AI Enable car parking using OpenCV

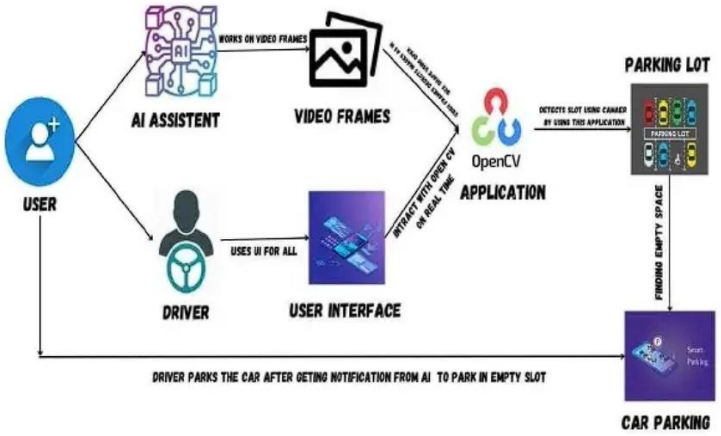
Car parking is a common problem faced by drivers in busy urban areas. For example, imagine you are driving to a shopping mall during peak hours. As you approach the mall, you notice that the parking lot is full, and several other cars are circling around looking for available spots.

You join the queue of cars, hoping to find an available spot soon. However, as time passes, you realize that the parking lot is overcrowded, and it's becoming increasingly difficult to find a spot. You start to feel frustrated and anxious, knowing that you might be late for your appointment or miss out on a great shopping opportunity.

AI-enabled car parking using OpenCV is a computer vision-based project that aims to automate the parking process. The project involves developing an intelligent system that can identify empty parking spaces and give the count of available parking spots.

The system uses a camera and OpenCV (Open Source Computer Vision) library to capture live video footage of the parking lot.

Architecture:



Pre-requisites:

1. To complete this project you must have the following software versions and packages.

- VScode (https://code.visualstudio.com/Download)
- Python (Download: https://www.python.org/downloads/release/python-370/)
- opencv-python
- cvzone
- flask

2. To make a responsive python script you must require the following packages.

Flask:

- Web framework used for building web applications.
- Flask Basics: Click here

If you are using VScode, follow the below steps to download the required packages:

- Open VScode prompt.
- Type "pip install opencv-python" and click enter.
- Type "pip install cvzone" and click enter.
- Type "pip install Flask" and click enter

If you are using VScode IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project objectives:

By the end of this project, you will:

- Know fundamental concepts and techniques of computer vision (OpenCV).
- Gain a broad understanding of image thresholding.

