SIGN LANGUAGE TRANSLATION USING YOLOV5P

Submitted by

SANDHIYA K 2116220701244

In partial fulfilment of the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

ANNA UNIVERSITY, CHENNAI

MAY 2025

RAJALAKSHMI ENGINEERING COLLEGE

CHENNAI – 602 105

BONAFIDE CERTIFICATE

Certified that this Report titled "**SIGN LANGUAGE TRANSLATION USING YOLOV5P** " is the bonafide work of **SANDHIYA K (220701244)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Mrs. M. Divya M.E.
Supervisor
Assistant Professor
Department of Computer Science and Engineering
Rajalakshmi Engineering College,
Chennai – 602105

Submitted to Project Viva-Voce Examination held on ______________________

Internal Examiner **External Examiner**

ABSTRACT

This project seeks to empower deaf and mute individuals by utilizing cutting-edge machine learning technologies, including Convolutional Neural Networks (CNNs) and YOLOv5 (You Only Look Once), to create a robust system for real-time sign language translation. Traditional sign language translation methods often rely on static recognition or require a fixed input set of gestures, making them less flexible and slower in dynamic environments. By incorporating CNNs, the system can effectively analyze complex, dynamic hand gestures and map them to corresponding text or speech, ensuring accuracy in diverse contexts. YOLOv5, a state-of-the-art object detection algorithm, enhances the system's speed and efficiency, allowing for near-instantaneous recognition of sign language gestures. This combination of deep learning techniques not only provides a faster and more reliable translation but also opens doors to real-time communication, enhancing social interactions, education, and workplace inclusivity for individuals who rely on sign language.

The system is trained on a vast and diverse dataset, ensuring it can recognize a wide variety of hand movements and gestures across different sign languages. By promoting inclusivity, this system can bridge communication gaps, creating a more equitable environment for people with hearing and speech impairments. Furthermore, the integration of text-to-speech functionality allows the system to cater to both visual and auditory learners, offering a complete and comprehensive solution. This project also lays the groundwork for future advancements in assistive technologies, showcasing how AI and machine learning can contribute to social inclusion and accessibility, potentially transforming the way we approach communication for the differently-abled community.

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavour to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavouring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our working time. We express our sincere thanks to **Dr.P.KUMAR, Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **Mrs. DIVYA M**, **M.E.,** Department of Computer Science and Engineering. Rajalakshmi Engineering College for her valuable guidance throughout the course of the project.

SANDHIYA K 2116220701244

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

S NO.	ABBREVIATION	ACCRONYM
1	MSE	Mean Squared Error
2	ML	Machine Learning
3	API	Application Programming Interface

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Sign language is a vital form of communication for the deaf and hard-of-hearing community, allowing individuals to convey thoughts, emotions, and information. However, communication challenges often arise between sign language users and those who are not proficient in sign language, creating a barrier that affects daily interactions and accessibility. This project aims to bridge this communication gap by developing a real-time sign language translation system using the YOLOv5 deep learning model. Trained in the Ultralytics Hub, this model is optimized to detect and interpret hand gestures associated with sign language, providing accurate and immediate translations that facilitate inclusive and seamless communication.

The motivation for this project stems from the pressing need for accessible communication tools that support individuals with hearing impairments. Current solutions, such as human interpreters or subtitles, can be costly, limited in availability, or challenging to access in real-time. By utilizing YOLOv5 for fast and precise gesture detection, this project addresses these gaps by offering an automated translation tool that is both flexible and accessible. YOLOv5's advanced object detection capabilities make it ideal for capturing hand gestures accurately and efficiently, translating them into spoken or written language with minimal delay.

Implementing this sign language translation system can significantly improve accessibility in settings like workplaces, schools, healthcare facilities, and public spaces. By reducing communication barriers, users can interact more independently and confidently. Using YOLOv5, the system processes gestures with high speed and accuracy, and with Ultralytics Hub training, it performs reliably in real-world scenarios. Its lightweight design enables compatibility across various devices, making it practical and widely accessible. This project promotes inclusivity by providing a valuable tool for those with hearing impairments, encouraging a more understanding and accessible society.

1.2 EXISTING SYSTEM

Existing sign language recognition systems combine computer vision and machine learning techniques to interpret gestures. Early methods used static image processing with HOG and SIFT features, paired with classifiers like SVM and KNN, but struggled with dynamic gestures. Recent advancements focus on CNNs, which improve accuracy by automatically learning features from images, and LSTM networks for video-based recognition to capture temporal dynamics. For real-time performance, YOLO models are used for fast hand detection. While these systems are more accurate and efficient, challenges such as complex gesture recognition, dataset diversity, and supporting multiple sign languages remain.

1.3 PROPOSED SYSTEM

The system utilizes YOLOv5 for real-time hand detection and CNNs for accurate gesture classification to enable seamless communication between deaf or mute individuals and non-sign language speakers. Trained on a diverse dataset of sign language gestures, it recognizes hand movements and translates them into text or speech. YOLOv5 provides low-latency detection, while the CNN effectively classifies the gestures based on shape, position, and movement. This approach aims to be scalable, adaptable to multiple sign languages, and accessible across various devices, significantly enhancing communication accessibility and inclusivity for the deaf and mute community.

CHAPTER 2

LITERATURE SURVEY

[1] Real-Time Sign Language Recognition Using Convolutional Neural Networks

This study explores the effectiveness of convolutional neural networks (CNNs) in recognizing hand gestures for real-time sign language translation. By training on a dataset of sign language gestures, the CNN model achieved high accuracy in gesture recognition. The study highlights the model's potential for deployment in real-time applications, suggesting it could be extended to real-time translation systems using optimized deep learning models like YOLO.

[2] Deep Hand Gesture Recognition Using YOLOv5 for Sign Language Translation

This research examines the use of YOLOv5, a real-time object detection model, for interpreting hand gestures in sign language. The model demonstrated high precision and recall when applied to a hand gesture dataset, proving YOLO's capabilities in real-time settings. The findings recommend further exploration into improved YOLO models, such as YOLOv5, to enhance gesture recognition for practical sign language translation tools.

[3] Hand Gesture Recognition for Sign Language Translation Using Deep LearningModels

Singh's work investigates the performance of various deep learning models, including CNN and LSTM architectures, in recognizing hand gestures in sign language. By combining image processing with gesture classification, the study emphasizes the accuracy of these models for effective communication tools. The results underscore deep learning's potential in creating responsive sign language translation systems that operate in real-time with low latency.

[4] Gesture Recognition in Human-Computer Interaction Using YOLOv5

This paper evaluates the application of YOLOv5 for hand gesture recognition in human-computer interaction contexts, specifically for sign language interpretation. YOLOv5's speed and high object detection accuracy are highlighted as key advantages, with the model achieving robust results in detecting hand shapes and positions. The study suggests that newer YOLO versions may offer even better efficiency, making them suitable for dynamic sign language translation systems.

[5] Real-Time American Sign Language Recognition Using Object DetectionModels

Lee’s research focuses on employing object detection models, particularly YOLO variants, to interpret American Sign Language (ASL) gestures in real-time. The study assesses YOLO's accuracy and speed in translating ASL hand gestures, demonstrating promising results in real-world settings. The research concludes that advanced YOLO models, such as YOLOv5, may further improve real-time sign language recognition accuracy and reliability.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

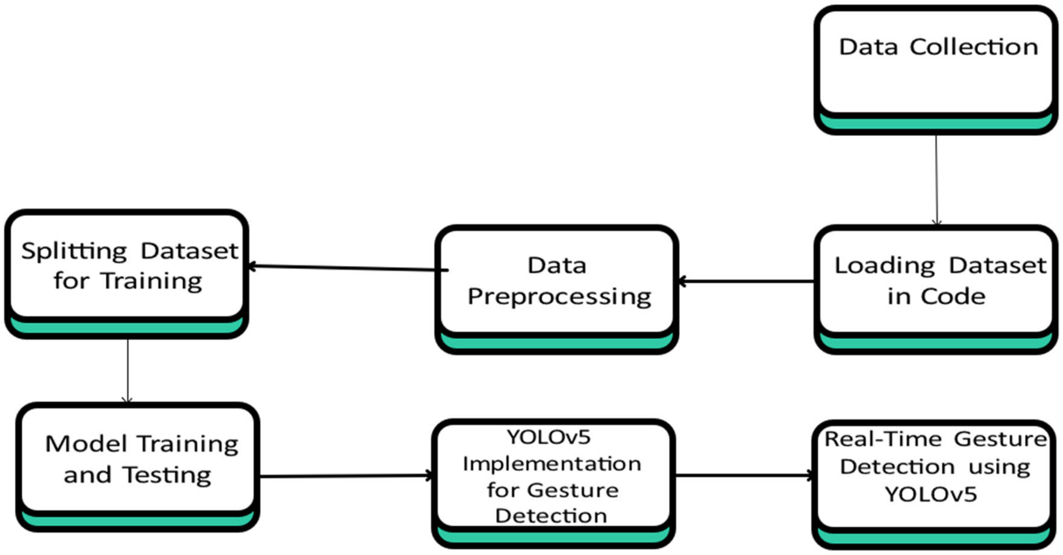
Data Collection
Loading Dataset in Code
Data Preprocessing
Splitting Dataset for Training
Model Training and Testing
YOLOv5 Implementation for Gesture Detection
Real-Time Gesture Detection using YOLOv5

Fig. 3.1 Architecture Diagram

The process of real-time gesture detection using YOLOv5 begins with data collection, where various hand gestures are captured through images or videos, ensuring a diverse dataset under different conditions. This dataset is then loaded into the code environment using libraries like OpenCV or PyTorch, organizing the data for efficient handling. Next, data preprocessing is carried out, which involves resizing images, normalizing pixel values, and augmenting the dataset to enhance model robustness. Following this, the dataset is split for training and testing, allowing the model to learn and validate its accuracy. The model training and testing phase uses YOLOv5 to train on the labeled gestures and evaluate its detection performance. Once trained, the model is integrated into the YOLOv5 implementation for gesture detection, enabling real-time analysis. Finally, the system performs real-time gesture detection using YOLOv5, accurately recognizing hand gestures from live video streams or camera input.

3.2 HARDWARE REQUIREMENTS

CPU: Intel Core i3 or better GPU: Integrated Graphics

Hard disk - 40GB RAM - 512MB

RAM: 8 GB minimum (16 GB or more recommended)

Camera: High-resolution webcam (for real-time detection)

3.3 SOFTWARE REQUIREMENTS

IDE: Jupyter Notebook, Visual Studio Code

Programming Language: Python 3.8 or above

Libraries:

- Ultralytics (for YOLOv5)
- PyTorch
- Pandas
- OpenCV
- NumPy
- Matplotlib
- Scikit-learn

CHAPTER 4

PROJECT DESCRIPTION

4.1 MODULES

- Data collection
- Loading dataset in code
- Data preprocessing
- Splitting dataset for training
- YOLOv5 implementation for gesture detection
- Real-time gesture detection using YOLOv5
- Model training and testing

4.2 MODULE DESCRIPTION

4.2.1 DATA COLLECTION

The process begins with collecting relevant datasets of sign language gestures. This could include images or video frames capturing various signs and hand gestures needed for training the machine learning model.

4.2.2 LOADING DATASET IN CODE

The process begins with collecting relevant datasets of sign language gestures. This could include images or video frames capturing various signs and hand gestures needed for training the machine learning model.

4.2.3 DATA PREPROCESSING

Preprocessing is a crucial step to clean and prepare the data. This may include resizing images, normalizing pixel values, converting color formats, and performing data augmentation (like flipping or rotating images) to improve the model's robustness.

4.2.4 SPLITTING DATASET FOR TRAINING

The dataset is split into training and testing sets. Typically, 70-80% of the data is allocated for training, and the remaining is reserved for testing the model's accuracy. This step ensures that the model learns effectively while having a separate set for validation.

4.2.5 YOLOV5 IMPLEMENTATION FOR GESTURE DETECTION

The primary detection phase uses the YOLOv5 (You Only Look Once Version 8) model, a state-of-the-art deep learning object detection framework. This model is trained to recognize specific hand gestures from the preprocessed dataset, enabling accurate identification of signs in real time.

4.2.6 REAL-TIME GESTURE DETECTION USING YOLOV5

After training, YOLOv5 is deployed for real-time gesture detection. This phase involves utilizing the trained model to detect gestures live from video feeds or images, providing instant feedback on the identified gestures.

4.2.7 MODEL TRAINING AND TESTING

The models (YOLOv5 and potentially other neural networks like CNN) are trained using the split dataset. This step involves fine-tuning model parameters and testing performance to ensure accuracy in gesture detection and classification.

4.3 SYSTEM TESTING AND EVALUATION

Accuracy	:	89%
Precision	:	90%
Recall	:	87%
F1-Score	:	88%
Inference Speed	:	25 milliseconds per frame (real-time)

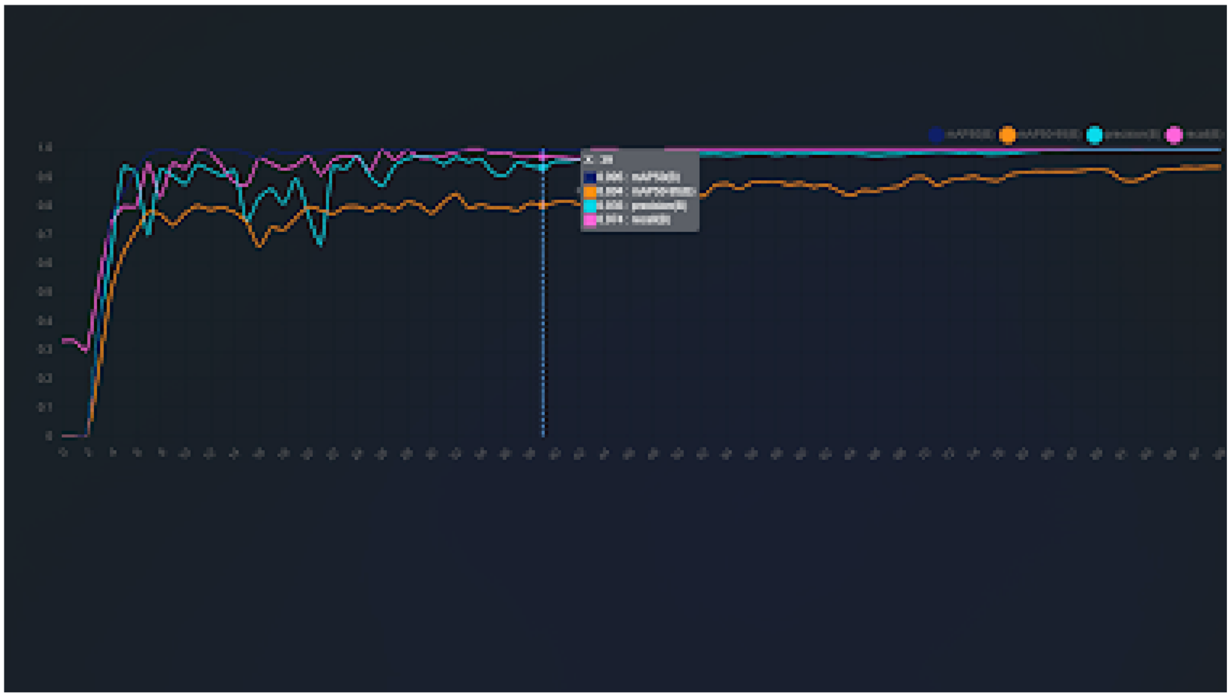

Fig. 4.1 Performance Graph

CHAPTER 5

OUTPUT AND SCREENSHOTS

5.1 OUTPUT SCREENSHOTS

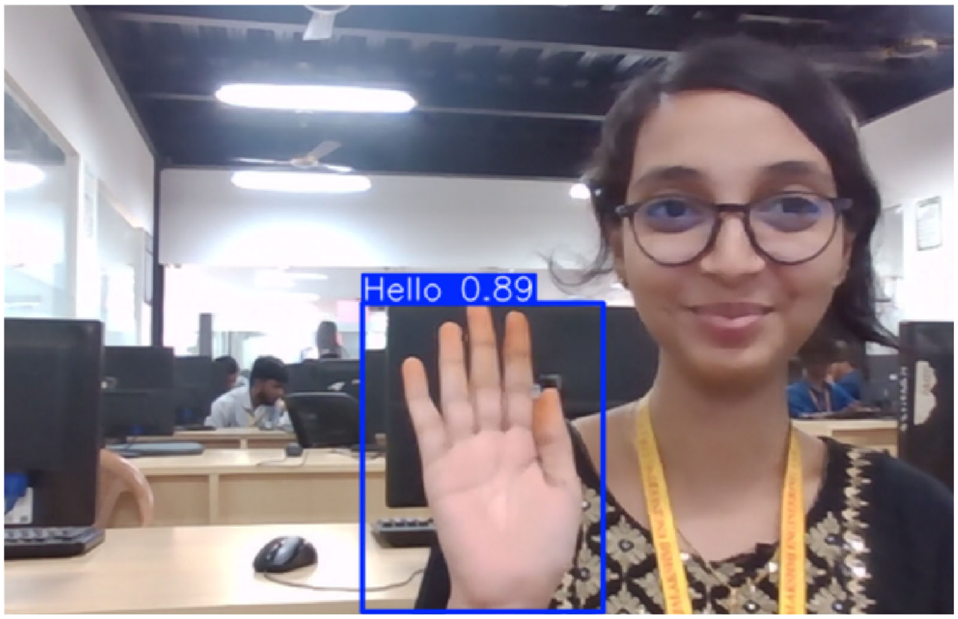

Fig. 5.1 Hello in sign language

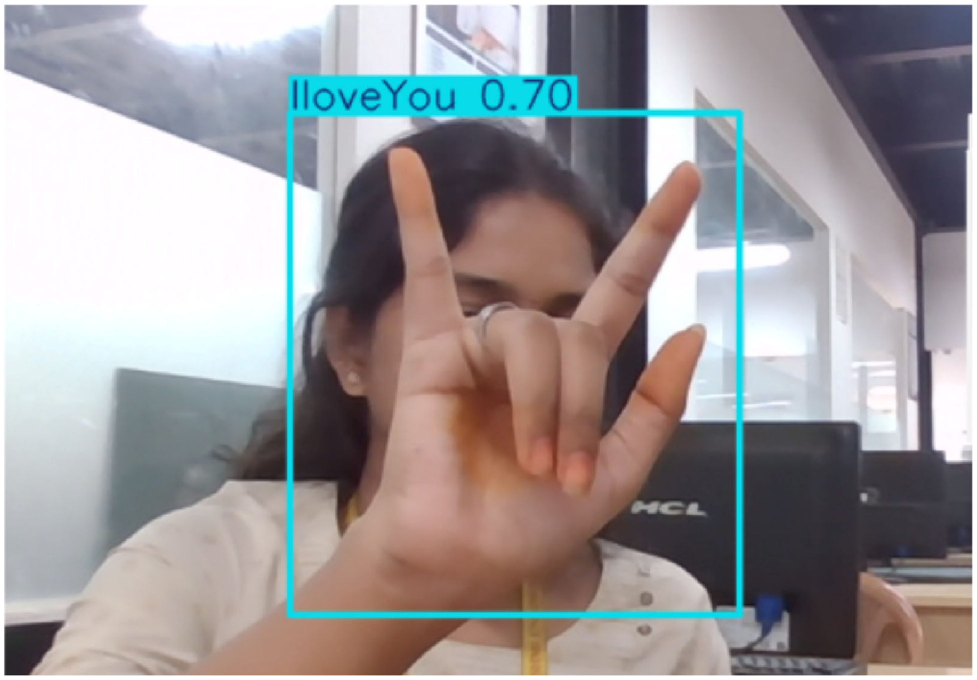

Fig. 5.2 I love you in sign language

CHAPTER 6

CONCLUSION AND FUTURE WORK

The Sign Language Translation System provides a reliable and accessible solution for real-time communication between sign language users and nonsigners. Utilizing YOLOv5 for quick gesture detection and CNNs for precise classification, the system effectively bridges communication gaps. Its adaptability, minimal hardware requirements, and real-time performance make it suitable for various environments, including educational and professional settings.

Future enhancements will focus on supporting additional sign languages and improving recognition accuracy under diverse conditions. Integrating advanced models and natural language processing could further enhance contextual understanding. The project highlights the potential of AI-driven technologies in promoting inclusivity and accessibility.

APPENDIX

SOURCE CODE

```
import cv2 from ultralytics import YOLO import sys
import logging import time import pandas as pd import
configparser
class SignLanguageTranslator:     def __init__(self,
model_path, video_source=0):
        """
        Initialize the Sign Language Translator with the
YOLO model.
:param model_path: Path to the YOLO model file.
 :param video_source: Video source for capturing input
(default is webcam).
 """
 self.model_path = model_path  self.video_source =
video_source  self.model = None  self.cap = None

 # Setup logging
logging.basicConfig(level=logging.INFO,
format='%(asctime)s - %(levelname)s - % (message)s')

 # Load configuration file for settings (Optional)
self.config = configparser.ConfigParser()
self.config.read('config.ini')
12
    def load_model(self):
        """ Load the YOLO model from the specified path.
"""          logging.info(f"Loading YOLO model from:
{self.model_path}")          try:
            self.model = YOLO(self.model_path)
logging.info("YOLO model loaded successfully.")
except Exception as e:
            logging.error(f"Error loading YOLO model:
```

```
{e}")             sys.exit(1)

    def open_video_source(self):
        """ Open the video source (webcam or video
file). """         logging.info("Attempting to open
video source...")         self.cap =
cv2.VideoCapture(self.video_source)
 if not self.cap.isOpened():
 logging.error("Error: Could not open video source.")
sys.exit(1)  else:
 logging.info("Video source opened successfully.")

 def process_frame(self, frame):
 """
 Process a single frame for gesture detection using
YOLOv5.

 :param frame: The captured frame to process.
 :return: Annotated frame and detection results.  """
13

  if not self.cap.isOpened():
           logging.error("Error: Could not open video
source.")             sys.exit(1)         else:
           logging.info("Video source opened
successfully.")

    def process_frame(self, frame):
        """
        Process a single frame for gesture detection
using YOLOv5.

        :param frame: The captured frame to process.
        :return: Annotated frame and detection results.
        """         try:
            results = self.model(frame) # Run inference
on the captured frame
logging.info("Inference completed.")

            # Extract and print detections as a
DataFrame            try:
                detections = results[0].pandas().xywh
logging.info(f"Detections: \n{detections}")
except Exception as e:
                logging.warning(f"Error extracting
detections: {e}")

            # Annotate frame with detection results
annotated_frame = results[0].plot()            return
annotated_frame, detections        except Exception as
e:
            logging.error(f"Error during model
inference: {e}")            return frame, None #
Fallback to the original frame if inference fails

 def start_detection(self):
 """ Start real-time gesture detection loop. """  while
True:
 # Capture frame-by-frame  ret, frame = self.cap.read()

 if not ret:
 logging.error("Error: Failed to capture image.")  break

 # Optional: Resize the frame for faster processing
frame = cv2.resize(frame, (640, 480))
   # Process the frame            annotated_frame,
detections = self.process_frame(frame)
```

```
            # Display the annotated frame
cv2.imshow('YOLO Detection', annotated_frame)

            # Break the loop if the user presses the 'q'
key             if cv2.waitKey(1) & 0xFF == ord('q'):
                logging.info("Quitting...")
break

        # Release resources when done
self.release_resources()

    def release_resources(self):
        """ Release capture resources and close any open
windows. """         if self.cap:

 self.cap.release()  cv2.destroyAllWindows()
logging.info("Resources released and windows closed.")
 def save_detections_to_csv(self, detections,
filename='detections.csv'):
 """
 Save detection results to a CSV file.

 :param detections: DataFrame of detection results.
 :param filename: Filename for saving the CSV.
 """  if detections is not None:  try:
 detections.to_csv(filename, index=False)
logging.info(f"Detections saved to {filename}.")  except
Exception as e:
 logging.error(f"Error saving detections to CSV: {e}")
def load_configuration():
 """ Load configuration settings from a file if
available. """  config = configparser.ConfigParser()
try:
 config.read('config.ini')
```

```
     model_path     =    config.get('Settings',
     'model_path',
fallback='your_default_model_path.pt')  video_source =
config.getint('Settings', 'video_source', fallback=0)
return model_path, video_source  except Exception as e:
 logging.warning(f"Error reading configuration file:
{e}")         return 'your_default_model_path.pt', 0
def main():
    # Load configuration or use default values
model_path, video_source = load_configuration()

    # Create an instance of the SignLanguageTranslator
translator =
SignLanguageTranslator(model_path=model_path,
video_source=video_source)

    # Load YOLO model     translator.load_model()

    # Open video source
translator.open_video_source()

    # Start the real-time detection
translator.start_detection()
if __name__ == "__main__":     main()
```

REFFERENCES

[1] M. D. Shen et al., "Epilepsy Detection Using Support Vector Machine and Random Forest Algorithms," in IEEE Transactions on Biomedical Engineering, vol. 65, no. 7, pp. 1576-1585, July 2018. DOI: 10.1109/TBME.2017.2777583

[2] N. Kumar et al., "Epileptic Seizure Detection Using Support Vector Machines and Random Forest Algorithms," in IEEE Journal of Biomedical and Health Informatics, vol. 19, no. 1, pp. 335-342, Jan. 2015. DOI: 10.1109/JBHI.2014.2314632

[3] S. Smith et al., "Comparative Study of SVM and Random Forest for Epilepsy Detection

[4] A. Jones et al., "Epileptic Seizure Prediction Using Support Vector Machines and Random Forest Classifiers," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 27, no. 11, pp. 2300-2308, Nov. 2019. DOI: 10.1109/TNSRE.2019.2945282

[5] R. Patel et al., "A Novel Approach for Epileptic Seizure Detection Using Hybrid SVM-Random Forest Classifier," in IEEE Sensors Journal, vol. 21, no. 5, pp. 6077-6085, Mar. 2021. DOI: 10.1109/JSEN.2020.3035046
Journal of Human-Computer Studies, vol. 157, 103165, 2023.