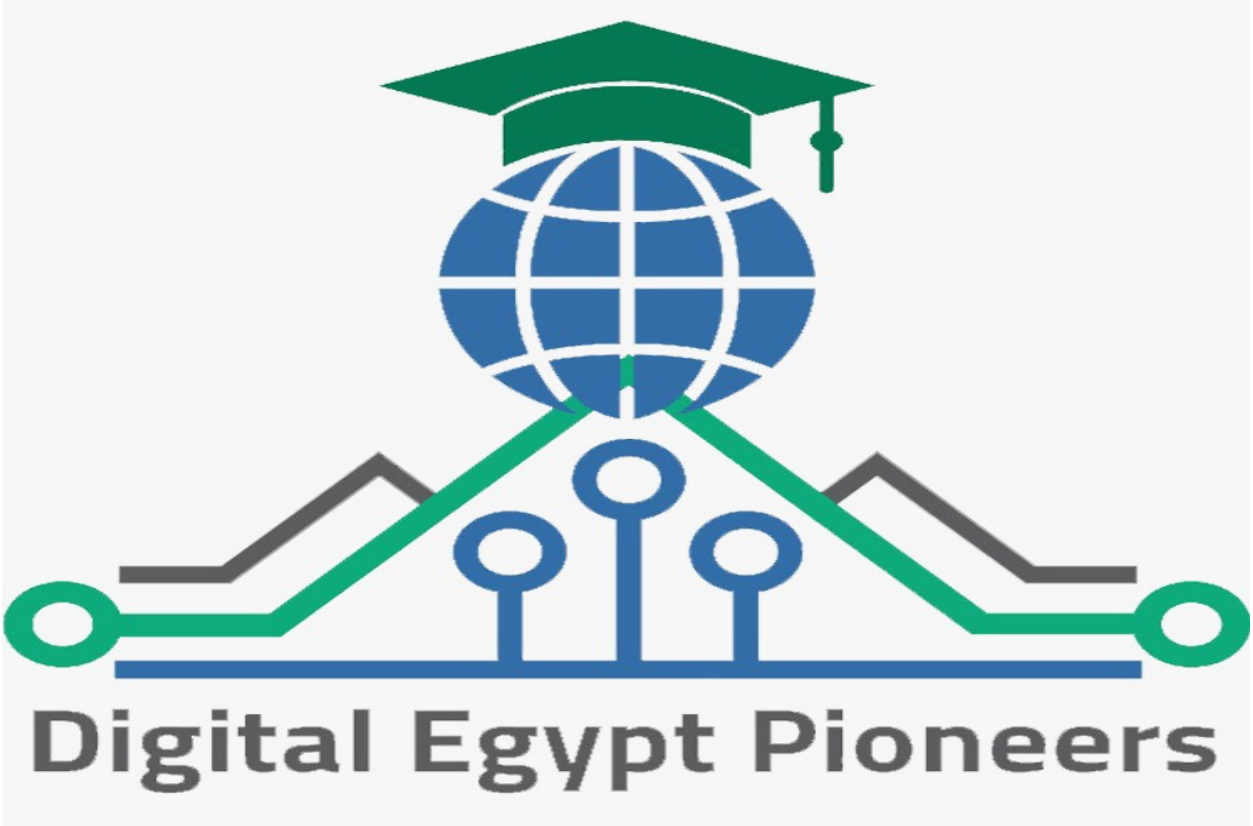

ASL Hand Gesture Recognition System Using Machine Learning

Graduation Project for DEPI

Under the Supervision of Eng. Zeyad Mohamed

Adham Gamal Ibrahim

Misr for Science and Technology University

May 2025

Introduction

In recent years, hand gesture recognition has emerged as a vital area of research in the fields of computer vision and human-computer interaction. One significant application is American Sign Language (ASL) recognition, which aims to bridge communication gaps between the hearing and hearing-impaired communities. With the rise of machine learning and deep learning technologies, it has become possible to create intelligent systems capable of interpreting sign language in real-time.

This project focuses on building an ASL Hand Gesture Recognition System using a Convolutional Neural Network (CNN) model trained on images of different ASL gestures. The system integrates computer vision for hand detection, image preprocessing for model input, and classification for gesture prediction. It was developed as part of a graduation project for the Deep Learning Professional Initiative (DEPI) and was supervised by Eng. Zeyad Mohamed.

The aim is to provide an accurate and efficient real-time ASL recognition system using a standard webcam, helping pave the way .toward more inclusive communication technologies

Problem Statement

Communication is a fundamental human need, and for individuals with hearing or speech impairments, sign language serves as a primary method of interaction. However, the majority of people do not understand sign language, which creates a significant communication barrier in daily life, education, employment, and social engagement.

