Name: **GANESH SANJAY PANDHRE**

Roll Nos : **41** | Div: **D20B**

Aim: To build a Cognitive based application to acquire knowledge through images for a Customer service application/ Insurance/ Healthcare Application/ Smarter Cities/Government etc.

Experiment No: 3

Theory:

Introduction

Cognitive computing refers to systems that can **mimic human intelligence** by using Artificial Intelligence (AI), Machine Learning (ML), and Computer Vision. In this experiment, we focus on **building a cognitive-based application that learns from images** and provides useful knowledge. Such applications are used in **customer service, insurance, healthcare, smarter cities, and government sectors**.

In our case, we continue with the topic from the previous experiment — **Apartment Assistant** — and extend its features by adding **vehicle license plate detection**. This makes the system smarter and more useful for apartment security and management.

Cognitive Applications and Their Role

A **cognitive application** is an AI system that can:

1. **Understand** data (images, text, speech, etc.).
2. **Reason** based on the patterns it finds.
3. **Learn** continuously from new data.
4. **Assist** in decision-making.

In this experiment, the input is **images**, and the system tries to extract **knowledge** from them. Example use cases:

- Recognizing **visitors** entering the apartment.
- Detecting and reading **vehicle number plates** for parking and security.
- Extracting **text from bills or notices** using OCR.

Concepts Used in This Experiment

1. **Image Classification (Computer Vision)**
    - Deep learning models like **ResNet50** are used to identify objects from images.
    - The model recognizes whether the image contains a person, a car, or other objects.
2. **Optical Character Recognition (OCR)**
    - OCR is used to extract text from images (bills, notices, or even vehicle number plates).
    - It converts printed or handwritten text into digital form for easy processing.
3. **Vehicle License Plate Detection**
    - A key new addition in this experiment is the **ability to detect and read vehicle plates**.
    - This is important for apartments as it can **automate vehicle entry and parking management**.
    - The system captures a car's image, detects the plate, and extracts the alphanumeric characters.
    - This can be used for **security verification**, **visitor vehicle logging**, and **automatic gate access**.
4. **Knowledge Acquisition from Images**

- Beyond detection, the system interprets the extracted data to make it useful.
- Example: If the license plate number is recognized, it can be checked against the apartment's database of registered vehicles.

Applications in Apartment Assistant

By extending the Apartment Assistant with **vehicle plate detection**, the system becomes more practical:

- **Visitor Identification** → Security can identify and log visitors.
- **Vehicle Management** → Detect cars entering, check if they are registered, and allow/deny access automatically.
- **Billing and Notices** → OCR extracts details from electricity bills, maintenance slips, or apartment notices.

Working of the Experiment

1. **Input** → User uploads an image (visitor photo, car image, or bill).
2. **Processing** →
   - Deep learning model identifies objects.
   - OCR extracts text (from number plates or documents).
3. **Output** →
   - Recognized object category (Car, Person, Document).
   - Extracted license plate text or bill details.
   - Useful knowledge like identifying whether a vehicle is registered or not.

Advantages of Apartment Assistant with Vehicle Plate Detection

- Provides **better security** by tracking vehicles entering the apartment.
- Saves **manual effort** of security guards for logging visitor cars.
- Helps in **automation of parking management**.
- Can be combined with other features (face recognition, bill scanning) to form a **complete smart apartment solution**.

```
import os
import cv2
import shutil
import numpy as np
import pandas as pd
from glob import glob
import matplotlib.pyplot as plt
import xml.etree.ElementTree as xet
from sklearn.model_selection import train_test_split
```

```
import os
import cv2
import shutil
import numpy as np
import pandas as pd
from glob import glob
import matplotlib.pyplot as plt
import xml.etree.ElementTree as xet
from sklearn.model_selection import train_test_split



import torch

print(f'{torch.cuda.is_available() = }')
print(f'{torch.cuda.device_count() = }')
```

```
torch.cuda.is_available() = True
torch.cuda.device_count() = 1
```

```
!pip install ultralytics
```

Show hidden output

```
!pip install -U ipywidgets
```

Show hidden output

```
from google.colab import files
files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 67 bytes, last modified: 8/30/2025 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"robstark143","key":"cfc32d778a754be45dee9d7207d5eca4"}'}
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d andrewmvd/car-plate-detection
```

```
Dataset URL: https://www.kaggle.com/datasets/andrewmvd/car-plate-detection
License(s): CC0-1.0
Downloading car-plate-detection.zip to /content
 57% 116M/203M [00:00<00:00, 1.18GB/s]
100% 203M/203M [00:00<00:00, 800MB/s]
```

```
!unzip car-plate-detection.zip -d car-plate-detection
```

Show hidden output

```
dataset_path = '/content/car-plate-detection'
```

```
import re

def the_number_in_the_string(filename):
    """
    Extracts the first sequence of digits from the given filename string and returns it as an integer.
    If no digits are found, returns 0.

    Parameters:
    filename (str): The input string to search for digits.

    Returns:
    int: The first sequence of digits found in the input string, or 0 if no digits are found.
    """
    # Search for the first occurrence of one or more digits in the filename
    match = re.search(r'(\d+)', filename)

    # If a match is found, return the matched number as an integer
    if match:
        return int(match.group(0))
    # If no match is found, return 0
    else:
        return 0

# Example usage
```

```
print(the_number_in_the_string("file123.txt"))  # Output: 123
print(the_number_in_the_string("no_numbers_here"))  # Output: 0
```

```
123
0
```

```
import os
import cv2
import pandas as pd
import xml.etree.ElementTree as xet
from glob import glob

# Initialize a dictionary to store labels and image information
labels_dict = dict(
    img_path=[],
    xmin=[],
    xmax=[],
    ymin=[],
    ymax=[],
    img_w=[],
    img_h=[]
)

# Get the list of XML files from the annotations directory
xml_files = glob(f'{dataset_path}/annotations/*.xml')

# Process each XML file, sorted by the numerical value in the filename
for filename in sorted(xml_files, key=the_number_in_the_string):
    # Parse the XML file
    info = xet.parse(filename)
    root = info.getroot()

    # Find the 'object' element in the XML and extract bounding box information
    member_object = root.find('object')
    labels_info = member_object.find('bndbox')
    xmin = int(labels_info.find('xmin').text)
    xmax = int(labels_info.find('xmax').text)
    ymin = int(labels_info.find('ymin').text)
    ymax = int(labels_info.find('ymax').text)

    # Get the image filename and construct the full path to the image
    img_name = root.find('filename').text
    img_path = os.path.join(dataset_path, 'images', img_name)

    # Append the extracted information to the respective lists in the dictionary
    labels_dict['img_path'].append(img_path)
    labels_dict['xmin'].append(xmin)
    labels_dict['xmax'].append(xmax)
    labels_dict['ymin'].append(ymin)
    labels_dict['ymax'].append(ymax)

    # Read the image to get its dimensions
    height, width, _ = cv2.imread(img_path).shape
    labels_dict['img_w'].append(width)
    labels_dict['img_h'].append(height)

# Convert the dictionary to a pandas DataFrame
alldata = pd.DataFrame(labels_dict)

# Display the DataFrame
alldata
```

	img_path	xmin	xmax	ymin	ymax	img_w	img_h
0	/content/car-plate-detection/images/Cars0.png	226	419	125	173	500	268
1	/content/car-plate-detection/images/Cars1.png	134	262	128	160	400	248
2	/content/car-plate-detection/images/Cars2.png	229	270	176	193	400	400
3	/content/car-plate-detection/images/Cars3.png	142	261	128	157	400	225
4	/content/car-plate-detection/images/Cars4.png	156	503	82	253	590	350
...	...	...	...	...	...	...	...
428	/content/car-plate-detection/images/Cars428.png	142	258	128	157	400	225
429	/content/car-plate-detection/images/Cars429.png	86	208	166	195	301	400
430	/content/car-plate-detection/images/Cars430.png	38	116	159	197	400	225
431	/content/car-plate-detection/images/Cars431.png	55	343	82	147	400	192
432	/content/car-plate-detection/images/Cars432.png	95	196	258	284	467	300

433 rows × 7 columns

Next steps: Generate code with alldata | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
# Use 10% of the data for the test set
train, test = train_test_split(alldata, test_size=1/10, random_state=42)

# Split the training data further into training and validation sets
# Use 8/9 of the remaining data for the training set, resulting in an 80/10/10 split overall
train, val = train_test_split(train, train_size=8/9, random_state=42)

# Print the number of samples in each set
print(f'''
      len(train) = {len(train)}
      len(val) = {len(val)}
      len(test) = {len(test)}
''')
```

```
      len(train) = 345
      len(val) = 44
      len(test) = 44
```

```
import os
import shutil
import pandas as pd

# Remove the 'datasets' directory if it exists
if os.path.exists('datasets'):
    shutil.rmtree('datasets')
```

```
def make_split_folder_in_yolo_format(split_name, split_df):
    """
    Creates a folder structure for a dataset split (train/val/test) in YOLO format.

    Parameters:
    split_name (str): The name of the split (e.g., 'train', 'val', 'test').
    split_df (pd.DataFrame): The DataFrame containing the data for the split.

    The function will create 'labels' and 'images' subdirectories under 'datasets/cars_license_plate/{split_name}',
```

```
    and save the corresponding labels and images in YOLO format.
    """
    labels_path = os.path.join('datasets', 'cars_license_plate_new', split_name, 'labels')
    images_path = os.path.join('datasets', 'cars_license_plate_new', split_name, 'images')

    # Create directories for labels and images
    os.makedirs(labels_path)
    os.makedirs(images_path)

    # Iterate over each row in the DataFrame
    for _, row in split_df.iterrows():
        img_name, img_extension = os.path.splitext(os.path.basename(row['img_path']))

        # Calculate YOLO format bounding box coordinates
        x_center = (row['xmin'] + row['xmax']) / 2 / row['img_w']
        y_center = (row['ymin'] + row['ymax']) / 2 / row['img_h']
        width = (row['xmax'] - row['xmin']) / row['img_w']
        height = (row['ymax'] - row['ymin']) / row['img_h']

        # Save the label in YOLO format
        label_path = os.path.join(labels_path, f'{img_name}.txt')
        with open(label_path, 'w') as file:
            file.write(f"0 {x_center:.4f} {y_center:.4f} {width:.4f} {height:.4f}\n")

        # Copy the image to the images directory
        shutil.copy(row['img_path'], os.path.join(images_path, img_name + img_extension))

    print(f"Created '{images_path}' and '{labels_path}'")
```

```
# Create YOLO format folders for train, validation, and test splits
make_split_folder_in_yolo_format("train", train)
make_split_folder_in_yolo_format("val", val)
make_split_folder_in_yolo_format("test", test)
```

```
Created 'datasets/cars_license_plate_new/train/images' and 'datasets/cars_license_plate_new/train/labels'
Created 'datasets/cars_license_plate_new/val/images' and 'datasets/cars_license_plate_new/val/labels'
Created 'datasets/cars_license_plate_new/test/images' and 'datasets/cars_license_plate_new/test/labels'
```

```
os.getcwd()
```

```
'/content'
```

```
import os
import cv2
import matplotlib.pyplot as plt

# Directory paths
image_dir = 'datasets/cars_license_plate_new/train/images'
label_dir = 'datasets/cars_license_plate_new/train/labels'

# Get the first image file
image_files = sorted(os.listdir(image_dir))
first_image_file = image_files[0]

# Construct paths for the image and its corresponding label
image_path = os.path.join(image_dir, first_image_file)
label_path = os.path.join(label_dir, os.path.splitext(first_image_file)[0] + '.txt')

# Load the image using OpenCV
image = cv2.imread(image_path)
# Convert the image from BGR (OpenCV default) to RGB (matplotlib default)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Read the label file to get bounding box information
with open(label_path, 'r') as f:
```

```
    lines = f.readlines()

# Plot the bounding box on the image
for line in lines:
    # Parse the label file line to extract bounding box information
    class_id, x_center, y_center, width, height = map(float, line.strip().split())
    img_height, img_width, _ = image.shape

    # Convert YOLO format to bounding box format
    x_center *= img_width
    y_center *= img_height
    width *= img_width
    height *= img_height

    # Calculate the top-left and bottom-right coordinates of the bounding box
    x1 = int(x_center - width / 2)
    y1 = int(y_center - height / 2)
    x2 = int(x_center + width / 2)
    y2 = int(y_center + height / 2)

    # Draw the bounding box on the image using a green rectangle
    cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Display the image with bounding box using matplotlib
plt.imshow(image)
plt.axis('off')  # Hide the axis
plt.show()  # Display the image
```


```
# Define the content of the datasets.yaml file
datasets_yaml = '''
path: cars_license_plate_new

train: train/images
val: val/images
test: test/images

# number of classes
nc: 1

# class names
names: ['license_plate']
'''

# Write the content to the datasets.yaml file
with open('datasets.yaml', 'w') as file:
    file.write(datasets_yaml)
```

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')
```

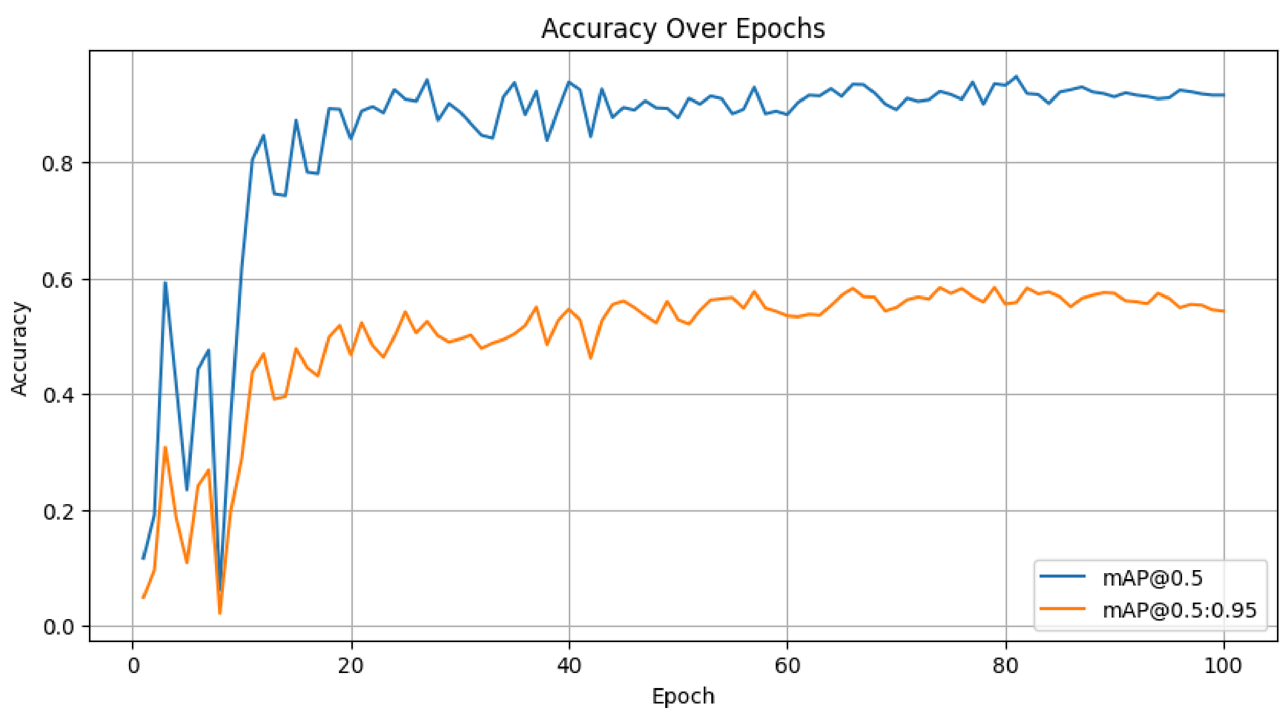
```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8n.pt to 'yolov8n.pt': 100% ━━━━━━━━━
```

```
model.train(
    data='datasets.yaml',  # Path to the dataset configuration file
    epochs=100,            # Number of training epochs
    batch=16,              # Batch size
    device='cuda',         # Use GPU for training
    imgsz=320,             # Image size (width and height) for training
    cache=True             # Cache images for faster training
)
```

```
Ultralytics 8.3.189 🚀 Python-3.12.11 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=16, bgr=0.0, box=7.5,
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf': 100% ━━━━━━━━━━━━ 75
Overriding model.yaml nc=80 with nc=1

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22        [15, 18, 21]  1    751507  ultralytics.nn.modules.head.Detect           [1, [64, 128, 256]]
Model summary: 129 layers, 3,011,043 parameters, 3,011,027 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks...
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to 'yolo11n.pt': 100% ━━━━━━━━
AMP: checks passed ✅
train: Fast image access ✅ (ping: 0.0±0.0 ms, read: 2208.3±1283.1 MB/s, size: 607.4 KB)
train: Scanning /content/datasets/cars_license_plate_new/train/labels... 345 images, 0 backgrounds, 0 corrupt: 100%
train: New cache created: /content/datasets/cars_license_plate_new/train/labels.cache
WARNING ⚠️ cache='ram' may produce non-deterministic training results. Consider cache='disk' as a deterministic alt
train: Caching images (0.1GB RAM): 100% ━━━━━━━━━━━━ 345/345 58.8it/s 5.9s
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weig
val: Fast image access ✅ (ping: 0.0±0.0 ms, read: 1154.2±1237.2 MB/s, size: 402.1 KB)
val: Scanning /content/datasets/cars_license_plate_new/val/labels... 44 images, 0 backgrounds, 0 corrupt: 100% ━━━━
val: New cache created: /content/datasets/cars_license_plate_new/val/labels.cache
WARNING ⚠️ cache='ram' may produce non-deterministic training results. Consider cache='disk' as a deterministic alt
val: Caching images (0.0GB RAM): 100% ━━━━━━━━━━━━ 44/44 72.4it/s 0.6s
Plotting labels to runs/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0'
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bi
Image sizes 320 train, 320 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 100 epochs...
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100     0.594G      1.758      3.007      1.347         11        320: 100% ━━━━━━━━━━━━ 22/22 4.0it/s 5.4s
```

```
import os
import pandas as pd
import matplotlib.pyplot as plt
from glob import glob


# Find the most recent training log directory
log_dir = max(glob('runs/detect/train*'), key=the_number_in_the_string)

# Load the training results from the CSV file
results = pd.read_csv(os.path.join(log_dir, 'results.csv'))
results.columns = results.columns.str.strip()  # Remove any leading/trailing whitespace from column names

# Extract epochs and accuracy metrics
epochs = results.index + 1  # Epochs are zero-indexed, so add 1
mAP_0_5 = results['metrics/mAP50(B)']  # Mean Average Precision at IoU=0.5
mAP_0_5_0_95 = results['metrics/mAP50-95(B)']  # Mean Average Precision at IoU=0.5:0.95

# Plot the accuracy over epochs
plt.figure(figsize=(10, 5))
plt.plot(epochs, mAP_0_5, label='mAP@0.5')
plt.plot(epochs, mAP_0_5_0_95, label='mAP@0.5:0.95')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy Over Epochs')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Save the trained model
model.save('best_license_plate_model.pt')
```

```
import cv2
import matplotlib.pyplot as plt
from ultralytics import YOLO
```

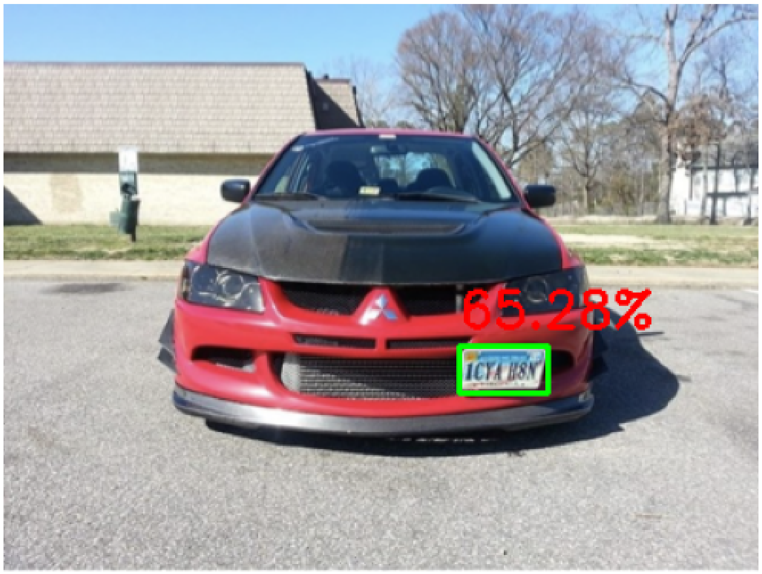
```
def predict_and_plot(path_test_car):
    """
    Predicts and plots the bounding boxes on the given test image using the trained YOLO model.

    Parameters:
    path_test_car (str): Path to the test image file.
    """
    # Perform prediction on the test image using the model
    results = model.predict(path_test_car, device='cpu')

    # Load the image using OpenCV
    image = cv2.imread(path_test_car)
    # Convert the image from BGR (OpenCV default) to RGB (matplotlib default)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Extract the bounding boxes and labels from the results
    for result in results:
        for box in result.boxes:
            # Get the coordinates of the bounding box
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            # Get the confidence score of the prediction
            confidence = box.conf[0]

            # Draw the bounding box on the image
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
            # Draw the confidence score near the bounding box
            cv2.putText(image, f'{confidence*100:.2f}%', (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

    # Plot the image with bounding boxes
    plt.imshow(image)
    plt.axis('off')  # Hide the axis
    plt.show()  # Display the image
```

```
predict_and_plot(test.iloc[0].img_path)
```

```
image 1/1 /content/car-plate-detection/images/Cars425.png: 160x320 1 license_plate, 47.8ms
Speed: 0.7ms preprocess, 47.8ms inference, 24.9ms postprocess per image at shape (1, 3, 160, 320)
```


```
predict_and_plot(test.iloc[15].img_path)
```

```
image 1/1 /content/car-plate-detection/images/Cars395.png: 256x320 1 license_plate, 70.7ms
Speed: 1.2ms preprocess, 70.7ms inference, 0.9ms postprocess per image at shape (1, 3, 256, 320)
```




```
predict_and_plot(test.iloc[18].img_path)
```

```
image 1/1 /content/car-plate-detection/images/Cars9.png: 256x320 1 license_plate, 79.0ms
Speed: 1.4ms preprocess, 79.0ms inference, 0.9ms postprocess per image at shape (1, 3, 256, 320)
```


```
!pip3 install pytesseract
```

```
Collecting pytesseract
  Downloading pytesseract-0.3.13-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.12/dist-packages (from pytesseract) (25.0)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from pytesseract) (11.3.0)
Downloading pytesseract-0.3.13-py3-none-any.whl (14 kB)
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.13
```

```
import pytesseract
from pytesseract import Output
```

```
import cv2
import matplotlib.pyplot as plt
from ultralytics import YOLO
import pytesseract
from pytesseract import Output


def predict_and_plot(path_test_car):
    """
    Predicts and plots the bounding boxes on the given test image using the trained YOLO model.
    Also performs OCR on the detected bounding boxes to extract text.

    Parameters:
    path_test_car (str): Path to the test image file.
    """
    # Perform prediction on the test image using the model
    results = model.predict(path_test_car, device='cpu')

    # Load the image using OpenCV
    image = cv2.imread(path_test_car)
    # Convert the image from BGR (OpenCV default) to RGB (matplotlib default)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Extract the bounding boxes and labels from the results
    for result in results:
        for box in result.boxes:
            # Get the coordinates of the bounding box
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            # Get the confidence score of the prediction
            confidence = box.conf[0]

            # Draw the bounding box on the image
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
            # Draw the confidence score near the bounding box
            cv2.putText(image, f'{confidence*100:.2f}%', (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

            # Crop the bounding box from the image for OCR
            roi = image[y1:y2, x1:x2]

            # Perform OCR on the cropped image
            text = pytesseract.image_to_string(roi, config='--psm 6')
            print(f"Detected text: {text}")

    # Plot the image with bounding boxes
    plt.imshow(image)
    plt.axis('off')  # Hide the axis
    plt.show()  # Display the image
```

```
predict_and_plot(test.iloc[0].img_path)
```

```
image 1/1 /content/car-plate-detection/images/Cars425.png: 160x320 1 license_plate, 55.3ms
Speed: 2.7ms preprocess, 55.3ms inference, 1.2ms postprocess per image at shape (1, 3, 160, 320)
Detected text: G526 JHD
```


```python
predict_and_plot(test.iloc[2].img_path)
```

```
image 1/1 /content/car-plate-detection/images/Cars181.png: 320x192 1 license_plate, 46.9ms
Speed: 0.9ms preprocess, 46.9ms inference, 0.9ms postprocess per image at shape (1, 3, 320, 192)
Detected text: SHIT HAD
```


```python
predict_and_plot(test.iloc[25].img_path)
```

```
image 1/1 /content/car-plate-detection/images/Cars198.png: 256x320 1 license_plate, 54.7ms
Speed: 1.1ms preprocess, 54.7ms inference, 0.9ms postprocess per image at shape (1, 3, 256, 320)
Detected text: ~-MHO1AV8866)
```

Conclusion:

This experiment demonstrates how **cognitive image-based applications** can enhance the **Apartment Assistant** system. By combining **image classification, OCR, and license plate detection**, the system can acquire knowledge from real-world apartment-related images. The addition of **vehicle plate detection** improves apartment security and makes parking and visitor management more efficient. Such applications show the power of cognitive computing in **real-life problem-solving** for smarter living spaces.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.