Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847

Project 3: Semantic Segmentation

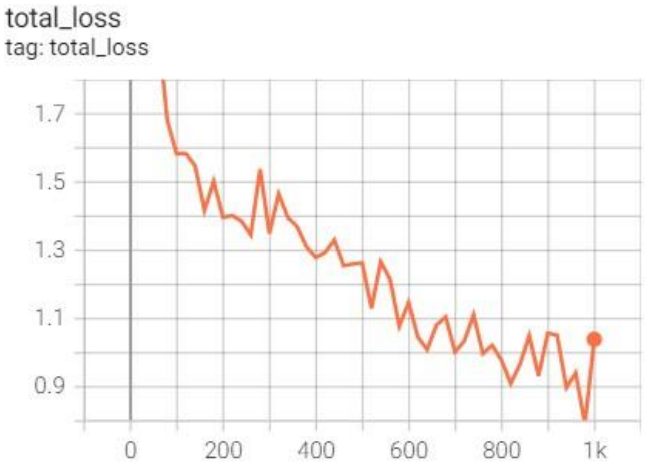
Part 1:

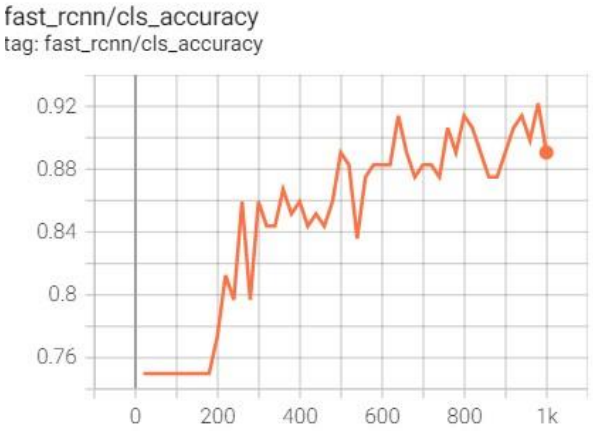
1) List of configs used:

For my model I used the "`faster_rcnn_R_101_FPN_3x.yaml`" as suggested, but I changed some configurations changes for high accuracy

Firstly, I increased MAX_ITER to 1000 and then reduced BATCH_SIZE to 32. I let rest of the configurations to remain same.

2) Just by increasing the number of iterations, the accuracy increased as the model was trained longer. Reducing batchsize per image also helped a lot as a smaller number of images ran through the model at a time, hence they were generalized less which also reduced loss while training longer.
3) Ploting total training loss and accuracy for the model

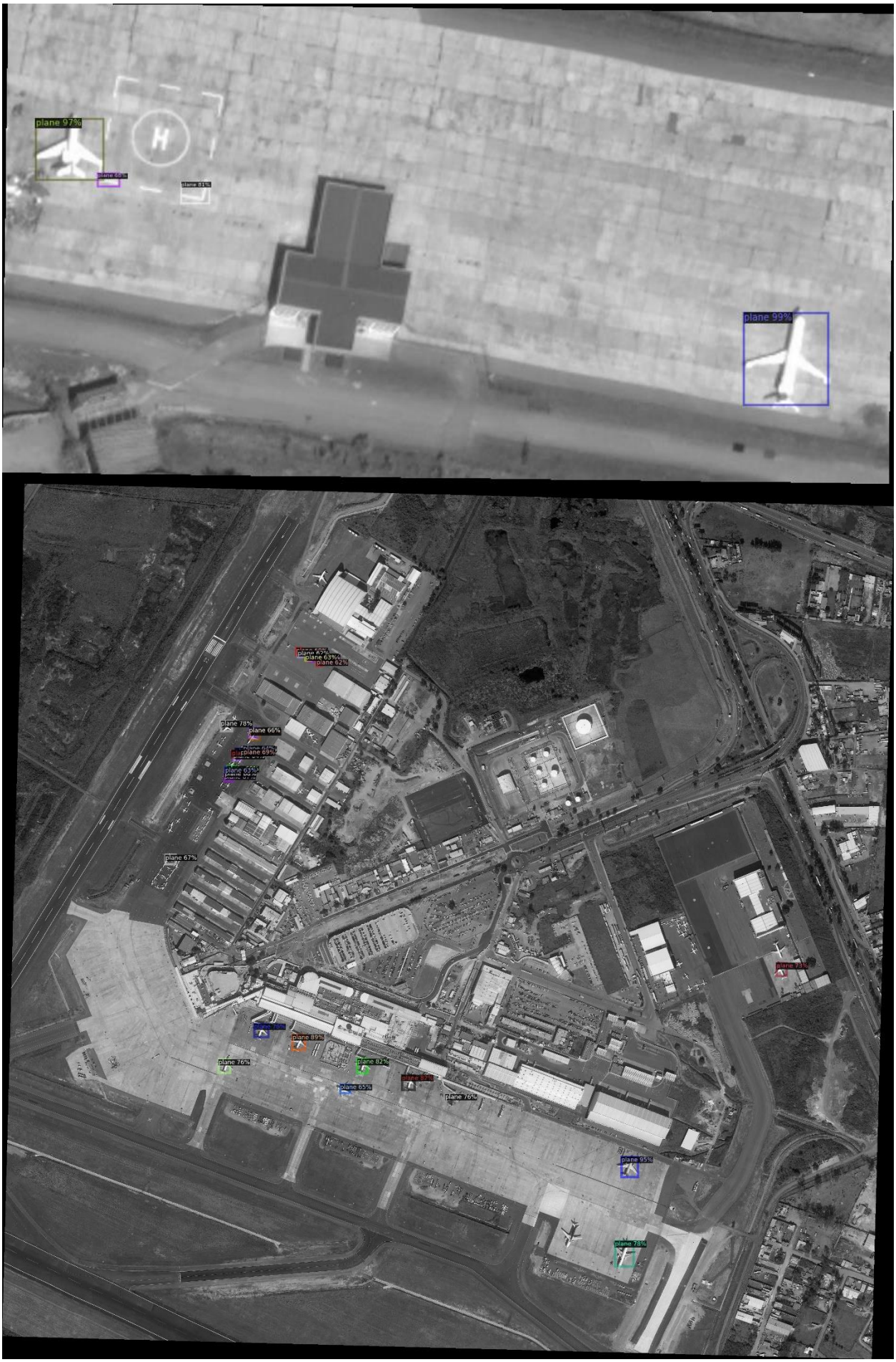




4) Visualizing test sample:

Vaibhav Saini
301386847

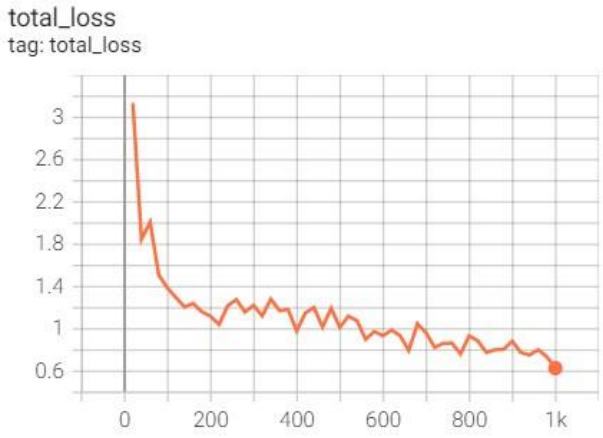
Late days used: 2
Team Name: 301386847

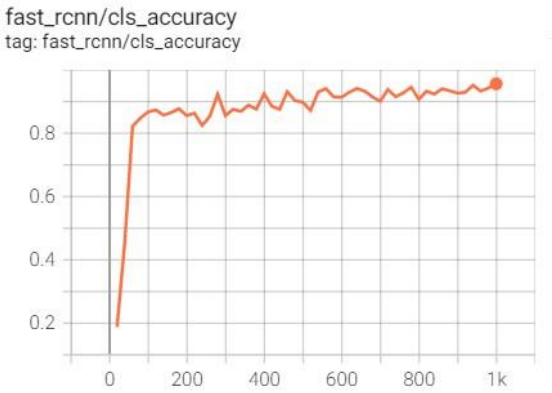


From the test images we can note that while the model is able to object planes in the images, there are also misidentifications along with failing to detect some images made by the model. As seen in the image 2, the model misidentified 2 corners of a helicopter pad as planes. Whereas in image 3, the model failed to detect some planes.

5) Now conducting an ablation study trying to understand how batchsize affect a model. We can run our model again with BATCHSIZE set as 512 and see what happens.

After training the new model, we can get plot its loss and accuracy as





Comparing the total_loss and accuracy plots for the models, we see that batchsize512 model's plots for total_loss and accuracy are very smooth and gradual compared to batchsize32 model. However, looking at the plots we can also say the accuracy of batchsize32 model is a bit better.

If we look the APs for both the models

```
[11/07 08:12:55 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[11/07 08:12:55 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.18 seconds.
[11/07 08:12:55 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/07 08:12:55 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 ] = 0.240
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.429
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.243
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.149
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.305
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.551
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.014
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.114
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.270
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.155
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.336
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.676
[11/07 08:12:55 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 24.017 | 42.879 | 24.266 | 14.916 | 30.493 | 55.077 |
```

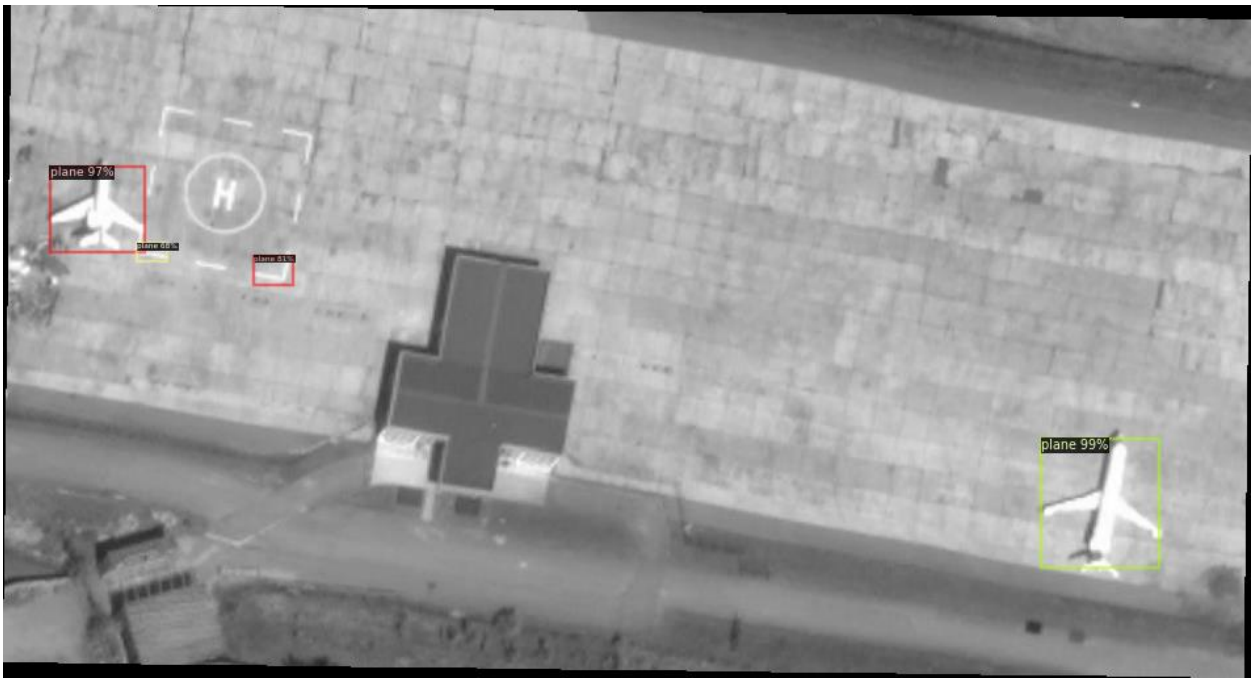
```
[11/07 07:20:58 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[11/07 07:20:58 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.20 seconds.
[11/07 07:20:58 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/07 07:20:58 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.238
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.403
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.257
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.137
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.318
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.559
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.014
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.115
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.267
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.136
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.347
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.693
[11/07 07:20:58 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 23.832 | 40.276 | 25.665 | 13.725 | 31.794 | 55.903 |
```

AP50 of BATCHSIZE32 model is much better than the other model.

Getting the same test images for BATCHSIZE512 model and comparing them with BATCHSIZE32

Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847



Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847

From the test images it becomes clear that both the models perform almost similarly and made the same mistakes. However, prediction accuracies are sightly better for BATCH32 model.

Part 2:

1) For training the model, I set up hyperparameters as num_epochs = 5, batch_size = 64, learning_rate = 0.001, weight_decay = 1e-5 and I used Adam optimizing algorithm.
2) In the network, first I add more down layers going from 3 channels to 16 and then 32, 64, 128, 256, 512 and 1024. Then , to match with the downsampling I added up layers from 1024 back to 3. I also added skip connections between corresponding channels. Lastly, I created a new class called block which pretty much does (convolution, batchnorm, relu)x2 for same in/out channels just to have more convolution the networks.

Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847

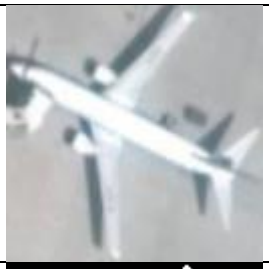
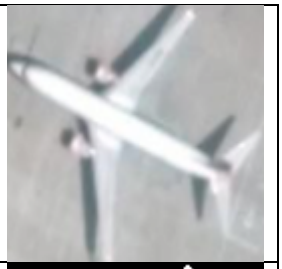
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 128, 128]	448
BatchNorm2d-2	[-1, 16, 128, 128]	32
ReLU-3	[-1, 16, 128, 128]	0
conv-4	[-1, 16, 128, 128]	0
Conv2d-5	[-1, 16, 128, 128]	2,320
BatchNorm2d-6	[-1, 16, 128, 128]	32
ReLU-7	[-1, 16, 128, 128]	0
Conv2d-8	[-1, 16, 128, 128]	2,320
BatchNorm2d-9	[-1, 16, 128, 128]	32
ReLU-10	[-1, 16, 128, 128]	0
block-11	[-1, 16, 128, 128]	0
Conv2d-12	[-1, 32, 128, 128]	4,640
BatchNorm2d-13	[-1, 32, 128, 128]	64
ReLU-14	[-1, 32, 128, 128]	0
conv-15	[-1, 32, 128, 128]	0
MaxPool2d-16	[-1, 32, 64, 64]	0
down-17	[-1, 32, 64, 64]	0
Conv2d-18	[-1, 32, 64, 64]	9,248
BatchNorm2d-19	[-1, 32, 64, 64]	64
ReLU-20	[-1, 32, 64, 64]	0
Conv2d-21	[-1, 32, 64, 64]	9,248
BatchNorm2d-22	[-1, 32, 64, 64]	64
ReLU-23	[-1, 32, 64, 64]	0
block-24	[-1, 32, 64, 64]	0
Conv2d-25	[-1, 64, 64, 64]	18,496
BatchNorm2d-26	[-1, 64, 64, 64]	128
ReLU-27	[-1, 64, 64, 64]	0
conv-28	[-1, 64, 64, 64]	0
MaxPool2d-29	[-1, 64, 32, 32]	0
down-30	[-1, 64, 32, 32]	0
Conv2d-31	[-1, 64, 32, 32]	36,928
BatchNorm2d-32	[-1, 64, 32, 32]	128
ReLU-33	[-1, 64, 32, 32]	0
Conv2d-34	[-1, 64, 32, 32]	36,928
BatchNorm2d-35	[-1, 64, 32, 32]	128
ReLU-36	[-1, 64, 32, 32]	0
block-37	[-1, 64, 32, 32]	0
Conv2d-38	[-1, 128, 32, 32]	73,856
BatchNorm2d-39	[-1, 128, 32, 32]	256
ReLU-40	[-1, 128, 32, 32]	0
conv-41	[-1, 128, 32, 32]	0
MaxPool2d-42	[-1, 128, 16, 16]	0
down-43	[-1, 128, 16, 16]	0
Conv2d-44	[-1, 128, 16, 16]	147,584
BatchNorm2d-45	[-1, 128, 16, 16]	256
ReLU-46	[-1, 128, 16, 16]	0
Conv2d-47	[-1, 128, 16, 16]	147,584
BatchNorm2d-48	[-1, 128, 16, 16]	256
ReLU-49	[-1, 128, 16, 16]	0
block-50	[-1, 128, 16, 16]	0
Conv2d-51	[-1, 256, 16, 16]	295,168
BatchNorm2d-52	[-1, 256, 16, 16]	512
ReLU-53	[-1, 256, 16, 16]	0
conv-54	[-1, 256, 16, 16]	0
MaxPool2d-55	[-1, 256, 8, 8]	0
down-56	[-1, 256, 8, 8]	0
Conv2d-57	[-1, 256, 8, 8]	590,080
BatchNorm2d-58	[-1, 256, 8, 8]	512
ReLU-59	[-1, 256, 8, 8]	0
Conv2d-60	[-1, 256, 8, 8]	590,080
BatchNorm2d-61	[-1, 256, 8, 8]	512
ReLU-62	[-1, 256, 8, 8]	0
block-63	[-1, 256, 8, 8]	0
Conv2d-64	[-1, 512, 8, 8]	1,180,160
BatchNorm2d-65	[-1, 512, 8, 8]	1,024
ReLU-66	[-1, 512, 8, 8]	0
conv-67	[-1, 512, 8, 8]	0
MaxPool2d-68	[-1, 512, 4, 4]	0
down-69	[-1, 512, 4, 4]	0
Conv2d-70	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-71	[-1, 512, 4, 4]	1,024
ReLU-72	[-1, 512, 4, 4]	0
Conv2d-73	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-74	[-1, 512, 4, 4]	1,024
ReLU-75	[-1, 512, 4, 4]	0
block-76	[-1, 512, 4, 4]	0
Conv2d-77	[-1, 1024, 4, 4]	4,719,616
BatchNorm2d-78	[-1, 1024, 4, 4]	2,048
ReLU-79	[-1, 1024, 4, 4]	0
conv-80	[-1, 1024, 4, 4]	0
MaxPool2d-81	[-1, 1024, 2, 2]	0
down-82	[-1, 1024, 2, 2]	0
ConvTranspose2d-83	[-1, 1024, 4, 4]	4,195,328
Conv2d-84	[-1, 512, 4, 4]	4,719,104
BatchNorm2d-85	[-1, 512, 4, 4]	1,024
ReLU-86	[-1, 512, 4, 4]	0
conv-87	[-1, 512, 4, 4]	0
up-88	[-1, 512, 4, 4]	0
BatchNorm2d-89	[-1, 512, 4, 4]	1,024
ConvTranspose2d-90	[-1, 512, 8, 8]	1,049,088
Conv2d-91	[-1, 256, 8, 8]	1,179,904
BatchNorm2d-92	[-1, 256, 8, 8]	512
ReLU-93	[-1, 256, 8, 8]	0
conv-94	[-1, 256, 8, 8]	0
up-95	[-1, 256, 8, 8]	0
BatchNorm2d-96	[-1, 256, 8, 8]	512
ConvTranspose2d-97	[-1, 256, 16, 16]	262,400
Conv2d-98	[-1, 128, 16, 16]	295,040
BatchNorm2d-99	[-1, 128, 16, 16]	256
ReLU-100	[-1, 128, 16, 16]	0
conv-101	[-1, 128, 16, 16]	0
up-102	[-1, 128, 16, 16]	0
BatchNorm2d-103	[-1, 128, 16, 16]	256
ConvTranspose2d-104	[-1, 128, 32, 32]	65,664
Conv2d-105	[-1, 64, 32, 32]	73,792
BatchNorm2d-106	[-1, 64, 32, 32]	128
ReLU-107	[-1, 64, 32, 32]	0
conv-108	[-1, 64, 32, 32]	0
up-109	[-1, 64, 32, 32]	0
BatchNorm2d-110	[-1, 64, 32, 32]	128
ConvTranspose2d-111	[-1, 64, 64, 64]	16,448
Conv2d-112	[-1, 32, 64, 64]	18,464
BatchNorm2d-113	[-1, 32, 64, 64]	64
ReLU-114	[-1, 32, 64, 64]	0
conv-115	[-1, 32, 64, 64]	0
up-116	[-1, 32, 64, 64]	0
BatchNorm2d-117	[-1, 32, 64, 64]	64
ConvTranspose2d-118	[-1, 32, 128, 128]	4,128
Conv2d-119	[-1, 16, 128, 128]	4,624
BatchNorm2d-120	[-1, 16, 128, 128]	32
ReLU-121	[-1, 16, 128, 128]	0
conv-122	[-1, 16, 128, 128]	0
up-123	[-1, 16, 128, 128]	0
BatchNorm2d-124	[-1, 16, 128, 128]	32
Conv2d-125	[-1, 3, 128, 128]	435
BatchNorm2d-126	[-1, 3, 128, 128]	6
ReLU-127	[-1, 3, 128, 128]	0
conv-128	[-1, 3, 128, 128]	0
BatchNorm2d-129	[-1, 3, 128, 128]	6
Conv2d-130	[-1, 1, 128, 128]	28
conv-131	[-1, 1, 128, 128]	0

3) I kept the loss function same as baseline using `BCEWithLogitsLoss`. Plotting epochs vs total training loss.

4) The final IoU for the model is:

```
#images: 7980, Mean IoU: 0.7702353964455629
```

5) Testing Samples:

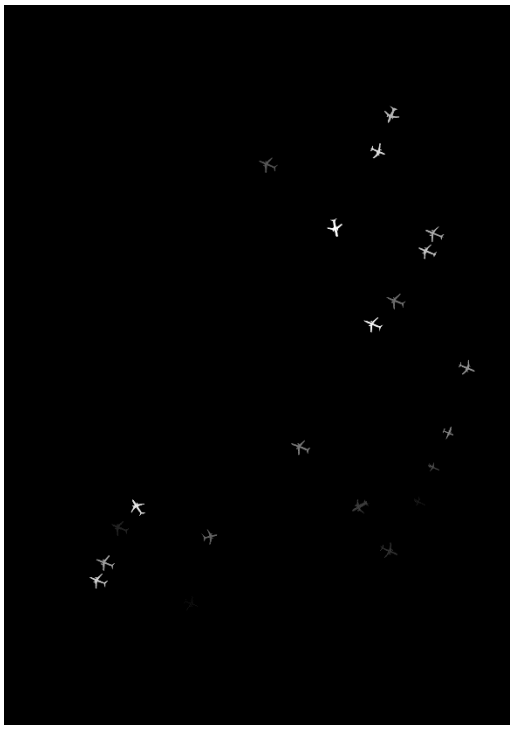
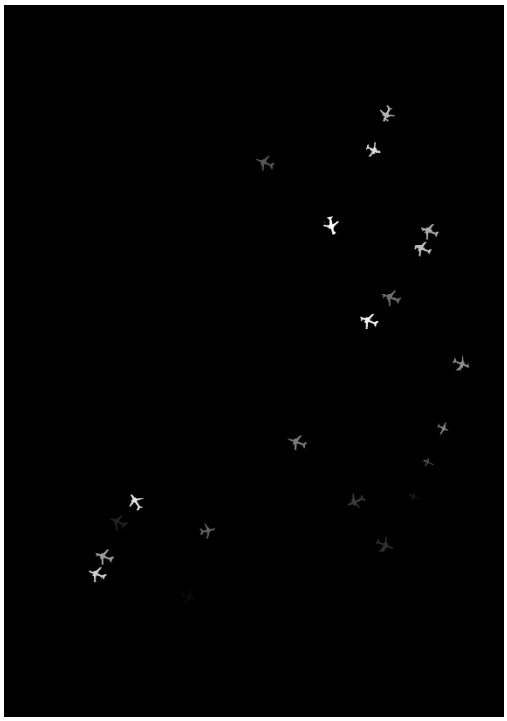
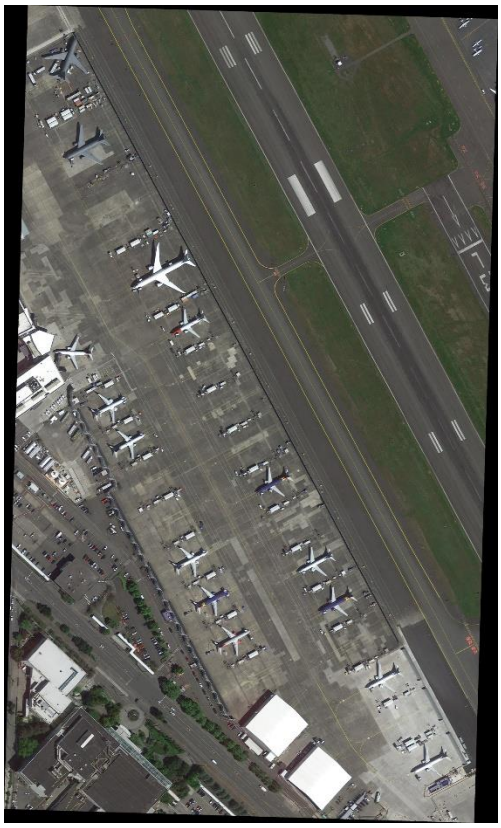
Image			
Mask			
Prediction			

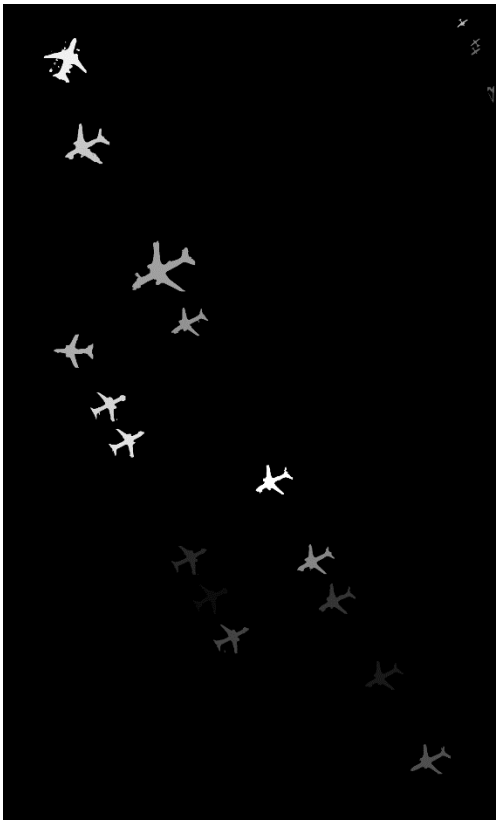
Part 3:

1) Kaggle teamname: 301386847
2) Score on Kaggle leaderboard: 0.46282
3) Test Samples

Vaibhav Saini
301386847

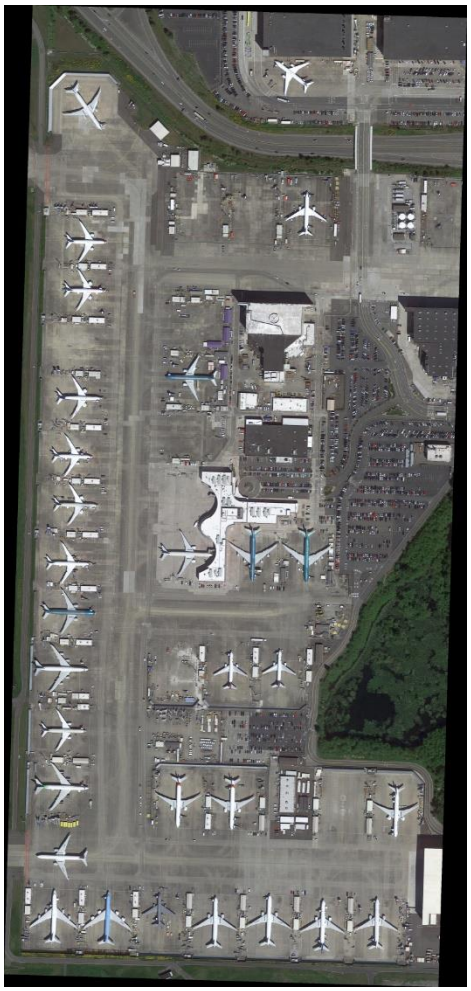
Late days used: 2
Team Name: 301386847

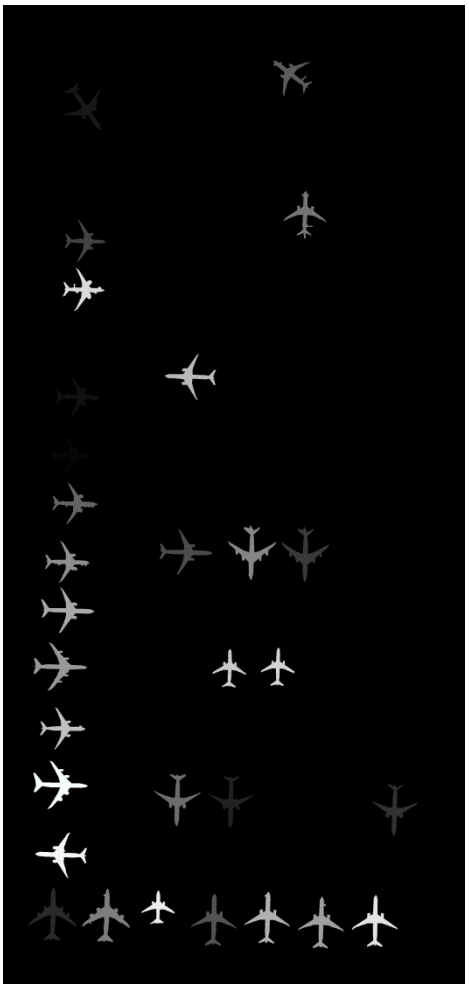





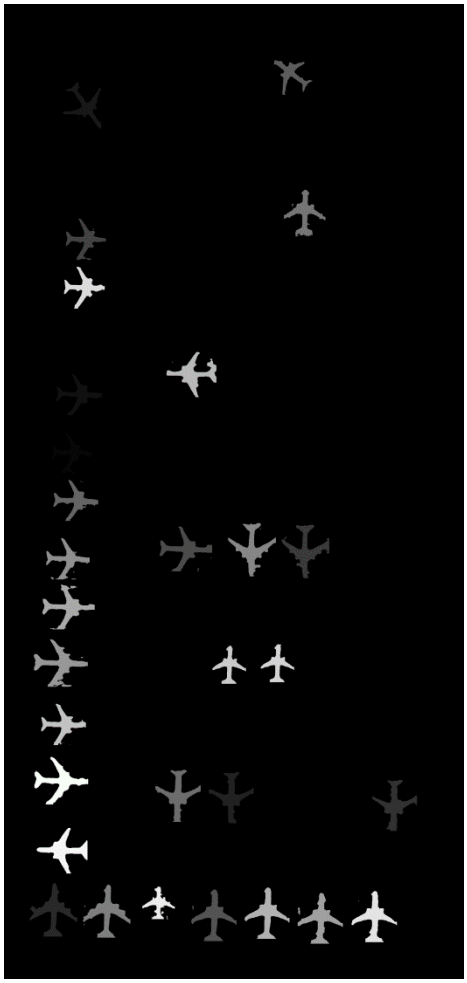


Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847







Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847

Part 4:

1) Visualizations

Segs evaluation

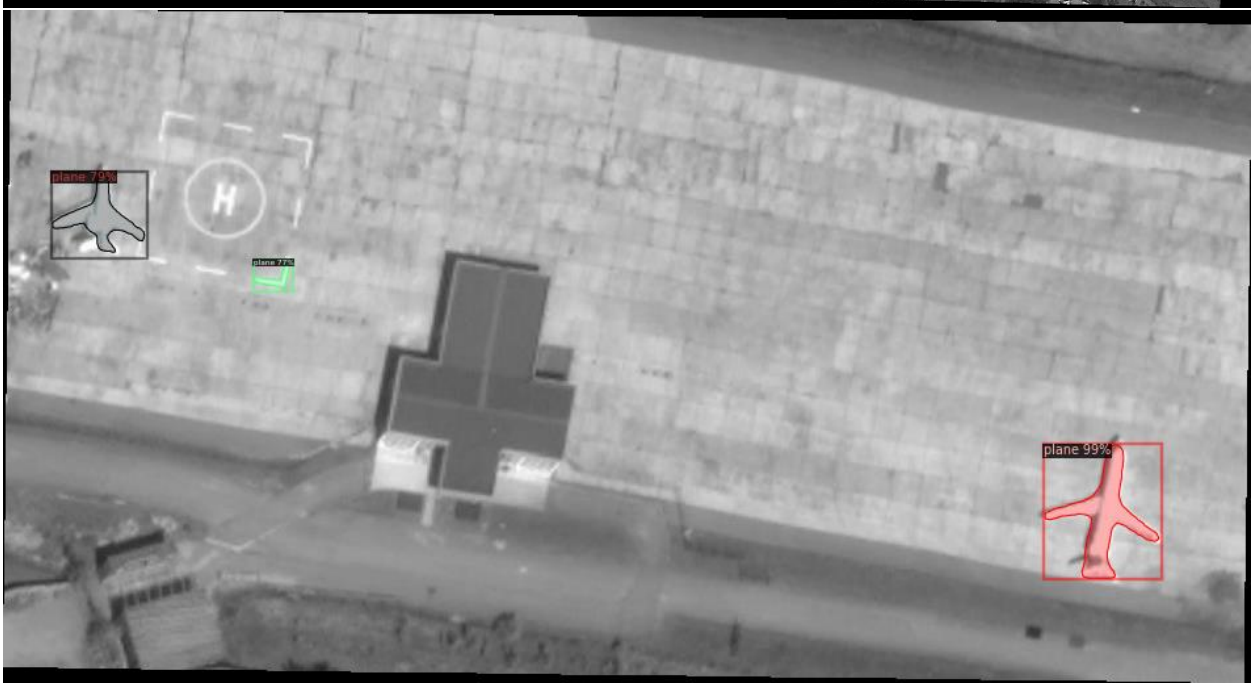
```
[11/05 10:21:35 d2.evaluation.fast_eval_api]: Evaluate annotation type *segm*
[11/05 10:21:36 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.89 seconds.
[11/05 10:21:36 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/05 10:21:36 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.077
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.272
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.012
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.036
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.087
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.317
 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 ] = 0.052
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.106
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.050
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.129
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.374
[11/05 10:21:36 d2.evaluation.coco_evaluation]: Evaluation results for segm:
|  AP   |  AP50  |  AP75  |  APs  |  APm  |  APl   |
|:-----:|:------:|:------:|:-----:|:-----:|:------:|
| 7.669 | 27.247 | 1.248  | 3.618 | 8.737 | 31.681 |
```

Bbox evaluation

```
[11/05 10:21:34 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[11/05 10:21:35 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.23 seconds.
[11/05 10:21:35 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/05 10:21:35 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 ] = 0.278
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.483
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.290
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.173
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.360
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.581
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.015
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.123
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.313
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.180
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.398
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.711
[11/05 10:21:35 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 27.781 | 48.258 | 28.988 | 17.347 | 35.970 | 58.070 |
```

Vaibhav Saini
301386847

Late days used: 2
Team Name: 301386847



2) Comparing this mask RCNN model with Part3 model, we can see that our part3 model was actually able to perform better than mask RCNN model. With the masks making fewer mistakes and misclassifications.

References:

My experiment with UNet - building an image segmentation model (analyticsindiamag.com)

Gentle Introduction to the Adam Optimization Algorithm for Deep Learning (machinelearningmastery.com)

python - What does "unsqueeze" do in Pytorch? - Stack Overflow

Detectron2 Train a Instance Segmentation Model (gilberttanner.com)

Quick intro to Instance segmentation: Mask R-CNN (kharshit.github.io)

Use Custom Datasets — detectron2 0.6 documentation

detectron2/MODEL_ZOO.md at main · facebookresearch/detectron2 · GitHub

Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS (lilianweng.github.io)

Understanding Semantic Segmentation with UNET | by Harshall Lamba | Towards Data Science

Metrics to Evaluate your Semantic Segmentation Model | by Ekin Tiu | Towards Data Science

python - Google Colab is very slow compared to my PC - Stack Overflow