

Homework #1 Report

Deep Learning for Computer Vision

資工碩一 張凱庭 R10922178

Problem 1

1. Print the network architecture of your model.

In this task I implemented the Resnet50 network by using the torchvision's pretrained model as backbone and fine-tuning the model. During the training I freezed the weights before layer "(1): Bottleneck()", only tuning the rest.

Resnet50 architecture

```
PretrainedResnet(
  (features): 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): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (8): AdaptiveAvgPool2d(output_size=(1, 1))
)
(classifier): Sequential(
  (0): Linear(in_features=2048, out_features=50, bias=True)
)
```

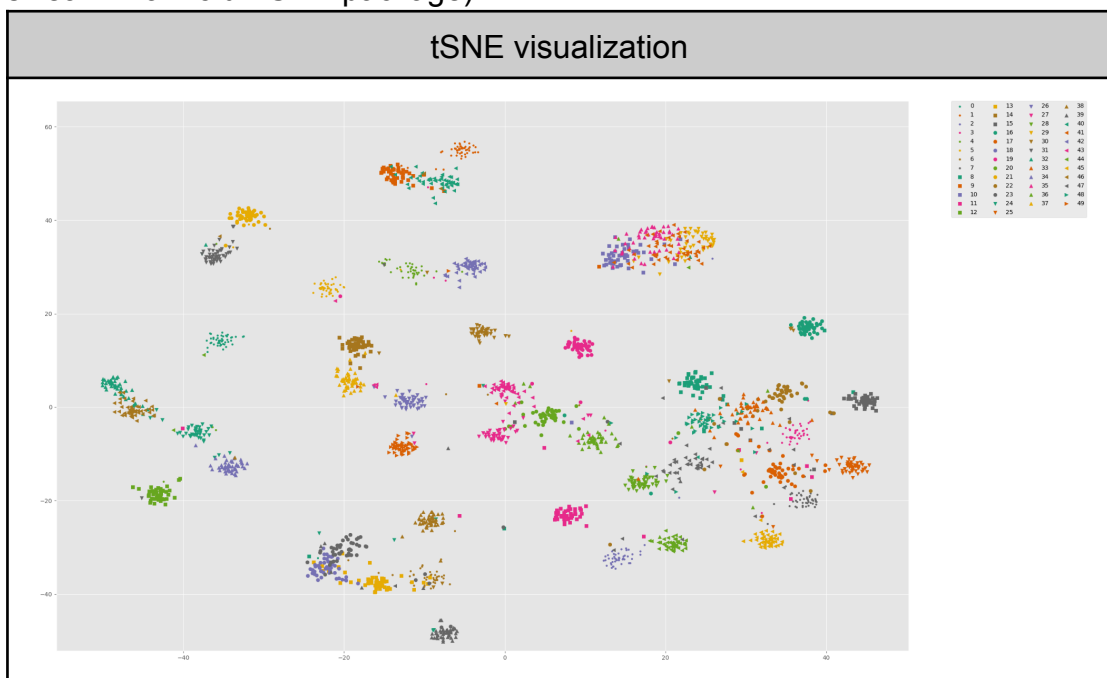
2. Report accuracy of model on the validation set.

I trained the model with 20 epochs, and it resulted best on the 19th epoch.

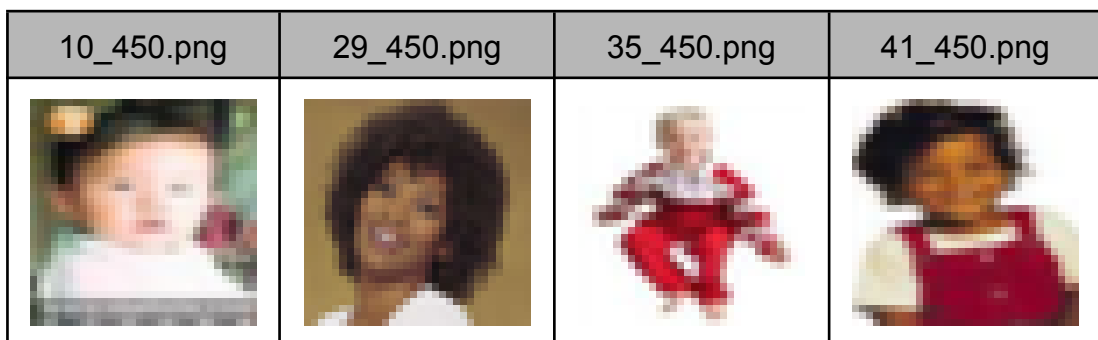
Pretrained Resnet50	0.862
---------------------	-------

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 tSNE visualization.

The features extracted from the second last layer has a *dimension of* 2048, and was then reduced to the *dimension of* 2 by the tSNE algorithm(using the sklearn.manifold.TSNE package).



As the above picture shows, most of the 50 classes were separated successfully. On the top right corner, the *class* 10, 29, 35, 41 was pretty close, it is probably because these classes are all human faces and the features of these classes are similar.



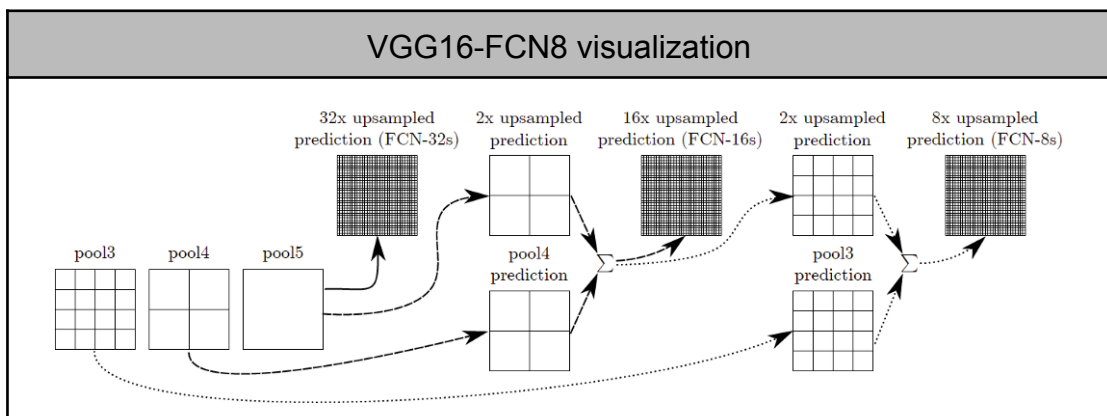
Problem 2

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

```
VGG16-FCN32s architecture

Vgg16FCN32(
  (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))
    (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)
  )
  (fcn6): Sequential(
    (0): Conv2d(512, 4096, kernel_size=(2, 2), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Dropout2d(p=0.5, inplace=False)
  )
  (fcn7): Sequential(
    (0): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Dropout2d(p=0.5, inplace=False)
  )
  (score_fr): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
  (upscore): ConvTranspose2d(7, 7, kernel_size=(64, 64), stride=(32, 32), bias=False)
)
```

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



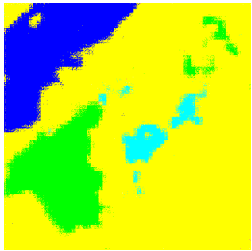
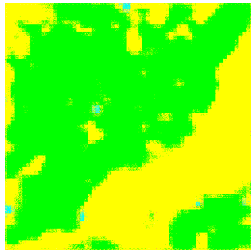
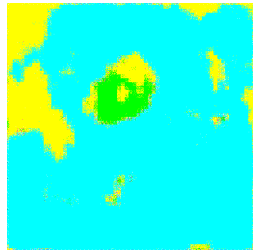
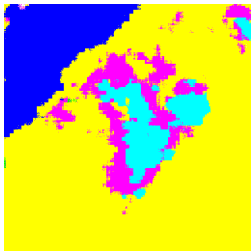
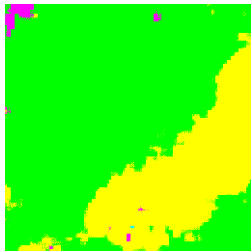
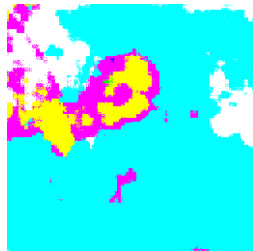
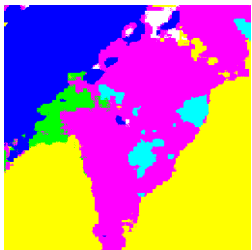
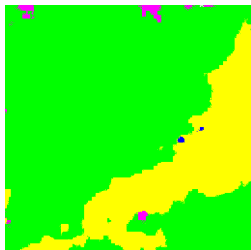
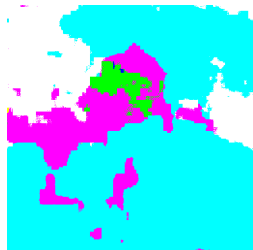
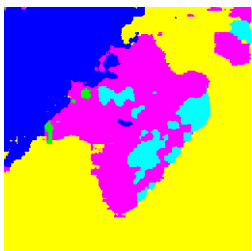
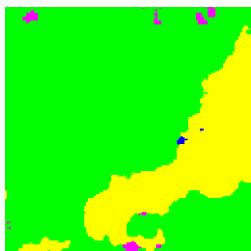
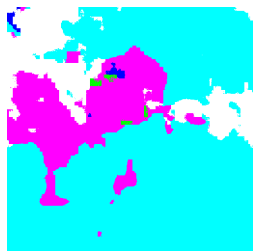

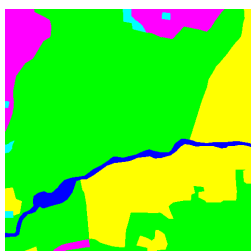
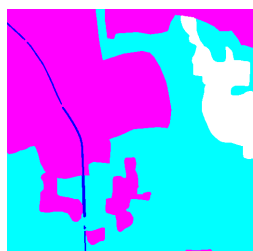
VGG16-FCN8s architecture

```
Vgg16FCN8(  
  (pool1_to_3): 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)  
  )  
  (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)  
  )  
  (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)  
  )  
  (fcn6): Sequential(  
    (0): Conv2d(512, 7, kernel_size=(1, 1), stride=(1, 1))  
    (1): ReLU()  
  )  
  (upsampling2): ConvTranspose2d(7, 7, kernel_size=(4, 4), stride=(2, 2), bias=False)  
  (upsampling8): ConvTranspose2d(7, 7, kernel_size=(16, 16), stride=(8, 8), bias=False)  
  (score_pool3): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))  
  (score_pool4): Conv2d(512, 7, kernel_size=(1, 1), stride=(1, 1))  
)
```

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

Vgg16-FCN32	0.676
Vgg16-FCN8	0.696

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.

	0010	0097	0107
Epoch 1			
Epoch 10			
Epoch 20			
Epoch 27			
Ground Truth			

Reference

1. [sklearn.manifold.TSNE](#)
2. medium: [Ideas on how to fine-tune a pre-trained model in PyTorch](#)
3. [torchvision.models](#)
4. FCN paper: <https://arxiv.org/abs/1605.06211>
5. github repository: [wkentaro/pytorch-fcn](#)
6. github repository: [IanTaehoonYoo/semantic-segmentation-pytorch](#)