

CS_IOC5008_0856043_HW3 Report

[Github link](#)

Brief introduction

1. Development environment

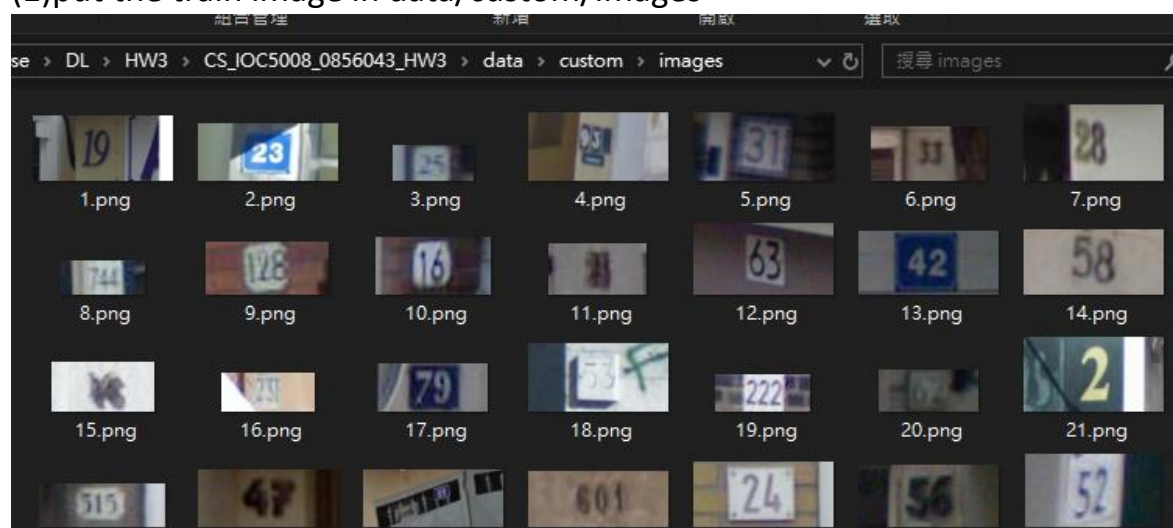
Python version : 3.7.4

Framework : Pytorch

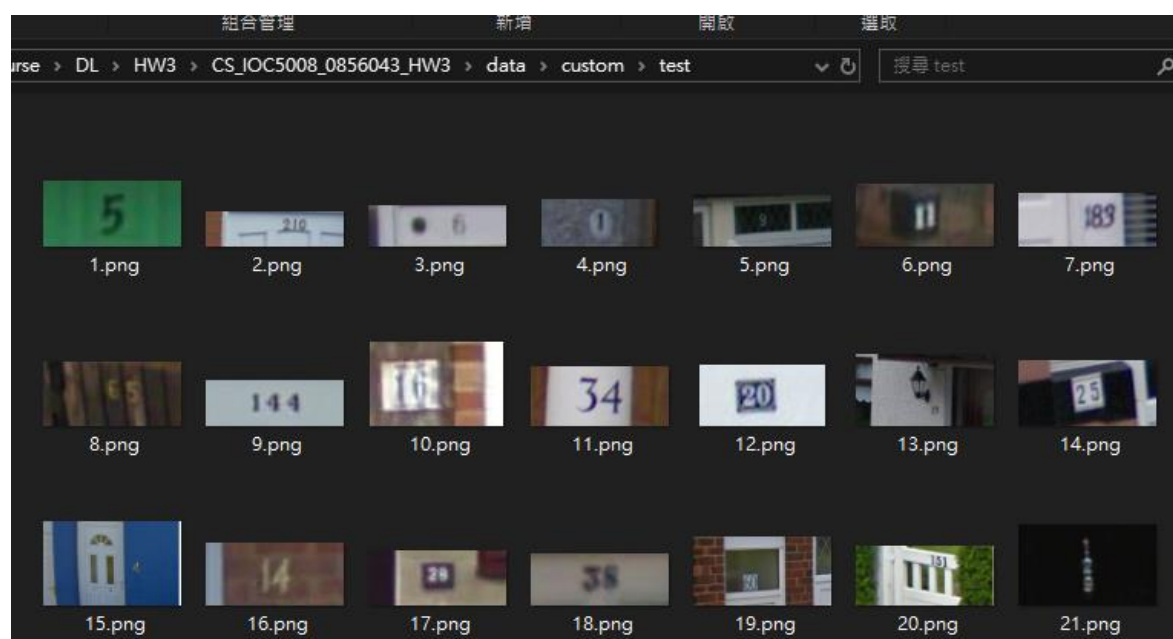
Hardware : NVIDIA GeForce GTX 1080 Ti 11GB

