Train.py:# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import print\_function, division

import argparse

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

from torch.optim import lr\_scheduler

from torch.autograd import Variable

from torchvision import datasets, transforms

from folder import ImageFolder

import torch.backends.cudnn as cudnn

import matplotlib

matplotlib.use('agg')

import matplotlib.pyplot as plt

# from PIL import Image

import copy

import time

import os

from model import two\_view\_net, three\_view\_net

from random\_erasing import RandomErasing

from autoaugment import ImageNetPolicy, CIFAR10Policy

import yaml

from shutil import copyfile

from utils import update\_average, get\_model\_list, load\_network, save\_network, make\_weights\_for\_balanced\_classes

from pytorch\_metric\_learning import losses, miners # pip install pytorch-metric-learning

from circle\_loss import CircleLoss, convert\_label\_to\_similarity

version = torch.\_\_version\_\_

# fp16

try:

from apex.fp16\_utils import \*

from apex import amp, optimizers

except ImportError: # will be 3.x series

print(

'This is not an error. If you want to use low precision, i.e., fp16, please install the apex with cuda support (https://github.com/NVIDIA/apex) and update pytorch to 1.0')

######################################################################

# Options

# --------

parser = argparse.ArgumentParser(description='Training')

parser.add\_argument('--gpu\_ids', default='0', type=str, help='gpu\_ids: e.g. 0 0,1,2 0,2')

parser.add\_argument('--name', default='two\_view', type=str, help='output model name')

parser.add\_argument('--resume', action='store\_true', help='use resume trainning')

#data

parser.add\_argument('--data\_dir', default='./data/train', type=str, help='training dir path')

parser.add\_argument('--extra\_Google', action='store\_true', help='using extra noise Google')

parser.add\_argument('--train\_all', action='store\_true', help='use all training data')

parser.add\_argument('--color\_jitter', action='store\_true', help='use color jitter in training')

parser.add\_argument('--batchsize', default=8, type=int, help='batchsize')

parser.add\_argument('--pad', default=10, type=int, help='padding')

parser.add\_argument('--h', default=384, type=int, help='height')

parser.add\_argument('--w', default=384, type=int, help='width')

parser.add\_argument('--views', default=2, type=int, help='the number of views')

parser.add\_argument('--erasing\_p', default=0, type=float, help='Random Erasing probability, in [0,1]')

parser.add\_argument('--DA', action='store\_true', help='use Color Data Augmentation')

#backbone

parser.add\_argument('--share', action='store\_true', help='share weight between different view')

parser.add\_argument('--droprate', default=0.5, type=float, help='drop rate')

parser.add\_argument('--pool', default='avg', type=str, help='pool avg')

parser.add\_argument('--stride', default=2, type=int, help='stride')

parser.add\_argument('--use\_dense', action='store\_true', help='use densenet121')

parser.add\_argument('--use\_NAS', action='store\_true', help='use NAS')

#optimizer

parser.add\_argument('--warm\_epoch', default=0, type=int, help='the first K epoch that needs warm up')

parser.add\_argument('--lr', default=0.01, type=float, help='learning rate')

parser.add\_argument('--moving\_avg', default=1.0, type=float, help='moving average')

parser.add\_argument('--fp16', action='store\_true',

help='use float16 instead of float32, which will save about 50% memory')

# extra losses (default is cross-entropy loss. You can fuse different losses for further performance boost.)

parser.add\_argument('--arcface', action='store\_true', help='use ArcFace loss')

parser.add\_argument('--circle', action='store\_true', help='use Circle loss')

parser.add\_argument('--cosface', action='store\_true', help='use CosFace loss')

parser.add\_argument('--contrast', action='store\_true', help='use contrast loss')

parser.add\_argument('--triplet', action='store\_true', help='use triplet loss')

parser.add\_argument('--lifted', action='store\_true', help='use lifted loss')

parser.add\_argument('--sphere', action='store\_true', help='use sphere loss')

parser.add\_argument('--loss\_merge', action='store\_true', help='combine perspectives to calculate losses')

opt = parser.parse\_args()

if opt.resume:

model, opt, start\_epoch = load\_network(opt.name, opt)

else:

start\_epoch = 0

fp16 = opt.fp16

data\_dir = opt.data\_dir

name = opt.name

str\_ids = opt.gpu\_ids.split(',')

gpu\_ids = []

for str\_id in str\_ids:

gid = int(str\_id)

if gid >= 0:

gpu\_ids.append(gid)

# set gpu ids

if len(gpu\_ids) > 0:

torch.cuda.set\_device(gpu\_ids[0])

cudnn.benchmark = True

######################################################################

# Load Data

# ---------

#

transform\_train\_list = [

# transforms.RandomResizedCrop(size=(opt.h, opt.w), scale=(0.75,1.0), ratio=(0.75,1.3333), interpolation=3), #Image.BICUBIC)

transforms.Resize((opt.h, opt.w), interpolation=3),

transforms.Pad(opt.pad, padding\_mode='edge'),

transforms.RandomCrop((opt.h, opt.w)),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]

transform\_satellite\_list = [

transforms.Resize((opt.h, opt.w), interpolation=3),

transforms.Pad(opt.pad, padding\_mode='edge'),

transforms.RandomAffine(90),

transforms.RandomCrop((opt.h, opt.w)),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]

transform\_val\_list = [

transforms.Resize(size=(opt.h, opt.w), interpolation=3), # Image.BICUBIC

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]

if opt.erasing\_p > 0:

transform\_train\_list = transform\_train\_list + [RandomErasing(probability=opt.erasing\_p, mean=[0.0, 0.0, 0.0])]

if opt.color\_jitter:

transform\_train\_list = [transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1,

hue=0)] + transform\_train\_list

transform\_satellite\_list = [transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1,

hue=0)] + transform\_satellite\_list

if opt.DA:

transform\_train\_list = [ImageNetPolicy()] + transform\_train\_list

print(transform\_train\_list)

data\_transforms = {

'train': transforms.Compose(transform\_train\_list),

'val': transforms.Compose(transform\_val\_list),

'satellite': transforms.Compose(transform\_satellite\_list)}

train\_all = ''

if opt.train\_all:

train\_all = '\_all'

image\_datasets = {}

image\_datasets['satellite'] = datasets.ImageFolder(os.path.join(data\_dir, 'satellite'),

