中国华录杯-交通卡口车辆信息精准识别

<http://www.dcjingsai.com/common/cmpt/%E4%BA%A4%E9%80%9A%E5%8D%A1%E5%8F%A3%E8%BD%A6%E8%BE%86%E4%BF%A1%E6%81%AF%E7%B2%BE%E5%87%86%E8%AF%86%E5%88%AB_%E7%AB%9E%E8%B5%9B%E4%BF%A1%E6%81%AF.html>

当你选择了一个比赛后，可以先“热热身”，稍微熟悉一下数据，粗略地跑出一些简单的模型，看看自己在榜上的排名，然后再去慢慢迭代。

## 数据增广

### 色相

### 饱和度

### 亮度

### 对比度

### 归一化

上面这几种方式可以在训练时，对图像进行变换，不必单独制作数据集。比较方便。

下面这几种方式需要修改 annotation，工作量比较大。

### 水平翻转

### 裁剪

### 缩放

## 训练策略

### 课程学习：先易后难

把整个数据集分成一系列子集， 将子集按简单干净数据到复杂有噪数据进行排序，然后在模型训练过程中，让网络从简单干净的数据集开始学习，通过增加数据集的难度逐渐提高模型的能力。

如何进行学习进程的切换以及学习时间段和学习任务是如何对应的呢？

首先在第一个子集上训练一个模型，然后将前两个子集合起来作为训练集， 并在第一个模型的基础上再训练， 最后将三个子集结合在一起， 并在第二个模型的基础上再训练。我们在训练过程中给三个子集的样本设置不同的权重（1，0.5，0.5），更多依赖于置信度高的子集以提高模型的准确率。

每个mini-batch里面，将第一子集图片个数定义为128，其他两个分别为64，在训练过程中它们的比重是相同的；

### 模型融合

我们使用了一个十分庞大的卷积神经网络结构，包含超过 100 个改造过的卷积神经网络和类似于 VGG 的结构，然后我们使用 greedy blending 策略和两层结合了其他的图像特征的集成学习算法将模型结果融合起来。

确保所有的模型都要经过4折交叉验证，尽管这会增加计算开销，但是却能降低过拟合的风险。使用固定的k-fold 避免Valid Set过拟合。

## 模型融合

### 一级算法：内部融合权重

#### YOLOv3——召回率太低0.71

YOLOv3 tensorflow 实现 Keras

#### R-FCN

#### Faster R-CNN

### 最终结果：bbox计算相似度，求平均

# 实验结果记录

## 小数据集 train\_b =1000 faster\_rcnn\_resnet101最高

### Yolo3：测试快 1.3 个/秒

--learning\_rate=1e-3 --batch\_size=2 冻结模式

ehualu\_000/ep018-loss36.388-val\_loss33.409.h5 = summit\_yolo3\_000\_ep018.csv = 0.450568

ehualu\_001/ep027-loss32.777-val\_loss28.460.h5 = summit\_yolo3\_001\_ep027.csv = 0.493876

--learning\_rate=**1e-4** --batch\_size=2 冻结模式**=降低学习率更好**

ehualu\_003/ep018-loss33.639-val\_loss29.568.h5 = summit\_yolo3\_003\_ep018.csv = 0.605286

ehualu\_003/ep030-loss32.399-val\_loss28.667.h5 = summit\_yolo3\_003\_ep030.csv = 0.603008

--learning\_rate=1e-3 --batch\_size=**1** 冻结模式**=降低batchSzie 不好**

ehualu\_004/ep003-loss37.179-val\_loss34.935.h5 = summit\_yolo3\_004\_ep003.csv = 0.25006

ehualu\_004/ep045-loss35.318-val\_loss30.547.h5 = summit\_yolo3\_004\_ep045.csv = 0.26011

--learning\_rate=**1e-4** --batch\_size=**4** 冻结模式**=提高batchSzie 更好**

ehualu\_005/ep003-loss32.284-val\_loss31.022.h5 = summit\_yolo3\_005\_ep003.csv = 0.622305

ehualu\_005/ep027-loss32.829-val\_loss27.931.h5 = summit\_yolo3\_005\_ep027.csv = 0.641009

ehualu\_005/ep045-loss32.360-val\_loss27.721.h5 = summit\_yolo3\_005\_ep045.csv = 0.641117

只解冻1/5 --learning\_rate=1e-4 --batch\_size=4 --batch\_size\_2=2 =更好

ehualu\_006/ep006-loss31.141-val\_loss31.847.h5 = summit\_yolo3\_006\_ep006.csv = 0.631003

ehualu\_006/ep012-loss30.376-val\_loss29.428.h5 = summit\_yolo3\_006\_ep012.csv = 0.630015

还没收敛，loss 还在继续下降

只解冻1/2 --learning\_rate=1e-4 --batch\_size=4 --batch\_size\_2=2 =效果最好

ehualu\_007/trained\_weights\_stage\_1.h5 = summit\_yolo3\_007\_stage\_1.csv = 0.62007

ehualu\_007/ep003-loss30.044-val\_loss27.589.h5 = summit\_yolo3\_007\_ep003.csv = 0.641236

tiny --learning\_rate=1e-4 --batch\_size=4 --batch\_size\_2=2 **=效果太差了**

ehualu\_008/ep060-loss36.011-val\_loss35.004.h5 = summit\_yolo3\_008\_ep060.csv = 0.14006

ehualu\_008/trained\_weights\_final.h5 = summit\_yolo3\_008\_final.csv = 0.150006

### faster\_rcnn\_inception\_resnet\_v2：测试慢 0.08个/秒

Exec#3/model.ckpt-4950 = summit\_frir2\_003\_4950.csv = 0.83005

不转换为整数 summit\_frir2\_003\_4950-float.csv = 0.83002 分数低

Exec#3/model.ckpt-5400 = summit\_frir2\_003\_5400.csv = 0.840194

Exec#15/model.ckpt-13846 = summit\_frir2\_015\_13846.csv = 0.83006 = 没有提升

### faster\_rcnn\_resnet101：测试速度 0.5个/秒

Exec#7/model.ckpt-12686 = summit\_frr101\_007\_12686.csv = 0.85637

Exec#7/model.ckpt-13480 = summit\_frr101\_007\_13480.csv = 0.856898

Exec#7/model.ckpt-14275 = summit\_frr101\_007\_14275.csv = 0.83018

Exec#9/model.ckpt-8100= summit\_frr101\_009\_8100.csv = ？？分数应该略低，暂不测试

Exec#9/model.ckpt-9000 = summit\_frr101\_009\_9000.csv = 0.83106

不转换为整数 summit\_frr101\_009\_9000\_float.csv = 0.83009 分数低

### rfcn\_resnet101：测试速度 0.8个/秒

Exec#10/model.ckpt-9000 = summit\_rfcnr101\_010\_9000.csv = 0.801346 = batch\_size=1

Exec#12/model.ckpt-8679= summit\_rfcnr101\_012\_8679.csv = ？？分数应该略低，暂不测试

Exec#12/model.ckpt-9000 = summit\_rfcnr101\_012\_9000.csv = 0.81003 = batch\_size=2 更好

### ssd\_inception\_v2：性能太差，不用了

## 小数据集增广 train\_b\_aug=10000 faster\_rcnn\_resnet101 最高

train\_b\_aug=1000 \* (1+9) =10000

### Yolo3：测试快 1.3 个/秒

--learning\_rate=**1e-5** --batch\_size=4 --batch\_size\_2=2 --epoch=10 --epoch\_2=20 **解冻1/2**

ehualu\_009/ep018-loss31.269-val\_loss31.500.h5 = summit\_yolo3\_009\_ep018.csv = 0.67003

ehualu\_009/trained\_weights\_final.h5 = summit\_yolo3\_009\_final.csv = 0.67105

### faster\_rcnn\_inception\_resnet\_v2：测试慢 0.08个/秒

Exec#27/model.ckpt-40343= summit\_frir2\_027\_40343.csv = 0.831020

Exec#27/model.ckpt-41097= summit\_frir2\_027\_41097.csv = 0.857898

Exec#27/model.ckpt-41098= summit\_frir2\_027\_41098.csv = 0.835005

### faster\_rcnn\_resnet101：测试速度 0.5个/秒

Exec#28/model.ckpt-28190= summit\_frr101\_028\_28190.csv = 0.852002

Exec#28/model.ckpt-29124= summit\_frr101\_028\_29124.csv = 0.863238

Exec#28/model.ckpt-29700= summit\_frr101\_028\_29700.csv = 0.851008

### rfcn\_resnet101：测试速度 0.8个/秒

Exec#29/model.ckpt-27984= summit\_rfcnr101\_029\_27984.csv = 0.85005

Exec#29/model.ckpt-29700= summit\_rfcnr101\_029\_29700.csv = ？？最后

Exec#29/model.ckpt-29701= summit\_rfcnr101\_029\_29701.csv = 0.841008

## 小数据集+大数据集+都增广 = 41806

train\_b\_1w\_aug=10000+10602 \* (1+2) = 41806

### Yolo3：测试快 1.3 个/秒

--learning\_rate=**1e-5** --batch\_size=4 --batch\_size\_2=2 --epoch=10 --epoch\_2=20 **解冻1/2**

ehualu\_010/trained\_weights\_final.h5 = summit\_yolo3\_010\_final.csv =0.71\*\*\*9

### faster\_rcnn\_inception\_resnet\_v2：测试慢 0.1个/秒

Exec#53/model.ckpt-130000 = summit\_frr101\_053\_130000.csv = 0.？？

Exec#55/model.ckpt-140000 = summit\_frr101\_055\_140000.csv = 0.？？

Exec#55/model.ckpt-139128= summit\_frr101\_055\_139128.csv = 0.？？

### faster\_rcnn\_resnet101：测试速度 0.5个/秒

Exec#57/ final.ckpt= summit\_frr101\_057\_final.csv = 0.866112

Exec#60/model.ckpt-180000= summit\_frr101\_060\_180000.csv = 0.866324

Exec#59/model.ckpt-150000= summit\_frr101\_059\_150000.csv = 0. ？？

Exec#59/model.ckpt-149679= summit\_frr101\_059\_149679.csv = 0. ？？

Exec#58/model.ckpt-140000= summit\_frr101\_058\_140000.csv = 0. ？？

### rfcn\_resnet101：测试速度 1个/秒

## 模型融合：测试结果合并bbox 取平均

### 两个合并，过滤太小的 bbox

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] = merged\_001 = 0.861106

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] 过滤bbox < 5 = merged\_002 = 0.861221

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] 过滤bbox < 10 = merged\_003 = 0.863697 上升

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] 过滤bbox < 11 = merged\_004 = 0.863999 上升

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] **过滤bbox < 12 = merged\_005 = 0.864368 上升**

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] 过滤bbox < 13 = merged\_006 = 0.863876 下降

['summit\_frir2\_027\_41097', 'summit\_frr101\_028\_29124'] 过滤bbox < 14 = merged\_007= 0.863121 下降

### 多个合并

['merged\_005', 'summit\_frir2\_027\_40343'] 过滤bbox < 12 = merged\_008 = 0.85316 下降

['merged\_005', 'summit\_frir2\_027\_40343'] 过滤bbox < 13 = merged\_009 =

['merged\_005', 'summit\_frir2\_027\_40343'] 过滤bbox < 14 = merged\_010 =

### 合并霍炜佳的

['8a4d58e2-eda1-4e4e-a64b-b5118a8499d1', 'e0e6176b-847e-4ad1-b5a7-54d8e78f5ea2'] = merged\_huo\_001 = 0.52

# 日程记录

## 2018.06.18 比赛报名，环境准备

比赛报名，数据探索，比赛方案设计，算法选择。

1、安装 Keras 下载 yolo3 ：

1. git clone https://github.com/qqwweee/keras-yolo3

2、下载yoloV3权重文件：

1. https://pjreddie.com/media/files/yolov3.weights

3、将darknet下的yolov3配置文件转换成keras适用的h5文件：

1. python convert.py yolov3.cfg yolov3.weights model\_data/yolo.h5

4、运行预测图像程序（预测视频程序跳过）：

1. python yolo.py

输入需要预测的图片路径即可。

## 2018.06.21 yolo3训练数据准备

<https://blog.csdn.net/yuhq3/article/details/80281929>

1、将VOC数据集放在代码同一目录下，然后先生成需要进行训练的txt文件，包括图片路径、类别和bbox，一张图片有多个目标时用' '隔开，根据使用VOC数据集的年份修改voc\_annotation.py文件中的sets，根据需要训练的类别修改classes（如注释中只使用person类）。

**[python]** [view plain](https://blog.csdn.net/yuhq3/article/details/80281929) [copy](https://blog.csdn.net/yuhq3/article/details/80281929)

1. sets=[('2007', 'train'), ('2007', 'val'), ('2007', 'test')]
3. # classes = ['person']
4. classes = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

运行生成txt文件程序，根据sets得到三个不同的txt文件，方便用于做训练、验证和测试：

**[python]** [view plain](https://blog.csdn.net/yuhq3/article/details/80281929) [copy](https://blog.csdn.net/yuhq3/article/details/80281929)

1. python voc\_annotation.py
2. python ehualu\_annotation.py

## 2018.06.22 使用train\_b训练yolo3、测试

### 开始训练

2、修改train.py中的annotation\_path为刚刚生成的txt文件，log\_dir记录了每一次迭代生成的模型，classes\_path是需要训练的目标种类，根据上一步修改过的classes调整（如注释中选择的txt文件中只包含自己需要训练的类别），anchors\_path是锚点文件。

