Voice Assiatant for Prononciation Practicing

Zhoukatong Xia
Erik Jonsson School of Engineering and Computer Science
The University of Texas at Dallas
Richardson, Texas
Email: zxx180009@utdallas.edu

Yingning Yuan
Erik Jonsson School of Engineering and Computer Science
The University of Texas at Dallas
Richardson, Texas
Email: yxy180049@utdallas.edu

Abstract – We build this program as a voice assistant for pronunciation practicing, it can provide the following benefit to the user, First, currently, most software is use mouse and keyboard as input device and output visual information from the screen. But our program provided an alternative way for manipulation, based on speech recognition, you can use your microphone to command this program, this will free your hands and relieve eye strain. Second, it can train user's pronunciation in an efficient way. For example, when I intend to replay what I have just said every time, the program will satisfy my needs and do an automatic replay after I stopped speaking, also, I might have needs on repeating for a second time, or show me the standard pronunciation, this program can be manipulated by speaking out some keywords as commands like repeat or speak. In this report, you will see the architecture of this program, including a speech to text module, a text to speech module, GUI, dataflow, and decision-making logics.

Keywords—voice assistant, pronunciation training, speech recognition, hidden Markov model, neural network, gTTS.

I. INTRODUCTION

(Zhoukatong Xia: 20%, Yingning Yuan: 80%)

Since the 21st century, as our work and life are gradually becoming globalized, we are always in situations where we need to communicate with people from other countries. Therefore, it has become extremely important to learn a universal language in the world to ensure people's cross-regional and cross-cultural communication. The importance of English as the universal language used by billions of people around the world is self-evident. It can be said that learning and improving English is the foundation for us to achieve global communication. The ability to have a good spoken English level is particularly important in communication and communication, especially the use of standard English pronunciation in spoken expressions can help us avoid a lot of embarrassment and misunderstandings in communication. Therefore, it is more necessary than ever to learn English, communicate proficiently in English, and master standard English pronunciation.

Traditional speaking courses are useful and important. However, traditional courses often require us to spend a lot of time, and it is difficult to intuitively understand our own pronunciation problems. The result of learning traditional courses is usually that we do understand what the correct pronunciation is, but we don't know where our pronunciation problems are, so it is difficult to correct the pronunciation on our own. With the development of technology, more and more intelligent English learning methods are emerging. How to use new intelligent technology to make the learning of spoken pronunciation in a more automatic way, using less time and more targeted practice and improvement is still a challenging and open research field.[1]

Voice assistants engage users like no other interface. Users can speak to the applications naturally to ask for whatever they'd like and can do so while multitasking. This also reduces the cognitive effort for the user to get what they want from the application, and applications that take advantage of voice assistants are seen as easier to use than applications that do not. According to a recent PwC report, "Consumers see voice assistants as the smarter, faster, and easier way to perform everyday activities." and that 93% of consumers are satisfied with their voice assistants; 50%, very satisfied. Voice assistants help people feel organized, informed, happy, smart, and confident.[1]

Common problems with existing Graphical User Interfaces that use touch, type, and mouse are that users have to figure out how to use the application, and oftentimes miss the value of the app and get frustrated in the process. This friction is a top reason why users drop off from applications after a single use. With a voice assistant in your app, users are able to ask for exactly what they want and get help when they need it. This is a natural way for users to discover your application value and for you to make the perfect first impression. Voice also guides the user during the initial onboarding flow to prevent single-use drop-offs and bring every user to the "wow" moment of your application faster.

The purpose of our research is to implement a self-service voice assistant for practicing spoken English pronunciation, helping users to intuitively compare their own pronunciation with standard pronunciation, and to be able to compare repeatedly. This voice assistant can display a text language interface, allowing users to smoothly interact with the computer in a smoother and more natural way. Suitable for users who want to improve their English proficiency quickly, no need to participate in actual courses, no specific time and place, no need to watch or input text, practice spoken English pronunciation anytime, anywhere. And it is very friendly to people who have almost no eyesight or people who are inconvenient to watch the screen interface and other text sources.

II. RELATED WORK

(Zhoukatong Xia: 50%, Yingning Yuan: 50%)

Voice assistants or are programs using natural language processing (NLP) and speech synthesis to perform certain tasks on the user command. Currently, they have been a major part of our smartphones, computers for the past few years, If you are an iPhone user certainly you are using Siri, or if you are an android user you know your Google Assistant.

Currently, there are three ways to make your program understand verbal language and keep up a conversation. [2]

The first method involves integrating existing voice technologies into your application by means of special APIs and other development tools. The second method allows you to build an intelligent assistant with the help of open source services and APIs. The third method is to create your own voice assistant from scratch with it's further integration into your application. Each method is worthy of attention. Note that the big names like Apple or Google reluctantly offer their beloved creations to the third-party developers.

Siri

If you ever studied Siri, you certainly noticed that it was unavailable for most of the third-party applications. With iOS 10 release, the situation has changed a lot. At WWDC 2016, it was announced that Siri can be integrated with the apps that work in the following areas:

Audio and video calls

Messaging and contacts

Payments via Siri

Photos search

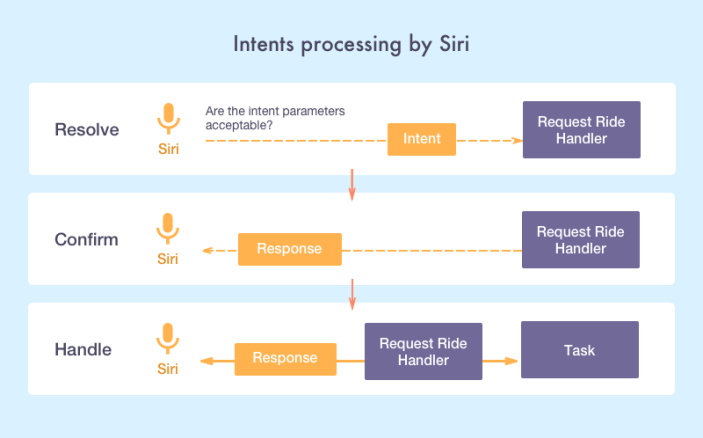
Workout Car booking

To enable the integration, Apple's introduced a special that consists of two frameworks. The first one covers the range of tasks to be supported in your app, and the second one advises on a custom visual representation when one of the tasks is performed.

Each of the app types above defines a certain range of tasks which are called intents. The term refers to the users' intentions and, as a result, to the particular scenarios of their behavior.

In SiriSDK, all the intents have corresponding custom classes with defined properties. The properties accurately describe the task they belong to. For instance, if a user wants to start a workout, the properties may include the type of exercises and the time length of a session. Having received the voice request, the system completes the intent object with the defined characteristics and sends it to the app extension. The last part processes the data and show the correct result at the output.

You can find more information about how to work with intents and objects on Apple's official website. Below is a scheme on intents processing:



Google Now and voice actions

Remember the good cop, bad cop technique? Well, Google has always been the first one to show the maximum loyalty to developers. Unlike Apple, Google doesn't have strict requirements for design. The approvement period in the Play Market is much shorter and not as fastidious as in the Apple App Store too.

Nevertheless, in a question of smart assistant integration, it appears quite conservative. For now, Google assistant works with the selected apps only. The list includes such hot names as eBay, Lyft, Airbnb, and others. They are allowed to make their own Now Cards via special API.

The good news is that you still have a chance to create Google Assistant app command for your own app. For that, you need to register the application with Google.

Remember not to confuse Google Now with the voice commands. Now is not just about listening and responding. It is an intelligent creature that can learn, analyze, and conclude. Voice actions are the narrower concept. It works on the basis of speech recognition followed by information search.

Google provides the developers with a step by step guide for integrating such functionality into an app teaches how to include a voice mechanism both in mobile and wearable apps.

Cortana

Microsoft encourages developers to use the Cortana voice assistant in their mobile and desktop apps. You can provide the users with an opportunity to set a voice control without directly calling Cortana. In the Cortana Dev Center, it describes how to make a request to a specific application. Basically, it offers three ways to integrate the app name into a voice command:

1. Prefixal, when the app name stands in front of the speech command, e.g., 'Fitness Time, choose a workout for me!'

2. Infixal, when the app name is placed in the middle of the vocal command, e.g., 'Set a Fitness Time workout for me, please!'

