DLCV_HW1_report

B10901098 電機四 蔡承恩

P1_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)

- **SSL Method**

  The SSL method used for pre-training is **BYOL** (Bootstrap Your Own Latent). BYOL is a self-supervised learning method that doesn't rely on negative pairs or contrastive learning.

- **Data Augmentation Techniques**

  The data augmentation pipeline includes several techniques:

  1. Resizing to 144×144 pixels
  2. Random cropping to 128×128 pixels with padding of 4
  3. Random horizontal flipping
  4. Random rotation up to 15 degrees
  5. Color jittering (brightness, contrast, saturation, and hue)
  6. Random grayscale conversion (20% probability)
  7. Random Gaussian blur (30% probability)
  8. Random erasing (20% probability)

9. Normalization using ImageNet mean and standard deviation

- Learning Rate Schedule

  The learning rate is managed using the **OneCycleLR** scheduler. This scheduler:

  - Starts with a low learning rate
  - Increases to a maximum of 1e-3
  - Then decreases back down
  - Uses 10% of the total steps for the warm-up phase

- Optimizer

  The optimizer used is **AdamW** with the following parameters:

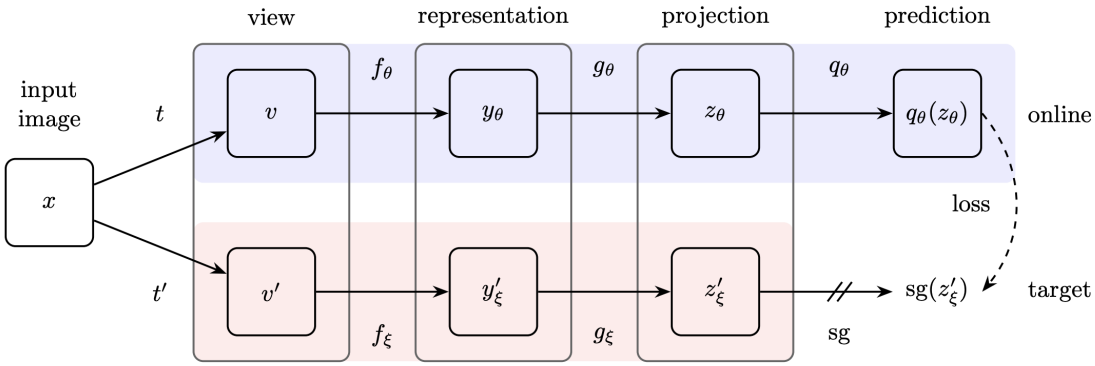
  - Initial learning rate: 3e-4
  - Weight decay: 0.1

- Batch Size

  The batch size is set to **128** for the pre-training phase.

- Additional Implementation Details

  1. **Model Architecture**: ResNet50 is used as the backbone network.
  2. **BYOL Configuration**:
     - Image size: 128×128
     - Hidden layer: 'avgpool'
     - Projection size: 256
     - Projection hidden size: 2048
     - Moving average decay: 0.996
  3. **BYOL Architecture**



4. **Training Duration**: The model is trained for 200 epochs.
5. **Mixed Precision Training**: The script uses automatic mixed precision training with a GradScaler for improved performance and memory efficiency.
6. **Gradient Clipping**: Gradients are clipped to a maximum norm of 1.0 to prevent exploding gradients.
7. **Moving Average Update**: After each optimization step, the BYOL learner updates its moving average.

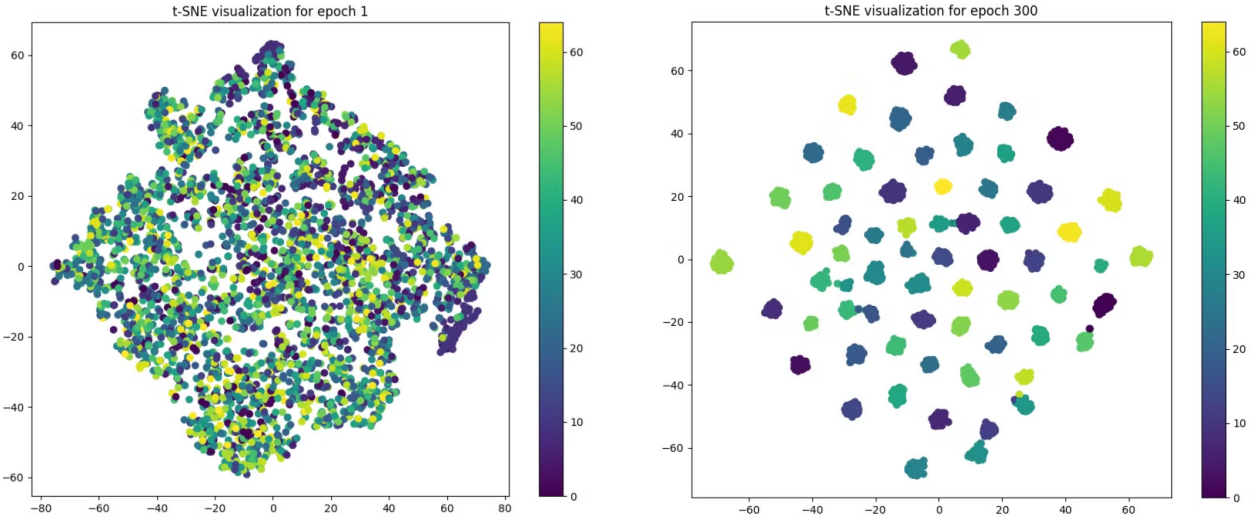
2. (20%) Please conduct the Image classification on Office-Home dataset as the downstream task. Also, please complete the following Table, which contains different image classification setting, and discuss/analyze the results.