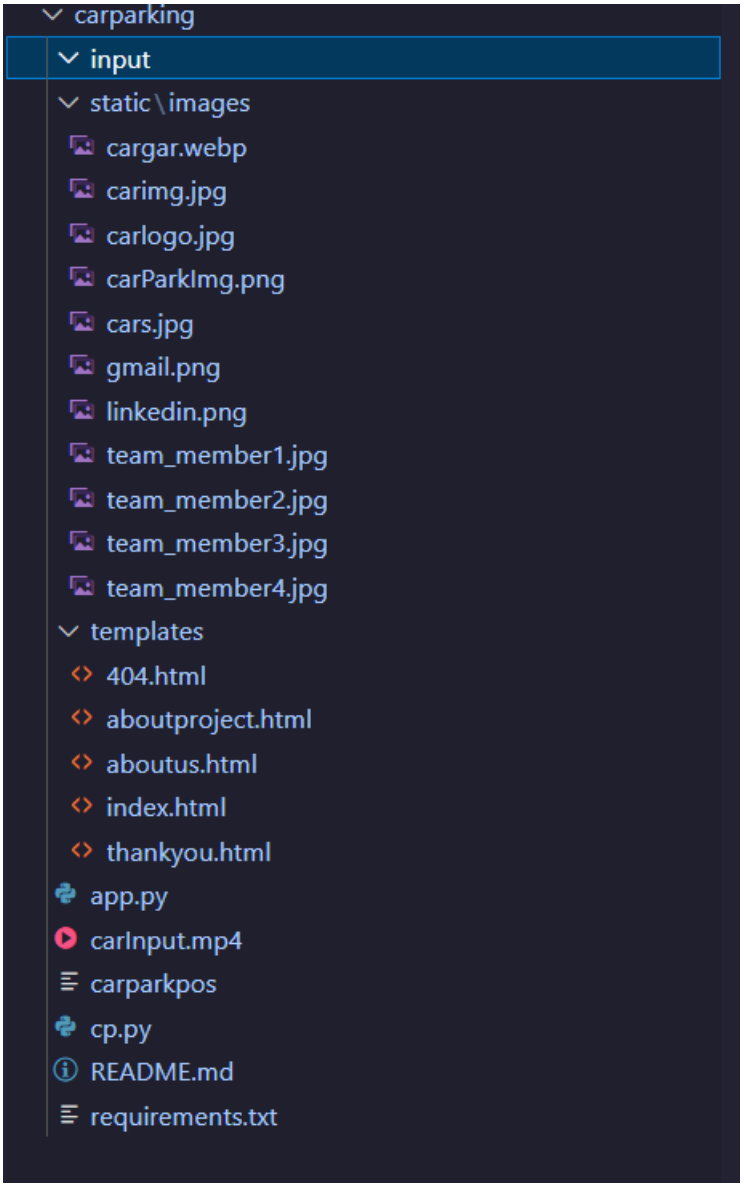
Project Flow:

- Data Collection ○ Download the video
- Video Processing and object detection ○ Import required libraries ○ Checking for parking space ○ Looping the video ○ Frame processing and empty parking slot counters
- Application building ○ Build HTML
    - ○ Build python script for Flask

Project Structure:

Create a project folder that contains files as shown below:

GITHUB LINK - https://github.com/kkhushitolanii/AI-Enabled-Car-Parking-System-using-OpenCV



1) Input folder will contain the uploaded video file which is then used by the ML model to do further Image processing.
2) The static folder contains a folder “image” which contains all the images used in the website.
3) The templates folder contains all the HTML files used in the website.
4) app.py is the main flask file from which we will be able to run the website on the local device.
5) Carinput.mp4 is the video on which the model is been trained on.

Milestone – 1: Data Collection

Activity 1: Download the video on the local device

Click the below link to download the video for AI-Enabled Car Parking using OpenCV. Click here.

Milestone – 2: ROI (Region of interest)

ROI stands for Region of Interest, which refers to a specific rectangular portion of an image or video frame that is used for processing or analysis.

Activity 1: Create a python file

Create a Python file in the directory and name it ‘cp.py’. This Python file is used for creating ROI and deleting ROI.

Activity 2: Importing the required packages

- **Opencv:** OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more. It supports multiple languages including Python, Java, and C++.
- **Pickle:** The pickle library in Python is used for serialization and de-serialization of Python objects. Serialization is the process of converting a Python object into a stream of bytes that can be stored in a file or sent over a network. De-serialization is the process of converting the serialized data back into a Python object.

```
import pickle
import cvzone
```

Activity 3: Define ROI width and height

- Calculating the ROI width and height (manually width and height are calculated and given as 107 & 48).
- In Python, try and except are used for error handling, to catch and handle exceptions that may occur during program execution. The try block is used to enclose the code that may raise an exception, and the except block is used to define what should happen if an exception is raised.

```
try:
    cap = cv2.VideoCapture('carparking/input/' + video_name)
    width, height = 107, 48
```

