baseline：

decay=0， epoch=40时，rank 1为0.76，map为0.58

market\_softmax\_pretrain\_f2:

epoch=50,decay=0.01, rank1为0.288599；mAP为0.176

结果是因为decay设置得太大，因为decay是每个batch都要执行一次衰减，不是每个epoch

===========================================

rank-reid复现Dis. /baseline， /pretrain

1)baseline:

结果是rank 1为0.76，map为0.58。调整过训练epoch，没有使得效果变好

2)I+V：

利用baseline的结果，再用V+I训练，结果rank 1为0.80，map为0.65

1)和2)都效果不如文章中呈现的。

可能的原因：1）利用resnet的时候，不应该加上avg\_pooling层；2）不需要用baseline预训练，直接用pre-trained来V+I （认识误区，rank-reid中实现的是single query，我们对比的文章结果不对）

/baseline-dis 我们的复现

evaluate\_v2.py 目的是优化predict的效率。把predict的batchsize设置为128，时间开销从250s到50s；增加了multiple query；增加了rank-k的acc，即CMC

train.py 跟原本的train函数一样，修改了batchsize的大小，16-v1，32-v2，16效果好一点

bs48-ep75 v3,

经考虑，把train.py改回了原始的样子，即选用avgpooling而不是globalavgpooling，bs为16，epoch为40

Single query：Rank 1: 0.745249, Rank 5: 0.890143, Rank 10: 0.927850 mAP: 0.572520

Mult queries：Rank 1: 0.826306, Rank 5: 0.937648, Rank 10: 0.957838 mAP: 0.670916

Pair-train：

Single query：Rank 1: 0.778207, Rank 5: 0.902019, Rank 10: 0.934976 mAP: 0.627558

Mult queries：Rank 1: 0.848278, Rank 5: 0.942696, Rank 10: 0.963183 mAP: 0.712499

一个测试：根据dis-reid的matlab代码，修改batchsize大小为48，epoch为75

epoch05-----Rank 1: 0.514549, Rank 5: 0.700713, Rank 10: 0.781770 mAP: 0.338359

epoch10-----Rank 1: 0.649347, Rank 5: 0.818884, Rank 10: 0.878563 mAP: 0.472597

epoch15-----Rank 1: 0.676663, Rank 5: 0.839371, Rank 10: 0.892518 mAP: 0.503466

epoch20-----Rank 1: 0.694774, Rank 5: 0.852732, Rank 10: 0.904394 mAP: 0.516377

epoch25-----Rank 1: 0.690914, Rank 5: 0.855998, Rank 10: 0.903800 mAP: 0.521718

epoch30-----Rank 1: 0.695962, Rank 5: 0.858076, Rank 10: 0.904988 mAP: 0.524569

epoch35-----Rank 1: 0.695368, Rank 5: 0.857185, Rank 10: 0.907660 mAP: 0.524908

epoch40-----Rank 1: 0.699525, Rank 5: 0.862530, Rank 10: 0.907660 mAP: 0.531269

epoch45-----Rank 1: 0.692399, Rank 5: 0.858967, Rank 10: 0.907067 mAP: 0.525371

epoch50-----Rank 1: 0.695962, Rank 5: 0.861639, Rank 10: 0.912411 mAP: 0.530570

epoch55-----Rank 1: 0.694181, Rank 5: 0.866983, Rank 10: 0.914192 mAP: 0.531990

epoch60-----Rank 1: 0.700416, Rank 5: 0.867874, Rank 10: 0.916271 mAP: 0.532622

epoch65-----Rank 1: 0.694774, Rank 5: 0.861639, Rank 10: 0.911817 mAP: 0.529852

epoch70-----Rank 1: 0.700713, Rank 5: 0.865202, Rank 10: 0.914192 mAP: 0.534875

epoch75-----Rank 1: 0.698337, Rank 5: 0.865796, Rank 10: 0.915677 mAP: 0.534003

multiple queries：Rank 1:0.789786, Rank 5: 0.915380, Rank 10: 0.948337 mAP: 0.623164（这个是基于v3的结果来的，使用最原先的设置，可以变到论文的acc）

noted：single/multiple shot和single/multiple queries的区别

single/multiple shot是指两种测试场景，分别对应gallery中有一个gt和多个gtz

single/multiple queries：是指两种测试场景/或者把后者看成提高鲁棒性的一种方式

指的是testing过程中，每个query是单张image，或者是多个image的结合（avg/max）

对于market-1501数据集来说，考虑single/multiple queries两种场景，前者是利用query数据集，后者是利用gt\_bbox,具体可以参考代码

https://github.com/layumi/2016\_person\_reID/blob/master/evaluation/zzd\_evaluation\_res\_fast.m

对于VIPeR，GRID，CUHK03数据来说，考虑single shot

对于CUHK01数据来说，考虑single/multiple shot

学到的知识：

keras中调用pretrained resnet和vgg的时候，数据的预处理需要用caffe mode，因为这个.h5文件是直接从caffe模型转化过来的

numpy读入图像时，想要把多张图像读入内存，成为numpy，但是numpy的扩展拼接函数效率都不高，在已知数组总大小时可以

考虑直接分配好，然后赋值；

目前发现最快是，用list.append来循环读，然后用np.array(list)转化

numpy的两个拼接 append和concatenate，都比较慢，不适合读入大规模数据

numpy 计算矩阵平均值,最大值 np.mean(axis)， np.max(axis)，都是按轴

报错：typeerror：list object is not callable，一般是与python built-in的函数名冲突了

Glob模块，按照通配符来搜索特定格式的文件或文件夹（如检索某个文件夹下所有.txt、abc开头的文件，返回name list）

Python中a.py调用b.py，在b中执行函数时，当前路径仍然为执行a时的路径，可以用

import os

current\_path = os.path.dirname(\_\_file\_\_)来表示b的路径

=======================

single\_eval.py

目的是可以自定义probe和gallery，突然发现了为啥这么慢，因为net.predicts是逐个样本预测

whu数据集测试，如果probe和gallery都是whu，可以rank1 100%（multishot）

注意考虑一下有没有必要使得probe和gallery不相交，似乎还好，因为本来就会把重复摄像头的去掉，至少不影响rank acc

Re-ranking是一种提高acc的方法（具体有多种）：After an initial ranking list is obtained, a good practice consists of adding a re-ranking step, with the expectation that the relevant images will receive higher ranks