Two Way Sign Language Translation(Speakd)

Graduation Project

Faculty of Artificial Intelligence, Kafrelsheikh University

By

Alaa Hussien Fathy Hasaneen

Hagar Ashraf Mohamed El-Said

Mariam Mohamed Mahmoud Gabr

Nada Nasser Badawi Ahmed

Hosam Mohamed Ali Mohamed

Atef Yasser Wassif Fathi

SUPERVISED BY

Assoc. Prof. Mahmoud Y.Shams

Dept. of Machine Learning and Information Retrieval

Faculty of Artificial Intelligence, Kafrelsheikh University

Kafrelshiekh-2024

Acknowledgement

We are incredibly grateful for the opportunity to have gained valuable experience through this real-world project. This project not only equipped us with the knowledge to design and analyze, but also fueled our passion for the field.

First and foremost, we would like to express our sincere appreciation to His Excellency, **President Abdel Fattah el-Sisi**, for his unwavering commitment to fostering technological advancement and digitization in Egypt. This vision has paved the way for countless opportunities, including the establishment of our esteemed **Faculty of Artificial Intelligence** at Kafr El-Sheikh University.

We are deeply indebted to our faculty for its unwavering support. We extend our heartfelt gratitude to:

Prof. Abdelrazek Desouki, President of Kafrelsheikh University and Supervisor of the Faculty of Artificial Intelligence, for his instrumental role in establishing our faculty. **Dr. Mahmoud Y. Shams**, our graduation project supervisor, for her patience, guidance, and invaluable insights throughout the year. **Eng. Abdelmawla Yousef**, our graduation project coordinator, for her continuous encouragement, support, and expert guidance. **Prof. Tamer Medhat**, Vice Dean of the Faculty of Artificial Intelligence, for his dedication to the faculty's excellence.

We are immensely grateful to our college for providing us with the resources and knowledge necessary to excel. The faculty's commitment to equipping us with the skills to succeed in the job market after graduation is truly commendable.

Finally, we extend our sincere thanks to everyone who has supported and encouraged us on this journey, especially those who played a role in the successful completion of our graduation project.

Abstract

People with speech impairments or hearing loss frequently utilize sign language as a way of communication. However, sign language is not universally understood, creating a significant communication barrier between the deaf community and the general public. This paper proposes a mobile application designed to facilitate two-way sign language translation, enabling seamless communication between deaf individuals and those who do not understand sign language.

To achieve this, we leverage advanced deep learning techniques, specifically YOLO v8 and YOLO v5 models, for accurate sign language recognition and translation. YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system known for its speed and precision. By utilizing these models, our system is capable of recognizing and translating sign language gestures with high accuracy.

Our experimental results demonstrate that the YOLO v8 model achieves high precision, recall, and mean average precision (MAP) scores of 98%, 97%, and 99%, respectively, on the Sign Hands dataset. On the American Sign Language (ASL) dataset, YOLO v8 achieves precision, recall, and MAP scores of 95%, 96%, and 98%, respectively. Similarly, the YOLO v5 model shows competitive performance with precision, recall, and MAP scores of 96.5%, 96.7%, and 98%, respectively, on the Sign Hands dataset, and 94%, 95%, and 98%, respectively, on the ASL dataset.

The mobile application provides two key functionalities:

-Text/Speech to ASL: Converts spoken or written English into text and translates it into ASL animations using a 3D avatar.

-ASL to Text/Speech: Recognizes ASL gestures captured through the camera and translates them into text or spoken English.

This mobile application has the potential to significantly improve accessibility for both deaf and hearing communities by bridging the communication gap with high-precision sign language recognition and translation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

With the great progress of technology in our time, it has become necessary to introduce technology in all areas of our lives from agriculture. Industry, commerce, and tourism.

The impact of technology on our daily life is much greater than we realize. It is developing and growing quickly. The way we access resources has altered as a result. It has also altered how we pick up new information. People these days frequently depend on technology for everything. We can instantaneously text someone whenever we need to get in touch with them. It used to move considerably more slowly because of letters and meetings. This is how technology has altered how we speak to one another. We are eventually testing the limits of technology and the ways in which it affects us as our needs and technical expectations continue to increase.

At this moment, it is challenging to envision a world without technology. The quality of life has changed significantly as a result. The way we behave, and function has altered because of technology permeating every part of our lives. Technology has improved every aspect of our life, from networking and healthcare to communication and transportation. The finest thing is that it always improves by enabling more sophisticated functions. For instance, facetime and instant messaging have come a long way from conventional voice calls. Not to mention that technology has helped during

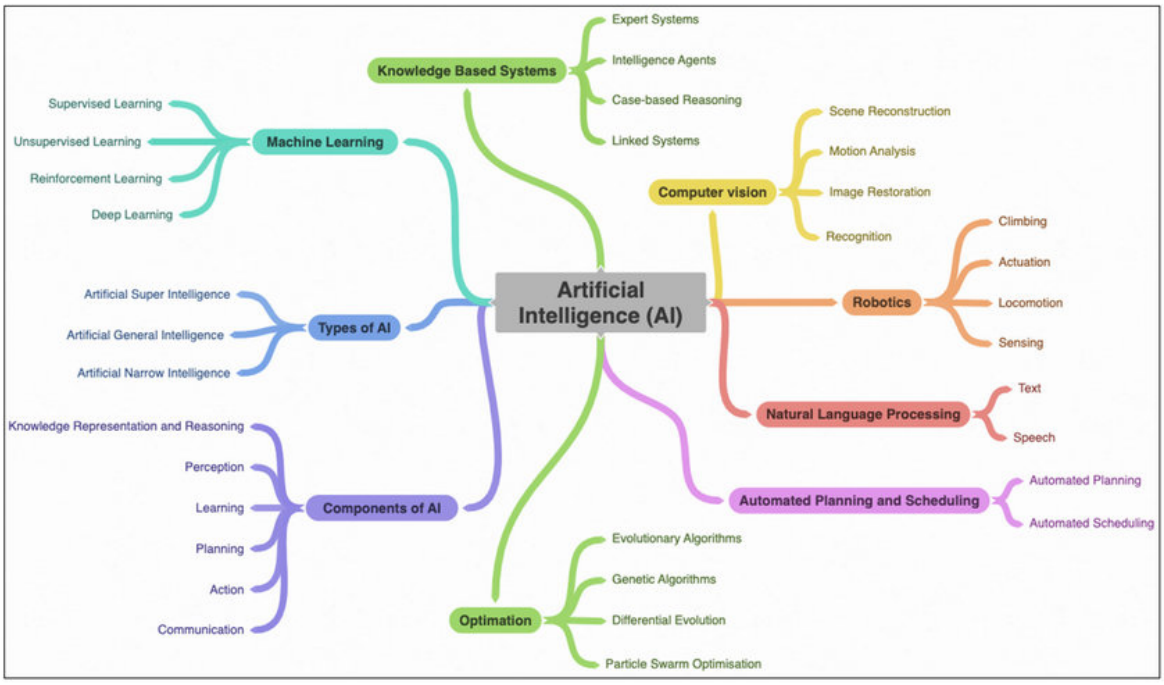


Figure 1.1: Artificial Intelligence branches

the pandemic. Video calling and posting messages on the internet have paved the road for people to contact with one another during times of global lockdown.

And through the new technologies in the field of artificial intelligence, especially the field of natural language processing, as well as computer vision and machine learning, their effects at the present time and in the future will affect many different fields. The field of computer science known as "natural language processing" (NLP) is more particularly the field of "artificial intelligence" (AI) that is concerned with providing computers the capacity to comprehend written and spoken words in a manner like that of humans. **NLP** blends statistical, machine learning, and deep learning models with computational linguistics—rule-based modelling of human language. With the use of these technologies, computers are now able to process human language in the form of text or audio data and fully "understand" what is being said or written, including the speaker's or writer's intentions and Sentiment. Computer programs that translate text between languages, reply to spoken requests, and quickly summarize vast amounts of text—even in real time— are all powered by NLP. You have probably used NLP in

the form of voice- activated GPS devices, digital assistants, speech-to-text dictation programs, customer service chatbots, and other consumer conveniences. The use of NLP in corporate solutions, however, is expanding as a means of streamlining business operations, boosting worker productivity, and streamlining mission-critical business procedures.

Computer vision is a branch of artificial intelligence (AI) that enables computers and systems to extract useful information from digital photos, videos, and other visual inputs and to execute actions or make recommendations based on that information. AI makes it possible for computers to think, while computer vision makes it possible for them to see, hear, and comprehend.

Humans have an advantage over computers when it comes to how eyesight works. The benefit of human sight is that it has had a lifetime to learn how to distinguish between things, determine their distance from the viewer, whether they are moving, and whether something is off about an image. With cameras, data, and algorithms instead of retinas, optic nerves, and a visual cortex, computer vision teaches computers to execute similar tasks in a much less time. A system trained to inspect products or monitor a production asset can swiftly outperform humans since it can analyze thousands of products or processes per minute while spotting imperceptible flaws or problems.

It was necessary to use such fields to benefit from them in the fullest way, and within the integration between the different fields and activities, the choice fell on the field of the helping of the Deaf and dumb. Therefore, attention is paid to the deaf and dumb as we find difficulty in dealing with them and understanding their needs or translating their sign language. So, we decided to make a simple app project to convert our speech to sign language and vice versa to ease our communication with them.

1.2 Purpose of the project

With 350 million in the world,10 million in the Middle East, and 7.5 million in Egypt have not only lost their voice and the ability to talk, but they have also been isolated

from the world with an ignored language. So, we decided to make a simple app project to convert our speech to sign language and vice versa to ease our communication with them. The user will use this app to say something and the service will translate it to sign language by 3D animation on the other side someone else will open the application camera to capture sign language in real-time and the system will convert it to text and voice to understand their need and ease our communication between us and we haven't to learn sign language. So, if we can do this app, we will help them and help us to understand each other and avoid embarrassment. 70 million people around the world use sign language and there's a lack of communication between the deaf and dump community and our community. The deaf and dump community moves back When it comes to the interactive part with people with special needs, this communication gap has been exacerbated because our community has no idea about sign language.

1.3 Objectives

The main objective of this project is to design and develop a technological solution that enhances our communication with the deaf and mute. The sign language translator will achieve this by easy-to-use, the project aims to:

Easy-to-use Our Sign language translator is very easy to use as it can be used by The user to say something and the service will translate it to sign language by 3D animation on the other side someone else will open the application camera to capture sign language in real-time and the system will convert it to text and voice to understand their need.

1.4 Scope

The project will focus on the design and development of a Sign Language Translator Application for the deaf and mute. The application will be equipped with advanced software technologies that enable it to translate languages to sign language by 3D animation and provide instant translations. it also enables it to recognize sign language and convert it to text and voice to provide instant translations.

1.5 Methodology

To achieve the objectives of the project, we will use YOLOv5 for Computer Vision, Deep Learning, and 3D Animation to represent sign language. The application camera will allow us to implement recognition technology and provide accurate translation.

We will also conduct extensive research on Sign Language and culture to ensure that the translations provided by the application are accurate and reliable. Additionally, we will conduct user testing to ensure that the application meets the needs of communication between the deaf and dump community and our community.

1.6 Deliverable

The deliverables of the project include: -application to translate voice to sign language by 3D animation

-the application camera captures sign language in real-time and the system will convert it to text and voice to understand the needs of the deaf and mute and ease our communication between us if we haven't learned sign language.

1.7 Conclusion

The Sign Language Translator project aims to enhance communication between the deaf and dump community and our community . The project will use advanced software technologies to achieve these features, and the 3D Animation, Deep Learning and YOLO will serve as the platform for the project

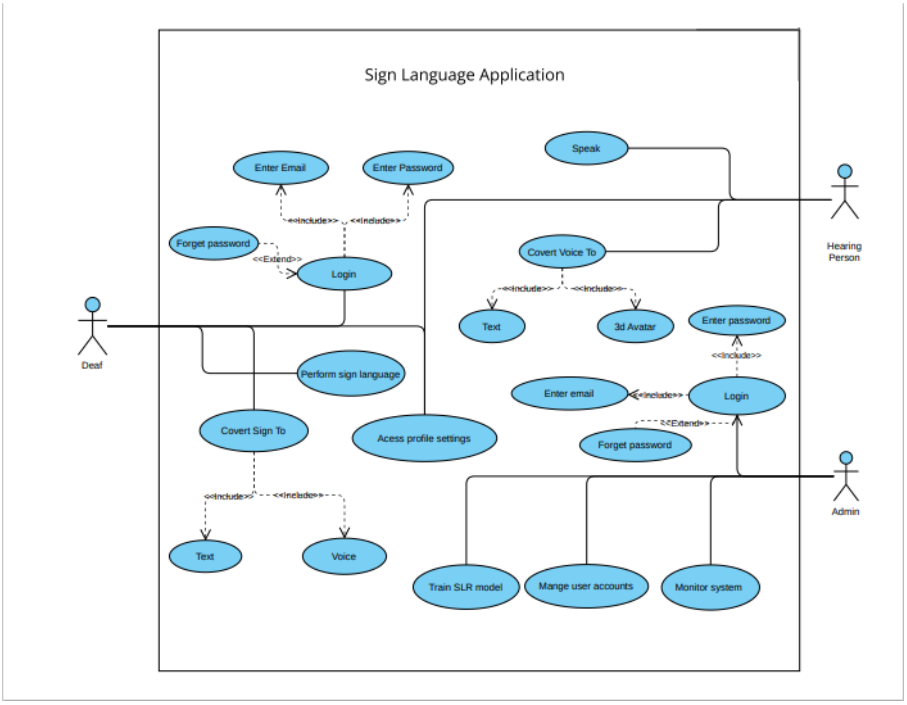
Chapter 2

System Methdology

2.1 Introduction

We all have at least one person with special needs (disabled people). This project concerns the deaf and dumb as we find difficulty in dealing with them and understanding their needs or translating their sign language. So, we decided to make a simple app project to convert our speech to sign language and vice versa to ease our communication with them. The user will use this app to say something and the service will translate it to sign language by 3D animation on the other side someone else will open the application camera to capture sign language in real-time and the system will convert it to text and voice to understand their need and ease our communication between us and we haven't to learn sign language. So, if we can do this app, we will help them and help us to understand each other and avoid embarrassment. 70 million people around the world use sign language and there's a lack of communication between the deaf and dump community and our community. The deaf and dump community moves back When it comes to the interactive part with people with special needs, this communication gap has been exacerbated because our community has no idea about sign language.

Use case that demonstrates the application



Objective: Enable seamless communication between deaf and hearing individuals through a mobile application.

Key Features:

1-Sign Language to Voice and Text:

-Deaf Actor: Uses sign language.
-App Function: Captures sign language via camera, converts it to text and synthesized voice.
-Output: Hearing actor receives the message in text and/or voice.

2-Voice to 3D Avatar and Text:

-Hearing Actor: Speaks into the app.
-App Function: Converts speech to text and animates a 3D avatar to perform sign language.
-Output: Deaf actor sees the 3D avatar's gestures and/or reads the text.

3-System Admin:

-Trains the machine learning model with new sign language data.
-Monitors app performance and resolves technical issues.
-Ensures data security and compliance.

2.2 Overview of sign language

Sign language, any means of communication through bodily movements, especially of the hands and arms, used when spoken communication is impossible or not desirable. The practice is probably older than speech. Sign language may be as coarsely expressed as mere grimaces, shrugs, or pointings; or it may employ a delicately nuanced combination of coded manual signals reinforced by facial expression and perhaps augmented by words spelled out in a manual alphabet. Wherever vocal communication is impossible, as between speakers of mutually unintelligible languages or when one or more would-be communicators is deaf, sign language can be used to bridge the gap.

2.2.1 Inability to speak

The Indian sign language was codified by use into an explicit vocabulary of gestures representing or depicting objects, actions, and ideas, but it made no attempt to "spell out" or otherwise represent words that could not be conveyed by gestures. Several forms of sign language were developed to enable signers to spell out words and sounds, however. Most of these are as complex and flexible as spoken languages. It was long thought in many cultures that the deaf were ineducable, and the few teachers willing to try were available only to the wealthy. In the mid-18th century, however, the first educator of poor deaf children, Charles-Michel, abbé de l'Epée, developed a system for spelling out French words with a manual alphabet and expressing whole concepts with simple signs. From l'Epée's system developed French Sign Language (FSL), still in use in France today and the precursor of American Sign Language (ASL) and many other national sign languages. FSL was brought to the United States in 1816 by Thomas Gallaudet, founder of the American School for the Deaf in Hartford, Connecticut. The new sign language was combined with the various systems already in use in the United States to form ASL, which today is used by more than 500,000 deaf people in the United States and Canada; it is the fourth most common language in the United States. National sign languages such as ASL have more in common with one another than with the spoken languages of their country of origin, since their signs represent concepts and

not English or French or Japanese words. One system, Cued Speech, first developed by the American physicist R. Orin Cornett in 1966, does, however, successfully employ hand signs representing only sounds (not concepts), used in conjunction with lipreading. It has been adapted to more than 40 languages.

2.3 Dataset

The dataset used as input for sign language recognition system is a dataset con- tains American Sign Language characters only sign hands[9] and other data contains characters and words American-sign-language (ASL)

2.3.1 Sign hands alphabet:

Contains 26 class from A to Z and data is spilt into train and valid and test. Data specification is given in Table 1

Table 2.1: Sign hands data description

specification	value
Numbers of images	1815
Train data	1271(70%)
Valid data	363(20%)
Test data	181(10%)
Number of classes	26
Number of images per class	70

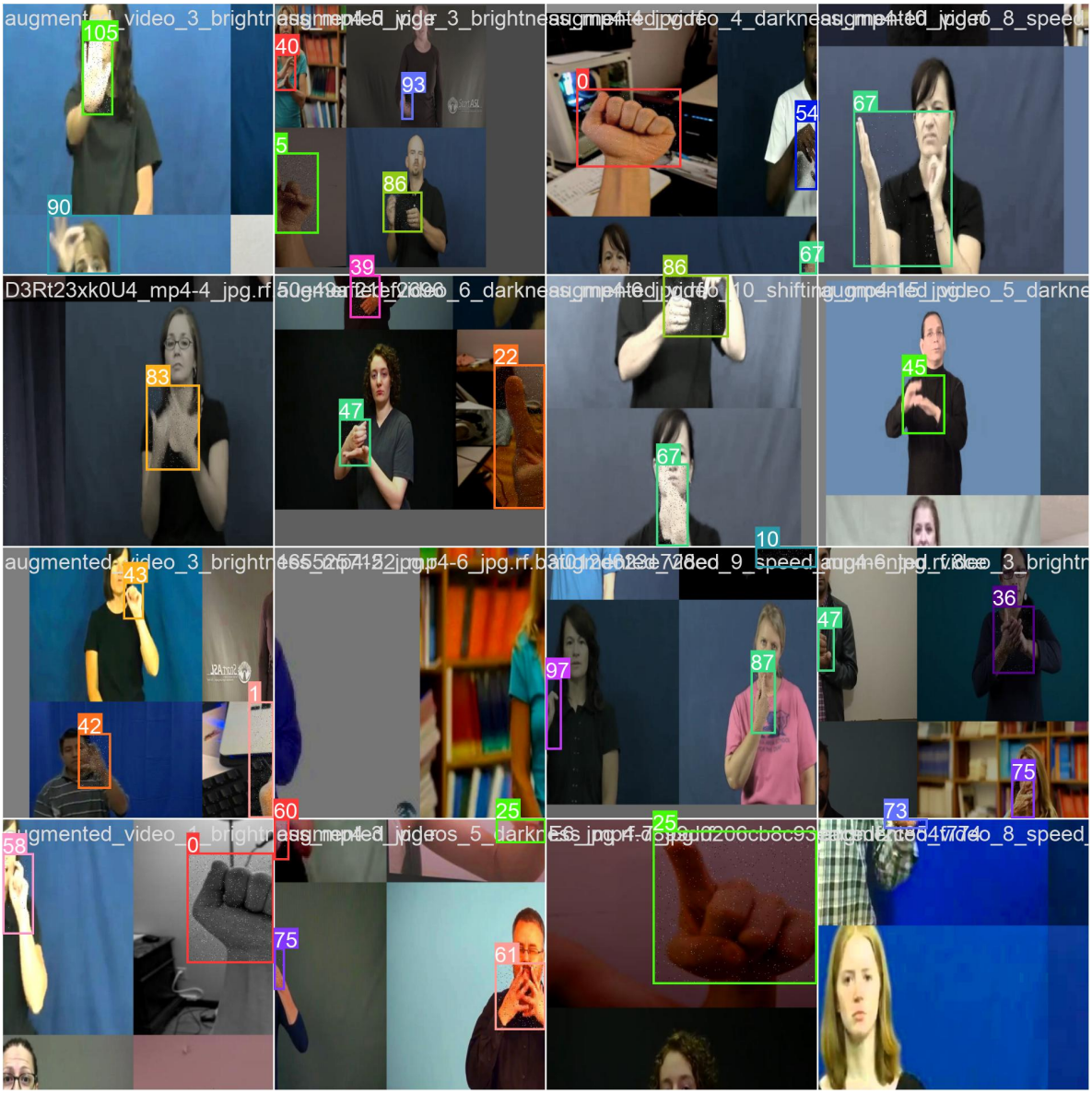
Sign hands dataset

2.3.2 American-sign-language (ASL):

This dataset is videos and these videos was spilt into frames and every frame was annotated and contains 106 class (26 classes are characters, 80 classes are words) and this data is spilt into train and valid and test. Data specification is given in Table 2

Table 2.2: ASL data description

specification	value
Numbers of images	23343
Train data	20630(88%)
Valid data	1768(8%)
Test data	945(4%)
Number of classes	106
class	A-B-C-D-E-F-G-H-J-K-L-M-N-O-P-Q-S-T-U-V-X-Y-Z-allergy-bag-barbecue-bill-straw-bitter-additional-bread-coupon-bye-cake–cheese-chicken-coke-cold-cost-credit card-cup-enjoy-dessert-drive-eat-ketchup-egg-fork-French-fries-fresh-hello-hot-bacon-ice cream-ingredients-juicy-lactose-lettuce-mustard-lid-manager-pepper-menu-alcohol-milk-napkin-no-order-cash-pickle-burger-please-ready-refill-repeat-safe-drink-salt–sandwich-sauce-biscuit-small-soda-tissues-sorry-spicy-spoon-sugar-sweet-thank you–tomato-vegetables-total-urgent-receipt-wait-what-water-yoghurt-warm-would–your-pizza



ASL dataset

2.4 Data Preprocessing

2.4.1 Image Annotation

Image annotation is the process of labeling images in a given dataset to train machine learning models. When the manual annotation is completed, labeled images are processed by a machine learning or deep learning model to replicate the annotations

without human supervision. Image annotation sets the standards, which the model tries to copy, so any error in the labels is replicated too. Therefore, precise image annotation lays the foundation for neural networks to be trained, making annotation one of the most important tasks in computer vision. The process of a model labeling images on its own is often referred to as model-assisted labeling. Image annotations can be performed both manually and by using an automated annotation tool. Auto annotation tools are generally pre-trained algorithms that can annotate images with a certain degree of accuracy. Their annotations are essential for complicated annotation tasks like creating segment masks, which are time-consuming to create. In these cases, auto-annotate tools assist manual annotation by providing a starting point from which further annotation can proceed. Manual annotation is also generally assisted by tools that help record key points for easy data labeling and storage of data. Image annotation creates the training data that supervised AI models can learn from. The way we annotate images indicates the way the AI will perform after seeing and learning from them. As a result, poor annotation is often reflected in training and results in models providing poor predictions. Annotated data is specifically needed if we are solving a unique problem and AI is used in a relatively new domain. For common tasks like image classification and segmentation, there are pre-trained models often available and these can be adapted to specific use cases with the help of Transfer Learning with minimal data. Training a complete model from scratch, however, often requires a huge amount of annotated data split into the train, validation, and test sets, which is difficult and time-consuming to create. Unsupervised algorithms on the other hand do not require annotated data and can be trained directly on the raw collected data.

2.4.2 Image Data Augmentation

Image data augmentation is the process of generating new transformed versions of images from the given image dataset to increase its diversity. To a computer, images are just a 2-dimensional array of numbers. These numbers represent pixel values, which you can tweak in many ways to generate new, augmented images. These augmented images

Figure 2.1: Class Bag Augmentation

resemble those already present in the original dataset but contain further information for better generalization of the machine learning algorithm. Image data augmentation is beneficial for improving the performance of object detection, classification, or segmentation models. For deployable computer vision solutions, more extensive datasets are preferable; datasets that cover all visual aspects of a target object. But this is easier said than done. Image data collection requires manual capturing and annotations of images, and it is impossible to capture every single scenario that may be useful for the computer vision model.

Say you want to collect images of scenic landscapes for a CV project. It is not humanly possible to capture pictures in all lighting conditions. No matter how hard you try, your dataset will always have some information missing, which limits the ability of your CV model to learn, and the final output might not be as expected. Data augmentation techniques can help create new images to fill in this missing data. Image Augmentation Techniques for Computer Vision Image data is perhaps one of the easiest to augment due to the wide-ranging availability of relevant techniques. This plethora of augmentation techniques performs very well for computer vision tasks. Some of these include:

Position Manipulation: You can change the image position in many different ways.

Scaling: Increase or decrease the picture size. Rotation: Rotate the image to generate new angles. Flipping: Flip the image left, right, or upside down. Color Manipulation:The colors of an image hold vital information for the machine learning models. We can change these colors for different effects. Examples of such tweaks are: -Brightness

-Contrast

-Saturation

-Hue

Changing the above settings can help us create different lighting conditions. These modified images can be helpful.

Image Manipulation: You can manipulate images in multiple ways to synthesize data representing different situations. We can use the following techniques for image augmentation:

Blur: Images can be blurred using different kernel configurations depending on the amount of blur required. Blur allows us to generate pictures with varying levels of focus and degraded image quality.

Sharpening: Sharpness has the opposite effect on an image compared to blur, but it serves the same purpose; to get different levels of focus and higher image clarity.

Random cropping: Randomly crop out bits and pieces from an image, allowing the model to learn from non-perfect data, which is a better fit for real-world situations.

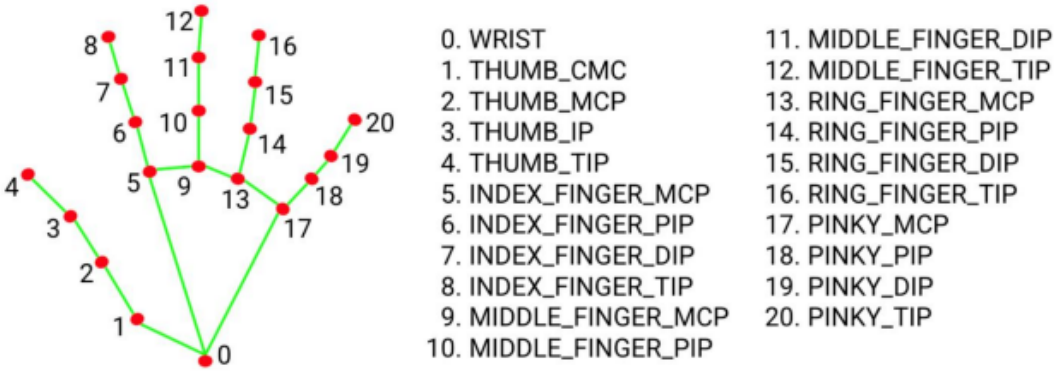
We can use all the methods mentioned above to increase the size of our dataset manifold.

2.5 Sign Language Detection

2.5.1 Hand Detector Module:

is a key component of a hand tracking module that is responsible for detecting the presence of hands in an image or video frame. The hand detector is usually based on computer vision algorithms or machine learning models that are trained to identify the shape, color, texture, and other features of human hands. Once the hand detector has

identified the position and orientation of the hands in the image or video frame, the hand tracking module can use this information to track the movement of the hands over time. This involves estimating the 3D position and orientation of the hands relative to the camera or sensor and predicting the next position of the hands based on their past movement. Hand detectors can vary in their accuracy and robustness, depending on the quality of the training data, the complexity of the algorithms or models used, and the environmental conditions in which the hand tracking module is used. Some hand detectors may be optimized for specific types of hands or hand poses, while others may be more general purpose and able to detect a wide range of hand shapes and movements. Overall, the quality and performance of a hand tracking module depend on the effectiveness of its hand detector, as well as the other components of the module, such as the tracking algorithm, the user interface, and the hardware platform. The MediaPipe Hand Landmarker task lets you detect the landmarks of the hands in an image. You can use this Task to localize key points of the hands and render visual effects over the hands. This task operates on image data with a machine learning (ML) model as static data or a continuous stream and outputs hand landmarks in image coordinates, hand landmarks in world coordinates and handedness (left/right hand) of multiple detected hands.



Mediapipe Hand Tracking Module Landmarks

landmark model bundle detects the key points localization of 21 hand-knuckle coor-

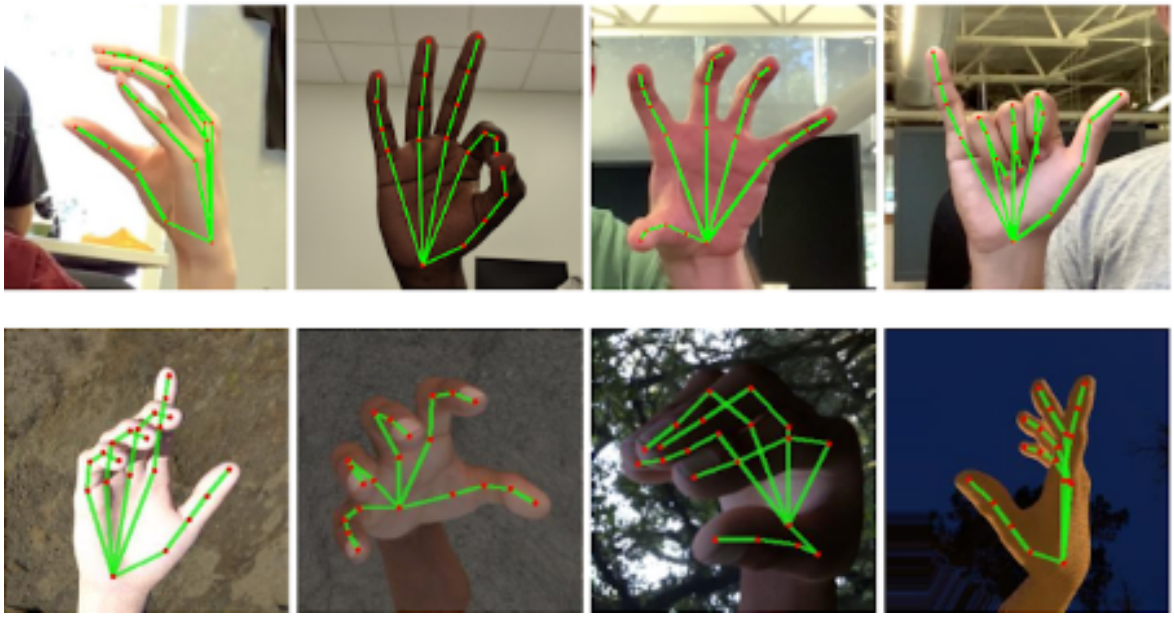


Figure 2.3: sign language detection using mediapipe

dinates within the detected hand regions. The model was trained on approximately 30K real-world images, as well as several rendered synthetic hand models imposed over various backgrounds. The hand landmarker model bundle contains a palm detection model and a hand landmarks detection model. The Palm detection model locates hands within the input image, and the hand landmarks detection model identifies specific hand landmarks on the cropped hand image defined by the palm detection model. Since running the palm detection model is time consuming, when in video or live stream running mode, Hand Landmarker uses the bounding box defined by the hand landmarks model in one frame to localize the region of hands for subsequent frames. Hand Landmarker only re-triggers the palm detection model if the hand landmarks model no longer identifies the presence of hands or fails to track the hands within the frame. This reduces the number of times Hand Landmarker triggers the palm detection model.

2.5.2 Classification Module:

also known as a classifier, is a type of machine learning model that is trained to assign input data to one of several predefined categories or classes. The classifier takes a set

of features as input and outputs a predicted class label based on its learned knowledge from the training data. Classification modules are commonly used in a wide range of applications, such as image recognition, speech recognition, natural language processing, fraud detection, and spam filtering. In sign language, each hand sign refers to a certain character. And the collection of signs can construct a certain word. As flow:

So, after hand detection and tracking, the classifier take the output from hand detector module as input for classifier and classify the sign of the hand to certain character. In this project, we used deep learning classifier based on Keras to classify hand sign characters. The model works on the bound boxes for hands detected in hand detector module, and for bound box it predicted the most match class for it.

2.5.3 Training process

In this research, we trained both the medium versions of YOLO v5 and YOLO v8 for sign language recognition.

Key Training Parameters

- **Image size:** We resized all images in the training dataset to a uniform size of 640x640 pixels to ensure consistency for the model's input.
- **batch size:** we used 16 during training.
- **Number of epochs:** For YOLO v5, we used 150 epochs, while for YOLO v8, we used for 100 epochs.

Our YOLO v5 and YOLO v8 models achieved superior performance compared to approaches utilizing MediaPipe and the SSD-MobileNet-V2 model. Yolo can overcome the challenges of other object detection models as it offers many features:

1-**Single Shot Detection**: Yolo is a single-stage object detection model that predicts bounding boxes and class probabilities for objects by processing the full image at once. Because of its efficiency, it can be used for real-time tasks like hand detection.

2-**Robustness to Size Variations**: YOLO is made to recognize items in an image at various scales. It effectively handles item size fluctuations by using a grid-based method to anticipate bounding boxes and class probabilities throughout the entire image.

3-**End-to-End Training**: Yolo is trained end-to-end, meaning it acquires the ability to simultaneously optimize the tasks of localization and classification. This may help improve robustness and generalization when identifying hands of different sizes and orientations.

2.6 Results

We used two different datasets split into training, testing, and validation. Dataset with characters only achieved in Yolo v5 96.7% recall, 96.5% precision, and 98% mean average precision(MAP) whereas achieved in Yolo v8 97% recall, 98% precision, and 99% mean average precision (Table 2.3).

The second dataset achieved in Yolo v5 95% recall, 94% precision, and 98% mean Average precision whereas achieved in Yolo v8 96% recall, 95% precision, and 98% mean average precision (Table 2.4).

Figure 2. Shows the confusion matrix of v5 and v8 on the second dataset. Yolo v5 has declined faster than Yolo v8 in both classification loss and bounding box (Figure 2.2). Yolo v8 is faster and more accurate than Yolo v5 in detecting sign language

Table 2.3: results of sign hands dataset

model type	precision	recall	MAP	Box loss	Classification loss
YOLO v5	96.5%	96.7%	98%	0.01	0.009
YOLO v8	98%	97%	99%	0.06	0.09

Table 2.4: results of American-sign-language (ASL) dataset

model type	precision	recall	MAP	Box loss	Classification loss
YOLO v5	94%	95%	98%	0.01103	0.00135
YOLO v8	95%	96%	98%	0.02	0.066

2.7 Figures Of Results

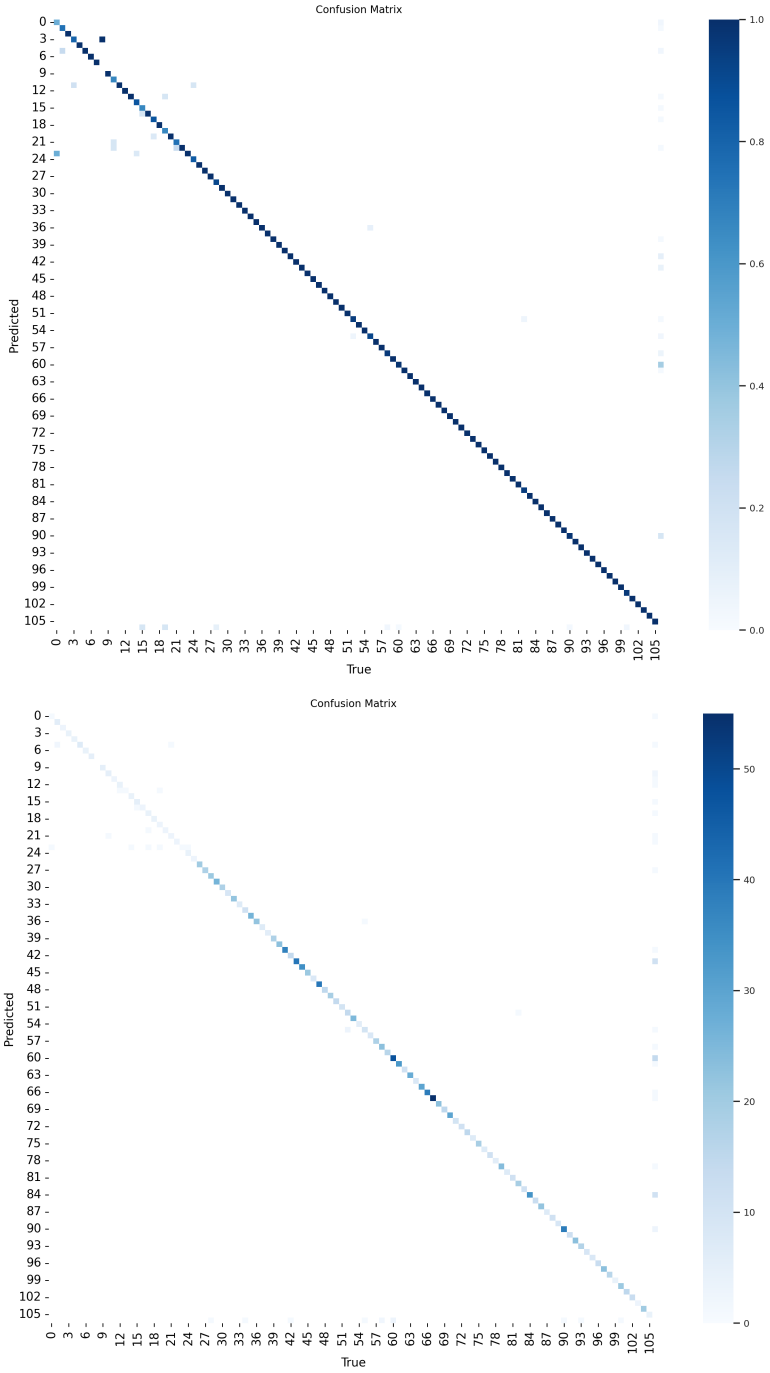


Figure 2: Confusion matrix for YOLOv5(above) and YOLOv8(below)

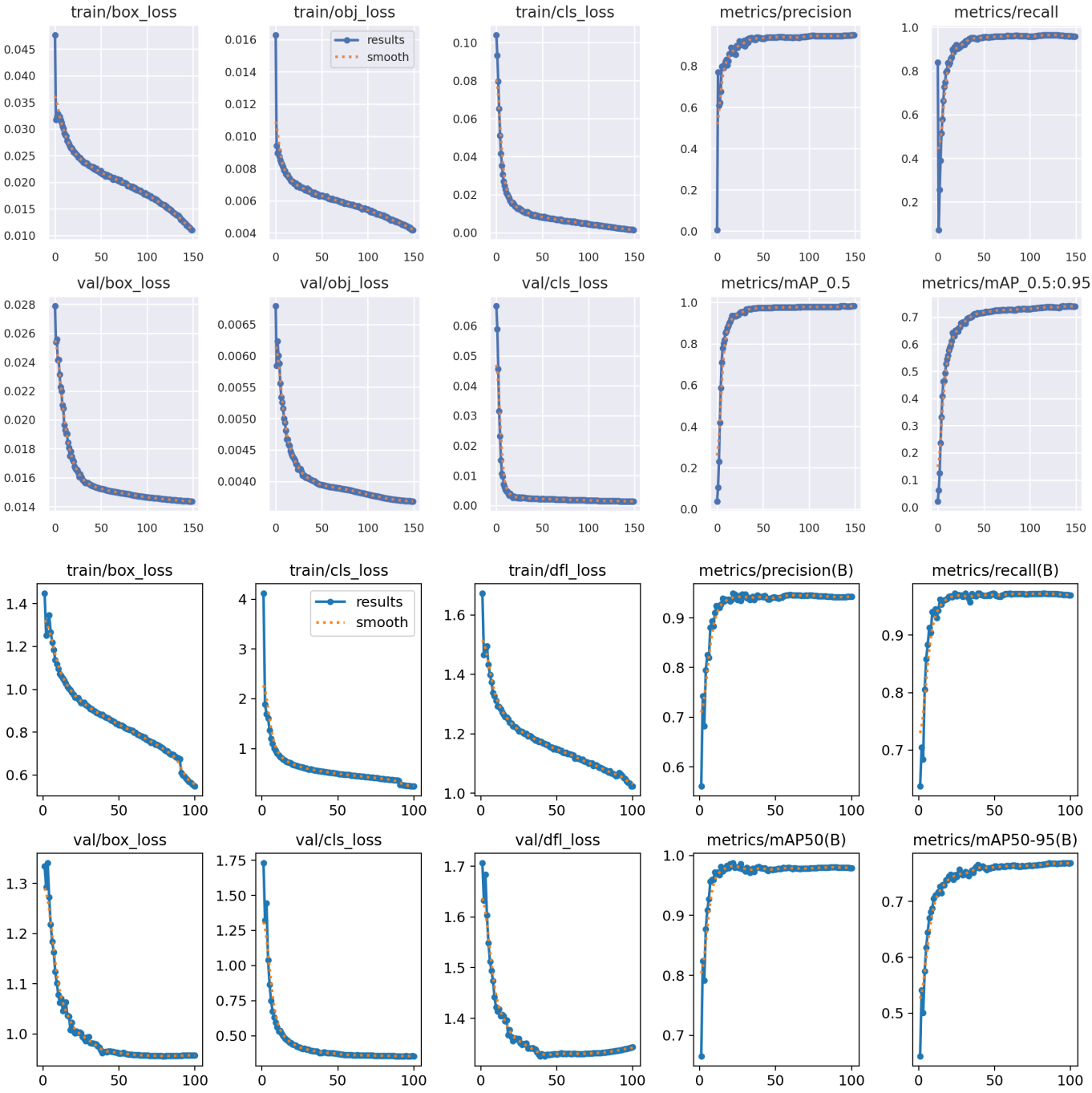


Figure 2.4: Results and loss reduction for YOLOv5(above) and YOLOv8(below)

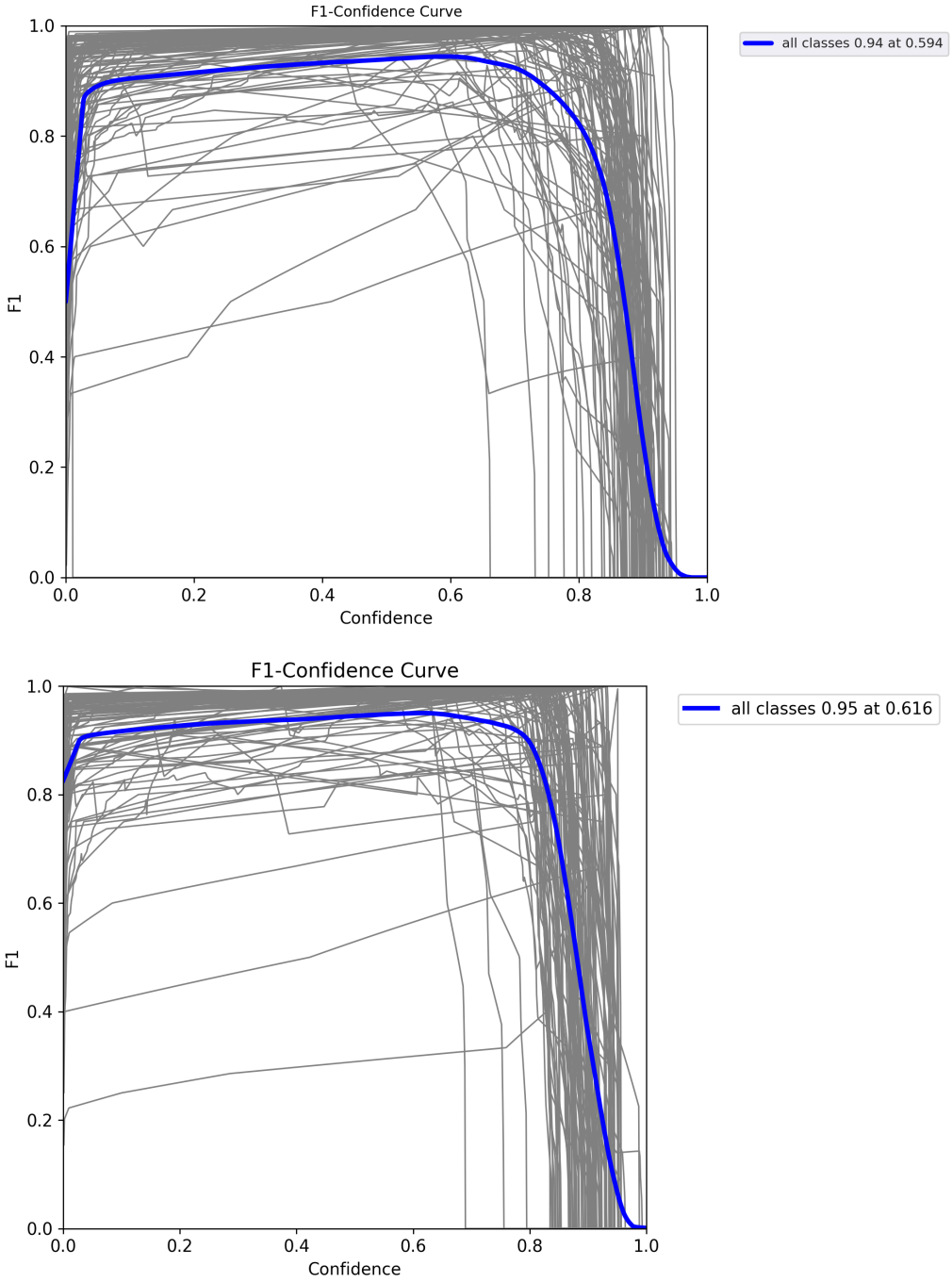


Figure 2.5: F1-confidence for YOLOv5(above) and YOLOv8(below)

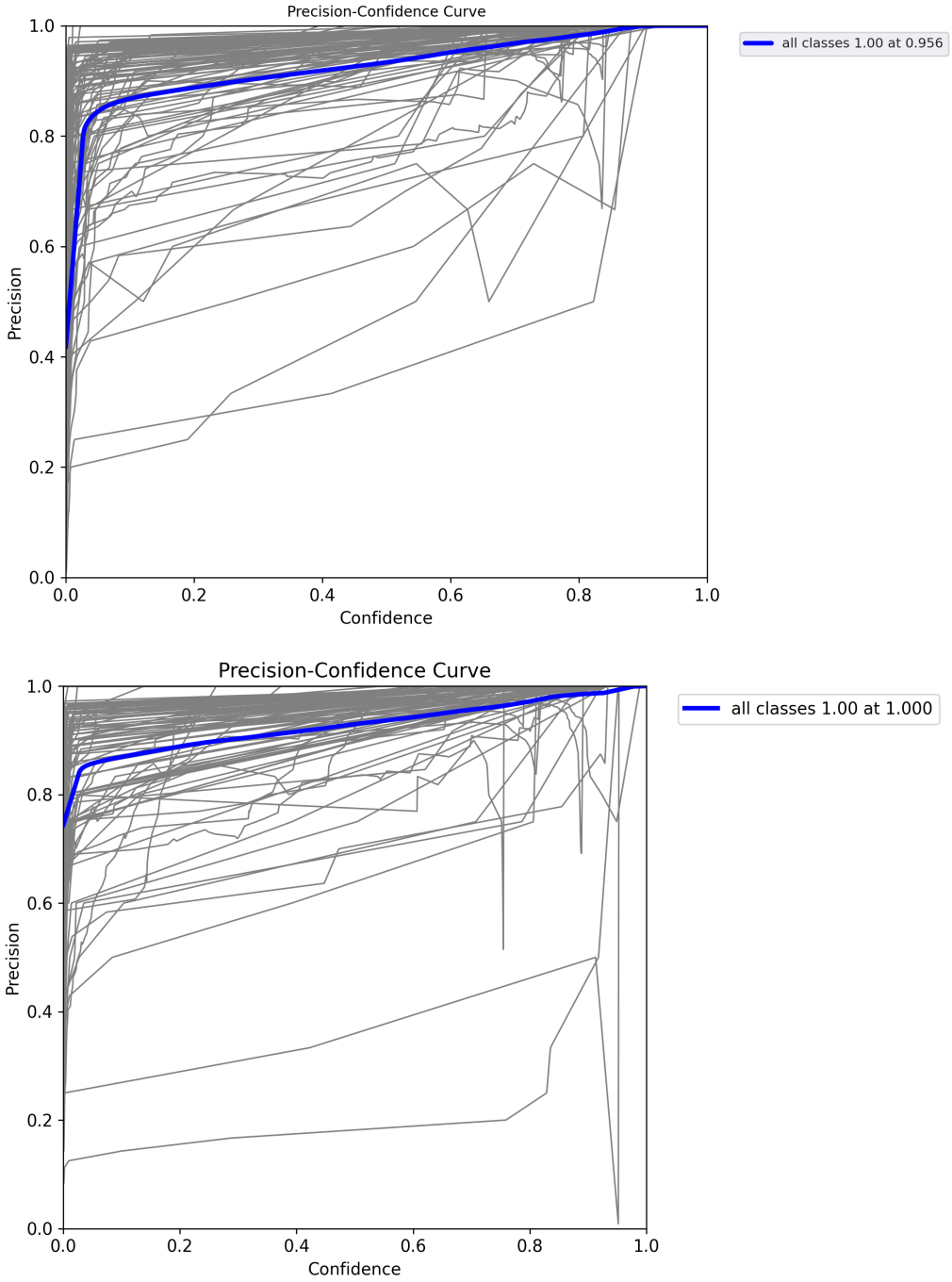


Figure 2.6: Precision curve for YOLOv5(above) and YOLOv8(below)

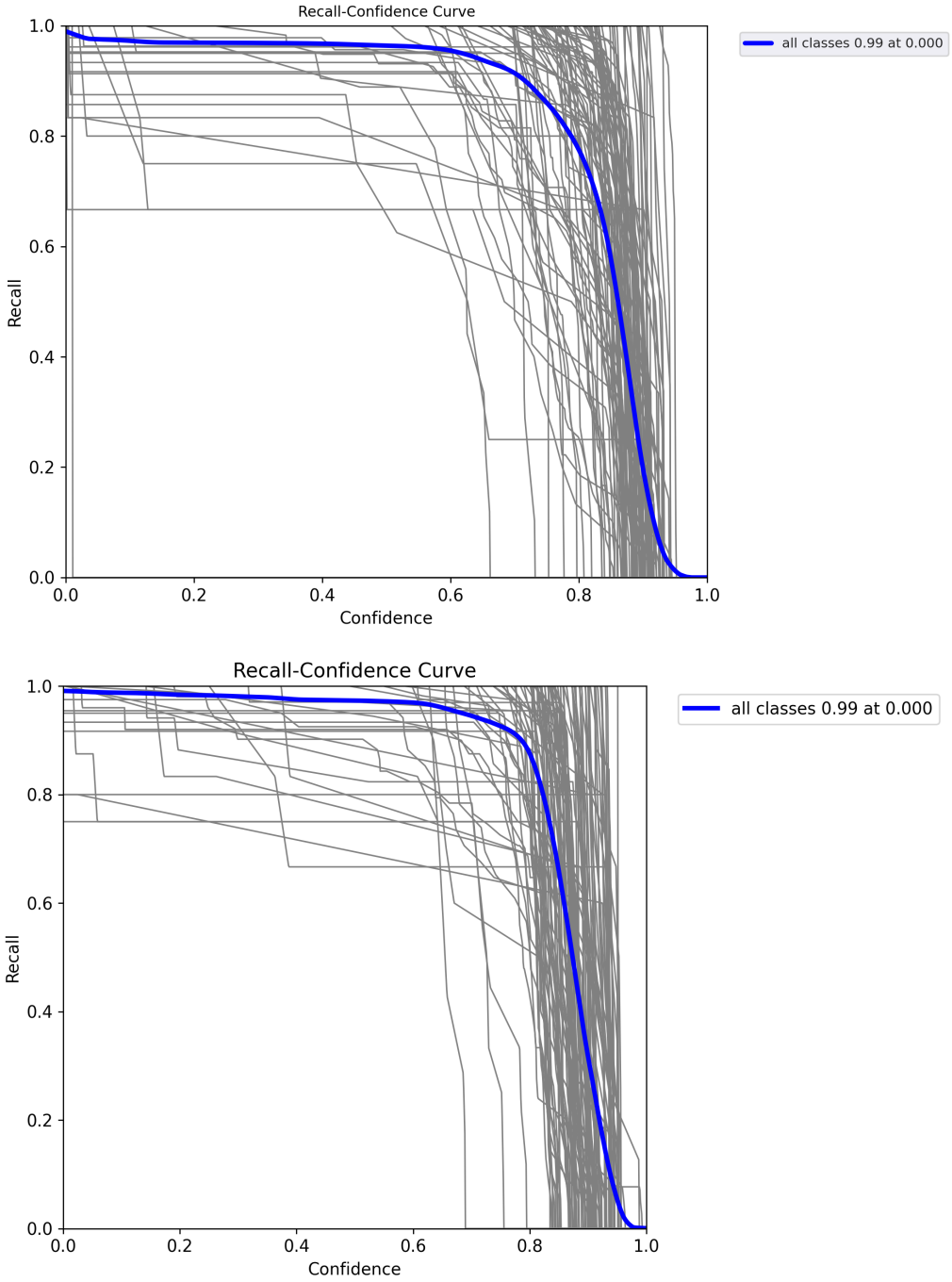


Figure 2.7: Recall curve for YOLOv5(above) and YOLOv8(below)

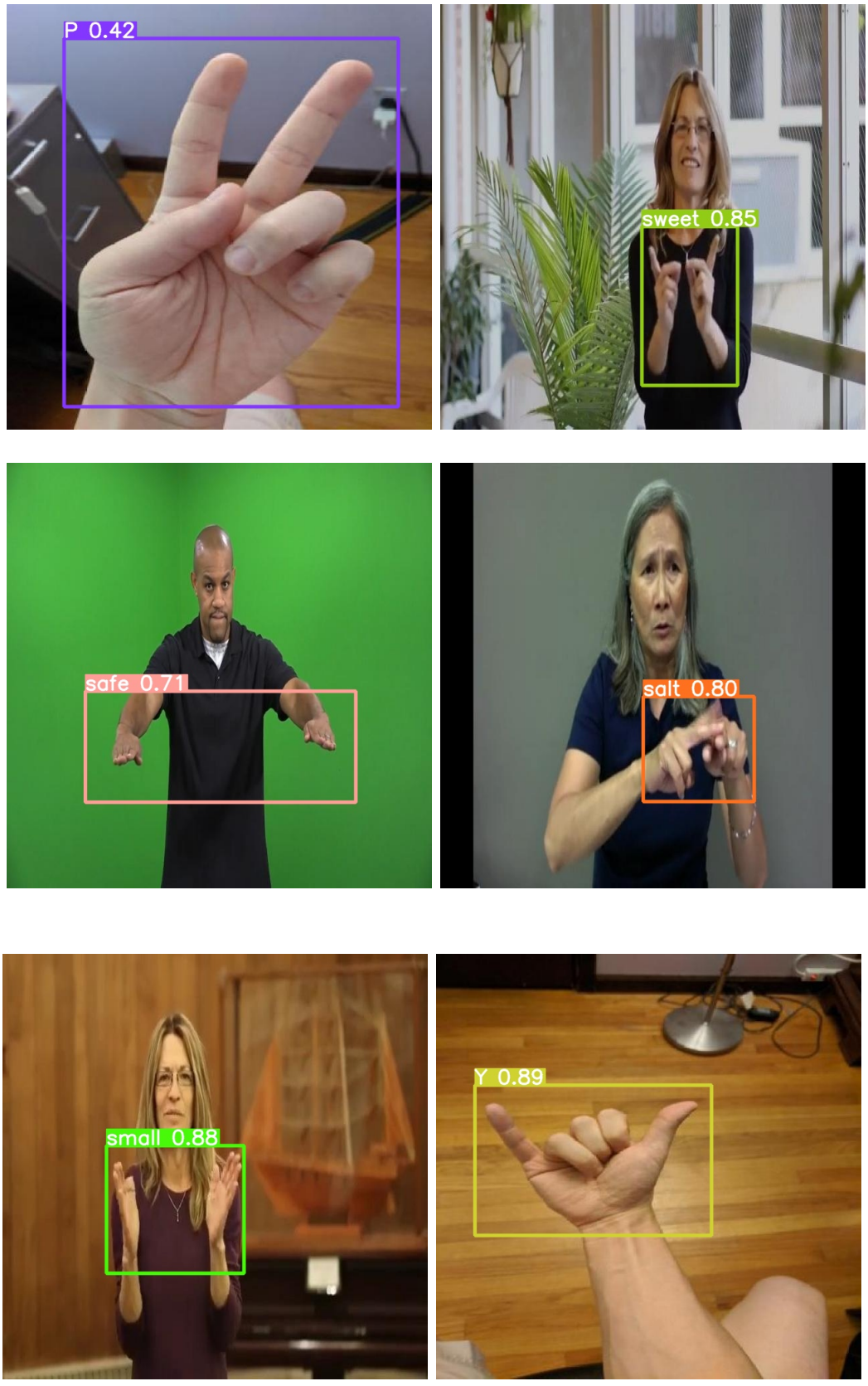


Figure 2.8: Prediction (YOLO V5)

Figure 2.9: Prediction (YOLO V8)

Chapter 3

Used Software

3.1 Introduction

In this chapter we will discuss the software used in each part in the project and how they integrate with each other. We can divide software used in the project to two main parts:

1. Jupyter notebook
2. Google colab

3.1.1 Jupyter notebook

3.1.2 Overview

It is a web-based notebook environment for interactive computing in which you can combine code execution, rich text, mathematics, plots and rich media. Jupyter Notebooks are primarily used by data professionals, particularly data analysts and data

scientists. According to the Kaggle Survey 2022 results, Jupyter Notebooks are the most popular data science IDE, used by over 80% of respondents

the dashboard will give you access only to the files and sub-folders contained within Jupyter's start-up directory (i.e., where Jupyter or Anaconda is installed). However, the start-up directory can be changed.

3.1.3 Main features of Jupyter Notebook

Originally developed for data science applications written in Python, R, and Julia, Jupyter Notebook is useful in all kinds of ways for all kinds of projects:

- **Data visualizations**:

Most people have their first exposure to Jupyter Notebook by way of a data visualization, a shared notebook that includes a rendering of some data set as a graphic. Jupyter Notebook lets you author visualizations, but also share them and allow interactive changes to the shared code and data set.

- **Code sharing**:

Cloud services like GitHub and Pastebin provide ways to share code, but they're largely non-interactive. With a Jupyter Notebook, you can view code, execute it, and display the results directly in your web browser.

- **Live interactions with code:**

Jupyter Notebook code isn't static; it can be edited and re-run incrementally in real time, with feedback provided directly in the browser. Notebooks can also embed user controls (e.g., sliders or text input fields) that can be used as input sources for code.

• **Documenting code samples:** If you have a piece of code and you want to explain line-by-line how it works, with live feedback all along the way, you could embed it in a Jupyter Notebook. Best of all, the code will remain fully functional—you can add interactivity along with the explanation, showing and telling at the same time

3.2 Programming

The main language used in this project is Python. Python's open-source libraries are not the only feature that make it favorable for machine learning and AI tasks. Python is also highly versatile and flexible, meaning it can also be used alongside other programming languages when needed. Even further, it can operate on nearly all OS and platforms on the market. Implementing deep neural networks and machine learning algorithms can be extremely time consuming, but Python offers many packages that cut down on this. It is also an object-oriented programming (OOP) language, which makes it extremely useful for efficient data use and categorization.

3.3 Hand detection

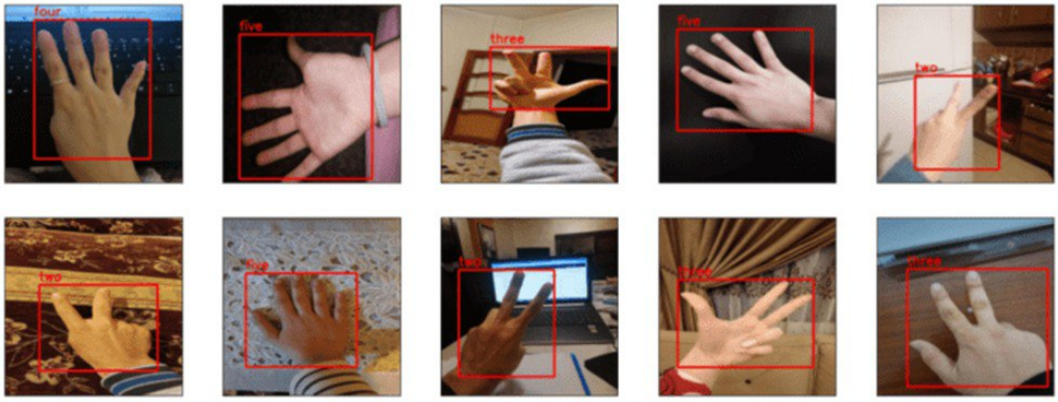
Hand detection is a computer vision task that involves identifying and tracking the hands in a digital image or video stream. It is a challenging task due to the variability in hand appearance and the complex motions that hands can perform

Hand detection is used in a variety of applications, such as:

- Sign language recognition
- Gesture control
- Virtual reality and augmented reality
- Gaming
-Security and surveillance

3.3.1 Hand Detection Using OpenCV and MediaPipe

OpenCV is a popular computer vision library that provides a variety of functions for image and video processing. MediaPipe is a cross-platform machine learning framework that provides a variety of pre-trained models for computer vision tasks, such as face detection, hand detection, and pose estimation.



Gesture recognition(Hand Detection)is an active research field in Human-Computer Interaction technology. It has many applications in virtual environment control and sign language translation, robot control, or music creation. In this machine learning project on Hand Detection, we are going to make a real-time Hand detector using the CVZONE module and OpenCV Python

3.4 Libraries

3.4.1 OpenCV

OpenCV is one of the image processing libraries in python. OpenCV provides a wide range of computer vision algorithms and functions, including image and video processing, feature detection, object recognition, machine learning, and more. It has a large community of developers and users, making it a popular choice for computer vision projects The image processing library provides access to over 2,500 state of the art and classic algorithms. Users can use OpenCV to perform several specific tasks like

removing red eyes and following eye movements. Some of the features of OpenCV include:

- Image and video capture and processing

- Object detection and tracking

- Facial recognition and detection

- Optical character recognition (OCR)

- Machine learning algorithms for image and data analysis

- 3D reconstruction and augmented reality OpenCV is easy to install and use. It comes with pre-built libraries and is compatible with multiple platforms such as Windows, Linux, macOS, iOS, and Android. It also has bindings for various programming languages like Python, Java, and MATLAB

3.4.2 CVZONE

This is a Computer vision package that makes its easy to run Image processing and AI functions. At the core it uses OpenCV and Mediapipe libraries. You can simply use pip to install the latest version of **cvzone**.

```
pip install cvzone
```

Basic Code Example

```
import cvzone

import cv2

cap = cv2.VideoCapture(0)

cap.set(3, 1280)

cap.set(4, 720)

detector = cvzone.HandDetector(detectionCon=0.5, maxHands=1)



while True:

    # Get image frame

    success, img = cap.read()

    # Find the hand and its landmarks

    img = detector.findHands(img)

    lmList, bbox = detector.findPosition(img)

    # Display

    cv2.imshow("Image", img)
```

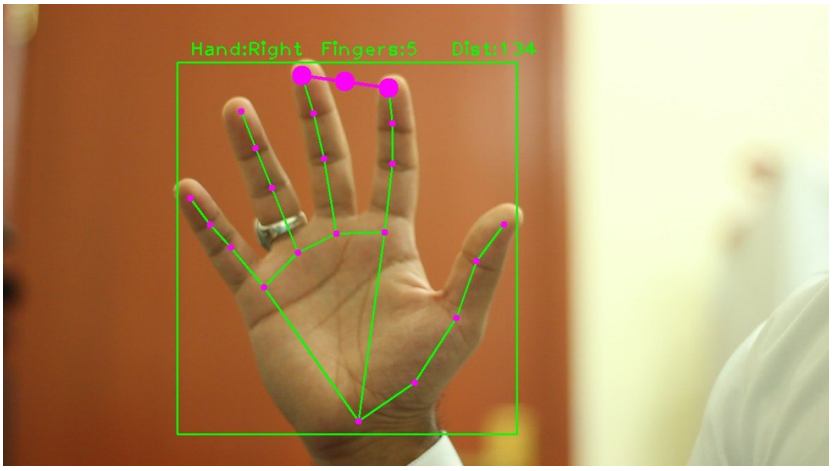
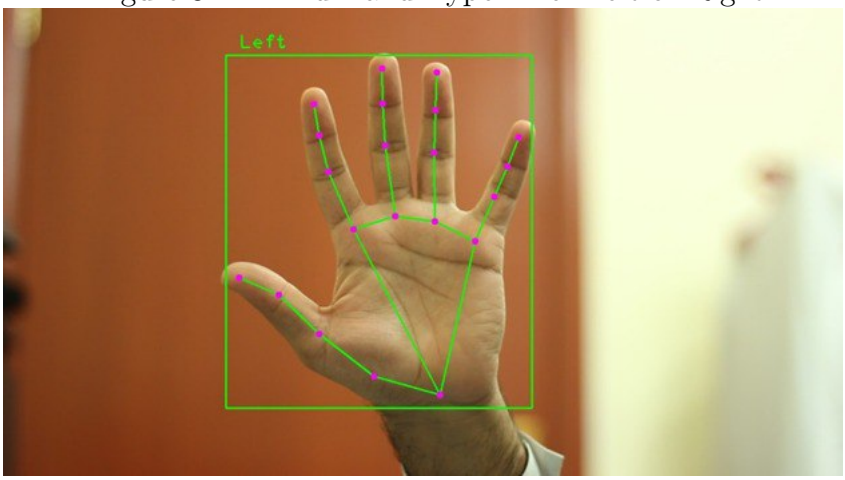
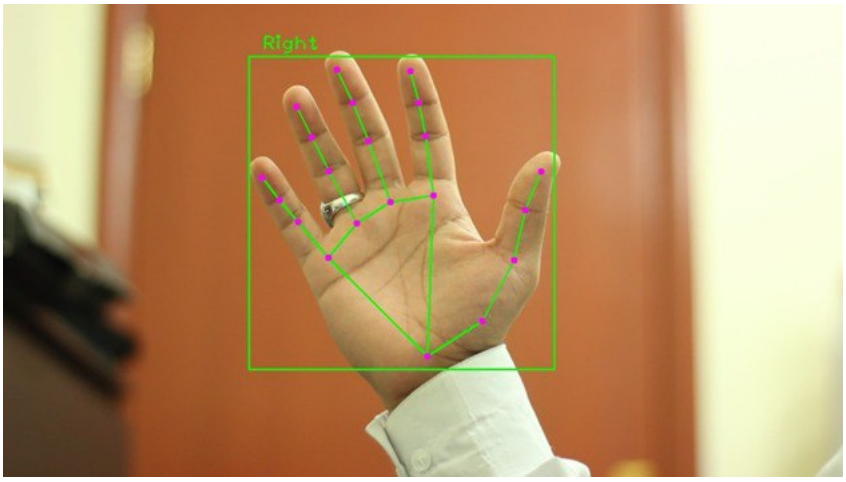


Figure 3.1: Find Hand Type – i.e. Left or Right





Find Hand Type – i.e. Left or Right

```
if lmList:

        # Find Hand Type

        myHandType = detector.handType()

        cv2.putText(img, f'Hand:{myHandType}', (bbox[0], bbox[1] - 30),

                    cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 0), 2)
```

3.4.3 Speech Recognition

Movies and TV shows love to depict robots who can understand and talk back to humans. Shows like Westworld, movies like Star Wars and I, Robot are filled with such marvels. But what if all of this exists in this day and age? Which it certainly does. You can write a program that understands what you say and respond to it. All of this is possible with the help of speech recognition. Using speech recognition in Python, you can create programs that pick up audio and understand what is being said. In this tutorial titled "Everything You Need to Know About Speech Recognition in Python", you will learn the basics of speech recognition

What is Speech Recognition?

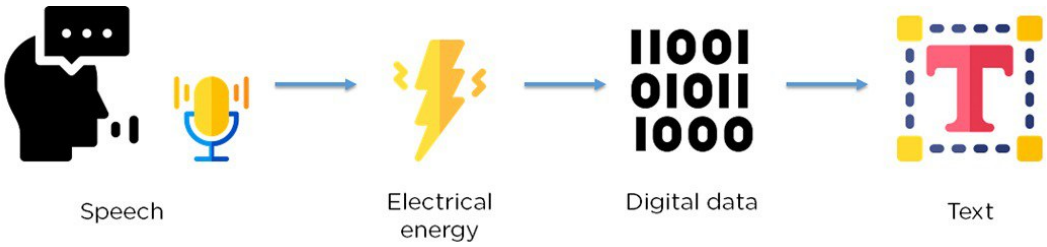
Speech Recognition incorporates computer science and linguistics to identify spoken words and converts them into text. It allows computers to understand human language.

Speech Recognition

Speech recognition is a machine's ability to listen to spoken words and identify them. You can then use speech recognition in Python to convert the spoken words into text, make a query or give a reply. You can even program some devices to respond to these spoken words. You can do speech recognition in python with the help of computer programs that take in input from the microphone, process it, and convert it into a suitable form. Speech recognition seems highly futuristic, but it is present all around you. Automated phone calls allow you to speak out your query or the query you wish to be assisted on; your virtual assistants like Siri or Alexa also use speech recognition to talk to you seamlessly

How Does Speech Recognition work?

Speech recognition in Python works with algorithms that perform linguistic and acoustic modeling. Acoustic modeling is used to recognize phenones/phonetics in our speech to get the more significant part of speech, as words and sentences



Working of Speech Recognition

Speech recognition starts by taking the sound energy produced by the person speaking and converting it into electrical energy with the help of a microphone. It then converts this electrical energy from analog to digital, and finally to text. It breaks the audio data down into sounds, and it analyzes the sounds using algorithms to find the most probable word that fits that audio. All of this is done using **Natural Language Processing** and **Neural Networks**. Hidden Markov models can be used to find temporal patterns in speech and improve accuracy. Speech Recognition package. It allows:
- Easy speech recognition from the microphone.

- Makes it easy to transcribe an audio file.

- It also lets us save audio data into an audio file.

- It also shows us recognition results in an easy-to-understand format.

Speech Recognition package. It allows:

- Easy speech recognition from the microphone.

- Makes it easy to transcribe an audio file.

- It also lets us save audio data into an audio file.

- It also shows us recognition results in an easy-to-understand format

Installing Speech Recognition

Installing speech recognition in Python is a crucial step towards incorporating powerful voice recognition capabilities into your projects. Speech recognition, a Python library, facilitates easy access to various speech recognition engines and APIs, making

it an indispensable tool for a diverse array of applications. Let's embark on a journey to explore the process of installing Speech Recognition and unlock its potential for your projects.

Methods and Usage

The Recognizer class provides a set of methods for performing speech recognition tasks, including: - recognize-google(): This method performs speech recognition using the Google Web Speech API. It requires an internet connection to send audio data to Google's servers for processing.

- recognize-sphinx(): Utilizes the CMU Sphinx engine for offline speech recognition. This method is suitable for scenarios where internet connectivity is unavailable or for applications with privacy concerns.

- recognize-wit(): Interfaces with the Wit.ai API for speech recognition. Wit.ai offers natural language processing capabilities, enabling developers to extract intent and entities from the transcribed text.

- listen(): Captures audio input from the specified source, such as the microphone or an audio file, and returns a SpeechRecognition AudioData object containing the raw audio data.

- record(): Records audio input from the microphone for a specified duration and returns the recorded audio as a SpeechRecognition AudioData object.

Example Usage

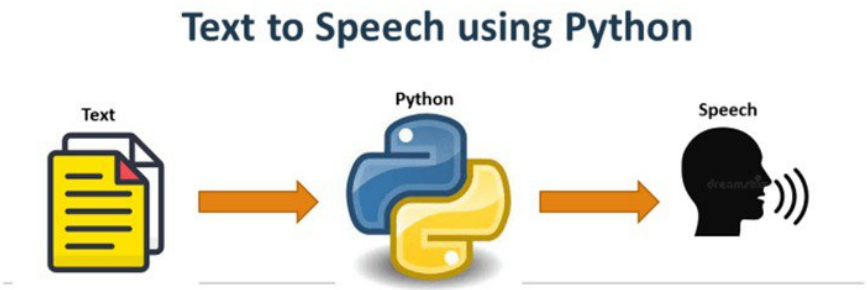
```
import speech_recognition as sr

# Create a Recognizer instance
recognizer = sr.Recognizer()

# Capture audio input from the microphone
with sr.Microphone() as source:
 print("Speak something...")
 audio_data = recognizer.listen(source)

# Perform speech recognition using Google Web Speech API
try:
 text = recognizer.recognize_google(audio_data)
 print("You said:", text)
except sr.UnknownValueError:
 print("Sorry, could not understand audio.")
except sr.RequestError as e:
 print("Error: Could not request results from Google Speech Recognition
service;"
```

Text-To-Speech Conversion



pyttsx3 Library

pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline, and is compatible with both Python 2 and 3. It needs only few lines

of code to get the basic program running. Install the dependent package with the command below.

3.5 Yolo Algorithm

3.5.1 What is YOLO?

You Only Look Once (YOLO) is a state-of-the-art, real-time object detection algorithm introduced in 2015 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in their famous research paper "You Only Look Once: Unified, Real-Time Object Detection". The authors frame the object detection problem as a regression problem instead of a classification task by spatially separating bounding boxes and associating probabilities to each of the detected images using a single convolutional neural network (CNN). By taking the Image Processing with Keras in Python course, you will be able to build Keras based deep neural networks for image classification tasks. If you are more interested in Pytorch, Deep Learning with Pytorch will teach you about convolutional neural networks and how to use them to build much more powerful models

What Makes YOLO Popular for Object Detection?

Some of the reasons why YOLO is leading the competition include its: - Speed

- Detection accuracy

- Good generalization

- Open-source

1- Speed

YOLO is extremely fast because it does not deal with complex pipelines. It can process images at 45 Frames Per Second (FPS). In addition, YOLO reaches more than twice the mean Average Precision (mAP) compared to other real-time systems, which makes it a great candidate for real-time processing. From the graphic below, we observe that YOLO is far beyond the other object detectors with 91 FPS.

2- High detection accuracy

YOLO is far beyond other state-of-the-art models in accuracy with very few back-

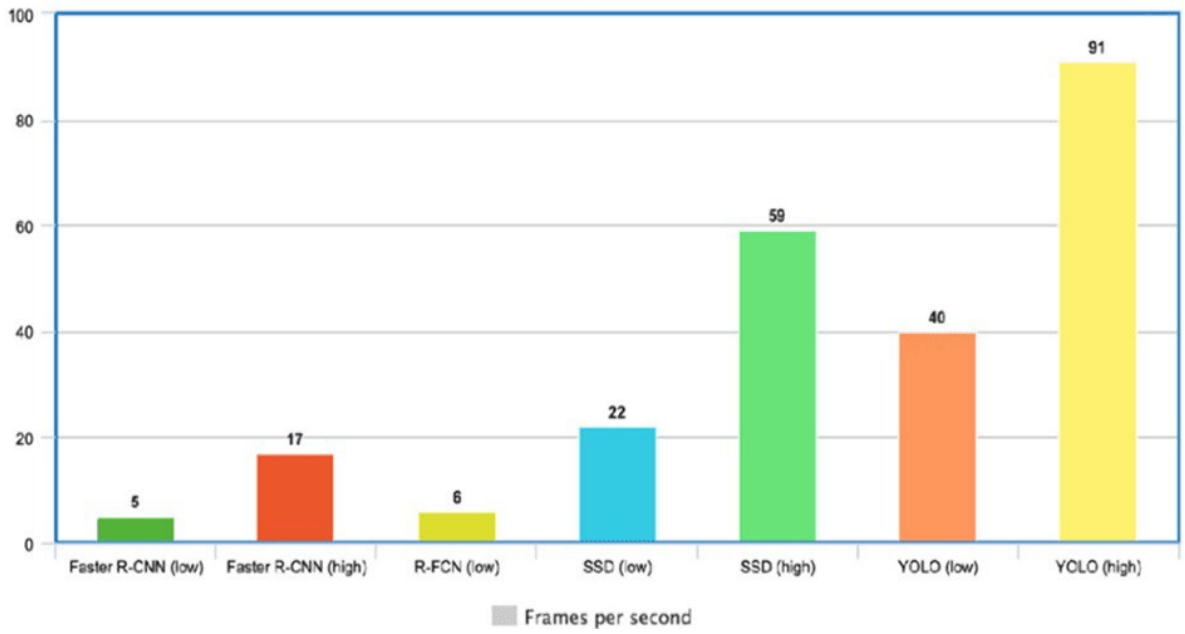


Figure 3.2: YOLO Speed compared to other state-of-the-art object detectors

ground errors.

3- Better generalization

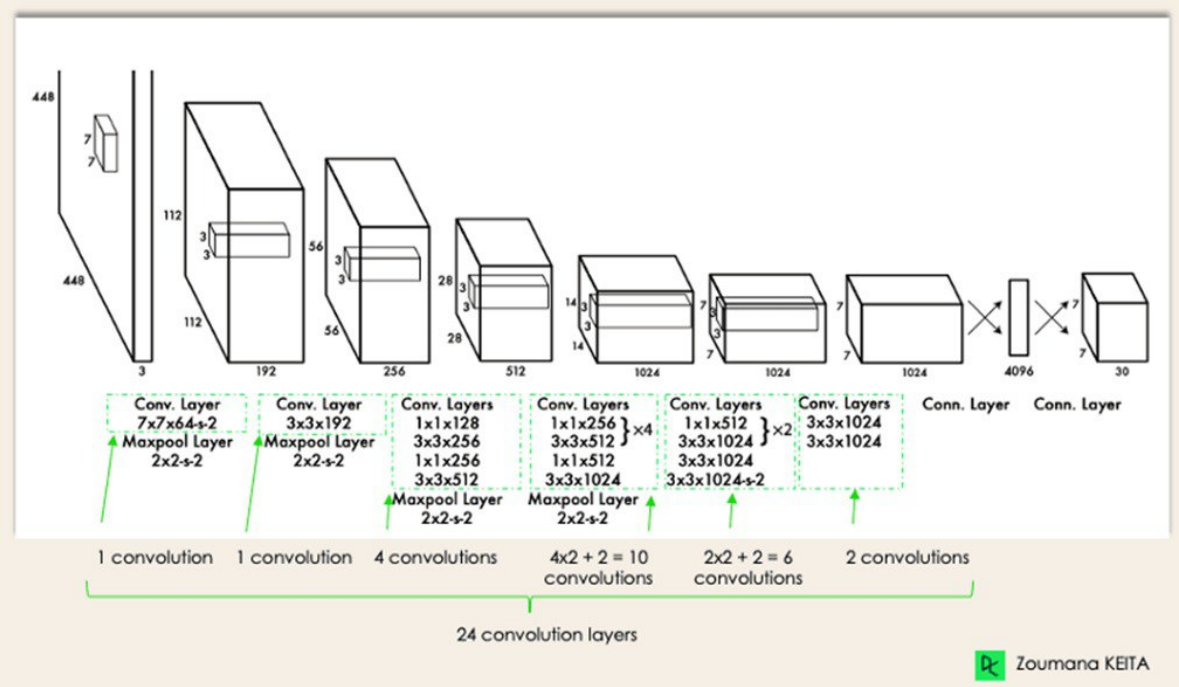
This is especially true for the new versions of YOLO, which will be discussed later in the article. With those advancements, YOLO pushed a little further by providing a better generalization for new domains, which makes it great for applications relying on fast and robust object detection. For instance the Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks paper shows that the first version YOLOv1 has the lowest mean average precision for the automatic detection of melanoma disease, compared to YOLOv2 and YOLOv3.

4- Open source

Making YOLO open-source led the community to constantly improve the model. This is one of the reasons why YOLO has made so many improvements in such a limited time.

3.5.2 YOLO Architecture

YOLO architecture is similar to GoogleNet. As illustrated below, it has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers.



YOLO Architecture from the original paper (Modified by Author)

The architecture works as follows:

- Resizes the input image into 448x448 before going through the convolutional network.

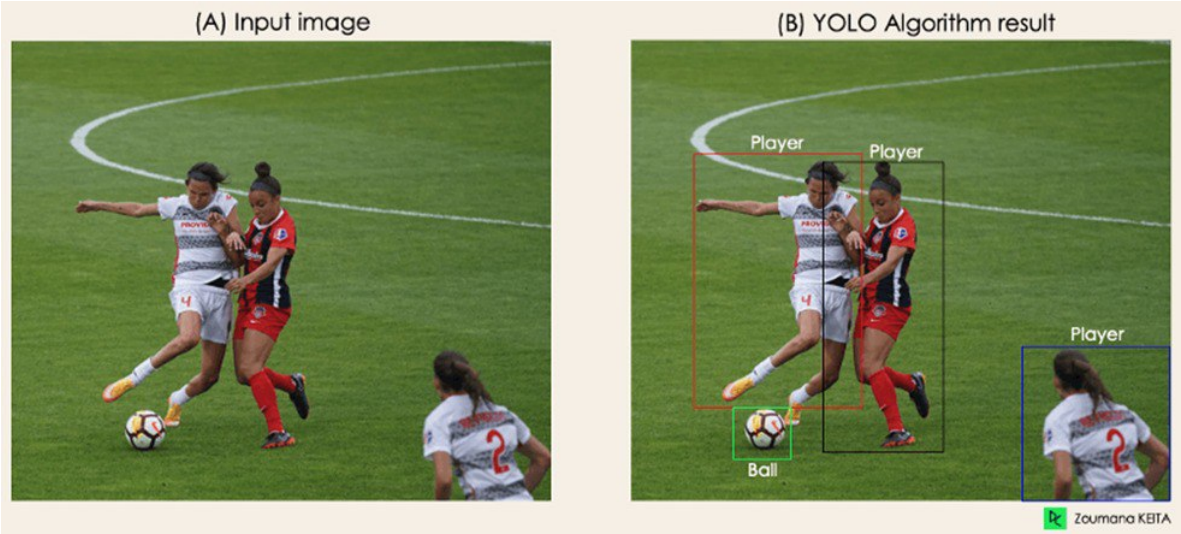
- A 1x1 convolution is first applied to reduce the number of channels, which is then followed by a 3x3 convolution to generate a cuboidal output.

- The activation function under the hood is ReLU, except for the final layer, which uses a linear activation function.

- Some additional techniques, such as batch normalization and dropout, respectively regularize the model and prevent it from overfitting. By completing the Deep Learning in Python course, you will be ready to use Keras to train and test complex, multi-output networks and dive deeper into deep learning

3.5.3 How Does YOLO Object Detection Work?

Now that you understand the architecture, let's have a high-level overview of how the YOLO algorithm performs object detection using a simple use case. "Imagine you built a YOLO application that detects players and soccer balls from a given image. But how can you explain this process to someone, especially non-initiated people? → That is the whole point of this section. You will understand the whole process of how YOLO performs object detection; how to get image (B) from image (A)"



The algorithm works based on the following four approaches:

- Residual blocks
- Bounding box regression
- Intersection Over Unions or IOU for short
- Non-Maximum Suppression.

Let's have a closer look at each one of them

1- Residual blocks

This first step starts by dividing the original image (A) into NxN grid cells of equal shape, where N in our case is 4 shown on the image on the right. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value.

2- Bounding box regression

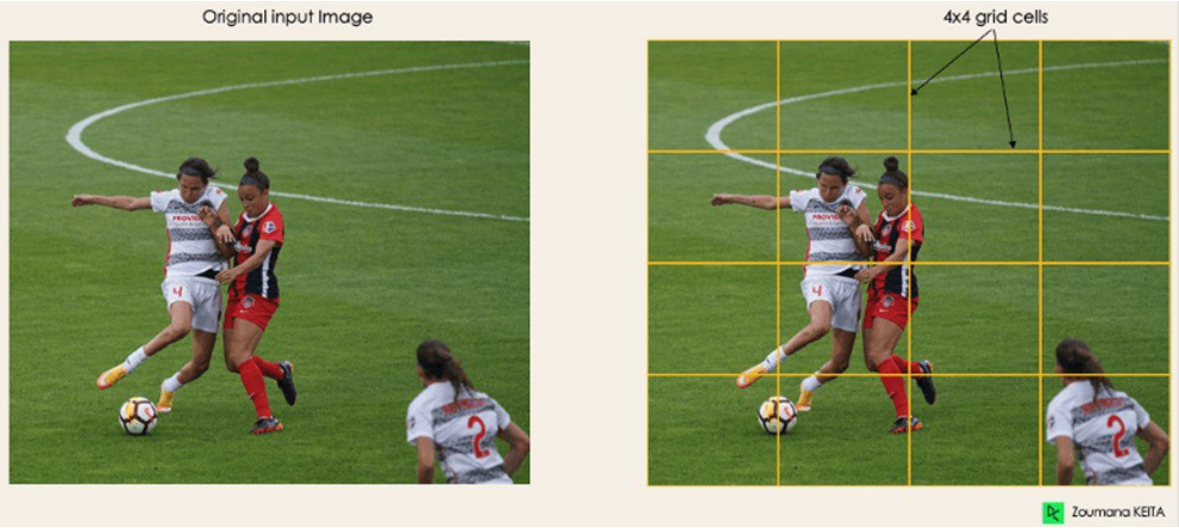


Figure 3.3: Residual blocks

The next step is to determine the bounding boxes which correspond to rectangles highlighting all the objects in the image. We can have as many bounding boxes as there are objects within a given image. YOLO determines the attributes of these bounding boxes using a single regression module in the following format, where Y is the final vector representation for each bounding box. Y = [pc, bx, by, bh, bw, c1, c2] This is especially important during the training phase of the model.

- pc corresponds to the probability score of the grid containing an object. For instance, all the grids in red will have a probability score higher than zero. The image on the right is the simplified version since the probability of each yellow cell is zero (insignificant)

- bx, by are the x and y coordinates of the center of the bounding box with respect to the enveloping grid cell.

- bh, bw correspond to the height and the width of the bounding box with respect to the enveloping grid cell.

- c1 and c2 correspond to the two classes Player and Ball. We can have as many classes as your use case requires. To understand, let's pay closer attention to the player on the bottom right.

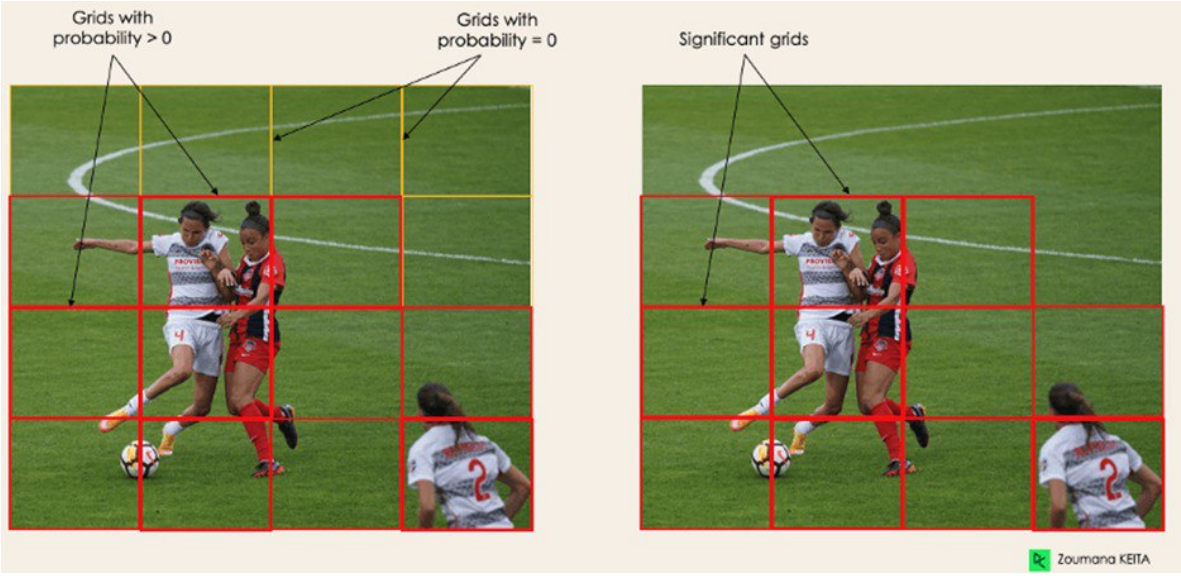
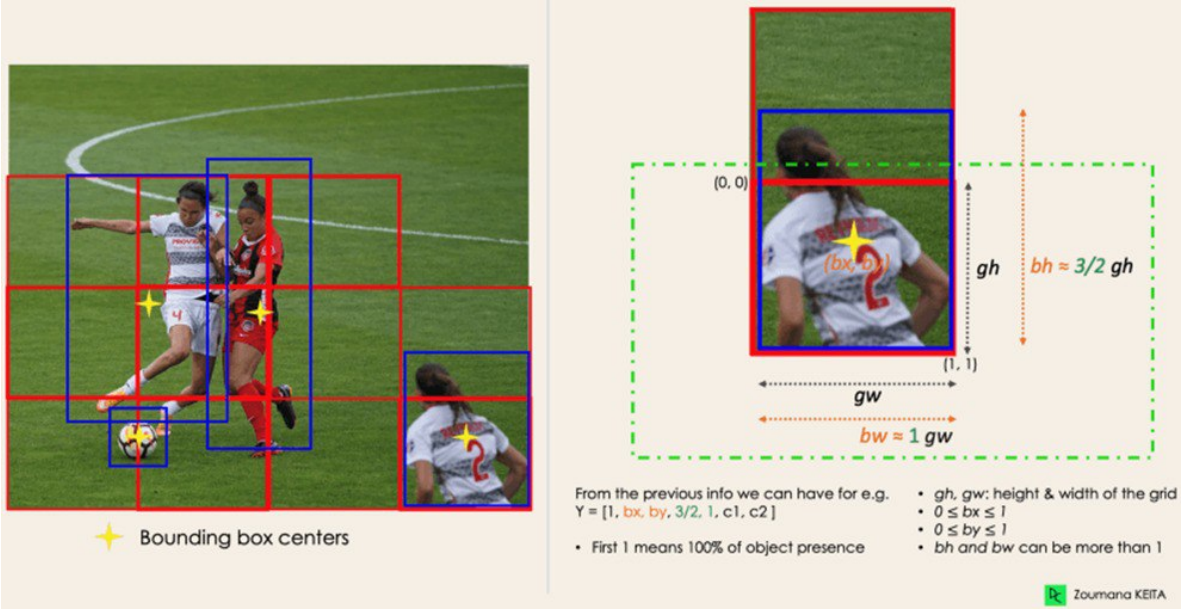


Figure 3.4: Bounding box



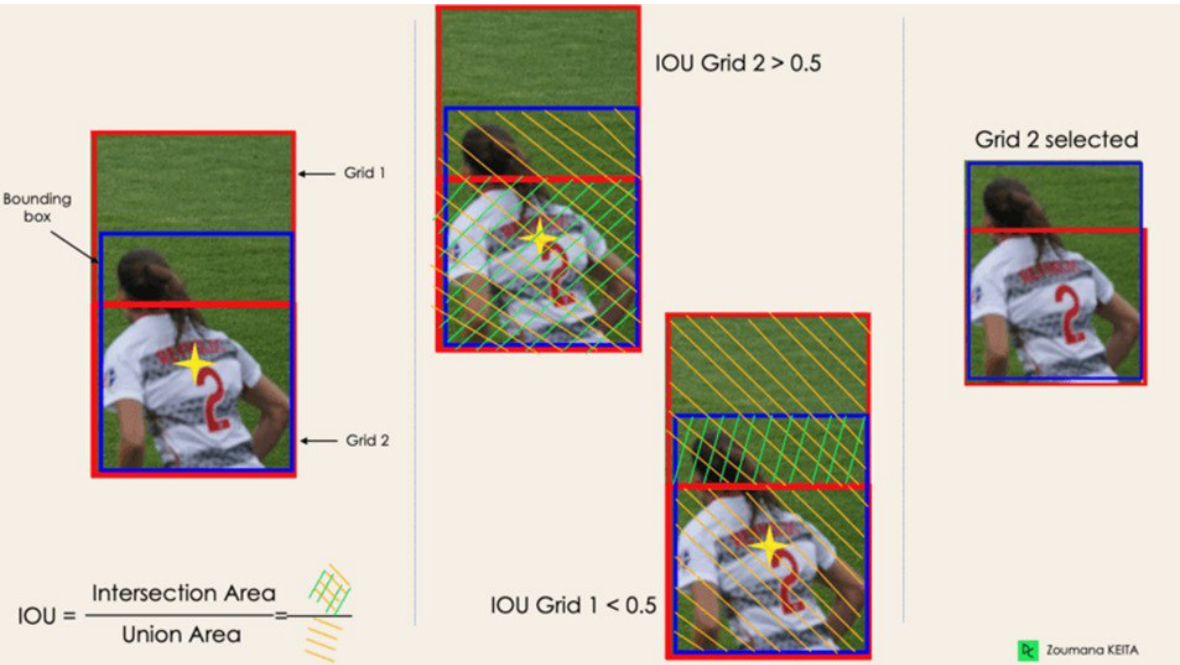
3- Intersection Over Unions or IOU

Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant. Here is the logic behind it:

- The user defines its IOU selection threshold, which can be, for instance, 0.5.

- Then YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area.

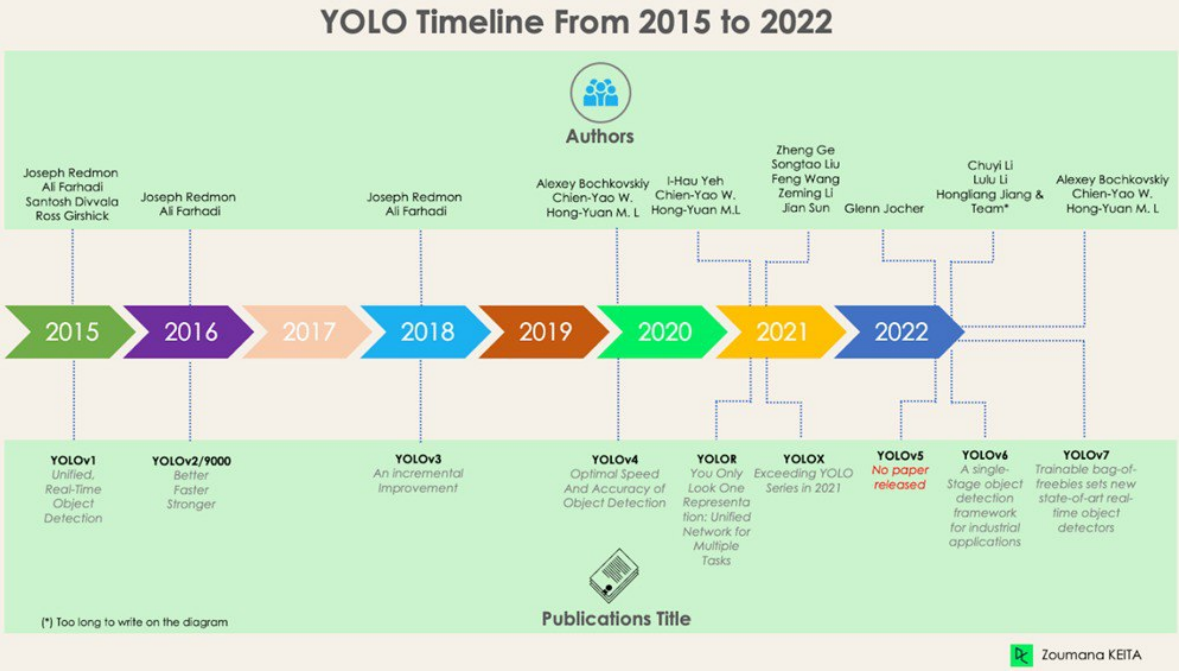
- Finally, it ignores the prediction of the grid cells having an IOU $\leq thresholdand$ considers those with an $IOU >$threshold. Below is an illustration of applying the grid selection process to the bottom left object. We can observe that the object originally had two grid candidates, then only "Grid 2" was selected at the end.



4- Non-Max Suppression or NMS

Setting a threshold for the IOU is not always enough because an object can have multiple boxes with IOU beyond the threshold, and leaving all those boxes might include noise. Here is where we can use NMS to keep only the boxes with the highest probability score of detection

YOLO, YOLOv2, YOLO9000, YOLOv3, YOLOv4, YOLOR, YOLOX, YOLOv5, YOLOv6, YOLOv7, YOLOv8 and Differences Since the first release of YOLO in 2015, it has evolved a lot with different versions. In this section, we will understand the differences between each of these versions.



3.6 YOLO V5

To address acceptable inference speeds and size, Yolov5 was chosen for modeling. This was released in June 10th of this year, and is still in active development. Although Yolov5 by Ultralytics is not created by the original Yolo authors, Yolo v5 is said to be faster and more lightweight, with accuracy on par with Yolo v4 which is widely considered as the fastest and most accurate real-time object detection model.

Yolo was designed as a convolutional neural network for real time object detection. Its more complex than basic classification as object detection needs to identify the objects and locate where it is on the image. This single stage object detector, has 3 main components: The backbone basically extracts important features of an image, the neck mainly uses feature pyramids which help in generalizing the object scaling for better performance on unseen data. The model head does the actual detection part where anchor boxes are applied on features that generate output vectors. These vectors include the class probabilities, the objectness scores, and bounding boxes. The model used was yolov5m with transfer learning on pretrained weights.

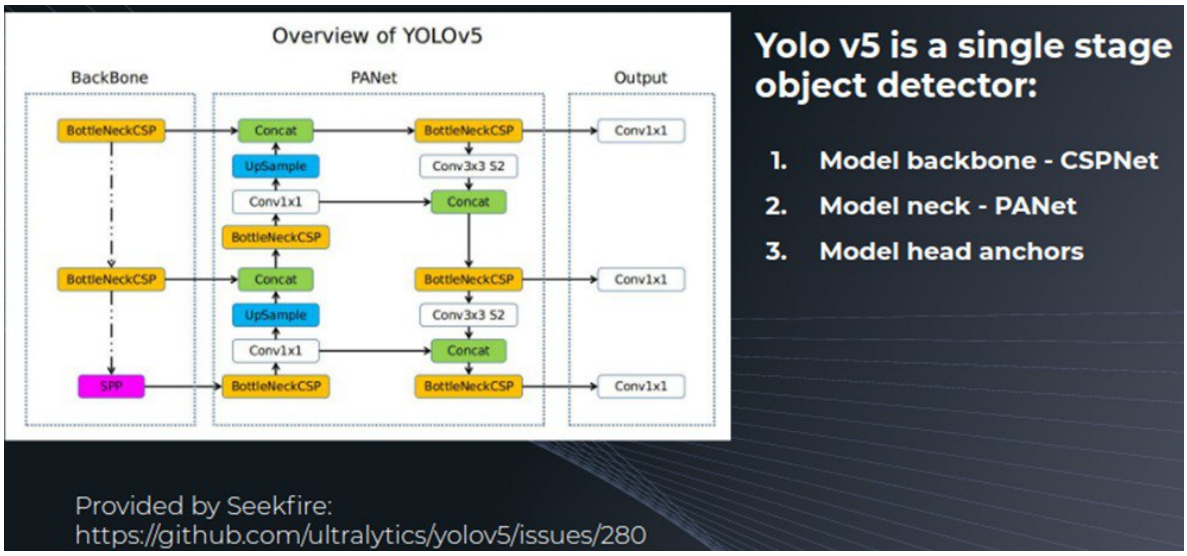
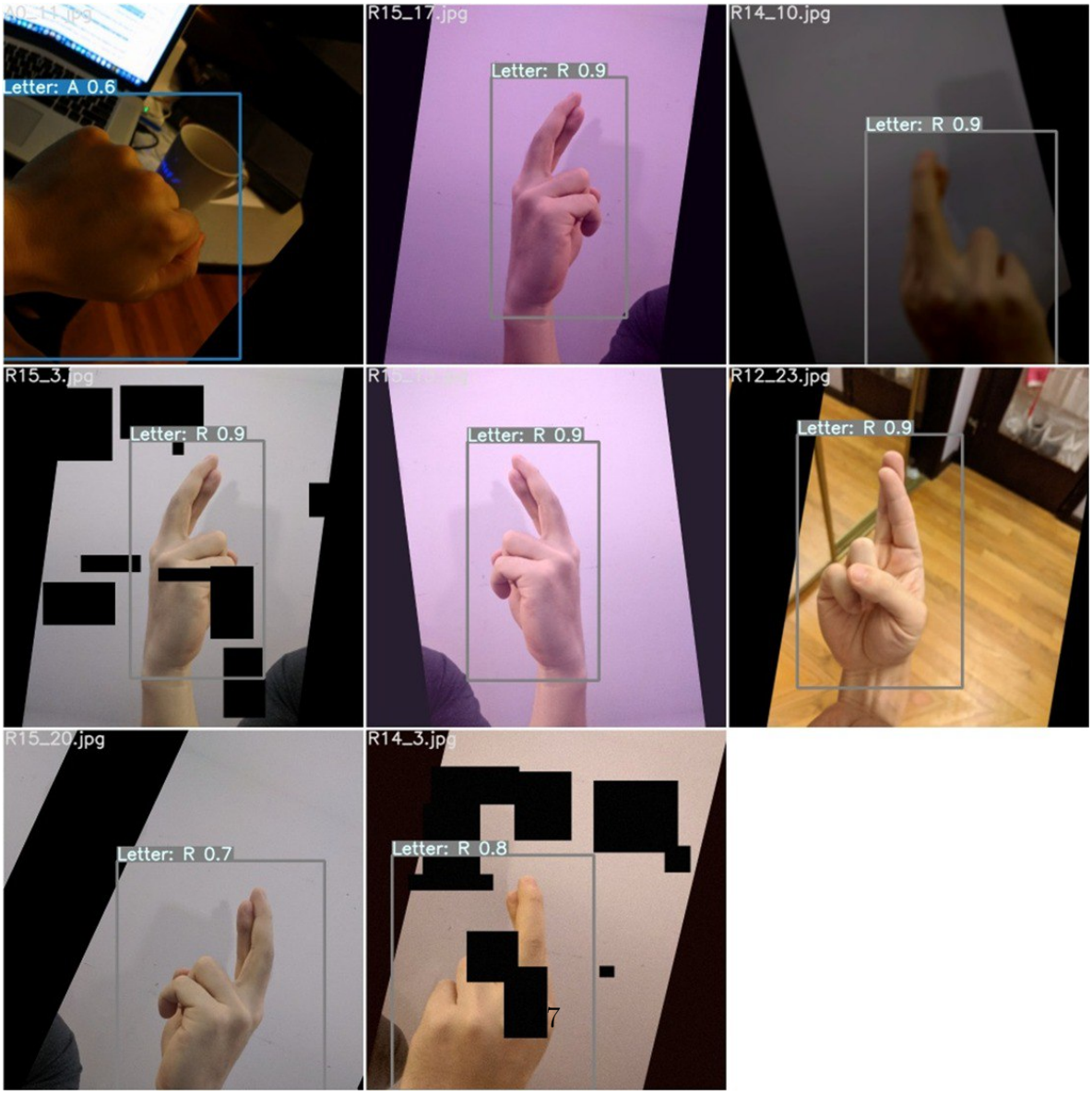


Figure 3.5: YOLO V5



3.7 YOLO V8

Architecture: YOLOv8 represents an evolution in the YOLO family with improvements in both architecture and functionality. It aims to enhance accuracy and efficiency.

Anchor-free: Moves towards an anchor-free approach, simplifying the model by not requiring predefined anchor boxes.

Backbone: Utilizes a more advanced backbone network for improved feature extraction and higher accuracy.

Neck: Employs an improved neck design to enhance multi-scale feature fusion, crucial for detecting objects of varying sizes.

Head: Improved detection head to predict bounding boxes and classes more accurately and efficiently.

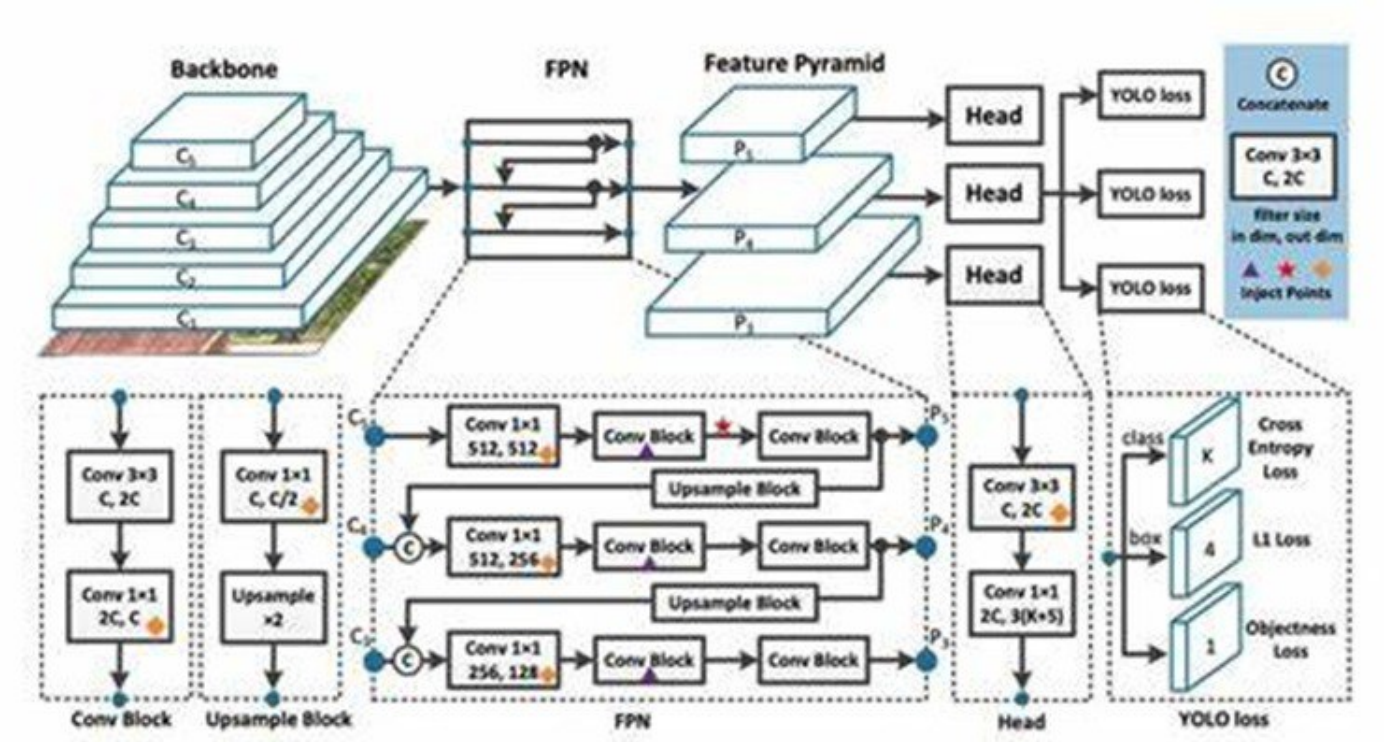


Figure 3.6: YOLO V8

Benefits:

Higher Accuracy: With architectural improvements and an anchor-free approach, YOLOv8 offers higher accuracy in object detection tasks

Simplification: The anchor-free design simplifies the training process and reduces the complexity of the model.

Adaptability: Better suited for a wider range of applications with enhanced flexibility in handling various object scales and aspect ratios.

Cutting-edge Performance: Incorporates the latest research advancements in object detection, providing state-of-the-art performance. Comparison

Performance: YOLOv8 generally provides higher accuracy and better detection capabilities due to its updated architecture and anchor-free approach. Speed: Both models are optimized for speed, but YOLOv5 might still be slightly faster in certain scenarios due to its more mature optimization.

Complexity: YOLOv8 simplifies the detection process by eliminating the need for anchor boxes, potentially making it easier to train and tune. **Community and Ecosystem:** YOLOv5 benefits from a more extensive community and ecosystem, providing robust support and resources. However, YOLOv8 is catching up as it integrates newer research insights.

Conclusion

Both YOLOv5 and YOLOv8 have their own strengths and are suitable for different use cases. YOLOv5 is highly reliable with a strong support system, making it great for production-ready applications needing real-time performance. YOLOv8, with its improved accuracy and anchor-free design, is ideal for applications that demand the latest advancements in object detection technology. Choosing between them depends on the specific requirements of your project— whether you prioritize speed and established support (YOLOv5) or cutting-edge accuracy and simplified training (YOLOv8)

Chapter 4

3D Avatar Design and Animation

4.1 Introduction

The utilization of 3D avatars in sign language translation represents a significant advancement in bridging communication barriers for individuals with speech impairments or hearing loss. Unlike traditional systems that rely on text or 2D animations, 3D avatars offer a more natural and expressive means of conveying signs, and enhancing the overall communication experience.

Sign language, as a primary mode of communication for many individuals with speech impairments or hearing loss, poses unique challenges in translation and interpretation. Traditional translation methods often fall short of capturing the nuances and expressiveness inherent in sign language. The integration of 3D avatars into sign language translation technology opens new possibilities for enhancing communication. By leveraging the capabilities of 3D modeling and animation software, such as Blender, we can create dynamic and lifelike avatars capable of accurately mimicking sign language gestures and expressions.

Throughout this chapter, we will explore the various stages of 3D avatar design and animation, from initial modeling and rigging to dynamic animation and rendering. We will discuss the tools, techniques, and methodologies employed in the creation of 3D avatars tailored specifically for sign language translation. Additionally, we will examine

the challenges and considerations involved in optimizing rendering times and ensuring seamless integration with sign-language translation systems.

4.1.1 What is a 3D Model?

3D, or three-dimensional, refers to the three spatial dimensions of width, height, and depth. The physical world and everything that is observed in it are three-dimensional. While many flat images such as films and photographs register visually as two-dimensional (2D) to the human brain, nothing can physically exist without all three dimensions.

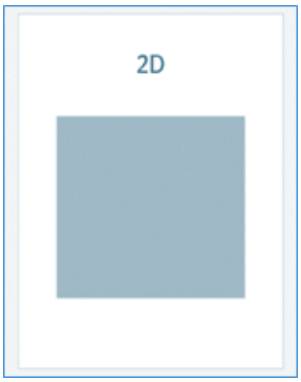


Fig. 1: 2D

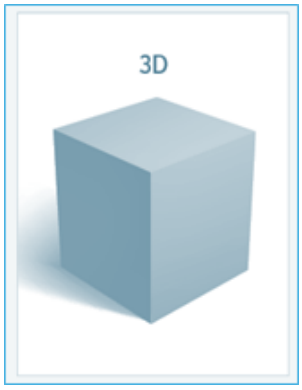


Fig. 2: 3D

A **3D model** is a digital representation of a three-dimensional object or scene created using specialized software tools. Unlike 2D images, which only have height and width, 3D models also incorporate depth, allowing them to accurately depict the spatial relationships between objects. These models can range from simple geometric shapes to highly detailed and realistic renderings of complex structures, characters, or environments.

In the context of 3D avatar design for sign language translation, a 3D model serves as the foundation for creating the visual representation of the avatar. It defines the shape, structure, and proportions of the avatar's body, limbs, and facial features, providing a framework for further customization and animation. In the context of sign language translation, 3D models provide a visually compelling and expressive means of conveying sign language gestures and expressions, enhancing communication accessibility and inclusivity for individuals with speech impairments or hearing loss.

Figure 3: 3D MODEL

4.1.2 Limitations of 2D Animation for Sign Language

While 2D animation has been widely used for various purposes, including entertainment and educational content, it comes with several limitations when applied to sign language translation. These limitations can impact the accuracy, expressiveness, and effectiveness of conveying sign language gestures and expressions. Here are some key limitations of 2D animation in the context of sign language:

1. **Lack of Depth and Dimensionality:** 2D animations typically lack the depth and dimensionality required to accurately represent the spatial relationships and movements involved in sign language. Sign language relies heavily on handshapes, movements, and facial expressions, which may not be fully captured in two-dimensional representations.

2. **Limited Expressiveness:** Sign language is inherently expressive, with subtle nuances and variations in handshapes, gestures, and facial expressions conveying different meanings. 2D animations may struggle to convey the full range

of expressiveness and intricacies present in sign language, leading to potential misinterpretations or loss of meaning.

3. **Challenges in Lip Syncing:** In sign language interpretation, facial expressions and lip movements play a crucial role in conveying tone, emotion, and emphasis. Achieving accurate lip-syncing in 2D animations can be challenging, especially when syncing facial expressions with sign language gestures in real time.

4. **Complexity in Animation:** Creating lifelike animations for sign language gestures in 2D can be complex and time-consuming. Achieving smooth transitions between handshapes, gestures, and facial expressions may require meticulous frame-by-frame animation, resulting in increased production time and costs.

While 2D animation techniques have been valuable in various contexts, they may not fully meet the requirements and demands of sign language translation. The transition to 3D animation offers potential solutions to overcome these limitations and enhance the accuracy, expressiveness, and accessibility of sign language interpretation in digital environments.

4.1.3 Advantages of 3D Avatars for Sign Language Translation

The integration of 3D avatars into sign language translation technology offers numerous advantages over traditional 2D animation methods. These advantages enhance the accuracy, expressiveness, and accessibility of sign language interpretation, ultimately improving communication accessibility for individuals with speech impairments or hearing loss. Here are some key advantages of using 3D avatars for sign language translation:

1. **Enhanced Realism and Depth:** 3D avatars provide a more realistic and immersive representation of sign language gestures and expressions by incorporating depth, dimensionality, and lifelike movements, improving comprehension and making communication more inclusive.

2. **Expressive Facial Animations:** Facial expressions play a crucial role in sign language interpretation, conveying emotions, emphasis, and nuances of mean-

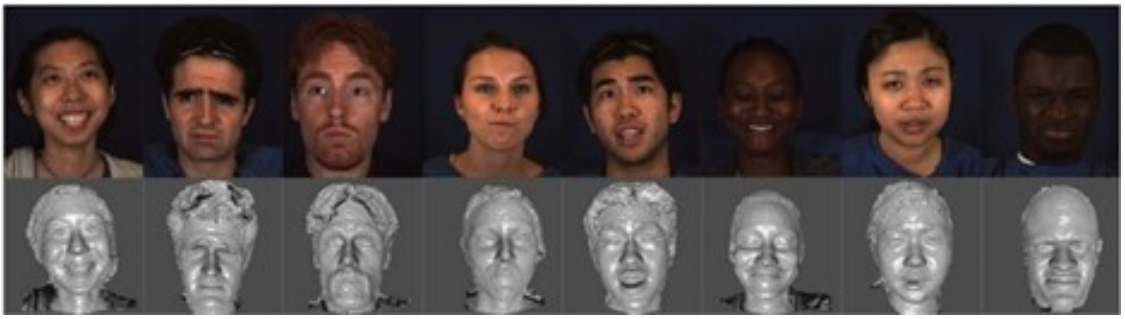


Figure 4: 2D AND 3D EXAMPLES OF EIGHT EMOTIONAL EXPRESSIONS

ing. 3D avatars allow for detailed and expressive facial animations, including lip-syncing, eyebrow movements, and eye expressions, accurately capturing the subtleties of sign language communication.

3. **Interactivity and Engagement:** Users can interact with 3D avatars through gestures, voice commands, or input devices, making the learning and communication experience more engaging.

4. **Customization and Personalization:** 3D avatars offer greater flexibility for customization and personalization, allowing users to tailor the avatar's appearance, gender, age, and cultural characteristics to their preferences, ensuring inclusivity and cultural sensitivity.

5. **Accessibility Features:** 3D avatars can include subtitles, voiceovers, and alternative communication modes, making sign language translation accessible to users with diverse needs.

The use of 3D avatars in sign language translation technology represents a significant advancement in improving communication accessibility and inclusivity for individuals with speech impairments or hearing loss. By leveraging the advantages of 3D animation, we can create more immersive, expressive, and accessible solutions for sign language interpretation in digital environments.

4.1.4 Significance of 3D Integration in the Project

The integration of 3D technology within our project plays a pivotal role, constituting a substantial portion of the overall system. Here's an in-depth exploration of the significance of 3D integration:

1. **Enhanced User Experience:** Using 3D avatars makes the app more immersive and interactive. Users enjoy a visually engaging and dynamic interface, which helps them better understand and engage with sign language communication.

2. **Central Role in Communication Accessibility:** As one of the core services offered within the app, the 3D integration serves as a primary vehicle for facilitating communication accessibility. They provide lifelike representations of sign language gestures and expressions, bridging communication gaps and improving accessibility for people with speech impairments or hearing loss (see Figure 3 for a visualization of the 3D generation system from text input).

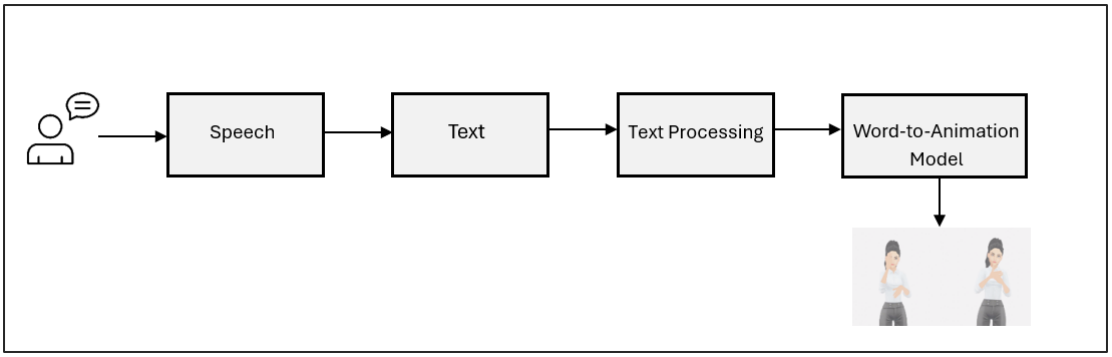


Fig. 5 3D GENERATION SYSTEM FROM TEXT INPUT

3. **Technological Innovation:** Using 3D technology shows our commitment to technological innovation and advancement within the field of sign language translation. It highlights our dedication to exploring new solutions and improving traditional methods.

4. **Comprehensive Learning Environment:** The 3D feature creates a thorough

learning environment for sign language learners. Users can practice and refine their skills interactively with realistic 3D avatars.

5. **Strategic Differentiation:** The integration of 3D avatars sets our project apart from other sign language translation solutions by offering a unique and innovative approach to communication accessibility.

By integrating a 3D avatar, our project achieves a more accurate, engaging, and user-friendly approach to sign language translation. This innovative approach has the potential to not only benefit our project's goals but also contribute to advancements in sign language communication accessibility in the future.

4.2 Technology Stack

In the realm of 3D avatar design and animation for sign language translation, a variety of tools and libraries are available to support the creation of lifelike and expressive avatars. Here's an overview of the technology stack specific to 3D avatar design and animation:

1. **3D Modeling and Animation Software:**

    - **Blender:** Blender stands out as a versatile and powerful open-source tool for creating 3D animations and models. It offers a comprehensive suite of features for modeling, rigging, animation, and rendering, making it an ideal choice for developing lifelike 3D avatars.

Fig. 6: BLENDER

2. **Game Development Platform:**

   - **Unity Engine:** Unity offers robust tools and workflows for creating interactive 3D experiences, including real-time rendering, physics simulation, and asset management. It can be used in conjunction with Blender for developing immersive and interactive environments featuring 3D avatars.

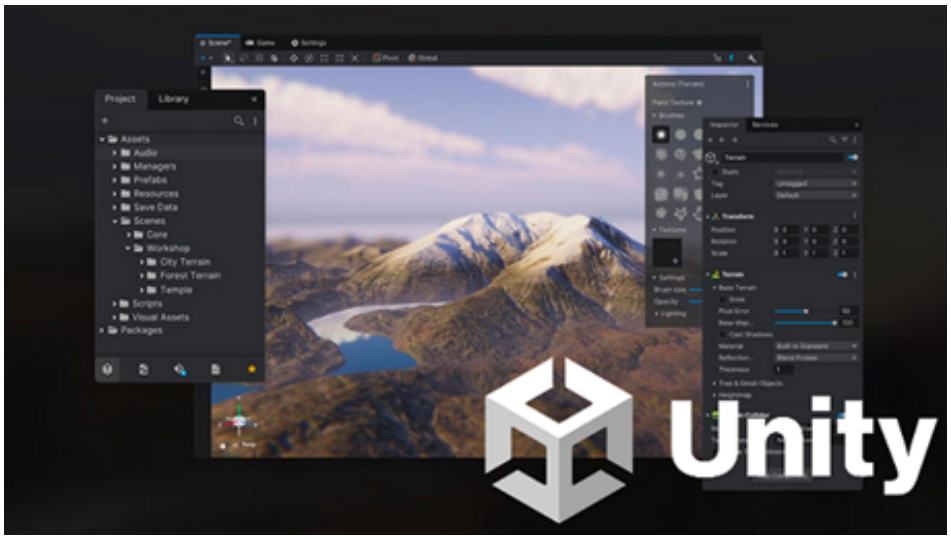


Fig. 7: UNITY

3. **Motion Capture and Animation Synthesis:**

   - **DeepMotion:** DeepMotion specializes in AI-driven animation synthesis, providing solutions for motion capture, animation synthesis, and character simulation. Its technology enhances the realism and fluidity of 3D avatar animations through deep learning techniques.

Fig. 8: DEEPMOTION

4. **Character Animation Libraries:**

- **Mixamo:** Mixamo is an online platform with a large library of pre-animated 3D characters and motions. It offers customizable character models and animations, adding variety and realism to 3D avatar animations. Mixamo easily integrates with Blender, simplifying the process of importing and using pre-made animations for sign language translation.

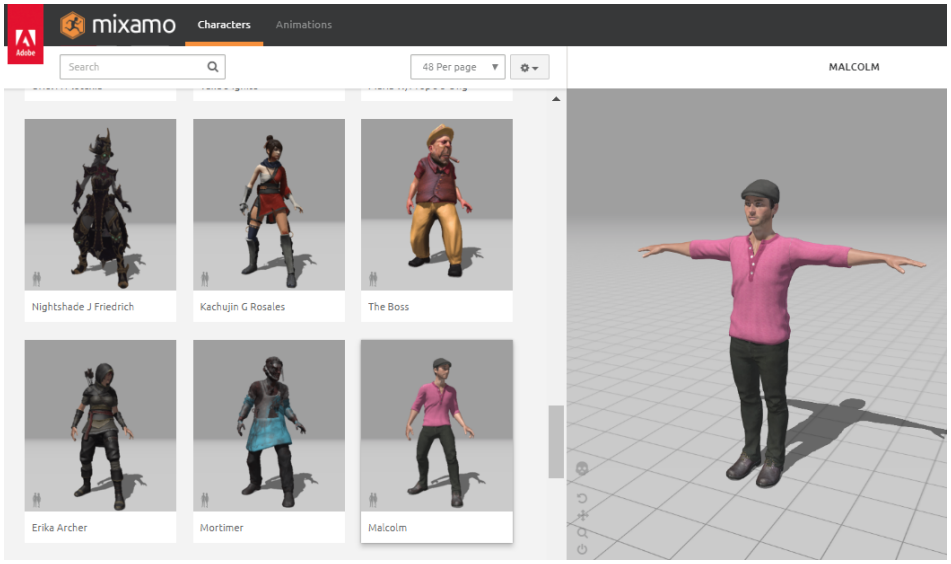


Fig. 9: MIXAMO

5. **Rendering Engines:**

    - **Cycles:** Blender's built-in rendering engine, known for its physically-based rendering capabilities and high-quality output, is utilized for rendering life-like 3D avatars with realistic lighting and materials.

6. **Python Scripting:**

    - **Blender Python API:** Blender's comprehensive Python API allows developers to extend and customize its functionality through scripting, enhancing the efficiency and versatility of Blender for 3D avatar design and animation.

7. **Blender Toolkit:**

    - **Blender Toolkit:** A collection of add-ons, scripts, and resources that extend Blender's functionality. It helps with tasks like character rigging, animation retargeting, and motion capture integration, enhancing Blender's capabilities for 3D avatar design and animation.

Blender: The Backbone of 3D Avatar Design and Animation

When it comes to 3D avatar design and animation for sign language translation within our project, Blender emerges as the cornerstone of our technology stack. Here's a detailed breakdown of the tools and resources within Blender that we utilize:

- Blender serves as the primary software tool for creating, rigging, animating, and rendering 3D avatars. Its robust feature set and open-source nature make it an ideal choice for our project's needs.

- When you run 3D Blender, you will see an interface like the screen below (Fig. 10). The large window in the center is the "3D Viewport" where most editing takes place.

- 1. **Camera**: This allows you to view your scene from different angles.

- 2. **Object**: This represents any 3D element in your scene, such as avatars, props, or backgrounds.

- 3. **Light**: Lights illuminate your scene and create shadows, affecting the overall look.

- 4. **Outliner**: This panel displays a list of all objects in your scene, including cameras and lights, for easy selection and management.

- 5. **Properties**: This panel shows the properties of the currently selected object (or the scene itself), allowing you to modify its characteristics.

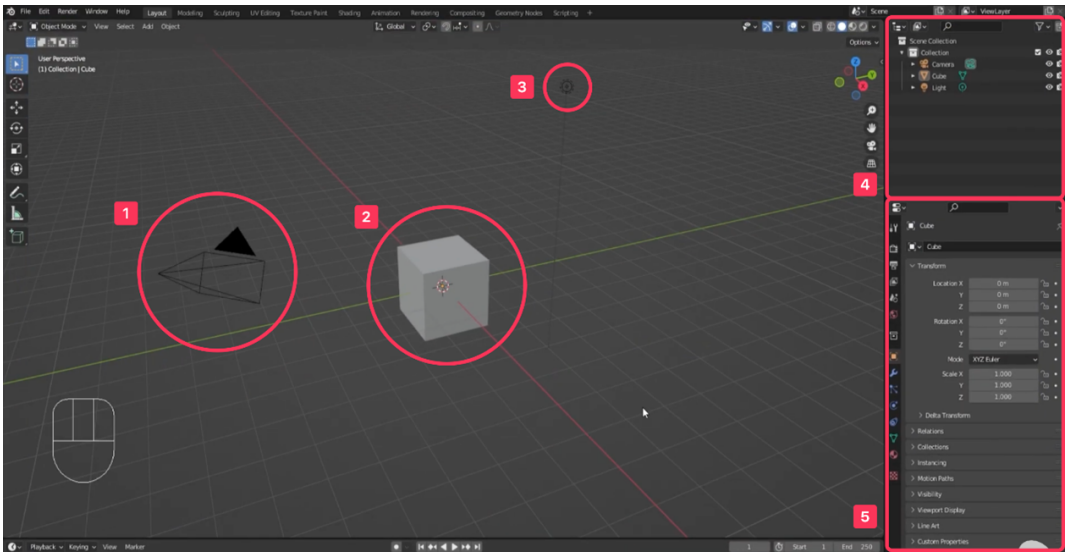


Fig. 10: BLENDER INTERFACE

Key features of Blender include:

- **Modeling**: Blender offers powerful modeling tools for creating the basic structure of avatars, including mesh editing, sculpting, and texturing.

- **Rigging**: With Blender, we can rig our avatars by adding bones and defining their relationships, allowing for realistic movement and animation.

- **Animation**: Blender's animation tools enable us to animate our rigged avatars, defining keyframes and creating lifelike movements that accurately convey sign language gestures.

- **Rendering**: Blender's Cycles rendering engine allows us to render high-quality images and animations of our avatars, with realistic lighting, materials, and effects.

We will discuss them in detail in the coming sections.

4.3 Avatar Design Process

This section dives into the steps involved in creating the 3D sign language avatar for our project. The design process can be broken down into several key stages:

4.3.1 Modelling: Building the Basic Structure

Creating the foundational structure of our avatars involves several key steps:

1. **Reference Gathering**: We gather anatomical references to ensure accuracy in our designs.

2. **Blocking Out**: Using simple shapes, like spheres and cylinders, we establish the avatar's overall form.

3. **Proportional Editing**: Blender's tools help us refine proportions while maintaining overall shape.

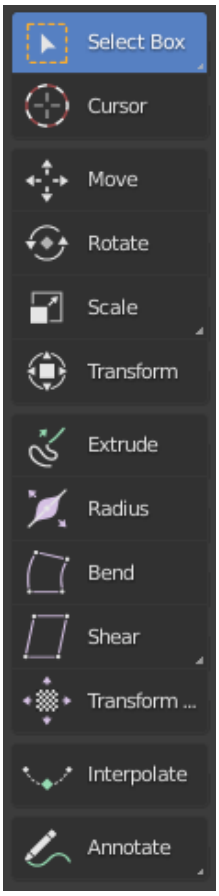


Fig. 11: BLENDER TOOLS

4. **Detail Sculpting**: We add finer details such as facial features and clothing wrinkles.

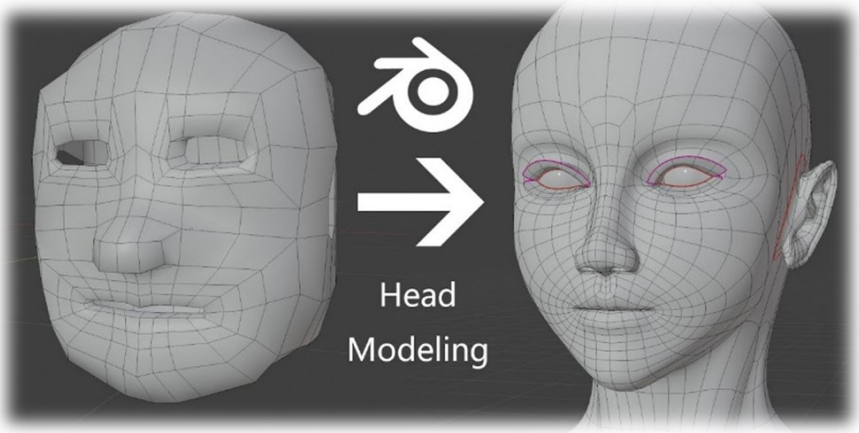


Fig. 12: DETAIL SCULPTING IN MODELING

5. **Iterative Refinement**: Continuous adjustments and feedback refine the model until it meets our standards.

6. **Finalization**: Once satisfied, we finalize the model for further development.

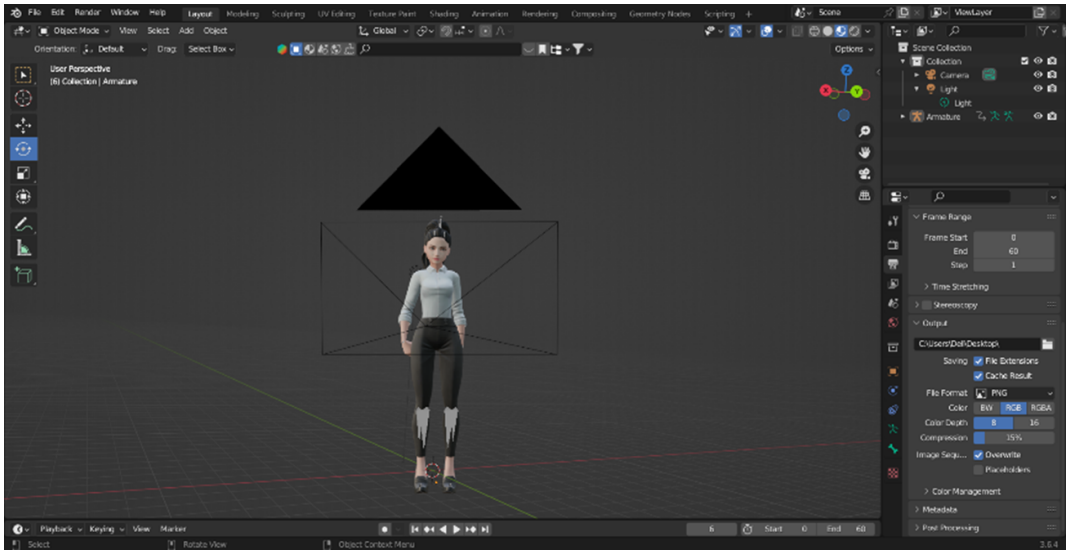


Fig. 13: FINAL MODEL

4.3.2 Rigging: Adding the Skeletal Framework

Rigging in Blender involves creating an "armature," a virtual skeleton made of connected bones. These bones act as control points, allowing you to bend and twist your character's mesh (3D model) during animation.

1. **Manual Rigging**: This method offers full control by building the armature bone by bone, positioning and parenting them strategically to the character's mesh.

2. **Riggify**: Blender's built-in automatic rigging tool, Riggify, streamlines the process for humanoid characters. It analyzes your mesh and generates a basic rig with pre-defined bones for limbs, spine, and head. You can then fine-tune and add more bones for complex features like fingers or facial expressions.

   Essential Steps in Rigging:

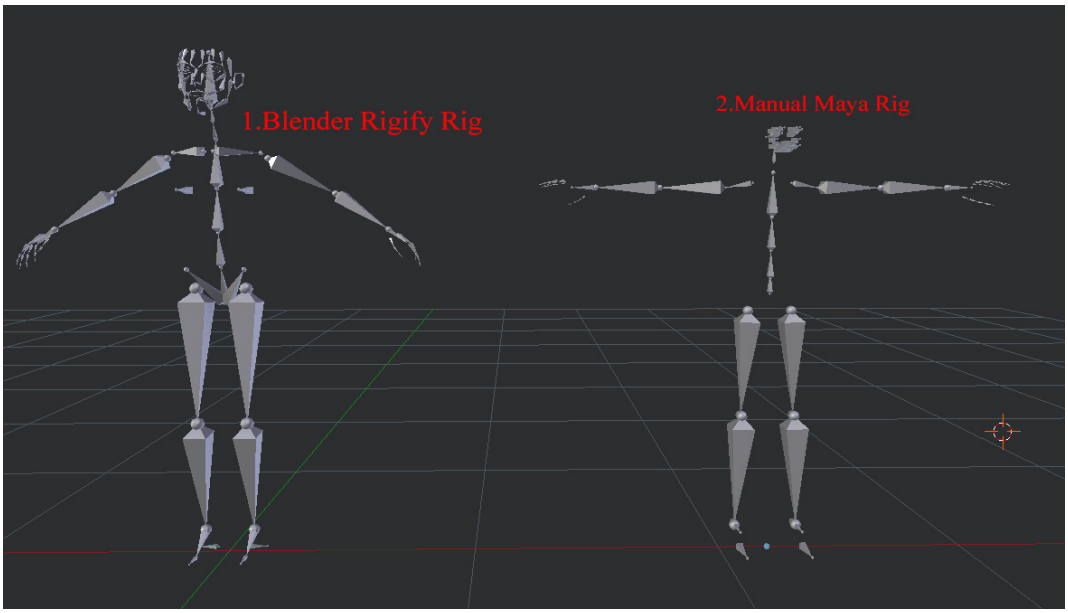


Figure 14: MANUAL RIGGING VS. RIGGIFY

- **Armature Creation**: We build the armature for the character, either manually or through Riggify.
- **Weight Painting**: It's utilized to assign vertex groups to bones, determining how much influence each bone has over the surrounding mesh. This step is crucial for smooth and natural-looking deformations during animation.
- **Parenting**: Connecting the character's mesh to the armature's bones using parenting techniques. This allows the mesh to deform and move along with the bone movements.
- **IK (Inverse Kinematics)**: This system helps control limbs and other complex joints more intuitively by setting target points for the hand or foot, and the rig automatically adjusts the bones to reach those targets.

Fig. 15: RIGGED MODEL

4.3.3 Animation: Bringing Life to Avatars

Animating avatars in Blender is a dynamic process that imbues them with personality and expression. Here's a breakdown of the key concepts: **Core Principles:**

- **Keyframes**: These capture the state (position, rotation, scale) of an object at a specific point in time. By setting keyframes at different points, you define the animation path.
- **Timeline**: This visualizes the animation sequence, showing the keyframes you've set. You can adjust timing, playback speed, and add effects here.
- **Pose Mode and Rigging**: Blender's Pose Mode allows us to manipulate the avatar's rig directly, posing the bones to create dynamic movements.

Fig. 16: POSE MODE

Animation Techniques:

1. **Object Animation**: Animate object movement, rotation, and scale by setting keyframes for their transform properties.

2. **Shape Keys**: Define different shapes (keys) for the mesh and animate the transition between these shapes.

3. **Non-Linear Animation (NLA)**: Manage and combine different animation clips independently on the timeline for modularity and reusability.

Once the animation is complete, we render the final frames using Blender's Cycles. We adjust rendering settings to achieve the desired visual quality and output format, ready for integration into our sign language translation system or other applications.

4.4 Rendering & Time

Rendering is the process of transforming your 3D scene in Blender into a final image or animation sequence. It's like taking a picture of your virtual world, capturing all the lighting, materials, and textures you've meticulously crafted. However, rendering can

also be time-consuming. There are many factors that affect how long a render takes in Blender, here are some key ones:

1. **Render Settings**: We begin by configuring Blender's render settings to match our project's requirements. This includes setting the resolution, frame rate, and choosing the render engine (Cycles or Eevee) based on the desired quality and performance.

2. **Lighting and Shading**: Proper lighting and shading are essential for creating realistic and visually appealing renders. We set up lights to enhance the visibility and aesthetics of the avatars and their movements.

3. **Camera Setup**: The camera setup determines the viewpoint and framing of the rendered scenes. We position and animate the camera to capture the best angles and ensure that the sign language gestures are clearly visible.

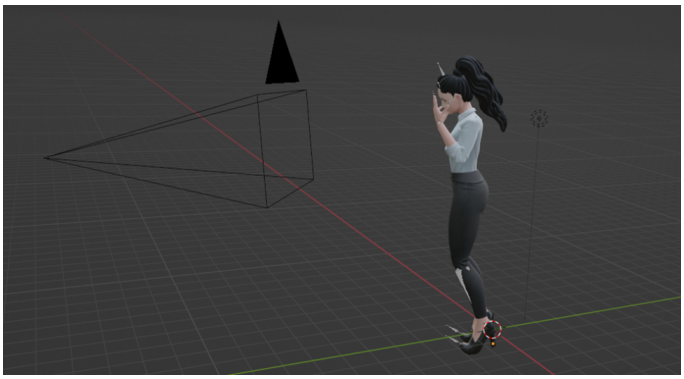


Fig. 17: CAMERA PLACEMENT

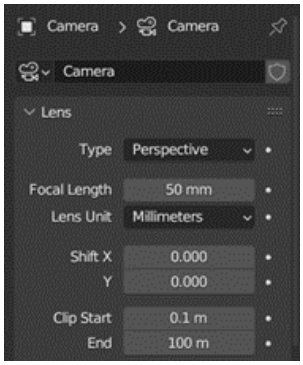


Fig. 18: CAMERA SETTINGS

4. **Rendering Process**: Once all settings are configured, we initiate the rendering process. Blender converts each frame of the animation into a 2D image. We're working on 60 frames, with Frame start: 0, End:60, and Step:1.

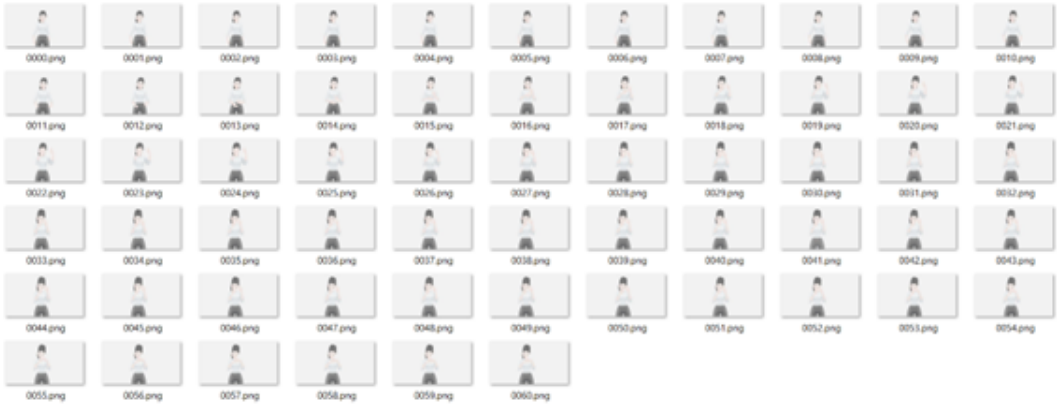


Fig. 19: FRAMES RENDERING

5. **Optimizing Render Times**: To optimize render times and maintain quality, we consider several factors:

   - Simplifying Scenes: Reducing the complexity of scenes, such as the number of objects and their detail levels.
   - Adjusting Sample Rates: Lowering the number of samples for quicker renders, especially when using Cycles.
   - Utilizing GPU Rendering: Leveraging GPU rendering capabilities for faster processing times, if supported by the hardware.
   - Resolution Management: Choosing an appropriate resolution that balances quality and rendering speed.

6. **Output Formats**: Blender supports various output formats for rendered animations, including MPEG-1, AVI, MPEG-2, and H.264 (MP4). We rendered the videos in H.264 format for a good balance between quality and file size.

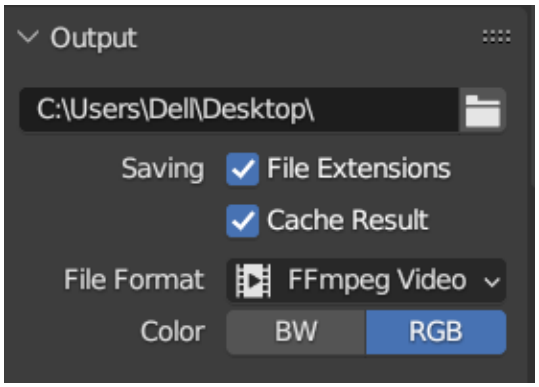


Fig. 20: OUTPUT SETTINGS

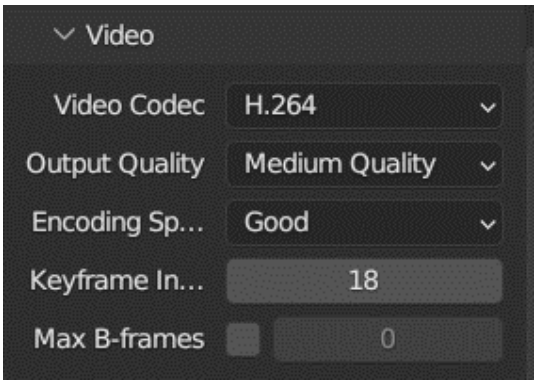


Fig. 21: VIDEO SETTINGS

7. **Time Management**: Rendering can be time-intensive, so managing render times effectively is crucial:

   - Short Animations: Rendering a single word, like "sad," takes approximately 70 seconds.
   - Longer Animations: Rendering a 70-frame video, such as "I love you," takes around 120 seconds.
   - Factors Influencing Render Time:
     - Complexity of Animation: Detailed animations with intricate movements require more time.
     - Number of Frames: Higher frame counts increase render times.
     - Hardware Resources: More powerful CPUs and GPUs reduce rendering durations.
     - Render Settings: Higher resolutions and quality settings extend render times.

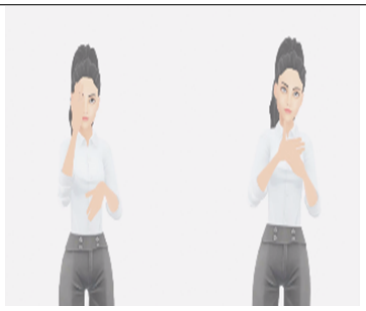
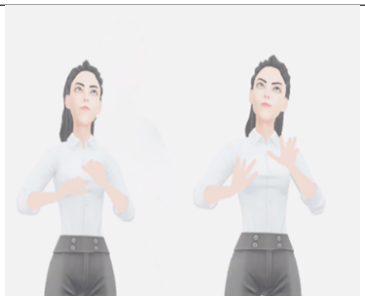
By optimizing these factors, we efficiently manage rendering times while maintaining high-quality output. The final rendered animations are then ready for integration into our sign language translation system, enhancing the overall user experience.

Figure 22: BLENDER RENDER WINDOW

4.5 Illustrative Examples

This section shows some examples of the rendered animations from Blender. The following table presents a sample of English words and their corresponding sign language translations:

Table 4.1: English word-to-sign animation

English Word	Sign Movement	English Word	Sign Movement
Stop		Finish	
Play		Calm Down	
Love you		Happy	
Angry		Sick	
Baby		Dad	
School		Car	

Chapter 5

System Design and Deployment using Mobile Application

5.1 System Design

In this chapter, we provide branding overview considering logo design and full branding. As Sign Language Application brand, it is important to have a strong and recognizable design that reflects the professionalism and expertise of your brand. A well-designed brand can help build trust and credibility with clients .

5.1.1 Speakd Brand Guidelines

Brand Identity: Our brand identity is modern, professional, and trustworthy.
logo: The logo should feature a simple, recognizable icon that incorporates American Sign Language Icon representing two hands. The logo should be displayed in our primary brand colors: blue and white. The logo should be used consistently across all touchpoints and should not be altered or distorted in any way. Color Palette: Our primary brand colors are blue and white, which represent trust, calmness, and comfort. Typography: The main font for our brand identity is a clean and modern.
The font can be used for accent text, but should be used sparingly and in moderation. The font size and style should be consistent across all touchpoints and

should be legible and easy to read. Imagery: Imagery used in our brand identity should be high-quality, professional, and relevant to our sign language services. The imagery should be consistent with our brand colors and overall aesthetic. The imagery should be used consistently across all touchpoints and should not be altered or distorted in any way

5.1.2 What Should App Represent?

The Speakd app should represent the values and personality of the Speakd brand, which is caring, knowledgeable, and trustworthy. The app should be designed to enable people communicate with deaf people while also reflecting the high level of professionalism and expertise associated with the Speakd brand. Here are some specific elements
that the Speakd app should represent: Accessibility: The Speakd app should be designed to be easily accessible to all users, regardless of their ability level or technological proficiency.
Trustworthiness: The Speakd app should convey a sense of trustworthiness and reliability to users. This can be achieved through features like secure user authentication, clear and concise Sign Language word, and professional design elements. By representing these core values and attributes, the Speakd app can become a trusted and reliable resource for users to communicate with each athor.

5.1.3 Speakd UX Case Study

Overview

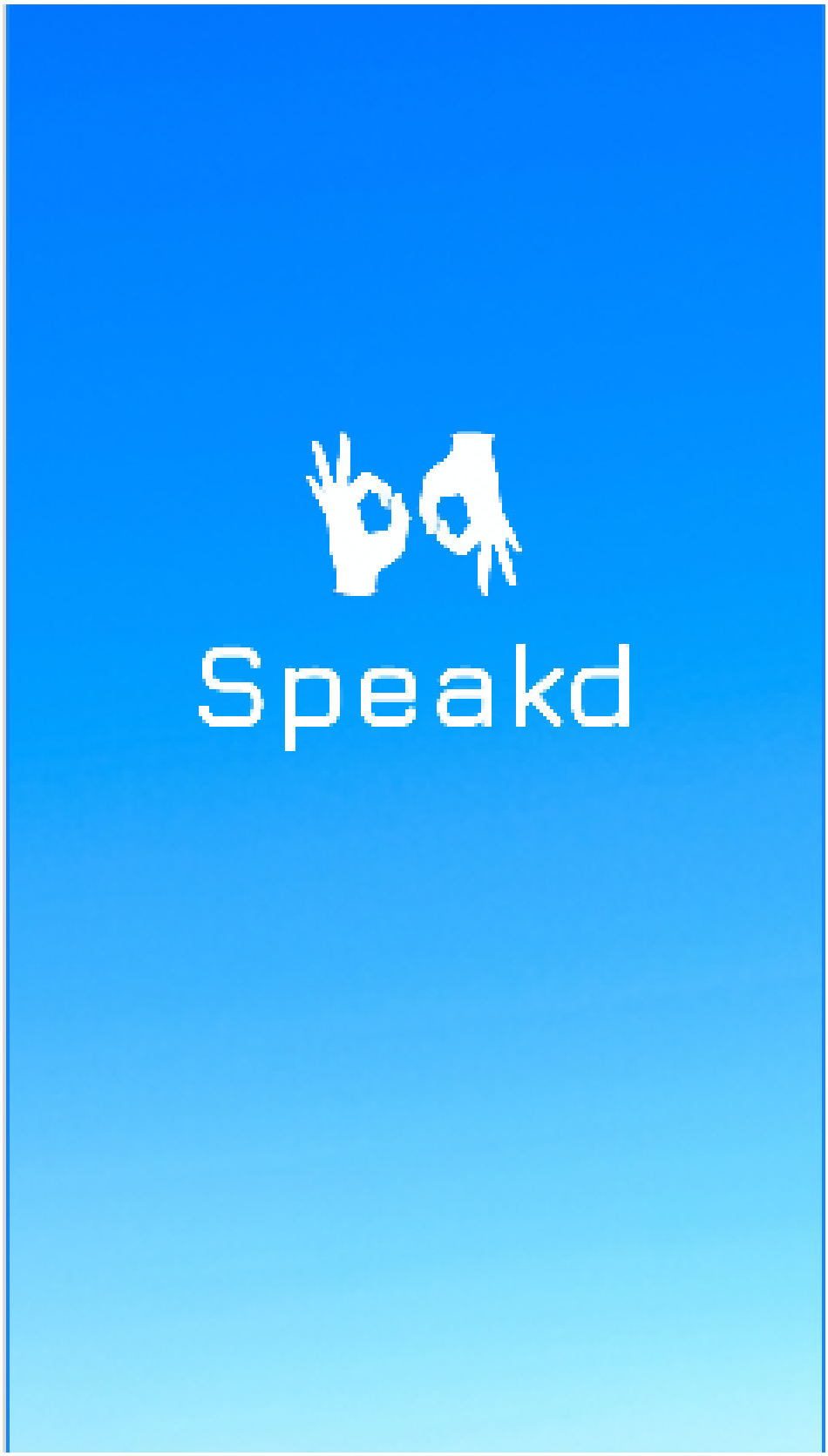
A mobile application poised to transform communication for the deaf
and hard of hearing community. In a world where inclusivity and accessibility are paramount, Speakd bridges the gap between sign language users and the broader society by offering seamless translation capabilities right at your fingertips.

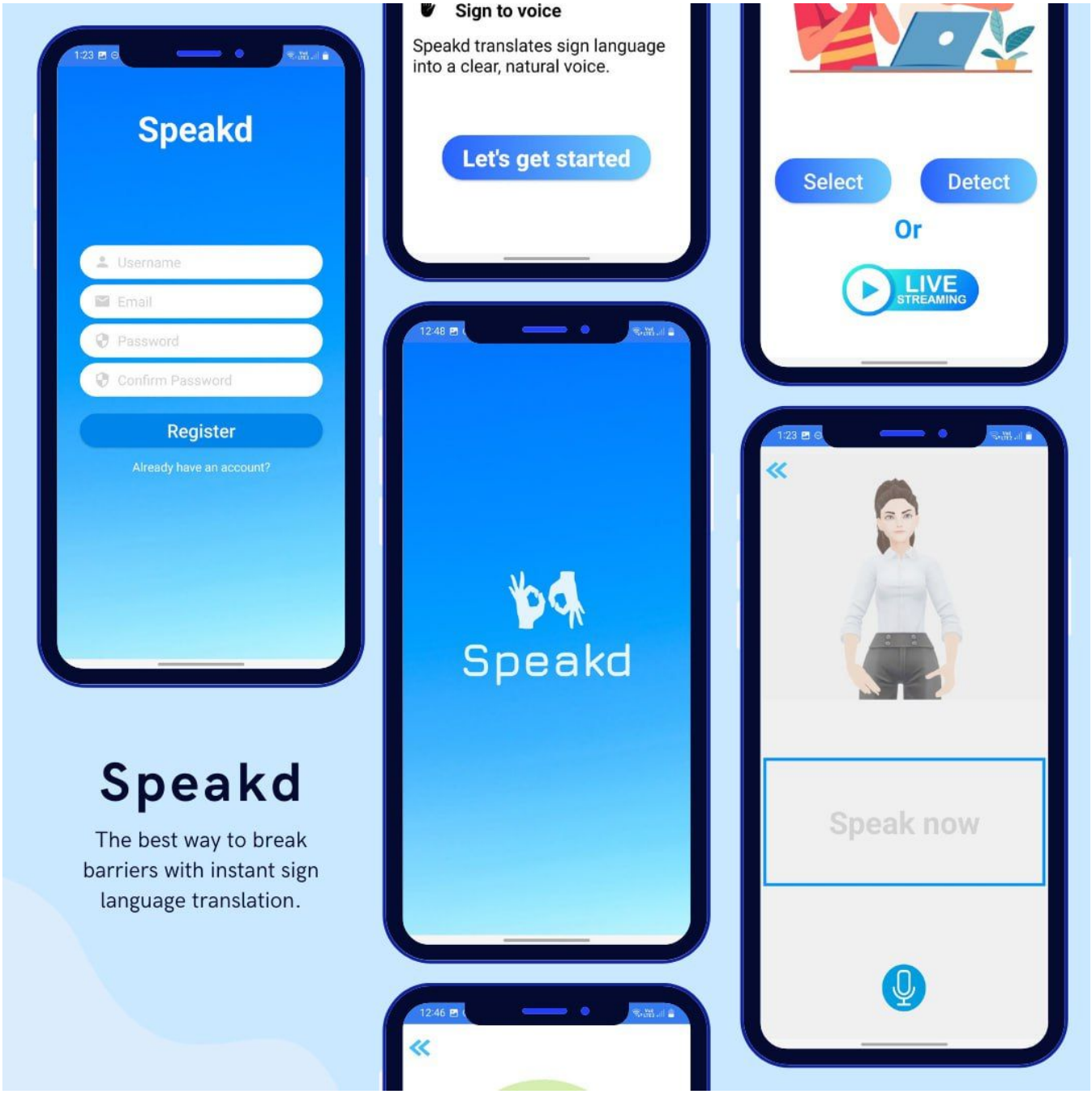
User Flow

1-User opens the app and is prompted to sign in or create an account.

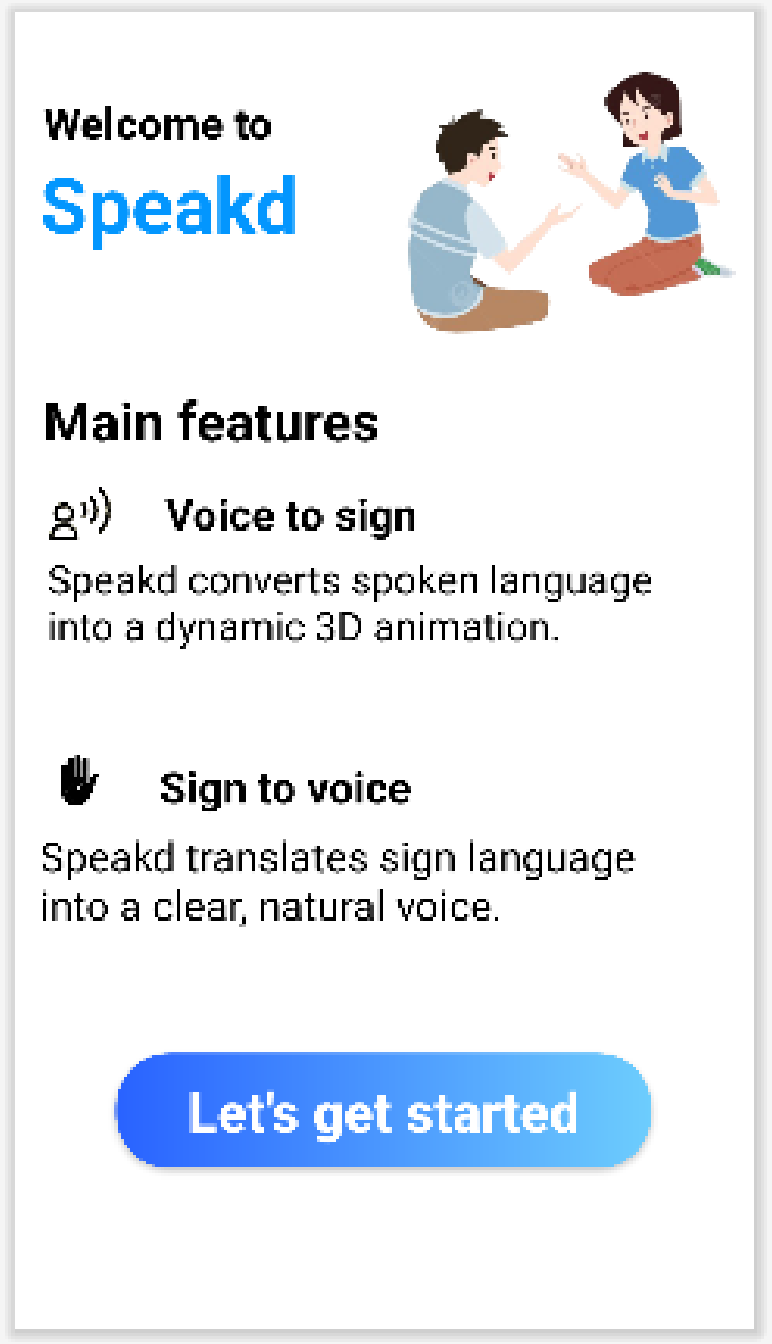
2-Non deaf users can use the "sign to voice" button to understand what deaf people want to say.

3- Deaf users can use the "voice to sign" button to understand what people say.

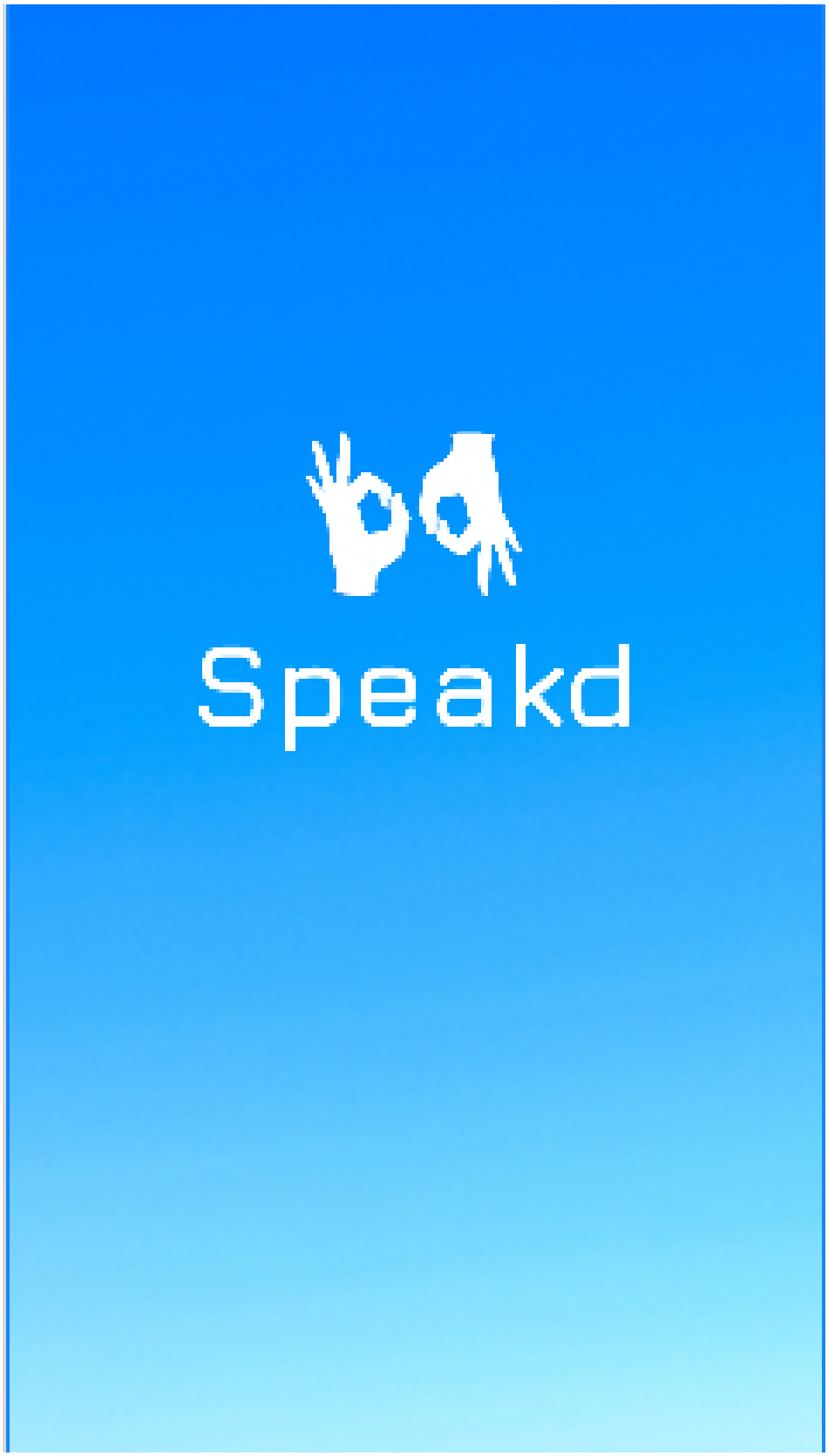

Speakd

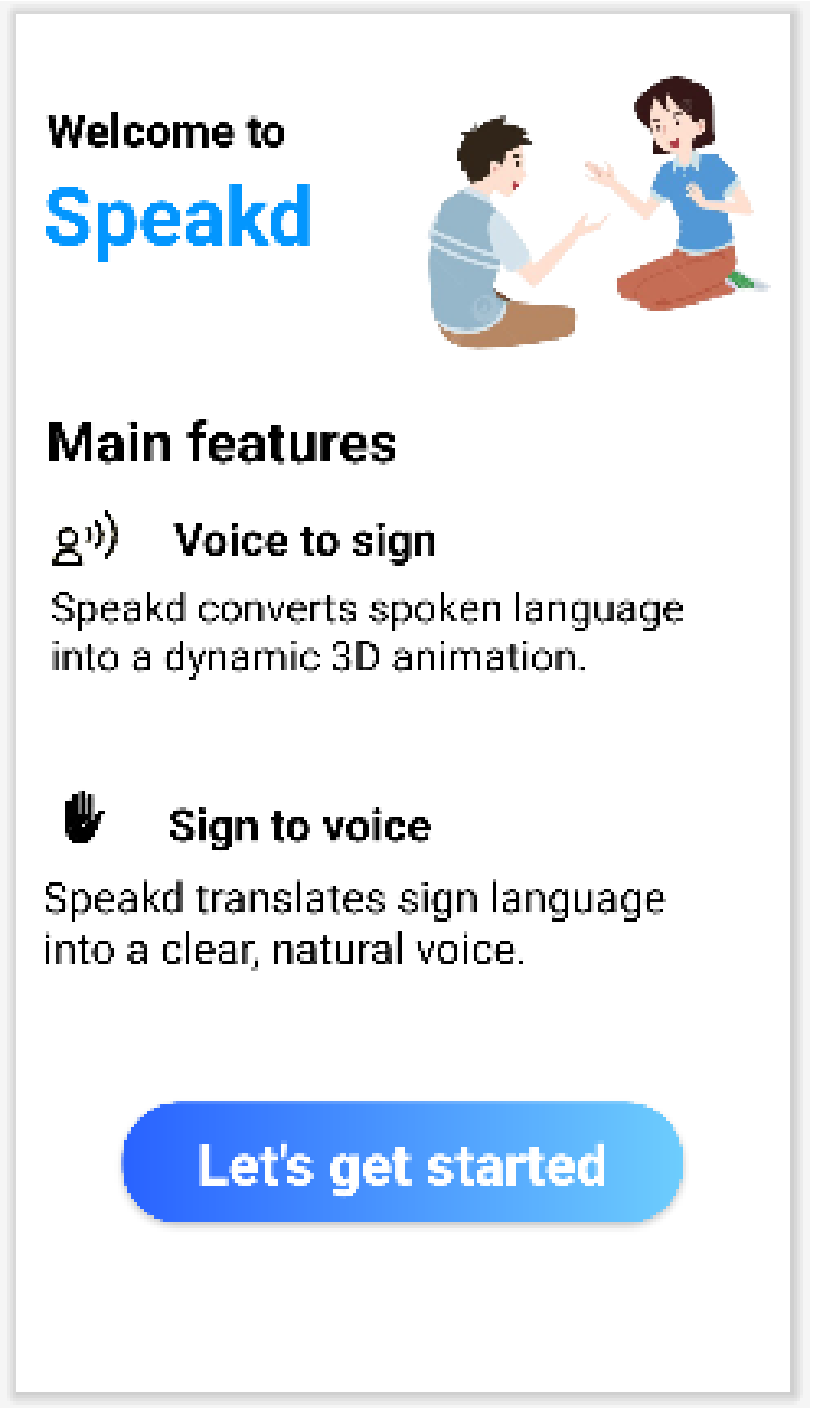

Speakd
Username
Email
Password
Confirm Password
Register
Already have an account?
Speakd
The best way to break barriers with instant sign language translation.
Sign to voice
Speakd translates sign language into a clear, natural voice.
Let's get started
Speakd
Select
Detect
Or
LIVE STREAMING
Speak now

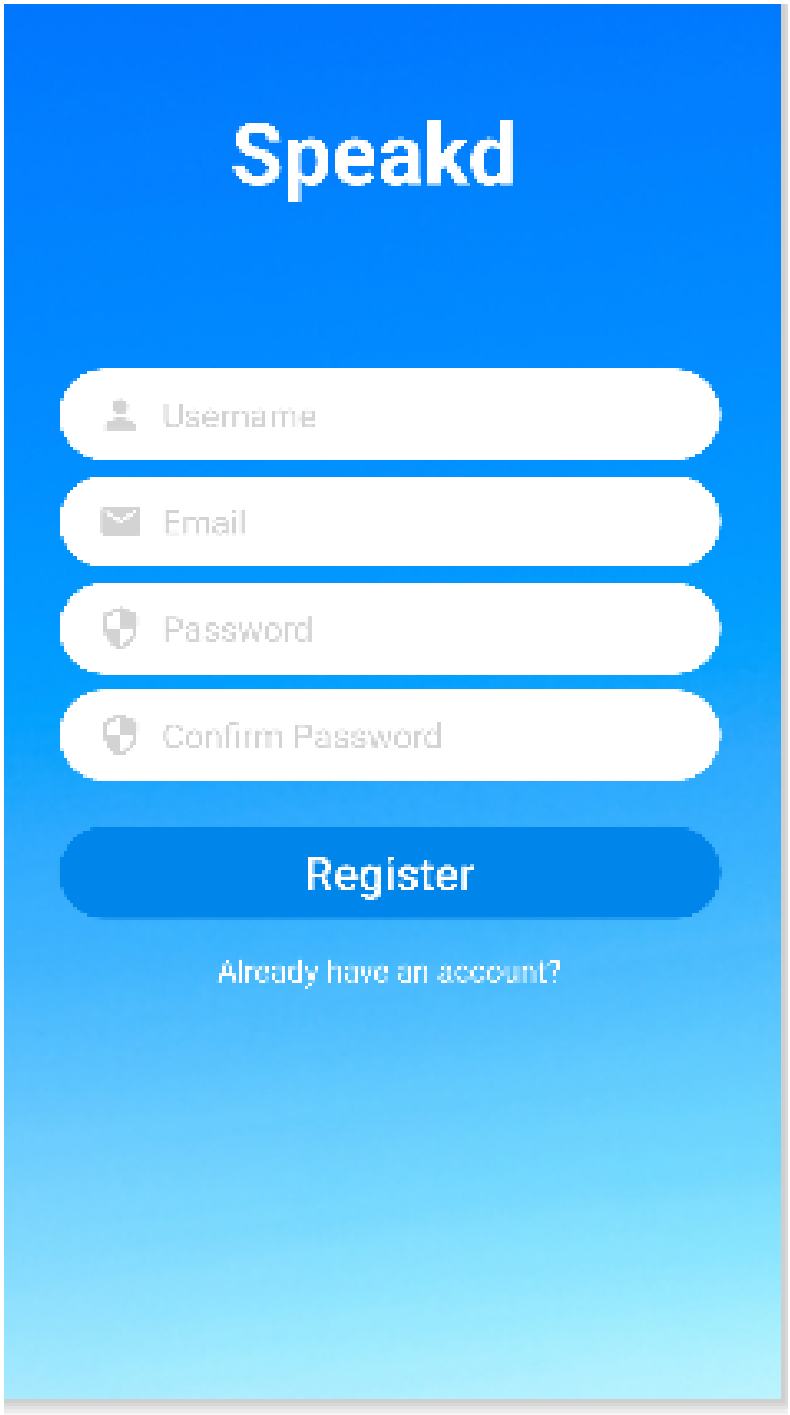
5.1.4 Interface Design

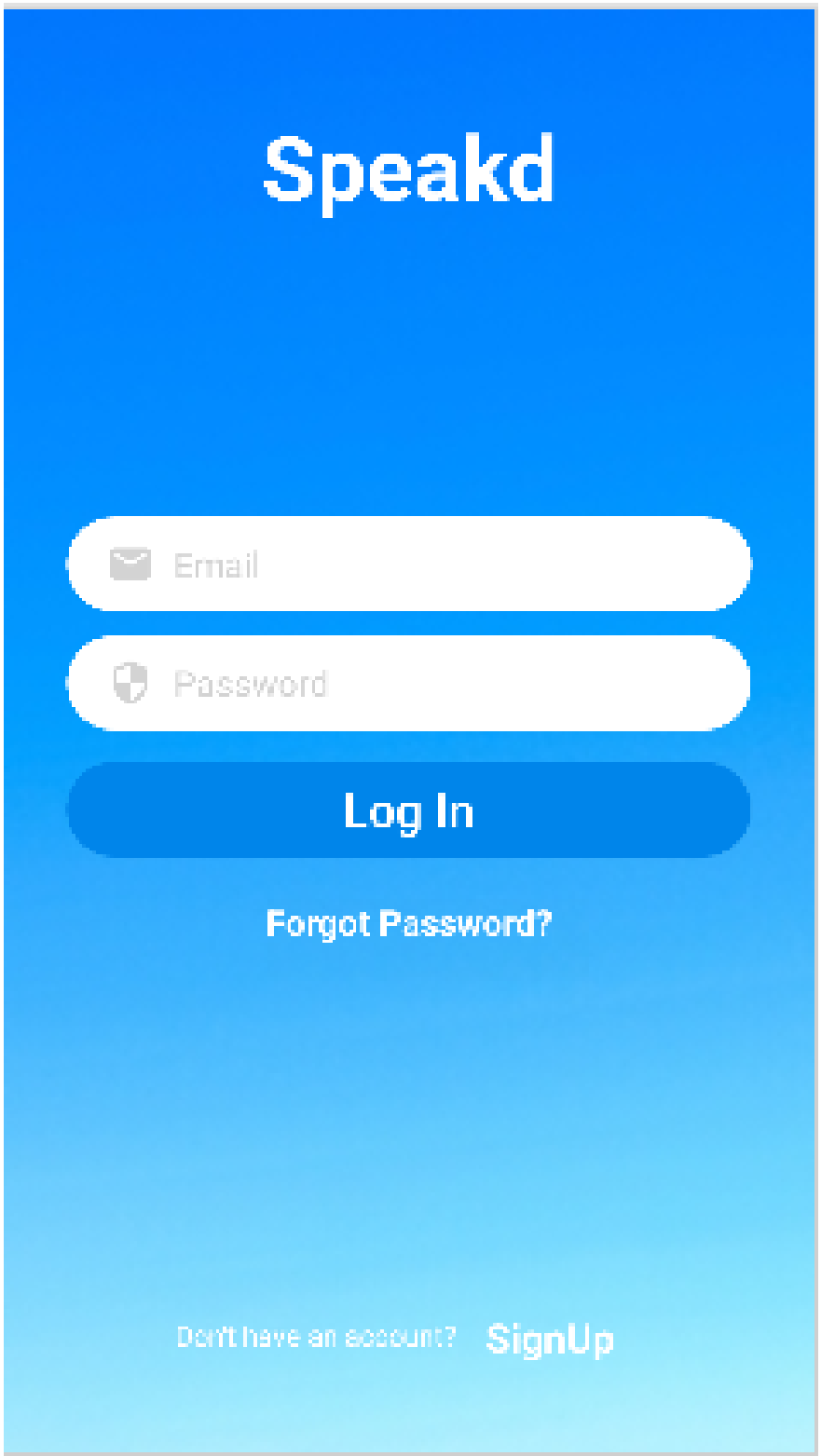


At the beginning the user can see an introduction about the application and how it works

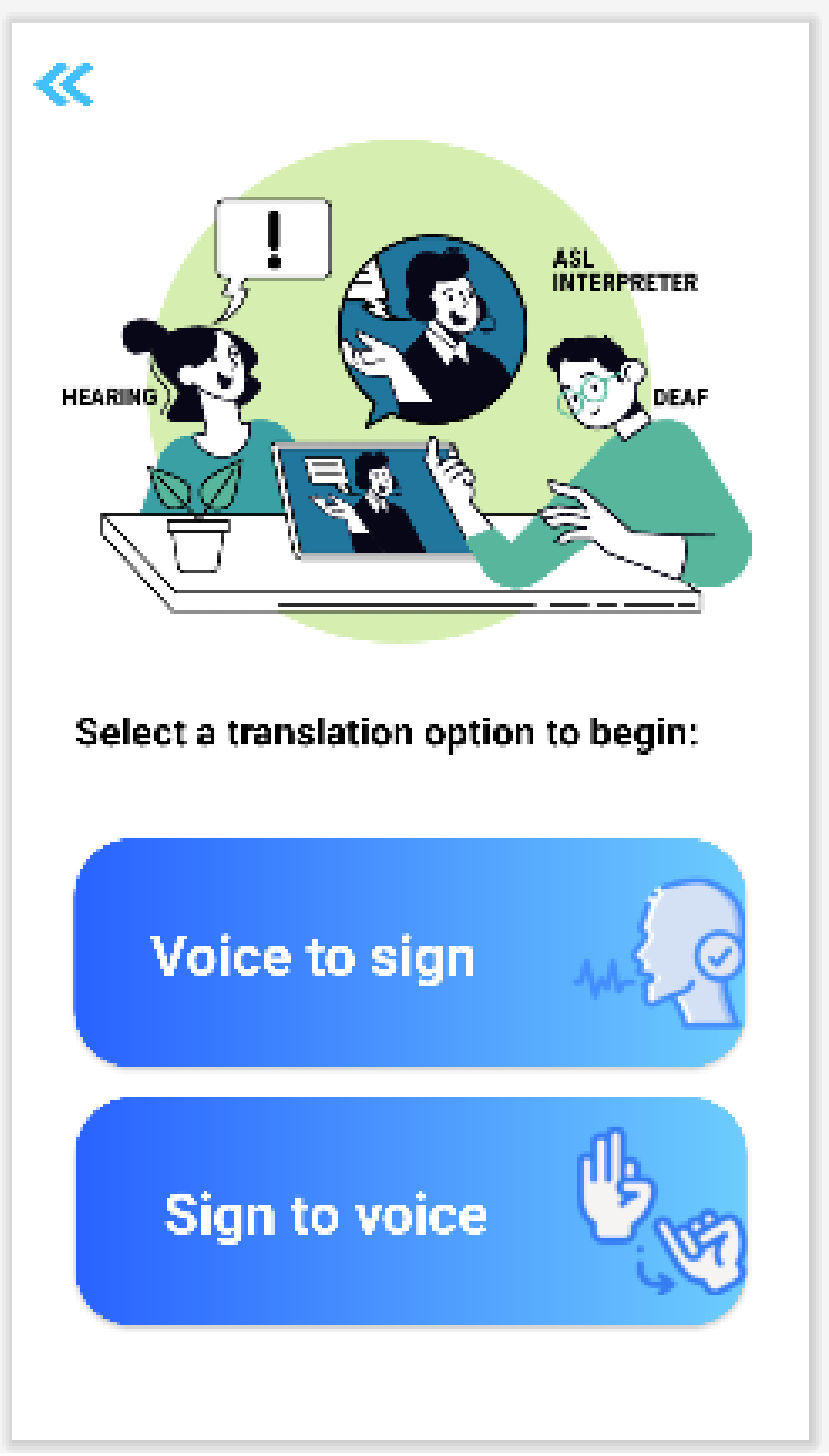

Speakd


Welcome to
Speakd
Main features
Voice to sign
Speakd converts spoken language into a dynamic 3D animation.
Sign to voice
Speakd translates sign language into a clear, natural voice.
Let's get started

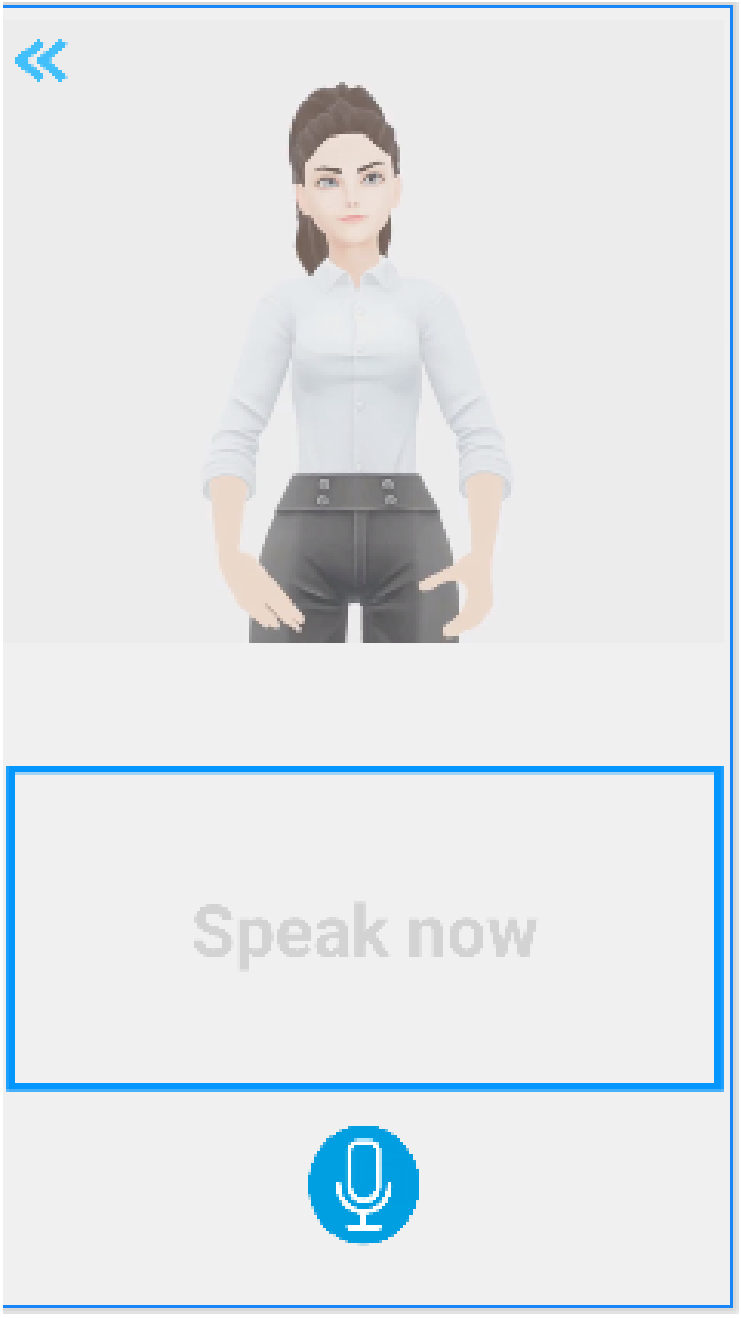

Speakd
Username
Email
Password
Confirm Password
Register
Already have an account?


Speakd
Email
Password
Log In
Forgot Password?
Don't have an account? SignUp

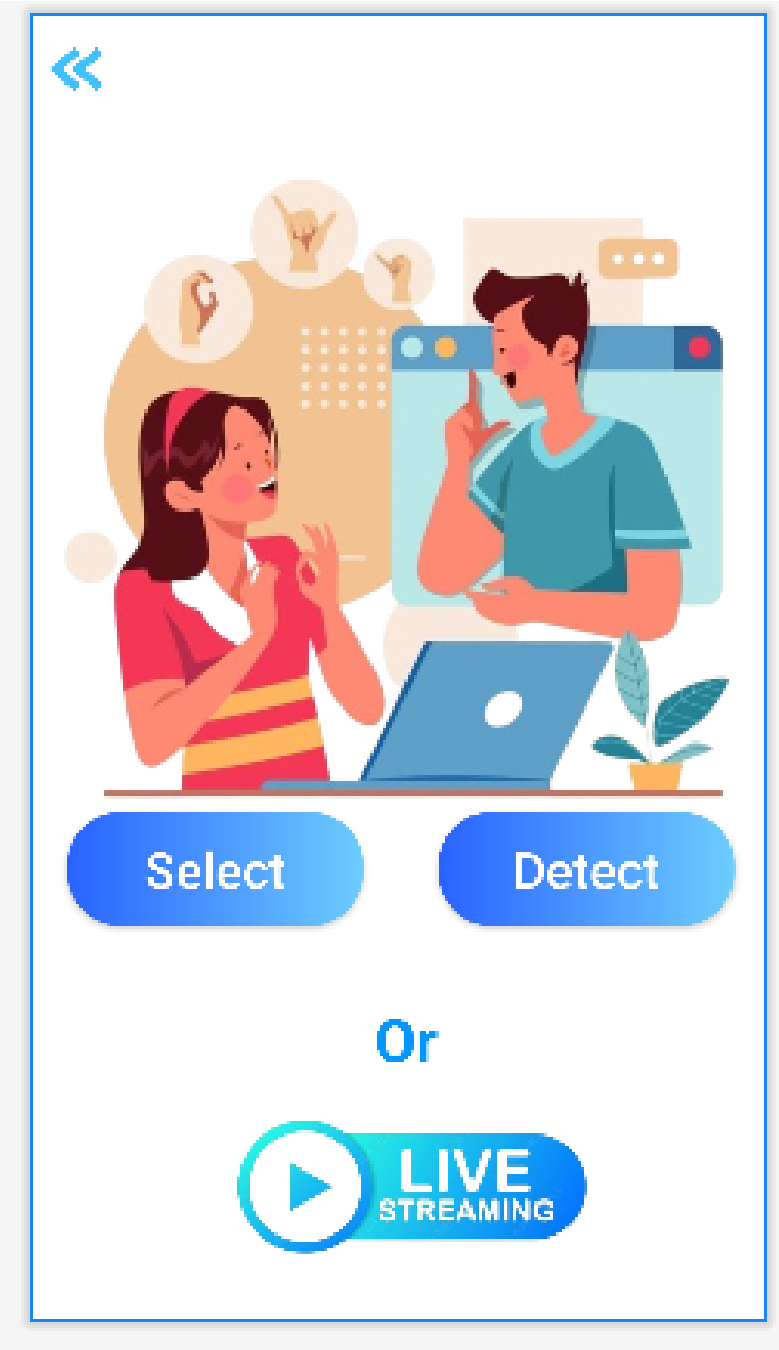
Then, the welcome page appears, allowing the user can sign in easily with his email and password, just in case the user had signed up before. Else, the user is up to move to the registration page so the user is able to sign up as available. The user can create an account on the application. In case the user forgets the password, he is able to create a new one by verifying his email.


ASL INTERPRETER
HEARING
DEAF
Select a translation option to begin:
Voice to sign
Sign to voice

After logging in, the application shows a page containing two buttons. It's a gateway to understanding and connection. With its innovative technology, Speakd empowers users by: 1-Sign to Voice: Speakd leverages cutting-edge algorithms to translate sign language gestures into text and clear, natural voice representations. 2-Voice to Text and 3D Avatar Conversion: Conversely, Speakd also facilitates communication for sign language users by converting spoken language into text and dynamic 3D avatars that accurately portray the corresponding signs.



When user taps on microphone icon, he can speak and his voice convert into text appearing in a box.Additionally, an Avatar character will perform the animations that express the speech.



This Activity contains 3 buttons when user taps the select button he can choose a sign language photo from his phone or take a picture using the camera then when he press detect sign language will convert into text, and user can hear the voice.The live streaming button enables user to detect sign language in real time.

5.2 Mobile Application Implementation

5.2.1 what is android studio?

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on Jet Brains' IntelliJ IDEA software and designed specifically for Android development It is available for download on Windows, mac OS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development. Android Studio is licensed under the Apache license but it ships with some SDK updates that are under a non-free license, making it not open source. Android Studio was announced on May 16, 2013, at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014.The first stable build was released in December 2014, starting from version 1.0.At the end of 2015, Google dropped support for Eclipse ADT, making Android Studio the only officially supported IDE for Android development. On May 7, 2019, Kotlin replaced Java as Google's preferred language for Android app development. Java is still supported, as is C++.

Features

1-Gradle-based build support

2-Android-specific refactoring and quick fixes

3-Lint tools to catch performance, usability, version compatibility and other problems

4- Pro Guard integration and app-signing capabilities

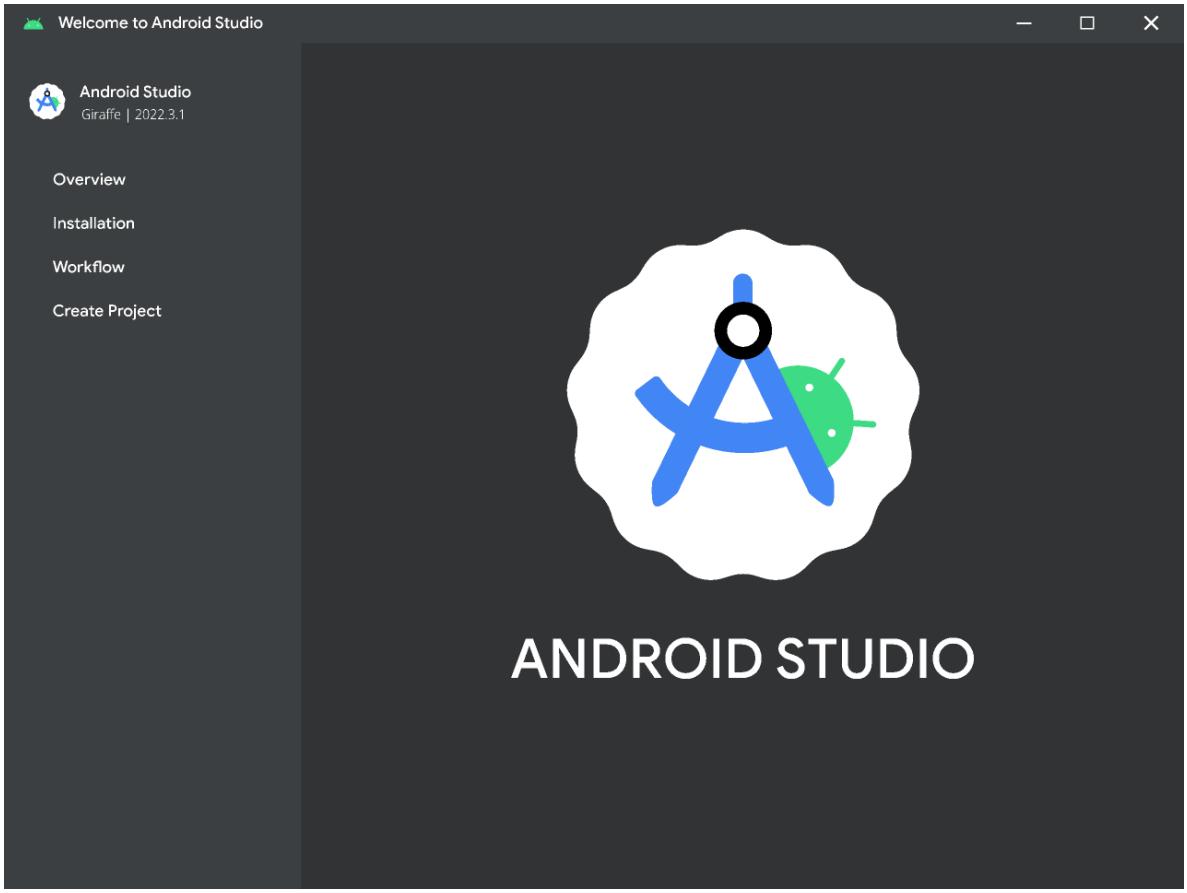
5- Template-based wizards to create common Android designs and components

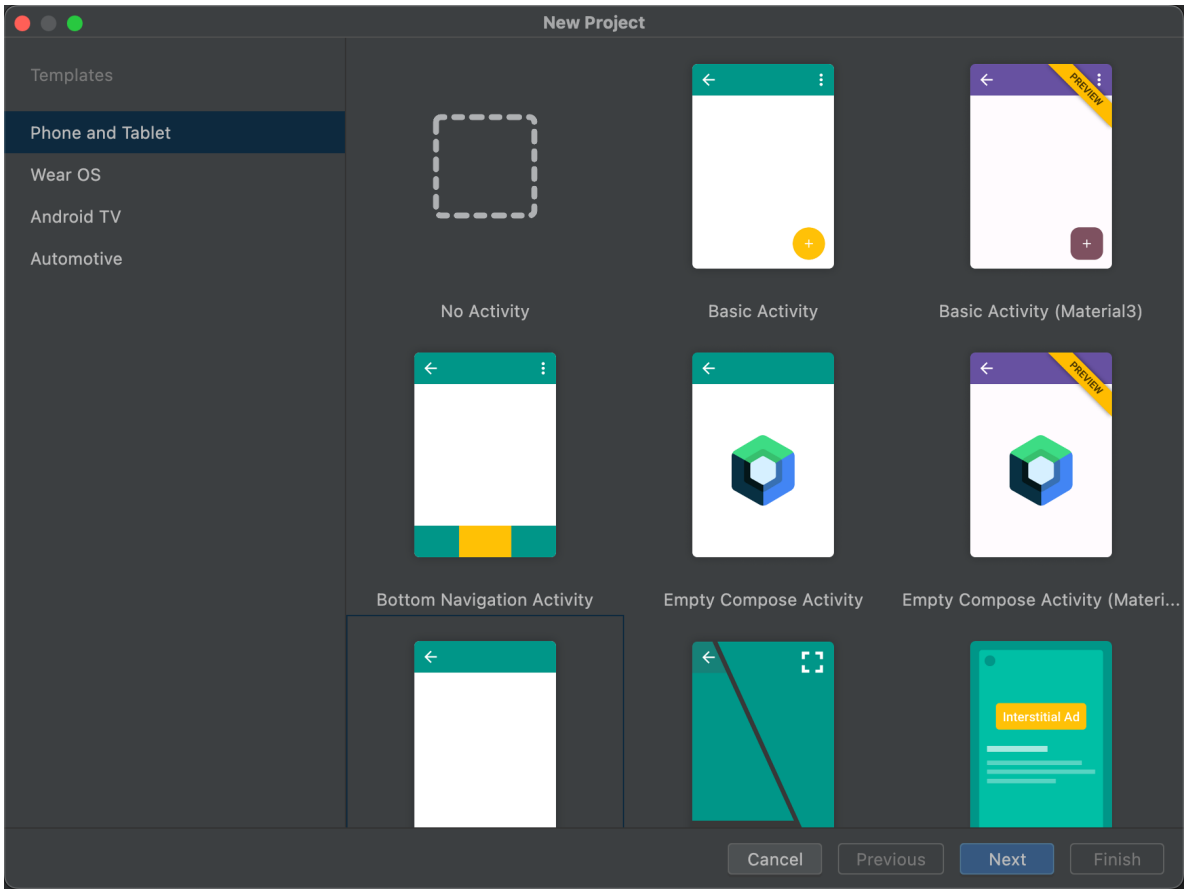
6- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations

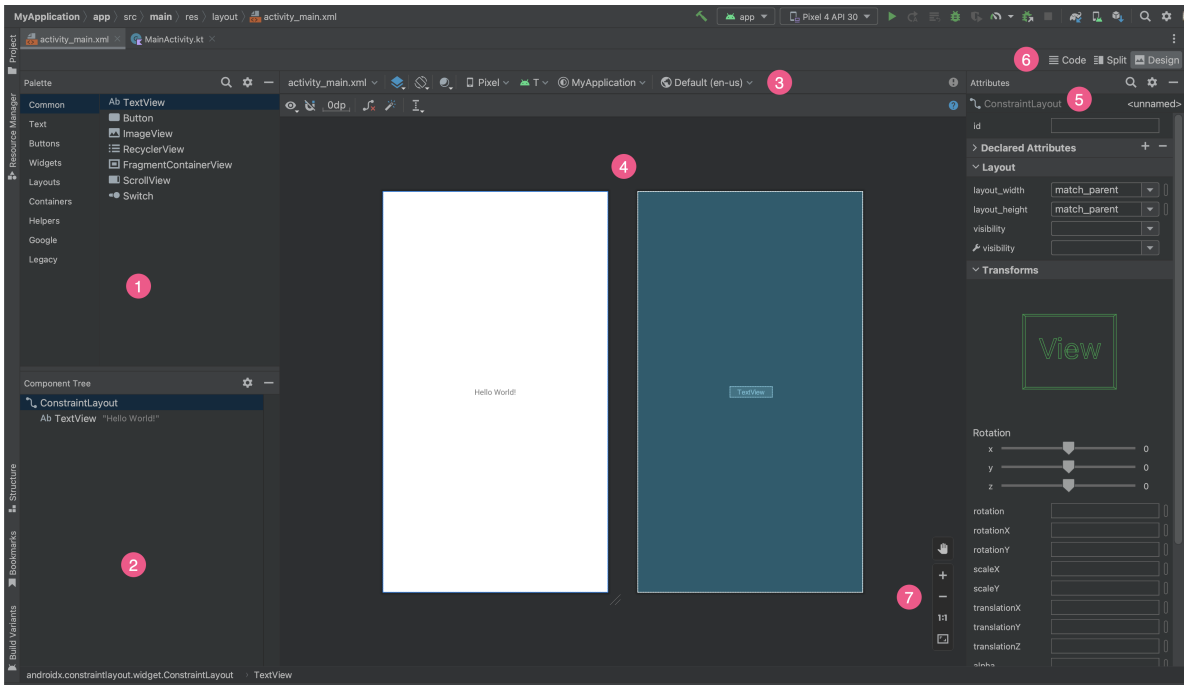
7- Support for building Android Wear apps

8-Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine

9- Android Virtual Device (Emulator) to run and debug apps in the Android studio.




New Project
Templates
Phone and Tablet
Wear OS
Android TV
Automotive
No Activity
Basic Activity
Basic Activity (Material3)
Bottom Navigation Activity
Empty Compose Activity
Empty Compose Activity (Materi...
Interstitial Ad
Cancel
Previous
Next
Finish


Palette
Component Tree
ConstraintLayout
TextView
Hello World!
Attributes
Declared Attributes
Layout
Transforms
View
Rotation

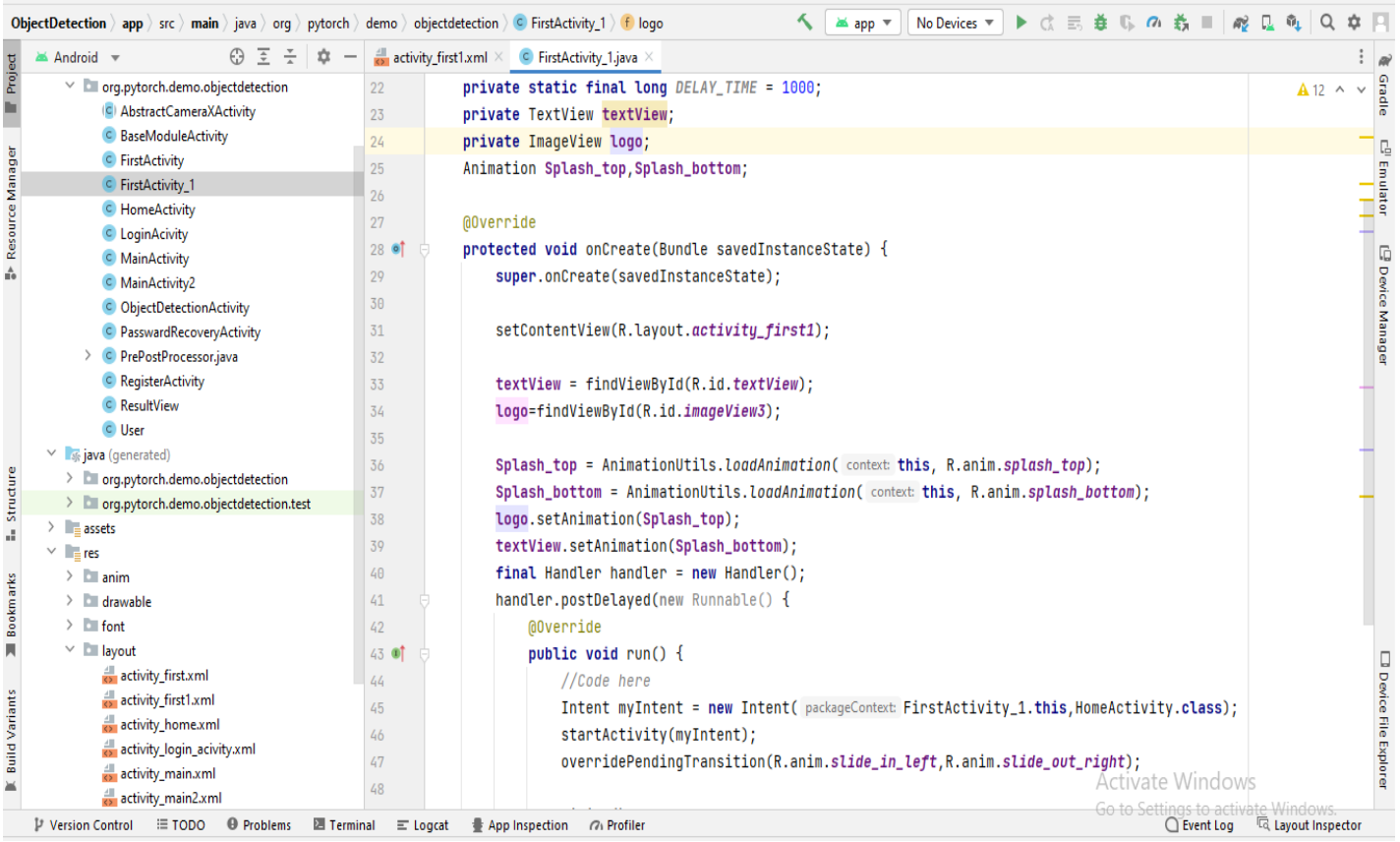
5.2.2 Backend

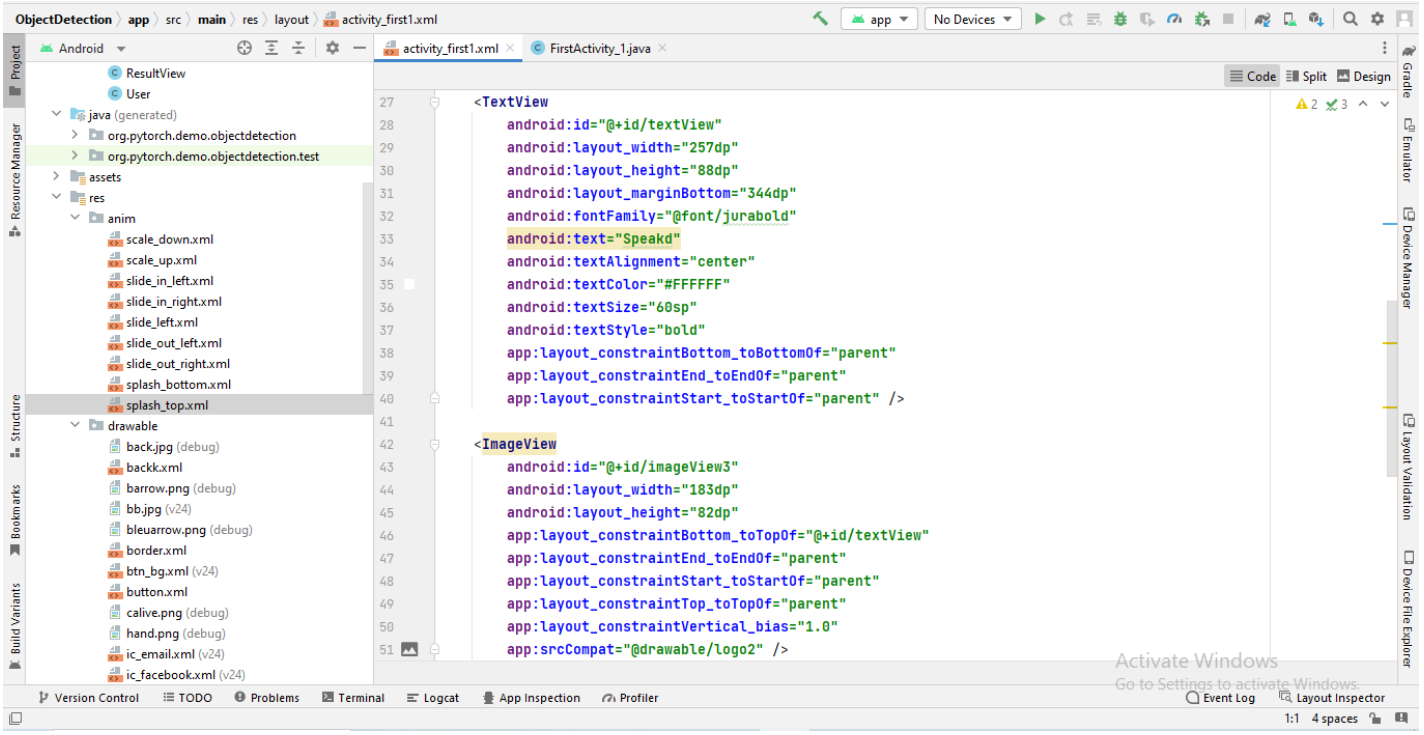
Application Pages:

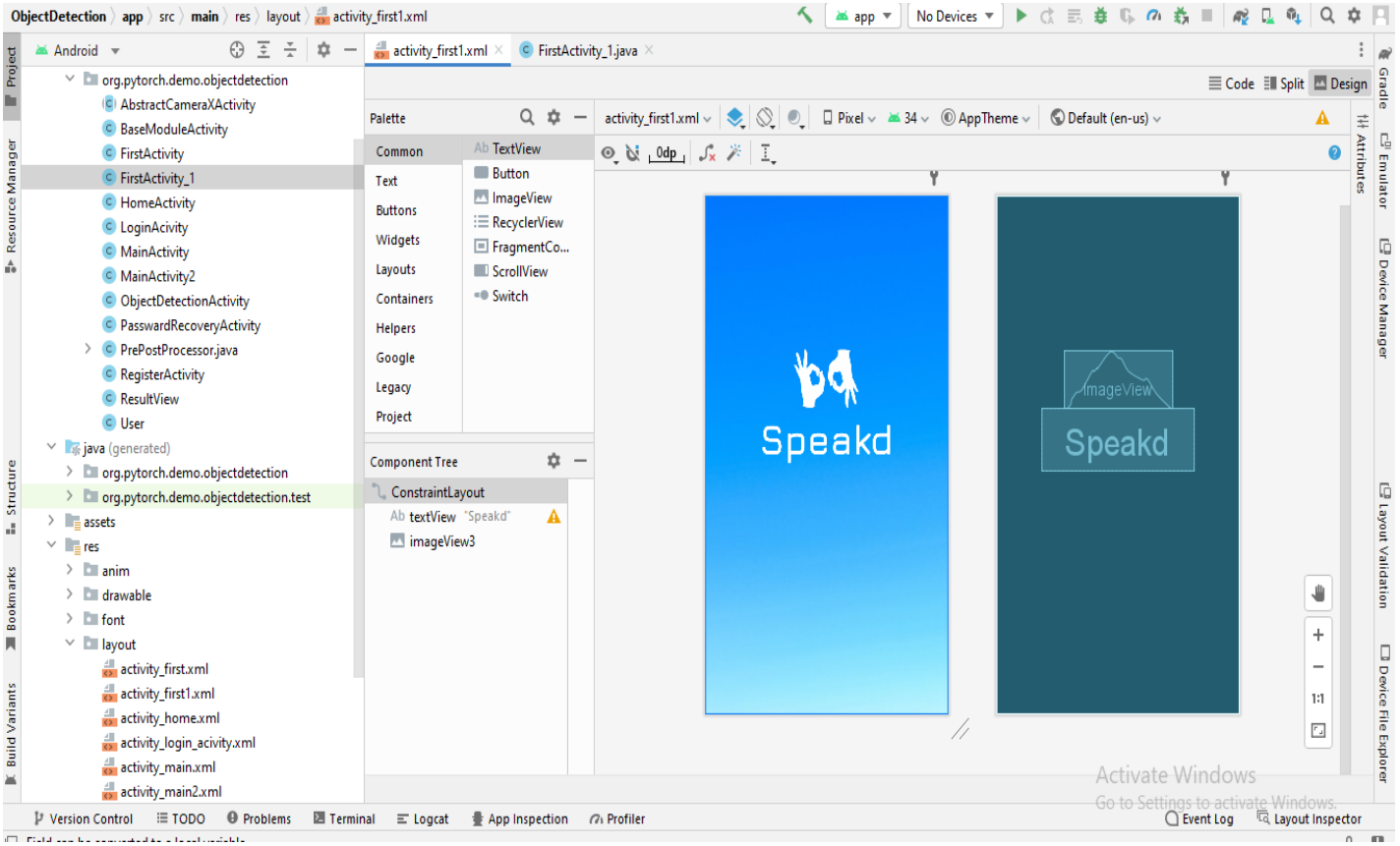
This is an example of splash screen

1-Java Firstactivity-1 In this activity we make splash screen that displays 2500

MS then turn to home activity .the text and image logo has a slight animation







activity-first.xml

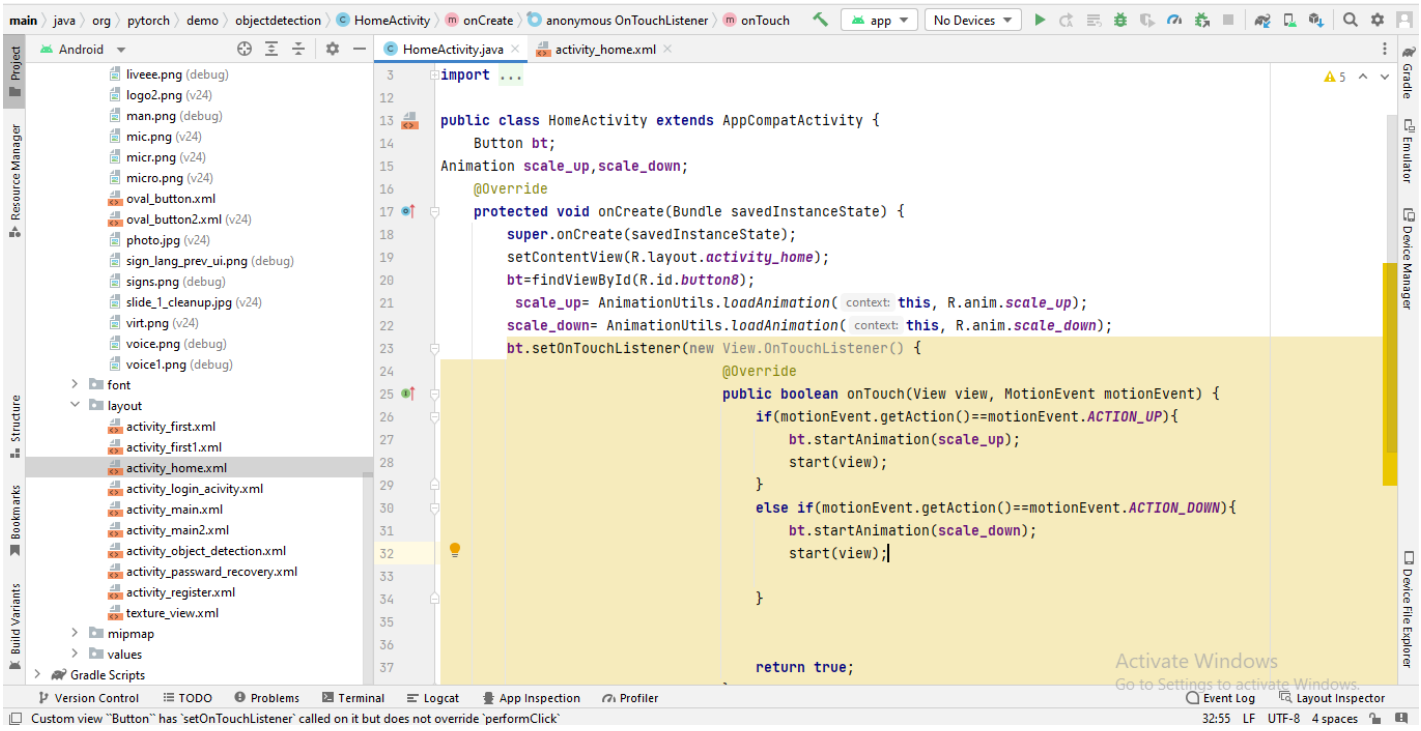
2- Java Home Activity:

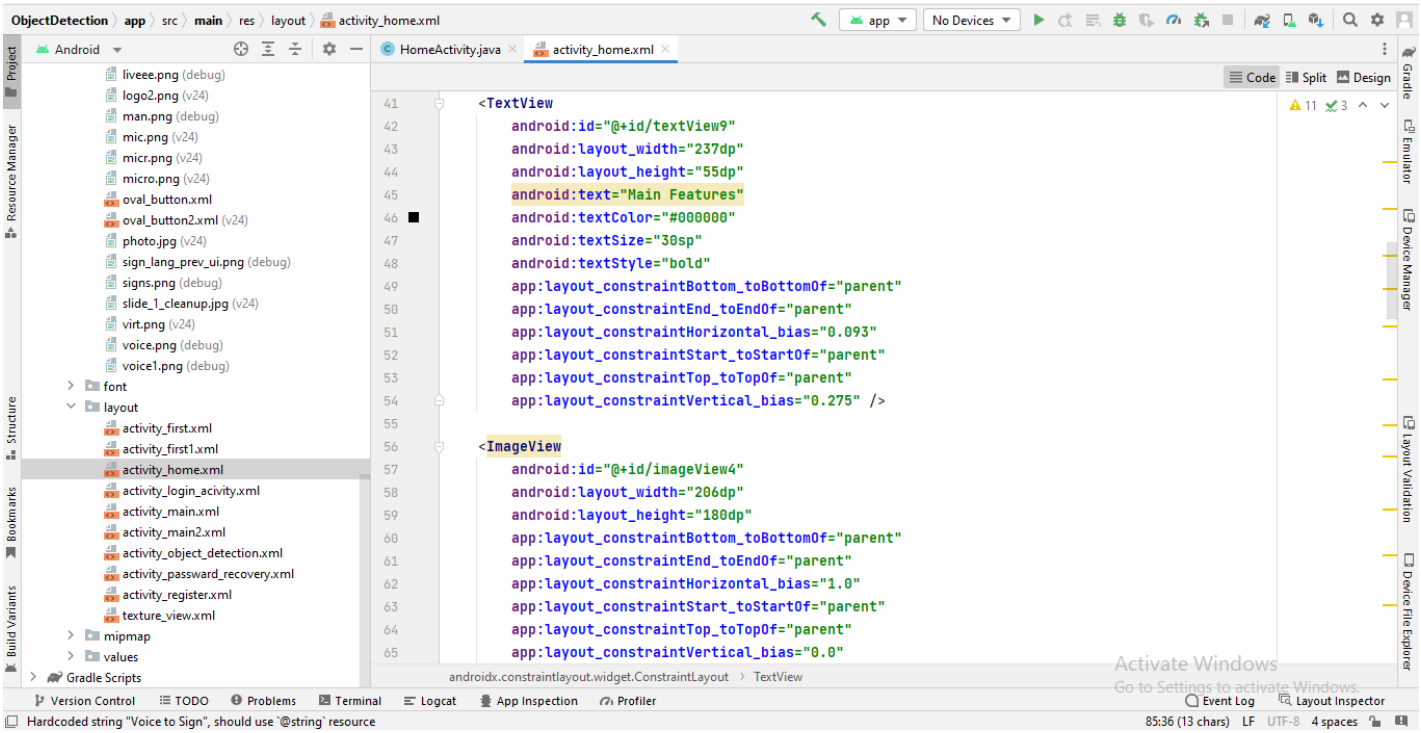
1-This activity contains main features of speakd Application

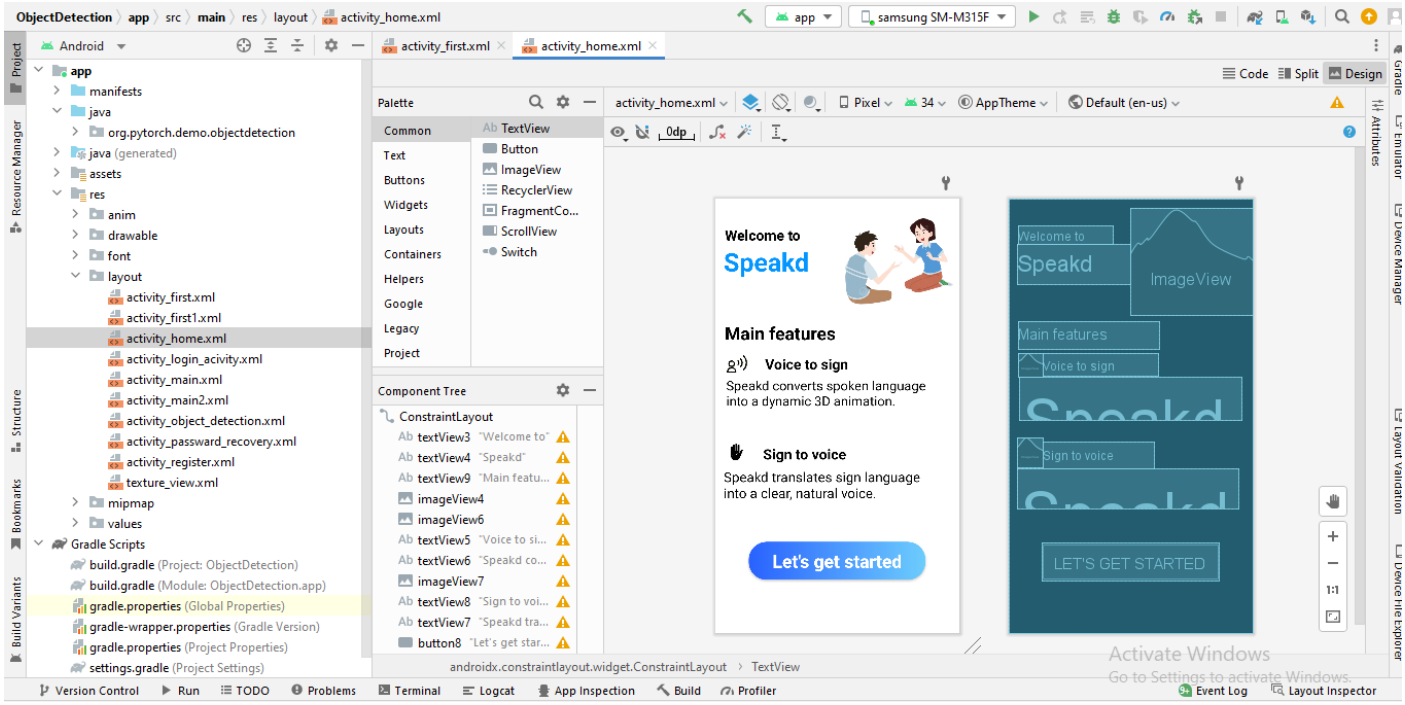
Voice to Sign: Speakd leverages cutting-edge algorithms to translate sign language gestures into text and clear, natural voice representations.

Voice to Text and 3D Avatar Conversion: Conversely, Speakd also facilitates communication for sign language users by converting spoken language into text and dynamic 3D avatars that accurately portray the corresponding signs.

2-Button of let's get started will turn to register activity.







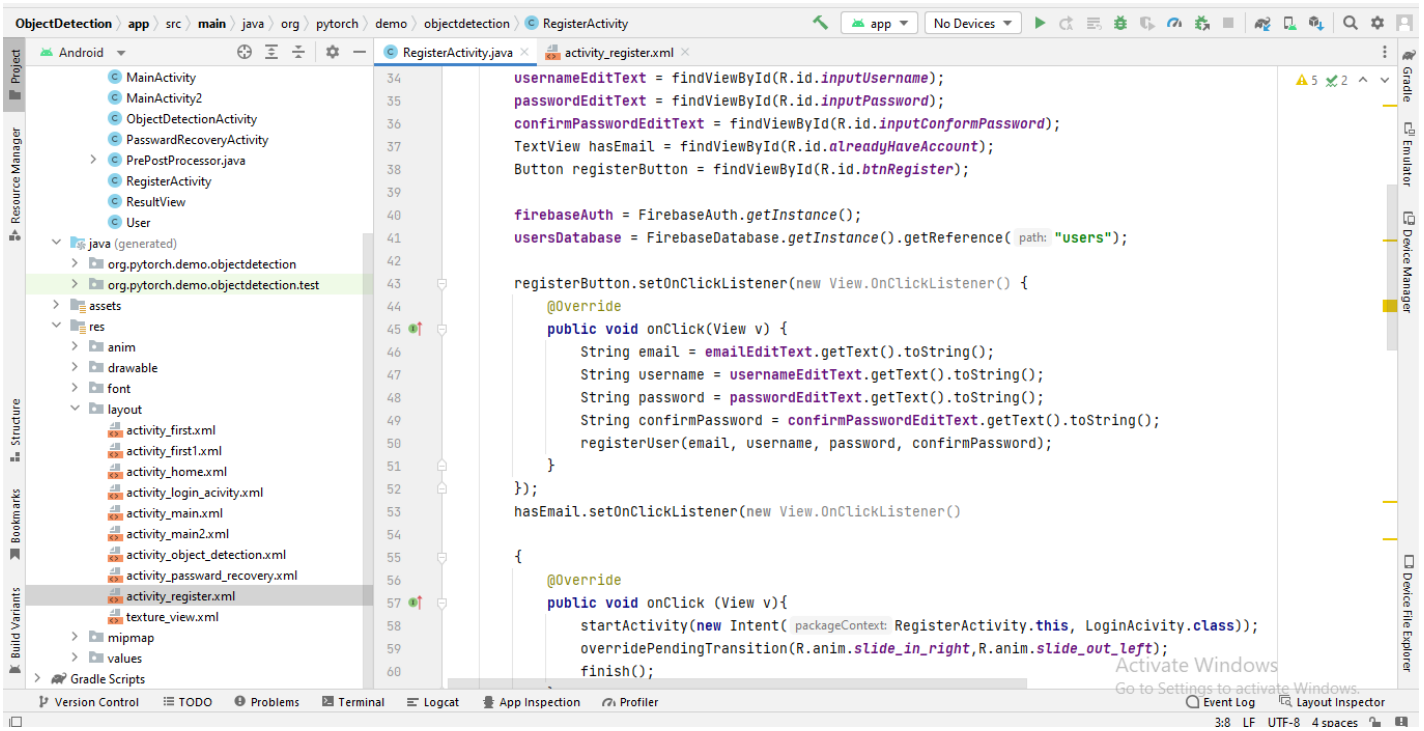
Home activity.xml

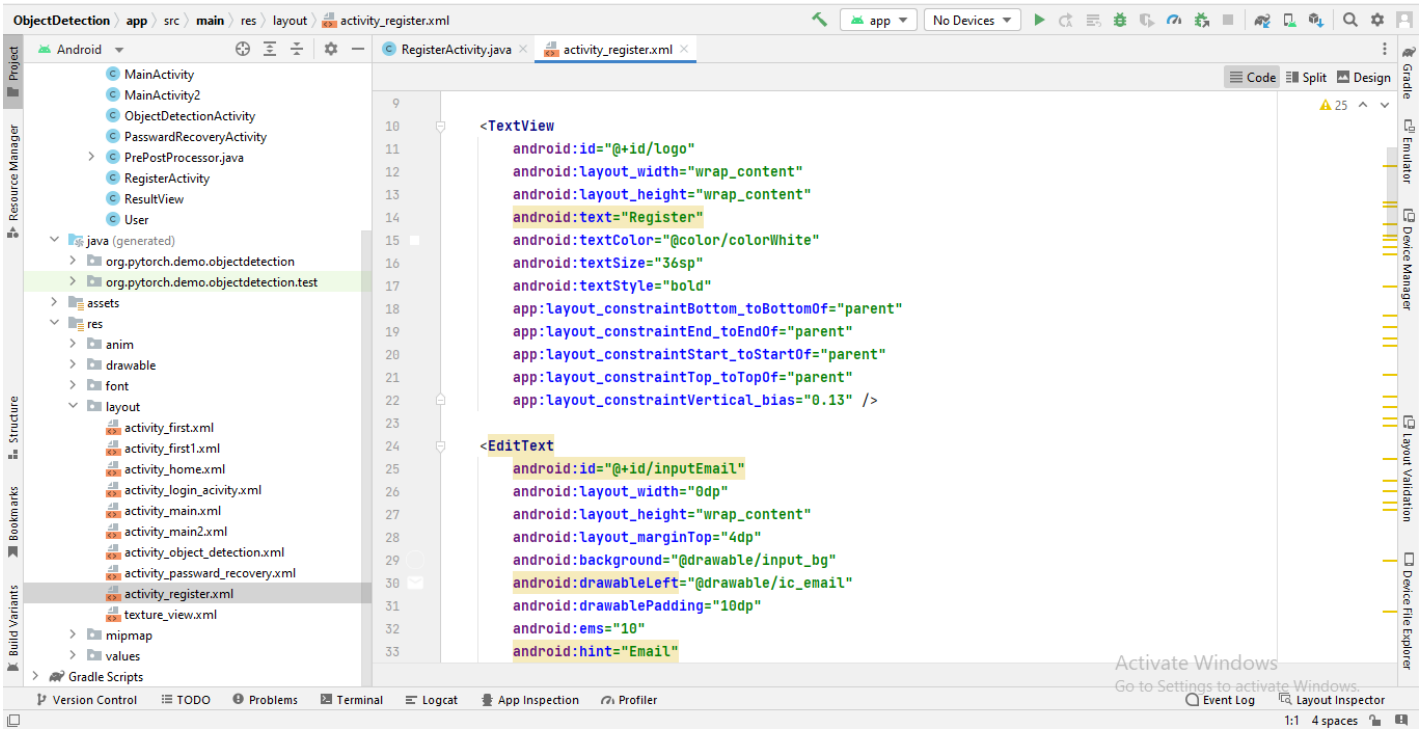
3- Java Register Activity: In this activity we use Google Firebase to store username , email and password to have an account on speakd application and to enable user to login to application.

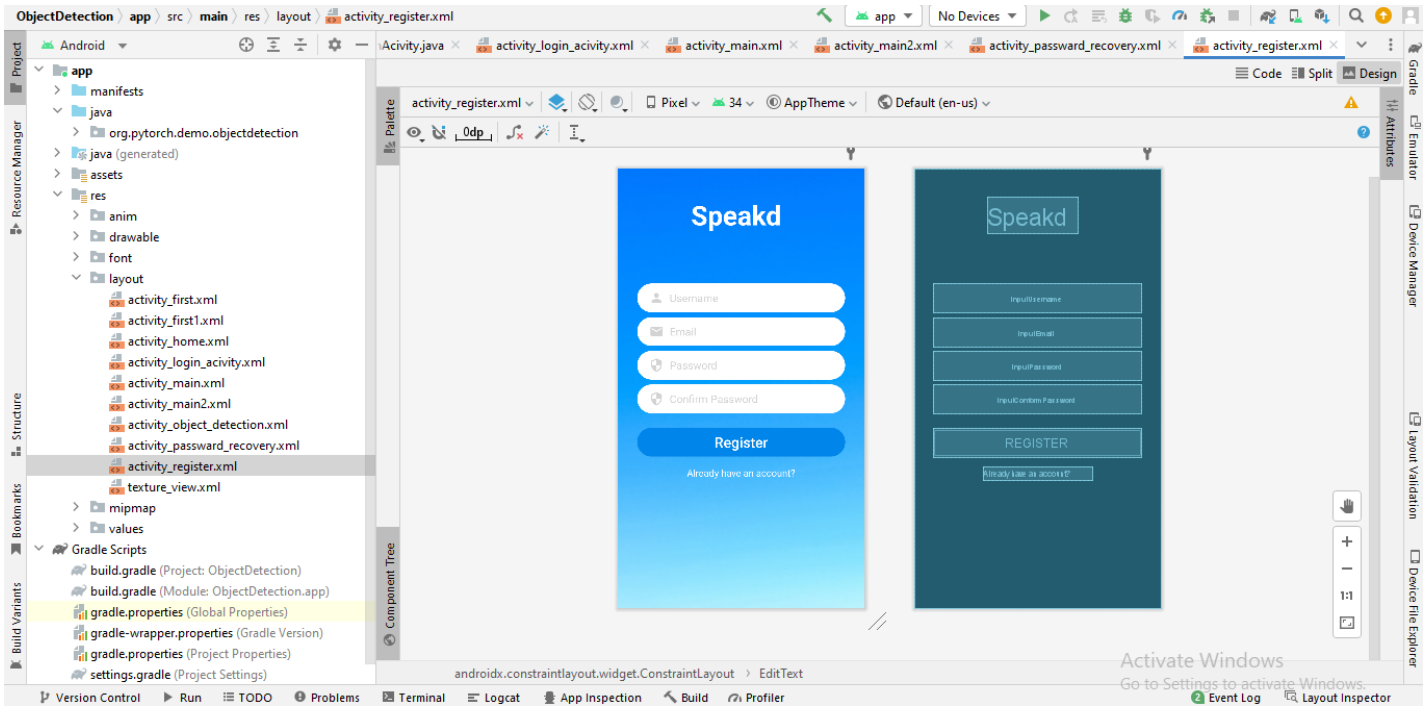
Firebase Features:

1-Authentication. Firebase provides a secure and easy way for users to sign into their app. Developers can use Firebase Authentication to support email and password login, Google Sign-In, Facebook Login and more.

2-Real time Database. The Firebase Real time Database is a cloud-hosted NoSQL database that lets organizations store and sync data in real time across all of their users' devices. This makes it easy to build apps that are always up to date, even when users are offline.



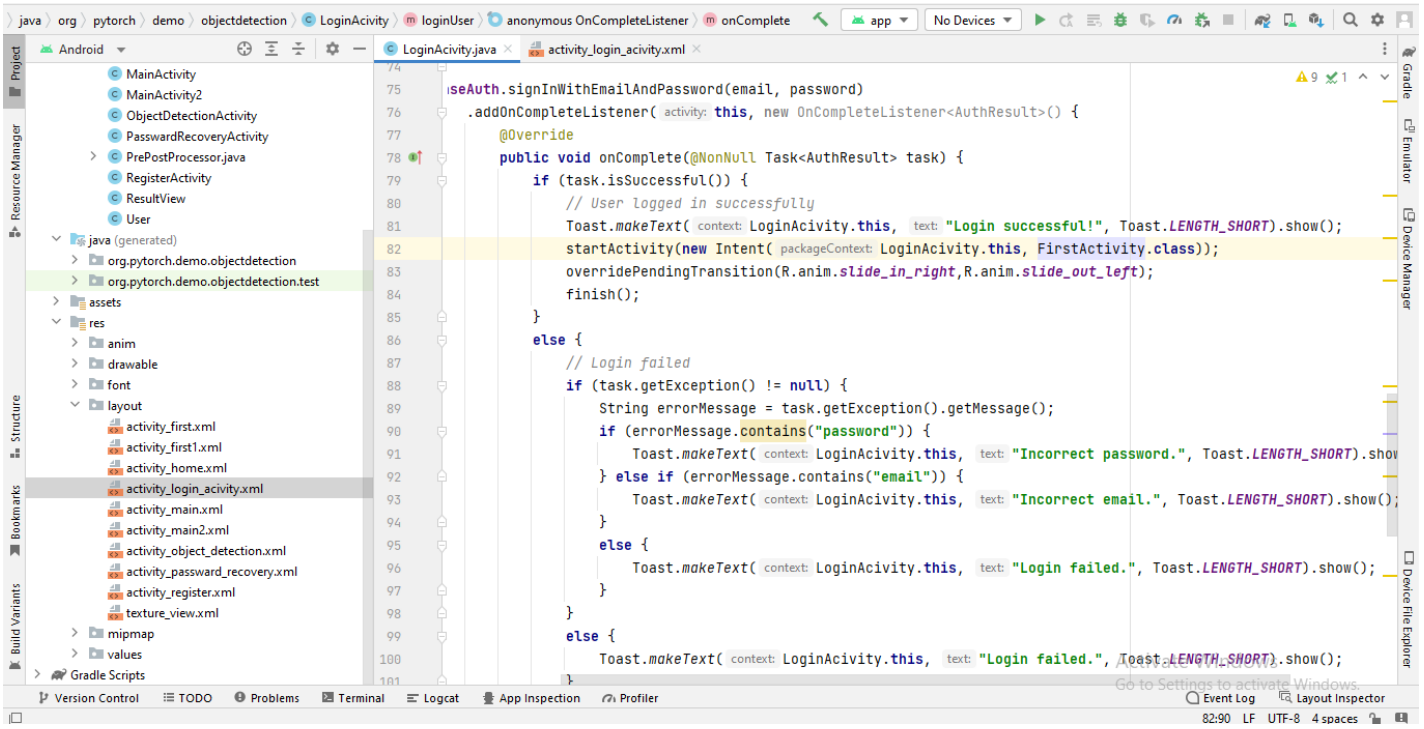


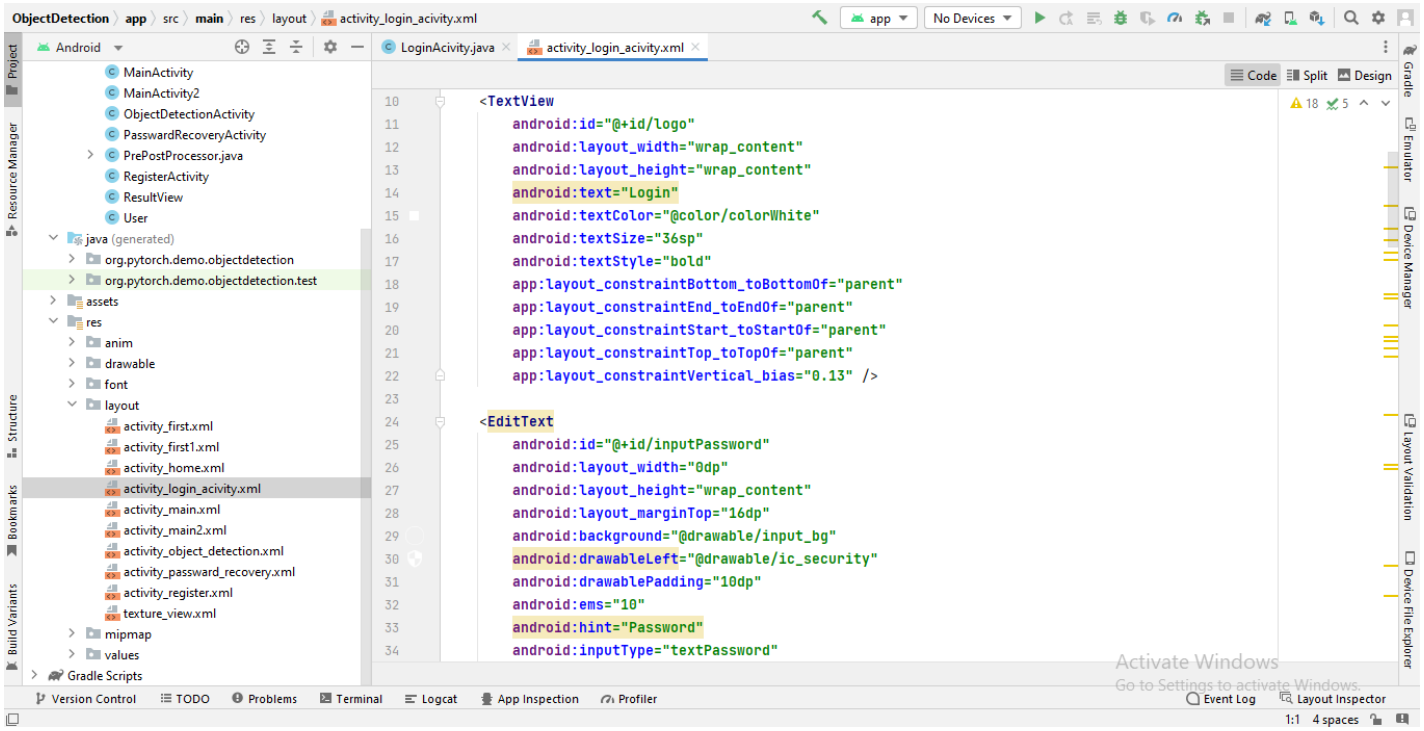


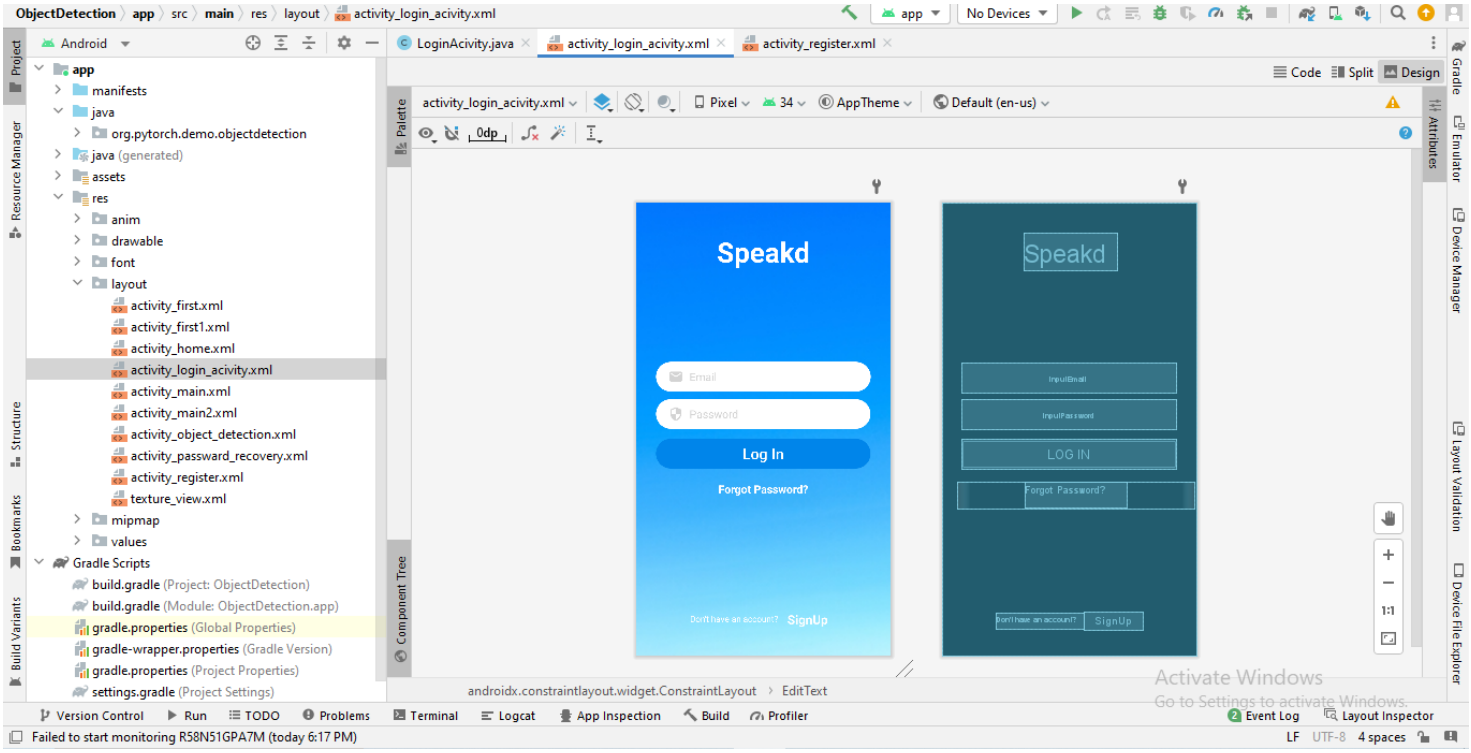
Register Activity.xml

4-Java Login Activity:

In this activity user enter email and password to enable login to the application then the entered email and password will be checked to ensure that they meet standard conditions



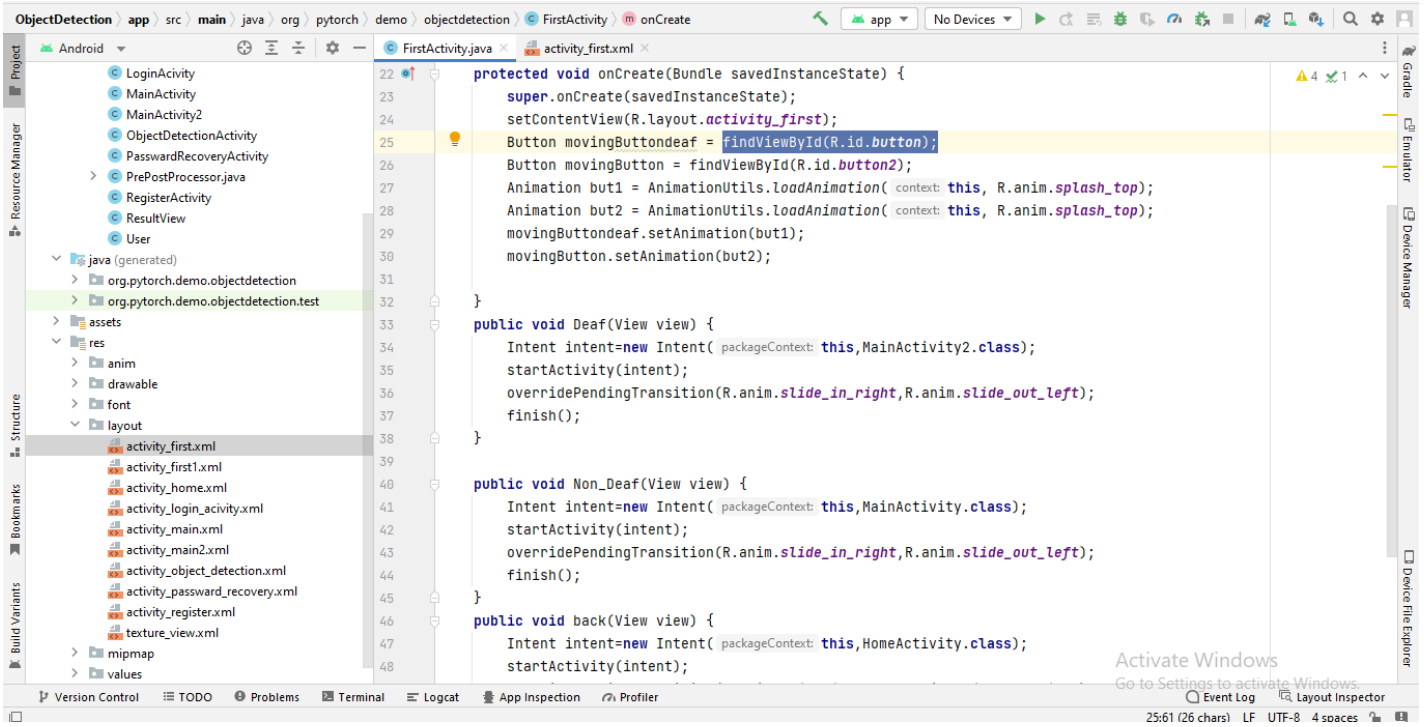


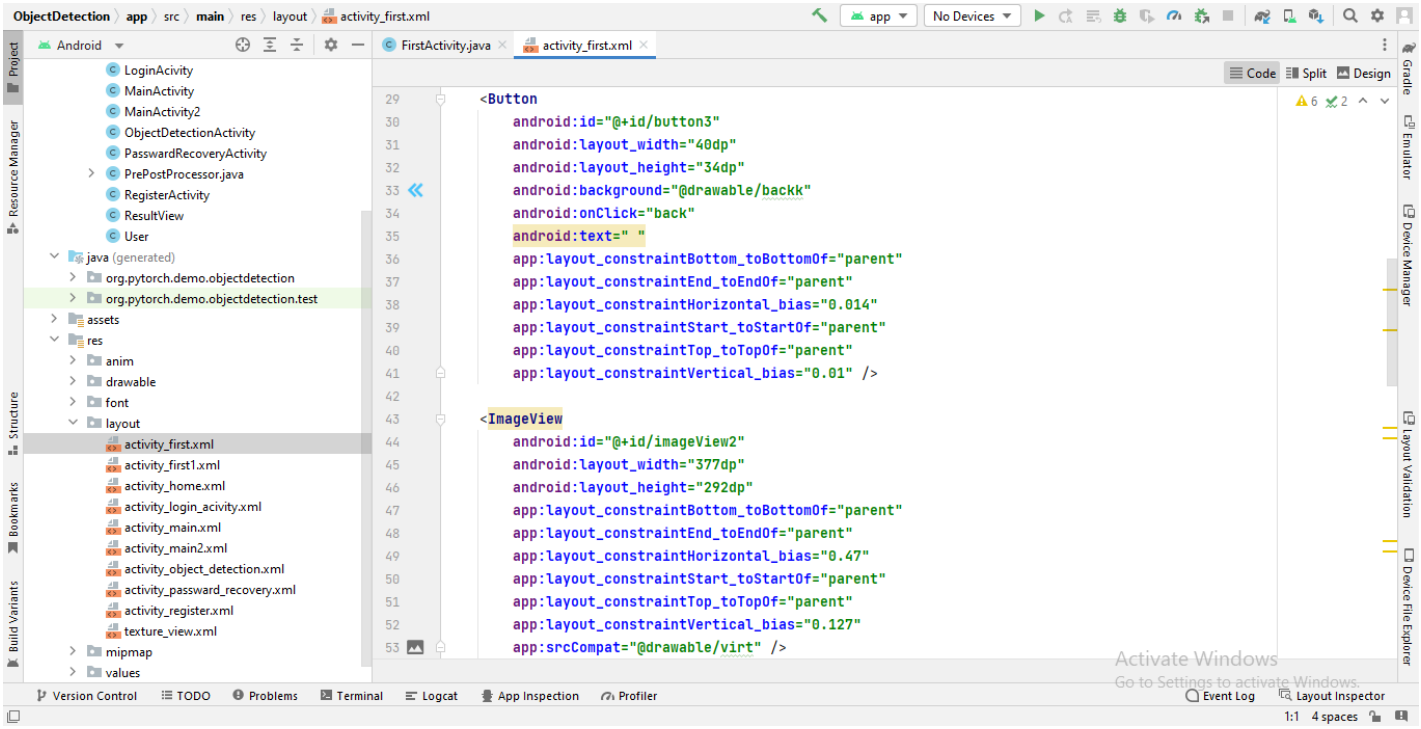


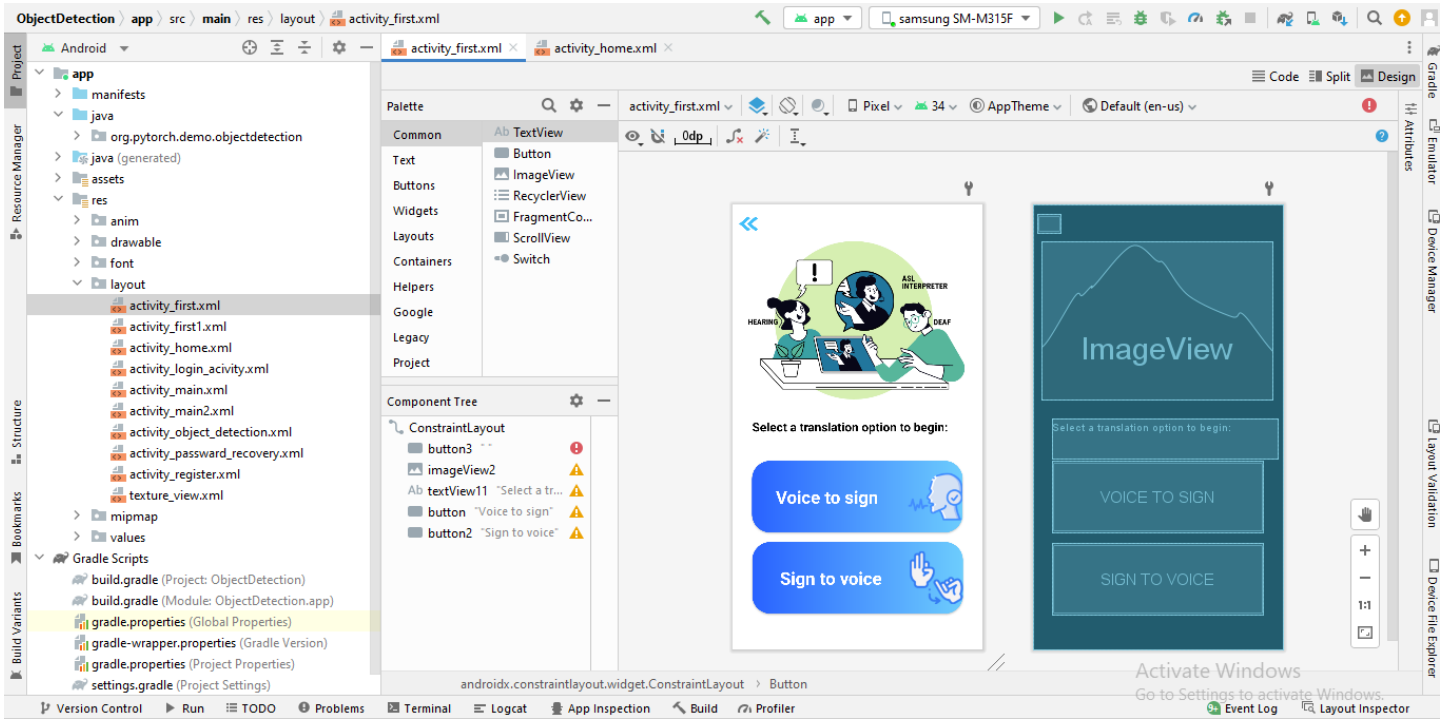
Login Activity.xml

5- Java First Activity:

This activity contains two buttons that represent main features of mobile Application when user taps on Voice to sign activity will turn to MainActivity2 and when taps on Sign to voice activity will turn to Main Activity.





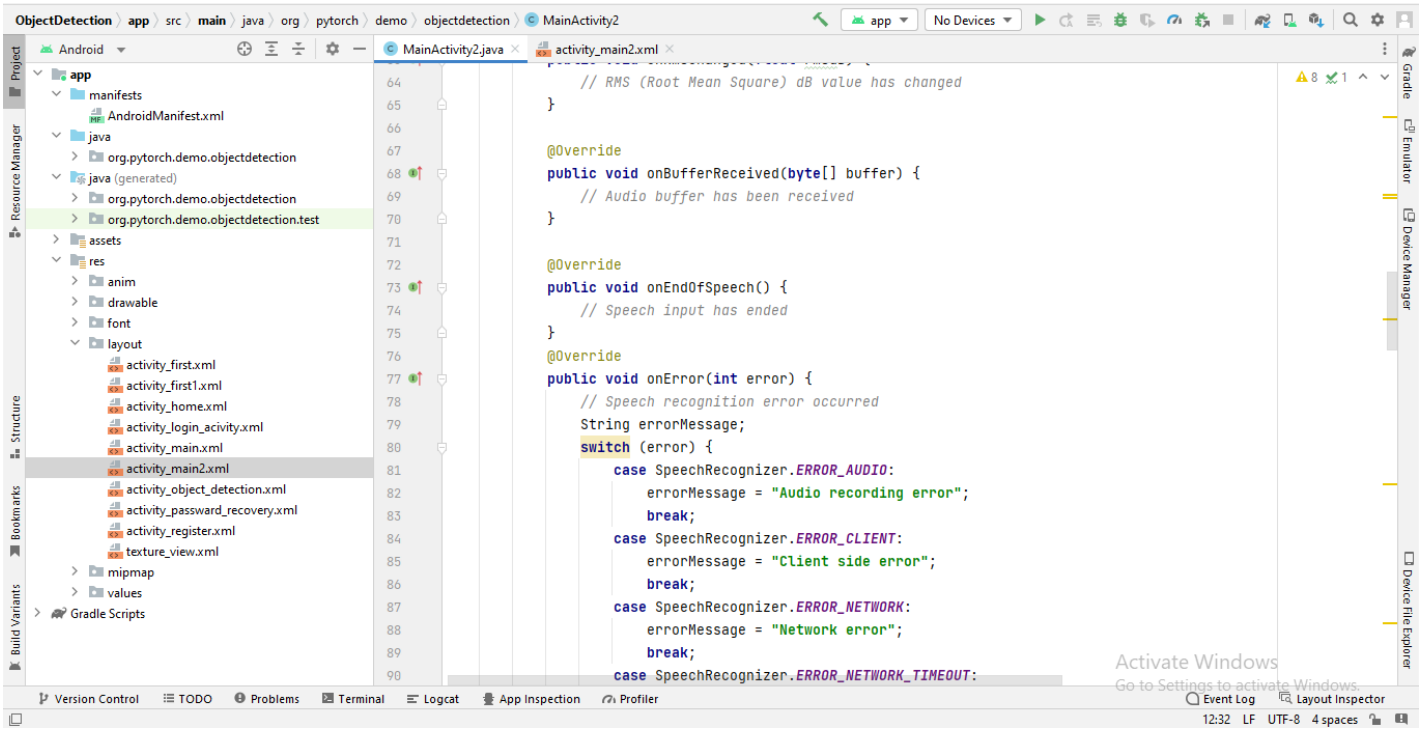


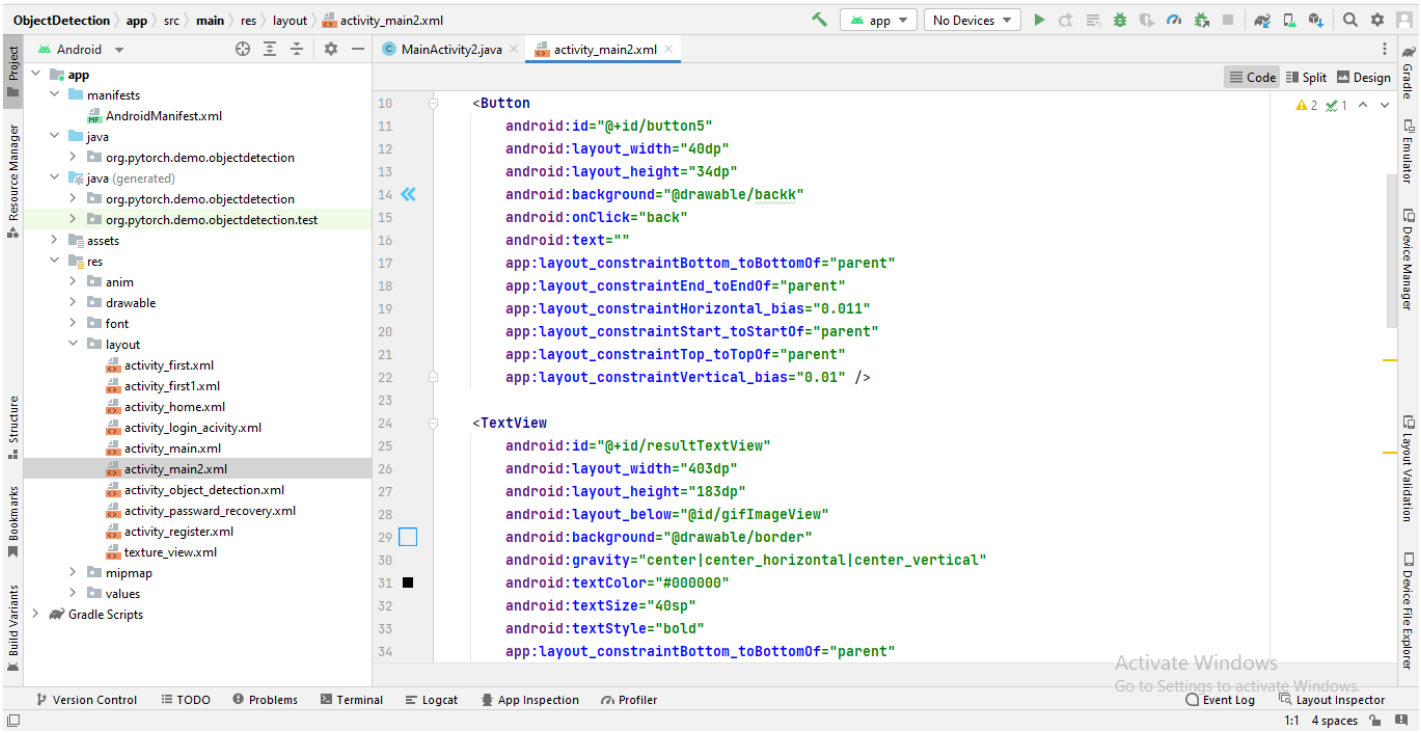
First Activity.xml

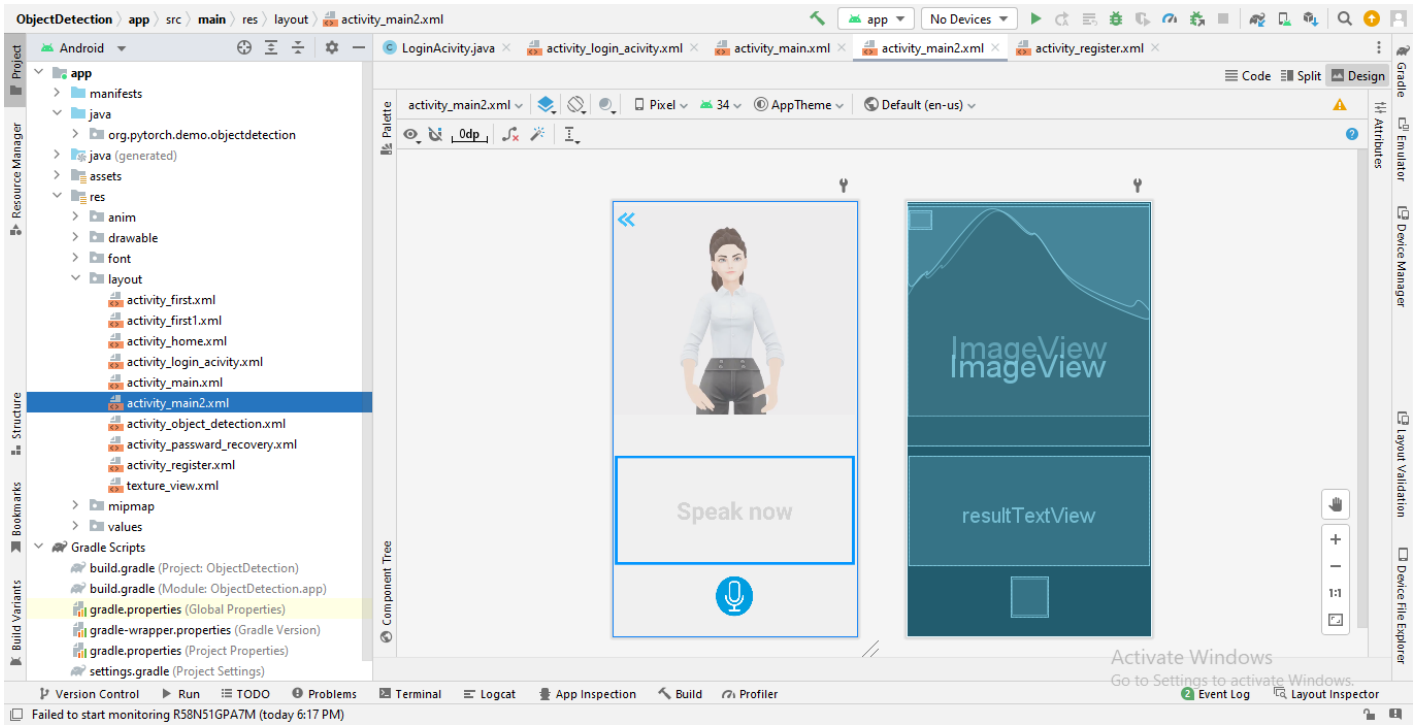
6- Java MainActivity2

In this activity, when user taps on the microphone and starts speaking, his voice will turn to text which appears in a box and Additionally,A 3D Avatar will animate to represent this sentence. We use **android.speech.RecognitionListener** to recognize voice

android.speech.RecognitionListener: is an interface in Android that provides methods to handle speech recognition events. It's used to listen for events from the speech recognizer service, such as when speech is recognized, when an error occurs, or when the recognition process starts or stops. Developers can implement this interface in their Android apps to customize the behavior of speech recognition functionality.







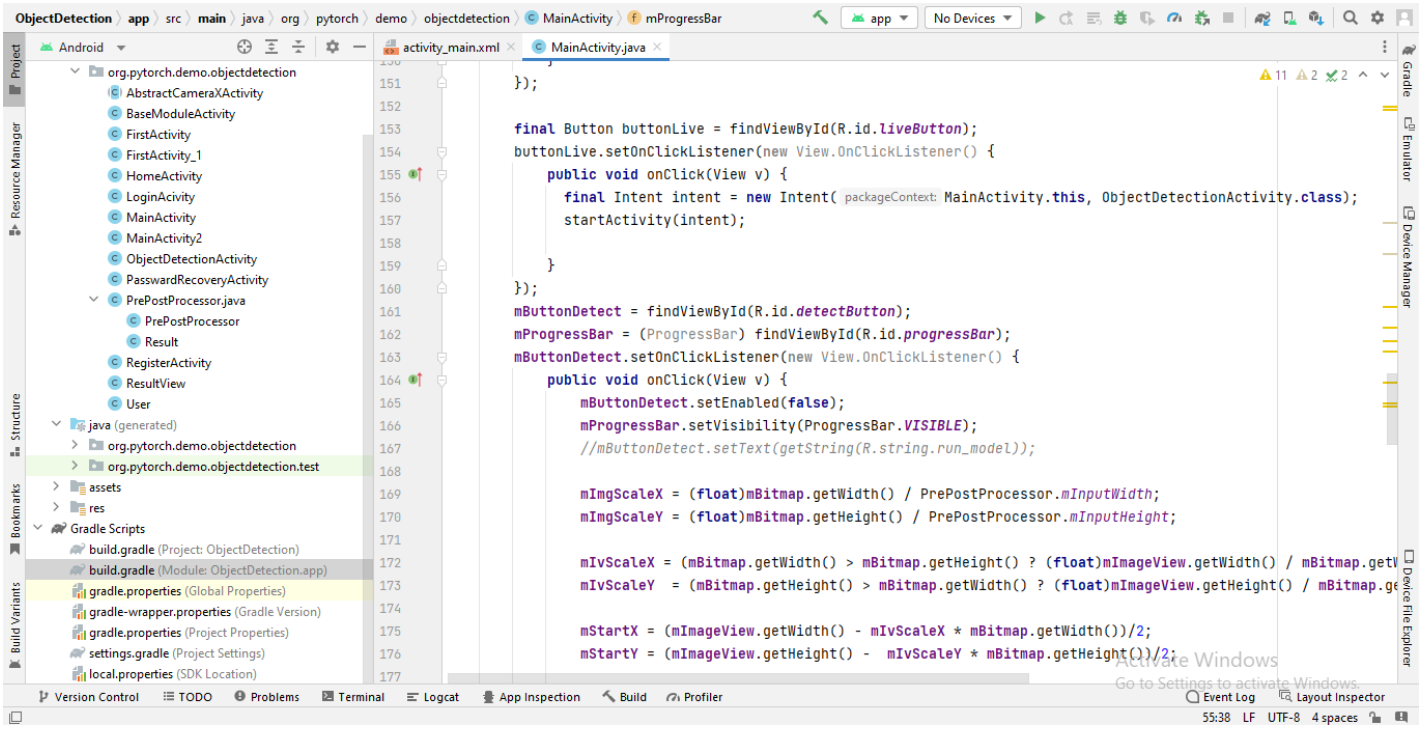
MainActivity2.xml

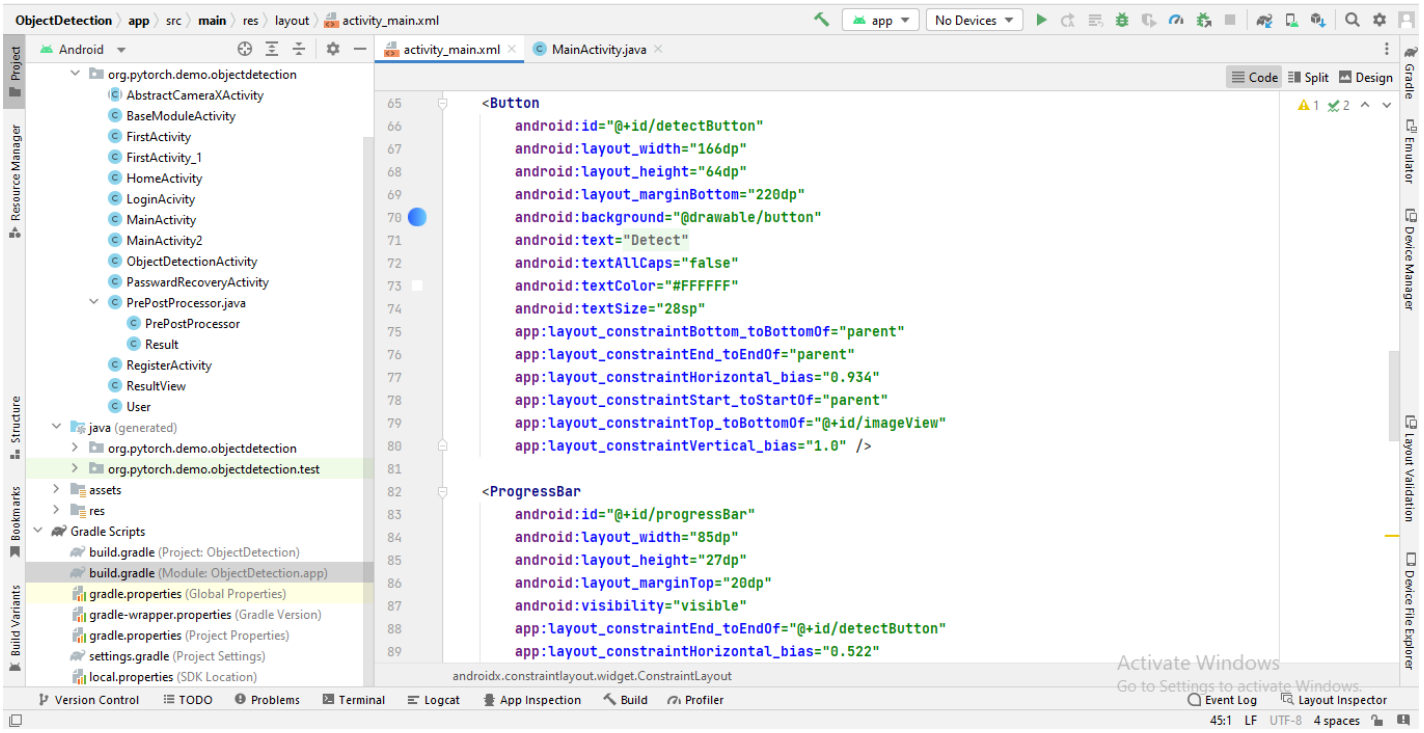
7-Java MainActivity

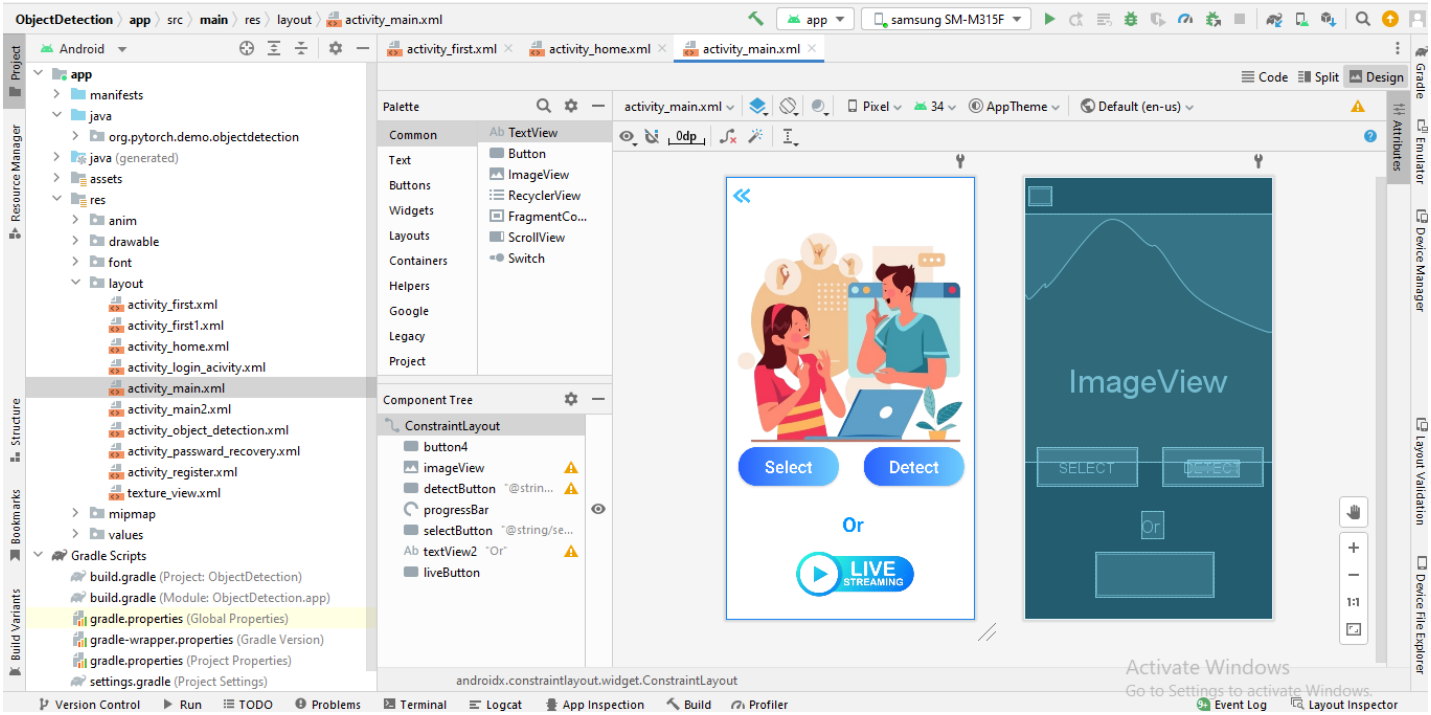
In this activity, user can understand sign language as when select an image or take pictures containing sign languages and taps on detect button, the model will detect sign languages and convert them to text.Subsequently,The text will be converted into voice.

android.speech.tts.TextToSpeech:

is a class in Android that provides functionality for converting text into spoken words. It's part of the Text-to-Speech (TTS) engine available on Android devices. Developers can use this class to synthesize speech from text in their Android applications. It allows customization of speech parameters such as pitch, rate, and volume, and provides methods for controlling speech synthesis, such as starting, pausing, resuming, and stopping speech output.







MainActivity.xml

5.3 Model Deployment

We convert from Yolo model from .pt to .tourchscript and then we optimize this model to be suitable for mobile applications, converting it .ptl Exporting YOLO models to TorchScript is crucial for moving from research to real-world applications. TorchScript, part of the PyTorch framework, helps make this transition smoother by allowing PyTorch models to be used in environments that don't support Python. .ptl refers to PyTorch models that target the lite interpreter (mobile) on Android and iOS.

5.3.1 Main android modules used for deployment:

1- androidx.camera:camera-core:$camerax_version$

2- androidx.camera:camera-camera2:$camerax - version$

3- org.pytorch:pytorch-android-lite:1.12.2

4- org.pytorch:pytorch-android-torchvision-lite:1.12.2

5- com.github.bumptech.glide:glide:4.12.0

5.3.2 Sign Language detection steps:

1-Selecting an image or opening the camera for live streaming.

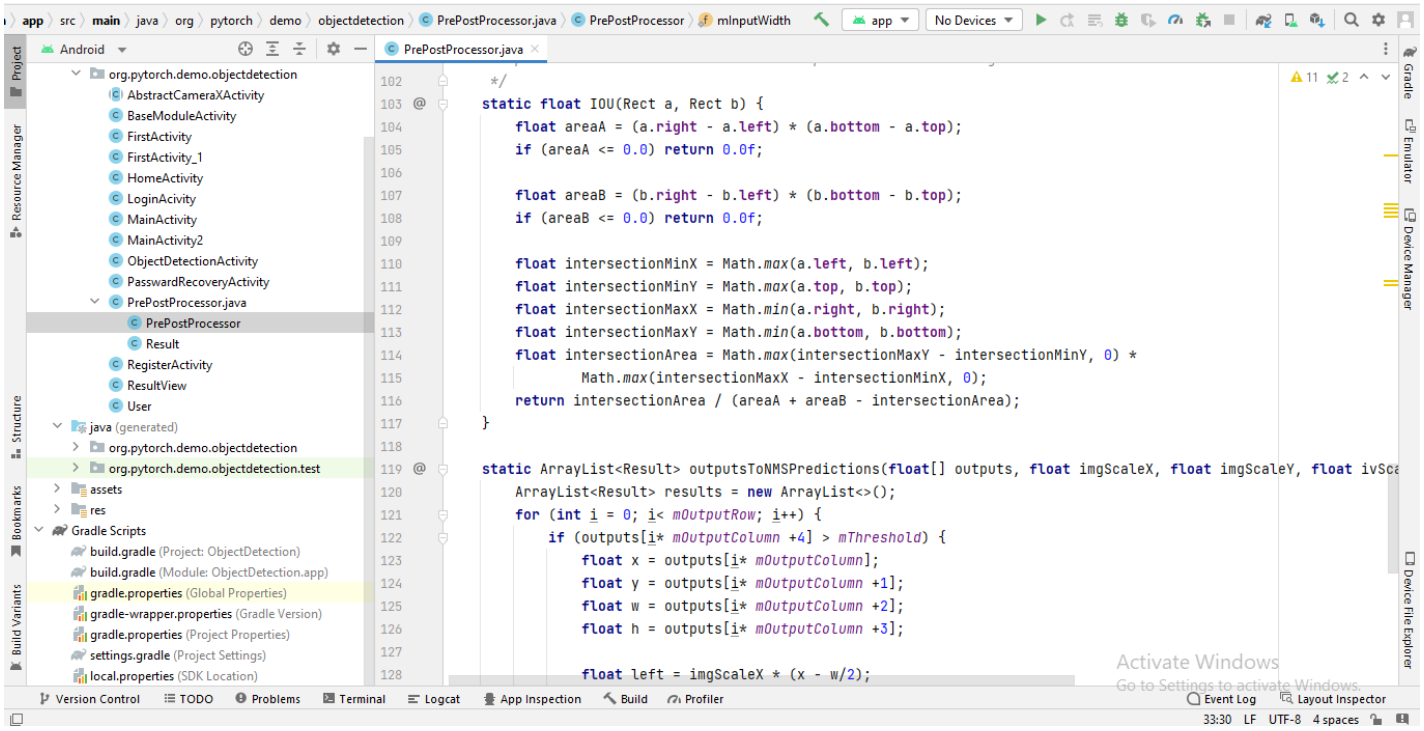
2-The selected image will be preprocessed to be ready for the model.

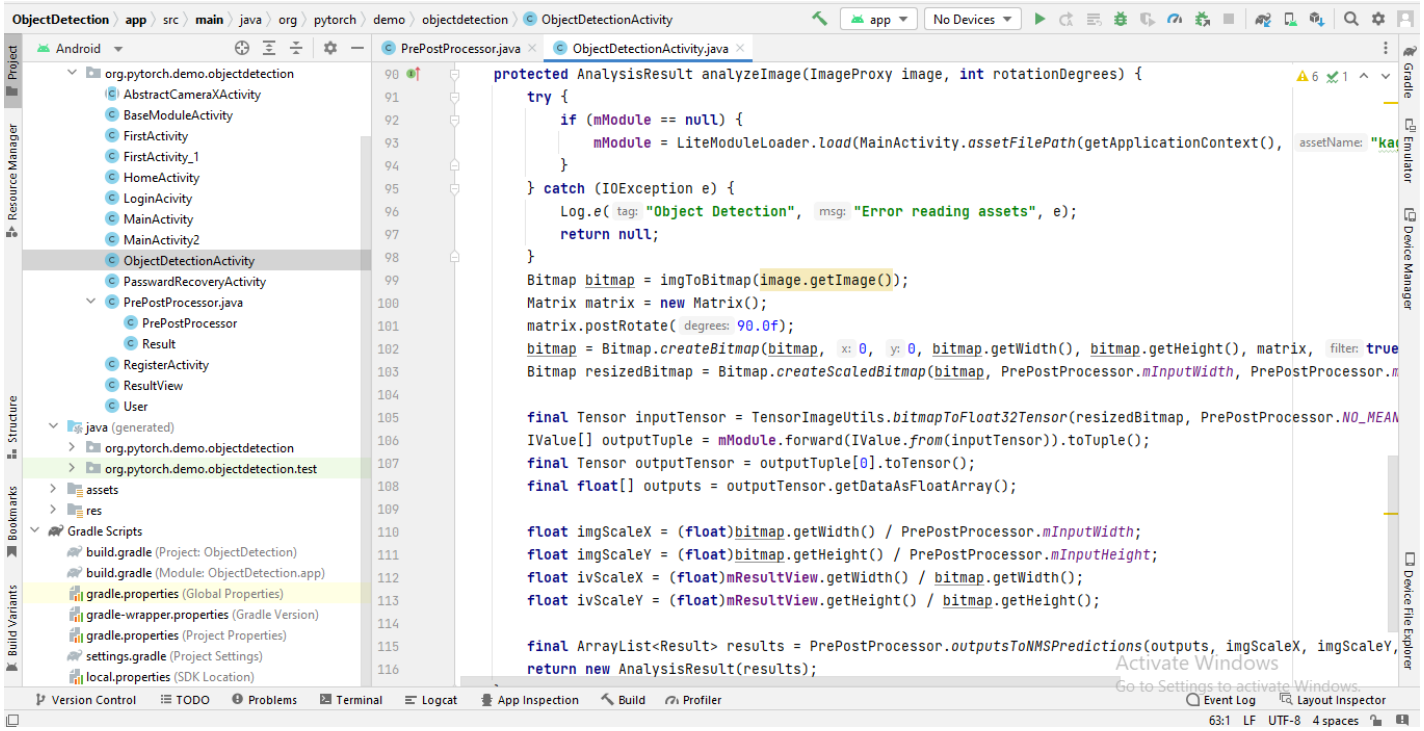
3-We will determine the width and height of image, threshold values, and number of classes.

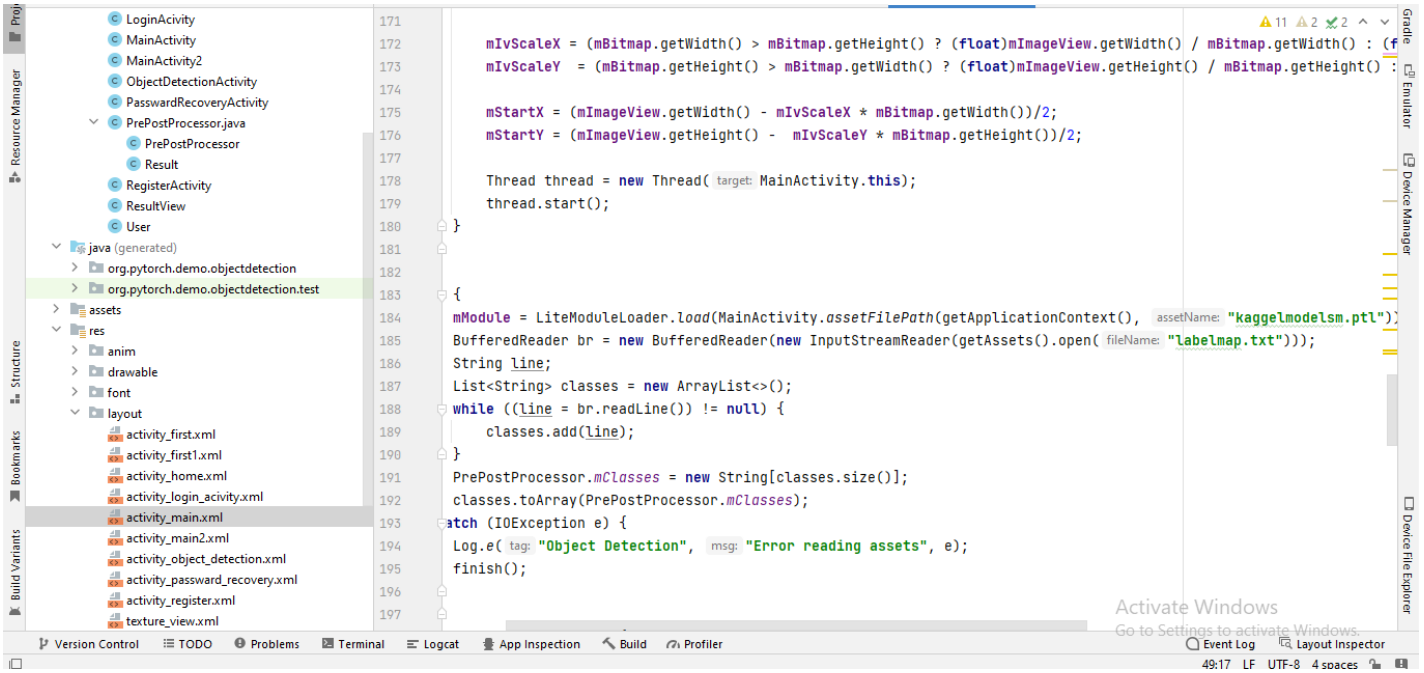
4-The model will start to detect image and select the best

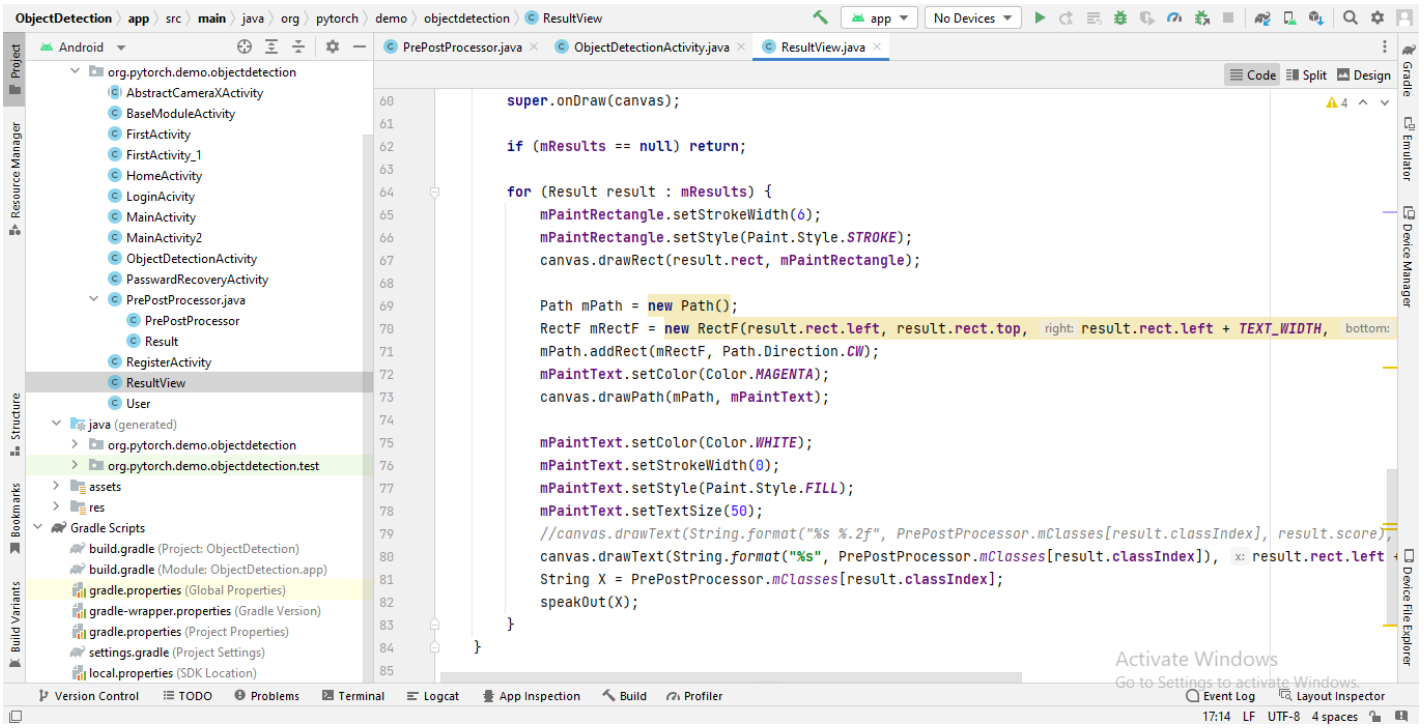
class from labelmap.txt file located in the android assets folder.

6- The identified class will be overlaid on the image and then converted into clear voice.


app src main java org pytorch demo objectdetection PrePostProcessor.java PrePostProcessor mInputWidth
No Devices
Android
org.pytorch.demo.objectdetection
AbstractCameraXActivity
BaseModuleActivity
FirstActivity
FirstActivity_1
HomeActivity
LoginAcivity
MainActivity
MainActivity2
ObjectDetectionActivity
PasswardRecoveryActivity
PrePostProcessor.java
PrePostProcessor
Result
RegisterActivity
ResultView
User
java (generated)
org.pytorch.demo.objectdetection
org.pytorch.demo.objectdetection.test
assets
res
Gradle Scripts
build.gradle (Project: ObjectDetection)
build.gradle (Module: ObjectDetection.app)
gradle.properties (Global Properties)
gradle-wrapper.properties (Gradle Version)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
PrePostProcessor.java
*/
static float IOU(Rect a, Rect b) {
float areaA = (a.right - a.left) * (a.bottom - a.top);
if (areaA <= 0.0) return 0.0f;
float areaB = (b.right - b.left) * (b.bottom - b.top);
if (areaB <= 0.0) return 0.0f;
float intersectionMinX = Math.max(a.left, b.left);
float intersectionMinY = Math.max(a.top, b.top);
float intersectionMaxX = Math.min(a.right, b.right);
float intersectionMaxY = Math.min(a.bottom, b.bottom);
float intersectionArea = Math.max(intersectionMaxY - intersectionMinY, 0) *
Math.max(intersectionMaxX - intersectionMinX, 0);
return intersectionArea / (areaA + areaB - intersectionArea);
}
static ArrayList<Result> outputsToNMSPredictions(float[] outputs, float imgScaleX, float imgScaleY, float ivSca
ArrayList<Result> results = new ArrayList<>();
for (int i = 0; i< mOutputRow; i++) {
if (outputs[i* mOutputColumn +4] > mThreshold) {
float x = outputs[i* mOutputColumn];
float y = outputs[i* mOutputColumn +1];
float w = outputs[i* mOutputColumn +2];
float h = outputs[i* mOutputColumn +3];
float left = imgScaleX * (x - w/2);
Activate Windows
Go to Settings to activate Windows.
Version Control TODO Problems Terminal Logcat App Inspection Build Profiler
Event Log Layout Inspector
33:30 LF UTF-8 4 spaces


ObjectDetection app src main java org pytorch demo objectdetection ObjectDetectionActivity
No Devices
PrePostProcessor.java
ObjectDetectionActivity.java
protected AnalysisResult analyzeImage(ImageProxy image, int rotationDegrees) {
try {
if (mModule == null) {
mModule = LiteModuleLoader.load(MainActivity.assetFilePath(getApplicationContext(),
}
} catch (IOException e) {
Log.e(tag: "Object Detection", msg: "Error reading assets", e);
return null;
}
Bitmap bitmap = imgToBitmap(image.getImage());
Matrix matrix = new Matrix();
matrix.postRotate(degrees: 90.0f);
bitmap = Bitmap.createBitmap(bitmap, x: 0, y: 0, bitmap.getWidth(), bitmap.getHeight(), matrix,
Bitmap resizedBitmap = Bitmap.createScaledBitmap(bitmap, PrePostProcessor.mInputWidth, PrePostProcessor.
final Tensor inputTensor = TensorImageUtils.bitmapToFloat32Tensor(resizedBitmap, PrePostProcessor.NO_MEA
IValue[] outputTuple = mModule.forward(IValue.from(inputTensor)).toTuple();
final Tensor outputTensor = outputTuple[0].toTensor();
final float[] outputs = outputTensor.getDataAsFloatArray();
float imgScaleX = (float)bitmap.getWidth() / PrePostProcessor.mInputWidth;
float imgScaleY = (float)bitmap.getHeight() / PrePostProcessor.mInputHeight;
float ivScaleX = (float)mResultView.getWidth() / bitmap.getWidth();
float ivScaleY = (float)mResultView.getHeight() / bitmap.getHeight();
final ArrayList<Result> results = PrePostProcessor.outputsToNMSPredictions(outputs, imgScaleX, imgScaleY,
return new AnalysisResult(results);
Activate Windows
Go to Settings to activate Windows.
Version Control TODO Problems Terminal Logcat App Inspection Build Profiler
Event Log Layout Inspector
63:1 LF UTF-8 4 spaces


LoginAcivity
MainActivity
MainActivity2
ObjectDetectionActivity
PasswardRecoveryActivity
PrePostProcessor.java
PrePostProcessor
Result
RegisterActivity
ResultView
User
java (generated)
org.pytorch.demo.objectdetection
org.pytorch.demo.objectdetection.test
assets
res
anim
drawable
font
layout
activity_first.xml
activity_first1.xml
activity_home.xml
activity_login_acivity.xml
activity_main.xml
activity_main2.xml
activity_object_detection.xml
activity_passward_recovery.xml
activity_register.xml
texture_view.xml
mIvScaleX = (mBitmap.getWidth() > mBitmap.getHeight() ? (float)mImageView.getWidth() / mBitmap.getWidth() : (f
mIvScaleY = (mBitmap.getHeight() > mBitmap.getWidth() ? (float)mImageView.getHeight() / mBitmap.getHeight() :
mStartX = (mImageView.getWidth() - mIvScaleX * mBitmap.getWidth())/2;
mStartY = (mImageView.getHeight() - mIvScaleY * mBitmap.getHeight())/2;
Thread thread = new Thread(target: MainActivity.this);
thread.start();
}
{
mModule = LiteModuleLoader.load(MainActivity.assetFilePath(getApplicationContext(), assetName: "kaggelmodelsm.ptl"));
BufferedReader br = new BufferedReader(new InputStreamReader(getAssets().open(fileName: "labelmap.txt")));
String line;
List<String> classes = new ArrayList<>();
while ((line = br.readLine()) != null) {
classes.add(line);
}
PrePostProcessor.mClasses = new String[classes.size()];
classes.toArray(PrePostProcessor.mClasses);
atch (IOException e) {
Log.e(tag: "Object Detection", msg: "Error reading assets", e);
finish();
Activate Windows
Go to Settings to activate Windows.
Version Control
TODO
Problems
Terminal
Logcat
App Inspection
Build
Profiler
Event Log
Layout Inspector
49:17 LF UTF-8 4 spaces


ObjectDetection app src main java org pytorch demo objectdetection ResultView
app
No Devices
Android
PrePostProcessor.java
ObjectDetectionActivity.java
ResultView.java
Code
Split
Design
org.pytorch.demo.objectdetection
AbstractCameraXActivity
BaseModuleActivity
FirstActivity
FirstActivity_1
HomeActivity
LoginAcivity
MainActivity
MainActivity2
ObjectDetectionActivity
PasswardRecoveryActivity
PrePostProcessor.java
PrePostProcessor
Result
RegisterActivity
ResultView
User
java (generated)
org.pytorch.demo.objectdetection
org.pytorch.demo.objectdetection.test
assets
res
Gradle Scripts
build.gradle (Project: ObjectDetection)
build.gradle (Module: ObjectDetection.app)
gradle.properties (Global Properties)
gradle-wrapper.properties (Gradle Version)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
super.onDraw(canvas);
if (mResults == null) return;
for (Result result : mResults) {
mPaintRectangle.setStrokeWidth(6);
mPaintRectangle.setStyle(Paint.Style.STROKE);
canvas.drawRect(result.rect, mPaintRectangle);
Path mPath = new Path();
RectF mRectF = new RectF(result.rect.left, result.rect.top, right: result.rect.left + TEXT_WIDTH, bottom:
mPath.addRect(mRectF, Path.Direction.CW);
mPaintText.setColor(Color.MAGENTA);
canvas.drawPath(mPath, mPaintText);
mPaintText.setColor(Color.WHITE);
mPaintText.setStrokeWidth(0);
mPaintText.setStyle(Paint.Style.FILL);
mPaintText.setTextSize(50);
//canvas.drawText(String.format("%s %.2f", PrePostProcessor.mClasses[result.classIndex], result.score),
canvas.drawText(String.format("%s", PrePostProcessor.mClasses[result.classIndex]), x: result.rect.left
String X = PrePostProcessor.mClasses[result.classIndex];
speakOut(X);
}
}
Activate Windows
Go to Settings to activate Windows.
Version Control
TODO
Problems
Terminal
Logcat
App Inspection
Build
Profiler
Event Log
Layout Inspector
17:14 LF UTF-8 4 spaces

Chapter 6

Conclusion & Future Work

6.1 Conclusion

The advancement of sign language translator technology represents a significant leap toward a more inclusive and accessible society for deaf and hard of hearing individuals.

By leveraging cutting-edge machine learning algorithms, multi-model integration, and real-time processing capabilities, these translators are poised to revolutionize communication across various domains such as healthcare, education, the workplace, public services, and entertainment.

The widespread implementation of sign language will bridge the communication gap between deaf and hearing individuals, fostering greater understanding and interaction.

This technological progress ensures that deaf individuals can access critical services, participate fully in social and professional settings, and enjoy media and entertainment without barriers.

Moreover, the development of personalized and adaptive systems, augmented reality (AR) and virtual reality (VR) applications, and the incorporation of cultural and linguistic nuances into translations will enhance the user experience and effectiveness of these tools.

Ethical considerations, such as data privacy and bias reduction, will be paramount in developing these applications to ensure they serve the community fairly and securely.

6.2 Future work

• **Expanding the ASL sign vocabulary** within the application for broader communication capabilities.

• **Embedding Hardware for Enhanced Accessibility.** This might include wearable devices like gloves or sensors that can capture sign language data with greater precision and potentially improve recognition accuracy in various lighting conditions or hand positions.

• **Multilingual Support for Global Accessibility.** To broaden its reach and foster communication across international borders, future development will involve incorporating additional sign languages.

• **Integration of Augmented Reality (AR) Technology.** Imagine a user holding their phone up and seeing corresponding ASL signs displayed as virtual annotations alongside a spoken or written sentence. This immersive approach could revolutionize sign language education and communication for both deaf/mute and hearing users

Many Fields

1-Healthcare

- Doctor-Patient Communication: Facilitate clear communication between healthcare providers and deaf patients, improving the quality of medical consultations, diagnostics, and treatment.

- Emergency Services: Enhancing communication in emergency situations where quick, clear interactions can be critical.

2-Education

- Classroom integration: Assisting deaf students by providing real-time translation of lectures and classroom interactions, making mainstream education more accessible.

- Online learning: Enhancing accessibility in virtual classrooms and online courses, allowing for real-time translation of education content.

3-Social Interaction

- Social media and communication platforms: Integrating sign language translation into social media and messaging apps to facilitate seamless communication between

Deaf and hearing individuals.

- Events and Conferences: Providing translation services at public events, ensuring accessibility for Deaf attendees.

REFERENCES

[1] Kyle, Jim G., James Kyle, and Bencie Woll. Sign language: The study of deaf people and their language. Cambridge university press, 1988.

[2] Padden, Carol, and Claire Ramsey. "American Sign Language and reading ability in deaf children." Language acquisition by eye 1 (2000): 65-89.

[3] Saiful Bahri, Iffah Zulaikha, et al. "Interpretation of Bahasa Isyarat Malaysia (BIM) Using SSD-MobileNet-V2 FPNLite and COCO mAP." Information 14.6 (2023): 319.

[4] Harris, Moh, and Ali Suryaperdana Agoes. "Applying hand gesture recognition for user guide application using MediaPipe." In 2nd International Seminar of Science and Applied Technology (ISSAT 2021), pp. 101-108. Atlantis Press, 2021.

[5] Wu, Wentong, et al. "Application of local fully Convolutional Neural Network combined with YOLO v5 algorithm in small target detection of remote sensing image." PloS one 16.10 (2021): e0259283.

[6] Terven, Juan, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas." Machine Learning and Knowledge Extraction 5.4 (2023): 1680-1716.

[7] https://universe.roboflow.com/atuzim/american_sign_language-sgkct

[8] https://universe.roboflow.com/majorproject-25tao/american-sign-language-v36cz

[9] Blender 4.1 Reference Manual. https://docs.blender.org/manual/en/latest/

[10] Blender Foundation, "Game Engine - Blender Manual," Blender Foundation, [Online]. Available: https://docs.blender.org/manual/en/dev/game_engine/index.html

[11] Flavell, Lance. Beginning blender: open source 3d modeling, animation, and game design. Apress, 2011.

[12] Das Chakladar, Debashis, Pradeep Kumar, Shubham Mandal, Partha Pratim Roy, Masakazu Iwamura, and Byung-Gyu Kim. 2021. "3D Avatar Approach for Continuous Sign Movement Using Speech/Text" Applied Sciences 11, no. 8: 3439. https://doi.org/10.3390/app11083439

[13] Petkar, Tanmay, Tanay Patil, Ashwini Wadhankar, Vaishnavi Chandore, Vaishnavi Umate, and Dhanshri Hingnekar. "Real Time Sign Language Recognition System for Hearing and Speech Impaired People." Int J Res Appl Sci Eng Technol 10, no. 4 (2022): 2261-2267.

[14] Adamo-Villani, Nicoletta. "3d rendering of american sign language finger-spelling: a comparative study of two animation techniques." International journal of human and social sciences 3, no. 4 (2008): 24.

[15] Alaftekin, Melek, Ishak Pacal, and Kenan Cicek. "Real-time sign language recognition based on YOLO algorithm." Neural Computing and Applications (2024): 1-16.

[16] Patel, M. (2023). American sign language detection (Doctoral dissertation, California State University, Northridge).

[17] Halder, Arpita, and Akshit Tayade. "Real-time vernacular sign language recognition using mediapipe and machine learning." Journal homepage: www. ijrpr. com ISSN 2582 (2021): 7421.

[18] 1.Elakkiya R (2021) Machine learning based sign language recognition: a review and its research frontier. J Ambient Intell Humaniz Comput 12:7205–7224

[19] Vidhyasagar, B. S., An Sakthi Lakshmanan, M. K. Abishek, and Sivakumar Kalimuthu. "Video Captioning Based on Sign Language Using YOLOV8 Model." In IFIP International Internet of Things Conference, pp. 306-315. Cham: Springer Nature Switzerland, 2023.

[20] Lui, Michael Stephen, and Fitri Utaminingrum. "A Comparative Study of YOLOv5 models on American Sign Language Dataset." Proceedings of the 7th International Conference on Sustainable Information Engineering and Technology. 2022.

[21] Bora, Jyotishman, et al. "Real-time assamese sign language recognition using mediapipe and deep learning." Procedia Computer Science 218 (2023): 1384-1393.

[22] Shams, Mahmoud Y., et al. "Food Item Recognition and Calories Estimation Using YOLOv5." International Conference on Computer & Communication Technologies. Singapore: Springer Nature Singapore, 2023.

[23] Shams, Mahmoud Y., Omar M. Elzeki, and Hanaa Salem Marie. "Towards 3D virtual dressing room based user-friendly metaverse strategy." The Future of Metaverse in the Virtual Era and Physical World. Cham: Springer International Publishing, 2023. 27-42.