

Problem 1: Self-supervised pre-training for image classification

Report (30%)

1. (5%) Describe the implementation details of your **SSL method** for pre-training the ResNet50 backbone.

Ans: For SSL method, I pretrain the ResNet50 backbone by implementing BYOL(Bootstrap your own latent) framework. The detail settings are shown below:

```
learner = BYOL(  
    model,  
    image_size = 128,  
    hidden_layer = 'avgpool',  
    use_momentum = False      # turn off momentum in the target encoder  
)  
optimizer = torch.optim.Adam(learner.parameters(), lr=lr)
```

, where the learning rate are set to 0.001 by default, and the model is ResNet50 without using pretrained weight.

As for the dataset transformation, I used random flip, random crop, ... for data augmentation (shown below).

```
self.transform = transforms.Compose([  
    transforms.Resize(128),  
    transforms.RandomCrop(128),  
    transforms.RandomHorizontalFlip(),  
    transforms.RandomVerticalFlip(),  
    transforms.TrivialAugmentWide(),  
  
    transforms.ToTensor(),  
    transforms.Normalize( mean=[0.485, 0.456, 0.406],  
                           std =[0.229,0.224,0.225] )  
) # by hw1 constrain
```

And for other parameters, the batch sized is set to 64, the pretrained epoch is set to 10.

2. (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.

Ans:

The result table are shown below. It shows that setting B (pretrain using supervised learning) has the best validation accuracy, while the setting C has the second-best validation accuracy with a 5% accuracy difference, which implies that pretraining using SSL has the potential to approach the performance of supervised learning method, saving the labor of data labeling at the same time.

As for the setting D and E, which freeze the backbone and train only on classifier during finetuning, has the worse result compared to Setting B and C(do not freeze backbone), which imply that the domain difference between the pretrain dataset and the finetune dataset can affect the model performance greatly.

Finally, the setting A get a drastically better result compared to the setting D and E but close to setting B and C, which might also suggest the influence of domain difference between the pretrain and finetune dataset and the need to adjust parameters in pretrained backbone model.

Setting	Validation Accuracy (best checkpoint)
A	42.11%
B	49.75%
C	41.13%
D	28.33%
E	11.08%

(all model are fintuned under the following setting: epoch 200, batch_size 128 learning_rate 0.0001, weight_decay 0.0001, and the classifer are all the same (nn.Conv1d(2048, 256, 1, stride=1),.ReLU(), nn.Dropout(0.5), and nn.Linear(256, 65)))

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.

Ans:

The result plot of setting C about PCA and t-SNE are shown below.

According to the plot in Fig 1. and Fig.2, the distribution of the output in the same class becomes more condense and neater (i.e. dots with the same color are getting closer to each other) for each class across the training process.

As for the t-SNE plot shown in Fig. 3. and Fig.4.,throughout the training process,

not only the distribution of each class becomes more condense, but also the distribution between classes becomes more discrete (i.e. cluster of each class are less overlapped).

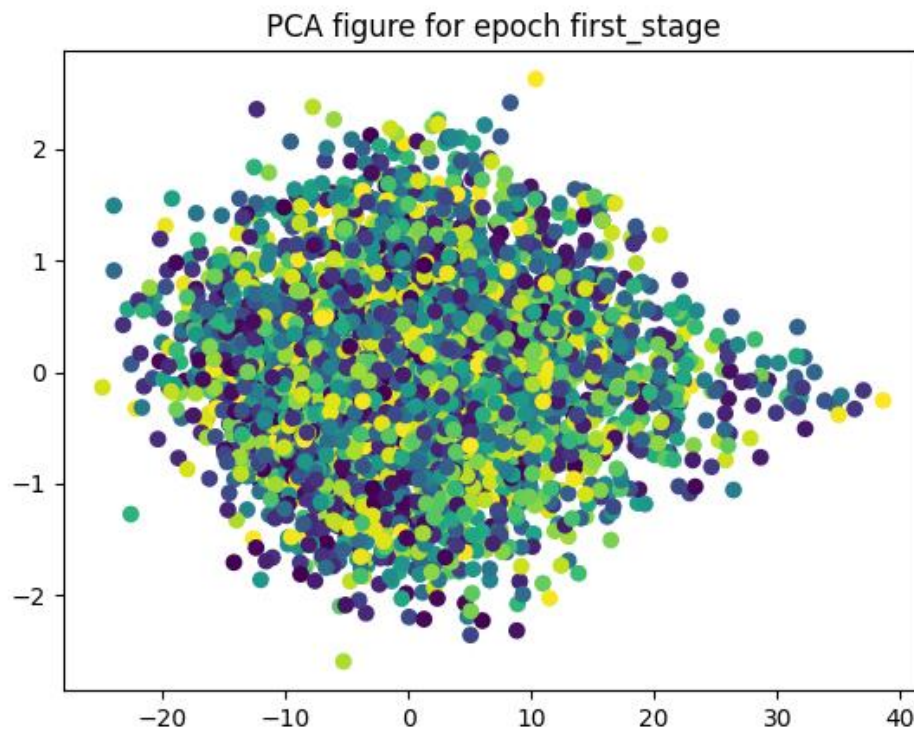


Figure 1. The PCA plot of embedding at the last second layer (first stage)

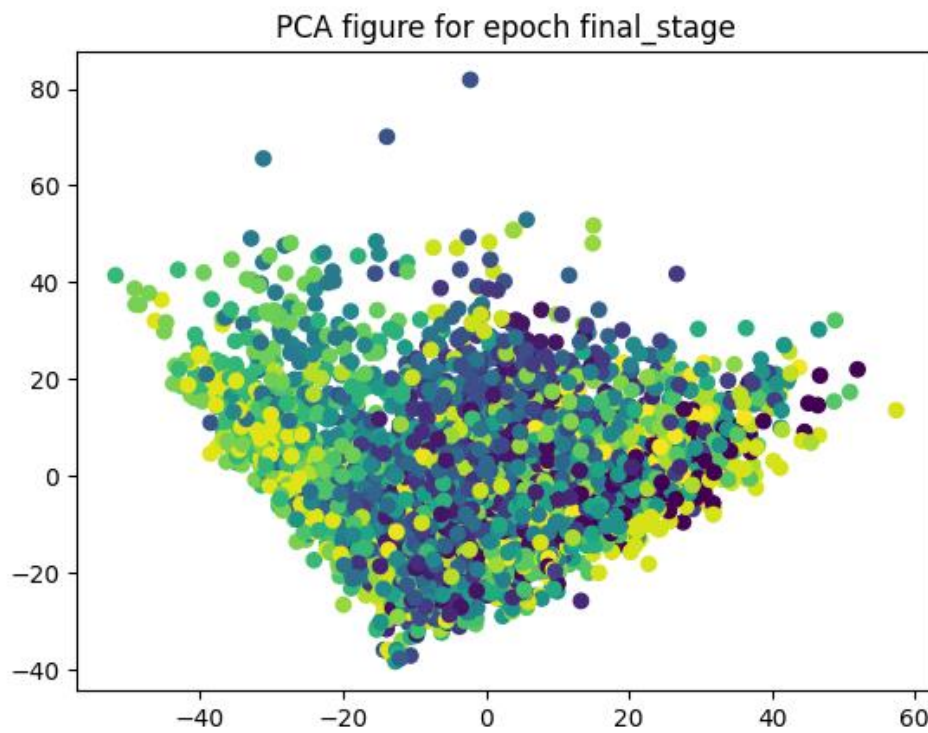


Figure 2. The PCA plot of embedding at the last second layer (final stage)

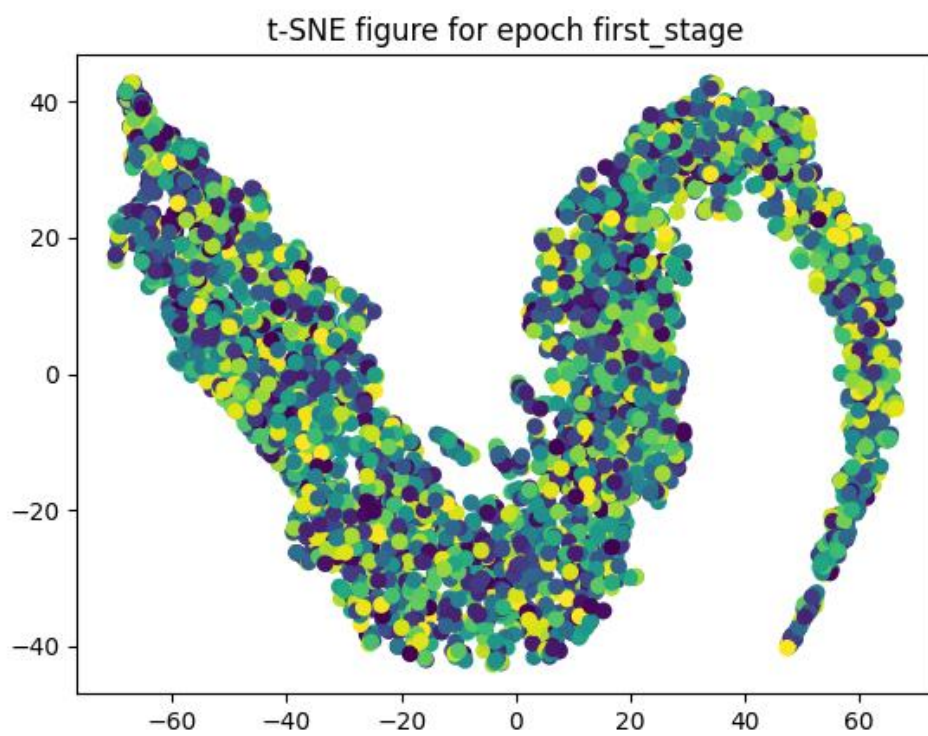


Figure 3. The t-SNE plot of embedding at the last second layer (first)

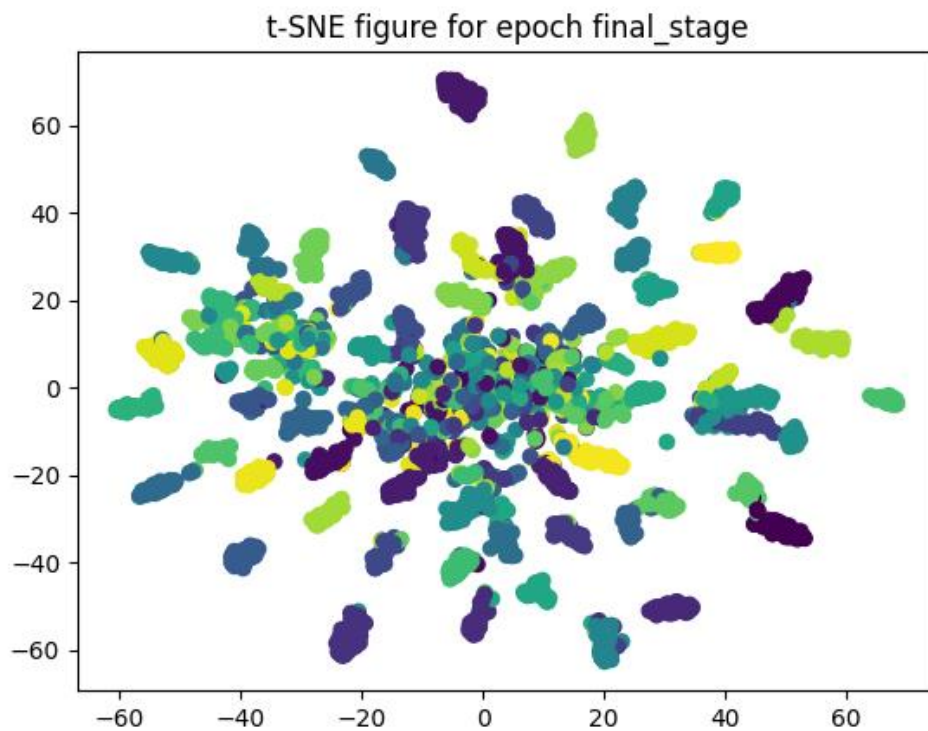


Figure 4. The t-SNE plot of embedding at the last second layer (last)

Model Performance (20%)

Problem 2: Semantic segmentation

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

```

VGG16_FCN32s(
  (vgg16): 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)
  )
  (conv6_1): Sequential(
    (0): Conv2d(512, 4096, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Dropout2d(p=0.5, inplace=False)
  )
  (conv7_1): Sequential(
    (0): Conv2d(4096, 4096, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Dropout2d(p=0.5, inplace=False)
  )
  (score_pred): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
  (upsample): ConvTranspose2d(7, 7, kernel_size=(32, 32), stride=(32, 32))
)

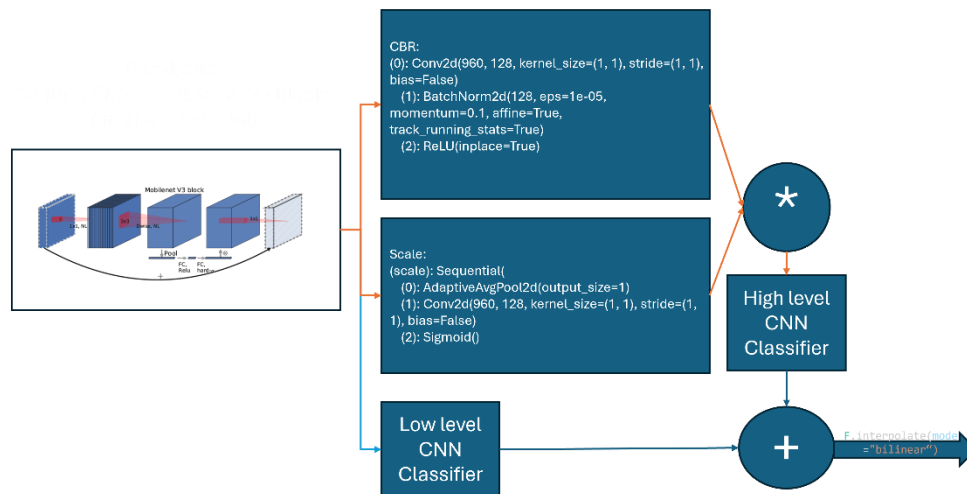
```

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

Ans:

I use lraspp_mobilenet_v3_large (<https://arxiv.org/abs/1905.02244>), a Lite R-ASPP Network model with a MobileNetV3-Large backbone on semantic segmentation task, and it is available on pytorch with pretrained weight on MS-COCO.

(https://pytorch.org/vision/0.18/models/generated/torchvision.models.segmentation.lraspp_mobilenet_v3_large.html).



These difference between these two model is that, the first model(VGG16-FCN32s) uses basic convolution layers(VGG16) to extract features and finally use a `conv2dTranspose()` to do a one step upsampling (FCN32s), while the second model use MobileNet V3 block as backbone, which implement Inverted Residual and Linear Bottleneck with Squeeze-and-Excite techniques in order to reduce the inference latency without lowering the accuracy, and use `interpolate(bilinear)` for upsampling.

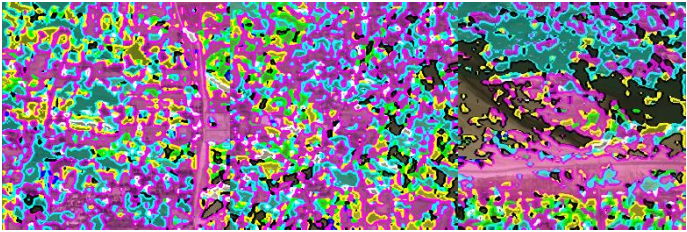


3. (1%) Report mIoUs of two models on the validation set.

model	mIOU
VGG16_FCN32s	68.67%
Deeplabv3_Resnet101	71.70%

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**.

Stage	Image with mask
-------	-----------------

Early	
Middle	
Final	

5. (10%) Use segment anything model (SAM) to segment three of the images in the validation dataset, report the result images and the method you use.

Ans:

Method: upload the image in “hw1_data/p2_data/validation” (0001_sat.jpg, 0002_sat.jpg and 0003_sat.jpg) to <https://segment-anything.com/demo#> and click “Everything” button on the left side of the web page.

Segment Anything

Research by Meta AI

[Home](#)
[Demo](#)
[Dataset](#)
[Blog](#)
[Paper](#)


Tools

Upload

Gallery

Hover & Click

Box

Everything


Find all the objects in the image automatically.

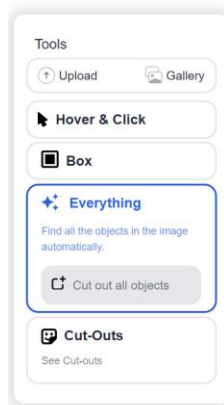
Cut out all objects

Cut-Outs

See Cut-outs

Final points after de-duplicating predicted masks





Interested in learning more? Check out the [Paper](#), [Blog Post](#), or [Code](#).



Figure 5. method of using SAM to implement sematic segmentation

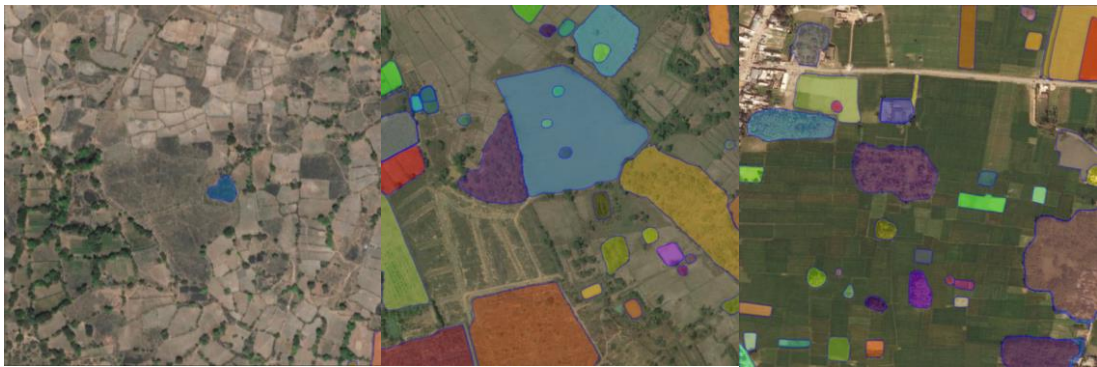


Figure 6. result of SAM using data 0001_sat.jpg, 0002_sat.jpg and 0003_sat.jpg