CMPT412 Project 3
Digit recognition with convolutional neural networks

Student: Sibei Zhou(301416917)
Computer ID: palamzad@sfu.ca
Team members: Sibei Zhou, Honghao Yu, ChenKun Zhang
Kaggle entry name: Sibei Zhou
(use 3 free-late days)

- Part 1:
  - List of the configs and modifications that you used.

(1) Default configs:

```
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url
     ("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")

cfg.SOLVER.IMS_PER_BATCH = 2

cfg.SOLVER.BASE_LR = 0.00025 #0.00025->0.0005->0.001->0.0005(final)

cfg.SOLVER.MAX_ITER = 500   # 500->1000->5000(final)

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512

cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
```

(2) Final configs: (Modified the `MAX_ITER`, `BASE_LR` and change the default model to `faster_rcnn_X_101_32x8d_FPN_3x.yaml`)

```
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url

("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml")

cfg.SOLVER.IMS_PER_BATCH = 2

cfg.SOLVER.BASE_LR = 0.0005

cfg.SOLVER.MAX_ITER = 5000

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512

cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
```

- Factors which helped improve the performance. Explain each factor in 2-3 lines.
  - MAX_ITER

    Since the loss is still decreasing rapidly when max_iter = 500, the higher iterations might improve the model and get higher accuracy, so we keep trying on more iteration until total_loss becomes stable.

  - BASE_LR

    We have modified the learning rate a bit higher (0.0005) and higher (0.001) than the original (0.00025), the accuracy of lr=0.0005 has increased but another with lr=0.001 got a very low accuracy. So we choose a learning rate 0.0005 at the end.

    Result compared with different learning rate at `MAX_ITER = 500`:

    `BASE_LR = 0.00025(default):`

    ```
    [03/01 04:04:23 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
    |   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
    |:------:|:------:|:------:|:------:|:------:|:------:|
    | 29.045 | 50.717 | 30.527 | 20.255 | 36.919 | 59.305 |
    ```

    `BASE_LR = 0.0005(best):`

    ```
    [03/01 05:34:18 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
    |   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
    |:------:|:------:|:------:|:------:|:------:|:------:|
    | 32.265 | 55.007 | 35.175 | 22.874 | 40.727 | 59.979 |
    ```

    `BASE_LR = 0.001(worse):`

    ```
    [03/01 05:00:23 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
    |  AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
    |:-----:|:------:|:------:|:-----:|:------:|:------:|
    | 5.088 | 13.425 | 2.088  | 6.700 | 10.433 | 11.504 |
    ```

  - MODEL.WEIGHTS

    On the Model_ZOO.md of Detectron2 we found in github, it introduces the X101-FPN model has a slightly higher box AP compared to the original R101-FPN model (picture below), so we modified the model to use the X101-FPN.

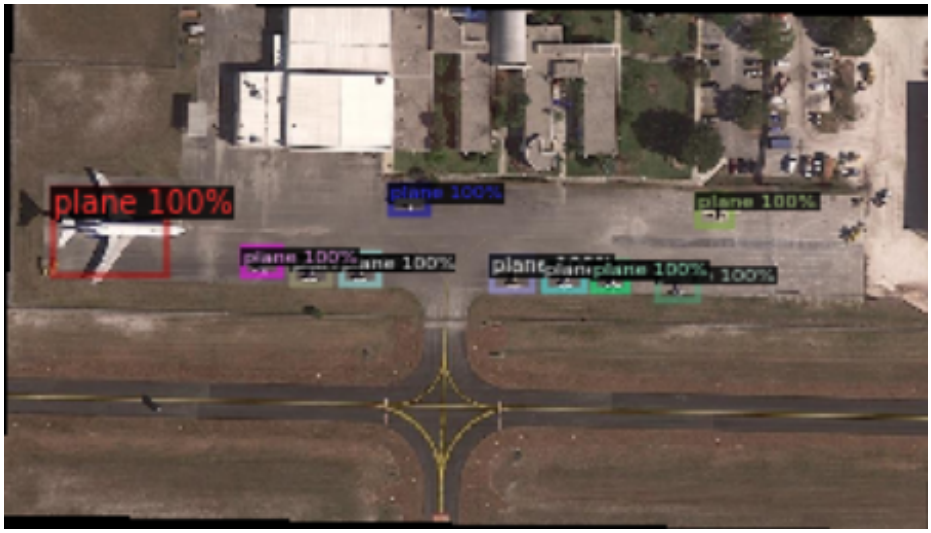
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

- Final plot for total training loss and accuracy. This would have been auto-generated by the notebook.

```
[03/01 10:49:08 d2.utils.events]:  eta: 0:03:57  iter: 4519  total_loss: 0.4471
[03/01 10:49:19 d2.utils.events]:  eta: 0:03:47  iter: 4539  total_loss: 0.4143
[03/01 10:49:30 d2.utils.events]:  eta: 0:03:36  iter: 4559  total_loss: 0.4004
[03/01 10:49:41 d2.utils.events]:  eta: 0:03:26  iter: 4579  total_loss: 0.4839
[03/01 10:49:53 d2.utils.events]:  eta: 0:03:16  iter: 4599  total_loss: 0.3368
[03/01 10:50:04 d2.utils.events]:  eta: 0:03:06  iter: 4619  total_loss: 0.3388
[03/01 10:50:16 d2.utils.events]:  eta: 0:02:56  iter: 4639  total_loss: 0.4339
[03/01 10:50:28 d2.utils.events]:  eta: 0:02:47  iter: 4659  total_loss: 0.3659
[03/01 10:50:40 d2.utils.events]:  eta: 0:02:37  iter: 4679  total_loss: 0.4107
[03/01 10:50:51 d2.utils.events]:  eta: 0:02:27  iter: 4699  total_loss: 0.3912
[03/01 10:51:02 d2.utils.events]:  eta: 0:02:18  iter: 4719  total_loss: 0.4032
[03/01 10:51:13 d2.utils.events]:  eta: 0:02:08  iter: 4739  total_loss: 0.3949
[03/01 10:51:24 d2.utils.events]:  eta: 0:01:58  iter: 4759  total_loss: 0.4526
[03/01 10:51:36 d2.utils.events]:  eta: 0:01:48  iter: 4779  total_loss: 0.3394
[03/01 10:51:46 d2.utils.events]:  eta: 0:01:38  iter: 4799  total_loss: 0.4053
[03/01 10:51:57 d2.utils.events]:  eta: 0:01:28  iter: 4819  total_loss: 0.4539
[03/01 10:52:09 d2.utils.events]:  eta: 0:01:19  iter: 4839  total_loss: 0.4308
[03/01 10:52:20 d2.utils.events]:  eta: 0:01:09  iter: 4859  total_loss: 0.3203
[03/01 10:52:32 d2.utils.events]:  eta: 0:00:59  iter: 4879  total_loss: 0.3234
[03/01 10:52:43 d2.utils.events]:  eta: 0:00:49  iter: 4899  total_loss: 0.3849
[03/01 10:52:55 d2.utils.events]:  eta: 0:00:39  iter: 4919  total_loss: 0.3821
[03/01 10:53:06 d2.utils.events]:  eta: 0:00:29  iter: 4939  total_loss: 0.2878
[03/01 10:53:16 d2.utils.events]:  eta: 0:00:19  iter: 4959  total_loss: 0.3816
[03/01 10:53:27 d2.utils.events]:  eta: 0:00:09  iter: 4979  total_loss: 0.3736
[03/01 10:53:41 d2.utils.events]:  eta: 0:00:00  iter: 4999  total_loss: 0.379
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.486
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.695
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.561
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.387
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.553
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.821
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.020
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.164
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.511
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.392
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.587
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.870
[03/01 10:55:28 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 48.584 | 69.491 | 56.087 | 38.702 | 55.279 | 82.125 |
```

- The visualization of 3 samples from the test set and the predicted results.
  - Sample test 1



  - Sample test 2

- Sample test 3

- At least one ablation study to validate the above choices, i.e., a comparison of performance for two variants of a model, one with and one without a certain feature or implementation choice. In addition, provide visualization of a sample from the test set for qualitative comparison.
  - Model with `MAX_ITER = 1000`

    With the lower iterations, the training process only takes 10 mins, and the result looks great with only a few plane missed (left figure):

  - Model with `MAX_ITER = 5000`

    When the iterations increase, the training process takes over 30 mins, but it got lower total_loss and the result looks greater than before (right figure):

