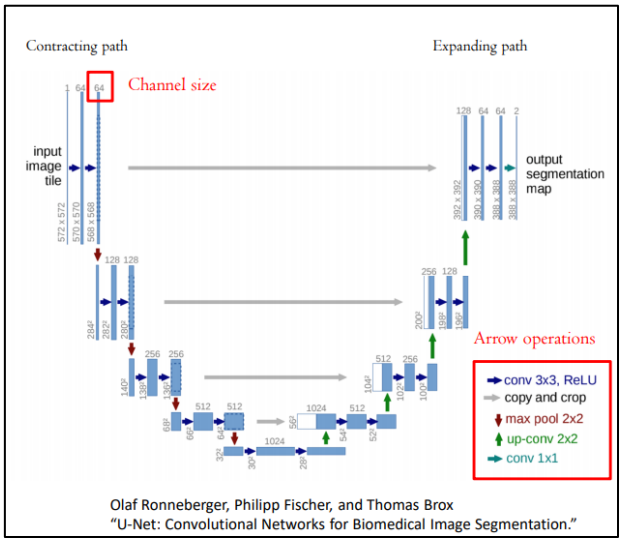
The explanation of the codes

[Question 1]

"UNet_skeleton.py" 코드이다.

Encoder와 decoder로 구성되어 있어서 encoder에서는 downsampling으로 input의 spatial dimension을 줄이고 channel size는 늘린다. 여러 개의 convolutional layer가 있고 downsampling과 upsampling이 이루어진다. Unet의 구조는 아래의 그림에 나와있는 구조를 따라 코드를 작성하였다.




```
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 practice lecture 15 ppt slides 5-6)
        self.convDown1 = conv(in_channels, 64) #blank#)
        self.convDown2 = conv(64, 128)#blank#, #blank#)
        self.convDown3 = conv(128, 256)#blank#, #blank#)
        self.convDown4 = conv(256, 512)#blank#, #blank#)
        self.convDown5 = conv(512, 1024)#blank#, #blank#)
        self.maxpool = nn.MaxPool2d(2, stride=2)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.convUp4 = conv(1536, 512)#blank#, #blank#)
        self.convUp3 = conv(768, 256)#blank#, #blank#)
        self.convUp2 = conv(384, 128)#blank#, #blank#)
        self.convUp1 = conv(192, 64)#blank#, #blank#)
        self.convUp_fin = nn.Conv2d(64, out_channels, 1)#blank#, out_channels, 1)
```

__init__ 에서 unet을 구성하는 layer들을 정의한다. unet에서 사용되는 layer로는 convolutional layer, max pooling layer 그리고 up sampling을 위한 up convolutional layer가 있다. Convolution layer의 경우 input channel과 output channel size에 따라 각각의 layer를 정의해주었다. Pooling layer의 stride크기는 2로 해당 layer를 지나면 channel 수가 1/2배가 된다. Upsampling 시 bilinear를 사용하여 unpooling을 진행하며, 해당 layer를 지나면 channel 수가 2배가 된다.

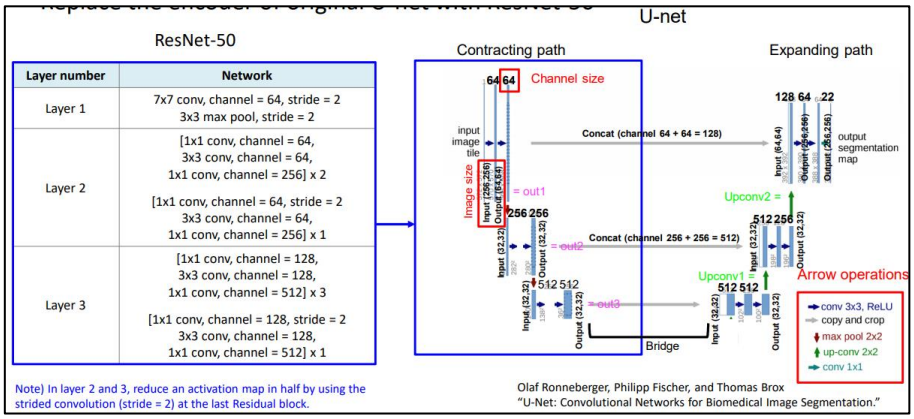
```
def forward(self, x):
    conv1 = self.convDown1(x)
    x = self.maxpool(conv1)
    conv2 = self.convDown2(x)
    x = self.maxpool(conv2)
    conv3 = self.convDown3(x)
    x = self.maxpool(conv3)
    conv4 = self.convDown4(x)
    x = self.maxpool(conv4)
    conv5 = self.convDown5(x)
    x = self.upsample(conv5)
    x = torch.cat([conv4, x], 1)########fill in here ####### hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp4(x)
    x = self.upsample(x)
    x = torch.cat([conv3, x], 1)########fill in here ####### hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp3(x)
    x = self.upsample(x)
    x = torch.cat([conv2, x], 1)########fill in here ####### hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp2(x)
    x = self.upsample(x)
    x = torch.cat([conv1, x], 1)########fill in here ####### hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp1(x)
    out = self.convUp_fin(x)

    return out
```

