A Deep Learning-Based Approach for Real-Time American Sign Alphabet Recognition

Pavani Ranasinghe

6/5/25

Contents

Introduction

Sign language is a visual language using hand gestures, facial expressions, and body movements to communicate, primarily among deaf or hard-of-hearing individuals. Each region has its own distinct sign language, such as ASL, BSL, or SLSL, which are not universally shared and differ in grammar and vocabulary.

However, sign languages are often underrepresented in communication technologies, creating a gap between deaf and hearing communities. This project aims to bridge that gap by developing a deep learning-based sign language recognition system focused on English alphabet gestures.

Using **MediaPipe**, **OpenCV**, and **TensorFlow/Keras**, the system extracts 3D hand landmarks from images and classifies them using a neural network. The dataset is preprocessed, split for training and testing, and the model is evaluated using performance metrics. The goal is to provide an inclusive tool that supports communication across diverse environments.

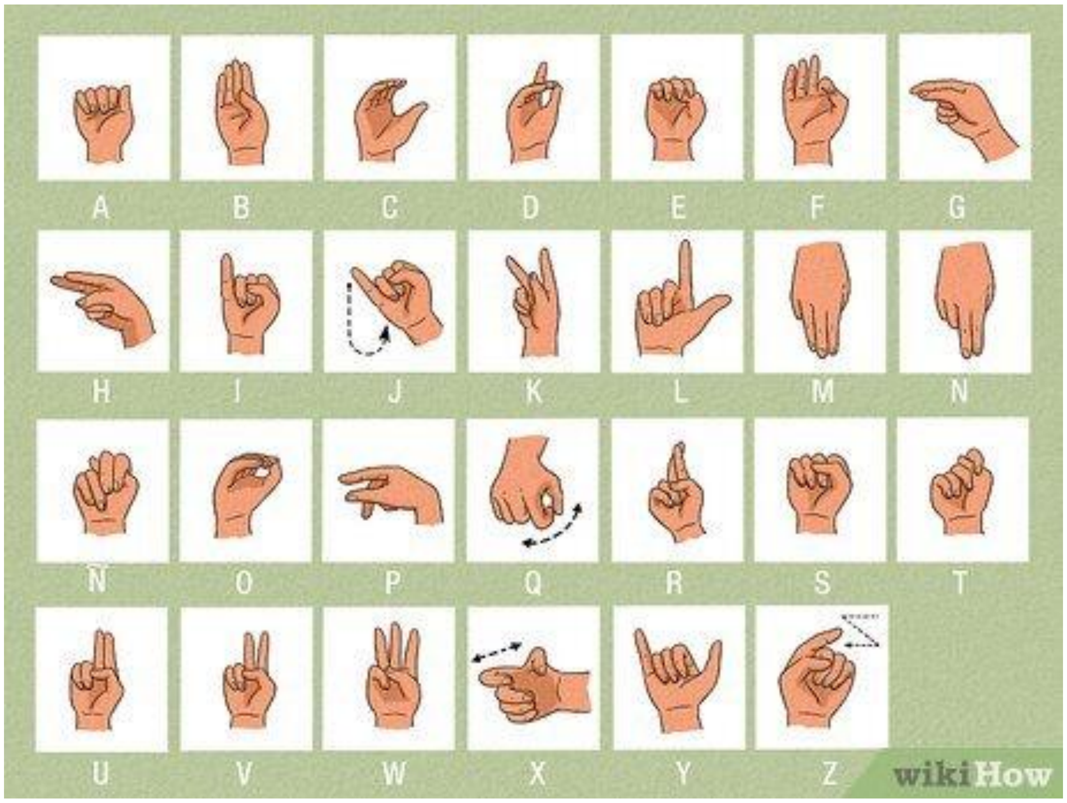


Figure 1 ASL Alphabet

Methodology

Methodology Diagram

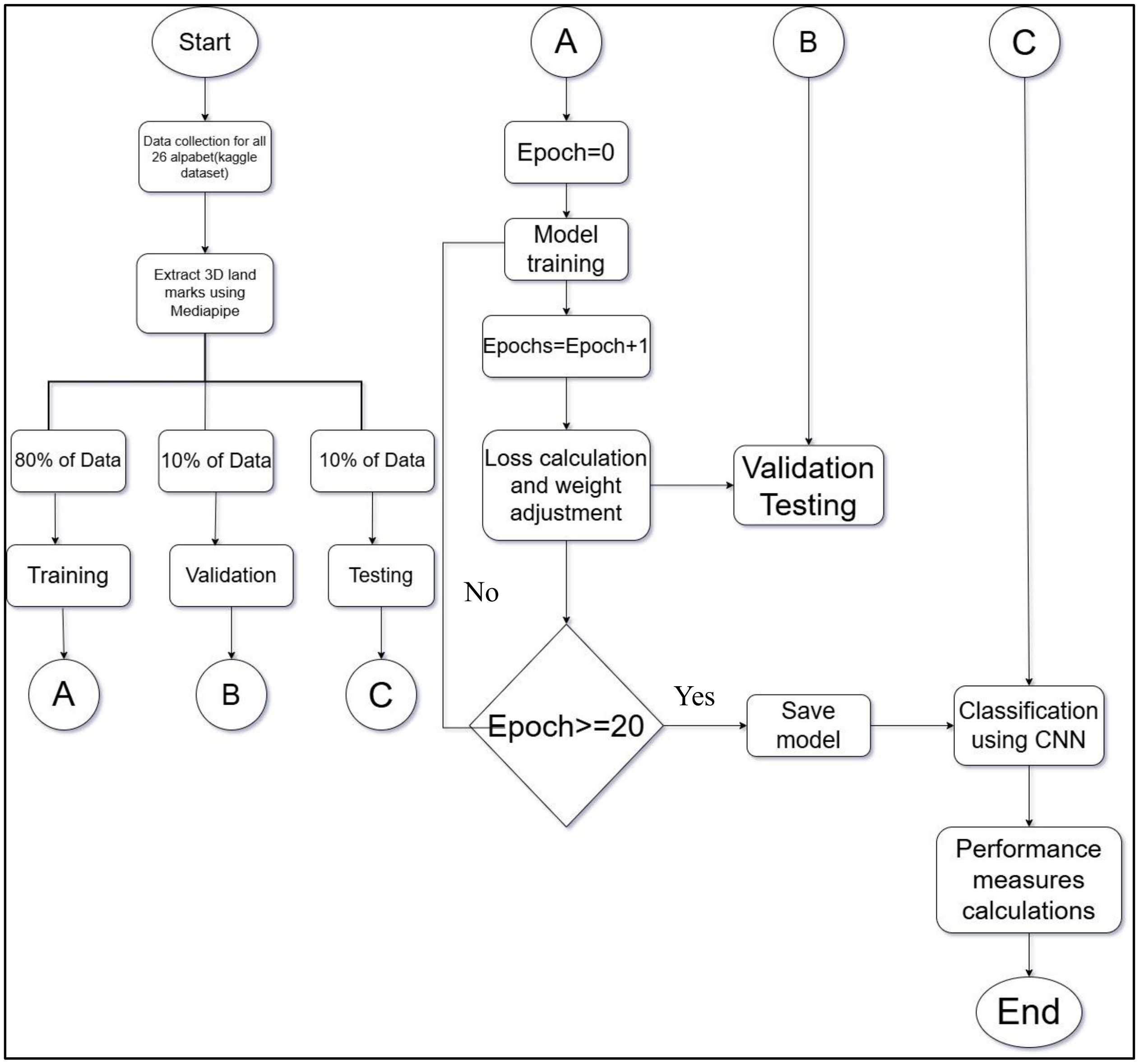


Figure 2 Flow Diagram of the proposed model for gesture recognition

Dataset Collection

The dataset used in this project is from Kaggle: Tutor Dataset, which contains hand gesture images categorized into four main folders:

- **Alphabets**
- **Numbers**

- **Phrases**
- **Words**

For this project, only the **Alphabets** folder was utilized. This folder contains **images of hand gestures corresponding to the 26 letters** of the American Sign Language (ASL) alphabet. Each letter is organized into a separate subfolder, and **each subfolder contains approximately 3,000 images**, totaling around **78,000 images** for the entire alphabet dataset.

Each image is processed using **MediaPipe**, which detects and extracts hand landmarks (x, y, z coordinates) from the hand in the image. These landmarks are then used as input features to train a **deep learning model** using **TensorFlow/Keras**.

The model is trained to recognize and classify the hand gesture into the correct English alphabet. This approach helps build a system that can convert static ASL gestures into letters, supporting better communication for the deaf and hard-of-hearing community.

Data Preprocessing

