Encoder-Decoder

2317038

허지원

개요

Encoder-Decoder 는 VOC 2012 데이터 세트를 활용하여 U-Net 과 ResNet-Encoder-UNet 기반의 분류 모델을 학습 및 평가하는 프로그램이다.

- Original U-Net

: CNN 기반의 encoder-decoder 구조 모델로, 다음과 같은 단계로 진행된다.

① Encoder 에서 이미지를 다운 샘플링하여 특징을 추출한다
② Decoder 에서 업샘플링을 통해 원래 해상도로 복원한다.

각 encoder 단계의 피처 맵을 대응하는 decoder 단계와 skip connection 로 직접 연결하여, 공간적 위치 정보와 세밀한 특징을 효과적으로 복원하는 방식이다.

- U-Net with ResNet Encoder

: U-Net 의 인코더 부분을 ResNet 구조로 대체한 모델로, 다음과 같은 특징을 가진다.

① ResNet 인코더는 skip connection 을 통해 훨씬 더 깊고 강력한 특징 추출이 가능하다.
② 디코더는 U-Net 구조와 동일하게 업샘플링과 skip connection 을 활용하여 해상도를 복원한다.

코드 설명

1. main_skeleton.py

1) 필요한 라이브러리 및 모듈

```
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 *
# from UNet_skeleton import *
```

A. 데이터 세트 로딩, 이미지 변환, 모델 구조, 학습/평가 함수 등을 위한 라이브러리 및 모듈을 불러옴

2) 하이퍼파라미터 및 데이터 세트 경로 설정

```
# batch size
batch_size = 16
learning_rate = 0.005

# VOC2012 data directory
data_dir = "/Users/heojiwon/OSProject/Assignment11_2317038/skeleton-code-Lec15/VOC2012"
resize_size = 256
```

A. 하이퍼 파라미터
   i. batch_size: 한 번에 처리할 데이터 샘플 수
   ii. learning_rate: 학습률
B. 데이터 경로 및 이미지 크기 설정
   i. data_dir: VOC 2012 데이터 세트가 저장된 경로 지정
   ii. resize_size: 이미지 크기를 리사이즈

3) 데이터 전처리(Transform) 정의

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

A. 이미지를 PIL 이미지로 변환
B. resize_size 크기로 이미지를 리사이즈
C. 이미지를 tensor 로 변환
D. 이미지의 각 채널을 평균과 표준편차로 정규화

4) 데이터 세트 및 DataLoader 생성

```
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)
```

A. trainset, valset 을 각각 DataLoader 로 감싸서 배치 단위로 데이터를 불러올 수 있게 함

5) 모델, 손실함수, 옵티마이저, 스케줄러 준비

```python
model = UNetWithResnet50Encoder()

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

# Loss Function
##### fill in here -> hint : set the loss function #####
criterion = nn.CrossEntropyLoss(ignore_index=21)

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

# parameters
epochs = 3

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
scheduler = StepLR(optimizer, step_size=1, gamma=0.1)
```

A. 모델 초기화: 사용할 모델을 선택하여 인스턴스 생성
B. 손실함수 설정: CrossEntropyLoss 손실 함수를 설정
    A. ignore_index=21: 레이블이 21 인 픽셀은 손실 계산에서 제외
C. 옵티마이저 설정: Adam 옵티마이저를 사용하여 모델 파라미터를 업데이트
D. epoch 수 설정: 학습을 반복할 횟수 지정
E. 디바이스 설정 및 모델 이동: 사용 가능한 환경에 맞게 디바이스를 설정하고 모델을 이동
F. 학습률 스케줄러 설정: 일정 에폭마다 학습률을 gamma 비율로 줄임

6) 결과 저장 폴더 및 기록 변수 초기화

```python
# Train
import os
from datetime import datetime

now = datetime.now()
date = now.strftime('%Y-%m-%d(%H:%M)')
def createFolder(dir):  4개의 사용 위치
    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)
```

A. 필요한 모듈 불러오기: os, datetime 등 필요한 모듈을 임포트함
B. 디렉토리 생성 함수 정의: createFolder 함수를 정의하여 입력한 경로가 없으면 새로 만듦
C. 결과 저장 폴더 및 예측 결과 폴더 생성
    A. result_save_dir: 학습 결과를 저장할 폴더
    B. predict_save_dir: 예측 결과를 저장할 하위 폴더
D. 학습 이력 기록 용 딕셔너리 초기화: history 딕셔너리에 에폭별로 학습/검증 손실과 정확도를 저장
E. 학습 시작 메세지 출력
F. 모델 파라미터 저장 폴더 생성: 학습 중간마다 모델의 파라미터를 저장할 savepath1 폴더 생성

