APPLICATION BUILDING

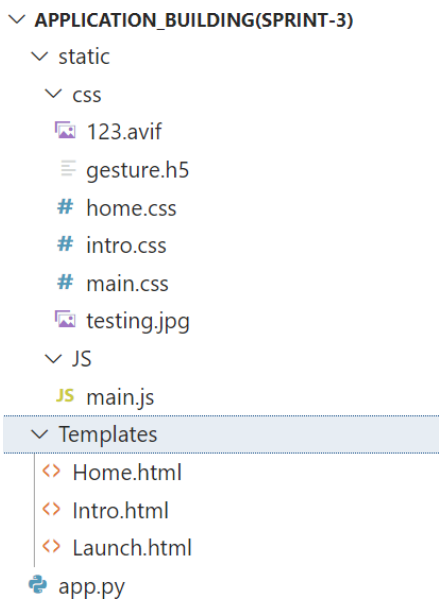
Date	**10 November 2022**
Team Id	**PNT2022TMID50713**
Project Name	**A Gesture-based Tool for Sterile Browsing of Radiology Images**

Application Building

Now after the model is trained in this particular milestone, we will be building our flask application which will be running in our local browser with a user interface.

Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we created 3 html pages- Home.html, Intro.html and Launch.html
- Home.html displays home page.
- Intro.html displays introduction about the hand gesture recognition
- Launch.html accepts input from the user and predicts the values.
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and viewof HTML pages.



Build Python Code

- Let us build flask file 'app.py' which is a web framework written in python for server-sidescripting. Let's see step by step procedure for building the backend application.
- App starts running when "__name__" constructor is called in main.
- Render template is used to return html file.
- "GET" method is used to take input from the user.
- "POST" method is used to display the output to the user.

Importing libraries

```
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
from tensorflow.keras.models import load_model#to load our trained model
import os
from werkzeug.utils import secure_filename
```

Creating our flask application and loading our model

```
app = Flask(__name__,template_folder="templates") # initializing a flask app
# Loading the model
model=load_model('gesture.h5')
print("Loaded model from disk")
```

Routine to the HTML Pages

```
@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("index6.html")
```

The above three route are used to render the home, introduction and the index html pages.

```
@app.route('/predict',methods=['GET', 'POST'])# route to show the predictions in a web UI
def launch():
```

And the predict route is used for prediction and it contains all the codes which are used forpredicting our results.

- Firstly, inside launch function we are having the following things:
    - Getting our input and storing it
    - Grab the frames from the web cam.
    - Creating ROI
    - Predicting our results
    - Showcase the results with the help of opencv
    - Finally run the application

- Getting our input and storing it

  Once the predict route is called, we will check whether the method is POST or not it is POST then we will request the image files and with the help of os function we will be storing the image in the uploads folder in our local system.

```
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)
```

- Grab the frames from the web cam

  Now when we run the code a web cam will be opening to take the gesture input so we will be capturing the frames of the gesture for predicting our results.

```
cap = cv2.VideoCapture(0)
while True:
    _, frame = cap.read() #capturing the video frame values
    # Simulating mirror image
    frame = cv2.flip(frame, 1)
```

- Creating ROI - A region of interest (ROI) is a portion of an image that you want to filter oroperate on in some way. The toolbox supports a set of ROI objects that you can use to create ROIs of many shapes, such circles, ellipses, polygons, rectangles, and hand-drawn shapes A common use of an ROI is to create a binary mask image.
  So, we will be creating a ROI to mask our gesture.

```
# 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)
```

- Predicting our results - After placing the ROI and getting the frames from the web cam now it's time to predict the gesture result using the model which we trained and stored it into a variable for the further operations.

```
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)
```

- Showcase the results with the help of opencv

  Finally according to the result predicted with our model we will be performing certainoperations like resize, blur, rotate etc.

```
#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, (11, 11), 0)
    cv2.imshow("Blurred", blurred)
    key=cv2.waitKey(3000)
    if (key & 0xFF) == ord("3"):
        cv2.destroyWindow("Blurred")
else:
    continue
```

```
        interrupt = cv2.waitKey(10)
        if interrupt & 0xFF == 27: # esc key
            break


    cap.release()
    cv2.destroyAllWindows()
return render_template("home.html")
```

