
SIGN LANGUAGE TO TEXT USING CONVOLUTIONAL NEURAL NETWORK



Submitted by:

Name: Arsh Mohan Nishant

Course: B.Tech (3rd Year)

Field of Interest: Data Science

Email: arshmohan789@gmail.com

GitHub: github.com/arsh-mohan6

LinkedIn: www.linkedin.com/in/arsh-mohan-nishant-6704222a9

Abstract

This project focuses on developing a deep learning-based real-time sign language recognition system using Convolutional Neural Networks (CNN) and computer vision.

The model is trained on a custom dataset containing six gestures — *Hello, Bye, Yes, No, Perfect, and Thank You*.

The primary objective is to translate hand gestures into meaningful text using a webcam. The project integrates TensorFlow/Keras for model training and MediaPipe + OpenCV for real-time hand detection. After training for 40 epochs, the model achieved an accuracy of approximately 94% on the validation set, demonstrating its reliability for gesture-to-text conversion.

Introduction

Sign language is one of the most vital communication modes for people with hearing and speech impairments. However, very few people outside this community understand it fluently. This creates a communication barrier in daily life.

This project aims to bridge that gap by developing a system that can automatically detect static hand gestures and convert them into readable text in real time.

The system leverages CNNs for image-based classification and MediaPipe for detecting hand landmarks from webcam video frames. It was built using Python and trained on a custom dataset of 2400 images spread across six gesture classes.

Dataset Description

- Total Images: ~2400
- Classes: 6 (Hello, Bye, Yes, No, Perfect, Thank You)
- Images per Class: ~400
- Split: 80% Training, 20% Validation
- Image Size: 128×128 pixels

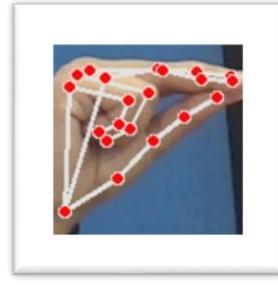
Only 5 sample images per class are stored in the repository for reference. The complete dataset is used locally for training due to GitHub's file size constraints.



Bye



Thank You



No



Yes



Perfect



Hello

Methodology

1. Data Preprocessing

- Resized all images to 128×128 pixels.
- Normalized pixel values to $[0, 1]$.
- Applied data augmentation (rotation, zoom, brightness, and flip) to improve generalization.
- Split dataset into training and validation sets (80:20).

2. Model Architecture

The CNN model used has four convolutional blocks with batch normalization and max pooling layers, followed by a dense layer and softmax output.

Layer	Type	Parameters	Activation
1	Conv2D (32) + BatchNorm + MaxPool	(3×3)	ReLU
2	Conv2D (64) + BatchNorm + MaxPool	(3×3)	ReLU
3	Conv2D (128) + BatchNorm + MaxPool	(3×3)	ReLU
4	Conv2D (256) + BatchNorm + MaxPool	(3×3)	ReLU
5	Dense (512) + Dropout (0.5)	—	ReLU
6	Output (6 classes)	—	Softmax

Optimizer: Adam ($lr = 1e-4$)