Forward 함수에서 feed forward의 과정을 정의한다. Encoder와 decoder에는 4번의 downsampling과 upsampling을 하는 neural network 구조가 이루어진다. unet에서는 loss를 낮추기 위하여 encoder에서의 convolutional layer 결과 matrix를 decoder에서 같은 channel size를 가지는, 같은 단계의 matrix에 복사하여 convolution을 진행한다. 이를 구현하기 위하여 pytorch 내장함수인 cat을 이용하여 두개의 matrix를 이어주었으며, 이때 기준은 1로 두개의 matrix가 옆으로 붙도록 하였다.

[Question 2]

"resnet_encoder_unet_skeleton.py" 코드이다.

Resnet을 이용하여 unet을 구현하는데, unet의 encoder 부분을 resnet으로 구성하는 방식이다. ResidualBlock, ResNet50_layer4 class는 assignment 9에서 수행한 내용과 동일하기 때문에 설명을 생략한다. Unet의 구조는 아래의 그림에 나와있는 구조를 따라 코드를 작성하였다.



```
class UNetWithResnet50Encoder(nn.Module):
    def __init__(self, n_classes=22):
        super().__init__()
        self.n_classes = n_classes
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 64, stride=2, kernel_size=7, padding=3),# Code overlaps with previous assignments
            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),  # blank#, #blank#, #blank#, #blank#),
            ResidualBlock(256, 64, 256),  # blank#, #blank#, #blank#, #blank#),
            ResidualBlock(256, 64, 256, downsample=True)# Code overlaps with previous assignments
        )
        self.layer3 = nn.Sequential(
            ResidualBlock(256, 128, 512),  # blank#, #blank#, #blank#),
            ResidualBlock(512, 128, 512),  # blank#, #blank#, #blank#),
            ResidualBlock(512, 128, 512),  # blank#, #blank#, #blank#),
            ResidualBlock(512, 128, 512, downsample=False)  # Code overlaps with previous assignments
```

UNetWithResnet50Encoder class의 __init__ 에서 unet을 구성하는 layer들을 정의한다. unet에서 사용되는 layer 중 convolutional layer의 코드이다. Resnet의 구조를 따라 encoder를 구성하기 때문에 앞에서 정의한 ResidualBlock함수를 사용하여 layer를 만들었다.

```
########################################################################
# Question 2 : Implement the forward function of Resnet_encoder_UNet.
# Understand ResNet, UNet architecture and fill in the blanks below. (20 points)
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 = torch.cat([out2, x], 1)#######fill in here ####### hint : concatenation (Practice Lecture slides 6p)
    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 = torch.cat([out1, x], 1)#######fill in here ####### hint : concatenation (Practice Lecture slides 6p)
    x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 128, 128, 128]))
    x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 128, 256, 256]))
    x = self.UnetConv2_3(x)
    x = self.UnetConv3(x)
    return x
```

Forward 함수에서 feed forward의 과정을 정의한다. Encoder와 decoder에는 3번의 downsampling과 upsampling을 하는 neural network 구조가 이루어진다. 또한 여기서는 "UNet_skeleton.py"의 unet과 달리 bridge함수를 이용하여 encoder와 decoder를 연결한다. unet에서는 loss를 낮추기 위하여 encoder에서의 convolutional layer 결과 matrix를 decoder에서 같은 channel size를 가지는, 같은 단계의 matrix에 복사하여 convolution을 진행한다. 이를 구현하기 위하여 pytorch 내장함수인 cat을 이용하여 두개의 matrix를 이어주었으며, 이때 기준은 1로 두개의 matrix가 옆으로 붙도록 하였다.

