-oriented database

Document databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents [24]. In this research, MongoDB Community Server 4.0.2 as database server and MongoDB Compass 1.15.4 as MongoDB UI.

6. Integrated Development Environment (IDE)

IDE is used as application development environment. In this research, Visual Studio Code 1.30.0 is used.

7. Git

   Git is used as version control. The repository is placed on local and cloud as preventive action. In this research, Git 2.18.0.windows.1 and GitLab as cloud repository is used.

3.4 System Architecture

The system architecture explains about the use-case diagram and narrative for this application in both point of view, the actors and the system.

3.4.1 Use-Case Diagram

Use-Case Diagram defines the functionality of a system and explains it in user point of view. The actors in this research is the application user. The diagram is shown in Figure 3.1.

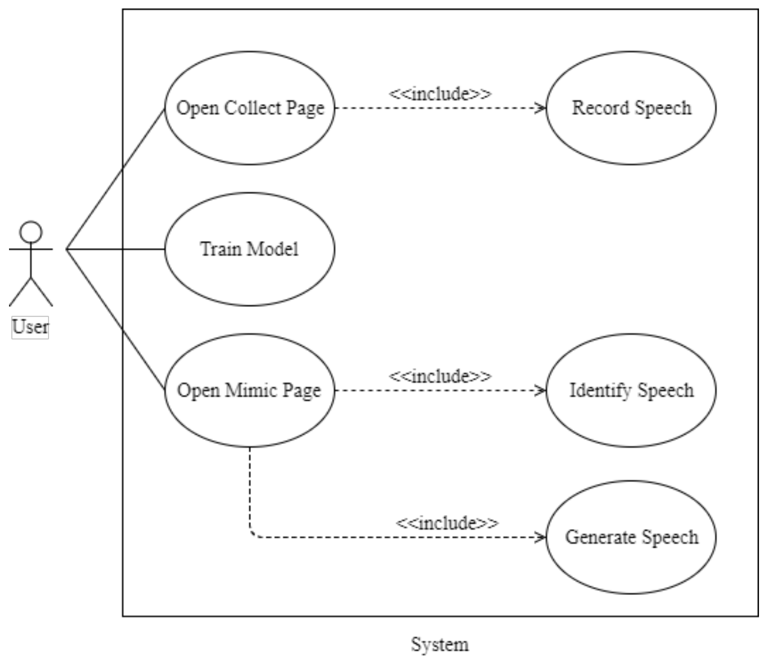


Figure 3.1 Use-case diagram.

3.4.2 Use-Case Narrative

Use-Case Narrative explains the interaction between the actors and the system. It describes the detail of use-cases such as name, description, pre-condition, post-condition, business rules, and the course of events that happened in the system. The Use-Case Narrative is shown in Table 3.4 up to Table 3.9.

Table 3.4 Use-case narrative – Open Collect page.

<table>
<tr><td>User Case Name</td><td colspan="2">Open Collect page</td></tr>
<tr><td>Use Case ID</td><td colspan="2">UC01</td></tr>
<tr><td>Priority</td><td colspan="2">High</td></tr>
<tr><td>Primary System Actor</td><td colspan="2">User</td></tr>
<tr><td>Another Participating Actor</td><td colspan="2">None</td></tr>
<tr><td>Description</td><td colspan="2">This use-case describes the event when user opens Collect page through browser.</td></tr>
<tr><td>Precondition</td><td colspan="2">None</td></tr>
<tr><td>Trigger</td><td colspan="2">User opens Collect page.</td></tr>
<tr><td rowspan="2">Typical Course of Event</td><td>Actor Action</td><td>System Response</td></tr>
<tr><td>Click Start button.</td><td>Collect Syllables page is shown.</td></tr>
<tr><td>Alternate Course</td><td colspan="2">None</td></tr>
<tr><td>Post Condition</td><td colspan="2">Collect Home page is shown.</td></tr>
<tr><td>Business Rule</td><td colspan="2">None</td></tr>
<tr><td>Implementation Constraint and Specifications</td><td colspan="2">None</td></tr>
</table>

Table 3.5 Use-case narrative – Record speech.

User Case Name	**Record speech**
Use Case ID	UC02
Priority	High
Primary System Actor	User
Another Participating Actor	None
Description	This use-case describes the event when user opens Collect Syllables page.
Precondition	User clicks Start button from Collect Home page.
Trigger	User opens Collect Syllables page.

Table 3.5 (continued).

Typical Course of Event	**Actor Action**	**System Response**
	Click Record button and say asked syllable.	Record the speech and upload to save it.
Alternate Course	**Actor Action**	**System Response**
	Click Next button.	Next Collect Syllables page or Collect Home page is shown.
Post Condition	Recorded speech is played on audio element.	
Business Rule	None	
Implementation Constraint and Specifications	None	

Table 3.6 Use-case narrative – Train model.

User Case Name	**Train model**	
Use Case ID	UC03	
Priority	High	
Primary System Actor	User	
Another Participating Actor	None	
Description	This use-case describes the event when user start Train application.	
Precondition	None	
Trigger	User starts Train application.	
Typical Course of Event	**Actor Action**	**System Response**
		Load and extract collected data.
		Split into train, validation, and test data.
		Convert data to model's input and output.
		Train the model with train data and validate with validation data.
		Test the model with test data and insert the model to MongoDB.
Alternate Course	None	
Post Condition	Model is trained and inserted in MongoDB.	

Table 3.6 (continued).

Business Rule	None.
Implementation Constraint and Specifications	None

Table 3.7 Use-case narrative – Open Mimic page.

User Case Name	**Open Mimic page**	
Use Case ID	UC04	
Priority	High	
Primary System Actor	User	
Another Participating Actor	None	
Description	This use-case describes the event when user opens Mimic page through browser.	
Precondition	None	
Trigger	User opens Mimic page.	
Typical Course of Event	**Actor Action**	**System Response**
	Click Identify Speech or Generate Speech button.	Mimic Identify page or Mimic Generate page is shown.
Alternate Course	None	
Post Condition	Mimic Home page is shown.	
Business Rule	None	
Implementation Constraint and Specifications	None	

Table 3.8 Use-case narrative – Identify speech.

User Case Name	**Identify speech**
Use Case ID	UC05
Priority	High
Primary System Actor	User
Another Participating Actor	None
Description	This use-case describes the event when user opens Mimic Identify page.
Precondition	User clicks Identify Speech button from Mimic Home page.
Trigger	User opens Mimic Identify page.

Table 3.8 (continued).

Typical Course of Event	**Actor Action**	**System Response**
	Click Record button and say something.	Record the speech and upload to save it.
		Recorded speech is played on audio element.
	Insert speech ID in input element and click Identify Speech button.	Recorded speech is identified with trained model.
		Speech data is updated to MongoDB if recognized.
Alternate Course	**Actor Action**	**System Response**
	Click Finish button.	Mimic Home page is shown.
Post Condition	Information whether recognized or not is alerted in the browser.	
Business Rule	None	
Implementation Constraint and Specifications	None	

Table 3.9 Use-case narrative – Generate speech.

User Case Name	**Identify speech**	
Use Case ID	UC06	
Priority	High	
Primary System Actor	User	
Another Participating Actor	None	
Description	This use-case describes the event when user opens Mimic Generate page.	
Precondition	User clicks Generate Speech button from Mimic Home page.	
Trigger	User opens Mimic Generate page.	
Typical Course of Event	**Actor Action**	**System Response**
		Load speech data from MongoDB.
	Choose speech ID from datalist element.	Speech ID selected.

Table 3.9 (continued).

	Insert text in input element and click Generate Speech button.	Speech is generated based selected speech ID and inserted text.
Alternate Course	**Actor Action**	**System Response**
	Click Finish button.	Mimic Home page is shown.
Post Condition	Generated speech is played on audio element.	
Business Rule	None	
Implementation Constraint and Specifications	None	

3.4.3 Activity Diagram

Activity Diagram presents a series of actions and flow of control in a system. The activity diagram is separated into each application.

3.4.3.1 Collect Application

This activity describes the Collect application which is start when user opens the application. Home page will be shown as the default page. As user clicks Start button, the Syllables page will be shown. User can record asked syllables by click Record button. Recorded speech is played in audio element. User can also change the next asked syllables by click Next button and the next Syllables page will be shown. If there are no more asked syllables, Home page will be shown.

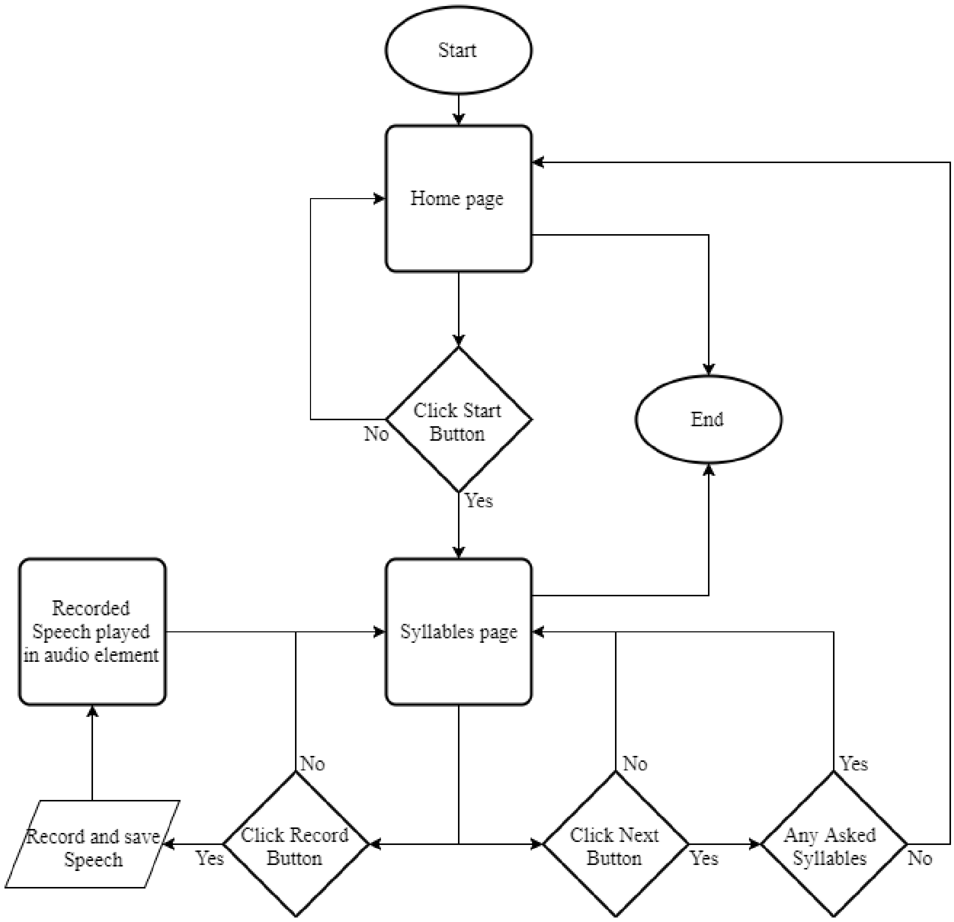


Figure 3.2 Collect application activity diagram.

3.4.3.2 Train Application

This activity describes the Train application which is start when user starts the application. Collected data will be loaded and extracted. Then, data split into train, validation, and test data. Each split data converted to model's input and output. The model is trained with train data and validated with validation data. Finally, the model is tested with test data and inserted to MongoDB. The model is trained and ready to use by Mimic application.

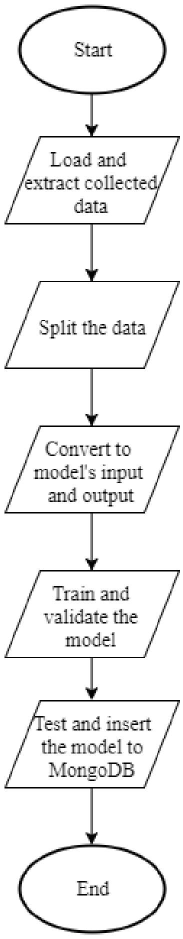


Figure 3.3 Train application activity diagram.

3.4.3.3 Mimic Application

This activity describes the Mimic application which is start when user opens the application. Home page will be shown as the default page. As user clicks Identify Speech button, the Identify page will be shown. User can record speech by click Record button. Then, user inserts speech ID on input element to name the speech data and click Identify Speech button to identify the recorded speech. Recognized speech will update the speech data to MongoDB. Any information regarding to identify process is alerted when finish to the browser. User can also click Finish button and Home page will be shown again. As user clicks Generate Speech button, the Generate page will be shown. Speech ID is loaded as shown as the page is loaded. User can choose speech ID. Inserts text on textarea element. And clicks Generate Speech button to generate speech based on the speech ID and the text. Generated speech is played in audio element. There's also Finish button in Generate page that also shown Home page when clicked.

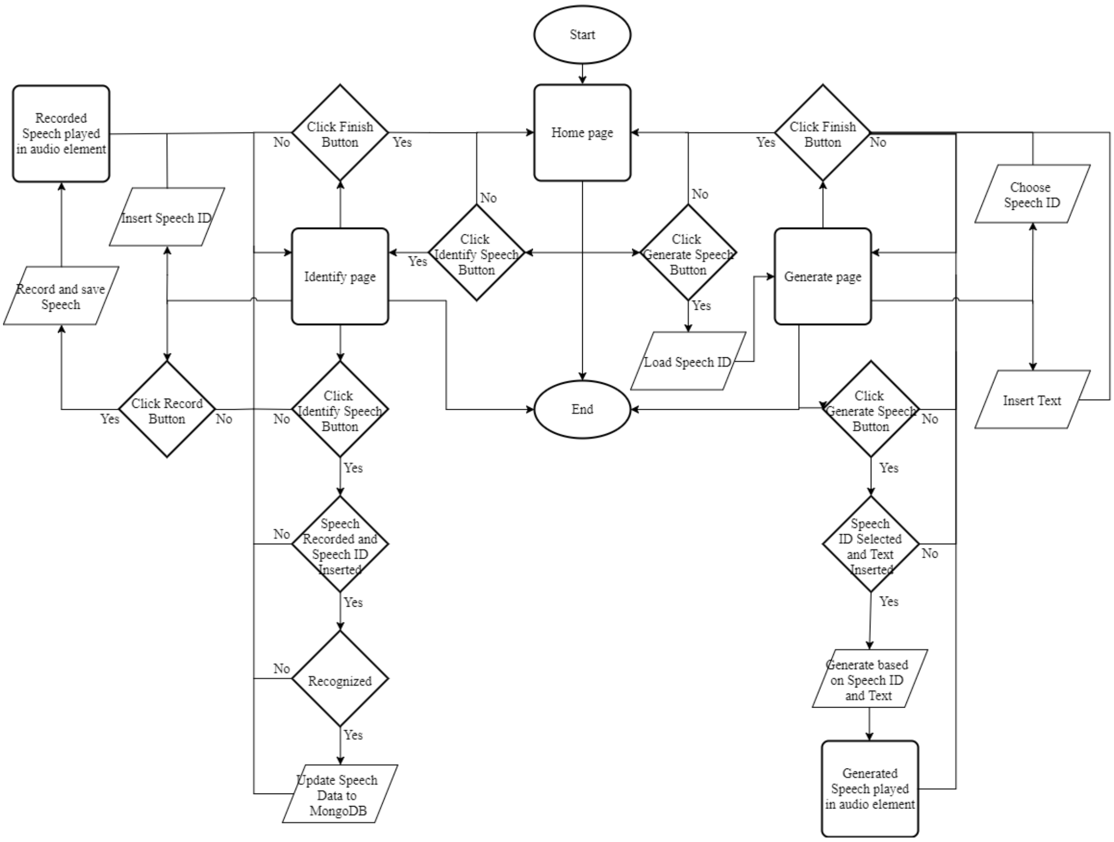


Figure 3.4 Mimic application activity diagram.

CHAPTER IV

SYSTEM DESIGN

4.1 User Interface Design

The UI is applied on Collect and Mimic application. There are two core pages in Collect application and three core pages in Mimic Application.

4.1.1 Collect Application

Two core pages in Collect application are Home page and Syllables page.

4.1.1.1 Home Page

Home page is the default page of the Collect application. When user opens the Collect application or directed from Mimic application or redirect back from Syllables page, it will show the Home page that consist of title, information regarding to Collect application, and 2 buttons which are Start and Mimic Home.

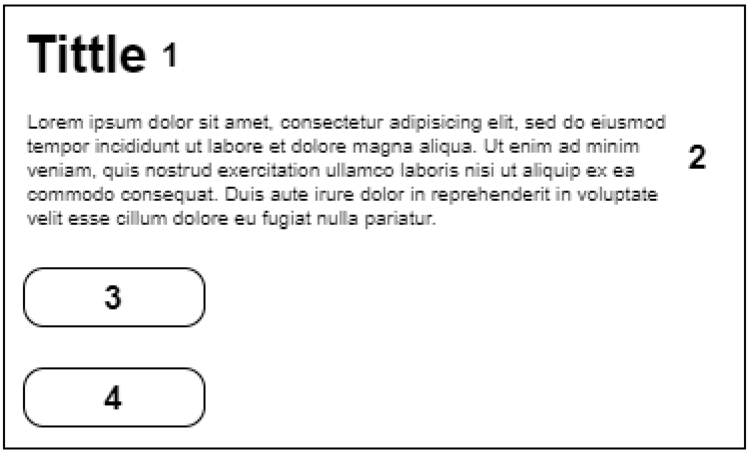


Figure 4.1 Collect Home page.

Table 4.1 Collect Home page description.

No	Description
1	Title
2	Information
3	Start Button
4	Mimic Home Button

4.1.1.2 Syllables Page

Syllables page is the main page of the Collect application. When user click Start in the Home page, it will show the Syllables page that consist of title regarding to which syllables should be recorded, audio element, and 2 buttons which are Record and Next.

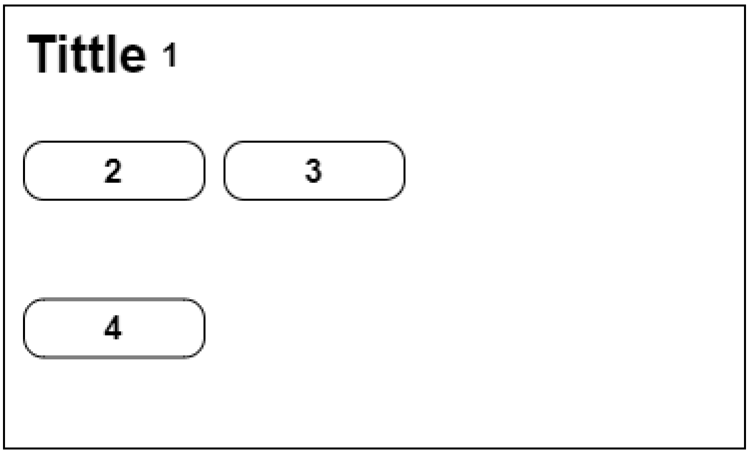


Figure 4.2 Collect Syllables page.

Table 4.2 Collect Syllables page description.

No	Description
1	Title
2	Audio Element
3	Record Button
4	Next Button

4.1.2 Mimic Application

Three core pages in Mimic application are Home page, Identify page, and Generate page.

4.1.2.1 Home Page

Home page is the default page of the Mimic application. When user opens the Mimic application or directed from Collect application or redirect back from Identify or Generate page, it will show the Home page that consist of title, information regarding to Mimic application, and 3 buttons which are Identify Speech, Generate Speech and Collect Home. The description of the design layout is shown in Table 4.3.

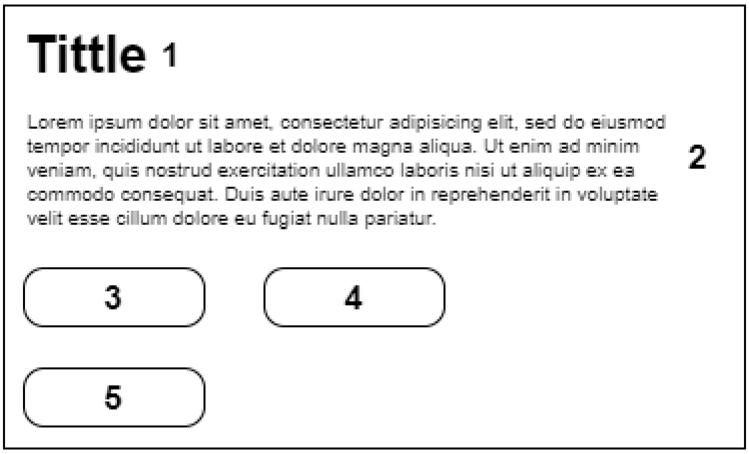


Figure 4.3 Mimic Home page.

Table 4.3 Mimic Home page description.

No	Description
1	Title
2	Information
3	Identify Speech Button
4	Generate Speech Button
5	Collect Home Button

4.1.2.2 Identify Page

Identify page is the one of the main pages of the Mimic application. When user click Identify Speech in the Home page, it will show the Identify page that consist of title, information registered syllables, 2 elements which are input and audio, and 3 buttons which are Record, Identify Speech, and Finish.

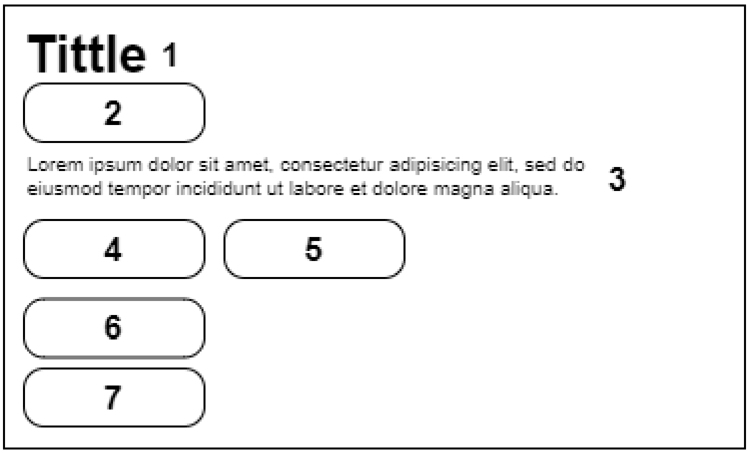


Figure 4.4 Mimic Identify page.

Table 4.4 Mimic Identify page description.

No	Description
1	Title
2	Input Element
3	Information
4	Audio Element
5	Record Button
6	Identify Button
7	Finish Button

4.1.2.3 Generate Page

Generate page is the other main pages of the Mimic application. When user click Generate Speech in the Home page, it will show the Generate page that consist of title, 3 elements which are datalist, textarea, and audio, and 2 buttons which are Generate Speech, and Finish.

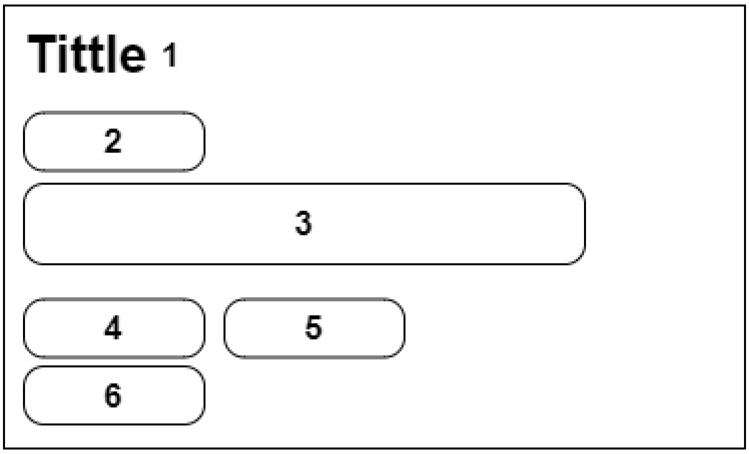


Figure 4.5 Mimic Generate page.

Table 4.5 Mimic Generate page description.

No	Description
1	Title
2	Datalist Element
3	Textarea Element
4	Audio Element
5	Generate Button
6	Finish Button

4.2 Database Design

The database in this research is design to be file structure storing files path that application would need. It will develop in MongoDB because the database scheme rigidity is not present. It also helps the development process easier and faster. The database has 2 collections, `models` and `speechDatas`.

4.2.1 models

`models` is the collection to store trained model's path. The model will be loaded and used to recognize speech later in the application. `models` has 2 main columns, `location` and `createAt`. `location` is used to store the path. It uses string as the datatype. `createAt` is used to store the date when the path is store. It uses date as the datatype and has current time as default value when the value is not defined.

4.2.2 speechDatas

`speechDatas` is the collection to store identified and recognized speech's path. Speeches that have same speech ID is stored in the same document. Because of that, `speechDatas` can be called a collection to store user's digital speech. `speechDatas` has 2 main columns, `name` and `syllables`. `name` is used to store the speech ID, identification to differentiate one digital speech to another. It uses string as the datatype. `syllables` is used to store the path. It uses object as the datatype. Each key in the object is named by syllable from identified speech and store the corresponding path.

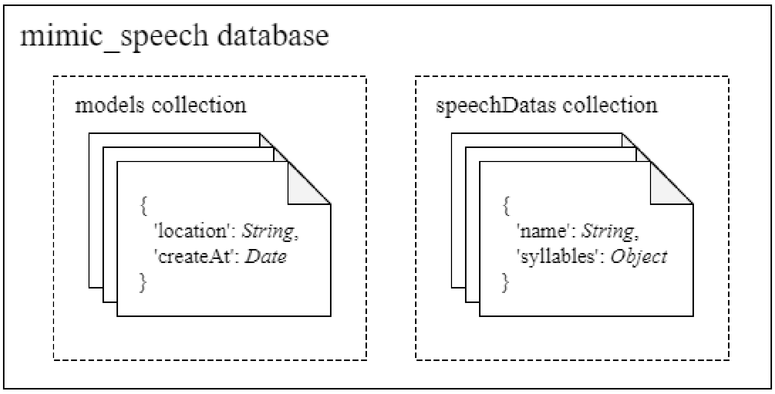


Figure 4.6 Collect application class diagram.

4.3 Class Diagram

The class diagram is the structure of the system used toward this research. The class diagram in this research consist of Collect, Train, and Mimic application, server that handle Collect and Mimic application, and database that handle connection to MongoDB.

4.3.1 Collect Application

The core classes in the Collect application are `home` and `syllables`. They are the front-end of Collect application.

4.3.1.1 home

`home` is HTML that render the Home page. It consists html tags based on the UI design on the previous section and JavaScript to handle the buttons.

4.3.1.2 syllables

`syllables` is HTML that render the Syllables page. It consists html tags based on the UI design on the previous section, JavaScript to handle the buttons, and JavaScript to handle the record that will be uploaded.

The upload handler contains `file`, `randomString`, and `XMLHttpRequest` methods. `file` is used to create wav blob from the record. `randomString` is used to randomize string for naming the record file and reducing duplicate that will overwrite one file to another. `XMLHttpRequest` is used to send upload request to the server and receive response from the server.

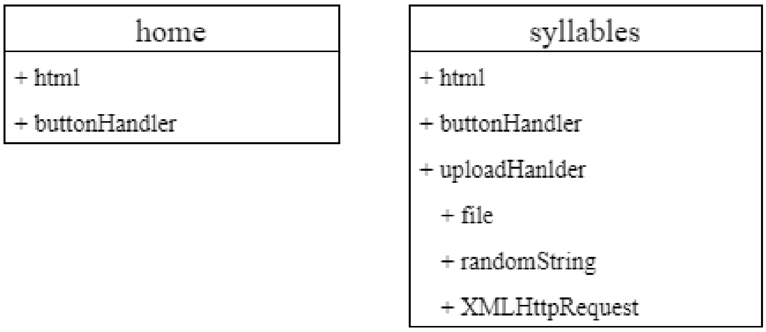


Figure 4.7 Collect application class diagram.

4.3.2 Train Application

The core classes in the Train application are `model` and `train`. They are the code to train the machine learning model with collected data.

4.3.2.1 model

`model` is JavaScript that define `model` to be Convolutional Neural Network model that will be used by `train`. It contains `add` method. `add` is used to add layer to the model.

4.3.2.2 train

`train` is JavaScript that setup and train the model based on `model`. It contains `loadData`, `nextBatch`, `saveModelDB`, `fit`, and `test` methods. `loadData` is used to load collected data from Collect application, split to train, validation, and test data, and extract all of them. `nextBatch` is used to get portion of `loadData` result and convert it to model input output. `saveModelDB` is used to insert the model to MongoDB. `fit` is used to train the model with train data and validate with validate data. `test` is used to test the model with test data.

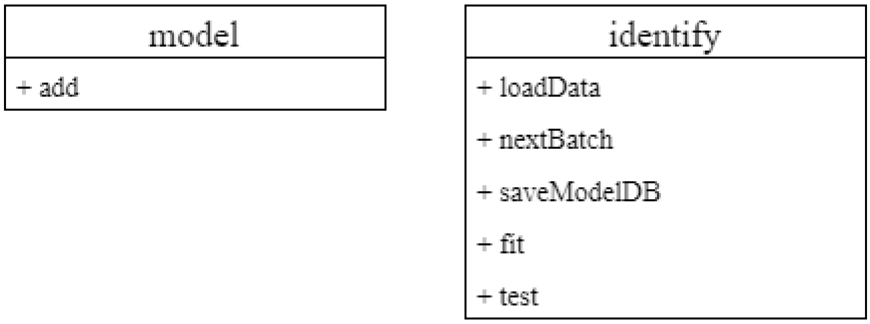


Figure 4.8 Train application class diagram.

