

代码阅读笔记

reproduce_paper_results.py

- train_network
 1. get_model_train_cfg: 使用对应模型, 开始训练
 2. train_eval_pcnet:
 3. train_eval_compennet_pp:
- projector_based_attack
 1. summarize_all_attackers:
- utils
 1. print_sys_info:
 2. set_torch_reproducibility:

train_network.py

get_model_train_cfg:

1. 首先设置训练网络的参数:

```
# default training configs
cfg_default = DictConfig({})
cfg_default.data_root = data_root
cfg_default.setup_list = setup_list
cfg_default.device = 'cuda'
cfg_default.device_ids = device_ids
cfg_default.load_pretrained = load_pretrained
cfg_default.max_iters = 2000
cfg_default.batch_size = 24
cfg_default.lr = 1e-3
cfg_default.lr_drop_ratio = 0.2
cfg_default.lr_drop_rate = 800 #
cfg_default.l2_reg = 1e-4
cfg_default.train_plot_rate = 50
cfg_default.valid_rate = 200 #
cfg_default.plot_on = plot_on #
cfg_default.center_crop = center_crop #

if single: # single model, no
    cfg_default.model_name = model_list[0]
    cfg_default.num_train = 500/
    cfg_default.loss = 'l1+ssim'
else:
    cfg_default.model_list = model_list
    cfg_default.num_train_list = [500]
    cfg_default.loss_list = ['l1+ssim']
```

train_eval_pcnet:

1. torch.device: 包含一个设备类型（'cpu'或'cuda'）和可选的设备序号。例: device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
2. os.path.join(): 将多个路径组合后返回
3. 加载训练和验证数据:

```
# train and evaluate all setups
for setup_name in cfg_default.setup_list:
    # load training and validation data
    cam_scene, cam_train, cam_valid, prj_train, prj_valid, cam_mask, mask_corners, setup_info = load_data(data_root, setup_name)
    cfg_default.setup_info = setup_info
```

4. 是否居中剪裁:

```
# center crop, decide whether PCNet output is center cropped square image (classifier_crop_sz) or not (cam_im_sz)
if cfg_default.center_crop:
    cp_sz = setup_info.classifier_crop_sz
    cam_scene = cc(cam_scene, cp_sz)
    cam_train = cc(cam_train, cp_sz)
    cam_valid = cc(cam_valid, cp_sz)
    cam_mask = cc(cam_mask, cp_sz)
```

5. ShadingNetSPAA model

- 详见models.ShadingNetSPAA

1. WarpingNet model

- 详见models.WarpingNet

7. PCNet model

- 详见models.PCNet

8. 剩余部分为训练后的计算过程，特别注意 evaluate_model

```
print('----- Training Options -----')
print('\n'.join(f'{k}: {v}' for k, v in cfg.items()))

if not cfg.load_pretrained:
    print(f'----- Start training {model_name:s} -----')
    pcnet, valid_psnr, valid_rmse, valid_ssim = train_pcnet(pcnet, train_data, valid_data, cfg)
else:
    # load the previously trained PCNet instead of retraining a new one
    print(f'----- Loading pretrained {model_name:s} -----')
    checkpoint_filename = join(data_root, '../checkpoint', ut.opt_to_string(cfg) + '.pth')
    pcnet.load_state_dict(torch.load(checkpoint_filename))

# [validation phase] after training we evaluate and save results
cam_valid_infer = evaluate_pcnet(pcnet, valid_data)[-1]
cam_valid_infer = evaluate_model(pcnet, valid_data)[-1]
valid_psnr, valid_rmse, valid_ssim, valid_l2, valid_linf, valid_dE = ut.calc_img_dists(cam_valid_infer, cam_v

# save results to log file
ret.loc[len(ret)] = [setup_name, model_name, loss, num_train, cfg.batch_size, cfg.max_iters, valid_psnr, valid
                    valid_ssim, valid_l2, valid_linf, valid_dE]
ut.write_log_file(ret, log_txt_filename, log_xls_filename) # in case unexpected interruptions, we save logs
```

9. evaluate_model函数:

- torch.zeros() 返回一个由标量值0填充的张量

torch.chunk() 对张量分块, 返回一个张量列表

- evaluate_model 输入选择的模型及相关属性和有效数据, 获取模型的相关参数(目前从程序中可以看出训练与验证/测试时都需要)

models.py

- CompenNet
 - nn.Conv2d() 二维卷积
 - nn.ConvTranspose2d() 二维反卷积
 - nn.ReLU() 对数据进行激活, 即非正取0, 其余不变
 - 骨干分支5个卷积层, 表面图像特征提取分支4个卷积层
 - skip layers 三个卷积层, 第一个卷积层由三个卷积层组合而成
 - _initialize_weights 初始化权重参数
 - 注意要剪枝表面分支, 以简化训练模型
 - 下图为网络结构, 结合其结构图解释更为形象

```
# x is the input uncompensated image, s is a 1x3x256x256 surface image
# Bingyao Huang
def forward(self, x, s):
    # surface feature extraction
    res1_s = self.relu(self.conv1_s(s)) if self.res1_s is None else self.res1_s
    res2_s = self.relu(self.conv2_s(res1_s)) if self.res2_s is None else self.res2_s
    res3_s = self.relu(self.conv3_s(res2_s)) if self.res3_s is None else self.res3_s
    res4_s = self.relu(self.conv4_s(res3_s)) if self.res4_s is None else self.res4_s

    # backbone
    res1 = self.skipConv1(x)
    x_____ = self.relu(self.conv1(x) + res1_s)
    res2 = self.skipConv2(x)
    x_____ = self.relu(self.conv2(x) + res2_s)
    res3 = self.skipConv3(x)
    x_____ = self.relu(self.conv3(x) + res3_s)
    x_____ = self.relu(self.conv4(x) + res4_s)
    x_____ = self.relu(self.conv5(x) + res3)
    x_____ = self.relu(self.transConv1(x) + res2)
    x_____ = self.relu(self.transConv2(x))
    x_____ = torch.clamp(self.relu(self.conv6(x) + res1), max=1)

    return x
```

- CompenNet++
 - 在 CompenNet 的基础上, 首先注意是否从已有的模型上获取初始化数据, 以及注意使用剪枝表面分支, 以简化训练模型, 最后注意使用 WarpingNet 进行几何矫正
- WarpingNet

- nn.leakyRelu() 类似于nn.ReLU(), 但是对于非正值考虑乘以一个系数
- 注意tuple定义以后不能修改里面的元素
- nn.Parameter() 将一个不可训练的tensor转换成可以训练的类型 parameter
- pytorch_tps.uniform_grid: TPS变换
- grid refinement net: 分是否需要改进, 对初始参数加工
- 简化训练模型到单一的采样网格, 以便进行更快的测试
- torch.nn.functional 常用模块, nn 中的大多数 layer在 functional 中都有一个与之对应的函数
- F.affine_grid 进行仿射变换
- F.grid_sample 进行网格采样, 一般用于对原图做采样, 得到trans_feature
- torch.clamp()方法将所有输入元素限制在[min, max]范围内, 并返回结果张量
- 下图为网络结构, 结合其结构图解释更为形象

```

Bingyao Huang
def forward(self, x):

    if self.fine_grid is None:
        # not simplified (training/validation)
        # generate coarse affine and TPS grids
        coarse_affine_grid = F.affine_grid(self.affine_mat, torch.Size([1, x.shape[1], x.shape[2], x.shape[3]]), align_corners=True)
        coarse_tps_grid = pytorch_tps.tps_grid(self.theta, self.ctrl_pts, (1, x.size()[1]) + self.out_size)

        # use TPS grid to sample affine grid
        tps_grid = F.grid_sample(coarse_affine_grid, coarse_tps_grid, align_corners=True).repeat(x.shape[0], 1, 1, 1)

        # refine TPS grid using grid refinement net and save it to self.fine_grid
        if self.with_refine:
            fine_grid = torch.clamp(self.grid_refine_net(tps_grid) + tps_grid, min=-1, max=1).permute((0, 2, 3, 1))
        else:
            fine_grid = torch.clamp(tps_grid, min=-1, max=1).permute((0, 2, 3, 1))
    else:
        # simplified (testing)
        fine_grid = self.fine_grid.repeat(x.shape[0], 1, 1, 1)

    # warp
    x = F.grid_sample(x, fine_grid, align_corners=True)
    return x

```

- ShadingNetSPAA
 - 与 CompenNet 整体差别不大
- PCNet
 - 注意 use_mask 和 use_rough(对应论文中两个操作: 光掩模和 rough shading image, 即实验最后两个可选项)
 - 下图为网络结构, 结合其结构图解释更为形象

```

# simplify trained model to a single sampling grid for faster testing
# Bingyao Huang
def simplify(self, s):
    self.warping_net.simplify(s)
    self.shading_net.simplify(self.warping_net(s))

# s is Bx3x256x256 surface image
# Bingyao Huang
def forward(self, x, s):
    # geometric correction using WarpingNet
    x = self.warping_net(x)

    if self.use_mask:
        x = x * self.mask
    if self.use_rough:
        x = self.shading_net(x, s, x * s)
    else:
        x = self.shading_net(x, s)

    return x

```

utils.py

- print_sys_info:
 - torch.cuda.device_count() 计数可用的GPU数量
 - torch.cuda.get_device_name() 获得GPU name
- set_torch_reproducibility:
 - torch.backends.cudnn.deterministic: flag 置为 True 时，每次返回的卷积算法将是确定的，即默认算法
 - torch.backends.cudnn.benchmark: 对 Pytorch 的 cudnn 底层库进行设置, 若设置为True，会使得 cuDNN 来衡量自己库里面的多个卷积算法的速度，然后选择其中最快的那个卷积算法(启动算法的前期会比较慢，但算法跑起来以后会非常快)

projector_based_attack

- summarize_all_attackers:
 - 完成实验复现需要的 Table 输出

```

for setup_name in tqdm(setup_list):
    setup_path = join(data_root, 'setups', setup_name)
    for attacker_name in attacker_names:
        attacker_cfg_str = to_attacker_cfg_str(attacker_name)[0]
        ret_path = join(setup_path, 'ret', attacker_cfg_str)
        print(f'Gathering stats of {ret_path}')

        # (time-consuming) recreate stats.txt, stats.xls and images in [ret] folder for each setup
        if recreate_stats_and_imgs:
            summarize_single_attacker(attacker_name=attacker_name, data_root=data_root, setup_list=[setup_name])
            table.append(pd.read_csv(join(ret_path, 'stats.txt'), index_col=None, header=0, sep='\t'))

table = pd.concat(table, axis=0, ignore_index=True)

# pivot_table is supplementary Table 2, and SPAA paper's Table 1 is its subset
pivot_table = pd.pivot_table(table,
                              # values=['T.top-1_real', 'T.top-5_real', 'U.top-1_real', 'T.real_L2', 'T.real_Linf', 'T.
                              values=['T.top-1_real', 'T.top-5_real', 'U.top-1_real', 'T.real_L2', 'T.real_Linf', 'T.re
                              index=['Attacker', 'd_thr', 'Stealth_loss', 'Classifier'], aggfunc=np.mean, sort=False)
pivot_table = pivot_table.sort_index(level=[0, 1], ascending=[False, True])_# to match SPAA Table order

# save tables
table.to_csv(join(data_root, 'setups/stats_all.txt'), index=False, float_format='%.4f', sep='\t')
table.to_excel(join(data_root, 'setups/stats_all.xlsx'), float_format='%.4f', index=False)
pivot_table.to_excel(join(data_root, 'setups/pivot_table_all.xlsx'), float_format='%.4f', index=True)

return table, pivot_table

```