```python
# coding=utf-8
import torch
import torch.nn as nn
from torchvision import transforms
import torch.nn.functional as F
import os
import numpy as np
from PIL import Image
import argparse
import pickle
from datasets import Folder
from models import EncDec, FCN, DeepLab
from evaluate import fm_and_mae
from datasets.build_vocab import Vocabulary

from tqdm import tqdm


home = os.path.expanduser("~")

parser = argparse.ArgumentParser()
parser.add_argument('--img_dir', default='%s/data/datasets/saliency_Dataset/ECSSD/images' % (home))  # training dataset
parser.add_argument('--gt_dir', default='%s/data/datasets/saliency_Dataset/ECSSD/masks' % (home))  # training dataset
parser.add_argument('--result_dir', default='./results')  # training dataset
parser.add_argument('--batchSize', type=int, default=24)  # batch size
opt = parser.parse_args()
print(opt)


def make_dir(dir):
    if not os.path.exists(dir):
        os.makedirs(dir)


def main():
    img_size = 256
    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]
    make_dir(opt.result_dir)

    # data
    # Load vocabulary wrapper
    with open('vocab.pkl', 'rb') as f:
        vocab = pickle.load(f)
    loader = torch.utils.data.DataLoader(
        Folder(img_dir=opt.img_dir, gt_dir=opt.gt_dir,
source_transform=transforms.Compose([transforms.Resize((img_size, img_size))]),
target_transform=transforms.Compose([transforms.Resize((img_size, img_size))]),
               mean=mean, std=std),
        batch_size=opt.batchSize, shuffle=False, num_workers=4, pin_memory=True)
    # caption and classification networks
    cls_net = FCN(base='densenet169')
    cls_net = cls_net.cuda()
    cap_net = EncDec(len(vocab))
    cap_net = cap_net.cuda()
    # saliency network
    sal_net = DeepLab(base='densenet169', c_output=1)
    sal_net = nn.DataParallel(sal_net).cuda()
    # the 1st, 2nd and 3rd rows of Table 1
    cls_net.load_state_dict(torch.load('net-cls-init.pth'))
    cap_net.load_state_dict(torch.load('net-cap-init.pth'))
    output_dir = '/'.join([opt.result_dir, 'init', 'cls'])
    make_dir(output_dir)
    validate_one(loader, cls_net, output_dir)
    fm, mae, _,_ = fm_and_mae(output_dir, opt.gt_dir)
    print('cls fm %.3f'%fm)
    # the 2nd row of Table 1
    output_dir = '/'.join([opt.result_dir, 'init', 'cap'])
    make_dir(output_dir)
    validate_one(loader, cap_net, output_dir)
    fm, mae, _,_ = fm_and_mae(output_dir, opt.gt_dir)
    print('cap fm %.3f'%fm)
    # the 3rd row of Table 1
    output_dir = '/'.join([opt.result_dir, 'init', 'avg'])
    make_dir(output_dir)
    validate_two(loader, cls_net, cap_net, output_dir)
    fm, mae, _,_ = fm_and_mae(output_dir, opt.gt_dir)
    print('cls cap fm %.3f'%fm)
    # the 4th row of Table 1
    cls_net.load_state_dict(torch.load('cls-two-woun.pth'))
    cap_net.load_state_dict(torch.load('cap-two-woun.pth'))
    output_dir = '/'.join([opt.result_dir, 'at', 'avg'])
    make_dir(output_dir)
    validate_two(loader, cls_net, cap_net, output_dir)
    fm, mae, _,_ = fm_and_mae(output_dir, opt.gt_dir)
    print('cls cap at fm %.3f'%fm)
    # the 5th row of Table 1
    cls_net.load_state_dict(torch.load('cls-two-mr.pth'))
    cap_net.load_state_dict(torch.load('cap-two-mr.pth'))
    output_dir = '/'.join([opt.result_dir, 'ac', 'avg'])
    make_dir(output_dir)
    validate_two(loader, cls_net, cap_net, output_dir)
    fm, mae, _,_ = fm_and_mae(output_dir, opt.gt_dir)
    print('cls cap at ac fm %.3f'%fm)
    # the 6th row of Table 1
    sal_net.load_state_dict(torch.load('sal.pth'))
    output_dir = '/'.join([opt.result_dir, 'sal'])
    make_dir(output_dir)
    validate_one(loader, sal_net, output_dir)
    fm, mae, _,_ = fm_and_mae(output_dir, opt.gt_dir)
    print('sal fm %.3f'%fm)


def validate_two(loader, net_cls, net_cap, output_dir):
    if not os.path.exists(output_dir):
        os.mkdir(output_dir)
    net_cls.eval()
    net_cap.eval()
    loader = tqdm(loader, desc='validating')
    for ib, (data, lbl, img_name, w, h) in enumerate(loader):
```

```python
        with torch.no_grad():
            outputs_cls, _, _ = net_cls(data.cuda())
            outputs_cap = net_cap(data.cuda())
        outputs = (F.sigmoid(outputs_cls.cpu()) + F.sigmoid(outputs_cap.cpu()))/2
        outputs = outputs.squeeze(1).cpu().numpy()
        outputs *= 255
        for ii, msk in enumerate(outputs):
            msk = Image.fromarray(msk.astype(np.uint8))
            msk = msk.resize((w[ii], h[ii]))
            msk.save('{}/{}.png'.format(output_dir, img_name[ii]), 'PNG')
    net_cls.train()
    net_cap.train()


def validate_one(loader, net, output_dir):
    net.eval()
    loader = tqdm(loader, desc='validating')
    for ib, (data, lbl, img_name, w, h) in enumerate(loader):
        with torch.no_grad():
            outputs = net(data.cuda())
        if isinstance(outputs, tuple):
            outputs = outputs[0]
        outputs = F.sigmoid(outputs.cpu())
        outputs = outputs.squeeze(1).cpu().numpy()
        outputs *= 255
        for ii, msk in enumerate(outputs):
            msk = Image.fromarray(msk.astype(np.uint8))
            msk = msk.resize((w[ii], h[ii]))
            msk.save('{}/{}.png'.format(output_dir, img_name[ii]), 'PNG')
    net.train()


if __name__ == "__main__":
    main()
```