UNet

1. Overall Explanation

: this program trains UNet model with Pascal VOC 2012 datasets and conduct the image segmentation

2. Code

UNet.py

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)  
 )  
  
  
class Unet(nn.Module):  
 def \_\_init\_\_(self, in\_channels, out\_channel):  
 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(1024, 512)  
 self.convUp3 = conv(512, 256)  
 self.convUp2 = conv(128, 64)  
 self.convUp\_fin = nn.Conv2d(64, out\_channels, 1) # 1x1convolution  
  
  
  
 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 (Practice Lecture slides 6p)  
 x = torch.cat([x, conv4], dim=1)  
 x = self.convUp4(x)  
 x = self.upsample(x)  
 #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)  
 x = torch.cat([x, conv3], dim=1)  
 x = self.convUp3(x)  
 x = self.upsample(x)  
 #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)  
 x = torch.cat([x, conv2], dim=1)  
 x = self.convUp2(x)  
 x = self.upsample(x)  
 #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)  
 x = torch.cat([x, conv1], dim=1)  
 x = self.convUp1(x)  
 out = self.convUp\_fin(x)  
  
 return out

modules.py

import numpy as np  
import torch  
  
###########################################################################  
 # 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  
 outputs = model(inputs)  
 loss = criterion(outputs, labels)  
  
 optimizer.zero\_grad() # set gradients to zero  
 loss.backward() # conduct backpropagation  
 optimizer.step()  
  
 #########################################  
  
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  
  
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)  
  
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  
  
 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()  
  
 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 (both for image and predicted result)  
 ####### image should become temp\_rgb, result should become temp\_label  
 ####### You should use cls\_invert[]  
 temp\_rgb = cls\_invert[temp\_l[j]]  
 temp\_label = cls\_invert[temp\_l[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)

main.py

from datasets import Loader  
import torchvision.transforms as transforms  
import PIL.Image as PIL  
from modules import \*  
from torch.utils.data import DataLoader  
from torch.optim.lr\_scheduler import StepLR  
from resnet\_encoder\_unet import \*  
from UNet 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 = "./VOCdevkit"  
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))  
])  
  
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("trainLoader")  
trainLoader = DataLoader(trainset, batch\_size = batch\_size, shuffle=True)  
print("valLoader")  
validLoader = DataLoader(valset, batch\_size = batch\_size, shuffle=True)  
  
##### fill in here #####  
##### Hint : Initialize the model (Options : UNet, resnet\_encoder\_unet)  
model = Unet()  
# model = UNetWithResnet50Encoder()  
###############################################################################  
  
# 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)  
  
# parameters  
epochs = 40  
device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')  
model = model.to(device)  
  
##### fill in here ####s#  
##### Hint : load the model parameter, which is given  
model.load\_state\_dict(torch.load('./trained\_model/UNet\_trained\_model.pth'))  
# model.load\_state\_dict(torch.load('./trained\_model/resnet\_encoder\_unet.pth'))  
model.eval()  
  
# 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")  
  
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)  
 # torch.save(model.state\_dict(), savepath2)  
  
print('Finish Training')  
  
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")

3. Explanation and Analysis

UNet architecture consists of a contracting path an an expansive path. The contracting path follows consists of two 3x3 convolutions, each followed by a ReLU and a 2x2 max pooling for downsampling. At downsampling step, the number of feature channels is doubled. The expansive path consists of an upsampling step followed by a 2x2 convolution (up-conv), a concatenation(copy and crop) with feature map from the contracting path, and two 3x3 convolutions(also each followed by a ReLU). At the final layer a 1x1 convolution is used.

4. Results and Analysis

UNet with ResNet Encoder

1. Overall Explanation

: In this program, encoder part of UNet is replaced to ResNet-50, and train the model with Pascal VOC 2012 datasets and conduct the image segmentation.

