Description of code – UNet_skeleton.py, resnet_encoder_unet_skeleton.py

```
########## fill in the blanks (Hint : check out the channel size in lecture)
self.convDown1 = conv(in_channels, 64)
self.convDown2 = conv(64, 128)
self.convDown3 = conv(128, 256)
self.convDown4 = conv(256, 512)
self.convDown5 = conv(512, 1024)
self.maxpool = nn.MaxPool2d(2, stride=2)
self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
self.convUp4 = conv(1536, 512)
self.convUp3 = conv(768, 256)
self.convUp2 = conv(384, 128)
self.convUp1 = conv(192, 64)
self.convUp_fin = nn.Conv2d(64, out_channels, 1)
```

```
conv5 = self.convDown5(x)
x = self.upsample(conv5)
#######fill in here ###### hint : concatenation (Lecture slides)
x = torch.cat([conv4, x], dim=1)
x = self.convUp4(x)
x = self.upsample(x)
#######fill in here ###### hint : concatenation (Lecture slides)
x = torch.cat([conv3, x], dim=1)
x = self.convUp3(x)
x = self.upsample(x)
#######fill in here ###### hint : concatenation (Lecture slides)
x = torch.cat([conv2, x], dim=1)
x = self.convUp2(x)
x = self.upsample(x)
#######fill in here ###### hint : concatenation (Lecture slides)
x = torch.cat([conv1, x], dim=1)
x = self.convUp1(x)
out = self.convUp_fin(x)

return out
```

(Above image is captured image of Unet_skeleton.py with *diff* command)

As required by instruction, I filled in the code. In original U-Net's encoder part, through four downsampling (max pool) and five convolutional operations, the size of the image was reduced and the size of the channel was increased. And in decoder part, through four times upsampling and four convolutional operations, the size of image was increased and the size of the channel was decreased. And finally, by convolutional opeartion, the channel size become 2. In addition, the result of the contracting path was added to the expanding path of the same level by combining the channel as the axis through concatenation operation.

```
if self.downsample:
    self.layer = nn.Sequential(
        ########################################
        ############ fill in here
        # Hint : use these functions (conv1x1, conv3x3)
        conv1x1(in_channels, middle_channels, 2, 0),
        conv3x3(middle_channels, middle_channels, 1, 1),
        conv1x1(middle_channels, out_channels, 1, 0),
        ########################################
    )
    self.downsize = conv1x1(in_channels, out_channels, 2, 0)

else:
    self.layer = nn.Sequential(
        ########################################
        ############ fill in here
        conv1x1(in_channels, middle_channels, 1, 0),
        conv3x3(middle_channels, middle_channels, 1, 1),
        conv1x1(middle_channels, out_channels, 1, 0),
        ########################################
    )
    self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)
```

```
super().__init__()
self.n_classes = n_classes
self.layer1 = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=64, kernel_size=7, stride=2, padding=3),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True)#,
)
self.pool = nn.MaxPool2d(3, 2, 1, return_indices=True)

self.layer2 = nn.Sequential(
    ResidualBlock(64, 64, 256, False),
    ResidualBlock(256, 64, 256, False),
    ResidualBlock(256, 64, 256, downsample=True) # Code overlaps with previous ass
)
self.layer3 = nn.Sequential(
    ResidualBlock(256, 128, 512),
    ResidualBlock(512, 128, 512),
    ResidualBlock(512, 128, 512),
    ResidualBlock(512, 128, 512, downsample=False) # Code overlaps with previous a
)
```

(Above image is captured image of resnet_encoder_unet_skeleton.py with *diff* command)

I also filled in this file as required by instruction. First of all, as in project 2, ResNet was implemented, stride and padding were adjusted according to the boolean value of the *downsample*. Also, I coded it as it was while looking at the specifications of layers 2 and 3 required by instruction.

Checking the skeleton code will make for better understandings the structure of ResNet-encoder-Unet. (Overall, it is similar to the original Unet because it has an encoder-decoder structure.)

Description of code – main_skeleton.py, modules_skeleton.py

This part was written without captured images of the code as a volume(1 page of A4) issue.

In main_skeleton.py, the model (UNet, ResNet_encoder_Unet) was selected to initialize it, and the appropriate checkpoint could be retrieved. In addition, the loss function was set to Cross Entropy Loss, and the optimizer was set to Adam.

In modules_skeleton.py, There were many utility functions needed in the project (ex. train_model, accountability_check, get_loss_train, etc.) Among them, the codes of the train_model, get_loss_train, and val_model functions were filled. In the train_model, output and loss were obtained, and optimization (update weights) was performed through backpropagation. In get_loss_train, the loss in training is calculated. In the val_model, the validation of the model is in charge. In this step, the accuracy is measured by unused data during training. In addition, there is a part that generate an image composed of single colors to show how the model understands the segmentation.

Results & Discussions

```
epoch 1 train loss :  0.7254153745223398 train acc :  0.8054500594085655
epoch 1 val loss :  0.9785944513372473 val acc :  0.7560705408319697
Finish Training
Fin
```

Original U-Net

```
epoch 1 train loss :  0.9245471341969216 train acc :  0.737000582969352
epoch 1 val loss :  1.0696808518590153 val acc :  0.7182771235972911
Finish Training
Fin
```

ResNet-encoder-Unet

Original Unet and ResNet_encoder_Unet each showed around 75% and 71% accuracy, respectively, in the validation phase. This seems to be a satisfactory figure. For a better accuracy, discussions might be needed on what tasks need to be undertaken, such as deeper models.