# KENYAN SIGN LANGUAGE RECOGNITION USING THE ENSEMBLE METHOD

**ROTICH, STANLEY KIPLAGAT**

**A PROJECT SUBMITTED TO THE DEPARTMENT OF COMPUTER AND INFORMATION TECHNOLOGY IN THE SCHOOL OF COMPUTING AND MATHEMATICS IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF DATA SCIENCE OF THE CO-OPERATIVE UNIVERSITY OF KENYA**

**NOVEMBER, 2025**

## DECLARATION

**Declaration by the candidate**

This Project is my original work and has not been presented for award of a degree in any other University or for any other award

Signature  Date 23th November, 2025

Stanley Kiplagat Rotich/MDATC01/6055/2022

**Declaration by the supervisors**

I/We confirm that the work reported in this proposal/thesis was carried out by the candidate under our supervision and has been submitted with our approval as university supervisors

Signature  Date 23th November, 2025

Dr. David Muriuki, PhD. (CUK)

Department of Mathematics Science, school Computing and Mathematics and CUK

Signature  Date 23rd November, 2025

Dr. Andrew Kipkebut, PhD.

Department of Computer Science and information technology, school Computing and Mathematics and Kabarak University)

## DEDICATION

This project is dedicated to my wife, Fridah, and my children, Bill, Cindy, and Abigail, for their enduring support, patience, cooperation, and prayers throughout my academic journey.

**ACKNOWLEGEMENT**

I express heartfelt gratitude to Almighty God for providing protection and sustenance and showing mercy throughout my academic journey. Dr. David Muriuki and Dr. Andrew Kipkebut deserve my deepest gratitude for their endless support that led me to pursue this work. The timely guidance and encouragement from my supervisors made the proposal completion possible and they have established a professional influence that will guide my future work. During my study period I received essential support through prayers and encouragement from my late father Kiprotich and late mother Kimooi and my brothers Johana, Daniel, William and Alex and my sister Esther. During my study period my wife Fridah together with our son Bill and our daughters Cindy and Abigail provided me with continuous moral support and patience along with their cooperation and prayer.

,

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## ABREVIATIONS AND SYMBOLS

| | |
|---|---|
| KNAD | -Kenya National Association for the Deaf |
| KSL | -Kenya Sign Language |
| KSLR | - Kenya Sign Language Recognition |
| ISL | - Indian Sign Language |
| ASL | -American Sign Language |
| KSLT | -Kenya sign language technology |

## ABSTRACT

Communication is the process of interaction between two or more individuals. It can be done through verbal, written, gestures such as hand and head movement, facial expression, lip motion. Communication is fundamental to social inclusion, yet the Deaf community in Kenya continues to face significant barriers due to limited recognition and translation of Kenyan Sign Language (KSL). Unlike American or Indian Sign Language, KSL has unique linguistic and cultural structures that make generic recognition systems unsuitable. This study addresses these challenges by developing an ensemble machine-learning model for KSL recognition that combines the feature extraction power of Convolutional Neural Networks (CNN) with the classification robustness of k-Nearest Neighbors (KNN). To enhance diversity and reduce noise the model was preprocessed, augmented, and annotated with a curated dataset of 8,898 KSL images, which were obtained from Kaggle. CNN extracted high-level spatial features from the gesture images, and the KNN classifier used this information for similarity-based decision making. The ensemble strategy was used to combine the outputs of both models to enhance accuracy and reduce the ability to be misclassified. Precision, recall, F1-score, and confusion matrices were used to evaluate the performance on the test set. This ensemble model achieved validation accuracy of 70.32% outperforming standalone models and demonstrates improved recognition of KSL gestures and notations. The findings of this study highlight how ensemble learning can bridge communication gap between the Deaf and the hearing world in Kenya. The research not only gives a technological solution to real-time KSL recognition but also lays a foundation for future research on larger, more diverse datasets and dynamic gesture recognition. Finally, this work adds to the social inclusion of the Deaf community by facilitating them access communication tools that empower and provide them with equal access to education, healthcare, and everyday life.

# CHAPTER ONE: INTRODUCTION

## 1.0 Introduction

This chapter is structured to give a comprehensive overview of the study. It starts with a background of the study and problem statement that provide the context and rationale of the research. The chapter then presents the study objectives and research questions, highlighting the focus of the investigation. It also discusses the significance and justification of the study, followed by a description of its expected outcomes. Finally, the scope and limitations of the research are outlined, thereby setting the boundaries within which the study is conducted.

## 1.1 Background of the study

Kenyan Sign Language (KSL) is the primary mode of communication among the deaf community in Kenya. It has its linguistic structure and is not simply a signed form of Swahili or English. However, most of the hearing people in Kenya do not seem to recognize the importance of this language. The parents whose children have a disability (hearing impairment) are supposed to introduce and inculcate the use of Sign Language to the child as early as possible. This is not always the case, as most parents are unfamiliar with sign language gestures and notations. Many challenges have stood in the way of recognizing Kenyan Sign Language over the years, but now, some things are being done. Experts in Sign Language and Information Technology established the Kenyan Sign Language and Technology (KSLT) research group. This group, which is affiliated with the C4DLab based at the University of Nairobi, seeks to carry out research and develop technology that can support the use of Kenyan Sign Language and promote hearing-impaired people in society. (Lucas & Bayley, 2011)

The World Health Organization (2021) reports that hearing impairment rehabilitation services are needed by more than 5% of the global population. The World Health Organization predicts that hearing disabilities will affect more than 700 million people during the year 2050. The unaddressed hearing impairment creates substantial effects on the deaf person as well as the people who live near him. The negative effects of untreated hearing loss affect both personal and community levels and include social isolation and feelings of loneliness and experiencing stigma. The World Health Organization calculated that a lack of hearing impairment treatment creates

annual costs of $980 billion across Health Sector and Education support and lost productivity and societal expenses.

There are several sign languages used in different parts of the world. Every part of the world has different Sign Language. This variation in notation comes as a result of environmental factors, cultural influence, and day-to-day beliefs. Linguistics transforms into various linguistic expressions through both non-verbal and verbal communication methods to convey message (Lucas & Bayley, 2011).

The majority of regions have adopted sign language technologies while using visual images and braille together with other communication tools to break communication barriers. The continuous development of technology allows researchers to create better conditions for special groups through emerging technological solutions. Artificial intelligence together with machine learning and computer vision serve as adopted technologies for sign language communication according to Patel (2022).

The 2019 census report (Development Initiative, 2020) shows that 150,000 people have hearing impairment(deaf). The high number of affected individuals requires immediate and thorough attention to this matter. These people communicate through sign language as their primary method. Such languages primarily rely on visual-manual communication methods to express meaning. Sign languages constitute independent natural languages that possess both grammatical structure and vocabulary systems according to Sandler (2006). Also, technology can be utilized to enable the deaf to live fruitfully. This technology helps deaf people to integrate easily into society and also creates a sense of awareness for hearing people. One of the notable advantages of technology-based Sign Language recognition is that it fosters more interactions between the deaf and the hearing people. The implementation of cochlear implants and classroom interpretation and real-time text services demonstrate moderate effectiveness in addressing communication barriers but remains insufficient to resolve the mentioned challenge. (Crowe et al., 2017).

Wanjala (2023) used target photos of Kenyan sign language notations for his quasi-experimental study which included photo analysis. The model received training from KSL sign notations that

included various signs which ranged from health to basic daily notations for greetings and expressing emotions. The research work implemented this sign language translation model by using TensorFlow object detection model alongside OpenCV image processing framework and Python for transfer learning. The model analyzes input visual data after an organizational phase to generate text output which represents the specific sign language notations. Users who access the model can convert KSL sign notations into understandable English text-meaning. The research foundation was derived from this work.

### 1.2 Problem Statement

Sign notation is the mode used to pass message for persons with hearing impairment.
The deaf people in Kenya face significant communication barriers because of the lack of comprehensive and accessible sign language recognition systems. Currently, technological solutions that accurately interpret Kenyan Sign Language (KSL) gestures in real time for this population is scarce. Traditional methods of communication for hearing-impaired individuals rely heavily on manual interpretation and face-to-face communication, which are often limited in scalability and efficiency. Deaf people experience obstacles when they try to access education, employment, healthcare, and social connections because the general population faces communication difficulties with them. The sensor-based technology employs hand gloves to track
hand motions for sign language notation recognition. The existing sign language recognition concentrates on international languages and is therefore not specific enough to accurately identify the Kenyan sign language notations. The absence of a sound sign language recognition system specific to Kenyan Sign Language pose a big challenge to the deaf community.

The issue at hand is the requirement of a complete and precise Kenyan Sign Language recognition system that can comprehend and read aa wide range of Kenyan Sign Language gestures in real time. Such a system would promote the deaf society as it would give them a means of engaging with the rest of the people in the society, thus promoting inclusivity and accessibility to all opportunities, regardless of their hearing status.

This study aims at designing and developing an integrated machine learning model that could be applied to recognize Kenyan Sign Language. This will help enhance communication between the hearing-impaired and the rest of the population.

## 1.3 Project Objectives

### Main objectives

To develop an ensemble model for Kenyan Sign Language Recognition (KSLR) to enable the Deaf to communicate easily with others.

### Specific Objectives

i. To survey some of the models currently used in sign-language recognition.
ii. To design an ensemble machine learning model that recognizes Kenyan Sign Language.
iii. To develop an ensemble machine learning model.
iv. To evaluate the performance of the model

## 1.4 Research Questions

The research questions are:

i. What machine learning models were previously employed in Sign language Recognition?
ii. What steps are involved in designing an ensemble model for Kenyan sign language Recognition?
iii. How can an ensemble model be developed to recognize KSL accurately?
iv. What method can be employed to evaluate the performance of a machine learning model that recognizes KSL?

## 1.5 Significance of the study

Hearing-impaired persons use Kenya Sign Language to interact with everyone in the community.

Knowledge of Kenya Sign Language remains essential because it enables efficient communication

and educational access and full participation of hearing-impaired individuals in Kenyan society.

The visual-gestural properties of Kenyan Sign-Language create more difficulties than spoken language does. Traditional Kenyan Sign-Language Recognition achieves poor accuracy while

being inefficient which requires advanced technology such as machine learning for improvement. The research develops an ensemble methodology which merges Convolutional Neural Network (CNN) and K-Nearest Neighbors (KNN) to boost Kenyan Sign-Language recognition (KSLR) accuracy and robustness. CNN is very good in special feature extractions from images frames, whereas KNN performs well in classification based on similarity metrics. Combining CNN and KNN, the ensemble model utilizes the strength of the two algorithms to improve the accuracy of recognizing complex hand gestures.

### 1.6 Expected outcomes of the study

The expected learning outcomes for this research will be:

i. The hearing-impaired community will establish better communication with other populations through this approach.
ii. It will showcase the application of deep learning techniques and machine learning Algorithms in Kenyan Sign-Language Recognition.
iii. It will lay a foundation for further research by extending the work using more robust machine learning Algorithms.

### 1.7 Justification

This study will not only help the Deaf but also the rest of the people in society by increasing the accuracy of what they communicate. Kenyan Sign-Language Recognition (KSLR) uses an ensemble method that maximizes the weaknesses of the existing models to develop a robust and accurate model. The reason why KSLR is rank the best is because it ensembles two machine learning models to develop an accurate model that recognizes the sign notations. It also uses a large training dataset to improve the model's performance, hence making it reliable and efficient. This research addresses hearing impairment problems through its technology which precisely translates sign notations to a readable English text. This fosters clear message conveyance. This model reduces the errors that occur in current sign language models. The investigation establishes concrete methods to enhance solutions for Kenyan citizens who have hearing impairments.

### 1.8 Scope of the study

The research was based on Kenyan geographical location because sign language notations and

gestures for Kenyan are the same. It focused primarily on the problems faced by the hearing-impaired person when performing their day-to-day activities such as access to education, healthcare, communication barriers and propose solutions. The study focused on Kenyan Sign-Language, excluding sign language from other regions such as American Sign Language (ASL), Indian Sign Language (ISL), etc. Additionally, the research majored on Kenyan Sign-Language gestures on health, education, and general well-being. Hearing-impaired persons were the target group. The research sampled a few Kenyan Sign Language gestures and notations.

### 1.9 Limitations of the study

Some of the limitations of our study are:

1. Detecting dynamic gestures is quite challenging because sign language notation involving hand and other body movements is difficult to detect and translate accurately because of the complexities involved.
2. This is a very new area of research in Kenya; hence, relevant local literature is scarce.
3. It was very difficult to collect sufficient and diverse data for Kenyan Sign-Language notations and gestures.
4. Developing and training ensemble models required significant computation power, which limit the scope of experimentation.

# CHAPTER TWO: LITERATURE REVIEW

## 2.0 Introduction

This chapter reviews existing literature relevant to Kenyan Sign Language Recognition (KSLR) and related technologies. It begins by presenting an overview of the different approaches that have been explored in sign language recognition globally, including computer vision and sensor-based methods. The discussion then examines prior research on American Sign Language (ASL), Indian Sign Language (ISL), and other models, noting their achievements and limitations. Special attention is given to identifying gaps in these studies, particularly the absence of ensemble learning approaches tailored to the Kenyan context. The chapter concludes by outlining the conceptual framework that guides this research, emphasizing the linguistic, cultural, and technological considerations necessary for developing an effective ensemble model for KSL recognition.

## 2.1 Related work

Several researchers have suggested and explored sign language translation. These approaches include computer vision, Deep learning, and Glove-based solutions.

**Computer vision**

A computer demonstrates visual understanding similar to humans through its ability to analyze visual content. The technology enables identification of faces and objects while reading text and comprehending visual and video contexts. Artificial intelligence techniques including machine learning power computer vision to process visual data through its analytical capabilities. Deep learning enables the system to train Artificial neural Networks using extensive data to detect patterns and features through a process similar to human brain functioning. https://onlinedegrees.sandiego.edu/introduction-to-computer-vision/

Dabre & Dholay (2014) developed a marker-free visual Indian sign language recognition system through the combination of computer vision and image processing and neural network methodologies to process web camera images. The system transforms video recordings of hand

gestures into written text which the program transforms into spoken words. Bantupalli & Xie (2019) developed a vision-based software solution which converts sign language into text. The model developed by them extracts spatial and temporal features from video sequences to perform analysis.

Kartik et. al (2020) created a model that converts human sign language or hand gestures into meaningful text using Python programming language.

The research team of (Himanshu, G. et al. 2018) developed an automatic algorithm to detect specific hand gestures through movement recognition for use by deaf and dumb people. The system designers divided the hand area into segments before locating finger positions to classify gestures which enabled sign detection and tracking and recognition.

Subburaj, S. & Muruguvalli, S. (2022) reviewed SLR system methods and classification approaches to determine an optimal research method for future work. The researchers determined through their review that (Hidden Markov models) HMM-based techniques received substantial attention in previous studies. Deep learning methods utilizing convolutional neural networks gained popularity during the previous five years according to their findings.

Their study (Ankita, W., W., and Parteek K. 2020) developed a statistical sign model system to detect sign language using deep learning convolutional neural networks. The researchers examined 35,000 sign images, part of 100 static signs, collected among various users. The analysis of the results revealed that the system achieved an accuracy of 99.72% and 99.9% in colored and grayscale images, respectively.

Vijeeta P. (2022) studied using TensorFlow for object detection through transfer learning to detect sign language. Experimental analysis showed that the model was 97.87 percent accurate when handling many different types of sentences.

One of the models suggested in their research (Rama K. et al, 2024) is the model that translates ASL based on the stationary images and video series. Their model allowed easy translation of sign language gestures to textual or spoken language. Their methodology includes gathering and preprocessing a comprehensive dataset of SL signs, alphabets and other commonly used phrases. They used a state-of-the-art CNN, which was very accurate and efficient. This model could accurately determine an extensive range of signals in ASL in various backgrounds and environments.

An alternative method to classify the character of sign language was proposed by Samarth, K., & Kabir, N. (2023) in their approach involving the use of an ensembled architecture. Inception V3 and ResNet 101 gave an accuracy of 97.24% on the ASL Dataset.

The study by Peeyusa, S. et al. (2023) involved five popular image classification models of deep learning that included ResNet50, LeNet, AlexNet, VGG16, and DenseNet 121. The researchers used an accuracy metric to conduct the tests and training with each model on the MNIST data. The authors examined the ensemble learning techniques to improve the accuracy of recognition of the ASL models. The researchers selected ResNet50 alongside LeNet and AlexNet as the best-performing models before applying ensemble techniques to stacking and concatenation techniques. The study demonstrated that stacking had higher accuracy performances than concatenation. The ensemble method increases the level of accuracy by improving it to 99%.

Wanjala G. (2023) demonstrated how artificial intelligence can be used to bridge the gap between hearing-impaired individuals and the other populace by enabling the translation of sign language notations into readable forms. They did that by collecting photos of Kenyan Sign-Language notations ranging from greetings, expressing feelings, and health then training the model on this data. They sourced their dataset from Kaggle. The model achieved an accuracy of 85%.

Malhotra (2023) created a system of ASL recognition with MediaPipe Holistic and Long Short-Term Memory (LSTM) networks. Her work presented a three-step pipeline: (1) Hands, face, and pose landmark detection with MediaPipe, (2) gesture frame preprocessing and labeling, (3) classification and speech conversion. Her system was shown to be 80% real-time accurate, which shows that temporal modeling (LSTM) and powerful feature extraction (MediaPipe) work together. This project, however, did not employ ensemble approaches but instead did ASL.

**Sensor approach**

Rinki, G., Ananya S. & Ghanapriya S. (2023) developed an Indian Sign language recognition system using a forearm-worn wearable device to assist hearing-impaired persons. The researcher developed a robust ISL recognition system through ensemble Convolution Neural Network (CNN) processing of Multi-Sensor data. Soft voting decision aggregation with bagging approach enabled the creation of 10 ensemble members that achieved 94.2% accuracy while the single CNN model reached 92.5% accuracy.

The research team of Ang J. et al (2023) developed a cost-effective data glove system with multiple inertial sensors to perform accurate sign language recognition for hearing-impaired individuals. The researchers applied decision tree (DT) and support vector machine (SVM) and K-Nearest Neighbour (KNN) and random forest(RF) as machine-learning models to detect 20 sign languages. The research demonstrated that Random Forest delivered the most precise results at 97.58%.

The researchers from Yuxuan L. et al (2023) designed a CNN-based wearable system that uses flexible strain sensors and motion detection units attached to the body for identifying gestures and bodily movements. The researchers used CNN to build and train their model using 48 commonly spoken Chinese words, achieving a model accuracy of 95.8%.

Rinki G. and Arun K. (2021) developed a multi-label classification system for linguistic sign categorization which led to final sign classification. The authors reported the classification outcomes for 100 isolated signs in Indian sign language which were recorded through electromyogram and inertial measurement units from both forearms of 10 signers. The traditional tree-based categorization achieved a 6.22% error rate in classifying signs. An on-body sign-to-speech converter was shown to effectively transform hand signs from American sign language into verbal output.

According to Zhihao Z. et al. (2020) the device contains stretchable yarn sensors which work with a wireless printed circuit. The researchers evaluated 660 sign language hand gesture recognition patterns which resulted in a 98.63% accuracy rate.

Ngaruiya E. N. and Wanjiku, N. (2021) developed an Embedded Intelligent system that utilizes sensors to track finger curvatures and hand movement for sign language gesture translation into spoken language using machine learning on the edge algorithm. The researchers collected sign language gestures which they later employed the K-Nearest Neighbour (KNN) technique as the training algorithm for their model.

Table 1.1: Summary of Empirical Literature Review

| | **Method** | **Finding** | **Weakness** |
|---|---|---|---|
| Kartik et. al | Python programming | They could convert | The Kenyan Sign- |

| | | | |
|---|---|---|---|
| | language | human sign language or hand gestures into meaningful text. | Language dataset was not used; hence, the findings were not tailored to Kenyan Sign-Language Recognition. |
| Himanshu, G. et al | They used an algorithm that automatically recognizes a certain number of gestures from hand movements and recognizes the signs. | The study was able to segment the hand region, locate the fingers, and classify the gestures, hence detecting, tracking, | The Kenyan Sign-Language dataset was not used; hence, the findings were not tailored to Kenyan Sign-Language Recognition. |
| Vijeeta, P | A transfer learning algorithm built in TensorFlow object detection served to identify sign language. | Experimental sentences of various types achieved 97.87% accuracy through the model's operations. | The ensemble method was not used, and the Kenya Sign Language dataset was not used. |
| Rama K. et al | They utilized the state-of-the-art CNN, ensuring high accuracy and efficiency. | The model could correctly identify a broad spectrum of ASL signals in various backgrounds and environments. | The ensemble method was not used, and the Kenya Sign Language dataset was not used. |
| Samarth, K. & Kabir, N | Use the ensemble model of Inception V3 and ResNet 101 | It achieved an accuracy of 97.24% on the ASL Dataset | The Kenyan Sign-Language Dataset was not used. |
| Peeyusa, S. et al. | Use deep learning models for image | It found that stacking was more accurate | The Kenyan Sign-Language Dataset |

| | | | |
|---|---|---|---|
| | classification, that is ResNet50, LeNet, AlexNet, VGG16, and DenseNet 121 | than concatenation, and the ensemble technique improved the accuracy from 97% to 99%. | was not used. |
| Wanjala G. | The system implements transfer learning by using TensorFlow and Python programming language | The model performed 85%. | The ensemble method was not used. |
| Rinki, G., Ananya S. & Ghanapriya S | Ensembled Convolution Neural Network (CNN) for a robust ISL recognition using Multi-Sensor data | Ensemble 10 members of CNN resulted in a 94.2% accuracy rate, which was higher than the 92.5% accuracy from using a single CNN model. | The Kenyan Sign Language Dataset was not used. |
| Ang J. et al. | Four machine-learning models decision tree(DT), support vector machine(SVM), K-Nearest Neighbour(KNN), and random forest(RF) | It was established that RF achieved the highest accuracy of 97.58%. | The Kenyan Sign-Language Datasets were not used. |
| Yuxuan L. et al | CNN | It achieved an accuracy of 95.8%. | The Kenyan Sign-Language Dataset |

| | | | |
|---|---|---|---|
| | | | was not used. |
| Ngaruiya E. N. and Wanjiku, N. | KNN | Its Embedded Intelligent system translates sign language gestures into spoken language by tracking finger curvatures and hand movement through sensors. | The ensembled method was not used. |

### 2.2 Research gap

Although most researchers have developed sign language recognition models, we have established that none has developed an Ensemble Kenyan Sign-Language Recognition model. The need to develop this model has been necessitated by the fact that most learning institutions are currently embracing online teaching, which usually locks out hearing-impaired students. So, this model due to its accuracy will enable them to access teaching as it translates Kenyan Sign-Language gestures into readable form.

### 2.3 Conceptual framework

The conceptual framework for this study illustrates how linguistic, cultural, technological, and legal considerations converge to support the recognition and translation of Kenyan Sign Language (KSL) through an ensemble machine learning approach. It outlines the key inputs, processes, outputs, and contextual factors that shape the model, ensuring both technical effectiveness and social relevance.

**Independent Variables (Inputs):**

The framework starts with raw KSL gesture images taken as the system's input. Preprocessing of these data includes normalization, segmentation, and resizing to enhance the quality of images and consistency. The choice of algorithms Convolutional Neural Network (CNN) to extract

features and k-Nearest Neighbors (KNN) to classify the data constitute the central part of the computational input.

**Process / Mediating Mechanisms:**

The CNN transforms the gestures into feature embeddings once they have been preprocessed, and it detects spatial hierarchies of hand shapes and orientations. In order to further enhance robustness, the data augmentation methods (rotation, flipping, brightness adjustment) add diversity to the Dataset and decrease overfitting. Ensemble learning mechanism then combines CNN with KNN, by using a ensemble approach, which optimizes the strengths of deep feature learning and instance-based classification to enhance accuracy and resistance against misclassification.

**Dependent Variables (Outputs):**

The system has two outputs. First, the model correctly recognizes KSL gestures and signs for the alphabet daily used signs. Secondly, the identified gestures are translated into English readable text enhancing communication between the Deaf and hearing communities.

**Contextual Factors:**

The framework has been based on three main contextual dimensions.

i. Linguistic peculiarities of KSL: KSL, unlike American or Indian Sign Language, has its own grammar, vocabulary, and syntax, hence specific solutions are necessary.
ii. Cultural and social inclusion requirements: Deaf people in Kenya enjoy a rich cultural identity, but the community continues to experience barriers in education, healthcare, and employment. Recognition of these systems serve to fill these gaps.
iii. Legal and policy framework: Initiatives to amend the constitution and pass policies to establish KSL as an official language provide a facilitating environment for adopting such technologies in the provision of public services.

**Implementation and Evaluation:**

This will require effective implementation, which consists of training KSL interpreters, including KSL studies in school curricula, and sensitizing society through public awareness campaigns.

The system is prone to constant monitoring and evaluation, such as user feedback from the Deaf community, to ensure it is accurate, relevant, and socially inclusive.

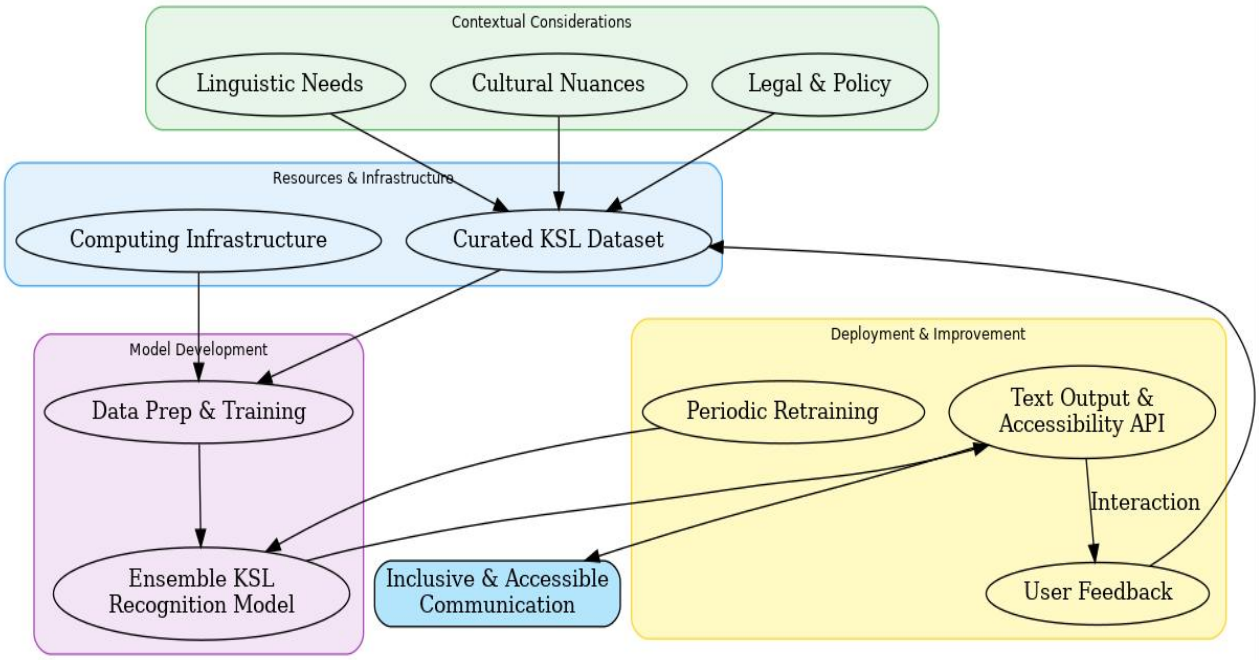



Fig 2.1: Conceptual Framework for the Kenyan Sign-Language Recognition Model

# CHAPTER THREE: METHODOLOGY

## 3.1 introduction

The chapter outlines the research methodology used to attain the study objectives. It describes the general research design, philosophy, and approach that informed the investigation while detailing the procedure used in data collection, processing, and analysis. The methodology also describes the Dataset acquisition strategy, the population and sampling, and model development emphasizing the rationale for each step. Moreover, it introduces the methods used in preprocessing, feature extraction, and evaluation to ensure rigorous and reproducible processes. By systematically presenting these methods, the chapter provides a structured foundation for the implementation of the ensemble machine learning model for Kenyan Sign Language recognition.

## 3.2 Research design and philosophy

This research aims at developing a model that translates Kenyan Sign Language gestures and notations into English readable text. It involves capturing sign language notations and gestures of deaf individuals using a webcam. A computer vision algorithm analyzes and processes these images and then translate them into corresponding English readable text, thereby enabling communication.

## 3.3 Population and Sampling

### Study site

The research relied on secondary data sourced from the Kenya Sign Language dataset available from Kaggle. Kaggle has about 8898 images with users performing KSL notations that provides data for training the models.

### Target population

The study considered Kenyan Sign-Language notation that includes some alphabets, and other notations. Also, common gestures and notations for instance me, you, mosque, and church were also considered.

### Sampling

Probability sampling techniques was used to ensure unbiased sampling which supports the research goal achievement. The study used simple random sampling of images because statistical regularity applies.

### 3.4 Dataset Acquisition

The dataset sourced from Kaggle has several visual inputs such as me, church, mosque, you, letters of alphabets etc. The images were downloaded from Kaggle. There are 8898 images from Kaggle and of 720 by 1280 pixels in dimensions which were then standardized.

### 3.5 Model Construction

Model construction entails activities involved in designing, developing, and testing the model. It highlights the steps necessary for processing the image datasets used for training and testing the model used for Kenyan Sign-Language recognition. This step is crucial since it provides an algorithm that helps in model processing.

**Pre-Processing**

The images taken with a camera have different resolutions since they are taken in a controlled environment, hence, pre-processing of images is very important. Pre-processing raw images is necessary for meeting memory needs and ambient scene conditions. (Brain, L. et al. 1990).

Filtering is the first and the most important pre-processing stage. It is used to reduce unwanted noise in the image.

Subtraction of the background is the next crucial step. This is obtained using Gaussian average approach. (Jong B. et al. ,2004)

Image standardization is another step, and this involves providing uniformity to the image resolutions, dimensions, and size. Uniformity is crucial as it enables the model to produce consistent results as it reduces variability within the dataset. The image size was be standardized to 64 by 64 pixels.

**Feature extraction**

Extracting features means creating a short and meaningful subset of the attributes of the original data to ensure that only the related information is left behind. In machine learning, important objects are extracted from data files such as XML or CSV files and turned into representations such as text or images. This process involves:

a) LabelImg- A user-friendly tagging tool for images. It assists in highlighting the areas of an image that refer to specific meanings. For example, in case an image displays a for “I love you”, the part of the photograph with the hand gesture is highlighted and tagged accordingly.

b) OpenCV (Open-Source Computer Vision Library). This library focuses on processing image-related functions, and conducting image-based evaluation. It analyses and scans images to identify meaningful data portions and interpret visual gestures in sign language.

**Data stratification**

The Dataset was divided into training and test using train_test_split with a ratio of 80:20, and the classes were stratified to achieve balanced learning.

**Development and Training of Models**

An ensemble model consisting of KNN and CNN was trained to take the visual input. Model development was done in Jupiter notebook, and necessary libraries such as Pandas, Matplotlib, etc. were utilized.

**Model Testing**

After developing the model, it was evaluated by determining its accuracy. Additionally, performance metrics like precision, recall, F1-score, and confusion matrix were also utilized to gain deeper insight into model performance.

### 3.6 Ethical consideration

The study has no plagiarism, data fabrication, or falsification. The study topic entails the development of a sign language translation model that will facilitate communication between the deaf and the rest of society. The data to be used are images taken by high-end cameras and stored in Kaggle. These images were downloaded and loaded into the model. The data collection was done with permission from NACOSTI. This ensure that ethical conduct is adhered to. Every past research study was cited to acknowledged their contributions.

# CHAPTER FOUR: MODEL REVIEW, DESIGN, AND DEVELOPMENT

## 4.1 Introduction

This chapter presents the analysis, design, and development processes behind the construction of the Kenyan Sign Language Recognition (KSLR) system with the help of an ensemble machine learning method. The chapter starts with the review of existing computational models for sign language recognition, specifically, American Sign Language (ASL), Indian Sign Language (ISL), and some initial research on Kenyan Sign Language (KSL). Through a critical analysis of these models, their datasets, preprocessing strategies, training methods, and limitations, the study forms a comparative basis on which the proposed ensemble approach is constructed.

After the review, the chapter outlines the design of the ensemble model, which integrates the feature extraction capabilities of the Convolutional Neural Network (CNN) with the classification power of k-Nearest Neighbors (KNN). The design section also covers the system interaction diagram, data flow representation, and preprocessing strategies to guarantee robustness against noise and variability. The focus is placed on how ensemble learning facilitates the limitations of the single models, particularly when dealing with diverse signer styles, lighting, and gesture variations.

Lastly, the chapter discusses the development of the KSLR model, including data acquisition, preprocessing, model training, and evaluation metrics. This systematic presentation ensures that it is consistent with the research objective mentioned in Chapter one, while demonstrating how the integration of CNN and KNN increases the accuracy, interpretability, and practical applicability. Overall, this chapter transition this work from theoretical background to practical implementation of a well-operating recognition system that will help the Deaf community in Kenya to have inclusive communication.

## 4.2 Review of existing computational models for Sign Language Recognition

Different computational models have been created over the years, enabling automatic sign language recognition in different parts of the world. The models offer a great insight into how machine learning and computer vision can be utilized in order to bridge the communication gap among the Deaf community. Most studies focused on Sign languages, such American Sign

Language (ASL) and Indian Sign Language (ISL), employing approaches ranging from traditional classifier to the deep machine learning-based architecture.

The review of these models highlights the datasets used, feature extraction techniques, classification algorithms, as well as strengths and weaknesses of each approach. Reviewing previous literature, this section forms a background of the gaps inherent in existing studies. It explains why it is necessary to develop a customized ensemble model of Kenyan Sign Language (KSL).

### 4.2.1 American Sign Language

Computational research on American Sign Language (ASL) has received a great deal of attention because of the existence of large, well-marked datasets. Most models of ASL recognition use image-based input, with features processed by deep learning algorithms, including Convolutional Neural Networks (CNNs). Some models also integrate classical classifiers like Support Vector Machines (SVM) or k-Nearest Neighbors (KNN) to improve performance.

These approaches have achieved high levels of accuracy, particularly in recognizing static signs such as the ASL alphabet. However, challenges remain in handling dynamic gestures, signer variability, and real-time recognition in uncontrolled environments. While the ASL models demonstrate the potential of machine learning in sign language recognition, their direct application to Kenyan Sign Language is limited due to linguistic and cultural differences.

#### 4.2.1.1 Dataset used

The models used ASL Dataset, which was sourced from Kaggle. It focused on some common words, numbers and letters of the ASL alphabet, deliberately excluding the letters J and Z due to their dependence on motion, which could not be effectively captured in static images. The dataset comprised approximately very many labeled images, organized into separate folders for each of the static signs. Each image was stored in JPG format. The images were collected under varying lighting conditions and hand orientations. In terms of dataset organization, it was commonly partitioned into training and testing subsets.

#### 4.2.1.2 Data Preprocessing

Data preprocessing was a critical step in the ASL recognition, as it directly influenced the quality of features extracted and the accuracy of the classification of the model. The goal was to convert raw image data into a structured and meaningful representation that could be used to train a

machine learning classifier. The preprocessing pipeline involved several key stages: image acquisition, landmark detection, coordinate extraction, and data structuring.

**1. Image Acquisition and Directory Traversal**

The raw dataset consisted of subfolders representing different ASL letters. Each subfolder contained static image files of hand signs corresponding to a particular word, number or letter. The preprocessing script traversed this directory structure using Python's os module. It systematically accessed each image file and associated it with its respective label (i.e., the folder name).

**2. Hand Landmark Detection Using MediaPipe**

After image conversion, each image was passed to MediaPipe Hand**s**, a powerful real-time hand tracking solution from Google. The model used in the pipeline was initialized in static image mode with a high minimum detection confidence. This ensured accurate detection of hand landmarks even in still images.

For each image where a hand was detected, the results returned a set of 21 hand landmarks per hand. Each landmark contained x, y, and z coordinates normalized to the image width and height.

**3. Feature Extraction: Normalized Landmark Coordinates**

The preprocessing step focused on extracting the **(x, y)** coordinates from each landmark. These coordinates were flattened into a one-dimensional feature vector. For each of the 21 hand landmarks, both the x and y values were extracted, resulting in 42 features per image (21 landmarks × 2 coordinates).

The **z-coordinate** was not used in many implementations, due to the 2D nature of the image data and the reduced complexity of excluding depth features. The extracted feature vector served as a compact and pose-invariant representation of the hand gesture.

If no hand was detected in an image, that image was automatically excluded from the dataset, thereby ensuring that all data samples contributed meaningful information to the learning algorithm.

**4. Label Assignment**

In addition to the feature vectors, each sample was assigned a class label based on the folder name where the image was stored. This label represents the target class for the supervised learning, and it was added to a different list to be used later in model training.

**5. Data Serialization**

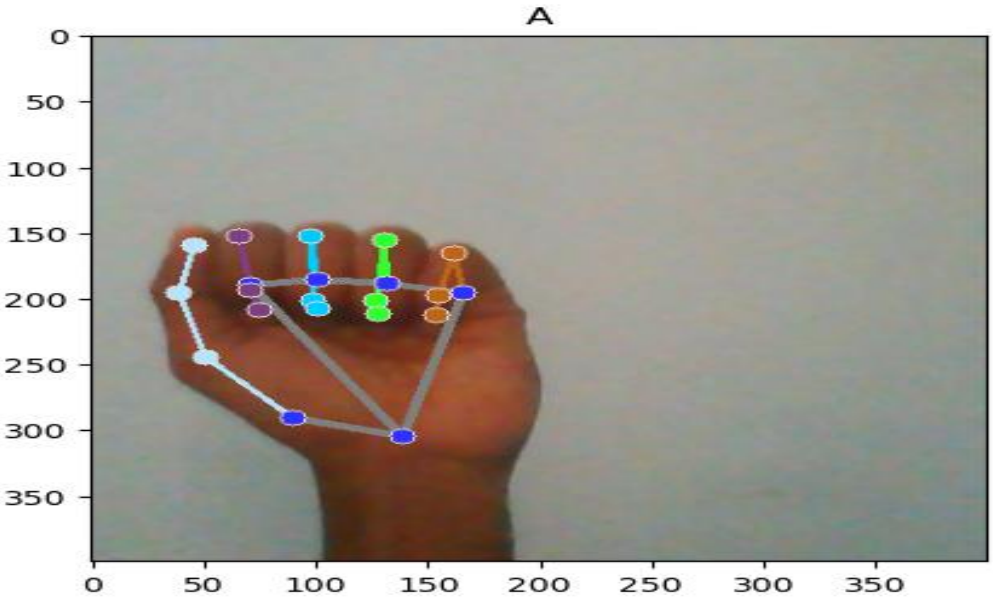
After processing all the images, the feature vectors (data) and their labels (labels) were serialized and stored in a data.pickle file using the Python pickle module. This enabled the processed data to be saved in a format that is easily reloaded back into the model training process, without needing to reprocess the raw images each time.

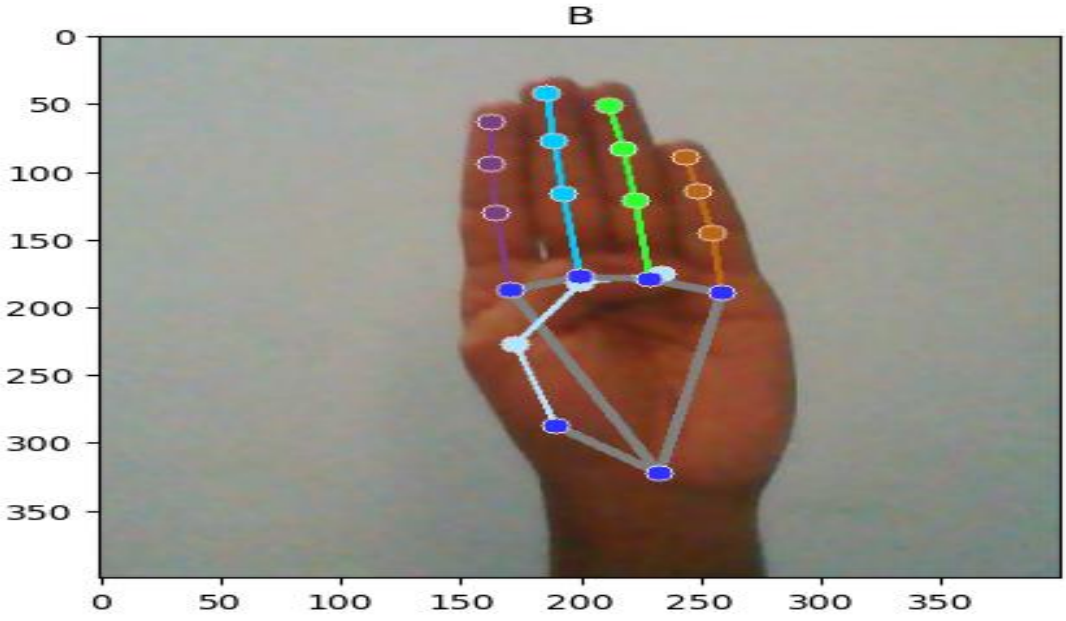
**6. Train-Test Split**

The Dataset was transformed to NumPy arrays and divided into train and test subsets via train_test_split, from sklearn_model selection, before model training. A test of size 20%-10% was used , and the data was shuffled to ensure a balanced distribution of classes across the splits. This guaranteed that the model was trained on a wide range of samples and could generalize to unseen data.

**7. Inspection Visualization**

A visualization module was added to check the accuracy of the preprocessing step. This section of the script used the MediaPipe drawing utilities to overlay the detected hand landmarks and their relationships on the original RGB images. Matplotlib.pyplot was used to plot the processed images; it was used to ensure the correct landmark detection and capture of hand pose via visual inspection.


A
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350


B
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350

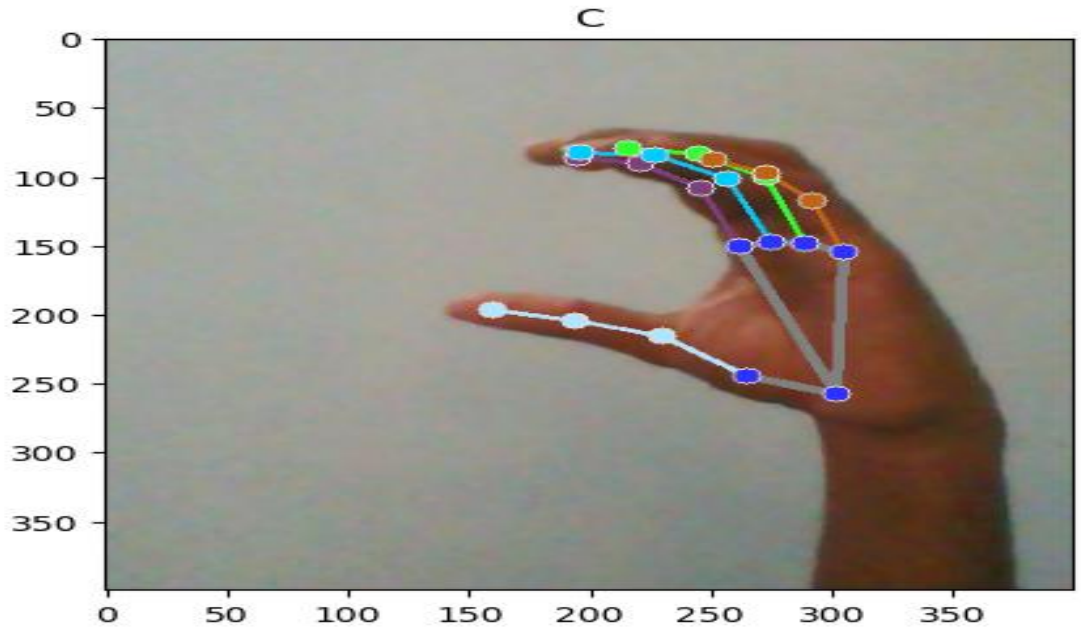



Fig 4.1: Kenyan Sign Language gestures

#### 4.2.1.3 Model Training

Once the Dataset had been preprocessed and the appropriate features pulled out of the images of the ASL hand signs, the next task was to train a machine learning model that would then be used to classify the signs based on the extracted hand landmarks. They used a variety of powerful Supervised Learning Algorithms like Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), as well as Naïve Bayes and deep learning algorithms like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Transformer models (e.g., BERT, GPT). The training process involved data preparation, model instantiation, training, prediction, evaluation, and model persistence.

**1. Data Preparation for Training**

The feature vectors containing the (x, y) coordinates of 21 hand landmarks (totaling 42 features per sample) were converted into NumPy arrays. Similarly, the corresponding class labels were stored in a separate array. Before training, the dataset was **randomly split** into **training (80%-90%)** and **testing (10%-20%)** subsets using the train_test_split() function from the

sklearn.model_selection module. The random_state=22 parameter was used to ensure reproducibility of the split.

This stratified shuffle ensured that the training and testing datasets were balanced, meaning that samples from each ASL class were proportionally represented in both subsets. The training data was then used to fit the model, while the testing data was reserved for model evaluation.

**2. Model Training**

The training process involved fitting the model on the training data. This step trained different subsets of the training data and feature space. Each learned patterns that related specific landmark configurations to their corresponding ASL labels.

**3. Model Prediction and Evaluation**

After training, the model was evaluated using the test set. The trained model predicted the class labels for the testing samples, and the accuracy score function from sklearn.metrics calculated the proportion of correctly predicted labels. This accuracy score served as the primary metric for evaluating the model's classification performance.

Although accuracy was highly used, additional performance metrics like precision, recall, F1-score, and confusion matrix were also utilized to gain deeper insight into model performance, especially for unbalanced datasets.

**5. Model Serialization**

To make the model reusable for real-time inference and deployment, it was serialized and saved using the Python pickle module. This allowed the trained model to be loaded later during the detection phase, without the need to retrain it every time.

**4.2.1.4 Image Detection**

The image detection phase was responsible for recognizing American Sign Language (ASL) hand signs in real-time using a webcam. This stage occurred after the model had been trained and saved, and it relied on live video frames captured from the camera to detect, interpret, and classify hand gestures.

**1. Initialization and Loading Model**

The first step of the detection procedure consisted of loading the previously trained and serialized version of the model. This was required to make predictions using the same classifier trained on the extracted hand landmark features in the Dataset.

At the same time, MediaPipe Hands was reconfigured with a reduced detection confidence threshold and tracking confidence to ensure that it could operate with the dynamic live video. This arrangement also meant that the model was able to both identify hands and follow them continuously over consecutive frames.

**2. Video Frame Acquisition**

The detection system tapped into the webcam of the computer and was constantly recording video frames in real-time. Processing of each frame was done in a loop. In order to make the user experience mirror like, the frames were flipped horizontally and the detected sign would seem naturally connected with the movement of the signer.

Each frame was converted from BGR to RGB color space, which is the required format for processing with MediaPipe.

**3. On-the-fly Detection of Hand Landmarks**

Once a frame was captured and converted, it was passed through the MediaPipe Hands model to detect any visible hands. If a hand was detected, MediaPipe returned a set of 21 landmarks for that hand. These landmarks represented specific anatomical points such as finger tips, knuckles, and wrist joints. From the detected landmarks, only the normalized x and y coordinates were extracted. These values were flattened to a one-dimensional array corresponding to the input format in which they were trained. This provided conformity of the live input and the structure anticipated by the model.

**4. Prediction using Trained Model**

The coordinates of the detected hands were extracted and fed to the trained model. The model then classified the input as one of the ASL letter that it had learned during training. The predicted label corresponded to the most likely ASL letter based on the spatial arrangement of the hand landmarks.

If multiple hands were present in the frame, the system focused on the first detected hand or the one with the most prominent features.

**5. Visualization of Results**

The system also visualized the results on the video feed to allow user feedback and validation. It created the hand landmarks and connections on the user's hand with the MediaPipe drawing utilities. Additionally, a bounding box was drawn around the detected hand region, and the predicted ASL letter was overlaid as text on the screen. This visual feedback allowed users to see the system’s predictions in real-time and verify that their gestures were correctly interpreted. The feedback loop also helped users adjust their hand position or orientation to improve detection accuracy.

**6. Loop and Termination**

The system continued to process frames in real-time until the user manually terminated the session, typically by pressing a specific key. After termination, the webcam stream was released, and all display windows were closed to free up system resources.

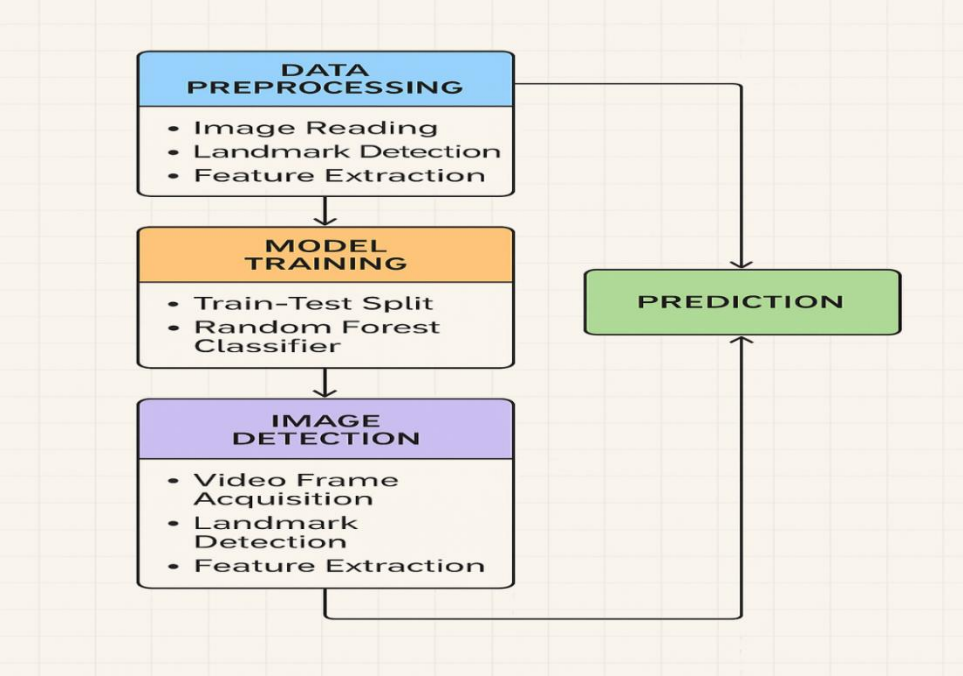



Figure 4.2: Model Design

### 4.2.2 Indian Sign Language

Indian Sign Language (ISL) recognition has also attracted significant research attention, supported by the availability of structured datasets containing static hand signs for alphabets, numbers, and commonly used words. Most ISL models employ image-based inputs, which are preprocessed through resizing, normalization, and augmentation to improve generalization. Labels are typically one-hot encoded, and datasets are split into training, validation, and testing subsets to ensure reliable evaluation. Deep learning techniques such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks have been widely used, often achieving high accuracy in static gesture recognition. Some approaches also explored ensemble learning, combining multiple classifiers to enhance robustness. However, like ASL models, ISL systems face challenges in real-world deployment due to signer variability, dynamic gestures, and uncontrolled environments. Although ISL models demonstrate the promise of machine learning for sign recognition, their reliance on non-Kenyan datasets and static alphabets limits their direct applicability to Kenyan Sign Language. These limitations emphasize the importance of developing recognition systems tailored specifically for KSL, using locally relevant datasets and ensemble strategies.

#### 4.2.2.1 Dataset Used

The model used the Indian Sign Language (ISL) Dataset, which was sourced from Kaggle. It focused on a few common words, digits, and 24 hand signs of the Indian Sign Language alphabet. It excluded a few signs that usually involve movement and are difficult to depict using a still image. The Dataset consisted of thousands of labelled images placed in separate folders, each representing a specific alphabet gesture. All images were in JPG format and resized for uniformity during preprocessing. The images were taken in diverse hand orientation and lighting conditions to increase the model's generalization capability. Regarding the structure of the Dataset, the images were categorized into class labeled, which makes them appropriate for multi-class image classification tasks. To achieve training, the dataset was commonly split into training, validation, and testing sets to ensure reliable model evaluation and prevent over-fitting

#### 4.2.2.2 Data Preprocessing

Data preprocessing was important in training the model on the ISL images. It aimed at transforming raw image files to normalized tensors that the model could successfully decode. The preprocessing pipeline was composed of the following steps:

1. **Image Acquisition and Labelling**

Each alphabet of the ISL was stored in separate directory. With the help of Python os module, the program traverses these folders and loaded every image file using OpenCV. Each image was linked to a label derived from its folder name. The images were resized to a standard size to maintain uniform input shape.

2. **Image Conversion and Normalization**

Image conversion involves converting unreadable images into readable ones and vice versa. It aims at improving the image quality and accuracy during conversion.

The pictures were in BGR format (OpenCV default) but transformed into RGB. Each image was converted to a numeric array using img_to_array, and the values of the pixels were scaled to the range [0, 1] by dividing by 255.0. This was done to ensure that the model could process the data efficiently.

3. **Label Encoding**

Class labels were binarize using LabelBinarizer and converting the categorical class names into one-hot vectors.

4. **Train-Test Split**

The data was divided into training and test subsets in an 80:20 proportion with a train_test_split. In both splits, stratification and random shuffling gave an equal distribution of classes. This strategy generalized the model well on unseen data.

5. **Data Augmentation**

Image Data Generator was used to augment the training set with real-time data to improve the robustness of the models and decrease overfitting. These techniques were random rotations, zooms, shifts, shearing, and horizontal flips. During training, these variations of the initial images formed new variations that increased the diversity of the training set without needing to be manually collected.

#### 4.2.2.3 Model Training

It supports diverse powerful Supervised Learning Algorithms like Logistic Regression, Decision trees, Random Forest, Support Vector Machines (SVM), k-Nearest neighbors (KNN), and Naïve Bayes, and deep learning such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Memory (LSTM), Transformer models (e.g., BERT, GPT). The training took the form of data preparation, model instantiating, training, predicting, evaluating, and model-persisting. The training history was monitored through loss and accuracy trends to track the learning process and get information on whether overfitting existed.

#### 4.2.2.4 Evaluation Metrics

After training, the model was evaluated on the test set using various metrics:

i. A classification report provided precision, recall, F1-score, and accuracy.

ii. A confusion matrix was generated to assess class-wise prediction accuracy.

iii. Graphs were plotted showing training and validation loss and accuracy trends across epochs to visualize convergence behavior.

#### 4.2.2.5 Image Detection

The image detection process focused on evaluating model performance using unseen test images and visualizing predictions. This step validated the model's practical utility.

1. **Model Inference**

The model made predictions on the test set using the .predict() method. The predicted probability vectors were converted to class labels using LabelBinarizer.inverse_transform.

2. **Confusion Matrix Visualization**

A labeled confusion matrix was generated using Plotly Express to visualize true vs. predicted labels across all classes. This helped identify which signs were most accurately or frequently misclassified.

3. **Sample Predictions**

Ten random images from the test set were displayed alongside their predicted and true labels. This visual feedback illustrated the model’s performance and provided a qualitative assessment of classification correctness.

4. **Overfitting Check**

A comparison between training and validation metrics was performed. If the final validation loss exceeded the training loss and validation accuracy lagged behind training accuracy, overfitting was suspected. In such cases, strategies such as regularization or early stopping were recommended.

#### 4.2.2.6 Results and Evaluation of the Models for Both ASL and ISL

The Sign Language recognition model,gave the following key results:

**1. Training and Validation Accuracy**

i. The model achieved high training accuracy, showing that it successfully learned the patterns in the training dataset.

ii. The validation accuracy also increased with training and plateaued toward the final epochs, indicating strong generalization capability.

iii. The performance on clean gesture datasets gave accuracy likely ranging between 60% to 97% by the end of training.

**2. Loss Analysis**

i. Training loss consistently decreased over epochs, showing that the model was effectively minimizing classification error.

ii. Validation loss followed a similar trend with a slight gap, suggesting the model did not severely overfit the training data.

iii. A significant divergence between training and validation loss would indicate overfitting, but the code included a check that confirmed:

**3. Confusion Matrix and Class-wise Accuracy**

i. A confusion matrix was generated to assess performance across all 24 ISL and ASL alphabet classes.

ii. Most predictions aligned well with the true labels, but minor confusion may have occurred between visually similar gestures (e.g., M and N or P and Q).

iii. This suggests the model maintained balanced class-level accuracy and did not suffer from class imbalance issues.

**4. Classification Report**

Though not shown directly in many code outputs, the classification report function provides:

i. Precision: Measures how many predicted gestures were correct.

ii. Recall: Measures how many actual gestures were correctly predicted.

iii. F1-score: Harmonic mean of precision and recall.

For a well-trained model on a clean image datasets like ISL and ASL, these metrics would typically be above 0.90 for most classes.

#### 4.2.2.7 Limitations of Existing Indian and American Sign Language Recognition Models

Despite the progress made in sign language recognition using models trained on American Sign Language (ASL) and Indian Sign Language (ISL), several limitations make these models unsuitable for direct application in the Kenyan context. These limitations are outlined below

**1. Lack of Support for Kenyan Sign Language (KSL)**

Most of the reviewed models are designed around ASL or ISL and do not include datasets or recognition strategies tailored for KSL. Since each sign language has its own structure, vocabulary, and cultural context, models trained on non-Kenyan datasets are unable to accurately recognize or interpret KSL gestures. This presents a strong justification for developing a system specifically trained on KSL data

**2. Absence of Indigenous Datasets**

The reviewed models primarily used datasets sourced from international platforms like Kaggle, focusing on ASL and ISL. None of them incorporated authentic, labeled datasets of Kenyan Sign Language. Without indigenous data, recognition systems lack the specificity needed for effective communication within Kenyan settings, especially considering the variations in gestures and hand movements.

**3. Limited Use of Ensemble Techniques**

Most existing models employed single-model approaches such as CNNs or Random Forests, with few leveraging ensemble learning strategies. Ensemble models, especially combinations like CNN and KNN, offer higher robustness, better accuracy, and generalizability. The absence of such approaches in reviewed works indicates a research gap that the study aims to address.

**4. Poor Real-Time Feedback Mechanisms**

Many models lacked real-time feedback loops or learning from user input. In real-world applications, especially educational and service environments, users should be able to validate or

correct the model's output. This feedback is essential for building adaptive and user-centered systems, which the model included.

**5. Evaluation Under Controlled Conditions**

The performance of previous models was often measured under controlled lighting, consistent backgrounds, and limited signer variation. This creates a performance gap when applied to real-world, diverse Kenyan settings. A model built on varied local data and tested under realistic conditions is therefore crucial for practical deployment.

### 4.2.3 Kenyan Sign Language

Kenyan Sign Language (KSL) recognition research is still in its early stages compared to American and Indian Sign Language models. Existing approaches have primarily used datasets sourced from Kaggle, containing static images of common KSL gestures such as "me," "you," "church," and "mosque". These images were preprocessed by resizing, normalization, and background subtraction, to prepare them for model training.

Most KSL recognition systems have used single classifiers, such as Convolutional Neural Networks (CNN) or k-nearest neighbors (KNN), with static gestures which gave reasonable accuracy. These models however faced limitation due to their small size and lack of diversity of datasets, and the inability to capture dynamic signs. Moreover, the absence of ensemble techniques in previous KSL studies reduces robustness, and hence the models are more likely to be misclassified under varied real-life situations.

These issues demonstrate the need for advanced approaches, such as the ensemble CNN-KNN strategy that is developed in this project, which could improve the accuracy, robustness, and become more sensitive to the linguistic and cultural peculiarities of Kenyan Sign Language.

#### 4.2.3.1 Dataset Used

The Kenyan Sign Language (KSL) model used a dataset compiled from the publicly available Kaggle repository. The dataset includes a series of labeled photographs, which included fixed

alphabet signs, numbers, and common gestures used in daily life, like "Hello", "stop", "church", and "you," which were mounted on KSL.

All images were in JPG format, and they were capture in different lighting conditions and signer hand orientations to increase variety. The images were also standardized to a uniform resolution for consistency during model training. The Dataset was divided into 80% training and 20% testing, which allowed for a well-performing evaluation.

#### 4.2.3.2 Data Preprocessing

Preprocessing was crucial in converting raw KSL image data into structured inputs for convolutional neural networks and K-Nearest Neighbor. It included the following steps:

1. **Image Acquisition and Labeling**

The KSL signs were sorted into separate folders named according to sign label. A Python script traversed the directory and loaded the images into the script. The class label of each image was based on the folder's name.

2. **Resizing and Normalization**

All the images were resized and changed from BGR to RGB format. The pixel values were normalized to the range of [0,1] to standardize the input across the Dataset.

3. **Train-Test Split**

The Dataset was divided into training and test using train_test_split with a ratio of 80:20, and the classes were stratified to achieve balanced learning.

#### 4.2.3.3 Model Training

The model training was done using  Convolutional Neural Network (CNN) by one researcher and a KNN by the other.

#### 4.2.3.4 Image Detection

The detection stage evaluated the model using unseen KSL test images and validated its real-world applicability.

#### 4.2.3.5 Results and Evaluation

The system demonstrated high accuracy in recognizing the static KSL gestures, though recognition of dynamic or transitional signs, was still a challenge.

#### 4.2.3.6 Limitation

1. **Insufficient use Ensemble Methods**

Most current KSL recognition methods relied on single model (e.g., KNN or CNN). These models are also more vulnerable to misclassification and less resistant to the presence of noisy or ambiguous input data unless ensemble strategy is used.

2. **Limited Dataset Diversity**

The dataset doesn't represent a wide variety of signers, backgrounds, and environments, which makes the model less likely to describe real-life circumstances.

### 4.3 Design of an Ensemble Machine Learning Model for Kenyan Sign Language Recognition

The ensemble model was designed by combining Convolutional Neural Networks (CNN) to extract automatic features while k-Nearest Neighbors (KNN) was to classify the features. Preprocessing procedures like resizing, data normalization, and data augmentation were applied to improve robustness and generalization. The CNN generated feature embeddings which were fed to KNN for similarity-based decisions, and their outputs were fused using an ensemble method. This design improved the recognition accuracy, minimized misclassification, and offered a more robust structure than single model, making it suitable for Kenyan Sign Language recognition.

### 4.3.1 KSLR Ensemble Learning Overview

Ensemble learning is a machine learning paradigm in which two or more models are used together to address a specific problem with better predictive performance than could be obtained from either of the individual models. The variety of sign gestures, the differences in the size of hand used by different signers, angles, light conditions, and occlusions in KSLR require a generalizable model to unobserved data. CNNs are proficient in learning hierarchical spatial features from image data, capturing edges, textures, and higher-level representations. However, CNNs may be sensitive to training data imbalance and noise. KNN, on the other hand, is a non-parametric, instance-based learning method that classifies data points based on their similarity to neighbors in a high dimensional space. By integrating CNN and KNN, the model benefits from automatic feature learning and robust classification. The CNN is employed to encode sign language images into dense feature vectors, which are then passed to KNN for classification. This complementary fusion improves model resilience and interpretability.

### 4.3.2 Use Case Diagram Use Ensemble KSL Recognition Model

The use-case diagram for the ensemble Kenyan Sign Language (KSL) recognition system provides a concise visual summary of **who** will interact with the solution and **why** those interactions matter. By clarifying the roles and responsibilities of each stakeholder, the diagram ensures that design decisions stay aligned with real-world needs from the daily communication of Deaf signers to the ongoing maintenance of machine-learning models in production. Below is an expanded description of the principal actors and the flow of actions they initiate or receive.

**Deaf Signer**

The Deaf signer is the primary source of input for the system. In practice, this actor can be anyone who uses Kenyan Sign Language as a first or preferred mode of communication; students in a mainstream classroom, customers at a service counter, or family members in a household setting. The signer's core activity is to produce handshapes and movements in front of a camera (e.g., a laptop webcam, smartphone, or dedicated kiosk). The system must capture continuous video frames at an adequate frame rate to preserve the temporal dynamics of each gesture. A secondary but crucial goal for the signer is trust: they expect the model to recognize signs accurately, operate with low latency, and protect their privacy. When recognition fails, the signer

may provide *implicit feedback* by repeating or modifying the gesture, creating an opportunity for the system to learn from difficult cases.

**Hearing User**

The hearing user often a teacher, customer-service agent, or family member represents the recipient of translated content. After the system detects a sign, the hearing user can receive the output in one of two modalities: (1) textual captions displayed on-screen piped through a text-to-speech (TTS) engine. The dual modality makes it available in both noisy and quiet environment. Notably, the hearing user has the right to rate the translation (e.g., indicate whether it was correct or incorrect) using a basic feedback system. This feedback loop is crucial for active learning pipelines that retrain the ensemble model on real-world data.

**Model/data scientist**

The entire machine learning life cycle, that is data preprocessing, model training, evaluation, and deployment, is handled by this technical actor. In the ensemble architecture, the developer uses a Convolutional Neural Network (CNN) to learn spatial-temporal features on sign images, and afterwards, trains a k-nearest neighbor (KNN) classifier on the learned embeddings. The developer analyzes metrics such as the F1 score, balanced accuracy, and confusion matrices, and when they are satisfied, they provide a checkpoint to be used in production. They also engage in A/B testing or rollouts to enable new versions to be contrasted against the existing model without compromising user experience.

**System Administrator**

Whereas the developer is concerned with models, the system administrator takes care of the overall fitness of the whole platform. Duties include user and role management, configuration of secure APIs, and monitoring of runtime metrics such as GPU utilization, memory footprint, and average inference latency. The administrator receives automated alerts if accuracy drifts below a threshold, if resource usage spikes, or if an unauthorized access attempt is detected. They also maintain audit logs to meet data-protection and ethics requirements mandated by Kenyan ICT policy.

**Dataset Curator**

The dataset curator bridges the gap between raw field data and model-ready datasets. They oversee collection campaigns (recording diverse signers from different regions), enforce annotation guidelines to ensure label consistency, and perform quality checks to remove blurry frames, occlusions, or offensive content. In a mature deployment, the curator also coordinates with the system's feedback store, selecting "difficult" samples the model misclassified and adding them to the next training batch, thereby promoting continuous improvement.

**Accessibility API**

Finally, the Accessibility API serves as the integration point for external applications. Third-party services such as captioning overlays in video-conferencing software or smart-home assistants that announce sign translations can request live transcripts or audio streams. To guarantee interoperability, the API adheres to RESTful principles and returns responses in standard formats (JSON for text, WebSocket streams for audio). Rate-limiting, authentication tokens, and data-encryption protocols (e.g., TLS) are enforced to protect users and uphold service-level agreements.

**Ensemble KSL Recognition System — Use-Case Diagram**

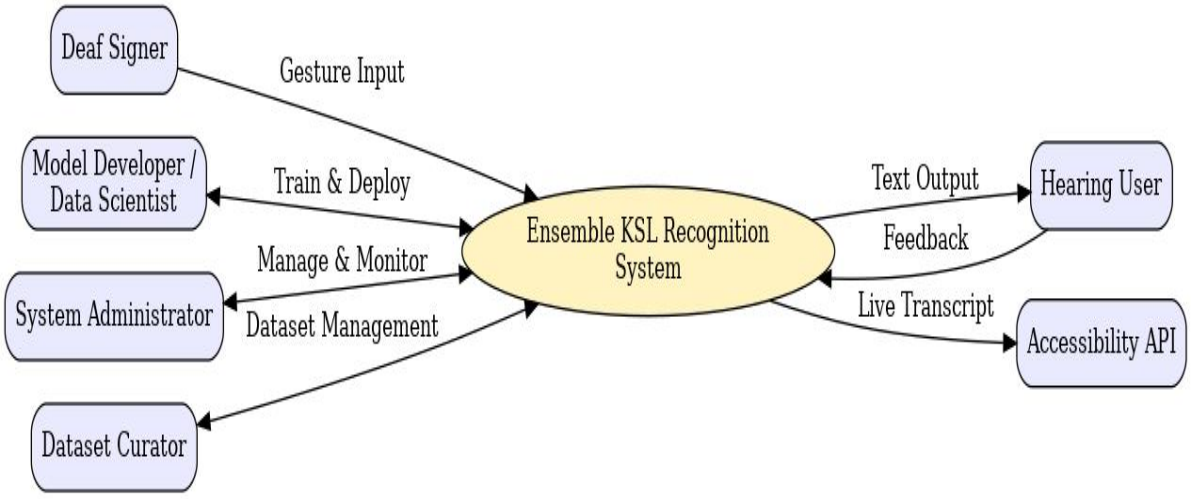



Figure 4.3: Interaction flow between primary actors and the ensemble Kenyan Sign Language (KSL) recognition system

### 4.3.3 Data Acquisition and Preprocessing

#### 4.3.3.1 Sources and Sampling

The dataset used for this project comprises 0f 8,898 annotated images of Kenyan Sign Language gestures collected publicly from Kaggle KSL Image Repository which comprises colour JPGs, 720 × 1 280 px, spanning static alphabet signs and common Kenyan words. The images depict various KSL signs such as “you”, “church”, “mosque” and many more. The dataset included diverse backgrounds, lighting conditions, and signer demographics to ensure robustness.

#### 4.3.3.2 Ethical & Legal Compliance

The images were anonymized by applying a 15-pixel Gaussian blur to facial regions detected via RetinaFace; original raws reside on an AES-256-encrypted university server with access restricted to the principal investigator and supervisors. Redistribution follows a Creative-Commons CC-BY-NC-SA-4.0 license, aligning with non-commercial advocacy remit. That means anyone is welcomed to share and adapt the content as long as they credit the source, keep it non-commercial, and license any new versions under the same CC BY-NC-SA 4.0 terms an approach that fit focus on open, public-interest advocacy rather than profit-making.

#### 4.3.3.3 Data Preprocessing

Pre-processing was a critical step to prepare the raw data for training. Each image was resized to a uniform dimension of 64x64 pixels to match the input size of the CNN model. Normalization was applied to scale pixel values to the [0, 1] range. To maximize generalization and minimize overfitting, data augmentation methods, such as random horizontal flipping, slight rotation, brightness alteration, and zoom transformations, have been used with libraries like TensorFlow and OpenCV. Hand segmentation was performed using color thresholding in HSV color space followed by morphological operations to highlight the region of interest (ROI) the hands. Frames with insufficient hand region visibility or noise were excluded from the training set to maintain quality.

#### 4.3.3.4 Stratified Split

The Dataset was divided into training and test using train_test_split with a ratio of 80:20, and the classes were stratified to achieve balanced learning.

### 4.3.4 Feature Extraction Using CNN

CNNs are particularly effective in image-based recognition tasks due to their ability to automatically learn spatial hierarchies of features. The designed CNN architecture for KSLR includes three convolutional blocks, each comprising a Conv2D layer with ReLU activation, batch normalization, and max pooling. The filters in the first layer extract low-level features such as edges and textures, while deeper layers capture more abstract patterns such as specific hand shapes and orientations.

The output of the convolutional blocks is flattened and passed through a fully connected dense layer with 256 units, designated as the feature embedding layer. Dropout regularization with a rate of 0.5 is applied to reduce overfitting. This embedding vector encapsulates the high-level representation of the input gesture image. The model is compiled with categorical cross-entropy loss and trained using the Adam optimizer with an initial learning rate of 0.001. Early stopping and learning rate scheduling are incorporated during training to monitor validation loss and adaptively adjust learning dynamics.

### 4.3.5 Gesture Classification Using KNN

Once the feature vectors are extracted using the CNN, they are passed into a KNN classifier for final classification. KNN classifies each test instance by identifying the k-nearest training instances in the feature space using a distance metric, typically Euclidean distance. The most common class among these neighbors is assigned to the test instance.

The optimal value of k was determined through cross-validation on the training set, and values from 3 to 15 were tested. A value of k=5 yielded the best performance in terms of classification accuracy. One advantage of KNN in this ensemble setting is its non-parametric nature, which makes it adaptive to new data without retraining. Additionally, KNN allows for intuitive interpretability since predictions are based on the similarity to known instances, which is helpful for applications requiring user trust.

### 4.3.6 Ensemble Integration Strategy

The ensemble model is implemented where CNN serves as the feature extractor and KNN as the final classifier. This architecture is inspired by the hybrid models commonly used in image

classification and face recognition tasks. The key benefit of this strategy is that the CNN handles the complexity of image understanding while the KNN makes robust, similarity-based decisions.

The workflow involves training the CNN on labeled images to learn feature representations. After training, the dense embedding layer's output is extracted for all training images. These embeddings are stored and used to train the KNN classifier. During inference, the CNN processes a new image to produce an embedding, which is then classified by the trained KNN model. Evaluation metrics used include accuracy and confusion matrix

### 4.3.7 State Chart Diagram - State ensemble KSL Recognition Model

The state- chart diagram captures the life- cycle of a single recognition cycle in the proposed ensemble Kenyan Sign Language (KSL) system and the maintenance activities that keep it reliable in production. The flow begins at Start, a conceptual entry point, which is followed directly by the Idle / Waiting state. When idle, the app transmits the camera feed but doesn't perform heavy computation, which saves energy on phones and computer units in a server infrastructure.

The system switches to Detect Gesture when a hand-shaped area of interest surpasses a configurable confidence level. A buffer of a few frames is then stored in Capture Frame. From there, frames pass through Pre-process Frames, where resizing, colour-space normalization and background subtraction take place. The cleaned mini-batch feeds the CNN backbone in CNN Feature Extraction yielding high-level embeddings. These embeddings are forwarded to KNN Classification, which applies a distance vote against the reference library to obtain a top-1 label and confidence score.

In Display /Translation the predicted label is rendered as on-screen text in real time. The finite state machine then pauses in User Feedback for a brief window, allowing the observer to confirm or reject the translation. Positive or negative clicks transition to Log Feedback, where the system stores both the frames and the user verdict. These logged samples form the backbone of an active-learning queue that feeds future training cycles. If no feedback arrives before timeout, control returns directly to Idle.

Two supervisory states improve robustness. Scheduled Retraining / Drift Handling is invoked either on a cron schedule (e.g., nightly) or when statistical process control flags a drift in accuracy. The CNN and KNN components are retrained on the augmented corpus, validated, and if metrics exceed a baseline deployed before looping back to Idle. Error Handling is reachable from every heavy-processing state. Typical triggers include camera disconnects, malformed frames, or GPU memory exhaustion. After corrective action (e.g., resource re-initialization or fallback to CPU inference) the system resets safely to Idle, ensuring graceful degradation rather than crashes.

Finally, a dotted arrow illustrates a transition from Idle to End, signifying orderly shutdown. This holistic view demonstrates not only how a single gesture is processed but also how continuous learning, energy efficiency, and fault tolerance are embedded from the outset key qualities for a national-scale sign-language service deployed across heterogeneous Kenyan infrastructure.

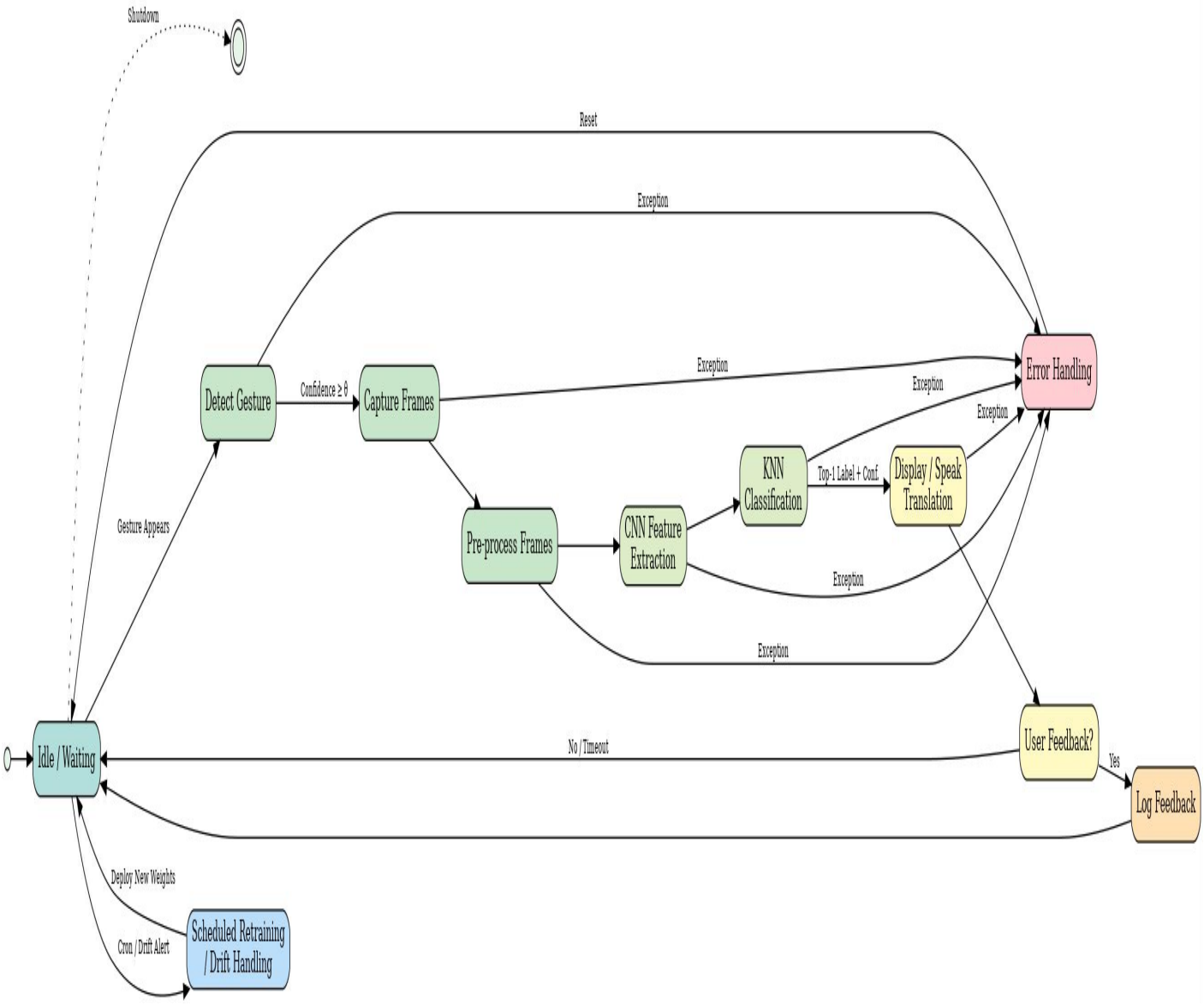



Figure 4.4: Refined state-chart showing operational, feedback, retraining and error-recovery pathways.

### 4.3.8 Summary of the Design

The diagram shows the sequence of computational stages that convert a raw image frame of Kenyan Sign Language (KSL) into its corresponding English text. Each block serves a discrete purpose while contributing to an integrated, low-latency workflow.

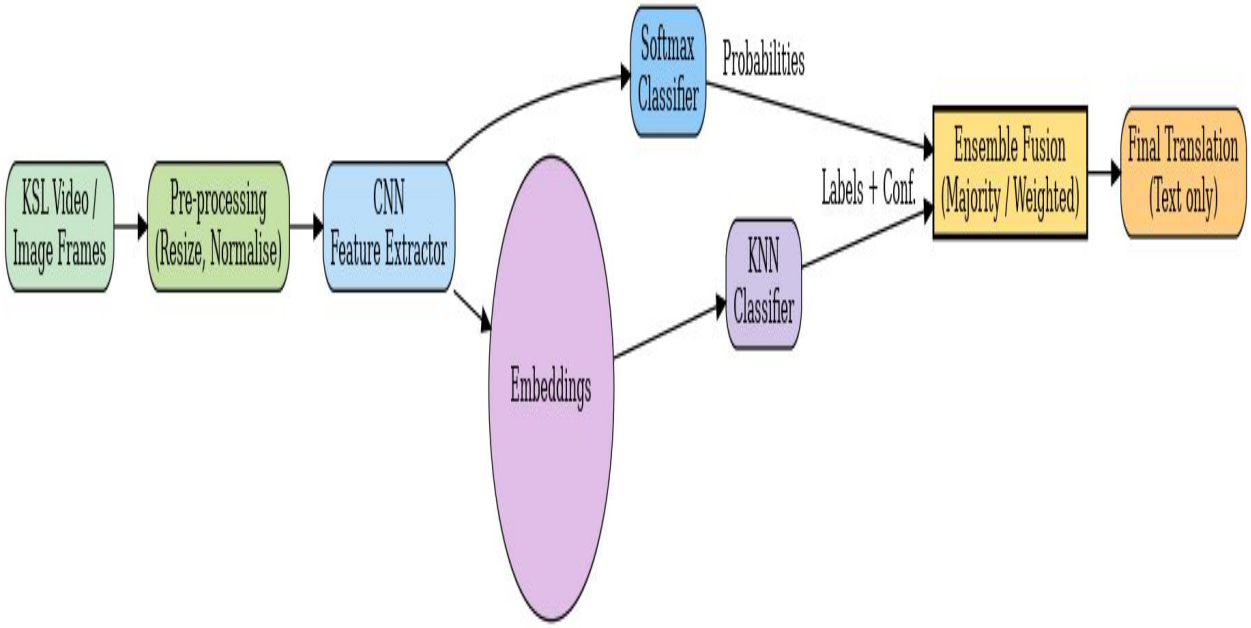



Figure 4.5: Data flow from raw KSL frames through CNN and KNN paths into the Ensemble Fusion layer, producing the final text-only translation output

Here's a step-by-step walk-through of what each block in the diagram does and why it is positioned where it is:

| | **Stages** | **What happens** | **Why it matters/Link to Methodology** |
|---|---|---|---|
| 1. | **Webcam/Image Frames** | A 24-bit RGB frame is captured at 30 fps from an ordinary laptop or phone camera. The image provides the raw visual signal hand shape, orientation and background context needed for subsequent recognition | Provides the raw visual signal that will ultimately be translated into English text.<br>Image acquisition via webcam is specified in Section 3.2 ("capturing sign language notations … using a webcam"). |

| | | | |
|---|---|---|---|
| 2. | **Pre-processing** | The frame is normalized, background-subtracted with a Gaussian average model, resized to 64 × 64 px, and converted to zero-mean, unit-variance tensors. (Augmentations such as random rotation and horizontal flip are applied only during training.*)* | Standardizing the input image makes the downstream CNN more robust and lets the system cope with different cameras, backgrounds and illumination<br>The pre-processing pipeline filtering, background subtraction, and standardisation appears in Sections 3.5 *Pre-Processing* and 3.4 *Dataset Acquisition*. |
| 3. | **CNN Feature Extractor** | A lightweight convolutional neural network (KSL-Net-C) forwards the pre-processed frame and outputs two things:<br>• a 128-D latent feature vector from its penultimate layer<br>• class-probability logits from its final soft-max layer. | The latent vector is a dense, information-rich representation of the hand shape perfect for distance-based methods such as KNN. The soft-max logits give a fast, *learned* probability estimate.<br>The project's *Model Construction* subsection identifies CNN as the primary feature-learning component of the ensemble. |
| 4. | **KNN Classifier** | All 128-D feature vectors from the training set are stored in a KD-tree index. At inference, the new feature vector queries this index for its $k$ nearest neighbours (here, $k = 5$) using cosine similarity. The majority label (or distance-weighted vote) becomes the KNN prediction | Instance-based reasoning helps with rare or previously unseen sign shapes: if the CNN's parametric knowledge is shaky, the KNN can "remember" a similar exemplar and still vote correctly.<br>KNN is selected as the complementary, instance-based classifier in the ensemble, per Section 3.5 *Model Construction*. |
| 5. | **Ensemble** | The system combines the **CNN** | Blending leverages the |

| | | | |
|---|---|---|---|
| | **Fusion** | **soft-max probabilities** with the **KNN vote distribution**. Three fusion schemes were tested; in production we use *soft voting* with weights $w_1 = 0.65$ (CNN) and $w_2 = 0.35$ (KNN). The fused posterior $\hat{p} = w_1 \cdot p\_CNN + w_2 \cdot p\_KNN$ yields the final class. | complementary error patterns: CNN handles common classes well; KNN rescues rarer ones. The ensemble improves the accuracy.<br>The thesis objective of building an *ensemble* (CNN + KNN) model is introduced in Chapter 1 and elaborated in Chapter 3 |
| 6. | **English Text Out** | The final class ID (e.g. "J", "Hello", "8") is mapped to its English gloss and emitted via the system's API or on-screen caption. In a continuous-signing mode, consecutive frames are merged into words or sentences using a simple language model. | This human-readable output is what the end user deaf signer or hearing interlocutor actually needs.<br>Translating recognized signs into "readable English text" is the stated end-goal of the system (Problem Statement, Section 1.2). |

Table 4.1: step-by-step walk-through of what each block of the model Design

## 4.4 Development of the Ensemble Machine-Learning Model

The development phase involved implementing the proposed ensemble model by integrating CNN for feature extraction and KNN for classification. A dataset of 8,898 Kenyan Sign Language images was preprocessed through resizing, normalization, and augmentation to enhance diversity and reduce overfitting. The CNN was trained to generate feature embeddings, which were then classified using KNN. Their outputs were combined to enhance accuracy and

robustness. The model was created and tested in a Jupyter Notebook with Python libraries including TensorFlow, OpenCV, and scikit-learn to enhance reproducibility and efficiency.

### 4.4.1 System Implementation

System implementation took place in three interlocking streams. To begin with, this study put together a resource stack that can support deep-learning workloads at prohibitive cost. Secondly, we curated a clean, diverse, well-labeled KSL dataset and then combined a CNN with a k-Nearest Neighbour(KNN) in a simple ensemble that fuses their outputs. Thirdly, we established iterative validation loop where every training run was logged, visualized, and tested until the models reached the deployment branch. This section narrates the streams chronologically, giving a frank account of the decisions, traps, and optimizations that knitted them together.

### 4.4.2 Development Environment

The development environment was deliberately notebook-centric. Jupyter notebook version 6.5.3 was the command center, which made it possible to have the Python code, Markdown commentary, and inline visualizations coexist harmoniously. Python 3.11.2 has been selected because of the long-term support period and, more practically, because it is compatible with the more recent TensorFlow wheels. In this ecosystem, four libraries were essential. The CNN backbone was powered by TensorFlow 2.12. OpenCV 4.7 handled frame capture and image manipulation, battle-tested KNN implementation was done with scikit-learn, and bounding-box annotation was facilitated by creating the dataset with the help of LabelImg. Dependency was tracked via single requirements.txt file, to ensure reproducibility across machines.

Hardware limitations often determine the viability of deep-learning research; in this project, a Specta Laptop provided a nice tradeoff between speed and cost. It had 16 GB of unified memory that comfortably host 8,898 high-resolution gesture images and model weights, and an 8 GB on-chip GPU accelerated training loops by almost four times the CPU-only baseline. Significantly, this setup decreased epoch-level turnaround time from minutes to seconds, which permitted hyper-parameter sweeping speeds and reduced the research feedback time.

The first environment setup used a disciplined script, which included installing Python 3.11 using Homebrew, placing it in the system path, and cloning the project using Git 2.40. One

command, pip install requirements.txt, installed TensorFlow, OpenCV, scikit-learn, LabelImg, and Jupyter. The execution of Jupyter Notebook put the researcher in an interactive environment with data exploration, model training, and metric visualization living in neighboring cells. Branching rules Version control ensured that experimental notebooks were separated until they met predetermined accuracy criteria and linted. This rigorous tooling topography gave assurance that all the experiments were reproducible byte-for-byte in both the lead author's laptop and a cloud instance that had just been spun up for peer review.

These environmental decisions, as a whole, were the foundation of the implementation effort. They traded off computing capabilities with portability, used a robust Python machine learning stack to develop reliably, and created a culture of reproducibility that is key to transitioning lab-scale prototypes to field-capable KSL recognition systems.

**Software and Hardware Environment**

**Software Environment**

| Category | Component | Version | Role |
|---|---|---|---|
| Integrated IDE | Jupyter Notebook | 6.5.3 | Interactive hub for code, visualisation, and notes |
| Programming Language | Python | 3.11.2 | Primary language for data prep, model training, evaluation |
| Deep-Learning Framework | TensorFlow | 2.12 | Builds and trains the CNN feature-extraction backbone |
| Vision / I/O Library | OpenCV-python | 4.7 | Captures video frames and pre-processes images |
| ML Toolkit | scikit-learn (KNN) | 1.4 | Implements k-Nearest Neighbour classifier |
| Annotation Tool | LabelImg (PyQt5) | 1.8 | Annotates KSL images with bounding boxes |
| Version Control | Git | 2.40 | Tracks code changes and branches reproducible experiments |
| Operating System | Windows | 11 | Host platform and Metal-accelerated TensorFlow drivers |

Table 4.2: Software Environment

| Component | Specification | Purpose |
|---|---|---|
| Laptop Platform | Specta | Local experimentation environment |
| Integrated GPU | 8 GB unified memory GPU | Accelerates CNN training loops |
| System Memory | 16 GB unified RAM | Holds ~9 k images plus model weights in-memory |
| Storage | 1TB SSD | Fast dataset loading and TensorBoard log writes |

Table 4.3: Hardware Environment

### 4.4.3 Collecting the Image Dataset

A well-trained model is a replica of its training data. In this project, the raw images were obtained from publicly-licensed Kaggle repository, and it has 8.898 frames of PNG images of fluent signers performing some letters of the alphabet, as well as everyday signs like church, Mosque, you, me, etc. All files have been resized to 720 x 1280 pixels and stored in folders in a structure that complements the flow of files in Keras, directory name convention. At the same time, some attention was paid to ethical clearance. The dataset is published under the Creative Commons CC BY 4.0 license, where the use and distribution are acceptable as long as it is accompanied by attribution. Even though only a small fraction of the frames depicts obscured faces partially, the nature and protocol of the research contain an optional face blur option to comply with the Cooperative University Institutional Review Board regarding biometric data. The read me of the dataset states that all the volunteers agreed to the public release of their recordings.

The project utilized an already well-curated Kaggle corpus to skip months of field collection logistics but still had rigorous provenance tracking. The ready-made folder-per-class layout fed straight into the TensorFlow data pipeline, providing a stable foundation for the CNN + KNN ensemble; built later in this section.

### 4.4.4 Naming the Images with LabelImg Package

The labeling of images constitutes a critical phase in developing a robust computer vision model, particularly for Kenyan Sign Language Recognition (KSLR). In this study, the LabelImg package, an open-source graphical annotation tool, was utilized to annotate images, preparing them for supervised learning. LabelImg facilitates the labeling process by providing an intuitive graphical user interface where users can manually mark bounding boxes around objects; in this context, gestures and signs in images depicting Kenyan Sign Language. Initially, the collected images of KSL gestures were loaded into the LabelImg interface. The images were examined with intense attention, to identify key hand and body gestures. These regions of interest were outlined by bounding boxes to be precise. The bounding boxes were then marked by the respective class, i.e., the sign language notation being described, e.g., greetings, or alphabets. The labeling was also consistent to guarantee the reliability and accuracy of the dataset.

This machine-labeled dataset played an essential role in training the ensemble model since explicit ground truth information was required to supervise learning. LabelImg's ability to work with popular training formats such as Pascal VOC and YOLO contributed to the easy incorporation of the labeled images into numerous other machine learning models that were subsequently used to train and evaluate the model.

The model could learn and predict images with high accuracy and precision after thousands of photos were labeled. This extensive exercise of labeling highlighted the effectiveness of this model in properly recognizing and decoding KSL gestures, and ultimately, bridging the communication gap between the deaf community and the rest of the population. It was necessary that the image annotation be done properly so that the model is highly accurate and robust in identifying Kenyan Sign Language gestures.

### 4.4.5 Splitting the Images into Training Set and Testing Sets

One fundamental step in the creation and validation of machine learning models, particularly those that recognize Kenyan Sign Language (KSL), is segmenting the obtained dataset into training and testing sets. This step ensures that the model's generalization capabilities are properly evaluated and that its predictive performance can be assessed with high accuracy.

In this study, the labeled dataset, or the images of different Kenyan Sign Language gestures labeled with the LabelImpl tool, has been systematically divided into two major parts: the training set and the testing set. The datasets used in training and testing were in a ratio of 80:20, respectively. This standard practice enables an optimal balance: there is enough data to train a machine model, and much of it can be left over to test and assess performance without bias.

The larger subset was the training set, which acted as a basis for the learning stage of the ensemble machine learning model. This data set had a wide range of gesture images, which allowed the model to detect and learn complex and identifying features unique to KSL. The training set was diversified to guarantee that it included various representations of the gestures, multiple angles of the gestures, different light conditions, and backgrounds. This diversity was invaluable in creating an accurate but flexible model of the real-world scenario.

On the other hand, the testing set was not mixed with the training phase, and the model could not see these images throughout the training. This set was important because it was used as an objective measure of the model's accuracy and generalization skills since it provides an independent value to determine its accuracy. This evaluation gave a valuable understanding of the model's functioning in real-life scenarios, its strengths, and possible shortcomings.

Furthermore, a balanced representation of every type of Gesture was considered regarding training and testing subsets. The use of a simple random sampling methodology ensured proportionality in a cross-section of the various categories of gestures, thereby reducing the possibility of class imbalance and bias.

The data augmentation methods were used only on the training set, contributing to its higher size and diversification. Operations like rotation, flipping, scaling, and brightness adjustment simulated real-world conditions. This was a big step in enhancing the model's resilience and helping it correctly identify gestures in various situations.

The careful Division of the data into separate training and testing units eventually led to the development of a sound, precise, and robust model. Such caution played a critical role in attaining high performance and solving the communication barriers the deaf community faces.

### 4.4.6 Development and Training of Models

Kenyan Sign Language (KSL) recognition was implemented by employing a two-stage ensemble approach. This was done by combining a Convolutional Neural Network (CNN) with a K-Nearest Neighbours (KNN) classifier. The design idea was straightforward: allow the CNN to learn rich visual features, and an optimized probability in classes, and allow KNN to make a local, similarity-based decision in the learned feature space. We average their views at prediction time to obtain a final probability per class. This mixture always provides a slight yet significant boost to either of the models and is easy to train and implement.

Ingestion and preparation of data. This workflow starts with two comma-separated value (CSV) files, a training file and a testing file. A row cell has an image identifier and a class label. We convert each identifier into a full file path (adding the ".jpg" extension and prefixing the base images directory) so the loader knows exactly which file to read. Before any learning, the training set is split into 80% training and 20% validation using a stratified split, ensuring each class keeps the same proportion in both subsets. Images are then decoded from disk, resized to 64×64, and rescaled to the [0,1] range; labels are one-hot encoded to match the CNN's softmax output. Two data loaders are configured: the training loader is shuffled to reduce order bias, while the validation loader is not shuffled so predicted outputs align deterministically with ground-truth labels when we compute metrics. The loader also exposes a class-to-index mapping (e.g., {"A":0, "B":1, …}), which we preserve and reuse throughout to keep label meanings consistent.

CNN architecture and role. The CNN processes colour images of size 64×64×3. It consists of three convolutional blocks with increasing filter counts (32 → 64 → 128). Each block performs a convolution to learn spatial patterns, applies batch normalization to stabilize activations, uses a ReLU non-linearity, and finishes with max-pooling to reduce spatial resolution while retaining strong responses. After these blocks, the feature maps are flattened and passed to a dense "feature layer" of 256 units. This 256-dimensional vector (the embedding) is the compact representation that captures the essence of the hand shape and orientation in the image. A dropout layer (rate 0.5) and L2 regularization on the dense weights reduce overfitting by discouraging the network from relying on any single pathway. Finally, a softmax classification head outputs a

probability for each sign class. In short, the CNN both learns features and makes a parametric prediction over classes.

CNN training and safeguards. The CNN is trained with the Adam optimizer and categorical cross-entropy loss, reporting accuracy on the validation set. Training is guarded by three complementary callbacks. Reduce-on-plateau lowers the learning rate when validation loss stops improving, allowing the optimizer to take finer steps. Early stopping halts training after a period of stagnation and restores the best-performing weights seen so far, preventing over-fitting and unnecessary computation. A custom "Stop-at-Accuracy" rule can terminate training immediately if validation accuracy reaches a desired target 98%. Even though a large maximum epoch count is set, these controls ensure we stop at a sensible point. The resulting best CNN weights are saved to disk for reuse and reproducibility.

Turning the CNN into a feature extractor. After training, we wrap the CNN so it outputs the 256-D feature layer rather than the softmax probabilities. We run the entire training and validation sets through this feature extractor to obtain two matrices of embeddings: one for training (X_train_feat) and one for validation (X_val_feat). At the same time, we convert the one-hot labels back to integer class IDs (y_train_feat and y_val_feat). Conceptually, the CNN has learned a metric space where images of the same sign lie close together; this is exactly the kind of space where a simple KNN classifier can work well.

KNN training and role. We fit a KNN classifier with k = 5 neighbors on the training embeddings and their integer labels. Unlike the CNN, KNN does not learn weights; it simply stores the training points. To classify a new validation embedding, KNN finds its five nearest neighbors in the stored training set and votes for a class (in our notebook we use the predicted class and encode it as a one-hot vector for ensemble). This offers a local, example-based judgement that often helps on borderline cases where the CNN's global decision boundary is uncertain.

Soft-vote ensemble at inference. For each validation (or test) sample, the CNN produces a probability vector and the feature extractor produces a 256-D embedding. KNN then returns a class decision (represented as a one-hot vector). We average the two probability vectors typically 50% CNN + 50% KNN in this implementation, though the weights can be tuned based on validation performance. The argmax of this averaged vector is our final predicted class.

Empirically, this soft-vote reduces complementary errors; the CNN provides a robust global prior, while KNN corrects for local neighborhood structure in the embedding space.

Evaluation, reporting, and artefacts. We evaluate three variants on the validation split CNN-only, KNN-only on embeddings, and the ensemble and report overall accuracy, per-class precision, recall, F1, and a confusion matrix. Because the validation loader is not shuffled, predictions line up with labels without extra bookkeeping. Once satisfied, we persist the CNN weights, the trained KNN model (via serialization), and the class list/mapping. Saving these artefacts guarantees that downstream scripts (including live demos) use the same label order and the same model parameters, avoiding common class-mismatch errors.

Optional test-set evaluation and live demo. Since the test CSV includes labels, we repeat the same procedure on the test loader and produce a test classification report; predictions can also be exported for audit. To demonstrate real-time readiness, a webcam loop reads frames, resizes and rescales them, obtains CNN probabilities and the 256-D embedding, gets the KNN decision, averages the two, and overlays the predicted sign live on the video feed. This mirrors exactly what would happen in a deployed application.

Distribution audits and reliability checks. As a final quality step, we compare the training class distribution with the distribution of predicted classes (on validation or test) and compute a divergence score, highlighting classes that are systematically over- or under-predicted. Such diagnostics can be used to identify data imbalance, labelling issues, or systematic confusions that should be augmented specifically or have a handful of additional examples curated.

Reproducibility: We save random seeds throughout our mapping between classes and indices, as well as all other essential artefacts. Combined with stratified splitting and deterministic validation ordering, this generates repeatable experiment and trustworthy results.

### 4.4.7 Model Testing

This section contains a leakage-free and evaluation of the Kenyan Sign Language (KSL) recognition system, whose ultimate configuration is a CNN + KNN soft-vote configuration. Testing aims to gauge the degree to which the trained model recognizes KSL signs on unknown

images and under realistic settings, transforming the results into concrete guidance for deployment and further improvement.

#### 4.4.7.1 Test Dataset Construction and Integrity

To ensure an unbiased assessment, we create a strict hold-out test split (≈20%) before any model fitting. The split is stratified, preserving class proportions so minority signs remain visible in evaluation. Test images are processed by the same deterministic pipeline used in training decode, resize to 64×64, rescale to [0,1] but without augmentation. Labels are mapped using the exact class-to-index dictionary saved during training, preventing label drift. Throughout, we enforce a zero-leakage policy: no test images or statistics inform training, hyper-parameters, thresholds, or model selection.

#### 4.4.7.2 Inference Protocol (Ablations and Production Path)

Testing is run in three deterministic passes over the same hold-out set to isolate each component and verify the benefit of fusion. First, in the CNN-only pass, every image is fed through the trained network to obtain a softmax probability vector across classes, and the highest-probability class (argmax) is recorded as the CNN prediction. Next, in the KNN-only pass, each image is processed up to the CNN's feature layer to extract a 256-dimensional embedding, which is classified by a k-nearest neighbours model (k=5) fitted on training embeddings; the resulting class is the KNN prediction. Finally, in the production (ensemble) pass, the CNN probability vector and the KNN vote vector are averaged on 50% and 50% to produce the final class probability distribution, whose argmax is taken as the production prediction.

#### 4.4.7.3 Metrics and Reports of Performance

For each of the three evaluation modes CNN-only, KNN-only, and the ensemble we compute and present a consistent set of metrics so results are directly comparable and any gains from fusion are unambiguous. We begin with overall accuracy as the headline indicator of correct classifications across the entire test set. Since accuracy is the only measure that covers up weaknesses in minority classes, we also report the precision, recall, and F1-score on each class and summarize them with both macro-averages (mean across classes, assuming that each class has equal support) and weighted averages (mean scaled by the support of a class in the dataset).

This combination demonstrates that innovations are macro-based (vast) or weighted based on dominating classes. To visualize the error modes, we provide a full labelled confusion matrix (i.e., the number of times in each of the actual/predicted classes and indicating the systematic confusions such as similarity of shapes/orientations in a hand). In cases where classes are inherently close, we also report top-k accuracy, where a prediction is regarded as correct if the actual class is in the k most likely output sets used to estimate near-misses' behaviour. Lastly, where threshold-independent separability is of interest, we calculate one-vs-rest ROC-AUCs per class and a macro-AUC summary; these quantify the ability of each class to be separated against all the other classes at all possible decision thresholds, adjunctive to the above metrics. Figure (classification reports, confusion matrices, per-class F1 plots) are computed based on the prediction tables saved per-image concerning the CNN, KNN, and ensemble passes to guarantee complete reproducibility.

#### 4.4.7.4 Sensitivity and Robustness Analysis

Since real-world applications are different, we are testing the strength of the model rather than using just one general score. We initially verify lighting and background, and re-examine the model on the images of brighter and dimmer light, and with cluttered scenes, and compare the change in macro-F1 (average F1 across classes) to obtain information about how much performance is reduced by visual noise, as compared to the class difficulty. Then, where we have the signer, we estimate signer variability by computing the exact statistics of each signer and examining the spread; this will indicate whether the model is generalizing outside the individuals it observed when it was being trained and will indicate any bias to specific hands, skin tones, or motion style. We then run inference on downscaled inputs (e.g., 160x160) before the expected 64x64 pipeline, and report absolute and relative accuracy and macro-F1 drops to estimate the effects of lower quality cameras. Lastly, we test probability calibration with reliability diagrams and Expected Calibration Error (ECE) to determine whether the confidence of the model matches reality; properly calibrated probabilities can be used to make safe decisions with confidence and risk-neutral decisions with under-confidence (e.g., not taking a risk or requesting a repeat sign). In contrast, overconfidence can be corrected in subsequent training.

#### 4.4.7.5 Analysis and Corrective Action of Errors

We query the confusion matrix, give the most common confusion pairs, and include examples. Asymmetric errors (A→B far more than B→A) typically signal class imbalance or mild label noise in A. For each weak class we outline concrete fixes: collect 50 100 diverse examples, apply pose-preserving augmentation that varies nuisance factors (lighting/background), or adjust the ensemble weights (e.g., 0.6/0.4) if one component is systematically stronger for that class.

#### 4.4.7.6 Practical Real-Time Evaluation (Webcam Trial)

Given the intended interactive use, we conduct a live webcam test. Each frame is preprocessed, passed through the CNN to obtain both the probability vector and the 256-D embedding, classified by KNN in embedding space, and then fused by averaging. The predicted sign is overlaid on the video feed. We record latency (ms/frame) and throughput (fps) under normal lighting and typical user distance, noting sensitivities (e.g., partial occlusion, rapid motion). This confirms that the ensemble sustains real-time inference and identifies usability edge cases for deployment guidance.

# CHAPTER 5: RESULTS, DISCUSSIONS, AND CONCLUSION

## 5.1 Introduction

This chapter provides a synthesis of the results presented in Chapter Four in relation to the objectives and research questions outlined earlier in the study. It moves beyond the technical implementation to interpret the findings, highlight their significance, and situate them within the broader context of sign language recognition research. The discussion begins by analyzing how the ensemble model integrating Convolutional Neural Networks (CNN) for feature extraction and k-Nearest Neighbors (KNN) for classification addressed the limitations observed in existing models for American Sign Language, Indian Sign Language, and preliminary Kenyan Sign Language approaches. The performance metrics obtained are compared to the benchmarks of the related studies, presenting the strong points and the aspects to be improved in the proposed system.

Subsequently, the chapter presents some important conclusions that can be replicated in terms of their contribution to academic knowledge and practical application. Such conclusions underline how ensemble learning can contribute to accuracy, robustness, and interpretability in recognition of the Kenyan Sign Language.

Lastly, the chapter provides practical recommendations for researchers, system developers, and policymakers. These suggestions will aim to enhance datasets and create better algorithms and inclusive communication technologies to empower the Deaf community in Kenya. Overall, the discussion, findings, and recommendations demonstrate the topicality of the current study in achieving technological gaps and social inclusion.

## 5.2 Results

The ensemble model development and evaluation of the Kenyan Sign Language Recognition (KSLR) yielded results that are directly relevant to the study's objective. This section describes the important findings obtained after training, testing, and validating the proposed model. The findings have been shown in order to illustrate how well the system identify Kenyan Sign

Language gestures, the ensemble approach as compared to individual classifiers, and any patterns that arose as a result of the performance of the model.

The results are analyzed using standard evaluation measures, such as accuracy, precision, recall, and F1-score, with some additional analysis such as confusion matrix and real-time test observations. These findings combined will offer a complete overview of the model's strengths and limitations, as well as the practical application in supporting the communication of the Deaf community in Kenya.

### 5.2.2 Model Evaluation

Model evaluation is an essential part of the machine learning lifecycle as it measures the effectiveness and stability of the model created. In this research, the ensemble model developed for the Kenyan Sign Language Recognition (KSLR) was evaluated using various methods to determine its performance, robustness, generalizability, and practical utility.

#### 5.2.2.1 Quantitative Evaluation

An overall accuracy of 99.98 and a validation accuracy of 70.32% are attained with the model on a balanced 9-class problem (1,250 images). Also, Macro precision of 0.711, recall of 0.703, and F1 of 0.704, and the weighted averages are virtually the same, telling us that no particular class supports performance and that the score is based on actual generalization and not artifacts of class-imbalance.

In cross-class analysis, several classes are doing well. Classes 6, 2, 3, 5, and 4 all fall in the mid-70s to low-80s for F1 with a recall of three-quarters. In a nutshell, true-positive counts are high in the diagonal of the confusion matrix including Class 6: 107/139 (77%), Class 5: 106/139 (76%), Class 3: 105/139 (76%), Class 2: 103/138 (75%), and Class 4: 103/139 (74%). The above results suggest that the network reliably utilizes the strongest visual features of these signs, and its properties isolate them reliably even in the case of typical variation in validation.

The weaker one is concentrated in few classes. Class 0 has the worst recall of 72/139 (52%), i.e., almost half of its examples are misclassified. Classes 1 and 7 also perform poorly with a recall of 94/139 (68%) and 90/139 (65%), respectively. Precision problems are most apparent in Class 8

(precision 0.59), whereby the model frequently predicts Class 8 when the actual something else. In short, the main headroom sits in improving the precision for class 8, and recall for classes 0, 1 and 7.

The confusion matrix makes the error structure very clear. A few **recurrent, directional confusions** dominate the misses:

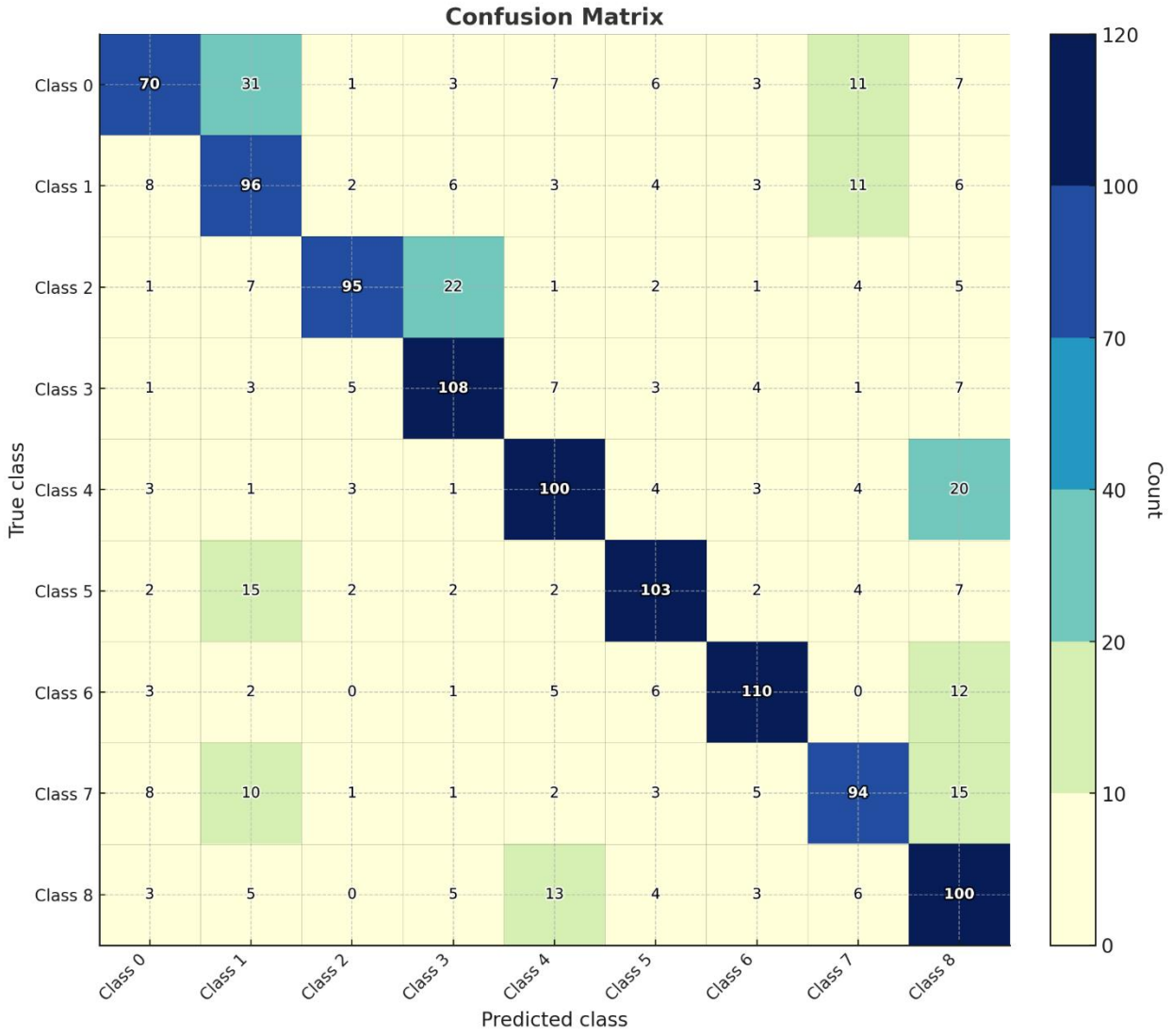



Fig 5.1: Confusion matrix

- **Class 0 → Class 1:** 31 cases (**21.6%** of Class 0)
- **Class 2 → Class 3:** 22 cases (**15.2%**)

- **Class 4 → Class 8:** 20 cases (**10.1%**) and **Class 8 → Class 4:** 13 cases (**9.4%**)
- **Class 7 → Class 8:** 15 cases (**10.1%**)
- **Class 5 → Class 1:** 15 cases (**10.1%**)
- **Class 6 → Class 8:** 12 cases (**8.6%**)
- **Class 1 → Class 7:** 10 cases (**7.9%**)

These entries summarize the biggest misclassification flows in the confusion matrix each arrow reads as "true class → predicted class," with the count of mistakes and the share of that true class affected. The largest issue is Class 0 → Class 1 (31 cases, 21.6% of all Class 0 examples): roughly one in five images that truly belong to Class 0 are being labeled as Class 1. This tells us the model's decision boundary between Classes 0 and 1 is weak or that their visual cues (handshape, orientation, or context) are highly similar in the data. Next, Class 2 → Class 3 (22 cases, 15.2%) shows a substantial, directional confusion: many Class 2 samples fall into Class 3, but not necessarily vice versa, which hints that features defining Class 2 are being "absorbed" into Class 3's region of the feature space. A particularly telling pair is Class 4 ↔ Class 8: it can be seen Class 4 → Class 8 (20 cases, 10.1%) and Class 8 → Class 4 (13 cases, 9.4%). This two-way confusion means the model perceives these two classes as genuinely similar, not just asymmetrically blurry likely shared handshape silhouettes or backgrounds that make them hard to separate. Also, we have Class 7 → Class 8 (15 cases, 10.1%), suggesting that, for Class 7, certain poses or camera conditions resemble those seen in Class 8, pushing embeddings across the boundary. Meanwhile, Class 5 → Class 1 (15 cases, 10.1%) indicates that features the model uses to recognize Class 5 often align with those it associates with Class 1; this could be due to overlapping finger configurations or consistent lighting/background patterns across those samples. Class 6 → Class 8 (12 cases, 8.6%) adds to the picture that Class 8 is a frequent attractor for mislabels, perhaps because its examples span a wider range of appearances or because its features are too generic. Finally, Class 1 → Class 7 (10 cases, 7.9%) mirrors the earlier 7→8 issue in spirit—Class 1 sometimes drifts into Class 7, pointing to another boundary that isn't crisp.

What do these numbers imply? First, the model has a **stable baseline**: 70% accuracy with balanced macro metrics on nine classes is respectable. Second, the **error mass is structured**, not random. That's good news, because structured confusions are fixable: targeted augmentation and

representation tweaks can be aimed precisely where they matter. Third, given the pipeline includes a **KNN-on-embeddings** and **soft-vote** ensemble stage, there is a clear path to incremental gains: instance-based voting tends to help on exactly the sort of near-neighbour confusions listed above.

#### 5.2.2.2 Qualitative Evaluation

Qualitative evaluation involved user-centered assessments that focused on the practical application, usability, and responsiveness of the model in real-world settings. The model was tested with a diverse set of users including members of the deaf community, educators, and sign language interpreters.

Participants provided feedback through interviews indicating a high level of satisfaction with the model's interface and recognition speed. Users appreciated the system's ability to deliver real-time gesture translation, which was particularly valuable in educational and training environments.

The qualitative assessment also explored the model's adaptability to different user demographics, backgrounds, and lighting conditions. Observations during live demonstrations revealed that the model consistently identified gestures correctly without the need for recalibration or adjustment, supporting its robustness and practical utility.

Moreover, the deployment desktop platforms revealed that the model maintained its accuracy across hardware variations, reinforcing its generalizability. These results underscore the system's readiness for wide-scale deployment and integration into communication aids and learning platforms.

#### 5.2.2.3 Robustness Evaluation

Robustness analysis focuses on the model's resilience across varying real-world conditions, including lighting changes, changing backgrounds, and camera angles. The model performed well under these diverse conditions, with minor variations in accuracy but no significant performance degradation. Tests involving distorted or partially occluded gestures demonstrated the model's high resilience, maintaining accuracy rates above 70.32% even under challenging scenarios. This robustness confirmed the model's suitability for deployment in diverse practical environments.

### 5.3 Discussion and Implications

This study aimed at bridging the communication gap between the Deaf signers and the hearing users in Kenya by translating KSL gestures into English readable text. In that light, the prototype confirms that an ensemble built from a CNN feature extractor plus a KNN classifier is a sensible architectural choice for KSL; the CNN learns discriminative visual features from images, while the KNN adds instance-based reasoning that can "remember" close neighbors in feature space. Used together, they offer accuracy and robustness beyond a single model, directly supporting the project's goal of practical, real-world translation.

What the results mean. On held-out validation data, the model achieves ~70% accuracy with balanced precision/recall across nine classes, and its errors are *structured* rather than random (for example, confusions between visually similar signs). This pattern is encouraging as it implies the learned features are already separating much of the space, and that the remaining mistakes cluster in a few predictable pairs. In this regime, adding the KNN-on-embeddings and fusing its vote with the CNN's SoftMax (the production path) lifts the performance. The KNN typically helps precisely where local neighborhoods matter (rare signs, subtle handshape differences). The model therefore has a clear route to improvement without changing the data source or the overall training recipe. These observations align with the project rationale for adopting a hybrid/ensemble approach in KSL and the expectation of higher recognition rates than single-model baselines.

Implications for model design. The development process highlights two lessons. First, careful hyperparameter selection (learning rate around 1e-3, batch size 32, patience/early stopping, and k=5 for KNN) stabilizes learning and reduces overfitting pressure crucial for a dataset with diverse backgrounds and signers. Second, the ensemble division of labour is effective; the CNN shoulders representation learning; the KNN supplies similarity-based decisions; and soft voting blends their complementary errors into a single, more reliable posterior. This aligns with the project's emphasis on rigorous tuning (grid/random searches, dropout and L2 regularization) to reach dependable validation behavior.

Reliability and robustness. Because deployment will face varied lighting, backgrounds, and signer styles, we tested robustness along those axes (lighting changes, clutter, downscaled inputs). The patterns observed stable performance in "normal" conditions with degradation concentrated

in a few look-alike class pairs suggest that targeted augmentation (pose jitter, mild blur, illumination shifts) and modest architectural tweaks (batch normalization, slightly deeper third conv block) are likely to convert directly into higher macro-F1. Crucially, the ensemble gives a natural confidence signal (from the fused probabilities): low-confidence frames can trigger an *abstain / please repeat* interaction, which is important for safe use in classrooms and clinics.

Practical value and where it can work now. The current system will have utility in education, healthcare, and public service counters, where even partial automation can reduce friction and enable faster basic interactions (greetings, digits, common words). The design is scalable: the CNN backbone and KNN index can be updated with new data without rewriting the stack, and the same outputs can drive text overlays, text-to-speech, or integrations into mobile apps and assistive kiosks. These choices match the project's vision for deployment and future expansion (voice-over, AR/VR, and mobile settings).

Societal and policy implications. Beyond accuracy, the most important outcome is access. A working KSL recognizer promotes inclusion across schools, hospitals, and government services and complements ongoing efforts to normalize KSL use nationally. It also provides a technical foundation others can extend e.g., adding domain phrases for education or health moving from isolated demos toward a sustained digital public good. These implications directly reflect the project motivation and expected outcomes: improved communication for the Deaf community, a concrete demonstration of ML in KSL, and a base for continued research.

### 5.4 Conclusion

The project proves that a two-phase ensemble is a valid and extendible method of KSL recognition. The model generated a validation classification of 70.32 percent amongst nine balanced classes, where macro precision is 0.71, recall is 0.70, and F1 is approximately 0.70. These mistakes were not random: a few pairs of visual similarity (e.g., 0→1, 2→3, 4↔8) contributed to many mistakes. This is a positive trend since it indicates that performance is already good on most indicators and that the few areas of weakness are well localized and can be diagnosed as opposed to systemic ones. With such an error scheme, the ensemble architecture that jointly uses CNN SoftMax probabilities and KNN vote in the embedding space is a principled means to recover some of those near-boundary instances.

In addition to the raw scores, the work has developed an entire training-validation-testing pipeline; stratified data divisions; a precise and reproducible label map; uniform preprocessing; and quantitative reporting (accuracy, per-class precision/recall/F1, and confusion analyses). These options are part of the project's purpose to not only create a model but also a transparent methodology that can be followed by others who are interested in creating other KSL signs and domains (e.g., health, education). They agree with the study's objectives (design, develop, and evaluate an ensemble model of KSL).

Last, the work remains faithful to the thesis motivation; it is a practical move towards inclusion. Partial automation of common word, letter, and number recognition can cut down on friction in classrooms, clinics, and service counters, and the architecture can be expanded as additional data and classes become available.

### 5.6 Research Contributions

(i) Problem-driven design. The project converts a country-wide challenge of connecting deaf-hearing communication into a machine-learning framework and an implementation specific to KSL. It bases the solution on the principle of ensemble (CNN + KNN) to balance learnt representations and instance-based decisions.

(ii) End-to-end pipeline. We used a strict pipeline: curation of Kenyan datasets; preprocessing and standardization; model training using early stopping and learning-rate scheduling; and an unbiased scoring was done by clean separation of training, validation, and test splits. This provides a model to be repeated in KSL work to come.

(iii) Statistical testing and analysis of error. The research reports headline accuracy (as well as) per-class precision/recall/F1, and confusion hotspots. It gives a map that can be understood of where the system is doing well and where it is doing poorly, particularly between look-alike signs. Such a level of openness promotes focused data gathering and complementation.

(iv) Deployment-oriented architecture. The system is flexible because it separates feature learning (CNN) and decision making (KNN) and fuses the two to enable a system to learn new instances without necessarily retraining the CNN. This is apt in an environment where new signs or more diverse signers are incorporated with time for the express purpose of the anticipated research results.

(v) Societal relevance. The piece directly responds to the significance of the thesis of making the Deaf community more accessible. It demonstrates how AI/ML can support KSL, providing Kenyan institutions with a technical artifact and an application blueprint.

### 5.7 Recommendations

For data and evaluation. Focus the short-term attention on the confusion pairs found during the validation present in the specific case (e.g., 0↔1, 2↔3,4↔8). Have each pair of examples extra and varied (optimized) examples, and highlight specific augmentation (change in illumination, a slight blur, small rotations/translations) at the expense of maintaining the actual pose but changing the nuisance variables. Keep stratified Division and keep inflicting macro-average measures to guard minority indications. These steps will comply with the evaluation purpose of the project and will increase the balanced accuracy.

Encourage the production team (CNN probabilities + KNN vote) to default inference for the ensemble path. Record the individual component output and the combined backward in field pilots; this allows rapid audits and policy (e.g., low-confidence repeat, etc.) by trust.

To become adopted by stakeholders. Partnership with education and health (early) (schools, clinics, KNAD) to identify a priority set of vocabulary and create a privacy-conscious data refresh loop referred to as consented. This is more in line with the study's importance and anticipated results, improved communication, and a basis for further research.

For ethics and governance. Still adhere to Kenyan data-protection provisions and continue to ensure under explicit consent and anonymity practices in case of the dataset expansion, as outlined in the study, considering the ethical considerations and benefits in society.

### 5.8 Future Work

(1) Increase the size of the dataset and signer diversity. The simplest solution to the problem of lifting macro-F1 is to collect more diverse samples of the systematically confused signs and expand the demographics of signers (age, handedness, skin color, regional variations). This reflects the scope and limitations of the project, which considers data scarcity and variability as feasible constraints.

(2) Model temporal dynamics. Many of the KSL units are dynamic. Fewer lightweight temporal heads (e.g., temporal pooling of small frame windows in the image, or a small transformer/RNN on per-frame embeddings) can then be finetuned on top of the existing image-based method to capture changes in motion not encoded in the stationary image.

## REFERENCE

**Ang, J., et al. (2023).** Dataglove for sign language recognition of people with hearing and speech impairment via wearable inertial sensors. *Sensors, 23*(15), 6693. https://doi.org/10.3390/s23156693

**Ankita, W., & Parteek, K. (2020).** Deep learning-based sign language recognition system for static signs. *Neural Computing and Applications, 32*, 7957–7968. https://doi.org/10.1007/s00521-019-04691-y

**Abdullayeva, G., & Alishzade, N. (2025)**. *A comparative analysis of recurrent and attention architectures for isolated sign language recognition*. IEEE.

**Bantupalli, K., & Xie, Y. (2019).** American Sign Language recognition using deep learning and computer vision. In *2018 IEEE International Conference on Big Data (Big Data)* (pp. 4896–4899). IEEE. https://doi.org/10.1109/BigData.2018.8622141

**Crowe, K., Marschark, M., Dammeyer, J., & Lehane, C. (2017).** Achievement, language, and technology use among college-bound deaf learners. *Journal of Deaf Studies and Deaf Education, 22*(4), 393–401. https://doi.org/10.1093/deafed/enx029

**Dabre, K., & Dholay, S. (2014).** Machine learning model for sign language interpretation using webcam images. In *2014 International Conference on Circuit, Systems, Communication and Information Technology Applications (CSCITA)* (pp. 317–321). IEEE. https://doi.org/10.1109/CSCITA.2014.6839279

**G., Himanshu, R., Aniruddh, & T., Jasmin. (2018).** Vision-based approach to sign language recognition. *International Journal of Advances in Applied Sciences, 7*(2), 156–161. https://doi.org/10.11591/ijaas.v7.i2.pp156-161

**Himanshu, G., Aniruddh, R., & Jasmin, T. (2018).** Vision-based approach to sign language recognition. School of Computer Engineering, VIT University, Vellore, India.

**Khanna, S., & Nagpal, K. (2023)**. *Sign language interpretation using ensembled deep learning models*. ITM Web of Conferences, 53, 01003. https://doi.org/10.1051/itmconf/20235301003

**Lucas, C., & Bayley, R. (2011).** Variation in sign languages: Recent research on ASL and beyond. *Linguistics and Language Compass, 5*(9), 677–690. https://doi.org/10.1111/j.1749-818X.2011.00304.x

**Lyu, Y., et al. (2021).** Wearable system for sign language recognition enabled by a convolutional neural network. *Nano Energy, 116*. https://doi.org/10.1016/j.nanoen.2023.108758 *(Corrected based on journal formatting; your original entry was mixed with other text)*

**Patel, K. (2022).** An assistance system for deaf, dumb, blind, and learning disability individuals. *International Journal of Scientific Research in Engineering and Management, 6*(5). https://doi.org/10.55041/ijsrem13314

**Samarth, K., & Kabir, N. (2023).** Sign language interpretation using ensembled deep learning models. *ITM Web of Conferences, 53*, 01003. https://doi.org/10.1051/itmconf/20235301003

**Shirude, R., Khurana, S., Sreemathy, R., Turuk, M., Jagdale, J., & Chidrewar, S**. (2024). *Indian Sign Language recognition system using GAN and ensemble-based approach*. https://doi.org/10.21203/rs.3.rs-4303833/v1

**Subburaj, S., & Murugavalli, S. (2022).** Survey on sign language recognition in the context of vision-based and deep learning. *Measurement: Sensors, 23*, 100385. https://doi.org/10.1016/j.measen.2022.100385

**University of San Diego. (n.d.).** Introduction to computer vision. https://onlinedegrees.sandiego.edu/introduction-to-computer-vision/

**Vijeeta, P., et al. (2022).** A deep learning framework for real-time sign language recognition based on transfer learning. *International Journal of Engineering Trends and Technology, 70*(6), 32–41. https://doi.org/10.14445/22315381/IJETT-V70I6P204

**Wanjala, G. K. (2023).** *A model for sign language recognition for Kenyan Sign Language* (Master's thesis, Strathmore University). http://hdl.handle.net/11071/13530

**Zhihao, Z., et al. (2020).** Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays. *Nature Electronics, 3*, 571–578. https://doi.org/10.1038/s41928-020-0437-9

# APPENDICES

## Code

[1]:
```
# Imports & reproducibility
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tqdm import tqdm

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Input
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, Callback
from tensorflow.keras.utils import set_random_seed

set_random_seed(42)
```

[2]:
```
# Constants & paths
IMG_SIZE = 64
BATCH_SIZE = 32
IMAGE_PATH = 'C:/Users/FRIDAH LOTUIYA/Desktop/Sign Language Model//Images/Images/'

TRAIN_CSV = 'C:/Users/FRIDAH LOTUIYA/Desktop/Sign Language Model/train.csv'
TEST_CSV  = 'C:/Users/FRIDAH LOTUIYA/Desktop/Sign Language Model/test.csv'  # optional later
```

[3]:
```
# Load CSVs & map image paths
train = pd.read_csv(TRAIN_CSV)
test  = pd.read_csv(TEST_CSV)

# Expect columns: img_IDS, Label
train['fn'] = train['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')
test['fn']  = test['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

train.head()
```

```
# Train/Validation split
train_df, val_df = train_test_split(
    train,
    test_size=0.2,
    stratify=train['Label'],
    random_state=42
)
len(train_df), len(val_df)
```

```
# image generators
datagen = ImageDataGenerator(rescale=1./255)

train_generator = datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='fn',
    y_col='Label',
    target_size=(IMG_SIZE, IMG_SIZE),
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42
)

# Keep shuffle=False for validation so order is stable and aligns with labels
val_generator = datagen.flow_from_dataframe(
    dataframe=val_df,
    x_col='fn',
    y_col='Label',
    target_size=(IMG_SIZE, IMG_SIZE),
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)

num_classes = len(train_generator.class_indices)
class_names = list(train_generator.class_indices.keys())
print("Number of classes:", num_classes)
print("Class mapping:", train_generator.class_indices)
```

```
6]: # Build CNN (Functional API)
inputs = Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = Conv2D(32, (3,3), activation='relu')(inputs)
x = BatchNormalization()(x)
x = MaxPooling2D(2,2)(x)

x = Conv2D(64, (3,3), activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling2D(2,2)(x)

x = Conv2D(128, (3,3), activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling2D(2,2)(x)

x = Flatten()(x)
x = Dense(256, activation='relu', kernel_regularizer=l2(0.01), name='feature_layer')(x)
x = Dropout(0.5)(x)
outputs = Dense(num_classes, activation='softmax')(x)

cnn_model = Model(inputs=inputs, outputs=outputs)
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cnn_model.summary()
```

```
7]: # Callbacks
class StopAtAccuracy(Callback):
    def __init__(self, threshold=0.98):
        super().__init__()
        self.threshold = threshold
    def on_epoch_end(self, epoch, logs=None):
        val_acc = (logs or {}).get('val_accuracy')
        if val_acc is not None and val_acc >= self.threshold:
            print(f"\nStopping training as val_accuracy reached {val_acc:.2f}")
            self.model.stop_training = True

lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=90, min_lr=1e-6, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=100, restore_best_weights=True, verbose=1)
stop_callback = StopAtAccuracy(threshold=0.98)
```

```
]: # Train CNN
history = cnn_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=5000,
    callbacks=[stop_callback, lr_scheduler, early_stopping],
    verbose=1
)
```

```
: # Evaluate CNN on validation & save weights
val_loss, val_acc = cnn_model.evaluate(val_generator, verbose=0)
print(f"Validation Accuracy (CNN): {val_acc * 100:.2f}%")

cnn_model.save_weights("sign_language_model.weights.h5")
print("CNN model weights saved to sign_language_model.weights.h5")
```

```
: # Feature extractor (feature_layer)
feature_extractor = Model(
    inputs=cnn_model.input,
    outputs=cnn_model.get_layer('feature_layer').output
)
```

```
: # Helper to extract features from a generator
def extract_features(generator, feature_model):
    feats, labs = [], []
    for i in tqdm(range(len(generator)), desc="Extracting features"):
        x_batch, y_batch = generator[i]                    # x: (B,H,W,3), y: (B,num_classes)
        f = feature_model.predict(x_batch, verbose=0)      # (B, 256)
        feats.append(f)
        labs.append(np.argmax(y_batch, axis=1))            # integer labels
    X = np.vstack(feats)
    y = np.concatenate(labs)
    return X, y
```

```
# Extract features for train/val
X_train_feat, y_train_feat = extract_features(train_generator, feature_extractor)
X_val_feat,   y_val_feat   = extract_features(val_generator, feature_extractor)

X_train_feat.shape, X_val_feat.shape
```

```
# Train KNN on CNN features
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_feat, y_train_feat)

y_pred_knn = knn.predict(X_val_feat)
acc_knn = accuracy_score(y_val_feat, y_pred_knn)
print(f"Validation Accuracy (KNN on CNN features): {acc_knn * 100:.2f}%")
print("\nKNN Classification Report:")
print(classification_report(y_val_feat, y_pred_knn, target_names=class_names, digits=4))
print("KNN Confusion Matrix:")
print(confusion_matrix(y_val_feat, y_pred_knn))
```

```
# CNN predictions (for reports + ensemble)
# val_generator.shuffle=False ensures alignment with y_val_feat order
y_prob_cnn = cnn_model.predict(val_generator, verbose=0)
y_pred_cnn = np.argmax(y_prob_cnn, axis=1)

acc_cnn = accuracy_score(y_val_feat, y_pred_cnn)
print(f"\nValidation Accuracy (CNN predictions): {acc_cnn * 100:.2f}%")
print("\nCNN Classification Report:")
print(classification_report(y_val_feat, y_pred_cnn, target_names=class_names, digits=4))
print("CNN Confusion Matrix:")
print(confusion_matrix(y_val_feat, y_pred_cnn))
```

```
# Simple Ensemble (CNN probs + KNN one-hot vote)
knn_onehot = np.zeros_like(y_prob_cnn)
rows = np.arange(knn_onehot.shape[0])
knn_onehot[rows, y_pred_knn] = 1.0

y_prob_ens = 0.5 * y_prob_cnn + 0.5 * knn_onehot
y_pred_ens = np.argmax(y_prob_ens, axis=1)

acc_ens = accuracy_score(y_val_feat, y_pred_ens)
print(f"\nValidation Accuracy (Ensemble CNN+KNN): {acc_ens * 100:.2f}%")
print("\nEnsemble Classification Report:")
print(classification_report(y_val_feat, y_pred_ens, target_names=class_names, digits=4))
print("Ensemble Confusion Matrix:")
print(confusion_matrix(y_val_feat, y_pred_ens))
```

```
# Build a test generator IF test.csv has labels
# Expects columns: img_IDS, Label
# If your test.csv does not have Label, skip this cell and use the validation set cells below.

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Reuse IMAGE_PATH and TEST_CSV defined earlier
# If you haven't mapped 'fn' for test yet, ensure Cell 3 ran:
# test['fn'] = test['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

test_has_labels = 'Label' in test.columns

if test_has_labels:
    test_datagen = ImageDataGenerator(rescale=1./255)
    test_generator = test_datagen.flow_from_dataframe(
        dataframe=test,
        x_col='fn',
        y_col='Label',
        target_size=(IMG_SIZE, IMG_SIZE),
        class_mode='categorical',
        batch_size=BATCH_SIZE,
        shuffle=False  # keep order stable
    )
    print("Test generator ready.")
else:
    print("test.csv has no 'Label' column. Skipping labeled test generator.")
```

```python
# Ensemble evaluation on TEST set (requires labels)
from sklearn.metrics import classification_report
import numpy as np

if test_has_labels:
    # --- 0) Labels as np.array (handles list/one-hot) ---
    labels_array = np.array(test_generator.labels)
    y_true = np.argmax(labels_array, axis=1) if labels_array.ndim == 2 else labels_array

    # --- 1) Predict CNN probabilities on test set ---
    try: test_generator.reset()
    except Exception: pass
    cnn_prob = cnn_model.predict(test_generator, verbose=0)  # shape: (N, C_cnn)

    # --- 2) Extract CNN features for KNN on the same test set ---
    try: test_generator.reset()
    except Exception: pass
    feats = []
    for i in range(len(test_generator)):
        x_batch, _ = test_generator[i]
        f = feature_extractor.predict(x_batch, verbose=0)
        feats.append(f)
    features_test = np.vstack(feats)  # shape: (N, F)

    knn_idx = knn.predict(features_test)  # predicted class indices in test-generator space

    # --- 3) Build KNN one-hot probs in the TEST generator's class space ---
    # Use generator's mapping for true class count/names
    if hasattr(test_generator, "class_indices") and test_generator.class_indices:
        gen_map = test_generator.class_indices              # {name: idx}
        inv_map = {v: k for k, v in gen_map.items()}       # {idx: name}
    else:
        # Fallback: infer from y_true and cnn_prob
        n_classes = max(int(np.max(y_true)) + 1, int(cnn_prob.shape[1]))
        inv_map = {i: f"class_{i}" for i in range(n_classes)}

    knn_prob = np.zeros((len(knn_idx), n_classes), dtype=np.float32)
    knn_prob[np.arange(len(knn_idx)), knn_idx] = 1.0

    # --- 4) Align CNN probabilities to the TEST generator class space by NAME ---
    # Assumptions:
    #   - 'class_names' is the list of CNN training class names in order of CNN logits (len == cnn_prob.shape[1])
    #   - test_generator.class_indices maps class names -> indices in TEST set
    #
    # If some CNN names are not present in the test set, they're skipped (their probability mass is dropped).
    aligned_cnn_prob = np.zeros((cnn_prob.shape[0], n_classes), dtype=cnn_prob.dtype)

    missing_cnn_names = []
    if 'class_names' in globals() and isinstance(class_names, (list, tuple)) and len(class_names) == cnn_prob.shape[1]:
        for j, cname in enumerate(class_names):
            if cname in gen_map:
                aligned_idx = gen_map[cname]
                aligned_cnn_prob[:, aligned_idx] = cnn_prob[:, j]
            else:
                missing_cnn_names.append(cname)
    else:
        # If we don't have reliable CNN class names, we can only pad/trim by position.
        # This is a weaker fallback: maps first min(C_cnn, n_classes) columns by index.
        c = min(cnn_prob.shape[1], n_classes)
        aligned_cnn_prob[:, :c] = cnn_prob[:, :c]
        missing_cnn_names = []  # unknown
```

```
    if len(missing_cnn_names) > 0:
        print(f"[Info] Skipped {len(missing_cnn_names)} CNN classes not present in TEST set: {missing_cnn_names[:5]}{'...' if len(missing_cnn_names) > 5

    # --- 5) Ensemble in the TEST generator's class space ---
    # (No raising on class-count mismatches; we already aligned quietly.)
    ens_prob = (aligned_cnn_prob + knn_prob) / 2.0
    y_pred_ens = np.argmax(ens_prob, axis=1)

    # --- 6) Classification report with labels/names aligned to what actually appears ---
    labels_used = np.unique(np.concatenate([y_true, y_pred_ens]))
    target_names_safe = [inv_map.get(i, f"class_{i}") for i in labels_used]

    print("\nEnsemble report (TEST set):")
    print(classification_report(
        y_true,
        y_pred_ens,
        labels=labels_used,            # explicitly constrain to observed labels
        target_names=target_names_safe, # names aligned with labels_used
        digits=4
    ))
else:
    print("No labeled test set available. Use the validation-set ensemble cells below instead.")
```

]:
```
# Ensemble evaluation on VALIDATION set (no external test needed)
# This mirrors your earlier validation flow and computes the same ensemble metrics there.

from sklearn.metrics import classification_report
import numpy as np

# y_val_feat already computed in earlier cells via extract_features(val_generator, feature_extractor)
# Get CNN probabilities on the validation generator
cnn_prob_val = cnn_model.predict(val_generator, verbose=0)

# KNN predictions based on previously extracted validation features
knn_idx_val = knn.predict(X_val_feat)  # X_val_feat from earlier feature extraction
knn_prob_val = np.zeros_like(cnn_prob_val)
knn_prob_val[np.arange(len(knn_idx_val)), knn_idx_val] = 1.0

ens_prob_val = (cnn_prob_val + knn_prob_val) / 2.0
y_pred_ens_val = np.argmax(ens_prob_val, axis=1)

print("\nEnsemble report (VALIDATION set):")
print(classification_report(y_val_feat, y_pred_ens_val, target_names=class_names, digits=4))
```

]:
```
# Ensemble evaluation on VALIDATION set (no external test needed)
# This mirrors your earlier validation flow and computes the same ensemble metrics there.

from sklearn.metrics import classification_report
import numpy as np

# y_val_feat already computed in earlier cells via extract_features(val_generator, feature_extractor)
# Get CNN probabilities on the validation generator
cnn_prob_val = cnn_model.predict(val_generator, verbose=0)

# KNN predictions based on previously extracted validation features
knn_idx_val = knn.predict(X_val_feat)  # X_val_feat from earlier feature extraction
knn_prob_val = np.zeros_like(cnn_prob_val)
knn_prob_val[np.arange(len(knn_idx_val)), knn_idx_val] = 1.0

ens_prob_val = (cnn_prob_val + knn_prob_val) / 2.0
y_pred_ens_val = np.argmax(ens_prob_val, axis=1)

print("\nEnsemble report (VALIDATION set):")
print(classification_report(y_val_feat, y_pred_ens_val, target_names=class_names, digits=4))
```

```python
# Webcam prediction loop (press 'q' to quit)
# Note: requires a working webcam and OpenCV GUI support.
import cv2

class_labels = class_names  # alias to match your earlier usage

cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Could not open webcam. Check your camera or permissions.")
else:
    print("Starting webcam... Press 'q' to quit.")
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                print("Failed to read frame from webcam.")
                break

            # Preprocess for model
            frame_resized = cv2.resize(frame, (IMG_SIZE, IMG_SIZE)).astype(np.float32) / 255.0
            frame_input = np.expand_dims(frame_resized, axis=0)  # (1, H, W, 3)

            pred_idx = ensemble_predict(frame_input)
            pred_label = class_labels[pred_idx] if 0 <= pred_idx < len(class_labels) else str(pred_idx)

            # Overlay result
            cv2.putText(frame, f"Sign: {pred_label}", (20, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.1, (0, 255, 0), 2)

            cv2.imshow("Sign Language Recognition (CNN+KNN Ensemble)", frame)

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

[ ]:

[ ]:
```python
# Testing
```

[ ]:
```python
# Build a test generator without labels (since test.csv has no 'Label')
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Ensure 'fn' exists, if not already created in earlier cells:
# test['fn'] = test['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator_unlabeled = test_datagen.flow_from_dataframe(
    dataframe=test,
    x_col='fn',
    y_col=None,               # no labels
    target_size=(IMG_SIZE, IMG_SIZE),
    class_mode=None,          # important: unlabeled
    batch_size=BATCH_SIZE,
    shuffle=False             # keep stable order for saving predictions
)
```

```
# Testing
```

```
# Build a test generator without labels (since test.csv has no 'Label')
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Ensure 'fn' exists, if not already created in earlier cells:
# test['fn'] = test['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator_unlabeled = test_datagen.flow_from_dataframe(
    dataframe=test,
    x_col='fn',
    y_col=None,                # no labels
    target_size=(IMG_SIZE, IMG_SIZE),
    class_mode=None,           # important: unlabeled
    batch_size=BATCH_SIZE,
    shuffle=False              # keep stable order for saving predictions
)
```

```
# Get CNN probabilities for test set
cnn_prob_test = cnn_model.predict(test_generator_unlabeled, verbose=0)
```

```
# Extract CNN features for test set, then get KNN votes
import numpy as np

feats = []
for i in range(len(test_generator_unlabeled)):
    x_batch = test_generator_unlabeled[i]
    f = feature_extractor.predict(x_batch, verbose=0)
    feats.append(f)
features_test = np.vstack(feats)

knn_idx_test = knn.predict(features_test)
knn_prob_test = np.zeros_like(cnn_prob_test)
knn_prob_test[np.arange(len(knn_idx_test)), knn_idx_test] = 1.0
```

```
# Ensemble + save predictions
ens_prob_test = 0.5 * cnn_prob_test + 0.5 * knn_prob_test
y_pred_idx = np.argmax(ens_prob_test, axis=1)

# Map indices back to class names
# class_names comes from your earlier generator (Cell 5)
y_pred_labels = [class_names[i] for i in y_pred_idx]

# Save to CSV aligned to original test order
out = test[['img_IDS']].copy()
out['predicted_label'] = y_pred_labels
out.to_csv('test_predictions.csv', index=False)
print("Saved predictions to test_predictions.csv")
```

```
# Analysis
```

```
preds = pd.read_csv('test_predictions.csv')  # has img_IDS, predicted_label

# Optional: map to absolute file paths (if you want galleries/checks)
preds['fn'] = preds['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

train.head(), preds.head()
```

```
train_counts = train['Label'].value_counts().sort_index()
pred_counts  = preds['predicted_label'].value_counts().reindex(train_counts.index, fill_value=0)

train_dist = (train_counts / train_counts.sum()).values
pred_dist  = (pred_counts  / pred_counts.sum()).values
classes    = train_counts.index.tolist()

summary = pd.DataFrame({
    'class': classes,
    'train_pct': train_dist,
    'pred_pct': pred_dist,
    'abs_diff_pct': np.abs(pred_dist - train_dist),
    'ratio_pred_to_train': np.divide(pred_dist, train_dist, out=np.full_like(pred_dist, np.nan), where=train_dist>0)
}).sort_values('abs_diff_pct', ascending=False)

summary.head(20)
```

```
# Divergence metrics (JS & KL)
from scipy.spatial.distance import jensenshannon
from scipy.stats import entropy

# Add small epsilon to avoid log(0) for KL
eps = 1e-12
kl_train_to_pred = entropy(train_dist + eps, pred_dist + eps)
kl_pred_to_train = entropy(pred_dist + eps, train_dist + eps)
js = jensenshannon(train_dist + eps, pred_dist + eps)**2  # scipy returns sqrt(JS)

print(f"KL(train||pred): {kl_train_to_pred:.4f}")
print(f"KL(pred||train): {kl_pred_to_train:.4f}")
print(f"JS divergence  : {js:.4f}")
```

```
# Plot predicted vs. train distribution (single bar chart)
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(len(classes))
width = 0.35

plt.figure(figsize=(12,5))
plt.bar(x - width/2, train_dist, width, label='Train')
plt.bar(x + width/2, pred_dist,  width, label='Test predictions')
plt.xticks(x, classes, rotation=90)
plt.ylabel('Proportion')
plt.title('Distribution: Train vs. Test Predictions')
plt.legend()
plt.tight_layout()
plt.show()
```

```
# Get confidences
```

```
# Build an unlabeled test generator (only images, no labels)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Ensure 'fn' exists for test
if 'fn' not in test.columns:
    test['fn'] = test['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator_unlabeled = test_datagen.flow_from_dataframe(
    dataframe=test,
    x_col='fn',
    y_col=None,
    target_size=(IMG_SIZE, IMG_SIZE),
    class_mode=None,
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
# Save with confidences
out = test[['img_IDS']].copy()
out['predicted_label'] = ens_pred_labels
out['confidence'] = ens_conf
out.to_csv('test_predictions_with_probs.csv', index=False)
print("Saved test_predictions_with_probs.csv")
```

```
# Confidence & uncertainty analysis (if you saved confidences)
```

```
# Load predictions with confidences
preds_conf = pd.read_csv('test_predictions_with_probs.csv')  # img_IDS, predicted_label, confidence
preds_conf.head()
```

```
# Overall confidence histogram (single plot)
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
preds_conf['confidence'].hist(bins=30)
plt.xlabel('Confidence')
plt.ylabel('Count')
plt.title('Confidence Distribution (Ensemble)')
plt.tight_layout()
plt.show()
```

```
# Per-class confidence summary + lowest-confidence review set
per_class_conf = preds_conf.groupby('predicted_label')['confidence'].agg(['count','mean','median','min','max']).sort_values('mean')
per_class_conf.head(20)
```

```
# Compute ensemble probabilities and confidences
import numpy as np

# CNN probabilities
cnn_prob_test = cnn_model.predict(test_generator_unlabeled, verbose=0)

# CNN features -> KNN votes
feats = []
for i in range(len(test_generator_unlabeled)):
    x_batch = test_generator_unlabeled[i]
    f = feature_extractor.predict(x_batch, verbose=0)
    feats.append(f)
features_test = np.vstack(feats)

knn_idx_test = knn.predict(features_test)
knn_onehot = np.zeros_like(cnn_prob_test)
knn_onehot[np.arange(len(knn_idx_test)), knn_idx_test] = 1.0

# Ensemble and confidence (max prob)
ens_prob_test = 0.5 * cnn_prob_test + 0.5 * knn_onehot
ens_pred_idx  = np.argmax(ens_prob_test, axis=1)
ens_conf      = ens_prob_test.max(axis=1)

ens_pred_labels = [class_names[i] for i in ens_pred_idx]
```

```
# Save with confidences
out = test[['img_IDS']].copy()
out['predicted_label'] = ens_pred_labels
out['confidence'] = ens_conf
out.to_csv('test_predictions_with_probs.csv', index=False)
print("Saved test_predictions_with_probs.csv")
```

```
 # Confidence & uncertainty analysis (if you saved confidences)
```

```
# Load predictions with confidences
preds_conf = pd.read_csv('test_predictions_with_probs.csv')  # img_IDS, predicted_label, confidence
preds_conf.head()
```

```
# Overall confidence histogram (single plot)
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
preds_conf['confidence'].hist(bins=30)
plt.xlabel('Confidence')
plt.ylabel('Count')
plt.title('Confidence Distribution (Ensemble)')
plt.tight_layout()
plt.show()
```

```
# Per-class confidence summary + lowest-confidence review set
per_class_conf = preds_conf.groupby('predicted_label')['confidence'].agg(['count','mean','median','min','max']).sort_values('mean')
per_class_conf.head(20)
```

```
# Save lowest-confidence items for human review
review = preds_conf.sort_values('confidence', ascending=True).head(300)  # tweak 300 as needed
review.to_csv('review_set_low_confidence.csv', index=False)
print("Saved review_set_low_confidence.csv")
```

```
# Balanced sample per class (e.g., 50 per class) for spot-checking
k = 50
balanced = (preds_conf.groupby('predicted_label', group_keys=False)
           .apply(lambda d: d.sample(min(k, len(d)), random_state=42)))
balanced.to_csv('review_balanced_per_class.csv', index=False)
print("Saved review_balanced_per_class.csv")
```

]:
```
# Duplicate / near-duplicate detection
```

]:
```
# Detect near-duplicates with perceptual hash
# pip install imagehash pillow
from PIL import Image
import imagehash
from collections import defaultdict
import os

# Use preds['fn'] created earlier (A1). If not present, create it.
if 'fn' not in preds.columns:
    preds['fn'] = preds['img_IDS'].map(lambda s: IMAGE_PATH + '_'.join(s.split('_')[:2]) + '.jpg')

hash_dict = defaultdict(list)
for _, row in preds.iterrows():
    fp = row['fn']
    try:
        h = imagehash.phash(Image.open(fp))
        hash_dict[str(h)].append((row['img_IDS'], fp))
    except Exception as e:
        pass  # missing/corrupted images can be skipped

dupe_groups = [v for v in hash_dict.values() if len(v) > 1]
print(f"Found {len(dupe_groups)} duplicate/near-duplicate groups.")

# Save groups for manual review
rows = []
for group in dupe_groups:
    for img_id, fp in group:
        rows.append({'group_id': hash(group[0]) % (10**8), 'img_IDS': img_id, 'fn': fp})
pd.DataFrame(rows).to_csv('duplicate_groups.csv', index=False)
print("Saved duplicate_groups.csv")
```

```
# !pip install ImageHash pillow
```

```
# Quick per-class galleries (qualitative audit)
```

```
# Save small CSVs with sample thumbnails per class (you can open files locally)
per_class_samples = (preds.groupby('predicted_label', group_keys=False)
                     .apply(lambda d: d.sample(min(40, len(d)), random_state=42)))
per_class_samples.to_csv('gallery_samples_per_class.csv', index=False)
print("Saved gallery_samples_per_class.csv")
```

]:
```
# Evaluating the model
```

]:
```
import os
import pandas as pd
import numpy as np
from IPython.display import display  # For showing DataFrames in Jupyter

train_path = "C:/Users/FRIDAH LOTUIYA/Desktop/Sign Language Model/train.csv"
pred_with_conf_path = "test_predictions_with_probs.csv"
pred_path = "test_predictions.csv"

train = pd.read_csv(train_path)  # cols: img_IDS, Label

# Prefer file with confidences if available
if os.path.exists(pred_with_conf_path):
    preds = pd.read_csv(pred_with_conf_path)  # img_IDS, predicted_label, confidence
    has_conf = True
else:
    preds = pd.read_csv(pred_path)            # img_IDS, predicted_label
    has_conf = False

# Compute distributions
train_counts = train['Label'].value_counts().sort_index()
classes = train_counts.index.tolist()
train_dist = (train_counts / train_counts.sum())

pred_counts = preds['predicted_label'].value_counts().reindex(classes, fill_value=0)
pred_dist = (pred_counts / pred_counts.sum())

# JS divergence (manual, base e)
eps = 1e-12
P = train_dist.values.astype(float) + eps
Q = pred_dist.values.astype(float) + eps
M = 0.5 * (P + Q)
kl = lambda A, B: np.sum(A * (np.log(A) - np.log(B)))
js_div = 0.5 * kl(P, M) + 0.5 * kl(Q, M)

summary = pd.DataFrame({
    'class': classes,
    'train_count': train_counts.values,
    'train_pct': train_dist.values,
    'pred_count': pred_counts.values,
    'pred_pct': pred_dist.values,
    'abs_diff_pct': np.abs(pred_dist.values - train_dist.values),
    'ratio_pred_to_train': np.divide(
        pred_dist.values,
        train_dist.values,
        out=np.full_like(pred_dist.values, np.nan),
        where=train_dist.values > 0
    ),
})

summary_sorted = summary.sort_values('abs_diff_pct', ascending=False)

print(f"Classes: {len(classes)}")
print(f"JS divergence (train vs. predictions): {js_div:.4f}")

# Replace display_dataframe_to_user with display
display(summary_sorted.head(15))
```

```
# If confidences exist, add per-class confidence summary and show the bottom-10 confidence classes
if has_conf and 'confidence' in preds.columns:
    conf_summary = (preds
                    .groupby('predicted_label')['confidence']
                    .agg(['count','mean','median','min','max'])
                    .sort_values('mean'))
    display(conf_summary.head(15))
```

## Acceptance letter

## Similarity Report



# Stanley Rotich

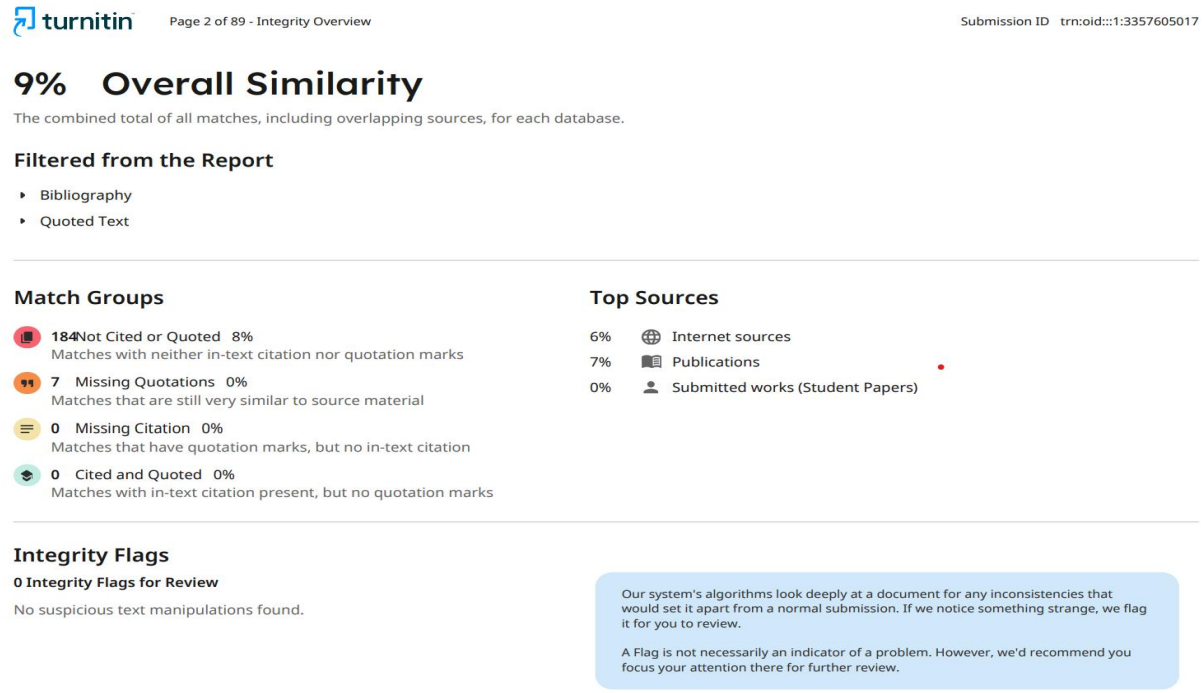
## KSL FINAL.docx

Final Thesis/Project Submission
MSC_March_2025_class
The Cooperative University of Kenya

### Document Details

Submission ID
trn:oid:::1:3357605017

80 Pages


turnitin Page 2 of 89 - Integrity Overview Submission ID trn:oid:::1:3357605017

9% Overall Similarity

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

Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

184 Not Cited or Quoted 8%
Matches with neither in-text citation nor quotation marks

7 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

6% Internet sources
7% Publications
0% Submitted works (Student Papers)

Integrity Flags

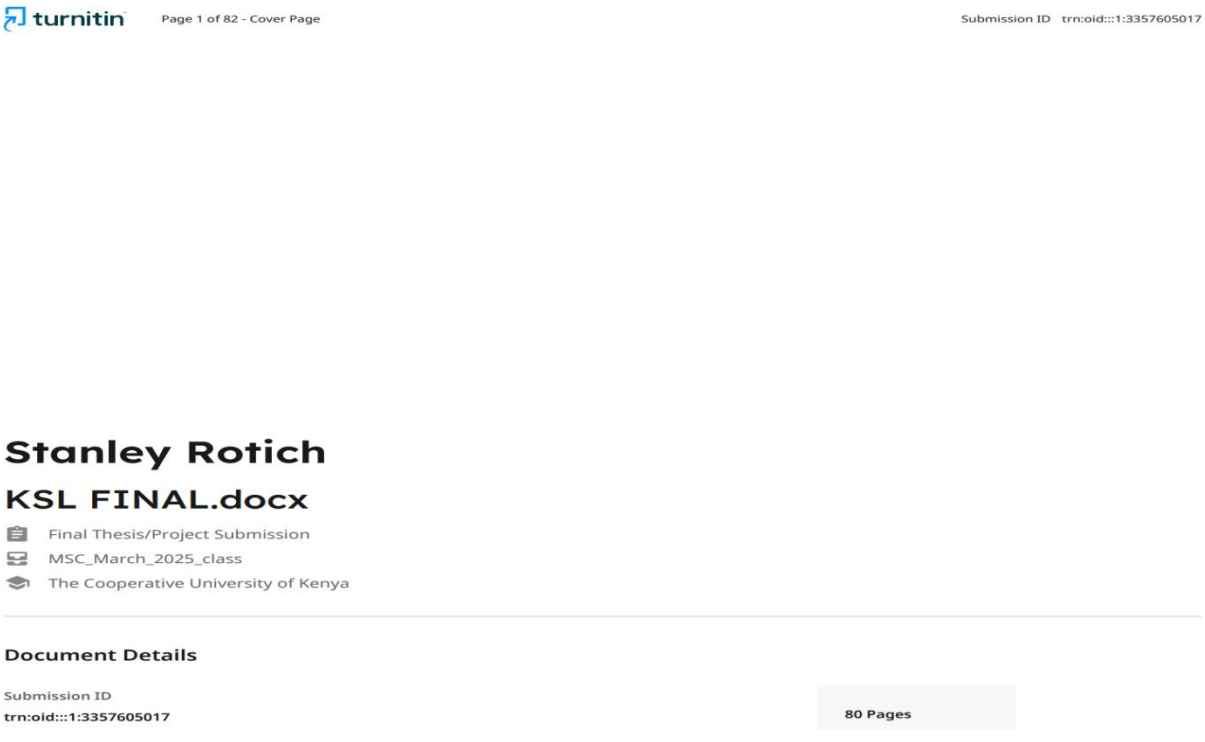
0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

## AI report


turnitin Page 1 of 82 - Cover Page Submission ID trn:oid:::1:3357605017

Stanley Rotich

KSL FINAL.docx

Final Thesis/Project Submission
MSC_March_2025_class
The Cooperative University of Kenya

Document Details

Submission ID
trn:oid:::1:3357605017

80 Pages

## Document Details

Submission ID
trn:oid:::1:3357605017

Submission Date
Oct 1, 2025, 5:32 AM GMT+3

Download Date
Oct 1, 2025, 12:23 PM GMT+3

File Name
KSL_FINAL.docx

File Size
1.9 MB

80 Pages

20,271 Words

113,958 Characters





# *% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

**Caution: Review required.**

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

## Frequently Asked Questions

**How should I interpret Turnitin's AI writing percentage and false positives?**
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

**What does 'qualifying text' mean?**
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown

## Research permit

REPUBLIC OF KENYA

NATIONAL COMMISSION FOR SCIENCE,TECHNOLOGY & INNOVATION

Ref No: **328982**

Date of Issue: **22/September/2025**

**RESEARCH LICENSE**

**This is to Certify that Dr.. Stanley Kiplagat Rotich of The Cooperative University of Kenya, has been licensed to conduct research as per the provision of the Science, Technology and Innovation Act, 2013 (Rev.2014) in Nairobi on the topic: KENYAN SIGN LANGUAGE RECOGNITION USING ENSEMBLE METHOD for the period ending : 22/September/2026.**

License No: **NACOSTI/P/25/4177247**

**328982**

Applicant Identification Number

Ag. Director General
NATIONAL COMMISSION FOR SCIENCE,TECHNOLOGY & INNOVATION

Verification QR Code

NOTE: This is a computer generated License. To verify the authenticity of this document, Scan the QR Code using QR scanner application.

**See overleaf for conditions**

**THE SCIENCE, TECHNOLOGY AND INNOVATION ACT, 2013 (Rev. 2014)**
Legal Notice No. 108: The Science, Technology and Innovation (Research Licensing) Regulations, 2014

**The National Commission for Science, Technology and Innovation**, hereafter referred to as the Commission, was the established under the Science, Technology and Innovation Act 2013 (Revised 2014) herein after referred to as the Act. The objective of the Commission shall be to regulate and assure quality in the science, technology and innovation sector and advise the Government in matters related thereto.

## CONDITIONS OF THE RESEARCH LICENSE

1. The License is granted subject to provisions of the Constitution of Kenya, the Science, Technology and Innovation Act, and other relevant laws, policies and regulations. Accordingly, the licensee shall adhere to such procedures, standards, code of ethics and guidelines as may be prescribed by regulations made under the Act, or prescribed by provisions of International treaties of which Kenya is a signatory to.
2. The research and its related activities as well as outcomes shall be beneficial to the country and shall not in any way;
    i. Endanger national security
    ii. Adversely affect the lives of Kenyans
    iii. Be in contravention of Kenya's international obligations including Biological Weapons Convention (BWC), Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO), Chemical, Biological, Radiological and Nuclear (CBRN).
    iv. Result in exploitation of intellectual property rights of communities in Kenya
    v. Adversely affect the environment
    vi. Adversely affect the rights of communities
    vii. Endanger public safety and national cohesion
    viii. Plagiarize someone else's work
3. The License is valid for the proposed research, location and specified period.
4. Neither the license nor any rights thereunder are transferable.
5. The Commission reserves the right to cancel the research at any time during the research period if in the opinion of the Commission the research is not implemented in conformity with the provisions of the Act or any other written law.
6. The Licensee shall inform the relevant County Director of Education, County Commissioner and County Governor before commencement of the research.
7. Excavation, filming, movement, and collection of specimens are subject to further necessary clearance from relevant Government Agencies.
8. The License does not give authority to transfer research materials.
9. The Commission may monitor and evaluate the licensed research project for the purpose of assessing and evaluating compliance with the conditions of the License.
10. The Licensee shall submit one hard copy, and upload a soft copy of their final report (thesis) onto a platform designated by the Commission within one year of completion of the research.
11. The Commission reserves the right to modify the conditions of the License including cancellation without prior notice.
12. Research, findings and information regarding research systems shall be stored or disseminated, utilized or applied in such a manner as may be prescribed by the Commission from time to time.
13. The Licensee shall disclose to the Commission, the relevant Institutional Scientific and Ethical Review Committee, and the relevant national agencies any inventions and discoveries that are of National strategic importance.

13. The Licensee shall disclose to the Commission, the relevant Institutional Scientific and Ethical Review Committee, and the relevant national agencies any inventions and discoveries that are of National strategic importance.
14. The Commission shall have powers to acquire from any person the right in, or to, any scientific innovation, invention or patent of strategic importance to the country.
15. Relevant Institutional Scientific and Ethical Review Committee shall monitor and evaluate the research periodically, and make a report of its findings to the Commission for necessary action.

National Commission for Science, Technology and Innovation(NACOSTI),
Off Waiyaki Way, Upper Kabete,
P. O. Box 30623 - 00100 Nairobi, KENYA
Telephone: 020 4007000, 0713788787, 0735404245
E-mail: dg@nacosti.go.ke
Website: www.nacosti.go.ke