COSE474 Deep Learning
Project 3 Report

컴퓨터학과 2022320006 최진혁

1. Description of Code

i. Original UNet

The conv function is a set of two convolutions. According to the pdf file of Project 3, the output channel size of each encoder block is 64, 128, 256, 512, 1024, and the input channel is same as the output channel of previous block. Thus we can easily fill the blanks of convDown1~5 with (in_channels, 64), (64, 128), …, (512, 1024). The input channel of each decoder block is the sum of previous output channels and the output channels of encoder block of same level. The output channel is 512, 256, 128, 64. Thus we can fill the blanks of convUp4~1 with (1024 + 512, 512), (512 + 256, 256), …, (128 + 64, 64). This implies the input channels of convUp_fin is 64.

In the forward(), concatenation is performed with torch.cat() function. The input of convUP4~1 will be concatenated with convDown4~1, which are conv4~1. The concat. Will be performed on the channel dimension. Thus fill the blanks with torch.cat((x, conv4), dim=1), …, torch.cat((x, conv1), dim=1).

ii. UNet with ResNet Encoder

Question 1 is overlapped with Project 2, so I will skip this part. In the question 2, there is two concatenations. According to the pdf file, the output of UpConv1, 2 will be concatenated with out2, 1 respectively. Thus we can fill the blanks with torch.cat((x, out2), dim=1), torch.cat((x, out1), dim=1), respectively.

iii. Modules

In train_model(), we should set gradients to zero with optimizer.zero_grad(). Then forward and get the output of model, comput loss with output and labels, perform back propagation and update parameters with optimizer.step().

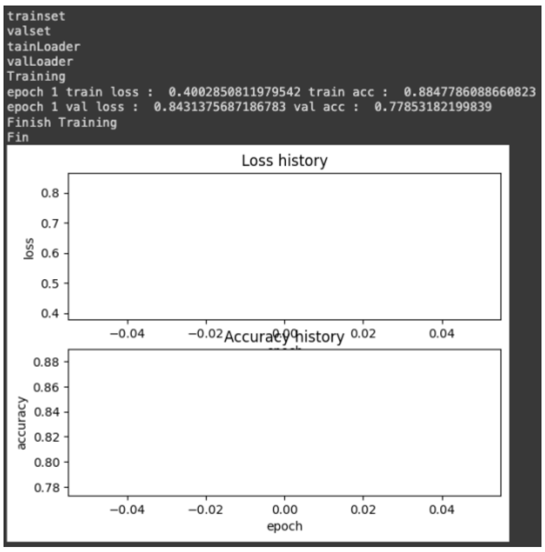
In get_loss_train(), we do not need to back propagation and updating parameters. Thus we just get the output of model, and compute the loss.

In val_model(), same as in get_loss_train(), we just get the output and then compute the loss. In this, because it is not training, we first need model.eval() which is not given, while it is given in get_loss_train(). In the converting result and label to RGB image, temp is the output of the model, and temp_l is the label. The comment in the code said that the image (label) should become temp_rgb, and result (model's prediction) should become temp_label. The cls_invert is the dictionary of RGB values corresponding the class number as a key. Thus, we can convert using temp_rgb[j][k] = cls_invert[temp_l[j][k]], and temp_label[j][k] = cls_invert[temp[j][k]].

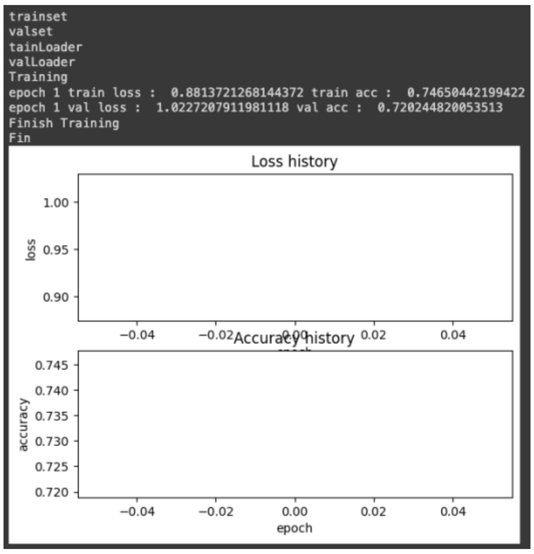
iv. Main

First, import UNet_skeleton and set the model. Define cross entropy loss and adam optimizer. Then load the model from checkpoint. There was an issue that 'map_location' is needed in the torch.load() when. Thus we add 'map_location=device'. Finally, save the checkpoint using torch.save().

2. Results



```
trainset
valset
tainLoader
valLoader
Training
epoch 1 train loss :  0.4002850811979542 train acc :  0.8847786088660823
epoch 1 val loss :  0.8431375687186783 val acc :  0.77853182199839
Finish Training
Fin
```



```
trainset
valset
tainLoader
valLoader
Training
epoch 1 train loss :  0.8813721268144372 train acc :  0.74650442199422
epoch 1 val loss :  1.0227207911981118 val acc :  0.720244820053513
Finish Training
Fin
```

The left figure is the result of original UNet. The training process was done on the Google Colaborary with GPU A100. Model was trained 1 epoch, because we use the pre-trained model checkpoint. Final train and validation accuracy was about 88%, 78%, respectively. Because we trained the model with 1 epoch, the graph was not drawn.

The right figure is the result of UNet with ResNet50 encoder. Training settings are equal to the previous one. Final train and validation accuracy was about 75%, 72%.

3. Discussions

In both cases, models are not performed well. We can consider train the model with more epochs. Especially, in the original UNet, the validation accuracy is relatively lower than the training accuracy. This implies the generalization ability of the model is not good. The possible reason would be an overfitting. To deal with this, we can decrease the complexity of the model, use more training data, or add regularization term. Otherwise, if it is not an overfitting, we can just train the model with more epochs.

The training process of UNet with ResNet50 encoder took a time longer than original UNet. The UNet with ResNet50 encoder took about 100 minutes, while the original UNet took about 30 minutes. This is because of the higher complexity of the UNet with ResNet50 encoder.

In the concatenation, torch.cat((x, out), dim=1) leads to significantly better performance rather than torch.cat((out, x), dim=1). I think this is because the checkpoint model was implemented using previous manner.