

Problem 1: Self-supervised pre-training for image classification (50%)

1. (5%) Describe the implementation details of your SSL method for pre-training the ResNet50 backbone. (including but not limited to the name of the SSL method & data augmentation techniques you used, learning rate schedule, optimizer, and batch size setting for this pre-training phase)

- SSL method: BYOL (from Github)
- Data augmentation: None (only normalize and resize the data with configuration shown below)

```
TRANSFORM_IMG = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
])
```

- learning rate and scheduler: learning rate = $1e-4$ (without learning rate scheduler)
- optimizer: Adam
- batch size: 64

(20%) Please conduct the Image classification on Office-Home dataset as the downstream task. Also, please complete the following Table, which contains different image classification setting, and discuss/analyze the results.

使用了一層的 nn.Linear(1000, num_classes) 搭配 pretrained model 的 fc layer 作為 classifier

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation accuracy (Office-Home dataset)
A	-	Train full model (backbone + classifier)	<u>0.4828</u>
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	<u>0.4926</u>
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)	<u>0.5567</u>
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	<u>0.2266</u>
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	<u>0.2291</u>

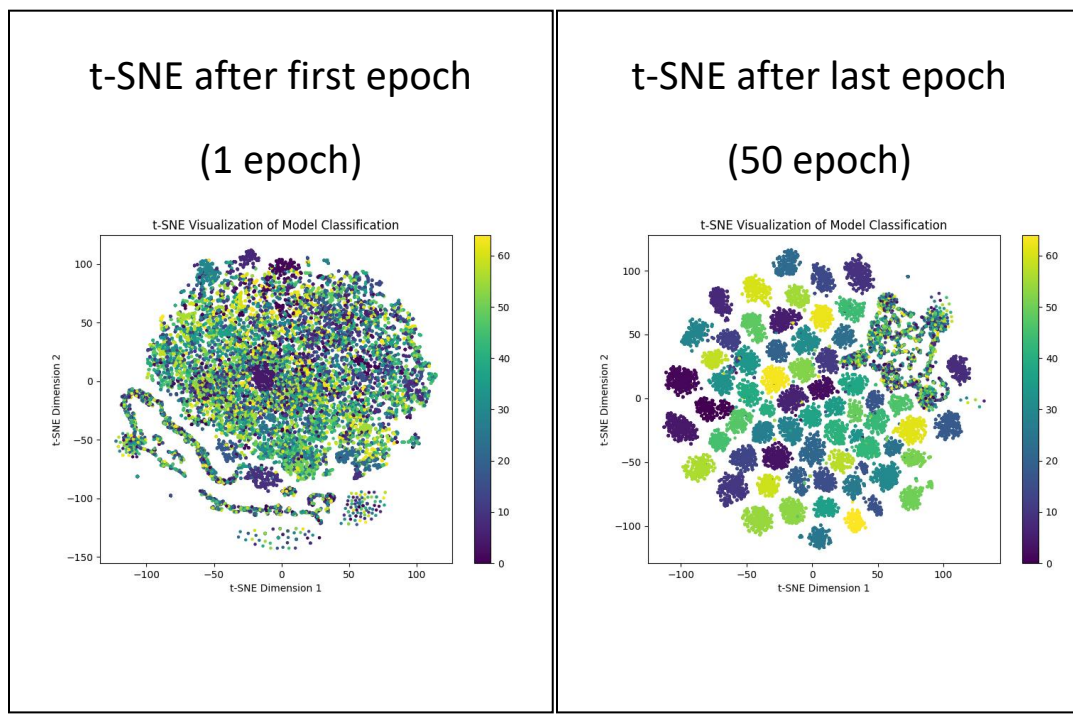
A 在訓練的時候準確率如下 a:		B 在訓練的時候準確率如下:		C 在訓練的時候準確率如下:	
5	epochs accuracy=0.3793	5	epochs accuracy=0.4039	5	epochs accuracy=0.4212
10	epochs accuracy=0.4163	10	epochs accuracy=0.4507	10	epochs accuracy=0.4631
15	epochs accuracy=0.4113	15	epochs accuracy=0.4261	15	epochs accuracy=0.4901
17	epochs accuracy=0.4828	20	epochs accuracy=0.4507	20	epochs accuracy=0.5345
20	epochs accuracy=0.4163	25	epochs accuracy=0.4236	25	epochs accuracy=0.5025
25	epochs accuracy=0.4483	30	epochs accuracy=0.4532	30	epochs accuracy=0.4877
30	epochs accuracy=0.4532	35	epochs accuracy=0.4606	35	epochs accuracy=0.4803
35	epochs accuracy=0.4335	38	epochs accuracy=0.4926	40	epochs accuracy=0.5567
40	epochs accuracy=0.4163	40	epochs accuracy=0.4754	45	epochs accuracy=0.4803
45	epochs accuracy=0.4138	45	epochs accuracy=0.4483	50	epochs accuracy=0.5296
50	epochs accuracy=0.4310	50	epochs accuracy=0.4433		

D 在訓練的時候準確率如下：			E 在訓練的時候準確率如下：		
5	epochs	accuracy=0.1995	5	epochs	accuracy=0.1404
10	epochs	accuracy=0.2044	10	epochs	accuracy=0.1773
15	epochs	accuracy=0.2044	15	epochs	accuracy=0.1724
20	epochs	accuracy=0.1921	20	epochs	accuracy=0.1798
25	epochs	accuracy=0.1675	25	epochs	accuracy=0.1995
30	epochs	accuracy=0.2167	30	epochs	accuracy=0.1897
35	epochs	accuracy=0.2044	35	epochs	accuracy=0.1946
40	epochs	accuracy=0.2020	40	epochs	accuracy=0.2020
45	epochs	accuracy=0.1921	44	epochs	accuracy=0.2291
48	epochs	accuracy=0.2266	45	epochs	accuracy=0.1946
50	epochs	accuracy=0.2143	50	epochs	accuracy=0.2044

根據以上的結果可以看到 A 的最高準確率雖然與 B、C 相近但其實只是剛好一個 epoch 運氣比較好而已，其他 epoch 均弱於 B、C，而 C 的 pretrained 因為是用 self-supervised 所以一開始（第 1 個 epochs）也弱於 B，但是之後不曉得為何會反超 B（features 沒有像 B 學的那麼好？所以有機會突破 local 最優？變成 global 最優？）。D、E 則是只有一層 nn.Linear 被 finetune 所以會遇到 pretrained 模型的 domain 限制有些 features 無法很好獲得因此只有較低的準確率。並且兩者無明顯差距。

3. (5%) Visualize the learned visual representation of setting C on the train set by implementing t-SNE (t-distributed Stochastic Neighbor Embedding) on the output of the second last layer. Depict your visualization from both the first and the last epochs. Briefly explain the results.

As the graph show, after 50 epochs from SSL pretrained model, the model success groups the different input to 65 classes that has been defined.



Problem 2: Semantic segmentation (50%)

1. (3%) Draw the network architecture of your VGG16-FCN32s model (model A).

Architecture of VGG16-FCN32s

```
VGG_FCN32(  
  (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)  
  )  
  (fcn32_conv): Sequential(  
    (0): Conv2d(512, 4096, kernel_size=(1, 1), stride=(1, 1))  
    (1): ReLU(inplace=True)  
    (2): Dropout(p=0.5, inplace=False)  
    (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))  
    (4): ReLU(inplace=True)  
    (5): Dropout(p=0.5, inplace=False)  
  )  
  (fcn32_upsampling): Sequential(  
    (0): ConvTranspose2d(4096, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace=True)  
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (5): ReLU(inplace=True)  
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (8): ReLU(inplace=True)  
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (11): ReLU(inplace=True)
```

```
(12): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
(13): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(14): ReLU(inplace=True)
(15): Conv2d(32, 7, kernel_size=(1, 1), stride=(1, 1))
)
)
```

2. (3%) Draw the network architecture of the improved model (model B) and explain it differs from your VGG16-FCN32s model.

DeepLabV3_ResNet101 使用了 ResNet101 作為其 backbone 而非 VGG16，并且使用 Dilated Convolution 來節省運算資源，同時擁有完整的 pretrained model 可以使用，進一步提高了準確率。

Architecture of DeepLabV3 ResNet101

```
DeepLab(  
  (model): DeepLabV3(  
    (backbone): IntermediateLayerGetter(  
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)  
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)  
      (layer1): 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)  
        (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)  
      )  
    )  
    (layer2): Sequential(  
      (0): Bottleneck(  
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)  
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```

(conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
(bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=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): Bottleneck(
  (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
  (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(2): Bottleneck(
  (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
  (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(3): Bottleneck(
  (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
  (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
)
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)

```


[illegible]

[illegible]

```
(bn2): BatchNorm2d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
)
(18): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(19): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(20): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(21): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(22): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
)
(layer4): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(1024, 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=(2, 2), dilation=(2, 2), 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)
    (downsample): Sequential(
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(1, 1), 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=(4, 4), dilation=(4, 4), 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=(4, 4), dilation=(4, 4), 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)
)
)
)
(classifier): DeepLabHead(
  (0): ASPP(
    (convs): ModuleList(
      (0): Sequential(
        (0): Conv2d(2048, 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)
        (2): ReLU()
      )
      (1): ASPPConv(
        (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(12, 12), dilation=(12, 12), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU()
      )
      (2): ASPPConv(
        (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(24, 24), dilation=(24, 24), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU()
      )
      (3): ASPPConv(
        (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(36, 36), dilation=(36, 36), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU()
      )
      (4): ASPPPooling(
        (0): AdaptiveAvgPool2d(output_size=1)
        (1): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (3): ReLU()
      )
    )
  )
  (project): Sequential(
    (0): Conv2d(1280, 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)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
  )
  )
  (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): ReLU()
  (4): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
)
(aux_classifier): FCNHead(
  (0): Conv2d(1024, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU()
  (3): Dropout(p=0.1, inplace=False)
)

```

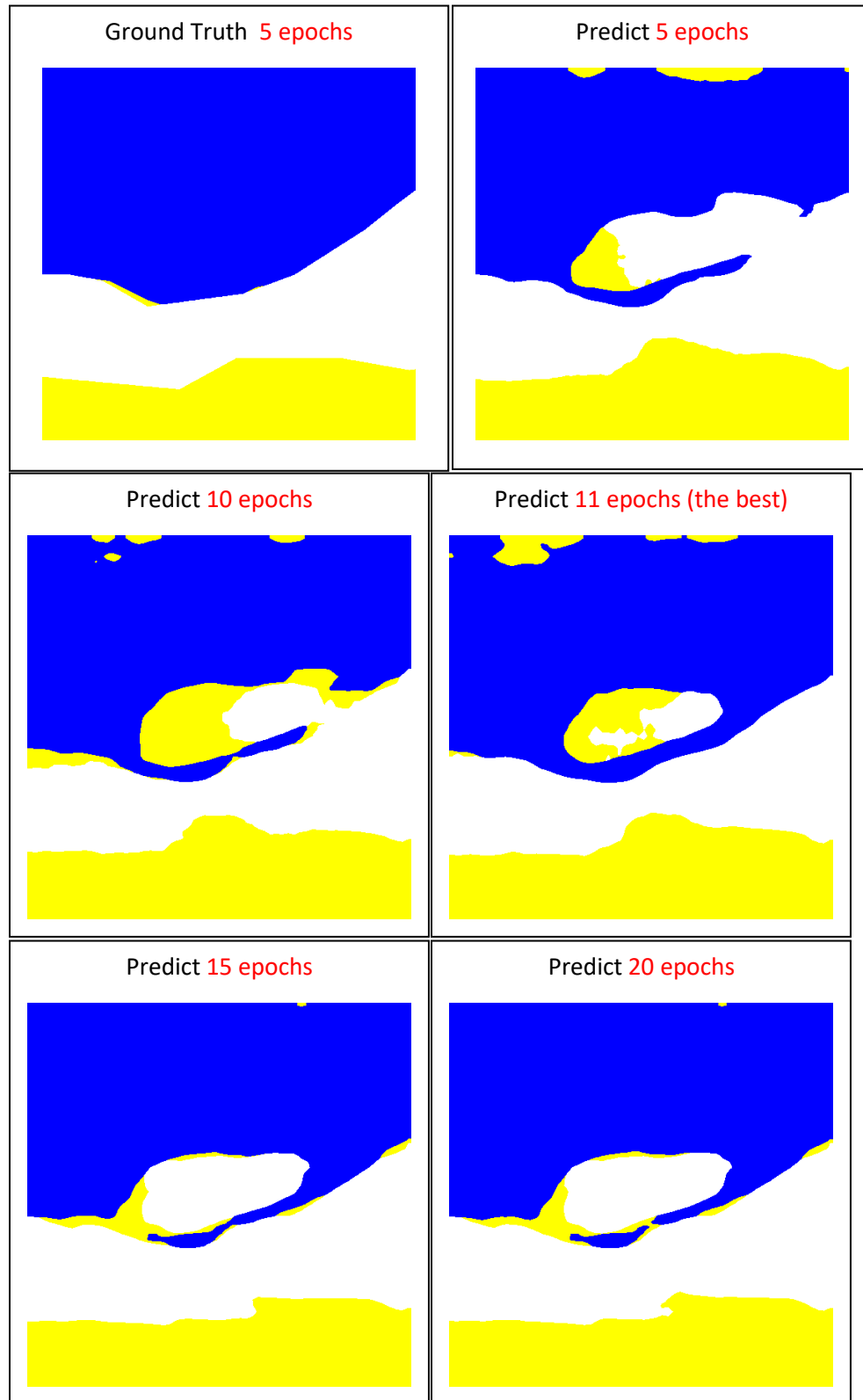
```
(4): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

3. (1%) Report mIoUs of two models on the validation set.

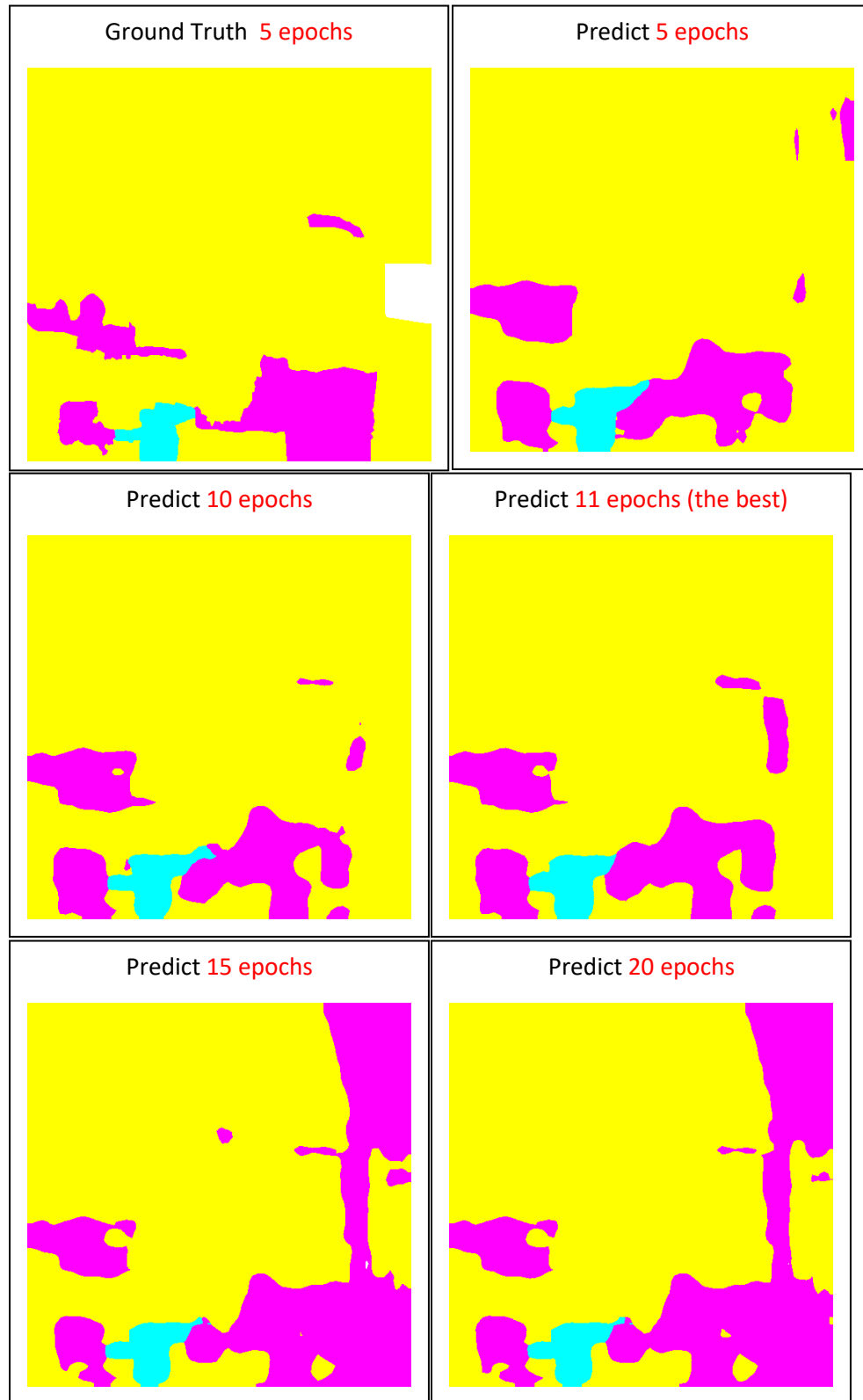
VGG16-FCN32s:	<u>0.7049</u>
DeepLabV3_ResNet101:	<u>0.7609</u>

4. (3%) Show the predicted segmentation mask of “validation/0013_sat.jpg”, “validation/0062_sat.jpg”, “validation/0104_sat.jpg” during the early, middle, and the final stage during the training process of the improved model.

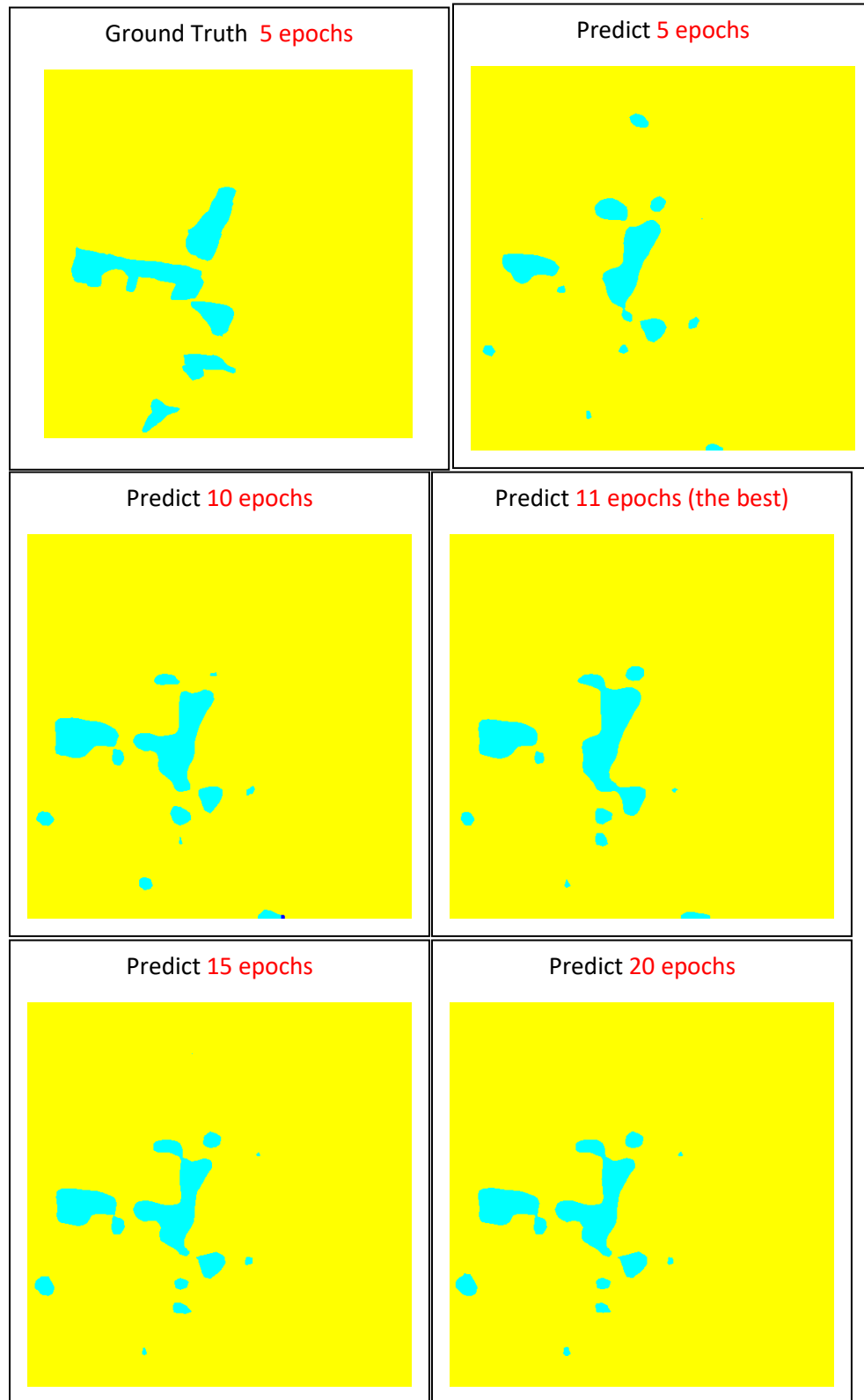
Inference compare of validation/0013_sat.jpg



Inference compare of validation/0062_sat.jpg



Inference compare of validation/00104_sat.jpg



5. Use segment anything model (SAM) to segment three of the images in the validation dataset, report the result images and the

Generate from

https://github.com/facebookresearch/segment-anything/blob/main/notebooks/automatic_mask_generator_example.ipynb

Validation/0013_sat.jpg(Origin)



Validation/0013_sat.jpg(SAM)



Validation/0062_sat.jpg(Origin)



Validation/0062_sat.jpg(SAM)



Validation/0104_sat.jpg(Origin)



Validation/0104_sat.jpg(SAM)

