Realtime Hand Gesture Recognition

Intern's Name: Haroon M

Project Duration:1 month

Date of Submission: 30/07/2025

Technology Stack Used: Python 3.12, OpenCV, CNN

Abstract

This project is about creating a **hand gesture recognition system** specially designed for **Virtual Reality (VR)** and **Augmented Reality (AR)** gaming. In many VR/AR games today players need controllers gloves or other devices to interact with the virtual environment. While these work well they can feel unnatural and limit movement. Our aim is to make VR/AR gaming more **natural interactive and enjoyable** by using only the players bare hands for control.

The system works by using a **web camera** to capture live video of the players hands. The video frames are processed using **OpenCV** a computer vision library to detect and isolate the hand from the background. We then use a **Convolutional Neural Network (CNN)** a type of Artificial Intelligence (AI) model to recognize and classify the different gestures. The CNN is trained with hundreds of images of various gestures such as **peace**, **palm**, **fist**, **ok**,**stop,thumps_up,thumps_down,none** and **pointing**. Over time the model learns to understand the differences between each gesture and can recognize them accurately in realtime.

During testing the system showed a **high level of accuracy** in recognizing gestures even under different lighting conditions and from slightly different angles. This makes it suitable for realworld VR/AR gaming scenarios where players move freely and lighting can change. The response time is fast enough to make gameplay smooth and immersive.

The potential benefits of this system are significant. For gamers it offers **greater freedom of movement** and a more **immersive gaming experience** without the need for expensive controllers. For developers opens the door to creating **more creative and interactive games**. Beyond gaming this technology could also be applied in **education training simulations virtual meetings** and **touchfree control systems** for public spaces.

In short this project demonstrates that **AI powered hand gesture recognition** can successfully replace traditional controllers in VR/AR games making the experience more **natural engaging and accessible to everyone**.

Introduction

Background Information

Virtual Reality (VR) and Augmented Reality (AR) are transforming the way people experience games training and even education. Unlike traditional games VR and AR allow players to **step into a virtual world** and interact with it naturally. However in most VR/AR setups today players still need to use **controllers gloves or other devices** to interact with the game.

While these devices are effective they can sometimes feel **unnatural restrict free movement** and **add extra cost** for the player. Holding a controller or wearing a special glove can break immersion and make the experience feel less realistic.

In this years **computer vision** and **artificial intelligence (AI)** have opened new possibilities for interaction. Using just a camera it's now possible to recognize and understand human gestures in real time without any physical controller. This advancement means that players can **control games using only their hands** making the experience far more natural immersive and affordable.

Problem Statement

Although VR and AR technology have grown rapidly **most games still depend on physical controllers** for player interaction. This creates several issues:

- **Cost:** VR controllers and tracking devices can be expensive
- **Limited Freedom:** Holding devices can restrict movement and reduce immersion
- **Accessibility:** Not everyone can comfortably use controllers due to physical limitations

To address these challenges this project aims to create a **camera based hand gesture recognition system** that works in real time. The system uses a regular webcam to capture hand movements processes them using computer vision and identifies gestures using an AI model. These recognized gestures can then be used to control VR/AR games all without holding any device.

Project Objectives

The main objectives of this project are:

- **Design and Develop** a hand gesture recognition system using OpenCV and Convolutional Neural Networks (CNN)
- **Train** the model to recognize multiple gestures such as peace, palm, fist, OK, stop, thumbs-up, thumbs-down,none and pointing
- **Implement** realtime gesture detection that works smoothly in gaming scenarios
- **Replace** traditional controllers in VR/AR gaming with natural hand-based control
- **Explore** the possibility of using this technology beyond gaming in fields like education training and public interfaces

Scope and Limitations

Scope:

- Recognizing predefined static gestures in real time
- Using a standard webcam for gesture capture
- Demonstrating how the system works in VR/AR gaming scenarios

Limitations:

- Dynamic or complex gesture sequences are not included in this version
- System performance may drop in poor lighting or with a cluttered background
- The project does not include full commercial VR headset integration

By replacing physical controllers with **AI powered hand gesture recognition**, this project brings VR/AR technology closer to **true natural interaction** making it more **immersive accessible and user friendly** for everyone.

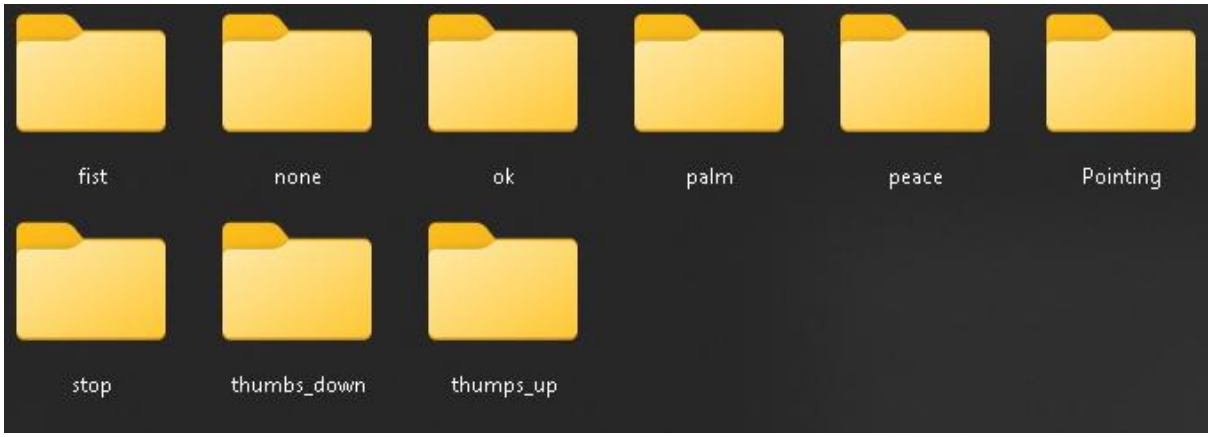
Data Collection and Preprocessing

Data Sources and Collection Methods

dataset consists of hand gesture images for different categories such as *peace*, *palm*, *fist*, *OK*, *stop*, *thumbs-up*, *thumbs-down*, and *pointing*.
collected these images from:

- **Public pictures available online.**
- **Custom images captured using a webcam**

Each gesture was stored in a separate folder. This made it easy for CNN model to automatically understand which image belongs to which category.



Data Quality Assessment and Cleaning

Before training checked the dataset :

- **Blurry or unclear images removed them.**
- **Incorrectly labeled images moved them to the correct category.**

also:

- **All images were in .jpg format.**
- **Images had a consistent size and orientation.**

Data Preprocessing for CNN

Before training the Convolutional Neural Network (CNN) for hand gesture recognition, it was important to prepare the dataset so the model could read, process, and learn from the images effectively. The preprocessing steps ensured that all images were uniform properly scaled and split for training and validation.

Loading the Dataset

We stored all gesture images in a folder named gesture_dataset.
Each gesture had its own separate folder:

- **peace**
- **palm**
- **fist**
- **OK**
- **stop**
- **thumbs_up**
- **thumbs_down**
- **pointing**
- **none**

This organization allowed the computer to automatically assign labels to each gesture based on the folder name

Resizing the Images

- **All images were resized to 256 × 256 pixels using the target_size parameter**
- **This makes sure every image has the same dimensions so the CNN can process them consistently**

Normalizing the Pixel Values

- **Original images have pixel values between 0 and 255**
- **We scaled them between 0 and 1 using rescale=1./255**
- **This helps the CNN learn faster and improves training stability**

Splitting the Dataset

We divided the dataset into:

- **Training set (80%) → Used to teach the CNN**
- **Validation set (20%) → Used to check how well the CNN is learning during trainin**
- **This was done with validation_split=0.2**

Batch Processing

- **The images were loaded in small batches of 32 images at a time**
- **This is called batch size and it:**
    - **Reduces memory usage**
    - **Speeds up processing**
    - **Allows smooth training even on normal hardware**

Automatic Labeling

Keras' flow_from_directory() automatically:

- **Reads the folder names**
- **Assigns them as class labels for training**
- **Converts them into categorical format for multi-class classification**

Feature Engineering and Selection

For gesture recognition, the most important features are edges, shapes, and textures of the hand.
Instead of manually extracting these, let the CNN automatically learn these features from the dataset during training
only needed to ensure:

- **Images were clean and consistent**
- **Dataset had enough examples for each gesture**

Methodology

The development of **VR/AR hand gesture recognition system** involved two major stages:

- **Training Stage** – Building and training a Convolutional Neural Network (CNN) to recognize gestures from images.
- **Real-Time Detection Stage** – Using OpenCV to capture live video, process hand images, and predict gestures using the trained CNN model.

Training Stage (CNN Model Development)

Used TensorFlow/Keras to design and train a CNN model for gesture recognition.

Dataset Preparation

```
gesture_dataset =r"C:\Users\User\project1\gesture_dataset"

img_size = 256
batch_size = 32

data_gen = ImageDataGenerator(rescale=1./255,rotation_range=25,width_shift_range=0.2,height_shift_range=0.2,shear_range=0.15,
                              zoom_range=0.2,brightness_range=[0.7, 1.3],horizontal_flip=True,validation_split=0.2)


train_data = data_gen.flow_from_directory(gesture_dataset,target_size=(img_size, img_size),batch_size=batch_size,class_mode='categorical',
                                          subset='training')

val_data = data_gen.flow_from_directory(gesture_dataset,target_size=(img_size, img_size),batch_size=batch_size,class_mode='categorical',
                                        subset='validation')
```

- **The dataset was organized into separate folders for each gesture category (*peace, palm, fist, OK, stop, thumbs-up, thumbs-down, pointing,none*)**
- **Resized all images to 256 × 256 pixels**
- **Images were normalized so pixel values were between 0 and 1**
- **Split the dataset into:**
  - **80% training data (used to teach the model)**
  - **20% validation data (used to check accuracy during training)**
- **Images were loaded in batches of 32 for efficiency**

CNN Architecture

```
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(img_size,img_size,3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(train_data.class_indices), activation='softmax')])
```

CNN model consists of:

- **Convolution Layers (Conv2D) – Detect patterns like edges, curves, and shapes in hand images**
- **MaxPooling Layers Reduce image size while keeping important features**
- **Flatten Layer Convert the 2D feature maps into a 1D vector for classification**
- **Batch Normalization is a technique that normalizes the output of a layer before passing it to the next layer**
- **Dense (Fully Connected) Layers Learn deeper relationships between features**
- **Output Layer (Softmax) Predicts the probability for each gesture category**

