CMPT 762 Assignment 3

301474102 Sihui Wang

Part 1 Object Detection

1.1 Baseline Training

Note: To speed up experiments, I opened multiple Colab documents and did many experiments in parallel. For this part, please refer to:
https://colab.research.google.com/drive/1TSSWE-MK9oMgyO7_nkwZrFnukKzbco5P?usp=sharing

1.1.1 Original Configuration

Parameters for Training:

Model: Faster R-CNN + ResNet-101 + FPN
(COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml)

BATCH_SIZE_PER_IMAGE: 512

IMS_PER_BATCH: 2

BASE_LR: 0.00025

MAX_ITER: 500

Parameters for Validation:

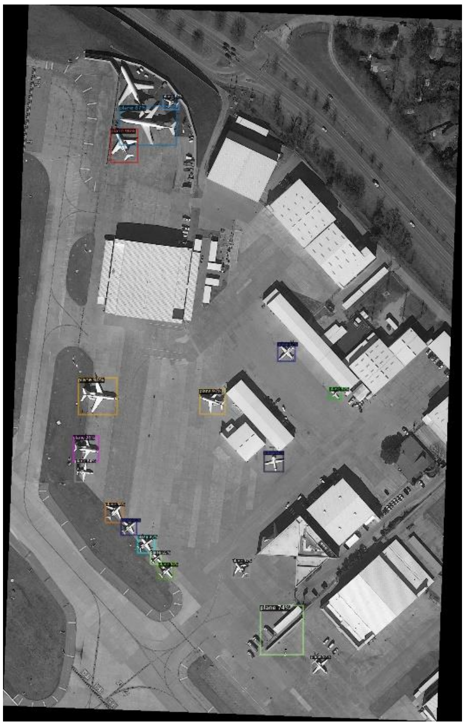
Train-Validation split: 80% - 20%

There are 198 images under 'train' folder. 80% are used for training, and 20% are used for validation.

Parameters for Testing:

SCORE_THRESH_TEST: 0.6

1.1.2 Visualization on Random Test Data

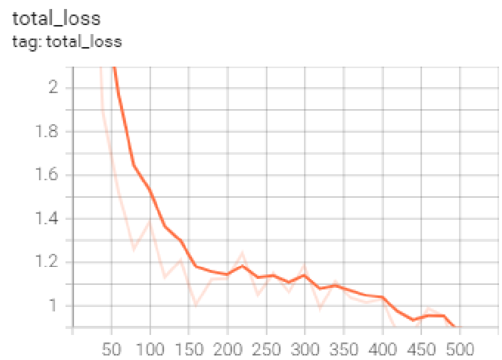



1.1.3 Training Losses and Class Accuracy

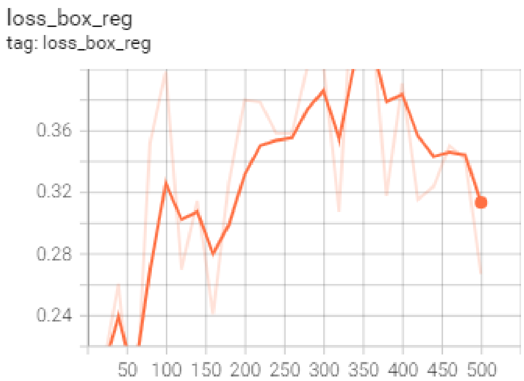
Note: the lighter curve reflects the original results, whereas the darker curve is smoothed with the coefficient 0.6.

Total Training Loss of the Original Model:

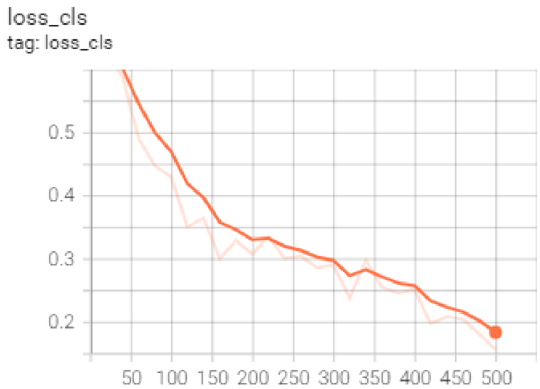
Total Loss



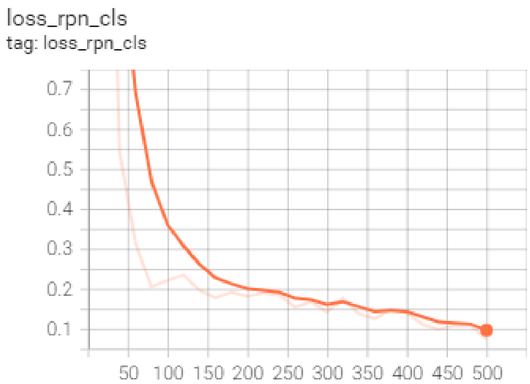
Loss of Classification



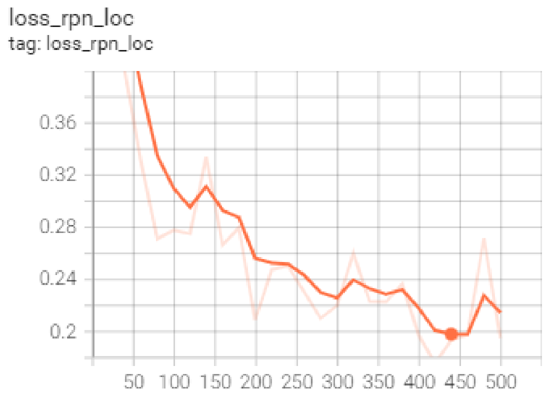
Loss of Box Regression



Loss of RPN Classification

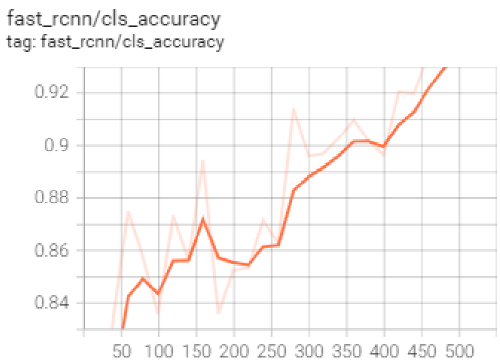


Loss of RPN Box Location

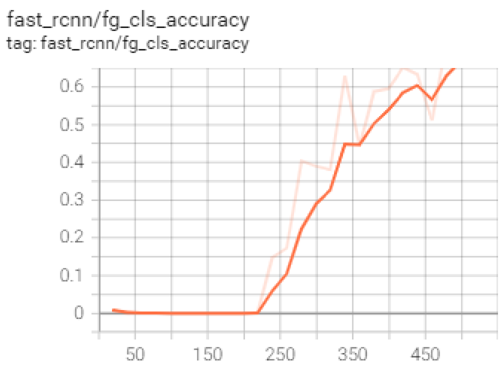


Class Accuracy of the Original Model:

Classification Accuracy



Foreground Classification Accuracy



1.1.4 Evaluation on the Validation Set

AP-50 is 35.409 for the original model over 500 epochs of training.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.188
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.354
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.174
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.144
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.325
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.432
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 ] = 0.116
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.215
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.143
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.370
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.758
[03/07 01:10:22 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 18.803 | 35.409 | 17.426 | 14.441 | 32.493 | 43.177 |
```

1.2 List of Edits and Ablation Studies

1.2.1 List of Edits

Edit 1: Change the Model to Faster RCNN + ResNeXt – 101 + FPN

https://colab.research.google.com/drive/1BcEbpz3u2e1BbFBASVScfuyOs-DUwQZH?usp=sharing

Edit 2: Increase Training Epochs from 500 to 1000

https://colab.research.google.com/drive/1z8i98agpiGRPmhdew3NRhjgbjN5g_1Zj?usp=sharing

Negative Attempts:

I also tried the following modifications; however, they didn't lead to improvements in the performance in my experiments.

Change BATCH_SIZE_PER_IMAGE from 512 to 768

Change BATCH_SIZE_PER_IMAGE from 512 to 384

Use a Custom Data Mapper to randomly crop 320×240 regions in the image during training

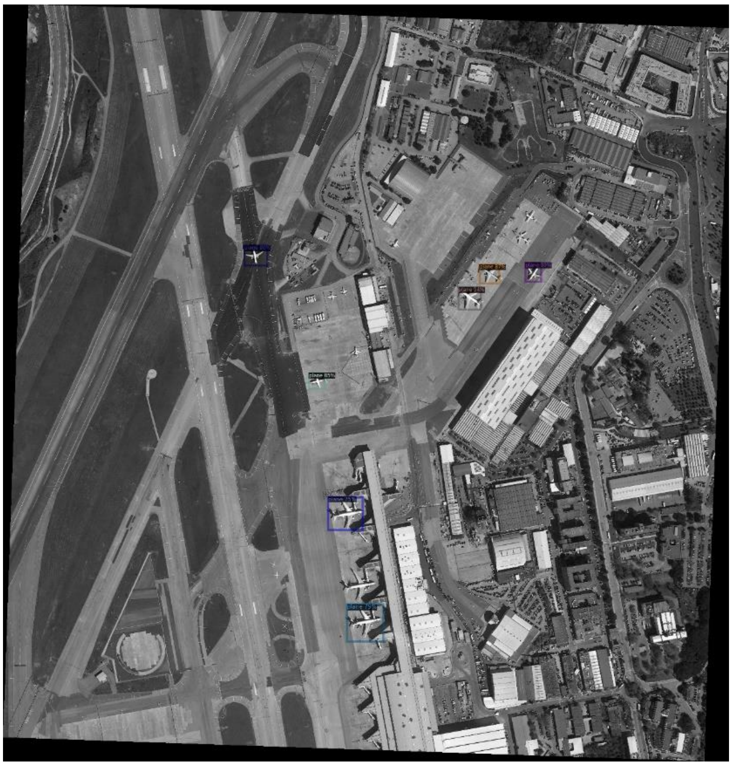
1.2.2 Ablation Study

Quantitative Results:

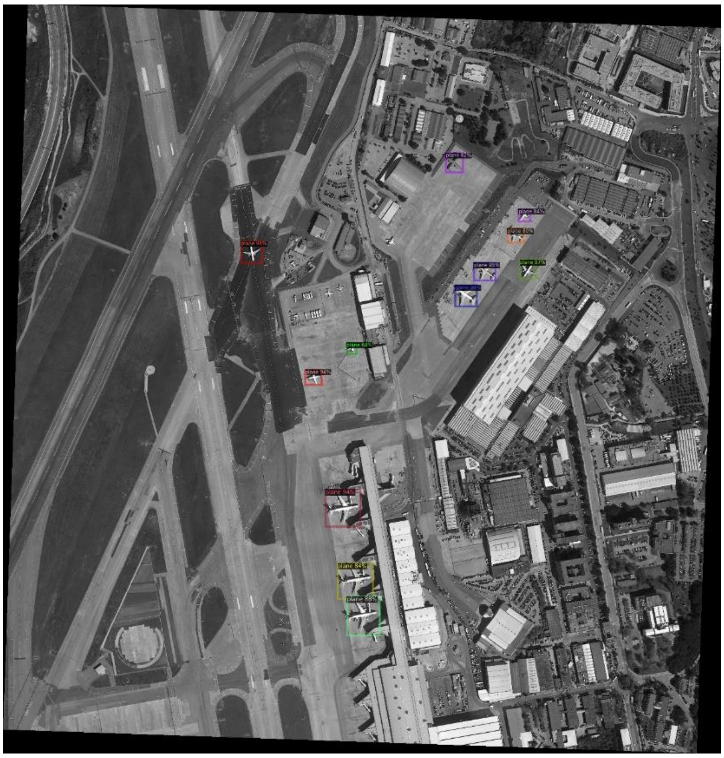
Edits	AP-50
Original	35.409
Edit 1: Change the model	50.089
Edit 2: Increase the number of epochs	57.384

Qualitative Results:

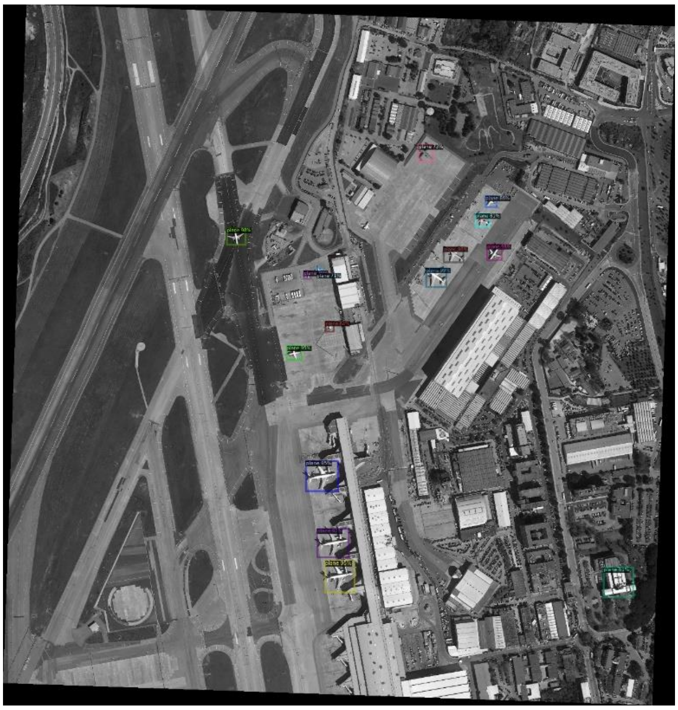
Original Implementation:



Edit 1: Change the Model:



Edit 2: Increase the number of epochs



From the above visualization we can conclude that after edit 1, the model is able to detect more planes compared with the baseline; after edit 2, the model is able to detect more planes compared with edit 1. Hence, the above visualization validates that th e edits improves the framework's performance in object detection.

1.2.3 Explanation of Factors:

Change the Model: According to Detectron's model zoo web page:

[detectron2/MODEL_ZOO.md at main · facebookresearch/detectron2 (github.com)]

X101-FPN is superior to R101-FPN in performance measured by box AP. The original paper, [1611.05431.pdf (arxiv.org)], also shows that the ResNeXt backbone network improves classification accuracy by increasing cardinality, and it outperforms the ResNet counterpart on ImageNet-5K and COCO detection dataset, which is why I choose to replace the ResNet backbone by the ResNext backbone for better performance.

Increasing the number of epochs: With limited number of training epochs, the model would be underfitting with the training data, and the performance would be below the full capacity of the model. By increasing the number of training of epochs without overfitting, the model would be better at explaining the training data and generalizing to the validation datasets. In my experiments, increasing the training epochs can improve AP-50, which indicates greater object detection accuracy with increased training epochs.

1.3 Final Model:

1.3.1 Final Configuration

Parameters for Training:

Model: Faster R-CNN + ResNeXt-101 + FPN (COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml)

BATCH_SIZE_PER_IMAGE: 512

IMS_PER_BATCH: 2

BASE_LR: 0.00025

MAX_ITER: 1000

Parameters for Validation:

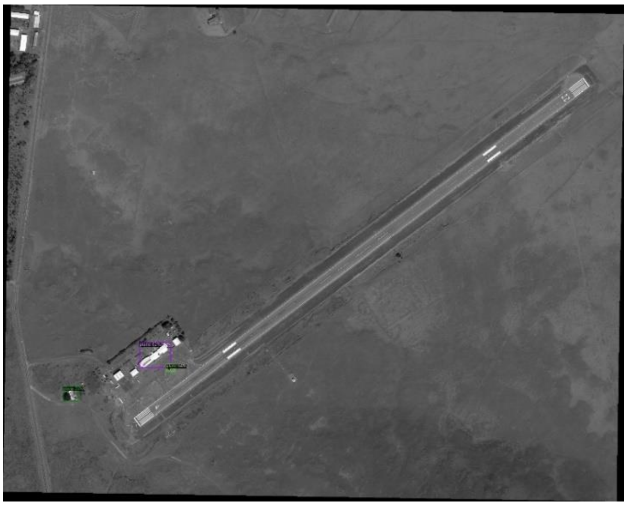
Train-Validation split: 80% - 20%

There are 198 images under 'train' folder. 80% are used for training, and 20% are used for validation.

Parameters for Testing:

SCORE_THRESH_TEST: 0.6

1.3.2 Visualization on Random Test Data

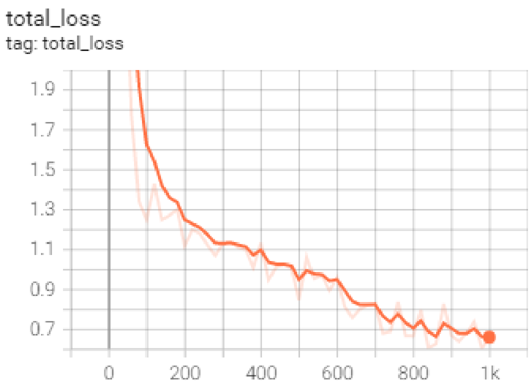


1.3.3 Training Losses and Class Accuracy

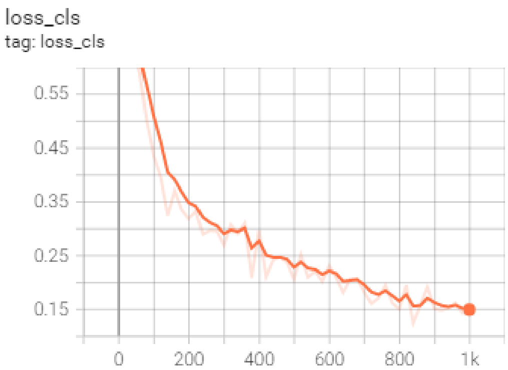
Note: the lighter curve reflects the original results, whereas the darker curve is smoothed with the coefficient 0.6.

Total Training Loss of the Final Model:

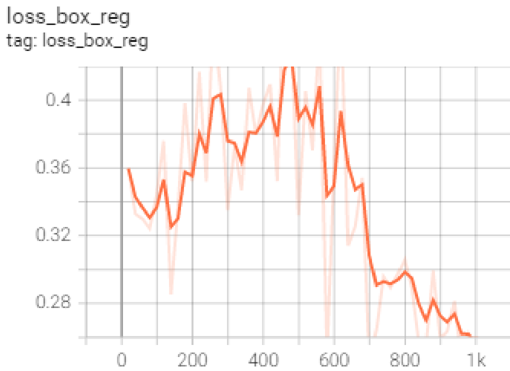
Total Loss



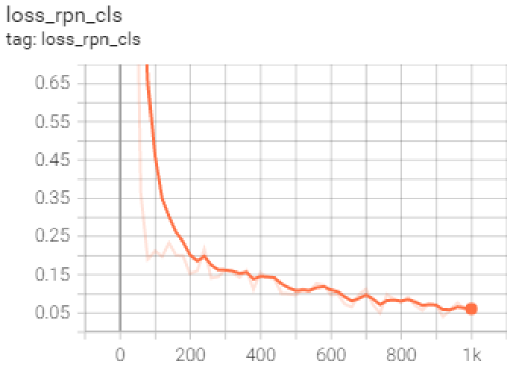
Loss of Classification



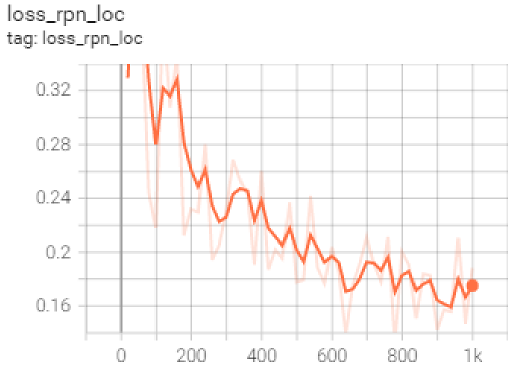
Loss of Box Regression



Loss of RPN Classification

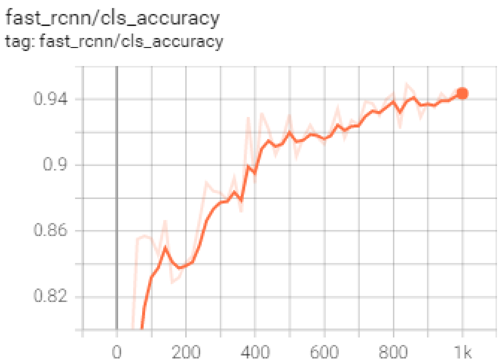


Loss of RPN Box Location

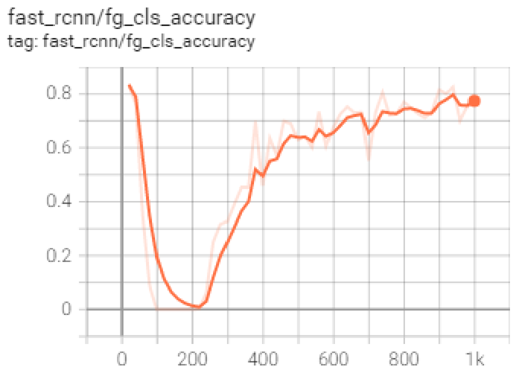


Class Accuracy of the Final Model:

Classification Accuracy



Foreground Classification Accuracy



1.3.4 Evaluation on the Validation Set

AP-50 is 57.384 for the final model over 1000 epochs of training.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.331
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.574
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.365
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.271
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.522
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.677
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.017
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.149
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.368
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.272
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.581
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.817
[03/07 06:12:43 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 33.068 | 57.384 | 36.503 | 27.134 | 52.185 | 67.699 |
```

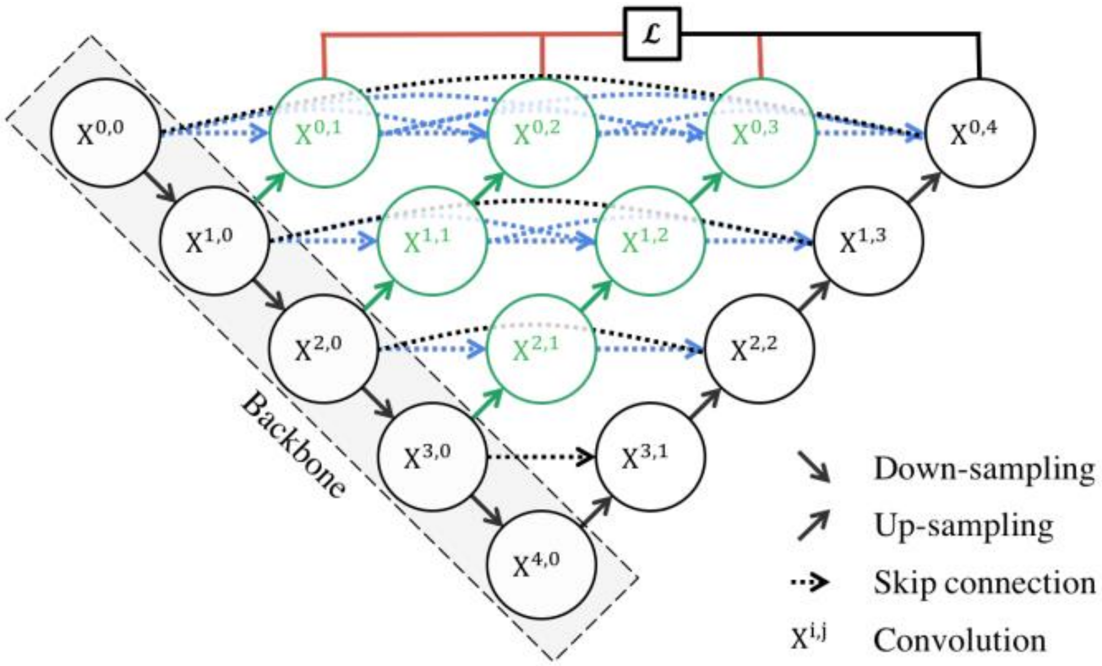
Part 2 Semantic Segmentation

2.1 Final Model Architecture

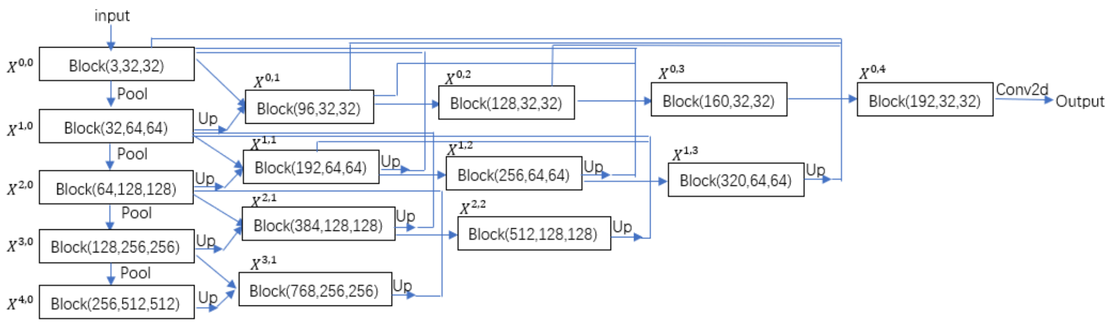
The final model is: Nested UNet (UNet++ L^4)

https://colab.research.google.com/drive/1kEhLJPAgAhhUHkAh3tqjc_RDHT83LiJr?usp=sharing

Below is the architecture of Nested UNet illustrated in the original paper. The red pathways compute the losses for the “deep supervision” mechanism. Since I don’t use “deep supervision”, in my implementation the output $\mathcal{L}$ is computed by the black pathway.

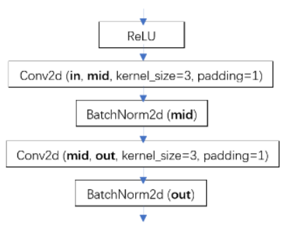


The following is the network architecture of my implementation of Nested UNet:



In the above figure, “Pool” is implemented by nn.MaxPool2d (2,2), “up” is implemented by nn.UpSample (scale_factor=2, mode=’bilinear’, align_corners=True). The “Conv2d” for the output layer is implemented by nn.Conv2d (32,1, kernel_size=1).

In Nested UNet, $Block(in, mid, out)$ is the main building block. It has the following architecture:



Reason for the modification:

I have learned in the course CMPT 733 that UNet is well tailored for the problem of medical image segmentation. So, it is very likely that UNet architecture will also receive good performance on the problem of semantic segmentation.

UNet is an encoder-decoder network where the encoder and decoder subnetworks are connected by skip pathways. Based on UNet, UNet++ (1807.10165.pdf (arxiv.org)) proposed to add more layers and redesign the dense skip pathways, in order to reduce the semantic gap between the feature maps of the encoder and the decoder. It is reported that UNet++ achieves IoU gains compared with UNet and wide UNet, which is why I choose nested UNet (UNet++) as the architecture for segmentation.

2.2 Configurations of Hyper-parameters

Training – Validation Split: 85% - 15%
Input Image Size: 256 × 256
Batch size: 4
Number of Epochs: 35
Optimizer: optim.SGD (stochastic gradient descent)
Learning Rate: 0.006
Momentum: 0.9
Weight_decay: 0.0005
Learning Rate Scheduler: CosineAnnealingLR

2.3 Loss Function and Training Loss

Loss Function: weighted sum of binary cross entropy (BCE) loss and dice loss.
Ideally, dice loss between input and target is defined by:

$$loss_{dice} = 1 - DSC = 1 - \frac{2|input \cap target|}{|input| + |target|}$$

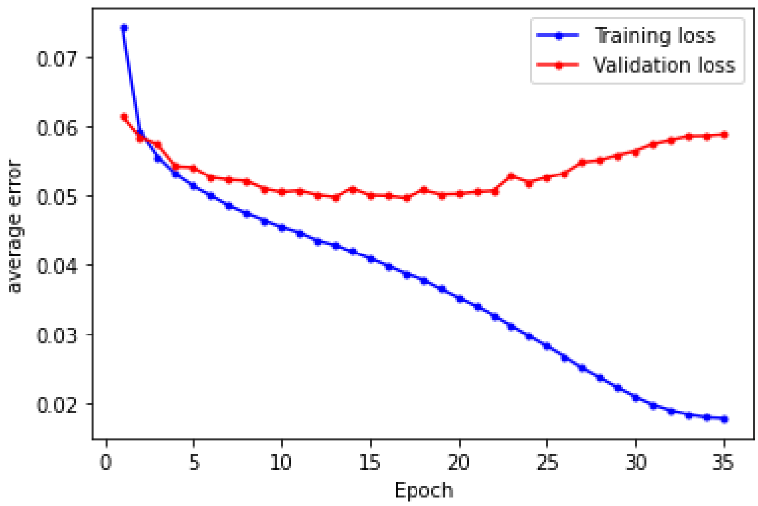
In practice, $|input \cap target|$ is calculated by $(input * target).sum()$, and $|input| + |target|$ is calculated by $input.sum() + target.sum()$. To guarantee stability, we add a smooth term to both the numerator and denominator:

$$loss_{dice} = 1 - 2 * ((input * target).sum + smooth)/(input.sum + target.sum + smooth)$$

The final loss function is defined as follows:

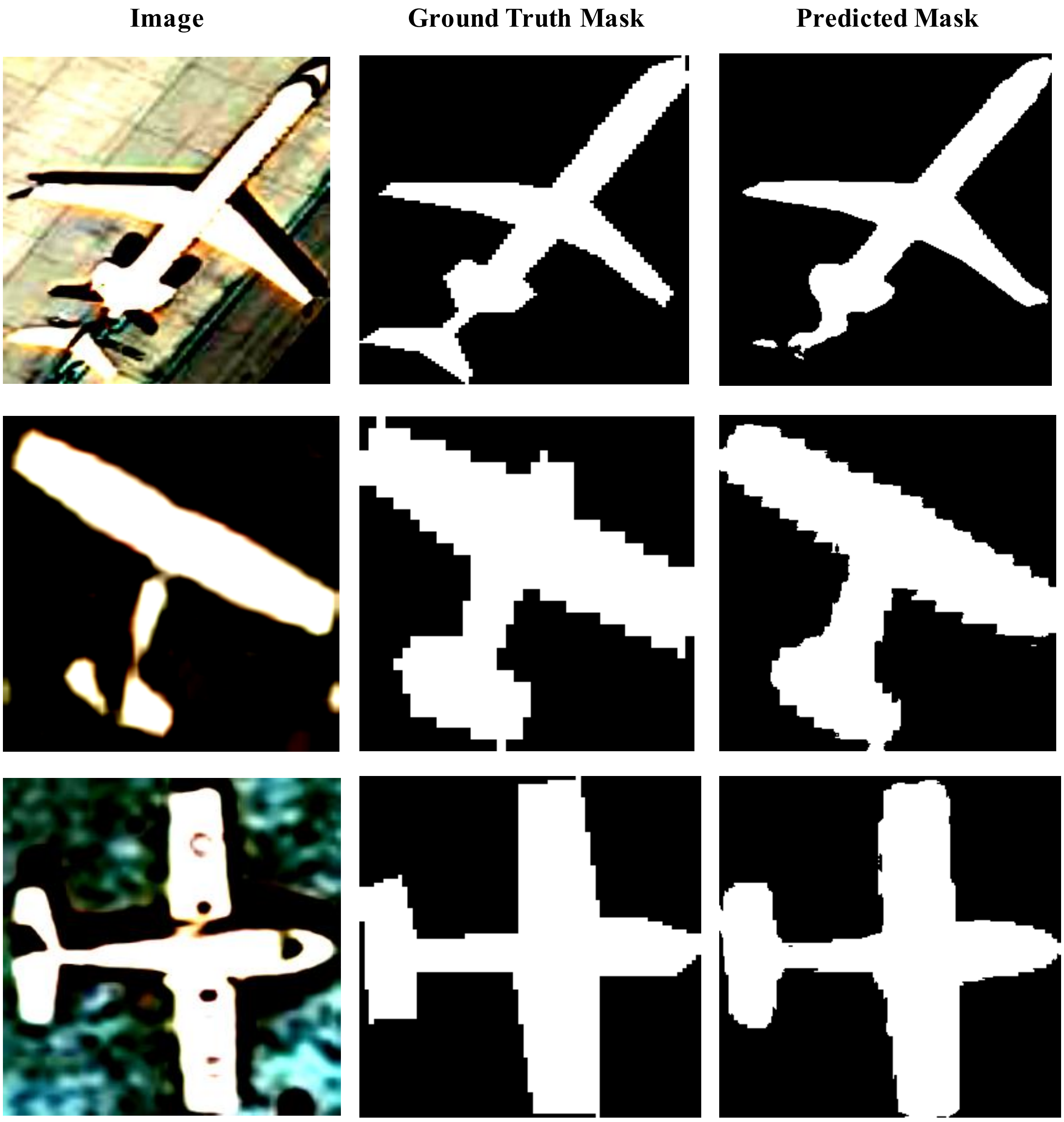
$$loss = \frac{1}{2} loss_{BCE} + loss_{dice}$$

Below is the plot of losses during the training process:



2.4 Final Mean IoU: 0.8576

2.5 Visualization of Test Results on Validation Set



Part 3 Instance Segmentation

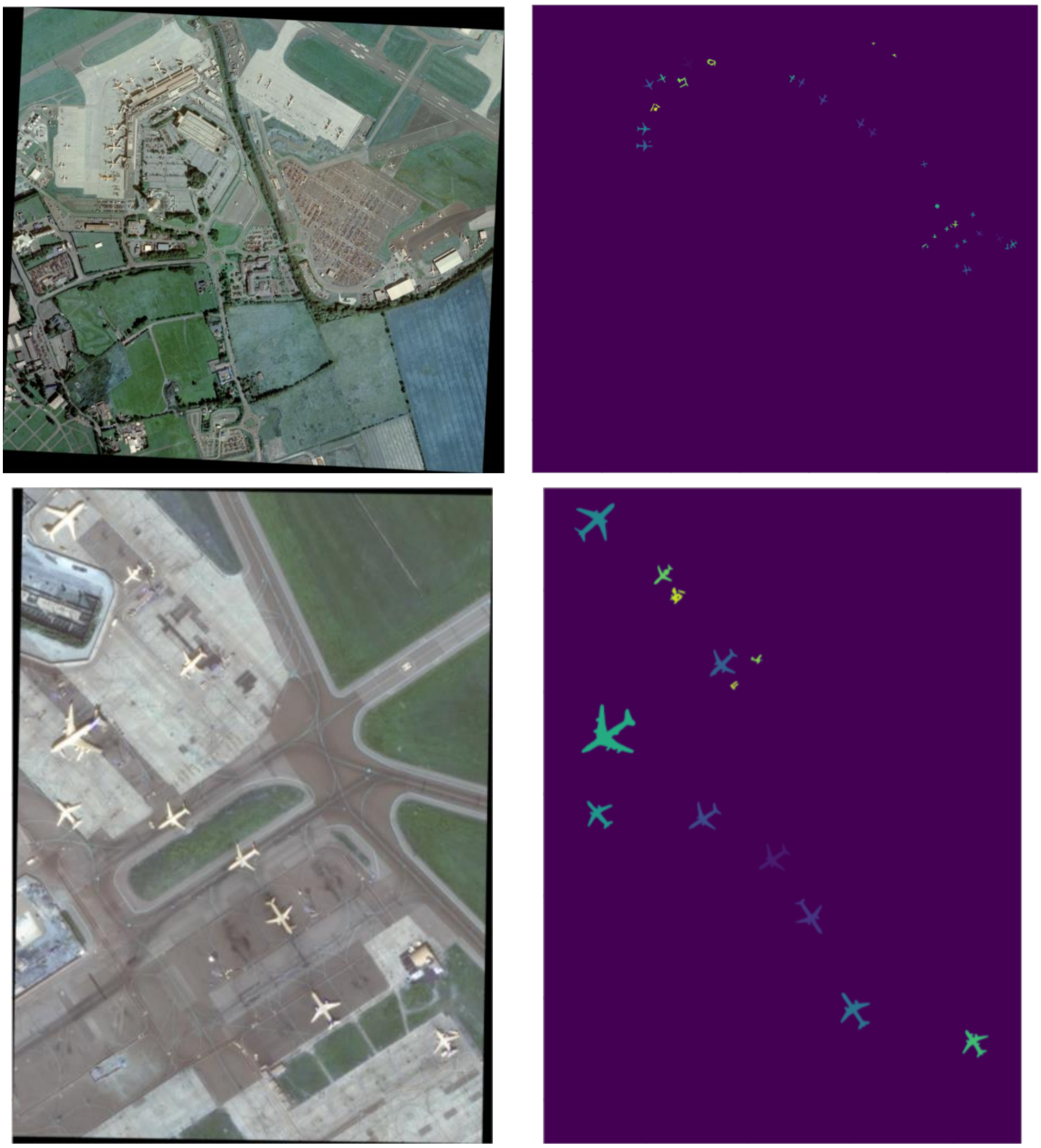
3.1 Kaggle Account Name: Scott

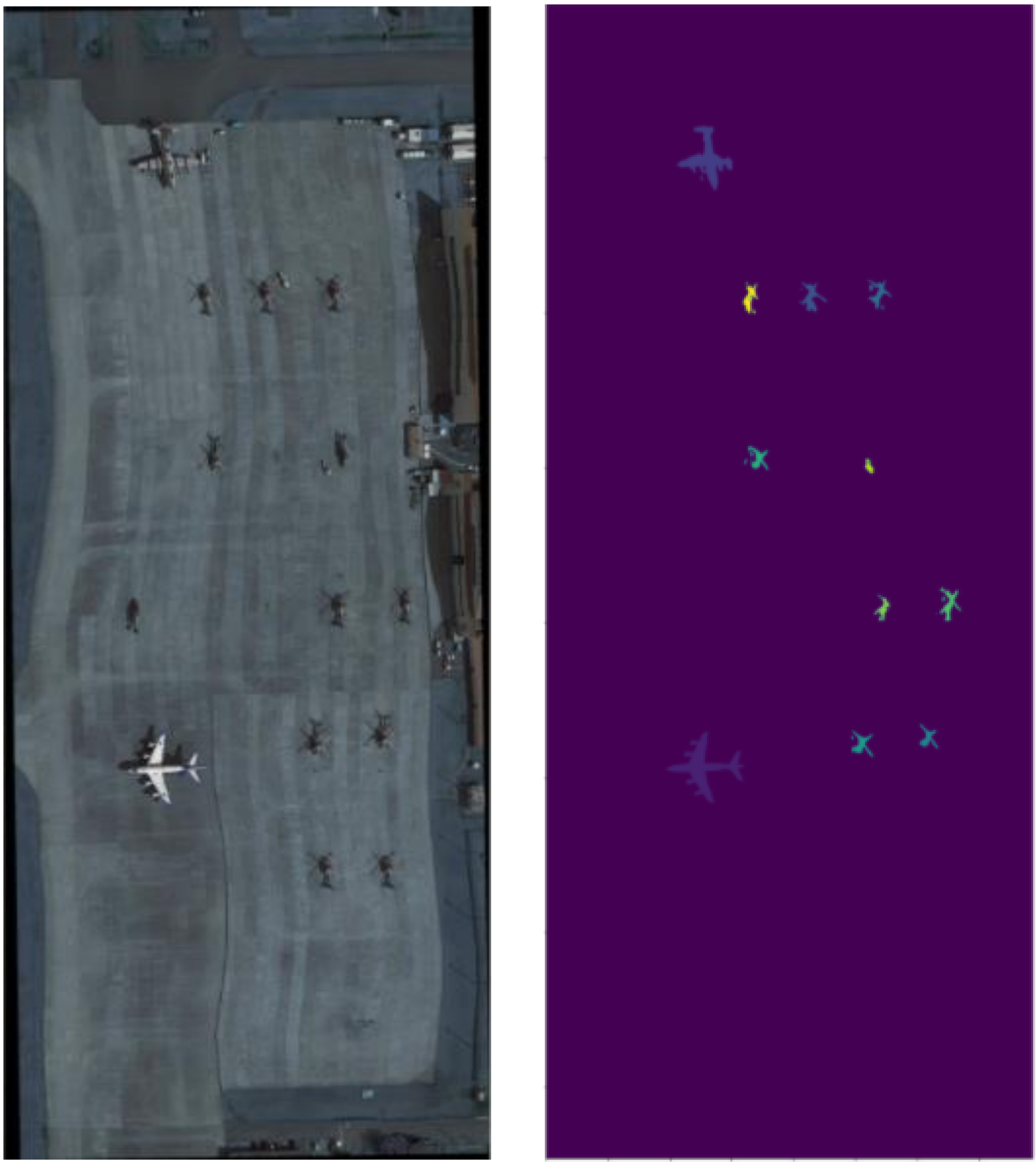
3.2 Best Score: 0.64247

3.3 Visualization of the Results

Image

Prediction

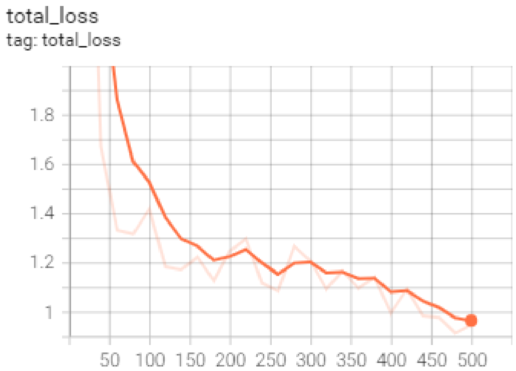




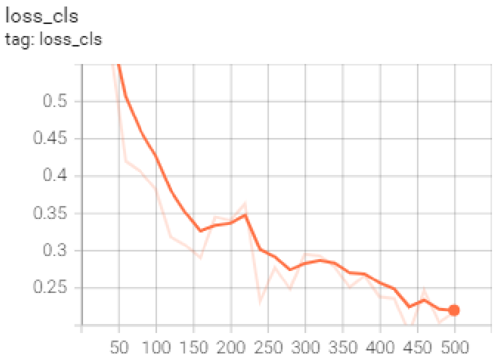
Part 4 Mask-RCNN

4.1 Training Loss and Accuracy of Mask RCNN on Instance Segmentation Task

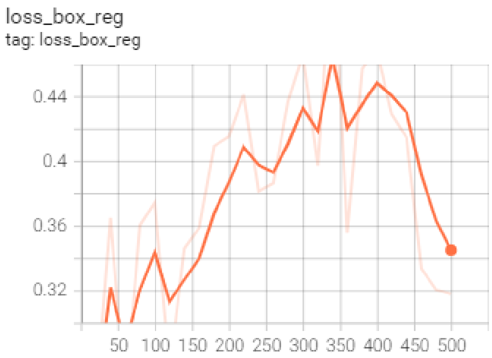
Total Loss



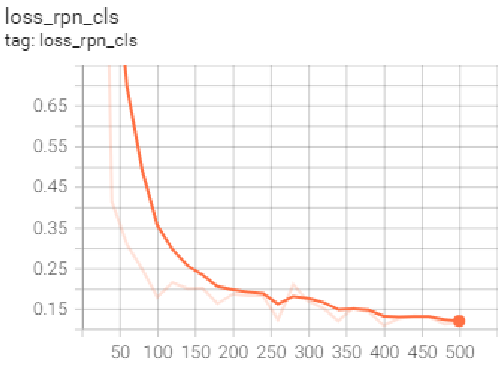
Loss of Classification



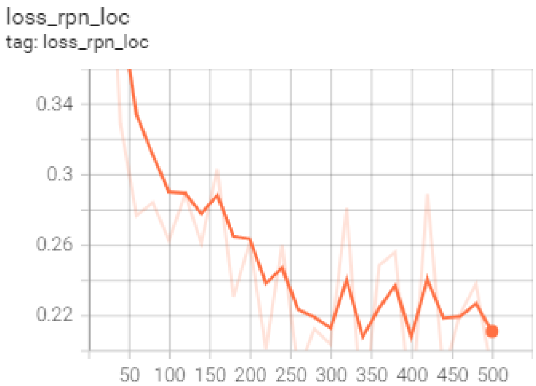
Loss of Box Regression



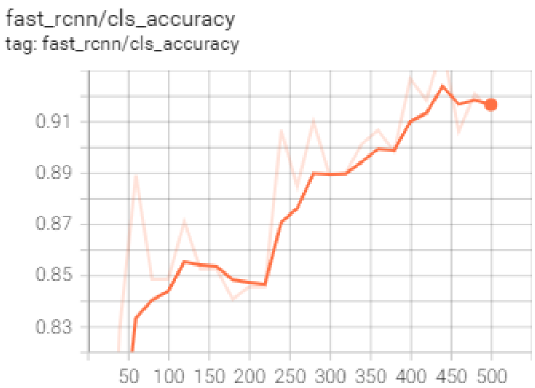
Loss of RPN Classification



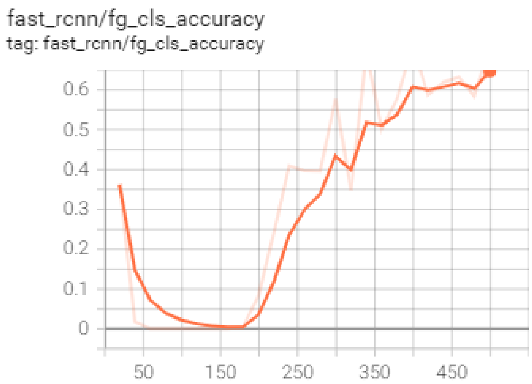
Loss of RPN Box Location



Classification Accuracy



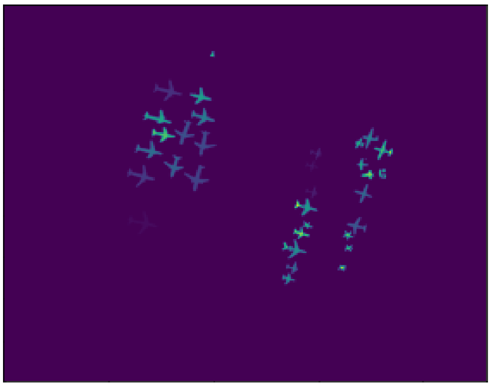
Foreground Classification Accuracy



4.2 Visualization on Test Images:

Object Detection Result from Part 1

Semantic Segmentation Result from Part 3



Object Detection and Semantic Segmentation Result from Part 4 (Mask RCNN)

4.3 Comparison

4.3.1 Comparison of Object Detection

On the above sample image, it seems that "Faster R-CNN + ResNeXt-101 + FPN" in part 1 outperforms Mask RCNN. While "Faster R-CNN + ResNeXt-101 + FPN" is able to detect most of the planes in the image, Mask RCNN missed a substantial proportion of planes, which indicates that "Faster R-CNN + ResNeXt-101 + FPN" wins by a large margin in object detection.

4.3.2 Comparison of Instance Segmentation

According to the above sample image, "Faster R-CNN + ResNeXt-101 + FPN + Nested UNet" outperforms Mask RCNN in instance segmentation accuracy. Mask RCNN's instance masks sometimes can't fully cover the object areas, and the shapes of the instance masks are not always correct. Although "Faster R-CNN + ResNeXt-101 + FPN + Nested UNet" also makes inaccurate instance masks, its masks can better match the shape of the planes. So, "Faster R-CNN + ResNeXt-101 + FPN + Nested UNet" has better overall performance in instance segmentation compared with Mask RCNN.