Video relationship detection code Readme

Edited by Zehan Song 2018/7/10

目录

[Video relationship detection code Readme 1](#_Toc521678899)

[一、 实验所用的数据说明 2](#_Toc521678900)

[relation 2](#_Toc521678901)

[traj\_cls 3](#_Toc521678902)

[traj\_cls\_gt 4](#_Toc521678903)

[train和test 5](#_Toc521678904)

[videos 6](#_Toc521678905)

[二、 实验的代码介绍 6](#_Toc521678906)

[dataset.py 7](#_Toc521678907)

[config.py 8](#_Toc521678908)

[utils.py 8](#_Toc521678909)

[main.py 8](#_Toc521678910)

[trajectory.py 9](#_Toc521678911)

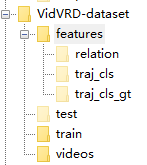
[association.py 9](#_Toc521678912)

[metric.py 10](#_Toc521678913)

## 实验所用的数据说明

数据下载渠道：<https://github.com/xdshang/VidVRD-helper>

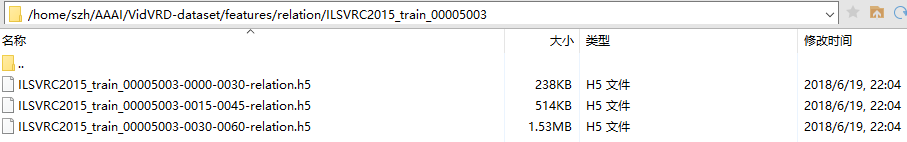
虽然里面有对数据的解释，不过还是不太直观。本人就自己的理解对实验所用到的数据进行一番说明，以下是我把下载的数据整理好建立的文件夹树示意图。



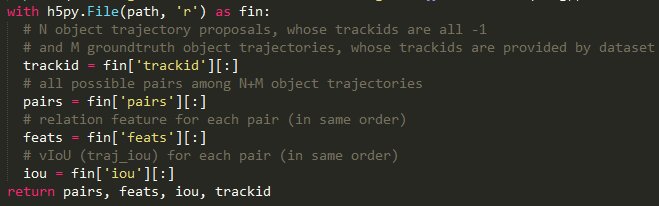
VidVRD-dataset文件夹下有四个子文件夹，分别是features，test，train，videos。features文件夹下又有三个子文件夹relation，traj\_cls，traj\_cls\_gt。

### relation

而relation文件夹下是存放着对于每个视频的每个分段提取的relation feature。



对于其中的每一个h5文件，可从中提取如下几项特征。



Example:

trackid: shape(5，) groundtruth的trackid都是-1

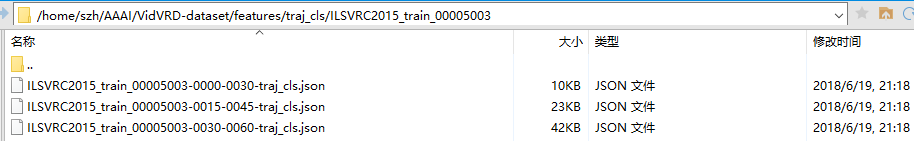
pairs: shape(20，2，) 所有的trackid的可能的组合，就是A(2，5) = 20

feats: shape(20，11070，) 每个pair所对应的11070维的关系特征

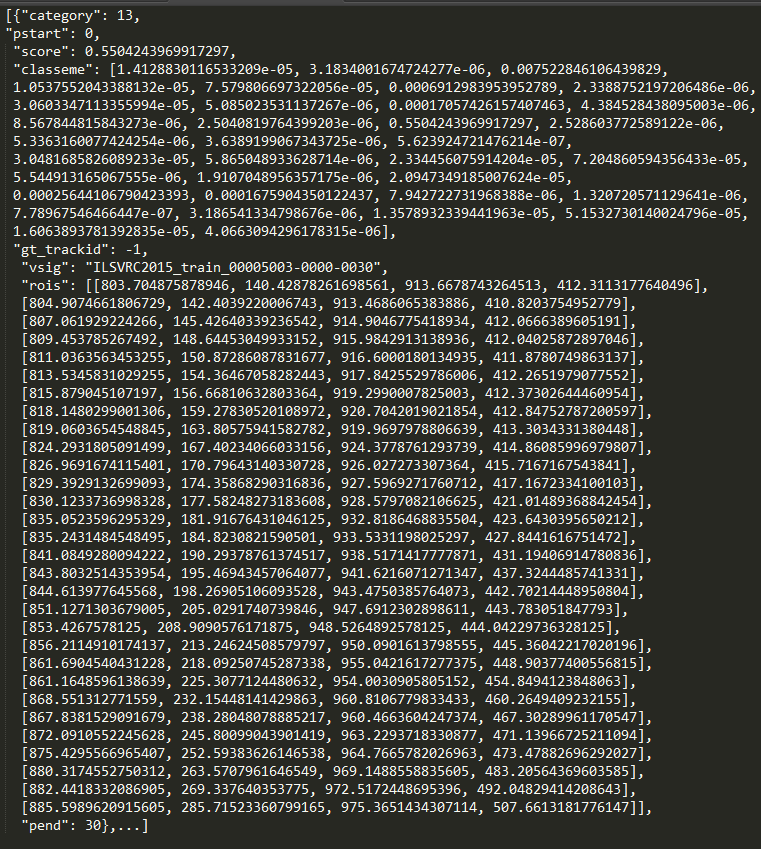
iou: shape(5，5) trajectory之间的vIoU即体积重合度

### traj\_cls

而traj\_cls文件夹下是存放着对于每个视频的每个分段(30)提取的object tracklet proposal。

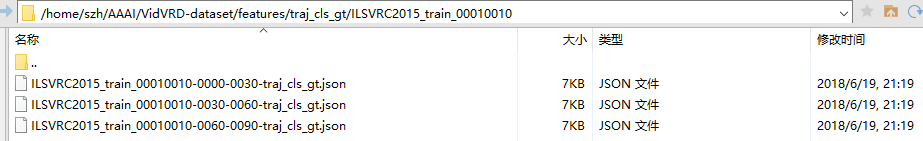


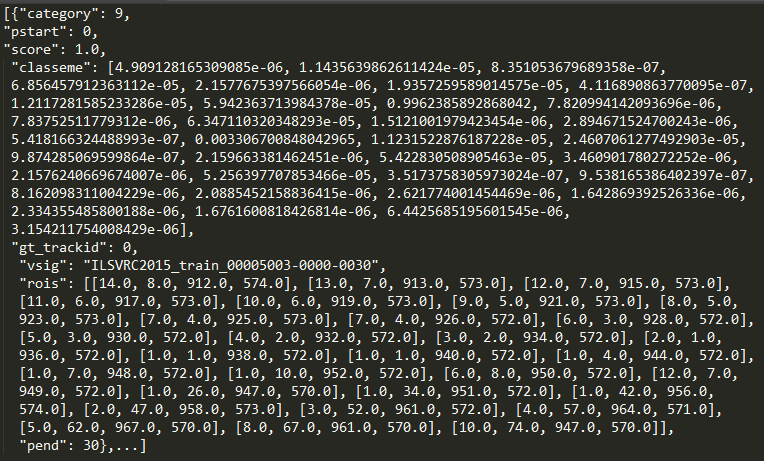
里面的json文件内容示例如下：



里面是一个字典列表，有category（物体类别号），pstart（段内开始帧号，注意不是全局），score（proposal分数），classeme（物体种类的概率分布），gt\_trackid，vsig，rois（bbox sequence），pend等几个关键字。

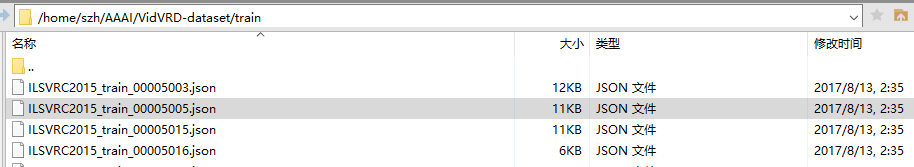
### traj\_cls\_gt

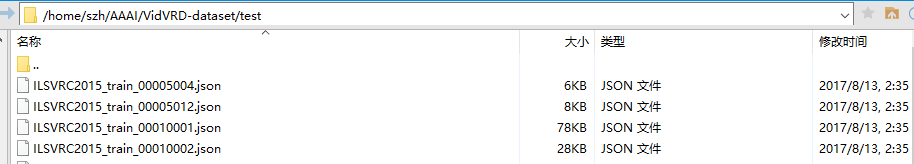
而traj\_cls\_gt文件夹下是存放着对于每个视频的每个分段（30）标注的ground truth。

里面的json文件内容示例如下：

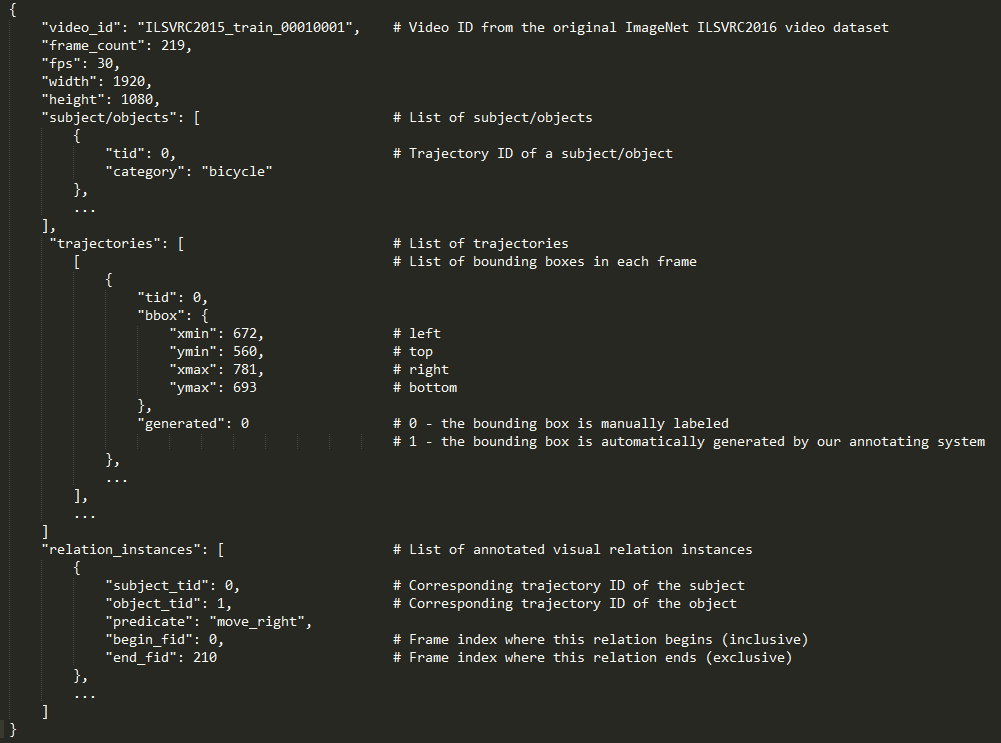
### train和test

train和test文件夹下分别放的是用于训练的800个视频的标注和用于测试的200个视频的标注（对整个视频里面出现的关系区间，而不是对区间的分段）。



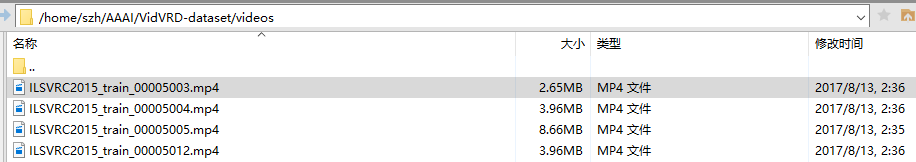


里面json文件内容如下：



### videos

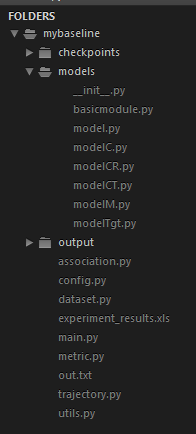
videos文件夹下存放的是原始的视频。



实验所用的数据介绍完毕，下面再从代码着手介绍该实验的baseline。

## 实验的代码介绍

本实验代码是用pytorch基于VidVRD官方的keras+tensorflow版本进行改进而来，代码组织如下图所示。



mybaseline是根目录文件夹，文件夹下有三个子文件夹，分别为checkpoints，models以及output，然后是源代码文件。在checkpoints文件夹下存放着是训练好的模型。在models文件夹下存放着是模型的源代码，放入\_\_init\_\_.py文件以将models文件夹作为package看待。在output文件夹下存放着是预测结果的json文件。

下面按pipeline的顺序来分别叙述这些源代码文件的功能。

进行一个实验首先得充分了解实验所用的数据集的结构，我们先从train和test文件夹下的json文件入手。由于这些json是对整个视频的标注，根据***video visual relation detection***一文的思路，将整个视频进行分段，然后从视频段中提取object tracklet proposal，预测short term relation，最后将这些预测出来的short term relation进行联结。

### dataset.py

该文件的功能是将所有的数据文件进行结构化，也就是读取json文件和h5文件的数据，然后存储到相应的变量中，下面具体叙述文件的代码。

该文件里定义了两个类，其中一个是SharedArray，该类定义了一个共享内存结构，用于多进程处理的实现；另一个类是Dataset类，继承Process类，这样做是为了多进程存取数据。

Dataset类主要分两块，一块是对train和test文件夹下的json文件进行处理，另一块是对relation文件夹下的h5文件进行处理。对json文件处理时，根据关键字和数据类型（train or test），将相应的数据存入list、set和dict之中。在处理数据时，先将待训练或测试的每一个video按里面出现的每一个关系分割成有重复部分的段，这些段共享同一个关系，也就是一个video里面含有很多关系，那么把这些关系所在的这些视频区间都分成有重复部分的段。最后以所有的这些段为索引关键字，对应到关系实例，需要用到时再读取文件数据。根据之前json（train or test）建立的索引关系，将json文件里的那些ground truth trackid与h5文件里的proposal trackid进行比较，找出满足条件的pair，总共需要batch\_size个。然后Dataset类里面也有一些功能型函数，如get系列的函数以及用于多进程处理数据的函数。

### config.py

该文件中定义了一个Param类，该类定义了模型参数，训练参数等超参。并定义了一个传递参数的函数，用来动态更新设定的超参数。

### utils.py

该文件中定义了一系列功能型函数，如获得段的签名戳，获得数据的路径，分割视频区间，读取h5文件的数据，特征的归一化以及将数据写到excel的函数。

### main.py

该文件中定义了训练模型以及评价模型的函数，主要有train、eval、assess等函数。

train函数中先传入设置的超参数，然后获取数据的迭代器，之后获取要训练的模型、优化器以及损失函数。数据迭代器每次输出batch\_size个满足条件的proposal pair对应的特征和相应的ground truth关系三元组的index，将batch\_size个数据传入模型中，在训练阶段就输出batch\_size个关系三元组的分数分布（2961），在测试阶段就输出batch\_size个谓词的分数分布。训练时，将输出传入损失函数中计算损失，然后进行反向传播，更新权重。每隔一定的训练次数后，输出训练的信息；每隔一定的轮数后，保存模型。

eval函数中将训练出来的模型参数加载到对应的模型中，并重新获取数据迭代器，之后在数据迭代器将数据（不是batch\_size个而是一个segment里出现的所有实例）送入到训练好的模型后中，计算出结果。对于subject、predicate、object的分数分布，都只取前20名。然后将subject的前20名与predicate和object的前20名分数进行连乘得到20\*20\*20=8000个联合分数值，然后再取前20名，并计算前20名联合分数的成员索引即哪一个subject和哪一个predicate以及哪一个object。最后将这些预测信息存入list中，之后将list中的short term relation。

assess函数中将联结好的relation instances与ground truth进行比较，计算mAP、recall以及mPrecision。

### trajectory.py

该文件中定义了Trajectory类，其实就是对物体的bounding box sequences进行结构化。然后实现了计算两个轨迹之间的iou的相关函数，还有加载轨迹的proposal的函数。

### association.py

该文件中定义了VideoRelation类，是对融合后的relation instances进行结构化。然后实现了融合short term relation的贪心算法，具体解释如下：

1. 输入预测的short term relation instances集合S(S中的元素都是属于同一个vid)
2. 初始化融合后的relation instances集合L为空以及vIoU的阈值γ=0.5
3. 假设该vid总共有T段，从第t=1段开始往后进行遍历
4. 假设A是L中前t-1段包含的融合过的relation instances，假设B是第t段包含的short term relation instances
5. 将A中的所有relation instances按分数从高到低进行排序，并将B中的所有short term relation instances按分数从高到低进行排序
6. 对B中的所有short term relation instances进行遍历，对于单个short term relation instance，再对A中所有融合的relation instances进行遍历（其实就是两个for循环）
7. 对于遍历到的B中短实例和A中长实例，比较它们的关系三元组以及subject和object的轨迹vIoU。如果关系三元组一致并且两种vIoU都超过了设定的阈值γ，那么将短实例与长实例进行融合。如果没有发生融合，那么直接将该段中的所有short term relation instances加到L中

### metric.py

该文件中定义了用于评价输出结果的几个函数，刚开始的viou函数用于计算两个轨迹的重复率，然后voc\_ap函数用于计算mAP，然后还有几个评价标记和检测的函数。

eval\_detection\_scores函数是用于评价预测结果的检测分数，先将该视频中预测到的video-level relation instances按照分数从高往低进行排序。并声明两个数组，一个是gt\_detected，长度与ground truths的长度一致，用来标记ground truth是否被检测到过，初始化为False；另一个是hit\_scores，长度与预测结果一致，初始化为-inf。对预测结果进行遍历时，对于每一个预测的video-level instance，又对ground truths进行遍历，如果遍历到的ground truth之前未被检测到过并且预测结果与ground truth的三元组一致，则进行vIoU的比较。总之，对于每一个预测结果，我们要从ground truth里面找到与其具有最大的ov=min(s\_vIoU，o\_vIoU)，并且ov得达到阈值，那么我们就将该gt\_detected上的该ground truth位置标记为True，下次就不检测，并且将hit\_scores上的该预测结果位置上分数更新为该预测结果的分数。但是，如果对于遍历到的预测结果而言，在未被检测到的ground truth里面，没有一个与它的ov达到阈值，那么hit\_scores上该预测结果的位置上的分数仍为-inf。

最后计算true positive和false positive，以此来计算累计recall(分母为len(gt\_relations))和累计precision(分母为cum\_fp+cum\_tp)。true positive、false positive、recall、precision的数组长度均预测结果的长度保持一致。

eval\_tagging\_scores函数是用于评价预测结果的标记分数，将ground truths的关系三元组单独抽出来建立一个集合(去除重复)gt\_triplets，并且把预测结果的关系三元组也单独抽出来建立一个列表pred\_triplets，同时声明一个hit\_scores列表，长度与预测结果一致，将预测结果的分数装进去。然后对pred\_triplets进行遍历，如过遍历到的预测结果在gt\_triplets里面，那么保留该预测结果在hit\_scores上对应位置的分数，否则赋值为-inf。然后通过hit\_scores来计算true positive，并计算同上的recall、precision。

eval\_visual\_relation函数用于评价对所有测试视频的预测结果，上述两个函数的作用对象是单个视频里的预测结果(video level relation instances)。对于mAP，我们先计算每个视频的预测结果的AP(Average precision)，即计算precision和recall所形成的PR曲线的面积，然后所有的AP的均值就是mAP了。对于detection recall@K，我们将所有视频的detection hit\_scores的前K（不足K就取len(hit\_scores)）加入tot\_scores[K]列表，同时统计所有视频的gt\_relations的长度，并将对应的true positive加入tot\_tp[K]列表。之后将合并操作(concatenate)之后的tot\_scores[K]里面所有的预测分数从高往低进行排序，并求出对应的true positive，之后计算累计tp，然后通过cum\_tp计算出recall@K(分母为tot\_gt\_relations即所有的len(gt\_relations)的总和)。对于tagging precision @K，我们将所有视频的tagging precision的前K（不足K就取len(precision)）加入prec\_at\_n[K]列表，之后计算prec\_at\_n[K]列表元素的平均值，就得到了precision @K。