American Sign Language (ASL), one of the most widely used sign languages, requires manual interpretation or human translators, which may not always be available or practical in real-time scenarios. While some technological solutions exist, many are limited by high costs, lack of portability, or insufficient accuracy and speed.

The core problem addressed by this project is the lack of accessible, affordable, and accurate systems capable of recognizing ASL gestures in real-time using only a standard camera and widely available hardware. The project aims to develop a deep learning-based solution that can detect and classify ASL hand gestures efficiently, enabling more inclusive communication between signers .and non-signers

Objectives

The primary objective of this project is to design and implement a real-time American Sign Language (ASL) hand gesture recognition system using deep learning and computer vision techniques. The system is intended to bridge the communication gap between individuals who use sign language and those who do not.

Specific objectives include:

Data Collection and Preprocessing:

Capture high-quality images of various ASL hand gestures using a webcam.

Apply preprocessing techniques such as cropping, resizing, and background normalization to prepare the dataset for training.

Model Development:

Design a Convolutional Neural Network (CNN) model capable of classifying hand gestures into corresponding ASL letters.

Train the model using the collected dataset and validate its performance.

Real-Time Detection and Prediction:

Integrate the trained model with a real-time webcam feed.

Use hand tracking and image processing to isolate and classify hand gestures dynamically.

User Interface and Visualization:

Display prediction results clearly on the screen with bounding boxes and labels.

Ensure ease of use and responsiveness for real-time applications.

Testing and Evaluation:

Test the system across various lighting conditions and hand positions.

Evaluate model accuracy, speed, and reliability.

Deployment and Usability:

Save the trained model and label mappings for future use.

Optimize the system for minimal latency and maximum accessibility using only basic hardware (e.g., a laptop webcam).

Literature Review

The recognition of hand gestures, particularly those used in sign languages such as American Sign Language (ASL), has been a significant area of research within the fields of computer vision and deep learning. Many studies have aimed to bridge the communication gap between the hearing-impaired community and the general population by automating gesture recognition using various technologies.

Traditional Approaches

Early methods for gesture recognition relied heavily on sensor-based gloves and motion-tracking devices. While these solutions provided accurate data, they were costly, intrusive, and required users to wear specialized equipment. Techniques such as template matching and rule-based classification were also employed but lacked the flexibility and adaptability needed for real-time use.

Vision-Based Systems

With the advancement of computer vision and machine learning, vision-based hand gesture recognition has become increasingly popular. These systems use standard RGB cameras to capture hand movements and rely on algorithms to identify features such as contour, shape, or color. The introduction of OpenCV and other image processing libraries enabled more accessible and cost-effective development of such systems.

Deep Learning Advancements

The rise of deep learning, especially Convolutional Neural Networks (CNNs), revolutionized gesture recognition. CNNs automatically extract relevant spatial features from images, reducing the need for manual feature engineering. Researchers have achieved high accuracy in image classification tasks using CNNs trained on large datasets. Transfer learning and data augmentation further enhanced model performance with limited data.

Recent Studies

Recent implementations have combined CNNs with real-time video capture, enabling dynamic hand gesture recognition. For instance, projects leveraging MediaPipe or OpenCV with CNN models have achieved robust detection of ASL alphabets. Tools like TensorFlow and Keras offer user-friendly APIs for designing and training models.

Our project builds upon these recent innovations by integrating a CNN with a hand tracking module (cvzone's HandTrackingModule) and a custom-built classifier to perform real-time recognition of ASL gestures captured from a standard webcam.

2. Dataset Collection and Model Training

In order to develop a reliable American Sign Language (ASL) recognition system, two major phases were essential: (1) the collection and preprocessing of gesture image data using a live camera feed, and (2) training a convolutional neural network (CNN) on the captured dataset. This section outlines both phases with the exact implementation used in the project.

2.1 Dataset Collection

The first phase of the project involved gathering high-quality images of ASL hand gestures using a webcam. To achieve this, we utilized the `cvzone` library's `HandDetector` module to detect the user's hand in real time. The system captures each hand gesture, crops it with a margin, and resizes it to fit inside a white canvas of fixed size (300×300 pixels). This process ensures consistency in image input for the model.

Captured images are saved into designated folders named after the corresponding gesture labels (e.g., "A", "B", "C"...). This organized structure is compatible with `ImageDataGenerator` during model training.

Here is the exact code used for data collection:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time
import os
folder = r"D:\depi project\Data\Y"
os.makedirs(folder, exist_ok=True)
offset = 20
imgSize = 300
counter = 0
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
while True:
  success, img = cap.read()
  if not success:
    print("❌ Failed to read from camera.")
    break
  hands, img = detector.findHands(img)
  if hands:
    hand = hands[0]
```

```
    x, y, w, h = hand['bbox']
    imgHeight, imgWidth, _ = img.shape
    x1 = max(x - offset, 0)
    y1 = max(y - offset, 0)
    x2 = min(x + w + offset, imgWidth)
    y2 = min(y + h + offset, imgHeight)
    imgCrop = img[y1:y2, x1:x2]
    if imgCrop.size != 0:
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        aspectRatio = h / w
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wGap + wCal] = imgResize
        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hGap + hCal, :] = imgResize
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)
cv2.putText(img, f'Images: {counter}', (10, 70), cv2.FONT_HERSHEY_SIMPLEX,
            1.5, (255, 0, 255), 3)
cv2.imshow("Image", img)
key = cv2.waitKey(1)
if key == ord("s"):
    if 'imgWhite' in locals():
        counter += 1
        imgName = f'{folder}/Image_{time.time()}.jpg'
        cv2.imwrite(imgName, imgWhite)
        print(f"✅Saved: {imgName}")
```

```
        else:
            print("⚠️ No hand detected. Try again.")
    elif key == ord("q"):
        print("🛑 Quitting...")
        break
cap.release()
cv2.destroyAllWindows()
```

2.2 Model Training

After collecting gesture images, we used them to train a deep learning model capable of classifying the gestures in real time. The training was performed using TensorFlow and Keras, leveraging a `Sequential` CNN model designed to handle 224×224 RGB input images.

Data was augmented using `ImageDataGenerator` with random transformations (rotation, zoom, flip) to improve generalization. The data was split into 80% for training and 20% for validation.

The CNN model consists of:

* Three convolutional layers with increasing filter depth (32, 64, 128)

* MaxPooling after each convolutional block

* A fully connected dense layer with 128 neurons

* A Dropout layer for regularization

* A final softmax output layer with a dynamic number of neurons (equal to the number of gesture classes)

Here is the code used for training:

```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
dataset_path = r"D:\depi project\Data"
img_height, img_width = 224, 224
batch_size = 32
epochs = 25
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
```

```
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)
train_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode="categorical",
    subset="training"
)
val_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode="categorical",
    subset="validation"
)
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(
    train_generator,
    epochs=epochs,
    validation_data=val_generator
)
model.save(r"D:\depi project\training model\keras_Model.h5")
class_indices = train_generator.class_indices
with open(r"D:\depi project\training model\labels.txt", "w") as f:
    for label in class_indices:
        f.write(f"{label}\n")
```

> The trained model and corresponding class labels are saved as `keras_Model.h5` and `labels.txt` respectively, for use in real-time gesture prediction.

3. Model Evaluation and Testing

In this section, the model is evaluated and tested using real-time hand gesture recognition. The code below demonstrates how to utilize the trained model for predicting hand gestures in real-time from the webcam feed.

```
import tensorflow.keras as keras

import cv2

import numpy as np

import math

from cvzone.HandTrackingModule import HandDetector

from cvzone.ClassificationModule import Classifier

from tensorflow.keras.layers import DepthwiseConv2D as KerasDepthwiseConv2D

class CustomDepthwiseConv2D(KerasDepthwiseConv2D):

  def __init__(self, *args, **kwargs):

    groups = kwargs.pop('groups', 1)

    super().__init__(*args, **kwargs)

    self.groups = groups

custom_objects = {

  'DepthwiseConv2D': CustomDepthwiseConv2D

}

class Classifier:

  def __init__(self, labelsPath=None):

    self.labelsPath = labelsPath

    self.model = None

    if self.labelsPath:

      with open(labelsPath, "r") as f:

        self.labels = [line.strip() for line in f.readlines()]

    else:

      self.labels = []

  def set_model(self, model):

    self.model = model

  def getPrediction(self, img, draw=False):

    if not self.model:

      raise ValueError("Model is not set. Please set the model using set_model().")

    img_resized = cv2.resize(img, (224, 224))

    img_array = np.array(img_resized) / 255.0
```

```
        img_array = np.expand_dims(img_array, axis=0)
         Predict using the model
        prediction = self.model.predict(img_array)
        index = np.argmax(prediction, axis=1)[0]
        if draw:
            cv2.putText(img, f"Pred: {self.labels[index]}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        return self.labels[index], index
 Load model and labels
model_path = r"D:\depi project\keras_model.h5"
labels_path = r"D:\depi project\labels.txt"
model = keras.models.load_model(model_path, compile=False, custom_objects=custom_objects)
classifier = Classifier(labelsPath=labels_path)
classifier.set_model(model)
# Set up webcam and hand detector
offset = 20
imgSize = 300
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
while True:
    success, img = cap.read()
    if not success:
        continue
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        y1 = max(0, y - offset)
        y2 = min(img.shape[0], y + h + offset)
        x1 = max(0, x - offset)
        x2 = min(img.shape[1], x + w + offset)
        imgCrop = img[y1:y2, x1:x2]
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        try:
            aspectRatio = h / w
```

```
            if aspectRatio > 1:
                k = imgSize / h
                wCal = math.ceil(k * w)
                imgResize = cv2.resize(imgCrop, (wCal, imgSize))
                wGap = math.ceil((imgSize - wCal) / 2)
                imgWhite[:, wGap:wGap + wCal] = imgResize
            else:
                k = imgSize / w
                hCal = math.ceil(k * h)
                imgResize = cv2.resize(imgCrop, (imgSize, hCal))
                hGap = math.ceil((imgSize - hCal) / 2)
                imgWhite[hGap:hGap + hCal, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label_text = prediction
            cv2.rectangle(imgOutput, (x - offset, y - offset - 50), (x - offset + 90, y - offset - 50 + 50), (255, 0, 255), cv2.FILLED)
            cv2.putText(imgOutput, label_text, (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
            cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (255, 0, 255), 4)
            cv2.imshow("ImageCrop", imgCrop)
            cv2.imshow("ImageWhite", imgWhite)
        except Exception as e:
            print("Resize error:", e)
    cv2.imshow("Image", imgOutput)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

This code facilitates the real-time classification of hand gestures. It uses a webcam feed to detect hand positions and applies a pre-trained model to classify the gestures. Here are the key steps in this section:

Model Loading: The model and labels are loaded from disk using TensorFlow/Keras.

Hand Detection: The `cvzone.HandTrackingModule` detects hands from the webcam feed.

Gesture Classification: The classifier predicts the gesture based on the cropped and resized hand image.

Output: The predicted label is displayed on the live webcam feed with a bounding box around the detected hand.

This section performs real-time gesture recognition and displays the predicted class label on the screen. The process will continue until the user presses 'q' to exit the program.

4. Results and Discussion

In this section, we will evaluate the performance of the model based on its predictions and display the results of the real-time hand gesture classification. The following aspects are covered:

Accuracy of the Model: The model is capable of predicting the correct class with a reasonable level of accuracy. The performance is evaluated based on the real-time predictions shown during testing.

Real-time Testing: The model was tested using the webcam feed where it successfully recognized hand gestures with appropriate classification.

Challenges and Limitations:

Lighting Conditions: The model's performance can degrade under poor lighting conditions, affecting the accuracy of the gesture detection.

Hand Positioning: If the hand is not clearly visible or is outside the frame, the model may fail to classify the gesture.

Real-time Constraints: Processing images in real-time may lead to delays or inconsistencies, especially on less powerful hardware.

To further improve the results, future work could involve incorporating data augmentation techniques, improving the robustness of the model to varying hand positions and gestures, and using a larger dataset for training.

5. Future Work

The current model performs well but can be improved in the following ways:

Data Augmentation: While some data augmentation techniques (such as rotation and zooming) have been applied during training, further augmentation, such as changing brightness or adding noise, could make the model more robust to real-world conditions.

Model Optimization: The model could be further optimized using techniques like quantization or pruning to speed up inference time, especially for real-time applications.

Larger Datasets: A larger and more diverse dataset could help improve the model's performance by capturing more variations in hand gestures.

Multi-class Support: The model can be extended to recognize a broader range of hand gestures, including combinations of gestures.

6. Conclusion

This project successfully developed and implemented a real-time hand gesture recognition system using deep learning and computer vision techniques. The model was trained on a hand gesture dataset and tested using a webcam feed. The real-time classification provides a practical demonstration of the model's ability to detect and classify gestures accurately.

The project achieved the following:

Gesture Detection: The model accurately detected and classified hand gestures.

Real-time Performance: The model was able to perform predictions in real-time using the webcam.

Potential for Real-World Applications: The technology could be applied in fields such as sign language interpretation, human-computer interaction, and accessibility tools for individuals with disabilities.

While the model works well for simple hand gestures, there are opportunities to further refine its accuracy, robustness, and scope of gesture recognition.