COMP451 Project
Final Report

Erhan Tezcan & Öykü Zeynep Bayramoğlu

January 7, 2020

1 Introduction

We have implemented YOLOv3 (You Look Only Once, version 3), which is an algorithm for object detection problem. It is one of the state-of-the-art algorithms on object detection, noticeable for its good performance on both classification and time complexity.

1.1 YOLOv3

YOLO treats object detection problem as a regression problem to spatially separated bounding boxes and associated class probabilities. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes in one evaluation.

Since the whole detection pipeline is a single network, YOLO is orders of magnitude faster (45 frames per second) than other object detection algorithms. Second, YOLO sees the entire image during training and test time so it is less likely to predict false positives on background compared to Fast R-CNN [1].

There were 2 previous versions of YOLO: YOLO, and later YOLO9000. The first improvement: YOLOv2, uses batch normalization which eliminates the need for other forms of regularizations and improves mAP[1] score by 2%. One of the most notable changes which is visible in YOLOv2 is the introduction of the anchor boxes. Instead of predicting the coordinates directly

[1]Mean Average Precision

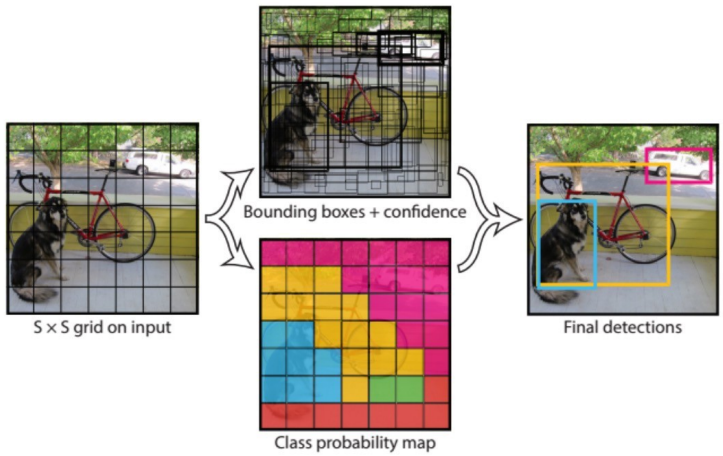

Figure 1: A basic model of YOLO [1].

using fully-connected layers, the prediction was done using predefined anchor boxes. Also, instead of fixed input dimensions, YOLOv2 is trained with random images with different dimensions range between 320×320 to 608×608. Finally, YOLOv2 has a passthrough layer that brings features from an earlier layer at 26×26 resolution to help detecting smaller objects.

Earlier versions of YOLO still struggled with small objects; now, YOLOv3 makes prediction across 3 different scales. The detection layer is used make detection at feature maps of three different sizes, having strides 32, 16, 8 respectively [2]. YOLOv3 also uses shortcut connections to address vanishing gradient problem. From here on in this report, we will address YOLOv3 as YOLO.

1.2 Microsoft COCO: Common Objects in Context

COCO is a large, richly-annotated dataset comprised of images depicting complex everyday scenes of common objects in their natural context. The dataset contains photos of 91 objects types with a total of 2.5 million labeled instances in 328k images. Objects are labeled using per-instance segmentations to aid in precise object localization [4].

2 YOLO

2.1 Configuration of YOLO Network

YOLO uses a configuration file to build the network. The configuration file (with the extension `.cfg`) describes the layout of the network, block by block. Each block describes the parameters of that layer such as the number of filters, stride and activation function. In our project, we used the official configuration file[2] released by the author. There are 6 different blocks defined in the configuration file:

- **Convolutional**: In YOLO, predictions are done by a convolutional layer which use 1 x 1 convolutions.

```
[convolutional]
batch_normalize=1 # boolean
filters=64
size=3
stride=1
pad=1
activation=leaky # leaky, relu, tanh etc.
```

- **Shortcut**: A Shortcut connection adds feature maps of a previous layer to current outputs. This is a countermeasure to avoid vanishing gradient, similar to what was done by "ResNet" with their "Residual Connection".

```
[shortcut]
from=-3 # from which layer are we taking the shortcut
activation=linear
```

- **Upsample**: An Upsample block upsamples the feature map by a factor of stride using bilinear upsampling.

```
[upsample]
stride=2
```

- **Route**: A Route block literally acts as a route from one layer to another. It only has one field called "layers" and that can either take a single value or two values.

[2] https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg

- *1 Value*
  If given a single value only, the route layer outputs the feature maps of the indexed layer. In the code snippet below this would mean outputting the feature map from 4^{th} later backwards from this route layer.
- *2 Values*
  If given two values, the route layer outputs the concatenation of the feature maps of the two indexed layers.

```
[route]
layers = -4

[route]
layers = -1, 61
```

- **YOLO**: This block specifies the coordinates of anchors and the thresholds to be used for detection. The two attributes: `mask` and `anchors` are important. `anchors` specify the anchors to be used by the detection layer, but only those specified by the `mask` are taken into consideration. In the configuration file, we have 3 different detection layers, each with same `anchors` attribute but with different `mask` attributes, so in total all anchors are being used. Anchors are given in more detail in section 2.4.

```
[yolo]
mask = 0,1,2
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119,
116,90, 156,198, 373,326
classes=80
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

- **Net**: Though being a block in the configuration file, this one is not particularly a "layer" of the network. It instead describes hyper-parameters such as momentum, decay and batch size for our network.

```
[net]
# If Testing, use these:
batch=1
subdivisions=1
# If Training, use these:
#batch=64
#subdivisions=16

width= 416
height = 416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

The layer structure all together is given in figure 2.

2.2 Object Confidence Thresholding

How does YOLO decide on a classification? For that, first we should explain what a class confidence is. Class confidences represent the probabilities of the detected object belonging to a particular class. Prior to YOLOv3, the scores where given using Softmax. Now, YOLOv3 uses Sigmoid function. The reason is that Softmax'ing class scores makes the assumption that the classes are mutually exclusive: if an object belongs to one class, then it's guaranteed it cannot belong to another class. This is in fact true for COCO dataset that we use in our project. However in reality, this does not always hold. A very simple example would be the classes: "Man", "Woman" and "Person".

YOLO divides every image into a grid of $S \times S$. Output tensor consists of

	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			

Figure 2: YOLOv3 network structure.

$S \times S \times (B \times (5 + C))$ entries where $S \times S$ is the number of cells and B is the number of bounding boxes each cell can predict. The center coordinates, the dimensions, the objectness score and C class confidences are predicted for each bounding box. Boxes with object confidence score below the threshold are ignored. The threshold can be configured in the configuration file.

2.3 Non-Maximum Suppression

Non-Maximum Suppression [5] is for when there are multiple detections of the same class, which is possible since all bounding boxes of a cell may detect a box or the adjacent cells may detect the same object. To remedy this problem, "Intersection over Union" (IoU) values between the bounding box with highest confidence score and every other box are calculated. If two bounding boxes of the same class having an IoU larger than a threshold, then the one with lower class confidence is eliminated. This process is repeated with the remaining bounding boxes.

Intersection over Union is an evaluation metric used to measure the accuracy of an object detector. IoU score between two bounding boxes is computed by dividing the area of overlap of boxes by the area of union, an example is given in figure 3.

Figure 3: An example of computing Intersection over Unions for various bounding boxes.

2.4 Anchors & Bounding Boxes

Prior to YOLO9000 [3] bounding boxes were predicted directly using fully-connected layers on top of the convolutional feature extractor. Now, motivated by the performance of Faster R-CNN [6] where the region proposal network (RPN) was part of the CNN, authors of YOLO decided to follow the same approach. YOLO9000 and later versions all have their RPN as a part of the CNN.

Bounding boxes are the heart of the object detection. Each bounding box has output has $5 + C$ dimensions: box center coordinates x and y, width, height, objectness score and the remaining C are the class confidence scores.

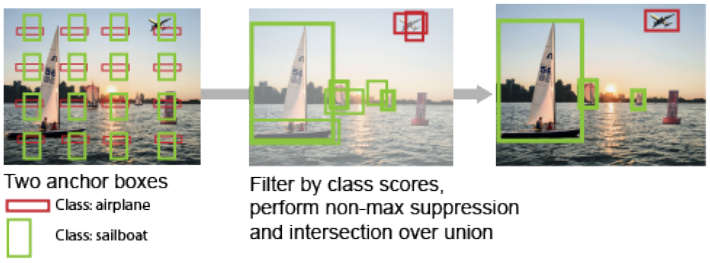

Figure 4: Anchor boxes. (source: `https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html`)

Figure 5 shows the bounding box of YOLOv3. Notice how Sigmoid (σ) function is used.

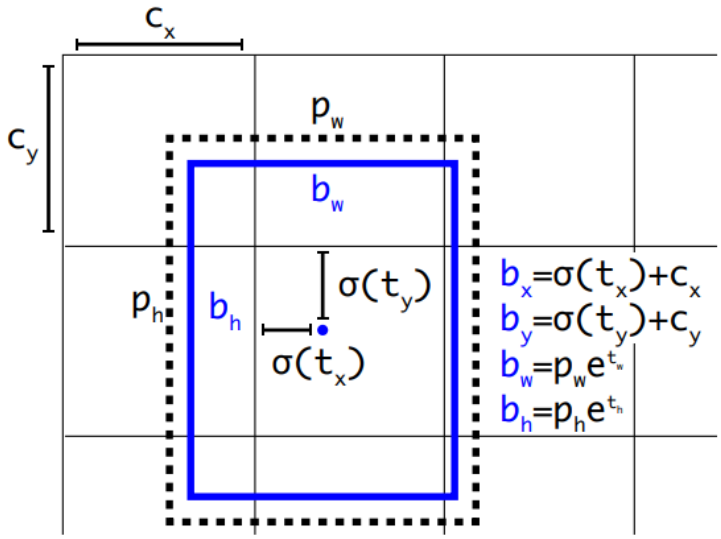

Figure 5: Bounding box, from the original YOLOv3 paper [2]. t_w, t_h, t_x, t_y are predictions by the network.

3 Implementation

We have used PyTorch[3] and OpenCV[4] to implement YOLOv3. We took help from the series of tutorials at `https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/`. The code is composed of two parts:

1. **Loading the network**

   - Load the configuration.[5]
   - Load the weights.[6]

   Both files are official, and they are meant to be used together. Therefore, one should be careful when reading them.

2. **Detection over images**
   This is the "test" over the network. We can run detection over a folder of images, we can run detection on a video or our webcam, where in that case the frames become the input to the network! There are several command line arguments that we pass:

   - `-images`: Directory of the images or the image file that we will test.
   - `-det`: Directory to store the detections to.
   - `-bs`: Batch size.
   - `-confidence`: Object confidence threshold to filter the predictions. This is explained in detail under section 2.2.
   - `-nms_thresh`: Non-maximum suppression threshold. This is explained in detail under section 2.3.
   - `-cfg`: The configuration file.
   - `-weights`: The YOLO weights file.
   - `-reso`: Input resolution of the network. Higher resolution means higher accuracy, but it takes longer to execute. Lower resolution means lower accuracy, but faster execution.

[3] `https://pytorch.org/`
[4] `https://pypi.org/project/opencv-python/`
[5] `https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg`
[6] `https://pjreddie.com/media/files/yolov3.weights` (237 MB)

There are three important files: `detect.py`, `darknet.py` and `util.py`.

- `darknet.py` is the file where our CNN is created and modeled with PyTorch.
- `detect.py` is like our *main* file. We call it to start the script, and it is where the argument parsing happens.
- `util.py` is the file with utility functions, such as preparing the image, calculating IoU, writing the boxes on image etc.

There is also `video.py` file, that takes `–video` command line argument instead of `–images`. Here, we can either give the path to the video file, or we can write *webcam* there and it will be using the webcam instead of video file.

4 Conclusions

We are implementing YOLOv3 in our laptops. One of them is an ASUS UX303UB, which in it has an NVIDIA GT940M GPU, which supports CUDA. This way, we can use advantage of parallel GPU processing power while testing YOLO.

Since we could not experiment with the training, we modified the code a bit so that we can remove some layers from the network. This is done by adding a `skip_this=1` attribute (1 is arbitrary) to the block in configuration file. This block is ignored during the model creation, however it is kept in mind, because while reading the weights we should skip the corresponding weights too. We removed route and shortcut layers, but it is still able to predict the boxes, with at most 0.005 millisecond difference on average prediction time on an image. This is mostly because route and shortcut layers are residual connections, which play an important role on preventing vanishing gradients during training. So it did not affect much during the test time.

No we will show some examples. We will also include a video in our presentation, though we can not include a video in this report.

The overall runtime of detecting the images shown on figures7, 8, 9, 11, 12 was 3.583 seconds, with an average of 0.751 seconds per image.

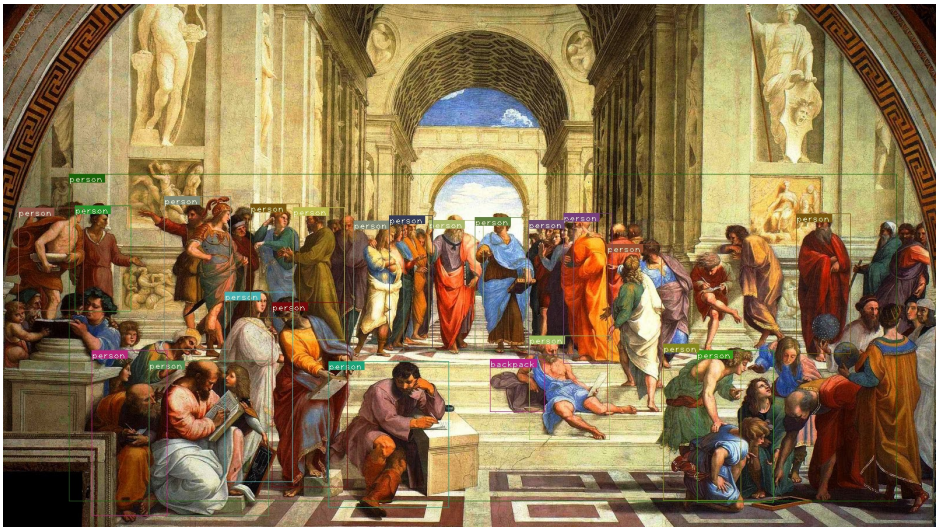

Figure 6: Persons detected in *School of Athens* by Raphael (circa 1510)

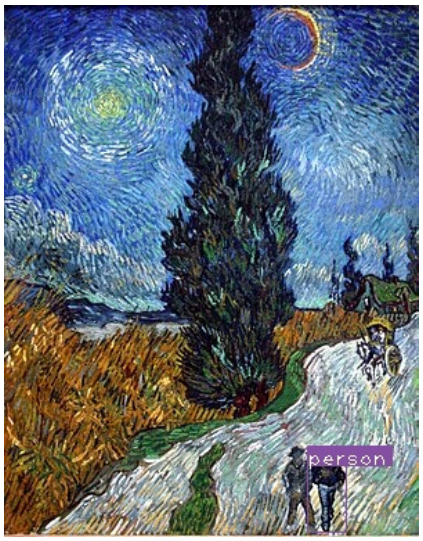

Figure 7: Person detected in *Road with Cypress and Star* by Van Gogh (1890). Notice how both persons are drawn same, but only one of them is detected.

References

[1] Redmon, J., et al. *You only look once: Unified, real-time object detection.* Proceedings of the IEEE conference on computer vision and pattern

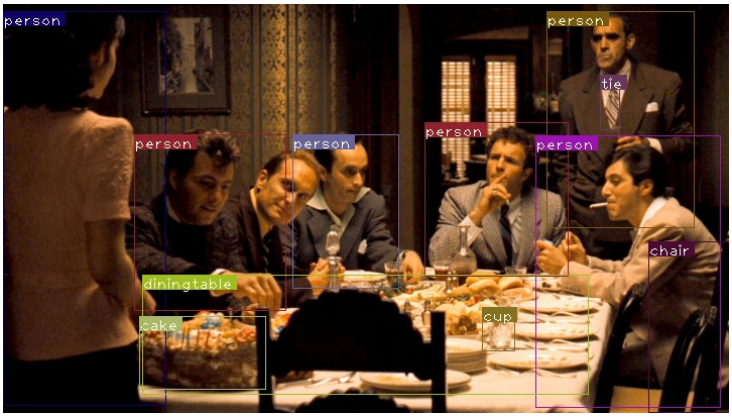

Figure 8: A scene from Godfather II. Notice how "dining table" is detected all together too!

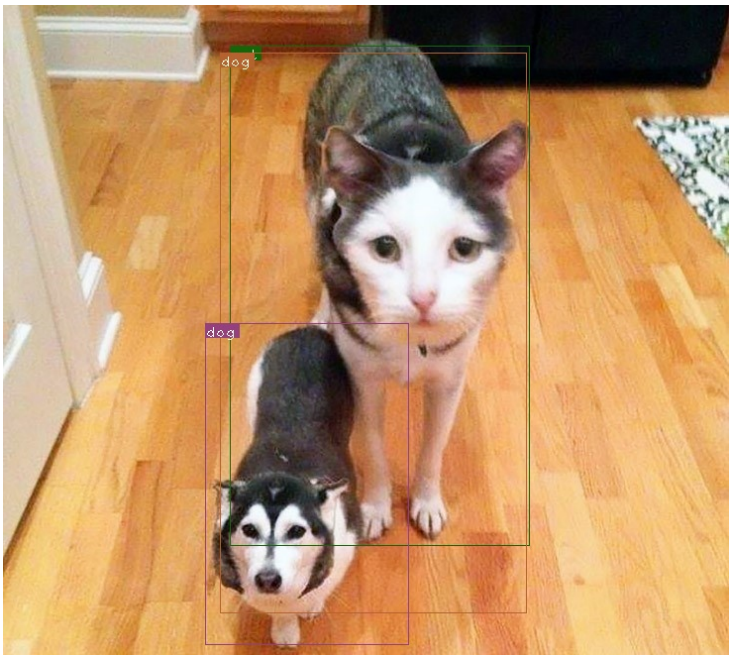

Figure 9: A cat and dog face-swap. The cat with dog face is detected as a dog, but the dog with cat face is detected both as a cat and a dog. This was ran with confidence parameter 0.5

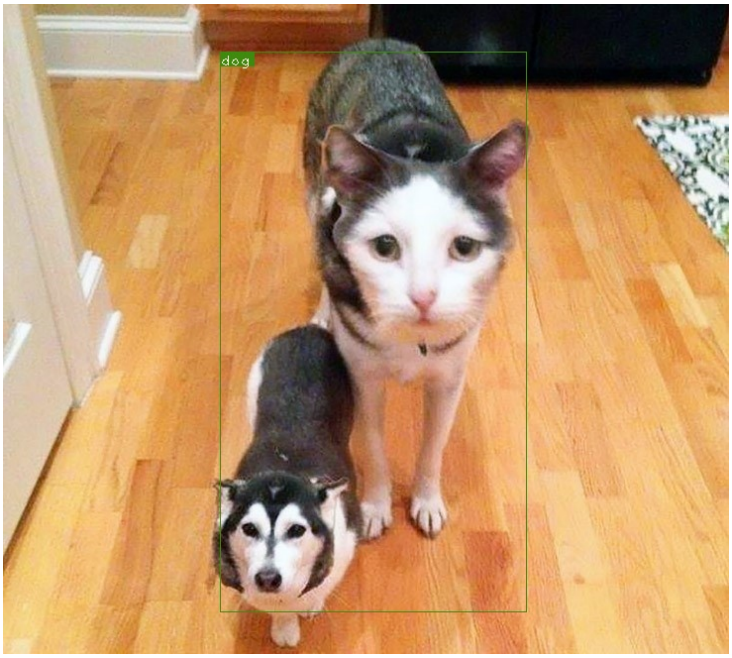

Figure 10: The same image from figure 9 is given, but now we use a confidence threshold of 0.8. YOLO does not trust the dog-faced cat on the left, and detects the cat-faced dog as a dog.

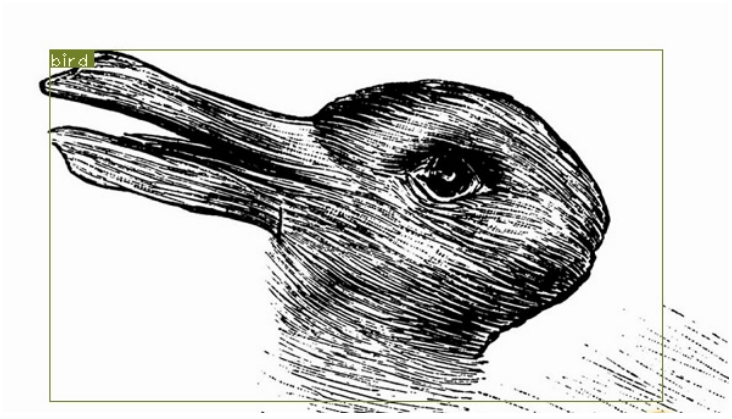

Figure 11: The famous "rabbit or duck?" illusion. According to YOLO, it is a duck; since ducks belong to the class *Aves* (Birds), and YOLO detected this as a bird.

Figure 12: Okay, figure 11 is fallacious, this YOLO does not know about rabbits!

recognition. **(2016)**

[2] Redmon, J., and Farhadi, A. *YOLOv3: An incremental improvement.* arXiv e-prints arXiv:1804.02767 **(2018)**

[3] Redmon, J., and Farhadi, A. *YOLO9000: Better, Faster, Stronger.* arXiv e-prints, arXiv:1612.08242 **(2016)**

[4] Lin, Tsung-Yi, et al. *Microsoft coco: Common objects in context.* European Conference on Computer Vision. Springer, Cham, **(2014)**

[5] Rothe R., Guillaumin M., Van Gool L. *Non-maximum Suppression for Object Detection by Passing Messages Between Windows.* In: Cremers D., Reid I., Saito H., Yang MH. (eds) Computer Vision – ACCV 2014. Lecture Notes in Computer Science, vol 9003. Springer, Cham, **(2015)**

[6] Ren, S., He, K., Girshick, R., et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* arXiv e-prints, arXiv:1506.01497 **(2015)**