Assignment 3

Wanyi Su
Student # 301445656

Use one late day

Part 1

1. List of the configs and modifications that you used.

Adjusted configuration:
MAX_ITER = 900, BATCH_SIZE_PER_IMAGE = 512, IMS_PER_BATCH = 4, BASE_LR = 0.00025,
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))

Change MAX_ITER to a larger value (MAX_ITER = 900).
Change IMS_PER_BATCH to a slightly larger value (IMS_PER_BATCH = 4).
Change model type to faster_rcnn_X_101_32x8d_FPN_3x
(cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))).

2. Factors which helped improve the performance. Explain each factor in 2-3 lines.

With a larger MAX_ITER value, more iterations the model would be trained, more times it can learn from the training process to improve the performance and accuracy.

For “faster_rcnn_X_101_32x8d_FPN_3x” model, it is the same category under Faster RCNN x101 as “faster_rcnn_R_101_FPN_3x”. We find it makes the result converging to a more optimal value compared to running “faster_rcnn_R_101_FPN_3x” model.

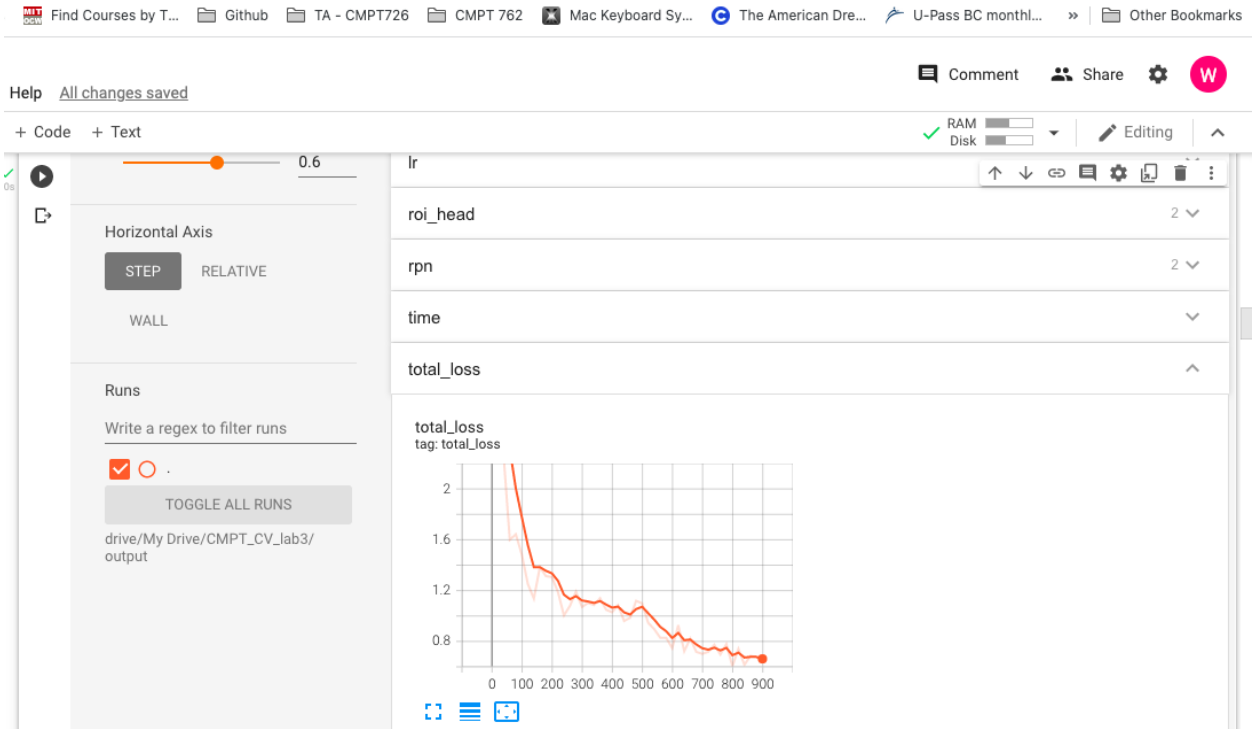
For batch size: Small values give a learning process that converges quickly at the cost of noise in the training process, while large values give a learning process that converges slowly with accurate estimates of the error gradient. After fine tuning with other parameters, we find that with a slightly larger batch size 4 than the current value of 2, we can have the training faster and more accurate.

3. Final plot for total training loss and accuracy.

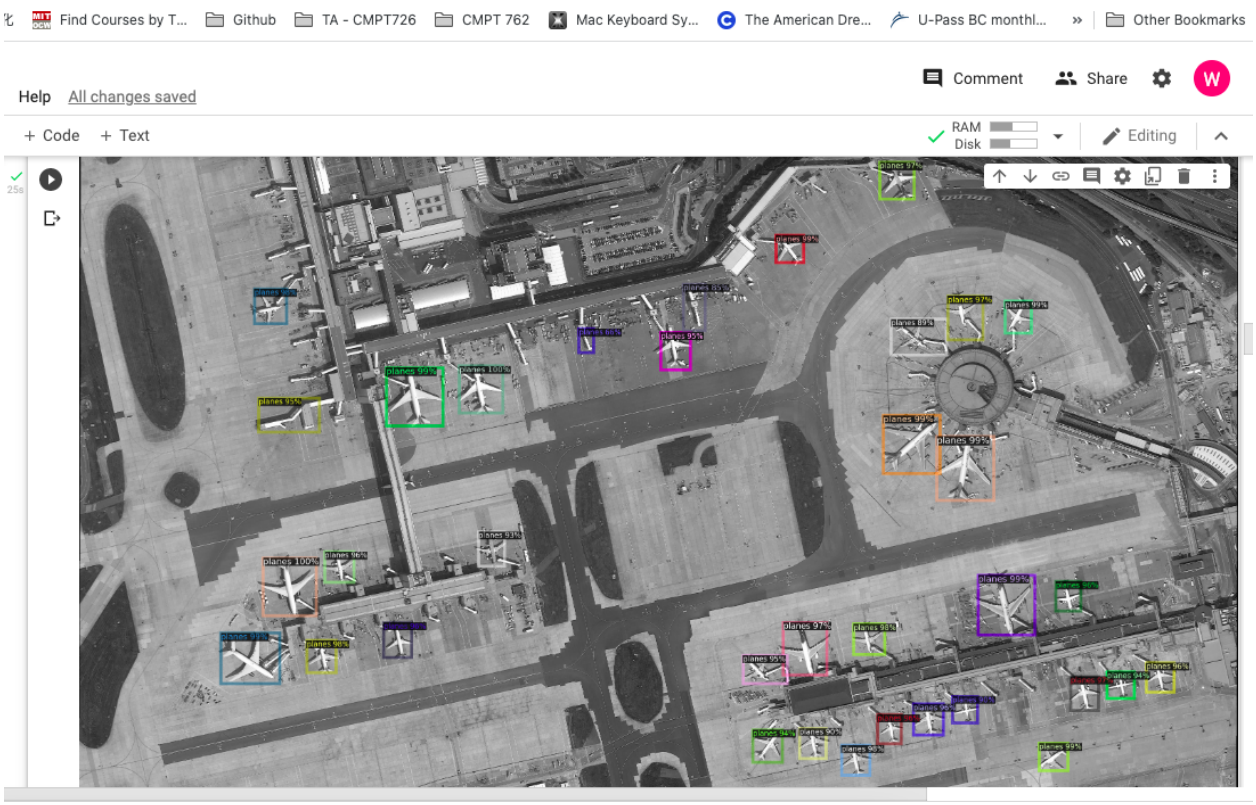
Average Precision (AP) = 30.961%,
AP at IoU=.50 at 54.634%.

```
index created!
[10/30 23:43:25 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[10/30 23:43:25 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.04 seconds.
[10/30 23:43:25 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[10/30 23:43:25 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.00 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.310
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.546
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.322
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.252
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.498
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.636
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.017
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.142
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.345
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.253
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.561
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.792
[10/30 23:43:25 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 30.961 | 54.634 | 32.216 | 25.171 | 49.825 | 63.561 |
OrderedDict([('bbox', {'AP': 30.96129078246458, 'AP50': 54.63409649626968, 'AP75': 32.216046047734075, 'APs': 25.17082267
```