7) 학습 및 검증 루프 실행

```python
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')
```

A. epoch 반복 루프: 설정한 epoch 수 만큼 학습을 반복
B. 모델 학습: train_model 함수를 통해 한 epoch 동안 학습 데이터 세트를 이용해 모델을 학습
C. 학습 데이터 세트에서 성능 평가: 학습 데이터 세트 전체에 대해 모델의 손실과 정확도를 평가하고, 각 epoch 마다 결과 출력
D. 검증 결과 저장 폴더 생성: epoch 별로 예측 결과를 저장할 폴더 생성
E. 검증 데이터 세트에서 성능 평가: 검증 데이터 세트 전체에 대해 모델의 손실과 정확도를 평가하고, 예측 분할 결과를 이미지로 저장
F. 학습 이력 기록: epoch 별로 학습/검증 손시로가 정확도를 기록
G. 모델 파라미터 저장: 4 의 배수 epoch 마다 모델의 파라미터를 파일로 저장
H. 학습 종료

8) 학습 결과 시각화

```python
import matplotlib.pyplot as plt

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

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

plt.subplot( *args: 2,1,2)
plt.plot( *args: range(epoch+1), history['train_acc'], label='Accuracy', color='red')
plt.plot( *args: 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")
```

A. matplotlib 의 pyplot :데이터 시각화를 위한 라이브러리 불러오기
B. Loss 그래프 그리기
C. Accuracy 그래프 그리기

2. modules_skeleton.py

1) train_model 함수

```python
def train_model(trainloader, model, criterion, optimizer,scheduler, device):  1개의 사용 위치
    model.train()
    total_loss = 0
    recent_losses = []
    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()

        #Get the output out of model
        output = model(inputs)
        #Get the Loss
        loss = criterion(output, labels)

        #Optimizer and Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if scheduler:
            scheduler.step()
```

A. 모델 학습: train() 함수를 통해 모델을 학습모드로 설정
B. 손실 누적 변수 초기화: 전체 손실 합과 채근 배치들의 손실을 기록하기 위한 리스트를 초기화

C. 학습 데이터 배치 반복: trainloader 에서 데이터를 한 batch 씩 꺼내 학습을 반복
D. 데이터를 디바이스로 이동: 입력/정답 값을 디바이스로 이동
E. 순전파 및 손실 계산: 모델에 입력값을 넣어 예측값을 얻고, 예측값과 정답 레이블을 비교해 손실 계산
F. 역전파 및 파라미터 업데이트: 손실에 대한 모델 파라미터의 그래디언트를 계산하여 파라미터를 업데이트
G. 학습률 스케쥴러 적용: scheduler 에 맞게 학습률 조정

```
        total_loss += loss.item()
        recent_losses.append(loss.item())

        # 매 batch마다 loss 출력
        print(f"Iteration [{i+1}/{len(trainloader)}], Loss: {loss.item():.4f}")

        ########################################
        ############# fill in here (10 points) -> train
        ######## Hint :
        ######## 1. Get the output out of model, and Get the Loss
        ######## 3. optimizer
        ######## 4. backpropagation
        ########################################
```

H. 손실 기록 및 출력: 현재 batch 의 손실을 누적 및 기록하고, 매 batch 마다 손실을 출력

2) accuracy_check 함수

```
def accuracy_check(label, pred):  1개의 사용 위치
    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())
```

A. 파라미터
   i. label: 정답 레이블
   ii. pred: 모델의 예측 결과
B. label, pred 이 같은 위치에서 같은 값을 가지는지 비교(bool)
C. True 인 원소의 개수를 모두 더해, 맞춘 픽셀 수를 계산
D. 전체 픽셀 수로 나누어 정확도를 반환

3) accuracy_check_for_batch 함수

```python
def accuracy_check_for_batch(labels, preds, batch_size):  2개의 사용 위치
    total_acc = 0
    for i in range(batch_size):
        total_acc += accuracy_check(labels[i], preds[i])
    return total_acc/batch_size
```

A. 파라미터
   i. labels: 배치 단위의 정답 레이블
   ii. preds: 배치 단위의 예측 결과
   iii. batch_size: 배치에 포함된 이미지 개수
B. for 문을 통한 각 샘플의 정확도 계산: accuracy_check 함수를 호출하여 각 이미지의 픽셀 정확도 계산
C. 평균 정확도 반환

4) get_loss_train 함수

```python
def get_loss_train(model, trainloader, criterion, device):  1개의 사용 위치

    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()

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            ########################################
            ############ 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 = np.transpose(outputs.cpu(),  axes: (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)
```

A. 모델 평가 모드로 전환: eval() 함수를 통해 모델을 평가 모드로 전환
B. 정확도/손실 누적 변수 초기화: 전체 평균을 구하기 위해 누적할 변수 초기화
C. 배치 단위로 반복 평가: trainloader 에서 데이터를 배치 단위로 꺼내 반복하여 평가
D. 데이터를 디바이스로 이동
E. 순전파 및 손실 계산: 모델의 예측값을 얻고, 이를 정답 레이블과 비교
F. 예측 결과 및 정확도 계산: 가장 높은 클래스 인덱스를 예측값으로 선택한 후, accuracy_check_for_batch 함수를 통해 배치 평균 픽셀 정확도를 계산

G. 정확도/손실 누적: 각 배치의 정확도와 손실을 누적
H. 평균값 반환: 전체 배치의 평균 정확도와 평균 손실 반환

5) val_model 함수

```python
from PIL import Image
def val_model(model, valloader, criterion, device, dir):  1개의 사용 위치

    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
```

A. 클래스별 색상 맵 정의: 각 클래스(0~21)에 대해 RGB 색상을 지정
B. 평균 손실/정확도 누적 변수 초기화: 전체 평균을 구하기 위해 누적할 변수를 초기화

```python
    for batch, (inputs, labels) in enumerate(valloader):
        with torch.no_grad():

            inputs = inputs.to(device)
            labels = labels.to(device=device, dtype=torch.int64)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            ########################################
            ############ 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 = np.transpose(outputs.cpu(), axes: (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()
```

C. 배치 단위로 검증 데이터 평가: 검증 데이터 세트를 배치 단위로 반복
D. 입력/정답을 디바이스로 이동
E. 모델 예측 및 손실 계산
F. 예측 결과 후처리 및 정확도 계산

```
            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]):
                        pred_class = int(temp[j][k])
                        label_class = int(temp_l[j][k])
                        temp_rgb[j, k] = cls_invert[pred_class]
                        temp_label[j, k] = cls_invert[label_class]
                        ########################################
                        ############ 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[]
                        ########################################
```

```
                    img = inputs[i].cpu()
                    img = np.transpose(img, axes: (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)
```

G. 예측/정답 마스크를 RGB 이미지로 변환 및 저장
   i. 각 픽셀의 클래스 인덱스를 RGB 색상으로 변환하여 temp_rgb, temp_label 에 저장하고, PIL.Image 로 변환 후 파일로 저장
H. 평균 정확도와 손실 반환: 전체 배치의 평균 정확도와 평균 손실 반환

3. resnet_encoder_unet_skeleton.py

1) conv1x1, conv3x3 함수

```python
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):  6개의 사용 위치
    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):  2개의 사용 위치
    model = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=padding),
        nn.BatchNorm2d(out_channels)
    )
    return model
```

A. 파라미터
   i. in_channels. out_channels
   ii. stride, padding
B. 커널을 사용하는 합성곱 계층 생성
C. 배치 정규화와 ReLU 활성화 함수를 순차적으로 추가

2) ResidualBlock 클래스

```python
class ResidualBlock(nn.Module):  8개의 사용 위치
    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(
                conv1x1(in_channels, middle_channels, stride: 2, padding: 0),
                conv3x3(middle_channels, middle_channels, stride: 1, padding: 1),
                conv1x1(middle_channels, out_channels, stride: 1, padding: 0)
            )
            self.downsize = conv1x1(in_channels, out_channels, stride: 2, padding: 0)

        else:
            self.layer = nn.Sequential(
                conv1x1(in_channels, out_channels, stride: 1, padding: 0),
                conv3x3(out_channels, out_channels, stride: 1, padding: 1),
                conv1x1(out_channels, out_channels, stride: 1, padding: 0)
            )
            self.make_equal_channel = conv1x1(in_channels, out_channels, stride: 1, padding: 0)

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

A. 파라미터
   i. in_channels: 입력 feature map의 채널 수
   ii. middle_channels: 블록 내부에서 사용하는 중간 채널 수
   iii. out_channels: 블록의 출력 채널 수
   iv. downsample: 다운 샘플링 여부를 결정
      ▫ True: 입력 피처 맵의 공간 크기를 절반으로 줄이고 채널

수를 늘림
- False: 입력과 출력 공간의 크기를 유지

B. downsample = True일 때: 입력 피처 맵의 공간 해상도를 절반으로 줄이고, 채널 수를 늘려줌
   i. self.layer
      - 1x1conv: 채널 수를 줄이고 stride=2로 다운 샘플링
      - 3x3conv: 공간 크기를 유지하고 채널 확장(middle → out)
      - 1x1conv: 공간 크기는 유지하고 채널 수를 out_channels로 맞춤
   ii. downsize: Resid connection에서 입력 x의 해상도와 채널 수를 출력과 맞춤
C. downsample = False일 때의 내부 계층 구성
   i. self.layer
      - 1x1convL 채널 수를 out_channels로 맞춤
      - 3x3conv: 공간 정보 추출
      - 1x1conv: 채널 수 유지
   ii. self.make_equal_channel: 입력 x의 채널 수가 출력과 다를 때, Residual connection에서 채널 수를 맞춤
D. self.activation: 마지막에 ReLU 활성화 함수 적용
E. forward 함수

```
49      def forward(self, x):
50          if self.downsample:
51              out = self.layer(x)
52              x = self.downsize(x)
53              return self.activation(out + x)
54          else:
55              out = self.layer(x)
56              if x.size() is not out.size():
57                  x = self.make_equal_channel(x)
58              return self.activation(out + x)
```

   i. downsample 여부에 따라 residual connection 에서 다운샘플링 수행
   ii. out 결과와 입력 x 를 더한 값에 ReLU 를 적용

3) conv 함수

```
60  def conv(in_channels, out_channels):  2개의 사용 위치
61      return nn.Sequential(
62          nn.Conv2d(in_channels, out_channels, kernel_size: 3, padding=1),  # 3: kernel size
63          nn.BatchNorm2d(out_channels),
64          nn.ReLU(inplace=True),  # When inplace = TRUE, ReLU modifies input activations, witho
65          nn.Conv2d(out_channels, out_channels, kernel_size: 3, padding=1),
66          nn.BatchNorm2d(out_channels),
67          nn.ReLU(inplace=True)
68      )
```

A. 파라미터

i. in_channel, out_channel: 입력/출력 채널 수

B. nn.Sequential 를 통해 레이어를 순차적으로 묶어 하나의 블록 생성

i. Conv2d → BathchNorm2d → ReLU 를 두번 반복

- Conv2d: 입력 채널 → 출력 채널로 3x3 합성곱, padding=1 로 입력과 출력 공간의 크기를 유지
- BatchNorm2d: 첫 합성곱 결과를 정규화하여 학습을 안정화
- ReLU: 비선형성 부여

4) UNetWithResnet50Encoder 클래스

```
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 overlaps with previous assignments
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)#,
        )
        self.pool = nn.MaxPool2d( kernel_size: 3,  stride: 2,  padding: 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) # Code overlaps with previous assignments
        )
        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: False)# Code overlaps with previous assignments
        )
```

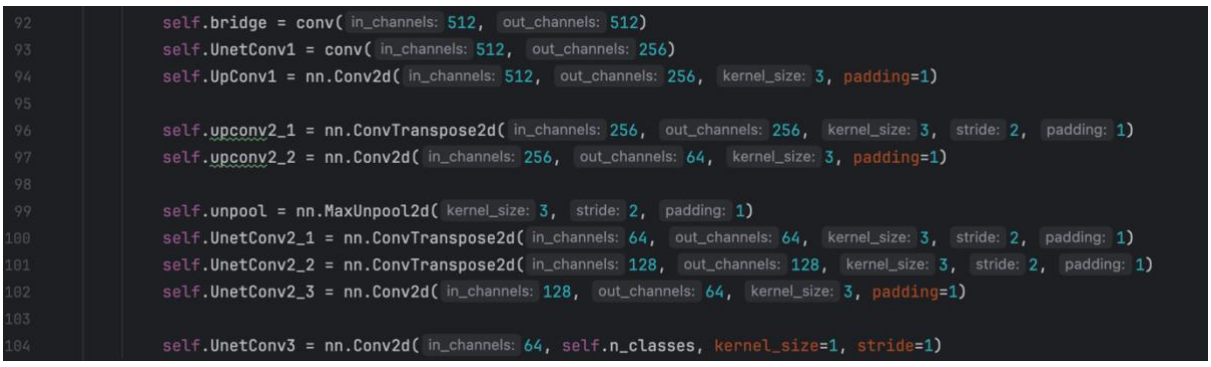
A. 인코더(Encoder)

i. layer1: 7x7 Conv2d + BatchNorm + ReLU → 입력 이미지를 다운샘플링하여 특징 추출

ii. pool: 3x3 Maxpooling → 공간 크기를 절반으로 줄임

iii. layer2: ResidualBlock 3 개 → ResNet 으로 더 깊은 특징 추출

iv. layer3: ResidualBlock 4 개로 더 깊은 특징 추출, 채널 수 증가