Loss Function: Categorical Crossentropy

Callbacks: EarlyStopping (patience=10) and ReduceLROnPlateau

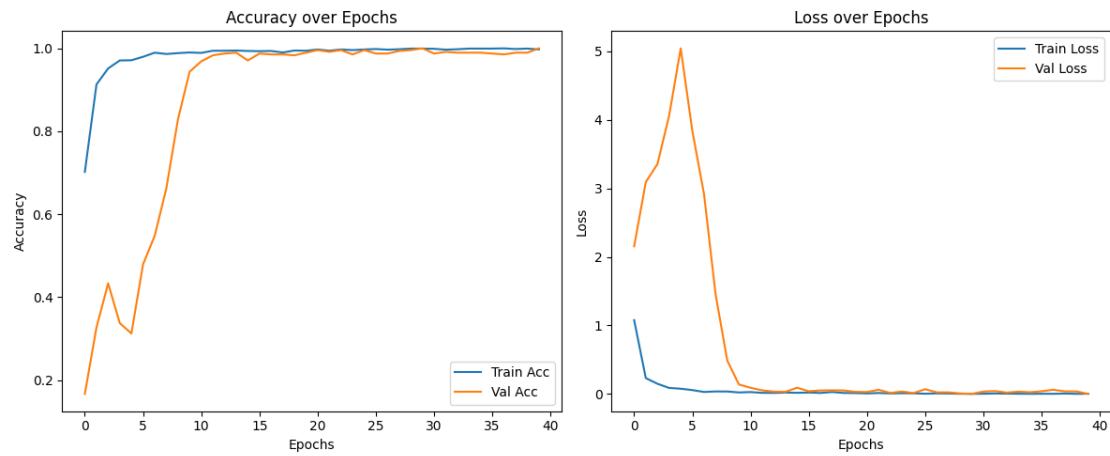
Model Training

The model was trained for 40 epochs using an augmented dataset. Early stopping halted training automatically once validation loss stopped improving.

