3/28/2019

Speech Radar

Shah Ali 33455846

Supervised by Dr Jamie Ward

Computer Science

Goldsmiths University of London

Abstract

To do…..

Contents page

To do…

# **Introduction**

## **Background**

Since 1784, Speech recognition is something that was just a topic of talk. It wasn't until 1952 when a six-foot machine was created by Bell Labs, *capable of recognizing spoken digits with 90% accuracy* [3], but when uttered by its owner. The development would continue in 1962, where IBM created a machine the size of a shoebox that could *understand 16 English words* [3]. In 1971 a student of Carnegie Mellon University created the Harpy that could *comprehend 1011 words and some phrases* [3]. In 1986, IBM would create another ground-breaking machine that used the *Hidden Markov Model* [3] to recognise 20000 different words from various speakers and type them on paper. The list of inventions would go on with Google launching a voice search application in 2008, *bringing speech recognition to mobile devices* [3]. In 2011, Apple would announce Siri, *ushering in the age of the voice-enabled digital assistant* [3]*.*

At present, we are seeing digital assistants decentralise from smartphones and are seeing companies primarily focus on voice-activated home speakers that can query and control smart home devices. From a subjective point of view, these innovations appear to be an approach to accumulate billions of audio data from people that have different accents, so companies in the future can improve the detection rate for fluent and non-fluent English speakers. On the off chance that this improvement happens, we will see speech recognition being used for more advanced tasks, possibly in robotics.

## **Purpose**

In the area of speech recognition, it is said that Microsoft can now interpret *human speech with a 5.1% error rate* [1]. Google have enhanced its *accuracy by more than 20% in the past five years* [1]. And to date, Amazon's Alexa has been getting better at responding to users’ question. *Researchers asked the voice assistant 800 different queries* [2] in various categories. *On average, Alexa answered queries accurately 73% of the time, up 12 percentage points from 61%* [2]. These are little known facts of how top companies are taking speech recognition to a whole new level and how they've made it worth looking into and investing in for personal and business use. Given that the unexpected rise has happened so recently, this project will aim to contribute to the area by researching the tools and techniques that are being used to make it work so efficiently. The research will then be used to implement an Android application called Speech Radar, that will allow users to locate their phone through speech recognition. Suppose for instance an individual must rush to work and is unable to find their phone, but knows it’s located somewhere inside their room. Speech Radar can speed up the search time just by an utterance of a specific word from the individual. Once the phone detects the utterance, it will begin ringing at maximum sound level, which will help the individual locate their phone. This app can be of use to a great number of people as it is common for individuals nowadays to lose sight of their phone and to search everywhere for it.

## **Aims and objectives**

* To research on a feasible approach of creating a TensorFlow model for speech recognition by going through implementations of certain users public GitHub repositories
* To research on an appropriate dataset to use for training the model. Preferably, the dataset should contain audio files for each word in different accents, rather than phrases.
* To design our implementation in TensorFlow and Android Studio using pseudocode, flowcharts, wireframes and UML diagrams
* To implement our designs and test the speech recognitions detection rate with many random users. Gather the results for analysis, and use it to improve the application

## **Section overview**

**Chapter 2** - This chapter consists of the research for the project through literature reviews, in the attempt to critically evaluate other people’s work in the area of speech recognition

**Chapter 3** - This chapter contains the specification of the project. The requirements are stated precisely and in detail

**Chapter 4** - The design of the appearance and functionality is mentioned in this chapter.

**Chapter 5** – The implementation of the designs is critically discussed in this chapter

**Chapter 6** - This chapter specifies the results for user testing and analyse key areas that'll help improve the application. Software testing will also be included through black-box and white-box testing techniques. An evaluation will conclude this chapter, assessing the strengths/weaknesses of the application and what improvements could be made

**Chapter 7** - This chapter provides a conclusion for the report, with mentions of future work that can be done, successes/failures of the project and how it compared to what others have done.

**Chapter 8** – This chapter includes all bibliography’s and references used for the project

**Chapter 9** – Final chapter includes appendices to help explain all findings and analysis

# **Literature review**

This chapter introduces the research done in preparation for the next stages of the project. It analyses relevant published articles that can have an impact on the project. The research can help to find solutions for tasks that need to be prioritised and managed carefully during implementation phase. This will help gain knowledge and prepare for any problems that could take place. The focus area is in TensorFlow and Android Studio, where implementation of the project will take place.