3. Suffixal, when the app name is put to the end of a command phrase, e.g., 'Adjust some workout in Fitness Time!'

You can activate either the background or foreground app with speech commands through Cortana. The first type is suitable for apps with simple commands that don't require additional instructions, e.g., 'Show the current date and time!'. The second - for the apps that work with more complex commands, like 'Send the Hello message to Ann'. In the last case, besides setting the command, you specify it's parameters: What message? - Hello message; Who should it be sent to? - To Ann.

Other voice assistant

1. Melissa is a real finding for the newcomers in development who want to create a custom voice assistant. The whole system consists of many parts. So, in case you want to add or modify a certain feature, you can do it without changing a complete algorithm. Melissa can speak, take notes, read news, upload pictures, play music, and do many other things. Written in Python, it works on OS X, Windows, and Linux. The web interface is developed with the help of JavaScript.

2. Jasper would be suitable for those who prefer to program the biggest part of artificial intelligence without the external support and create the custom AI assistant relying on themselves. It is also a great tool for the Raspberry Pi fans because it runs on it's Model B. Jasper is written in Python. It can listen and learn. The first capability is introduced by the active module, the second - by the passive. Always being on, it is ready to perform the tasks at any moment of day or night. Silently studying your habits, it can provide you with the precise information just in time. So, if you want to build your own voice assistant, you should consider Jasper.

3. Api.ai covers a wide range of tasks allowing to make your own personal assistant. Along with voice recognition, it also supports converting voice into text followed by the execution of the relevant tasks. Analyzing and drawing conclusions isn't alien for this service either. Api.ai has both free and paid versions. The last one enables working in a private cloud. So, if privacy is your priority, this is just what you need. Api.ai provides a wide range of APIs including iOS, Android, Windows Phone, Cordova, Python, Node.js, Unity, C#, and others.

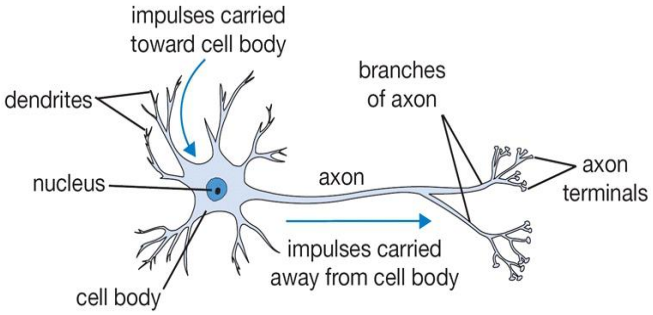
4. Wit.ai is similar to Api.ai service. There are two elements to set up in your app if you want to use it - intents and entities. Similar to the Siri system, intent stands for the action that a user wants to perform, e.g., show the weather. Entities clarify the characteristics of a given intent, e.g., the time and place of a user. One pleasant thing is that you don't need to create the intents on your own. Wit.ai provides the developers with a long list to choose from. Another piece of good news is that it is totally free both for public and private usage. However, to create your own personal assistant with the help of Wit.ai, you should follow it's terms.

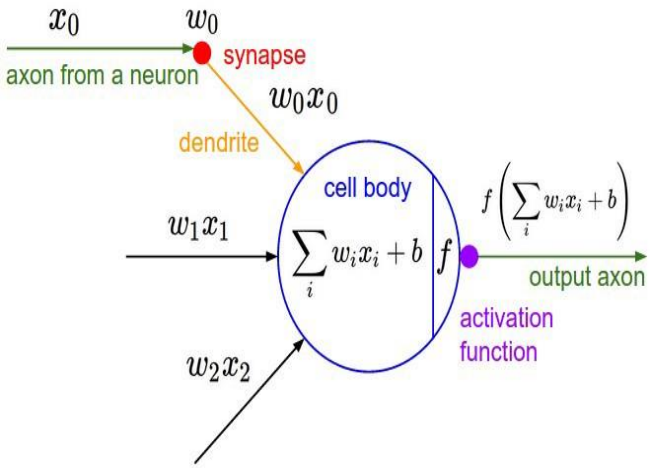
Introduction to neural networks[1]

The definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as: "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." Or you can also think of Artificial Neural Network as computational model that is inspired by the way biological neural networks in the human brain process information.

Biological motivation and connections

The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14} — 10^{15} synapses. The diagram below shows a cartoon drawing of a biological neuron (up) and a common mathematical model (down).





The basic unit of computation in a neural network is the neuron , often called a node or unit. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function to the weighted sum of its inputs.