data\_transforms['satellite'])

image\_datasets['street'] = datasets.ImageFolder(os.path.join(data\_dir, 'street'),

data\_transforms['train'])

image\_datasets['drone'] = datasets.ImageFolder(os.path.join(data\_dir, 'drone'),

data\_transforms['train'])

image\_datasets['google'] = ImageFolder(os.path.join(data\_dir, 'google'),

# google contain empty subfolder, so we overwrite the Folder

data\_transforms['train'])

dataloaders = {x: torch.utils.data.DataLoader(image\_datasets[x], batch\_size=opt.batchsize,

shuffle=True, num\_workers=2, pin\_memory=True) # 8 workers may work faster

for x in ['satellite', 'street', 'drone', 'google']}

dataset\_sizes = {x: len(image\_datasets[x]) for x in ['satellite', 'street', 'drone', 'google']}

class\_names = image\_datasets['street'].classes

print(dataset\_sizes)

use\_gpu = torch.cuda.is\_available()

######################################################################

# Training the model

# ------------------

#

# Now, let's write a general function to train a model. Here, we will

# illustrate:

#

# - Scheduling the learning rate

# - Saving the best model

#

# In the following, parameter ``scheduler`` is an LR scheduler object from

# ``torch.optim.lr\_scheduler``.

y\_loss = {} # loss history

y\_loss['train'] = []

y\_loss['val'] = []

y\_err = {}

y\_err['train'] = []

y\_err['val'] = []

def train\_model(model, model\_test, criterion, optimizer, scheduler, num\_epochs=25):

since = time.time()

# best\_model\_wts = model.state\_dict()

# best\_acc = 0.0

warm\_up = 0.1 # We start from the 0.1\*lrRate

warm\_iteration = round(dataset\_sizes['satellite'] / opt.batchsize) \* opt.warm\_epoch # first 5 epoch

if opt.arcface:

criterion\_arcface = losses.ArcFaceLoss(num\_classes=opt.nclasses, embedding\_size=512)

if opt.cosface:

criterion\_cosface = losses.CosFaceLoss(num\_classes=opt.nclasses, embedding\_size=512)

if opt.circle:

criterion\_circle = CircleLoss(m=0.25, gamma=32) # gamma = 64 may lead to a better result.

if opt.triplet:

miner = miners.MultiSimilarityMiner()

criterion\_triplet = losses.TripletMarginLoss(margin=0.3)

if opt.lifted:

criterion\_lifted = losses.GeneralizedLiftedStructureLoss(neg\_margin=1, pos\_margin=0)

if opt.contrast:

criterion\_contrast = losses.ContrastiveLoss(pos\_margin=0, neg\_margin=1)

if opt.sphere:

criterion\_sphere = losses.SphereFaceLoss(num\_classes=opt.nclasses, embedding\_size=512, margin=4)

for epoch in range(num\_epochs - start\_epoch):

epoch = epoch + start\_epoch

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

print('-' \* 10)

# Each epoch has a training and validation phase

for phase in ['train']:

if phase == 'train':

model.train(True) # Set model to training mode

else:

model.train(False) # Set model to evaluate mode

running\_loss = 0.0

running\_corrects = 0.0

running\_corrects2 = 0.0

running\_corrects3 = 0.0

# Iterate over data.

for data, data2, data3, data4 in zip(dataloaders['satellite'], dataloaders['street'], dataloaders['drone'],

dataloaders['google']):

# get the inputs

inputs, labels = data

inputs2, labels2 = data2

inputs3, labels3 = data3

inputs4, labels4 = data4

now\_batch\_size, c, h, w = inputs.shape

if now\_batch\_size < opt.batchsize: # skip the last batch

continue

if use\_gpu:

inputs = Variable(inputs.cuda().detach())

inputs2 = Variable(inputs2.cuda().detach())

inputs3 = Variable(inputs3.cuda().detach())

labels = Variable(labels.cuda().detach())

labels2 = Variable(labels2.cuda().detach())

labels3 = Variable(labels3.cuda().detach())

if opt.extra\_Google:

inputs4 = Variable(inputs4.cuda().detach())

labels4 = Variable(labels4.cuda().detach())

else:

inputs, labels = Variable(inputs), Variable(labels)

# zero the parameter gradients

optimizer.zero\_grad()

# forward

if phase == 'val':

with torch.no\_grad():

outputs, outputs2 = model(inputs, inputs2)

else:

if opt.views == 2:

outputs, outputs2 = model(inputs, inputs2)

elif opt.views == 3:

if opt.extra\_Google:

outputs, outputs2, outputs3, outputs4 = model(inputs, inputs2, inputs3, inputs4)

else:

outputs, outputs2, outputs3 = model(inputs, inputs2, inputs3)

return\_feature = opt.arcface or opt.cosface or opt.circle or opt.triplet or opt.contrast or opt.lifted or opt.sphere

if opt.views == 2:

\_, preds = torch.max(outputs.data, 1)

\_, preds2 = torch.max(outputs2.data, 1)

loss = criterion(outputs, labels) + criterion(outputs2, labels2)

elif opt.views == 3:

if return\_feature:

logits, ff = outputs

logits2, ff2 = outputs2

logits3, ff3 = outputs3

fnorm = torch.norm(ff, p=2, dim=1, keepdim=True)

fnorm2 = torch.norm(ff2, p=2, dim=1, keepdim=True)

fnorm3 = torch.norm(ff3, p=2, dim=1, keepdim=True)

ff = ff.div(fnorm.expand\_as(ff)) # 8\*512,tensor

ff2 = ff2.div(fnorm2.expand\_as(ff2))

ff3 = ff3.div(fnorm3.expand\_as(ff3))

loss = criterion(logits, labels) + criterion(logits2, labels2) + criterion(logits3, labels3)

\_, preds = torch.max(logits.data, 1)

\_, preds2 = torch.max(logits2.data, 1)

\_, preds3 = torch.max(logits3.data, 1)