## **Speech Commands dataset**

### **Broad analysis**

An article composed by Pete Warden examines his speech recognition implementation as a web app, using TensorFlow to create the model with the Speech Commands dataset. The main aim of the dataset is to supply a way in building and testing *small models that detect when a single word is spoken* [5] from a range of target words. This task is known as *keyword spotting* [5]. The final dataset consists of *105,829 utterances of 35 words* [5]. Each of the utterances in the Speech Commands dataset are stored as a *WAVE file- format* [5] lasting for one second. The sample data is *encoded as linear 16-bit single-channel PCM values* [5], at 16,000 Hz sampling rate. Over 2600 speakers are recorded, some with different accents.

The author of the article explains in a separate section how he wanted to have *a limited vocabulary* [5] of 10 words to ensure that the web app was lightweight and fast in making correct predictions. The choice of words were chosen carefully, so they do not have similar pronunciations and are easily detected. For example, the word 'three' is pronounced as 'TH R IY' and ‘tree’ similarly as 'T R IY'. This can cause the model to make wrong predictions, especially for someone with an accent. Thus, why the author decided to avoid these kinds of words.

Speech Commands dataset would be an incredible asset to the Speech Radar project. A limited vocabulary of code words would seem appropriate and feasible than having a large vocabulary. The article mentioned only 10 words were used. The Speech Radar app will attempt to use between 20 to 30 words from this dataset and ensure correct predictions are made most of the times. But for this to work, additional audio clips may need to be added to the dataset so the TensorFlow model can learn more.

### **Additional information**

What was incredibly helpful for the Speech Radar project was a brief mention in the article of how the TensorFlow model was loaded in the web app. The article states that it was exported as a protobuf file, which carries the graphs definition and weights of the model. This file can then be loaded in Android Studio via the assets folder. A visual flowchart inspired from this article can be seen in Appendix

## **Convolutional neural network (CNN)**

### **Keyword spotting task**

An article written by Tara N. Sainath and Carolina Parada mentions how they used a CNN to solve the keyword spotting problem on a mobile device. This is an issue that manages the recognition of keywords in articulations. Virtual assistants such as Google Assistant and Amazons Alexa use keyword spotting such as "Ok Google" or "Alexa" to wake up when their name is spoken. But for this to work properly especially with mobile devices, it must have a *small memory footprint and low computational power* [4]. Since this article has proved convincingly that CNN is the best choice for speech recognition tasks like keyword spotting, it will be analysed further to help in our implementation.

### **Wav to spectrogram**

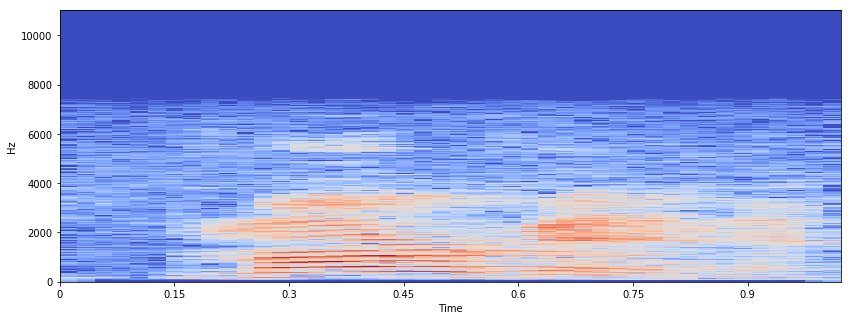
A WAV audio file can be represented as a spectrogram, which can be used as an input for a CNN. A spectrogram is an image of the signal strength/loudness of a signal over time at different frequencies present in a specific waveform. Not exclusively would one be able to see that there is energy at a certain hertz but can see how energy levels change after some time. Given that CNN's achieve great results in image classification problems, the information represented in the spectrogram is invaluable and can be extracted and learnt by the network. Figure 1 shows a spectrogram for the word ‘Marvin’. We see the frequency for every utterance needed to be made to say the word. Notice from 0.17 to 0.75 seconds, we see a high level of energy for the first part of the word pronounced as ‘M AA R’ and a low level of energy for the final part pronounced as ‘V IH N’ roughly from 0.79 to 0.95 seconds. This makes sense, since the loudness of voice for each part of the word alters i.e. goes from high to low. The low level of energy seen in 6000 hertz would most likely be the pronunciation of ‘AA’, as it is a high frequency vowel.

Figure 1: Marvin

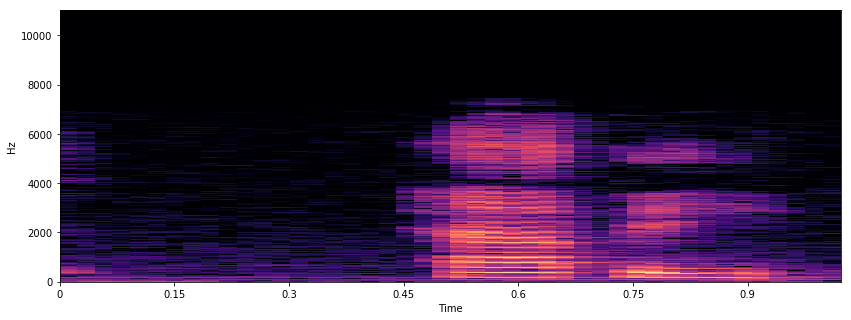
Figure 2 shows a spectrogram for another word ‘Happy’. Similarly, like figure 1 we see high levels of energy and frequency for the first part of the word pronounced as ‘HH AE’ and low levels for the final part pronounced as ‘P IY’. Notice how the frequency levels are much higher than figure 1. This tells us that the speech recognition system would be better at detecting the word ‘Happy’ than ‘Marvin’. Thus, it would be unsurprising for the results in the testing section to show different detection rates for each word.

Figure 2: happy

### **Network architecture**

A typical CNN architecture contains convolutional, pooling, normalization and fully connected layers. The table in figure 3 shows a basic model created for the keyword spotting task using the Speech Commands dataset (greater detail mentioned in section 2.1.2). We will call this model A. It is capable in recognizing three words i.e. bed, cat and happy, which is why the last layer contains only three neurons with Softmax as its activation function. Model A is small with only 82,368 parameters and will be interesting to see how it could be improved later.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Layer | Neurons & rate | Activation function | Optimizer | Learning rate | Batch size | Parameters |
| A | Conv2d | 32 | Relu | Adadelta | 0.001 | 100 | 160 |
| Max\_pooling2d | - | - | 0 |
| Dropout | 0.2 | - | 0 |
| Flatten | - | - | 0 |
| Dense | 55 | Relu | 79,255 |
| Dropout | 0.2 |  | 0 |
| Dense | 50 | Relu | 2,800 |
| Dropout | 0.2 | - | 0 |
| Dense | 3 | Softmax | 153 |
| 82,368 |

Figure 3 model A

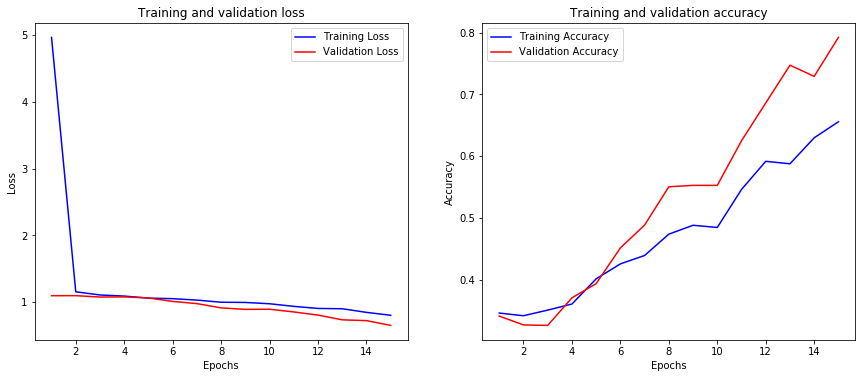
The graph in figure 4 shows the result for model A, where validation loss gradually reduces in each epoch. The loss for training and validation overall is very high and will certainly need improvement. A significant increase can be seen in the validation accuracy after 5 epochs, causing it to be much greater than the training accuracy. This means model A is underfitting. To avoid this, more parameters and layers can be added. Adjusting certain hyperparameters could also help (e.g. good learning rate, batch size etc.) Since a CNN architecture is being used, it might be best to add BatchNormalization layers to the model, as it works well with other regularization techniques such as dropout and L2. It can also speed up the learning process.