Loss plot:



4. The visualization of 3 samples from the test set and the predicted results.


Find Courses by T...
Github
TA - CMPT726
CMPT 762
Mac Keyboard Sy...
The American Dre...
U-Pass BC monthl...
Other Bookmarks
Help
All changes saved
Comment
Share
W
+ Code
+ Text
RAM
Disk
Editing


planes 97%
planes 65%
planes 90%
planes 84%
planes 69%
planes 69%
planes 64%
planes 99%
planes 76%
planes 83%

5. Ablation study.

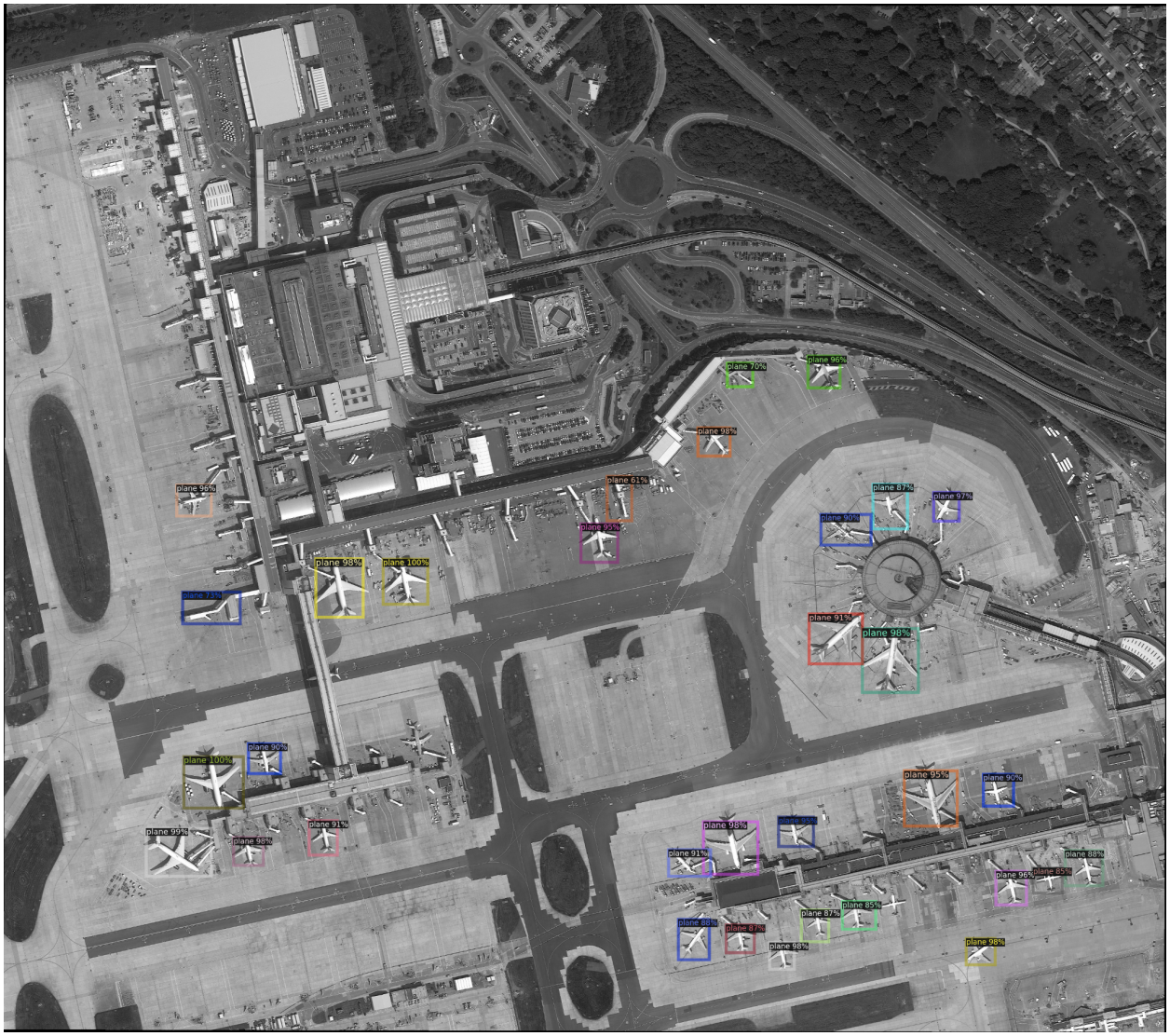
Comparing baseline model with hyperparameter-adjusted model, AP and AP 50 have improved by around 5% and 6% due to increase in max iteration number, slightly larger batch size, and different model selection. Details are as below.

- Baseline configuration:

MAX_ITER = 500, BATCH_SIZE_PER_IMAGE = 512, IMS_PER_BATCH = 2, BASE_LR = 0.00025
With baseline configuration, we have AP = 25.902% and AP50 = 48.830% as below.

```
[10/26 20:33:05 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[10/26 20:33:05 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.259
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.488
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.253
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.208
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.429
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.490
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.016
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.130
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.292
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.212
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.485
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.740
[10/26 20:33:05 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 25.902 | 48.830 | 25.297 | 20.838 | 42.909 | 48.975 |
OrderedDict([('bbox', {'AP': 25.901741705289556, 'AP50': 48.83023977346562, 'AP75': 25.296878032977578, 'APs': 20.837763473946953, 'APm': 42.909234631375725,
```

With baseline configuration, we can have a look at a test sample as below.



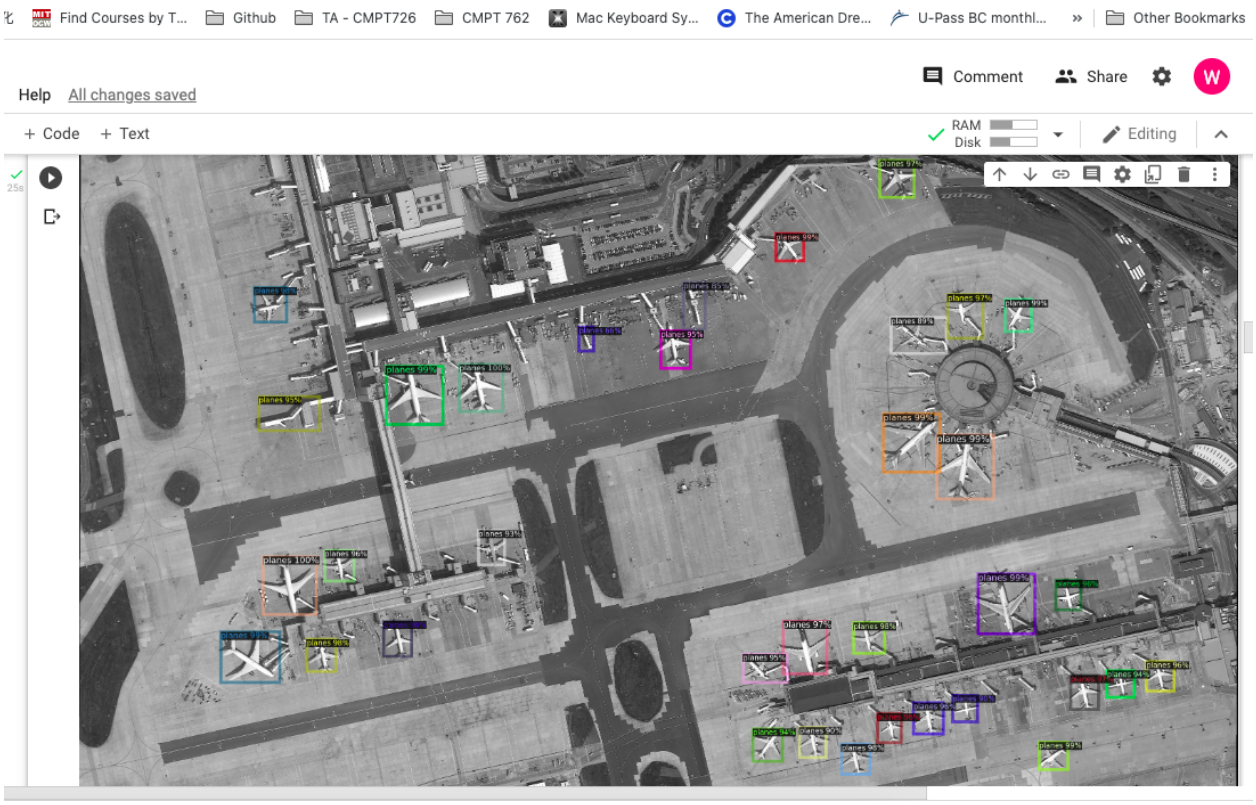
- Improved configuration:

With compared configuration, we increase iteration to 900, change model type to “faster_rcnn_X_101_32x8d_FPN_3x”, and adjust batch size to 4. Details of configuration as below:
MAX_ITER = 900, BATCH_SIZE_PER_IMAGE = 512, IMS_PER_BATCH = 4, BASE_LR = 0.00025,
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))

With improved configuration, we have improved AP = 30.961% and AP50 = 54.634% as below.

```
index created!
[10/30 23:43:25 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[10/30 23:43:25 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.04 seconds.
[10/30 23:43:25 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[10/30 23:43:25 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.00 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.310
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.546
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.322
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.252
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.498
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.636
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.017
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.142
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.345
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.253
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.561
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.792
[10/30 23:43:25 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 30.961 | 54.634 | 32.216 | 25.171 | 49.825 | 63.561 |
OrderedDict([('bbox', {'AP': 30.96129078246458, 'AP50': 54.63409649626968, 'AP75': 32.216046047734075, 'APs': 25.17082267
```

With improved configuration, we can have a look at a test sample as below. We can see the detection accuracy for planes are higher as the sample image shows.



Part 2

1. Report any hyperparameter settings you used (batch_size, learning_rate, num_epochs, optimizer).

num_epochs = 10
batch_size = 4
learning_rate = 0.01
weight_decay = 1e-5

2. Report the final architecture of your network including any modification that you have for the layers. Briefly explain the reason for each modification.

Overall, 7 up layers and 6 down layers. First augment channels and then decrease channels. We use summary() from library to display the model architecture as below:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 8, 128, 128]	224
BatchNorm2d-2	[-1, 8, 128, 128]	16
ReLU-3	[-1, 8, 128, 128]	0
conv-4	[-1, 8, 128, 128]	0
Conv2d-5	[-1, 16, 128, 128]	1,168
BatchNorm2d-6	[-1, 16, 128, 128]	32
ReLU-7	[-1, 16, 128, 128]	0
conv-8	[-1, 16, 128, 128]	0
MaxPool2d-9	[-1, 16, 64, 64]	0
down-10	[-1, 16, 64, 64]	0
Conv2d-11	[-1, 32, 64, 64]	4,640
BatchNorm2d-12	[-1, 32, 64, 64]	64
ReLU-13	[-1, 32, 64, 64]	0
conv-14	[-1, 32, 64, 64]	0
MaxPool2d-15	[-1, 32, 32, 32]	0
down-16	[-1, 32, 32, 32]	0
Conv2d-17	[-1, 64, 32, 32]	18,496
BatchNorm2d-18	[-1, 64, 32, 32]	128
ReLU-19	[-1, 64, 32, 32]	0
conv-20	[-1, 64, 32, 32]	0
MaxPool2d-21	[-1, 64, 16, 16]	0
down-22	[-1, 64, 16, 16]	0
Conv2d-23	[-1, 128, 16, 16]	73,856
BatchNorm2d-24	[-1, 128, 16, 16]	256
ReLU-25	[-1, 128, 16, 16]	0
conv-26	[-1, 128, 16, 16]	0
MaxPool2d-27	[-1, 128, 8, 8]	0
down-28	[-1, 128, 8, 8]	0
Conv2d-29	[-1, 256, 8, 8]	295,168
BatchNorm2d-30	[-1, 256, 8, 8]	512
ReLU-31	[-1, 256, 8, 8]	0
conv-32	[-1, 256, 8, 8]	0
MaxPool2d-33	[-1, 256, 4, 4]	0

down-34	[-1, 256, 4, 4]	0
Conv2d-35	[-1, 512, 4, 4]	1,180,160
BatchNorm2d-36	[-1, 512, 4, 4]	1,024
ReLU-37	[-1, 512, 4, 4]	0
conv-38	[-1, 512, 4, 4]	0
MaxPool2d-39	[-1, 512, 2, 2]	0
down-40	[-1, 512, 2, 2]	0
ConvTranspose2d-41	[-1, 512, 4, 4]	1,049,088
Conv2d-42	[-1, 256, 4, 4]	1,179,904
BatchNorm2d-43	[-1, 256, 4, 4]	512
ReLU-44	[-1, 256, 4, 4]	0
conv-45	[-1, 256, 4, 4]	0
up-46	[-1, 256, 4, 4]	0
BatchNorm2d-47	[-1, 256, 4, 4]	512
ConvTranspose2d-48	[-1, 256, 8, 8]	262,400
Conv2d-49	[-1, 128, 8, 8]	295,040
BatchNorm2d-50	[-1, 128, 8, 8]	256
ReLU-51	[-1, 128, 8, 8]	0
conv-52	[-1, 128, 8, 8]	0
up-53	[-1, 128, 8, 8]	0
BatchNorm2d-54	[-1, 128, 8, 8]	256
ConvTranspose2d-55	[-1, 128, 16, 16]	65,664
Conv2d-56	[-1, 64, 16, 16]	73,792
BatchNorm2d-57	[-1, 64, 16, 16]	128
ReLU-58	[-1, 64, 16, 16]	0
conv-59	[-1, 64, 16, 16]	0
up-60	[-1, 64, 16, 16]	0
BatchNorm2d-61	[-1, 64, 16, 16]	128
ConvTranspose2d-62	[-1, 64, 32, 32]	16,448
Conv2d-63	[-1, 32, 32, 32]	18,464
BatchNorm2d-64	[-1, 32, 32, 32]	64
ReLU-65	[-1, 32, 32, 32]	0
conv-66	[-1, 32, 32, 32]	0
up-67	[-1, 32, 32, 32]	0
BatchNorm2d-68	[-1, 32, 32, 32]	64
ConvTranspose2d-69	[-1, 32, 64, 64]	4,128
Conv2d-70	[-1, 16, 64, 64]	4,624
BatchNorm2d-71	[-1, 16, 64, 64]	32
ReLU-72	[-1, 16, 64, 64]	0
conv-73	[-1, 16, 64, 64]	0
up-74	[-1, 16, 64, 64]	0
BatchNorm2d-75	[-1, 16, 64, 64]	32
ConvTranspose2d-76	[-1, 16, 128, 128]	1,040
Conv2d-77	[-1, 8, 128, 128]	1,160
BatchNorm2d-78	[-1, 8, 128, 128]	16
ReLU-79	[-1, 8, 128, 128]	0
conv-80	[-1, 8, 128, 128]	0

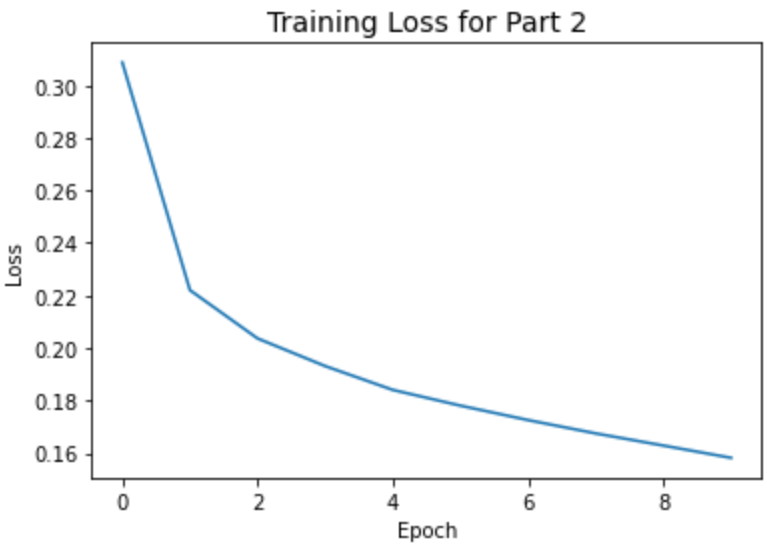
```
              up-81          [-1, 8, 128, 128]                   0
      BatchNorm2d-82          [-1, 8, 128, 128]                  16
            Conv2d-83          [-1, 3, 128, 128]                 219
      BatchNorm2d-84          [-1, 3, 128, 128]                   6
             ReLU-85          [-1, 3, 128, 128]                   0
             conv-86          [-1, 3, 128, 128]                   0
            Conv2d-87          [-1, 1, 128, 128]                  28
             conv-88          [-1, 1, 128, 128]                   0
================================================================
Total params: 4,549,765
Trainable params: 4,549,765
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 39.22
Params size (MB): 17.36
Estimated Total Size (MB): 56.76
----------------------------------------------------------------
```

Why choose the architecture:
Firstly, in order to get a good segmentation result and regarding previous experience, the channels of images need to be augmented from 3 to a large number like 512, with channel number double each layer. Also some pooling, normalization, and ReLU layers need to be added after conv blocks.
Secondly, I searched for famous segmentation networks online and found architectures like U-Net, which is developed for segmenting biomedical images. The purpose of U-Net is similar to the plane segmentation in graphs as in the assignment. So I use a similar layer structure to have a try and the result turns out to be good.
I also tried augmented to 1024 channels but the result turns out to be not as optimal as before, also the gpu cuda memory is hard to afford when training the model.

3. Report the loss functions that you used and then plot the total training loss of the training procedure.

Loss function used is Binary Cross Entropy Loss with logits, which is nn.BCEWithLogitsLoss().

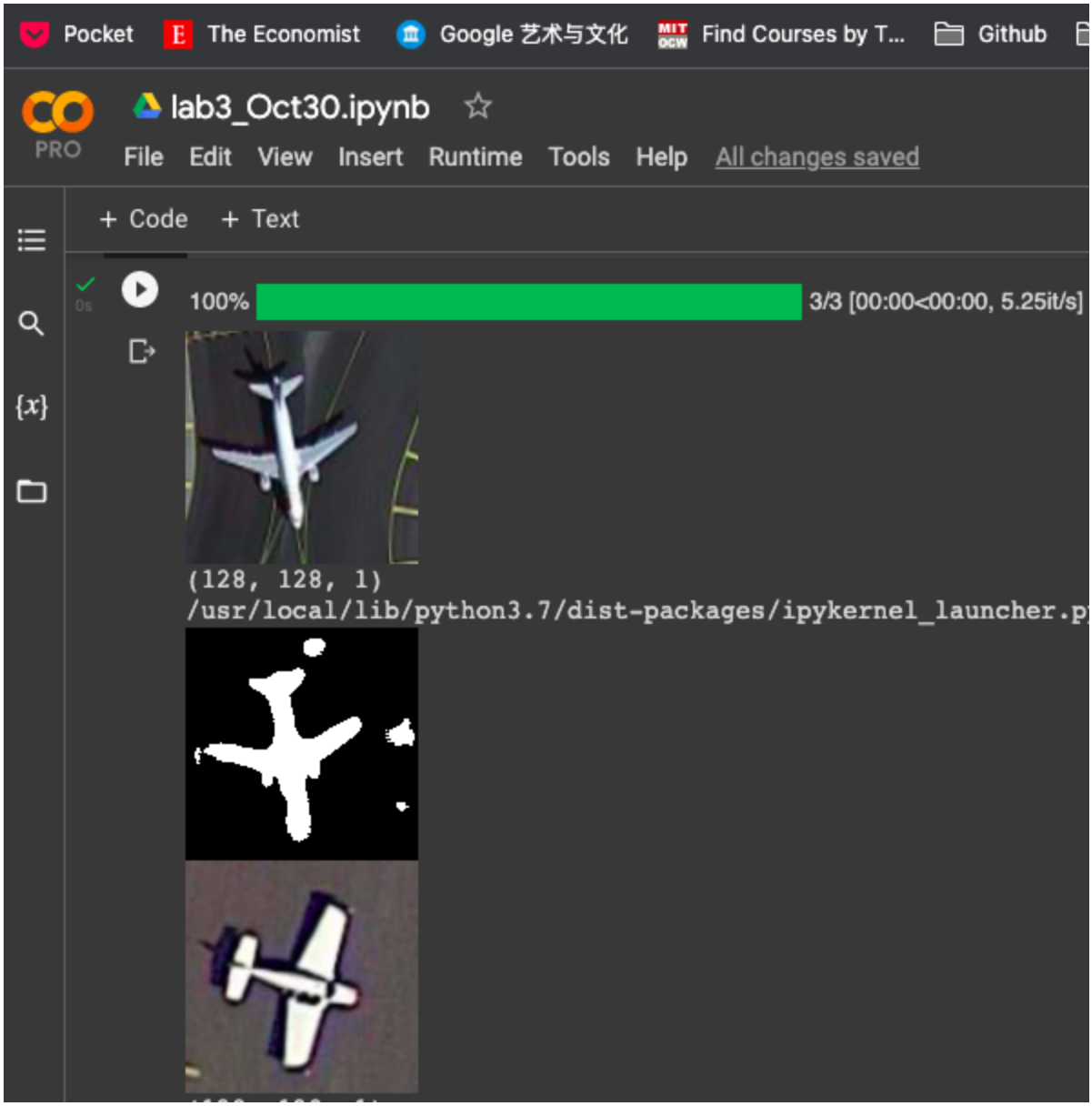
This training loss plot keeps track of average total loss for each epoch.

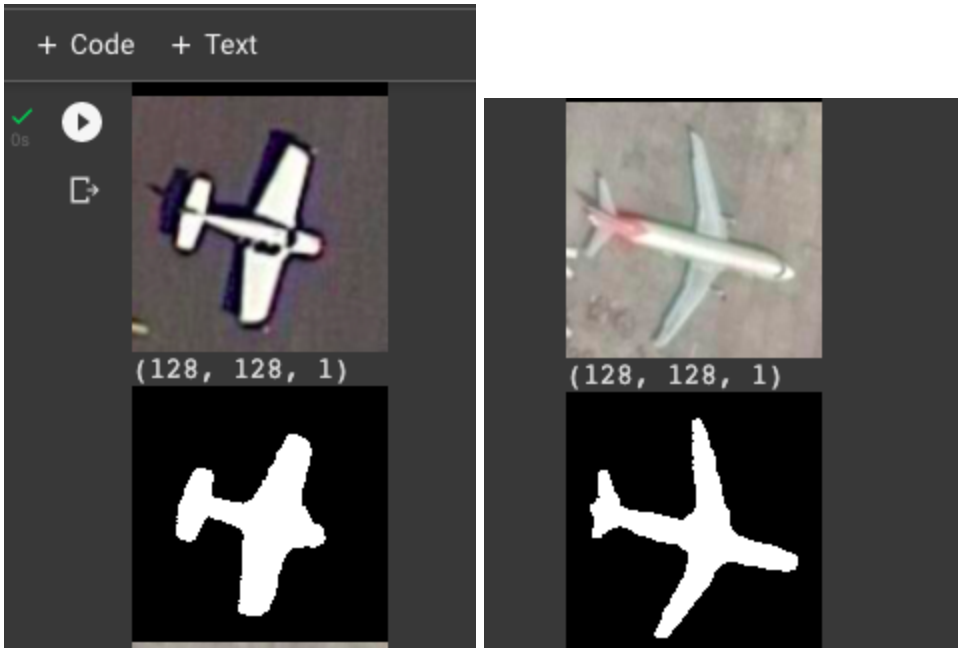


4. Report the final mean IoU of your model.
   Mean IoU = 0.776

```
+ Code  + Text

    for i in range(img.shape[0]):
        t_mask = np.array(mask[i])[0] # np array
        p_mask = np.array(pred[i])[0]
        # print(type(t_mask))
        total_iou += iou(t_mask, p_mask)
        cnt += 1


print("\n #images: {}, Mean IoU: {}".format(cnt, total_iou/cnt))

100% ████████████████████ 6382/6382 [00:46<00:00, 135.71it/s]

 #images: 6382, Mean IoU: 0.7761640252911344
```

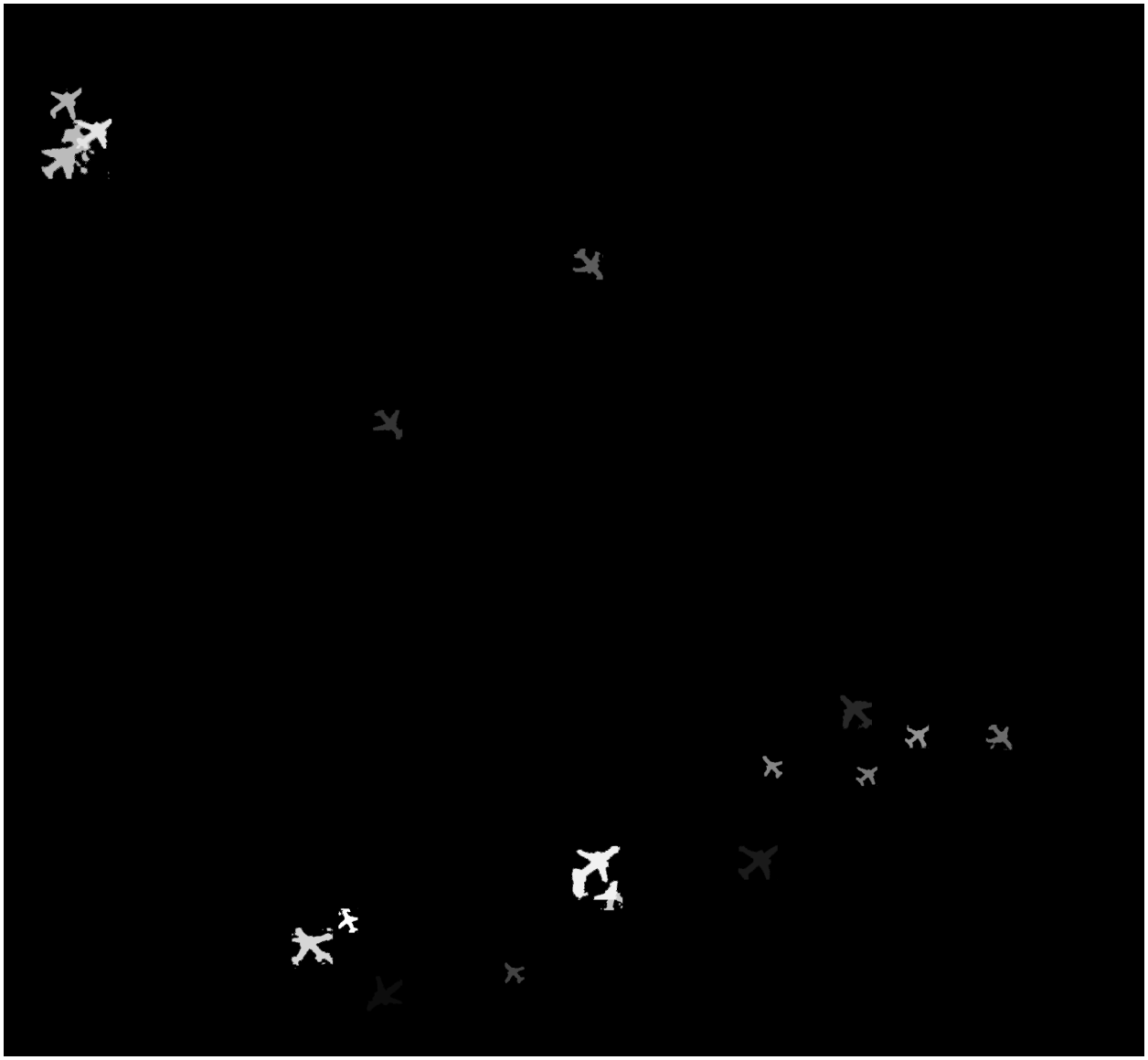
5. Visualize 3 images from the test set and the corresponding predicted masks.

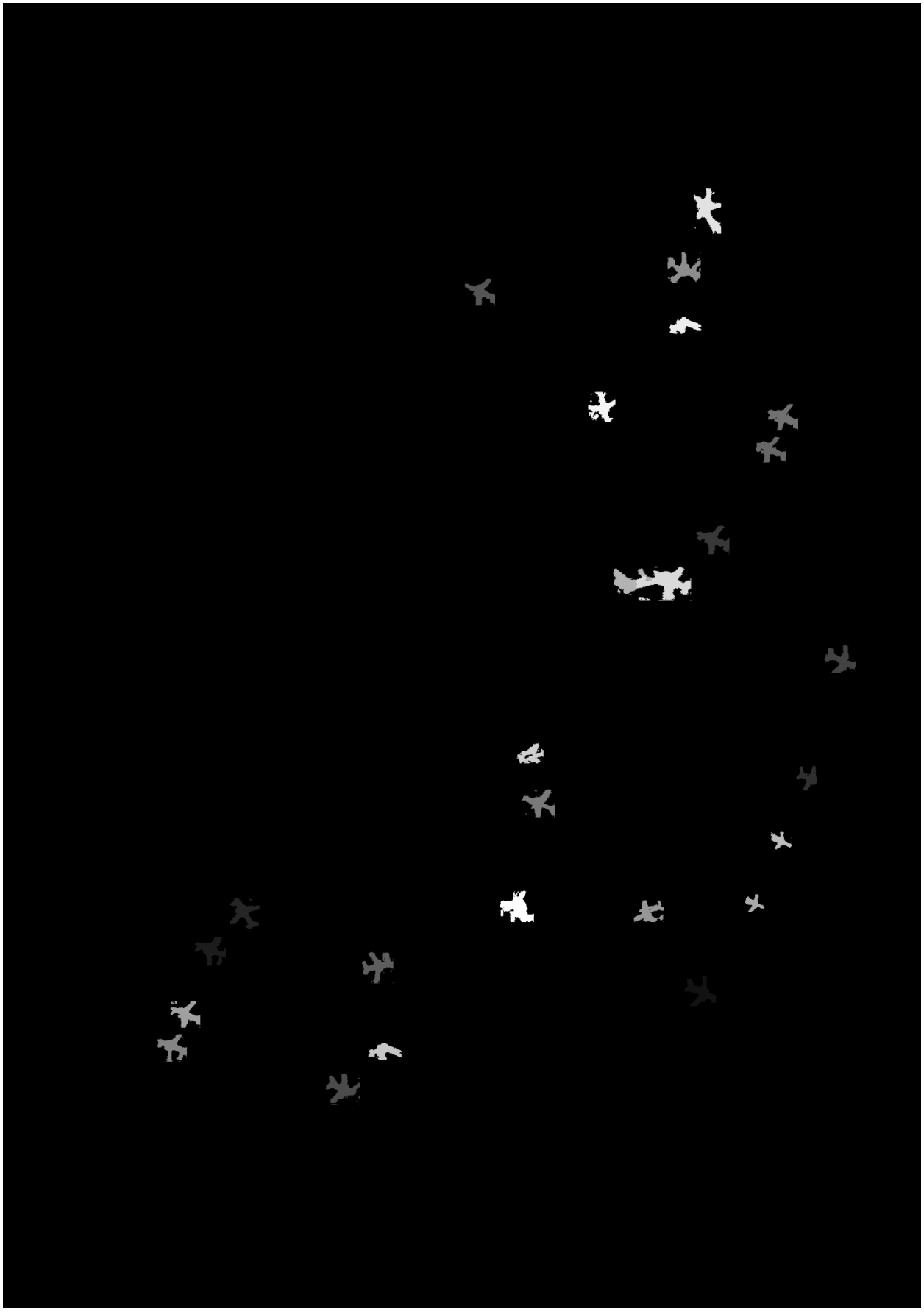

Pocket
The Economist
Google 艺术与文化
Find Courses by T...
Github
lab3_Oct30.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
100% 3/3 [00:00<00:00, 5.25it/s]
(128, 128, 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.p

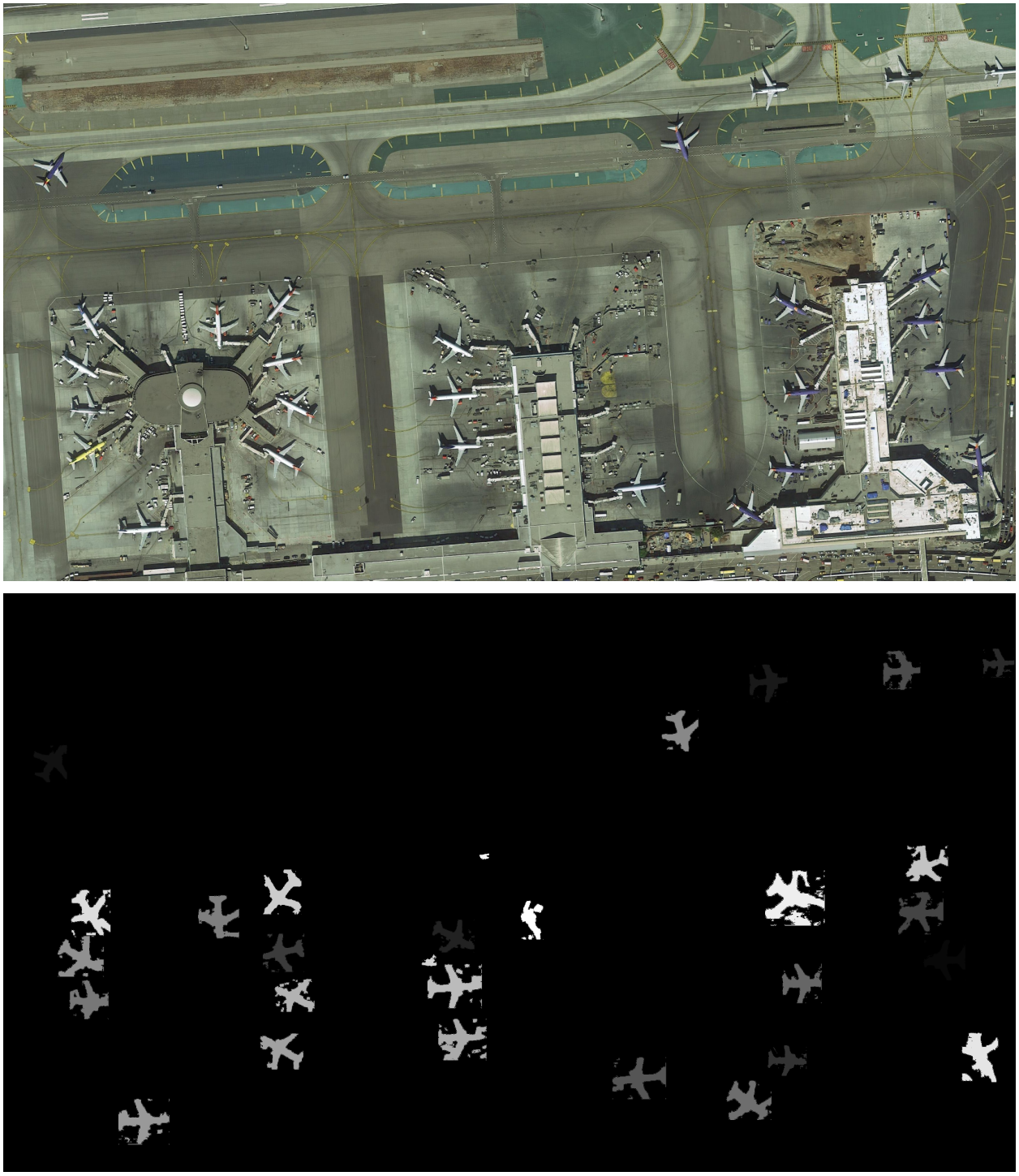


Part 3

1. Name on Kaggle board: Wanyi Su
2. Best score on Kaggle: 0.47556
3. Test sample visualization:







4. Csv file has been submitted.

Part 4

1. The visualization and the evaluation results similar to Part 1.

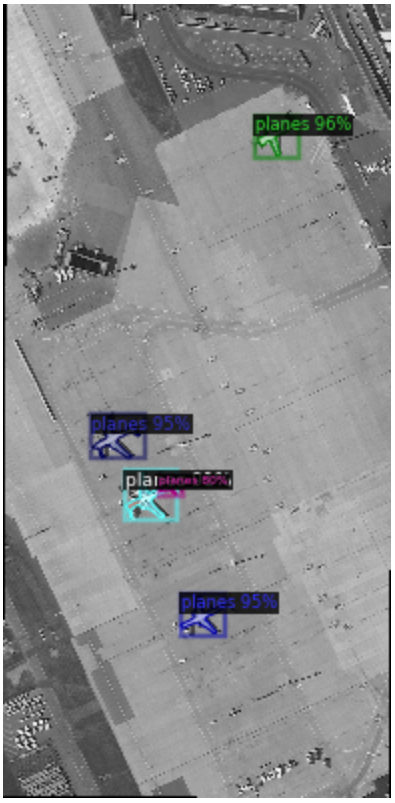
The result of part 4 seems better than that of part 1, regarding higher AP as well as AP50 values, more detected planes in sample images, and loss plot as below.

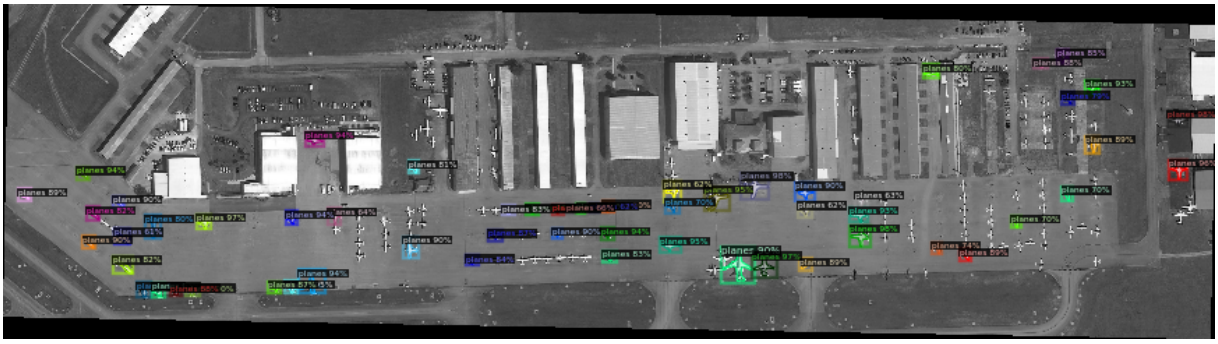
- Bbox AP:

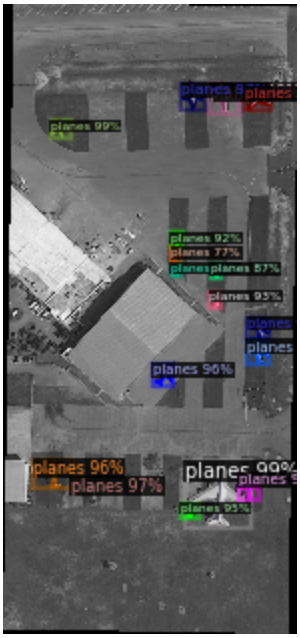
```
creating index...
index created!
[10/31 06:34:22 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[10/31 06:34:22 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.05 seconds.
[10/31 06:34:22 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[10/31 06:34:22 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.332
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.645
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.305
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.289
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.498
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.524
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.016
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.139
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.382
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.291
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.596
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.767
[10/31 06:34:22 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 33.223 | 64.467 | 30.473 | 28.898 | 49.838 | 52.395 |
Loading and preparing results...
```

- Segmentation AP:

```
creating index...
index created!
[10/31 06:34:23 d2.evaluation.fast_eval_api]: Evaluate annotation type *segm*
[10/31 06:34:23 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.14 seconds.
[10/31 06:34:23 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[10/31 06:34:23 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.101
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.399
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.005
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.073
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.165
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.438
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.007
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.064
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.136
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.093
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.232
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.475
[10/31 06:34:23 d2.evaluation.coco_evaluation]: Evaluation results for segm:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 10.073 | 39.921 | 0.513  | 7.287  | 16.496 | 43.768 |
OrderedDict([('bbox', {'AP': 33.22340486671494, 'AP50': 64.46673068396537, 'AP75': 30.473046696051103, 'APs': 28.898274788436527, 'APm': 49.83830537023678, '
```


planes 96%
planes 95%
planes 95%





2. Explain the differences between the results of Part 3 and Part 4 in a few lines.

Part 3 first uses model in part 1 to detect bounding boxes for planes, then extract bounding boxes and convert masks to segment out planes. Part 4 uses Mask R-CNN model “mask_rcnn_R_50_FPN_3x” to do bounding box assignment and segmentation for objects directly in fewer steps.

For detecting bounding boxes, AP (33.223%) and AP50 (64.467%) results of part 4 are higher than part 1 by 3% and 10% respectively. This can be validated by more planes being detected with high scores in sample images. For segmentation objects, part 4 also performs well.
I think MASK R-CNN is better since it's more efficient with libraries and API functions and its detection accuracy is higher.

Pros of Mask R-CNN:

- Mask R-CNN model is simple to implement and train. Users can simply put images into different Mask R-CNN models using libraries and API functions, then perform training to get results.
- Mask R-CNN model has good performance. Mask R-CNN outperforms single-model entries on instance and semantic segmentation tasks.
- Mask R-CNN model is efficient in time and adds only a small computational overhead to Faster R-CNN.
- Mask R-CNN is easy to generalize to other tasks, like some estimation task. For example, human pose estimation in the same framework.

Cons of Mask R-CNN:

- Mask R-CNN implements object detection and segmentation at the same time, which is not as flexible as normal separate method, particularly if we want to adjust bounding boxes or segmentation for specific purposes separately.
- Implemented as blackbox. Difficult for users to figure out details and intuition.
- Mask R-CNN would add a small computational overhead to Faster R-CNN.

Pros of Object Detection:

- Provides more flexibility about hyperparameter configuration, so users are able to tune configuration and get results towards their expectation.
- Many models and API functions to choose from.
- Provides more steps and details towards results, so users can check each step rather than facing a blackbox.

Cons of Object Detection:

- If segmentation is needed, more complex steps are required. Not friendly to users with few computer vision backgrounds.
- Time inefficient for training of large iterations and required lots of hyperparameter tuning to reach an optimal result.
- Some challenges are faced by object detection, like the influence of lighting conditions, different forms of objects, background mess.