Ai Enable Car Parking Using Opencv

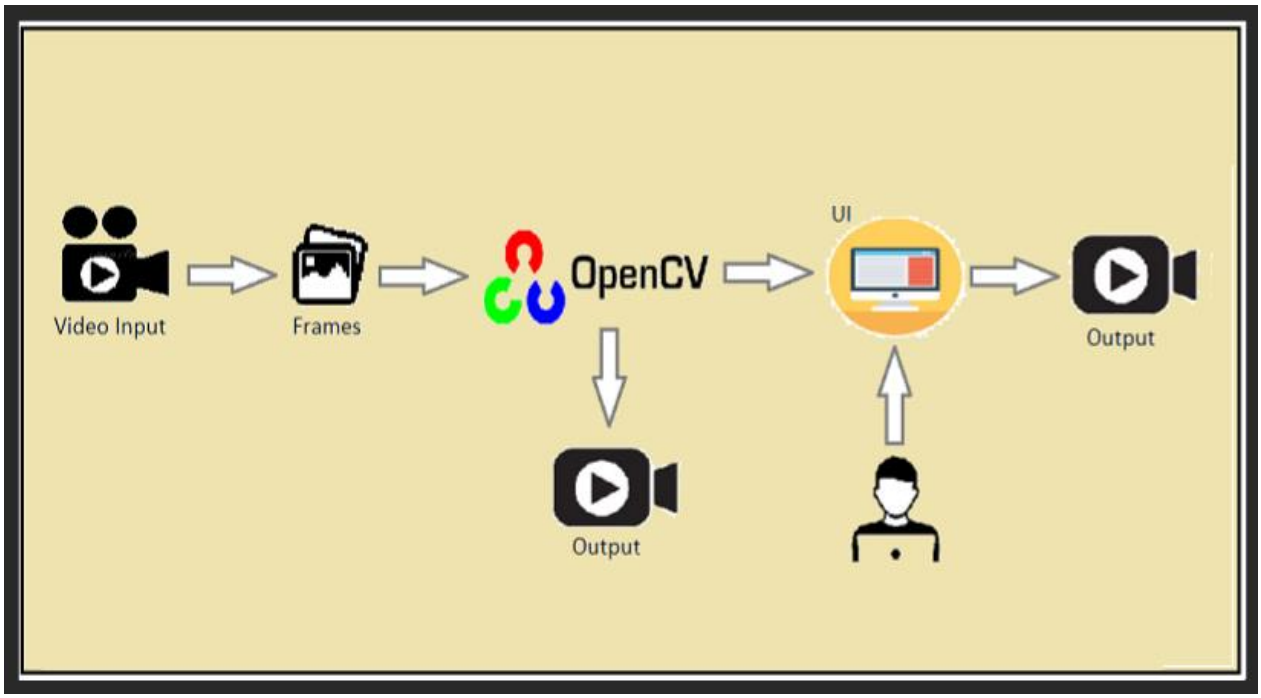
Car parking is a common problem faced by drivers in busy urban areas. For example, imagine you are driving to a shopping mall during peak hours. As you approach the mall, you notice that the parking lot is full, and several other cars are circling around looking for available spots.

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 gives 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.
   To make a responsive Python script you must require the following packages.

- PyCharm (Download: https://www.jetbrains.com/pycharm/)
- Python 3.7.0 (Download: https://www.python.org/downloads/release/python-370/)
- Numpy
- cvzone

Flask:

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

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

- Open anaconda 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 Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Flow

- Data Collection
  - Download the dataset

  ROI (Region of interest)

- Create python file
- Import required libraries
- Define ROI width and height
- Select and deselect ROI
- Denote ROI with BBOX

Video Processing and object detection

- Import required libraries
- Reading input and loading the ROI file
- Checking for parking space
- Looping the video
- Frame processing and empty parking slot counters

IBM Database Connection

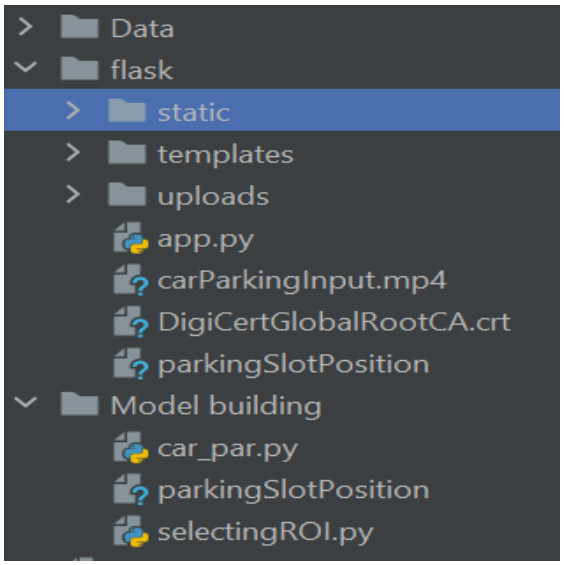
- Create IBM DB2 service and table

Application building

- Build HTML
- Build Python script for Flask

Project Structure

Create a project folder that contains files as shown below:



