PA3

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:

```
import os
import sys
os.chdir('/content/drive/MyDrive/3_face_segmentation')
print(os.getcwd())
```

In [ ]:

```
import torch.nn as nn
from model import SegNet
from PIL import Image
import torchvision
import tqdm
from utils import *
import cv2
from torchvision.utils import save_image
import torch.nn.functional as F
```

In [ ]:

```
class Dataset(object):
    def __init__(self, img_path, label_path, method='train'):
        self.img_path =  img_path
        self.label_path = label_path
        self.train_dataset = []
        self.test_dataset = []
        self.mode = method == 'train'
        self.preprocess()
        if self.mode:
            self.num_images = len(self.train_dataset)
        else:
            self.num_images = len(self.test_dataset)

    def preprocess(self):
        if self.mode:
            len = 4500
        else:
            len = 500
        for i in range(len):
            # 4500 넘어까지지 되는 문제로 수정
            # len([name for name in os.listdir(self.img_path) if os.path.isfile(os.path.j
oin(self.img_path, name))])):
            img_path = os.path.join(self.img_path, str(i) + '.jpg')
            label_path = os.path.join(self.label_path, str(i) + '.png')
            if self.mode == True:
                self.train_dataset.append([img_path, label_path])
            else:
                self.test_dataset.append([img_path, label_path])
        print('Finished preprocessing the CelebA dataset...')

    def __getitem__(self, index):
        dataset = self.train_dataset if self.mode == True else self.test_dataset
        img_path, label_path = dataset[index]
        image = Image.open(img_path)
        label = Image.open(label_path)
        transform = torchvision.transforms.Compose(
            [torchvision.transforms.ToTensor(), torchvision.transforms.Resize((512, 512)
)])
```

```
        return transform(image), transform(label), img_path.split("/")[-1]

    def __len__(self):
        """Return the number of images."""
        return self.num_images
```

In [ ]:

```
class Tester(object):
    def __init__(self, batch_size, epochs, lr):
        self.batch_size = batch_size
        self.epochs = epochs
        self.learning_rate = lr
        self.model = self.build_model()
        # Load of pretrained_weight file
        weight_PATH = "finetuned_{}_{}_{}_softmax_onehot.pth".format(self.epochs, self.batch_size, self.learning_rate)
        # weight_PATH = 'pretrained_weight.pth'
        self.model.load_state_dict(torch.load(weight_PATH))
        dataset = Dataset(img_path="data/test_img", label_path="data/test_label", method='test')
        self.dataloader = torch.utils.data.DataLoader(dataset=dataset,
                                                      batch_size=self.batch_size,
                                                      shuffle=True,
                                                      num_workers=2,
                                                      drop_last=False)
        self.criterion = nn.CrossEntropyLoss()
        print("Testing...")

    def test(self):
        make_folder("test_mask_{}_{}_{}_softmax_onehot".format(self.epochs, self.batch_size, self.learning_rate), '')
        make_folder("test_color_mask_{}_{}_{}_softmax_onehot".format(self.epochs, self.batch_size, self.learning_rate), '')
        self.model.eval()
        self.test_loss = 0
        for i, data in enumerate(self.dataloader):
            imgs = data[0].cuda()
            target = data[1].cuda()
            labels_predict = self.model(imgs)
            labels_predict_plain = generate_label_plain(labels_predict, 512)
            labels_predict_color = generate_label(labels_predict, 512)
            batch_size = labels_predict.size()[0]

            labels_predict_SM = torch.nn.functional.softmax(labels_predict, dim=1)

            # Generat GT
            # one-hot
            hair = target > 0.005 # 0.007843137718737125 -> 2 (hair)
            face = target > 0.003
            face2 = target < 0.005
            face = face * face2 # 0.003921568859368563 -> 1 (face)
            back = target == 0 # 0 -> 0 (bg)
            gt = torch.concat([back, face, hair], dim = 1).float()


            # index1
            # hair = target > 0.005 # 0.007843137718737125 -> 2 (hair)
            # face = target > 0.003
            # gt = hair.long() + face.long()

            # index2
            # gt = target * 255

            # gt = gt.type(torch.LongTensor).cuda()
            # gt = gt.squeeze()

            loss = self.criterion(labels_predict_SM, gt)
            self.test_loss += loss.item()

            for k in range(batch_size):
```

```
                cv2.imwrite(os.path.join("test_mask_{}_{}_{}_softmax_onehot".format(self
.epochs, self.batch_size, self.learning_rate), data[2][k]), labels_predict_plain[k])
                save_image(labels_predict_color[k], os.path.join("test_color_mask_{}_{}_
{}_softmax_onehot".format(self.epochs, self.batch_size, self.learning_rate), data[2][k]))
        print(f"{len(self.dataloader)}")
        epoch_loss = self.test_loss / len(self.dataloader)

        print(f"epoch_loss is {epoch_loss}")

    def build_model(self):
        model = SegNet(3).cuda()
        return model
```

In [ ]:

```
class Trainer(object):
    def __init__(self, epochs, batch_size, lr):
        self.epochs = epochs
        self.batch_size = batch_size
        self.learning_rate = lr
        self.model = self.build_model()
        self.optimizer = torch.optim.Adam(self.model.parameters(), self.learning_rate)

        dataset = Dataset(img_path="data/train_img", label_path="data/train_label", meth
od='train')
        self.dataloader = torch.utils.data.DataLoader(dataset=dataset,
                                                      batch_size=self.batch_size,
                                                      shuffle=True,
                                                      num_workers=2,
                                                      drop_last=False)
        self.criterion = nn.CrossEntropyLoss()

    def train(self):
        for epoch in tqdm.tqdm(range(self.epochs + 1)):
            epochLoss = 0
            self.model.train()
            self.train_loss = 0
            for batch_idx, data in enumerate(self.dataloader):
                imgs = torch.autograd.Variable(data[0]).cuda()
                target = torch.autograd.Variable(data[1]).cuda()

                labels_predict = self.model(imgs)
                labels_predict = torch.nn.functional.softmax(labels_predict, dim=1)

                # Generat GT
                # one-hot
                hair = target > 0.005 # 0.007843137718737125 -> 2 (hair)
                face = target > 0.003
                face2 = target < 0.005
                face = face * face2 # 0.003921568859368563 -> 1 (face)
                back = target == 0 # 0 -> 0 (bg)
                gt = torch.concat([back, face, hair], dim = 1).float()

                # index1
                # hair = target > 0.005 # 0.007843137718737125 -> 2 (hair)
                # face = target > 0.003
                # gt = hair.long() + face.long()

                # index2
                # gt = target * 255

                # gt = gt.type(torch.LongTensor).cuda()
                # gt = gt.squeeze()

                self.optimizer.zero_grad()
                loss = self.criterion(labels_predict, gt)
                self.train_loss += loss.item()

                loss.backward()
                self.optimizer.step()
```

```
                print(f"Epoch {epoch}/{self.epochs} - Batch {batch_idx}/{len(self.datalo
ader)}")

            epoch_loss = self.train_loss / len(self.dataloader)

            print(f"Epoch {epoch}/{self.epochs}, loss is {epoch_loss}")

        train_path = "finetuned_{}_{}_{}_softmax_onehot.pth".format(self.epochs, self.ba
tch_size, self.learning_rate)
        torch.save(self.model.state_dict(), train_path)
        print('Finish training.')



    def build_model(self):
        model = SegNet(3).cuda()
        return model
```

In [ ]:

```
epochs = 10
lr = 0.01
batch_size = 32
trainer = Trainer(epochs, batch_size, lr)
trainer.train()
tester = Tester(batch_size, epochs, lr)
tester.test()
```