PAPER NAME

Pantomime-Manuscript-1-1.docx.pdf

WORD COUNT

11116 Words

CHARACTER COUNT

57373 Characters

PAGE COUNT

81 Pages

FILE SIZE

13.5MB

SUBMISSION DATE

Jul 29, 2023 6:41 PM GMT+8

REPORT DATE

Jul 29, 2023 6:43 PM GMT+8

12% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 11% Internet database
- 5% Publications database
- Crossref database
- Crossref Posted Content database

Excluded from Similarity Report

- Submitted Works database
- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 10 words)
- Manually excluded sources
- Manually excluded text blocks

TECHNOLOGICAL UNIVERSITY OF THE PHILIPPINES

College of Engineering

Electronics Engineering Department

ELE 11 – METHODS OF RESEARCH

Pantomime: Tensorflow based Dynamic Filipino Sign Language Gesture Recognition using YOLO

PAULE, Ella Mae S.

POMASIN, Anchelo Oscar C.

REGUDO, Gian Eduard T.

TENCHAVEZ, Ron Gerard L.

ZABALA, Arnell B.

Chapter 1

THE PROBLEM AND ITS BACKGROUND

1.1 Introduction

Speaking is an important part of human life. Most of the people today communicate in this manner and not all 7 billion people can communicate. Deafness can have an influence on childrens' ability to speak and develop or learn a language. When a child has difficulty in hearing, the areas of the brain responsible for communication may not develop properly, making understanding and speaking difficult. When a child's hearing loss is detected early and treated appropriately, he or she can become an effective communicator. Caregivers and professionals collaborate in this process. The majority of hearing losses are discovered during a newborn screening. Some children are not given a diagnosis until their speech or language skills are failing. Early recognition and treatment of hearing loss leads to better outcomes for children.

The WHO estimates that 460+ million people, or around 5% of the world's living population have some form of deafness, but only 1.23% of people in the Philippines are said to be deaf or mute.

However, since it takes a long time to understand properly, the majority of hearing people are unaware of it.

The concept of a sign language translator is not new. However, the majority of the common population do not comprehend sign language, and learning it is a challenging task. Consequently, there is still a sizable disparity between the hearing majority and the deaf population.

Assistive machine learning technologies such as Artificial Neural Network (ANN) and Artificial Intelligence were used to include in the sign language recognition system that assists in the creation of a fully inclusive community. Using the right machine learning algorithm to interpret sign languages and translate them into words, or vice versa, in the shortest time possible is a huge aid in the everyday lives of the people without speech and hearing disability, and other PWDs who have difficulty communicating and participating in the society.

1.2 Background of the Study

Many efforts have been made over the past few decades to develop a Sign Language Recognition (SLR) system. SLR was previously divided into two groups depending on the tools, such as datagloves-based SLR and vision-based SLR, that were utilized to record hand motions. As technology advances, gesture-based SLR has been added to the list. The user's range of motion is highly constrained in the first scenario, and data gloves must be properly cared for. The second scenario, however, allows for more natural user interaction with the computer, but many technical issues with image processing remain unresolved or need to be addressed due to the limitations of current computer vision techniques, necessitating the use of colorful gloves or other visual aids.

SLR is divided into two categories: Isolated Sign Language and Continuous Sign Classification. Isolated Sign Recognition is concerned with the recognition of signs that are performed alone, with no signs before or following the performed sign. As a result, the sign is done without regard for the preceding or consecutive

signs. SLR faces difficulty in creating ways to tackle continuous vocabulary-heavy sign issues. A significant contribution to large-vocabulary continuous SLR research is unquestionably necessary and has an impact on how natural human-computer interfaces feel. to encourage the widespread use of an SLR system.

Managing epenthetic movement is the main difficulty with Continuous SLR. The movement epenthesis, also known as the shift between two adjoining hand signs, differs depending on the context of the preceding and succeeding signs and occurs at their respective ends and beginnings. Since the epenthesis adds a variety of additional movements that are not present in the grammatical forms of the signs and doesn't just affect the performance of nearby signs, its inclusion significantly increases the difficulty of recognizing the signs.

In context-dependent models for continuous SLR, like the triphone or biphone are frequently used to simulate the coarticulation. In constant SLR, however, no fundamental unit such as the phoneme of speech has yet been identified in the sign lexicon. The number of subunits retrieved manually or mechanically for the entire sign language is so great that training data becomes extremely scarce. This makes it impossible to train context-dependent models for large-vocabulary SLR, such as those described in the literature. The same problem affects direct models of the movement of epenthesis between signs.

Some used probabilistic video processing techniques, including the renowned MotionSavvy's Leap's 3D Motion and Hidden Markov Model or Artificial Neural Network classifiers. While these studies aim to translate sign

language into voice and/or text, a proposed system will be as accurate as or more accurate than the studies shown.

You Only Look Once or YOLO, a new approach to object detection was made last 2016 (by Redmon et al). In earlier labor on object detection, classifiers were repurposed to carry out detection. The creators envisioned object recognition as a decline problem that involves the class and bounding boxes probabilities. Within an assessment, the neural network guesses the boxes and class possibilities straight from the whole images. Since the whole detection conduit is a singular network, the spotting operation may be tuned point-to-point.

In recent years, the involvement of vision-based approaches for sign language recognition became a major subject for research development with a lot of influential and modern implementations such as gesture recognition interpretation, motion control, and human-computer interaction (HCI) had already made recognition in ongoing research topics. Due to a number of problems, including the complex nature of static or dynamic hand gestures, distortions, and complex backgrounds increases the difficulty.

The need of very intensive computer resources to address the issue in its generality which requires elaborate algorithms.

While these studies aim to translate sign language into text, a system will be proposed that will match and perhaps even outperform the studies' degree of accuracy. Thus, You Only Look Once or YOLO Convolutional Neural Network in

our proposed system for recognizing hand gestures will be used for faster and more accurate results.

YOLO's centralized structure develops very quickly. The basic YOLO model performs 45 frames per second of real-time image processing. Fast YOLO, a scaled-down version of the network, processes astonishingly 155 frames per second while still outperforming other real-time detectors in mAP by a factor of two. If compared side to side with other detector systems, However, YOLO is less likely to forecast false positives in the background and creates further localization errors. Lastly, YOLO picks up very broad representations of objects. When applied to other domains like artwork, it performs better than other detection techniques like DPM and R-CNN when generalizing from natural images.

1.3 Statement of the Problem

Through hand gestures, body language, facial expressions, people can express themselves visually. Although sign language is the primary means of communication for the Deaf and Hard-of-Hearing community, it can also be helpful for other populations, including those with autism, with speech disorders, cerebral palsy, and down syndrome.

In a way, the statement that learning sign language is difficult is both accurate and untrue. It primarily relies on the kind of sign language you're attempting to learn. Having said that, learning a language is much harder as you become older than it is while you're younger. Additionally, it's typically impossible to fully immerse yourself in sign language outside of the classroom. Your

interactions may frequently be restricted to one deaf family member. Your objectives and demands would be very different if you were speaking with a child as opposed to an adult. This could affect how quickly or slowly you pick up the language.

The usual speed of social communication, especially when done in a group, can be overwhelming, which is another difficulty in learning sign language. It elevates communication to a whole new level and necessitates the mastery of eye gaze to more adeptly manage the give-and-take of social relationships.

The main problem is how people who have no background in sign language will be able to communicate to people with hearing and speech disabilities. Considering the limitations of both parties, the portability, functionality, and accessibility of the translator should also be considered.

1.4 Objectives of the Study

1.4.1 General Objectives

The main objective of this study is to develop a mobile application that will translate Filipino dynamic gestures into words and vice versa using machine learning.

1.4.2 Specific Objectives

1. To develop a dynamic mobile translator application that utilizes a deep learning called You Only Look Once (YOLO).
2. To design an interactive easy-to-use user's interface using Tensorflow and Python as the programming language.

3. To provide a learning module in the mobile application for those who wanted to learn sign language.
4. To test the developed application, and evaluate the accuracy of the translation in its functionality, reliability, usability, efficiency and its maintainability as specified in the ISO 9126 Software Quality Standards.

1.5 Significance of the Study

There are 10,000 words that cannot be translated into single sign language. It can be efficient for everyone who has difficulty in communicating with people who have hearing impairment to translate the dynamic gestures into words so people without understanding of the sign language will be able to communicate with them. It can lessen the stigma, promote awareness, and give more job opportunities to the people who have hearing impairment. Also, it can help their family and friends to understand and learn sign language. The proposed system is designed to create an application that would convert one of the two inputs to an output form; to text provided by the device display, and video taken using the device camera to an audio recorded by a microphone. Therefore, the proposed system will be using a YOLO and will be testing which variable of YOLO is suited for mobile application development. Tensorflow and python will be used for the mobile application development that will be programmed for the automated translations that will help the conversations between those who are not well versed in the Filipino Sign Language and those who have speech disability or have hearing disability, providing a convenient way to bridge the language barrier between the two populations.

1.6 Scope and Limitation

1.6.1 Scope

The study focuses on developing a system that could translate common words and expressions used by the deaf community in their everyday lives. As we will be using YOLOv4 for the accuracy of detecting hand gestures and the reliability of the detection. Then all of the detected gestures will be processed and be converted into words understandable for the intended receiver. Thus, an application that can recognize a list of Filipino Sign Language (FSL) dynamic gestures.

1.6.2 Delimitation

Sign Language doesn't only involve hand gestures but it includes the whole body in communicating, this includes facial expressions and body movements. This study will not cover facial expression recognition. The lexicon of gestures will be also limited in the most frequently used phrases of a person such as the terms used in transportation, school or work and basic conversation.

The proposed system will be using Filipino Sign Language (FSL) as the main sign language instead of American Sign Language (ASL). Although FSL was originally rooted from the ASL and soon developed and became a separate language from ASL, the Filipinos who's suffering from hearing disability and speech disability used it in communication and was mandated to be the official sign language of the Philippines. As Filipino signs that emerged naturally through

generations and adding new signs, especially those that are related to technology (Apurado and Agravante, 2006)

1.7 Definition of Terms

The following are the definitions of words used in the paper.

- **Deep Learning -** consists of a subset of machine learning in Artificial Intelligence (AI) that comprises networks that can learn autonomously from unstructured, unlabeled input.
- **Python** - is an interpreted, readable, object-oriented programming language with a simple syntax. Since its statements can be interpreted in a variety of ways, it is said to be relatively simple to learn and portable. systems operating.
- **American Sign Language (ASL)** - is the sign language most commonly practised by deaf communities in the United States and most of Anglophone Canada.
- **Filipino Sign Language (FSL) -** Sign language from the Philippines is known as Philippine Sign Language. FSL is a distinct language with its own grammar, syntax, and morphology like other sign languages; it is neither based on nor similar to Filipino or English. Most institutions across the nation use FSL as their primary sign language.
- **Lexicons -** a book that contains the words in a language alphabetically and their respective definitions.

- **Keras -** is an open-source software library that offers Python support for implementations and tools to simplify the coding required for an artificial neural network by making image and text data easier to handle.
- **Neural Network** - is a series of algorithms that aims to mimic the human brain in perceiving the relationships between data in a set.
- **TensorFlow -** is a machine learning and artificial intelligence software library that is open-source and free. It can be used for a variety of tasks, but focuses particularly on deep neural network training and inference.
- **Convolutional Neural Network** - also called CNN, or ConvNet, is a class of deep neural networks that specializes in analyzing and processing data with grid-like topology, like digital images.
- **Dynamic Gesture Recognition Interface** - is an interface that allows the recognition of gestures like hand signals from an active camera footage.
- **OpenCV -** developed by Intel, is a free-to-use cross-platform library of programming functions aimed at real-time identification and understanding of digital images or videos.
- **Yolobile Framework -** a real-time object framework that is used for mobile devices to improve speed and accuracy of object detection in mobile devices

Chapter 2

REVIEW OF RELATED LITERATURE

This part of the research is where the ideas in different studies come together to develop an innovative project with a background that has a basis and purpose for improvement. It circles on theories and principles of related literature and studies in conceptualizing ideas that are useful to the whole research process.

2.1 Conceptual Literature

2.1.1 You Only Look Once used for Object Detection

YOLO is a method for detecting objects. Earlier object detection research uses classifiers in a new way to carry out detection. Instead, the researchers (Redmon, Divvala, Girshick, Farhadi, 2016) frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Bounding boxes and class probabilities are directly predicted by a single neural network from whole pictures in a single assessment. The whole detection process consists of a single network, therefore may be fully tuned based on detecting performance.

The underlying YOLO model analyses pictures in real-time at a rate of 45 frames per second because to the unified architecture's lightning-fast speed. Fast YOLO, a scaled-down version of the network, processes just 155 frames per second

while still outperforming other real-time detectors in mAP by a factor of two. YOLO produces more localization mistakes than cutting-edge detection algorithms, but it is less likely to anticipate false positives in the background. Last but not least, YOLO picks up very broad representations of things. When applied to other domains like artwork, it performs better than other detection techniques like DPM and R-CNN when generalizing from natural pictures.

Capitalizing on the speed and accuracy that YOLO offers on object detection, a recent study shows that YOLO can be used in mobile devices, via compression-compilation co-design. The resulting model is called the YOLObile framework.

Using a noble-block pruning scheme and a GPU-CPU collaborative scheme ,to improve computational efficiency, the experimental results indicate that the pruning scheme achieves 14× compression rate of YOLOv4 with 49.0 mAP (Yuxuan Cai et. al, 2020).

Under the YOLObile framework, the researchers achieved 17 FPS inference speed using GPU on Samsung Galaxy S20, and after incorporating the GPU-CPU collaborative scheme, the resulting inference speed increased to 19.1 FPS, and outperformed the original YOLOv4 by 5 times (Yuxuan Cai et. al, 2020).

Table 2.1 Summary Table for Object Detection using You Only Look Once

Title	Objective	Approach
You Only Look Once: Unified, Real-Time Object	To test YOLO for its accuracy and speed in	Uses a network that creates bounding boxes and

Detection	object detection.	confidence scores to find an object. The resulting box should contain an object and should be able to name it.
YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design	To utilize YOLO in creating a real-time object detection on mobile devices via compression-compilation co-design.	Uses a DNN-based object detector (YOLOv4) to detect the object, enhance the accuracy by using block-punched pruning, and use GPU-CPU collaborative scheme to improve computational efficiency.

2.1.2 Sign Language Recognition Systems

An creative, organic, and consumer way to interact with a computer that is more tailored to human needs is made available by sign recognition systems. Gesture recognition has many uses, including in interactive game technology, human-machine interaction, and sign language. By keeping in mind the similarities of human hand shape with four fingers and one thumb, the Sign Language Recognition Using Neural Network of (Bachani, Dixit, Chadha, & Prof. Bagul, 2020) aims to present a real time system for hand gesture recognition on the basis of detection of some meaningful shape based features like orientation, center of mass (centroid), status of fingers, thumb in terms of raised or folded fingers of hand and their respective location in image. The method that was put forth completely depended on the hand gesture's shape parameters. Skin tone and texture are not taken into account as additional methods of hand gesture recognition because they

are so sensitive to changes in lighting and other factors. They used a basic webcam that operates at 20 frames per second and has a resolution of 7 mega pixels to implement the strategy.

Another study uses sign language or gesture recognition in order to provide the deaf people a means to control home systems using hand signs and gestures. The system developed uses a real-time dynamic gesture recognition to detect the gestures and is trained through machine learning to understand the given command and perform it (Kraljevic et. al, 2020).

Table 2.2 Summary Table for Sign Language Recognition Systems

Title	Objective	Approach
Sign Language Recognition using Neural Network	To create a real-time system for hand gesture recognition based on shape-based features.	Uses a webcam that usss 20 fps and 7 megapixel intensity to recognize the shape of the hand.
A Dynamic Gesture Recognition Interface for Smart Home Control based on the Croatian Sign Language	To utilize YOLO in creating a real-time object detection on mobile devices via compression-compilation co-design.	Uses a DNN-based object detector (YOLOv4) to detect the object, enhance the accuracy by using block-punched pruning, and use GPU-CPU collaborative scheme to improve computational efficiency.

2.2 Related Studies

2.2.1 SignSpeak

Research on the recognition of sign language has been investigated for many years. Numerous studies have employed sensor-based tools like "SignSpeak."

This technology made use of a variety of sensors, including flex and touch sensors for the motions of the fingers and palms and accelerometer and gyroscopes for the movements of the hands. Principal Component Analysis was then used to train the gloves to identify various gestures and to categorize them into alphabets in real-time. To display the text and words received from the gloves via Bluetooth, it also utilized an Android phone. SignSpeak was found to have a 92% accuracy. (Bukhari, Rehman, Malik, Kamboh, & Salman, 2015) Though Sensor Gloves like this can give a precise hand gesture recognition, it can also give the user an unnatural experience due to the bulkiness of the gloves and its sensors. It will also encounter difficulties in setting up the device and it can be very expensive. Because of this many researchers jumped from sensor-based to visual-based Sign Language Recognition.

2.2.2 Static Sign Language Recognition using Deep Learning

In 2019, a study entitled "Static Sign Language Recognition using Deep Learning" by Tolentino, LK and et al. was published in International Journal of Machine Learning and Computing; they also created a system that will act as a teaching aid for those learning sign language for the first time that involves hand detection. The system was built using explicit complexion space thresholding, a

skin complexion modeling technique. In order to separate pixels (hand) from non-pixels (background), a skin-color spectrum has been defined.

The Convolutional Neural Network (CNN) model was used to classify the pictures after being fed the photos. Images were trained using Keras. The system achieved an average testing accuracy of 93.67% in ideal lighting and a consistent background, of which 90.04% was ascribed to ASL letter recognition, 93.44% to number recognition, and 97.52% to static word identification.

The method is real-time and utilized for quick computing and also based on a method that is explicitly defining the skin-color threshold and it was pre-determined that it will extract pixels from non-pixels and feed into the CNN model for classification of images.

2.2.3 Baybayin Handwritten Letters using VGG16 Deep Convolutional Network Model

In Recognition of Baybayin (Ancient Philippine Character) Handwritten Letters using VGG16 Deep Convolutional Neural Network Model (Bague, Jorda, Fortaleza, Evanculla, Paez, Velasco, 2020), they proposed a system that can convert 45 handwritten baybayin Philippines character/s into their corresponding Tagalog word/s equivalent through convolutional neural network (CNN) using Keras. The VGG16 network used in the implemented design is more compact and smaller.

Each of the 45 Baybayin characters has 1500 photos in the categorization. The segmentation stage's scaled characters (50x50 pixels) produced pixel values that were used to train the system, resulting in a 99.54% accuracy rate. 90

handwritten baybayin characters were taken in real time using the device's 1080P Full-HD web camera to test the detection technology.

The test sample will then be classified by the system. The user is then shown the relevant Tagalog word output on the screen. The testing phase's total accuracy is 98.84%. From the results, the suggested approach might thus be used for character extraction from documents and any associated conversion of handwritten materials into structured text.

2.2.4 Shape Based Continuous Real Time Hand Gesture Recognition System of American Sign Language using KNN Classifier

Shivashankara and Srinath (2018) suggested an ideal technique., Aiming to achieve the translation of 24 static sign language alphabets and numerals of American Sign Language into humanoid or machine readable English writing

The first step involves the signed input gesture's pre-processing processes. In the next phase, the various region properties of pre-processed gesture images are computed. The transcription of signed motions into text has been done in the final step based on the attributes computed in earlier phases. The statistical results evaluation and a graphic comparison of the current and suggested methodologies are also included in this study.

2.2.5 A Dynamic Gesture Recognition Interface for Smart Home Control based on Croatian Sign Language

People who are hard of hearing or deaf face several difficulties in daily life. Their means of communication are sign languages, and whether or not a language is accessible to them depends on how well understood it is in the cultural and social context. So according to the study, the introduction of a smart home automatization system specifically designed to provide real-time sign language recognition is needed. (Kraljevic et. al).

The researchers took a Conv3d network-based wake-up system and high - performing sign language recognition and applied them to this problem.

Based on a tiny Croatian sign language database with 25 distinct language signs, the embedded platform was implemented on a Nvidia Jetson TX2 with a StereoLabs ZED M stereo camera.

The system was trained and was tested by 40 users. The results show that the system achieved 80.9% accuracy, and concluded that the system was efficient enough to be used in a smart home environment (Kraljevic et. al.).

According to the study, the system is limited to the basics of the Croatian sign language, and is not adopted yet to understand complex grammar. The study also does not yet include a higher degree of user modalities such as face expressions and postures (Kraljevic et. al.).

Table 2.3 Summary Table for Related Studies

Title	Objective	Approach	Advantage	Disadvantage

SignSpeak	To create a sensor-based device that can translate hand signs and gestures into words.	Using gloves and sensors that were connected via Bluetooth, the system aims to recognize the hand gestures and translate it to words.	The device is very accurate in translating the hand gestures that were performed.	The gloves are said to be bulky and cause uncomfortability for the users. There are difficulties encountered with the device and the cost can be too high for it to be justified.
Static Sign Language Recognition using Deep Learning	To develop a system that serves as a learning tool for starters in sign language that uses hand detection.	Uses a predetermined skin-color range that pixels can be extracted from, a CNN model for classification of images and Keras for image training.	The results show that the system is very accurate in recognizing the hand signs that were shown.	The system is limited to static images, requires proper lighting and uniform background, and is slow in processing.
Baybayin Handwritten Letters using VGG16 Deep Convolutional Network Model	To create a system that can convert handwritten baybayin characters into their corresponding Tagalog words.	Using the VGG16 network the system will be trained to identify 45 baybayin from 1500 images.	Upon testing the system to real-time footage of a hand-written baybayin character, the test results show that the system is very accurate in its translation.	

Shape Based Continuous Real Time Hand Gesture Recognition System of American Sign Language using KNN Classifier	To create a continuous real-time hand gesture recognition system using KNN.	Uses a HOG, a feature descriptor used for digital image processing and KNN classifier, a machine learning algorithm for the pattern classification.	The study shows that the KNN classifier is very accurate in identifying the hand gestures in comparison to the other systems in its time.	The system is currently limited in identifying the ASL alphabet gestures and ASL number gestures.
A Dynamic Gesture Recognition Interface for Smart Home Control based on the Croatian Language	To create a smart home automatization system that uses dynamic gesture recognition to read gestures based on the Croatian sign language.	Uses a wake-up module and a high performance sign recognition module based on the Conv3d network.	The study shows that a dynamic gesture recognition system is very accurate and effective. This study also shows that the system has a great potential and is still open for more improvements .	The study is limited to a small database which limits the commands and gestures that can be used.

Chapter 3

METHODOLOGY

The project's development and implementation methods and procedures are discussed in this chapter. The design flow process and program algorithms were also included by the authors in this chapter.

3.1 Theoretical Framework

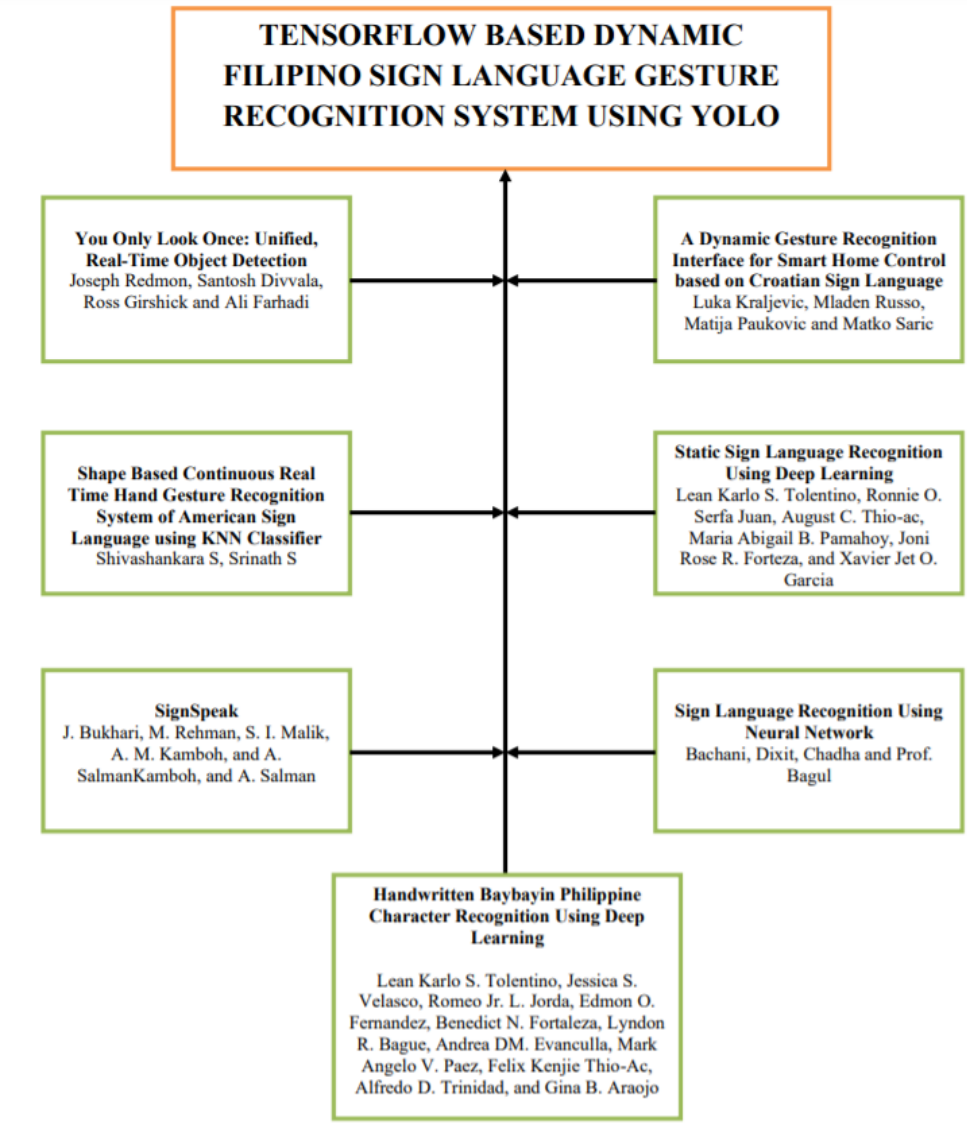


Figure 3.1 *Theoretical Framework of the Proposed Project*

Figure 1 shows the theoretical framework of the proposed project. The study takes inspiration from the previous studies of sign language gesture recognition systems which use other methods, like sensor-based gloves (SignSpeak) or visual-based neural networks (Static Sign Language Recognition using Deep Learning). The difference is the study aims to use the You Only Look Once deep learning to further increase the speed and accuracy of the translation and allow the system to be used on mobile devices.

3.2 Conceptual Framework

3.2.1 Input-Process-Output Model

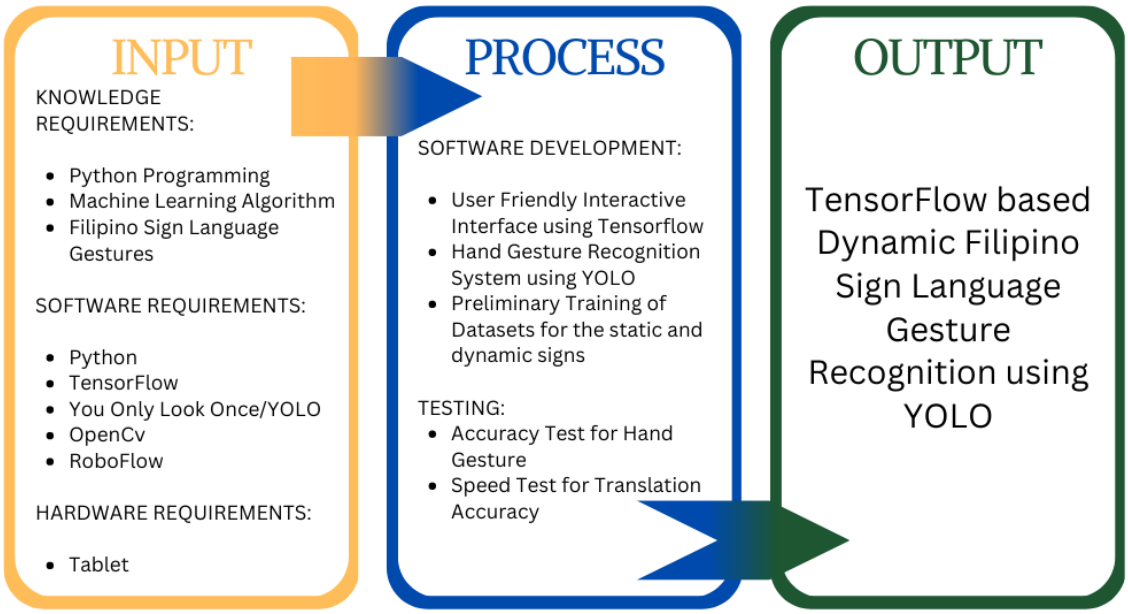


Figure 3.2 *INPUT - OUTPUT Model of the Application*

Figure 2 shows the input-output model of the application. This study focuses on developing a mobile application that uses YOLO to identify the hand gesture and translate it into text for the user to understand. YOLO is complemented in achieving this process using OpenCV which allows the system to identify the hand from the footage. The hand gesture recognition system is trained through machine learning. On the programming side, the study uses Python as the programming language to develop the mobile app with the aid of Tensorflow. The mobile application is intended to come with an FSL tutorial module to help the users who want to learn more about Filipino sign language.

3.2.2 Research Process Flow

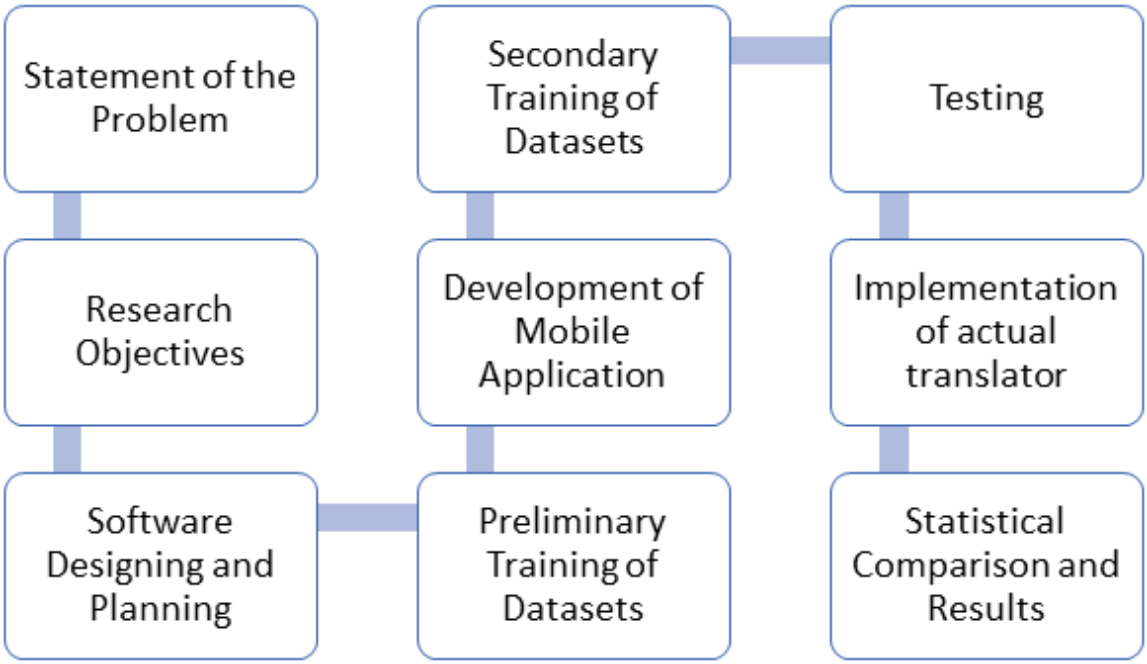


Figure 3.3 *Research Process Flow*

3.2.3 Testing Procedures

The proposed project will be having two parts of testing, the preliminary training and the secondary training. In Preliminary Training, we will be using the Desktop Computer to test the gathered datasets and it's accuracy. After gathering information about the datasets that were tested in the preliminary testing, it will be transferred to the secondary training which is the conversion of a system into a mobile application.

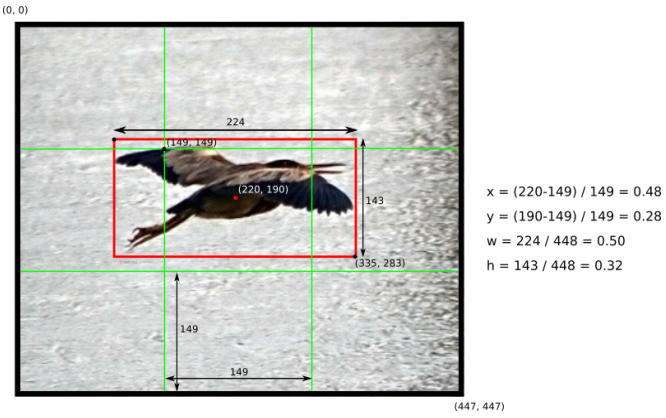
An additional module will be added to the mobile application that will be a guide for the users who want to learn the FSL aside from using the dynamic gesture translator.

3.3 How does YOLO work?

In simpler terms, images taken as input are processed by a neural network resembling a standard CNN, and the output is a vector of bounding boxes and class predictions.

A system using YOLO divides the input image into an S x S grid of cells, and for each object present in the image, the grid cell that the object's center falls into predicts the object.

Predictions for bounding boxes and class probabilities are made in each grid cell. Five elements make up a bounding box: ***x***, ***y***, ***w***, ***h***, and confidence. In relation to the position of the grid cell, the ***x*** and ***y*** components reflect the coordinates of the box's center. The ***w*** and ***h*** box measurements are located in relation to the size of the picture. The ***x***, ***y***, ***w***, and ***h*** coordinates are normalized to lie between 0 and 1. The formula for calculating confidence is ***confidence = Pr(Object) *IOU(pred,truth)***. The confidence score should be equal to the *intersection over union* (*IOU*) between the *predicted box* and the *ground truth* if there is no object in that cell; otherwise, it should be equal to zero.



(Source: HackerNoon - Understanding YOLO)

Figure 3.4 *An example of the box coordinates are calculated.*

The new formula is ***S x S x B* 5*** outputs linked to the bounding box predictions since each grid cell predicts a bounding box.

To estimate the class probabilities, ***Pr(Class(i) | Object)***, we must first anticipate the bounding box predictions. This likelihood is dependent on the grid cell having only one item. It implies that the loss function will penalize the grid cell for the incorrect class prediction if there is no item present on the grid cell. Regardless of the quantity of bounding boxes ***B***, the network can only predict one set of class probabilities per cell.

We obtain ***S x S x (B * 5 + C)*** as the result after including class probabilities into the output vector.

The network topology of YOLO seems, on *Network,* to be similar to a standard CNN, with convolutional and max pooling layers followed by 2 fully connected layers towards the very end.

On the *Loss Function*, there are 4 equations that present the loss in different areas. The first one relates to the loss to the predicted bounding box location ***(x,y)***.

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

The second relates to the loss connected to the predicted box's height/width..

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

The third, the loss relates to each bounding box predictor's confidence score.

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

And the fourth and last, the classification loss.

$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - (\hat{p}_i(c))^2$$

3.4 YOLO and its different versions

YOLO has many different versions. The first 3 versions of YOLO are created by the Joseph Redmon et al., while the other 3 are created by 3 other individuals namely, Alexey Bochkovskiy et al. - YOLOv4, Glenn Jocher - YOLOv5 and PP-YOLO by Xiang Long et al.

3.4.1 YOLOv1

- Similar to R-CNN, but is faster. It does not have a complicated process for regression.
- Can predict using the entire image. Cannot find small images.
- Predicts less background errors in comparison to Fast R-CNN.
- Input size is 224 x 224.
- Based on Darknet trained on ImageNet-1000.

3.4.2 YOLOv2

- Input size is increased to 448 x 448.
- Based on Darknet-19 which employs 5 maximum pool layers and 19 convolutional layers.
- Uses anchor boxes.
- Can find and predict small images.
- Processes images at 40-90 frames per second.
- Faster and an overall improvement compared to YOLO and sometimes even faster than YOLOv3.

3.4.3 YOLOv3

- Input size is increased and appears to be no longer limited.
- Based on Darknet-53 which now has 53 convolutional layers and for detection, another 53 layers are added, giving a total of 106 layers.
- Uses residual blocks.
- Better at YOLOv2 at small image detection.
- Even if YOLOv2 can be faster, by altering the model size, YOLOv3 may quickly trade off accuracy for speed and vice versa without the need for retraining.

3.4.4 YOLOv4

- Created by Alexey Bochkovskiy et al.
- Still based on the Darknet.

- It makes use of a number of BoF (bag of freebies) and BoS (bag of specials) to increase detector accuracy without extending inference time at the expense of training.
- It has outperformed the quickest and most accurate detectors in terms of both accuracy and speed, obtaining a 43.5 percent in AP value on the COCO dataset and a real-time speed of 65 frames per second on the Tesla V100.
- In comparison to YOLOv3, YOLOv4 shows an increase in AP value by 10 percent and increase in FPS by 12 percent.

3.4.5 YOLOv5

- developed by Ultralytics, the same firm that created the Pytorch version of YOLOv3, announced their most recent object detection in June 2020.
- In comparison to the entire YOLOv3 model, it produces results with around 75% less processes.
- It benefits from the PyTorch ecosystem because it was built using PyTorch, making deployment and support simpler.
- They observed inference speeds up to 0.007 seconds per image in the YOLOv5 Collab notebook running a Tesla P100, translating to 140 frames per second (FPS), as opposed to the YOLOv4 which only manages 50 FPS after being ported to the same Ultralytics PyTorch library.

3.4.6 PP-YOLO

- Introduced by Xiang Long et al. in July 2020.
- Based on Parallel Distributed Deep Learning (PaddlePaddle).

- Same as YOLOv5, PP-YOLO is based on the YOLOv3 model.
- The notable changes include the replacement of Darknet53 with ResNet and increase of batch training size from 64 to 192.
- The report claims that when evaluated on a V100 with batch size = 1, PP-YOLO can reach a mAP of 45 percent COCO datasets and an inference speed of 72.9 FPS. This demonstrates that it outperforms YOLOv4 in mAP by 43.5 percent and in inference speed by 65 FPS.

3.5 YOLO vs Other Machine Learning

There are numerous other object detection approaches that can be used , so why choose YOLO over the others? The information below detailed two commonly used object detection algorithms to show the reason why YOLO is the better choice currently.

3.5.1 Fast R-CNN

- Introduced in 2015.
- Fast R-CNN is an enhanced version of the R-CNN with the goal of enhancing the R-CNN's image processing.
- Three separate models were combined into a single training framework and their shared computational outputs. In particular, each picture now utilizes a CNN forward channel and shares the feature matrix rather than trainers setting separate feature vectors for each feature. Both the classifier and the border regression matrix are built using the same eigen matrix.
- Many of the steps of R-CNN are shared with Fast R-CNN, in addition to the fact that Fast R-CNN has additional processes.

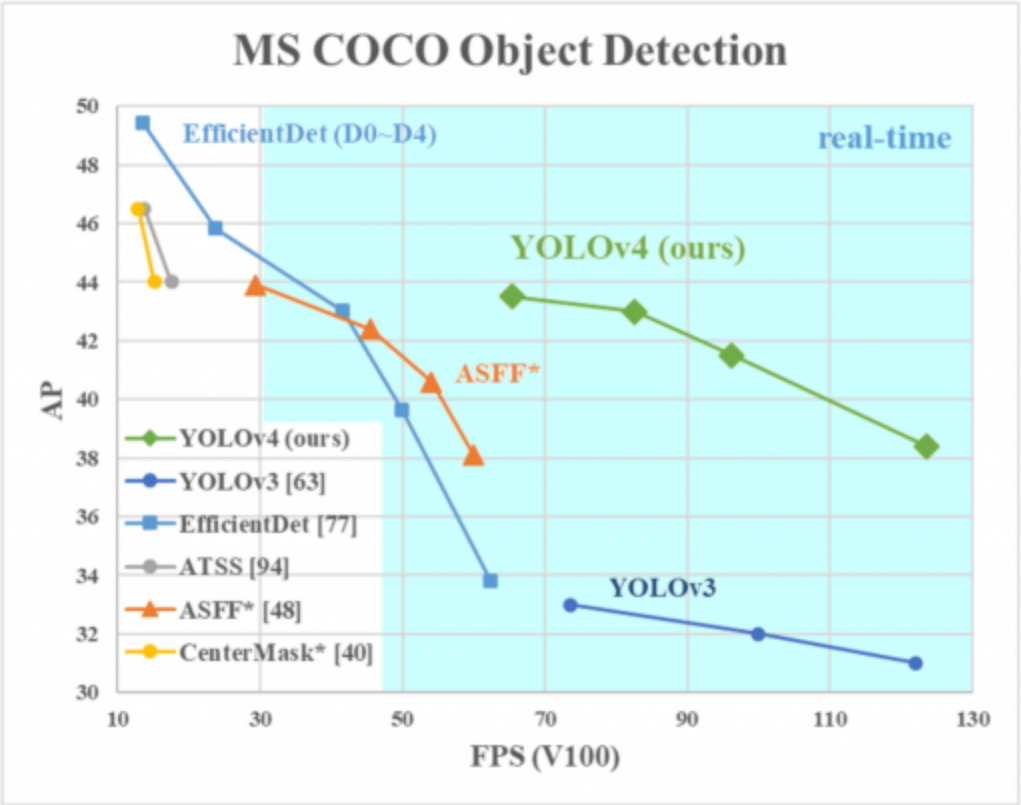
- Fast R-CNN achieved an improvement over the original R-CNN, but trade it off by being time-consuming in the process

3.5.2 Single Shot Multibox Detector (SSD)

- Perform its operations at a much higher speed than YOLO and Fast R-CNN.
- SSD employs a free-forward convolution layer technique that generates a fixed-sized collection of bounding boxes, scores each instance of an object class that is found within those bounding boxes, and also employs a Non-Max Suppression to generate the results.
- The SSD architecture is really straightforward.
- The disadvantage of SSD is that it has worse performance than Fast R-CNN when it comes to small object detection.
- Another drawback is that, because of its intricate data augmentation, SSD requires a lot of data for training, which, depending on the application, may be expensive and time-consuming.

The SSD and Fast R-CNN are tested and compared to a YOLOv3, the tests from different studies like Comparative Analysis of Deep Learning Image Detection Algorithms (Srivastava et al., 2021) showed that YOLOv3's overall performance surpassed the two algorithms used.

In addition to this, the latest YOLOv4 is tested using COCO datasets against other object detection approaches. The result is shown in the figure shown below.



(Source: YOLOv4 Paper)

Figure 3.5 *YOLOv4 comparison with other Object Detection using COCO datasets.*

Figure 3.4 showcases the improvement of YOLOv4 from YOLOv3 which shows the increase in AP without suffering the FPS. It also highlights the FPS of YOLOv4 against its competition in real-time object detection.

3.6 Conceptualization of the Application that will be used to Implement the Translator

When using the sign language translator function, the application will be using OpenCV to locate the hand from the rest of the camera footage. Once the hand is detected, YOLO will identify the hand gesture and translate it by matching it with the datasets. After training, the translation should appear as text on the tablet screen.

Other than the translator function, there is also the module which teaches the user basic words and phrases in Filipino sign language. There is also an optional voice-to-text feature that simply converts the user's words and sentences into text on the screen to allow the user to answer back if it needs to be.

3.7 FSL101: Filipino Dynamic Gestures Recognition Module

(Source: VERA Files - Filipino Sign Language)

Figure 3.6 *An image of a man performing the "NO" word in Filipino Sign Language*

The application contains a module for basic tutorials to those who want to learn about sign language fully. The intended feature contains common responses, like yes or no, and short phrases like, hello. The feature also contains the FSL alphabet and numbers.

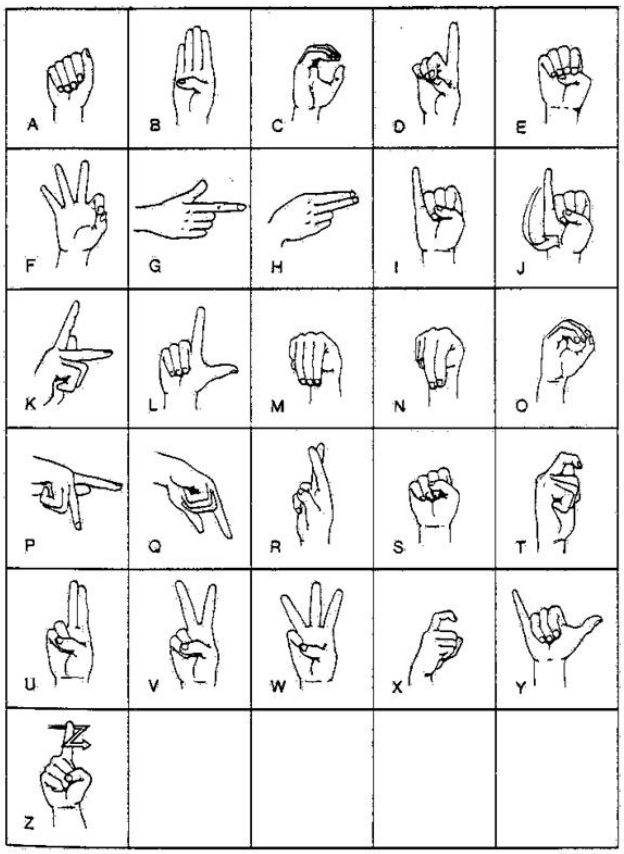


Figure 3.7 *Filipino Sign Language Alphabet*

3.8 Design of Overall System

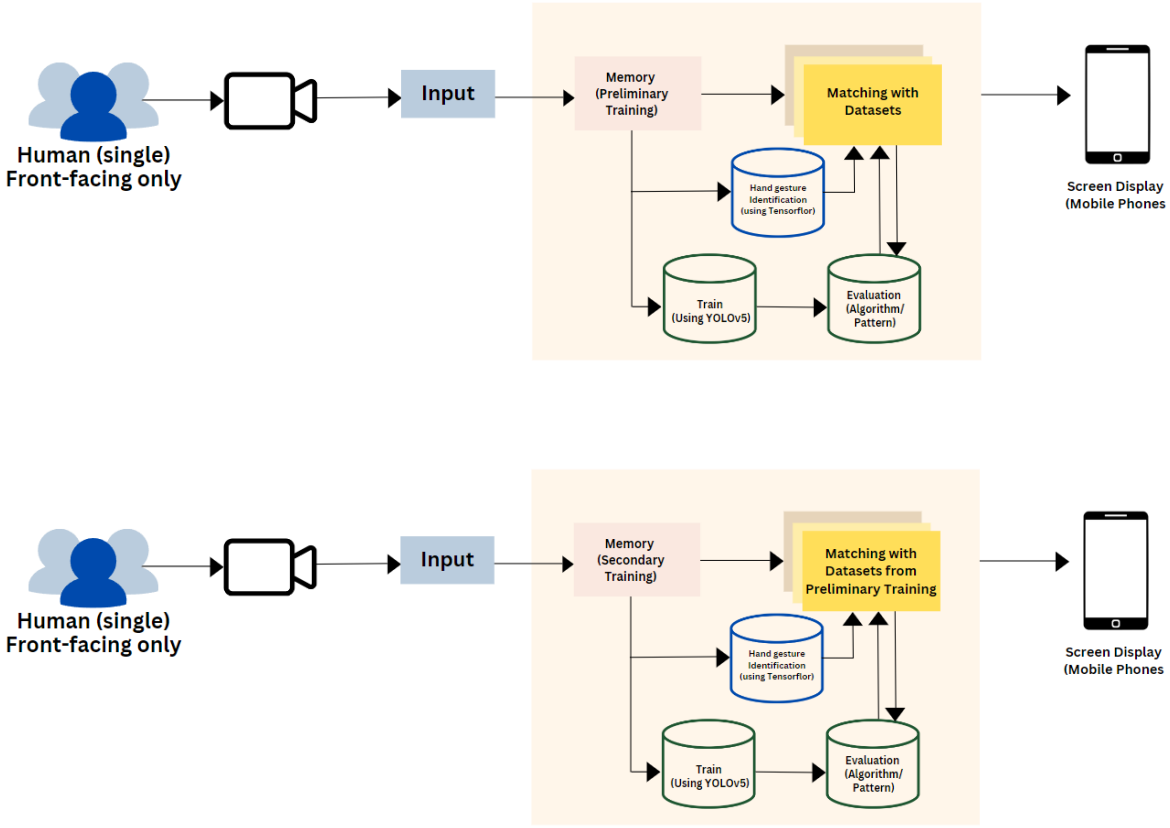


Figure 3.8 *Block Diagram of the Proposed Application during the Preliminary Training and the Secondary Training*

Figure 3.2 shows the proposed project's block diagram shows the preliminary training of the hand gesture recognition system and its secondary training.

The diagram shows that under the preliminary training, the system is trained using a computer set-up. Using the computer, the system is trained to identify the hand from an active camera footage using OpenCV. Once the hand is separated from the rest of the footage, the system is then trained using YOLO to

identify the gesture of the hand that is being presented by matching the gesture with the datasets from the memory. The outcome of the identification is then projected to a monitor for evaluation and correction of data.

Once the preliminary training is finished and the hand sign identification system works as intended, the hand sign identification system is then prepared to be integrated into an application for mobile devices, like tablets, which can be seen in the second block diagram.

On the second block diagram, the application undergoes the same training the computer did with an additional set of dynamic gestures. The dynamic gesture includes simple phrases that we use in everyday life.

3.9 Flowchart of the Application

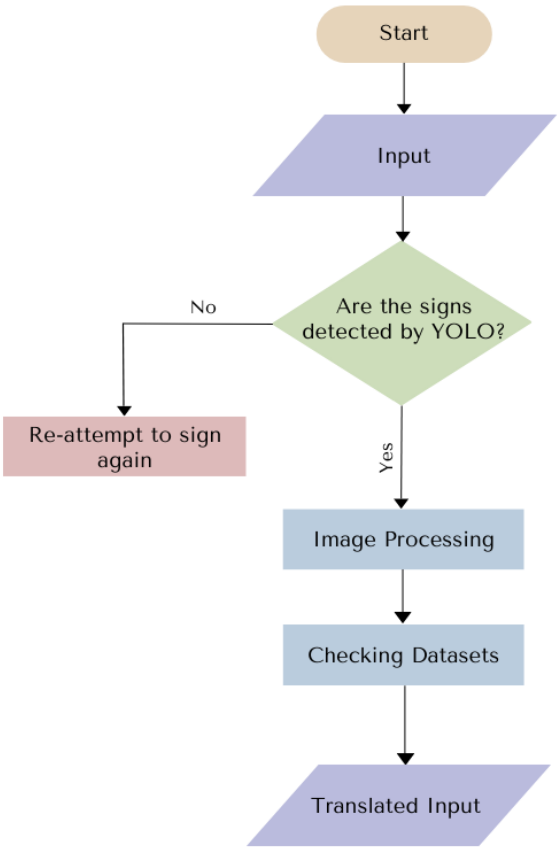


Figure 3.9 *Flow chart of the system*

3.11 Applying the ISO 9126 model to the evaluation of the system

An effective framework for evaluating software is created with the aid of the international standard ISO 9126 software. This common method of evaluating software may be divided into four (4) separate categories. These are employed to handle issues of various kinds. This program is widely and effectively used to accept several concepts and metrics. The suggested characteristics explain outwardly when software is discovered to be a result of internal software properties.

The Quality Model, External Metrics, Internal Metrics, and Quality in Use Metrics are the four categories that were mentioned.

It is possible to develop a mobile application that adheres to the universal model and makes it simpler to compare products by utilizing the four distinct categories of evaluating software quality.

Table 3.1 Testing of the System based on the ISO 9126 standards

Characteristic	Sub-characteristic	Description
Functionality	Suitability (F1)	Can the system carry out the necessary tasks?
	Accurateness (F2)	Are the system's outcomes what were predicted?
	Interoperability (F3)	Does the system respond to other systems?
	Security (F4)	Does the system guard against unauthorized entry?
Reliability	Maturity (R1)	Have the system's and the hardware's flaws been fixed over time?
	Fault tolerance (R2)	Does the system have the ability to handle errors?
	Recoverability (R3)	In the event of a failure, can the system function again and retrieve any lost data?
	Recoverability compliance (R4)	Does the system follow the reliability and quality criteria in place?
Usability	Understandability (U1)	Does the system user understand how to utilize it without any difficulty?
	Learnability (U2)	Is it possible to learn the

		system quickly?
	Operability (U3)	Can the system function with little effort?
	Attractiveness (U4)	Is the system well-designed?
	Usability compliance (U5)	Does the system adhere to current usability guidelines?
Efficiency	Time behavior (E1)	How fast is the system reacting?
	Resource utilization (E2)	Does the system make good use of its resources?
	Efficiency compliance (E3)	Does the system meet the current efficiency requirements?
Maintainability	Analyzability (M1)	Can the flaws be identified with ease?
	Changeability (M2)	Can the system be changed easily?
	Stability (M3)	Can the system still work once changes are made?
	Testability (M4)	Is it simple to test the system?
Portability	Adaptability (P1)	Can the system be transferred to different settings?
	Installability (P2)	Can the system be simply installed?
	Portability compliance (P3)	Does the system adhere to portability requirements?

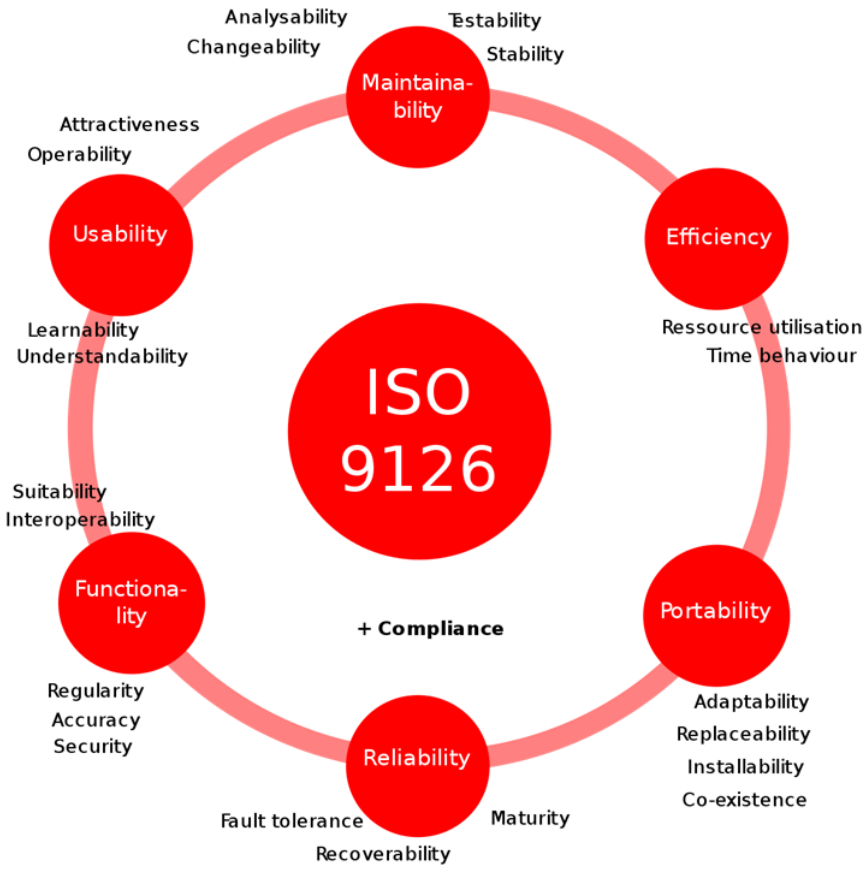
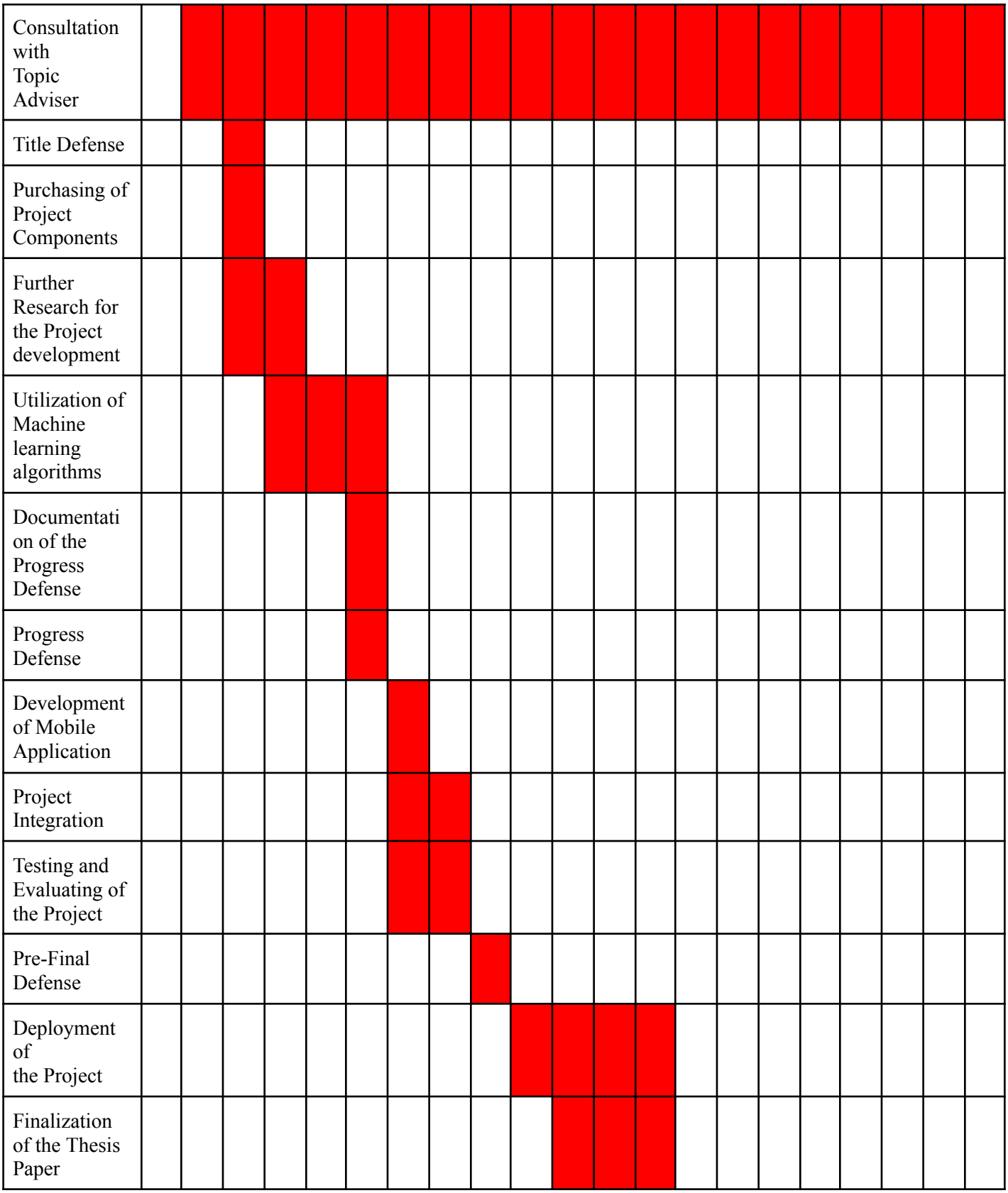


Figure 3.10 *Flow chart of the system*

3.12 Gantt Chart

	2021									2022										
Activities Description	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sept	Oct	Nov	Dec
Preparation of Chapter 1to 3																					
Topic Defense																					


Consultation with Topic Adviser
Title Defense
Purchasing of Project Components
Further Research for the Project development
Utilization of Machine learning algorithms
Documentation of the Progress Defense
Progress Defense
Development of Mobile Application
Project Integration
Testing and Evaluating of the Project
Pre-Final Defense
Deployment of the Project
Finalization of the Thesis Paper

Final Defense																				
Finalization of the Project Document and Bookbinding																				
Submission of Research Book																				

Table 3.2 *Gannt Chart for the Entire Study*

Chapter 4

PRESENTATION, ANALYSIS AND DISCUSSION OF DATA

The technical description, structural organization, interpretation of the data, and consequences of the study are presented in this chapter.

4.1 Technical Description of the Project

The technology developed for the project entitled “TensorFlow based Dynamic Filipino Sign Language Gesture Recognition using YOLO” aims to detect static and dynamic Filipino Sign Language Gestures that will translate into words and phrases. The detection uses a fast R-CNN called YOLO or You Only Look Once which is trained to convert the signs. The Mobile Application is called Pantomime and was built using Python, Android Studio and TensorFlow.

The users will also be able to access the tutorial or the learning modules attached in the application which can serve as reference and guide in performing sign languages. Inside the manual, it contains the basic words, phrases, alphabet, and numbers.

4.2 Project Structure

4.2.1 Preliminary Training of Datasets

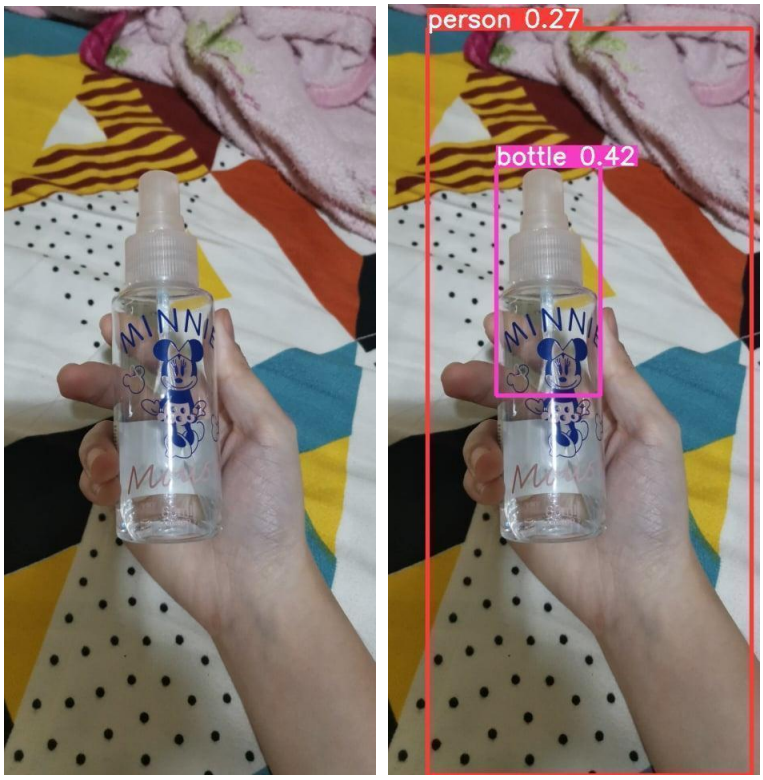
4.2.1.1 Utilizing YOLO’s Object Detection


```
Fusing layers...
Model Summary: 224 layers, 7266973 parameters, 0 gradients
Image 1/1 C:\Users\User\Desktop\yolov5-master\ron2.jpg: 640x352 1 person, 1 cell phone, Done. (0.037s)
Speed: 2.0ms pre-process, 37.0ms inference, 25.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp3
```

Figure 4.1 *Test Image Results of the Python Script using YOLO and PyCOCO datasets and the original images in a hand holding a phone*

Figure 4.1 shows the results of the object detection and image recognition of the program with the use of YOLO and PyCOCO datasets. The first pair of images shows a hand holding a phone. The program creates boxes around the objects it identifies which, in the case of the first image, is the phone and the person holding it. The numerical value next to the guess represents the program's level of confidence in that guess.

The illustration displays the first pair of the results summary. It also displays the processing time, which was 0.037 seconds.

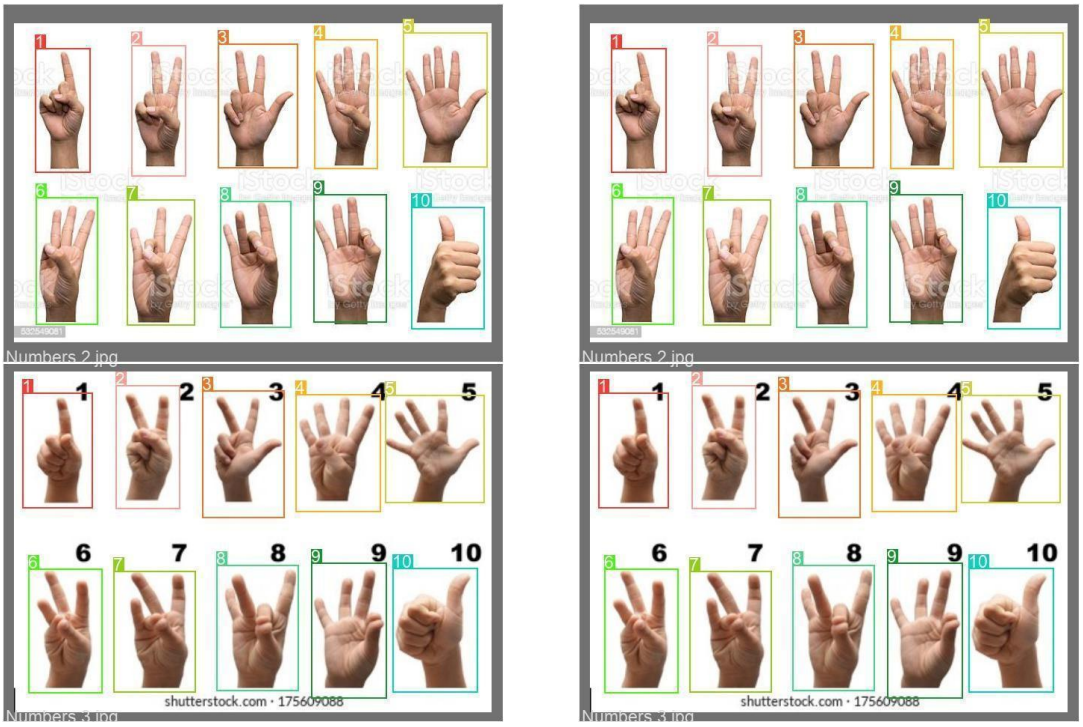



```
Fusing layers...
Model Summary: 224 layers, 7266973 parameters, 0 gradients
image 1/1 C:\Users\User\Desktop\yolov5-master\ella.jpg: 640x320 1 person 1 bottle Done. (0.031s)
Speed: 0.0ms pre-process, 31.2ms inference, 0.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp4
```

Figure 4.2 *Test Image Results of the Python Script using YOLO and PyCOCO datasets and the original images in a hand holding a bottle*

The second pair of images shows a hand holding a bottle. The program does the same method where it guesses the objects in the photo. The results show that the program guessed that there is 1 person and 1 bottle, which is correct.

4.2.1.2 Applying YOLO in the Static Datasets



Epoch	gpu_mem	box	obj	cls	labels	img_size
535/599	1.07G	0.042	0.05242	0.05961	61	640: 100% 1/1 [00:00<00:00, 2.43it/s]

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95: 100% 1/1 [00:00<00:00, 9.1
all	2	20	0.276	0.548	0.321	0.228
1	2	2	0.144	0.5	0.199	0.119
2	2	2	1	0	0.249	0.181
3	2	2	0.11	1	0.153	0.0986
4	2	2	0.294	0.5	0.314	0.251
5	2	2	0.191	1	0.995	0.746
6	2	2	0	0	0.117	0.0772
7	2	2	0.318	0.477	0.332	0.217
8	2	2	0.194	0.5	0.331	0.203
9	2	2	0.176	1	0.276	0.227
10	2	2	0.33	0.5	0.242	0.155

lyStopping patience 100 exceeded, stopping training.

Figure 4.3 *Image Results of the First Round Test using a Custom dataset*

Figure 4.3 shows the image results of the first round of tests using the custom datasets which are the hand signs of 1 to 10.

33 The image to the left shows the expected result of the test and the image to the right shows the outcome of the test. The only hand sign recognized by the program is the hand

sign of number 5. This supported by the image showing that the 99.5% of the runs recognized the hand sin whilst the others maxed mostly around 30%.

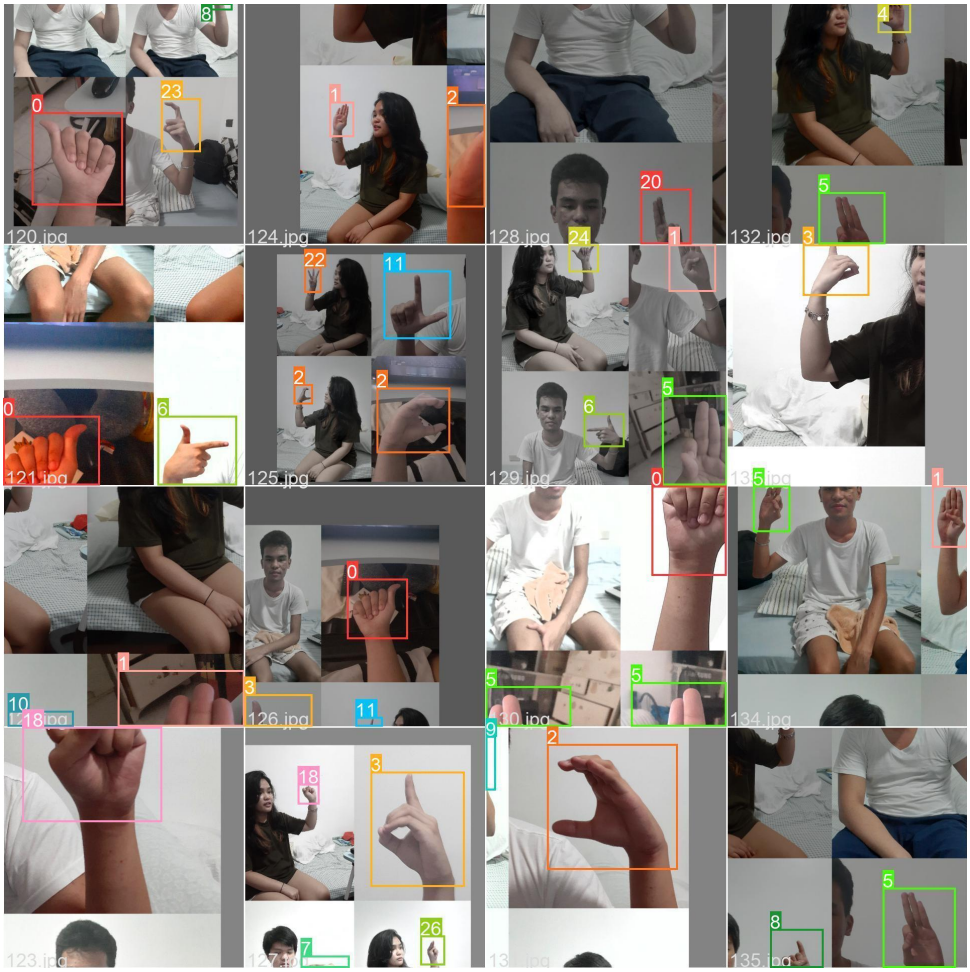
The first run was performed by a small amount of dataset which could be the main reason for the low results.

Table 4.1: Letter Recognition Accuracy

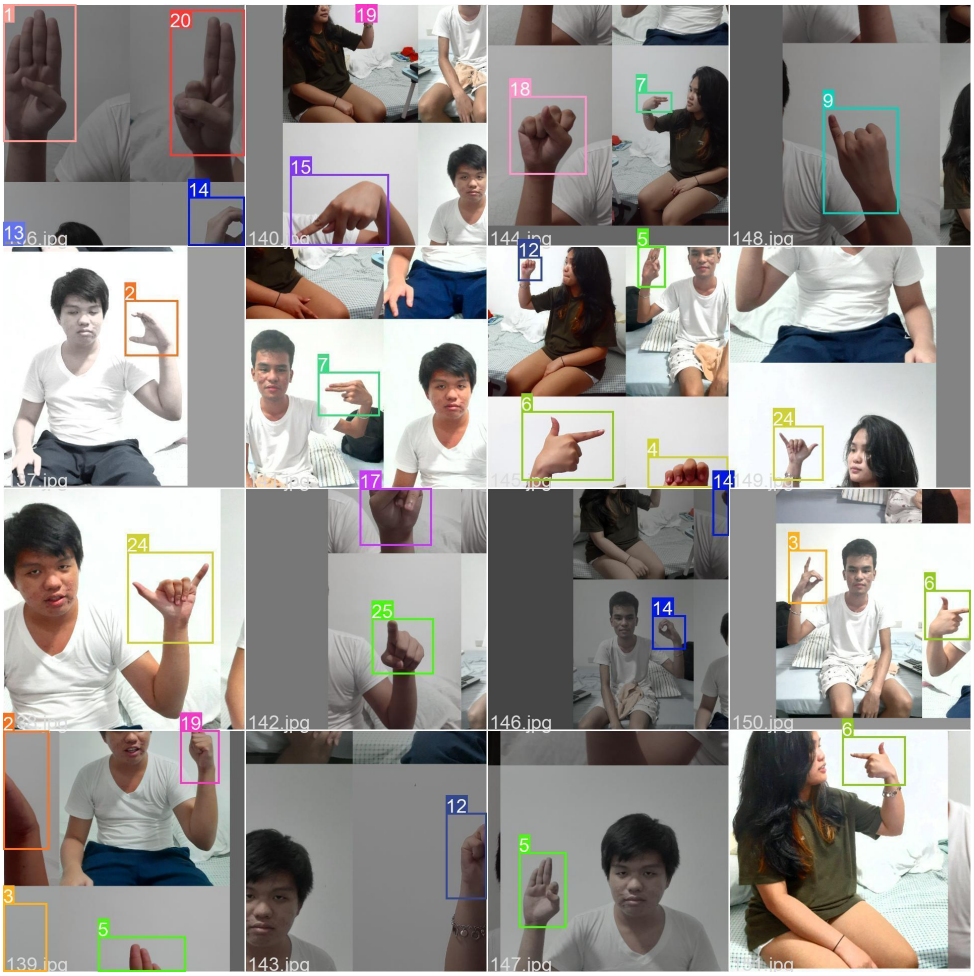
Letter	Preliminary Training of Tensorflow based Dynamic Filipino Sign Language Gesture Recognition using YOLO		Compared to the Previous Study: Static Sign Language Recognition Using Deep Learning	
	Accuracy (%)	Average Time (s)	Accuracy (%)	Average Time (s)
A	97	0.027	100	2.02
B	97	0.027	91.11	4.01
C	97.1	0.028	100	2.2
D	96.9	0.027	100	2.46
E	97.7	0.027	96.67	3.59
F	96.3	0.028	93.33	4.95
G	96	0.027	98.89	2.85
H	97.5	0.027	84.44	5.39
I	96.5	0.027	94.44	3.37
J	98	0.028	90	4.57
K	97.1	0.027	94.44	4.06
L	97	0.028	98.89	2.38
M	96.3	0.027	82.22	5.98
N	98.2	0.027	90	3.92
O	99	0.028	90	3.62

P	96.5	0.027	86.67	4.97
Q	100	0.027	95.56	3.76
R	96.3	0.027	78.89	6.24
S	99.1	0.028	86.67	4.6
T	100	0.027	74.44	7.62
U	100	0.027	85.56	5.11
V	99.6	0.027	84.44	5.07
W	96.9	0.027	96.67	2.46
X	97.7	0.027	86.67	4.98
Y	97.2	0.027	93.33	3.5
Z	96.4	0.027	67.78	8.31
Ñ	96.6	0.027	-	-
Overall Rating	97.6%	0.02722	90.04%	4.31

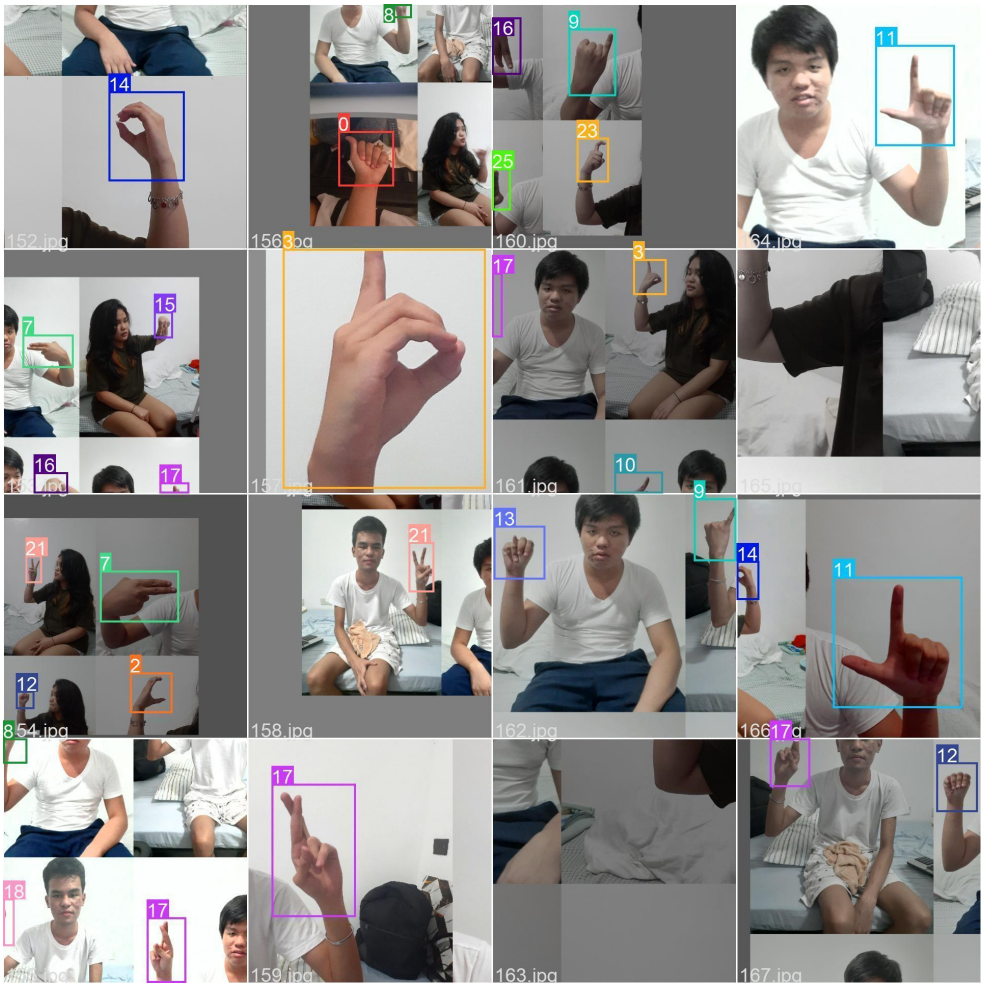
Table 4.1 shows the accuracy rate of each letter and its corresponding average time and was compared to the previous study with the same criteria



(a)


1
20
13
14
19
15
18
7
9
2
7
12
5
6
4
24
17
25
14
24
19
3
5
12
5
6
142.jpg
146.jpg
150.jpg
139.jpg
143.jpg
147.jpg

(b)

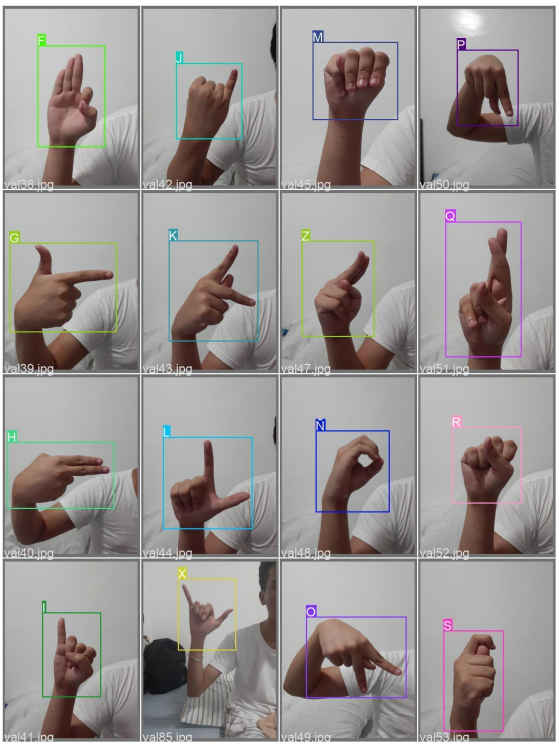


(c)

Figure 4.4 *The collected letter dataset on which the system will be based (a) batch 0 (b) batch 1 (c) batch 2 of the database*

(a)

(b)



(c)

(d)

Figure 4.5 *Image Results of the Second Round Test Using the Collected Static Dataset (a) dataset that was identified as a hand sign from the Figure 4.4 (b) Validation result from the system (c) dataset that was identified as a hand sign from the Figure 4 (d) Validation result from the system*

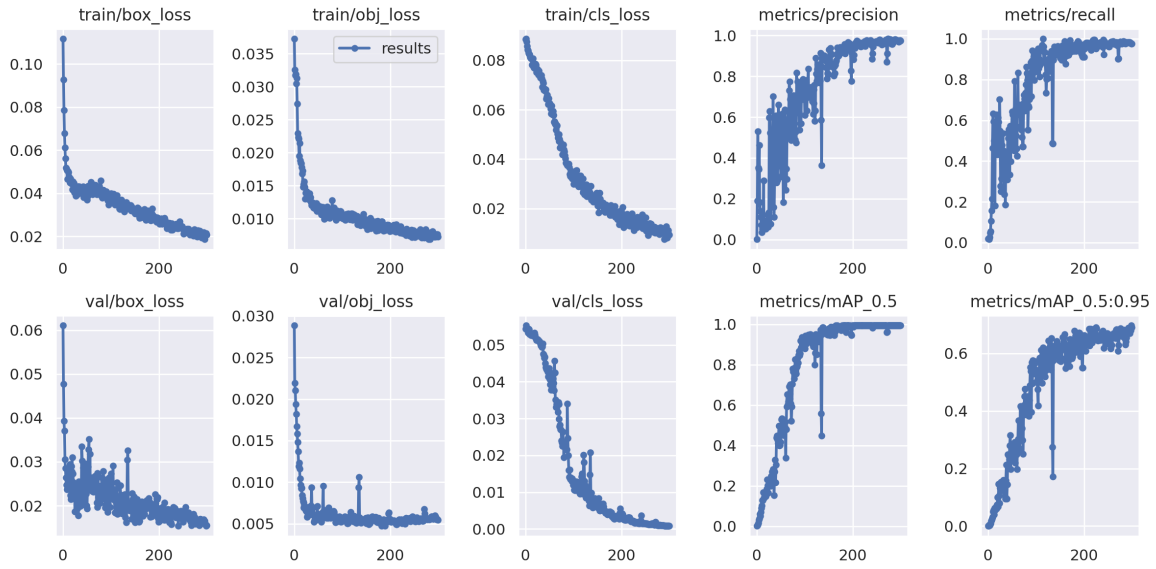


Figure 4.6 *Plots of box loss, object_loss, classification loss, precision, recall, and mean average precision (mAP) over the training dataset and validation set*

In Figure 4.6, it is possible to comprehend how the respective losses are evolving with each iteration by looking at the graph's lower box_loss and object_loss values. The box loss measures how accurately the algorithm can pinpoint an item's center and how completely the anticipated bounding box encloses an object. The reverse of objectness, Object_loss, quantifies the likelihood that an object will exist inside a suggested zone of interest. How successfully the algorithm can determine the proper class of a given item is indicated by Classification_loss.

Precision, mean average location, and other metrics for the model are excellent. The validation data's box, object_loss, and classification losses all quickly improved.

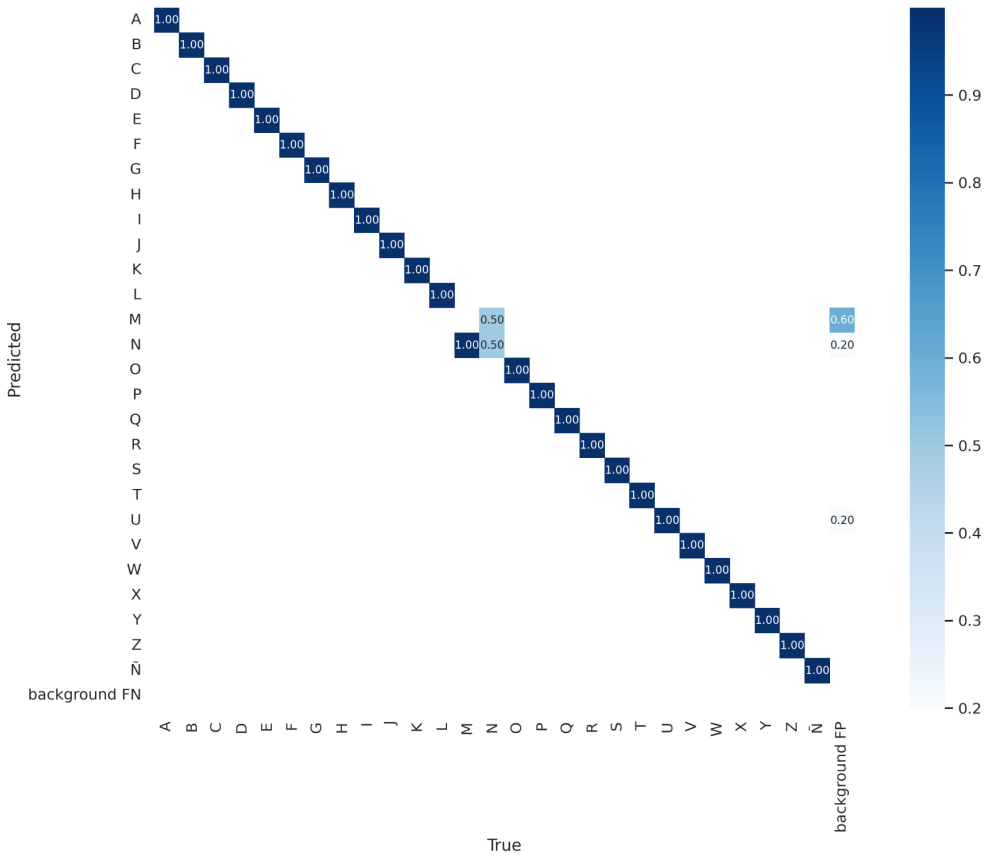
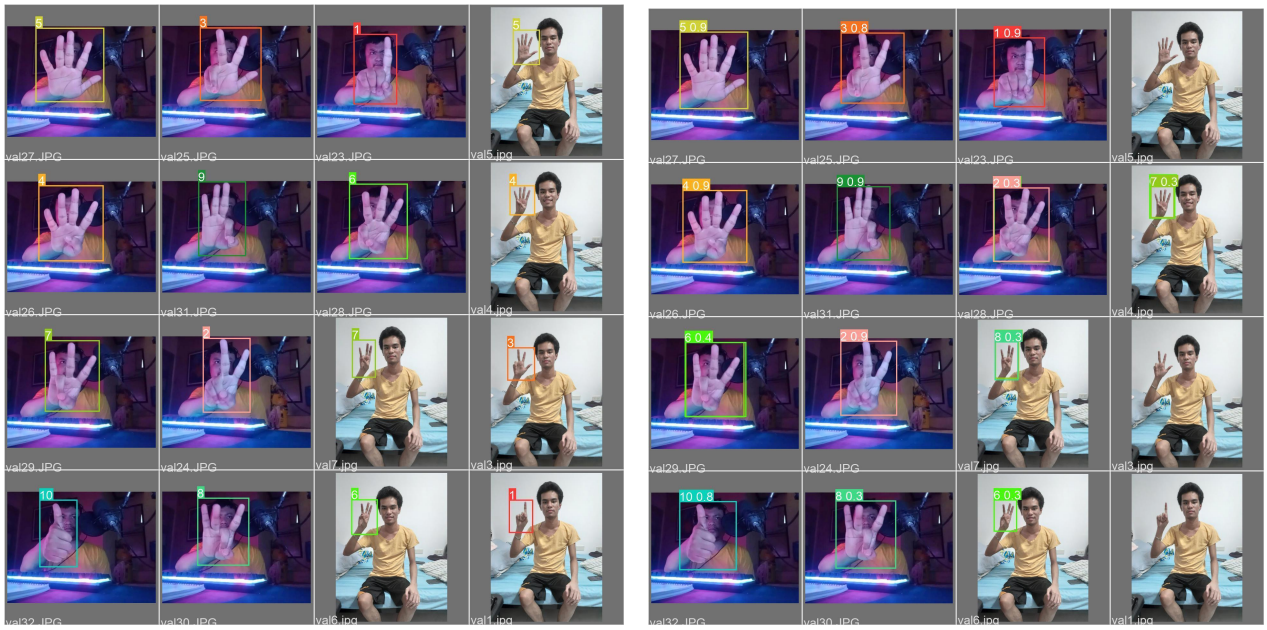


Figure 4.7 *Confusion Matrix of the Static Letters*

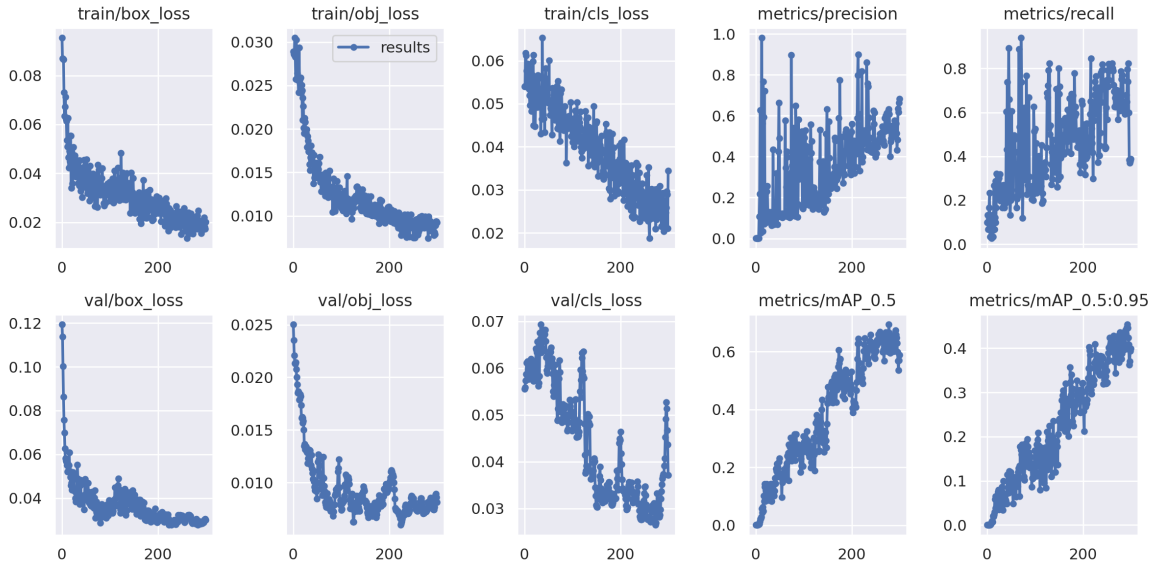
Figure 4.8 *Image of the footage captured for the Learning Module*

Figure 4.8 shows the image of the footage and data concept for the development of the Learning Module aiming to learn the different hand signals. It will be integrated into the Mobile Application.

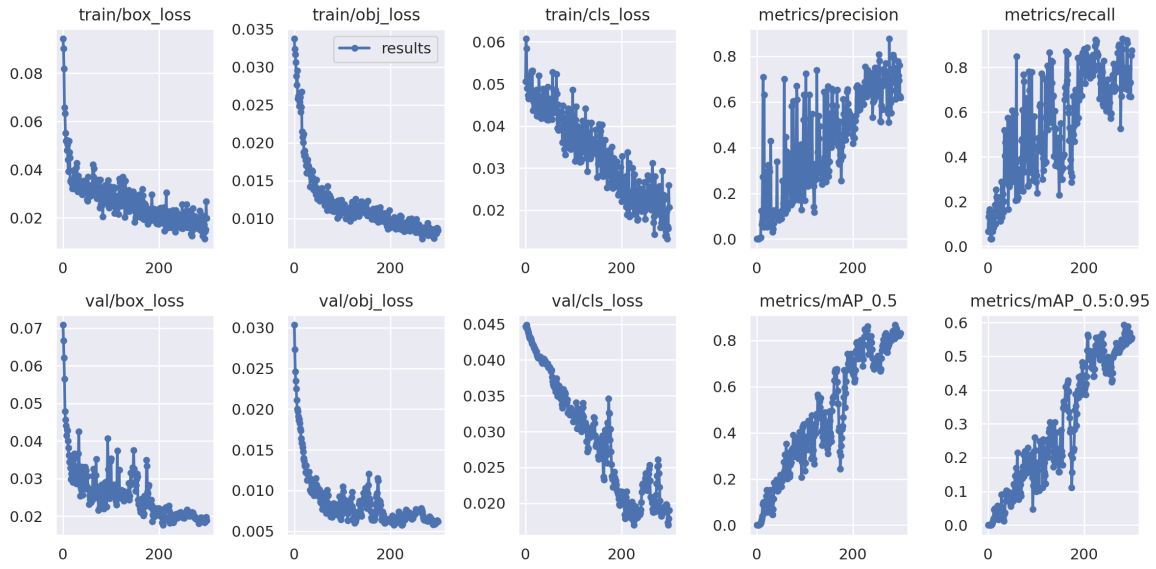


(a) (b)

Figure 4.9 *Image Results of the Second Round Test Using the Collected Static Number Dataset (a) dataset that was identified as a hand sign from the Figure 4.4 (b) Validation result from the system*



(a)



(b)

Figure 4.10 *Plots of box loss, object_loss, classification loss, precision, recall, and mean average precision (mAP) over the training dataset and validation set (a) before (b) after*

4.2.2 Creating Mobile Application

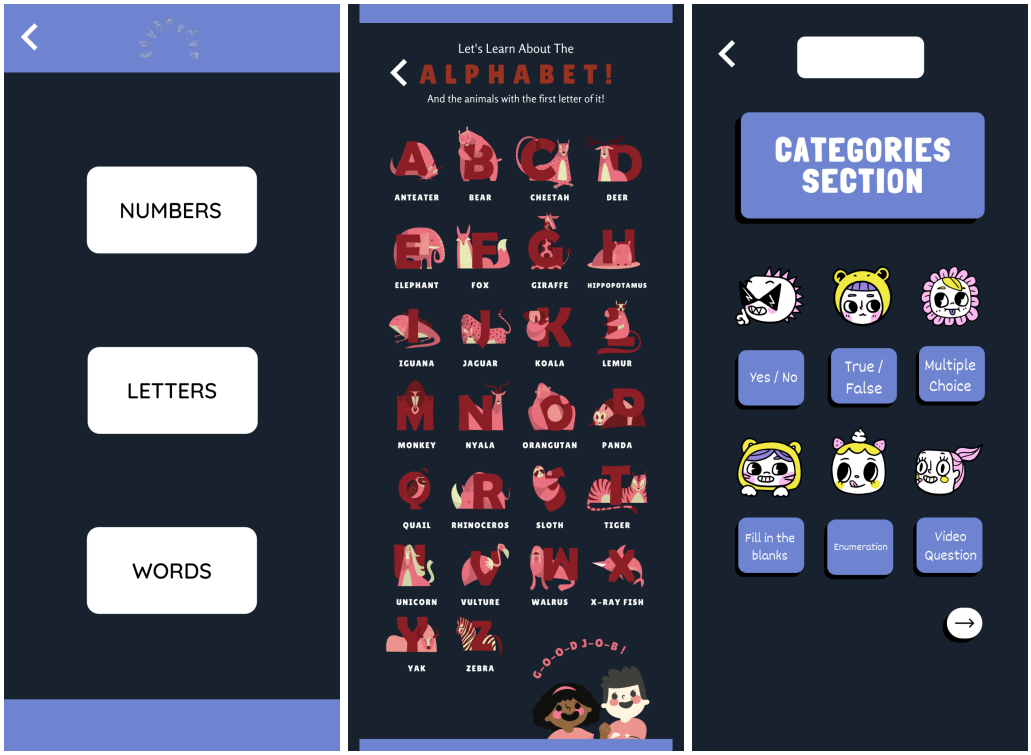
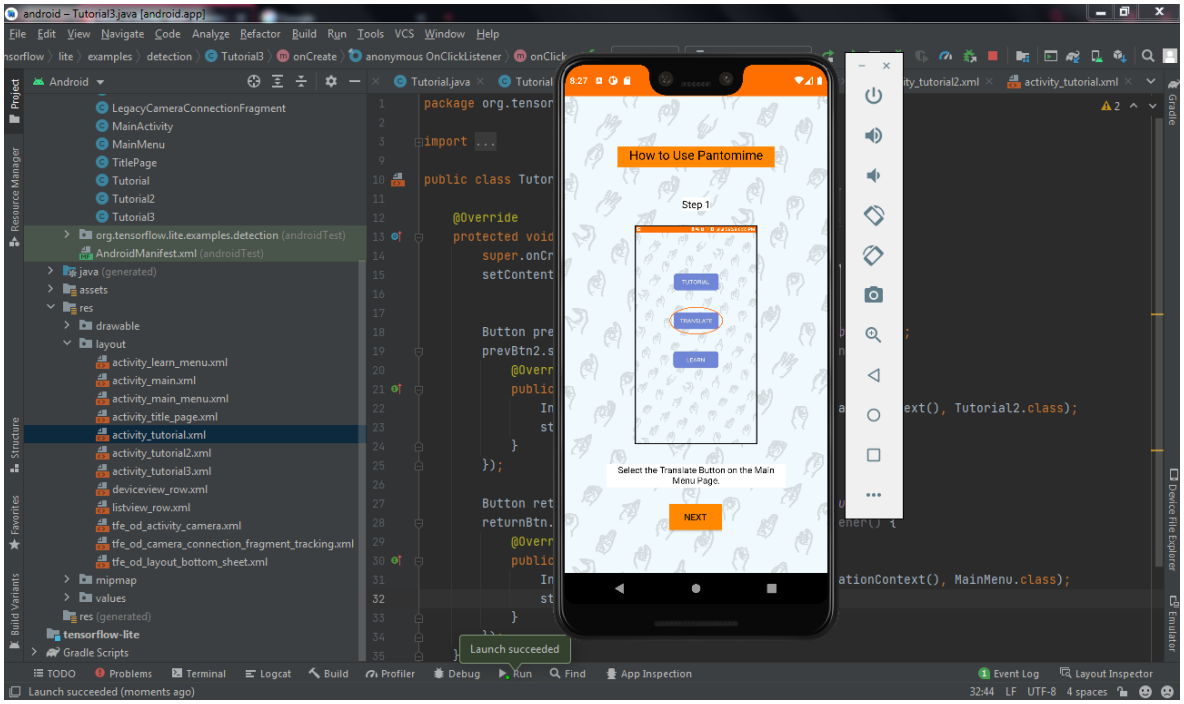
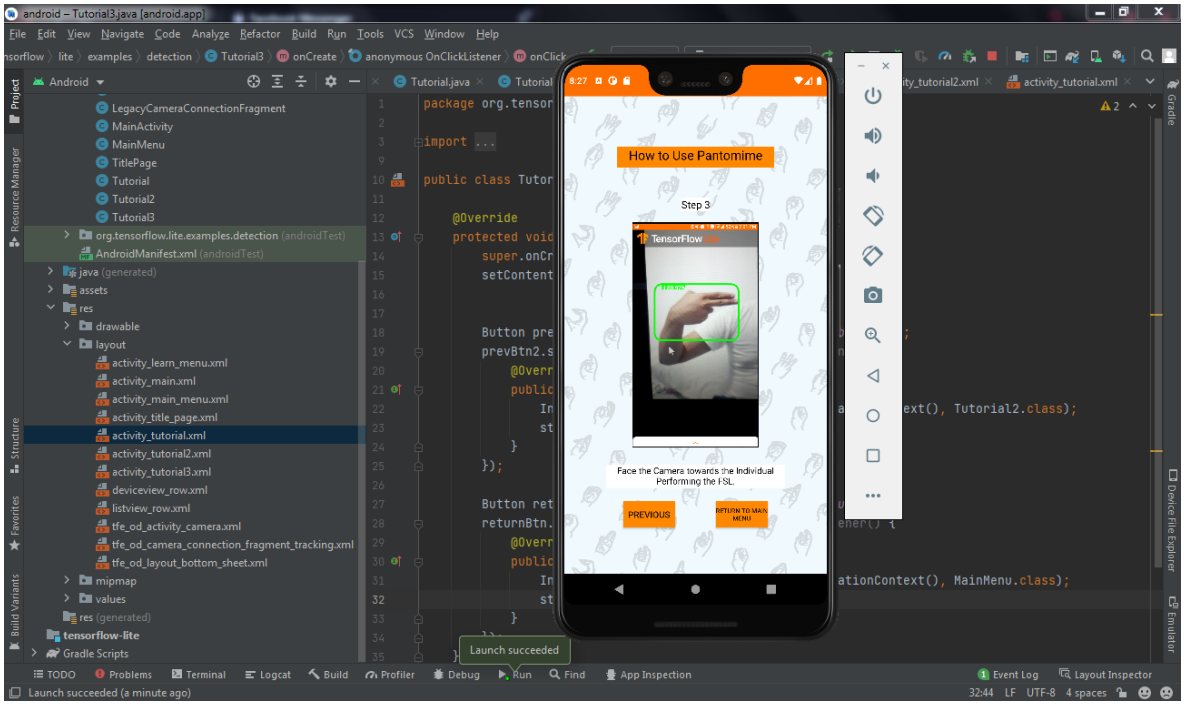


Figure 4.11 *Design Concept of the Learning Module into the Mobile Application*


How to Use Pantomime
Step 1
Select the Translate Button on the Main Menu Page.
NEXT
Launch succeeded


How to Use Pantomime
Step 3
Face the Camera towards the Individual Performing the FSL
PREVIOUS
RETURN TO MAIN MENU
Launch succeeded

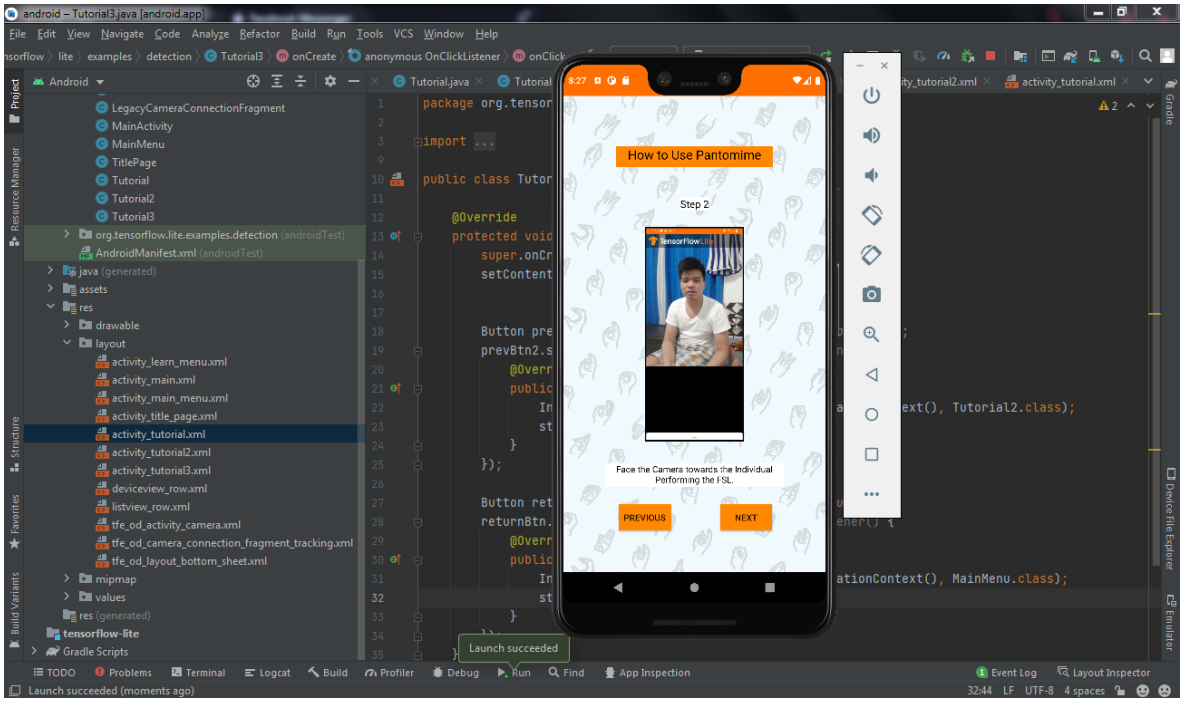


Figure 4.12 *Image of Integrating the Design Concept of the User Interface into Android Application with Tutorial for Usage*

Figure 4.12 shows the image of testing and integrating the design concept of the user interface along with the trained datasets. Included the tutorial on how to use the application. TensorFlow, Python, and Android Studio would be used to design the final interface.

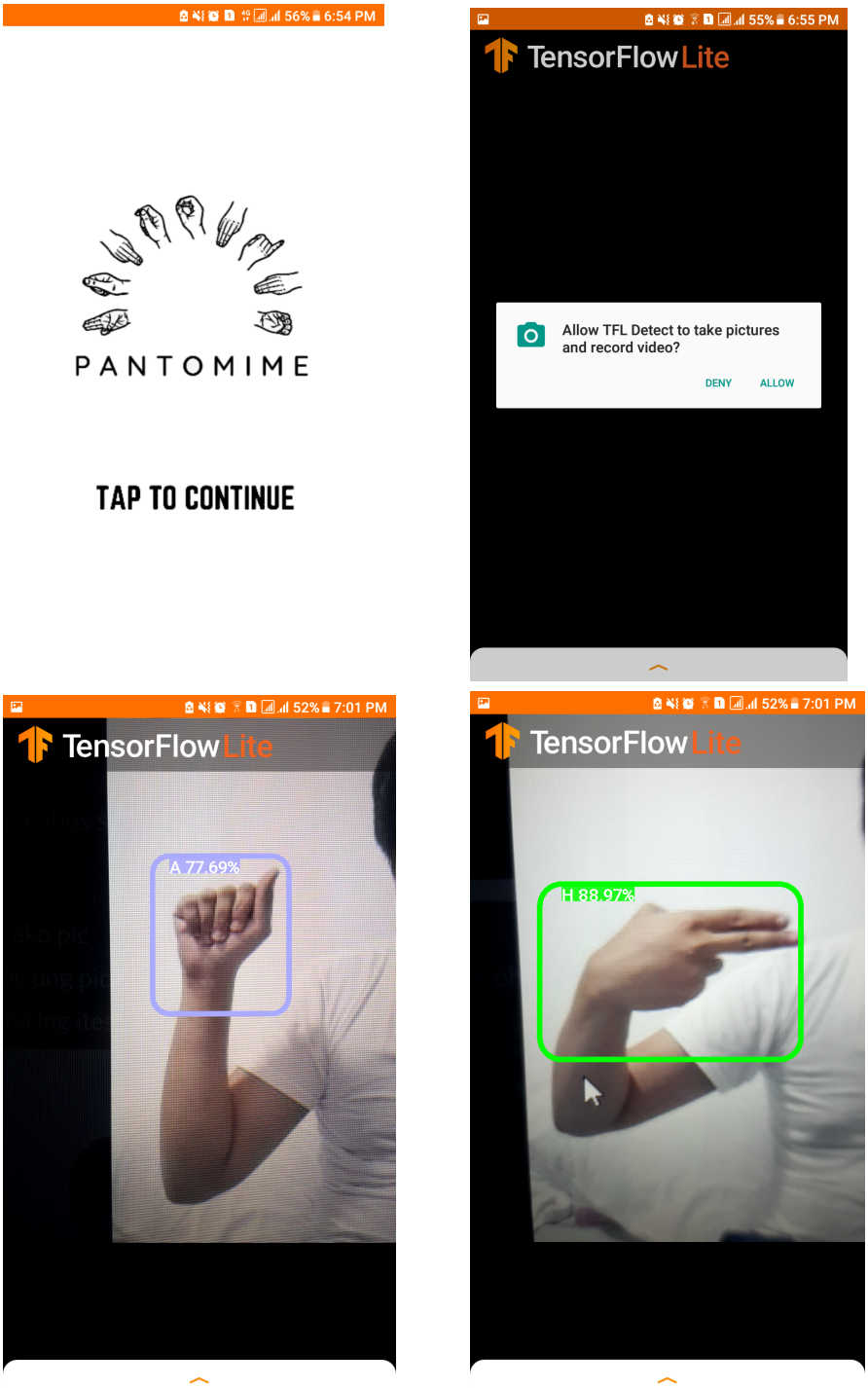
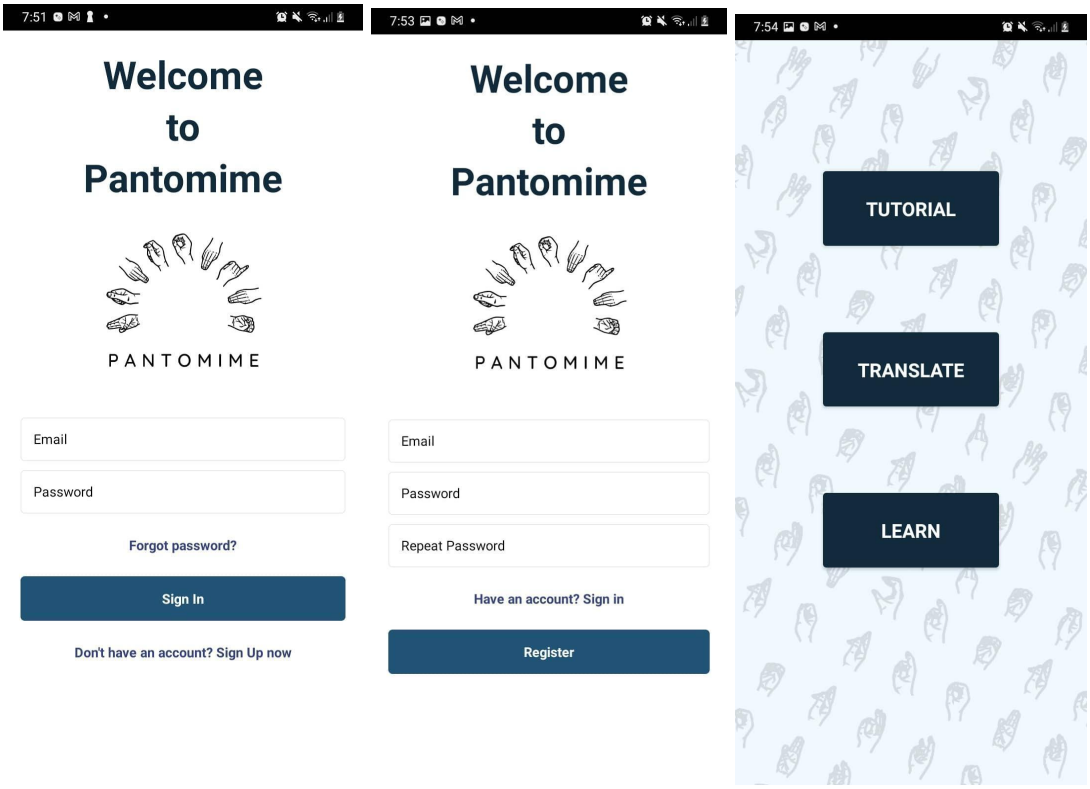


Figure 4.13 *Images of the camera and detector test using an actual android device of one of the researchers*

Figure 4.13 shows the first actual android application interface with the researcher's hand.

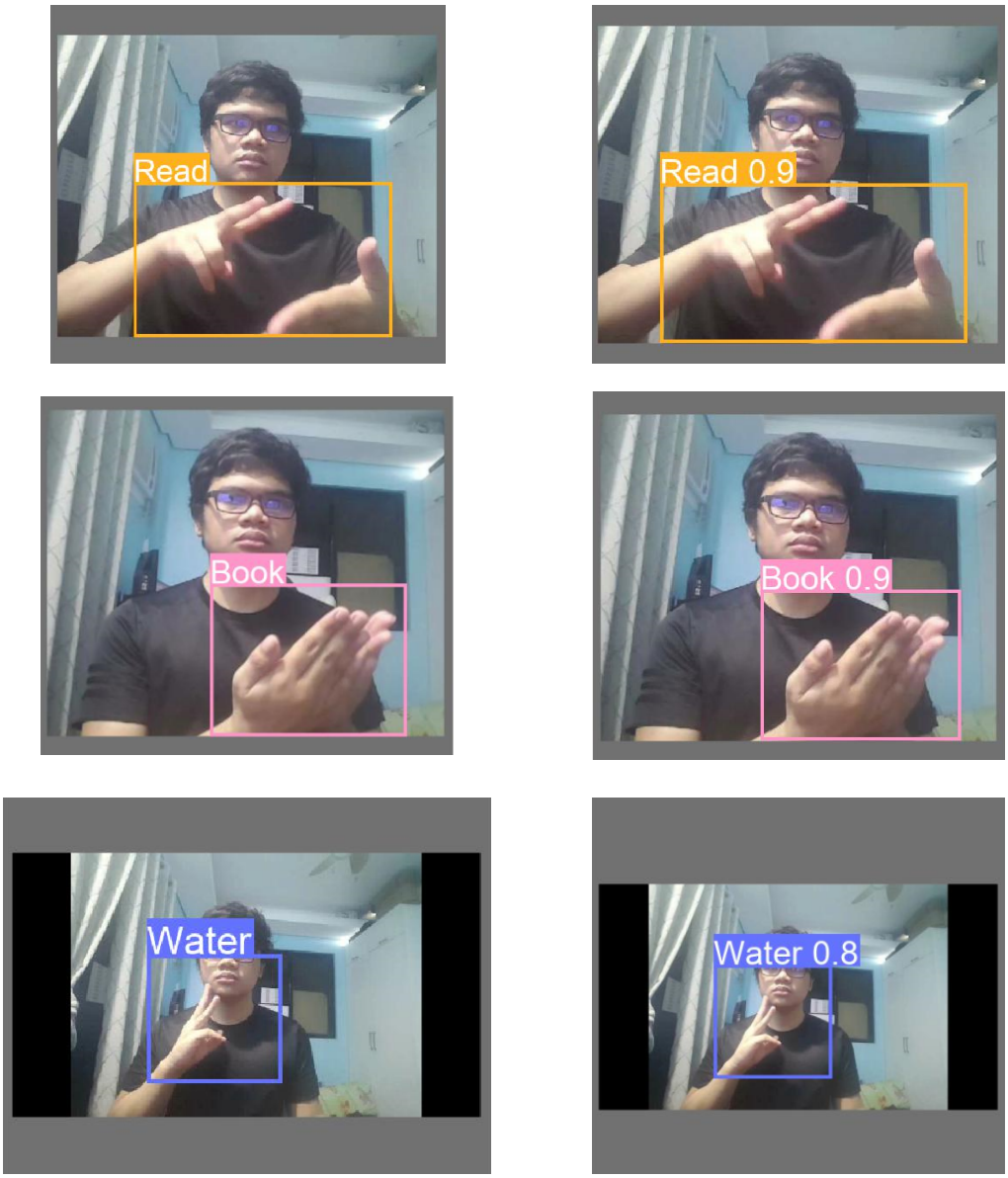


(a) (b) (c)

Figure 4.14 *Images of the New User Interface of the App*

Figure 4.14 shows images of the new user interface of the mobile application using an emulator. (a) the login screen, (b) the sign up menu and (c) the main menu.

4.2.3 Secondary Training of Datasets



(a) **(b)**

Figure 4.15 *Image Results of a Test using Collected Dynamic Dataset. (a) the images on the left are images to be identified during the training, and (b) the images to the right are results of the training.*

Figure 4.15 shows the image results during the secondary training.As shown, the hand signs that correspond to read, book, and take care are recognized by the system.

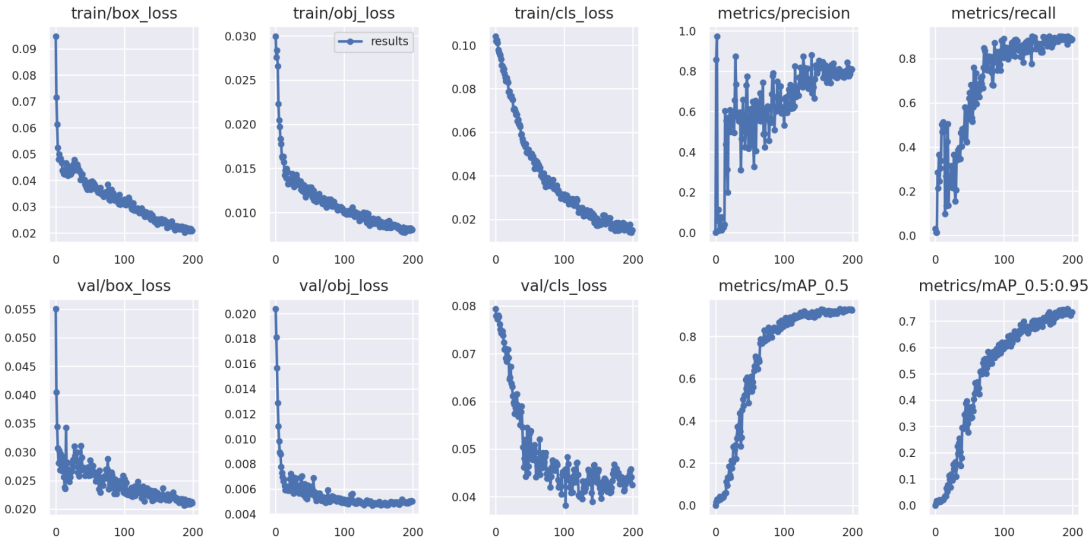


Figure 4.16 *Plots of box loss, object_loss, classification loss, precision, recall, and mean average precision (mAP) over the training dataset and validation set from the secondary training.*

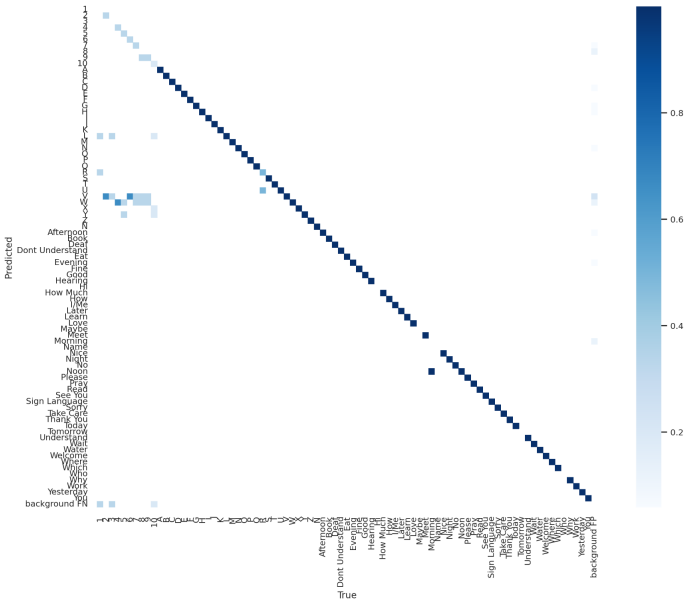


Figure 4.17 *Confusion Matrix of the Entire Dataset*

Chapter 5

SUMMARY OF FINDINGS, CONCLUSION, AND RECOMMENDATION

This chapter provides an overview of the study's outcomes, including findings, conclusions, and suggestions.

5.1 Summary of Findings

Results of Evaluation based on the Accuracy of the Translation	
Hand Sign	Average Accuracy Score based on the Evaluation (1 -5)
1	4
2	4
3	4

4	3
5	2
6	3
7	3
8	2
9	4
10	5
A	5
B	5
C	5
D	5
E	5
F	5
G	5
H	5
I	5
J	5
K	5
L	5
M	2
N	3
Ñ	5
O	5
P	5
Q	4

R	5
S	5
T	5
U	4
V	5
W	4
X	5
Y	5
Z	4
Afternoon	4
Book	5
Deaf	5
Don’t Understand	3
Eat	5
Evening	5
Fine	5
Good	5
Hearing	5
How much-	4
How-	4
Language	5
Later	5
Learn	5
Love one another	3
Maybe	5

Meet	4
Morning	5
Name	5
Nice	5
Night	5
No	5
Noon	2
Please	5
Pray	5
Read	3
See You	5
Sign	5
Sorry	5
Take Care	5
Thank You	5
Today	4
Tomorrow	5
Two Hands	4
Understand	5
Wait	5
Water	5
Welcome	5
Where	4
Which	4
Who	5

Why	4
Yesterday	5

Table 5.1 *Results of Evaluation based on the Accuracy of the Translation*

5.2 Conclusions

The study's primary objective was to create a system that could convert static and dynamic indicators into their corresponding word equivalents. A first training period and a subsequent training phase made up the research. The Preliminary Stage emphasizes static symbols like the alphabet and numbers. The system places a focus on mobile applications and dynamic gestures throughout the secondary stage.

One of the primary objectives of the study is to surpass the accuracy and efficiency of previous deep learning-based studies. Our system achieved a training accuracy of about 99%, a testing accuracy of 97.6% for static signs, and an average time of 0.02722 seconds. For dynamic gestures, it has accuracy of 89.25% with an average time of 0.02812 seconds.

We also compared YOLO to its older versions and demonstrated that Yolov5 is significantly more dependable and quicker than its predecessor. Yolov5 is analogous to the machine learning techniques utilized by the other study. With the aid of YOLO, we were able to identify the provided signs without using gloves or hand markings, as was done in the previous study.

5.3 Recommendation

Based on the research findings, and for the further improvement of this study, the researchers proposed the following recommendations:

1. For the dynamic accuracy, continuous training of data sets can improve the overall accuracy but it occupies more storage space in the cloud which made the group upgrade the subscription with the Roboflow.

2. Speech and facial recognition can be added so it can provide a true to life translation of some signs or a sentence based translation.

3. A better UI for the android application and web browser site for PC/MAC users.

APPENDIX A

CODE

Install Requirements

```
git clone https://github.com/ultralytics/yolov5  # clone
cd yolov5
pip install -r requirements.txt  # install
```

Run Inference using detect.py with Trained Weights

```
python detect.py --weights model.pt --img 416 --conf-thres 0.7 --source video.mp4
```

Training

```
python train.py --img 416 --batch 128 --epochs 350 --data data.yaml --weights yolov5s.pt --cache
```

Pantomime Flask

```
import argparse
import os
from runpy import run_module
import sys
from pathlib import Path

import torch
import torch.backends.cudnn as cudnn

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
```

```
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative


from models.common import DetectMultiBackend
from utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from utils.general import (LOGGER, check_file, check_img_size, check_imshow,
check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args, scale_coords,
strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync



7
from flask import Flask, render_template, Response
import cv2
app=Flask(__name__)
default_vid_path = "http://192.168.1.11:8080/video"
default_model_path = "runs/train/exp/weights/best.pt"



@torch.no_grad()
def run(
    weights=ROOT / default_model_path,  # model.pt path(s)
    source=ROOT / "http://192.168.55.107:8080/video",  # file/dir/URL/glob, 0 for
webcam
    data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
    imgsz=(640, 640),  # inference size (height, width)
    conf_thres=0.25,  # confidence threshold
    iou_thres=0.45,  # NMS IOU threshold
    max_det=1000,  # maximum detections per image
    device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
```

```
        view_img=True,  # show results // CHANGE TO TRUE
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
):
    print(source, "DEBUG SOURCE")
    print("RUNNING")
    source = default_vid_path
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)

    print(webcam, "WEBCAM CHECK!")
    webcam = True
```

```
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset)  # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
        bs = 1  # batch_size
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt else bs, 3, *imgsz))  # warmup
    dt, seen = [0.0, 0.0, 0.0], 0
    for path, im, im0s, vid_cap, s in dataset:
        t1 = time_sync()
```

```
        im = torch.from_numpy(im).to(device)
        im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        t2 = time_sync()
        dt[0] += t2 - t1

        # Inference
        visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
        pred = model(im, augment=augment, visualize=visualize)
        t3 = time_sync()
        dt[1] += t3 - t2

        # NMS
        pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
        dt[2] += time_sync() - t3

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
            else:
```

```
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness, example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
```

```
                label = None if hide_labels else (names[c] if hide_conf else f'{names[c]}
{conf:.2f}')
                annotator.box_label(xyxy, label, color=colors(c, True))
                if save_crop:
                    save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] /
f'{p.stem}.jpg', BGR=True)

        # Stream results
        im0 = annotator.result()
        if view_img:
            # cv2.imshow(str(p), im0)
            ret, buffer = cv2.imencode('.jpg', im0)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
            # cv2.waitKey(1)  # 1 millisecond

        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
            else:  # 'video' or 'stream'
                if vid_path[i] != save_path:  # new video
                    vid_path[i] = save_path
                    if isinstance(vid_writer[i], cv2.VideoWriter):
                        vid_writer[i].release()  # release previous video writer
                    if vid_cap:  # video
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

```
                else:  # stream
                    fps, w, h = 30, im0.shape[1], im0.shape[0]
                    save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix on results videos
                    vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                vid_writer[i].write(im0)


        # Print time (inference-only)
        LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')


    # Print results
    t = tuple(x / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights)  # update model (to fix SourceChangeWarning)


def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / default_vid_path, help='model path(s)')
    parser.add_argument('--source', type=str, default=ROOT / default_vid_path, help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
```

```
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    opt = parser.parse_args()
```

```
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt


def gen_frames():
    opt = parse_opt()
    print("Printing OPT", "*"*100)
    print(opt)
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))


@app.route('/')
def index():
    return render_template('index.html')
@app.route('/video_feed')
def video_feed():
    return Response(run(), mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == "__main__":
    app.run(host="0.0.0.0",debug=True)
```

Pantomime Main

```
from flask import Flask, render_template, Response
import cv2
app=Flask(__name__)
camera = cv2.VideoCapture("http://192.168.55.107:8080/video")
```

```
def gen_frames():
    while True:
        success, frame = camera.read()  # read the camera frame
        if not success:
            break
        else:

detector=cv2.CascadeClassifier('Haarcascades/haarcascade_frontalface_default.xml')
            eye_cascade = cv2.CascadeClassifier('Haarcascades/haarcascade_eye.xml')
            faces=detector.detectMultiScale(frame,1.1,7)
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
             #Draw the rectangle around each face
            for (x, y, w, h) in faces:
                cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
                roi_gray = gray[y:y+h, x:x+w]
                roi_color = frame[y:y+h, x:x+w]
                eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 3)
                for (ex, ey, ew, eh) in eyes:
                    cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
if __name__=='__main__':
    app.run(debug=True)
```

12% Overall Similarity

Top sources found in the following databases:

- 11% Internet database
- 5% Publications database
- Crossref database
- Crossref Posted Content database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	its301.com Internet	2%
2	coursehero.com Internet	1%
3	jb51.net Internet	1%
4	researchgate.net Internet	<1%
5	irjmets.com Internet	<1%
6	medium.com Internet	<1%
7	codeddevil-01blogs.blogspot.com Internet	<1%
8	Gunawan Wang, D Y Bernanda, J F Andry, Ahmad Nurul Fajar, Sfenrian... Crossref	<1%
9	ijisae.org Internet	<1%

10	astesj.com Internet	<1%
11	towardsdatascience.com Internet	<1%
12	answers.opencv.org Internet	<1%
13	Shaik Vaseem Akram, Anil Kumar Dixit, Kailash Bisht. "AutomaticRobot... Crossref	<1%
14	Gaolin Fang, Wen Gao, Debin Zhao. "Large-Vocabulary Continuous Sign... Crossref	<1%
15	mdpi.com Internet	<1%
16	pnrjournal.com Internet	<1%
17	Saurav Kumar, Sarthak Malik, P. Sumathi. "Deep Learning-based Borde... Crossref	<1%
18	Yang Yang, Wensheng Zhang, Zewen He, Dongjie Chen. "Locator slope ... Crossref	<1%
19	miic.gov.jm Internet	<1%
20	El Mehdi Ben Laoula, Marouane Midaoui, Mohamed Youssfi, Omar Bou... Crossref	<1%
21	Yashal Railkar, Aditi Nasikkar, Sakshi Pawar, Pranjal Patil, Rohini Pise. ... Crossref	<1%

22	scilit.net Internet	<1%
23	dspace.library.uvic.ca Internet	<1%
24	J Sudhakar, Viswesh V Iyer, Sree T Sharmila. "Image Caption Generatio... Crossref	<1%
25	ijirset.com Internet	<1%
26	K. Suresh Kumar Patro, Vinod Kumar Yadav, V. S. Bharti, Arun Sharma, ... Crossref	<1%
27	R. L. Weerasinghe, G. U. Ganegoda. "A Comprehensive Review on Visio... Crossref	<1%
28	en.wikipedia.org Internet	<1%
29	slideshare.net Internet	<1%
30	etd.astu.edu.et Internet	<1%
31	silo.tips Internet	<1%
32	myscholar.umk.edu.my Internet	<1%
33	sandipanweb.wordpress.com Internet	<1%

34	Rafa E. Al-Qutaish. "An Investigation of the Weaknesses of the ISO 912... Crossref	<1%
35	cual.openrepository.com Internet	<1%
36	dokumen.pub Internet	<1%

Excluded from Similarity Report

- Submitted Works database
- Quoted material
- Small Matches (Less then 10 words)
- Manually excluded text blocks
- Bibliographic material
- Cited material
- Manually excluded sources

EXCLUDED SOURCES

github.com Internet	12%
gitlab.sliit.lk Internet	12%
stackoverflow.com Internet	11%
forums.developer.nvidia.com Internet	11%
huggingface.co Internet	11%
git.etud.insa-toulouse.fr Internet	11%
ykkim.gitbook.io Internet	10%
blog.csdn.net Internet	10%
cnblogs.com Internet	10%

code84.com Internet	10%
uj5u.com Internet	9%
learnopencv.com Internet	9%
its404.com Internet	9%
cxymm.net Internet	9%
codetd.com Internet	9%
git.openi.org.cn Internet	8%
khuhub.khu.ac.kr Internet	7%
csdn.net Internet	7%
programtalk.com Internet	7%
codebaoku.com Internet	5%
gitee.com Internet	5%

iter01.com

Internet

3%

s-cube-network.eu

Internet

<1%

EXCLUDED TEXT BLOCKS

import sysfrom pathlib import Pathimport torchimport torch.backends.cudnn as c...

www.coursehero.com

if webcam:view_img = check_imshow()cudnn.benchmark = True # set True to spe...

git.openi.org.cn

Pantomime: Tensorflow based Dynamic Filipino Sign Language Gesture Recognition using YOLO

A Project Study Presented to the Faculty of
Electronics Engineering Department
College of Engineering
Technological University of the Philippines

In Partial Fulfillment of the Course Requirements for the Degree of
Bachelor of Science in Electronics Engineering

The Proponents:

PAULE, Ella Mae S.

POMASIN, Anchelo Oscar C.

REGUDO, Gian Eduard T.

TENCHAVEZ, Ron Gerard L.

ZABALA, Arnell B.

Adviser:
ENGR. JESSICA S. VELASCO

August 2022

APPROVAL SHEET

This project entitled "**Pantomime: TensorFlow based Dynamic Filipino Sign Language Gesture Recognition using YOLO**" has been prepared and submitted by the following proponents:

PAULE, Ella Mae S.
POMASIN, Anchelo Oscar C.
REGUDO, Gian Eduard T.
TENCHAVEZ, Ron Gerard L.
ZABALA, Arnell B.

In partial fulfillment of the requirements for the degree of Bachelor of Science in Electronics Engineering is hereby recommended for approval.

ENGR. JESSICA S. VELASCO
Project Adviser

ENGR. AUGUST THIO-AC
Panel Member

ENGR. MARK P. MELEGRITO
Panel Member

ENGR. GLENN C. VIRREY
Panel Member

ENGR. IOMER V. CATIPON
Panel Member

ENGR. LEAN KARLO S. TOLENTINO
Panel Member

ENGR. JUNE ANTHONY ASISTIO
Panel Member

Accepted and approved in partial fulfillment of the course requirements for the degree of **Bachelor of Science in Electronics Engineering.**

ENGR. TIMOTHY M. AMADO
Head, ECE Department

ENGR. NILO M. ARAGO
Dean, College of Engineering

ACKNOWLEDGEMENT

The authors wish to convey their sincere thanks to all the individuals and organizations whose support was essential to the success of this study.

First, we want to express our gratitude to our Lord, God, for bestowing upon us the strength, wisdom, vitality, resources, and chance that we will need to carry out this study and provide direction from beginning to end.

Second, we want to thank our project study's advisor, Engr Jessica S. Velasco, from the bottom of our hearts. Thank you for all of her hard work, encouragement, and vital assistance during our study. Her advice, criticism, and knowledge were extremely helpful in determining the focus and outcome of our study.

We are extremely grateful to Engr. Nilo M. Arago, our Methods subject professor, Project Study professor, and the college dean, for his tremendous help in helping us create and select our thesis subjects and leading us during our study. We also cannot forget about the contributions of Engr. Lean Karlo Tolentino who generously shared his time, expertise, and knowledge with us, which had a huge impact. improved the quality and depth of our thesis. We sincerely value his advice and assistance.

We also want to express our sincere appreciation to our panel members, engineers August Thio-Ac, Glenn Virrey, June Anthony Asistio, Jomer Catipon and Mark Melegrito, for their questions, comments, and ideas that helped us refine our research.

We also acknowledge the contributions of Mr. Ken Bulahan and his friends in the deaf community, who shared their knowledge of Filipino Sign Language during consultations and helped make this study possible. To our friends and coworkers who have accompanied us and helped us along the way. And to our loving parents and other family members, whose never-ending support and encouragement enabled us to complete our education. We sincerely thank them for supporting us throughout the years, even in the most trying circumstances. With all our hearts, thank you.

ABSTRACT

In recent decades, several efforts have been made to construct an SLR system. Previously, SLRs were separated into data gloves-based and vision-based types. Gesture-based SLRs are evolving as technology advances. First, the user's movement is severely limited and data gloves must be maintained. In the second example, individuals can interact with the computer more naturally; nevertheless, numerous technological concerns regarding image processing remain unresolved or must be improved, and colored gloves or other visual assistance are necessary for vision-based medium or big vocabulary SLR activities. Diverse and sophisticated hand motions might affect final scenario reliability and recognition rates.

While the aforementioned studies convert sign language into voice and/or text, a system will be recommended that matches or exceeds their accuracy. You Only Look Once Convolutional Neural Network will be used to recognise hand motions faster and more accurately. YOLO's architecture is fast. YOLO processes 45 frames per second in real-time. Fast YOLO processes 155 frames per second while double the mAP of existing real-time detectors. YOLO makes more localization errors than modern detection algorithms but is less prone to predict false positives. YOLO gets general object representations. It outperforms DPM and R-CNN when generalizing from natural photographs to artwork..

TABLE OF CONTENTS

List of Figures

List of Tables and Graphs

CHAPTER 1

THE PROBLEM AND ITS BACKGROUND

Introduction, research background and history, statement of the problem, objectives, importance and significance, scope, and limitations of the project are all included in this chapter.

1.1 Introduction

Speaking is an important part of human life. Most of the people today communicate in this manner and not all 7 billion people can communicate. Deafness can have an influence on childrens' ability to speak and develop or learn a language. When a child has difficulty in hearing, the areas of the brain responsible for communication may not develop properly, making understanding and speaking difficult. When a child's hearing loss is detected early and treated appropriately, he or she can become an effective communicator. Caregivers and professionals collaborate in this process. The majority of hearing losses are discovered during a newborn screening. Some children are not given a diagnosis until their speech or language skills are failing. Early recognition and treatment of hearing loss leads to better outcomes for children.

The WHO estimates that 460+ million people, or around 5% of the world's living population have some form of deafness, but only 1.23% of people in the Philippines are said to be deaf or mute.

However, since it takes a long time to understand properly, the majority of hearing people are unaware of it.

The concept of a sign language translator is not new. However, the majority of the common population do not comprehend sign language, and learning it is a challenging task. Consequently, there is still a sizable disparity between the hearing majority and the deaf population.

Assistive machine learning technologies such as Artificial Neural Network (ANN) and Artificial Intelligence were used to include in the sign language recognition system that assists in the creation of a fully inclusive community. Using the right machine learning algorithm to interpret sign languages and translate them into words, or vice versa, in the shortest time possible is a huge aid in the everyday lives of the people without speech and hearing disability, and other PWDs who have difficulty communicating and participating in the society.

1.2 Background of the Study

Many efforts have been made over the past few decades to develop a Sign Language Recognition (SLR) system. SLR was previously divided into two groups depending on the tools, such as datagloves-based SLR and vision-based SLR, that were utilized to record hand motions. As technology advances, gesture-based SLR has been added to the list. The user's range of motion is highly constrained in the first scenario, and data gloves must be properly cared for. The second scenario, however, allows for more natural user interaction with the computer, but many technical issues with image processing remain unresolved or need to be addressed due to the limitations of current computer vision techniques, necessitating the use of colorful gloves or other visual aids.

SLR is divided into two categories: Isolated Sign Language and Continuous Sign Classification. Isolated Sign Recognition is concerned with the recognition of signs that are performed alone, with no signs before or following the performed sign. As a result, the sign is done without regard for the preceding or consecutive signs. SLR faces difficulty in creating ways to tackle continuous vocabulary-heavy sign issues. A significant contribution to large-vocabulary continuous SLR research is unquestionably necessary and has an impact on how natural human-computer interfaces feel. to encourage the widespread use of an SLR system.

Managing epenthetic movement is the main difficulty with Continuous SLR. The movement epenthesis, also known as the shift between two adjoining hand signs, differs depending on the context of the preceding and succeeding signs and occurs at their respective ends and beginnings. Since the epenthesis adds a variety

of additional movements that are not present in the grammatical forms of the signs and doesn't just affect the performance of nearby signs, its inclusion significantly increases the difficulty of recognizing the signs.

In context-dependent models for continuous SLR, like the triphone or biphone are frequently used to simulate the coarticulation. In constant SLR, however, no fundamental unit such as the phoneme of speech has yet been identified in the sign lexicon. The number of subunits retrieved manually or mechanically for the entire sign language is so great that training data becomes extremely scarce. This makes it impossible to train context-dependent models for large-vocabulary SLR, such as those described in the literature. The same problem affects direct models of the movement of epenthesis between signs.

Some used probabilistic video processing techniques, including the renowned MotionSavvy's Leap's 3D Motion and Hidden Markov Model or Artificial Neural Network classifiers. While these studies aim to translate sign language into voice and/or text, a proposed system will be as accurate as or more accurate than the studies shown.

You Only Look Once or YOLO, a new approach to object detection was made last 2016 (by Redmon et al). In earlier labor on object detection, classifiers were repurposed to carry out detection. The creators envisioned object recognition as a decline problem that involves the class and bounding boxes probabilities. Within an assessment, the neural network guesses the boxes and class possibilities straight from the whole images. Since the whole detection conduit is a singular network, the spotting operation may be tuned point-to-point.

In recent years, the involvement of vision-based approaches for sign language recognition became a major subject for research development with a lot of influential and modern implementations such as gesture recognition interpretation, motion control, and human-computer interaction (HCI) had already made recognition in ongoing research topics. Due to a number of problems, including the complex nature of static or dynamic hand gestures, distortions, and complex backgrounds increases the difficulty.

The need of very intensive computer resources to address the issue in its generality which requires elaborate algorithms.

While these studies aim to translate sign language into text, a system will be proposed that will match and perhaps even outperform the studies' degree of accuracy. Thus, You Only Look Once or YOLO Convolutional Neural Network in our proposed system for recognizing hand gestures will be used for faster and more accurate results.

YOLO's centralized structure develops very quickly. The basic YOLO model performs 45 frames per second of real-time image processing. Fast YOLO, a scaled-down version of the network, processes astonishingly 155 frames per second while still outperforming other real-time detectors in mAP by a factor of two. If compared side to side with other detector systems, However, YOLO is less likely to forecast false positives in the background and creates further localization errors. Lastly, YOLO picks up very broad representations of objects. When applied to other domains like artwork, it performs better than other detection techniques like DPM and R-CNN when generalizing from natural images.

1.3 Statement of the Problem

Through hand gestures, body language, facial expressions, people can express themselves visually. Although sign language is the primary means of communication for the Deaf and Hard-of-Hearing community, it can also be helpful for other populations, including those with autism, with speech disorders, cerebral palsy, and down syndrome.

In a way, the statement that learning sign language is difficult is both accurate and untrue. It primarily relies on the kind of sign language you're attempting to learn. Having said that, learning a language is much harder as you become older than it is while you're younger. Additionally, it's typically impossible to fully immerse yourself in sign language outside of the classroom. Your interactions may frequently be restricted to one deaf family member. Your objectives and demands would be very different if you were speaking with a child as opposed to an adult. This could affect how quickly or slowly you pick up the language.

The usual speed of social communication, especially when done in a group, can be overwhelming, which is another difficulty in learning sign language. It elevates communication to a whole new level and necessitates the mastery of eye gaze to more adeptly manage the give-and-take of social relationships.

The main problem is how people who have no background in sign language will be able to communicate to people with hearing and speech disabilities. Considering the limitations of both parties, the portability, functionality, and accessibility of the translator should also be considered.

1.4 Objectives of the Study

1.4.1 General Objectives

The main objective of this study is to develop a mobile application that will translate Filipino dynamic gestures into words and vice versa using machine learning.

1.4.2 Specific Objectives

1. To develop a dynamic mobile translator application that utilizes a deep learning called You Only Look Once (YOLO).
2. To design an interactive easy-to-use user's interface using Tensorflow and Python as the programming language.
3. To provide a learning module in the mobile application for those who wanted to learn sign language.
4. To test the developed application, and evaluate the accuracy of the translation in its functionality, reliability, usability, efficiency and its maintainability as specified in the ISO 9126 Software Quality Standards.

1.5 Significance of the Study

There are 10,000 words that cannot be translated into single sign language. It can be efficient for everyone who has difficulty in communicating with people who have hearing impairment to translate the dynamic gestures into words so people without understanding of the sign language will be able to communicate with them. It can lessen the stigma, promote awareness, and give more job opportunities to the people who have hearing impairment. Also, it can help their family and friends to understand and learn sign language. The proposed system is designed to create an application that would convert one of the two inputs to an output form; to text provided by the device display, and video taken using the device camera to an audio recorded by a microphone. Therefore, the proposed system will be using a YOLO and will be testing which variable of YOLO is suited for mobile application development. Tensorflow and python will be used for the mobile application development that will be programmed for the automated translations that will help the conversations between those who are not well versed in the Filipino Sign Language and those who have speech disability or have hearing disability, providing a convenient way to bridge the language barrier between the two populations.

1.6 Scope and Limitations

The study focuses on developing a system that could translate common words and expressions used by the deaf community in their everyday lives. As we will be using YOLOv4 for the accuracy of detecting hand gestures and the reliability of the detection. Then all of the detected gestures will be processed and be converted into words understandable for the intended receiver. Thus, an application that can recognize a list of Filipino Sign Language (FSL) dynamic gestures.

Sign Language doesn't only involve hand gestures but it includes the whole body in communicating, this includes facial expressions and body movements. This study will not cover facial expression recognition. The lexicon of gestures will be also limited in the most frequently used phrases of a person such as the terms used in transportation, school or work and basic conversation.

The proposed system will be using Filipino Sign Language (FSL) as the main sign language instead of American Sign Language (ASL). Although FSL was originally rooted from the ASL and soon developed and became a separate language from ASL, the Filipinos who's suffering from hearing disability and speech disability used it in communication and was mandated to be the official sign language of the Philippines. As Filipino signs that emerged naturally through generations and adding new signs, especially those that are related to technology (Apurado and Agravante, 2006)

1.7 Definition of Terms

The following are the definitions of words used in the paper.

- **Deep Learning -** consists of a subset of machine learning in Artificial Intelligence (AI) that comprises networks that can learn autonomously from unstructured, unlabeled input.
- **Python** - is an interpreted, readable, object-oriented programming language with a simple syntax. Since its statements can be interpreted in a variety of ways, it is said to be relatively simple to learn and portable. systems operating.
- **American Sign Language (ASL)** - is the sign language most commonly practised by deaf communities in the United States and most of Anglophone Canada.
- **Filipino Sign Language (FSL) -** Sign language from the Philippines is known as Philippine Sign Language. FSL is a distinct language with its own grammar, syntax, and morphology like other sign languages; it is neither based on nor similar to Filipino or English. Most institutions across the nation use FSL as their primary sign language.
- **Lexicons -** a book that contains the words in a language alphabetically and their respective definitions.
- **Keras -** is an open-source software library that offers Python support for implementations and tools to simplify the coding required for an artificial neural network by making image and text data easier to handle.
- **Neural Network** - is a series of algorithms that aims to mimic the human brain in perceiving the relationships between data in a set.

- **TensorFlow -** is a machine learning and artificial intelligence software library that is open-source and free. It can be used for a variety of tasks, but focuses particularly on deep neural network training and inference.
- **Convolutional Neural Network** - also called CNN, or ConvNet, is a class of deep neural networks that specializes in analyzing and processing data with grid-like topology, like digital images.
- **Dynamic Gesture Recognition Interface** - is an interface that allows the recognition of gestures like hand signals from an active camera footage.
- **OpenCV -** developed by Intel, is a free-to-use cross-platform library of programming functions aimed at real-time identification and understanding of digital images or videos.
- **Yolobile Framework -** a real-time object framework that is used for mobile devices to improve speed and accuracy of object detection in mobile devices

CHAPTER 2

REVIEW OF RELATED LITERATURE

This part of the research is where the ideas in different studies come together to develop an innovative project with a background that has a basis and purpose for improvement. It circles on theories and principles of related literature and studies in conceptualizing ideas that are useful to the whole research process.

2.1 Conceptual Literature

2.1.1 You Only Look Once used for Object Detection

YOLO is a method for detecting objects. Earlier object detection research uses classifiers in a new way to carry out detection. Instead, the researchers (Redmon, Divvala, Girshick, Farhadi, 2016) frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Bounding boxes and class probabilities are directly predicted by a single neural network from whole pictures in a single assessment. The whole detection process consists of a single network, therefore may be fully tuned based on detecting performance.

The underlying YOLO model analyses pictures in real-time at a rate of 45 frames per second because of the unified architecture's lightning-fast speed. Fast YOLO, a scaled-down version of the network, processes just 155 frames per second while still outperforming other real-time detectors in mAP by a factor of two. YOLO produces more localization mistakes than cutting-edge detection algorithms,

but it is less likely to anticipate false positives in the background. Last but not least, YOLO picks up very broad representations of things. When applied to other domains like artwork, it performs better than other detection techniques like DPM and R-CNN when generalizing from natural pictures.

Capitalizing on the speed and accuracy that YOLO offers on object detection, a recent study shows that YOLO can be used in mobile devices, via compression-compilation co-design. The resulting model is called the YOLObile framework.

Using a noble-block pruning scheme and a GPU-CPU collaborative scheme ,to improve computational efficiency, the experimental results indicate that the pruning scheme achieves 14× compression rate of YOLOv4 with 49.0 mAP (Yuxuan Cai et. al, 2020).

Under the YOLObile framework, the researchers achieved 17 FPS inference speed using GPU on Samsung Galaxy S20, and after incorporating the GPU-CPU collaborative scheme, the resulting inference speed increased to 19.1 FPS, and outperformed the original YOLOv4 by 5 times (Yuxuan Cai et. al, 2020).

Table 2.1 Summary Table for Object Detection using You Only Look Once

Title	Objective	Approach
You Only Look Once: Unified, Real-Time Object Detection	To test YOLO for its accuracy and speed in object detection.	Uses a network that creates bounding boxes and confidence scores to find an object. The resulting box should contain an object and should be able to name

		it.
YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design	To utilize YOLO in creating a real-time object detection on mobile devices via compression-compilation co-design.	Uses a DNN-based object detector (YOLOv4) to detect the object, enhance the accuracy by using block-punched pruning, and use GPU-CPU collaborative scheme to improve computational efficiency.

2.1.2 Sign Language Recognition Systems

An creative, organic, and consumer way to interact with a computer that is more tailored to human needs is made available by sign recognition systems. Gesture recognition has many uses, including in interactive game technology, human-machine interaction, and sign language. By keeping in mind the similarities of human hand shape with four fingers and one thumb, the Sign Language Recognition Using Neural Network of (Bachani, Dixit, Chadha, & Prof. Bagul, 2020) aims to present a real time system for hand gesture recognition on the basis of detection of some meaningful shape based features like orientation, center of mass (centroid), status of fingers, thumb in terms of raised or folded fingers of hand and their respective location in image. The method that was put forth completely depended on the hand gesture's shape parameters. Skin tone and texture are not taken into account as additional methods of hand gesture recognition because they are so sensitive to changes in lighting and other factors. They used a basic webcam

that operates at 20 frames per second and has a resolution of 7 mega pixels to implement the strategy.

Another study uses sign language or gesture recognition in order to provide the deaf people a means to control home systems using hand signs and gestures. The system developed uses a real-time dynamic gesture recognition to detect the gestures and is trained through machine learning to understand the given command and perform it (Kraljevic et. al, 2020).

Table 2.2 Summary Table for Sign Language Recognition Systems

Title	Objective	Approach
Sign Language Recognition using Neural Network	To create a real-time system for hand gesture recognition based on shape-based features.	Uses a webcam that usss 20 fps and 7 megapixel intensity to recognize the shape of the hand.
A Dynamic Gesture Recognition Interface for Smart Home Control based on the Croatian Sign Language	To utilize YOLO in creating a real-time object detection on mobile devices via compression-compilation co-design.	Uses a DNN-based object detector (YOLOv4) to detect the object, enhance the accuracy by using block-punched pruning, and use GPU-CPU collaborative scheme to improve computational efficiency.

2.2 Related Studies

2.2.1 SignSpeak

Research on the recognition of sign language has been investigated for many years. Numerous studies have employed sensor-based tools like "SignSpeak."

This technology made use of a variety of sensors, including flex and touch sensors for the motions of the fingers and palms and accelerometer and gyroscopes for the movements of the hands. Principal Component Analysis was then used to train the gloves to identify various gestures and to categorize them into alphabets in real-time. To display the text and words received from the gloves via Bluetooth, it also utilized an Android phone. SignSpeak was found to have a 92% accuracy. (Bukhari, Rehman, Malik, Kamboh, & Salman, 2015) Though Sensor Gloves like this can give a precise hand gesture recognition, it can also give the user an unnatural experience due to the bulkiness of the gloves and its sensors. It will also encounter difficulties in setting up the device and it can be very expensive. Because of this many researchers jumped from sensor-based to visual-based Sign Language Recognition.

2.2.2 Static Sign Language Recognition using Deep Learning

In 2019, a study entitled "Static Sign Language Recognition using Deep Learning" by Tolentino, LK and et al. was published in International Journal of Machine Learning and Computing; they also created a system that will act as a teaching aid for those learning sign language for the first time that involves hand

detection. The system was built using explicit complexion space thresholding, a skin complexion modeling technique. In order to separate pixels (hand) from non-pixels (background), a skin-color spectrum has been defined.

The Convolutional Neural Network (CNN) model was used to classify the pictures after being fed the photos. Images were trained using Keras. The system achieved an average testing accuracy of 93.67% in ideal lighting and a consistent background, of which 90.04% was ascribed to ASL letter recognition, 93.44% to number recognition, and 97.52% to static word identification.

The method is real-time and utilized for quick computing and also based on a method that is explicitly defining the skin-color threshold and it was pre-determined that it will extract pixels from non-pixels and feed into the CNN model for classification of images.

2.2.3 Baybayin Handwritten Letters using VGG16 Deep Convolutional Network Model

In Recognition of Baybayin (Ancient Philippine Character) Handwritten Letters using VGG16 Deep Convolutional Neural Network Model (Bague, Jorda, Fortaleza, Evanculla, Paez, Velasco, 2020), they proposed a system that can convert 45 handwritten baybayin Philippines character/s into their corresponding Tagalog word/s equivalent through convolutional neural network (CNN) using Keras. The VGG16 network used in the implemented design is more compact and smaller.

Each of the 45 Baybayin characters has 1500 photos in the categorization. The segmentation stage's scaled characters (50x50 pixels) produced pixel values that were used to train the system, resulting in a 99.54% accuracy rate. 90 handwritten baybayin characters were taken in real time using the device's 1080P Full-HD web camera to test the detection technology.

The test sample will then be classified by the system. The user is then shown the relevant Tagalog word output on the screen. The testing phase's total accuracy is 98.84%. From the results, the suggested approach might thus be used for character extraction from documents and any associated conversion of handwritten materials into structured text.

2.2.4 Shape Based Continuous Real Time Hand Gesture Recognition System of American Sign Language using KNN Classifier

Shivashankara and Srinath (2018) suggested an ideal technique., Aiming to achieve the translation of 24 static sign language alphabets and numerals of American Sign Language into humanoid or machine readable English writing

The first step involves the signed input gesture's pre-processing processes. In the next phase, the various region properties of pre-processed gesture images are computed. The transcription of signed motions into text has been done in the final step based on the attributes computed in earlier phases. The statistical results evaluation and a graphic comparison of the current and suggested methodologies are also included in this study.

2.2.5 A Dynamic Gesture Recognition Interface for Smart Home Control based on Croatian Sign Language

People who are hard of hearing or deaf face several difficulties in daily life. Their means of communication are sign languages, and whether or not a language is accessible to them depends on how well understood it is in the cultural and social context. So according to the study, the introduction of a smart home automatization system specifically designed to provide real-time sign language recognition is needed. (Kraljevic et. al).

The researchers took a Conv3d network-based wake-up system and high - performing sign language recognition and applied them to this problem.

Based on a tiny Croatian sign language database with 25 distinct language signs, the embedded platform was implemented on a Nvidia Jetson TX2 with a StereoLabs ZED M stereo camera.

The system was trained and was tested by 40 users. The results show that the system achieved 80.9% accuracy, and concluded that the system was efficient enough to be used in a smart home environment (Kraljevic et. al.).

According to the study, the system is limited to the basics of the Croatian sign language, and is not adopted yet to understand complex grammar. The study also does not yet include a higher degree of user modalities such as face expressions and postures (Kraljevic et. al.).

Table 2.3 Summary Table for Related Studies

Title	Objective	Approach	Advantage	Disadvantage
SignSpeak	To create a sensor-based device that can translate hand signs and gestures into words.	Using gloves and sensors that were connected via Bluetooth, the system aims to recognize the hand gestures and translate it to words.	The device is very accurate in translating the hand gestures that were performed.	The gloves are said to be bulky and cause uncomfortability for the users. There are difficulties encountered with the device and the cost can be too high for it to be justified.
Static Sign Language Recognition using Deep Learning	To develop a system that serves as a learning tool for starters in sign language that uses hand detection.	Uses a predetermined skin-color range that pixels can be extracted from, a CNN model for classification of images and Keras for image training.	The results show that the system is very accurate in recognizing the hand signs that were shown.	The system is limited to static images, requires proper lighting and uniform background, and is slow in processing.

Baybayin Handwritten Letters using VGG16 Deep Convolutional Network Model	To create a system that can convert handwritten baybayin characters into their corresponding Tagalog words.	Using the VGG16 network the system will be trained to identify 45 baybayin from 1500 images.	Upon testing the system to real-time footage of a hand-written baybayin character, the test results show that the system is very accurate in its translation.	
Shape Based Continuous Real Time Hand Gesture Recognition System of American Sign Language using KNN Classifier	To create a continuous real-time hand gesture recognition system using KNN.	Uses a HOG, a feature descriptor used for digital image processing and KNN classifier, a machine learning algorithm for the pattern classification.	The study shows that the KNN classifier is very accurate in identifying the hand gestures in comparison to the other systems in its time.	The system is currently limited in identifying the ASL alphabet gestures and ASL number gestures.
A Dynamic Gesture Recognition Interface for Smart Home Control based on the Croatian Language	To create a smart home automatization system that uses dynamic gesture recognition to read gestures based on the Croatian sign language.	Uses a wake-up module and a high performance sign recognition module based on the Conv3d network.	The study shows that a dynamic gesture recognition system is very accurate and effective. This study also shows that the system has a great potential and is still open for more improvements .	The study is limited to a small database which limits the commands and gestures that can be used.

CHAPTER 3

METHODOLOGY

The project's development and implementation methods and procedures are discussed in this chapter. The design flow process and program algorithms were also included by the authors in this chapter.

3.1 Theoretical Framework

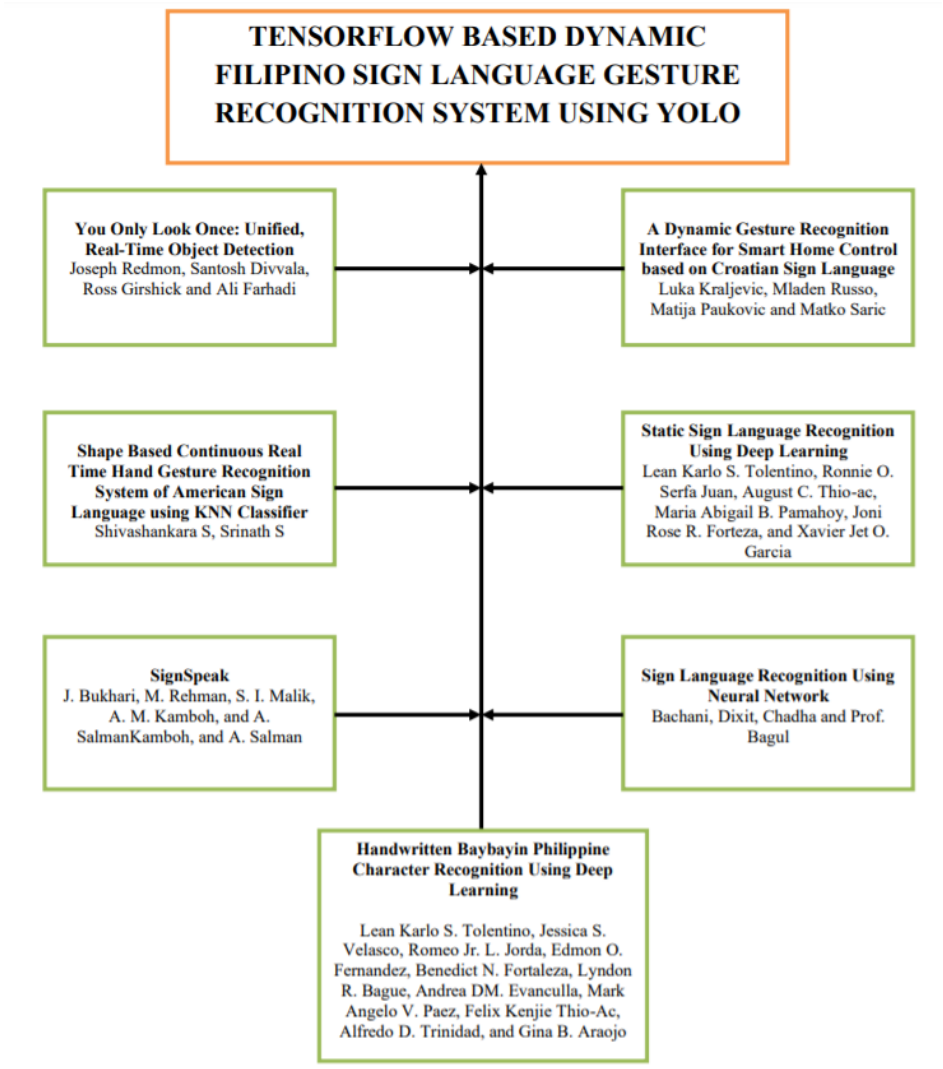


Figure 3.1 *Theoretical Framework of the Proposed Project*

Figure 1 shows the theoretical framework of the proposed project. The study takes inspiration from the previous studies of sign language gesture recognition systems which use other methods, like sensor-based gloves (SignSpeak) or visual-based neural networks (Static Sign Language Recognition using Deep Learning). The difference is the study aims to use the You Only Look Once deep learning to further increase the speed and accuracy of the translation and allow the system to be used on mobile devices.

3.2 Conceptual Framework

3.2.1 Input-Process-Output Model

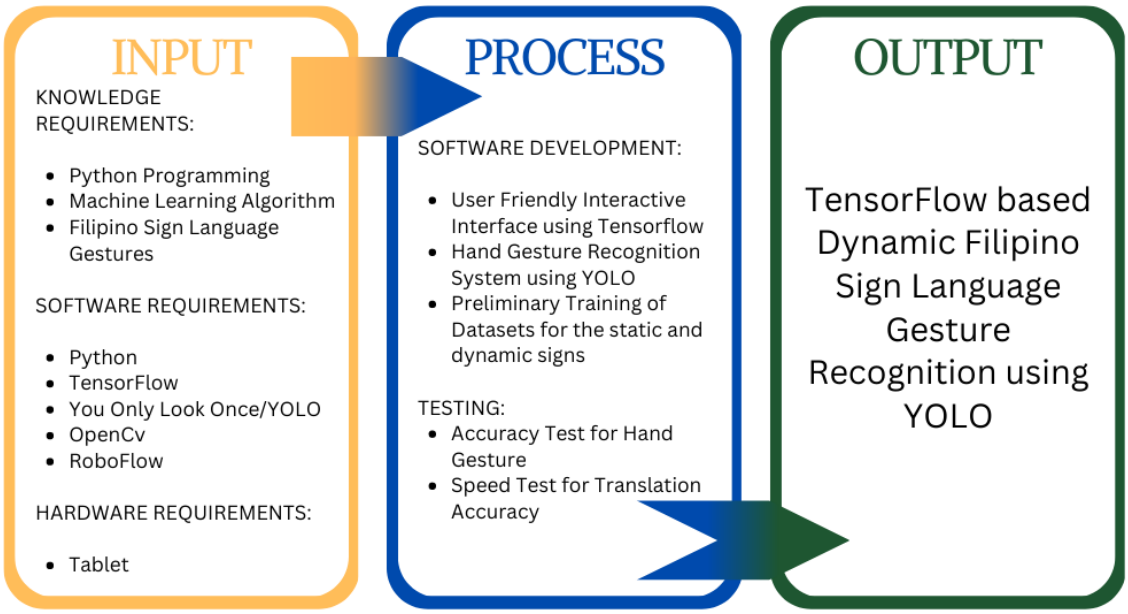


Figure 3.2 *INPUT - OUTPUT Model of the Application*

Figure 2 shows the input-output model of the application. This study focuses on developing a mobile application that uses YOLO to identify the hand gesture and translate

it into text for the user to understand. YOLO is complemented in achieving this process using OpenCV which allows the system to identify the hand from the footage. The hand gesture recognition system is trained through machine learning. On the programming side, the study uses Python as the programming language to develop the mobile app with the aid of Tensorflow. The mobile application is intended to come with an FSL tutorial module to help the users who want to learn more about Filipino sign language.

3.2.2 Research Process Flow

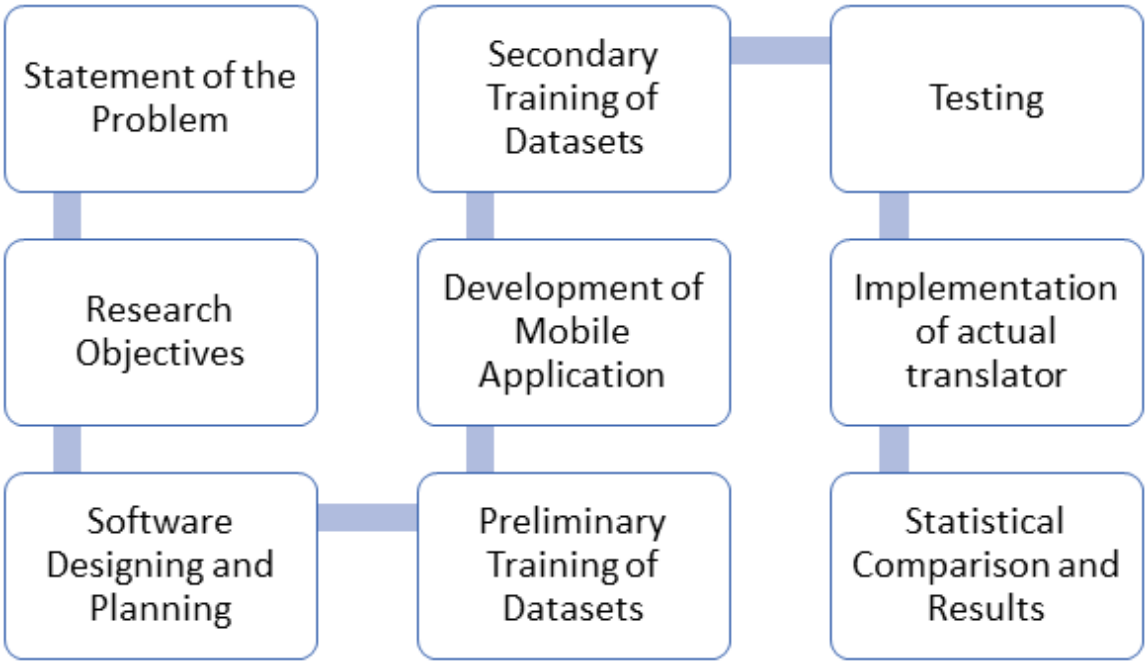


Figure 3.3 *Research Process Flow*

3.2.3 Testing Procedures

The proposed project will be having two parts of testing, the preliminary training and the secondary training. In Preliminary Training, we will be using the Desktop Computer to test the gathered datasets and it's accuracy. After gathering information about the datasets that were tested in the preliminary testing, it will be transferred to the secondary training which is the conversion of a system into a mobile application.

An additional module will be added to the mobile application that will be a guide for the users who want to learn the FSL aside from using the dynamic gesture translator.

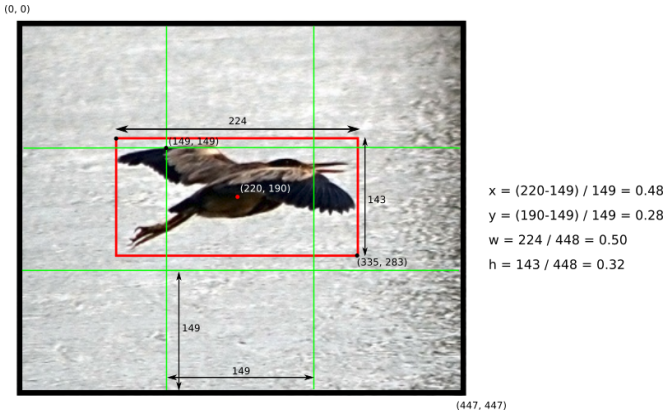
3.3 How does YOLO work?

In simpler terms, images taken as input are processed by a neural network resembling a standard CNN, and the output is a vector of bounding boxes and class predictions.

A system using YOLO divides the input image into an S x S grid of cells, and for each object present in the image, the grid cell that the object's center falls into predicts the object.

Predictions for bounding boxes and class probabilities are made in each grid cell. Five elements make up a bounding box: ***x***, ***y***, ***w***, ***h***, and confidence. In relation to the position of the grid cell, the ***x*** and ***y*** components reflect the coordinates of the box's center. The ***w*** and ***h*** box measurements are located in relation to the size of the picture. The ***x***, ***y***, ***w***, and ***h*** coordinates are normalized to lie between 0 and 1. The formula for calculating

confidence is ***confidence = Pr(Object) *IOU(pred,truth)***. The confidence score should be equal to the *intersection over union* (*IOU*) between the *predicted box* and the *ground truth* if there is no object in that cell; otherwise, it should be equal to zero.



(Source: HackerNoon - Understanding YOLO)

Figure 3.4 *An example of the box coordinates are calculated.*

The new formula is ***S x S x B* 5*** outputs linked to the bounding box predictions since each grid cell predicts a bounding box.

To estimate the class probabilities, ***Pr(Class(i) | Object)***, we must first anticipate the bounding box predictions. This likelihood is dependent on the grid cell having only one item. It implies that the loss function will penalize the grid cell for the incorrect class prediction if there is no item present on the grid cell. Regardless of the quantity of bounding boxes ***B***, the network can only predict one set of class probabilities per cell.

We obtain ***S x S x (B * 5 + C)*** as the result after including class probabilities into the output vector.

The network topology of YOLO seems, on *Network,* to be similar to a standard CNN, with convolutional and max pooling layers followed by 2 fully connected layers towards the very end.

On the *Loss Function*, there are 4 equations that present the loss in different areas. The first one relates to the loss to the predicted bounding box location ***(x,y)***.

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

The second relates to the loss connected to the predicted box's height/width..

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

The third, the loss relates to each bounding box predictor's confidence score.

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

And the fourth and last, the classification loss.

$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \epsilon classes} (p_i(c) - (\hat{p}_i(c))^2$$

3.4 YOLO and its different versions

YOLO has many different versions. The first 3 versions of YOLO are created by the Joseph Redmon et al., while the other 3 are created by 3 other individuals namely, Alexey Bochkovskiy et al. - YOLOv4, Glenn Jocher - YOLOv5 and PP-YOLO by Xiang Long et al.

3.4.1 YOLOv1

- Similar to R-CNN, but is faster. It does not have a complicated process for regression.
- Can predict using the entire image. Cannot find small images.
- Predicts less background errors in comparison to Fast R-CNN.
- Input size is 224 x 224.
- Based on Darknet trained on ImageNet-1000.

3.4.2 YOLOv2

- Input size is increased to 448 x 448.
- Based on Darknet-19 which employs 5 maximum pool layers and 19 convolutional layers.
- Uses anchor boxes.
- Can find and predict small images.
- Processes images at 40-90 frames per second.
- Faster and an overall improvement compared to YOLO and sometimes even faster than YOLOv3.

3.4.3 YOLOv3

- Input size is increased and appears to be no longer limited.
- Based on Darknet-53 which now has 53 convolutional layers and for detection, another 53 layers are added, giving a total of 106 layers.
- Uses residual blocks.
- Better at YOLOv2 at small image detection.
- Even if YOLOv2 can be faster, by altering the model size, YOLOv3 may quickly trade off accuracy for speed and vice versa without the need for retraining.

3.4.4 YOLOv4

- Created by Alexey Bochkovskiy et al.
- Still based on the Darknet.
- It makes use of a number of BoF (bag of freebies) and BoS (bag of specials) to increase detector accuracy without extending inference time at the expense of training.
- It has outperformed the quickest and most accurate detectors in terms of both accuracy and speed, obtaining a 43.5 percent in AP value on the COCO dataset and a real-time speed of 65 frames per second on the Tesla V100.
- In comparison to YOLOv3, YOLOv4 shows an increase in AP value by 10 percent and increase in FPS by 12 percent.

3.4.5 YOLOv5

- developed by Ultralytics, the same firm that created the Pytorch version of YOLOv3, announced their most recent object detection in June 2020.
- In comparison to the entire YOLOv3 model, it produces results with around 75% less processes.
- It benefits from the PyTorch ecosystem because it was built using PyTorch, making deployment and support simpler.
- They observed inference speeds up to 0.007 seconds per image in the YOLOv5 Collab notebook running a Tesla P100, translating to 140 frames per second (FPS), as opposed to the YOLOv4 which only manages 50 FPS after being ported to the same Ultralytics PyTorch library.

3.4.6 PP-YOLO

- Introduced by Xiang Long et al. in July 2020.
- Based on Parallel Distributed Deep Learning (PaddlePaddle).
- Same as YOLOv5, PP-YOLO is based on the YOLOv3 model.
- The notable changes include the replacement of Darknet53 with ResNet and increase of batch training size from 64 to 192.
- The report claims that when evaluated on a V100 with batch size = 1, PP-YOLO can reach a mAP of 45 percent COCO datasets and an inference speed of 72.9 FPS. This demonstrates that it outperforms YOLOv4 in mAP by 43.5 percent and in inference speed by 65 FPS.

3.5 YOLO vs Other Machine Learning

There are numerous other object detection approaches that can be used , so why choose YOLO over the others? The information below detailed two commonly used object detection algorithms to show the reason why YOLO is the better choice currently.

3.5.1 Fast R-CNN

- Introduced in 2015.
- Fast R-CNN is an enhanced version of the R-CNN with the goal of enhancing the R-CNN's image processing.
- Three separate models were combined into a single training framework and their shared computational outputs. In particular, each picture now utilizes a CNN forward channel and shares the feature matrix rather than trainers setting separate feature vectors for each feature. Both the classifier and the border regression matrix are built using the same eigen matrix.
- Many of the steps of R-CNN are shared with Fast R-CNN, in addition to the fact that Fast R-CNN has additional processes.
- Fast R-CNN achieved an improvement over the original R-CNN, but trade it off by being time-consuming in the process

3.5.2 Single Shot Multibox Detector (SSD)

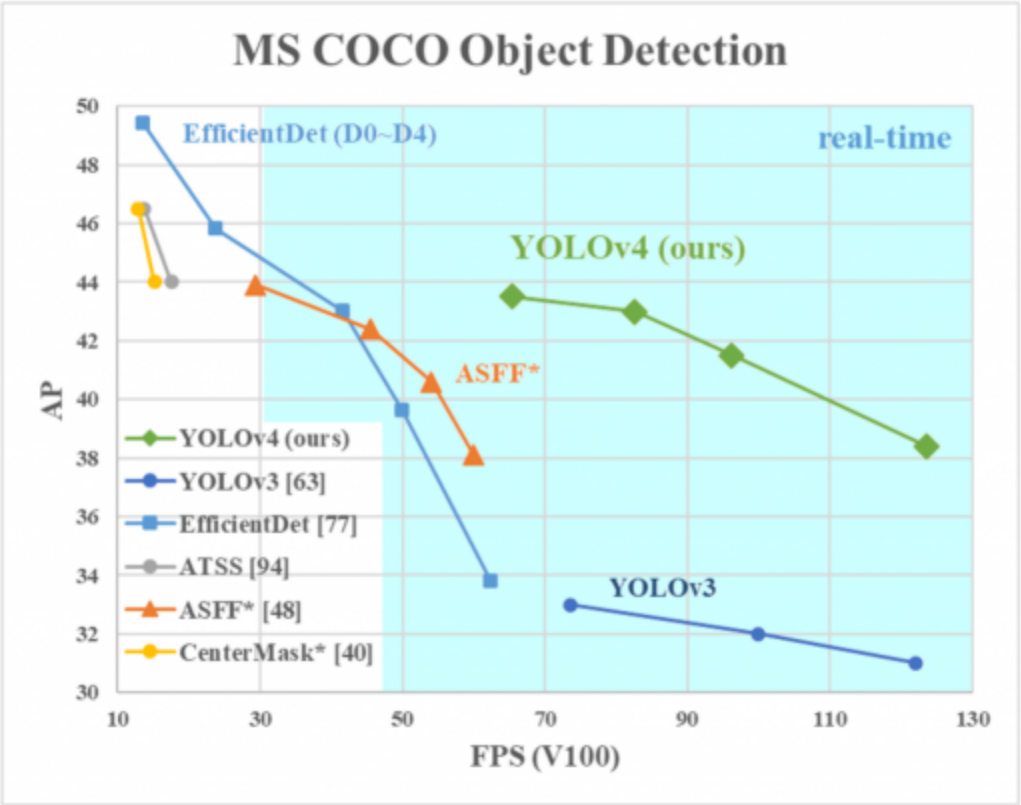
- Perform its operations at a much higher speed than YOLO and Fast R-CNN.
- SSD employs a free-forward convolution layer technique that generates a fixed-sized collection of bounding boxes, scores each instance of an object

class that is found within those bounding boxes, and also employs a Non-Max Suppression to generate the results.

- The SSD architecture is really straightforward.
- The disadvantage of SSD is that it has worse performance than Fast R-CNN when it comes to small object detection.
- Another drawback is that, because of its intricate data augmentation, SSD requires a lot of data for training, which, depending on the application, may be expensive and time-consuming.

The SSD and Fast R-CNN are tested and compared to a YOLOv3, the tests from different studies like Comparative Analysis of Deep Learning Image Detection Algorithms (Srivastava et al., 2021) showed that YOLOv3's overall performance surpassed the two algorithms used.

In addition to this, the latest YOLOv4 is tested using COCO datasets against other object detection approaches. The result is shown in the figure shown below.



(Source: YOLOv4 Paper)

Figure 3.5 *YOLOv4 comparison with other Object Detection using COCO datasets.*

Figure 3.4 showcases the improvement of YOLOv4 from YOLOv3 which shows the increase in AP without suffering the FPS. It also highlights the FPS of YOLOv4 against its competition in real-time object detection.

3.6 Conceptualization of the Application that will be used to Implement the Translator

When using the sign language translator function, the application will be using OpenCV to locate the hand from the rest of the camera footage. Once the hand is detected, YOLO will identify the hand gesture and translate it by matching it with the datasets. After training, the translation should appear as text on the tablet screen.

Other than the translator function, there is also the module which teaches the user basic words and phrases in Filipino sign language. There is also an optional voice-to-text feature that simply converts the user's words and sentences into text on the screen to allow the user to answer back if it needs to be.

3.7 FSL101: Filipino Dynamic Gestures Recognition Module

(Source: VERA Files - Filipino Sign Language)

Figure 3.6 *An image of a man performing the "NO" word in Filipino Sign Language*

The application contains a module for basic tutorials to those who want to learn about sign language fully. The intended feature contains common responses, like yes or no, and short phrases like, hello. The feature also contains the FSL alphabet and numbers.

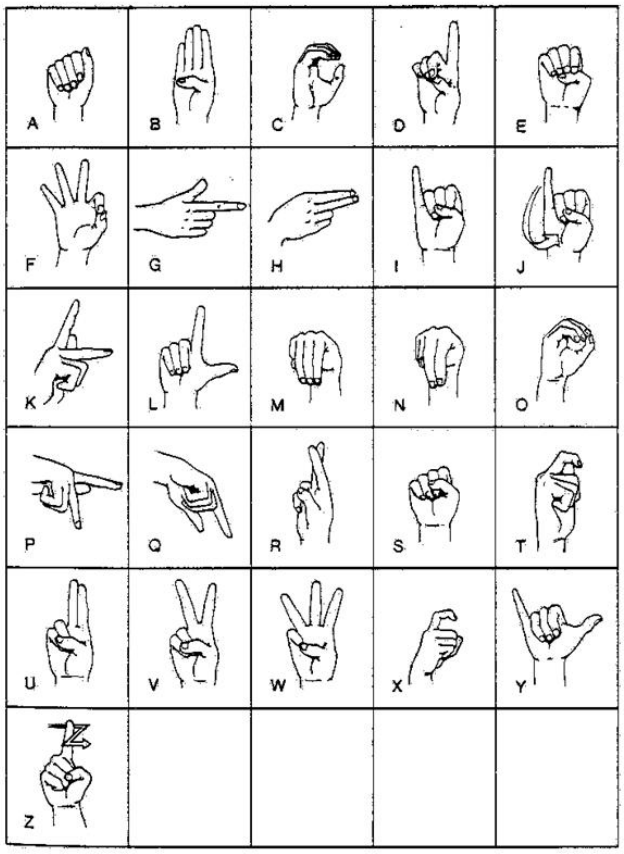


Figure 3.7 *Filipino Sign Language Alphabet*

3.8 Design of Overall System

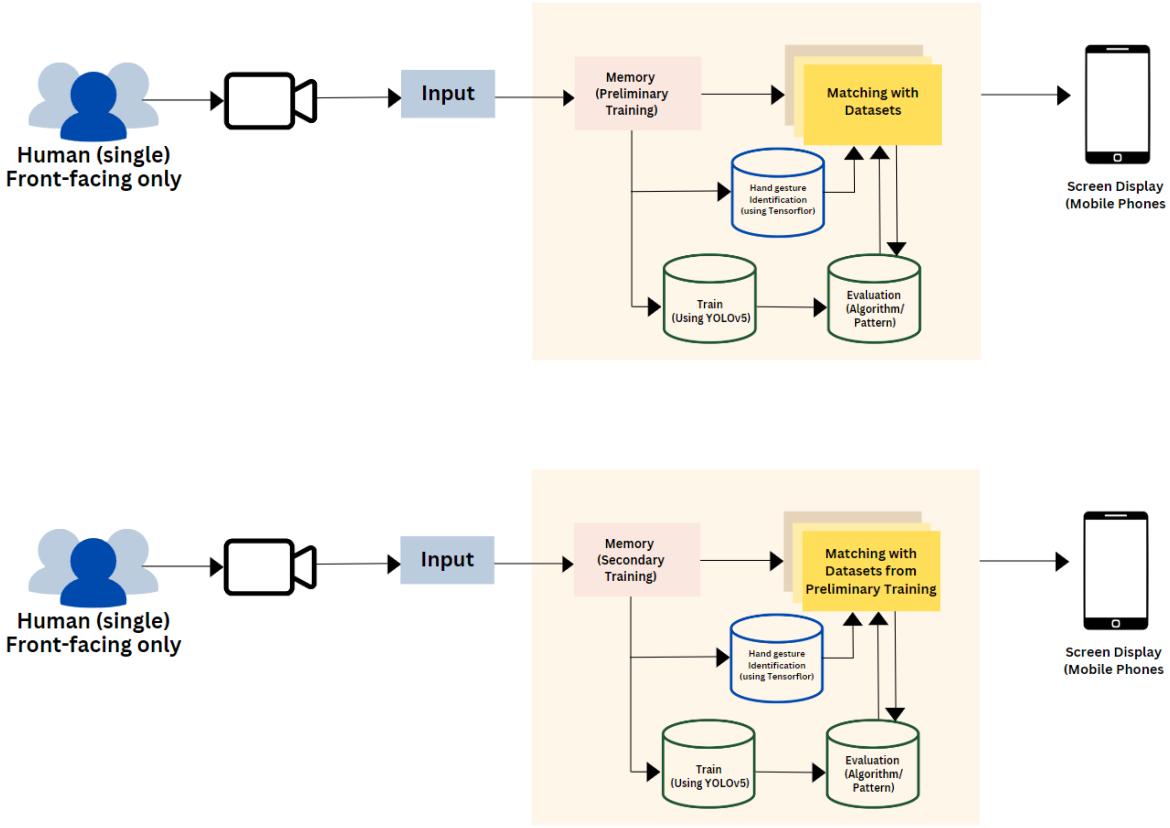


Figure 3.8 *Block Diagram of the Proposed Application during the Preliminary Training and the Secondary Training*

Figure 3.2 shows the proposed project's block diagram shows the preliminary training of the hand gesture recognition system and its secondary training.

The diagram shows that under the preliminary training, the system is trained using a computer set-up. Using the computer, the system is trained to identify the hand from an active camera footage using OpenCV. Once the hand is separated from the rest of the footage, the system is then trained using YOLO to

identify the gesture of the hand that is being presented by matching the gesture with the datasets from the memory. The outcome of the identification is then projected to a monitor for evaluation and correction of data.

Once the preliminary training is finished and the hand sign identification system works as intended, the hand sign identification system is then prepared to be integrated into an application for mobile devices, like tablets, which can be seen in the second block diagram.

On the second block diagram, the application undergoes the same training the computer did with an additional set of dynamic gestures. The dynamic gesture includes simple phrases that we use in everyday life.

3.9 Flowchart of the Application

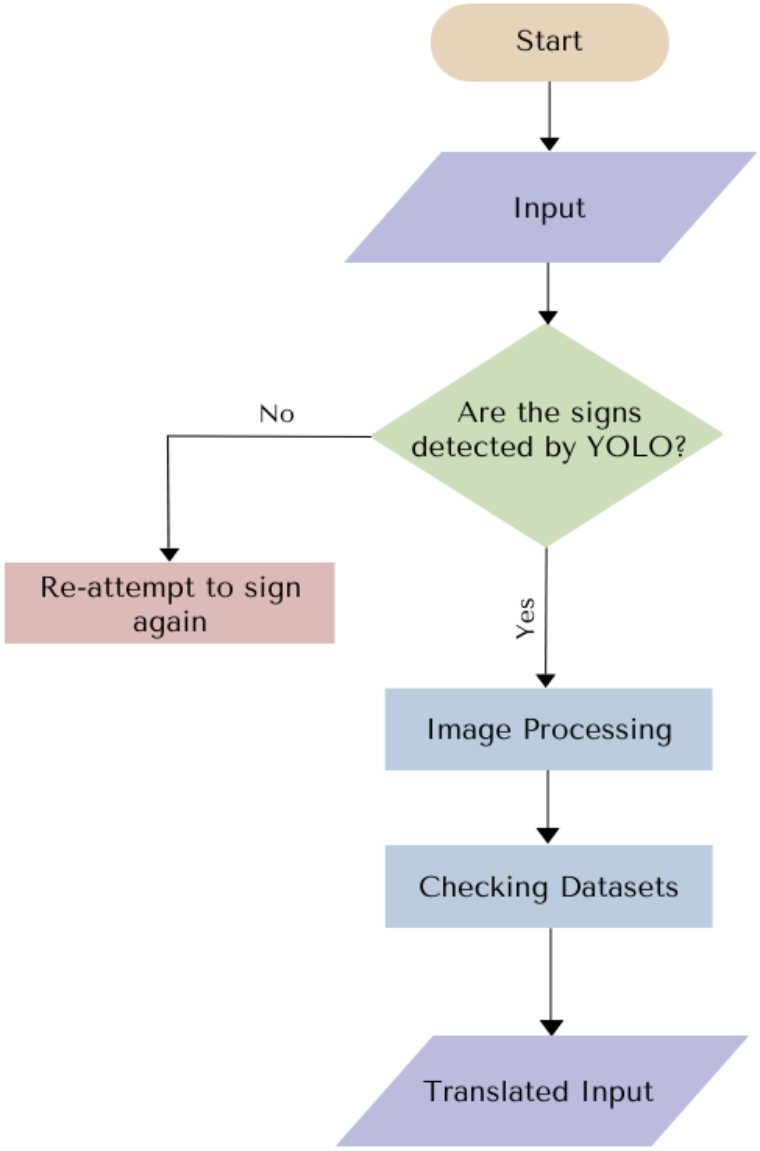


Figure 3.9 *Flow chart of the system*

3.10 Applying the ISO 9126 model to the evaluation of the system

An effective framework for evaluating software is created with the aid of the international standard ISO 9126 software. This common method of evaluating software may be divided into four (4) separate categories. These are employed to handle issues of various kinds. This program is widely and effectively used to accept several concepts and metrics. The suggested characteristics explain outwardly when software is discovered to be a result of internal software properties.

The Quality Model, External Metrics, Internal Metrics, and Quality in Use Metrics are the four categories that were mentioned.

It is possible to develop a mobile application that adheres to the universal model and makes it simpler to compare products by utilizing the four distinct categories of evaluating software quality.

Table 3.1 Testing of the System based on the ISO 9126 standards

Characteristic	Sub-characteristic	Description
Functionality	Suitability (F1)	Can the system carry out the necessary tasks?
	Accurateness (F2)	Are the system's outcomes what were predicted?
	Interoperability (F3)	Does the system respond to other systems?
	Security (F4)	Does the system guard against unauthorized entry?
	Maturity (R1)	Have the system's and the hardware's flaws been fixed over time?

Reliability	Fault tolerance (R2)	Does the system have the ability to handle errors?
	Recoverability (R3)	In the event of a failure, can the system function again and retrieve any lost data?
	Recoverability compliance (R4)	Does the system follow the reliability and quality criteria in place?
Usability	Understandability (U1)	Does the system user understand how to utilize it without any difficulty?
	Learnability (U2)	Is it possible to learn the system quickly?
	Operability (U3)	Can the system function with little effort?
	Attractiveness (U4)	Is the system well-designed?
	Usability compliance (U5)	Does the system adhere to current usability guidelines?
Efficiency	Time behavior (E1)	How fast is the system reacting?
	Resource utilization (E2)	Does the system make good use of its resources?
	Efficiency compliance (E3)	Does the system meet the current efficiency requirements?
Maintainability	Analyzability (M1)	Can the flaws be identified with ease?
	Changeability (M2)	Can the system be changed easily?
	Stability (M3)	Can the system still work once changes are made?
	Testability (M4)	Is it simple to test the system?

Portability	Adaptability (P1)	Can the system be transferred to different settings?
	Installability (P2)	Can the system be simply installed?
	Portability compliance (P3)	Does the system adhere to portability requirements?

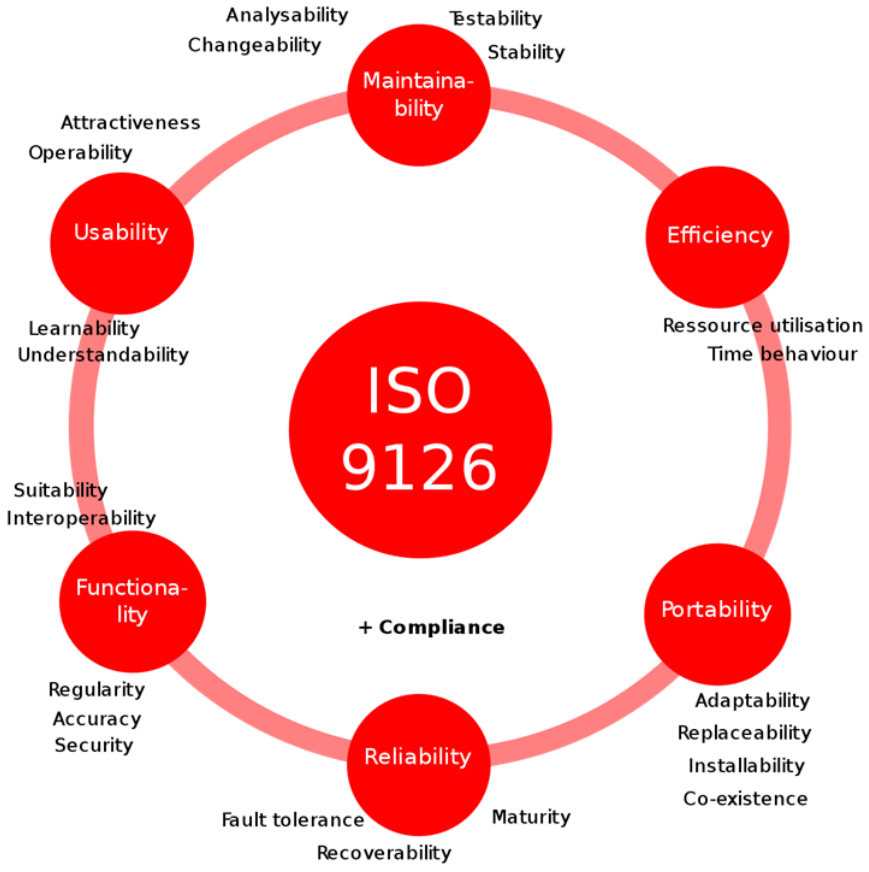


Figure 3.10 *Flow chart of the system*

3.11 Gantt Chart

	2021									2022										
Activities Description	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sept	Oct	Nov	Dec
Preparation of Chapter 1 to 3																					
Topic Defense																					
Consultation with Topic Adviser																					
Title Defense																					
Purchasing of Project Components																					
Further Research for the Project development																					
Utilization of Machine learning algorithms																					
Documentation of the Progress Defense																					
Progress Defense																					
Development of Mobile Application																					
Project Integration																					

Testing and Evaluating of the Project							■	■												
Pre-Final Defense									■											
Deployment of the Project										■	■	■	■							
Finalization of the Thesis Paper											■	■	■							
Final Defense													■							
Finalization of the Project Document and Bookbinding														■	■	■	■	■	■	■
Submission of Research Book															■	■	■	■	■	■

Table 3.2 *Gantt Chart for the Entire Study*

CHAPTER 4

PRESENTATION, ANALYSIS AND DISCUSSION OF DATA

The technical description, structural organization, interpretation of the data, and consequences of the study are presented in this chapter.

4.1 Technical Description of the Project

The technology developed for the project entitled "TensorFlow based Dynamic Filipino Sign Language Gesture Recognition using YOLO" aims to detect static and dynamic Filipino Sign Language Gestures that will translate into words and phrases. The detection uses a fast R-CNN called YOLO or You Only Look Once which is trained to convert the signs. The Mobile Application is called Pantomime and was built using Python, Android Studio and TensorFlow.

The users will also be able to access the tutorial or the learning modules attached in the application which can serve as reference and guide in performing sign languages. Inside the manual, it contains the basic words, phrases, alphabet, and numbers.

4.2 Project Structure

4.2.1 Preliminary Training of Datasets

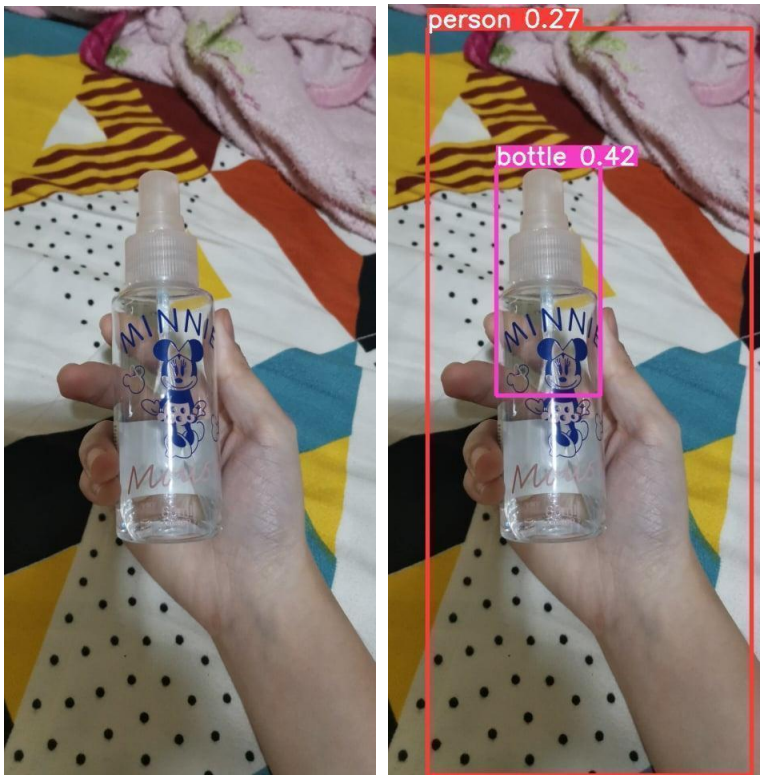
4.2.1.1 Utilizing YOLO's Object Detection


```
Fusing layers...
Model Summary: 224 layers, 7266973 parameters, 0 gradients
Image 1/1 C:\Users\User\Desktop\yolov5-master\ron2.jpg: 640x352 1 person, 1 cell phone, Done. (0.037s)
Speed: 2.0ms pre-process, 37.0ms inference, 25.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp3
```

Figure 4.1 *Test Image Results of the Python Script using YOLO and PyCOCO datasets and the original images in a hand holding a phone*

Figure 4.1 shows the results of the object detection and image recognition of the program with the use of YOLO and PyCOCO datasets. The first pair of images shows a hand holding a phone. The program creates boxes around the objects it identifies which, in the case of the first image, is the phone and the person holding it. The numerical value next to the guess represents the program's level of confidence in that guess.

The illustration displays the first pair of the results summary. It also displays the processing time, which was 0.037 seconds.

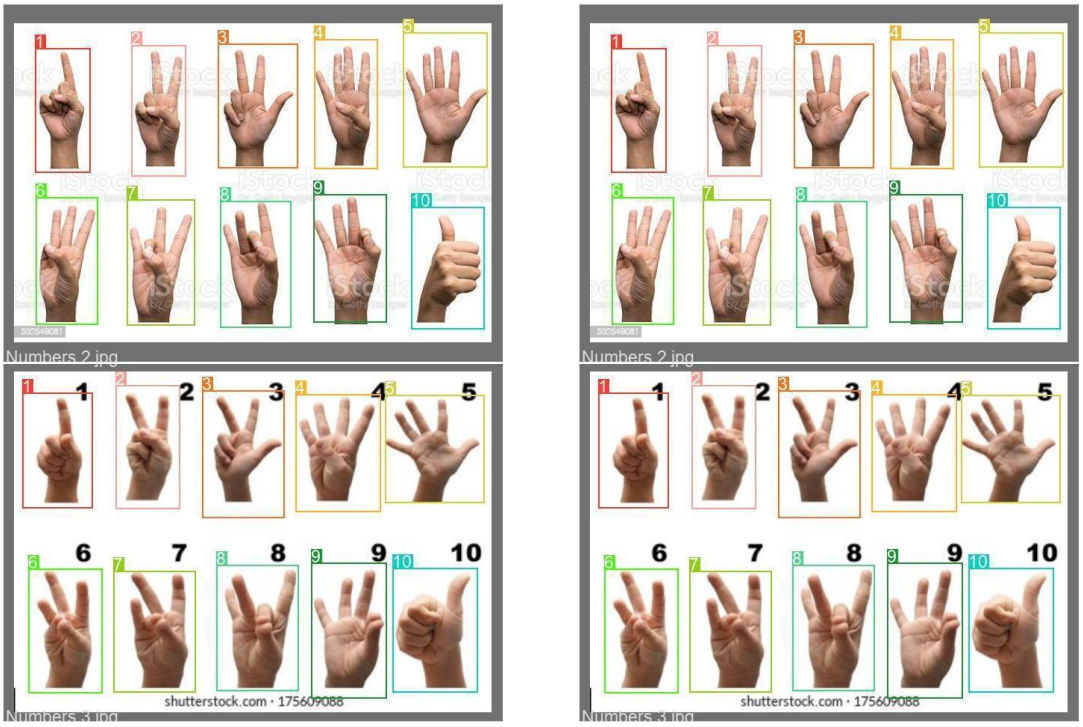



```
Fusing layers...
Model Summary: 224 layers, 7266973 parameters, 0 gradients
image 1/1 C:\Users\User\Desktop\yolov5-master\ella.jpg: 640x320 1 person 1 bottle Done. (0.031s)
Speed: 0.0ms pre-process, 31.2ms inference, 0.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp4
```

Figure 4.2 *Test Image Results of the Python Script using YOLO and PyCOCO datasets and the original images in a hand holding a bottle*

The second pair of images shows a hand holding a bottle. The program does the same method where it guesses the objects in the photo. The results show that the program guessed that there is 1 person and 1 bottle, which is correct.

4.2.1.2 Applying YOLO in the Static Datasets




```
  Epoch    gpu_mem       box       obj       cls    labels  img_size
535/599     1.07G     0.042   0.05242   0.05961        61       640: 100% 1/1 [00:00<00:00,  2.43it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  9.1
             all          2         20      0.276      0.548      0.321      0.228
               1          2          2      0.144        0.5      0.199      0.119
               2          2          2          1          0      0.249      0.181
               3          2          2       0.11          1      0.153     0.0986
               4          2          2      0.294        0.5      0.314      0.251
               5          2          2      0.191          1      0.995      0.746
               6          2          2          0          0      0.117     0.0772
               7          2          2      0.318      0.477      0.332      0.217
               8          2          2      0.194        0.5      0.331      0.203
               9          2          2      0.176          1      0.276      0.227
              10          2          2       0.33        0.5      0.242      0.155
lyStopping patience 100 exceeded, stopping training.
```

Figure 4.3 *Image Results of the First Round Test using a Custom dataset*

Figure 4.3 shows the image results of the first round of tests using the custom datasets which are the hand signs of 1 to 10.

The image to the left shows the expected result of the test and the image to the right shows the outcome of the test. The only hand sign recognized by the program is the hand

sign of number 5. This supported by the image showing that the 99.5% of the runs recognized the hand sin whilst the others maxed mostly around 30%.

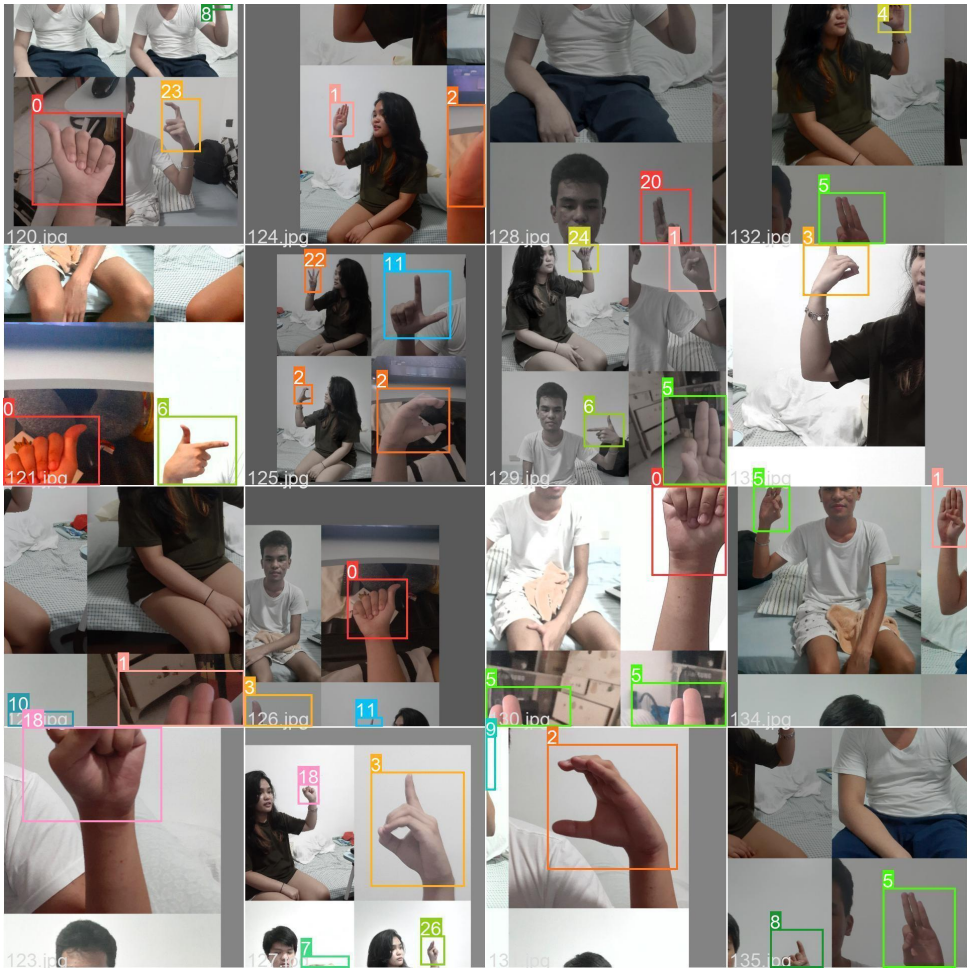
The first run was performed by a small amount of dataset which could be the main reason for the low results.

Table 4.1: Letter Recognition Accuracy

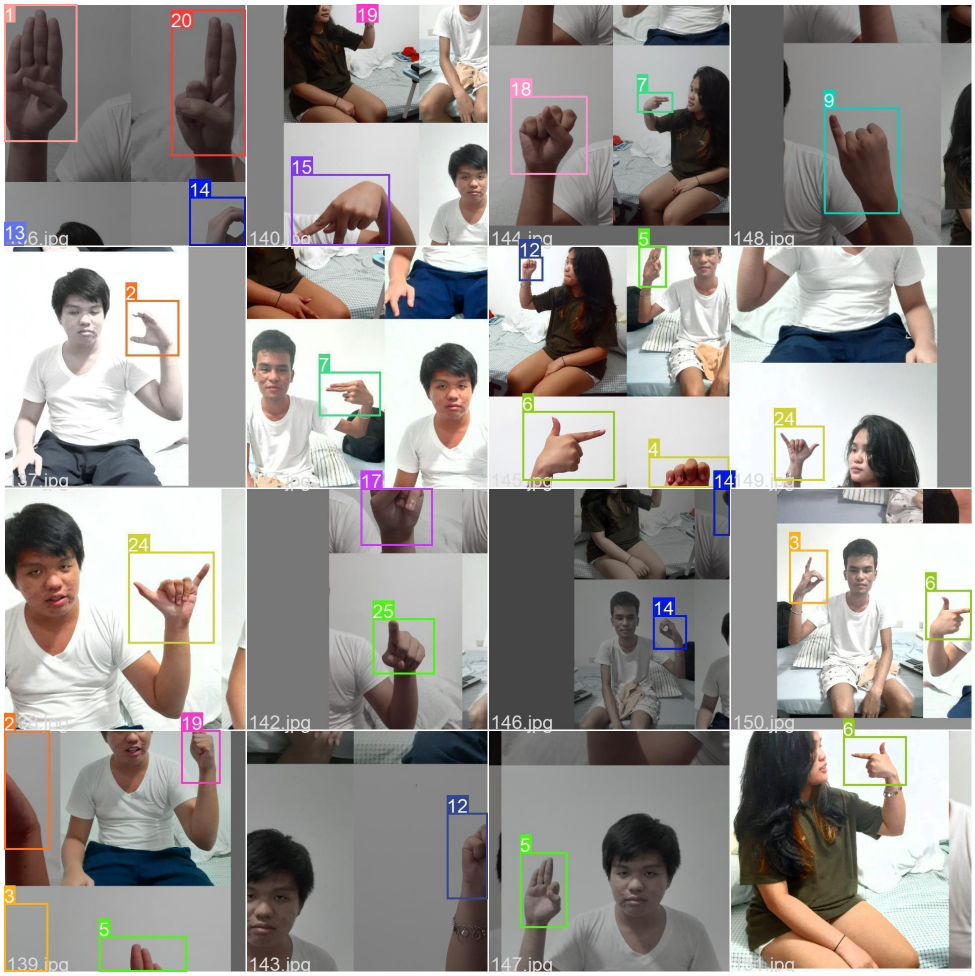
Letter	Preliminary Training of Tensorflow based Dynamic Filipino Sign Language Gesture Recognition using YOLO		Compared to the Previous Study: Static Sign Language Recognition Using Deep Learning	
	Accuracy (%)	Average Time (s)	Accuracy (%)	Average Time (s)
A	97	0.027	100	2.02
B	97	0.027	91.11	4.01
C	97.1	0.028	100	2.2
D	96.9	0.027	100	2.46
E	97.7	0.027	96.67	3.59
F	96.3	0.028	93.33	4.95
G	96	0.027	98.89	2.85
H	97.5	0.027	84.44	5.39
I	96.5	0.027	94.44	3.37
J	98	0.028	90	4.57
K	97.1	0.027	94.44	4.06
L	97	0.028	98.89	2.38
M	96.3	0.027	82.22	5.98
N	98.2	0.027	90	3.92
O	99	0.028	90	3.62
P	96.5	0.027	86.67	4.97

Q	100	0.027	95.56	3.76
R	96.3	0.027	78.89	6.24
S	99.1	0.028	86.67	4.6
T	100	0.027	74.44	7.62
U	100	0.027	85.56	5.11
V	99.6	0.027	84.44	5.07
W	96.9	0.027	96.67	2.46
X	97.7	0.027	86.67	4.98
Y	97.2	0.027	93.33	3.5
Z	96.4	0.027	67.78	8.31
Ñ	96.6	0.027	-	-
Overall Rating	97.6%	0.02722	90.04%	4.31

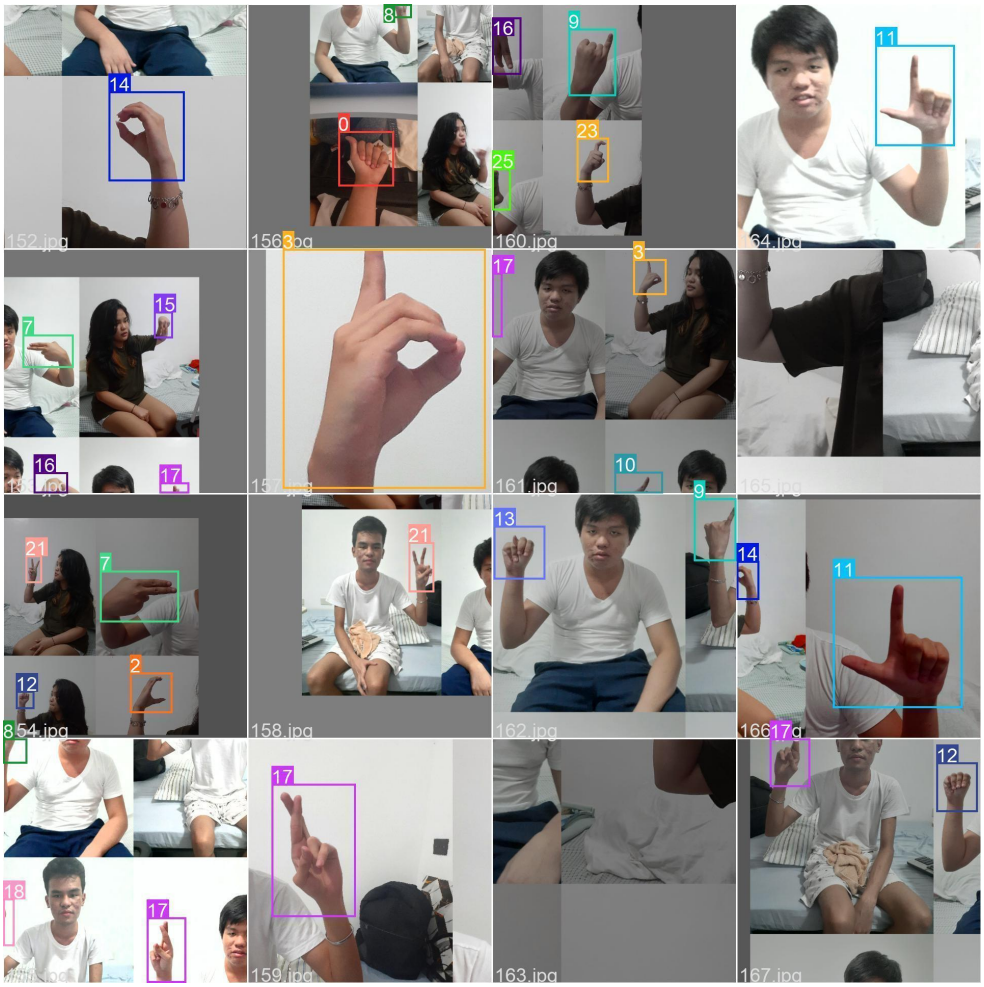
Table 4.1 shows the accuracy rate of each letter and its corresponding average time and was compared to the previous study with the same criteria



(a)



(b)

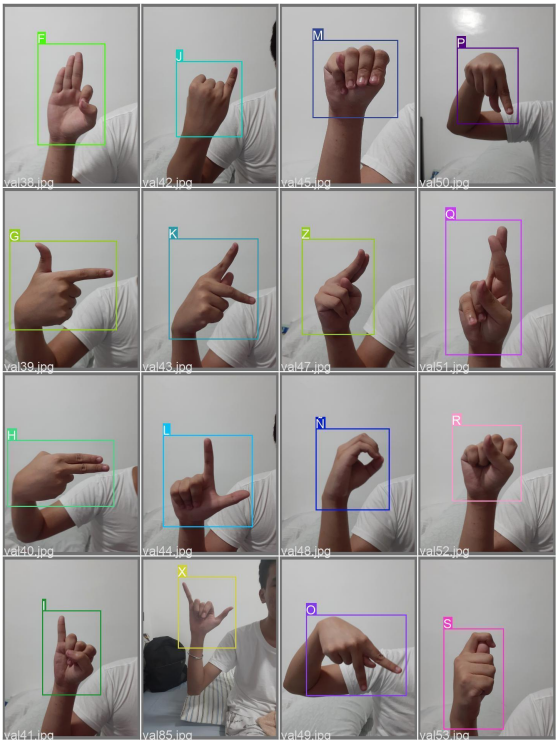


(c)

Figure 4.4 *The collected letter dataset on which the system will be based (a) batch 0 (b) batch 1 (c) batch 2 of the database*

(a)

(b)



(c)

(d)

Figure 4.5 *Image Results of the Second Round Test Using the Collected Static Dataset (a) dataset that was identified as a hand sign from the Figure 4.4 (b) Validation result from the system (c) dataset that was identified as a hand sign from the Figure 4 (d) Validation result from the system*

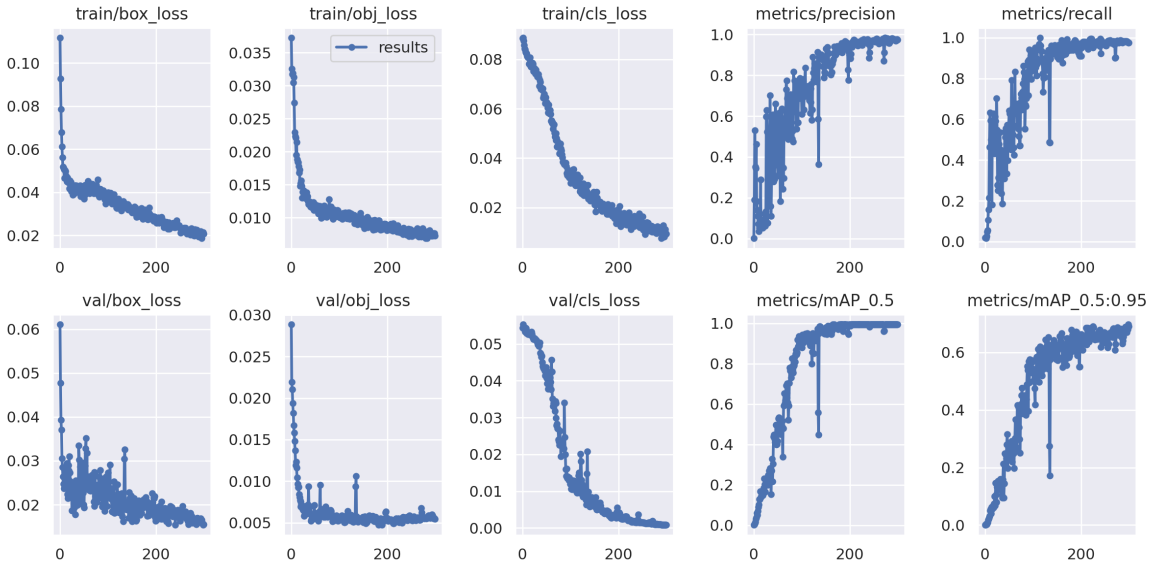


Figure 4.6 *Plots of box loss, object_loss, classification loss, precision, recall, and mean average precision (mAP) over the training dataset and validation set*

In Figure 4.6, it is possible to comprehend how the respective losses are evolving with each iteration by looking at the graph's lower box_loss and object_loss values. The box loss measures how accurately the algorithm can pinpoint an item's center and how completely the anticipated bounding box encloses an object. The reverse of objectness, Object_loss, quantifies the likelihood that an object will exist inside a suggested zone of interest. How successfully the algorithm can determine the proper class of a given item is indicated by Classification_loss.

Precision, mean average location, and other metrics for the model are excellent. The validation data's box, object_loss, and classification losses all quickly improved.

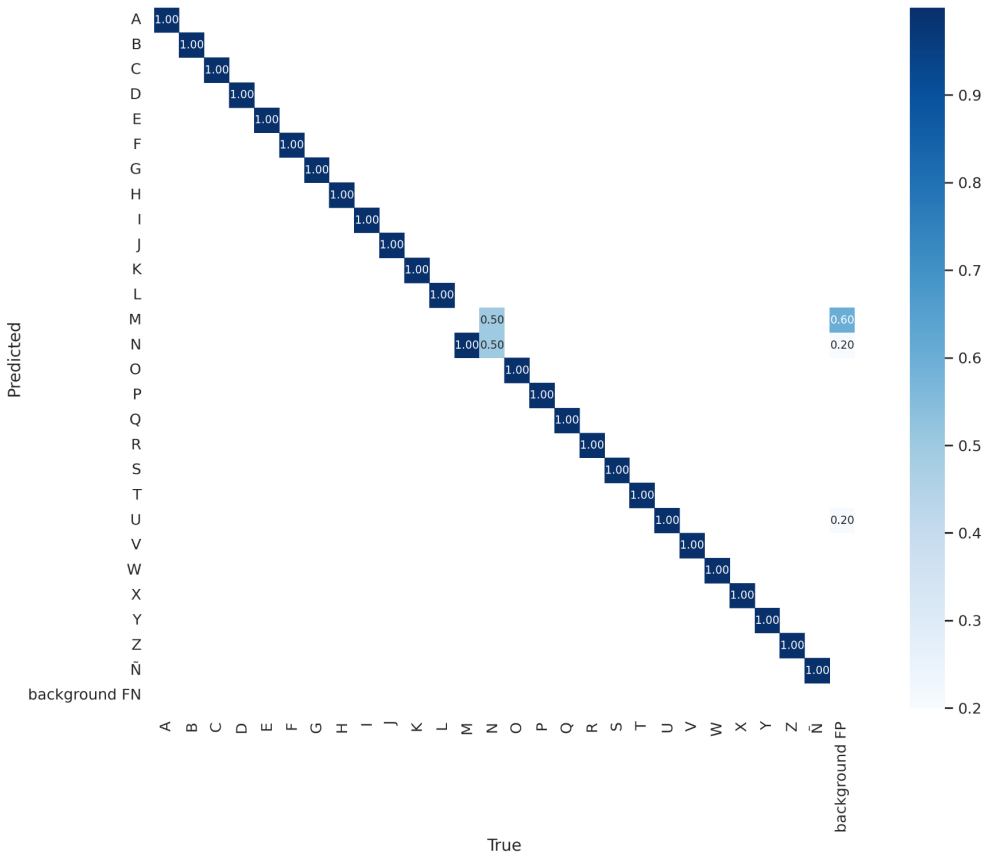


Figure 4.7 *Confusion Matrix of the Static Letters*

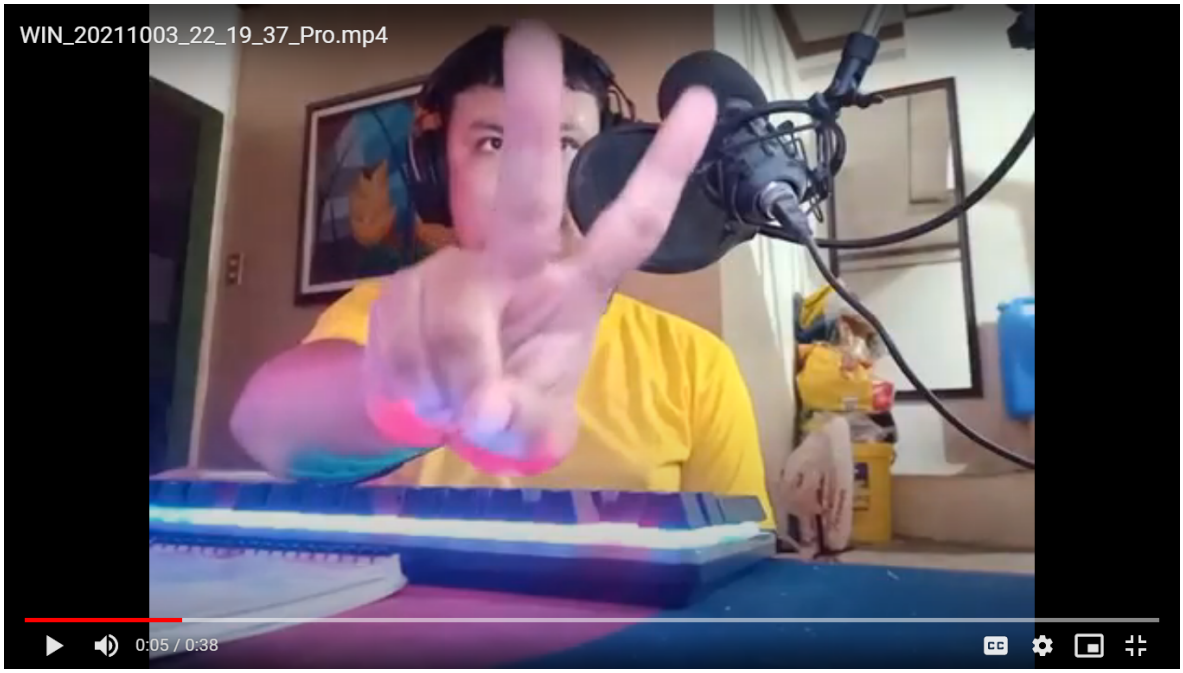
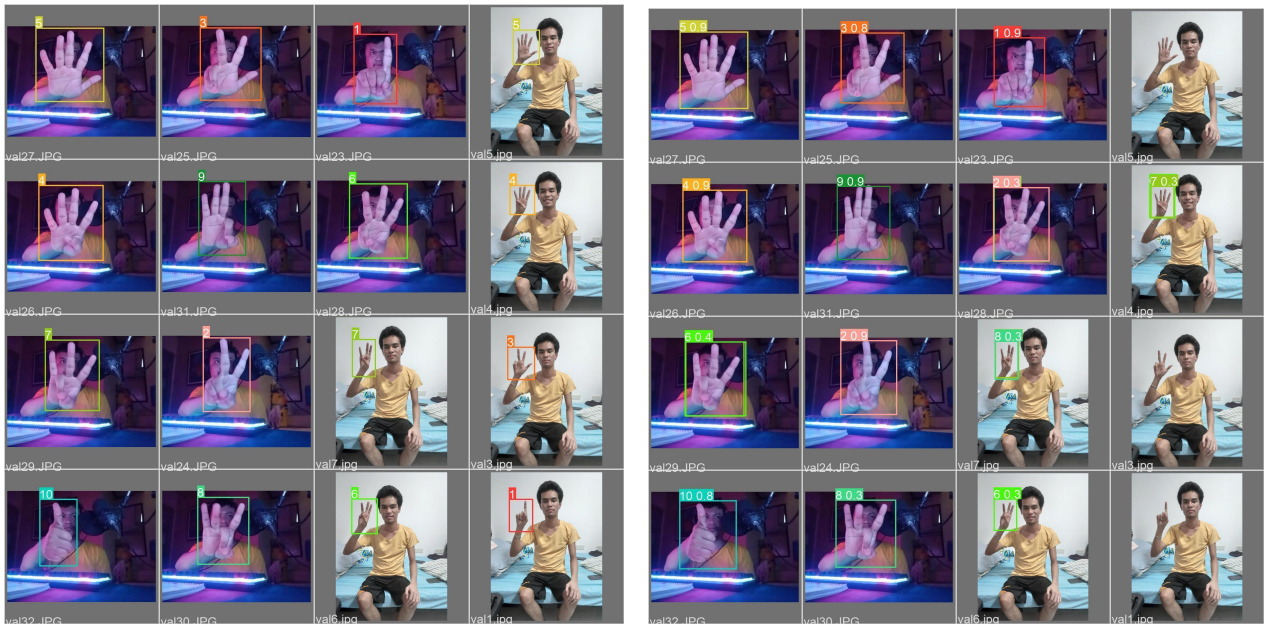


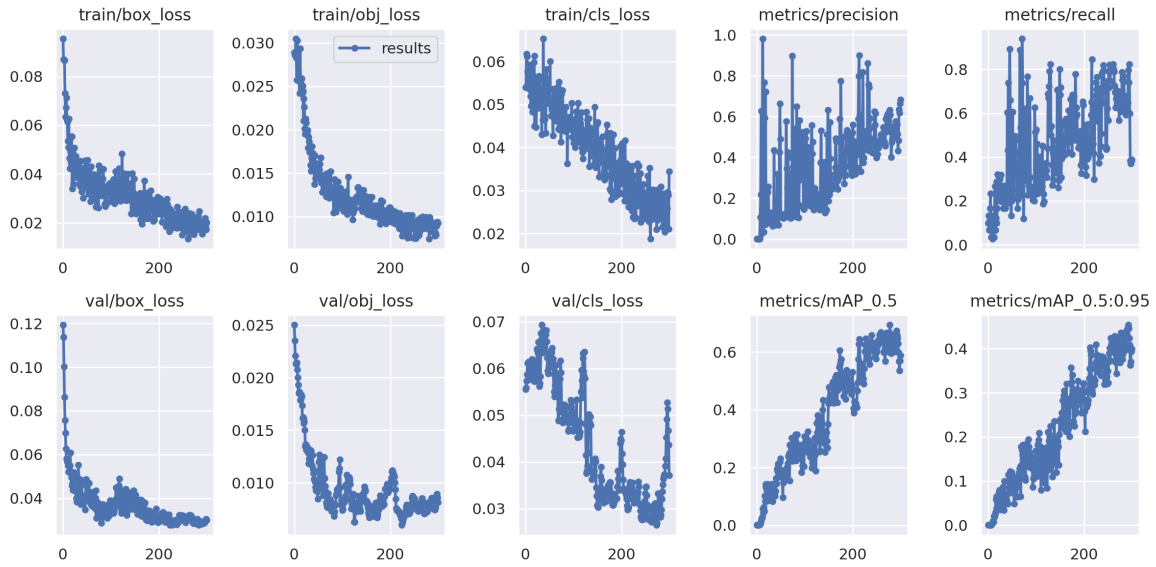
Figure 4.8 *Image of the footage captured for the Learning Module*

Figure 4.8 shows the image of the footage and data concept for the development of the Learning Module aiming to learn the different hand signals. It will be integrated into the Mobile Application.

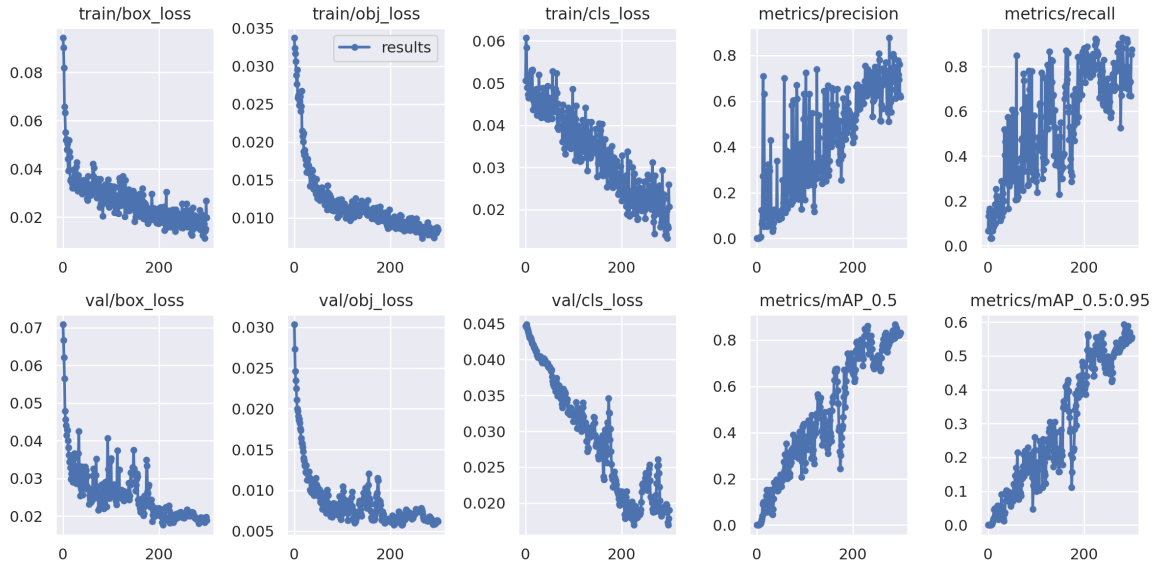


(a) (b)

Figure 4.9 *Image Results of the Second Round Test Using the Collected Static Number Dataset (a) dataset that was identified as a hand sign from the Figure 4.4 (b) Validation result from the system*



(a)



(b)

Figure 4.10 *Plots of box loss, object_loss, classification loss, precision, recall, and mean average precision (mAP) over the training dataset and validation set (a) before (b) after*

4.2.2 Creating Mobile Application

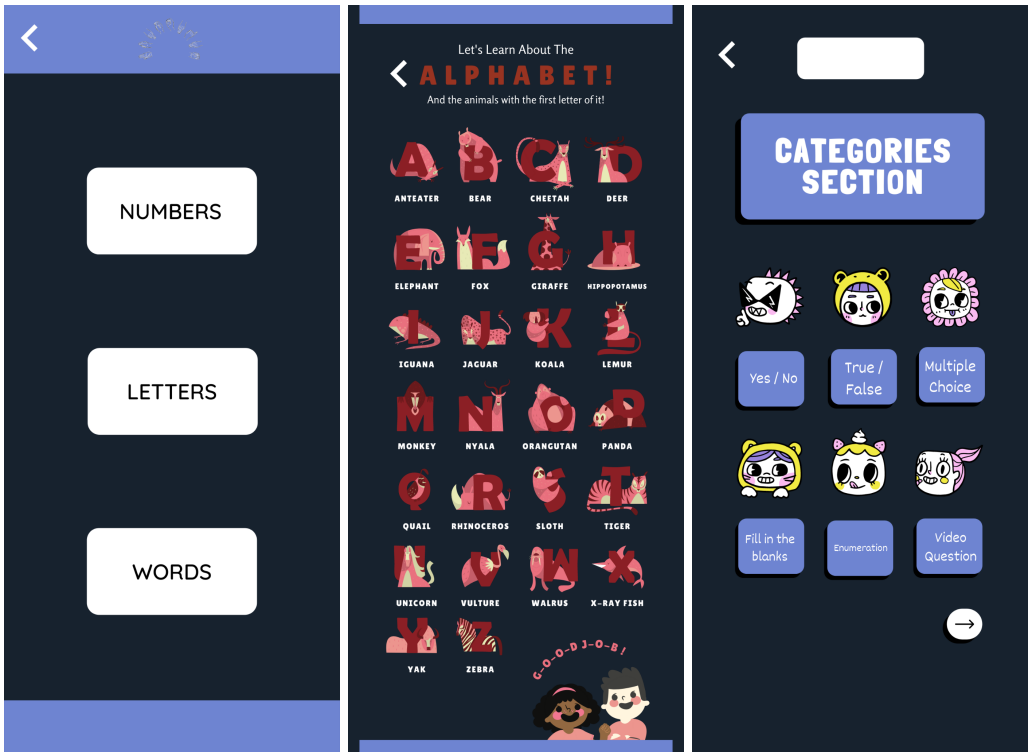
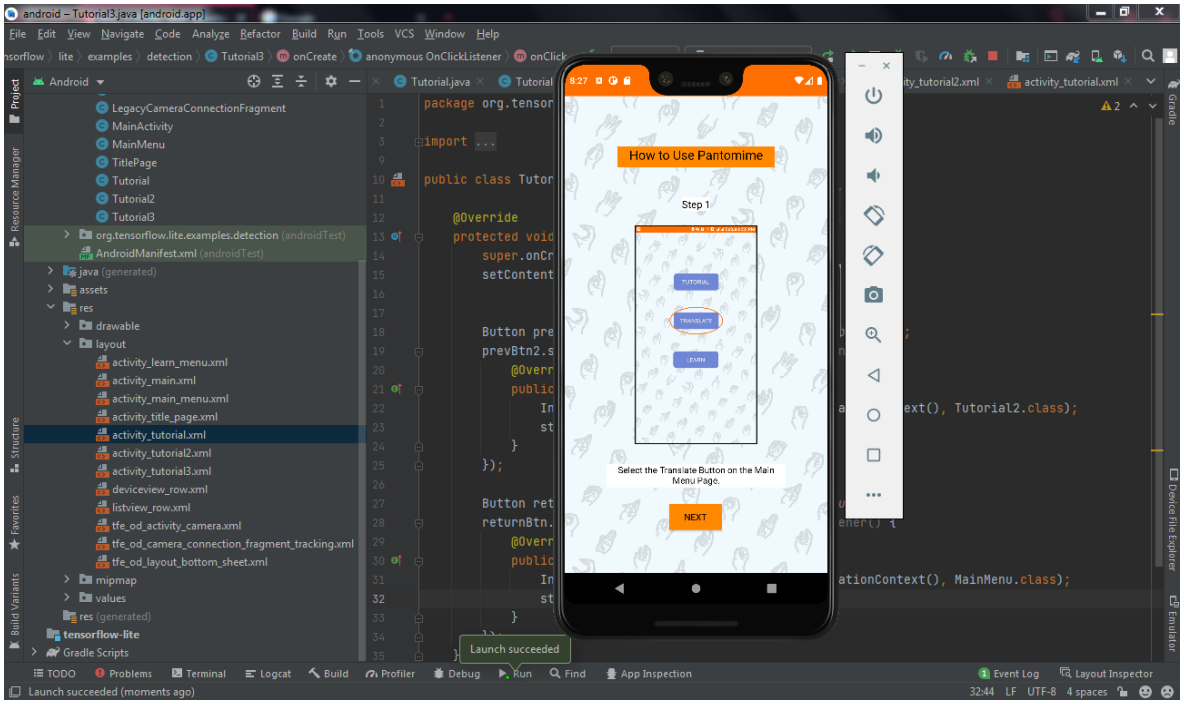
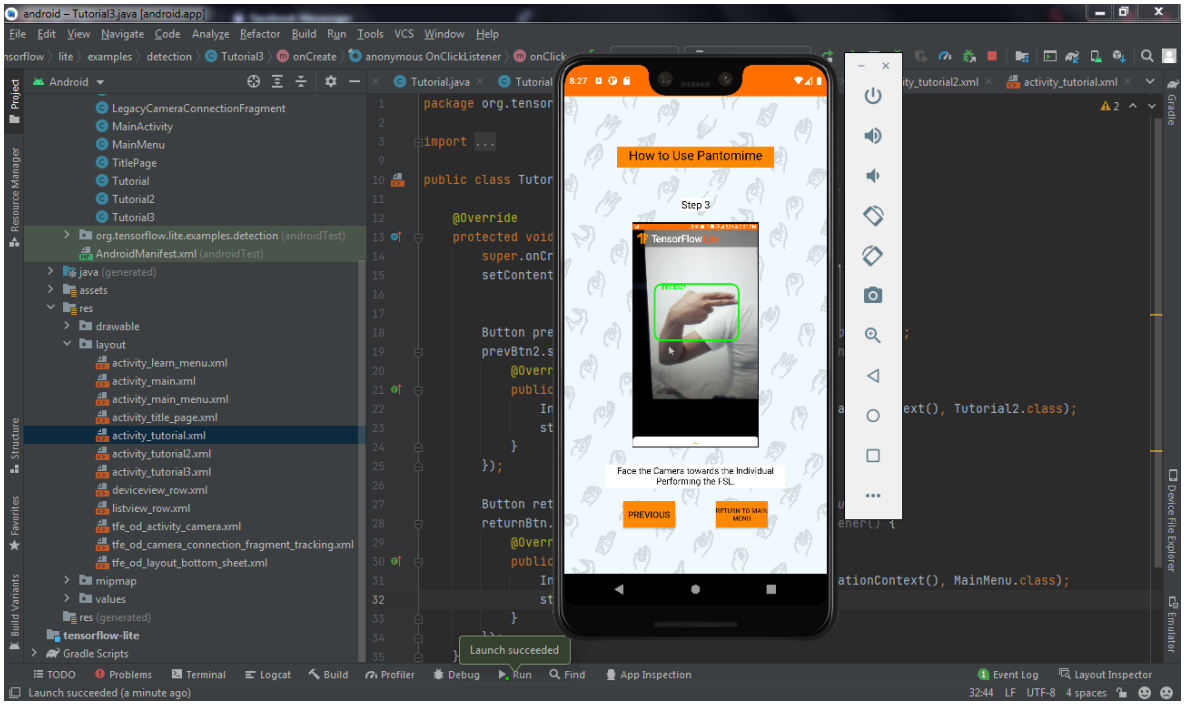


Figure 4.11 *Design Concept of the Learning Module into the Mobile Application*


How to Use Pantomime
Step 1
Select the Translate Button on the Main Menu Page.
NEXT
Launch succeeded


How to Use Pantomime
Step 3
Face the Camera towards the Individual Performing the FSL
PREVIOUS
RETURN TO MAIN MENU
Launch succeeded

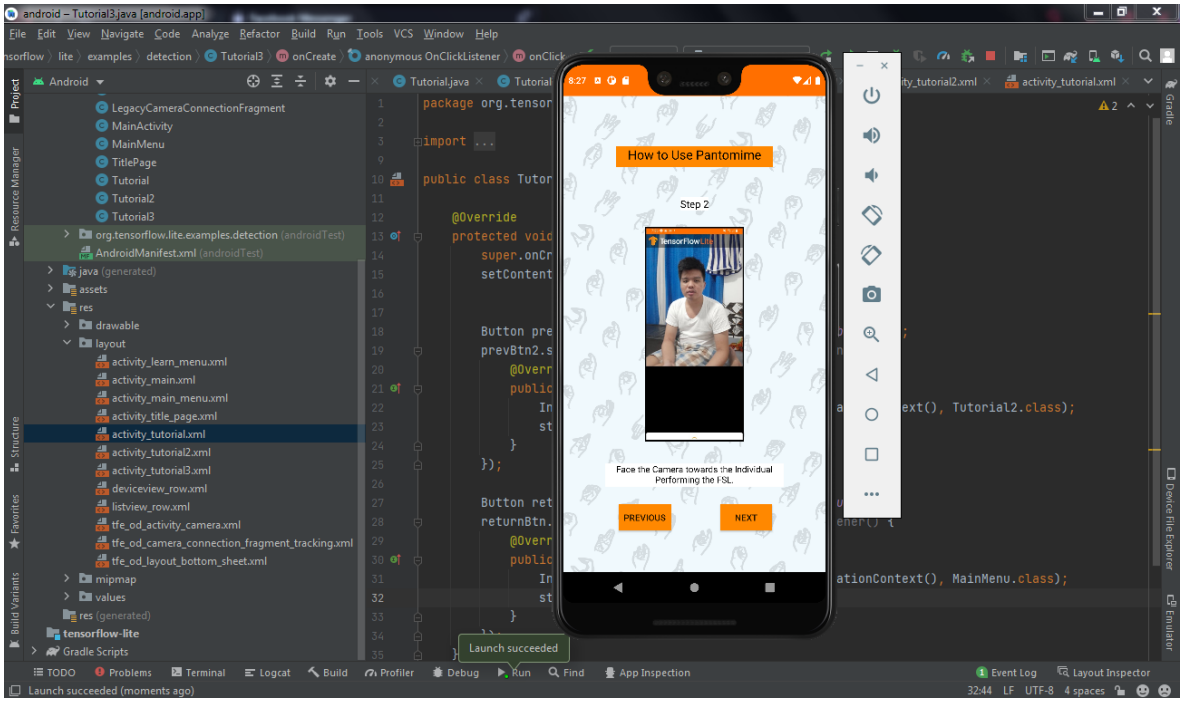


Figure 4.12 *Image of Integrating the Design Concept of the User Interface into Android Application with Tutorial for Usage*

Figure 4.12 shows the image of testing and integrating the design concept of the user interface along with the trained datasets. Included the tutorial on how to use the application. TensorFlow, Python, and Android Studio would be used to design the final interface.

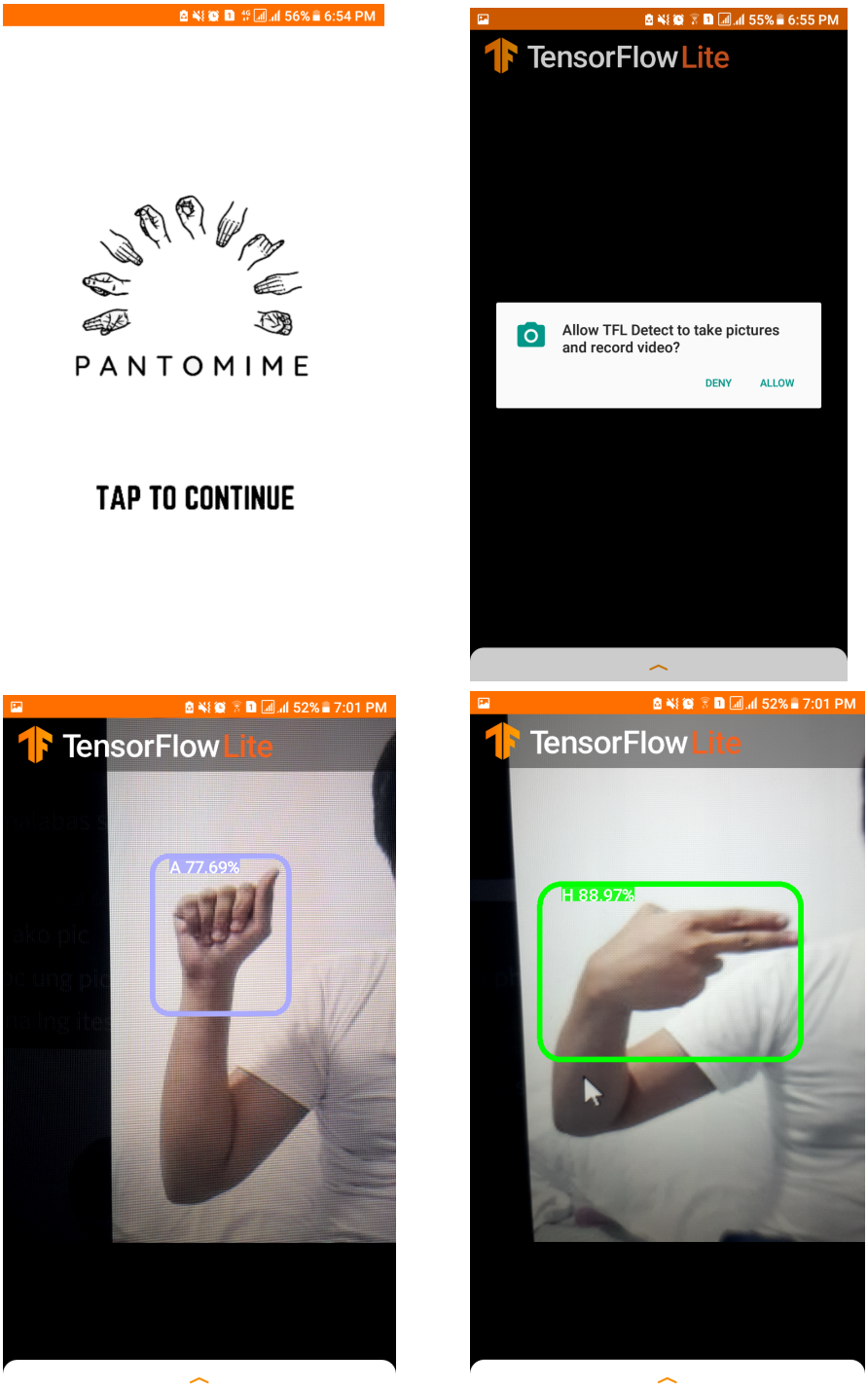
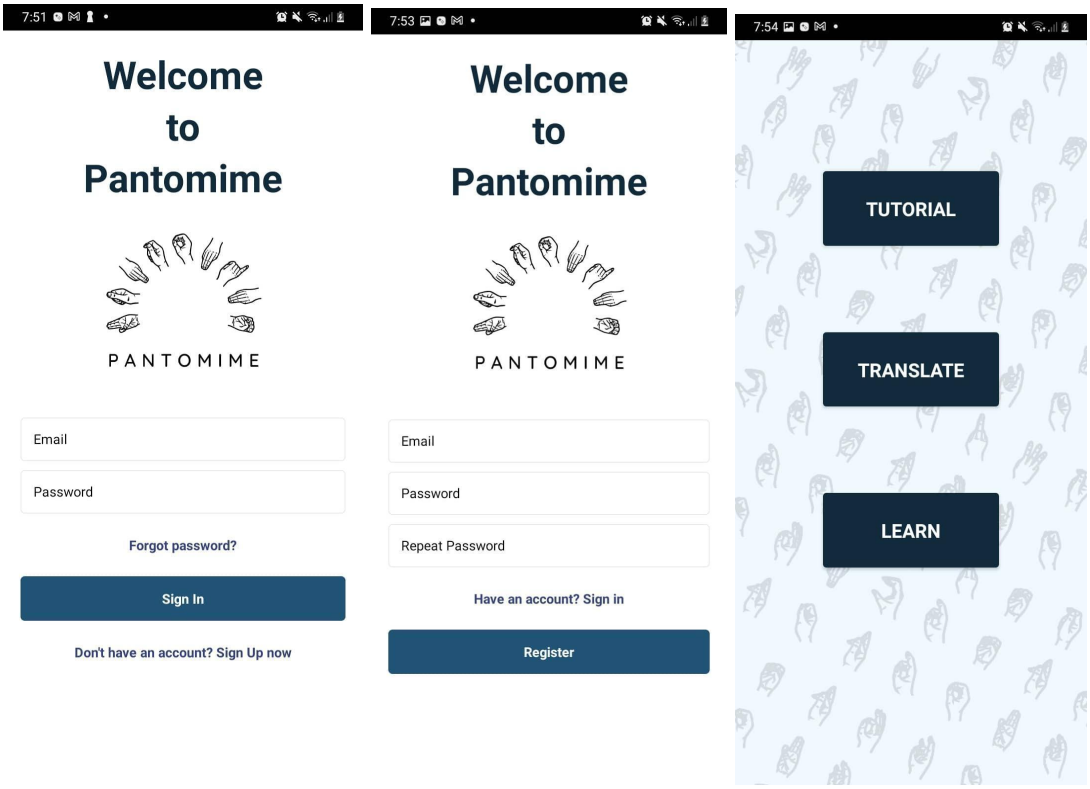


Figure 4.13 *Images of the camera and detector test using an actual android device of one of the researchers*

Figure 4.13 shows the first actual android application interface with the researcher's hand.

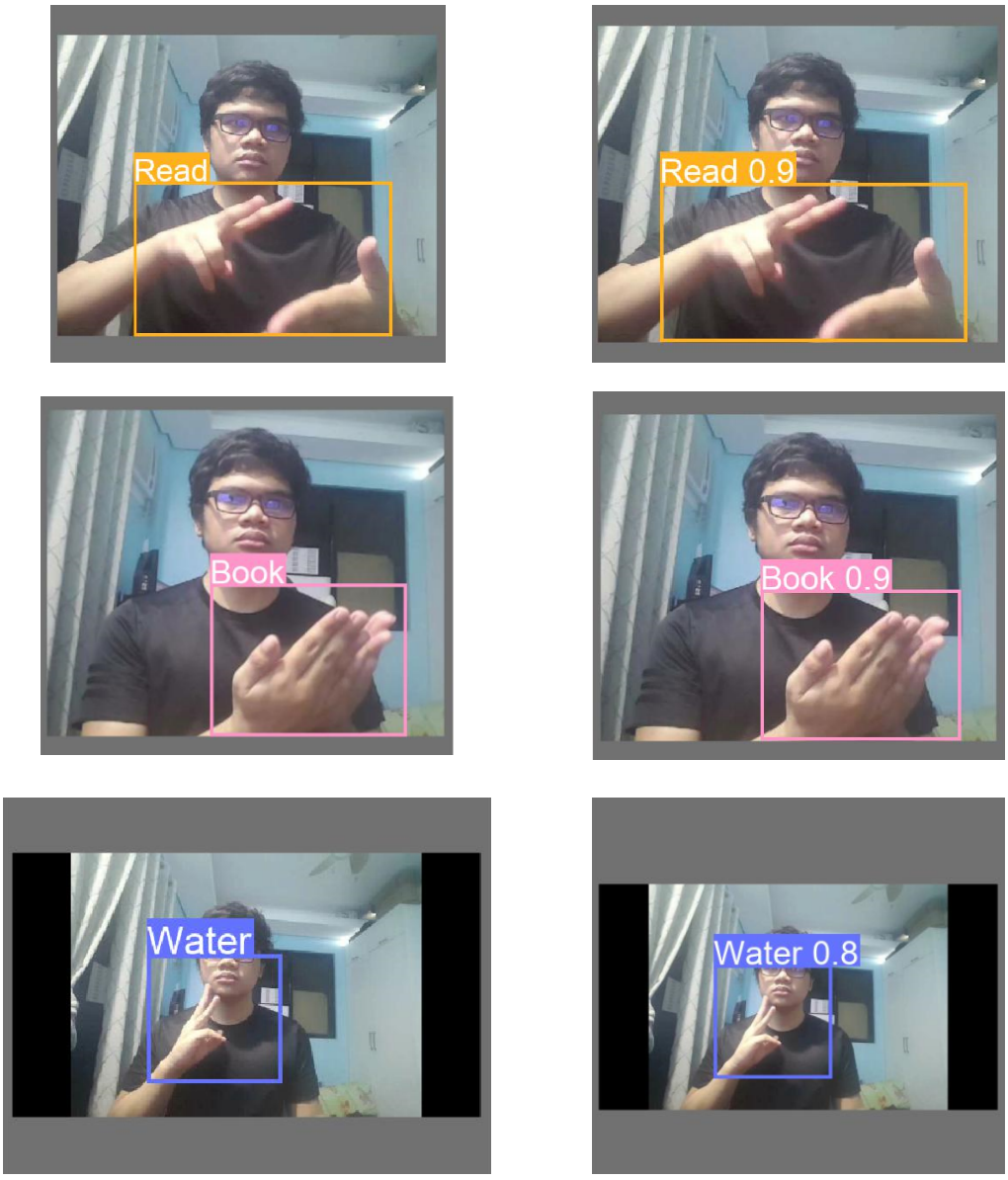


(a) (b) (c)

Figure 4.14 *Images of the New User Interface of the App*

Figure 4.14 shows images of the new user interface of the mobile application using an emulator. (a) the login screen, (b) the sign up menu and (c) the main menu.

4.2.3 Secondary Training of Datasets



(a) **(b)**

Figure 4.15 *Image Results of a Test using Collected Dynamic Dataset. (a) the images on the left are images to be identified during the training, and (b) the images to the right are results of the training.*

Figure 4.15 shows the image results during the secondary training.As shown, the hand signs that correspond to read, book, and take care are recognized by the system.

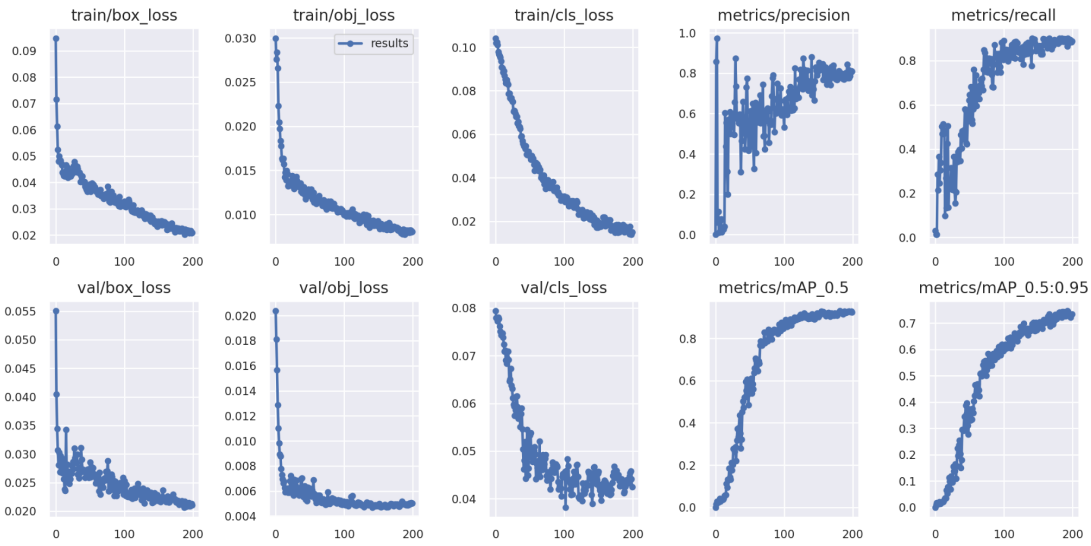


Figure 4.16 *Plots of box loss, object_loss, classification loss, precision, recall, and mean average precision (mAP) over the training dataset and validation set from the secondary training.*

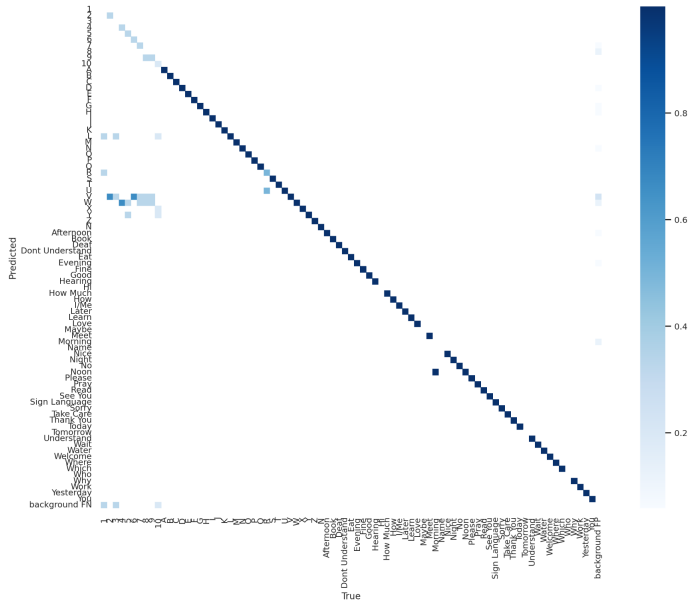


Figure 4.17 *Confusion Matrix of the Entire Dataset*

CHAPTER 5

SUMMARY OF FINDINGS, CONCLUSION, AND RECOMMENDATION

This chapter provides an overview of the study's outcomes, including findings, conclusions, and suggestions.

5.1 Summary of Findings

Results of Evaluation based on the Accuracy of the Translation	
Hand Sign	Average Accuracy Score based on the Evaluation (1 -5)
1	4
2	4
3	4
4	3
5	2
6	3
7	3
8	2
9	4
10	5
A	5
B	5
C	5
D	5
E	5
F	5

G	5
H	5
I	5
J	5
K	5
L	5
M	2
N	3
Ñ	5
O	5
P	5
Q	4
R	5
S	5
T	5
U	4
V	5
W	4
X	5
Y	5
Z	4
Afternoon	4
Book	5
Deaf	5
Don’t Understand	3

Eat	5
Evening	5
Fine	5
Good	5
Hearing	5
How much-	4
How-	4
Language	5
Later	5
Learn	5
Love one another	3
Maybe	5
Meet	4
Morning	5
Name	5
Nice	5
Night	5
No	5
Noon	2
Please	5
Pray	5
Read	3
See You	5
Sign	5
Sorry	5

Take Care	5
Thank You	5
Today	4
Tomorrow	5
Two Hands	4
Understand	5
Wait	5
Water	5
Welcome	5
Where	4
Which	4
Who	5
Why	4
Yesterday	5

Table 5.1 *Results of Evaluation based on the Accuracy of the Translation*

5.2 Conclusions

The study's primary objective was to create a system that could convert static and dynamic indicators into their corresponding word equivalents. A first training period and a subsequent training phase made up the research. The Preliminary Stage emphasizes static symbols like the alphabet and numbers. The system places a focus on mobile applications and dynamic gestures throughout the secondary stage.

One of the primary objectives of the study is to surpass the accuracy and efficiency of previous deep learning-based studies. Our system achieved a training accuracy of about 99%, a testing accuracy of 97.6% for static signs, and an average time of 0.02722 seconds. For dynamic gestures, it has accuracy of 89.25% with an average time of 0.02812 seconds.

We also compared YOLO to its older versions and demonstrated that Yolov5 is significantly more dependable and quicker than its predecessor. Yolov5 is analogous to the machine learning techniques utilized by the other study. With the aid of YOLO, we were able to identify the provided signs without using gloves or hand markings, as was done in the previous study.

5.3 Recommendation

Based on the research findings, and for the further improvement of this study, the researchers proposed the following recommendations:

1. For the dynamic accuracy, continuous training of data sets can improve the overall accuracy but it occupies more storage space in the cloud which made the group upgrade the subscription with the Roboflow.

2. Speech and facial recognition can be added so it can provide a true to life translation of some signs or a sentence based translation.

3. A better UI for the android application and web browser site for PC/MAC users.

REFERENCES

1. Deafness and Hearing Loss. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss
2. Redmon J.; Divvala, S.; Girshick, R.; Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27-30 June 2016.
3. Wu, D.; Liu, Y.; Tao, C. A Universal Accelerated Coprocessor for Object Detection Based on RISC-V. Electronics 2023, 12, 475. https://doi.org/10.3390/electronics12030475.
4. Bachani, S.; Dixit, S.; Chadha, R.; Bagul, A. International Research Journal of Engineering and Technology (IRJET) 2020, Volume 7, Issue 4, pp. 585-586.
5. Kraljević, L.; Russo, M.; Pauković, M.; Šarić, M. A Dynamic Gesture Recognition Interface for Smart Home Control based on Croatian Sign Language. Appl. Sci. 2020, 10, 2300. https://doi.org/10.3390/app10072300.
6. Tolentino, L.K.S.; Juan, R.O.S.; Thio-Ac, A.C.; Pamahoy, M.A.B.; Forteza, J.R.R.; Garcia, X.J.O. Static Sign Language Recognition Using Deep Learning. Int. J. Mach. Learn. Comput. 2019, 9, 821–827.
7. Tolentino, L.K.S.; Jorda Jr., R.; Fortaleza, B.N.; Evanculla, A.DM.; Paez, M.A.V., Velasco, J.S. International Journal of Emerging Trends in Engineering Research 2020, Volume 8, Issue 9, pp. 5233-5237
8. Bochkovskiy, A.; Wang, C.; & Liao, H.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. ArXiv, 2020 abs/2004.10934.

9. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Computer Vision and Pattern Recognition 2016. https://doi.org/10.48550/arXiv.1506.0149
10. Girshick, R. Fast R-CNN. IEEE Internation Conference on Computer Vision (ICCV), Santiago, Chile, 07-13 December 2015.
11. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Amsterdam, The Netherlands, 8–16 October 2016; Volume 9905 LNCS, pp. 21–37
12. Fergus, M.D.Z. Visualizing and Understanding Convolutional Networks, European Conference on Computer Vision, 2013
13. Li, Z.; Tian, X.; Liu, X.; Liu, Y.; Shi, X. A Two-Stage Industrial Defect Detection Framework Based on Improved-YOLOv5 and Optimized-Inception-ResnetV2 Models. Appl. Sci. 2022, 12, 834.
14. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, 21–26 January 2017; pp. 2117–2124.
15. Liu, S.; Qi, L.; Qin, H.; Shi, J.; Jia, J. Path Aggregation Network for Instance Segmentation. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8759–8768

16. Dewi, C.; Chen, R.C.; Yu, H.; Jiang, X. Robust Detection Method for Improving Small Traffic Sign Recognition Based on Spatial Pyramid Pooling. J. Ambient. Intell. Humaniz. Comput. 2021, 12, 1–18.
17. Yuan, D.L.; Xu, Y. Lightweight Vehicle Detection Algorithm Based on Improved YOLOv4. Eng. Lett. 2021, 29, 1544–1551

APPENDICES

APPENDIX A

PROGRAM CODE

Appendix A. Program Code

Install Requirements

```
git clone https://github.com/ultralytics/yolov5  # clone
cd yolov5
pip install -r requirements.txt  # install
```

Run Inference using detect.py with Trained Weights

```
python detect.py --weights model.pt --img 416 --conf-thres 0.7 --source video.mp4
```

Training

```
python train.py --img 416 --batch 128 --epochs 350 --data data.yaml --weights yolov5s.pt
--cache
```

Pantomime Flask

```
import argparse
import os
from runpy import run_module
import sys
from pathlib import Path

import torch
import torch.backends.cudnn as cudnn

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative
```

```
from models.common import DetectMultiBackend
from utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from utils.general import (LOGGER, check_file, check_img_size, check_imshow,
check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args, scale_coords,
strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync



from flask import Flask, render_template, Response
import cv2
app=Flask(__name__)
default_vid_path = "http://192.168.1.11:8080/video"
default_model_path = "runs/train/exp/weights/best.pt"



@torch.no_grad()
def run(
    weights=ROOT / default_model_path,  # model.pt path(s)
    source=ROOT / "http://192.168.55.107:8080/video",  # file/dir/URL/glob, 0 for
webcam
    data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
    imgsz=(640, 640),  # inference size (height, width)
    conf_thres=0.25,  # confidence threshold
    iou_thres=0.45,  # NMS IOU threshold
    max_det=1000,  # maximum detections per image
    device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
    view_img=True,  # show results // CHANGE TO TRUE
```

```
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
):
    print(source, "DEBUG SOURCE")
    print("RUNNING")
    source = default_vid_path
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)

    print(webcam, "WEBCAM CHECK!")
    webcam = True

    if is_url and is_file:
```

```
    source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make
dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
        bs = len(dataset)  # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
        bs = 1  # batch_size
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt else bs, 3, *imgsz))  # warmup
    dt, seen = [0.0, 0.0, 0.0], 0
    for path, im, im0s, vid_cap, s in dataset:
        t1 = time_sync()
        im = torch.from_numpy(im).to(device)
```

```
        im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        t2 = time_sync()
        dt[0] += t2 - t1


        # Inference
        visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else
False
        pred = model(im, augment=augment, visualize=visualize)
        t3 = time_sync()
        dt[1] += t3 - t2


        # NMS
        pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms,
max_det=max_det)
        dt[2] += time_sync() - t3


        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)


        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)
```

```
        p = Path(p)  # to Path
        save_path = str(save_dir / p.name)  # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else
f'_{frame}')  # im.txt
        s += '%gx%g ' % im.shape[2:]  # print string
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
        imc = im0.copy() if save_crop else im0  # for save_crop
        annotator = Annotator(im0, line_width=line_thickness, example=str(names))
        if len(det):
            # Rescale boxes from img_size to im0 size
            det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()

            # Print results
            for c in det[:, -1].unique():
                n = (det[:, -1] == c).sum()  # detections per class
                s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  #
normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                    with open(txt_path + '.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

                if save_img or save_crop or view_img:  # Add bbox to image
                    c = int(cls)  # integer class
                    label = None if hide_labels else (names[c] if hide_conf else f'{names[c]}
{conf:.2f}')
```

```
                annotator.box_label(xyxy, label, color=colors(c, True))
                if save_crop:
                    save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] /
f'{p.stem}.jpg', BGR=True)

        # Stream results
        im0 = annotator.result()
        if view_img:
            # cv2.imshow(str(p), im0)
            ret, buffer = cv2.imencode('.jpg', im0)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
            # cv2.waitKey(1)  # 1 millisecond

        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
            else:  # 'video' or 'stream'
                if vid_path[i] != save_path:  # new video
                    vid_path[i] = save_path
                    if isinstance(vid_writer[i], cv2.VideoWriter):
                        vid_writer[i].release()  # release previous video writer
                    if vid_cap:  # video
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                    else:  # stream
                        fps, w, h = 30, im0.shape[1], im0.shape[0]
```

```
                    save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix
on results videos
                    vid_writer[i] = cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                vid_writer[i].write(im0)


        # Print time (inference-only)
        LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')


    # Print results
    t = tuple(x / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image
at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if
save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights)  # update model (to fix SourceChangeWarning)



def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / default_vid_path,
help='model path(s)')
    parser.add_argument('--source', type=str, default=ROOT / default_vid_path,
help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml',
help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640],
help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence
threshold')
```

```
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
```

```
    print_args(vars(opt))
    return opt


def gen_frames():
    opt = parse_opt()
    print("Printing OPT", "*"*100)
    print(opt)
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))


@app.route('/')
def index():
    return render_template('index.html')
@app.route('/video_feed')
def video_feed():
    return Response(run(), mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == "__main__":
    app.run(host="0.0.0.0",debug=True)
```

Pantomime Main

```
from flask import Flask, render_template, Response
import cv2
app=Flask(__name__)
camera = cv2.VideoCapture("http://192.168.55.107:8080/video")
```

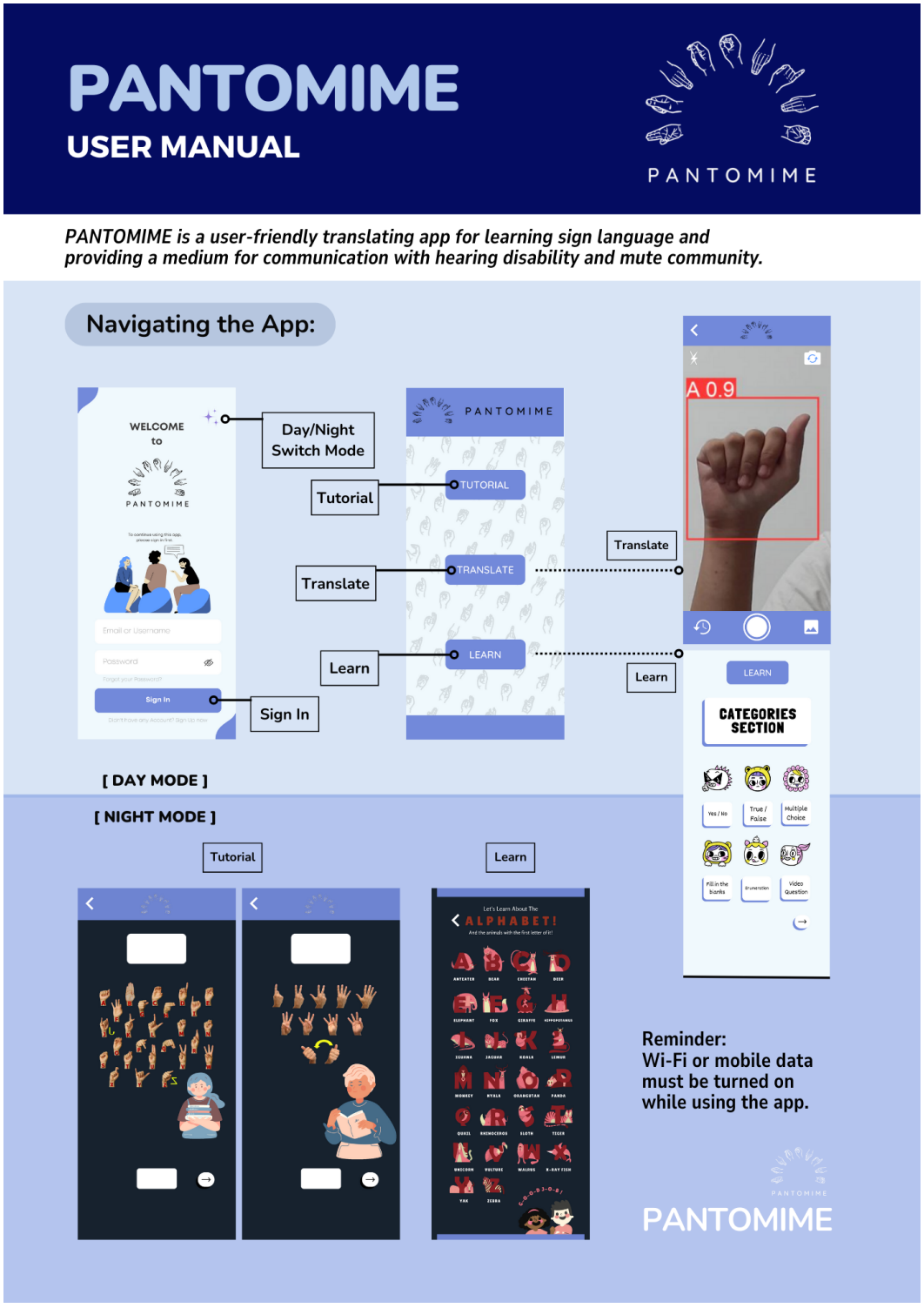
```
def gen_frames():
    while True:
        success, frame = camera.read()  # read the camera frame
        if not success:
            break
        else:

detector=cv2.CascadeClassifier('Haarcascades/haarcascade_frontalface_default.xml')
            eye_cascade = cv2.CascadeClassifier('Haarcascades/haarcascade_eye.xml')
            faces=detector.detectMultiScale(frame,1.1,7)
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
             #Draw the rectangle around each face
            for (x, y, w, h) in faces:
                cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
                roi_gray = gray[y:y+h, x:x+w]
                roi_color = frame[y:y+h, x:x+w]
                eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 3)
                for (ex, ey, ew, eh) in eyes:
                    cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/')
def index():
    return render_template('index.html')
@app.route('/video_feed')
```

```
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
if __name__=='__main__':
    app.run(debug=True)
```

APPENDIX B

USER MANUAL

Appendix B. User Manual



PANTOMIME

USER MANUAL

PANTOMIME is a user-friendly translating app for learning sign language and providing a medium for communication with hearing disability and mute community.

System Requirements:

MINIMUM:

Lowest required specifications
It may run the application at low to mid settings

- CPU: Mediatek MT6735, 4-core 2.1Ghz equivalents
- Storage: 32GB ROM
- RAM: 3GB / 4GB
- OS: Android 8.1 Oreo and above
- Screen Resolution: 800x1280
- Camera:
  - Back: 8 Megapixels and above
  - Front: 3 Megapixels and above

PANTOMIME

RECOMMENDED:

Optimal required specifications
It can run the application at mid to high settings

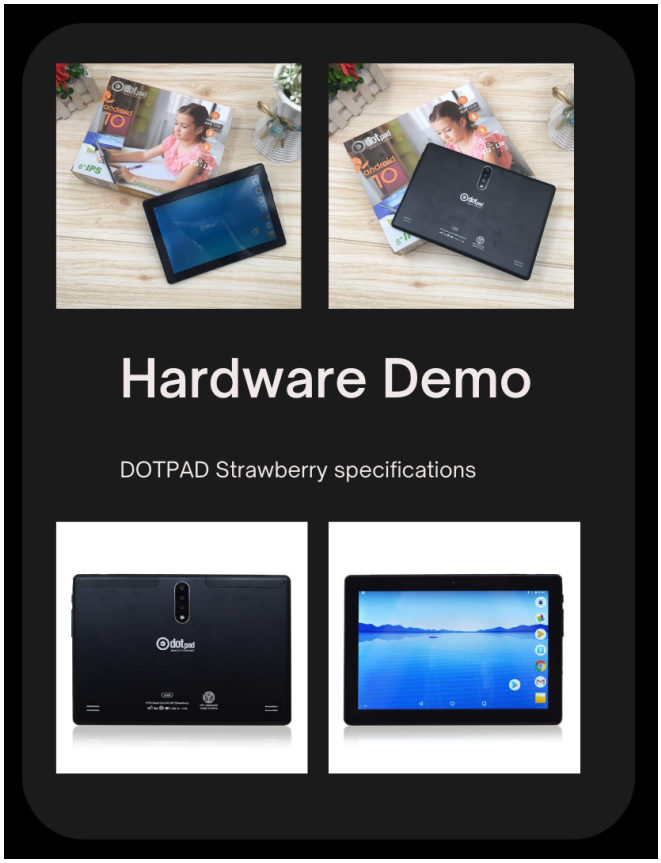
- CPU: Mediatek Helio G70, Qualcomm Snapdragon 730, Exynos 880, 8-core 3.2Ghz equivalents
- Storage: 64GB ROM and above
- RAM: 6GB and above
- OS: Android 9.1 Pie and above
- Screen Resolution: 800x1280
- Camera:
  - Back: 48 Megapixels and above
  - Front: 12 Megapixels and above

PANTOMIME

APPENDIX C

TABLET SPECIFICATIONS

Appendix C. Tablet Specifications



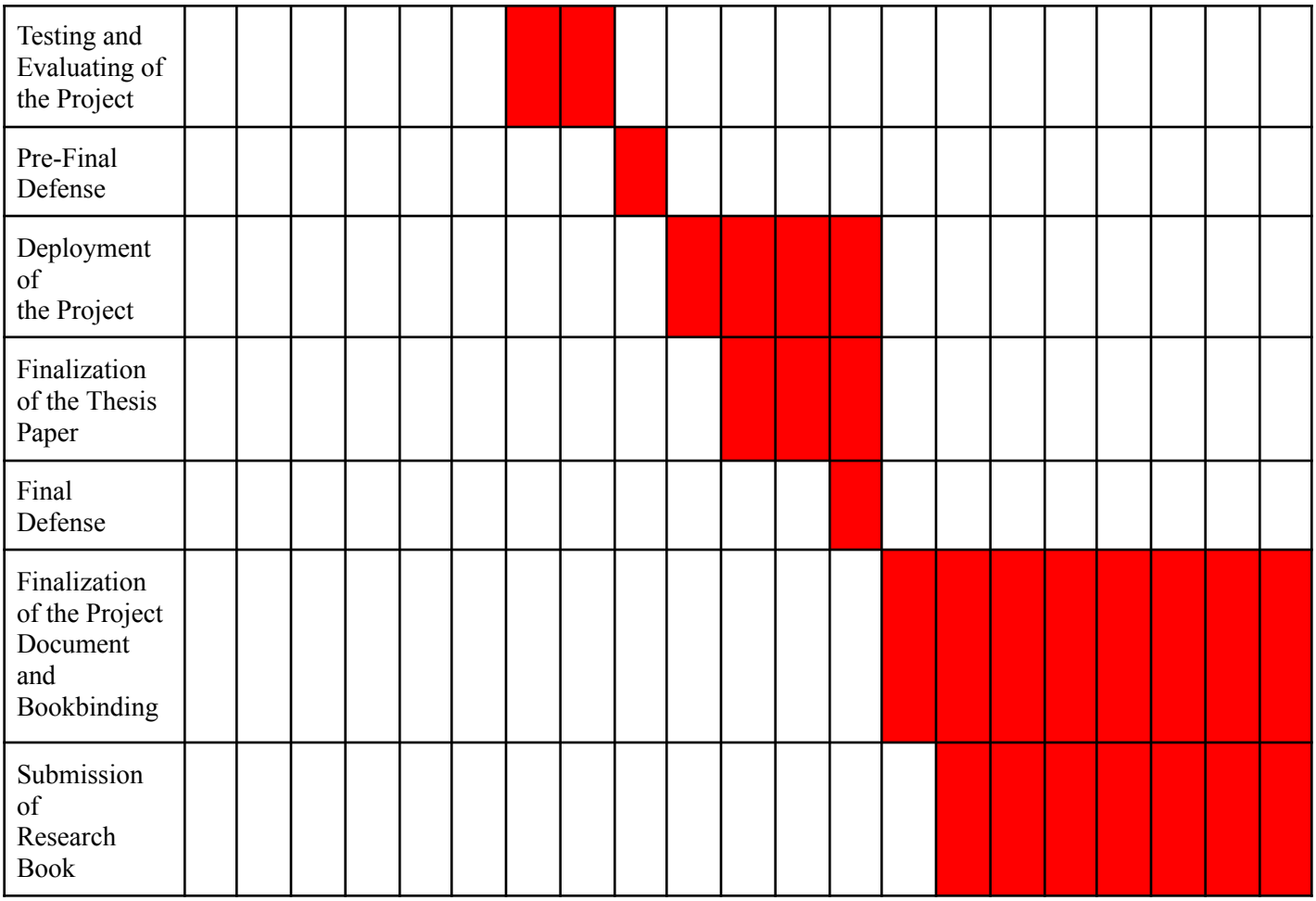
- OS: Android 10
- CPU: Mediatek MT6735
- Storage: 32GB ROM
- RAM: 3GB
- Screen Resolution: 800x1280
- Camera: 12 Megapixels(Back) / 8 Megapixels(Front) and above

APPENDIX D

GANTT CHART

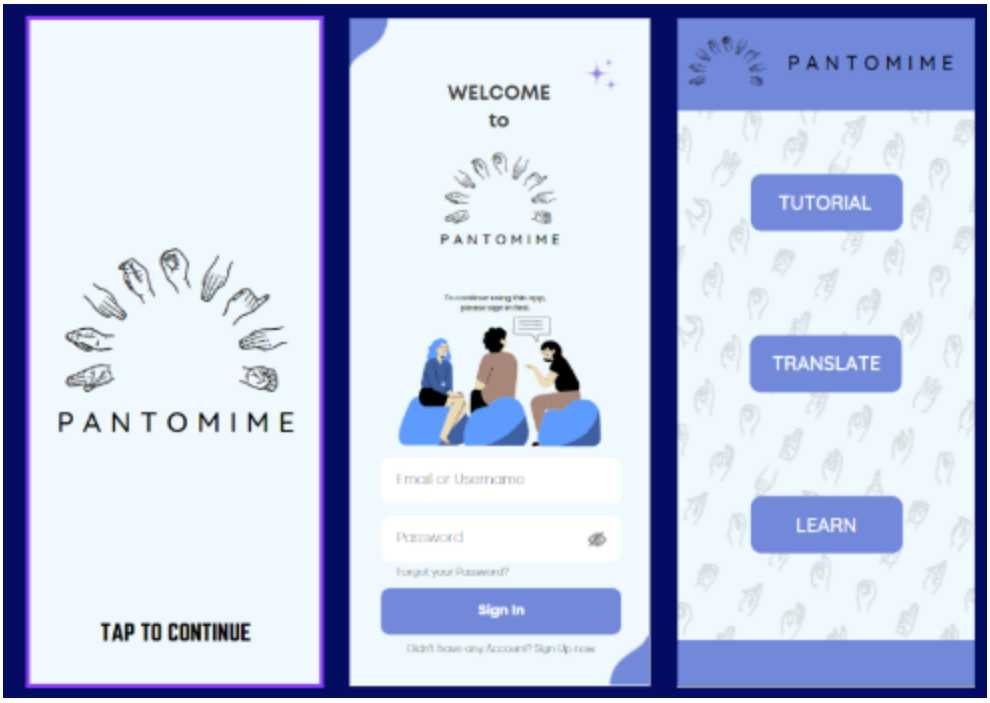
Appendix D. Gantt Chart

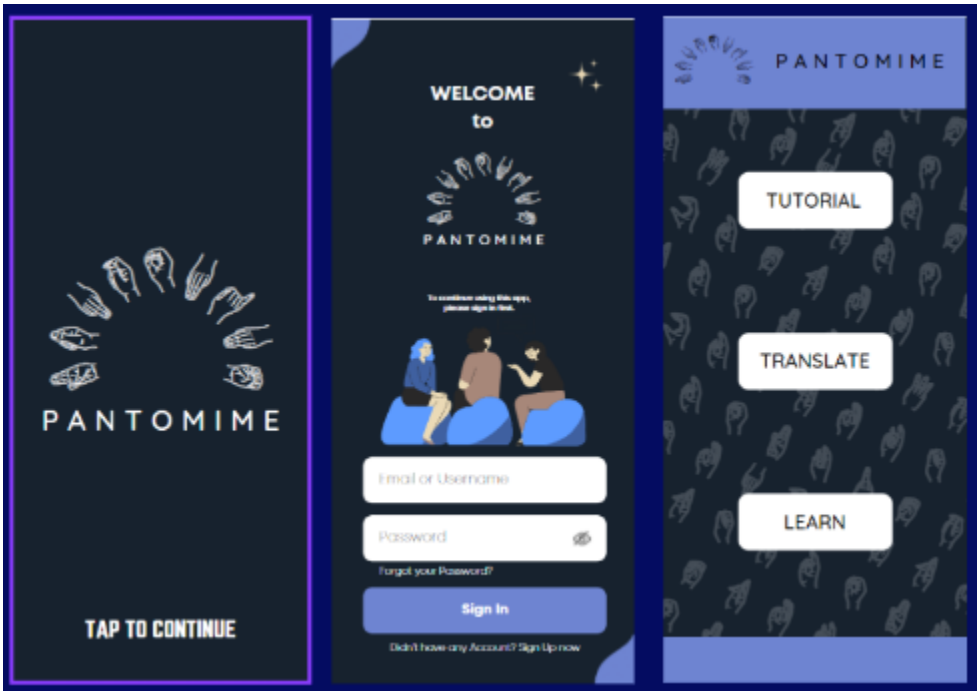
	2021									2022											
Activities Description	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sept	Oct	Nov	Dec
Preparation of Chapter 1 to 3																					
Topic Defense																					
Consultation with Topic Adviser																					
Title Defense																					
Purchasing of Project Components																					
Further Research for the Project development																					
Utilization of Machine learning algorithms																					
Documentation of the Progress Defense																					
Progress Defense																					
Development of Mobile Application																					
Project Integration																					


Testing and Evaluating of the Project
Pre-Final Defense
Deployment of the Project
Finalization of the Thesis Paper
Final Defense
Finalization of the Project Document and Bookbinding
Submission of Research Book

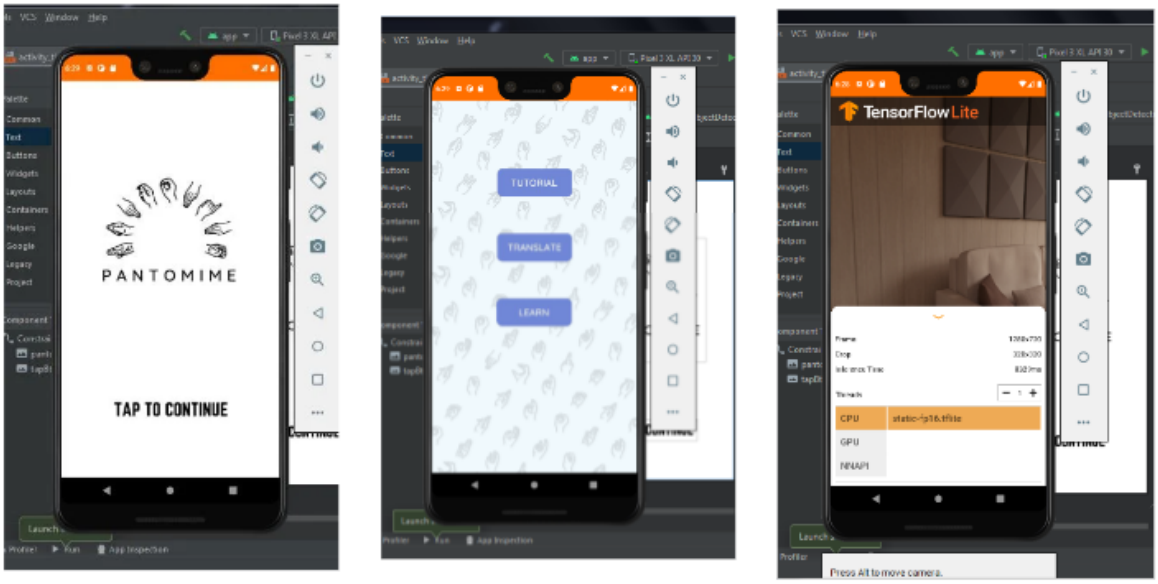
APPENDIX E

PROJECT DOCUMENTATION

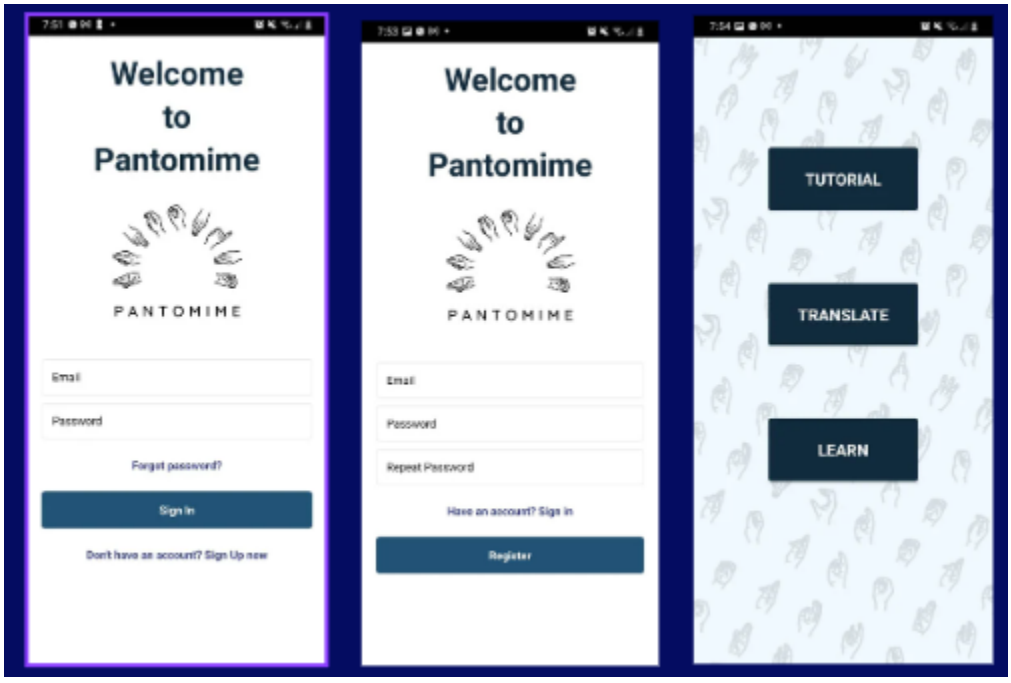




CONCEPT DESIGN OF THE APPLICATION



INITIAL USER INTERFACE DESIGN



FINAL USER INTERFACE DESIGN

TITLE DEFENSE

JUNE 2, 2021

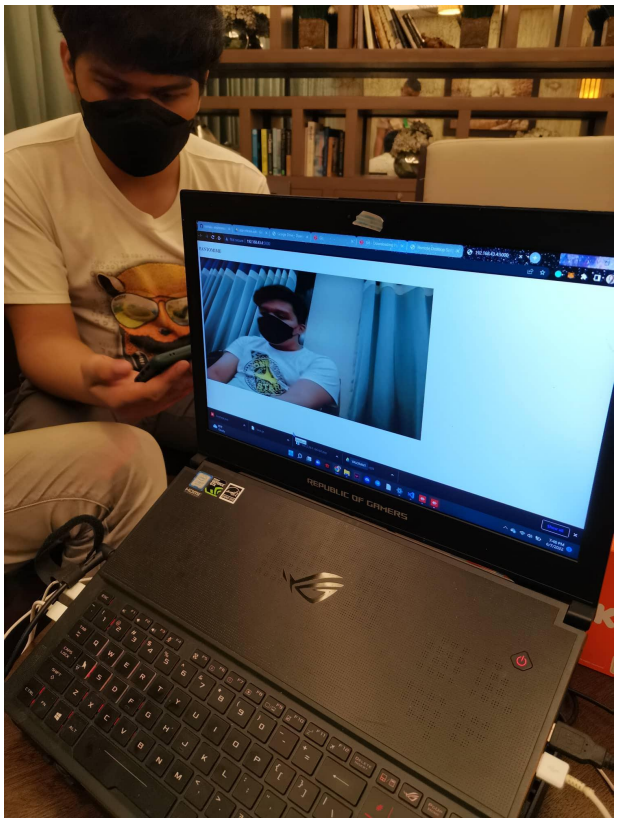
PROGRESS PRESENTATION

NOVEMBER 5, 2021

PRE-FINAL DEFENSE

FEBRUARY 2, 2022

DEPLOYMENT


REPUBLIC OF GAMERS

APPENDIX F

ABOUT THE AUTHORS

Appendix F. About the Authors