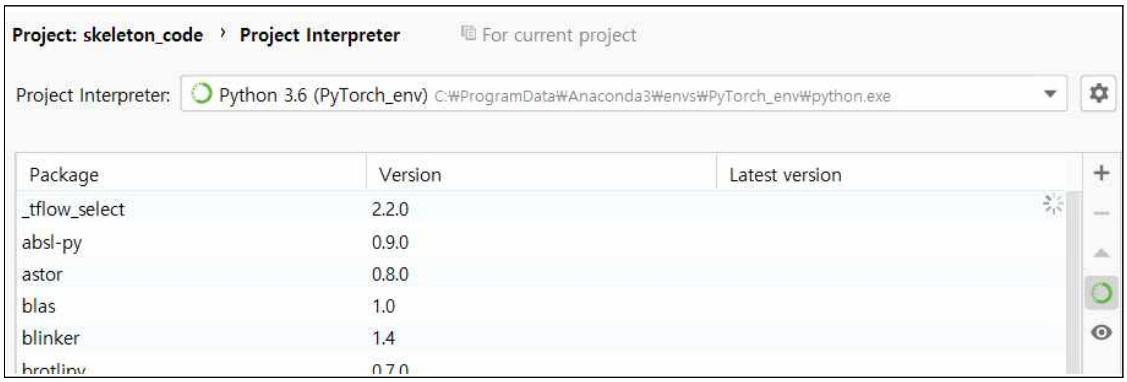
Open Software Project (01)
컴퓨터공학과 1771046 이혜진

Technical Report

Final Project : CNN Architecture / Encoder-Decoder Architecutre

1. Unet_skeleton.py

(1) Code analysis



우선 지난 과제에서 이용했던 가상 머신을 그대로 이용했기 때문에 torch, matplotlib 등의 라이브러리를 추가적으로 설치할 필요가 없었다.

// 처음엔 pip install torch 명령어를 사용하니 제대로 해당 라이브러리를 받을 수 없었다. torch 홈페이지에 들어가 명령어를 확인한 후 주어진 명령어대로 해당 라이브러리를 설치하니, 설치가 성공적으로 완료되었다. 이후 pip install torchvision을 입력하였더니, 이 라이브러리도 정상적으로 설치가 완료되었다.

15_practice에 포함된 링크로 들어가 약 2GB의 파일을 다운받고, 이를 압축을 풀어 사용하였다. tar 파일 압축을 해제하기 위해서 7-zip이라는 파일을 설치하여 사용하였다. 또한 기말 프로젝트는 지난 과제를 참고하여 작성해야 하는 부분이 있었으므로, 지난 과제의 파일들도 함께 준비하였다.

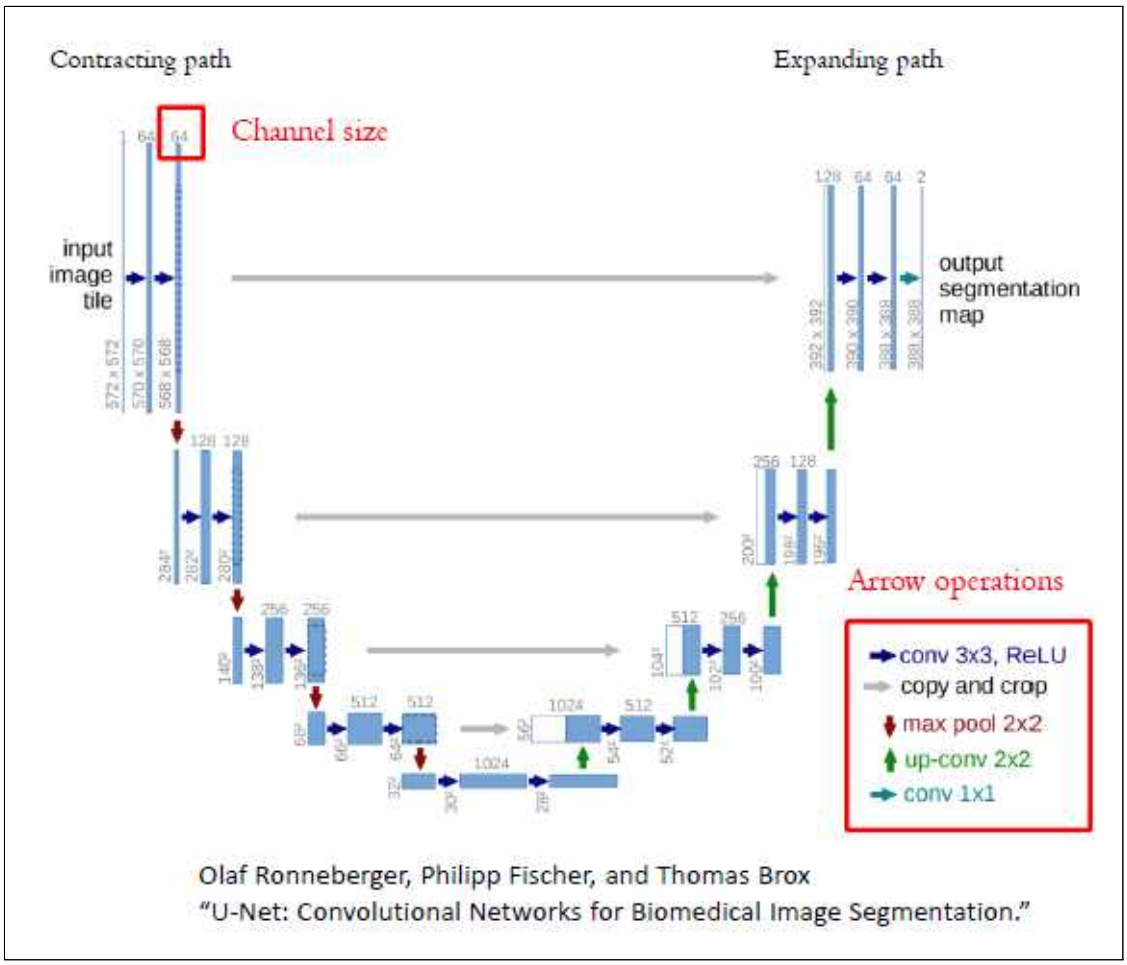
압축 해제된 폴더의 위치와 이름은 다음과 같다.

```python
import torch.nn as nn
import torch

##########################################################################
# Question 1 : Implement the UNet model code.
# Understand architecture of the UNet in practice lecture 15 -> slides 5-6 (30 points)

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)
    )
```

컨볼루션에 대한 함수가 conv로 정의되어 있다. in channels, out channels 두 개의 변수를 이용해, 2d Convolution → Batch Normalization → ReLU → 2d Convolution → Batch Normalization → ReLU 순으로 컨볼루션 과정이 진행되고 있음을 알 수 있다. 이후 conv 함수를 호출할 때마다 총 6단계가 수행되는 것을 파악할 수 있다.

```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 practice lect
        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)
```

Unet class 코드이다. Unet이 생성되면, 채널 값을 인자로 읽어와서 Unet을 실행하게 된다. 컨볼루션을 아랫방향과 윗방향으로 나누어서 실행하게 된다. 채널값을 확인하면 늘어났다가 줄어드는 것을 확인할 수 있다. 중간에 max pool 과정도 포함되어 있다. 이곳에서 정의한 다양한 함수들은 아래의 forward 부분에서 차례로 적용된다.

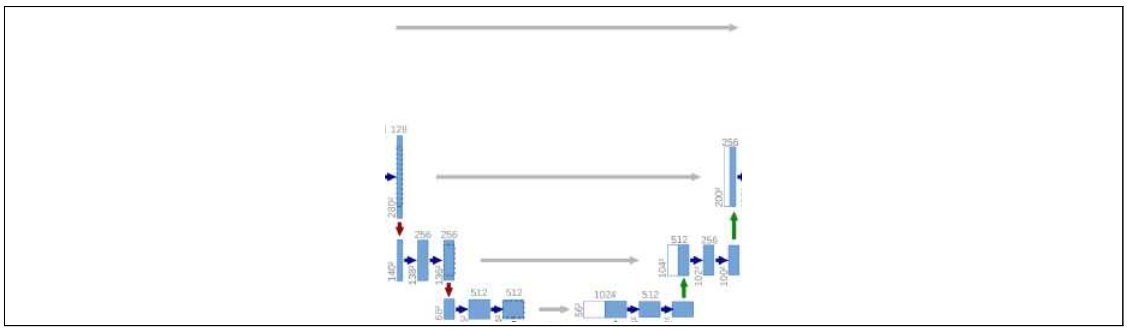


위의 함수를 구현할 때 참고한 이미지이다. 빨간색 네모로 Channel size가 적혀있는데, 해당 내용을 차례로 conv 부분의 빈칸에 작성해주면 된다.

```
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((x, conv4), dim=1) # hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp4(x)
    x = self.upsample(x)
    x = torch.cat((x, conv3), dim=1) # hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp3(x)
    x = self.upsample(x)
```

```
    x = torch.cat((x, conv3), dim=1) # hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp3(x)
    x = self.upsample(x)
    x = torch.cat((x, conv2), dim=1) # hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp2(x)
    x = self.upsample(x)
    x = torch.cat((x, conv1), dim=1) # hint : concatenation (Practice Lecture slides 6p)
    x = self.convUp1(x)
    out = self.convUp_fin(x)

    return out
```

해당 정의는 위 클래스 내에 정의한 함수들을 실제 forward 방향으로 학습을 실행할 때 사용되는 함수이다. 입력 인자를 바탕으로 차례대로 과정을 수행한다. Down 부분에서는 위의 정의 함수들을 차례로 이용하며, 이들 사이에 maxpool이 교차로 나타난다. Up 부분에서는 maxpool은 사용하지 않고, 대신 upsample과 concatenation이 이용된다. 이는 torch 라이브러리 내에 cat 함수로 정의되어 있으며 이를 그대로 사용하였다.



concatenation은 위의 그림에서 회색 화살표 부분에 해당되며, 이를 이용하기 위해서 단순하게 x에 계속 컨볼루션을 진행하는 것이 아니라, conv1, conv2, conv3, conv4 변수를 각각 이용해 해당 과정의 컨볼루션 결과를 버리지 않고 따로 저장해놓는 것이다.

2. resnet_encoder_unet_skeleton.py

(1) Code analysis

```python
# 1x1 convolution
def conv1x1(in_channels, out_channels, stride, padding):
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels)
    )
    return model


# 3x3 convolution
def conv3x3(in_channels, out_channels, stride, padding):
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels)
    )
    return model
```

1x1 컨볼루션과, 3x3 컨볼루션에 관한 정의이다. 2d convolution을 적용하 batch normalization을 순차적으로 적용한다. 이때 두 함수의 차이점은 kernel size가 다르고, 나머지 부분은 모두 동일하다.

```python
##########################################################################
# Code overlaps with previous assignments : Implement the "bottle neck building block" part.
# Hint : Think about difference between downsample True and False. How we make the difference by code?
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, middle_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        self.downsample = downsample

        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)
```

Resnet을 구현한 이전의 과제에서 크게 벗어나지 않는 과정이 진행된다. down sample 과정에서는 conv1x1 → conv3x3 → conv1x1 의 차례로 컨볼루션이 진행된다. 그리고 down size 과정이 conv1x1를 통해 이루어진다.

```
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)
self.activation = nn.ReLU(inplace=True)
```

마찬가지로 컨볼루션을 수행하는데, 들어가는 인자값이 조금 다른 것을 알 수 있다. 채널의 수를 같게 하기 위해서 make_equal_channel 함수를 생성한다. 마지막으로 두 과정 모두 동일하게 activation 함수로 ReLU를 적용한다.

```
def forward(self, x):
    if self.downsample:
        out = self.layer(x)
        x = self.downsize(x)
        return self.activation(out + x)
    else:
        out = self.layer(x)
        if x.size() is not out.size():
            x = self.make_equal_channel(x)
        return self.activation(out + x)
```

실제 채워 작성할 부분은 없는 코드이고, 이전 과제와 약간 다른 부분이 존재하는 코드이다. layer를 down size 하거나, equal channel로 만들고 위에서 정의한 ReLU를 적용하는 방법으로 진행된다.

```
def conv(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),  # When inplace = TRUE, ReLU m
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
```

해당 py에서 사용되는 컨볼루션 함수는 다음과 같다. 2d convolution → batch normalization → ReLU → 2d convoution → batch normalization

→ ReLU 순으로 함수가 진행된다. 컨볼루션 한 번에 총 6개의 과정이 실행됨을 알 수 있다.

```
class UNetWithResnet50Encoder(nn.Module):
    def __init__(self, n_classes=22):
        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), # Code ov
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)#,
        )
        self.pool = nn.MaxPool2d(3, 2, 1, return_indices=True)

        self.layer2 = nn.Sequential(
            ResidualBlock(in_channels=64, middle_channels=64, out_channels=256, downsample=False),
            ResidualBlock(in_channels=256, middle_channels=64, out_channels=256, downsample=False),
            ResidualBlock(in_channels=256, middle_channels=64, out_channels=256, downsample=True) #
        )
```

해당 클래스가 실행되면, init 함수가 제일 우선적으로 실행된다. 첫 번째 레이어에서는 컨볼루션, normalization, ReLU가 차례로 실행된다. 그리고 maxpool을 적용한다.

두 번째 레이어에서는 ResidualBlock 함수가 3회 실행되는데, 해당 코드는 이전 Lec14의 과제와 동일한 내용이므로 코드를 그대로 가져와 사용하였다. 이전 과제와 같은 채널 값을 채워 넣었다.

```
self.layer3 = nn.Sequential(
    ResidualBlock(in_channels=256, middle_channels=128, out_channels=512, downsample=False),
    ResidualBlock(in_channels=512, middle_channels=128, out_channels=512, downsample=False),
    ResidualBlock(in_channels=512, middle_channels=128, out_channels=512, downsample=False),
    ResidualBlock(in_channels=512, middle_channels=128, out_channels=512, downsample=True) #
)
self.bridge = conv(512, 512)
self.UnetConv1 = conv(512, 256)
self.UpConv1 = nn.Conv2d(512, 256, 3, padding=1)

self.upconv2_1 = nn.ConvTranspose2d(256, 256, 3, 2, 1)
self.upconv2_2 = nn.Conv2d(256, 64, 3, padding=1)

self.unpool = nn.MaxUnpool2d(3, 2, 1)
self.UnetConv2_1 = nn.ConvTranspose2d(64, 64, 3, 2, 1)
self.UnetConv2_2 = nn.ConvTranspose2d(128, 128, 3, 2, 1)
self.UnetConv2_3 = nn.Conv2d(128, 64, 3, padding=1)

self.UnetConv3 = nn.Conv2d(64, self.n_classes, kernel_size=1, stride=1)
```

세 번째 레이어에서는 ResidualBlock 함수가 3회 적용된다. 해당 부분도

Lec14의 과제와 동일하게 작성하였다. 이후 브릿지 과정과 Up convolution, Unet convolution 과정을 거치며 채널 사이즈를 조정한다.

```
##########################################################################
# 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((x, out2), dim=1) # 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((x, out1), dim=1) # 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 함수를 정의하는 과정이다. Resnet, Unet 구조에 대한 전반적인 이해를 통해 아래의 빈칸을 채워야 한다. 컨볼루션과 브릿지 과정을 수행한 후, concatenation 과정을 수행해야하는데, out2와 out1을 이용해 cat을 수행한다. 앞선 Unet에서와는 달리 총 2회의 cat 과정이 수행된다.

3. modules_skeleton.py

(1) Code analysis

```
##########################################################################
    # 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
        ####### 3. optimizer
        ####### 4. backpropagation
        ########################################
        # 1. Get the output out of model, and Get the Loss
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # 3. optimizer & 4. backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

해당 파일은 학습과 테스트와 관련된 모듈들을 포함하고 있는 파일이다. 이전의 두 개의 파일과 마찬가지로 이후 main 파일에서 import 되어 이용된다.

train_model 함수는 학습을 위한 모델을 생성하는 함수이다. 먼저 받아온 인수들로 inputs, labels, criterion을 정의한다.

아래의 Hint에 따라 코드를 만들어 나간다. outputs는 inputs을 통해 정의될 수 있으며, 앞서 정의된 criterion을 통해 loss를 구한다. 그리고 optimizer를 실행하고 backpropagation, 즉 역전파 과정을 진행한다. 마지막으로 optimizer를 한번 더 수행한다.

```
def accuracy_check(label, pred):
    ims = [label, pred]
    np_ims = []
    for item in ims:
        item = np.array(item)
        np_ims.append(item)
```

```
    compare = np.equal(np_ims[0], np_ims[1])
    accuracy = np.sum(compare)
    return accuracy / len(np_ims[0].flatten())

def accuracy_check_for_batch(labels, preds, batch_size):
    total_acc = 0
    for i in range(batch_size):
        total_acc += accuracy_check(labels[i], preds[i])
    return total_acc/batch_size
```

정확도를 체크 할 때 이용하기 위한 두 개의 함수이다. label과 preds를 이용해 현재 정확도를 구한다. 아래의 함수는 각 batch 마다 정확도를 구하는데에 이용되는 함수이다.

```
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)

            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)
```

학습 과정에서 loss를 구하는 함수이다. 어떤 학습에서 항상 그 학습의 목표는 loss를 최소한으로 만드는 것이다. total_loss를 정의하여 계속 loss를 합쳐나가 최종적인 loss를 도출한다. 모델의 outputs과 loss를 구해 total_loss의 계산에 이용한다.

```
from PIL import Image
def val_model(model, valloader, criterion, device, dir):

    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)}
    total_val_loss = 0
    total_val_acc = 0
    n=0
```

cls_invert는 정의된 모델에 관련된 rgb 값들을 저장하고 있는 배열이다. 예를 들어 2는 bicycle, 즉 자전거로 해당 rgb 값을 기준으로 갖는다.

```
total_val_loss = 0
total_val_acc = 0
n=0

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)

        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()
```

차례로 for문을 돌며 아래의 과정을 수행한다. 여기서도 마찬가지로 inputs으로부터 outputs을 도출하며, outputs와 lables을 인자로 전달해 criterion 함수를 수행하여 loss를 도출한다. 아래의 과정은 위의 get_loss_train 함수와 유사하다.

```
        for i in range(preds.shape[0]):
            temp = preds[i].cpu().data.numpy()
            temp_l = labels[i].cpu().data.numpy()
            temp_rgb = np.zeros((temp.shape[0], temp.shape[1], 3))
            temp_label = np.zeros((temp.shape[0], temp.shape[1], 3))

            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 (boi
                    ####### image should become temp_rgb, result shou
                    ####### You should use cls_invert[]
                    ########################################
                    for l in cls_invert:
                        temp_rgb[j][k]=cls_invert[temp[j][k]]
                        temp_label[j][k]=cls_invert[temp_l[j][k]]

            img = inputs[i].cpu()
            img = np.transpose(img, (2, 1, 0))

            img_print = Image.fromarray(np.uint8(temp_label))
            mask_print = Image.fromarray(np.uint8(temp_rgb))

            img_print.save(dir + str(n) + 'label' + '.png')
            mask_print.save(dir + str(n) + 'result' + '.png')

            n += 1

    return total_val_acc/(batch+1), total_val_loss/(batch+1)
```

temp_rgb와 temp_label을 생성해 해당 크기에 맞추어 0값을 채워넣는다. segmentation mask를 rgb로 바꾸기 위해 cls_invert 값을 이용한다. 이후 img_print, mask_print png 이미지를 각각 저장하고 이들 해당 dir 디렉토리 내에 저장된다.

4. main_skeleton.py

(1) Code analysis

```
from datasets import Loader
import torchvision.transforms as transforms
import PIL.Image as PIL
from modules_skeleton import *
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import StepLR
from resnet_encoder_unet_skeleton import *

################################################################################
# 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 = 16
learning_rate = 0.001

# VOC2012 data directory
data_dir = "C:\\Users\\HEAJIN\\Desktop\\VOCtrainval_11-May-2012\\VOCdevkit\\VOC2012"
resize_size = 256

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

파일명을 동일하게 맞추기 위해서 modules, resnet_encoder_unet에 각각 skeleton을 붙여주었다. 해당 파일은 실제 main 함수를 실행시키는 파일로, 앞서 작성했던 파일들을 import해서 사용한다.

먼저 batch_size와 running rate 값을 정의한다. 이후 학습에 이용할 데이터를 불러오기 위해 압축 해제된 파일의 내부로 들어가 data_dir를 설정한다. resize와 normalization 과정을 포함한 transforms를 정의한다.

```
print("trainset")
trainset = Loader(data_dir, flag ='train', resize = resize_size, transforms = transforms)
print("valset")
valset = Loader(data_dir, flag = 'val', resize = resize_size, transforms = transforms)

print("tainLoader")
trainLoader = DataLoader(trainset, batch_size = batch_size, shuffle=True)
print("valLoader")
validLoader = DataLoader(valset, batch_size = batch_size, shuffle=True)
```

Loader를 통해 trainset과 valset을 로드한다. Loader는 datasets.py 내에 존재하는 함수이다. 마찬가지로 trainLoader와 validLoader를 DataLoader 함수를 통해 정의한다.

```
##### fill in here #####
##### Hint : Initialize the model (Options : UNet, resnet_encoder_unet)
###############################################################################
model = Unet(in_channels=3, out_channels=22)
# model = UNetWithResnet50Encoder(in_channels=3, out_channels=22)

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

먼저 사용하기 위한 모델을 정의한다. Unet 클래스를 불러와 모델을 정의하는데, in_channels=3, out_channels=22를 사용하였다. 이는 학습에 이용되는 해당 이미지가 rgb 값을 갖기 때문이다. UnetWithResent50Encoder는 Resnet을 이용하여 학습을 실행하는 과정이고, 해당 모델은 주석처리 하였다. Lec14의 과제와 동일하게 모델을 변경하며 사용할 수 있다.

nn.CrossEntropyLoss : get Cross Entropy Loss, contains nn.LogSoftmax() and nn.NLLLoss()

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

loss function은 Lec14 practice에 포함된 Cross Entropy Loss Function을 사용하였다.

```
# Optimizer
##### fill in here -> hint : set the Optimizer #####
optimizer = torch.optim.RMSprop(model.module.parameters(), lr=0.001)
scheduler = StepLR(optimizer, step_size=4, gamma=0.1)

# parameters
# epochs = 40
epochs = 1

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

model = model.to(device)
```

optimizer와 scheduler를 정의하였다. Lec14의 과제와 동일하게 해당 부분

을 작성하였다. epochs는 원래 40으로 정의되어 있었으나, 빠르고 간결한 학습을 위해 이를 주석처리 하고 1로 값을 재정의 하였다. device는 gpu인 cuda가 있을 경우 이를 사용하지만, 존재하지 않을 경우 cpu를 이용한다. 해당 컴퓨터에서는 cuda를 설치하지 않았기 때문에 학습에 cpu를 이용한다. device를 통해 모델을 불러온다.

```
##### fill in here #####
##### Hint : load the model parameter, which is given
def update_lr(optimzer, lr):
    for param_group in optimzer.param_groups:
        param_group['lr']=lr

total_step=len(trainLoader)
current_lr=learning_rate
```

이전 과제와 유사하게 모델 파라미터와 관련된 값들을 정의한다.

```
# Train
import os
from datetime import datetime

now = datetime.now()
date = now.strftime('%Y-%m-%d(%H:%M)')
def createFolder(dir):
    try:
        if not os.path.exists(dir):
            os.makedirs(dir)
    except OSError:
        print('Error: Creating directory. ' + dir)

result_save_dir = './history/result'+date+'/'
createFolder(result_save_dir)
predict_save_dir = result_save_dir + 'predicted/'
createFolder(predict_save_dir)

history = {'train_loss':[], 'train_acc':[], 'val_loss':[], 'val_acc':[]}

print("Training")
```

실제 학습이 일어나는 부분이다. 현재 시간을 출력하기 위해 datetime을

import 하여 사용한다. Folder를 생성하기 위해서 os를 import 한다. 학습을 하며 나온 결과들을 저장하기 위해서 folder를 생성하는 함수, createFolder를 정의한다. 그 아래는 여러 디렉토리에 대한 정보들이다. “Training”이라는 스트링을 출력한 후 실제 학습을 시작한다.

```
savepath1 = "./output/model" + date + '/'
createFolder(savepath1)

for epoch in range(epochs):

    train_model(trainLoader, model, criterion, optimizer, scheduler, device)
    train_acc, train_loss = get_loss_train(model, trainLoader, criterion, device)
    print("epoch", epoch + 1, "train loss : ", train_loss, "train acc : ", train_acc)

    predict_save_folder = predict_save_dir + 'epoch' + str(epoch) + '/'
    createFolder(predict_save_folder)
    val_acc, val_loss = val_model(model, validLoader, criterion, device, predict_save_folder)
    print("epoch", epoch + 1, "val loss : ", val_loss, "val acc : ", val_acc)

    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)

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

print('Finish Training')
```

여러 결과들이 저장될 경로는 savepath1, savepath2이다. 이들은 date, epoch 값을 이용해 정의된다. createFoldr 함수를 통해 먼저 savepath1 경로의 폴더를 생성한다.

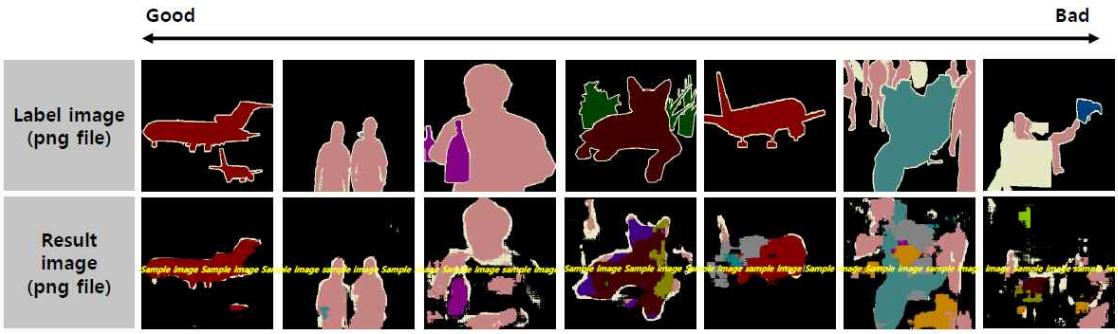
앞서 정의한 epochs(=1 #40)만큼 반복하며 학습을 진행한다. history에는 현재 수행되고 있는 값들을 append한다.

만약 epoch가 4로 나누었을 때, 나머지가 0일 경우 (0, 4, 8...) 에는 해당 정보를 savepath2 경로에 저장한다. 이때 epoch 값에 따라 결과는 각각 다른 폴더에 저장된다. epochs만큼의 반복이 끝나면 “Finish Training”을 출력하며 학습을 종료한다.

```
import matplotlib.pyplot as plt

plt.subplot(2,1,1)
plt.plot(range(epoch+1), history['train_loss'], label='Loss', color='red')
plt.plot(range(epoch+1), history['val_loss'], label='Loss', color='blue')

plt.title('Loss history')
plt.xlabel('epoch')
plt.ylabel('loss')
# plt.show

plt.subplot(2,1,2)
plt.plot(range(epoch+1), history['train_acc'], label='Accuracy', color='red')
plt.plot(range(epoch+1), history['val_acc'], label='Accuracy', color='blue')

plt.title('Accuracy history')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.savefig(result_save_dir+'result')

print("Fin")
```

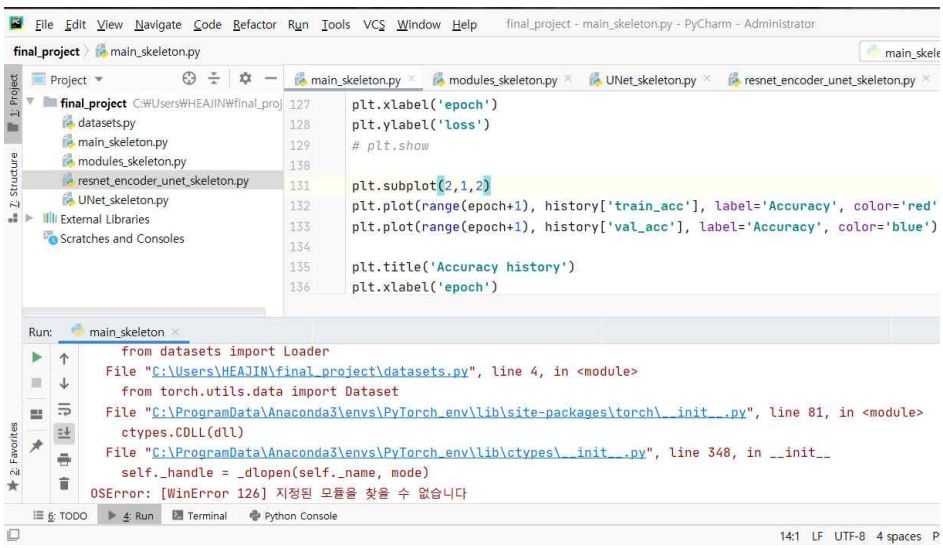
결과를 출력하기 위해 matplotlib 라이브러리를 import한다. 구역별로 나누고 color를 설정한 후, title을 정의한다. 이후 결과를 출력하고, 이를 저장한다. 마지막으로 “Fin”을 출력한 후 프로그램을 최종 종료한다.

(2) Result analysis

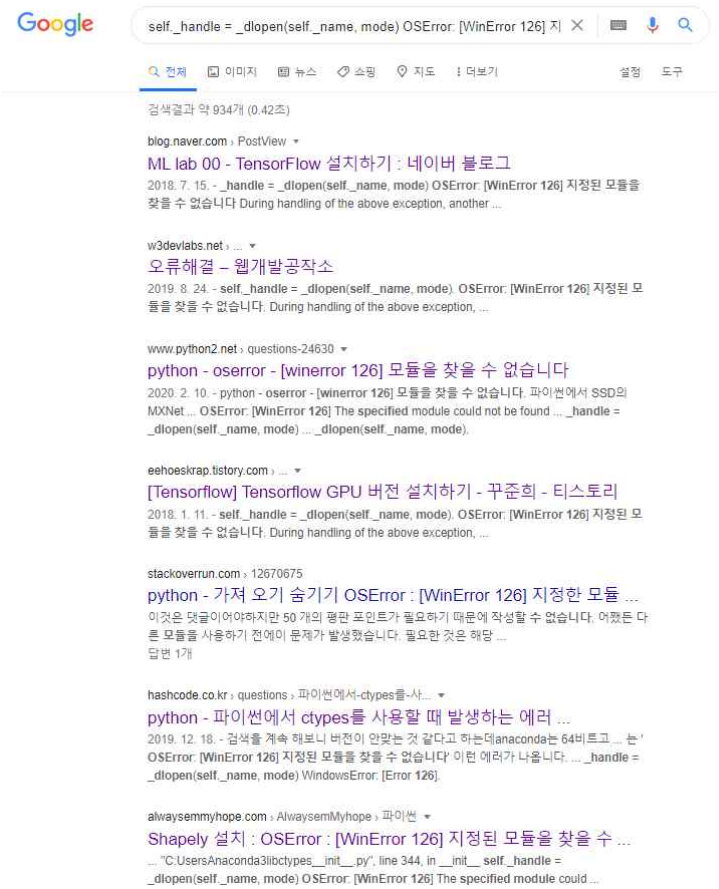


모델을 학습시켰을 때 나오는 결과는 위와 같아야 한다. 학습이 잘 될수록 이미지가 더욱 깔끔하게 나누어지는 것을 알 수 있다.

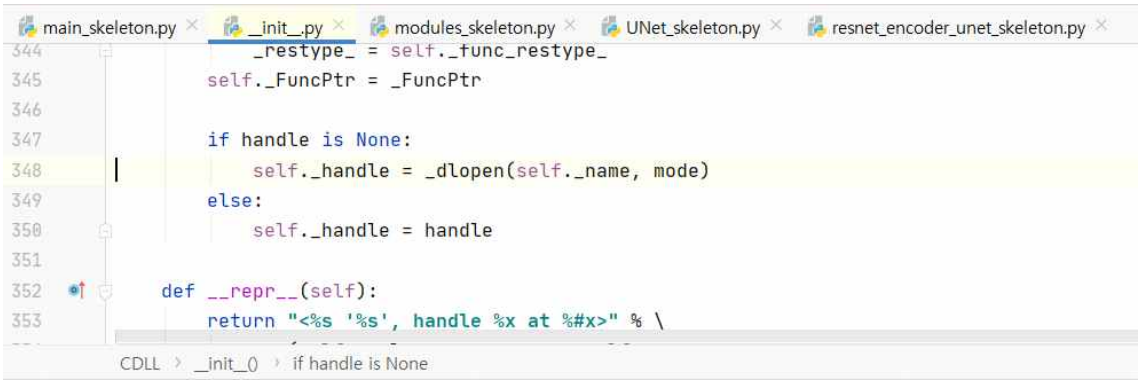
하지만 현재 제 컴퓨터에서는 해당 코드를 돌려볼 수가 없었습니다. 지난 Lec14의 과제에서는 오류가 발생하지 않았고 완전히 동일한 환경에서 프로젝트를 진행하였으나 다음과 같은 오류가 발생하였습니다.



해당 문제를 해결하기 위해서 새로운 프로젝트를 생성해 보기도 하고.. 구글링을 통해 많은 방법들을 시도해보았지만..



결국에는 문제를 해결할 수 없었습니다.. 이번 과제가 기말 프로젝트이고 성적에 10%가 반영되는 만큼 다른 사람에게 부탁하기란 어려웠고, (학습에 소요되는 시간도 꽤 오래 걸린다고 알고 있습니다..) 재설치 또한 진행해보았지만, 왜 이러한 문제가 발생하는지 파악하기가 어려웠습니다..



문제가 발생하는 해당 파일로 이동할 경우, 직접 작성한 파일이 아닌 기본 라이브러리에 포함되는 __init__ 함수에서 문제가 발생하고 있는 것 같았습니다. 하지만 이전 과제 뿐만 아니라, 다른 과제들을 할 때에는 이러한 문제가 발생하지 않았던 것을 보아 개인적으로 어느 부분에선가 기말 프로젝트의 파일들과의 충돌이 발생하고 있다고 추측하였습니다. (잘 모르겠어요..T-T)

따라서 학습 및 결과 출력은 어려웠으며 맨 처음 main의 첫 번째 줄 import에서부터 문제가 발생하였기 때문에 디버깅 과정조차도 어려웠습니다.. 그래서 작성한 코드 및 Techinal Report와 Readme 파일만 제출하겠습니다. 죄송하고 감사드립니다.