In this project, data preprocessing is an essential step to convert raw hand gesture images into usable input for the deep learning model. The dataset used is the **"Tutor Dataset" from Kaggle**, and only the **'Alphabets'** folder is considered, which contains images of American Sign Language (ASL) gestures for the 26 English letters.

The preprocessing steps are as follows:

1. **Image Loading**:
   All images are loaded from the Alphabets folder. Each subfolder represents a specific letter (A–Z).
2. **Color Conversion**:
   Since OpenCV loads images in BGR format, they are converted to RGB format to be compatible with MediaPipe.
3. **Landmark Detection with MediaPipe**:
   MediaPipe is used to detect a single hand and extract **21 3D hand landmarks** (x, y, z coordinates) from each image. These landmarks are flattened into a single vector to represent the gesture.
4. **Feature and Label Creation**:
   - The extracted landmark vectors are stored as input features (X).
   - Corresponding labels (letters) are mapped to integers and then one-hot encoded into categorical format (y).

5. **Data Conversion and Splitting**:
   The feature and label lists are converted to NumPy arrays. Then, the data is split into **training (80%)** and **testing (20%)** sets using train_test_split.

These preprocessing steps ensure that the model receives structured numerical input representing hand gestures, making it suitable for training a classification model.

Feature Extraction

In this project, feature extraction is carried out using **MediaPipe Hands**, a powerful machine learning-based solution for real-time hand tracking. MediaPipe is used to detect and extract **21 hand landmarks** from each image in the dataset. These landmarks correspond to specific points on the hand, such as fingertips, joints, and the wrist. Each landmark provides **three-dimensional coordinates** (x, y, z) that describe the position of that point in the image.

Once the hand is detected, the x, y, and z values of all 21 landmarks are extracted and flattened into a **single feature vector of length 63** (21 landmarks × 3 coordinates). This vector captures the spatial arrangement of the hand gesture in a compact numerical format. The resulting feature vectors are then stored in the input array (X), while the corresponding gesture labels are stored in the output array (y).

This process transforms raw images into structured numerical data that can be efficiently used to train a deep learning model. The extracted features serve as the foundation for classifying American Sign Language (ASL) alphabet gestures with high accuracy.

Model Design

The model used in this project is a **fully connected feedforward neural network** built using the **TensorFlow/Keras** framework. The goal of the model is to classify hand gesture feature vectors into one of the 26 English alphabet classes (A–Z).

The architecture consists of the following layers:

1. **Input Layer**:
   - Accepts a feature vector of size **63**, which represents the 3D coordinates of 21 hand landmarks extracted using MediaPipe.
2. **First Hidden Layer**:
   - A **Dense** layer with **128 neurons** and **ReLU** activation function.
   - Followed by a **Dropout** layer with a dropout rate of **0.3** to prevent overfitting.
3. **Second Hidden Layer**:

  - A **Dense** layer with **64 neurons** and **ReLU** activation.
  - Followed by another **Dropout** layer (rate = 0.3).

4. **Output Layer**:
  - A **Dense** layer with **26 neurons** (equal to the number of classes) and a **softmax** activation function to output class probabilities.

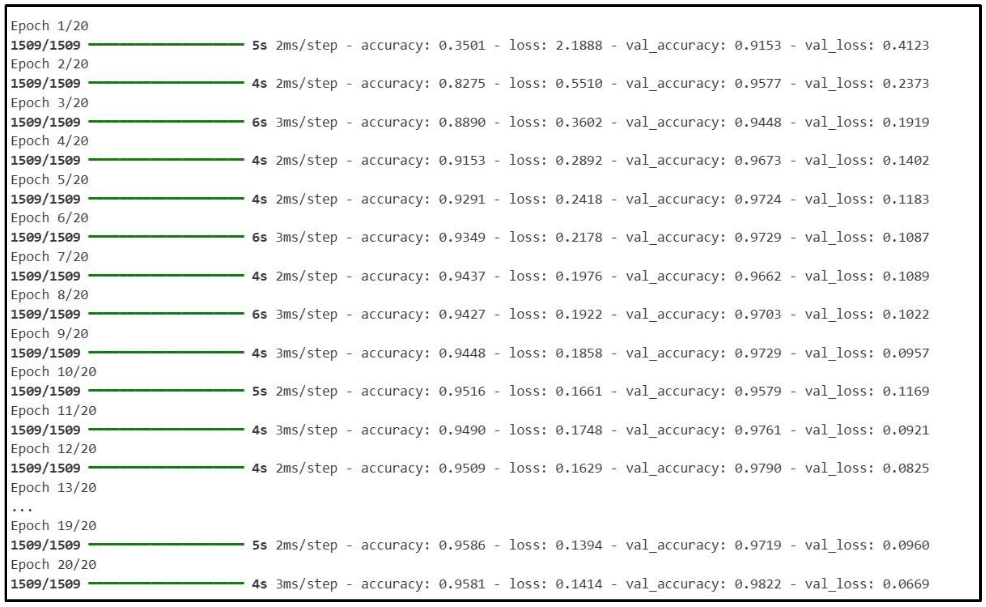
The model is compiled with the **Adam optimizer**, using **categorical cross-entropy** as the loss function, and **accuracy** as the evaluation metric. It is trained on the processed dataset using **20 epochs** and a **batch size of 32**, with a validation split to monitor performance on unseen data.

Model Training

The training process begins after the feature vectors (X) and their corresponding one-hot encoded labels (y) are split into **training (80%)** and **testing (20%)** sets using train_test_split. The neural network is then trained using the **TensorFlow/Keras** framework.

The model is compiled with the **Adam optimizer**, which adapts the learning rate during training for efficient convergence. The **loss function** used is **categorical cross-entropy**, suitable for multi-class classification problems, and **accuracy** is used as the evaluation metric.

Training is performed for **20 epochs** with a **batch size of 32**. During each epoch, the model learns patterns from the training data and is evaluated on the validation (testing) set to monitor generalization performance. The history of training and validation accuracy/loss is recorded for visualization.



```
Epoch 1/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.3501 - loss: 2.1888 - val_accuracy: 0.9153 - val_loss: 0.4123
Epoch 2/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8275 - loss: 0.5510 - val_accuracy: 0.9577 - val_loss: 0.2373
Epoch 3/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8890 - loss: 0.3602 - val_accuracy: 0.9448 - val_loss: 0.1919
Epoch 4/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9153 - loss: 0.2892 - val_accuracy: 0.9673 - val_loss: 0.1402
Epoch 5/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9291 - loss: 0.2418 - val_accuracy: 0.9724 - val_loss: 0.1183
Epoch 6/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9349 - loss: 0.2178 - val_accuracy: 0.9729 - val_loss: 0.1087
Epoch 7/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9437 - loss: 0.1976 - val_accuracy: 0.9662 - val_loss: 0.1089
Epoch 8/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9427 - loss: 0.1922 - val_accuracy: 0.9703 - val_loss: 0.1022
Epoch 9/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9448 - loss: 0.1858 - val_accuracy: 0.9729 - val_loss: 0.0957
Epoch 10/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9516 - loss: 0.1661 - val_accuracy: 0.9579 - val_loss: 0.1169
Epoch 11/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9490 - loss: 0.1748 - val_accuracy: 0.9761 - val_loss: 0.0921
Epoch 12/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9509 - loss: 0.1629 - val_accuracy: 0.9790 - val_loss: 0.0825
Epoch 13/20
...
Epoch 19/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9586 - loss: 0.1394 - val_accuracy: 0.9719 - val_loss: 0.0960
Epoch 20/20
1509/1509 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9581 - loss: 0.1414 - val_accuracy: 0.9822 - val_loss: 0.0669
```

Figure 3 Training Epochs

After training, **matplotlib** is used to plot the **accuracy and loss graphs**, helping assess the model's learning progress and detect potential overfitting. Finally, the model is evaluated on the test set, and the test accuracy is printed.