# Multiple perspectives are combined to calculate losses, please join ''--loss\_merge'' in run.sh

if opt.loss\_merge:

ff\_all = torch.cat((ff, ff2, ff3), dim=0)

labels\_all = torch.cat((labels, labels2, labels3), dim=0)

if opt.extra\_Google:

logits4, ff4 = outputs4

fnorm4 = torch.norm(ff4, p=2, dim=1, keepdim=True)

ff4 = ff4.div(fnorm4.expand\_as(ff4))

loss = criterion(logits, labels) + criterion(logits2, labels2) + criterion(logits3, labels3) +criterion(logits4, labels4)

if opt.loss\_merge:

ff\_all = torch.cat((ff\_all, ff4), dim=0)

labels\_all = torch.cat((labels\_all, labels4), dim=0)

if opt.arcface:

if opt.loss\_merge:

loss += criterion\_arcface(ff\_all, labels\_all)

else:

loss += criterion\_arcface(ff, labels) + criterion\_arcface(ff2, labels2) + criterion\_arcface(ff3, labels3) # /now\_batch\_size

if opt.extra\_Google:

loss += criterion\_arcface(ff4, labels4) # /now\_batch\_size

if opt.cosface:

if opt.loss\_merge:

loss += criterion\_cosface(ff\_all, labels\_all)

else:

loss += criterion\_cosface(ff, labels) + criterion\_cosface(ff2, labels2) + criterion\_cosface(ff3, labels3) # /now\_batch\_size

if opt.extra\_Google:

loss += criterion\_cosface(ff4, labels4) # /now\_batch\_size

if opt.circle:

if opt.loss\_merge:

loss += criterion\_circle(\*convert\_label\_to\_similarity(ff\_all, labels\_all)) / now\_batch\_size

else:

loss += criterion\_circle(\*convert\_label\_to\_similarity(ff, labels)) / now\_batch\_size + criterion\_circle(\*convert\_label\_to\_similarity(ff2, labels2)) / now\_batch\_size + criterion\_circle(\*convert\_label\_to\_similarity(ff3, labels3)) / now\_batch\_size

if opt.extra\_Google:

loss += criterion\_circle(\*convert\_label\_to\_similarity(ff4, labels4)) / now\_batch\_size

if opt.triplet:

if opt.loss\_merge:

hard\_pairs\_all = miner(ff\_all, labels\_all)

loss += criterion\_triplet(ff\_all, labels\_all, hard\_pairs\_all)

else:

hard\_pairs = miner(ff, labels)

hard\_pairs2 = miner(ff2, labels2)

hard\_pairs3 = miner(ff3, labels3)

loss += criterion\_triplet(ff, labels, hard\_pairs) + criterion\_triplet(ff2, labels2, hard\_pairs2) + criterion\_triplet(ff3, labels3, hard\_pairs3)# /now\_batch\_size

if opt.extra\_Google:

hard\_pairs4 = miner(ff4, labels4)

loss += criterion\_triplet(ff4, labels4, hard\_pairs4)

if opt.lifted:

if opt.loss\_merge:

loss += criterion\_lifted(ff\_all, labels\_all)

else:

loss += criterion\_lifted(ff, labels) + criterion\_lifted(ff2, labels2) + criterion\_lifted(ff3, labels3) # /now\_batch\_size

if opt.extra\_Google:

loss += criterion\_lifted(ff4, labels4)

if opt.contrast:

if opt.loss\_merge:

loss += criterion\_contrast(ff\_all, labels\_all)

else:

loss += criterion\_contrast(ff, labels) + criterion\_contrast(ff2,labels2) + criterion\_contrast(ff3, labels3) # /now\_batch\_size

if opt.extra\_Google:

loss += criterion\_contrast(ff4, labels4)

if opt.sphere:

if opt.loss\_merge:

loss += criterion\_sphere(ff\_all, labels\_all) / now\_batch\_size

else:

loss += criterion\_sphere(ff, labels) / now\_batch\_size + criterion\_sphere(ff2, labels2) / now\_batch\_size + criterion\_sphere(ff3, labels3) / now\_batch\_size

if opt.extra\_Google:

loss += criterion\_sphere(ff4, labels4)

else:

\_, preds = torch.max(outputs.data, 1)

\_, preds2 = torch.max(outputs2.data, 1)

\_, preds3 = torch.max(outputs3.data, 1)

if opt.loss\_merge:

outputs\_all = torch.cat((outputs, outputs2, outputs3), dim=0)

labels\_all = torch.cat((labels, labels2, labels3), dim=0)

if opt.extra\_Google:

outputs\_all = torch.cat((outputs\_all, outputs4), dim=0)

labels\_all = torch.cat((labels\_all, labels4), dim=0)

loss = 4\*criterion(outputs\_all, labels\_all)

else:

loss = criterion(outputs, labels) + criterion(outputs2, labels2) + criterion(outputs3, labels3)

if opt.extra\_Google:

loss += criterion(outputs4, labels4)

# backward + optimize only if in training phase

if epoch < opt.warm\_epoch and phase == 'train':

warm\_up = min(1.0, warm\_up + 0.9 / warm\_iteration)

loss \*= warm\_up

if phase == 'train':

if fp16: # we use optimier to backward loss

with amp.scale\_loss(loss, optimizer) as scaled\_loss:

scaled\_loss.backward()

else:

loss.backward()

optimizer.step()

##########

if opt.moving\_avg < 1.0:

update\_average(model\_test, model, opt.moving\_avg)

# statistics

if int(version[0]) > 0 or int(version[2]) > 3: # for the new version like 0.4.0, 0.5.0 and 1.0.0

running\_loss += loss.item() \* now\_batch\_size

else: # for the old version like 0.3.0 and 0.3.1

running\_loss += loss.data[0] \* now\_batch\_size

running\_corrects += float(torch.sum(preds == labels.data))

running\_corrects2 += float(torch.sum(preds2 == labels2.data))

if opt.views == 3:

running\_corrects3 += float(torch.sum(preds3 == labels3.data))

epoch\_loss = running\_loss / dataset\_sizes['satellite']

epoch\_acc = running\_corrects / dataset\_sizes['satellite']

epoch\_acc2 = running\_corrects2 / dataset\_sizes['satellite']

if opt.views == 2:

print('{} Loss: {:.4f} Satellite\_Acc: {:.4f} Street\_Acc: {:.4f}'.format(phase, epoch\_loss, epoch\_acc,

epoch\_acc2))

elif opt.views == 3:

epoch\_acc3 = running\_corrects3 / dataset\_sizes['satellite']

print('{} Loss: {:.4f} Satellite\_Acc: {:.4f} Street\_Acc: {:.4f} Drone\_Acc: {:.4f}'.format(phase,

epoch\_loss,

epoch\_acc,

epoch\_acc2,

epoch\_acc3))

y\_loss[phase].append(epoch\_loss)

y\_err[phase].append(1.0 - epoch\_acc)