1. annotation\_path = './data/new/train\_b\_train.txt'
2. log\_dir = './data/logs/000/'
3. classes\_path = 'model\_data/ehualu\_classes.txt'
4. anchors\_path = 'model\_data/yolo\_anchors.txt'
5. input\_shape = (416,416) # 不要改这里，否则训练报错

同时，在训练时还会存在这样的情况：可能数据集中某些图像中不包含你想要训练的目标，那么可以在生成txt文件时就筛除掉，或者在读取训练数据时跳过没有数据的那一行，这里采用前者，修改ehualu\_annotation.py：

1. # skip the rows contain no target
2. if len(coordinate) < 1:
3. print('%s = %s coordinate= %s' % (i,name,coordinate))
4. continue

这里需要注意的是，如果想要训练某一类或某几类特定的目标类别，必须要同时修改以上两步中ehualu\_annotation.py和train.py相应的位置，否则可能会报维数不符的错误。

1. python ehualu\_train.py
2. Train on 900 samples, val on 100 samples, with batch size 4.
3. Epoch 1/50
4. 225/225 [==============================] - 738s 3s/step - loss: 538.1864 - val\_loss: 91.1437
5. Epoch 2/50
6. 225/225 [==============================] - 713s 3s/step - loss: 77.6870 - val\_loss: 59.7911
7. Epoch 3/50
8. 225/225 [==============================] - 761s 3s/step - loss: 60.6114 - val\_loss: 58.2075
9. Epoch 4/50
10. 225/225 [==============================] - 759s 3s/step - loss: 54.5928 - val\_loss: 49.9354
11. Epoch 5/50
12. 225/225 [==============================] - 764s 3s/step - loss: 49.9323 - val\_loss: 46.5107
13. Epoch 6/50
14. 225/225 [==============================] - 760s 3s/step - loss: 47.8825 - val\_loss: 45.8500
15. Epoch 7/50
16. 225/225 [==============================] - 763s 3s/step - loss: 45.6873 - val\_loss: 44.2186
17. Epoch 8/50
18. 225/225 [==============================] - 760s 3s/step - loss: 43.9960 - val\_loss: 40.3991
19. Epoch 9/50
20. 225/225 [==============================] - 763s 3s/step - loss: 41.8639 - val\_loss: 40.4585
21. Epoch 10/50
22. 225/225 [==============================] - 763s 3s/step - loss: 41.8176 - val\_loss: 35.8561
23. Epoch 11/50
24. 225/225 [==============================] - 763s 3s/step - loss: 40.2615 - val\_loss: 38.5544
25. Epoch 12/50
26. 225/225 [==============================] - 762s 3s/step - loss: 39.3861 - val\_loss: 35.9146
27. Epoch 13/50
28. 225/225 [==============================] - 761s 3s/step - loss: 38.4485 - val\_loss: 36.4640
29. Epoch 14/50
30. 225/225 [==============================] - 762s 3s/step - loss: 37.9916 - val\_loss: 35.7364
31. Epoch 15/50
32. 225/225 [==============================] - 719s 3s/step - loss: 37.6281 - val\_loss: 36.4477
33. Epoch 16/50
34. 225/225 [==============================] - 722s 3s/step - loss: 37.3095 - val\_loss: 37.0751

如果想使用训练好的权重继续训练，在train.py中的create\_model函数中修改weights\_path。如果不想使用预训练的权重，设置传给create\_model函数的参数load\_pretrained=False。

另外train.py 不支持外部传参，不方便

weights\_path='model\_data/ehualu\_yolo\_weights.h5'

### 进行测试

3、修改yolo.py，使用自己训练的模型预测图像：

1. #self.model\_path = 'model\_data/yolo.h5'
2. self.model\_path = 'model\_data/my\_yolo.h5'
3. self.anchors\_path = 'model\_data/yolo\_anchors.txt'
4. #self.classes\_path = 'model\_data/coco\_classes.txt'
5. self.classes\_path = 'model\_data/my\_classes.txt'

# 测试，直接生成 summit 文件

python ehualu\_yolo.py --model\_path=./data/logs/ehualu\_000/ep018-loss36.388-val\_loss33.409.h5 --classes\_path=./data/ehualu/ehualu\_classes.txt --img\_root=./data/ehualu/raw/test\_a --summit\_dir=./data/ehualu/summit

### 支持外部传参

#标签数据转换

python ehualu\_annotation.py --data\_group=train\_b --raw\_dir=data/ehualu/raw --new\_dir=data/ehualu/new

#训练

python ehualu\_train.py --annotation\_path=data/ehualu/new/train\_b\_train.txt --log\_dir=data/logs/ehualu\_001 --classes\_path=data/ehualu/ehualu\_classes.txt --weights\_path=data/logs/ehualu\_000/ep018-loss36.388-val\_loss33.409.h5 --batch\_size\_1=4 --batch\_size\_2=2 --epoch\_1=1 --epoch\_2=2

### yolo3 读取zip压缩包中的图片

还没找到方法

### yolo3 使用GPU训练

还没找到方法

## 2018.06.23 Faster R-CNN

### Faster R-CNN 准备数据

tensorflow-models/research/ 执行脚本读取图片和标签数据，生成 TFRecord 文件:

python object\_detection/dataset\_tools/create\_ehualu\_tf\_record.py \

--label\_map\_path=data/jieming2002/ehualu/labels\_items.txt \

--data\_dir=/home/skye/anaconda3/envs/tensorflow/keras-yolo3/data/ehualu/raw/ \

--output\_dir=data/jieming2002/ehualu/ --group=train\_b

### Faster R-CNN 进行训练

#### 编辑pipline.config文件

以models/research/object\_detection/samples/configs/ faster\_rcnn\_resnet101\_coco.config为基础进行修改。

**注意要点：**

* num\_classes， 原文件里面为90,这里的数据集为1
* num\_examples， 这个是验证集中有多少数量的图片，请根据图片数量和数据准备脚本中的生成规则自行计算。
* PATH\_TO\_BE\_CONFIGURED，这个是原文件中预留的字段，一共5个，分别包含预训练模型的位置，训练集数据和label\_map文件位置，验证集数据和label\_map文件位置。这个字段需要将数据以及配置文件等上传到tinymind之后才能确定路径的具体位置。
* num\_steps，这个是训练多少step，后面的训练启动脚本会用到这个字段，直接将原始的200000改成0.注意不要添加或者删除空格等，后面的训练启动脚本使用sed对这个字段进行检测替换，如果改的有问题会影响训练启动脚本的执行。不通过run.sh本地运行需要将这个数字改成一个合适的step数，改成0的话会有问题。
* max\_evals，这个是验证每次跑几轮，这里直接改成1即可，即每个训练验证循环只跑一次验证。
* eval\_input\_reader 里面的shuffle， 这个是跟eval步骤的数据reader有关，如果不使用GPU进行训练的话，这里需要从false改成true，不然会导致错误，详细内容参阅 <https://github.com/tensorflow/models/issues/1936>

训练和验证过程次数相关的参数，后面在训练启动脚本中会自动进行处理，这里不需要过多关注，但是实际使用的时候，需要对这些参数进行合适的设置，比如**num\_steps**参数，后面的训练启动脚本中，每轮运行100个step，同时根据数据集图片总数all\_images\_count和batch\_size的大小，可以计算出epoch的数量，最后输出模型的质量与epoch的数量密切相关。epoch=num\_step\*batch\_size/all\_images\_count

config文件需要跟代码一起上传，运行的时候会先被复制到output文件夹里面。

python object\_detection/dataset\_tools/create\_ehualu\_tf\_record.py \

--label\_map\_path=data/ehualu/labels\_items.txt \

--data\_dir=/home/skye/anaconda3/envs/tensorflow/keras-yolo3/data/ehualu/raw/ \

--output\_dir=data/ehualu/ --group=train\_b

#### 修改run.sh

修改数据集目录为自己的数据集目录。dataset\_dir

# config文件

config=faster\_rcnn\_resnet101\_ehualu\_local.config

#### 执行训练脚本

*# 本地运行训练*

**python** ./object\_detection/train.**py** --train\_dir**=.**/output/train/ \

--pipeline\_config\_path**=.**/output/faster\_rcnn\_resnet101\_ehualu\_local.config

*#本地运行验证，注意这里的checkpoint就是训练时候的train\_dir,验证的结果放在eval\_dir，可以通过tensorboard看*

**python** ./object\_detection/eval.**py** --checkpoint\_dir**=.**/output/train/ --eval\_dir**=.**/output/*eval*/ --pipeline\_config\_path**=.**/output/faster\_rcnn\_resnet101\_ehualu\_local.config

*# 导出训练好的模型，这里注意trained\_checkpoint\_prefix的参数是个前缀，不是一个文件名，这个参数代表的是三个文件以这个参数值开头的文件，最后的last\_chekpoint\_number是一个数字*

**python** ./object\_detection/export\_inference\_graph.**py** --input\_type image\_tensor --pipeline\_config\_path **.**/output/faster\_rcnn\_resnet101\_ehualu\_local.config --trained\_checkpoint\_prefix **.**/output/train/model.ckpt-0 --output\_directory **.**/output/*eval*/

#### 训练报错

ImportError: cannot import name 'preprocessor\_pb2'

This particular preprocessor\_pb2.py exists in the path it is looking for i.e

C:\Users\SP-TestMc\Anaconda3\envs\tensorflow\models-master\models-master\research\object\_detection\protos

解决方法：

See <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md#protobuf-compilation>. Sounds like you need to compile the protos before using the Python scripts.

### Faster R-CNN 进行测试

直接测试，看看效果。单张图片，生成 summit 文件。

*# 用导出的模型运行inference，详情参考代码*

**python** object\_detection/inference.**py** --output\_dir**=.**/output/*infer*/ --dataset\_dir**=.**/data/*jieming2002*/ehualu/

### yolo3 读取zip压缩包中的图片

找到方法了，哈哈哈。

设置两种模式：压缩包，文件夹。(取决于 zip\_path 是否传递)

python ehualu\_train.py --annotation\_path=data/ehualu/new/train\_b\_train\_20.txt --log\_dir=data/logs/ehualu\_002/ --classes\_path=data/ehualu/ehualu\_classes.txt --weights\_path=data/logs/ehualu\_001/stage\_1\_loss35.4338-val\_loss29.9655.h5 --batch\_size\_1=2 --batch\_size\_2=1 --epoch\_1=1 --epoch\_2=2 --zip\_path=data/ehualu/new/train\_b.zip

### yolo3 使用GPU训练

<https://blog.csdn.net/u012135425/article/details/80294884>

只有 darknet 版本，没有 Keras 版本

<http://blog.sina.com.cn/s/blog_973657a00102x1kg.html>

如果采用TensorFlow作为后端，当机器上有可用的GPU时，代码会自动调用GPU进行并行计算。

<https://blog.csdn.net/odailidong/article/details/79332750>

让keras训练深度网络时使用多个显卡

### yolo3 自动读取输出权重进行迁移训练

还没找到方法。暂时采用手动，每次下载最新的权重，上传数据集。

## 2018.06.24 在tinymind训练 yolo3

新建模型 race003-ehualu-1

### 支持多个训练集

用groups=train\_b,train\_c 指定，每个训练集两个文件：

图片压缩包 {group}.zip、图片标签 {group}\_annotation.txt. 例如：train\_b.zip、train\_b\_annotation.txt

【注意】需要图片标签文件中，包含图片名称及其group，比如 {group}/{imageName} 而且 group 必须和图片所在压缩包名称一致。

### 新建数据集

模型权重，用于保存训练好的权重文件：race003-ehualu-yolo3-model

训练数据，用于保存多组数据文件：race003-ehualu-yolo3-data

## 2018.06.25 在tinymind训练 yolo3



### 报错1

model\_body.load\_weights(weights\_path, by\_name=True, skip\_mismatch=True)

TypeError: load\_weights() got an unexpected keyword argument 'skip\_mismatch'

Tinymind 不支持这个参数 skip\_mismatch ，解决方法，去掉这个参数。

### 报错2

raise ValueError(err.message)

ValueError: Shape must be rank 0 but is rank 5 for 'yolo\_loss/cond/Switch' (op: 'Switch') with input shapes: [?,13,13,3,1], [?,13,13,3,1].

权重shape 不匹配。

解决方法 <https://github.com/keras-team/keras/issues/1873>

没有太好的解决方法。只有从头开始训练，不再加载预先训练的权重。

### 还是报错

load\_pretrained = **False 不加载预训练的权重也不行**

Traceback (most recent call last):

File "/tinysrc/research/keras-yolo3/ehualu\_train.py", line 320, in

\_main()

File "/tinysrc/research/keras-yolo3/ehualu\_train.py", line 39, in \_main

freeze\_body=2, weights\_path=FLAGS.weights\_path) # make sure you know what you freeze

File "/tinysrc/research/keras-yolo3/ehualu\_train.py", line 170, in create\_model

[\*model\_body.output, \*y\_true])

File "/opt/conda/lib/python3.6/site-packages/keras/engine/topology.py", line 596, in \_\_call\_\_

output = self.call(inputs, \*\*kwargs)

File "/opt/conda/lib/python3.6/site-packages/keras/layers/core.py", line 647, in call

return self.function(inputs, \*\*arguments)

File "/tinysrc/research/keras-yolo3/yolo3/model.py", line 382, in yolo\_loss

raw\_true\_wh = K.switch(object\_mask, raw\_true\_wh, K.zeros\_like(raw\_true\_wh)) # avoid log(0)=-inf

File "/opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow\_backend.py", line 2577, in switch

else\_expression\_fn)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/util/deprecation.py", line 289, in new\_func

return func(\*args, \*\*kwargs)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/control\_flow\_ops.py", line 1803, in cond

p\_2, p\_1 = switch(pred, pred)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/control\_flow\_ops.py", line 302, in switch

return gen\_control\_flow\_ops.\_switch(data, pred, name=name)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/gen\_control\_flow\_ops.py", line 356, in \_switch

result = \_op\_def\_lib.apply\_op("Switch", data=data, pred=pred, name=name)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op\_def\_library.py", line 767, in apply\_op

op\_def=op\_def)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 2508, in create\_op

set\_shapes\_for\_outputs(ret)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 1873, in set\_shapes\_for\_outputs

shapes = shape\_func(op)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 1823, in call\_with\_requiring

return call\_cpp\_shape\_fn(op, require\_shape\_fn=True)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/common\_shapes.py", line 610, in call\_cpp\_shape\_fn

debug\_python\_shape\_fn, require\_shape\_fn)

File "/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/common\_shapes.py", line 676, in \_call\_cpp\_shape\_fn\_impl

raise ValueError(err.message)

ValueError: Shape must be rank 0 but is rank 5 for 'yolo\_loss/cond/Switch' (op: 'Switch') with input shapes: [?,13,13,3,1], [?,13,13,3,1].

### 测试支持读取zip文件

这样就能在 tinymind 生成 summit 文件。

【注意】：图片压缩包必须是zip文件，且压缩图片时不能包含目录，只包含图片。

## 2018.06.26 霍炜佳经验，准备 faster\_rcnn

gen\_scores.py 这个算法改造一下就能进行模型融合，加权平均。需要加一个算法，怎么判断同一个框。可以使用IOU来计算，交并比。交并比大于一定的数值，则认为是同一个框。

clean\_data.py 用于检测训练集噪声。用预测结果和训练集比较。

**我测试了 3 个模型，faster\_rcnn\_inception\_resnet\_v2 》 faster\_rcnn\_resnet101》faster\_rcnn\_nas**

**Nasnet 理论上会更高，结果居然最低。**

我跑6个 epoch 居然要3天3夜。faster\_rcnn\_nas

我最高分也只跑了 2 个 epoch。 faster\_rcnn\_inception\_resnet\_v2

不训练，直接检测，得分 53

### faster\_rcnn\_inception\_resnet\_v2

准备训练数据，本地跑起来，明天上传 tinymind 训练。

### faster\_rcnn\_resnet101

准备训练数据，本地跑起来，明天上传 tinymind 训练。

## 2018.06.27 tinymind跑起来

faster\_rcnn\_inception\_resnet\_v2，train\_b

<https://www.tinymind.com/jieming2002/race003-ehualu-2>

### Exec #1 跑起来，有报错

faster\_rcnn\_inception\_resnet\_v2\_ehualu

Exec #2 没有报错了

### Exec #3 基于 #2 =

faster\_rcnn\_inception\_resnet\_v2\_ehualu

step\_num=450

for i in {0..11} steps = 450\*12 = 5400

目测，准确度很高，测试集出个结果，看看评分：



### Exec #4 基于 #2 =报错啦，内存溢出

能否接受多个训练集文件： **不行！！！**

train\_input\_reader {  
 label\_map\_path: "/data/jieming2002/race003-ehualu-object-detection/labels\_items.txt"  
 tf\_record\_input\_reader {  
 # input\_path: "/data/jieming2002/race003-ehualu-object-detection/train\_b\_train.record"  
 input\_path: ["/data/jieming2002/race003-ehualu-object-detection/train\_b\_train.record"  
 , "/data/jieming2002/race003-ehualu-object-detection/train\_b\_train.record"]  
 }  
}

config=**faster\_rcnn\_inception\_resnet\_v2\_ehualu3**.config

batch\_size: 2

step\_num=450

for i in {0..6}

### Exec #5 基于 #2 =报错啦，内存溢出

config=**faster\_rcnn\_inception\_resnet\_v2\_ehualu2**.config

batch\_size: 2 **不行！！！**

step\_num=450

for i in {0..6}

### 本地yolo3继续训练

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_003/ --dataset\_path=data/ehualu/new/ --groups=train\_b --weights\_path=data/logs/ehualu\_001/stage\_1\_loss35.4338-val\_loss29.9655.h5 --learning\_rate=**1e-4** --batch\_size=2 --batch\_size\_2=1 --epoch=100 --epoch\_2=2

**得分取得突破 summit\_yolo3\_003\_ep018.csv = 0.605286**

## 2018.06.28 修改yolo3结构

### 本地yolo3继续训练

batch\_size=1 看看 ehualu\_004

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_004/ --dataset\_path=data/ehualu/new/ --groups=train\_b --weights\_path=data/logs/ehualu\_001/stage\_1\_loss35.4338-val\_loss29.9655.h5 --learning\_rate=**1e-3** --batch\_size=**1** --batch\_size\_2=1 --epoch=50 --epoch\_2=2

### tinymind 继续训练

#### Exec #6 faster\_rcnn\_resnet101

数据集不打勾不行。

#### Exec #7 faster\_rcnn\_resnet101

faster\_rcnn\_resnet101\_ehualu.config

batch\_size=1 step\_num=900 for i in {0..9} 左右边界都包含 steps = 900\*10 = 9000

**一直运行 14000 还没停止 train 0，提前终止**

**加上**num\_steps: 0  **好了，** faster\_rcnn\_resnet101\_ehualu2.config

#### Exec #8 faster\_rcnn\_resnet101

faster\_rcnn\_resnet101\_ehualu2.config

数据集不打勾不行。

#### Exec #9 faster\_rcnn\_resnet101

faster\_rcnn\_resnet101\_ehualu2.config

batch\_size=**2** step\_num=450 for i in {0..19} 左右边界都包含 steps = 450\*20 = 9000

可以跑起来，一切正常。和 Exec #7 比较一下效果。

目测，准确度很高，测试集出个结果，看看评分：



#### Exec #10 rfcn\_resnet101

rfcn\_resnet101\_ehualu.config

batch\_size=1 step\_num=900 for i in {0..9} 左右边界都包含 steps = 900\*10 = 9000

## 2018.06.30 tinymind 继续训练

### tinymind 继续训练

#### Exec #11 rfcn\_resnet101

rfcn\_resnet101\_ehualu2.config

【注意】更新数据集时，页面显示文件不全，需要删除数据集，重新添加、打钩文件才行。

#### Exec #12 rfcn\_resnet101

rfcn\_resnet101\_ehualu2.config

batch\_size=2 step\_num=900 for i in {0..9} 左右边界都包含 steps = 900\*10 = 9000

#### Exec #13 ssd\_inception\_v2

ssd\_inception\_v2\_ehualu.config

batch\_size=8 step\_num=900 for i in {0..9} 左右边界都包含 steps = 900\*10 = 9000

#### Exec #14 ssd\_inception\_v2

ssd\_inception\_v2\_ehualu2.config

batch\_size=1 step\_num=900 for i in {0..9} 左右边界都包含 steps = 900\*10 = 9000

## 2018.07.01 数据增广功能实现

### tinymind 继续训练

#### Exec #15 基于 #3 =

faster\_rcnn\_inception\_resnet\_v2\_ehualu.config

step\_num=900

for i in {6..14} steps = 450\*12 = 13500

**一直运行 14000 还没停止，提前终止**

### train\_b 数据增广

https://www.cnblogs.com/Magina-learning/p/7927666.html

<http://baijiahao.baidu.com/s?id=1602763169790772339&wfr=spider&for=pc>

https://blog.csdn.net/medium\_hao/article/details/79227056

在训练过程中，对图像进行变换，不再单独生成图片和训练集。节省存储，节省工作量。用一个训练集，达到10个训练集的效果。

Object detection 框架不知道修改哪里。Yolo3 应该可以。

但是这种方式，不方便控制数据集。

暂时使用生成实际图片的方式。制作多个数据集。

## 2018.07.02 数据增广

### 生成新图片train\_b

### 生成对应的 Annotation

### 制作object detection 数据集

图片必须是 JPEG 格式

### 制作yolo3数据集

## 2018.07.03 数据增广

### 生成新图片train\_1w

### 制作object detection 数据集

### 制作yolo3数据集

## 2018.07.04 修改yolo3结构，训练train\_b\_aug

### 修改yolo3结构，看看能否跑起来

#### 只解冻1/5 可以

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_006/ --dataset\_path=data/ehualu/new/ --groups=train\_b --weights\_path=data/logs/ehualu\_003/ep018-loss33.639-val\_loss29.568.h5 --learning\_rate=1e-4 --batch\_size=4 --batch\_size\_2=2 --epoch=1 --epoch\_2=12

#### 只解冻1/2 可以，半路挂了

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_007/ --dataset\_path=data/ehualu/new/ --groups=train\_b --weights\_path=data/logs/ehualu\_003/ep018-loss33.639-val\_loss29.568.h5 --learning\_rate=1e-4 --batch\_size=4 --batch\_size\_2=2 --epoch=1 --epoch\_2=30

Train on 900 samples, val on 100 samples, with batch size 4.

Epoch 1/1

225/225 [==============================] - 741s 3s/step - loss: 32.3879 - val\_loss: 31.5581

total\_layers=256 from\_layer=128 to\_layer=256

Unfreeze all of the layers.

Train on 900 samples, val on 100 samples, with batch size 2.

Epoch 2/30

450/450 [==============================] - 1638s 4s/step - loss: 31.8285 - val\_loss: 32.0548

Epoch 3/30

450/450 [==============================] - 1764s 4s/step - loss: 30.0436 - val\_loss: 27.5891

Epoch 4/30

450/450 [==============================] - 1852s 4s/step - loss: 30.2207 - val\_loss: 27.3706

Epoch 5/30

450/450 [==============================] - 1748s 4s/step - loss: 29.9700 - val\_loss: 27.0858

Epoch 6/30

450/450 [==============================] - 1750s 4s/step - loss: 29.6433 - val\_loss: 30.6468

Epoch 7/30

450/450 [==============================] - 1754s 4s/step - loss: 29.3645 - val\_loss: 30.8025

Epoch 8/30

450/450 [==============================] - 1753s 4s/step - loss: 28.6936 - val\_loss: 31.0346

Epoch 00008: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-07.

Epoch 9/30

450/450 [==============================] - 1754s 4s/step - loss: 28.5728 - val\_loss: 28.7129

Epoch 10/30

450/450 [==============================] - 1772s 4s/step - loss: 29.0925 - val\_loss: 27.6546

Epoch 11/30

450/450 [==============================] - 1755s 4s/step - loss: 28.6461 - val\_loss: 28.5059

Epoch 00011: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.

Epoch 12/30

450/450 [==============================] - 1793s 4s/step - loss: 29.0836 - val\_loss: 28.3152

#### 短粗结构：tiny

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/tiny\_yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_008/ --dataset\_path=data/ehualu/new/ --groups=train\_b --tiny\_weights\_path=data/logs/ehualu\_003/ep018-loss33.639-val\_loss29.568.h5 --learning\_rate=1e-4 --batch\_size=4 --batch\_size\_2=2 --epoch=50 --epoch\_2=100

### Tinymind继续训练train\_b\_aug=10000

#### Exec#17基于Exec#3/ckpt-5400

faster\_rcnn\_inception\_resnet\_v2\_ehualu4.config

基于已有的运行，数据集也要添加原始 checkpoint 文件。



GPU 显存溢出。

dwSun：

又是gpu1、似乎这个gpu1有问题。

## 2018.07.06训练 train\_b + aug\_train\_b

### 本地yolo3 解冻1/2

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_009/ --dataset\_path=data/ehualu/new/ --groups=train\_b,aug\_train\_b --weights\_path=data/logs/ehualu\_007/ep003-loss30.044-val\_loss27.589.h5 --learning\_rate=1e-5 --batch\_size=4 --batch\_size\_2=2 --epoch=10 --epoch\_2=20

Train on 9000 samples, val on 1000 samples, with batch size 4.

Epoch 1/10

2250/2250 [==============================] - 7526s 3s/step - loss: 34.0740 - val\_loss: 35.4848

Epoch 2/10

2250/2250 [==============================] - 7635s 3s/step - loss: 34.1370 - val\_loss: 35.2911

Epoch 3/10

2250/2250 [==============================] - 7606s 3s/step - loss: 34.1101 - val\_loss: 35.2587

Epoch 4/10

2250/2250 [==============================] - 7631s 3s/step - loss: 34.1616 - val\_loss: 35.1740

Epoch 5/10

2250/2250 [==============================] - 7642s 3s/step - loss: 34.3423 - val\_loss: 35.2170

Epoch 6/10

2250/2250 [==============================] - 7551s 3s/step - loss: 34.0074 - val\_loss: 34.3791

Epoch 7/10

2250/2250 [==============================] - 7571s 3s/step - loss: 33.8050 - val\_loss: 34.9258

Epoch 8/10

2250/2250 [==============================] - 7552s 3s/step - loss: 33.8284 - val\_loss: 35.3938

Epoch 9/10

2250/2250 [==============================] - 7543s 3s/step - loss: 33.7434 - val\_loss: 35.6715

Epoch 10/10

2250/2250 [==============================] - 7546s 3s/step - loss: 33.8684 - val\_loss: 35.1009

total\_layers=256 from\_layer=128 to\_layer=256

Unfreeze all of the layers.

Train on 9000 samples, val on 1000 samples, with batch size 2.

Epoch 11/20

4500/4500 [==============================] - 17478s 4s/step - loss: 33.5910 - val\_loss: 33.3894

Epoch 12/20

4500/4500 [==============================] - 17449s 4s/step - loss: 32.8295 - val\_loss: 32.8140

Epoch 13/20

4500/4500 [==============================] - 17448s 4s/step - loss: 32.2205 - val\_loss: 32.1839

Epoch 14/20

4500/4500 [==============================] - 17434s 4s/step - loss: 32.0574 - val\_loss: 32.6646

Epoch 15/20

4500/4500 [==============================] - 17458s 4s/step - loss: 31.6238 - val\_loss: 33.0535

Epoch 16/20

4500/4500 [==============================] - 17440s 4s/step - loss: 31.5213 - val\_loss: 31.9808

Epoch 17/20

4500/4500 [==============================] - 17396s 4s/step - loss: 31.1520 - val\_loss: 32.8945

Epoch 18/20

4500/4500 [==============================] - 17422s 4s/step - loss: 31.2693 - val\_loss: 31.4996

Epoch 19/20

4500/4500 [==============================] - 17423s 4s/step - loss: 31.1912 - val\_loss: 32.5148

Epoch 20/20

4500/4500 [==============================] - 17421s 4s/step - loss: 30.7935 - val\_loss: 31.5388

training complete!

第一阶段loss几乎不变，第二阶段loss一直在下降，还有下降空间。

### Tinymind继续训练

#### Exec#26基于Exec#3/ckpt-5400

faster\_rcnn\_inception\_resnet\_v2\_ehualu6.config

step\_num=9000

for i in {0..4} steps = 9000\*5 = 45000

指定多个训练文件的方式不对

<https://github.com/tensorflow/models/issues/3031>

You can simply assign list of the file path by changing config file

from

train\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "PATH\_TO\_BE\_CONFIGURED/train.record"

}

label\_map\_path: "PATH\_TO\_BE\_CONFIGURED/label\_map.pbtxt"

}

to

train\_input\_reader: {

tf\_record\_input\_reader {

input\_path: ["PATH\_TO\_BE\_CONFIGURED/train\_a.record",

"PATH\_TO\_BE\_CONFIGURED/train\_b.record"]

}

label\_map\_path: "PATH\_TO\_BE\_CONFIGURED/label\_map.pbtxt"

}

this change may only work when multiple tfrecord files use the same label\_map.

[izzrak](https://github.com/izzrak) commented [on 28 Dec 2017](https://github.com/tensorflow/models/issues/3031#issuecomment-354221660) • edited

|  |
| --- |
| This issue is closed since I found answers in the code  The object detection API use parallel reader to import your dataset, here are the comments by the developer  Usage:  data\_sources = ['path\_to/train\*']  key, value = parallel\_read(data\_sources, tf.CSVReader, num\_readers=4)  Args:  data\_sources: a list/tuple of files or the location of the data, i.e.  /path/to/train@128, /path/to/train\* or /tmp/.../train\*  So basically you can define a list of input path as [**@byungjae89**](https://github.com/byungjae89) mentioned above, or simply provide the input directory like  input\_path: my\_dataset/train/\*  The reader will read the entire folder for you. |

#### Exec#27基于Exec#3/ckpt-5400

faster\_rcnn\_inception\_resnet\_v2\_ehualu7.config #训练集 train\_b\_aug

step\_num=9900

for i in {0..4} steps = 9000\*5 = 49500

#### Exec#28基于Exec#9/ckpt-9000

faster\_rcnn\_resnet101\_ehualu3.config

batch\_size: 2

step\_num=9900

for i in {0..2} steps = 9900\*3 = 29700

#### 基于Exec#7/ckpt-13480 最后

faster\_rcnn\_resnet101\_ehualu.config

#### Exec#29基于Exec#12/ckpt-9000

rfcn\_resnet101\_ehualu3.config

step\_num=9900

for i in {0..2} steps = 9900\*3 = 29700

### 准备 train\_1w 训练数据=完成

## 2018.07.09 训练train\_b\_1w\_aug：41806

### 本地yolo3 解冻1/2，时间很长，放到空闲时间

#### 训练1 先分别来10+10个 epoch

python keras-yolo3/ehualu\_train.py --anchors\_path=keras-yolo3/model\_data/yolo\_anchors.txt --classes\_path=data/ehualu/ehualu\_classes.txt --output\_dir=data/logs/ehualu\_010/ --dataset\_path=data/ehualu/new/ --groups=train\_b,aug\_train\_b,train\_1w,aug\_train\_1w \

--weights\_path=data/logs/ehualu\_009/trained\_weights\_final.h5 --learning\_rate=1e-5 --batch\_size=4 --batch\_size\_2=2 --epoch=10 --epoch\_2=20

2018.07.14 09:10 大数据集，先分别来10个 epoch 共耗时 = 55+125 = 180 小时

Freeze the first 249 layers of total 252 layers.

Train on 23947 samples, val on 241 samples, with batch size 4.

Epoch 1/10

5986/5986 [==============================] - 18190s 3s/step - loss: 32.1690 - val\_loss: 30.2492

Epoch 2/10

Epoch 16/20

11973/11973 [==============================] - 44674s 4s/step - loss: 31.4602 - val\_loss: 29.7939

Epoch 00016: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.

Epoch 17/20

11973/11973 [==============================] - 44494s 4s/step - loss: 31.3518 - val\_loss: 29.4434

Epoch 18/20

11973/11973 [==============================] - 44669s 4s/step - loss: 31.3089 - val\_loss: 30.9939

Epoch 19/20

11973/11973 [==============================] - 44686s 4s/step - loss: 31.2888 - val\_loss: 31.5748

Epoch 20/20

11973/11973 [==============================] - 45152s 4s/step - loss: 31.2354 - val\_loss: 30.5017

Epoch 00020: ReduceLROnPlateau reducing learning rate to 1.0000000116860975e-08.

training complete!

#### 训练2 再训练1+10个epoch

### Tinymind继续训练

#### Exec#30 基于Exec#28

faster\_rcnn\_resnet101\_ehualu4.config

batch\_size: 2

step\_num=41388

for i in {0..2} steps = 41388\*3 = 124164, 90012 **提前终止了**

#### Exec#31 基于Exec#30 =训练完成

faster\_rcnn\_resnet101\_ehualu5.config

batch\_size: 2 调整学习率 90000，150000，降低

step\_num=41388

for i in {0..2} steps = 41388\*3 = 124164

#### Exec#32 基于Exec#27

faster\_rcnn\_inception\_resnet\_v2\_ehualu8.config

step\_num=41388 **提前终止 75555 没有收敛，不用测试**

for i in {0..2} steps = 41388\*3 = 124164

#### Exec#47基于Exec#32 =满24小时 = 113080终止

#### Exec#48基于Exec#47 = 训练完成

faster\_rcnn\_inception\_resnet\_v2\_ehualu8.config

step\_num=41388 调整学习率 100000，150000，降低

for i in {0..2} steps = 41388\*3 = 124164

#### Exec#42基于Exec#29 环境改成TensorFlow 1.5 可以了

rfcn\_resnet101\_ehualu4.config

step\_num=41388 调整学习率 100000，150000，降低

for i in {0..2} steps = 41388\*3 = 124164 **提前终止了**

#### Exec#46基于Exec#42 =训练完成

rfcn\_resnet101\_ehualu4.config

step\_num=41388 调整学习率 100000，150000，降低

for i in {0..2} steps = 41388\*3 = 124164

## 2018.07.18 小步训练 step\_num=10000

确认训练生成的pb文件是否和最后checkpoint效果一致——测试pb失败，可能是因为训练使用 TensorFlow1.5

确认导出文件夹的checkpoint是否和最后checkpoint效果一致。前者比后者小。——一致

如果一致，则只下载训练完成生成的pb文件即可。

#### Exec#57 基于Exec#31 faster\_rcnn\_resnet101 完成

faster\_rcnn\_resnet101\_ehualu5.config

batch\_size: 2 调整学习率 90000，150000

step\_num=65000

for i in {0..1} # steps = 65000\*2 = 130000 **TotalLoss=**0.10

#### Exec#58 基于Exec#57 完成

faster\_rcnn\_resnet101\_ehualu5.config

batch\_size: 2 调整学习率 90000，150000

step\_num=70000

for i in {0..1} # steps = 70000\*2 = 140000 **TotalLoss=**0.42

#### Exec#59 基于Exec#58 完成

faster\_rcnn\_resnet101\_ehualu5.config

batch\_size: 2 调整学习率 90000，150000

step\_num=75000

for i in {0..1} # steps = 75000\*2 = 150000 **TotalLoss=**0.17

#### Exec#60 基于Exec#59 完成

faster\_rcnn\_resnet101\_ehualu5.config

batch\_size: 2 调整学习率 90000，150000

step\_num=90000

for i in {0..1} # steps = 90000\*2 = 180000 **TotalLoss=**0.24

#### Exec#72 基于Exec#60 进行中

faster\_rcnn\_resnet101\_ehualu5.config

batch\_size: 2 调整学习率 90000，150000

step\_num=41300

for i in {0..5} # steps = 41300\*6 = 247800 **TotalLoss=**0.？？

#### Exec#53 基于Exec#48 faster\_rcnn\_inception\_resnet 完成

Tinymind 运行失败：Unable to gather files or build execution module.

解决方法，切换到 TensorFlow 1.4 再切换回 TensorFlow 1.5 就好了

faster\_rcnn\_inception\_resnet\_v2\_ehualu8.config

step\_num=65000

for i in {0..1} # steps = 65000\*2 = 130000 **TotalLoss=**0.09

#### Exec#55 基于Exec#53 完成

faster\_rcnn\_inception\_resnet\_v2\_ehualu8.config

step\_num=70000

for i in {0..1} # steps = 70000\*2 = 140000 **TotalLoss=**0.15

#### Exec#56 基于Exec#55 完成

faster\_rcnn\_inception\_resnet\_v2\_ehualu8.config

step\_num=75000

for i in {0..1} # steps = 75000\*2 = 150000 **TotalLoss=**0.21

#### Exec#62 基于Exec#46 rfcn\_resnet101 完成

rfcn\_resnet101\_ehualu4.config

step\_num=65000

for i in {0..1} # steps = 65000\*2 = 130000 **TotalLoss=**0.14

#### Exec#64 基于Exec#62 完成

rfcn\_resnet101\_ehualu4.config

step\_num=70000

for i in {0..1} # steps = 70000\*2 = 140000 **TotalLoss=**0.11

#### Exec#66 基于Exec#64 完成

rfcn\_resnet101\_ehualu4.config

step\_num=75000

for i in {0..1} # steps = 75000\*2 = 150000 **TotalLoss=**0.11

#### Exec#67 基于Exec#66 完成

rfcn\_resnet101\_ehualu4.config

step\_num=80000

for i in {0..1} # steps = 80000\*2 = 160000 **TotalLoss=**0.09

#### Exec#68 基于Exec#67 完成

rfcn\_resnet101\_ehualu4.config

step\_num=85000

for i in {0..1} # steps = 85000\*2 = 170000 **TotalLoss=**0.09

#### Exec#69 基于Exec#68 完成

rfcn\_resnet101\_ehualu4.config

step\_num=41300

for i in {0..4} # steps = 41300\*5 = 206500 **TotalLoss=**0.44

#### Exec#71 基于Exec#69 完成

rfcn\_resnet101\_ehualu4.config

step\_num=41300

for i in {0..5} # steps = 41300\*6 = 247800 **TotalLoss=**0.32

# 课程学习策略

小数据集 train\_b =1000——已完成

小数据集增广 train\_b\_aug=1000 \* (1+9) =10000 ——已完成

==== 4w 不行再 2w 不行再 1.1w

小数据集+大数据集+都增广 train\_b\_1w\_aug = 10000+10602 \* (1+2) =41806——可以

小数据集增广+大数据train\_b\_aug\_1w = 10000+10602=20602

小数据集+大数据train\_b\_1w = 1000+10602=11602

测试集+增广，1+4 = 5个版本

train\_csdn 数据集 <https://gitee.com/ai100/project-vehicle-detect> 不能用于物体检测

链接：https://pan.baidu.com/s/1SZTE9N1XyXb5LnSk-\_l58g 密码：cc6d

## 一级算法跑起来，使用train\_b

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>

* 1. YOLOv3：
  2. Faster R-CNN：faster\_rcnn\_resnet101\_coco=106/32
  3. R-FCN：[rfcn\_resnet101\_coco](http://download.tensorflow.org/models/object_detection/rfcn_resnet101_coco_2018_01_28.tar.gz)=92/30
  4. SSD：[ssd\_inception\_v2\_coco](http://download.tensorflow.org/models/object_detection/ssd_inception_v2_coco_2017_11_17.tar.gz)=42/24
  5. Faster R-CNN：faster\_rcnn\_inception\_resnet\_v2\_atrous\_coco=620/37
  6. Faster R-CNN：faster\_rcnn\_nas=1833/43
  7. light RCNN：？

### 实现内部融合权重，难点

### 导出一级模型特征图，重点

对测试集可行，合并特征图之后，用yolo3预测

### 实现外部融合特征图

这种方式行不通，融合特征图之后，如何进行次级模型训练？yolo3

## 使用train\_b+train\_1w

## 使用train\_b+train\_1w+train\_c

进行数据增广，训练集train\_c

## 使用train\_b+train\_1w+train\_c++train\_d

霍维佳的训练集

## 最后加权平均

权重取二级模型评分。

### 合并区域

关键是判断同一个区域，IOU

先计算各个区域中心之间的欧式距离，按距离排序。

再取两个区域较小的侧边距离。

如果中心距离<侧边距离，则是同一个区域。合并区域和概率。

### 先合并特征图，

再进行预测，取概率高的。