The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence and its direction: excitory (positive weight) or inhibitory (negative weight) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. we model the firing rate of the neuron with an activation function (e.x sigmoid function), which represents the frequency of the spikes along the axon.

Neural Network Architecture

From the above explanation we can conclude that a neural network is made of neurons, biologically the neurons are connected through synapses where information flows (weights for out computational model), when we train a neural network we want the neurons to fire whenever they learn specific patterns from the data, and we model the fire rate using an activation function.

But that's not everything…

Input Nodes (input layer): No computation is done here within this layer, they just pass the information to the next layer (hidden layer most of the time). A block of nodes is also called layer.

Hidden nodes (hidden layer): In Hidden layers is where intermediate processing or computation is done, they perform computations and then transfer the weights (signals or information) from the input layer to the following layer (another hidden layer or to the output layer). It is possible to have a neural network without a hidden layer and I'll come later to explain this.

Output Nodes (output layer): Here we finally use an activation function that maps to the desired output format (e.g. softmax for classification).

Connections and weights: The network consists of connections, each connection transferring the output of a neuron i to the input of a neuron j. In this sense i is the predecessor of j and j is the successor of i, Each connection is assigned a weight Wij.

Activation function: the activation function of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, it is the nonlinear activation function that allows such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

Learning rule: The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This learning process typically amounts to modifying the weights and thresholds.

III. Approaches and Implementations

(Zhoukatong Xia: 70%, Yingning Yuan: 30%)

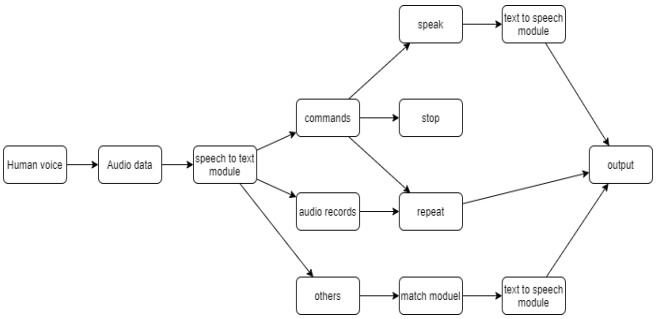
A. Design Goal

1. Build an automatic, time saving, English practice helper.
2. Free my hand, and my body. I am tired sitting and staring at the screen, if hope I can use this program when standing, waking, laying, or even when I am closing my eyes.
3. Good user experience, as I should be the main user.

B. How to achieve:

The program should need the following functions:
1. record voice
2. recognize voice command
3. Acting properly according to commands.
4. Able to speak, so people don't have to gather information from screen.

Data flow design:



1) Real time stream amplitude analyzes:

Knowing when the user is speaking or keeping silent is important. To make our program distinguish silence from sound, we need monitor the real time data flow. We used pyaudio to start a stream and calculated the mean amplitude in every 0.25 seconds. If the mean amplitude exceeds a threshold, we know something broke the silence, we suppose this is user start speaking, and this will trigger our next module. For how to decide the threshold, we will talk in the next part.

```
def test_ambient_sound():

    text = 'Now we are going to test your ambient sound, please say \
        something in the next 5 seconds'
    speak(text)

    cap = pyaudio.PyAudio()
    stream = cap.open(format = pyaudio.paInt16, channels = 1, \
                      rate = 16000,input = True, frames_per_buffer = 8192)
    stream.start_stream()

    test_time_ambient = 5
    blocks = []
    for i in range(test_time_ambient*16):
        print(i)
        data = stream.read(1000)
        block = byte_to_int(data)
        block_avg = sum([abs(x) for x in block])/len(block)*100
        blocks.append(block_avg)
    blocks.sort()
    low_level = sum(blocks[1:4])/3
    high_level = sum(blocks[-4:-1])/3

    stream.stop_stream()
    stream.close()
    cap.terminate()
    return (low_level,high_level)
```

2) Threshold Decide: background noise – human voice intensity gathering.

As we know, the boundary of speaking and silence is dependent on the input device and environment, so to decide this threshold, we need to gather intensity level of background noise and human voice

So, we can start an test to gather the intensity of background noise and human voice.