Figure 4: result

The table in figure 5 shows an extension from model A. we will call this model B. Everything highlighted in bold are changes/improvements made to the model. For example, the number of neurons has increased so that model B has more parameters to avoid underfitting. Another example would be the small reduction of the learning rate, which will hopefully prevent the performance from diverging and see the error rate reduce rapidly.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Layer | Neurons & rate | Activation function | Optimizer | Learning rate | Batch size | Parameters |
| B | Conv2d | **80** | Relu | **RMSprop** | **0.0005** | **90** | 400 |
| **Batch\_normalization** | - | Relu | 320 |
| Max\_pooling2d | - | - | 0 |
| Dropout | 0.2 | - | 0 |
| Flatten | - | - | 0 |
| Dense | **85** | Relu | 306,085 |
| **Batch\_normalization** | - | - | 340 |
| Dropout | 0.2 | - | 0 |
| Dense | **90** | Relu | 7,740 |
| **Batch\_normalization** | - | - | 360 |
| dropout | 0.2 | - | 0 |
| dense | **3** | Softmax | 273 |
| 315,518 |

Figure 5 model B

The graph in figure 6 shows the result of model B. Lower training and validation loss can be seen compared to model A. High validation and training accuracy can also be seen. This tells us how well it can make predictions based on unseen data, which is vital for when the model is used in the Android app. Training the model in under 15 epochs was probably a good estimate, as we see the validation loss slowly diverging at the end.

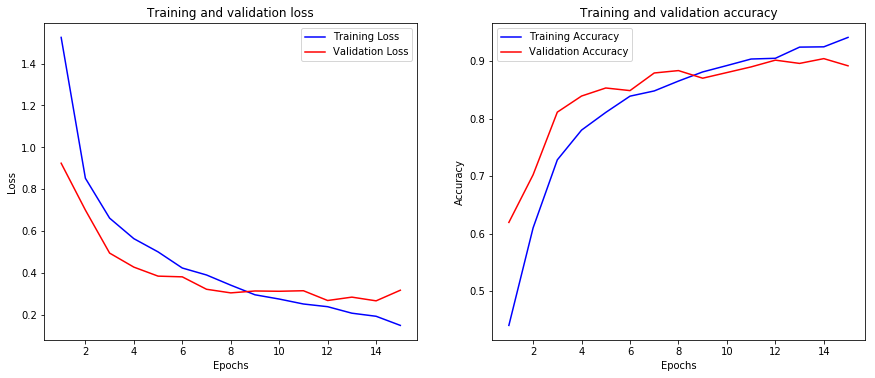


Figure 6 result

## **PocketSphinx**

### **Brief**

Another article written by Ajav Sharma and Rahul Bhalley explains their implementation of a *real-time speech recognition on portable devices* [6]. It mentions a library called PocketSphinx, a lightweight engine specifically made for mobile devices. It is a collection of *3 components: front end, decoder, and linguist* [6]. Linguist contains an acoustic model (reports the sounds of words in which any grapheme is uttered), dictionary and language model (states probability of each utterance.) Signals are inputted into the front end which are parameterized into an arrangement of features. The linguist interprets the pronunciation data present in the dictionary alongside the language model data and structural data from *acoustic model, into a search graph* [6]. Decoder incorporates *the search manager which inputs the features from front end and search graph from linguist* [6]. This is where the real decoding occurs and where results are produced. These results are sent to the application.

### **Background service**

The authors of this article used PocketSphinx to create an in-app speech recognition system. Given that this lightweight library can also run in the background of a mobile device continuously, the Speech Radar app may require PocketSphinx. On top of this, the article mentions it works without Internet connection, which can be useful if users’ phone were to suddenly disconnect.

# **Specification**

Chapter 3 states a detailed explanation of all the requirements needed to fully create the app. Examples include how it should function, how it could handle failures, what measure should be taken to prepare for potential disaster, how user can interact with the interface and many more. This would be a great section to look back at after implementation has finished, to see whether all requirements have been met.

## **Functional requirements**

Describes the behaviour of the system. Java with Android development and XML will be used to achieve this.

* System should send an email verification when user creates account to prove that provided email address exists
* Verified user should be recognised by the system and grant them access to their account
* System should ask user for permission to record audio and access the Internet when using the app first time
* When correct button is pressed, audio should record for 3 seconds to receive user input of the code word
* If user is happy with the code word, background service of continuous speech recognition should activate when button is pressed
* Background service should deactivate when user utters the code word or terminates the app
* App should work on all android devices that have API level of 16 (Jelly Bean) or above

## **Non-functional requirements**

Explains requirements of how the system should perform or is unrelated to the functionality of the app.

* System should continue to operate in an event of a failure (e.g. no Internet connection)
* System should load the TensorFlow model via the assets folder and use it to make predictions
* Quick response time when converting speech to correct text
* APK file should be 30MB or less
* A backup of all user accounts should be kept in separate database in case of a disaster
* The continuous background service should not deeply affect performance of the phone (e.g. drain too much battery life, slow down the phone etc.)
* All android devices must be touch-screen to use the interface
* Interface should be intuitive (e.g. add few objects and spread them across the screen so the layout is clean)
* System should be able to handle large number of users all online at the same time

## **Server-side requirements**

Discusses what operations should be executed by the server. Java will be used to achieve this

* Utilise Google's free SMTP server to send email verification and password reset emails upon request with Firebase Authentication
* Authenticate email address and password in login screen using the backend server
* The SpeechRecognizer API in Android Studio should be used for continuous recognition in background. The API will stream audio to servers in remote locations to carry out speech recognition

## **Database requirements**

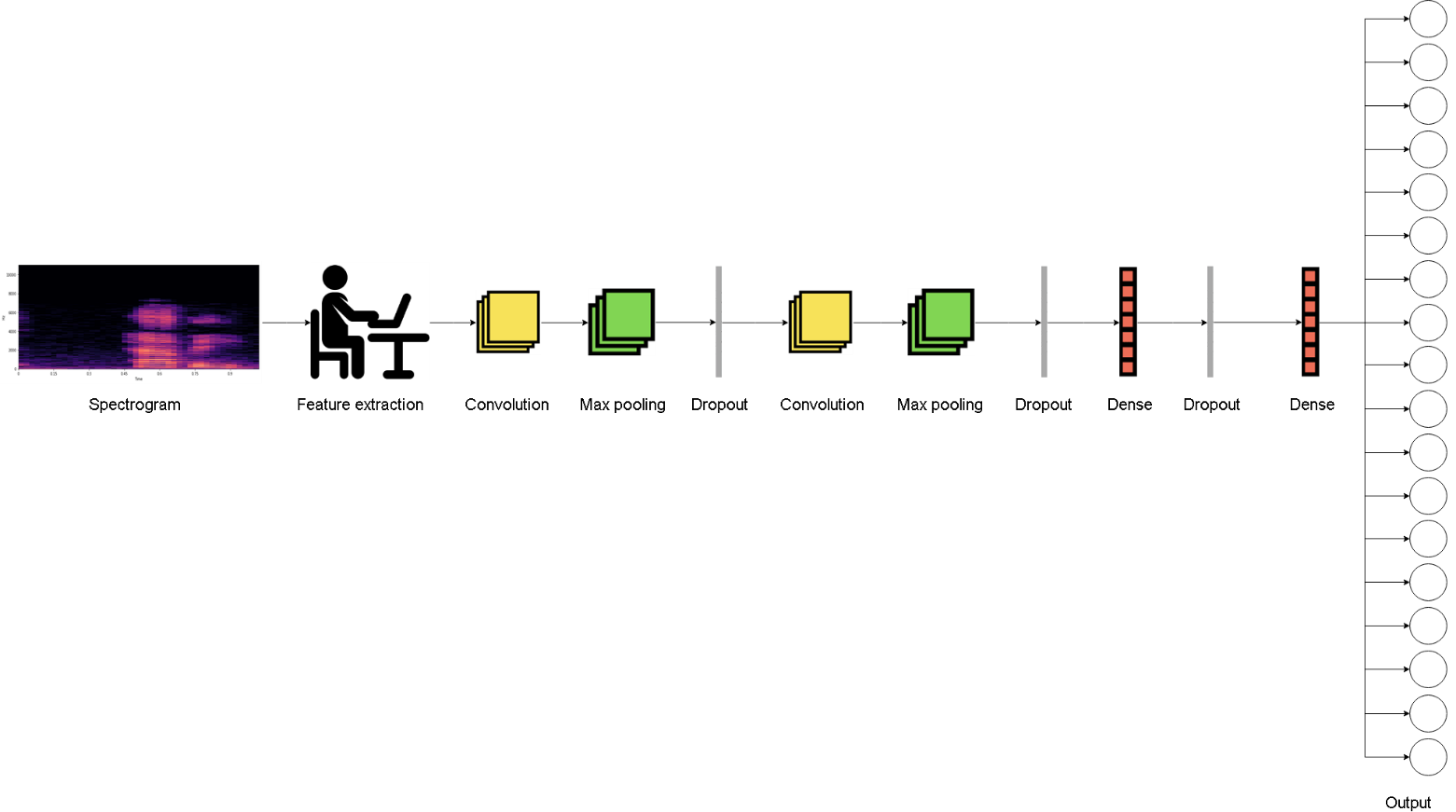
Mentions how the database should operate in the background of the app

