YOLO Object Detection Report

Haosong Chen, Xinran Xu, Yuxuan Xing, Shengzuo Yang, Yibo Yu

Contents

1 Application

YOLO, which stands for you only look once, is a real-time object detection network. Unlike other region proposal classification networks(RCNN), which performs their algorithm on multiple proposal region of a single picture. YOLO will pass the image into the network and predict bounding boxes as well as predict classes in one run. As a result, YOLO is much faster than the traditional approach.

The input image is cut into grid cells, typically 19x19. Each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × B x (5 + C) tensor. For example, if there are 2 bounding boxes and 3 output classes, this is will be the prediction for a single grid cell:

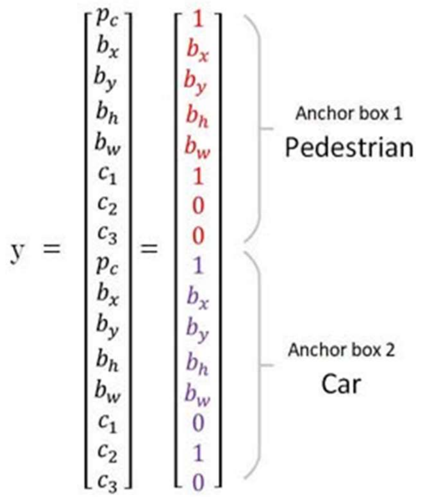


Figure 1 The output data structure

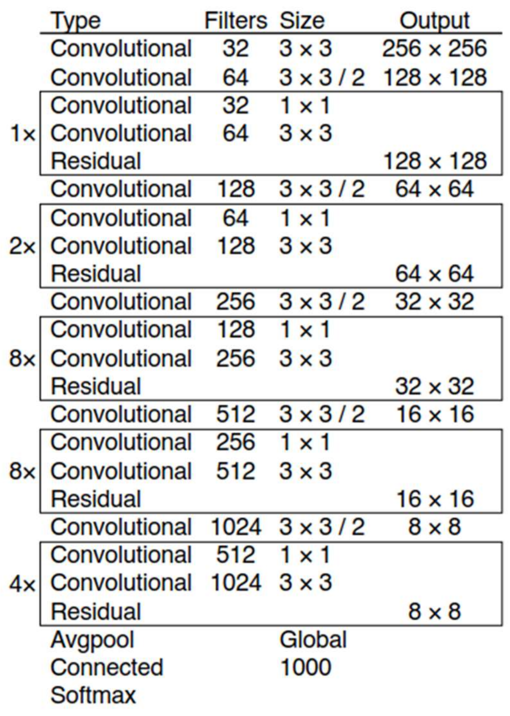
pc is the confidence of the box, bx by is the coordinate of the bounding box’s central point, bw bh are the bounding box’s width and height. And c1, c2, c3 are those three classes we want to detect.

For YOLO, an object may be detected multiple times and thus generate many redundant bounding boxes. To clean all these unnecessary bounding boxes, non-max suppression will be implemented. Non-max suppression will first look into the bounding box associated with the highest possibility of detection, then it will get rid of other bounding boxes which have high intersection-of-union(IOU) with the ground truth bounding box.

2 Network

Here is a figure of YOLO V3's network architecture, which is the third object detection algorithm in YOLO family. YOLO V3 has totally 252 layers, including 1 input layer, 72 Leaky ReLU layers, 2 upsampling layers, 5 zero padding layers, 23 add layers, 72 batch normalization layers, 2 concatenate layers, and 75 convolutional layers as its most powerful tool. No fully-connected layer is used. This structure makes it possible to deal with images with any sizes. Also, no pooling layers are used. Instead using convolutional layer with stride 3 is used to downsample the feature map, passing size-invariant feature forwardly.

Yolov3 use Darknet-53 as backbone to perform feature extraction. The figure 2 shows the architecture of Darknet-53, the new network uses successive 3X3 and 1X1 convolutional layers and we also use residual blocks here, this ResNet-alike structure is helpful to enhance the accuracy.



	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 3 Yolo V3 Darknet-53 Architecture

YOLO v3 predicts boxes at 3 different scales. It is desirable for the network to detect different size of objects. As the network goes deeper, it is harder to detect smaller objects because the feature map gets smaller. Therefore, it is necessary to detect different feature maps before small objects end up disappearing.

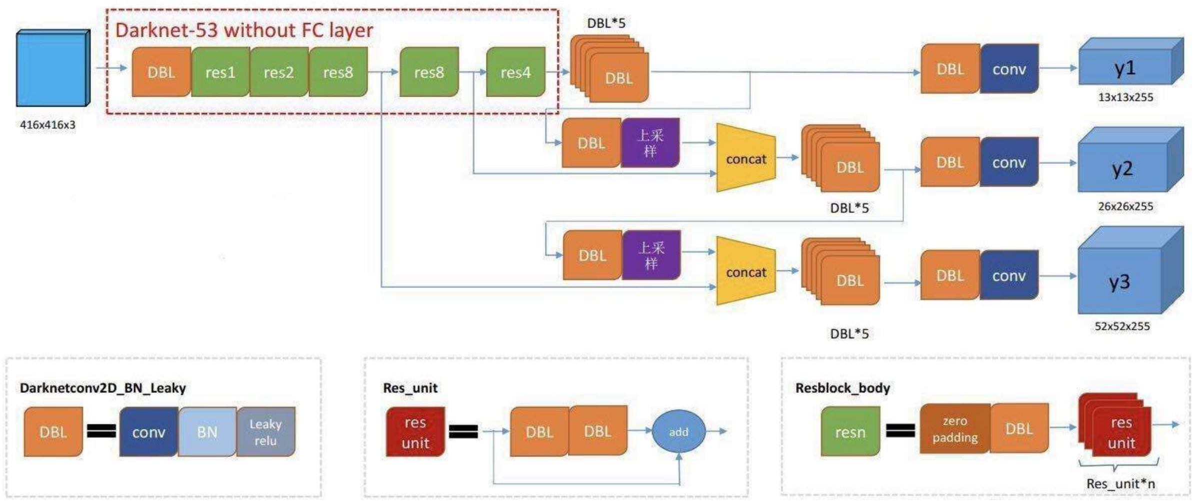
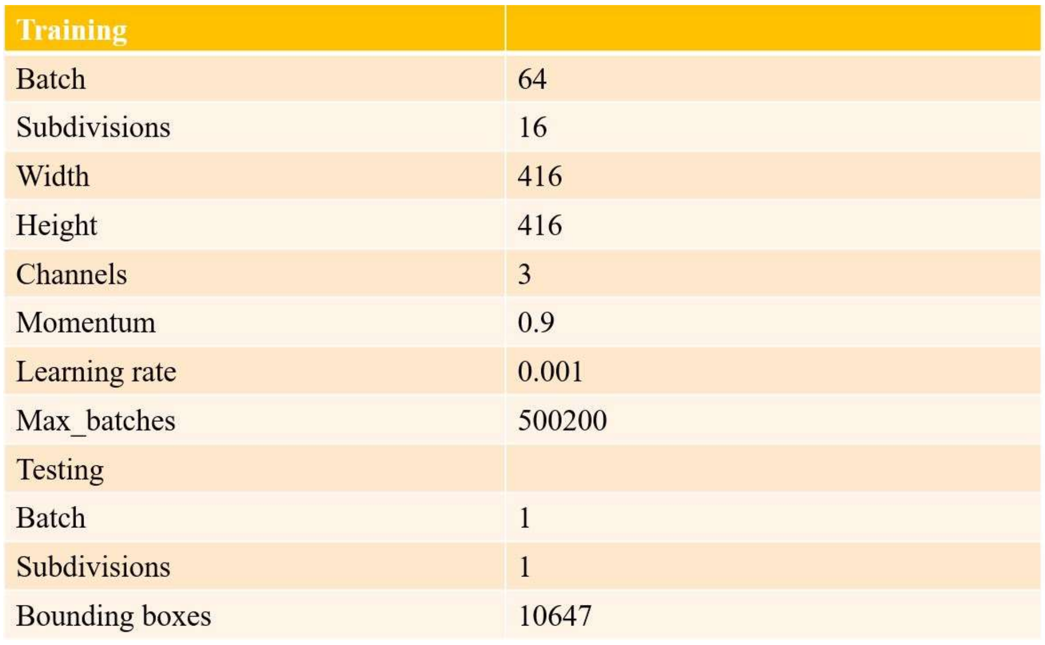


Figure 3 Yolo V3 network Architecture

3 Parameters of Yolo V3

3.1 Yolo V3 Parameter



Training	
Batch	64
Subdivisions	16
Width	416
Height	416
Channels	3
Momentum	0.9
Learning rate	0.001
Max_batches	500200
Testing	
Batch	1
Subdivisions	1
Bounding boxes	10647

Figure 4 Yolo V3 Parameter

3.2 Analysis of trained parameters and connections

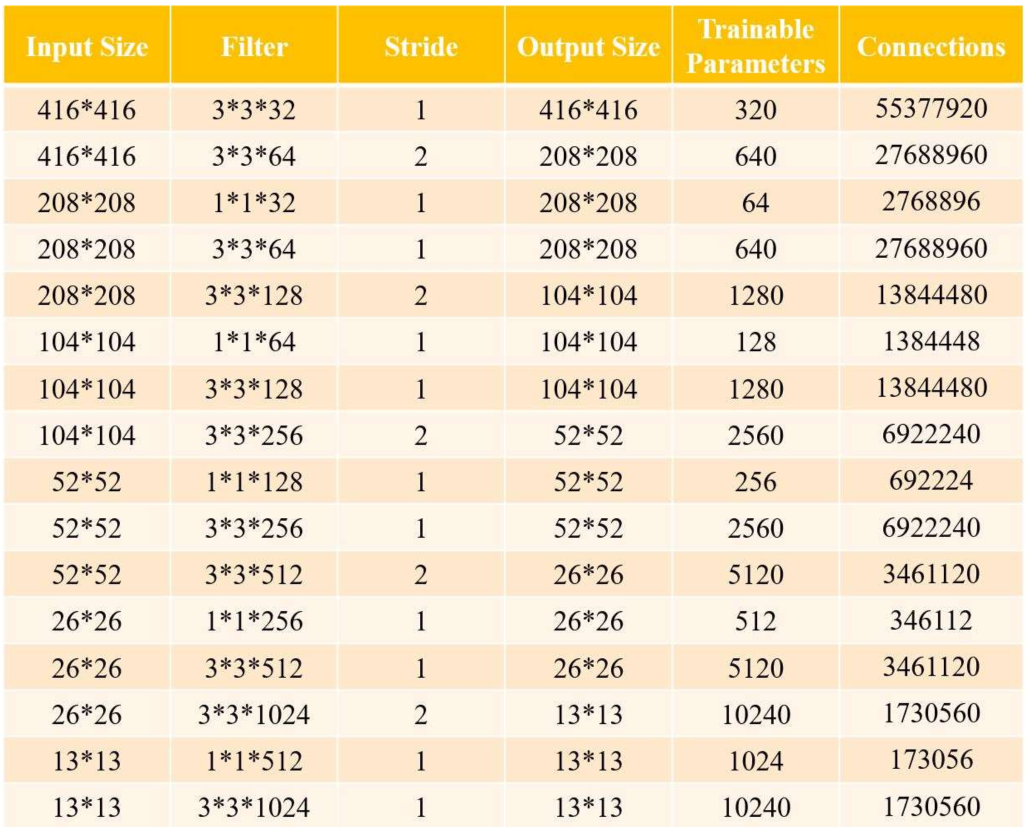
Based on the calculation method introduced in the course CSE691, for the convolutional layer.

If the size of input layer is W1×H1 and the filter size is F×F, the number of filters is K, stride is S the output size is W2×H2

The trained parameter will be (F×F+1) ×K

The total connection will be (F×F+1) ×K×W2×H2

Since for Yolo V3, it has 53 convolutional layers called Darknet-53, it is enough to just calculate all the convolutional layers for analysis. The initial input size is 416×416. So the results of calculated trained parameters and connections are shown in this table



Input Size	Filter	Stride	Output Size	Trainable Parameters	Connections
416*416	3*3*32	1	416*416	320	55377920
416*416	3*3*64	2	208*208	640	27688960
208*208	1*1*32	1	208*208	64	2768896
208*208	3*3*64	1	208*208	640	27688960
208*208	3*3*128	2	104*104	1280	13844480
104*104	1*1*64	1	104*104	128	1384448
104*104	3*3*128	1	104*104	1280	13844480
104*104	3*3*256	2	52*52	2560	6922240
52*52	1*1*128	1	52*52	256	692224
52*52	3*3*256	1	52*52	2560	6922240
52*52	3*3*512	2	26*26	5120	3461120
26*26	1*1*256	1	26*26	512	346112
26*26	3*3*512	1	26*26	5120	3461120
26*26	3*3*1024	2	13*13	10240	1730560
13*13	1*1*512	1	13*13	1024	173056
13*13	3*3*1024	1	13*13	10240	1730560

Figure 5 Trained parameter and connection for yolo v3

The sum of trained parameters is 41984. The sum of connections is 168037376

4 Structure of different packages and files used in our Project

4.1 The comparison of two dataset

There are two type of dataset that could be used in objection detection-Pascal Voc dataset and Coco dataset.

Pascal voc dataset: The following packages are included in voc datasets: JPEGImages, Annotations, Imagesets and SegmentationObjects & Segmentation classes. JPEGImages mainly provides all image data that is used for training, testing and validation. Annotations store the label file in XML format and each xml response to a picture in JPEGImage. Imageset includes action, layout and main packages (store 20 classes data. SegmentationObject is used for store the dataset after segmentation and do not used in object detection.

Coco dataset is one that the microsoft team can used to recognition + segmentation+captioning dataset. The datasets target the sense understanding, which is mainly intercepted from complex daily senses. The target in the image are calibrated by precise segmentation. The image includes 91 categories of targets, 328000 images and 2500000 labels. The datasets mainly solves 3 problems: target detection, context between target and precise positioning in 2 dimensions of the target.

Coco datasets contains 91 categories. Although the fewer categories than imageNet and Sun, there are many pictures in each category, which is beneficial to get more ability in certain kind of sense in each categories. Compared to voc dataset, there exists more classes and images. However, Coco is more difficult because it has large number of objects in each images, result in relatively low accuracy of detection of the datasets.

4.2 K-means used in yolo

K-means algorithms is used for the calculation of anchor box, which is used in yolo v2 and yolo v3.

In yolo v2, it use anchor box to predict bounding box. The anchor box in yolo v2 is different from the faster R-CNN. It is not a pre-selected a priori box and it is obtained by K-means. The yolo tag file for k-means is as follow:
<object class> <x> <y><width> <height>.

Object class is index of the class: x is the x coordinate of the center of the ROI, y is the y coordinate of the center of ROI, <weight> is the weight of ROI and <height> is the height of ROI. The conventional neural network has translation invariance and the position of the anchor box is fixed by each grid. So we only need to calculate the weight and height of anchor box through k-means and we do not need x, y, object class value. Yolo v2 directly predicts the coordinates of the bounding box through the anchor box, the coordinates are relatively to the slide length of the grid. So the weight and height of the anchor box is converted to the ratio relative to the slide of the grid. The convert formula is the following: *w=anchor_width*input_width/downsamples, h=anchor_height* input height/downsamples.*

If we use Euclidean distance, the large bounding box will have more error than small bound box. The aim of k-means cluster is we use anchor box to make sure the ground truth and anchor box has better IOU score, which is not related to the size of box. Here is the formula: *d(box,centroid)=1-IOU(box, centroid).* And we set initial anchor box x, y coordinate to be 0 to make sure each box is at same position.

The code is different in yolo v2 and yolo v3. In yolo v2, configuration file yolov2.cfg needs the anchors to be relative feature maps. The values are small and less than 13. In configuration of yolo v3, yolo v3.cfg requires 3 anchors and relative large than the original image. Also, the size of input image contributes to output size.

4.3 Other important packages or files used in yolo v3

4.3.1 Yolo video.py

provides two interface for image detection or video detection. It depends on the tag of our commend, if we input --image, it will calls detect_img() function in yolo.py, which is used for the detection of single image. If we input --input in command interface, it will calls detect_video() function in yolo.py for the video detection.

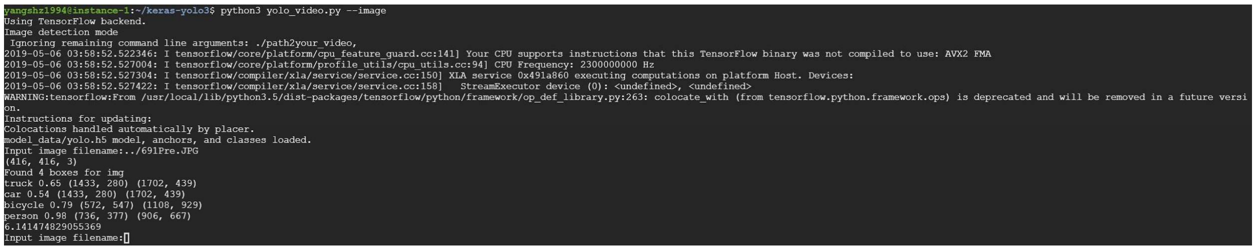
4.3.2 Yolo.py

It provides several function for detection like get_classes, get_anchors, generate, detect_image and detect_video. “Generate” function is used for generate colors for drawing bounding box or generate output tensor targets for filtered bounding box.”detect_image” function is used for image detection. “Detect_video” function is used for video detection. It use openCV package to capture video and consider the video as a sequence of images at different time span and use detect_image function to capture the correspond segmented image at different time continuously.

4.3.3 Train.py

It is used for model training and testing, which includes two versions yolo v3 and tiny yolo v3. The two different version has different hyperparameter and data size.

5 Demo of Yolo V3 code



```
yangshz1994@instance-1:~/keras-yolo3$ python3 yolo_video.py --image
Using TensorFlow backend.
Image detection mode
 Ignoring remaining command line arguments: ./path2your_video,
2019-05-06 03:58:52.522346: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-05-06 03:58:52.527004: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2300000000 Hz
2019-05-06 03:58:52.527304: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x491a860 executing computations on platform Host. Devices:
2019-05-06 03:58:52.527422: I tensorflow/compiler/xla/service/service.cc:158]   StreamExecutor device (0): <undefined>, <undefined>
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Colocations handled automatically by placer.
model_data/yolo.h5 model, anchors, and classes loaded.
Input image filename:../691Pre.JPG
(416, 416, 3)
Found 4 boxes for img
truck 0.65 (1433, 280) (1702, 439)
car 0.54 (1433, 280) (1702, 439)
bicycle 0.79 (572, 547) (1108, 929)
person 0.98 (736, 377) (906, 667)
6.141474829055369
Input image filename:
```

Figure 6 Screenshot of Execution of Program

The original picture in jpg form:

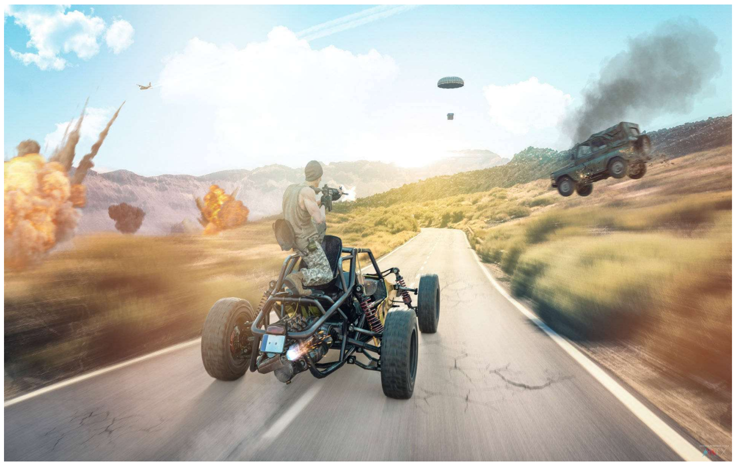


Figure 7 Original Plug graph

The output result picture after detected:

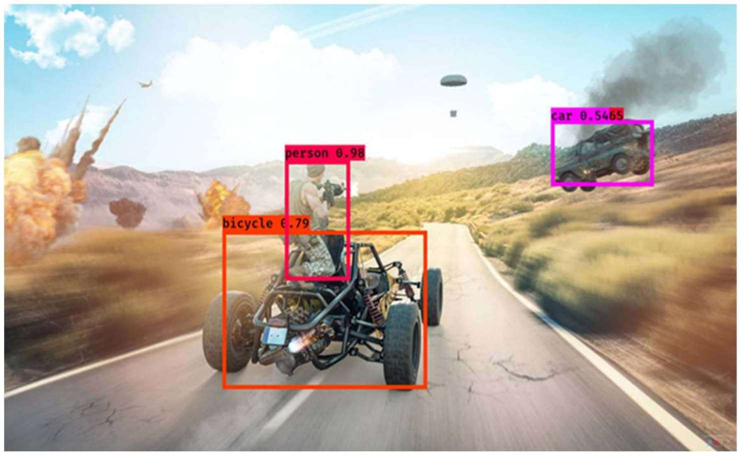
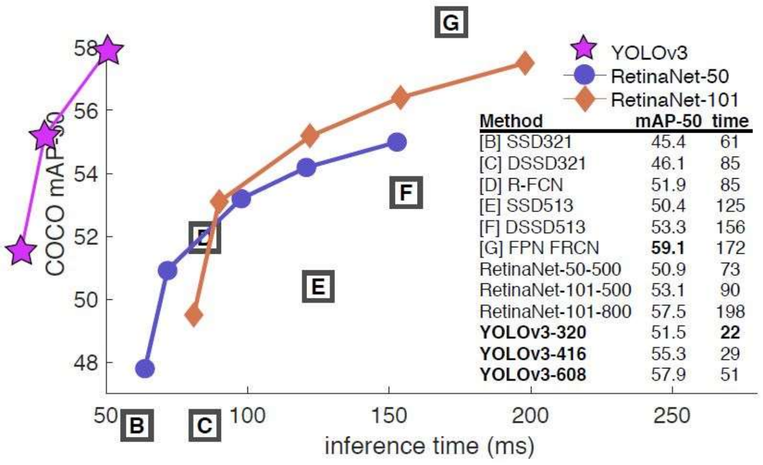


Figure 8 Yolo Detection result for the picture

6 Compare with other detection methods

Yolo v3 is a good detector. It is quite fast and accurate. Although it is not so great on the COCO average AP between 0.5 and 0.95 IOU metrics. It is very good on the old detection metric of 0.5 IOU.

Here is a diagram shows one plot of mAP versus inference time and one table of different detection methods.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 9 Plot of mAP versus inference time of different detection method

From this diagram it could be figured out that Yolo v3 is quite excellent. Yolo v3 runs significantly faster than most detection methods with comparable performance. At resolution 320*320 Yolo v3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. It is still quite a bit behind RetinaNet in this metric though but much faster. In conclusion, it could say that Yolo V3 is faster and better than other detection methods on mAP-50 metrics.

7 Use Our Own Annotated Data to train the model

We trained our network to detect object - NFPA 704 ‘fire diamond’.[1]

7.1 Data Annotation

We used annotation tool to annotate the objects of images.

We used scripts to convert the annotated images into txt files.

We Created txt files for every images, containing information of the objects.

Every txt file contains:

- <object-class>: integer number of object from 0 to (class - 1)
- <x><y><width><height>: float values of center coordinate, width and height of object. they are normalized to be equal from 0.0 to 1.0

A txt file looks like this:

```
0 0.73 0.507380073800738 0.395 0.5940959409594095
0 0.48375 0.48 0.7575000000000001 0.74
0 0.49353448275862066 0.6915887850467289 0.41810344827586204 0.616822429906542
```

Figure 2 Data annotate structure

7.2 Training Configuration

We need three configuration files to start our training process

7.2.1 cfg/nfpa.data

This file contains information of resource, input and output of training process.

```
classes = 1
train = train.txt
valid = test.txt
names = nfpa.names
backup = backup/
```

7.2.2 cfg/nfpa.names

This file contains classes that we plan to train.

```
nfpa
```

7.2.3 cfg/nfpa_yolo.cfg

We copied the tiny_yolo.cfg and configure it to fit our training requirement.

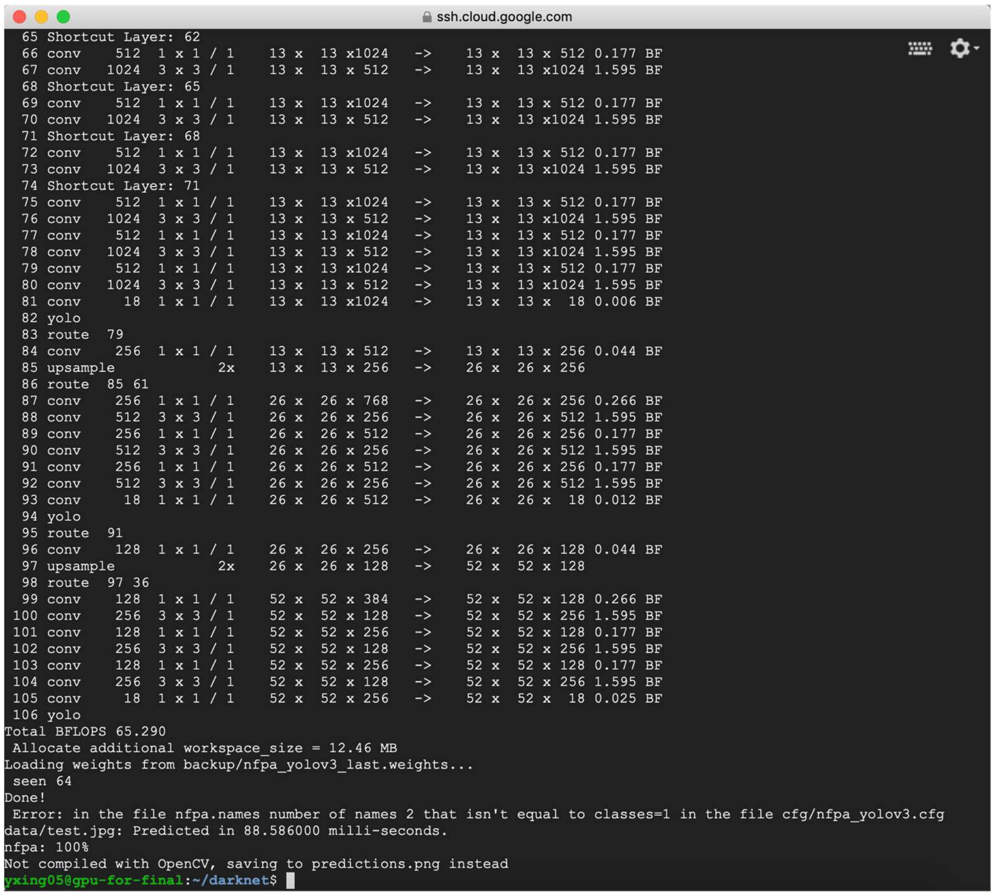
7.3 Training Process

When completed 100 iteration it will automatically store weights file and kill the process once the average loss is less than 0.06 to get good a accuracy.

7.4 Trained Model Test

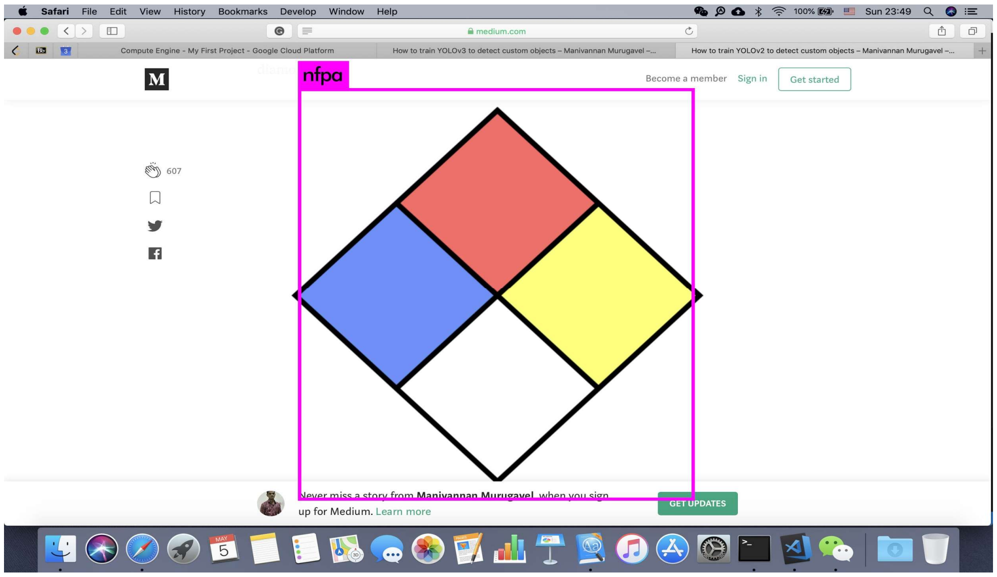
The trained model was pushed to github, it can be found here.

We tested the trained model by our own image. the result are as follows.



```
 65 Shortcut Layer: 62
 66 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
 67 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
 68 Shortcut Layer: 65
 69 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
 70 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
 71 Shortcut Layer: 68
 72 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
 73 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
 74 Shortcut Layer: 71
 75 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
 76 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
 77 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
 78 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
 79 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
 80 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
 81 conv     18  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x  18 0.006 BF
 82 yolo
 83 route  79
 84 conv    256  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 256 0.044 BF
 85 upsample            2x    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 0.266 BF
 88 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
 89 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
 90 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
 91 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
 92 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
 93 conv     18  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x  18 0.012 BF
 94 yolo
 95 route  91
 96 conv    128  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x 128 0.044 BF
 97 upsample            2x    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 0.266 BF
100 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
101 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
102 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
103 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
104 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
105 conv     18  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x  18 0.025 BF
106 yolo
Total BFLOPS 65.290
 Allocate additional workspace_size = 12.46 MB
Loading weights from backup/nfpa_yolov3_last.weights...
 seen 64
Done!
 Error: in the file nfpa.names number of names 2 that isn't equal to classes=1 in the file cfg/nfpa_yolov3.cfg
data/test.jpg: Predicted in 88.586000 milli-seconds.
nfpa: 100%
Not compiled with OpenCV, saving to predictions.png instead
yxing05@gpu-for-final:~/darknet$ 
```

Figure 11 Screenshot of trained process with our own dataset


Safari File Edit View History Bookmarks Develop Window Help
Sun 23:49
medium.com
Compute Engine – My First Project – Google Cloud Platform
How to train YOLOv3 to detect custom objects – Manivannan Murugavel –...
How to train YOLOv2 to detect custom objects – Manivannan Murugavel –...
nfpa
Become a member
Sign in
Get started
607
GET UPDATES

Never miss a story from Manivannan Murugavel, when you sign up for Medium. Learn more

Figure 12 Use test data for detection

8 Conclusion and Future Research

Yolo is used for object detection in varieties of fields, like image detection, video detection and camera real-time detection. The performance has greatly enhanced from Yolo v1 to yolo v3.

In our paper, we show our understanding of Yolo object detection technology, especially in yolo v3. We introduce some important technology in yolo v3 like anchor box, bounding box and so on. We analysis the structure of yolo v3 and compare with yolo v1 and yolo v2. In training and testing process, we describe the loss function, hyperparameter of different layer and so on. We download some pictures from website and use image label package to label our own dataset. And then use our own dataset for training and detection. We also evaluate the performance of yolo compared to other object detect function and analysis memory requirement of yolo v3(the trainable parameter) and introduce different packages that is useful in object detection.

Yolo v3 is a mature technology that overcome some shortcomes in yolo v1 and yolo v2. We have some ideas for improvement of yolo v3: Firstly, we wish to try focal loss for dense objection. Secondly, we wish to use linear prediction for the x,y of anchor box instead of using logical regression. Thirdly, we are planning to use denseNet instead of resNet instead of denseNet. DenseNet is a convolutional neural network with dense connection, there is direct connection between any two layers which means that the input of the union of the output of all previous layers and the feature map learned by the layer is transmitted to all layers behind it used as input, which incorporate the advantage of resNet and has some improvement for performance.

9 Reference

[1] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]//European conference on computer vision. Springer, Cham, 2016: 21-37.

[2] Redmon J, Farhadi A. YOLO9000: better, faster, stronger[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 7263-7271.

[3]. https://github.com/qqwweee/keras-yolo3

[4]. https://github.com/AlexeyAB/darknet.git