Here we will ask the user to say something in the next 5 seconds, and we regard the average of the top 15% intensity as the human voice intensity level and regard the bottom 15% as ambient sound level. Using these high level and low level values, we can set threshold = sound_level_low + (sound_level_high-sound_level_low)*0.2, this 0.2 is changeable, decreasing it will make the system more sensitive, and vice versa.

3) Prepare the audio file

Before we pass the audio data to speech to text module, there is a question, what data type should we pass? a numpy array, or a string of bytes, or a .wav file?

There are some reasons that we choose to transform our data into a .wav file first before giving it to other modules. First reason is that, local files are easy to find and manage after the program ends, and we have a plan to play previous record. Second reason is that most libraries working with sound will accept a *.wav file as input, or they have method to load. So, for future extensions of this program, we think save it into file could be a good idea.

For the implementations, we found pyaudio and wave works well together. After gathering data from pyaudio stream, the data can be directly passed into wave to generate a file, it will look like the following example

```
stream = pyaudio.PyAudio().open

audio = stream.read

waveFile = wave.open(WAVE_OUTPUT_FILENAME, 'wb')

waveFile.writeframes(audio)
```

4) Speech to text module outline design

Here is one of the most important part, speech to text.

After studying, we found there are many great APIs to achieve speech to text in few lines of code, but unfortunately, there are a few drawbacks in most of them which makes them not working well in this project.

The First drawback is a connected network are needed, your input will be uploaded to some server and the result will be send back. In our use case, we care about the latency to improve respond speed, and we think the program should be independent and robust.

Second drawback is that we do not have much control on those online models

Therefore, to make our model more fluent and robust and more controllable, I choose to use an offline, trainable model, which is called vosk. And we are not satisfied only using pretrained model, so we think add a neural network to our decision logic could be a good idea.

5) Speech to text- First model - vosk

For this model, there are some features worth to be noticed:

First, vosk support 20+ languages and dialects - English, Indian English, German, French, Spanish, Portuguese, Chinese, Russian, Turkish, Vietnamese, Italian, Dutch, Catalan, Arabic, Greek, Farsi, Filipino, Ukrainian, Kazakh, Swedish.

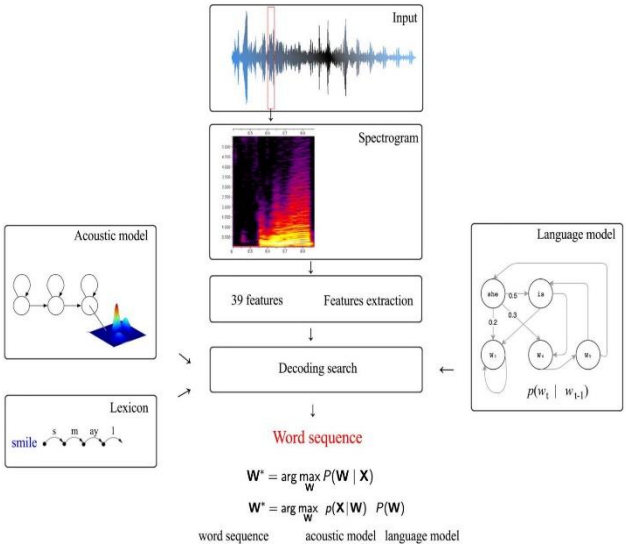
Second, it has a light-weight-version, which is what I am using in this program. It has a size is 67.7MB, runnable on lightweight devices.

Third, it is trainable, it is possible train the pretrained model using Kaldi. And Kaldi is a toolkit for speech recognition, if you prepare your training data in their specified format, you can train the model easily.

As it has so many good features we need, we continue to use vosk as one of our speech to text models

vosk structure:

vosk is Hidden Markov Model, and the schematic diagram are shown below

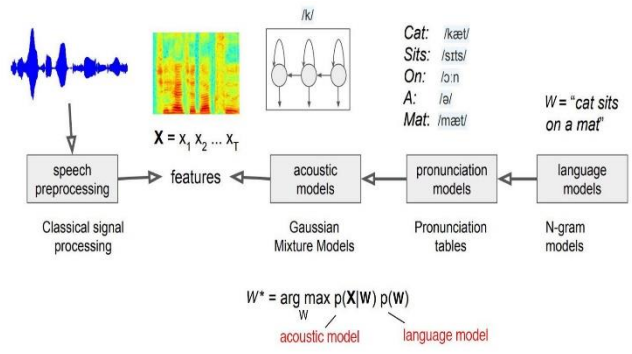


The Hidden Markov Model describes a hidden Markov Chain which at each step emits an observation with a probability that depends on the current state. In general both the hidden state and the observations may be discrete or continuous.