* Offer quick data retrieval
* Insert account details to Firebases Real-time Database when user creates an account
* For security purposes, passwords must be hashed and added to the database instead of the actual password
* System should pull the first name from a record in database and display it when user logs in to their account
* Update the code word field when user has chosen or changed the code word
* Password field should be updated when user resets password
* Use SQLite as recovery database and follow all bullet points above

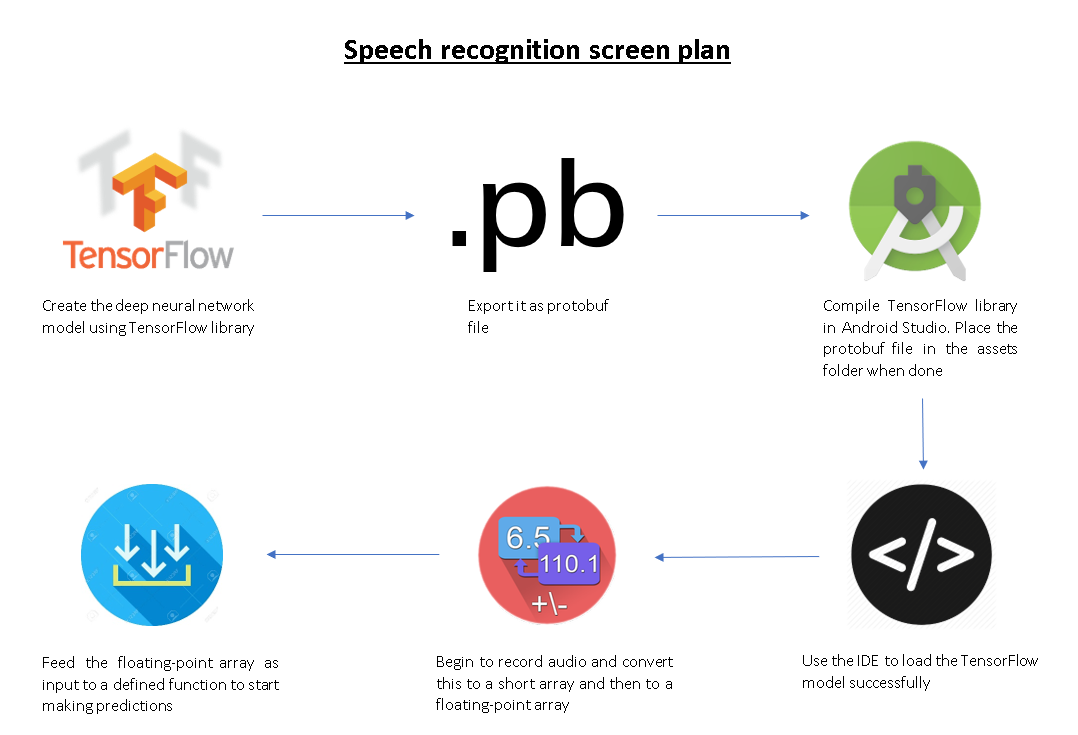
# **Design**

## **Visual flowchart**

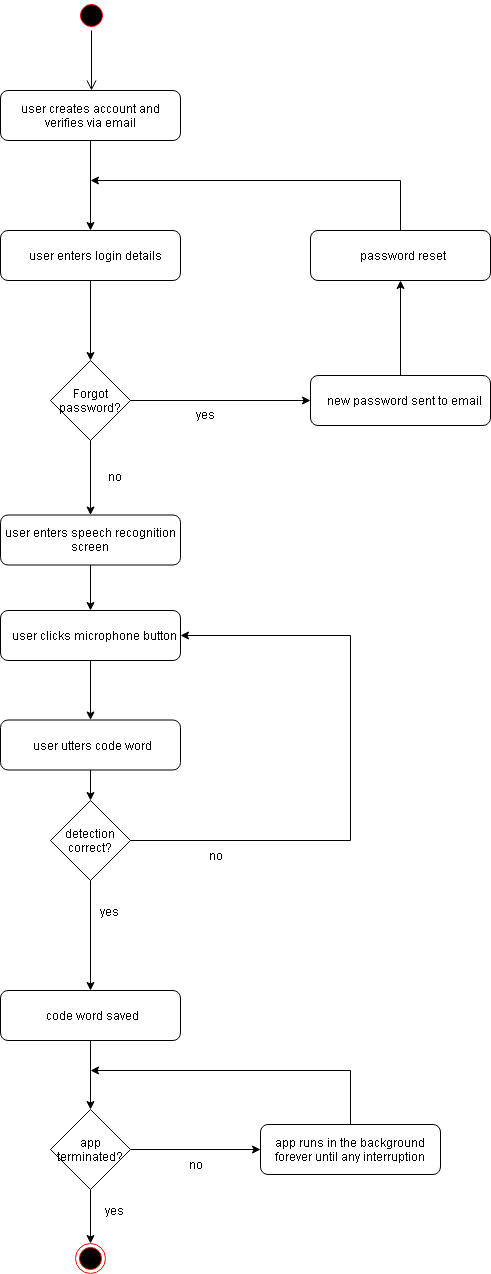
### **TensorFlow model**



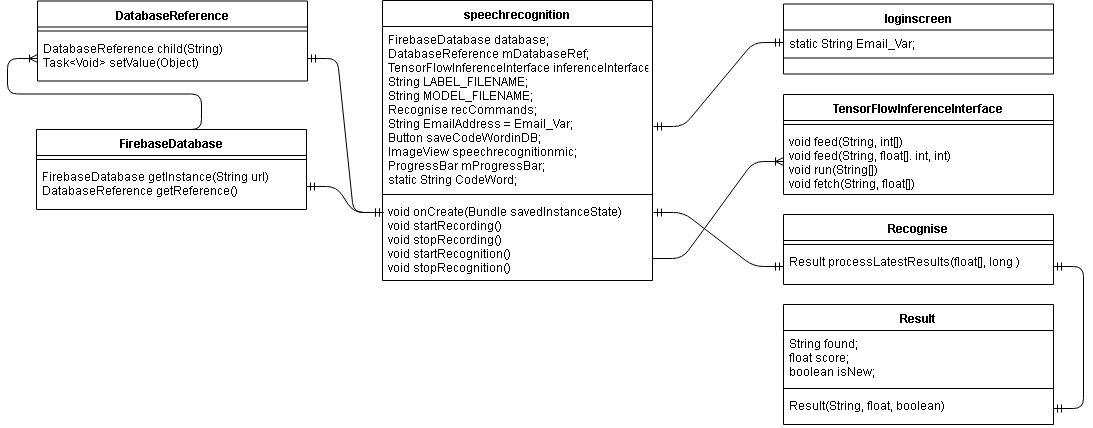
### **Loading model into Android Studio**

