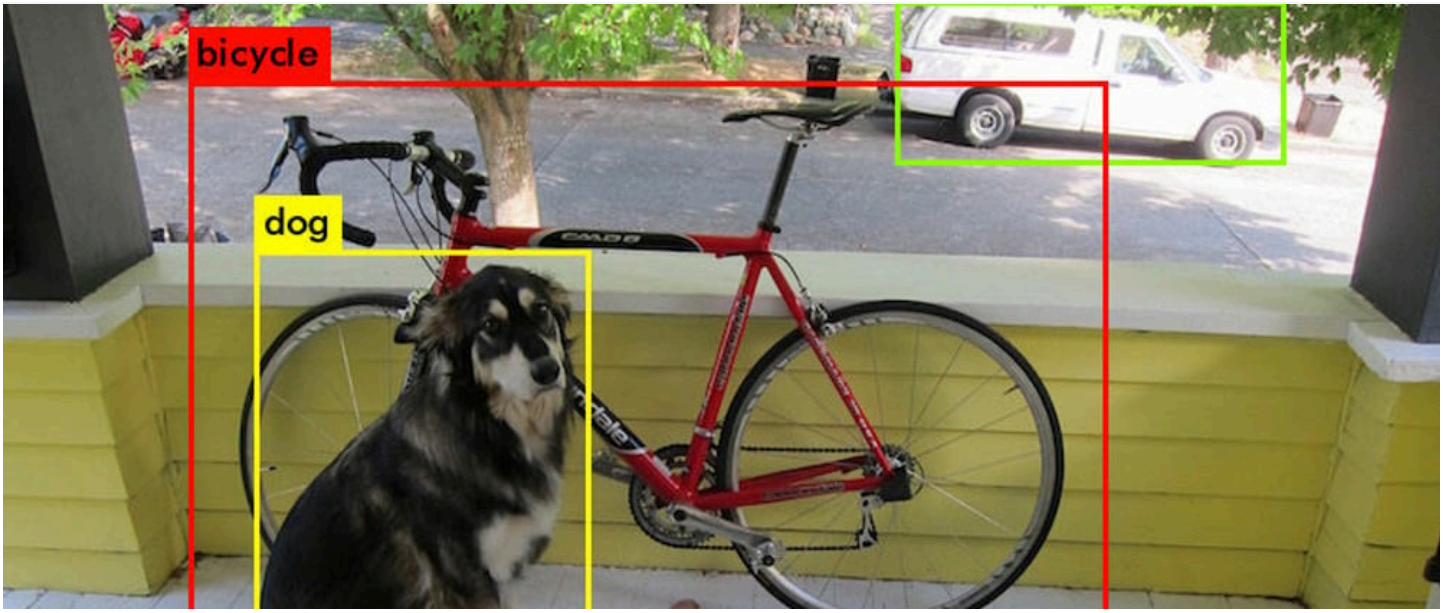


Afroz Chakure
Posted on Mar 2, 2021 • Edited on Sep 1

All you need to know about YOLO v3 (You Only Look Once)

#deeplearning #machinelearning #architecture #docker

This blog will provide an exhaustive study of YOLOv3 (You only look once), which is one of the most popular deep learning models extensively used for object detection, semantic segmentation, and image classification. In this blog, I'll explain the architecture of YOLOv3 model, with its different layers, and see some results for object detection that I got while running the inference program on some test images using the model.

So keep reading the blog to find out more about YOLOv3.

What is YOLO?

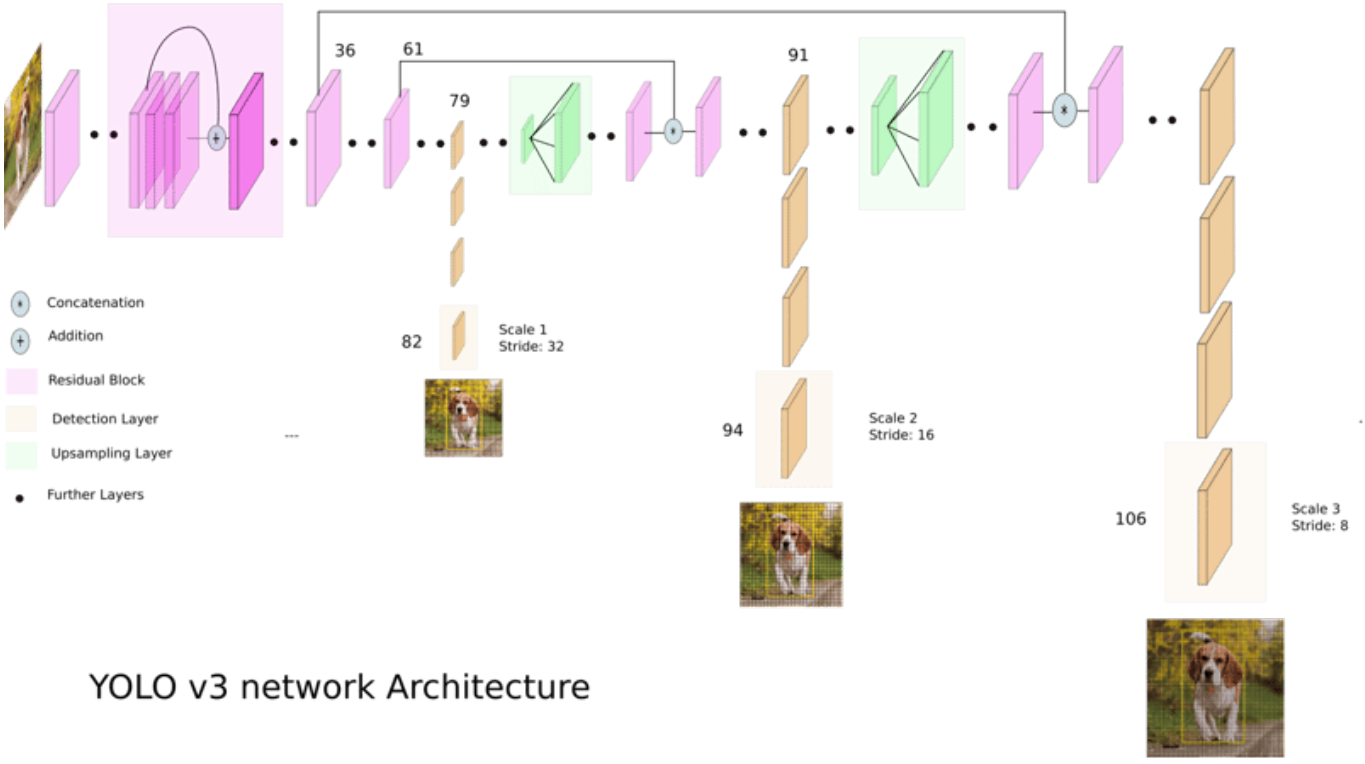
- "You Only Look Once" or YOLO is a family of deep learning models designed for fast object Detection.
- There are three main variations of YOLO, they are YOLOv1, YOLOv2, and YOLOv3.
- The first version proposed the general architecture, where the second version refined the design and made use of predefined anchor boxes to improve the

bounding box proposal, and version three further refined the model architecture and training process.

- It is based on the idea that :

" A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. "

1. Network Architecture Diagram of YOLOv3



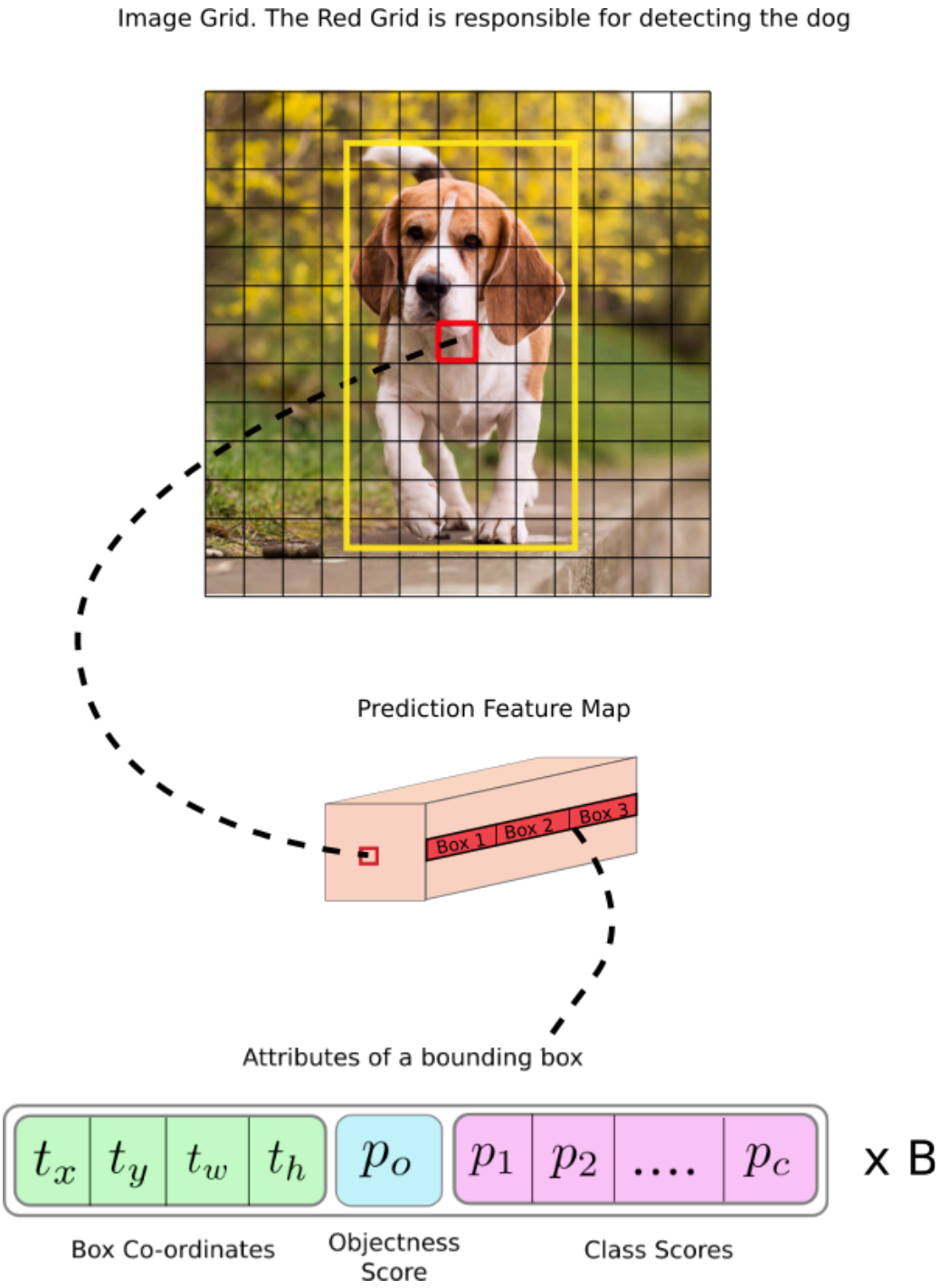
YOLO v3 network Architecture

2. Description of Architecture

Steps for object Detection using YOLO v3:

- The inputs is a batch of images of shape (m, 416, 416, 3).
- YOLO v3 passes this image to a convolutional neural network (CNN).
- The last two dimensions of the above output are flattened to get an output volume of (19, 19, 425):
  - Here, each cell of a 19 x 19 grid returns 425 numbers.
  - 425 = 5 * 85, where 5 is the number of anchor boxes per grid.
  - 85 = 5 + 80, where 5 is (pc, bx, by, bh, bw) and 80 is the number of classes we want to detect.
- The output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers **(pc, bx, by, bh, bw, c)**. If we expand c into an 80-dimensional vector, each bounding box is represented by 85 numbers.

- Finally, we do the IoU (Intersection over Union) and Non-Max Suppression to avoid selecting overlapping boxes.



Regarding the architecture:

- YOLO v3 uses a variant of Darknet, which originally has **53 layer network trained on Imagenet**.
- For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3.
- In YOLO v3, the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network.
- The shape of detection kernel is **1 x 1 x (B x (5 + C))**. Here **B is the number of bounding boxes a cell on the feature map can predict**, **'5' is for the 4**

bounding box attributes and one object confidence and **C is the no. of classes**.

- YOLO v3 uses **binary cross-entropy** for calculating the **classification loss** for each label while **object confidence and class predictions are predicted through logistic regression**.

Hyper-parameters used

- **class_threshold** - Defines probability threshold for the predicted object.
- **Non-Max suppression Threshold** - It helps overcome the problem of detecting an object multiple times in an image. It does this by taking boxes with maximum probability and suppressing the close-by boxes with non-max probabilities (less than the predefined threshold).
- **input_height & input_shape** - Image size to input.

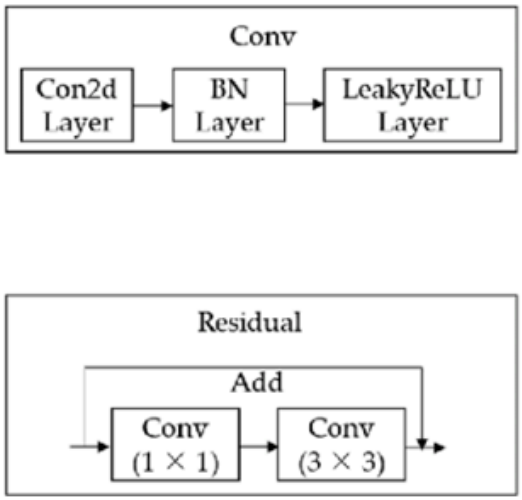
CNN architecture of Darknet-53

- **Darknet-53** is used as a feature extractor.
- Darknet-53 mainly composed of 3 x 3 and 1 x 1 filters with skip connections like the residual network in ResNet.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

2. Architecture diagram of YOLOv3

Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13



3. Layers Details

- YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN)
- In YOLOv3 a deeper architecture of feature extractor called Darknet-53 is used.

Convolution layers in YOLOv3

- It contains 53 convolutional layers which have been, each followed by batch normalization layer and Leaky ReLU activation.
- Convolution layer is used to convolve multiple filters on the images and produces multiple feature maps
- No form of pooling is used and a convolutional layer with stride 2 is used to downsample the feature maps.
- It helps in preventing loss of low-level features often attributed to pooling.

4. Different Layers inside YOLO

- **Code for Layer 1 to 53 in Tensorflow** : Consider res_block() method for below code

```
 python
def res_block(inputs, filters):
    shortcut = inputs
    net = conv2d(inputs, filters * 1, 1)
    net = conv2d(net, filters * 2, 3)
    net = net + shortcut
    return net
```

First two conv2d layers with 32 and 64 filters

```
 python
net = conv2d(inputs, 32,  3, strides=1)
net = conv2d(net, 64,  3, strides=2)
```

res_block * 1

```
    net = res_block(net, 32)
# Convolutional block with 128 filters
net = conv2d(net, 128, 3, strides=2)
```

res_block * 2

```
for i in range(2):
        net = res_block(net, 64)
# Convolutional layer with 256 filters
net = conv2d(net, 256, 3, strides=2)
```

res_block * 8

```
for i in range(8):
    net = res_block(net, 128)
# Convolutional layer with 512 filters
route_1 = net
net = conv2d(net, 512, 3, strides=2)
```

res_block * 8

```
for i in range(8):
    net = res_block(net, 256)
# Convolutional layer with 1024 filters
route_2 = net
net = conv2d(net, 1024, 3, strides=2)
```

res_block * 4

```
for i in range(4):
    net = res_block(net, 512)
route_3 = net
```

---

5. Input Details for Model Inference

i. Input Preprocessing:

The images need to be resized to 416 x 416 pixels before feeding it into our model or the dimensions can also be specified while running the python file

ii. Input Dimensions:

The model expects inputs to be color images with the **square shape of 416 x 416 pixels** or it can also be specified by the user.

---

6. Details on Model Inference program and output

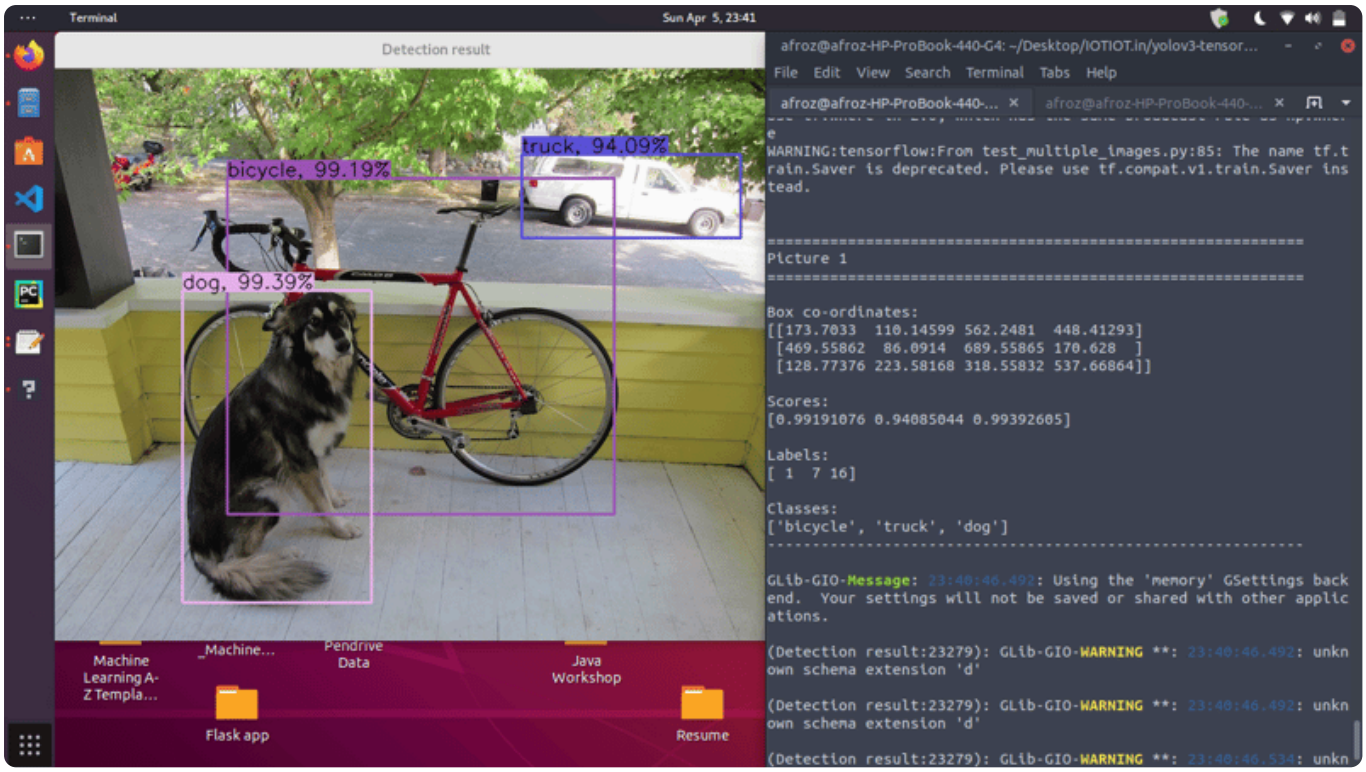
1. The output of the model

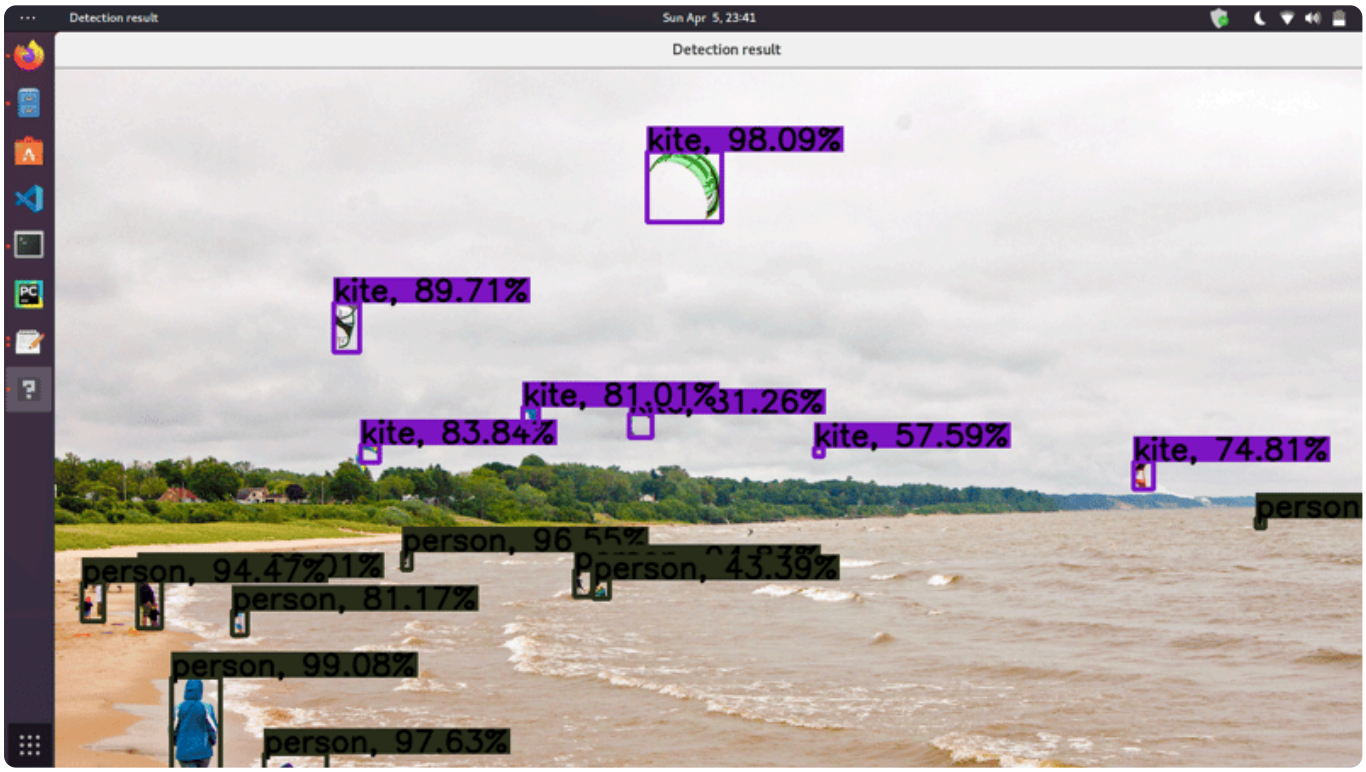
- The output is a list of bounding boxes along with the recognized classes.
- Each bounding box is represented by 6 numbers **(pc, bx, by, bh, bw, c)**. Here c is an 80-dimensional vector equivalent to the number of classes we want to predict, each bounding box is represented by 85 numbers.

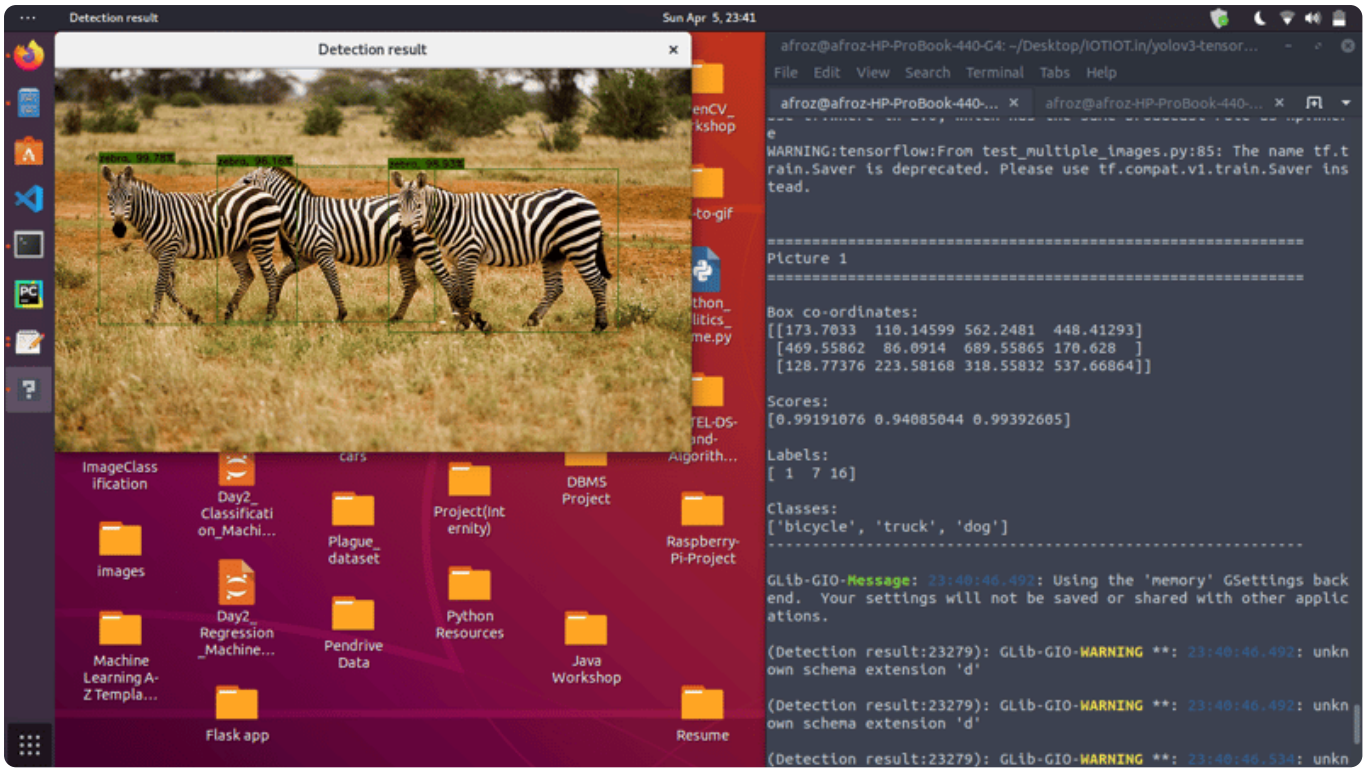
2. Output of Post-processing

- The Inference program **produces a list of numpy arrays**, the shape of which is displayed as the output. These arrays **predict both the bounding boxes and class labels but are encoded**.
- Further we will take each of the numpy arrays and decode the candidate bounding boxes and class predictions. If there are any **bounding boxes that don't confidently describe an object (having class probabilities less than the threshold, 0.3) we'll ignore them**.
- Here a **maximum of 200 bounding boxes** can be considered in an image.
- I have used the **correct_yolo_boxes()** function to **perform translation of bounding box coordinates so that we can plot the original image and draw the bounding boxes**.
- Again to **remove those candidate bounding boxes that may be referring to the same object**, we define the amount of overlap as **non-max suppression threshold = 0.45**.
- Also there is a need to rescale the coordinates of the bounding boxes to the original image along with displaying the label and scores on top of each of them.
- All these Post-processing steps need to be performed before we get the bounding boxes along with the recognized classes in our output image.

3. Output of inference program for the model


Terminal
Sun Apr 5, 23:41
Detection result
bicycle, 99.19%
truck, 94.09%
dog, 99.39%
Machine Learning A-Z Templa...
_Machine...
Pendrive Data
Java Workshop
Flask app
Resume
afroz@afroz-HP-ProBook-440-G4: ~/Desktop/IOTIOT.in/yolov3-tensor...
File Edit View Search Terminal Tabs Help
WARNING:tensorflow:From test_multiple_images.py:85: The name tf.train.Saver is deprecated. Please use tf.compat.v1.train.Saver instead.
Picture 1
Box co-ordinates:
[[173.7033 110.14599 562.2481 448.41293]
[469.55862 86.0914 689.55865 170.628]
[128.77376 223.58168 318.55832 537.66864]]
Scores:
[0.99191076 0.94085044 0.99392605]
Labels:
[1 7 16]
Classes:
['bicycle', 'truck', 'dog']
GLib-GIO-Message: 23:40:46.492: Using the 'memory' GSettings backend. Your settings will not be saved or shared with other applications.
(Detection result:23279): GLib-GIO-WARNING **: 23:40:46.492: unknown schema extension 'd'
(Detection result:23279): GLib-GIO-WARNING **: 23:40:46.492: unknown schema extension 'd'
(Detection result:23279): GLib-GIO-WARNING **: 23:40:46.534: unkn


Detection result
Sun Apr 5, 23:41
Detection result
kite, 98.09%
kite, 89.71%
kite, 81.01%
kite, 83.84%
kite, 57.59%
kite, 74.81%
person
person, 81.17%
person, 99.08%
person, 97.63%



---

7. Speed and Accuracy of YOLOv3

- **Model Speed (in FPS)** - On a Pascal Titan X it processes images at 30 FPS.
- **Model Accuracy (On Testing dataset)** - It has a mAP (Mean Average Precision) of 87.54% on VOC test set.

8. Run the model

Find link to my Dockerfile : [afrozchakure/yolov3-tensorflow]

To run the model using docker (docker should be preinstalled) on your machine use the below commands in the terminal :)

```
 shell

xhost +

sudo docker run --rm -ti --net=host --ipc=host -e DISPLAY=$DISPLAY -v /tmp/.X
```

1. Learn more about Working Principles of YOLOv3 from below resources

Referrence Link 1

Referrence Link 2

Well done if you came this far. Cheers 😇💖
Connect with me: https://www.linkedin.com/in/afrozchakure

You can support my work using Buy me a Coffee:
https://buymeacoffee.com/afrozchakure

Get Started

Top comments (0)

Code of Conduct • Report abuse

Afroz Chakure

Developer | Blogger | Competitive Programmer

LOCATION
Pune, Maharashtra, India

EDUCATION
B.Tech Computer Science and Engineering @ MIT-WPU, Pune

WORK
Application Developer Programmer Analyst 2 @ Citi

JOINED
Aug 19, 2019

More from Afroz Chakure

Transfer Learning with MobileNet-v2
#deeplearning #machinelearning #architecture #docker

Hyper-Parameter Tuning to Optimize ML Models (MobileNet v2)
#machinelearning #deeplearning #architecture #docker

Deep Neural Network (DNN) in a Brief
#machinelearning #deeplearning #architecture #datascience