Chapter 1: Introduction

1.1 Overview

This chapter will introduce the background and concepts regarding the project. That includes proposed work, expectations and features of the final product. Besides that, the methodology and software tools adopted will also be discussed here.

1.2 Background of the Project

Transign, an artificial intelligence-based real-time sign language to speech translation application, has the main features of detecting hand signs and then converting it them to speech.

This project aims to ease the difficulty of communication for deaf/mute people in everyday life as the skillset for understanding sign language is a skill rarely found among people within society in the current times. ASL (American Sign Language) will be the go-to sign language for this project due to its popularity. The project will go through a few phases to accomplish this objective. Firstly, custom data of hand signs are produced through a webcam. Following that, the dataset is cleaned and processed before it is fed to the model for training. Once a feasible model has been produced, the inference will go through a few confidence tests before it is passed as legible sign language and then converted to text for speech synthesizing. The resulting translation can also be translated to other spoken languages to further bridge the gap of communication between deaf/mute people and other ethnics around the world.

1.3 Key Concepts

At the base of this system is the hardware which will be the mobile handheld devices used daily by almost everyone known as smartphones. A device can be considered a smartphone when it has the processing power and memory to run applications, make phone calls and text messages, access to the internet, sound input and output, and lastly camera capabilities(*What is a smartphone? | Digital Unite*, no date).

With that said, the system will be using Android-based smartphones. Android is an operating system for smartphones developed by Google and is the operating system of more than 70% mobile handheld devices worldwide to date(*Mobile Operating System Market Share Worldwide | StatCounter Global Stats*, no date).

Moving along, a few keywords must be known that will be used throughout this proposal. The first one being **image processing.** The definition of image processing is to manipulate images by processing it through computers(Young *et al.*, 2004). Images can be broken down into a matrix of precise numbers that can be quantized and manipulated to do tasks such as image enhancement, restoration, and analysis(da Silva and Mendonça, 2005). The system will be mainly focused on the enhancement and analysis aspect of image processing to process then extract useful data from the video feedback given through the smartphone camera.

Besides that is **machine learning,** it is the core of what people of the current era knows as artificial intelligence(Shalev-Shwartz and Ben-David, 2014). It has the power to learn autonomously without any manipulation from a human being through the data fed into it to look for similarities or patterns through computing power and models made by human beings. It uses what it has learned to make decisions when provided input and output a desirable outcome(Decencière *et al.*, 2013).

Moving along is **deep learning**, a subset of machine learning algorithms mainly inspired by the function of a human brain called artificial neural networks(Socher, Bengio and Manning, 2012). Deep learning models achieve immaculate accuracy and speed, sometimes better than human-levels when it comes to classification tasks directly from images, text, or sound. These models train through enormous sets of labeled data and neural network architectures containing many layers similar to a human brain(Perez and Wang, 2017).

Last of all we have **real-time object recognition,** which has the aim of producing decisions on the spot when given live data on-site like video from a camera feed(Chadalawada, no date). It requires a robust deep learning model that can process information as quickly and as accurately as possible due to the time-sensitive nature of tasks object recognition is used for such as autopilot systems(Gavrila and Philomin, 1999).

1.4 Methodology

For this system, the Prototyping Model is the methodology the system's development will follow. This methodology is the most suitable for this project as the ideas behind this system are fairly new and never done before so requirements for it may change as functions are developed(Al-Husseini and Obaid, 2018).

Firstly, the system's requirements will be analyzed by doing more research on what deaf and mute people would like to see in an application that will mainly affect them or be used by them. Not only that, the frameworks and models practical usefulness will also be analyzed in this phase.

After that, once all the requirements have been detailed and documented, a feasibility study will be done to see whether the project is feasible to be developed. An example of this is checking what Android devices have adequate resources to run this system once it is developed.

Continuing on, a feasibility study will be conducted when the first design of the system is created. This design is considered the first prototype and once it is created, it will go through an evaluation to see what other requirements or enhancements should be done. This will repeat until a satisfying product is created hence the name prototyping. The first prototype may be functional and able to recognize sign gestures but does not have a friendly UI to be used. It will go through the second prototyping to redesign the UI with its sign gesture recognition functions and be reviewed again(Carr, 1998).

Lastly, once a satisfying prototype has been created, it will go through final testing and maintenance before becoming a finalized product. This is where all the test code and straggling bugs are cleaned up so that the final product used does not have any problems('SDLC - Software Prototype Model', no date).

1.5 Proposed Work

The proposed system is to create a mobile application in Android that will be able to recognize sign language gestures and translate it into speech. This application is mainly targeted to assist deaf and/or mute people with communicating with the general society when text or writing does not suffice.

As its nature states, this system will focus on Android handheld devices. This means that the application will only be able to run on devices that use the Android operating system. The device must have a camera which will not be a problem as the majority of Android devices nowadays come equipped with one.

The system will be designed with Kotlin, a language being pushed by the developers of Android, Google, as the new mainstay for Android application development(*Kotlin and Android | Android Developers*, no date). It will work in tandem with libraries such as OpenCV, TensorFlow, and

Keras to recognize objects within the live video feed of the camera and train the data into suitable models.

The system will not have any processes that run in the background and will only be an application that can be run on the main thread. This will comfort the user's privacy is maintained as the system does not have anything to do if it is not currently being used.

As far as cloud connectivity goes, the requirement for it will be decided as development goes during the prototyping. Whether the model will perform better as a live service on the cloud or run locally on the device will be analyzed and evaluated during the developing phase.

The system will have its model created by TensorFlow and Keras and then trained with data found on the web or data given to it through the OpenCV controlled camera video input from the Android device(Zebin, 2017). The system will learn to recognize the gestures through the model and link them to the correct translations.

There are a lot of basic American Sign Language datasets that can be found on Kaggle, an open-source dataset website, containing the datasets such as the alphabet and digit gestures which will become the foundation of the data used to train the model here(*Interpret Sign Language with Deep Learning*, no date). As for the image recognition techniques, TensorFlow has APIs with trained models to recognize objects and classify them into objects. For this system, once hands have been recognized then it can go deeper and look for what gestures are being performed to recognize what gestures are being made.

1.6 Summary

After identifying the aims and objectives for this proposed system, it is concluded that this system requires more research before implementation. All the other areas of research will be discussed in next chapters with proper justifications and examples.

Chapter 2: Literature Review

2.1 Overview

In this chapter, three areas of research will be discussed that includes image recognition, machine learning and object detection. This analysis was aimed to help in studying all the methodology and complexity in the implementation of the proposed system. Each of them will be supported by at least two existing system literature.

2.2 Image Recognition

The purpose of this study is to analyse all the existing image recognition projects to get some understanding and insights about their approach and methodology.

2.2.1 Face Recognition

Another important area of research in image recognition is face detection system. The threshold function for a system will extract the information from the human face. The first existing system in this field was by Fares Jalled in 2017 who build a face recognition system using EigenFace which discriminating image input into multiple classes for person identification. The researcher's proposed system structure starts from the image input to the pre-processing process of the image. In the image pre-processing stage, the image resolution, environment elimination, image rotation as well as illumination to the image were taken care of before the next step, feature extraction (Jalled, 2017). In the feature extraction, the system will retrieve all the distinctive elements from a person's face for classification purpose. The classification purpose will classify the person by their name and features on their face through the database. Another face recognition project considered for this area was from Priyanka Wagh and her colleague in 2017. This proposed system was aimed to use face recognition to help in taking attendance. This system achieves face recognition by setting up a static camera where the entire class is covered to periodically take a photo. The taken photos were converted to a grayscale image with histogram equalization to improve the contrast in the image (Wagh et al., 2015). For the student recognition in the classroom, Viola and Jones framework were used. Each recorded face was tested through EigenFace and Principal Component Analysis (PCA) for random value reduction as EigenFace often maximize variation so with PCA it helps the system to combine the variations based on a specific principle (Wagh et al., 2015).

2.2.2 Crop and Weeds Recognition

In the 1990s, a research was conducted in conjunction between related agricultural departments from the McGill University and Cornell University which was built on the rapid improvement of machine vision and image processing technology at the time to distinguish weeds from actual crops by leveraging the power of artificial neural networks (ANNs) (YANG, et al., 2000). The images were taken in bird's-eye view at random locations within the fields of the in campus farms of McGill University with slightly varied zooms but similar aspect ratios. For training and testing purposes, common weeds were put into one category and separated from the target crop which in this case were corn plants. The images would be processed into 8-bit color bitmaps and then further cropped into 100x100 pixels from the original image's resolution. This ensured that the crops and weeds were properly seen in each cropped section while optimizing it for processing within the ANN where PC memory was inadequate (YANG, et al., 2000). The images would then be fed into the ANN where the color index of a cluster of pixels would go through the processing elements (PEs) of the ANN and return 1 or 0s depending on whether a crop or weed was predicted for that cluster. The results from all cluster predictions would then be gathered and inferenced from a confidence of 0…1 to predict whether the plant seen within the cropped image is a crop or weed. Multiple configurations of PEs were tested for the recognition of crop and weeds which returned a noteworthy result. The ANN's prediction result of weeds accuracy rose from 40 to 50% to 70 to 80% when 200 PEs instead of 160 were incorporated to its hidden layer but the accuracy for the prediction of corn plants decreased when more PEs were added onto the ANN's hidden layer compared to the previous 200 PEs (YANG, et al., 2000).

2.2.3 Factors Taken

The feature extraction method used in the face recognition research paper will be used in this project as it is proven to work well with human skin and can easily find simpler limbs such as hands when it can detect and classify faces easily. As for the weeds and crops, the method of grid cell inferencing will be used in conjunction with the neural network architecture which will be talked about in the next section.

2.3 Deep Learning Neural Network

The proposed project uses a specific neural network architecture specifically known as convolutional neural network (CNN).

CNNs primarily focus on input made up of images, ensuring that the architecture is set up best suited for dealing with that type of data. The reason for this is a key difference between CNNs and artificial neural networks (ANNs). The neurons within the layers of CNNs are organized in three dimensions comprising of spatial attributes such as height, width and depth (O'Shea & Nash, 2015). The depth mentioned in this case does not refer to the outstanding number of layers in the ANN but the third dimension of an activation volume. Compared to usual ANNs, the neurons within any given layer in the CNN only connect to a small region of the layer before it.

CNNs are made up of three forms of layers. These layers are known as convolutional layers, pooling layers, and fully-connected layers. A CNN architecture is formed when these layers are stacked together (O'Shea & Nash, 2015).

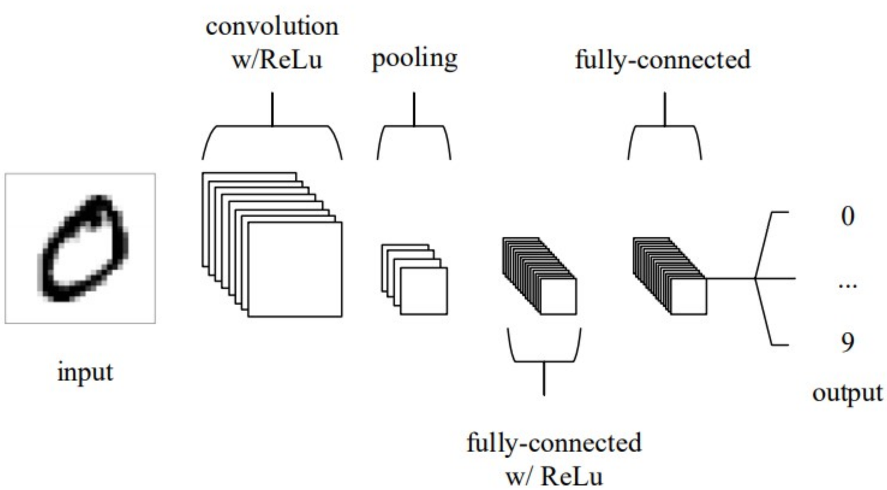


Figure 1 Simple CNN architecture (O'Shea & Nash, 2015)

Seen in figure 1 above is a simple CNN architecture made of only 5 layers for MNIST classification. The input layer contains the pixel values of the image fed into the CNN. Following that, the convolutional layer produces the output of neurons that are joined to local regions of the input through calculating the scalar product between their weights and the region joined to the input volume. The rectified linear unit (ReLu) applies an 'elementwise' activation to the output produced by the previous layer. The pooling layer has the role of downsampling along the input's spatial dimensionality, lowering the number of parameters within that activation. Lastly, the fully-connected layers perform similar roles as normal ANNs to produce class scores for classification (O'Shea & Nash, 2015).

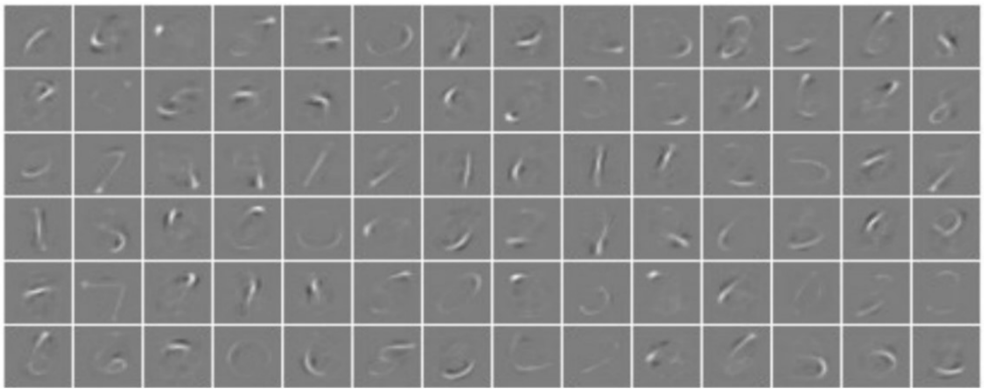


Figure 2 Activations from convolutional layer (O'Shea & Nash, 2015)

As seen in figure 2 above, the network picks up on unique characteristics of specific numeric digits from MNIST's database of handwritten digits.

2.4 Object detection

For object detection, there are a few algorithms that must be discussed and compared that are based on the CNN architecture mentioned in the previous subtopic. These algorithms will be discussed in the subtopic below.

2.4.1 Faster R-CNN

The R-CNN technique trains CNN end-to-end to do classification for proposal regions into object background or categories. R-CNN mostly act as a classifier, and the object bounds cannot not be predicted. The performance of the region proposal module defines the accuracy. Pierre Sermanet and his team proposed a paper in 2013 under the title of "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks" which shows the ways of using deep networks for predicting object bounding boxes (Sermanet et al., 2013). In the OverFeat method, a fully-connected layer is trained to predict the coordinates of the box for the localisation task that consider a single object. Then, the fully-connected layer is turned into a convolutional layer for multiple class object detection (Sermanet et al., 2013). Figure 3 shows the architecture of Faster R-CNN which is a single, unified network for object detection.

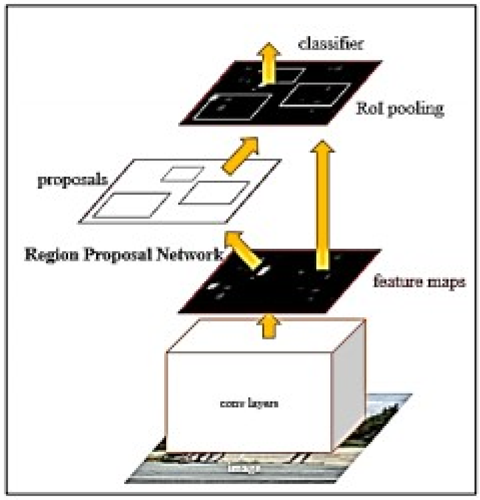


Figure 3 architecture of Faster R-CNN (Sermanet et al., 2013)

Fast R-CNN implement end-to-end detector training on shared convolutional features and display decent accuracy and speed. On the other hand, R-CNN fails to do real time detection because of its two step architecture.

2.4.2 YOLO

YOLO stands for "You only look once". According to Redmon (2016), it is an object detection algorithm that runs quicker than R-CNN because of its simpler architecture. Classification and bounding box regression will be done at the same time. Figure 4 shows how YOLO detection system works.

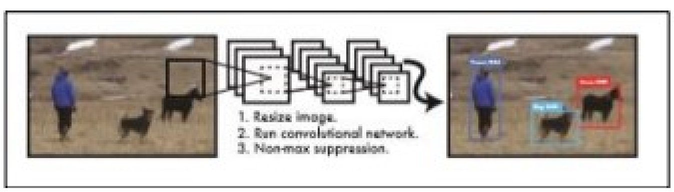


Figure 4 YOLO Detection

A single convolutional network predicts several bounding boxes and class probabilities for those boxes simultaneously. YOLO trains on full images and the detection performance will be optimised directly. This ensures YOLO to run extremely fast, hence producing real-time predictions.

2.4.3 SSD

Lastly, we have SSD which stands for "Single shot detection". SSD is another algorithm with similar performance metrics to YOLO where SSD wins in overall accuracy and speed but loses out in number of true positives and recall values (Morera, et al., 2020).

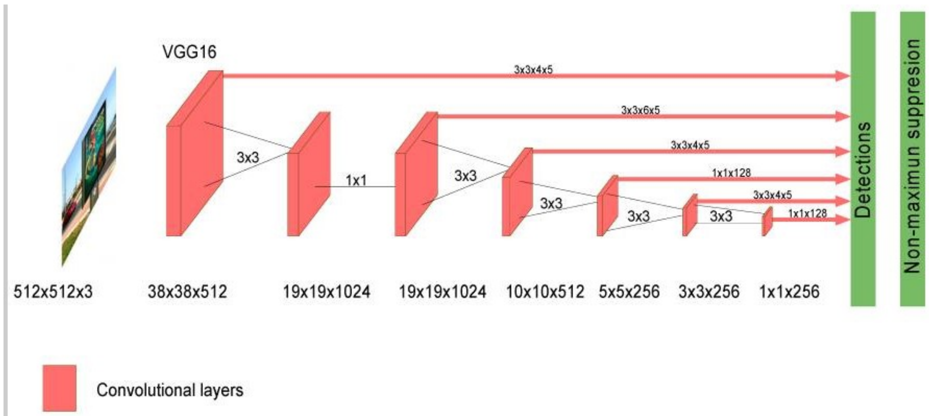


Figure 5 SSD Algorithm (Morera, et al., 2020)

As seen in figure 5 above, a CNN based on SSD defines a domain of anchor boxes with varying sizes and aspect ratios at multiple locations of an image. The detection box prediction is then done based on these references. The SSD algorithm does not resample features for bounding box hypotheses unlike R-CNN and is compatible with real-time applications.

2.5 Summary

In conclusion, the CNN architecture is chosen due to its performance and compatibility with processing image data. Moving on, the CNN-based SSD algorithm will be used due to the cost-performance ratio in terms of computing resources, making it suitable for the purpose of the project without sacrificing the accuracy of the results produced. As the CNN-based SSD model only needs to check the data once and then process it with different layers in different grid setups, accuracy is still maintained on objects that do not require immense detail while being efficient on processing power which is suitable for this project.

Chapter 3: Requirements Analysis

3.1 Overview

After completing all the research on relevant literatures, main functionalities of the system will be discussed in this chapter though a conceptual model. After discussing the conceptual model, commentary on chosen methods will be done in terms of functional and non-functional requirements to further the research.

3.2 Functional Requirements

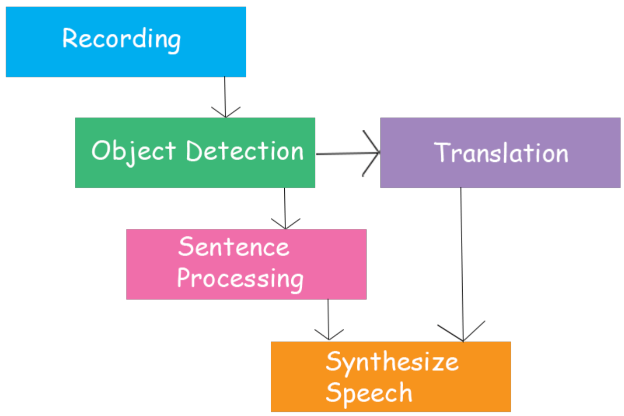


Figure 6 Transign functional requirements

3.2.1 Recording

As seen in figure 6 above, there are 5 functional requirements within this project. The first requirement is the recording function. The recording function is essential as it is the method to gather image data for detection in real time. This function will leverage the OpenCV application programming interface (API) to gather data through a recording hardware of the user's choice. The data will be gathered at a rate of 30 frames-per-second (FPS) and 800 by 600 pixel resolution to satisfy a few conditions of the application. These parameters ensure that none of the peripheral background or surroundings are recorded as they are not relevant to the required input, ensuring accuracy with the user as the main focus in the center of the image data gathered. Not only that, less memory resources are required without impacting prediction results when 30FPS is used instead of 60FPS. Lastly, the resolution specified satisfies the aspect ratio of data that will be used to train the model, ensuring that the image is not distorted when undergoing pre-processing phase before being sent into the model.

3.2.2 Object Detection

Following that is the object detection function. Once the image data has been recorded, it will then be processed to fit the parameters of the model. As mentioned in the previous chapter, the object detection will use the CNN architecture with the SSD algorithm. The image data will be divided into grid cells where each cell is responsible for inferencing classes and boxes within its boundary. If an object overlaps multiple regions at once, pre-defined anchor boxes will kick in to supplement the prediction of the objects within the image (Developers, n.d.).

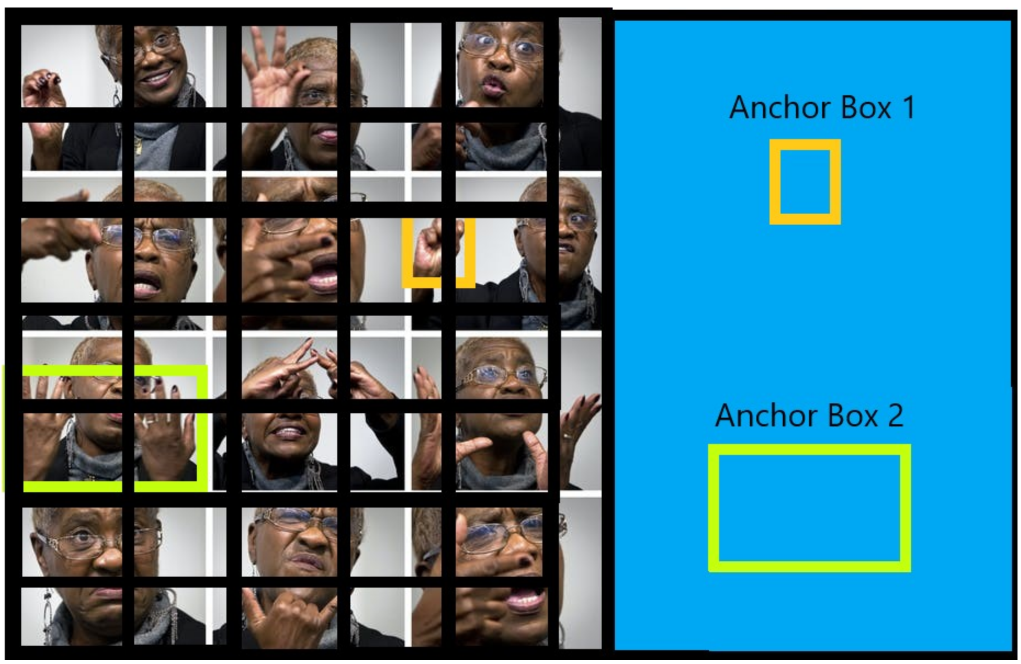


Figure 7 Grid cells and anchor boxes

3.2.3 Sentence Processing

Next up is the functional requirement called sentence processing. In all sign languages, hand signs usually convey meaning but don’t join up to complete sentences. For example, sentence helpers such as “is” or “are” aren’t present in the ASL dictionary. With that, converted sentences from hand signs will not sound legible in a conversational environment. Sentences converted from the hand signs have to go through sentence processing so that it will sound normal when used in a conversation context.

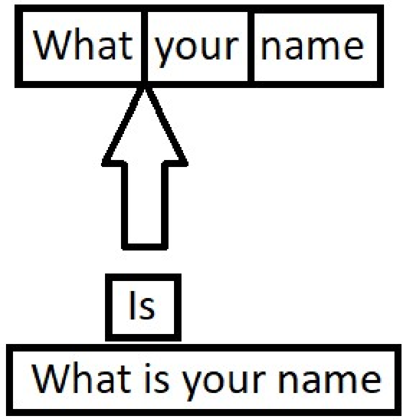


Figure 8 Sentence processing visualization

3.2.4 Translation

Another requirement is the translation function. The Transign application allows the user proficient in only one sign language to communicate with foreign people within or outside of their local country. This function will use the Google Translate API to facilitate its translation as it is one of the most readily available and competent translation APIs in the market.

3.2.5 Synthesize Speech

Lastly, we have the function to synthesize speech. Once all the translation or sentence processing has been completed, the output from that will be synthesized into a voice that is relevant to the chosen language. For example, an English speaking voice may not be able to pronounce Japanese words properly and thus will require some additional work to produce legible speech. This will require additional Windows internal speech packs to be installed.

3.3 Non-functional Requirements

3.3.1 Security

Security is a factor of importance in any application. Hence, this application does not store any of the user's image data locally or virtually and will be dumped once users have finished their usage purposes. Even though Internet access is required for some functions to work, no data is sent out of the application's memory scope.

3.3.2 Performance

Transign will be able to perform in any device with decent processing power as long as it has hardware to record image data and output sound. As no file writing or uploading tasks are done, processing power requirements are kept to a minimum and the only load observed will be from reading and processing image data from the recording hardware.

3.3.3 Usability

Usability is another factor to take note of within this project. There will be no login requirements which means that users can directly dive into using the system. Not only that, the Graphical User Interface(GUI) is not complex and does not show any technical jargon that would not be understandable by the layman user. The application is also usable out of the box for its basic functions and does not require additional external modules.

Chapter 4: Design

4.1 Overview

In this chapter, system design of this proposed system will be discussed. Furthermore, this chapter will also cover the design models such as Unified Modelling Language (UML) diagrams such as use case diagram, class diagram, sequence diagram and storyboard of the application.

4.2 Functional Design

Table 1 Use case descriptions

Use case	**Pre-conditions**	**Description**	**Flow of events**
Open Camera	GUI Initialized	Start the camera stream with OpenCV	1. The GUI loads 2. The user clicks on the start button 3. the camera frame begins streaming
Text-to-Speech	Text inferenced	Takes the results outputted from the model and converts it to speech	1. The model outputs its predictions 2. The predictions are displayed on screen 3. The predictions concurrently get synthesized into speech and played.
Detect Sign Language	Camera stream started	Takes the camera stream data and sends it for predictions	1. User begins doing hand signs 2. The hand sign frames are taken and sent to the model for detection 3. The model begins predicting
Translation	Internet access, User has it enabled	Runs translations on the predictions to other languages.	1. The user enables it on the GUI 2. The detected hand signs are converted to English and then sent for translation 3. Google Translate API returns translated results.
Internet Checking	GUI Initialized	Checks user's device for internet connection	1. The GUI loads 2. The user clicks on the Internet check button 3. A message box pops up to lock the UI 4. A ping is sent to Google's server for response.

Looking back at the previous chapter, there were in total 5 functional requirements. For the recording function, the Open Camera use case is linked to this as it will be how the image data is recorded. As for object detection, the Detect Sign Language use case is where the bulk of the work will be done for this function as it is the use case where the model will be situated in doing predictions. Following that, Text-to-Speech will cover the sentence processing and synthesize speech requirement. Text-to-Speech is where all the adjustments to the predictions will be done through an English dictionary to generate a legible speaking sentence. After that, the speech will be synthesized and then played in real time. Lastly, the translation requirement is covered by the translation and internet checking use case. Internet needs to be confirmed before translation can run and this is done by the internet checking use case. Following that, the translation can be enabled and will be done with the help of the Google Translate API. The use case diagram for this can be found in the appendix.

4.3 Interaction Design

In the interaction design there will be two figures to present the flow of the system's function. This section will describe how the system's internal functions will interact with each other within the system.

4.3.1 Detection Interaction

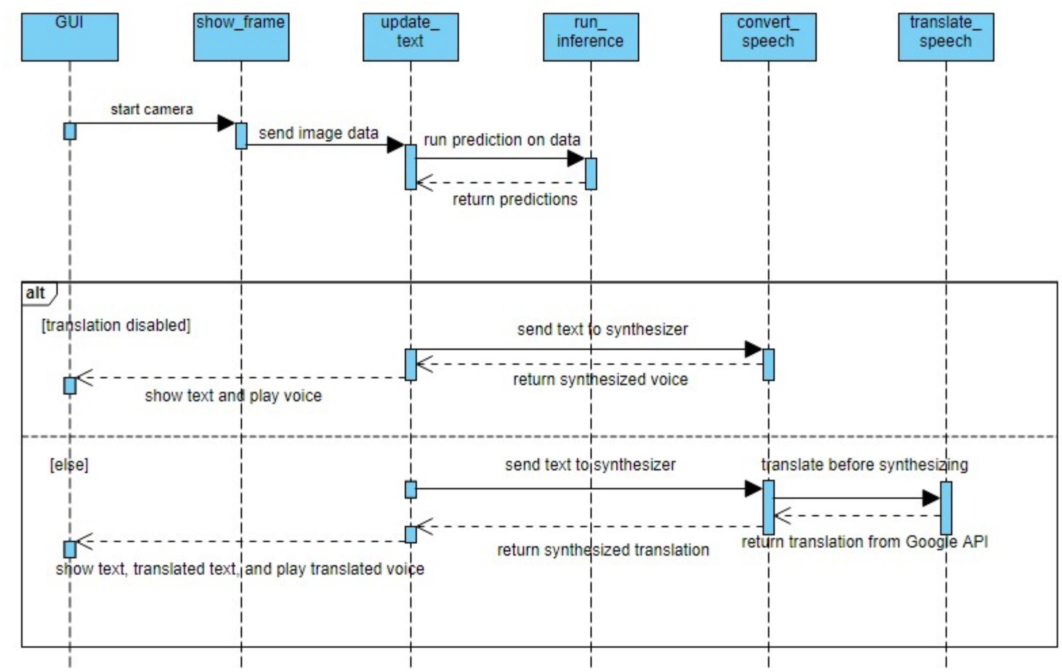


Figure 9 Detection Sequence Diagram

In figure 9, the interaction will begin at the GUI message. Once the user has a clicked the button to start the camera, the show_frame message will start up and begin sending image data to the update_text message. After that, the update_text message will call the run_inference message to begin predictions on the image data and wait for its predictions to be sent back. If the user did not enable translation, the update_text message will send the predictions to the convert_speech message where the synthesized speech will be directly sent back to the update_text message and back to the GUI message to be displayed and played. Otherwise, the predictions will be sent to the translate_speech message first before being synthesized into its destination language's suitable voice.

4.3.2 Translation option interaction

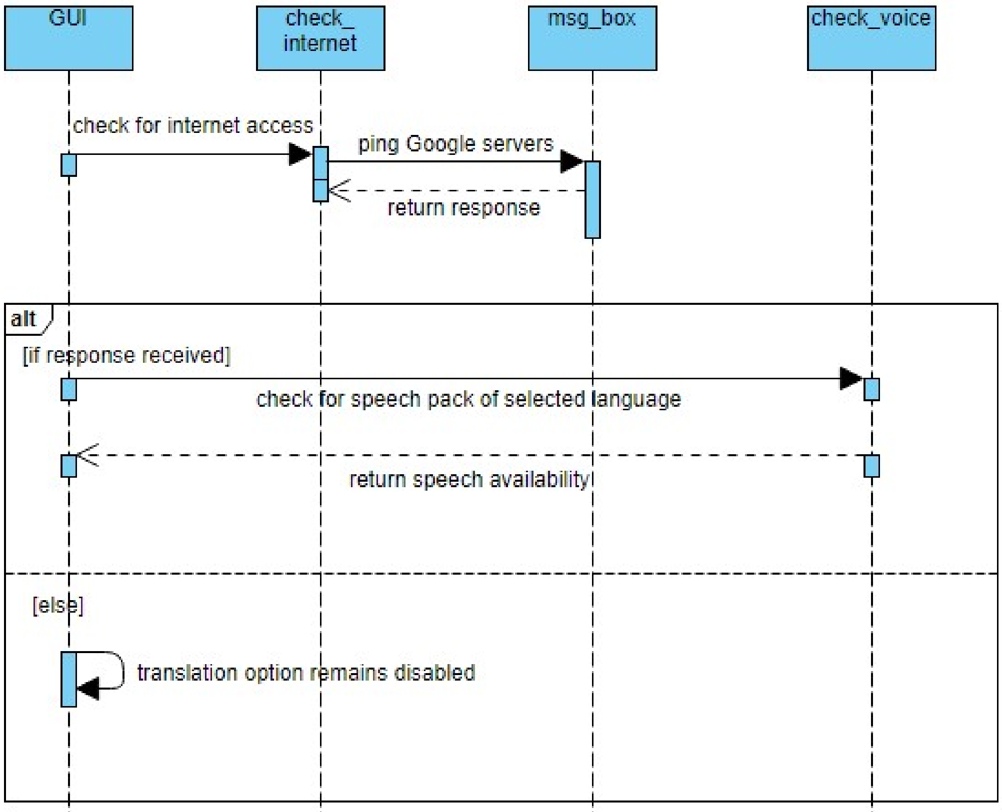


Figure 10 Translate option sequence diagram

In the translation option's interaction, the flow will once again begin at the GUI message and moves on to the check_internet message as the user has to confirm that they have internet access first before they can enable translation. A dialog box will pop-up to let the user know that internet access is being verified while a quick ping to Google's server is being done. This will all be done by the msg_box message. If a response to the server has been received, the check_voice message will then check if the language selected has a speech pack installed locally and display the results on the GUI. Otherwise, the translation option remains disabled on the GUI.

4.4 Graphical Design

The application will have a simple interface that can be easily understood within a few minutes of first opening the application. It will consist of a few text boxes to display translated and detected text, a frame to show the user as their hand signs are being detected, and lastly a few buttons to initialize the features of the application. A simple mock design can be seen in the figure below.

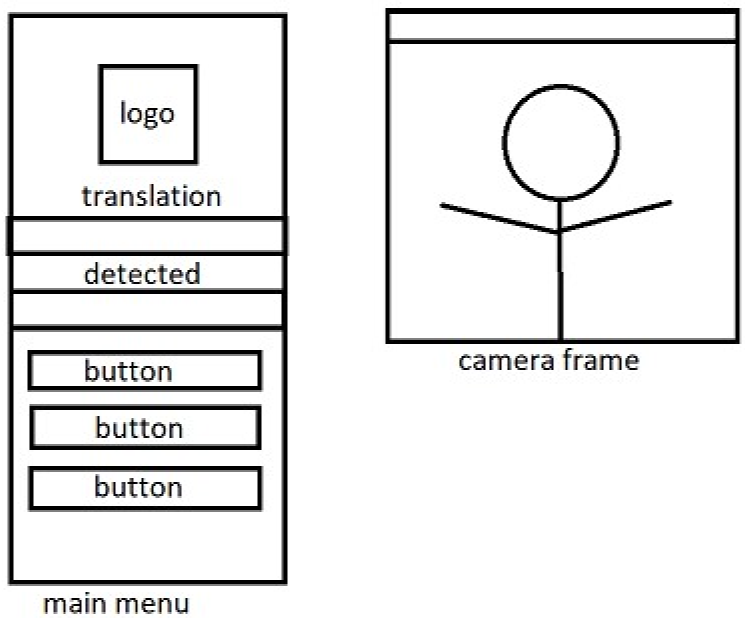


Figure 11 Mock design

4.5 Summary

After designing all the UML diagrams, a clearer picture can be seen to develop the proposed system. In next chapter, the implementation of the system will be discussed.

Chapter 5: Implementation

5.1 Overview

In this chapter, the implementations that were implemented in this project will be discussed which covers the methods and techniques used for the implementation and also the justification on the main functionalities along with code snippets.

5.2 Interpretation of datasets

5.2.1 Initial research and experimentation on datasets

In the initial research on datasets, a lot of data was found for sign languages in ASL on public sources such as Kaggle. An example of the data can be seen in figure 11 below.

label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel:
3	107	118	127	134	139	143	146	150	153	156	158	160	163	165	
6	155	157	156	156	156	157	156	158	158	157	158	156	154	154	
2	187	188	188	187	187	186	187	188	187	186	185	185	185	184	
2	211	211	212	212	211	210	211	210	210	211	209	207	208	207	
13	164	167	170	172	176	179	180	184	185	186	188	189	189	190	
16	161	168	172	173	178	184	189	193	196	202	206	208	212	214	
8	134	134	135	135	136	137	137	138	138	138	139	138	138	139	
22	114	42	74	99	104	109	117	127	142	152	155	162	165	168	
3	169	174	176	180	183	185	187	188	190	191	191	191	191	190	
3	189	189	189	190	190	191	190	190	190	189	188	187	187	186	
18	133	135	141	146	150	155	158	159	163	165	166	167	169	171	
10	0	25	38	40	41	46	50	56	69	81	91	96	99	104	
16	87	91	99	116	132	142	147	153	160	164	167	170	174	176	
22	80	98	121	39	53	94	100	107	110	128	144	152	162	165	
20	127	127	128	130	132	133	133	133	135	136	136	135	135	134	
16	86	87	89	93	104	114	122	131	137	140	142	146	151	155	
17	118	120	128	135	139	145	149	150	153	156	159	161	164	165	
13	223	225	226	227	228	229	230	230	230	231	232	231	231	232	
13	189	193	195	197	201	206	208	213	215	218	221	221	223	223	
19	173	174	176	177	176	177	177	177	177	177	175	175	175	175	
18	149	150	150	150	150	151	151	151	151	151	150	149	148	147	
21	131	135	139	143	145	146	149	152	153	156	157	158	159	162	
16	90	98	105	110	118	124	126	127	131	136	139	142	145	146	
23	85	90	97	103	119	130	138	147	152	157	161	164	163	167	
3	131	138	144	146	148	150	152	153	154	156	156	156	156	156	
23	153	157	163	166	170	172	175	177	178	179	181	182	183	184	
24	148	153	156	157	158	158	158	158	160	160	160	159	158	159	
18	138	141	143	147	149	150	150	151	154	154	155	155	157	159	
22	72	80	87	92	96	101	105	109	112	116	118	120	122	123	
1	146	149	150	150	151	152	152	153	153	154	153	153	154	154	
1	134	136	139	140	142	144	146	146	146	146	147	146	147	148	
12	148	151	157	164	170	173	175	179	182	186	190	193	195	197	

Figure 12 Example data

These were all data of alphabet hand signs and initially used to train a model with good results before encountering a major problem. After further research on existing data on public sources, there were no datasets found for complete words within ASL. Long consideration was made at this point of developing the application due to the imbalance in size of creating custom data compared to using existing data. In the end, creating custom data instead of using existing datasets was opted for due to the importance of complete words.

5.2.2 Creating custom data

```
for label in labels:
    !mkdir {"Tensorflow\workspace\images\collectedimages\\"+label}
    cap = cv2.VideoCapture(0)
    clear_output(wait=True)
    time.sleep(5)
    print('Collecting images for {}'.format(label))

    for imgnum in range(number_imgs):
        ret, frame = cap.read()
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        print('picture {} taken'.format(imgnum))
        cv2.imwrite(imgname,frame)
        cv2.imshow('frame',frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q') :
            break
    cap.release()
```

Figure 13 Code snippet to gather data

Once the decision was made, custom data was gathered using a webcam on a laptop through the code snippet above. The code snippet mainly used OpenCV's Python API to load up the webcam, capture the images and written to indexed folders with categorized names through the built-in OS library.

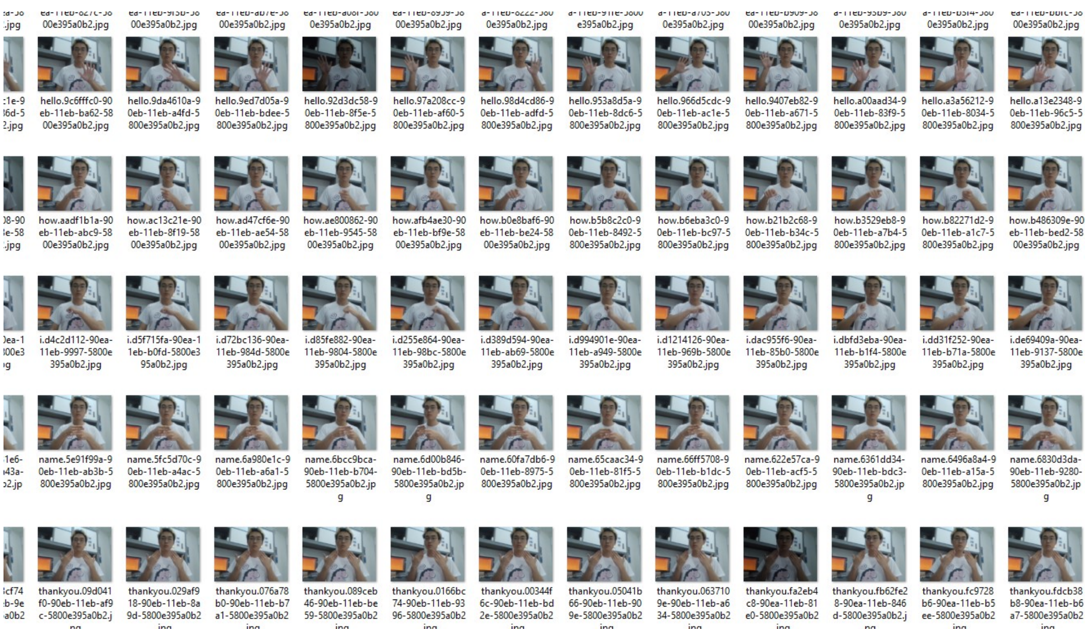


Figure 14 Custom dataset

In figure 13, the custom data can be seen gathered and contains around 9 classes with around 45 images gathered to represent each class, totaling up to slightly above 400 images.

5.2.3 Processing and splitting of dataset

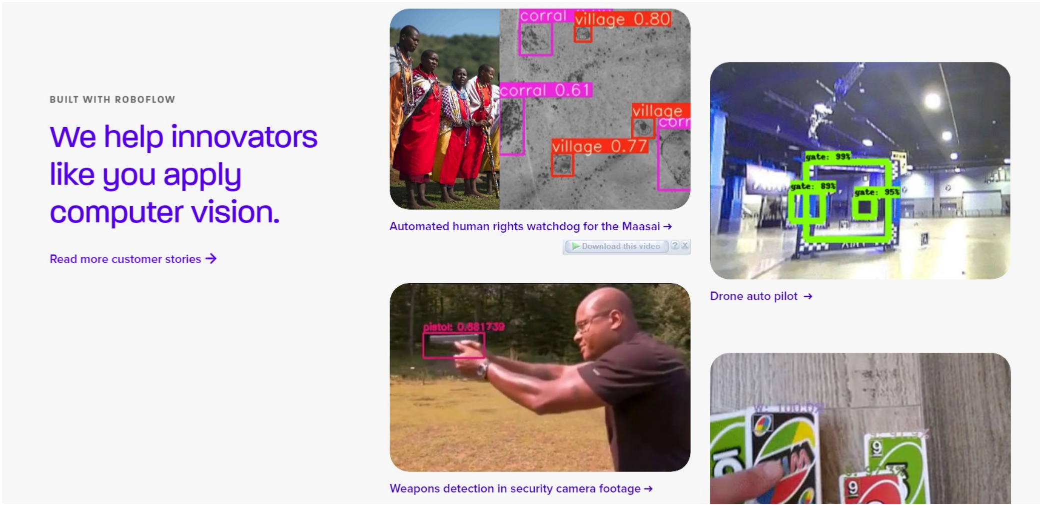


Figure 15 Robowflow home page

For the processing and splitting of the dataset, a third-party helper site was used to facilitate the processing of the dataset. The site is called Roboflow and contains many helpful tools such as annotating, labelling, organizing, processing, etc. for building computer vision projects.

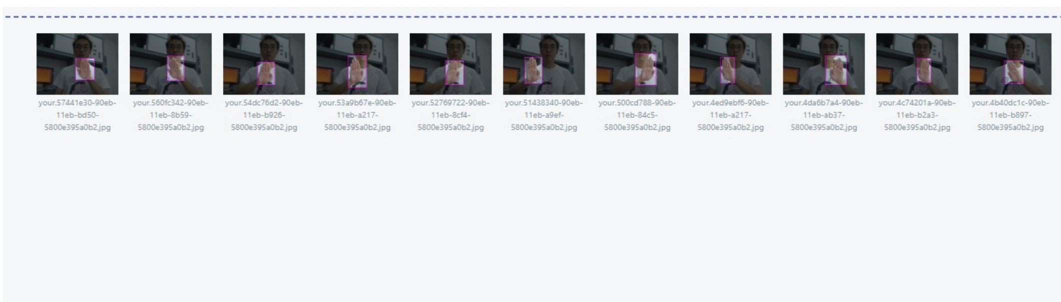


Figure 16 Data organisation in Roboflow

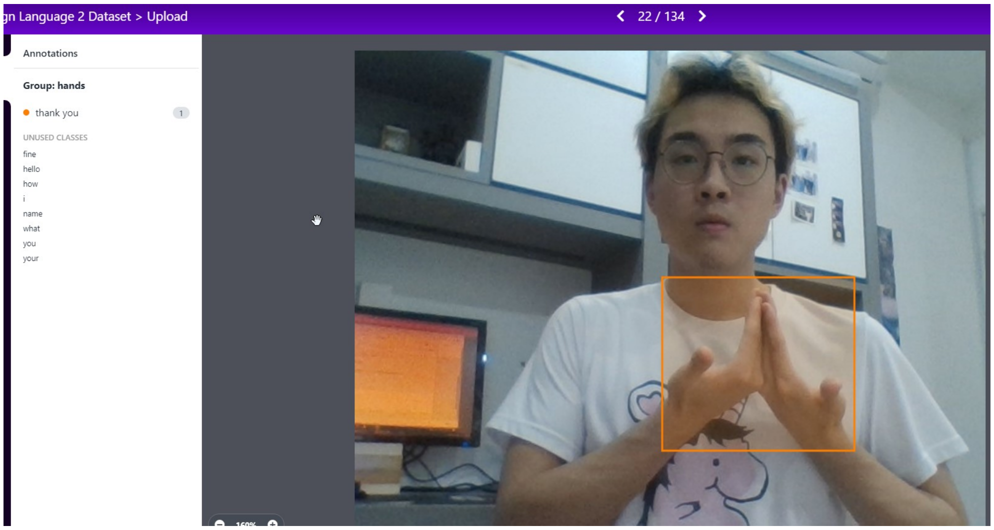


Figure 17 Image annotation tools in Roboflow

As seen in figure 15 and 16 above, Roboflow provides simple and easy to use tools for preparing datasets before moving on to the processing of the data which will be touched on more next.

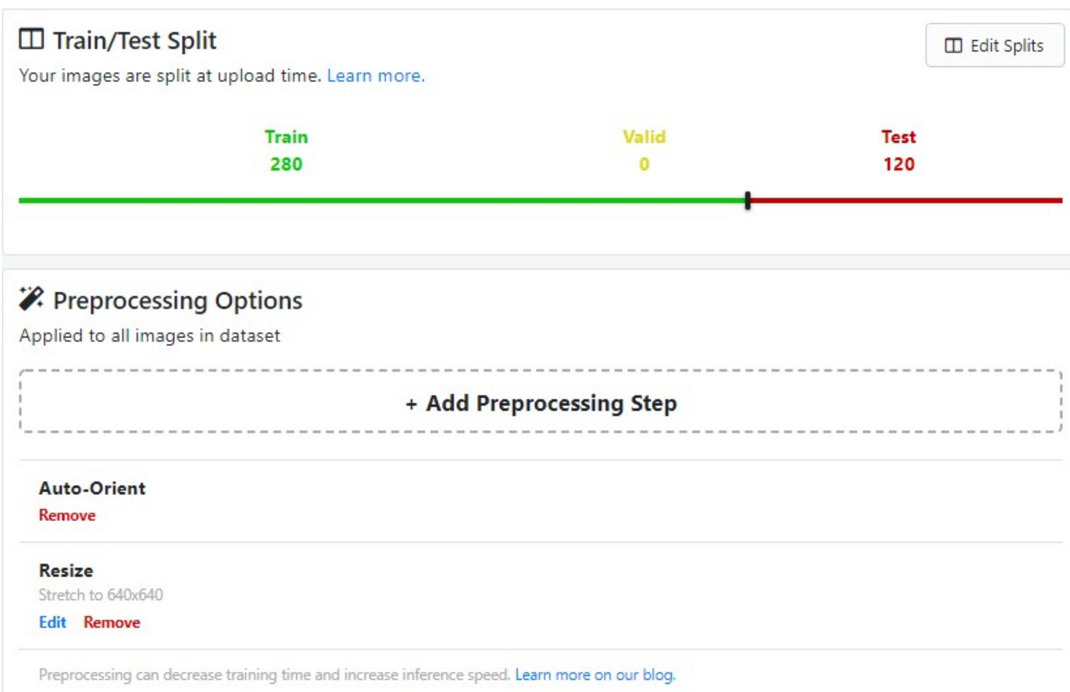


Figure 18 Pre-processing and data splitting

Roboflow also provided the ease of splitting up your dataset and pre-processing steps. In figure 17, resizing and cropping options were used to get an aspect ratio suitable to be used for the training of the model later on. The ratio used in splitting the data was a 7:3 ratio where 70% would be used for training the model while 30% would be used for evaluation during the training. Roboflow also provides the tools to do augmentations on images but those provided sub-optimal results which would be talked about later.

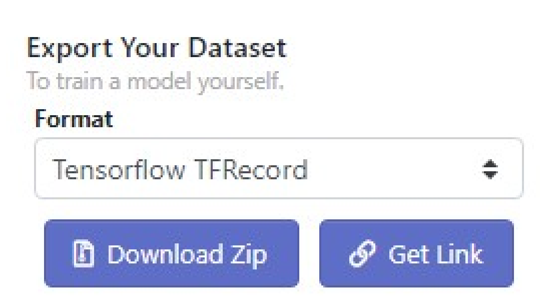


Figure 19 Exporting processed data

Once satisfied with the processing and splitting of the data, Roboflow also helps with exporting the dataset in your preferred format. Since the model training will be done through the Tensorflow API, the TFRecord format was chosen as the export format. As seen in figure 18 above, the dataset can also be downloaded locally to the device or through a link that provides a curl command with a unique token to the dataset that can be used to do training on the cloud.

5.3 Training the model

Model training will be done through Tensorflow, specifically the Tensorflow Object Detection API. There are a variety of models within the Tensorflow's Object Detection model zoo. As mentioned before in the literature review and design chapters, this application will use the "SSD MobileNet V2" which is built following the CNN architecture with SSD algorithm at its base. The model training will also be done on the cloud through Google's Colab as it provides top end computing power for free in a limited time period which will suffice for this project's training purposes.

5.3.1 Importing the required libraries and files

```
[ ]  # Install the Object Detection API
     %%bash
     cd models/research/
     protoc object_detection/protos/*.proto --python_out=.
     cp object_detection/packages/tf2/setup.py .
     python -m pip install .
```

Figure 20 Importing Tensorflow libraries

To train the model, the Tensorflow model repository has to first be downloaded from their github website. After downloading, the protos files have to be compiled with the Protocol Buffer library and then install all the relevant libraries with the setup file inside the respository as seen in figure 19.

```
#Downloading data from Roboflow
%cd /content
!curl -L "https://app.roboflow.com/ds/[redacted] > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

Figure 21 Importing data from Roboflow

The data that was processed and split in Roboflow now needs to be imported into the cloud and is done as seen in figure 20. As the token is unique to every user, it has been hidden here so that data is not leaked.

5.3.2 Setting up training parameters and configurations

```
    'mobilenet-v2-fpn': {
        'model_name' : 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'base_pipeline_file' : 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint' : 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
        'batch_size' : 4
    }

}

chosen_model = 'mobilenet-v2-fpn'

num_steps = 10000 #The more steps, the longer the training. Increase if your loss function is still decreasing and validation metrics are increasing.
num_eval_steps = 500 #Perform evaluation after so many steps

model_name = MODELS_CONFIG[chosen_model]['model_name']
pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']
batch_size = MODELS_CONFIG[chosen_model]['batch_size']
```

Figure 22 Parameters

In figure 21, the variables are setup with the appropriate names and values to make inputting easier later on as the names of the model files and checkpoints are long.

