MAJOR PROJECT REPORT

ON

SIGN LANGUAGE TRANSCRIPTION

AT

G.D GOENKA UNIVERSITY

Submitted partial in fulfillment of the requirements for the award of degree of

Bachelor of Technology

In

Computer Science Engineering & AI ML

SUBMITTED BY:

NAME: HARSHITA VISHNOI, ARYAN SINGH, MANAV SINGHAL, GARIMA, ARADHYA, ZAID KHAN

ENROLLMENT NO. : 200020223031, 200020223030, 200020223041, 200020223029, 200020203014, 200020203053

SUPERVISOR:

NAME: MS APEKSHA MITTAL

DESIGNATION: ASSISTANT PROFESSOR

DEPARTMENT: B. TECH CSE/AI & ML

Department of Computer Science

School of Engineering and Sciences, G D Goenka University

April 2024

Declaration

We, solemnly declare that the project report titled Sign language transcription submitted in partial fulfillment of the requirements for the bachelor's in technology at **GD Goenka University** is the result of our own work under the supervision of **Ms.Apeksha Mittal**, we researched all the insights and perspectives and did all the coding part. All sources of information used in the report have been duly acknowledged.

We further declare that:

1. The work contained in the report is original and has been done by us.
2. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.
3. We have followed the guidelines provided by the university in writing the report.

Date: 26 April 2024

Students Name

HARSHITA VISHNOI, ARYAN SINGH, MANAV SINGHAL, GARIMA, ARADHYA, ZAID KHAN
200020223031, 200020223030, 200020223041, 200020223029, 200020203014, 200020203053
B. Tech CSE & AI ML

CERTIFICATE

This is to certify that **Harshita Vishnoi, Aryan Singh, Manav Singhal, Garima, Aradhya and Zaid khan** student of **B.Tech CSE / Ai & ML** 8th Semester of **G.D Goenka University**, Gurugram has successfully completed In-house Project work from **January 2024 to April 2024.** During this project, they worked on the project titled Sign language transcription under the guidance of **Ms. Apeksha Mittal.** Their overall performance during the project duration was appreciable.

Signature

Name of the Project Supervisor: Ms. Apeksha Mittal

Designation: Assistant Professor

Department: Computer Science and Engineering

University Name: G.D Goenka University

Acknowledgment

We would like to express our sincere gratitude and appreciation to everyone who contributed to the successful completion of the project titled **Sign language transcription** using python and its libraries.

First and foremost, we extend my heartfelt thanks to my project supervisor, **Ms. Apeksha Mittal,** for their invaluable guidance, support, and constructive feedback throughout the development of this project. Their expertise and encouragement significantly enriched my understanding and approach to tackling the challenges associated with **Sign language transcription project.**

We are grateful to **GD Goenka University** for providing the necessary resources and environment for carrying out this project. The access to the latest technologies and libraries greatly facilitated the implementation and testing phases.

Date: 26 April 2024

Ms.Apeksha Mittal

(Assistant Professor

Department of CSE)

ABSTRACT

Title: Sign Language transcription

Project Period: January 2024 to April 2024

Sign language stands as a diverse and expressive means of communication primarily utilized by the Deaf and Hard of Hearing communities.

Sign language transcription, a field dedicated to studying and practicing the conversion of visual-gestural elements into digital formats, plays a pivotal role in facilitating communication, education, and accessibility for the Deaf community.

This abstract delves into the significance and challenges associated with sign language transcription, underscoring its pivotal role in dismantling communication barriers. The process of digitally transcribing sign languages entails capturing and representing the nuanced movements, facial expressions, and spatial relationships inherent in signing.

This project presents a gesture recognition system leveraging computer vision and deep learning techniques to interpret sign language gestures in real-time. The system utilizes the MediaPipe library for robust landmark detection, capturing intricate hand and body movements. A dataset comprising five American Sign Language (ASL) signs was collected and preprocessed for model training. Each sign was represented by sequences of 30 frames, containing the 3D coordinates of pose, face, and hand landmarks.

A Long Short-Term Memory (LSTM) neural network architecture was employed for sequence modeling, capable of capturing temporal dependencies within the data. The model was trained on the dataset, achieving a categorical accuracy above 92% on the test set. Additionally, a custom callback was implemented to halt training upon reaching the specified target accuracy, enhancing training efficiency.

Real-time gesture recognition was performed using a webcam feed and the trained model. As users perform gestures in front of the camera, the system detects landmarks and predicts the corresponding sign. A dynamic visualization overlays the recognized signs and their probabilities onto the video feed, providing immediate feedback to the user.

The developed system offers a user-friendly interface for real-time ASL gesture interpretation, aiding communication for individuals with hearing impairments. Future work could involve expanding the dataset to encompass a broader range of ASL signs, refining the model architecture for improved accuracy, and exploring deployment on mobile platforms for enhanced accessibility. Overall, this project demonstrates the potential of combining computer vision and deep learning for real-world applications in assistive technology and communication aid systems.

TABLE OF CONTENTS

INTRODUCTION......6

PROJECT REQUIREMENTS......7

Software Requirements:......7

Hardware Requirements:......7

LITERATURE REVIEW......8

1. Eweghe and Dambre (LREC 2020)......8

2. Jiang et al. (IEEE/CVF Conference 2021):......8

3. Liao et al. (IEEE Access 2019):......8

4. Adaloglou and Chatzis (IEEE Trans. Multimedia 2022):......9

5. Joint End-to-end Sign Language Recognition and Translation Necati Cihan Camgoz¨ , Oscar Kollerq, Simon Hadfield and Richard Bowden :......9

PROPOSED METHODOLOGY & IMPLEMENTATION......12

1. Data Collection:......12

2. Data Preprocessing:......13

3. Model Architecture:......15

4. Model Training:......16

5. Real-time Hand Gesture Detection:......18

6. Evaluation:......20

7. Optimization and Fine-tuning:......21

8. Deployment:......21

CONCLUSIONS......22

FUTURE ENHANCEMENTS......24

REFERENCES......25

INTRODUCTION

In a world where communication is predominantly verbal, individuals who rely on sign language face unique challenges in accessing information and engaging fully with society. Sign language, a rich and expressive mode of communication, is the primary means of interaction for millions worldwide, particularly among the Deaf and hard of hearing community. However, despite its importance, there exists a significant gap in accessibility when it comes to transcribing sign language into written or spoken forms.

Sign Language Transcription Project" seeks to address this gap by developing a comprehensive system for accurately transcribing sign language into written text. This initiative aims to empower individuals within the Deaf and hard of hearing community by providing them with greater access to education, employment opportunities, and participation in various aspects of society.

Through the utilization of advanced technologies such as computer vision, machine learning, and natural language processing, this project endeavors to create a platform capable of capturing and translating sign language gestures into written or spoken language in real-time. By harnessing the power of technology, we aspire to break down communication barriers and foster inclusivity for all individuals, regardless of their preferred mode of communication.

Moreover, the "Sign language transcription" project recognizes the importance of collaboration and community engagement in achieving its goals. We are committed to working closely with members of the Deaf community, sign language experts, researchers, and technologists to ensure that our transcription system is accurate, culturally sensitive, and aligned with the diverse linguistic nuances of sign languages around the world.

Ultimately, the Sign Language Transcription Project aims to revolutionize the way sign language is represented and understood, paving the way for a more inclusive and accessible future where communication knows no boundaries.

PROJECT REQUIREMENTS

Software Requirements:

- Jupyter notebook
- Python
- Mediapipe
- Opencv python
- Tensorboard
- Scikit-learn
- LSTM

Hardware Requirements:

- Camera
- Processor with minimum configuration (Intel 5 8th Gen)
- GPU with minimum configuration (4GB)
- Storage (15GB or more)
- RAM (8GB or more)
- Input Devices (Depth Sensing cameras)
- Microphone (Optional)
- Sensors (Optional)

LITERATURE REVIEW

Sign language recognition has emerged as a critical domain within the field of computer vision and artificial intelligence, addressing the unique challenge of understanding and interpreting non-verbal communication through gestures. In recent years, researchers have explored diverse methodologies and architectures to enhance the accuracy and efficiency of sign language recognition systems. This literature review aims to provide a comprehensive overview and analysis of key contributions in this evolving field. By delving into seminal works, recent advancements, and critical evaluations, this review seeks to identify trends, gaps, and potential avenues for future research. Understanding the landscape of sign language recognition is essential for the development of robust and inclusive communication technologies.

1. ***Eweghe and Dambre (LREC 2020)***

Eweghe and Dambre (LREC 2020) pioneered the use of Transformer Networks in sign language recognition. Their work focuses on enhancing efficiency through parallel processing, demonstrating promising results. However, a more extensive evaluation and comparison with existing models would strengthen the paper's contributions.

2. ***Jiang et al. (IEEE/CVF Conference 2021):***

Jiang et al. (IEEE/CVF Conference 2021) introduce a novel approach to sign language recognition by incorporating skeleton awareness and multi-modal strategies. The presentation at a prestigious conference reflects the significance of their work. Further exploration of potential limitations and challenges would enhance the paper's comprehensiveness.

3. ***Liao et al. (IEEE Access 2019):***

Liao et al. (IEEE Access 2019) contribute to the dynamic sign language recognition domain by introducing BLSTM-3D Residual Networks. Their publication in IEEE Access underscores the significance of their findings. A more in-depth discussion on the scalability and generalization aspects would enhance the paper's impact.

4. ***<u>Adaloglou and Chatzis (IEEE Trans. Multimedia 2022):</u>***

Adaloglou and Chatzis (IEEE Trans. Multimedia 2022) conduct a comprehensive study on deep learning methods for sign language recognition, providing valuable insights into existing approaches. However, a more focused presentation of their proposed method and rigorous experimental evaluations could further strengthen their contributions.

5. ***<u>Joint End-to-end Sign Language Recognition and Translation Necati Cihan Camgoz¨, Oscar Kollerq, Simon Hadfield and Richard Bowden :</u>***

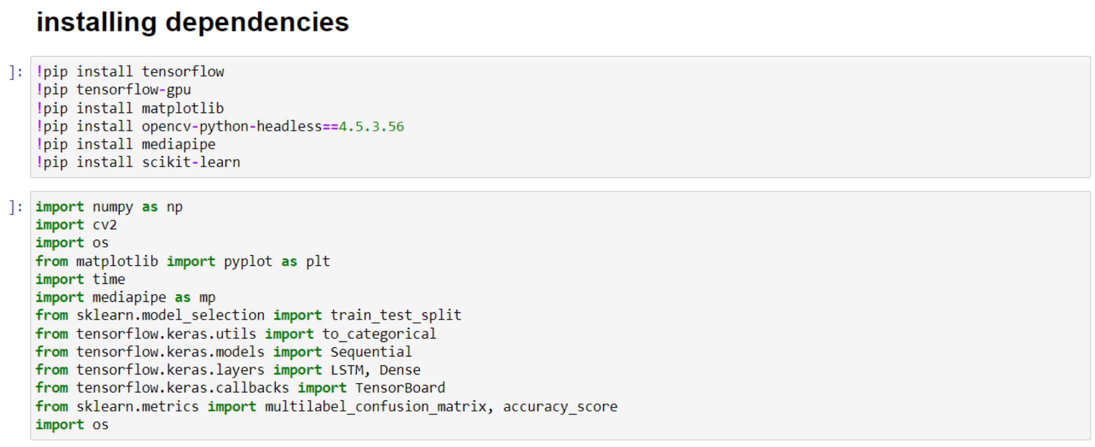
The paper explores Sign Language Transformers, leveraging transformer architectures for joint sign language recognition and translation. It enhances recognition via gloss-level supervision, achieving state-of-the-art results on the PHOENIX14 T dataset. EfficientNet-based spatial embeddings and multimodal transformers contribute to improved performance, advancing sign language understanding**.**

Name of paper	pros	cons
Eweghe and Dambre (LREC 2020)	Transformer networks offer parallel processing capabilities, which can enhance efficiency. The paper contributes to sign language recognition with a focus on transformer architecture	May lack comprehensive evaluation metrics or comparisons with other state-of-the-art models.
Jiang et al. (IEEE/CVF Conference 2021):	Integrates skeleton awareness and multi-modal approaches for sign language recognition. Presented at a reputable conference (IEEE/CVF).	Specific challenges and limitations of the proposed skeleton-aware multimodal recognition approach may not be thoroughly discussed.
Liao et al. (IEEE Access 2019):	Utilizes BLSTM-3D Residual Networks for dynamic sign language recognition. Published in IEEE Access.	The scalability and generalizability of the proposed BLSTM-3D Residual Networks may not be extensively discussed.
Adaloglou and Chatzis (IEEE Trans. Multimedia 2022):	Offers a comprehensive study on deep learning-based methods for sign language recognition.	The paper might lack a strong emphasis on the proposed novel approach and potential experimental evaluations.

Joint End-to-end Sign Language Recognition and Translation Necati Cihan Camgoz¨ , Oscar Kollerq, Simon Hadfieldand Richard Bowden CVSSP, University of Surrey, Guildford, UK, qMicrosoft, Munich, Germany	Introduces Sign Language Transformers for joint recognition and translation. Utilizes gloss-level supervision to enhance recognition. Achieves state-of-the-art results on the PHOENIX14 T dataset.	Reliance on pretraining for spatial embeddings may limit applicability. Limited discussion on real-world deployment challenges. May require substantial computational resources for training.

PROPOSED METHODOLOGY & IMPLEMENTATION

Sign Language Detection using LSTM and MediaPipe



installing dependencies

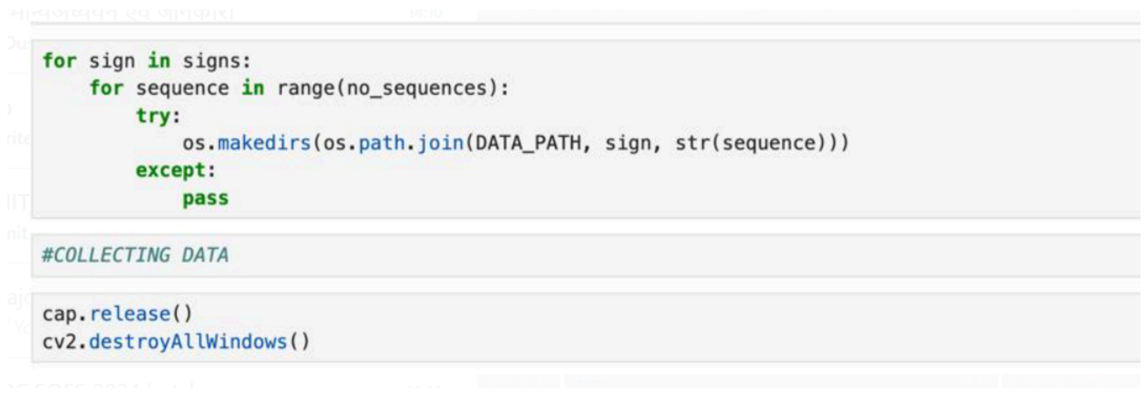
```
]: !pip install tensorflow
   !pip tensorflow-gpu
   !pip install matplotlib
   !pip install opencv-python-headless==4.5.3.56
   !pip install mediapipe
   !pip install scikit-learn
```

```
]: import numpy as np
   import cv2
   import os
   from matplotlib import pyplot as plt
   import time
   import mediapipe as mp
   from sklearn.model_selection import train_test_split
   from tensorflow.keras.utils import to_categorical
   from tensorflow.keras.models import Sequential
   from tensorflow.keras.layers import LSTM, Dense
   from tensorflow.keras.callbacks import TensorBoard
   from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
   import os
```

Figure 1.1

1. Data Collection:

- Gathered a dataset of sign language gestures, including various words and phrases commonly used in sign language communication.
- The dataset collected includes the following gestures - day, father, friend, hello, help, house, iloveyou, no,ok, peace, phone, please, thankyou, yes.
- Used a web camera to capture sequences of hand gestures representing each sign language word or phrase.
- Each gesture consists of 30 sequences of videos and each video consists of 30 frames in each.
- Ensured diversity in hand gestures, lighting conditions, backgrounds, and hand orientations to improve model robustness.

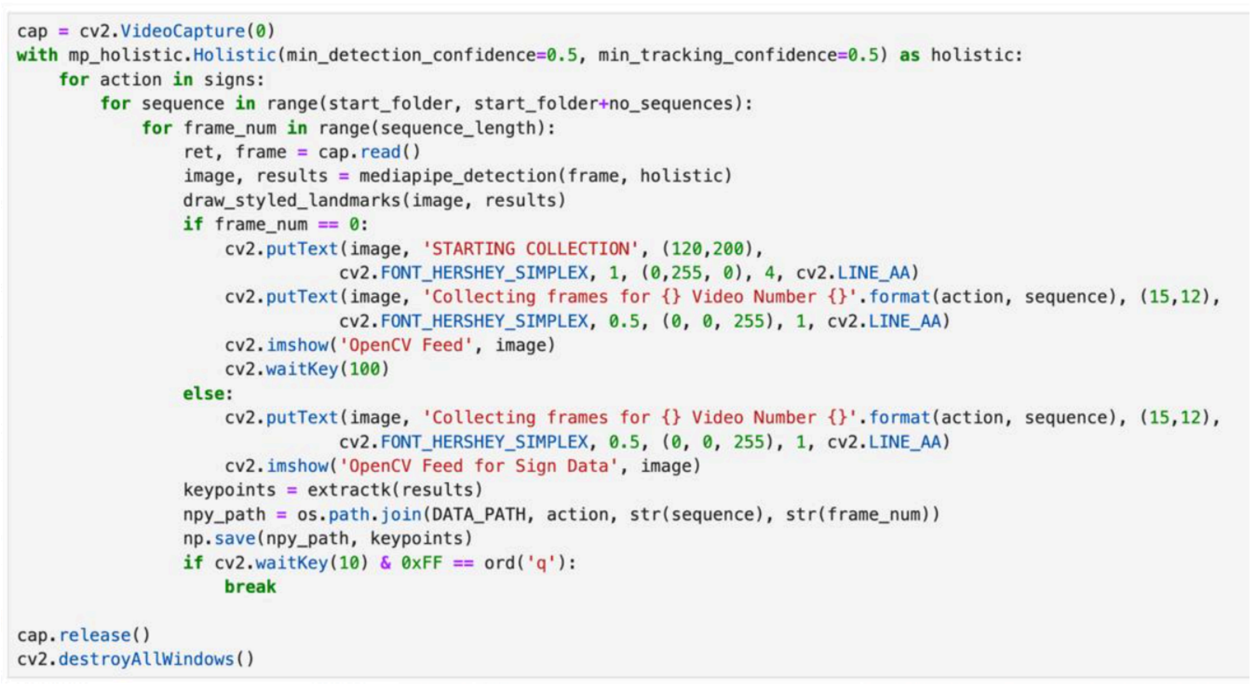


```
for sign in signs:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, sign, str(sequence)))
        except:
            pass
```

```
#COLLECTING DATA
```