# deep copy the model

if phase == 'train':

scheduler.step()

last\_model\_wts = model.state\_dict()

if epoch % 20 == 19:

save\_network(model, opt.name, epoch)

# draw\_curve(epoch)

time\_elapsed = time.time() - since

print('Training complete in {:.0f}m {:.0f}s'.format(

time\_elapsed // 60, time\_elapsed % 60))

print()

time\_elapsed = time.time() - since

print('Training complete in {:.0f}m {:.0f}s'.format(

time\_elapsed // 60, time\_elapsed % 60))

# print('Best val Acc: {:4f}'.format(best\_acc))

# save\_network(model\_test, opt.name+'adapt', epoch)

return model

######################################################################

# Draw Curve

# ---------------------------

x\_epoch = []

fig = plt.figure()

ax0 = fig.add\_subplot(121, title="loss")

ax1 = fig.add\_subplot(122, title="top1err")

def draw\_curve(current\_epoch):

x\_epoch.append(current\_epoch)

ax0.plot(x\_epoch, y\_loss['train'], 'bo-', label='train')

ax0.plot(x\_epoch, y\_loss['val'], 'ro-', label='val')

ax1.plot(x\_epoch, y\_err['train'], 'bo-', label='train')

ax1.plot(x\_epoch, y\_err['val'], 'ro-', label='val')

if current\_epoch == 0:

ax0.legend()

ax1.legend()

fig.savefig(os.path.join('./model', name, 'train.jpg'))

######################################################################

# Finetuning the convnet

# ----------------------

#

# Load a pretrainied model and reset final fully connected layer.

#

return\_feature = opt.arcface or opt.cosface or opt.circle or opt.triplet or opt.contrast or opt.lifted or opt.sphere

if opt.views == 2:

model = two\_view\_net(len(class\_names), droprate=opt.droprate, stride=opt.stride, pool=opt.pool,

share\_weight=opt.share, circle=return\_feature)

elif opt.views == 3:

model = three\_view\_net(len(class\_names), droprate=opt.droprate, stride=opt.stride, pool=opt.pool,

share\_weight=opt.share, circle=return\_feature)

opt.nclasses = len(class\_names)

print(model)

# For resume:

if start\_epoch >= 40:

opt.lr = opt.lr \* 0.1

ignored\_params = list(map(id, model.classifier.parameters()))

base\_params = filter(lambda p: id(p) not in ignored\_params, model.parameters())

optimizer\_ft = optim.SGD([

{'params': base\_params, 'lr': 0.1 \* opt.lr},

{'params': model.classifier.parameters(), 'lr': opt.lr}

], weight\_decay=5e-4, momentum=0.9, nesterov=True)

# Decay LR by a factor of 0.1 every 40 epochs

exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_ft, step\_size=80, gamma=0.1)

######################################################################

# Train and evaluate

# ^^^^^^^^^^^^^^^^^^

#

# It should take around 1-2 hours on GPU.

#

dir\_name = os.path.join('./model', name)

if not opt.resume:

if not os.path.isdir(dir\_name):

os.mkdir(dir\_name)

# record every run

copyfile('train.py', dir\_name + '/train.py')

copyfile('./model.py', dir\_name + '/model.py')

# save opts

with open('%s/opts.yaml' % dir\_name, 'w') as fp:

yaml.dump(vars(opt), fp, default\_flow\_style=False)

# model to gpu

model = model.cuda()

if fp16:

model, optimizer\_ft = amp.initialize(model, optimizer\_ft, opt\_level="O1")

criterion = nn.CrossEntropyLoss()

if opt.moving\_avg < 1.0:

model\_test = copy.deepcopy(model)

num\_epochs = 140

else:

model\_test = None

num\_epochs = 120

model = train\_model(model, model\_test, criterion, optimizer\_ft, exp\_lr\_scheduler,

num\_epochs=num\_epochs)

Model.py:import torch

import torch.nn as nn

from torch.nn import init

from torchvision import models

from torch.autograd import Variable

from torch.nn import functional as F

######################################################################

class GeM(nn.Module):

# GeM zhedong zheng

def \_\_init\_\_(self, dim = 2048, p=3, eps=1e-6):

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

self.p = nn.Parameter(torch.ones(dim)\*p, requires\_grad = True) #initial p

self.eps = eps

self.dim = dim

def forward(self, x):

return self.gem(x, p=self.p, eps=self.eps)

def gem(self, x, p=3, eps=1e-6):

x = torch.transpose(x, 1, -1)

x = x.clamp(min=eps).pow(p)

x = torch.transpose(x, 1, -1)

x = F.avg\_pool2d(x, (x.size(-2), x.size(-1)))

x = x.view(x.size(0), x.size(1))

x = x.pow(1./p)

return x

def \_\_repr\_\_(self):

return self.\_\_class\_\_.\_\_name\_\_ + '(' + 'p=' + '{:.4f}'.format(self.p.data.tolist()[0]) + ', ' + 'eps=' + str(self.eps) + ',' + 'dim='+str(self.dim)+')'

def weights\_init\_kaiming(m):

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

# print(classname)

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

init.kaiming\_normal\_(m.weight.data, a=0, mode='fan\_in') # For old pytorch, you may use kaiming\_normal.