- The Dataset folder contains the video and image file
- For building a Flask Application we need HTML pages stored in the templates folder, CSS for styling the pages stored in the static folder, and a python script app.py for server-side scripting

Model Building folder consists of car.py &

Download The Dataset

Click the below link to download the dataset for AI-Enabled Car Parking using OpenCV.
Link - https://drive.google.com/drive/folders/13vNq4XZLV15uxxXrVkj33Jr7VflrhKWr?usp=share_link

ROI (Region Of Interest)

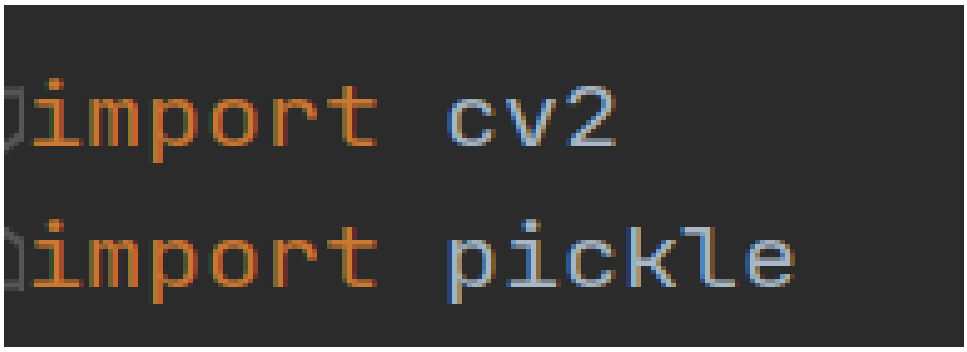
ROI stands for Region of Interest, which refers to a specific rectangular portion of an image or video frame that is used for processing or analysis.

Create A Python File

Create a python file in the directory and name it 'selectingROI.py'. This Python file is used for creating ROI and deleting ROI.

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 C++.
Pickle: The pickle library in Python is used for the 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.



Define ROI Width And Height

- Calculating the ROI width and height (manually width and height are calculated and given as 107 & 48).
- An empty file parkingSlotPosition is created to save all the ROI values. Try and except combo is used.
- 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.

```
# Define the width and height of ROI
width, height = 107, 48
# Creating an empty file and loading to a variable & Creating an empty list
try:
    with open('parkingSlotPosition', 'rb') as f:
        posList = pickle.load(f)
except:
    posList = []
```

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 click from the mouse.
- Python objects are converted into a stream of bytes and stored in the parkingSlotPosition file.

```python
def mouseClick(events, x, y, flags, params):
    # Adding ROI values to posList
    if events == cv2.EVENT_LBUTTONDOWN:
        posList.append((x, y))
    # Removing unwanted ROI from posList
    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)
    # Saving the posList values to parkingSlotPosition file
    with open('parkingSlotPosition', 'wb') as f:
        pickle.dump(posList, f)
```

Denote ROI With BBOX

- Reading the image with imread() method from cv2.
- All ROIs are saved in posList (refer to activity 4).
- The rectangle is created as BBOX with the starting value (x) and ending value (y) from posList by the rectangle() method. As 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.

```python
while True:
    img = cv2.imread('carParkImg.png')
    for pos in posList:
        cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 255, 255), 2)

    cv2.imshow("Image", img)
    cv2.setMouseCallback("Image", mouseClick)
    cv2.waitKey(1)
```

Video Processing And Object Detection

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

Import The Required Packages

Importing the required libraries to a new python file.

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

Reading Input And Loading The ROI File

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

```
# Video feed
cap = cv2.VideoCapture('carParkingInput.mp4')
# Loading the ROI from parkingSlotPosition file
with open('parkingSlotPosition', 'rb') as f:
    posList = pickle.load(f)
# Define width and height
width, height = 107, 48
```

Checking For Parking Space

- The function is defined with the name as carParkingSpace. As a parameter image has to be passed.
- Take the position from the position list and saved it to variables x and y.
- Cropping the image based on ROI (x and y value).

- To count the pixels from ROI, the countNonZero() method is used from cv2. The BBOX (rectangle) is drawn in green color if the pixel count is lesser than 900 else it is drawn in red color.
- The count of green color is displayed in the frame by the putTextRect() method from cvzone library.

```
def checkParkingSpace(imgPro):
    spaceCounter = 0
    for pos in posList:
        x, y = pos
        # Crop the image based on ROI
        imgCrop = imgPro[y:y + height, x:x + width]
        # Counting the pixel values from cropped image
        count = cv2.countNonZero(imgCrop)
        if count < 900:
            color = (0, 255, 0)
            thickness = 5
            spaceCounter += 1
        else:
            color = (0, 0, 255)
            thickness = 2
        # Draw the rectangle based on the condition defined above
        cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color, thickness)
    # Display the available parking slot count / total parking slot count
    cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50), scale=3,
                       thickness=5, offset=20, colorR=(0,200,0))
```

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

Looping The Video

- The current frame position is compared with the total number of frames. If it reaches the maximum frame, again the frame is set to zero. So, continuously it'll loop the frame.
- cv2.CAP_PROP_POS_FRAMES = Current frame
- cv2.CAP_PROP_FRAME_COUNT = Total no. of frame

```
while True:
    # Looping the video
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
```

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. And the grayscale image is blurred with GaussiamBlur() method from cv2.