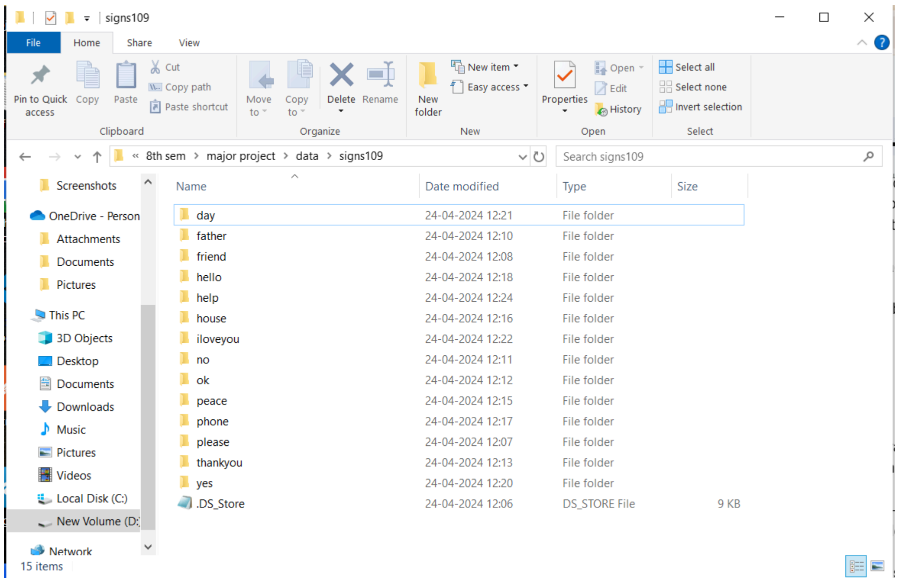
```
cap.release()
cv2.destroyAllWindows()
```

Figure 1.2



```python
cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    for action in signs:
        for sequence in range(start_folder, start_folder+no_sequences):
            for frame_num in range(sequence_length):
                ret, frame = cap.read()
                image, results = mediapipe_detection(frame, holistic)
                draw_styled_landmarks(image, results)
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(100)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed for Sign Data', image)
                keypoints = extractk(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

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

Figure 1.3

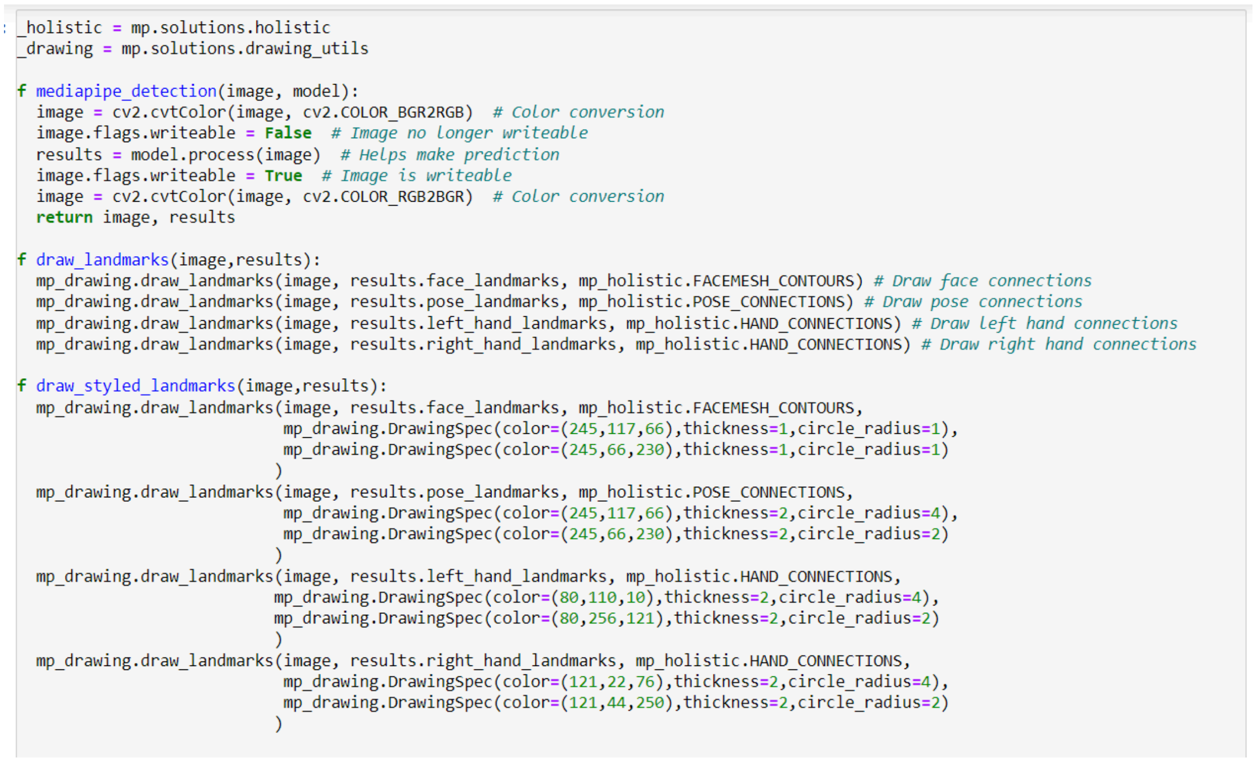


Name	Date modified	Type	Size
day	24-04-2024 12:21	File folder	
father	24-04-2024 12:10	File folder	
friend	24-04-2024 12:08	File folder	
hello	24-04-2024 12:18	File folder	
help	24-04-2024 12:24	File folder	
house	24-04-2024 12:16	File folder	
iloveyou	24-04-2024 12:22	File folder	
no	24-04-2024 12:11	File folder	
ok	24-04-2024 12:12	File folder	
peace	24-04-2024 12:15	File folder	
phone	24-04-2024 12:17	File folder	
please	24-04-2024 12:07	File folder	
thankyou	24-04-2024 12:13	File folder	
yes	24-04-2024 12:20	File folder	
.DS_Store	24-04-2024 12:06	DS_STORE File	9 KB

Figure 1.4

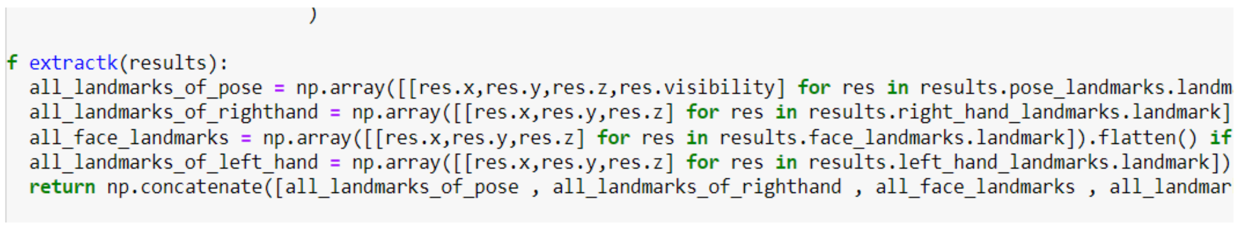
2. Data Preprocessing:

- Converted the captured video sequences into individual frames.
- Utilize the MediaPipe library to detect and extract both hands, face and pose landmarks from each frame.
- Normalize the extracted landmarks to a consistent scale(-1,1) and format.
- Organize the data into sequences of fixed length, suitable for input to the LSTM model.
- Label the sequences according to the corresponding sign language word or phrase.



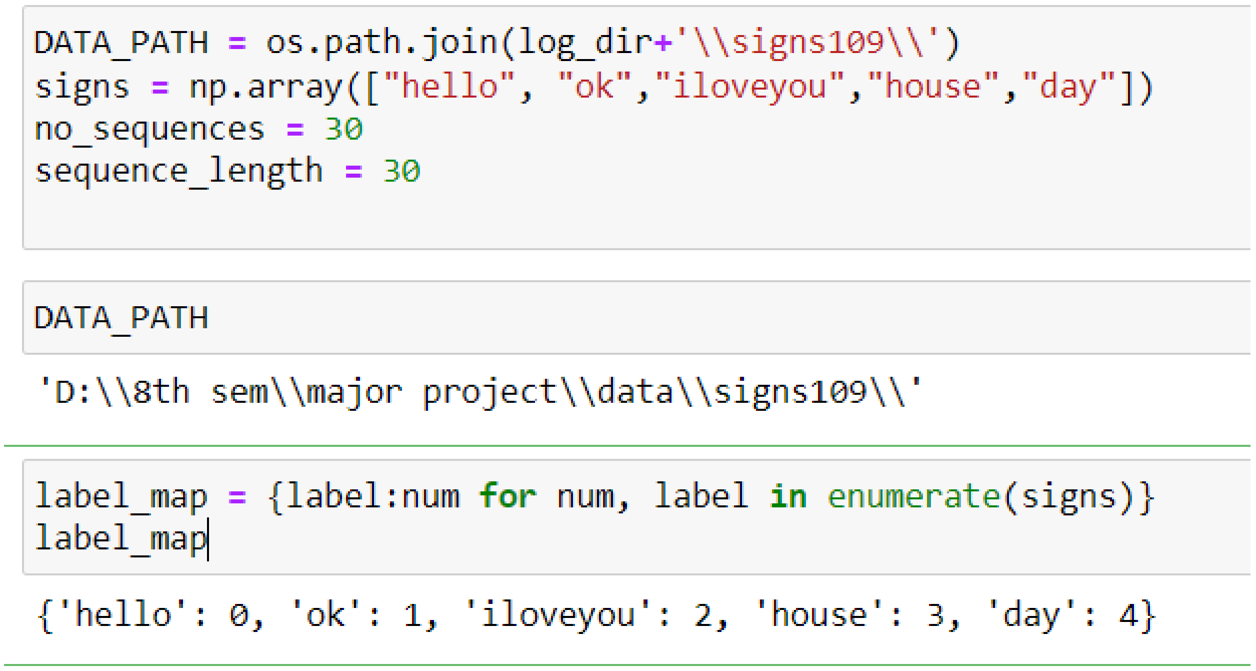
```
: _holistic = mp.solutions.holistic
 _drawing = mp.solutions.drawing_utils

f mediapipe_detection(image, model):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Color conversion
  image.flags.writeable = False  # Image no longer writeable
  results = model.process(image)  # Helps make prediction
  image.flags.writeable = True  # Image is writeable
  image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)  # Color conversion
  return image, results

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

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

Figure 1.5



```
                         )

f extractk(results):
  all_landmarks_of_pose = np.array([[res.x,res.y,res.z,res.visibility] for res in results.pose_landmarks.landm
  all_landmarks_of_righthand = np.array([[res.x,res.y,res.z] for res in results.right_hand_landmarks.landmark]
  all_face_landmarks = np.array([[res.x,res.y,res.z] for res in results.face_landmarks.landmark]).flatten() if
  all_landmarks_of_left_hand = np.array([[res.x,res.y,res.z] for res in results.left_hand_landmarks.landmark])
  return np.concatenate([all_landmarks_of_pose , all_landmarks_of_righthand , all_face_landmarks , all_landmar
```

Figure 1.6



```
DATA_PATH = os.path.join(log_dir+'\\signs109\\')
signs = np.array(["hello", "ok","iloveyou","house","day"])
no_sequences = 30
sequence_length = 30
```

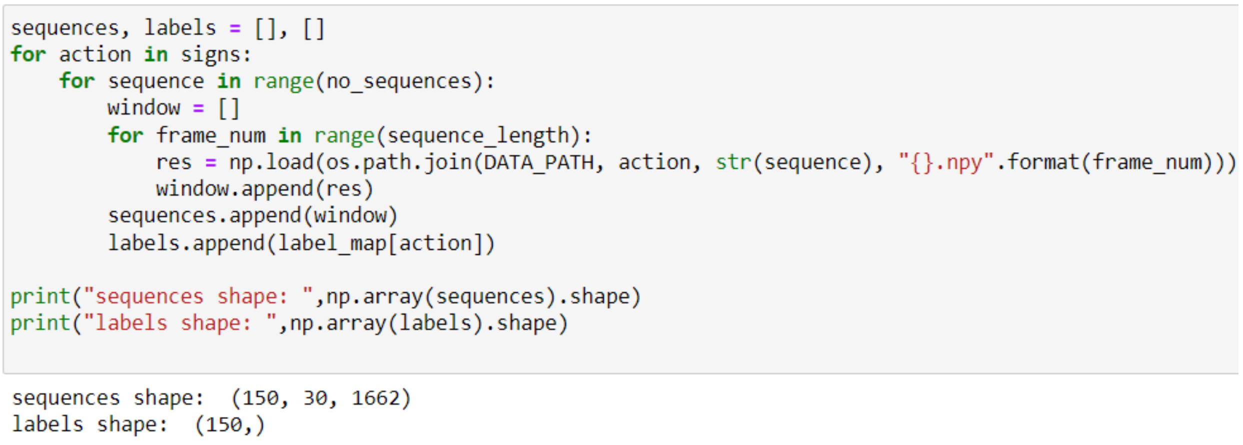
```
DATA_PATH
```

```
'D:\\8th sem\\major project\\data\\signs109\\'
```

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

```
{'hello': 0, 'ok': 1, 'iloveyou': 2, 'house': 3, 'day': 4}
```

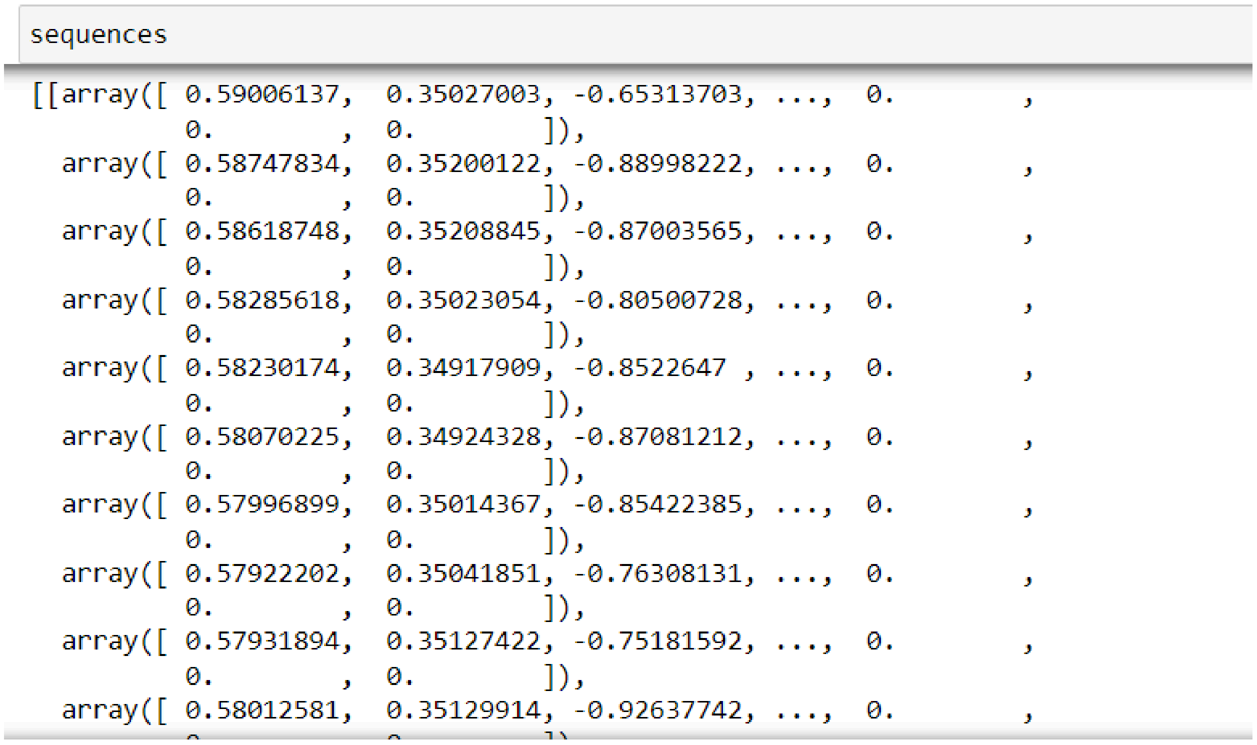
Figure 1.7



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

print("sequences shape: ",np.array(sequences).shape)
print("labels shape: ",np.array(labels).shape)
```

```
sequences shape:  (150, 30, 1662)
labels shape:  (150,)
```

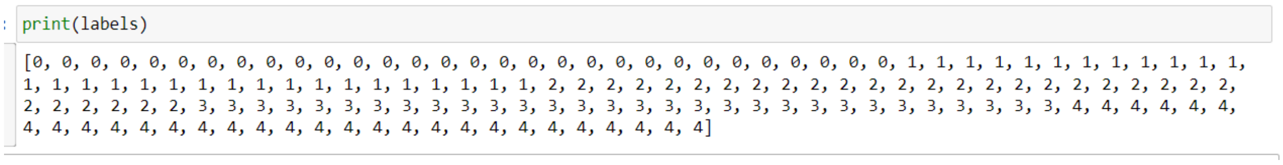
Figure 1.8



```
sequences
```

```
[[array([ 0.59006137,  0.35027003, -0.65313703, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.58747834,  0.35200122, -0.88998222, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.58618748,  0.35208845, -0.87003565, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.58285618,  0.35023054, -0.80500728, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.58230174,  0.34917909, -0.8522647 , ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.58070225,  0.34924328, -0.87081212, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.57996899,  0.35014367, -0.85422385, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.57922202,  0.35041851, -0.76308131, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.57931894,  0.35127422, -0.75181592, ...,  0.        ,
          0.        ,  0.        ]),
  array([ 0.58012581,  0.35129914, -0.92637742, ...,  0.        ,
```

Figure 1.9



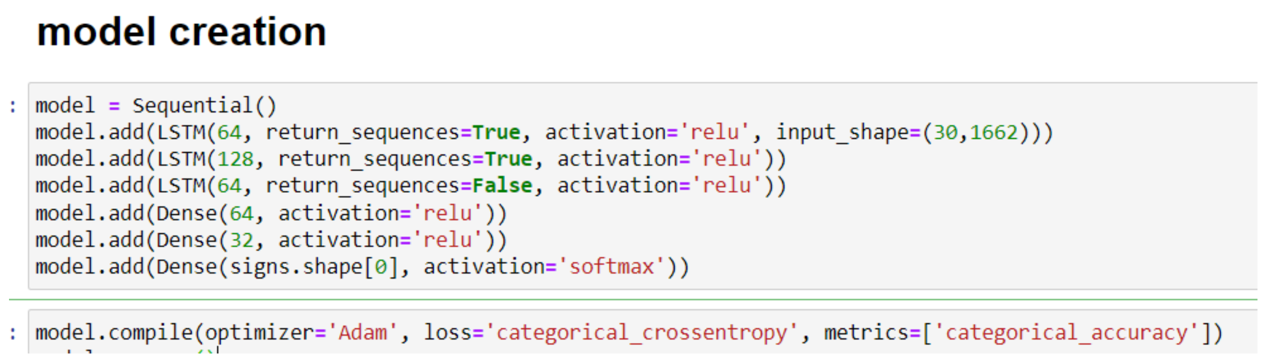
```
print(labels)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

Figure 1.10

3. Model Architecture:

- Construct an LSTM-based deep learning model for sequential data processing.
- Design the architecture to accept sequences of hands, face and pose landmarks as input and output the predicted sign language word or phrase.
- Experiment with different LSTM layer configurations, including the number of units, activation functions, and layer stacking.
- Add dense layers with appropriate activation functions for classification.
- Compile the model with suitable loss function (e.g., categorical cross-entropy) and optimizer (e.g., Adam).

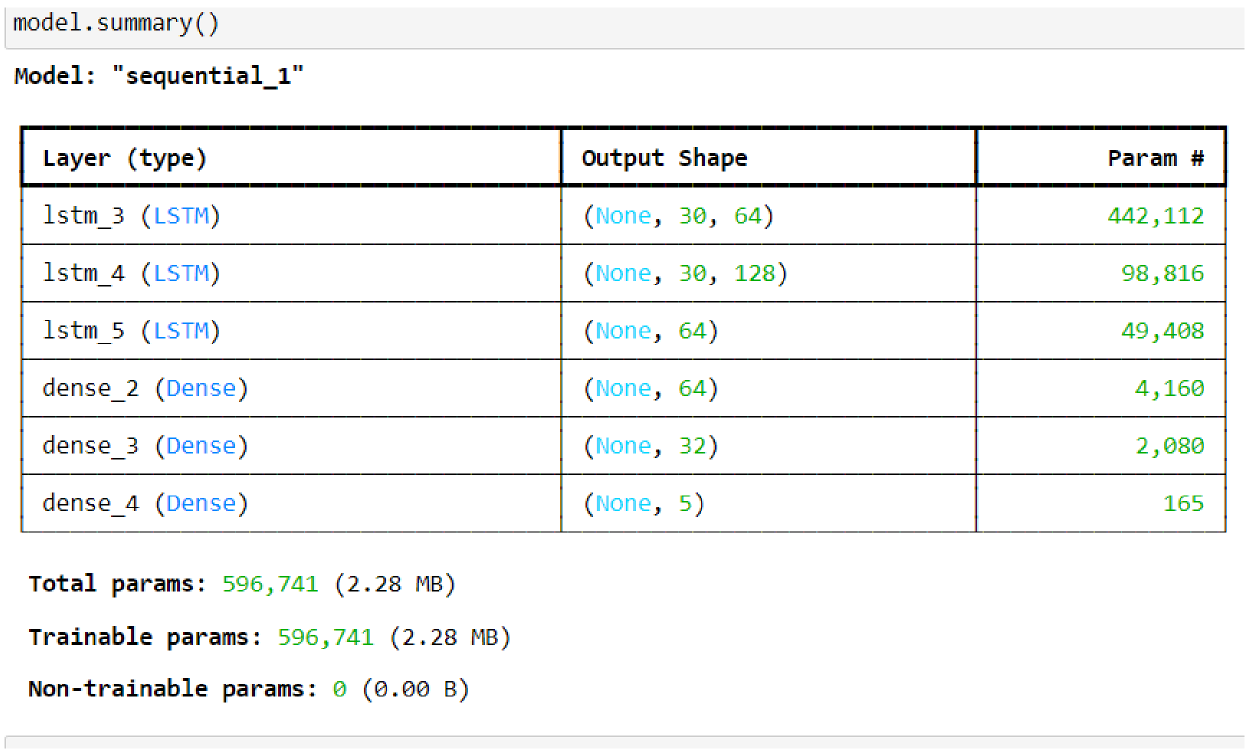


model creation

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

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

Figure 1.11



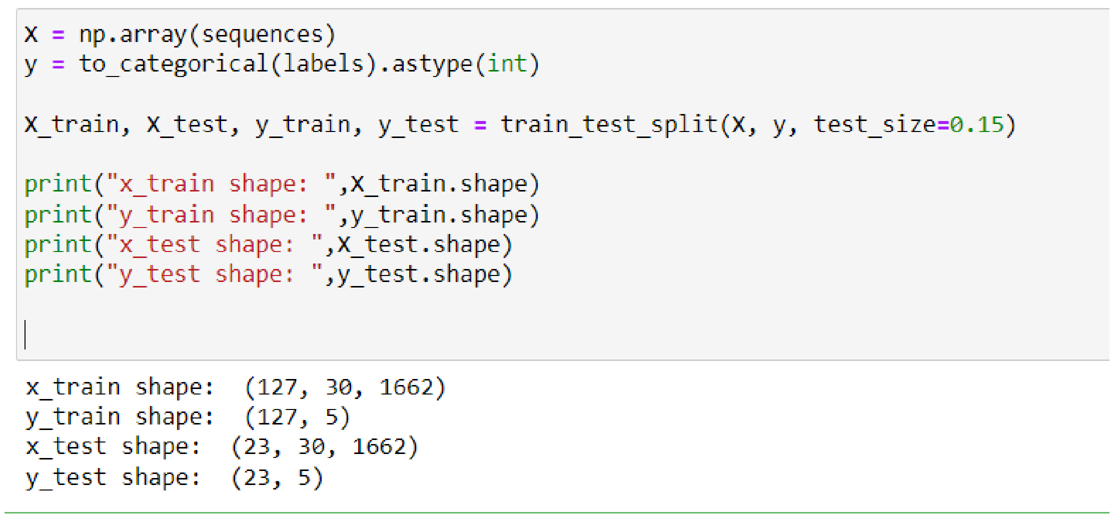
```
model.summary()
```

```
Model: "sequential_1"

┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_3 (LSTM)                   │ (None, 30, 64)         │       442,112 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_4 (LSTM)                   │ (None, 30, 128)        │        98,816 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm_5 (LSTM)                   │ (None, 64)             │        49,408 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 64)             │         4,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 32)             │         2,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_4 (Dense)                 │ (None, 5)              │           165 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 596,741 (2.28 MB)

 Trainable params: 596,741 (2.28 MB)

 Non-trainable params: 0 (0.00 B)
```

Figure 1.12

4. Model Training:

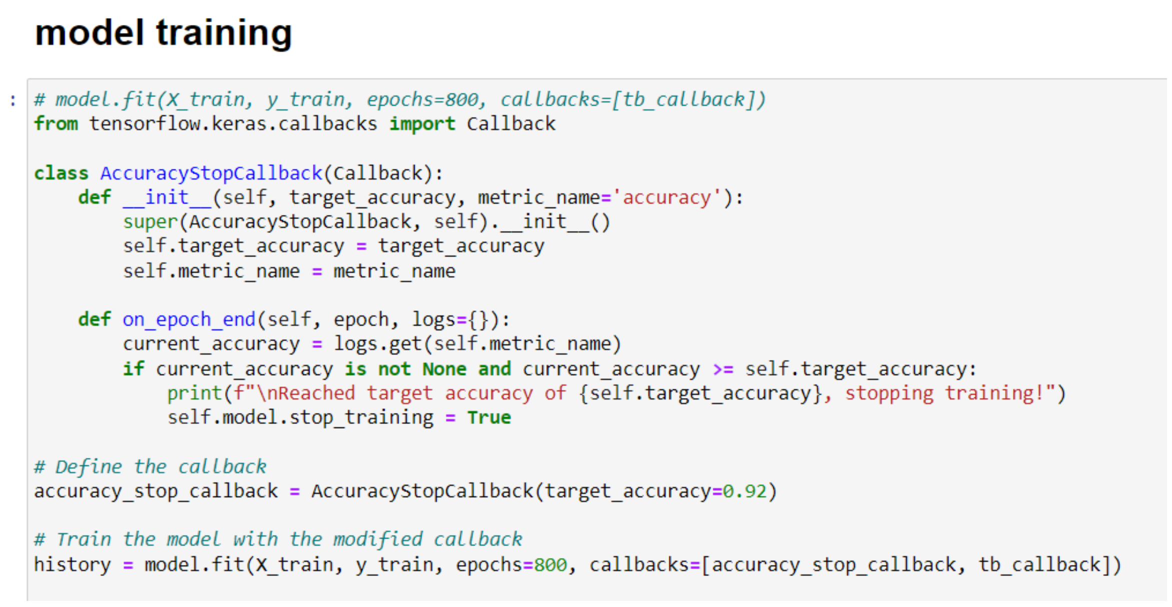
- Split the preprocessed data into training and validation sets.
- Train the LSTM model on the training set while monitoring performance on the validation set.
- Utilize techniques such as early stopping to prevent overfitting and save the best-performing model.
- Tune hyperparameters, including learning rate, batch size, and number of epochs, to optimize model performance.
- Monitor training progress using metrics such as loss and accuracy.



```python
X = np.array(sequences)
y = to_categorical(labels).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)

print("x_train shape: ",X_train.shape)
print("y_train shape: ",y_train.shape)
print("x_test shape: ",X_test.shape)
print("y_test shape: ",y_test.shape)
```

```
x_train shape:  (127, 30, 1662)
y_train shape:  (127, 5)
x_test shape:  (23, 30, 1662)
y_test shape:  (23, 5)
```

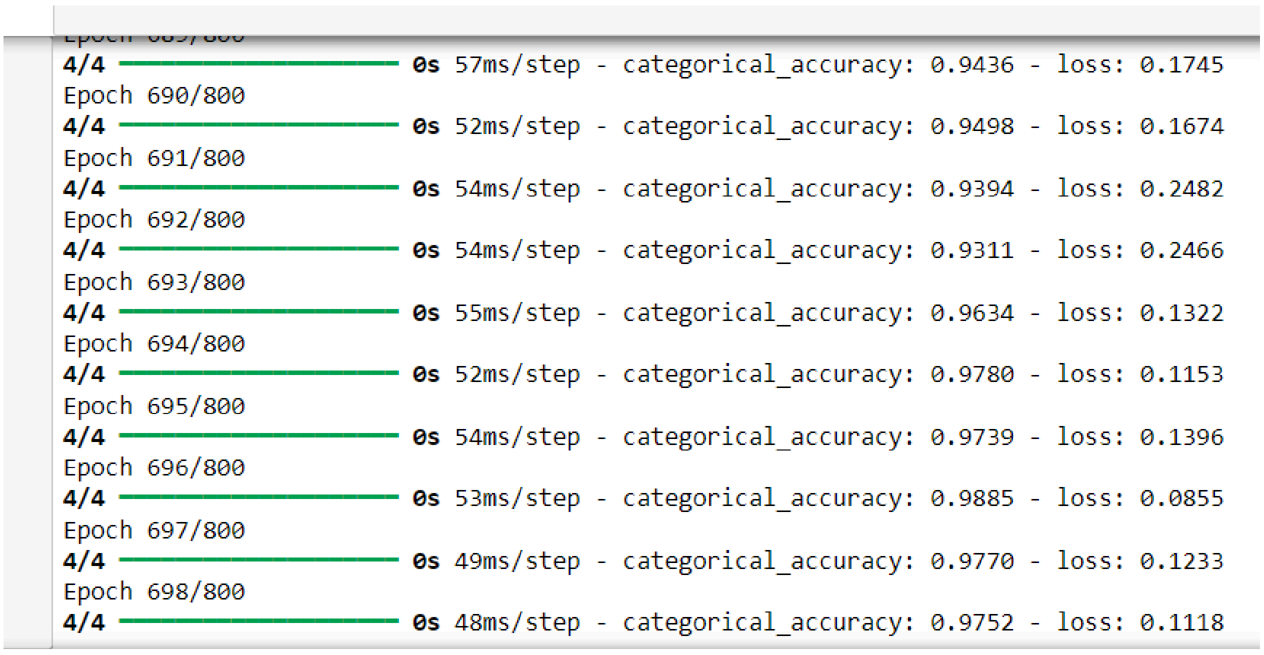
Figure 1.13



model training

```python
# model.fit(X_train, y_train, epochs=800, callbacks=[tb_callback])
from tensorflow.keras.callbacks import Callback

class AccuracyStopCallback(Callback):
    def __init__(self, target_accuracy, metric_name='accuracy'):
        super(AccuracyStopCallback, self).__init__()
        self.target_accuracy = target_accuracy
        self.metric_name = metric_name

    def on_epoch_end(self, epoch, logs={}):
        current_accuracy = logs.get(self.metric_name)
        if current_accuracy is not None and current_accuracy >= self.target_accuracy:
            print(f"\nReached target accuracy of {self.target_accuracy}, stopping training!")
            self.model.stop_training = True

# Define the callback
accuracy_stop_callback = AccuracyStopCallback(target_accuracy=0.92)

# Train the model with the modified callback
history = model.fit(X_train, y_train, epochs=800, callbacks=[accuracy_stop_callback, tb_callback])
```

Figure 1.14

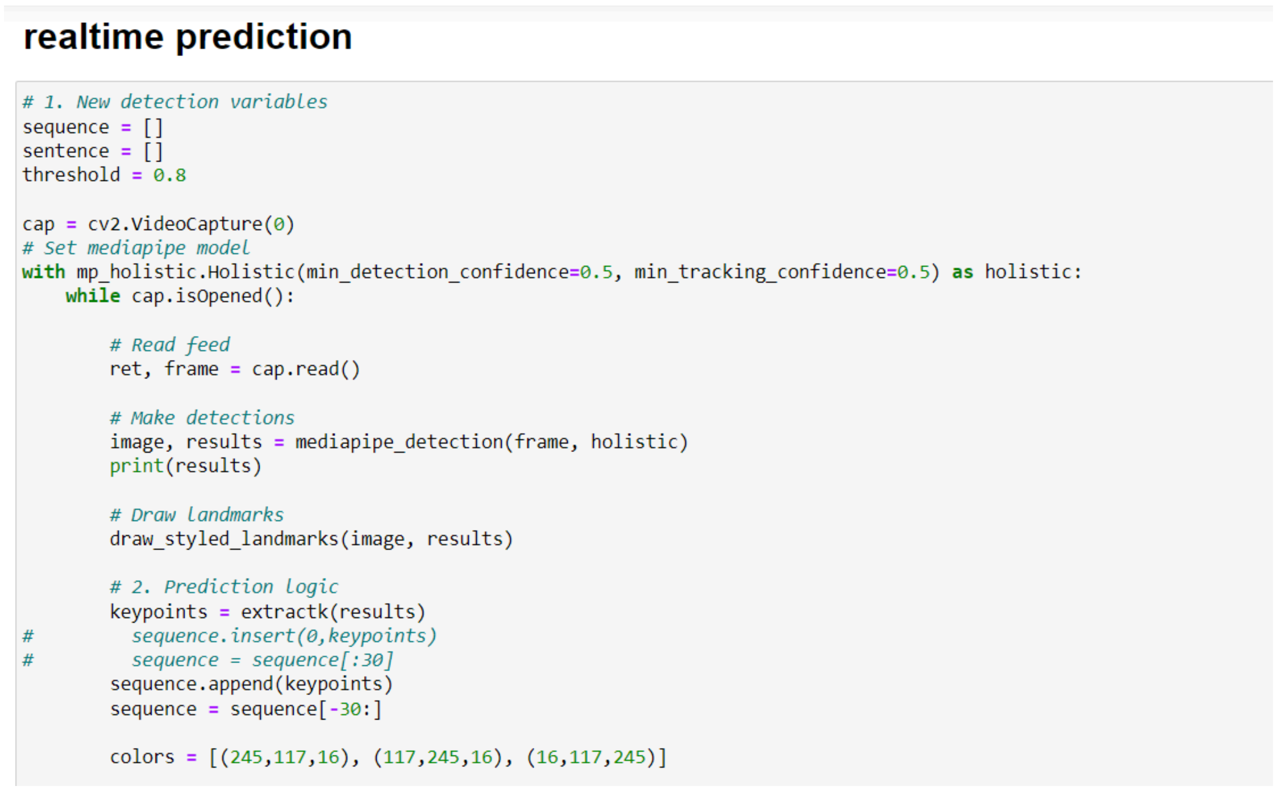


```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - categorical_accuracy: 0.9436 - loss: 0.1745
Epoch 690/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - categorical_accuracy: 0.9498 - loss: 0.1674
Epoch 691/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - categorical_accuracy: 0.9394 - loss: 0.2482
Epoch 692/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - categorical_accuracy: 0.9311 - loss: 0.2466
Epoch 693/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - categorical_accuracy: 0.9634 - loss: 0.1322
Epoch 694/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - categorical_accuracy: 0.9780 - loss: 0.1153
Epoch 695/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - categorical_accuracy: 0.9739 - loss: 0.1396
Epoch 696/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - categorical_accuracy: 0.9885 - loss: 0.0855
Epoch 697/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - categorical_accuracy: 0.9770 - loss: 0.1233
Epoch 698/800
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - categorical_accuracy: 0.9752 - loss: 0.1118
```

Figure 1.15

5. Real-time Hand Gesture Detection:

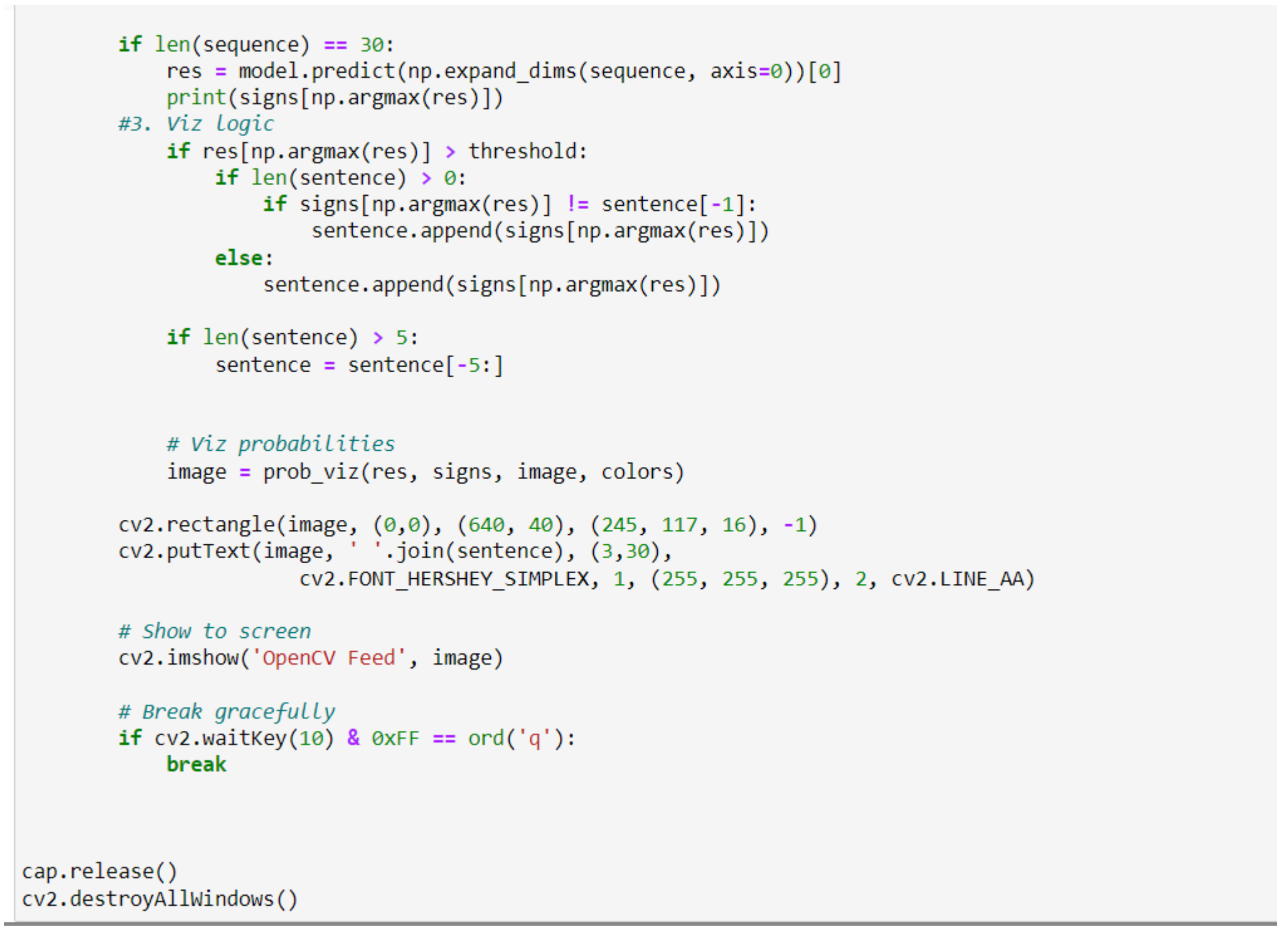
- Utilize the MediaPipe library for real-time hand gesture detection and landmark extraction from video feed.
- Preprocess the extracted hand landmarks to match the input format expected by the trained LSTM model.
- Implement logic to continuously feed sequences of hand landmarks into the LSTM model for prediction.
- Set a threshold for confidence level to filter out low-confidence predictions.
- Display the predicted sign language word or phrase in real-time alongside the video feed.



realtime prediction

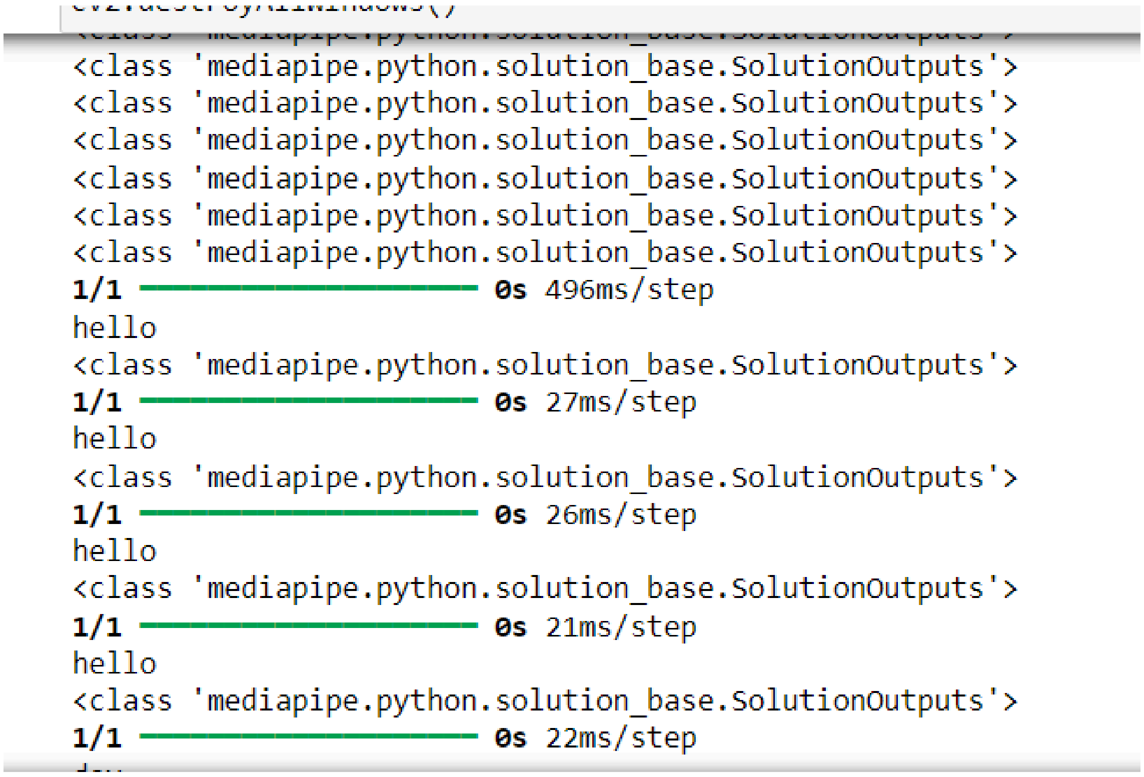
```python
# 1. New detection variables
sequence = []
sentence = []
threshold = 0.8

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extractk(results)
#         sequence.insert(0,keypoints)
#         sequence = sequence[:30]
        sequence.append(keypoints)
        sequence = sequence[-30:]

        colors = [(245,117,16), (117,245,16), (16,117,245)]
```

Figure 1.16



```python
        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(signs[np.argmax(res)])
        #3. Viz logic
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if signs[np.argmax(res)] != sentence[-1]:
                        sentence.append(signs[np.argmax(res)])
                else:
                    sentence.append(signs[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]


            # Viz probabilities
            image = prob_viz(res, signs, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                       cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)

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


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

Figure 1.16



```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 496ms/step
hello
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
hello
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
hello
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
hello
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Figure 1.17

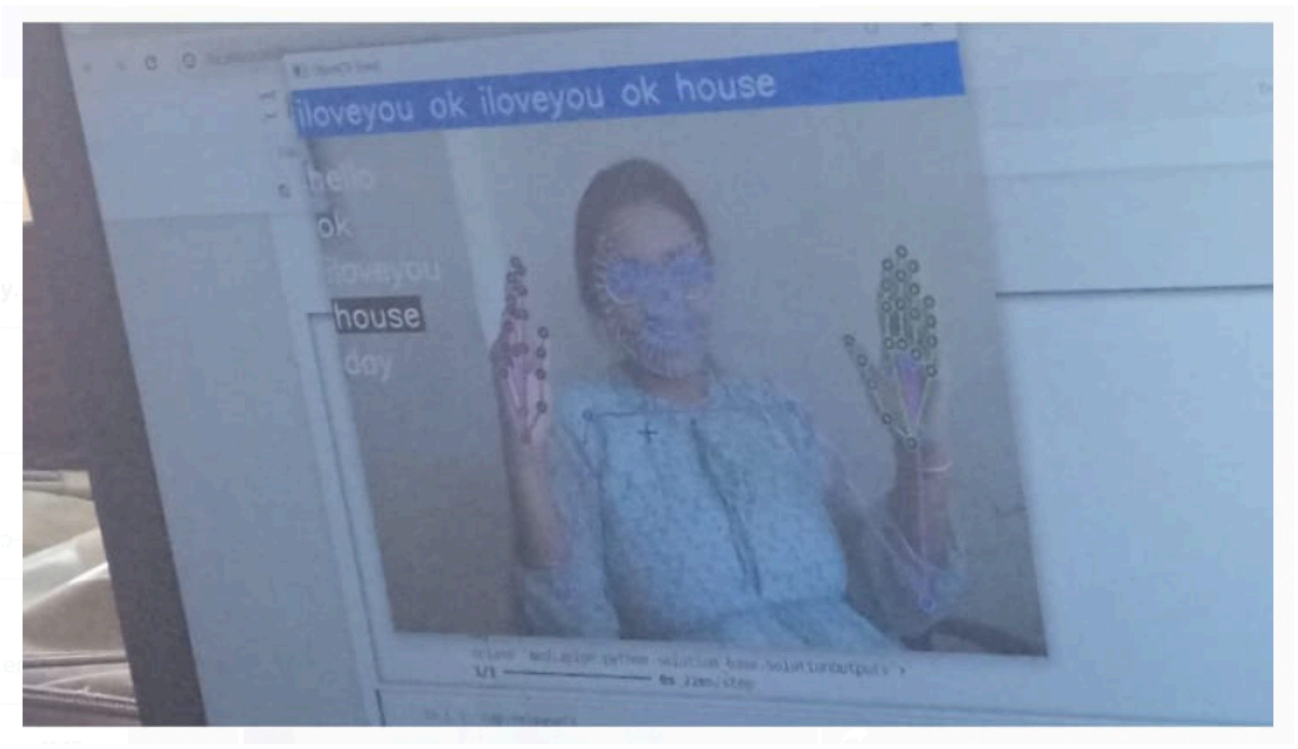
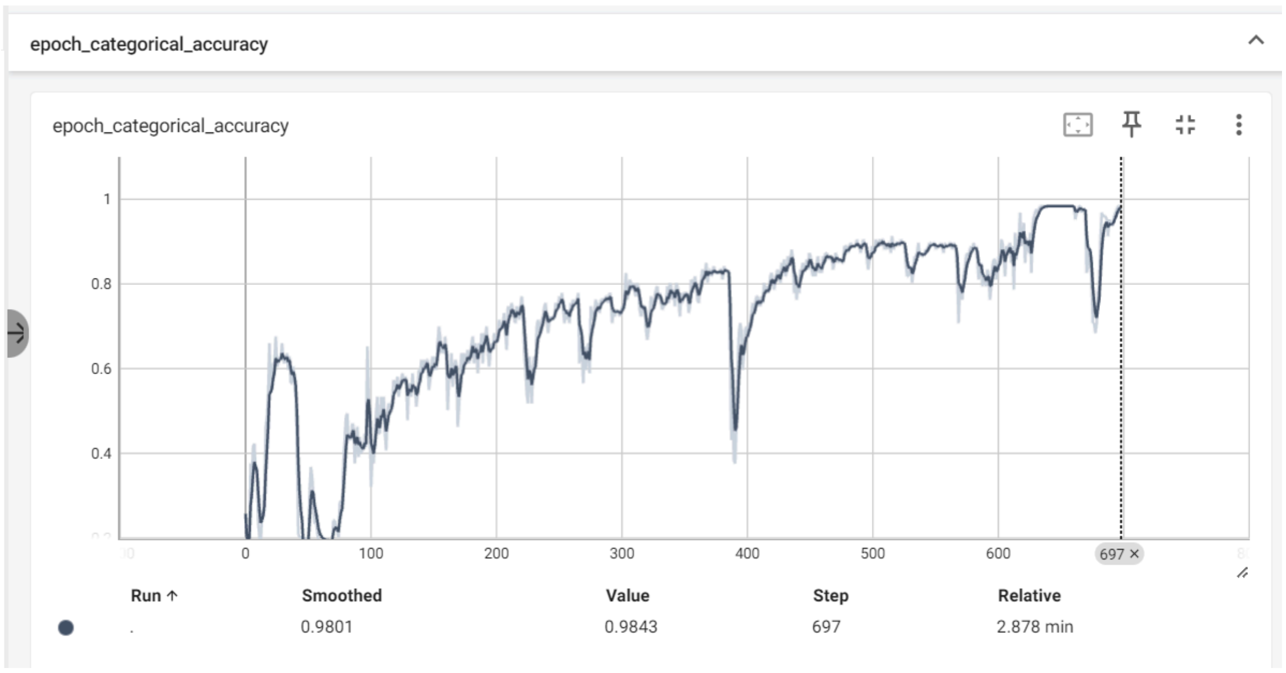


Figure 1.18

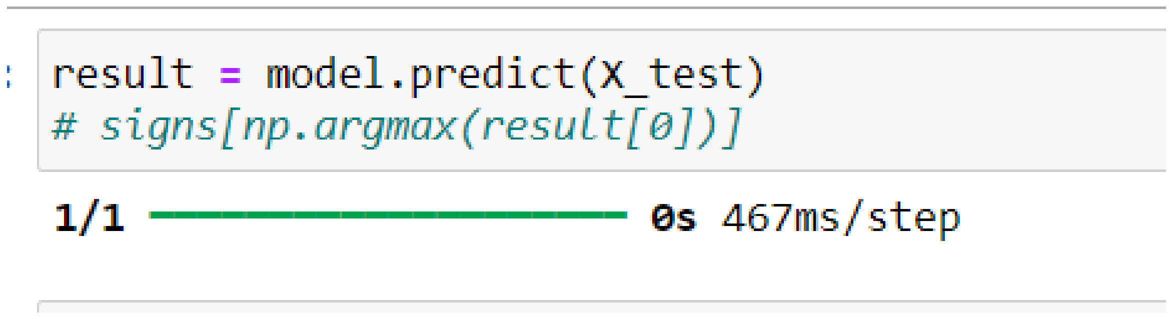


```
Run ↑   Smoothed   Value    Step   Relative
● .     0.9801     0.9843   697    2.878 min
```

Figure 1.19

6. Evaluation:

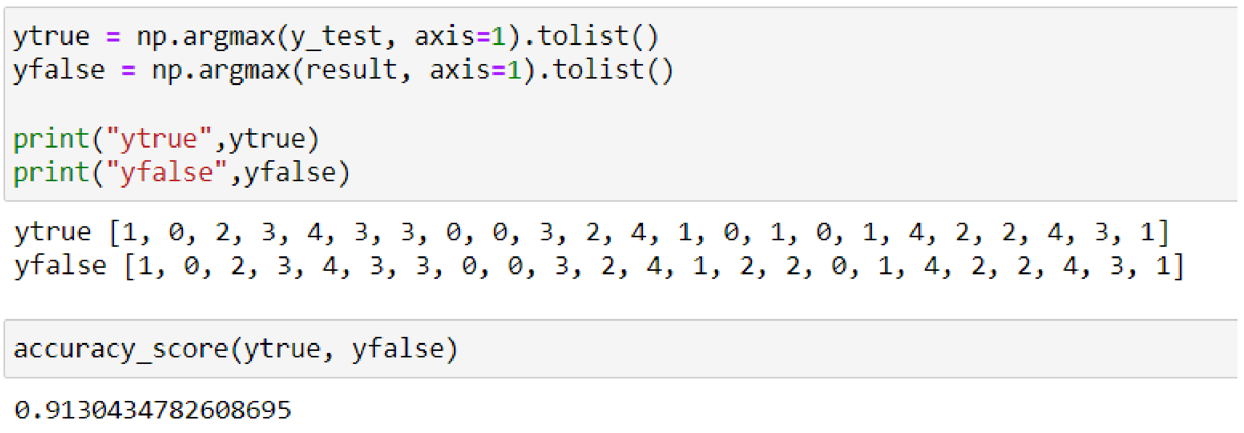
- Evaluate the trained LSTM model on a separate test set to assess its generalization performance.
- Measure metrics such as accuracy, precision, recall, and F1-score to quantify model performance.
- Conduct qualitative analysis by visually inspecting the real-time detection results and comparing them with ground truth.
- Perform error analysis to identify common misclassifications and areas for improvement.



```
result = model.predict(X_test)
# signs[np.argmax(result[0])]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 467ms/step
```

Figure 1.20



```
ytrue = np.argmax(y_test, axis=1).tolist()
yfalse = np.argmax(result, axis=1).tolist()

print("ytrue",ytrue)
print("yfalse",yfalse)
```

```
ytrue [1, 0, 2, 3, 4, 3, 3, 0, 0, 3, 2, 4, 1, 0, 1, 0, 1, 4, 2, 2, 4, 3, 1]
yfalse [1, 0, 2, 3, 4, 3, 3, 0, 0, 3, 2, 4, 1, 2, 2, 0, 1, 4, 2, 2, 4, 3, 1]
```

```
accuracy_score(ytrue, yfalse)
```

```
0.9130434782608695
```

Figure 1.21

7. Optimization and Fine-tuning:

- Explore techniques such as data augmentation to increase the diversity and size of the training dataset.
- Fine-tune model hyperparameters and architecture based on evaluation results and insights from error analysis.
- Experiment with advanced LSTM variants, such as bidirectional LSTM or attention mechanisms, to capture more complex temporal dependencies.
- Optimize inference speed and resource utilization for real-time deployment on various platforms, including desktops, mobile devices, and embedded systems.

8. Deployment:

- Package the trained LSTM model and associated preprocessing code into a deployable application or service.
- Develop user-friendly interfaces for interacting with the sign language detection system, including options for camera input and visualization of detected gestures.
- Ensure compatibility and performance across different hardware and operating systems.
- Conduct user testing and gather feedback to refine the system's usability and accessibility.

CONCLUSIONS

In conclusion, the developed gesture recognition system demonstrates promising capabilities in accurately recognizing hand gestures in real-time. By leveraging deep learning techniques and the MediaPipe framework, the system achieves efficient detection and classification of gestures, contributing to various applications such as sign language translation, human-computer interaction, and augmented reality. The model's performance, evaluated through metrics like accuracy and real-time responsiveness, indicates its effectiveness in understanding and interpreting user gestures.

However, there remain areas for enhancement, such as optimizing model architecture, refining training data, and improving user experience. Despite these potential improvements, the project underscores the feasibility and potential of employing deep learning and computer vision for developing intuitive and interactive systems that bridge the gap between humans and machines. Overall, the project lays a foundation for future advancements in gesture recognition technology and its integration into diverse domains.

FUTURE ENHANCEMENTS

Moving forward, there are several avenues for further research and enhancement of the gesture recognition system. Future references include:

Enhanced Model Architectures: Exploration of more complex deep learning architectures, including attention mechanisms and multi-modal fusion, to improve accuracy and robustness in gesture recognition.
Data Augmentation: Integration of data augmentation techniques to enhance model generalization and reduce overfitting, particularly in scenarios with limited training data.
Real-time Optimization: Optimization of real-time performance by implementing hardware acceleration techniques and model compression methods for efficient inference on resource-constrained devices.
User Interface Refinement: Refinement of the user interface to enhance user experience and accessibility, including interactive feedback mechanisms and customizable gesture sets.
Multimodal Integration: Investigation of multimodal approaches, such as combining hand gestures with voice commands or facial expressions, to develop more comprehensive human-computer interaction systems.By addressing these areas, the gesture recognition system can evolve into a robust and versatile tool for various real-world applications, contributing to advancements in human-computer interaction and accessibility technologies.

REFERENCES

1. https://www.sciencedirect.com/topics/computer-science/sign-language-recognition
2. https://www.nature.com/articles/s41598-023-43852-x
3. https://www.researchgate.net/publication/357618861_Sign_Language_Recognition_System_u sin g_TensorFlow_Object_Detection_API
4. https://www.researchgate.net/publication/262187093_Sign_language_recognition_State_of_th e_ art
5. https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API

   https://github.com/HumanSignal/labelImg
6. https://www.ijert.org/a-review-paper-on-sign-language-recognition-for-the-deaf-and-dumb
7. https://www.researchgate.net/publication/361079073_Deepsign_Sign_Language_Detection_an d_Recognition_Using_Deep_Learning