案例的由来

此案例是根据官方给的案例

Faster R-CNN [TensorFlow版的源码](https://github.com/dBeker/Faster-RCNN-TensorFlow-Python3)

https://github.com/dBeker/Faster-RCNN-TensorFlow-Python3

[Faster R-CNN网络解读](https://blog.csdn.net/BreakingDawn0/article/details/103073616)

<https://blog.csdn.net/BreakingDawn0/article/details/103073616>

Faster-R-CNN Tensorflow 源码理解

<https://blog.csdn.net/BreakingDawn0/article/details/103537773>

详细的Faster-R-CNN源码解析之proposal\_layer和proposal\_target\_layer源码解析

<https://blog.csdn.net/jiongnima/article/details/80478597>

复现Faster-RCNN-TensorFlow-Python3.5代码，并用自己的数据集来训练

<https://blog.csdn.net/kellyroslyn/article/details/92730328>

Ubuntu 16.04批量缩小图像、批量命名图像

https://blog.csdn.net/kellyroslyn/article/details/93364072

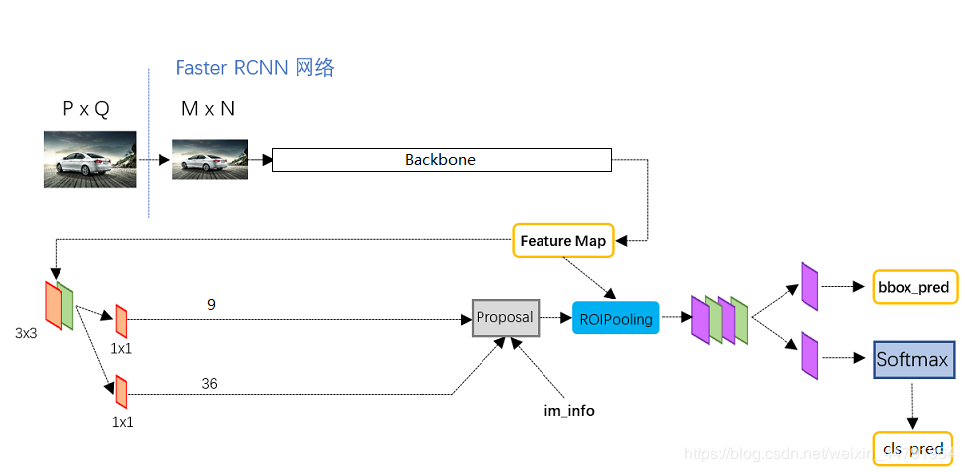
复现Faster-RCNN-TensorFlow-Python3.5代码，并用自己的数据集来训练（二）

https://blog.csdn.net/kellyroslyn/article/details/93294790

python3+Tensorflow+Faster R-CNN训练自己的数据

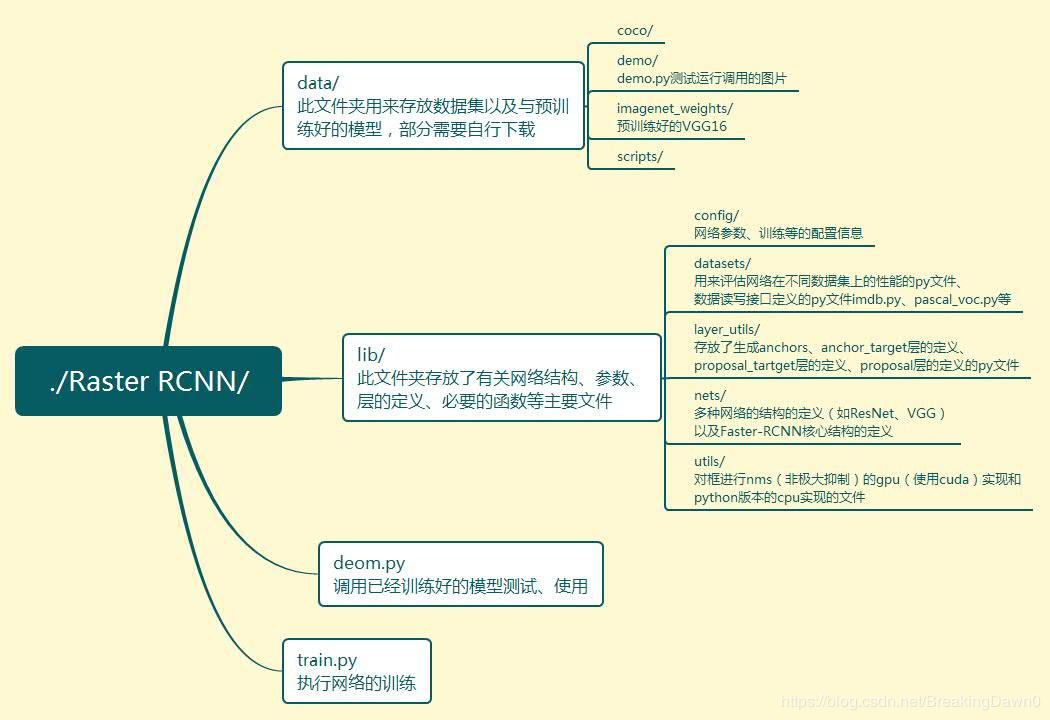
https://blog.csdn.net/char\_QwQ/article/details/80980505

# Fast-R-CNN算法框架流程图



Proposal:

# 工程目录结构



## lib/datasets

数据生成（imdb, roidb）

datasets/imdb.py: 定义通用的图像数据库类imdb

datasets/factory.py: 利用lambda表达式像工厂一样自定义自己所需的数据库类，以下都以voc\_2007\_trainval数据集为例，在继承imdb类的基础上，定义pascal\_voc类。

|  |
| --- |
| # 以voc数据集为例，按照imdb的命名，利用pascal\_voc()函数生成不同的imdb  for year in ['2007', '2012']:  for split in ['train', 'val', 'trainval', 'test']:  name = 'voc\_{}\_{}'.format(year, split) #year='2007', split='trainval'  \_\_sets[name] = (lambda split=split, year=year: pascal\_voc(split, year))  def get\_imdb(name):  """Get an imdb (image database) by name."""  if name not in \_\_sets:  raise KeyError('Unknown dataset: {}'.format(name))  return \_\_sets[name]() |

****datasets/pascal\_voc.py****: 定义pascal\_voc类（继承自imdb）。在这一部分，根据自己数据库的具体情况来定义成员变量和成员函数。下面列出一些重要的成员变量及成员函数：

部分成员变量

|  |
| --- |
| self.\_data\_path = os.path.join(self.\_devkit\_path, 'VOC' + self.\_year) #数据库路径  self.\_classes = ('\_\_background\_\_', # always index 0， 训练类别标签，包含背景类  'person')  # Default to roidb handler  self.\_roidb\_handler = self.gt\_roidb #感兴趣区域（ROI）数据库  self.\_salt = str(uuid.uuid4()) #  self.\_comp\_id = 'comp4' # |

部分成员函数

gt\_roidb(): 调用\_load\_pascal\_annotation()函数，返回ROI数据库。保存缓冲文件（第一次运行时），或载入数据库缓冲文件。

\_load\_pascal\_annotation(): 从VOC数据库的XML文件中载入图像和bbox等信息, 包括：bboxes坐标，类别，overlap矩阵，bbox面积等。

****model.train\_val.get\_training\_roidb(imdb)****: 返回roidb (RoI数据库) 用来训练模型。   
主要调用两个函数：

|  |
| --- |
| imdb.append\_flipped\_images() # imdb类的一个成员函数，用来水平翻转训练集（数据增强）  rdl\_roidb.prepare\_roidb(imdb) # roidb.py中定义的函数 |

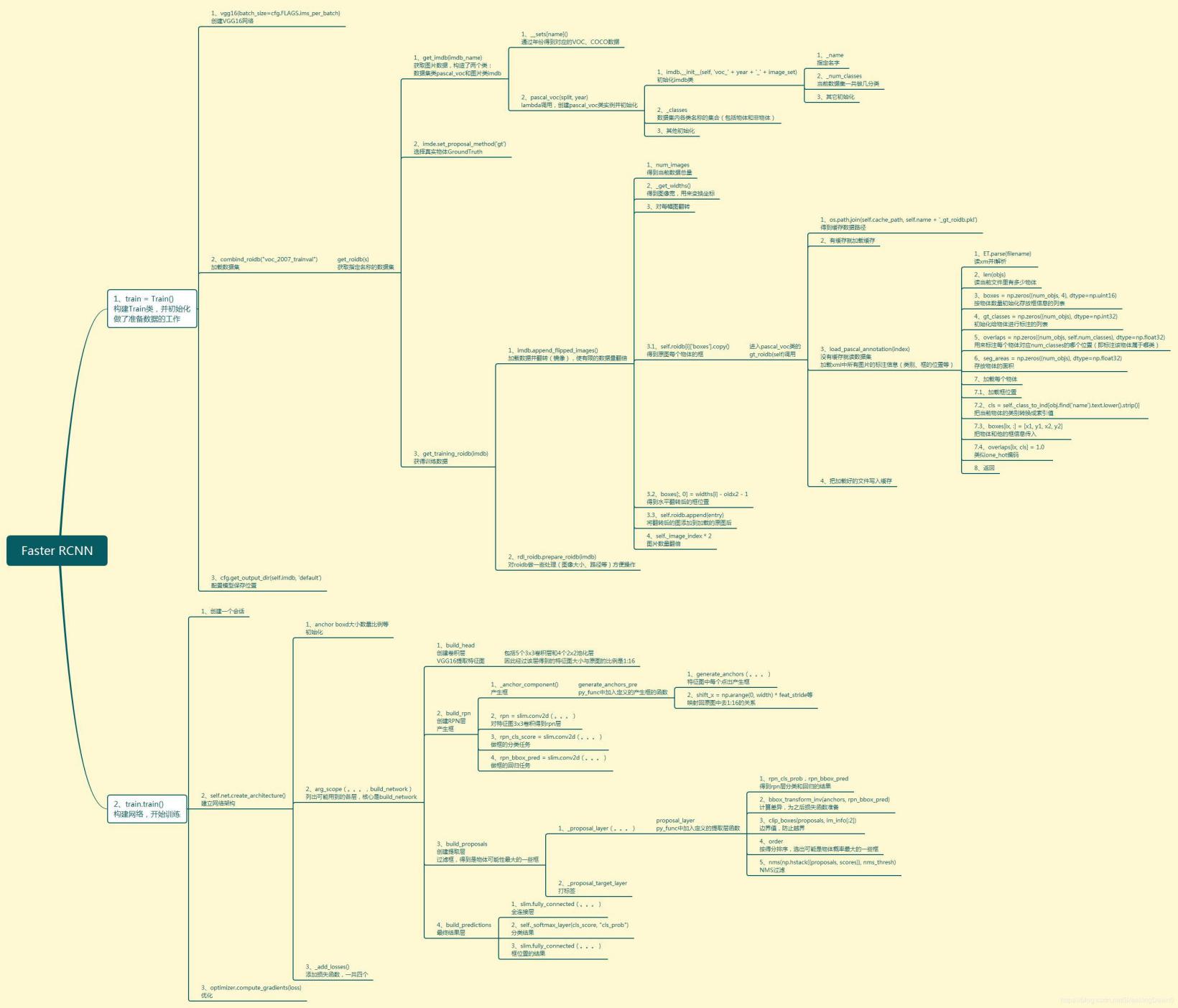
## Lib/layer\_utils

以下案例并非本文案例, 只是为了方便

|  |
| --- |
| # -\*- coding: utf-8 -\*-    import numpy as np  import tensorflow as tf    from Lib.bbox\_transform import bbox\_transform\_inv, clip\_boxes #bbox\_transform\_inv改变初始框的坐标，clip\_boxes把超出图像边界的框限制在图像边界内  from Lib.faster\_rcnn\_config import cfg #配置文件  from Lib.generate\_anchors import generate\_anchors #生成初始框  from Lib.nms\_wrapper import nms #去掉多余的重叠的框      #使用tf.py\_func接口，方便进行numpy运算  def proposal\_layer(rpn\_bbox\_cls\_prob, rpn\_bbox\_pred, im\_dims, cfg\_key, \_feat\_stride, anchor\_scales):  return tf.reshape(tf.py\_func(\_proposal\_layer\_py,[rpn\_bbox\_cls\_prob, rpn\_bbox\_pred, im\_dims[0], cfg\_key, \_feat\_stride, anchor\_scales], [tf.float32]),[-1,5])      def \_proposal\_layer\_py(rpn\_bbox\_cls\_prob, rpn\_bbox\_pred, im\_dims, cfg\_key, \_feat\_stride, anchor\_scales):  '''  # Algorithm:  #  # for each (H, W) location i  # generate A anchor boxes centered on cell i  # apply predicted bbox deltas at cell i to each of the A anchors  # clip predicted boxes to image  # remove predicted boxes with either height or width < threshold  # sort all (proposal, score) pairs by score from highest to lowest  # take top pre\_nms\_topN proposals before NMS  # apply NMS with threshold 0.7 to remaining proposals  # take after\_nms\_topN proposals after NMS  # return the top proposals (-> RoIs top, scores top)    '''  \_anchors = generate\_anchors(scales=np.array(anchor\_scales)) #生成9个锚点，shape: [9,4]  \_num\_anchors = \_anchors.shape[0] #\_num\_anchors值为9  rpn\_bbox\_cls\_prob = np.transpose(rpn\_bbox\_cls\_prob,[0,3,1,2]) #将RPN输出的分类信息维度变成[N,C,H,W]  rpn\_bbox\_pred = np.transpose(rpn\_bbox\_pred,[0,3,1,2]) #将RPN输出的边框变换信息维度变成[N,C,H,W]    # Only minibatch of 1 supported 核验一下batch\_size必须等于1  assert rpn\_bbox\_cls\_prob.shape[0] == 1, \  'Only single item batches are supported'    if cfg\_key == 'TRAIN': #如果是在训练的话  pre\_nms\_topN = cfg.TRAIN.RPN\_PRE\_NMS\_TOP\_N #12000  post\_nms\_topN = cfg.TRAIN.RPN\_POST\_NMS\_TOP\_N #2000  nms\_thresh = cfg.TRAIN.RPN\_NMS\_THRESH #0.7  min\_size = cfg.TRAIN.RPN\_MIN\_SIZE #16  else: # cfg\_key == 'TEST': 如果是在测试的话  pre\_nms\_topN = cfg.TEST.RPN\_PRE\_NMS\_TOP\_N #6000  post\_nms\_topN = cfg.TEST.RPN\_POST\_NMS\_TOP\_N #300  nms\_thresh = cfg.TEST.RPN\_NMS\_THRESH #0.7  min\_size = cfg.TEST.RPN\_MIN\_SIZE #16      # the first set of \_num\_anchors channels are bg probs  # the second set are the fg probs, which we want  #按照通道C取出RPN预测的框属于前景的分数，请注意，在18个channel中，前9个是框属于背景的概率，后9个才是属于前景的概率  scores = rpn\_bbox\_cls\_prob[:, \_num\_anchors:, :, :]  #bbox\_deltas代表了RPN输出的各个框的坐标变换信息  bbox\_deltas = rpn\_bbox\_pred    # 1. Generate proposals from bbox deltas and shifted anchors  height, width = scores.shape[-2:] #在这里得到了rpn输出的H和W，    # Enumerate all shifts  shift\_x = np.arange(0, width) \* \_feat\_stride #shape: [width,]  shift\_y = np.arange(0, height) \* \_feat\_stride #shape: [height,]  shift\_x, shift\_y = np.meshgrid(shift\_x, shift\_y) #生成网格 shift\_x shape: [height, width], shift\_y shape: [height, width]  shifts = np.vstack((shift\_x.ravel(), shift\_y.ravel(),  shift\_x.ravel(), shift\_y.ravel())).transpose() # shape[height\*width, 4]    # Enumerate all shifted anchors:  #  # add A anchors (1, A, 4) to  # cell K shifts (K, 1, 4) to get  # shift anchors (K, A, 4)  # reshape to (K\*A, 4) shifted anchors  A = \_num\_anchors # A = 9  K = shifts.shape[0] # K=height\*width(特征图上的)  anchors = \_anchors.reshape((1, A, 4)) + \  shifts.reshape((1, K, 4)).transpose((1, 0, 2)) #shape[K,A,4] 得到所有的初始框  anchors = anchors.reshape((K \* A, 4)) #把初始框的数组维度改变一下，变成[K×A,4]    # Transpose and reshape predicted bbox transformations to get them  # into the same order as the anchors:  #  # bbox deltas will be (1, 4 \* A, H, W) format  # transpose to (1, H, W, 4 \* A)  # reshape to (1 \* H \* W \* A, 4) where rows are ordered by (h, w, a)  # in slowest to fastest order  #将RPN输出的边框变换信息维度变回[N,H,W,C]，再改变一下维度，变成[1×H×W×A,4]  bbox\_deltas = bbox\_deltas.transpose((0, 2, 3, 1)).reshape((-1, 4))    # Same story for the scores:  #  # scores are (1, A, H, W) format  # transpose to (1, H, W, A)  # reshape to (1 \* H \* W \* A, 1) where rows are ordered by (h, w, a)  #将RPN输出的分类信息维度变回[N,H,W,C]，再改变一下维度，变成[1×H×W×A,1]  scores = scores.transpose((0, 2, 3, 1)).reshape((-1, 1))    # Convert anchors into proposals via bbox transformations  #在这里结合RPN的输出变换初始框的坐标，得到第一次变换坐标后的proposals  proposals = bbox\_transform\_inv(anchors, bbox\_deltas)    # 2. clip predicted boxes to image  #在这里讲超出图像边界的proposal进行边界裁剪，使之在图像边界之内  proposals = clip\_boxes(proposals, im\_dims)    # 3. remove predicted boxes with either height or width < threshold  #排除掉长或者宽太小的框，keep下标指的是需要保留的长宽合适的框的索引  keep = \_filter\_boxes(proposals, min\_size)  proposals = proposals[keep, :]  scores = scores[keep]    # 4. sort all (proposal, score) pairs by score from highest to lowest  # 5. take top pre\_nms\_topN (e.g. 6000)  #对框按照前景分数进行排序，order中指示了框的下标  order = scores.ravel().argsort()[::-1]  if pre\_nms\_topN > 0:  order = order[:pre\_nms\_topN] #选择前景分数排名在前pre\_nms\_topN(训练时为12000，测试时为6000)的框  proposals = proposals[order, :] #保留了前pre\_nms\_topN个框的坐标信息  scores = scores[order] #保留了前pre\_nms\_topN个框的分数信息    # 6. apply nms (e.g. threshold = 0.7)  # 7. take after\_nms\_topN (e.g. 300)  # 8. return the top proposals (-> RoIs top)  #使用nms算法排除重复的框  keep = nms(np.hstack((proposals, scores)), nms\_thresh)  if post\_nms\_topN > 0:  keep = keep[:post\_nms\_topN] #选择前景分数排名在前post\_nms\_topN(训练时为2000，测试时为300)的框  proposals = proposals[keep, :] #保留了前post\_nms\_topN个框的坐标信息  scores = scores[keep] #保留了前post\_nms\_topN个框的分数信息    # Output rois blob  # Our RPN implementation only supports a single input image, so all  # batch inds are 0  #因为要进行roi\_pooling，在保留框的坐标信息前面插入batch中图片的编号信息。此时，由于batch\_size为1，因此都插入0  batch\_inds = np.zeros((proposals.shape[0], 1), dtype=np.float32)  blob = np.hstack((batch\_inds, proposals.astype(np.float32, copy=False)))  return blob      def \_filter\_boxes(boxes, min\_size):  """Remove all boxes with any side smaller than min\_size.  \_filter\_boxes函数过滤掉proposals中边框长宽太小的框  """  ws = boxes[:, 2] - boxes[:, 0] + 1 #得到所有框的宽  hs = boxes[:, 3] - boxes[:, 1] + 1 #得到所有框的长  keep = np.where((ws >= min\_size) & (hs >= min\_size))[0] #返回满足长宽均在阈值之上的框的下标  return keep |

# **train.py网络训练**

训练的主要执行过程和函数调用关系如下图所示，对应源码的部分下文有较详细的注释.



# 准备工作

## Pascal VOC2007数据集

从以下网址下载数据集

wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval\_06-Nov-2007.tar

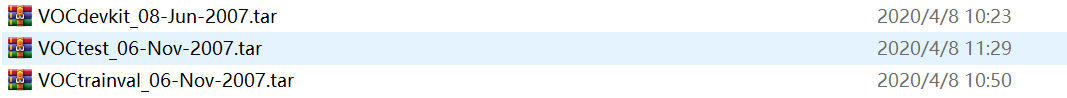
wget

http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest\_06-Nov-2007.tar

wget

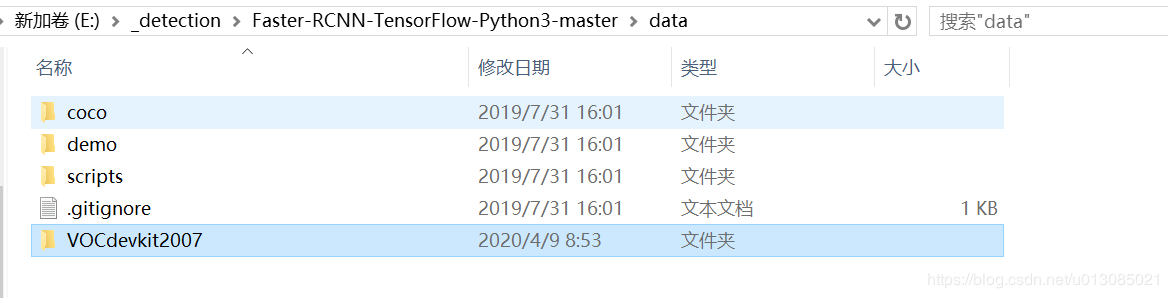
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCdevkit\_08-Jun-2007.tar

将下载好的VOC2007数据集放到一个目录下.



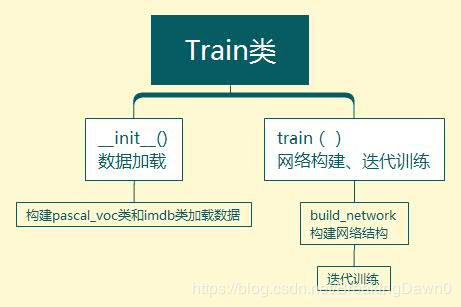
全选同时解压到”当前目录下”

解压后会在当前目录下生成VOCdevit文件夹，将****文件夹名修改为VOCdevkit2007****（因为在代码中使用该目录），然后移动该文件夹到data文件夹下, 如下图所示.



