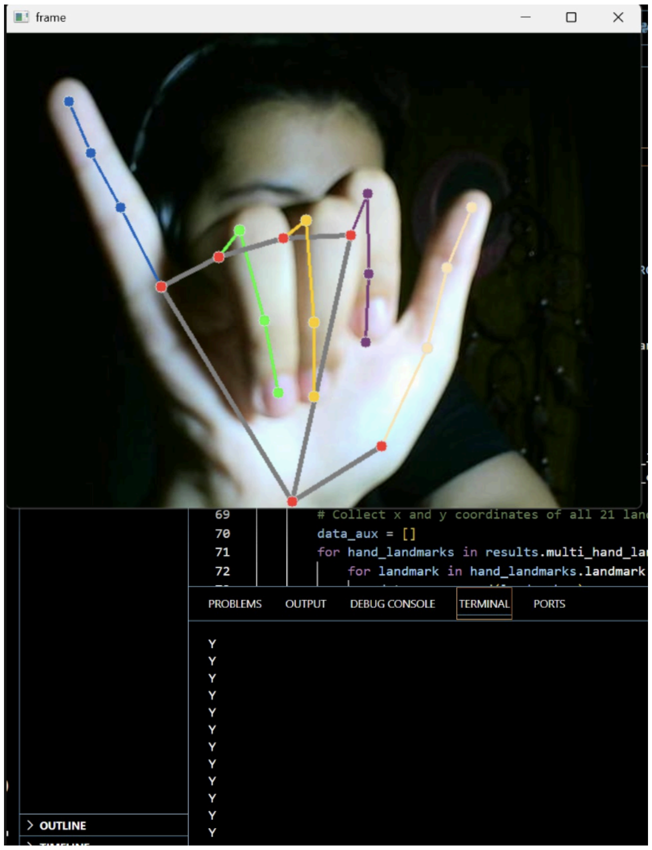
ASL Hand Gesture Classification System Using MediaPipe + Classical Classifier and CNN

Abstract

This project aims to recognize American Sign Language (ASL) hand gestures using computer vision techniques. Two parallel approaches were explored:

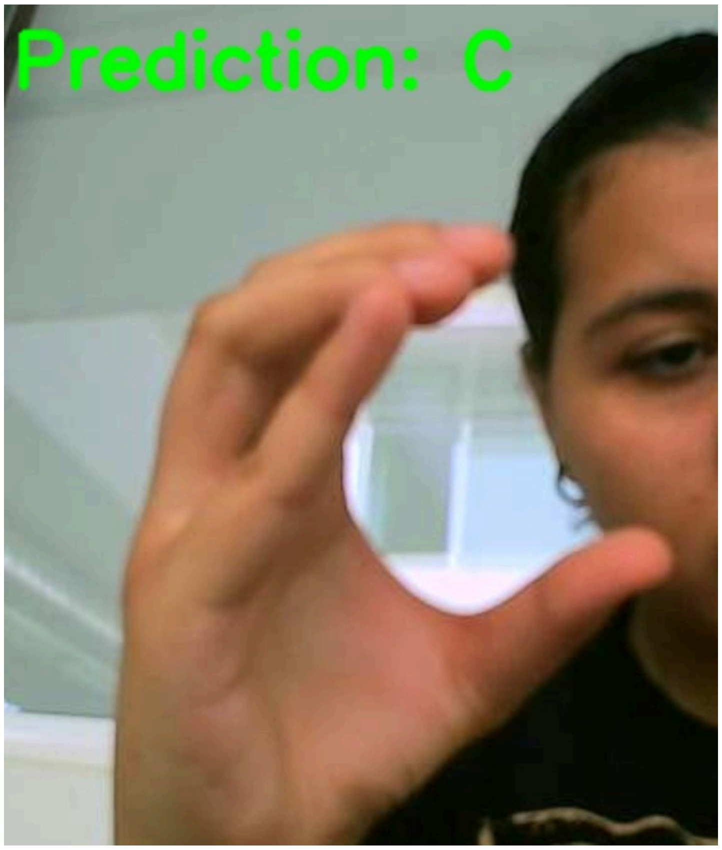
1. A landmark-based pipeline using MediaPipe to extract 2D hand key points and classify them with a traditional machine learning model.
2. A CNN-based pipeline that directly learns features from raw RGB images.