Activity 4: Select and deselect ROI

- A function is defined as mouseClick. As parameters we are passing events (mouse action), x (ROI starting point), y (ROI ending point), flags (Boolean flag), and params (other parameters).
- In 1st if condition: After the left click from the mouse, the starting and ending points will be added to posList by append method.
- In 2nd if condition: If ROI is selected in the unwanted region. Then we can remove that unwanted ROI by right-clicking from the mouse.
- Python objects are converted into a stream of bytes and stored in carparkpos file.

```
def mouseclick(events, x, y, flags, params):
    if events == cv2.EVENT_LBUTTONDOWN:
        poslist.append((x,y))
    if events==cv2.EVENT_RBUTTONDOWN:
        for i,pos in enumerate( poslist):
            x1,y1=pos;
            if x1<x<x1+width and y1<y<y1+height:
                poslist.pop(i)
    with open('carparkpos','wb') as f:
        pickle.dump(poslist,f)
```

Activity 5: Denote ROI with BBOX

- Reading the image with imread() method from cv2.
- All ROIs are saved in posList (refer activity 4).
- The rectangle is created as BBOX with the starting value (x) and ending value (y) from posList by rectangle() method. The parameters give image source, starting value, ending value, (starting value + width, ending value + height), (color), and thickness.
- For displaying the image imshow() method is used.
- setMouseCallback() function is used to perform some action on listening to the event which we have defined in activity 4.

```
while True:
    image = cv2.imread('carparking/images/carParkImg.png')
    cv2.rectangle(image,(50,190),(157,240),(255,0,255),2)

    for pos in poslist:
        cv2.rectangle(image, pos, (pos[0]+width,pos[1]+height), (255, 0, 255), 2)

    plt.imshow(image)
    cv2.setMouseCallback("image", mouseclick)
    cv2.waitKey(1)
```

Milestone - 3: Video processing and Object detection

Create a new Python file to perform video processing, object detection, and counters.

Activity 1: Import the required packages

Importing the required libraries to a new Python file.

```
import cv2
import pickle
import cvzone
import numpy as np
```

Activity 2: Reading input and loading ROI file

- VideoCapture() method from cv2 is used to capture the video input. Download the input video from milestone 1.
- Load the carparkpos file by the load() method from the pickle library. The carparkpos file is created from milestone 2. The ROI values are presented in the carparkpos file.
- Define width and height which we have used in milestone 2.

```
cap = cv2.VideoCapture('carparking/input/' + video_name)
width, height = 107, 48

with open('carparking/carparkpos', 'rb') as f:
    poslist = pickle.load(f)
```

Activity 3: Checking for parking space

- The function is defined with the name checkparkingspace. As a parameter, the image has to be passed.
- Taking the position from the position list and saving it to variables x and y.
- Cropping the image based on ROI (x and y value).
- To count the pixels from ROI, countNonZero() method is used from cv2. The BBOX (rectangle) is drawn in red color if the pixel count is lesser than 5000 else it is drawn in green color.
- The count of green color is displayed in the frame by putTextRect() method from the cvzone library.

```
def checkparkingspace(imgproceed):
    spacecouter=0;
    for pos in poslist:
        x,y=pos
        imgcrop=imgproceed[y:y+height,x:x+width]
        count = cv2.countNonZero(imgcrop)
        cvzone.putTextRect(image,str(count),(x,y+height-3),scale=1,thickness=2,offset=0,colorR=(0,0,255))

        if count>5000:
            color=(0,255,0)
            thickness=5
            spacecouter+=1
        else:
            color=(0,0,255)
            thickness=2
        cv2.rectangle(image, pos, (pos[0] + width, pos[1] + height), color, thickness)
    cvzone.putTextRect(image, f'Free:{spacecouter}/{len(poslist)}',(100, 50), scale=3, thickness=5, offset=20, colorR=(0, 200, 255))
```

Note: In this activity, function has been defined. But we have not called it.

