Project3 Report

Rongsheng Qian 301449387

Team: R.S Qian&Yuwen Jia&Isaac Ding

Names of group members: Rongsheng Qian, Isaac Ding, Yuwen Jia SFU

Using 3 free late day

Best accuracy: 0.42208

Part 1

1.1 List of the configs and modifications that you used.

1.1.1 Configs

```
cfg = get_cfg()
cfg.OUTPUT_DIR = "{}/output_custom/".format(BASE_DIR)

cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_101_FPN_3x.yaml"))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_101_FPN_3x.yaml")

cfg.DATASETS.TRAIN = ("custom_train",)
cfg.DATASETS.TEST = ("custom_val",)
cfg.DATALOADER.NUM_WORKERS = 2

cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.0001
cfg.SOLVER.MAX_ITER = 1000

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 5
```

1.1.2 Create 3 Dataloader (Data augmentation) (train 3 times using 3 CustomTrainer3)

```
class CustomDatasetMapper1(DatasetMapper):
    def __init__(self, cfg, is_train=True):
        super().__init__(cfg, is_train)
        augmentations = []
        if is_train:
            augmentations.append(T.Resize((800, 800)))
            augmentations.append(T.RandomRotation([-5,5]))
        self.augmentations = T.AugmentationList(augmentations)

    def __call__(self, dataset_dict):
        dataset_dict = super().__call__(dataset_dict)
        return dataset_dict


class CustomDatasetMapper2(DatasetMapper):
    def __init__(self, cfg, is_train=True):
        super().__init__(cfg, is_train)
        augmentations = []
        if is_train:
            augmentations.append(T.Resize((800, 800)))
            augmentations.append(T.RandomFlip(prob=1))
        self.augmentations = T.AugmentationList(augmentations)
```

```
    def __call__(self, dataset_dict):
        dataset_dict = super().__call__(dataset_dict)
        return dataset_dict

class CustomDatasetMapper3(DatasetMapper):
    def __init__(self, cfg, is_train=True):
        super().__init__(cfg, is_train)
        augmentations = []
        if is_train:
            augmentations.append(T.Resize((800, 800)))

        self.augmentations = T.AugmentationList(augmentations)

    def __call__(self, dataset_dict):
        dataset_dict = super().__call__(dataset_dict)
        return dataset_dict


class CustomTrainer1(DefaultTrainer):
    @classmethod
    def build_train_loader(cls, cfg):
        return build_detection_train_loader(cfg, mapper=CustomDatasetMapper1(cfg, is_train=True))

class CustomTrainer2(DefaultTrainer):
    @classmethod
    def build_train_loader(cls, cfg):
        return build_detection_train_loader(cfg, mapper=CustomDatasetMapper2(cfg, is_train=True))

class CustomTrainer3(DefaultTrainer):
    @classmethod
    def build_train_loader(cls, cfg):
        return build_detection_train_loader(cfg, mapper=CustomDatasetMapper3(cfg, is_train=True))
```

1.1.3 Data Processing.

Divide an image into smaller blocks (800*800) if the bounding box is less than 50% of its total area in this block, I drop it

```
def is_bbox_in_area(bbox_xywh,area_xyxy):
    bbox_x1 = bbox_xywh[0]
    bbox_y1 = bbox_xywh[1]
    bbox_x2 = bbox_xywh[0] + bbox_xywh[2]
    bbox_y2 = bbox_xywh[1] + bbox_xywh[3]

    bbox = box(bbox_x1, bbox_y1, bbox_x2, bbox_y2)
    area = box(area_xyxy[0], area_xyxy[1], area_xyxy[2], area_xyxy[3])
```

```
    intersect = bbox.intersection(area)

    intersect_area = intersect.area
    bbox_area = bbox.area
    threshold_area = 0.5 * bbox_area

    if threshold_area > intersect_area:
        return []

    if not intersect.is_empty:
        x1, y1, x2, y2 = intersect.bounds
        w = x2 - x1
        h = y2 - y1
        return [x1,y1,w,h]
    else:
        return []
```

1.1.4 Add validation set

Set 19 images as validation set which is 10% of whole training set

1.2 Factors which helped improve the performance.

1.2.1 Increase the size of val set from 8 images to 19 images

The val accuracy increased from baseline 23% to 28%

```
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 28.494 | 51.242 | 29.399 | 13.506 | 37.452 | 53.231 |
[11/09 21:05:59 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP   | category   | AP     | category   | AP   |
|:-----------|:-----|:-----------|:-------|:-----------|:-----|
|            | nan  |            | nan    |            | nan  |
|            | nan  | plane      | 28.494 |            |      |
OrderedDict([('bbox',
              {'AP': 28.49429523609384,
               'AP50': 51.24230960124314,
               'AP75': 29.398519989052847,
               'APs': 13.505629791280432,
               'APm': 37.45163273304162,
               'APl': 53.23101583029035,
               'AP-': nan,
               'AP-plane': 28.49429523609384})])
```

1.2.2 Data Processing.

The val accuracy increased from baseline 29% to 43%

1.2.2.1 if the bounding box is less than 50% of its total area in this block, I drop it. Comparing with the bounding box is less than 30% of its total area in this block

less than 50%

less than 30%

This factor can increase the accuracy from 26% to 31%.

1.2.3 Using `retinanet_R_101_FPN_3x.yaml` model and pre-train instead of `faster_rcnn_R_101_FPN_3x.yaml`

This factors can increase val accuracy from 60% to 63%

1.2.4 Create 3 Dataloader (Data augmentation) (original images, filping images, rotation images)

This factors can increase val accuracy from 63% to 67%

```
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 63.779 | 88.482 | 77.008 | 69.716 |  nan  |  nan  |
[11/11 06:37:59 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/11 06:37:59 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 63.779 |            |       |
OrderedDict([('bbox',
              {'AP': 63.77916105122618,
               'AP50': 88.48155756625302,
               'AP75': 77.0081022631427,
               'APs': 69.71618395818514,
               'APm': nan,
               'APl': nan,
               'AP-': nan,
               'AP-plane': 63.77916105122618})])
```

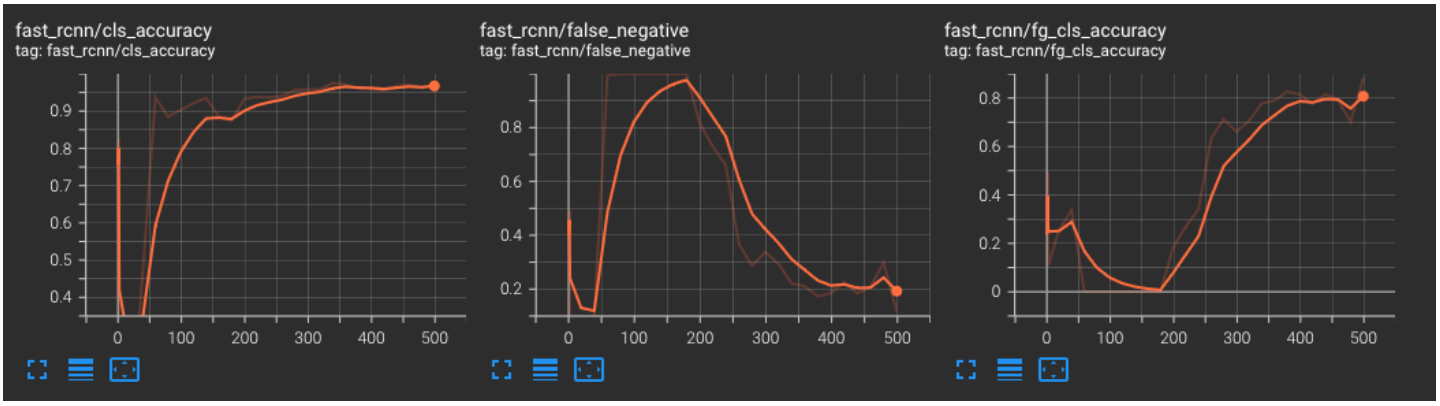
```
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 66.923 | 91.258 | 82.316 | 72.591 |  nan  |  nan  |
[11/09 22:49:24 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/09 22:49:24 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 66.923 |            |       |
OrderedDict([('bbox',
              {'AP': 66.92288550535133,
               'AP50': 91.2584717138306,
               'AP75': 82.31615013389118,
               'APs': 72.59099451522125,
               'APm': nan,
               'APl': nan,
               'AP-': nan,
               'AP-plane': 66.92288550535133})])
```

1.3 Final plot for total training loss and accuracy.

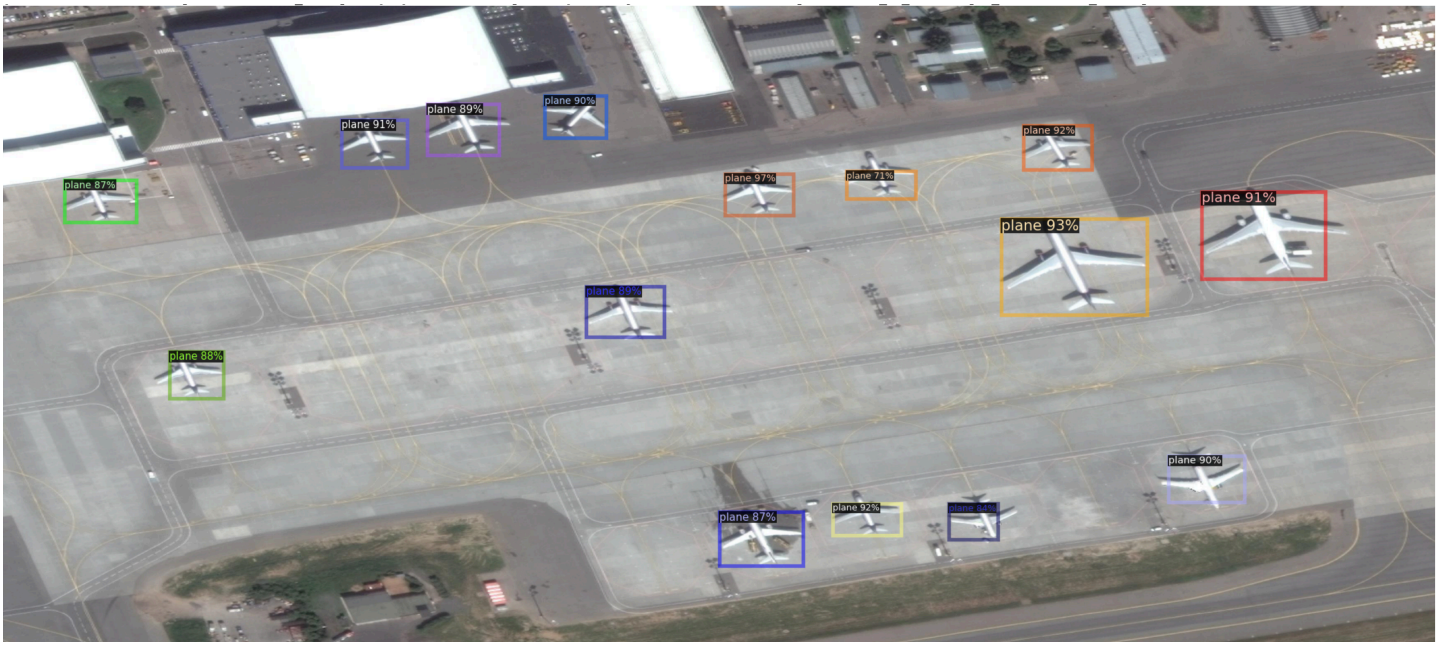
1.3.1 final accuracy:

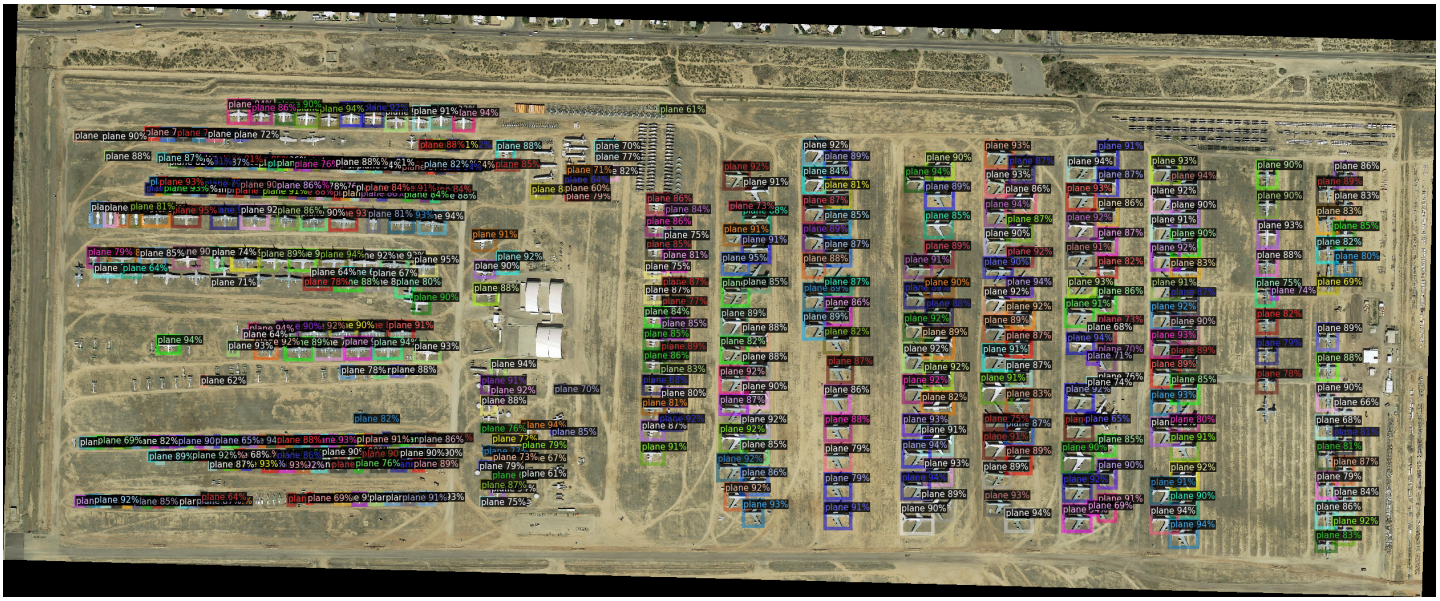

```
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 66.923 | 91.258 | 82.316 | 72.591 |  nan  |  nan  |
[11/09 22:49:24 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/09 22:49:24 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 66.923 |            |       |
OrderedDict([('bbox',
              {'AP': 66.92288550535133,
               'AP50': 91.2584717138306,
               'AP75': 82.31615013389118,
               'APs': 72.59099451522125,
               'APm': nan,
               'APl': nan,
               'AP-': nan,
               'AP-plane': 66.92288550535133})])
```


1.3.2 Plot



1.4 The visualization of 3 samples from the test set and the predicted results.





1.5 Ablation Study

1.5.1 Divide an image into smaller blocks (800*800) if the bounding box is less than 50% of its total area in this block drop it

This factor can increase the accuracy from 28% to 64%.

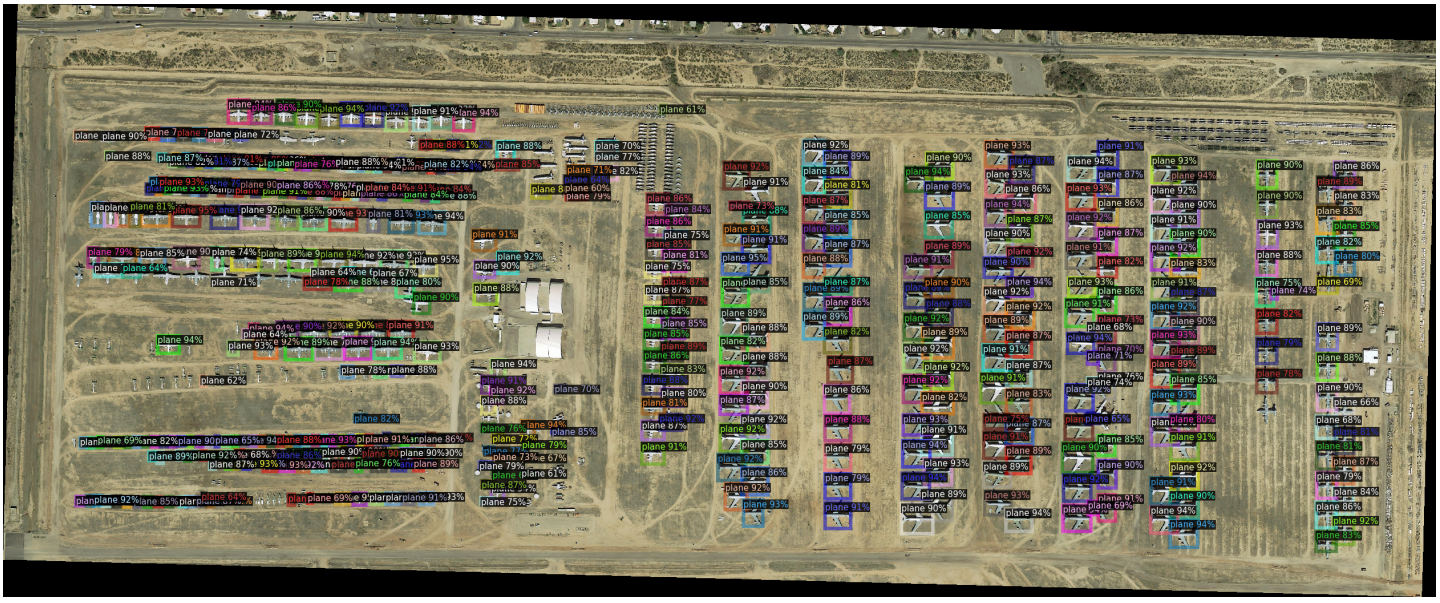
The small plane can be detected more easily. The visualization is shown below. The origin model can not detect the small plane in this sample, however the model after adding this factor can detect it with a acceptable precision.

Droping the bounding box which is less than 50% of its total area can help model to recognize how the plane looks like rather than predicted a building as a plane.

Without the Data Processing:

```
[11/09 21:05:59 d2.evaluation.coco_evaluation]: Evaluation results fo
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 28.494 | 51.242 | 29.399 | 13.506 | 37.452 | 53.231 |
[11/09 21:05:59 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 28.494 |            |       |
OrderedDict([('bbox',
              {'AP': 28.49429523609384,
               'AP50': 51.24230960124314,
               'AP75': 29.398519989052847,
               'APs': 13.505629791280432,
               'APm': 37.45163273304162,
               'APl': 53.23101583029035,
               'AP-': nan,
               'AP-plane': 28.49429523609384})])
```

With the Data Processing:



```
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 63.779 | 88.482 | 77.008 | 69.716 |  nan  |  nan  |
[11/11 06:37:59 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/11 06:37:59 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 63.779 |            |       |
OrderedDict([('bbox',
              {'AP': 63.77916105122618,
               'AP50': 88.48155756625302,
               'AP75': 77.0081022631427,
               'APs': 69.71618395818514,
               'APm': nan,
               'APl': nan,
               'AP-': nan,
               'AP-plane': 63.77916105122618})])
```

1.5.2 Data augmentation

The image visualization result doesn't have the obvious difference.

However this factors can increase val accuracy from 63% to 67%

Data augmentation can increase the training set and let the network learn more features, which can increase the subtability of model in different kinds of images.

```
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 63.779 | 88.482 | 77.008 | 69.716 |  nan  |  nan  |
[11/11 06:37:59 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/11 06:37:59 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 63.779 |            |       |
OrderedDict([('bbox',
              {'AP': 63.77916105122618,
               'AP50': 88.48155756625302,
               'AP75': 77.0081022631427,
               'APs': 69.71618395818514,
               'APm': nan,
               'APl': nan,
               'AP-': nan,
               'AP-plane': 63.77916105122618})])
```

```
|   AP   |  AP50  |  AP75  |  APs   |  APm  |  APl  |
|:------:|:------:|:------:|:------:|:-----:|:-----:|
| 66.923 | 91.258 | 82.316 | 72.591 |  nan  |  nan  |
[11/09 22:49:24 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/09 22:49:24 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP    | category   | AP     | category   | AP    |
|:-----------|:------|:-----------|:-------|:-----------|:------|
|            | nan   |            | nan    |            | nan   |
|            | nan   | plane      | 66.923 |            |       |
OrderedDict([('bbox',
              {'AP': 66.92288550535133,
               'AP50': 91.2584717138306,
               'AP75': 82.31615013389118,
               'APs': 72.59099451522125,
               'APm': nan,
               'APl': nan,
               'AP-': nan,
               'AP-plane': 66.92288550535133})])
```

Part 2

2.1 Any hyperparameter settings I used

```
num_epochs = 70
batch_size = 4
learning_rate = 0.01
weight_decay = 1e-5
```

2.2 The final architecture of my network

2.2.1 Modification

Increasing channels layers using conv2d: (3,16), (16, 32), (64,128), (128,256), (256,512)

Repeated layers (in_channel == out_channel) using conv2d after (16,32), (32,64), (64,128), (128,256), there isn't repeated layer after (3,16)

Added fc layer between the encoder and decoder

2.2.2 explain the reason for each modification

Repeating the CONV with the same number of input channels and output channels

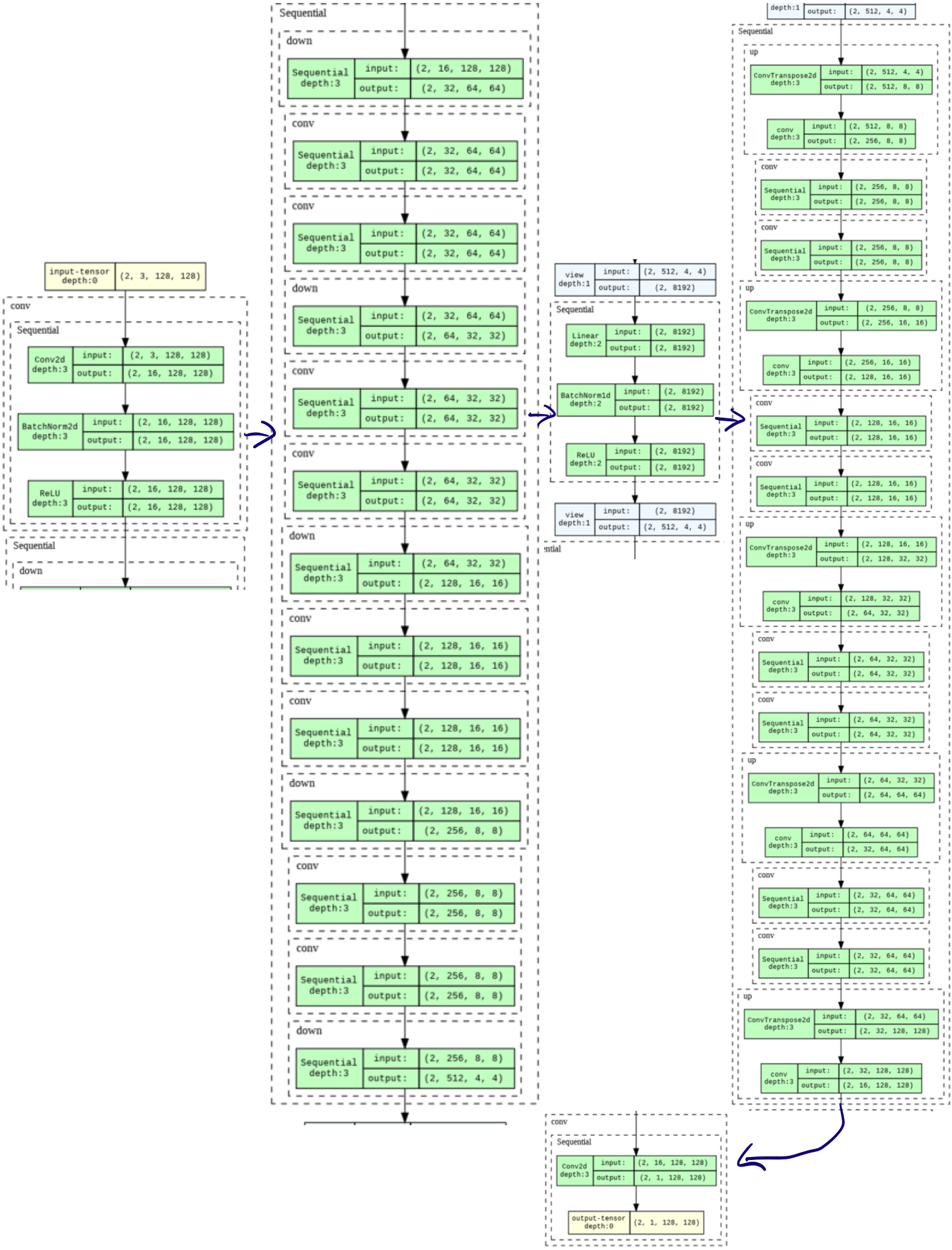
Each repeating convolutional layer can capture different features, which can let network learn more complex features. It works when the train data is limited, which can mitigate overfitting issues

It can increase the depth of the network and add more non-linear part in network.

2.2.3 Final architecture

```
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()

        # Encoder

        self.input_conv = conv(3, 16)
        self.down = nn.Sequential(
            down(16, 32),
            conv(32, 32),
            conv(32, 32),

            down(32, 64),
            conv(64, 64),
            conv(64, 64),

            down(64,128),
            conv(128, 128),
            conv(128, 128),

            down(128,256),
            conv(256, 256),
            conv(256, 256),

            down(256,512)
        )

        self.fc_net = nn.Sequential(
            nn.Linear(512*4*4, 512*4*4),
            nn.BatchNorm1d(512*4*4),
            nn.ReLU(inplace=True)
        )

```

```
        # Decoder

        self.up = nn.Sequential(
            up(512, 256),
            conv(256, 256),
            conv(256, 256),

            up(256,128),
            conv(128, 128),
            conv(128, 128),

            up(128,64),
            conv(64, 64),
            conv(64, 64),

            up(64,32),
            conv(32, 32),
            conv(32, 32),

            up(32,16)
        )
        self.output_conv = conv(16, 1, False) # ReLu activation is removed to keep the logits for the loss function


    def forward(self, input):
      y = self.input_conv(input)
      y = self.down(y)

      b,c,h,w = y.size(0),y.size(1),y.size(2),y.size(3)
      y = y.view(b, -1)
      y = self.fc_net(y)
      y = y.view(b,c,h,w)

      y = self.up(y)
      output = self.output_conv(y)
      #print(output.shape)
      return output
```


input-tensor depth:0 (2, 3, 128, 128)
conv
Sequential
Conv2d depth:3 input: (2, 3, 128, 128) output: (2, 16, 128, 128)
BatchNorm2d depth:3 input: (2, 16, 128, 128) output: (2, 16, 128, 128)
ReLU depth:3 input: (2, 16, 128, 128) output: (2, 16, 128, 128)
Sequential
down
Sequential
down
Sequential depth:3 input: (2, 16, 128, 128) output: (2, 32, 64, 64)
conv
Sequential depth:3 input: (2, 32, 64, 64) output: (2, 32, 64, 64)
conv
Sequential depth:3 input: (2, 32, 64, 64) output: (2, 32, 64, 64)
down
Sequential depth:3 input: (2, 32, 64, 64) output: (2, 64, 32, 32)
conv
Sequential depth:3 input: (2, 64, 32, 32) output: (2, 64, 32, 32)
conv
Sequential depth:3 input: (2, 64, 32, 32) output: (2, 64, 32, 32)
down
Sequential depth:3 input: (2, 64, 32, 32) output: (2, 128, 16, 16)
conv
Sequential depth:3 input: (2, 128, 16, 16) output: (2, 128, 16, 16)
conv
Sequential depth:3 input: (2, 128, 16, 16) output: (2, 128, 16, 16)
down
Sequential depth:3 input: (2, 128, 16, 16) output: (2, 256, 8, 8)
conv
Sequential depth:3 input: (2, 256, 8, 8) output: (2, 256, 8, 8)
conv
Sequential depth:3 input: (2, 256, 8, 8) output: (2, 256, 8, 8)
down
Sequential depth:3 input: (2, 256, 8, 8) output: (2, 512, 4, 4)
view depth:1 input: (2, 512, 4, 4) output: (2, 8192)
Sequential
Linear depth:2 input: (2, 8192) output: (2, 8192)
BatchNorm1d depth:2 input: (2, 8192) output: (2, 8192)
ReLU depth:2 input: (2, 8192) output: (2, 8192)
view depth:1 input: (2, 8192) output: (2, 512, 4, 4)
depth:1 output: (2, 512, 4, 4)
Sequential
up
ConvTranspose2d depth:3 input: (2, 512, 4, 4) output: (2, 512, 8, 8)
conv depth:3 input: (2, 512, 8, 8) output: (2, 256, 8, 8)
conv
Sequential depth:3 input: (2, 256, 8, 8) output: (2, 256, 8, 8)
conv
Sequential depth:3 input: (2, 256, 8, 8) output: (2, 256, 8, 8)
up
ConvTranspose2d depth:3 input: (2, 256, 8, 8) output: (2, 256, 16, 16)
conv depth:3 input: (2, 256, 16, 16) output: (2, 128, 16, 16)
conv
Sequential depth:3 input: (2, 128, 16, 16) output: (2, 128, 16, 16)
conv
Sequential depth:3 input: (2, 128, 16, 16) output: (2, 128, 16, 16)
up
ConvTranspose2d depth:3 input: (2, 128, 16, 16) output: (2, 128, 32, 32)
conv depth:3 input: (2, 128, 32, 32) output: (2, 64, 32, 32)
conv
Sequential depth:3 input: (2, 64, 32, 32) output: (2, 64, 32, 32)
conv
Sequential depth:3 input: (2, 64, 32, 32) output: (2, 64, 32, 32)
up
ConvTranspose2d depth:3 input: (2, 64, 32, 32) output: (2, 64, 64, 64)
conv depth:3 input: (2, 64, 64, 64) output: (2, 32, 64, 64)
conv
Sequential depth:3 input: (2, 32, 64, 64) output: (2, 32, 64, 64)
conv
Sequential depth:3 input: (2, 32, 64, 64) output: (2, 32, 64, 64)
up
ConvTranspose2d depth:3 input: (2, 32, 64, 64) output: (2, 32, 128, 128)
conv depth:3 input: (2, 32, 128, 128) output: (2, 16, 128, 128)
conv
Sequential
Conv2d depth:3 input: (2, 16, 128, 128) output: (2, 1, 128, 128)
output-tensor depth:0 (2, 1, 128, 128)

```
summary(model, (3, 128, 128))
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 128, 128]             448
       BatchNorm2d-2         [-1, 16, 128, 128]              32
              ReLU-3         [-1, 16, 128, 128]               0
            conv-4           [-1, 16, 128, 128]               0
            Conv2d-5         [-1, 32, 128, 128]           4,640
       BatchNorm2d-6         [-1, 32, 128, 128]              64
              ReLU-7         [-1, 32, 128, 128]               0
            conv-8           [-1, 32, 128, 128]               0
         MaxPool2d-9           [-1, 32, 64, 64]               0
           down-10             [-1, 32, 64, 64]               0
           Conv2d-11           [-1, 32, 64, 64]           9,248
      BatchNorm2d-12           [-1, 32, 64, 64]              64
             ReLU-13           [-1, 32, 64, 64]               0
           conv-14             [-1, 32, 64, 64]               0
           Conv2d-15           [-1, 32, 64, 64]           9,248
      BatchNorm2d-16           [-1, 32, 64, 64]              64
             ReLU-17           [-1, 32, 64, 64]               0
           conv-18             [-1, 32, 64, 64]               0
           Conv2d-19           [-1, 64, 64, 64]          18,496
      BatchNorm2d-20           [-1, 64, 64, 64]             128
             ReLU-21           [-1, 64, 64, 64]               0
           conv-22             [-1, 64, 64, 64]               0
         MaxPool2d-23           [-1, 64, 32, 32]               0
           down-24             [-1, 64, 32, 32]               0
           Conv2d-25           [-1, 64, 32, 32]          36,928
      BatchNorm2d-26           [-1, 64, 32, 32]             128
             ReLU-27           [-1, 64, 32, 32]               0
           conv-28             [-1, 64, 32, 32]               0
           Conv2d-29           [-1, 64, 32, 32]          36,928
      BatchNorm2d-30           [-1, 64, 32, 32]             128
             ReLU-31           [-1, 64, 32, 32]               0
           conv-32             [-1, 64, 32, 32]               0
           Conv2d-33          [-1, 128, 32, 32]          73,856
      BatchNorm2d-34          [-1, 128, 32, 32]             256
             ReLU-35          [-1, 128, 32, 32]               0
           conv-36            [-1, 128, 32, 32]               0
         MaxPool2d-37          [-1, 128, 16, 16]               0
           down-38             [-1, 128, 16, 16]               0
           Conv2d-39          [-1, 128, 16, 16]         147,584
      BatchNorm2d-40          [-1, 128, 16, 16]             256
             ReLU-41          [-1, 128, 16, 16]               0
           conv-42            [-1, 128, 16, 16]               0
           Conv2d-43          [-1, 128, 16, 16]         147,584
      BatchNorm2d-44          [-1, 128, 16, 16]             256
             ReLU-45          [-1, 128, 16, 16]               0
           conv-46            [-1, 128, 16, 16]               0
           Conv2d-47          [-1, 256, 16, 16]         295,168
      BatchNorm2d-48          [-1, 256, 16, 16]             512
             ReLU-49          [-1, 256, 16, 16]               0
           conv-50            [-1, 256, 16, 16]               0
         MaxPool2d-51            [-1, 256, 8, 8]               0
           down-52               [-1, 256, 8, 8]               0
           Conv2d-53             [-1, 256, 8, 8]         590,080
      BatchNorm2d-54             [-1, 256, 8, 8]             512
             ReLU-55             [-1, 256, 8, 8]               0
           conv-56               [-1, 256, 8, 8]               0
           Conv2d-57             [-1, 256, 8, 8]         590,080
      BatchNorm2d-58             [-1, 256, 8, 8]             512
             ReLU-59             [-1, 256, 8, 8]               0
           conv-60               [-1, 256, 8, 8]               0
           Conv2d-61             [-1, 512, 8, 8]       1,180,160
      BatchNorm2d-62             [-1, 512, 8, 8]           1,024
             ReLU-63             [-1, 512, 8, 8]               0
           conv-64               [-1, 512, 8, 8]               0
         MaxPool2d-65             [-1, 512, 4, 4]               0
           down-66              [-1, 512, 4, 4]               0
           Linear-67                  [-1, 8192]      67,117,056
      BatchNorm1d-68                  [-1, 8192]          16,384
             ReLU-69                  [-1, 8192]               0
  ConvTranspose2d-70             [-1, 512, 8, 8]       1,049,088
           Conv2d-71             [-1, 512, 8, 8]       1,179,904
      BatchNorm2d-72             [-1, 256, 8, 8]             512
             ReLU-73             [-1, 256, 8, 8]               0
           conv-74               [-1, 256, 8, 8]               0
             up-75               [-1, 256, 8, 8]               0
           Conv2d-76             [-1, 256, 8, 8]         590,080
      BatchNorm2d-77             [-1, 256, 8, 8]             512
             ReLU-78             [-1, 256, 8, 8]               0
           conv-79               [-1, 256, 8, 8]               0
           Conv2d-80             [-1, 256, 8, 8]         590,080
      BatchNorm2d-81             [-1, 256, 8, 8]             512
             ReLU-82             [-1, 256, 8, 8]               0
           conv-83               [-1, 256, 8, 8]               0
  ConvTranspose2d-84           [-1, 256, 16, 16]         262,400
           Conv2d-85           [-1, 128, 16, 16]         295,040
      BatchNorm2d-86           [-1, 128, 16, 16]             256
             ReLU-87           [-1, 128, 16, 16]               0
           conv-88             [-1, 128, 16, 16]               0
             up-89             [-1, 128, 16, 16]               0
           Conv2d-90           [-1, 128, 16, 16]         147,584
      BatchNorm2d-91           [-1, 128, 16, 16]             256
             ReLU-92           [-1, 128, 16, 16]               0
           conv-93             [-1, 128, 16, 16]               0
           Conv2d-94           [-1, 128, 16, 16]         147,584
      BatchNorm2d-95           [-1, 128, 16, 16]             256
             ReLU-96           [-1, 128, 16, 16]               0
           conv-97             [-1, 128, 16, 16]               0
  ConvTranspose2d-98           [-1, 128, 32, 32]          65,664
           Conv2d-99            [-1, 64, 32, 32]          73,792
     BatchNorm2d-100            [-1, 64, 32, 32]             128
            ReLU-101            [-1, 64, 32, 32]               0
          conv-102              [-1, 64, 32, 32]               0
            up-103              [-1, 64, 32, 32]               0
          Conv2d-104            [-1, 64, 32, 32]          36,928
     BatchNorm2d-105            [-1, 64, 32, 32]             128
            ReLU-106            [-1, 64, 32, 32]               0
          conv-107              [-1, 64, 32, 32]               0
          Conv2d-108            [-1, 64, 32, 32]          36,928
     BatchNorm2d-109            [-1, 64, 32, 32]             128
            ReLU-110            [-1, 64, 32, 32]               0
          conv-111              [-1, 64, 32, 32]               0
 ConvTranspose2d-112            [-1, 64, 64, 64]          16,448
          Conv2d-113            [-1, 32, 64, 64]          18,464
     BatchNorm2d-114            [-1, 32, 64, 64]              64
            ReLU-115            [-1, 32, 64, 64]               0
          conv-116              [-1, 32, 64, 64]               0
            up-117              [-1, 32, 64, 64]               0
          Conv2d-118            [-1, 32, 64, 64]           9,248
     BatchNorm2d-119            [-1, 32, 64, 64]              64
            ReLU-120            [-1, 32, 64, 64]               0
          conv-121              [-1, 32, 64, 64]               0
          Conv2d-122            [-1, 32, 64, 64]           9,248
     BatchNorm2d-123            [-1, 32, 64, 64]              64
            ReLU-124            [-1, 32, 64, 64]               0
          conv-125              [-1, 32, 64, 64]               0
 ConvTranspose2d-126           [-1, 32, 128, 128]           4,128
          Conv2d-127           [-1, 16, 128, 128]           4,624
     BatchNorm2d-128           [-1, 16, 128, 128]              32
            ReLU-129           [-1, 16, 128, 128]               0
          conv-130             [-1, 16, 128, 128]               0
            up-131             [-1, 16, 128, 128]               0
          Conv2d-132              [-1, 1, 128, 128]             145
          conv-133              [-1, 1, 128, 128]               0
================================================================
Total params: 74,818,113
Trainable params: 74,818,113
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 100.44
Params size (MB): 285.41
Estimated Total Size (MB): 386.03
----------------------------------------------------------------
```

Connected to Python 3 Google Compute Engine backend (GPU)

2.3 Report the loss functions that I used and the plot the total training loss of the training procedure

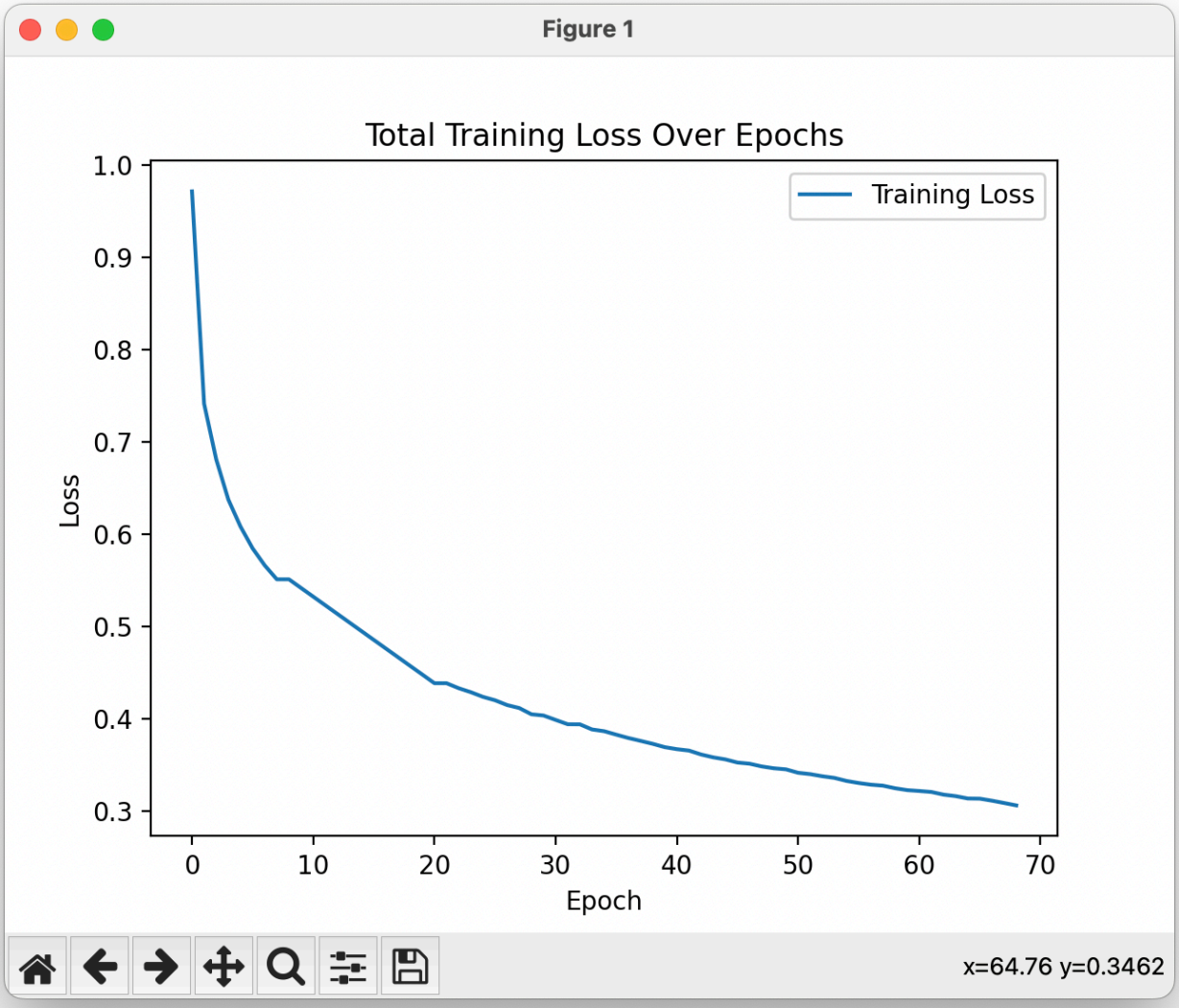
2.3.1 Loss function I used

```
crit = nn.BCEWithLogitsLoss() # Define the loss function
optim = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
```

2.3.2 training loss

My training loss is 3 times than the original one is becuase I did data augmentation inside the for loop, which have orginal image, h_filp image and v_filp image

```
loss_list =
[0.9716119170188904,0.7413926720619202,0.6809871792793274,0.6373889446258545,0.6082096695899963,0
.5845683217048645,0.5660731196403503,0.5509427785873413,

 0.55094278,0.54155553,0.53216827,0.52278102,0.51339377,0.50400652,0.49461927,0.48523201,0.475844
76,0.46645751,0.45707026,0.447683,0.43829575,

 0.43829575181007385,0.4328637719154358,0.428557425737381,0.42350563406944275,0.4198061525821686,
0.41460368037223816,0.41136375069618225, 0.4045928120613098,

 0.4032733142375946,0.39848583936691284,0.39379405975341797,0.3937068283557892,0.3881424069404602
,0.3862606883049011,0.38249462842941284,
                0.37887489795684814, 0.3757827877998352, 0.3725910782814026, 0.36895090341567993,
0.3667604923248291, 0.36521628499031067,0.36099541187286377,0.35789838433265686,

 0.35567939281463623,0.35227134823799133,0.35101622343063354,0.3480479121208191,0.346030175685882
57,0.3448733687400818, 0.3411721885204315, 0.3396941125392914,
                0.3374038338661194, 0.33560723066329956, 0.33230581879615784,
0.33006858825683594, 0.3282719850540161, 0.3272158205509186, 0.32439732551574707,
0.32234859466552734, 0.3214151859283447, 0.3203503489494324,
                0.3174724280834198, 0.31584280729293823, 0.3132993280887604, 0.31308993697166443,
0.31090065836906433, 0.3083840012550354, 0.3057791590690613]

```



2.4 Report the final mean IoU of my model.

```
#images: 93, Mean IoU: 0.7536434492757244
```

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

Part3

3.1 Names of my group members

Team Member: Rongsheng Qian & Yuwen Jia & Isaac Ding

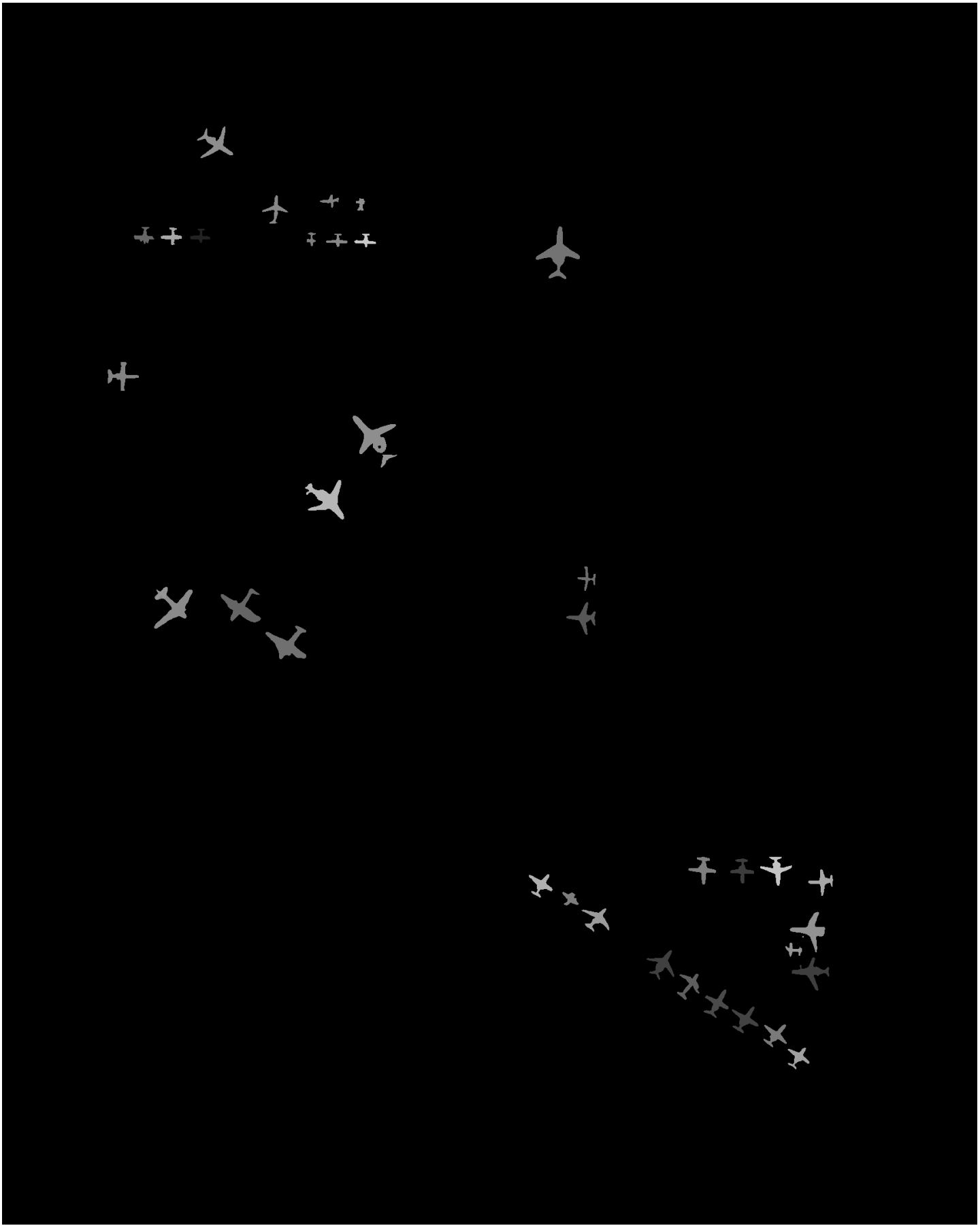
Names of group members: Rongsheng Qian, Isaac Ding, Yuwen Jia

3.2 Report the best score

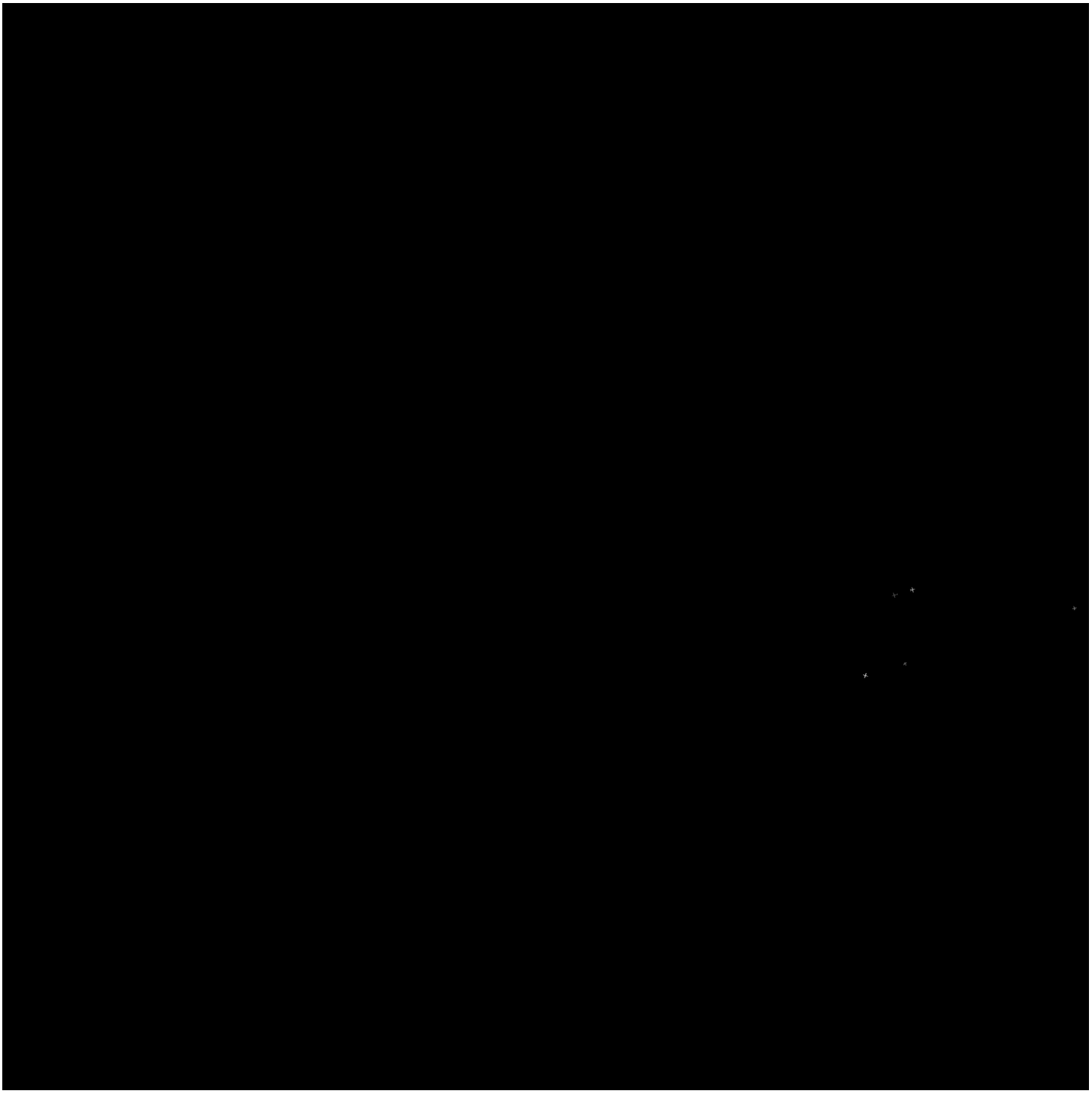
0.42208

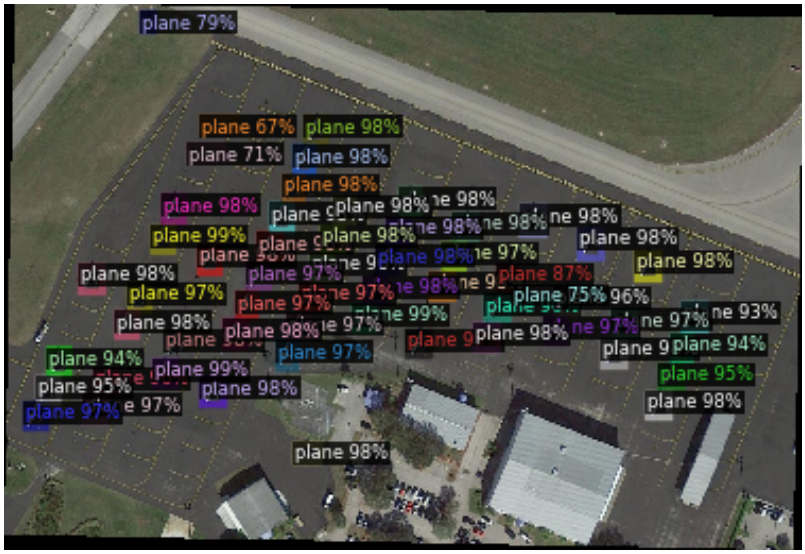
3.3 The visualization of results for 3 random samples from the test set.

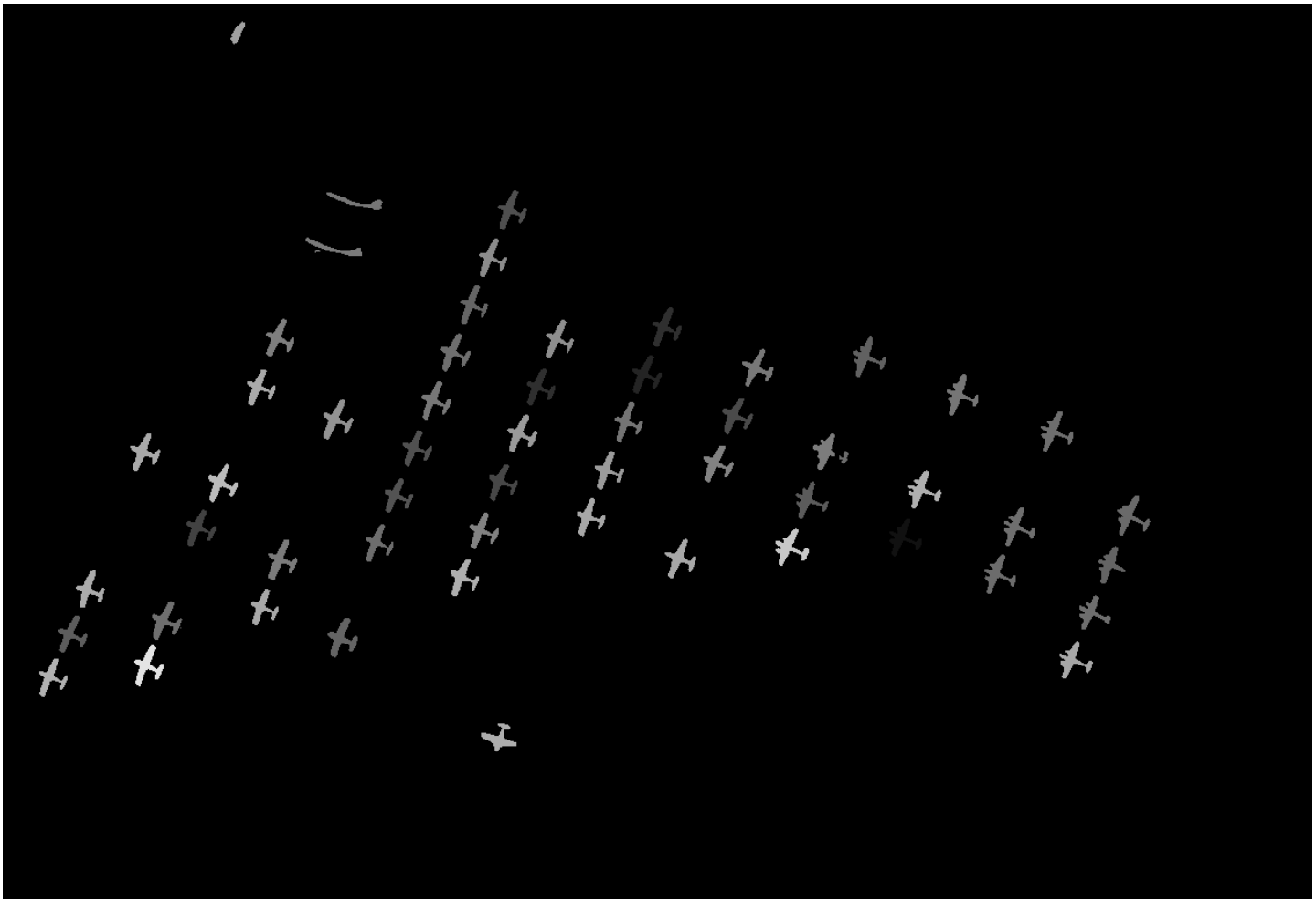

plane 98%
plane 64%
plane 88%
plane 77%
plane 92%
plane 99%
plane 98%
plane 64%
plane 95%
plane 99%
plane 99%
plane 96%
plane 99%
plane 95%
plane 95%
plane 98%
plane 92%
plane 71%
plane 98%
plane 96%
plane 96%
plane 98%
plane 98%
plane 99%
30




plane 83%
plane 77%
plane 74%

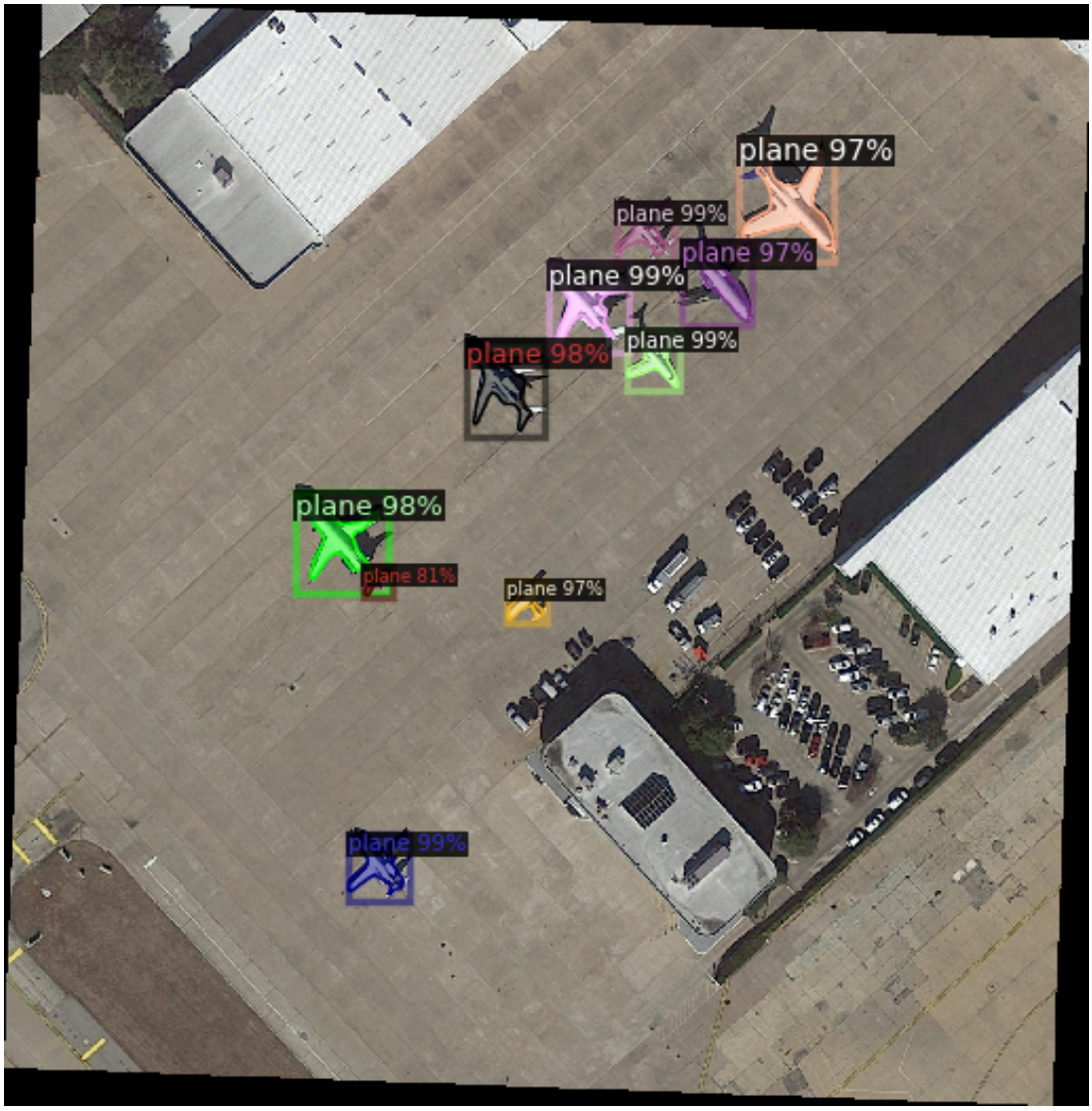






Part4

4.1 The visualization and the evaluation results similar to Part 1.


plane 97%
plane 99%
plane 97%
plane 99%
plane 98%
plane 99%
plane 98%
plane 81%
plane 97%
plane 99%


plane 97%
plane 97%
plane 98%
plane 96%


plane 99%
plane 98%
plane 99%
plane 79%
plane 99%
plane 98%
plane 73%
plane 98%

4.2 Explain the differences between the results of Part 3 and Part 4

```
OrderedDict([('bbox',
              {'AP': 28.398225937281584,
               'AP50': 47.84622204366393,
               'AP75': 30.779133903503457,
               'APs': 11.893842234937116,
               'APm': 38.731526537111584,
               'APl': 50.17012741157468,
               'AP-': nan,
               'AP-plane': 28.398225937281584}),
             ('segm',
              {'AP': 8.453292716889878,
               'AP50': 30.151730837392794,
               'AP75': 0.3140860304517846,
               'APs': 3.0730235536909922,
               'APm': 10.329902541380156,
               'APl': 24.477706032394554,
               'AP-': nan,
               'AP-plane': 8.453292716889878})])
```

4.2.1 Training & inference time

Its training & inference time is potentially slower to converge than models focused solely on object detection due to its dual-task nature.

4.2.2 Flexibility

Part 3 model have more flexibility than part 4, which can adapt model to specific task requirements. For example, the mask-rcnn cannot find the plane in last sample but the combination of RetinaNet and FCN can do it by adding post-processing and pre-processing.

4.2.3 Segmentation

The result of segmentation in part4 seems worse than part3, which cannot recognize the whole plane. Its Ap of segm only have 8.4 which is lower than the part 3