[Question 3]

"modules_skeleton.py" 코드이다.

```
#########################################################################
    # Question 3 : Implement the train/test module.
    # Understand train/test codes in Practice Lecture 14, and fill in the blanks.(30 points)
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 (10 points) -> train
        ######## Hint :
        ######## 1. Get the output out of model, and Get the Loss
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        ######## 3. optimizer
        optimizer.zero_grad()
        ######## 4. backpropagation
        loss.backward()
        optimizer.step()
        ########################################
```

Outputs 변수에 model에 inputs을 넣은 결과를 받아오고, criterion함수를 이용하여 결과와 labels 비교를 통한 loss값 계산을 하도록 코드를 작성하였다. Zero-grad 함수로 optimization을 진행하였으며, 앞에서 구한 loss로 backpropagation을 하여 최적의 parameter를 찾도록 했다.

Get_loss_train함수의 코드이다.

```
            ########################################
            ############## fill in here (5 points) -> (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)
            print("Loss: ", loss)
            ########################################
```

model에 input을 넣은 결과인 outputs를 통해 ground truth인 labels와 비교하였다. criterion함수로 loss결과를 얻었으며 지시사항에 따라 loss를 출력하였다.

validation에서 loss를 구하는 코드이다.

```
    for batch, (inputs, labels) in enumerate(valloader):
        with torch.no_grad():
            inputs = inputs.to(device)
            labels = labels.to(device=device, dtype=torch.int64)
            ########################################
            ############## fill in here (5 points) -> (validation)
            ######## 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)
            ########################################
```

Segmentation된 predict 결과를 class에 따라 RGB값으로 바꾸는 코드이다.

```
            for j in range(temp.shape[0]):
                for k in range(temp.shape[1]):
                    ########################################
                    ############## fill in here (10 points)
                    ######## Hint :
                    ######## convert segmentation mask into r,g,b (both for image and predicted result)
                    ######## image should become temp_rgb, result should become temp_label
                    ######## You should use cls_invert[]
                    temp_rgb[j][k]=torch.tensor(cls_invert[temp[j][k]])
                    temp_label[j][k]=torch.tensor(cls_invert[temp_l[j][k]])
                    ########################################
```

cls_invert 딕셔너리에 정의된 class별 rgb값을 각 pixel에 넣어준다. Temp에 image segmentation결과가 있다. 각 pixel이 어떤 class에 속하는지 class label 값이 들어있기 때문에 그 값을 key값으로 cls_invert의 value를 가져왔다. temp_l에는 정답이 들어있다. 이것을 temp_label로 옮겨주었다.

[Question 4]

"main_skeleton.py" 코드이다.

```
##### fill in here #####
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)
# model = UNetWithResnet50Encoder()
# PATH = 'resnet_encoder_unet.pth'

model = Unet(3, 22)
PATH = 'UNet_trained_model.pth'
################################################################################

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

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

# parameters
epochs = 1
```

Resnet을 이용한 unet 또는 unet 둘 중 하나의 model을 선택하여 image segmentation을 할 수 있다. model변수에 각 구조의 class 명을 사용하여 init을 하고, checkpoint를 이용하여 train을 진행하기 위해 PATH에 파일명을 지정해주었다.

Cross entropy로 loss 값을 계산하며, adam optimizer로 optimization을 진행하였다.

```
##### fill in here #####
checkpoint = torch.load(PATH, map_location=device)
model.load_state_dict(checkpoint)
##### Hint : load the model parameter, which is given
```

Load_state_dict함수를 이용하여 checkpoint에 이어서 train을 진행한다.

```
result_save_dir = './history/'
print(str(date))
# result_save_dir = './history/result'+ str(date) +'/'
```

```
savepath1 = "./output/"
print(str(date))
# savepath1 = "./output/model" + str(date) + '/'
```

파일의 결과를 저장할 때 기존 코드를 수정없이 돌리면 error가 발생한다. Date에서 시간으로 인해 파일명에 ":"가 포함되기 때문이다. 따라서 date를 포함하지 않는 파일 경로로 수정해주었다.

```
    if epoch % 4 == 0:
        savepath2 = savepath1 + str(epoch) + ".pth"
        ##### fill in here #####
        torch.save(model.state_dict(), savepath2)
        ##### Hint : save the model parameter
```

내가 설계한 구조로 train한 결과를 pth파일로 checkpoint를 저장한다. save함수를 이용하였다.

Experimental results

[Unet]

- 파일명 ':' 포함으로 오류 발생

```
Error: Creating directory. ./history/result2020-06-30(02:27)/
Error: Creating directory. ./history/result2020-06-30(02:27)/predicted/
Training
Error: Creating directory. ./output/model2020-06-30(02:27)/
```

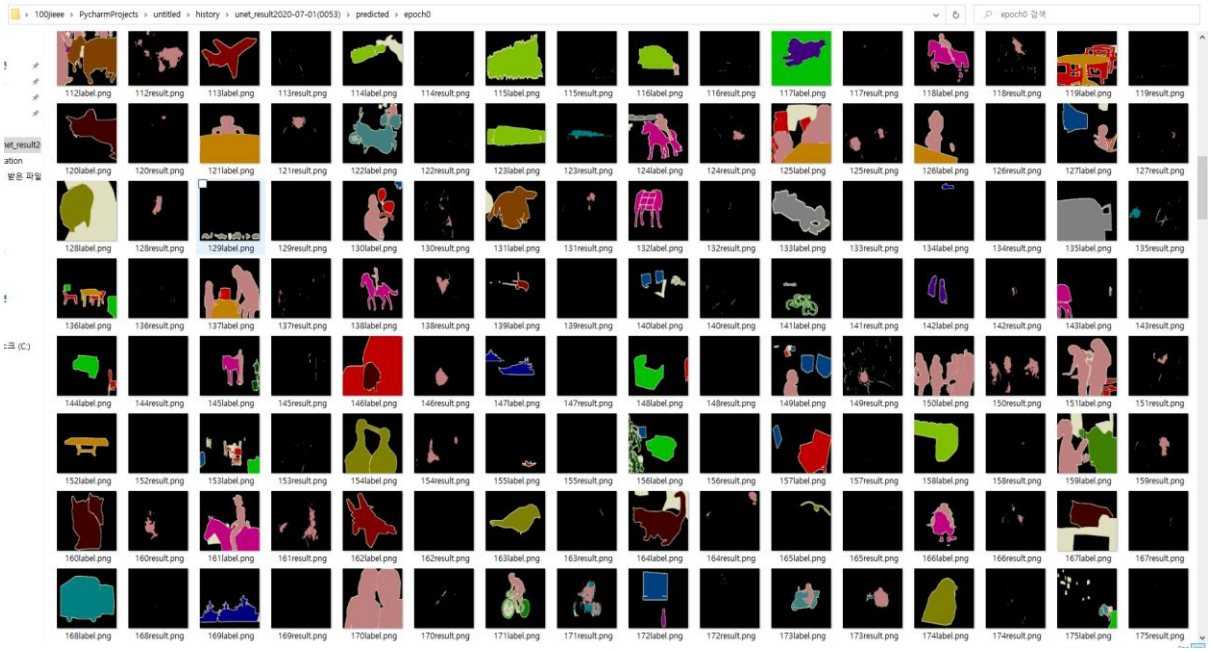
- Torch size로 오류 발생

```
RuntimeError: Error(s) in loading state_dict for Unet:
    size mismatch for convDown1.0.weight: copying a param with shape torch.Size([64, 3, 3, 3]) from checkpoint, the shape in current model is torch.Size([64, 1, 3, 3]).
    size mismatch for convUp4.0.weight: copying a param with shape torch.Size([512, 1536, 3, 3]) from checkpoint, the shape in current model is torch.Size([512, 1024, 3, 3]).
    size mismatch for convUp3.0.weight: copying a param with shape torch.Size([256, 768, 3, 3]) from checkpoint, the shape in current model is torch.Size([256, 512, 3, 3]).
    size mismatch for convUp2.0.weight: copying a param with shape torch.Size([128, 384, 3, 3]) from checkpoint, the shape in current model is torch.Size([128, 256, 3, 3]).
    size mismatch for convUp1.0.weight: copying a param with shape torch.Size([64, 192, 3, 3]) from checkpoint, the shape in current model is torch.Size([64, 128, 3, 3]).
    size mismatch for convUp_fin.weight: copying a param with shape torch.Size([22, 64, 1, 1]) from checkpoint, the shape in current model is torch.Size([2, 64, 1, 1]).
    size mismatch for convUp_fin.bias: copying a param with shape torch.Size([22]) from checkpoint, the shape in current model is torch.Size([2]).
```

- Train, validation loss

```
Loss:  tensor(1.7358, device='cuda:0')
Loss:  tensor(1.6457, device='cuda:0')
epoch 1 train loss :  1.1521497157664833 train acc :  0.7016328527711584
epoch 1 val loss :  1.2049126610595131 val acc :  0.6970087162817467
Finish Training
Fin

Process finished with exit code 0
```

- Image segmentation results (label / result)

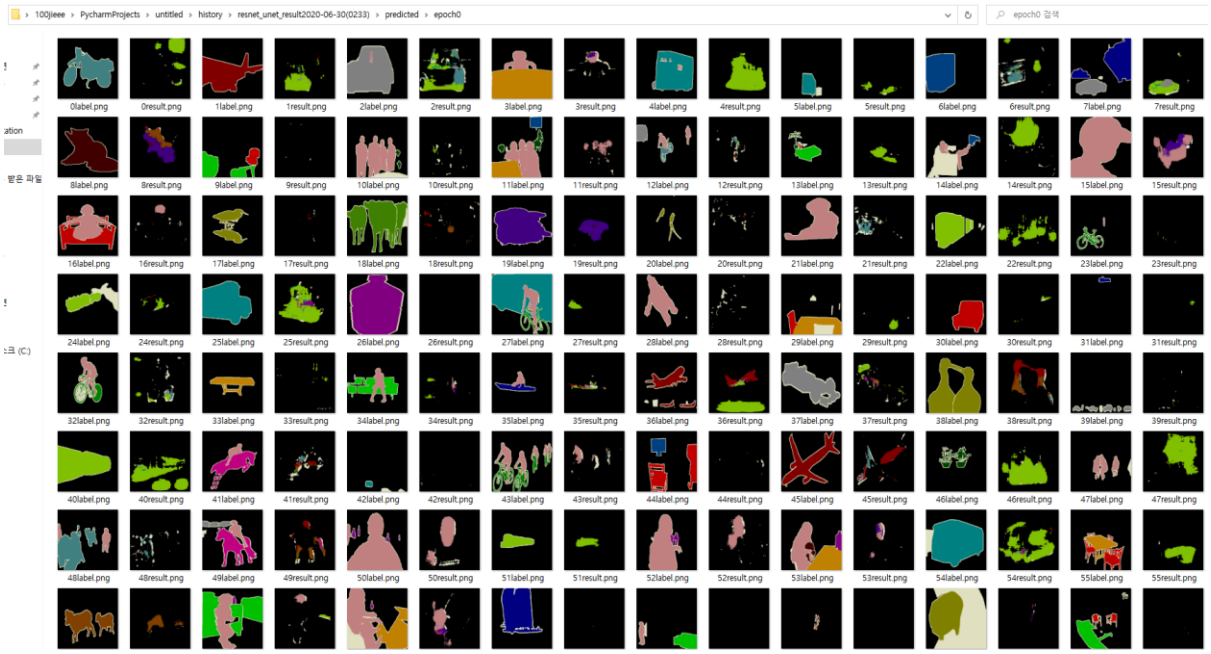


[resnet_encoder_unet]

- Train, validation loss

```
Loss:  tensor(0.8919, device='cuda:0')
Loss:  tensor(1.5961, device='cuda:0')
epoch 1 train loss :  1.1223850734851373 train acc :  0.7036817146070076
epoch 1 val loss :  1.2212495565006178 val acc :  0.6944459889033069
Finish Training
Fin

Process finished with exit code 0
```

- Image segmentation results (label / result)



Analysis

Unet과 resnet_encoder_unet의 result 이미지가 모양과 색상이 유사하지만 label과 정확히 일치하는 이미지가 나오지 않음을 볼 수 있다. 이는 epoch를 1로 설정하여 train되었기 때문에 정확도가 낮아 발생한 문제이며, epoch에 기존 코드대로 40을 주면 이러한 문제를 해결할 수 있을 것이다.

또한 Unet과 resnet_encoder_unet 둘 다 train과 validation의 accuracy 값 차이가 크지 않은 것을 통해 overfitting되지 않고 train이 잘 되었음을 확인할 수 있다.