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation accuracy (Office-Home dataset)
A	-	Train full model (backbone + classifier)	53.45%
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	58.87%
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone +	57.39%

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation accuracy (Office-Home dataset)
		classifier)	
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	40.39%
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	27.09%

- Analysis:
    1. Pre-training on a related dataset (Mini-ImageNet) with labeled data provides a good starting point for fine-tuning on the target dataset (Office-Home), as evident from the improved accuracy in setting B compared to setting A.
    2. Self-supervised learning (SSL) pre-training without labels (setting C) can still learn useful representations, resulting in better performance than training from scratch (setting A). However, it may not reach the same level of performance as pre-training with labeled data (setting B).
    3. Fine-tuning the entire model (backbone + classifier) is essential for adapting the pre-trained representations to the target dataset. Keeping the backbone fixed (settings D and E) leads to a significant drop in accuracy compared to fine-tuning the full model (settings B and C).
    4. The choice of pre-training approach (with or without labels) and the fine-tuning strategy (full model or classifier only) has a substantial impact on the final performance. The best results are obtained by pre-training with labeled data and fine-tuning the entire model (setting B).
- In conclusion, pre-training on a related dataset with labeled data, followed by fine-tuning the entire model on the target dataset, yields the best performance. SSL pre-training without labels can still provide benefits over training from scratch, but may not reach the same level of performance as pre-training with labels. The results highlight the importance of fine-tuning the entire model to effectively adapt the pre-trained representations to the target dataset.

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



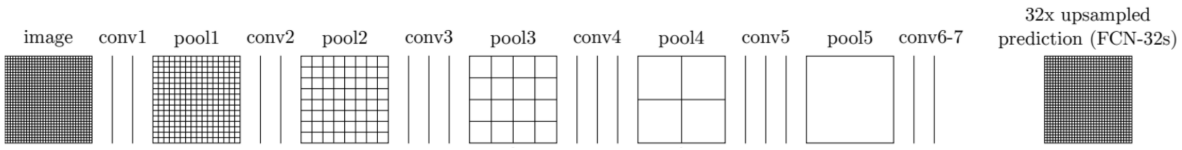
- Epoch 1 Visualization:
  - The data points are widely spread across the 2D space.
  - There's a lot of mixing between different colors (which likely represent different classes).
  - The structure is quite diffuse, without clear clusters or separations.
  - This suggests that at the beginning of training, the model hasn't yet learned to distinguish between different classes effectively.
- Epoch 300 Visualization:
  - The data points have formed distinct clusters.
  - There are clear separations between different colored groups.
  - The overall structure is more organized, with less overlap between different colors.

    - This indicates that by the end of training, the model has learned to separate different classes much more effectively.
- Key observations:
    1. Improved clustering: The transition from a diffuse spread to distinct clusters shows that the model has learned to group similar samples together and separate dissimilar ones.
    2. Better class separation: The clearer color distinctions in epoch 300 suggest that the model has learned features that are more discriminative between classes.
- In conclusion, these visualizations effectively demonstrate the power of the learning process in setting C. The model has clearly improved its ability to distinguish between different classes and learn relevant features for the task at hand, starting from an initially undifferentiated representation and evolving to a highly structured one.

