AI機器人於思覺失調患者之應用服務開發建置案

異狀行為預測軟體原始碼及資料庫說明

使用說明

1. 從 google drive取得原始碼並初始化：

https://drive.google.com/drive/folders/1Uvxwet8Z-JRPP3fllr9H5h4Tp7oIU7qb?usp=sharing

* + 下載資料集：

https://caer-dataset.github.io/

1. 建立 Python 虛擬環境並安裝需要的套件：

|  |
| --- |
| $ conda create -n pywork python=3.6  $ source activate pywork  $ pip install -r requirements.txt |

1. 建立 dataset list與預先偵測人臉：

|  |
| --- |
| $ python3 build\_train\_list.py  $ python3 det\_face.py |

1. 執行training code：

|  |
| --- |
| $ python3 train.py |

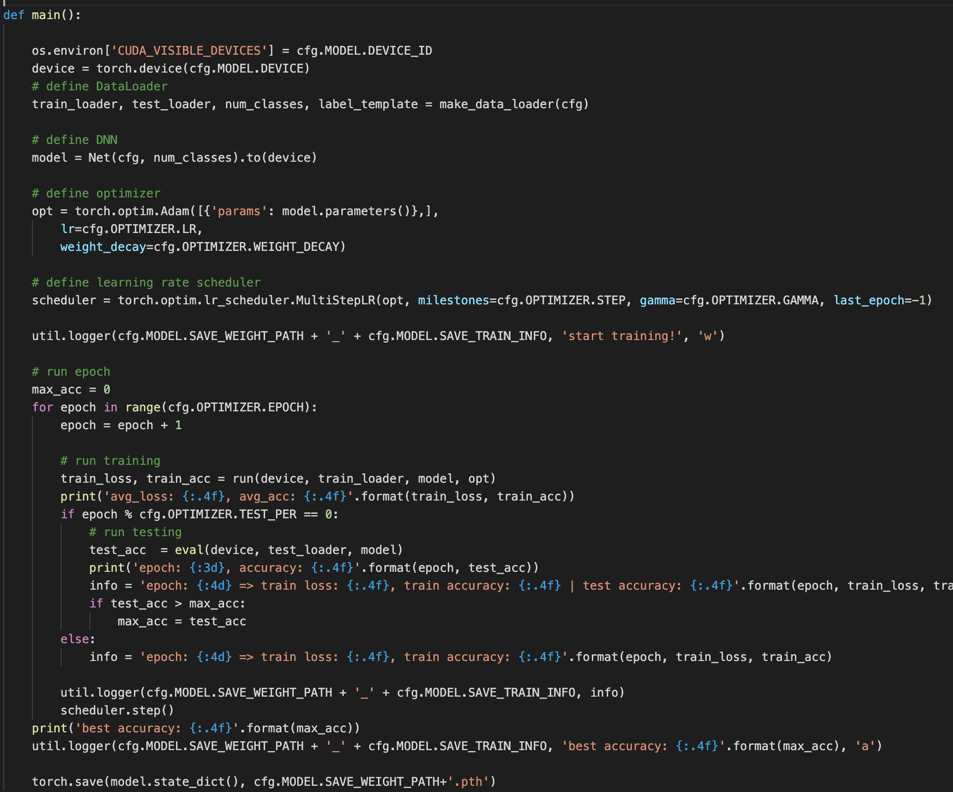
1. 執行inference code：

|  |
| --- |
| $ python3 inference.py |

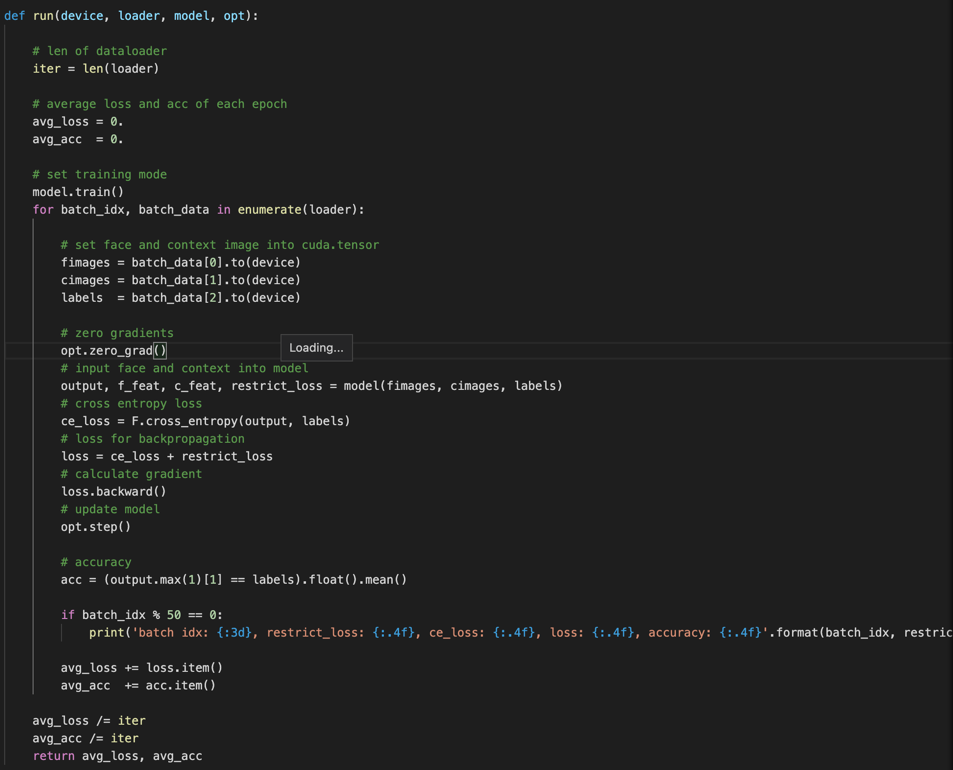
程式碼架構

|  |
| --- |
| project\_code  └── project\_code  ├── data #資料集，若無需再次訓練可忽略  ├── dataset # 資料集載入  ├── model # 模型定義  ├── model\_weight # model 儲存  ├── build\_train\_list.py # 前處理 (用於訓練的資料載入)  ├── inference.py # 偵測異常行為程式  ├── util.py # common function  └── det\_face.py ＃ 前處理 （偵測人臉） |

* 情緒識別: 透過人臉(face)與完整環境(context)來訓練模型
  + train.py: 訓練程式碼
    - main(): 主程式
      * 定義dataloader與model
      * 迴圈執行training與testing