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

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

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

elif classname.find('BatchNorm1d') != -1:

init.normal\_(m.weight.data, 1.0, 0.02)

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

def weights\_init\_classifier(m):

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

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

init.normal\_(m.weight.data, std=0.001)

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

def fix\_relu(m):

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

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

m.inplace=True

# Defines the new fc layer and classification layer

# |--Linear--|--bn--|--relu--|--Linear--|

class ClassBlock(nn.Module):

def \_\_init\_\_(self, input\_dim, class\_num, droprate, relu=False, bnorm=True, num\_bottleneck=512, linear=True, return\_f = False):

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

self.return\_f = return\_f

add\_block = []

if linear:

add\_block += [nn.Linear(input\_dim, num\_bottleneck)]

else:

num\_bottleneck = input\_dim

if bnorm:

add\_block += [nn.BatchNorm1d(num\_bottleneck)]

if relu:

add\_block += [nn.LeakyReLU(0.1)]

if droprate>0:

add\_block += [nn.Dropout(p=droprate)]

add\_block = nn.Sequential(\*add\_block)

add\_block.apply(weights\_init\_kaiming)

classifier = []

classifier += [nn.Linear(num\_bottleneck, class\_num)]

classifier = nn.Sequential(\*classifier)

classifier.apply(weights\_init\_classifier)

self.add\_block = add\_block

self.classifier = classifier

def forward(self, x):

x = self.add\_block(x)

if self.return\_f:

f = x

x = self.classifier(x)

return x,f

else:

x = self.classifier(x)

return x

class ft\_net\_VGG16(nn.Module):

def \_\_init\_\_(self, class\_num, droprate=0.5, stride=2, init\_model=None, pool='avg'):

super(ft\_net\_VGG16, self).\_\_init\_\_()

model\_ft = models.vgg16\_bn(pretrained=True)

# avg pooling to global pooling

#if stride == 1:

# model\_ft.layer4[0].downsample[0].stride = (1,1)

# model\_ft.layer4[0].conv2.stride = (1,1)

self.pool = pool

if pool =='avg+max':

model\_ft.avgpool2 = nn.AdaptiveAvgPool2d((1,1))

model\_ft.maxpool2 = nn.AdaptiveMaxPool2d((1,1))

#self.classifier = ClassBlock(4096, class\_num, droprate)

elif pool=='avg':

model\_ft.avgpool2 = nn.AdaptiveAvgPool2d((1,1))

#self.classifier = ClassBlock(2048, class\_num, droprate)

elif pool=='max':

model\_ft.maxpool2 = nn.AdaptiveMaxPool2d((1,1))

elif pool=='gem':

model\_ft.gem2 = GeM(dim = 512)

self.model = model\_ft

if init\_model!=None:

self.model = init\_model.model

self.pool = init\_model.pool

#self.classifier.add\_block = init\_model.classifier.add\_block

def forward(self, x):

x = self.model.features(x)

if self.pool == 'avg+max':

x1 = self.model.avgpool2(x)

x2 = self.model.maxpool2(x)

x = torch.cat((x1,x2), dim = 1)

elif self.pool == 'avg':

x = self.model.avgpool2(x)

elif self.pool == 'max':

x = self.model.maxpool2(x)

elif self.pool=='gem':

x = self.model.gem2(x)

x = x.view(x.size(0), x.size(1))

#x = self.classifier(x)

return x

# Define the ResNet50-based Model

class ft\_net(nn.Module):

def \_\_init\_\_(self, class\_num, droprate=0.5, stride=2, init\_model=None, pool='avg'):

super(ft\_net, self).\_\_init\_\_()

model\_ft = models.resnet50(pretrained=True)

# avg pooling to global pooling

if stride == 1:

model\_ft.layer4[0].downsample[0].stride = (1,1)

model\_ft.layer4[0].conv2.stride = (1,1)

self.pool = pool

if pool =='avg+max':

model\_ft.avgpool2 = nn.AdaptiveAvgPool2d((1,1))

model\_ft.maxpool2 = nn.AdaptiveMaxPool2d((1,1))

#self.classifier = ClassBlock(4096, class\_num, droprate)

elif pool=='avg':

model\_ft.avgpool2 = nn.AdaptiveAvgPool2d((1,1))

#self.classifier = ClassBlock(2048, class\_num, droprate)

elif pool=='max':

model\_ft.maxpool2 = nn.AdaptiveMaxPool2d((1,1))

elif pool=='gem':

model\_ft.gem2 = GeM(dim=2048)

self.model = model\_ft

if init\_model!=None:

self.model = init\_model.model

self.pool = init\_model.pool

#self.classifier.add\_block = init\_model.classifier.add\_block

def forward(self, x):

x = self.model.conv1(x)

x = self.model.bn1(x)

x = self.model.relu(x)

x = self.model.maxpool(x)

x = self.model.layer1(x)

x = self.model.layer2(x)

x = self.model.layer3(x)

x = self.model.layer4(x)

if self.pool == 'avg+max':

x1 = self.model.avgpool2(x)

x2 = self.model.maxpool2(x)

x = torch.cat((x1,x2), dim = 1)

elif self.pool == 'avg':

x = self.model.avgpool2(x)

elif self.pool == 'max':

x = self.model.maxpool2(x)

elif self.pool == 'gem':

x = self.model.gem2(x)

x = x.view(x.size(0), x.size(1))

#x = self.classifier(x)

return x

class two\_view\_net(nn.Module):

def \_\_init\_\_(self, class\_num, droprate, stride = 2, pool = 'avg', share\_weight = False, VGG16=False, circle=False,):

super(two\_view\_net, self).\_\_init\_\_()

if VGG16:

self.model\_1 = ft\_net\_VGG16(class\_num, stride=stride, pool = pool)

else:

self.model\_1 = ft\_net(class\_num, stride=stride, pool = pool)

if share\_weight:

self.model\_2 = self.model\_1

else:

if VGG16:

self.model\_2 = ft\_net\_VGG16(class\_num, stride = stride, pool = pool)

else:

self.model\_2 = ft\_net(class\_num, stride = stride, pool = pool)

self.circle = circle

self.classifier = ClassBlock(2048, class\_num, droprate, return\_f = circle)

if pool =='avg+max':

self.classifier = ClassBlock(4096, class\_num, droprate, return\_f = circle)

if VGG16:

self.classifier = ClassBlock(512, class\_num, droprate, return\_f = circle)

if pool =='avg+max':

self.classifier = ClassBlock(1024, class\_num, droprate, return\_f = circle)

def forward(self, x1, x2):

if x1 is None:

y1 = None

else:

x1 = self.model\_1(x1)

y1 = self.classifier(x1)

if x2 is None:

y2 = None

else:

x2 = self.model\_2(x2)

y2 = self.classifier(x2)

return y1, y2

class three\_view\_net(nn.Module):

def \_\_init\_\_(self, class\_num, droprate, stride = 2, pool = 'avg', share\_weight = False, VGG16=False, circle=False):

super(three\_view\_net, self).\_\_init\_\_()

if VGG16:

