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>

即对每个probe的照片，都去gt中找到该person在该摄像头下的全部照片，组成一个新的probe

Ref：Scalable Person Re-identification: A Benchmark

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

创建数据集\bobo

.\raw是原图，11个人，3个摄像头，12个点，正面、背面，其中存在数据顺序的问题（每个摄像头下人的顺序不同，有3组背面照没有紧跟正面照）。图片次序为摄像头1，u1，点1，正面，背面，栈读。

.\cropped是截取其中的行人，并label好了“ID\_

.JPG”，调整好了顺序。记录了每个点的bbox

.\resize 在\cropped的照片，resize到（550，220）

re-id模型训练时，用的样本是float32，是直接读入的img，没有做归一化，但为了输入到caffee模型中，做了一个预处理

from keras.applications.resnet50 import preprocess\_input

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

python枚举list比直接遍历list多一个index可以用

PIL对象在save时出错（乱码），代码目的是交换两个image的名字，一个解释是PIL对象类似文件流，load类似open，而不是把img全部读到了内存，因此save之前如果对原始图做了修改，也许会save乱码。解决办法，img\_to\_array，把img读入内存，那么再操作源文件就没关系了

字符串格式化

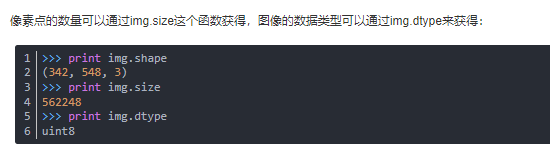
matplotlib显示opencv图像不正常

<https://blog.csdn.net/saltriver/article/details/78173219>

所有的图像变换都是对图像坐标做移动，与像素值无关

Open的图像操作（两图像相结合：融合、覆盖、抠图）

https://blog.csdn.net/poi7777/article/details/39891407



值得注意的是，cv2.imread得到的结果就是np类型。

Np对象的+操作和cv.add操作都可以融合两张图，区别在于前者超过了255会取模，后者会截断（如果是unit8）

Np的array与ndarray的区别。用np.array创建的数组其实都是ndarray类型，array本身不是一个类。不建议用ndarray创建np

Python下的各种读入图像的方法的区别

<https://zhidao.baidu.com/question/1865812457413808067.html>

关于python中存储img是否有损，后续可以研究一下，参考下面的博客。主要原因是jpg、png、bmp几种压缩的性质。对于jpg文件来说，读入jpg，然后保存成jpg，就会有值发生变化。注意的是，对于jpg和png，在保存之前都是要astype成np.uint8

https://www.jianshu.com/p/fed6a8a99625

python的包导入

<https://juejin.im/entry/570c6b6771cfe40067310370>

<https://www.cnblogs.com/lipijin/p/3848511.html>

常规导入（regular imports）

使用from语句导入

相对导入（relative imports）

可选导入（optional imports）

本地导入（local imports）

导入注意事项

Numpy创建数组常用函数

np.arange

去重index\_g = list(set(index\_g) - set(index))

(np.arange(18)).repeat(t)

x=np.concatenate((x, s\_x))

np.squeeze(axis)可以指定压缩的维度

利用Python 数据持久化方式——JSON与Pickle

但是tuple会保存后再load会变成list，图省事，就读出来之后再list()

一些tf的api

Tf.squeeze() 压缩维度值为1的维度

tf.contrib.image.下的函数库e.g. tf.contrib.image.transform

常用来做数据增强<https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9>

tf.image.resize\_images

tf.stack和tf.concat都能把多个rensor合在一起，区别在于前者会增加维度

tf.multiply与tf.matmul 前者对应相乘，后者矩阵乘

python下的生成器 每次next(g)后，会freeze住，下次从yield处接着运行，变量值会保留

tf.bool 不能自动用作mask，但可以用tf.cast把tf.bool转化到tf.float32，为0，1值

numpy的逆矩阵，用np.mat.I; 转置np.array.T; mat.T

tf.contrib.image.下的api在window下执行会crash

<https://github.com/tensorflow/tensorflow/issues/17485>

<https://github.com/tensorflow/tensorflow/issues/17747>

仿射变换的矩阵特性

<https://www.cnblogs.com/bnuvincent/p/6691189.html>

<https://blog.csdn.net/Yong_Qi2015/article/details/80071487>

<https://blog.csdn.net/xiaowei_cqu/article/details/26471527>

<https://blog.csdn.net/dcrmg/article/details/52475777>

在tf代码中调用keras .h5的代码遇到的问题：

基本流程：

Input=tf.placeholder()

Model=keras.load\_model(.h5)

Output=Model(input) #output为tensor

1．想要freeze住model的参数，model.trainable=False设置无效

解决办法：定义tf优化器的时候设定好优化的变量

如train\_step = tf.train.AdamOptimizer().minimize(loss, var\_list=trainable\_var)

参考<https://blog.csdn.net/weiwei9363/article/details/79673201>

2. sess.run(output,feed\_dict={}会报错，说使用了未初始化的tensor变量，Attempting to use uninitialized value根据名字发现是model中的tensor。这里引出一个问题，读入的keras model可以直接predict，但是用tf run之前要先初始化变量。用tf.global\_variables\_initializer()又会把全体参数全部初始化了。

一个解决办法是先用model建立新的model2，然后init=tf.global\_variables\_initializer()，再sess.run(init)，这时keras model的参数都随机了，再model.load\_weights()一次

参考：<http://blog.sina.com.cn/s/blog_1450ac3c60102xzxa.html>

这个办法不是很好，也可以选择初始化一部分变量（在部分场合下）；另外这个问题根本解法可能在.h5上，毕竟cw用了.pb就不需要初始化一次，待看

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

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

Ref：Re-ranking Person Re-identification with k-reciprocal Encoding

待看的几篇文章，与exp相关

ShapeShifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector