School of Computer Science and Electronic Engineering

BSc Computer Science

Computer Vision and Speech Synthesis Based Scene Understanding for People with Impaired Vision

Capstone Project Final Report

Student: Kyriakos Kyriakou
Student Registration Number: 1905770

Supervisor: Professor John Q. Gan
Second Assessor: Dr. Junhua Li

ABSTRACT

Vision impairment is a major health problem affecting millions of people around the globe. Scene understanding is a challenging task for these people; hence, the project presents a mobile application ("BlindVision"), that uses Computer Vision and Speech Synthesis techniques to help them understand a scene ahead and assist them with their daily tasks.

Deep learning for Object Detection and Image Classification has been combined with Optical Character Recognition, to give an in-depth understanding. Different CNN object detection models have been tested and evaluated including YOLOv4, MobileNetV1 SSD, and EfficientDet Lite0/1/2/3, all trained with the COCO dataset, and Google's ML Kit was used for the image labelling and text recognition which uses the free Google Vision API. The app can recognize specific text, people, objects, and give audio explanation of the environment by using Text-To-Speech tools. Using Speech Recognition, the system can understand the user's commands by voice. Finally, the system has built-in gestures to make the interaction with the users hands-free.

Testing and evaluation on the different object detection models showed that EfficientDet Lite0 achieves the overall best performance. The accuracy of the model on the device is 67%. Along with the image classification model, which achieves accuracy of 85%, and text recognition, the performance of the app while running, is 14.4 frames per second.

New features could be introduced, that will make the interaction with the users better, new object detection models could be evaluated, and lastly, system could make use of the Google's Cloud Vision APIs for further understanding

ACKNOWLEDGEMENTS

I want to thank all the people who helped me throughout the year with my Capstone Project.

Firstly, I would like to thank my supervisor, Professor John Q. Gan, whose guidance and dedication shaped my project with ideas. Your consistent feedback and advice during the year helped me reach my goals and pushed me beyond my limits to achieve the highest level of work possible. Without your assistance in every step throughout the year, this project would not have been possible. Thank you very much for your support and understanding.

Secondly, I feel like thanking my second assessor, Dr. Junhua Li, for his advice and suggestions given at the interim oral interview.

I would also like to acknowledge my parents for their patience and understanding. Furthermore, I want to thank my sister for her insightful advice, feedback, and time spent helping me perfectionate my presentations skills.

Finally, I want to thank my friends for their support throughout the year, spotting bugs and testing my application. Special thanks to my friend Federico Sidoni, who was always there providing me with suggestions for my project, throughout our stimulating discussions.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

According to World Health Organization (WHO), blindness and vision impairment is a major health problem affecting 2.2 billion people globally [1]. Vision impairment mainly affects older people, but vision problems are seen in people of all ages [1]. Blindness is divided in four categories: mid, moderate, severe, and blindness [1]. People affected by vision problems experience personal and economic impacts. Treating these health problems is usually costly and not affordable to many people. Moreover, vision impairment on personal level can have severe consequences. The quality life of a human can be impacted in many ways. For instance, it is observed that people with vision difficulties have lower rates of involvement in the society [1]. Social and cognitive development problems can also rise. Lastly, individuals with impaired vision may find it difficult to cope with daily tasks such as understanding what object is ahead [1]. Scene understanding is one of the most challenging tasks people with impaired vision are called to face. Understanding a scene ahead can assist them to make their daily tasks easier. A simple knowledge of objects ahead can lead to an overall more productive and easier life. These daily activities can be such as reading, writing, shopping, walking, and etcetera [2]. Nowadays, the development of smartphones has come a long way, giving accessibility features to these people in order to help them use the devices without any trouble. Few years ago, this wouldn't be possible, but with the newer cutting-edge technology everything has been made accessible to everyone.

1.1. Motivation

The Computer Vision technology has been one of the most important, fascinating, and fast-growing fields today that is employed to deal with tasks such as object detection, optical character recognition (OCR), image classification, and more. This technology has "unlocked" endless possibilities to help people with impaired vision cope-with their tasks. Its implementation is promising and expected to create effective mobile applications that make use of the computer vision field to give assistance to blind people dealing with problems of recognizing text marks, identifying objects, and generally understanding their surroundings [2][3]. Because of the problems blind people and individuals with impaired vision face regarding scene understanding, it is important to aid them, as to maintain their independence and ease their daily living. These are the main reasons pushing me towards working on such project. Mobile applications nowadays have become extremely popular. Therefore, creating a mobile app that employs computer vision for blind people to get assist from in their daily life was deemed suitable.

1.2. Project Goals

The project presents a mobile application, "BlindVision", that makes use of computer vision and speech synthesis techniques to assist a person with impaired vision in understanding the scene ahead. Using computer vision methods, the system should recognize specific text and marks, people, and objects. Based on a scene, the system should give audio feedback with a brief description of what has been seen. These methods are expected to be highly efficient and effective as the application is meant to be used in real-time by the users. The Computer Vision methods in the app aim to process the frames taken from smartphone's camera. Pointing the device camera to something, the application should give a description of the captured frame. The frame description then, must be passed to the users with Speech Synthesis techniques. The mobile application developed is expected to have an easy to use and friendly graphic user interface (GUI) to make the interaction with the users as simple as possible. The colors chosen for the GUI must be easy on the eyes to provide the best possible experience. The buttons and text labels within the graphic user interface should be large and visible too. When users launch the application, a welcome audio will automatically be employed, which will welcome the users and will provide them with instructions on how to interact with the application. The last goal of the application's user experience was to make the whole user interaction hands-free. Implemented Speech Recognition methods should listen to the users' commands by voice to deem the

navigation with buttons unnecessary. The whole hands-free experience is accomplished with the implementation of a device shaking gesture. When scene recognition is working, a simple shake of the device should take the users to the home page.
Moving on to the Computer Vision aspect of the application, the system's first goal was to implement Object Detection. This computer vision technique has the capacity to identify and localize different objects, which is more than useful for people with impaired vision problems. For example, a person could want to walk from their bed to their living room. Object detection could notify the person that a couch, a tv, a chair, or something else is in the way. Furthermore, if users know the environment of their house, and they know that they have a tv in their living room, a detection of a tv should immediately inform them that they are in the living room, and they are pointing to one. Therefore, this could help the person localize in a familiar environment too. The second goal was to make the system understand text. Optical Character Recognition (OCR) is a technology that recognizes printed or handwritten text characters from images or frames captured. It is an important feature for the application's concept because people with impaired vision may not be able to see marks, labels, warning signs, etc. Imagine walking down the street, and a road is closed because constructions are being made. In these cases, street signs are placed in the road to notify people of what is happening. With the proposed application users should point the mobile device and comprehend what is written on street signs. The third goal of the computer vision field was to implement Image Classification. Image Classification is categorizing and labelling the pixels of the image, or video's frames to get a scene understanding. By combining these three techniques, a more accurate, and reliable application will be created for people with vision difficulties. After adapting these three techniques in the app, a new goal was set, that being the better interaction between the application and users. Users should be able to select a mode of detection. They can either select text detection feature, or scene understanding feature (object detection and image classification combination), or both. Developing the above computer vision tasks wouldn't be possible without using the artificial intelligence field. Supervised deep learning for object detection and image classification has been combined with optical character recognition, to give an in-depth understanding of a scene. Machine learning algorithms, make these tasks highly efficient in the real world. The last goal of the computer vision field was to use different convolutional neural networks (CNN) models for object detection. Different models were tested and evaluated including YoloV4 tiny, MobileNetV1 SSD, and EfficientDet Lite 0/1/2/3, all trained with the Microsoft COCO dataset. Testing and evaluation on these different object detection models showed that EfficientDet Lite 0 achieves the overall best performance. The accuracy of the model on the device is 67%.
Google's ML Kit, a mobile-based software development kit (SDK) that supports Android and IOS development [4], is used for the image labelling and text recognition. Both tasks make use of the free Google Vision API. The image classification model achieves accuracy of 87%.
The mobile application, as mentioned above, must give an audio explanation of the environment by using Text-To-Speech techniques. The first goal that dealt with the Speech Synthesis was a simple audio explanation of the detected objects. When objects are detected, their labels are forwarded to the text-to-speech module, and a natural language conversion to sound is created. The conversion of the natural language to sound is outputted to the users. The audio module used is Google's Text-To-Speech API. The second goal of Speech Synthesis was to convert the text detection from OCR to sound and forward it again to the users. The final goal for the audio feedback was to create a scene understanding descriptions. This feature interprets in more depth with the detections made, and more complex scene understanding audio templates are generated. The templates are passed to the speech synthesis module, and then to the users as a sound.

1.3. Report Structure

The rest of the report is structured as follows. Section 2 gives an overview of the history, tools, and concepts of the project. Consequently, Section 3 reviews and compares related work with the current project. Then, on Section 4 technical details about BlindVision's design and implementation are discussed. Section 5 analyzes experimental investigation (testing) and evaluation of different object

detection models and the system's overall performance. Thereafter, on Section 6, project management and planning records are stated. Finally, Section 7, encapsulates the summary of the project.

2. BACKGROUND

We live in a digitalized world where advanced technologies exist, and they are very promising when it comes to using them in the real world. Smartphone devices are constantly involving in terms of software and hardware (faster CPUs, outstanding mobile cameras, etc.) something that enables these technologies to be used in our everyday life routines. Advanced and complicated applications that employ artificial intelligence are also being developed.

2.1. Mobile Application Development

Mobile application development because of its popularity today, has become high priority in the software world. Consequently, ton of frameworks have been created that are meant to be used by developers to build their own apps. There are two main platforms of applications, Android and IOS. When it comes to developing an application for one of these platforms, a native app is created. On the other hand, when a developer chooses to develop an app that can be run both on IOS and Android, a cross-platform app is made. The selection of framework is particularly important because they can shape the project differently since every framework comes with their own limitations and positives [5]. There are three types of mobile app development frameworks [5]. There is a framework for Android development, a framework for IOS development, and a framework type for cross-platform mobile development. The selection should be based on the dependency of framework, the performance, the platform feature support, and the cost of development [5][6]. The dependency is based on the features it can offer. Usually, a cross-platform framework does not offer many features as the integration to those frameworks is a lot harder [5]. The platform support is determining if the application will run on android, IOS or both environments [5]. If a developer aims to deliver the application to more people, creating it on a cross-platform framework could save a lot of time. That's because today's cross-platform frameworks give the option to write the code once and deploy to both IOS and Android devices at the same time[5]. When selecting a framework, the cost of deployment (how much money will take to develop an app) must be considered [5][6]. Cross-platform and IOS development are pricier [5]. Some cross-platform frameworks are React Native which uses JavaScript programming language, .NET Xamarin that uses C# for the backend and XAML for frontend, and Flutter which uses the Dart programming language. On the other hand, a native framework can offer a broader functionality with access to ton of APIs and tools [7]. Most of the times, a native developed application has a higher performance compared to a cross-platform one [7]. Publishing a native app is always easier since it delivers better performance, speed, and less testing is needed [7]. Often, an android native app development is not costly, but in contrast, an IOS native development is pricy [7]. The two best native frameworks are XCode for IOS which uses Apple's programming language Swift, and Android Studio for Android development which offers plenty of programming languages such as Java, Kotlin, C++, and XML for the designing aspect of the application [5].

2.2. Computer Vision and Deep Learning

Computer Vision field is used in various applications nowadays. Some of its applications are self-driving cars, face recognition, and object tracking [5]. The project uses supervised machine learning for its computer vision techniques, meaning that the system is trained using labelled examples [5]. The labelled examples for this project are image datasets for the object and image classification [5]. This field of artificial intelligence makes use of deep learning to make the computer vision algorithms efficient in the real world [5]. To conduct all the labelling and classifications, convolutional neural networks (CNN) are being used [5][8]. Other algorithms worth mentioning include Region-CNN and Faster R-CNN [5]. A CNN consists of convolutional layers, pooling layers (subsampling), and dense

layers (fully connections) [5][9]. A convolutional layer is a simple block that takes two inputs, an image matrix, and a filter [5]. Filters are used to extract different features from the passed images or video frames [5]. Each neuron/unit is connected to a smaller number of nearby units in the next layer of the convolutional neural network [5]. For computer vision tasks such as object detection, image classification, and OCR, convolutional neural networks are preferred, as they scale everything down [5]. When it comes to image processing, a CNN checks for pixels that are nearby to each other because there are much more alike [5]. CNN also help by limiting the searching of weights to the size of the convolution, something that prevents complexity in the machine learning models [5]. By using subsampling layers, less parameters are needed, therefore less memory is used [5]. Furthermore, a lot of information is removed because of the feature extraction [5][9]. This is also called pooling. The most common pooling type is max pooling [5][10]. It creates a small kernel and only the max value makes it to the next layer [5][10]. Activation functions are used to determine each time if neurons should fire or not by introducing a non-linearity into the output of the neurons [5]. Some activation functions are: Sigmoid function, Rectified Linear Unit (ReLU), and Hyperbolic Tangent tanh. The most used function is ReLU as it helps with the training of non-zero results [5][10]. Another layer of a convolutional neural network is the flattening layer. This layer is used to flatten the output of a convolutional layer and create a single long vector [5]. The last layer of a CNN after the flattening layer, is a simple dense layer [5].

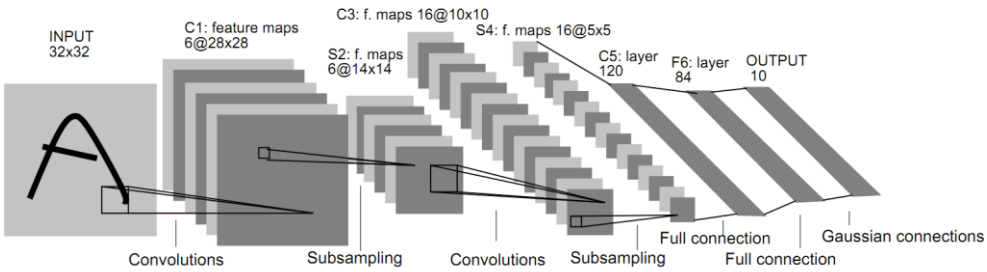

Figure 1: Architecture of LeNet-5, a Convolutional Neural Network, here for digit recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical
Image from: https://pythonmachinelearning.pro/wp-content/uploads/2017/09/lenet-5.png.webp

Popular CNN architectures are AlexNet, GoogleNet, and ResNet [5][10]. Some machine learning models created for mobile devices, which have drastically smaller size so they can run efficiently on mobile and embedded vision application, are Yolo Tiny, MobileNet SSD, and EfficientDet Lite. A frame is captured using the mobile's camera, and that is forwarded along with the set of classes (labels) to the CNN [5]. A threshold is set with the desirable accuracy, and if it is satisfied, a detection or classification is made [5]. Some of the best machine learning libraries for such tasks are TensorFlow, PyTorch, and Keras. Keras is a neural network library, while PyTorch and TensorFlow are open-source libraries for more machine learning tasks [5]. Keras, TensorFlow and PyTorch can be used to build a ML model too[5]. Moreover, a good dataset is required to build a model [5]. Existing good datasets include Microsoft COCO Dataset, ImageNet, and various Cloud APIs such as Google's Vision API (via Firebase ML kit) and Microsoft's Computer Vision API which are powerful tools supporting ton of functionalities with access to huge datasets. ML kit brings Google's machine learning expertise to mobile developers in an easy-to-use package. A state-of-the-art library for image processing is OpenCV.

2.3. Optical Character Recognition

Optical Character Recognition (OCR) for text detection is one more application of computer vision. OCR is used to identify printed or handwritten text characters in images and video frames. Then, it extracts the identified text into a machine-readable character stream [5][2]. Text recognition involves a lot of processing done in phases. These phases include pre-processing, segmentation, feature extraction, classification, and recognition [11]. The output of one phase is the input of the next [11].

One of the best text recognition APIs that can deploy OCR on mobile devices is Google's Firebase ML kit. Google's OCR system is developed using a modern neural network [12].

2.4. Speech Synthesis

Speech Synthesis for audio feedback to inform people of their surroundings is applied. Based on written input, a processing is being made to generate spoken language [5]. It involves natural language analysis and conversion to sound (Text-To-Speech) [5][8]. It is the process of parsing the text and extracting information [5]. A good audio module for Text-To-Speech is Google Text-To-Speech API.

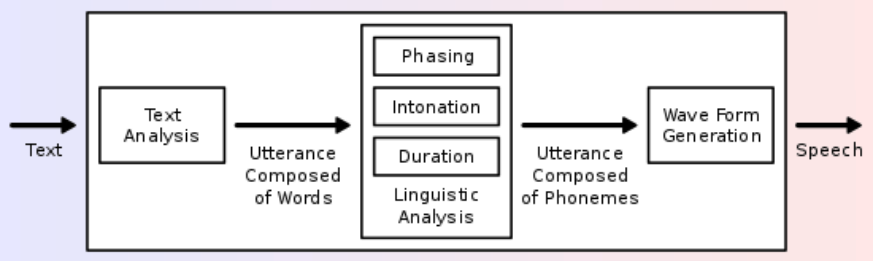

Figure 2: Overview of a typical TTS system
Image from https://en.wikipedia.org/wiki/Speech_synthesis

3. LITERATURE REVIEW

Mobile application development has been progressed dramatically the past few years with people creating constantly applications for the visual impaired. Some of the applications created include Aipoly Vision, BlindTool, Iris, and TapTapSee. These existing systems will be examined and compared with the proposed "BlindVision" application.

Aipoly Vision is an application used as a sight for blind and visually impaired people developed by V7 Ltd [13]. It has implemented object detection and color recognizer in the app to understand surroundings. Aipoly can recognize objects from one thousand categories, different dishes of food, and different species of plants [14]. Furthermore, it can read texts from twenty-six different alphabets [14]. The mobile application can work in real-time fast and does not require any internet connection [14]. It is a cross-platform application [14]. Additionally, text-to-speech is implemented in the app which is used to speak out the recognized objects [13]. A worth mentioning Aipoly Vision's feature is the Intelligent Torch [13]. This feature detects dark environments and enables the phone's flashlight when needed to enhance the detections [13]. The app supports single object detection [2]. Aipoly uses MobileNet SSD by CloudSight API [2].

TapTapSee is another similar technology developed by CloudSight Inc. [13]. It has implemented features that help identify surroundings and notify users aloud [15]. Its features include picture recognition, 10 seconds video recognition, upload pictures and videos from camera roll for recognition, save already identified images and videos, and automatic flash toggle in case of a dark scene [15]. TapTapSee is a cross-platform application [15]. The biggest strength of TapTapSee is not only determining the type of the object, but also their color and that highly accurate [13]. The mobile app uses VoiceOver and TalkBack, for Apple and Android devices respectively, for the audio feedback, which is limited because not all phones come with these settings [13]. Therefore, visually impaired people will be unable to get audio description of the recognitions unless they enable TalkBack or VoiceOver in their phone settings [13]. The app supports single object detection [2]. Text recognition is not supported [13]. TapTapSee uses MobileNet SSD by CloudSight API [2].

Another developed application for the visually impaired people is BlindTool. BlindTool has been developed by Joseph Paul Cohen. The developer created an application that can understand and classify objects from the ImageNet dataset (up to one thousand objects). The app can detect objects when pointing the camera to something and output the result aloud to the users [13]. One fascinating feature implemented in the app is the confidence vibration. When a detection is made, the phone vibrates according to the recognized accuracy, letting the visually impaired people know if the app has given a confident result that could be trusted [13]. The app can be used for single object detection [2]. BlindTool is an Android native application [2]. Its biggest weakness is the slow recognition process due to the huge dataset [13]. BlindTool lacks automatic flashlight toggle and OCR[13].

Lastly, one more worth mentioning mobile application, is Iris by Ismail Sahak, Ong Huey Fang, and Syuhada Abdul Rahman [2]. Iris' worth noting features include auto-flash toggle, multiple object detection, in-depth object analysis, and text recognition. The app uses Microsoft's Computer Vision API, which often produces in-depth descriptions of multiple objects [2]. The results are passed to the users with Speech Synthesis techniques [2]. Iris is an IOS native application [2]. The biggest flaw of the app is needing the frame to be captured by the user. The captured frame then, is sent to the API in order to get the results, something that can cause delay.

The table below (figure 3) presents a comparison between the proposed "BlindVision" application and the other existing ones. Some table data taken from references [2] and [13].

Application / **Functionalities**	AiPoly Vision	Iris	TapTapSee	BlindTool	BlindVision
Object Detection	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
OCR	**Yes**	**Yes**	**No**	**No**	**Yes**
Image Labelling	**No**	**Yes**	**No**	**No**	**Yes**
Color Detection	**Yes**	**No**	**Yes**	**No**	**No**
Audio Feedback	**Yes**	**Yes**	**No**	**Yes**	**Yes**
Multiple Object Detection	**No**	**Yes**	**No**	**No**	**Yes**
Speech Recognition	**No**	**No**	**No**	**No**	**Yes**
Scene Description	**No**	**Yes**	**No**	**No**	**Yes**
Flash Toggle	**Yes**	**Yes**	**Yes**	**No**	**No**
Platform	**IOS / Android**	**IOS**	**IOS / Android**	**Android**	**Android**

Figure 3: Related applications comparison

4. DESIGN AND IMPLEMENTATION

4.1. System Development

System requirements need to be set, as they are necessary for a healthy, and efficient development. Knowing the system's requirements, especially the major ones, from the beginning makes the entire process for the developers much clearer because they can follow certain steps to achieve goals. After setting the app's requirements, tools that will be used by the developers (first being the development framework) should be found. Lastly, the required hardware needs to be found in order to start the system's development. The methodology followed for the mobile application development is Agile. Agile allows plan changing according to the needs of a customer. Moreover, customer collaboration is involved, something that can change the shape of the application. Agile sees the following steps: Plan-Design-Develop-Test-Release-Feedback. New ideas were welcome throughout the year, which improved the proposed application. Plan changes were accepted according to the development process and project needs. The initial planning of the application was based on related created applications. Firstly, main functionalities such as object detection were more crucial to be developed as they belong to the main system requirements. After designing and creating the application with the main requirements, new functionalities and ideas were welcome.

4.1.1. System Requirements

System requirements of "BlindVision" are stated below, divided to main and secondary requirements.

Main system requirements:

- System must have a user interface for user – system interaction
- System must make use of the device camera
- System must implement Deep Learning Computer Vision techniques for Scene Understanding
    - System should implement Object Detection
    - System should Implement Text Recognition (Optical Character Recognition)
    - System should implement Image Classification
- System must use Text-To-Speech tools for audio explanation of the environment
- System must be able to work in real-time, providing as accurate results as possible

Secondary system requirements:

- System should use a friendly user interface that helps interaction with people with low vision
    - User interface colors should be friendly for the eyes
    - User interface should have big icons and buttons for easier navigation
- System should implement Speech Recognition to launch scene understanding with voice
- System should execute a welcome message and instructions for the users
- System should implement feature for detection mode selection
- System should test and evaluate different Object Detection models
- System should use device sensors for shaking gesture understanding in order to stop the scene understanding
- System's audio feedback should be able to be turned off

The system is intended to be used by people with impaired vision and make the invisible audible.

4.1.2. System Software Tools

Before starting with the application development, the development framework needs to be chosen. Later, more software technologies can be found and used to help the developer build the system. Bellow, all the software tools used for the development of BlindVision are stated.

- BlindVision is a native android application. Android Studio has been used. BlindVision has been programmed using Java language. Extensible Markup Language (XML) was used for building the user interfaces. Android Studio gives access to Android Developers tools too, which include useful functionalities.
  Android SDK version used: 29
  Android version release: 10 – Android Q

Figure 4 Android Studio Logo
*Image from Android Studio Logo free download - 750*450, 75.89 KB (subpng.com)*

- Image processing of the frames captured by the device' camera was done using OpenCV SDK

Figure 5 OpenCV Logo
Image from1200px-OpenCV_Logo_with_text_svg_version.svg.png (1200×1478)

- TensorFlow Lite Object Detection models were interpreter using TensorFlow API

Figure 6 TensorFlow Lite Logo
Image from https://www.tensorflow.org/site-assets/images/project-logos/tensorflow-lite-logo-social.png

- Google's Firebase ML Kit was used for implementing Image Labelling and Text Recognition (OCR)

Figure 7 Google's ML Kit Logo
Image from ml-kit-logo.png (1600×468) (bp.blogspot.com)

4.1.3. System Hardware

The system development can start once all required hardware is available. This project's required hardware is an android smartphone with a camera. For development and cost purposes, a mid-range phone was selected as the main hardware device. A mid-range device has relatively low cost, and its mid-specifications suggest that the application performance is tested on a device that will cover most people. If the performance is good on such a device, other more powerful devices will have zero issues.
The device selected for BlindVision development is Huawei P30 Lite. Its specifications are specified below and on figure 8.

- Model Used: MAR-LX1A
- Android Version updated to 10
- EMUI Version updated to 10.0.0
- Back cameras: 48MP + 8MP + 2MP
- Front camera: 24MP AI Selfie Superstar
- Processor: Huawei Kirin 710 (8 cores)
- Memory: 4GB RAM + 128GB ROM
- Battery: 3340 mAh
- Weight: 5.61 oz
- Display: 6.15 inches

Figure 8 Huawei P30 Lite specifications
Image from https://m.media-amazon.com/images/I/617FhjWOYXL._AC_SL1000_.jpg

4.2. System Design and Architecture

After choosing the methodology, setting the system's functional requirements, searching for proper software tools, and having all the hardware needed to develop such a system, coming up with the system's design, and architecture is essential . Software architecture and design is the foundation of the system. This will make the entire development process much more straight forward, as it will shape the project needs, and make any upcoming decisions much easier. A clever and good design will have a profound effect on the quality of the system, something that will increase the chances of the application succeeding.

Figure 9 below captures the system architecture of BlindVision adopted from Iris application research paper[2]. (All diagrams were created with Lucidchart).

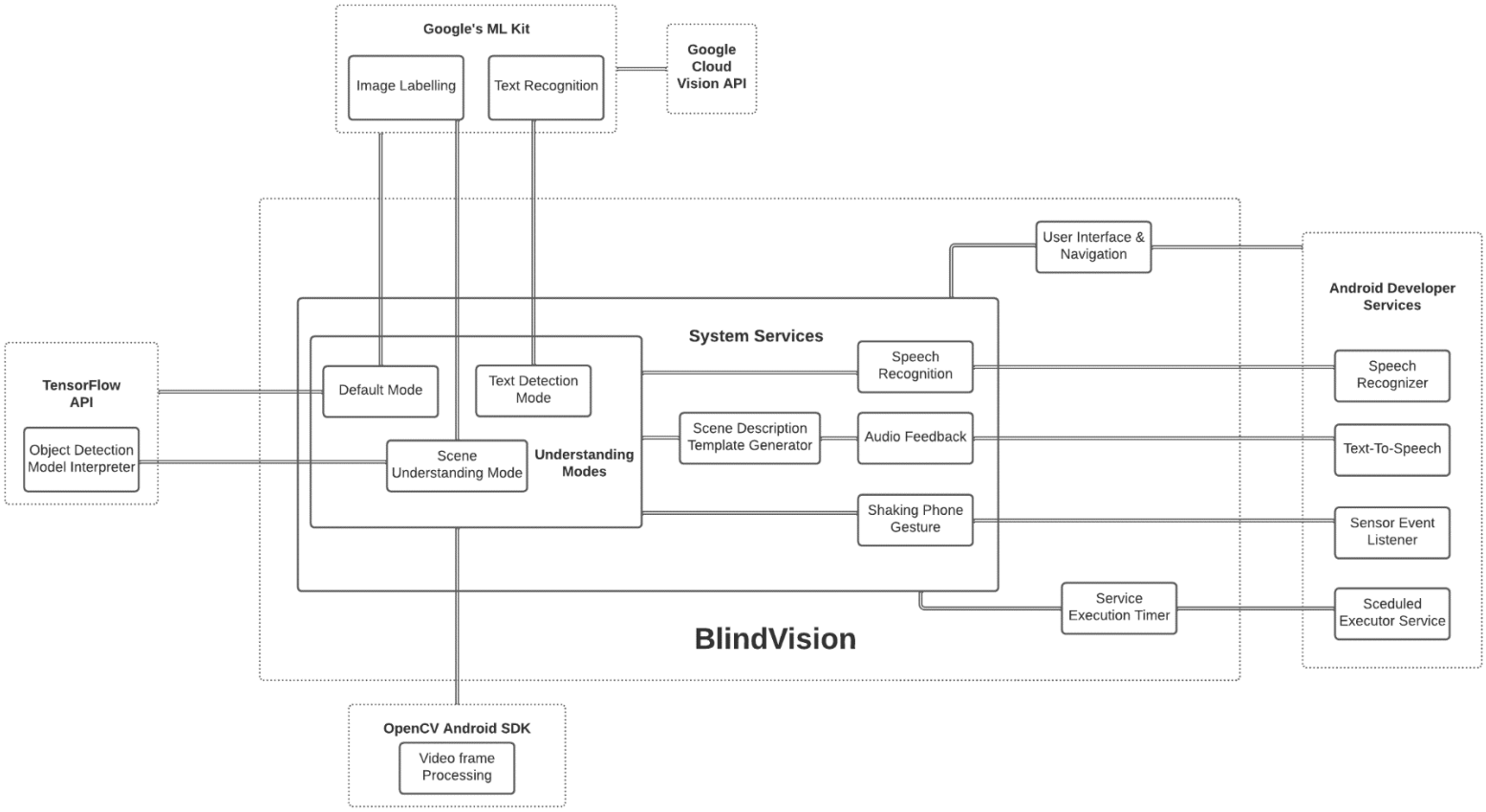

Figure 9 BlindVision system architecture

BlindVision's architecture consists of seven services, implements two APIs, uses one SDK, and adapts android operating system features via the Android Developers Services. Android operating system is used which gives access to plenty of ready-to-use features for the application development. Android developers service works more like an SDK library with access to ton of interfaces and Google's APIs tools. Android services provide useful tools to build the application's pages, design them, and establish the navigation between them. Another android developer's service used is the SpeechRecognizer for adding speech recognition features in the app. Text-To-Speech service is implemented to add the audio feedback feature. Additionally, a Sensor Event service was utilized in the app. This service gets access to the phone's sensors and helps to determine if the users are shaking the device. A Schedule Executor Service is applied for scheduling the recognitions with timer. This has been adapted from the android developers too. Moreover, frame capturing, and processing is done

with help of an external SDK library, OpenCV for android. This SDK gives access to various image processing tools and makes the entire process easier. The application makes use of the Google's Cloud Vision API via the ML Kit for the tasks of image labelling and text recognition. Note that the Google's Cloud Vision used is the free version, not the paid one. Furthermore, TensorFlow API is being utilized to interpret and execute object detection TensorFlow Lite models in order to perform object detections. When object detection with Yolo model was implemented, Darknet API was used as well. Now the system does not make use of the Darknet API. The structure idea of the design is adapted from Darren Tan Yung Shen's report[13].

Figure 10 below captures the system's use case diagram. In the diagram, interaction of user – application within the BlindVision system is represented.