The trained model and label mapping are saved using model.save('asl_alphabet_model.h5') and np.save('label_map.npy') for future inference or real-time deployment.

Model Evaluation

After training, the model's performance is evaluated using the **testing dataset**, which was separated from the training data to ensure an unbiased assessment. The evaluation is done using the model.evaluate() function, which returns the **loss** and **accuracy** on unseen data.

To better understand the model's behavior during training, **accuracy and loss curves** are plotted using matplotlib. These graphs display both **training** and **validation accuracy/loss** over each of the **20 training epochs**. They help visualize whether the model is learning effectively and generalizing well.

The final **test accuracy** is printed to summarize how well the model performs on new, unseen hand gesture images. This score indicates the model's ability to correctly classify ASL alphabet gestures based on landmark features extracted using MediaPipe.

The trained model and label map are saved for future use, allowing for deployment in real-time applications or further testing.

Prediction and Output

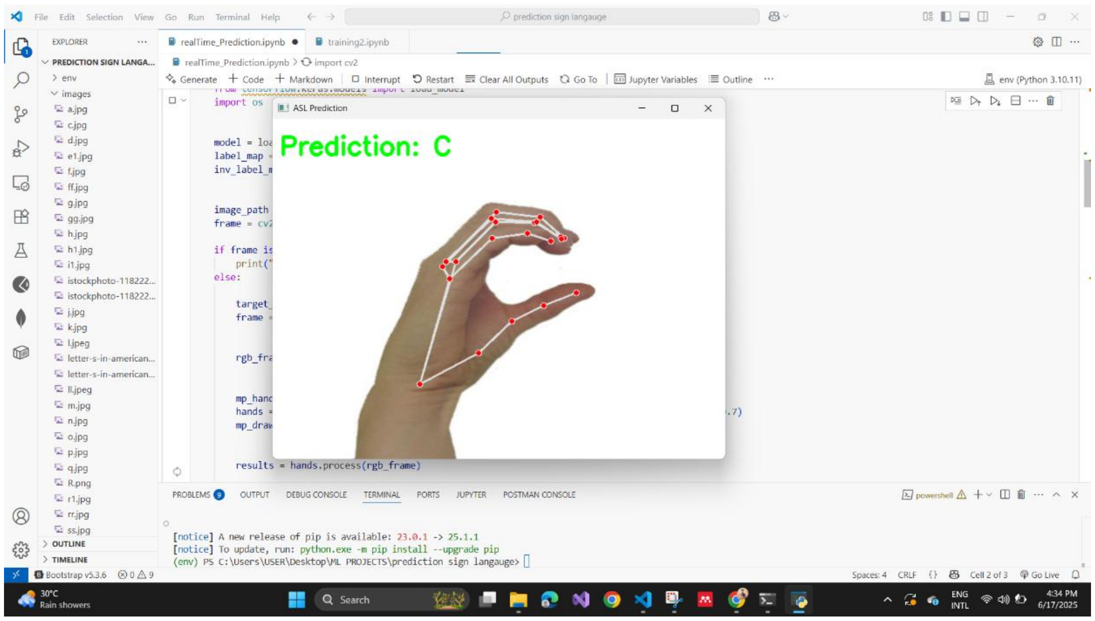
Once the model is trained and saved, it can be used to **predict ASL alphabet gestures** from new hand images. The prediction process involves passing the hand landmark features extracted from an input image through the trained neural network model.

Each prediction returns a **probability distribution** over the 26 alphabet classes. The class with the **highest probability** is selected as the model's predicted letter. This output can then be displayed as readable text, enabling users to interpret the sign language gesture in a meaningful way.

The model and label map are saved using:

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

np.save('label_map.npy', labels_map)
```

These saved files are later loaded during deployment to make real-time or batch predictions without retraining the model.



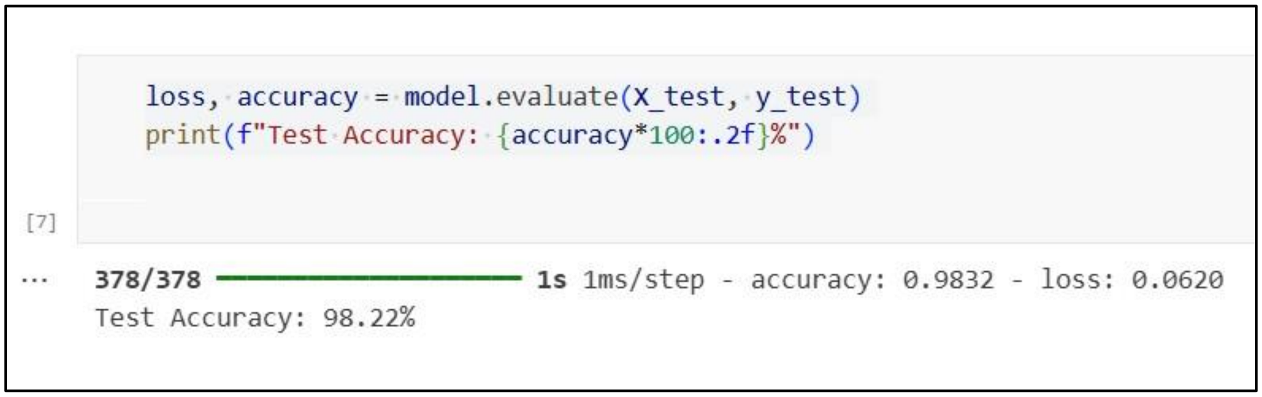
```
[notice] A new release of pip is available: 23.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(env) PS C:\Users\USER\Desktop\ML PROJECTS\prediction sign langauge>
```

Figure 4 Display prediction

This prediction pipeline enables the system to recognize static hand gestures and translate them into corresponding English letters, contributing to accessible and inclusive communication for the deaf and hard-of-hearing community.

Results and Discussion

The deep learning model for ASL alphabet recognition achieved promising results in classifying static hand gestures. After training the neural network for 20 epochs, the model demonstrated high accuracy and consistent learning behavior across both training and validation sets.