self.model\_1 = ft\_net\_VGG16(class\_num, stride = stride, pool = pool)

self.model\_2 = ft\_net\_VGG16(class\_num, stride = stride, pool = pool)

else:

self.model\_1 = ft\_net(class\_num, stride = stride, pool = pool)

self.model\_2 = ft\_net(class\_num, stride = stride, pool = pool)

if share\_weight:

self.model\_3 = self.model\_1

else:

if VGG16:

self.model\_3 = ft\_net\_VGG16(class\_num, stride = stride, pool = pool)

else:

self.model\_3 = ft\_net(class\_num, stride = stride, pool = pool)

self.circle = circle

self.classifier = ClassBlock(2048, class\_num, droprate, return\_f = circle)

if pool =='avg+max':

self.classifier = ClassBlock(4096, class\_num, droprate, return\_f = circle)

def forward(self, x1, x2, x3, x4 = None): # x4 is extra data

if x1 is None:

y1 = None

else:

x1 = self.model\_1(x1)

x1 = x1.view(x1.size(0), x1.size(1))

y1 = self.classifier(x1)

if x2 is None:

y2 = None

else:

x2 = self.model\_2(x2)

x2 = x2.view(x2.size(0), x2.size(1))

y2 = self.classifier(x2)

if x3 is None:

y3 = None

else:

x3 = self.model\_3(x3)

x3 = x3.view(x3.size(0), x3.size(1))

y3 = self.classifier(x3)

if x4 is None:

return y1, y2, y3

else:

x4 = self.model\_2(x4)

x4 = x4.view(x4.size(0), x4.size(1))

y4 = self.classifier(x4)

return y1, y2, y3, y4

'''

# debug model structure

# Run this code with:

python model.py

'''

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

# Here I left a simple forward function.

# Test the model, before you train it.

net = two\_view\_net(751, droprate=0.5, VGG16=True)

#net.classifier = nn.Sequential()

print(net)

input = Variable(torch.FloatTensor(8, 3, 256, 256))

output,output = net(input,input)

print('net output size:')

print(output.shape)

test\_160.py:# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import print\_function, division

import argparse

import torch

import torch.nn as nn

import torch.optim as optim

from torch.optim import lr\_scheduler

from torch.autograd import Variable

import torch.backends.cudnn as cudnn

import numpy as np

import torchvision

from torchvision import datasets, models, transforms

import time

import os

import scipy.io

import yaml

import math

from utils import load\_network

from image\_folder import CustomData160k\_sat, CustomData160k\_drone

#fp16

try:

from apex.fp16\_utils import \*

except ImportError: # will be 3.x series

print('This is not an error. If you want to use low precision, i.e., fp16, please install the apex with cuda support (https://github.com/NVIDIA/apex) and update pytorch to 1.0')

######################################################################

# Options

# --------

parser = argparse.ArgumentParser(description='Training')

parser.add\_argument('--gpu\_ids',default='0', type=str,help='gpu\_ids: e.g. 0 0,1,2 0,2')

parser.add\_argument('--which\_epoch',default='last', type=str, help='0,1,2,3...or last')

parser.add\_argument('--test\_dir',default='./data/test',type=str, help='./test\_data')

parser.add\_argument('--name', default='three\_view\_long\_share\_d0.75\_256\_s1\_google', type=str, help='save model path')

parser.add\_argument('--pool', default='avg', type=str, help='avg|max')

parser.add\_argument('--batchsize', default=128, type=int, help='batchsize')

parser.add\_argument('--PCB', action='store\_true', help='use PCB' )

parser.add\_argument('--h', default=256, type=int, help='height')

parser.add\_argument('--w', default=256, type=int, help='width')

parser.add\_argument('--views', default=2, type=int, help='views')

parser.add\_argument('--pad', default=0, type=int, help='padding')

parser.add\_argument('--use\_dense', action='store\_true', help='use densenet121' )

parser.add\_argument('--LPN', action='store\_true', help='use LPN' )

parser.add\_argument('--multi', action='store\_true', help='use multiple query' )

parser.add\_argument('--fp16', action='store\_true', help='use fp16.' )

parser.add\_argument('--scale\_test', action='store\_true', help='scale test' )

parser.add\_argument('--ms',default='1', type=str,help='multiple\_scale: e.g. 1 1,1.1 1,1.1,1.2')

parser.add\_argument('--query\_name', default='query\_street\_name.txt', type=str,help='load query image')

opt = parser.parse\_args()

###load config###

# load the training config

config\_path = os.path.join('./model',opt.name,'opts.yaml')

with open(config\_path, 'r') as stream:

config = yaml.load(stream,Loader=yaml.FullLoader)

opt.fp16 = config['fp16']

opt.use\_dense = config['use\_dense']

opt.use\_NAS = config['use\_NAS']

opt.stride = config['stride']

opt.views = config['views']

opt.LPN = False

opt.block = 0

scale\_test = opt.scale\_test

if 'h' in config:

opt.h = config['h']

opt.w = config['w']

print('------------------------------',opt.h)

if 'nclasses' in config: # tp compatible with old config files

opt.nclasses = config['nclasses']

else:

opt.nclasses = 729

str\_ids = opt.gpu\_ids.split(',')

#which\_epoch = opt.which\_epoch

name = opt.name

test\_dir = opt.test\_dir

query\_name = opt.query\_name

gpu\_ids = []

for str\_id in str\_ids:

id = int(str\_id)

if id >=0:

gpu\_ids.append(id)

print('We use the scale: %s'%opt.ms)

str\_ms = opt.ms.split(',')

ms = []

for s in str\_ms:

s\_f = float(s)

ms.append(math.sqrt(s\_f))

# set gpu ids

if len(gpu\_ids)>0:

torch.cuda.set\_device(gpu\_ids[0])

cudnn.benchmark = True

######################################################################

# Load Data

# ---------

#

# We will use torchvision and torch.utils.data packages for loading the

# data.

#

data\_transforms = transforms.Compose([

transforms.Resize((opt.h, opt.w), interpolation=3),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

#像素点平移动的transforms

transform\_move\_list = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

if opt.LPN:

data\_transforms = transforms.Compose([

# transforms.Resize((384,192), interpolation=3),

transforms.Resize((opt.h,opt.w), interpolation=3),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

data\_dir = test\_dir

image\_datasets = {}

image\_datasets['gallery\_satellite'] = CustomData160k\_sat(os.path.join(data\_dir, 'workshop\_gallery\_satellite'), data\_transforms)

image\_datasets['query\_street'] = CustomData160k\_drone( os.path.join(data\_dir,'workshop\_query\_street') ,data\_transforms, query\_name = query\_name)

print(image\_datasets.keys())

dataloaders = {x: torch.utils.data.DataLoader(image\_datasets[x], batch\_size=opt.batchsize,

shuffle=False, num\_workers=16) for x in

['gallery\_satellite','query\_street']}

use\_gpu = torch.cuda.is\_available()

######################################################################

# Extract feature

# ----------------------

#

# Extract feature from a trained model.

#

def fliplr(img):

'''flip horizontal'''

inv\_idx = torch.arange(img.size(3)-1,-1,-1).long() # N x C x H x W

img\_flip = img.index\_select(3,inv\_idx)

return img\_flip

def which\_view(name):

if 'satellite' in name:

return 1

elif 'street' in name:

return 2

elif 'drone' in name:

return 3

else:

print('unknown view')

return -1

def extract\_feature(model,dataloaders, view\_index = 1):

features = torch.FloatTensor()

count = 0

for data in dataloaders:

img, label = data

n, c, h, w = img.size()

count += n

print(count)

ff = torch.FloatTensor(n,512).zero\_().cuda()

# if opt.LPN:

# # ff = torch.FloatTensor(n,2048,6).zero\_().cuda()

for i in range(2):

if(i==1):

img = fliplr(img)

input\_img = Variable(img.cuda())

for scale in ms:

if scale != 1:

# bicubic is only available in pytorch>= 1.1

input\_img = nn.functional.interpolate(input\_img, scale\_factor=scale, mode='bilinear', align\_corners=False)

if view\_index == 1:

outputs,\_, \_ = model(input\_img, None,None)

elif view\_index == 2:

\_, outputs,\_ = model(None, input\_img, None)

# outputs, outputs\_se = model(input\_img)

ff += outputs

# norm feature

if opt.LPN:

# feature size (n,2048,6)

# 1. To treat every part equally, I calculate the norm for every 2048-dim part feature.

# 2. To keep the cosine score==1, sqrt(6) is added to norm the whole feature (2048\*6).

fnorm = torch.norm(ff, p=2, dim=1, keepdim=True) \* np.sqrt(10)

ff = ff.div(fnorm.expand\_as(ff))

ff = ff.view(ff.size(0), -1)

else:

fnorm = torch.norm(ff, p=2, dim=1, keepdim=True)

ff = ff.div(fnorm.expand\_as(ff))

features = torch.cat((features,ff.data.cpu()), 0)

return features

def get\_SatId\_160k(img\_path):

labels = []

paths = []

for path,v in img\_path:

labels.append(v)

paths.append(path)

return labels, paths

def get\_result\_rank10(qf,gf,gl):

query = qf.view(-1,1)

score = torch.mm(gf, query)

score = score.squeeze(1).cpu()

score = score.numpy()

index = np.argsort(score)

index = index[::-1]

rank10\_index = index[0:10]

result\_rank10 = gl[rank10\_index]

return result\_rank10

######################################################################

# Load Collected data Trained model

print('-------test-----------')

model, \_, epoch = load\_network(opt.name, opt)

if opt.LPN:

print('use LPN')

# model = three\_view\_net\_test(model)

for i in range(opt.block):

cls\_name = 'classifier'+str(i)

c = getattr(model, cls\_name)

c.classifier = nn.Sequential()

else:

model.classifier.classifier = nn.Sequential()

model = model.eval()

if use\_gpu:

model = model.cuda()

# Extract feature

since = time.time()

query\_name = 'query\_street' #1

gallery\_name = 'gallery\_satellite' #1

which\_gallery = which\_view(gallery\_name)

which\_query = which\_view(query\_name)

gallery\_path = image\_datasets[gallery\_name].imgs

gallery\_label, gallery\_path = get\_SatId\_160k(gallery\_path)

print('%d -> %d:'%(which\_query, which\_gallery))

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

with torch.no\_grad():

print('-------------------extract query feature----------------------')

query\_feature = extract\_feature(model,dataloaders[query\_name], which\_query)

print('-------------------extract gallery feature----------------------')

gallery\_feature = extract\_feature(model,dataloaders[gallery\_name], which\_gallery)

print('--------------------------ending extract-------------------------------')

time\_elapsed = time.time() - since

print('Test complete in {:.0f}m {:.0f}s'.format(

time\_elapsed // 60, time\_elapsed % 60))

query\_feature = query\_feature.cuda()

gallery\_feature = gallery\_feature.cuda()

save\_filename = 'results\_rank10.txt'

if os.path.isfile(save\_filename):

os.remove(save\_filename)

results\_rank10 = []

print(len(query\_feature))

gallery\_label = np.array(gallery\_label)

for i in range(len(query\_feature)):

result\_rank10 = get\_result\_rank10(query\_feature[i], gallery\_feature, gallery\_label)

results\_rank10.append(result\_rank10)

results\_rank10 = np.row\_stack(results\_rank10)

if os.path.isfile(save\_filename):

os.remove(save\_filename)

with open(save\_filename, 'w') as f:

for row in results\_rank10:

f.write('\t'.join(map(str, row)) + '\n')

utils.py:import os

import torch

import yaml

from model import two\_view\_net, three\_view\_net

def make\_weights\_for\_balanced\_classes(images, nclasses):

count = [0] \* nclasses

for item in images:

count[item[1]] += 1 # count the image number in every class

weight\_per\_class = [0.] \* nclasses

N = float(sum(count))

for i in range(nclasses):

weight\_per\_class[i] = N/float(count[i])

weight = [0] \* len(images)

for idx, val in enumerate(images):

weight[idx] = weight\_per\_class[val[1]]

return weight

# Get model list for resume

def get\_model\_list(dirname, key):

if os.path.exists(dirname) is False:

print('no dir: %s'%dirname)

return None

gen\_models = [os.path.join(dirname, f) for f in os.listdir(dirname) if

os.path.isfile(os.path.join(dirname, f)) and key in f and ".pth" in f]

if gen\_models is None:

return None

gen\_models.sort()

last\_model\_name = gen\_models[-1]

return last\_model\_name

######################################################################

# Save model

#---------------------------

def save\_network(network, dirname, epoch\_label):

if not os.path.isdir('./model/'+dirname):

os.mkdir('./model/'+dirname)

if isinstance(epoch\_label, int):

save\_filename = 'net\_%03d.pth'% epoch\_label

else:

save\_filename = 'net\_%s.pth'% epoch\_label

save\_path = os.path.join('./model',dirname,save\_filename)

torch.save(network.cpu().state\_dict(), save\_path)

if torch.cuda.is\_available:

network.cuda()

######################################################################

# Load model for resume

#---------------------------

def load\_network(name, opt):

# Load config

dirname = os.path.join('./model',name)

last\_model\_name = os.path.basename(get\_model\_list(dirname, 'net'))

epoch = last\_model\_name.split('\_')[1]

epoch = epoch.split('.')[0]

if not epoch=='last':

epoch = int(epoch)

config\_path = os.path.join(dirname,'opts.yaml')

with open(config\_path, 'r') as stream:

config = yaml.load(stream, Loader=yaml.FullLoader)

opt.name = config['name']

opt.data\_dir = config['data\_dir']

opt.train\_all = config['train\_all']

opt.droprate = config['droprate']

opt.color\_jitter = config['color\_jitter']

opt.batchsize = config['batchsize']

opt.h = config['h']

opt.w = config['w']

opt.share = config['share']

opt.stride = config['stride']

if 'pool' in config:

opt.pool = config['pool']

if 'h' in config:

opt.h = config['h']

opt.w = config['w']

if 'gpu\_ids' in config:

opt.gpu\_ids = config['gpu\_ids']

opt.erasing\_p = config['erasing\_p']

opt.lr = config['lr']

opt.nclasses = config['nclasses']

opt.erasing\_p = config['erasing\_p']

opt.use\_dense = config['use\_dense']

opt.fp16 = config['fp16']

opt.views = config['views']

if opt.use\_dense:

model = ft\_net\_dense(opt.nclasses, opt.droprate, opt.stride, None, opt.pool)

if opt.PCB:

model = PCB(opt.nclasses)

if opt.views == 2:

model = two\_view\_net(opt.nclasses, opt.droprate, stride = opt.stride, pool = opt.pool, share\_weight = opt.share)

elif opt.views == 3:

model = three\_view\_net(opt.nclasses, opt.droprate, stride = opt.stride, pool = opt.pool, share\_weight = opt.share)

if 'use\_vgg16' in config:

opt.use\_vgg16 = config['use\_vgg16']

if opt.views == 2:

model = two\_view\_net(opt.nclasses, opt.droprate, stride = opt.stride, pool = opt.pool, share\_weight = opt.share, VGG16 = opt.use\_vgg16)

elif opt.views == 3:

model = three\_view\_net(opt.nclasses, opt.droprate, stride = opt.stride, pool = opt.pool, share\_weight = opt.share, VGG16 = opt.use\_vgg16)

# load model

if isinstance(epoch, int):

save\_filename = 'net\_%03d.pth'% epoch

else:

save\_filename = 'net\_%s.pth'% epoch

save\_path = os.path.join('./model',name,save\_filename)

print('Load the model from %s'%save\_path)

network = model

network.load\_state\_dict(torch.load(save\_path))

return network, opt, epoch

def toogle\_grad(model, requires\_grad):

for p in model.parameters():

p.requires\_grad\_(requires\_grad)

def update\_average(model\_tgt, model\_src, beta):

toogle\_grad(model\_src, False)

toogle\_grad(model\_tgt, False)

param\_dict\_src = dict(model\_src.named\_parameters())

for p\_name, p\_tgt in model\_tgt.named\_parameters():

p\_src = param\_dict\_src[p\_name]

assert(p\_src is not p\_tgt)

p\_tgt.copy\_(beta\*p\_tgt + (1. - beta)\*p\_src)

toogle\_grad(model\_src, True)

要求：Important Dates

Submission of papers:

Challenge Start: 9 March 2025

Challenge End: 30 June 2025

Workshop Papers Submission End: 7 July 2025

Workshop Papers Notification: 24 July 2025

Student Travel Grants Application Deadline: TBD

Camera-ready Submission: 3 August 2025

Conference Dates: 27 October 2025 – 31 October 2025

Please note: The submission deadline is at 11:59 p.m. of the stated deadline date Anywhere on Earth

Workshop homepage is at https://github.com/spyflying/ACMMM2025Workshop-UAV

Challenge Overview

This year's focus is specifically on matching partial street images to corresponding satellite images (illustrated in Figure 3 of the proposal). By concentrating on partial views, our aim is to more accurately reflect real-world scenarios where obstructions or limited sensor angles may restrict the field of view, such as during low-altitude UAV operations for navigation, search-and-rescue missions, and autonomous flight. We harness University-1652 [40] as the challenge dataset, which provides 2,579 street images as query and 951 gallery satellite images. To encourage broader participation and innovation, we will make University-1652 training set available through our website with name-masked test set, along with a public leaderboard.

Check challenge details at Section 5 in https://www.zdzheng.xyz/files/MM25\_Workshop\_Proposal\_Drone.pdf

The training dataset can be download by sending the Request. Usually I will reply the download link in 5 minutes. The name-masked test set (query\_street & gallery\_satellite) can be downloaded from OnDrive. Please do not use the University-1652 test set for training. Using it for training will lead to the revocation of the participants’ award.

The submission example can be found at Baseline Submission. Please zip it as “answer.zip” to submit the result, and it is crucial to name the file exactly as answer.txt within the zip, as otherwise the evaluation will fail.

Please return the top-10 satellite names. For example, the first query is ``VdthudbGjJ4aaNkl.jpeg''. Therefore, the first line of returned result in ``answer.txt'' should be the format as follows from Rank-1 to Rank-10:

ptHYAN3piG3YwOft I9bzP8jnLlz9zpMi c3vVTLCzTAVzuapU gkriPL4PNtcWoHgg iIL2ASdQ5vrFsJs0 TinwNxUGYAzz0kTO XilyyHqywhUBxHfT WLasj720MnF13zPI Qz4NypYGPhHdiAvn gO2hUfIHC8N4ZWKz

Please return the result following the order of query at Query Name Txt. It will be 2759 lines.