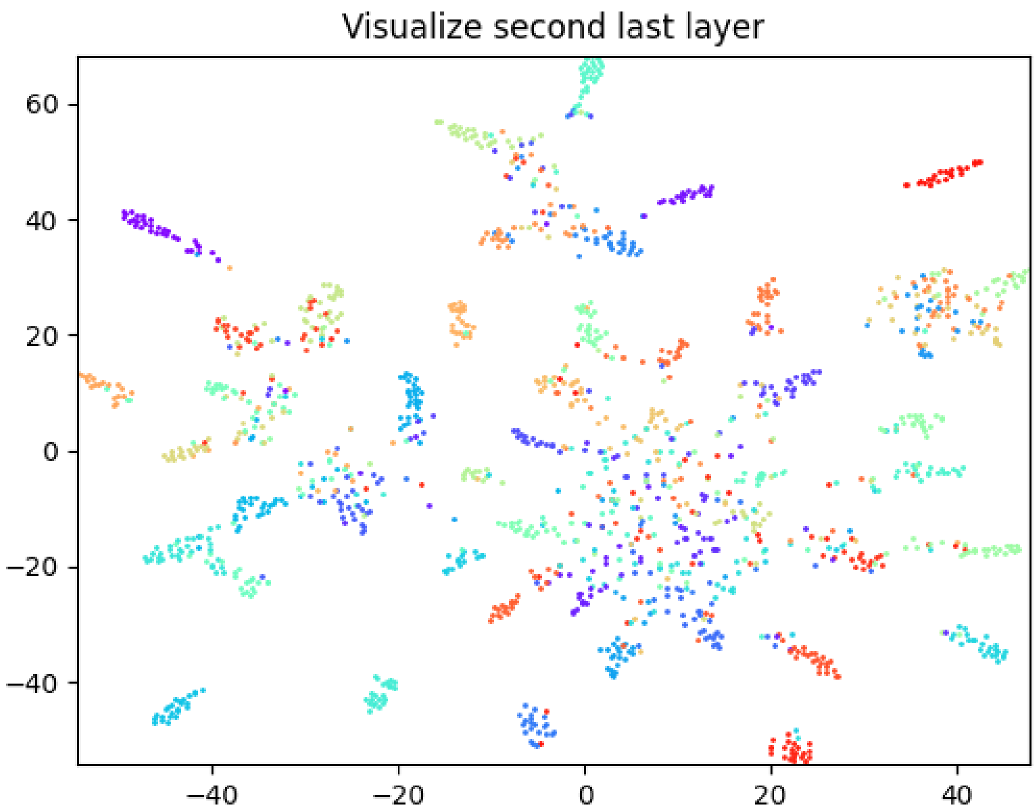
Problem 1: Image classification

1. (2%)Print the network architecture of your model.