- The adaptiveThreshold() method is used to apply the 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 the 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.

```
while True:
    # Looping the video
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    # Reading frame by frame from video
    success, img = cap.read()
    # Converting to gray scale image
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)  # Applying blur to image
    # Applying threshold to the image
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                         cv2.THRESH_BINARY_INV, 25, 16)
    imgMedian = cv2.medianBlur(imgThreshold, 5)  # Applying blur to image
    kernel = np.ones((3, 3), np.uint8)
    imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)
    # Passing dilate image to the function
    checkParkingSpace(imgDilate)
    cv2.imshow("Image", img)
    cv2.waitKey(10)
```

Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks

- Building HTML Pages
- Building python code

Building Html Pages

- For this project, we have created 2 HTML files and saved them in the templates folder. Let's see what those HTML pages looks like:

Build Python Code

- Import the libraries

```
from flask import Flask, render_template, request, session
import cv2
import pickle
import cvzone
import numpy as np
import ibm_db
import re
```

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

```
app = Flask(__name__)
```

Render HTML page:

- Here we will be using the declared constructor to route to the HTML page that we 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 project():
    return render_template('index.html')
@app.route('/hero')
def home():
    return render_template('index.html')
@app.route('/model')
def model():
    return render_template('model.html')
```

Create a decorator to route to '/predict_live'. Go to milestone 3 and copy the entire code and paste it inside the function liv_pred.

```
@app.route('/predict_live')
def liv_pred():
    # Video feed
    cap = cv2.VideoCapture('carParkingInput.mp4')
    with open('parkingSlotPosition', 'rb') as f:
        posList = pickle.load(f)
    width, height = 107, 48
    def checkParkingSpace(imgPro):
        spaceCounter = 0
        for pos in posList:
            x, y = pos
            imgCrop = imgPro[y:y + height, x:x + width]
            # cv2.imshow(str(x * y), imgCrop)
            count = cv2.countNonZero(imgCrop)
            if count < 900:
                color = (0, 255, 0)
                thickness = 5
                spaceCounter += 1
            else:
                color = (0, 0, 255)
                thickness = 2
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color, thickness)
```

```
    cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50), scale=3,
                       thickness=5, offset=20, colorR=(0, 200, 0))
while True:
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    success, img = cap.read()
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                         cv2.THRESH_BINARY_INV, 25, 16)
    imgMedian = cv2.medianBlur(imgThreshold, 5)
```

```
kernel = np.ones((3, 3), np.uint8)
imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)
checkParkingSpace(imgDilate)
cv2.imshow("Image", img)
# cv2.imshow("ImageBlur", imgBlur)
# cv2.imshow("ImageThres", imgMedian)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

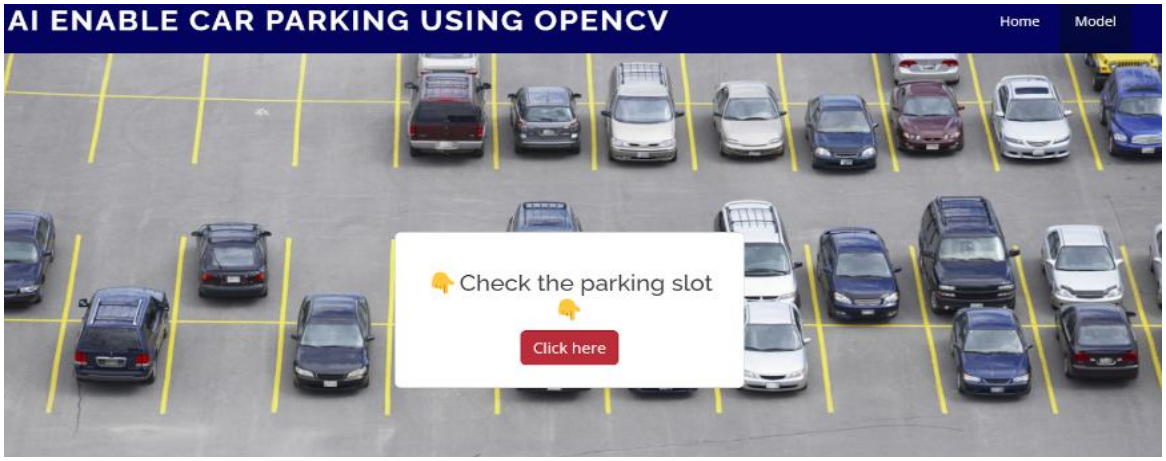
Run The Application

- Open the anaconda prompt from the start menu
- 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 predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
* 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 ‘Click here’ button.




Free: 15/69