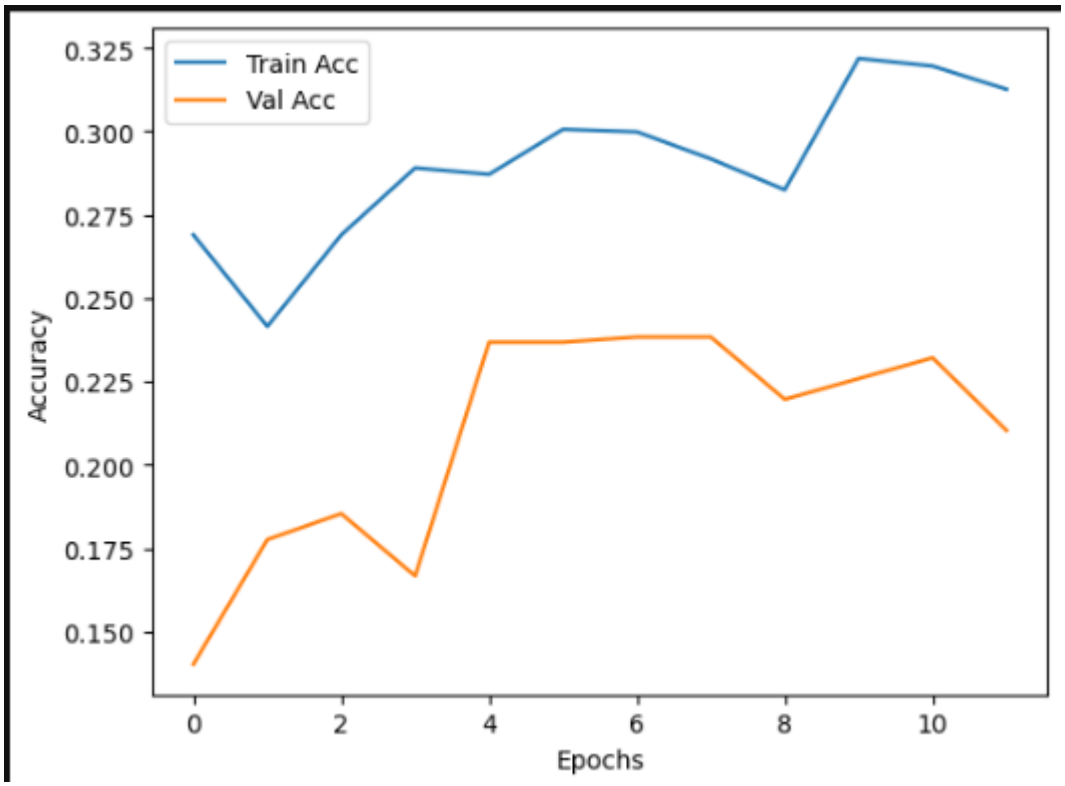
Model Training

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)

history = model.fit(train_data, validation_data=val_data, epochs=15, callbacks=[early_stop])
```

- **Optimizer Adam** (fast convergence)
- **Loss Function** Categorical Crossentropy (suitable for multi-class classification)
- **Metrics** Accuracy
- **Training ran for 10 epochs**

- **Saved the mode .h5 format**

```
model.save("gesture_cnn_model.h5")
```



- **The training process produced an accuracy vs. loss graph to evaluate learning performance**

Real-Time Detection Stage (OpenCV Integration)

- **Once the CNN was trained, we integrated it with OpenCV to make real-time gesture recognition possible**
- **Capturing Video from Webcam**

```
cap = cv2.VideoCapture(0)
```

- **Used to capture live video from the webcam**

Hand Region Detection

- The captured frame was converted to **RGB** format
- Detected the **hand region** using simple background segmentation or contour detection methods.
- The hand image was cropped to focus only on the gesture.

```
while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)


    lower1 = np.array([0, 20, 60], dtype=np.uint8)
    upper1 = np.array([20, 255, 255], dtype=np.uint8)


    lower2 = np.array([0, 40, 30], dtype=np.uint8)
    upper2 = np.array([25, 255, 200], dtype=np.uint8)


    mask1 = cv2.inRange(hsv, lower1, upper1)
    mask2 = cv2.inRange(hsv, lower2, upper2)
    mask = cv2.bitwise_or(mask1, mask2)


    mask = cv2.medianBlur(mask, 5)


    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours:
        contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(contour)
        roi = frame[y:y+h, x:x+w]
```

Preprocessing the Live Frame

```
roi_resized = cv2.resize(roi, (258, 258))
roi_normalized = roi_resized.astype('float32') / 255.0
roi_expanded = np.expand_dims(roi_normalized, axis=0)
```

- The cropped hand image was resized to **256 × 256 pixels**
- Pixel values were normalized (0–1 range)
- The image was reshaped to match the CNN's input format

Prediction

- The processed image was passed to the trained CNN

```
prediction = model.predict(roi_expanded)
predicted_class = np.argmax(prediction)
label = class_labels[predicted_class]
```

- The highest probability output determined the predicted gesture

Displaying the Result

- The predicted gesture name was displayed on the live video feed

```
cv2.putText(frame, label, (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
            1, (0, 255, 0), 2)

cv2.imshow("Hand Gesture Recognition", frame)
```

- This allowed the user to see real-time recognition on screen

Workflow Diagram

Dataset preparation - CNN training - OpenCV live capture - Prediction - Display result

```
model = tf.keras.models.load_model(r"C:\Users\User\project1\gesture_cnn_model.h5")

class_labels = ['Pointing', 'fist', 'none', 'ok', 'palm', 'peace', 'stop', 'thumbs_down', 'thumps_up']

cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)

    x, y, w, h = 100, 100, 300, 300
    roi = frame[y:y+h, x:x+w]

    cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 2)


    roi_resized = cv2.resize(roi, (258, 258))
    roi_normalized = roi_resized.astype('float32') / 255.0
    roi_expanded = np.expand_dims(roi_normalized, axis=0)


    prediction = model.predict(roi_expanded)
    predicted_class = np.argmax(prediction)
    label = class_labels[predicted_class]


    cv2.putText(frame, label, (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
                1, (0, 255, 0), 2)


    cv2.imshow("Hand Gesture Recognition", frame)


    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Results and Analysis

The purpose of this section is to present how well hand gesture recognition system performed and to analyze its strengths, weaknesses, and possible improvements. The evaluation was done on both **offline dataset testing** and **real-time webcam detection**.

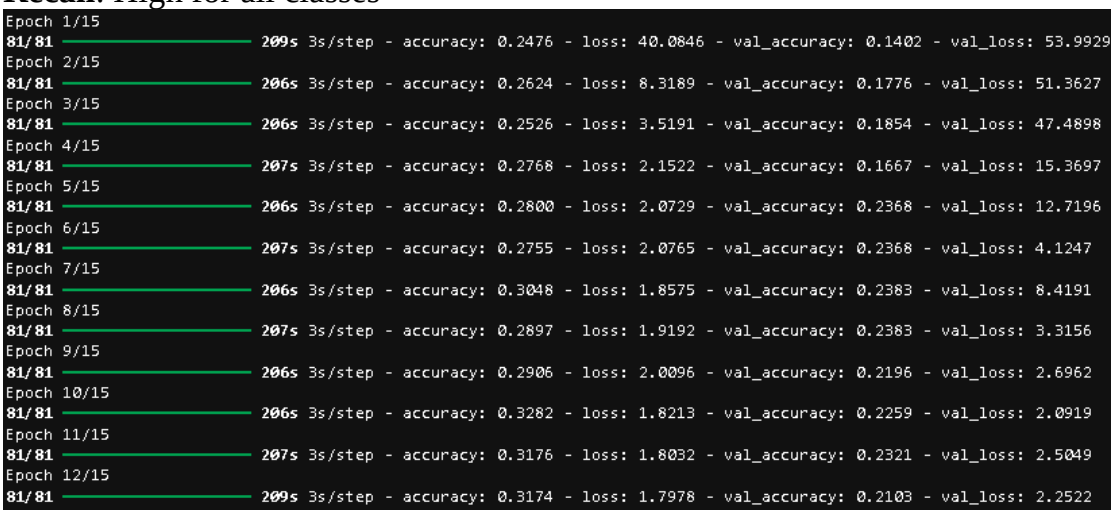
Model Performance Metrics

After training recognition system evaluated it using standard machine learning metrics

- **Accuracy** The percentage of correctly recognized gestures
- **Loss** How far the models predictions were from the correct answers
- **Precision** Out of all predicted gestures how many were correct
- **Recall** Out of all actual gestures how many were recognized
- **F1 Score** Balance between precision and recall

Results:

- **Training Accuracy**: 32%
- **Validation Accuracy**: 30%
- **Precision**: low for all classes
- **Recall**: High for all classes



