# Object Detection, Semantic Segmentation, and Instance Segmentation

CMPT 412 - Computational Vision

Fitzpatrick Laddaran

February 28, 2023

## 1 Part 1 - Object Detection

This section pertains to Part 1 of the project handout.

### 1.1 Configurations & Modifications

The configurations that I used are as follows:

- Baseline configuration: faster_rcnn_X_101_32x8d_FPN_3x.yaml
- `MAX_ITER` = 500
- `SOLVER.BATCH_SIZE_PER_IMAGE` = 512
- `IMS_PER_BATCH` = 2
- `BASE_LR` = 0.00075
- `ROI_HEADS.BATCH_SIZE_PER_IMAGE` = 128

The following results are based on the validation set that I created. The validation set contains 30 randomly selected images from the training set; consequently, the training set only contains 168 images. The selection was seeded with `numpy.random.seed(1234)`.

The first modification was changing the baseline configuration used; instead of the recommended `faster_rcnn_R_101_FPN_3x.yml`, the configuration `faster_rcnn_X_101_32x8d_FPN_3x.yaml` gave a significant boost. Originally, I had an AP score of about 0.25. Changing the configuration yielded a score of about 0.32 (with the baseline parameters).

The other modification that made slight changes to the AP score is changing the learning rate. Using configuration `faster_rcnn_X_101_32x8d_FPN_3x.yaml`, I tried a few different learning rates: 0.00025, 0.0005, 0.00075, and 0.01. Anything higher caused technical challenges since the loss function would fail to converge towards a local (or global) minima. As such, a learning rate of 0.00075 gave me the best AP score of about 0.35.

Finally, modifying the other values gave worse AP scores. I kept the recommended baseline values for `MAX_ITER`, `SOLVER.BATCH_SIZE_PER_IMAGE`, and `IMS_PER_BATCH`.

Figure 1 illustrates the resulting scores generated by running `inference_on_dataset()` on the validation set.

### 1.2 Sample Visualization & Predictions: Test Set

Using the configurations listed above, I visualized some results on the test set. Figure 2 illustrates results on image 21 (or `test_set[20]`) from the test set. Figure 3 illustrates results on image 31 (or `test_set[30]`). Figure 4 illustrates results on image 71 (or `test_set[70]`).

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.356
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.565
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.421
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.264
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.448
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.711
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.022
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.170
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.407
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.275
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.540
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.761
[03/03 02:12:54 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 35.563 | 56.481 | 42.120 | 26.404 | 44.755 | 71.077 |
```

Figure 1: Scores from running `inference_on_dataset()` using the modified configurations.

### 1.3 Ablation Study

As discussed above, the modification that improved my AP score the greatest was the change in baseline configuration. Instead of using the baseline configuration `faster_rcnn_R_101_FPN_3x.yml`, I used the configuration `faster_rcnn_X_101_32x8d_FPN_3x.yaml`. This modification yielded immediate performance improvements with no additional parameter optimization.

Comparing with configuration `faster_rcnn_R_101_FPN_3x.yml` (while using the same parameters), I got slightly different predictions. While using `faster_rcnn_R_101_FPN_3x.yml`, Figure 5 illustrates predictions on image 21 (or `test_set[20]`) from the test set. Figure 6 illustrates predictions on image 31 (or `test_set[30]`). Figure 7 illustrates predictions on image 71 (or `test_set[70]`).

Based on visual inspection, it seems that both models have relatively decent performance; however, I did notice some offsets. In Figure 3, the tail-end of the right-most plane is being detected as another plane. This is not the case in the recommended configuration as illustrated in Figure 6. In both Figures 4 and 7, some of the planes are not detected. And lastly, in Figure 5, the detector seems to be detecting planes in locations where planes are not visually-identifiable (to the human eye).

### 1.4 Training Loss and Accuracy

Figure 8 shows my model's accuracy, and Figure 9 shows the total loss as generated by the notebook.

## 2 Part 2 - Semantic Segmentation

This section pertains to Part 2 of the project handout.

The hyperparameters I used are as follows:

- `batch_size`: 32
- `learning_rate`: 0.01
- `num_epochs`: 25
- `optimizer`: `torch.optim.SGD`

Figure 10 and Figure 11 show the final architecture of my network. I modified the structure by using the down-sampling function first, then followed it with the up-sampling function. The loss function that I used is the one provided in the lab, the logit loss function. The value of my loss function was steady at around 0.32 after 25 epochs.

On the training set, the mean IOU of my model is about 0.65. On the validation set, the mean IOU of my model is 0.57.

Figures 12 illustrates three visual examples from the test set. The top image is the original image, the middle image is the predicted mask, and the bottom image is the ground truth mask.

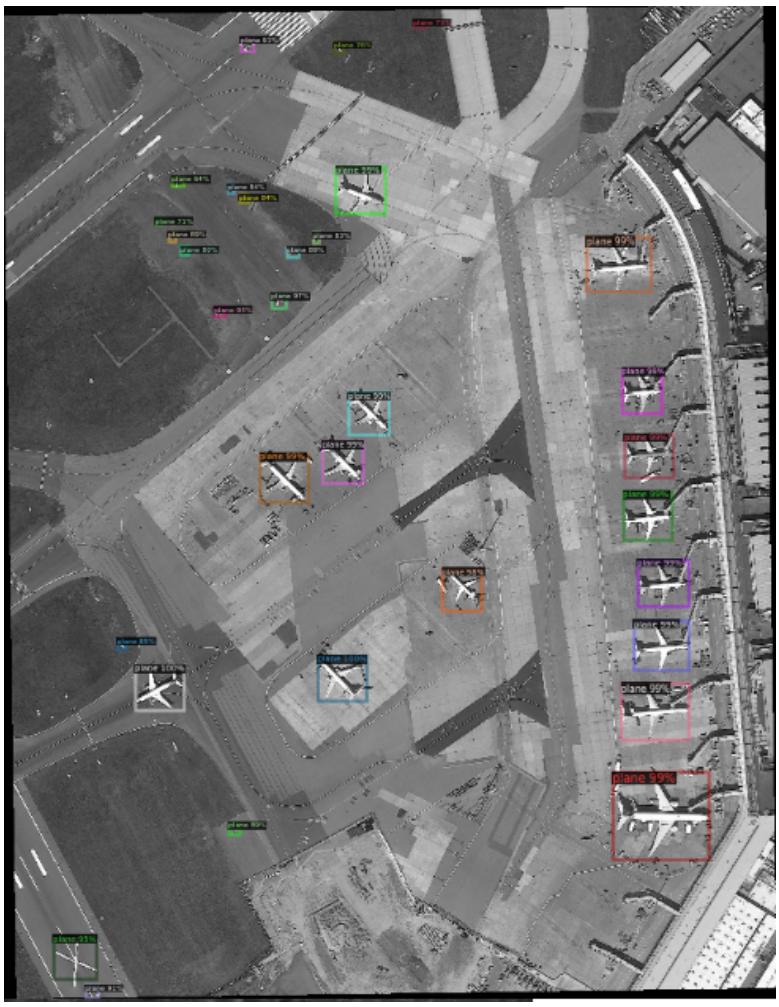

Figure 2: Visualization of predictions on `test_set[20]` using modified configurations.

# 3 Part 3 - Instance Segmentation

Kaggle ID: **Airplanes**

**Best prediction accuracy is 25.474%**, matching the results on Kaggle.

Figures 13, 14, and 15 are example visualizations from the test set.

# 4 Part 4 - Mask R-CNN

In comparison with the results from Part 1, it seems that the Mask R-CNN is really sensitive. Take a look at Figure 17, where the model is detecting more planes than there are in reality. Similarly with both Figures 16 and 18, the sensitivity seems to help because it is able to detect planes more easily; however, there are also instances where planes are detected but are not suppose to be.

Comparing the differences between Part 3 and Part 4, the models highly different in sensitivity when detecting the planes. Part 3 uses a model that is less sensitive in comparison to Part 4.

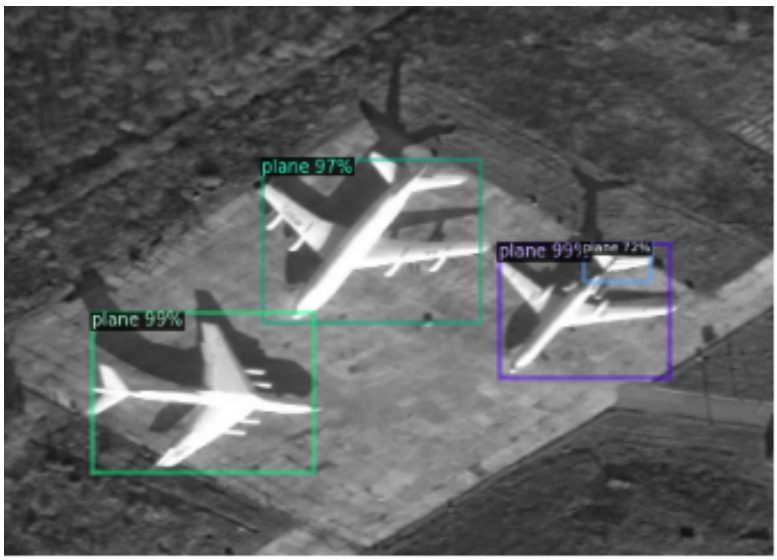


Figure 3: Visualization of predictions on `test_set[30]` using modified configurations.

Figure 4: Visualization of predictions on `test_set[70]` using modified configurations.

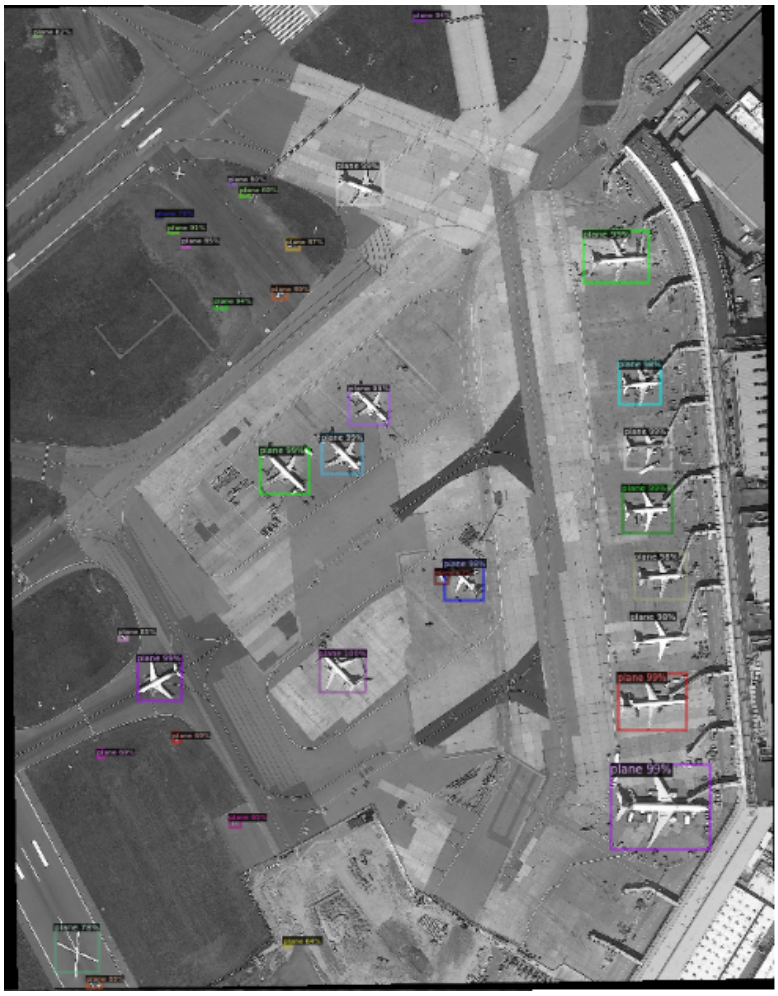

Figure 5: Visualization of predictions on `test_set[20]` using baseline configurations.

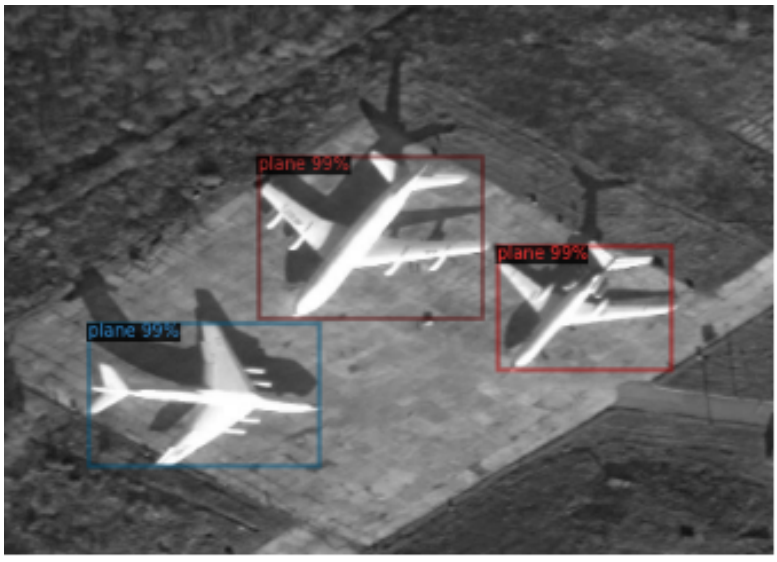


Figure 6: Visualization of predictions on `test_set[30]` using baseline configurations.

Figure 7: Visualization of predictions on `test_set[70]` using baseline configurations.

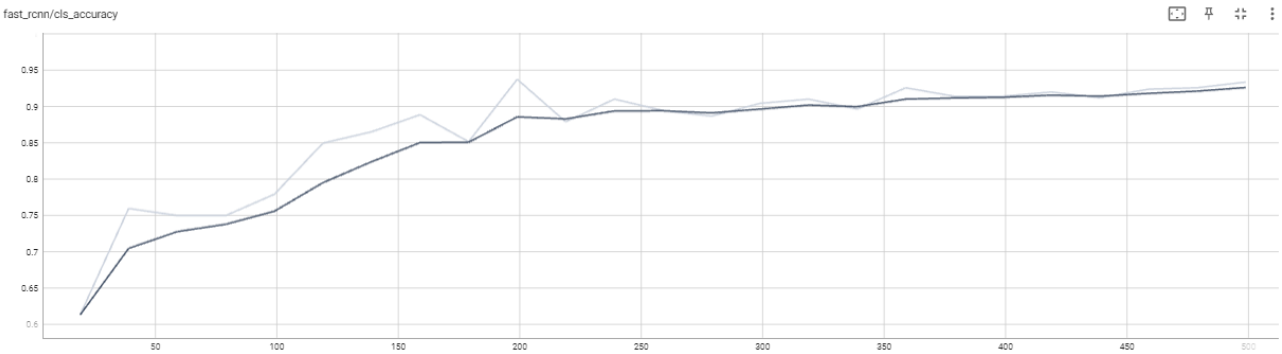


Figure 8: Plot of model accuracy as generated by the notebook.

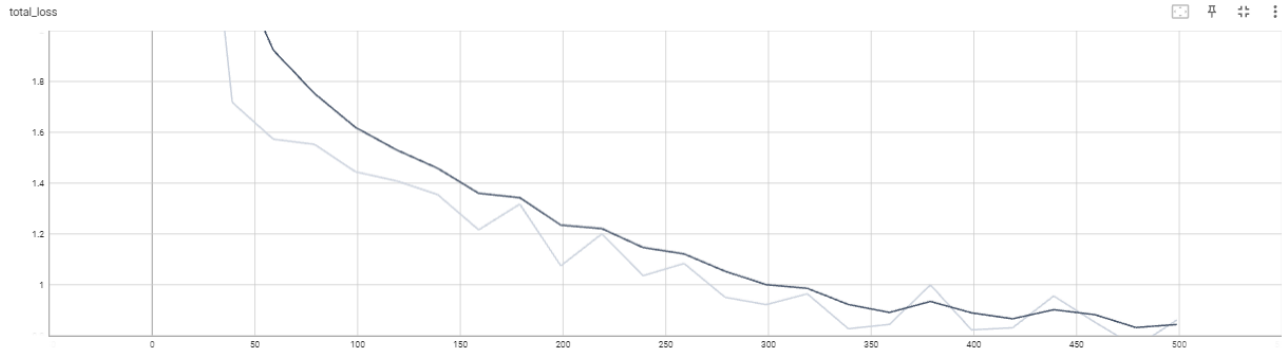


Figure 9: Plot of total loss as generated by the notebook.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 64, 128, 128] | 1,792 |
| BatchNorm2d-2 | [-1, 64, 128, 128] | 128 |
| ReLU-3 | [-1, 64, 128, 128] | 0 |
| conv-4 | [-1, 64, 128, 128] | 0 |
| Conv2d-5 | [-1, 64, 128, 128] | 36,928 |
| BatchNorm2d-6 | [-1, 64, 128, 128] | 128 |
| ReLU-7 | [-1, 64, 128, 128] | 0 |
| conv-8 | [-1, 64, 128, 128] | 0 |
| MaxPool2d-9 | [-1, 64, 64, 64] | 0 |
| down-10 | [-1, 64, 64, 64] | 0 |
| Conv2d-11 | [-1, 128, 64, 64] | 73,856 |
| BatchNorm2d-12 | [-1, 128, 64, 64] | 256 |
| ReLU-13 | [-1, 128, 64, 64] | 0 |
| conv-14 | [-1, 128, 64, 64] | 0 |
| MaxPool2d-15 | [-1, 128, 32, 32] | 0 |
| down-16 | [-1, 128, 32, 32] | 0 |
| Conv2d-17 | [-1, 128, 32, 32] | 147,584 |
| BatchNorm2d-18 | [-1, 128, 32, 32] | 256 |
| ReLU-19 | [-1, 128, 32, 32] | 0 |
| conv-20 | [-1, 128, 32, 32] | 0 |
| MaxPool2d-21 | [-1, 128, 16, 16] | 0 |
| down-22 | [-1, 128, 16, 16] | 0 |
| Conv2d-23 | [-1, 256, 16, 16] | 295,168 |
| BatchNorm2d-24 | [-1, 256, 16, 16] | 512 |
| ReLU-25 | [-1, 256, 16, 16] | 0 |
| conv-26 | [-1, 256, 16, 16] | 0 |
| MaxPool2d-27 | [-1, 256, 8, 8] | 0 |
| down-28 | [-1, 256, 8, 8] | 0 |
| Conv2d-29 | [-1, 256, 8, 8] | 590,080 |
| BatchNorm2d-30 | [-1, 256, 8, 8] | 512 |
| ReLU-31 | [-1, 256, 8, 8] | 0 |
| conv-32 | [-1, 256, 8, 8] | 0 |
| MaxPool2d-33 | [-1, 256, 4, 4] | 0 |
| down-34 | [-1, 256, 4, 4] | 0 |
| Conv2d-35 | [-1, 512, 4, 4] | 1,180,160 |
| BatchNorm2d-36 | [-1, 512, 4, 4] | 1,024 |
| ReLU-37 | [-1, 512, 4, 4] | 0 |
| conv-38 | [-1, 512, 4, 4] | 0 |
| MaxPool2d-39 | [-1, 512, 2, 2] | 0 |
| down-40 | [-1, 512, 2, 2] | 0 |
| Conv2d-41 | [-1, 512, 2, 2] | 2,359,808 |
| BatchNorm2d-42 | [-1, 512, 2, 2] | 1,024 |
| ReLU-43 | [-1, 512, 2, 2] | 0 |
| conv-44 | [-1, 512, 2, 2] | 0 |
| MaxPool2d-45 | [-1, 512, 1, 1] | 0 |
| down-46 | [-1, 512, 1, 1] | 0 |
| ConvTranspose2d-47 | [-1, 512, 2, 2] | 1,049,088 |
| Conv2d-48 | [-1, 512, 2, 2] | 2,359,808 |
| BatchNorm2d-49 | [-1, 512, 2, 2] | 1,024 |
| ReLU-50 | [-1, 512, 2, 2] | 0 |
| conv-51 | [-1, 512, 2, 2] | 0 |
| up-52 | [-1, 512, 2, 2] | 0 |
| BatchNorm2d-53 | [-1, 512, 2, 2] | 1,024 |
| ConvTranspose2d-54 | [-1, 512, 4, 4] | 1,049,088 |
| Conv2d-55 | [-1, 256, 4, 4] | 1,179,904 |

Figure 10: Model Architecture - 1.

```
          Conv2d-55          [-1, 256, 4, 4]       1,179,904
     BatchNorm2d-56          [-1, 256, 4, 4]             512
            ReLU-57          [-1, 256, 4, 4]               0
            conv-58          [-1, 256, 4, 4]               0
              up-59          [-1, 256, 4, 4]               0
     BatchNorm2d-60          [-1, 256, 4, 4]             512
  ConvTranspose2d-61          [-1, 256, 8, 8]         262,400
          Conv2d-62          [-1, 256, 8, 8]         590,080
     BatchNorm2d-63          [-1, 256, 8, 8]             512
            ReLU-64          [-1, 256, 8, 8]               0
            conv-65          [-1, 256, 8, 8]               0
              up-66          [-1, 256, 8, 8]               0
     BatchNorm2d-67          [-1, 256, 8, 8]             512
  ConvTranspose2d-68         [-1, 256, 16, 16]         262,400
          Conv2d-69         [-1, 128, 16, 16]         295,040
     BatchNorm2d-70         [-1, 128, 16, 16]             256
            ReLU-71         [-1, 128, 16, 16]               0
            conv-72         [-1, 128, 16, 16]               0
              up-73         [-1, 128, 16, 16]               0
     BatchNorm2d-74         [-1, 128, 16, 16]             256
  ConvTranspose2d-75         [-1, 128, 32, 32]          65,664
          Conv2d-76         [-1, 128, 32, 32]         147,584
     BatchNorm2d-77         [-1, 128, 32, 32]             256
            ReLU-78         [-1, 128, 32, 32]               0
            conv-79         [-1, 128, 32, 32]               0
              up-80         [-1, 128, 32, 32]               0
     BatchNorm2d-81         [-1, 128, 32, 32]             256
  ConvTranspose2d-82         [-1, 128, 64, 64]          65,664
          Conv2d-83          [-1, 64, 64, 64]          73,792
     BatchNorm2d-84          [-1, 64, 64, 64]             128
            ReLU-85          [-1, 64, 64, 64]               0
            conv-86          [-1, 64, 64, 64]               0
              up-87          [-1, 64, 64, 64]               0
     BatchNorm2d-88          [-1, 64, 64, 64]             128
  ConvTranspose2d-89         [-1, 64, 128, 128]         16,448
          Conv2d-90         [-1, 64, 128, 128]         36,928
     BatchNorm2d-91         [-1, 64, 128, 128]             128
            ReLU-92         [-1, 64, 128, 128]               0
            conv-93         [-1, 64, 128, 128]               0
              up-94         [-1, 64, 128, 128]               0
     BatchNorm2d-95         [-1, 64, 128, 128]             128
          Conv2d-96          [-1, 3, 128, 128]           1,731
     BatchNorm2d-97          [-1, 3, 128, 128]               6
            ReLU-98          [-1, 3, 128, 128]               0
            conv-99          [-1, 3, 128, 128]               0
         Conv2d-100          [-1, 1, 128, 128]              28
           conv-101          [-1, 1, 128, 128]               0
================================================================
Total params: 12,150,501
Trainable params: 12,150,501
Non-trainable params: 0
```

Figure 11: Model Architecture - 2.

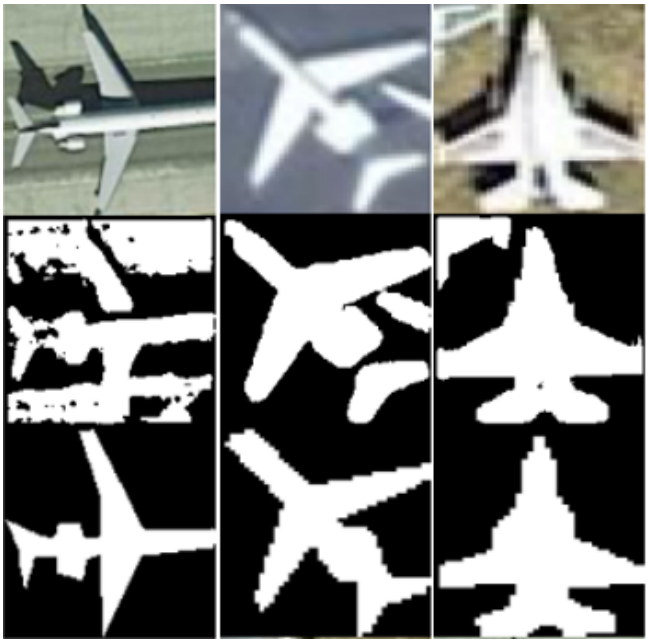

Figure 12: Example visualizations of masks from the test set.

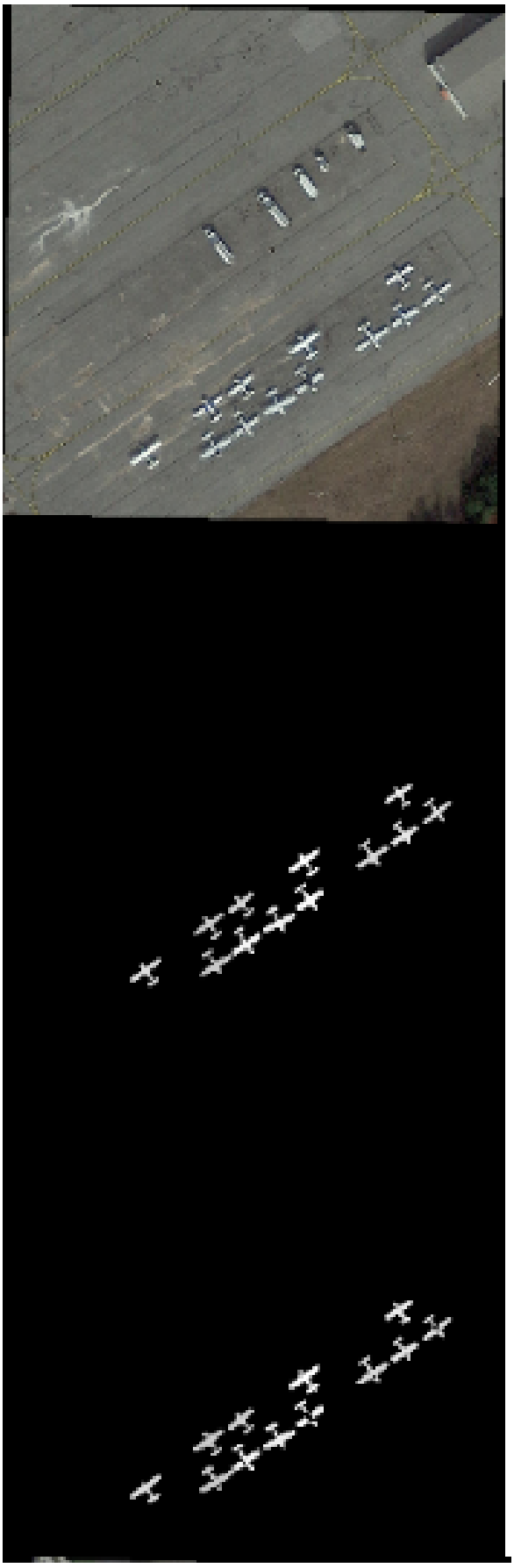

Figure 13: Example visualizations of masks.

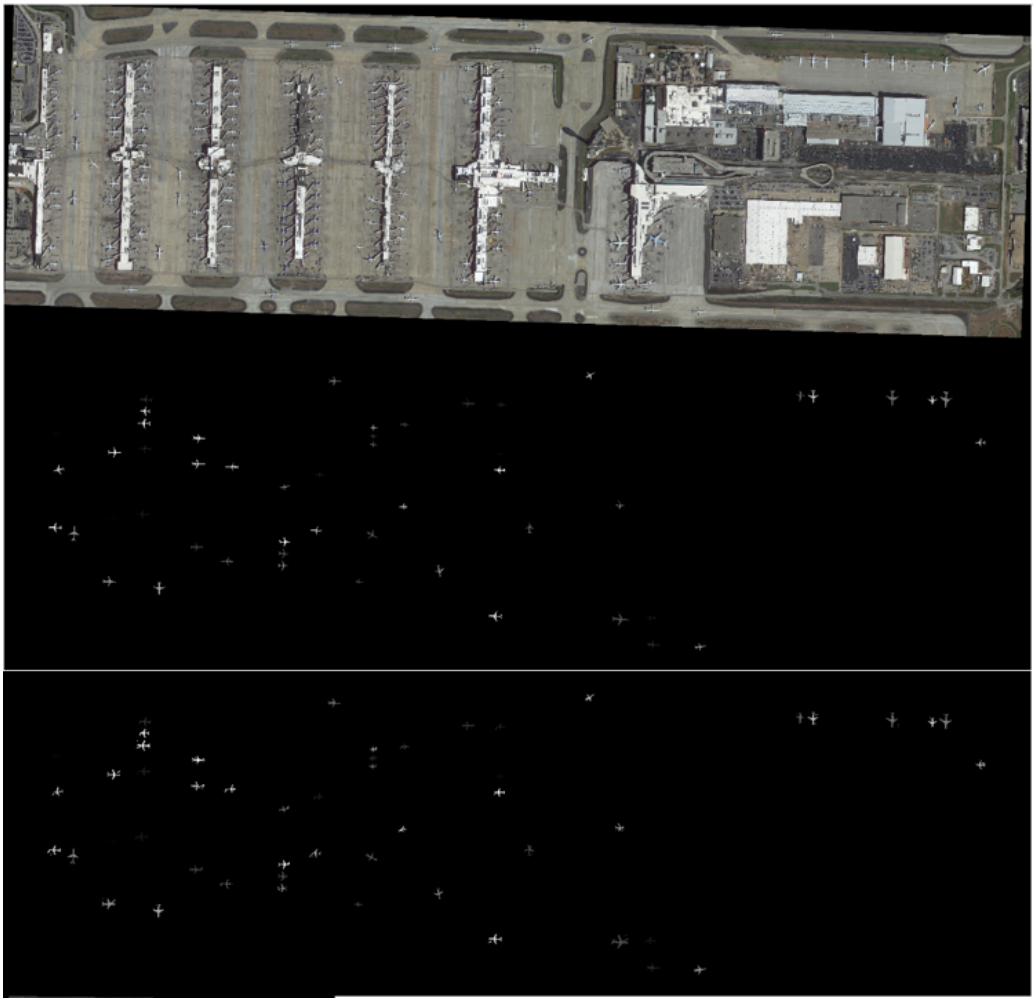

Figure 14: Example visualizations of masks.

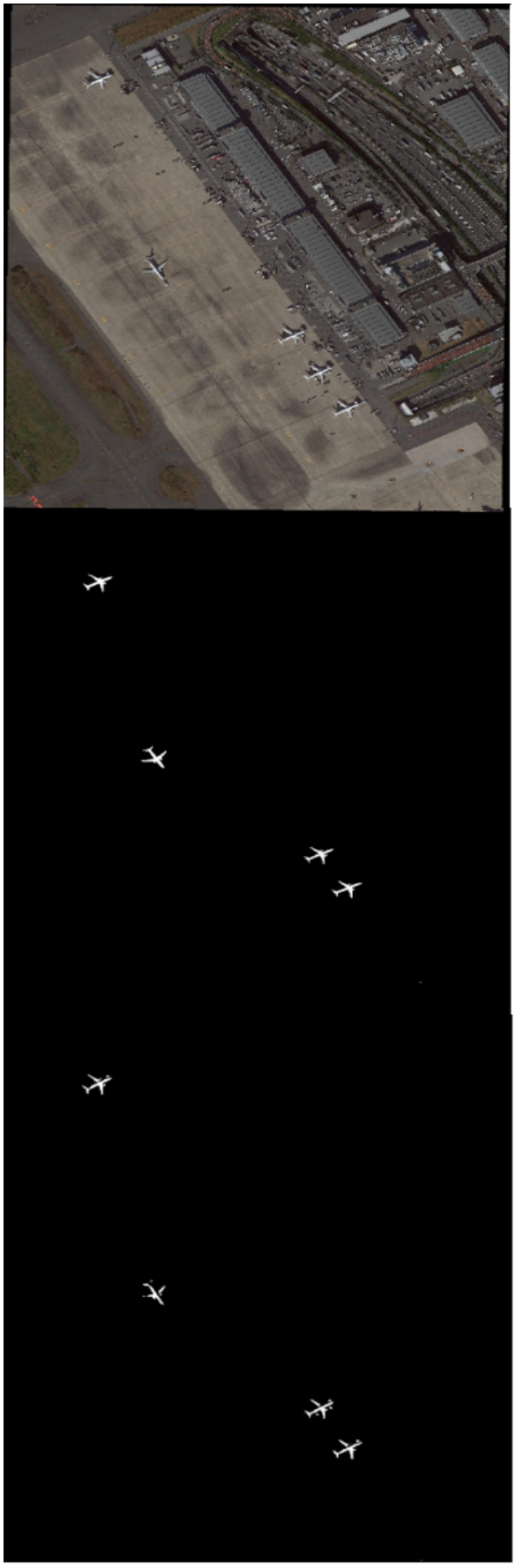

Figure 15: Example visualizations of masks.

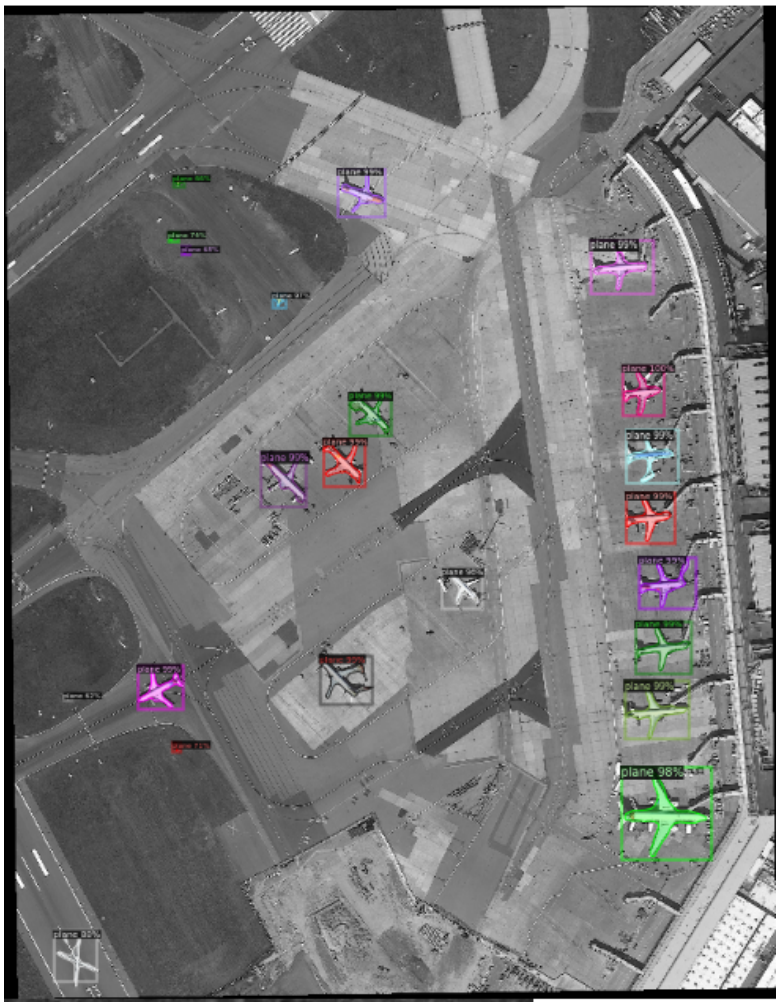

Figure 16: Visualization of predictions on `test_set[20]` using modified configurations.

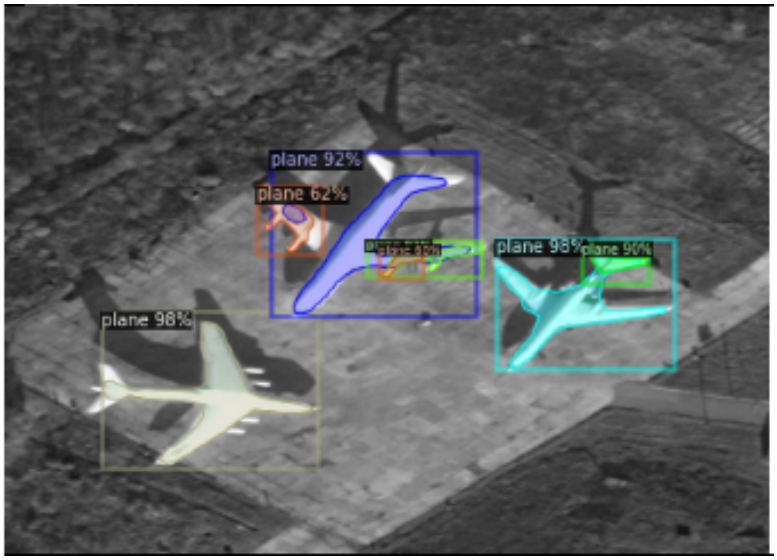


Figure 17: Visualization of predictions on `test_set[30]` using Mask R-CNN configurations.


Figure 18: Visualization of predictions on `test_set[70]` using Mask R-CNN configurations.