# Report Of Capstone Design Project
# “Real Time Korean Sign Language (KSL) Translator using Action Recognition”




Submitted By

Name : Nugroho Avilia Kusumaputeri
Student ID : 10211561
Advisor : 정민포

AI-컴퓨터공학과
영산대학교

[캡스톤디자인 서식 1] 팀별 작성 (1과제당 1팀)

# 캡스톤디자인 사업계획서

| | | | | | |
|---|---|---|---|---|---|
| 교 과 명 | 캡스톤디자인 III | | 학점 | 6 | 타학과 연계시 연계 학과명 | |
| 과 제 명 | Real Time Korean Sign Language (KSL) Translator using Action Recognition | | | | |
| 지도교수 **1** | 학 과 | AI컴퓨터공학과 | 성 명 | 정민포 | |
| | 연락처 | 010-2805-0969 | E-mail | minpo@ysu.ac.kr | |
| 지도교수 **2** | 학 과 | ※교과목 담당 교수님 | 성 명 | ※교과목 담당 교수님 | |
| | 연락처 | | E-mail | | |
| 참여기업 **(**필수 참여**)** | 기업명 | 울산정부산업진흥원 | 담당자명 | ※필수 기재 | |
| | 연락처 | | E-mail | | |
| | 주 소 | | | | |
| 참여기업 출연금 | | ※ 있을 경우 기재 | | | |

| | | 학 과 | 학번 | 학년 | 성 명 | 연락처 | E-mail |
|---|---|---|---|---|---|---|---|
| 과제 수행 **TEAM** | 팀장 | AI컴퓨터공학과 | 10211561 | 4 | 아빌리아 | 010-5946-1825 | 10211561@gm.ysu.ac.kr |
| | 팀원 | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| 기 간 | | 2023년 3월 6일 – 2023년 6월 19일 | | | | | |

위와 같이 캡스톤디자인 사업계획서를 제출합니다.

첨부 : 캡스톤디자인 계획서 1부.

20 23 년 3 월 11 일

신청인(대표학생/팀장) : 아빌리아

지도교수 : 정민포

영산대학교 대학교육혁신본부장 귀하

# How to Promote Project Development

## 1. Implementation Strategy

There are 4 different phases when it comes to this methodology. This is where the requirements of the system are gathered in order for this project to be completed. The second phase is known as the design phase. The design phase consists mostly of the conceptual and architectural design of the proposed system. The third phase is the construction or implementation phase, where the development of the sign language translator is done and all the errors are tested again and fixed. The last step is known as the evaluation phase, and it is when the entire functional system is ready for usage by the user. The program is being evaluated and documented for future developments.

## 2. Stage of Promotion and methods implementation

### 가. 1단계 Identification

The first phase is called the identification phase. At this phase, the sign language translator's planning, required specifics, and all relevant information were gathered in order for this project to be completed. For example, what hardware and software would be needed to carry out this project. What are the basic and essential requirements for this project, and how vital is the image detection and camera equipment in this project.

### 나. 2단계 Design

The second phase is known as the design phase. The design phase consists mostly of the conceptual and architectural design of the proposed system, in this case the sign language translator. In this step, the sign language translator will be given a nearly exact design to follow, which is vital because if there is no design to follow, there will be no consistency in the design,and it will all go to waste.

### 다. 3단계 Construction or implementation

The third phase is the construction or implementation phase, where the development of the sign language translator is done and all the errors are tested again and fixed. The construct phase, as the name implies, is the phase of the spiral in which the system comes to life. The process of developing the system begins here. The sign language translator will be created using image

recognition and TensorFlow, which utilizes the Python programming language. These mockups are distributed to clients for feedback.

라. 4 단계 Evaluation

The last step is known as the evaluation phase, and it is when the entire functional system is ready for usage by the user. At the end of the first iteration, the customer will review the application and give feedback after testing it. After finishing the first iteration, the entire sign language translation will be handed out to the client.

# Team Composition Role

1. 학생 팀 구성 및 역할분담

| 구 분 | 성 명 | 소 속 | 역 할 |
|---|---|---|---|
| 팀 장 | 아빌리아 | 영산대학교 | Full stack |
| 팀 원 | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

2. Advisor's participation and role

   The advisor's role is to give advice and provide an objective assessment of the projects.

3. Participation and role of the person in charge of participating companies

   -

# Abstract

This paper presents a real-time Korean Sign Language (KSL) recognition system for individuals with speech and hearing impairments. The system uses the Mediapipe framework and Tensorflow to detect hand gestures and track hand movements. Object detection is used to detect the signer's face and hands, and the system then recognizes and translates the gestures into text in real-time. The proposed system has the potential to improve communication between people with speech and hearing impairments and others who do not understand sign language. Additionally, the system can be extended to recognize other sign languages with appropriate training data. The implementation of this system in real-world scenarios can lead to a more inclusive and accessible environment for people with speech and hearing impairments. The proposed system has the potential to greatly improve communication between individuals with speech and hearing impairments and the wider community, and can be extended to other sign languages with appropriate training data.

# Chapter I

## Introduction

### 1.1 Overview

Our society contains a considerable population of hearing and speech-impaired people. According to the Ministry of Health and Welfare in South Korea, the number of hearing-impaired populations As of 2016, 2.511 million people were registered as disabled and speech/hearing disabilities accounted for 11.6%. Even if it is only a small part of the population, we cannot exclude them from our society. These disabilities could occur for several reasons. Some could be born with speech and hearing impairments. In contrast, others could lose their hearing ability due to exposure to loud noises, tumors, ear malformations, genetics, infectious diseases, drug use, certain accidents, or aging. When a person is born with hearing impairments, it directly affects his speech ability as well. Therefore, people with hearing and speech impairment problems are considered a burden to society.

Sign language is so widespread that people with hearing or speech impairments are familiar with it, while others are unfamiliar with it. Since the beginning of time, sign languages have mostly been ignored. It is a vital means for individuals who are speech and hearing impaired to communicate with people who are not, as well as for people who are speech and hearing impaired to communicate with those who are. As a result, there is a significant communication gap between people with speech and hearing impairments and the rest of the general population. A human sign language interpreter is a common solution for bridging this gap. However, because the number of sign language interpreters is small in comparison to the number of deaf and mute people in South Korea, some deaf and mute people cannot afford to use a human interpreter all of the time when communicating with others.

Because of this communication barrier, normal people who can hear and speak find it difficult to interact with them. As a result, persons with impairments are turned down for jobs. They cannot access some information. They may be exempt from taking part in certain social or public services or gatherings. They must face several challenges because of their inability to communicate like normal people.

Traditional methods of communicating between deaf-mutes and hearing individuals utilizing pen and paper to write down ideas or using body language.

Using pen and paper to express a concept takes time and may be difficult. Body language may be difficult at times and may not convey the true intent. We live in a hectic environment nowadays. As a result, communication cannot be impeded. Solutions should be developed using developing technology to bridge the gap between deaf-mute people and the general public.

The urge for a sign language translator has been expressed for many decades, and with today's technology, such a system is more than possible. The most basic system that can be constructed is a picture to text system, and in a face-paced country such as South Korea, a translating system would be extremely useful in communication

## 1.2 Background of the project

The background of the project includes a few different areas where it consists of research on the area and the reason for the development. This is mainly a detailed explanation of the technology adapted in this project.

The project is called Sign Language Translator, and this project's main aim is to capture images of the sign language and translate them to alphabets. This research focuses on the development of a sign language translator application using TensorFlow and Python.

To address this issue, developers have been developing sign language recognition systems that can translate sign language into text or speech in real-time. These systems typically use computer vision techniques and machine learning algorithms to detect and recognize hand gestures and movements.

The development of a real-time KSL translator is particularly important in Korea, where KSL is recognized as a separate language and is used by approximately 60,000 people. Despite this, KSL is not widely understood by the general population, leading to communication difficulties for those with speech and hearing impairments. The proposed system has the potential to greatly improve communication and social inclusion for individuals with speech and hearing impairments in Korea and beyond.

## 1.3 Problem Statement

The development of a real-time Korean Sign Language (KSL) translator has been motivated by the need to bridge the communication gap between individuals with speech and hearing impairments and those without. Sign language is a primary means of communication for deaf and hard of hearing individuals, and is recognized as a natural language. However, sign language is not widely understood by the

general population, leading to difficulties in communication and social inclusion for those with speech and hearing impairments.

## 1.4 Project Objective

The objective of this project is to help people with speech and hearing impairments communicate with normal people. The objectives are such as:

- To develop hand signs using image detection
- To integrate API called Tensorflow and Mediapipe to achieve Image Detection

## 1.5 Project Scope

This project contains a few fundamental principles, the first of which is picture detection. Image detection is complemented by image recognition in this example, which is done with the aid of TensorFlow and programmed in Python. Image detection is utilized to detect hand signals, which are alphabets of the sign language, which are then converted to text on the screen. This notion of image recognition is critical to the project's success since, without it, there would be no project.

The hardware is also important; for this desktop application to run, the hardware must be a laptop or desktop computer with a camera or webcam. And for web-based applications or Android-based applications, the camera role is also important. In this project, the camera is incredibly important since it works with image detection. Sign language translation is impossible without a camera lens to capture the image.

Detecting and recognizing pictures, hand patterns, and eventually displaying the alphabet on the screen once it has been translated are all methods for recognizing hand signals. TensorFlow is also important in this project. In this case, it is used to process the image. Picture categorization is the process of presenting an image to a neural structure and having it output some sort of label for that image. The organization's score will be compared to a predefined class. The picture may be assigned to numerous classes or just one.

# Chapter II

## Literature Review

### 2.1 Artificial Intelligence

Artificial intelligence (AI) is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence but does not have to confine itself to methods that are biologically observable. Alan Turing's seminal work, "Computing Machinery and Intelligence," was published in 1950 and denoted the birth of the AI conversation. Stuart Russell and Peter Norvig then published Artificial Intelligence: A Modern Approach, which delved into four potential goals or definitions of AI and differentiated computer systems on the basis of rationality and thinking vs. acting. In its simplest form, AI is a field that combines computer science and robust datasets to enable problem solving.

It also includes the fields of machine learning and deep learning, two branches of artificial intelligence that aim to develop professional systems that can classify or predict events based on input data. As conversations emerge around the ethics of AI, we can begin to see the initial glimpses of the trough of disillusionment. Artificial intelligence (AI) is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence but does not have to confine itself to methods that are biologically observable. Alan Turing's seminal work, "Computing Machinery and Intelligence," was published in 1950 and denoted the birth of the AI conversation.

Stuart Russell and Peter Norvig then published Artificial Intelligence: A Modern Approach, which delved into four potential goals or definitions of AI and differentiated computer systems on the basis of rationality and thinking vs. acting. In its simplest form, AI is a field that combines computer science and robust datasets to enable problem solving. It also encompasses the sub-fields of machine learning and deep learning, which are AI algorithms that seek to create expert systems that make predictions or classifications based on input data. As conversations emerge around the ethics of AI, we can begin to see the initial glimpses of the trough of disillusionment.

### 2.1.1 Neural Network of AI

Neural networks, also known as "artificial neural networks" (ANNs) or "simulated neural networks" (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. They are composed of a node layer, an input layer, one or more hidden layers, and an output layer. Each node is connected to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer. Neural networks rely on training data to learn and improve their accuracy over time, but once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity.

As a result, tasks in speech recognition or image recognition can take minutes versus hours when compared to manual identification by human experts. Neural networks, also known as "artificial neural networks" (ANNs) or "simulated neural networks" (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. They are composed of a node layer, an input layer, one or more hidden layers, and an output layer. Each node is connected to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer.

Neural networks rely on training data to learn and improve their accuracy over time, but once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. As a result, tasks in speech recognition or image recognition can take minutes versus hours when compared to manual identification by human experts. Neural networks, also known as "artificial neural networks" (ANNs) or "simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. They are composed of a node layer, an input layer, one or more hidden layers, and an output layer. Each node is connected to another and has an associated weight and threshold.

If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer. Neural networks rely on training data to learn and improve their accuracy over time, but once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity.

As a result, tasks in speech recognition or image recognition can take minutes versus hours when compared to manual identification by human experts.

## 2.2 Deep Learning

Deep learning is a subset of machine learning, which is a neural network with three or more layers that attempts to mimic the behavior of the human brain. Deep learning algorithms are complex and use forward propagation and backpropagation to make predictions and correct for any errors. Examples of deep learning applications include law enforcement, financial services, customer service, and healthcare. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are used to detect features and patterns within an image, enabling tasks like object detection or recognition. Financial institutions use predictive analytics to drive algorithmic trading of stocks, assess business risks for loan approvals, detect fraud, and help manage credit and investment portfolios for clients.

Customer service organizations incorporate deep learning technology into their customer service processes, such as chatbots and virtual assistants. Healthcare has benefited greatly from deep learning capabilities since the digitization of hospital records and images. Image recognition applications can support medical imaging specialists and radiologists, helping them analyze and assess more images in less time. Deep learning is a subset of machine learning, which is a neural network with three or more layers that attempts to mimic the behavior of the human brain. Deep learning algorithms are complex and use forward propagation and backpropagation to make predictions and correct for any errors.

Examples of deep learning applications include law enforcement, financial services, customer service, and healthcare. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are used to detect features and patterns within an image, enabling tasks like object detection or recognition. Financial institutions use predictive analytics to drive algorithmic trading of stocks, assess business risks for loan approvals, detect fraud, and help manage credit and investment portfolios for clients. Customer service organizations incorporate deep learning technology into their customer service processes, such as chatbots and virtual assistants. Healthcare has benefited greatly from deep learning capabilities since the digitization of hospital records and images. Image recognition applications can support medical imaging specialists and radiologists, helping them analyze and assess more images in less time.

## 2.3 Object Detection

Object detection is a critical computer vision task that involves constructing computational models and procedures that incorporate one of the most essential bits of data that computers require. It serves as the foundation for many other tasks in the field, such as instance segmentation, picture captioning, and object tracking. Object detection research can be divided into two topics: general object detection and detection applications, with the former aiming to investigate methods for detecting various types of objects in a unified framework to simulate human vision and cognition, and the latter referring to detection in specific application scenarios, such as surveillance.

The rapid growth of deep learning techniques in recent years has breathed new life into object detection, resulting in impressive breakthroughs and propelling it to the forefront of science with unparalleled attention. Object detection is now commonly used in a variety of real-world applications, including autonomous driving, robot vision, and video surveillance. (Zhao, Zheng, Xu and Wu, 2019)

The writer said that because of its strong relationship with video analysis and visual understanding, object detection has gained a lot of study interest in recent years. Traditional object identification methods are built on handcrafted features and shallow, trainable structures. Their output may easily plateau by generating complicated ensembles that mix several low-level picture characteristics with high-level background information from object detectors and scene classifiers. (Zhao, Zheng, Xu and Wu, 2019

### 2.3.1 Object Detection and Deep Learning

Object detection is a computer vision technique that focuses on identifying and labeling objects within images, videos, and live footage. It is often used interchangeably with techniques such as image classification, object recognition, segmentation, and more. Object detection models are trained with a surplus of annotated visuals and output a labeled version of them with bounding boxes around each corresponding object. Deep learning vs. machine learning is the state-of-the-art approach since it is considered to be much more intuitive and does not require as much human interference. Object detection models are trained with hundreds of thousands of pieces of visual content to optimize the detection accuracy later on, on an automatic basis.

Object detection algorithms and methods are designed especially for handling the

task and are trained with hundreds of thousands of pieces of visual content to optimize the detection accuracy on an automatic basis later on. Object detection is a computer vision technique that focuses on identifying and labeling objects within images, videos, and live footage. It is often used interchangeably with techniques such as image classification, object recognition, segmentation, and more. Object detection models are trained with a surplus of annotated visuals and output a labeled version of them with bounding boxes around each corresponding object. Deep learning vs. machine learning is the state-of-the-art approach since it is considered to be much more intuitive and does not require as much human interference.

Object detection models are trained with hundreds of thousands of pieces of visual content to optimize the detection accuracy later on, on an automatic basis. Object detection algorithms and methods are designed especially for handling the task and are accurate on an automatic basis. Object detection algorithms and methods are designed especially for handling the task and are trained with hundreds of thousands of pieces of visual content to optimize the detection accuracy on an automatic basis later on.

## 2.4 Dataset

Data sets are an ordered collection of data that can be organized in the form of graphs, charts, or tables. Data sets can be written as a list of integers in a random order, a table, or with curly brackets around them. Different types of data sets are available for different types of information, such as numerical data sets, bivariate data sets, multivariate data sets, categorical data sets, and correlation data sets. EDA techniques can be used to identify the properties of data.

Machine learning datasets are important for machine learning algorithms to learn from. There are three main types of machine learning methods: supervised (learning from examples), unsupervised clustering (through clustering), and reinforcement learning (rewards). Machine learning datasets come in many different forms and can be sourced from a variety of places. A good machine learning dataset has a few key characteristics: it’s large enough to be representative, of high quality, and relevant to the task at hand. It should contain variables and features that are appropriately structured, have minimal noise, are scalable to large numbers at points, and can be easy to work with.

Datasets can be created from any data source, even if it is unstructured. A good machine learning dataset contains variables and features that are appropriately

structured, have minimal noise, are scalable to large numbers of data points, and can be easy to work with. Machine learning datasets are important for machine learning algorithms to learn from. There are three main types of machine learning methods: supervised (learning from examples), unsupervised (learning through clustering), and reinforcement learning (rewards). Machine learning datasets come in many different forms and can be sourced from a variety of places.

A good machine learning dataset has a few key characteristics: it’s large enough to be representative, of high quality, and relevant to the task at hand. It should contain variables and features that are appropriately structured, have minimal noise, be scalable to large numbers of data points, and be easy to work with. Datasets can be created from any data source, even if it is unstructured. A good machine learning dataset contains variables and features that are appropriately structured, have minimal noise, are scalable to large numbers of data points, and can be easy to work with. Machine learning datasets are important for machine learning algorithms to learn from.

2.5 Media Pipe

MediaPipe Holistic is an open-source framework designed specifically for complex perception pipelines leveraging accelerated inference. It provides a novel state-of-the-art human pose topology that unlocks novel use cases. The pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. It is a multi-stage pipeline that treats the different regions using a region appropriate image resolution. First, MediaPipe estimates the human pose with BlazePose’s pose detector and keypoint model, then derives three ROI crops for each hand (2x) and the face, and employs a re-crop model to improve the ROI.

MediaPipe Holistic uses a tracking approach to identify ROIs, which assumes that the object doesn't move significantly between frames. Pose prediction is used as an additional ROI to reduce response time and retain semantic consistency. Applications include remote gesture interfaces, full-body AR, sports analytics, and sign language recognition. To demonstrate the quality and performance of the MediaPipe Holistic, a simple remote control interface was built that runs locally in the browser and enables a compelling user interaction. It relies on accurate hand detection with subsequent gesture recognition mapped to a "trackpad" space anchored to the user’s shoulder, enabling remote control from up to 4 meters.

## 2.6 Sign Language

Sign languages (also known as signed languages) are languages that use the visual-manual modality to convey meaning, instead of spoken words. Sign languages are full-fledged natural languages with their own grammar and lexicon. Linguists consider both spoken and signed communication to be types of natural language, meaning that both emerged through an abstract, protracted aging process and evolved over time without meticulous planning. As of 2021, Indo Sign Language is the most used sign language in the world, and Ethnologue ranks it as the 151st most "spoken" language in the world. Linguists distinguish natural sign languages from other systems that are precursors to them or obtained from them, such as constructed manual codes for spoken languages, home sign, "baby sign", and signs learned by non-human primates.

Deaf communities and culture are widespread in the world, with sign languages used in urban areas and throughout a nation. Black ASL is an example of sign language variation in the Deaf community. Sign languages have been taken up by an entire local community, forming what is sometimes called a "village sign language" or "shared signing community". Many Australian Aboriginal sign languages arose in a context of extensive speech taboos. Sign language is also used as a form of alternative or augmentative communication by people who can hear but have difficulties using their voices to speak.

Increasingly, hearing schools and universities are expressing interest in incorporating sign language. In New Zealand, a NZSL curriculum was released for schools to take NZSL as an optional subject. Deaf communities and culture are widespread in the world, with sign languages used in urban areas and throughout a nation. Black ASL is an example of sign language variation in the Deaf community. Sign languages have been taken up by an entire local community, forming what is sometimes called a "village sign language" or "shared signing community."

### 2.6.1 Korean Sign Language

Korean Sign Language or KSL (Korean: 한국 수화 언어; Hanja: 韓國手話言語; RR: Hanguk Suhwa Eoneo or 한국 수어; 韓國手語; Hanguk Sueo) is a sign language used for deaf communities of South Korea under the North-South Korean border. It is often referred to simply as 수화; 手話; suhwa, which means signing in general. KSL is currently one of two official languages in South Korea along with Korean.

The beginnings of KSL date from 1889, although standardization efforts have only begun in 2000. The first South Korean school for the Deaf was established on April 1, 1913, in Seoul, and it was renamed as the National School for the Deaf in 1945, to be later renamed the Seoul School for the Deaf in 1951.

Although the origins of KSL predate the Japanese colonial period (de jure beginning 1910), the sign language developed some features in common with Japanese Sign Language (JSL) grammar when Korea was under Japanese rule. KSL is considered part of the Japanese Sign Language family.

Korean Sign Language (KSL) is one of Korea's two official languages since receiving official recognition in late 2015 from the Ministry of Health and Welfare and the Korean Language Institute.

The designation came after the National Assembly in Seoul in December 2015 passed the Basic Act on Korean Sign Language, which has four main points and 20 articles on providing support, generating knowledge, conducting research and offering equal opportunity to all citizens whether hearing impaired or not.

According to the South Korean Ministry of Health and Welfare, there were 252,779 people with hearing impairment and 18,275 people with language disorders in South Korea as of late 2014. Recent estimated figures for the number of Deaf people in South Korea range from 180,000 to 300,000. This is approximately 0.36%–0.6% of the population of South Korea.

Like any nation, Korea needed to assist and support its deaf community since education, learning and entertainment centers, and job opportunities for the hearing impaired were limited, and in many cases nonexistent. This led to the founding of Korean Association of the Deaf to advocate for the estimated 350,000 hearing impaired people in the country.

The two main achievements of the act were legal recognition of KSL as one of the country's two official languages and the right of every citizen to have educational, recreational, cultural and religious opportunities whether hearing impaired or not.

The government consequently sought to implement the law through providing budget, training, venues, and translation and interpretation when needed.Today, Korea has universities that teach subjects entirely in KSL and are dedicated to the training of KSL teachers like Korean National University of Welfare. A master's degree in sign language is offered, and separate programs teach the acquisition and improvement of KSL from early in life.

### 2.6.2 Sign Language Translator

Hand gestures are used to transmit thoughts and feelings, as well as to reinforce facts presented in regular interactions. Sign language is a type of gesture communication and is a strong option for testing computer vision algorithms. Gesture and sign language recognition is the process of capturing and identifying gestures and turning them into semantically significant words and expressions. Gesture recognition techniques, such as Dynamic Time Warping (DTW), were first utilised in 1993. Sign language is the primary language of the deaf community, and has its own set of linguistic rules and do not verbatim translate spoken languages.

Machine translation, which focuses on developing methods for learning the mapping between two languages, is a frequent sequence, and the Encoder-Decoder Network (NMT) region was born. Encoder-Decoder Networks use an intermediate latent space, similar to the latent space in auto-encoders but for temporal sequences, to map two sequences. Sign language is a type of gesture communication and is a strong option for testing computer vision algorithms. Gesture and sign language recognition is the process of capturing and identifying gestures and turning them into semantically significant words and expressions.

Gesture recognition techniques, such as Dynamic Time Warping (DTW), were first used in 1993. Sign language is the primary language of the deaf community, and has its own set of linguistic rules and does not verbatim translate spoken languages. Machine translation, which focuses on developing methods for learning the mapping between two languages, is a frequent sequence, and the Encoder-Decoder Network (NMT) region was born. Encoder-Decoder Networks use an intermediate latent space, similar to the latent space in auto-encoders but for temporal sequences, to map two sequences.

# Chapter III

## Methodology

### 3.1 Background

The following section of the report describes the project's primary functions. Every project has its own set of needs, which may be divided into two categories: functional requirements and non-functional requirements. The major goal of this project is to create a system capable of identifying and interpreting hand signals from Korean Sign Language into text. The project methodology utilized is known as the "spiral method."

### 3.2 Methodology

There are four different phases when it comes to this methodology. This is where the requirements of the system are gathered in order for this project to be completed. The second phase is known as the design phase. The design phase consists mostly of the conceptual and architectural design of the proposed system. The third phase is the construction or implementation phase, where the development of the sign language translator is done and all the errors are tested again and fixed. The last step is known as the evaluation phase, and it is when the entire functional system is ready for usage by the user. The program is being evaluated and documented for future developments.

### 3.2 Project Phase

- 1단계 Identification

  The first phase is called the identification phase. At this phase, the sign language translator's planning, required specifics, and all relevant information were gathered in order for this project to be completed. For example, what hardware and software would be needed to carry out this project? What are the basic and essential requirements for this project, and how vital is the image detection and camera equipment in this project?

- 2단계 Design

  The second phase is known as the design phase. The design phase consists mostly of the conceptual and architectural design of the proposed system, in this case, the sign language translator. In this step, the sign language translator will be given a nearly exact design to follow, which is vital because if there is no design to follow, there would be no consistency in the design, and it will all go to waste.

- 3단계 Construction or implementation

  The third phase is the construction or implementation phase, where the development of the sign language translator is done and all the errors are tested again and fixed. The construct phase, as the name implies, is the phase of the spiral in which the system comes to life. The process of developing the system begins here. The sign language translator will be created using image recognition and TensorFlow, which utilizes the Python programming language. These mockups are distributed to clients for feedback.

- 4 단계 Evaluation

  The last step is known as the evaluation phase, and it is when the entire functional system is ready for usage by the user. At the end of the first iteration, the customer will review the application and give feedback after testing it. After finishing the first iteration, the entire sign language translation will be handed out to the client.

### 3.3 Hardware and Software Requirements

●Software Requirements :

1. Visual Studio

   This is the platform that is used to develop the project.

2. Tensorflow

   This is an API that is used to help train datasets to capture, etc. TensorFlow is an open-source AI and deep learning library developed by Google's minds. It is used for a wide range of tasks, with a particular emphasis on deep neural organization.

3. OpenCV

   To enable the usage of the language of Python in the development of this project.

4. Google Media Pipe

   Google's MediaPipe is a versatile AI arrangement structure. It is an open-source, multi-stage system that is extremely lightweight. MediaPipe includes some pre-programmed ML arrangements, for example, face location, present assessment, hand acknowledgment, object recognition, and so on

5. Korean Sign Language dataset

   A dataset's function is to hold Korean Sign Language information that can be used by a program running on the system. Data sets are also used to store information needed by applications or the operating system itself, such as source programs, macro libraries, or system variables or parameters.

6. Firebase

   Firebase is a mobile and web development platform by Google. It offers tools for real-time databases, user authentication, cloud storage, cloud functions, hosting, analytics, crash reporting, remote configuration, and more. It simplifies app development and provides scalable backend services.

7. Android Studio

   Android Studio is the official IDE for Android app development. It has a code editor, layout editor, build system, emulator, debugger, profiler, testing framework, version control integration, and SDK manager. It simplifies the development process and provides essential tools for building Android apps.

8. GitHub

   GitHub hosts Git repositories and provides developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, code review, or the use of a collection for free.

9. Google Drive

   Google Drive allows users to store files in the cloud (on Google's servers), synchronize files across devices, and share files.

●Hardware Requirements :

1. Computer

   to be able to create the project and run the project.

2. Webcam

   to detect the images in order to translate sign language.

3. GPU

   The TensorFlow API requires a very high amount of GPU in order for it to open and detect images and also to run smoothly.

**3.4 Functional Requirements User**

It should be able to show one of five or more different words from sign languages on the webcam. The user should be able to show one of five or more different words from the Korean Sign Language (KSL).

**3.5 Non-Functional Requirements**

1. Detection of Sign Language

   The five different words or more of the Korean Sign Language will be detected when the user shows them to the webcam.

2. Speed of Detection

The speed of detection solely depends on the GPU of the computer. The higher the GPU, the faster and smoother the detection rate of the sign language when the users show it to the camera.

# Chapter IV

## Result

This project has 2 results:

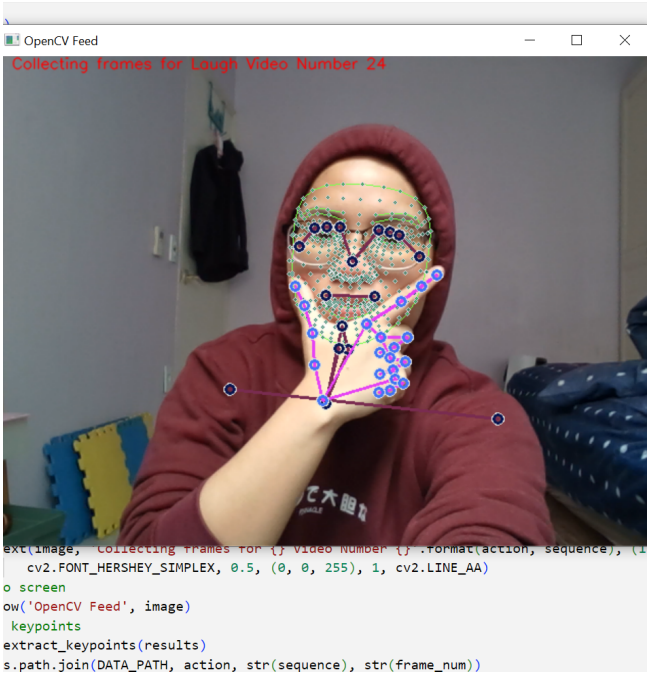
1. **Desktop Application**

   1.1 How to run the project

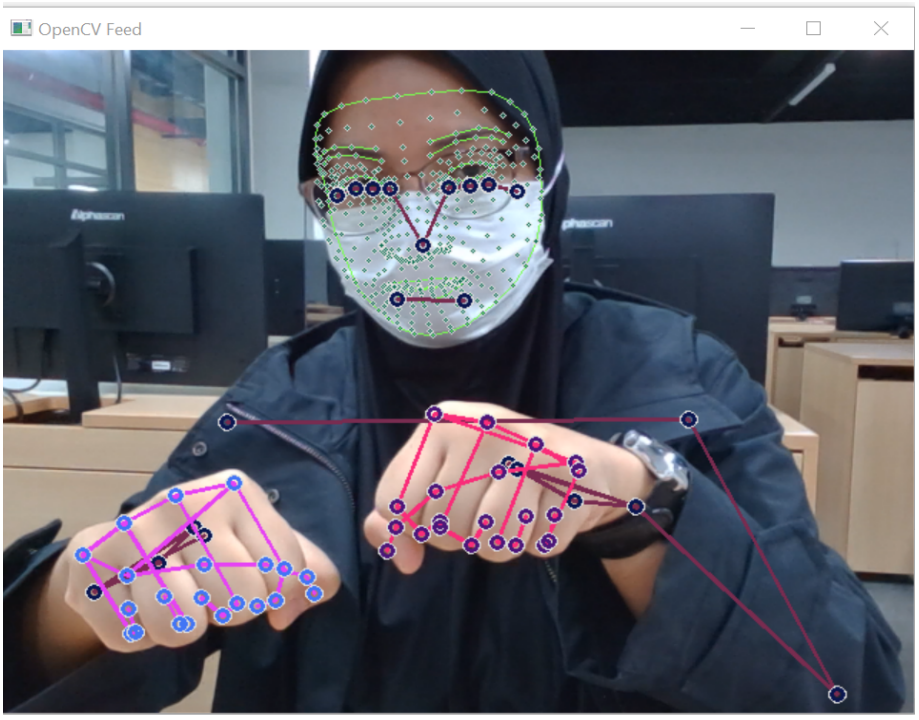
   - Open this project folder in Visual Studio or another editor.
   - Make sure that you have your virtual environment, like Anaconda Virtual Environment, etc.
   - Install all the dependencies
     - ❖ cv2: OpenCV library for computer vision tasks
     - ❖ numpy: library for numerical computations
     - ❖ os: Library for interacting with the operating system
     - ❖ matplotlib: Library for creating visualizations in Python
     - ❖ time: Library for working with time-related functions
     - ❖ mediapipe: Library for building perception pipelines
     - ❖ sklearn: Library for machine learning tasks including evaluation metrics
     - ❖ scipy: Library for scientific computing and statistics
     - ❖ tensorflow: Open-source library for machine learning
     - ❖ keras: High-level API for building and training deep learning models
     - ❖ sklearn: Library for machine learning tasks including evaluation metrics
     - ❖ tensorflow: Open-source library for machine learning
     - ❖ keras: High-level API for building and training deep learning models
   - Because it is Juptyer based project make sure run all the process except Set Up folder collection and Collect Keypoint Values for training and testing section
   - The last you can run the application in Test In Real Time Section

## 1.2 Result

It is a picture when collect the data.




It is when testing in real time




It is the result in the terminal, the result comes out wrong. I sign Hello but the result says

Nice to meet you.  Wrong results can appear because the accuracy of the dataset is only 15 percent.

```
Nice to meet you
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1 [==============================] - 0s 34ms/step
```

## 2. Android Application

2.1 How to run the project

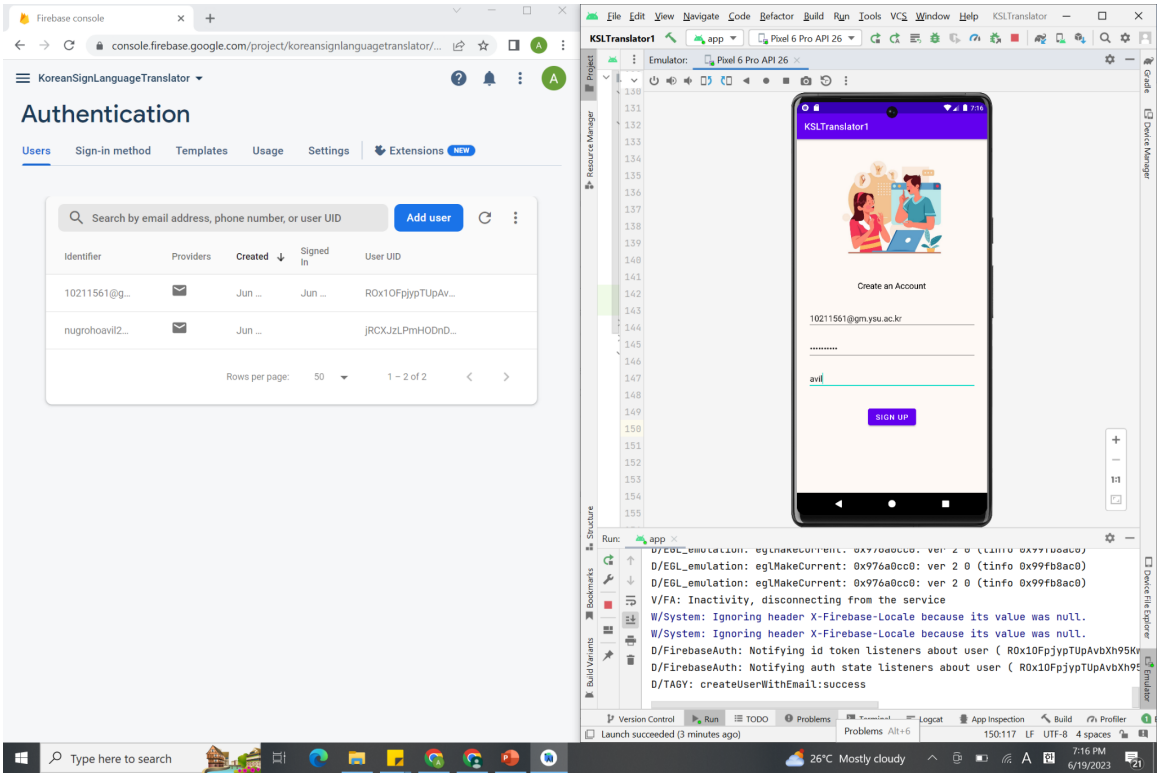
- Set up your development environment:
    - Install Android Studio: Download and install Android Studio from the official website (https://developer.android.com/studio).
    - Set up an Android device or emulator: Connect a physical Android device to your computer or set up an Android emulator through Android Studio.
- Open Android Studio: Launch Android Studio on your computer.
- Import the project:
    - Select "Open an existing Android Studio project" or "Import project" from the welcome screen.
    - Navigate to the directory where you have saved the project code and select the project's root directory. Click "OK" to import the project.
- Gradle build:
    - Once the project is opened, Android Studio will automatically start building the Gradle project.
    - Wait for the build process to finish. It may take a few moments, and you can monitor the progress in the Gradle console.
- Configure the emulator or device:
    - If you're using an Android emulator, launch it from Android Virtual Device (AVD) Manager. Ensure that the emulator is running before proceeding.
    - If you're using a physical Android device, make sure USB debugging is enabled in the device's developer options. Connect the device to your computer via USB.
- Run the application:

- In Android Studio, click the "Run" button (green triangle) on the toolbar or go to "Run" -> "Run 'app'" from the menu.
- Select the target device or emulator you want to run the application on, and click "OK".
- Android Studio will install the application on the selected device or emulator and launch it.
- Wait for the application to build and start on the device.

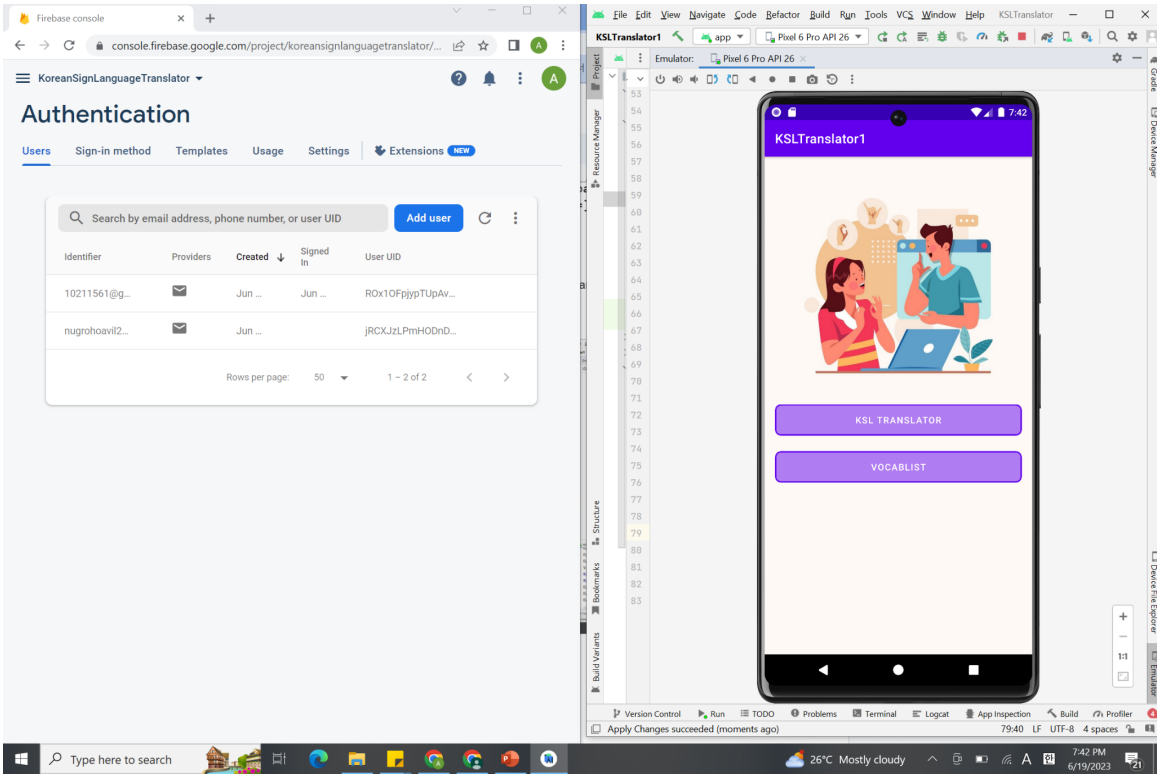
- Test the application:
  - Once the application is launched, interact with it on the emulator or device to test its functionality.
  - Verify if the screens, buttons, and features are working as expected based on the code you've implemented.

## 2.2 Result

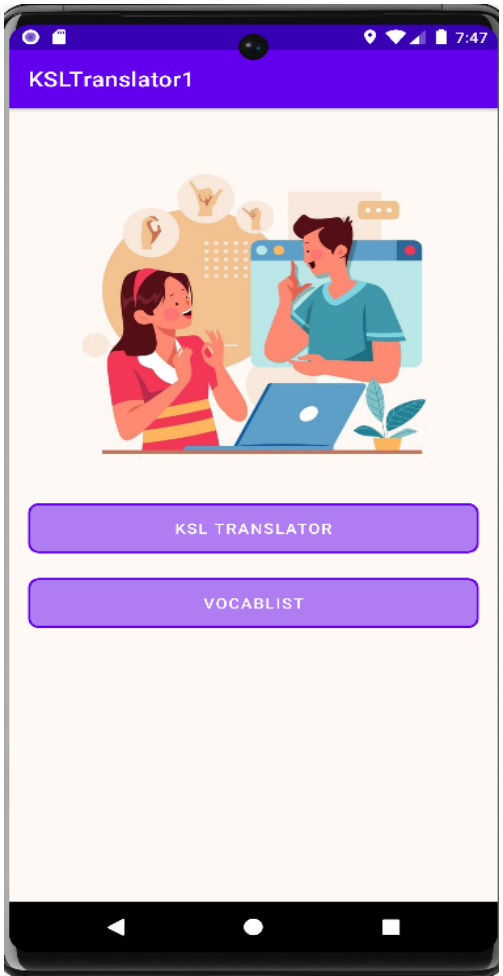
**Sign Up**



## Sign In



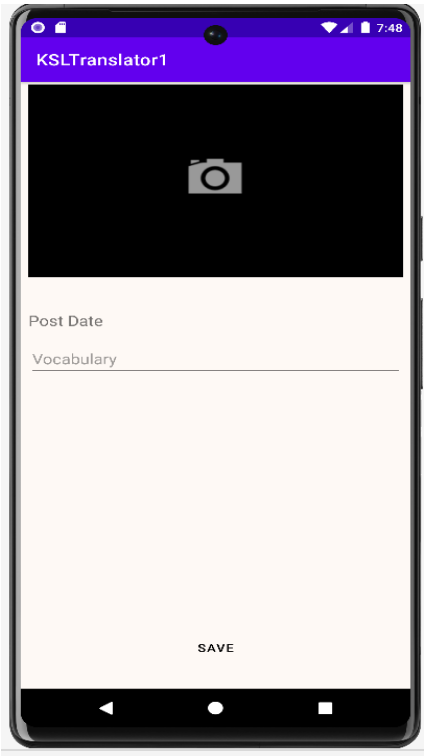

## Menu after sign in



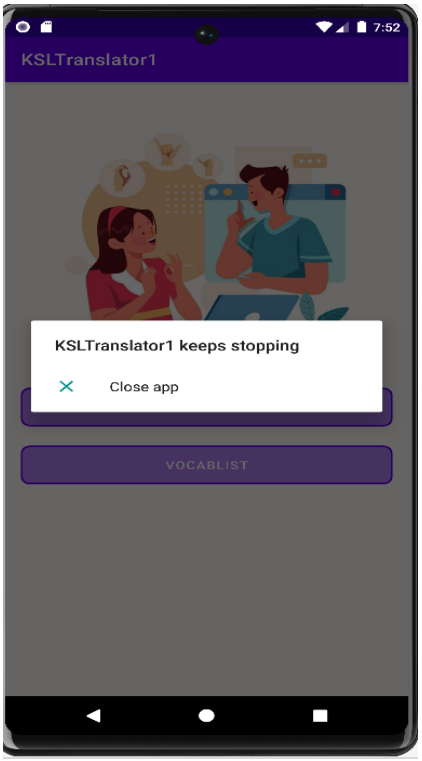

## Vocab List



**If want to add VocabList to expand Sign Language Vocabulary (still met some error with firestore database )**




**If you click KSL Translator it must be for real time translator but the code still met an error still can not use this menu**



# Chapter V

## Code Explanation

**Desktop application**

Import Dependencies

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

Installing dependencies is the process of installing all the necessary software libraries and components that a particular software application requires to function properly. Package managers can automatically download and install dependencies for a given project, saving time and reducing errors in the installation process.