- Part 2:
  - Report any hyperparameter settings you used (batch_size, learning_rate, num_epochs, optimizer).

```
num_epochs = 50

batch_size = 8

learning_rate = 0.05     # 0.01 -> 0.05

weight_decay = 1e-5

optim = torch.optim.Adam(model.parameters(), lr=learning_rate,
weight_decay=weight_decay)
```

  - 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, 8)
self.down_1 = down(8, 16)
self.down_2 = down(16, 32)
self.down_3 = down(32, 64)
self.down_4 = down(64, 128)
self.down_5 = down(128, 256)
self.down_6 = down(256, 512)

# Decoder

self.up_1 = up(512, 256)
self.up_2 = up(256, 128)
self.up_3 = up(128, 64)
self.up_4 = up(64, 32)
self.up_5 = up(32, 16)
self.up_6 = up(16, 8)
self.up_7 = conv(8, 3)
self.output_conv = conv(3, 1, False)

self.normal_1 = nn.BatchNorm2d(8)
self.normal_2 = nn.BatchNorm2d(16)
self.normal_3 = nn.BatchNorm2d(32)
self.normal_4 = nn.BatchNorm2d(64)
self.normal_5 = nn.BatchNorm2d(128)
self.normal_6 = nn.BatchNorm2d(256)
```

```
def forward(self, input):
    y_1 = self.input_conv(input)
    y_2 = self.down_1(y_1)
    y_3 = self.down_2(y_2)
    y_4 = self.down_3(y_3)
    y_5 = self.down_4(y_4)
    y_6 = self.down_5(y_5)
    y_7 = self.down_6(y_6)
    y_7 = self.normal_6(F.relu(self.up_1(y_7) + y_6))
    y_6 = self.normal_5(F.relu(self.up_2(y_7) + y_5))
    y_5 = self.normal_4(F.relu(self.up_3(y_6) + y_4))
    y_4 = self.normal_3(F.relu(self.up_4(y_5) + y_3))
    y_3 = self.normal_2(F.relu(self.up_5(y_4) + y_2))
    y_2 = self.normal_1(F.relu(self.up_6(y_3) + y_1))
    y = self.up_7(y_2)
    output = self.output_conv(y)
    return output
```

The model's architecture we used references the RNN from the last project, and we also add some skip layers that references the resnet on each decoder layer, which helps in improving accuracy a lot.

- Report the loss functions that you used and the plot the total training loss of the training procedure

  ```
  crit = nn.BCEWithLogitsLoss() # Define the loss function
  ```

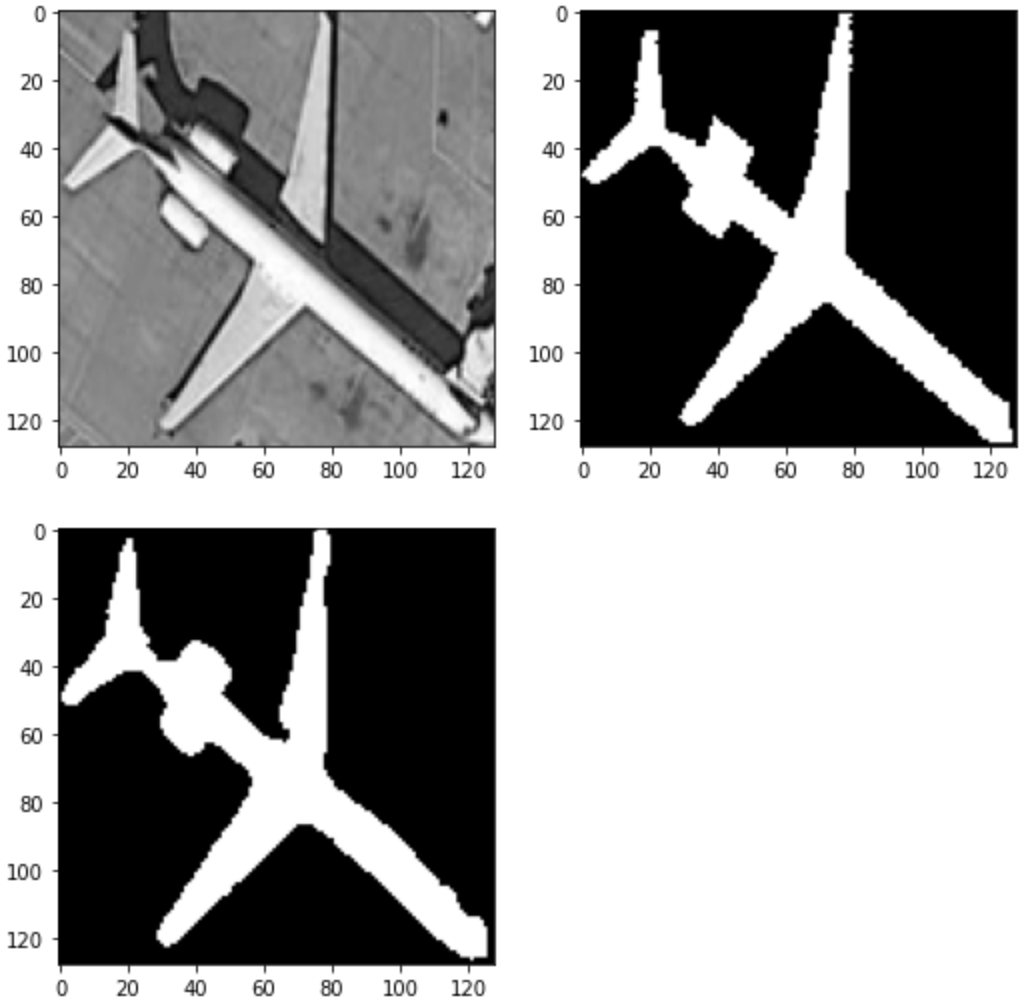
  ```
  Loss: 0.050518546253442764
  ```

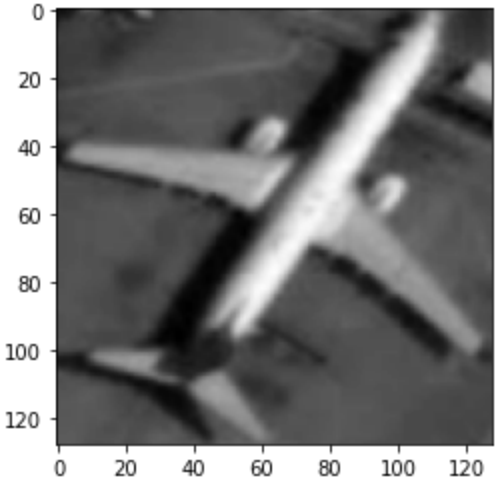
- Report the final mean IoU of your model

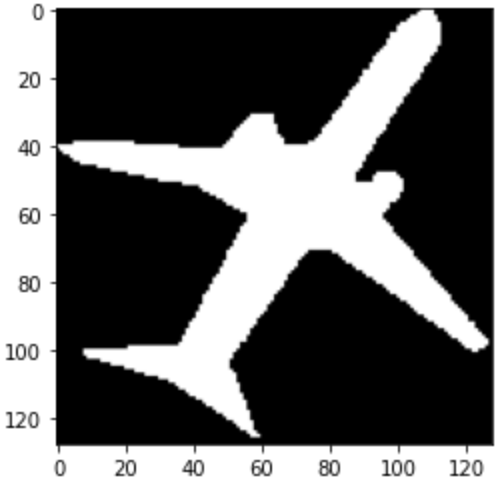
  Mean IoU = `0.8718697733076178`

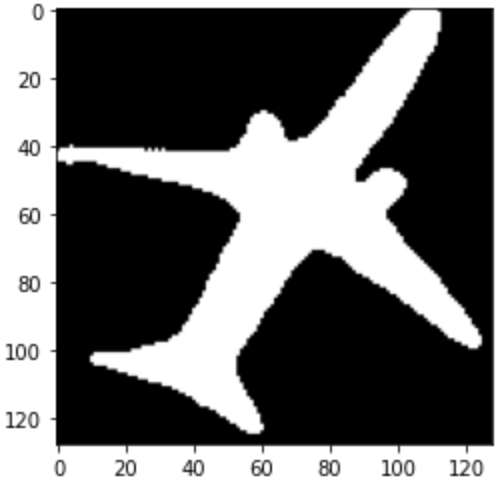
- Visualize 3 images from the test set and the corresponding predicted masks

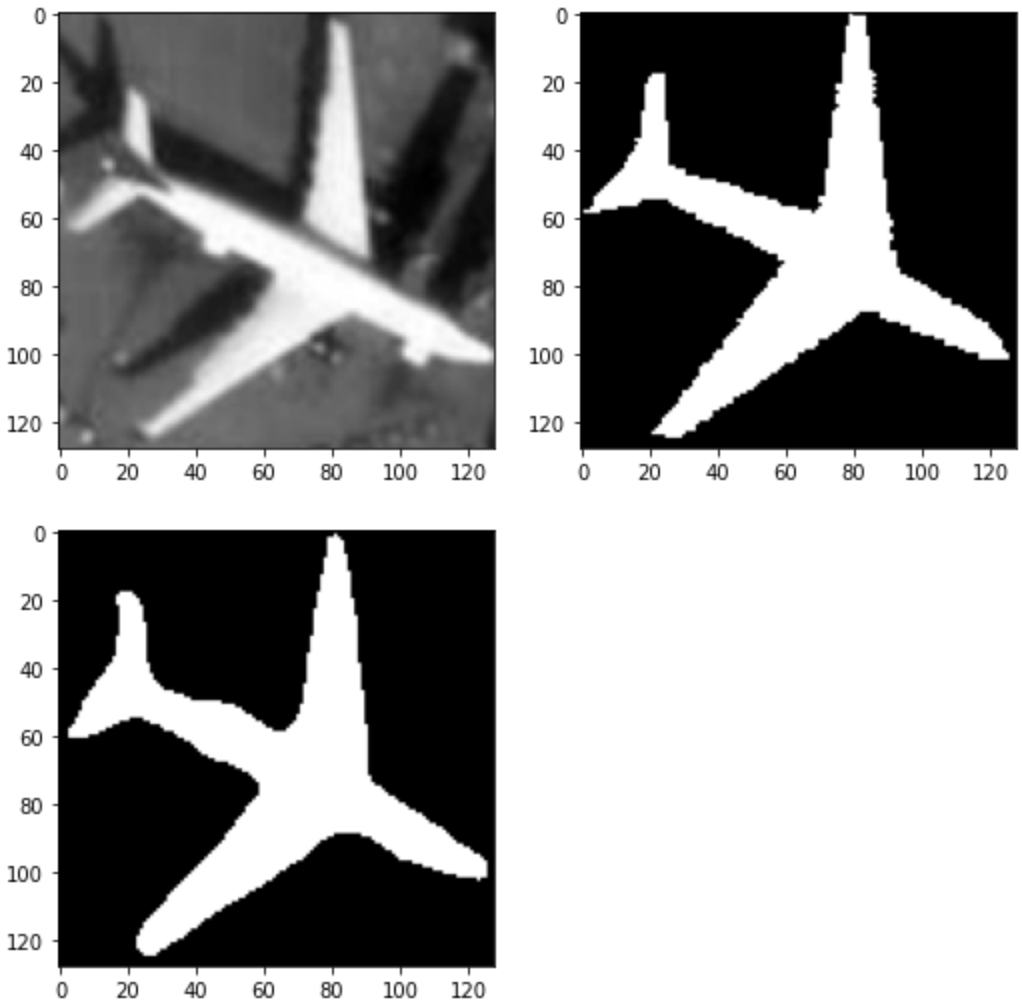
  (on the right of the original picture is the mask picture, and the one below it is the prediction mask)




0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120



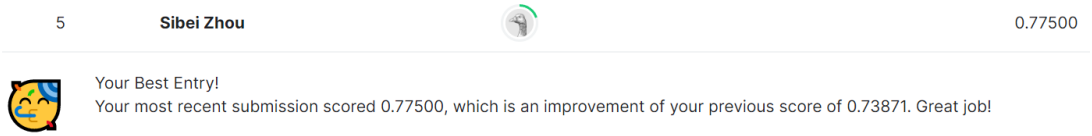
- Part 3:
  - The name under which you submitted on Kaggle. Names of your group members if you form a group.

    Name on Kaggle: Sibei Zhou

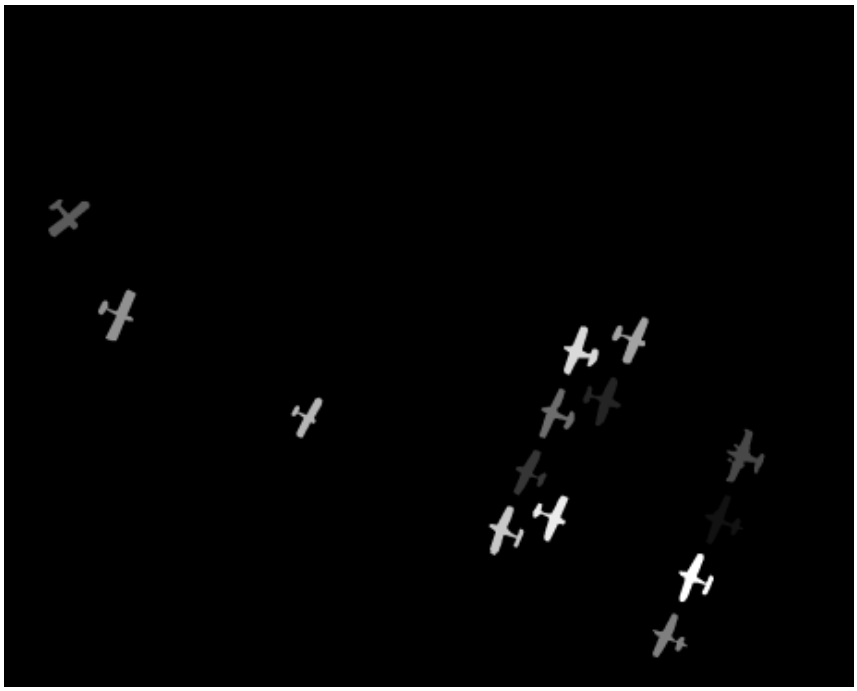
    Groups Members: Sibei Zhou, Honghao Yu, Chenkun Zhang

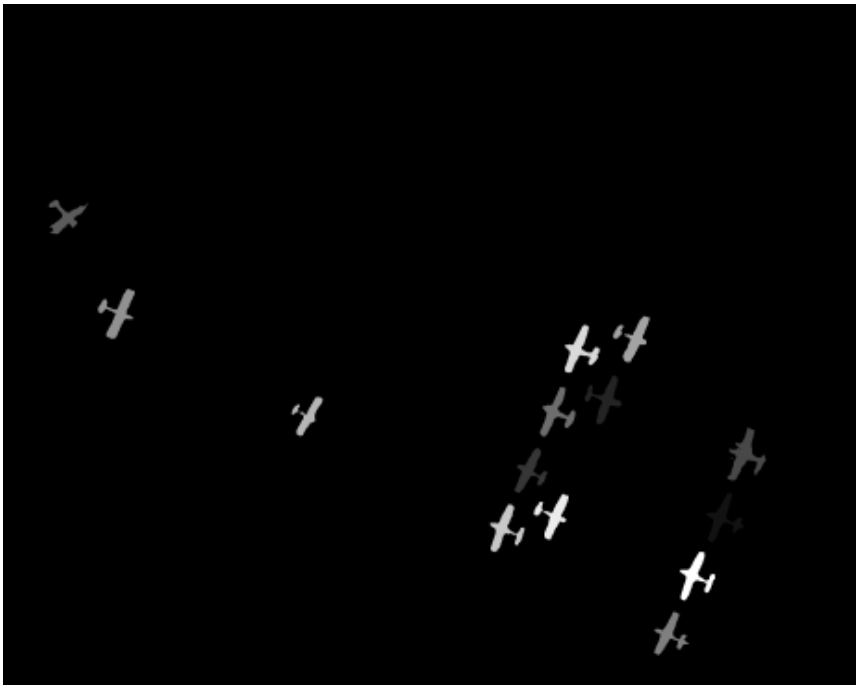
  - Report the best score (should match your score on Kaggle).

    Best score on kaggle: 0.775 (# 5 in Leaderboard)



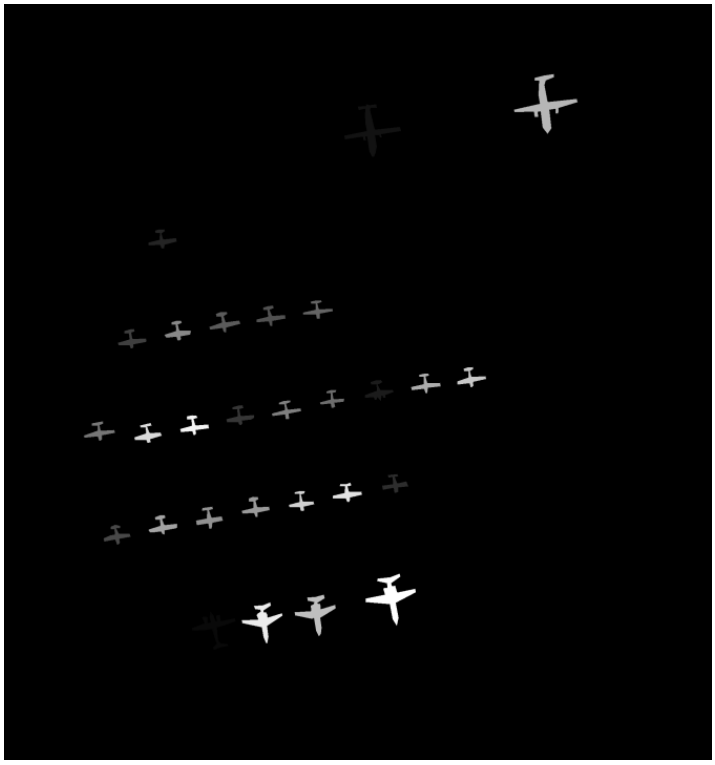
- The visualization of results for 3 random samples from the test set. (The first: original, the second: true mask, the third: predict mask)
  - Sample 1:

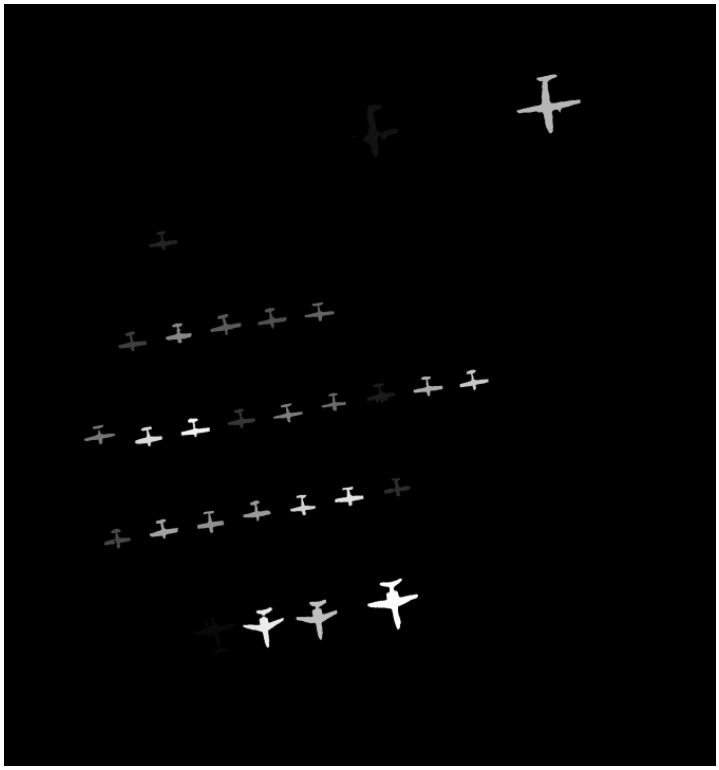




- Sample 2:

- Sample 3:





- CSV file of your predicted test labels needs to be uploaded to Kaggle.

- Part 4:
  - The visualization and the evaluation results similar to Part 1
    The hyperparameter used is the same as in Part 1, which are:

```
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.0005
cfg.SOLVER.MAX_ITER = 5000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
```

    visualization of 3 random samples:

- Explain the differences between the results of Part 3 and Part 4 in a few lines.

  The result of Part 4 got lower accuracy than the result in part3 in both bbox AP:

  Part 3 bbox AP:

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.486
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.695
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.561
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.387
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.553
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.821
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.020
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.164
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.511
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.392
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.587
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.870
[03/01 10:55:28 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 48.584 | 69.491 | 56.087 | 38.702 | 55.279 | 82.125 |
```

Part 4 bbox AP: 44.929

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.449
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.672
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.517
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.355
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.520
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.758
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.018
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.155
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.479
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.362
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.557
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.818
[03/02 11:21:43 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 44.929 | 67.197 | 51.741 | 35.464 | 51.965 | 75.786 |
```

And its performance in high-precision pictures is also worse than Part 3, in some high-precision pictures, it segmentation looks bad, for example: