HW1 Report

Hsuan-Ting Lin

B11901132

1 P1: Self-Supervised Pretraining for Image Classification

1.1 Implementation Details of SSL

For my self-supervised learning approach, I implemented Bootstrap Your Own Latent (BYOL) [1], leveraging its ability to learn representations without negative pairs. I used the PyTorch implementation available on GitHub[1].

1.1.1 Data Augmentation

A series of data augmentations were applied to enhance the robustness of our model. The augmentation pipeline is as follows:

```
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomResizedCrop(128),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

This augmentation pipeline includes resizing, random cropping, horizontal flipping, color jittering, and normalization, which help in creating diverse views of the same image.

1.1.2 ResNet50 Backbone Training

The ResNet50 backbone was trained with the following parameters and optimizer:

```
num_epochs = 300
batch_size = 256
lr = 0.001
weight_decay = 0.01

opt = torch.optim.AdamW(learner.parameters(), lr=lr, weight_decay=weight_decay)
```

The BYOL learner was initialized with the 'avgpool' layer of ResNet50, whose output is used as the latent representation used for self-supervised training.

[1] https://github.com/lucidrains/byol-pytorch

1.1.3 Training Process

A checkpoint system was implemented to save the best model based on the lowest loss achieved during training. To monitor the training progress, I plotted the training loss over epochs, providing visual insight into the model's learning curve.

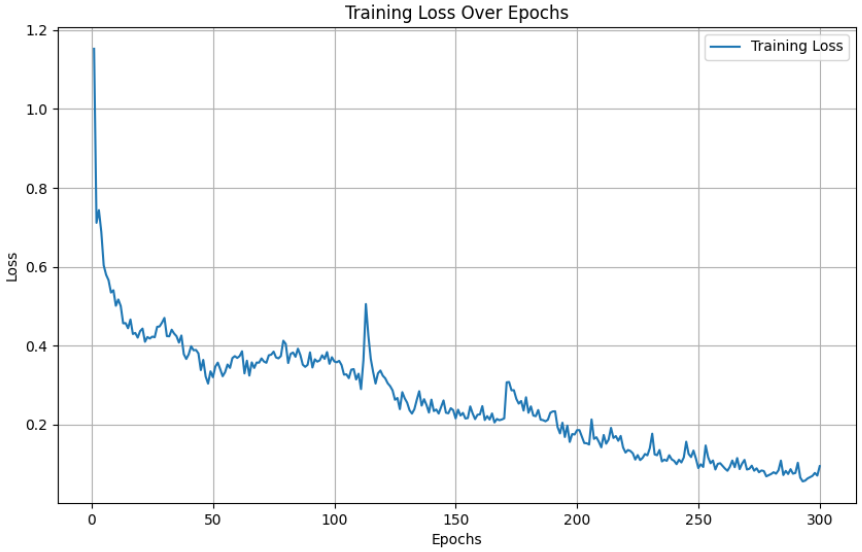

Figure 1: ResNet50 pretraining loss over epochs

1.2 Image Classification on Office-Home

1.2.1 Classifier Architecture

The FC layer of ResNet50 after pretraining was discarded ,and I added my own FC layers instead. The full classifier architecture is shown as below where the number of classes to classify is 7:

```
self.backbone = torchvision.models.resnet50(weights=None)
self.backbone.fc = torch.nn.Sequential(
    torch.nn.Linear(2048, 1024),
    torch.nn.BatchNorm1d(1024),
    torch.nn.Dropout(0.4),
    torch.nn.ReLU(),
    torch.nn.Linear(1024, 512),
    torch.nn.BatchNorm1d(512),
    torch.nn.Dropout(0.3),
    torch.nn.ReLU(),
    torch.nn.Linear(512, num_classes)
)
```

1.2.2 Performance Across Different Settings

I conducted image classification using my classification network that was fine-tuned on the training set of the Office-Home dataset. The validation accuracy across different fine-tune settings are shown in the table below.

Setting	Pre-training	Fine-tuning	Val Accuracy
A	-	Full model	45.8%
B	w/ label (TA provided backbone)	Train full model	52.0%
C	w/o label (Own backbone)	Train full model	48.5%
D	w/ label (TA provided backbone)	Train classifier only	37.2%
E	w/o label (Own backbone)	Train classifier only	16.7%

Table 1: Experiment results for different settings

The results indicate that models pre-trained with TA-provided backbones using supervised learning on Mini-ImageNet significantly outperform those trained using the self-supervised BYOL approach. This demonstrates that supervised pre-training typically yields stronger feature representations for downstream tasks.

Models that were fully fine-tuned (settings A, B, and C) outperformed those that only trained classifiers (settings D and E), underscoring the importance of end-to-end fine-tuning to maximize performance, as pre-training alone may not be sufficient.

1.3 t-SNE Visualization

Although not quite obvious due to the large number of classes and the vinicity of dots, some clusters are formed in the visualizations of the learned representations at the last epoch. It shows that after training, the representations of the images in the same class start to form similar representations. The effect would be more prominent if I continue training the model until the accuracy on training set reaches near 100% (I only trained to around 70%).

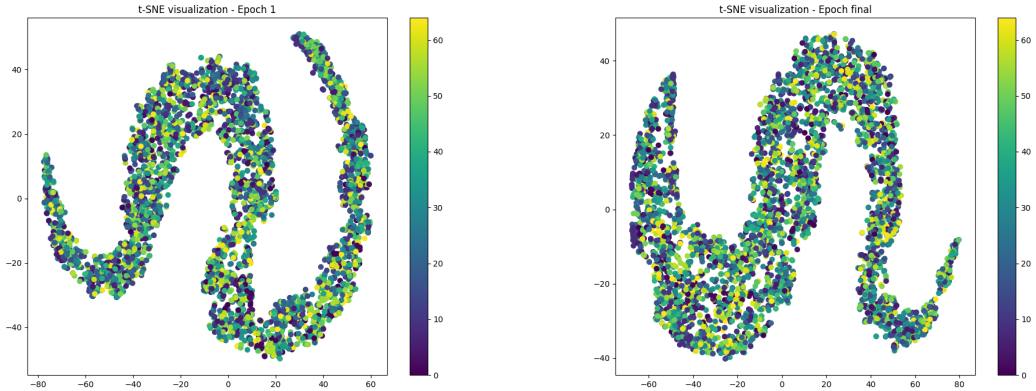

Figure 2: t-SNE visualizations at Epoch 1 (left) and Epoch 100 (right).

2 P2: Semantic Segmentation

2.1 Network Architecture of Model A (VGG16-FCN32s)

Figure 3: Visualizations of VGG16-FCN32s

2.2 Network Architecture of Model B (ResNet50-DeepLabV3)

