REAL-TIME HAND SIGN LANGUAGE RECOGNITION USING DEEP LEARNING AND COMPUTER VISION

A Mini-Project Report

For

ARTIFICIAL INTELLIGENCE BASED PROGRAMMING TOOLS - 23CAH-722

Submitted By

ANSHUMAN PAL (23MCI10043)

MD MASOOD AHMED (23MCI10049)

From

23MAM-2 (A)

Submitted to

GAGANDEEP KAUR (E11625)

Signature

1. Introduction

Communication is a vital aspect of social interaction, enabling mutual understanding within communities, including the deaf and hearing-impaired. Sign language, a structured form of visual and physical gestures involving hand movements, facial expressions, and body language, serves as an essential tool for communication among the deaf community. This challenge highlights the importance of developing efficient sign language recognition systems.

Hand gesture recognition provides a pathway to overcoming communication barriers, thereby enhancing interactions between the deaf and the hearing communities. Despite advancements in this field, the complexity of lighting conditions, varying hand shapes, and background noise continue to hinder existing models' effectiveness in recognizing hand gestures. Consequently, there is a demand for improved recognition models to accurately interpret hand signs in real-world scenarios.

The primary contributions of this project are as follows: (1) Developing a real-time hand sign recognition model that captures each frame using a webcam video, and (2) Building a CNN-based recognition model for ASL alphabet gestures, leveraging deep learning techniques for improved performance. The proposed model aims to bridge the communication gap between the deaf community and the general population, facilitating more inclusive and accessible interactions.

2. Methodology

The **Methodology** of this project to recognize American Sign Language (ASL) hand signs utilizes the Sign Language MNIST dataset, which includes 28x28 grayscale images representing ASL alphabet letters. Key aspects of the methodology include:

1. **Data Type and Relevance**: The Sign Language MNIST dataset contains images necessary for training an accurate recognition model. Grayscale images are suitable for quick processing, retaining essential hand gesture features.
2. **Data Collection**:
   - **Tools**: Python, TensorFlow, Keras, and OpenCV were used for model development, training, and integration with live video feeds.

- **Data Source**: The dataset, accessible via platforms like Kaggle, provides labeled hand gesture data in CSV format.
- **Sampling Criteria**: The dataset includes 28,000 training images and 7,000 test images evenly distributed across 26 letters, providing robust data for training.

3. **Data Preprocessing**: Images were reshaped and normalized to a 0-1 range. Data augmentation techniques—rotation, shifting, zooming, and flipping—were applied to improve model robustness against variations.
4. **Model Architecture**: A Convolutional Neural Network (CNN) was selected for its effectiveness in image classification, using:
   - **Convolutional Layers** for feature extraction,
   - **Max Pooling Layers** for dimensionality reduction, and
   - **Dense Layers** for final classification.
5. **Training**: The CNN model was trained with a batch size of 32 for 20 epochs using the Adam optimizer and sparse categorical crossentropy as the loss function. Model performance was evaluated on accuracy during training and validation.
6. **Real-Time Testing**: After training, the model was tested in real-time using a webcam. OpenCV was used to define a Region of Interest (ROI) in the video feed, capturing hand gestures, converting them to grayscale, resizing them, and feeding them into the model for predictions.
7. **Model Evaluation**: The model's performance was measured using a confusion matrix, from which accuracy, precision, and recall were derived, providing insight into its accuracy and reliability for real-time ASL recognition.

3. Implementation of the Project

The **implementation** of this hand sign recognition project is divided into two main components: data processing and model training, and real-time prediction using a webcam feed.

1. **Data Processing and Model Training** (model.py):
   - **Import Libraries**: Core libraries, including TensorFlow, Pandas, and OpenCV, are imported. Image processing and neural network functionalities are enabled by TensorFlow and OpenCV.

