YOLO

April 29, 2020

1 You Only Look Once (YOLO)

1.0.1 Introduction

As you learned in the previous lessons, YOLO is a state-of-the-art, real-time object detection algorithm. In this notebook, we will apply the YOLO algorithm to detect objects in images. We have provided a series of images that you can test the YOLO algorithm on. Below is a list of the available images that you can load:

- cat.jpg
- city_scene.jpg
- dog.jpg
- dog2.jpg
- eagle.jpg
- food.jpg
- giraffe.jpg
- horses.jpg
- motorbike.jpg
- person.jpg
- surf.jpg
- wine.jpg

These images are located in the `./images/`folder. We encourage you to test the YOLO algorithm on your own images as well. Have fun!

2 Importing Resources

We will start by loading the required packages into Python. We will be using *OpenCV* to load our images, *matplotlib* to plot them, `autils` module that contains some helper functions, and a modified version of *Darknet*. YOLO uses *Darknet*, an open source, deep neural network framework written by the creators of YOLO. The version of *Darknet* used in this notebook has been modified to work in PyTorch 0.4 and has been simplified because we won't be doing any training. Instead, we will be using a set of pre-trained weights that were trained on the Common Objects in Context (COCO) database. For more information on *Darknet*, please visit Darknet.

```
In [2]: import cv2
        import matplotlib.pyplot as plt
```

```python
from utils import *
from darknet import Darknet
```

3 Setting Up The Neural Network

We will be using the latest version of YOLO, known as YOLOv3. We have already downloaded the `yolov3.cfg` file that contains the network architecture used by YOLOv3 and placed it in the `/cfg/` folder. Similarly, we have placed the `yolov3.weights` file that contains the pre-trained weights in the `/weights/` directory. Finally, the `/data/` directory, contains the `coco.names` file that has the list of the 80 object classes that the weights were trained to detect.

In the code below, we start by specifying the location of the files that contain the neural network architecture, the pre-trained weights, and the object classes. We then use *Darknet* to setup the neural network using the network architecture specified in the `cfg_file`. We then use the `.load_weights()` method to load our set of pre-trained weights into the model. Finally, we use the `load_class_names()` function, from the `utils` module, to load the 80 object classes.

In [3]:
```python
# Set the location and name of the cfg file
cfg_file = './cfg/yolov3.cfg'

# Set the location and name of the pre-trained weights file
weight_file = './weights/yolov3.weights'

# Set the location and name of the COCO object classes file
namesfile = 'data/coco.names'

# Load the network architecture
m = Darknet(cfg_file)

# Load the pre-trained weights
m.load_weights(weight_file)

# Load the COCO object classes
class_names = load_class_names(namesfile)
```

```
Loading weights. Please Wait...100.00% Complete
```

4 Taking a Look at The Neural Network

Now that the neural network has been setup, we can see what it looks like. We can print the network using the `.print_network()` function.

In [4]:
```python
# Print the neural network used in YOLOv3
m.print_network()
```

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  32
```

```
 1 conv     64  3 x 3 / 2   416 x 416 x  32   ->   208 x 208 x  64
 2 conv     32  1 x 1 / 1   208 x 208 x  64   ->   208 x 208 x  32
 3 conv     64  3 x 3 / 1   208 x 208 x  32   ->   208 x 208 x  64
 4 shortcut 1
 5 conv    128  3 x 3 / 2   208 x 208 x  64   ->   104 x 104 x 128
 6 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64
 7 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
 8 shortcut 5
 9 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64
10 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
11 shortcut 8
12 conv    256  3 x 3 / 2   104 x 104 x 128   ->    52 x  52 x 256
13 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
14 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
15 shortcut 12
16 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
17 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
18 shortcut 15
19 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
20 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
21 shortcut 18
22 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
23 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
24 shortcut 21
25 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
26 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
27 shortcut 24
28 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
29 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
30 shortcut 27
31 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
32 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
33 shortcut 30
34 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
35 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
36 shortcut 33
37 conv    512  3 x 3 / 2    52 x  52 x 256   ->    26 x  26 x 512
38 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
39 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
40 shortcut 37
41 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
42 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
43 shortcut 40
44 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
45 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
46 shortcut 43
47 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
48 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
```

```
49 shortcut 46
50 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
51 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
52 shortcut 49
53 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
54 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
55 shortcut 52
56 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
57 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
58 shortcut 55
59 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
60 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
61 shortcut 58
62 conv   1024  3 x 3 / 2    26 x  26 x 512   ->    13 x  13 x1024
63 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
64 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
65 shortcut 62
66 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
67 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
68 shortcut 65
69 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
70 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
71 shortcut 68
72 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
73 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
74 shortcut 71
75 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
76 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
77 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
78 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
79 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
80 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
81 conv    255  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 255
82 detection
83 route  79
84 conv    256  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 256
85 upsample            * 2   13 x  13 x 256   ->    26 x  26 x 256
86 route  85 61
87 conv    256  1 x 1 / 1    26 x  26 x 768   ->    26 x  26 x 256
88 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
89 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
90 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
91 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
92 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
93 conv    255  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 255
94 detection
95 route  91
96 conv    128  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x 128
```

```
 97 upsample          * 2    26 x  26 x 128   ->    52 x  52 x 128
 98 route  97 36
 99 conv    128  1 x 1 / 1    52 x  52 x 384   ->    52 x  52 x 128
100 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
101 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
102 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
103 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
104 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
105 conv    255  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 255
106 detection
```

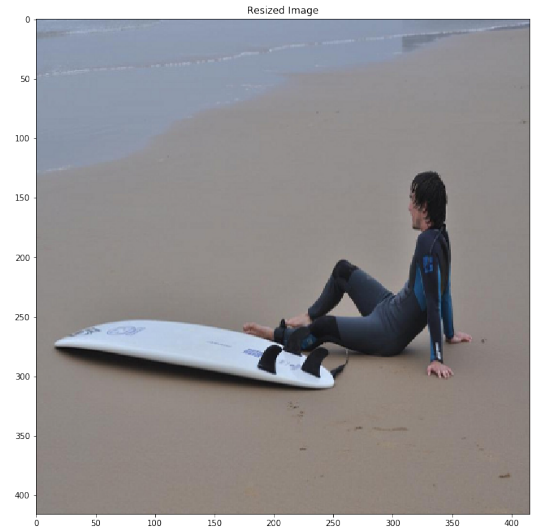
As we can see, the neural network used by YOLOv3 consists mainly of convolutional layers, with some shortcut connections and upsample layers. For a full description of this network please refer to the YOLOv3 Paper.

5 Loading and Resizing Our Images

In the code below, we load our images using OpenCV's `cv2.imread()` function. Since, this function loads images as BGR we will convert our images to RGB so we can display them with the correct colors.

As we can see in the previous cell, the input size of the first layer of the network is 416 x 416 x 3. Since images have different sizes, we have to resize our images to be compatible with the input size of the first layer in the network. In the code below, we resize our images using OpenCV's `cv2.resize()` function. We then plot the original and resized images.

In [5]:
```
# Set the default figure size
plt.rcParams['figure.figsize'] = [24.0, 14.0]

# Load the image
img = cv2.imread('./images/surf.jpg')

# Convert the image to RGB
original_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# We resize the image to the input width and height of the first layer of the network.
resized_image = cv2.resize(original_image, (m.width, m.height))

# Display the images
plt.subplot(121)
plt.title('Original Image')
plt.imshow(original_image)
plt.subplot(122)
plt.title('Resized Image')
plt.imshow(resized_image)
plt.show()
```



6 Setting the Non-Maximal Suppression Threshold

As you learned in the previous lessons, YOLO uses **Non-Maximal Suppression (NMS)** to only keep the best bounding box. The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold. In the code below, we set this NMS threshold to `0.6`. This means that all predicted bounding boxes that have a detection probability less than 0.6 will be removed.

```
In [6]: # Set the NMS threshold
        nms_thresh = 0.6
```

7 Setting the Intersection Over Union Threshold

After removing all the predicted bounding boxes that have a low detection probability, the second step in NMS, is to select the bounding boxes with the highest detection probability and eliminate all the bounding boxes whose **Intersection Over Union (IOU)** value is higher than a given IOU threshold. In the code below, we set this IOU threshold to `0.4`. This means that all predicted bounding boxes that have an IOU value greater than 0.4 with respect to the best bounding boxes will be removed.

In the `utils` module you will find the `nms` function, that performs the second step of Non-Maximal Suppression, and the `boxes_iou` function that calculates the Intersection over Union of two given bounding boxes. You are encouraged to look at these functions to see how they work.

```
In [7]: # Set the IOU threshold
        iou_thresh = 0.4
```

8 Object Detection

Once the image has been loaded and resized, and you have chosen your parameters for `nms_thresh` and `iou_thresh`, we can use the YOLO algorithm to detect objects in the image. We detect the objects using the `detect_objects(m, resized_image, iou_thresh, nms_thresh)`function from the `utils` module. This function takes in the model `m` returned by *Darknet*, the resized image, and the NMS and IOU thresholds, and returns the bounding boxes of the objects found.

Each bounding box contains 7 parameters: the coordinates *(x, y)* of the center of the bounding box, the width w and height h of the bounding box, the confidence detection level, the object class probability, and the object class id. The `detect_objects()` function also prints out the time it took for the YOLO algorithm to detect the objects in the image and the number of objects detected. Since we are running the algorithm on a CPU it takes about 2 seconds to detect the objects in an image, however, if we were to use a GPU it would run much faster.

Once we have the bounding boxes of the objects found by YOLO, we can print the class of the objects found and their corresponding object class probability. To do this we use the `print_objects()` function in the `utils` module.

Finally, we use the `plot_boxes()` function to plot the bounding boxes and corresponding object class labels found by YOLO in our image. If you set the `plot_labels` flag to `False` you will display the bounding boxes with no labels. This makes it easier to view the bounding boxes if your `nms_thresh` is too low. The `plot_boxes()`function uses the same color to plot the bounding boxes of the same object class. However, if you want all bounding boxes to be the same color, you can use the `color` keyword to set the desired color. For example, if you want all the bounding boxes to be red you can use:

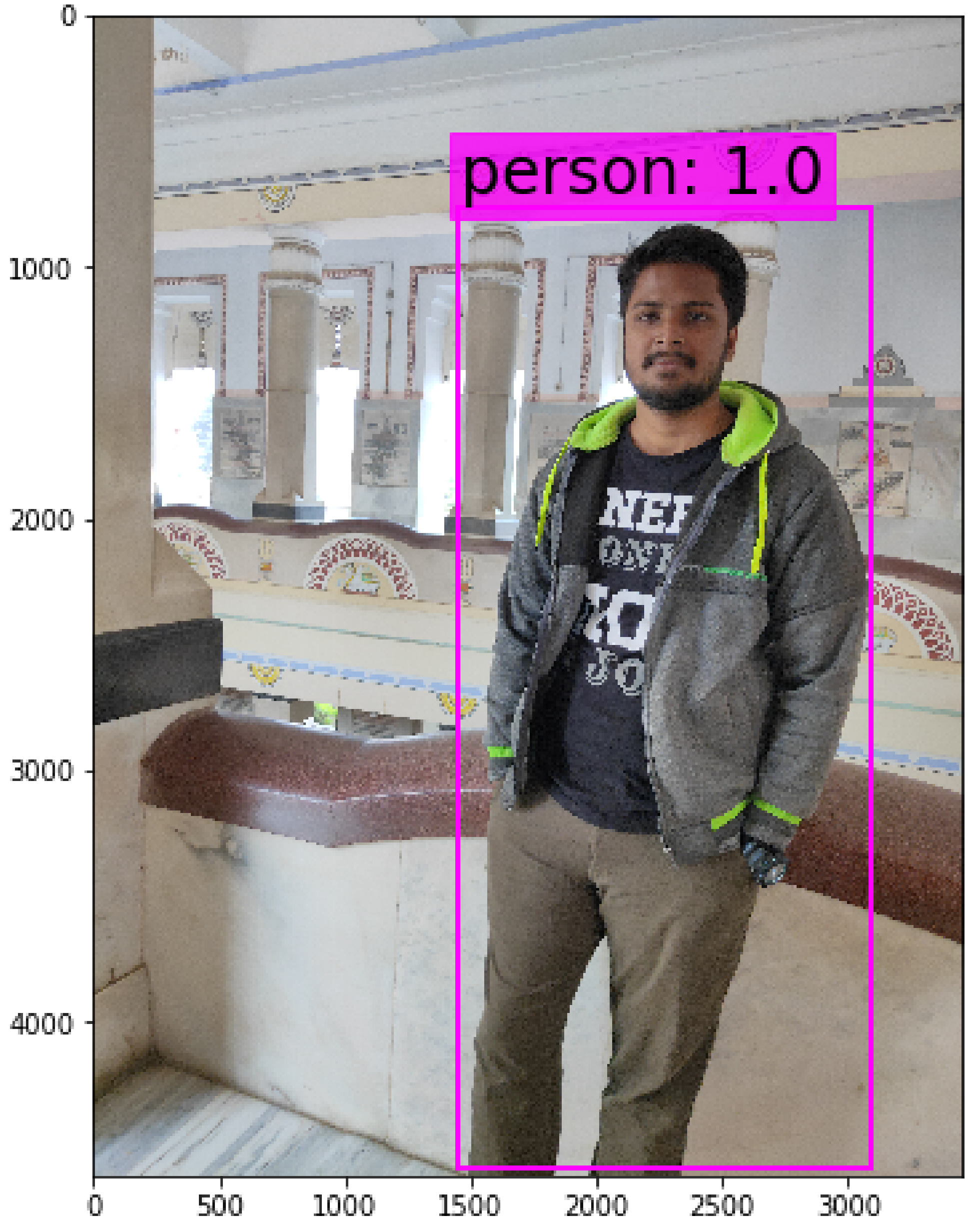
`plot_boxes(original_image, boxes, class_names, plot_labels = True, color = (1,0,0))`

You are encouraged to change the `iou_thresh` and `nms_thresh` parameters to see how they affect the YOLO detection algorithm. The default values of `iou_thresh = 0.4` and `nms_thresh = 0.6` work well to detect objects in different kinds of images. In the cell below, we have repeated some of the code used before in order to prevent you from scrolling up down when you want to change the `iou_thresh` and `nms_thresh`parameters or the image. Have Fun!

In [8]:
```
# Set the default figure size
plt.rcParams['figure.figsize'] = [16.0, 8.0]

# Load the image
img = cv2.imread('./images/test_images_20.jpg')

# Convert the image to RGB
original_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# We resize the image to the input width and height of the first layer of the network.
resized_image = cv2.resize(original_image, (m.width, m.height))

# Set the IOU threshold. Default value is 0.4
iou_thresh = 0.4

# Set the NMS threshold. Default value is 0.6
```

```python
nms_thresh = 0.6

# Detect objects in the image
boxes = detect_objects(m, resized_image, iou_thresh, nms_thresh)

# Print the objects found and the confidence level
print_objects(boxes, class_names)

#Plot the image with bounding boxes and corresponding object class labels
plot_boxes(original_image, boxes, class_names, plot_labels = True)
```

```
It took 2.502 seconds to detect the objects in the image.

Number of Objects Detected: 1

Objects Found and Confidence Level:

1. person: 0.999999
```



In [ ]: