

1. UNet_skeleton.py

Question1: Implement the UNet model code.

```
def conv(in_channels, out_channels):  
    return nn.Sequential(  
        nn.Conv2d(in_channels, out_channels, 3, padding=1), # 3은 kernel size  
        nn.BatchNorm2d(out_channels),  
        nn.ReLU(inplace=True),  
        nn.Conv2d(out_channels, out_channels, 3, padding=1),  
        nn.BatchNorm2d(out_channels),  
        nn.ReLU(inplace=True)  
    )
```

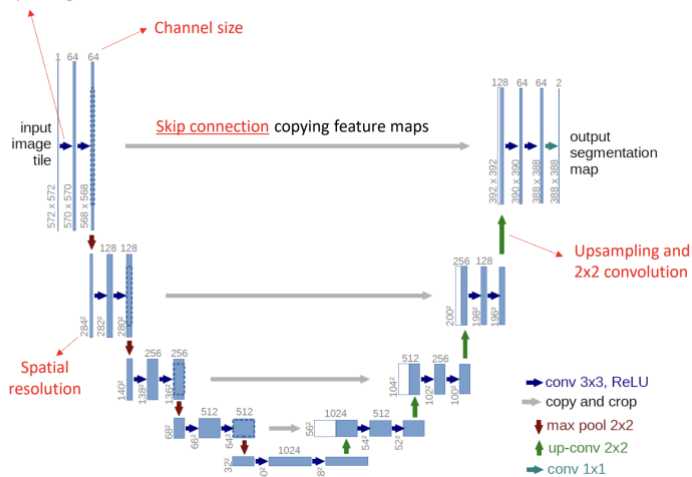
conv, Batch, ReLU 가 두번 반복된 과정을 코드에서 conv 라는 함수를 호출하여 사용할 것이다.
in_channels 과 out_channels 를 매개변수로 가져, out_channels 를 size 로 가지는 값을 출력한다.

Unet 클래스를 정의한다 init 함수는 Unet 객체 생성 시 수행되어 객체를 초기화한다.

Unet 은 conv 세번을 반복하며 padding 이 없어 사이즈는 2 씩 감소하고 channel 크기는 그대로이다. max pool 을 수행해 channel size 는 두 배가 된다. up-conv 에서는 channel size 가 두배 감소한다. up-conv 를 할 때에는 skip connection 이 일어나므로, 같은 높이에 있는 왼쪽 image 의 channel size 를 더한 것이 in_channels 값이 된다.

그림의 Unet 의 각 단계에 따라 값을 채운다

3x3 conv with
no zero-padding and ReLU

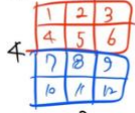


```
class Unet(nn.Module):  
    def __init__(self, in_channels, out_channels):  
        super(Unet, self).__init__()  
  
        ##### fill in the blanks (Hint : check out the channel size in practice lecture 15 ppt slides 5-6)  
        self.convDown1 = conv(in_channels, 64)  
        self.convDown2 = 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)
```

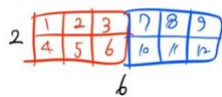
Unet 객체에 forward 함수가 있다.

Unet 에서는 loss 를 줄이기 위해 encoder 의 결과를 decoder 에서 같은 channel size 가지는 같은 단계에 복사해 계산을 진행하는 skip connection 을 수행한다. 이를 구현하기 위해 pytorch 에서 제공하는 함수 cat(concatenation)을 이용하여 두 feature map 을 연결해준다.

$\text{cat}([a, b], \text{dim} = 0)$



$\text{cat}([a, b], \text{dim} = 1)$



cat 함수의 마지막 매개변수는 dimension 으로, 왼쪽 그림과 같이 작용한다. UNet 그림처럼 좌우로 붙이기 위해 1 을 사용한다.

```
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 #####
    x = self.convUp4(x)
    x = self.upsample(x)
    x=torch.cat([conv3,x],1) #####fill in here #####
    x = self.convUp3(x)
    x = self.upsample(x)
    x=torch.cat([conv2,x],1) #####fill in here #####
    x = self.convUp2(x)
    x = self.upsample(x)
    x=torch.cat([conv1,x],1) #####fill in here #####
    x = self.convUp1(x)
    out = self.convUp_fin(x)

    return out
```

2. resnet_encoder_unet_skeleton.py

Resnet 의 일부분을 encoder 에 사용한다.

ResidualBlock, ResNet50_layer4 클래스는 assignment 10 과 동일하기 때문에 생략한다. ResNet 의 동작을 위해 구현한 부분이다.