```python
with open('pipeline_file.config', 'w') as f:

    # fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)
    s = re.sub('max_number_of_boxes: [0-9]+','max_number_of_boxes: {}'.format(1),s)
    s = re.sub('max_detections_per_class: [0-9]+','max_detections_per_class: {}'.format(1),s)
    s = re.sub('max_total_detections: [0-9]+','max_total_detections: {}'.format(1),s)

    # tfrecord files train and test.
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")', 'input_path: "{}"'.format(test_record_fname), s)

    # label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Set training batch_size.
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes.
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)

    #fine-tune checkpoint type
    s = re.sub(
        'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)

    f.write(s)
```

Figure 23 Setting up configurations

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 9
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: [1.0, 2.0, 0.5]
        scales_per_octave: 2
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
```

Figure 24 Snippet of the final configuration

Next up is editing the configurations for the model training. The base model comes with a template configuration file but it needs to be changed so that it is suitable for this application's purposes. In figure 22, there is an example of some of the parameters being changed to suit this project's purposes.

5.3.3 Beginning Model Training

```
!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1 \
    --num_eval_steps={num_eval_steps}
```

Figure 25 Model training code snippet

```
INFO:tensorflow:Step 6600 per-step time 0.168s loss=0.244
I0330 00:41:41.252398 140095675557760 model_lib_v2.py:682] Step 6600 per-step time 0.168s loss=0.244
INFO:tensorflow:Step 6700 per-step time 0.154s loss=0.232
I0330 00:41:54.915445 140095675557760 model_lib_v2.py:682] Step 6700 per-step time 0.154s loss=0.232
INFO:tensorflow:Step 6800 per-step time 0.164s loss=0.209
I0330 00:42:08.706420 140095675557760 model_lib_v2.py:682] Step 6800 per-step time 0.164s loss=0.209
INFO:tensorflow:Step 6900 per-step time 0.117s loss=0.188
I0330 00:42:22.205749 140095675557760 model_lib_v2.py:682] Step 6900 per-step time 0.117s loss=0.188
INFO:tensorflow:Step 7000 per-step time 0.121s loss=0.202
I0330 00:42:35.660460 140095675557760 model_lib_v2.py:682] Step 7000 per-step time 0.121s loss=0.202
INFO:tensorflow:Step 7100 per-step time 0.108s loss=0.231
I0330 00:42:49.805663 140095675557760 model_lib_v2.py:682] Step 7100 per-step time 0.108s loss=0.231
INFO:tensorflow:Step 7200 per-step time 0.135s loss=0.211
I0330 00:43:03.090872 140095675557760 model_lib_v2.py:682] Step 7200 per-step time 0.135s loss=0.211
INFO:tensorflow:Step 7300 per-step time 0.120s loss=0.231
I0330 00:43:16.200428 140095675557760 model_lib_v2.py:682] Step 7300 per-step time 0.120s loss=0.231
INFO:tensorflow:Step 7400 per-step time 0.148s loss=0.218
I0330 00:43:29.786302 140095675557760 model_lib_v2.py:682] Step 7400 per-step time 0.148s loss=0.218
```

Figure 26 Model training progress

In figure 24, the code to initiate the training can be seen. The code is simple because all the configurations have already been done in a text file from the sections before this. Figure 25 shows the model training progress. Only one version of configurations and model training is shown here for simplicity but different parameters and datasets have been attempted to find a model that fits the project's scope best.

5.4.4 Exporting the Model

```
last_model_path = '/content/training/'
print(last_model_path)
!python /content/models/research/object_detection/exporter_main_v2.py \
    --trained_checkpoint_dir {last_model_path} \
    --output_directory {output_directory} \
    --pipeline_config_path {pipeline_file}
```

Figure 27 Export model code

```
from google.colab import files
!zip -r /content/saved_model.zip /content/fine_tuned_model/saved_model/

files.download("/content/saved_model.zip")
files.download(label_map_pbtxt_fname)
```

Figure 28 Downloading model to device

Last but not least is exporting the model as seen in figure 26 and then downloading it to the local device following figure 27.

5.4 Creating the Application

5.4.1 Designing the GUI

```
frame_misc.place(x=0,y=0,height=HEIGHT,width=WIDTH)
frame_logo = tk.Frame(master=frame_misc,relief=tk.RIDGE,borderwidth=
frame_control = tk.Frame(master=frame_misc,relief=tk.RIDGE,borderwid
label_logo = tk.Label(master=frame_logo,image=img_logo,width=int(WID
label_detect = tk.Label(master=frame_logo,text="Detected word : ")
label_buffer = tk.Label(master=frame_logo,text="--------------------
label_result = tk.Label(master=frame_logo,text="Detected sentence")
label_voice = tk.Label(master=frame_control,text="test",fg="red")
tb_result = tk.Label(master=frame_logo,text="",bg="black",fg="white"
label_trans = tk.Label(master=frame_logo,text="Translated sentence")
tb_trans = tk.Label(master=frame_logo,text="",bg="black",fg="white")
option_lang = ttk.Combobox(master=frame_control,textvariable=CHOSEN_
cb_lang = tk.Checkbutton(master=frame_control, text="Translate", var
btn_con = tk.Button(master=frame_control,text="Check for WiFi",comma
btn_start = tk.Button(master=frame_control,text="Start",command=lamb


cb_lang.config(state=tk.DISABLED)
option_lang.config(state=tk.DISABLED)
label_detect["font"] = FONT
tb_result["font"] = FONT
tb_trans["font"] = FONT
label_buffer["font"] = FONT
label_trans["font"] = FONT
label_result["font"] = FONT
btn_start["font"] = FONT
cb_lang["font"] = FONT
btn_con["font"] = FONT

CHOSEN_LANGUAGE.trace("w", lambda *args: check_voice(label_voice))
TRANSLATE.trace("w",lambda *args: reset_voice())
frame_logo.pack(side=tk.TOP,fill=tk.BOTH,expand=True)
frame_control.pack(side=tk.BOTTOM,fill=tk.BOTH,expand=True)
label_logo.pack(side=tk.TOP,fill=tk.BOTH,expand=True)
label_detect.pack(side=tk.TOP,fill=tk.BOTH)
label_buffer.pack(side=tk.TOP,fill=tk.BOTH)
tb_trans.pack(side=tk.BOTTOM,fill=tk.BOTH)
label_trans.pack(side=tk.BOTTOM)
tb_result.pack(side=tk.BOTTOM,fill=tk.BOTH)
label_result.pack(side=tk.BOTTOM)
```

Figure 29 GUI code snippet

Figure 30 Transign GUI

The GUI for the application was designed with Tkinter, a python library for drawing graphical elements. In figure 28, a snippet of the GUI code can be seen followed by how the complete product looks in figure 29.

5.4.2 Sending Data for Predictions

```
def show_frame(model,labels,text,label_detect,tb_trans,window,cb,op,src=0):
    cap = cv2.VideoCapture(src)
    thread_inf = Thread(target=update_text,args=(cap,model,labels,text,label_detect,tb_trans,cb,op,))
    thread_inf.start()
    window.protocol("WM_DELETE_WINDOW",prevent_exit)

def update_text(cap,model,labels,text,label_detect,tb_trans,cb,op):
    counter = 0
    last_detect = ""
    last_insert = ""
    cb.config(state=tk.DISABLED)
    op.config(state=tk.DISABLED)
    now = datetime.now()
    while cap.isOpened() :
        _, frame = cap.read()
        output_dict = run_inference(model, frame)
        frame = cv2.resize(frame, (800, 600))
        if output_dict['detection_scores'] > 0.5 :
            value = output_dict['detection_classes'][0]
            obj = labels.get(value)
            obj_name = obj.get('name')
            label_detect.config(text = "Detected word : {}".format(obj_name))
            if  counter == 5 and obj_name == last_detect:
                if obj_name != last_insert :
                    text.config(text="{} {}".format(text["text"],obj_name))
                    last_insert = obj_name
                    now = datetime.now()
```

Figure 31 Camera reading snippet

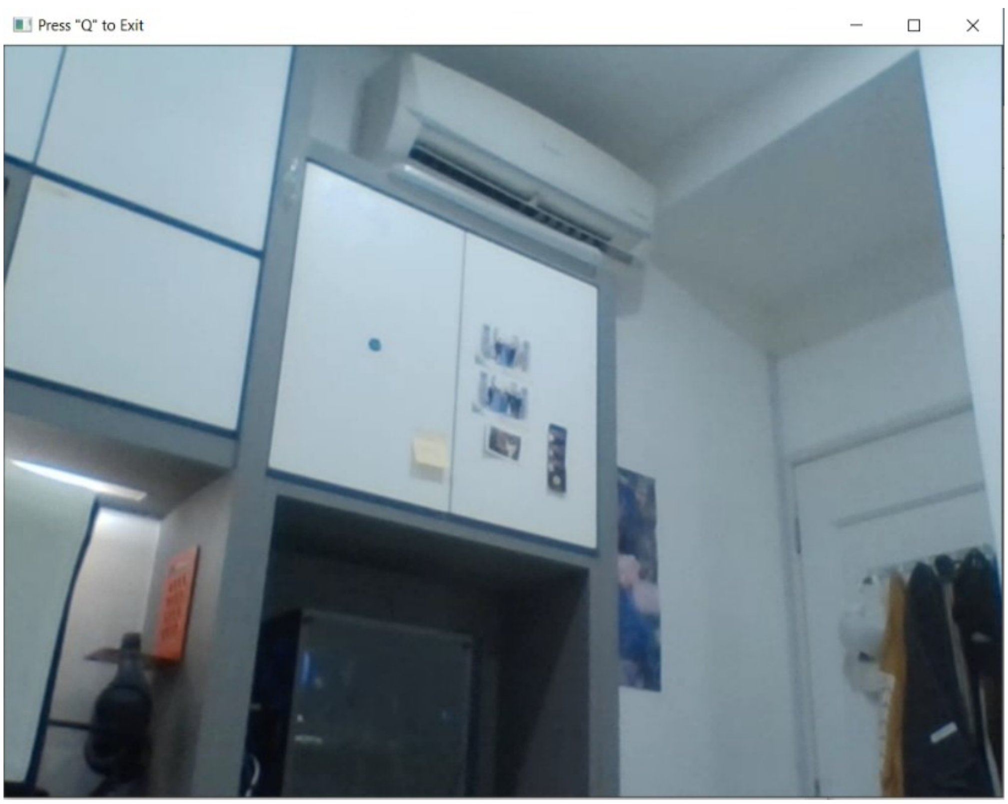


Figure 32 OpenCV Window

In figure 30, a snippet of code to start the camera and read the frames can be seen. The show_frame function initializes the camera and then passes the object over to update_text where tasks such as inference, updating text graphics, and calling the speech synthesizer will be done. Figure 31 shows what the OpenCV window looks like to the user.

```
def run_inference(model, image):
  image = np.asarray(image)
  # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
  input_tensor = tf.convert_to_tensor(image)
  # The model expects a batch of images, so add an axis with `tf.newaxis`.
  input_tensor = input_tensor[tf.newaxis,...]

  # Run inference
  model_fn = model.signatures['serving_default']
  output_dict = model_fn(input_tensor)

  # All outputs are batches tensors.
  # Convert to numpy arrays, and take index [0] to remove the batch dimension.
  # We're only interested in the first num_detections.
  num_detections = int(output_dict.pop('num_detections'))
  output_dict = {key:value[0, :num_detections].numpy()
                 for key,value in output_dict.items()}
  output_dict['num_detections'] = num_detections

  # detection_classes should be ints.
  output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

  # Handle models with masks:
  if 'detection_masks' in output_dict:
    # Reframe the the bbox mask to the image size.
    detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
              output_dict['detection_masks'], output_dict['detection_boxes'],
               image.shape[0], image.shape[1])
    detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                       tf.uint8)
    output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

  return output_dict
```

Figure 33 Inference code snippet

The frames read from the camera are passed to the function shown in figure 32. The function run_inference converts the frames into a 2d array and then converts it into tensors which is the input parameters of the model for predictions. The model will then output a dictionary of values, the key ones being the detected class, confidence, and their locations on the inferenced frame.

```python
def convert_speech(data,tb_trans):
    for i in range(len(PHRASE)):
        if SWITCH[i] in data:
            data = data.replace(SWITCH[i],PHRASE[i])
    if TRANSLATE.get() == 1 and not data is None:
        data = translate_speech(data)
    tb_trans["text"] = data
    engine.say(data)
    engine.runAndWait()
```

Figure 34 Speech code snippet

```python
def translate_speech(data):
    tler = Translator()
    end_lang = "en"
    if not CHOSEN_LANGUAGE.get() == "" :
        end_lang = LANGUAGES[CHOSEN_LANGUAGE.get()]
    tl = tler.translate(data,src="en",dest=end_lang)
    return tl.text
```

Figure 35 Translation code snippet

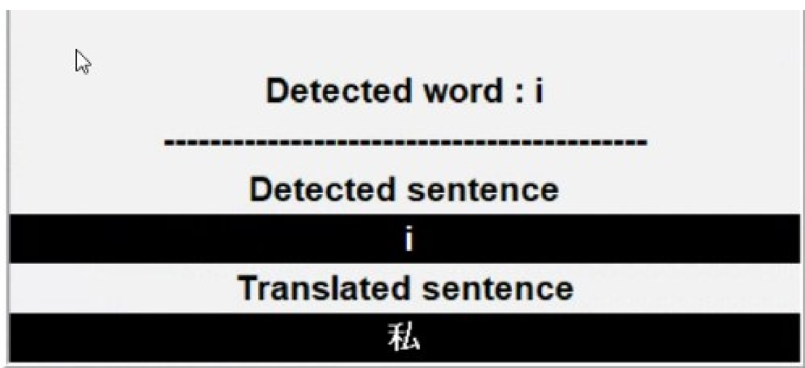


Figure 36 Displayed text

Once the predictions have been completed, the data will be sent over to the convert_speech function to be synthesized into a speech. If the user has the translation option enabled, the data will then again be passed over to translate_speech to obtain the translation in the user's choice before it gets synthesized. In figure 35, the results of the detection and translation can be seen.

5.4.3 Enabling translation

```
def msg_box(cb,btn,optn):
    btn.config(state=tk.DISABLED)
    temp_window = tk.Tk()
    temp_window.title("WiFi check")
    temp_label = tk.Label(temp_window,text="Hold on, chec
    temp_label.pack(side=tk.TOP,fill="both",expand=True)
    thread_int = Thread(target=check_internet,args=(cb,te
    thread_int.start()
    temp_window.mainloop()

def check_internet(c,l,w,b,o):
    try:
        response = urlopen('https://www.google.com/', tir
        c.config(state=tk.NORMAL)
        l.config(text="Connected")
        time.sleep(1)
        b.config(state=tk.NORMAL)
        w.destroy()
    except:
        c.config(state=tk.DISABLED)
        if TRANSLATE.get() == 1 :
            c.deselect()
        o.config(state=tk.DISABLED)
        l.config(text="Not connected")
        time.sleep(1)
        b.config(state=tk.NORMAL)
        w.destroy()
```

Figure 37 Internet check code snippet

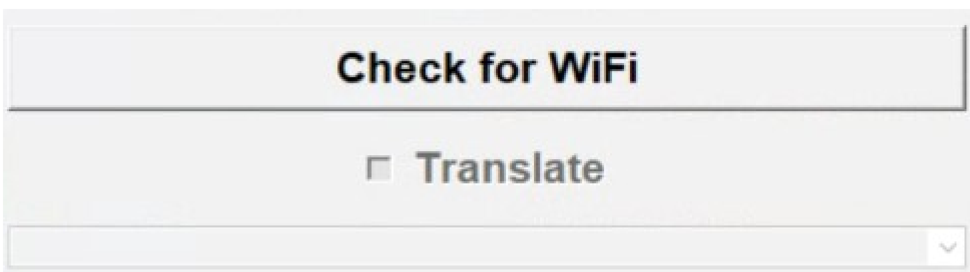


Figure 38 Disabled translation option

Figure 36 shows a snippet of the code to check for internet access. If internet access has not been confirmed, the translation options will be greyed out and the user cannot interact with the widgets as seen in figure 37.

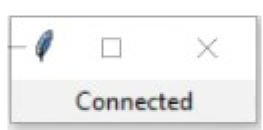


Figure 39 Message box for Internet access

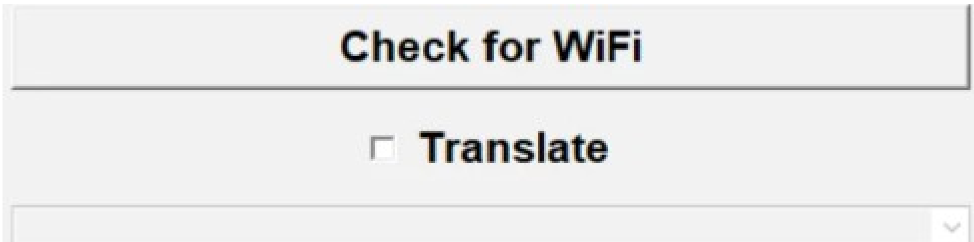


Figure 40 Enabled translation option

Figure 38 shows the message box created by the msg_box function. If the user has choppy connection to wifi, the message box would instead display another message saying "checking for internet connection" for 10 seconds before the functions deems the attempt at connecting a failure. At that time, a different failure message will be displayed instead. If internet access has been confirmed, the widgets for translation would be enabled and interactive with the user.

Chapter 6: Testing

6.1 Overview

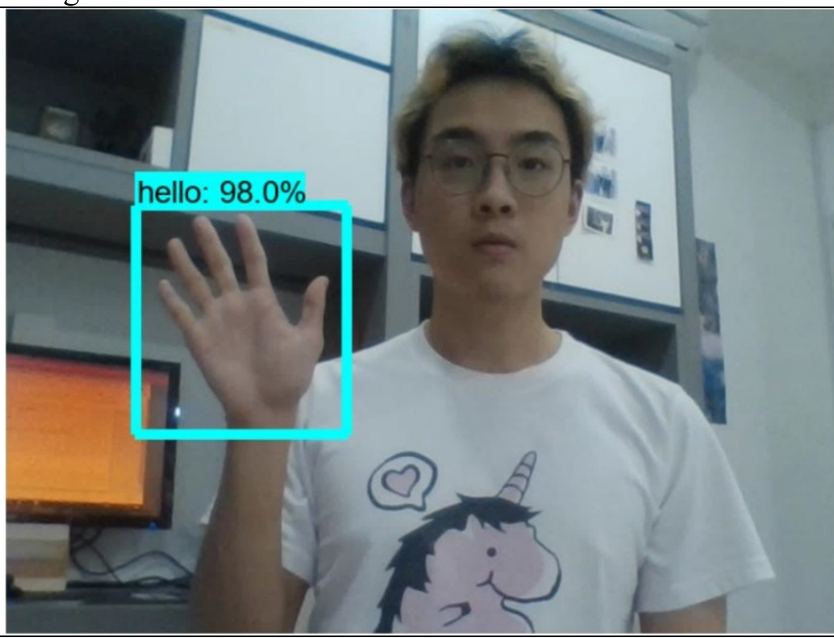
In this chapter, the overall testing done for Transign will be discussed. This is to evaluate the application based on the specified requirements. In this project, several testing methods were used to justify the quality of the product for end user to use it without any issues. Moreover, test plans and test results will be discussed under this chapter.

6.2 Black Box Testing

The first method that will be used for testing is called black box testing. Black Box Testing is a testing technique that tests the functionality of the Application Under Test (AUT) without seeing the internal code structure, knowledge of internal paths of the software and also implementation details (Nidhra, 2012). This type of testing is based on software specifications and requirements. In general, this testing will only focus on input and output of the software system without the internal knowledge of the software. Firstly, the specifications and requirements of the system are checked. Then, a tester chooses input that is valid and also few invalid inputs to check whether the system can detect them. Then the tester will write down the expected outputs for said inputs. The test cases will then be executed and the output produced will be compared to the expected output.

Defects detected will then be fixed and re-tested after (Nidhra, 2012). An example of the black box testing can be seen in the table below while the rest will be within the appendix.

Table 2 Black Box Test Case 1

Test Case	1
Test Case Description	Predict the "Hello" hand sign correctly and nothing else.
Input Data	Webcam stream
Pre-conditions	-
Expected Output	"Hello" is predicted with high confidence
Procedures	1. Start the application with GUI.py 2. Start the webcam 3. Sign "Hello"
Screenshot of produced output	
Post-condition	The correct hand sign and location is outputted.
Test Result	Passed

6.3 White Box Testing

The next testing method used is white box testing technique. The white box testing technique tests a software's internal structure, coding and design. For this kind of testing, code snippets can be seen by the tester (Anon., 2017). The tests will be carried out by a developer knowledgeable regarding the system. The first step that must be taken by the tester is to understand the functionality of the system after observing its source code. After that, the tester has to create the test cases and then execute them. An example test case can be seen in the table below while the rest will be attached in the appendix.

Table 3 White Box Test Case 1

Test Number: 1	**Function Name:** check_internet()
Parameters Taken: cb_trans	
Test Case Description: allow translation option after internet access is verified	
Expected Result: translation widget is enabled	
Pre-condition: Internet access is present	
Test Execution Steps: 1. Ping Google's server 2. Wait for response 3. Enable widget based on response	
Post-conditions: Translation widget is interactive	
Test Result: Passed	

6.4 Model Evaluation

For the evaluation of the custom trained object detection model's performance, the COCO mean Average Precision (mAP) metric. The COCO mAP metric has been the preferred method of displaying a model's performance in recent trends of research papers so this report will also use the COCO mAP metric (Hui, 2018). The COCO mAP metric is based off the COCO mAP, where a 101-point interpolated AP definition is used for the calculation. The COCO mAP metric has a factors to calculate the performance, Intersection over Union (IoU), Average Precision (AP), and Average Recall (AR). IoU is the overlap between the predicted boundary of the model and the real object boundary. AP is an average value of each prediction's accuracy while AR is the averaged value of positive cases found compared to total number of possible positive cases (Anon., 2019). Tensorflow's Object Detection API comes with evaluation tools and the results of the evaluation on said model can be seen in the figure below.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.746
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.991
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.962
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.746
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.781
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.781
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.781
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.781
```

Figure 41 Evaluation metrics

The area column has results displaying -1 due to the purpose the model was trained for. Some models trained for object detection targeted small objects such as detecting different color pieces of corn on a cob which is small in size in comparison to this model's purpose of detecting hand signs which take up a significant amount of space within a frame at any time.

Chapter 7: Evaluation

7.1 Overview

This chapter discusses about the functionalities of the application by evaluating the features and also the process to complete the product. This chapter will also explain the idea on the application's overall description and the process went through by the developer to develop this end to end deep learning for self-driving vehicles.

7.2 Product Evaluation

The detailed evaluation of product was done after the implementation and testing phase of the system. Strengths and weaknesses of the system based on the functionality will be discussed in product evaluation.

7.2.1 Product Functionality

The main objective of this application is to take in real time video feed of a person performing hand signs and converting it to speech through a mobile application which has been mostly achieved. The reason why the term "mostly" is used is due to the problem of implementing the idea on a mobile platform which will be explained more later on in problems faced. Moving on, the application produced achieves the objective by first presenting a user-friendly GUI for the user to start the camera stream. Once the camera stream is started, the user is free to perform the hand signs and this can be further confirmed by the user through the frame that shows up on the screen in 800x600 pixel resolution for resource efficiency as explained in the previous chapters.

The frames are then sent in real time to the model where predictions will be made on what hand signs are being made by the user. Not only that, the user can also see whether the meaning they want to convey is correctly predicted by the model as the sentence is built and shown in real time on the GUI.

Once the predictions have been made and the text has been displayed on the GUI, the output will be sent to the speech synthesizer to be voiced in an English speaking voice or another language if the translation option was enabled. The user will first have to tell the GUI to check for Internet connection first for the translation option to be enabled. Not only that, the user has to install the Windows speech pack in the language they wish to translate and voice the hand signs they perform. A warning message will be displayed to the user if the language they chose does not have a speech pack installed locally but the application will still function and display the translated text but only voice the English text version of the hand signs.

7.2.2 Strengths and Weaknesses

One of the strengths of the application is that it can be mostly run offline. The application does not need Internet access if the translation option is not needed which fulfills its core role of helping mute or deaf people communicate in sign language to its relative spoken language. Besides that, the application does not need a super computer to run and can be run as long as a speaker and camera is present with the device. Moving on, no data is of the user is stored anywhere locally or online as facial data is a sensitive intellectual property that has no right being exposed or stored.

Despite that, the system also has its weaknesses. The accuracy of the model is currently good in targeted situations only due to the lacking data repositories of different people performing different hand signs in different environments which is too severe to be solved by just data augmentation. All data used in the application thus far is created by the developer alone which makes this application more of a proof-of-concept right now.

7.2.3 Mobile Implementation

A special section is required to explain the above topic. The initial proposal was to have the application function within a mobile device running the Android operating system. During the analysis phase, the idea was research on and deemed feasible with the number of examples found in public sources. The neural network was to be implemented using TFLite and ran locally in real time. With that said, the implementation was a failure with the reason unknown. The model was retrained multiple times with different datasets and parameters, performed as expected when run on a non-mobile device but performed drastically worse, to the point that it was unusable, when converted to TFLite format on mobile. Even after much troubleshooting and discussion on online platforms, the problem was not solved. The decision to convert to an implementation with Python was made when problem of the deadline nearing came up.

7.3 Process Evaluation

In this sub chapter, the process and phases in development of AutoMove will be discussed based on the project plan created during the writing of the proposal and then reviewed on its execution. The planned durations can be found in the Gantt Chart of the Appendix.

7.3.1 Initiation Phase

Table 4 Initiation Review

Initiation Phase	Start Date	End Date	Duration (Days)
Project Plan	14-09-2020	02-10-2020	19
Actual Progress	14-09-2020	28-09-2020	14

During the initiation phase, project ideas were short listed and discussed based on their feasibility and practicality with the supervisor. An appropriate idea and title were then chosen followed by appropriate research on the topic. Lastly, the project initiation document (PID) is prepared and submitted after reviews from the supervisor.

7.3.2 Planning Phase

Table 5 Planning Review

Planning Phase	Start Date	End Date	Duration (Days)
Project Plan	03-10-2020	27-11-2020	56
Actual Progress	03-10-2020	20-11-2020	50

Planning phase is a particularly important phase for the development of this project. Neural networks and machine learning is still a relatively new technology recently commercialized for everyday use by society. Due to this, there is only so much research and papers on the specific subtopic that this project is looking for. During this phase, articles and papers were constantly brought up and discussed with the supervisor to discuss its relevancy to this project's topic. After that, the project proposal draft was written and then submitted to the supervisor and second marker. The second marker and supervisor would return this draft after reviewing it and then changed based on the feedback. The proposal is then submitted a few days earlier as extensive research and discussion was already done before the review so only basic changes were required. One thing of note is the start date of the planning phase is the same as planned despite the early finish on the initiation phase due to the need to focus on core subjects during the semester.

7.3.3 Analysis Phase

Table 6 Analysis Review

Analysis Phase	Start Date	End Date	Duration (Days)
Project Plan	01-12-2020	05-12-2020	5
Actual Progress	27-11-2020	05-12-2020	9

Similar to the planning phase, the start date is only a little earlier than the planned start date due to settling other assignments. Despite starting earlier, the analysis phase took longer than planned due to the under estimation of the difficulty of implementing the idea. The analysis phase began with research on how Tensorflow functioned, then on the different model architectures that existed and how they functioned with Tensorflow. After that, journals and articles related to object detection, human limbs recognition, and hand sign recognition were studied to determine the order of implementation. As explained in the literature review phase, CNN was chosen as the neural network with SSD algorithm to accompany it. The UML diagram, sequence diagram, and functional requirements were quickly done after the literature review was completed and had no problems that required it to be changed.

7.3.4 Implementation Phase

Table 7 Implementation Review

Implementation Phase	Start Date	End Date	Duration (Days)
Project Plan	10-12-2020	15-01-2021	39
Actual Progress	05-12-2020	20-01-2020	49

The implementation phase started right after the analysis phase as the difficulty of the project was realized during that time coupled together with the developer's inexperience with any practical implementations related to machine learning. The duration was also longer due to the problem with the mobile implementation as explained in the testing section of this report. Other than that, the application was implemented in the order of the functional, non-functional requirements and UML diagrams created. Most of the time used was gathering custom data for training and experimenting with different augmentations, parameters, and datasets to train the model. If the developer had more experience with machine learning prior to this project, the time taken for this phase could have been shaved down.

7.3.5 Testing Phase

Table 8 Testing Review

Testing Phase	Start Date	End Date	Duration (Days)
Project Plan	10-01-2021	24-01-2021	15
Actual Progress	15-01-2021	24-01-2021	10

Initially, the testing phase was planned to be from the 10th of January to 24th January, done in tandem with the implementation phase as the prototyping methodology was used. Due to the difficulty of the implementation, the testing phase was cut short to a 10 day duration instead of the planned 15. Luckily, no problems were met during the testing and documenting so the planned end date was met.

7.3.6 Documentation Phase

Table 9 Documentation Review

Documentation Phase	Start Date	End Date	Duration (Days)
Project Plan	24-01-2021	09-02-2021	17
Actual Progress	24-01-2021	09-02-2021	17

The last phase is the documentation phase, although documentation was to be done during this duration, bits and pieces were already written as the project was done during design and implementation so the rest of the report was written without a hitch in time. The draft was then sent to the supervisor for review and changes were then made accordingly.

Chapter 8: Conclusion and Recommendation

8.1 Conclusion

Transign is a deep-learning sign language-to-speech translation application that can convert American Sign Language to English and many other languages in real time. The backbone of this application is a model created following the CNN architecture using the SSD algorithm to perform real time object detection and classification.

This system was developed in hopes that deaf or mute people can have an easier way of communication with common people when common means such as writing text on a notepad or phone for people to read is not possible. According to the World Federation of the Deaf in 2020, there are approximately 72 million deaf people worldwide and using sign language. That population is only 1% of the world's population which makes sign language a rare skill found among the world's people. Not only that, sign language consists of 300+ different languages similar to the spoken languages of the world (Deaf, 2020). Transign hopes to help bridge the gap with technology so that communication is all the more accessible for deaf or mute people within the world.

8.2 Recommendation

Transign is closer to a proof of concept as said in the product evaluation and has many areas where it can definitely be improved on. The recommendations will be split into two sections, improvements on existing features and new features that can be implemented.

In existing features, the speech pack's installation can be automated instead of requiring the user to manually install the speech packs. This can be done by doing more research on how Windows