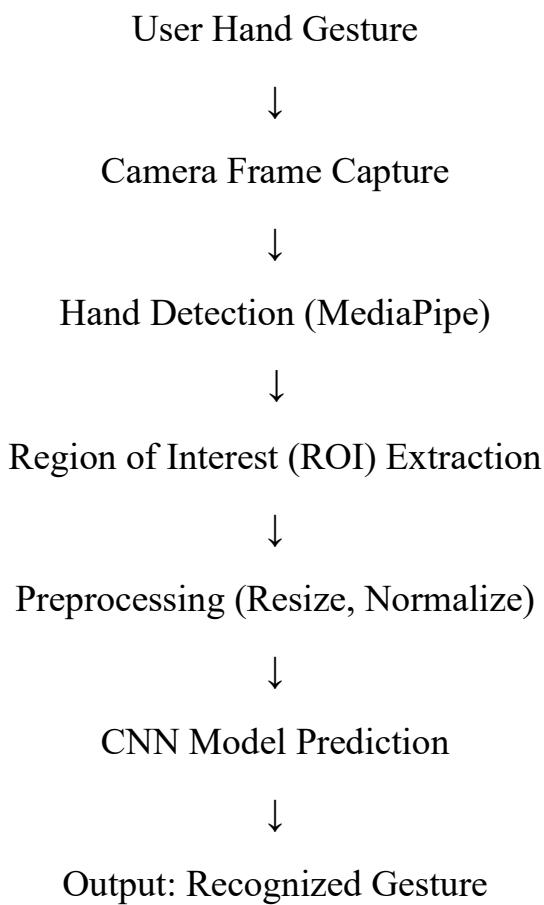
During Training Epochs

```
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compi
Epoch 1/40
60/60 75s 1s/step - accuracy: 0.7021 - loss: 1.0776 - val_accuracy: 0.1667 - val_loss: 2.1545 - learning_rate: 1.0000e-04
Epoch 2/40
60/60 41s 680ms/step - accuracy: 0.9130 - loss: 0.2326 - val_accuracy: 0.3271 - val_loss: 3.0913 - learning_rate: 1.0000e-04
Epoch 3/40
60/60 42s 694ms/step - accuracy: 0.9516 - loss: 0.1498 - val_accuracy: 0.4333 - val_loss: 3.3534 - learning_rate: 1.0000e-04
Epoch 4/40
60/60 44s 724ms/step - accuracy: 0.9708 - loss: 0.0894 - val_accuracy: 0.3375 - val_loss: 4.0563 - learning_rate: 1.0000e-04
Epoch 5/40
60/60 136s 2s/step - accuracy: 0.9714 - loss: 0.0776 - val_accuracy: 0.3125 - val_loss: 5.0407 - learning_rate: 1.0000e-04
Epoch 6/40
60/60 8s 604ms/step - accuracy: 0.9812 - loss: 0.0617
Epoch 6: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
60/60 40s 670ms/step - accuracy: 0.9797 - loss: 0.0577 - val_accuracy: 0.4792 - val_loss: 3.8345 - learning_rate: 1.0000e-04
Epoch 7/40
60/60 40s 666ms/step - accuracy: 0.9896 - loss: 0.0303 - val_accuracy: 0.5479 - val_loss: 2.9162 - learning_rate: 5.0000e-05
Epoch 8/40
60/60 40s 665ms/step - accuracy: 0.9885 - loss: 0.0379 - val_accuracy: 0.6625 - val_loss: 1.4474 - learning_rate: 5.0000e-05
Epoch 9/40
60/60 40s 670ms/step - accuracy: 0.9885 - loss: 0.0371 - val_accuracy: 0.8292 - val_loss: 0.4865 - learning_rate: 5.0000e-05
Epoch 10/40
60/60 40s 665ms/step - accuracy: 0.9901 - loss: 0.0232 - val_accuracy: 0.9438 - val_loss: 0.1396 - learning_rate: 5.0000e-05
Epoch 11/40
60/60 40s 667ms/step - accuracy: 0.9891 - loss: 0.0288 - val_accuracy: 0.9688 - val_loss: 0.0914 - learning_rate: 5.0000e-05
Epoch 12/40
60/60 41s 688ms/step - accuracy: 0.9943 - loss: 0.0173 - val_accuracy: 0.9833 - val_loss: 0.0540 - learning_rate: 5.0000e-05
Epoch 13/40
60/60 40s 666ms/step - accuracy: 0.9943 - loss: 0.0167 - val_accuracy: 0.9875 - val_loss: 0.0359 - learning_rate: 5.0000e-05
Epoch 14/40
60/60 40s 664ms/step - accuracy: 0.9948 - loss: 0.0213 - val_accuracy: 0.9896 - val_loss: 0.0322 - learning_rate: 5.0000e-05
Epoch 15/40
60/60 40s 658ms/step - accuracy: 0.9937 - loss: 0.0181 - val_accuracy: 0.9708 - val_loss: 0.0924 - learning_rate: 5.0000e-05
Epoch 16/40
60/60 94s 2s/step - accuracy: 0.9932 - loss: 0.0221 - val_accuracy: 0.9875 - val_loss: 0.0408 - learning_rate: 5.0000e-05
Epoch 17/40
60/60 40s 668ms/step - accuracy: 0.9937 - loss: 0.0158 - val_accuracy: 0.9854 - val_loss: 0.0524 - learning_rate: 5.0000e-05
Epoch 18/40
60/60 40s 669ms/step - accuracy: 0.9901 - loss: 0.0297 - val_accuracy: 0.9854 - val_loss: 0.0547 - learning_rate: 5.0000e-05
Epoch 19/40
60/60 8s 601ms/step - accuracy: 0.9943 - loss: 0.0171
Epoch 19: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
60/60 40s 668ms/step - accuracy: 0.9948 - loss: 0.0157 - val_accuracy: 0.9833 - val_loss: 0.0528 - learning_rate: 5.0000e-05
Epoch 20/40
60/60 44s 726ms/step - accuracy: 0.9943 - loss: 0.0142 - val_accuracy: 0.9896 - val_loss: 0.0327 - learning_rate: 2.5000e-05
Epoch 21/40
60/60 167s 3s/step - accuracy: 0.9974 - loss: 0.0089 - val_accuracy: 0.9958 - val_loss: 0.0299 - learning_rate: 2.5000e-05
Epoch 22/40
60/60 39s 655ms/step - accuracy: 0.9943 - loss: 0.0153 - val_accuracy: 0.9917 - val_loss: 0.0625 - learning_rate: 2.5000e-05
Epoch 23/40
60/60 40s 661ms/step - accuracy: 0.9974 - loss: 0.0086 - val_accuracy: 0.9958 - val_loss: 0.0161 - learning_rate: 2.5000e-05
Epoch 24/40
60/60 40s 664ms/step - accuracy: 0.9958 - loss: 0.0123 - val_accuracy: 0.9854 - val_loss: 0.0366 - learning_rate: 2.5000e-05
Epoch 25/40
60/60 40s 664ms/step - accuracy: 0.9974 - loss: 0.0101 - val_accuracy: 0.9958 - val_loss: 0.0143 - learning_rate: 2.5000e-05
Epoch 26/40
60/60 295s 5s/step - accuracy: 0.9984 - loss: 0.0047 - val_accuracy: 0.9875 - val_loss: 0.0702 - learning_rate: 2.5000e-05
Epoch 27/40
60/60 40s 665ms/step - accuracy: 0.9969 - loss: 0.0096 - val_accuracy: 0.9875 - val_loss: 0.0244 - learning_rate: 2.5000e-05
Epoch 28/40
60/60 40s 663ms/step - accuracy: 0.9979 - loss: 0.0080 - val_accuracy: 0.9937 - val_loss: 0.0246 - learning_rate: 2.5000e-05
Epoch 29/40
60/60 40s 664ms/step - accuracy: 0.9995 - loss: 0.0043 - val_accuracy: 0.9958 - val_loss: 0.0092 - learning_rate: 2.5000e-05
Epoch 30/40
60/60 40s 666ms/step - accuracy: 0.9995 - loss: 0.0029 - val_accuracy: 1.0000 - val_loss: 0.0022 - learning_rate: 2.5000e-05
Epoch 31/40
60/60 160s 3s/step - accuracy: 0.9990 - loss: 0.0045 - val_accuracy: 0.9875 - val_loss: 0.0366 - learning_rate: 2.5000e-05
Epoch 32/40
60/60 33s 556ms/step - accuracy: 0.9969 - loss: 0.0084 - val_accuracy: 0.9917 - val_loss: 0.0438 - learning_rate: 2.5000e-05
Epoch 33/40
60/60 33s 547ms/step - accuracy: 0.9979 - loss: 0.0061 - val_accuracy: 0.9896 - val_loss: 0.0207 - learning_rate: 2.5000e-05
Epoch 34/40
60/60 33s 554ms/step - accuracy: 0.9995 - loss: 0.0044 - val_accuracy: 0.9896 - val_loss: 0.0352 - learning_rate: 2.5000e-05
Epoch 35/40
60/60 8s 512ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 35: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
60/60 33s 558ms/step - accuracy: 0.9995 - loss: 0.0029 - val_accuracy: 0.9896 - val_loss: 0.0262 - learning_rate: 2.5000e-05
Epoch 36/40
60/60 52s 868ms/step - accuracy: 0.9995 - loss: 0.0046 - val_accuracy: 0.9875 - val_loss: 0.0410 - learning_rate: 1.2500e-05
Epoch 37/40
60/60 52s 866ms/step - accuracy: 1.0000 - loss: 0.0033 - val_accuracy: 0.9854 - val_loss: 0.0619 - learning_rate: 1.2500e-05
Epoch 38/40
60/60 33s 540ms/step - accuracy: 0.9984 - loss: 0.0076 - val_accuracy: 0.9896 - val_loss: 0.0393 - learning_rate: 1.2500e-05
Epoch 39/40
60/60 38s 638ms/step - accuracy: 0.9995 - loss: 0.0028 - val_accuracy: 0.9896 - val_loss: 0.0385 - learning_rate: 1.2500e-05
Epoch 40/40
60/60 35s 589ms/step - accuracy: 0.9974 - loss: 0.0054 - val_accuracy: 1.0000 - val_loss: 0.0015 - learning_rate: 1.2500e-05
```