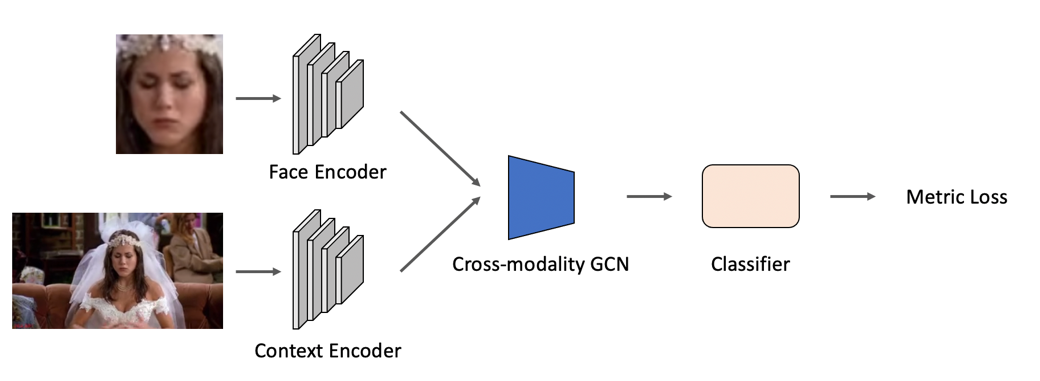
* + - run(): training function

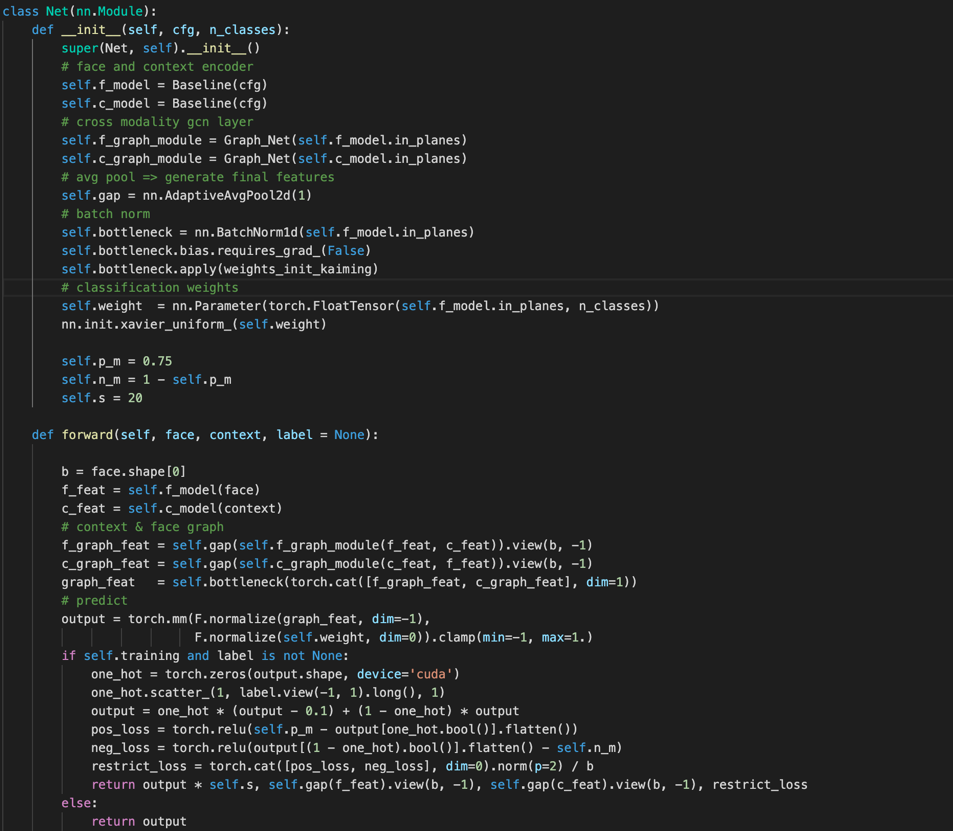


* + - eval(): testing function

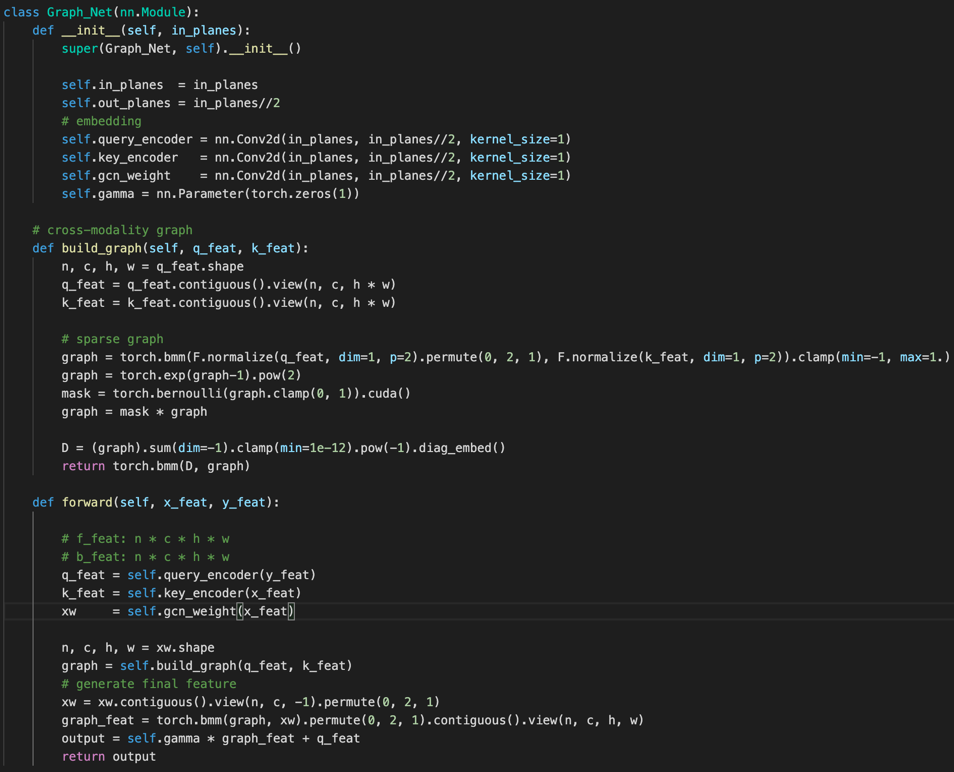


* + ./model/overall\_net.py:
    - Net: 定義model架構
      * Baseline: convolutional neural networks encoder
      * Graph\_Net: cross-modality graph convolutional networks
      * weight: classifier
      * p\_m, n\_m and s: 計算metric loss之參數

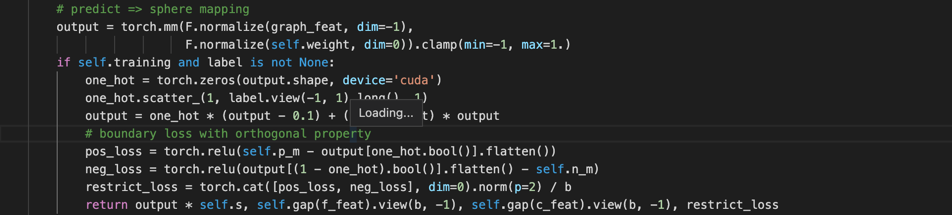




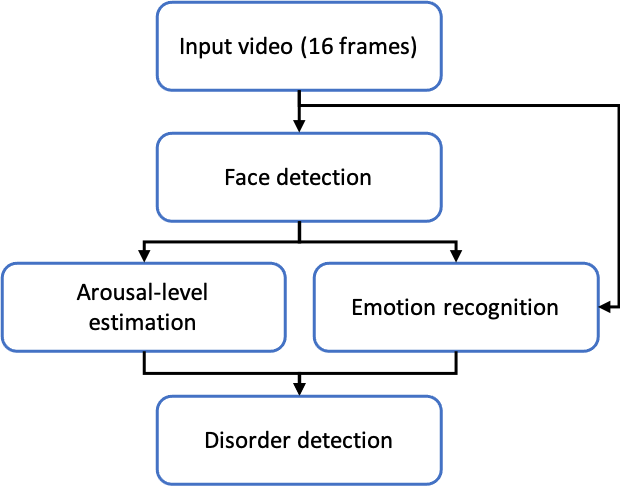
* + - Graph\_Net: cross-modality gcn
      * build\_graph():
        + 計算face與context的affinity graph
        + 利用bernoulli sampling來達成稀疏的graph



* + - metric loss: 定義boundary
      * 利用boundary（pos: 0.75, neg: 0.25）來更好的規範特徵分佈

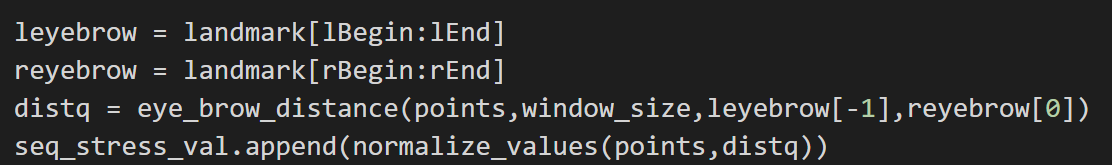


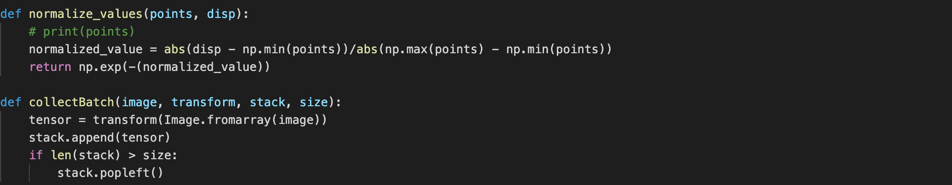
* ./inference.py: 異常偵測



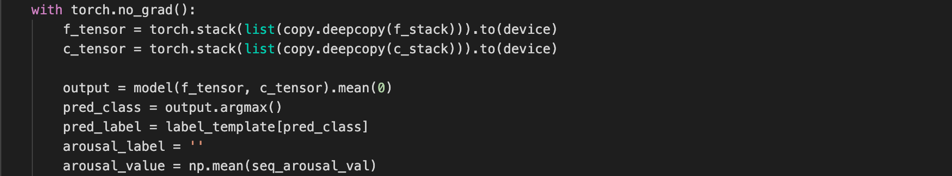
圖二、異常偵測流程圖

* + 每16個frame（0.5 s）組成一個video clips，並對整個clip進行人臉偵測，接下來分別進行movement estimation與emotion recognition。
  + Arousal-level estimation: 根據左右眼睛與眉毛的特徵點（action unit）的位移來估計病患當前的位移程度
    - eye\_brow\_distance: 計算點集合間的Euclidean distance
    - normalize\_values: 透過與temporal set (16 frame)的比較與標準化，可以衡量人的移動程度。

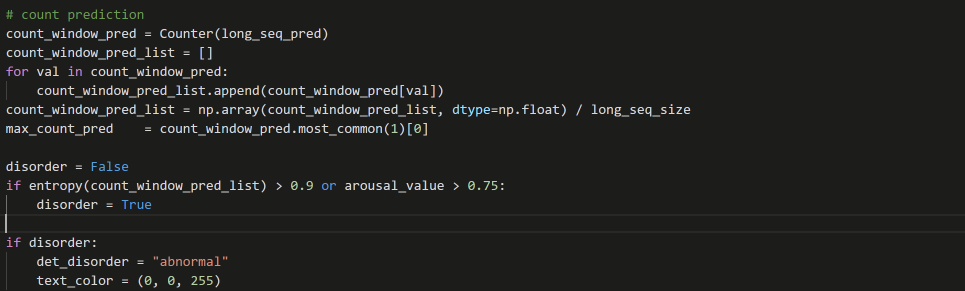




* + emotion recognition: 人顯露的情緒大概會維持1~2秒，因此我們對每個video clip (16 frames)預測一emotion，接著利用sliding window的方式來統計emotion的變化(64 clips)



* + disorder detection: 在這裏我們根據arousal-level與emotion的變化(entropy)來進行異常行為偵測。



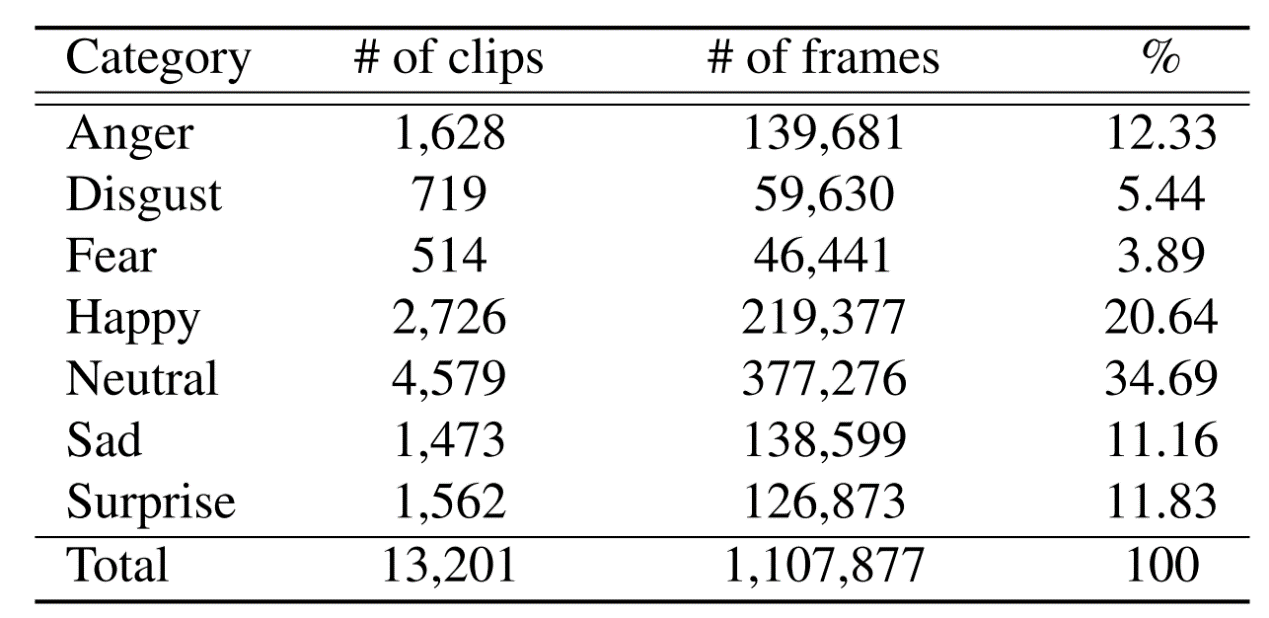
實驗結果:

* 資料集: 我們採用CAER(Context-aware Emotion Recognition)資料集來驗證模型的有效性。CAER資料集包含6種標準的情緒(Happy, Anger, Disgust, Sad, Surprise, Fear)與1中性類別(Neutral)，如下圖一所示。



圖一、CAER資料集的類別與範例圖片

CAER 資料集的分布如下:



* 結果:

|  |  |  |
| --- | --- | --- |
| method | Input type | Accuracy |
| CAER-2D | Image | 73.51 |
| CAER-3D | Video | 77.04 |
| ours | Image | 87.26 |

Source Code:

config

./configs/image\_config.py

from yacs.config import CfgNode as CN

\_C = CN()

\_C.DATASETS = CN()

\_C.DATASETS.NAMES = ('CAER\_S')

\_C.DATASETS.ROOT\_DIR = ('./data/CAER/')

\_C.DATALOADER = CN()

# Number of data loading threads

\_C.DATALOADER.NUM\_WORKERS = 8

# Sampler for data loading

\_C.DATALOADER.SAMPLER = 'PK\_BATCH'

# Number of instance for one batch

\_C.DATALOADER.NUM\_INSTANCE = 10

\_C.DATALOADER.IMS\_PER\_BATCH = 70

\_C.MODEL = CN()

\_C.MODEL.DEVICE = "cuda"

\_C.MODEL.DEVICE\_ID = '0'

# \_C.MODEL.BACKBONE = 'resnet18'

# \_C.MODEL.PRETRAIN\_CHOICE = 'imagenet'

\_C.MODEL.BACKBONE = 'convnet'

\_C.MODEL.PRETRAIN\_CHOICE = ''

\_C.MODEL.PRETRAIN\_PATH = './imagenet\_weights/' + \_C.MODEL.BACKBONE + '.pth'

\_C.MODEL.SAVE\_WEIGHT\_PATH = './model\_weights/' + \_C.MODEL.BACKBONE + '\_' + \_C.DATASETS.NAMES

\_C.MODEL.SAVE\_TRAIN\_INFO = 'training\_info.txt'

\_C.INPUT = CN()

# Size of the image during training

\_C.INPUT.SIZE\_TRAIN = [112, 112]

# Size of the image during test

\_C.INPUT.SIZE\_TEST = [112, 112]

# Random probability for image horizontal flip

\_C.INPUT.PROB = 0.5

# Random probability for random erasing

\_C.INPUT.RE\_PROB = 0.5

# Values to be used for image normalization

\_C.INPUT.PIXEL\_MEAN = [0.485, 0.456, 0.406]

# Values to be used for image normalization

\_C.INPUT.PIXEL\_STD = [0.229, 0.224, 0.225]

\_C.OPTIMIZER = CN()

\_C.OPTIMIZER.OPTIMIZER\_NAME = 'Adam'

if \_C.MODEL.PRETRAIN\_CHOICE == 'imagenet':

\_C.OPTIMIZER.EPOCH = 70

\_C.OPTIMIZER.STEP = [50]

\_C.OPTIMIZER.GAMMA = 0.1

\_C.OPTIMIZER.LR = 5e-4

else:

\_C.OPTIMIZER.EPOCH = 300

\_C.OPTIMIZER.STEP = [40, 250]

\_C.OPTIMIZER.GAMMA = 0.1

\_C.OPTIMIZER.LR = 5e-3

\_C.OPTIMIZER.WEIGHT\_DECAY = 1e-5

\_C.OPTIMIZER.E\_MARGIN = 0.5

\_C.OPTIMIZER.TEST\_PER = 10

Dataloader

./dataset/\_\_init\_\_.py

from .build import make\_data\_loader

./dataset/build.py

from torch.utils.data import DataLoader

from .collate\_batch import collate\_fn

from .datasets import init\_dataset, ImageDataset

from .samplers import RandomIdentitySampler

from .transforms import build\_transforms

def make\_data\_loader(cfg):

train\_transforms\_f = build\_transforms(cfg, is\_train=True)

train\_transforms\_c = build\_transforms(cfg, is\_drop=True, is\_train=True)

val\_transforms = build\_transforms(cfg, is\_train=False)

num\_workers = cfg.DATALOADER.NUM\_WORKERS

dataset = init\_dataset(cfg.DATASETS.NAMES, root=cfg.DATASETS.ROOT\_DIR)

label\_template = dataset.label\_template

num\_classes = dataset.num\_train\_eids

train\_set = ImageDataset(dataset.train, train\_transforms\_f, train\_transforms\_c)

train\_loader = DataLoader(

train\_set, batch\_size=cfg.DATALOADER.IMS\_PER\_BATCH,

sampler=RandomIdentitySampler(dataset.train, cfg.DATALOADER.IMS\_PER\_BATCH, cfg.DATALOADER.NUM\_INSTANCE),

num\_workers=num\_workers, collate\_fn=collate\_fn

)

test\_set = ImageDataset(dataset.test, val\_transforms, val\_transforms)

test\_loader = DataLoader(

test\_set, batch\_size=cfg.DATALOADER.IMS\_PER\_BATCH, shuffle=False, num\_workers=num\_workers,

collate\_fn=collate\_fn

)

return train\_loader, test\_loader, num\_classes, label\_template

./dataset/collate\_batch.py

import torch

def collate\_fn(batch):

c\_imgs, f\_imgs, eids = zip(\*batch)

eids = torch.tensor(eids, dtype=torch.int64)

return torch.stack(c\_imgs, dim=0), torch.stack(f\_imgs, dim=0), eids

./dataset/datasets/\_\_init\_\_.py

from .CAER\_S import CAER\_S

from .CAER import CAER

from .dataset\_loader import ImageDataset

\_\_factory = {

'CAER\_S': CAER\_S,

'CAER': CAER

}

def get\_names():

return \_\_factory.keys()

def init\_dataset(name, \*args, \*\*kwargs):

if name not in \_\_factory.keys():

raise KeyError("Unknown datasets: {}".format(name))

return \_\_factory[name](\*args, \*\*kwargs)

./dataset/datasets/bases.py

import numpy as np

class BaseDataset(object):

"""

Base class of CAER dataset

"""

def get\_imagedata\_info(self, data):

eids = []

for \_, \_, eid in data:

eids += [eid]

eids = set(eids)

num\_eids = len(eids)

num\_imgs = len(data)

return num\_eids, num\_imgs

def get\_videodata\_info(self, data):

eids = []

for \_, eid in data:

eids += [eid]

eids = set(eids)

num\_eids = len(eids)

num\_videos = len(data)

return num\_eids, num\_videos

def print\_dataset\_statistics(self):

raise NotImplementedError

class BaseImageDataset(BaseDataset):

"""

Base class of image dataset

"""

def print\_dataset\_statistics(self, train, test):

num\_train\_eids, num\_train\_imgs = self.get\_imagedata\_info(train)

num\_test\_eids, num\_test\_imgs = self.get\_imagedata\_info(test)

print("Dataset statistics:")

print(" ----------------------------------------")

print(" subset | # ids | # images |")

print(" ----------------------------------------")

print(" train | {:5d} | {:8d} |".format(num\_train\_eids, num\_train\_imgs))

print(" test | {:5d} | {:8d} |".format(num\_test\_eids, num\_test\_imgs))

print(" ----------------------------------------")

class BaseVideoDataset(BaseDataset):

"""

Base class of video dataset

"""

def print\_dataset\_statistics(self, train, validation, test):

num\_train\_eids, num\_train = self.get\_videodata\_info(train)

num\_validation\_eids, num\_validation = self.get\_videodata\_info(validation)

num\_test\_eids, num\_test = self.get\_videodata\_info(test)

print("Dataset statistics:")

print(" ----------------------------------------")

print(" subset | # ids | # videos |")

print(" ----------------------------------------")

print(" train | {:5d} | {:8d} |".format(num\_train\_eids, num\_train))

print(" validation | {:5d} | {:8d} |".format(num\_validation\_eids, num\_validation))

print(" test | {:5d} | {:8d} |".format(num\_test\_eids, num\_test))

print(" ----------------------------------------")

./dataset/datasets/CAER\_S.py

import os

import os.path as osp

from .bases import BaseImageDataset

class CAER\_S(BaseImageDataset):

dataset\_dir = 'CAER-S/CAER-S'

def \_\_init\_\_(self, root='./data/', \*\*kwargs):

super(CAER\_S, self).\_\_init\_\_()

self.dataset\_dir = osp.join(root, self.dataset\_dir)

self.train\_dir = osp.join(self.dataset\_dir, 'train')

self.test\_dir = osp.join(self.dataset\_dir, 'test')

self.label\_template = self.\_make\_id\_template(self.train\_dir)

self.train = self.\_process\_dir(self.train\_dir)

self.test = self.\_process\_dir(self.test\_dir)

self.num\_train\_eids, self.num\_train\_imgs = self.get\_imagedata\_info(self.train)

self.num\_test\_eids, self.num\_test\_imgs = self.get\_imagedata\_info(self.test)

self.print\_dataset\_statistics(self.train, self.test)

def \_make\_id\_template(self, train\_dir):

label\_template = []

for label in os.listdir(train\_dir):

label\_template.append(label)

return label\_template

def \_process\_dir(self, dir\_path):

dataset = []

no\_face = 0

for key in os.listdir(dir\_path):

eid = self.label\_template.index(key)

image\_dir = osp.join(dir\_path, key)

for image\_path in os.listdir(image\_dir):

context\_path = osp.join(image\_dir, image\_path)

face\_path = context\_path.replace('CAER-S/CAER-S', 'CAER-S/CAER-S-FACE')

if os.path.isfile(face\_path):

dataset.append((context\_path, face\_path, eid))

else:

no\_face += 1

dataset.append((context\_path, context\_path, eid))

return dataset

./dataset/datasets/dataset\_loader.py

import os.path as osp

from PIL import Image

from torch.utils.data import Dataset

def read\_image(img\_path):

"""Keep reading image until succeed.

This can avoid IOError incurred by heavy IO process."""

got\_img = False

if not osp.exists(img\_path):

raise IOError("{} does not exist".format(img\_path))

while not got\_img:

try:

img = Image.open(img\_path).convert('RGB')

got\_img = True

except IOError:

print("IOError incurred when reading '{}'. Will redo. Don't worry. Just chill.".format(img\_path))

pass

return img

class ImageDataset(Dataset):

def \_\_init\_\_(self, dataset, f\_transform = None, c\_transform=None):

self.dataset = dataset

self.c\_transform = c\_transform

self.f\_transform = f\_transform

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

return len(self.dataset)

def \_\_getitem\_\_(self, index):

context\_path, face\_path, id = self.dataset[index]

context = read\_image(context\_path)

face = read\_image(face\_path)

if (self.c\_transform is not None) and (self.f\_transform is not None):

context = self.c\_transform(context)

face = self.f\_transform(face)

return context, face, id

./dataset/samplers/\_\_init\_\_.py

from .triplet\_sampler import RandomIdentitySampler

./dataset/samplers/triplet\_sampler.py

import copy

import random

import torch

from collections import defaultdict

import numpy as np

from torch.utils.data.sampler import Sampler

class RandomIdentitySampler(Sampler):

"""

Randomly sample N identities, then for each identity,

randomly sample K instances, therefore batch size is N\*K.

Args:

- data\_source (list): list of (img\_path, pid, camid).

- num\_instances (int): number of instances per identity in a batch.

- batch\_size (int): number of examples in a batch.

"""

def \_\_init\_\_(self, data\_source, batch\_size, num\_instances):

self.data\_source = data\_source

self.batch\_size = batch\_size

self.num\_instances = num\_instances

self.num\_pids\_per\_batch = self.batch\_size // self.num\_instances

self.index\_dic = defaultdict(list)

for index, (\_, \_, pid) in enumerate(self.data\_source):

self.index\_dic[pid].append(index)

self.pids = list(self.index\_dic.keys())

# estimate number of examples in an epoch

self.length = 0

for pid in self.pids:

idxs = self.index\_dic[pid]

num = len(idxs)

if num < self.num\_instances:

num = self.num\_instances

self.length += num - num % self.num\_instances

def \_\_iter\_\_(self):

batch\_idxs\_dict = defaultdict(list)

for pid in self.pids:

idxs = copy.deepcopy(self.index\_dic[pid])

if len(idxs) < self.num\_instances:

idxs = np.random.choice(idxs, size=self.num\_instances, replace=True)

random.shuffle(idxs)

batch\_idxs = []

for idx in idxs:

batch\_idxs.append(idx)

if len(batch\_idxs) == self.num\_instances:

batch\_idxs\_dict[pid].append(batch\_idxs)

batch\_idxs = []

avai\_pids = copy.deepcopy(self.pids)

final\_idxs = []

while len(avai\_pids) >= self.num\_pids\_per\_batch:

selected\_pids = random.sample(avai\_pids, self.num\_pids\_per\_batch)

for pid in selected\_pids:

batch\_idxs = batch\_idxs\_dict[pid].pop(0)

final\_idxs.extend(batch\_idxs)

if len(batch\_idxs\_dict[pid]) == 0:

avai\_pids.remove(pid)

self.length = len(final\_idxs)

return iter(final\_idxs)

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

return self.length

./dataset/datasets/transforms/\_\_init\_\_.py

from .build import build\_transforms

./dataset/datasets/transforms/build.py

import torchvision.transforms as T

from .transforms import RandomErasing

from .transforms import RandomPatch

from .transforms import Cutout

def build\_transforms(cfg, is\_drop=False, is\_train=True, is\_face=False):

normalize\_transform = T.Normalize(mean=cfg.INPUT.PIXEL\_MEAN, std=cfg.INPUT.PIXEL\_STD)

if is\_train:

transform = [

T.Resize(cfg.INPUT.SIZE\_TRAIN),

T.Pad(10),

T.RandomCrop(cfg.INPUT.SIZE\_TRAIN),

T.RandomHorizontalFlip(p=cfg.INPUT.PROB),

T.ToTensor(),

normalize\_transform

]

if is\_drop:

transform.append(RandomErasing(probability=cfg.INPUT.RE\_PROB, mean=cfg.INPUT.PIXEL\_MEAN))

transform = T.Compose(transform)

else:

if is\_face:

transform = T.Compose([

T.Resize(cfg.INPUT.SIZE\_TEST),

T.ToTensor(),

normalize\_transform])

else:

transform = T.Compose([

T.Resize((128, 171)),

T.CenterCrop(cfg.INPUT.SIZE\_TEST),

T.ToTensor(),

normalize\_transform])

return transform

./dataset/datasets/transforms/transforms.py

from PIL import Image

import numpy as np

import torch

import math

import random

from collections import deque

class RandomPatch(object):

"""Random patch data augmentation.

There is a patch pool that stores randomly extracted pathces from person images.

For each input image, RandomPatch

1) extracts a random patch and stores the patch in the patch pool;

2) randomly selects a patch from the patch pool and pastes it on the

input (at random position) to simulate occlusion.

Reference:

- Zhou et al. Omni-Scale Feature Learning for Person Re-Identification. ICCV, 2019.

- Zhou et al. Learning Generalisable Omni-Scale Representations

for Person Re-Identification. arXiv preprint, 2019.

"""

def \_\_init\_\_(

self,

prob\_happen=0.5,

pool\_capacity=50000,

min\_sample\_size=100,

patch\_min\_area=0.01,

patch\_max\_area=0.5,

patch\_min\_ratio=0.1,

prob\_rotate=0.5,

prob\_flip\_leftright=0.5,

):

self.prob\_happen = prob\_happen

self.patch\_min\_area = patch\_min\_area

self.patch\_max\_area = patch\_max\_area

self.patch\_min\_ratio = patch\_min\_ratio

self.prob\_rotate = prob\_rotate

self.prob\_flip\_leftright = prob\_flip\_leftright

self.patchpool = deque(maxlen=pool\_capacity)

self.min\_sample\_size = min\_sample\_size

def generate\_wh(self, W, H):

area = W \* H

for attempt in range(100):

target\_area = random.uniform(

self.patch\_min\_area, self.patch\_max\_area

) \* area

aspect\_ratio = random.uniform(

self.patch\_min\_ratio, 1. / self.patch\_min\_ratio

)

h = int(round(math.sqrt(target\_area \* aspect\_ratio)))

w = int(round(math.sqrt(target\_area / aspect\_ratio)))

if w < W and h < H:

return w, h

return None, None

def transform\_patch(self, patch):

if random.uniform(0, 1) > self.prob\_flip\_leftright:

patch = patch.transpose(Image.FLIP\_LEFT\_RIGHT)

if random.uniform(0, 1) > self.prob\_rotate:

patch = patch.rotate(random.randint(-10, 10))

return patch

def \_\_call\_\_(self, img):

W, H = img.size # original image size

# collect new patch

w, h = self.generate\_wh(W, H)

if w is not None and h is not None:

x1 = random.randint(0, W - w)

y1 = random.randint(0, H - h)

new\_patch = img.crop((x1, y1, x1 + w, y1 + h))

self.patchpool.append(new\_patch)

if len(self.patchpool) < self.min\_sample\_size:

return img

if random.uniform(0, 1) > self.prob\_happen:

return img

# paste a randomly selected patch on a random position

patch = random.sample(self.patchpool, 1)[0]

patchW, patchH = patch.size

x1 = random.randint(0, W - patchW)

y1 = random.randint(0, H - patchH)

patch = self.transform\_patch(patch)

img.paste(patch, (x1, y1))

return img

class RandomErasing(object):

""" Randomly selects a rectangle region in an image and erases its pixels.

'Random Erasing Data Augmentation' by Zhong et al.

See https://arxiv.org/pdf/1708.04896.pdf

Args:

probability: The probability that the Random Erasing operation will be performed.

sl: Minimum proportion of erased area against input image.

sh: Maximum proportion of erased area against input image.

r1: Minimum aspect ratio of erased area.

mean: Erasing value.

"""

def \_\_init\_\_(self, probability=0.5, sl=0.02, sh=0.4, r1=0.3, mean=(0.4914, 0.4822, 0.4465)):

self.probability = probability

self.mean = mean

self.sl = sl

self.sh = sh

self.r1 = r1

def \_\_call\_\_(self, img):

if random.uniform(0, 1) >= self.probability:

return img

for attempt in range(100):

area = img.size()[1] \* img.size()[2]

target\_area = random.uniform(self.sl, self.sh) \* area

aspect\_ratio = random.uniform(self.r1, 1 / self.r1)

h = int(round(math.sqrt(target\_area \* aspect\_ratio)))

w = int(round(math.sqrt(target\_area / aspect\_ratio)))

if w < img.size()[2] and h < img.size()[1]:

x1 = random.randint(0, img.size()[1] - h)

y1 = random.randint(0, img.size()[2] - w)

if img.size()[0] == 3:

img[0, x1:x1 + h, y1:y1 + w] = self.mean[0]

img[1, x1:x1 + h, y1:y1 + w] = self.mean[1]

img[2, x1:x1 + h, y1:y1 + w] = self.mean[2]

else:

img[0, x1:x1 + h, y1:y1 + w] = self.mean[0]

return img

return img

class Cutout(object):

"""Randomly mask out one or more patches from an image.

Args:

n\_holes (int): Number of patches to cut out of each image.

length (int): The length (in pixels) of each square patch.

"""

def \_\_init\_\_(self, n\_holes, length):

self.n\_holes = n\_holes

self.length = length

def \_\_call\_\_(self, img):

"""

Args:

img (Tensor): Tensor image of size (C, H, W).

Returns:

Tensor: Image with n\_holes of dimension length x length cut out of it.

"""

h = img.size(1)

w = img.size(2)

mask = np.ones((h, w), np.float32)

for n in range(self.n\_holes):

y = np.random.randint(h)

x = np.random.randint(w)

y1 = np.clip(y - self.length // 2, 0, h)

y2 = np.clip(y + self.length // 2, 0, h)

x1 = np.clip(x - self.length // 2, 0, w)

x2 = np.clip(x + self.length // 2, 0, w)

mask[y1: y2, x1: x2] = 0.

mask = torch.from\_numpy(mask)

mask = mask.expand\_as(img)

img = img \* mask

return img

./model/\_\_init\_\_.py

from .baseline import Baseline

def build\_model(cfg):

model = Baseline(cfg.MODEL.PRETRAIN\_PATH, cfg.MODEL.BACKBONE, cfg.MODEL.PRETRAIN\_CHOICE)

return model

./model/baseline.py

import torch

from torch import nn

import torch.nn.functional as F

from torch.jit.annotations import Dict

from .backbones.resnet import ResNet, BasicBlock, Bottleneck

from .backbones.senet import SENet, SEResNetBottleneck, SEBottleneck, SEResNeXtBottleneck

from .backbones.resnet\_ibn\_a import resnet50\_ibn\_a

from .backbones.convnet import ConvNet

# show

from collections import OrderedDict

from PIL import Image

from torchvision import datasets, models, transforms

import matplotlib.pyplot as plt

def weights\_init\_kaiming(m):

classname = m.\_\_class\_\_.\_\_name\_\_

if classname.find('Linear') != -1:

nn.init.kaiming\_normal\_(m.weight, a=0, mode='fan\_out')

nn.init.constant\_(m.bias, 0.0)

elif classname.find('Conv') != -1:

nn.init.kaiming\_normal\_(m.weight, a=0, mode='fan\_in')

if m.bias is not None:

nn.init.constant\_(m.bias, 0.0)

elif classname.find('BatchNorm') != -1:

if m.affine:

nn.init.constant\_(m.weight, 1.0)

nn.init.constant\_(m.bias, 0.0)

class Baseline(nn.Module):

in\_planes = 2048

def \_\_init\_\_(self, cfg):

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

model\_path = cfg.MODEL.PRETRAIN\_PATH

model\_name = cfg.MODEL.BACKBONE

pretrain\_choice = cfg.MODEL.PRETRAIN\_CHOICE

last\_stride = 1

if model\_name == 'resnet18':

self.in\_planes = 512

self.base = ResNet(last\_stride=last\_stride,

block=BasicBlock,

layers=[2, 2, 2, 2])

self.expansion = BasicBlock.expansion

elif model\_name == 'resnet34':

self.in\_planes = 512

self.base = ResNet(last\_stride=last\_stride,

block=BasicBlock,

layers=[3, 4, 6, 3])

self.expansion = BasicBlock.expansion

elif model\_name == 'resnet50':

self.base = ResNet(last\_stride=last\_stride,

block=Bottleneck,

layers=[3, 4, 6, 3])

self.expansion = BasicBlock.expansion

elif model\_name == 'resnet101':

self.base = ResNet(last\_stride=last\_stride,

block=Bottleneck,

layers=[3, 4, 23, 3])

self.expansion = BasicBlock.expansion

elif model\_name == 'resnet152':

self.base = ResNet(last\_stride=last\_stride,

block=Bottleneck,

layers=[3, 8, 36, 3])

self.expansion = BasicBlock.expansion

elif model\_name == 'se\_resnet50':

self.base = SENet(block=SEResNetBottleneck,

layers=[3, 4, 6, 3],

groups=1,

reduction=16,

dropout\_p=None,

inplanes=64,

input\_3x3=False,

downsample\_kernel\_size=1,

downsample\_padding=0,

last\_stride=last\_stride)

self.expansion = SEResNetBottleneck.expansion

elif model\_name == 'se\_resnet101':

self.base = SENet(block=SEResNetBottleneck,

layers=[3, 4, 23, 3],

groups=1,

reduction=16,

dropout\_p=None,

inplanes=64,

input\_3x3=False,

downsample\_kernel\_size=1,

downsample\_padding=0,

last\_stride=last\_stride)

self.expansion = SEResNetBottleneck.expansion

elif model\_name == 'se\_resnet152':

self.base = SENet(block=SEResNetBottleneck,

layers=[3, 8, 36, 3],

groups=1,

reduction=16,

dropout\_p=None,

inplanes=64,

input\_3x3=False,

downsample\_kernel\_size=1,

downsample\_padding=0,

last\_stride=last\_stride)

self.expansion = SEResNetBottleneck.expansion

elif model\_name == 'se\_resnext50':

self.base = SENet(block=SEResNeXtBottleneck,

layers=[3, 4, 6, 3],

groups=32,

reduction=16,

dropout\_p=None,

inplanes=64,

input\_3x3=False,

downsample\_kernel\_size=1,

downsample\_padding=0,

last\_stride=last\_stride)

self.expansion = SEResNeXtBottleneck.expansion

elif model\_name == 'se\_resnext101':

self.base = SENet(block=SEResNeXtBottleneck,

layers=[3, 4, 23, 3],

groups=32,

reduction=16,

dropout\_p=None,

inplanes=64,

input\_3x3=False,

downsample\_kernel\_size=1,

downsample\_padding=0,

last\_stride=last\_stride)

self.expansion = SEResNeXtBottleneck.expansion

elif model\_name == 'senet154':

self.base = SENet(block=SEBottleneck,

layers=[3, 8, 36, 3],

groups=64,

reduction=16,

dropout\_p=0.2,

last\_stride=last\_stride)

self.expansion = SEResNeXtBottleneck.expansion

elif model\_name == 'convnet':

self.base = ConvNet()

self.in\_planes = 256

if pretrain\_choice == 'imagenet':

self.load\_param(model\_path)

print('Loading pretrained ImageNet model......')

def forward(self, x):

feat = self.base(x)

return feat

def load\_param(self, trained\_path):

param\_dict = torch.load(trained\_path)

for i in param\_dict:

if ('fc' in i) or ('classifier' in i):

continue

self.base.state\_dict()[i].copy\_(param\_dict[i])

./model/graph\_net.py

import torch

import torch.nn as nn

import torch.nn.functional as F

class Graph\_Net(nn.Module):

def \_\_init\_\_(self, in\_planes):

super(Graph\_Net, self).\_\_init\_\_()

self.in\_planes = in\_planes

self.out\_planes = in\_planes//2

# embedding

self.query\_encoder = nn.Conv2d(in\_planes, in\_planes//2, kernel\_size=1)

self.key\_encoder = nn.Conv2d(in\_planes, in\_planes//2, kernel\_size=1)

self.gcn\_weight = nn.Conv2d(in\_planes, in\_planes//2, kernel\_size=1)

self.gamma = nn.Parameter(torch.zeros(1))

# cross-modality graph

def build\_graph(self, q\_feat, k\_feat):

n, c, h, w = q\_feat.shape

q\_feat = q\_feat.contiguous().view(n, c, h \* w)

k\_feat = k\_feat.contiguous().view(n, c, h \* w)

# sparse graph

graph = torch.bmm(F.normalize(q\_feat, dim=1, p=2).permute(0, 2, 1), F.normalize(k\_feat, dim=1, p=2)).clamp(min=-1, max=1.)

graph = torch.exp(graph-1).pow(2)

mask = torch.bernoulli(graph.clamp(0, 1)).cuda()

graph = mask \* graph

D = (graph).sum(dim=-1).clamp(min=1e-12).pow(-1).diag\_embed()

return torch.bmm(D, graph)

def forward(self, x\_feat, y\_feat):

# f\_feat: n \* c \* h \* w

# b\_feat: n \* c \* h \* w

q\_feat = self.query\_encoder(y\_feat)

k\_feat = self.key\_encoder(x\_feat)

xw = self.gcn\_weight(x\_feat)

n, c, h, w = xw.shape

graph = self.build\_graph(q\_feat, k\_feat)

# generate final feature

xw = xw.contiguous().view(n, c, -1).permute(0, 2, 1)

graph\_feat = torch.bmm(graph, xw).permute(0, 2, 1).contiguous().view(n, c, h, w)

output = self.gamma \* graph\_feat + q\_feat

return output

./model/overall\_net.py

import torch

from torch import nn

import torch.nn.functional as F

from .baseline import Baseline

from .graph\_net import Graph\_Net

def weights\_init\_kaiming(m):

classname = m.\_\_class\_\_.\_\_name\_\_

if classname.find('Linear') != -1:

nn.init.kaiming\_normal\_(m.weight, a=0, mode='fan\_out')

nn.init.constant\_(m.bias, 0.0)

elif classname.find('Conv') != -1:

nn.init.kaiming\_normal\_(m.weight, a=0, mode='fan\_in')

if m.bias is not None:

nn.init.constant\_(m.bias, 0.0)

elif classname.find('BatchNorm') != -1:

if m.affine:

nn.init.constant\_(m.weight, 1.0)

nn.init.constant\_(m.bias, 0.0)

class Net(nn.Module):

def \_\_init\_\_(self, cfg, n\_classes):

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

# face and context encoder

self.f\_model = Baseline(cfg)

self.c\_model = Baseline(cfg)

# cross modality gcn layer

self.f\_graph\_module = Graph\_Net(self.f\_model.in\_planes)

self.c\_graph\_module = Graph\_Net(self.c\_model.in\_planes)

# avg pool => generate final features

self.gap = nn.AdaptiveAvgPool2d(1)

# batch norm

self.bottleneck = nn.BatchNorm1d(self.f\_model.in\_planes)

self.bottleneck.bias.requires\_grad\_(False)

self.bottleneck.apply(weights\_init\_kaiming)

# classification weights

self.weight = nn.Parameter(torch.FloatTensor(self.f\_model.in\_planes, n\_classes))

nn.init.xavier\_uniform\_(self.weight)

self.p\_m = 0.75

self.n\_m = 1 - self.p\_m

self.s = 20

def forward(self, face, context, label = None):

b = face.shape[0]

f\_feat = self.f\_model(face)

c\_feat = self.c\_model(context)

# context & face graph

f\_graph\_feat = self.gap(self.f\_graph\_module(f\_feat, c\_feat)).view(b, -1)

c\_graph\_feat = self.gap(self.c\_graph\_module(c\_feat, f\_feat)).view(b, -1)

graph\_feat = self.bottleneck(torch.cat([f\_graph\_feat, c\_graph\_feat], dim=1))

# predict => sphere mapping

output = torch.mm(F.normalize(graph\_feat, dim=-1),

F.normalize(self.weight, dim=0)).clamp(min=-1, max=1.)

if self.training and label is not None:

one\_hot = torch.zeros(output.shape, device='cuda')

one\_hot.scatter\_(1, label.view(-1, 1).long(), 1)

output = one\_hot \* (output - 0.1) + (1 - one\_hot) \* output

# boundary loss with orthogonal property

pos\_loss = torch.relu(self.p\_m - output[one\_hot.bool()].flatten())

neg\_loss = torch.relu(output[(1 - one\_hot).bool()].flatten() - self.n\_m)

restrict\_loss = torch.cat([pos\_loss, neg\_loss], dim=0).norm(p=2) / b

return output \* self.s, self.gap(f\_feat).view(b, -1), self.gap(c\_feat).view(b, -1), restrict\_loss

else:

return output

./model/backbones/\_\_init\_\_.py (empty file)

./model/backbones/convnet.py

import torch

import torch.nn as nn

import torch.nn.functional as F

import math

class Block(nn.Module):

def \_\_init\_\_(self, in\_planes, out\_planes, stride=1, padding = 1):

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

self.conv = nn.Conv2d(in\_planes, out\_planes, kernel\_size=3, stride=stride, padding=padding, bias=False)

self.bn = nn.BatchNorm2d(out\_planes)

self.relu = nn.ReLU(inplace=True)

def forward(self, x):

x = self.conv(x)

x = self.bn(x)

x = self.relu(x)

return x

class ConvNet(nn.Module):

def \_\_init\_\_(self):

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

self.pool = nn.MaxPool2d(2, 2)

self.block\_1 = Block(3, 32, 1, 1)

self.block\_2 = Block(32, 64, 1, 1)

self.block\_3 = Block(64, 128, 1, 1)

self.block\_4 = Block(128, 256, 1, 1)

self.block\_5 = Block(256, 256, 1, 1)

self.random\_init()

def forward(self, x):

x = self.block\_1(x)

x = self.pool(x)

x = self.block\_2(x)

x = self.pool(x)

x = self.block\_3(x)

x = self.pool(x)

x = self.block\_4(x)

x = self.pool(x)

x = self.block\_5(x)

return x

def random\_init(self):

for m in self.modules():

if isinstance(m, nn.Conv2d):

n = m.kernel\_size[0] \* m.kernel\_size[1] \* m.out\_channels

m.weight.data.normal\_(0, math.sqrt(2. / n))

elif isinstance(m, nn.BatchNorm2d):

m.weight.data.fill\_(1)

m.bias.data.zero\_()

./build\_train\_list.py

import os

def main():

path = './data/CAER/CAER/CAER/'

for type\_dir in os.listdir(path):

# train, validation, test

txt\_file = path + type\_dir + '.txt'

type\_dir = os.path.join(path, type\_dir)

for label\_dir in os.listdir(type\_dir):

# Anger, Disgust, Fear, Happy, Neutral, Sad, Suprise

label = label\_dir

label\_dir = os.path.join(type\_dir, label\_dir)

for avi\_file in os.listdir(label\_dir):

avi\_file = os.path.join(label\_dir, avi\_file)

f = open(txt\_file, 'a')

f.write(avi\_file + ' ' + label + '\n')

f.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

./det\_face.py

import os

import os.path as osp

import shutil

import face\_alignment

from PIL import Image

import numpy as np

import copy

if \_\_name\_\_ == "\_\_main\_\_":

# cuda for CUDA

fa = face\_alignment.FaceAlignment(face\_alignment.LandmarksType.\_2D, device='cuda')

org\_path = './data/CAER/CAER-S/CAER-S'

tgt\_path = './data/CAER/CAER-S/CAER-S-FACE'

log\_file = './log.txt'

if os.path.isdir(tgt\_path):

shutil.rmtree(tgt\_path)

os.mkdir(tgt\_path)

for data\_type in os.listdir(org\_path):

org\_data\_dir = osp.join(org\_path, data\_type)

tgt\_data\_dir = osp.join(tgt\_path, data\_type)

os.mkdir(tgt\_data\_dir)

for label\_dir in os.listdir(org\_data\_dir):

org\_label\_dir = osp.join(org\_data\_dir, label\_dir)

tgt\_label\_dir = osp.join(tgt\_data\_dir, label\_dir)

os.mkdir(tgt\_label\_dir)

for image\_file in os.listdir(org\_label\_dir):

image = Image.open(osp.join(org\_label\_dir, image\_file))

np\_image = np.array(image)

# det face

preds = fa.get\_landmarks(np\_image)

max\_area = 0

max\_pred = []

if preds is not None:

for i, pred in enumerate(preds):

xmin = int(min(pred[:, 0]))

xmax = int(max(pred[:, 0]))

ymin = int(min(pred[:, 1]))

ymax = int(max(pred[:, 1]))

w = xmax - xmin

h = ymax - ymin

b\_size = max(w, h)

mid\_x = (xmax + xmin) /2

mid\_y = (ymax + ymin) /2

area = w\*h

if area > max\_area:

max\_area = area

max\_pred = [mid\_x - b\_size\*0.5, mid\_y - b\_size\*0.5, mid\_x + b\_size\*0.5, mid\_y + b\_size\*0.5]

image.crop(max\_pred).save(osp.join(tgt\_label\_dir, image\_file))

else:

f = open(log\_file, "a")

f.write(osp.join(org\_label\_dir, image\_file) + '\n')

f.close()

./inference.py

import copy

import os

import time

from collections import Counter, deque

import cv2

import face\_alignment

import numpy as np

import torch

import torch.nn.functional as F

import torchvision

from imutils import face\_utils

from PIL import Image

from scipy.spatial import distance as dist

from scipy.stats import entropy

from torchvision import transforms, utils

import util

from configs.image\_cfg import \_C as cfg

from dataset import make\_data\_loader

from dataset.transforms import build\_transforms

from model.graph\_net import Graph\_Net

from model.overall\_net import Net

def main():

os.environ['CUDA\_VISIBLE\_DEVICES'] = cfg.MODEL.DEVICE\_ID

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

torch.backends.cudnn.benchmark = True

# train\_loader, test\_loader, num\_classes, label\_template = make\_data\_loader(cfg)

label\_template = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

pos\_template = ['Happy', 'Surprise']

neg\_template = ['Angry', 'Disgust', 'Fear', 'Sad']

neutral\_class = 'Neutral'

num\_classes = len(label\_template)

# define and load model

model\_path = cfg.MODEL.SAVE\_WEIGHT\_PATH+'.pth'

model = Net(cfg, num\_classes).to(device)

model.load\_state\_dict(torch.load(model\_path))

model.eval()

# face detector

fa = face\_alignment.FaceAlignment(face\_alignment.LandmarksType.\_2D, flip\_input=False, device='cuda')

(lBegin, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eyebrow"]

(rBegin, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eyebrow"]

f\_transform = build\_transforms(cfg, is\_train=False, is\_face=True)

c\_transform = build\_transforms(cfg, is\_train=False, is\_face=False)

# video path

video\_path = './test.mp4'

cap = cv2.VideoCapture(video\_path)

f\_stack = deque([])

c\_stack = deque([])

seq\_pred = deque([])

points = deque([])

seq\_arousal\_val = deque([])

long\_seq\_pred = deque([])

long\_seq\_prob = deque([])

long\_seq\_size = 64

window\_size = 16

if (cap.isOpened()== False):

print("Error opening video stream or file")

while(cap.isOpened()):

ret, frame = cap.read()

if ret == True:

if cv2.waitKey(25) & 0xFF == ord('q'):

break

frame = cv2.resize(frame,None,fx=0.5,fy=0.5,interpolation=cv2.INTER\_CUBIC)

# cv2.imwrite('./test.png', frame)

# print(Image.open('./test.png').size, torchvision.io.read\_image('./test.png').shape, frame.shape)

# assert i == -1

# cv2.imshow('Frame', frame)

# cv2.putText(frame, label\_table[np.argmax(emotion\_count)], (max\_face\_info[0], int(max\_face\_info[2] - 5)), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 4)

face, context, landmark = detFace(fa, frame)

collectBatch(face, f\_transform, f\_stack, window\_size)

collectBatch(context, c\_transform, c\_stack, window\_size)

leyebrow = landmark[lBegin:lEnd]

reyebrow = landmark[rBegin:rEnd]

distq = eye\_brow\_distance(points,window\_size,leyebrow[-1],reyebrow[0])

seq\_arousal\_val.append(normalize\_values(points,distq))

if len(seq\_arousal\_val) > window\_size:

seq\_arousal\_val.popleft()

if len(f\_stack) == window\_size:

with torch.no\_grad():

f\_tensor = torch.stack(list(copy.deepcopy(f\_stack))).to(device)

c\_tensor = torch.stack(list(copy.deepcopy(c\_stack))).to(device)

output = model(f\_tensor, c\_tensor).mean(0)

pred\_class = output.argmax()

pred\_label = label\_template[pred\_class]

arousal\_label = ''

arousal\_value = np.mean(seq\_arousal\_val)

# if arousal\_value > 0.75:

# arousal\_label = 'high arousal'

# else:

# arousal\_label = 'low arousal'

# print(label\_template[pred\_class], np.mean(seq\_arousal\_val))

det\_disorder = "normal"

long\_seq\_pred.append(pred\_label)

long\_seq\_prob.append(output.cpu().numpy())

text\_color = (0, 255, 0)

if len(long\_seq\_pred) > long\_seq\_size:

long\_seq\_pred.popleft()

long\_seq\_prob.popleft()

mean\_pred = np.mean(long\_seq\_prob, 0)

mean\_prob = prob(mean\_pred)

# count prediction

count\_window\_pred = Counter(long\_seq\_pred)

count\_window\_pred\_list = []

for val in count\_window\_pred:

count\_window\_pred\_list.append(count\_window\_pred[val])

count\_window\_pred\_list = np.array(count\_window\_pred\_list, dtype=np.float) / long\_seq\_size

max\_count\_pred = count\_window\_pred.most\_common(1)[0]

disorder = False

if entropy(count\_window\_pred\_list) > 0.9 or arousal\_value > 0.75:

disorder = True

if disorder:

det\_disorder = "abnormal"

text\_color = (0, 0, 255)

# assert i == -1

# if max\_count\_pred[1] > (long\_seq\_size // 2) and max\_count\_pred[0] != pred\_label:

# det\_disorder = "abnormal"

# text\_color = (0, 0, 255)

# print(count\_window\_pred, max\_count\_pred)

# assert i == -1

cv2.putText(frame, pred\_label,

(20, 30),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 4)

cv2.putText(frame, 'arousal => {:.4f}'.format(arousal\_value),

(20, 70),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 4)

cv2.putText(frame, det\_disorder,

(20, 110),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, text\_color, 4)

# show\_face = cv2.cvtColor(face, cv2.COLOR\_RGB2BGR)

cv2.imshow('frame', frame)

else:

break

cap.release()

cv2.destroyAllWindows()

def prob(log\_output):

return np.exp(log\_output)/sum(np.exp(log\_output))

def eye\_brow\_distance(points, size, leye, reye):

distq = dist.euclidean(leye,reye)

points.append(int(distq))

if len(points) > size:

points.popleft()

return distq

def normalize\_values(points, disp):

# print(points)

normalized\_value = abs(disp - np.min(points))/abs(np.max(points) - np.min(points))

arousal\_value = np.exp(-(normalized\_value))

return arousal\_value

def collectBatch(image, transform, stack, size):

tensor = transform(Image.fromarray(image))

stack.append(tensor)

if len(stack) > size:

stack.popleft()

def detFace(model, image):

process\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

w, h, c = process\_image.shape

preds = model.get\_landmarks(process\_image)

if preds != None and len(preds) > 0:

max\_face\_area = 0

max\_face\_info = [0, 0, 0, 0]

max\_landmark = None

try:

for i, landmark in enumerate(preds):

x1 = np.min(landmark[:, 0])

y1 = np.min(landmark[:, 1])

x2 = np.max(landmark[:, 0])

y2 = np.max(landmark[:, 1])

box\_len = int((float(max(x2 - x1, y2 - y1)) / 2.) \* 1.3)

center\_x = int(float(x1 + x2) / 2.)

center\_y = int(float(y1 + y2) / 2.)

if box\_len > max\_face\_area:

max\_face\_area = box\_len

max\_face\_info = [center\_x, center\_y]

max\_landmark = landmark

sx = max\_face\_info[0] - max\_face\_area

ex = max\_face\_info[0] + max\_face\_area

sy = max\_face\_info[1] - max\_face\_area

ey = max\_face\_info[1] + max\_face\_area

face = process\_image[sx:ex, sy:ey, :]

wf, hf, cf = face.shape

result = np.full((max\_face\_area \* 2,max\_face\_area \* 2, c), (0,0,0), dtype=np.uint8)

xx = (max\_face\_area \* 2 - wf) // 2

yy = (max\_face\_area \* 2 - hf) // 2

result[xx:xx+wf, yy:yy+hf] = process\_image[sx:ex, sy:ey, :]

return result, process\_image, landmark

except Exception as e:

print(e)

mid\_w = w // 2

mid\_h = h // 2

box\_len = min(mid\_h, mid\_w) // 2

return process\_image[(mid\_w - box\_len):(mid\_w + box\_len), (mid\_h - box\_len):(mid\_h + box\_len), :], process\_image, None

if \_\_name\_\_ == "\_\_main\_\_":

main()

./train.py

import os

import torch

import torch.nn.functional as F

import util

from configs.image\_cfg import \_C as cfg

from dataset import make\_data\_loader

from model.graph\_net import Graph\_Net

from model.overall\_net import Net

def main():

os.environ['CUDA\_VISIBLE\_DEVICES'] = cfg.MODEL.DEVICE\_ID

device = torch.device(cfg.MODEL.DEVICE)

# define DataLoader

train\_loader, test\_loader, num\_classes, label\_template = make\_data\_loader(cfg)

# define DNN

model = Net(cfg, num\_classes).to(device)

# define optimizer

opt = torch.optim.Adam([{'params': model.parameters()},],

lr=cfg.OPTIMIZER.LR,

weight\_decay=cfg.OPTIMIZER.WEIGHT\_DECAY)

# define learning rate scheduler

scheduler = torch.optim.lr\_scheduler.MultiStepLR(opt, milestones=cfg.OPTIMIZER.STEP, gamma=cfg.OPTIMIZER.GAMMA, last\_epoch=-1)

util.logger(cfg.MODEL.SAVE\_WEIGHT\_PATH + '\_' + cfg.MODEL.SAVE\_TRAIN\_INFO, 'start training!', 'w')

# run epoch

max\_acc = 0

for epoch in range(cfg.OPTIMIZER.EPOCH):

epoch = epoch + 1

# run training

train\_loss, train\_acc = run(device, train\_loader, model, opt)

print('avg\_loss: {:.4f}, avg\_acc: {:.4f}'.format(train\_loss, train\_acc))

if epoch % cfg.OPTIMIZER.TEST\_PER == 0:

# run testing

test\_acc = eval(device, test\_loader, model)

print('epoch: {:3d}, accuracy: {:.4f}'.format(epoch, test\_acc))

info = 'epoch: {:4d} => train loss: {:.4f}, train accuracy: {:.4f} | test accuracy: {:.4f}'.format(epoch, train\_loss, train\_acc, test\_acc)

if test\_acc > max\_acc:

max\_acc = test\_acc

else:

info = 'epoch: {:4d} => train loss: {:.4f}, train accuracy: {:.4f}'.format(epoch, train\_loss, train\_acc)

util.logger(cfg.MODEL.SAVE\_WEIGHT\_PATH + '\_' + cfg.MODEL.SAVE\_TRAIN\_INFO, info)

scheduler.step()

print('best accuracy: {:.4f}'.format(max\_acc))

util.logger(cfg.MODEL.SAVE\_WEIGHT\_PATH + '\_' + cfg.MODEL.SAVE\_TRAIN\_INFO, 'best accuracy: {:.4f}'.format(max\_acc), 'a')

torch.save(model.state\_dict(), cfg.MODEL.SAVE\_WEIGHT\_PATH+'.pth')

def run(device, loader, model, opt):

# len of dataloader

iter = len(loader)

# average loss and acc of each epoch

avg\_loss = 0.

avg\_acc = 0.

# set training mode

model.train()

for batch\_idx, batch\_data in enumerate(loader):

# set face and context image into cuda.tensor

fimages = batch\_data[0].to(device)

cimages = batch\_data[1].to(device)

labels = batch\_data[2].to(device)

# zero gradients

opt.zero\_grad()

# input face and context into model

output, f\_feat, c\_feat, restrict\_loss = model(fimages, cimages, labels)

# cross entropy loss

ce\_loss = F.cross\_entropy(output, labels)

# loss for backpropagation

loss = ce\_loss + restrict\_loss

# calculate gradient

loss.backward()

# update model

opt.step()

# accuracy

acc = (output.max(1)[1] == labels).float().mean()

if batch\_idx % 50 == 0:

print('batch idx: {:3d}, restrict\_loss: {:.4f}, ce\_loss: {:.4f}, loss: {:.4f}, accuracy: {:.4f}'.format(batch\_idx, restrict\_loss.item(), ce\_loss.item(), loss.item(), acc.item()))

avg\_loss += loss.item()

avg\_acc += acc.item()

avg\_loss /= iter

avg\_acc /= iter

return avg\_loss, avg\_acc

def eval(device, loader, model):

# len of dataloader

iter = len(loader)

# average acc of each epoch

avg\_acc = 0.

# set testing mode

model.eval()

with torch.no\_grad():

for batch\_idx, batch\_data in enumerate(loader):

# set face and context image into cuda.tensor

fimages = batch\_data[0].to(device)

cimages = batch\_data[1].to(device)

labels = batch\_data[2].to(device)

# input face and context into model

output = model(fimages, cimages)

# calculate accuracy

acc = (output.max(1)[1] == labels).float().mean()

avg\_acc += acc.item()

avg\_acc /= iter

return avg\_acc

if \_\_name\_\_ == "\_\_main\_\_":

main()

./util.py

import os

import shutil

def logger(txt, log\_info, mode = 'a'):

# print(log\_info)

f = open(txt, mode)

f.write(log\_info + '\n')

f.close()

def make\_dir(\_path, \_is\_del = True):

if \_path[-1] != '/':

\_path = \_path + '/'

if os.path.isdir(\_path):

if \_is\_del == True:

shutil.rmtree(\_path)

os.mkdir(\_path)

else:

if not os.path.exists(os.path.dirname(\_path)):

os.makedirs(os.path.dirname(\_path))

return \_path