CMPT 412-COMPUTER VISION

PROJECT-3 REPORT (NAVJOT KAUR: 301404765)

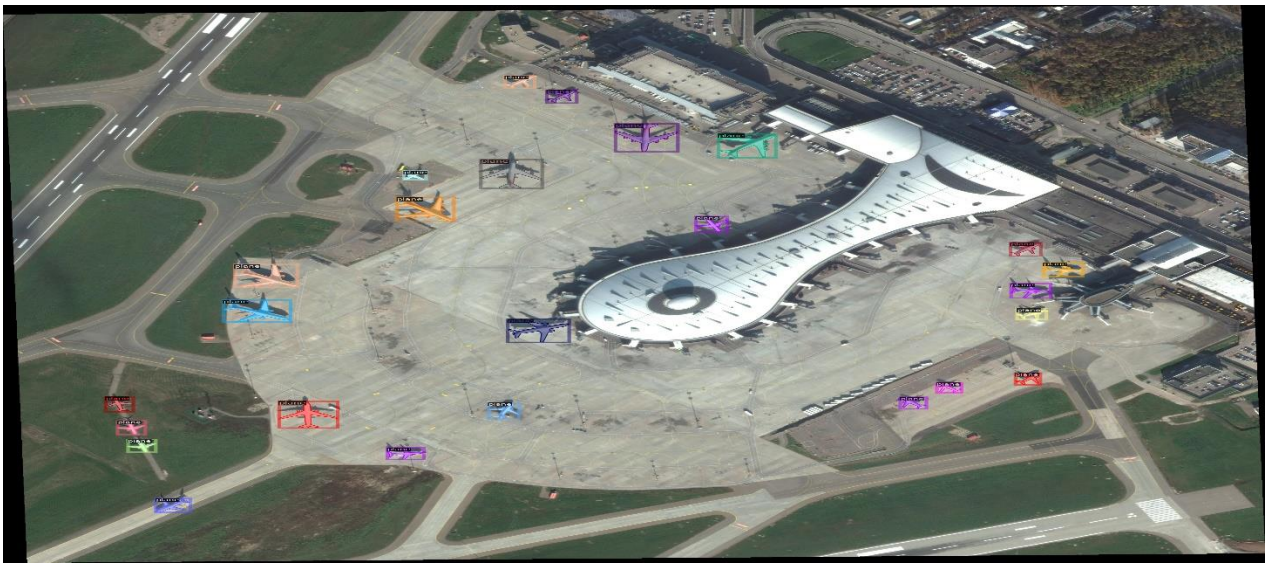
Kaggle username: Navjott77

I am claiming my 3 late days for this assignment

PART 1: OBJECT DETECTION

In this part of the project the train dataset is loaded from train.json and the steps for loading test dataset by keeping the annotations empty were followed as listed in the project description.

Images from train dataset to make sure the function works correctly:



1. **List of configs and modifications that I used:**

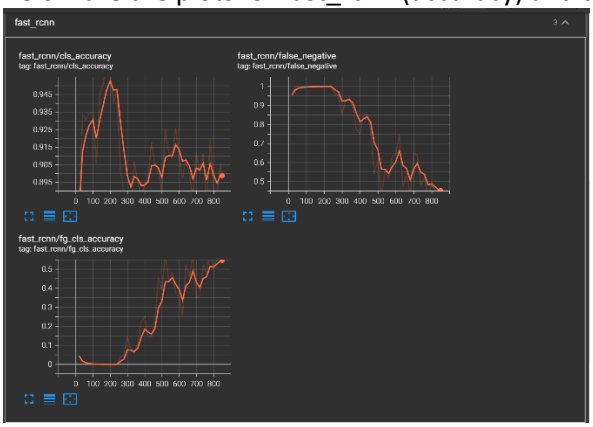
```
cfg = get_cfg()
cfg.OUTPUT_DIR = "{}/output/".format(BASE_DIR)
cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_101_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("plane_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 850
cfg.SOLVER.STEPS = []
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
```

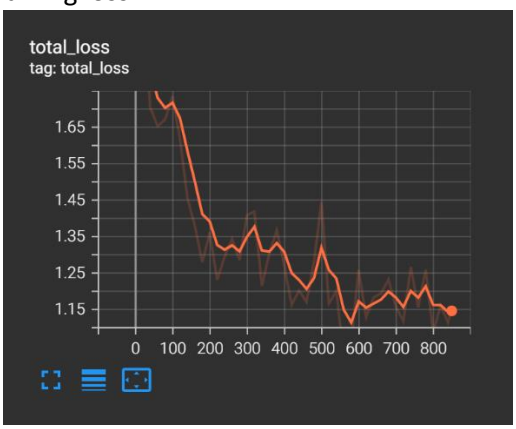
2. **Factors which helped improve the performance. Explain each factor in 2-3 lines.**
   The provided configurations in the description were tested first where only the max iterations is changed from 500 to 850 to minimize the total_loss. At the end the loss became as low as 1.02 starting from approximate 1.98, which was quite good.
   Base Learning Rate was kept as 0.00025 after using smaller and larger rates but the results responded better when the rate was 0.00025. When the LR was larger the accuracy was much low and when it was smaller then it started off with a good accuracy but remained constant over time.
   Other factors such as num_workers, ims_per_batch, batch_size_per_image and num_classes were kept constant because they did not have much effect on the model.

```
[d2.utils.events]:  eta: 0:13:51  iter: 19  total_loss: 1.989  loss_cls: 0.4796  loss_box_reg: 0.04301  loss_rpn_cls: 0.6727  loss_rpn_loc: 0.7133  time: 1.143
[d2.utils.events]:  eta: 0:13:32  iter: 39  total_loss: 1.704  loss_cls: 0.3004  loss_box_reg: 0.02583  loss_rpn_cls: 0.6791  loss_rpn_loc: 0.7375  time: 1.209
[d2.utils.events]:  eta: 0:11:37  iter: 59  total_loss: 1.653  loss_cls: 0.2651  loss_box_reg: 0.05806  loss_rpn_cls: 0.669  loss_rpn_loc: 0.6594  time: 1.1652
[d2.utils.events]:  eta: 0:11:47  iter: 79  total_loss: 1.671  loss_cls: 0.2472  loss_box_reg: 0.02023  loss_rpn_cls: 0.6685  loss_rpn_loc: 0.6038  time: 1.200
[d2.utils.events]:  eta: 0:11:50  iter: 99  total_loss: 1.735  loss_cls: 0.233  loss_box_reg: 0.02978  loss_rpn_cls: 0.6561  loss_rpn_loc: 0.6485  time: 1.2071
[d2.utils.events]:  eta: 0:11:48  iter: 119  total_loss: 1.618  loss_cls: 0.2931  loss_box_reg: 0.04188  loss_rpn_cls: 0.6158  loss_rpn_loc: 0.6087  time: 1.22
[d2.utils.events]:  eta: 0:11:50  iter: 139  total_loss: 1.456  loss_cls: 0.2178  loss_box_reg: 0.02754  loss_rpn_cls: 0.6193  loss_rpn_loc: 0.5675  time: 1.21
[d2.utils.events]:  eta: 0:11:10  iter: 159  total_loss: 1.379  loss_cls: 0.1718  loss_box_reg: 0.03791  loss_rpn_cls: 0.5909  loss_rpn_loc: 0.5754  time: 1.19
[d2.utils.events]:  eta: 0:11:02  iter: 179  total_loss: 1.28  loss_cls: 0.1642  loss_box_reg: 0.03669  loss_rpn_cls: 0.5875  loss_rpn_loc: 0.4436  time: 1.188
[d2.utils.events]:  eta: 0:10:31  iter: 199  total_loss: 1.361  loss_cls: 0.1576  loss_box_reg: 0.01372  loss_rpn_cls: 0.5567  loss_rpn_loc: 0.5021  time: 1.19
[d2.utils.events]:  eta: 0:10:33  iter: 219  total_loss: 1.231  loss_cls: 0.1802  loss_box_reg: 0.02521  loss_rpn_cls: 0.526  loss_rpn_loc: 0.4713  time: 1.209
[d2.utils.events]:  eta: 0:10:02  iter: 239  total_loss: 1.295  loss_cls: 0.1912  loss_box_reg: 0.0622  loss_rpn_cls: 0.5258  loss_rpn_loc: 0.3793  time: 1.204
[d2.utils.events]:  eta: 0:09:26  iter: 259  total_loss: 1.344  loss_cls: 0.2541  loss_box_reg: 0.1457  loss_rpn_cls: 0.5246  loss_rpn_loc: 0.3383  time: 1.195
[d2.utils.events]:  eta: 0:09:23  iter: 279  total_loss: 1.285  loss_cls: 0.26  loss_box_reg: 0.1851  loss_rpn_cls: 0.4763  loss_rpn_loc: 0.3601  time: 1.1880
[d2.utils.events]:  eta: 0:09:16  iter: 299  total_loss: 1.409  loss_cls: 0.2849  loss_box_reg: 0.2663  loss_rpn_cls: 0.47  loss_rpn_loc: 0.4357  time: 1.1999
[d2.utils.events]:  eta: 0:08:53  iter: 319  total_loss: 1.419  loss_cls: 0.2706  loss_box_reg: 0.2499  loss_rpn_cls: 0.4533  loss_rpn_loc: 0.4148  time: 1.193
[d2.utils.events]:  eta: 0:08:36  iter: 339  total_loss: 1.214  loss_cls: 0.207  loss_box_reg: 0.1247  loss_rpn_cls: 0.4362  loss_rpn_loc: 0.3025  time: 1.1962
[d2.utils.events]:  eta: 0:08:11  iter: 359  total_loss: 1.304  loss_cls: 0.2583  loss_box_reg: 0.2649  loss_rpn_cls: 0.4322  loss_rpn_loc: 0.3127  time: 1.188
[d2.utils.events]:  eta: 0:07:51  iter: 379  total_loss: 1.367  loss_cls: 0.2458  loss_box_reg: 0.2758  loss_rpn_cls: 0.3953  loss_rpn_loc: 0.3102  time: 1.1919
[d2.utils.events]:  eta: 0:07:32  iter: 399  total_loss: 1.272  loss_cls: 0.2721  loss_box_reg: 0.2384  loss_rpn_cls: 0.4059  loss_rpn_loc: 0.3034  time: 1.1944
[d2.utils.events]:  eta: 0:07:13  iter: 419  total_loss: 1.163  loss_cls: 0.244  loss_box_reg: 0.2479  loss_rpn_cls: 0.3817  loss_rpn_loc: 0.3307  time: 1.1966
[d2.utils.events]:  eta: 0:06:56  iter: 439  total_loss: 1.202  loss_cls: 0.2128  loss_box_reg: 0.2024  loss_rpn_cls: 0.3551  loss_rpn_loc: 0.3553  time: 1.1959
[d2.utils.events]:  eta: 0:06:34  iter: 459  total_loss: 1.17  loss_cls: 0.2401  loss_box_reg: 0.2799  loss_rpn_cls: 0.3294  loss_rpn_loc: 0.3639  time: 1.1958
[d2.utils.events]:  eta: 0:06:12  iter: 479  total_loss: 1.28  loss_cls: 0.2501  loss_box_reg: 0.2888  loss_rpn_cls: 0.3228  loss_rpn_loc: 0.2843  time: 1.1927
[d2.utils.events]:  eta: 0:05:52  iter: 499  total_loss: 1.444  loss_cls: 0.2733  loss_box_reg: 0.3554  loss_rpn_cls: 0.3615  loss_rpn_loc: 0.3654  time: 1.2001
[d2.utils.events]:  eta: 0:05:31  iter: 519  total_loss: 1.166  loss_cls: 0.1991  loss_box_reg: 0.3024  loss_rpn_cls: 0.3478  loss_rpn_loc: 0.343  time: 1.1980
[d2.utils.events]:  eta: 0:05:12  iter: 539  total_loss: 1.201  loss_cls: 0.2205  loss_box_reg: 0.4125  loss_rpn_cls: 0.2776  loss_rpn_loc: 0.3082  time: 1.1965
[d2.utils.events]:  eta: 0:04:51  iter: 559  total_loss: 1.02  loss_cls: 0.2177  loss_box_reg: 0.3416  loss_rpn_cls: 0.2504  loss_rpn_loc: 0.2154  time: 1.2015
[d2.utils.events]:  eta: 0:04:31  iter: 579  total_loss: 1.062  loss_cls: 0.1984  loss_box_reg: 0.3109  loss_rpn_cls: 0.266  loss_rpn_loc: 0.326  time: 1.1959
[d2.utils.events]:  eta: 0:04:11  iter: 599  total_loss: 1.259  loss_cls: 0.2247  loss_box_reg: 0.319  loss_rpn_cls: 0.2865  loss_rpn_loc: 0.3539  time: 1.1989
[d2.utils.events]:  eta: 0:03:52  iter: 619  total_loss: 1.129  loss_cls: 0.2592  loss_box_reg: 0.2354  loss_rpn_cls: 0.2446  loss_rpn_loc: 0.2778  time: 1.1974
[d2.utils.events]:  eta: 0:03:32  iter: 639  total_loss: 1.183  loss_cls: 0.2188  loss_box_reg: 0.3855  loss_rpn_cls: 0.2364  loss_rpn_loc: 0.286  time: 1.2014
[d2.utils.events]:  eta: 0:03:13  iter: 659  total_loss: 1.194  loss_cls: 0.2214  loss_box_reg: 0.3656  loss_rpn_cls: 0.231  loss_rpn_loc: 0.2967  time: 1.1988
[d2.utils.events]:  eta: 0:02:52  iter: 679  total_loss: 1.232  loss_cls: 0.2482  loss_box_reg: 0.4428  loss_rpn_cls: 0.1991  loss_rpn_loc: 0.2291  time: 1.1938
[d2.utils.events]:  eta: 0:02:32  iter: 699  total_loss: 1.158  loss_cls: 0.21  loss_box_reg: 0.3829  loss_rpn_cls: 0.2342  loss_rpn_loc: 0.2702  time: 1.1948
[d2.utils.events]:  eta: 0:02:12  iter: 719  total_loss: 1.118  loss_cls: 0.218  loss_box_reg: 0.3564  loss_rpn_cls: 0.2321  loss_rpn_loc: 0.2577  time: 1.1948
[d2.utils.events]:  eta: 0:01:51  iter: 739  total_loss: 1.266  loss_cls: 0.2343  loss_box_reg: 0.4074  loss_rpn_cls: 0.1711  loss_rpn_loc: 0.2232  time: 1.1956
[d2.utils.events]:  eta: 0:01:31  iter: 759  total_loss: 1.156  loss_cls: 0.2881  loss_box_reg: 0.2838  loss_rpn_cls: 0.246  loss_rpn_loc: 0.2995  time: 1.1985
[d2.utils.events]:  eta: 0:01:10  iter: 779  total_loss: 1.26  loss_cls: 0.2055  loss_box_reg: 0.4206  loss_rpn_cls: 0.2209  loss_rpn_loc: 0.2877  time: 1.1968
[d2.utils.events]:  eta: 0:00:50  iter: 799  total_loss: 1.086  loss_cls: 0.2485  loss_box_reg: 0.428  loss_rpn_cls: 0.1921  loss_rpn_loc: 0.2428  time: 1.1997
[d2.utils.events]:  eta: 0:00:30  iter: 819  total_loss: 1.161  loss_cls: 0.2586  loss_box_reg: 0.4602  loss_rpn_cls: 0.1788  loss_rpn_loc: 0.2605  time: 1.196
[d2.utils.events]:  eta: 0:00:10  iter: 839  total_loss: 1.114  loss_cls: 0.2279  loss_box_reg: 0.3929  loss_rpn_cls: 0.1791  loss_rpn_loc: 0.2728  time: 1.195
[d2.utils.events]:  eta: 0:00:00  iter: 849  total_loss: 1.151  loss_cls: 0.2334  loss_box_reg: 0.3865  loss_rpn_cls: 0.1961  loss_rpn_loc: 0.2742  time: 1.194
[d2.engine.hooks]: Overall training speed: 848 iterations in 0:16:53 (1.1948 s / it)
[d2.engine.hooks]: Total training time: 0:16:55 (0:00:02 on hooks)
```

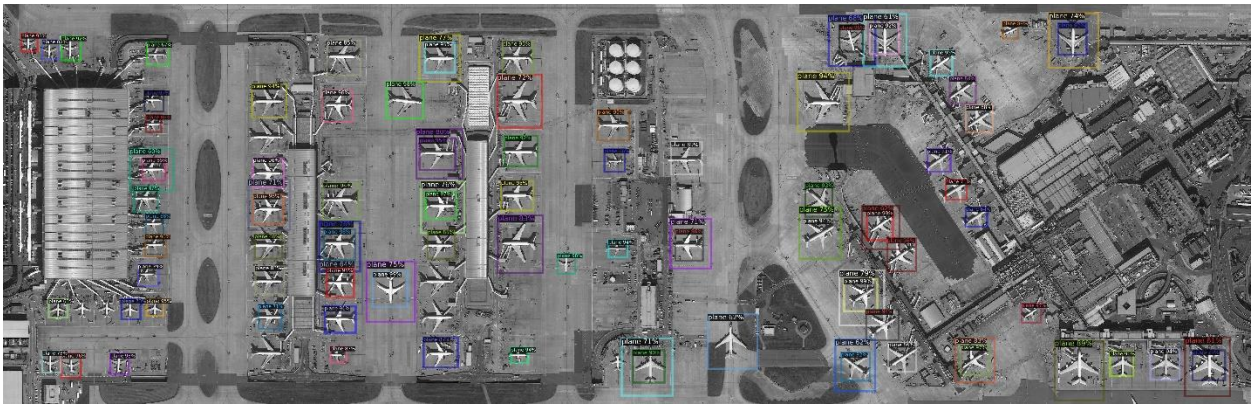
3. **Final plot for total training loss and accuracy**

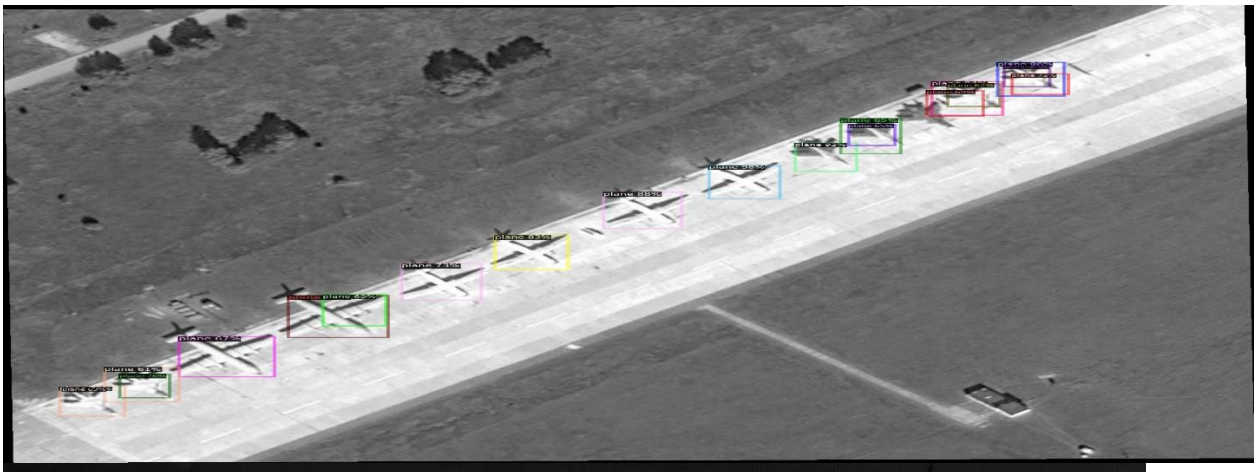
Below are the plots for fast_rcnn (accuracy) and training loss:





4. **Visualization of 3 samples from the test set and the predicted results**



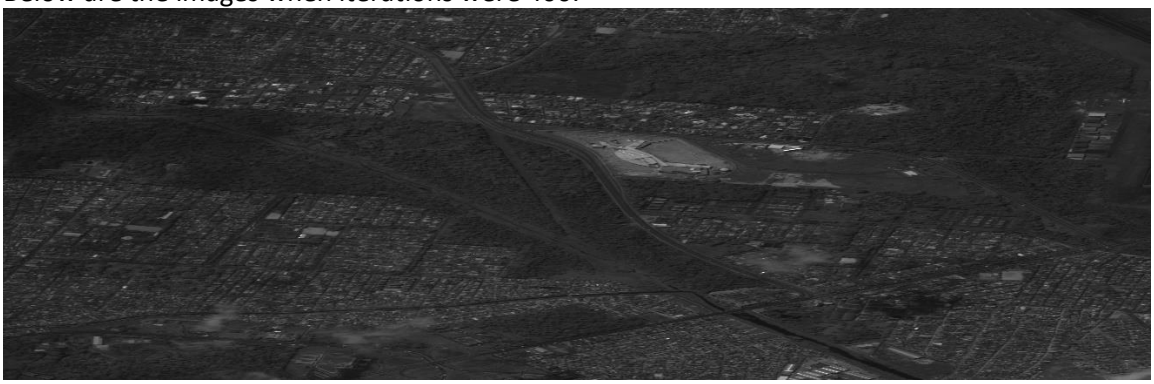


5. Ablation study

The improvements made to learning rate and max_iter were found to improve the performance of the model. Below are the images that show the difference.

Above 3 images are when iterations were 850.

Below are the images when iterations were 400:



PART 2: SEMANTIC SEGMENTATION

1. **Report any hyperparameter settings you used**

```
num_epochs = 50
batch_size = 4
learning_rate = 0.001
weight_decay = 1e-5
optimizer = SGD
```

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

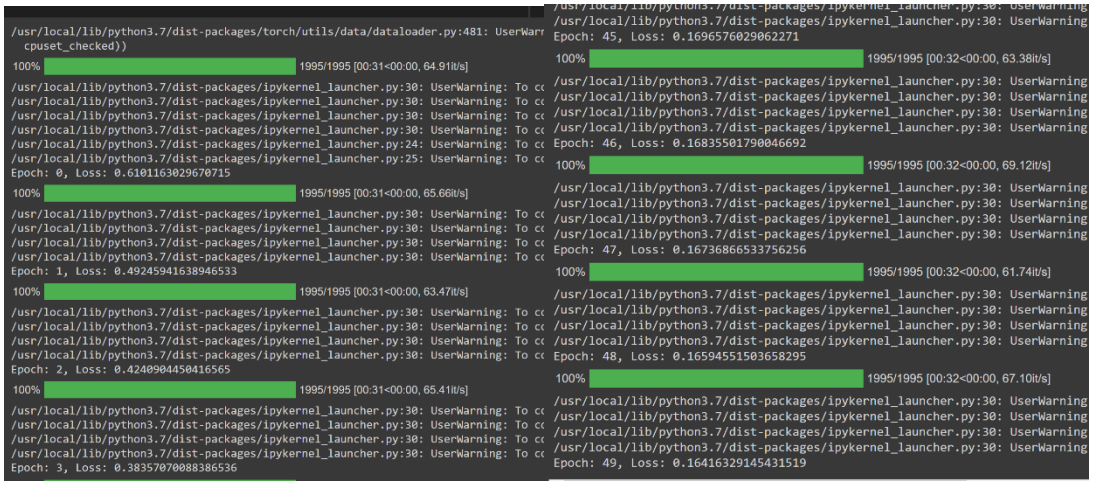
```
#Encoder

self.input_conv = conv(3, 4)
self.down1 = down(4, 16)
```

```
self.down2 = down(16, 32)
self.down3 = down(32, 64)
self.down4 = down(64, 128)
self.down5 = down(128, 256)
self.down6 = down(256, 512)

# Decoder

self.up1 = up(512, 256)
self.up2 = up(256, 128)
self.up3 = up(128, 64)
self.up4 = up(64, 32)
self.up5 = up(32, 16)
self.up6 = up(16, 4)

self.output_conv = conv(4, 1, False)
```

The above given architecture is used for the model. The reasoning for the modifications were that firstly increasing the encoder and decoder layers helped in better visualizations and predictions of the plane images. Second the loss reduced drastically from 0.6101 to 0.1641. By keeping the hyperparameters to 50 epochs, batch_size = 4, learning_rate as 0.001 and SGD optimizer (given), the feature detector was improved for better detection, visualizations and predictions.

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

   Loss functions used is nn.BCEWithLogistsLoss, which is already given in the code. NO changes were made here. The loss achieved is 0.1641.



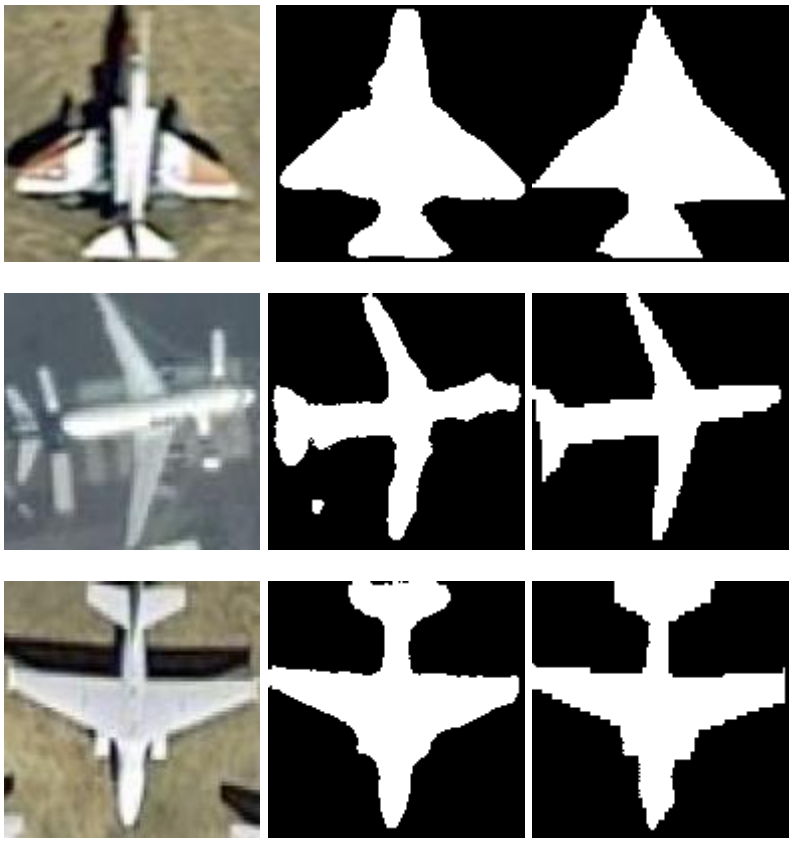
4. **Report the final mean IoU of your model**.
   IoU is calculated by first calculating the intersection and union as follows and then dividing them.

```
true_img = np.rint(np.array(nn.Sigmoid()(pred[i])[0]))

true_mask = np.rint(np.array(nn.Sigmoid()(mask[i])[0]))

intersection = np.sum(np.logical_and(true_mask, true_img))
union = np.sum(np.logical_or(true_mask, true_img))
iou = intersection/union

total_iou = total_iou + iou
```

Final IoU of the model is 0.80

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max
  cpuset_checked))
100% 998/998 [00:11<00:00, 87.02it/s]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:30: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detac
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:30: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detac
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:30: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detac
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:30: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detac

#images: 7980, Mean IoU: 0.8088094262411628
```

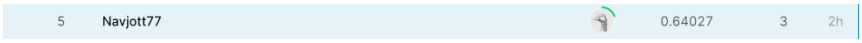
5. **Visualize 3 images from the test set and the corresponding predicted masks.**



PART 3: INSTANCE SEGMENTATION

1. **The name under which you submitted on Kaggle.** Navjott77

2. **Report the best score (should match your score on Kaggle**).
   0.64027



3. **The visualisation of results for 3 random samples from the test set.**

4. **CSV file has been uploaded to the Kaggle.**

PART 4: Mask R-CNN

1. **The visualisation and the evaluation results similar to Part 1.**

2. **Explain the differences between the results of Part 1 and Part 4 in a few lines**
   The model in Part 1 was better than the one used in Part 4, which is evident from the AP scores and the visualization difference as the planes detected were more in Part 1 than Part 4.

Part 1: bbox

AP	AP50	AP75	APs	APm	APl
14.768	37.314	7.014	12.389	21.734	16.720

Part 4: segm

AP	AP50	AP75	APs	APm	APl
3.979	15.033	1.173	1.774	4.441	25.165