Run the Application

At last, we will run our flask application

```
if __name__ == "__main__":
    # running the app
     app.run(debug=False)
```

Run the app in local browser

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page

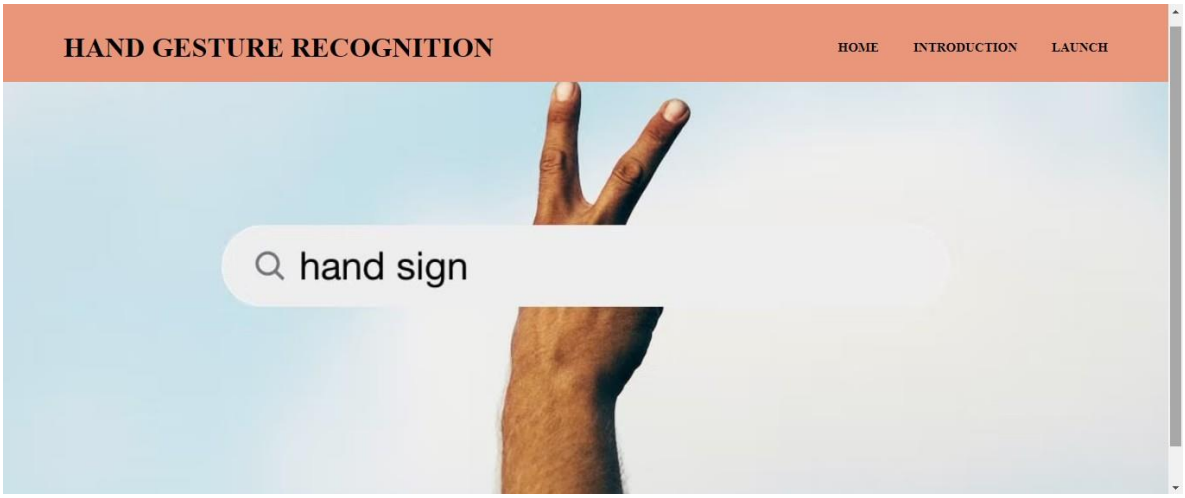
```
(base) E:\>cd E:\PROJECTS\number-sign-recognition\Flask

(base) E:\PROJECTS\number-sign-recognition\Flask>python app.py
```

Then it will run on localhost: 5000

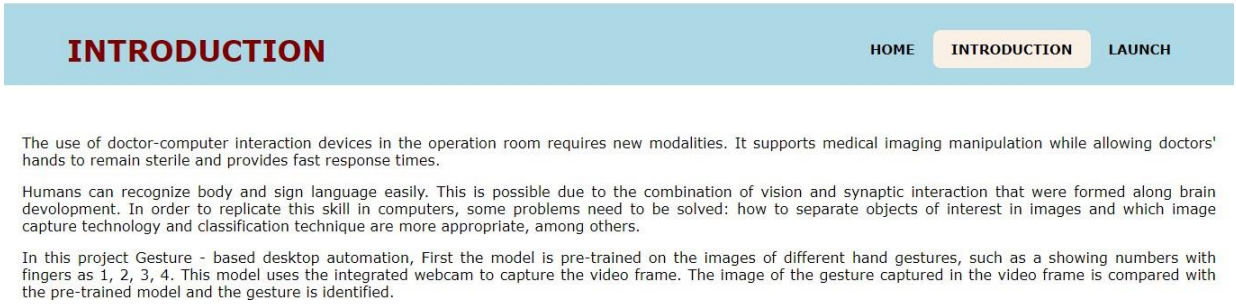
```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.

Let’s see how our Home.html page looks like:



When “Info” button is clicked, localhost redirects to “Intro.html”



Upload the image and click on Predict button to view the result