# **Train类**

网络训练的核心是Train类，Train类的\_\_init\_\_函数进行了数据加载的全部操作，Train类的函数tain()实现了网络构建和迭代训练的操作。



|  |
| --- |
| class Train:  def \_\_init\_\_(self):    # Create network  if cfg.FLAGS.network == 'vgg16':  self.net = vgg16(batch\_size=cfg.FLAGS.ims\_per\_batch)  else:  raise NotImplementedError    # 加载数据集  self.imdb, self.roidb = combined\_roidb("voc\_2007\_trainval")    self.data\_layer = RoIDataLayer(self.roidb, self.imdb.num\_classes)  self.output\_dir = cfg.get\_output\_dir(self.imdb, 'default') # 模型保存位置      def train(self):    # Create session  tfconfig = tf.ConfigProto(allow\_soft\_placement=True)  tfconfig.gpu\_options.allow\_growth = True  sess = tf.Session(config=tfconfig)    with sess.graph.as\_default():    tf.set\_random\_seed(cfg.FLAGS.rng\_seed)  # 建立一个网络架构  layers = self.net.create\_architecture(sess, "TRAIN", self.imdb.num\_classes, tag='default')  loss = layers['total\_loss']  lr = tf.Variable(cfg.FLAGS.learning\_rate, trainable=False)  momentum = cfg.FLAGS.momentum  optimizer = tf.train.MomentumOptimizer(lr, momentum)    gvs = optimizer.compute\_gradients(loss)  # ...... |

4 \_\_init\_\_（self）Train类的初始化函数

4.1 combined\_roidb(imdb\_names)由数据集名称准备加载对应数据

考虑到可能不只有一个数据集，roidbs = [get\_roidb(s) for s in imdb\_names.split('+')]将用+连接的数据集分别加载进来。

imdb = get\_imdb(imdb\_name)调用中构造了两个类：数据集类pascal\_voc和图片类imdb，见下文。

|  |
| --- |
| def combined\_roidb(imdb\_names):  """  Combine multiple roidbs  """    def get\_roidb(imdb\_name):  imdb = get\_imdb(imdb\_name) # 构造了两个类：数据集类pascal\_voc和图片类imdb  print('Loaded dataset `{:s}` for training'.format(imdb.name))  imdb.set\_proposal\_method("gt") # 即选择真实物体groundtruth  print('Set proposal method: {:s}'.format("gt"))  roidb = get\_training\_roidb(imdb) # 获得训练数据  return roidb    # 考虑到可能不只有一个数据集，将用+连接的数据集分别加载进来  roidbs = [get\_roidb(s) for s in imdb\_names.split('+')]  roidb = roidbs[0]  if len(roidbs) > 1:  for r in roidbs[1:]:  roidb.extend(r)  tmp = get\_imdb(imdb\_names.split('+')[1])  imdb = imdb2(imdb\_names, tmp.classes)  else:  imdb = get\_imdb(imdb\_names)  return imdb, roidb |

## **4.2 factory.py工厂类**

利用lambda表达式像工厂一样自定义自己所需的数据库类，返回数据库供网络训练和测试使用

|  |
| --- |
| # Set up voc\_<year>\_<split>  for year in ['2007', '2012']:  for split in ['train', 'val', 'trainval', 'test']:  name = 'voc\_{}\_{}'.format(year, split)  \_\_sets[name] = (lambda split=split, year=year: pascal\_voc(split, year)) # pascal\_voc类 |

## 4.3 class pascal\_voc(imdb)

pascal\_voc是继承自imdb的类，主要针对数据集中生成roidb

class pascal\_voc(imdb):

def \_\_init\_\_(self, image\_set, year, devkit\_path=None):

imdb.\_\_init\_\_(self, 'voc\_' + year + '\_' + image\_set) # 构造imdb

self.\_year = year

self.\_image\_set = image\_set

self.\_devkit\_path = self.\_get\_default\_path() if devkit\_path is None \

else devkit\_path

self.\_data\_path = os.path.join(self.\_devkit\_path, 'VOC' + self.\_year)

self.\_classes = ('\_\_background\_\_', # always index 0

'aeroplane', 'bicycle', 'bird', 'boat',

'bottle', 'bus', 'car', 'cat', 'chair',

'cow', 'diningtable', 'dog', 'horse',

'motorbike', 'person', 'pottedplant',

'sheep', 'sofa', 'train', 'tvmonitor')

self.\_class\_to\_ind = dict(list(zip(self.classes, list(range(self.num\_classes)))))

self.\_image\_ext = '.jpg'

self.\_image\_index = self.\_load\_image\_set\_index()

# Default to roidb handler

self.\_roidb\_handler = self.gt\_roidb

self.\_salt = str(uuid.uuid4())

self.\_comp\_id = 'comp4'

# ......

## 4.4 class imdb(object)数据库读写类

是数据库读写类的基类，封装了许多db的操作

def \_\_init\_\_(self, name, classes=None):

self.\_name = name # 指定名字

self.\_num\_classes = 0 # 当前数据集一共做多少分类

if not classes:

self.\_classes = []

else:

self.\_classes = classes

self.\_image\_index = []

self.\_obj\_proposer = 'gt'

self.\_roidb = None # 得到的结果

self.\_roidb\_handler = self.default\_roidb

# Use this dict for storing dataset specific config options

self.config = {}

# ......

## 4.5 get\_training\_roidb(imdb)获得训练数据

再回到get\_roidb(imdb\_name)函数中，get\_training\_roidb(imdb)是在获得训练数据，包括

加载数据并翻转（镜像），数据量翻倍

对roidb做一些处理（图像大小、路径），方便操作

def get\_training\_roidb(imdb):

"""Returns a roidb (Region of Interest database) for use in training."""

if True:

print('Appending horizontally-flipped training examples...')

imdb.append\_flipped\_images() # 加载数据并翻转（镜像），数据量翻倍

print('done')

print('Preparing training data...')

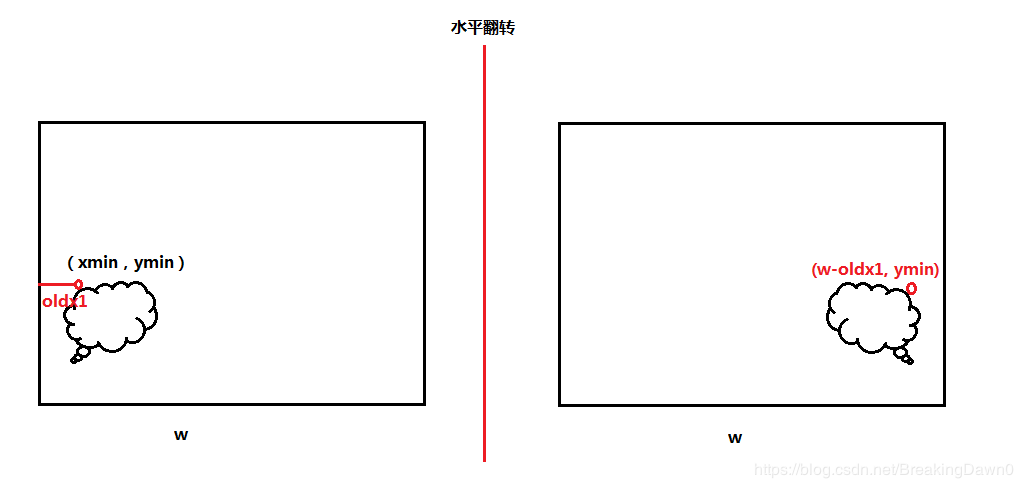
rdl\_roidb.prepare\_roidb(imdb)

print('done')

return imdb.roidb

## 4.6 imdb.append\_flipped\_images()翻转原始图像

翻转即：



主要进行了：

得到当前的数据总量

得到图像宽用于坐标变换

所有的图都做一次镜像

得到原图每个物体的框，boxes = self.roidb[i]['boxes'].copy()详细见下文

将翻转后的图添加到加载的原图后

### 数据镜像的源代码

|  |
| --- |
| def append\_flipped\_images(self):  num\_images = self.num\_images # 得到当前的数据总量  widths = self.\_get\_widths() # 得到图像宽，用来变换坐标:图像镜像以后，框的坐标也一起改变  for i in range(num\_images): # 所有的图都做一次镜像  boxes = self.roidb[i]['boxes'].copy() # 得到原图每个物体的框  oldx1 = boxes[:, 0].copy()  oldx2 = boxes[:, 2].copy()  boxes[:, 0] = widths[i] - oldx2 - 1 # 得到水平翻转后的框位置  boxes[:, 2] = widths[i] - oldx1 - 1  assert (boxes[:, 2] >= boxes[:, 0]).all()  entry = {'boxes': boxes,  'gt\_overlaps': self.roidb[i]['gt\_overlaps'],  'gt\_classes': self.roidb[i]['gt\_classes'],  'flipped': True}  self.roidb.append(entry) # 将翻转后的图添加到加载的原图后  self.\_image\_index = self.\_image\_index \* 2 # 图片数量翻倍 |

## 4.7 self.roidb[i]['boxes'].copy()得到原图每个物体的框gt

注释中描述了这个过程

def \_load\_pascal\_annotation(self, index):

"""

Load image and bounding boxes info from XML file in the PASCAL VOC

format.

"""

filename = os.path.join(self.\_data\_path, 'Annotations', index + '.xml')

tree = ET.parse(filename) # xml解析

objs = tree.findall('object')

if not self.config['use\_diff']:

# Exclude the samples labeled as difficult

non\_diff\_objs = [

obj for obj in objs if int(obj.find('difficult').text) == 0]

# if len(non\_diff\_objs) != len(objs):

# print 'Removed {} difficult objects'.format(

# len(objs) - len(non\_diff\_objs))

objs = non\_diff\_objs

num\_objs = len(objs) # 读当前文件里有多少物体

boxes = np.zeros((num\_objs, 4), dtype=np.uint16) # 按物体数量初始化存放框信息的列表

gt\_classes = np.zeros((num\_objs), dtype=np.int32) # 初始化给物体进行标注的列表

overlaps = np.zeros((num\_objs, self.num\_classes), dtype=np.float32) # 用来标注每个物体对应num\_classes的哪个位置（即标注该物体属于哪类）

# "Seg" area for pascal is just the box area

seg\_areas = np.zeros((num\_objs), dtype=np.float32) # 存放物体的面积

# Load object bounding boxes into a data frame.

for ix, obj in enumerate(objs):

bbox = obj.find('bndbox')

# Make pixel indexes 0-based

x1 = float(bbox.find('xmin').text) - 1

y1 = float(bbox.find('ymin').text) - 1

x2 = float(bbox.find('xmax').text) - 1

y2 = float(bbox.find('ymax').text) - 1

cls = self.\_class\_to\_ind[obj.find('name').text.lower().strip()] # 把当前物体的类别转换成索引值

boxes[ix, :] = [x1, y1, x2, y2] # 把物体和他的框信息传入

gt\_classes[ix] = cls

overlaps[ix, cls] = 1.0 # 类似one\_hot编码

seg\_areas[ix] = (x2 - x1 + 1) \* (y2 - y1 + 1)

overlaps = scipy.sparse.csr\_matrix(overlaps)

return {'boxes': boxes,

'gt\_classes': gt\_classes,

'gt\_overlaps': overlaps,

'flipped': False,

'seg\_areas': seg\_areas}

5.train（self）Train类的函数

建立网络，迭代训练

def train(self):

# Create session

tfconfig = tf.ConfigProto(allow\_soft\_placement=True)

tfconfig.gpu\_options.allow\_growth = True

sess = tf.Session(config=tfconfig)

with sess.graph.as\_default():

tf.set\_random\_seed(cfg.FLAGS.rng\_seed)

# 建立一个网络架构

layers = self.net.create\_architecture(sess, "TRAIN", self.imdb.num\_classes, tag='default')

loss = layers['total\_loss']

lr = tf.Variable(cfg.FLAGS.learning\_rate, trainable=False)

momentum = cfg.FLAGS.momentum

optimizer = tf.train.MomentumOptimizer(lr, momentum)

gvs = optimizer.compute\_gradients(loss)

# ......

# network.py 网络模块类

## 5.1 create\_architecture()构建网络

先进行一些初始化，如anchor box大小、比例的初值的个数等

self.build\_network(sess, training)是核心，构建网络

def create\_architecture(self, sess, mode, num\_classes, tag=None, anchor\_scales=(8, 16, 32), anchor\_ratios=(0.5, 1, 2)):

self.\_image = tf.placeholder(tf.float32, shape=[self.\_batch\_size, None, None, 3])

self.\_im\_info = tf.placeholder(tf.float32, shape=[self.\_batch\_size, 3])

self.\_gt\_boxes = tf.placeholder(tf.float32, shape=[None, 5])

self.\_tag = tag

self.\_num\_classes = num\_classes

self.\_mode = mode

self.\_anchor\_scales = anchor\_scales # anchor box的初始大小

self.\_num\_scales = len(anchor\_scales) # 几钟大小

self.\_anchor\_ratios = anchor\_ratios # anchor box的长宽比

self.\_num\_ratios = len(anchor\_ratios) # 几钟比例

self.\_num\_anchors = self.\_num\_scales \* self.\_num\_ratios # anchor box一共几钟类型

training = mode == 'TRAIN'

testing = mode == 'TEST'

assert tag != None

# handle most of the regularizer here

weights\_regularizer = tf.contrib.layers.l2\_regularizer(cfg.FLAGS.weight\_decay)

if cfg.FLAGS.bias\_decay:

biases\_regularizer = weights\_regularizer

else:

biases\_regularizer = tf.no\_regularizer

# list as many types of layers as possible, even if they are not used now

with arg\_scope([slim.conv2d, slim.conv2d\_in\_plane,

slim.conv2d\_transpose, slim.separable\_conv2d, slim.fully\_connected],

weights\_regularizer=weights\_regularizer,

biases\_regularizer=biases\_regularizer,

biases\_initializer=tf.constant\_initializer(0.0)):

rois, cls\_prob, bbox\_pred = self.build\_network(sess, training) # 核心build\_network

# ......

## 5.2 build\_network(self, sess, is\_training=True)构建网络

build\_network被网络vgg16和resnet网络模块所继承,并且重写

build head创建卷积层

build rpn创建rpn层

build proposals创建过滤层

build predictions创建全连接输出层

def build\_network(self, sess, is\_training=True):

with tf.variable\_scope('vgg\_16', 'vgg\_16'):

# select initializer

if cfg.FLAGS.initializer == "truncated":

initializer = tf.truncated\_normal\_initializer(mean=0.0, stddev=0.01)

initializer\_bbox = tf.truncated\_normal\_initializer(mean=0.0, stddev=0.001)

else:

initializer = tf.random\_normal\_initializer(mean=0.0, stddev=0.01)

initializer\_bbox = tf.random\_normal\_initializer(mean=0.0, stddev=0.001)

# Build head

net = self.build\_head(is\_training)

# Build rpn

rpn\_cls\_prob, rpn\_bbox\_pred, rpn\_cls\_score, rpn\_cls\_score\_reshape = self.build\_rpn(net, is\_training, initializer)

# Build proposals

rois = self.build\_proposals(is\_training, rpn\_cls\_prob, rpn\_bbox\_pred, rpn\_cls\_score)

# Build predictions

cls\_score, cls\_prob, bbox\_pred = self.build\_predictions(net, rois, is\_training, initializer, initializer\_bbox)

self.\_predictions["rpn\_cls\_score"] = rpn\_cls\_score

self.\_predictions["rpn\_cls\_score\_reshape"] = rpn\_cls\_score\_reshape

self.\_predictions["rpn\_cls\_prob"] = rpn\_cls\_prob

self.\_predictions["rpn\_bbox\_pred"] = rpn\_bbox\_pred

self.\_predictions["cls\_score"] = cls\_score

self.\_predictions["cls\_prob"] = cls\_prob

self.\_predictions["bbox\_pred"] = bbox\_pred

self.\_predictions["rois"] = rois

self.\_score\_summaries.update(self.\_predictions)

return rois, cls\_prob, bbox\_pred

## 5.3 build\_head（）创建卷积层

这里是一个ZF网络，5个卷积4个池化，经过该网络的特征图与原图大小是1:16

def build\_head(self, is\_training):

# Main network

# Layer 1

net = slim.repeat(self.\_image, 2, slim.conv2d, 64, [3, 3], trainable=False, scope='conv1')

net = slim.max\_pool2d(net, [2, 2], padding='SAME', scope='pool1')

# Layer 2

net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], trainable=False, scope='conv2')

net = slim.max\_pool2d(net, [2, 2], padding='SAME', scope='pool2')

# Layer 3

net = slim.repeat(net, 3, slim.conv2d, 256, [3, 3], trainable=is\_training, scope='conv3')

net = slim.max\_pool2d(net, [2, 2], padding='SAME', scope='pool3')

# Layer 4

net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], trainable=is\_training, scope='conv4')

net = slim.max\_pool2d(net, [2, 2], padding='SAME', scope='pool4')

# Layer 5

net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], trainable=is\_training, scope='conv5')

# Append network to summaries

self.\_act\_summaries.append(net)

# Append network as head layer

self.\_layers['head'] = net

return net

## 5.4 build\_rpn()创建RPN层

见注释的标注

def build\_rpn(self, net, is\_training, initializer):

# Build anchor component

self.\_anchor\_component()

# Create RPN Layer

rpn = slim.conv2d(net, 512, [3, 3], trainable=is\_training, weights\_initializer=initializer, scope="rpn\_conv/3x3")

# 做框的分类任务 self.\_num\_anchors \* 2：一个框有前景或背景两类

self.\_act\_summaries.append(rpn)

rpn\_cls\_score = slim.conv2d(rpn, self.\_num\_anchors \* 2, [1, 1], trainable=is\_training, weights\_initializer=initializer, padding='VALID', activation\_fn=None, scope='rpn\_cls\_score')

# Change it so that the score has 2 as its channel size

rpn\_cls\_score\_reshape = self.\_reshape\_layer(rpn\_cls\_score, 2, 'rpn\_cls\_score\_reshape') # 18个结果

rpn\_cls\_prob\_reshape = self.\_softmax\_layer(rpn\_cls\_score\_reshape, "rpn\_cls\_prob\_reshape")

rpn\_cls\_prob = self.\_reshape\_layer(rpn\_cls\_prob\_reshape, self.\_num\_anchors \* 2, "rpn\_cls\_prob")

# 做框的回归任务self.\_num\_anchors \* 4：一个框的位置信息有4个坐标

rpn\_bbox\_pred = slim.conv2d(rpn, self.\_num\_anchors \* 4, [1, 1], trainable=is\_training, weights\_initializer=initializer, padding='VALID', activation\_fn=None, scope='rpn\_bbox\_pred')

return rpn\_cls\_prob, rpn\_bbox\_pred, rpn\_cls\_score, rpn\_cls\_score\_reshape

5.5 build\_proposals（）创建过滤层

如何过滤见第一节

def build\_proposals(self, is\_training, rpn\_cls\_prob, rpn\_bbox\_pred, rpn\_cls\_score):

if is\_training:

rois, roi\_scores = self.\_proposal\_layer(rpn\_cls\_prob, rpn\_bbox\_pred, "rois")

rpn\_labels = self.\_anchor\_target\_layer(rpn\_cls\_score, "anchor")

# Try to have a deterministic order for the computing graph, for reproducibility

with tf.control\_dependencies([rpn\_labels]):

rois, \_ = self.\_proposal\_target\_layer(rois, roi\_scores, "rpn\_rois")

else:

if cfg.FLAGS.test\_mode == 'nms':

rois, \_ = self.\_proposal\_layer(rpn\_cls\_prob, rpn\_bbox\_pred, "rois")

elif cfg.FLAGS.test\_mode == 'top':

rois, \_ = self.\_proposal\_top\_layer(rpn\_cls\_prob, rpn\_bbox\_pred, "rois")

else:

raise NotImplementedError

return rois

## 5.6 build\_predictions（）创建全连接输出层

def build\_predictions(self, net, rois, is\_training, initializer, initializer\_bbox):

# Crop image ROIs

pool5 = self.\_crop\_pool\_layer(net, rois, "pool5")

pool5\_flat = slim.flatten(pool5, scope='flatten')

# Fully connected layers

fc6 = slim.fully\_connected(pool5\_flat, 4096, scope='fc6')

if is\_training:

fc6 = slim.dropout(fc6, keep\_prob=0.5, is\_training=True, scope='dropout6')

fc7 = slim.fully\_connected(fc6, 4096, scope='fc7')

if is\_training:

fc7 = slim.dropout(fc7, keep\_prob=0.5, is\_training=True, scope='dropout7')

# Scores and predictions

cls\_score = slim.fully\_connected(fc7, self.\_num\_classes, weights\_initializer=initializer, trainable=is\_training, activation\_fn=None, scope='cls\_score')

cls\_prob = self.\_softmax\_layer(cls\_score, "cls\_prob")

bbox\_prediction = slim.fully\_connected(fc7, self.\_num\_classes \* 4, weights\_initializer=initializer\_bbox, trainable=is\_training, activation\_fn=None, scope='bbox\_pred')

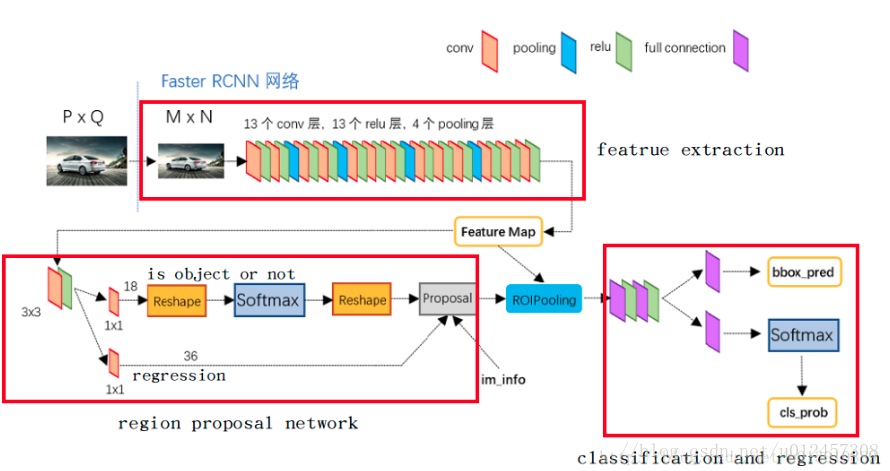
return cls\_score, cls\_prob, bbox\_prediction

# RPN网络

参考文献

<https://blog.csdn.net/u012457308/article/details/79566195>

## RPN网络在Taster-R-CNN算法框架中所在的位置

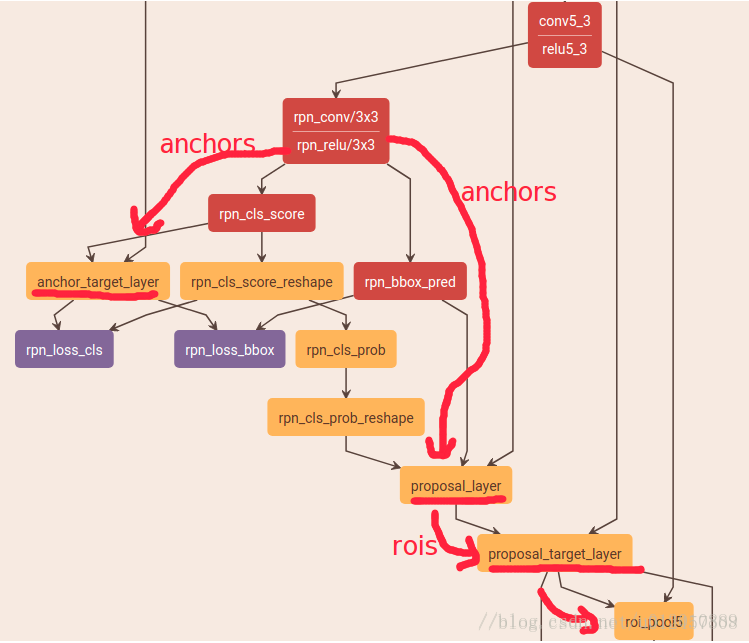


****构建RPN网络的代码为network.py中的build\_network()函数****

****此模块被vgg16网络和netrest\_v1网络所继承, 并且重写了build\_network()方法****

|  |
| --- |
| def \_build\_network(self, is\_training=True):  # select initializers  if cfg.TRAIN.TRUNCATED:  initializer = tf.truncated\_normal\_initializer(mean=0.0, stddev=0.01)  initializer\_bbox = tf.truncated\_normal\_initializer(mean=0.0, stddev=0.001)  else:  initializer = tf.random\_normal\_initializer(mean=0.0, stddev=0.01)  initializer\_bbox = tf.random\_normal\_initializer(mean=0.0, stddev=0.001)  net\_conv = self.\_image\_to\_head(is\_training)  with tf.variable\_scope(self.\_scope, self.\_scope):  # 生成anchors  self.\_anchor\_component()  # RPN网络  rois = self.\_region\_proposal(net\_conv, is\_training, initializer)  # RoI pooling  if cfg.POOLING\_MODE == 'crop':  pool5 = self.\_crop\_pool\_layer(net\_conv, rois, "pool5")  else:  raise NotImplementedError  fc7 = self.\_head\_to\_tail(pool5, is\_training)  with tf.variable\_scope(self.\_scope, self.\_scope):  # 分类/回归网络  cls\_prob, bbox\_pred = self.\_region\_classification(fc7, is\_training,  initializer, initializer\_bbox)  self.\_score\_summaries.update(self.\_predictions)  return rois, cls\_prob, bbox\_pred |

****RPN网络流程图****



这部分介绍RPN网络的构建，首先在Conv5\_3特征图的基础上，生成anchors；然后预测每个anchor的类别及位置

self.\_anchor\_component(), 主要调用layer\_utils/generate\_anchors.py: 生成anchors。

输入图像的尺寸(W, H)， 经过feature extraction模块后，得到尺寸为(W/16, H/16)的特征图，记为(w, h)（VGG16的网络结构，所有stride的乘积为16，具体原理请参考论文）；然后特征图的每个点生成k个anchors，论文中设置3种ratios:[0.5, 1, 2]， 3种sacles:[8, 16, 32]，每个特征图共产生w\*h\*9个anchors。

|  |
| --- |
| # array([[ -83., -39., 100., 56.],  # [-175., -87., 192., 104.],  # [-359., -183., 376., 200.],  # [ -55., -55., 72., 72.],  # [-119., -119., 136., 136.],  # [-247., -247., 264., 264.],  # [ -35., -79., 52., 96.],  # [ -79., -167., 96., 184.],  # [-167., -343., 184., 360.]])  # 上述结果是在batchsize=16（什么意思？）的基础上，即以(0, 0, 15, 15)作为参考窗口，生成9个anchors。注意：生成不同ratio的anchor的时候，anchor的面积保持不变，只是高宽比发生改变。 |

****self.\_region\_proposal()****，首先预测anchors属于前/背景的分数，以及坐标位置。包括两层网络结构：

第一层：3\*3的卷积层

|  |
| --- |
| rpn = slim.conv2d(net\_conv, 512, [3, 3], trainable=is\_training, weights\_initializer=initializer,  scope="rpn\_conv/3x3") |

第二层：两个分支，都用了1\*1的卷积核；第一支得到特征图(height, width, 9\*2)，用于判断bbox中是否含有物体；第二支得到特征图 (height, width, 9\*4)，用于得到bbox的坐标。

|  |
| --- |
| # shape = (1, ?, ?, 18) , 其中，batchsize=1  rpn\_cls\_score = slim.conv2d(rpn, self.\_num\_anchors \* 2, [1, 1], trainable=is\_training,  weights\_initializer=initializer,  # change it so that the score has 2 as its channel size  # shape = (1, ?, ?, 2)  rpn\_cls\_score\_reshape = self.\_reshape\_layer(rpn\_cls\_score, 2, 'rpn\_cls\_score\_reshape')  # shape = (1, ?, ?, 2)  rpn\_cls\_prob\_reshape = self.\_softmax\_layer(rpn\_cls\_score\_reshape, "rpn\_cls\_prob\_reshape")  # shape = (?,)  rpn\_cls\_pred = tf.argmax(tf.reshape(rpn\_cls\_score\_reshape, [-1, 2]), axis=1, name="rpn\_cls\_pred")  # shape = (1, ?, ?, 18)  rpn\_cls\_prob = self.\_reshape\_layer(rpn\_cls\_prob\_reshape, self.\_num\_anchors \* 2, "rpn\_cls\_prob")  # shape = (1, ?, ?, 36)  rpn\_bbox\_pred = slim.conv2d(rpn, self.\_num\_anchors \* 4, [1, 1], trainable=is\_training,  weights\_initializer=initializer,  padding='VALID', activation\_fn=None, scope='rpn\_bbox\_pred') |

\_region\_proposal() 中的\_anchor\_target\_layer()调用anchor\_target\_layer.py函数得到训练RPN所需的标签。为了训练RPN网络，需要构建两个损失函数：用于分类（前景/背景2类）的softmax\_cross\_entropy, 另一类是用于回归bbox的smooth\_l1\_loss。该函数根据cls\_prob和bbox\_pred为anchors分配标签（1：前景，0：背景，-1：忽略），即rpn\_labels；并计算anchor与gt bbox之间的差值， 即rpn\_bbox\_targets。另外，bbox\_inside\_weights, rpn\_bbox\_outside\_weights ？？？？

|  |
| --- |
| def \_anchor\_target\_layer(self, rpn\_cls\_score, name):  rpn\_labels, rpn\_bbox\_targets, rpn\_bbox\_inside\_weights, rpn\_bbox\_outside\_weights = tf.py\_func(  anchor\_target\_layer，[rpn\_cls\_score, self.\_gt\_boxes, self.\_im\_info, self.\_feat\_stride, self.\_anchors, self.\_num\_anchors],  [tf.float32, tf.float32, tf.float32, tf.float32])  #省略了部分代码  self.\_anchor\_targets['rpn\_labels'] = rpn\_labels  self.\_anchor\_targets['rpn\_bbox\_targets'] = rpn\_bbox\_targets  self.\_anchor\_targets['rpn\_bbox\_inside\_weights'] = rpn\_bbox\_inside\_weights  self.\_anchor\_targets['rpn\_bbox\_outside\_weights'] = rpn\_bbox\_outside\_weights  self.\_score\_summaries.update(self.\_anchor\_targets)  return rpn\_labels |

正负样本生成策略：

只保留图像内部的anchors

对于每个gt\_box，找到与它IoU最大的anchor则设为正样本

对于每个anchor，与任意一个gt\_box的IoU>0.7则为正样本，IoU<0.3设为负样本

其他anchor则被忽略

假如正负样本过多，则进行采样，采样比例由RPN\_BATCHSIZE， RPN\_FG\_FRACTION等控制

\_region\_proposal()中的\_proposal\_layer()调用proposal\_layer()函数。功能：生成proposal，并进行筛选（NMS等）。主要流程可概括为以下四点：

利用坐标变换生成proposal：proposals = bbox\_transform\_inv(anchors, rpn\_bbox\_pred)

按前景概率对proposal进行降排，然后留下RPN\_PRE\_NMS\_TOP\_N个proposal

对剩下的proposal进行NMS操作，阈值是0.7

对剩下的proposal，保留RPN\_POST\_NMS\_TOP\_N个， 得到最终的rois和相应的rpn\_socre。

\_region\_proposal()中的\_proposal\_target\_layer()为上一步中得到的proposal分配所属物体类别，并得到proposal和 gt\_bbox的的坐标位置间的差别，便于训练后续Fast R-CNN的分类和回归网络。（注：这一步在测试中没有，因为测试时没有ground truth）

|  |
| --- |
| def \_proposal\_target\_layer(self, rois, roi\_scores, name):  rois, roi\_scores, labels, bbox\_targets, bbox\_inside\_weights, bbox\_outside\_weights = tf.py\_func(  proposal\_target\_layer, [rois, roi\_scores, self.\_gt\_boxes, self.\_num\_classes],  [tf.float32, tf.float32, tf.float32, tf.float32, tf.float32, tf.float32])  self.\_proposal\_targets['rois'] = rois  self.\_proposal\_targets['labels'] = tf.to\_int32(labels, name="to\_int32")  self.\_proposal\_targets['bbox\_targets'] = bbox\_targets  self.\_proposal\_targets['bbox\_inside\_weights'] = bbox\_inside\_weights  self.\_proposal\_targets['bbox\_outside\_weights'] = bbox\_outside\_weights  return rois, roi\_scores |

主要调用****proposal\_target\_layer()****函数，其主要步骤如下：

1. 确定每张图片中roi的数目，以及前景fg\_roi的数目
2. 从所有的rpn\_rois中进行采样，并得到rois的类别标签以及bbox的回归目标（bbox\_targets），即真值与预测值之间的偏差。

|  |
| --- |
| labels, rois, roi\_scores, bbox\_targets, bbox\_inside\_weights = \_sample\_rois( all\_rois,  all\_scores, gt\_boxes, fg\_rois\_per\_image, rois\_per\_image, \_num\_classes) |

计算rois与gt\_bboxes之间的overlap矩阵，对于每一个roi，最大的overlap的gt\_bbox的标签即为该roi的类别标签，并根据TRAIN.FG\_THRESH和TRAIN.BG\_THRESH\_HI/LO 选择前景roi和背景roi。

小结：

RPN网络主要进行了三个工作：

预测anchor的类别（属于前景/背景）及其位置

self.\_predictions["rpn\_cls\_score"] = rpn\_cls\_score

self.\_predictions["rpn\_cls\_score\_reshape"] = rpn\_cls\_score\_reshape

self.\_predictions["rpn\_cls\_prob"] = rpn\_cls\_prob

self.\_predictions["rpn\_cls\_pred"] = rpn\_cls\_pred

self.\_predictions["rpn\_bbox\_pred"] = rpn\_bbox\_pred

self.\_predictions["rois"] = rois

生成训练RPN网络的标签信息（anchor target layer）

self.\_anchor\_targets['rpn\_labels'] = rpn\_labels

self.\_anchor\_targets['rpn\_bbox\_targets'] = rpn\_bbox\_targets

self.\_anchor\_targets['rpn\_bbox\_inside\_weights'] = rpn\_bbox\_inside\_weights

self.\_anchor\_targets['rpn\_bbox\_outside\_weights'] = rpn\_bbox\_outside\_weights

生成训练分类和回归网络的RoI（proposal layer）以及对应的标签信息（proposal target layer）

self.\_proposal\_targets['rois'] = rois

self.\_proposal\_targets['labels'] = tf.to\_int32(labels, name="to\_int32")

self.\_proposal\_targets['bbox\_targets'] = bbox\_targets

self.\_proposal\_targets['bbox\_inside\_weights'] = bbox\_inside\_weights

self.\_proposal\_targets['bbox\_outside\_weights'] = bbox\_outside\_weights

# RoI Pooling

FC layer需要固定尺寸的输入。在最早的R-CNN算法中，将输入的图像直接resize成相同的尺寸。而Faster R-CNN对输入图像的尺寸没有要求，经过Proposal layer和 Proposal target layer之后，会得到许多不同尺寸的RoI。Faster R-CNN采用RoI Pooling层（原理参考SPPNet 论文），将不同尺寸ROI对应的特征图采样为相同尺寸，然后输入后续的FC层。这版代码中没有实现RoI pooling layer， 而是把RoI对应的特征图resize成相同尺寸后，再进行max pooling。

# 没有实现RoI pooling layer

pool5 = self.\_crop\_pool\_layer(net\_conv, rois, "pool5")

def \_crop\_pool\_layer(self, bottom, rois, name):

with tf.variable\_scope(name) as scope:

batch\_ids = tf.squeeze(tf.slice(rois, [0, 0], [-1, 1], name="batch\_id"), [1])

# 得到归一化的bbox坐标（相对原图的尺寸进行归一化）

bottom\_shape = tf.shape(bottom)

height = (tf.to\_float(bottom\_shape[1]) - 1.) \* np.float32(self.\_feat\_stride[0])

width = (tf.to\_float(bottom\_shape[2]) - 1.) \* np.float32(self.\_feat\_stride[0])

x1 = tf.slice(rois, [0, 1], [-1, 1], name="x1") / width

y1 = tf.slice(rois, [0, 2], [-1, 1], name="y1") / height

x2 = tf.slice(rois, [0, 3], [-1, 1], name="x2") / width

y2 = tf.slice(rois, [0, 4], [-1, 1], name="y2") / height

# Won't be back-propagated to rois anyway, but to save time

bboxes = tf.stop\_gradient(tf.concat([y1, x1, y2, x2], axis=1))

pre\_pool\_size = cfg.POOLING\_SIZE \* 2

# 裁剪特征图，并resize成相同的尺寸

crops = tf.image.crop\_and\_resize(bottom, bboxes, tf.to\_int32(batch\_ids), [pre\_pool\_size, pre\_pool\_size], name="crops")

# 进行标准的max pooling

return slim.max\_pool2d(crops, [2, 2], padding='SAME')

需要说明的是，我感觉这是一种比较取巧的方法。和标准的ROI pooling之间有什么区别，还是本质上是等价的？

## ROIs

在Fast RCNN中，指的是Selective Search的输出；在Faster RCNN中指的是RPN的输出，一堆矩形候选框框，形状为1x5x1x1（4个坐标+索引index），其中值得注意的是：坐标的参考系不是针对feature map这张图的，而是针对原图的（神经网络最开始的输入）。下面给出roi pooling层的流程及代码（C++）。

坐标映射。将roi坐标映射到feature map

int roi\_start\_w = round(rois\_flat[index\_roi + 1] \* spatial\_scale); // spatial\_scale 1/16

int roi\_start\_h = round(rois\_flat[index\_roi + 2] \* spatial\_scale);

int roi\_end\_w = round(rois\_flat[index\_roi + 3] \* spatial\_scale);

int roi\_end\_h = round(rois\_flat[index\_roi + 4] \* spatial\_scale);

1

2

3

4

在feature map上的roi区域做max pooling或者average pooling。

% 确定pooling的窗口。应为roi的尺寸不同，所以窗口的尺寸也会自适应变化

float bin\_size\_h = (float)(roi\_height) / (float)(pooled\_height); // 9/7

float bin\_size\_w = (float)(roi\_width) / (float)(pooled\_width); // 7/7=1

for (ph = 0; ph < pooled\_height; ++ph){

for (pw = 0; pw < pooled\_width; ++pw){

int hstart = (floor((float)(ph) \* bin\_size\_h));

int wstart = (floor((float)(pw) \* bin\_size\_w));

int hend = (ceil((float)(ph + 1) \* bin\_size\_h));

int wend = (ceil((float)(pw + 1) \* bin\_size\_w));

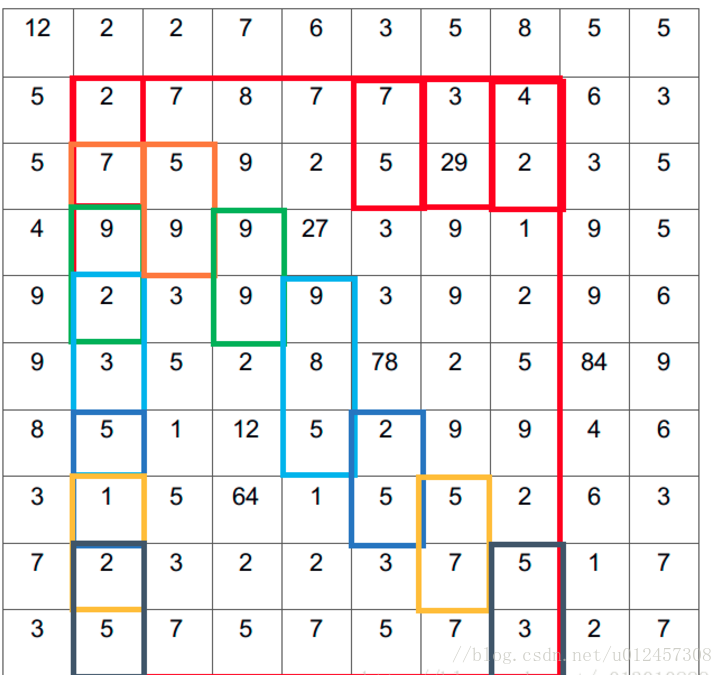
hstart = fminf(fmaxf(hstart + roi\_start\_h, 0), data\_height);

hend = fminf(fmaxf(hend + roi\_start\_h, 0), data\_height);

wstart = fminf(fmaxf(wstart + roi\_start\_w, 0), data\_width);

wend = fminf(fmaxf(wend + roi\_start\_w, 0), data\_width);

% max/average pooling 这部分代码省略



# ****Classification and regression分类回归网络****

参考文献

<https://blog.csdn.net/u012457308/article/details/79566195>

直接上代码

def \_region\_classification(self, fc7, is\_training, initializer, initializer\_bbox):

# 分类

cls\_score = slim.fully\_connected(fc7, self.\_num\_classes,

weights\_initializer=initializer,

trainable=is\_training,

activation\_fn=None, scope='cls\_score')

cls\_prob = self.\_softmax\_layer(cls\_score, "cls\_prob")

cls\_pred = tf.argmax(cls\_score, axis=1, name="cls\_pred")

# 回归

bbox\_pred = slim.fully\_connected(fc7, self.\_num\_classes \* 4,

weights\_initializer=initializer\_bbox,

trainable=is\_training,

activation\_fn=None, scope='bbox\_pred')

self.\_predictions["cls\_score"] = cls\_score

self.\_predictions["cls\_pred"] = cls\_pred

self.\_predictions["cls\_prob"] = cls\_prob

self.\_predictions["bbox\_pred"] = bbox\_pred

return cls\_prob, bbox\_pred

小结：

至此，数据准备和整个Faster R-CNN的网络已经搭建完成。为了训练网络，需要构建损失函数。

Loss

Loss分为4部分：RPN, class loss，RPN, bbox loss，RCNN, class loss，RCNN, bbox loss。

# RPN, class loss

rpn\_cls\_score = tf.reshape(self.\_predictions['rpn\_cls\_score\_reshape'], [-1, 2])

rpn\_label = tf.reshape(self.\_anchor\_targets['rpn\_labels'], [-1])

rpn\_select = tf.where(tf.not\_equal(rpn\_label, -1))

rpn\_cls\_score = tf.reshape(tf.gather(rpn\_cls\_score, rpn\_select), [-1, 2])

rpn\_label = tf.reshape(tf.gather(rpn\_label, rpn\_select), [-1])

rpn\_cross\_entropy = tf.reduce\_mean(

tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=rpn\_cls\_score, labels=rpn\_label))

# RPN, bbox loss

rpn\_bbox\_pred = self.\_predictions['rpn\_bbox\_pred']

rpn\_bbox\_targets = self.\_anchor\_targets['rpn\_bbox\_targets']

rpn\_bbox\_inside\_weights = self.\_anchor\_targets['rpn\_bbox\_inside\_weights']

rpn\_bbox\_outside\_weights = self.\_anchor\_targets['rpn\_bbox\_outside\_weights']

rpn\_loss\_box = self.\_smooth\_l1\_loss(rpn\_bbox\_pred, rpn\_bbox\_targets, rpn\_bbox\_inside\_weights,

rpn\_bbox\_outside\_weights, sigma=sigma\_rpn, dim=[1, 2, 3])

# RCNN, class loss

cls\_score = self.\_predictions["cls\_score"]

label = tf.reshape(self.\_proposal\_targets["labels"], [-1])

cross\_entropy = tf.reduce\_mean(

tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(

logits=tf.reshape(cls\_score, [-1, self.\_num\_classes]), labels=label))

# RCNN, bbox loss

bbox\_pred = self.\_predictions['bbox\_pred']

bbox\_targets = self.\_proposal\_targets['bbox\_targets']

bbox\_inside\_weights = self.\_proposal\_targets['bbox\_inside\_weights']

bbox\_outside\_weights = self.\_proposal\_targets['bbox\_outside\_weights']

loss\_box = self.\_smooth\_l1\_loss(bbox\_pred, bbox\_targets, bbox\_inside\_weights, bbox\_outside\_weights)

分类loss都采用的是：softmax\_cross\_entropy；回归loss都采用的是：smooth\_L1\_loss

模型训练

论文中采用4步交替训练策略。

先用预训练好的ImageNet来初始化RPN网络，然后微调(finetune)RPN网络；

根据第一步训练好的RPN来生成RoIs，然后单独训练 Fast R-CNN。在这一步训练过程中，Fast R-CNN的参数初始化也是采用ImageNet预训练的模型。两个网络完全分开训练，不存在共享网络层。

采用上一步Fast R-CNN训练好的网络参数，来重新初始化RPN的共享卷积层。（注意：这一步只对RPN的局部网络进行微调，前半部分和Fast R-CNN共享的卷积层训练好后就固定不变了）

继续固定共享网络层参数，用步骤3微调后的RPN网络生成的bbox对Fast R-CNN的非共享层进行参数微调。

本文所用的代码采用近似联合训练策略。

思路：把RPN的损失函数和Fast R-CNN的损失函数根据一定比例加在一起，然后进行整体的SGD训练

\*\*问题：\*\*RPN后续的网络层，无法对RPN的bbox坐标进行求导更新，即ROI的误差无法反向传播到RPN网络，因此只能称之为近似联合训练。

loss = cross\_entropy + loss\_box + rpn\_cross\_entropy + rpn\_loss\_box

## 损失函数

## 优化器

# 优化  
gvs = optimizer.compute\_gradients(loss)

调用的是tensorflow.python.training.optimizer

def compute\_gradients(self, loss, var\_list=None,  
 gate\_gradients=GATE\_OP,  
 aggregation\_method=None,  
 colocate\_gradients\_with\_ops=False,  
 grad\_loss=None):

### 官方描述

*C*ompute gradients of `loss` for the variables in `var\_list`.  
计算“var\_list”中变量的“loss”梯度  
This is the first part of `minimize()`. It returns a list  
of (gradient, variable) pairs where "gradient" is the gradient  
for "variable". Note that "gradient" can be a `Tensor`, an  
`IndexedSlices`, or `None` if there is no gradient for the  
given variable.

这是“minimize（）”的第一部分。它返回一个列表（梯度，变量）对，其中“梯度”是梯度对于“变量”。注意“gradient”可以是一个“Tensor”，一个`indexedlices`，如果没有给定变量。  
  
Args:

参数  
 loss: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.

损失：包含要最小化的值或可调用的取数的张量没有返回要最小化的值的参数。当急切的执行已启用它必须是可调用的。

var\_list: Optional list or tuple of `tf.Variable` to update to minimize `loss`. Defaults to the list of variables collected in the graph under the key `GraphKeys.TRAINABLE\_VARIABLES`.

var\_list：要更新以最小化的可选列表或'tf.Variable'元组`损失。默认为图表中收集的变量列表在“GraphKeys.TRAINABLE”变量下。门梯度：如何门梯度的计算。可以是`门无”、“门操作”或“门图”。

gate\_gradients: How to gate the computation of gradients. Can be `GATE\_NONE`, `GATE\_OP`, or `GATE\_GRAPH`.

门梯度：如何门梯度的计算。可以是“GATE\_NONE”、“GATE\_OP”或“GATE\_GRAPH”。

aggregation\_method: Specifies the method used to combine gradient terms.Valid values are defined in the class `AggregationMethod`.

aggregation\_method：指定用于组合渐变项的方法。有效值在“AggregationMethod”类中定义。

colocate\_gradients\_with\_ops: If True, try colocating gradients with the corresponding op.  
colocate\_gradients\_with\_ops: 如果为真，请尝试使用相应的op复制渐变。

grad\_loss: Optional. A `Tensor` holding the gradient computed for `loss`.

grad\_loss: 可选。一种“张量”，用于保持为“loss”计算的梯度。  
  
Returns:  
 A list of (gradient, variable) pairs. Variable is always present, but gradient can be `None`.

(渐变，变量）对的列表。变量始终存在，但渐变可以为“None”。  
  
Raises:  
 TypeError: If `var\_list` contains anything else than `Variable` objects.

类型错误:如果“varu list”包含除“Variable”对象之外的任何内容。  
 ValueError: If some arguments are invalid.

值错误:某些参数无效  
 RuntimeError: If called with eager execution enabled and `loss` is not callable.

运行错误:如果调用时启用了紧急执行，并且“loss”不可调用  
  
@compatibility(eager)  
When eager execution is enabled, `gate\_gradients`, `aggregation\_method`,  
and `colocate\_gradients\_with\_ops` are ignored.  
@end\_compatibility

## 开始迭代训练

# 补充说明

## 程序运行时间

方法1

import datetime

starttime = datetime.datetime.now()

#long running

endtime = datetime.datetime.now()

print (endtime - starttime).seconds

方法 2

start = time.time()

run\_fun()

end = time.time()

print end-start

方法3

start = time.clock()

run\_fun()

end = time.clock()

print end-start

方法1和方法2都包含了其他程序使用CPU的时间，是程序开始到程序结束的运行时间。

方法3算只计算了程序运行的CPU时间

在train模块的main方法中添加程序运行时间, 可以查看当程序运行完毕时, 所消耗的时间是多少.

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  Import datetime  # start running time  starttime = datetime.datetime.now()    train = Train()  train.train()   # end running time  endtime = datetime.datetime.now()  print("程序运行时间:",endtime - starttime).seconds |

# 案例使用流程

|  |
| --- |
| # tf-faster-rcnn Tensorflow Faster R-CNN for Windows and Linux by using Python 3  This is the branch to compile Faster R-CNN on Windows and Linux. It is heavily inspired by the great work done [here](https://github.com/smallcorgi/Faster-RCNN\_TF) and [here](https://github.com/rbgirshick/py-faster-rcnn). I have not implemented anything new but I fixed the implementations for Windows, Linux and Python 3.  Currently, this repository supports Python 3.5, 3.6 and 3.7. Thanks to @morpheusthewhite  ### PLEASE BE AWARE: I do not have time or intention to fix all the issues for this branch as I do not use it commercially. I created this branch just for fun. If you want to make any commitment, it is more than welcome. Tensorflow has already released an object detection api. Please refer to it. https://github.com/tensorflow/models/tree/master/research/object\_detection  ### If you find a solution to an existing issue in the code, please send a PR for it.  ### Also, instead of trying to deal with Tensorflow, use Chainer. It is ready to be used with all the common models https://github.com/chainer/chainercv & https://github.com/chainer/chainer . I can reply all of your questions about Chainer  # How To Use This Branch 1. Install tensorflow, preferably GPU version. Follow [instructions]( https://www.tensorflow.org/install/install\_windows). If you do not install GPU version, you need to comment out all the GPU calls inside code and replace them with relavent CPU ones.  安装tensorflow，最好是GPU版本。遵循[说明]（https://www.tensorflow.org/install/install\_windows）。如果不安装GPU版本，则需要注释掉代码中的所有GPU调用，并用相关的CPU调用替换它们。  2. Checkout this branch  3. Install python packages (cython, python-opencv, easydict) by running  `pip install -r requirements.txt`  (if you are using an environment manager system such as `conda` you should follow its instruction)  4. Go to ./data/coco/PythonAPI  Run `python setup.py build\_ext --inplace`  Run `python setup.py build\_ext install`  Go to ./lib/utils and run `python setup.py build\_ext --inplace`  5. Follow [these instructions]  (https://github.com/rbgirshick/py-faster-rcnn#beyond-the-demo-installation-for-training-and-testing-models) to download PyCoco database. I will be glad if you can contribute with a batch script to automatically download and fetch. The final structure has to look like  `data\VOCDevkit2007\VOC2007`   1. Download pre-trained VGG16 from [here]  (http://download.tensorflow.org/models/vgg\_16\_2016\_08\_28.tar.gz) and place it as `data\imagenet\_weights\vgg16.ckpt`. For rest of the models, please check [here](https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models)  7. Run train.py  Notify me if there is any issue found. |

# 运行环境

TensorFlow-GPU

使用ResNEt50(残差神经网络)主干特征提取

学习前言

什么是FasterRCNN目标检测算法

源码下载

Faster-RCNN实现思路

一、预测部分

1、主干网络介绍

2、获得Proposal建议框

3、Proposal建议框的解码

4、对Proposal建议框加以利用（RoiPoolingConv）

5、在原图上进行绘制

6、整体的执行流程

二、训练部分

1、建议框网络的训练

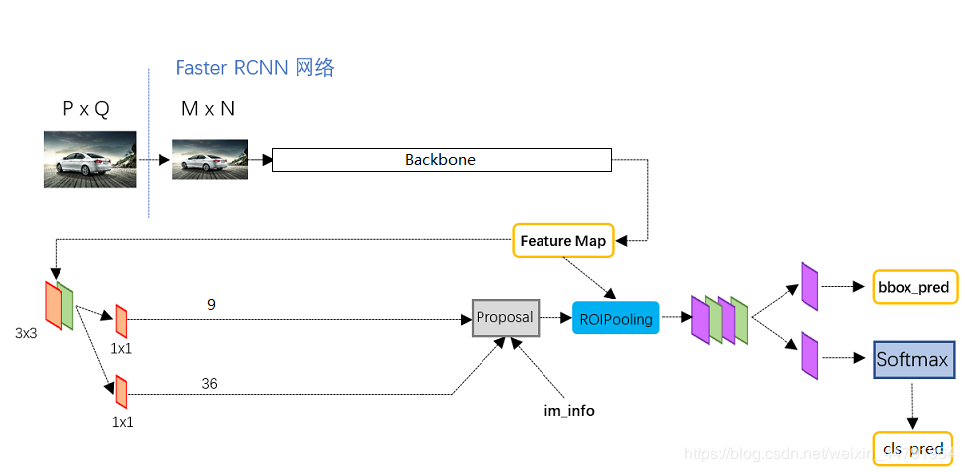
2、Roi网络的训练

训练自己的Faster-RCNN模型

学习前言

最近对实例分割感兴趣了，不过实例分割MaskRCNN是基于FasterRCNN的，之前学了非常多的One-Stage的目标检测算法，对FasterRCNN并不感兴趣，这次我们来学学FasterRCNN。

什么是FasterRCNN目标检测算法



Faster-RCNN是一个非常有效的目标检测算法，虽然是一个比较早的论文， 但它至今仍是许多目标检测算法的基础。

Faster-RCNN作为一种two-stage的算法，与one-stage的算法相比，two-stage的算法更加复杂且速度较慢，但是检测精度会更高。

事实上也确实是这样，Faster-RCNN的检测效果非常不错，但是检测速度与训练速度有待提高。

源码下载

https://github.com/bubbliiiing/faster-rcnn-keras

喜欢的可以点个star噢。

Faster-RCNN实现思路

一、预测部分

1、主干网络介绍

Faster-RCNN可以采用多种的主干特征提取网络，常用的有VGG，Resnet，Xception等等，本文采用的是Resnet网络，关于Resnet的介绍大家可以看我的另外一篇博客https://blog.csdn.net/weixin\_44791964/article/details/102790260。

FasterRcnn对输入进来的图片尺寸没有固定，但是一般会把输入进来的图片短边固定成600，如输入一张1200x1800的图片，会把图片不失真的resize到600x900上。

ResNet50有两个基本的块，分别名为Conv Block和Identity Block，其中Conv Block输入和输出的维度是不一样的，所以不能连续串联，它的作用是改变网络的维度；Identity Block输入维度和输出维度相同，可以串联，用于加深网络的。

Conv Block的结构如下：

Identity Block的结构如下：

这两个都是残差网络结构。

Faster-RCNN的主干特征提取网络部分只包含了长宽压缩了四次的内容，第五次压缩后的内容在ROI中使用。即Faster-RCNN在主干特征提取网络所用的网络层如图所示。

以输入的图片为600x600为例，shape变化如下：

最后一层的输出就是公用特征层。

实现代码：

def identity\_block(input\_tensor, kernel\_size, filters, stage, block):

filters1, filters2, filters3 = filters

conv\_name\_base = 'res' + str(stage) + block + '\_branch'

bn\_name\_base = 'bn' + str(stage) + block + '\_branch'

x = Conv2D(filters1, (1, 1), name=conv\_name\_base + '2a')(input\_tensor)

x = BatchNormalization(name=bn\_name\_base + '2a')(x)

x = Activation('relu')(x)

x = Conv2D(filters2, kernel\_size,padding='same', name=conv\_name\_base + '2b')(x)

x = BatchNormalization(name=bn\_name\_base + '2b')(x)

x = Activation('relu')(x)

x = Conv2D(filters3, (1, 1), name=conv\_name\_base + '2c')(x)

x = BatchNormalization(name=bn\_name\_base + '2c')(x)

x = layers.add([x, input\_tensor])

x = Activation('relu')(x)

return x

def conv\_block(input\_tensor, kernel\_size, filters, stage, block, strides=(2, 2)):

filters1, filters2, filters3 = filters

conv\_name\_base = 'res' + str(stage) + block + '\_branch'

bn\_name\_base = 'bn' + str(stage) + block + '\_branch'

x = Conv2D(filters1, (1, 1), strides=strides,

name=conv\_name\_base + '2a')(input\_tensor)

x = BatchNormalization(name=bn\_name\_base + '2a')(x)

x = Activation('relu')(x)

x = Conv2D(filters2, kernel\_size, padding='same',

name=conv\_name\_base + '2b')(x)

x = BatchNormalization(name=bn\_name\_base + '2b')(x)

x = Activation('relu')(x)

x = Conv2D(filters3, (1, 1), name=conv\_name\_base + '2c')(x)

x = BatchNormalization(name=bn\_name\_base + '2c')(x)

shortcut = Conv2D(filters3, (1, 1), strides=strides,

name=conv\_name\_base + '1')(input\_tensor)

shortcut = BatchNormalization(name=bn\_name\_base + '1')(shortcut)

x = layers.add([x, shortcut])

x = Activation('relu')(x)

return x

def ResNet50(inputs):

img\_input = inputs

x = ZeroPadding2D((3, 3))(img\_input)

x = Conv2D(64, (7, 7), strides=(2, 2), name='conv1')(x)

x = BatchNormalization(name='bn\_conv1')(x)

x = Activation('relu')(x)

x = MaxPooling2D((3, 3), strides=(2, 2), padding="same")(x)

x = conv\_block(x, 3, [64, 64, 256], stage=2, block='a', strides=(1, 1))

x = identity\_block(x, 3, [64, 64, 256], stage=2, block='b')

x = identity\_block(x, 3, [64, 64, 256], stage=2, block='c')

x = conv\_block(x, 3, [128, 128, 512], stage=3, block='a')

x = identity\_block(x, 3, [128, 128, 512], stage=3, block='b')

x = identity\_block(x, 3, [128, 128, 512], stage=3, block='c')

x = identity\_block(x, 3, [128, 128, 512], stage=3, block='d')

x = conv\_block(x, 3, [256, 256, 1024], stage=4, block='a')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='b')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='c')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='d')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='e')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='f')

return x

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

2、获得Proposal建议框

获得的公用特征层在图像中就是Feature Map，其有两个应用，一个是和ROIPooling结合使用、另一个是进行一次3x3的卷积后，进行一个9通道的1x1卷积，还有一个36通道的1x1卷积。

在Faster-RCNN中，num\_priors也就是先验框的数量就是9，所以两个1x1卷积的结果实际上也就是：

9 x 4的卷积 用于预测 公用特征层上 每一个网格点上 每一个先验框的变化情况。（为什么说是变化情况呢，这是因为Faster-RCNN的预测结果需要结合先验框获得预测框，预测结果就是先验框的变化情况。）

9 x 1的卷积 用于预测 公用特征层上 每一个网格点上 每一个预测框内部是否包含了物体。

当我们输入的图片的shape是600x600x3的时候，公用特征层的shape就是38x38x1024，相当于把输入进来的图像分割成38x38的网格，然后每个网格存在9个先验框，这些先验框有不同的大小，在图像上密密麻麻。

9 x 4的卷积的结果会对这些先验框进行调整，获得一个新的框。

9 x 1的卷积会判断上述获得的新框是否包含物体。

到这里我们可以获得了一些有用的框，这些框会利用9 x 1的卷积判断是否存在物体。

到此位置还只是粗略的一个框的获取，也就是一个建议框。然后我们会在建议框里面继续找东西。

实现代码为：

def get\_rpn(base\_layers, num\_anchors):

x = Conv2D(512, (3, 3), padding='same', activation='relu', kernel\_initializer='normal', name='rpn\_conv1')(base\_layers)

x\_class = Conv2D(num\_anchors, (1, 1), activation='sigmoid', kernel\_initializer='uniform', name='rpn\_out\_class')(x)

x\_regr = Conv2D(num\_anchors \* 4, (1, 1), activation='linear', kernel\_initializer='zero', name='rpn\_out\_regress')(x)

x\_class = Reshape((-1,1),name="classification")(x\_class)

x\_regr = Reshape((-1,4),name="regression")(x\_regr)

return [x\_class, x\_regr, base\_layers]

1

2

3

4

5

6

7

8

9

3、Proposal建议框的解码

通过第二步我们获得了38x38x9个先验框的预测结果。预测结果包含两部分。

9 x 4的卷积 用于预测 公用特征层上 每一个网格点上 每一个先验框的变化情况。\*\*

9 x 1的卷积 用于预测 公用特征层上 每一个网格点上 每一个预测框内部是否包含了物体。

相当于就是将整个图像分成38x38个网格；然后从每个网格中心建立9个先验框，一共38x38x9个，12996个先验框。

当输入图像shape不同时，先验框的数量也会发生改变。

先验框虽然可以代表一定的框的位置信息与框的大小信息，但是其是有限的，无法表示任意情况，因此还需要调整。

9 x 4中的9表示了这个网格点所包含的先验框数量，其中的4表示了框的中心与长宽的调整情况。

实现代码如下：

def decode\_boxes(self, mbox\_loc, mbox\_priorbox):

# 获得先验框的宽与高

prior\_width = mbox\_priorbox[:, 2] - mbox\_priorbox[:, 0]

prior\_height = mbox\_priorbox[:, 3] - mbox\_priorbox[:, 1]

# 获得先验框的中心点

prior\_center\_x = 0.5 \* (mbox\_priorbox[:, 2] + mbox\_priorbox[:, 0])

prior\_center\_y = 0.5 \* (mbox\_priorbox[:, 3] + mbox\_priorbox[:, 1])

# 真实框距离先验框中心的xy轴偏移情况

decode\_bbox\_center\_x = mbox\_loc[:, 0] \* prior\_width / 4

decode\_bbox\_center\_x += prior\_center\_x

decode\_bbox\_center\_y = mbox\_loc[:, 1] \* prior\_height / 4

decode\_bbox\_center\_y += prior\_center\_y

# 真实框的宽与高的求取

decode\_bbox\_width = np.exp(mbox\_loc[:, 2] / 4)

decode\_bbox\_width \*= prior\_width

decode\_bbox\_height = np.exp(mbox\_loc[:, 3] /4)

decode\_bbox\_height \*= prior\_height

# 获取真实框的左上角与右下角

decode\_bbox\_xmin = decode\_bbox\_center\_x - 0.5 \* decode\_bbox\_width

decode\_bbox\_ymin = decode\_bbox\_center\_y - 0.5 \* decode\_bbox\_height

decode\_bbox\_xmax = decode\_bbox\_center\_x + 0.5 \* decode\_bbox\_width

decode\_bbox\_ymax = decode\_bbox\_center\_y + 0.5 \* decode\_bbox\_height

# 真实框的左上角与右下角进行堆叠

decode\_bbox = np.concatenate((decode\_bbox\_xmin[:, None],

decode\_bbox\_ymin[:, None],

decode\_bbox\_xmax[:, None],

decode\_bbox\_ymax[:, None]), axis=-1)

# 防止超出0与1

decode\_bbox = np.minimum(np.maximum(decode\_bbox, 0.0), 1.0)

return decode\_bbox

def detection\_out(self, predictions, mbox\_priorbox, num\_classes, keep\_top\_k=300,

confidence\_threshold=0.5):

# 网络预测的结果

# 置信度

mbox\_conf = predictions[0]

mbox\_loc = predictions[1]

# 先验框

mbox\_priorbox = mbox\_priorbox

results = []

# 对每一个图片进行处理

for i in range(len(mbox\_loc)):

results.append([])

decode\_bbox = self.decode\_boxes(mbox\_loc[i], mbox\_priorbox)

for c in range(num\_classes):

c\_confs = mbox\_conf[i, :, c]

c\_confs\_m = c\_confs > confidence\_threshold

if len(c\_confs[c\_confs\_m]) > 0:

# 取出得分高于confidence\_threshold的框

boxes\_to\_process = decode\_bbox[c\_confs\_m]

confs\_to\_process = c\_confs[c\_confs\_m]

# 进行iou的非极大抑制

feed\_dict = {self.boxes: boxes\_to\_process,

self.scores: confs\_to\_process}

idx = self.sess.run(self.nms, feed\_dict=feed\_dict)

# 取出在非极大抑制中效果较好的内容

good\_boxes = boxes\_to\_process[idx]

confs = confs\_to\_process[idx][:, None]

# 将label、置信度、框的位置进行堆叠。

labels = c \* np.ones((len(idx), 1))

c\_pred = np.concatenate((labels, confs, good\_boxes),

axis=1)

# 添加进result里

results[-1].extend(c\_pred)

if len(results[-1]) > 0:

# 按照置信度进行排序

results[-1] = np.array(results[-1])

argsort = np.argsort(results[-1][:, 1])[::-1]

results[-1] = results[-1][argsort]

# 选出置信度最大的keep\_top\_k个

results[-1] = results[-1][:keep\_top\_k]

# 获得，在所有预测结果里面，置信度比较高的框

# 还有，利用先验框和Faster-RCNN的预测结果，处理获得了真实框（预测框）的位置

return results

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

4、对Proposal建议框加以利用（RoiPoolingConv）

让我们对建议框有一个整体的理解：

事实上建议框就是对图片哪一个区域有物体存在进行初步筛选。

通过主干特征提取网络，我们可以获得一个公用特征层，当输入图片为600x600x3的时候，它的shape是38x38x1024，然后建议框会对这个公用特征层进行截取。

其实公用特征层里面的38x38对应着图片里的38x38个区域，38x38中的每一个点相当于这个区域内部所有特征的浓缩。

建议框会对这38x38个区域进行截取，也就是认为这些区域里存在目标，然后将截取的结果进行resize，resize到14x14x1024的大小。

每次输入的建议框的数量默认情况是32。

然后再对每个建议框再进行Resnet原有的第五次压缩。压缩完后进行一个平均池化，再进行一个Flatten，最后分别进行一个num\_classes的全连接和(num\_classes-1)x4全连接。

num\_classes的全连接用于对最后获得的框进行分类，(num\_classes-1)x4全连接用于对相应的建议框进行调整，之所以-1是不包括被认定为背景的框。

通过这些操作，我们可以获得所有建议框的调整情况，和这个建议框调整后框内物体的类别。

事实上，在上一步获得的建议框就是ROI的先验框。

对Proposal建议框加以利用的过程与shape变化如图所示：

建议框调整后的结果就是最终的预测结果了，可以在图上进行绘画了。

class RoiPoolingConv(Layer):

def \_\_init\_\_(self, pool\_size, num\_rois, \*\*kwargs):

self.dim\_ordering = K.image\_dim\_ordering()

assert self.dim\_ordering in {'tf', 'th'}, 'dim\_ordering must be in {tf, th}'

self.pool\_size = pool\_size

self.num\_rois = num\_rois

super(RoiPoolingConv, self).\_\_init\_\_(\*\*kwargs)

def build(self, input\_shape):

self.nb\_channels = input\_shape[0][3]

def compute\_output\_shape(self, input\_shape):

return None, self.num\_rois, self.pool\_size, self.pool\_size, self.nb\_channels

def call(self, x, mask=None):

assert(len(x) == 2)

img = x[0]

rois = x[1]

outputs = []

for roi\_idx in range(self.num\_rois):

x = rois[0, roi\_idx, 0]

y = rois[0, roi\_idx, 1]

w = rois[0, roi\_idx, 2]

h = rois[0, roi\_idx, 3]

x = K.cast(x, 'int32')

y = K.cast(y, 'int32')

w = K.cast(w, 'int32')

h = K.cast(h, 'int32')

rs = tf.image.resize\_images(img[:, y:y+h, x:x+w, :], (self.pool\_size, self.pool\_size))

outputs.append(rs)

final\_output = K.concatenate(outputs, axis=0)

final\_output = K.reshape(final\_output, (1, self.num\_rois, self.pool\_size, self.pool\_size, self.nb\_channels))

final\_output = K.permute\_dimensions(final\_output, (0, 1, 2, 3, 4))

return final\_output

def identity\_block\_td(input\_tensor, kernel\_size, filters, stage, block, trainable=True):

nb\_filter1, nb\_filter2, nb\_filter3 = filters

if K.image\_dim\_ordering() == 'tf':

bn\_axis = 3

else:

bn\_axis = 1

conv\_name\_base = 'res' + str(stage) + block + '\_branch'

bn\_name\_base = 'bn' + str(stage) + block + '\_branch'

x = TimeDistributed(Conv2D(nb\_filter1, (1, 1), trainable=trainable, kernel\_initializer='normal'), name=conv\_name\_base + '2a')(input\_tensor)

x = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '2a')(x)

x = Activation('relu')(x)

x = TimeDistributed(Conv2D(nb\_filter2, (kernel\_size, kernel\_size), trainable=trainable, kernel\_initializer='normal',padding='same'), name=conv\_name\_base + '2b')(x)

x = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '2b')(x)

x = Activation('relu')(x)

x = TimeDistributed(Conv2D(nb\_filter3, (1, 1), trainable=trainable, kernel\_initializer='normal'), name=conv\_name\_base + '2c')(x)

x = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '2c')(x)

x = Add()([x, input\_tensor])

x = Activation('relu')(x)

return x

def conv\_block\_td(input\_tensor, kernel\_size, filters, stage, block, input\_shape, strides=(2, 2), trainable=True):

nb\_filter1, nb\_filter2, nb\_filter3 = filters

if K.image\_dim\_ordering() == 'tf':

bn\_axis = 3

else:

bn\_axis = 1

conv\_name\_base = 'res' + str(stage) + block + '\_branch'

bn\_name\_base = 'bn' + str(stage) + block + '\_branch'

x = TimeDistributed(Conv2D(nb\_filter1, (1, 1), strides=strides, trainable=trainable, kernel\_initializer='normal'), input\_shape=input\_shape, name=conv\_name\_base + '2a')(input\_tensor)

x = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '2a')(x)

x = Activation('relu')(x)

x = TimeDistributed(Conv2D(nb\_filter2, (kernel\_size, kernel\_size), padding='same', trainable=trainable, kernel\_initializer='normal'), name=conv\_name\_base + '2b')(x)

x = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '2b')(x)

x = Activation('relu')(x)

x = TimeDistributed(Conv2D(nb\_filter3, (1, 1), kernel\_initializer='normal'), name=conv\_name\_base + '2c', trainable=trainable)(x)

x = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '2c')(x)

shortcut = TimeDistributed(Conv2D(nb\_filter3, (1, 1), strides=strides, trainable=trainable, kernel\_initializer='normal'), name=conv\_name\_base + '1')(input\_tensor)

shortcut = TimeDistributed(BatchNormalization(axis=bn\_axis), name=bn\_name\_base + '1')(shortcut)

x = Add()([x, shortcut])

x = Activation('relu')(x)

return x

def classifier\_layers(x, input\_shape, trainable=False):

x = conv\_block\_td(x, 3, [512, 512, 2048], stage=5, block='a', input\_shape=input\_shape, strides=(2, 2), trainable=trainable)

x = identity\_block\_td(x, 3, [512, 512, 2048], stage=5, block='b', trainable=trainable)

x = identity\_block\_td(x, 3, [512, 512, 2048], stage=5, block='c', trainable=trainable)

x = TimeDistributed(AveragePooling2D((7, 7)), name='avg\_pool')(x)

return x

def get\_classifier(base\_layers, input\_rois, num\_rois, nb\_classes=21, trainable=False):

pooling\_regions = 14

input\_shape = (num\_rois, 14, 14, 1024)

out\_roi\_pool = RoiPoolingConv(pooling\_regions, num\_rois)([base\_layers, input\_rois])

out = classifier\_layers(out\_roi\_pool, input\_shape=input\_shape, trainable=True)

out = TimeDistributed(Flatten())(out)

out\_class = TimeDistributed(Dense(nb\_classes, activation='softmax', kernel\_initializer='zero'), name='dense\_class\_{}'.format(nb\_classes))(out)

out\_regr = TimeDistributed(Dense(4 \* (nb\_classes-1), activation='linear', kernel\_initializer='zero'), name='dense\_regress\_{}'.format(nb\_classes))(out)

return [out\_class, out\_regr]

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

5、在原图上进行绘制

在第四步的结尾，我们对建议框进行再一次进行解码后，我们可以获得预测框在原图上的位置，而且这些预测框都是经过筛选的。这些筛选后的框可以直接绘制在图片上，就可以获得结果了。

6、整体的执行流程

几个小tip：

1、共包含了两次解码过程。

2、先进行粗略的筛选再细调。

3、第一次获得的建议框解码后的结果是对共享特征层featuremap进行截取。

二、训练部分

Faster-RCNN的训练过程和它的预测过程一样，分为两部分，首先要训练获得建议框网络，然后再训练后面利用ROI获得预测结果的网络。

1、建议框网络的训练

公用特征层如果要获得建议框的预测结果，需要再进行一次3x3的卷积后，进行一个9通道的1x1卷积，还有一个36通道的1x1卷积。

在Faster-RCNN中，num\_priors也就是先验框的数量就是9，所以两个1x1卷积的结果实际上也就是：

9 x 4的卷积 用于预测 公用特征层上 每一个网格点上 每一个先验框的变化情况。（为什么说是变化情况呢，这是因为Faster-RCNN的预测结果需要结合先验框获得预测框，预测结果就是先验框的变化情况。）

9 x 1的卷积 用于预测 公用特征层上 每一个网格点上 每一个预测框内部是否包含了物体。

也就是说，我们直接利用Faster-RCNN建议框网络预测到的结果，并不是建议框在图片上的真实位置，需要解码才能得到真实位置。

而在训练的时候，我们需要计算loss函数，这个loss函数是相对于Faster-RCNN建议框网络的预测结果的。我们需要把图片输入到当前的Faster-RCNN建议框的网络中，得到建议框的结果；同时还需要进行编码，这个编码是把真实框的位置信息格式转化为Faster-RCNN建议框预测结果的格式信息。

也就是，我们需要找到 每一张用于训练的图片的每一个真实框对应的先验框，并求出如果想要得到这样一个真实框，我们的建议框预测结果应该是怎么样的。

从建议框预测结果获得真实框的过程被称作解码，而从真实框获得建议框预测结果的过程就是编码的过程。

因此我们只需要将解码过程逆过来就是编码过程了。

实现代码如下：

def encode\_box(self, box, return\_iou=True):

iou = self.iou(box)

encoded\_box = np.zeros((self.num\_priors, 4 + return\_iou))

# 找到每一个真实框，重合程度较高的先验框

assign\_mask = iou > self.overlap\_threshold

if not assign\_mask.any():

assign\_mask[iou.argmax()] = True

if return\_iou:

encoded\_box[:, -1][assign\_mask] = iou[assign\_mask]

# 找到对应的先验框

assigned\_priors = self.priors[assign\_mask]

# 逆向编码，将真实框转化为Retinanet预测结果的格式

# 先计算真实框的中心与长宽

box\_center = 0.5 \* (box[:2] + box[2:])

box\_wh = box[2:] - box[:2]

# 再计算重合度较高的先验框的中心与长宽

assigned\_priors\_center = 0.5 \* (assigned\_priors[:, :2] +

assigned\_priors[:, 2:4])

assigned\_priors\_wh = (assigned\_priors[:, 2:4] -

assigned\_priors[:, :2])

# 逆向求取ssd应该有的预测结果

encoded\_box[:, :2][assign\_mask] = box\_center - assigned\_priors\_center

encoded\_box[:, :2][assign\_mask] /= assigned\_priors\_wh

encoded\_box[:, :2][assign\_mask] \*= 4

encoded\_box[:, 2:4][assign\_mask] = np.log(box\_wh / assigned\_priors\_wh)

encoded\_box[:, 2:4][assign\_mask] \*= 4

return encoded\_box.ravel()

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

利用上述代码我们可以获得，真实框对应的所有的iou较大先验框，并计算了真实框对应的所有iou较大的先验框应该有的预测结果。

但是由于原始图片中可能存在多个真实框，可能同一个先验框会与多个真实框重合度较高，我们只取其中与真实框重合度最高的就可以了。

因此我们还要经过一次筛选，将上述代码获得的真实框对应的所有的iou较大先验框的预测结果中，iou最大的那个真实框筛选出来。

通过assign\_boxes我们就获得了，输入进来的这张图片，应该有的预测结果是什么样子的。

实现代码如下：

def iou(self, box):

# 计算出每个真实框与所有的先验框的iou

# 判断真实框与先验框的重合情况

inter\_upleft = np.maximum(self.priors[:, :2], box[:2])

inter\_botright = np.minimum(self.priors[:, 2:4], box[2:])

inter\_wh = inter\_botright - inter\_upleft

inter\_wh = np.maximum(inter\_wh, 0)

inter = inter\_wh[:, 0] \* inter\_wh[:, 1]

# 真实框的面积

area\_true = (box[2] - box[0]) \* (box[3] - box[1])

# 先验框的面积

area\_gt = (self.priors[:, 2] - self.priors[:, 0])\*(self.priors[:, 3] - self.priors[:, 1])

# 计算iou

union = area\_true + area\_gt - inter

iou = inter / union

return iou

def encode\_box(self, box, return\_iou=True):

iou = self.iou(box)

encoded\_box = np.zeros((self.num\_priors, 4 + return\_iou))

# 找到每一个真实框，重合程度较高的先验框

assign\_mask = iou > self.overlap\_threshold

if not assign\_mask.any():

assign\_mask[iou.argmax()] = True

if return\_iou:

encoded\_box[:, -1][assign\_mask] = iou[assign\_mask]

# 找到对应的先验框

assigned\_priors = self.priors[assign\_mask]

# 逆向编码，将真实框转化为Retinanet预测结果的格式

# 先计算真实框的中心与长宽

box\_center = 0.5 \* (box[:2] + box[2:])

box\_wh = box[2:] - box[:2]

# 再计算重合度较高的先验框的中心与长宽

assigned\_priors\_center = 0.5 \* (assigned\_priors[:, :2] +

assigned\_priors[:, 2:4])

assigned\_priors\_wh = (assigned\_priors[:, 2:4] -

assigned\_priors[:, :2])

# 逆向求取ssd应该有的预测结果

encoded\_box[:, :2][assign\_mask] = box\_center - assigned\_priors\_center

encoded\_box[:, :2][assign\_mask] /= assigned\_priors\_wh

encoded\_box[:, :2][assign\_mask] \*= 4

encoded\_box[:, 2:4][assign\_mask] = np.log(box\_wh / assigned\_priors\_wh)

encoded\_box[:, 2:4][assign\_mask] \*= 4

return encoded\_box.ravel()

def ignore\_box(self, box):

iou = self.iou(box)

ignored\_box = np.zeros((self.num\_priors, 1))

# 找到每一个真实框，重合程度较高的先验框

assign\_mask = (iou > self.ignore\_threshold)&(iou<self.overlap\_threshold)

if not assign\_mask.any():

assign\_mask[iou.argmax()] = True

ignored\_box[:, 0][assign\_mask] = iou[assign\_mask]

return ignored\_box.ravel()

def assign\_boxes(self, boxes, anchors):

self.num\_priors = len(anchors)

self.priors = anchors

assignment = np.zeros((self.num\_priors, 4 + 1))

assignment[:, 4] = 0.0

if len(boxes) == 0:

return assignment

# 对每一个真实框都进行iou计算

ingored\_boxes = np.apply\_along\_axis(self.ignore\_box, 1, boxes[:, :4])

# 取重合程度最大的先验框，并且获取这个先验框的index

ingored\_boxes = ingored\_boxes.reshape(-1, self.num\_priors, 1)

# (num\_priors)

ignore\_iou = ingored\_boxes[:, :, 0].max(axis=0)

# (num\_priors)

ignore\_iou\_mask = ignore\_iou > 0

assignment[:, 4][ignore\_iou\_mask] = -1

# (n, num\_priors, 5)

encoded\_boxes = np.apply\_along\_axis(self.encode\_box, 1, boxes[:, :4])

# 每一个真实框的编码后的值，和iou

# (n, num\_priors)

encoded\_boxes = encoded\_boxes.reshape(-1, self.num\_priors, 5)

# 取重合程度最大的先验框，并且获取这个先验框的index

# (num\_priors)

best\_iou = encoded\_boxes[:, :, -1].max(axis=0)

# (num\_priors)

best\_iou\_idx = encoded\_boxes[:, :, -1].argmax(axis=0)

# (num\_priors)

best\_iou\_mask = best\_iou > 0

# 某个先验框它属于哪个真实框

best\_iou\_idx = best\_iou\_idx[best\_iou\_mask]

assign\_num = len(best\_iou\_idx)

# 保留重合程度最大的先验框的应该有的预测结果

# 哪些先验框存在真实框

encoded\_boxes = encoded\_boxes[:, best\_iou\_mask, :]

assignment[:, :4][best\_iou\_mask] = encoded\_boxes[best\_iou\_idx,np.arange(assign\_num),:4]

# 4代表为背景的概率，为0

assignment[:, 4][best\_iou\_mask] = 1

# 通过assign\_boxes我们就获得了，输入进来的这张图片，应该有的预测结果是什么样子的

return assignment

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

focal会忽略一些重合度相对较高但是不是非常高的先验框，一般将重合度在0.3-0.7之间的先验框进行忽略。

2、Roi网络的训练

通过上一步已经可以对建议框网络进行训练了，建议框网络会提供一些位置的建议，在ROI网络部分，其会将建议框根据进行一定的截取，并获得对应的预测结果，事实上就是将上一步建议框当作了ROI网络的先验框。

因此，我们需要计算所有建议框和真实框的重合程度，并进行筛选，如果某个真实框和建议框的重合程度大于0.5则认为该建议框为正样本，如果重合程度小于0.5大于0.1则认为该建议框为负样本

因此我们可以对真实框进行编码，这个编码是相对于建议框的，也就是，当我们存在这些建议框的时候，我们的ROI预测网络需要有什么样的预测结果才能将这些建议框调整成真实框。

每次训练我们都放入32个建议框进行训练，同时要注意正负样本的平衡。

实现代码如下：

# 编码

def calc\_iou(R, config, all\_boxes, width, height, num\_classes):

# print(all\_boxes)

bboxes = all\_boxes[:,:4]

gta = np.zeros((len(bboxes), 4))

for bbox\_num, bbox in enumerate(bboxes):

gta[bbox\_num, 0] = int(round(bbox[0]\*width/config.rpn\_stride))

gta[bbox\_num, 1] = int(round(bbox[1]\*height/config.rpn\_stride))

gta[bbox\_num, 2] = int(round(bbox[2]\*width/config.rpn\_stride))

gta[bbox\_num, 3] = int(round(bbox[3]\*height/config.rpn\_stride))

x\_roi = []

y\_class\_num = []

y\_class\_regr\_coords = []

y\_class\_regr\_label = []

IoUs = []

# print(gta)

for ix in range(R.shape[0]):

x1 = R[ix, 0]\*width/config.rpn\_stride

y1 = R[ix, 1]\*height/config.rpn\_stride

x2 = R[ix, 2]\*width/config.rpn\_stride

y2 = R[ix, 3]\*height/config.rpn\_stride

x1 = int(round(x1))

y1 = int(round(y1))

x2 = int(round(x2))

y2 = int(round(y2))

# print([x1, y1, x2, y2])

best\_iou = 0.0

best\_bbox = -1

for bbox\_num in range(len(bboxes)):

curr\_iou = iou([gta[bbox\_num, 0], gta[bbox\_num, 1], gta[bbox\_num, 2], gta[bbox\_num, 3]], [x1, y1, x2, y2])

if curr\_iou > best\_iou:

best\_iou = curr\_iou

best\_bbox = bbox\_num

# print(best\_iou)

if best\_iou < config.classifier\_min\_overlap:

continue

else:

w = x2 - x1

h = y2 - y1

x\_roi.append([x1, y1, w, h])

IoUs.append(best\_iou)

if config.classifier\_min\_overlap <= best\_iou < config.classifier\_max\_overlap:

label = -1

elif config.classifier\_max\_overlap <= best\_iou:

label = int(all\_boxes[best\_bbox,-1])

cxg = (gta[best\_bbox, 0] + gta[best\_bbox, 2]) / 2.0

cyg = (gta[best\_bbox, 1] + gta[best\_bbox, 3]) / 2.0

cx = x1 + w / 2.0

cy = y1 + h / 2.0

tx = (cxg - cx) / float(w)

ty = (cyg - cy) / float(h)

tw = np.log((gta[best\_bbox, 2] - gta[best\_bbox, 0]) / float(w))

th = np.log((gta[best\_bbox, 3] - gta[best\_bbox, 1]) / float(h))

else:

print('roi = {}'.format(best\_iou))

raise RuntimeError

# print(label)

class\_label = num\_classes \* [0]

class\_label[label] = 1

y\_class\_num.append(copy.deepcopy(class\_label))

coords = [0] \* 4 \* (num\_classes - 1)

labels = [0] \* 4 \* (num\_classes - 1)

if label != -1:

label\_pos = 4 \* label

sx, sy, sw, sh = config.classifier\_regr\_std

coords[label\_pos:4+label\_pos] = [sx\*tx, sy\*ty, sw\*tw, sh\*th]

labels[label\_pos:4+label\_pos] = [1, 1, 1, 1]

y\_class\_regr\_coords.append(copy.deepcopy(coords))

y\_class\_regr\_label.append(copy.deepcopy(labels))

else:

y\_class\_regr\_coords.append(copy.deepcopy(coords))

y\_class\_regr\_label.append(copy.deepcopy(labels))

if len(x\_roi) == 0:

return None, None, None, None

X = np.array(x\_roi)

# print(X)

Y1 = np.array(y\_class\_num)

Y2 = np.concatenate([np.array(y\_class\_regr\_label),np.array(y\_class\_regr\_coords)],axis=1)

return np.expand\_dims(X, axis=0), np.expand\_dims(Y1, axis=0), np.expand\_dims(Y2, axis=0), IoUs

# 正负样本平衡

X2, Y1, Y2, IouS = calc\_iou(R, config, boxes[0], width, height, NUM\_CLASSES)

if X2 is None:

rpn\_accuracy\_rpn\_monitor.append(0)

rpn\_accuracy\_for\_epoch.append(0)

continue

neg\_samples = np.where(Y1[0, :, -1] == 1)

pos\_samples = np.where(Y1[0, :, -1] == 0)

if len(neg\_samples) > 0:

neg\_samples = neg\_samples[0]

else:

neg\_samples = []

if len(pos\_samples) > 0:

pos\_samples = pos\_samples[0]

else:

pos\_samples = []

rpn\_accuracy\_rpn\_monitor.append(len(pos\_samples))

rpn\_accuracy\_for\_epoch.append((len(pos\_samples)))

if len(neg\_samples)==0:

continue

if len(pos\_samples) < config.num\_rois//2:

selected\_pos\_samples = pos\_samples.tolist()

else:

selected\_pos\_samples = np.random.choice(pos\_samples, config.num\_rois//2, replace=False).tolist()

try:

selected\_neg\_samples = np.random.choice(neg\_samples, config.num\_rois - len(selected\_pos\_samples), replace=False).tolist()

except:

selected\_neg\_samples = np.random.choice(neg\_samples, config.num\_rois - len(selected\_pos\_samples), replace=True).tolist()

sel\_samples = selected\_pos\_samples + selected\_neg\_samples

loss\_class = model\_classifier.train\_on\_batch([X, X2[:, sel\_samples, :]], [Y1[:, sel\_samples, :], Y2[:, sel\_samples, :]])

训练自己的Faster-RCNN模型

Faster-RCNN整体的文件夹构架如下：

本文使用VOC格式进行训练。

训练前将标签文件放在VOCdevkit文件夹下的VOC2007文件夹下的Annotation中。

训练前将图片文件放在VOCdevkit文件夹下的VOC2007文件夹下的JPEGImages中。

在训练前利用voc2faster-rcnn.py文件生成对应的txt。

再运行根目录下的voc\_annotation.py，运行前需要将classes改成你自己的classes。

classes = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

1

就会生成对应的2007\_train.txt，每一行对应其图片位置及其真实框的位置。

在训练前需要修改model\_data里面的voc\_classes.txt文件，需要将classes改成你自己的classes。

运行train.py即可开始训练。

# 批量预测

这里的预测为单张预测，需要手动输入图片的名字，可以把while True一下的所有内容改为：  
fi = open('VOCdevkit/VOC2007/ImageSets/Main/test.txt')  
txt = fi.readlines()  
im\_names = 【】  
for line in txt:  
line = line.strip('\n')  
line = ('VOCdevkit/VOC2007/JPEGImages/'+line+'.jpg')  
im\_names.append(line)  
print(im\_names)  
fi.close()  
for im\_name in im\_names:  
image = Image.open(im\_name)  
r\_image = frcnn.detect\_image(image)  
r\_image.show()  
frcnn.close\_session()  
进行批量测试

# 利用共享特征层获取建议框

共享特征层(Feature Map)经过一个3\*3卷积, 和两个1\*1的卷积.

# 报错

运行了voc2faster-rcnn.py，就会报错Traceback (most recent call last):  
File "D:/rcc/VOCdevkit/VOC2007/voc2faster-rcnn.py", line 10, in <module>  
temp\_xml = os.listdir(xmlfilepath)  
FileNotFoundError: 【WinError 3】 系统找不到指定的路径。: './VOCdevkit/VOC2007/Annotations'

解决方案

voc2faster-rcnn.py在voc2007文件夹下，所以xmlfilepath='./Annotations'  
saveBasePath="./ImageSets/Main/"

错误描述

|  |
| --- |
| WARNING: You are using pip version 20.0.2; however, version 20.1 is available.  You should consider upgrading via the 'c:\users\administrator\appdata\local\programs\python\python37\python.exe -m pip install --upgrade pip' command. |

解决方法

|  |
| --- |
| ****python -m pip install --upgrade pip -i** <https://pypi.douban.com/simple>** |

# python生成唯一标识uuid

import uuid

s\_uuid = str(uuid.uuid4())  
l\_uuid = s\_uuid.split('-')  
s\_uuid = ''.join(l\_uuid)

注意：正常调用UUID时，返回的结果是4717cf79-1a64-4156-a3e8-883f1b76210d，字符串中间有“-”连接。使用split可以去掉字符串中间的“-”。

# 批量命名图片的代码

|  |
| --- |
| import os  path = "JPEGImages"  filelist = os.listdir(path) #该文件夹下所有的文件（包括文件夹）  # 命名规则从什么数字开始  count=0  for file in filelist:  print(file)  for file in filelist: #遍历所有文件  Olddir=os.path.join(path,file) #原来的文件路径  if os.path.isdir(Olddir): #如果是文件夹则跳过  continue  filename=os.path.splitext(file)[0] #文件名  filetype=os.path.splitext(file)[1] #文件扩展名  Newdir=os.path.join(path,str(count).zfill(6)+filetype) #用字符串函数zfill 以0补全所需位数  os.rename(Olddir,Newdir)#重命名  count+=1 |

### **所需环境**

tensorflow-gpu==1.13.1  
 keras==2.1.5

### **文件下载**

训练所需的voc\_weights.h5可以去百度网盘下载  
链接: <https://pan.baidu.com/s/1xDRhw0U4dWfy_2rceH-YnA> 提取码: cm3q

### **训练步骤**

1、本文使用VOC格式进行训练。  
2、训练前将标签文件放在VOCdevkit文件夹下的VOC2007文件夹下的Annotation中。  
3、训练前将图片文件放在VOCdevkit文件夹下的VOC2007文件夹下的JPEGImages中。  
4、在训练前利用voc2faster-rcnn.py文件生成对应的txt。  
5、再运行根目录下的voc\_annotation.py，运行前需要将classes改成你自己的classes。

classes = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

6、就会生成对应的2007\_train.txt，每一行对应其图片位置及其真实框的位置。  
7、在训练前需要修改model\_data里面的voc\_classes.txt文件，需要将classes改成你自己的classes。  
8、运行train.py即可开始训练。

### **mAP目标检测精度计算更新**

更新了get\_gt\_txt.py、get\_dr\_txt.py和get\_map.py文件。  
get\_map文件克隆自<https://github.com/Cartucho/mAP>  
具体mAP计算过程可参考：<https://www.bilibili.com/video/BV1zE411u7Vw>

### **Reference**

<https://github.com/qqwweee/keras-yolo3/>  
<https://github.com/pierluigiferrari/ssd_keras>  
<https://github.com/kuhung/SSD_keras>  
<https://github.com/jinfagang/keras_frcnn>  
<https://github.com/Cartucho/mAP>

# 训练自己的数据集

## 参考文献

1. 复现Faster-RCNN-TensorFlow-Python3.5代码，并用自己的数据集来训练（二）.<https://blog.csdn.net/kellyroslyn/article/details/93294790>

## demo.py验证案例介绍

## 第一步:图片命名规范

把图片弄成大小和Voc数据集差不多，重命名成000000.jpg格式.

批量更改名称

|  |
| --- |
| import os  filepath = 'G:\\data'  filelist = os.listdir(filepath)  k = 0  for file in filelist:  olddir = os.path.join(filepath,file)  if os.path.isdir(olddir): #判断当前文件是否为文件夹，是则跳过  continue  filetype = os.path.splitext(file)[1] #得到图片后缀名    #zfill方法返回指定长度的字符串，原字符串右对齐，前面填充0  newdir = os.path.join(filepath,str(k).zfill(6)+filetype)  os.rename(olddir,newdir)  k = k + 1 |

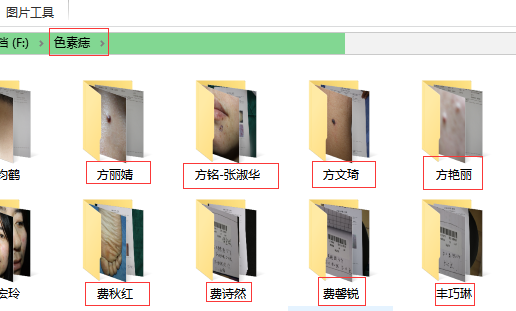
## 第二步：图片格式规范

使用Faster-R-CNN-TensorFlow2-Python3训练数据时，图片的格式要求为时jpg格式。

|  |
| --- |
| import os  from PIL import Image    filepath = 'G:\\数据预处理\\alldata'  new\_path = 'G:\\数据预处理\\alldatajpg'  filelist = os.listdir(filepath)  for file in filelist:  if os.path.splitext(file)[1] == '.png':  img = Image.open(os.path.join(filepath,file))  img = img.convert('RGB')  newimg\_name = file.replace(".png",".jpg")  img.save(os.path.join(new\_path,newimg\_name)) |

## 第三步：图片整理为一个文件夹

数据资源可能并不在同一个文件夹内，比如以下案例，色素痣文件夹内存放的都是色素痣的相关图片。



但是，此资源图片是由医院提供的病人案例，所以具体的病灶图片是在个人文件夹内，手动将所以病灶图片合并到一个文件夹内费时费力，十几G大小的图片手动得干上一天的时间。为此，整理出此步骤希望以后再参考时可以引以为戒，切勿再干出如此傻事。能用机器，就不要用手。

### 实现思路

获取父目录下所有非目录的子文件，将子文件的绝对地址保存到一个列表中，遍历列表读取所有的绝对路径，新建jpg文件，将旧的二进制内容输入到新的文件中。

### 图片复制

|  |
| --- |
| fp1=open("e:\\1.jpeg","rb") 读字节  content=fp1.read()  fp1=open("e:\\new\_1.jpg","wb") 写字节  fp1.write(content)  fp1.close() |

### 获取指定目录下的所以文件

|  |
| --- |
| L = [] for root, dirs, files in os.walk(directories\_path):  # print(root) # 当前目录路径  # print(dirs) # 当前路径下所有子目录  # print(files) # 当前路径下所有非目录子文件  for file in files:  # if os.path.splitext(file)[1] == '.JPG':  # L.append(os.path.join(root, file))  L.append(os.path.join(root, file)) print(L) print(len(L)) |

### 判断文件后缀名

if os.path.splitext(file)[1] == '.JPG':  
 L.append(os.path.join(root, file))

### 完整代码

在Faster-R-CNN-TensorFlow2-Python3项目的lib/utils/PictureSize.py脚本中添加一个专门用来统一一个文件的方法image\_unified()，有两个参数directories\_path，Save\_path。

directories\_path是需要被统一文件的父路径。

Save\_path是将统一好的文件存放路径。

|  |
| --- |
| class PictureSize():  def image\_unified(self, directories\_path, Sava\_path):  *"""  @Author: 徐峰  @create time: 2020/5/28  @Function: 将同一个目录下所有的子文件夹里的图片合并到一个文件夹下  @param directories\_path: 含有子文件的父目录  @param Sava\_path: 将子文件夹里的图片合并到那个文件夹下  """* L = []  for root, dirs, files in os.walk(directories\_path):  # print(root) # 当前目录路径  # print(dirs) # 当前路径下所有子目录  # print(files) # 当前路径下所有非目录子文件  for file in files:  # if os.path.splitext(file)[1] == '.JPG':  # L.append(os.path.join(root, file))  L.append(os.path.join(root, file))  print(L)  print(len(L))  for i in L:  fp1 = open(i, "rb")  # 读字节  content = fp1.read()   s\_uuid = str(uuid.uuid4())  l\_uuid = s\_uuid.split('-')  s\_uuid = ''.join(l\_uuid)  fp1 = open(os.path.join(Sava\_path, s\_uuid+'.jpg'), "wb")  # 写字节  fp1.write(content)  fp1.close() |

调用该方法

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  pictureSize = PictureSize()  directories\_path = r"G:\PyCharm\data\_set\data\picture\色素痣\门诊色素痣\S色素痣"  Sava\_path = r"F:\色素痣"  pictureSize.image\_unified(directories\_path, Sava\_path) |

## 第二步:数据标注

使用[labelImg](https://github.com/tzutalin/labelImg)对数据进行标注的.

labelImg下载地址:

<https://github.com/tzutalin/labelImg>

### **Windows安装**labelImg

安装PyQt5\_sip, 此库是PyQt5的依赖库

pip install -i <https://pypi.tuna.tsinghua.edu.cn/simple> PyQt5\_sip

Install [Python](https://www.python.org/downloads/windows/), [PyQt5](https://www.riverbankcomputing.com/software/pyqt/download5) and [install lxml](http://lxml.de/installation.html).

Open cmd and go to the [labelImg](https://github.com/tzutalin/labelImg" \l "labelimg) directory

pyrcc4 -o lib/resources.py resources.qrc

For pyqt5, pyrcc5 -o libs/resources.py resources.qrc

python labelImg.py

python labelImg.py [IMAGE\_PATH] [PRE-DEFINED CLASS FILE]

详细安装过程参考以下链接

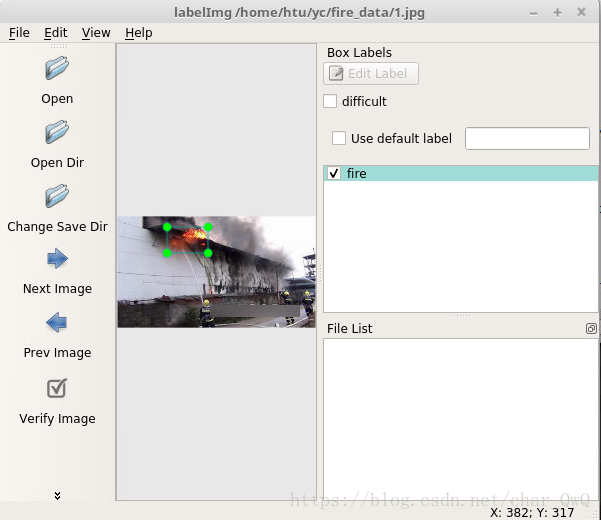
<https://blog.csdn.net/rj1457365980/article/details/83536590>

pyrcc5 -o resources.py resources.qrc

将Qt文件格式转为Python格式

会在labelImg目录下生成一个resources.py的文件

labelImg界面如下图所示



### xml数据格式介绍

|  |
| --- |
| <annotation>  <folder>fire\_data</folder>  <filename>000001.jpg</filename> #图像名称  <path>/home/htu/yc/fire\_data/000001.jpg</path> #路径  <source>  <database>Unknown</database>  </source>  <size>  <width>450</width> #宽  <height>253</height> #高  <depth>3</depth> #RGB三通道  </size>  <segmented>0</segmented>  <object>  <name>fire</name> #目标名称  <pose>Unspecified</pose>  <truncated>0</truncated>  <difficult>0</difficult>  <bndbox> #BBOX值  <xmin>146</xmin>  <ymin>22</ymin>  <xmax>207</xmax>  <ymax>60</ymax>  </bndbox>  </object>  </annotation> |

其中需要说明的是：图像名称一定要命名为000001.jpg，以及xml也要命名为000001.xml格式(事实上如果你对图像名称是六位数字组成的，xml也会自动生成六位数字)。我之前用的是1,2,3,4这样命名，结果运行总是会提示Keyerror:"1"。

一般出现keyerror的原因有两种，一是图像以及xml命名不符合要求，这就需要你重命名；二是标注的object信息有误，什么意思呢？拿我的数据来说，我是对火焰进行标注的，所以label是fire，如果因为操作失误导致fire变成如w或者其他字母，就会提示keyerror。为什么会是w呢？因为标注的快捷键是w，所以有时候会因为粗心把w写到label里。我就犯过这样的错误，这样的错误排除起来很麻烦，还需要再写一段代码，对xml的所有name节点的值进行访问，如果不是你的label，把对应的索引记录下来，手动再修改。

### 生成XML操作步骤：

打开labelImg，在OpenDir中，选择JPEGimage文件夹

（1）W键快捷creatbox，进行框选待测图片中的检测目标；

（2）输入目标的标签，后面和config.py，pascal\_voc.py等调整时匹配；

（2）左侧save到Annotations文件夹下，存为xml文件（放到Annotations里）；

（3）下一张图片继续，直至结束。

然后打开labelImg.py标注数据，然后生成ImageSet/Main/下面的四个txt文件。

然后分别把Annotations、JPRGImages、ImageSets/Main三个文件放到data/VOCdevkit2007/VOC2007文件夹下。

Annotations文件夹里是制作好的标签

JPEGImages文件夹里是源图像

ImageSets\Main 文件夹里是生成的txt文件

### 运行labelImg

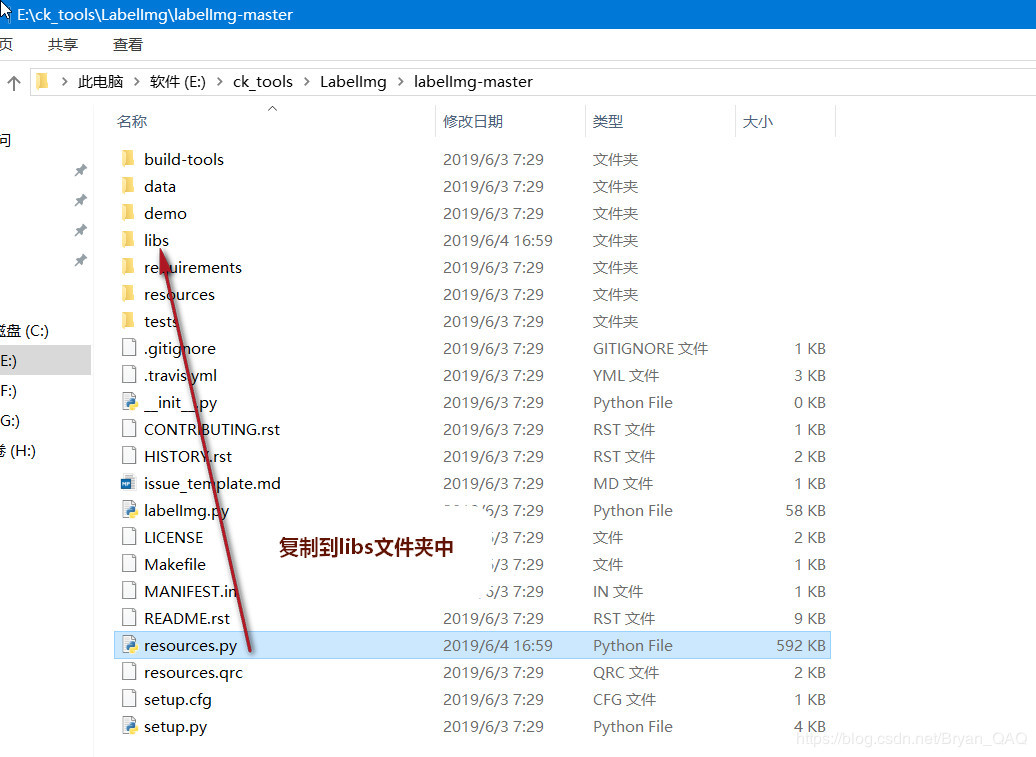
python [labelImg.py](http://labelimg.py/)

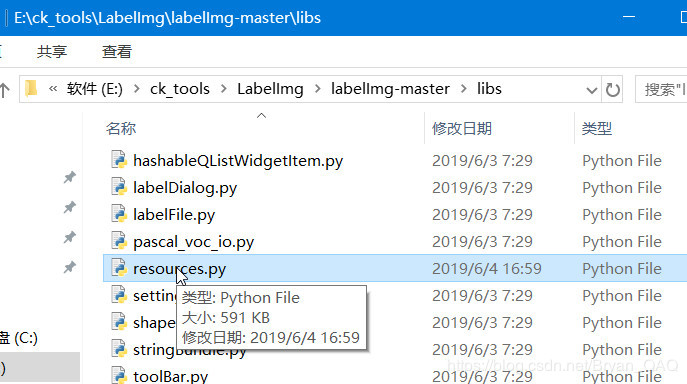
### 运行报错

libs.resources

解决方法

将生成的resources.py拷贝到同级的libs目录下：





### 生成txt文件

Txt文件的内容

图片名字（无后缀）,test.txt是测试集，train.txt是训练集，val.txt是验证集，trainval.txt是训练和验证集.VOC2007中，trainval大概是整个数据集的50%，test也大概是整个数据集的50%；train大概是trainval的50%，val大概是trainval的50%。

此步直接跳转到第五步.

### labellmg保存位置

我将labellmg保存在了G:\PyCharm\Doc\whl和jar\文件夹下。

## 第三步:划分数据集

划分训练验证以及测试集，这里我使用的别人写好的划分代码Creat\_FRCNN\_DataSet(<https://github.com/ruyiweicas/Creat_FRCNN_DataSet>)，是matlab的，下载后更改下VOC2007txt.m中d 路径就能用了。执行结果会按比例对你的数据划分为训练集验证集以及测试集，同时会生成四个txt文档，分别是test.txt，train.txt，trainval.txt，val.txt。

也可以使用Python脚本划分数据集。

|  |
| --- |
| # 生成ImageSets/Main下的 txt文件 import os import random   root\_path = r'/data/VOCdevkit2007/VOC2007'  xmlfilepath = root\_path + '/Annotations'  txtsavepath = root\_path + '/ImageSets/Main'   if not os.path.exists(txtsavepath):  os.makedirs(txtsavepath)  trainval\_percent = 0.9 train\_percent = 0.8 total\_xml = os.listdir(xmlfilepath) num = len(total\_xml) list = range(num) tv = int(num \* trainval\_percent) tr = int(tv \* train\_percent) trainval = random.sample(list, tv) train = random.sample(trainval, tr)  print("train and val size:", tv) print("train size:", tr)  ftrainval = open(txtsavepath + '/trainval.txt', 'w') ftest = open(txtsavepath + '/test.txt', 'w') ftrain = open(txtsavepath + '/train.txt', 'w') fval = open(txtsavepath + '/val.txt', 'w')  for i in list:  name = total\_xml[i][:-4] + '\n'  if i in trainval:  ftrainval.write(name)  if i in train:  ftrain.write(name)  else:  fval.write(name)  else:  ftest.write(name)  ftrainval.close() ftrain.close() fval.close() ftest.close() |

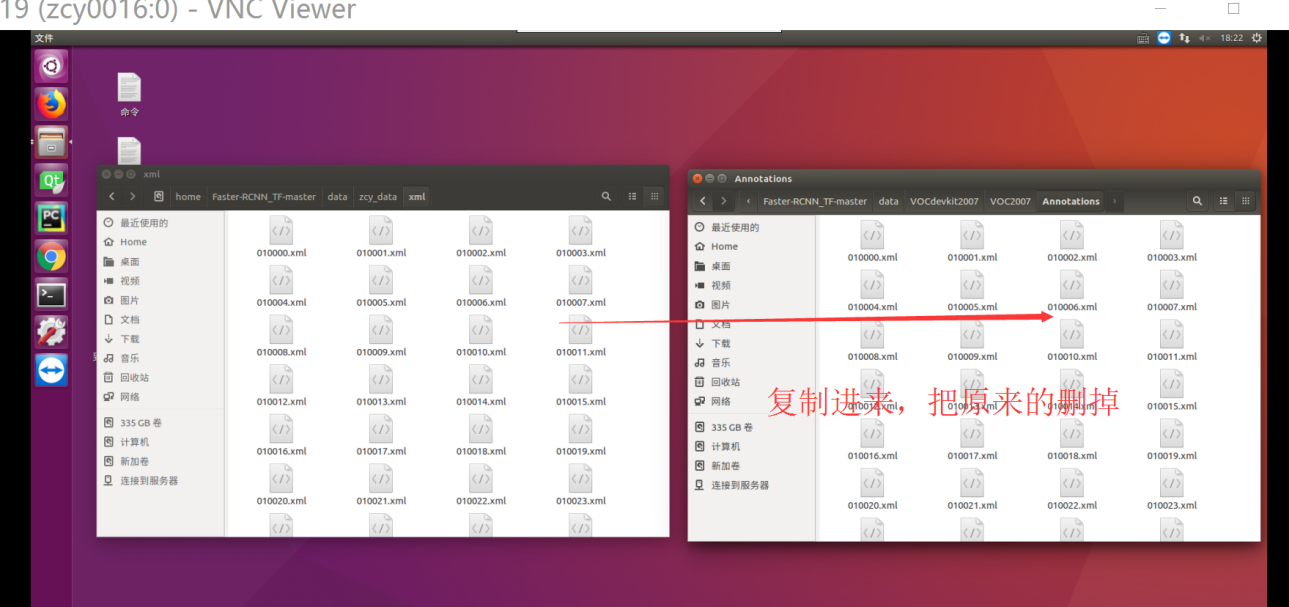
## 第四步：替换数据集

首先，在tf-faster-rcnn/lib/datasets目录下的pascal\_voc.py里第36行更改自己的类别，'\_\_background\_\_'切记不可删掉，把后面的原来的20个label换成自己的，不用更改类别数目，也没有地方可以更改。

然后把你的xml文件放置在tf-faster-rcnn/data/VOCdevkit2007/VOC2007/Annotations路径下，记得把原来的删掉；

同时把你的jpg文件放在tf-faster-rcnn/data/VOCdevkit2007/VOC2007/JPEGImages路径下，xml和jpg替换完了.

|  |
| --- |
| /home/Faster-RCNN\_TF-master/data/VOCdevkit/VOC2007/JPEGImages  /home/Faster-RCNN\_TF-master/data/VOCdevkit/VOC2007/Annotations |



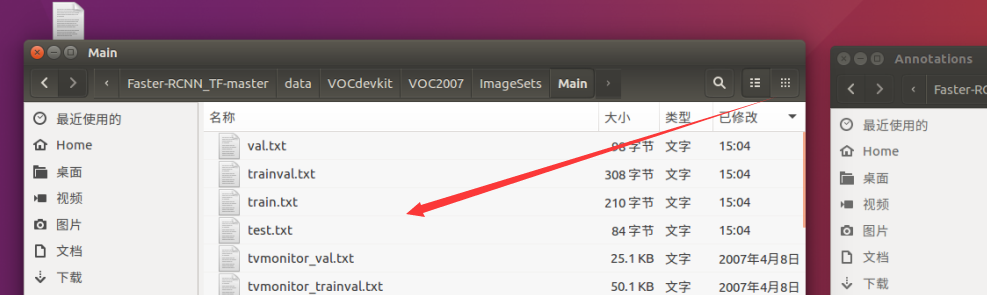
现在该txt了，把之前matlab生成是四个txt文档放在tf-faster-rcnn/data/VOCdevkit2007/VOC2007/ImageSets/Layout 和 tf-faster-rcnn/data/VOCdevkit2007/VOC2007/ImageSets/Main。

在开始训练之前，还需要把之前训练产生的模型以及cache删除掉，分别在tf-faster-rcnn/output/vgg16/voc\_2007\_trainval/default路径下和tf-faster-rcnn/data/cache路径下.

## 第五步:文件索引

生成训练和测试需要的txt文件索引，程序是根据这个索引来获取图像的.

|  |
| --- |
| # !/usr/bin/python  # -\*- coding: utf-8 -\*-  import os  import random    trainval\_percent = 0.8 #trainval占比例多少  train\_percent = 0.7 #test数据集占比例多少  xmlfilepath = 'Annotations'  txtsavepath = 'ImageSets\Main'  total\_xml = os.listdir(xmlfilepath)    num=len(total\_xml)  list=range(num)  tv=int(num\*trainval\_percent)  tr=int(tv\*train\_percent)  trainval= random.sample(list,tv)  train=random.sample(trainval,tr)    ftrainval = open('ImageSets/Main/trainval.txt', 'w')  ftest = open('ImageSets/Main/test.txt', 'w')  ftrain = open('ImageSets/Main/train.txt', 'w')  fval = open('ImageSets/Main/val.txt', 'w')    for i in list:  name=total\_xml[i][:-4]+'\n'  if i in trainval:  ftrainval.write(name)  if i in train:  ftrain.write(name)  else:  fval.write(name)  else:  ftest.write(name)    ftrainval.close()  ftrain.close()  fval.close()  ftest .close() |



## 第五步:参数修改

lib/config.py

### 迭代n次数,保存模型

默认5000次保存一次模型, 保存间隔太小时, 会导致内存爆掉.

比如: 训练10000次, 迭代此数是50时, 当训练2600次时, 产生的文件就搞达53GB. 所以应该合理运用迭代保存模型的次数.

tf.app.flags.DEFINE\_integer('snapshot\_iterations',50,"Iteration to take snapshot")

### 迭代总次数

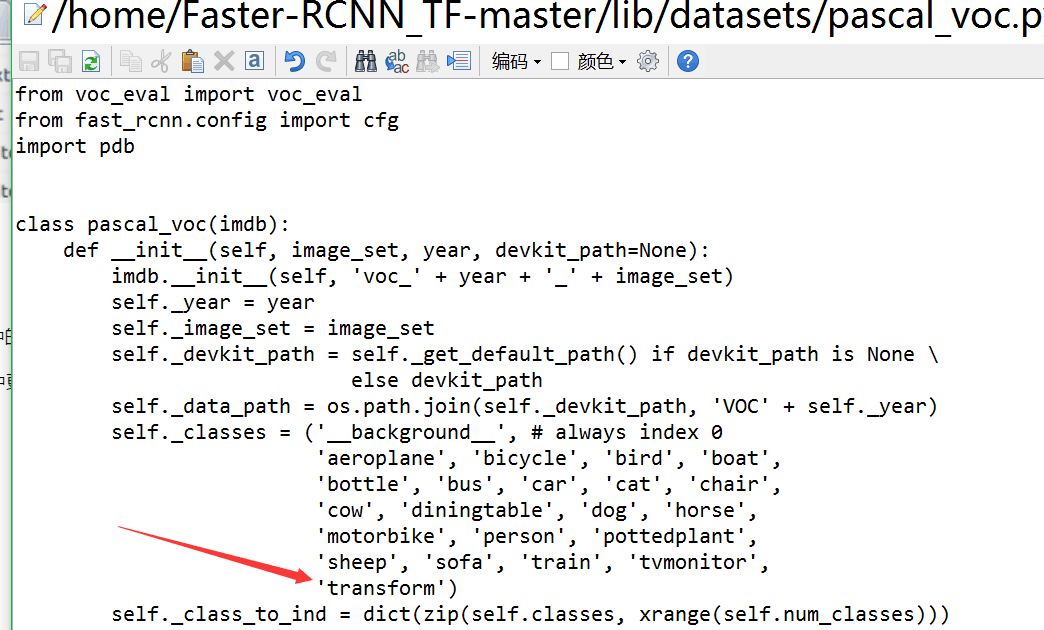
|  |
| --- |
| tf.app.flags.DEFINE\_integer('max\_iters', 20000, "Max iteration") |

### 类别

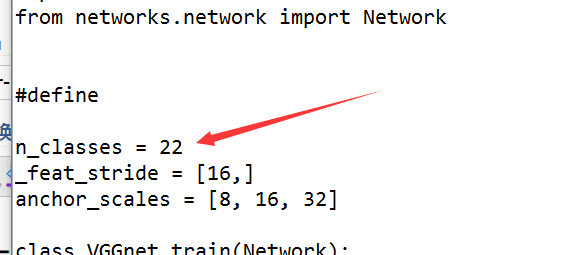
lib/pascal\_voc.py

self.\_classes = ('\_\_background\_\_', '自己的类别')

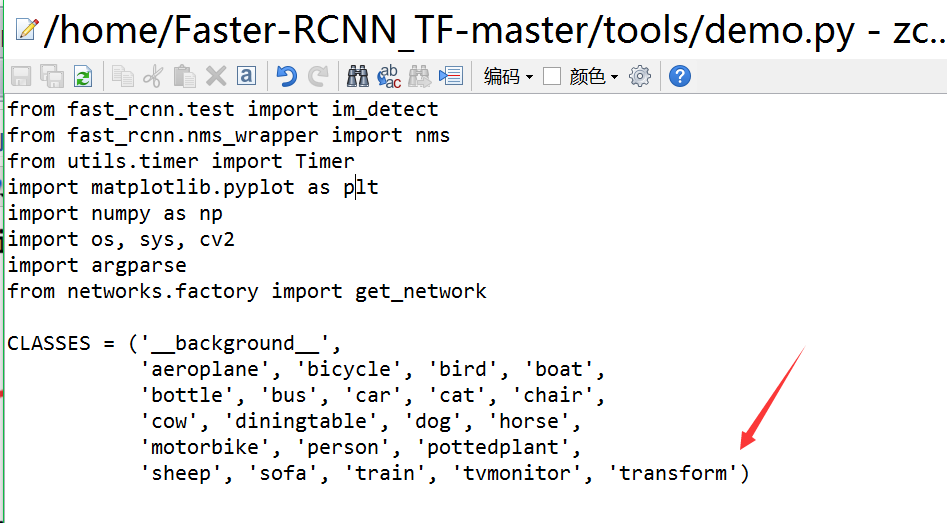
1. lib\datasets\pascal\_voc.py中更改self.\_classes中的类别，添加自己的类名字“transform”



2. lib\networks中VGGnet\_train.py和VGG\_test.py中更改n\_classes为自己的类的个数+1，这里我的是21+1=22



3. tools/demo.py中CLASSES的类别改为自己的类



### 权值衰减(weight\_decay)

lib/config/config.py

### 学习率(learning\_rate)

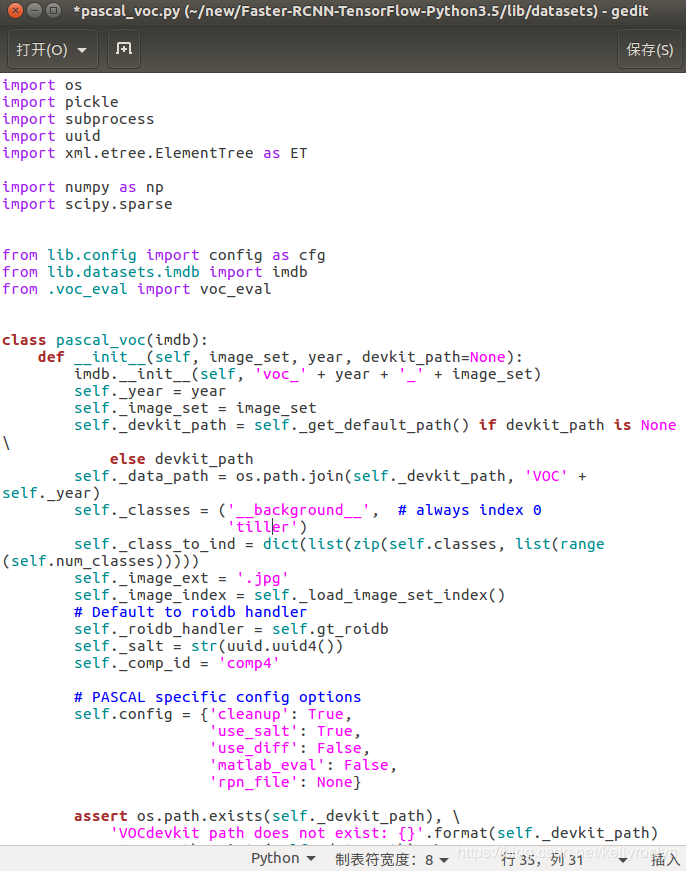
lib/config/config.py

### 批次大小(batch\_size)

lib/config/config.py

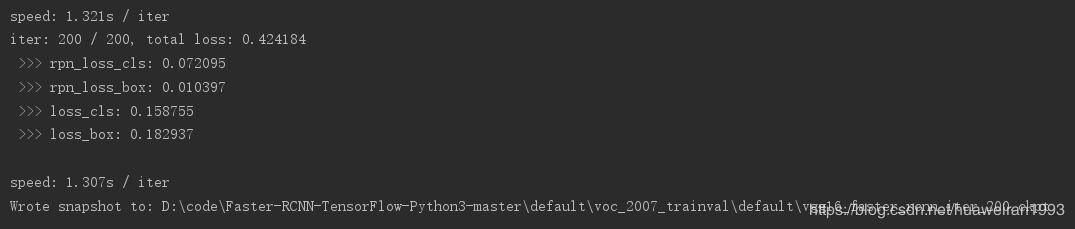
## 第六步:训练数据集

把pascal\_voc.py line 34行代码的类别改为自己的类别



## 第七步:训练结果保存位置

结果如下，训练结果被保存到了Faster-RCNN-TensorFlow-Python3-master\default\voc\_2007\_trainval\default



注意：

如果要再次进行训练，需要把Faster-RCNN-TensorFlow-Python3-master\default\voc\_2007\_trainval\default

和

Faster-RCNN-TensorFlow-Python3-master/output/vgg16/voc\_2007\_trainval/default路径下之前训练产生的模型和Faster-RCNN-TensorFlow-Python3-master/data/cache路径下的cache删掉.

## 第八步:测试结果集的质量

在demo.py中

    1. 修改类别，跟pascal\_voc.py里的类别一致

CLASSES = ('\_\_background\_\_',

'person')

2. 测试数据集改成自己的数据，并替换Faster-RCNN-TensorFlow-Python3-master\data\demo中的图像文件，图像文件的名字与修改后的文件名要一致.

im\_names = ['000001.jpg', '000002.jpg', '000003.jpg', '000004.jpg',

'000005.jpg']

3. 将

NETS = {'vgg16': ('vgg16\_faster\_rcnn\_iter\_70000.ckpt',), 'res101': ('res101\_faster\_rcnn\_iter\_110000.ckpt',)}

DATASETS = {'pascal\_voc': ('voc\_2007\_trainval',), 'pascal\_voc\_0712': ('voc\_2007\_trainval+voc\_2012\_trainval',)}

    改为

NETS = {'vgg16': ('vgg16.ckpt',), 'res101': ('res101\_faster\_rcnn\_iter\_110000.ckpt',)}

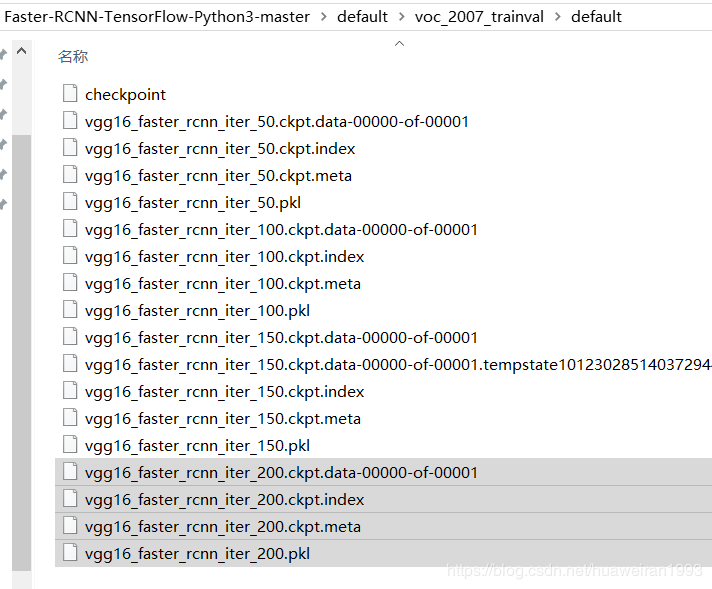
DATASETS = {

'pascal\_voc': ('voc\_2007\_trainval',),

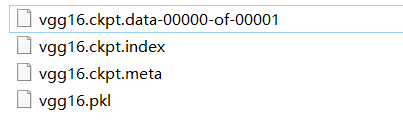
'pascal\_voc\_0712': ('voc\_2007\_trainval',)

}

4. 新建Faster-RCNN-TensorFlow2-Python3\output\vgg16\voc\_2007\_trainval\default目录，将Faster-RCNN-TensorFlow-Python3-master\default\voc\_2007\_trainval\default下的这四个文件复制到output\vgg16\voc\_2007\_trainval\default中



并改名为



5. 将104行代码中的default='res101'改为default='vgg16'

6. 运行demo.py，结果如下



## 第九步:输出mAP多类平均正确率

mAP介绍:

mean Average Precision, 即各类别AP的平均值. 也就是评价目标检测的一个度量标准.

1. 在demo.py的import下面添加两行代码

from lib.utils.timer import Timer

# 计算mAP的值

from lib.utils.test import test\_net

from lib.datasets.factory import get\_imdb

2. 在demo.py的最后添加两行代码

print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

print('Demo for data/demo/{}'.format(im\_name))

demo(sess, net, im\_name)

imdb = get\_imdb("voc\_2007\_trainval")

test\_net(sess, net, imdb, 'default')

plt.show()

    3. 打开lib/datasets/pascal\_voc.py 修改下面的函数中的filename

def \_get\_voc\_results\_file\_template(self):

# VOCdevkit/results/VOC2007/Main/<comp\_id>\_det\_test\_aeroplane.txt

# filename = self.\_get\_comp\_id() + '\_det\_' + self.\_image\_set + '\_{:s}.txt'

filename = self.\_image\_set + '\_{:s}'

path = os.path.join(

self.\_devkit\_path,

'results',

'VOC' + self.\_year,

'Main',

filename)

return path

    4. 新建\data\VOCDevkit2007\results\VOC2007\Main目录

    5. 运行demo.py，结果如下



## 第十步:AP平均正确率也就是PR曲线下面积

### 参考文献

<https://blog.csdn.net/weixin_45798949/article/details/105794168?utm_medium=distribute.pc_relevant.none-task-blog-baidujs-2>

P=检测正确/（检测正确+检测误以为正确）   R=检测正确/（检测正确+检测误以为错误）

ap的计算

根目录lib/dataset 下voc\_eval.py有这样的代码

|  |
| --- |
| def voc\_ap(rec, prec, use\_07\_metric=False):  """ ap = voc\_ap(rec, prec, [use\_07\_metric])  Compute VOC AP given precision and recall.  If use\_07\_metric is true, uses the  VOC 07 11 point method (default:False).  """  if use\_07\_metric:  # 11 point metric  ap = 0.  for t in np.arange(0., 1.1, 0.1):  if np.sum(rec >= t) == 0:  p = 0  else:  p = np.max(prec[rec >= t])  ap = ap + p / 11.  else:  # correct AP calculation  # first append sentinel values at the end  mrec = np.concatenate(([0.], rec, [1.]))  mpre = np.concatenate(([0.], prec, [0.]))    # compute the precision envelope  for i in range(mpre.size - 1, 0, -1):  mpre[i - 1] = np.maximum(mpre[i - 1], mpre[i])    # to calculate area under PR curve, look for points  # where X axis (recall) changes value  i = np.where(mrec[1:] != mrec[:-1])[0]    # and sum (\Delta recall) \* prec  ap = np.sum((mrec[i + 1] - mrec[i]) \* mpre[i + 1])  return ap |

意思:

如果你的数据集是voc2007，ap就取最大的精确率，11点取平均循环加起来。否则，比如数据集voc2012那么就使用precision和recall积分求面积得到ap。

## 第十一步:**precision精确率,准确率,查准率**

precision准确率说明:

假如, 有一个池塘里有100条鱼, 其中草鱼10条. 使用分类系统结果100条鱼里面没有草鱼, 所以准确率为90%.

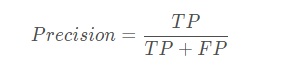
混淆矩阵

　　True Positive(真正，TP)：将正类预测为正类数

　　True Negative(真负，TN)：将负类预测为负类数

　　False Positive(假正，FP)：将负类预测为正类数误报 (Type I error)

　　False Negative(假负，FN)：将正类预测为负类数→漏报 (Type II error)



## 第十二步:recall召回率,查全率

recall查全率说明:

假如, 有一个池塘里有100条鱼, 其中草鱼10条. 使用分类系统结果100条鱼里面没有草鱼, 所以查全率为0%, 因为没有找到草鱼.

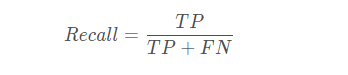
混淆矩阵

　　True Positive(真正，TP)：将正类预测为正类数

　　True Negative(真负，TN)：将负类预测为负类数

　　False Positive(假正，FP)：将负类预测为正类数误报 (Type I error)

　　False Negative(假负，FN)：将正类预测为负类数→漏报 (Type II error)



将检查的阈值提高, 但会降低recall查全率.

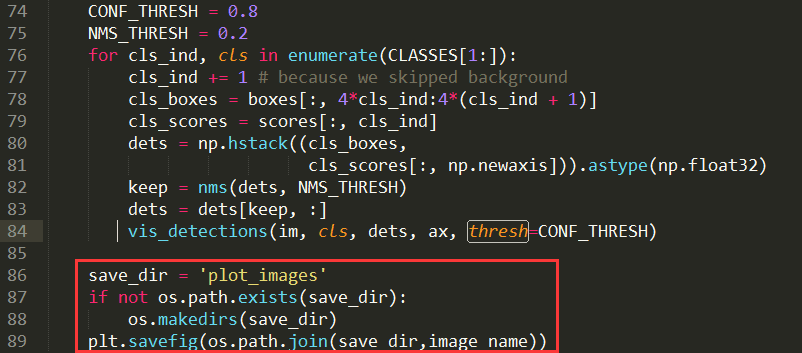
## 第十步:画PR曲线

## 第十步:计算程序运行时间

|  |
| --- |
| import time import datetime # start running time starttime = datetime.datetime.now()   time.sleep(20) # end running time endtime = datetime.datetime.now() print("程序运行时间:", endtime - starttime) |

## 第十一步:保存标记的图片

在demo.py里添加我们要保存图片的路径



## 第十二步:绘制特征图

可视化RPN输入的特征图. 使用的版本是faster-RCNN-TensorFlow2-Python3, 此版本是根据Faster-RCNN-TensorFlow-python3改造而成.

主要修改lib/network.py文件, 此文件是生成网络的结构的.

1.在network.py中的初始化\_\_init\_\_()添加变量, self.\_bb = {}, 此变量用来保存RPN层特征图的内容.

2.本案例使用的是vgg16, 卷积的结构在lib/vgg16.py内, 通过赋值将获取到的rpg 值赋值给self.\_bb, 代码如下:

|  |
| --- |
| # 用来保存特征图的内容, 将tensor的数值提取出来 self.\_bb = rpn |

3.绘制特征图. 在network.py文件中, 添加绘制代码, 将self.\_bb中的参数绘制成特征图. 代码如下:

|  |
| --- |
| # 绘制特征图 draw\_feature\_map = sess.run(self.\_bb, feed\_dict=feed\_dict) print(draw\_feature\_map.shape[0]) print(draw\_feature\_map.shape[1]) print(draw\_feature\_map.shape[2]) print(draw\_feature\_map.shape[3])  for i in range(10):  show\_img = draw\_feature\_map[:, :, :, i]  # print type(show\_img)  show\_img.shape = [draw\_feature\_map.shape[1], draw\_feature\_map.shape[2]]  plt.imshow(show\_img, cmap='jet')  plt.axis('off')  plt.show() |

实现思想: 上面绘图程序的print[0]到[3]是我打印出来前几张特征图看了一下，后面写了个for循环，将前10个特征图放到同一个画布上显示

特征图的数量: 实际上rpn输入的特征图数量如果是resnet101的话应有有512张.

## 第十三步:多幅特征图融合为一幅特征图

1.新建列表feature\_map\_combination = []用来保存多副特征的信息.

2.在绘制多幅特征图的基础上, 进行列表的添加. 红色字体为添加代码.

feature\_map\_combination.append(show\_img)

源码如下:

|  |
| --- |
| for i in range(10):  show\_img = draw\_feature\_map[:, :, :, i]  feature\_map\_combination.append(show\_img)  # print type(show\_img)  show\_img.shape = [draw\_feature\_map.shape[1], draw\_feature\_map.shape[2]]  plt.imshow(show\_img, cmap='jet')  plt.axis('off')  plt.show() |

3.绘制特征图

|  |
| --- |
| # 按照特征图 进行 叠加代码 feature\_map\_sum = sum(one for one in feature\_map\_combination) plt.title("将多个特征图融合成一幅特征图") plt.imshow(feature\_map\_sum) plt.savefig('././data/save\_feature\_map/feature\_map\_sum1.jpg') # 保存图像到本地 plt.show() |

## 第十四步:ckpt转换为PB格式

https://blog.csdn.net/u014311125/article/details/103522886

## 图片格式转换

TensorFlow验证图片的格式得是jpg格式，只通过修改后缀名的方式，并不能够修改图片的实际格式。cv2.imread()图片图片时，会因为实际格式与后缀名格式不同，进而导致报AttributeError: 'NoneType' object has no attribute 'astype'错误。因此，需要将验证的图片格式修改为jpg的格式。

|  |
| --- |
| # 修改验证图片的格式 img = Image.open(im\_file).convert('RGB') img.save(im\_file) |

除了格式上得错误，路径中含有中文，使用单”\”反斜杠也会报AttributeError: 'NoneType' object has no attribute 'astype'错误。

## 训练时生成PB文件

python中训练完成，模型保存使用如下api保存：

|  |
| --- |
| # 保存二进制模型  output\_graph\_def = tf.graph\_util.convert\_variables\_to\_constants(sess, sess.graph\_def, output\_node\_names=['y\_conv\_add'])  with tf.gfile.FastGFile('/logs/mnist.pb', mode='wb') as f:  f.write(output\_graph\_def.SerializeToString()) |

保存为二进制pb文件，主要的点是output\_node\_names数组，该数据的名称表示需要保存的tensorflow tensor名。既是在python中定义模型时指定的计算操作的name。填写什么就保存到什么节点。在cnn模型中，通常是分类输出的名称。

例如模型定义时代码为：

|  |
| --- |
| y\_conv = tf.add(tf.matmul(h\_fc1\_drop, W\_fc2), b\_fc2, name='y\_conv\_add') # cnn输出层，名称y\_conv\_add  # 训练和评价模型  softmax = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y\_conv) |

## 相似值与类型

在验证图片的demo.py脚本中的vis\_detections()，在以下循环中添加输出score和class\_name。

for i in inds:  
 bbox = dets[i, :4]  
 score = dets[i, -1]

|  |
| --- |
| # inds [0] # dets [[ 0. 7.872568 63. 55.951324 0.7933805]] # i 0 # score 0.7933805 for i in inds:  bbox = dets[i, :4]  score = dets[i, -1]  # print("========inds========\n", inds)  # print("=====dets======\n", dets)  # print("=========i=========\n", i)  # print("========score=========\n", score)   # 输出检测出来的类型  print("class\_name:{:s}".format(class\_name))  # 输出相似度  print("Mark:{}".format(score)) |

## 输出字符串，内容是json格式

print('{"class\_name":"%s"}' % class\_name)

输出字符串内容是json格式的不能使用.format()实现，字符串内容会因为存在两个{}进而导致报错，所以应该使用%s形式的占位符实现功能。Java调用时可以先判断是否是json格式，然后再进行取值，这样可以去除掉TensorFlow自带的输出的信息，只保留需要的内容。

以下是Java代码，用来判断字符串是否是json格式。使用的是三方jar，fastjson-1.2.68版本。

|  |
| --- |
| **boolean** isjson(String string){  **try** {  JSONObject jsonStr= JSONObject.*parseObject*(string);  **if** (jsonStr.isEmpty()){  **return false**;  }**else** {  **return true**;  }  } **catch** (Exception e) {  **return false**;  } } |

# Java调用py脚本

通过Runtime.getRuntime()开启进程来执行脚本文件，因为Python有时需要用到第三方库，比如requests，而Jython不支持。所以本地安装Python环境并且安装第三库再用Java调用是最好的方法。

## python代码

因为要通过使用Java来调用py脚本，Java通过Runtime.getRuntime().exec()方法调用服务器命令脚本实现py脚本的调用，所以py脚本接受参数时，要通过sys.argv[]来接受参数。

|  |
| --- |
| import sys    def hello(name,age):  print('name:'+name)  print('age:'+age)    if \_\_name\_\_ == '\_\_main\_\_':  hello(sys.argv[1], sys.argv[2]) |

注意：此方法是通过sys进行传参，如果方法使用的方法中使用了argparse.ArgumentParser()进

进行参数的获取，切勿再使用sys进行参数传递，否则会报错。

parse\_args()方法是专门进行命令行参数接收的方法

|  |
| --- |
| parser = argparse.ArgumentParser(description='Tensorflow Faster R-CNN demo') parser.add\_argument('--net', dest='demo\_net', help='Network to use [vgg16 res101]',  choices=NETS.keys(), default='vgg16') parser.add\_argument('--dataset', dest='dataset', help='Trained dataset [pascal\_voc pascal\_voc\_0712]',  choices=DATASETS.keys(), default='pascal\_voc\_0712') parser.add\_argument('--im\_file', dest='im\_file', type=str, help='this picture url') args = parser.parse\_args() while len(sys.argv) > 1:  sys.argv.pop() return args |

参数介绍

--xx：是命令行指定参数提交的标识

dest：脚本用来接收命令行提交数据的变量

type：变量的类型

help：提示信息

脚本获取参数

args.dest

eg：

im\_file = args.im\_file

案例代码中的

|  |
| --- |
| while len(sys.argv) > 1:  sys.argv.pop() |

是解决absl.flags.\_exceptions.UnrecognizedFlagError: Unknown command line flag 'in'错误问题.



## Java代码

此案例是单线程案例Python脚本返回的信息过多时会导致卡死，建议使用多线程，如案例二。

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;    public class HelloPython {  public static void main(String[] args) {  // 传参案例  // String[] arguments = new String[] {"python", "E://workspace/hello.py","lei","23"};  String[] arguments = new String[] {"python", "E://workspace/hello.py"};  try {  Process process = Runtime.getRuntime().exec(arguments);  BufferedReader in = new BufferedReader(new InputStreamReader(process.getInputStream(),"GBK"));  String line = null;  while ((line = in.readLine()) != null) {  System.out.println(line);  }  in.close();  //java代码中的process.waitFor()返回值为0表示我们调用python脚本成功，  //返回值为1表示调用python脚本失败，这和我们通常意义上见到的0与1定义正好相反  int re = process.waitFor();  System.out.println(re);  } catch (Exception e) {  e.printStackTrace();  }  }  } |

### 案例二

多线程类，实现了脚本返回的数据读取以及存储到txt文件中。

|  |
| --- |
| **import** java.io.\*; **public class** StreamReaderThread **implements** Runnable {  */\*  \* python的输出流  \*/* **private** InputStream **inputStream**;  */\*  \* 输出信息保存的文件名称  \*/* **private** String **logName**;   **public** StreamReaderThread(InputStream inputStream, String logName){  **this**.**inputStream** = inputStream;  **this**.**logName** = logName;  }  **public void** run()  {  BufferedReader in = **null**;  FileWriter fwriter = **null**;  **try** {  in = **new** BufferedReader(**new** InputStreamReader(**this**.**inputStream**,**"GBK"**));  fwriter = **new** FileWriter(**logName**, **true**);  String line = **null**;  **while** ( (line = in.readLine()) != **null**)  {  fwriter.write(line);  System.***out***.println(line);  }   } **catch** (IOException e){  e.printStackTrace();  } **finally** {  **try** {  **inputStream**.close();  fwriter.flush();  fwriter.close();  in.close();  } **catch** (IOException e) {  e.printStackTrace();  }   }  } } |

调用Python脚本类

|  |
| --- |
| **import** java.io.BufferedReader; **import** java.io.IOException; **import** java.io.InputStreamReader; **public class** JavaCallsPython {   **public static void** exePython(){  System.***out***.println(**"正在执行python程序"**);  String[] arguments = **new** String[] {**"python"**, **"G:\\PyCharm\\PyCharmSpaceWork\\Faster-RCNN-TensorFlow2-Python3\\JavaCallsPython.py"**,  **"--im\_file"**, **"E:\\ChromeCoreDownloads\\2.jpg"**};  Process proc = **null**;  **try** {  proc = Runtime.*getRuntime*().exec(arguments);*// 执行py文件* Thread thread1 = **new** Thread(**new** StreamReaderThread(proc.getInputStream(),**"normal.txt"**));  Thread thread2 = **new** Thread(**new** StreamReaderThread(proc.getErrorStream(),**"error.txt"**));  thread2.start();  thread1.start();*//必须后执行，否则正确消息容易接收不到* proc.waitFor();  Thread.*sleep*(1000);*//等待后台线程读写完毕* System.***out***.println(**"python program done!!!"**);  } **catch** (IOException e) {  e.printStackTrace();  } **catch** (InterruptedException e) {  e.printStackTrace();  } **finally** {  **try** {  proc.getErrorStream().close();  proc.getInputStream().close();  } **catch** (IOException e) {  e.printStackTrace();  }  proc.destroy();  }  }    **public static void** main(String[] args) {  *exePython*();   } } |

Did you download the proper networks from our server and place them properly? 错误解决方法

[修改demo.py](http://xn--demo-9z2h93o.py/)文件，修改第120行的代码

|  |
| --- |
| tfmodel = os.path.join('output', demonet, DATASETS[dataset][0], 'default',  NETS[demonet][0]) |

将’output’路径全部写上去，比如我的output在电脑的位置是G:\PyCharm\PyCharmSpaceWork\Faster-RCNN-TensorFlow2-Python3则改成

|  |
| --- |
| tfmodel = os.path.join(r'G:\PyCharm\PyCharmSpaceWork\Faster-RCNN-TensorFlow2-Python3\output',  demonet, DATASETS[dataset][0], 'default', NETS[demonet][0]) |

# Java web调用TensorFlow图片分类模型

JDK安装

java.lang.IllegalArgumentException: Invalid characters in hostname错误

环境

Java 8

IDEA 2017.2.6

apache-maven-3.6.3

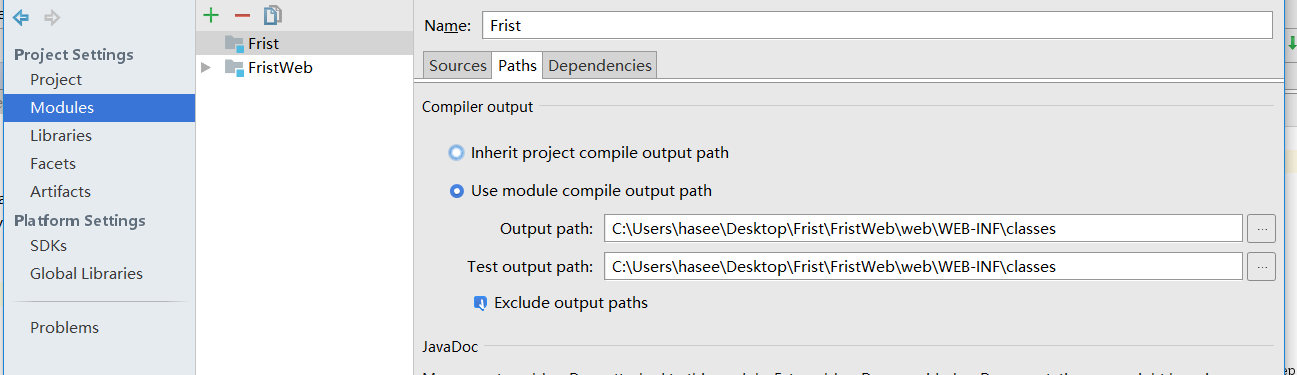
## IDEA添加lib文件夹

IDEA创建web项目时, 不会自己创建lib文件, 需要手工添加文件夹, 并且指定目录的位置.

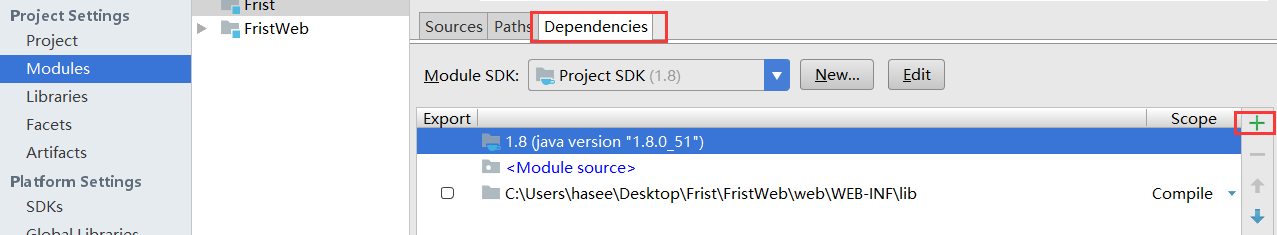
以下为指定lib文件的位置, 如图所示:

1.点击File>Project Structure(或者快捷键Ctrl+AIT+shift+s)

2.在如图页面点击modules



3.然后选择当前窗口的Dependencies，点击+号选中你刚刚创建的lib文件夹,选择后会弹出一个框框，再选择Jar Directory>OK,最后OK



## 读取词典文件

这个词典文件wordIndexMap.txt，就是上一篇对应训练模型之前生成的词典文件。每行一个词和词的编号。

|  |
| --- |
| // 从文件读取词典文件存入Map  private static Map<String, Integer> readVocabFromFile(String pathname) throws IOException{  Map<String, Integer> wordMap = new HashMap<String, Integer>();  File filename = new File(pathname);  InputStreamReader reader = new InputStreamReader(new FileInputStream(filename));  BufferedReader br = new BufferedReader(reader);  String line = "";  line = br.readLine();  String[] lineArray;  while(line != null){  lineArray = line.split(" ");  wordMap.put(lineArray[0], Integer.parseInt(lineArray[1]));  line = br.readLine();  }  return wordMap;  } |

## 加载Tensorflow模型文件

这里加载上一篇中训练完成保存的模型文件lstm\_attention.pb。

|  |
| --- |
| // 读取tensorflow二进制的模型文件  private static byte[] readAllBytes(String pathname) throws IOException{  File filename = new File(pathname);  BufferedInputStream in = new BufferedInputStream(new FileInputStream(filename));  ByteArrayOutputStream out = new ByteArrayOutputStream(1024);  byte[] temp = new byte[1024];  int size = 0;  while((size = in.read(temp)) != -1){  out.write(temp, 0, size);  }  in.close();  byte[] content = out.toByteArray();  return content;  } |

## 读取预测数据

预测可以是一条数据，也可以是一个batch的数据。

|  |
| --- |
| // 读取分词后的一个样本，并建立索引  public static int[][] getInputFromSentence(String sentence, Map<String, Integer> wordIndexMap) {  int[][] indexArray = new int[1][MAX\_SEQUENCE\_LENGTH];  String[] words = sentence.split(" ");  for(int i=0; i<words.length; i++){  if(wordIndexMap.containsKey(words[i])){  indexArray[0][i] = wordIndexMap.get(words[i]);  }  }  return indexArray;  }  // 对一个batch的样本建立索引  public static int[][] getInputFromSentenceBatch(String[] sentences, Map<String, Integer> wordIndexMap){  int[][] indexArray = new int[sentences.length][MAX\_SEQUENCE\_LENGTH];  for(int i=0; i<sentences.length; i++){  String[] words = sentences[i].split(" ");  for(int j=0; j<words.length; j++){  if(wordIndexMap.containsKey(words[j])){  indexArray[i][j] = wordIndexMap.get(words[j]);  }  }  }  return indexArray;  } |

## 预测结果

需要新建Tensorflow的Session会话，读取训练好的模型计算图和参数

|  |
| --- |
| import java.io.\*;  import java.util.ArrayList;  import java.util.HashMap;  import java.util.Map;  import org.tensorflow.Graph;  import org.tensorflow.Session;  import org.tensorflow.Tensor;  public class TensorflowDemo {  private static String TensorFlow\_MODEL\_PATH = "lstm\_attention.pb";  private static String WORD\_INDEX\_PATH = "wordIndexMap.txt";  private static int MAX\_SEQUENCE\_LENGTH = 60;  private static int CLASS\_NUM = 2;  public static void main(String[] args) throws IOException{  // 构建词典Map  Map<String, Integer> wordsMap = readVocabFromFile(WORD\_INDEX\_PATH);  System.out.println("vocabulary size:"+wordsMap.size());  // 加载Tensorflow训练好的模型  byte[] graphDef = readAllBytes(TensorFlow\_MODEL\_PATH);  Graph graph = new Graph();  graph.importGraphDef(graphDef);  Session session = new Session(graph);  String test\_sentence = "再也 不用 愁 看不起 病 了 ， 老祖宗 留下 此表 ！";  System.out.println("sentence: "+test\_sentence);  // 输入模型的测试语句  int[][] sentenceBuf = getInputFromSentence(test\_sentence, wordsMap);  int[] sentLength = {sentenceBuf[0].length};    Tensor inputTensor = Tensor.create(sentenceBuf);  Tensor lengthTensor = Tensor.create(sentLength);  // 输入数据，得到预测结果  Tensor result = session.runner()  .feed("Input\_Layer/input\_x:0", inputTensor)  .feed("Input\_Layer/length:0", lengthTensor)  .fetch("Accuracy/score:0")  .run().get(0);  long[] rshape = result.shape();  int batchSize = (int) rshape[0];  // int nlabels = (int) rshape[1];  float[][] resultArray = new float[batchSize][CLASS\_NUM];  result.copyTo(resultArray); // 输出结果Tensor复制到二维数组中  System.out.println(resultArray[0][0]+" "+resultArray[0][1]);  } |

### 注意预测结果时同样要保持模型输入输出格式名称一致。预测的输入输出要与模型最初构建时保持一致。 模型构建时是下面这种写法： IMG_256 IMG_257 java预测时是这种写法： IMG_258

|  |
| --- |
| public static void main(String[] args) throws Exception {  // TODO Auto-generated method stub  try (Graph graph = new Graph()) {  byte[] graphBytes = IOUtils.toByteArray(new FileInputStream("model/first.pb"));  graph.importGraphDef(graphBytes);    try (Session session = new Session(graph)) {  Tensor<?> out = session.runner().feed("X", Tensor.create(2.0f)).fetch("results").run().get(0);  float[] r = new float[1];  out.copyTo(r);  System.out.println(r[0]);  }  }  } |

### 案例二

模型在java中使用需要关心模型输入tensor和输出tensor名，所以定义模型时，所有的输入tensor最好指定名称，如输入x和dropout名。

|  |
| --- |
| public static void main(String[] args) {  String labels = "17,16,7,8,3,15,4,14,2,5,12,18,9,10,1,11,13,6";    TensorFlowInferenceInterface tfi = new TensorFlowInferenceInterface("D:/tf\_mode/output\_graph.pb","imageType");  final Operation operation = tfi.graphOperation("y\_conv\_add");  Output output = operation.output(0);  Shape shape = output.shape();  final int numClasses = (int) shape.size(1);  float[] floatValues = getImagePixel("D:/tf\_mode/ci/ci/333.jpg"); //将图片处理为输入对应张量格式    // 输入图片  tfi.feed("x\_input", floatValues, 1, 2048); //将数据复制给输入张量x\_input即为模型定义时的x名称  tfi.run(new String[] { "y\_conv\_add" }, false);//输出张量  float[] outPuts = new float[numClasses];//结果分类  tfi.fetch("y\_conv\_add", outPuts);//接收结果 outPuts保存的即为预测结果对应的概率，最大的一个通常为本次预测结果} |

### 案例三

|  |
| --- |
| private static Session loadSession() {  SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");  Graph graph = new Graph(); //创建图结构  InputStream is = getStreamFromPb("car\_model.pb"); //加载本地 pb 文件到内存  byte[] graphBytes = new byte[0];  try {  graphBytes = IOUtils.toByteArray(is);  } catch (Exception e) {  e.printStackTrace();  }  graph.importGraphDef(graphBytes); //内存数据 ——> 图结构  Session session = new Session(graph); //通过图结构初始化会话  return session;    }  private static String faceEmbedding(Session session, String imagePath) {  float[][] embeddingsRes = new float[1][128];    try {  float[][][] rgbImage = readImage(imagePath);  float[][][] rgbWhitened = whiten(rgbImage);  float[][][][] rgbFloat = new float[1][160][160][3];  rgbFloat[0] = rgbWhitened;    Tensor<Float> imageTensor = Tensors.create(rgbFloat); //输入 Tensor  Tensor phaseTensor = Tensor.create(new Boolean(Boolean.FALSE)); //输入 Tensor  Session.Runner result = session.runner().feed("input", imageTensor).feed("phase\_train", phaseTensor);    Tensor embeddings = result.fetch("embeddings").run().get(0); //执行图，输出 Tensor  System.out.println("embeddings.toString(): " + embeddings.toString());  embeddings.copyTo(embeddingsRes);    } catch (Exception e) {  e.printStackTrace();  }    JSONObject json = new JSONObject();  json.put(image\_path, embeddingsRes[0]);    return json.toString();  } |

# 源码改造

## 第十二步:返回验证后列表

在demo.py的main方法中添加全局变量verificationResult列表类型, 在main中声明的变量默认是全局变量. 在vis\_detections方法中添加列表追加.append(). 因为append()只是修改元素, 并非添加元素, 所以无需使用global, global只有当是赋值时, 放开使用.

## 第十三步:验证集自动加载

demo.py是Faster-RCNN-tensorflow-Python3工程的数据验证方法, 测试图片放在工程data/demo文件夹下, 在demo.py中的main方法中通过im\_names列表, 挨个声明data/demo文件夹下所有照片的名字, 此方法费时费力, 故有此功能解决此问题.

在lib/utils下创建filePath.py, 此文件用来处理绝对路径, 返回绝对路径的文件目录.

|  |
| --- |
| # -\*- coding: utf-8 -\*-  import os class filePath:  *'''文件相关操作'''* def file\_name(self, file\_dir):  *'''获取绝对路径下的文件名  file\_dir:绝对路径 G:\PyCharm\PyCharmSpaceWork\Faster-RCNN-TensorFlow-Python3\data\demo  return root当前目录路径 dirs当前路径下所有子目录 files当前路径下所有非目录子文件  '''* for root, dirs, files in os.walk(file\_dir):  # print("=======当前目录路径=========\n",root) # 当前目录路径  # print("=======当前路径下所有子目录=========\n",dirs) # 当前路径下所有子目录  # print("=======当前路径下所有非目录子文件=========\n",files) # 当前路径下所有非目录子文件  return root, dirs, files |

使用方式, demo.py文件main方法下, 修改im\_names列表, 将原始的列表修改为

im\_names = []

|  |
| --- |
| filePath = filePath() root, dirs, files = filePath.file\_name(r"G:\PyCharm\PyCharmSpaceWork\Faster-RCNN-TensorFlow-Python3\data\demo") im\_names = files |

## 模型优化

在调参数之前，首先要知道需要调的参数是什么.可以参考下面链接，学习率、权重初始化方法，加减网络层数，增加正则化都是常见的可以调节的参数。

***[http://blog.csdn.net/qq\_20259](http://www.voidcn.com/article/javascript:void()" \t "http://www.voidcn.com/article/_blank)***

1、 首先是学习率

这是可能在调参数过程中第一个应该调的参数，调整学习率可以调大或者调小。

上面网络的初始的学习率为0.1，这对很多数据集的分类来说已经够了，通常来说也不应该有过小的初始化学习率，这通常会导致过拟合，但并不总是这样，何凯明大神一次接受采访时说过，在有些情况下即使学习率达到0.000001，也不会过小。也不应该有过大的学习率，这会导致结果在极值点两边跳动。

分析训练结果，基本可以知道我们的学习率过大了。逐渐的降低学习率并逐步对照，观察训练结果的变化，发现预测的总体精度确实在上升，在学习率达到0.005时测试集整体平均精度达到了最高，为75%左右，再调小会出现测试集精度下降的情况，且训练速度变慢。

这一步的训练结果在测试集上面出现了令人不愉快的表现，召回率非常低，大部分预测都偏向了痣，即恶黑的测试集大部分都预测为了痣，这显然是不对的。

在以上工作的基础上，继续采用一些常见的优化模型的方法，如****修改常规的梯度下降优化器****， ****尝试带有动量的梯度下降优化器****和****学习率下降的优化器****，****尝试采用不同的激活函数****，****尝试使用不同的drop率****等常用的优化训练过程，再训练和收敛速度上，最终训练结果再测试集上的表现都有少量的提升，但始终不能解决预测明显不均衡的问题，这时候可以考虑数据是不是出了什么问题

2、 解决数据的问题

　　首先，在出现上面的结果后，我查了一些资料，对样本不均衡有了一些新的认识。考虑当前训练出现了样本不均衡，导致训练出现一些问题．参考文章  
***[在分类中如何处理训练集中不平衡问题](http://www.voidcn.com/article/p-nrlspmzz-ue.html)***  
***[深度 | 解决真实世界问题：如何在不平衡类上使用机器学习？](http://www.voidcn.com/article/javascript:void()" \t "http://www.voidcn.com/article/_blank)***

一般推荐从增加数据来源开始，但在此例中，因为数据集来源于公开数据集，并且含有恶黑和痣的分类数据的公开数据集几乎不存在，已不存在增加数据集的可能性。

这时我们考虑使用重采样和降采样

1> 首先降采样，即把痣的训练集减少到和恶黑一样，调整了痣数据的训练集和测试集的比例。继续训练，得到的结果在测试集上中，对恶黑的测试精度得到了大幅提升，但对于负样本痣的精度确下降到合格线以下

2> 然后再采用重采样的方式，将恶黑的图片进行复制，将其扩展到和痣的数量相等。继续训练，得到的结果正好和降采样相反，在测试集上，恶黑精度不高，但总体的精度确实比之前高了，这样的记过同样也是我们不能接受的

考虑上面出现的两个极端情况，就是训练中经常说的欠拟合和过拟合，减少痣的训练集，导致对痣的训练出现了欠拟合，对恶黑重采样，导致对恶黑的训练出现了过拟合，那么是否可以通过一定方式生成一些数据呢，考虑原图是256*X的尺寸，考虑采用剪裁的方式而不是通过resize的方式去获取图片，于是我对图片进行预处理，对每一幅图片从左上角开始，水平方向每隔１个像素进行一次224*高度的裁剪，并且对高度垂直方向进行了zero padding，然后对生成的图片进行了水平和垂直方向的随机翻转，生成了新的训练数据集和测试数据集（同样保持训练集中正样本和负样本数量基本相同），继续训练，情况并没有多少改善，提升也有限，并且依然是痣的准确率明显高于恶黑的准确率．

　　这个时候我不对数据集产生了怀疑，是不是因为痣的数据集中存在一些过大的干扰呢，于是我将痣的数据集换成脂溢性角化病的数据集，isic数据集中也含有1000多例的脂溢性角化病的数据，比恶黑的数据略少，采用上一次的网络进行训练，发现再两个分类上的准确率都可以达到９８％以上，于是我再返回对痣的数据集进行了审视，发现其中有一半以上的数据中包含不知是干什么用的有色圆圈，我只好再训练集中把这部分去除掉，用剩下的数据进行训练，把带有有色圆圈的数据先行搁置，对剩余的数据进行训练（恶黑和痣），发现结果有了明显的改善， 在恶黑数据集上的召回率可以达到８５％左右，准确率达到８９％左右，但实际上我们并没有达到原论文中的对ｒｇｂ空间中预测准确率９３％的结果，但这个时候训练集的精度已经达到了９９％，到了无法再继续训练的状况，继续训练就必须考虑提升模型泛化能力的方法，调整drop率从0.8到0.5，没有明显提高，就考虑继续从训练集下手，因为测试集表现低于训练集表现，这种现象就是过拟合的现象，增强泛化能力就必须减轻过拟合的影响，突然想到上一步中的对原图像每隔１像素进行裁剪获取训练集和测试机的方法是不是和之前的重采样一样，出现了很大程度的过拟合呢，虽然从一张图片中新生成的训练集都不相同，但他们都是从一张原图中裁剪出来的，对生成的训练图像进行训练是不是相当于对原图进行了多次训练呢，为了减轻这个情况，我调整了生成图片的间隔，即把每隔１像素裁剪，变成每隔５个像素再去裁剪一次，这样子，最终生成的训练数据和测试数据必然会变少，但可以做此尝试，实践证明，这样的训练确实可以减轻过拟合，再恶黑的表现召回率可以达到８８％左右，准确率达到９４．５％左右，考虑到我们的测试集是滑动生成的，预测时是根据原测试集生成的数据预测，但实际要进行的预测则是，对原图进行滑动预测，统计平均值得出预测结果的做法，那么可以想想最终的预测效果必然会更好（没有验证过，但至少结果不会差与当前结果，因为准确率和召回率都超过了５０％）

### AP平均正确率也就是PR曲线下的面积

源代码中默认的计算AP时的门限为0.5，可将这一数值降低，从而提高AP的结果。

操作步骤:

1、打开/lib/datasets/pascal\_voc.py

2、修改\_do\_python\_eval()函数

找到代码位置如下:

filename = self.\_get\_voc\_results\_file\_template().format(cls)

rec, prec, ap = voc\_eval(

filename, annopath, imagesetfile, cls, cachedir, ovthresh=0.5,

use\_07\_metric=use\_07\_metric, use\_diff=self.config['use\_diff'])

aps += [ap]

将filename, annopath, imagesetfile, cls, cachedir, ovthresh=0.5,

改为filename, annopath, imagesetfile, cls, cachedir, ovthresh=0,

# 测试

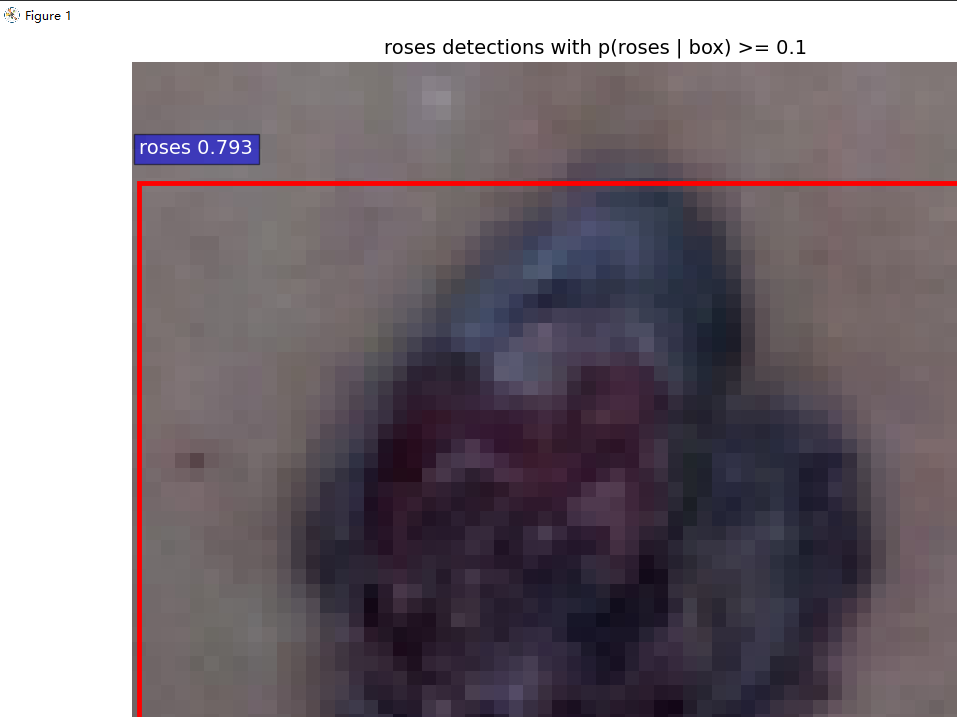
电脑配置

Intel(R) Core(TM) i3-4150 CPU @ 3.50GHz 3.50 GHz

系统类型: 64位操作系统, 基于x64的处理器

一张图片 输出mAP

用时0:22:56.670533



不输出mAP值,23张图片运行时常为0:01:00.717650

测试:

13张基底细胞癌图片

3张皮肤图片

7张胎记图片

测试结果:共找出20张, 胎记图片全部被识别为基底细胞癌图片.

## 第三次测试

没有设置保存结果的次数,导致内存爆掉.

默认5000次保存一次模型, 保存间隔太小时, 会导致内存爆掉.

比如: 训练10000次, 迭代次数是50时, 当训练2600次时, 产生的文件就搞达53GB. 所以应该合理运用迭代保存模型的次数.

## 第四次测试

迭代次数设置到5000次. 遍历次数1万.

训练用时19-20个小时.

测试样本改为27张图片, 其中13张基底细胞癌编号是1-13, 3张皮肤图片编号是14-16, 7张胎记图片编号是17-23, 四张皮肤受损图片编号是24-27.

运行时长:0:01:20.835294

实验结果, 找出23张符合基底细胞癌的图片.

'000001.jpg','000002.jpg', '000003.jpg', '000004.jpg', '000005.jpg', '000006.jpg', '000007.jpg','000008.jpg', '000009.jpg', '000010.jpg', '000011.jpg', '000012.jpg', '000013.jpg','000017.jpg', '000018.jpg', '000019.jpg', '000020.jpg', '000022.jpg', '000023.jpg','000024.jpg', '000025.jpg', '000026.jpg', '000027.jpg'

皮肤受损图片编号27

皮肤受损图片编号26

皮肤受损图片编号25

皮肤受损图片编号24

Mean AP = 0.9091

# pycharm连接GitHub

Please tell me who you are.

Run

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

to set your account's default identity.

Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got ['Administrator@PC-202004220840.(none)')](mailto:'Administrator@PC-202004220840.(none)'))

解决方案

在命令行中执行  
# git config --global user.email "你的邮箱"  
# git config --global user.name "你的名字"

git config --global user.email ["1162505585@qq.com"](mailto:\"1162505585@qq.com\")

git config --global user.name "徐峰"

错误

author 'xxx' is not 'Name ' and matches no existing author

1、首先看看是不是你的用户名没有权限

2.如果权限是有的，看看用户名是不是填错了。

3.查看自己的用户名和账号：

（1）点击——view>tool windows>terminal——。

（2）在出现的界面中输入以下代码：——git config user.name——（查询用户名）。

——git config user.email——（查询用户邮箱）。

4.在你的项目的./git/config中输入以下字段。（和上一步查询的要填写一样）

——[user]

    name = xxx

    email = xxx@qq.com——

5.在Author中输入你的用户和邮箱，记得格式是xxx<xxx@qq.com>,然后点击commit，然后你就会发现你成功啦！。

6.在你的idea中的右下角的event log中就会出现files committed啦，就表示你成功了！

错误Push to origin/master was rejected

是因为远程仓库里头有文件，新文件无法上传的远程仓库。

在Terminal中使用

git pull

git pull origin master

# 允许不相关的历史提交

git pull origin master --allow-unrelated-histories

命令，将远程仓库中文件合并到本地仓库。

然后在重写push。

错误Push rejected: Push master to origin/master was rejected by remote

推送被拒绝

直接打开你要上传代码的文件夹位置鼠标右键git Bash Here然后直接下面两行命令解决问题

git pull origin master --allow-unrelated-histories  
 git push -u origin master -f

# 参考文献

1. <https://blog.csdn.net/u013085021/article/details/105471252/>
2. 使用 Java API 进行预测. <https://blog.csdn.net/xuanguqiufeng/article/details/104058720>
3. tensorflow训练的模型在java中的使用. <https://blog.csdn.net/uyerp/article/details/70045614>
4. java调用tensorflow训练好的模型.<https://blog.csdn.net/qq_27009517/article/details/84942151>
5. Java调用Tensorflow训练模型预测结果. <https://blog.csdn.net/huanghaocs/article/details/85310256>

[6]输入和输出tensor的确定. <https://blog.csdn.net/u014311125/article/details/103522886>

[7]https://github.com/tensorflow/tensorflow/tree/master/tensorflow

[8]https://github.com/tensorflow/tensorflow/tree/master/tensorflow/java

[9]https://www.jianshu.com/p/e11891418bc1

[10]pycharm连接GitHub.<https://www.jianshu.com/p/60b8440792e2>