**YOLOv3 Object Detection using OpenCV**

This code uses the YOLO (You Only Look Once) object detection model to identify objects in an image. Below is a detailed breakdown of each section of the code:

**1. Import Libraries**

import cv2

import numpy as np

* **cv2**: OpenCV library for image processing and computer vision tasks.
* **numpy**: Library for handling arrays and numerical operations.

**2. Download YOLO Files**

!wget -O yolov3.weights https://pjreddie.com/media/files/yolov3.weights

!wget -O yolov3.cfg https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg

!wget -O coco.names https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names

* Downloads the YOLOv3 **weights** (yolov3.weights), **config file** (yolov3.cfg), and **class labels** (coco.names) from the web.
  + **yolov3.weights**: Contains the trained weights for the YOLO model.
  + **yolov3.cfg**: Contains the architecture configuration for the YOLOv3 model.
  + **coco.names**: A list of class labels that YOLOv3 can detect (e.g., "person", "car", "dog").

**3. Load YOLO Network**

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

* **cv2.dnn.readNet**: Reads the YOLO model configuration and weights files to load the neural network into the variable net.

**4. Load Class Names**

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

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

* Loads the class labels (from the coco.names file) into a list called classes. These are the names of the objects that YOLO can detect.

**5. Get Layer Names**

layer\_names = net.getLayerNames()

* **net.getLayerNames()**: Returns the names of all layers in the YOLO network. These are used to retrieve output layers where detections will occur.

**6. Get Output Layers**

outputlayers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

* **net.getUnconnectedOutLayers()**: This function returns the indexes of the layers that produce outputs (predictions) from the network.
* The -1 is used because OpenCV indexes layers starting from 1.

**7. Generate Colors for Visualization**

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

* Generates a random color for each class in the dataset using NumPy, which will be used to draw bounding boxes around detected objects.

**8. Load and Resize Image**

img = cv2.imread("/content/istockphoto-1201306396-612x612.jpg")

img = cv2.resize(img, None, fx=0.4, fy=0.3)

height, width, channels = img.shape

* **cv2.imread()**: Reads the input image from the specified path.
* **cv2.resize()**: Resizes the image by scaling it down to 40% width and 30% height.
* **img.shape**: Returns the dimensions of the image (height, width, channels).

**9. Prepare the Image for Detection**

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

* **cv2.dnn.blobFromImage()**: Prepares the image to be passed into the YOLO model:
  + Scales pixel values by 0.00392 (1/255) to normalize the image.
  + Resizes the image to 416x416 pixels (a standard input size for YOLO).
  + Normalizes the color channels (mean subtraction (0,0,0)).
  + **True** means that the image is cropped if necessary (not used here).

**10. Forward Pass and Get Output**

net.setInput(blob)

outs = net.forward(outputlayers)

* **net.setInput(blob)**: Sets the input image (blob) for the YOLO model.
* **net.forward(outputlayers)**: Performs a forward pass through the network to get the output from the detected objects.

**11. Process the Detections**

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:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

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)

* **detection**: The output from the YOLO network, which contains information about each detected object.
* **scores**: Confidence scores for each class (starting from index 5 because the first 5 values in each detection represent bounding box parameters).
* **np.argmax(scores)**: Finds the index of the class with the highest score (i.e., the predicted class).
* **confidence**: The confidence level for the detection. If it is higher than 50% (0.5), we consider the object detected.
* **Bounding Box**: For each object, the center and width/height of the bounding box are computed, and the coordinates of the top-left corner are stored in boxes.

**12. Non-Maximum Suppression (NMS)**

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.4, 0.6)

* **cv2.dnn.NMSBoxes()**: Applies Non-Maximum Suppression (NMS) to eliminate redundant overlapping boxes. The parameters are:
  + **0.4**: Minimum confidence threshold.
  + **0.6**: Overlap threshold. Boxes with overlap greater than 60% are discarded.

**13. Draw Bounding Boxes and Labels**

font = cv2.FONT\_HERSHEY\_PLAIN

for i in range(len(boxes)):

if i in indexes:

x, y, w, h = boxes[i]

label = str(classes[class\_ids[i]])

color = colors[i]

cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)

cv2.putText(img, label, (x, y + 30), font, 1, (255, 255, 255), 2)

* Loops over all detected boxes, and if the box is kept after NMS, it draws a bounding box and labels it.
  + **cv2.rectangle()**: Draws a rectangle around the detected object.
  + **cv2.putText()**: Adds the class label (e.g., "person", "car") near the bounding box.

**14. Display the Image**

from google.colab.patches import cv2\_imshow

cv2\_imshow(img)

* **cv2\_imshow()**: Displays the resulting image with the bounding boxes and labels in the Colab notebook.

**Summary**

1. **Download the necessary YOLO files** (weights, configuration, and class labels).
2. **Load the YOLO model** using OpenCV.
3. **Process the image**: Rescale it, pass it through the YOLO model, and retrieve the object detections.
4. **Apply Non-Maximum Suppression** to reduce overlapping bounding boxes.
5. **Draw the bounding boxes and labels** for each detected object in the image.
6. **Display the result** in the Colab notebook.

The final result is an image with detected objects and their labels (e.g., "person", "dog", etc.) annotated with bounding boxes.