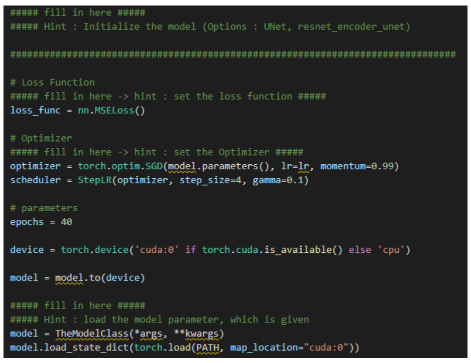
COSE474 Deep Learning

Project #3: Encoder-Decoder Implementation

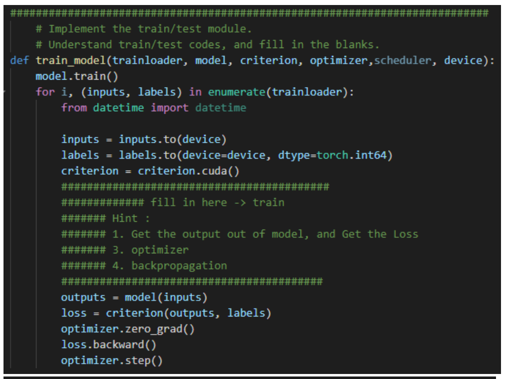
2017320122 김정규

Code1 – Main Skeleton

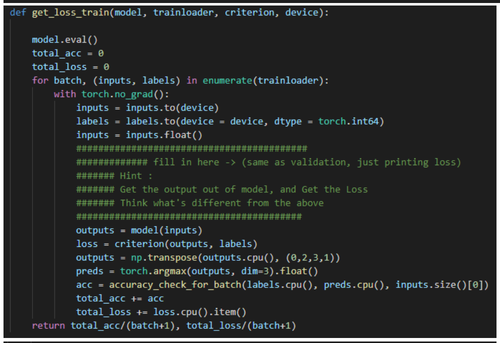


```python
##### fill in here #####
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)

################################################################################

# Loss Function
##### fill in here -> hint : set the loss function #####
loss_func = nn.MSELoss()

# Optimizer
##### fill in here -> hint : set the Optimizer #####
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.99)
scheduler = StepLR(optimizer, step_size=4, gamma=0.1)

# parameters
epochs = 40

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

model = model.to(device)

##### fill in here #####
##### Hint : load the model parameter, which is given
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH, map_location="cuda:0"))
```

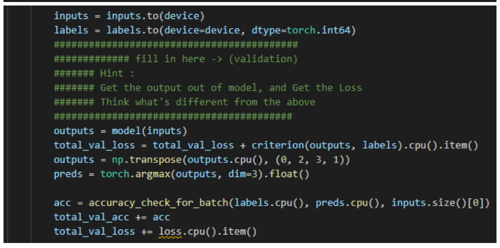
Code2 – Modules Skeleton



```python
################################################################################
    # Implement the train/test module.
    # Understand train/test codes, and fill in the blanks.
def train_model(trainloader, model, criterion, optimizer,scheduler, device):
    model.train()
    for i, (inputs, labels) in enumerate(trainloader):
        from datetime import datetime

        inputs = inputs.to(device)
        labels = labels.to(device=device, dtype=torch.int64)
        criterion = criterion.cuda()
        ###########################################
        ############# fill in here -> train
        ####### Hint :
        ####### 1. Get the output out of model, and Get the Loss
        ####### 3. optimizer
        ####### 4. backpropagation
        ##########################################
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```



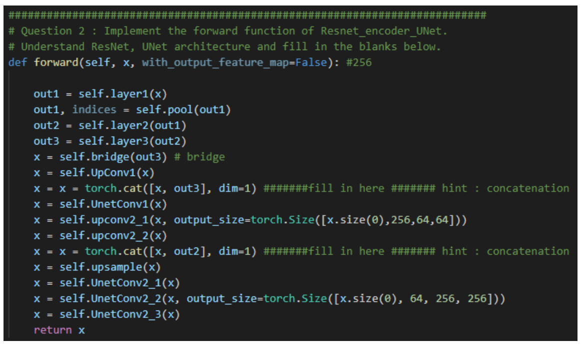
```python
def get_loss_train(model, trainloader, criterion, device):

    model.eval()
    total_acc = 0
    total_loss = 0
    for batch, (inputs, labels) in enumerate(trainloader):
        with torch.no_grad():
            inputs = inputs.to(device)
            labels = labels.to(device = device, dtype = torch.int64)
            inputs = inputs.float()
            ##########################################
            ############# fill in here -> (same as validation, just printing loss)
            ####### Hint :
            ####### Get the output out of model, and Get the Loss
            ####### Think what's different from the above
            #########################################
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            outputs = np.transpose(outputs.cpu(), (0,2,3,1))
            preds = torch.argmax(outputs, dim=3).float()
            acc = accuracy_check_for_batch(labels.cpu(), preds.cpu(), inputs.size()[0])
            total_acc += acc
            total_loss += loss.cpu().item()
    return total_acc/(batch+1), total_loss/(batch+1)
```



```python
            inputs = inputs.to(device)
            labels = labels.to(device=device, dtype=torch.int64)
            ##########################################
            ############# fill in here -> (validation)
            ####### Hint :
            ####### Get the output out of model, and Get the Loss
            ####### Think what's different from the above
            #########################################
            outputs = model(inputs)
            total_val_loss = total_val_loss + criterion(outputs, labels).cpu().item()
            outputs = np.transpose(outputs.cpu(), (0, 2, 3, 1))
            preds = torch.argmax(outputs, dim=3).float()

            acc = accuracy_check_for_batch(labels.cpu(), preds.cpu(), inputs.size()[0])
            total_val_acc += acc
            total_val_loss += loss.cpu().item()
```

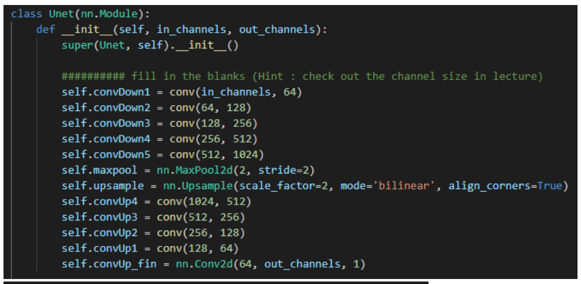
Code3 – Resnet Encoder Unet

Skipped the parts which is overlapping with the previous assignment.

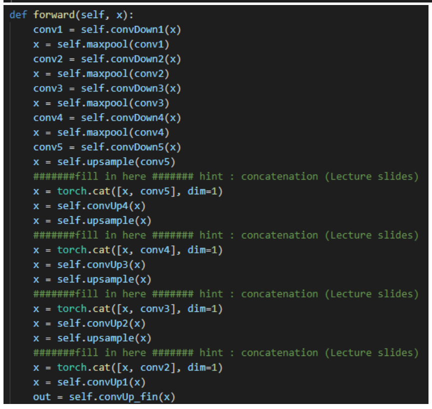


```python
##############################################################################
# Question 2 : Implement the forward function of Resnet_encoder_UNet.
# Understand ResNet, UNet architecture and fill in the blanks below.
def forward(self, x, with_output_feature_map=False): #256

    out1 = self.layer1(x)
    out1, indices = self.pool(out1)
    out2 = self.layer2(out1)
    out3 = self.layer3(out2)
    x = self.bridge(out3) # bridge
    x = self.UpConv1(x)
    x = x = torch.cat([x, out3], dim=1) #######fill in here ####### hint : concatenation
    x = self.UnetConv1(x)
    x = self.upconv2_1(x, output_size=torch.Size([x.size(0),256,64,64]))
    x = self.upconv2_2(x)
    x = x = torch.cat([x, out2], dim=1) #######fill in here ####### hint : concatenation
    x = self.upsample(x)
    x = self.UnetConv2_1(x)
    x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 64, 256, 256]))
    x = self.UnetConv2_3(x)
    return x
```

Code4 – Unet Skeleton



```python
class Unet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Unet, self).__init__()

        ########## 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(1024, 512)
        self.convUp3 = conv(512, 256)
        self.convUp2 = conv(256, 128)
        self.convUp1 = conv(128, 64)
        self.convUp_fin = nn.Conv2d(64, out_channels, 1)
```



```python
def forward(self, x):
    conv1 = self.convDown1(x)
    x = self.maxpool(conv1)
    conv2 = self.convDown2(x)
    x = self.maxpool(conv2)
    conv3 = self.convDown3(x)
    x = self.maxpool(conv3)
    conv4 = self.convDown4(x)
    x = self.maxpool(conv4)
    conv5 = self.convDown5(x)
    x = self.upsample(conv5)
    #######fill in here ####### hint : concatenation (Lecture slides)
    x = torch.cat([x, conv5], dim=1)
    x = self.convUp4(x)
    x = self.upsample(x)
    #######fill in here ####### hint : concatenation (Lecture slides)
    x = torch.cat([x, conv4], dim=1)
    x = self.convUp3(x)
    x = self.upsample(x)
    #######fill in here ####### hint : concatenation (Lecture slides)
    x = torch.cat([x, conv3], dim=1)
    x = self.convUp2(x)
    x = self.upsample(x)
    #######fill in here ####### hint : concatenation (Lecture slides)
    x = torch.cat([x, conv2], dim=1)
    x = self.convUp1(x)
    out = self.convUp_fin(x)
```

Results

Main_skeleton.py, modules_skeleton.py, resnet_encoder_unet_skeleton.py, and UNet_skeleton.py were modified to create a training model based on Resnet and Unet. In most cases, I think I wrote the correct code, but I failed to write a part of modules_skeleton.py. As a result, final testing and debugging of other codes could not be performed. It is probably because I did not understand enough about validation model.