2. Code

resnet\_encoder\_unet.py

import torchvision  
import torch.nn as nn  
import torch  
  
# resnet = torchvision.models.resnet.resnet50(pretrained=True)  
  
# 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  
  
###########################################################################  
# 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)  
  
 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)  
 def forward(self, x):  
 if self.downsample:  
 out = self.layer(x)  
 x = self.downsize(x)  
 return self.activation(out + x) # This part is slightly different from previous assignments of 'OSP-Lec14-CNN architecture-practice-v2.pdf'  
 else:  
 out = self.layer(x)  
 if x.size() is not out.size():  
 x = self.make\_equal\_channel(x)  
 return self.activation(out + x) # This part is slightly different from previous assignments of 'OSP-Lec14-CNN architecture-practice-v2.pdf'  
  
def conv(in\_channels, out\_channels):  
 return nn.Sequential(  
 nn.Conv2d(in\_channels, out\_channels, 3, padding=1), # 3: kernel size  
 nn.BatchNorm2d(out\_channels),  
 nn.ReLU(inplace=True), # When inplace = TRUE, ReLU modifies input activations, without allocating additional outputs. This often decrease the memory usage, but may sometimes cause some errors.  
 nn.Conv2d(out\_channels, out\_channels, 3, padding=1),  
 nn.BatchNorm2d(out\_channels),  
 nn.ReLU(inplace=True)  
 )  
  
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, 7, 2, 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, False),  
 ResidualBlock(256, 64, 256, False),  
 ResidualBlock(256, 64, 256, True) # Code overlaps with previous assignments  
 )  
 self.layer3 = nn.Sequential(  
 ResidualBlock(256, 128, 512, False),  
 ResidualBlock(512, 128, 512, False),  
 ResidualBlock(512, 128, 512, False),  
 ResidualBlock(512, 128, 512, True) # Code overlaps with previous assignments  
 )  
 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)  
  
 ###########################################################################  
 # 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)  
 x = self.UnetConv1(x)  
 x = self.upconv2\_1(x, output\_size=torch.Size([x.size(0),256,64,64]))  
 x = self.upconv2\_2(x)  
 #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)  
 x = torch.cat([x, out1], dim=1)  
 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

modules.py

import numpy as np  
import torch  
  
###########################################################################  
 # 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  
 outputs = model(inputs)  
 loss = criterion(outputs, labels)  
  
 optimizer.zero\_grad() # set gradients to zero  
 loss.backward() # conduct backpropagation  
 optimizer.step()  
  
 #########################################  
  
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  
  
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)  
  
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  
  
 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()  
  
 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 (both for image and predicted result)  
 ####### image should become temp\_rgb, result should become temp\_label  
 ####### You should use cls\_invert[]  
 temp\_rgb = cls\_invert[temp\_l[j]]  
 temp\_label = cls\_invert[temp\_l[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)

main.py

from datasets import Loader  
import torchvision.transforms as transforms  
import PIL.Image as PIL  
from modules import \*  
from torch.utils.data import DataLoader  
from torch.optim.lr\_scheduler import StepLR  
from resnet\_encoder\_unet import \*  
from UNet 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 = "./VOCdevkit"  
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))  
])  
  
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("trainLoader")  
trainLoader = DataLoader(trainset, batch\_size = batch\_size, shuffle=True)  
print("valLoader")  
validLoader = DataLoader(valset, batch\_size = batch\_size, shuffle=True)  
  
##### fill in here #####  
##### Hint : Initialize the model (Options : UNet, resnet\_encoder\_unet)  
# model = Unet()  
model = UNetWithResnet50Encoder()  
###############################################################################  
  
# 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)  
  
# parameters  
epochs = 40  
device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')  
model = model.to(device)  
  
##### fill in here ####s#  
##### Hint : load the model parameter, which is given  
# model.load\_state\_dict(torch.load('./trained\_model/UNet\_trained\_model.pth'))  
model.load\_state\_dict(torch.load('./trained\_model/resnet\_encoder\_unet.pth'))  
model.eval()  
  
# 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")  
  
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)  
 # torch.save(model.state\_dict(), savepath2)  
  
print('Finish Training')  
  
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")

3. Code Explanation

Encoder part of UNet is replaced to ResNet-50 with Layer3. ResNet-50 uses bottle neck building block. Each Residual block consists of 1x1, 3x3, 1x1 conv layers, and these residual blocks and other layers are repeated. Layer1 consists of 7x7x64 conv layer and 3x3 max pooling layer, Layer2 consists of 3 residual blocks, and Layer3 consists of 3 residual blocks. Decoder part(Expanding path) is similar to original UNet.

4. Results and Analysis