```
        self.bridge = conv( in_channels: 512,  out_channels: 512)
        self.UnetConv1 = conv( in_channels: 512,  out_channels: 256)
        self.UpConv1 = nn.Conv2d( in_channels: 512,  out_channels: 256,  kernel_size: 3, padding=1)

        self.upconv2_1 = nn.ConvTranspose2d( in_channels: 256,  out_channels: 256,  kernel_size: 3,  stride: 2,  padding: 1)
        self.upconv2_2 = nn.Conv2d( in_channels: 256,  out_channels: 64,  kernel_size: 3, padding=1)

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

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

B. 브릿지(Bridge): 인코더와 디코더를 연결하는 중간 블록

C. 디코더(Decoder)

i. UnetConv1: conv 함수를 통해 채널 수를 줄이고 skip connection 을 통해 합쳐진 특징을 추출

ii. UpConv1: 채널 수를 줄임
iii. upconv2_1: ConvTranspose2d 를 통해 공간 해상도를 2 배로 늘리면서 채널 수는 유지
iv. upconv2_2: Conv2d 를 통해 채널수를 줄임
v. unpool: Maxunpooling 하여 공간 해상도를 복원
vi. UnetConv2_1: ConvTranspose2d 를 통해 특징 맵을 업샘플링
vii. UnetConv2_2: skip connection 후 채널 수를 유지하며 공간 해상도를 늘림
viii. UnetConv2_3: 채널 수 줄임
ix. UnetConv3: 픽셀별로 클래스 점수 출력

5) forward 함수

```
109     def forward(self, x, with_output_feature_map=False): #256
110
111         out1 = self.layer1(x)
112         out1, indices = self.pool(out1)
113         out2 = self.layer2(out1)
114         out3 = self.layer3(out2)
115         x = self.bridge(out3) # bridge
116         x = self.UpConv1(x)
117         x = torch.cat( tensors: (x, out2), dim=1)
118         x = self.UnetConv1(x)
119         x = self.upconv2_1(x, output_size=torch.Size([x.size(0),256,64,64]))
120         x = self.upconv2_2(x)
121         x = torch.cat( tensors: (x, out1), dim=1)
122         x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 128, 128, 128]))
123         x = self.UnetConv2_2(x, output_size=torch.Size([x.size(0), 128, 256, 256]))
124         x = self.UnetConv2_3(x)
125         x = self.UnetConv3(x)
126         return x
```

A. 초기 특징 추출 및 다운 샘플링(out1, indices)
B. ResNet 인코더(out2, out3): 더 깊은 특징 추출
C. Bridge: 인코더와 디코더 연결
D. 디코더: 업샘플링과 Skip connection 을 반복
E. 출력: 1x1 Conv2d 로 채널 수를 클래스 개수(n_classes)로 맞추고, segmentation map 을 반환

4. UNet_skeleton.py

1) conv 함수

```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):  9개의 사용 위치
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size: 3, padding=1),  # 3은 kernel size
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, kernel_size: 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
```

A. 파라미터
   i. in_channels, out_channels: 입력/출력 채널 수
B. nn.Sequential 을 통한 블록 생성
   i. Conv2d → BatchNorm2d → ReLU 두 차례 반복

2) Unet 클래스

```python
class Unet(nn.Module):  2개의 사용 위치
    def __init__(self, in_channels=3, out_channels=22):
        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, out_channels: 64)
        self.convDown2 = conv( in_channels: 64, out_channels: 128)
        self.convDown3 = conv( in_channels: 128, out_channels: 256)
        self.convDown4 = conv( in_channels: 256, out_channels: 512)
        self.convDown5 = conv( in_channels: 512, out_channels: 1024)
        self.maxpool = nn.MaxPool2d( kernel_size: 2, stride=2)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.convUp4 = conv(1024+512, out_channels: 512)
        self.convUp3 = conv(512+256, out_channels: 256)
        self.convUp2 = conv(256+128, out_channels: 128)
        self.convUp1 = conv(128+64, out_channels: 64)
        self.convUp_fin = nn.Conv2d( in_channels: 64, out_channels, kernel_size: 1)
```

A. 파라미터
   i. in_channels, out_channels: 입력/출력 채널 수
B. 인코더 (Downsampling)
   i. 각 단계마다 conv 블록을 거치며 채널 수를 증가시킴
   ii. 이미지는 각 단계마다 maxpool 을 통해 이미지의 공간 크기를 절반으로 줄임
C. max pooling: 2x2 영역에서 최대값을 뽑아내며, 이미지 크기를 절반으로 줄임
D. upsampling
E. 디코더 (Upsampling)

i. 각 단계마다 업샘플링된 피처 맵과 인코더에서 건너온(skip connection) 특징 맵을 채널 방향으로 합친 뒤, 두 번의 3x3 Conv + BatchNorm + ReLU 로 처리

F. 최종 출력 레이어: 마지막 1x1 Conv 로 64 채널을 분할 클래스 개수(out_channels)로 변환

G. forward 함수

```
    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( tensors: (x, conv4), dim=1)
        x = self.convUp4(x)
        x = self.upsample(x)
        #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)
        x = torch.cat( tensors: (x, conv3), dim=1)
        x = self.convUp3(x)
        x = self.upsample(x)
        #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)
        x = torch.cat( tensors: (x, conv2), dim=1)
        x = self.convUp2(x)
        x = self.upsample(x)
        #######fill in here ####### hint : concatenation (Practice Lecture slides 6p)
        x = torch.cat( tensors: (x, conv1), dim=1)
        x = self.convUp1(x)
        out = self.convUp_fin(x)

        return out
```

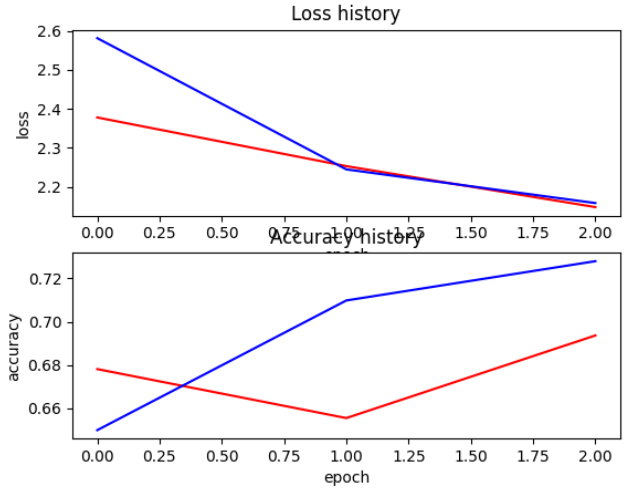
i. ㅇㅇ

실행 결과

① ResNet_Encoder_UNet의 결과

- epoch=3, 이미지 50개 활용(database 코드 수정)

```
        if self.flag == 'train':
            # self.imgnames = self.lines[:50] # Tip
            self.imgnames = self.lines[self.fold:]
            self.imgnames = self.imgnames[:50]

        else:
            # self.imgnames = self.lines[50:60] # Ti
            self.imgnames = self.lines[:self.fold]
            self.imgnames = self.imgnames[:50]
```



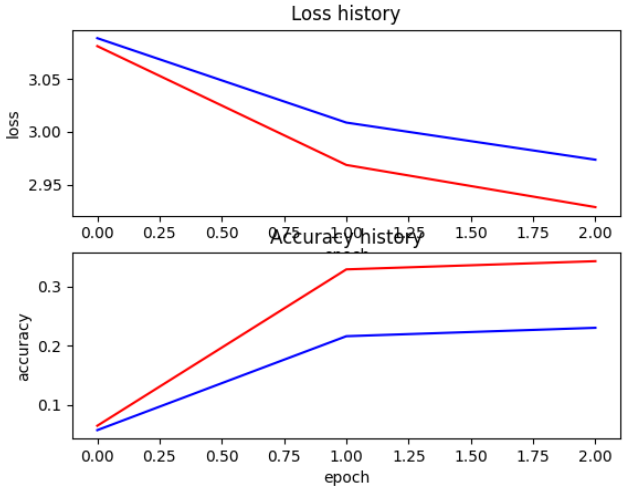

```
trainset
valset
tainLoader
valLoader
Training
Iteration [1/4], Loss: 3.1301
Iteration [2/4], Loss: 2.8051
Iteration [3/4], Loss: 1.9830
Iteration [4/4], Loss: 2.1329
epoch 1 train loss :  2.37767893075943 train acc :  0.6781454086303711
epoch 1 val loss :  2.5810789465904236 val acc :  0.6500198841094971
Iteration [1/4], Loss: 1.9666
Iteration [2/4], Loss: 2.0784
Iteration [3/4], Loss: 1.9461
Iteration [4/4], Loss: 2.1090
epoch 2 train loss :  2.2528865337371826 train acc :  0.655583381652832
epoch 2 val loss :  2.2445003986358643 val acc :  0.709770917892456
Iteration [1/4], Loss: 2.0471
Iteration [2/4], Loss: 2.0945
Iteration [3/4], Loss: 1.8517
Iteration [4/4], Loss: 1.7009
epoch 3 train loss :  2.1477534472942352 train acc :  0.6936483383178711
epoch 3 val loss :  2.158144325017929 val acc :  0.72786545753479
Finish Training
Fin

종료 코드 0(으)로 완료된 프로세스
```

➔ train loss: [2.38, 2.25, 2.14], val loss: [2.58, 2.24, 2.16]으로, epoch이 진행될수록 손실이 감소한다.

train acc: [0.68, 0.66, 0.69], val acc: [0.65, 0.71, 0.73]이다. 학습 정확도의 경우 꾸준히 오르지는 않았지만 검증 정확도의 경우 꾸준히 상승하는 추세이다.

② UNet 결과

- epoch=3, batch_size = 16, learning_rate = 0.005, 이미지 50개 활용 (database 코드 수정)




```
trainset
valset
tainLoader
valLoader
Training
Iteration [1/4], Loss: 3.1746
Iteration [2/4], Loss: 2.9829
Iteration [3/4], Loss: 2.8987
Iteration [4/4], Loss: 2.8044
epoch 1 train loss :  3.0809731483459473 train acc :  0.06494688987731934
epoch 1 val loss :  3.0884820222854614 val acc :  0.05755209922790527
Iteration [1/4], Loss: 2.7677
Iteration [2/4], Loss: 2.9038
Iteration [3/4], Loss: 2.8005
Iteration [4/4], Loss: 2.9651
epoch 2 train loss :  2.968714416027069 train acc :  0.32915687561035156
epoch 2 val loss :  3.00874000787735 val acc :  0.21624016761779785
Iteration [1/4], Loss: 2.8896
Iteration [2/4], Loss: 2.8215
Iteration [3/4], Loss: 2.7339
Iteration [4/4], Loss: 2.9222
epoch 3 train loss :  2.9288838505744934 train acc :  0.3429830074310303
epoch 3 val loss :  2.9736807346343994 val acc :  0.23043203353881836
Finish Training
Fin

종료 코드 0(으)로 완료된 프로세스
```

➔ train loss: [3.08, 2.97, 2.93], val loss: [3.09, 3.01, 2.97]로, epoch이 진행될수록 손실이 꾸준히 감소한다.

train acc: [0.065, 0.329, 0.343], val acc: [0.058, 0.216, 0.230]으로, 정확도 역시 epoch이 진행될수록 상승하는 추세이다.

➢ ResNet-Encoder-UNet과 UNet의 성능 비교

손실(loss): 두 모델 모두 손실이 감소하는 경향을 보이지만, ResNet-Encoder-UNet의 손실이 항상 더 낮다.

정확도(accuracy): 두 모델 모두 정확도가 증가하지만, ResNet-

Encoder_UNet이 2배 이상 높은 정확도를 보인다.

➔ 이를 통해 ReNet-Encoder-UNet이 UNet에 비해 성능이 좋다는 것을 알 수 있다. 이는 ResNet 기반 인코더가 더 깊고 강력한 특징 추출 능력을 갖추고 있기 때문이다. VOC2012와 같은 복잡한 데이터세트에서는 ResNet-Encoder-UNet이 훨씬 더 효과적이다.