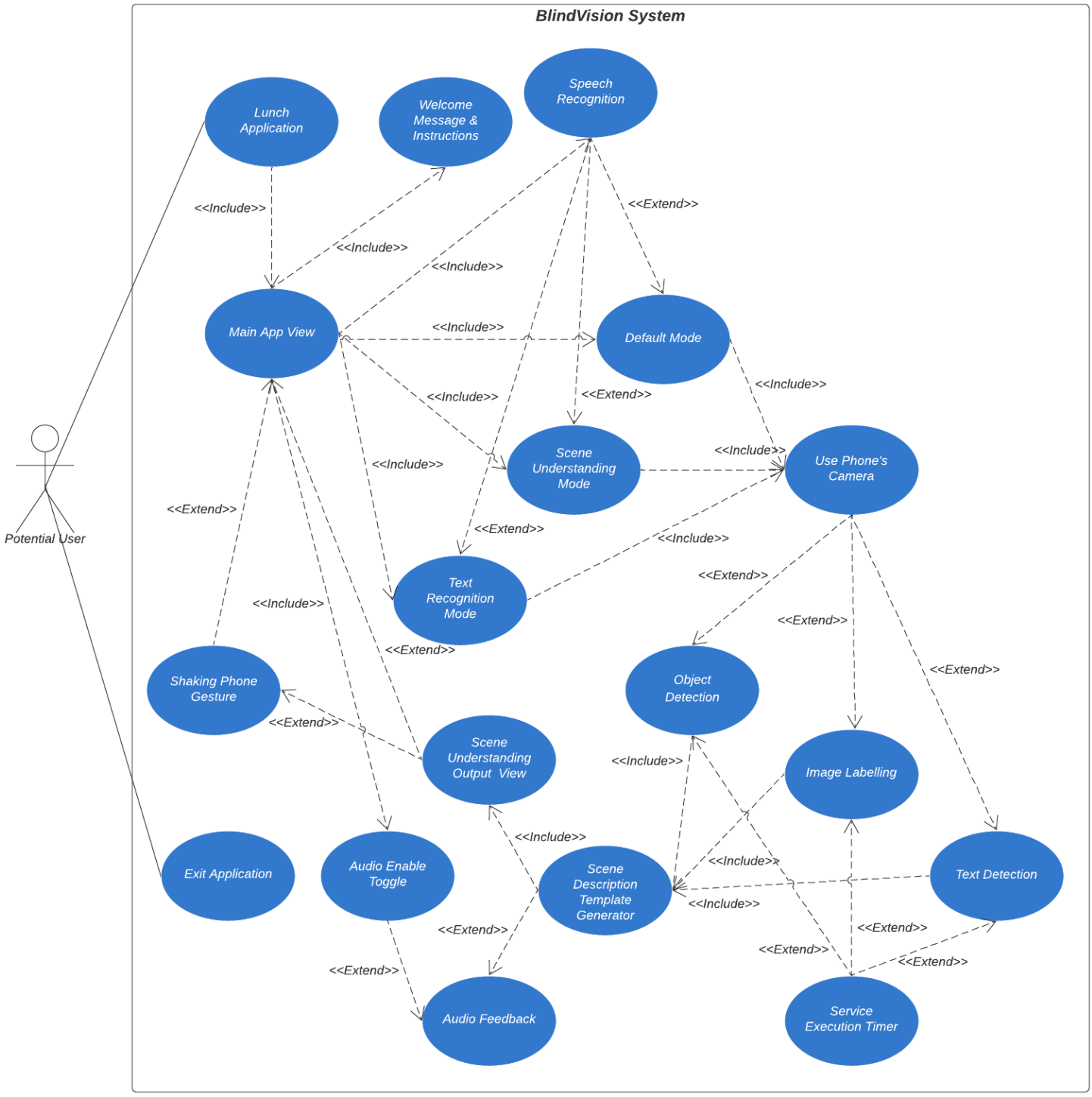

Figure 10 BlindVision use case diagram

Application's design is a crucial aspect of the entire project as it is addressed to people with low vision. The system's design should be able to help people interact with it effortlessly. Hands-free interaction was the ultimate goal. The development process of the application should start from something simple that implements the most important goals, and then, with version updates, add more features and functionalities. When the application prototype is ready, testing and evaluation should be carried out to test the design and implementation of the internal structure of the system (white box and black box testing). After software testing, potential users should carry out a product testing. Lastly, user's feedback should be considered so new improvements and fixes are made.

The application has been designed to have four activities and two view pages:

- Main page view activity
- Default recognition (object detection + image labelling + text recognition) activity – Detection view page
- Scene understanding (object detection + image labelling) activity – Detection view page
- Text Recognition activity – Detection view page

Below, figure 11 shows main page view activity diagram:

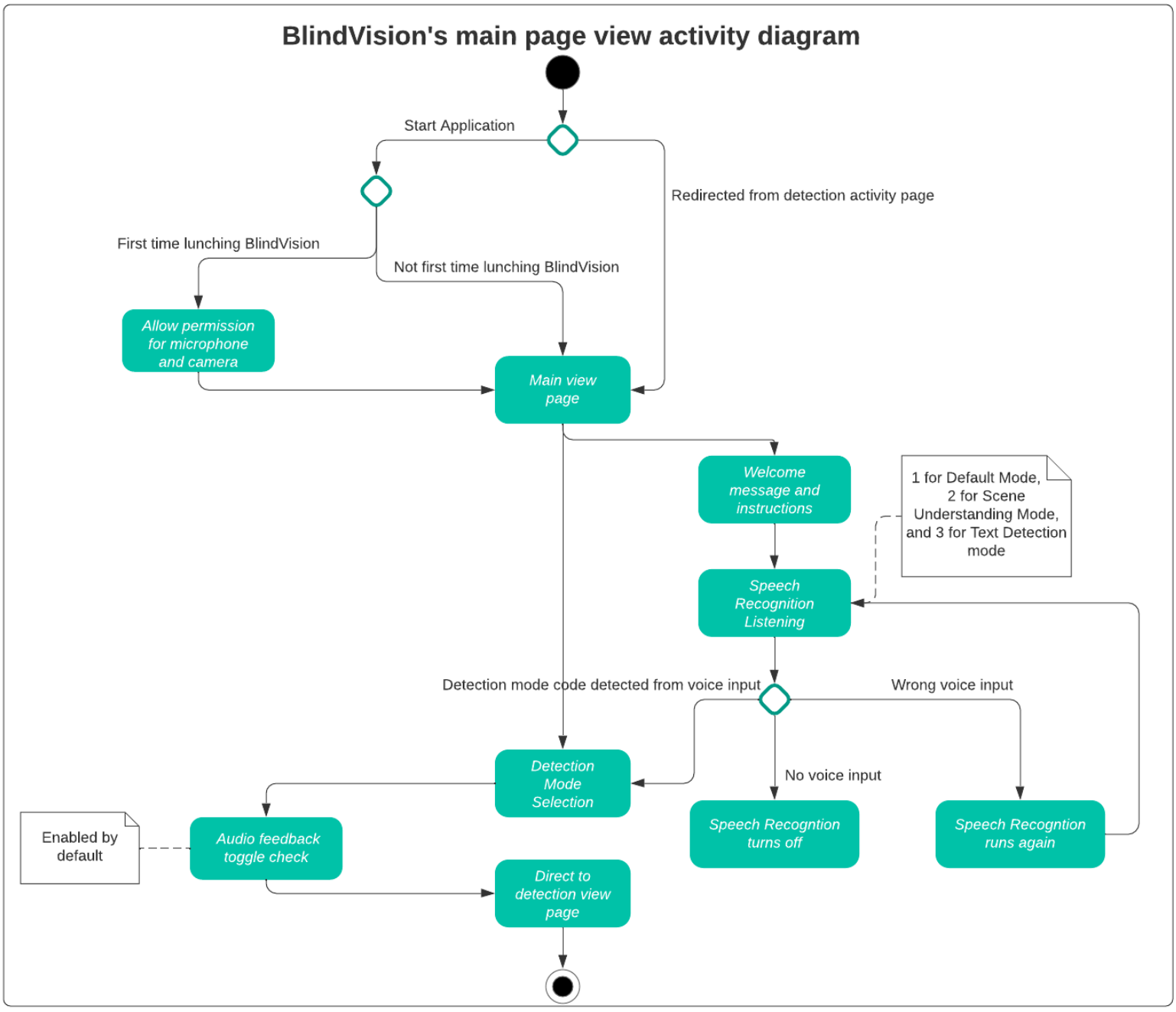

Figure 11 BlindVision's main page view activity diagram

The main view page activity is the application's startup view. When the application is launched, users are prompted there. If the application is launched for the first-time, users must allow the system to make use of mobile's camera and microphone. After accepting these requirements, application can be used. Main page view activity is meant to give brief instructions to the potential users and access to all the offered modes. When starting this activity, the system executes a welcome message along with instructions of how to use the application. The instructions include details on how to use the speech recognition to access the three detection modes with voice, and how to return to the main page when users are within the other three activities (shaking gesture). Users can also use buttons for their navigation within the app. Audio feedback toggle for disabling / enabling audio feedback can be also seen in the main activity. Audio toggle is being checked to determine if the system will use audio feedback or not.

Below, figure 12 shows default recognition's activity diagram:

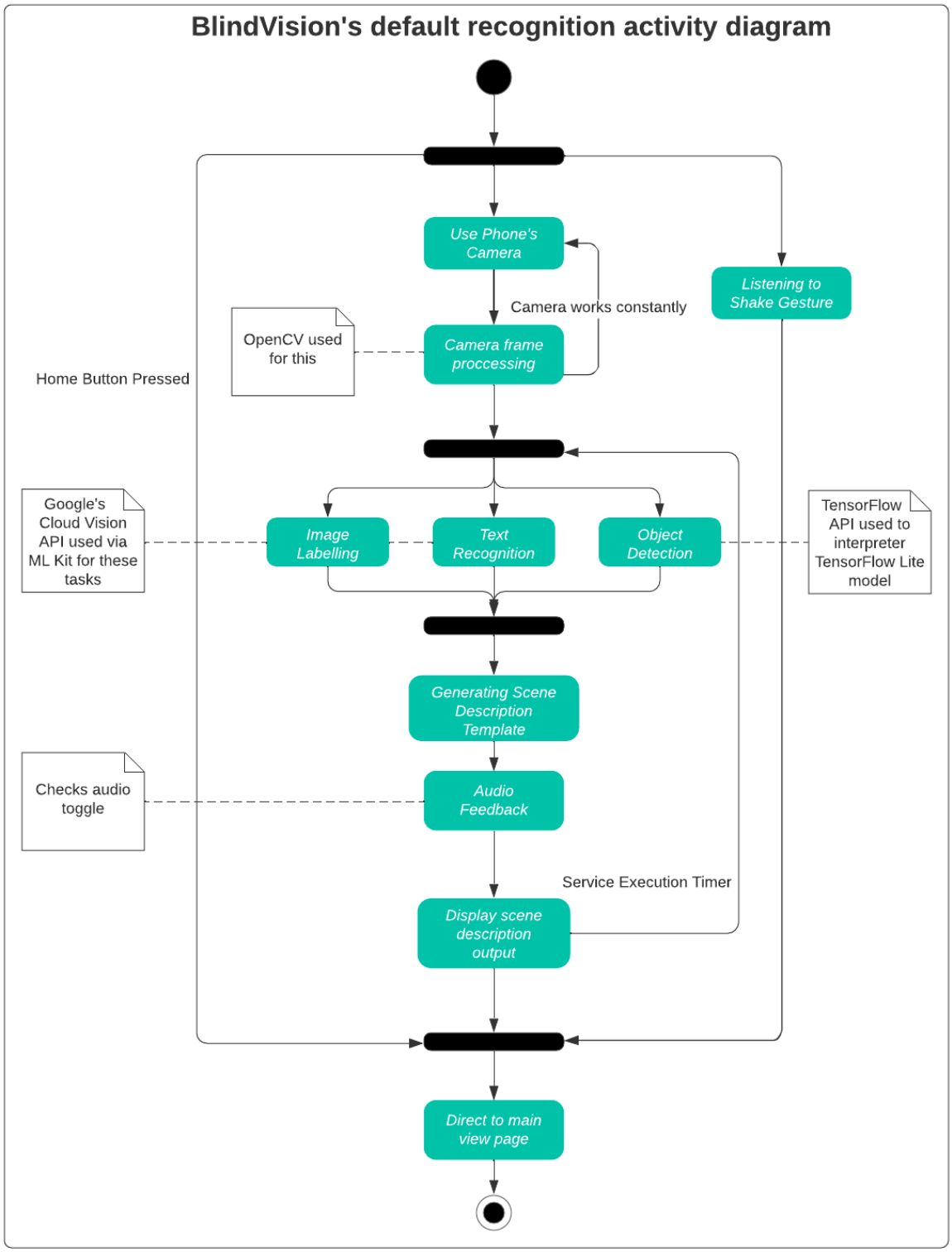

Figure 12 BlindVision's default recognition activity diagram

The default recognition activity is the application's most functional activity, and the one that is aimed to be used more by the users. This activity belongs to the detections' view page and is used as the default mode of recognitions. It uses all three functionalities for scene understanding including image labelling, text detection, and object detection. All three functionalities are conducted in parallel. To redirect to the main view page, a home button can be pressed, or a listener for shaking detection is working in the background for a hands-free navigation. When a detection is made, a scene label is generated. This label is fed or not (according to the audio toggle information passed to this activity) to the speech synthesis module and finally outputted to the users. The whole recognition service is executed under a scheduled timer.

Below, figure 13 shows scene understanding recognition's activity diagram:

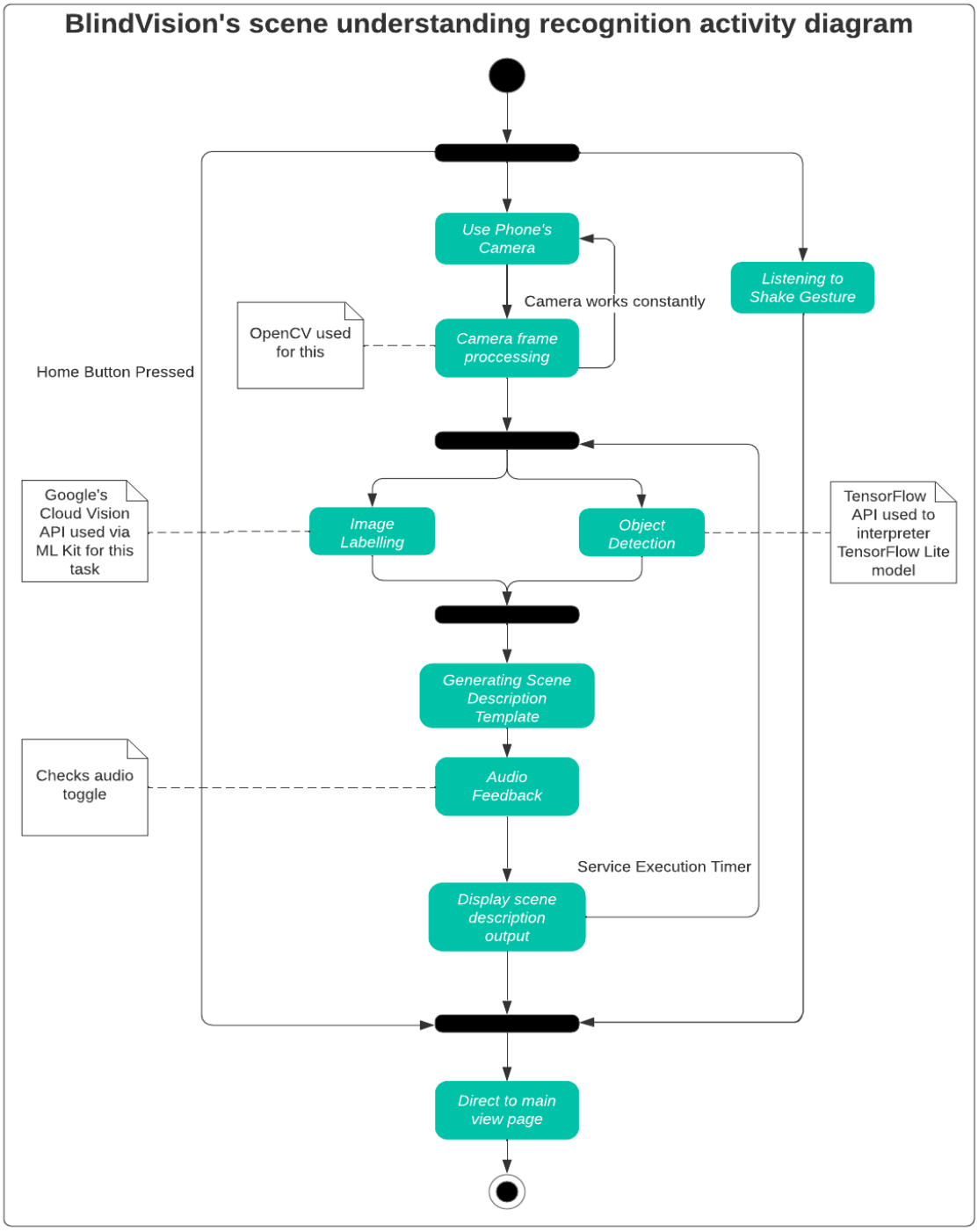

Figure 13 BlindVision's scene understanding recognition activity diagram

The scene understanding recognition activity is the application's activity dealing with objects, places, activities, animals, products, and more. It is the one aimed to be used more when users care more of their surroundings excluding texts. This activity belongs to the detections' view page and is used as the scene (understanding) mode of recognitions. It uses the two functionalities for scene understanding including image labelling, and object detection. The two functionalities are carried out in parallel. To redirect to the main view page, users can press the home button, or shake the phone for a hands-free navigation. When an object detection or image classification is made, a label is generated. This label is fed or not (according to the audio toggle information passed to this activity) to the speech synthesis module and finally outputted to the users. The scene understanding recognition service is executed under a scheduled timer.