But for simplicity's sake let's consider the case where both the hidden and observed spaces are discrete. Then, the Hidden Markov Model is parameterized by two matrices:

The transition matrix: which defines the probabilities of the Markov chain transitioning from one state to the next

The emission matrix: which defines the probabilities of each observed state given the corresponding hidden state.



6) Speech to text- Second model - Model nn

Intending to build another model to support our decision logic, we are looking for a proper model to train. Here are some options:

First, we can build a similar speech model like vosk, which can deal with real time audio stream, and updating its prediction in real time. But this plan has a huge time cost in training, we think a successful training should take over seven days. And a week is not enough to make it perform better than vosk, so, this plan has big cost but little benefit.

As I has asked my professor for advice about how to implement each part on my project, I think using neural network to build a classifier to process commands is a proper choice.

Next, we start to work on investigating and looking for training data. The training data we found is Speech Commands dataset which is collected by Google and released under a CC BY license.

Download link:
http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip

Using this data set, we designed our neural network to project input into eight categories: ['left' 'right' 'no' 'up' 'down' 'stop' 'yes' 'go'], these categories come with the data set, so, I even though I hope there could be other categories like 'repeat' and 'speak', there are not. So currently this model can only support the decision on my 'stop' command.

Introduction to convolutional layers:

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input result in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image. The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images. [4]

Introduction to drop out layers:

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "dropped out." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "view" of the configured layer. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility

for the inputs. This conceptualization suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust. Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation as a side-effect. As such, it may be used as an alternative to activity regularization for encouraging sparse representations in autoencoder models. Because the outputs of a layer under dropout are randomly subsampled, it has the effect of reducing the capacity or thinning the network during training. As such, a wider network, e.g. more nodes, may be required when using dropout. [5]

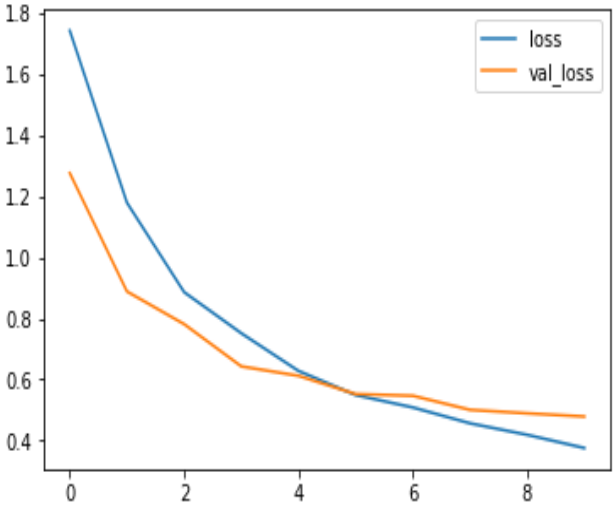
Implementation:

The first step is to convert waveforms to spectrograms. A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. It is kind like an image, and we can use convolutional layers to deal with it.

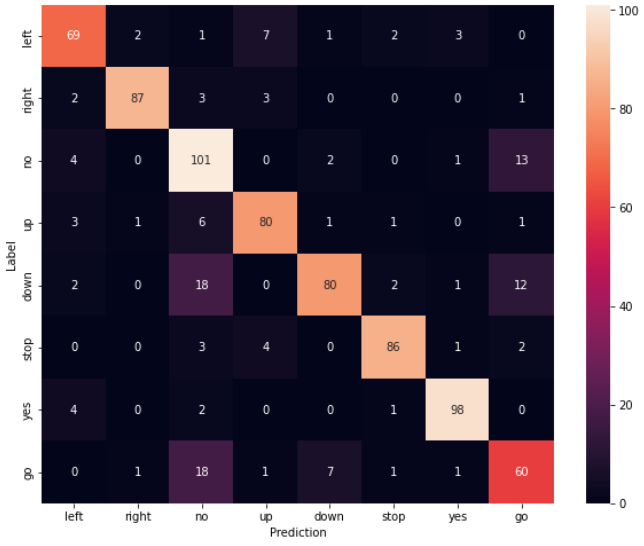
In the model design, we use conv2d, max_pooling and dense layers to build this neural network, and we also added normalization and drop_out layer for better training time and accuracy. Here is the final structure of our neural network.

```
Input shape: (124, 129, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing (Resizing)         (None, 32, 32, 1)         0

 normalization (Normalizatio  (None, 32, 32, 1)        3
 n)

 conv2d (Conv2D)             (None, 30, 30, 32)        320

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 14, 14, 64)       0
 )

 dropout (Dropout)           (None, 14, 14, 64)        0

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 128)               1605760

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 8)                 1032

=================================================================
Total params: 1,625,611
Trainable params: 1,625,608
Non-trainable params: 3
```