Activity 4: Looping the video

- Here we calculate the total number of frames in the given video and we loop to each and every flame of the video to draw the rectangles around the parking slot. If no frames are left it will break.

```
while frame_count > 0:
    success, image = cap.read()

    if not success:
        break
```

Activity 5: Frame Processing and empty parking slot counters

- The captured video has to be read frame by frame. The read() method is used to read the frames.
- Opencv reads the frame in BGR (Blue Green Red).
- Converting BGR frame to grayscale image. The grayscale image is blurred with the GaussiamBlur() method from cv2.
- The adaptiveThreshold() method is used to apply a threshold to the blurred image. And again blur is applied to the image. Click here to understand about adaptiveThreshold().
- The dilate() method is used to compute the minimum pixel value by overlapping the kernel over the input image. The blurred image after thresholding and kernel are passed as the parameters.
- The checkparkingspace function is called with a dilated image. As per activity 3, we will get the count of empty parking slots.
- Display the frames in the form of video. The frame will wait for 10 seconds and it'll go to the next frame.

```
imggray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
imageblur = cv2.GaussianBlur(imggray, (3, 3), 1)
imgThreshold = cv2.adaptiveThreshold(imageblur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 25, 16)
imahemedian = cv2.medianBlur(imgThreshold, 5)
kernel = np.ones((3, 3), np.uint8)
imagedilate = cv2.dilate(imahemedian, kernel, iterations=1)

checkparkingspace(imagedilate)
cv2.imshow("output",image)

if cv2.waitKey(10) & 0xFF == ord('q'):
    break
frame_count -= 1
```

Milestone - 4: Application building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he/she has to navigate to detect button. Then the video will be showcased on the UI.

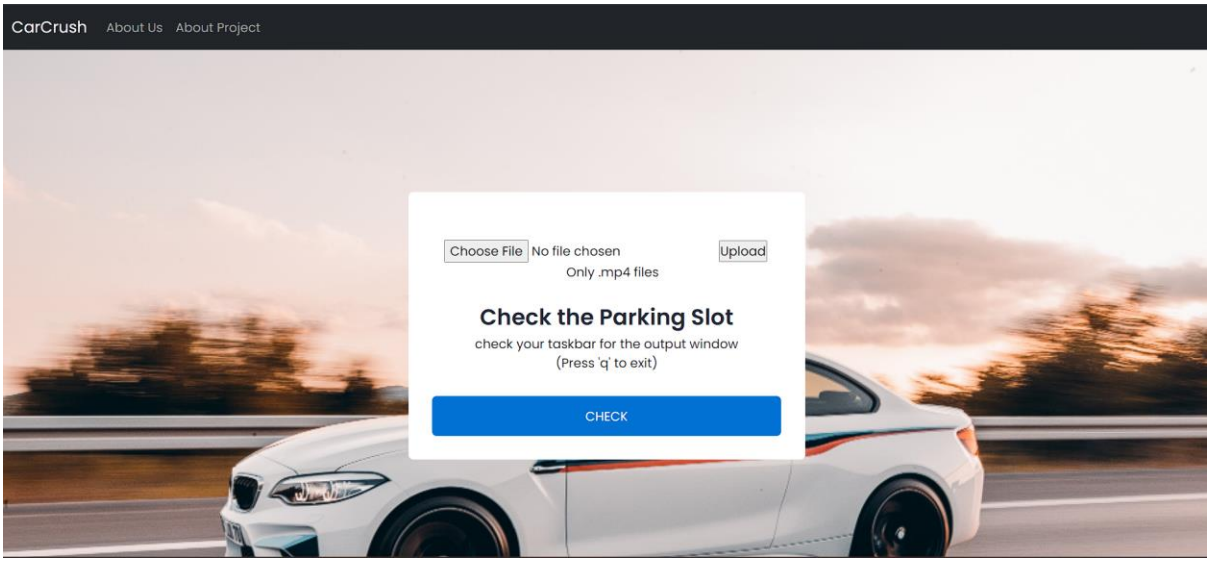
This section has the following tasks

- Building HTML Pages
- Building server-side script

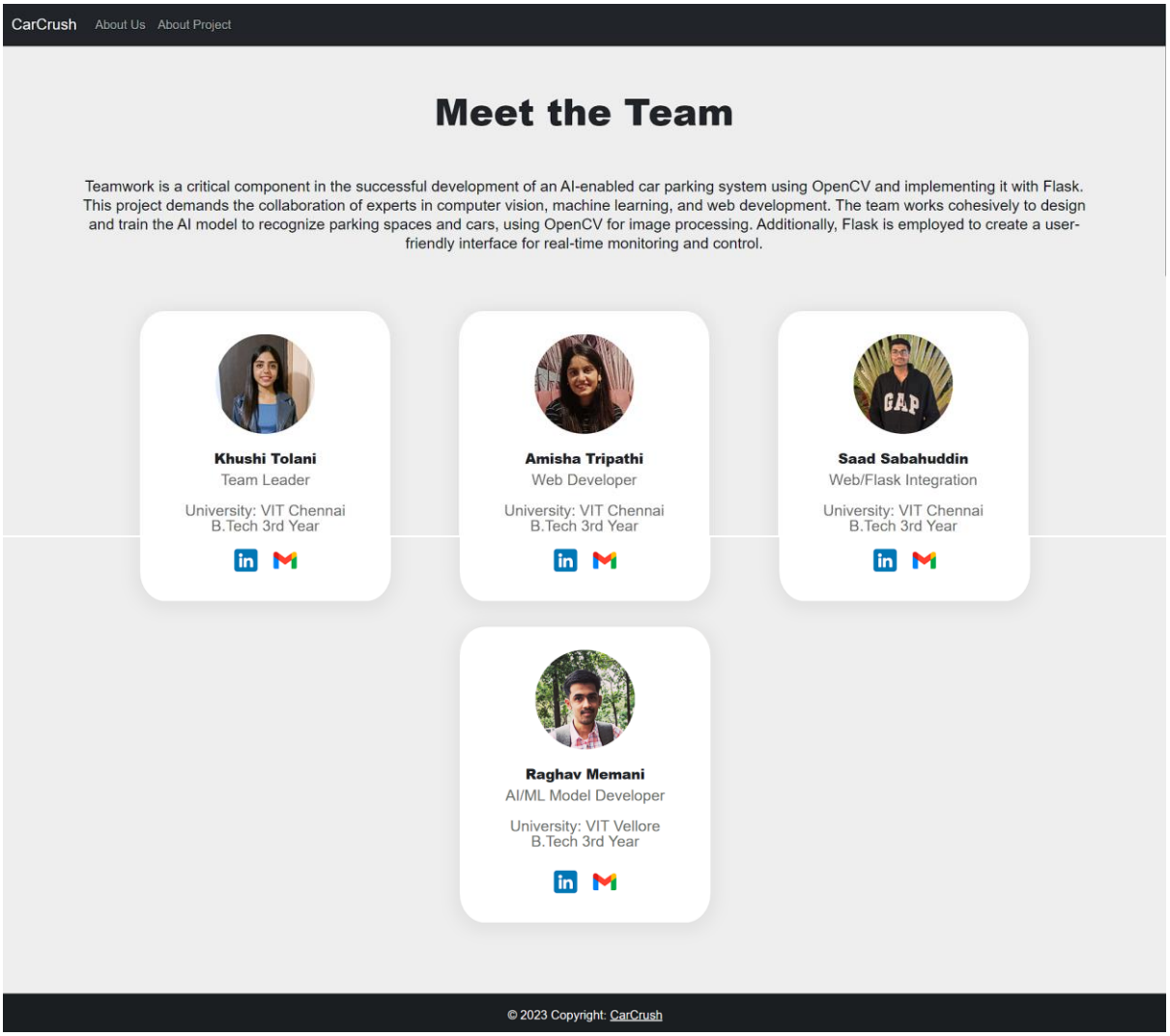
Activity1: Building Html Pages:

For this project, we have created 5 HTML files and saved them in the templates folder. Let's see what HTML pages look like:

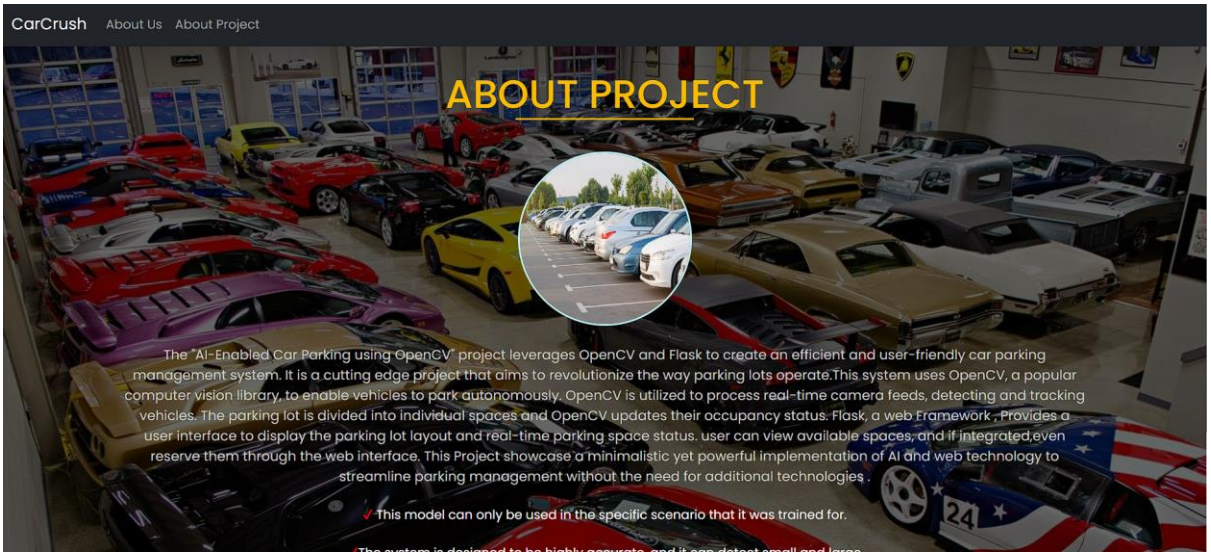
1) **main page:** It contains the home page button(CarCrush), About us, About Project, and upload button where you can upload your video(for the same scenario) and check how many slots are empty/full.



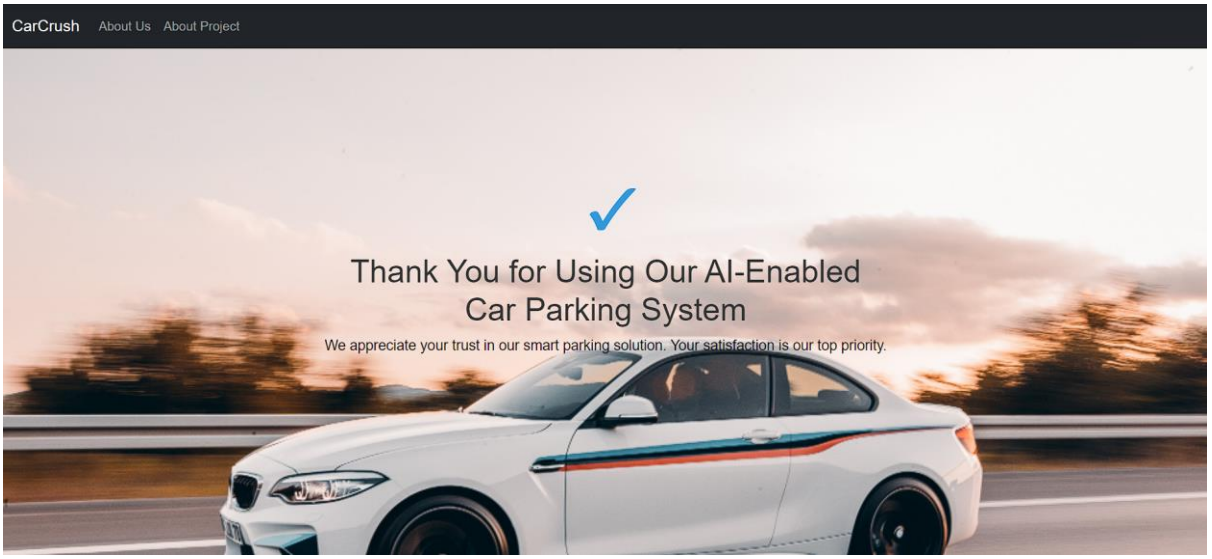
2) **About us:** In this we have given a brief description and the role played by each of them in the project, contact information.



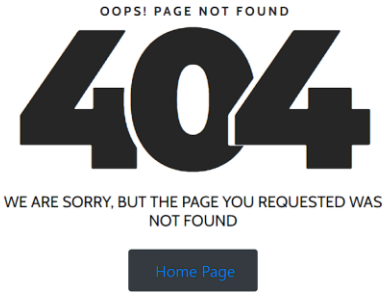
3) **About Project:** In this we have a brief explanation of our project and how it can be used further by integrating it with different technologies, some draw backs of this model/system.



4) **Thankyou page:**



5) **Error 404 page:** This is a customized 404 page not found page which is shown when you search for a directory which doesn't exist in the server.



- We have also added message box which helps the user to understand the error in a better manner.

Activity 2: Build Python code:

- Import the libraries

```
from flask import Flask, render_template, request, flash
import cv2
import pickle
import cvzone
import numpy as np
```

- Importing the flask module into the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as an argument.

```
app = Flask(__name__)
app.secret_key = 'your_secret_key'
```

Render HTML page:

- Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/aboutus')
def aboutus():
    return render_template('aboutus.html')

@app.route('/aboutproject')
def aboutproject():
    return render_template('aboutproject.html')

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html')
```

- create a decorator to route to '/aboutus goes to the About Us page, /aboutproject goes to the About Project page, errorhandler(404) goes to 404.html page.

Complete liv_pred() function:

```
def liv_pred():
    try:
        cap = cv2.VideoCapture('carparking/input/' + video_name)
        width, height = 107, 48

        with open('carparking/carparkpos', 'rb') as f:
            poslist = pickle.load(f)

        def checkparkingspace(imgproceed):
            spacecouter=0;
            for pos in poslist:
                x,y=pos
                imgcrop=imgproceed[y:y+height,x:x+width]
                count = cv2.countNonZero(imgcrop)
                cvzone.putTextRect(image,str(count),(x,y+height-3),scale=1,thickness=2,offset=0,colorR=(0,0,255))

                if count>5000:
                    color=(0,255,0)
                    thickness=5
                    spacecouter+=1
                else:
                    color=(0,0,255)
                    thickness=2
                cv2.rectangle(image, pos, (pos[0] + width, pos[1] + height), color, thickness)
            cvzone.putTextRect(image, f'Free:{spacecouter}/{len(poslist)}',(100, 50), scale=3, thickness=5, offset=20, colorR=(0, 200, 255))
```

```
    while frame_count > 0:
        success, image = cap.read()

        if not success:
            break

        imggray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        imageblur = cv2.GaussianBlur(imggray, (3, 3), 1)
        imgThreshold = cv2.adaptiveThreshold(imageblur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 25, 16)
        imahemedian = cv2.medianBlur(imgThreshold, 5)
        kernel = np.ones((3, 3), np.uint8)
        imagedilate = cv2.dilate(imahemedian, kernel, iterations=1)

        checkparkingspace(imagedilate)
        cv2.imshow("output",image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
        frame_count -= 1

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

- Main Function: Used to run the current module.

```
if __name__ == "__main__":
    app.run(debug=True)
```

Activity 3: Run the application

- Navigate to the folder where your python script is.
- Now type the "python app.py" command
- Navigate to the localhost where you can view your web page.
- Click on the check button to see the result/prediction.

```
* 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: on
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 135-972-108
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Output:

Click on the ‘Check’ button.