Eyenovative

Intelligent Solution for Visually Impaired

Project Report

By

Areesha Tariq

Table of Contents

TABLE OF FIGURES:

LIST OF TABLES

ABSTRACT

"Eyenovative" is a simple solution to the problems faced by the visually impaired population of the world. There is a population of 2.2 billion visually impaired people in the world and 90% of these people are located in developing countries like Pakistan. The biggest obstacles faced by the visually impaired are in navigation and in reading text. "Eyenovative" is a smart mobile application which caters towards the needs of people who have low vision or are even fully blind. The mobile application has four key features which help fulfill the needs of the visually impaired. The object detection feature helps with indoor navigation, optical character navigation feature is used for text reading, the voice command feature allows for a hands-free and hassle-free experience for the users and the text to speech feature is used to make the application more interactive. The existing solutions in the market do not provide all three features provided by "Eyenovative" with a simple and an easy to use interface.

Chapter 1

INTRODUCTION

"Eyenovative" is a smart mobile application that is used to help the visually impaired in their day to day life. There are 1.3 billion people in the world living with some sort of visual impairment and 87% of these people are located in developing countries like Pakistan.

The biggest obstacles faced by the visually impaired are in navigation and in reading text. As a solution to these problems we have made a self-contained product to be deployed on an android mobile application. Our product has been made to aid the people who have low vision or are even fully blind, in reading text and moving around obstacles.

Our application provides four key features to overcome the most common and tiresome obstacles faced by the visually impaired. The object detection feature helps with indoor navigation, the optical character recognition (OCR) feature helps with text reading, speech recognition feature is used in voice commands and finally the text to speech feature is used to make the application more interactive.

Apps have made life much easier for numerous people living with visual impairment or blindness. There are numerous useful apps present in today's market for the visually impaired. However, most of these apps do not provide all four features present in our product, "Eyenovative", with a smart and easy to use interface. Moreover, the accuracy provided by our application is unlike any free app present on the market.

1.1 Objectives

The objectives we aim to achieve using our product are:

- Provide a hands-free and easily accessible visual aid app to the blind or visually impaired population of the world.
- Assist the visually impaired in reading text present in visual print.
- A cheap and cost-effective solution for the problems faced by the impoverished visually impaired population,
- Providing indoor navigation which is often a tricky task.

1.2 Structure

The four main modules of our product are:

- **Object Detection:** Our product makes use of object detection technology to help the user navigate around obstacles by detecting any obstacles and objects which may be in the user's path. This feature can not only help the user in indoor navigation and allow them to walk around freely but it can also help the user identify the object in front of them.
- **OCR:** This technology is used in the text reading feature allows the user to listen to any printed text placed in front of the mobile camera. This feature can help the visually impaired people read text in visual print without the help of a sighted person.
- **Speech recognition:** The application is voice activated so audio input is taken from the user's microphone to navigate through the application. With the help of this feature the user can switch between the OCR and Object detection modules by simply giving a voice

command. This removes the extra hassle of looking at the mobile application and trying to navigate it manually. This feature further improves the user interface and the quality of the product.

- **Text to speech:** This function is used to give the user audio queues and is also useful in the object detection and text reading features of the application. The object detection module makes use of the Text to Speech feature to speak out the objects which are detected. Moreover, the OCR module uses this feature to read out the text which was detected.

Following the brief overview of the product and its features given in the introduction, in Chapter 2 a literature review is given where other similar products are discussed. In Chapter 3 the problem definition consisting of the problems which are product aims to tackle. In Chapter 4, the methodology used for our provided solution along with the tools and techniques used are discussed. In Chapter 5 detailed design and architecture of our application is discussed. In Chapter 6, the details of the product implementation are discussed along with the testing done on the product. In Chapter 7, the output and results of our implementation is discussed. Finally, in Chapter 8 the report is concluded and future improvements are discussed.

Chapter 2

LITERATURE REVIEW

There are many visual aid apps and technologies present in the market today which have made life easier for people living with visual impairment or blindness. Various machine learning algorithms are used to provide a smart solution and improve the accuracy of these solutions to the problems of the visually impaired.

As mentioned earlier, our project deals with object detection, OCR and speech recognition modules. There are few apps in the market which deal with all three modules. Therefore, when doing research for this project we also looked into apps and technologies which dealt with these modules separately.

2.1 Object Detection

Object detection is a computer vision technique used to locate objects in videos or images. In our project we have opted to use object detection techniques and algorithms in videos for real time interpretation. There are many hardware as well as software-oriented object detection tools present.

2.1.1 Hardware Approach

In the start of the project we explored a more hardware focused solution to the problems mentioned earlier. In many of the hardware-based solutions researched, an Arduino or other such microcontrollers were used to make smart glasses with object detection and text reading features.

R. Mohanapriya et al. [20] used an AT89C51 microcontroller along with an accelerometer, a UV sensor and a web cam to make smart glasses for the visually impaired with object detection features. This approach however, was not as efficient

and not as portable. Moreover, their product was not as cost effective and did not provide text reading feature.

Razu Miah et al. [18] made a similar obstacle detection product to aid the visually impaired in travelling. They made use of ultrasonic sensors and an Atmega328p microcontroller. This solution like the previous one was not quite as cost effective and did not fulfill all our requirements.

Jinqiang Bai et al. [2] presented a solution with obstacle avoiding feature smart glasses. Their product although portable, did not provide the user with a comfortable using experience.

None of the above-mentioned products are commercially manufactured, however, there are such product available on the market as well. The eSight electronic eyewear [8] is a product which can be used by visually impaired people. It works by using image processing techniques to enhance the visual input coming from a camera. This product however, cannot be used by blind people and is much too expensive for our target audience with a cost of $5950.

The OrCam MyEye [21] glasses are smart glasses with object detection as well as text reading features which related most closely to the requirements we were trying to fulfill. These glasses however, were once again, too expensive with a cost anywhere between $2,500 and $3,500.

Since many of the product discussed above were either too expensive, not portable and simply used resources we did not have access to, we decided to focus on a more software-oriented approach, with minimum hardware use.

2.1.2 Software Approach

While focusing on a more software-oriented approach we decided to develop an android application so that the processing of the visual input from the camera is done by the CPU of the user's mobile. This approach allowed us to make a more portable product which was inexpensive and easily accessible by people across the world.

There is a total of 3.5 billon people in the world who use smart phones and this number is predicted to increase over the coming years. Moreover, as of 2018 there are 27,730,000 smartphone users in Pakistan. This means that a significant amount of population across the world can have access to our application.

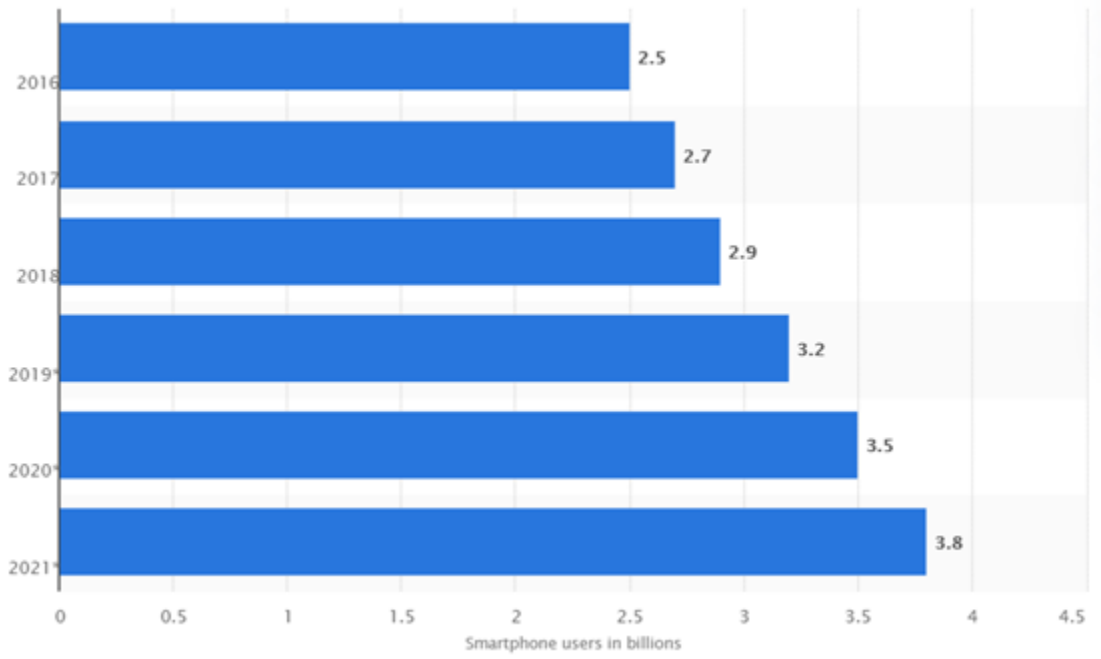


Figure 1: Smartphone users globally

There are many smartphone object detection related applications present. BeSpecular [4] is an application which connects the visually impaired with sighted

volunteers who can help them with their problems. BeMyEyes [3] is a similar mobile application.

Aipoly vision [1] is an object and colour recognizer available for both iOS and android. This app however, is not as accurate and does not provide text reading functionality.

Seeing AI [19] is an iOS app which provides the object detection and text reading feature. It is however only available on iOS. Similarly, iDentifi [14] is another app providing object detection and text reading functionality, however, it is not yet available on android. Moreover, these apps require internet access in order to function, our app does not require internet connectivity. Therefore, more people living in places with limited or no internet connection can also use our app.

Since there are 74.13% of the smartphone users in the world use android, we wanted to make an android application which would therefore, be accessible to a greater number of people in the world.

2.2 OCR

OCR or optical character recognition is the conversion of images of printed text into machine encoded text.

2.2.1 Hardware Approach

Esra Ali Hassan et al. [13] used a raspberry pi 2 single board computer and a raspberry pi camera to develop smart glasses for with text reading feature for visually impaired. These glasses however, did not provide the object detection nor the voice command feature.

OrCam MyEye [5] glasses, as discussed earlier, has the text reading feature, but it's expensive and not as accessible as a purely software solution can be.

2.2.2 Software Approach

The software available in the market for OCR are of varying levels of quality. The MD_evReader App [15] is a free iOS and android app which provides text reading functionality.

There are many digital book reading apps like Kindle [6] and DAISY Talk [22]. Such apps however, do not provide printed text reading functionality.

Chapter 3

PROBLEM DEFINITION

According to World Health Organization (WHO) there are 2.2 billion people globally, who have blindness or a visual impairment. One of the most common problems faced by the visually impaired is access to information. Reading a book or other such textual information can be quite a hassle. Even though, there is a wide variety of braille and audio books available, many people don't have access to such books.

87% of the visually impaired and blind population of the world lives in developing countries like Pakistan which means that they have limited access to the technologies which are more readily available to people living in first world countries.

Lack of education is another problem faced by the blind and visually impaired population of the world. Even in a developed country like America, almost 90 percent of blind children are not learning to read and write because they are not being taught Braille or given access to it.

The visually impaired often require assistance in navigation especially indoor navigation in unfamiliar places. This can limit their independence and can often limit their mobility. In this era of technology there aren't many cheap, easily accessible and easy to use devices or applications which can be used to assist the visually impaired.

Due to lack of education or limited mobility, the visually impaired are often unable to secure jobs. Many public work places don't provide aids for the visually impaired. This makes life for the visually impaired more difficult and isolated.

Since the majority of the visually impaired face problems with reading printed text and navigation, there needs to be an all in one solution to these issues. Many of the assistive technology for the visually impaired is either too expensive or isn't as accessible and easy to use for the visually impaired and blind population living in developing countries.

The solution that we aim to provide is an all-in-one application with text reading and object detection features. The object detection feature can be used for indoor navigation. Moreover, we aim to provide a cheap and portable solution which can be accessed by a wide range of people, in the form of an android mobile application.

As of 2017 only 15.5% population in Pakistan has stable internet connection. So, apps and devices which require steady internet can be rendered quite useless to a large percentage of population in Pakistan and other developing countries of the world. Our application does not require internet connectivity once it has been installed onto the user's device.

Moreover, 65 % of people visually impaired and 82% of all blind are 50 years and older. So, it is essential that the application has an easy to use interface. The speech recognition feature in or application makes it even easier for the user to navigate through our application.

Chapter 4

METHODOLOGY

4.1 Problem Analysis

When researching the topic for our project, we first wanted to take into account the most common problems the visually impaired people faced in their everyday life.

4.1.1 Online Studies

We therefore, first started by researching the studies done regarding the visually impaired globally. We read into studies done by the world health organization which gave us an idea on the global prevalence of visual impairment [26].

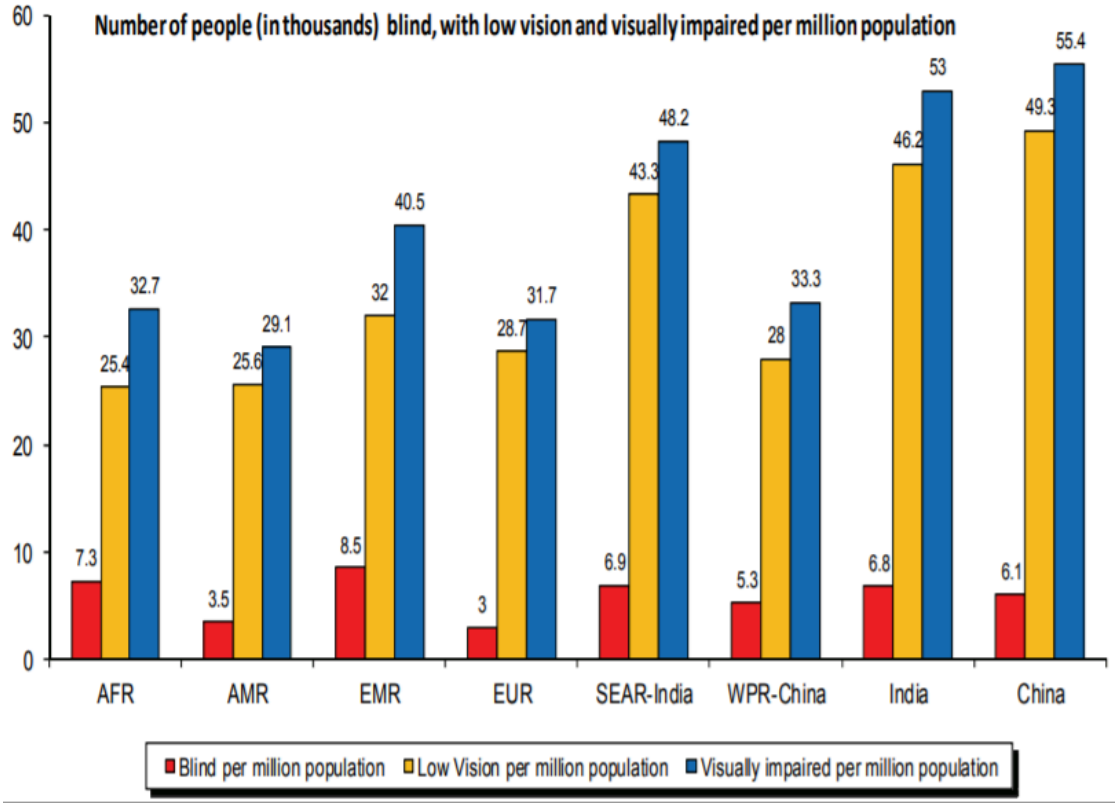


Figure 2: Distribution of visually impaired globally

Doing this research also gave us a good idea of what problems the visually impaired face globally, which we could target.

4.1.2 Surveys

We also wanted to know what problems the local visually impaired population of Pakistan but more in particular, Islamabad faced to get a better idea on what type of

product they would be willing to use. In doing so we met with local optometrists and asked them what problems their patients encounter the most.

This gave us a better idea on what kind of device the local visually impaired population would be willing to use. We decided to focus more on the indoor navigation and text reading problems faced by the visually impaired. This allowed us to narrow down the scope of our project.

4.2 Development Method

4.2.1 Initial Approach

When first developing our product, we went towards a more hardware-oriented approach. We planned on developing smart glasses with a webcam attached to a pair of glasses as shown in the figure below:

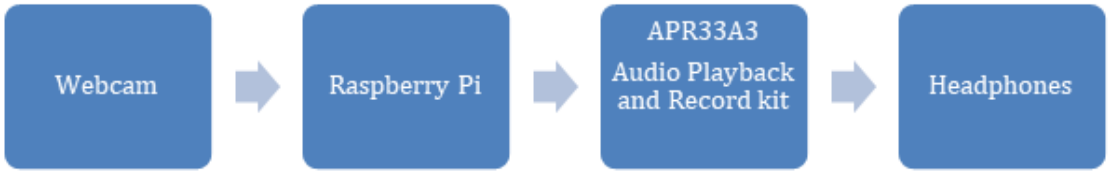


Figure 3: Block diagram of initial approach

Input from the webcam would be processed by the raspberry pi 2 controller which would have our OCR and object detection modules stored in it. The APR33A3 audio playback and record kit would be used to play the audio output from the raspberry pi into the user's headphones.

4.2.1.1 OCR Module

When developing the OCR module, we used Tesseract OCR to extract text from the image.

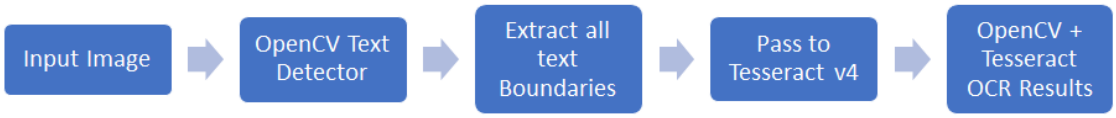


Figure 4: Initial Approach for OCR Module

Initially, we will use Google Tesseract and OpenCV for OCR. Tesseract is highly popular OCR engine that uses two pass approach to character recognition and can be trained to recognize new fonts. It also supports (v4) deep-learning based OCR providing more accuracy. We used MaryTTS to read out the extracted text to the user.

4.2.1.2 Object Detection Module

We used You Only Look Once: Unified, Real-Time Object Detection system and OpenCV to develop our object detection module [15]. YOLO is a solitary neural network which predicts bonding boxes and class probabilities directly from full pictures in a single assessment. Since the entire detection pipeline is a solitary network, directly on detection performance it can be improved from end-to-end.

YOLO model can process pictures at 45 frames every second in real-time. Compared with other cutting-edge recognition frameworks, YOLO makes more localization mistakes yet is far less inclined to give false positives where nothing exists.

Following figure describes our approach:

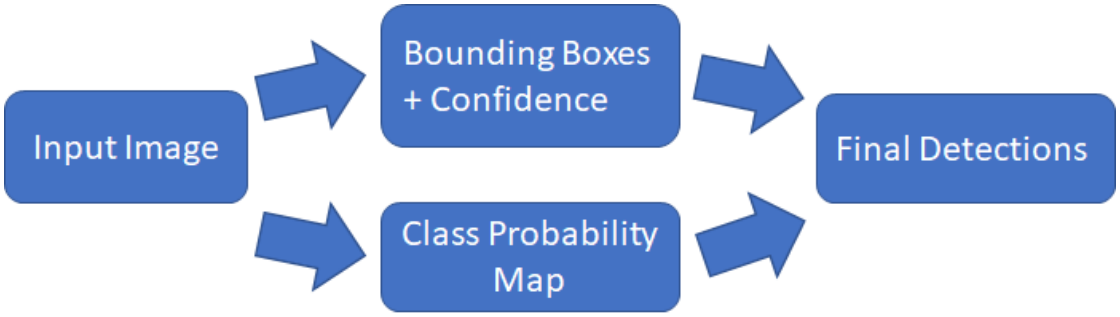


Figure 5: Initial approach of object detection module

We however, chose not to go with this initial hardware-based approach. We soon realized that the hardware components in our initial approach would make the product less portable and less efficient. Moreover, acquiring the hardware components like the microcontroller which could provide the processing power needed for our project was proving to be hard.

We also wanted to make our product inexpensive and accessible to a wide variety of people across the globe. Due to these reasons we ended up going with a more software focused approach.

4.2.2 Software Based Approach

When shifting our attention to a software-based solution, we decided to make an android application. The android application would allow us to use the user's smart phone processor which is much faster than the average microcontroller and is more easily accessible. Moreover, as mentioned in the first chapter of this report, there are 3.5 billon smart phone users in the world. Which means that a smartphone application would be much more easily available to people around the world. Also, by opting to focus on a software-based solution we could cut the building cost of our product, all the while improving the efficiency and performance of our product.

The high-level view of our solution can be explained in the following figure:

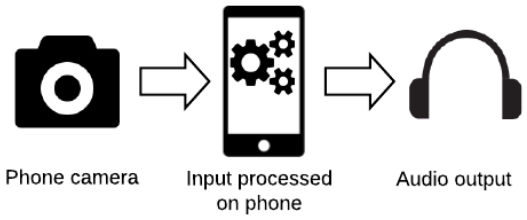


Figure 6: High level block diagram

The real time input is taken from the mobile's camera. This input is processed by our mobile app's object detection or OCR module, depending on which module the user has open. The audio output from the object detection module makes the user aware of what obstacle may be in front of them, therefore helping with indoor navigation. Audio output from the OCR module reads out the printed text which the user has place in front of the camera.

We used android studio to develop our application, using Java programming language. Android studio provides an easy to use interface and there is plenty of documentation available for it online.

4.2.2.1 First Prototype

Our We divided our project into four parts when developing our mobile application; the object detection module, the speech recognition module, the text to speech module and the OCR module. The user can operate the application by voice commands. If the user wants to open the OCR module, he/she can say "Open text reading" into the mobile's microphone. To open the object detection module, they can say "Open object detection".

Each of the object detection and OCR modules make use of the device's main camera. The input from the camera is processed and the audio output is conveyed to the user through the headphones or the mobile's speakers. Details of the processing done in each module is explained later.

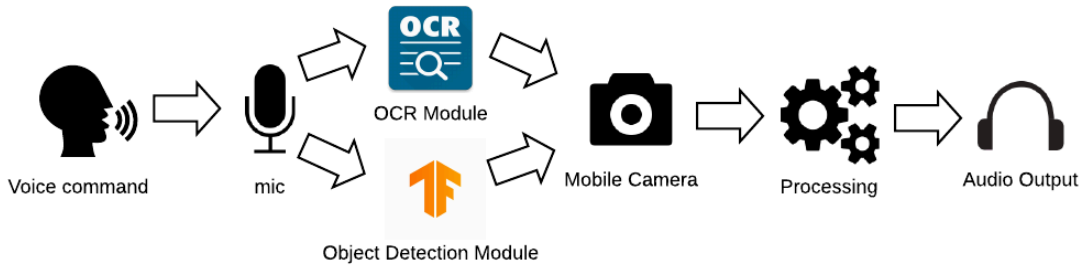


Figure 7: Block diagram of final prototype

1. **Object Detection Module**

Using a quantized MobileNet SSD model trained on the COCO dataset for the object recognition app, this continually identifies the artifacts (bounding boxes and classes) in frames captured by your device's back camera.

An entity detection model is trained to identify the existence and the location of several entity types.For eg, a model can be equipped with images comprising various pieces of fruit along with a mark identifying the form of fruit they represent (e.g. an apple, a banana, or a strawberry), and data showing where each item occurs on the image.

It will then emit a list of the items it identifies, the location of a bounding box for each object, and a score showing the trust that the detection was accurate.

```
for (final Classifier.Recognition result : results) {
  final RectF location = result.getLocation();
  if (location != null && result.getConfidence() >= minimumConfidence) {
    canvas.drawRect(location, paint);

    cropToFrameTransform.mapRect(location);

    result.setLocation(location);
    mappedRecognitions.add(result);
//    System.out.println(Arrays.asList(result).indexOf(2));
//    System.out.println(result);

    String[] labelAndBoundingBoxes = String.valueOf(result).split( regex: " ");

    labels.add(labelAndBoundingBoxes[1]);


  }
}
```

Figure 8: Object detection bounding box code

2. OCR Module

A Text Recognizer was implemented in mobile vision text API which provided a framework for detecting text in video. The text recognizer uses the following:

- **Camera Source:** To get the live feed of the text in the form of video
- **Processor:** To detect text directly from the video instead of detecting text as individual frames from images

Images are processed through the detector object. TextRecognizer can be used to recognize text from images and videos only once it is initialized. In the beginning, we could only detect text from images but what we wanted for our application was to detect text from real-time video. To get the live feed in the form of a video, a camera source is created. It is a camera manager and should be configured for vision processing.

Resolution is set to high and autofocus is turned on so that small text can also be recognized. Though low resolution would be able to process more

frames in less time but since our user might be looking at smaller text, therefore high resolution is required. After building the application at this point, we were able to get a live camera view. But we also wanted our optical character recognition module to handle text in real-time, which means that it should be able to detect and recognize text as soon as the text appears in the video from live camera view. For this, we implemented a Processor.

3. Speech Detection Module

The aim of the speech detection module was to provide a hands-free experience for the user. Using this module, the user could open the OCR or the object detection module by simply giving “Open text reading” or “Open navigation” voice commands.

We used the Android SpeechRecognizer API to build the speech module. The SpeechRecognizer API streams audio to remote servers in order to perform speech recognition. This feature allowed us to conduct speech recognition offline. It was very important for us to make our application so that it can work offline. The interface of the speech

The SpeechRecognizer API started running as soon as the user opened the application. However, this API cannot perform continuous recognition. This means once the OCR or object detection module had been opened, the SpeechRecognizer API stopped running. So, the user couldn’t switch from the OCR module to the object detection module and vice versa by giving voice commands.

Moreover, the SpeechRecognizer API does not have hotword or wake word detection functionality. Through hotword detection the device can listen to specific predetermined key words to activate the application's speech recognition interface.

Continuous listening along with hotword detection are two very important components which were needed to switch between the OCR and object detection modules thereby making the application completely hands-free.

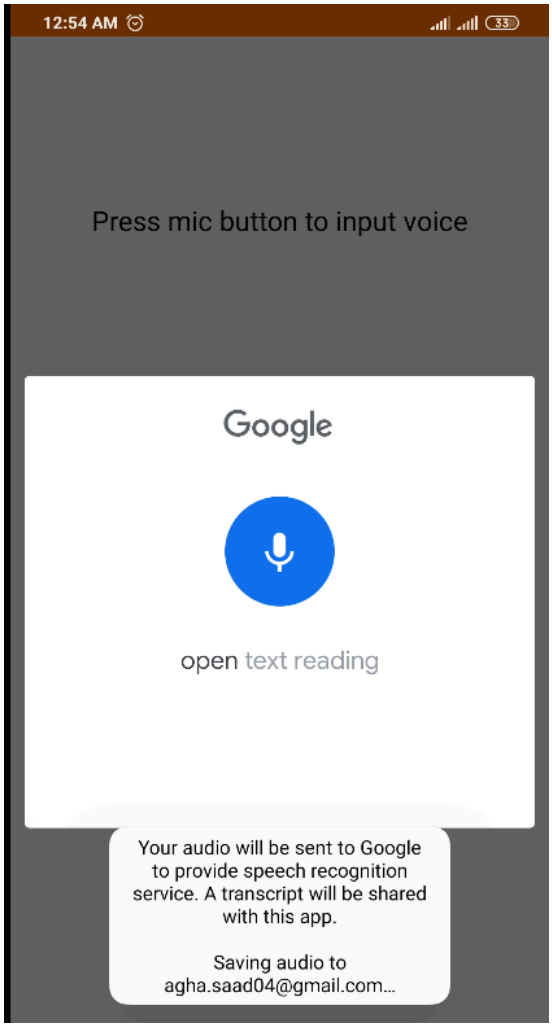


Figure 9: Output of first prototype speech module

4. **Text to Speech Module**

The text to speech module is used to convert the text output from the object detection and OCR modules to speech thereby making the application interactive. Once an object has been detected by the object detection module, the text to speech module is used to make the user aware of the obstacle in front of them.

Similarly, the text to speech module is used to read out the text processed by the OCR module. This allows the user to listen to the book, newspaper or any other printed text.

The android TextToSpeech class was used to implement the text to speech module. The TextToSpeech class is a part of the TTS API and has been supported by all android 1.6+ versions.

4.2.2.2 Final Prototype

In the final prototype certain new functionalities were added which were previously absent. Moreover, the accuracy of the modules was increased. Details of the changes made are given below.

1. **Object Detection Module**

   Using a quantized MobileNet SSD model trained on the COCO dataset for the object recognition app, this continually identifies the artifacts (bounding boxes and classes) in frames captured by your device's back camera.

   An entity detection model is trained to identify the existence and the location of several entity types.For eg, a model can be equipped with images comprising various pieces of fruit along with a mark identifying the form of

fruit they represent (e.g. an apple, a banana, or a strawberry), and data showing where each item occurs on the image.

We have added Euclidean distance to make the object detection more accurate. We compare the 10 consecutive frames, if frames in all frames have the same object then the app reads out the object after reading it checks if the object is still in the frames then we compare the centers using euclidean distance.

2. OCR Module

No new changes were made to the text to speech module since the previous version was working adequately.

3. Speech Recognition Module

The problem of continuous listening and hotword detection was solved in the final prototype. We researched other speech recognition engines which are compatible with android. Snowboy hotword and wake word detection toolkit provided the functionality which we required, however, due to lack of documentation of Snowboy on Android, we decided not to use it.

Droid Speech is another Android library which gives continuous speech recognition. It supports Android SDK versions 16+. However, we did not end up using it since parts of it are not configurable.

Another speech recognizer system is CMU PocketSphinx. PocketSphinx is a lightweight version of CMU Sphinx for handheld devices. CMU Sphinx is a group of speech recognition systems made by the Carnegie Melon University.

PocketSphinx is free to use, opensource engine designed specifically for mobile and other handheld devices. The PocketSphinx engine allows developers to customize the dictionary. This feature allowed us to add a new word “Eyenovative” to the dictionary which was later used as a hotword or a wake word. We used “Wake-up Eyenovative” hotword to activate the application and have it start listening for key phrases.

Also, by being able to customize the dictionary, we could further improve the accuracy of the speech recognizer module. We removed many words which weren’t being used or didn’t resemble the as hotwords or key phrases, from the dictionary. This allowed us to improve the accuracy of our speech recognition module.

Moreover, the PocketSphinx engine allowed us to develop a custom menu grammar file. The menu grammar file contains special key phrases which are used as voice commands to open OCR and object detection module.

“Open object detection” and “Open navigation” key phrases can be said after saying the hotword “wake-up Eyenovative”, to open the object detection module. Similarly, saying “Open text reading” or “Open book reading” after saying the hotword can open the OCR module.

Furthermore, the hotwords can be said while the OCR or the object detection module is running to activate speech recognition and key phrases can then be said to switch between the modules as shown in the figure below.

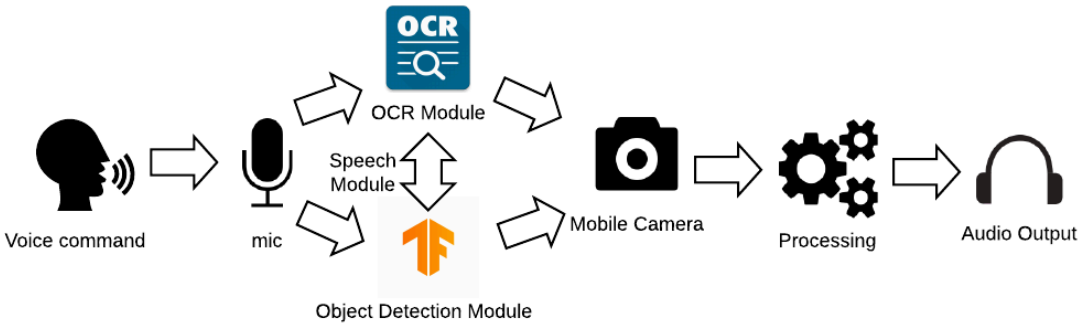


Figure 10: Final prototype block diagram

4. Text to Speech Module

No new changes were made to the text to speech module since the previous version was working adequately.

Chapter 5

DETAILED DESIGN AND ARCHITECTURE

5.1 System Architecture

5.1.1 Architecture Design Approach

The whole system is based on an Android Application. The architecture of the overall system includes the following four main modules:

- Object Detection
- Speech Module
- Optical Character Recognition
- Camera Module

These modules mainly make up this system and perform the core functionalities.

As our application comprises multiple features and functionalities, a separate module was developed for every functionality. The modules are very much closely related to each other. Speech module is used to open the required module (object detection or optical character recognition module). Speech module uses voice input to switch to either module. The commands that are used in our application are: ‘Wakeup Eyenovative’. By this command, the speech module switches to the listening mode. The user can then use the command ‘Open Object Detection’ or ‘Open Text Reading’ to open the object detection or optical character recognition module respectively. Speech module is also used to switch from one module to the other.

Both object detection module and optical character recognition module uses camera source to get live feed of the surroundings and text. The block diagram below

depicts the overall system architecture. It also shows how the modules collaborate with each other to achieve functionality. All the modules are assigned separate responsibilities but are interconnected to achieve complete working of the system.

For Example, the object detection module cannot be opened without the speech module and since the detected objects cannot be seen by the user (blind person), hence without the text to speech module, the results of the object detection module would be useless to the user.

5.1.2 Architecture Design

The figure depicts the overall architecture of the system. To achieve the complete functionality of the system, the responsibilities were partitioned and assigned to subsystems. How the subsystems collaborate and interconnect can also be seen.

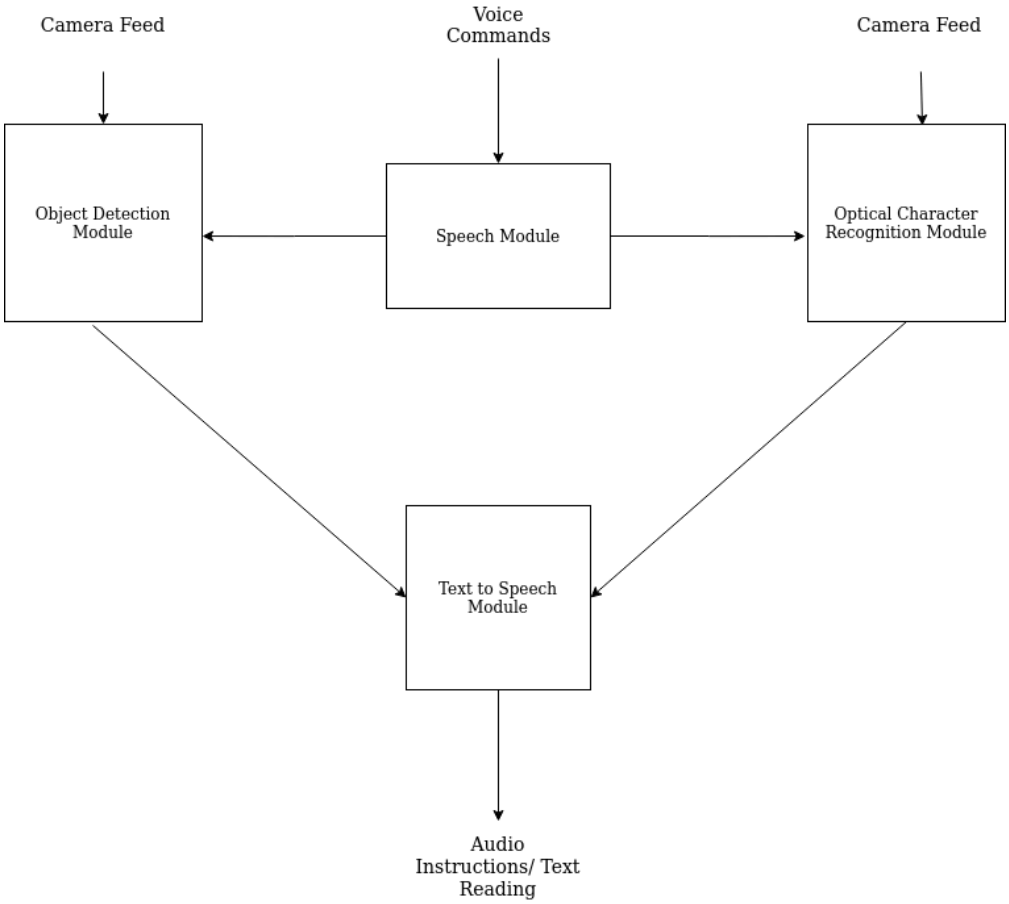


Figure 11: Architecture of the system

5.1.3 Subsystem Architecture

- **Object Detection Module**

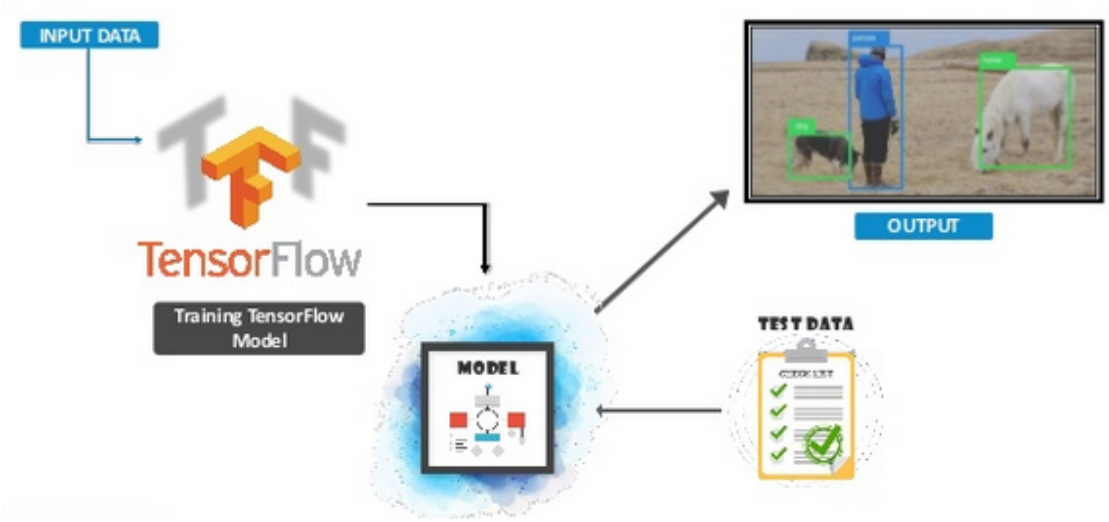


Figure 12: Architecture of Object Detection Module

- **Optical Character Recognition Module (OCR)**

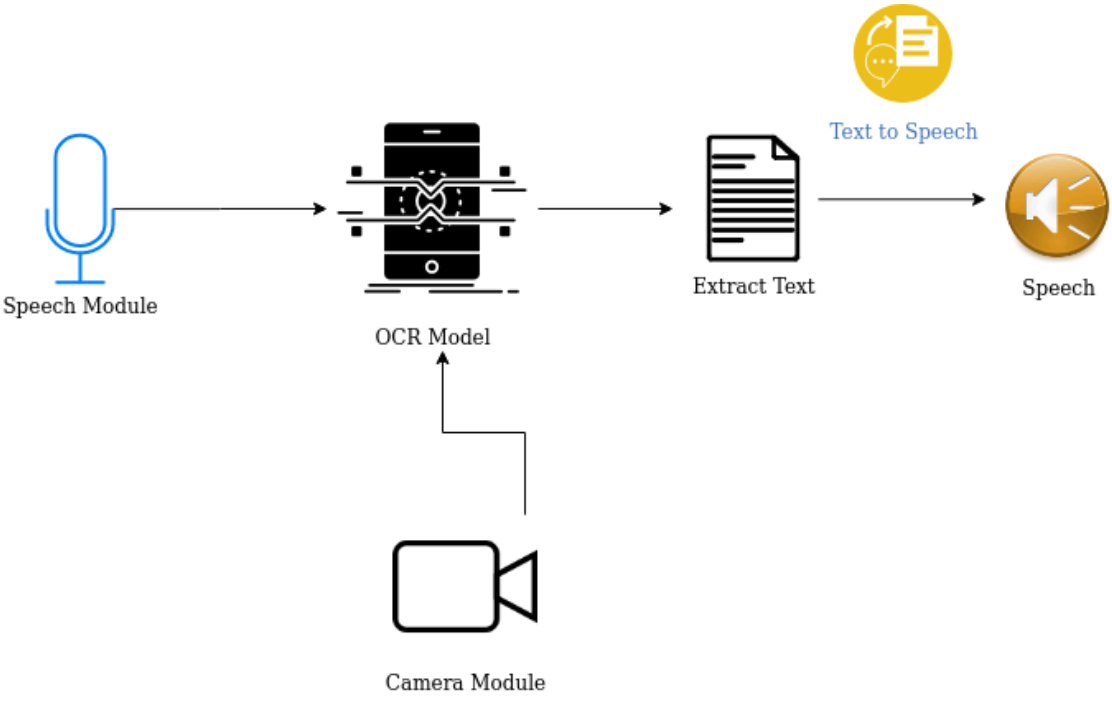


Figure 13: Architecture of OCR Module

- **Speech Recognition Module**

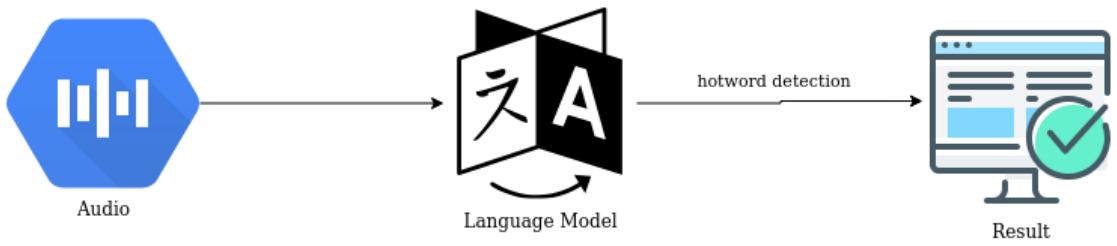


Figure 14: Architecture of Speech Recognition Module

The results can vary depending on the input audio:

- The application comes to listening state
- Object Detection Module can open
- Optical Character Recognition Module can open

- **Text to Speech Module**

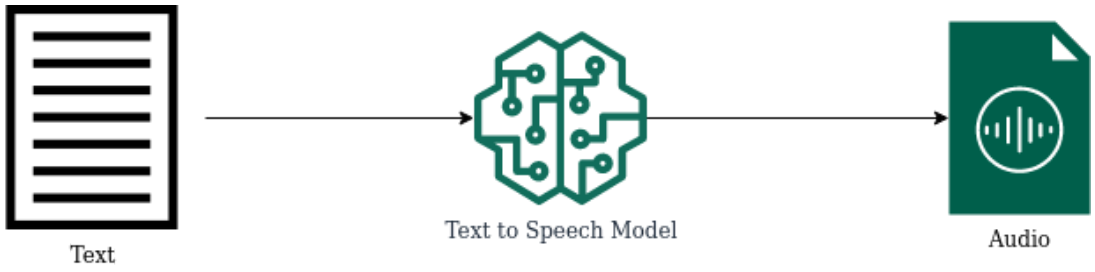


Figure 15: Architecture of Text to Speech module

5.2 Detailed System Design

This section contains the detailed specifications of each component in the system architecture.

5.2.1 Classification

There are four main modules in the overall system:

- Object Detection
- Speech Recognition
- Optical Character Recognition
- Text to Speech

5.2.2 Definition

- **Object Detection Module**

  This module helps the visually impaired to navigate through the obstacles. The user gets the directions in audio to make navigating around more seamless and effortless. Our product provides real-time object recognition.

- **Optical Character Recognition Module (OCR)**

  This module reads the text and converts it into audio and the user can hear it through the attached hands-free (text to speech).

- **Speech Recognition Module**

  The application operates through voice commands. The user can open the application and switch between modules using voice commands, thus providing ease of use. Our product provides affordability and a one-stop solution to people who are visually impaired.

- **Text to Speech Module**

The text to speech module is to guide the user. As the user can't see the instructions, it's important to read them out to the user to fulfill the purpose of this application. This module is also used by object detection and optical character recognition modules.

5.2.3 Responsibilities

- **Object Detection**

  The object detection feature detects any obstacles and objects which may be in the user's path. This feature can help the user in indoor navigation and allow them to walk around freely. This feature can also help the user identify the object in front of them.

  The object detection module will make use of the Text to Speech feature to speak out the objects which are detected, to the user's headphones. This feature has a high priority and is a core feature of the product.

- **Speech Module**

  This feature is used to convert text output from the object detection and OCR module into speech which the user can listen to. This module is used by both the OCR and object detection module and is an essential part of the product. It further improves the user interface and fulfills the needs of the visually impaired, who are the target audience of this product.

- **Optical Character Recognition**

  The optical character recognition (OCR) module is used to read out printed text to the user. This is a core feature of the product and with the help of this feature the user can switch between the OCR and Object detection

modules by simply giving a voice command. This removes the extra hassle of looking at the mobile application and trying to navigate it manually. This feature further improves the user interface and the quality of the product.

- **Text to Speech Module:**

  This module is used to give the user audio queues and is also useful in the object detection and text reading features of the application. The object detection module makes use of the Text to Speech feature to speak out the objects which are detected. Moreover, the OCR module uses this feature to read out the text which was detected.

5.2.4 Constraints

Our product design and implementation are limited by the following constraints:

- **Object Detection Module**

  Dataset: At this stage, the dataset that we are using is COCO Dataset which is trained on 80 objects.

- **Optical Character Recognition Module**

  Handwritten-text: For now, the OCR only detects computer printed text with high accuracy, but is not able to detect hand-written text.

  Language: The OCR only extracts and reads the text written in English. We would like to train it on Urdu as well to make it more inclusive for local people.

- **Camera Module**

  Built-in Camera: Due to lack of resources, we have used an Android built-in camera instead of a mini wireless camera.

- **Application Deployment:**

  Due to lack of resources and a slight time constraint, mobile application that is developed is for Android OS only.

5.2.5 Composition

- **Object Detection Module**

  Object detection module is composed of SSD Mobile Net trained on COCO dataset. The architecture of SSD is such that a convolution network is created for the prediction of bounding box locations. These locations are then classified in one pass. The base architecture of the SSD network is MobileNet. SSD network is then followed by multiple convolution layers. Every bounding box carries the following information: [16]

  - Class Probability
  - 4 Corner Locations

- **Optical Character Recognition Module**

  A Text Recognizer was implemented in mobile vision text api which provided a framework for detecting text in video. The text recognizer uses the following:

  - **Camera Source:** To get the live feed of the text in the form of video

- **Processor:** To detect text directly from the video instead of detecting text as individual frames from images

- **Speech Module**

Speech Module uses PocketSphinx. PocketSphinx is a lightweight version of CMU Sphinx for handheld devices. We built a language model and dictionary in which we added hotwords such as “Wake-up Eyenovative” to activate the application and have it start listening for key phrases. “Open object detection” and “Open navigation” key phrases can be said after saying the hotword “wake-up Eyenovative”, to open the object detection module. Similarly, saying “Open text reading” or “Open book reading” after saying the hotword can open the OCR module.

- **Text to Speech Module**

The basic composition of the text to speech module is the android speech text to speech engine which comprises of the following: [9]

- Listener
- Mapping of text strings to sound
- Pronunciations
- Language
- Voice
- Pitch
- Silence
- Speech Rate

5.2.6 Uses/Interactions

- **Object Detection Module**

  Object Detection Module is opened using the Speech Module when the user says ‘Open Object Detection’ after ‘Wake up Eyenovative’. Once the module has opened, it gets the live feed of the surroundings through a camera source. When objects are detected, it uses the Text to Speech module to guide the user to navigate through the obstacles.

- **Optical Character Recognition Module**

  Optical Character Recognition Module is opened using the Speech Module when the user says ‘Open Text Reading’ after ‘Wake up Eyenovative’. Once the module has opened, it gets the live feed of the text that is to be read through a camera source. When the text is detected and extracted, it uses the Text to Speech module to read the text to the user.

- **Speech Module**

  The speech module is used to open the object detection module and optical character recognition modules through hotword detection. It wakes up the application by the key phrase ‘Wake-up Eyenovative’. It is also used to switch between the modules once any module is opened.

- **Text to Speech Module**

  Text to Speech module is used to convert the results of object detection module and optical character recognition module into audio. This module reads

the extracted text to the user and informs the user about the detected objects in the form of speech.

5.2.7 Resources

- **Object Detection Module**

  The implementation of Object Detection module has been done in:

  - TensorFlow Lite
  - Java

- **Optical Character Recognition Module**

  The implementation of Optical Character Recognition module has been done in:

  - Java
  - Mobile Vision Text API

- **Speech Module**
  - CMU PocketSphinx
- **Text to Speech Module**
  - Text to Speech API

5.2.8 Processing

- **Object Detection Module**

  In the label package, images were annotated. These images were the ones that were used for the model testing. Once the images were annotated, 90% images were used for the training purpose, while the remaining 10% images were used for the testing purposes. Images are stored along with them

.xml files. A label map is required by TensorFlow in which the used labels are mapped to integer values such as:

```
item {
   id: 1
   name: 'cat'
}
item {
   id: 2
   name: 'dog'
}
```

These label maps are stored as .pbtxt files. The images had been annotated, and the dataset had been splitted into training and testing subsets, the next step was to create tensorflow records. Two steps were done for this task:

- For each dataset, convert .xml files to .csv files
- Convert .csv files to .record files (TensorFlow Record format)

Then, the training pipeline was configured. A pre-trained model was used and we later applied transfer learning for the purpose of training new objects.

- **Optical Character Recognition Module**

Initially, a TextRecognizer is created. Images are processed through the detector object. TextRecognizer can be used to recognize text from images and videos only once it is initialized. In the beginning, we could only detect text from images but what we wanted for our application was to detect text from real-time video. To get the live feed in the form of a video, a camera source is created. It is a camera manager and should be configured for vision processing. [24]

Resolution is set to high and autofocus is turned on so that small text can also be recognized. Though low resolution would be able to process more frames in less time but since our user might be looking at smaller text, therefore high resolution is required.[5] After building the application at this point, we were able to get a live camera view. But we also wanted our optical character recognition module to handle text in real-time, which means that it should be able to detect and recognize text as soon as the text appears in the video from live camera view.[10] For this, we implemented a Processor.

```
public class OcrDetectorProcessor implements Detector.Processor<TextBlock> {
    private GraphicOverlay<OcrGraphic> graphicOverlay;

    OcrDetectorProcessor(GraphicOverlay<OcrGraphic> ocrGraphicOverlay) {
        graphicOverlay = ocrGraphicOverlay;
    }
}
```

Figure 16: Code Snippet of OCR Processor

- **Speech Module**

We used PocketSphinx which is free to use, open source engine designed specifically for mobile and other handheld devices. The PocketSphinx engine allows developers to customize the dictionary. This feature allowed us to add a new word “Eyenovative" to the dictionary which

was later used as a hotword or a wake word. We used the “Wake-up Eyenovative” hotword to activate the application and have it start listening for key phrases.

Also, by being able to customize the dictionary, we could further improve the accuracy of the speech recognizer module. We removed many words which weren’t being used or didn’t resemble the as hotwords or key phrases, from the dictionary. This allowed us to improve the accuracy of our speech recognition module.

Moreover, the PocketSphinx engine allowed us to develop a custom menu grammar file. The menu grammar file contains special key phrases which are used as voice commands to open OCR and object detection modules.

“Open object detection” and “Open navigation” key phrases can be said after saying the hotword “wake-up Eyenovative”, to open the object detection module. Similarly, saying “Open text reading” or “Open book reading” after saying the hotword can open the OCR module.

Furthermore, the hotwords can be said while the OCR or the object detection module is running to activate speech recognition and key phrases can then be said to switch between the modules

- **Text to Speech Module**

Speech is synthesized so that it is instantly playedback. The first step is the initialization, only after which the speech can be synthesized. The constructor of the TextToSpeech class initializes the TTS engine. After the initialization of the TTS engine, the listener can be called.[11] Public method is used to add a mapping that associates strings of text with sound resources.

Custom pronunciations are also added. Sounds can be synthesized even if the associated sound resource is missing for a text.

The language that we are currently using is English. A defaut voice is also added for speaking text out. If multiple text strings are to be converted into speech, the TTS engine is indicated to be busy speaking one string. Once the speech is sent to the audio mixer, only then it is considered to be complete. We also use a public method to add silence for a specified time between speech data. Speed pitch is set in the setPitch() method. We have kept the normal 1.0 speech rate in the method setSpeechRate().

5.2.9 Interface/Exports

- **Object Detection Module**

Below is shown the code snippet for when the module fetches the labels for the resultant object array:

```
for (final Classifier.Recognition result : results) {
  final RectF location = result.getLocation();
  if (location != null && result.getConfidence() >= minimumConfidence) {
    canvas.drawRect(location, paint);

    cropToFrameTransform.mapRect(location);

    result.setLocation(location);
    mappedRecognitions.add(result);
//    System.out.println(Arrays.asList(result).indexOf(2));
//    System.out.println(result);

    String[] labelAndBoundingBoxes = String.valueOf(result).split( regex: " ");

    labels.add(labelAndBoundingBoxes[1]);


  }
}
```

Figure 17: Code Snippet of label fetching in object detection module

Below is shown the code snippet for the integration of object detection module with the text to speech module.

```
textToSpeech = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
  @Override
  public void onInit(int status) {
    if (status == TextToSpeech.SUCCESS) {
      int ttsLang = textToSpeech.setLanguage(Locale.US);
      for (String labels_speak : labels_array_to_speak){
        String data = "There is a "+labels_speak+"in front of you";
        tts(data);
        if (ttsLang == TextToSpeech.LANG_MISSING_DATA
              || ttsLang == TextToSpeech.LANG_NOT_SUPPORTED) {
          Log.e( tag: "TTS",  msg: "The Language is not supported!");
        } else {
          Log.i( tag: "TTS",  msg: "Language Supported.");
        }
        Log.i( tag: "TTS",  msg: "Initialization success.");
      }
    } else {
      Toast.makeText(getApplicationContext(),  text: "TTS Initialization failed!", Toast.LENGTH_SHORT).show();
    }
  }
});
```

Figure 18: Code snippet of integration of object detection module with the text to speech module

- **Optical Character Recognition Module**

We created camera source:

The parameters that were passed to the camera source were autoFocus and useFlash as boolean.

```
private void createCameraSource(boolean autoFocus, boolean useFlash) {
    Context context = getApplicationContext();
    // A text recognizer is created to find text.  An associated processor instance
    // is set to receive the text recognition results and display graphics for each text block
    // on screen.
    TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();
    textRecognizer.setProcessor(new OcrDetectorProcessor(mGraphicOverlay));
    if (!textRecognizer.isOperational()) {

        Log.w(TAG,  msg: "Detector dependencies are not yet available.");

        IntentFilter lowstorageFilter = new IntentFilter(Intent.ACTION_DEVICE_STORAGE_LOW);
        boolean hasLowStorage = registerReceiver( receiver: null, lowstorageFilter) != null;

        if (hasLowStorage) {
            Toast.makeText( context: this, "Ocr dependencies cannot be downloaded due to low device...",
            Log.w(TAG, "Ocr dependencies cannot be downloaded due to low device...");
        }
    }
    mCameraSource =
            new CameraSource.Builder(getApplicationContext(), textRecognizer)
            .setFacing(CameraSource.CAMERA_FACING_BACK)
            .setRequestedPreviewSize( width: 1280,  height: 1024)
            .setRequestedFps(2.0f)
            .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH : null)
            .setFocusMode(autoFocus ? Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE : null)
            .build();
}
```

Figure 19: Code snippet of creating camera source in OCR module

We wanted our optical character recognition module to handle text in real-time, which means that it should be able to detect and recognize text as soon as the text appears in the video from live camera view. For this, we implemented a Processor.

```
public class OcrDetectorProcessor implements Detector.Processor<TextBlock> {

    private GraphicOverlay<OcrGraphic> mGraphicOverlay;

    OcrDetectorProcessor(GraphicOverlay<OcrGraphic> ocrGraphicOverlay) {
        mGraphicOverlay = ocrGraphicOverlay;
    }

    /**
     * Called by the detector to deliver detection results.
     * If your application called for it, this could be a place to check for
     * equivalent detections by tracking TextBlocks that are similar in location and content from
     * previous frames, or reduce noise by eliminating TextBlocks that have not persisted through
     * multiple detections.
     */
    @Override
    public void receiveDetections(Detector.Detections<TextBlock> detections) {
        mGraphicOverlay.clear();
        SparseArray<TextBlock> items = detections.getDetectedItems();
        for (int i = 0; i < items.size(); ++i) {
            TextBlock item = items.valueAt(i);
            OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
            mGraphicOverlay.add(graphic);
        }
    }

    /**
     * Frees the resources associated with this detection processor.
     */
    @Override
    public void release() { mGraphicOverlay.clear(); }
}
```

Figure 20: Code snippet of implementing live camera view in OCR module

- **Speech Module**

Here is the code snippet for the waking up of the application (listening mode) and switching of the modules:

```
@Override
public void onPartialResult(Hypothesis hypothesis) {
    if (hypothesis == null)
        return;

    String text = hypothesis.getHypstr();
    if (text.equals(KEYPHRASE)){
        texttospeechfunction( data: "hi!");
        switchSearch(MENU_SEARCH);
    }
    else if (text.equals(OBJECTDETECTION)){
        OpenObjectDetection();
        switchSearch(OBJECTDETECTION);
    }
    else if (text.equals(OCR)){
        OpenOCR();
        switchSearch(OCR);
    }

    else{
        ((TextView) findViewById(R.id.result_text)).setText(text);
    }
}
```

Figure 21: Code snippet of module switching in Speech module

5.2.10 Detailed Subsystem Design

- **Object Detection Module:**

  The basic workflow principles of the object detection module are:

  - Training Data
  - Feature Extraction
  - Classification
  - Testing Data

First, a deep learning model or algorithm is used to generate a large set of bounding boxes spanning the full image (that is, an object localization component)

TensorFlow Lote model is used to generate a large set of bounding boxes.

The camera module is used for the live feed of the surroundings to detect objects.

MobileNet-SSD

The architecture of SSD is a single network of convolution that trains to predict bounding box positions and identify those positions in one step. Hence, end-to - end SSD can be trained. The SSD network is composed of base architecture (in this case MobileNet) followed by several layers of convolution:

SSD operates feature maps to detect where bounding boxes are located. A feature map is Df x Df x M in size. K bounding boxes are projected for each characteristic position of the map. Each bounding box bears the following information with it:

- 4 offset locations of the bounding box corner points
- Class probabilities

The box shape is not predicted by SSD, rather just where the box is. The bounding k boxes each have a default form. Prior to actual training the shapes are set. For instance, if there are 5 boxes, it means k=5.

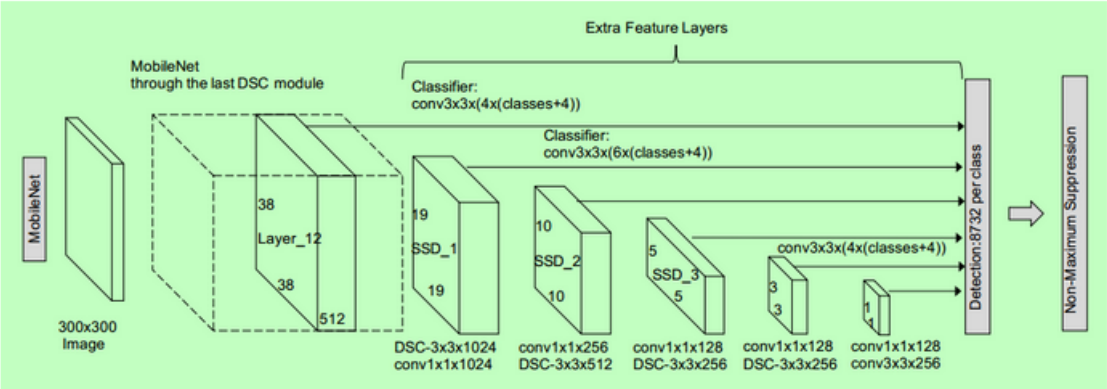


Figure 22: Mobile Net-SSD model

Loss in MobileNet-SSD

We can calculate the loss so with the final set of matched boxes:

$$L = 1/N \, (L \text{ class} + L \text{ box})$$

Here N is the cumulative number of boxes that fit. L class is the classified softmax loss and 'L box' is the smooth loss of L1 which reflects the matched box error. L1 smooth loss is a more robust modification of L1 loss to outliers. If N is 0, then the loss is also set to 0.

MobileNet Model

The MobileNet model is based on the depth-separable convolutions that are a factorized version of convolutions. These factor a standard convolution into a convolution of depth and a convolution of 1 / 1 called point-sensitive convolution.

For MobileNets, a single filter is applied to each input channel by the Depthwise Convolution. Then the pointwise convolution applies a 1 x 1 convolution to combine the profound convolution outputs.

A regular convolution filters both and in one step transforms inputs into a new set of outputs. This is divided into two layers by the depth-separable convolution-a single filter layer and a single layer for blending. **This factorisation results in a drastic reduction in computation and model size.**

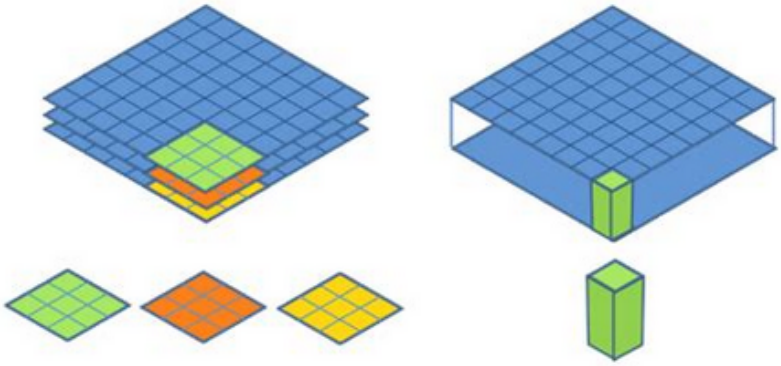


Figure 23: Depthwise and Pointwise convolutional filters

- **Optical Character Recognition Module**

Text Structure

The text is segmented into words, lines and blocks by the Text Recognizer.

- A block is an adjoining set of lines of text, such as a paragraph or column,
- On the same vertical axis a line is a contiguous set of words and

- A word, on the same vertical axis, is a contiguous set of alphanumeric characters.

The image below highlights each of these examples in descending order. A Line of text is the first illuminated row, in cyan. The second set of highlighted blocks are text lines, in blue. Finally, the third group of illuminated blocks are Words, in dark blue. [17]

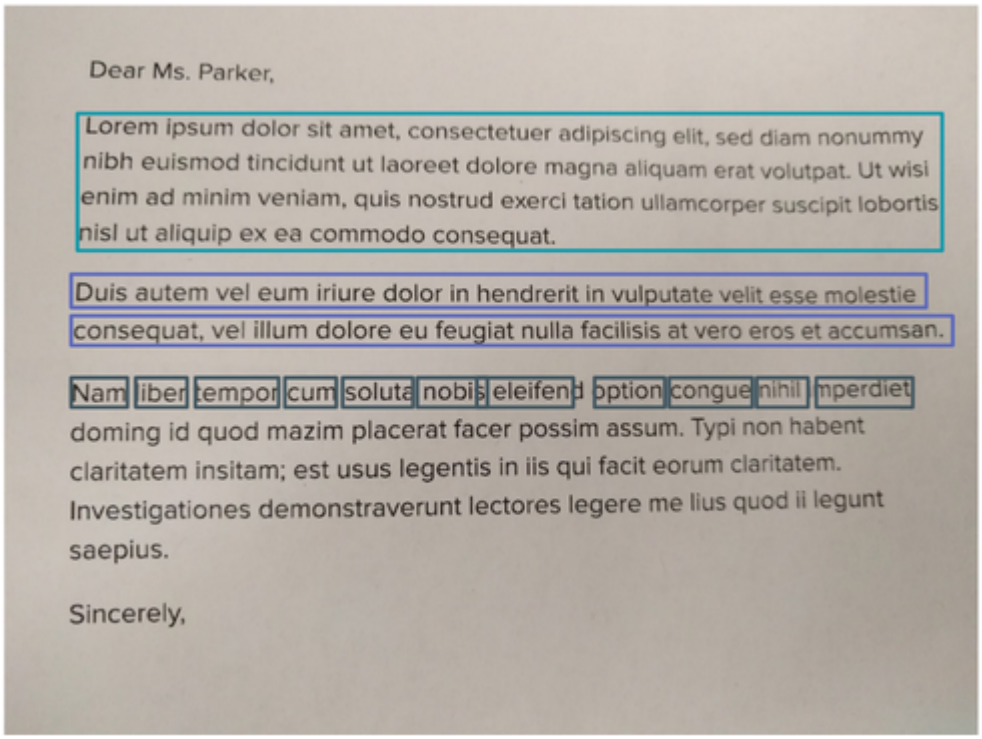


Figure 24: Block, lines and words in text

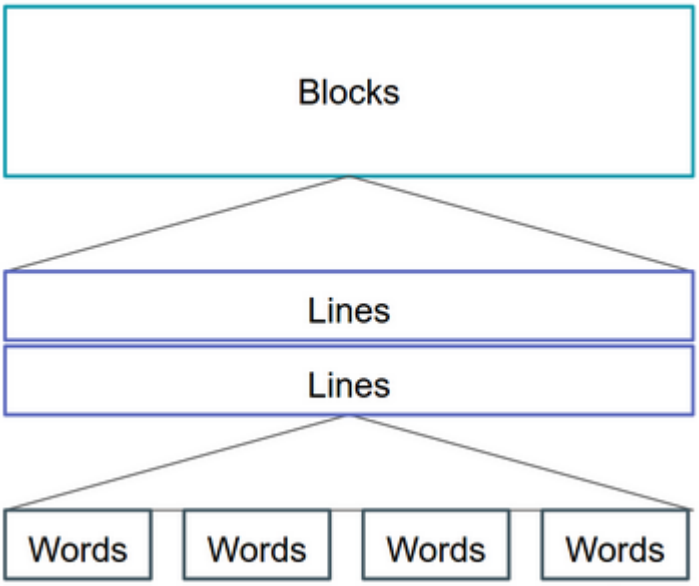


Figure 25: Block, lines and words

- **Speech Module**

The architecture of CMU Sphinx

Sphinx2 includes a number of libraries that provide both key speech recognition and auxiliary functions such as low-level audio recording. Those libraries are in C language. Its key characteristics include:

- Continuous encoding of expression (against independent comprehension of words)
- Speaker-independent (doesn't need the machine to be trained)
- Ability to deliver single best or multiple alternate recognitions
- Half-step acoustic versions
- Models of bilingual or tri-gram languages

Included in Sphinx2 were several features specifically intended for the development of real applications. Many elements of the decoder can be reconfigured at runtime, for example. New language models can be dynamically loaded or switched. Similarly , it is possible to add new words and pronunciations. For any future analysis the audio input data can be automatically logged into files.

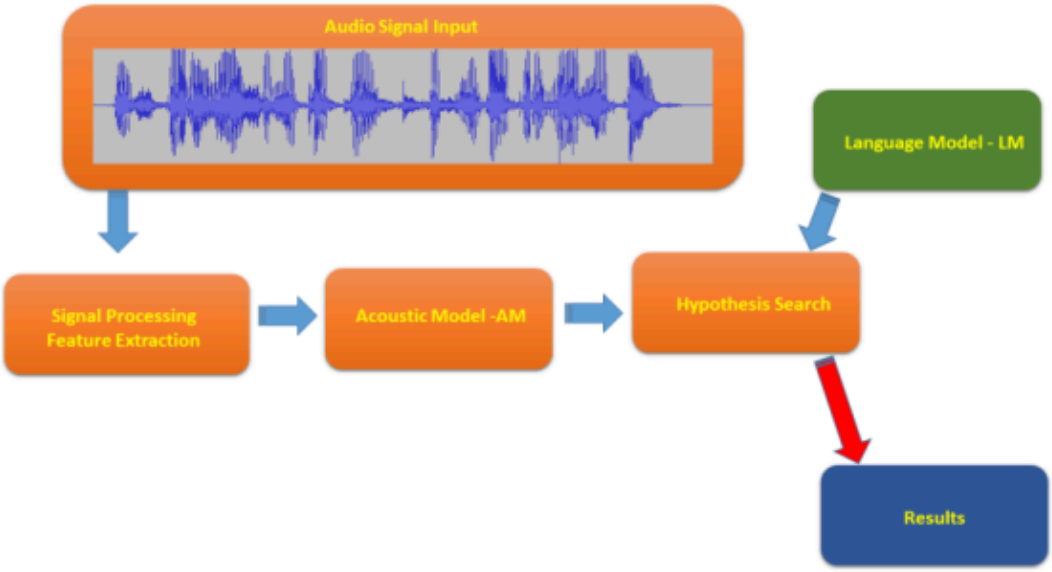


Figure 26: Architecture of Speech Recognition system

Hotword Detection Flowchart

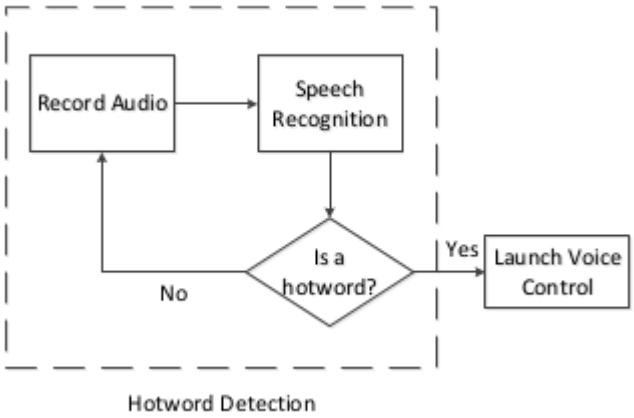


Figure 27: Hotword detection flowchart

Chapter 6

IMPLEMENTATION AND TESTING

6.1 Implementation And Testing

6.1.1 Tools and Technologies

The tools and technologies used for the development of the system are as follows:

- Android Studio
- Google Colab
- Tensorflow lite
- Sphinx

Since the system has been developed by focusing on the designing and user-friendly features in the application, the tools specific to Android application development were Android Studio, for voice commands in the application, Android Studio provides an easy way to insert audio clips in the application. Also, after the final design was confirmed, the implementation of the design in the final working application was easily carried out in Android Studio.

Android Studio:

Android Studio is a platform unique to Android application development that offers various SDK tools and libraries for the creation and launch of android applications. The studio provides an activity design window where the user can either

design the screen layout to be displayed on the mobile phone by drag-and-drop or write the XML code.The studio also offers the ability to write the Java class files that may be associated with the designed activities. There is also the option to use an emulator to test the application, though the application has been tested on our own mobile phones in our case.

Colaboratory:

Colaboratory or "Colab" for short, helps you to write and execute Python in your browser, with Zero setup needed, unlimited access to GPUs Quick collaboration If you are a student, a data scientist, or an AI expert, Colab will make your job simpler.

Colab notebooks allow you to combine executable code and rich text with images, HTML, LaTeX and more in a single document. They'll be stored in your Google Drive account when you create your own Colab notebooks. You can quickly share your Colab notebooks with coworkers or friends, making them report on or even change your notebooks.

TensorFlow Lite:

TensorFlow Lite is an open-source, in-device inference deep learning platform. TensorFlow Lite is a suite of software that supports developers to operate TensorFlow models on mobile devices, embedded devices, and IoTs. It allows machine-learning inference with low latency and small binary size on-device. TensorFlow Lite is made up of two primary components:

The parser TensorFlow Lite, which operates specifically designed versions on several various forms of equipment, including smartphones, embedded Linux computers, and microcontrollers.

The TensorFlow Lite adapter, which transforms TensorFlow models into an optimized type for interpreter usage, which can implement enhancements for binary size which efficiency improvements.

CMUSphinx:

The CMUSphinx toolkit is a leading toolkit for speech recognition which provides numerous tools to create voice applications. CMUSphinx includes a range of modules

which are designed for different activities and applications. What to pick is frustrating occasionally. To shed some light on the pieces from the toolkit, here's a list of:

- Pocketsphinx — a lightweight library of recognizers written in C.
- Sphinxbase — supporting library Pocketsphinx includes
- Sphinx4 — Java-written dynamic, modifiable recogniser
- Sphinxtrain — Training tools for acoustic models

6.1.2 Implementation

6.1.2.1 Android Application

The android app has been fully developed in Java and XML. Android Studio has been the platform used for this purpose. The development of the interface has been done in XML, while the remainder of the back-end functionality was coded using Java. The development process began by first designing the central android 62 program framework. This included designing the android activities, and developing them.

The screens for various functionalities of this component were developed using the prototypes developed during the Requirements Analysis and Design phase. Selected text fields and voice commands and selection buttons bearing in mind the user's ease of use and interaction with the application. The theme and color scheme for the application design was carefully selected so that the user will not get annoyed when using the application. On Adobe Photoshop the application's logo was developed and designed.

6.1.2.2 Object Detector

This is a camera software that uses a quantized MobileNet SSD model trained on the COCO dataset to continuously identify the artifacts (bounding boxes and classes) in the frames used by your device's back camera.

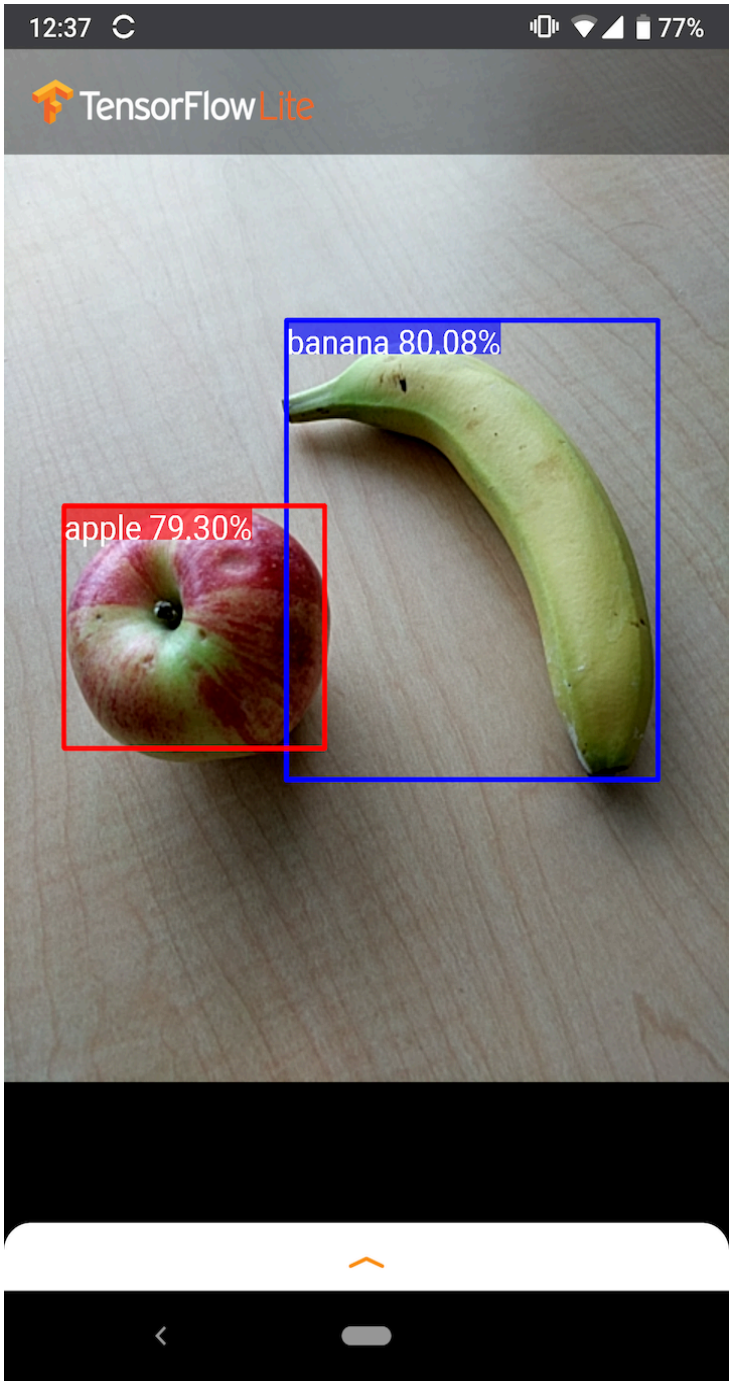


Figure 28 Object detector detecting apple and banana in real time.

Train/Val accuracy/loss graph:

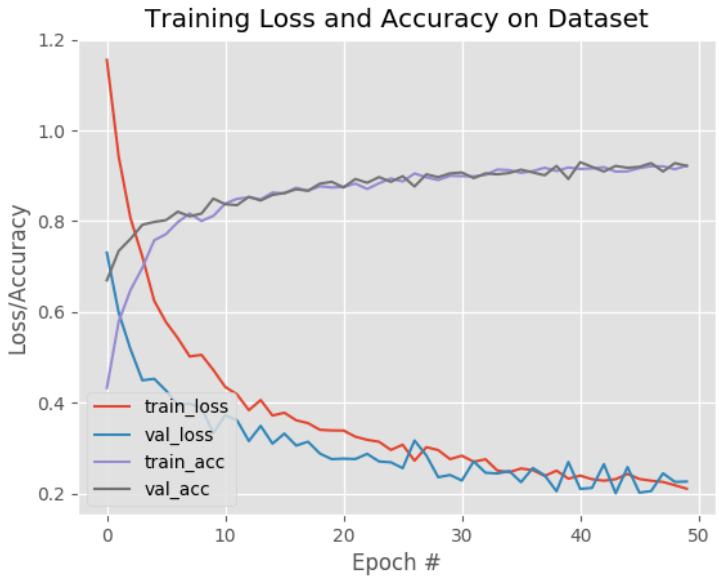


Figure 29 Loss and Accuracy graphs of training and validating dataset on 50 epochs

An object detection model is learned to detect the presence and the position of multiple object classes. For example, a model can be trained with images containing different pieces of fruit, along with a label showing the type of fruit they represent (e.g. an apple, a banana, or a strawberry), and data showing where each object appears on the image.

It will then emit a list of the items it identifies, the location of a bounding box for each object, and a score showing the trust that the detection was accurate.

Imagine a model has been trained on a COCO dataset. When we pass it an image, it will output a set number of detection results - in this example, 5.

Table 1: Table of output array that object detector returns

Class	Score	Location

Car	0.91	[18, 21, 57, 63]
Cup	0.83	[100, 30, 180, 150]
Monitor	0.81	[17, 82, 89, 163]
Cat	0.43	[42,66, 27,1 83]
Vase	0.36	[6, 42, 31, 58]

Confidence score:

To analyze those findings we should look at the score and position of every detected item. A score is a number between 0 and 1 that shows the assurance that the object was truly detected. The nearest number is to 1, the more likely the trend is.

You should agree on a cut-off point, depending on your query, at which the detection results will be discarded. In our case, we could agree that a reasonable cut-off is a score of 0.5 (meaning the identification would be valid at a 50 percent chance). The last two items in the sequence will be overlooked in this case, as those confidence scores are below 0.5.

Getting labels from the resultant object array:

```
for (final Classifier.Recognition result : results) {
  final RectF location = result.getLocation();
  if (location != null && result.getConfidence() >= minimumConfidence) {
    canvas.drawRect(location, paint);

    cropToFrameTransform.mapRect(location);

    result.setLocation(location);
    mappedRecognitions.add(result);
//    System.out.println(Arrays.asList(result).indexOf(2));
//    System.out.println(result);

    String[] labelAndBoundingBoxes = String.valueOf(result).split( regex: " ");

    labels.add(labelAndBoundingBoxes[1]);



  }
}
```

Figure 30 Code snippet for getting labels from the resultant object array.

Object Detector integration with text to speech library:

```
textToSpeech = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
  @Override
  public void onInit(int status) {
    if (status == TextToSpeech.SUCCESS) {
      int ttsLang = textToSpeech.setLanguage(Locale.US);
      for (String labels_speak : labels_array_to_speak){
        String data = "There is a "+labels_speak+"in front of you";
        tts(data);
        if (ttsLang == TextToSpeech.LANG_MISSING_DATA
                || ttsLang == TextToSpeech.LANG_NOT_SUPPORTED) {
          Log.e( tag: "TTS",  msg: "The Language is not supported!");
        } else {
          Log.i( tag: "TTS",  msg: "Language Supported.");
        }
        Log.i( tag: "TTS",  msg: "Initialization success.");
      }
    } else {
      Toast.makeText(getApplicationContext(),  text: "TTS Initialization failed!", Toast.LENGTH_SHORT).show();
    }
  }
});
```

Figure 31 Code snippet of object detector integration with text to speech library

6.1.2.3 Optical Character recognition

Optical Character Recognition (OCR) gives a computer the ability to read text that appears on a picture, enabling applications to make sense of signs, articles, flyers, text pages, menus, or any other place where text appears as part of an image.[12] The

Mobile Vision Text API features a strong and stable OCR functionality for Android developers that fits for most Ios devices and doesn't increase in size

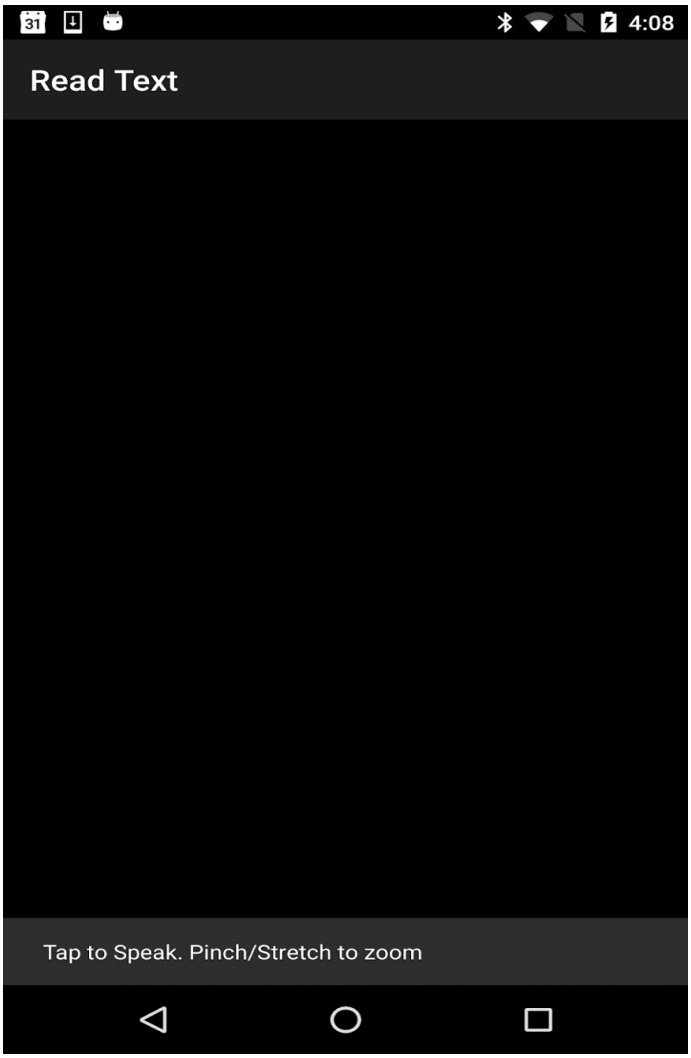
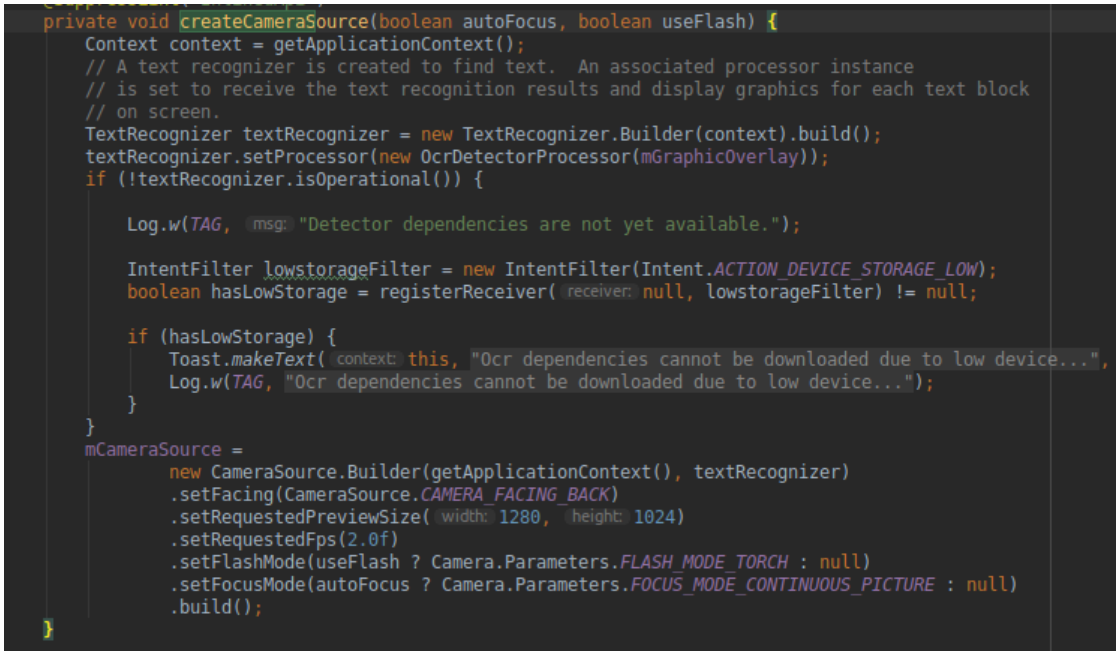


Figure 32 UI of OCR

TextRecognizer. This item detector analyses the pictures and decides which text exists in them. After it has been configured a TextRecognizer can be used to identify text in all image forms



```
private void createCameraSource(boolean autoFocus, boolean useFlash) {
    Context context = getApplicationContext();
    // A text recognizer is created to find text.  An associated processor instance
    // is set to receive the text recognition results and display graphics for each text block
    // on screen.
    TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();
    textRecognizer.setProcessor(new OcrDetectorProcessor(mGraphicOverlay));
    if (!textRecognizer.isOperational()) {

        Log.w(TAG,  msg: "Detector dependencies are not yet available.");

        IntentFilter lowstorageFilter = new IntentFilter(Intent.ACTION_DEVICE_STORAGE_LOW);
        boolean hasLowStorage = registerReceiver( receiver: null, lowstorageFilter) != null;

        if (hasLowStorage) {
            Toast.makeText( context: this, "Ocr dependencies cannot be downloaded due to low device...",
            Log.w(TAG, "Ocr dependencies cannot be downloaded due to low device...");
        }
    }
    mCameraSource =
            new CameraSource.Builder(getApplicationContext(), textRecognizer)
            .setFacing(CameraSource.CAMERA_FACING_BACK)
            .setRequestedPreviewSize( width: 1280,  height: 1024)
            .setRequestedFps(2.0f)
            .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH : null)
            .setFocusMode(autoFocus ? Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE : null)
            .build();
}
```

Figure 33 CameraSource function

The TextRecognizer is operational, so we could use it to individually detect frames. But we want to do something a little more interesting: seeing the camera, read text live. To do this, we must build a CameraSource which is a pre-configured camera manager for Vision processing. We're going to set the high resolution and turn autofocus on because this is a good match for recognizing small text.

It's helpful to add a Processor to interpret the text straight from the frame, which can manage detections as soon as they are visible.

```
public class OcrDetectorProcessor implements Detector.Processor<TextBlock> {

    private GraphicOverlay<OcrGraphic> mGraphicOverlay;

    OcrDetectorProcessor(GraphicOverlay<OcrGraphic> ocrGraphicOverlay) {
        mGraphicOverlay = ocrGraphicOverlay;
    }

    /**
     * Called by the detector to deliver detection results.
     * If your application called for it, this could be a place to check for
     * equivalent detections by tracking TextBlocks that are similar in location and content from
     * previous frames, or reduce noise by eliminating TextBlocks that have not persisted through
     * multiple detections.
     */
    @Override
    public void receiveDetections(Detector.Detections<TextBlock> detections) {
        mGraphicOverlay.clear();
        SparseArray<TextBlock> items = detections.getDetectedItems();
        for (int i = 0; i < items.size(); ++i) {
            TextBlock item = items.valueAt(i);
            OcrGraphic graphic = new OcrGraphic(mGraphicOverlay, item);
            mGraphicOverlay.add(graphic);
        }
    }

    /**
     * Frees the resources associated with this detection processor.
     */
    @Override
    public void release() { mGraphicOverlay.clear(); }
}
```

Figure 34 OcrDetectorProcessor function

Screenshot:

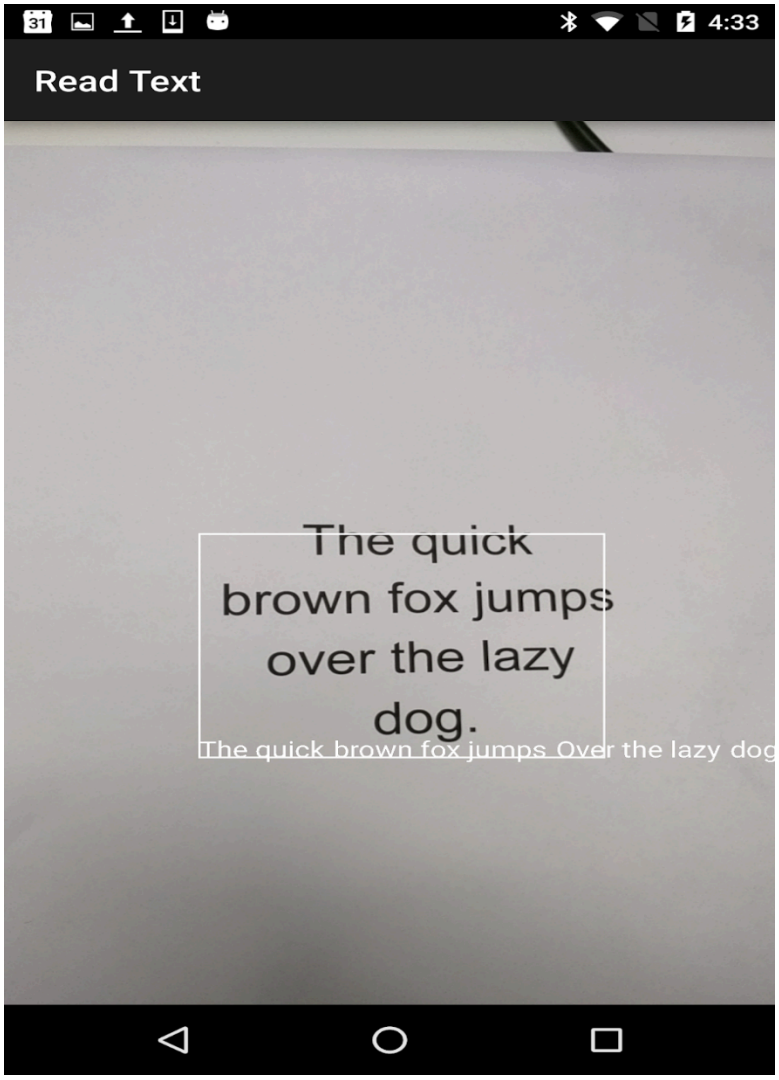


Figure 35 OCR recognizing text in real time

Text to speech conversion:

Android helps you to translate your text into your speech. It not only helps you to translate it, it also allows you to speak text in multiple languages.

To this end, Android provides class TextToSpeech. To use this class, you must place an object of this class and define the initListener [11]

```
textToSpeech = new TextToSpeech(getApplicationContext(), (status) → {
        if (status == TextToSpeech.SUCCESS) {
            int ttsLang = textToSpeech.setLanguage(Locale.US);
                tts(text);
                if (ttsLang == TextToSpeech.LANG_MISSING_DATA
                        || ttsLang == TextToSpeech.LANG_NOT_SUPPORTED) {
                    Log.e( tag: "TTS",  msg: "The Language is not supported!");
                } else {
                    Log.i( tag: "TTS",  msg: "Language Supported.");
                }
                Log.i( tag: "TTS",  msg: "Initialization success.");
        } else {
            Toast.makeText(getApplicationContext(),  text: "TTS Initialization failed!",
        }
});
```

Figure 36 Code snippet of text to speech library

Speech recognition:

The pocket sphinx-android classes and methods were built to imitate the same workflow used in pocket sphinx, except that simple data structures are transformed into classes and functions that interact with those structures are transformed into methods of the respective classes.

6.1.3 Testing

6.1.3.1 UI testing

Performance testing of the user interface (UI) ensures that your app not only meets its functional requirements but also that user interactions with your app are smooth butter. One solution to UI monitoring is essential to make a human tester run a series of user operations on the device and to check that they are operating properly. So we did the same thing and checked the app's user interface ourselves by offering different styles of input from the microphone.

6.1.3.2 Speech testing

Speech recognition as tested using microphone input. Different input was given to application in different accents. Most of the inputs get recognized by the application. The accuracy of speech recognition module is around 85%.

6.1.3.3 Model evaluation:

Cross-validation is a technique that includes partitioning the original observation dataset used to train the model into a training set, and an independent set used to evaluate the analysis. K-fold cross-validation is the most common cross-validation process, where the original dataset is divided into sub-samples k equivalent in size, called folds.The k is a user-specified number which typically has 5 or 10 as its preferred value. This is repeated k times, such that each time one of the k subsets is used as the test set / evaluation set and the other k-1 subsets are brought together to create a training sample. The error estimate is averaged over all k trials to get the total effectiveness of our model.

Intersection over union

- Intersection over Union is an assessment metric used for calculating the accuracy of an event detector on a given dataset. In order to use Intersection over Union to determine a (arbitrary) object detector, we must:
- Bounding boxes for ground-truth (i.e. the test set's hand-labeled bounding boxes indicating where the representation of our entity is in).
- Our model predicts bounding boxes

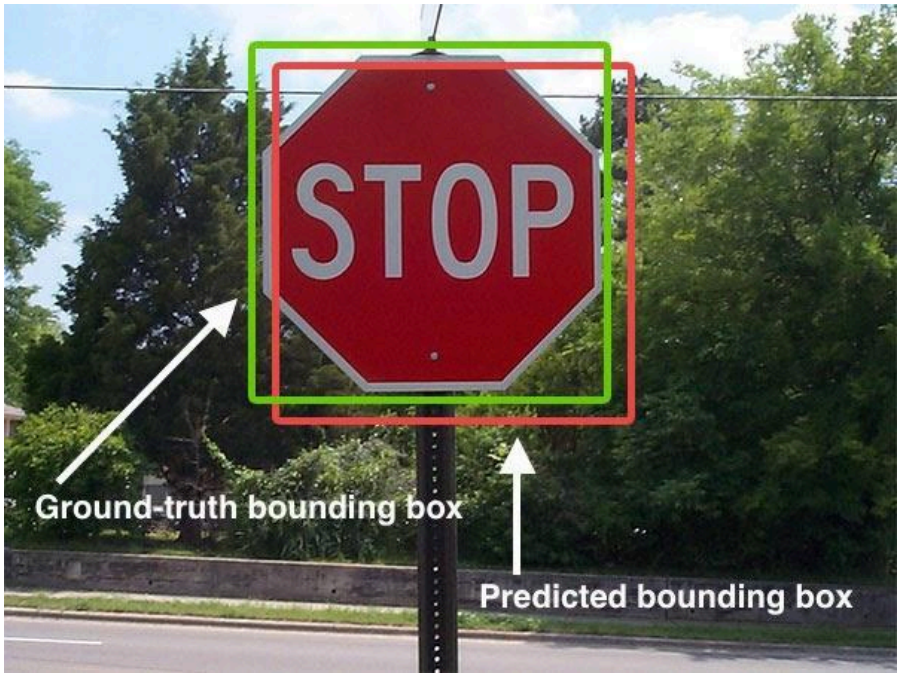


Figure 37 Figure shows predicted bounding box and ground truth bounding box

We can see in the figure above that our object detector has detected the presence of a stop sign on an image.

The predicted bounding box is drawn in red while the bounding box for the ground-truth (i.e., hand-labeled) is drawn in green.

Thus, the intersection of computation over the Union can be calculated via: [16]

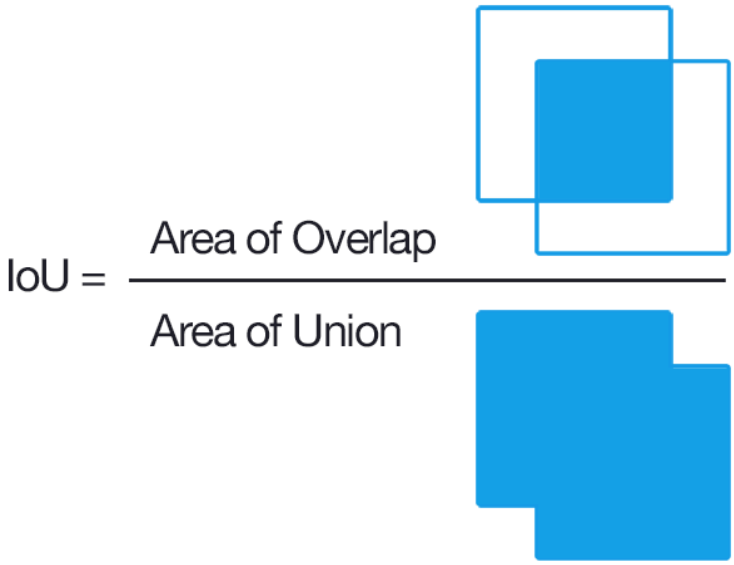


Figure 38 Intersection over union equation

We measure the region of variance between the predicted bounding box and the bounding box of ground-truth in the numerator.

The denominator is the union field, or more precisely, the field that contains both the projected bounding box and the bounding box of ground-truth.

Dividing the area of union overlap yields our final score — the intersection over the Union

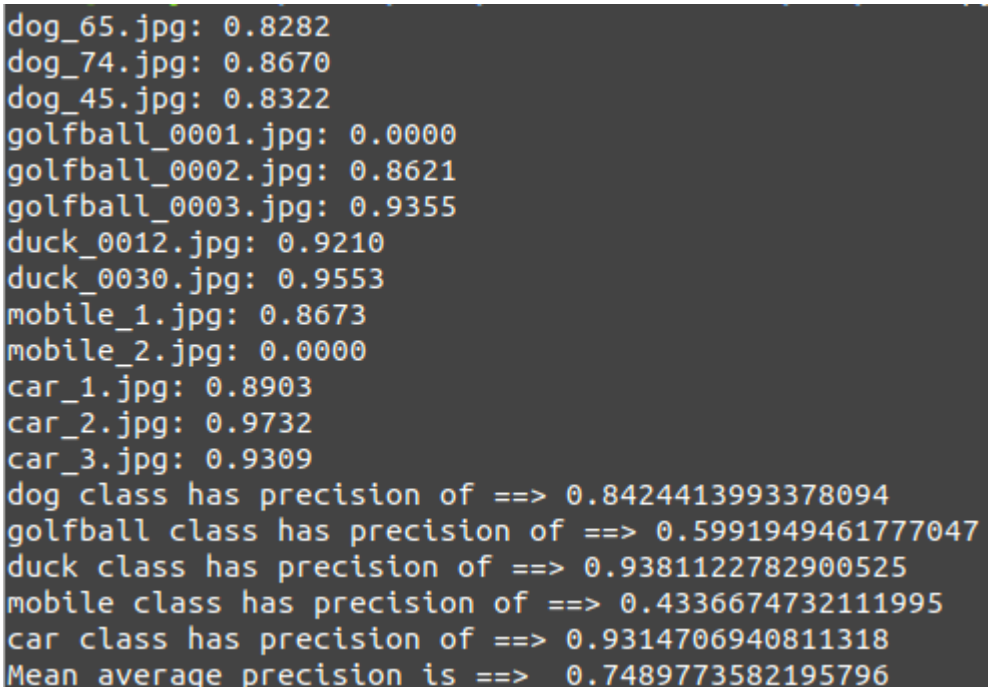


Figure 39 Mean average precision of multiple classes

Cosine Similarity/ Levenshtein distance:

The distance from Levenshtein (LD) is a measure of the similarity between two strings, namely the source string(s) and the target string(t). The gap is the amount of deletions, insertions or substitutions required to turn s into t.

Cosine similarity is a measure of the similarity of two non-zero vectors within an internal product space that measures the cosine angle between them

These two similarity comparing algorithms were used for finding the similarity in Optical character recognition and speech recognition. These two similarities algorithm play an important role in finding the accuracy of our speech and OCR modules.

Chapter 7

RESULTS AND DISCUSSION

A comprehensive evaluation of the solution is presented with supporting figures and graphics.

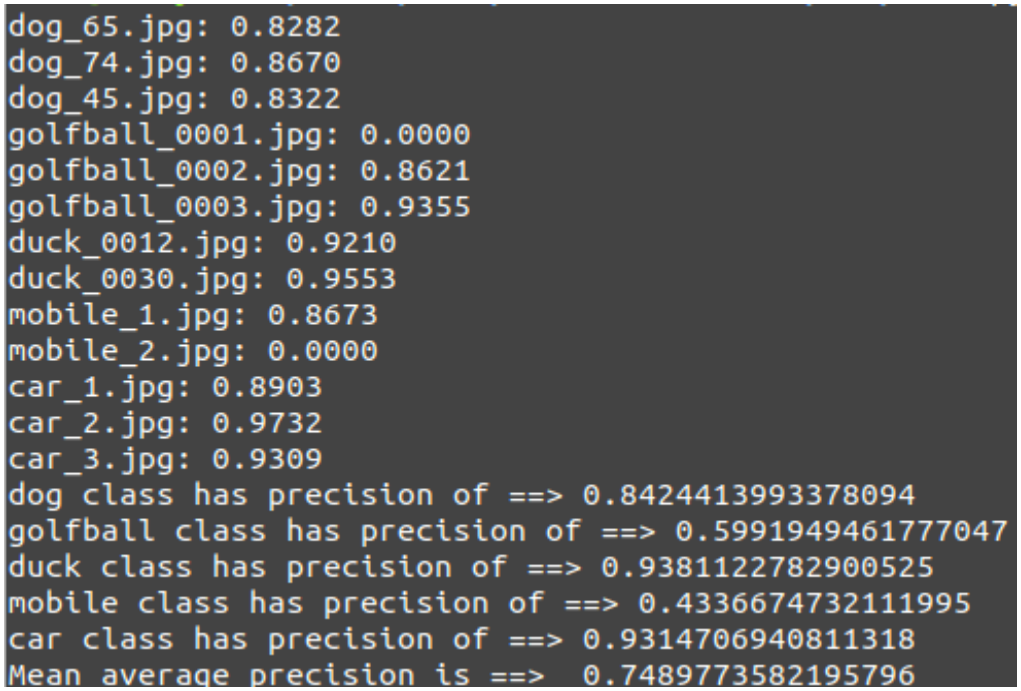
System testing is performed through a strong testing strategy and the test cases cover all the use cases.

7.1 Object Detection Module Testing

Mean Average precision:

We have calculated mean average precision of the object detection model on different 5 classes using intersection over union concept.

The results are:



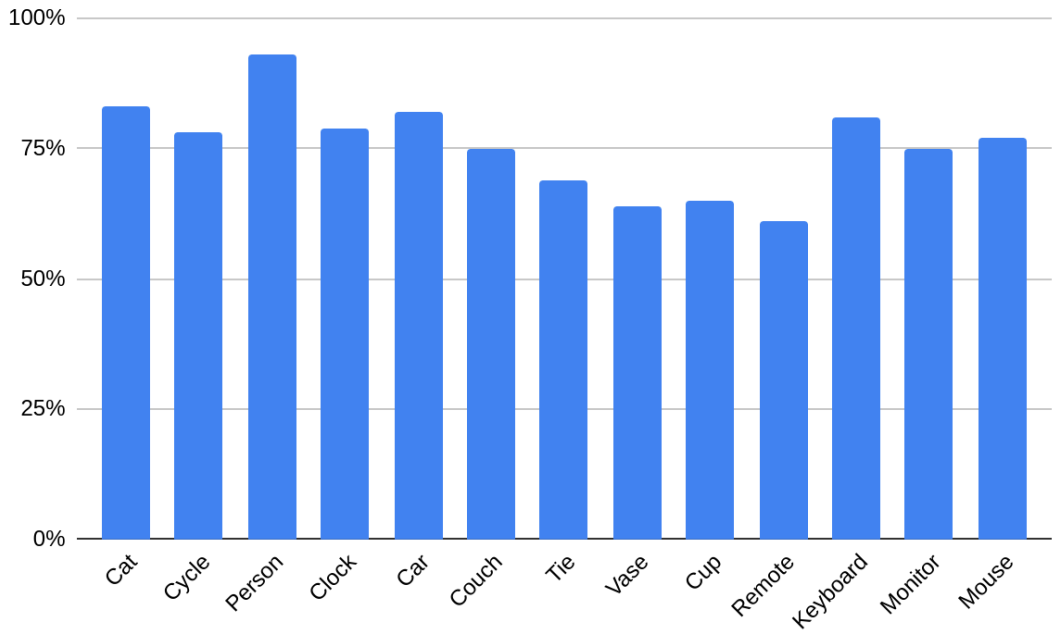
mAP = 74.89%

Accuracy of different classes:

Class	**Accuracy**
Cat	83%
Cycle	78%
Person	93%
Clock	79%
Car	82%
Couch	75%
Tie	69%
Vase	64%
Cup	65%
Remote	61%
Keyboard	81%
Monitor	75%

Mouse	77%

Table 2: Accuracy of different classes of object detection



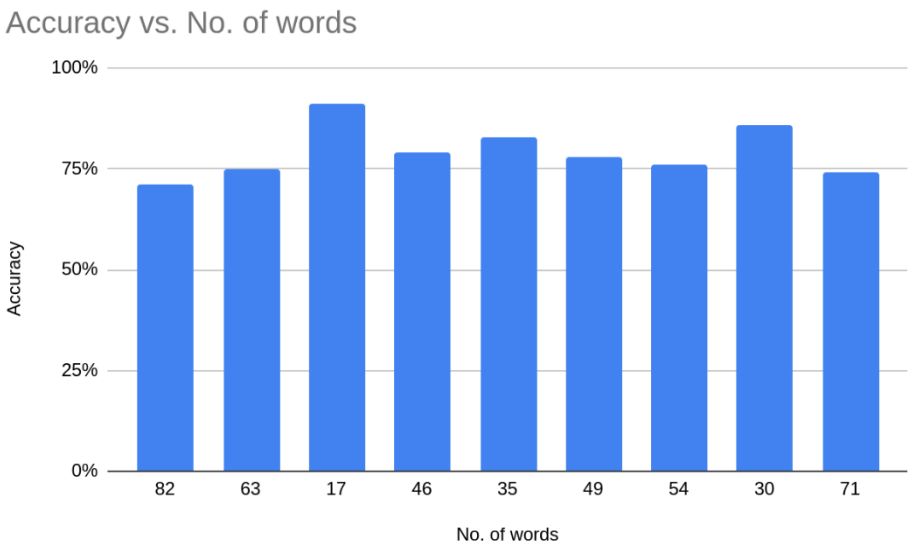
7.2 Optical Character Recognition Module Testing

When testing the optical character recognition module, we did testing on paragraphs with different number of words. Some paragrahs were less dense while some were more dense. The general pattern that we observed in our testing was that OCR module has higher accuracy on paragraphs that are less dense while paragraphs with less number of words, have lesser accuracy.

Number of words	**Accuracy**
82	71%
63	75%
17	91%
46	79%
35	83%

49	78%
54	76%
30	86%
71	74%

Table 3: OCR Module Accuracy



7.3 Speech Recognition Module Testing

When testing the speech recognition module, we took 100 one second audio clips from different audio dictionaries like Project Shkoota, Wikimedia Commons and Forvo of the key words used in our application.

We only tested the accuracy of our speech recognition module on the key words used in our application since calculating the accuracy of the module on other words would have proven to be redundant. The key words we tested are, “wakeup”, “Eyenovative”, “open”, “object”, “detection”, “text”, “reading”, “navigation”, “book”.

The audio data we used consisted of different pronunciations of the key words in different accents. However, Eyenovative does not exist in any dictionary, so we

took a sample of 100 different audio files collected from our friends, pronouncing the word in different ways.

Word	Samples	Accuracy
Wakeup	100	89%
Eyenovative	100	88%
Open	100	85%
Text	100	90%
Book	100	89%
Reading	100	90%
Object	100	87%
Detection	100	90%
Navigation	100	91%

Table 4: Speech recognition module accuracy

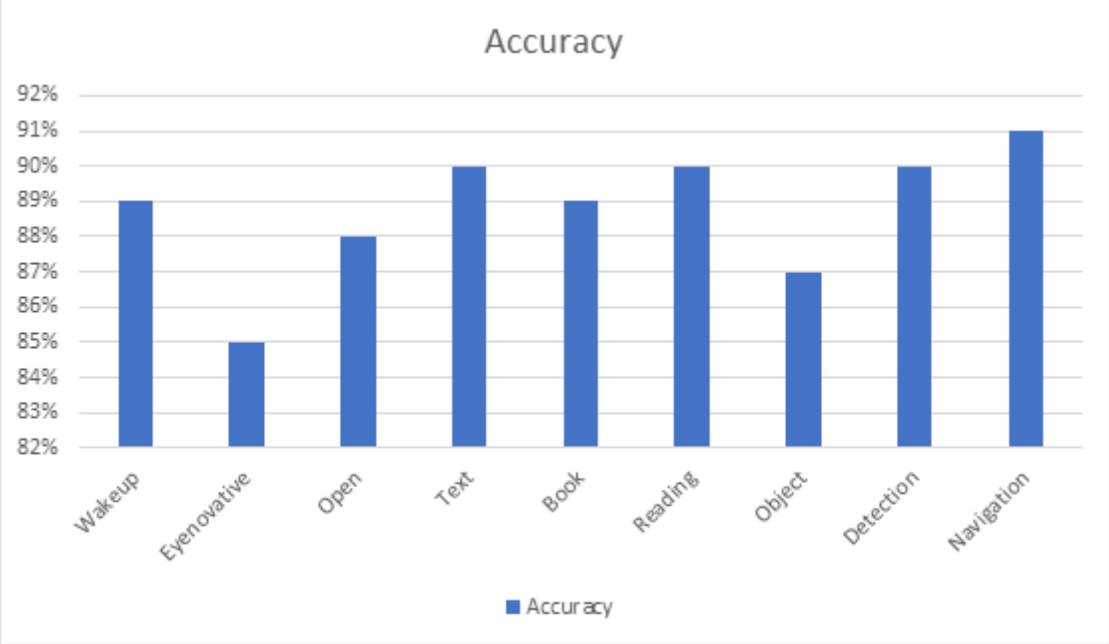


Figure 40: Accuracy graph of speech recognition module

As discussed in Chapter 4, we improved the accuracy of the speech module by removing any unnecessary words from our dictionary and adding different phonetic pronunciations of the same word in our language model.

Chapter 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion and Future work

8.1.1 Summary of action of the proposed solution

There is a shortage of assistive technology for blind or visually impaired people, and the technology already available is way too expensive for a common man to afford. Certain assistive technology is in dire need.

Our project is to create smart applications for those with visual impairments. In order to obtain real-time input, a phone camera is integrated into the application. This data is transferred to the Android application. It contains the following three modules:

- **Navigation Module:** This module helps the visually impaired to navigate through the obstacles. The user gets the directions in audio to make navigating around more seamless and effortless. Our product provides real-time object recognition.
- **Optical Character Recognition Module (OCR):** This module reads the text and converts it into audio and the user can hear it through the attached hands-free (text to speech).
- **Voice Commands Module:** The application operates through voice commands. The user can open the application and switch between modules using voice commands, thus providing ease of use. Our product provides affordability and a one-stop solution to people who are visually impaired.

8.1.2 Recommendations/ Future work

In the phase of usability testing, we received a really good response to our design during our design process. Using the program was a genuinely simple and enjoyable experience for them. We have a lot of ideas and guidelines, which can be included as potential research that is:

- Adding glasses to go with the application and mount camera on glasses to make it a wearable product.
- Making the app iOS compatible
- Adding OCR feature for handwriting recognition
- Making voice recognition for different languages
- OCR for different languages
- Increasing the number of objects for object detection
- Improving the accuracy of object detection and OCR

Chapter 9

REFERENCES

[1] Aipoly.com, V7 AiPoly. [Online]. Available: https://www.aipoly.com/. [Accessed: 10- Jun- 2020].

[2] Bai, J., Lian, S., Liu, Z. and Liu, D. (2017). Smart Guiding Glasses for Visually Impaired People in Indoor Environment. *IEEE Transactions on Consumer Electronics,* vol. 63, (3), pp. 258-266.

[3] Bemyeyes.com, Be My Eyes - See the world together. [Online]. Available: https://www.bemyeyes.com/. [Accessed: 10- Jun- 2020].

[4] BeSpecular, BeSpecular. [Online]. Available: https://www.bespecular.com/. [Accessed: 10- Jun- 2020].

[5] Codelabs.developers.google.com, See And Understand Text Using OCR With Mobile Vision Text API For Android. [online] Available at: <https://codelabs.developers.google.com/codelabs/mobile-vision-ocr/> [Accessed 15 June 2020].

[6] En.wikipedia.org, Amazon Kindle. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Kindle. [Accessed: 10- Jun- 2020].

[7] En.wikipedia.org. Android Studio. [online] Available at: <https://en.wikipedia.org/wiki/Android_Studio> [Accessed 15 June 2020].

[8] eSight, eSight - Electronic eyewear for the visually impaired. [Online]. Available: https://esighteyewear.com/. [Accessed: 10- Jun- 2020].

[9] GitHub, goxr3plus/Java-Google-Text-To-Speech, [Online]. Available: https://github.com/goxr3plus/Java-Google-Text-To-Speech. [Accessed: 15- Jun- 2020].

[10] Google Cloud, Detect Text in Images | Cloud Vision API | Google Cloud, [Online]. Available: https://cloud.google.com/vision/docs/ocr. [Accessed: 15- Jun- 2020].

[11] Google Cloud, Text-to-Speech Client Libraries | Cloud Text-to-Speech Documentation, [Online]. Available: https://cloud.google.com/text-to-speech/docs/reference/libraries. [Accessed: 15- Jun- 2020].

[12] Google Developers, Text Recognition API Overview | Mobile Vision | [Online]. Available: https://developers.google.com/vision/android/text-overview. [Accessed: 15- Jun- 2020].

[13] Hassan, A. and Tang, T. (2016). Smart Glasses for the Visually Impaired People. *15th biennial International Conference on Computers Helping People with Special Needs (ICCHP),* pp. 579-582.

[14] Identifi, Identifi - Connect What Matters. [Online]. Available: https://identifi.net/. [Accessed: 10- Jun- 2020].

[15] Macular Society, Reading App 2020. [Online]. Available: https://www.macularsociety.org/reading-app. [Accessed: 10- Jun- 2020].

[16] Medium, Calculating String Similarity In Python. [online] Available at: <https://towardsdatascience.com/calculating-string-similarity-in-python-276e18a7d33a> [Accessed 15 June 2020].

[17] Medium, Text Recognition for Android using Google Mobile Vision, [Online]. Available: https://medium.com/@prakash_pun/text-recognition-for-android-using-google-mobile-vision-a8ffabe3f5d6. [Accessed: 15- Jun- 2020].

[18] Miah, R. and Hussain, S. (2018). A Unique Smart Eye Glass for Visually Impaired People. *2018 International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE),* pp. 1-4.

[19] Microsoft.com, Seeing AI App from Microsoft [Online]. Available: https://www.microsoft.com/en-us/ai/seeing-ai. [Accessed: 10- Jun- 2020].

[20] Mohanapriya, R., Nirmala, U. and Priscillai, C. (2016). Smart vision for the blind people. *International Journal of Advanced Research in Electronics and Communication Engineering*, vol 5, (7); pp. 2014-2017.

[21] OrCam, OrCam MyEye 2.0. [Online]. Available: https://www.orcam.com/en/myeye2/. [Accessed: 10- Jun- 2020].

[22] RNIB - See differently, A complete guide to DAISY. [Online]. Available: https://www.rnib.org.uk/a-complete-guide-to-daisy. [Accessed: 10- Jun- 2020].

[23] Shmyrev, N., Pocketsphinx On Android. [online] CMUSphinx Open Source Speech Recognition. Available at: <https://cmusphinx.github.io/wiki/tutorialandroid/> [Accessed 15 June 2020].

[24] TensorFlow, Object detection | TensorFlow Lite, [Online]. Available: https://www.tensorflow.org/lite/models/object_detection/overview#uses_and_limitations. [Accessed: 15- Jun- 2020].

[25] TensorFlow, Object Detection | Tensorflow Lite. [online] Available at: <https://www.tensorflow.org/lite/models/object_detection/overview> [Accessed 15 June 2020].

[26] Who.int [Online]. Available: https://www.who.int/blindness/GLOBALDATAFINALforweb.pdf?ua. [Accessed: 07- Jun- 2020].