As you can see, this model has a total of 1,625,608 parameters for training, the model is not very big, for each epoch, we only need less than 1 minute to train, and the validation accuracy reached its peak after 10 epochs. Here is the loss during 10 epochs:



Then, we can use a confusion matrix to see how it really works, see the picture below:



7) User interaction design:

In the previous section, we have delt with the input from humans.

Now let design how the program output its information and interact with humans.

Our first version will just print out some text.

Our second version let the program speak, by adding a text to speech module.

Our third version has a simple user interface to show recording state and text prediction result. I found knowing the current recording state is handy, and the text are bigger and easier to read.

So, in the next part, let talk about the details about each version are implemented.

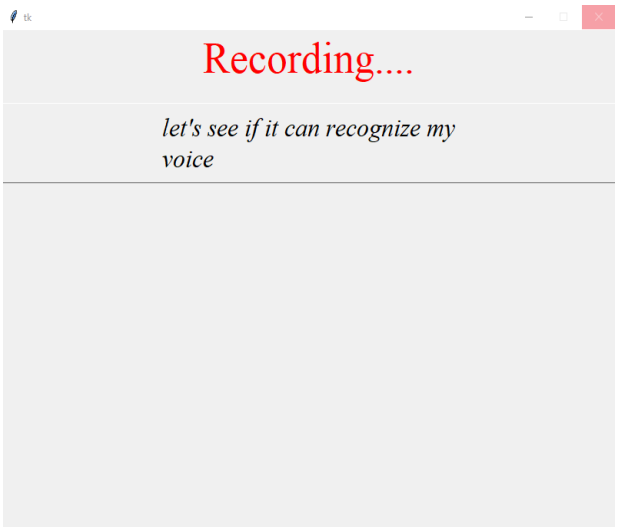
8) Text to speech

To achieve my design goal, which is free your hand and eyes, the program should be able to speak to the user.

The model we used is gTTS (Google Text-to-Speech), it is a Python library and CLI tool to interface with Google Translates text-to-speech API. It is like a black box for me, but it performs great, that is enough for a lazy programmer.

```
def speak(text):
    tts = gTTS(text = text, lang = 'en')
    filename = "abc.mp3"
    tts.save(filename)
    playsound.playsound(filename)
    os.remove(filename)
```

9) GUI



There are two problem we need to solve while build our GUI: First, real time updating text contents and text color. Second, let GUI work together with my existing code. For the first problem is easy to solve, We created a label like:

myLabel = tk.Label(root,text = 'ddddd')

And changing contents using:

myLabel.configure(text = str(count))

For the second problem, as we need to keep the GUI updating all the times, so the program could enter into an endless while loop for GUI updating. Here are some solutions:

The first solution: you can use two threads, one running the tkinter GUI, and the other running your program.

The second solution, you can only update the gui when needed, this has a draw back that the gui will not respond you until the next update, but, currently, as we only need it to show contents and states, so we chose the second approach. And call root.update() when needed, where root is the object name of this interface.

REFERENCES

[1] Zhang, Yangyong, et al. "Life after speech recognition: Fuzzing semantic misinterpretation for voice assistant applications." *Proc. of the Network and Distributed System Security Symposium (NDSS'19)*. 2019.

[2] vosk: https://alphacephei.com/vosk/

[3] kaldi: https://kaldi-asr.org/doc/about.html

[4] Brownlee, Jason. "How do convolutional layers work in deep learning neural networks?." *Machine Learning Mastery* (2020).

[5] Brownlee, Jason. "A gentle introduction to dropout for regularizing deep neural networks." *Machine Learning Mastery* 3 (2018).