DLCV Hw-1

Name: 李維釗

Student ID: R12942089

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

1. Describe the implementation details of your SSL method for pre-training the ResNet50 backbone.
   - Pre-train
     - Utilizing `BYOL` to pre-train the ResNet50 backbone model.
     - In the pre-trained phase, I utilize `AdamW` optimizer with the `lr = 1e-4`, `weight_decay = 0.05`, betas = (0.9, 0.999); and utilize a `linear warm-up for 4000 iterations` and then use `CosineAnnealingLR`. And the `batch size = 256`, `training epoch = 1000`.
   - Finetune
     - In the finetune phase, I utilize `AdamW` optimizer with the `lr = 1e-4`, `weight_decay = 0.05`, betas = (0.9, 0.999); and utilize a `linear warm-up for 100 iterations` and then use `CosineAnnealingLR`. And the `batch size = 128`, `training epoch = 500`.
2. 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.

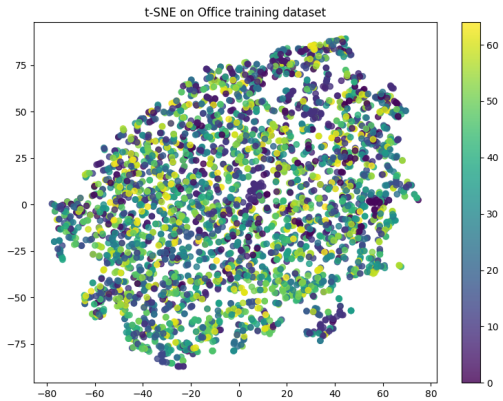
Setting	**Pre-training (Mini-ImageNet)**	**Fine-tuning (Office-Home dataset)**	**Validation Accuracy (Office-Home dataset)**
A	-	Train full model (backbone +	0.4877

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation Accuracy (Office-Home dataset)
		classifier)	
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	0.5714
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)	0.5960
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	0.2734
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	0.3719

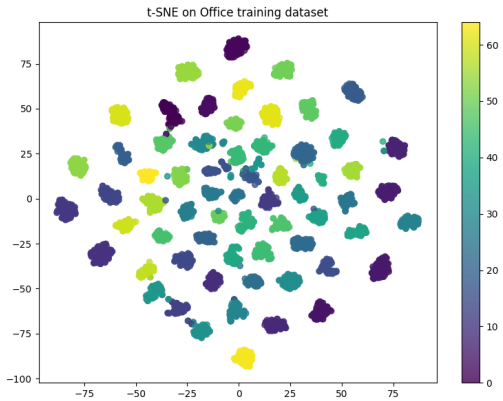
- The results accuracy can observed that whether the backbone parameter is trainable has a higher relevance with the prediction accuracy.

3. 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.

- First



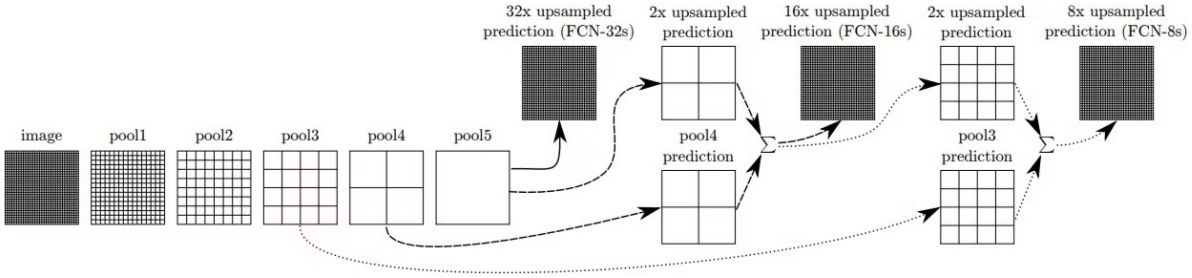
- Last



- Comparison
  - At a glance of the t-SNE result from the first epoch checkpoint, we can see that the compressed embeddings of training data are in a mess. We cannot see any cluster in the 2D dimensions embedding.
  - As for the result from the last epoch checkpoint, we can easily see the clusters in the 2D dimensions embedding, which means the encoder learn well makes the embedding in the low dimension can be easily separated.

Problem 2: Semantic segmentation

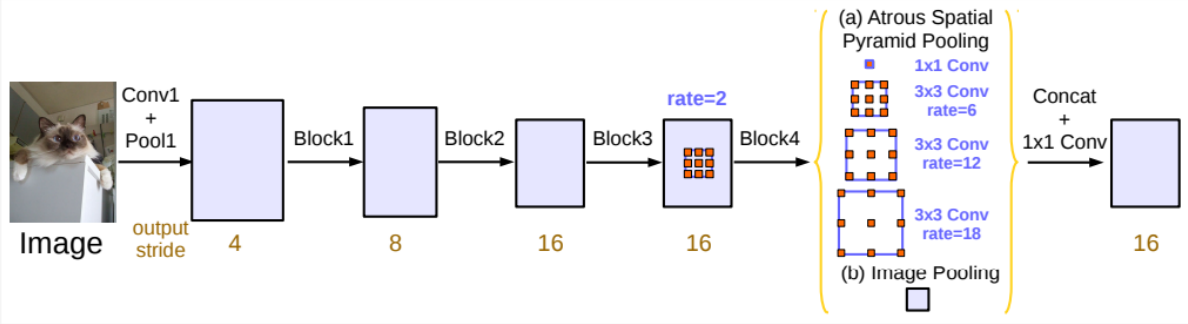
1. Draw the network architecture of your VGG16-FCN32s model (model A).
   - By utilize the last pooling result of VGG16, the FCN32s conduct a 32x upsampled to predict the class of the pixels of the image.




```
fcn32(
  (features): 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)
  )
```

```
  (conv1): Conv2d(512, 4096, kernel_size=(1, 1), stride=(1, 1))
  (relu1): ReLU(inplace=True)
  (drop1): Dropout2d(p=0.5, inplace=False)
  (conv2): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
  (relu2): ReLU(inplace=True)
  (drop2): Dropout2d(p=0.5, inplace=False)
  (upsample32x): Upsample(scale_factor=32.0, mode='bilinear')
)
```

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



- When I implement setting A, I found that the mIoU of class 2 is very low. I take a glance of the data and found the data imbalance problem. Thus, different from setting A, I conduct data augmentation and use a focal loss to cope with this problem.
  - Augmentation
    - I utilize `identity`, `horizontal flip` and `vertical flip` for every image, which make the dataset 3 times large.
  - Loss criterion (Focal loss)
    - Based on CE (cross-entropy) loss, add 2 concepts to improve.
      1. The cross-entropy loss evaluates the class prediction for each pixel separately and then averages the loss across all pixels, so we are essentially learning equally for every pixel in the image. Due to the data imbalance, we want to let the model take more attention on the imbalance class, so use a weighted factor ($\alpha \in [0, 1]$) to re-scale the CE loss.

$$CE(p_t) = -\alpha \log(p_t)$$

2. But in a dataset, there are some samples which are hard to clarify their class, called hard samples and the other are easy samples. To let the hard samples be more influence in the optimization process, introducing a modulating factor ($\gamma \geqq 0$). Combine the two concepts, the final focal loss can be reformulated by:

$$FL(p_t) = -\alpha(1-p_t)^r \log(p_t)$$

- In the setting B, I set the `α = 0.25`, `γ = 2`.

- Model
  - In the pytorch website, they provide 3 different types of model to use, and I follow the table below, select a model with highest performance (mIoU) called `deeplabv3_resnet101`. And adjust the last classifier to map the image feature to the correct class number.

    https://pytorch.org/vision/stable/models.html#semantic-segmentation

Weight	Mean IoU	pixelwise Acc	Params	GFLOPS	Recipe
DeepLabV3_MobileNet_V3_Large_Weights.COCO_WITH_VOC_LABELS_V1	60.3	91.2	11.0M	10.45	link
DeepLabV3_ResNet101_Weights.COCO_WITH_VOC_LABELS_V1	67.4	92.4	61.0M	258.74	link
DeepLabV3_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1	66.4	92.4	42.0M	178.72	link
FCN_ResNet101_Weights.COCO_WITH_VOC_LABELS_V1	63.7	91.9	54.3M	232.74	link
FCN_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1	60.5	91.4	35.3M	152.72	link
LRASPP_MobileNet_V3_Large_Weights.COCO_WITH_VOC_LABELS_V1	57.9	91.2	3.2M	2.09	link

2. Report mIoUs of two models on the validation set.

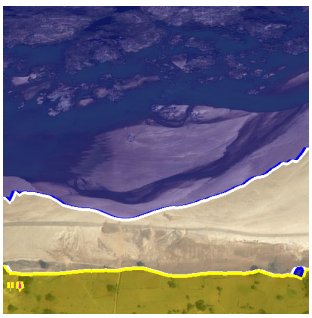
- In the 2 setting, I implement the same optimizer, scheduler setting. I utilize `AdamW` optimizer with the `lr = 1e-4`, `weight_decay = 0.05`, betas = (0.9, 0.999); and utilize a `linear warm-up for 500 iterations` and then use `CosineAnnealingLR`. And the `batch size is 16 and 8 for A and B respectfully`.

Setting	mIOUs
A	0.700817

Setting	mIOUs
B	0.759669

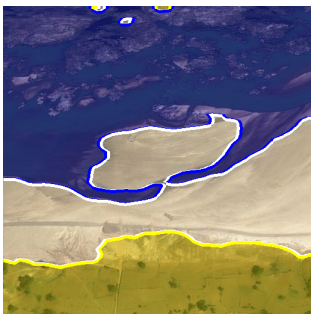
3. Show the predicted segmentation mask of “validation/0013_sat.jpg”, “validation/0062_sat.jpg”, “validation/0104_sat.jpg” during the early, middle, and the final stage during the training process of the improved model.

- 0013_sat.jpg (early)

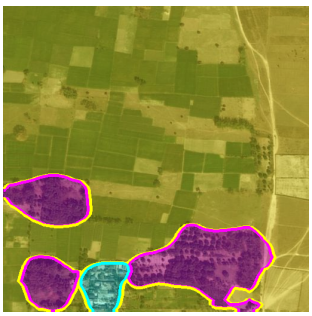


- 0013_sat.jpg (middle)

- 0013_sat.jpg (final)

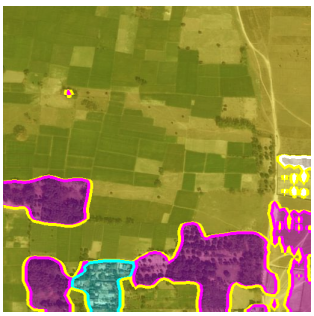


- 0062_sat.jpg (early)



- 0062_sat.jpg (middle)

- 0062_sat.jpg (final)

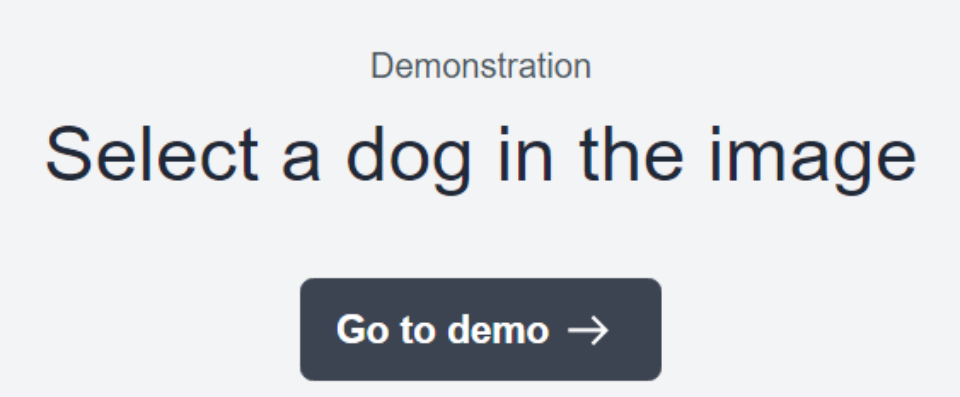


- 0104_sat.jpg (early)

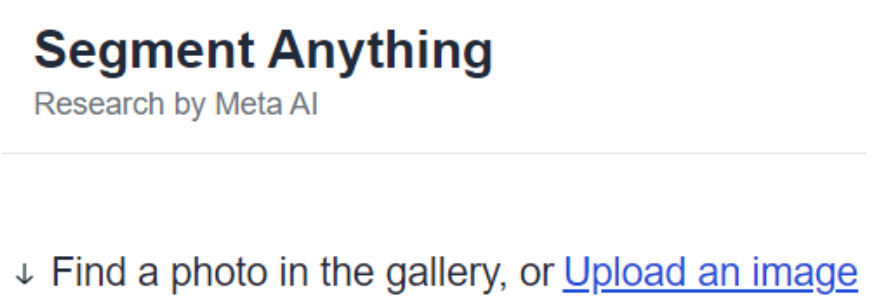
- 0104_sat.jpg (middle)

- 0104_sat.jpg (final)

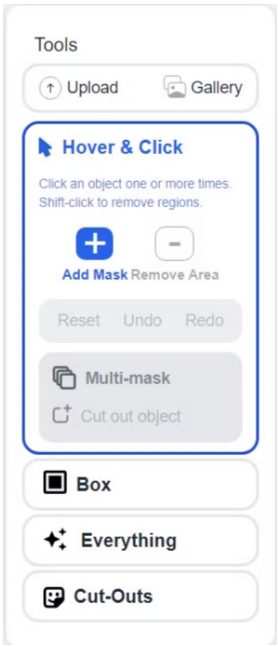
4. Use SAM to segment three of the images in the validation dataset, report the result images and the method you use.
    - Using meta released website to inference segmentation.
      URL:
      https://segment-anything.com/
        - Scrolling down find the `Go to demo ->` and click it



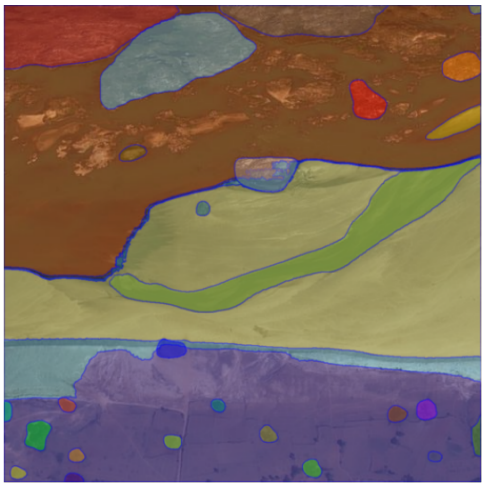
        - Click `Upload an image`



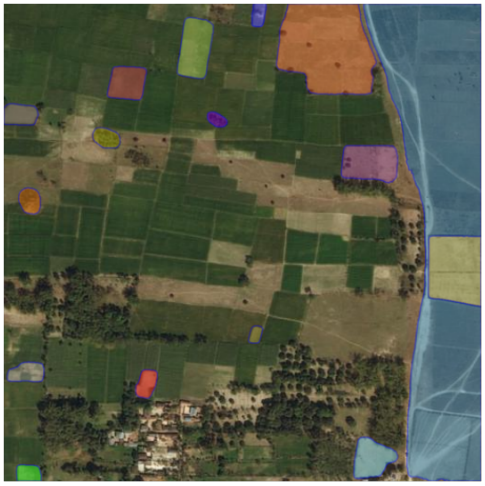
        - After uploading an image, click the `Everything` in the left side tool bar and get a segmented image.



- Results
  - 0013_sat.jpg



  - 0062_sat.jpg



- 0104_sat.jpg