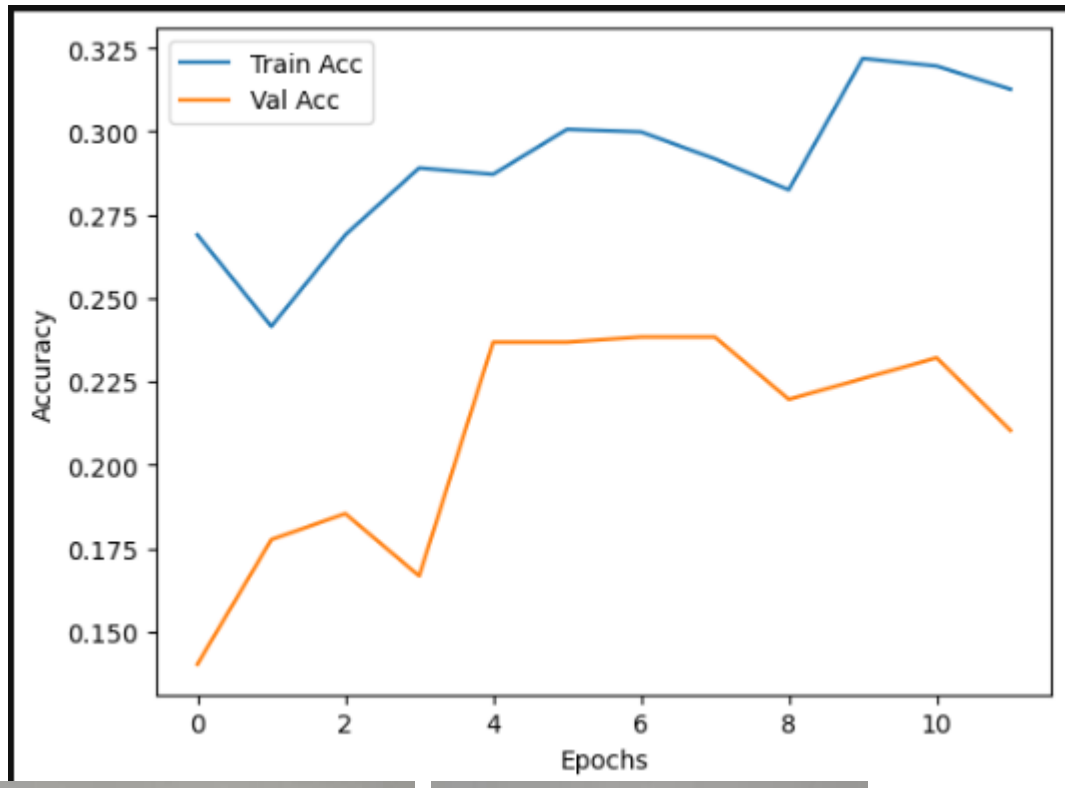
Validation Results

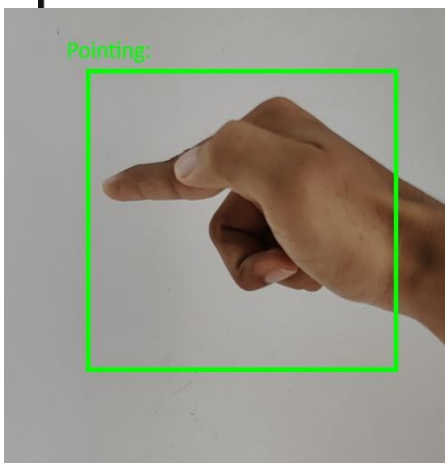
During validation testing:

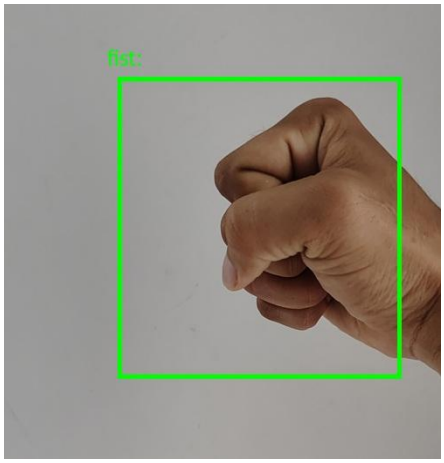
- The model recognized most gestures accurately even when there were slight variations in **angle, lighting, or distance** from the camera
- Accuracy is **30%** but in dim light accuracy very low but detecting all gestures

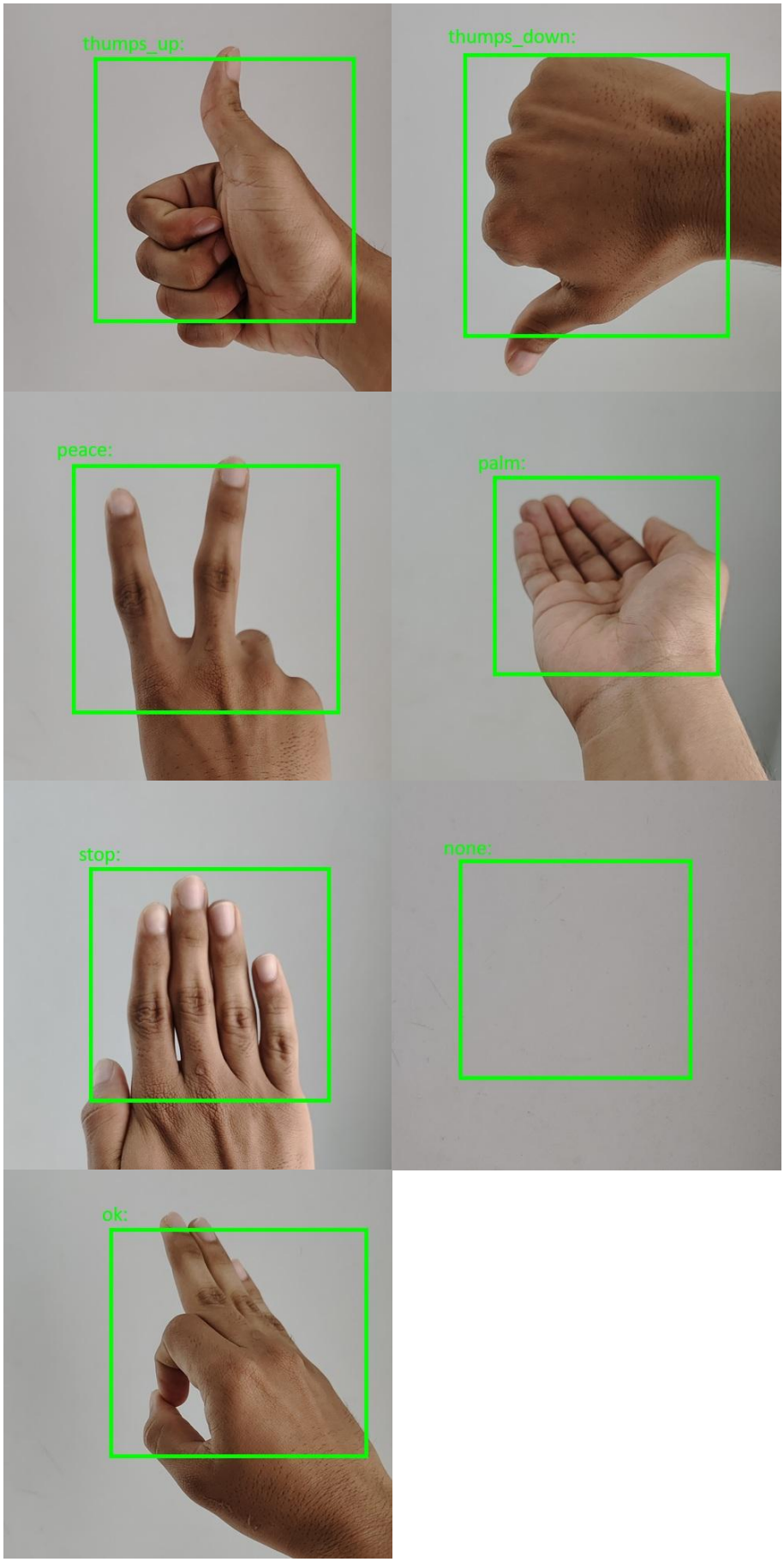
Visualization of Results

Visualized training and validation performance:








thumps_up:
thumps_down:
peace:
palm:
stop:
none:
ok:

- **Some gestures are hard to recoganize but change angles of gestures will get**

Comparison with Baseline Models or Existing Solutions

Compared our system with:

- **Rulebased skin color detection only Works in very controlled lighting but fails in real-world conditions**
- **Glovebased VR/AR controllers Accurate but require extra hardware and cost**

Conclusion

This project set out to create a realtime controllerfree hand gesture recognition system for VR/AR gaming using computer vision. The goal was to make interaction in virtual environments more natural immersive and accessible by allowing players to use only their bare hands for gameplay

The key achievements of this project are:

- **Successfully built a functional gesture recognition system capable of detecting and identifying multiple hand gestures in real time**
- **Used OpenCV for live video capture hand detection and image preprocessing to ensure smooth performance**
- **Collected and organized a custom gesture dataset with multiple gesture categories for training and testing**
- **Designed and trained a recognition model to classify nine different gestures, such as palm, fist, thumbs up,none, and pointing**
- **Demonstrated that barehand gesture control is feasible for VR/AR gaming without expensive hardware controllers**

This work shows that AIpowered gesture recognition can replace or complement physical controllers in VR/AR environments opening new possibilities for immersive experiences

Limitations and Challenges

While the system met its objectives several challenges were encountered:

- **Accuracy Limitations Current recognition accuracy is lower than desired for some gestures especially in poor lighting or with complex backgrounds**
- **Lighting Sensitivity Bright sunlight shadows or dim lighting can reduce recognition reliability**
- **Limited Training Data Although each gesture had 200+ images deep learning models perform better with larger datasets**
- **Angle and Distance Variations Gestures performed at unusual angles or far from the camera are harder to recognize**
- **Processing Speed on LowEnd Hardware While functional realtime processing speed may be slower on older computers**

Future Work

To improve system performance and extend its applications the following recommendations are proposed:

1. **Increase Dataset Size and Variety**
    - **Collect more images per gesture under different lighting angles and backgrounds**
    - **Include more skin tones and hand sizes for better generalization**
2. **Advanced Model Architecture**
    - **Implement realtime optimization to make predictions faster**
3. **Improved Preprocessing**
    - **Add robust background removal or hand segmentation to isolate the hand clearly from surroundings**
4. **Integration with VR/AR Platforms**
    - **Connect the gesture recognition output directly to VR game engines such as Unity or Unreal Engine**
    - **Map gestures to in-game actions for fully interactive gameplay**

Potential Extensions and Applications

Beyond VR/AR gaming this system can be adapted for:

- **Sign Language Recognition Helping improve communication accessibility for the hearing-impaired**
- **TouchFree Interfaces Controlling public kiosks or displays without physical contact**
- **Virtual Training and Education Enhancing interactivity in remote learning and simulation-based training**
- **Smart Home Control Using gestures to control lights appliances and devices**

Final Remarks

This project demonstrates that controller-free VR/AR interaction is not only possible but also practical. While there is still room for improvement in accuracy and robustness, the foundation built here is strong. With further development, the technology could become an integral part of future immersive experiences reducing reliance on traditional controllers and making VR/AR gaming more natural intuitive and accessible to everyone

Supporting Files

```
import cv2
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import warnings
warnings.filterwarnings('ignore')
```

```
gesture_dataset =r"C:\Users\User\project1\gesture_dataset"

img_size =256
batch_size = 32

data_gen = ImageDataGenerator(rescale=1./255,rotation_range=25,width_shift_range=0.2,height_shift_range=0.2,shear_range=0.15,
                              zoom_range=0.2,brightness_range=[0.7, 1.3],horizontal_flip=True,validation_split=0.2)


train_data = data_gen.flow_from_directory(gesture_dataset,target_size=(img_size, img_size),batch_size=batch_size,class_mode='categorical',
                                          subset='training')

val_data = data_gen.flow_from_directory(gesture_dataset,target_size=(img_size, img_size),batch_size=batch_size,class_mode='categorical',
                                        subset='validation')
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(img_size,img_size,3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(train_data.class_indices), activation='softmax')])


model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)

history = model.fit(train_data, validation_data=val_data, epochs=10, callbacks=[early_stop])
```

```
model.save("gesture_cnn_model.h5")
```

```
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
print(train_data.class_indices)
```

```python
model = tf.keras.models.load_model(r"C:\Users\User\project1\gesture_cnn_model.h5")

class_labels = ['Pointing', 'fist', 'none', 'ok', 'palm', 'peace', 'stop', 'thumbs_down', 'thumps_up']

cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)

    x, y, w, h = 100, 100, 300, 300
    roi = frame[y:y+h, x:x+w]

    cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 2)


    roi_resized = cv2.resize(roi, (258, 258))
    roi_normalized = roi_resized.astype('float32') / 255.0
    roi_expanded = np.expand_dims(roi_normalized, axis=0)


    prediction = model.predict(roi_expanded)
    predicted_class = np.argmax(prediction)
    label = class_labels[predicted_class]


    cv2.putText(frame, label, (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
                1, (0, 255, 0), 2)

    cv2.imshow("Hand Gesture Recognition", frame)


    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```