workFlow

12.17

任务：

1. 搞清楚网络的输入输出
2. 对图片进行resize进行实验

工作记录：

---

1. **数据输入部分：** train.json文件，文件路径在H["data"]["train.idl"] 里头改。同理，text.json文件是数据的训练集部分。

```
train.json文件格式：
{ "image_path”:"brainwash_11_13_2014_images/0163.png",
"rect" : [{"x1":329.0, "x2":344.0, "y1": 137.0, "y2": 152.0},
            {"x1":391.0, "x2":341.0, "y1": 127.0, "y2": 128.},
             {},{}...]
},{...}
```

2. **使用自己的图片：**只需要修改train.json以及text.json文件即可。
3. **自己的数据图片大小**：需要时32的整数倍，该网络使用的图片大小是：640*480。（32*20,32*15），（20,15）貌似是feature map的大小（不确定？）
4. **evalute.py执行的时候**，需要先进到utils文件夹下，修改Makefile文件，原来的Makefile文件编译器的版本是python2，当我们用python3编译其他文件时会出现以下错误：undefined symbol: _Py_ZeroStruct.
   解决方法就是将all里头的代码改成下面这样，然后重新make一下。

```
all:
        pip install runcython3
        makecython3++ stitch_wrapper.pyx "" "stitch_rects.cpp
./hungarian/hungarian.cpp"
```

5. **预测结果如下(python evaluate.py)：**

---

some tips：

执行train.py:

```
python train.py --hypes hypes/overfeat_rezoom.json --gpu 0 --logdir
output
```

执行evaluate.py:

```
python evaluate.py --weights
output/overfeat_rezoom_2018_12_18_00.03/save.ckpt-999 --test_boxes
data/brainwash/val_boxes.json
```

12.18

任务：

1. 搞清楚网络的预测部分，想办法把输出拿出来
2. 对图片进行resize进行实验
   a. draw a rect with noted location on the image
   b. resize the image
   c. draw a new rect with the calculated loc(according to the resize step),check whether this one matchs to the original one.
3. 研究如何resize image，研究rcnn 的 resize

工作记录：

---

1. **模型的输出：**模型最后会在模型所在文件夹内输出一个预测的**pred_boxes.json，gt_val_boxes.json**文件，里头存着边框的置信度以及边框的大小信息。需要设置一个threshold，来剔除掉一些置信度很低的预测结果。

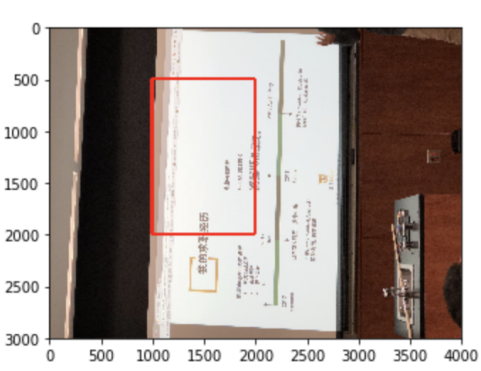
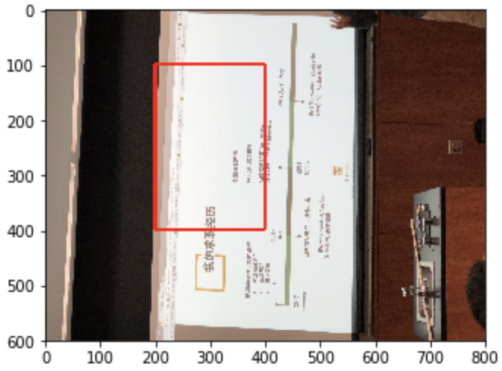
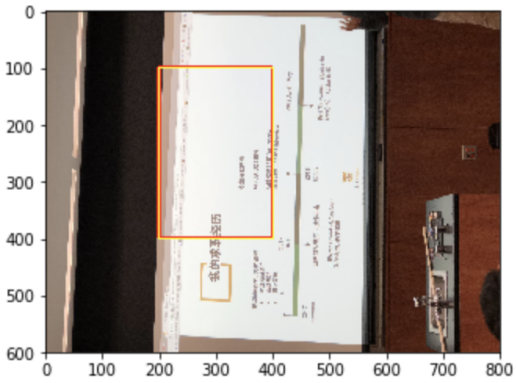
```
"image_path":
"/home/zhouwenhui/AI/head_Detection/TensorBox/data/brainwash/brainwash
_11_24_2014_images/06966500_640x480.png",
    "rects": [
      {
        "score": 0.5663752555847168,
        "x1": 198.5,
        "x2": 229.5,
        "y1": 223.5,
        "y2": 256.5
      },{}......
```

2. **resize 结果：**
   使用的库是

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageDraw,ImageFont
```

在4000*3000的图像上画了一个500*1000的红框坐标为(1000,500),(1000,2000)，(2000,500),(2000,2000)

对合成的图像resize成800*600, 按缩放比绘制黄框，结果重叠，可行。

---

some tips：

Makefile:

> makefile 文件：记录了整个文件的编译以及链接的规则，例如使用的编译器，编译先后顺序等。同时支持执行操作系统指令，与shell类似。
> 使用方法：cd到含有makefile的文件夹目录下，执行make操作即可。

Makefile的学习：

https://seisman.github.io/how-to-write-makefile/introduction.html

http://bbs.chinaunix.net/forum.php?mod=viewthread&tid=408225

12.22

1. **任务：**找到置信度的threshold
2. 将自己数据集的文件转化为可以训练的数据集文件输入格式出将输出数据整理成比赛要求的格式

工作记录：

1. 网络默认**置信度>0.2**的框表示可以接受的框，可以接受。
2. **数据输入输出部分：**

网络要求输入的数据格式如下：

```
{ "image_path”:"brainwash_11_13_2014_images/0163.png",
"rect" : [{"x1":329.0, "x2":344.0, "y1": 137.0, "y2": 152.0},
            {"x1":391.0, "x2":341.0, "y1": 127.0, "y2": 128.},
             {},{}...]
},{...}
```

比赛要求输入的数据格式如下：

> **(图片路径+图片名) 个数 1 x y w h 1 x y w h 1.....**
> **train/path/to/image/2001.jpg 50 1 41 232 52 54 1.....**

网络输出数据格式如下：

```
{ "image_path”:"brainwash_11_13_2014_images/0163.png",
"rect" : [
{"score":0.507844,"x1":329.0, "x2":344.0, "y1": 137.0, "y2": 152.0},
{"score":0.507844,"x1":391.0, "x2":341.0, "y1": 127.0, "y2": 128.},
{},{}...]
},{...}
```

比赛要求输入的测评数据格式如下：

> **图片路径+图片名（train/path/to/image/2001 不带后缀）**
> **5（人头个数)**
> **x y w h confidence**
> **x y w h confidence**
> **...**

3. 将比赛训练数据的格式转换成模型可用的格式:**def** input_matchToModel(path)
4. 将模型输出数据格式转换成比赛输入：**def** output_ModelToMatch(path,conf_threshold)

12.23

1. 对图片进行resize，记录缩放比，修改label。

工作记录：

1. Reinsepect网络中，读入json文件，读取图片的代码如下：

```
def train(H,test_images):  # main.py
    ......
    gen = train_utils.load_data_gen(H, phase, jitter=
                                    H['solver']['use_jitter'])
    d = next(gen) # d = output[image,confs,boxes,flags]
    sess.run(enqueue_op[phase], feed_dict=make_feed(d))
    t = threading.Thread(target=thread_loop,
                         args=(sess, enqueue_op, phase, gen))
    ......
#第三行调用：train_utils.py
def load_data_gen(H,phase, jitter):
    ...
    output['image'] = d['image']
    output['confs'] = np.array([[make_sparse(int(detection),
                         d=H['num_classes'])
                         for detection in cell] for cell in flags])
    output['boxes'] = boxes
    output['flags'] = flags
    yield output
#第六行调用，每次将image，confs,boxes的信息传入enqueue_op
def make_feed(d):
    return {x_in: d['image'],
            confs_in: d['confs'], boxes_in: d['boxes'],
            learning_rate: H['solver']['learning_rate']}
#线程操作，第七行调用
def thread_loop(sess, enqueue_op, phase, gen):
    for d in gen:
       sess.run(enqueue_op[phase], feed_dict=make_feed(d))
```

2. 图片的resize:

