Yide Ma, 301476100

Hi dear TA, May I use four free delay days(all my left) for this lab. Thanks!

Part1
Pre_trained model="COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"
MAX_ITER=500
IMS_PER_MATCH=3
BATCH_SIZE_PER_IMAGE=512
NUM_WORKERS=4

Some modification I made:
Pre_trained model="COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"

MAX_ITER=5000

CROP_PATCH_WIDTH=800
CROP_PATCH_HEIGHT=800
CROP_B_BOX_IOU_THRESHOLD=0.6

#for inference
ROI_HEADS.SCORE_THRESH_TEST=0.5

LR_SCHEDULER_NAME=”WarmupCosineLR”
LR=0.00025
MOMENTUM=0.9

PIXEL_STD=[57.375,57,120,58.395]

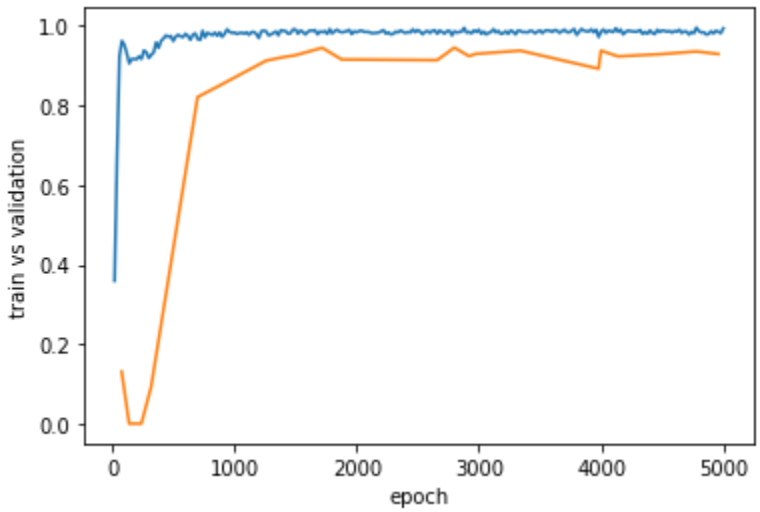
Some preprocessing of image:

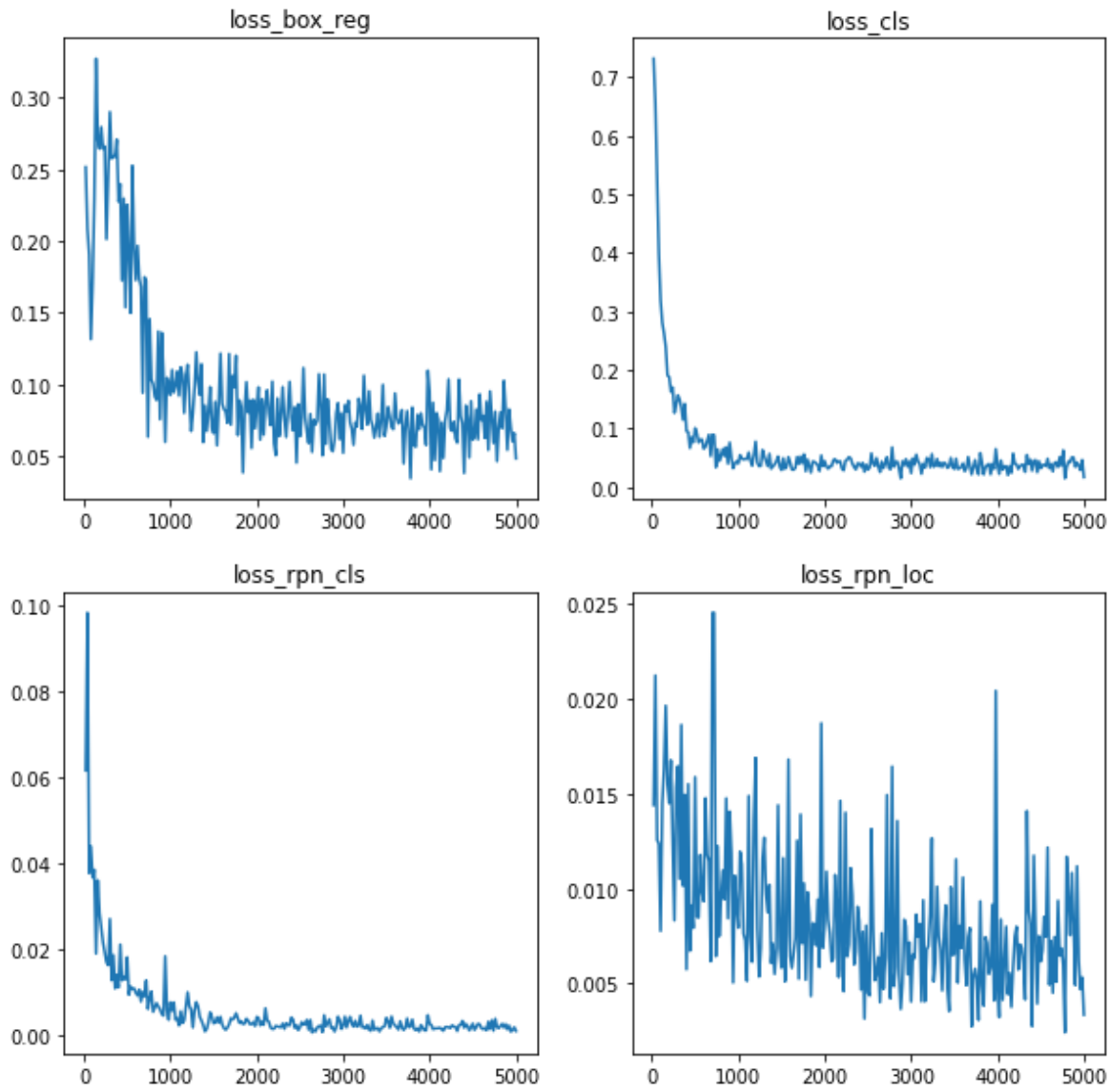
Resize
Custom crop
random Brightness
RandomFlip horizontal
RandomFlip vertical
Rotation
Normalize:
TRANSFORM_MEAN= [0.485, 0.456, 0.406]
TRANSFORM_STD = [0.229, 0.224, 0.225]

Factors:
1.The pre_trained model used for training, deeper, larger network may better
2.The preprocessing techniques for images, data augmentation

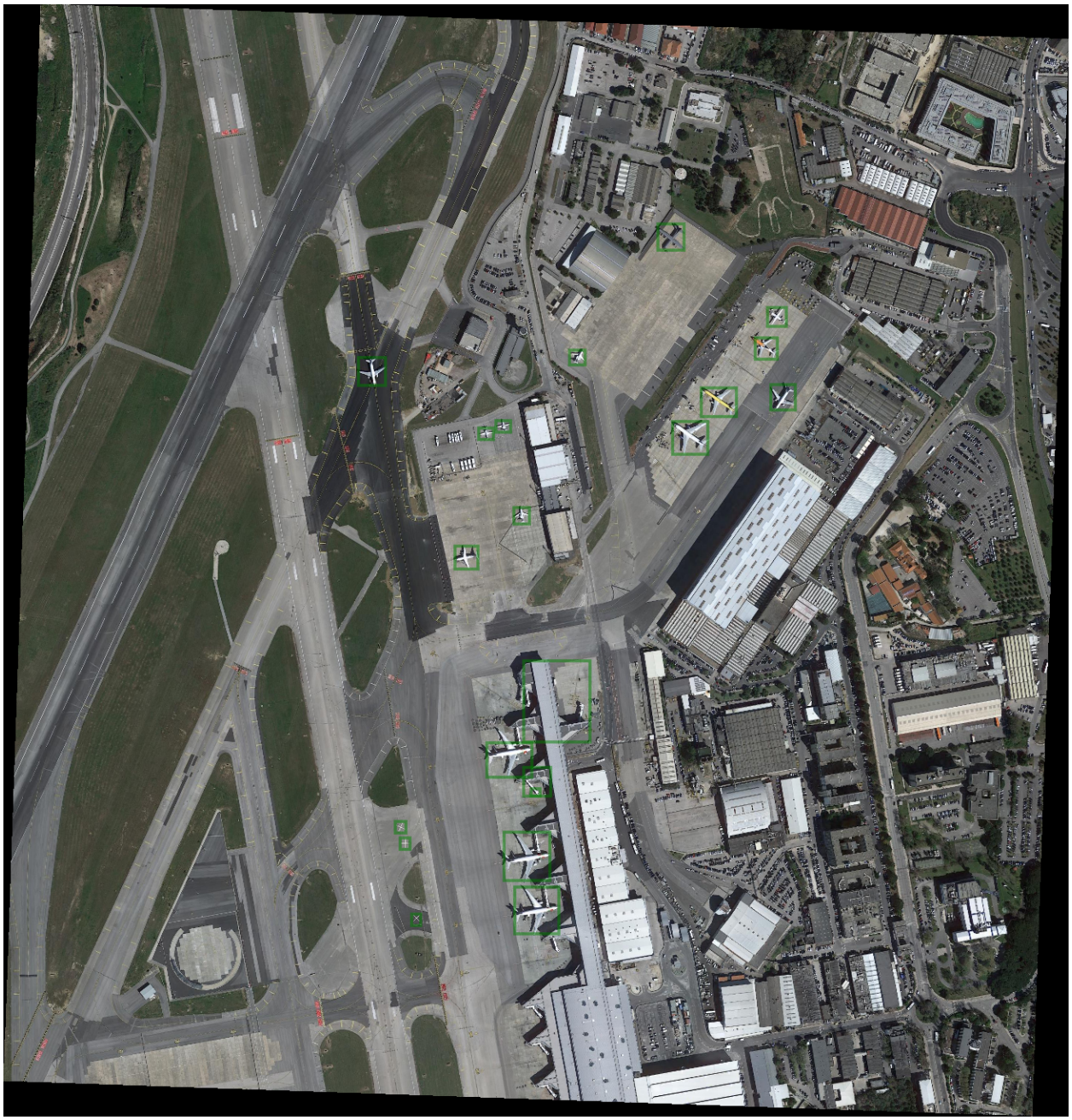
3.How to choose best checkpoint as a pre-trained model when inference the model
4.Divide image into patches then send to the detection model

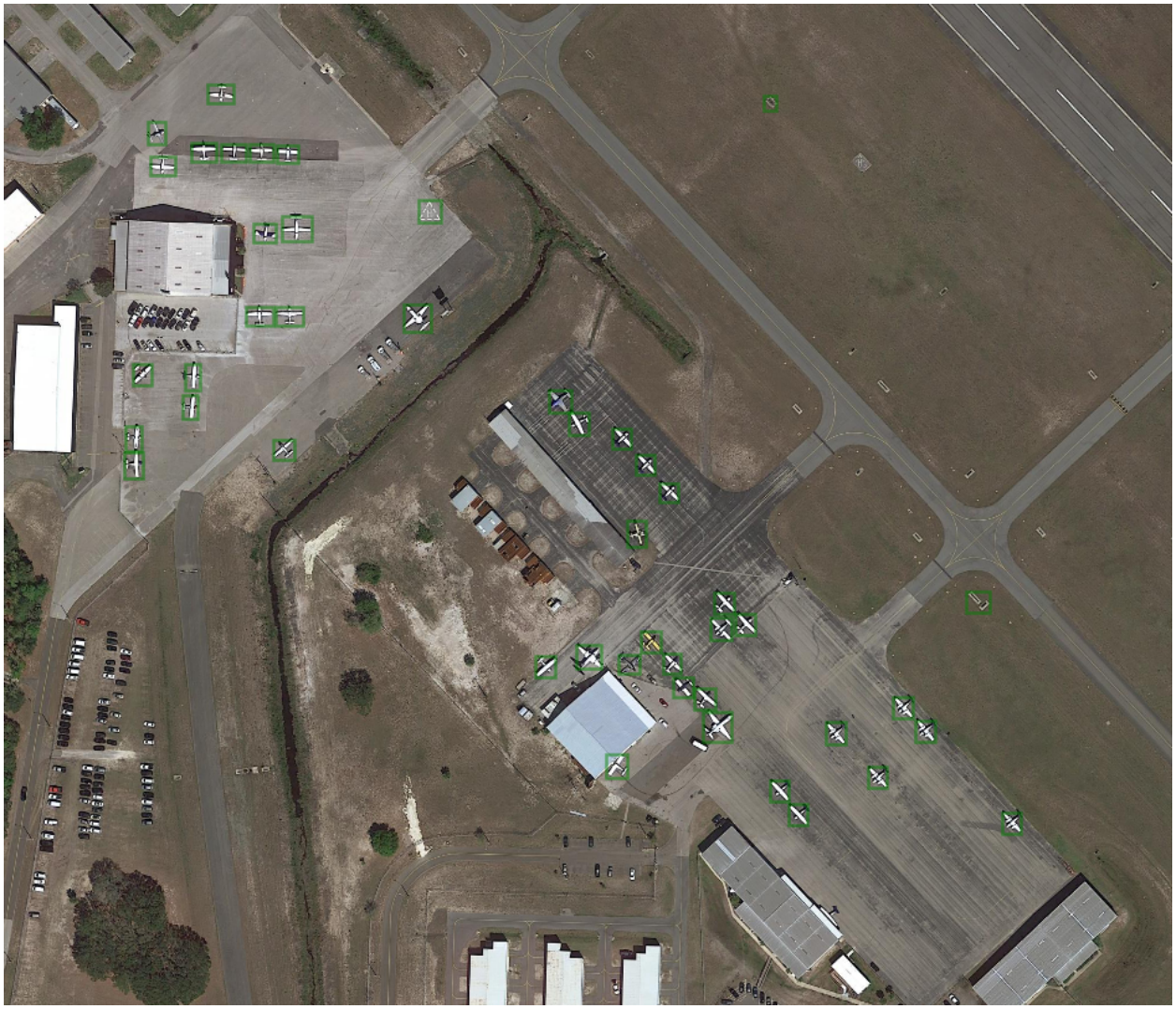
Chart:

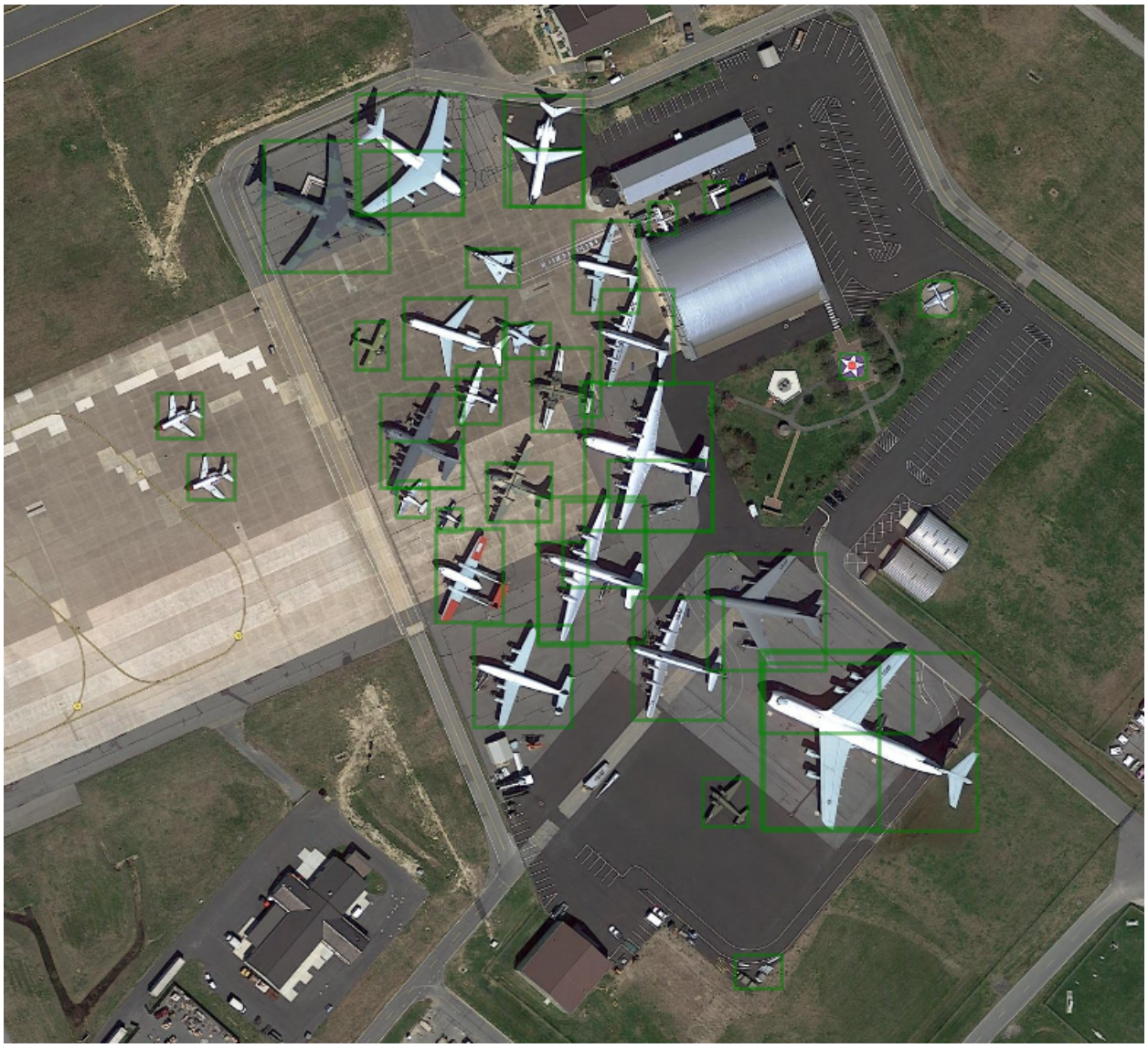




3 visuailization examples:







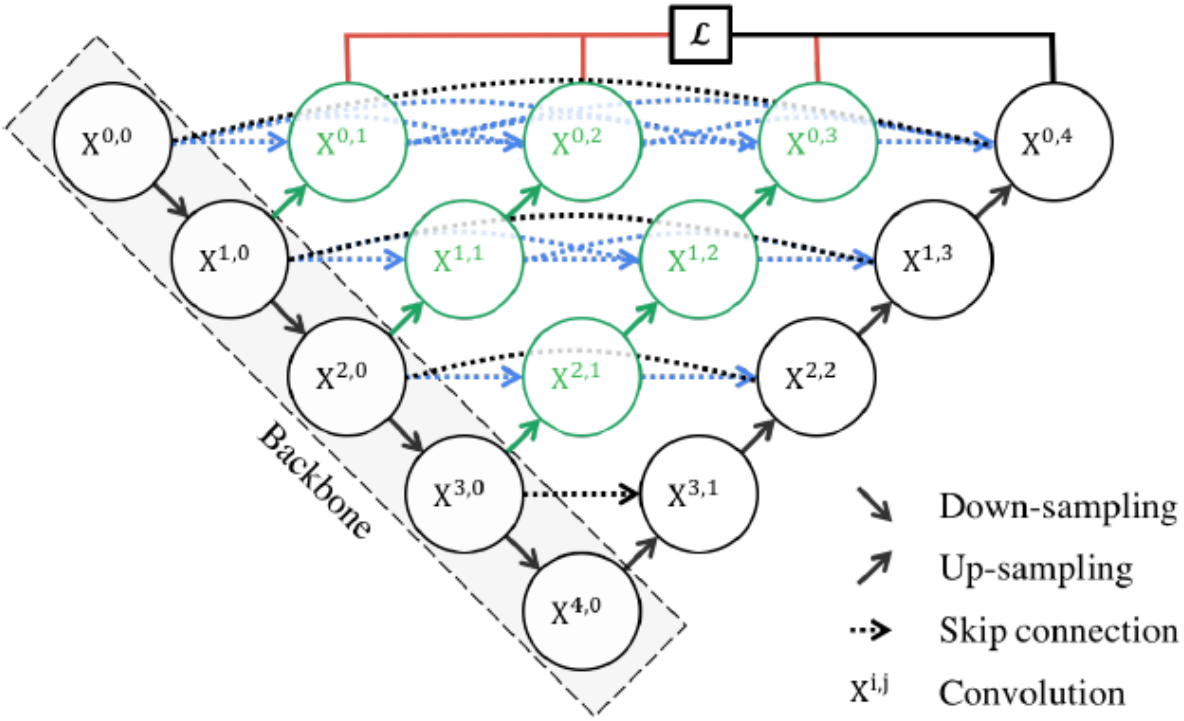
Ablation study: The choice of pre-trained model, deeper model may use more powerful computing resources, but its performance is better. For instance, I have used faster_rcnn_R_101_FPN_3x.yml, the total loss is 0.794 at 500 iterations. If use faster_rcnn_X_101_32x8d_FPN_3x.yml, after 5000 iteration, the total loss is 0.077

Another improvement is using patches for the model, from the size of patches, there could be around 50% improvement of AP value.

Part2
For the configures I used for this part,
LEARNING_RATE= 0.006
BATCH_SIZE = 4

```
NUM_CLASSES= 1    # Binary classification: just use 1 out channel.
N_EPOCHS = 120
VAL_PERCENTAGE= 0.15
INPUT_WIDTH= 256
INPUT_HEIGHT= 256
CHECKPOINT_SAVE_DIR= "seg_output"
CHECKPOINT_SAVE_EPOCH_INTERVAL = 30
TRAIN_LOG_FILENAME = os.path.join(CHECKPOINT_SAVE_DIR, "loss_log.csv")
optimizer = optim.SGD(network.parameters(), lr=LEARNING_RATE, momentum=0.9,
weight_decay=5e-4)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=N_EPOCHS)
```

I have implemented the UNet++ for segmentation model, here is its architecture:



For each convolution X^i,j , I used VGGBlock, with corresponding to this chart

conv0_0	VGGBlock(3,32,32)
conv1_0	VGGBlock(32,64,64)

conv2_0	VGGBlock(64,128,128)
conv3_0	VGGBlock(128,256,256)
conv4_0	VGGBlock(256,512,512)
conv0_1	VGGBlock(96,32,32)
conv1_1	VGGBlock(192,64,64)
conv2_1	VGGBlock(384,128,128)
conv3_1	VGGBlock(768,256,256)
conv0_2	VGGBlock(160,32,32)
conv1_2	VGGBlock(320,64,64)
conv2_2	VGGBlock(640,128,128)
conv0_3	VGGBlock(224,32,32)
conv1_3	VGGBlock(448,64,64)
conv0_4	VGGBlock(288,32,32)

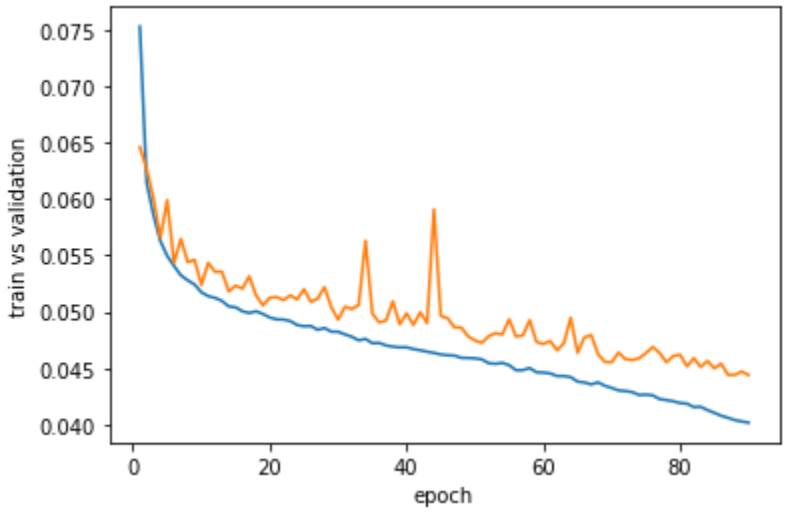
For the conv size and image size between layers

x0_0	32conv, 256x256
x1_0	64conv, 128x128
x0_1	32conv, 256x256
x2_0	128conv, 64x64
x1_1	64conv, 128x128
x0_2	32conv, 256x256
x3_0	256conv, 32x32
x2_1	128conv, 64x64
x1_2	64conv, 128x128
x0_3	32conv, 256x256
x4_0	512conv, 16x16
x3_1	256conv, 32x32

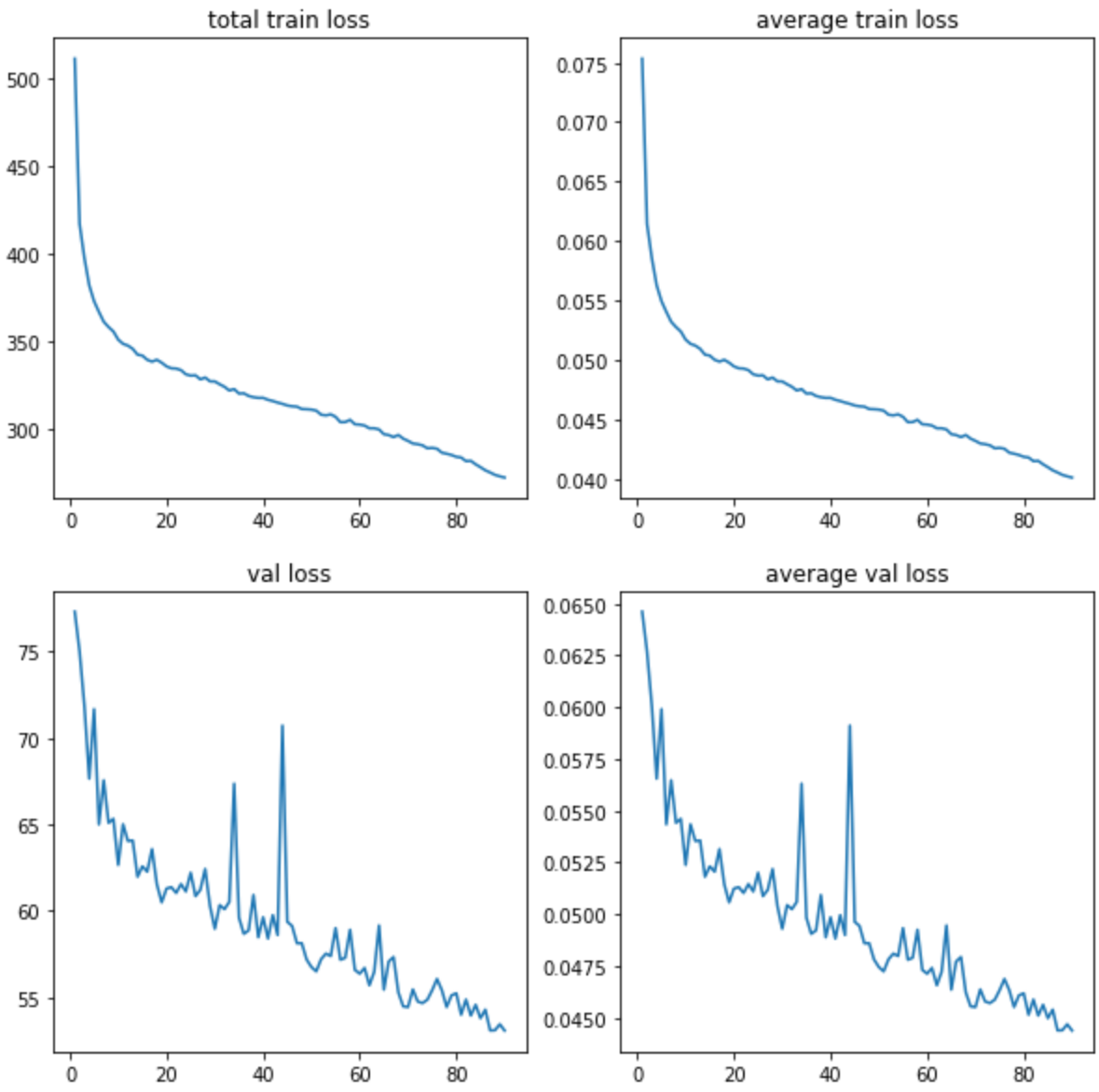
x2_2	128conv, 64x64
x1_3	64conv, 128x128
x0_4	32conv. 256x256

The reason for using this model is because this model performs very well on Paper UNet++. It learns better features when placing different combinations of VGG blocks, like what densenet did.

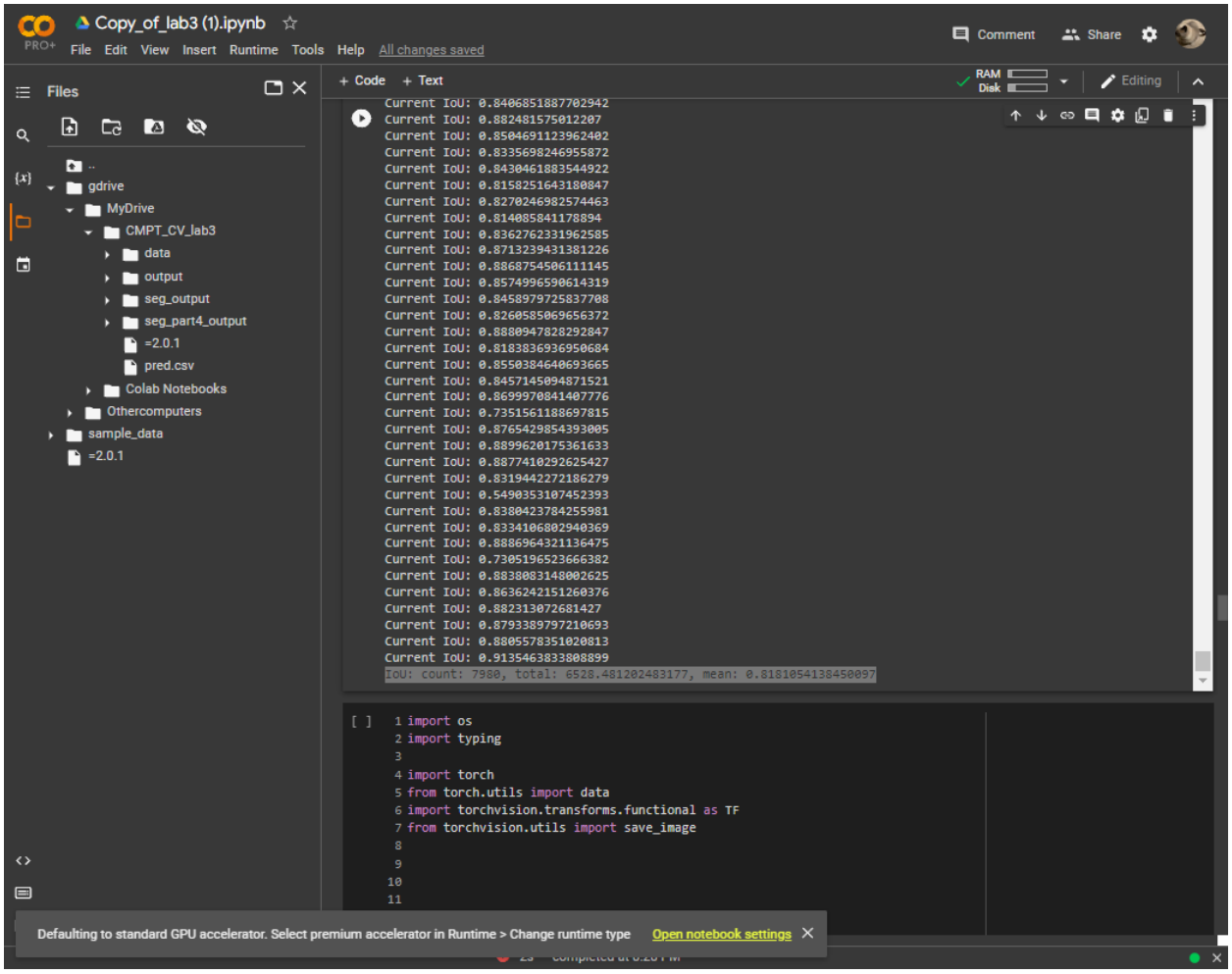
The loss I have used is dice_loss and binary cross entropy loss.
Accuracy chart:



Loss chart:

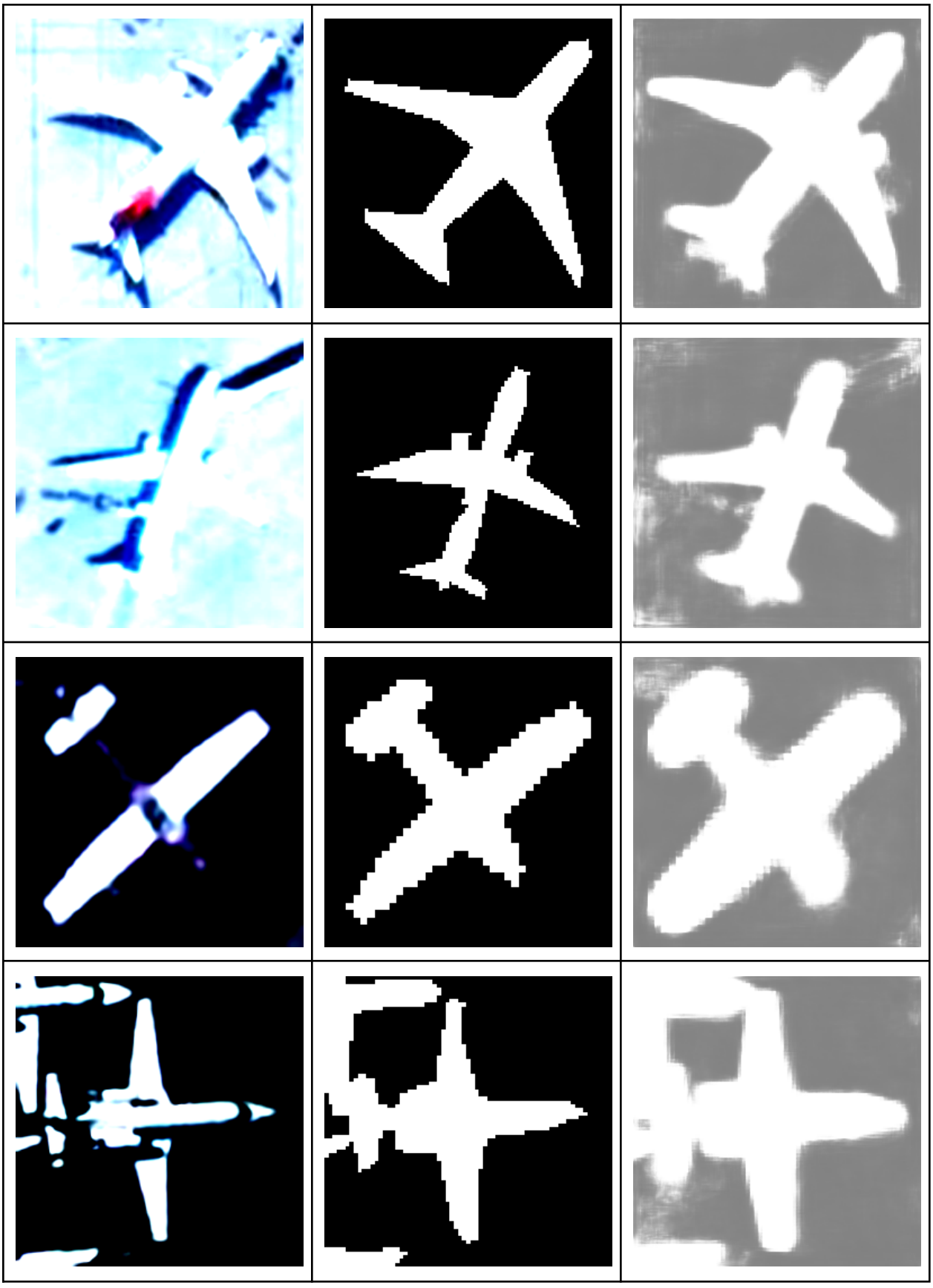


Final IOU:
IoU: count: 7980, total: 6528.481202483177, mean: 0.8181054138450097



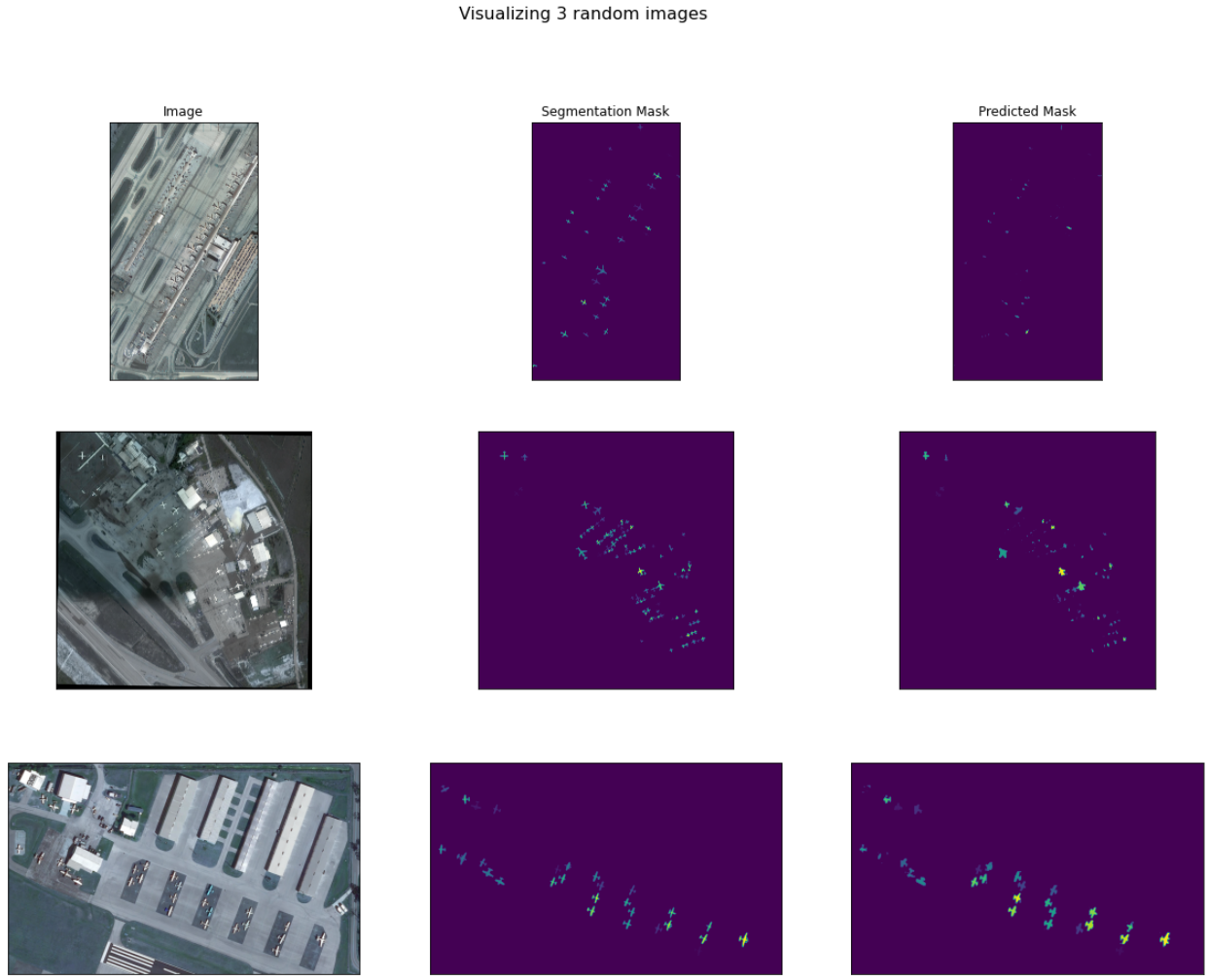
Visualization:

image	mask	prediction
		

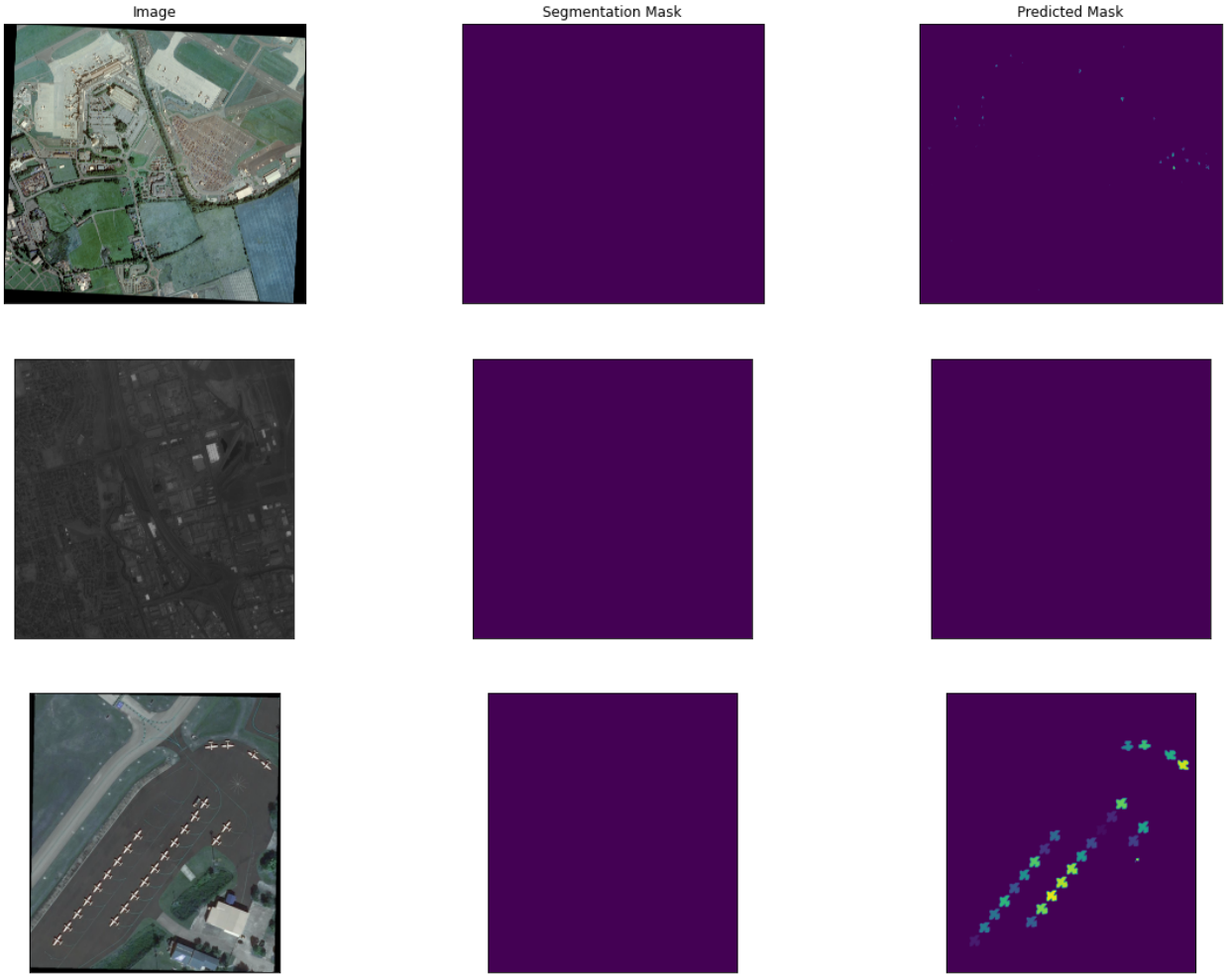


Part3

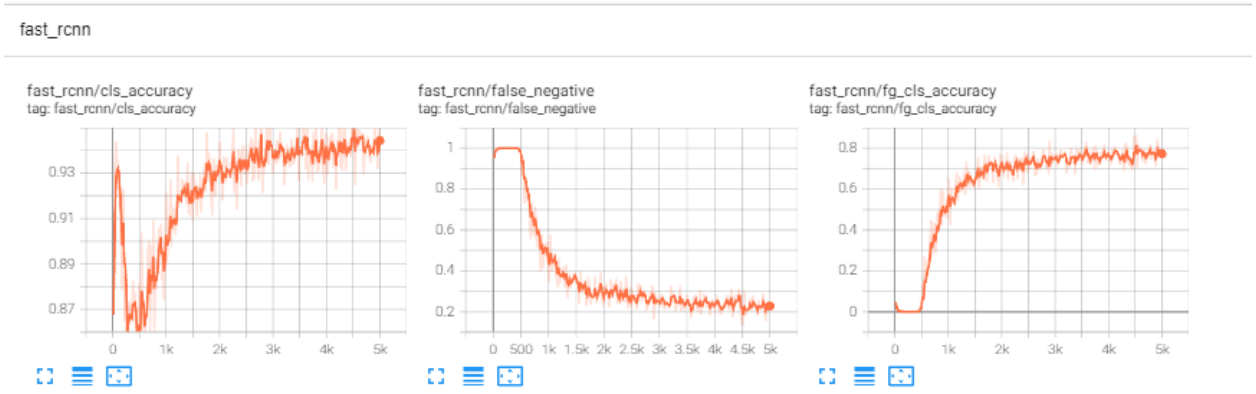
The account is Yide(Edward) Ma at Kaggle.The score I got is 0.09189. Here is some visualization in the training dataset. The mid column one is gt, the right column is pred_mask.

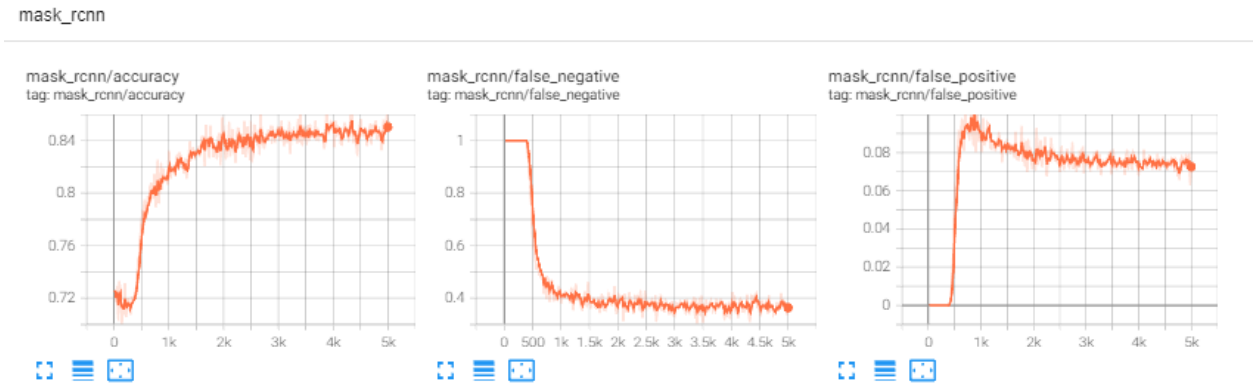


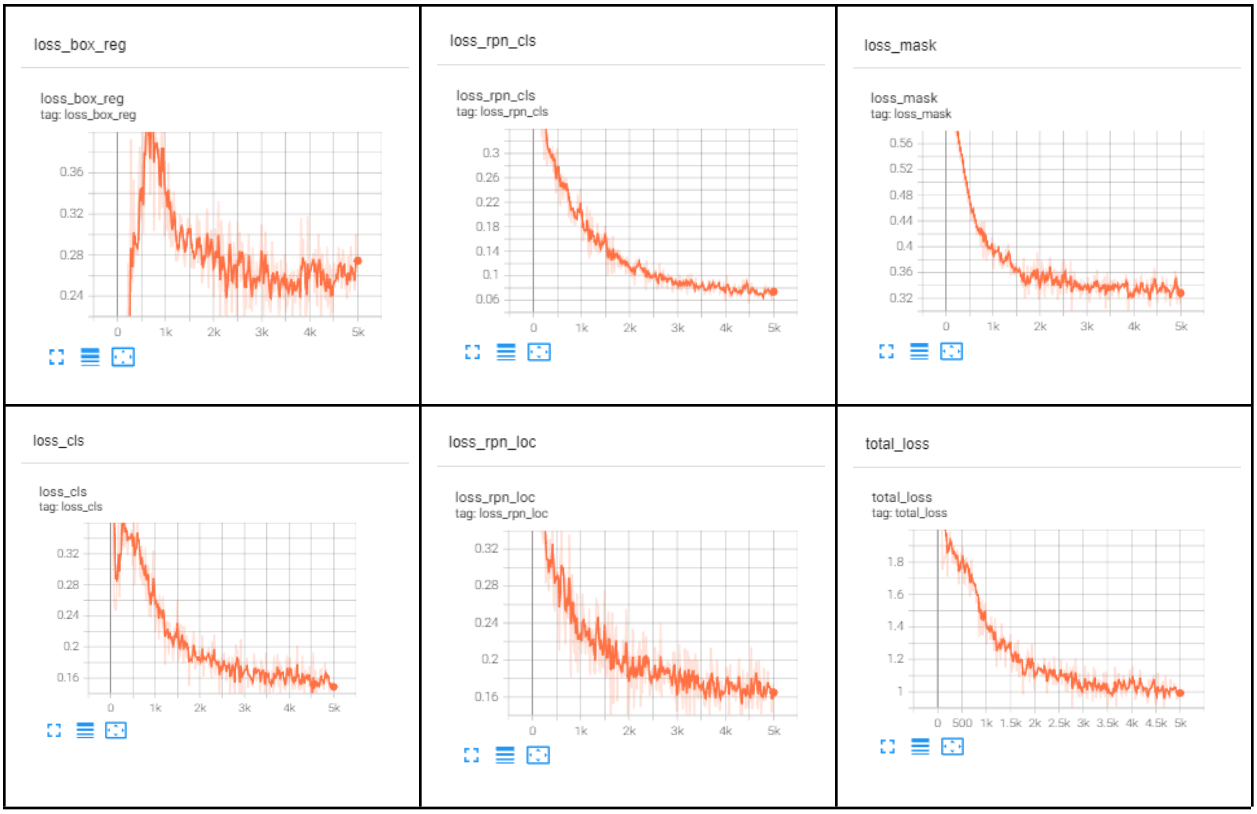
For testset(no gt) will be:



Part4
For this part,

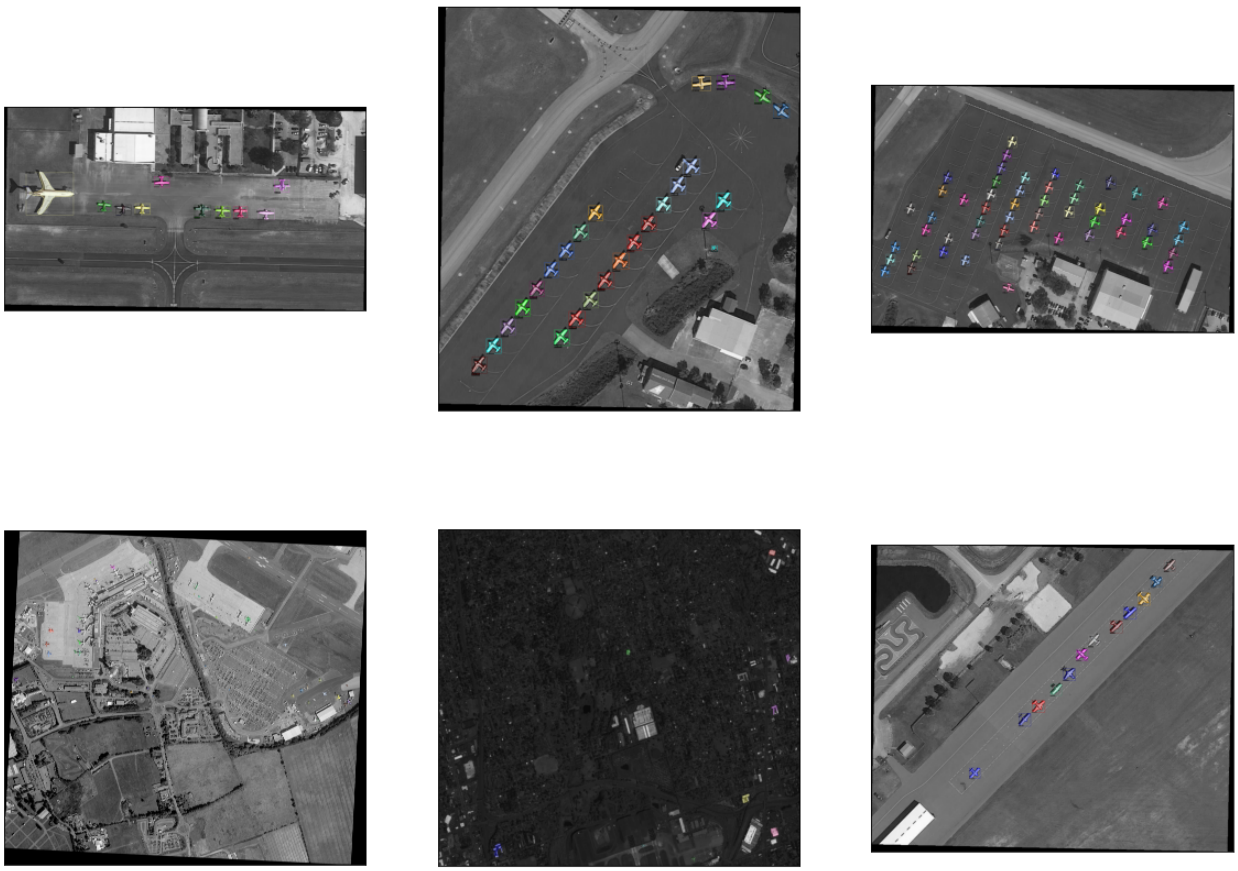






Visualization:

Visualizing 6 random prediction results



Comparison:

1. The inference time of this part4 end to end model is much faster than the detection+segmentation instance pipeline. Also, for part4 model, mask_rcnn we only need to train it once, but for the part3 pipeline, it requires two model to train. Also the end to end model needs less things to care about, it can directly handle images.

(Ablation study:For the part3 detection+segmentation instance pipeline, the training for detection model is about 1 hour for 5000 iteration, the segmentation instance takes about 2 hours for my 90 epoch pre-trained model(because I have used out of colab pro+, so I use the 90 epoch checkpoint as my pre-trained mode. For the end-to-end mask_rcnn, it only takes 40 mins for 5000 iterations. This is a huge improvement.,)

2. From this assignment, we can conclude an end-to-end model can have many benefits of doing the task step by step and build a pipeline. Not only from the convenience of training, converting of format, but also for training, since end to end is doing bp algorithm for the whole model, it can learn better features which leads to better performance.