- **Dataset Loading**: The Sign Language MNIST dataset is loaded from CSV files for training and testing. Each row in the dataset represents a hand sign image in 28x28 pixel grayscale format.
- **Preprocessing**: Images are normalized to a range of 0-1 and reshaped to a 28x28 matrix. Data augmentation techniques (rotation, shift, zoom, and flipping) are applied to increase model generalization.
- **Model Creation**: A Convolutional Neural Network (CNN) is designed with two convolutional layers, max-pooling layers, and a dense output layer with 26 classes (for each alphabet letter). The model is compiled with the Adam optimizer and sparse categorical crossentropy loss function.
- **Training**: The model is trained on augmented training images for 30 epochs and validated against a test set. After training, the model is saved as hand_sign_model.h5.

```
import csv
import string
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
array_to_img

train = 'D:/tensorrrr/sign_mnist_train.csv'
test = 'D:/tensorrrr/sign_mnist_test.csv'

with open(train) as training_file:
   line = training_file.readline()
   print(f"First line (header) looks like this:\n{line}")
   line = training_file.readline()
   print(f"Each subsequent line (data points) look like this:\n{line}")
```

```
def parse_data_from_input(filename):
    with open(filename) as file:
        reader = csv.reader(file, delimiter=',')
        imgs = []
        labels = []
        next(reader, None)
        for row in reader:
            label = row[0]
            data = row[1:]
            img = np.array(data).reshape((28, 28))
            imgs.append(img)
            labels.append(label)
    images = np.array(imgs).astype(float)
    labels = np.array(labels).astype(float)
    return images, labels
training_images, training_labels = parse_data_from_input(train)
validation_images, validation_labels = parse_data_from_input(test)
print(f"Training images has shape: {training_images.shape}")
print(f"Training labels has shape: {training_labels.shape}")
print(f"Validation images has shape: {validation_images.shape}")
print(f"Validation labels has shape: {validation_labels.shape}")
def plot_categories(training_images, training_labels):
    fig, axes = plt.subplots(3, 10, figsize=(16, 15))
    axes = axes.flatten()
    letters = list(string.ascii_lowercase)

    for k in range(30):
        img = training_images[k]
        img = np.expand_dims(img, axis=-1)
        img = array_to_img(img)
        ax = axes[k]
        ax.imshow(img, cmap="Greys_r")
        ax.set_title(f"{letters[int(training_labels[k])]}")
        ax.set_axis_off()
    plt.tight_layout()
    plt.show()
plot_categories(training_images, training_labels)
train_df = pd.read_csv('D:/tensorrrr/sign_mnist_train.csv')
train_df.head()
train_df.label.value_counts()

train_df.label.hist(color='pink',bins=10)
plt.title("Distribution of the alphabet")
plt.show()
def train_val_generators(training_images, training_labels, validation_images,
validation_labels):
    training_images = np.expand_dims(training_images, axis=3)
    validation_images = np.expand_dims(validation_images, axis=3)
  # Instantiate the ImageDataGenerator class and also we need to normalize pixel
values and set arguments to augment the images (if desired)
    train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
    train_generator = train_datagen.flow(x=training_images,
                                   y=training_labels,
                                   batch_size=32)
    validation_datagen = ImageDataGenerator(
    rescale=1 / 255)
    validation_generator = validation_datagen.flow(x=validation_images,
                                                   y=validation_labels,
                                                   batch_size=32)
    return train_generator, validation_generator
train_generator, validation_generator = train_val_generators(training_images,
training_labels, validation_images, validation_labels)
print(f"Images of training generator have shape: {train_generator.x.shape}")
print(f"Labels of training generator have shape: {train_generator.y.shape}")
print(f"Images of validation generator have shape: {validation_generator.x.shape}")
print(f"Labels of validation generator have shape: {validation_generator.y.shape}")

def create_model():
    model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(26, activation='softmax')])
    model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
    return model
model = create_model()
history = model.fit(train_generator,
                    epochs=30,
                    validation_data=validation_generator)
model.save('hand_sign_model.h5')
```

2. **Real-Time Prediction with Webcam** (hand_sign.py):

