CS_IOC5008_0856043_HW4 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.

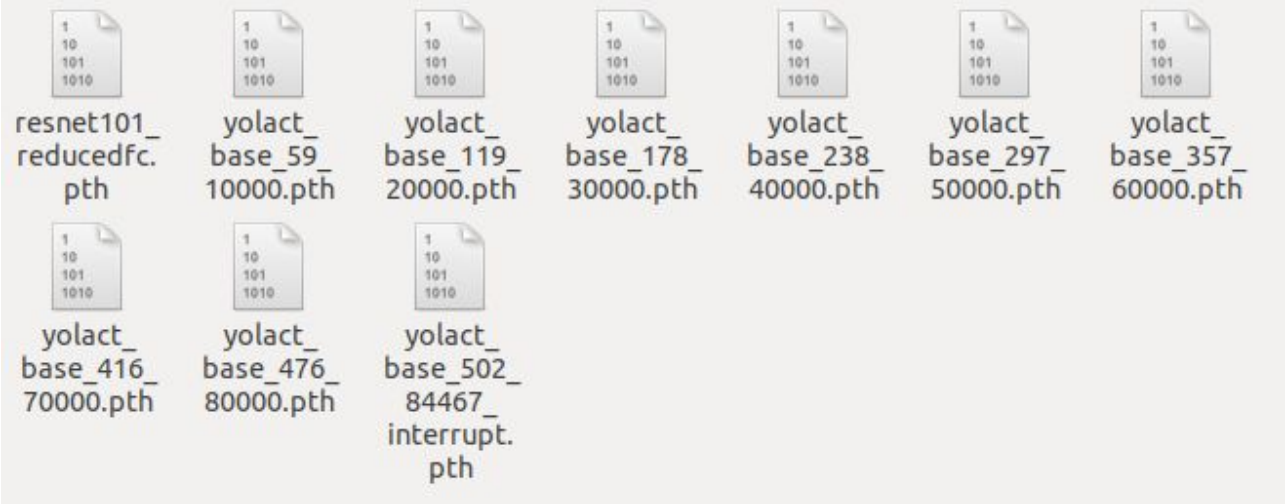
(1)Setup the environment.

- Set up a Python3 environment.
- Install Pytorch 1.0.1 (or higher) and TorchVision.
- Install some other packages:

```
# Cython needs to be installed before pycocotools
pip install cython
pip install opencv-python pillow pycocotools matplotlib
```

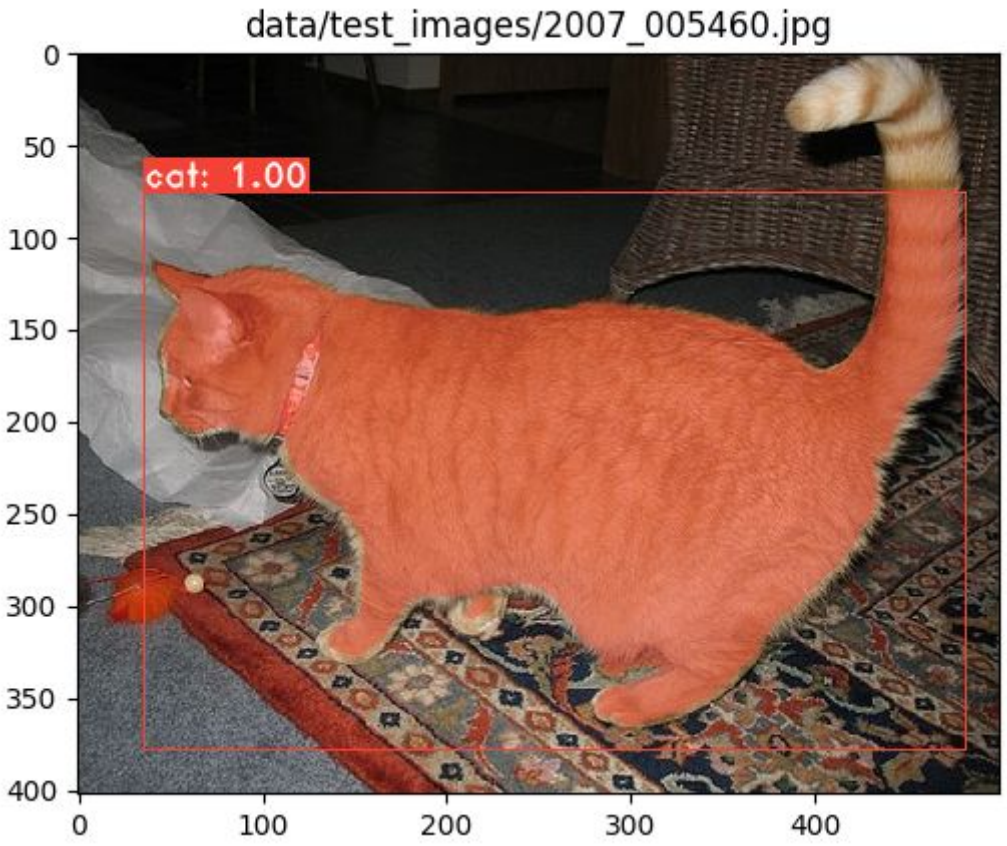
(2)Prepared an imagenet-pretrained model and put it in ./weights
Prepared testing images in data/train_images
Prepared gound truth file in data/pascal_train.json
I use Pretrained on ImageNet dataset and ResNet101 based.

(3)Run
"python train.py --config=yolact_base_config --batch_size=5"
It will create .pth files in the ./weights every 10000 iterators and after interruption.

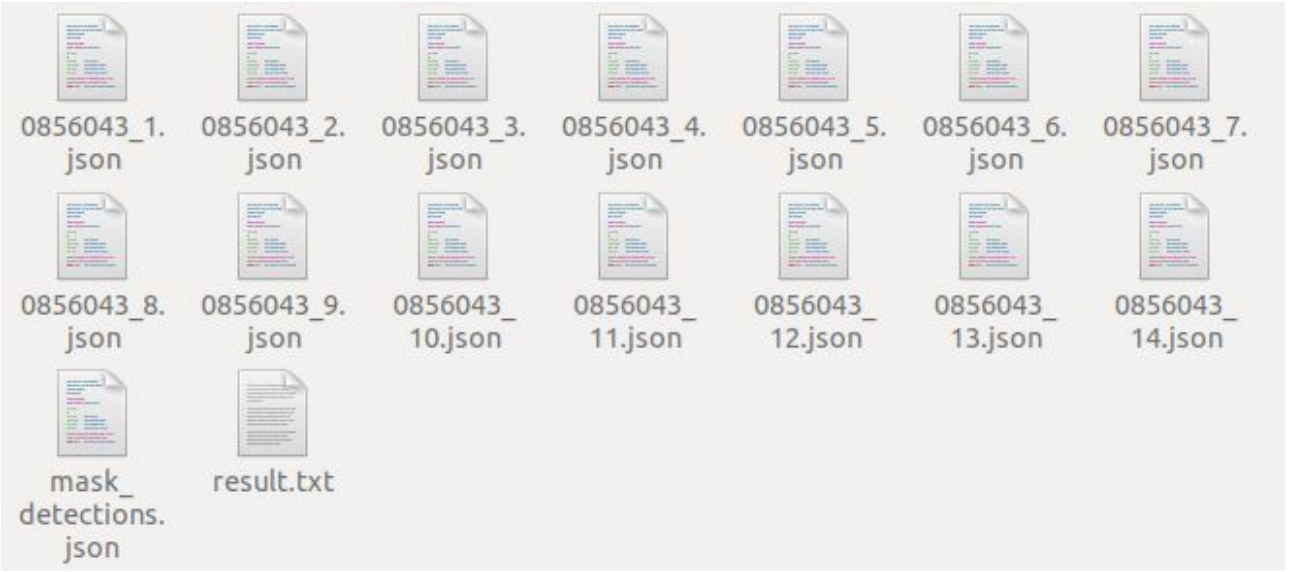


(4)Resume training, Run.
"python3 train.py --config=yolact_base_config
--resume=weights/yolact_base_297_50000.pth --start_iter=-1
"

(5)Predict one single image
Run “python3 eval.py
--trained_model=weights/res101/yolact_base_297_50000.pth
--score_threshold=0.15 --top_k=15
--image=data/test_images/2007_005460.jpg”



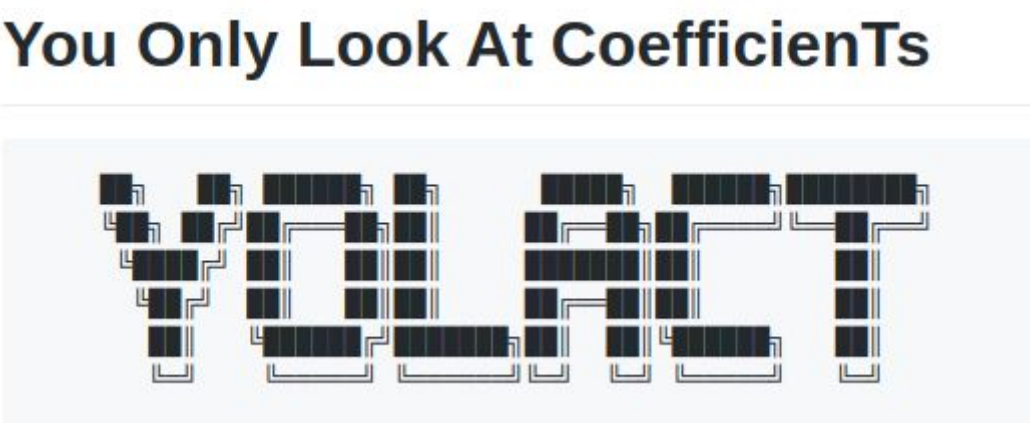
(6)Create JSON file.
Make sure testing images is in data/test_images.
Make sure the test.json is in data/
Run
"python3 createJSON.py
--trained_model=weights/res101/yolact_base_297_50000.pth
--score_threshold=0.15"
The predict file will create in results/mask_detections.json


0856043_1.json
0856043_2.json
0856043_3.json
0856043_4.json
0856043_5.json
0856043_6.json
0856043_7.json
0856043_8.json
0856043_9.json
0856043_10.json
0856043_11.json
0856043_12.json
0856043_13.json
0856043_14.json
mask_detections.json
result.txt

Methodology

1.Models

[paper, 2019/4/5, by UCD]

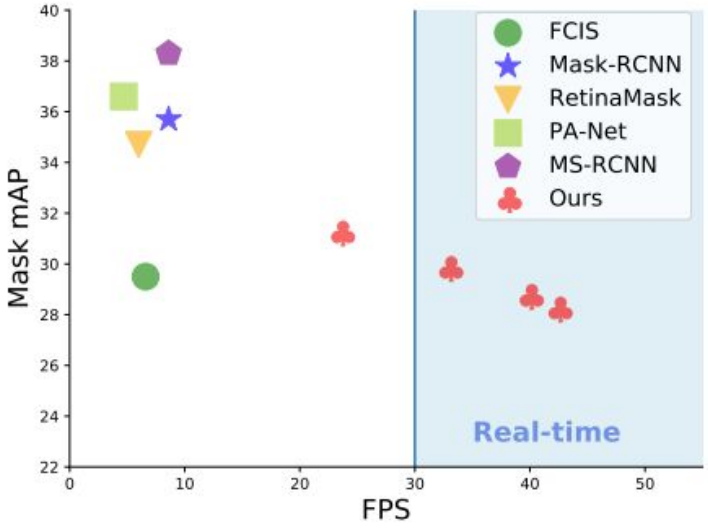


This model brings a realtime instance segmentation framework that forget localization step.

YOLACT breaks up instance segmentation into two parallel tasks:
(1) Generating a dictionary of non-local prototype masks over the entire image.
(2) Predicting a set of linear combination coefficients per instance.

The most advantages is that it is fast: because of its parallel structure and extremely lightweight assembly process.

It is the first real-time (> 30 fps) instance segmentation algorithm.



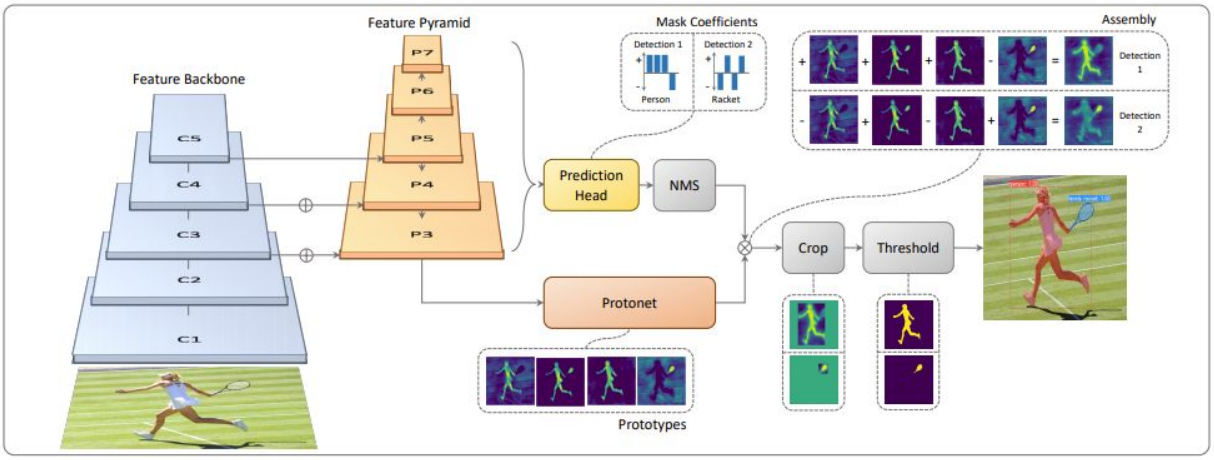


Figure 2: **YOLACT Architecture** Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and $k = 4$ in this example. We base this architecture off of RetinaNet [27] using ResNet-101 + FPN.

The YOLACT base this architecture off of RetinaNet [27] using ResNet-101 + FPN.

Method	Backbone	FPS	Time	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
PA-Net [29]	R-50-FPN	4.7	212.8	36.6	58.0	39.3	16.3	38.1	53.1
RetinaMask [14]	R-101-FPN	6.0	166.7	34.7	55.4	36.9	14.3	36.7	50.5
FCIS [24]	R-101-C5	6.6	151.5	29.5	51.5	30.2	8.0	31.0	49.7
Mask R-CNN [18]	R-101-FPN	8.6	116.3	35.7	58.0	37.8	15.5	38.1	52.4
MS R-CNN [20]	R-101-FPN	8.6	116.3	**38.3**	58.8	41.5	17.8	40.4	54.4
YOLACT-550	R-101-FPN	**33.5**	**29.8**	29.8	48.5	31.2	9.9	31.3	47.7
YOLACT-400	R-101-FPN	45.3	22.1	24.9	42.0	25.4	5.0	25.3	45.0
YOLACT-550	R-50-FPN	45.0	22.2	28.2	46.6	29.2	9.2	29.3	44.8
YOLACT-550	D-53-FPN	40.7	24.6	28.7	46.8	30.0	9.5	29.6	45.5
YOLACT-700	R-101-FPN	23.4	42.7	31.2	50.6	32.8	12.1	33.3	47.1

This model falls behind state-of-the-art instance segmentation methods in overall performance, but bring out a fast advantage.

3.Running Results

1	yolact_base_119_20000	0.51732
2	yolact_base_135_22687_interrupt	0.51079
3	yolact_base_59_10000.pth	0.49529
4	/res101/yolact_base_476_80000.pth	0.48846
5	/res101/yolact_base_416_70000.pth	0.48920
6	/res101/yolact_base_357_60000.pth	0.48792
7	/res101/yolact_base_297_50000.pth	0.54421(best)
8	/res101/yolact_base_238_40000.pth	0.50820
9	/res101/yolact_base_178_30000.pth	0.51500
10	/res50/yolact_resnet50_59_10000.pth	0.49617
11	/res50/yolact_resnet50_119_20000.pth	0.51141
12	/res50/yolact_resnet50_178_30000.pth	0.49389
13	/res50/yolact_resnet50_238_40000.pth	0.46195
14	/res50/yolact_resnet50_297_50000.pth	0.48077

2.Hyperparameters

Batch size=5 (And it almost use all of my 11GB GPU)
Training iterator=50000 (The best)
Base=Resnet101
optimizer=optim.SGD(lr=1e-3,momentum=0.9,weight_decay=5e-4)
lr_scheduler=multiply the lr with 0.1 in (280000, 360000, 400000).

Findings

1.Time test.

I try to add time test in my program.

```
pre_time = time.time()
preds = net(batch)
aft_time = time.time()

times += aft_time - pre_time
```

And divide it by 100(100 testing images).

```
0.033426289558410644
Done.
```

So it took about 0.033 sec or 3.342 ms per image, it performs well.

2.Faster and Better.

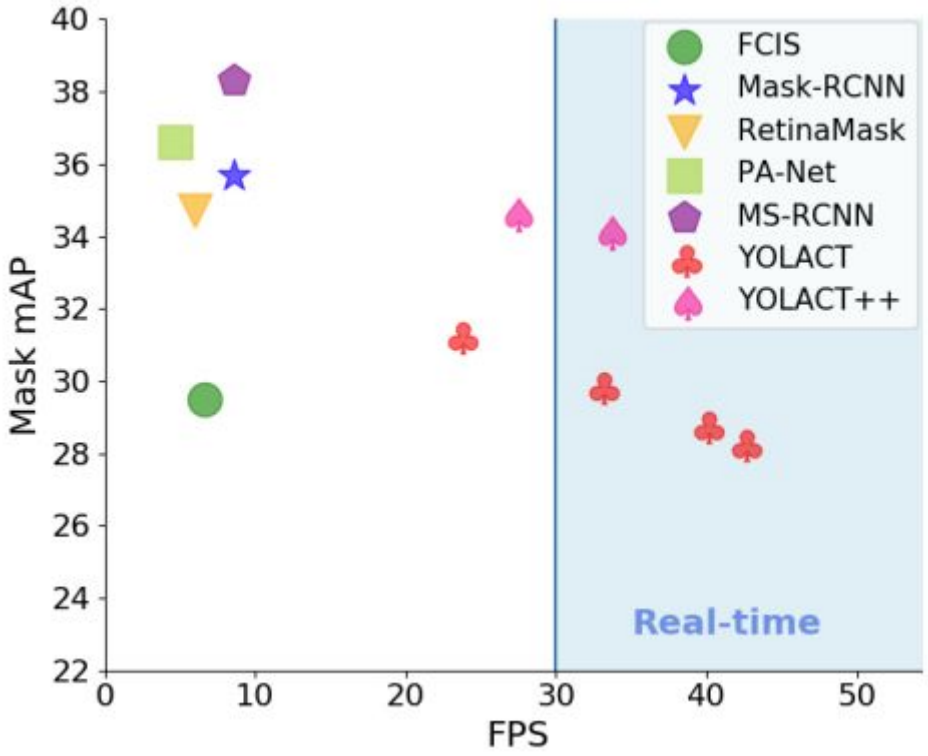
This model is obviously chasing more for speed than for accuracy, but with the combination of the latest method and model(base on RetinaNet using ResNet-101 + FPN), it actually reaches a good result than other classmates.

名稱	擁有者	上次修改時間	檔案大小
mAP_0.64310_0856052_4.json	溫孝婕	2019年12月13日 楊証琨	127 KB
mAP_0.64137_0856052_1.json	溫孝婕	2019年12月13日 楊証琨	127 KB
mAP_0.56066_0856054.json	莊紹平	下午4:50 莊紹平	245 KB
mAP_0.55767_0856049.json	吳毓軒	2019年12月13日 吳毓軒	202 KB
mAP_0.55679_0856605.json	呂哲宇	下午4:52 呂哲宇	155 KB
mAP_0.54421_0856043.json	我	下午9:15 我	176 KB

(In fifth place, now.)

3.New Version.

After I finished my work in 12/10, the author of YOLACT released a new version of YOLACT: YOLACT++!



paper link

It seems that the new model performs better than the old version, but I borrowed my GPU to my classmate, so maybe after this semester, I will try this one.

Reference

paper
original github
YOLACT introduction
YOLACT introduction(2)