DLCV HW2 Report

學號：B06901045 系級：電機四 姓名：曹林熹

Problem 1: Image classification (10%)

1. (2%) (no collaborators)

Print the network architecture of your model.

我使用的是 pretrained 好的 VGG19_bn，代表是有經過 batch normalization 的 pretrained model。

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (25): ReLU(inplace=True)
    (26): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (27): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (34): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (35): ReLU(inplace=True)
    (36): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (37): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (38): ReLU(inplace=True)
    (39): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (42): ReLU(inplace=True)
    (43): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (44): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (45): ReLU(inplace=True)
    (46): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (47): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (48): ReLU(inplace=True)
    (49): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (50): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (51): ReLU(inplace=True)
    (52): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

2. (2%) (no collaborators)

Report accuracy of model on the validation set.

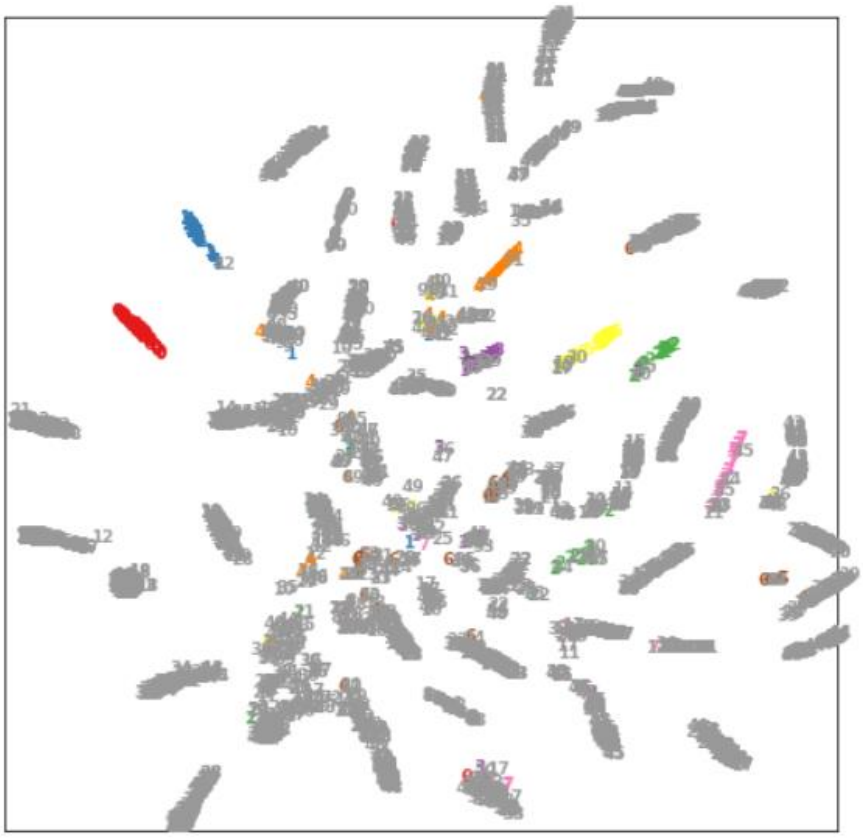
最好的 val_acc 如下：

```
best_val_acc = 0.7224
```

3. (6%) (no collaborators)

Visualize the classification result on validation set by implementing t-SNE on output features of the second last layer. Briefly explain your result of t-SNE visualization.

這裡的 second last layer 為 Linear 出來的結果，可以看到在 t-SNE 的結果下，我們成功地將這些圖片分類，數量接近五十組。而下圖接近左偏下的地方有比較多組聚集在一起，因此我認為這塊區域容易使得模型搞混，導致錯誤率提高。



Problem 2: Semantic segmentation (30%)

1. (5%) (no collaborators)

Print the network architecture of your VGG16-FCN32s model.

```
VGG16_FCN32s(
  (down_sampling): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fcn): Sequential(
    (0): Conv2d(512, 4096, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Dropout2d(p=0.7, inplace=False)
    (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): Dropout2d(p=0.7, inplace=False)
    (6): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
    (7): ReLU(inplace=True)
  )
  (up_sampling): Sequential(
    (0): ConvTranspose2d(7, 7, kernel_size=(64, 64), stride=(32, 32), padding=(16, 16))
  )
)
```

2. (5%) (no collaborators)

Show the predicted segmentation mask of

a. validation/0010_sat.jpg

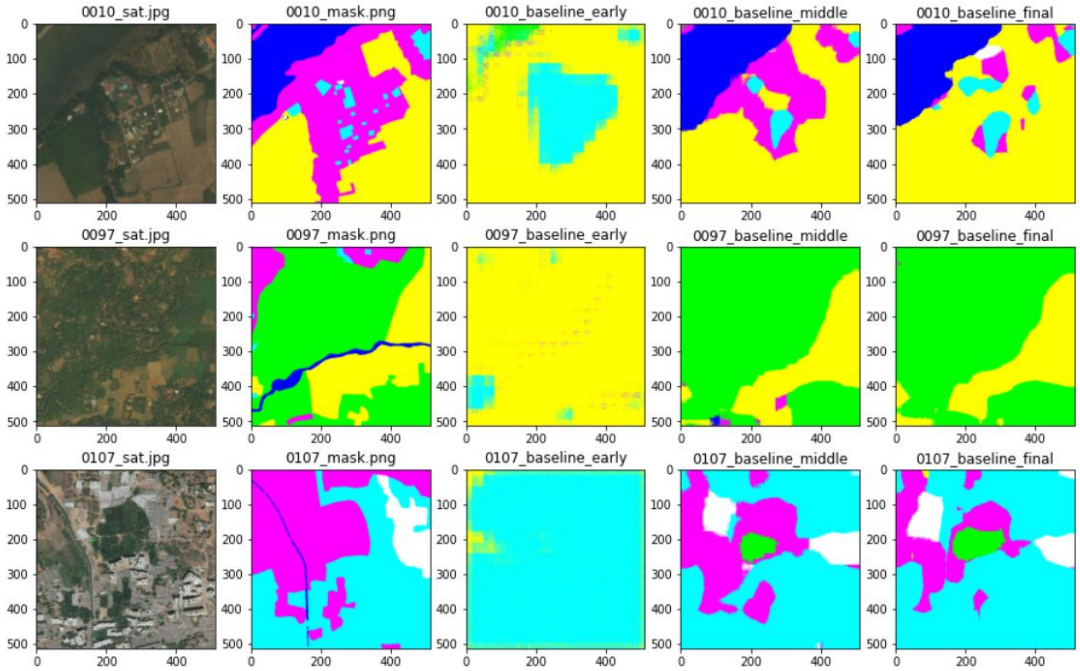
b. validation/0097_sat.jpg

c. validation/0107_sat.jpg

during the early, middle, and the final stage during the training stage. (For example, results of 1st, 10th, 20th epoch)

Baseline Model for different stages：

Stage	Early (epoch = 1)	Middle (epoch = 15)	Final (epoch = 30)	*Best
Mean_iou	0.16361	0.66881	0.67100	0.67166



*註：

1. 因為 torch 在 colab 版本不同的問題，我的 baseline_model_best 以及 improved_model_best 最後有用同樣的架構重新訓練過， 因此所顯示不同 stage 的圖片是之前 model 的，不過兩者最後的 mean_iou 並沒有差很多，而最後一題會再探討最後 summit 的 baseline_model_best 以及 improved_model_best 不同的差異，請助教見諒。

2. final model 只代表 epoch 為最後的 model，並不代表 best model。

3. (5%) (no collaborators)

Implement an improved model which performs better than your baseline model. Print the network architecture of this model.

```
VGG16_FCN8s(
  (down_sampling_8): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (down_sampling_16): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (down_sampling_32): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fcn8): Sequential(
    (0): Conv2d(256, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (fcn16): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (fcn32): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (up_sampling_2): Sequential(
    (0): ConvTranspose2d(1024, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
  (up_sampling_4): Sequential(
    (0): ConvTranspose2d(1024, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
  (up_sampling_8): Sequential(
    (0): Conv2d(1024, 4096, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): Conv2d(4096, 1024, kernel_size=(1, 1), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): Dropout2d(p=0.5, inplace=False)
    (6): Conv2d(1024, 7, kernel_size=(1, 1), stride=(1, 1))
    (7): ReLU(inplace=True)
    (8): ConvTranspose2d(7, 7, kernel_size=(32, 32), stride=(8, 8), padding=(12, 12))
  )
)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 512, 512]           1,792
              ReLU-2         [-1, 64, 512, 512]               0
            Conv2d-3         [-1, 64, 512, 512]          36,928
              ReLU-4         [-1, 64, 512, 512]               0
         MaxPool2d-5         [-1, 64, 256, 256]               0
            Conv2d-6        [-1, 128, 256, 256]          73,856
              ReLU-7        [-1, 128, 256, 256]               0
            Conv2d-8        [-1, 128, 256, 256]         147,584
              ReLU-9        [-1, 128, 256, 256]               0
        MaxPool2d-10        [-1, 128, 128, 128]               0
           Conv2d-11        [-1, 256, 128, 128]         295,168
             ReLU-12        [-1, 256, 128, 128]               0
           Conv2d-13        [-1, 256, 128, 128]         590,080
             ReLU-14        [-1, 256, 128, 128]               0
           Conv2d-15        [-1, 256, 128, 128]         590,080
             ReLU-16        [-1, 256, 128, 128]               0
        MaxPool2d-17          [-1, 256, 64, 64]               0
           Conv2d-18          [-1, 512, 64, 64]       1,180,160
             ReLU-19          [-1, 512, 64, 64]               0
           Conv2d-20          [-1, 512, 64, 64]       2,359,808
             ReLU-21          [-1, 512, 64, 64]               0
           Conv2d-22          [-1, 512, 64, 64]       2,359,808
             ReLU-23          [-1, 512, 64, 64]               0
        MaxPool2d-24          [-1, 512, 32, 32]               0
           Conv2d-25          [-1, 512, 32, 32]       2,359,808
             ReLU-26          [-1, 512, 32, 32]               0
           Conv2d-27          [-1, 512, 32, 32]       2,359,808
             ReLU-28          [-1, 512, 32, 32]               0
           Conv2d-29          [-1, 512, 32, 32]       2,359,808
             ReLU-30          [-1, 512, 32, 32]               0
        MaxPool2d-31          [-1, 512, 16, 16]               0
           Conv2d-32         [-1, 1024, 16, 16]       4,719,616
  ConvTranspose2d-33         [-1, 1024, 32, 32]      16,778,240
           Conv2d-34         [-1, 1024, 32, 32]       4,719,616
  ConvTranspose2d-35         [-1, 1024, 64, 64]      16,778,240
           Conv2d-36         [-1, 1024, 64, 64]       2,360,320
           Conv2d-37         [-1, 4096, 64, 64]      37,752,832
             ReLU-38         [-1, 4096, 64, 64]               0
        Dropout2d-39         [-1, 4096, 64, 64]               0
           Conv2d-40         [-1, 1024, 64, 64]       4,195,328
             ReLU-41         [-1, 1024, 64, 64]               0
        Dropout2d-42         [-1, 1024, 64, 64]               0
           Conv2d-43            [-1, 7, 64, 64]           7,175
             ReLU-44            [-1, 7, 64, 64]               0
  ConvTranspose2d-45          [-1, 7, 512, 512]          50,183
================================================================
Total params: 102,076,238
Trainable params: 102,076,238
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.00
Forward/backward pass size (MB): 1717.44
Params size (MB): 389.39
Estimated Total Size (MB): 2109.83
```

4. (5%) (no collaborators)

Show the predicted segmentation mask of

a. validation/0010_sat.jpg

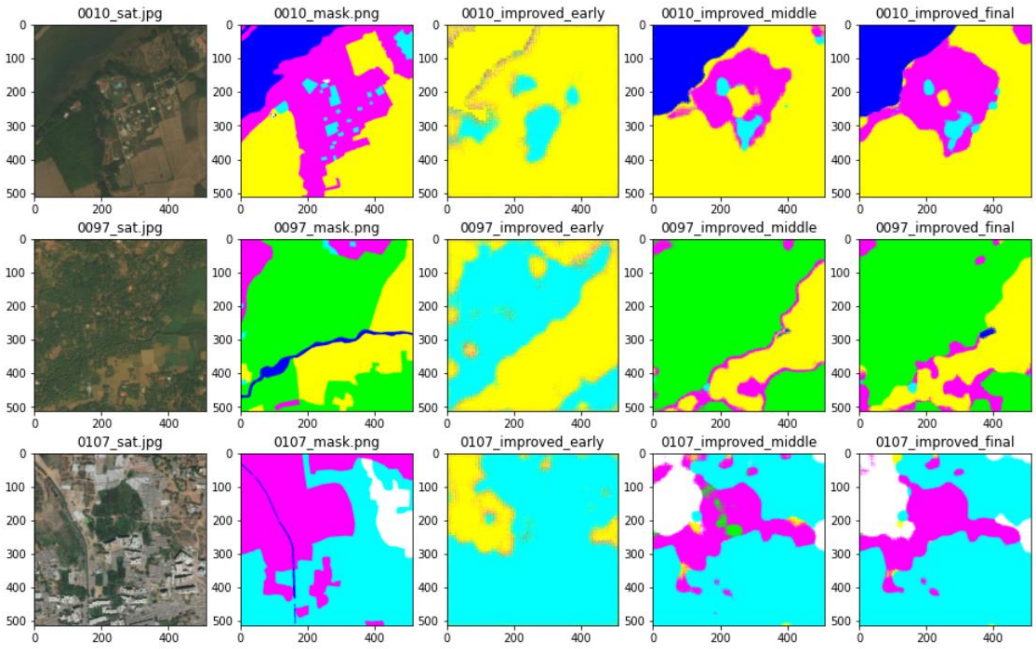
b. validation/0097_sat.jpg

c. validation/0107_sat.jpg

during the early, middle, and the final stage during the this Improved model.

Improved Model for different stages：

Stage	Early (epoch = 1)	Middle (epoch = 15)	Final (epoch = 30)	*Best
Mean_iou	0.19761	0.66256	0.67035	0.67528



5. (10%) (no collaborators)

Report mIoU score of both models on the validation set. Discuss the reason why the improved model performs better than the baseline one. You may conduct some experiments and show some evidences to support your reasoning.

Baseline_model (FCN32s) Best: 0.67166

Improved_model (FCN8s) Best: 0.67528

我的 Improved_model 並沒有高 baseline 很多，不過對於分割模型來說差 0.4 個百分點也算是有所突破，為了了解這中間的改變，我先比較了 improved model 在哪個 class 中有更好的表現，接著拿出這個 class 出現較多的 images。

首先可以看出，在多數 class 中 improved 的表現都比 baseline 好，將比較重點放在 class 0 (Urban-Cyan) 與 class 4 (Water-Blue)，然而在 class 5 (Barren land-White) 中反而輸給了 baseline 0.3，因此我主要會觀察 class 0、class 4 以及順便探討 class 5 的狀況。

Baseline_model	Improved_model
val loss = 8.525625802576542	val loss = 10.96209604665637
class #0 : 0.73257	class #0 : 0.74765
class #1 : 0.87004	class #1 : 0.87178
class #2 : 0.27131	class #2 : 0.28363
class #3 : 0.78799	class #3 : 0.79312
class #4 : 0.72761	class #4 : 0.74175
class #5 : 0.64041	class #5 : 0.61376
mean_iou: 0.671657	mean_iou: 0.675282

首先可以看到我們的 class 0 (id=0006)、class 4 (id=0013) 做為比較，可以看到 id=0006 時，improved 預測 Cyan 比較準確，我認為是 fcn8s 的 feature map 可以看到更廣的範圍，較可以看到全局的影像，因為多了只 maxpooling 三次的資訊，不過可以看到兩個 output 都對紫色有錯誤的認知。接著看到 id=0013 兩者皆有不錯的表現，然而小區域一個是誤認紫色，一個多誤認白色。而探討 class 5 (id=0120)，發現 improved 會將我們的白色區域預測為紫色，我認為是反映到了 improved 比 baseline 更會誤辨紫色區域的問題。