**Train.py**

**from** glob **import** glob

**import** os

**from** albumentations **import** Compose, Normalize

**import** albumentations.pytorch **as** albu\_torch

**import** sys

sys.path.insert(1,**r'..\utility'**)

sys.path.insert(1,**r'..\models'**)

**from** dataloader **import** Mobile\_Dataset\_RAM

**from** logger **import** Logger

**from** loss **import** loss\_l2

**from** torch.utils.data **import** DataLoader

**from** models **import** ResNet18\_conv\_fc, ResNet8\_conv\_fc, ResNet12\_conv\_fc

**import** torch.optim **as** optim

**import** torch

**import** time

**import** argparse

**import** numpy **as** np

**import** pickle

device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)

TIME\_STAMP=time.strftime(**'%Y-%m-%d-%H-%M-%S'**)

parser=argparse.ArgumentParser()

parser.add\_argument(**'--dir\_project'**, help=**'project directory'**, default=**r'..'**)

parser.add\_argument(**'--dir\_lf'**, help=**'directory large files'**,default=**r'D:\Data\cs-8395-dl'**)

parser.add\_argument(**'--folderData'**, help=**'data directory'**, default=**'assignment1\_data'**)

parser.add\_argument(**'--encoder'**,help=**'encoder'**,default=**'resnet12'**)

parser.add\_argument(**'--lr'**, help=**'learning rate'**, type=float, default=0.001)

parser.add\_argument(**'--batchSize'**, help=**'batch size'**, type=int, default=32)

parser.add\_argument(**'--epoch'**, help=**'epoch'**, type=int, default=400)

parser.add\_argument(**'--resume\_from'**, help=**'filepath to resume training'**)

parser.add\_argument(**'--bottleneckFeatures'**, help=**'bottleneck the encoder Features'**, type=int, default=1)

args=parser.parse\_args()

*# setting up directories*

DIR\_LF = args.dir\_lf*#r'D:\Data\cs-8395-dl'*

dir\_data = os.path.join(DIR\_LF,args.folderData) *#os.path.join(DIR\_LF,'assignment1\_data')*

dir\_model = os.path.join(args.dir\_lf, **'model'**,TIME\_STAMP)

dir\_history = os.path.join(args.dir\_project, **'history'**)

dir\_log = os.path.join(args.dir\_project, **'log'**)

dir\_config = os.path.join(args.dir\_project, **'config'**)

**if** os.path.exists(dir\_history) **is False**:

os.mkdir(dir\_history)

**if** os.path.exists(dir\_log) **is False**:

os.mkdir(dir\_log)

**if** os.path.exists(dir\_config) **is False**:

os.mkdir(dir\_config)

**if** os.path.exists(os.path.join(args.dir\_lf, **'model'**)) **is False**:

os.mkdir(os.path.join(args.dir\_lf, **'model'**))

filepath\_hist = os.path.join(dir\_history, **'{}.bin'**.format(TIME\_STAMP))

filepath\_log = os.path.join(dir\_log, **'{}.log'**.format(TIME\_STAMP))

filepath\_cfg = os.path.join(dir\_config, **'{}.cfg'**.format(TIME\_STAMP))

sys.stdout = Logger(filepath\_log)

print(TIME\_STAMP)

print(os.path.basename(\_\_file\_\_))

config=vars(args)

config\_ls=sorted(list(config.items()))

print(**'--------------------------------------------------------------------------------------------------------------------'**)

**for** item **in** config\_ls:

print(**'{}: {}'**.format(item[0],item[1]))

print(**'--------------------------------------------------------------------------------------------------------------------'**)

**with** open(filepath\_cfg, **'w'**) **as** file:

**for** item **in** config\_ls:

file.write(**'{}: {}\n'**.format(item[0], item[1]))

**if** os.path.exists(dir\_model)==0:

print(**'creating directory to save model at {}'**.format(dir\_model))

os.mkdir(dir\_model)

filepath\_model\_best = os.path.join(dir\_model, **'{}\_{}\_best.pt'**.format(TIME\_STAMP, args.encoder)) *##*

dir\_data\_train = os.path.join(dir\_data, **'train'**)

filepaths\_train = glob(os.path.join(dir\_data\_train, **'\*.jpg'**))

flnames\_train = [os.path.basename(path)**for** path **in** filepaths\_train]

dir\_data\_valid = os.path.join(dir\_data, **'validation'**)

filepaths\_valid = glob(os.path.join(dir\_data\_valid, **'\*.jpg'**))

flnames\_valid = [os.path.basename(path)**for** path **in** filepaths\_valid]

filepath\_labels = os.path.join(dir\_data, **'labels'**, **'labels.txt'**)

**with** open(filepath\_labels, **'r'**) **as** f:

label\_data = f.readlines()

label\_dict = {}

**for** data **in** label\_data:

name, x, y = data.strip().split(**' '**)

label\_dict[name] = (float(x), float(y))

*# Dataloader*

aug = Compose([

*# Resize(256,256),*

*# RandomRotate90(),*

Normalize(),

albu\_torch.ToTensorV2()

],

)

BATCH\_SIZE=args.batchSize

LR = args.lr

EPOCH=args.epoch

Dataset\_train = Mobile\_Dataset\_RAM(dir\_data=dir\_data\_train,files=flnames\_train,label\_dict=label\_dict,transform=aug)

loader\_train=DataLoader(Dataset\_train,batch\_size=BATCH\_SIZE, shuffle=**True**)

print(**'train samples {}'**.format(len(Dataset\_train)))

Dataset\_valid = Mobile\_Dataset\_RAM(dir\_data=dir\_data\_valid,files=flnames\_valid,label\_dict=label\_dict,transform=aug)

loader\_valid=DataLoader(Dataset\_valid,batch\_size=BATCH\_SIZE, shuffle=**False**)

print(**'validation samples {}'**.format(len(Dataset\_valid)))

*# Model*

**if** args.encoder == **'resnet18'**:

model = ResNet18\_conv\_fc(pretrained=**True**, bottleneckFeatures=args.bottleneckFeatures).to(device)

**if** args.encoder == **'resnet8'**:

model = ResNet8\_conv\_fc(pretrained=**True**, bottleneckFeatures=args.bottleneckFeatures).to(device)

**if** args.encoder == **'resnet12'**:

model = ResNet12\_conv\_fc(pretrained=**True**, bottleneckFeatures=args.bottleneckFeatures).to(device)

print(model)

*# Optimizer*

optimizer = optim.Adam(model.parameters(), lr=LR, betas=(0.9, 0.999), eps=1e-08, weight\_decay=0,

amsgrad=**False**)

*# Train*

**if** args.resume\_from **is not None**:

*# Resume?*

print(**'resuming training from {}'**.format(args.resume\_from))

train\_states = torch.load(args.resume\_from)

model.load\_state\_dict(train\_states[**'model\_state\_dict'**])

optimizer.load\_state\_dict(train\_states[**'optimizer\_state\_dict'**])

epoch\_range = np.arange(train\_states[**'epoch'**]+1, train\_states[**'epoch'**]+1+EPOCH)

**else**:

train\_states = {

**'epoch'**: 0,

**'model\_state\_dict'**: model.state\_dict(),

**'optimizer\_state\_dict'**: optimizer.state\_dict(),

**'model\_save\_criteria'**: np.inf,

}

epoch\_range = np.arange(1,EPOCH+1)

loss\_train=[]

loss\_valid=[]

**for** epoch **in** epoch\_range:

running\_loss = 0

model.train()

**for** i, sample **in** enumerate(loader\_train):

optimizer.zero\_grad()

img = sample[0].to(device)

target = sample[1].to(device)

output = model(img)

loss = loss\_l2(target,output)

loss.backward()

optimizer.step()

running\_loss += loss.item()

mean\_loss = running\_loss / (i + 1)

print(**'train >>> epoch: {}/{}, batch: {}/{}, mean\_loss: {:.4f}'**.format(

epoch,

epoch\_range[-1],

i+1,

len(loader\_train),

mean\_loss

))

loss\_train.append(mean\_loss)

model.eval()

running\_loss = 0

**with** torch.no\_grad():

**for** i, sample **in** enumerate(loader\_valid):

img = sample[0].to(device)

target = sample[1].to(device)

output = model(img)

loss = loss\_l2(target, output)

*# print(loss.item())*

running\_loss += loss.item()

mean\_loss = running\_loss / (i + 1)

*# img\_r = reverse\_transform(img.cpu().squeeze())*

*# print(img\_r.shape)*

*# plt.imshow(img\_r)*

*# plt.plot(target.cpu().squeeze()[0] \* img\_r.shape[1], target.cpu().squeeze()[1]\* img\_r.shape[0], 'r\*')*

*# plt.plot(output.cpu().squeeze()[0] \* img\_r.shape[1], output.cpu().squeeze()[1] \* img\_r.shape[0], 'b\*')*

*# plt.show()*

print(**'valid >>> epoch: {}/{}, mean\_loss: {:.4f}'**.format(

epoch,

epoch\_range[-1],

mean\_loss

))

loss\_valid.append(mean\_loss)

log = {

**'loss\_train'**:loss\_train,

**'loss\_valid'**:loss\_valid

}

**with** open(filepath\_hist, **'wb'**) **as** pfile:

pickle.dump(log, pfile)

**if** mean\_loss<train\_states[**'model\_save\_criteria'**]:

print(**'criteria decreased from {:.4f} to {:.4f}, saving best model at {}'**.format(train\_states[**'model\_save\_criteria'**],

mean\_loss,

filepath\_model\_best))

train\_states = {

**'epoch'**: epoch,

**'model\_state\_dict'**: model.state\_dict(),

**'optimizer\_state\_dict'**: optimizer.state\_dict(),

**'model\_save\_criteria'**: mean\_loss,

}

torch.save(train\_states, filepath\_model\_best)

print(TIME\_STAMP)

**Test.py**

**import** os

**from** matplotlib **import** pyplot **as** plt

**import** argparse

**from** glob **import** glob

**import** sys

sys.path.insert(1,**r'..\utility'**)

sys.path.insert(1,**r'..\models'**)

**from** dataloader **import** Mobile\_Dataset\_RAM, reverse\_transform

**from** albumentations **import** Compose, Normalize

**import** albumentations.pytorch **as** albu\_torch

**from** torch.utils.data **import** DataLoader

**from** models **import** ResNet12\_conv\_fc

**import** torch

**import** torch.nn **as** nn

**from** skimage **import** io, transform

**import** numpy **as** np

device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)

parser=argparse.ArgumentParser()

parser.add\_argument(**'--filepath'**,required=**True**)

parser.add\_argument(**'--filepath\_model'**,default=**r'..\model\_weight\2020-01-27-20-33-41\_resnet12\_best.pt'**)

args = parser.parse\_args()

dir\_img = os.path.dirname(args.filepath)

flnames = os.path.basename(args.filepath)

aug = Compose([

Normalize(),

albu\_torch.ToTensorV2()

],

)

Dataset = Mobile\_Dataset\_RAM(dir\_data=dir\_img, files=[flnames], label\_dict=**None**, transform=aug)

print(**'number of samples {}'**.format(len(Dataset)))

loader=DataLoader(Dataset,batch\_size=1, shuffle=**False**)

model = ResNet12\_conv\_fc(pretrained=**False**,bottleneckFeatures=**False**).to(device)

res\_last\_conv = nn.Sequential(\*list(model.children())[:-2])

train\_states=torch.load(args.filepath\_model)

print(**'loading model from epoch {}, with criteria {}'**.format(train\_states[**'epoch'**],train\_states[**'model\_save\_criteria'**]))

model.load\_state\_dict(train\_states[**'model\_state\_dict'**])

model.eval()

**with** torch.no\_grad():

**for** i, sample **in** enumerate(loader):

img = sample[0].to(device)

output = res\_last\_conv(img)

img\_r = reverse\_transform(img.cpu().squeeze())

am\_np=output.squeeze().cpu().numpy()

am\_np\_rz=transform.resize(

am\_np,

img\_r.shape[:2])

x,y=np.unravel\_index(am\_np\_rz.argmax(), am\_np\_rz.shape)

print(x / img\_r.shape[0], y / img\_r.shape[1])

print(**'row, column ==> {:.4f}, {:.4f}'**.format(x / img\_r.shape[0], y / img\_r.shape[1]))

plt.imshow(img\_r)

plt.imshow(

am\_np\_rz,

alpha=0.3

)

plt.plot(y,x,**'ro'**)

plt.show()

**Models.py**

**import** torch

**import** torch.nn **as** nn

**import** torch.nn.functional **as** F

**from** torchvision **import** models

**from** torch **import** nn

**from** torch.nn **import** functional **as** F

**import** torch

**from** torchvision **import** models

**import** torchvision

**class** ResNet18(nn.Module):

**def** \_\_init\_\_(self, pretrained,bottleneckFeatures=1):

super(ResNet18,self).\_\_init\_\_()

self.resnet18 = models.resnet18(pretrained=pretrained)

self.resnet18\_fc\_stripped = nn.Sequential(\*list(self.resnet18.children())[:-1])

**if** bottleneckFeatures ==1:

print(**'freezing feature extracting layers'**)

**for** param **in** self.resnet18\_fc\_stripped.parameters():

param.requires\_grad = **False**

self.fc1 = nn.Linear(in\_features=512, out\_features=2)

**def** forward(self, x):

x = self.resnet18\_fc\_stripped(x)

x = x.reshape(x.size(0), -1)

*# print(x.shape)*

x = self.fc1(x)

**return** x

**class** ResNet18\_conv\_fc(nn.Module):

**def** \_\_init\_\_(self, pretrained,bottleneckFeatures=1):

super(ResNet18\_conv\_fc,self).\_\_init\_\_()

resnet18 = models.resnet18(pretrained=pretrained)

self.resnet18\_fc\_stripped = nn.Sequential(\*list(resnet18.children())[:-2])

**if** bottleneckFeatures ==1:

print(**'freezing feature extracting layers'**)

**for** param **in** self.resnet18\_fc\_stripped.parameters():

param.requires\_grad = **False**

self.conv\_last = nn.Conv2d(512,1,kernel\_size=(1,1),stride=(1,1))

self.fc1 = nn.Linear(in\_features=16\*11, out\_features=32)

self.fc2 = nn.Linear(in\_features=32, out\_features=2)

**def** forward(self, x):

x = self.resnet18\_fc\_stripped(x)

x = self.conv\_last(x)

x = x.reshape(x.size(0), -1)

x = self.fc1(x)

x = self.fc2(x)

**return** x

**class** ResNet8\_conv\_fc(nn.Module):

**def** \_\_init\_\_(self, pretrained,bottleneckFeatures=1):

super(ResNet8\_conv\_fc,self).\_\_init\_\_()

resnet18 = models.resnet18(pretrained=pretrained)

self.resnet\_partial = nn.Sequential(nn.Sequential(\*list(resnet18.children())[:5],

list(resnet18.children())[5][0]))

**if** bottleneckFeatures ==1:

print(**'freezing feature extracting layers'**)

**for** param **in** self.resnet\_partial.parameters():

param.requires\_grad = **False**

self.conv\_last = nn.Conv2d(128,1,kernel\_size=(1,1),stride=(1,1))

self.fc1 = nn.Linear(in\_features=62\*41, out\_features=32)

self.fc2 = nn.Linear(in\_features=32, out\_features=2)

**def** forward(self, x):

x = self.resnet\_partial(x)

*# print(x.shape)*

x = self.conv\_last(x)

x = x.reshape(x.size(0), -1)

x = self.fc1(x)

x = self.fc2(x)

**return** x

**class** ResNet12\_conv\_fc(nn.Module):

**def** \_\_init\_\_(self, pretrained,bottleneckFeatures=1):

super(ResNet12\_conv\_fc,self).\_\_init\_\_()

resnet18 = models.resnet18(pretrained=pretrained)

*# print(resnet18)*

self.resnet\_partial = nn.Sequential(\*list(resnet18.children())[:6], \*list(list(resnet18.children())[6][0].children())[:-1])

**if** bottleneckFeatures ==1:

print(**'freezing feature extracting layers'**)

**for** param **in** self.resnet\_partial.parameters():

param.requires\_grad = **False**

self.conv\_last = nn.Conv2d(256,1,kernel\_size=(1,1),stride=(1,1))

self.fc1 = nn.Linear(in\_features=31\*21, out\_features=32)

self.fc2 = nn.Linear(in\_features=32, out\_features=2)

**def** forward(self, x):

x = self.resnet\_partial(x)

*# print(x.shape)*

x = self.conv\_last(x)

x = x.reshape(x.size(0), -1)

x = self.fc1(x)

x = self.fc2(x)

**return** x

**if** \_\_name\_\_ ==**'\_\_main\_\_'**:

model=ResNet12\_conv\_fc(pretrained=**False**)

print(model)

data=torch.rand(2,3,490,326)

print(data.shape)

output=model(data)

print(output.shape)

**Dataloader.py**

**from** torch.utils.data **import** Dataset

**from** PIL **import** Image

**from** tqdm **import** tqdm

**import** os

**import** random

**import** numpy **as** np

**import** cv2

**import** torch

**from** glob **import** glob

**from** skimage **import** io

**from** scipy.ndimage **import** gaussian\_filter

**from** albumentations **import** (

Compose,

Normalize,

)

**import** albumentations.pytorch **as** albu\_torch

**from** matplotlib **import** pyplot **as** plt

**def** reverse\_transform(img\_t,mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]):

img\_r = np.array(img\_t)

img\_r = img\_r.transpose([1,2,0])

img\_r = img\_r\*std+mean

img\_r \*=255

img\_r=img\_r.astype(np.uint8)

img\_r = np.squeeze(img\_r)

**return** img\_r

**class** Mobile\_Dataset\_RAM(Dataset):

**def** \_\_init\_\_(self, dir\_data, files, label\_dict, transform=**None**):

self.dir\_data = dir\_data

self.transform = transform

self.files = files

self.image\_all=[]

self.label\_dict=label\_dict

print(**'loading images to RAM'**)

**for** file **in** tqdm(self.files):

*# file = self.files[idx]*

path\_img = os.path.join(self.dir\_data, file)

image = cv2.imread(path\_img)

self.image\_all.append(image)

**def** \_\_len\_\_(self):

size = len(self.files)

**return** size

**def** \_\_getitem\_\_(self, idx):

image=self.image\_all[idx]

**if 'test' in** self.dir\_data:

target=[0.5,0.5]

**else**:

target=self.label\_dict[self.files[idx]]

*# print(self.files[idx],image.shape)*

transformed=self.transform(image=image)

img = transformed[**'image'**]

**return** img,torch.tensor(target)

**if** \_\_name\_\_==**'\_\_main\_\_'**:

dir\_data = **r'D:\Data\cs-8395-dl\assignment1\_data'**

dir\_data\_train = os.path.join(dir\_data,**'train'**)

filepaths\_train = glob(os.path.join(dir\_data\_train, **'\*.jpg'**))

filepaths\_train\_label = os.path.join(dir\_data, **'labels'**, **'labels.txt'**)

**with** open(filepaths\_train\_label, **'r'**) **as** f:

label\_data = f.readlines()

label\_dict = {}

**for** data **in** label\_data:

name, x, y = data.strip().split(**' '**)

label\_dict[name] = (float(x), float(y))

aug = Compose([

*# Resize(256,256),*

*# RandomRotate90(),*

Normalize(),

albu\_torch.ToTensorV2()

],

)

files = list(label\_dict.keys())

Mobile\_Dataset = Mobile\_Dataset\_HM\_RAM(dir\_data=dir\_data\_train,files=files,label\_dict=label\_dict,transform=aug)

sample = Mobile\_Dataset[0]

img = sample[0]

target = sample[1].cpu().numpy()

img = reverse\_transform(img)

plt.imshow(img)

plt.imshow(target, alpha=0.3)

plt.show()

**Logger.py**

**import** sys

**class** Logger(object):

**def** \_\_init\_\_(self,path):

self.terminal = sys.stdout

self.log = open(path, **"a+"**)

**def** write(self, message):

self.terminal.write(message)

self.log.write(message)

**def** flush(self):

*#this flush method is needed for python 3 compatibility.*

*#this handles the flush command by doing nothing.*

*#you might want to specify some extra behavior here.*

**pass**

**loss.py**

**import** torch

**def** loss\_l2(y,y\_p, gamma=0):

*#calculate loss per sample*

y=y.double()

y\_p=y\_p.double()

loss = (((y-y\_p)\*\*gamma)\*((y-y\_p)\*(y-y\_p))).sum()/y.shape[0]

**return** loss