A custom dataset covering 25 ASL classes was collected, and both models were evaluated for accuracy and real-time usability.



```
        # Collect x and y coordinates of all 21 lan
        data_aux = []
        for hand_landmarks in results.multi_hand_la
            for landmark in hand_landmarks.landmark
```

```
Y
Y
Y
Y
Y
Y
Y
Y
Y
Y
Y
Y
```



Introduction

Hand gesture recognition has significant applications in accessibility for the hearing impaired, human-computer interaction, and sign language translation. This project focuses on recognizing static ASL gestures (excluding 'J' and 'Z' due to their dynamic nature) using two distinct approaches:

- The first leverages MediaPipe's hand landmark detection to extract 21 keypoints, which are then used to train a classical classifier.
- The second trains a convolutional neural network (CNN) from scratch using raw RGB images as input.

The goal is to compare their performance and feasibility for static gesture recognition.

Tools and Libraries

- Python 3.9 (in a virtual environment)
- OpenCV
- MediaPipe
- Scikit-learn
- TensorFlow / Keras
- Pickle

Data Collection

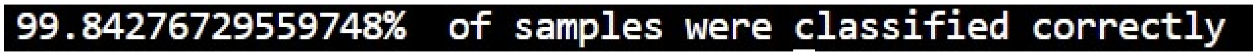
A custom dataset was collected for 25 ASL gesture classes (A–Y excluding J and Z, and adding the digit 7). For each class, 200 images were captured using a webcam, resulting in a dataset of 5,000 labeled RGB images.

Methodology

Approach 1: MediaPipe + Classical Classifier (Random Forest)

- Preprocessing: Each image was converted from BGR to RGB.

- Feature Extraction: MediaPipe extracted 21 hand landmarks per image. Only (x, y) coordinates were retained, producing 42-dimensional vectors.
- Training: A RandomForestClassifier was trained using these vectors.
- Inference: Webcam frames were processed with MediaPipe, features extracted, and classified in real time.



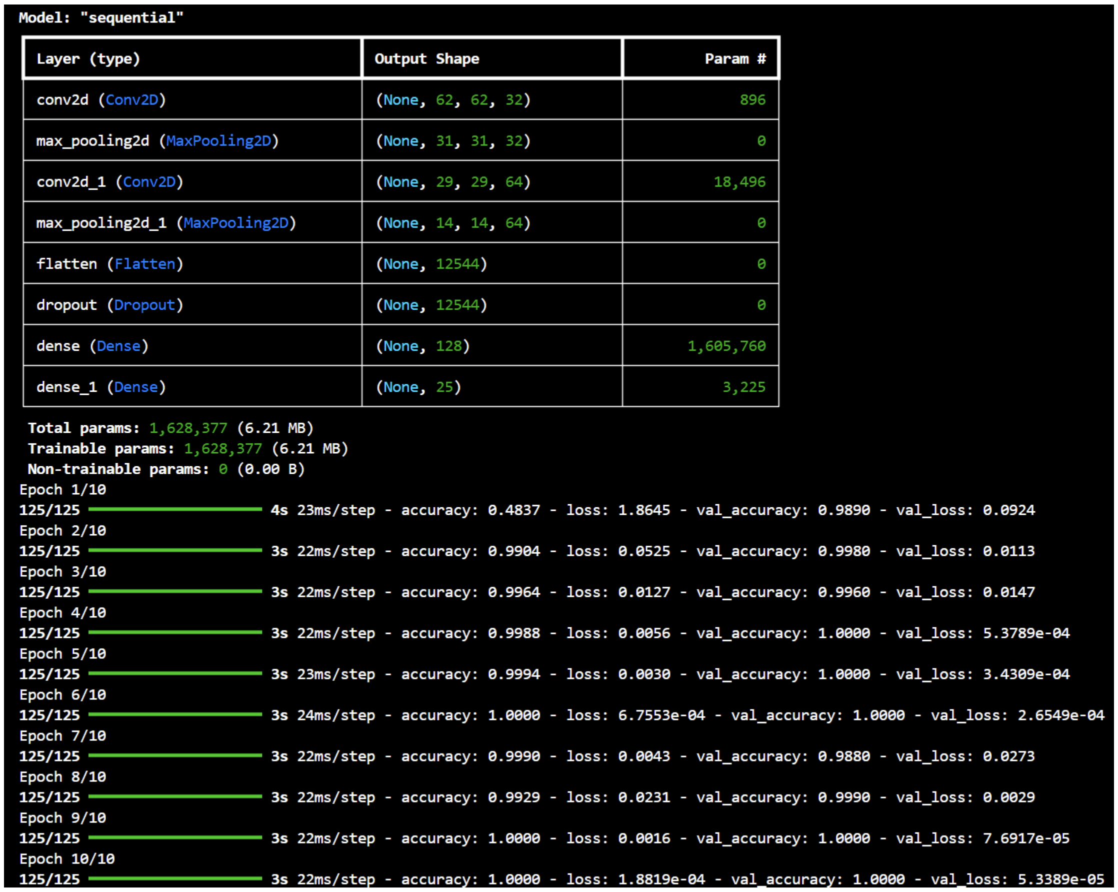
```
99.84276729559748%  of samples were classified correctly
```