I simplified the DeepLabV3 model solely for visualization by reducing the number of bottleneck blocks, omitting batch normalization and ReLU layers, and condensing the ASPP module. These changes maintain the core architecture while making the visualization more interpretable.

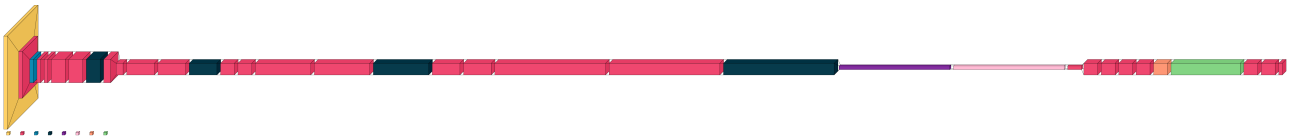

Figure 4: Simplified visualizations of ResNet50-DeepLabV3

The VGG16-FCN32s model is simpler, using a VGG16 backbone with fully convolutional layers and transposed convolution for upsampling, focusing on dense predictions from the start. In contrast, the DeepLabV3 model uses a deeper ResNet backbone with bottleneck blocks and an ASPP module to capture multi-scale context, followed by upsampling.

2.3 mIoUs of Models A and B on Validation Set

Model	**Description**	**Val mIoU (%)**
A	VGG16-FCN32s	73.4%
B	ResNet50-DeepLabV3	75.9%

Table 2: Comparison of mIoU on validation set for Model A and B

2.4 Predicted Segmentation Masks During Training

This is a comparison of the original images and their corresponding predicted segmentation masks at different stages of training: early, middle, and final stages. The images selected for comparison include images 0013, 0062, and 0104 from the validation set.

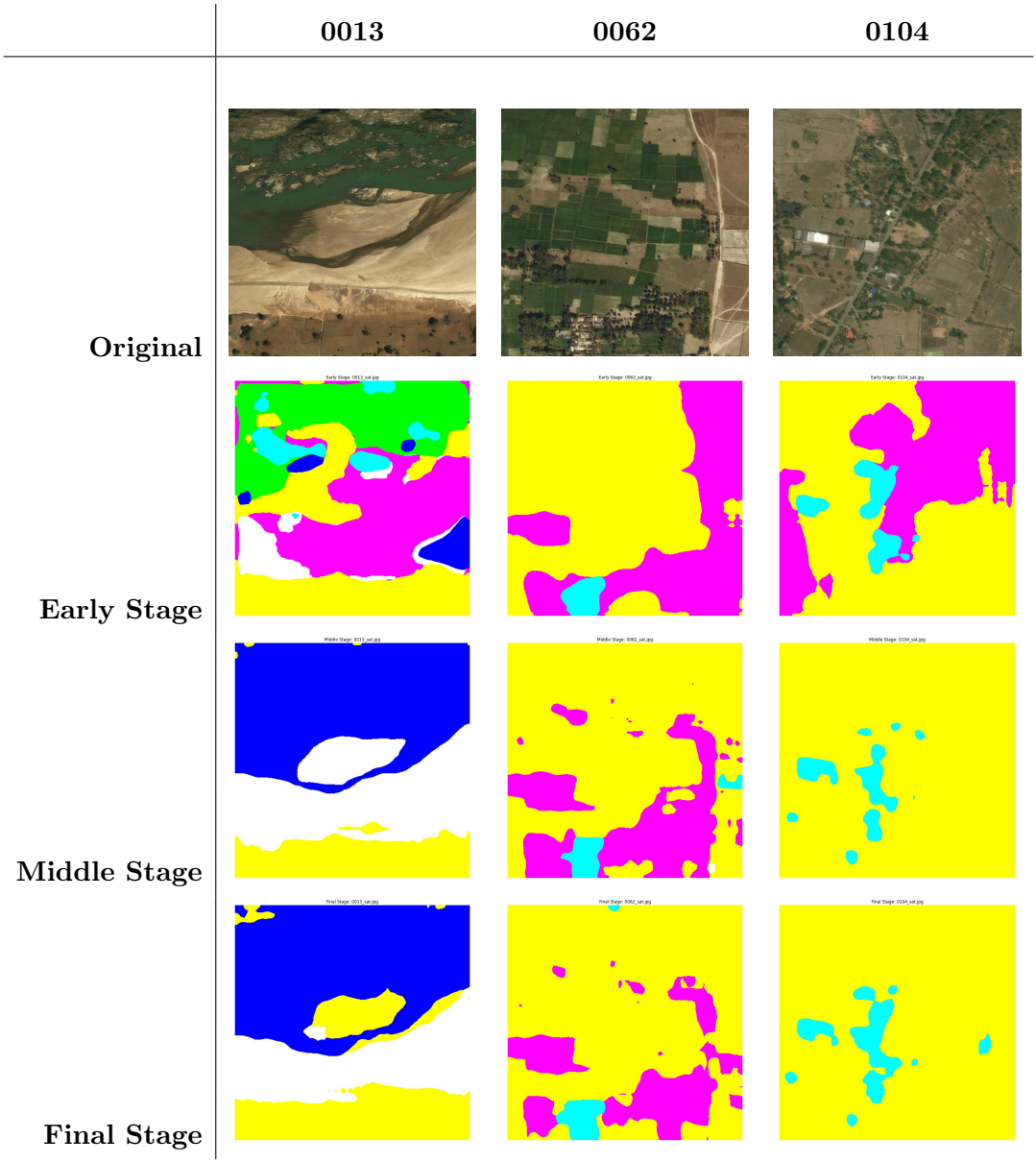

Figure 5: Comparison of original images and predicted segmentation masks at different training stages for images 0013, 0062, and 0104.

2.5 SAM Segmentation

I used the pre-trained SAM ViT-H model from Meta AI [2] to generate masks for input images loaded via OpenCV, configured with default parameters such as 32 points per side, a prediction IOU threshold of 0.88, and a minimum mask area of 100. The generated masks, overlaid with random colors for visualization, were sorted by area to prioritize larger segments.

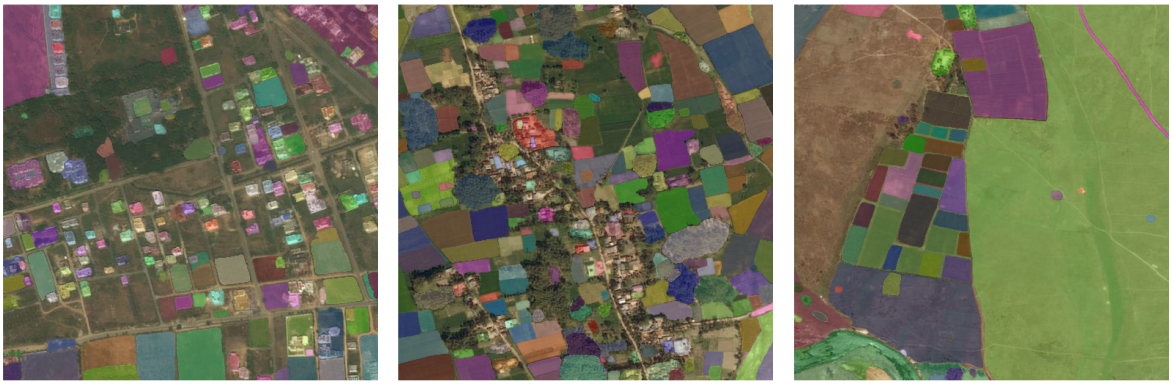

Figure 6: SAM segmentation of three images in the validation set

Acknowledgments

Some of the code in this project was generated with assistance from `ChatGPT` [3] and `Claude` [4]. Their contributions include code for model training, model evaluation, and general Python scripting.

References

[1] P. Wang, "byol-pytorch," https://github.com/lucidrains/byol-pytorch, 2020, accessed: 2024-09-27.

[2] facebookreasearch, "segment-anything," https://github.com/facebookresearch/segment-anything, 2023, accessed: 2024-09-27.

[3] OpenAI, "Chatgpt: Gpt-4 model," `https://chat.openai.com`, 2024, `https://chat.openai.com`.

[4] Anthropic, "Claude: A large language model by anthropic," `https://www.anthropic.com`, 2024, `https://www.anthropic.com`.