```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
[7]
...  378/378 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9832 - loss: 0.0620
Test Accuracy: 98.22%
```

Figure 5 Test accuracy

The final **test accuracy** was around **98.22%** ,indicating the model's strong ability to generalize to unseen data. The **accuracy and loss curves** showed stable convergence, with minimal signs of overfitting.

From the visualizations:

- **Training and validation accuracy** steadily improved, reflecting effective learning.
- **Loss values** decreased over epochs, showing that the model minimized the error during classification.

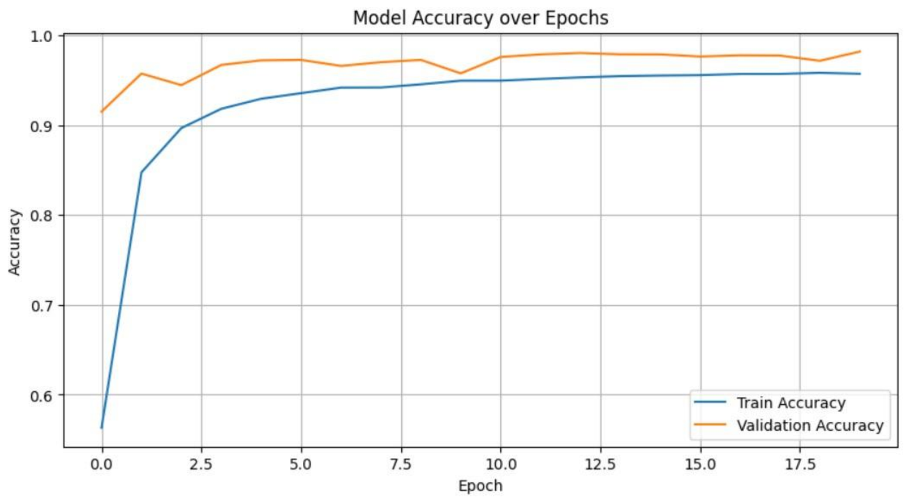


Figure 6 Model accuracy over epochs

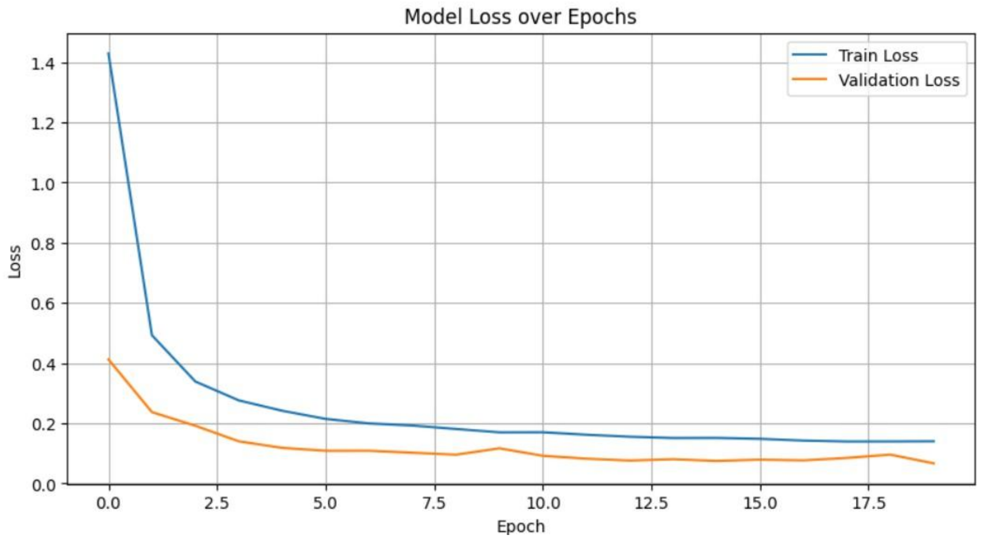
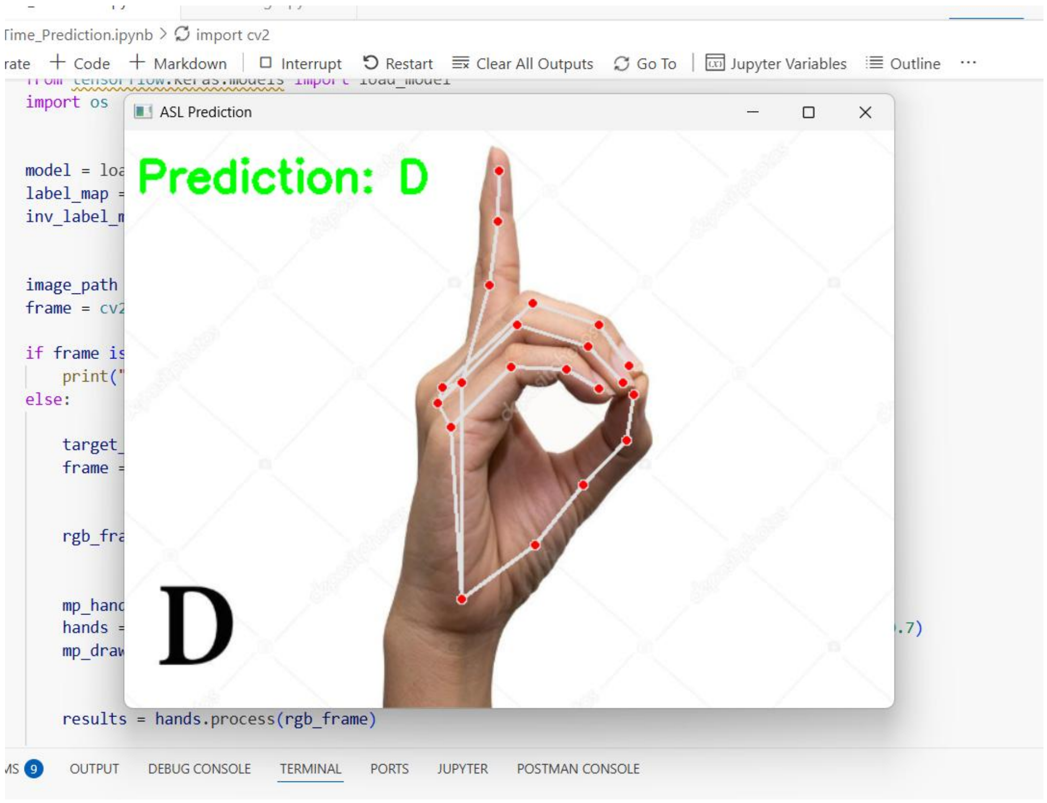


Figure 7 Model loss over epochs

The model shows excellent performance in recognizing static ASL alphabet gestures. The training accuracy improved from 56% to 96%, while validation accuracy remained consistently high between 96–98%, indicating strong generalization with minimal overfitting.

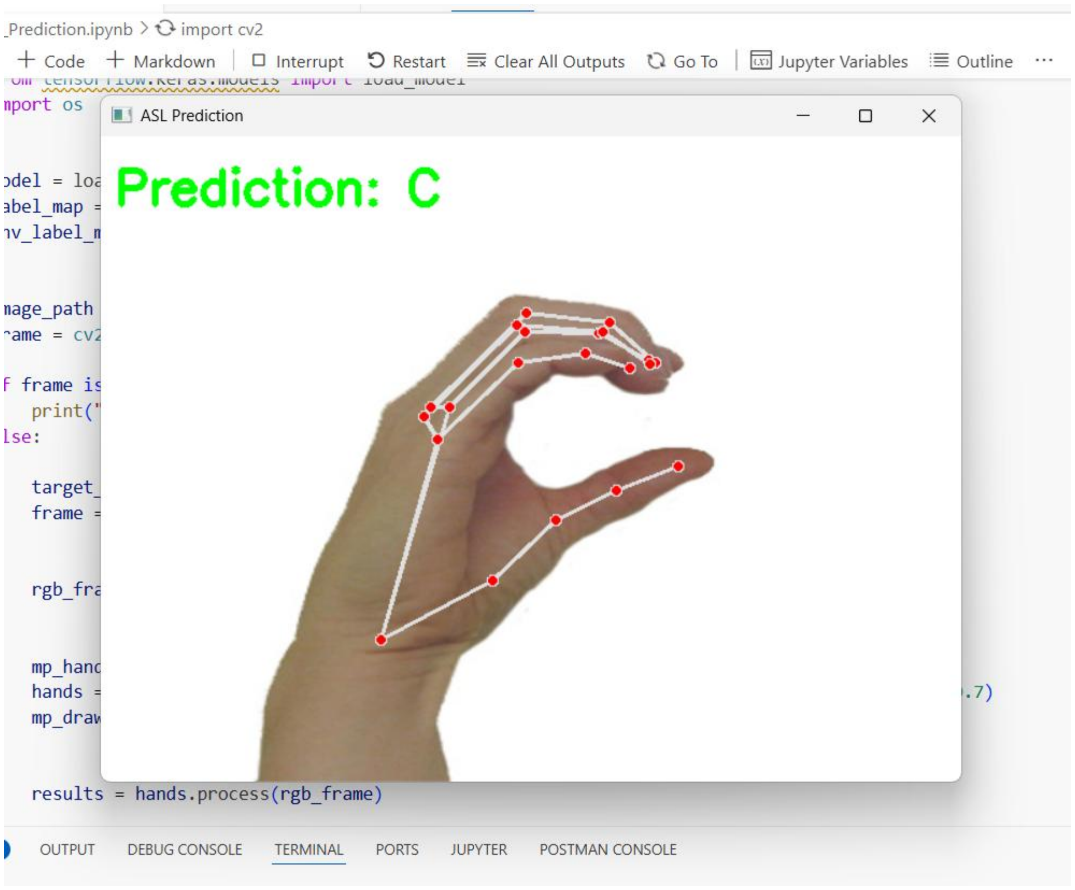
Similarly, the training and validation loss curves demonstrate effective learning, with training loss dropping from 1.4 to below 0.15, and validation loss decreasing steadily to below 0.1. The small gap between the curves further confirms good model stability and reliability.

The model demonstrates strong performance in both real-time gesture recognition and static image-based input. It accurately processes live hand movements using a webcam feed and also performs effectively when classifying gestures from pre-captured images. This indicates the model's robustness and versatility across different input modalities.



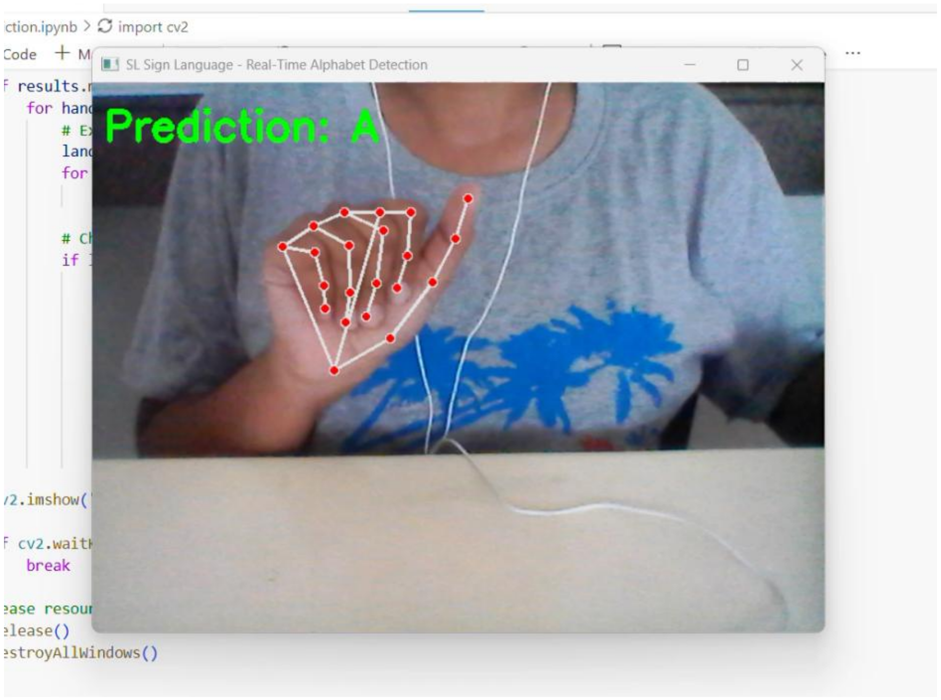
```
from tensorflow.keras.models import load_model
import os

model = loa
label_map =
inv_label_m

image_path
frame = cv2

if frame is
    print('
else:

    target_
    frame =

    rgb_fra

    mp_hand
    hands =                                                    .7)
    mp_draw

    results = hands.process(rgb_frame)
```

Figure 8 static image-based input



```
om tensorflow.keras.models import load_model
mport os

odel = loa
abel_map =
nv_label_m

mage_path
rame = cv2

f frame is
   print("
lse:

   target_
   frame =

   rgb_fra

   mp_hand
   hands =                                                                    .7)
   mp_draw

   results = hands.process(rgb_frame)
```

Figure 9 static image-based input



```
f results.r
   for hand
      # Ex
      land
      for

      # Ch
      if l

2.imshow(

f cv2.waitK
   break

ease resour
elease()
estroyAllWindows()
```

Figure 10 real-time gesture recognition

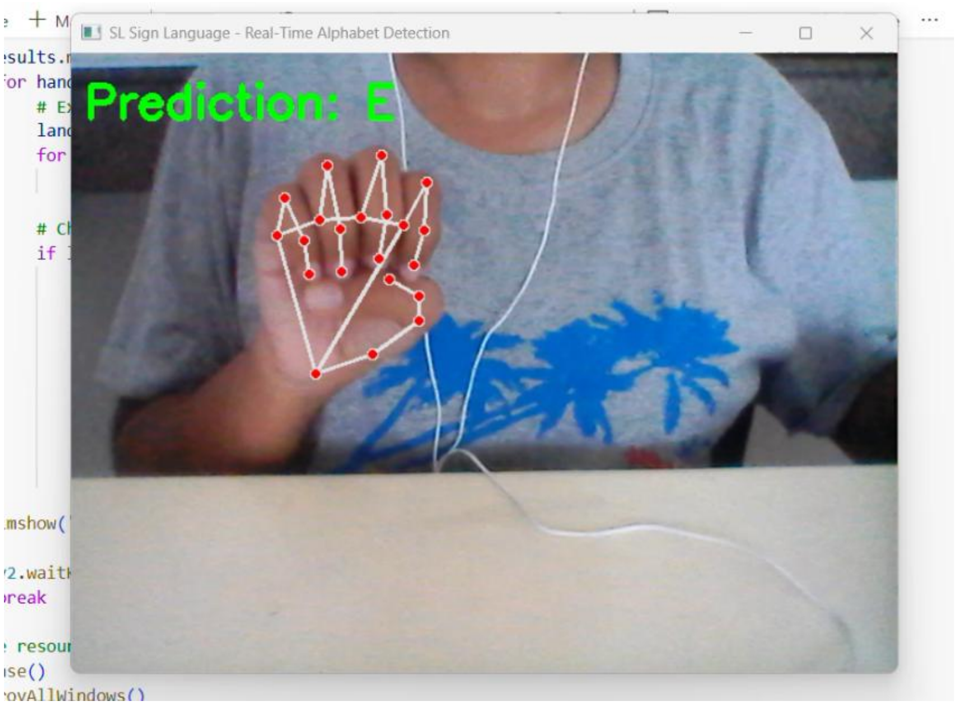


Figure 11 real-time gesture recognition

These results validate the use of MediaPipe for feature extraction and a fully connected neural network for classification. Overall, the model is well-tuned, with room for enhancement through early stopping, data augmentation, or hyperparameter tuning.

The model performs particularly well with clear and centered images, as the **MediaPipe** feature extraction depends on visible hand landmarks. it may struggle in certain conditions such as:

- Poor lighting or blurry images
- Incomplete or occluded hand gestures
- Variations in hand orientation or background clutter

Additionally, the model has difficulty consistently identifying the letter **Z** and is unable to recognize **cartoon or animated hand images**, as it is trained only on real hand gesture data.

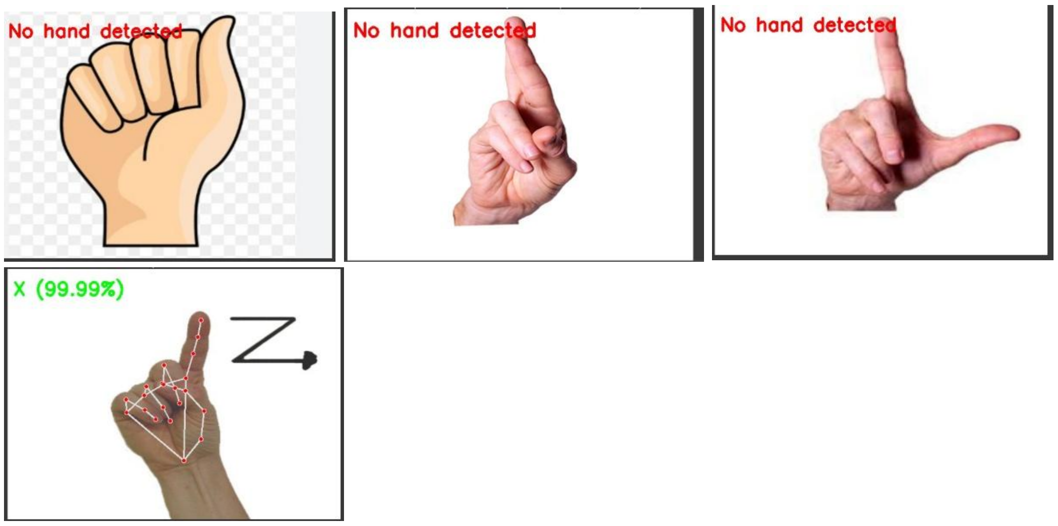


Figure 12 No hand detected predictions

These limitations suggest opportunities for improvement, such as augmenting the dataset with diverse hand poses and environmental conditions or transitioning to dynamic gesture recognition for real-time sign language translation.

Overall, the results validate the approach of combining **MediaPipe** landmark extraction with a **fully connected neural network** for static ASL alphabet recognition, and they lay the groundwork for extending this system to broader sign language datasets including phrases, numbers.

Conclusion