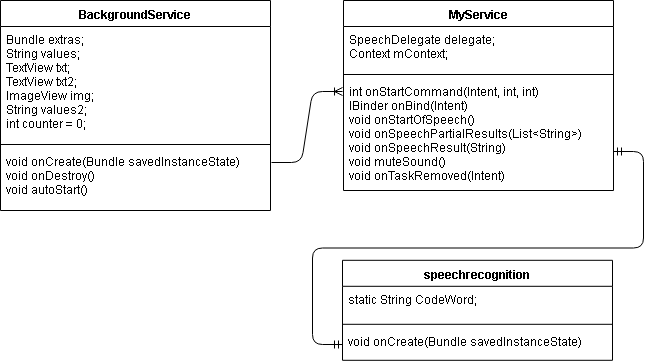


## **Activity diagram**

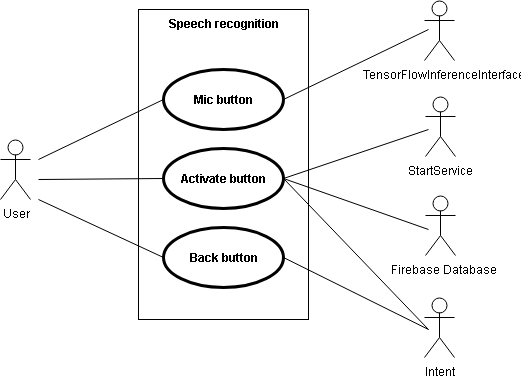


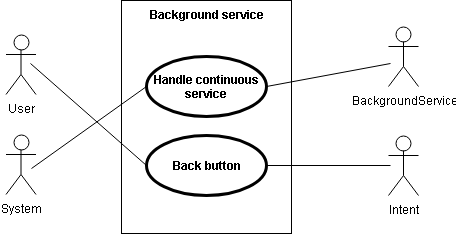
## **Class diagram**



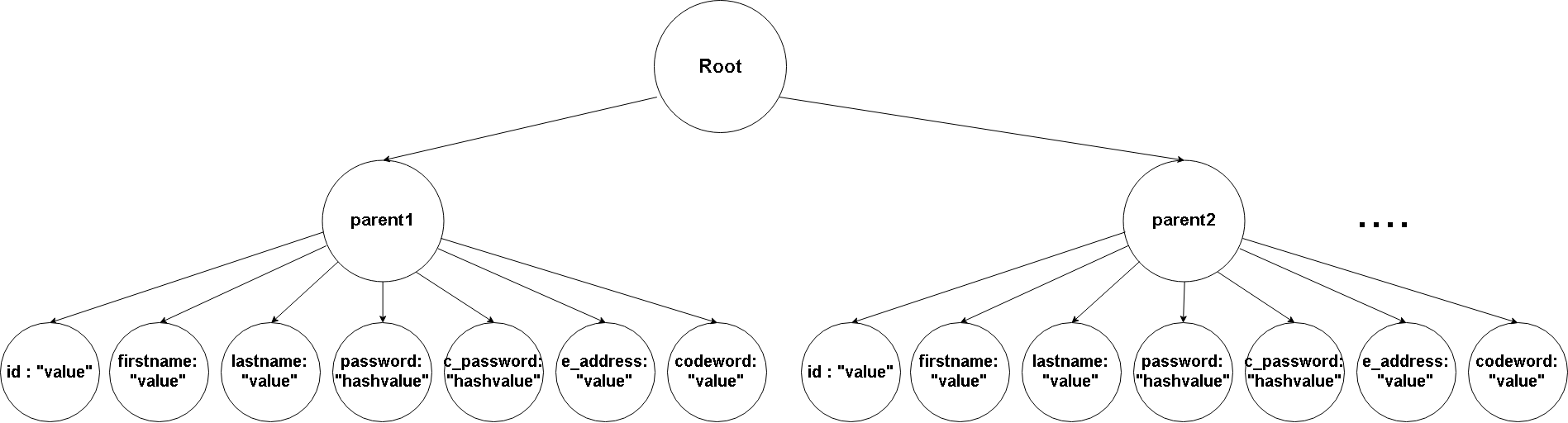


## **Use-case diagram**





## **Database schema**



## **Wireframes**

Done

## **Pseudocode**

Done

# **Implementation**

To do…

# **Testing and evaluation**

To do…

# **Conclusions and future work**

To do…

# **Appendices**

## **Design**

### Class diagram

### Activity diagram

## **Implementation**

### Class diagram

### Activity diagram

## **Testing**

### **Class diagram**

### **Activity diagram**

# **Reference list/bibliography**

**[1]** Erin Myers. "Little Known Facts About Speech Recognition Technology.” (2017). Retrieved from <https://www.temi.com/blog/little-known-facts-about-speech-recognition-technology/>

**[2]** Peter Samoff. "Alexa is getting better at answering users' questions.” (2018). Retrieved from <https://www.businessinsider.com/amazon-improving-alexa-voice-accuracy-2018-12?r=US&IR=T>

**[3]** Clark Boyd. "The Past, Present, and Future of Speech Recognition Technology.” (2018). Retrieved from <https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf>

**[4]** Sainath, Tara, and Carolina Parada. "Convolutional neural networks for small-footprint keyword spotting." (2015). *Keyword Spotting Task*, p. 1, *CNN architectures,* p. 2

**[5]** Graves, Alex, and Navdeep Jaitly. "Towards end-to-end speech recognition with recurrent neural networks." *International Conference on Machine Learning*, pp. 1764-1772. (2014). *Network Architecture,* p. 2

<https://arxiv.org/pdf/1804.03209.pdf>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7749004>

# **Evaluation**

Personal statement..