# YOLOv3 Object Detection Code Documentation

## Dependencies and Libraries

This script utilizes various libraries essential for image processing and deep learning:  
- `tensorflow`: TensorFlow for deep learning.  
- `opencv-python`: OpenCV for image processing.  
- `matplotlib`: Matplotlib for image visualization.  
Install these using the command:  
```python  
!pip install tensorflow opencv-python matplotlib  
```

## YOLO Model Files and Configuration

YOLOv3 model requires specific files to function:  
1. \*\*Weights\*\*: `yolov3.weights` file, pre-trained on the COCO dataset.  
2. \*\*Configuration File\*\*: `yolov3.cfg`, which defines the network structure.  
3. \*\*Class Labels\*\*: `coco.names`, a list of class labels.  
  
The files are downloaded using `wget`:  
```python  
!wget https://pjreddie.com/media/files/yolov3.weights  
!wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg  
!wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names  
```

## Loading the YOLO Model

The code loads YOLOv3 with OpenCV's `cv2.dnn.readNet`, specifying the weights and configuration files.  
Extracting the layer names, it identifies output layers required for detection:  
```python  
net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')  
layer\_names = net.getLayerNames()  
output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]  
```

## Loading and Preprocessing Images

The function `load\_image(img\_path)` loads an image and pre-processes it for YOLO input by resizing it and normalizing the pixel values:  
```python  
def load\_image(img\_path):  
 image = cv2.imread(img\_path)  
 blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)  
 net.setInput(blob)  
 outs = net.forward(output\_layers)  
 return image, outs, height, width  
```

## Processing Detections

`process\_detections` function iterates through the network's output to filter out detections with high confidence. For each detection, a bounding box is drawn around the detected object, and the class label and confidence score are displayed:  
```python  
def process\_detections(image, outs, height, width):  
 for out in outs:  
 for detection in out:  
 # Draw bounding box and label  
 cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)  
 label = f"{classes[class\_id]}: {confidence:.2f}"  
 cv2.putText(image, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)  
```

## Visualizing the Results

To visualize the results, `visualize\_image` converts the image to RGB and displays it using Matplotlib:  
```python  
def visualize\_image(image):  
 plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  
 plt.axis('off')  
 plt.show()  
```

## Testing YOLO on Multiple Images

The `test\_yolo\_on\_images` function accepts a list of image paths and processes each image using the defined functions. This can be customized to evaluate a larger dataset of images:  
```python  
def test\_yolo\_on\_images(image\_paths):  
 for img\_path in image\_paths:  
 image, outs, height, width = load\_image(img\_path)  
 process\_detections(image, outs, height, width)  
 visualize\_image(image)  
```

## Example Usage

To test the YOLO model, provide a list of image paths as shown below:  
```python  
test\_images = ["image1.jpg" , "image2.jpg", "image3.jpg"]  
test\_yolo\_on\_images(test\_images)  
```  
Ensure that the image paths are valid and accessible in the current directory.