

EMPTY PARKING SPOT DETECTION SYSTEM

A PROJECT REPORT

SUBMITTED BY

THANDA SRI DIKSHITHA

THATIPAMULA SOUMYA

NAMDHARI ANUSHA

in partial fulfillment of academic requirements in VI – Semester (Sixth Semester)

IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Under the guidance of

Dr. B.SUJATHA

Assistant Professor

UNIVERSITY COLLEGE OF ENGINEERING

OSMANIA UNIVERSITY , HYDERABAD – 500 007



**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND
MACHINE LEARNING**

UNIVERSITY COLLEGE OF ENGINEERING

OSMANIA UNIVERSITY

CERTIFICATE

This is to certify that the Project work entitled **EMPTY PARKING SPOT DETECTION SYSTEM** submitted by **THANDA SRI DIKSHITHA** (100521729087), **THATIPAMULA SOUMYA** (100521729084), **Namdhari Anusha** (100521729082) students of department of **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING** in the partial fulfillment of academic requirements in VI- Semester (Sixth Semester) is a record of bonafide work carried out by them in academic year 2024.

Project Guide

Dr. B.Sujatha

Assistant Professor

Dept. of CSE, OU

Head of the department

Prof.P.V.Sudha

Professor

Dept. of CSE, OU

ACKNOWLEDGEMENT

"We would like to express our deep gratitude and wholehearted thanks to our project guide, Dr. B. Sujatha, Assistant Professor in the Department of Computer Science and Engineering at University College of Engineering, Osmania University. We are grateful for her guidance, support, constructive criticism, and engaging discussions throughout our dissertation work.

We also extend our thanks to Prof. P. V. Sudha, Head of the Department of Computer Science and Engineering, for her support and providing us with access to all the resources available to students.

Our appreciation also goes out to the entire faculty of the Department of Computer Science and Engineering at University College of Engineering, Osmania University, for their encouragement and for allowing us to utilize the various resources within the department.

Special thanks to our parents and friends for their valuable suggestions, moral support, strength, and encouragement throughout the completion of our project."

THANDA SRI DIKSHITHA(100521729087)

THATIPAMULA SOUMYA(100521729084)

NAMDHARI ANUSHA(100521729082)

ABSTRACT

Parking slot detection is a critical aspect of modern urban management and smart city applications, providing efficient and effective use of parking resources. This project presents an automated system for detecting empty parking slots using a Convolutional Neural Network (CNN) model implemented with Keras and OpenCV for image processing. The system leverages a pre-trained CNN model to classify parking slots as empty or occupied based on grayscale image inputs.

The workflow involves several stages: image preprocessing to enhance features, defining regions of interest (ROI) for each parking slot, and using the trained CNN model to classify these regions. The CNN model, trained on a dataset of labeled parking slot images, achieves accurate predictions by learning from the features extracted during the preprocessing stage. The model's architecture includes multiple convolutional and pooling layers, followed by dense layers for classification, ensuring robust feature extraction and decision-making capabilities.

The system is evaluated on real-world parking lot images, demonstrating its ability to accurately identify empty parking slots. The results are visualized by drawing bounding boxes around detected slots and displaying the count of available parking spaces on the image. The system's performance is enhanced by careful selection of model parameters, appropriate image preprocessing techniques, and the implementation of an efficient prediction threshold. This automated parking slot detection system provides a foundation for developing advanced parking management solutions, contributing to reduced traffic congestion and improved urban mobility..

Keywords

- Parking Slot Detection
- Convolutional Neural Network (CNN)
- Keras
- OpenCV
- Image Processing

- Automated Systems
- Real-time Analysis
- Transfer Learning

Table of Contents

CHAPTER 1- INTRODUCTION

- 1.1 OVERVIEW OF THE PROJECT
- 1.2 OBJECTIVES
- 1.3 MODULES
- 1.4 SCOPE OF THE PROJECT

CHAPTER 2 LITERACY SURVEY

- 2.1 EXISTING SYSTEM
 - 2.1.1 DEMERITS OF THE EXISTING SYSTEM
- 2.2 PROPOSED METHODOLOGY
 - 2.2.1 ADVANTAGES OF PROPOSED METHODOLOGY

CHAPTER 3- PROBLEM STATEMENT

CHAPTER 4-SYSTEM REQUIREMENTS

- 4.1 HARDWARE REQUIREMENT
- 4.2 SOFTWARE REQUIREMENT

CHAPTER 5- SYSTEM DESIGN

- 5.1 INTRODUCTION
- 5.2 DATAFLOW
 - 5.2.1 CONTEXT LEVEL DATAFLOW DIAGRAM

CHAPTER 6- TECHNOLOGIES USED

- 6.1 COMPUTER VISION

6.2 COMPUTATIONAL NEURAL NETWORK

6.3 KERAS

6.4 MATPLOTLIB

CHAPTER 7- IMPLEMENTATION

7.1 Installation

CHAPTER 8- RESULT

8.1 OUTPUT SCREENSHOTS

CHAPTER 9- CONCLUSION

9.1 FUTURE SCOPE

CHAPTER 10- BIBIOGRAPHY AND REFERENCE

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

This project aims to develop an automated system for detecting empty parking slots using a combination of Convolutional Neural Networks (CNN) and image processing techniques with OpenCV. Leveraging a pre-trained CNN model built with Keras, the system processes input images of parking lots to identify and classify parking slots as either empty or occupied. The workflow involves several key steps: preprocessing the images to enhance features, defining regions of interest for each parking slot, and using the trained model to classify these regions. The model, trained on a dataset of labeled parking slot images, utilizes convolutional and pooling layers to extract relevant features and dense layers for classification, achieving accurate predictions. The system's accuracy is further enhanced through careful parameter tuning and the implementation of a suitable prediction threshold.

The results of the parking slot detection are visualized by drawing bounding boxes around the detected slots, with green indicating empty slots and red indicating occupied ones. Additionally, the count of available parking spaces is displayed on the image, providing a clear and concise overview of parking availability.

1.2 OBJECTIVES

- 1. Development of an Automated System:** Create an automated system capable of accurately identifying empty parking slots from images of parking lots, reducing the need for manual inspection and enhancing efficiency in parking management.
- 2. Implementation of Convolutional Neural Networks (CNN):** Utilize CNNs, a powerful deep learning technique, to classify parking slots as either empty or occupied based on image features, enabling accurate and robust detection.
- 3. Integration of Image Processing Techniques:** Incorporate image preprocessing techniques using OpenCV to enhance image quality, extract relevant features, and define regions of interest (ROIs) corresponding to individual parking slots.
- 4. Visualization of Results:** Provide visual feedback by marking detected parking slots on the image and displaying the count of empty slots, facilitating quick and intuitive assessment of parking availability.
- 5. Optimization for Accuracy:** Train the CNN model on a dataset of labeled parking slot images, optimizing model parameters and tuning hyperparameters to achieve high accuracy in classification and detection.
- 6. Foundation for Future Enhancements:** Lay the groundwork for future advancements in parking management solutions, such as real-time video feed analysis, integration with smart city infrastructure, and exploration of advanced machine learning techniques for improved accuracy and efficiency.

1.2. MODULES

1. Data Collection Module:

- Responsible for gathering images of parking slots, including both training and testing datasets.
- Involve manual annotation of parking slots to indicate their occupancy status (empty or occupied).

2. Preprocessing Module:

- Performs preprocessing tasks on the input images to enhance features and prepare them for input into the CNN model.
- Includes operations such as resizing images to a standard size, converting to grayscale, and normalizing pixel values.

3. CNN Model Training Module:

- Constructs and trains the Convolutional Neural Network (CNN) model using Keras or a similar deep learning framework.
- Involves defining the architecture of the CNN, compiling the model with appropriate loss and optimization functions, and fitting the model to the training data.

4. Image Processing Module:

- Utilizes OpenCV for image processing tasks, such as defining regions of interest (ROIs) corresponding to parking slots in the input images.
- Responsible for extracting ROIs, resizing them, and converting them to the appropriate format for input into the CNN model.

5. Detection and Classification Module:

- Implements the logic for detecting and classifying parking slots as either empty or occupied.
- Uses the trained CNN model to predict the occupancy status of each ROI extracted from the input images.

6. Visualization Module:

- Provides visual feedback by overlaying bounding boxes on the input images to indicate the detected parking slots and their occupancy status (e.g., green for empty, red for occupied).
- Displays additional information, such as the count of empty parking slots, to facilitate easy interpretation of the results.

7. Integration and Deployment Module:

- Integrates all the components/modules into a cohesive system.
- Handles deployment aspects, such as running the system on different environments (e.g., local machines, cloud platforms) and interfacing with external systems or devices (if applicable).

8. Evaluation and Testing Module:

- Conducts evaluation and testing of the system to assess its performance, accuracy, and reliability.
- Involves validating the system against ground truth data, identifying any issues or areas for improvement, and iterating on the design accordingly.

1.4. SCOPE OF THE PROJECT

The parking slot detection is to develop an automated system capable of accurately identifying empty parking slots from images of parking lots. This involves leveraging Convolutional Neural Networks (CNN) and image processing techniques, particularly with OpenCV, to preprocess input images, define regions of interest (ROIs), and classify parking slots as either empty or occupied. A significant aspect of the project is the training and evaluation of the CNN model using a dataset of labeled parking slot images, ensuring its effectiveness in real-world scenarios. The system provides visual feedback by marking detected parking slots on the input images, using bounding boxes to indicate their locations and occupancy status. Additionally, the count of empty parking slots is displayed to facilitate quick and intuitive assessment of parking availability.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SYSTEM

Traffic congestion at parking slots is a significant issue due to the rapid increase in vehicles without adequate parking infrastructure. Sensor-based methods, such as ultrasonic detectors and RFID systems, offer real-time data but come with high installation and maintenance costs. Vision-based methods, using web cameras and algorithms like OpenCV, provide a cost-effective solution but require substantial computational resources and maintenance. Learning-based algorithms, such as PSDL, and deep learning models, like Mask R-CNN, enhance accuracy and adaptability. Mask R-CNN, in particular, shows high potential in detecting and classifying parking slot occupancy, though it demands significant computational power. The evolution from traditional to deep learning models underscores the need for future research to optimize efficiency and real-world application.

2.1.1. DEMERITS OF THE EXISTING SYSTEM

1. High computational requirements.
2. Complex setup process.
3. Dependence on equipment reliability.
4. Sensitivity to data quality.
5. Variable performance in different environments.
6. Expensive implementation.
7. Potential privacy concerns.
8. Limited applicability in diverse parking settings.
9. Insufficient real-world testing.
10. Ethical considerations.

2.2. PROPOSED METHODOLOGY

Parking slot detection focus on using advanced techniques like Mask R-CNN and Artificial Neural Networks (ANN) to address traffic congestion. By installing surveillance cameras, manually segmenting parking slots, and geo-tagging them, the system captures real-time video streams and processes frames at regular intervals. The Mask R-CNN model detects vehicles, calculates the area they occupy, and classifies slots as fully or partially occupied. This data is stored in a centralized database, enabling users to reserve slots based on real-time availability. Additionally, a multi-camera

network ensures accuracy by covering the entire parking area and avoiding duplications. These methods offer high accuracy and robust performance under various conditions, reducing traffic congestion and optimizing parking space usage

2.2.1. ADVANTAGES OF PROPOSED METHODOLOGY

1. High precision in detecting empty slots.
2. Real-time monitoring using cameras.
3. Accurate classification of parking slots.
4. Efficient storage in a centralized database.
5. Reduction in traffic congestion.
6. Cost-effective compared to sensor-based systems.
7. Flexibility in adapting to various environments.
8. Automated check-in and check-out process.

CHAPTER 3

PROBLEM STATEMENT

This project aims to address this challenge by developing an automated system using deep learning and computer vision techniques to detect and classify empty parking slots from images of parking Slots. The system will provide real-time feedback on parking availability, helping to streamline parking management, reduce traffic congestion, and enhance the overall urban mobility experience for drivers.

CHAPTER 4

SYSTEM REQUIREMENTS SPECIFICATIONS

4.1 HARDWARE REQUIREMENTS

- Processor : i5 intel processor
- Hard disk space : 160GB
- RAM : 8GB
- Monitor : 15” LED
- Input devices : Keyboard , mouse

4.2 SOFTWARE REQUIREMENTS

- Operating System : Windows 11
- IDE : Visual Studio Code
- Technology : CNN, Open CV, Keras, Deep learning
- Web Browser : Google Chrome/Mozilla Firefox/OPERA

CHAPTER 5

SYSTEM DESIGN

5.1 INTRODUCTION

Designing an empty parking spot detection system involves a comprehensive approach that integrates various technologies and architectural elements. Here's a detailed outline for your project report

5.2 DATA FLOW

1. Data Collection:

- Take pictures of parking lots.

2. Preparing Data:

- Resize, convert to black and white, and adjust colors.

3. Augmentation :

- Create variations of images for better learning.

4. Splitting Data:

- Divide images into training, validation, and test sets.

5. Annotation:

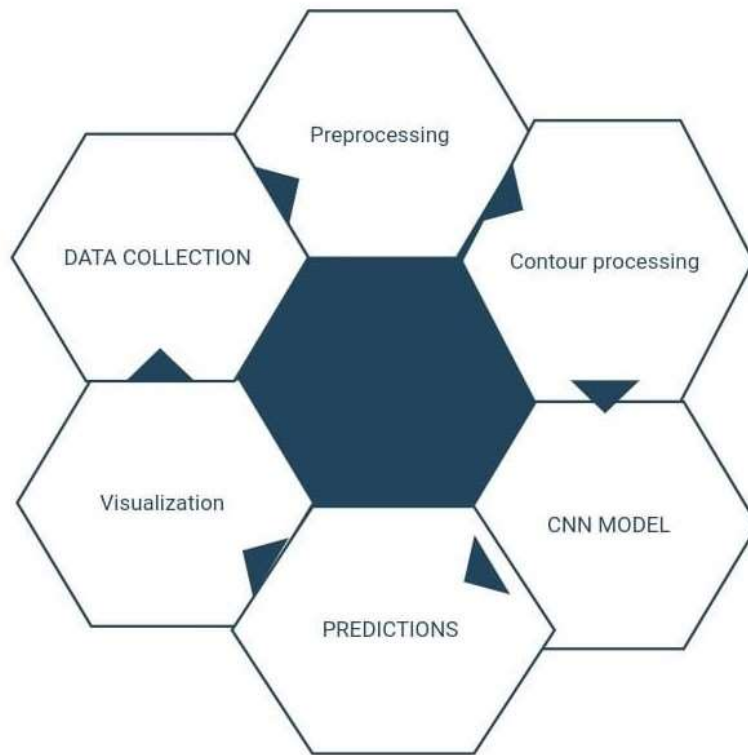
- Label parking spots as empty or occupied.

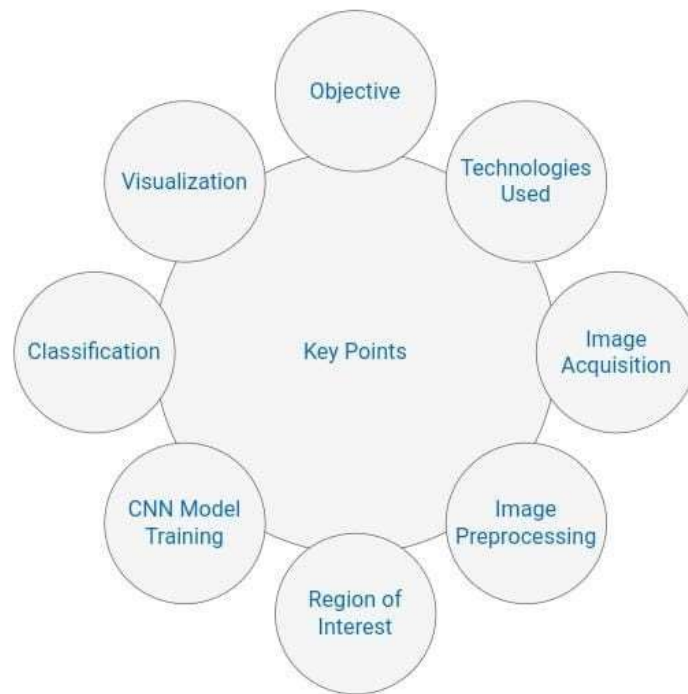
6. Storage:

- Store preprocessed images and labels for training.

5.2.1 CONTEXT LEVEL DATA FLOW DIAGRAM

A context diagram is a top level data flow diagram. which provides an overview of the entire system. It shows the major processes, data flows, and data stores in the system.





The system is able to check if the parking spot is empty or occupied by car. So when a parking spot is empty, then there is a green rectangle, but when the parking spot is occupied by car, then there is a red rectangle. In addition to that, this system is also able to count how many empty parking spots left. We are integrating advanced computer vision technologies into traffic management

why empty parking spot detection system.?

the empty parking spot detection system offers numerous benefits to individuals and communities by providing real time information on available parking spaces. This system helps to reduce time wasted searching for parking, particularly in densely populated urban areas. Finding parking spot is very difficult, especially during the weekend. Sometimes it takes ten minutes or even 15 minutes to find a parking spot. One of the most effective solution is to build empty parking spot detection system, so it can help drivers to find empty parking spot. my intention behind doing this project is to work on computer vision, object detection, and image processing to build complex projects with real use cases in traffic management and smart city development.

Tools, IDE, and Datasets Tools:

Programming language: Python <https://www.python.org/>

Libraries: OpenCV, Numpy, and keras

IDE: <https://code.visualstudio.com/>

Dataset:

"C:\Users\Sridi\Desktop\mini2\images.jpg"



<https://www.kaggle.com/datasets/ljkeller/matchbox-car-parking-occupancy>

vehicle detection system using: OpenCV and tensorflow

CHAPTER 6

TECHNOLOGIES USED

6.1 COMPUTER VISION

Computer vision can be utilized to analyze live or recorded video feeds from parking lots, identifying vacant parking spaces through image processing techniques such as object detections and segmentation. This technology enables real time monitoring of parking availability, improving parking efficiency, and providing valuable information to drivers who are searching for parking spots.

6.2 CONVOLUTIONAL NEURAL NETWORKS

we are going to use it to analyze images captured by cameras placed in parking slots. So these convolutional neural networks are trained to detect patterns and features indicative of empty parking spaces, such as the absence of vehicles

6.3 KERAS

High-level neural networks API, used with TensorFlow backend, for building and training Convolutional Neural Networks (CNNs) for image classification tasks

6.4 Matplotlib

Library for plotting and visualizing data, utilized for displaying images with annotated bounding boxes and other visualizations .

CHAPTER 7

IMPLEMENTATION

Implementation Details:

This project can be broken down into 4 modules:

1.IMAGE PROCESSING

```
parking > 1.ImageProcessing.py > ...
1 import cv2
2
3 img = cv2.imread(r"C:\Users\Sridi\Desktop\mini2\images.jpg")
4 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5 blurred_gray = cv2.GaussianBlur(gray, (7,7), 0)
6 edges = cv2.Canny(blurred_gray, 30, 150)
7 contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
8 for contour in contours:
9     approx = cv2.approxPolyDP(contour, 0.02 * cv2.arcLength(contour, True), True)
10    if cv2.contourArea(contour) > 20 and len(approx) > 4:
11        x,y,w,h = cv2.boundingRect(contour)
12        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0),2)
13
14 cv2.imshow("Cars", img)
15 cv2.waitKey(0)
16 cv2.destroyAllWindows()
17
18
19
```

2.MANUAL ANNOTATION

```
parking > 2.parkingspotcoordinate.py > ...
1 import cv2
2
3 drawing = False
4 ix, iy = -1,-1
5 img = cv2.imread(r"C:\Users\Sridi\Desktop\mini2\images.jpg")
6 def draw_rectangle(event, x, y, flags, param):
7     global ix, iy, drawing
8     if event == cv2.EVENT_LBUTTONDOWN:
9         drawing = True
10        ix, iy = x, y
11    elif event == cv2.EVENT_MOUSEMOVE:
12        if drawing:
13            img_rect = img.copy()
14            cv2.rectangle(img_rect, (ix,iy), (x,y), (0, 255, 0), 2)
15            cv2.imshow("image",img_rect)
16    elif event == cv2.EVENT_LBUTTONUP:
17        drawing = False
18        cv2.rectangle(img, (ix,iy), (x,y), (0, 255, 0), 2)
19        cv2.imshow("image",img)
20        print("Top Left:", (ix,iy))
21        print("Bottom Right:", (x,y))
22
23 cv2.namedWindow('image')
24 cv2.setMouseCallback('image', draw_rectangle)
25 cv2.imshow('image', img)
26
27 while True:
28     if cv2.waitKey(1) & 0xFF == ord('q'):
29         break
30 cv2.destroyAllWindows()
31
32
```

3.MODEL-TRAINING

```
1.ImageProcessing.py 2.parkingspotcoordinate.py 3.emptyparkingspotdetectionmodel.py 3 kk.py 3 4.emptyparkingspotdetection.py 1
parking > 3.emptyparkingspotdetectionmodel.py > ...
1 import os
2 import cv2
3 import numpy as np
4 from sklearn.model_selection import train_test_split
5 from tensorflow.keras.models import Sequential
6 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
7 from tensorflow.keras.utils import to_categorical
8
9 training_data = [
10     "C:\\Users\\Sridi\\Desktop\\mini2\\archive\\matchbox_cars_parkinglot\\empty",
11     "C:\\Users\\Sridi\\Desktop\\mini2\\archive\\matchbox_cars_parkinglot\\occupied"
12 ]
13
14 def load_images(training_data):
15     images = []
16     labels = []
17     for i, folder in enumerate(training_data):
18         label = i
19         for filename in os.listdir(folder):
20             try:
21                 img = cv2.imread(os.path.join(folder, filename), cv2.IMREAD_GRAYSCALE)
22                 img = cv2.resize(img, (48,48))
23                 images.append(img)
24                 labels.append(label)
25             except Exception as e:
26                 print(f"Error loading image {os.path.join(folder, filename)}: {e}")
27     return np.array(images), np.array(labels)
28
29 images, labels = load_images(training_data)
30 X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
31 X_train = X_train.reshape(X_train.shape[0], 48, 48, 1).astype('float32') / 255
32 X_test = X_test.reshape(X_test.shape[0], 48, 48, 1).astype('float32') / 255
33 y_train = to_categorical(y_train)
34 y_test = to_categorical(y_test)
35
36 model = Sequential()
37 model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(48,48,1)))
38 model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
39 model.add(MaxPooling2D(pool_size=(2,2)))
40 model.add(Dropout(0.25))
41 model.add(Flatten())
42 model.add(Dense(128, activation='relu'))
43 model.add(Dropout(0.5))
44 model.add(Dense(2, activation='softmax'))
45
46 model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
47 model.fit(X_train, y_train, batch_size=64, epochs= 20, verbose=1, validation_data=(X_test, y_test))
48 model.save("emptyparkingspotdetectionmodel.h5")
```

4.REAL-TIME DETECTION

```
parking > 4.emptyparkingspotdetection.py > ...
1 import cv2
2 import numpy as np
3 from keras.models import load_model
4
5 model = load_model("emptyparkingspotdetectionmodel.h5")
6 model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
7
8 coordinates = [
9     [(8, 0), (27, 40)],
10    [(29, 2), (45, 41)],
11    [(45, 2), (63, 40)],
12    [(62, 1), (79, 38)],
13    [(79, 4), (98, 36)],
14    [(97, 2), (118, 34)],
15    [(119, 0), (135, 36)],
16    [(135, 1), (155, 39)],
17    [(156, 2), (171, 39)],
18    [(173, 2), (191, 36)],
19    [(191, 2), (204, 36)],
20    [(209, 6), (225, 39)],
21    [(227, 3), (253, 40)],
22    [(11, 82), (30, 120)],
23    [(34, 82), (47, 119)],
24    [(49, 82), (63, 120)],
25    [(63, 82), (81, 118)],
26    [(83, 82), (98, 115)],
27    [(100, 83), (117, 116)],
28    [(116, 83), (137, 118)],
29    [(139, 82), (153, 118)],
30    [(154, 85), (171, 117)],
31    [(173, 81), (190, 117)],
32    [(193, 84), (209, 120)],
33    [(213, 86), (224, 120)],
34    [(226, 82), (252, 119)],
35    [(12, 121), (29, 154)],
36    [(30, 120), (45, 157)],
37    [(49, 119), (64, 155)],
38    [(66, 123), (84, 156)],
39    [(85, 120), (100, 156)],
```

```
parking > 4.emptyparkingspotdetection.py > ...
45    [(170, 119), (191, 155)],
46    [(193, 124), (213, 155)],
47    [(215, 124), (233, 156)],
48    [(238, 121), (258, 157)]
49 ]
50
51 def detect_empty_parking(image, spot):
52     x1, y1 = spot[0]
53     x2, y2 = spot[1]
54     if x1 >= x2 or y1 >= y2 or x1 < 0 or y1 < 0 or x2 > image.shape[1] or y2 > image.shape[0]:
55         print("Invalid coordinates for ROI")
56         return False
57     roi = image[y1:y2, x1:x2]
58     if roi.size == 0:
59         print("Empty ROI")
60         return False
61     gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
62     resized_roi = cv2.resize(gray_roi, (48,48))
63     resized_roi = resized_roi.astype('float32') / 255
64     resized_roi = np.expand_dims(resized_roi, axis=0)
65     resized_roi = np.expand_dims(resized_roi, axis=-1)
66     prediction = model.predict(resized_roi)
67     threshold = 0.01
68     if prediction[0][0] > threshold:
69         return True
70     else:
71         return False
72
73 current_image = cv2.imread(r"C:\Users\Sridi\Desktop\mini2\images.jpg")
74 empty_count = 0
75
76 for spot in coordinates:
77     if detect_empty_parking(current_image, spot):
78         cv2.rectangle(current_image, spot[0], spot[1], (0,255,0), 2)
79         empty_count += 1
80     else:
81         cv2.rectangle(current_image, spot[0], spot[1], (0,0,255), 2)
82
83 font = cv2.FONT_HERSHEY_SIMPLEX
```

```

cv2.rectangle(current_image, spot[0], spot[1], (0,0,255), 2)

font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(current_image, f"Empty Spots: {empty_count}", (65,65), font, 0.69, (0,0,0), 2, cv2.LINE_AA)

cv2.imshow("Parking Lot", current_image)
desired_width = 800
desired_height = 600
image_resized = cv2.resize(current_image, (desired_width, desired_height))

# Create a window and resize it
window_name = 'Training and Validation Plot'
cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)
cv2.resizeWindow(window_name, desired_width, desired_height)

# Display the image
cv2.imshow(window_name, image_resized)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

INSTALLATION

Step 1: Install the required packages

- pip install cv2
- pip install tensorflow
- pip install keras
- pip install sklearn
- pip install numpy

Step 2:

Import the required libraries

```
import os
```

```
import cv2
```

```
import numpy as np
```

```
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
from tensorflow.keras.utils import to_categorical
```

Step 3: Run the code

```

# To run 1.ImageProcessing
$ python ImageProcessing.py

```

```
# To run 2.parkingspotcoordinate.py
$ python 2.parkingspotcoordinate.py

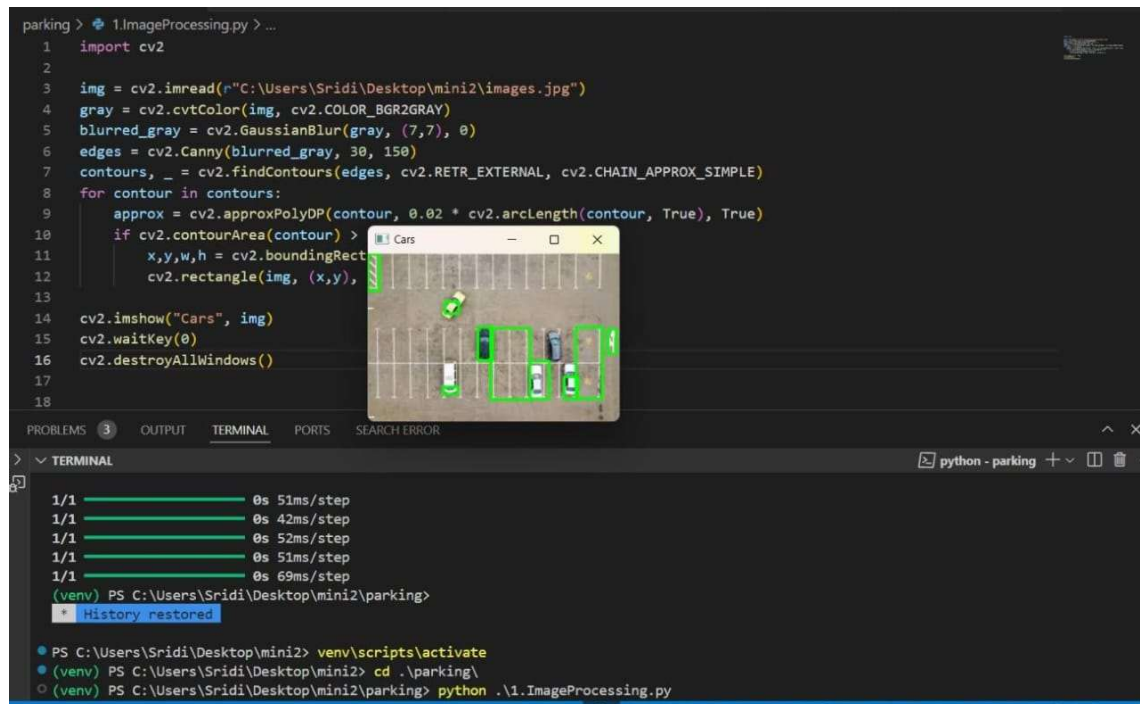
# To run 3.emptyparkingspotdetectionmodel.py
$ python 3.emptyparkingspotdetectionmodel.py

# To run 4.emptyparkingspotdetection.py
$ python 4.emptyparkingspotdetection.py
```

CHAPTER 8

RESULT

8.1 OUTPUT SCREENSHOTS



The screenshot displays a Python script in an IDE (VS Code) titled "1.ImageProcessing.py". The script uses OpenCV to process an image of a parking lot. It reads the image, converts it to grayscale, blurs it, and then applies the Canny edge detection algorithm. Contours are found using `cv2.findContours`, and a bounding box is drawn around each car using `cv2.rectangle`. The resulting image is shown in a window titled "Cars".


```
1 import cv2
2
3 img = cv2.imread(r"C:\Users\Sridi\Desktop\mini2\images.jpg")
4 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5 blurred_gray = cv2.GaussianBlur(gray, (7,7), 0)
6 edges = cv2.Canny(blurred_gray, 30, 150)
7 contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
8 for contour in contours:
9     approx = cv2.approxPolyDP(contour, 0.02 * cv2.arcLength(contour, True), True)
10    if cv2.contourArea(contour) > 1000:
11        x,y,w,h = cv2.boundingRect(contour)
12        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
13
14 cv2.imshow("Cars", img)
15 cv2.waitKey(0)
16 cv2.destroyAllWindows()
17
18
```

The "Cars" window shows a parking lot with several cars. Green bounding boxes are drawn around each car, indicating successful detection. The IDE's terminal shows the command `python .\1.ImageProcessing.py` being executed.

```

parking > 1.ImageProcessing.py > ...
1  import cv2
2
3  img = cv2.imread(r"C:\Users\Sridi\Desktop\mini2\images.jpg")
4  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5  blurred_gray = cv2.GaussianBlur(gray, (7,7), 0)
6  edges = cv2.Canny(blurred_gray, 30, 150)
7  contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
8  for contour in contours:
9      approx = cv2.approxPolyDP(contour, 0.02 * cv2.arcLength(contour, True), True)
10     if cv2.contourArea(approx) > 4:
11         x,y,w,h = cv2.boundingRect(approx)
12         cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
13
14  cv2.imshow("Cars",img)
15  cv2.waitKey(0)
16  cv2.destroyAllWindows()
17
18
19
20

```



```

> TERMINAL
Top Left: (176, 120)
Bottom Right: (191, 162)
Top Left: (195, 117)
Bottom Right: (209, 160)
Top Left: (212, 121)
Bottom Right: (229, 162)
Top Left: (230, 122)
Bottom Right: (251, 159)

```

```

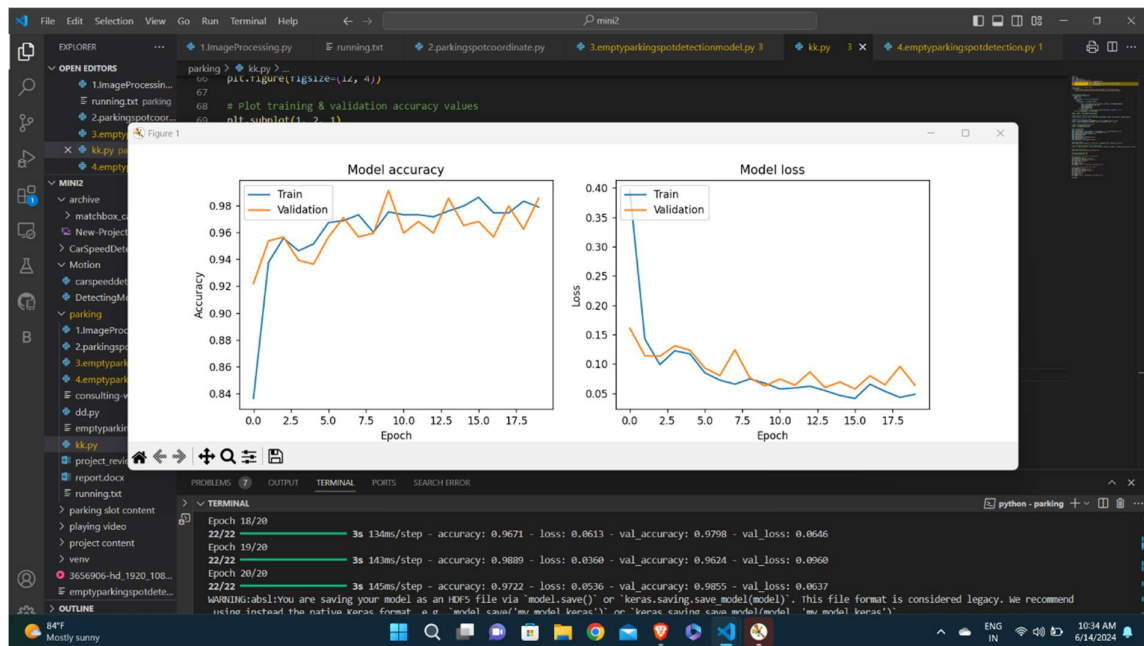
(venv) PS C:\Users\Sridi\Desktop\mini2\parking> python .\2.parkingspotcoordinate.py
Top Left: (7, 0)
Bottom Right: (26, 38)
Top Left: (27, 0)
Bottom Right: (44, 39)
Top Left: (46, 1)
Bottom Right: (63, 38)
Top Left: (63, 0)
Bottom Right: (82, 37)
Top Left: (83, 0)
Bottom Right: (101, 38)
Top Left: (100, 1)
Bottom Right: (119, 39)
Top Left: (121, 1)
Bottom Right: (138, 37)
Top Left: (139, 0)
Bottom Right: (153, 40)
Top Left: (153, 2)
Bottom Right: (172, 40)
Top Left: (174, 1)
Bottom Right: (191, 39)
Top Left: (192, 3)
Bottom Right: (207, 39)
Top Left: (209, 1)
Bottom Right: (226, 41)
Top Left: (228, 0)
Bottom Right: (251, 42)
Top Left: (15, 81)
Bottom Right: (13, 77)
Top Left: (10, 82)
Bottom Right: (28, 117)
Top Left: (29, 80)
Bottom Right: (44, 118)
Top Left: (48, 83)

```



```
KeyboardInterrupt
o (venv) PS C:\Users\Sridi\Desktop\mini2\parking> python .\3.emptyparkingspotdetectionmodel.py
2024-06-07 15:15:29.231750: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round
-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-06-07 15:15:30.690100: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round
-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
c:\Users\Sridi\Desktop\mini2\venv\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a laye
r. When using Sequential models, prefer using an 'Input(shape)' object as the first layer in the model instead.
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-06-07 15:15:35.402458: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critica
l operations.
To enable the following instructions: AVX2 AVXS12F AVXS12_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/20
22/22 ----- 5s 165ms/step - accuracy: 0.6767 - loss: 0.6849 - val_accuracy: 0.9509 - val_loss: 0.1937
Epoch 2/20
22/22 ----- 3s 149ms/step - accuracy: 0.9236 - loss: 0.1867 - val_accuracy: 0.8671 - val_loss: 0.3430
Epoch 3/20
22/22 ----- 3s 150ms/step - accuracy: 0.9232 - loss: 0.1825 - val_accuracy: 0.9191 - val_loss: 0.1715
Epoch 4/20
22/22 ----- 3s 151ms/step - accuracy: 0.9359 - loss: 0.1479 - val_accuracy: 0.9422 - val_loss: 0.1185
Epoch 5/20
22/22 ----- 3s 150ms/step - accuracy: 0.9576 - loss: 0.1112 - val_accuracy: 0.9509 - val_loss: 0.1382
Epoch 6/20
22/22 ----- 3s 151ms/step - accuracy: 0.9503 - loss: 0.0944 - val_accuracy: 0.9595 - val_loss: 0.1015
Epoch 7/20
22/22 ----- 3s 149ms/step - accuracy: 0.9519 - loss: 0.1232 - val_accuracy: 0.9566 - val_loss: 0.1093
Epoch 8/20
22/22 ----- 3s 150ms/step - accuracy: 0.9540 - loss: 0.0865 - val_accuracy: 0.9595 - val_loss: 0.1029
Epoch 9/20
22/22 ----- 3s 153ms/step - accuracy: 0.9560 - loss: 0.0836 - val_accuracy: 0.9566 - val_loss: 0.1013
Epoch 10/20
22/22 ----- 3s 149ms/step - accuracy: 0.9641 - loss: 0.0867 - val_accuracy: 0.9653 - val_loss: 0.0731
Epoch 11/20
22/22 ----- 3s 148ms/step - accuracy: 0.9634 - loss: 0.0720 - val_accuracy: 0.9740 - val_loss: 0.0722
Epoch 12/20
22/22 ----- 3s 155ms/step - accuracy: 0.9711 - loss: 0.0676 - val_accuracy: 0.9711 - val_loss: 0.0639
Epoch 13/20
22/22 ----- 3s 152ms/step - accuracy: 0.9660 - loss: 0.0658 - val_accuracy: 0.9682 - val_loss: 0.0748
Epoch 14/20
22/22 ----- 4s 158ms/step - accuracy: 0.9654 - loss: 0.0693 - val_accuracy: 0.9509 - val_loss: 0.0830
Epoch 15/20
1/22 ----- 4s 221ms/step - accuracy: 0.9219 - loss: 0.1008
```

```
o TERMINAL
To enable the following instructions: AVX2 AVXS12F AVXS12_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
1/1 ----- 0s 83ms/step
1/1 ----- 0s 38ms/step
1/1 ----- 0s 35ms/step
1/1 ----- 0s 44ms/step
1/1 ----- 0s 45ms/step
1/1 ----- 0s 34ms/step
1/1 ----- 0s 41ms/step
1/1 ----- 0s 39ms/step
1/1 ----- 0s 46ms/step
1/1 ----- 0s 40ms/step
1/1 ----- 0s 39ms/step
1/1 ----- 0s 40ms/step
1/1 ----- 0s 40ms/step
1/1 ----- 0s 41ms/step
1/1 ----- 0s 39ms/step
1/1 ----- 0s 46ms/step
1/1 ----- 0s 39ms/step
1/1 ----- 0s 46ms/step
1/1 ----- 0s 41ms/step
1/1 ----- 0s 37ms/step
1/1 ----- 0s 41ms/step
1/1 ----- 0s 44ms/step
1/1 ----- 0s 37ms/step
1/1 ----- 0s 45ms/step
1/1 ----- 0s 35ms/step
1/1 ----- 0s 35ms/step
1/1 ----- 0s 40ms/step
1/1 ----- 0s 36ms/step
1/1 ----- 0s 36ms/step
1/1 ----- 0s 38ms/step
1/1 ----- 0s 43ms/step
1/1 ----- 0s 37ms/step
1/1 ----- 0s 45ms/step
1/1 ----- 0s 39ms/step
1/1 ----- 0s 37ms/step
1/1 ----- 0s 30ms/step
1/1 ----- 0s 42ms/step
1/1 ----- 0s 37ms/step
1/1 ----- 0s 42ms/step
1/1 ----- 0s 43ms/step
```



CHAPTER 9

9.1 CONCLUSION

In conclusion, the project aims to address the challenge of efficient parking space management by leveraging computer vision and machine learning techniques to detect empty parking spots in real-time. Through the development of a convolutional neural network (CNN) model trained on a dataset of parking slot images, the system can accurately identify and monitor vacant parking spaces, providing valuable insights for optimizing parking allocation and improving overall parking efficiency.

The implementation process involved various stages, including data collection, preprocessing, model training, and real-time detection. By carefully designing the model architecture, optimizing training parameters, and integrating the system with live camera feeds, we have created a robust and scalable solution capable of detecting empty parking spots with high accuracy and reliability.

9.2 FUTURE SCOPE

The automated parking slot detection system can significantly contribute to the development of smart cities by integrating with urban management systems to reduce traffic congestion and enhance parking efficiency. The scalability of the system allows it to be deployed in multi-level parking structures and across entire cities, making it a versatile solution. Future applications could include the development of mobile apps and in-car systems that provide real-time parking availability to drivers, improving user experience. Additionally, the system can generate valuable data insights, helping city planners optimize parking resource management. Advanced features like license plate recognition and dynamic pricing based on demand could be integrated, further enhancing its utility. By reducing the time drivers spend searching for parking, the system can also lower vehicle emissions, contributing to environmental sustainability. Finally, the integration with Internet of Things (IoT) devices and preparation for use with autonomous vehicles will ensure the system remains relevant as technology evolves.

CHAPTER 10

BIBLIOGRAPHY & REFERENCES

10.1 BIBLIOGRAPHY

1 .Automated Vehicle Parking Slot Detection System Using Deep Learning

IEEE Xplore Part Number:CFP20K25-ART;

ISBN:978-1-7281-4889-2

2.Automated parking space detection using convolutional neural networks

10.2 REFERENCES

<https://ieeexplore.ieee.org/document/8221062>

<https://ieeexplore.ieee.org/document/8261114>