"Object Detection with Python“

Using OpenCV & YOLO

(Step By Step Tutorial)

Pravin Varak

Introduction to Object Detection

"Object detection is a computer vision technique used to identify and locate objects within images or videos. It involves two primary tasks: localizing objects within an image (drawing bounding boxes around them) and classifying the type of objects present."

Object Detection Applications

1. **Autonomous Vehicles:** Object detection is crucial for vehicles to recognize pedestrians, other vehicles, traffic signs, and obstacles on the road for safe navigation.
2. **Surveillance and Security:** Identifying people, intruders, or suspicious activities in surveillance footage helps enhance security.
3. **Retail**: Retailers use object detection for inventory management, tracking products on shelves, and analysing customer behaviour in stores.
4. **Medical Imaging**: Object detection assists in analysing medical images, locating anomalies, tumours, or other medical conditions.
5. **Augmented Reality**: Recognizing and tracking objects in real-time facilitates AR applications, enhancing user experiences.
6. **Industrial Automation**: Object detection is used in quality control, robotics, and monitoring production lines.

"Object detection continues to evolve, with ongoing research focusing on improving accuracy, speed, and applicability across various domains."

List of Prerequisites

- Python basics
- Required libraries (OpenCV, NumPy, MatPlotlib.Pyplot)
- Installation commands (pip install opencv-python numpy)

Programming Environment

- To set the Programming environment we have to choose An Integrated Development Environment (**IDE**).
- **IDE** is a comprehensive software application that consolidates various tools essential for software development into a single integrated platform. It aims to enhance the efficiency and productivity of developers by providing a unified environment for writing, testing, and debugging code.
- To name few popular IDE; **Visual Studio Code, Spyder, Jupiter Note Book, PyCharm, Atom, Eclipse etc.**
- Most popular IDE used for Object Detection is Google Colab is a cloud-based platform provided by Google that offers free access to GPU and TPU resources.
- I have used Spyder IDE.
- We also need to download YOLO weights and configuration file from the official YOLO website:

https://pjreddie.com/darknet/yolo/

Code for Object Detection

1. Code snippet for importing libraries & loading YOLO model

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Load pretrained YOLO Model


net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

classes = []

with open("coco.names", "r") as f:

    classes = [line.strip() for line in f.readlines()]
```


```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Load pretrained YOLO Model

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
```


Name	Type	Size	Value
classes	list	80	['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', ...
f	TextIOWrapper	1	TextIOWrapper object of _io module
net	dnn.Net	1	Net object of cv2.dnn module

2. Code Snippet for Output Layers of the YOLO Network

```
layer_names = net.getUnconnectedOutLayersNames()
```

"This section gets the output layers of the YOLO network. YOLO has some output

layers that give predictions at different scales. This code retrieves those

output layers."

layer_names	tuple	3	('yolo_82', 'yolo_94', 'yolo_106')

layer_names = net.getUnconnectedOutLayersNames()

3. Code snippet for loading and pre-processing an image

Load image

image = cv2.imread("pexels-pixabay-52500.jpg")

height, width, channels = image.shape

Preprocess Image

blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(layer_names)

```
# Load image

image = cv2.imread("pexels-pixabay-52500.jpg")


"""This part reads an image ("sample_image.jpg") using OpenCV's imread() function
and extracts its height, width, and number of channels (RGB channels)."""

height, width, channels = image.shape



# Preprocess Image

blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(layer_names)
```

Input Image

4. Code snippet for extracting class IDs, confidences, and bounding boxes

Get class IDs, confidences, and bounding boxes

class_ids = []

confidences = []

boxes = []

#Code continues on next slide.....

```
# Get class IDs, confidences, and bounding boxes

class_ids = []
confidences = []
boxes = []

for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:  # Confidence threshold
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

"""The above section processes the detections obtained from the YOLO network.
It iterates through the detections and filters out low-confidence detections
(confidence threshold set at 0.5). For each detected object, it extracts the
class ID, confidence, and bounding box coordinates."""
```

```
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:  # Confidence threshold
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)
```

5. Code snippet for non-maximum suppression

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

"""Non-Maximum Suppression (cv2.dnn.NMSBoxes()) is applied to eliminate overlapping bounding boxes, keeping only the most confident ones."""

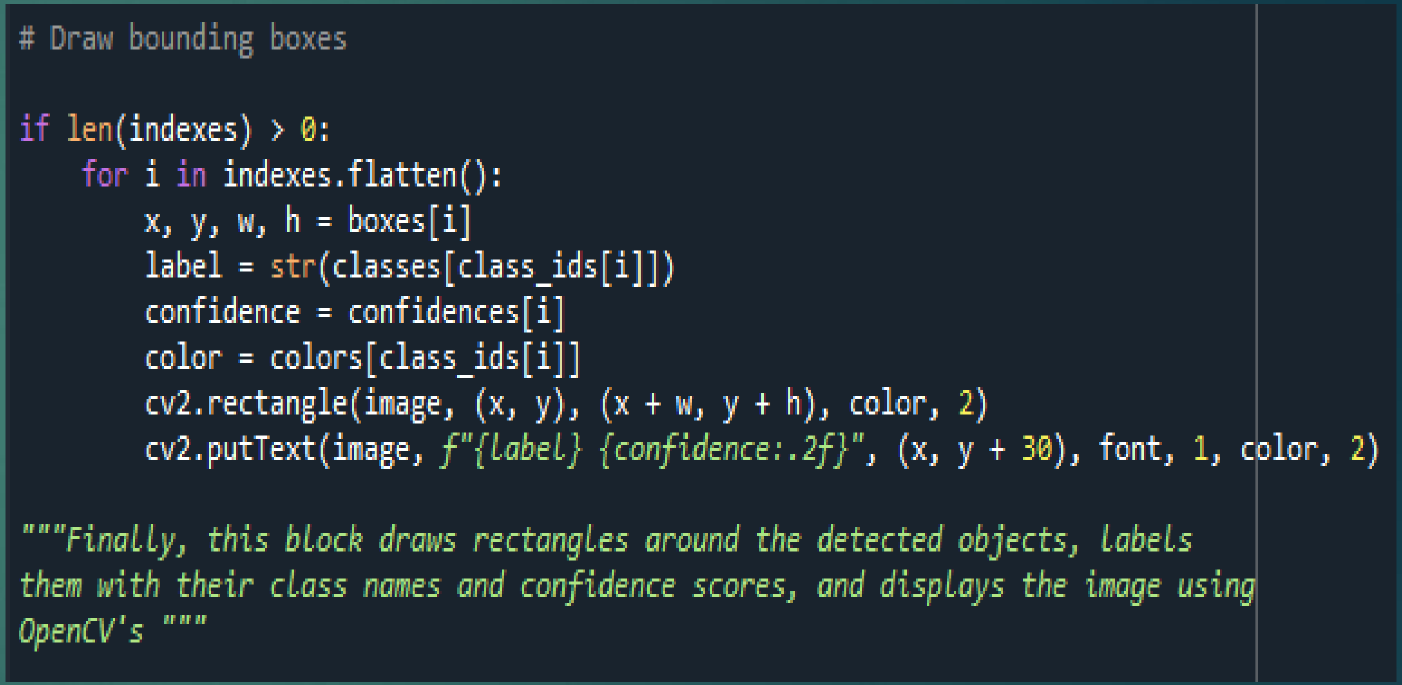
```python
# Non-maximum suppression to remove overlapping bounding boxes

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

"""Non-Maximum Suppression (cv2.dnn.NMSBoxes()) is applied to eliminate
overlapping bounding boxes, keeping only the most confident ones."""
```

6. Code snippet for Font Style

```
# Choose font style

font = cv2.FONT_HERSHEY_PLAIN

colors = np.random.uniform(0, 255, size=(len(classes), 3))

"""Here, a font style is chosen for the labels, and random colours are generated for each class."""
```


```
# Choose font style

font = cv2.FONT_HERSHEY_PLAIN
colors = np.random.uniform(0, 255, size=(len(classes), 3))

"""Here, a font style is chosen for the labels, and random colours are generated
for each class."""
```

7. Code snippet for To Draw bounding boxes



```python
# Draw bounding boxes

if len(indexes) > 0:
    for i in indexes.flatten():
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = colors[class_ids[i]]
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        cv2.putText(image, f"{label} {confidence:.2f}", (x, y + 30), font, 1, color, 2)

"""Finally, this block draws rectangles around the detected objects, labels
them with their class names and confidence scores, and displays the image using
OpenCV's """
```

```python
if len(indexes) > 0:
    for i in indexes.flatten():
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = colors[class_ids[i]]
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        cv2.putText(image, f"{label} {confidence:.2f}", (x, y + 30), font, 1, color, 2)
```

8. Code snippet for displaying the result

```
# Display the result

cv2.imshow("Object Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

plt.imshow(image)
```


```
# Display the result

cv2.imshow("Object Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()


plt.imshow(image)
```

Output Image:

We can see the boxes around horses with labels in top left corners.

Model Adjustment to Different Objects, Images or Videos

- This pre-trained model will have to be adjusted to different objects, images or videos to recognize specific classes relevant to particular use case.
- Annotate the image or video with bounding boxes around the objects of interest and label each bounding box with the corresponding class.
- Open the YOLO configuration file (e.g., "yolov3.cfg") and modify the ‘classes’ parameter in the ‘[net]’ section to match the number of classes in our custom dataset.
- Optionally, update the ‘anchors’ parameter based on the aspect ratios of objects in our dataset.
- We also might need to update our class labels file.
- We can also fine tune our model if we have large & diverse data set.

Conclusion

This script reads an image, detects objects using the YOLO model, and displays the result with bounding boxes and labels.

Potential Next Steps

This model with slight modification can be used for various other images, video inputs & real time web cam detection.