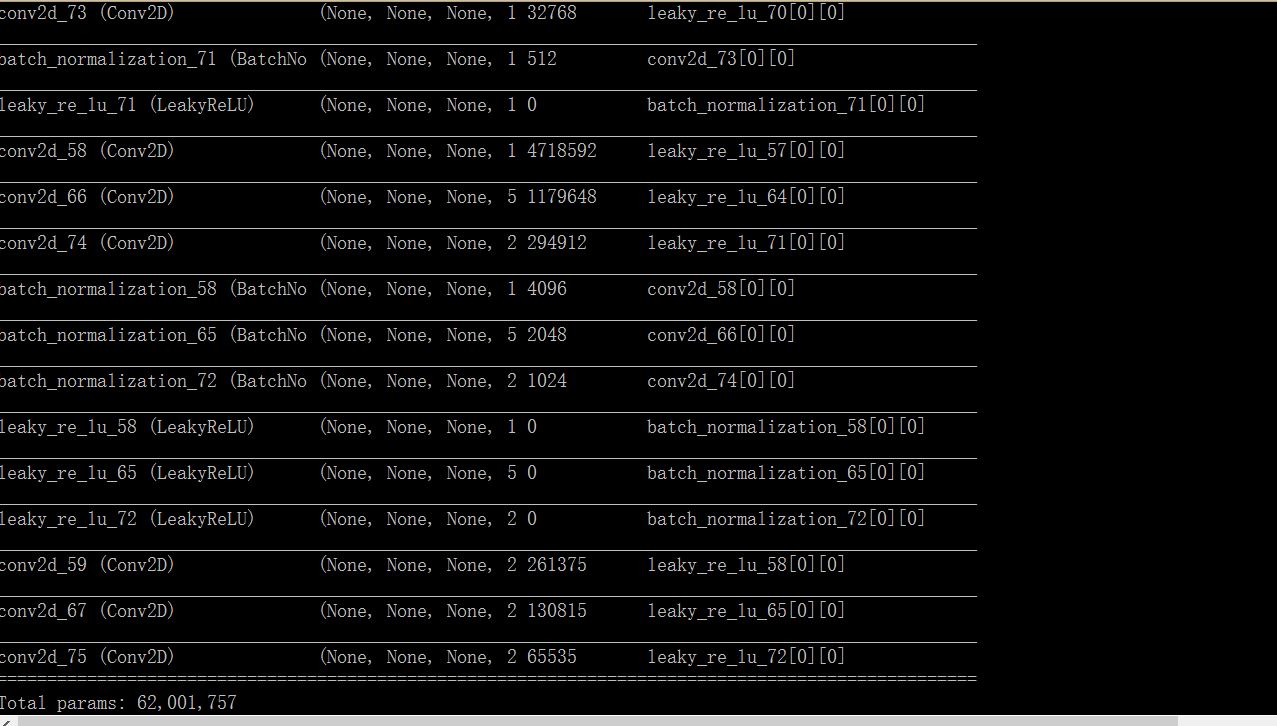
关于简单目标识别与意图分析的机器学习实战研究（神经网络目标识别——使用keras-yolo3训练自己的数据集）

我首先尝试一下keras-yolo3的可靠性，我首先下载了keras-yolo3的官方训练好的权重文件，附链接：https://pjreddie.com/media/files/yolov3.weights，而后将darknet的yolo转换为可以用于keras的h5文件，生成的h5被保存在model\_data下。命令及结果如下：

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



从训练文件的源代码可以看出，yolov3.weights中只训练了aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa,train, tvmonitor，下面我们拿一张汽车car图片作为例子试验一下，由于目标是图片，执行命令为pythno3 yolo\_video.py --image：



测试样本



测试结果

效果十分出众，那么我们自己训练的数据集会不会也可以达到这样的效果呢？

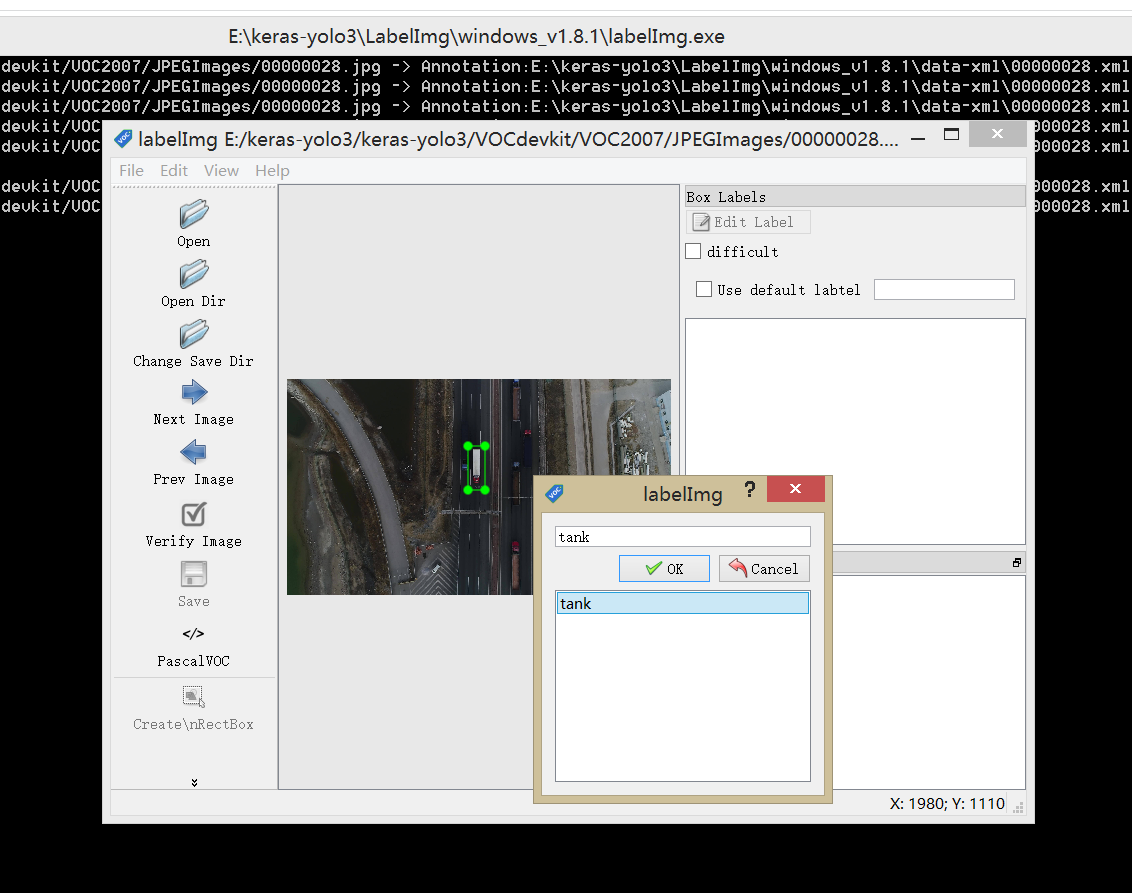
我们下面开始训练自己的数据集：

1.首先对自己的少量数据进性增强（数据多的就不用了）

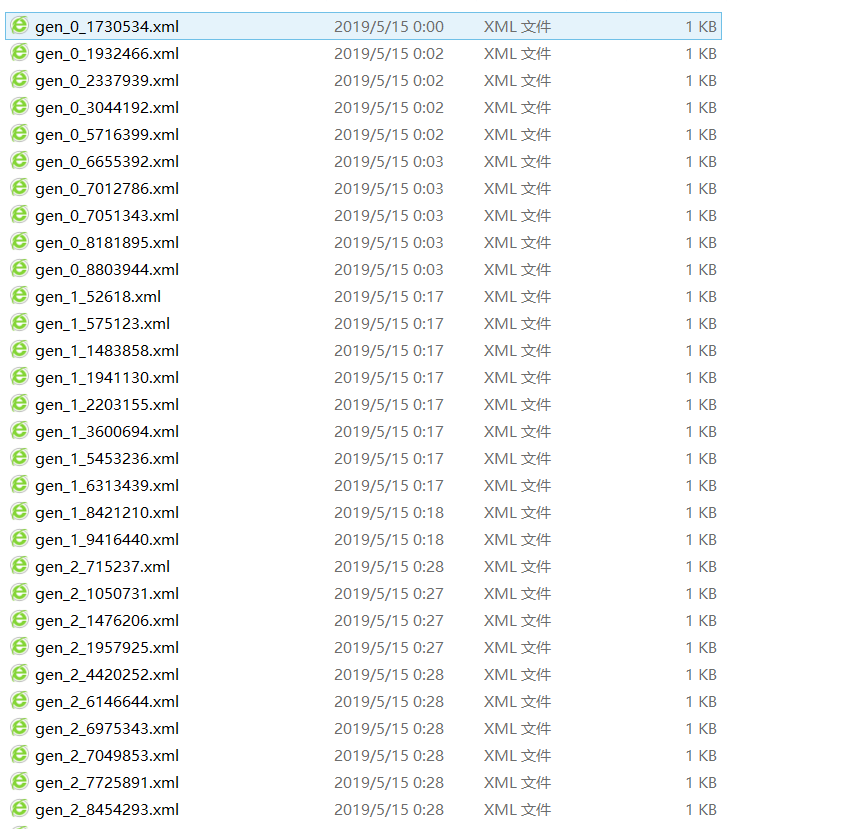
只要写代码对图像进行操作即可，参考文章：<https://www.cnblogs.com/siyuan1998/p/10686616.html>，我这里主要是进行了翻转、拉伸和裁剪，从100多张图片扩充到了900多张，拉伸的效果像这样：



2.然后对得到的数据集进行标注，这里用了GitHub上的一个开源工具LabelImg进行了快速标注，不够900多张图片自己一个人标注也是要一天的，再标注的过程中，主要是为了生成xml文件，对应每一张被标注的图片，内含信息是目标的位置坐标，以及目标的标签（比如：tank）：



LabelImg使用举例



得到了一堆xml文件



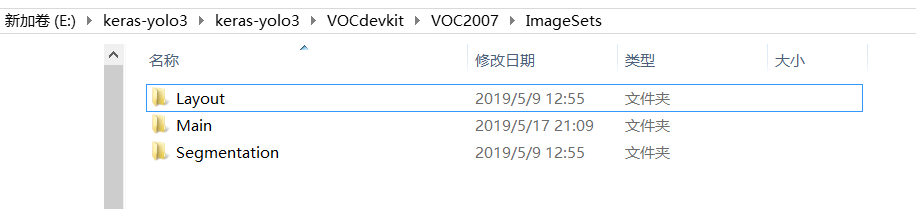
xml文件内部具体内容

3.生成对应配置文件：

首先在keras-yolo3文件夹中新建VOCdevkit文件夹，在VOCdevkit中再新建VOC2007文件夹，最后在VOC2007文件夹中再新建Annotations、ImageSets、JPEGImages、SegmentationClass、SegmentationObject文件夹，路径很重要，具体见图：



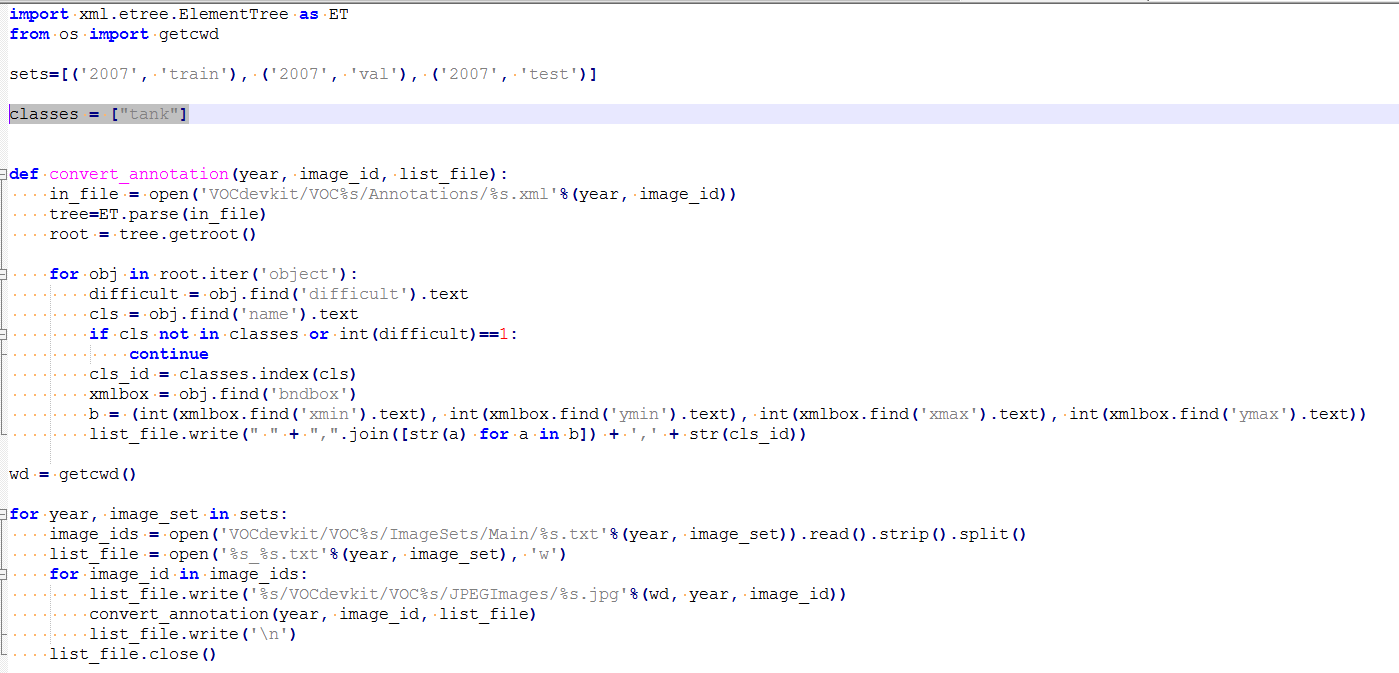
然后把训练用的数据集放入JPEGImages，把xml文件放入Annotations，然后在ImageSets中再新建三个文件夹：



然后执行下面代码使在Main文件中生成图像的配置文件：

1. **import** os
2. **import** random
4. trainval\_percent = 0.1
5. train\_percent = 0.9
6. xmlfilepath = 'Annotations'
7. txtsavepath = 'ImageSets\Main'
8. total\_xml = os.listdir(xmlfilepath)
10. num = len(total\_xml)
11. list = range(num)
12. tv = int(num \* trainval\_percent)
13. tr = int(tv \* train\_percent)
14. trainval = random.sample(list, tv)
15. train = random.sample(trainval, tr)
17. ftrainval = open('ImageSets/Main/trainval.txt', 'w')
18. ftest = open('ImageSets/Main/test.txt', 'w')
19. ftrain = open('ImageSets/Main/train.txt', 'w')
20. fval = open('ImageSets/Main/val.txt', 'w')
22. **for** i **in** list:
23. name = total\_xml[i][:-4] + '\n'
24. **if** i **in** trainval:
25. ftrainval.write(name)
26. **if** i **in** train:
27. ftest.write(name)
28. **else**:
29. fval.write(name)
30. **else**:
31. ftrain.write(name)
33. ftrainval.close()
34. ftrain.close()
35. fval.close()
36. ftest.close()

下一步，打开keras-yolo3文件夹中的voc\_annotation.py文件，classes类改写我们刚刚写的标签：



运行voc\_annotation.py，将生成三个txt文本文件，手动删除文件头前缀2007\_。

4.修改参数文件yolo3.cfg：

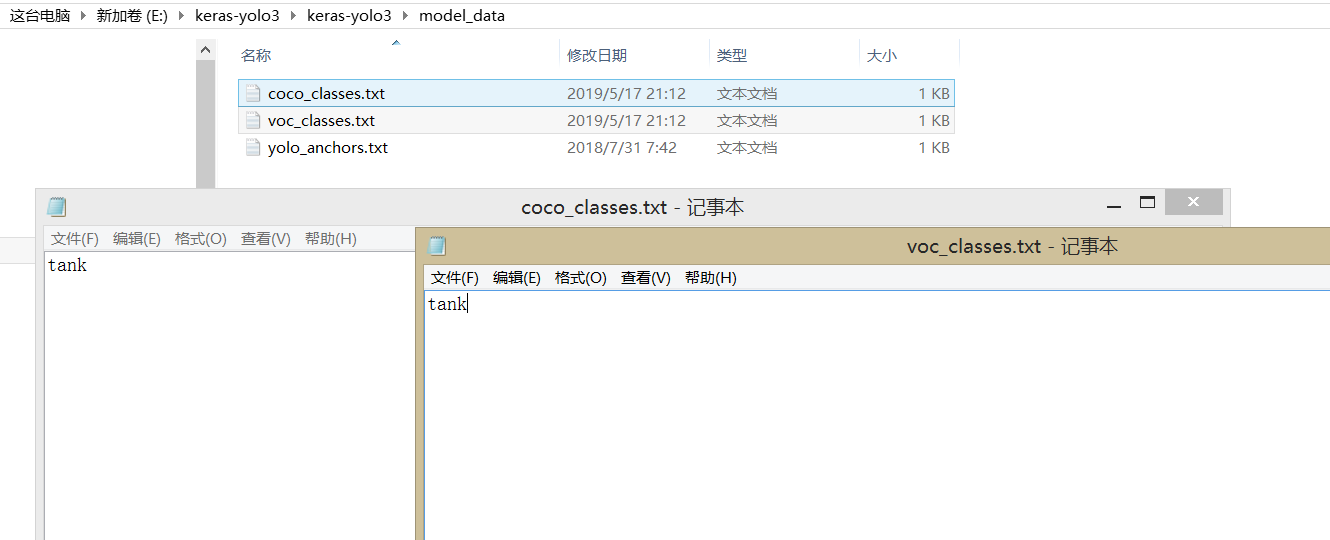
打开keras-yolo3文件夹中的yolov3.cfg文件，搜索[yolo]关键词，共有三处位置，分别修改其下3个参数：

filters：3\*（5+len（classes））

classes: len(classes) = 1，这里以只有一个tank标签为例

random：原来是1，显存小改为0

然后修改model\_data下coco,voc这两个文件，放入你的类别，这里存入tank。



5.修改训练代码train.py，开始训练：

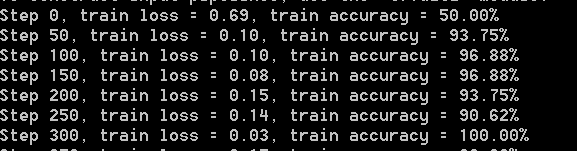
keras-yolo3自带的train.py会加载预先对coco数据集已经训练完成的yolo3权重文件，并冻结了开始到最后倒数第N层，以及在训练中只会保存最后一次训练完毕的数据，没办法在规定次数中保存训练数据，这里对代码进行了修改，更正了上述问题：

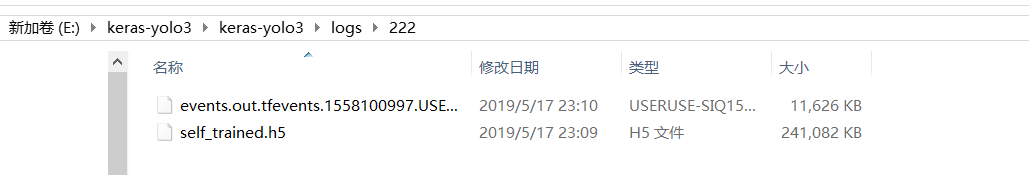
1. """
2. Retrain the YOLO model for your own dataset.
3. """
4. **import** numpy as np
5. **import** keras.backend as K
6. **from** keras.layers **import** Input, Lambda
7. **from** keras.models **import** Model
8. **from** keras.callbacks **import** TensorBoard, ModelCheckpoint, EarlyStopping
10. **from** yolo3.model **import** preprocess\_true\_boxes, yolo\_body, tiny\_yolo\_body, yolo\_loss
11. **from** yolo3.utils **import** get\_random\_data

14. **def** \_main():
15. annotation\_path = 'train.txt'
16. log\_dir = 'logs/222/'
17. classes\_path = 'model\_data/voc\_classes.txt'
18. anchors\_path = 'model\_data/yolo\_anchors.txt'
19. class\_names = get\_classes(classes\_path)
20. anchors = get\_anchors(anchors\_path)
21. input\_shape = (416,416) # multiple of 32, hw
22. model = create\_model(input\_shape, anchors, len(class\_names) )
23. train(model, annotation\_path, input\_shape, anchors, len(class\_names), log\_dir=log\_dir)
25. **def** train(model, annotation\_path, input\_shape, anchors, num\_classes, log\_dir='logs/'):
26. model.compile(optimizer='adam', loss={
27. 'yolo\_loss': **lambda** y\_true, y\_pred: y\_pred})
28. logging = TensorBoard(log\_dir=log\_dir)
29. #checkpoint = ModelCheckpoint(log\_dir + "ep{epoch:03d}-loss{loss:.3f}-val\_loss{val\_loss:.3f}.h5",
30. #    monitor='val\_loss', save\_weights\_only=True, save\_best\_only=True, period=1)
31. #checkpoint = ModelCheckpoint(log\_dir + "ep{epoch:03d}-loss{loss:.3f}-val\_loss{val\_loss:.3f}.h5",monitor='val\_acc', verbose=0, save\_best\_only=True, mode='max', period=1)
33. tensorboard = TensorBoard(log\_dir=log\_dir)
34. checkpoint = ModelCheckpoint(log\_dir + "self\_trained.h5",
35. monitor='val\_loss', save\_weights\_only=True, save\_best\_only=True,
36. period=1)

39. callbacks\_list = [tensorboard,checkpoint]
41. batch\_size = 10             #you can fix it
42. val\_split = 0.1
43. with open(annotation\_path) as f:
44. lines = f.readlines()
45. np.random.shuffle(lines)
46. num\_val = int(len(lines)\*val\_split)
47. num\_train = len(lines) - num\_val
48. **print**('Train on {} samples, val on {} samples, with batch size {}.'.format(num\_train, num\_val, batch\_size))
50. model.fit\_generator(data\_generator\_wrap(lines[:num\_train], batch\_size, input\_shape, anchors, num\_classes),
51. steps\_per\_epoch=max(1, num\_train//batch\_size),
52. validation\_data=data\_generator\_wrap(lines[num\_train:], batch\_size, input\_shape, anchors, num\_classes),
53. validation\_steps=max(1, num\_val//batch\_size),
54. epochs=500,
55. initial\_epoch=0,
56. callbacks = callbacks\_list,
57. verbose=2)
59. model.save\_weights(log\_dir + 'trained\_weights.h5')
61. **def** get\_classes(classes\_path):
62. with open(classes\_path) as f:
63. class\_names = f.readlines()
64. class\_names = [c.strip() **for** c **in** class\_names]
65. **return** class\_names
67. **def** get\_anchors(anchors\_path):
68. with open(anchors\_path) as f:
69. anchors = f.readline()
70. anchors = [float(x) **for** x **in** anchors.split(',')]
71. **return** np.array(anchors).reshape(-1, 2)
73. **def** create\_model(input\_shape, anchors, num\_classes, load\_pretrained=False, freeze\_body=False,
74. weights\_path='model\_data/yolo\_weights.h5'):
75. K.clear\_session() # get a new session
76. image\_input = Input(shape=(None, None, 3))
77. h, w = input\_shape
78. num\_anchors = len(anchors)
79. y\_true = [Input(shape=(h//{0:32, 1:16, 2:8}[l], w//{0:32, 1:16, 2:8}[l], \
80. num\_anchors//3, num\_classes+5)) **for** l **in** range(3)]
82. model\_body = yolo\_body(image\_input, num\_anchors//3, num\_classes)
83. **print**('Create YOLOv3 model with {} anchors and {} classes.'.format(num\_anchors, num\_classes))
85. **if** load\_pretrained:
86. model\_body.load\_weights(weights\_path, by\_name=True, skip\_mismatch=True)
87. **print**('Load weights {}.'.format(weights\_path))
88. **if** freeze\_body:
89. # Do not freeze 3 output layers.
90. num = len(model\_body.layers)-3
91. **for** i **in** range(num): model\_body.layers[i].trainable = False
92. **print**('Freeze the first {} layers of total {} layers.'.format(num, len(model\_body.layers)))
94. model\_loss = Lambda(yolo\_loss, output\_shape=(1,), name='yolo\_loss',
95. arguments={'anchors': anchors, 'num\_classes': num\_classes, 'ignore\_thresh': 0.5})(
96. [\*model\_body.output, \*y\_true])
97. model = Model([model\_body.input, \*y\_true], model\_loss)
98. **return** model
99. **def** data\_generator(annotation\_lines, batch\_size, input\_shape, anchors, num\_classes):
100. n = len(annotation\_lines)
101. np.random.shuffle(annotation\_lines)
102. i = 0
103. **while** True:
104. image\_data = []
105. box\_data = []
106. **for** b **in** range(batch\_size):
107. i %= n
108. image, box = get\_random\_data(annotation\_lines[i], input\_shape, random=True)
109. image\_data.append(image)
110. box\_data.append(box)
111. i += 1
112. image\_data = np.array(image\_data)
113. box\_data = np.array(box\_data)
114. y\_true = preprocess\_true\_boxes(box\_data, input\_shape, anchors, num\_classes)
115. **yield** [image\_data, \*y\_true], np.zeros(batch\_size)
117. **def** data\_generator\_wrap(annotation\_lines, batch\_size, input\_shape, anchors, num\_classes):
118. n = len(annotation\_lines)
119. **if** n==0 **or** batch\_size<=0: **return** None
120. **return** data\_generator(annotation\_lines, batch\_size, input\_shape, anchors, num\_classes)
122. **if** \_\_name\_\_ == '\_\_main\_\_':
123. \_main()

注意路径，文件保存在log\_dir，也可以自己修改，保存次数是period参数设置，period = 1代表每训练一次就会保存一次训练数据。大约训到losses值小于15即可，这里是训练好的数据（h5文件）：





6.数据测试：

源代码是将测试集代码随机抽取一张的，这里再改下detect\_img函数使其能够测试指定的图片：

1. **def** detect\_img(yolo):
2. **while** True:
3. img = input('Input image filename:')
4. **try**:
5. image = Image.open(img)
6. **except**:
7. **print**('Open Error! Try again!')
8. **continue**
9. **else**:
10. r\_image = yolo.detect\_image(image)
11. r\_image.show()
12. r\_image.save('E:/keras-yolo3/keras-yolo3/picback/' + img)
13. yolo.close\_session()

然后执行，测试过程和前面所讲一样，因为我就训练了不到200次（这根本不是我的重点），就不贴出很挫的测试效果图了。不过很多大神跑了很久，还是测试成功的，主要注意标签标记时准一些就好了。