DLCV HW1

NTU, FALL 2024

B10901151 林祐群
指導教授： 王鈺強
2024年9月12日

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)

In the pre-training phase of the ResNet50 backbone, I implemented the **BYOL (Bootstrap Your Own Latent)** self-supervised learning (SSL) method. Here's an in-depth look at the implementation details:

1. SSL Method: BYOL (Bootstrap Your Own Latent)

- **BYOL** is a self-supervised learning method that does not rely on negative pairs like contrastive learning. Instead, it learns by comparing two augmentations of the same image through a dual-encoder architecture—a main network and a target network—that progressively update each other. This encourages the model to learn meaningful representations without labels.

2. Data Augmentation Techniques

To ensure robustness and variability in the learned representations, I used the following data augmentations:

- **Random Resized Crop**: Randomly crops the image and resizes it to 128x128 pixels.
- **Random Horizontal Flip**: Flips the image horizontally with a probability of 0.5.
- **Color Jitter**: Applies random changes to brightness, contrast, saturation, and hue.
- **Random Grayscale**: Converts the image to grayscale with a probability of 0.2.
- **Normalization**: Normalizes the pixel values based on the mean and standard deviation of the ImageNet dataset.

3. Learning Rate Schedule

- I used a **CosineAnnealingLR** scheduler to adjust the learning rate over the training epochs. The technique gradually decreases the learning rate following a cosine curve, ensuring a smooth annealing process as the training proceeds. The minimum learning rate was set to 0.001, while the initial learning rate was relatively low at **1e-5** to prevent large updates during self-supervised learning.

4. Optimizer

- The optimizer used for pre-training is **AdamW** (Adam with Weight Decay), which improves regularization by decoupling weight decay from the learning rate schedule. This optimizer is well-suited for self-supervised learning, especially when working with large models like ResNet50.
- The weight decay was set at **0.01**, which helps to prevent overfitting by penalizing large weights in the network.

5. Batch Size

- I used a **batch size of 64** during the pre-training phase. This is a choice that balances the availability of my computational resources and training efficiency.

6. Epochs and Training Setup

- The pre-training phase was run for **50 epochs** with a total learning process tracked carefully to save the best-performing checkpoint based on the loss.

This combination of SSL techniques, data augmentation, and optimizations ensured that the ResNet50 backbone learned meaningful visual representations even without access to labeled data.

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 (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation accuracy (Office-Home dataset)
A	-	Train full model (backbone + classifier)	0.2562
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	0.4655
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)	0.4828
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	0.2931
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	0.1256

Based on the results provided in the table, I rank the image classification performance on the Office-Home dataset from high to low, along with an analysis:

Ranked Results (from highest to lowest accuracy)

1. **Setting C:**
   - **Pre-training:** w/o label (Your SSL pre-trained backbone on Mini-ImageNet)
   - **Fine-tuning:** Train full model (backbone + classifier)
   - **Validation Accuracy: 0.4828**
   - **Analysis:** The result suggests that my aforementioned SSL pre-training was highly effective at learning useful representations from unlabeled data, which transferred well to the Office-Home dataset during fine-tuning.

2. **Setting B:**
   - **Pre-training:** w/ label (TA-provided backbone pre-trained on Mini-ImageNet)
   - **Fine-tuning:** Train full model (backbone + classifier)
   - **Validation Accuracy: 0.4655**

- **Analysis:** This setting uses a backbone provided by the TAs, and the entire model is fine-tuned on the Office-Home dataset. The performance is slightly lower than setting C, which again highlights the effectiveness of SSL in pre-training.

3. **Setting D:**

- **Pre-training:** w/ label (TA-provided backbone pre-trained on Mini-ImageNet)
- **Fine-tuning:** Fix the backbone. Train classifier only
- **Validation Accuracy: 0.2931**
- **Analysis:** Here, the backbone is frozen, and only the classifier is trained on the Office-Home dataset. The lower accuracy compared to setting B shows that not fine-tuning the backbone limits the model's ability to adapt to the new task. The fixed backbone pre-trained with labels still provides decent performance, but it lacks flexibility for the downstream task.

4. **Setting A:**

- **Pre-training:** No pre-training
- **Fine-tuning:** Train full model (backbone + classifier)
- **Validation Accuracy: 0.2562**
- **Analysis:** This setting does not use any pre-training and starts training from scratch on the Office-Home dataset. The significantly lower accuracy compared to settings B and C shows the importance of pre-training in learning generalizable features. Without pre-training, the model struggles to learn efficiently from the relatively small Office-Home dataset.

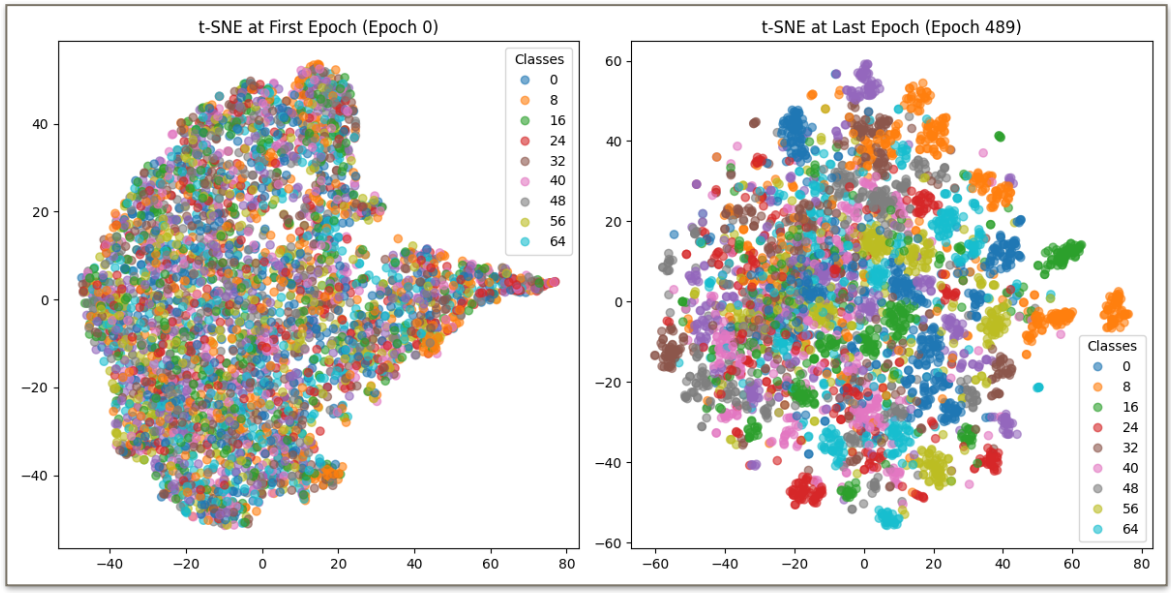
5. **Setting E:**

- **Pre-training:** w/o label (Your SSL pre-trained backbone on Mini-ImageNet)
- **Fine-tuning:** Fix the backbone. Train classifier only
- **Validation Accuracy: 0.1256**
- **Analysis:** This setting also uses the SSL pre-trained backbone but does not allow the backbone to be fine-tuned. The poor performance indicates that while SSL pre-training provided useful initial representations, freezing the backbone prevents the model from adapting to the new domain, especially when only the classifier is being trained.

Overall Discussion:

The results show the importance of both pre-training and fine-tuning. SSL pre-training (setting C) outperformed supervised pre-training (setting B), demonstrating that SSL can learn highly transferable features even without labeled data. Fine-tuning the entire model (backbone + classifier) led to significantly better results than training the classifier only (settings D and E), indicating that adapting the entire network to the new task is crucial for success. Additionally, starting from scratch without any pre-training (setting A) resulted in poor performance, highlighting the necessity of leveraging external datasets and pre-training techniques when fine-tuning on smaller, domain-specific datasets.

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 t-SNE visualizations for the first and last epochs in **Setting C** show the progression of the learned representations over the course of training. Here is an explanation:

First Epoch (Epoch 0) – Left Plot:

- **Observation:** In the t-SNE plot at the first epoch, the representations appear largely scattered with considerable overlap among different classes. The lack of clear clustering indicates that the model has not yet learned meaningful features to distinguish between the classes. This is expected at the start of training, as the model has not yet to fine-tune its parameters on the Office-Home dataset.

  To explain the result, at the initial stage of training, the model has random weights, which causes the representations of different classes to be mixed. t-SNE shows that there is no apparent structure, and class separation is not observable.

Last Epoch (Epoch 489) – Right Plot:

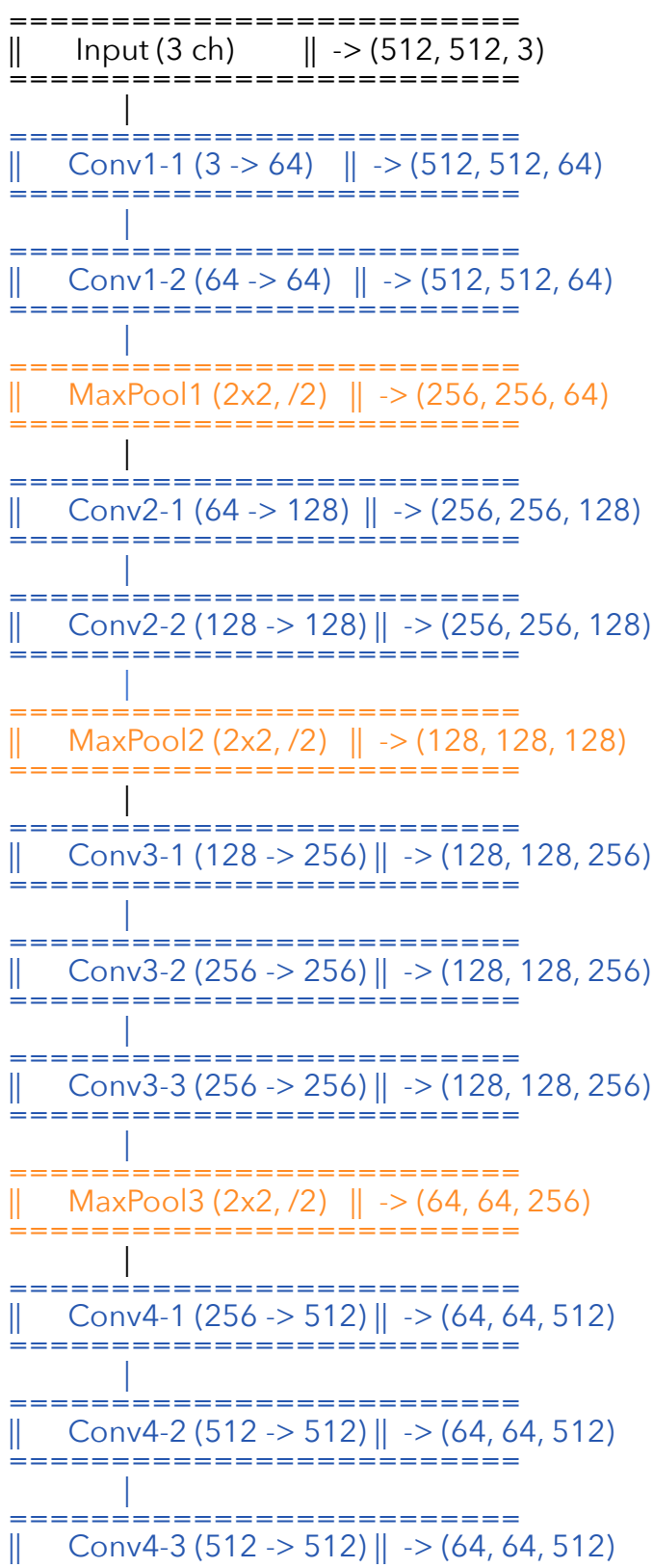
- **Observation:** By the last epoch, the t-SNE plot reveals more distinct clusters. Although some overlap remains, there is a clearer grouping of data points for certain classes, which reflects that the model has learned representations that better capture the differences between classes. Classes like 8 and 40 seem to form tighter, more distinct clusters, indicating the model's ability to differentiate between them more effectively.
  After almost 500 epochs of training, the model has fine-tuned its parameters, and as a result, the representations of different classes become more distinguishable. However, the presence of some overlapping clusters suggests that the learned features are still not fully separable for all classes, possibly due to the inherent complexity of the Office-Home dataset or insufficient training on certain classes.

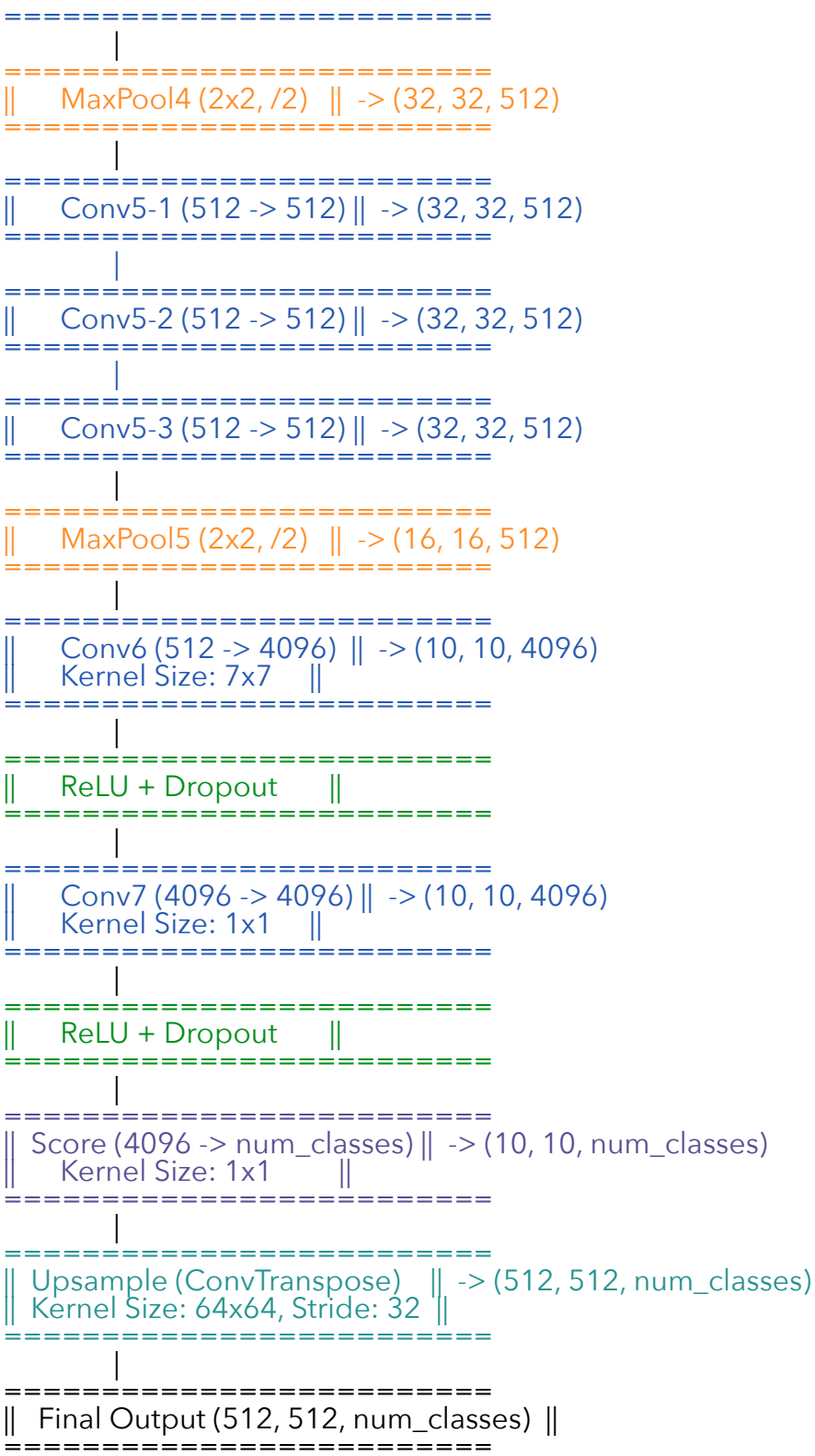
General Conclusion:

- The progression from a disorganized, overlapping space to a more structured and separated representation space reflects the effectiveness of training in Setting C. The SSL pre-trained backbone (without labeled data) has transferred well to the Office-Home dataset, and the fine-tuning process has allowed the model to refine its representations. However, some overlap indicates that the representations could still be further improved, either by extending training or refining my model architecture.

Problem 2: Semantic segmentation

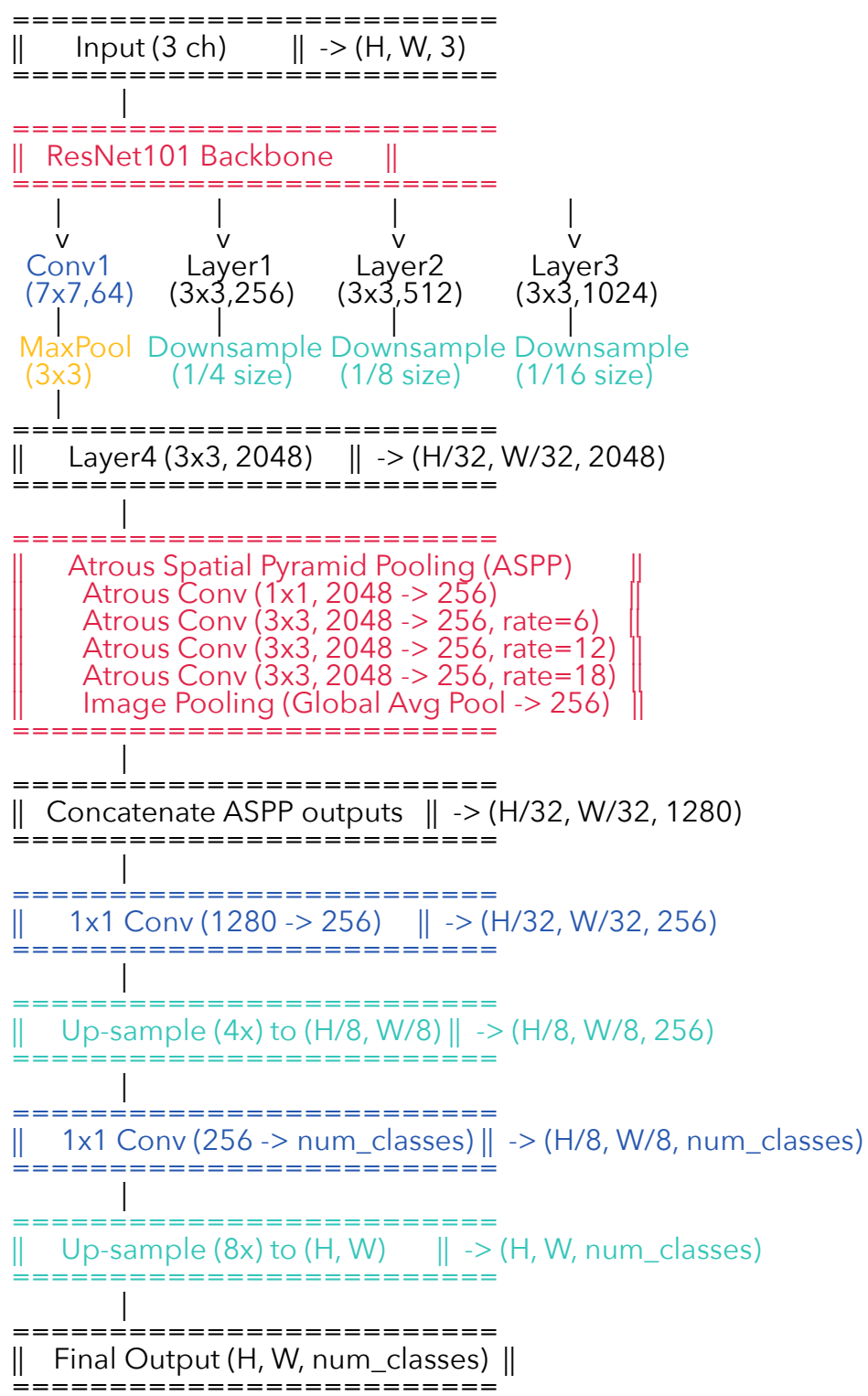
1. **(3%)** Draw the network architecture of your VGG16-FCN32s model (model A).




MaxPool4 (2x2, /2) -> (32, 32, 512)
Conv5-1 (512 -> 512) -> (32, 32, 512)
Conv5-2 (512 -> 512) -> (32, 32, 512)
Conv5-3 (512 -> 512) -> (32, 32, 512)
MaxPool5 (2x2, /2) -> (16, 16, 512)
Conv6 (512 -> 4096) -> (10, 10, 4096)
Kernel Size: 7x7
ReLU + Dropout
Conv7 (4096 -> 4096) -> (10, 10, 4096)
Kernel Size: 1x1
ReLU + Dropout
Score (4096 -> num_classes) -> (10, 10, num_classes)
Kernel Size: 1x1
Upsample (ConvTranspose) -> (512, 512, num_classes)
Kernel Size: 64x64, Stride: 32
Final Output (512, 512, num_classes)

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

H: 512, W: 512 as the input size of the images after resize.



Feature	VGG16-FCN32s	DeepLabV3 with ResNet101
Backbone	VGG16 (13 layers)	ResNet101 (101 layers with residuals)
Contextual Information	Less (no ASPP, single-scale)	High (ASPP, multi-scale context)
Learning Capacity	Simpler, fewer parameters	Complex, deeper model
Upsampling Strategy	Single 32x upsampling	Two-stage upsampling (4x, then 8x)

DeepLabV3 with ResNet101 offers more advanced segmentation capabilities compared to VGG16-FCN32s, particularly due to its multi-scale context understanding through ASPP, deeper feature extraction with ResNet101, and a more refined upsampling strategy. This is also demonstrated in the mIOU calculated on test set. It's clear that the mIOU predicted from DeepLabV3 with ResNet101 outperforms that predicted by VGG16-FCN32s model significantly.

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

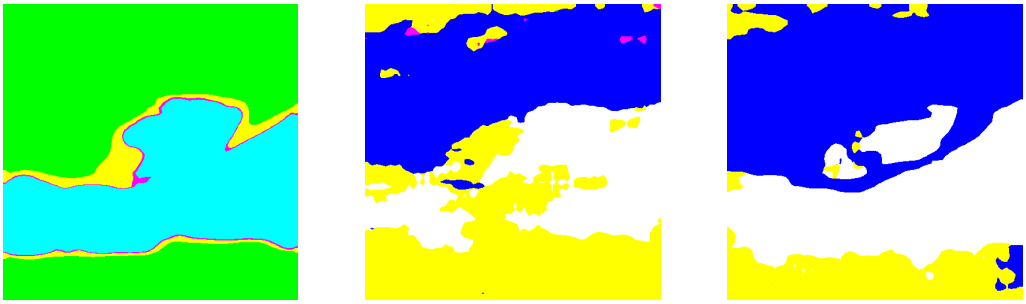
Model A: 0.4153

Model B: 0.7472

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.

Validation/0013_dat.jpg:

early: middle: final:

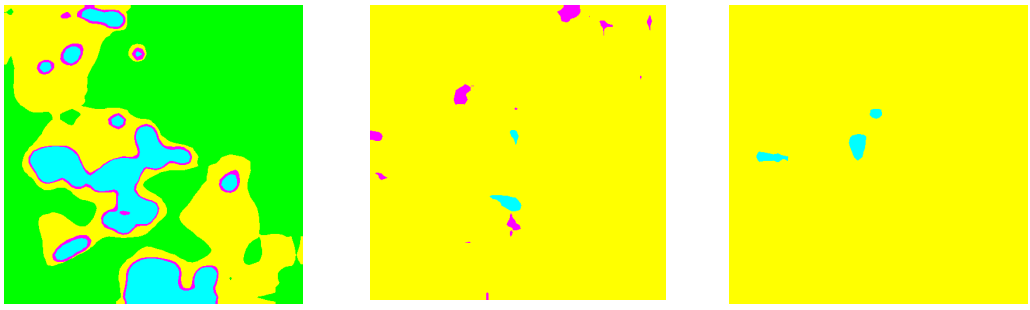


Validation/0062_dat.jpg:

early: middle: final:

Validation/0104_dat.jpg:

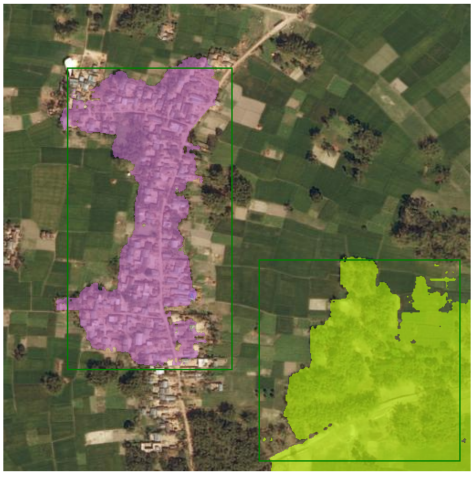
early: middle: final:



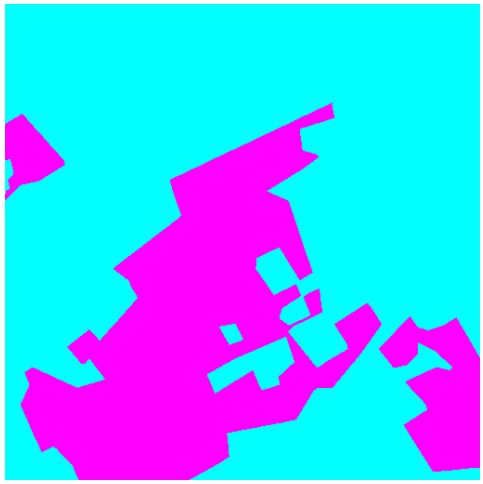
5. (10%) Use segment anything model (SAM) to segment three of the images in the validation dataset, report the result images and the method you use.

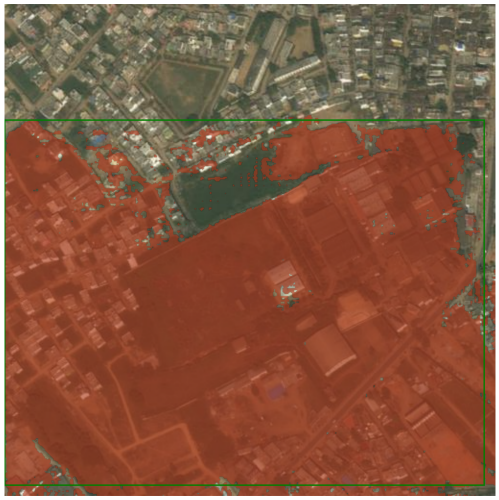
Validation: image 0028



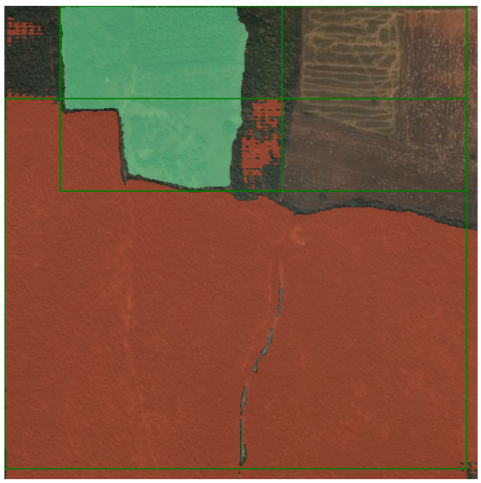



Validation: image 0112



Validation: image 0252



Methods I utilize:

For each image, four items are illustrated follow the arrangement as below:

Ground Truth Satellite Image	Ground Truth Mask Image
SAM with box prompts	SAM with import SamAutomaticMaskGenerator

Follow the guidance from (https://colab.research.google.com/github/facebookresearch/segment-anything/blob/main/notebooks/predictor_example.ipynb#scrollTo=e9c227a6) , I implement SAM with prompts for boxes in xyxy format. As shown above, the method can somehow segment given picture followed the prompt with reasonable performance. However, to attain superb accuracy, more prompt engineering might be needed. Here, I use 1-4 box prompts for segmentation.

As for automatic segmentation using SAM, I import package SamAutomaticMaskGenerator from augment-anything to segment images and combine masks with ground truth satellite images with transparency set as 0.5. While it can deal with fine-grained details, its result is largely different from what we expected, which is as expected.

All of the images above utilize pre-trained model of sam_vit_b_01ec64.pth. I've also test the model of vit_h, which is visually similar to what vit_b performs in automatic SAM. Slightly improvement have been seen in the test with prompts, I believe the model capacity do help in such task given the resolution.

References

1. DeepLabV3: https://paddlepedia.readthedocs.io/en/latest/tutorials/computer_vision/semantic_segmentation/DeeplabV3/DeeplabV3.html#id7
2. SAM with Prompt: https://github.com/facebookresearch/segment-anything/blob/main/notebooks/predictor_example.ipynb
3. Loss Function: https://paperswithcode.com/method/multi-loss-bce-loss-focal-loss-dice-loss
4. t-SNE Graph: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
5. BYOL: https://github.com/lucidrains/byol-pytorch
6. How to choose between Adam, AdamW, SGD: https://www.opt-ml.org/papers/2021/paper53.pdf
7. CosineAnnealingLR Scheduler: https://www.opt-ml.org/papers/2021/paper53.pdf
8. Semantic Segmentation on Satellite Images: https://github.com/JenAlchimowicz/Semantic-segmentation-with-PyTorch-Satellite-Imagery
9. ChatGPT
10. Claude
11. Tabnine AI