Below, figure 14 shows text recognition's activity diagram:

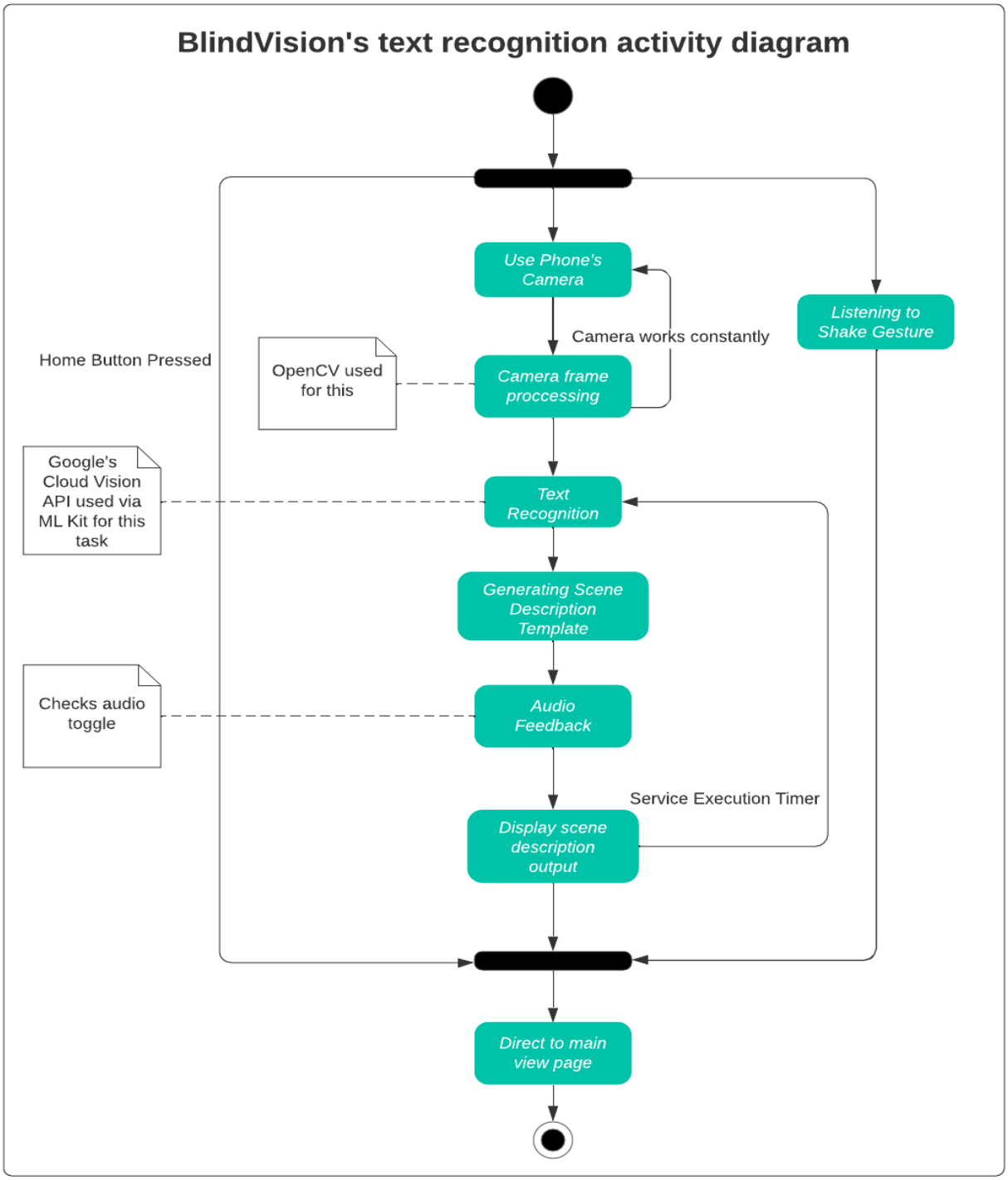

Figure 14 BlindVision's text recognition activity diagram

The text recognition activity is the application's activity dealing with text understanding. Any signs, scanned documents, text inside photos, and more, can be detected with this activity. It is the activity aimed to be used more by users when only text from a scene is needed. This activity belongs to the detections' view page and is used as text mode of recognitions. It uses the text recognition functionality alone. To redirect to the main view page, a home button can be pressed, or a listener for shaking detection is working in the background for a hands-free navigation. When a text detection is made, a label is generated. This label is fed or not (according to the audio toggle information passed to this activity) to the speech synthesis module and finally outputted to the users. The text recognition service is executed under a scheduled timer.

4.3. System Implementation

4.3.1. Implementation Start-Up and Structure

Before starting the application development, a basic knowledge for understanding Android Studio IDE is required. Android Studio works like other IDEs, and it looks similar to IntelliJ as it adapts many of its functionalities. When creating a project, the IDE provides app templates to get users started. Developers can choose the template they like and find more suitable. The developers can later change the template of the application. Having selected the starting template, the programmers can deploy and run their created application on a physical device. Android Studio also gives the option to deploy the application on a virtual phone of their choice. Below figures 15 and 16 show the top toolbar of the IDE with the options of deployment. Figure 15 shows a virtual phone as the target device for the deployment, and figure 16 shows a physical device. When plugging a physical device, an option for deploying the app there will be displayed. For the BlindVision development, no virtual device was used. Physical devices are better to be used when developing because the deployment is much quicker. Launching everything on a virtual machine makes the process slower.

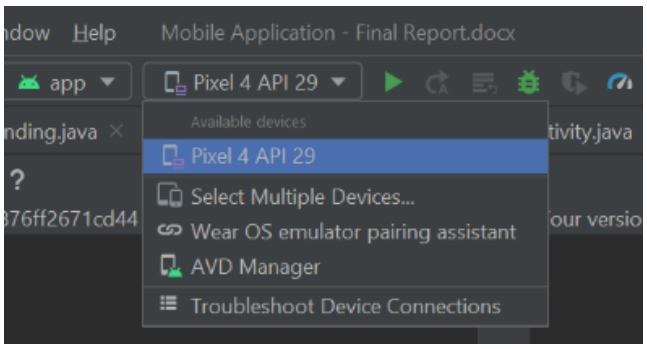

Figure 15 Application's deployment options menu (virtual device)

Figure 16 Application's deployment options menu (physical device)

The first deployment is always the slowest as the IDE installs the application on the device and applies all the settings needed. After launching the application once, app changes are deployed faster.

The next important thing on android studio, is understanding the folder's structure. The folder's structure is divided into one main module, to zero or more external modules, and lastly to the Gradle scripts. The main Java module is the module used for the application's development (designing and implementation). The external modules include libraries, SDKs, and whatever the developer chooses. The Gradle Scripts are advanced build toolkits, to automate and manage the build process. In addition, they allow developers to define flexible custom build configurations (adding dependencies for API calls, external SDKs, etc.). Figure 17 shows the three main folders of an android app.

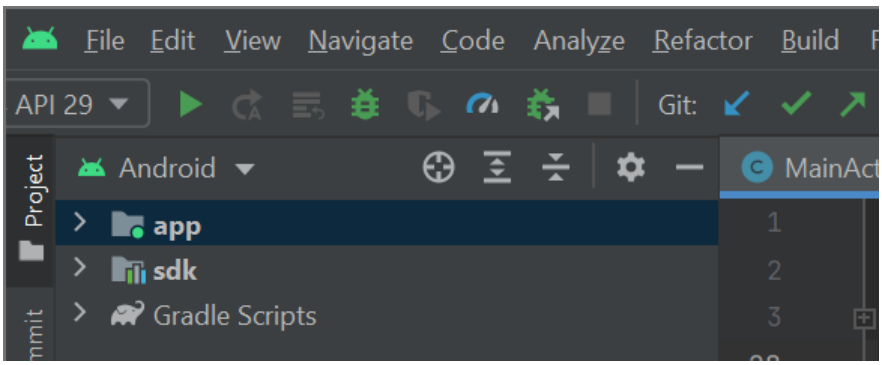

Figure 17 Android Studio's project main structure

BlindVision's development required OpenCV library tools for the frame processing. OpenCV for android SDK was added to the project. Figures 18 & 19 present the OpenCV's SDK inner folders.

Figure 18 OpenCV SDK folder

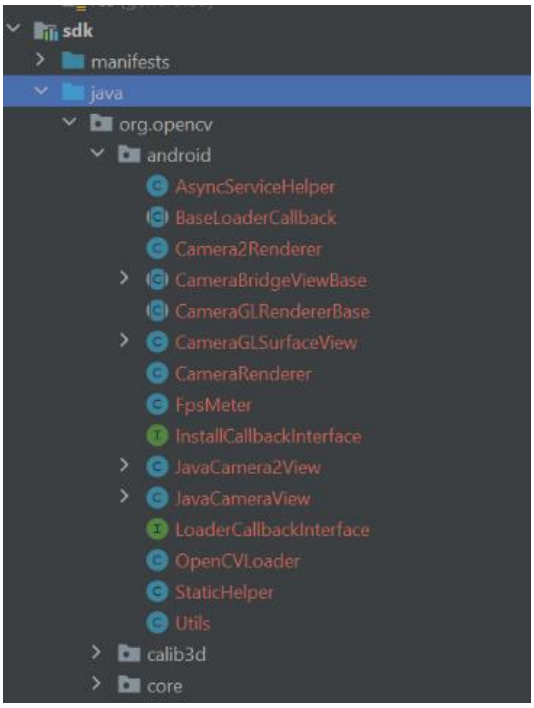

Figure 19 OpenCV SDK inner android folder

The main project's folder is used to build the mobile application. It contains four inner folders. Manifests is used to describe essential information about the app to the android build tools, the android OS, and Google Play. The Java folder is the storage of all the source code files written for the application development. It contains all the programming done (with Java) for the project functionalities. The res folder (resources) contains all the non-code sources (images, logos), XML layouts, UI strings, and more, needed for the application. It is separated to other folders for the different designing elements. The BlindVisions's user interface layouts, logos, strings, colors, fonts, and more are based in there. Assets folder is used to add external arbitrary files in the application like object detection models (.tflite), .txt files, and more. Figure 20 displays the project's resource files.

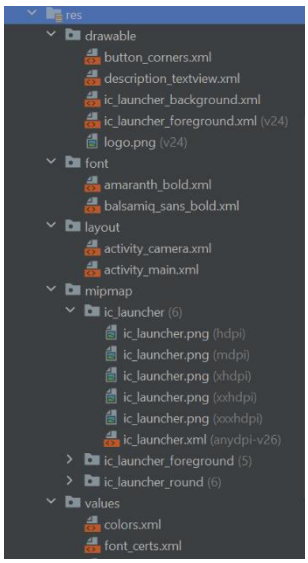

Figure 20 Project's resource folder

4.3.2. BlindVision Implementation

BlindVision has two interfaces. The first interface of the application is the main activity view page, and the second is the detection view page. The main activity page view is meant to be used when launching the application as the “menu” page. The second page view interface adapts three activities. The default mode recognition activity, the scene understanding mode activity, and the text recognition mode activity menu. The code structure is displayed on figure 21 listing all the created classes.

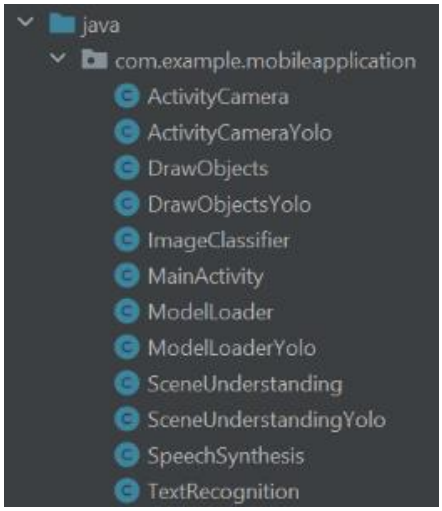

Figure 21 BlindVision's code structure

BlindVision’s development required twelve Java classes. All the above classes implement different functionalities to the app. The main classes responsible for the connection between the design module and the services modules are *MainActivity*, *ActivityCamera*, and *ActivityCameraYolo*. Based on these three classes, the other remaining classes, build on top with different functionalities.

4.3.2.1. User Interface – Main Page

The *MainActivity* class is responsible for the main activity view page, and the other two for the activities on the detection view page.
Figure 22 below shows the main user interface of the application.

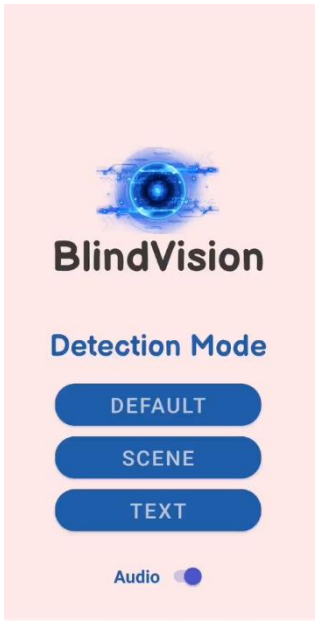

Figure 22 Main view page interface
App Logo taken from https://www.jing.fm/clipimg/detail/14-144252_and-blue-eyes-eye-light-science-technology-clipart.png

Main user interface is responsible to give a welcome message to the users, instruction on how to use the speech recognition and gestures, give the detection modes choices, and an option to disable or enable the audio feedback via a toggle button. The *activity_main.xml* file is responsible for the interface layout. The *MainActivity*.java class is the one communicating with the activity_main.xml for the functional part of the interface. This layout of this interface is shown on figure 22. It consists of the application's name and logo, of a text label along with three buttons for the three detection modes, and of an audio toggle button which is enabled by default. The *MainActivity*.java class is in control of the buttons' functionalities, of the welcome message and instructions, and of the speech recognition service. Its structure is presented on figure 23. The class also makes sure that the OpenCV SDK has been loaded successfully in the application. Furthermore, it checks for permissions to let the app use phone's camera and microphone. When the activity is created, *getPermissions* function is being called which checks if the permissions have been granted. If they have not being granted, a request will be shown on the interface to prompt the users allow them. The class utilizes *ttsWelcome* and *welcomeAudio* methods, which use Google's Text-To-Speech API via Android Developers services to output instructions to the users. The message is sounded when the app launches or when users redirect to the main interface. The message outputted is the following: "Say 1 to choose Default mode, 2 to choose Scene Understanding mode, or 3 to choose Text mode. Shake Phone to return to Home Page.". Utterance Listener is being utilized to control the start and the end of the TTS. When the text-to-speech message ends, the *speechListener* method is called to listen to the users' voice input. If users don't give any input, the speech recognition service will shut down. If it gets input but the said sentence does not include any of the detection modes, the recognizer will listen again for a new input. If a detection mode is stated (1, 2, or 3), one of the three *activityChange* methods will be called which will take users to the second view page interface for the scene recognition. The *activityChange* methods will be also called if one of the detection mode buttons is pressed instead. A *modeOfDetection* variable is passed in the second view page interface to indicate which activity will be executed. A *Switch* object to indicate if audio feedback will be enabled or disabled is passed as well.

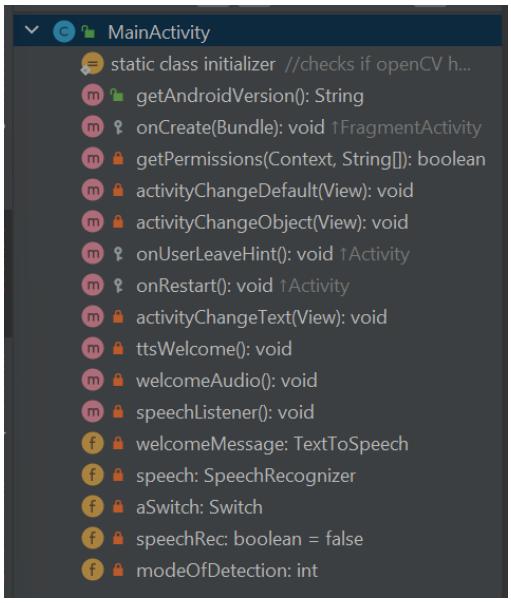

Figure 23 MainActivity.java structure

4.3.2.2. User Interface – Detection Page

The *ActivityCamera* class is linked with *activity_camera.xml* layout file to build the second view page and is responsible for the detection activities.
Figure 24 below shows the detection user interface of the application with default mode running.

Figure 24 Detection view page interface (Default Mode)

This class is responsible for most of the application's services. Figure 25 shows the *ActivityCamera.java* structure. Firstly, when this page is created, OpenCV Listener is implemented and loaded to the app. The OpenCV interface implemented (*CvCameraViewListener2*) inherits three methods, with the most important being *onCameraFrame* which is used to process each frame taken by the camera. Then, *modeOfDetection* and *audioFeedbackCheck* methods are called to determine the detection mode and indicate if audio feedback will be enabled. Follows the Text-To-Speech service initialization (*ttsAudio* method) and shaking sensor activation (*activateShakingSensor*). The shaking sensor method is used to allow hands-free experience inside the detection interface to navigate users back to home main interface as speech recognition isn't possible to be implemented on top of the audio feedback service. Using either the shaking gesture, or by pressing the HOME button, *activityChange* method will be called to navigate users back to the home page. Next, *loadModel* method is used to load the object detection TensorFlow Lite model with its labels. This method loads the model's labels from the assets folder and creates a *ModelLoader* class that loads the TFLite model and establishes the connection with the TensorFlow API for interpreting with the model.

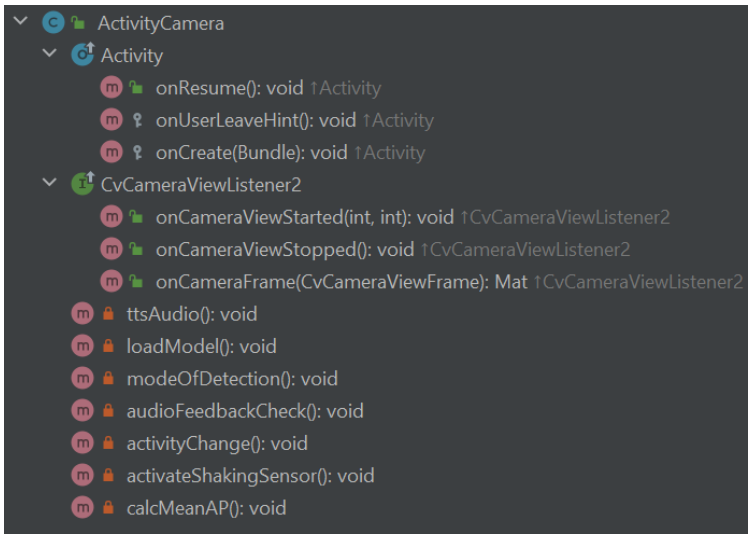

Figure 25 ActivityCamera.java structure

ActivityCamera class, creates a *TextRecogntion* instance that connects with the OCR module of Google's ML Kit API. OpenCV's *onCameraFrame* method creates a *SceneUnderstanding* instance for every frame, which is used to do all the frame processing. In the *SceneUnderstanding* class, *imageScene* method deals with all the detections' processing. First, it creates a bitmap from captured mat frames. For the text recognition and image labelling, an *InputImage* object is created from the bitmap, which is passed to the *TextRecogntion* and *ImageClassifier* classes for their processing.

4.3.2.3. Text Recognition

TextRecogntion process the image via ML Kit API tools. The text is segmented into blocks, lines, and elements [16]. Using API's modern neural network, text is recognized and segmented into three categories. A block of text is a contiguous set of text lines, such as paragraph or column[16]. A line of text captured is a contiguous set of words on the same axis, and lastly, elements are words captured in a line [16]. See figure 26 for how detected text is processed.

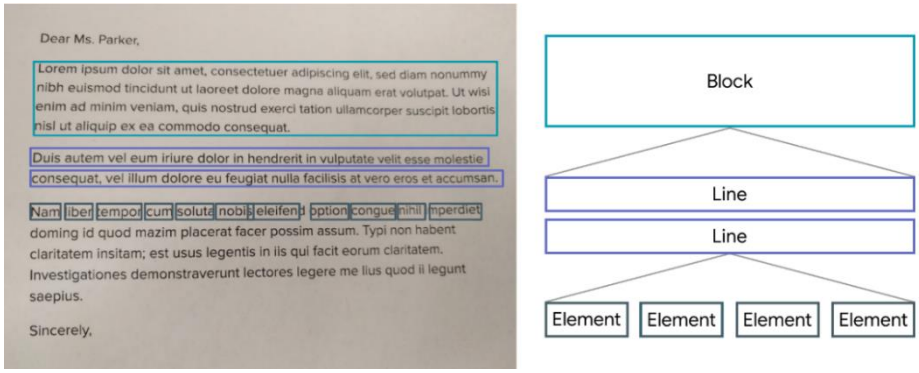

Figure 26 Text recognition segmentation
Image from https://developers.google.com/ml-kit/vision/text-recognition

The API gives results of recognized languages, text, bounding boxes, and corners [16]. All the captured blocks of text are added to a list of strings.

4.3.2.4. Image Labelling

ImageClassifier class works in an analogous way, but the API returns a list of entities, their score of confidence, and their label index [17]. Image Labelling via ML Kit can recognize more than four hundred categories of objects, places, activities, animals, and products[17]. A TensorFlow Lite model is used, which does all the processing within the API, and extracts the results. The results are text descriptions with confidences and labelled numbers[17]. The returned labels with accuracy over 80% are saved in a set of strings. For BlindVision, the base model was used which is free. Switching to Google's Firebase API, and paying for the Cloud API, can make the application recognize more than ten thousand labels. Figure 27 shows an example of Image Labelling taken from [17].

Label 0
Text: Stadium
Confidence: 0.9205354

Label 1
Text: Sports
Confidence: 0.7531109

Label 2
Text: Event
Confidence: 0.66905296

Figure 27 Image labeling example
Image from https://developers.google.com/ml-kit/vision/image-labeling

4.3.2.5. Object Detection

After finishing with the ML Kit API, the *InputImage* object is converted back to a bitmap. Later, the bitmap is scaled to match the object detection's model input size. A byte buffer is created from the bitmap (*bitmapToBuffer*) to fed that in the model as a flattened buffer of [320 x 320 x 3] byte values (for EfficientDet Lite 0 model). The object detection model outputs four arrays mapped to indexes 0-4. The final model used (EfficientDet Lite 0) can recognize up to twenty-five objects per frame. Three arrays (boxes, accuracy, classes) are used to save the coordinates of the detected boxes, the classes of the objects, and their accuracy. All the three arrays are saved in a map with their corresponding indexes. Using TensorFlow API, the input is passed. API interprets with the model, and the output is returned.

Thereafter, *DrawObjects* class is created which is used to draw the detected objects achieving accuracy over 60% according to their bounding boxes. This class also saves the labels of the detected objects in a set of strings (same set used for image labelling). Object detection example is shown on figure 28. Boolean variables are used to determine if a detection mode will be executed or not. The processed image matrix with all the drawn objects is returned and displayed on the app's interface.

Figure 28 Object detection example

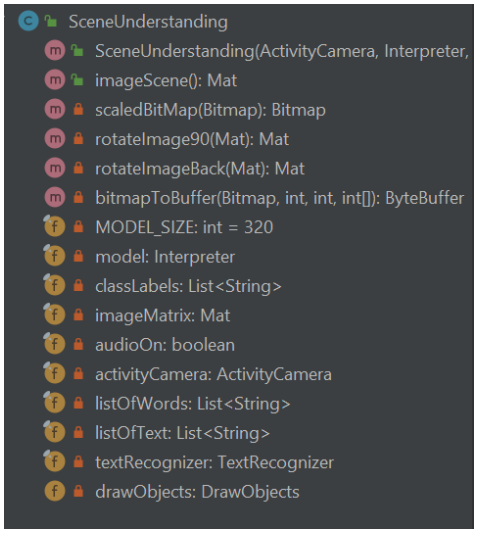

Figure 29 SceneUnderstanding.java structure

4.3.2.6. Speech Synthesis

ScheduledExecutorService is implemented to execute a runnable every two seconds. If audio feedback service is talking, the scheduled service will wait for it to end. When audio feedback is disabled, the scheduled service is not working every two seconds, but it is executing everything every frame. This runnable, checks if audio feedback is not working so it can recognize a scene. To monitor that, boolean variables are being used to control the recognitions and audio feedback. When a scene with objects, and a text are detected, a new *SpeechSynthesis* class is created passing the detected strings. *SpeechSynthesis* class is used to process the natural language of the detected items and use text-to-speech to output everything to the users. The *naturalLanguageProcessing* method calls the *audioTemplates* method which process the results of the set of entities. It creates description templates according to the logic behind seen objects. For example, if a bed and a room is detected, a template with the word bedroom will be created for a more general understanding. Another example is detecting a road and a car. Instead of outputting the objects alone “car” and “road”, a more logical template will be created saying ”road with car(s)”. This result is returned to the NLP method. Then, the method concatenates the result with the text and creates a phrase. The phrase is “Ahead is” with all the seen entities and “Text ahead saying” with the text. If no objects are detected, it will be only “Text ahead saying”, and if no text detected, phrase will be “Ahead is” and the objects. Figure 30 displays a scene understanding description which detects a tv, and a couch, and suggests that scene is a living room. If the audio feedback is enabled, the scene description is forwarded to the speech synthesis module (*audioFeedbackMain* method) and is outputted with sound to the users. For every detection, any sets, and lists containing the saved entities and texts are cleared. Generating description templates is the most challenging implementation task because the templates are manually created. Using the app daily shows new scenarios and ideas which are then implemented to enhance the scene understanding.

Figure 30 BlindVision’s Scene Understanding

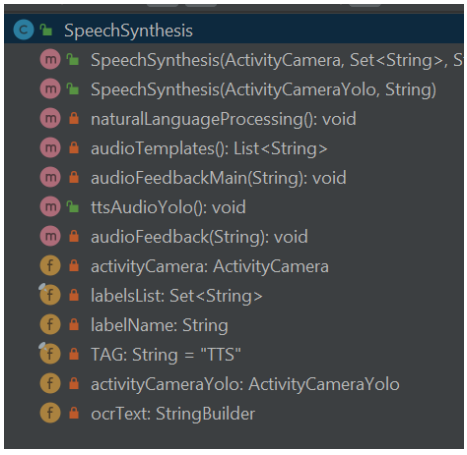

Figure 31 SpeechSynthesis.java structure

4.3.2.7. YOLO Object Detection

The remaining classes have been created to implement object detection with the Yolo model. The Yolo model implemented is Yolov4 Tiny and it is using DarkNet API for the interpretation. Model is loaded using *ModelLoaderYolo* class. There, the configuration, and weight files of the model are passed to DarkNet to get back the model. The *ActivityCameraYolo* class misses a lot of the later-implemented services because when it was implemented, only object detection and simple audio feedback were developed. YoloV4 tiny was beaten by MobileNet model in almost all fields, so decided to proceed with the MobileNet model and implement new features only there to reduce development time. *SceneUnderstandingYolo* instance is created for every frame captured, and processing using OpenCV is done to get results from the model. Results achieving accuracy over 50% are passed to the *DrawObjectsYolo* class to draw everything on the interface. Note that all the Yolo classes and methods are not used in the current version of the application.

4.3.3. BlindVision Version Control

This section states the BlindVision's development versions history.

Version 1:
Implemented simple app with basic GUI and loaded OpenCV.

Version 2.1:
Implemented Text-To-Speech for welcome message and Speech Recognition to start camera and detection. Switch button has been implemented to check if user wants to have audio feedback enable or not.

Figure 32 BlindVision version 2.1

Version 2.2:
Speech Recognition improvements.

Version 3.1:
Implemented in the app the MobileNet SSD TensorFlow lite model (trained with COCO dataset). FPS: 7

Version 3.2:
Added GPU delegate of TensorFlow for better performance. Also implemented the real-time Object Detection. FPS: 10, Mean Accuracy on testing 60%.

Version 3.3:
Added multiple object detection (up to 10 objects).

Version 4.1:
Audio Feedback implemented. Outputs the first object detected in a frame. Method implemented called → First Seen – First Out. Saving, per frame, detected object with most confidence, and outputting it with speech synthesis.

Figure 33 BlindVision version 3.1

Version 4.2:
More optimized Speech Synthesis.

Version 4.3:
More optimized Speech Synthesis.

Version 5.1:
Speech Synthesis Revised. New method implemented called → Scene Understanding (prototype). Adds detected objects in a set, and every 3.5 seconds audio is fed out back with all seen objects.

Version 5.2:
Speech Synthesis improvement.

Version 6.1:
Implemented in the app the YoloV4 Tiny model (trained with COCO dataset). Yolo also supports audio feedback (Fists Seen – First Out). FPS: 3.5, Mean Accuracy on testing 73%.

Version 6.2:
YoloV4 model Input Size reduced to 224 for better performance. Accuracy has been reduced. FPS: 6.4, Mean Accuracy 68%.

Version 6.3:
Added new feature to the app. Users can now choose model selection on the start-up. It also supports speech recognition. Say 1 for MobileNet model and 2 for Yolo model. Yolo also supports Scene Understanding (prototype) audio feedback.

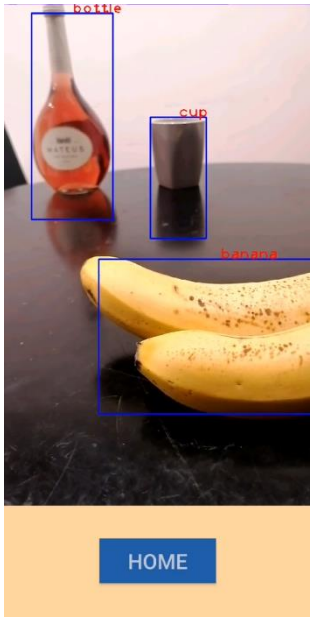

Figure 34 BlindVision version 5.1

After Interim Interview. The app uses MobileNet model for the default model as it beats Yolo model.

Version 7.1:
Added Optical Character Recognition. Implemented with ML kit of Google. FPS: 8 when not recognizing text, 6 when recognizing text and objects at the same time.

Version 8.1:
Added to the speech synthesis the text recognized output.

Figure 35 BlindVision version 6.3

New scene understanding update: Now, the scene recognition (text + objects) is executed every 2 seconds or when the audio feedback of previous recognition is ended (monitoring text-to-speech when is starting and ending). This improved the performance of a lot because when the audio feedback is working there is no need to recognize things again. Also, this could be the base for more improved scene understanding description. The only downside of this method is that the object and text boxes are drawn (in rectangles) for a millisecond. FPS: 14.5 – 15 when not recognizing, 13-13.5 when objects are recognized and 11.5 when objects + text are recognized (previously 8 fps when recognizing objects and 5-6 fps when recognizing objects and text).

Figure 36 BlindVision version 7.1

Version 9.1:
Added a new feature for detection mode selection. Now users can select or use their voice to choose between object detection, text detection or both. There are three modes. It also supports Speech Recognition. Users can say 1 for default mode (both object and text detection), 2 for object detection only, and 3 for text detection only. This also required a change at the GUI of the app. A new third button is created. The old model buttons have been removed. Now the default model is MobileNet SSD.

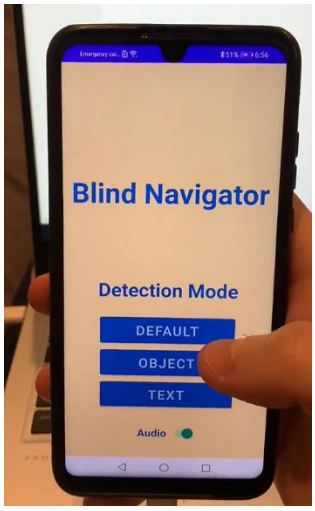

Figure 37 BlindVision version 9.1

Version 10.1:
Added a new Computer Vision technique, Image Classification. Image Labelling from Google's ML Kit has been implemented to give a further scene understanding to the users of what is ahead. The new image classification has been made to work with the already Speech Synthesis too. Also, some audio templates have been created that process the seen objects to give a better description to the users ahead (room for improvement). On the GUI of the app, a new Text Area has been created below the camera frame viewer to show the generated scene understanding description. Changed application's name from Blind Navigator to BlindVision. FPS 13-13.5.

Version 11.1:
Added new Shaking Mobile gesture feature. This feature can be used when users are in the recognition page. When in scene understanding, users can shake the phone to return to the home page of the application. Added new audio templates for better audio description. Graphic User Interface redesigned. Changed app colors and added app logo to make the it more modern and user friendly on the eyes. Finally, fixed some bugs with text-to-speech.

Version 12.1:
EfficientDet lite models (lite0, lite1, lite2, lite3) were implemented and evaluated. The models output the detection boxes, the detection classes, the detection scores, and the number of detections. The maximum number of output detections they can make are 25. These models are TFLite models using TensorFlow API like MobileNetv1 SSD model.
EfficientDet lite0 model is used as the main object detection model in the app as it performs better than the other models.

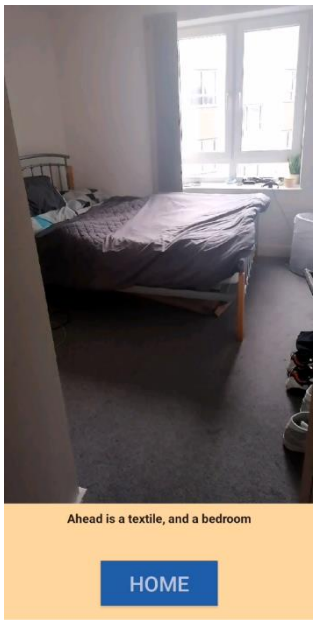

Figure 38 BlindVision version 10.1

Version 12.2:
Changed application's name (on device) from Mobile Application to BlindVision. Changed application's logo from android logo to the BlindVision logo. Change the way of drawing the objects boxes when audio feedback is enabled. Now, instead of drawing the boxes of detected objects for one frame, the boxes are drawn for 10 frames. This was done to make the boxes with the labels visible to the users for more time.

BlindVision's final size: 547mb

Figure 39 BlindVision version 11.1

Figure 40 BlindVision version 12.2

5. EXPERIMENTAL INVESTIGATION (TESTING) AND EVALUATION

5.1. System Testing and Evaluation

Agile methodology allows flexibility which is especially useful for the application development, as tests are carried out continuously and changes with new requirements if needed can be introduced. For the BlindVision app white-box testing was conducted in parallel with the development of new functionalities, and if bugs, errors, or any possible improvements were spotted, newer versions were launched to improve and fix the overall experience of the app. At the end, when every requirement was implemented, a black box testing was performed to test the functionalities of the system, the services implemented, catch any bugs existed, and test the overall performance of the app. The test strategy followed for this was a boundary value testing determining if the overall performance of the application is acceptable.

Later, after the application was tested by the developer, user testing was performed. Users with impaired vision should have tested the developed application so more accurate feedback could have been given. Unfortunately, no such users were found. Friends tested the application in their houses, in their flats, in the university, and outside, and feedback was given straight to the developer stating possible improvements, fixes, and new ideas.

Different object detection models were implemented and evaluated throughout the year. This was done to improve the application's reliability and overall performance. Some of the models tested include YOLOv4 tiny, MobileNetV1 SSD, and EfficientDet Lite0/1/2/3, all trained with the COCO dataset (80 categories of objects). A side-to-side comparison between the models was suggested because all of them were trained with the same dataset. Before the Interim Interview, YOLOv4 and MobileNetV1 SSD were tested alone, and in term 2, EfficientDet Lite models were also tested and evaluated. When Yolo and MobileNet models were compared, image labelling, OCR, and scheduled recognitions were not implemented to affect the performance. Yolo (You Only Look Once) object detection model applies a single neural network to the frame, and it divides that into regions. Then, it predicts the probabilities and the bounding boxes for each region [18]. YOLOv4 tiny implemented is communicating with the DarkNet API. The configurations and weights of the model were fed to the API. The MobileNetV1 SSD model applied, is a TensorFlow Lite model communicating with TensorFlow API for the interpretations. MobileNet SSD (Single Shot Detector) works like the Yolo model (10 detections per frame). EfficientDet models are based on EfficientNet CNN architecture which improved the MobileNetV2 model's architecture. EfficientDet Lite models introduced two new contributions in that architecture for a faster and more accurate detections (25 detections per frame). The four EfficientDet models implemented are TensorFlow Lite models found on TensorFlow Hub, and they use the TensorFlow API for interpretation exactly like MobileNet model.

Details of the models based on evaluation done in research papers.

- EfficientDet lite3 model size: 11.4mb, 37.70% mAP (Mean Average Precision) on COCO dataset, Input image size512x512
- EfficientDet lite2 model size: 7.2mb, 33.97% mAP on COCO dataset, Input image size 448x448
- EfficientDet lite1 model size: 5.79mb, 30.55% mAP on COCO dataset, Input image size 384x384
- EfficientDet lite0 model size: 4.35mb, 25.69% mAP on COCO dataset, Input image size 320x320
- MobileNetV1 model size: 4.10mb, 21% mAP on COCO dataset, Input image size 300x300
- YoloV4 Tiny model size: n/a (via DarkNet), 22% mAP on COCO dataset, Input image size 416x416

5.2. Performance Comparison

BlindVision's performance comparison was done on Huawei P30 Lite.

YoloV4 Tiny and MobileNetV1 SSD accuracy comparison (threshold used 0.5):
YoloV4 Tiny input size reduced to 224x224

- mobileNetV1 SSD --> Mean Accuracy: 0.60, Objects Detected: 4600, Time passed: 270 sec
- yoloV4 tiny--> Mean Accuracy: 0.68, Objects Detected: 1113, Time passed: 270 sec

Comparison of speed metric:

- mobileNetV1 SSD --> 10fps
- yoloV4 tiny --> 6.4fps

	MobileNet SSDv1	Yolov4 Tiny
Mean Accuracy	60%	68%
Avg. FPS	10FPS (Input frame size 300x300)	6.4FPS (Input frame size 224x224, original 416x416)
Objects Detected	Detects more objects per frame but with less confidence (more miss-predictions) Testing: Detected 4600 objects in 4.5 min	Detects less objects per frame but with higher confidence Testing: Detected 1113 objects in 4.5 min
Loading Time	Loading model is fast and light	Loading model is fast, but heavy. Need to fed conf. and weights to Darknet and the load model from there
Trade-Offs	Reduce accuracy for better performance (by reducing input frame size) – no needed	Reduce accuracy for better performance (frame size reduced to 224 x 224) - needed

Figure 41 MobileNetV1 and Yolov4 Tiny table comparison

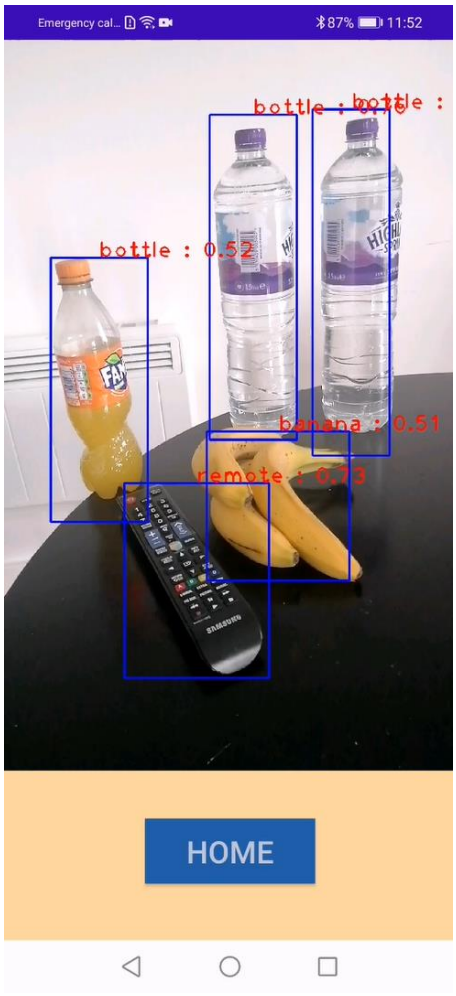

Figure 42 MobileNetV1 object detection

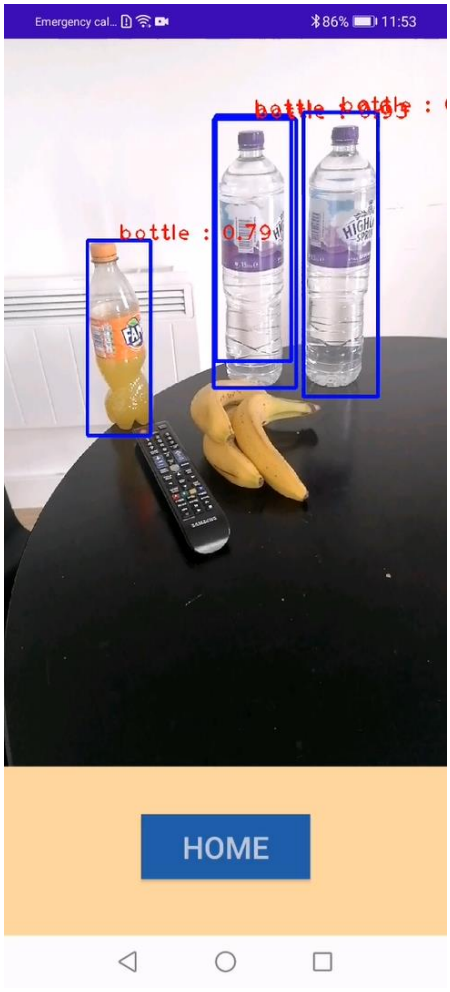

Figure 43 YOLOV4 Tiny object detection

Image Classification & Text Recognition enabled too for the following models testing.
Comparison of speed metric when schedule recognition enabled (every 2 seconds):

- efficientDet_lite3 --> 12 and when detecting -> 9.4 (3 frame drops)
- efficientDet_lite2 --> 12.8 and when detecting -> 10.3 (2.7 frame drops)
- efficientDet_lite1 --> 13.7 and when detecting -> 11 (3 frame drops)
- efficientDet_lite0 --> 14.4 and when detecting -> 12.5 (2.1 frame drops)
- mobileNetV1 SSD --> 14.9 and when detecting -> 13.3 (1.6 frame drops)

Comparison of speed metric when recognition is always active:

- efficientDet_lite3 --> 1.75 fps
- efficientDet_lite2 --> 2.5 fps
- efficientDet_lite1 --> 3.4 fps
- efficientDet_lite0 --> 4.2fps
- mobileNetV1 SSD--> 5.5 fps

Average accuracy was calculated with threshold 0.6.
Comparison of average accuracy achieved between the models (recognition always active, 2 minutes video testing):

- efficientDet_lite3 --> Mean Accuracy: 0.70, Objects detected: 158, Time passed 122.5 sec
- efficientDet_lite2 --> Mean Accuracy: 0.72, Objects detected: 211, Time passed 122.5 sec
- efficientDet_lite1 --> Mean Accuracy: 0.68, Objects detected: 216, Time passed 122.5 sec
- efficientDet_lite0 --> Mean Accuracy: 0.67, Objects detected: 223, Time passed 122.5 sec
- mobileNetV1 SSD--> Mean Accuracy: 0.65, Objects detected: 223, Time passed 122.5 sec

Comparison of average accuracy achieved between the models (recognition every 2 seconds, 2 minutes video testing):

- efficientDet_lite3 --> Mean Accuracy: 0.70, Objects detected: 55, Time passed 122.5 sec
- efficientDet_lite2 --> Mean Accuracy: 0.69, Objects detected: 48, Time passed 122.5 sec
- efficientDet_lite1 --> Mean Accuracy: 0.66, Objects detected: 46, Time passed 122.5 sec
- efficientDet_lite0 --> Mean Accuracy: 0.66, Objects detected: 42, Time passed 122.5 sec
- mobileNetV1 SSD--> Mean Accuracy: 0.63, Objects detected: 36, Time passed 122.5 sec

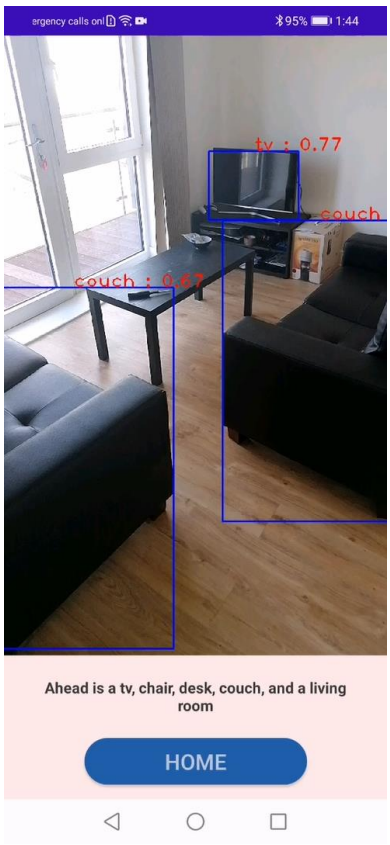

Figure 44 EfficientDet Lite 0 object detection

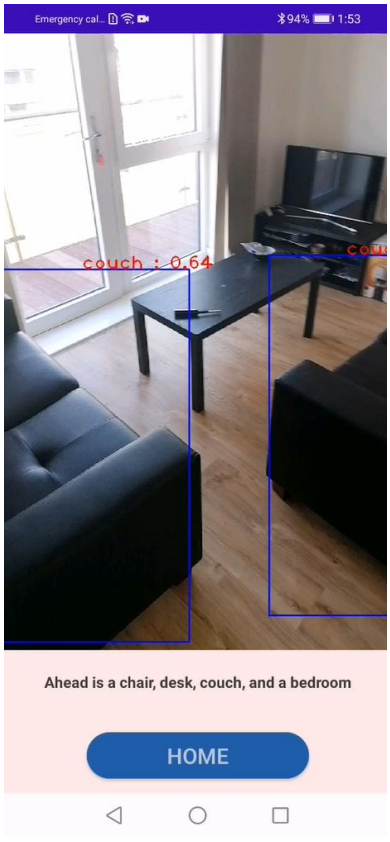

Figure 45 MobileNetV1 object detection

EfficientDet lite0 model is used as the main object detection model in the app as it outperforms MobileNetV1 in accuracy. It detects more objects in each frame because it makes detections with more confidence. Speed performance has been sacrificed slightly, but the overall performance between the accuracy-speed trade-off is better. Lastly, image labelling TensorFlow model used from Google's ML Kit achieves an accuracy of 85% (threshold 0.8 used, tested in 2-minute video). Figure 46 summarizes the performance comparison between the object detection models.

Models (Object Detection)	Mean Accuracy	Avg. FPS
MobileNetV1	65%	14.9
Yolov4 Tiny (via DarkNet API)	68%	6.4
EfficientDet_Lite0	67%	14.4
EfficientDet_Lite1	68%	13.7
EfficientDet_Lite2	69%	12.8
EfficientDet_Lite3	70%	12

Figure 46 Performance comparison table

6. PROJECT PLANNING

Project Planning helped with setting the objectives of the project and identifying the deliverables. The methodology followed the entire year was Agile. This allowed managing the project by breaking it into phases. This year's project was broken into three phases. The phases are Planning, Execution – Control, and Closeout. The milestones behind each phase are different. Planning focuses more on the background reading, on related work research, and setting project goals. Execution and Control phase aims more on the application design and implementation (mobile app, computer vision, speech synthesis, and interim oral interview), and Closeout on the final year's deliverables of the project. There were five milestones in total. In planning phase, the milestone was Challenge Week's deliverables (20-minute presentation, background reading, project goals & technical achievements, and project planning). In the execution and control phase, the milestone was the Interim Oral Interview (work quality, project organization, and Kanban record), and designing and implementing the application. In the final closeout phase, the milestones include the project's abstract and poster, the final report, a planning record, and the presentation, demonstration, and oral examination. Below, figure 47 demonstrates a work breakdown diagram, breaking down the project and visualizing all the tasks required.

Work Breakdown Diagram

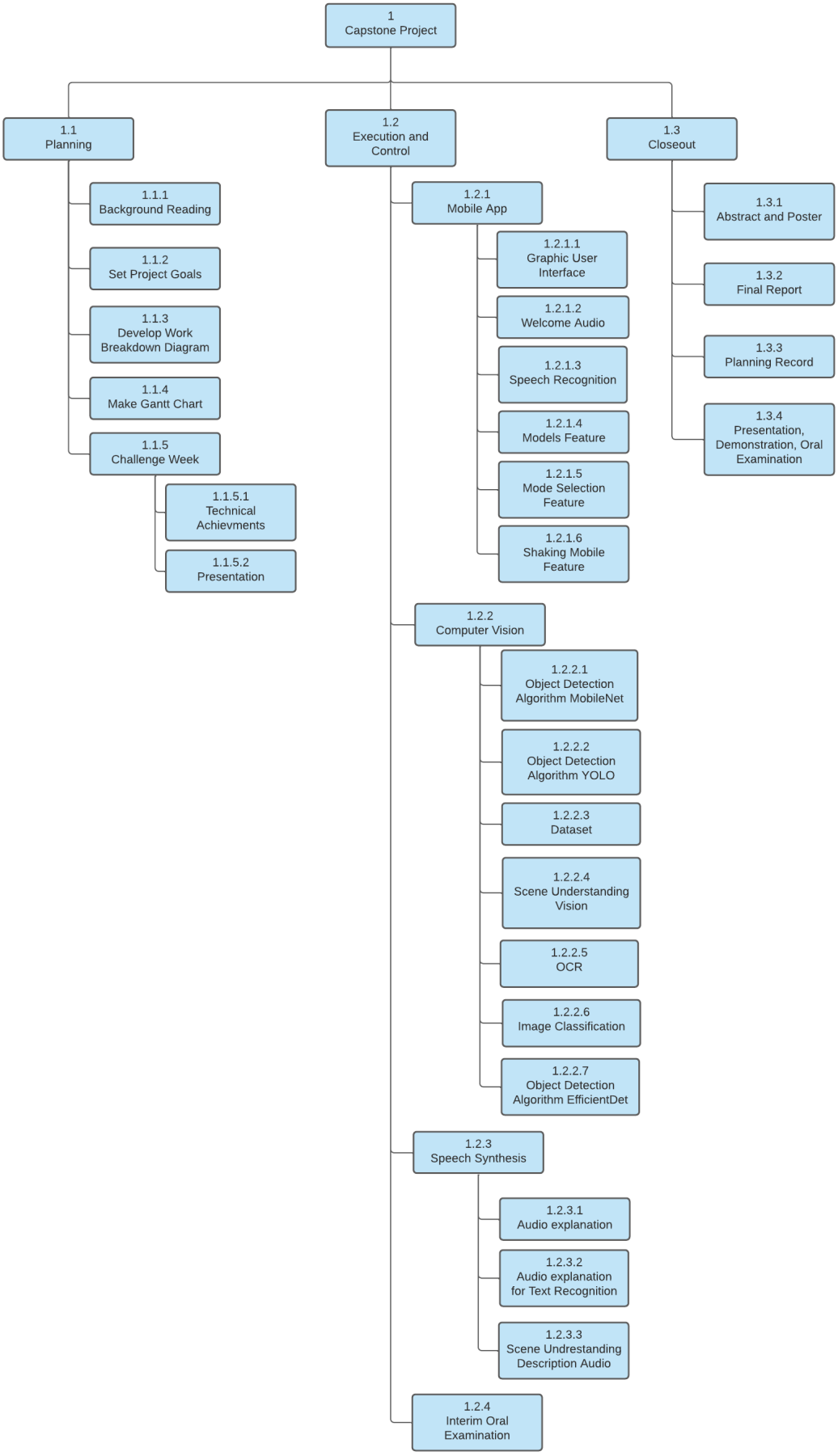

Figure 47 Work breakdown diagram

A Gantt chart has been used throughout the year, which helped with the work planning. This chart was used to monitor deadlines, task dates, milestones, and dependent tasks. Figure 48 displays project's up to date Gantt Chart (zoom in needed for better observation).

Figure 48 Gantt chart 19/04/2022

Jira was used for managing the project. It supports agile methodology making the development much easier. Phases of the development are represented with Epics and their tasks with issues. Each main issue was made up from other smaller sub-tasks. Kanban boards were used as the main project management tool to visualize work and make the project flow better. Furthermore, weakly supervisor meetings were held, which also were captured using the Kanban board. Supervisor meetings helped with the organization of the project, provided guidance, support, and shaped the project flow during the year. Figure 49 shows the latest updated Kanban board (date captured 19/04/2022) of the project and figure 50 the Kanban board captured right before the Interim Interview.

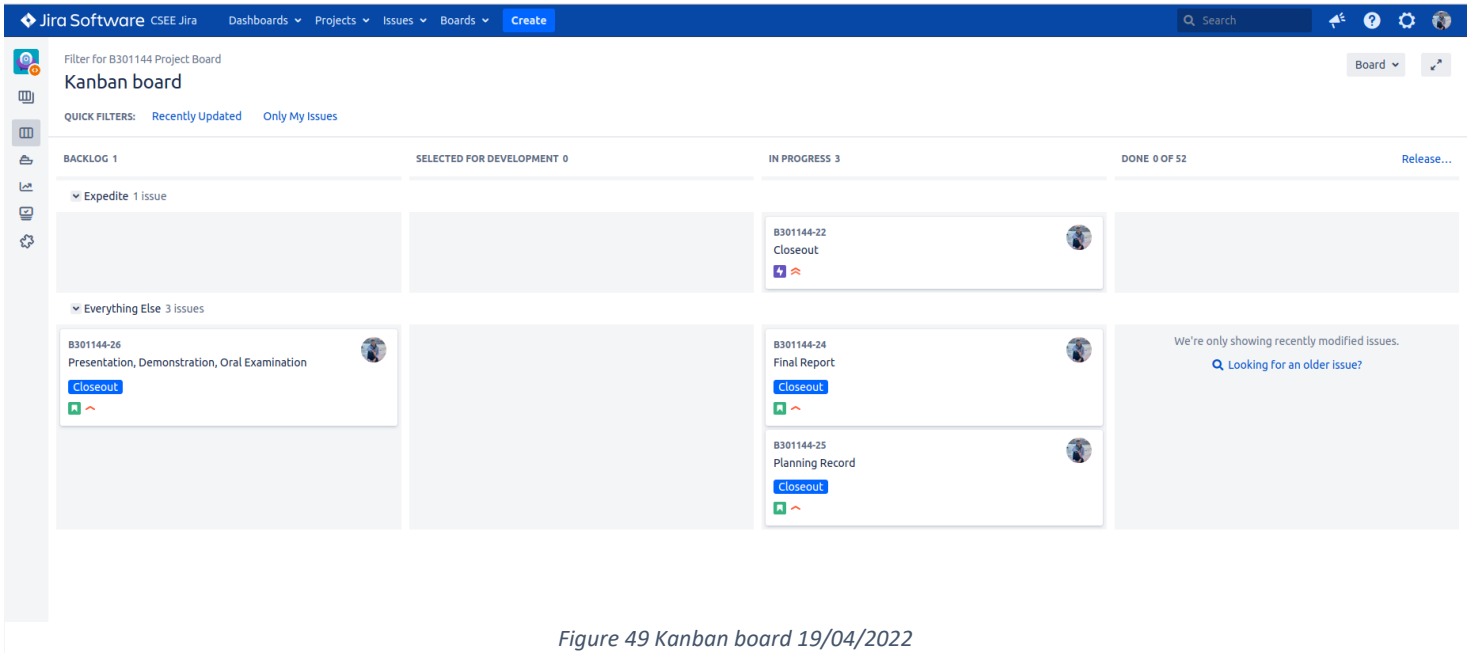

Figure 49 Kanban board 19/04/2022

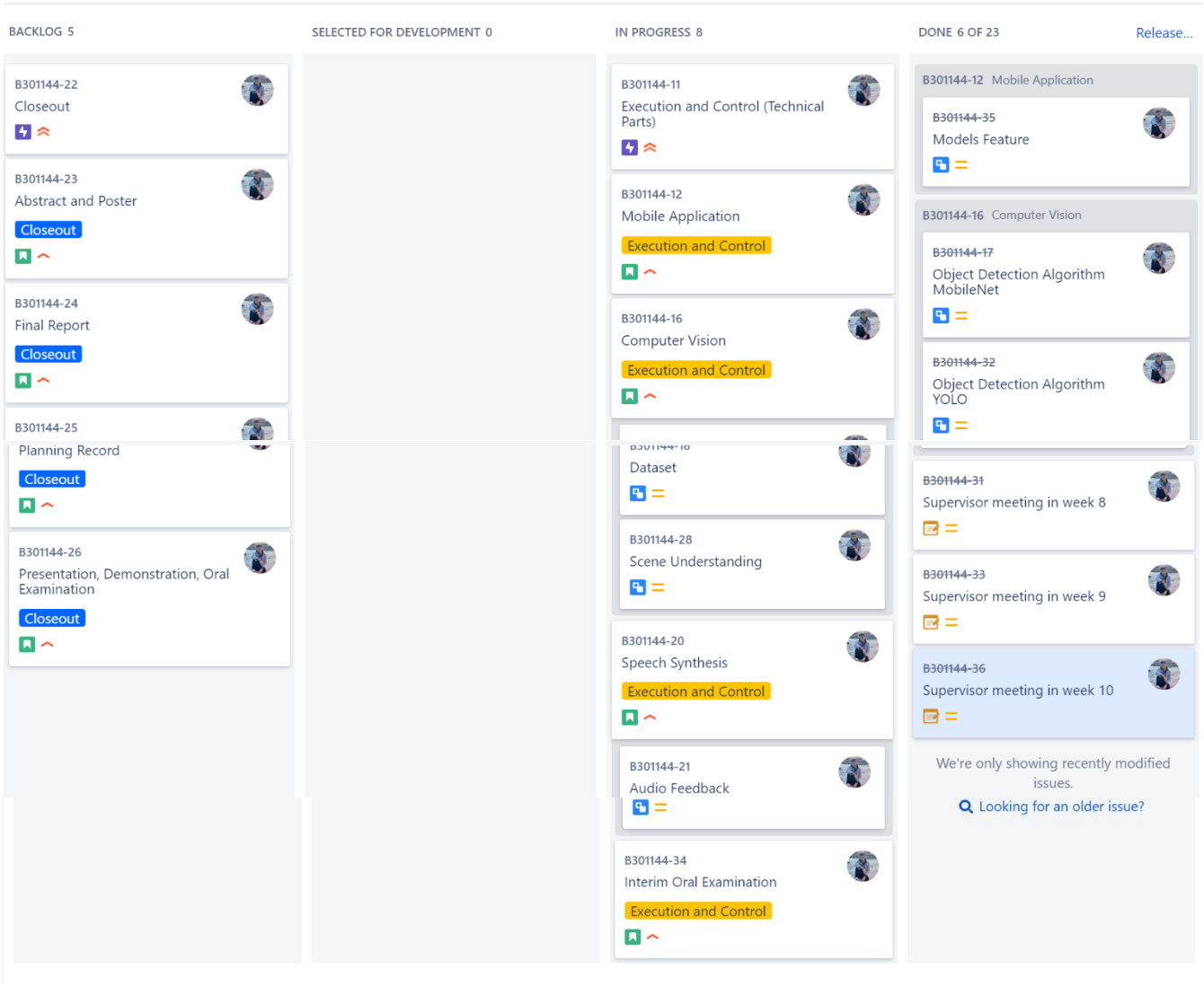

Figure 50 Kanban board before Interim Interview

Figure 51 presents all the issues captured on JIRA software. This figure is updated up to 19/04/2022 and it captures everything created on Jira, including Epics, Issues, Feedbacks, and Meetings.

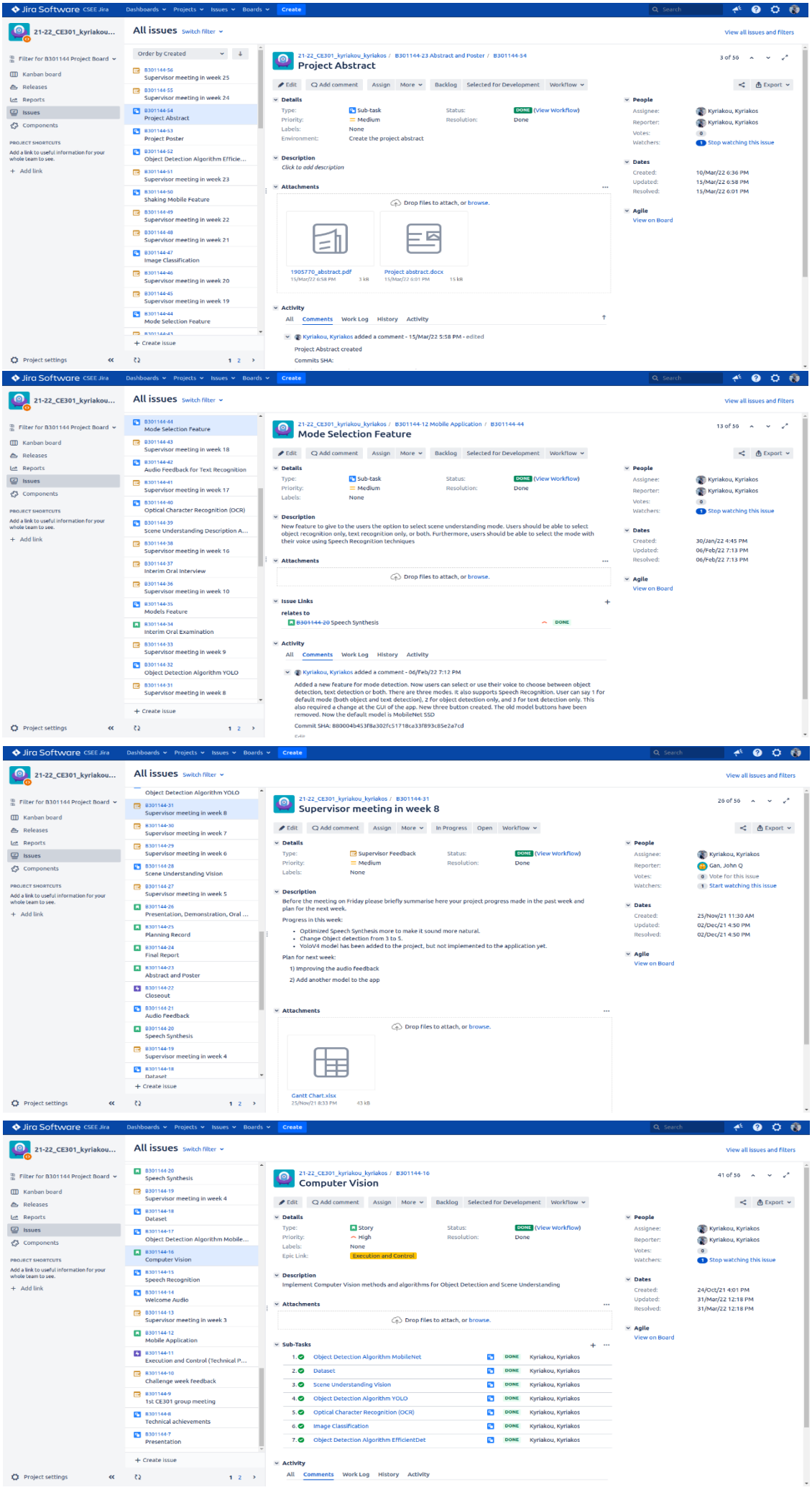

Figure 51 All issues captured on JIRA 19/04/2022

A cumulative flow diagram is captured on figure 52 (date 19/04/2022), displaying how stable the flow is and gives insight analytics to understand the project process.

