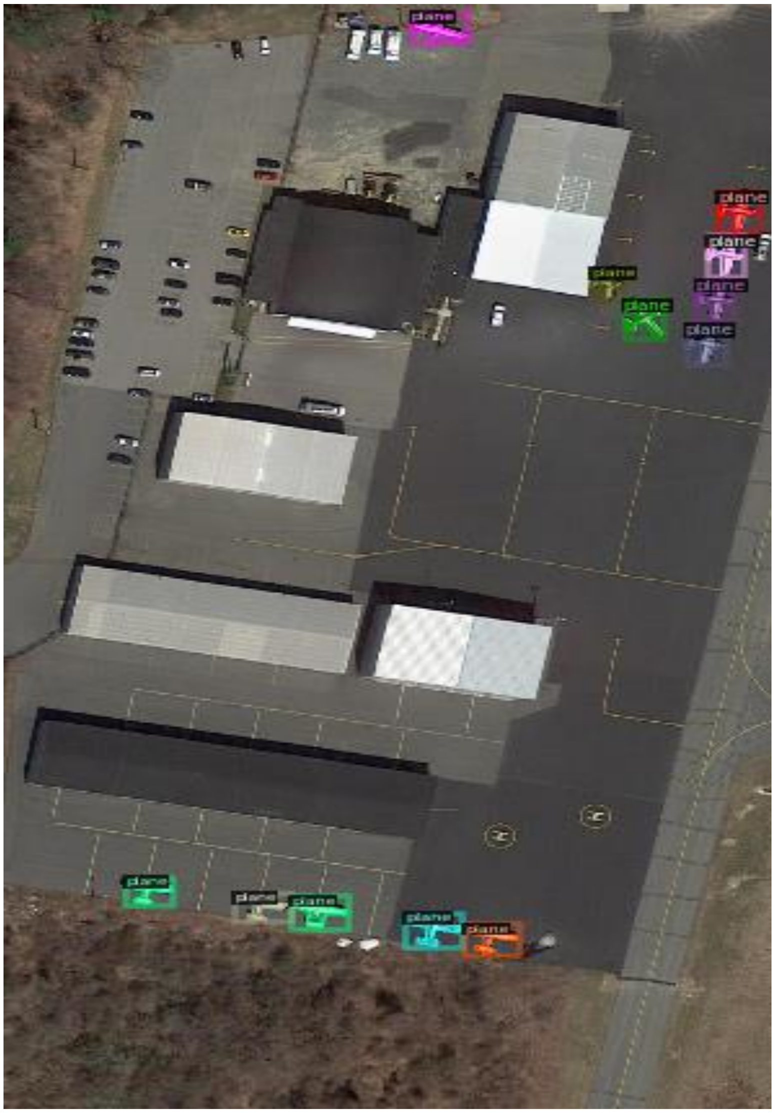
I want to use one free late day for this submission.

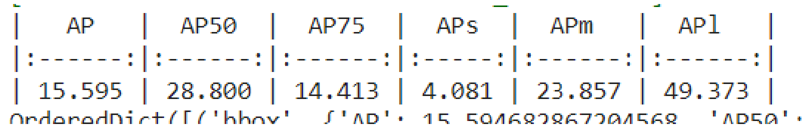
*I think sth in the shared dataset has deleted because I wanted to re-run my code last night (at9PM) and but didn’t work, so I uploaded all of the Test and Train image again to my Drive and this is the reason of the late submission as well.

First, we implement the Loader function and visualize on of Train set images and it has been tested to make sure functions work correctly:

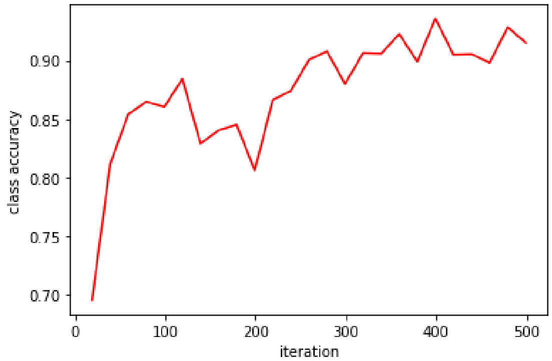


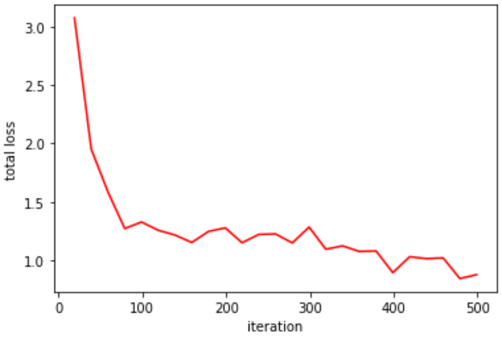
The above image shows that the implemented functions work correctly.

Then, the configuration has been set and “ faster_rcnn_R_101_FPN_3x.yml” has been used as a baseline with the following parameters: MAX_ITER = 500, BATCH_SIZE_PER_IMAGE = 512, IMS_PER_BATCH = 2, BASE_LR = 0.00025. In such circumstance, the evaluation results are as follow:

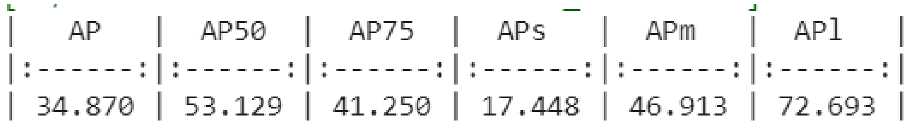


```
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 15.595 | 28.800 | 14.413 | 4.081 | 23.857 | 49.373 |
OrderedDict([('bbox', {'AP': 15.594682867204568, 'AP50':
```





Then, I changed some configuration and made, and finally this is the best result I got:



```
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 34.870 | 53.129 | 41.250 | 17.448 | 46.913 | 72.693 |
```

Figure 1: Best AP50 resulted from iteration 5000.

This result shows AP50 value 53.129. The training loss and accuracy plot are as follows:

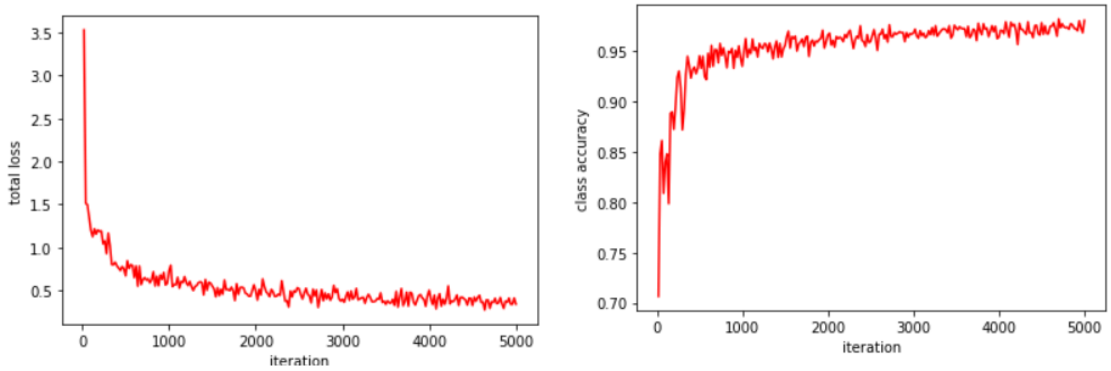


Figure 2: Loss and class accuracy obtained by mentioned hyper parameters.

As it can be seen, one of the reason of increasing the accuracy and as a result AP50 is increasing the iteration from the 500 to 5000. Following are the rest of the changes that I tried and the given results:

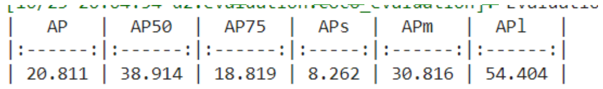
I tried the Faster_rnn_50_FPN and it had the less AP50 value rather than rnn_101 (it was near 25), so I continued my tries with the based on rnn_101 different models. This conclusion makes sense since deeper models (with approximately same architecture) leads to higher aP50 values in object detection. Therefore, followings are the models I have tried:

```
faster_rcnn_R_101_DC5_3x
faster_rcnn_R_101_FPN_3x
faster_rcnn_R_101_C4_3x
```

Also, I changed the LR from 0.00025 to 0.001 and the best LR was 0.0009 for my final result. I also split the Train dataset to train and validation with the ration of 193/50. I also tried to divide the images to the smaller sizes (256,256) but it leaded to the worst results. I believe that this happened because in some cases, the image has cut in the middle of the object, leading to the less samples and therefore worst results.

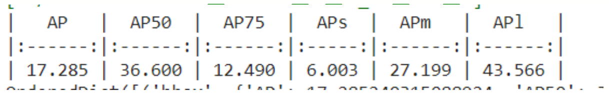
Following some of the results I got with different baseline models for iteration 1000 has been shown:

Method: faster rcnn R 101 DC5 3x



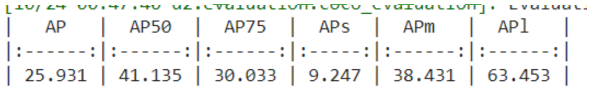
```
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 20.811 | 38.914 | 18.819 | 8.262 | 30.816 | 54.404 |
```

Model: faster_rcnn_R_101_C4_3x



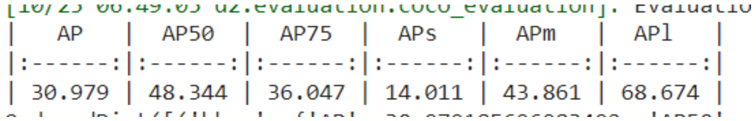
```
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 17.285 | 36.600 | 12.490 | 6.003 | 27.199 | 43.566 |
```

Model: faster rcnn R 101 FPN 3x



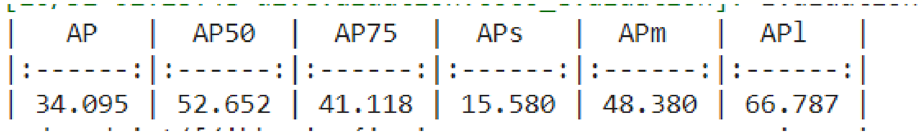
```
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 25.931 | 41.135 | 30.033 | 9.247 | 38.431 | 63.453 |
```

Iteration 1500, faster_rcnn_X_101_32x8d_FPN_3x model:



```
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 30.979 | 48.344 | 36.047 | 14.011 | 43.861 | 68.674 |
```

Iterations 4000:

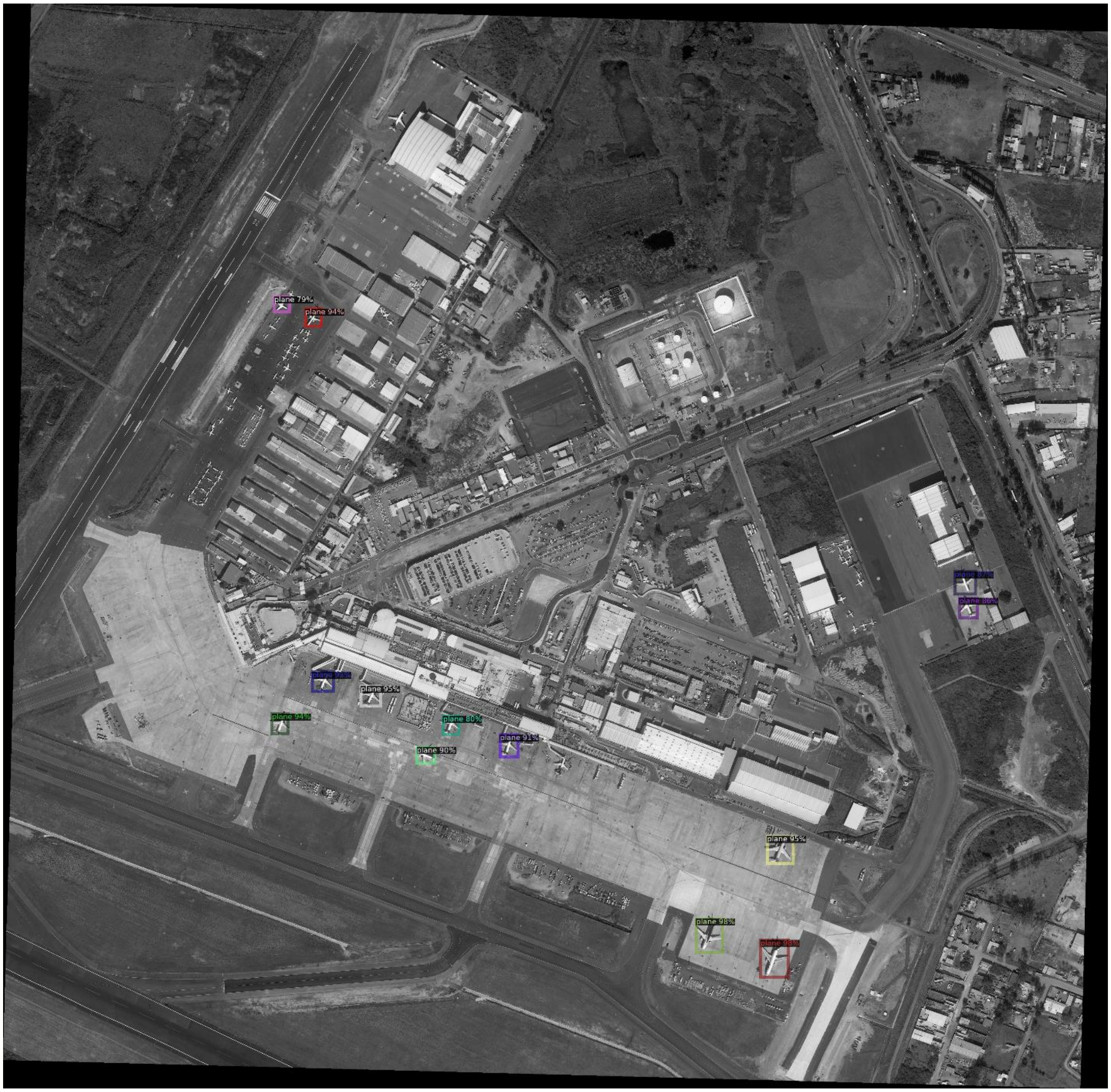


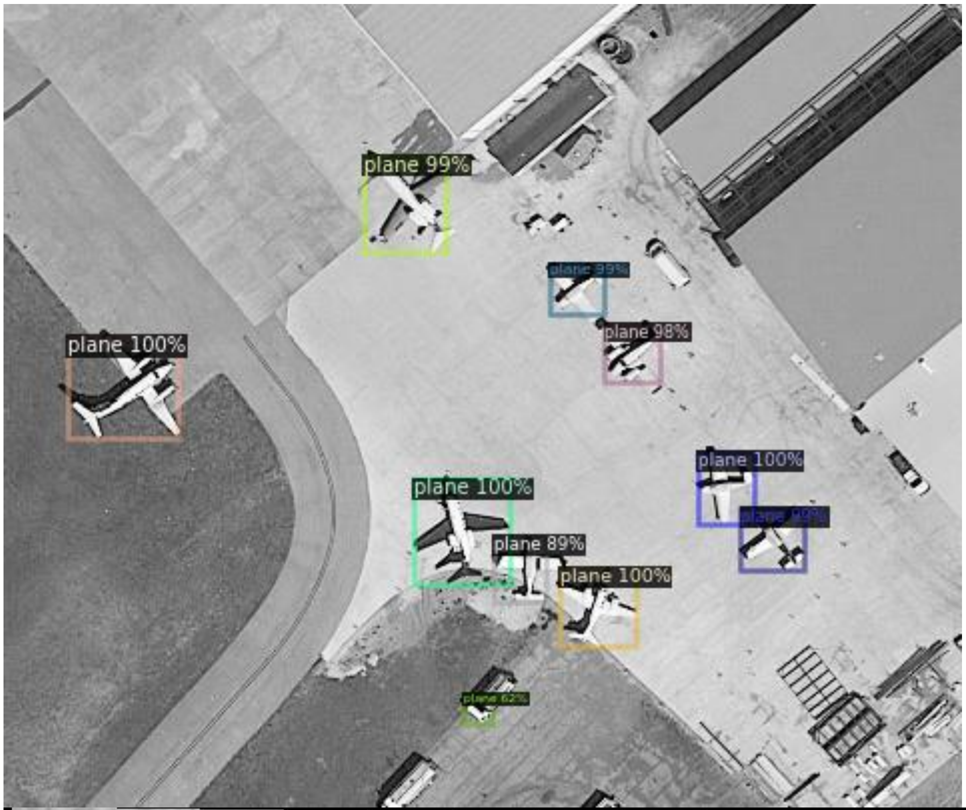
```
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 34.095 | 52.652 | 41.118 | 15.580 | 48.380 | 66.787 |
```

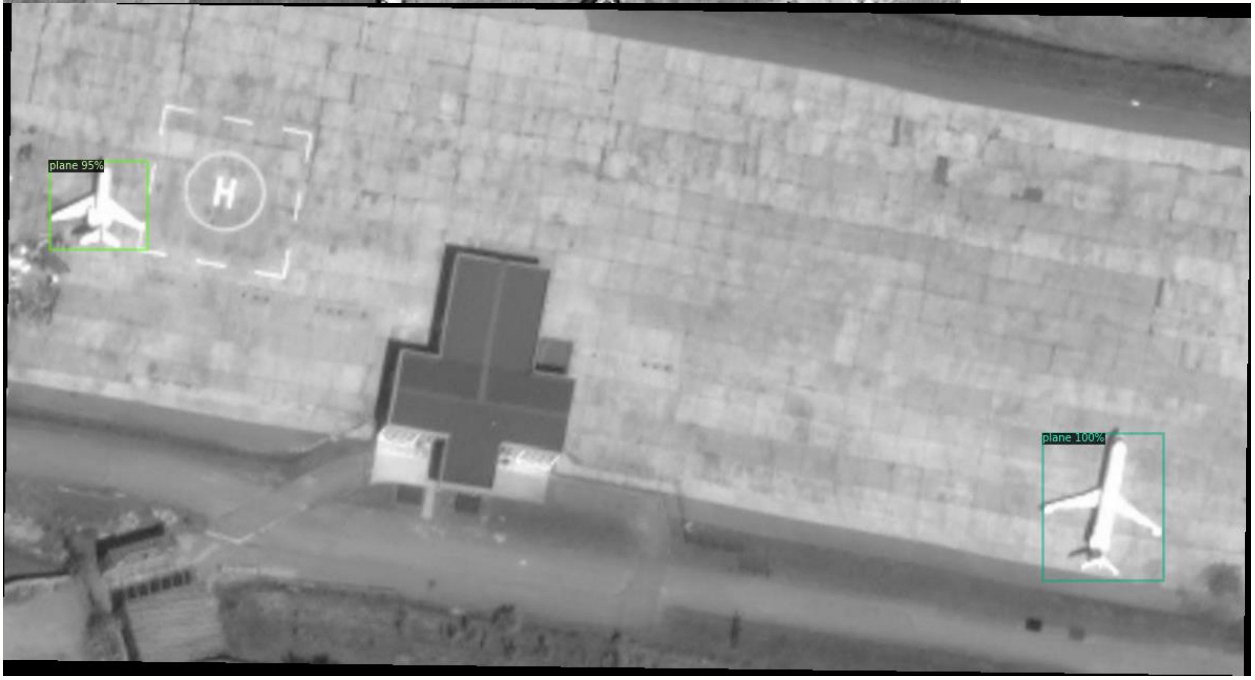
Since with the same iteration and other hyper-parameters X_101_32x8d_FPN model had the best result, I continued with this model and increased the iteration to be 4000 leading to the 52.652 AP50 value.

I also visualized three samples:

Iteration: 4000

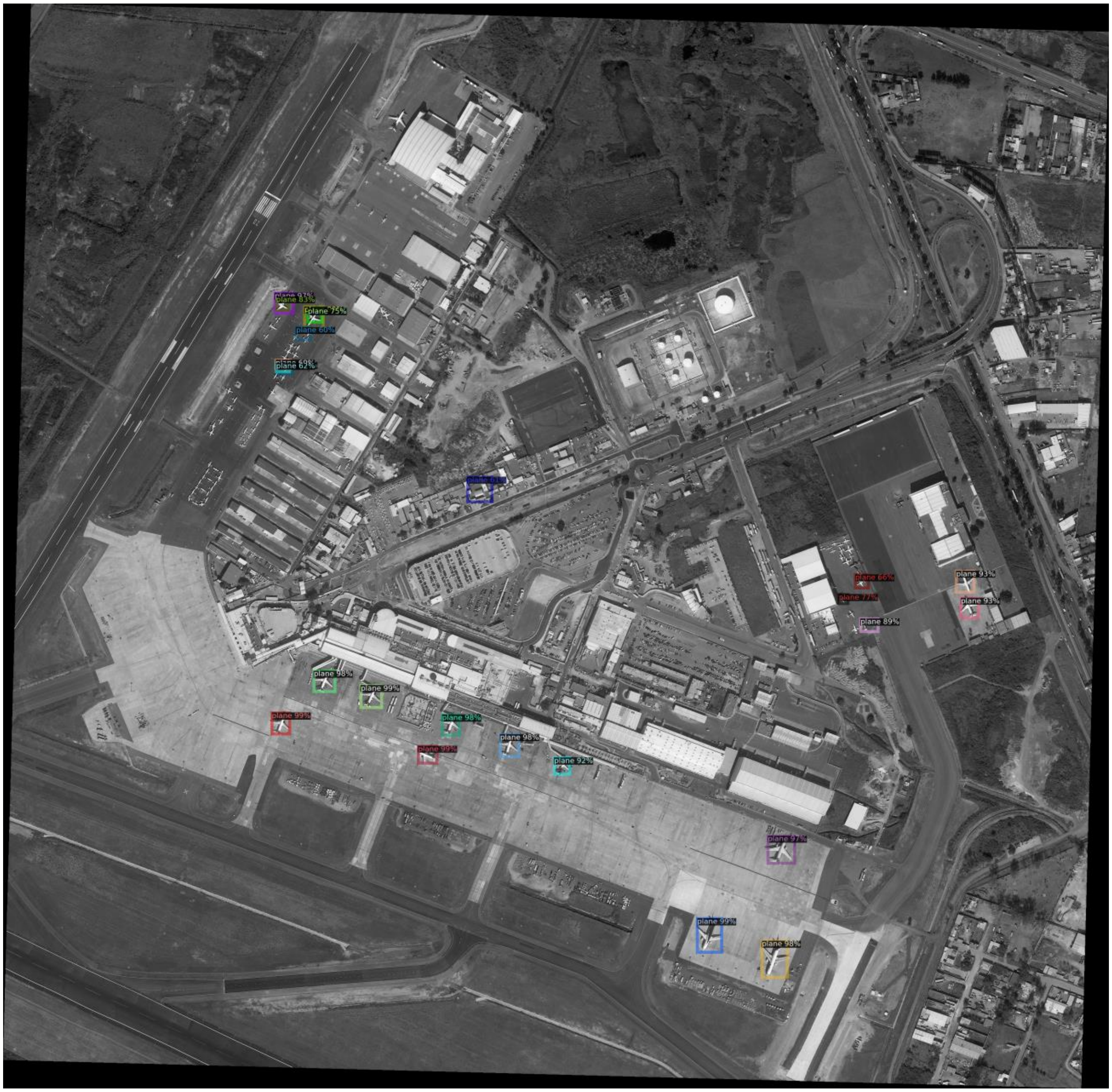

plane 79%
plane 94%
plane 87%
plane 88%
plane 95%
plane 94%
plane 80%
plane 91%
plane 90%
plane 95%
plane 98%
plane 98%



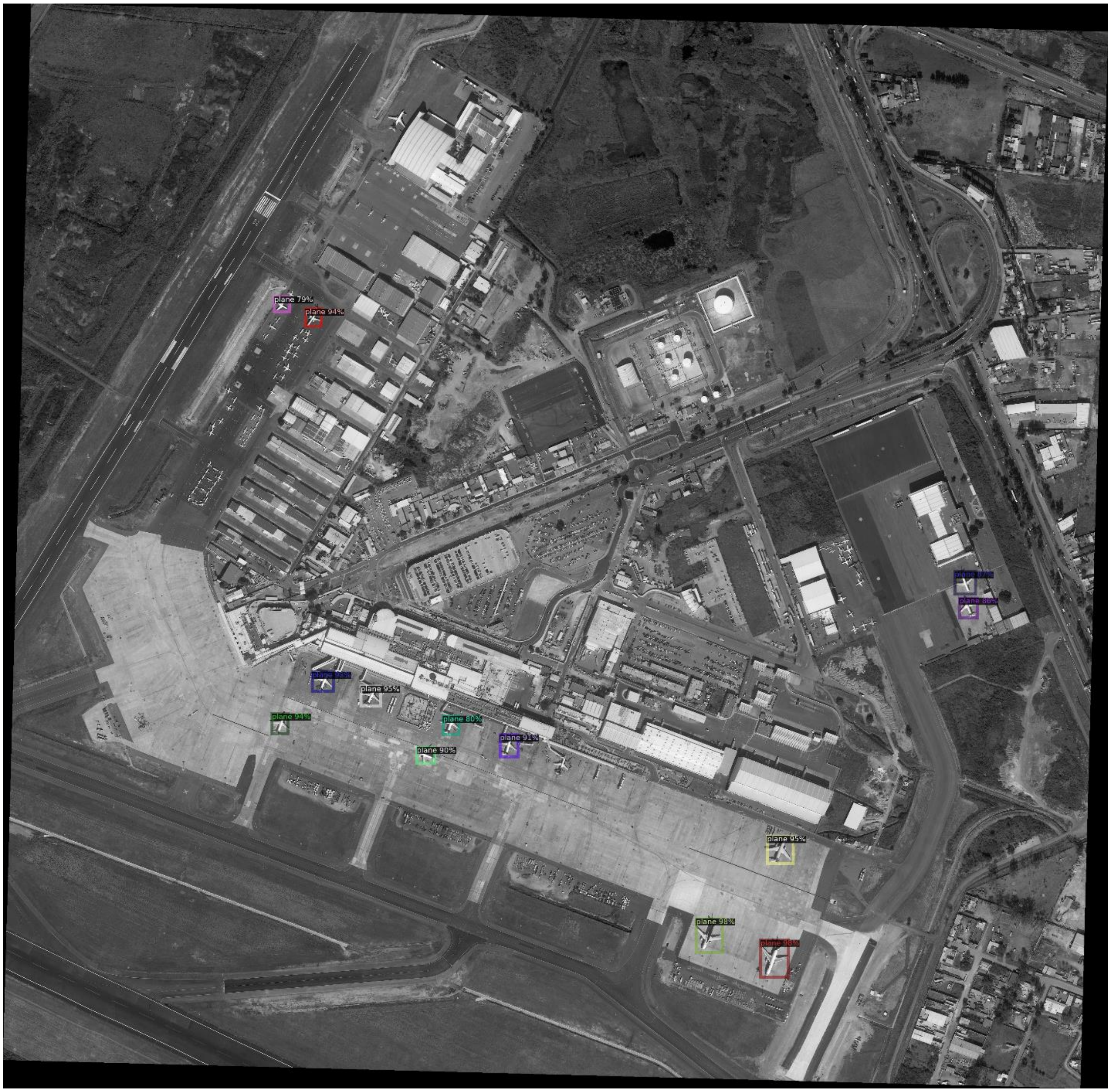


Also, for the ablation study, I have compared the difference between the output of two same baseline model with different iteration:

Iteration: 500



Iteration 4000:

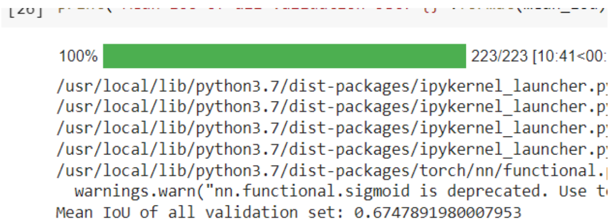


As we saw, the AP50 value has increased from 48 to 53 by increasing the iterations. Also, in the resulted images we can see that the when the iteration is 500, the number of objects detected as a plane are more but they're not accurate. Above figure shows that some planes has been counted duplicate or in some cases the house has counted as a plane. The model in the output has removed this inaccuracy results by iteration 4000, results are more reliable, and AP50 values increases as a result.

Another things I tired were changing Learning Rate, Different baseline models, splitting to the blocks and so on.

Part 2:

First, I tried the network that has been written as a bias. In this case, the mean of IOU calculated as a 0.67.



```
100% 223/223 [10:41<00:
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.p
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.p
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.p
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.p
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.
  warnings.warn("nn.functional.sigmoid is deprecated. Use t
Mean IoU of all validation set: 0.6747891980007953
```

Following schematic shows the architecture I used for my network. It has consisted of 5 down sample-convolution and then up-sampling-conv. Since the input dimension is high and the image is a high-resolution one, I have done the down-sampling first.

	layer	output
1	conv(3,64)	64*128*128
2	down1(64,64)	64*64*64
3(merge)	conv(64,128)	128*64*64
4	down(128,128)	128*32*32
5(merge)	conv(128,256)	256*32*32
6	down(256,256)	256*16*16
7	conv(256,512)	512*16*16
8	down(512,512)	512*8*8
9	conv(512,1024)	1024*8*8
10	down(1024,1024)	1024*4*4
11	conv(1024,2048)	2048*4*4
12	conv(2048,1024)	1024*4*4
13	up(1024,512)	1024*8*8
14	conv(1024,512)	512*8*8
15	conv(512,512)	512*8*8
16	up(512,512)	512*16*16
17	conv(512,512)	512*16*16
18	conv(512,256)	256*16*16
19	up(256,256)	256*32*32
20 (merge with layer 5)	conv(256,256)	256*32*32
21	conv(256,128)	128*64*64
22	up(128,128)	128*64*64
23	conv(128,128)	128*64*64
24	conv(128,64)	64*64*64
25(merge with layer 3)	up(64,64)	64*128*128
26	conv(64,64)	64*128*128
27	conv(64,32)	32*128*128

28	conv(32,1)	1*128*12821

In addition, the hyper-parameters are:

batch_size: 8, Num_epochs: 6, Learning_rate= 0.005, Weight_decay: 0.00001, loss: max_entropy. Every epochs take 35 minutes to run. I tried to increase the number of epochs but I couldn't since the google colab resources are limited. Following image shows the Loss of each iterations:

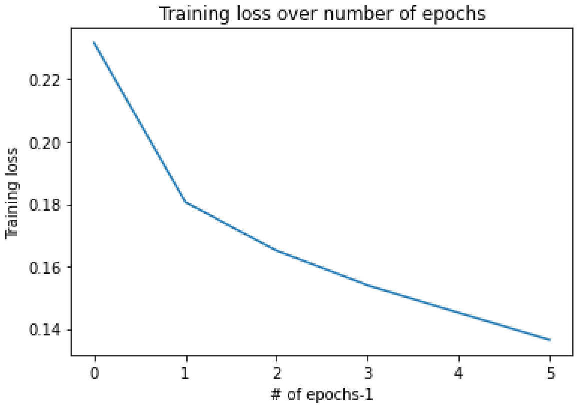
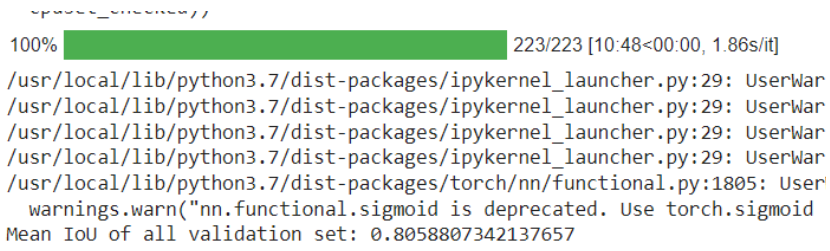


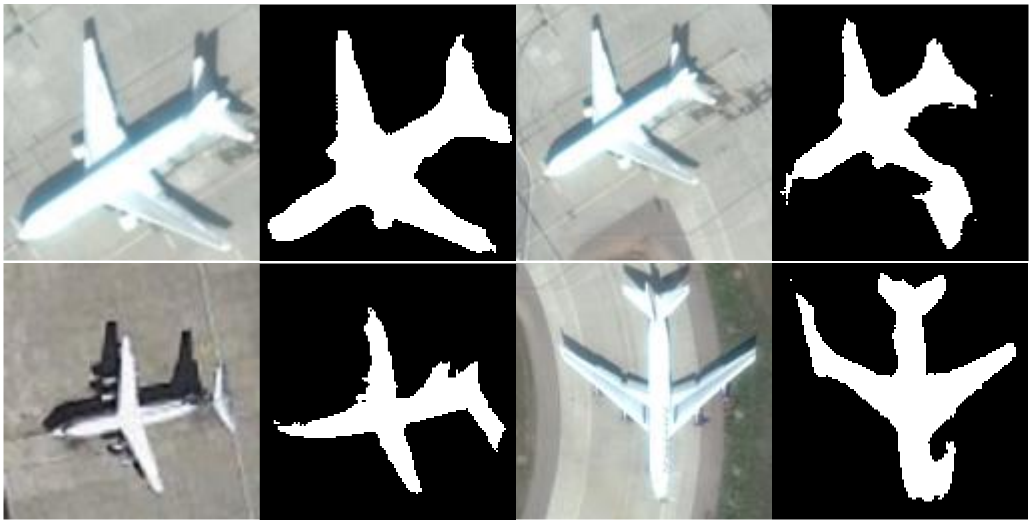
Figure 3: Total training loss

The obtained mean of IOU for validation in this part is 80%:



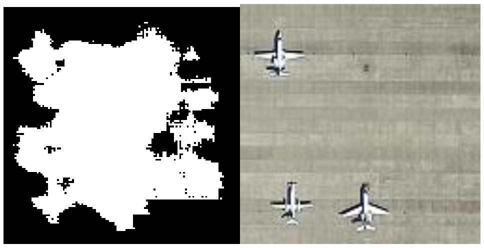
```
  cpuset_checked))
100% 223/223 [10:48<00:00, 1.86s/it]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:29: UserWar
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:29: UserWar
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:29: UserWar
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:29: UserWar
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1805: User
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
Mean IoU of all validation set: 0.8058807342137657
```

I also cropped some images and set them to be an input of designed network and here are the results:



As it can be seen, the developed network could segment images properly.

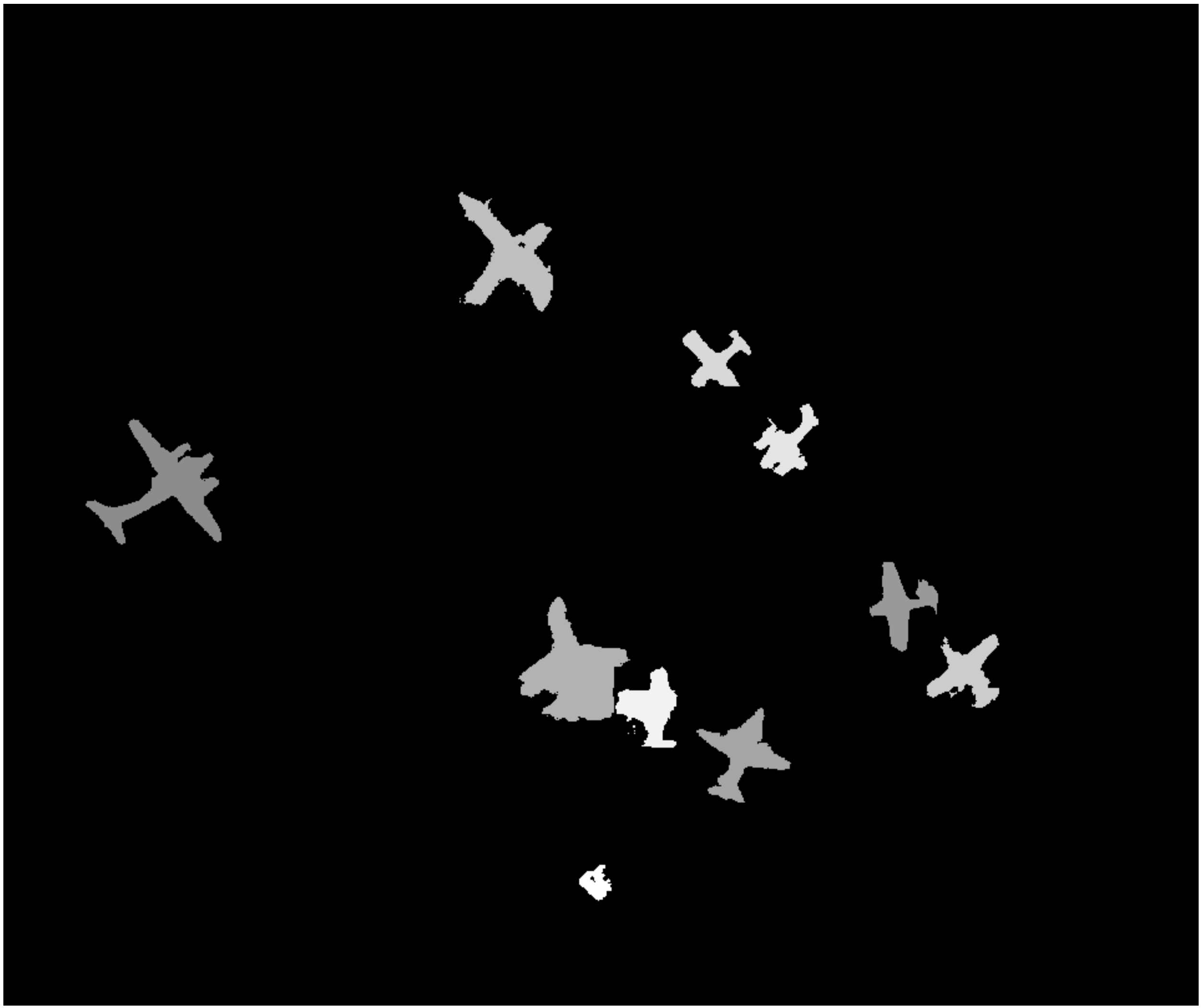
As we expected, this part fails for multi segmentations.

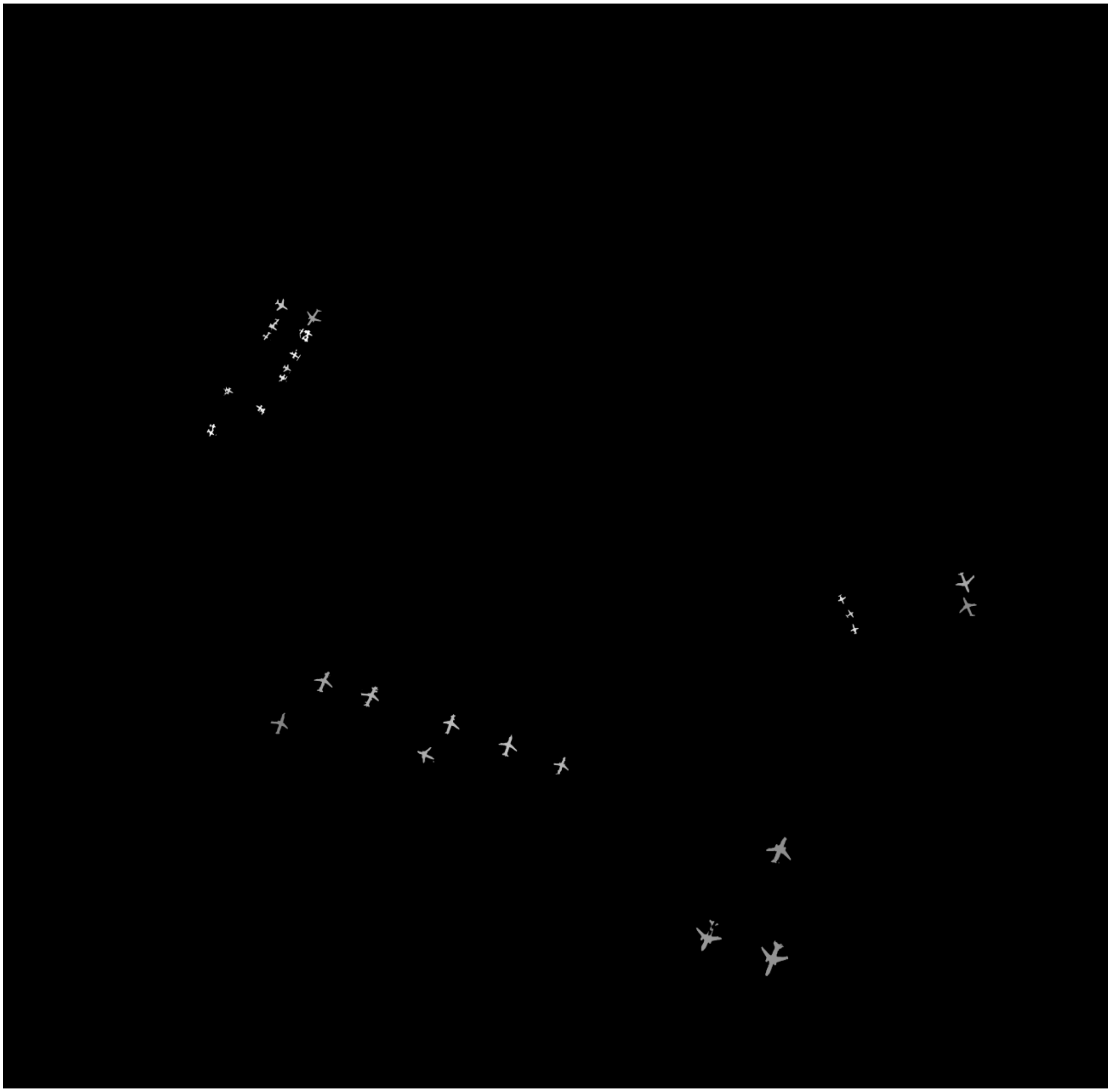


part3)

My final work has submitted as a name of **Mahdie Ghane(Private name)** and **<u>Migmig (public name)</u>** on Kaggle competition page. The score of my final submission is 0.47039. Also, my rank was 17.

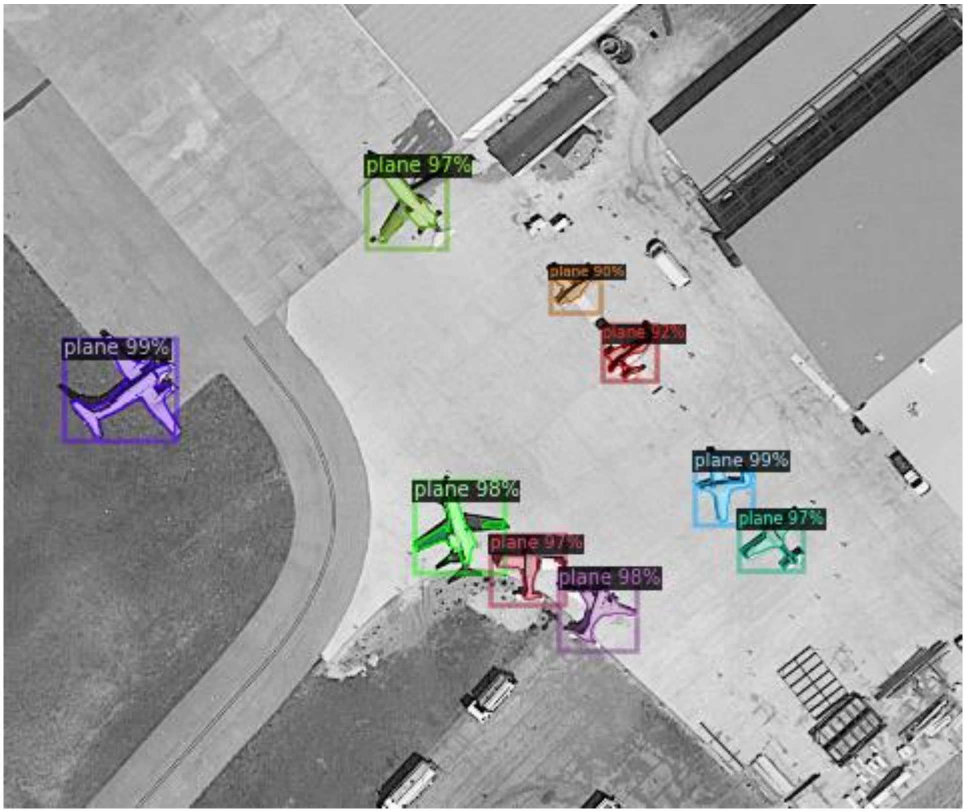
The three test samples has been shown below:

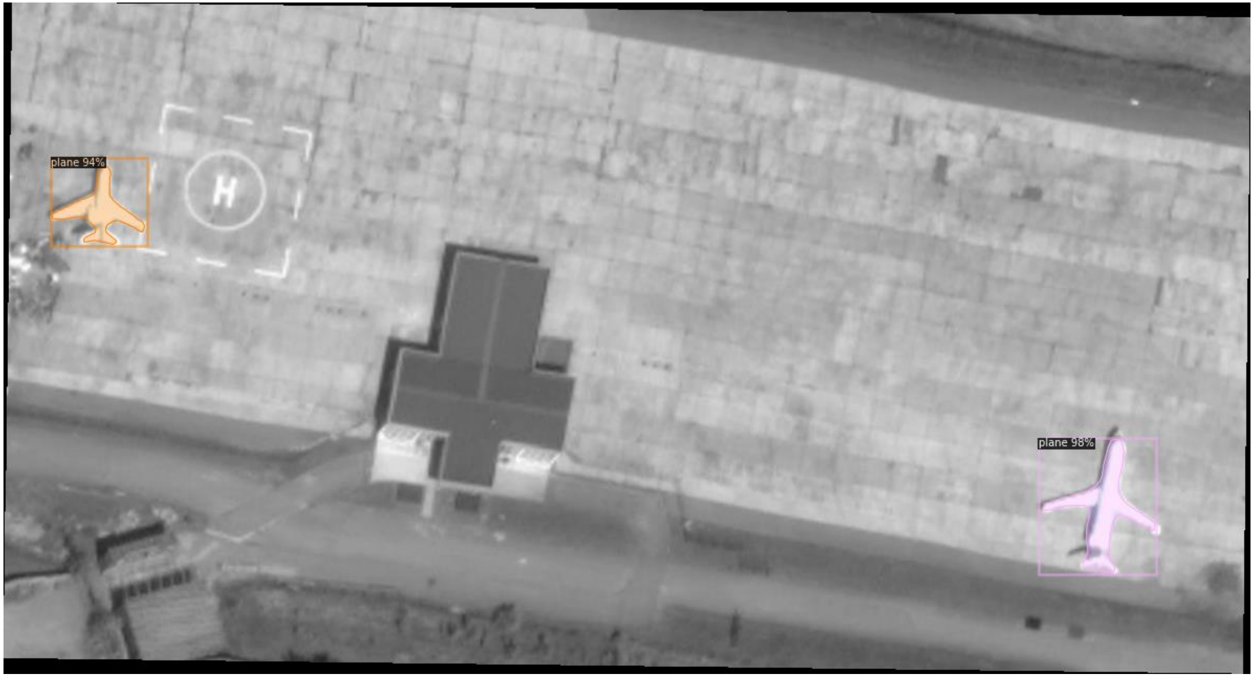


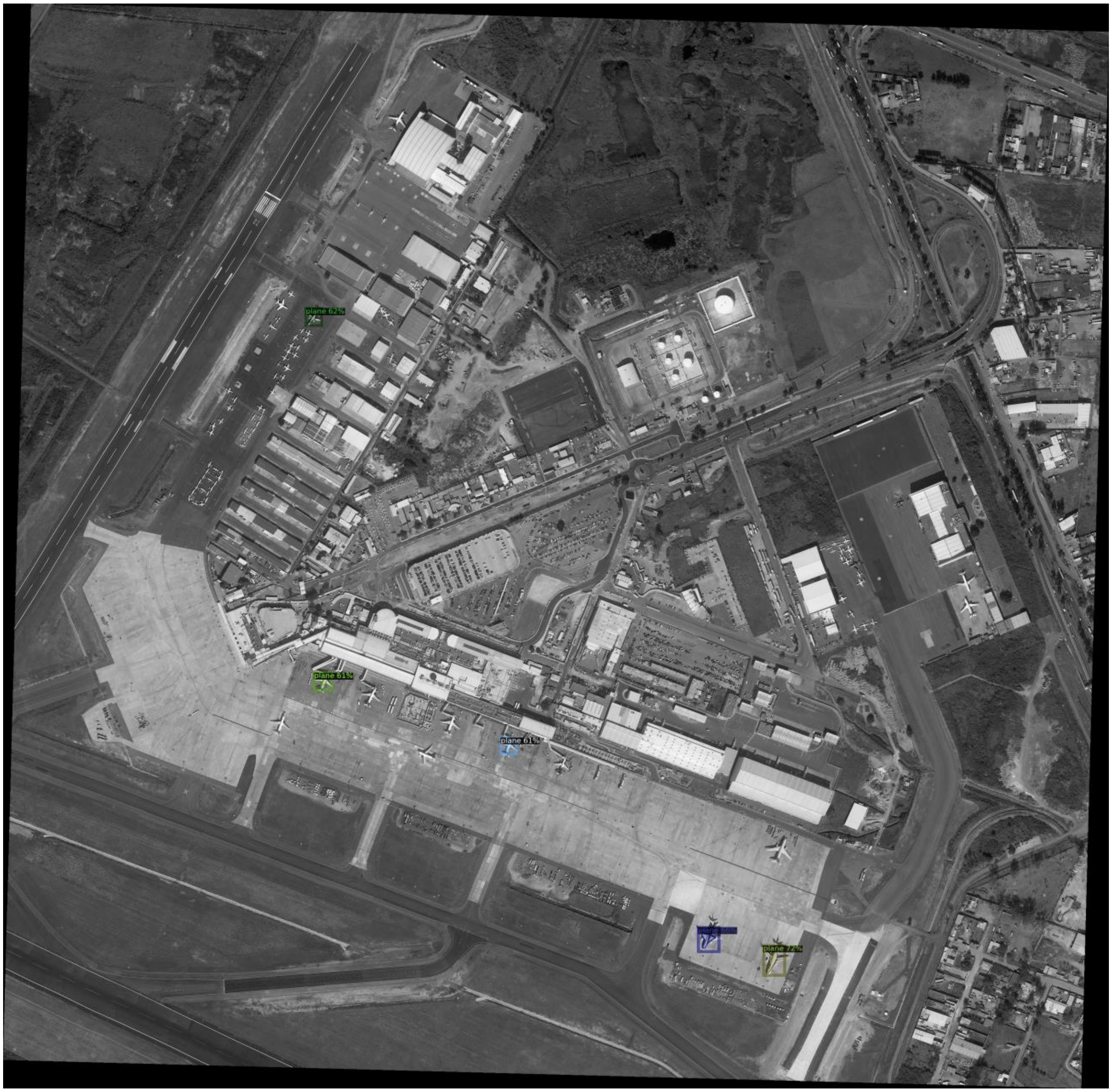


Part4)

In this part, Mask R-CNN has been implemented and here are the detection results:


plane 97%
plane 90%
plane 99%
plane 92%
plane 99%
plane 98%
plane 97%
plane 97%
plane 98%


plane 94%
plane 98%



Also, the accuracy and loss plot has been obtained as a bellow:

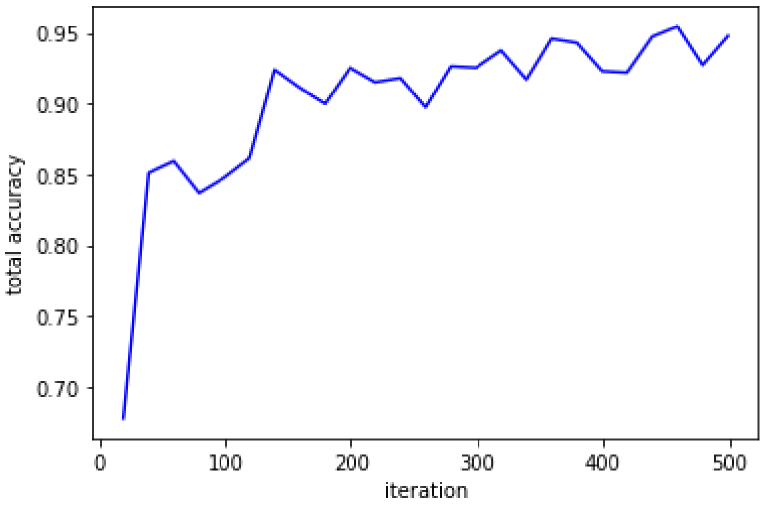


Figure 4: Total accuracy

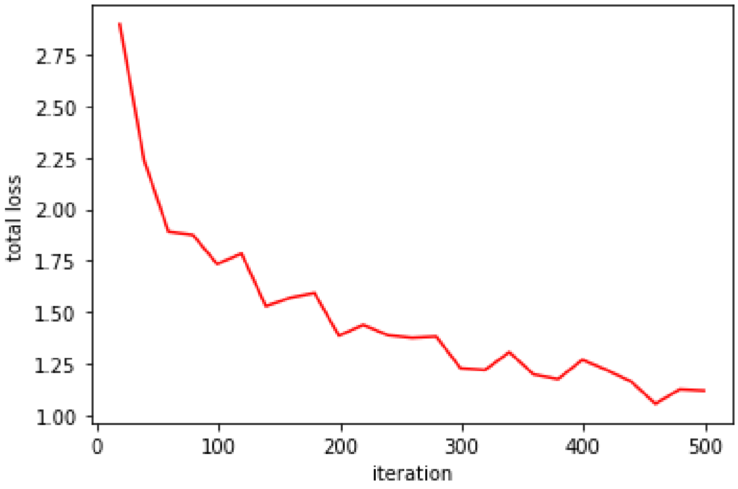


Figure 5: Total loss

Comparison with part1:

As it can be seen, both parts were able to detect planes properly; however, the part 1 has a better results. It could be because of a deeper network (in part1, we had R-101 but in part 4, we have tried R-50). In addition, in part 1, when there is R_101 model with with 500 iterations like part 4, we can see there are many flaws in detecting planes. For example, looking at two following images illustrate two points which mentioned:

part4:

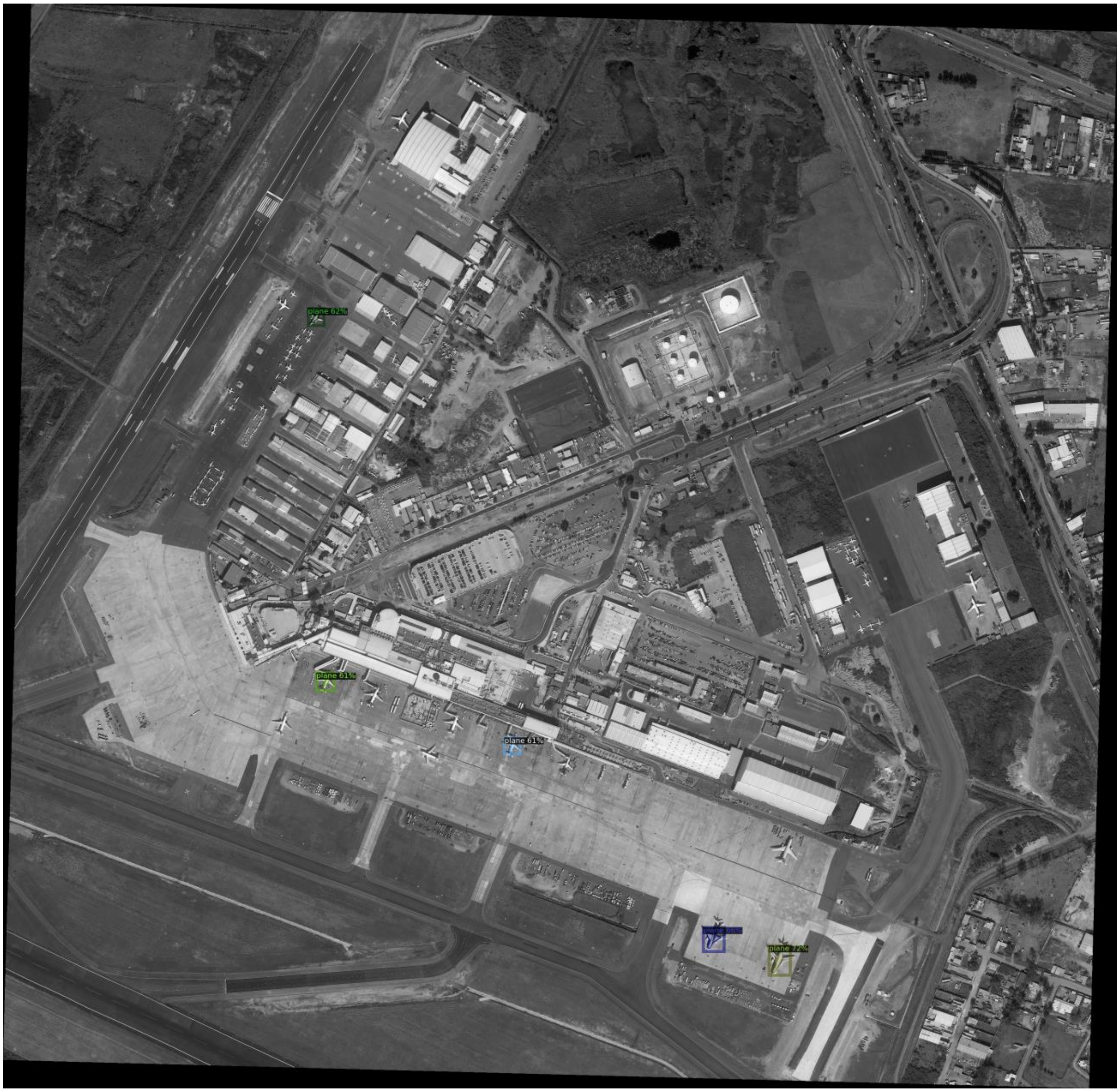


Figure 6: one of part 4 visualized images.

part1:

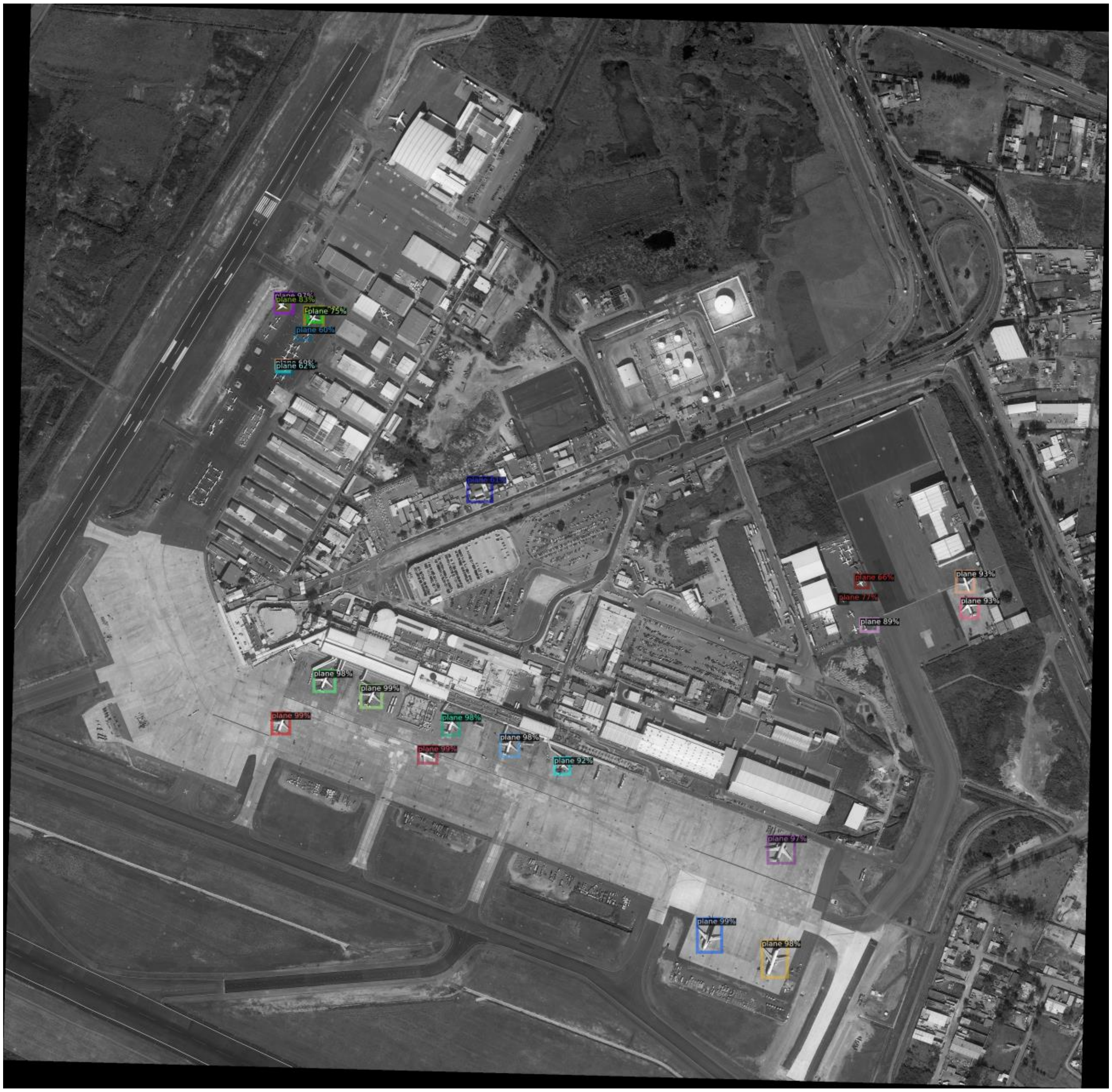
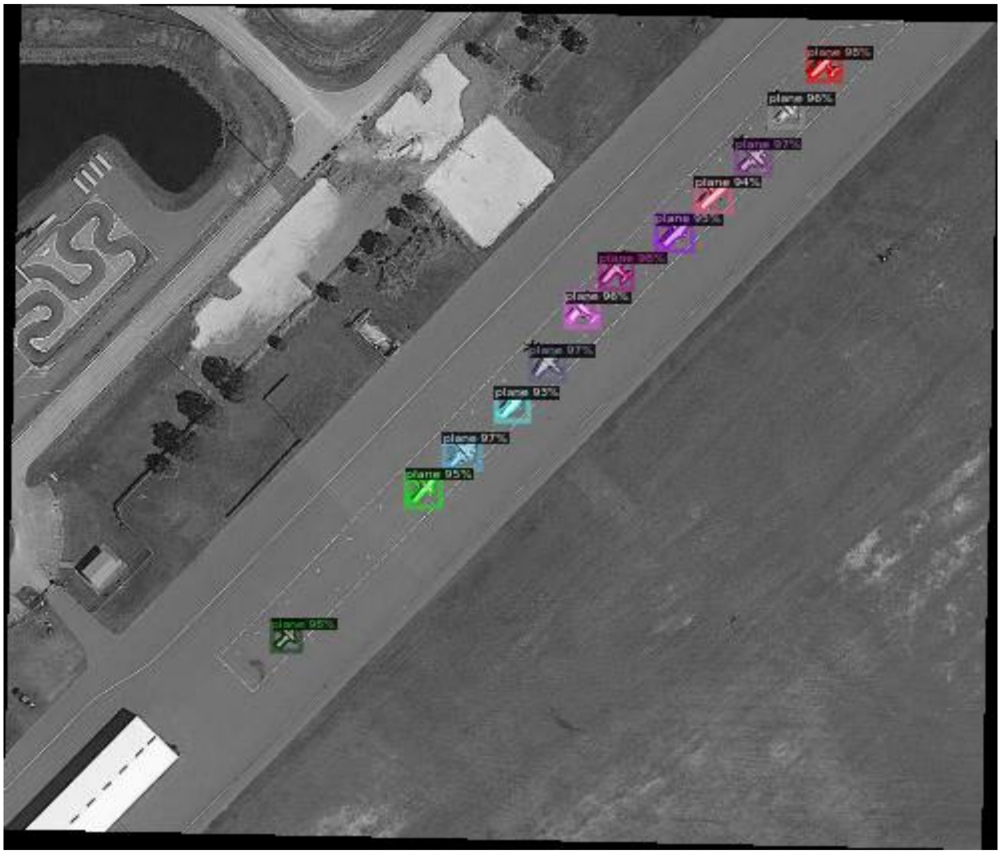


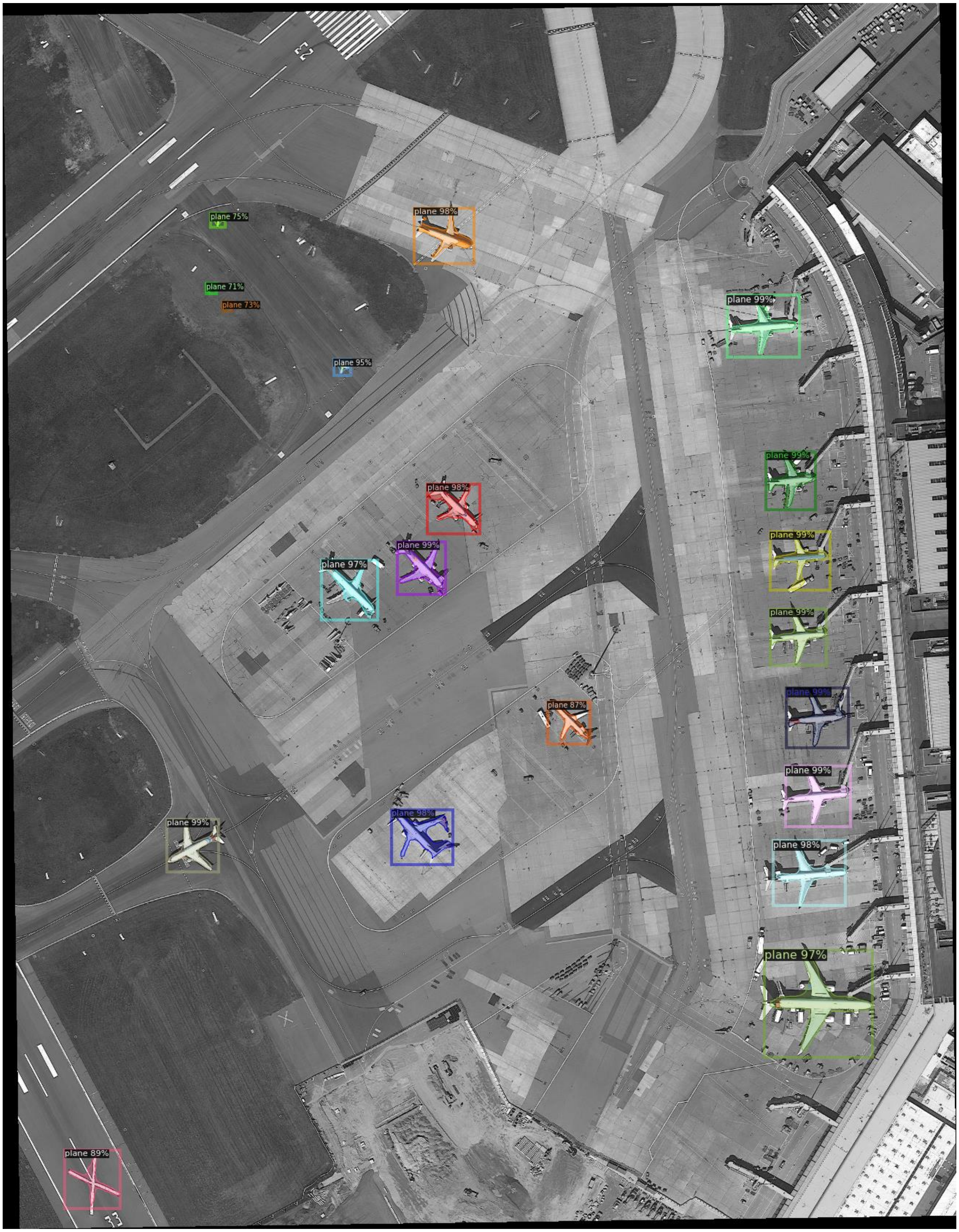
Figure 7: There are many flaws in part 1.

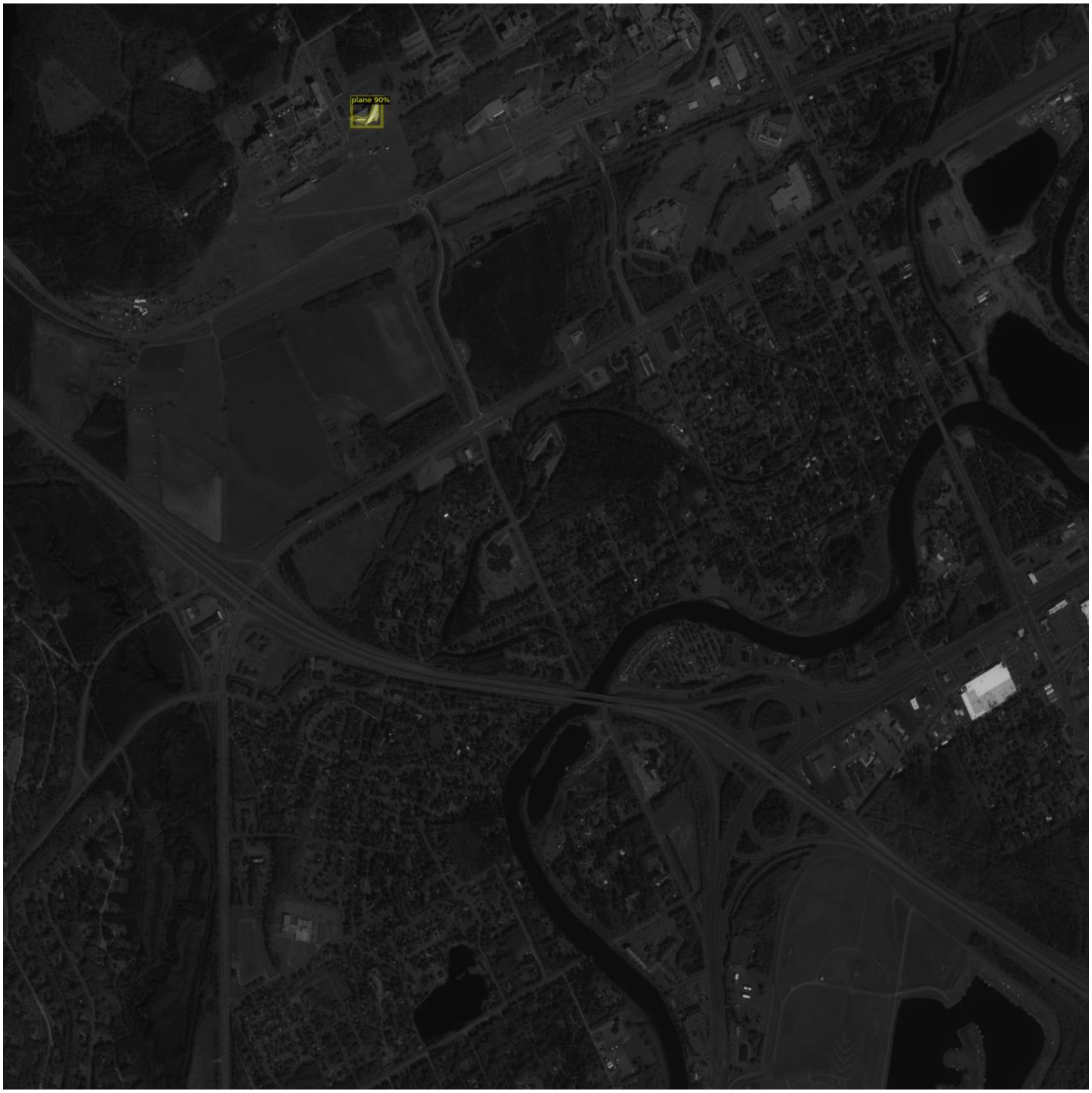
Comparing part 3 and part 4:

As it can be concluded from the output of part 3 and part 4, in part 3, there are many FP in the images, meaning that although they are not part of the mask, they have labeled. In part4, there is a many FN ones, meaning the area were part of the object (plane) didn't detect and segment properly.

Three more images for part 4


plane 95%
plane 96%
plane 97%
plane 94%
plane 93%
plane 96%
plane 96%
plane 97%
plane 93%
plane 97%
plane 95%
plane 95%


plane 98%
plane 75%
plane 71%
plane 73%
plane 99%
plane 95%
plane 99%
plane 98%
plane 99%
plane 99%
plane 97%
plane 99%
plane 99%
plane 87%
plane 99%
plane 99%
plane 98%
plane 98%
plane 97%
plane 89%


plane 90%