Question2 Implement the forward function of Resnet_encoder_UNet.

```
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 = 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
```

conv-down 을 resNet 을 이용해 수행하였다. skip connection 또한 resNet 의 결과를 가져온다.

resNet 의 결과 layer1 와 layer2 에 해당하는 out1 과 out2 를 가져와 upconv 의 과정에 붙여준다.

1 번과 같이 cat 함수를 사용한다.

3. modules_skeleton.py

Question3 Implement the train/test module.

```
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()          # gradients를 0으로 설정_ PyTorch가 gradients를 누적하기 때문
        ##### 4. backpropagation
        loss.backward()                 # loss.backward()으로 backpropagation 수행
        optimizer.step()
        #####
```

train model 을 만든다. 이는 assignment10 에서 main.py 의 epoch 만큼 train 을 하는 코드를 참고하였다.

outputs 변수에 model 에 inputs 를 넣은 결과를 저장한다. criterion 함수를 이용하여 결과 output 과 labels 을 비교하여 loss 를 계산하고, loss 변수에 저장한다.

gradients 를 0 으로 설정해 optimization 을 한다. loss.backward()로 backpropagation 을 진행한다.

```
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 (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: "+str(loss))
            #####
            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)
```

train 에서 loss 를 가져온다.

위와 동일하게 input 을 model 에 넣어 outputs 를 구해내고, 이를 GT 인 labels 와 비교하여 loss 를 구한다. loss 를 print 한다.

```
def val_model(model, valloader, criterion, device, dir):
```

```
    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)
            #####
```

validation 에서 loss 를 구한다. 위 코드들과 같다. output 을 구한 뒤 labels 와 비교해 loss 를 구한다.

predict 된 결과를 class 에 따라 RGB 값으로 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]])           # predicted label에 따라 rgb이미지 만든다
        temp_label[j][k]=torch.tensor(cls_invert[temp_l[j][k]])       # GT label에 따라 rgb 이미지 만든다
        #####
```

cls_invert 배열에는, 0~21 에 해당하는 class 를 어떤 색으로 변환할지가 저장되어 있다.

```
cls_invert = {0: (0, 0, 0), 1: (128, 0, 0), 2: (0, 128, 0), # 0:background, 1:aeroplane, 2:bicycle
              3: (128, 128, 0), 4: (0, 0, 128), 5: (128, 0, 128), # 3:bird, 4:boat, 5:bottle
              6: (0, 128, 128), 7: (128, 128, 128), 8: (64, 0, 0), # 6:bus, 7:car, 8:cat
              9: (192, 0, 0), 10: (64, 128, 0), 11: (192, 128, 0), # 9:chair, 10:cow, 11:diningtable
              12: (64, 0, 128), 13: (192, 0, 128), 14: (64, 128, 128), # 12:dog, 13:horse, 14:motorbike
              15: (192, 128, 128), 16: (0, 64, 0), 17: (128, 64, 0), # 15:person, 16:pottedplant, 17:sheep
              18: (0, 192, 0), 19: (128, 192, 0), 20: (0, 64, 128), # 18:sofa, 19:train, 20:tvmonitor
              21: (224, 224, 192)}
```

따라서 각 pixel 이 속하는 class 에 따라 다른 색으로 색칠된다.

temp 에는 예상한 preds 값이 들어있기에 temp_rgb 는 예상한 결과에 대한 이미지이고, temp_l 은 labels 의 값이 들어있기에 temp_label 는 GT 의 결과 이미지이다.

4. main_skeleton.py

Question 4 : Implement the main code.

```
from resnet_encoder_unet_skeleton import *
from UNet_skeleton import *

##### fill in here #####
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)
# model = UNetWithResnet50Encoder(22)
# PATH = 'resnet_encoder_unet.pth'

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

Unet 과 ResNet 을 사용한 Unet 중 하나를 model 로 선택할 수 있도록 한다. 각 class 를 가져오기 위해 코드 맨 위에 import 해준다. checkpoint 를 위해 PATH 에 파일명을 지정한다.

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

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

assignment10 에서 main.py 의 코드를 참고하였다. Loss Function 과 Optimizer 를 setting 해준다.

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

model 을 load 해준다. 저장된 체크포인트 파일에 변수가 로드된다.

epoch 만큼 training 을 진행한다

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

savepath2 라는 파일 이름을 만들고, 이 이름을 가지는 파일에 model 을 저장한다.
즉, epoch 4 번마다 다른 이름의 파일에 결과를 저장한다.

실행 결과:

시간이 너무 오래 걸리는 관계로 batch_size=1, epochs=1 로 진행한다.

오류로 PIL.NEAREST 대신 아래와 같은 코드 사용

```
from modules_skeleton import *
import torchvision.transforms as T

#####
# Question 4 : Implement the main code.
# Understand loading model, saving model, model initialization,
# setting optimizer and loss in Practice Lecture 14, and fill in the blanks.(20 points)

# batch size
batch_size = 1
learning_rate = 0.001

# VOC2012 data directory
data_dir = './VOC2012'
resize_size = 256

transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize([resize_size, resize_size], T.InterpolationMode.BICUBIC),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
```

데이터가 너무 많으므로, datasets.py 에서 아래와 같이 변경해 20 개 이미지로 training 진행

```
if self.flag == 'train':
    self.imgnames = self.lines[:20] # Tip : you can adjust the number
    # self.imgnames = self.lines[self.fold:]
else:
    self.imgnames = self.lines[20:30] # Tip : you can adjust the numb
    # self.imgnames = self.lines[:self.fold]
```

1) Unet

```
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)
# model = UNetWithResnet50Encoder(22)
# PATH = '../trained_model/resnet_encoder_unet.pth'

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

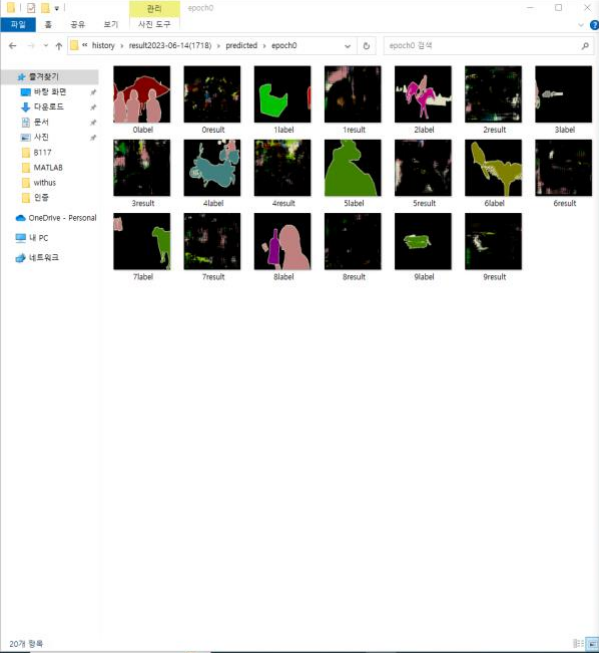
The screenshot shows a Python IDE with two main windows. The top window displays the code for initializing the UNet model, including the hint to use either UNet or resnet_encoder_unet, and the specific initialization of the model and its path. The bottom window shows the training progress, with a grid of images (labeled and predicted) and a terminal output displaying the loss and accuracy over 10 epochs. The training is completed with a message 'Process Finished with exit code 0'.

epoch	train loss	train acc	val loss	val acc
1	2.3858445015888215	0.6447052001953125	2.4721875965595244	0.6531219482421875

2) ResNet-encoder-Net

```
##### fill in here #####
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)
model = UNetWithResnet50Encoder(22)
PATH = '../trained_model/resnet_encoder_unet.pth'

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



```
##### fill in here #####
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)
model1 = UNetWithResnet50Encoder(22)
PATH = 'resnet_encoder_unet.pth'

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

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

# Optimizer
##### fill in here -> hint : set the optimizer #####
optimizer = torch.optim.Adam(model1.parameters(), lr=learning_rate)
```

```
Run: main_skeleton.py
Loss: tensor(0.4220)
Loss: tensor(2.4780)
Loss: tensor(2.1656)
Loss: tensor(1.5532)
Loss: tensor(1.0807)
Loss: tensor(1.9833)
Loss: tensor(2.4481)
Loss: tensor(2.3429)
Loss: tensor(0.8730)
Loss: tensor(0.8222)
Loss: tensor(1.5674)
Loss: tensor(1.2782)
Loss: tensor(1.8238)
Loss: tensor(12.1190)
epoch 1 train loss : 0.264278715848922 train acc : 0.8164306040825
epoch 1 val loss : 1.7121202550354004 val acc : 0.6457351084578313
Finish Training
Fin
Process finished with exit code 0
```

실행 결과, 각 class 별로 구분되어 색이 칠해진 output 이미지 파일이 생성된다.

예) 말들은 모두 같은 색인 분홍색으로 색칠되어있다.

