

```
1 # coding=utf-8
2 import torch
3 import torch.nn as nn
4 from torchvision import transforms
5 import torch.nn.functional as F
6 import os
7 import numpy as np
8 from PIL import Image
9 import argparse
10 import pickle
11 from datasets import Folder
12 from models import EncDec, FCN, DeepLab
13 from evaluate import fm_and_mae
14 from datasets.build_vocab import Vocabulary
15
16 from tqdm import tqdm
17
18
19 home = os.path.expanduser("~")
20
21 parser = argparse.ArgumentParser()
22 parser.add_argument('--img_dir',
23                     default='%s/data/datasets/saliency_Dataset/ECSSD/images' % (home)) #
24                     training dataset
25 parser.add_argument('--gt_dir',
26                     default='%s/data/datasets/saliency_Dataset/ECSSD/masks' % (home)) #
27                     training dataset
28 parser.add_argument('--result_dir', default='./results') # training
29                     dataset
30 parser.add_argument('--batchSize', type=int, default=24) # batch size
31 opt = parser.parse_args()
32 print(opt)
33
34
35 def make_dir(dir):
36     if not os.path.exists(dir):
37         os.makedirs(dir)
38
39
40 def main():
41     img_size = 256
42     mean = [0.485, 0.456, 0.406]
43     std = [0.229, 0.224, 0.225]
44     make_dir(opt.result_dir)
45
46     # data
47     # Load vocabulary wrapper
48     with open('vocab.pkl', 'rb') as f:
49         vocab = pickle.load(f)
50     loader = torch.utils.data.DataLoader(
51         Folder(img_dir=opt.img_dir, gt_dir=opt.gt_dir,
52               source_transform=transforms.Compose([transforms.Resize((img_size,
53                               img_size))]),
54               target_transform=transforms.Compose([transforms.Resize((img_size,
55                               img_size))]),
56               mean=mean, std=std),
57         batch_size=opt.batchSize, shuffle=False, num_workers=4,
58         pin_memory=True)
```

```

51     # caption and classification networks
52     cls_net = FCN(base='densenet169')
53     cls_net = cls_net.cuda()
54     cap_net = EncDec(len(vocab))
55     cap_net = cap_net.cuda()
56     # saliency network
57     sal_net = DeepLab(base='densenet169', c_output=1)
58     sal_net = nn.DataParallel(sal_net).cuda()
59     # the 1st, 2nd and 3rd rows of Table 1
60     cls_net.load_state_dict(torch.load('net-cls-init.pth'))
61     cap_net.load_state_dict(torch.load('net-cap-init.pth'))
62     output_dir = '/'.join([opt.result_dir, 'init', 'cls'])
63     make_dir(output_dir)
64     validate_one(loader, cls_net, output_dir)
65     fm, mae, __ = fm_and_mae(output_dir, opt.gt_dir)
66     print('cls fm %.3f'%fm)
67     # the 2nd row of Table 1
68     output_dir = '/'.join([opt.result_dir, 'init', 'cap'])
69     make_dir(output_dir)
70     validate_one(loader, cap_net, output_dir)
71     fm, mae, __ = fm_and_mae(output_dir, opt.gt_dir)
72     print('cap fm %.3f'%fm)
73     # the 3rd row of Table 1
74     output_dir = '/'.join([opt.result_dir, 'init', 'avg'])
75     make_dir(output_dir)
76     validate_two(loader, cls_net, cap_net, output_dir)
77     fm, mae, __ = fm_and_mae(output_dir, opt.gt_dir)
78     print('cls cap fm %.3f'%fm)
79     # the 4th row of Table 1
80     cls_net.load_state_dict(torch.load('cls-two-woun.pth'))
81     cap_net.load_state_dict(torch.load('cap-two-woun.pth'))
82     output_dir = '/'.join([opt.result_dir, 'at', 'avg'])
83     make_dir(output_dir)
84     validate_two(loader, cls_net, cap_net, output_dir)
85     fm, mae, __ = fm_and_mae(output_dir, opt.gt_dir)
86     print('cls cap at fm %.3f'%fm)
87     # the 5th row of Table 1
88     cls_net.load_state_dict(torch.load('cls-two-mr.pth'))
89     cap_net.load_state_dict(torch.load('cap-two-mr.pth'))
90     output_dir = '/'.join([opt.result_dir, 'ac', 'avg'])
91     make_dir(output_dir)
92     validate_two(loader, cls_net, cap_net, output_dir)
93     fm, mae, __ = fm_and_mae(output_dir, opt.gt_dir)
94     print('cls cap at ac fm %.3f'%fm)
95     # the 6th row of Table 1
96     sal_net.load_state_dict(torch.load('sal.pth'))
97     output_dir = '/'.join([opt.result_dir, 'sal'])
98     make_dir(output_dir)
99     validate_one(loader, sal_net, output_dir)
100    fm, mae, __ = fm_and_mae(output_dir, opt.gt_dir)
101    print('sal fm %.3f'%fm)
102
103
104    def validate_two(loader, net_cls, net_cap, output_dir):
105        if not os.path.exists(output_dir):
106            os.mkdir(output_dir)
107        net_cls.eval()
108        net_cap.eval()
109        loader = tqdm(loader, desc='validating')
110        for ib, (data, lbl, img_name, w, h) in enumerate(loader):

```

```
111         with torch.no_grad():
112             outputs_cls, _, _ = net_cls(data.cuda())
113             outputs_cap = net_cap(data.cuda())
114             outputs = (F.sigmoid(outputs_cls.cpu()) +
115 F.sigmoid(outputs_cap.cpu()))/2
116             outputs = outputs.squeeze(1).cpu().numpy()
117             outputs *= 255
118             for ii, msk in enumerate(outputs):
119                 msk = Image.fromarray(msk.astype(np.uint8))
120                 msk = msk.resize((w[ii], h[ii]))
121                 msk.save('{}{}/{}.png'.format(output_dir, img_name[ii]), 'PNG')
122         net_cls.train()
123         net_cap.train()
124
125 def validate_one(loader, net, output_dir):
126     net.eval()
127     loader = tqdm(loader, desc='validating')
128     for ib, (data, lbl, img_name, w, h) in enumerate(loader):
129         with torch.no_grad():
130             outputs = net(data.cuda())
131             if isinstance(outputs, tuple):
132                 outputs = outputs[0]
133             outputs = F.sigmoid(outputs.cpu())
134             outputs = outputs.squeeze(1).cpu().numpy()
135             outputs *= 255
136             for ii, msk in enumerate(outputs):
137                 msk = Image.fromarray(msk.astype(np.uint8))
138                 msk = msk.resize((w[ii], h[ii]))
139                 msk.save('{}{}/{}.png'.format(output_dir, img_name[ii]), 'PNG')
140         net.train()
141
142
143 if __name__ == "__main__":
144     main()
```