Project 3

Object Detection, Semantic Segmentation, and Instance Segmentation

Name: Muhammad Shahzaib Waseem

Part 1: Object Detection

Configuration

The initial model configuration that I opted for is as mentioned in the handout:

- Pretrained Model used: “faster_rcnn_R_101_FPN_3x.yml”
- MAX_ITER = 500
- BATCH_SIZE_PER_IMAGE = 512
- IMS_PER_BATCH = 2
- BASE_LR = 0.00025
- SCORE_THRESH_TEST = 0.7

Improvements

The improvements that I made (which will be touched upon in the ablation study) are as follows:

- Pretrained Model used: “faster_rcnn_X_101_32x8d_FPN_3x.yml”
- MAX_ITER = 1500
- BASE_LR = 0.00025
- SCORE_THRESH_TEST = 0.6

Other than that, I employed certain augmentation transforms, which are as follows:

- Resize
- Random Crop
- Random Brightness
- Random Flip (Horizontal)
- Random Flip (Vertical)
- Random Rotation

In the following sections of the document, I divided input images into patches and then sent them to the detection model. I skipped the parts of the original image where there was no plane in the corresponding patch/block. As the number of images increased and the resolution decreased (by dividing the image into blocks), I increased the number of iterations and changed the batch size per image to 4 instead of 2.

Losses

Original Losses

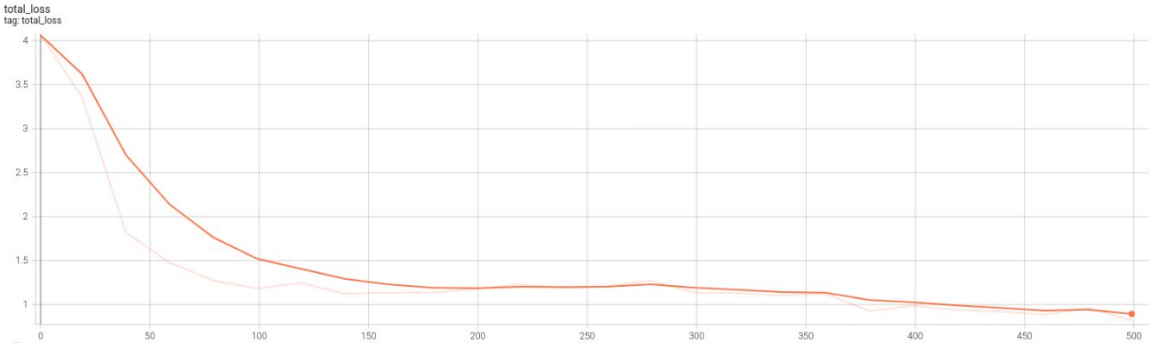

Figure 1: Training Loss Plot (baseline)

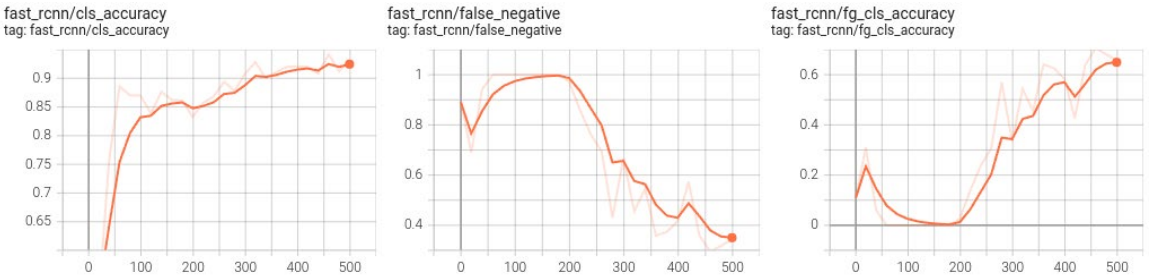

Figure 2: Classification Accuracy Plot (baseline)

After Improvements

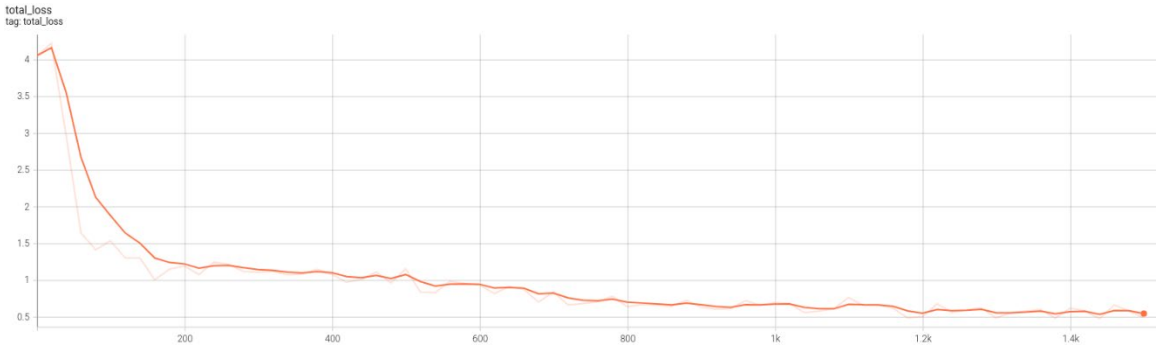

Figure 3: Training Loss Plot (after suggested improvements)

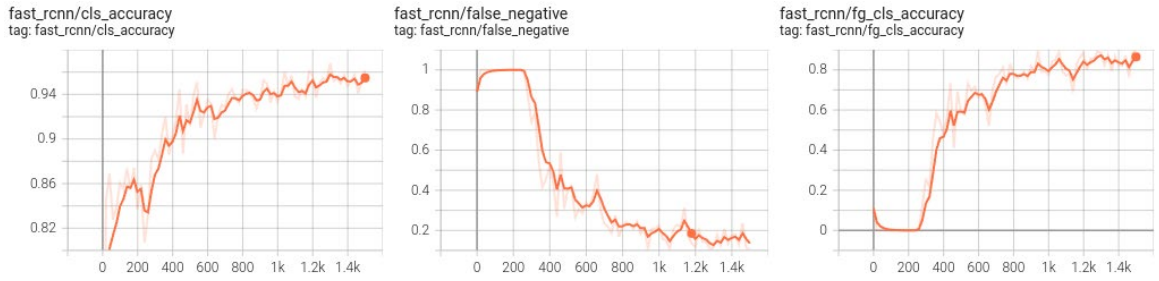

Figure 4: Classification Accuracy Plot (after suggested improvements)

Using Patches

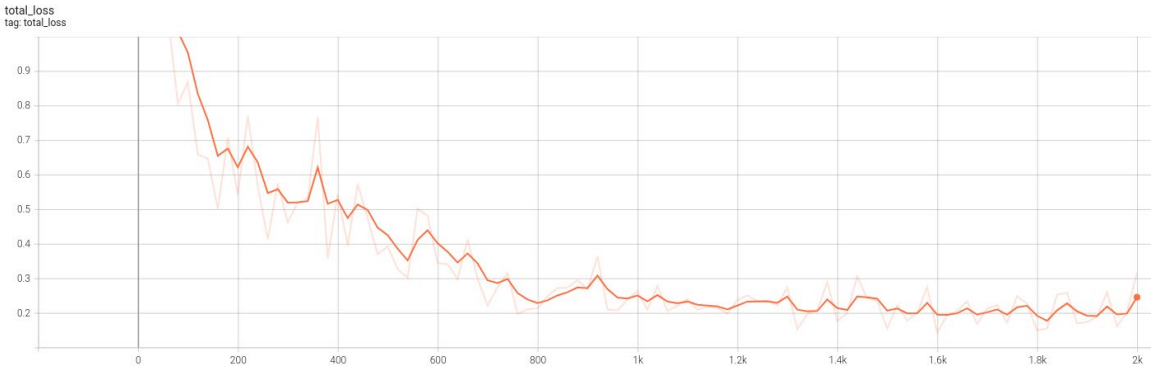

Figure 5: Training Loss Plot (patches data)

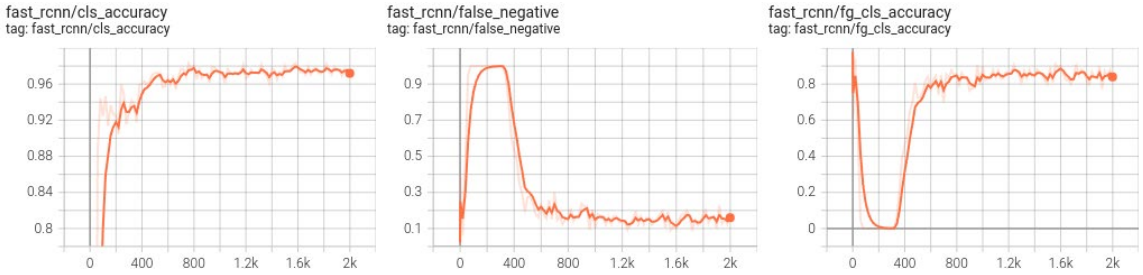

Figure 6: Classification Accuracy Plot (patches data)

Visualization

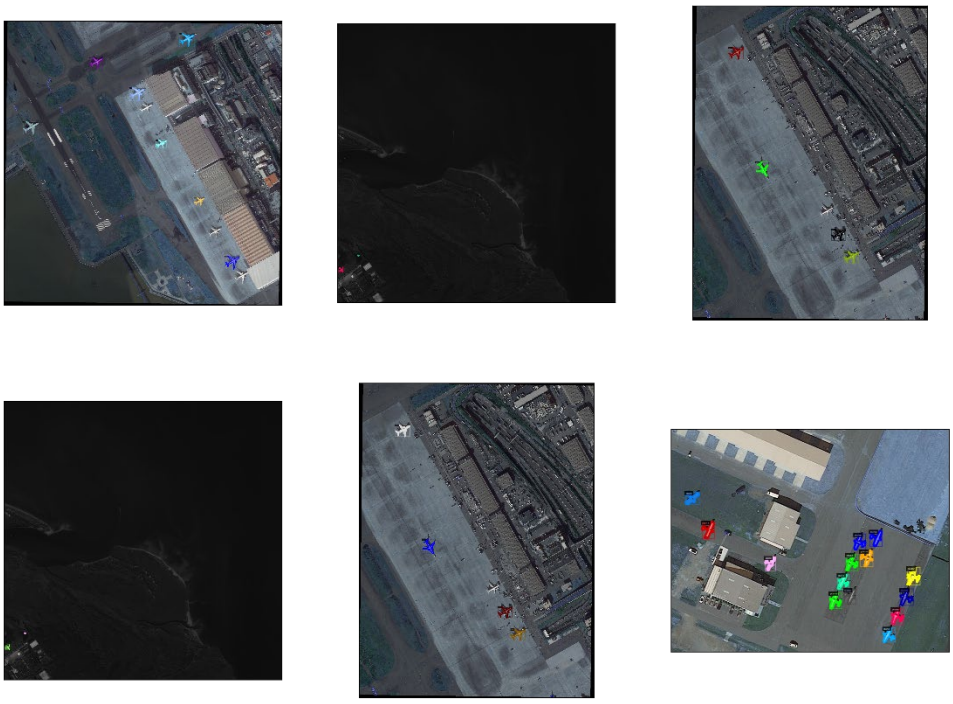

Figure 7: Samples from Training Set

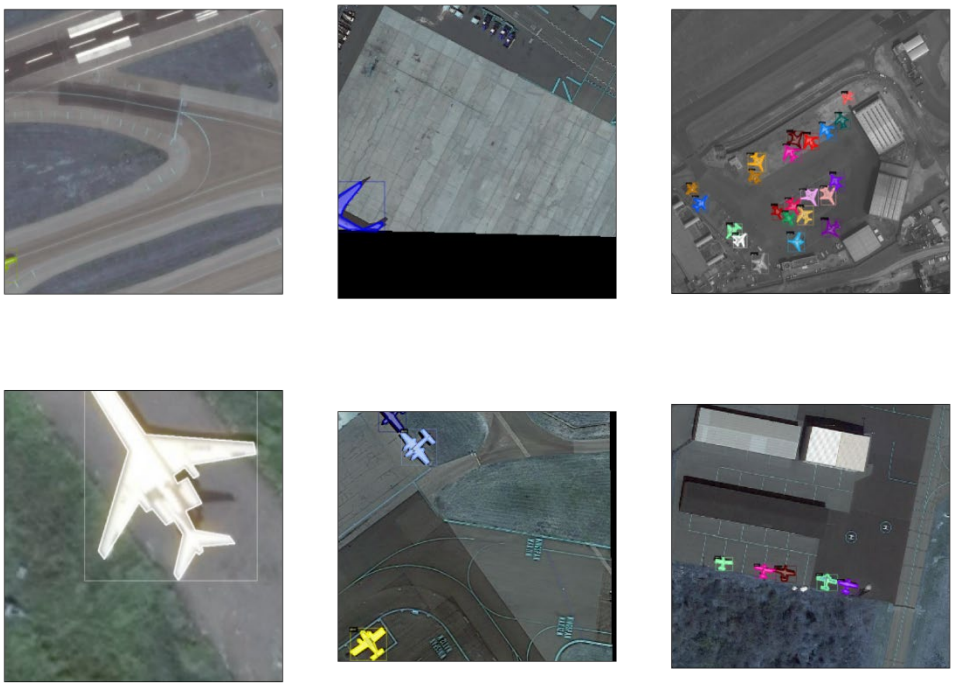

Figure 8: Samples from Training Set (patches data)

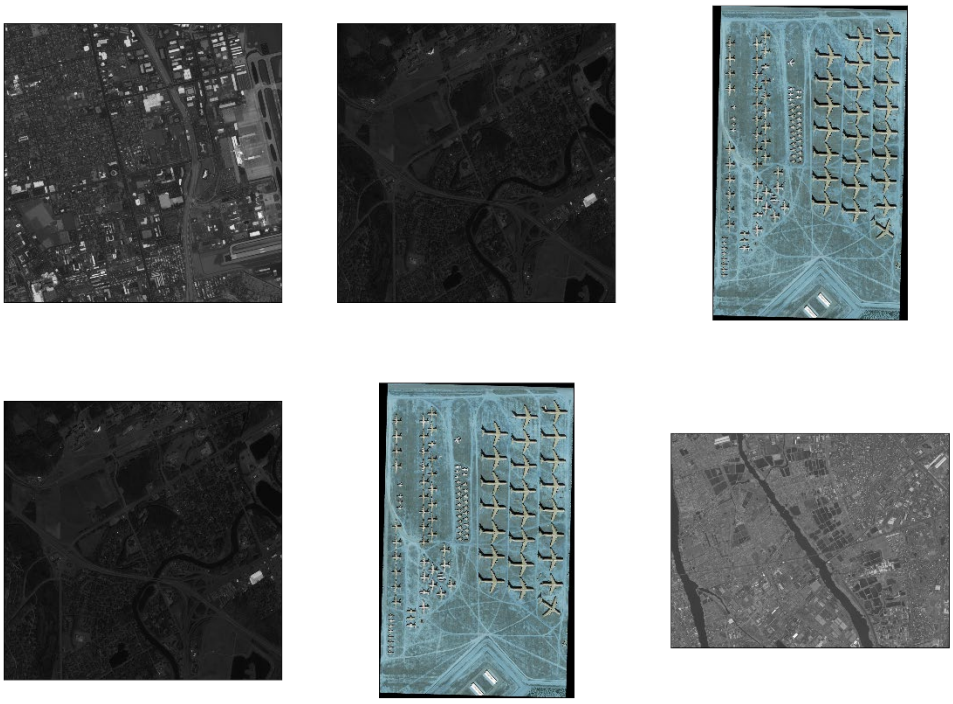

Figure 9: Samples from Test Set

Ablation Study

The model, loaded from faster_rcnn_R_101_FPN_3x.yml, with the configuration provided did not perform that well, it gave the total loss of **0.8193**, when run on 500 iterations. Whereas the improved model, loaded from faster_rcnn_X_101_32x8d_FPN_3x.yml, gave a much better accuracy and loss values, the total loss went as low as **0.4923**, when run on 1500 iterations. Other than that, the transformations that I employed as mentioned in the above sections made the model much more robust as the airplane, which is the only class in this task, appear far and close in multiple instances. This makes the model much more robust and does not allow the model to overfit that easily. The other thing I noticed was that the model would work well only if after employing the augmentation techniques the model would not perform that well if the number of iterations were not increased. Lastly, the new model faster_rcnn_X_101_32x8d_FPN_3x.yml over the previous model (even with augmentations and the updated configurations) gave a 10-15% increase in all the values in the Evaluation step.

After this I converted the images into small patches and then processed them using the detector and this shows that there is a huge performance improvement all round with increase of around 20 – 50% in the Average precision metrics. We can see a decrease in the APl score and not a huge gain in the APm score and that is because as the images were converted into small patches and no longer have medium or large sized objects (l in APl is for large and upon reading the documentation the exact classification of medium and large are: Medium $32^2 <$ area $< 96^2$ and Large: area $> 96^2$). As there are no medium and large objects the model shows a decrease in the Average Precision score for these two categories.

Loss Comparison

Table 1: Training Loss (baseline)

Iteration	Total Loss	Classification Loss (ROI)	Box Regression Loss (ROI)	Classification Loss (RPN)	Localization Loss (RPN)
499	0.8193	0.182	0.3049	0.1266	0.1876

Table 2: Validation Loss (baseline)

AP	AP50	AP75	APs	APm	APl
25.532	50.884	23.175	22.917	40.961	50.750

Table 3: Training Loss (after suggested improvements)

Iteration	Total Loss	Classification Loss (ROI)	Box Regression Loss (ROI)	Classification Loss (RPN)	Localization Loss (RPN)
1499	0.4923	0.09955	0.2605	0.01884	0.09092

Table 4: Validation Loss (after suggested improvements)

AP	AP50	AP75	APs	APm	APl
39.029	66.136	42.902	34.872	57.326	67.355

Table 5: Training Loss (patches data)

Iteration	Total Loss	Classification Loss (ROI)	Box Regression Loss (ROI)	Classification Loss (RPN)	Localization Loss (RPN)
4999	0.2171	0.07587	0.1101	0.005773	0.0167

Table 6: Validation Loss (patches data)

AP	AP50	AP75	APs	APm	APl
61.261	86.990	72.992	65.188	58.283	40.558

Visualization Comparison

Figure 10: Detection Predictions (after suggested improvements)

Figure 11: Detection Predictions (patches data, 3000 iterations)

Part 2: Semantic Segmentation

Network Architecture

Table 7: Convolution Class (Double Convolution)

Layer No.	Layer Type	Kernel size (For Conv layers)	Input | Output dimension	Input | Output channels (For Conv layers)
1	Conv2D	3	in| mid	in | mid
2	BatchNorm	-	mid | mid	-
3	ReLU	-	mid | mid	-
4	Conv2D	3	mid | out	mid | out
5[1]	BatchNorm	-	out | out	-
6[1]	ReLU	-	out | out	-

Table 8: Up Class (Upscales the feature map)

Layer No.	Layer Type	Kernel size (For Conv layers)	Input | Output dimension	Input | Output channels (For Conv layers)
1	Upsample / ConvTranspose	-	in | 2*in	in | in/2 (In case of CT2D)
2	Concatenation	-	2*in | 2*in	-
3[2]	Convolution	3, 3	2*in | 2*in	in | out

Table 9: Down Class (Downscales the feature map)

Layer No.	Layer Type	Kernel size (For Conv layers)	Input | Output dimension	Input | Output channels (For Conv layers)
1	MaxPool2D	-	in| in/2	-
2	Convolution	3, 3	in/2 | out	in | out

Table 10: Proposed Model (UNET)

Layer No.	Layer Type	Kernel size (For Conv layers)	Input | Output dimension	Input | Output channels (For Conv layers)
1	Convolution	3, 3	x | x	3 | 64
2	Down	-	x | x/2	64 | 128
3	Down	-	x/2 | x/4	128| 256
4	Down	3	x/4 | x/8	256 | 512
5	Down	-	x/8 | x/16	512 | 1024
6	Up	-	x/16 | x/8	1024 | 512
7	Up		x/8 | x/4	512 | 256
8	Up		x/4 | x/2	256 | 128
9	Up		x/2 | x	128 | 64
10	Conv2D	1	x | x	64 | 1

[1] This Layer is not always executed (if activation is passed as False, These Layers do not run)

[2] This is not same as Conv2D

This class's implementation is shown in this section.

Explanation

The convolution class has twice the number of Convolutions, BatchNorm and ReLU as there originally were. The convolution class has the last two layers depending on what the input to the class is. If the activation is True the last two layers will work, else not. The down class downscales the feature map using MaxPooling which halves the spatial resolution and the up class upscales the feature maps and the input to the up class are two feature maps which are concatenated and passed through the model. The Concatenation is done based on what feature map is received from the corresponding down class and that is then sent to the convolution (after appropriate padding). The overall network works in Encoder Decoder fashion. It increases the channel size from 64 (after the initial convolution from 3 to 64) up to 1024 and then it is decreased back to 64 and then a convolution is applied to get only one channel which would form the one class for the object mask.

Configuration

- Number of Epochs: 50
- Batch Size: 64
- Learning Rate: 0.001
- Weight Decay: 1e-4
- Optimizer: Stochastic Gradient Descent
- Losses used:
    - Binary Cross Entropy with Logits Loss
    - Intersection over Union Loss

Losses

The Total Loss is composed of two losses: Intersection over Union (IoU) loss and Binary Cross Entropy. A simple summation (equally weighted) was applied to the two losses. The total summed loss is: **0.5465046167373657**, whereas BCE Loss is equal to **0.24857157468795776** and the IoU loss is equal to **0.29793304204940796** after 50 epochs.

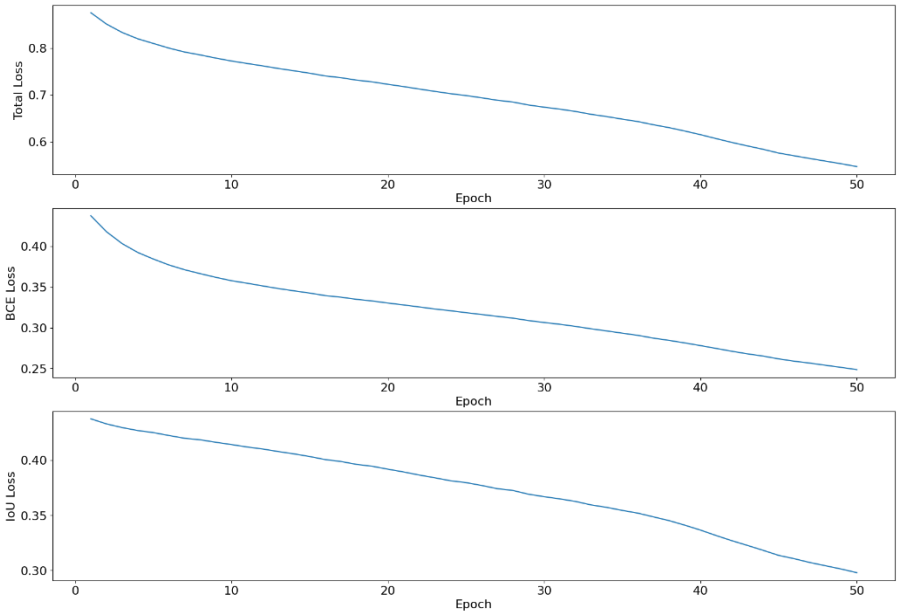

Figure 12: Segmentation Loss Plots (Total loss: top, BCE loss: middle, IoU loss: bottom)

Mean Intersection-over-Union score (Validation set)

The Mean Intersection-over-Union score for the validation set is equal to **0.7134938019712923**.

Visualization

Figure 13: Semantic Segmentation Predictions (thin model)

Figure 14: Semantic Segmentation Predictions (after suggested improvements)

Part 3: Instance Segmentation

Kaggle Submission

I have submitted my csv and it got **36.922%** on Kaggle.

Visualization

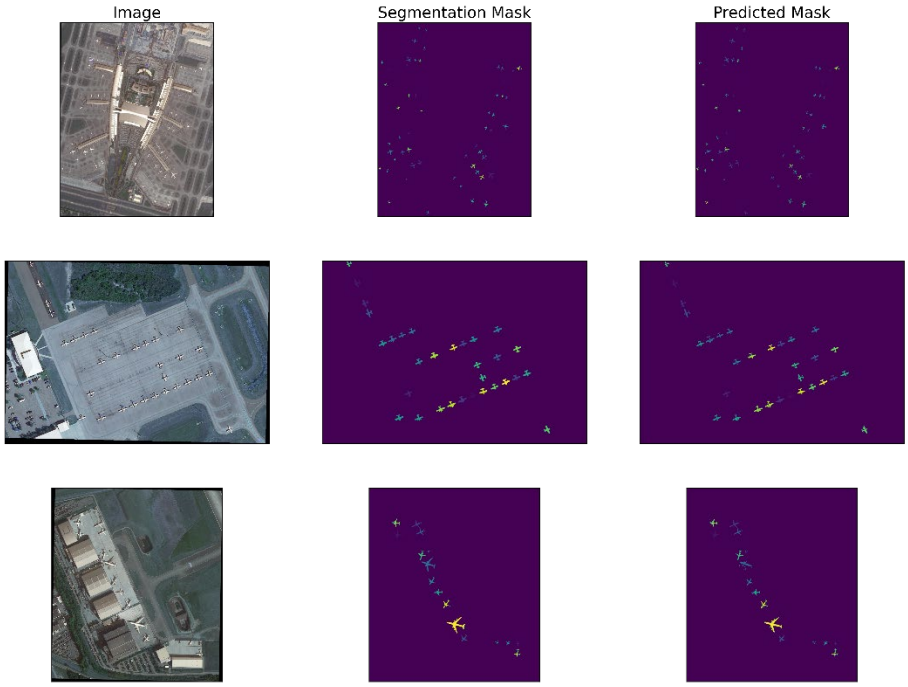

Figure 15: Instance Segmentation Predictions (validation set)

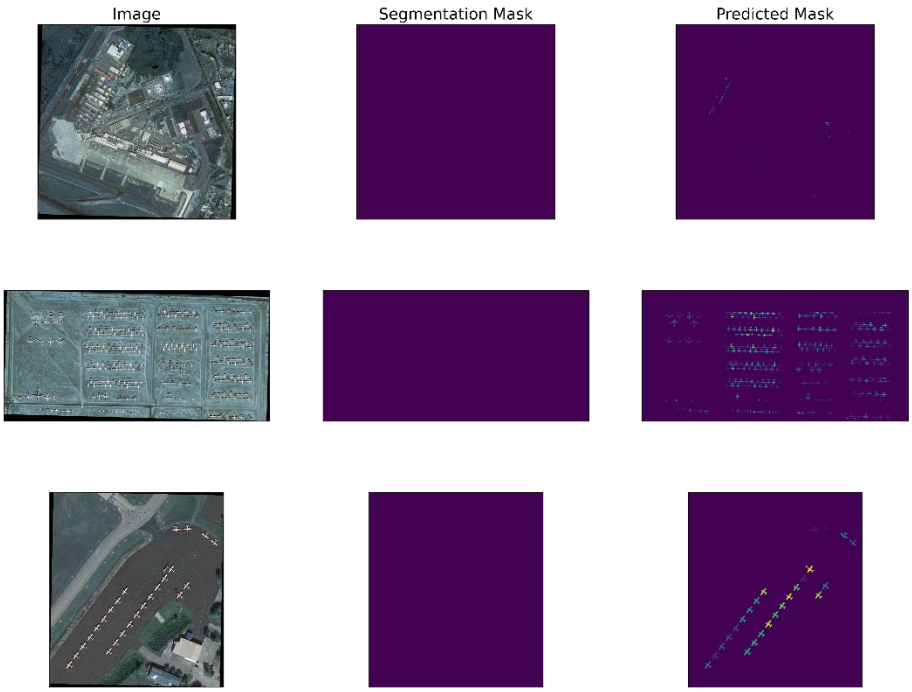

Figure 16: Instance Segmentation Predictions (test set)

Part 4: Mask RCNN

Losses

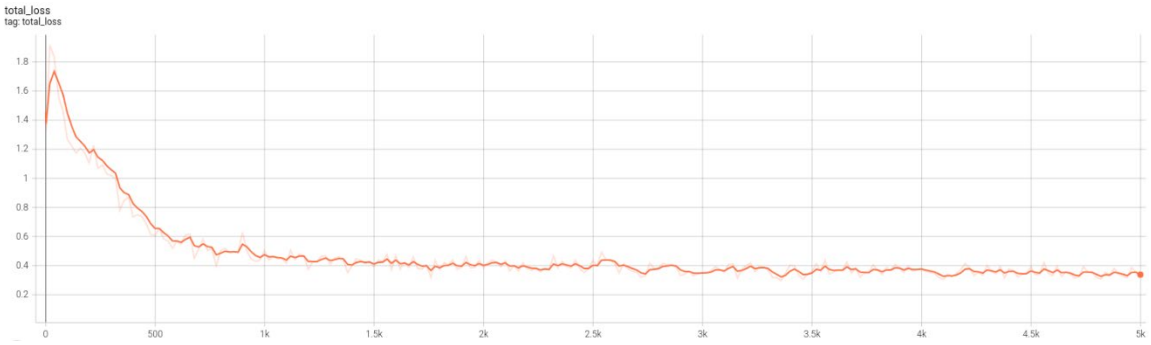

Figure 17: Training Loss Plot

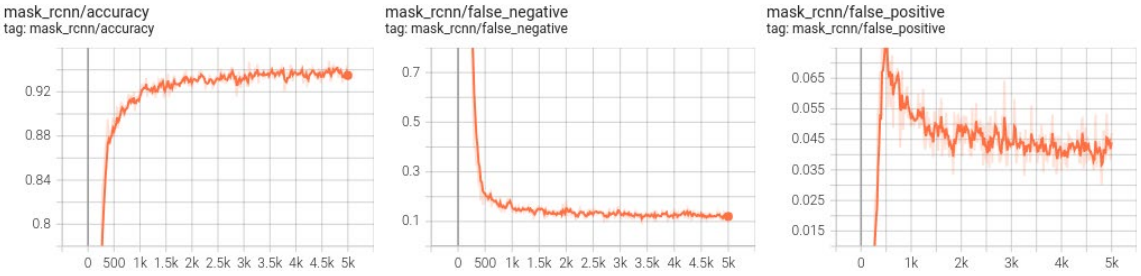

Figure 18: Classification Accuracy Plot

Comparison to Detection and Instance Segmentation Model

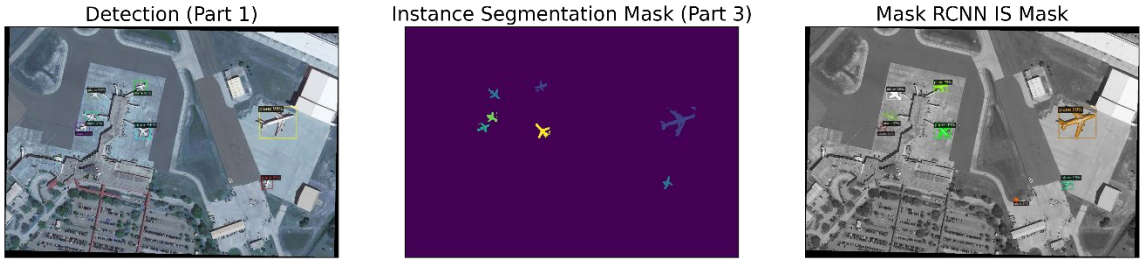

Figure 19: Comparison between My Model (Detection: left, Semantic Segmentation: middle) and Mask R-CNN (right)

Comparison with Detection Model

Table 11: Validation Loss (Mask R-CNN detection, 3000 iterations)

AP	AP50	AP75	APs	APm	APl
60.377	86.210	72.843	65.647	57.667	37.212

Table 12: Validation Loss (Mask R-CNN detection, 5000 iterations)

AP	AP50	AP75	APs	APm	APl
61.517	85.989	73.176	66.696	59.688	31.893

The Mask RCNN Model is much faster than the other model that I used, in part 1, and in my testing, I needed to run the Mask RCNN model for longer. The original model, from part 1, when run for 5000 iterations was comparable to the Mask RCNN model run on 3000 iterations. This is because Mask RCNN is a combination of two models, Fully Convolutional Network and Faster RCNN. The Faster RCNN has two outputs: label and bounding box for that label. Mask RCNN has another module which outputs directly the object segmentation mask (which does the instance segmentation without any postprocessing in an end-to-end pipeline).

Comparison with Instance Segmentation Model

Table 13: Validation Loss (Mask R-CNN segmentation, 3000 iterations)

AP	AP50	AP75	APs	APm	APl
43.532	89.775	31.635	41.465	49.153	57.615

As the Mask RCNN model is an end-to-end model and has the losses which Faster RCNN has and in addition to that the object segmentation mask pipeline is optimized by its own pipeline. So, as already mentioned, Mask RCNN is much faster than my previous two networks as it is end-to-end model. My detection model took about an hour for 3000 iterations, whereas Mask RCNN took less than 20 minutes for both detection and instance segmentation. Whereas, for the visual aspect of the images, both my proposed semantic segmentation model and Mask RCNN did quite similar.