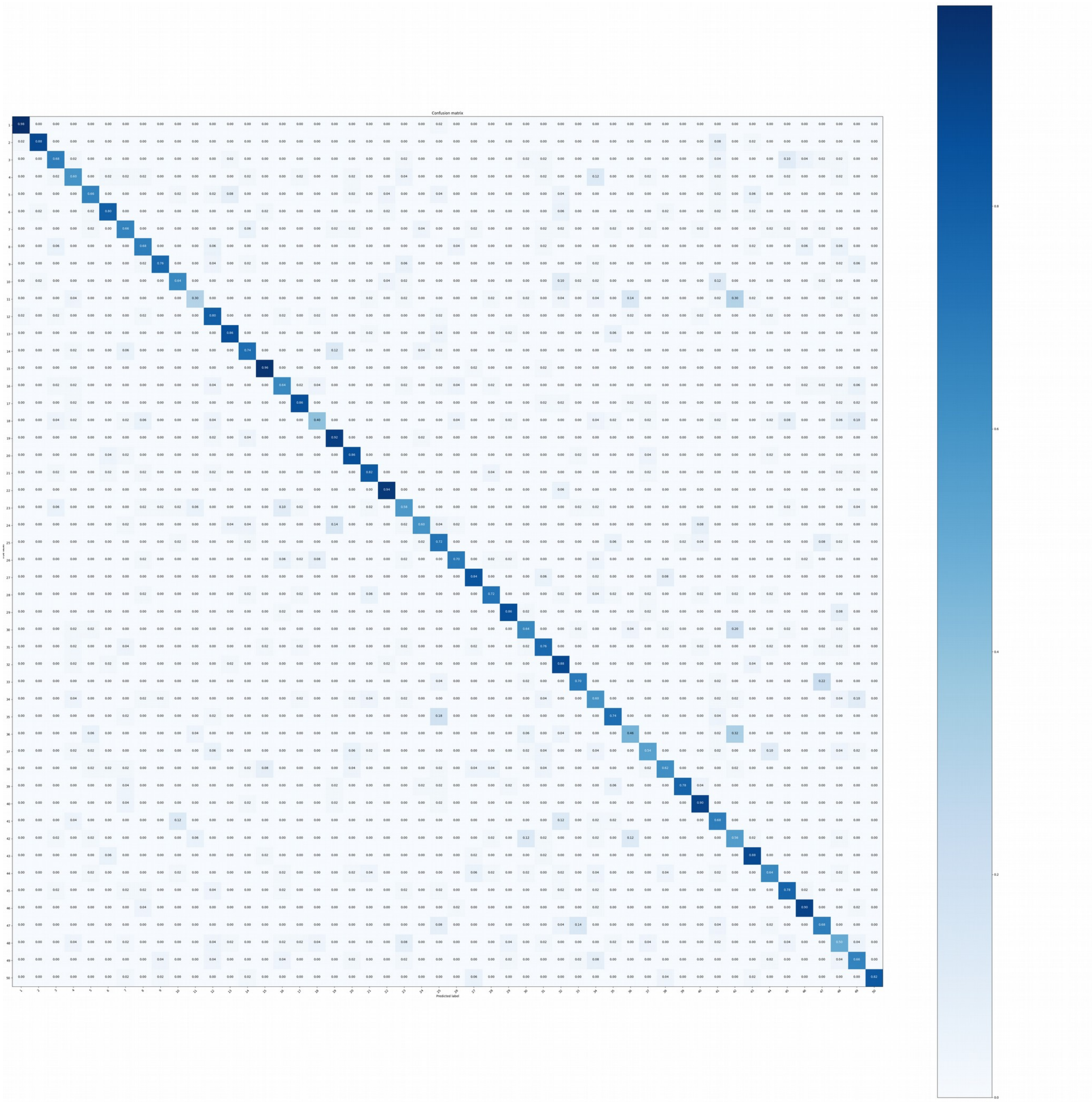
```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
              ReLU-2           [-1, 64, 32, 32]               0
            Conv2d-3           [-1, 64, 32, 32]          36,928
              ReLU-4           [-1, 64, 32, 32]               0
         MaxPool2d-5           [-1, 64, 16, 16]               0
            Conv2d-6          [-1, 128, 16, 16]          73,856
              ReLU-7          [-1, 128, 16, 16]               0
            Conv2d-8          [-1, 128, 16, 16]         147,584
              ReLU-9          [-1, 128, 16, 16]               0
        MaxPool2d-10            [-1, 128, 8, 8]               0
           Conv2d-11            [-1, 256, 8, 8]         295,168
             ReLU-12            [-1, 256, 8, 8]               0
           Conv2d-13            [-1, 256, 8, 8]         590,080
             ReLU-14            [-1, 256, 8, 8]               0
           Conv2d-15            [-1, 256, 8, 8]         590,080
             ReLU-16            [-1, 256, 8, 8]               0
        MaxPool2d-17            [-1, 256, 4, 4]               0
           Conv2d-18            [-1, 512, 4, 4]       1,180,160
             ReLU-19            [-1, 512, 4, 4]               0
           Conv2d-20            [-1, 512, 4, 4]       2,359,808
             ReLU-21            [-1, 512, 4, 4]               0
           Conv2d-22            [-1, 512, 4, 4]       2,359,808
             ReLU-23            [-1, 512, 4, 4]               0
        MaxPool2d-24            [-1, 512, 2, 2]               0
           Conv2d-25            [-1, 512, 2, 2]       2,359,808
             ReLU-26            [-1, 512, 2, 2]               0
           Conv2d-27            [-1, 512, 2, 2]       2,359,808
             ReLU-28            [-1, 512, 2, 2]               0
           Conv2d-29            [-1, 512, 2, 2]       2,359,808
             ReLU-30            [-1, 512, 2, 2]               0
        MaxPool2d-31            [-1, 512, 1, 1]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                 [-1, 1000]       4,097,000
================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.84
Params size (MB): 527.79
Estimated Total Size (MB): 532.65
----------------------------------------------------------------
```

2. (2%)Report accuracy of model on the validation set.
0.723600

3. (6%)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.