2. How to run the code.

(1)put the train image in data/custom/images



(2)put the test images in data/custom/test

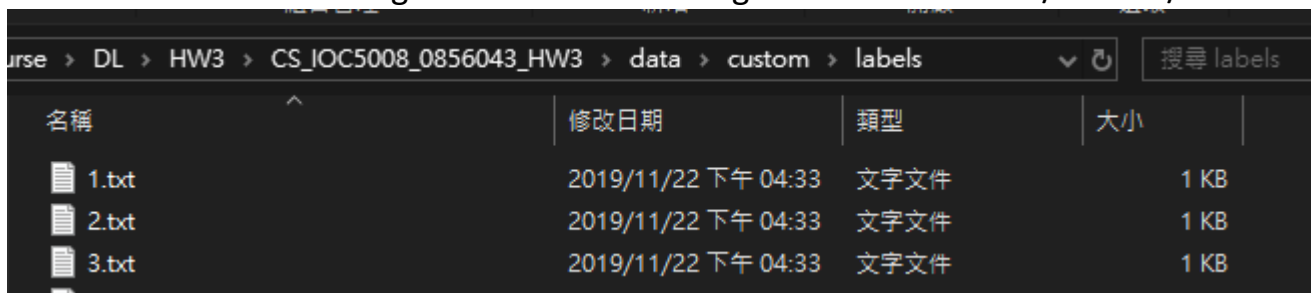


(3) analysis the digitStruct.mat file.

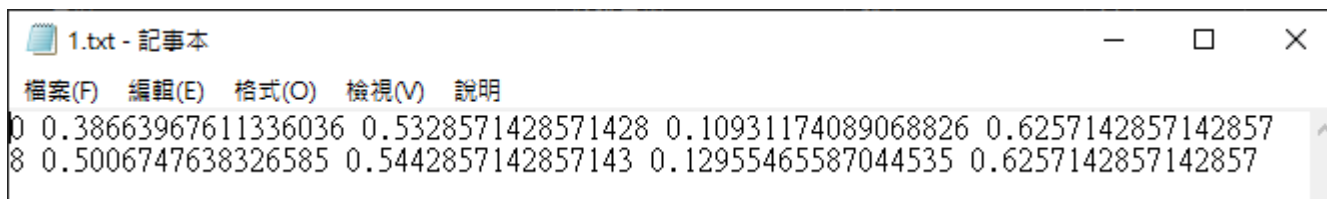
Run data/custom/parsedata.py

```
parsedata.py
128 def makeTxt():
129     dsFileName = './digitStruct.mat'
130     testCounter = 0
131     for dsObj in yieldNextDigitStruct(dsFileName):
132
133         fp = open("./labels/"+dsObj.name[:-4]+'.txt', "a")
134         im=Image.open('./images/'+dsObj.name)
135         x = im.size[0]
136         y = im.size[1]
137         for bbox in dsObj.bboxList:
138
139             fp.write("{} {} {} {} {}{}\n".format(
140                 bbox.label-1, bbox.left/x+bbox.width/(2*x), bbox.top/y+bbox.height/(2*y), bbox.width/x, bbox.height/y))
141         fp.close()
142         if testCounter >= 5:
143             break
144         print('done')
145
```

It will make each test images one txt file for the ground-truth in data/custom/labels.



名稱	修改日期	類型	大小
1.txt	2019/11/22 下午 04:33	文字文件	1 KB
2.txt	2019/11/22 下午 04:33	文字文件	1 KB
3.txt	2019/11/22 下午 04:33	文字文件	1 KB

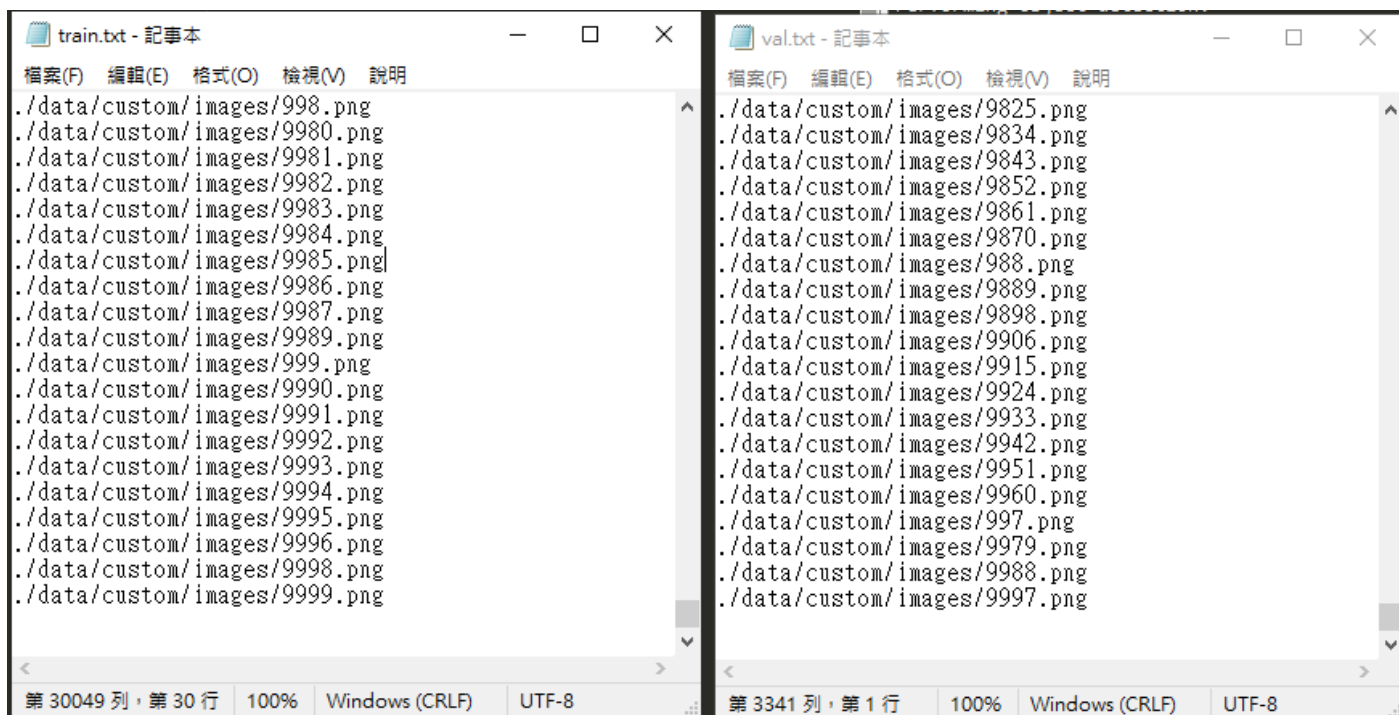


```
0 0.38663967611336036 0.5328571428571428 0.10931174089068826 0.6257142857142857
8 0.5006747638326585 0.5442857142857143 0.12955465587044535 0.6257142857142857
```

(4)Spilt the train data using maketxt.py

It will store 90% data in data/custom/train.txt for train-set and 10% in data/custom/val.txt for validation-set.

```
maketxt.py
1 import glob
2 import os
3 import numpy as np
4 import sys
5 current_dir = "./data/custom/images"
6 split_pct = 10 # 10% validation set
7 file_train = open("data/custom/train.txt", "w")
8 file_val = open("data/custom/val.txt", "w")
9 counter = 1
10 index_test = round(100 / split_pct)
11 for fullpath in glob.iglob(os.path.join(current_dir, "*.png")):
12     title, ext = os.path.splitext(os.path.basename(fullpath))
13     if counter == index_test:
14         counter = 1
15         file_val.write(current_dir + "/" + title + '.png' + "\n")
16     else:
17         file_train.write(current_dir + "/" + title + '.png' + "\n")
18         counter = counter + 1
19 file_train.close()
20 file_val.close()
```



(5)Run train.py

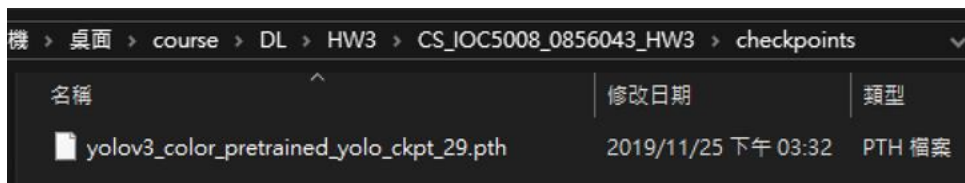
Change the pretrained_weights to train from the pretrained network.

```

59 # If specified we start from checkpoint
60 #opt.pretrained_weights="checkpoints/yolov3_color_pretrained_yolo_ckpt_29.pth"
61 opt.pretrained_weights="weights/yolov3.weights"

```

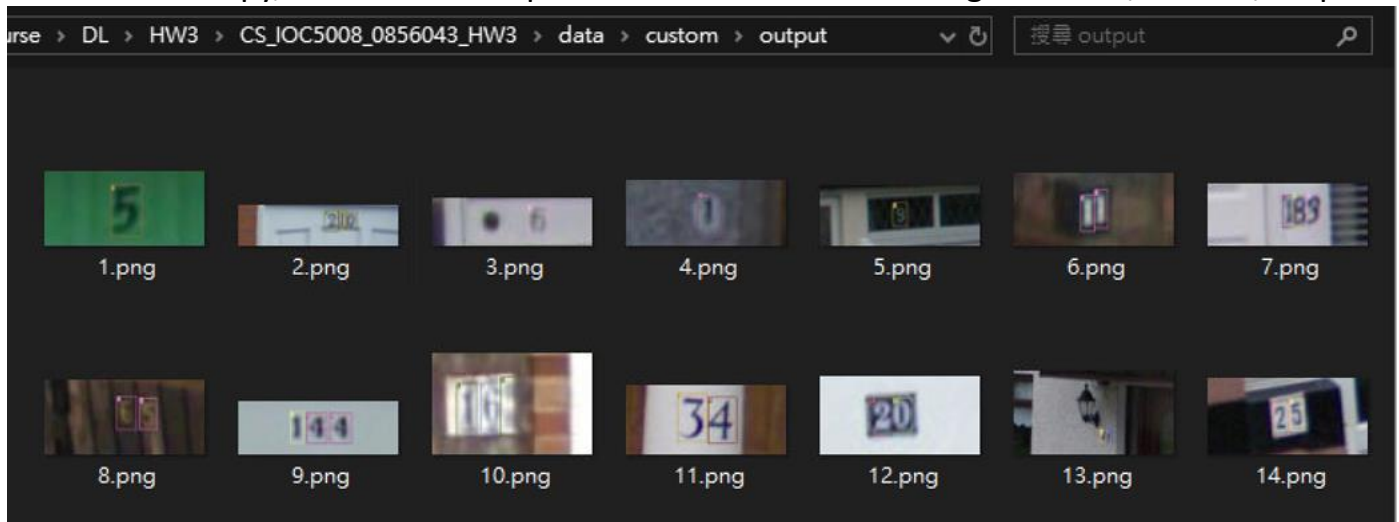
And then run this file

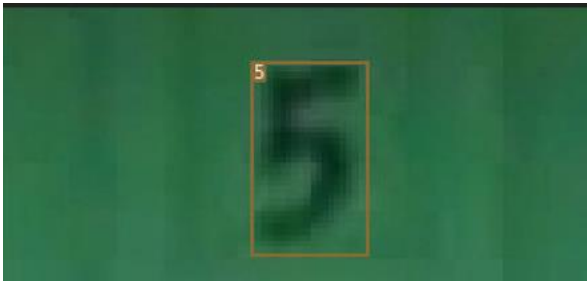


After train, it will create .pth file and we can do some test or predict the result.

(6)detect

Run the detect.py, it will draw the predict result on the test images in data/custom/output





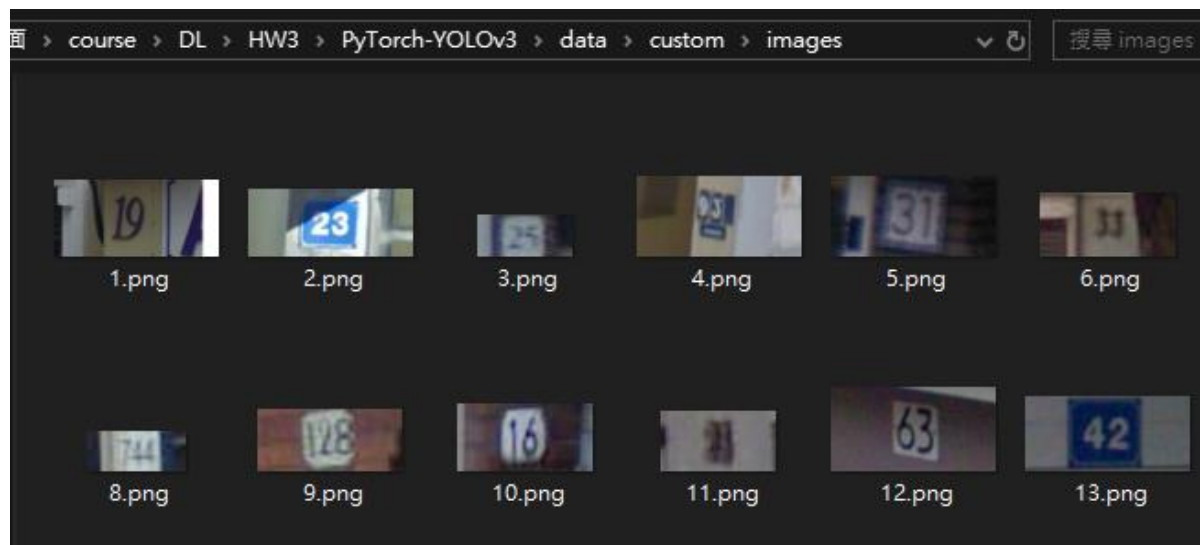
(7)Print Json

Run printjson.py and it will create the result that dump into the json form in data/custom/0856043_.json

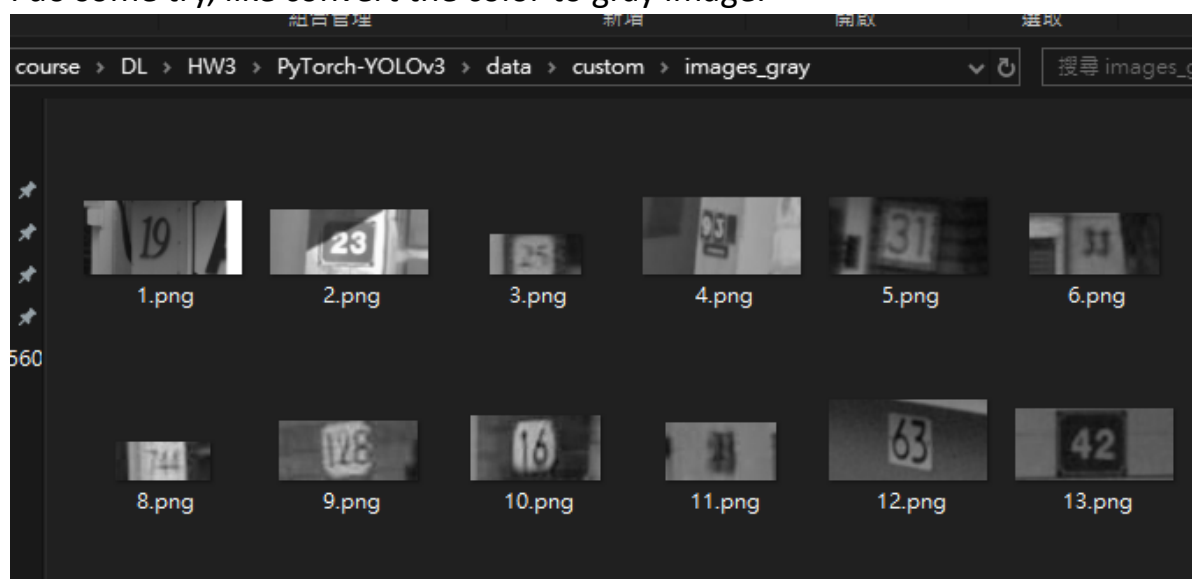
```
0856043_.json x
1 [
2   {
3     "bbox": [
4       [
5         9,
6         41,
7         41,
8         60
9       ]
10    ],
11    "label": [
12      5
13    ],
14    "score": [
15      0.998767614364624
16    ]
17  },
18 ]
```

Methodology

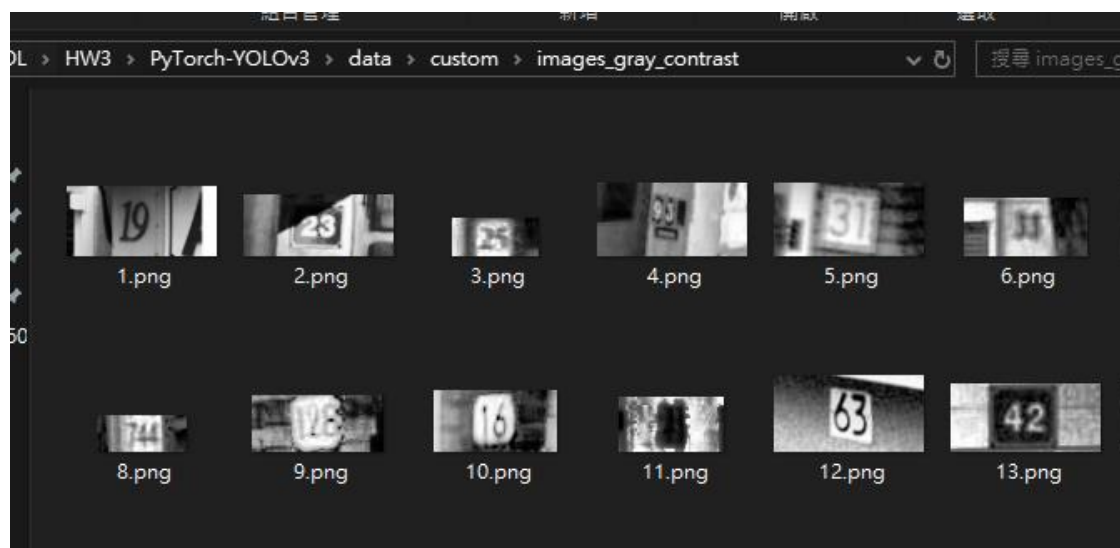
1. Data preprocess



I do some try, like convert the color to gray image.



And increase the contrast.



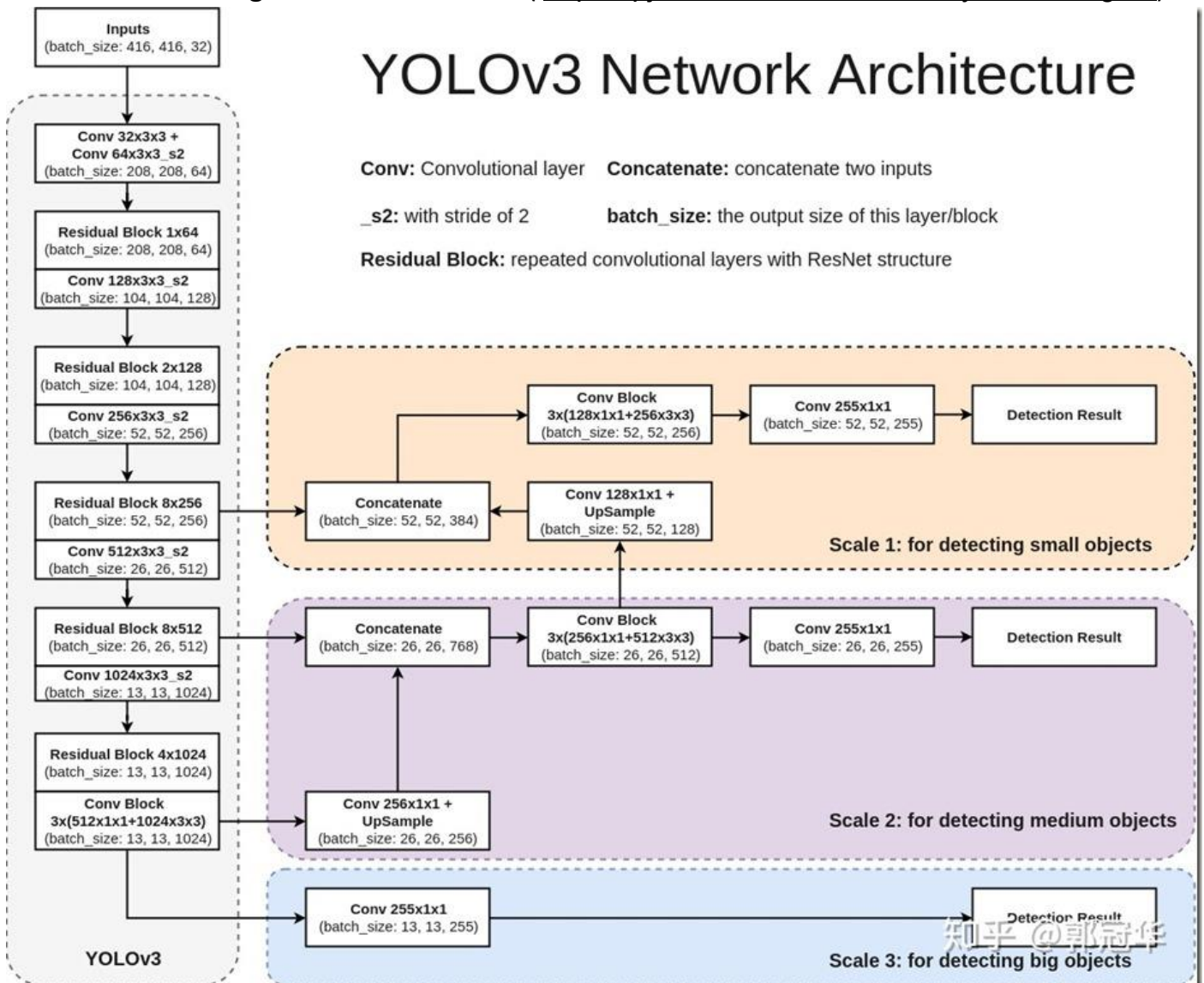
But they don't increase the mAP, so I just use the original images.
And also use do some random flip.

```
# Apply augmentations
if self.augment:
    if np.random.random() < 0.5:
        img, targets = horizontal_flip(img, targets)
```

2. Models

I use yolov3 as my model.

Pre-trained on imagenet+Coco datasets.(<https://pjreddie.com/media/files/yolov3.weights>)



Compare to yolov2, version 3 use 3 different feature (13*13, 26*26, 52*52) to detect big, medium, and small objects.

3. Hyperparameters

Image size: 416*416

Loss function: binary cross-entropy loss

Output activation: Logistic loss

Base model: ResNet

Darknet model: darknet-53

learning_rate: 0.001

batch_size: 8

epochs: 30

Model speed benchmark

Upload my weight and test code into Colab.

```
[32] imgs = [] # Stores image paths
img_detections = [] # Stores detections for each image index
total_times=0
print("\nPerforming object detection:")
prev_time = time.time()
for batch_i, (img_paths, input_imgs) in enumerate(dataloader):
    # Configure input
    input_imgs = Variable(input_imgs.type(Tensor))
    # Get detections
    with torch.no_grad():
        detections = model(input_imgs)
        detections = non_max_suppression(detections, conf_thres, nms_thres)
    # Log progress
    current_time = time.time()
    inference_time = datetime.timedelta(seconds=current_time - prev_time)
    time_v = current_time - prev_time
    prev_time = current_time
    print("\t+ Batch %d, Inference Time: %s" % (batch_i, inference_time))
    total_times += time_v

    # Save image and detections
    imgs.extend(img_paths)
    img_detections.extend(detections)

print("average time:", total_times/200)
```

Test for 200 images

```
+ Batch 189, Inference Time: 0:00:00.027554
+ Batch 190, Inference Time: 0:00:00.029346
+ Batch 191, Inference Time: 0:00:00.027081
+ Batch 192, Inference Time: 0:00:00.027916
+ Batch 193, Inference Time: 0:00:00.030166
+ Batch 194, Inference Time: 0:00:00.027901
+ Batch 195, Inference Time: 0:00:00.027860
+ Batch 196, Inference Time: 0:00:00.027483
+ Batch 197, Inference Time: 0:00:00.029546
+ Batch 198, Inference Time: 0:00:00.027969
+ Batch 199, Inference Time: 0:00:00.027547
average time: 0.029423400163650512
```

Average time=29.4ms

Test for 100 images

```
+ Batch 92, Inference Time: 0:00:00.028591
+ Batch 93, Inference Time: 0:00:00.030214
+ Batch 94, Inference Time: 0:00:00.028774
+ Batch 95, Inference Time: 0:00:00.028023
+ Batch 96, Inference Time: 0:00:00.028052
+ Batch 97, Inference Time: 0:00:00.028789
+ Batch 98, Inference Time: 0:00:00.026763
+ Batch 99, Inference Time: 0:00:00.028151
average time: 0.0297226619720459
```

Average time=29.7ms

Test for 1000 images

```
+ Batch 991, Inference Time: 0:00:00.033592
+ Batch 992, Inference Time: 0:00:00.028442
+ Batch 993, Inference Time: 0:00:00.027667
+ Batch 994, Inference Time: 0:00:00.027280
+ Batch 995, Inference Time: 0:00:00.027413
+ Batch 996, Inference Time: 0:00:00.028444
+ Batch 997, Inference Time: 0:00:00.027150
+ Batch 998, Inference Time: 0:00:00.028098
+ Batch 999, Inference Time: 0:00:00.028050
average time: 0.028286385536193847
```

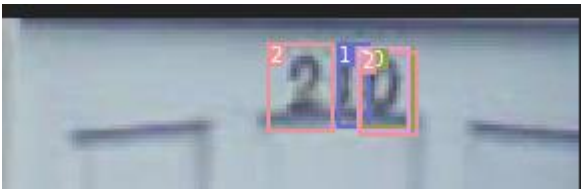
Average time=28.2ms

Findings

I think this task is very difficult because the image has some many other things that we don't want to predict.



And it also very easy to predict overlapping.



So I also do some work to prevent this thing happened.

I use the concept of IOU.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Object

Intersection

Detected box

Object

Detected box

When the bounding-box that have a big confident, we can delete the other when IOU value greater than some threshold.



Here is the result.

But things got weird, when I use this idea in the whole prediction, the score of using IOU is always worse than not doing anything...

gray contrast	0.38461
gray contrast count iou	0.37262
color pretrained_darknet iou	0.42054
color pretrained_darknet	0.42929
gray contrast pretrained_darknet	0.39479
gray contrast pretrained_darknet iou	0.38529
color pretrained_yolov3 9	0.44315
color pretrained_yolov3 9 iou	0.43667
color pretrained_yolov3 19	0.45788
color pretrained_yolov3 19 iou	0.45259
color pretrained_yolov3 29	(1)0.46305
color pretrained_yolov3 29 iou	0.45732

Here is my grade records. The scores that using iou method to delete overlapping objects always get about 1% lower than original.

Reference

[Yolov3-pytorch-implement](#)

[Train Yolo on my dataset\(Medium article\)](#)

[Yolov3 explain\(Chinese\)](#)

[Yolo website](#)