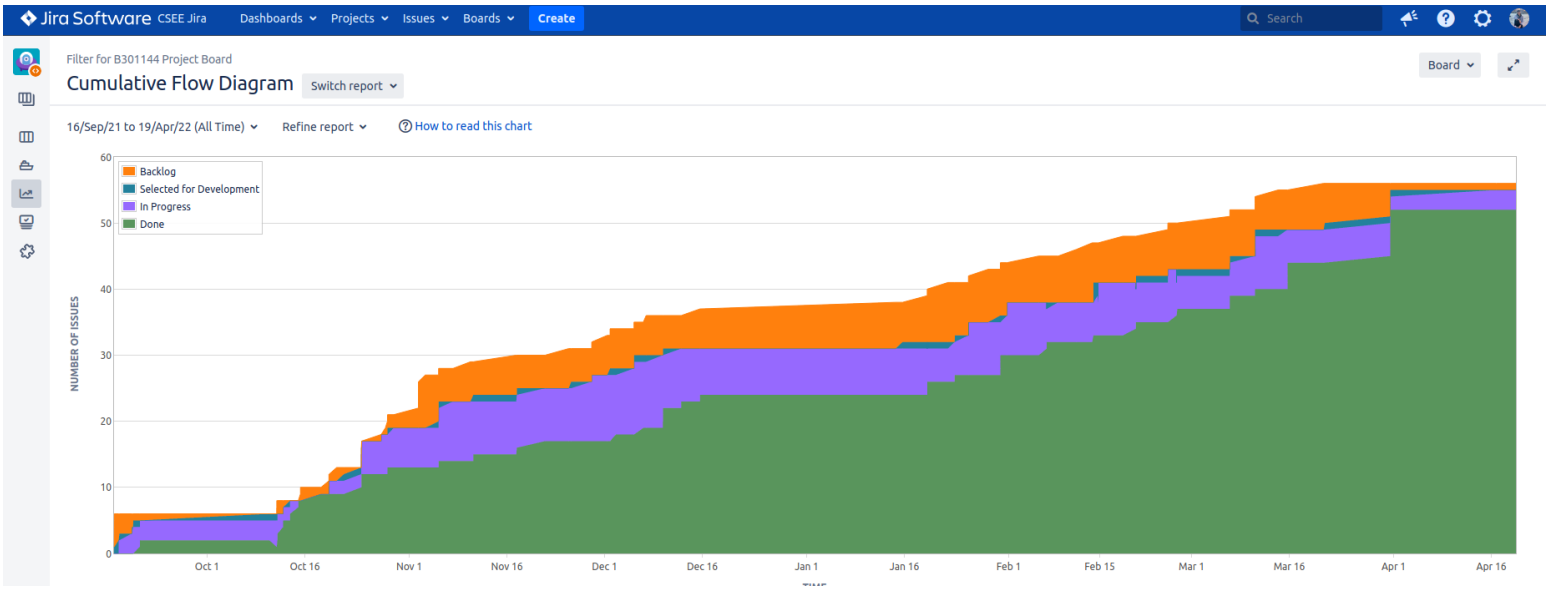

Figure 52 Cumulative flow diagram 19/04/2022

GitLab is another software tool used during the year for executing tasks and developing the BlindVision application. New branches were created for the deliverables required in each phase. After finalizing a deliverable, merging to the main branch was made. Image below shows the branches created. Summer work branch was used for the background reading, challenge week branch for the deliverables of challenge week, mobile application branches for the application development (before and after interim interview), and the closeout branch for the final report and final presentation.

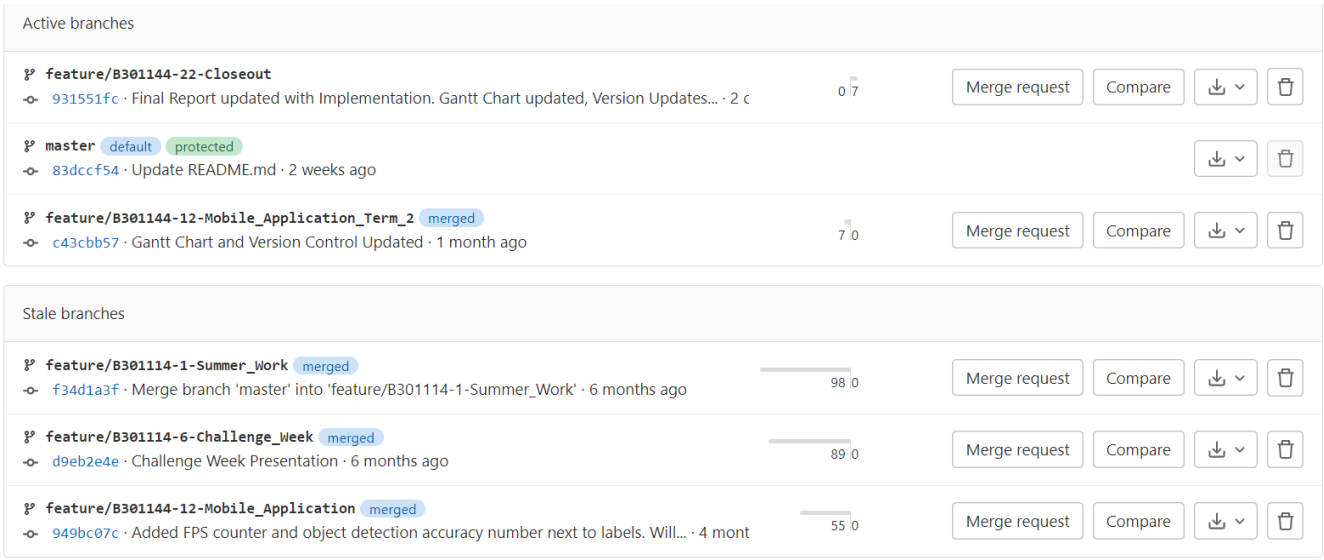

Figure 53 GitLab project branches

GitLab engagement over the year is captured on figure 54.

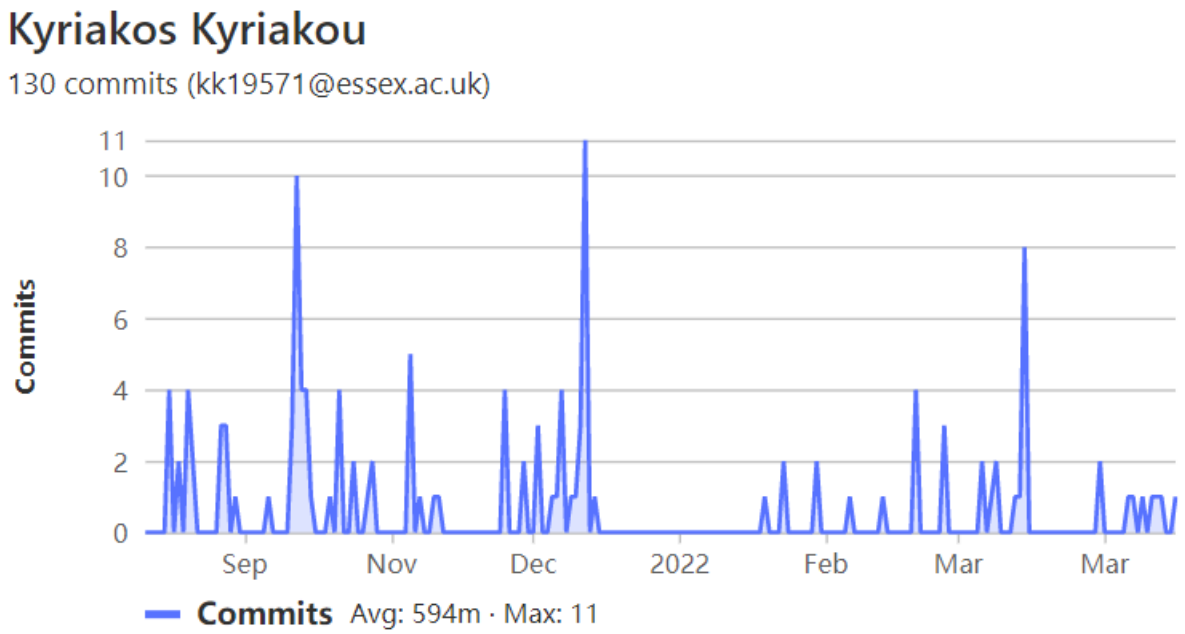

Figure 54 GitLab engagement

Project repository of GitLab shown on figure 55 (date: 19/04/2022)

Name	Last commit	Last update
..		
Abstract and Poster	Abstract booklet pdf	1 month ago
Challenge Week	Final Report updated with Implementation. Gantt Chart updated, Versi...	2 days ago
Final Report	Final Report updated with Implementation. Gantt Chart updated, Versi...	2 days ago
Interim Oral Interview	Minor Updates	4 months ago
Mobile Application	Added small code block to determine the build android version used	6 days ago
Summer Work	Gantt Chart (Updated)	6 months ago
.gitkeep	Add new directory	7 months ago
Gantt Chart.xlsx	Final Report updated with Implementation. Gantt Chart updated, Versi...	2 days ago
Version Updates.docx	Final Report updated with Implementation. Gantt Chart updated, Versi...	2 days ago

Figure 55 GitLab repository

7. CONCLUSIONS

BlindVision application was developed, which adopts multiple computer vision techniques such as object detection, OCR, and image classification. Combining these tools with speech synthesis techniques, this system can be used by people with impaired vision to get a scene understanding of their surroundings.

7.1. Technical Achievements

The application developed implements object detection, text recognition, and image labelling techniques to help people understand a scene. BlindVision aims to label scenes and generate in-depth text descriptions based on what have been seen using device's camera. Text-to-speech tools have been implemented to give audio feedback to the users. Different CNN object detection models have been tested and evaluated so the best one could be used as the main model in the app. The app can recognize different text, people, objects, animals, activities, and give audio explanation of the environment. The BlindVision system is capable of understanding users' commands by voice due to the implemented speech recognition feature. Finally, BlindVision has built-in shaking gesture that makes the interaction with the users hands-free.

7.2. Limitations and Future Work

The project has three main limitations. Firstly, and most importantly is the reliability of the application. The application must be seen by the users as an additional tool to help understand surroundings, and not as an 100% reliable application that can understand everything with infallible accuracy. Uncertainty in a lot of the recognitions exists so not everything should be taken as granted. Secondly, the app's recognition range is limited to only fixed datasets, something that reduces the usability of the app. The incapability of detecting certain objects can lead to a misleading scene understanding. Another limitation is the scene understanding templates implemented that provide a better scene understanding. Because the templates are added manually, the app lacks on giving in-depth descriptions in many scenarios. That having been said, the application has a lot of room for improvement.

New object detection models can be tested and evaluated, so a better model can be used that will provide more accurate detections and will be able to detect even more objects than the already implemented one. Moreover, paid Google's Cloud Vision API could replace the current implemented image labelling model and extend the possible recognized labels from 400+ to 10000+. Additional features could be introduced to make the user-system interaction even better. For instance, flash toggle feature is the next update coming to the application, making the invisible visible in the dark. Customer testing will be performed, and feedback will be taken into consideration for any further improvements. More audio templates could also be generated with more users using the application. The implementation of said features is recommended, as they can improve the BlindVision application and enhance the experience of the visually impaired users.

REFERENCES:

[1] Blindness and vision impairment (who.int)

[2] Assistive Mobile Application for the Blind, Ismail Sahak, Ong Huey Fang, Syuhada Abdul Rahman, Faculty of Computing, University Malaysia of Computer Science & Engineering, Malaysia, School of Information Technology, Monash University Malaysia, Malaysia ong.hueyfang@monash.edu

[3] "Artificial Vision for Blind Peoples using OCR Technology " , RAMKISHOR V, RAJESH L, International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), Volume 6, Issue 3, May - June 2017 , pp. 030-033 , ISSN 2278-6856

[4] PicToText: Text Recognition using Firebase Machine Learning Kit, Desmond Kim Seng Neoh, Eric Chun Meng Lock, Hong Yip Tang, Kem Ho Lew, Dong Ling Tong, Tun Tai Tan, Kwan Lee Tseu, 2021 2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS), 8-9 Sept. 2021, ISBN 978-1-6654-1726-6

[5] Background Reading from Challenge Week, October 2021, Kyriakos Kyriakou

[6] Framework Choice Criteria for Mobile Application Development, Proc. of the 2nd International Conference on Electrical, Communication and Computer Engineering (ICECCE) 12-13 June 2020, Istanbul, Turkey

[7] Native vs Cross-Platform Development: Pros & Cons Revealed (uptech.team)

[8] Blind - Sight: Object Detection with Voice Feedback A. Annapoorani, Nerosha Senthil Kumar, Dr. V. Vidhya, Volume 7, Issue 2, March-April-2021, ISSN (Online): 2395-566X

[9] Understanding Convolutional Neural Network, Jayanth Koushik Language Technologies Institute Carnegie Mellon University Pittsburgh, PA 15213

[10] Understanding of a Convolutional Neural Network, ICET2017, Antalya, Turkey, Saad ALBAWI , Tareq Abed MOHAMMED, Saad AI, 978-1-5386-1949-0/17/$31.00 ©2017 IEEE

[11] A review: Optical Character Recogntion Swati Tomar & Amit Kishore, M. Tech Scholar, Cyber Security, Site, Subharti University, Assistant professor, Cyber Security, Site, Subharti University, International journal of engineering sciences & research technology, ISSN: 2277-9655, April 2018, Impact Factor: 5.164, IC Value: 3.00, CODEN: IJESS7

[12] Optical Character Recognition Research at Google, Yasuhisa Fujii, 9-12 Oct. 2018, ISBN 978-1-5386-6309-7, ISSN 978-1-5386-6310-3, IEEE 13 Dec. 2018

[13] Mobile Application for The Visually Impaired by Darren Tan Yung Shen, Universiti Tunku Abdul Rahman, June 2020

[14] https://www.aipoly.com/

[15] https://taptapseeapp.com/

[16] https://developers.google.com/ml-kit/vision/text-recognition

[17] https://developers.google.com/ml-kit/vision/image-labeling

[18] CICERONE- A Real Time Object Detection for Visually Impaired People, Therese Yamuna Mahesh, Parvathy S, Shibin Thomas, Shilpa Rachel Thomas and Thomas Sebastian, IOP Conf. Series: Materials Science and Engineering 1085 (2021) 012006 doi:10.1088/1757-899X/1085/1/012006

APPENDIXES

Instructions for Installing and Running the Application

- Download OpenCV SDK from https://opencv.org/releases/
- From Android Studio navigation top bar:
  - Select New
  - Import Module
  - Select the path to the downloaded OpenCV SDK folder (you can rename it to whatever you want)
- In the build.gradle file (the Module one), find the dependencies section and add implementation project (path: 'OpenCV SDK folder name goes in here')
- Re-build project
- Select virtual or physical device for deployment
- Run project

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation project(path: ':sdk')
    implementation 'androidx.camera:camera-core:1.0.2'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
```

Figure 56 Part of dependencies section in Gradle script showing OpenCV SDK addition (folder named sdk)

Project Poster