From the above plotting result, we can briefly find that there are lots of outputs(among 50 classes) from the last second layer have merged together with the same labels. However, there are still quite a lot of them can't be distinguished by our eyes, especially the part at the middle left of the picture, there are a bunch of data can't be tell from each other. Still, we can find good cluster results from the outlier. Maybe the result can be explained by the wrong recognition of labels(Below is the 50 classes confusion matrix), or slightly difference from those features output.



Problem 2: Semantic segmentation(70%)

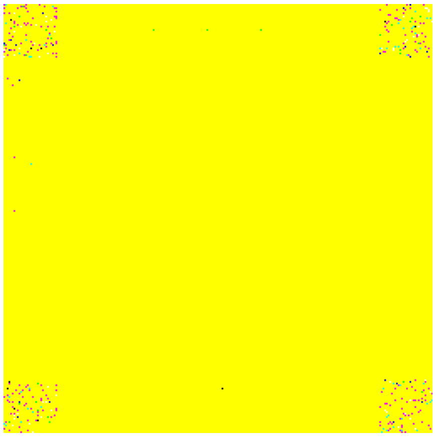
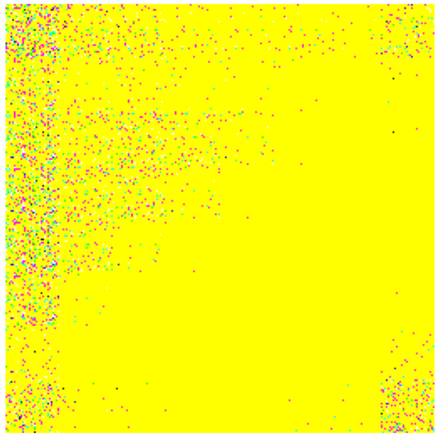
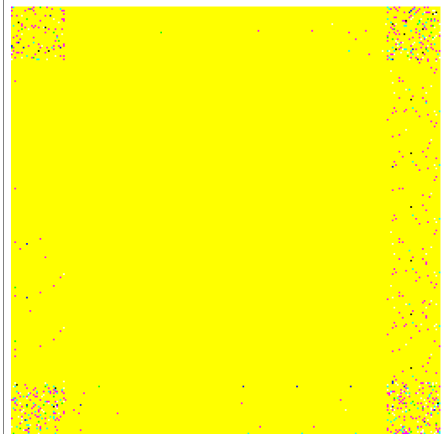
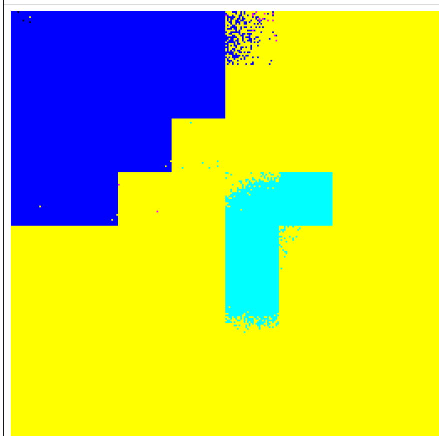
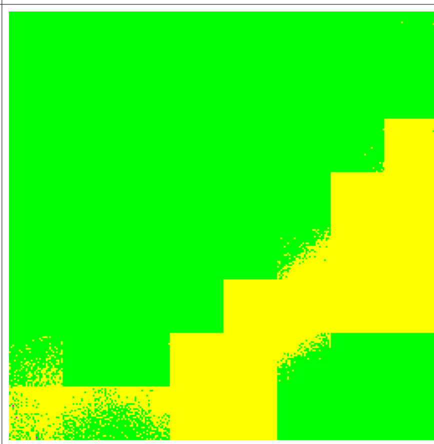
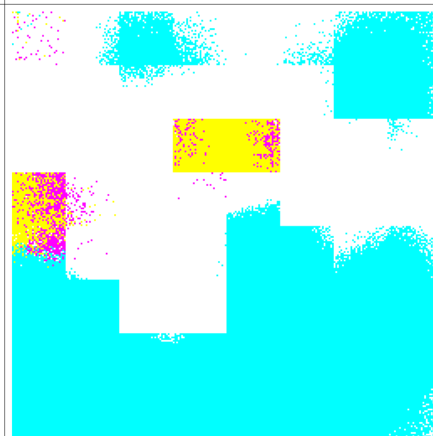
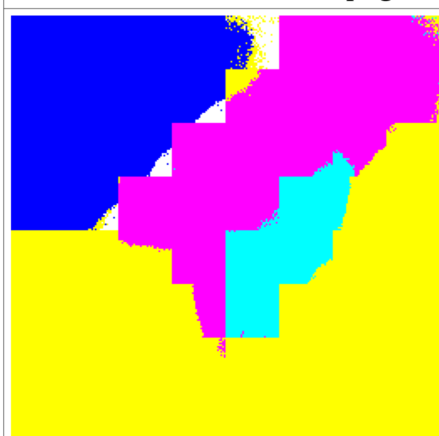
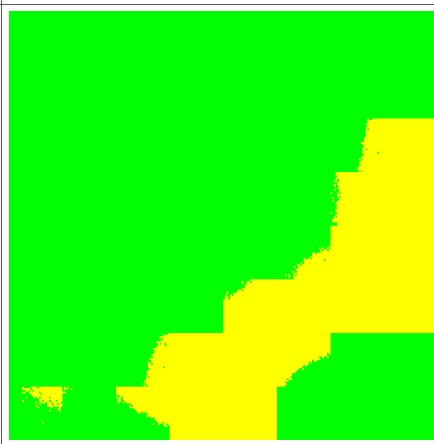
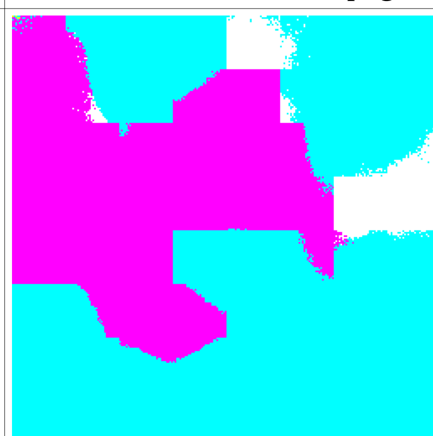
reference: https://github.com/wkentaro/pytorch-fcn

https://github.com/pochih/FCN-pytorch

1. (5%) Print the network architecture of your VGG16-FCN32s model.

```
File  Edit  View  Search  Terminal  Tabs  Help
                                    smartyam@smartyam-PS63-Modern-8SC
Estimated Total Size (MB): 797.06
----------------------------------------------------------------
(dlcv) dala@winyam:~/Documents/dlcv/hw2$ python hw2_p2_test.py ./hw
fcn32s
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 64, 256, 256]           1,792
              ReLU-2        [-1, 64, 256, 256]               0
            Conv2d-3        [-1, 64, 256, 256]          36,928
              ReLU-4        [-1, 64, 256, 256]               0
         MaxPool2d-5        [-1, 64, 128, 128]               0
            Conv2d-6       [-1, 128, 128, 128]          73,856
              ReLU-7       [-1, 128, 128, 128]               0
            Conv2d-8       [-1, 128, 128, 128]         147,584
              ReLU-9       [-1, 128, 128, 128]               0
        MaxPool2d-10         [-1, 128, 64, 64]               0
           Conv2d-11         [-1, 256, 64, 64]         295,168
             ReLU-12         [-1, 256, 64, 64]               0
           Conv2d-13         [-1, 256, 64, 64]         590,080
             ReLU-14         [-1, 256, 64, 64]               0
           Conv2d-15         [-1, 256, 64, 64]         590,080
             ReLU-16         [-1, 256, 64, 64]               0
        MaxPool2d-17         [-1, 256, 32, 32]               0
           Conv2d-18         [-1, 512, 32, 32]       1,180,160
             ReLU-19         [-1, 512, 32, 32]               0
           Conv2d-20         [-1, 512, 32, 32]       2,359,808
             ReLU-21         [-1, 512, 32, 32]               0
           Conv2d-22         [-1, 512, 32, 32]       2,359,808
             ReLU-23         [-1, 512, 32, 32]               0
        MaxPool2d-24         [-1, 512, 16, 16]               0
           Conv2d-25         [-1, 512, 16, 16]       2,359,808
             ReLU-26         [-1, 512, 16, 16]               0
           Conv2d-27         [-1, 512, 16, 16]       2,359,808
             ReLU-28         [-1, 512, 16, 16]               0
           Conv2d-29         [-1, 512, 16, 16]       2,359,808
             ReLU-30         [-1, 512, 16, 16]               0
        MaxPool2d-31           [-1, 512, 8, 8]               0
           Conv2d-32          [-1, 4096, 7, 7]       8,392,704
             ReLU-33          [-1, 4096, 7, 7]               0
        Dropout2d-34          [-1, 4096, 7, 7]               0
           Conv2d-35          [-1, 4096, 7, 7]      16,781,312
             ReLU-36          [-1, 4096, 7, 7]               0
        Dropout2d-37          [-1, 4096, 7, 7]               0
           Conv2d-38             [-1, 7, 7, 7]          28,679
  ConvTranspose2d-39         [-1, 7, 256, 256]         200,704
================================================================
Total params: 40,118,087
Trainable params: 40,118,087
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 297.94
Params size (MB): 153.04
Estimated Total Size (MB): 451.73
----------------------------------------------------------------
(dlcv) dala@winyam:~/Documents/dlcv/hw2$
```

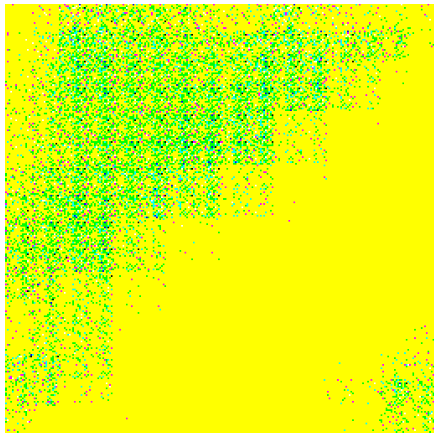
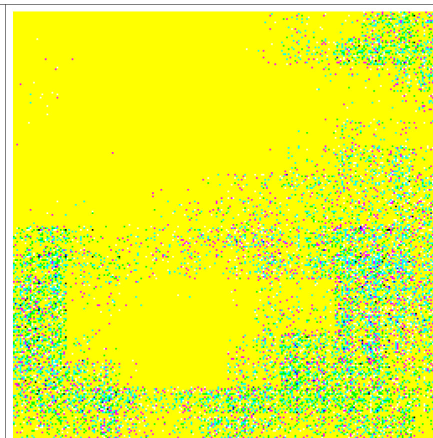
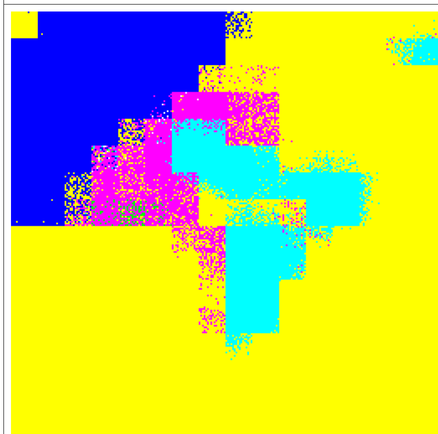
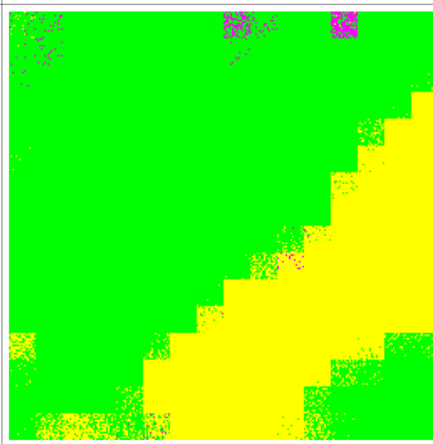
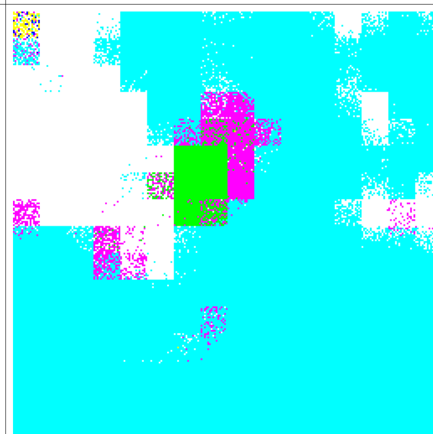
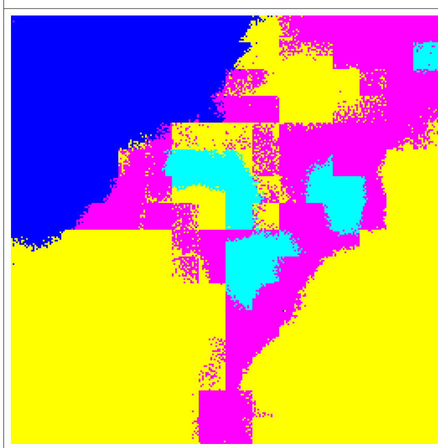
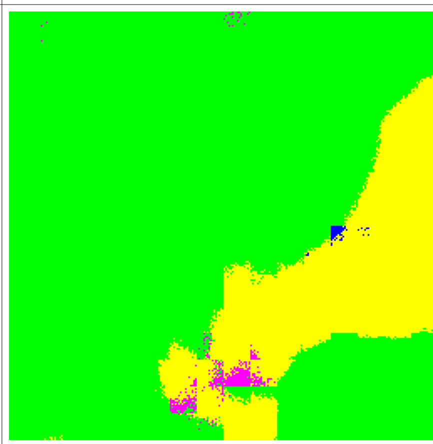
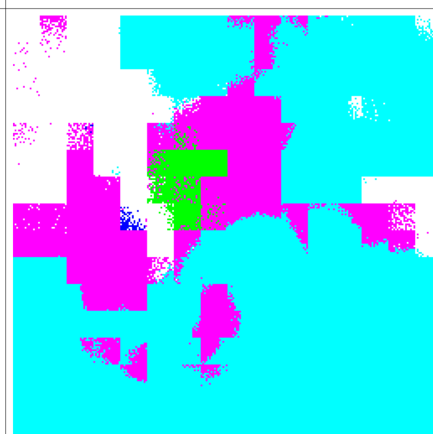
2. (5%) Show the predicted segmentation mask of “validation/0010_sat.jpg”, “validation/0097_sat.jpg”, “validation/0107_sat.jpg” during the early, middle, and the final stage during the training stage. (For example, results of 1st, 10th, 20th epoch)

		
fcn32s_1_0010_mask	fcn32s_1_0097_mask.png	fcn32s_1_0107_mask.png
		
fcn32s_1_0010_mask.png	fcn32s_1_0097_mask.png	fcn32s_1_0107_mask.png
		
fcn32s_1_0010_mask.png	fcn32s_1_0097_mask.png	fcn32s_1_0107_mask.png

3. (5%) Implement an improved model which performs better than your baseline model. Print the network architecture of this model.

```
fcn16s(
  (vgg): 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): Conv2d(512, 4096, kernel_size=(2, 2), stride=(1, 1))
      (1): ReLU(inplace=True)
      (2): Dropout2d(p=0.5, inplace=False)
      (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
      (4): ReLU(inplace=True)
      (5): Dropout2d(p=0.5, inplace=False)
      (6): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
      (7): ConvTranspose2d(7, 512, kernel_size=(4, 4), stride=(2, 2), bias=False)
    )
  )
  ,
  (to_pool4): 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)
  )
  (to_pool5): 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)
  )
  (upsample16): ConvTranspose2d(512, 7, kernel_size=(16, 16), stride=(16, 16), bias=False)
)
```

4. (5%) Show the predicted segmentation mask of “validation/0010_sat.jpg”, “validation/0097_sat.jpg”, “validation/0107_sat.jpg” during the early, middle, and the final stage during the training process of this improved model.

		
fcn16s_1_0010_mask	fcn16s_1_0097_mask	fcn16s_1_0107_mask
		
fcn16s_1_0010_mask	fcn16s_1_0097_mask	fcn16s_1_0107_mask
		
fcn16s_1_0010_mask	fcn16s_1_0097_mask	fcn16s_1_0107_mask

5. (10%) 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 mIoU	FCN16s mIoU
0.670	0.709

epoch	FCN32s mIoU	FCN16s mIoU
1	0.43	0.54
5	0.56	0.58
10	0.59	0.61
15	0.61	0.64
20	0.65	0.67

From the above table, we can find out that FCN16s predicts masks better than FCN32s does. Since the improved model combine predictions from the final layer and the former pooling layer (pool4), the model could predict finer details on the mask from the value of mean_IOU and get high-level information from FCN. The previous sections have shown the output masks of segmentation by FCN32s and FCN16s. Also,we can find that FCN16s converge faster than FCN32s in this case, which could be observed by the output mask after the first epoch(fcn16s_1_0097_mask on the above question has the significant improvement).From the score of meanIOU in each epoch, the FCN16s indeed improves segmentation detail by combining information from previous layers.