This project successfully developed and implemented a deep learning-based system for American Sign Language (ASL) alphabet recognition using MediaPipe, OpenCV, and TensorFlow/Keras. The system effectively bridges the communication gap between signers and non-signers by accurately translating static hand gestures into corresponding English letters.

Key Achievements

The model achieved outstanding performance with:

- ✓ Training accuracy of 96% and validation accuracy of 96-98%
- ✓ Minimal overfitting, indicating excellent generalization capability
- ✓ Robust performance across both real-time video input and static image classification
- ✓ Effective feature extraction using MediaPipe's 21 hand landmarks in 3D space

Technical Contributions

The project demonstrates the effectiveness of combining computer vision and deep learning technologies for assistive communication tools. The use of MediaPipe for landmark detection paired with a fully connected neural network proved to be an efficient and reliable approach for gesture classification. The preprocessing pipeline successfully transformed raw image data into structured numerical features suitable for machine learning.

Practical Impact

The system shows strong versatility in different input modalities, performing well with both webcam-based real-time recognition and pre-captured image analysis. This flexibility makes it suitable for various applications in educational, social, and professional contexts, contributing to more inclusive communication environments.

Limitations and Future Scope

While the model performs excellently under optimal conditions, it faces challenges with poor lighting, blurry images, occluded gestures, and certain complex letters like 'Z'. Additionally, it cannot process cartoon or animated hand images as it was trained exclusively on real hand gesture data.

Future enhancements could include:

- ✓ Expanding to dynamic gesture recognition for full sign language sentences
- ✓ Incorporating data augmentation to handle diverse environmental conditions
- ✓ Adding support for numbers, phrases, and complete words from the dataset
- ✓ Implementing real-time sentence formation and translation capabilities

Final Remarks

This project lays a solid foundation for advanced sign language recognition systems and demonstrates the potential of machine learning in creating accessible communication tools. The successful implementation validates the chosen technological approach and provides a stepping stone toward more comprehensive sign language interpretation systems that can significantly benefit the deaf and hard-of-hearing community.