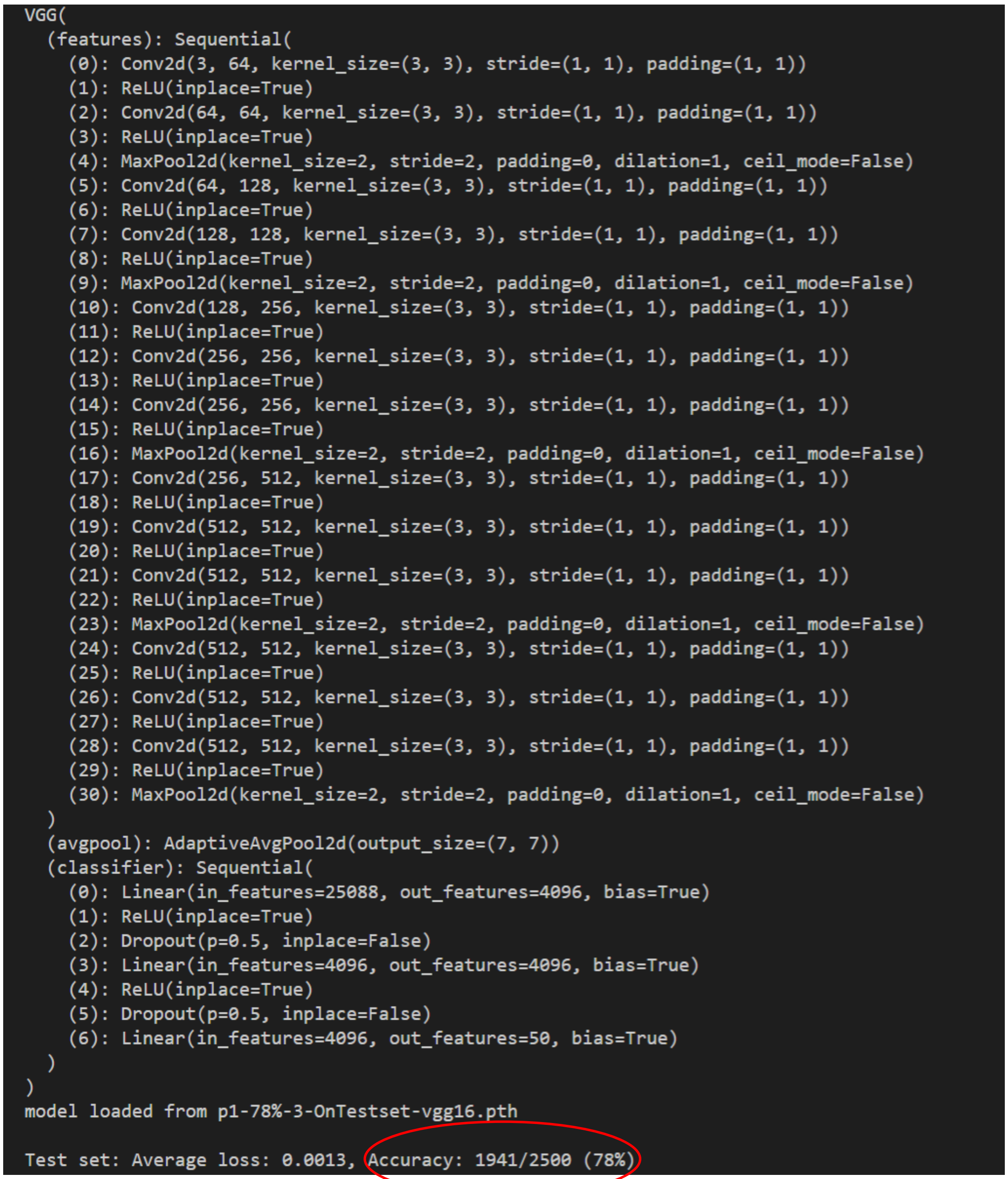
DLCV HW2 REPORT

學生: 葉政樑　學號:　R08945006

Problem1-1: Print the network architecture of your model



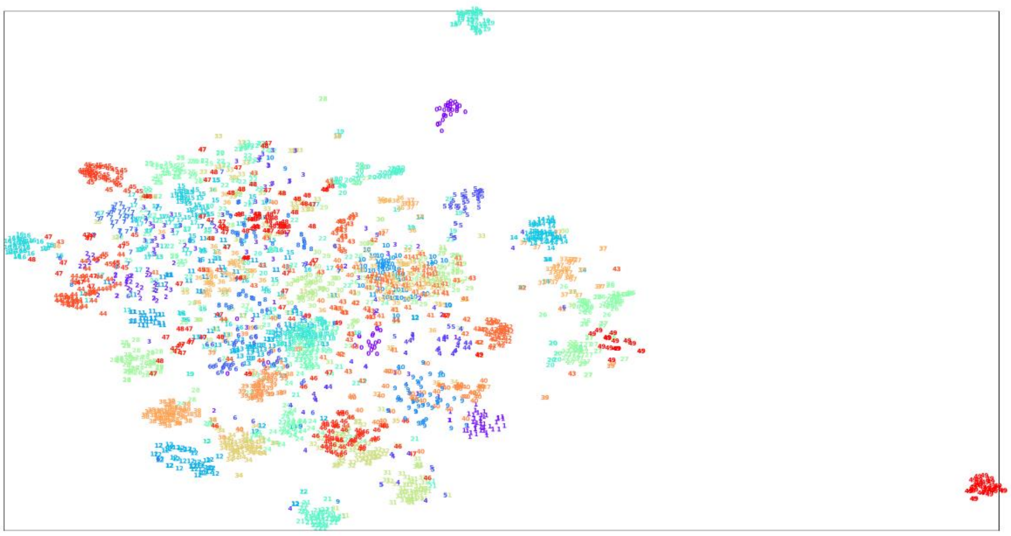
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 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)
  )
  (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=50, bias=True)
  )
)
model loaded from p1-78%-3-OnTestset-vgg16.pth

Test set: Average loss: 0.0013, Accuracy: 1941/2500 (78%)
```

Problem1-2: Report accuracy of model on the validation set

Accuracy is 78%.

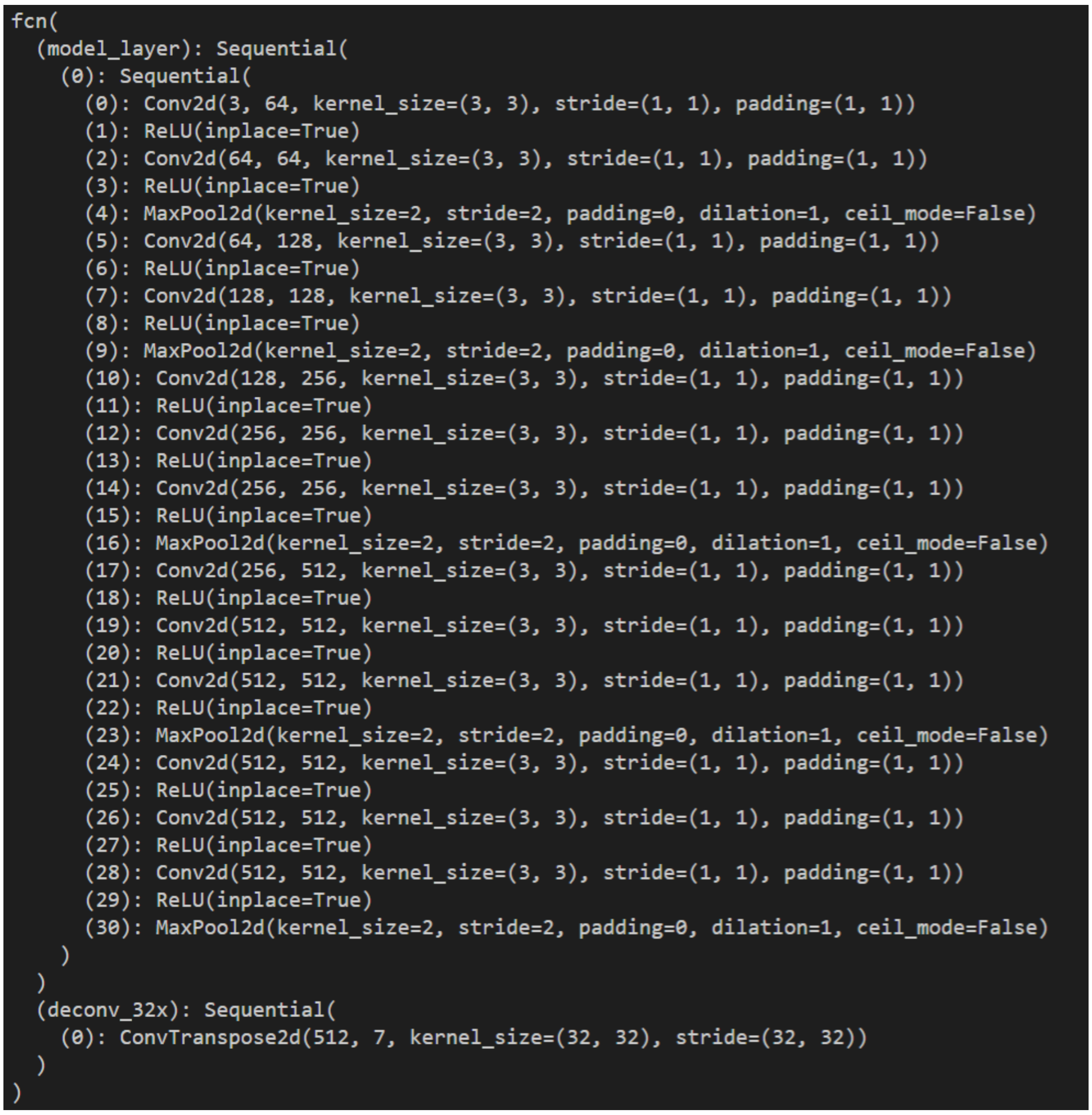
Problem1-3: Visualize the classification result on validation set by implementing t-SNE on output features of the second last layer. Briefly explain your result of tSNE visualization.



由上圖可見，tSNE 將 model 學習到的分類結果從高維空間(4096)投影至平面(2)並且由此視覺化後我們可以看到大致上相同 label 的資料被分類在平面空間中的某一區。而在中間較混亂的區域中大致上 13、23 混雜在一起，可能是在手寫辨識上都有 3 而且 1、2 不好分辨，導致特別相近。33、41 則是分散比較廣的類別，並沒有特別聚在一起，推測可能是手寫圖片上外型與其他數字較難區分。最後可以看到 12，16，49 等等分類相當成功，沒有與其他類別交雜。

備註: 上圖顏色由 matplotlib rainbow 配色，因此 label 數字較相近會有較相近的顏色，實際上還是有差異，如果肉眼區分不出可以直接看 label 數字區分。

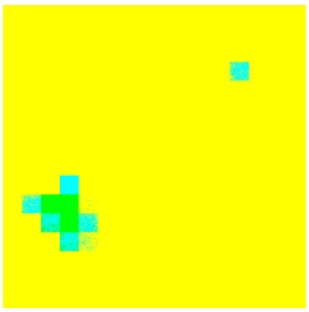
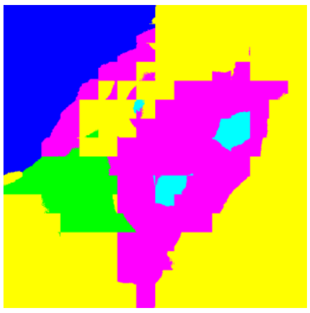
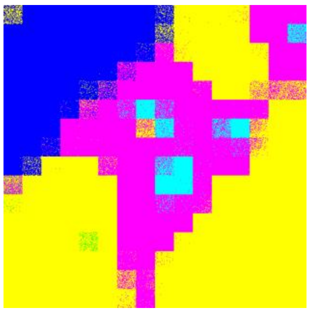
Problem2-1: Print the network architecture of your VGG16-FCN32s model.



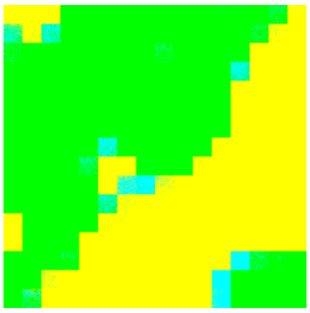
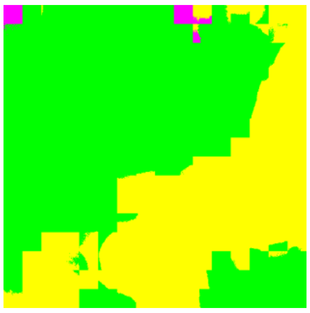
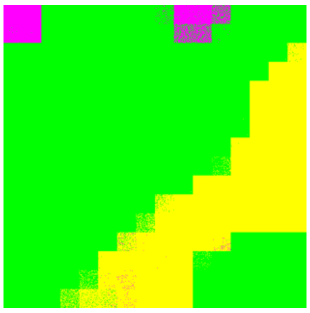
```
fcn(
  (model_layer): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
      (4): MaxPool2d(kernel_size=2, 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)
    )
  )
  (deconv_32x): Sequential(
    (0): ConvTranspose2d(512, 7, kernel_size=(32, 32), stride=(32, 32))
  )
)
```

Problem2-2: Show the predicted segmentation mask

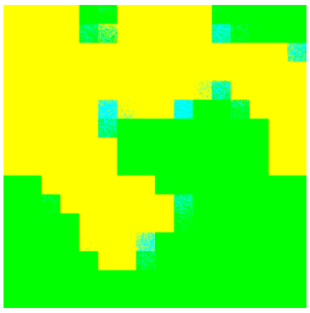
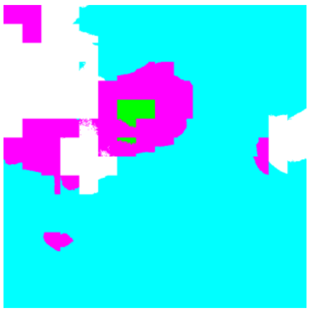
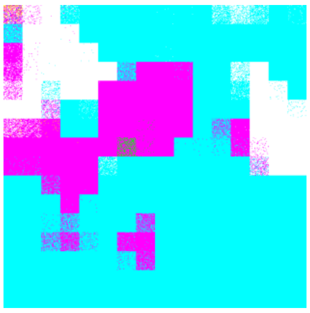
0010:

Epoch = 1	Epoch = 25	Epoch = 50
		

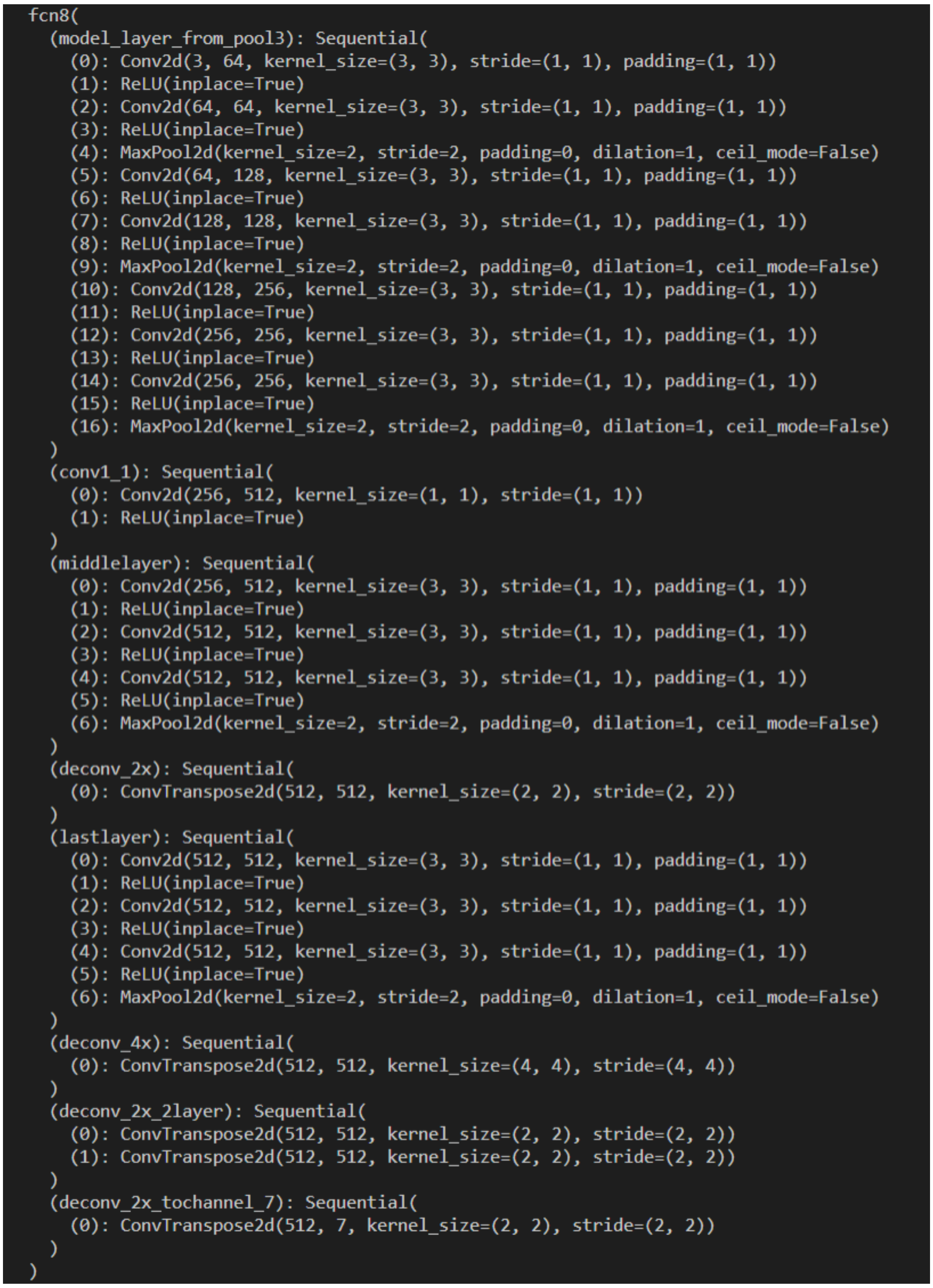
0097:

Epoch = 1	Epoch = 25	Epoch = 50
		

0107:

Epoch = 1	Epoch = 25	Epoch = 50
		

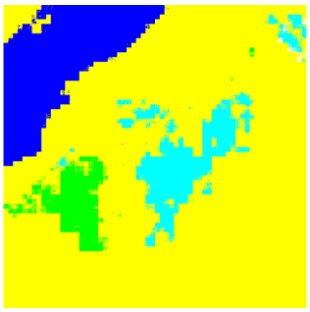
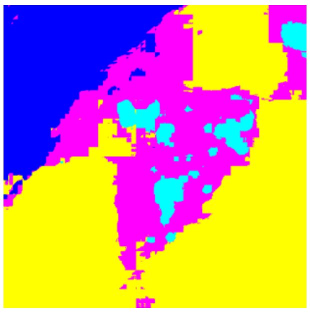
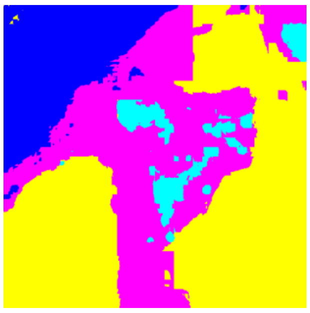
Problem2-3: Implement an improved model which performs better you're your baseline model. Print the network architecture of this model.



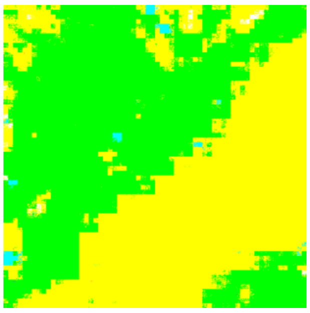
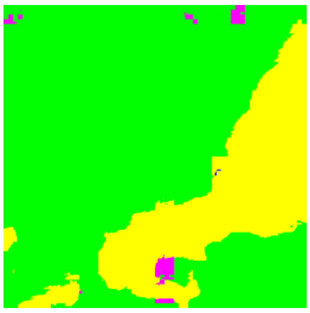
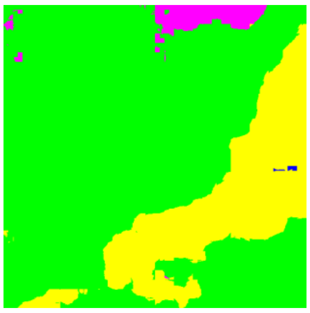
```
fcn8(
  (model_layer_from_pool3): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, 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)
  )
  (conv1_1): Sequential(
    (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (middlelayer): 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)
  )
  (deconv_2x): Sequential(
    (0): ConvTranspose2d(512, 512, kernel_size=(2, 2), stride=(2, 2))
  )
  (lastlayer): 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)
  )
  (deconv_4x): Sequential(
    (0): ConvTranspose2d(512, 512, kernel_size=(4, 4), stride=(4, 4))
  )
  (deconv_2x_2layer): Sequential(
    (0): ConvTranspose2d(512, 512, kernel_size=(2, 2), stride=(2, 2))
    (1): ConvTranspose2d(512, 512, kernel_size=(2, 2), stride=(2, 2))
  )
  (deconv_2x_tochannel_7): Sequential(
    (0): ConvTranspose2d(512, 7, kernel_size=(2, 2), stride=(2, 2))
  )
)
```

Problem2-4: Show the predicted segmentation mask

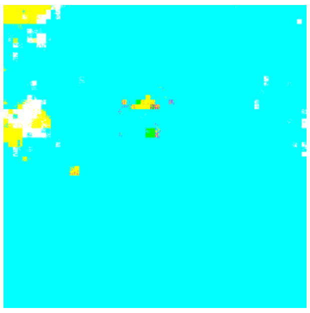
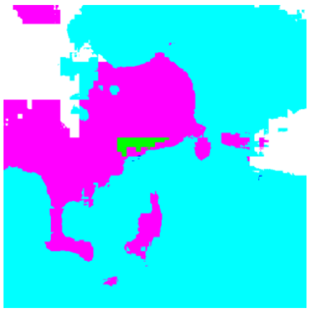
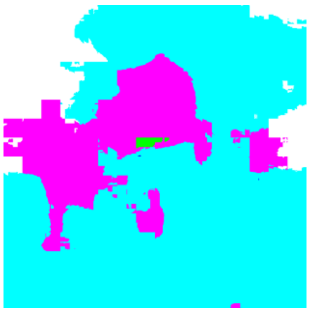
0010:

Epoch = 1	Epoch = 40	Epoch = 74
		

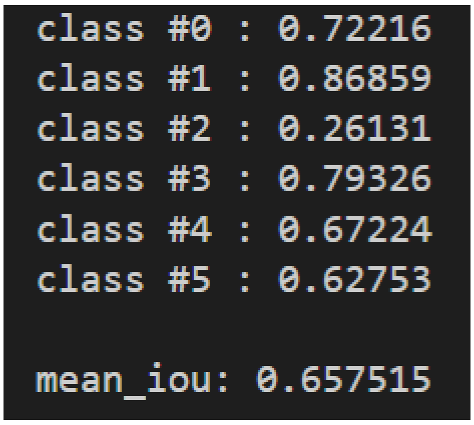
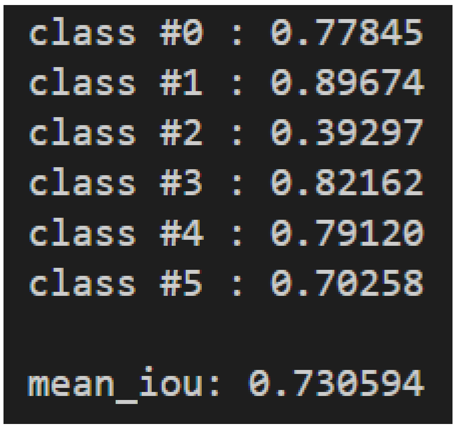
0097:

Epoch = 1	Epoch = 40	Epoch = 74
		

0107:

Epoch = 1	Epoch = 40	Epoch = 74
		

Problem2-5: 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.

Fcn32s	Fcn8s
 `class #0 : 0.72216` `class #1 : 0.86859` `class #2 : 0.26131` `class #3 : 0.79326` `class #4 : 0.67224` `class #5 : 0.62753` `mean_iou: 0.657515`	 `class #0 : 0.77845` `class #1 : 0.89674` `class #2 : 0.39297` `class #3 : 0.82162` `class #4 : 0.79120` `class #5 : 0.70258` `mean_iou: 0.730594`

由結果可見使用 improve model FCN8s 在各類別 iou 都比 FCN32s 好上很多，並且在 mean iou 上升 7.3%。

原因為在 FCN32s 時，我們是將最後一層的 feature maps 直接放大 32 倍回到原圖 size，這樣會損失較多空間資訊，由結果也可以看到比較多類似馬賽克的樣子，邊緣也較為粗糙。

使用 FCN8s 時有使用 skip architecture 參考從 pool4，pool3 來的 feature maps，分別將最後一層 feature maps 放大 4 倍與 pool4 的 feature maps 放大 2 倍，最後將 pool3 + pool4 + 最後一層的 feature maps，這樣可以多增加前面幾層的空間訊息，因此最後在結果上表現較好，邊緣表現也較為細緻。

下圖為 FCN32s、FCN8S 的結構圖:

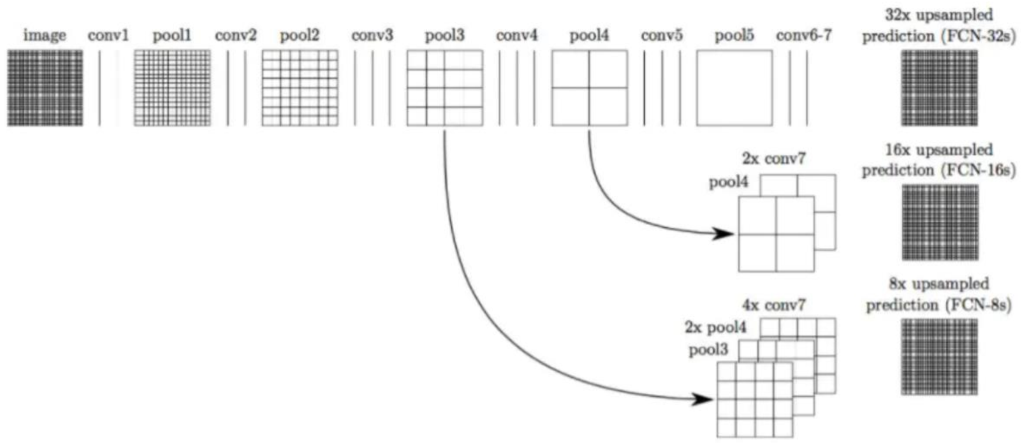


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

[2]

個人使用技巧: 我在 FCN8s 時也並非直接放大 8 倍，最終我使用的技巧是 2 倍 2 倍放大，我認為這樣可以在保留 feature maps 數量的同時也保留空間資訊。最終在測試之下比直接放大 8 倍效果好。

參考資料:

1. 歷屆 CNN 模型介紹:
   https://meetonfriday.com/posts/1e087b70/
2. Fully Convolutional Networks for Semantic Segmentation paper:
   https://arxiv.org/abs/1411.4038.
3. 【语义分割系列：三】FCN:
   https://blog.csdn.net/qq_31622015/article/details/90573667
4. FCN 的简单实现:
   https://zhuanlan.zhihu.com/p/32506912
5. t-SNE 实践:
   https://blog.csdn.net/hustqb/article/details/80628721
6. FCN 的学习及理解:
   https://blog.csdn.net/qq_36269513/article/details/80420363
7. 如何解决神经网络训练时 loss 不下降的问题:
   https://blog.ailemon.me/2019/02/26/solution-to-loss-doesnt-drop-in-nn-train/