Top of Form

Bottom of Form

|  |  |
| --- | --- |
|  | from flask import Flask,render\_template,request |
|  | # Flask-It is our framework which we are going to use to run/serve our application. |
|  | #request-for accessing file which was uploaded by the user on our application. |
|  | import operator |
|  | import cv2 # opencv library |
|  | import matplotlib.pyplot as plt |
|  | import matplotlib.image as mpimg |
|  | import numpy as np |
|  |  |
|  | from tensorflow.keras.models import load\_model#to load our trained model |
|  | import os |
|  | from werkzeug.utils import secure\_filename |
|  |  |
|  | app = Flask(\_\_name\_\_,template\_folder="templates") # initializing a flask app |
|  | # Loading the model |
|  | model=load\_model('gesture.h5') |
|  | print("Loaded model from disk") |
|  |  |
|  |  |
|  | @app.route('/')# route to display the home page |
|  | def home(): |
|  | return render\_template('home.html')#rendering the home page |
|  |  |
|  |  |
|  | @app.route('/intro') # routes to the intro page |
|  | def intro(): |
|  | return render\_template('intro.html')#rendering the intro page |
|  |  |
|  | @app.route('/image1',methods=['GET','POST'])# routes to the index html |
|  | def image1(): |
|  | return render\_template("launch.html") |
|  |  |
|  |  |
|  | @app.route('/predict',methods=['GET', 'POST'])# route to show the predictions in a web UI |
|  | def launch(): |
|  | if request.method == 'POST': |
|  | print("inside image") |
|  | f = request.files['image'] |
|  |  |
|  | basepath = os.path.dirname(\_\_file\_\_) |
|  | file\_path = os.path.join(basepath, 'uploads', secure\_filename(f.filename)) |
|  | f.save(file\_path) |
|  | print(file\_path) |
|  | cap = cv2.VideoCapture(0) |
|  | while True: |
|  | \_, frame = cap.read() #capturing the video frame values |
|  | # Simulating mirror image |
|  | frame = cv2.flip(frame, 1) |
|  |  |
|  | # Got this from collect-data.py |
|  | # Coordinates of the ROI |
|  | x1 = int(0.5\*frame.shape[1]) |
|  | y1 = 10 |
|  | x2 = frame.shape[1]-10 |
|  | y2 = int(0.5\*frame.shape[1]) |
|  | # Drawing the ROI |
|  | # The increment/decrement by 1 is to compensate for the bounding box |
|  | cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1) |
|  | # Extracting the ROI |
|  | roi = frame[y1:y2, x1:x2] |
|  |  |
|  | # Resizing the ROI so it can be fed to the model for prediction |
|  | roi = cv2.resize(roi, (64, 64)) |
|  | roi = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY) |
|  | \_, test\_image = cv2.threshold(roi, 120, 255, cv2.THRESH\_BINARY) |
|  | cv2.imshow("test", test\_image) |
|  | # Batch of 1 |
|  | result = model.predict(test\_image.reshape(1, 64, 64, 1)) |
|  | prediction = {'ZERO': result[0][0], |
|  | 'ONE': result[0][1], |
|  | 'TWO': result[0][2], |
|  | 'THREE': result[0][3], |
|  | 'FOUR': result[0][4], |
|  | 'FIVE': result[0][5]} |
|  | # Sorting based on top prediction |
|  | prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True) |
|  |  |
|  | # Displaying the predictions |
|  | cv2.putText(frame, prediction[0][0], (10, 120), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1) |
|  | cv2.imshow("Frame", frame) |
|  |  |
|  | #loading an image |
|  | image1=cv2.imread(file\_path) |
|  | if prediction[0][0]=='ONE': |
|  |  |
|  | resized = cv2.resize(image1, (200, 200)) |
|  | cv2.imshow("Fixed Resizing", resized) |
|  | key=cv2.waitKey(3000) |
|  |  |
|  | if (key & 0xFF) == ord("1"): |
|  | cv2.destroyWindow("Fixed Resizing") |
|  |  |
|  | elif prediction[0][0]=='ZERO': |
|  |  |
|  | cv2.rectangle(image1, (480, 170), (650, 420), (0, 0, 255), 2) |
|  | cv2.imshow("Rectangle", image1) |
|  | cv2.waitKey(0) |
|  | key=cv2.waitKey(3000) |
|  | if (key & 0xFF) == ord("0"): |
|  | cv2.destroyWindow("Rectangle") |
|  |  |
|  | elif prediction[0][0]=='TWO': |
|  | (h, w, d) = image1.shape |
|  | center = (w // 2, h // 2) |
|  | M = cv2.getRotationMatrix2D(center, -45, 1.0) |
|  | rotated = cv2.warpAffine(image1, M, (w, h)) |
|  | cv2.imshow("OpenCV Rotation", rotated) |
|  | key=cv2.waitKey(3000) |
|  | if (key & 0xFF) == ord("2"): |
|  | cv2.destroyWindow("OpenCV Rotation") |
|  |  |
|  | elif prediction[0][0]=='THREE': |
|  | blurred = cv2.GaussianBlur(image1, (21, 21), 0) |
|  | cv2.imshow("Blurred", blurred) |
|  | key=cv2.waitKey(3000) |
|  | if (key & 0xFF) == ord("3"): |
|  | cv2.destroyWindow("Blurred") |
|  |  |
|  | elif prediction[0][0]=='FOUR': |
|  |  |
|  | resized = cv2.resize(image1, (400, 400)) |
|  | cv2.imshow("Fixed Resizing", resized) |
|  | key=cv2.waitKey(3000) |
|  | if (key & 0xFF) == ord("4"): |
|  | cv2.destroyWindow("Fixed Resizing") |
|  |  |
|  | elif prediction[0][0]=='FIVE': |
|  | '''(h, w, d) = image1.shape |
|  | center = (w // 2, h // 2) |
|  | M = cv2.getRotationMatrix2D(center, 45, 1.0) |
|  | rotated = cv2.warpAffine(image1, M, (w, h))''' |
|  | gray = cv2.cvtColor(image1, cv2.COLOR\_RGB2GRAY) |
|  | cv2.imshow("OpenCV Gray Scale", gray) |
|  | key=cv2.waitKey(3000) |
|  | if (key & 0xFF) == ord("5"): |
|  | cv2.destroyWindow("OpenCV Gray Scale") |
|  |  |
|  | else: |
|  | continue |
|  |  |
|  |  |
|  | interrupt = cv2.waitKey(10) |
|  | if interrupt & 0xFF == 27: # esc key |
|  | break |
|  |  |
|  |  |
|  | cap.release() |
|  | cv2.destroyAllWindows() |
|  | return render\_template("home.html") |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | # running the app |
|  | app.run(debug=False) |