DLCV HW1 Report

R11943004 黃子青

Problem 1: Self-Supervised Pre-training for Image Classification

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)

I used the SSL method from the TA-provided GitHub repo BYOL (https://github.com/lucidrains/byol-pytorch) to pre-train ResNet50 with the Mini-ImageNet dataset.

The implementation details are as follows:

Model: Resnet50

Data augmentation:

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

Learning rate scheduler:

```
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(opt, T_max=500, eta_min=0)
```

Optimizer:

```
opt = torch.optim.Adam(learner.parameters(), lr=3e-4)
```

Batchsize: 128

Total training epochs:500

I will record the smallest loss value during the training process and set the current model as the best model accordingly.

```
# 計算平均損失
avg_loss = epoch_loss / len(dataloader)
print(f"Epoch [{epoch+1}/100] - Avg Loss: {avg_loss:.4f}")

# 如果當前平均損失小於最小損失，則保存模型
if avg_loss < best_loss:
    best_loss = avg_loss
    torch.save(resnet.state_dict(), save_path)
    print(f"Model saved with loss: {best_loss:.4f}")
```

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.

Setting	Pre-training	Fine-tuning	Validation accuracy
A		Train full model (backbone + classifier)	0.5025
B	w/ label	Train full model (backbone + classifier)	0.5961
C	w/o label	Train full model (backbone + classifier)	0.5296
D	w/ label	Fix the backbone. Train classifier only	0.2956
E	w/o label	Fix the backbone. Train classifier only	0.2217

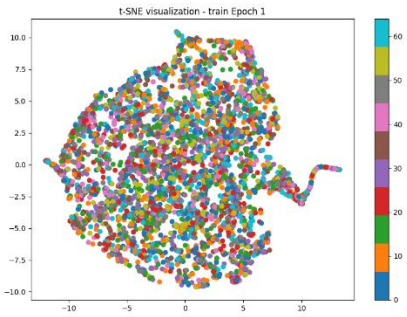
The three fine-tuning methods for model, A, B, and C, perform significantly better than the two methods, D and E, which only fine-tune the classifier. This indicates that when transferring to a new dataset, the backbone network needs adjustment to adapt to the new task's requirements.

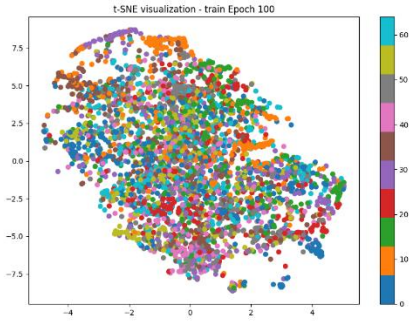
Based on the results of settings A and B/C, pre-training is crucial for improving the performance of image classification models. Whether supervised or self-supervised, pre-training allows the model to achieve better results on new datasets.

The comparison between settings B and C demonstrates that models with supervised pre-training learn more task-relevant features, thus performing better. While self-supervised pre-training helps with initialization, its effect is still not as strong as that of supervised learning.

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.

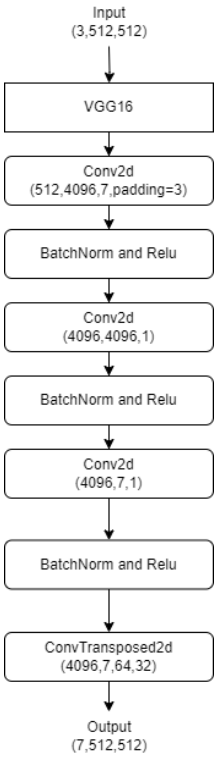
The output of the second-to-last layer is 1000-dimensional, and after reducing it to 2 dimensions using t-SNE, it can be observed that as the training reaches the final epoch, the clustering effect becomes more distinct compared to the first epoch. However, since the accuracy is only around 53%, the improvement in clustering is not visually apparent.





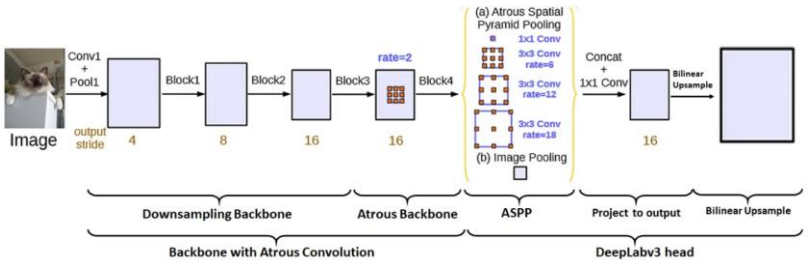
Problem 2: Semantic Segmentation

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



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

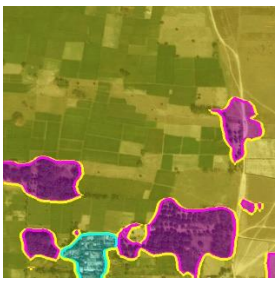
   I used the Deeplab v3+Resnet101 architecture, with the following advantages: (1) ASPP: It uses dilated convolutions with different rates to capture information from different spatial scales(VGG16-FCN32s only up-sample one feature map (1/32) to get segmentation result), which are then concatenated and passed to the next layer. (2) Aux-loss: The intermediate hidden layer outputs are also used to make predictions on the input, and this loss is incorporated into the parameter update formula, helping to mitigate the impact of vanishing gradients.
   and the accuracy reported by two models also shows that model B surpass model A



Ref: https://medium.com/@itberrios6/deeplabv3-c0c8c93d25a4

3. **(1%) Report mIoUs of two models on the validation set.**
   mIOU of model A: 0.6526
   mIOU of model B: 0.7519

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.**

	Early	Middle	Final	Groundtruth
0013_sat.jpg				
0062_sat.jpg				
0104_sat.jpg				

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**.
I used the pre-trained model 'sam_vit_h_4b8939.pth' from the TA-provided official SAM GitHub repository and referred to the automatic_mask_generator_example.ipynb for performing inference on 0013_sat.jpg, 0062_sat.jpg, and 0104_sat.jpg.
ref:(https://github.com/facebookresearch/segment-anything/blob/main/notebooks/automatic_mask_generator_example.ipynb).

```python
# 選擇 SAM 模型並加載權重
model_type = "vit_h"  # 你可以選擇 'vit_h', 'vit_l', 或 'vit_b'
sam_checkpoint = "sam_vit_h_4b8939.pth"  # SAM 預訓練權重的檔案路徑
sam = sam_model_registry[model_type](checkpoint=sam_checkpoint).to(device)

# 加載測試圖像
image_path = ['../hw1_data/p2_data/validation/0013_sat.jpg','../hw1_data/p2_data/validation/0062_sat.jpg','../hw1_data/p2_data/validation/0104_sat.jpg']
for i in range(len(image_path)):
    image = Image.open(image_path[i])
    image = np.array(image)

    # 定義輸出目錄
    output_dir = 'sam_test_output'
    os.makedirs(output_dir, exist_ok=True)

    # 將 SAM 模型移動到 GPU
    sam.to(device=device)

    # 使用 SAM 自動生成掩膜
    mask_generator = SamAutomaticMaskGenerator(sam)
    masks = mask_generator.generate(image)

    # 展示並保存生成的掩膜
    show_anns(masks, idx=i, output_dir=output_dir)

    print(f"Inference complete and results saved in {output_dir}!")
```

I made sure that the output colors match the specifications of P2. However, since SAM doesn't know what class it's segmenting, you may notice that different class areas might have the same color.

```python
# 定義 7 種顏色
cls_color = [
    [0, 255, 255],    # Cyan: Urban land
    [255, 255, 0],    # Yellow: Agriculture land
    [255, 0, 255],    # Purple: Rangeland
    [0, 255, 0],      # Green: Forest land
    [0, 0, 255],      # Blue: Water
    [255, 255, 255],  # White: Barren land
    [0, 0, 0]         # Black: Unknown
]
```

The segmentation of:

0013_sat.jpg: 0062_sat.jpg 0104_sat.jpg