Keypoints using MP Holistic

```
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
```

0.0s

```
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION
    return image, results
```

0.1s

```
def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic. FACEMESH_CONTOURS) # draw face connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # raw pose connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # draw left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # draw right hand connections
```

0.1s

This code is a script that demonstrates how to use the MediaPipe library to perform holistic human body pose detection in images.

These mp_holistic and mp_drawing lines import the MediaPipe library and initialize two variables, mp_holistic and mp_drawing, which represent the holistic model and the drawing utilities, respectively. The holistic model is a machine learning model trained to detect human body poses in images and videos, while the drawing utilities are used to draw the detected poses onto the image.

This code is a Python function called draw_landmarks that uses the MediaPipe library to draw the detected landmarks (key points) and connections for different body parts on an input image. This function uses the MediaPipe library to detect and visualize the landmarks and connections for the face, pose, and hands on an input image, which can be useful for various computer vision applications such as human body tracking, sign language recognition, and gesture-based interfaces.

```
def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_CONTOURS,
                              mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                              )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )
```

✓ 0.1s

This is a function called draw_styled_landmarks that uses the MediaPipe library to draw the detected landmarks and connections for different body parts on an input image. In contrast to the previous draw_landmarks function, this function applies different colors, thicknesses, and circle radii to the different body parts for better visualization.

Accessing Webcam

```
cap = cv2.VideoCapture(0)
# set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():
        # read feed
        ret, frame = cap.read()
        # make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)
        # draw landmarks
        draw_styled_landmarks(image, results)
        # show to screen
        cv2.imshow('OpenCV Feed', image)
        # break gracefully
        if cv2.waitKey(10) & 0xFF == ord('s'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

✓ 34.6s

This code captures video from the default camera (index 0) and initializes the Holistic model from the MediaPipe library. It then reads frames from the video capture and passes them to the mediapipe_detection() function, which applies the Holistic model to the input frame. The draw_styled_landmarks() function then draws the detected landmarks with specific styling. The output frame is then displayed to the screen using the imshow() method. The loop exits when the's' key is pressed, releasing the video capture and closing all OpenCV windows.

```
len(results.left_hand_landmarks)
```

```
results
```

✓ 0.1s

```
mediapipe.python.solution_base.SolutionOutputs
```

```
frame
```

✓ 0.0s

```
array([[[ 69,  79,  94],
        [ 71,  80,  95],
        [ 74,  82,  98],
        ...,
        [ 21,  16,  26],
        [ 21,  16,  26],
        [ 21,  16,  26]],
```

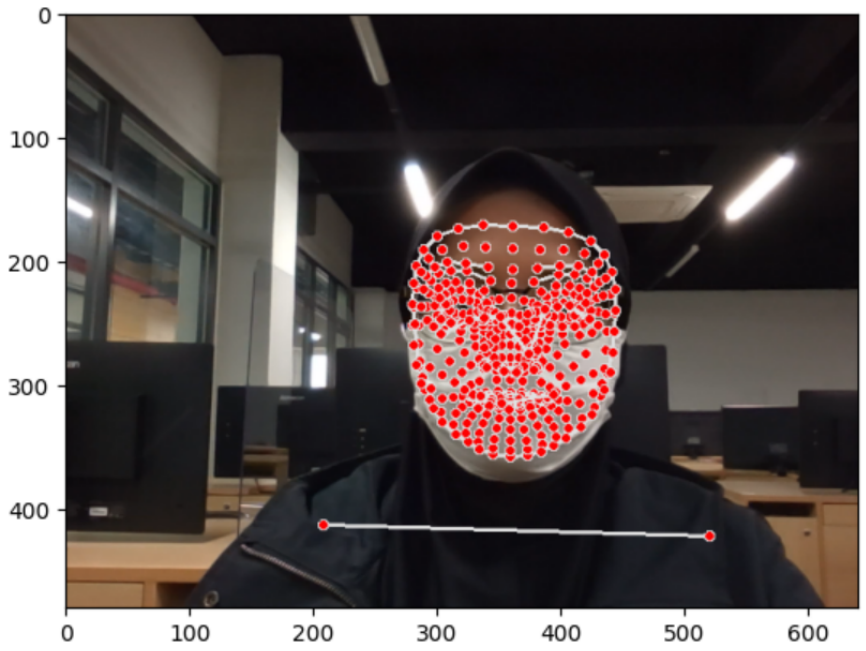
Len(results.left_hand_landmarks) returns the number of landmarks detected in the left hand in the most recent frame of video processed by the MediaPipe Holistic model, as I explained in the previous answer.

The result contains the frame by the Holistic model, including the detected landmarks and other information about the pose and facial expression of the subject in the frame. frame contains the image data for the most recent frame captured by the webcam or video source.

```
draw_landmarks(frame, results)
```
✓ 0.0s

```
plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```
✓ 0.7s

```
<matplotlib.image.AxesImage at 0x2d8ad1f9e80>
```




These two lines of code are used to display the input frame with the detected landmarks visualized on top of it, allowing the user to see how the Holistic model is performing and how the subject's body and face are being tracked over time.

```
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
```
✓ 0.0s Python

```
pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(132)
face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(1404)
lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
```
✓ 0.0s Python

The first code cell extracts the X, Y, and Z coordinates and visibility of each landmark point of the detected pose from the results object and saves them in a list called pose. A for loop is used to iterate through all the landmark points and append them to the pose list.

The second code cell creates numpy arrays containing landmark data for the pose, face, left hand, and right hand. It checks if each landmark is detected and flattens the array of coordinates into a 1-dimensional numpy array of shape. If no landmarks are detected, it creates a numpy array of zeros.

```
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
```
✓ 0.0s

The function takes input from the Holistic model and extracts keypoints for pose, face, left hand, and right hand. It then concatenates the flattened arrays into a single numpy array, which is then returned.

```
result_test = extract_keypoints(results)
```
✓ 0.0s

```
result_test
```
✓ 0.0s

```
array([ 0.5852474 ,  0.57470703, -1.59965801, ...,  0.        ,
        0.        ,  0.        ])
```

```
np.save('0', result_test)
```

```
np.load('0.npy')
```
✓ 0.0s

```
array([ 0.45947069,  1.05394256, -0.34535879, ...,  0.        ,
        0.        ,  0.        ])
```

```
extract_keypoints(results).shape
```
✓ 0.0s

```
(1662,)
```

```
468*3+33*4+21*3+21*3
```
✓ 0.0s

```
1662
```

The code defines a function extract_keypoints that extracts the 3D coordinates of the keypoints for the pose, face, left hand, and right hand landmarks and concatenates them into a single 1D numpy array. The result_test variable is assigned the output of the extract_keypoints function, and the np.save function is used to save the result_test numpy array to a file named "0.npy". The np.load function is used to load the saved numpy array from the file "0.npy". The last line of code checks the shape of the output of the extract_keypoints function and calculates the expected shape manually by summing up the total number of keypoints detected by all four models and multiplying it by the number of coordinates for each keypoint.

Setup folder for collection

```
# path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')
# actions that we try to detect
# actions = np.array(['안녕하세요', '처음 뵙겠습니다', '반가워요 ', '수고하셨습니다', '감사합니다',' 죄송합니다', '도와주세요', '부탁합니다'])
actions = np.array(['Hello', 'Nice to meet you', 'Thank you', 'You have done a good job','Sorry', 'Please Help me', 'I beg you'])
# thirty videos worth of data
no_sequences = 30
# videos are going to be 30 frames in length
sequence_length = 30
```

✓ 0.0s

```
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

The code defines several variables for the training data of a gesture recognition model, such as DATA_PATH, actions, no_sequences, and sequence_length. These variables are used to create the training data for the model.

This code creates directories for each action and sequence combination in the DATA_PATH directory. Inside the loop, os.makedirs is called with the path to create the directory for the current action and sequence. If the directory already exists, it will pass silently.

## DATASET

현재 데이터셋은 7개의 단어로 구성되어 있습니다 ⇒ 단어 더 필요하면 추가 가능

ord 쓰는것 아직 실패해끼 때문에 데이터셋 label은 영어로 바꾸는데 데이터셋 내용이 한국어 수화임 ⇒ 영어에서 한국어로 번역기 만들 예정

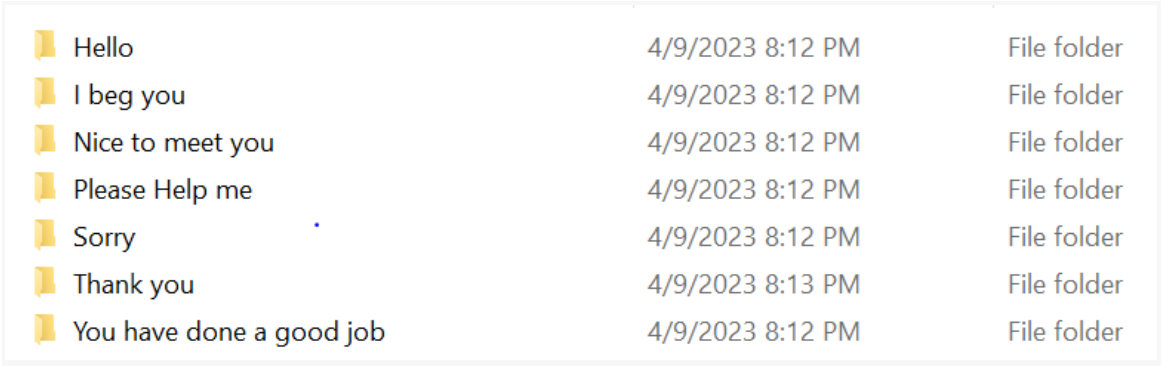
For the first 7 dictionaries for the dataset, I make a 인사말 dictionary.

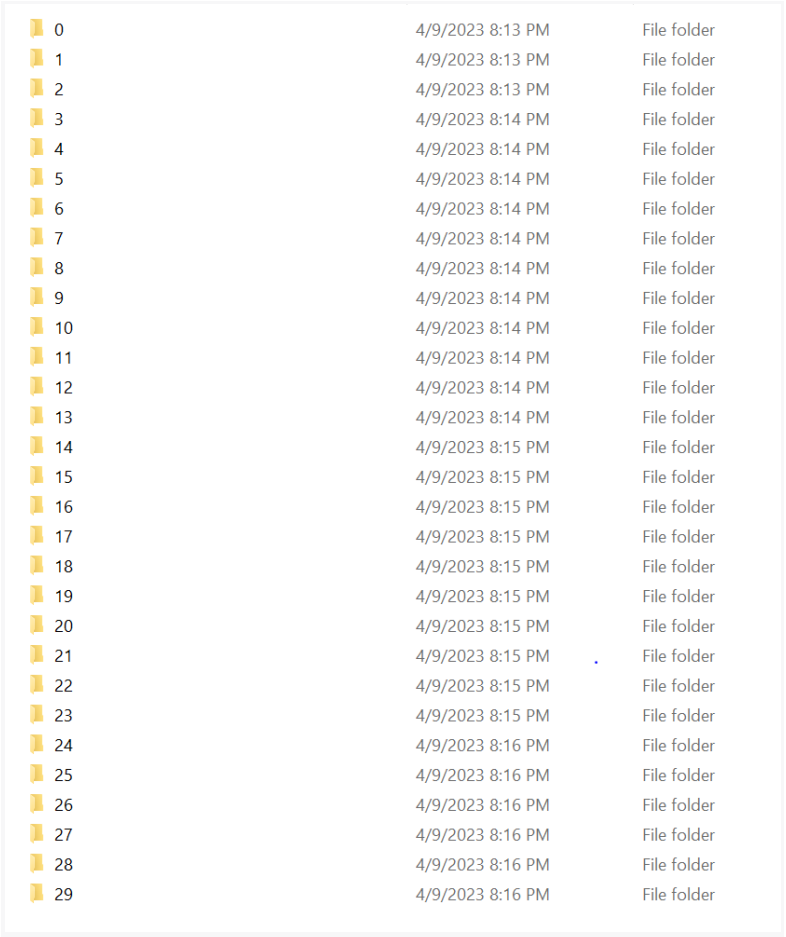
- 안녕하세요 Hello
- 처음 뵙겠습니다. nice to meet you
- 수고하셨습니다. You've done a good job.

- 감사합니다. thank you
- 죄송합니다. sorry
- 도와주세요. Please help me.
- 부탁합니다.  I beg you.

Content of dataset:

actions = np.array(['Hello', 'Nice to meet you', 'Thank you', 'You have done a good job','Sorry', 'Please Help me', 'I beg you'])



| | | | |
|---|---|---|---|
| 0.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 1.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 2.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 3.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 4.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 5.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 6.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 7.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 8.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 9.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 10.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 11.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 12.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 13.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 14.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 15.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 16.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 17.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 18.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 19.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 20.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 21.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 22.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 23.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 24.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 25.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 26.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 27.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 28.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |
| 29.npy | 4/9/2023 8:13 PM | NPY File | 14 KB |

30 frame length of video

Collect Keypoint Values for training and testing

```
cap = cv2.VideoCapture(0)
# set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    action = "I beg you" # replace "your_action" with the action you want to collect frames for
    #loop trough sequences aka video
    for sequence in range (no_sequences):
        #loop trough video length aka sequence length
        for frame_num in range (sequence_length):

            # read feed
            ret, frame = cap.read()
            # make detections
            image, results = mediapipe_detection(frame, holistic)
            print(results)
            # draw landmarks
            draw_styled_landmarks(image, results)
            #  Apply wait logic
            if frame_num == 0:
                # get current text from text array
                cv2.putText(image, 'STARTING COLLECTION', (120,200),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)
                cv2.waitKey(2000)
            else:
                cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                # show to screen
                cv2.imshow('OpenCV Feed', image)
            # NEW Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
            np.save(npy_path, keypoints)

            # break gracefully
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break

cap.release()
cv2.destroyAllWindows()
```

This code captures video feed from the default camera and applies MediaPipe Holistic model to detect body landmarks. It then draws the detected landmarks on the image and saves the flattened keypoint array as a numpy file. The code runs in a loop, collecting frames for each sequence of a particular action. It displays a message on the screen indicating the starting of the collection or the collection of frames for a specific sequence, and sets a wait time of 2 seconds for the first frame to display the start collection message. Finally, the OpenCV window is destroyed using the cv2.destroyAllWindows() function.

```
Preprocess Data and Created Labels and Features
```

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
```

✓ 0.0s

```
label_map = {label:num for num, label in enumerate(actions)}
```

✓ 0.0s

```
label_map
```

✓ 0.0s

```
{'Hello': 0,
 'Nice to meet you': 1,
 'Thank you': 2,
 'You have done a good job': 3,
 'Sorry': 4,
 'Please Help me': 5,
 'I beg you': 6}
```

```
sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

✓ 2.6s

This code reads in sequential data from a directory and creates a dataset of sequential data and their corresponding labels for action recognition. It loops through each action in a list, reads in each sequence of frames for that action, loads the corresponding numpy array files for each frame in the sequence, and appends the sequence to a list of sequences and its corresponding label to a list of labels.

```
np.array(sequences).shape
```
✓ 0.0s

```
(210, 30, 1662)
```

```
np.array(labels).shape
```
✓ 0.0s

```
(210,)
```

```
X = np.array(sequences)
```
✓ 0.0s

```
X.shape
```
✓ 0.0s

```
(210, 30, 1662)
```

```
y = to_categorical(labels).astype(int)
```
✓ 0.0s

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```
✓ 0.0s

```
y_test.shape
```
✓ 0.0s

```
(11, 7)
```

Build and Train LSTM Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
```

```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
```

```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

```
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

Building and training an LSTM neural network involves preparing sequential data, defining the architecture, compiling the model, training it using backpropagation, evaluating its performance, and fine-tuning the model. The LSTM model can then be

used to make predictions on new sequential data.

This code creates a LSTM neural network model with three LSTM layers followed by three dense layers. The model is designed for sequence-to-sequence prediction problems and uses ReLU activation function in LSTM layers and softmax activation function in the final dense layer. The number of units in the final dense layer depends on the number of possible actions.

```
▶ Launch TensorBoard Session
%load_ext tensorboard
✓ 0.1s
```

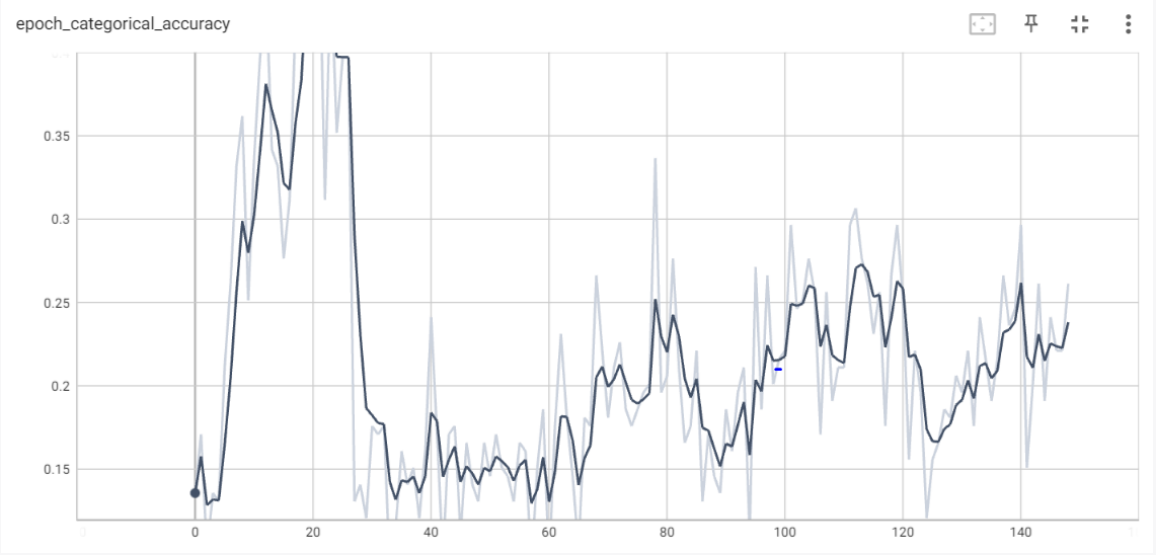
```
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
✓ 28m 18.2s
```

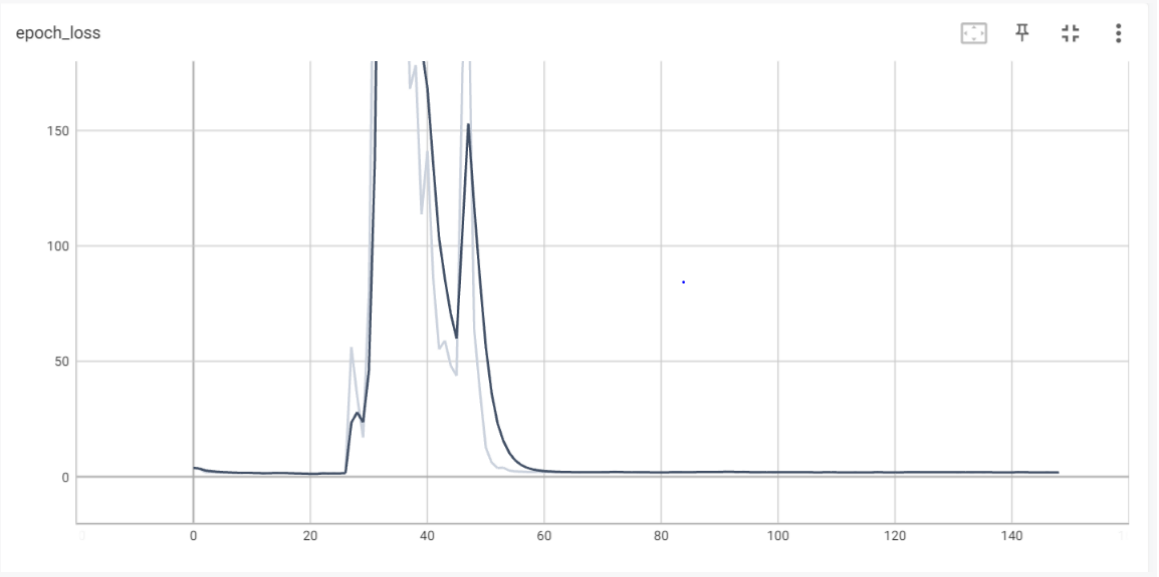
Result:

**The result has really low accuracy : 15.08%**

```
...
Epoch 1999/2000
7/7 [==============================] - 1s 94ms/step - loss: 1.9442 - categorical_accuracy: 0.1508
Epoch 2000/2000
7/7 [==============================] - 1s 95ms/step - loss: 1.9442 - categorical_accuracy: 0.1508

<keras.callbacks.History at 0x192dc967e50>
```






```
model.summary()
```

✓ 0.0s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            442112

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 7)                 231

=================================================================
Total params: 596,807
Trainable params: 596,807
Non-trainable params: 0
_________________________________________________________________
```

Make Prediction

```
res = model.predict(X_test)
```

✓ 0.4s

```
1/1 [==============================] - 0s 341ms/step
```

```
actions[np.argmax(res[4])]
```

```
actions[np.argmax(y_test[4])]
```

model.predict(X_test) is a method that takes input data X_test and returns the predicted output for each input data sample using the trained neural network model.

The result of actions[np.argmax(res[4])] and actions[np.argmax(y_test[4])] must be same action 4 == Sorry

Because my accuracy was really low so my result did not same (still need improvement on my dataset)

```
model.save('action.h5')
```

✓ 0.0s

```
WARNING:tensorflow:Compiled the loaded
```

```
del model
```

✓ 0.0s

```
model.load_weights('action.h5')
```

✓ 0.0s

Confusion matrix is a table that summarizes the performance of a classification model by comparing the predicted class labels to the true class labels. Accuracy is the ratio of correct predictions to the total number of predictions. By comparing the predicted and true class labels for a test set, we can generate a confusion matrix and calculate the accuracy to evaluate the performance of a classification model. A high accuracy and a

well-balanced confusion matrix indicate good performance.

```
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```
✓ 0.0s

```
yhat = model.predict(X_test)
```
✓ 0.3s

```
1/1 [==============================] - 0s 303ms/step
```

```
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
```
✓ 0.0s

```
multilabel_confusion_matrix(ytrue, yhat)
```
✓ 0.0s

```
array([[[ 8.,  0.],
        [ 3.,  0.]],

       [[10.,  0.],
        [ 1.,  0.]],

       [[10.,  0.],
        [ 1.,  0.]],

       [[10.,  0.],
        [ 1.,  0.]],

       [[ 0., 11.],
        [ 0.,  0.]],

       [[ 6.,  0.],
        [ 5.,  0.]]])
```

```
accuracy_score(ytrue, yhat)
```

```
# 1. New detection variables
sequence = []
sentence = []
threshold = 0.8
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():
        # Read feed
        ret, frame = cap.read()
        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)
        # Draw landmarks
        draw_styled_landmarks(image, results)
        # 2. Prediction logic
        keypoints = extract_keypoints(results)
#         sequence.insert(0,keypoints)
#         sequence = sequence[:30]
        sequence.append(keypoints)
        sequence = sequence[-30:]
        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
        #  logic
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])
            if len(sentence) > 5:
                sentence = sentence[-5:]
        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                           cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
        # Show to screen
        cv2.imshow('OpenCV Feed', image)
        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

This code performs real-time gesture recognition using Mediapipe's Holistic model and a pre-trained machine learning model. It starts a video capture object with cv2.VideoCapture(0) and sets up the Holistic model with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5). It then enters a while loop where it reads frames from the video capture object and performs gesture recognition on each frame. It first makes detections using mediapipe_detection(frame, holistic) and draws landmarks on the image. It then extracts keypoints and appends them to the sequence list.

If the confidence of the predicted action is greater than the threshold, the code adds the recognized action to the sentence list using sentence.append(actions[np.argmax(res)]). Finally, the recognized actions are displayed as text on the screen using cv2.putText() and cv2.rectangle(). The while loop continues until the user presses the 'q' key, at which point the video capture object is

**Android Application**

**build.gradle(Project:KSLTranslator1)**

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    repositories {
        // Make sure that you have the following two repositories
        google()  // Google's Maven repository

        mavenCentral()  // Maven Central repository

    }
    dependencies {
        // Add the dependency for the Google services Gradle plugin
        classpath 'com.google.gms:google-services:4.3.15'

    }
}

plugins {
    id 'com.android.application' version '7.3.1' apply false
    id 'com.android.library' version '7.3.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.20' apply false
}
```

This top-level build file sets up the dependencies and plugins for an Android project. It adds repositories for Google and Maven Central, and includes the Google services Gradle plugin as a dependency. It also applies plugins for Android application, Android library, and Kotlin support.

**build.gradle(Module:KSLTranslator1.app)**

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
    id 'com.google.gms.google-services'
    id 'kotlin-kapt'

}

android {
    namespace 'com.ysu.ksltranslator1'
    compileSdk 33

    defaultConfig {
        applicationId "com.ysu.ksltranslator1"
        minSdk 26
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
```

```
    viewBinding {
        enabled = true
    }
    buildFeatures{
        dataBinding = true
    }
}

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.9.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

    implementation 'org.tensorflow:tensorflow-lite:2.7.0'
    implementation 'org.tensorflow:tensorflow-lite-support:0.4.0'
    implementation 'androidx.camera:camera-core:1.1.0'
    implementation 'androidx.camera:camera-camera2:1.1.0'
    implementation 'androidx.camera:camera-lifecycle:1.1.0'
    implementation 'androidx.camera:camera-view:1.0.0-alpha32'


    implementation platform('com.google.firebase:firebase-bom:31.2.0')
    implementation 'com.google.firebase:firebase-analytics-ktx'
    implementation 'com.google.firebase:firebase-firestore-ktx'
    implementation 'com.google.firebase:firebase-auth-ktx'
    // Storage
    implementation 'com.google.firebase:firebase-storage-ktx'
    // Glide
    implementation 'com.github.bumptech.glide:glide:4.14.2'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.14.2'

}
```

The provided build.gradle file is the configuration file for an Android project. It includes the following key elements:

1. Plugins: Specifies the plugins used in the project, including the Android application plugin, Kotlin Android plugin, Kotlin extensions plugin, Google Services plugin, and Kotlin annotation processing plugin.
2. Android Block: Contains configurations specific to the Android project, such as the namespace, compile SDK version, defaultConfig settings (including application ID, min SDK version, target SDK version, version code, and version name), build types (e.g., release), and compatibility options for Java and Kotlin.

3. Dependencies: Lists the project dependencies, which include various AndroidX libraries for core functionality (e.g., core-ktx, appcompat, material, constraintlayout), testing libraries (e.g., JUnit, Espresso), TensorFlow Lite and TensorFlow Lite Support, CameraX libraries, Firebase libraries for analytics, Firestore, authentication, storage, and Glide library for image loading and processing.

These configurations ensure that the project is set up correctly with the required dependencies and build settings to support Android app development, Kotlin language support, testing, machine learning, camera functionality, and Firebase integration.

**MainActivity.kt**

```
package com.ysu.ksltranslator1

import ...

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var btn:Button=findViewById(R.id.get_started_button)
        btn.setOnClickListener(){ it: View!
            //Explicit Intents
            var i = Intent( packageContext: this, Login::class.java)
            startActivity(i)
        }

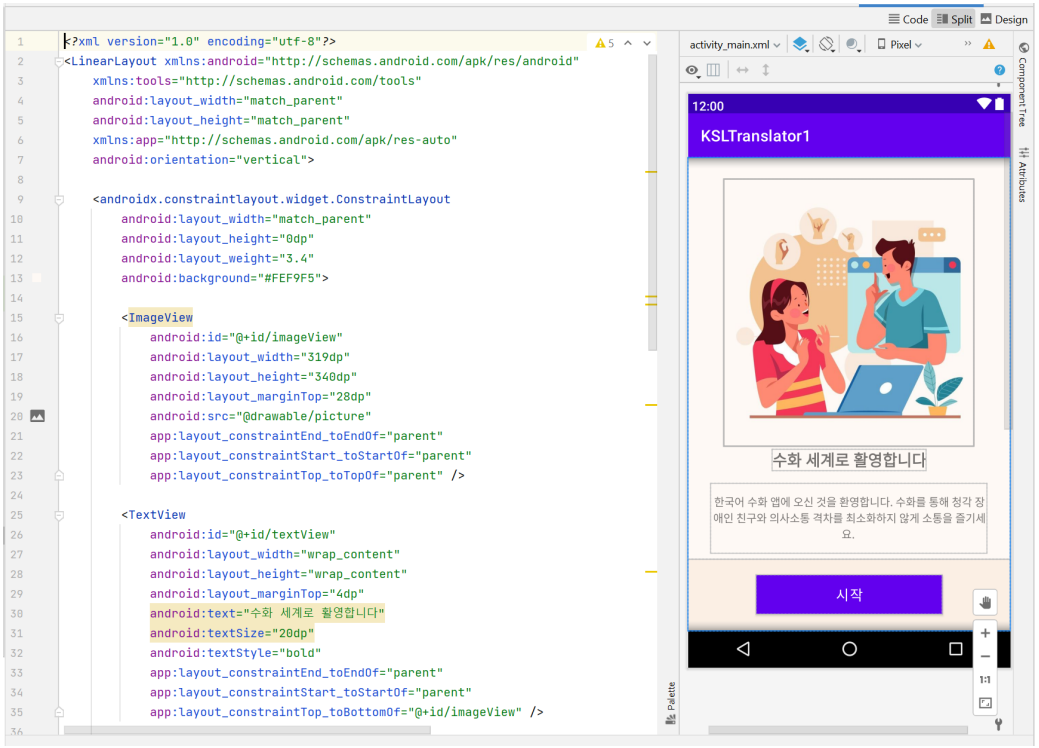
    }
}
```

The code defines the `MainActivity` class, which is the entry point of the Android application. It sets the layout for the activity using `setContentView` method, which inflates the XML layout file `activity_main.xml`.

The code then retrieves the reference to the "Get Started" button using `findViewById` method and sets a click listener on it using `setOnClickListener`. When the button is clicked, it creates an explicit `Intent` to start the `Login` activity and initiates the activity transition using `startActivity`.

## activity.main.xml

XML layout for the initial screen of the application




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="3.4"
        android:background="#FEF9F5">

        <ImageView
            android:id="@+id/imageView"
            android:layout_width="319dp"
            android:layout_height="340dp"
            android:layout_marginTop="28dp"
            android:src="@drawable/picture"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="4dp"
            android:text="수화 세계로 활영합니다"
            android:textSize="20dp"
            android:textStyle="bold"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/imageView" />
```

## Login.kt

```kotlin
package com.ysu.ksltranslator1

import ...

class Login : AppCompatActivity() {
    lateinit var binding: ActivityLoginBinding
    // Firebase Auth
    private lateinit var auth: FirebaseAuth

    // Firebase Connection

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView( activity: this,R.layout.activity_login)

        binding.createAcctBTN.setOnClickListener(){ it: View!
            val intent = Intent( packageContext: this, SignUp::class.java)
            startActivity(intent)
        }

        binding.emailSignInButton.setOnClickListener(){ it: View!
            LoginWithEmailPassword(
                binding.email.text.toString().trim(),
                binding.password.text.toString().trim()
            )
        }
        // Auth Ref
        auth = Firebase.auth


    }

    private fun LoginWithEmailPassword(email: String, password: String) {
        auth.signInWithEmailAndPassword(email,password)
            .addOnCompleteListener(this){ task ->
                if (task.isSuccessful){
                    // Sign in Success
```

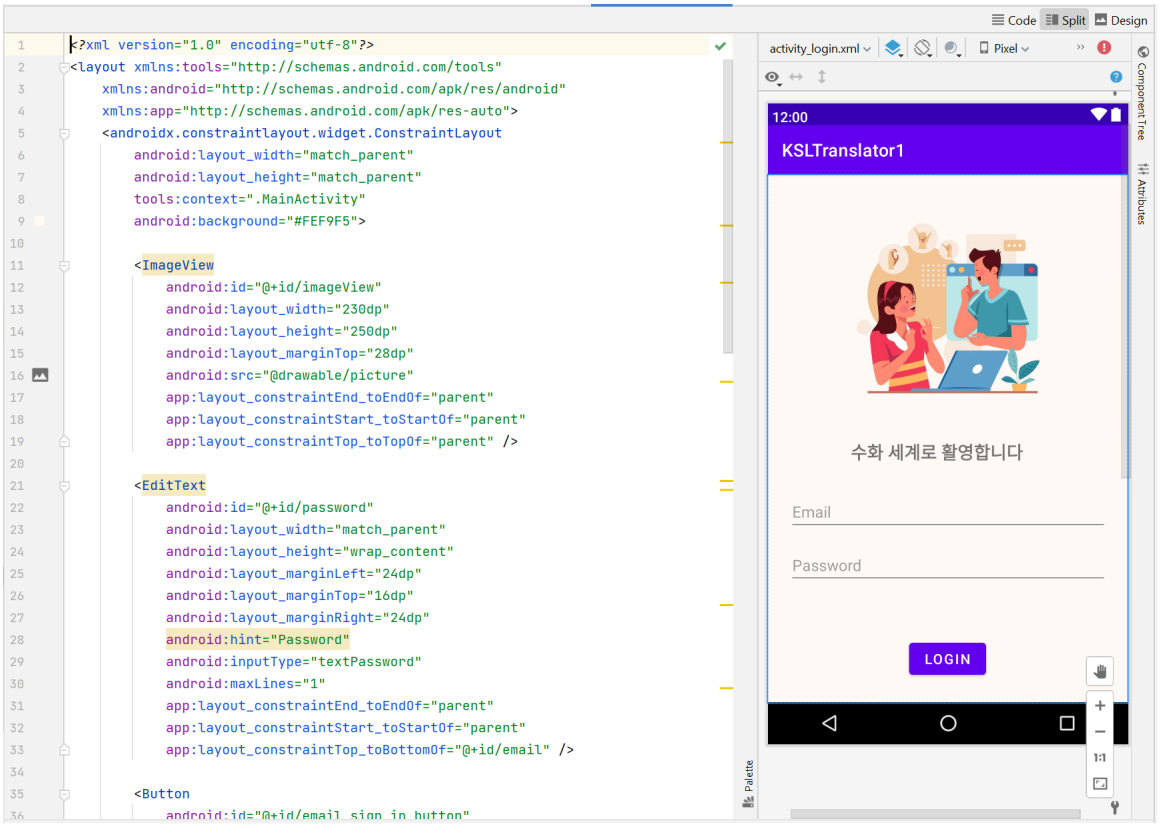
The `Login` class represents the login functionality of an Android application.

- The `onCreate` function sets the layout using Data Binding, sets click listeners for the "Create Account" and "Sign In" buttons, and initializes the Firebase Authentication instance.
- The `LoginWithEmailPassword` function handles email and password authentication using Firebase. It sets the user ID and username if authentication is successful and navigates to the `Menu` activity.
- The `onStart` function checks if a user is already signed in and navigates to the `Menu` activity .
- The `gotoMenu` function creates an intent and starts the `Menu` activity.

Overall, this code allows users to sign in with email and password, validates their credentials, and redirects them to the main menu of the application.

**activity_login.xml**

XML layout for the login of the application



**SignUp.kt**

```
package com.ysu.ksltranslator1

import ...

class SignUp : AppCompatActivity() {
    lateinit var binding: ActivitySignUpBinding

    private lateinit var auth: FirebaseAuth
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView( activity: this,R.layout.activity_sign_up)


        auth = Firebase.auth

        binding.accSignUpButton.setOnClickListener(){  it: View!
            createUser()
        }
    }

    private fun createUser() {
        val email = binding.emailCreate.text.toString()
        val password = binding.passwordCreate.text.toString()

        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    // Sign in success, update UI with the signed-in user's information
                    Log.d( tag: "TAGY",  msg: "createUserWithEmail:success")
                    val user = auth.currentUser
                    updateUI(user)
                } else {
                    // If sign in fails, display a message to the user.
                    Log.w( tag: "TAGY",  msg: "createUserWithEmail:failure", task.exception)
                    Toast.makeText(baseContext,  text: "Authentication failed.",
                        Toast.LENGTH_SHORT).show()
                    updateUI( user: null)
```

The `SignUp` class represents the sign-up functionality of an Android application.

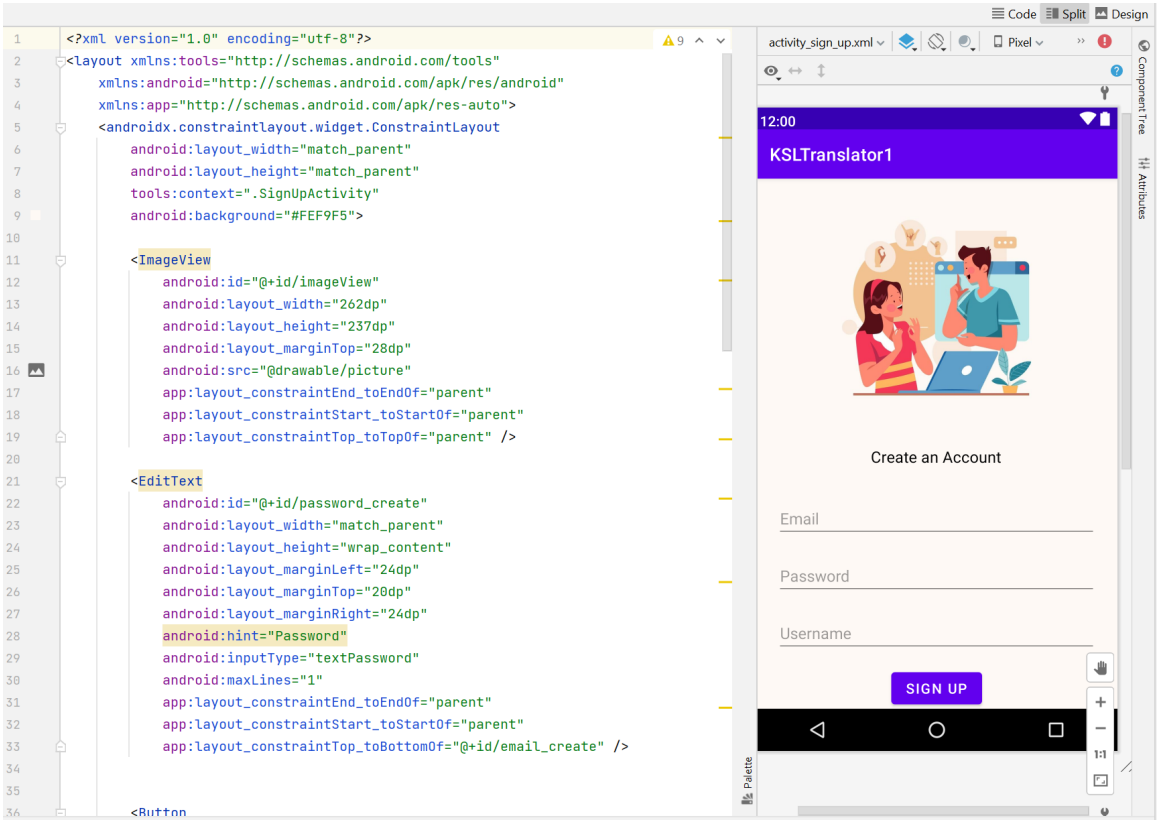
- The `onCreate` function sets the layout using Data Binding and initializes the Firebase Authentication instance.
- The `createUser` function creates a new user with the provided email and password using Firebase Authentication. It handles success and failure cases and updates the UI accordingly.
- The `updateUI` function is used to update the user interface based on the sign-up result. You can add code here to navigate to the next screen or perform any other UI-related actions.

- The `onStart` function checks if a user is already signed in and updates the UI accordingly. If a user is signed in, the `reload` function can be called to perform any necessary actions for a signed-in user.
- The `reload` function can be implemented to handle actions when the user is already signed in.

Overall, this code allows users to create a new account by providing an email and password. It utilizes Firebase Authentication to handle the sign-up process and provides basic UI update functionality.

**activity_sign_up.xml**

XML layout for the sign up of the application



**Menu.kt**

```
package com.ysu.ksltranslator1

import ...

class Menu : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_menu)

        var btn: Button =findViewById(R.id.kslTranslatorButton)
        btn.setOnClickListener(){ it: View!
            //Explicit Intents
            var i1 = Intent( packageContext: this, CameraPreview::class.java)
            startActivity(i1)
        }


        var btn1: Button =findViewById(R.id.vocabListButton)
        btn1.setOnClickListener(){ it: View!
            //Explicit Intents
            var i2 = Intent( packageContext: this, VocabList::class.java)
            startActivity(i2)
        }
    }
}
```

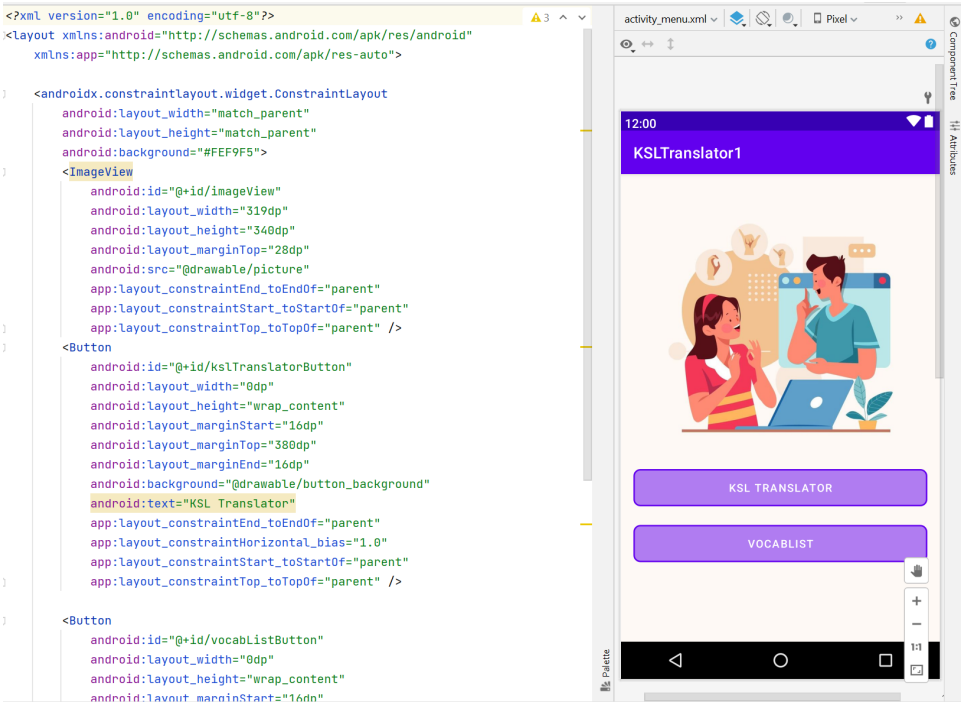
The `Menu` class represents the main menu of the Android application.

- The `onCreate` function sets the layout for the menu activity using `setContentView`.
- Two buttons, `kslTranslatorButton` and `vocabListButton`, are defined and initialized.
- Click listeners are set for each button to handle button click events.
- When the `kslTranslatorButton` is clicked, an explicit intent is created to navigate to the `CameraPreview` activity.
- When the `vocabListButton` is clicked, an explicit intent is created to navigate to the `VocabList` activity.
- The `startActivity` function is called with the appropriate intent to start the corresponding activity.

Overall, this code sets up the main menu of the application with two buttons that allow the user to navigate to different features of the app: the KSL Translator and the Vocabulary List.

**activity_menu.xml**

XML layout for the sign up of the application**.** KSL Translator button for to use the real time translator and VocabList button, if you want to add new sign vocabulary in the database(in the future development)




**KSLUser.kt**

```
package com.ysu.ksltranslator1

import android.app.Application

class KSLUser : Application() {

    var username: String? = null
    var userId: String? = null

    companion object{
        var instance: KSLUser? = null
            get() {
                if (field == null){
                    // create a new instance from journal user
                    field = KSLUser()
                }
                return field
            }
            private set
    }
}
```

The `KSLUser` class extends the `Application` class, which represents the base class for maintaining global application state.

- The class defines two variables: `username` and `userId`, which store the user's username and user ID, respectively.
- The class also defines a companion object, which provides a singleton instance of the `KSLUser` class.
- The `instance` property is a getter that returns the singleton instance of `KSLUser`. If the instance is `null`, it creates a new instance of `KSLUser`.
- The setter for the `instance` property is private, which ensures that the instance can only be set within the class.

The purpose of this class is to maintain user-related information throughout the application's lifecycle. It allows access to the user's username and ID from different parts of the app by accessing the singleton instance of `KSLUser`.

**CameraPreview.kt**

```
package com.ysu.ksltranslator1

import ...

class CameraPreview : AppCompatActivity() {
    private val REQUEST_CODE_PERMISSIONS = 10
    private val REQUIRED_PERMISSIONS = arrayOf(Manifest.permission.CAMERA)

    private lateinit var cameraExecutor: ExecutorService
    private lateinit var interpreter: Interpreter

    private lateinit var resultTextView: TextView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_camera_preview)

        resultTextView = findViewById(R.id.resultTextView)

        cameraExecutor = Executors.newSingleThreadExecutor()

        // Load the TensorFlow Lite model
        interpreter = Interpreter(loadModelFile())

        // Request camera permissions
        if (allPermissionsGranted()) {
            startCamera()
        } else {
            ActivityCompat.requestPermissions(
                activity: this, REQUIRED_PERMISSIONS, REQUEST_CODE_PERMISSIONS
            )
        }
    }

    private fun allPermissionsGranted() = REQUIRED_PERMISSIONS.all { it: String
        ContextCompat.checkSelfPermission(
            baseContext, it
        ) == PackageManager.PERMISSION_GRANTED
    }
```

The `CameraPreview` class is an `AppCompatActivity` that allows users to preview the camera feed, perform image analysis using TensorFlow Lite, and display the resulting classification label.

- The class defines constants for camera permissions and initializes variables for the camera executor and TensorFlow Lite interpreter.
- In the `onCreate` method, the layout is set, and the camera and TensorFlow Lite models are initialized.
- The `allPermissionsGranted` method checks if all the required camera permissions are granted.
- The `startCamera` method configures the camera provider, sets up the preview and image analysis use cases, and binds the camera to the lifecycle.
- The `SignLanguageAnalyzer` class implements the `ImageAnalysis.Analyzer` interface, which is responsible for image analysis.
- In the `analyze` method of `SignLanguageAnalyzer`, the captured image is converted to a `Bitmap`, and input and output buffers are created for TensorFlow Lite inference.
- Inference is performed by passing the input buffer to the TensorFlow Lite interpreter and obtaining the output buffer.
- The result is processed to obtain the predicted label, which is displayed in the `resultTextView` on the UI thread.
- The `ImageProxy.toBitmap` method converts the YUV image from the camera into a `Bitmap` format for further processing.
- The `loadModelFile` method loads the TensorFlow Lite model file into memory.
- The `getLabel` method retrieves the predicted label based on the output of the TensorFlow Lite model and a predefined set of class labels.

This class integrates camera functionality, TensorFlow Lite inference, and UI updates to enable real-time sign language classification using the device's camera.

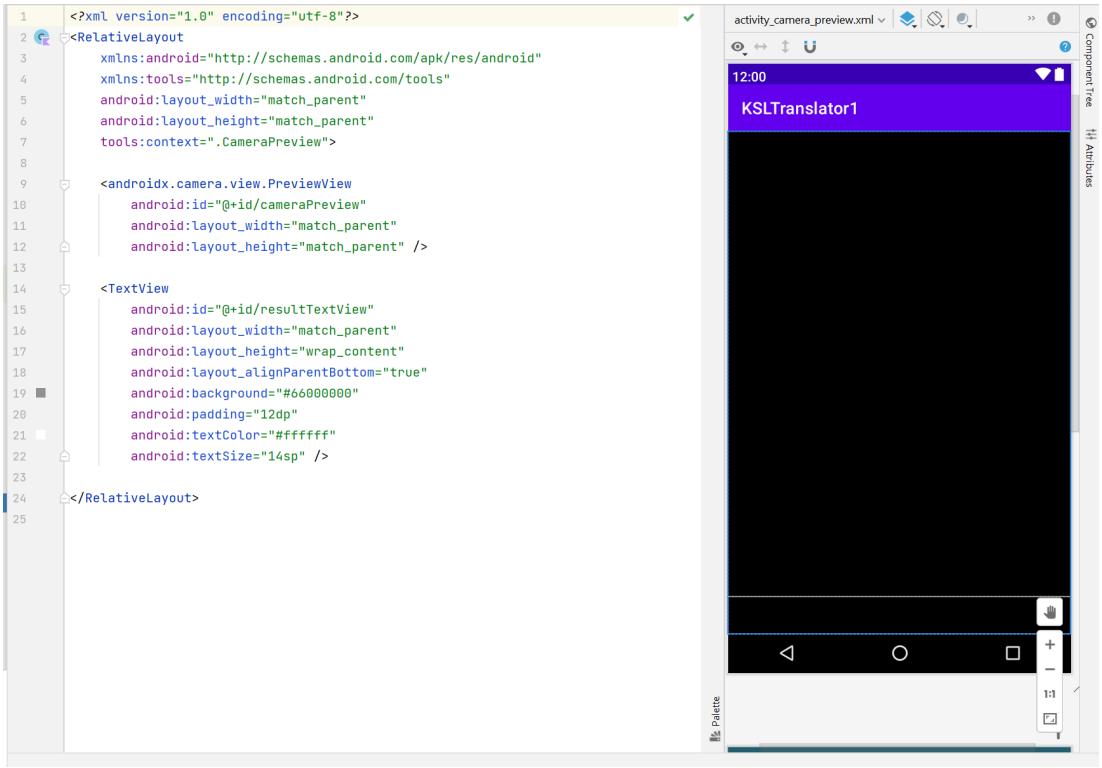
## activity_camera_preview.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CameraPreview">

    <androidx.camera.view.PreviewView
        android:id="@+id/cameraPreview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <TextView
        android:id="@+id/resultTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:background="#66000000"
        android:padding="12dp"
        android:textColor="#ffffff"
        android:textSize="14sp" />

</RelativeLayout>
```




## SignVocab.kt

```kotlin
package com.ysu.ksltranslator1

import ...

data class SignVocab(
    val vocabulary: String,
    val videoUrl: String,

    val userId: String,
    val timeAdded: Timestamp,
    var username: String
) {
    // Binding Adapter
    object DataBindingAdapter {
        @BindingAdapter("videoUrl")
        @JvmStatic
        fun setVideoByRes(videoView: VideoView, videoUrl: String) {
            Glide.with(videoView.context) RequestManager
                .load(videoUrl) RequestBuilder<Drawable!>
                .into(videoView)
        }
    }
}

private fun <TranscodeType> RequestBuilder<TranscodeType>.into(videoView: VideoView) {

}
```

The code represents a `SignVocab` data class and a binding adapter for loading a video into a `VideoView` using the Glide library.

- The `SignVocab` data class represents a sign language vocabulary item. It has properties such as `vocabulary` (the vocabulary word), `videoUrl` (the URL of the corresponding video), `userId`, `timeAdded`, and `username`.
- Inside the `SignVocab` data class, there is a nested object called `DataBindingAdapter`. This object contains a binding adapter method, `setVideoByRes`, annotated with `@BindingAdapter`.
- The `setVideoByRes` binding adapter method is responsible for loading the video from the provided URL into the `VideoView` using Glide library. It uses the `Glide.with()` method to load the video URL into the `videoView`.
- The `@BindingAdapter("videoUrl")` annotation specifies that this method should be used as a binding adapter for the `videoUrl` attribute in XML layouts.
- The private extension function `into` is empty and not being used in the provided code snippet.

Overall, this code provides a data class representing sign language vocabulary and a binding adapter to facilitate the loading of videos into a `VideoView` using Glide library.

**VocabList.kt**

```
package com.ysu.ksltranslator1

import ...


class VocabList : AppCompatActivity() {
    lateinit var binding: ActivityVocabListBinding

    // Firebase References
    lateinit var  firebaseAuth:FirebaseAuth
    lateinit var  user: FirebaseUser
    var db =  FirebaseFirestore.getInstance()
    lateinit var storageReference: StorageReference
    var collectionReference: CollectionReference = db.collection( collectionPath: "SignLanguage")


    lateinit var vocabList: MutableList<SignVocab>
    lateinit var adapter: SignVocabRecyclerAdapter

    lateinit var noPostsTextView: TextView



    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView( activity: this,R.layout.activity_vocab_list)

        // Firebase Auth
        firebaseAuth = Firebase.auth
        user = firebaseAuth.currentUser!!

        // RecyclerView
        binding.recyclerView.setHasFixedSize(true)
        binding.recyclerView.layoutManager =
            LinearLayoutManager( context: this)

        // Posts arraylist
        vocabList = arrayListOf<SignVocab>()
```

The `VocabList` class is an activity that displays a list of sign language vocabulary items.

- The class extends `AppCompatActivity` and uses data binding for the layout.
- It includes Firebase references for authentication and Firestore.
- The activity has a RecyclerView to show the vocabulary items.
- The `onCreate()` method initializes the activity and sets up the RecyclerView.
- The `onCreateOptionsMenu()` and `onOptionsItemSelected()` methods handle the menu options for adding items and signing out the user.
- The `onStart()` method retrieves the vocabulary items from Firestore based on the current user and populates the RecyclerView.

The shortened code focuses on essential functionality, Firebase integration, RecyclerView setup, and menu handling.

**SignVocabRecyclerAdapter.kt**

```
package com.ysu.ksltranslator1

import ...

class SignVocabRecyclerAdapter(val context: Context, var Vocablist:List<SignVocab>)
    : RecyclerView.Adapter<SignVocabRecyclerAdapter.MyViewHolder>()
{

    lateinit var binding:SignRowBinding

    // View Holder
    class MyViewHolder(var binding: SignRowBinding) :
        RecyclerView.ViewHolder(binding.root ) {
        fun bind(signVocab: SignVocab){
            binding.signVocab  = signVocab

        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {

        binding = SignRowBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            attachToRoot: false
        )
        return  MyViewHolder(binding)
    }

    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {

        val journal = Vocablist[position]
        holder.bind(journal)
    }

    override fun getItemCount(): Int = Vocablist.size
```

The code represents a RecyclerView adapter class called `SignVocabRecyclerAdapter`.

- The class extends `RecyclerView.Adapter` and takes a context and a list of `SignVocab` objects as parameters.
- It includes a `binding` property of type `SignRowBinding` for data binding.
- The `MyViewHolder` class is a subclass of `RecyclerView.ViewHolder` and holds the view elements for each item in the RecyclerView.
- The `bind()` function in `MyViewHolder` binds the data of a `SignVocab` object to the view elements using data binding.
- The `onCreateViewHolder()` method inflates the layout for the item view using data binding and returns a new instance of `MyViewHolder`.
- The `onBindViewHolder()` method binds the data of a `SignVocab` object at the specified position to the corresponding view holder.
- The `getItemCount()` method returns the number of items in the list.

This adapter is responsible for inflating the layout for each item in the RecyclerView and binding the data of `SignVocab` objects to the view elements.

**AddVocab.kt**

```
package com.ysu.ksltranslator1

import ...


class AddVocab : AppCompatActivity() {
    lateinit var binding: ActivityAddVocabBinding

    // Credentials
    var currentUserId: String = ""
    var currentUserName: String = ""

    // Firebase
    lateinit var auth: FirebaseAuth
    lateinit var user: FirebaseUser

    // Firebase Firestore
    var db: FirebaseFirestore = FirebaseFirestore.getInstance()
    lateinit var storageReference: StorageReference

    var collectionReference: CollectionReference = db.collection( collectionPath: "SignLanguage")
    lateinit var videoUri: Uri

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_add_vocab)

        storageReference = FirebaseStorage.getInstance().reference

        auth = Firebase.auth

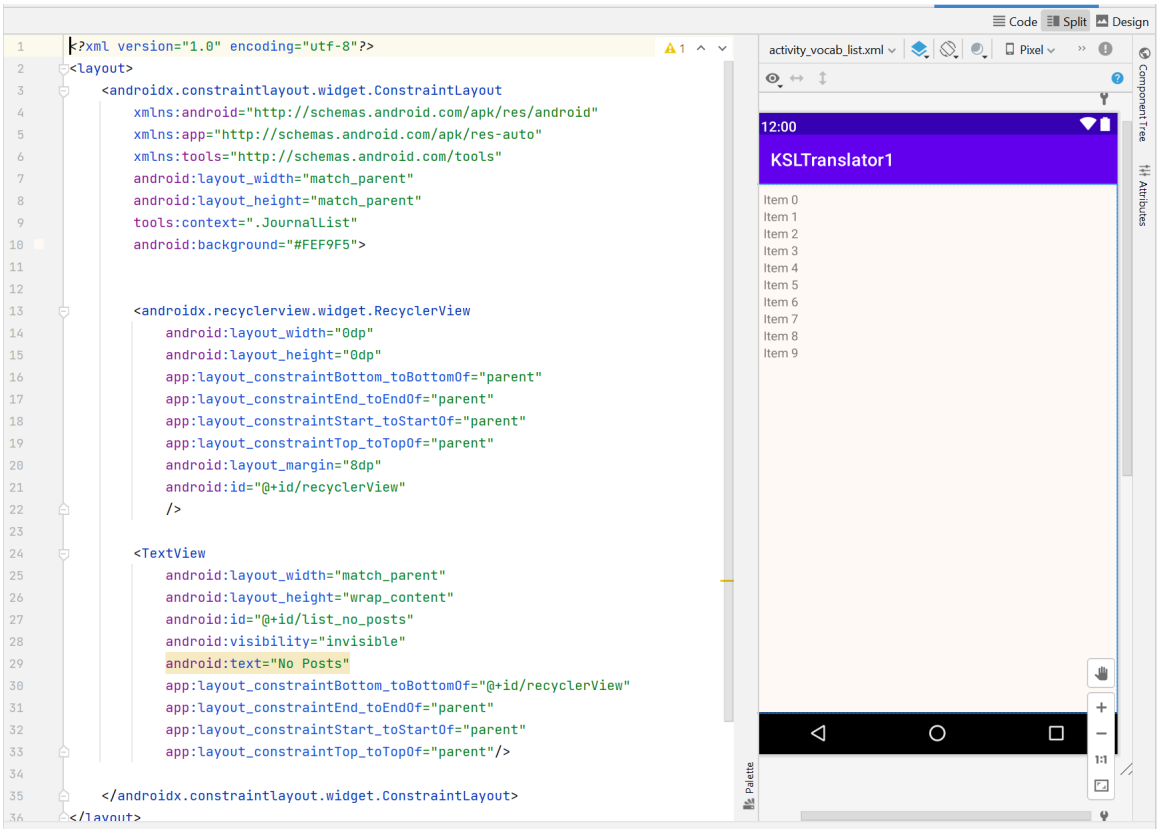
        binding.apply { this: ActivityAddVocabBinding
            postProgressBar.visibility = View.INVISIBLE
            if (KSLUser.instance != null) {
                currentUserId = auth.currentUser?.uid.toString()
                currentUserName = auth.currentUser?.displayName.toString()
                postUsernameTextview.text = currentUserName
```

The code represents an activity class called `AddVocab` that allows users to add sign language vocabulary items.

- The `AddVocab` class extends `AppCompatActivity` and contains various properties and methods.
- The `binding` property is an instance of `ActivityAddVocabBinding`, which is generated from the corresponding layout XML file.
- There are properties for storing user credentials (`currentUserId` and `currentUserName`).
- Firebase references and variables are declared for authentication (`auth`), the current user (`user`), Firestore (`db`), and storage (`storageReference`).
- The `collectionReference` represents a collection in Firestore where the vocabulary items will be stored.
- The `videoUri` property is used to store the URI of the selected video.
- The `onCreate()` method initializes the activity and sets the layout using data binding. It also sets up the Firebase references and handles button click events.
- The `saveJournal()` method is called when the "Save" button is clicked. It saves the vocabulary item to Firestore, including uploading the video file to Firebase Storage.
- The `onActivityResult()` method is called after selecting a video from the gallery. It retrieves the video URI and displays the video in a video view.
- The `onStart()` and `onStop()` methods handle the lifecycle of the activity and ensure that the user is authenticated and signed in when necessary.

This code provides the functionality to add sign language vocabulary items, including uploading videos, to a Firebase Firestore collection.

## activity_vocab_list.xml



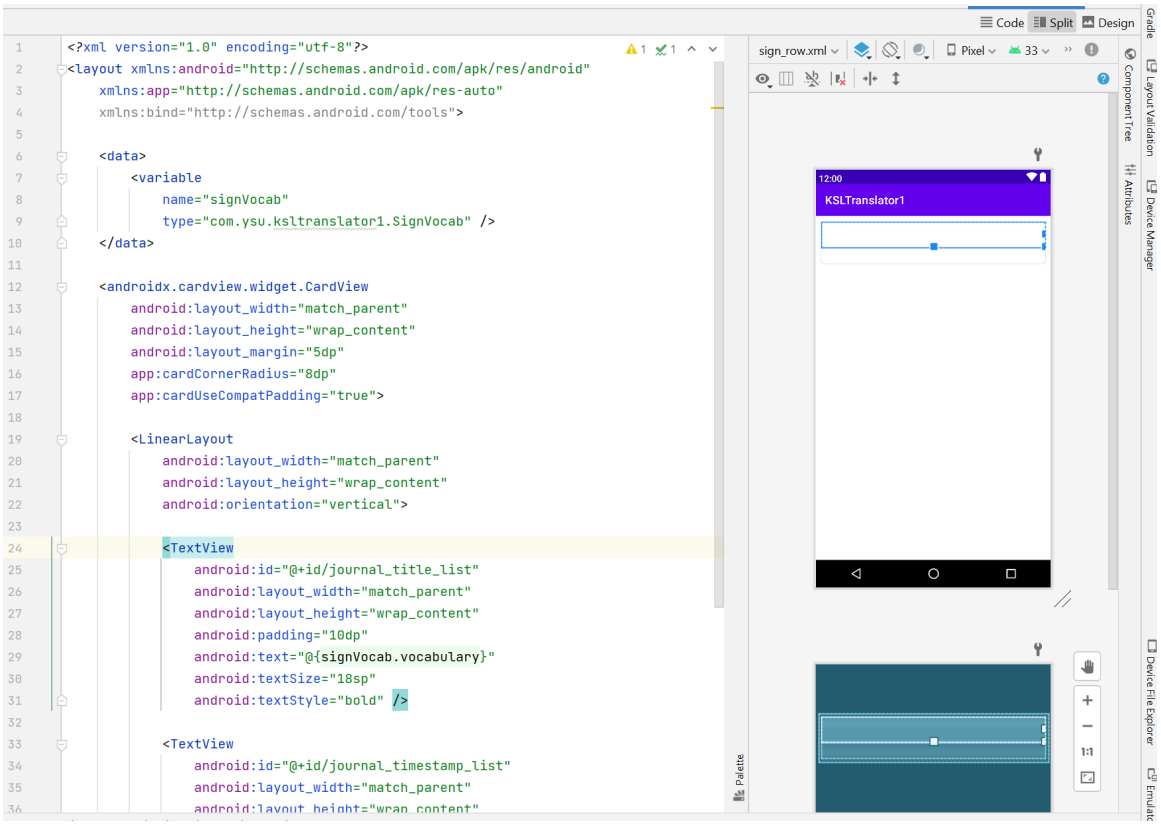
```xml
<?xml version="1.0" encoding="utf-8"?>
<layout>
    <androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".JournalList"
        android:background="#FEF9F5">


        <androidx.recyclerview.widget.RecyclerView
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            android:layout_margin="8dp"
            android:id="@+id/recyclerView"
            />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/list_no_posts"
            android:visibility="invisible"
            android:text="No Posts"
            app:layout_constraintBottom_toBottomOf="@+id/recyclerView"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"/>

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

## sign_row.xml



```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:bind="http://schemas.android.com/tools">

    <data>
        <variable
            name="signVocab"
            type="com.ysu.ksltranslator1.SignVocab" />
    </data>

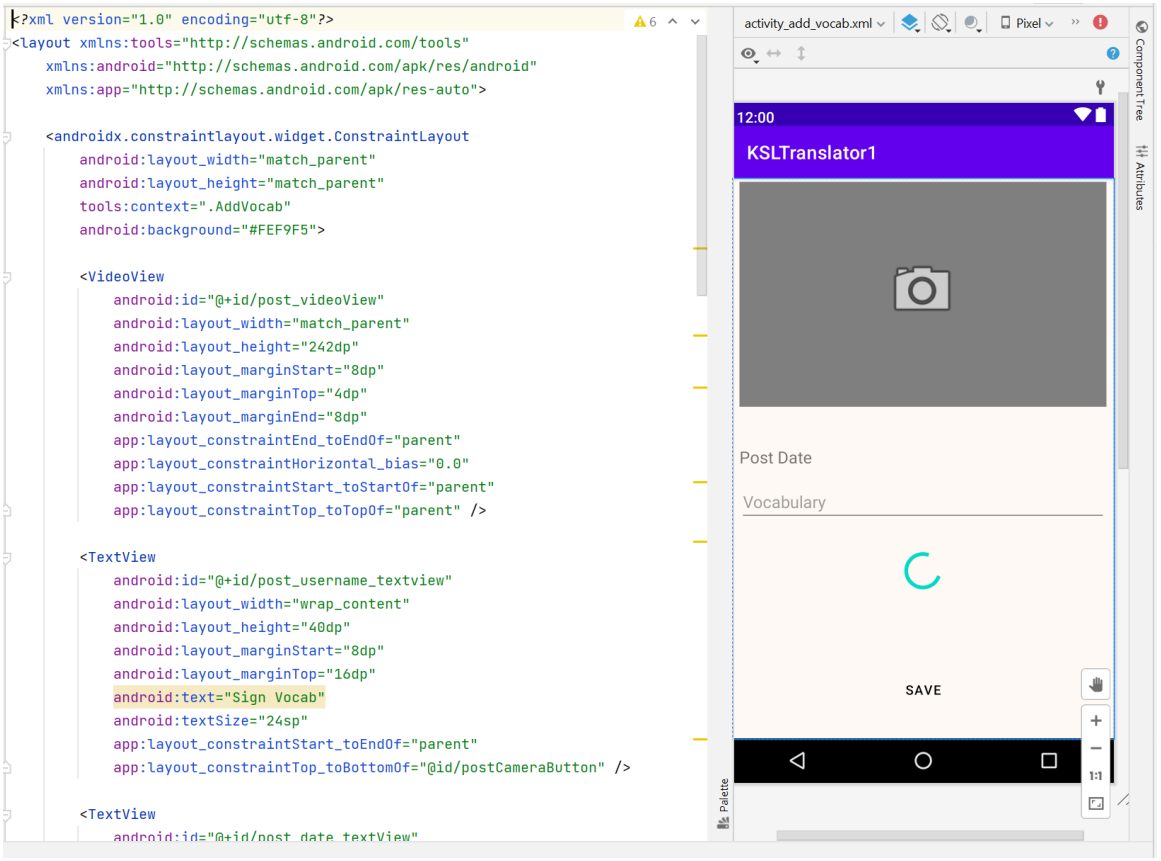
    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        app:cardCornerRadius="8dp"
        app:cardUseCompatPadding="true">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <TextView
                android:id="@+id/journal_title_list"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:padding="10dp"
                android:text="@{signVocab.vocabulary}"
                android:textSize="18sp"
                android:textStyle="bold" />

            <TextView
                android:id="@+id/journal_timestamp_list"
                android:layout_width="match_parent"
```

## activity_add.vocab.xml



```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".AddVocab"
        android:background="#FEF9F5">

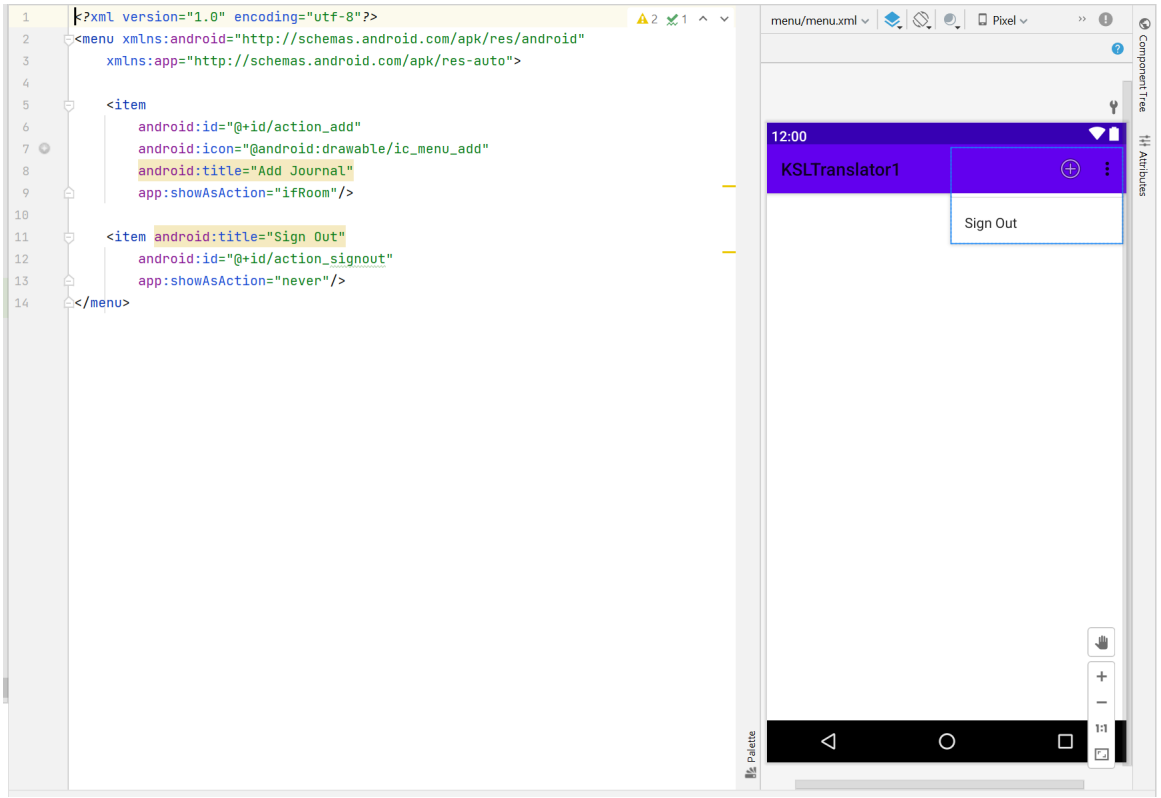
        <VideoView
            android:id="@+id/post_videoView"
            android:layout_width="match_parent"
            android:layout_height="242dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="4dp"
            android:layout_marginEnd="8dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.0"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/post_username_textview"
            android:layout_width="wrap_content"
            android:layout_height="40dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="16dp"
            android:text="Sign Vocab"
            android:textSize="24sp"
            app:layout_constraintStart_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/postCameraButton" />

        <TextView
            android:id="@+id/post_date_textView"
```



## menu.xml



```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/action_add"
        android:icon="@android:drawable/ic_menu_add"
        android:title="Add Journal"
        app:showAsAction="ifRoom"/>

    <item android:title="Sign Out"
        android:id="@+id/action_signout"
        app:showAsAction="never"/>
</menu>
```

# REFERENCES


I. *What is Artificial Intelligence (AI) ? | IBM*. (n.d.). What Is Artificial Intelligence (AI) ? | IBM. https://www.ibm.com/topics/artificial-intelligence

II. *What are Neural Networks? | IBM*. (n.d.). What Are Neural Networks? | IBM. https://www.ibm.com/topics/neural-networks

III. *Ministry of Health & Welfare : Health Statistics > Number of Registered Persons with Disabilities and Disability Pension Recipients*. (n.d.). Ministry of Health & Welfare : Health Statistics > Number of Registered Persons With Disabilities and Disability Pension Recipients. https://www.mohw.go.kr/eng/hs/hs0106.jsp?PAR_MENU_ID=1006&MENU_ID=100606

IV. *EIS Lab @ Yonsei - Publications*. (n.d.). EIS Lab @ Yonsei - Publications. https://www.eis-lab.org/publications

V. 인사말 수화 *signing time nice to meet you ( korean sign language )*. (2019, September 22). YouTube. https://www.youtube.com/watch?v=NqxbCPEwtvQ&list=PLPPJtVaQ6Kf1c8DWYmzKCYi8kU4msA-uT

VI. *Sign Language Detection using ACTION RECOGNITION with Python | LSTM Deep Learning Model*. (2021, June 18). YouTube. https://www.youtube.com/watch?v=doDUihpj6ro

VII. *What is Deep Learning? | IBM*. (n.d.). What Is Deep Learning? | IBM. https://www.ibm.com/topics/deep-learning

VIII. Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019, November). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212–3232. https://doi.org/10.1109/tnnls.2018.2876865

IX. *Introduction to object detection with deep learning | SuperAnnotate*. (2021, October 19). Introduction to Object Detection With Deep Learning | SuperAnnotate. https://www.superannotate.com/blog/object-detection-with-deep-learning

X. Koch, R. (2022, July 7). *Machine Learning Datasets - Definition, Applications, Resources*. clickworker.com. https://www.clickworker.com/customer-blog/machine-learning-datasets/

XI. *Datasets | Definition, Types, Properties and Examples*. (n.d.). BYJUS. https://byjus.com/maths/data-sets/

XII. *Sign language - Wikipedia*. (2018, March 10). Sign Language - Wikipedia. https://en.wikipedia.org/wiki/Sign_language
XIII. *Korean Sign Language - Wikipedia*. (2017, September 5). Korean Sign Language - Wikipedia. https://en.wikipedia.org/wiki/Korean_Sign_Language
XIV. Bantupalli, K. and Xie, Y., (2018). "American Sign Language Recognition using Deep Learning and Computer Vision," IEEE International Conference on Big Data
XV. *Media Pipe*. (n.d.). Github. Retrieved April 17, 2023, from https://google.github.io/mediapipe/solutions/holistic.html
XVI. *MediaPipe Holistic — Simultaneous Face, Hand and Pose Prediction, on Device*. (2020, December 10). MediaPipe Holistic — Simultaneous Face, Hand and Pose Prediction, on Device – Google AI Blog. https://ai.googleblog.com/2020/12/mediapipe-holistic-simultaneous-face.html