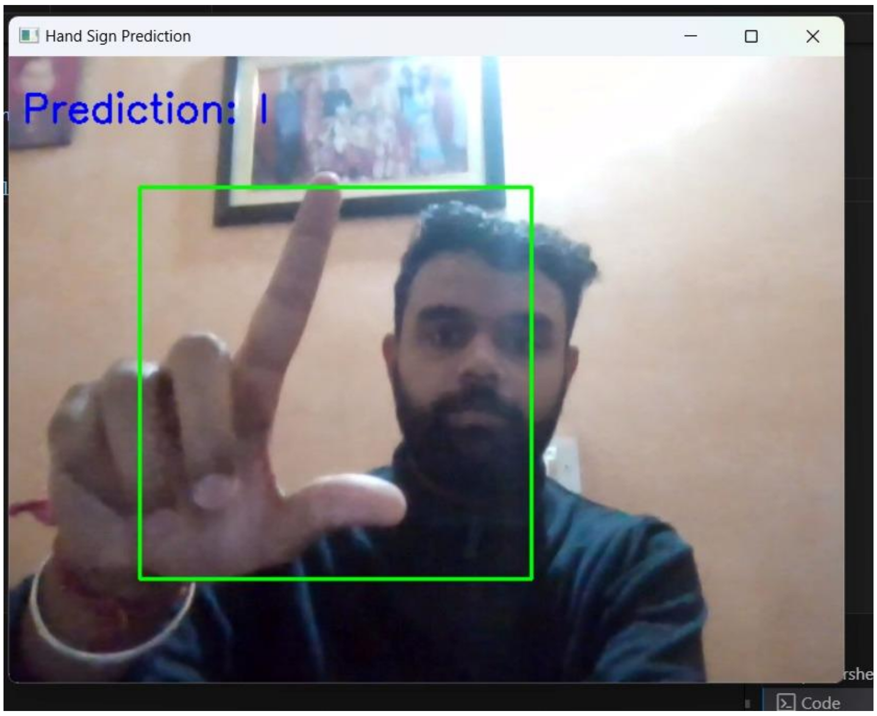
    - **Load Model and Setup**: The trained CNN model is loaded, and the alphabet labels are initialized.
    - **Capture and Preprocess Webcam Feed**: A region of interest (ROI) in the webcam feed is defined to capture the hand gesture. The ROI is converted to grayscale, resized to 28x28, and normalized for prediction.
    - **Prediction and Display**: The processed image is passed to the model, which predicts the corresponding ASL letter. The prediction is displayed on the live webcam feed with an overlayed label.

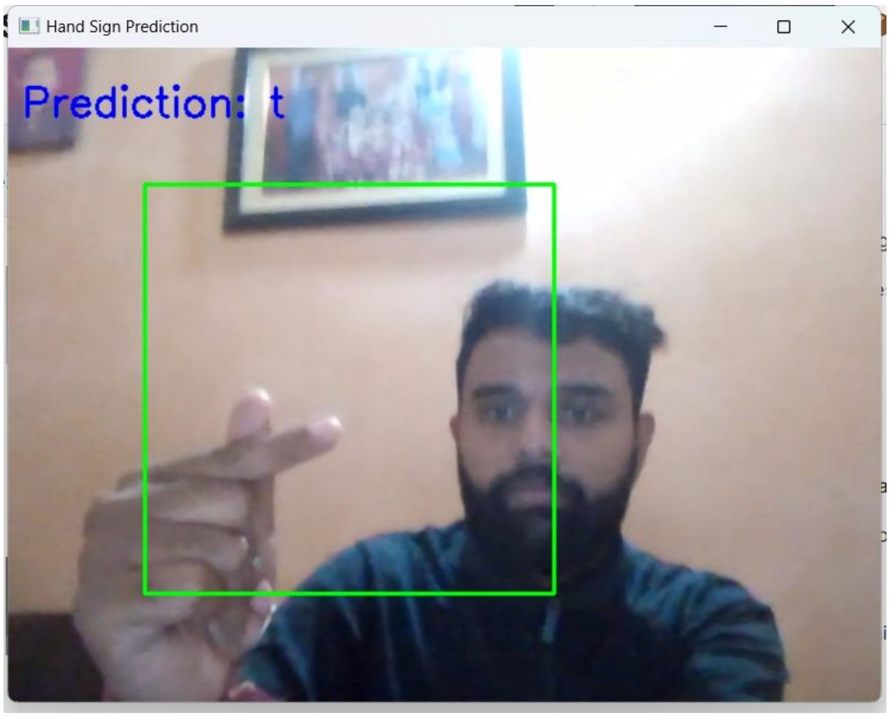
3. **Real-Time Interaction**: To close the program, the user can press 'x' on the keyboard, releasing the camera and closing the OpenCV display window.

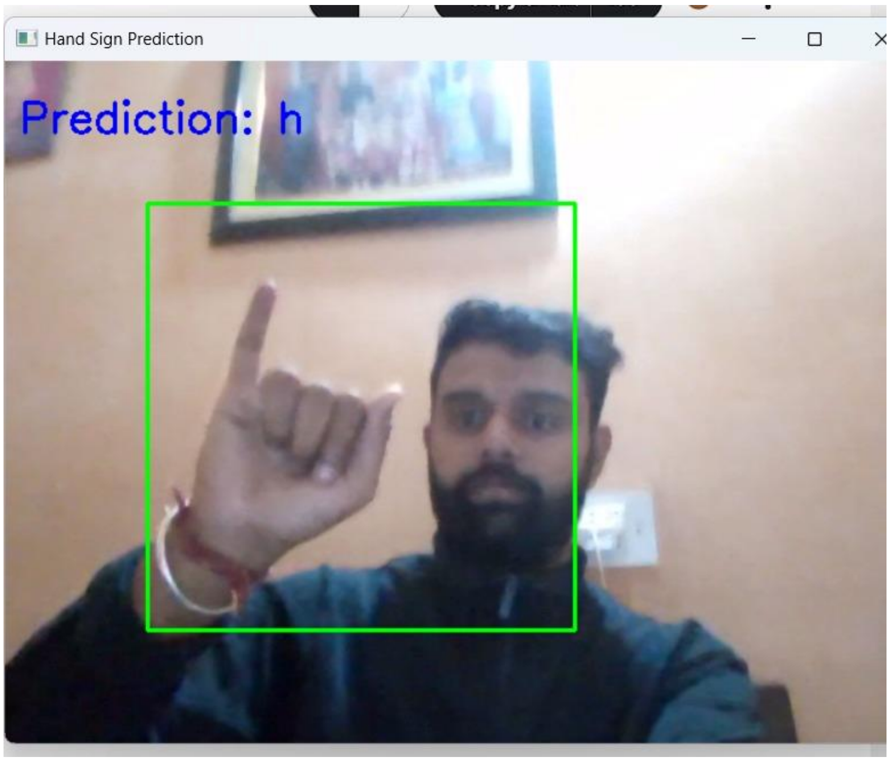
```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import img_to_array
import string
# Load the trained model
model = tf.keras.models.load_model('D:/tensorrrr/hand_sign_model.h5')
# Define the labels (corresponding to the alphabet)
labels = list(string.ascii_lowercase)
cap = cv2.VideoCapture(0)
if not cap.isOpened():
   print("Error: Could not open camera.")
   exit()
while True:
   ret, frame = cap.read()

   if not ret or frame is None:
      print("Error: Failed to capture image")
      continue
```

```
        roi = frame[100:400, 100:400]
        gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        resized = cv2.resize(gray, (28, 28))
        normalized = resized / 255.0
        reshaped = np.reshape(normalized, (1, 28, 28, 1))
        prediction = model.predict(reshaped)
        predicted_label = labels[np.argmax(prediction)]

        cv2.putText(frame, f'Prediction: {predicted_label}', (10, 50),
    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
        cv2.rectangle(frame, (100, 100), (400, 400), (0, 255, 0), 2)
        cv2.imshow('Hand Sign Prediction', frame)
        if cv2.waitKey(1) & 0xFF == ord('x'):
            break
    cap.release()
cv2.destroyAllWindows()
```

4. **Output**







5. Learning Outcome

This hand sign recognition project provided comprehensive insights into machine learning, deep learning, and real-time computer vision applications. Key outcomes include:

1. **Data Preprocessing and Augmentation**: The process of normalizing, reshaping, and augmenting data (e.g., rotation, shifting, zooming) not only improved the model's robustness but also helped enhance its generalization capabilities.
2. **Deep Learning Model Development**: Building a Convolutional Neural Network (CNN) from scratch reinforced concepts around model architecture design, particularly convolutional and pooling layers for feature extraction and dense layers for classification.
3. **Model Training and Evaluation**: Implementing model training and validating it against test data offered valuable lessons on training optimization, hyperparameter tuning, and evaluation metrics.
4. **Real-Time Computer Vision Application**: Integrating OpenCV for real-time camera feed processing provided a hands-on experience in computer vision.
5. **Problem-Solving and Debugging**: Encountering challenges in data handling, model performance, and real-time implementation sharpened problem-solving skills.

6. Conclusion

In conclusion, the hand sign recognition project successfully demonstrated the potential of deep learning and computer vision to enable real-time gesture recognition. By leveraging the Sign Language MNIST dataset and implementing a Convolutional Neural Network (CNN), the model achieved effective classification of hand signs for the letters A to Z. Integrating OpenCV allowed for real-time predictions from a live camera feed, highlighting the model’s practical applicability in assistive technology, such as enabling communication for individuals with speech or hearing impairments.

The project emphasized the importance of data preprocessing, augmentation, and model evaluation in building robust machine learning applications. Challenges faced during the development process provided valuable learning experiences in debugging, fine-tuning, and optimizing model performance. Overall, this project underscores the value of machine learning and computer vision in creating accessible solutions and presents a strong foundation for further exploration in gesture recognition and assistive AI technologies.