P2_Semantic segmentation

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

- FCN32s Architecture



- My VGG16-FCN32s model

```
Input Image
   |
VGG16 Features (pre-trained)
   |
FC6 (Conv2d: 512 → 2048, kernel=7×7, padding='same')
   |
ReLU
   |
Dropout2d
   |
FC7 (Conv2d: 2048 → 2048, kernel=1×1, padding='same')
   |
ReLU
   |
Dropout2d
   |
Score_fr (Conv2d: 2048 → num_classes, kernel=1×1)
   |
Upscore (ConvTranspose2d: num_classes → num_classes, kernel=64×64,
stride=32)
   |
Crop to input size
   |
Output (num_classes channels)
```

Key points of this architecture:

1. It starts with the feature extraction layers of a pre-trained VGG16 network.
2. The fully connected layers of VGG16 are replaced with fully convolutional layers (fc6 and fc7).
3. A final scoring convolutional layer (score_fr) reduces the channel dimension to the number of classes.

4. The upscore layer uses transposed convolution to upsample the feature maps to the original input size.
5. The output is cropped to ensure it matches the input image dimensions exactly.

This architecture follows the FCN32s (Fully Convolutional Network) approach, which performs a single step upsampling from the last layer to the input size.

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

- My improved model with DeepLabV2 Architecture

```
Input Image
   |
   v
ResNet Backbone (with atrous convolutions)
   |
   |---------> Auxiliary Classifier (FCNHead)
   |
   v
Atrous Spatial Pyramid Pooling (ASPP)
   |
   v
DeepLabHead (Classifier)
   |
   v
Output (Segmentation Map)
```

Key components of DeepLabv3:

1. ResNet Backbone: Extracts features using atrous convolutions in later layers to maintain spatial resolution.

2. ASPP: Applies multiple atrous convolutions with different rates to capture multi-scale context.
3. DeepLabHead: A series of convolutions that process the ASPP output to produce the final segmentation map.
4. Auxiliary Classifier: An additional classifier connected to an intermediate layer of the backbone for improved training.

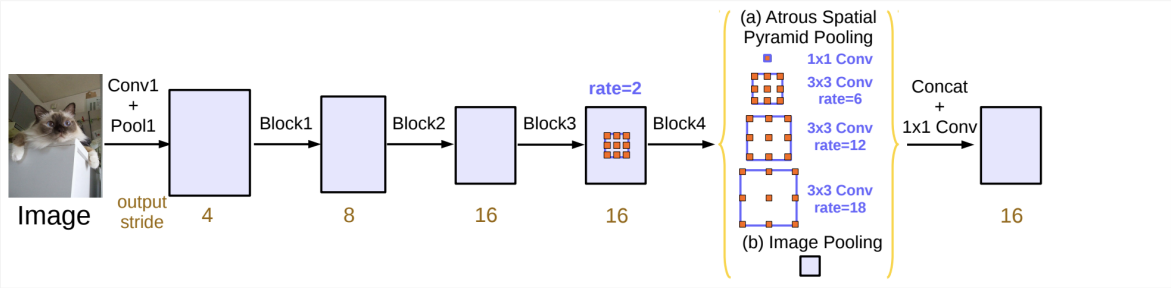


Figure 5. Parallel modules with atrous convolution (ASPP), augmented with image-level features.

ref: https://arxiv.org/pdf/1706.05587

- **Key differences and an overview of the DeepLabv3 architecture:**
  1. Backbone Network:
     - VGG16-FCN32s: Uses VGG16 as the backbone.
     - DeepLabv3: Uses ResNet50 (or optionally ResNet101) as the backbone.
  2. Atrous Convolutions:
     - VGG16-FCN32s: Does not use atrous convolutions.
     - DeepLabv3: Employs atrous (dilated) convolutions in the backbone to increase the receptive field without losing spatial resolution.
  3. Atrous Spatial Pyramid Pooling (ASPP):
     - VGG16-FCN32s: Does not have an ASPP module.
     - DeepLabv3: Incorporates an ASPP module to capture multi-scale context.
  4. Decoder:

    - VGG16-FCN32s: Uses a simple transposed convolution for upsampling.
    - DeepLabv3: Has a more sophisticated decoder with the DeepLabHead.
5. Auxiliary Loss:
    - VGG16-FCN32s: Does not use an auxiliary loss.
    - DeepLabv3: Includes an auxiliary classifier (FCNHead) for improved training.

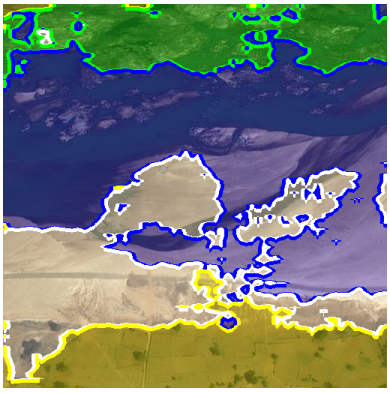
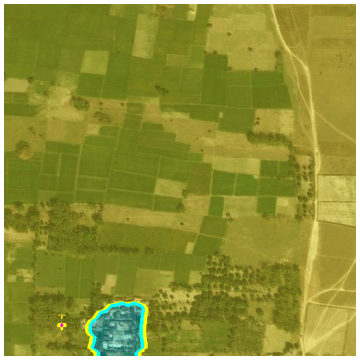
- The DeepLabv3 architecture is more advanced and typically achieves better performance in semantic segmentation tasks compared to the VGG16-FCN32s model. It addresses the limitations of FCN32s by using atrous convolutions to maintain spatial resolution and incorporating multi-scale context through the ASPP module.

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

- Model A (VGG16-FCN32s) :mIoU = 0.7133
- Model B (Improved) : mIoU = 0.7436

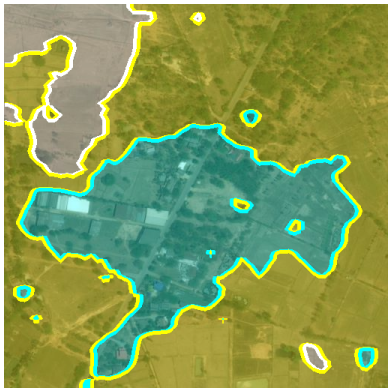
4. (3%) Show the predicted segmentation mask of “validation/0013_sat.jpg”, “validation/0062_sat.jpg”, “validation/0104_sat.jpg” during the early, middle, and the final stage during the training process of the improved model.

Early Stage (1st)

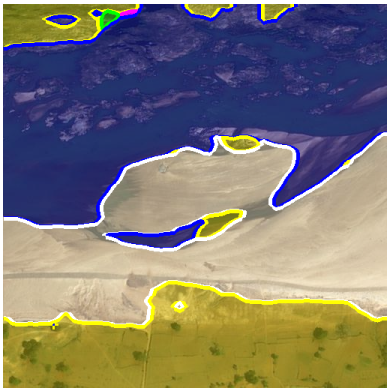




Middle Stage (n/2 th)



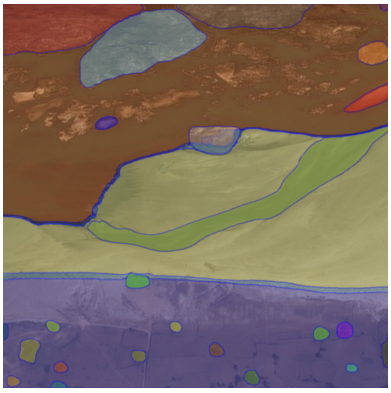
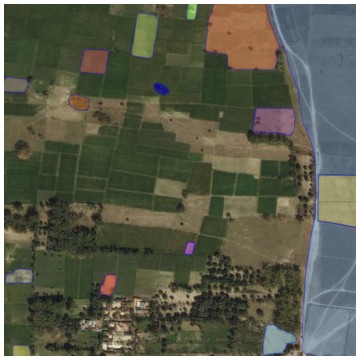
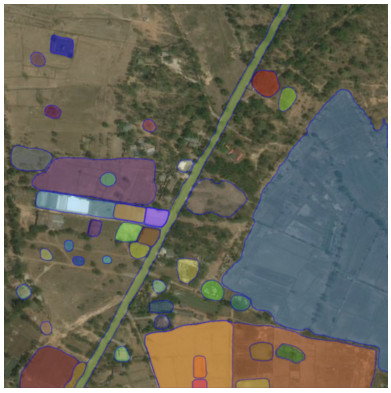


Final Stage (n th)



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.

- 0013_sat.jpg / 0062_sat.jpg / 0104_sat.jpg

- I use the Segment Anything Model (SAM) demo on segment-anything.com, which showcases the core capabilities of SAM using a user-friendly interface:
    1. Image Encoding:
       When we upload an image, SAM's image encoder (a Vision Transformer) processes the entire image once, creating a dense visual representation.
    2. Mask Generation:
       SAM's mask decoder uses the encoded image and prompts to generate segmentation masks in real-time. It can produce multiple mask predictions for ambiguous cases.
    3. Ambiguity Handling:
       SAM can generate multiple valid segmentations for a single prompt, reflecting different possible interpretations.
    4. Zero-shot Capability:

       Training on a diverse dataset (SA-1B), SAM can generalize to new objects and scenes it hasn't explicitly been trained on.SAM can segment objects it wasn't specifically trained on, demonstrating its generalization ability.

5. Automatic Segmentation:
   The "Segment Everything" option uses SAM to automatically detect and segment all objects in the image without specific prompts.