**Hand Gesture Recognition**

Hand gesture recognition involves detecting and interpreting human hand movements and gestures through various techniques, usually using computer vision and machine learning.

**Collecting images**

The first step involves collecting images of different hand gestures to create a dataset.

import os

import cv2

DATA\_DIR = './data'

if not os.path.exists(DATA\_DIR):

    os.makedirs(DATA\_DIR)

number\_of\_classes = 3

dataset\_size = 100

cap = cv2.VideoCapture(0)

for j in range(number\_of\_classes):

    if not os.path.exists(os.path.join(DATA\_DIR, str(j))):

        os.makedirs(os.path.join(DATA\_DIR, str(j)))

    print('Collecting data for class {}'.format(j))

    done = False

    while True:

        ret, frame = cap.read()

        cv2.putText(frame, 'Ready? Press "Q" ! :)', (100, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1.3, (0, 255, 0), 3,

                    cv2.LINE\_AA)

        cv2.imshow('frame', frame)

        if cv2.waitKey(25) == ord('q'):

            break

    counter = 0

    while counter < dataset\_size:

        ret, frame = cap.read()

        cv2.imshow('frame', frame)

        cv2.waitKey(25)

        cv2.imwrite(os.path.join(DATA\_DIR, str(j), '{}.jpg'.format(counter)), frame)

        counter += 1

cap.release()

cv2.destroyAllWindows()

**Creating dataset**

Convert the collected images into a dataset suitable for training.

import os

import pickle

import mediapipe as mp

import cv2

import matplotlib.pyplot as plt

mp\_hands = mp.solutions.hands

mp\_drawing = mp.solutions.drawing\_utils

mp\_drawing\_styles = mp.solutions.drawing\_styles

hands = mp\_hands.Hands(static\_image\_mode=True, min\_detection\_confidence=0.3)

DATA\_DIR = data

data = []

labels = []

for dir\_ in os.listdir(DATA\_DIR):

    for img\_path in os.listdir(os.path.join(DATA\_DIR, dir\_)):

        data\_aux = []

        x\_ = []

        y\_ = []

        img = cv2.imread(os.path.join(DATA\_DIR, dir\_, img\_path))

        img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

        results = hands.process(img\_rgb)

        if results.multi\_hand\_landmarks:

            for hand\_landmarks in results.multi\_hand\_landmarks:

                for i in range(len(hand\_landmarks.landmark)):

                    x = hand\_landmarks.landmark[i].x

                    y = hand\_landmarks.landmark[i].y

                    x\_.append(x)

                    y\_.append(y)

  for i in range(len(hand\_landmarks.landmark)):

                    x = hand\_landmarks.landmark[i].x

                    y = hand\_landmarks.landmark[i].y

                    data\_aux.append(x - min(x\_))

                    data\_aux.append(y - min(y\_))

            data.append(data\_aux)

            labels.append(dir\_)

f = open('data.pickle', 'wb')

pickle.dump({'data': data, 'labels': labels}, f)

f.close()

**Train Classifer**

Train a machine learning classifier on the prepared dataset.

import pickle

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import numpy as np

data\_dict = pickle.load(open('./data.pickle', 'rb'))

data = np.asarray(data\_dict['data'])

labels = np.asarray(data\_dict['labels'])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, shuffle=True, stratify=labels)

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_predict = model.predict(x\_test)

score = accuracy\_score(y\_predict, y\_test)

print('{}% of samples were classified correctly !'.format(score \* 100))

f = open('model.p', 'wb')

pickle.dump({'model': model}, f)

f.close()

**Inference Classifier**

Use the trained model to make predictions on live video input.

import pickle

import cv2

import mediapipe as mp

import numpy as np

model\_dict = pickle.load(open('./model.p', 'rb'))

model = model\_dict['model']

cap = cv2.VideoCapture(0)

mp\_hands = mp.solutions.hands

mp\_drawing = mp.solutions.drawing\_utils

mp\_drawing\_styles = mp.solutions.drawing\_styles

hands = mp\_hands.Hands(static\_image\_mode=True, min\_detection\_confidence=0.3)

labels\_dict = {0: 'A', 1: 'L', 2: 'B'}

while True:

    data\_aux = []

    x\_ = []

    y\_ = []

    ret, frame = cap.read()

    H, W, \_ = frame.shape

    frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    results = hands.process(frame\_rgb)

    if results.multi\_hand\_landmarks:

        for hand\_landmarks in results.multi\_hand\_landmarks:

            mp\_drawing.draw\_landmarks(

                frame,  # image to draw

                hand\_landmarks,  # model output

                mp\_hands.HAND\_CONNECTIONS,  # hand connections

                mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),

                mp\_drawing\_styles.get\_default\_hand\_connections\_style())

        for hand\_landmarks in results.multi\_hand\_landmarks:

            for i in range(len(hand\_landmarks.landmark)):

                x = hand\_landmarks.landmark[i].x

                y = hand\_landmarks.landmark[i].y

                x\_.append(x)

                y\_.append(y)

            for i in range(len(hand\_landmarks.landmark)):

                x = hand\_landmarks.landmark[i].x

                y = hand\_landmarks.landmark[i].y

                data\_aux.append(x - min(x\_))

                data\_aux.append(y - min(y\_))

        x1 = int(min(x\_) \* W) - 10

        y1 = int(min(y\_) \* H) - 10

        x2 = int(max(x\_) \* W) - 10

        y2 = int(max(y\_) \* H) - 10

        prediction = model.predict([np.asarray(data\_aux)])

        predicted\_character = labels\_dict[int(prediction[0])]

        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)

        cv2.putText(frame, predicted\_character, (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 1.3, (0, 0, 0), 3,

                    cv2.LINE\_AA)

    cv2.imshow('frame', frame)

    cv2.waitKey(1)

cap.release()

cv2.destroyAllWindows()