4.3.3 Mimic Application

The core classes in the Mimic application are `home`, `identify`, and `generate`. They are the front-end of Mimic application.

4.3.3.1 home

`home` is HTML that render the Home page. It consists html tags based on the UI design on the previous section and JavaScript to handle the buttons.

4.3.3.2 identify

`identify` is HTML that render the Identify page. It consists html tags based on the UI design on the previous section, JavaScript to handle the elements and buttons, JavaScript to handle the record that will be uploaded, and JavaScript to handle the record that will be identified.

The upload handler contains `file`, `randomString`, and `XMLHttpRequest` methods. `file` is used to create wav blob from the record. `randomString` is used to randomize string for naming the record file and reducing duplicate that will overwrite one file to another. `XMLHttpRequest` is used to send upload request to the server and receive response from the server.

The identify handler contains `XMLHttpRequest` methods. `XMLHttpRequest` is used to send identify request to the server and receive response from the server.

4.3.3.3 generate

`generate` is HTML that render the Generate page. It consists html tags based on the UI design on the previous section, JavaScript to handle the elements and buttons, JavaScript to handle the speech data that will be loaded from database, and JavaScript to handle the speech ID and text that will be generated to audio.

The load handler contains `XMLHttpRequest` methods. `XMLHttpRequest` is used to send load request to the server and receive response from the server.

The generate handler contains `XMLHttpRequest` methods. `XMLHttpRequest` is used to send generate request to the server and receive response from the server.

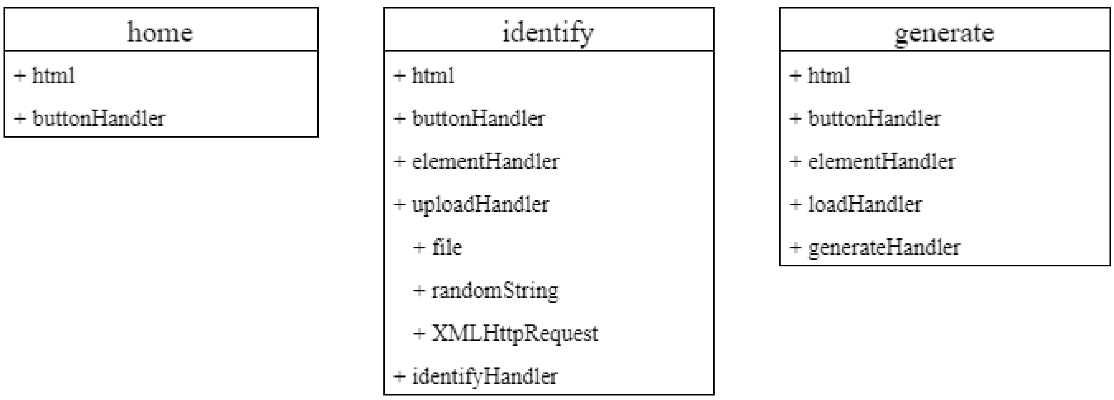


Figure 4.9 Mimic application class diagram.

4.3.4 Server

The core classes in the server are `serverHandler` and `createServer`. They are the back-end of Collect and Mimic application.

4.3.4.1 serverHandler

`serverHandler` is JavaScript that define server listener that will be used by `createServer`. It consists of JavaScript to handle upload from Collect application, JavaScript to handle upload and identify from Mimic Identify page, JavaScript to handle load and generate from Mimic Generate page, and JavaScript to handle URL that will display and load in the browser.

The collect handler contains `router` and `upload` methods. `router` is used to check whether the request is point to collect handler or not. If yes, then `upload` will be executed. `upload` is used to upload file based on the request. Then, convert its upload path location to URL. It returns the URL as response.

The identify handler contains `router`, `upload`, and `identify` methods. `router` is used to check whether the request is point to identify handler or not. If yes, then `upload` or `identify` will be executed. `upload` is used to upload recorded file based on the request, then convert its upload path location to URL. It returns the URL and upload path as response. `identify` is used to identify the recorded file based on the request. It loads the latest trained machine learning model from database. Then, the extract result is test with the model. If the test result show high accuracy the file path is inserted to MongoDB based on the speech ID in the request and the identify result. It returns the information whether the model can identify or not as response.

The generate handler contains `router`, `load`, and `generate` methods. `router` is used to check whether the request is point to generate handler or not. If yes, then `load` or `generate` will be executed. `load` is used to load all the speech ID from speech data in the MongoDB. It returns the result as response. `generate` is used to generate speech based on text from the request. It loads the speech data from speech ID based on the request. Then, the text is check with the loaded speech data. If unmatched is found, it returns error as response. If all matched, the speech data corresponding to text is combined to 1 audio file. Finally, the file path is converted to URL and returns the result as response

4.3.4.2 createServer

`createServer` is JavaScript that create the server. It contains `listen` method. `listen` is used to listen any request that comes to the server. The request will be proceeded to `serverHandler`.

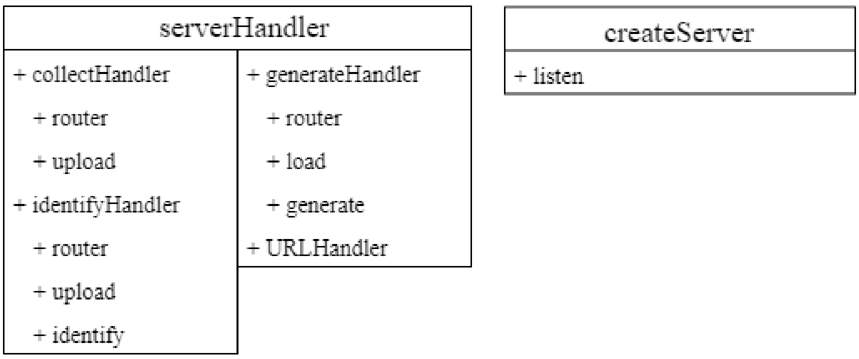


Figure 4.10 Server class diagram.

4.3.5 Database

The core classes in the database are `connection` and `model`. They are the code that create connection to MongoDB and collection model scheme of the database on MongoDB.

4.3.5.1 connection

`connection` is JavaScript that handle the connection with MongoDB. It contains `connect` and `disconnect` methods. `connect` is used to establish the connection to MongoDB. `disconnect` is used to close the established connection.

4.3.5.2 model

`model` is JavaScript that create the collection model scheme of the database on MongoDB. It contains `speechDatas` and `models` schemes which same as the database collection. `speechDatas` is used to define model for speech data collection scheme that is used on Mimic application. Each collection record speech ID and syllables. `models` is used to define model for machine learning model collection scheme that is used on Train and Mimic application. Each collection record model's path and the date the model is saved.

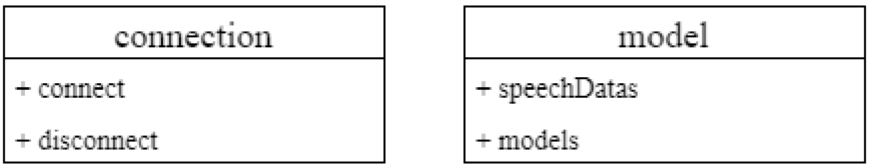


Figure 4.11 Database class diagram.

CHAPTER V

SYSTEM DEVELOPMENT

5.1 User Interface Development

The UI is applied on Collect and Mimic application. There are two core pages in Collect application and three core pages in Mimic Application.

5.1.1 Collect Application

Two core pages in Collect application are Home page and Syllables page.

5.1.1.1 Home Page

Home page is the default page of the Collect application. It shows information about how the collecting process works and information regarding to the collecting process. There are also 2 buttons, Start and Mimic Home button.

Start button will direct user to Syllables page to start collecting. Mimic Home button will direct user to Mimic application Home page.

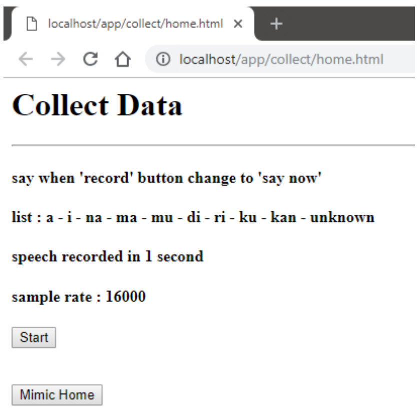


Figure 5.1 Collect Home page.

5.1.1.2 Syllables Page

Syllables page is the main page of the Collect application. It shows information about which syllable should be recorded and replayed the record. There are also 2 buttons, Record and Next button and an audio element. Any response from the server is also printed in the console.

Record button will start the record process. When the record process is starting, Record button will be renamed to 'Say Now' which tell user that the record process is starting and indicate user to say the relevant syllable. When the record process is finish, the recorded blob is uploaded to the server and Record button will be renamed to 'Record Again' which tell user that the record process is finish and user can do the next record. Next button will direct user to the next Syllables page. In this application there are 9 Syllables and 1 unknown Syllable. On the last syllables, Next button will direct user back to Home page.

Audio element will be shown when the record process is finish with an audio from the record. User can hear the replay of the record from the audio element.

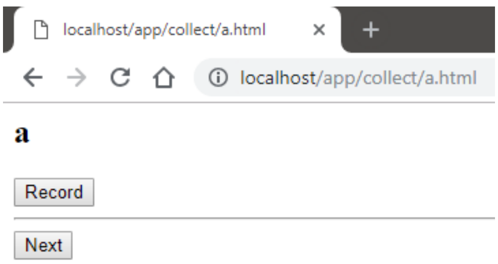


Figure 5.2 Collect Syllables page with 'a' syllable.

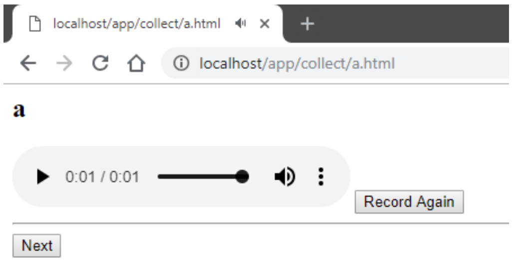


Figure 5.3 Collect Syllables page after record process is finish.

5.1.2 Mimic Application

Three core pages in Mimic application are Home page, Identify page, and Generate page.

5.1.2.1 Home Page

Home page is the default page of the Mimic application. It shows information about how the mimic process works and information regarding to the identify and generate process. There are also 3 buttons, Identify, Generate, and Collect Home button.

Identify button will direct user to Identify page to start identifying speech. Generate button will direct user to Generate page to start generating speech. Collect Home button will direct user to Collect application Home page.

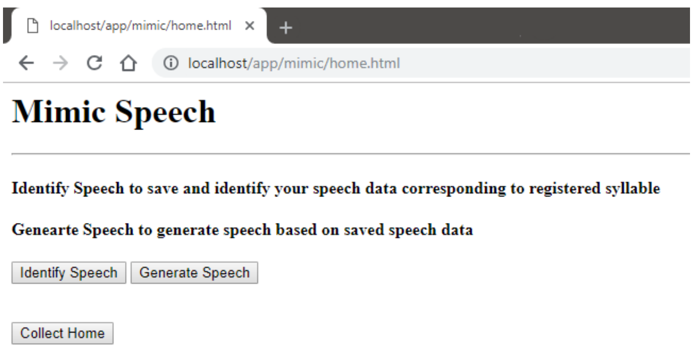


Figure 5.4 Mimic Home page.

5.1.2.2 Identify Page

Identify page is the one of the main pages of the Mimic application. It shows information about which syllable is recognizable and form of input and audio that will be used to identify the speech and update it to database. There are also 3 buttons, Record, Identify Speech and Finish button, and an input and an audio element. Any response from the server is also printed in the console.

Record button will start the record process. When the record process is starting, Record button will be renamed to ‘Say Now’ which tell user that the record process is starting and indicate user to say the syllable. When the record process is finish, the recorded blob is uploaded to the server Record button will be renamed to ‘Record Again’ which tell user that the record process is finish and user can do the next record. Identify Speech button will take the value of input element and file source of audio element, then, send them to the server to be identify. The indication of identify process in the server is success or error is there is alert on the browser that show any information regarding to identify process in the server. If there is no value of input element or file source of audio element, Identify Speech button will alert user that there is not any value. Finish button will direct user back to the Home page.

Audio element will be shown when the record process is finish with an audio from the record. User can hear the replay of the record from the audio element. Input element is used as input of user speech ID value.

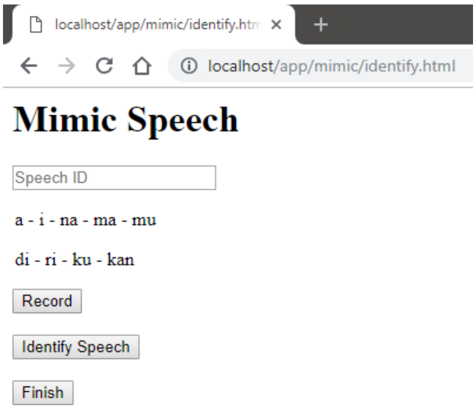


Figure 5.5 Mimic Identify page.

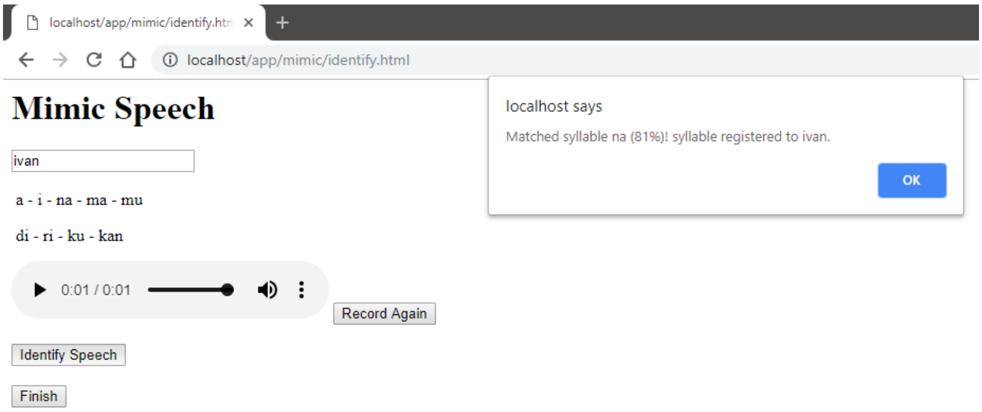


Figure 5.6 Mimic Identify page alert user indicating identify process in the server is finish.

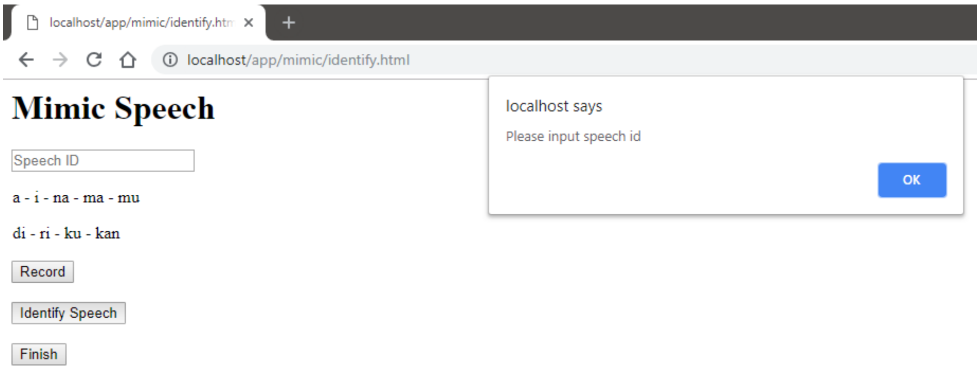


Figure 5.7 Mimic Identify page alert user indicating there is no value in input element.

5.1.2.3 Generate Page

Generate page is the one of the main pages of the Mimic application. It shows form of data list and text area that will be used to generate the speech. When the page is loading, it also loads speech data from the database and then store it on data list. If there is no speech data found, it will alert user and direct user back to Home page. There are also 2 buttons, Generate Speech and Finish button, and a datalist element, a textarea element and an audio element. Any response from the server is also printed in the console.

Generate Speech button will take the value of datalist element and textarea element, then, send them to the server to be generate. The indication of generate process in the server is error is there is alert on the browser that show any error information regarding to generate process in the server. The indication of generate process in the server is success is the audio element is shown with an audio from the generated speech. If there is no value of datalist element or textarea element, Identify Speech button will alert user that there is not any value. Finish button will direct user back to the Home page.

Audio element will be shown when the generate process is success with an audio from the generated speech. User can hear the generated speech from the audio element. Datalist element is shows all speech data from database and it is used as input of user speech ID value. Textarea element is used as input of words that user want to generate.

Figure 5.8 Mimic Generate page.

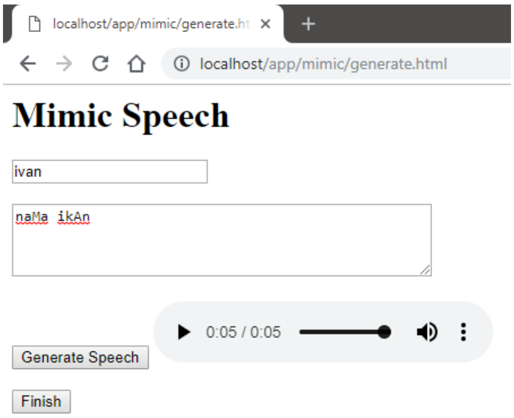


Figure 5.9 Mimic Generate page play the generated speech after generate process success.

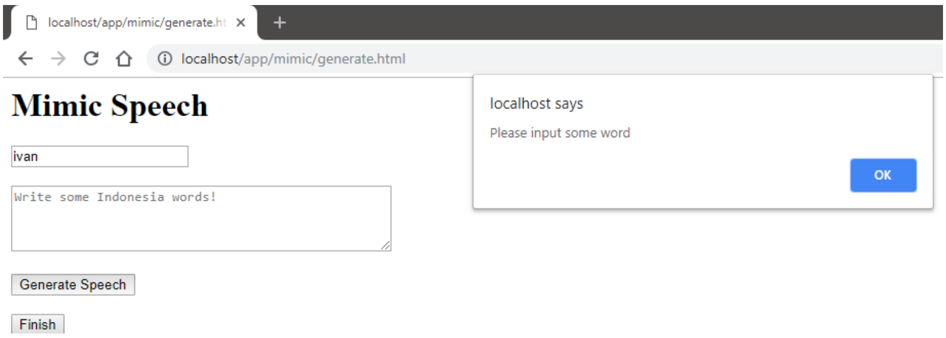


Figure 5.10 Mimic Generate page alert user indicating there is no value in textarea element.

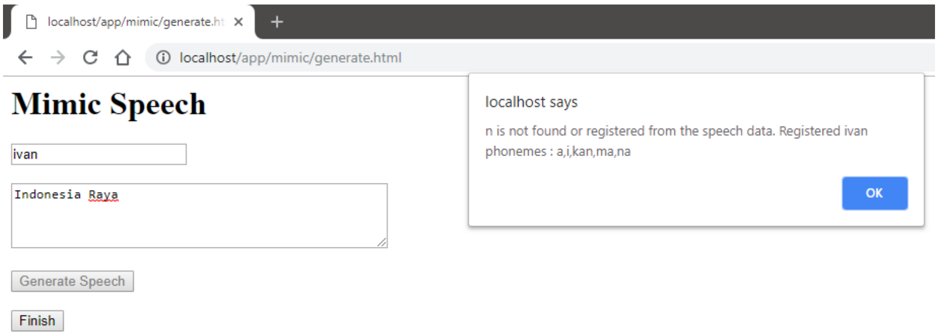


Figure 5.11 Mimic Generate page alert user indicating there is error in the generate process.

5.2 Application Details

The application details explain some codes on Collect, Train, and Mimic application, server and database code based on previous chapter.

5.2.1 Collect Application

Collect application code is used to create the front-end of Collect application. It consists of `home` and `syllables`.

5.2.1.1 home

`home` is used to render the Home page as shown in Figure 5.1 and also handle the button in Home page. It is written on HTML and JavaScript.

`home` contains `html`, `btnStart`, and `btnMimic`. `html` is used to render Home page. `btnStart` is used to direct user to Syllables page. `btnMimic` is used to direct user to Mimic Home page.

home.html

```html
<html>

<body>
  <h1> Collect Data </h1>
  <hr />
  <h4> say when 'record' button change to 'say now' </h4>
  <h4> list : a - i - na - ma - mu - di - ri - ku - kan - unknown </h4>
  <h4> speech recorded in 1 second </h4>
  <h4> sample rate : 16000 </h4>
  <button id='btnStart'>Start</button>
  <br />
  <br />
  <br />
  <button id='btnMimic'>Mimic Home</button></Link>

  <script src='./script/home.js'></script>
</body>

</html>
```

Figure 5.12 Collect home html code.

5.2.1.2 syllables

`syllables` is used to render the Syllables page as shown in Figure 5.2 up to 5.3, handle the button in Syllables page, and also handle upload record. It is written on HTML and JavaScript. The upload handler, that contains `file`, `randomString`, and `XMLHttpRequest`, is forked from RecordRTC package. RecordRTC is a JavaScript-based media-recording library for modern web-browsers. It is optimized for different devices and browsers to bring all client-side recording solutions in single place [25].

`syllables` contains `html`, `btnNext`, `btnRecord`, `captureUserMedia`, `postFiles`, `generateRandomString`, and `xhrPostUploadCollect`. `html` is used to render Syllables page based on corresponding syllable. `btnNext` is used to direct user to the next Syllables page or redirect back to Home page if there is not any next syllable.

`btnRecord` is used to start record. It runs `captureUserMedia` first, that is used to ask user's microphone permission. If the permission granted, it then starts recording for 1 seconds. After that it runs `postFiles`, that is used to store the wav blob from the record and generate random string for naming the file files from `generateRandomString`. Finally, `xhrPostUploadCollect`, that is used to send upload request based on corresponding syllable with the wav blob to the server and receive response from the server as the upload result, is run.

upload.js

```
btnRecord.onclick = function () {
  captureUserMedia(function (stream) {
    btnRecord.disabled = true;
    mediaStream = stream;

    recorder = RecordRTC(mediaStream, {
      mimeType: 'audio/wav',
      recorderType: StereoAudioRecorder,
      numberOfAudioChannels: 1,
      desiredSampRate: 16000,
      onAudioProcessStarted: function () {
        btnRecord.innerHTML = 'Say Now';

        setTimeout(function () {
          btnRecord.disabled = false;
          btnRecord.innerHTML = 'Record Again';

          recorder.stopRecording(postFiles);
        }, 1000);
      }
    });

    recorder.startRecording();
  });
};
```

Figure 5.13 Collect syllables btnRecord code.

5.2.2 *Train Application*

Train application code is used to train the machine learning model with collected data. It consists of `model` and `train`. It is developed using TensorFlow.js.

TensorFlow is a JavaScript library for training and deploying ML models in the browser and on Node.js [26].

5.2.2.1 model

`model` is used to define `model` to be Convolutional Neural Network model that will be used by `train`. It is written on JavaScript. The model is inspired from TensorFlow speech_commands [27] and forked from TensorFlow.js speech_commands [28].

`model` contains `modelML`. `modelML` is used to contain defining process. As TensorFlow.js imported, use `sequential` to define model as a stack like. After that, add layer by simply use `add` and followed by the name of the layer. Finally, use `compile` to configures and prepares the model for training and evaluation.

model.js

```
function modelML(input, output) {
  const model = tf.sequential();
  model.add(tf.layers.conv2d({ inputShape: input, filters: 8, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 2], strides: [2, 1] }));
  model.add(tf.layers.conv2d({ filters: 32, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 2], strides: [2, 1] }));
  model.add(tf.layers.conv2d({ filters: 32, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 2], strides: [2, 2] }));
  model.add(tf.layers.conv2d({ filters: 32, kernelSize: [4, 2], activation: 'relu' }));
  model.add(tf.layers.maxPooling2d({ poolSize: [2, 2], strides: [1, 2] }));
  model.add(tf.layers.flatten({}));
  model.add(tf.layers.dropout({ rate: 0.25 }));
  model.add(tf.layers.dense({ units: 2000, activation: 'relu' }));
  model.add(tf.layers.dropout({ rate: 0.5 }));
  model.add(tf.layers.dense({ units: output, activation: 'softmax' }));

  model.compile({
    loss: 'categoricalCrossentropy',
    optimizer: tf.train.sgd(0.01),
    metrics: ['accuracy']
  });

  return model;
}
```

Figure 5.14 Train model modelML code.

5.2.2.2 train

`train` is used to setup and train the model based on `model`. It is written on JavaScript. The code is inspired from TensorFlow speech_commands [27] and TensorFlow.js speech_commands [28].

`train` contains `extractWav`, `createMelFilterbank`, `loadData`, `nextBatch`, `train`, and `saveModelDB`. `loadData` is used to load collected data from Collect application, split to train, validation, and test data, and run `extractWav` to extract all of them. `extractWav` is used MFCC to do extraction process. `createMelFilterbank` is used to create the filterbank model that is used by `extractWav`. The extraction is inspired from many sources, python_speech_features [29], node-mfcc [30], and meyda [31], as there are many variations in the process. In the process node-wav package [32] is used to decode the collected file in the beginning of extraction process. Also, dct package [33] is used to calculate DCT in the last step of extraction process.

As there are so many variations in MFCC extraction, in this MFCC extraction does not include pre-emphasis, windowing, and power spectrum although the step is already been coded so that one could debug or explore in MFCC extraction. Pre-emphasis is not used because it is not required and avoiding reducing features on low signal. Windowing and power spectrum also are not used because those steps make the sample add more negative exponents of 10. It makes the features have to many trailing zeros and it makes the machine learning hard to trained because of the small value.

When the data is loaded, it is shuffled by shuffle-array package [34] to reduce overfitting. Every iteration when extracting the loaded files is printed. If inconsistent data is met, the process stops and exit. Then, it loads `model` first, before run `nextBatch`, that is used to get portion of `loadData` result and convert it to model input output. It converts test data to be prepared for testing. After that `train` is run. At first, it converts train and validation data using `nextBatch` corresponding to train iteration. Then, it runs `fit`, that is provided by TensorFlow.js to train using train data and evaluate using validation data. `fit` is iterate based on the epoch that is defined.

Every train iteration evaluation result is printed. Test is run every n time in train iteration and last iteration. It uses predict to `predict` the test data and `confusionMatrix` to print all the test result. Both is provided by TensorFlow.js. Every test iteration the model saved with random name to avoid overwrite using crypto-random-string package [35]. It also run `saveModelDB`, that is used to insert the model to MongoDB.

```
extractWav.js
// calculate mel filterbank and dot product with the sample.
let filSample = [];
let fbank = require('./createMelFilterbank')(lowFreq, highFreq, melFilter, fftSize, sampleRate);
let fbankTranspose = [];
for (i = 0; i < fbank[0].length; i++) {
  let filter = new Float32Array(fbank.length);
  for (j = 0; j < fbank.length; j++) {
    filter[j] = fbank[j][i];
  }
  fbankTranspose.push(filter);
}
for (i = 0; i < powSample.length; i++) {
  let frame = new Float32Array(fbankTranspose[0].length);;
  for (j = 0; j < fbankTranspose[0].length; j++) {
    for (k = 0; k < powSample[i].length; k++) {
      frame[j] += powSample[i][k] * fbankTranspose[k][j];
    }
  }
  filSample.push(frame);
}

// logarithma the sample.
let logSample = [];
logSample = filSample.map(frame => frame.map(x => Math.log(1 + x)));
```

Figure 5.15 Some part of Train train extractWav code.

5.2.3 *Mimic Application*

Mimic application code is used to create the front-end of Mimic application. It consists of `home`, `identify`, and `generate`.

5.2.3.1 home

`home` is used to render the Home page as shown in Figure 5.4 and also handle the button in Home page. It is written on HTML and JavaScript.

`home` contains `html`, `btnIdentify`, `btnGenerate`, and `btnCollect`. `html` is used to render Home page. `btnIdentify` is used to direct user to Identify page. `btnGenerate` is used to direct user to Generate page. `btnCollect` is used to direct user to Collect Home page.

JS home.js ×

```
const btnIdentify = document.querySelector('#btnIdentify');
const btnGenerate = document.querySelector('#btnGenerate');
const btnCollect = document.querySelector('#btnCollect');

btnIdentify.onclick = function () {
    open('/app/mimic/identify.html', '_self');
};

btnGenerate.onclick = function () {
    open('/app/mimic/generate.html', '_self');
};

btnCollect.onclick = function () {
    open('/app/collect/home.html', '_self');
};
```

Figure 5.16 Mimic home html code.

5.2.3.2 identify

`identify` is used to render the Identify page as shown in Figure 5.5 up to 5.7, handle the button and element in Identify page, and also handle upload and identify record. It is written on HTML and JavaScript.

`identify` contains `html`, `inSpeechId`, `btnFinish`, `btnRecord`, `btnIdentify`, `captureUserMedia`, `postFiles`, `generateRandomString`, `xhrPostUploadCollect`, and `xhrPostIdentifySpeech`. `html` is used to render Identify page. `inSpeechId` is used to place for user to input speech ID. `btnFinish` is used to redirect back to Home page.

`btnRecord` is used to start record same like Collect `syllables` `btnRecord`. `btnIdentify` is used to identify recorded speech. It alerts error to user if there is not any value in `inSpeechId` and recorded speech. It identifies by run `xhrPostIdentifySpeech`, that is used to send identify request with `inSpeechId` and recorded speech to the server and receive response from the server.

identify.js

```
function xhrPostIdentifySpeech(data) {
  const request = new XMLHttpRequest();
  request.onreadystatechange = function () {
    if (request.readyState == 4 && request.status == 200) {
      const status = JSON.parse(request.responseText).status;

      console.info('status', status);
      alert(status);

      btnRecord.disabled = false;
      btnIdentify.disabled = false;
    }
  };
  request.open('POST', '/identifySpeech');

  const formData = new FormData();
  formData.append('name', data.name);
  formData.append('filePath', data.filePath);
  request.send(formData);
}
```

Figure 5.17 Mimic identify xhrPostIdentifySpeech code.

5.2.3.3 generate

`generate` is used to render the Generate page as shown in Figure 5.8 up to 5.11, handle the button and element in Generate page, and also handle load and generate speech. It is written on HTML and JavaScript.

`generate` contains `html`, `inSpeechId`, `listSpeechId`, `txtAreaText`, `btnFinish`, `btnGenerate`, `xhrGetSpeechId`, and `xhrPostGenerateSpeech`. `html` is used to render Generate page. After render the page, it runs `xhrGetSpeechId`, that is used to send load request to the server and receive response from the server. The loads result which is speech data is stored to `listSpeechId`. If there is no speech data found, it alerts error to user and redirect user back to Home page.

`inSpeechId` is used to place for user to input speech ID that also show speech data list. `txtAreaText` is used to place for user to input text. `btnFinish` is used to redirect back to Home page. `btnGenerate` is used to generate speech. It alerts error to user if there is not any value in `inSpeechId` and `txtAreaText` or inputted speech ID in `inSpeechId` is not exist in `listSpeechId`. It generates by run `xhrPostIdentifySpeech`, that is used to send generate request with `inSpeechId` and `txtAreaText` to the server and receive response from the server.

generate.js

```javascript
function xhrGetSpeechId() {
  const request = new XMLHttpRequest();
  request.onreadystatechange = function () {
    if (request.readyState == 4 && request.status == 200) {
      speechId = JSON.parse(request.responseText).speechId;
      const error = JSON.parse(request.responseText).error;

      console.info('speechId', speechId);

      if (typeof error !== 'undefined') {
        alert(error);
        open('/app/mimic/home.html', '_self');
      } else {
        for (i = 0; i < speechId.length; i++) {
          const option = document.createElement('option');
          option.value = speechId[i];

          listSpeechId.appendChild(option);
        }

        btnGenerate.disabled = false;
        btnFinish.disabled = false;
      }
    }
  };
  request.open('GET', '/loadSpeech');
  request.send();
}
```

Figure 5.18 Mimic generate xhrPostIdentifySpeech code.

5.2.4 Server Code

Server code is used to create the back-end of Collect and Mimic application. It consists of `collect`, `identify`, and `generate` routers, `serverHandler` and `createServer`. The server is inspired and forked from RecordRTC package [25]. The routers use formidable package [36] that is used to parse request data and upload file from request.

5.2.4.1 collect Router

`collect` router is used to handle upload request from Collect Syllables page that will be used by `serverHandler`. It is written on JavaScript.

`collect` router contains `uploadCollect` and `router`. `router` is used to check whether the request is from Collect Syllables page or not. If yes, it runs `uploadCollect` corresponding to matched syllable request. If no, it returns nothing. `uploadCollect` is used to upload file from the request to the directory that have the same syllable name in collect directory. It converts its upload path location to URL. It returns the URL as response.

```
collect.js
function uploadCollect(address, port, label, request) {
  const dirData = '\\data';
  const dirCollect = '\\collect';
  const dirLabel = '\\' + label + '\\';

  const form = new formidable.IncomingForm();
  form.uploadDir = process.cwd() + dirData + dirCollect + dirLabel;
  form.keepExtensions = true;
  form.maxFieldsSize = 10 * 1024 * 1024;
  form.maxFields = 1000;
  form.multiples = false;

  return new Promise(function (resolve) {
    form.parse(request, function (err, fields, files) {
      const file = util.inspect(files);

      const fileName = file.split('path:')[1].split('\',')[0].split(dirData)[1].split(dirCollect)[1].split
      (dirLabel)[1].toString().replace(/\\/g, '').replace(/\//g, '');
      const fileURL = 'http://' + address + ':' + port + '/data/collect/' + label + '/' + fileName;

      console.log('fileURL: ', fileURL);
      resolve(JSON.stringify({ fileURL: fileURL }));
      return;
    });
  });
}
```

Figure 5.19 Server collect router uploadCollect code.

5.2.4.2 identify Router

`identify` router is used to handle upload request from Mimic Identify page that will be used by `serverHandler`. It is written on JavaScript.

`identify` router contains `extractAndConvert`, `createMelFilterbank`, `uploadSpeech`, `identifySpeech`, and `router`. `router` is used to check whether the request is from Mimic Identify page or not. If yes, it runs `uploadSpeech` or `identifySpeech` depends to matched request. If no, it returns nothing. `uploadSpeech` is used to upload file from the request same like `collect` router `uploadCollect`. The different are, `uploadSpeech` upload to the speech directory and returns the URL and upload path as response.

`identifySpeech` is used to identify the recorded file from request. First, it loads latest inserted trained model from MongoDB `models` collection. Then, it runs `extractAndConvert` alongside with `createMelFilterbank` ,that is used to extract the file and convert to model input same like `extractWav` and `nextBatch` combined, then feed it to model. Error connection or found model will return the information as response. Unsatisfied identification or results below 0.75 (75%) from identification also will return information as response. If none of those is occurred, which means model identify the recorded file with high accuracy, will update the MongoDB `speechDatas` collection based on its speech ID from the request and identified syllable. Then it returns the information as response.

identify.js

```javascript
// identify the speech and process data from it.
const prediction = tfModel.predict(data).dataSync();
let predictionLabel = 0;
for (i = 0; i < prediction.length; i++) {
  if (prediction[predictionLabel] <= prediction[i]) predictionLabel = i;
}
const label = labels[predictionLabel]
const labelData = prediction[predictionLabel]

// return if result unsatisfied, below 75%.
if (labelData < 0.75 || label === 'unknown') {
  status = 'No syllables matched.';
  console.log('Highest predictions ' + label + ' with ' + labelData + ' accuracy.');
  console.log('Status:', status);

  resolve(JSON.stringify({ status: status }));
  return;
} else {
  // find if `name` is already registered in MongoDB or not.
  try {
    model.speechDatas.db = await connection.connect();
    resultDB = await model.speechDatas.findOne({ name: name }).select('syllables -_id').exec();

    // create the document if not found, update if found before update to MongoDB.
    let updateData = {};
    if (resultDB === null) {
      updateData.name = name;
      updateData.syllables = {};
      updateData.syllables[label] = filePath
    }
    else {
      updateData = resultDB;
      updateData.syllables[label] = filePath
    }
    await model.speechDatas.updateOne({ name: name }, updateData, { upsert: true });
```

Figure 5.20 Some part of Server identify router identifySpeech code.

5.2.4.3 generate Router

`generate` router is used to handle upload request from Mimic Generate page that will be used by `serverHandler`. It is written on JavaScript.

`generate` router contains `loadSpeech`, `generateSpeech`, and `router`. `router` is used to check whether the request is from Mimic Generate page or not. If yes, it runs `loadSpeech` or `generateSpeech` depends to matched request. If no, it returns nothing. `loadSpeech` is used to load all registered speech ID from MongoDB `speechDatas` collection. If error connection or no speech ID is found it return error message as response. If not, it returns the load result as response.

`generateSpeech` is used to generate speech based on text from the request. First, it loads registered syllables from MongoDB `speechDatas` collection based on speech ID from the request. Error connection will return error message as response. Then, the text is extracted by check to loaded syllable one by one. If unmatched is found, it returns error as response. If all matched, the extract result is decoded and combined to 1 audio file. the speech data corresponding to extract result is combined to 1 audio with node-wav package, `encode` and `decode` [32]. The audio file is saved into generate directory. It converts its file path location to URL. It returns the URL as response.

```
generate.js ×
const extractWord = [];
for (i = 0; i < words.length; i++) {
  if (words[i] === ' ') { // check if it is space.
    extractWord.push(' ');
  } else if (i + 3 <= words.length && syllables.includes(words.substring(i, i + 3))) { // check if it the next 3 char is in `syllables`.
    extractWord.push(words.substring(i, i + 3));
    i = i + 2;
  } else if (i + 2 <= words.length && syllables.includes(words.substring(i, i + 2))) { // check if it the next 2 char is in `syllables`.
    extractWord.push(words.substring(i, i + 2));
    i = i + 1;
  } else if (syllables.includes(words.substring(i, i + 1))) { // check if it the next char is in `syllables`.
    extractWord.push(words.substring(i, i + 1));
  } else { // return if unregistered syllable is found.
    error = words[i] + ' is not found or registered from the speech data. ' +
      'Registered ' + name + ' syllables : ' + syllables;
    console.log('Error:', error);

    resolve(JSON.stringify({ fileURL: fileURL, error: error }));
    return;
  }
}
```

Figure 5.21 Some part of Server genearte router generateSpeech code.

5.2.4.4 serverHandler

`serverHandler` is used to define server listener, `collect`, `identify`, `generate` router and file handler, that will be used by `createServer`. It is written on JavaScript.

`serverHandler` contains `fileHandler`. It first will import `collect`, `identify`, and `generate` router to check whether the request is for them or not. If yes, it runs to corresponding router depends to matched request. As it returns response it also returns 200 that means the request is successful [37]. If no, it run `fileHandler`. `fileHandler` is used to check whether the corresponding URL has file in the directory. If the file does not exist it returns 404, that means the requested resource could not be found [37], to the browser. If the file does exist but cannot be read or displayed, it returns 500, that means unexpected condition was encountered and no more specific message is suitable [37], to the browser. And if the file does exist and can be read or displayed, it will return the binary so that the browser can load and display alongside with 200.

```
server.js ×
async function serverHandler(request, response) {
  const uri = url.parse(request.url).pathname;
  const filename = path.join(process.cwd(), uri);

  const collect = await require('./router/collect')(address, port, filename, request);
  if (typeof collect !== 'undefined') {
    response.writeHead(200);
    response.write(collect);
    response.end();
    return;
  }

  const identify = await require('./router/identify')(address, port, filename, request);
  if (typeof identify !== 'undefined') {
    response.writeHead(200);
    response.write(identify);
    response.end();
    return;
  }

  const generate = await require('./router/generate')(address, port, filename, request);
  if (typeof generate !== 'undefined') {
    response.writeHead(200);
    response.write(generate);
    response.end();
    return;
  }

  fs.exists(filename, function (exists) {
    if (!exists) {
      response.writeHead(404);
      response.write('404 Not Found: ' + filename + '\n');
      response.end();
      return;
    }

    fs.readFile(filename, 'binary', function (err, file) {
      if (err) {
        response.writeHead(500);
        response.write(err + '\n');
        response.end();
        return;
      }

      response.writeHead(200);
      response.write(file, 'binary');
      response.end();
    });
  });
```

Figure 5.22 Some part of Server serverHandler code.

5.2.4.5 createServer

`createServer` is used to create http server in the computer that use `serverHandler` as request listener. It is written on JavaScript.

`createServer` contains `createServer` and `listen`. `createServer` is used to create the server and followed by `listen` that is used to listening for connection that comes to the server. It listens on defined host and port. When the server established, it prints information in console.

server.js

```
// create the server and listen with `serverHandler`.
app = server.createServer(serverHandler);
app = app.listen(port, address, function () {
  // print information regarding to the app.
  console.log('Could take time to load tensorflow before the page could be open.');
  console.log('Collect: ', 'http://' + address + ':' + port + '/app/collect/home.html');
  console.log('Mimic: ', 'http://' + address + ':' + port + '/app/mimic/home.html');
  console.log('Readme: ', 'http://' + address + ':' + port + '/readme.md');
});
```

Figure 5.23 Server createServer code.

5.2.5 Database Code

Database code is used to connection to MongoDB and collection model scheme of the database on MongoDB. It consists of `connection` and `model`. There is no need to set up the database structure inside MongoDB. When `connection` and `model` is imported and a document will be inserted, MongoDB automatically setup the structure based on defined schemes on `model`.

5.2.5.1 connection

`connection` is used to handle the connection with MongoDB. It is written on JavaScript.

`connection` contains `connect` and `disconnect` methods. `connect` is used to establish the connection to MongoDB `mimic_speech` database on localhost. `disconnect` is used to close the established connection with MongoDB

connection.js

```
const mongoose = require('mongoose');

let connection;

/**
 * `connect` establish connection to mongodb mimic_speech database on localhost.
 *
 * Establised connection is saved in `connection`.
 */
async function connect() {
  connection = await mongoose.connect('mongodb://localhost/mimic_speech', { useNewUrlParser: true });
  return connection;
};

/**
 * `disconnect` close established `connection` to mongodb.
 */
async function disconnect() {
  await connection.disconnect();
}

// export `connect` and `disconnect` functions.
module.exports = {
  connect,
  disconnect
};
```

Figure 5.24 Database connection code.

5.2.5.2 model

`model` is used to create the collection model scheme of the database on MongoDB. It is written on JavaScript.

`model` contains `speechDatas` and `models` methods. `speechDatas` is used to define scheme with `name` as string and `syllables` as object. `speechDatas` scheme model is used for store the speech data. `name` is speech ID and `syllables` is the registered syllables by the corresponding speech ID. `models` is used to define scheme with `location` as string and `createAt` as date with current date being default value when create or update collection when `createAt` is not defined. `models` scheme model is used for store the machine learning model. `location` is the model location path in directory and `createAt` is the date when the model is created or saved to MongoDB.

model.js

```
const mongoose = require('mongoose');

// define `speechDatas` schema for its mongodb collection.
const speechDatas = mongoose.model('speechdatas', mongoose.Schema({
  name: String,
  syllables: Object,
}));

// define `models` schema for its mongodb collection.
const models = mongoose.model('models', mongoose.Schema({
  location: String,
  createAt: {
    type: Date,
    default: Date.now
  }
}));

// export `speechDatas` and `models` functions.
module.exports = {
  speechDatas,
  models
};
```

Figure 5.25 Database model code.

CHAPTER VI

SYSTEM TESTING

6.1 Testing Environment

The testing environment specify the environment during testing. The environment specification are as follows:

1. Windows 10.
2. Chrome browser.

There is additional environment only on the speech recognition, Mimic application identify process. Male user that the records is trained by the application is called Tester 1. Female user that the records is trained by the application is called Tester 2. Male user that the record has not trained by the application called Tester 3. Female user that the record has not trained by the application called Tester 4. Every tester is test in each the following environment:

1. Noisy background (Loud music or people chit-chat).
2. Semi noisy background (Rain noise or sound from the other rooms).
3. Not noisy background.

6.2 Testing Scenario

The testing scenario conducted by evaluating all the features with a set of cases or scenario in the application based on its functionality requirement and defined testing environment.

The testing scenarios of the application is categorized based on Collect application, Train application, and Mimic application. There also additional testing scenarios on speech recognition which is Mimic application identify process and speech synthesis which is Mimic application generate process.

6.2.1 Collect Application

The Collect application has 3 subcategorized scenarios, Collect server, Home page and Syllables page. The Collect server has scenarios to allow user to access Home page, Syllables page in the browser and process the upload record request. The Home page has scenarios to allow user to direct to Syllables page to start the Collect application and direct to Mimic application Home page to change to Mimic application. Syllables page has scenarios to allow user to upload the record to the server and direct to the next Syllables page or direct back to Home page.

Table 6.1 Collect application scenarios.

No	Scenario	Expected Result	Result
1	Access Home page	Home page is shown	As expected
2	Access Syllables page	Syllables page is shown	As expected
3	Process upload record request	Printed process results information on the console	As expected
4	Direct to Syllables page	Directed to Syllables page	As expected
5	Direct to Mimic application Home page	Directed to Mimic application Home page	As expected
6	Upload the record	Audio element is shown with the user record	As expected
7	Direct to the next Syllables page or direct back to Home page	Directed to the next Syllables page or Directed back to Home Page	As expected

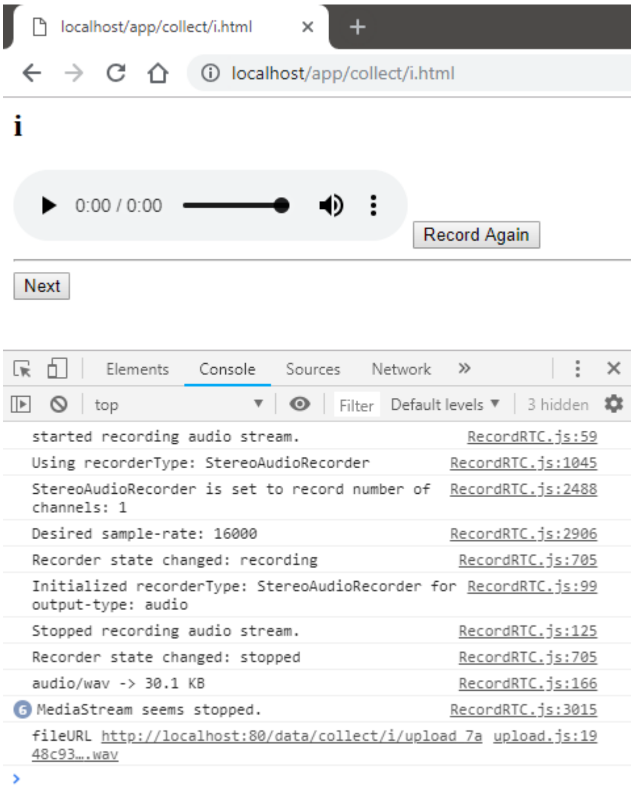


Figure 6.1 Screenshot of process upload record request scenario from Collect application.

6.2.2 *Train Application*

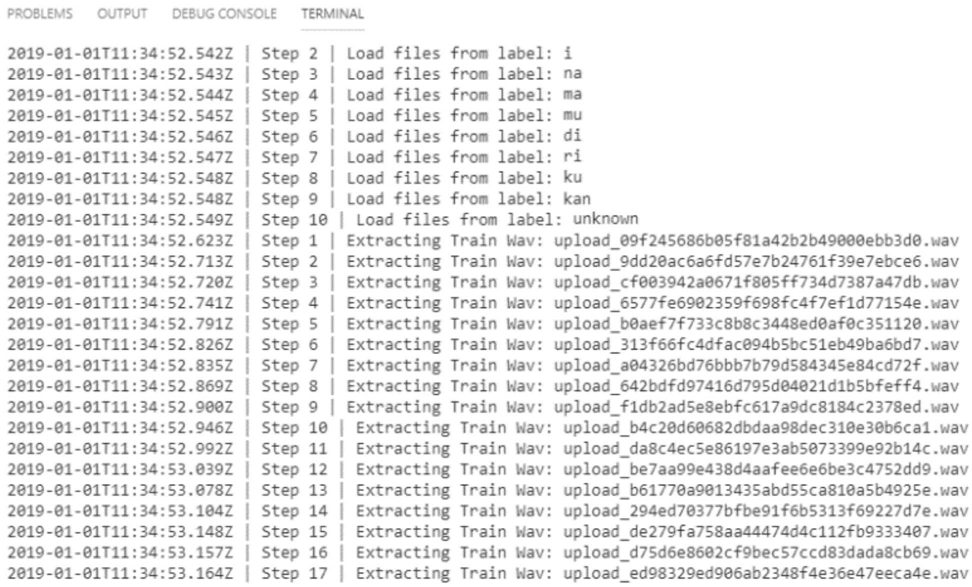
The Train application has scenario to allows user to train model with collected data. When train is run, it loads and splits collected data, extracts the data, gets the corresponding batch data, trains the TensorFlow model to fit the batch train data, tests the TensorFlow model from test data, and saves the TensorFlow model to database.

Table 6.2 Train application scenarios.

No	Scenario	Expected Result	Result
1	Load and split collected data	Printed information on the console	As expected

Table 6.2 (continued).

2	Extract the data	Printed information on the console	As expected
3	Get corresponding batch data	Printed information on the console	As expected
4	Train the TensorFlow model to fit batch train data	Printed information on the console	As expected
5	Test the TensorFlow model from test data	Printed information on the console	As expected
6	Save the TensorFlow model to database	Recorded document in `mimic_speech` database `models` collection	As expected



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

2019-01-01T11:34:52.542Z | Step 2 | Load files from label: i
2019-01-01T11:34:52.543Z | Step 3 | Load files from label: na
2019-01-01T11:34:52.544Z | Step 4 | Load files from label: ma
2019-01-01T11:34:52.545Z | Step 5 | Load files from label: mu
2019-01-01T11:34:52.546Z | Step 6 | Load files from label: di
2019-01-01T11:34:52.547Z | Step 7 | Load files from label: ri
2019-01-01T11:34:52.548Z | Step 8 | Load files from label: ku
2019-01-01T11:34:52.548Z | Step 9 | Load files from label: kan
2019-01-01T11:34:52.549Z | Step 10 | Load files from label: unknown
2019-01-01T11:34:52.623Z | Step 1 | Extracting Train Wav: upload_09f245686b05f81a42b2b49000ebb3d0.wav
2019-01-01T11:34:52.713Z | Step 2 | Extracting Train Wav: upload_9dd20ac6a6fd57e7b24761f39e7ebce6.wav
2019-01-01T11:34:52.720Z | Step 3 | Extracting Train Wav: upload_cf003942a0671f805ff734d7387a47db.wav
2019-01-01T11:34:52.741Z | Step 4 | Extracting Train Wav: upload_6577fe6902359f698fc4f7ef1d77154e.wav
2019-01-01T11:34:52.791Z | Step 5 | Extracting Train Wav: upload_b0aef7f733c8b8c3448ed0af0c351120.wav
2019-01-01T11:34:52.826Z | Step 6 | Extracting Train Wav: upload_313f66fc4dfac094b5bc51eb49ba6bd7.wav
2019-01-01T11:34:52.835Z | Step 7 | Extracting Train Wav: upload_a04326bd76bbb7b79d584345e84cd72f.wav
2019-01-01T11:34:52.869Z | Step 8 | Extracting Train Wav: upload_642bdfd97416d795d04021d1b5bfeff4.wav
2019-01-01T11:34:52.900Z | Step 9 | Extracting Train Wav: upload_f1db2ad5e8ebfc617a9dc8184c2378ed.wav
2019-01-01T11:34:52.946Z | Step 10 | Extracting Train Wav: upload_b4c20d60682dbdaa98dec310e30b6ca1.wav
2019-01-01T11:34:52.992Z | Step 11 | Extracting Train Wav: upload_da8c4ec5e86197e3ab5073399e92b14c.wav
2019-01-01T11:34:53.039Z | Step 12 | Extracting Train Wav: upload_be7aa99e438d4aafee6e6be3c4752dd9.wav
2019-01-01T11:34:53.078Z | Step 13 | Extracting Train Wav: upload_b61770a9013435abd55ca810a5b4925e.wav
2019-01-01T11:34:53.104Z | Step 14 | Extracting Train Wav: upload_294ed70377bfbe91f6b5313f69227d7e.wav
2019-01-01T11:34:53.148Z | Step 15 | Extracting Train Wav: upload_de279fa758aa44474d4c112fb9333407.wav
2019-01-01T11:34:53.157Z | Step 16 | Extracting Train Wav: upload_d75d6e8602cf9bec57ccd83dada8cb69.wav
2019-01-01T11:34:53.164Z | Step 17 | Extracting Train Wav: upload_ed98329ed906ab2348f4e36e47eeca4e.wav
```

Figure 6.2 Screenshot of load and split collected data and extract the data scenarios from Train application.

6.2.3 Mimic Application

The Mimic application has 4 subcategorized scenarios, Mimic server, Home page, Identify page, and Generate page. The Mimic server has scenarios to allow user to access Home page, Identify page, and Generate page in the browser and process the upload record and identify speech on Identify request and load speech data and generate speech on Generate request. The Home page has scenarios to allow user to direct to Identify page to start the identify speech process, direct to Generate page to start the generate speech process, and direct to Collect application Home page to change to Collect application. The Identify page has scenarios to allow user to upload the record to the server, send form request for identify speech to the server, and direct back to Home page. The Generate page has scenarios to allow user to load speech data from the server, send form request for generate speech to the server, and direct back to Home page.

Table 6.3 Mimic application scenarios.

No	Scenario	Expected Result	Result
1	Access Home page	Home page is shown	As expected
2	Access Identify page	Identify page is shown	As expected
3	Access Generate page	Generate page is shown	As expected
3	Process upload record on Identify request	Printed process results information on the console	As expected
4	Process identify speech on Identify request	Printed process results information on the console	As expected
5	Process load speech data on Generate request	Printed process results information on the console	As expected
6	Process generate speech on Generate request	Printed process results information on the console	As expected
7	Direct to Identify page	Directed to Identify page	As expected
8	Direct to Generate page	Directed to Generate page	As expected
9	Direct to Collect application Home page	Directed to Collect application Home page	As expected
10	Upload the record	Audio element is shown with the user record	As expected
11	Send form for identify speech	Alert information despite success or error in the process	As expected

Table 6.3 (continued).

12	Direct back to Home page from Identify page	Directed back to Home Page	As expected
13	Load speech data	Datalist element store the speech data or alert if no speech data is found	As expected
14	Send form for generate speech	Audio element is shown with the generated speech or alert if error is occurred	As expected
15	Direct back to Home page from Generate page	Directed back to Home Page	As expected

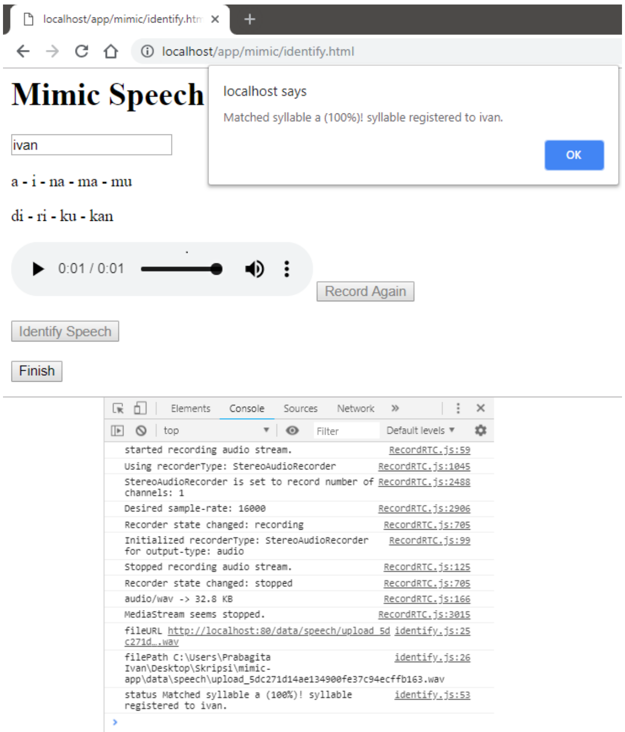


Figure 6.3 Screenshot of process identify speech on Identify request scenario from Mimic application.

6.2.4 Speech Recognition

The dataset during the speech recognition testing on identify process is 10000 male and female speech data, each 500 on each syllable, on not noisy background and each 500 unknown sounds. Corrected result shows from the more than 75% of model accuracy. The table also filled with accuracy with the syllable result along with it. Random or unknown condition is tested with silent condition, o, and mi syllables. The following tables is the tests results below and figure that shows screenshot of identify process results:

Table 6.4 Tester 1 results on noisy background.

No	**Spoken Syllable**	**Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	52 *(a)*	100 *(a)*	100 *(unknown)*	1/3
2	i	62 *(na)*	85 *(ri)*	96 *(ma)*	0/3
3	na	100 *(unknown)*	86 *(kan)*	100 *(unknown)*	0/3
4	ma	100 *(a)*	84 *(unknown)*	100 *(a)*	0/3
5	mu	99 *(kan)*	71 *(ku)*	88 *(unknown)*	0/3
6	di	59 *(mu)*	100 *(ri)*	97 *(ri)*	0/3
7	ri	50 *(na)*	100 *(ri)*	92 *(a)*	1/3
8	ku	100 *(ku)*	85 *(a)*	100 *(kan)*	1/3
9	kan	72 *(unknown)*	61 *(kan)*	79 *(kan)*	3/3
10	Unknown 1 *(silent)*	99 *(unknown)*	100 *(a)*	99 *(unknown)*	2/3
11	Unknown 2 *(o)*	100 *(na)*	100 *(a)*	100 *(ku)*	0/3

Table 6.4 (continued).

12	Unknown 3 *(mi)*	100 *(a)*	51 *(kan)*	96 *(ma)*	1/3

Table 6.5 Tester 1 results on semi noisy background.

No	**Spoken Syllable**	**Semi Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	100 *(a)*	100 *(a)*	100 *(a)*	3/3
2	i	100 *(i)*	36 *(ri)*	100 *(i)*	2/3
3	na	97 *(mu)*	48 *(ma)*	99 *(na)*	1/3
4	ma	49 *(i)*	98 *(ri)*	79 *(mu)*	0/3
5	mu	55 *(ri)*	99 *(ku)*	50 *(ri)*	0/3
6	di	57 *(i)*	100 *(di)*	100 *(i)*	1/3
7	ri	100 *(ri)*	75 *(ri)*	100 *(ri)*	3/3
8	ku	92 *(ku)*	68 *(ku)*	91 *(ku)*	2/3
9	kan	100 *(kan)*	94 *(na)*	100 *(kan)*	2/3
10	Unknown 1 *(silent)*	100 *(a)*	99 *(unknown)*	100 *(ma)*	2/3
11	Unknown 2 *(o)*	100 *(ku)*	89 *(ku)*	72 *(ku)*	1/3
12	Unknown 3 *(mi)*	100 *(ri)*	94 *(ri)*	100 *(ri)*	0/3

Table 6.6 Tester 1 results on not noisy background.

No	**Spoken Syllable**	**Not Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	100 (a)	100 (a)	100 (a)	3/3

Table 6.6 (continued).

2	i	94 (i)	48 (kan)	94 (i)	2/3
3	na	80 (na)	99 (na)	100 (na)	3/3
4	ma	96 (di)	90 (i)	57 (di)	0/3
5	mu	100 (i)	86 (ku)	95 (i)	0/3
6	di	99 *(ri)*	100 *(i)*	80 *(ri)*	0/3
7	ri	100 *(ri)*	96 *(ri)*	91 *(ri)*	3/3
8	ku	85 *(ku)*	96 *(ku)*	100 *(ku)*	3/3
9	kan	92 *(kan)*	100 *(kan)*	100 *(kan)*	3/3
10	Unknown 1 *(silent)*	98 *(unknown)*	98 *(unknown)*	98 *(unknown)*	3/3
11	Unknown 2 *(o)*	100 *(ku)*	98 *(ku)*	65 *(ku)*	1/3
12	Unknown 3 *(mi)*	54 *(mu)*	100 *(ri)*	67 *(mu)*	2/3

Table 6.7 Tester 2 results on noisy background.

No	**Spoken Syllable**	**Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	99 *(unknown)*	99 *(a)*	99 *(a)*	2/3
2	i	82 *(unknown)*	98 *(ma)*	99 *(ri)*	0/3
3	na	52 *(ri)*	99 *(ma)*	61 *(ma)*	0/3
4	ma	50 *(unknown)*	37 *(unknown)*	95 *(ri)*	0/3
5	mu	99 *(unknown)*	52 *(di)*	99 *(unknown)*	0/3
6	di	99 *(i)*	99 *(unknown)*	99 *(unknown)*	0/3

Table 6.7 (continued).

7	ri	86 *(i)*	20 *(kan)*	57 *(di)*	0/3
8	ku	99 *(ku)*	68 *(kan)*	90 *(unknown)*	1/3
9	kan	84 *(kan)*	99 *(kan)*	78 *(a)*	2/3
10	Unknown 1 *(silent)*	100 *(unknown)*	100 *(unknown)*	100 *(unknown)*	3/3
11	Unknown 2 *(o)*	99 *(ma)*	99 *(ku)*	99 *(ku)*	0/3
12	Unknown 3 *(mi)*	36 *(di)*	96 *(mu)*	99 *(di)*	1/3

Table 6.8 Tester 2 results on semi noisy background.

No	**Spoken Syllable**	**Semi Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	100 *(a)*	100 *(unknown)*	99 *(a)*	2/3
2	i	92 *(ri)*	97 *(unknown)*	96 *(ri)*	0/3
3	na	83 *(ma)*	62 *(unknown)*	75 *(kan)*	0/3
4	ma	87 *(unknown)*	64 *(ma)*	90 *(na)*	2/3
5	mu	76 *(ma)*	78 *(ku)*	66 *(di)*	0/3
6	di	58 *(mu)*	48 *(mu)*	30 *(kan)*	0/3
7	ri	99 *(kan)*	89 *(kan)*	99 *(ri)*	1/3
8	ku	99 *(unknown)*	99 *(ku)*	88 *(unknown)*	1/3
9	kan	99 *(kan)*	65 *(kan)*	85 *(kan)*	2/3
10	Unknown 1 *(silent)*	100 *(unknown)*	79 *(a)*	99 *(ri)*	1/3
11	Unknown 2 *(o)*	99 *(a)*	82 *(ku)*	99 *(na)*	0/3

Table 6.8 (continued).

12	Unknown 3 *(mi)*	89 *(kan)*	35 *(ku)*	96 *(ri)*	1/3

Table 6.9 Tester 2 results on not noisy background.

No	**Spoken Syllable**	**Not Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	A	88 *(a)*	99 *(a)*	99 *(a)*	3/3
2	I	99 *(i)*	99 *(i)*	52 *(i)*	2/3
3	Na	97 *(a)*	54 *(kan)*	99 *(a)*	0/3
4	Ma	99 *(na)*	99 *(na)*	98 *(na)*	0/3
5	Mu	81 *(i)*	95 *(i)*	44 *(i)*	0/3
6	Di	99 *(i)*	36 *(i)*	96 *(i)*	0/3
7	Ri	47 *(ri)*	32 *(unknown)*	48 *(i)*	0/3
8	Ku	98 *(ku)*	67 *(kan)*	99 *(ku)*	2/3
9	Kan	84 *(na)*	85 *(na)*	87 *(na)*	0/3
10	Unknown 1 *(silent)*	99 *(unknown)*	99 *(unknown)*	100 *(unknown)*	3/3
11	Unknown 2 *(o)*	100 *(ku)*	99 *(ku)*	95 *(ku)*	0/3
12	Unknown 3 *(mi)*	67 *(ri)*	78 *(di)*	77 *(i)*	1/3

Table 6.10 Tester 3 results on noisy background.

No	**Spoken Syllable**	**Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	A	100 *(a)*	99 *(a)*	100 *(a)*	3/3

Table 6.10 (continued).

2	I	71 *(ma)*	99 *(ri)*	99 *(ri)*	0/3
3	Na	70 *(ku)*	99 *(a)*	57 *(unknown)*	0/3
4	Ma	99 *(a)*	100 *(a)*	100 *(a)*	0/3
5	mu	74 *(mu)*	96 *(kan)*	74 *(mu)*	0/3
6	di	99 *(ri)*	96 *(ri)*	89 *(ri)*	0/3
7	ri	100 *(unknown)*	90 *(na)*	86 *(a)*	0/3
8	ku	98 *(kan)*	64 *(mu)*	99 *(a)*	0/3
9	kan	99 *(na)*	100 *(kan)*	98 *(kan)*	2/3
10	Unknown 1 *(silent)*	99 *(a)*	99 *(a)*	100 *(a)*	0/3
11	Unknown 2 *(o)*	92 *(na)*	84 *(na)*	61 *(mu)*	1/3
12	Unknown 3 *(mi)*	64 *(kan)*	53 *(na)*	99 *(ri)*	2/3

Table 6.11 Tester 3 results on semi noisy background.

No	**Spoken Syllable**	**Semi Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	100 *(a)*	99 *(a)*	99 *(a)*	3/3
2	i	99 *(ri)*	90 *(di)*	92 *(di)*	0/3
3	na	100 *(kan)*	99 *(kna)*	92 *(ma)*	0/3
4	ma	99 *(a)*	100 *(kan)*	100 *(kan)*	0/3
5	mu	99 *(mu)*	99 *(mu)*	65 *(ku)*	2/3
6	di	99 *(ri)*	100 *(ri)*	99 *(ri)*	0/3

Table 6.11 (continued).

7	ri	95 *(kan)*	76 *(i)*	78 *(di)*	0/3
8	ku	99 *(a)*	96 *(a)*	100 *(a)*	0/3
9	kan	99 *(a)*	96 *(a)*	99 *(a)*	0/3
10	Unknown 1 *(silent)*	99 *(ri)*	73 *(di)*	60 *(unknown)*	2/3
11	Unknown 2 *(o)*	100 *(a)*	99 *(ku)*	100 *(a)*	0/3
12	Unknown 3 *(mi)*	99 *(na)*	99 *(ri)*	99 *(i)*	0/3

Table 6.12 Tester 3 results on not noisy background.

No	**Spoken Syllable**	**Not Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	100 *(a)*	100 *(a)*	100 *(a)*	3/3
2	i	92 *(ri)*	95 *(ri)*	41 *(ku)*	0/3
3	na	99 *(na)*	74 *(ku)*	80 *(a)*	1/3
4	ma	59 *(a)*	84 *(ku)*	48 *(kan)*	0/3
5	mu	99 *(mu)*	99 *(ri)*	99 *(ri)*	1/3
6	di	100 *(ri)*	99 *(i)*	99 *(i)*	0/3
7	ri	99 *(ri)*	96 *(ri)*	94 *(i)*	2/3
8	ku	73 *(di)*	100 *(mu)*	99 *(mu)*	0/3
9	kan	100 *(kan)*	99 *(kan)*	96 *(kan)*	3/3
10	Unknown 1 *(silent)*	98 *(unknown)*	98 *(unknown)*	98 *(unknown)*	3/3
11	Unknown 2 *(o)*	99 *(ku)*	68 *(mu)*	100 *(ku)*	1/3

Table 6.12 (continued).

12	Unknown 3 *(mi)*	99 *(ri)*	93 *(i)*	100 *(ri)*	0/3

Table 6.13 Tester 4 results on noisy background.

No	Spoken Syllable	Noisy Background			Correct Result
		1	2	3	
1	a	100 *(unknown)*	99 *(unknown)*	94 *(ma)*	0/3
2	i	97 *(unknown)*	99 *(ri)*	99 *(ri)*	0/3
3	na	100 *(na)*	99 *(ri)*	99 *(ma)*	1/3
4	ma	93 *(na)*	99 *(ma)*	100 *(na)*	1/3
5	mu	55 *(ri)*	99 *(unknown)*	43 *(unknown)*	0/3
6	di	99 *(ri)*	54 *(i)*	99 *(ma)*	0/3
7	ri	97 *(ri)*	74 *(ri)*	100 *(ri)*	2/3
8	ku	68 *(ma)*	92 *(ku)*	52 *(52)*	1/3
9	kan	48 *(ku)*	22 *(ku)*	99 *(a)*	0/3
10	Unknown 1 *(silent)*	100 *(unknown)*	100 *(unknown)*	100 *(unknown)*	3/3
11	Unknown 2 *(o)*	100 *(a)*	100 *(a)*	100 *(a)*	0/3
12	Unknown 3 *(mi)*	55 *(mu)*	99 *(ri)*	94 *(i)*	1/3

Table 6.14 Tester 4 results on semi noisy background.

No	Spoken Syllable	Semi Noisy Background			Correct Result
		1	2	3	
1	a	73 *(a)*	91 *(unknown)*	99 *(unknown)*	0/3

Table 6.14 (continued).

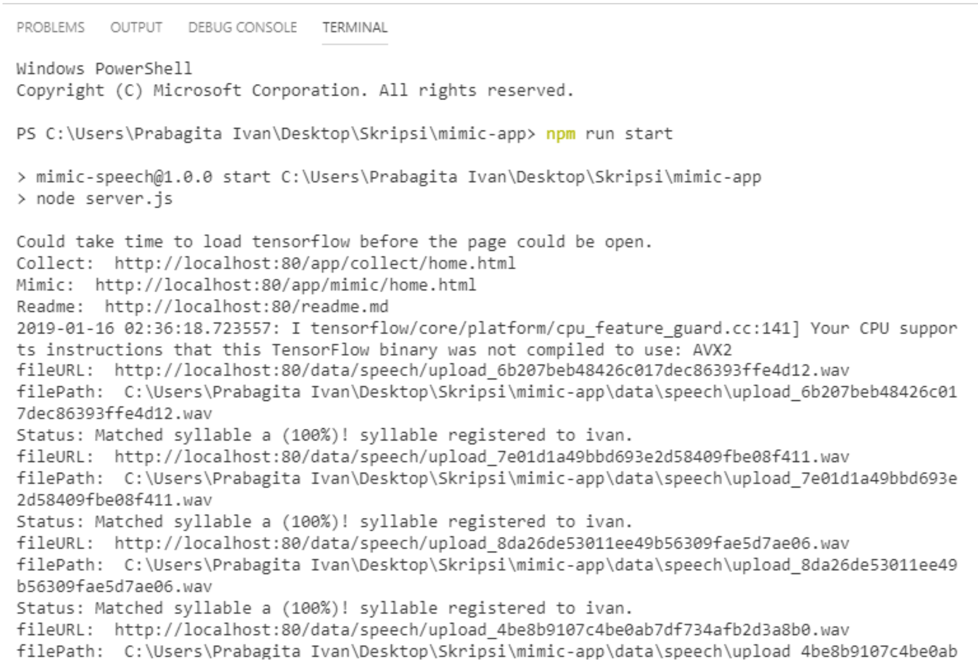
2	i	81 *(ri)*	99 *(i)*	100 *(ri)*	1/3
3	na	100 *(ma)*	99 *(unknown)*	99 *(unknown)*	0/3
4	ma	99 *(ma)*	99 *(na)*	78 *(ma)*	2/3
5	mu	81 *(unknown)*	99 *(unknown)*	99 *(unknown)*	0/3
6	di	100 *(ri)*	99 *(i)*	99 *(i)*	0/3
7	ri	85 *(ri)*	99 *(ri)*	98 *(ri)*	3/3
8	ku	96 *(ku)*	89 *(ku)*	99 *(a)*	2/3
9	kan	100 *(a)*	99 *(a)*	64 *(ma)*	0/3
10	Unknown 1 *(silent)*	99 *(unknown)*	99 *(ri)*	100 *(unknown)*	2/3
11	Unknown 2 *(o)*	99 *(a)*	100 *(a)*	99 *(a)*	0/3
12	Unknown 3 *(mi)*	43 *(a)*	59 *(ri)*	99 *(unknown)*	3/3

Table 6.15 Tester 4 results on not noisy background.

No	**Spoken Syllable**	**Not Noisy Background**			**Correct Result**
		1	**2**	**3**	
1	a	99 *(unknown)*	75 *(a)*	100 *(a)*	2/3
2	i	88 *(i)*	99 *(i)*	99 *(ri)*	2/3
3	na	99 *(na)*	99 *(ma)*	99 *(ma)*	1/3
4	ma	55 *(ma)*	72 *(kan)*	99 *(na)*	0/3
5	mu	99 *(unknown)*	94 *(unknown)*	28 *(i)*	0/3
6	di	89 *(i)*	99 (ri)	99 *(ri)*	0/3

Table 6.15 (continued).

7	ri	99 *(ri)*	100 *(ri)*	99 *(ri)*	3/3
8	ku	39 *(kan)*	99 *(ku)*	58 *(mu)*	1/3
9	kan	100 *(a)*	99 *(a)*	100 *(a)*	3/3
10	Unknown 1 *(silent)*	40 *(unknown)*	24 *(unknown)*	60 *(unknown)*	3/3
11	Unknown 2 *(o)*	100 *(a)*	99 *(a)*	100 *(a)*	0/3
12	Unknown 3 *(mi)*	60 *(ri)*	100 *(i)*	99 *(ri)*	1/3


```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Prabagita Ivan\Desktop\Skripsi\mimic-app> npm run start

> mimic-speech@1.0.0 start C:\Users\Prabagita Ivan\Desktop\Skripsi\mimic-app
> node server.js

Could take time to load tensorflow before the page could be open.
Collect:  http://localhost:80/app/collect/home.html
Mimic:  http://localhost:80/app/mimic/home.html
Readme:  http://localhost:80/readme.md
2019-01-16 02:36:18.723557: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
ts instructions that this TensorFlow binary was not compiled to use: AVX2
fileURL:  http://localhost:80/data/speech/upload_6b207beb48426c017dec86393ffe4d12.wav
filePath:  C:\Users\Prabagita Ivan\Desktop\Skripsi\mimic-app\data\speech\upload_6b207beb48426c01
7dec86393ffe4d12.wav
Status: Matched syllable a (100%)! syllable registered to ivan.
fileURL:  http://localhost:80/data/speech/upload_7e01d1a49bbd693e2d58409fbe08f411.wav
filePath:  C:\Users\Prabagita Ivan\Desktop\Skripsi\mimic-app\data\speech\upload_7e01d1a49bbd693e
2d58409fbe08f411.wav
Status: Matched syllable a (100%)! syllable registered to ivan.
fileURL:  http://localhost:80/data/speech/upload_8da26de53011ee49b56309fae5d7ae06.wav
filePath:  C:\Users\Prabagita Ivan\Desktop\Skripsi\mimic-app\data\speech\upload_8da26de53011ee49
b56309fae5d7ae06.wav
Status: Matched syllable a (100%)! syllable registered to ivan.
fileURL:  http://localhost:80/data/speech/upload_4be8b9107c4be0ab7df734afb2d3a8b0.wav
filePath:  C:\Users\Prabagita Ivan\Desktop\Skripsi\mimic-app\data\speech\upload_4be8b9107c4be0ab
```

Figure 6.4 Screenshot of Tester 1 with a and i syllables on not noisy background.

Although noisy and semi background on every tester not show a good result, it still can predict about 2 or 3 correct spoken syllables with high accuracy. Not noisy background is not guaranteed all spoken syllables are correct even on tester 1 and 2.

But, the result is better than noisy and semi background. This is because all the training data is recorded on not noisy background. Also, noise removal or reduction is not applied when extraction process is done before training begin.

Tester 1 and 2 at most moment have good result than tester 3 and 4. But, at some moment tester 3 and 4 have better result than tester 1 and 2. The most good result on tester 1 and 2 is due to the training data is all contain tester 1 and 2. When tester 3 and tester 4 have better result it can be cause by the background condition or the microphone ability to record the speech.

Syllable ma, mu, di, unknown o, and unknown mi have bad result on most time compare to others. The unknown o and mi are caused by the unknown training data is random and mostly background noise instead of focusing o and mi. The machine learning model cannot predict syllable ma, mu, and di well. Most times they have low accuracy but correct prediction or they predict the relative syllable, in example ri or i is the result on di. This can be improved by adding more the training data as the training data is still relatively small. An optimum machine learning model can also improve the result on them and also other syllables as well.

6.2.5 Speech Synthesis

The following are random text to test generating the speech:

1. diriku
2. aku makan ikan
3. di mana mamamu

4. halo namaku ivan

‘diriku’, ‘aku makan ikan’, and ‘di mana mamamu’ can be generated. But, ‘halo namaku ivan’ cannot. It happens because ‘h’ is not found in the database as the text analysed from the beginning. The application alert user on the browser that ‘h’ is not found and listed registered syllables. Actually ‘h’ itself is not listed on available syllables. As the text contains syllable that not listed on available syllables or even contains available syllables but not identified yet will alert user. Any syllable that not found in the database will stop the generating process.

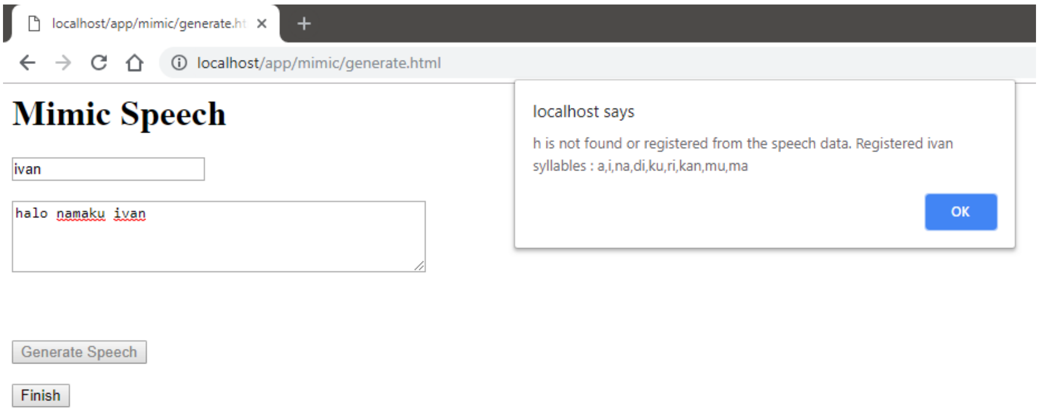


Figure 6.5 Screenshot of user generate speech from text 'halo namaku ivan'.

The following is the result from ‘diriku’, ‘aku makan ikan’, and ‘di mana mamamu’ in sequence:

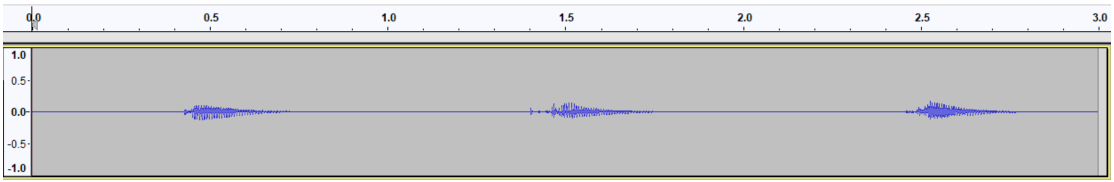


Figure 6.6 Waveform of ‘diriku’.

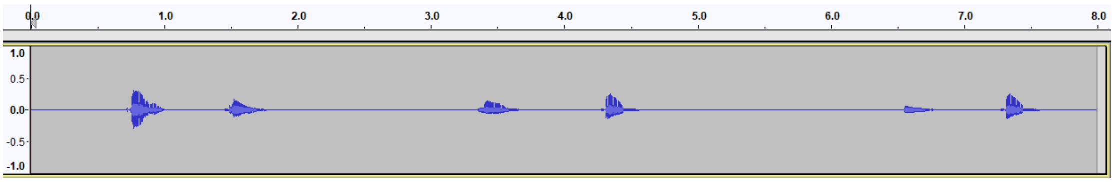


Figure 6.7 Waveform of ‘aku makan ikan’.

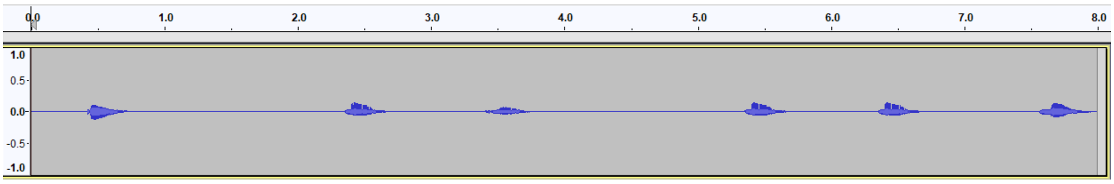


Figure 6.8 Waveform of ‘di mana mamamu’.

As shown in figures above, each text can be distinguished very easily. As the recorded speech is take 1 second, the duration of the generated speech simply sums up of syllables in the text. Although the speech is generated well, can be heard and understood, there is still silence part or redundant part between the syllables except for spaces. It makes the speech become not fluently enough. This is because there is no processing to analysed and delete the redundant part in concatenative process or right after identifying the speech in the application.

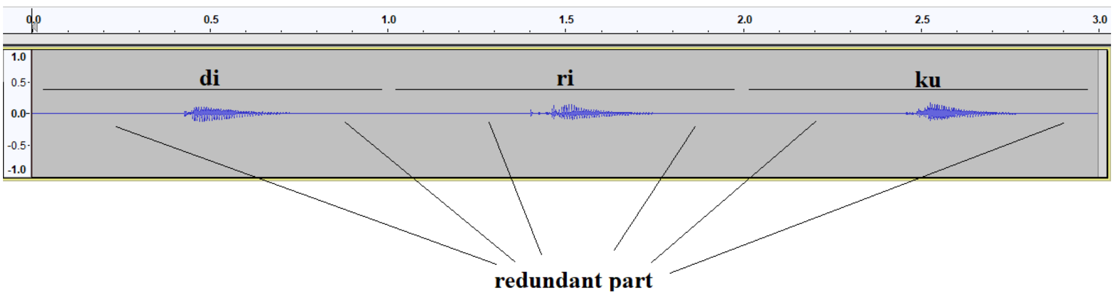


Figure 6.9 Waveform of 'diriku' show syllables and redundant parts.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

The following list sums up several points achieved in the application based on the research objectives:

1. This application is able to collect speech data through website.
2. This application is able to train machine learning model with collected data through command prompt.
3. This application is able to mimic speech in Bahasa Indonesia through website.
4. This application is able to recognize speech from record audio although the prediction and accuracy are not perfect. But, the machine learning able to predict well most of the times.
5. This application is able to generate speech based on inputted text although the result still has silent or redundant part. But, the generated speech can be heard and understood.

7.2 Future Work

The following suggestion for further development and improvements of the research or application:

1. User Interface

   Improvement on UI will always help the user experience. With some colourful theme, clear button and inputs, the application will be more comfortable to be used.

2. Speech Recognition

Improvement on machine learning model can be made. When there is no right or wrong in modelling the machine learning model, there is always optimal model to get the best prediction and accuracy. A research to find the optimal model to be compare with big improvement in the application or even in speech recognition itself for mimic speech.

3. Speech Synthesis

Improvement on Speech Synthesis is when generating the speech. By removing some silence or unused part of the speech and also reducing background will make generated speech more fluently and good to hear. A research in determining Bahasa Indonesia phonemes can be a big improvement since the application use syllables as concatenative synthesis.

REFERENCES

[1] wikibooks.org. (2018, February 28). *Introduction to Software Engineering/Process/Rapid Application Development*. Retrieved from Wikibooks: https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Process/Rapid_Application_Development

[2] Naz, R., & Khan, M. N. (2015). Rapid Applications Development Techniques: A Critical Review. *International Journal of Software Engineering and Its Applications*, 163-176.

[3] wikipedia.org. (2018, December 4). *Rapid application development*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Rapid_application_development

[4] Rabiner, L. R., & Schafer, R. W. (2007). Introduction to Digital Speech Processing. *Foundations and Trends® in Signal Processing*, Volume 1, Issue 1–2.

[5] Hande, S. S. (2014). A Review on Speech Synthesis an Artificial Voice Production. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 8.

[6] oxforddictionaries.com. (2018). *syllable*. Retrieved from Oxford Dictionaries: https://en.oxforddictionaries.com/definition/syllable

[7] Panitia Pengembang Pedoman Bahasa Indonesia, K. P. (2016). *Pedoman Umum Ejaan Bahasa Indonesia.* Jakarta: Badan Pengembangan dan Pembinaan Bahasa,.

[8] M.A.Anusuya, & S.K.Katti. (2009). Speech Recognition by Machine: A Review. *International Journal of Computer Science and Information Security*, 25.

[9] Dave, B., & Pipalia, P. D. (2014). SPEECH RECOGNITION: A REVIEW. *International Journal of Advance Engineering and Research*, 7.

[10] Sainath, T. N., & Parada, C. (2015). *Convolutional Neural Networks for Small-footprint Keyword Spotting.* New York, NY, U.S.A: Google, Inc.

[11] Geitgey, A. (2016, December 24). *Machine Learning is Fun Part 6*. Retrieved from Medium: https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

[12] wikipedia.org. (2018, October). *Nyquist–Shannon sampling theorem*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

[13] practicalcryptography.com. (2012). *Mel Frequency Cepstral Coefficient (MFCC) tutorial*. Retrieved from Practical Cryptography:

http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[14] haythamfayek.com. (2016, April 21). *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. Retrieved from Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[15] wikipedia.org. (2018, December 7). *Mel-frequency cepstrum*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

[16] Hurwitz, J., & Kirsch, D. (2018). *Machine Learning For Dummies®, IBM Limited Edition.* New Jersey: John Wiley & Sons, Inc.

[17] Nilsson, N. J., & Laboratory, R. (2005). *INTRODUCTION TO MACHINE LEARNING.* California: Nils J. Nilsson.

[18] Society, T. R. (2017). *Machine learning: the power and promise.* London: The Royal Society.

[19] Geitgey, A. (2016, Juny 14). *Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks*. Retrieved from Medium: https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[20] wikipedia.org. (2018, December 26). *Convolutional neural network*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Convolutional_neural_network

[21] lyrebird.ai. (2018). *Our Voice Products*. Retrieved from Lyrebird: https://about.lyrebird.ai/products

[22] translate.google.com. (2018). *Languages*. Retrieved from Google Translate: https://translate.google.com/intl/en/about/languages/

[23] w3school.com. (2018). *Node.js Introduction*. Retrieved from W3School: https://www.w3schools.com/nodejs/nodejs_intro.asp

[24] mongodb.com. (2018). *NoSQL Databases Explained*. Retrieved from MongoDb: https://www.mongodb.com/nosql-explained

[25] muaz-khan. (2019, January 1). *RecordRTC*. Retrieved from Github: https://github.com/muaz-khan/RecordRTC

[26] tensorflow.org. (n.d.). *TensorFlow.js*. Retrieved from TensorFlow: https://js.tensorflow.org/

[27] tensorflow. (2018, November 30). *tensorflow/examples/speech_commands*. Retrieved from Github: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/speech_commands

[28] tfjs-models. (2018, December 22). *speech-commands*. Retrieved from Github: https://github.com/tensorflow/tfjs-models/tree/master/speech-commands

[29] jameslyons. (2018, December 20). *python_speech_features/tree/master/python_speech_features*. Retrieved from Github: https://github.com/jameslyons/python_speech_features/tree/master/python_speech_features

[30] vail-systems. (2016, December 7). *node-mfcc*. Retrieved from Github: https://github.com/vail-systems/node-mfcc

[31] meyda. (2018, December 6). *meyda*. Retrieved from Github: https://github.com/meyda/meyda

[32] andreasgal. (2016, August 26). *node-wav*. Retrieved from Github: https://github.com/andreasgal/node-wav

[33] vail-systems. (2015, Juny 23). *node-dct*. Retrieved from Github: https://github.com/vail-systems/node-dct

[34] pazguille. (2017, February 16). *shuffle-array*. Retrieved from Github: https://github.com/pazguille/shuffle-array

[35] sindresorhus. (2014, November 14). *crypto-random-string*. Retrieved from Github: https://github.com/sindresorhus/crypto-random-string

[36] felixge. (2018, Juny 25). *node-formidable*. Retrieved from Github: https://github.com/felixge/node-formidable

[37] wikipedia.org. (2019, January 2). *List_of_HTTP_status_codes*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes