CMPT412 assignment 3

Name: Nontawat Janpongsri

Student ID: 301311427

Note: for this assignment I would like to use 2 late days

Note2: this assignment was discussed with Jiangpei Chen

Part 1:

- List of the configs and modifications that you used.

```
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.005
cfg.SOLVER.MAX_ITER = 1000
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.

```
cfg.SOLVER.BASE_LR = 0.005
cfg.SOLVER.MAX_ITER = 1000
```

- For MAX_ITER it is the total amount of iteration the program uses to train. The original was at 500 and I have increased to 1000. Thus, the program can train in more iteration to further improve it accuracy.
- For BASE_LR it is the learning rate at which the program begins to train. The original BASE_LR was 0.00025 but I have increased to 0.005 which I see significant improve in accuracy.

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

```
[11/04 01:50:01 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  AP1   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 33.322 | 60.227 | 33.693 | 24.810 | 40.840 | 60.498 |
OrderedDict([('bbox', {'AP': 33.32171243514948, 'AP50': 60.22665698553333, 'AP75': 33.69256282105782, 'APs': 24.81030603585426, 'APm': 40.84017905669789, 'AP1': 60.497668144482})])
```

- The visualization of 3 test samples and the predicted results.

- 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 visualisation of a test sample for qualitative comparison.
    - With the original BASE_LR and MAX_ITER the accuracy was only around 25 but with the new value the accuracy jumps up to 60

```
[11/04 06:09:04 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 18.351 | 36.671 | 15.325 | 13.190 | 25.018 | 36.730 |
```

Note: The image above is train using the original BASE_LR and MAX_ITER

Part 2:

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

```
# Set the hyperparameters
num_epochs = 5
batch_size = 64
learning_rate = 0.05
weight_decay = 1e-5
```

- Report the final architecture of your network including any modification that you have for the layers. Briefly explain the reason for each modification

```
class conv(nn.Module):
    def __init__(self, in_ch, out_ch, activation=True):
        super(conv, self).__init__()
        if(activation):
          self.layer = nn.Sequential(
             nn.Conv2d(in_ch, out_ch, 3, padding=1),
             nn.BatchNorm2d(out_ch),
             nn.ReLU(inplace=True)
          )
        else:
          self.layer = nn.Sequential(
             nn.Conv2d(in_ch, out_ch, 3, padding=1)
             )

    def forward(self, x):
        x = self.layer(x)
        return x

'''
# downsampling module equal to a conv module followed by a max-pool layer
'''
class down(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(down, self).__init__()
        self.layer = nn.Sequential(
            conv(in_ch, out_ch),
            nn.MaxPool2d(2)
            )

    def forward(self, x):
        x = self.layer(x)
        return x

'''
# upsampling module equal to a upsample function followed by a conv module
'''
class up(nn.Module):
    def __init__(self, in_ch, out_ch, bilinear=False):
        super(up, self).__init__()
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        else:
            self.up = nn.ConvTranspose2d(in_ch, in_ch, 2, stride=2)

        self.conv = conv(in_ch, out_ch)

    def forward(self, x):
        y = self.up(x)
        y = self.conv(y)
        return y
```

```
'''
# the main model which you need to complete by using above modules.
# you can also modify the above modules in order to improve your results.
'''
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()

        # Encoder

        self.input_conv = conv(3, 4)
        self.down = down(4, 8)

        # Decoder

        self.up = up(8, 4)
        self.output_conv = conv(4, 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)
      y = self.up(y)
      output = self.output_conv(y)
      return output
```

Note: no change

- 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
```

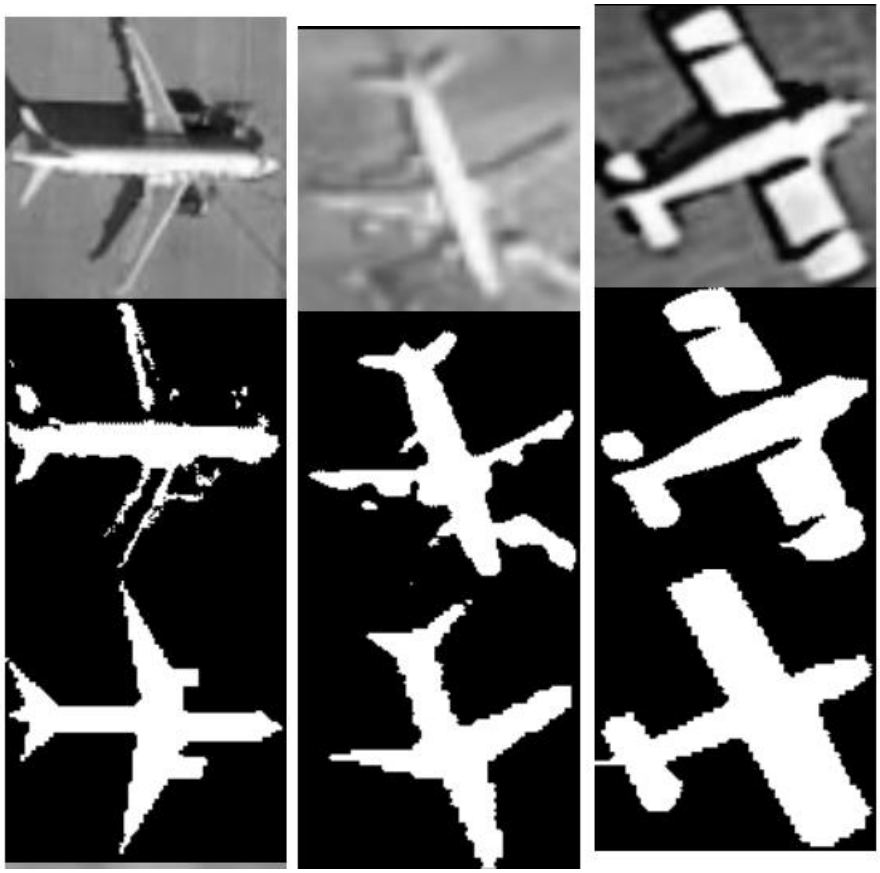
```
Epoch: 4, Loss: 0.2930546700954437
```

- Report the final mean IoU of your model.

```
#images: 998, Mean IoU: 0.6105884869662657
```

- Visualize 3 test images and the corresponding predicted masks.

Note: the first is the original, second is predicted and third is the mask



Part 3:

- the name under which you submitted on Kaggle

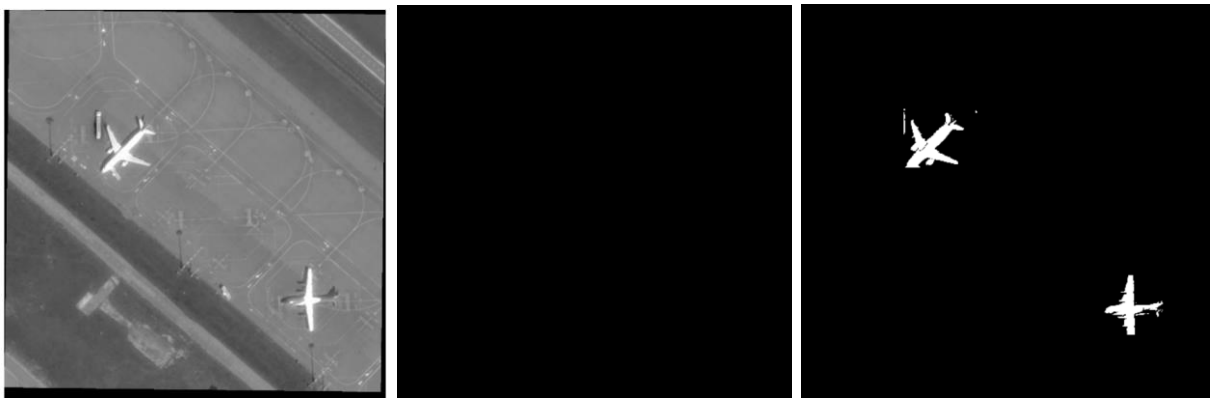
cloud427

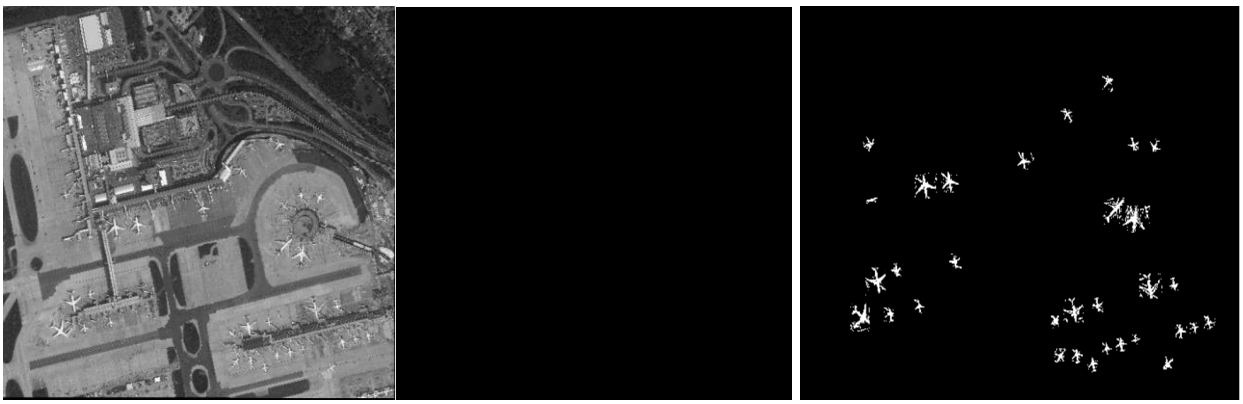
- Report the best score (should match your score on Kaggle).

60 cloud427 0.17945

- The visualisation of results for 3 random test samples.

Note: the first image is the original in greyscale, the second is mask and third is predicted. Also for test data because we did not have the segmentation for it initially so the mask image would just be all black.





Part 4:

- The visualisation and the evaluation results similar to Part 1

- Explain the differences

- With the change that we have apply to our training by using the mask_rcnn_R_50_FPN_3x.yaml instead of the `faster_rcnn_R_101_FPN_3x.yml` now the image that we visualize after training have some mask applied to it. as you can see in the image above compare to the image in part 1 the plane color has color mask on top of it.