Deep Learning for Computer Vision – HW#1

宋體淮, R09921135, Electrical Engineering

Problem 1: Image classification

1. Print the network architecture of your model.

Pre-trained *Rsenet34* is used as the backbone model.

```
Model(
  (backbone): Sequential(
    (0): Conv2d(3, 64, kernel size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel size=3, stride=2, padding=1, dilation=1, ceil mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
```

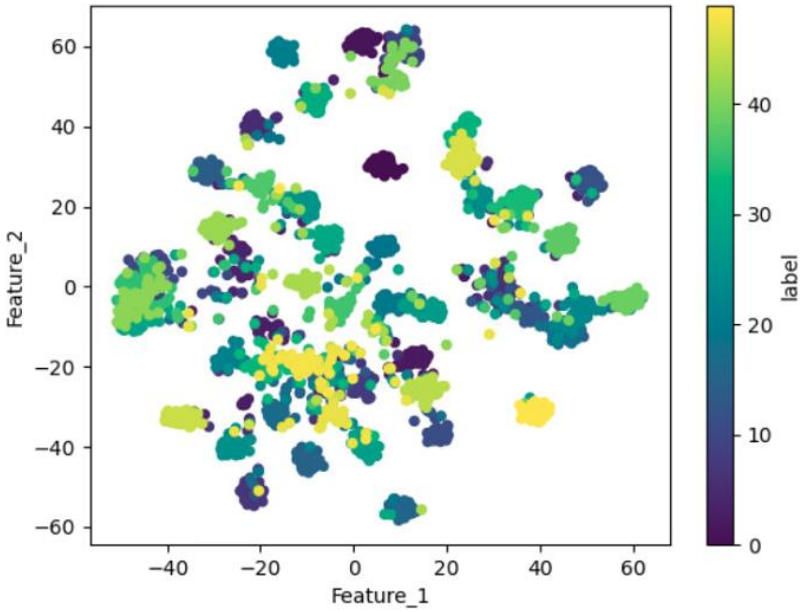
```
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
  )
  (6): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
  )
  (7): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track running stats=True)
    )
  )
  (8): AdaptiveAvgPool2d(output size=(1, 1))
)
(classifier): Linear(in features=512, out features=50, bias=True)
)
```

2. **Report accuracy of model on the validation set.**

   Accuracy: **0.8256**

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 the t-SNE visualization.**

   From the t-SNE results, we can see that the same category data (shown in the same color) will be grouped together in the feature space, while the different category data will be separated away.



Problem 2: Semantic Segmentation

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

```
FCN32(
  (backbone): 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)
  )
  (classifier): Sequential(
    (0): Conv2d(512, 4096, kernel size=(2, 2), stride=(1, 1))
    (1): ReLU()
    (2): Dropout2d(p=0.5, inplace=False)
    (3): Conv2d(4096, 4096, kernel size=(1, 1), stride=(1, 1))
    (4): ReLU()
    (5): Dropout2d(p=0.5, inplace=False)
    (6): Conv2d(4096, 7, kernel size=(1, 1), stride=(1, 1))
    (7): ConvTranspose2d(7, 7, kernel size=(64, 64), stride=(32, 32))
  )
)
```

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

   I implement *VGG16-FCN8s* model.

```
FCN8(
  (to 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)
  )
  (to pool4): 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)
  )
  (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)
  )
  (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, 256, kernel size=(8, 8), stride=(4, 4))
  )
  (pool4 upsample2): ConvTranspose2d(512, 256, kernel size=(2, 2), stride=(2, 2))
  (upsample8): ConvTranspose2d(256, 7, kernel size=(8, 8), stride=(8, 8))
)
```

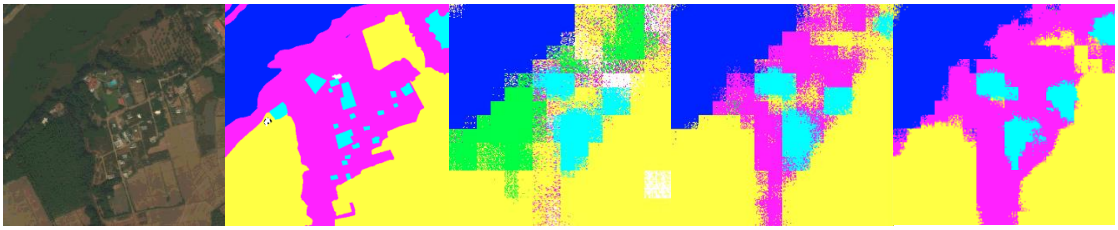
3. **Report mIoU of the improved model on the validation set.**

   mIoU: **0.7029**

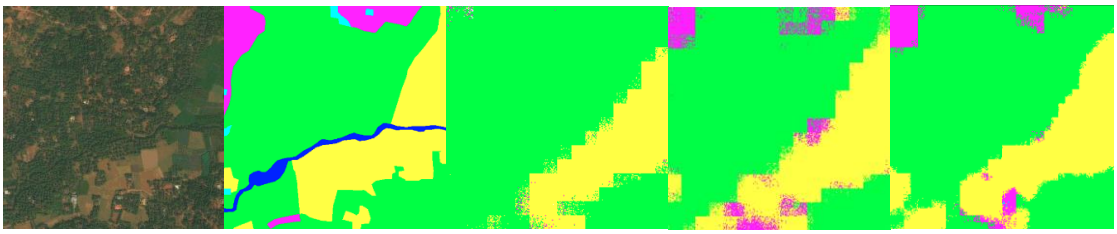
4. **Show the predicted segmentation mask of "validation/0010_sat.jpg", "validation/0097_sat.jpg", "validation/0107_sat.jpg" during the early, middle, and the final stage during the training process of this improved model.**

   I save the predicted segmentation masks in epoch 5, 10, 30. The following results are shown in the order: image, ground truth, epoch 5, epoch 10, epoch 30 respectively.

   - validation/0010_sat.jpg:



- validation/0097_sat.jpg:



- validation/0107_sat.jpg: