hw1_r11944004

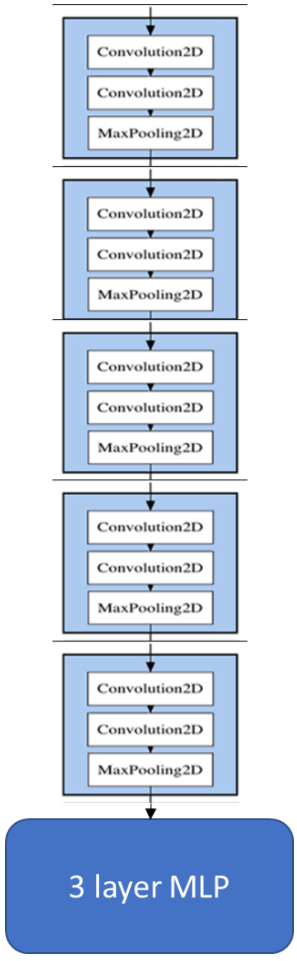
Student ID: r11944004

Department: GINM 11

Name: 李勝維

Problem 1: Image Classification

1. Draw the network architecture of method A or B.



I choose the famous VGG13 as my architecture for method A, which consist of 10 convolution layers as feature extractor and three fully-connected layers as classifier.

2. Report accuracy of your models (both A, B) on the validation set.

	Validation accuracy	Trained Epochs
Method A (VGG)	0.7454	300
Method B (BEiT)	0.9367	6

3. Report your implementation details of model A.

First, torchvision already implements VGG, thus, I’ve adapted the source code from vision/vgg.py at main · pytorch/vision (github.com) for more detailed control over the model (ex: shrink the model size).

Model Settings

- The kernel size is three, padding is one for all conv. layers.
- Batch normalization layer between each convolution layer and ReLU layer.
- Number of output channels for convolution layers: [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'], where M stands for max-pooling layer, FYR.

Training Settings

- Data augmentations: random crop, random horizontal flip
- Loss function: cross entropy loss
- Optimizer: AdamW optimizer with learning rate 0.001 (since I’m not using weight decay, adamw degrades to adam.)
- Epoch: trained for 300 epochs
- Cross validation method: I calculate the validation accuracy every epoch, and saves a checkpoint whenever the best record is updated. Although this may increase the chances of over-fitting, VGG13 is a small network and will likely to not over-fit.

4. Report your alternative model or method in B, and describe its difference from model A.

Brief introduction to BEiT

I choose to finetune BEiT as my method B.

[2106.08254] BEiT: BERT Pre-Training of Image Transformers (arxiv.org)

In short, BEiT is a vision transformer pre-trained on ImageNet21K in a self-supervised manner, and is capable of finetuned on several downstream vision tasks.

During finetuning, I initialize one layer of FC (without nonlinearity) as classification head after the embedding of [CLS] token. Then I finetune the whole model with our training data.

Difference from model A

Since BEiT is based on vision transformer, which is totally different from CNN in the following aspects

- Transformers is able to have better global understanding of images compared to CNN
- Transformers splits input image into visual tokens while CNN uses pixel arrays
- Empirically, Transformers needs more training data than CNN.

Model Settings

- Architecture: beit base (i.e., 12 layers of transformer encoder block)
- Pretrained weight: 'beit_base_patch16_224_in22k' from timm library, which in pre-trained on ImageNet 22K dataset without the labels.

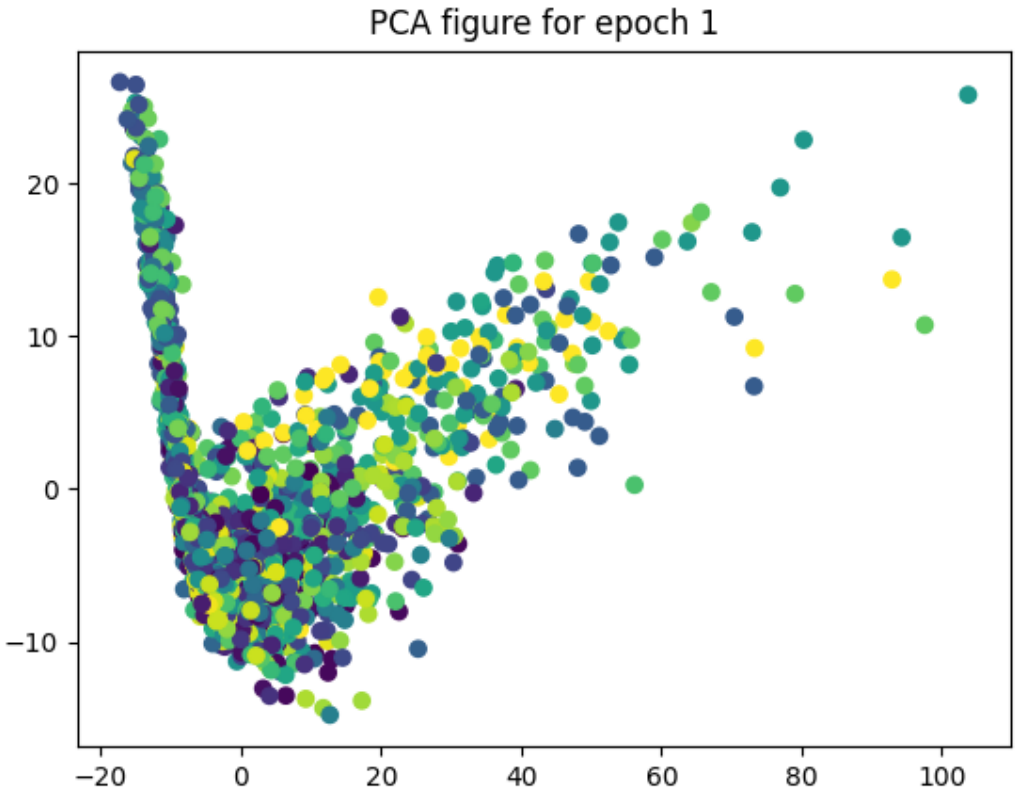
Training Settings

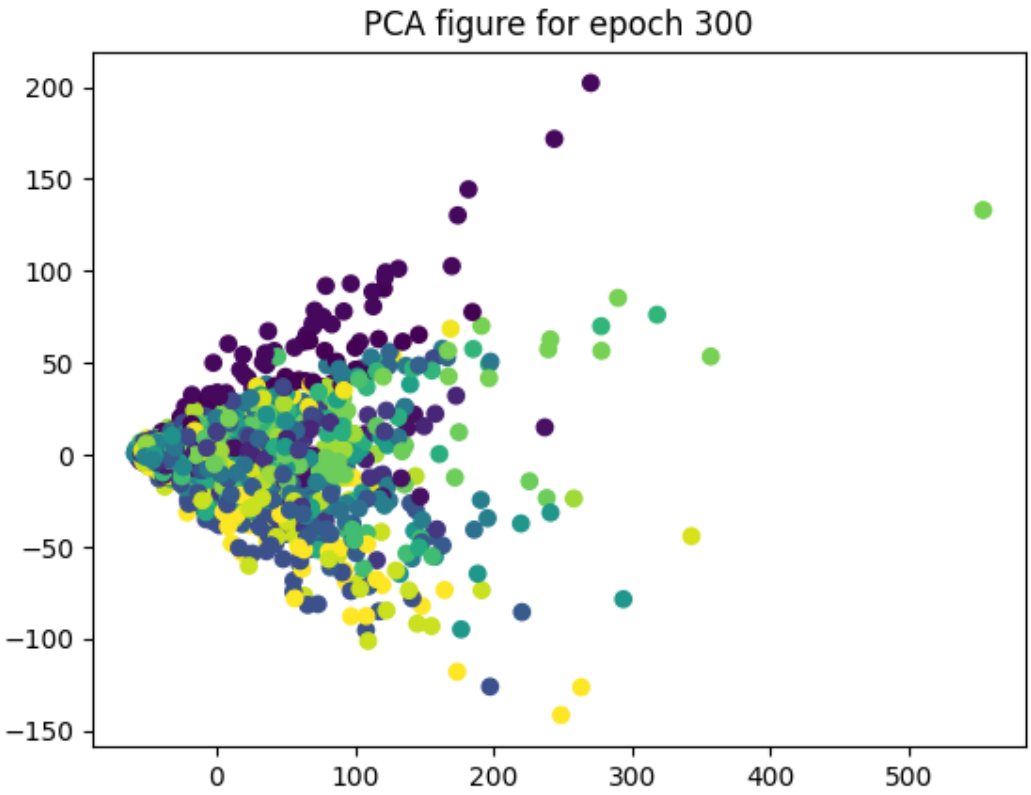
- Data augmentations: mixup and cutmix
  - mixup alpha = 0.1
  - randomly apply mixup and cutmix with equal probabilities
- Loss function: cross-entropy with label-smoothing factor = 0.1
- Optimizer: SGD with learning rate 0.003
- Epoch: trained for 6 epochs
- Cross validation is the same as method A

5. Visualize the learned visual representations of model A on the validation set by implementing PCA

(Principal Component Analysis) on the output of the second last layer. Briefly explain your result of the PCA visualization.

PCA visualization of the second last layer of model A:

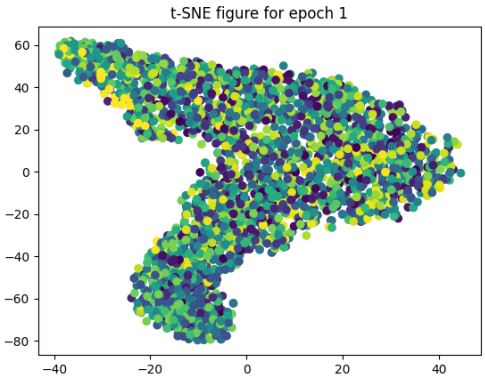


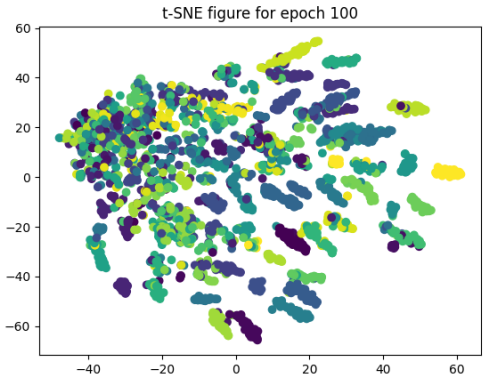


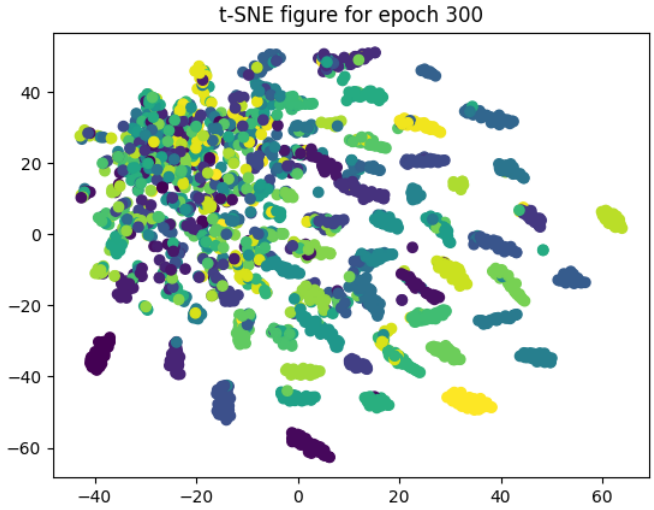
We can see that after learning, the model is able to seperate different classes (ex: we can clearly seperate the “purple” class from the others)

The reason why most classes seem to be overlapping, my guess is though the classes are seperatable in a high dimension space, linear projections cannot project them very well. We can confirm that by using t-SNE to visualize the data instead.

6. Visualize the learned visual representation of model A, again on the output of the second last layer, but using t-SNE (t-distributed Stochastic Neighbor Embedding) instead. Depict your visualization from three different epochs including the first one and the last one. Briefly explain the above results.



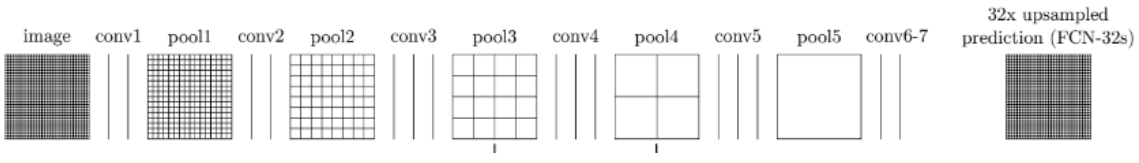




We can find that the classes seperations becomes more and more obvious after training. Compared to PCA, t-SNE can seperate classes more clearly.

Problem 2: Semantic Segmentation

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

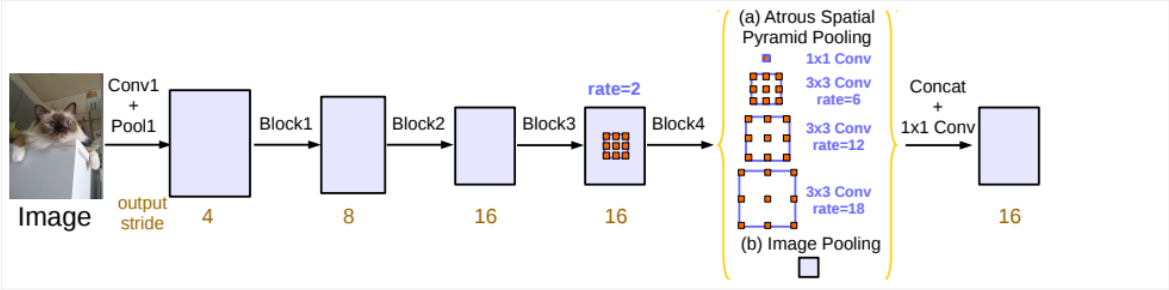


I replaced the last 2 layers (fc6, fc7) in VGG16 by convolution layers, then I append a transpose convolution layer at the end to upscale the 16x16 feature map to 512x512 (original input resolution).

Training Settings

- Data augmentations: None
- Loss function: cross-entropy
- Optimizer: SGD with learning rate 0.003
- Epoch: trained for 100 epochs
- Cross validation method: I calculate the validation accuracy every epoch, and saves a checkpoint whenever the best record is updated.

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



Brief introduction to Deeplab v3

I use Deeplab v3 introduced in Rethinking Atrous Convolution for Semantic Image Segmentation as by model B. The model is pre-trained on MSCOCO dataset.

First, we feed the input image into a CNN encoder to obtain a low-resolution feature map.

Second, we obtain multi-scale feature pyramid by utilizing three 3x3 atrous(dilated) convolution with rates = [6, 12, 18], a 1x1 convolution and global average pooling.

Finally, we concatenate all the feature maps obtained in the previous step, then pass the feature map through 1x1 convolutions to get the final score for each class on every pixel.

Difference from model A

Compared to FCN-32s, which only considers a single feature map, Deeplab v3 considers different scales of feature map to obtain the segmentation result. This technique not only captures more details from the image, but also produces a “higher resolution”(sharper) result.

Besides model architecture, I also use a different optimization objective to train my model B. Since the dataset is class imbalance, the model often cannot segment class #2 (Purple, Rangeland) correctly. Therefore, I decide to use focal loss introduced in Focal Loss for Dense Object Detection, which aims to solve the label imbalance problem in object detection (and the same technique can be applied on semantic segmentation too), to optimize my model.

Training Settings

- Data augmentations: randomly horizontal flip and vertical flip with rate = 0.5
- Loss function: focal loss with alpha = 0.2 and gamma = 2
- Optimizer: SGD with learning rate = 0.1 and weight decay = 1E-5. Plus one cycle learning rate scheduler introduced in Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates with pytorch default arguments
- Epoch: trained for 100 epochs
- Cross validation method: I calculate the validation accuracy every epoch, and saves a checkpoint whenever the best record is updated.

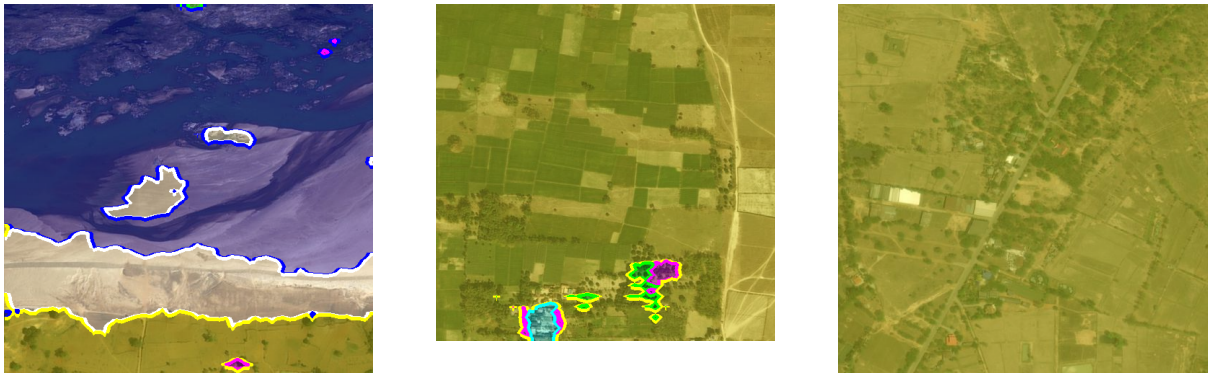
3. Report mIoUs of two models on the validation set.

	Validation mIoU	Trained Epoch
Model A	0.589259	100
Model B	0.743783	100

4. 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.

Images from left to right are “validation/0013_sat.jpg”, “validation/0062_sat.jpg”, and “validation/0104_sat.jpg” respectively.

First Epoch



N/2th Epoch

Last Epoch

Ground Truth



After training, the model is able to segment better compared to early epochs.

But the scarce classes (cyan and purple) still cannot be segmented very well.

References

- torchvision — Torchvision main documentation (pytorch.org)
- [2106.08254] BEiT: BERT Pre-Training of Image Transformers (arxiv.org)
- Fully Convolutional Networks for Semantic Segmentation (cv-foundation.org)
- [1706.05587] Rethinking Atrous Convolution for Semantic Image Segmentation (arxiv.org)
- [1708.02002] Focal Loss for Dense Object Detection (arxiv.org)
- [1708.07120] Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates (arxiv.org)