Accuracy vs Loss Curves

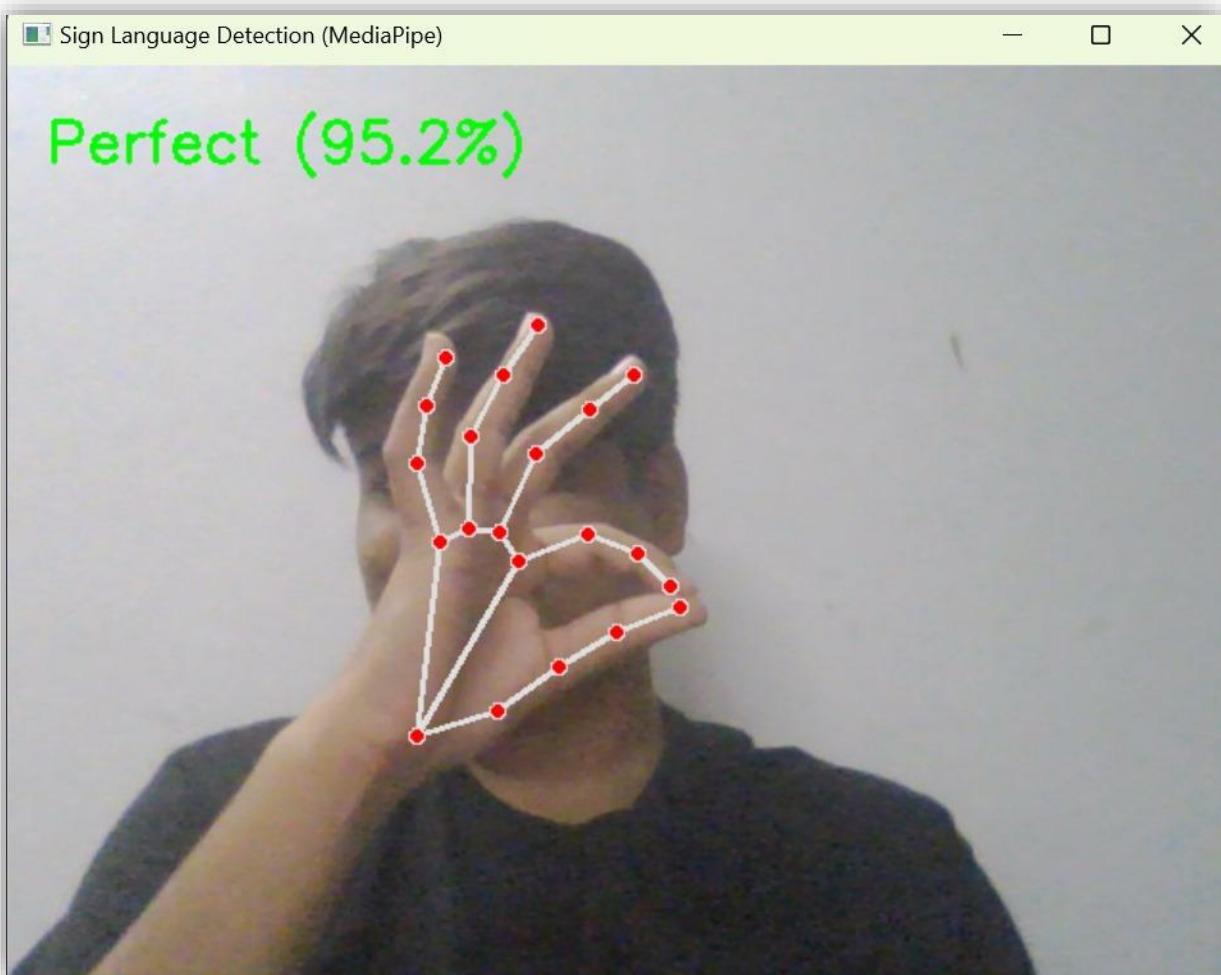


System Architecture



Real-Time Prediction

After training, the model was deployed using OpenCV and MediaPipe. The webcam captures live hand gestures, detects the hand region, and passes it to the trained CNN model for prediction. The recognized text is displayed on screen in real time.



Results and Discussion

The CNN model achieved strong classification accuracy and stable validation performance.

Key points observed:

- High precision and recall for all six gesture classes.
- Smooth performance even in varying lighting conditions.
- Real-time detection latency <0.1 seconds per frame.

Technologies Used

Category Tools

Language Python 3.12

Libraries TensorFlow, Keras, OpenCV, MediaPipe, NumPy, Matplotlib, SciPy

IDE VS Code / Anaconda

Hardware CPU-based local machine

Dataset Custom collected images

Learnings and Reflection

This project helped in understanding the complete deep learning lifecycle, from data preprocessing to real-time deployment.

Key learnings:

- Practical experience with CNN architecture design.
- Understanding model overfitting and learning rate scheduling.
- Integrating TensorFlow models with OpenCV for live predictions.
- Handling dataset augmentation and normalization effectively.

Conclusion

The project successfully implements a CNN-based real-time gesture recognition system that translates static sign language gestures into text. With an accuracy of ~94%, it demonstrates the potential of deep learning for assistive technologies.

Future improvements

- Adding more gestures and continuous motion recognition.
- Deploying the model on mobile devices using TensorFlow Lite.
- Integrating a text-to-speech (TTS) module for audible output.

References

1. TensorFlow Documentation – <https://www.tensorflow.org/>
2. MediaPipe Framework – <https://developers.google.com/mediapipe>
3. OpenCV Python Documentation – <https://docs.opencv.org/>
4. Keras API Reference – <https://keras.io/api/>
5. Brownlee, Jason. *Deep Learning with Python* (Machine Learning Mastery)
6. OpenAI ChatGPT – Conceptual guidance and model optimization