Approach 2: CNN on Raw RGB Images

- Preprocessing: All images were resized to 64x64 pixels and normalized to the [0, 1] range with dtype float32.
- Model Architecture:
  - Conv2D (32 filters, 3x3) + ReLU + MaxPooling
  - Conv2D (64 filters, 3x3) + ReLU + MaxPooling
  - Flatten → Dense (128) + ReLU → Dense (25) + Softmax
- Training: The model was trained end-to-end on the labeled image dataset.
- Inference: Webcam frames were resized and fed directly into the CNN for class prediction.

Layer	Output Shape	Description
Conv2D(32, 3x3)	(IMG_SIZE-2, IMG_SIZE-2, 32)	First convolutional layer: detects local patterns like edges. 32 filters of size 3x3
MaxPooling2D(2x2)	Half the width/height	Reduces spatial size to prevent overfitting and computation.
Conv2D(64, 3x3)	More filters, deeper layer	Extracts more complex features. 64 filters =

		more feature maps.
MaxPooling2D(2x2)	Again halves spatial size	Reduces data further before flattening.
Flatten()	Converts 2D → 1D vector	Prepares for dense layers (fully connected).
Dropout(0.5)	Randomly drops 50% neurons	Prevents overfitting by forcing redundancy.
Dense(128)	128 neurons with ReLU	Learns high-level combinations of features.
Dense(NUM_CLASSES)	Softmax classifier	Output layer. One neuron per class (ASL gesture). Gives class probabilities.



```
Model: "sequential"
┌──────────────────────────────────┬─────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape            │      Param # │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d (Conv2D)                  │ (None, 62, 62, 32)      │          896 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 31, 31, 32)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d_1 (Conv2D)                │ (None, 29, 29, 64)      │       18,496 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 14, 14, 64)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ flatten (Flatten)                │ (None, 12544)           │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dropout (Dropout)                │ (None, 12544)           │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dense (Dense)                    │ (None, 128)             │    1,605,760 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dense_1 (Dense)                  │ (None, 25)              │        3,225 │
└──────────────────────────────────┴─────────────────────────┴──────────────┘
 Total params: 1,628,377 (6.21 MB)
 Trainable params: 1,628,377 (6.21 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.4837 - loss: 1.8645 - val_accuracy: 0.9890 - val_loss: 0.0924
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9904 - loss: 0.0525 - val_accuracy: 0.9980 - val_loss: 0.0113
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9964 - loss: 0.0127 - val_accuracy: 0.9960 - val_loss: 0.0147
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9988 - loss: 0.0056 - val_accuracy: 1.0000 - val_loss: 5.3789e-04
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9994 - loss: 0.0030 - val_accuracy: 1.0000 - val_loss: 3.4309e-04
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 24ms/step - accuracy: 1.0000 - loss: 6.7553e-04 - val_accuracy: 1.0000 - val_loss: 2.6549e-04
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9990 - loss: 0.0043 - val_accuracy: 0.9880 - val_loss: 0.0273
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9929 - loss: 0.0231 - val_accuracy: 0.9990 - val_loss: 0.0029
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 1.0000 - val_loss: 7.6917e-05
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 1.0000 - loss: 1.8819e-04 - val_accuracy: 1.0000 - val_loss: 5.3389e-05
```

Model Summary:

Part	Function
Total Parameters	Depends on image size & filters (small/medium-sized)
Activation	ReLU in hidden layers, Softmax in output
Optimizer	Typically used with optimizer='adam'
Loss Function	categorical_crossentropy (multi-class)
Input Shape	(64, 64, 3) or whatever IMG_SIZE is

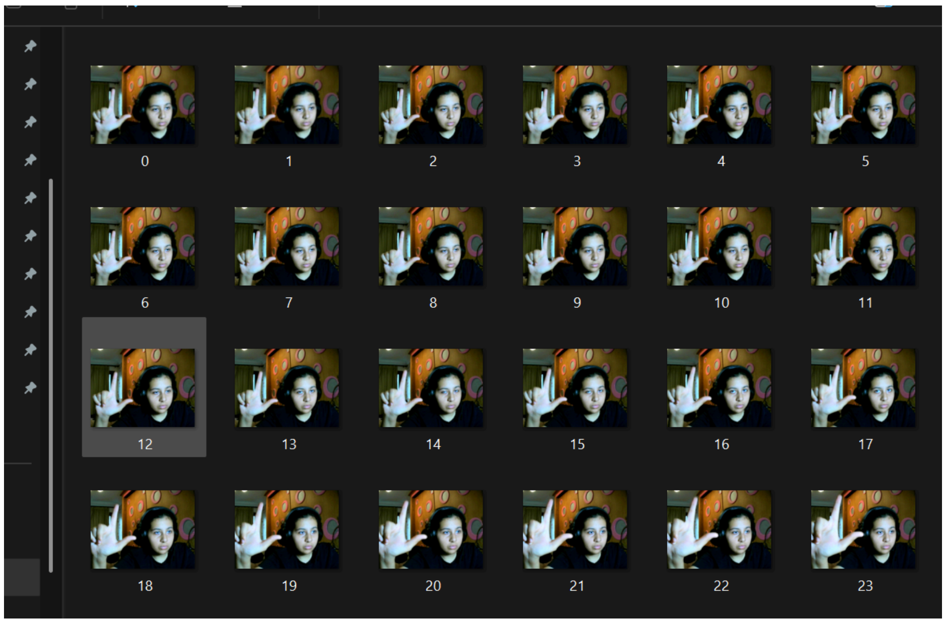
categorical_crossentropy is a loss function used when themodel outputs a probability distribution over multiple classes (via softmax), and your true labels are one-hot encoded.

ReLU, or Rectified Linear Unit, is one of the most widely used activation functions in deep learning, especially in convolutional neural networks (CNNs). It introduces non-linearity into the model by transforming the input using a simple function: it outputs the input value if it is positive, and zero otherwise. This makes it computationally efficient while also helping to mitigate the vanishing gradient problem that affects other activation functions like sigmoid and tanh. By zeroing out negative values, ReLU promotes sparsity in neural activations, which can improve learning efficiency and reduce overfitting. Despite the possibility of some neurons becoming inactive (a problem known as the “dying ReLU”), it remains the default choice in many deep learning architectures due to its simplicity and effectiveness.

Experiments and Results

Dataset Overview

- Total images: 5,000
- Classes: 25
- Split: 80% training, 20% testing



Capturing 200 photos for the dataset of the letter ‘L’

MediaPipe Classifier Results

- Classifier: RandomForestClassifier
- Advantages: Lightweight, fast, minimal data processing

CNN Results

- Architecture: 2 Conv2D layers + MaxPooling + Dense layers
- Advantages: End-to-end learning from raw pixels; no hand-crafted features

Common Observations

- Success cases: Both models performed semi-okay under good lighting and static hand positions unchanged from the dataset much.
- Failure cases:
  - CNN struggled with blurry or cluttered backgrounds and different hand positions.
  - MediaPipe struggled with partial hand visibility or occlusion.
- Confusions: Some signs with similar structures (e.g., M vs N, A vs E) led to misclassification.
- MediaPipe implementation performed better overall.
- The quality of the camera may have had a huge negative impact on the overall accuracy of the models due to its low resolution

Conclusion

This project successfully implemented and compared two ASL gesture classification systems:

- A classical ML pipeline using MediaPipe-extracted hand landmarks
- A deep learning pipeline using a CNN trained on RGB images

The MediaPipe approach was more lightweight and interpretable, while the CNN approach was more flexible and required less manual processing. MediaPipe was more effective in the detection

Future Work

- Add support for dynamic gestures like J and Z using temporal models (e.g., LSTMs)
- Integrate z-coordinates and hand orientation from MediaPipe
- Increase dataset diversity (lighting, skin tone, hand shape)
- Experiment with hybrid models that combine landmarks and raw images

References

(2023, January 23). Sign language detection with Python and Scikit Learn | Landmark detection | Computer vision tutorial [Video]. YouTube. https://www.youtube.com/watch?v=MJCSjXepaAM

Dayalan, K., Saman, D., Murugesan, H., & Prabhakaran, J. (2025). *Sign language translation using random forest classifier*. AIP Conference Proceedings, 3279, 020001. https://doi.org/10.1063/5.0262805

Nemri, M. (2024). *Sign language recognition system using MediaPipe and Random Forest* (Undergraduate thesis, University of Piraeus). University of Piraeus Digital Library. https://dione.lib.unipi.gr/xmlui/handle/unipi/17466