```
for anno in annos:
    I = imread(anno.imageName) # 读到数据集图片了
#Skip Greyscale images
    if len(I.shape) < 3:
        continue
    if I.shape[2] == 4:
        I = I[:, :, :3]
    if I.shape[0] != H["image_height"] or I.shape[1] != H["image_width"]:
        if epoch == 0:
            anno = rescale_boxes(I.shape, anno, H["image_height"], H["image_width"]) # 将标记的边框也进行放大
        I = imresize(I, (H["image_height"], H["image_width"]), interp='cubic') # 对图像进行放大，直接缩放
```

3. label中坐标为**(x1,y1)左下，(x2,y2)右上。**

问题：出现了out of memory 的问题，我觉得可能是将图片进行resize的时候载入了太多的图片，导致内存溢出的问题，图片resize的问题需要优化，可以先把图片resize到780*480大小然后载入试试看。

12.24

1. 看代码，解决OOM的问题，重点看resize部分。
2. 实验：将图片事先resize。

工作记录：

1. 将图片resize到640*480，保存到文件夹后，在跑一次数据集。明天看数据结果。

   数据model在output 里面但是不知道是哪个，应该是12.25 0.54这个

12.25

1. 理清网络结构，解决iteration=1000出错，无法存储model的问题。
2. 把所有的图片都resize以及分成test和train，然后放到model中训练。
3. 执行 evaluate.py，查看检测效果。
4. 整理model输出，将输出转化为比赛要求格式。
5. 提交输出。

工作记录：

1. 记录一下chrome的快捷键：

> **ctrl + T：　新建一个页面**
>
> **ctrl + w ： 关闭当前页面**
>
> **ctrl + tab： 切换页面**

2.将part B 分成训练和测试集，重新训练

```
018_12_17_23.10  overfeat_rezoom_2018_12_23_21.08
018_12_17_23.12  overfeat_rezoom_2018_12_23_21.12
018_12_17_23.14  overfeat_rezoom_2018_12_23_21.17
018_12_17_23.38  overfeat_rezoom_2018_12_23_21.26
018_12_17_23.40  overfeat_rezoom_2018_12_25_00.32
018_12_18_00.03  overfeat_rezoom_2018_12_25_00.47
018_12_23_19.39  overfeat_rezoom_2018_12_25_00.54
018_12_23_19.43  overfeat_rezoom_2018_12_25_15.27
018_12_23_19.49  overfeat_rezoom_2018_12_25_15.36
018_12_23_20.55  overfeat_rezoom_2018_12_25_15.38
018_12_23_21.00  overfeat_rezoom_2018_12_25_16.01
018_12_23_21.01
```

3. max_iter 无论多大，到次数都会崩

12.26

1. data_transform.py: 用于将比赛文件与模型文件格式的相互转换。

   比赛数据转换成模型接受数据，用于模型训练：

```
python data_transform.py --txt_path path/to/txt/ --method 1 --
conf_threshold 0.2
```

   模型数据转化为比赛能够提交的数据，用于比赛结果的测评:

```
python data_transform.py --txt_path path/to/txt/ --method 2
实验：将图片事先resize。
```

工作记录：

1. 将图片resize到640*480，保存到文件夹后，在跑一次数据集。明天看数据结果。

12.27

1. 解决evaluate.py文件查找不到图片的问题
2. 将所有的图片上传到服务器，将train里头图片分为valuation和train部分，并写一个json，将所有的图片的json全放到一个json文件里头
3. 将数据传完，使用全部数据进行一下训练，现在先去吃饭
4. 测试一下iteration = 1000次时为啥出错

工作记录：

1. evaluate.py使用如下语句进行文件的读取：

```
orig_img = imread('%s/%s' % (data_dir, true_anno.imageName))[:,:,:3]
```

其中**data_dir为输入的test_boxes的目录，true_anno.imageName**为传入的json文件的**image_path**字段的值，例子：

> **--test_boxes: data/yuncong/valuation.json**
> **[{'image_path':path_A/IMG132.jpg,.....}....]**
> **最终图片路径为：data/yuncong/path_A/IMG132.jpg**
> **json文件中的image_path为相对路径**
> **因此目录结构应为：**
> **data/**
> **--valuation.json**
> **--image_folders**

2. 编写：**jsonOperation.py，**完成数据的分割

用法：

```
## 将所有json文件合为一个
python jsonOperation.py --method 1 --pathList
F:/yuncong/yuncong_data/Mall_train.json,
F:/yuncong/yuncong_data/our_train.json,
F:/yuncong/yuncong_data/Part_A_train.json,
F:/yuncong/yuncong_data/Part_B_train.json
(test到底有没有参与修正model)
## 将所有json文件分为train，test(训练过程中)两部分
python jsonOperation.py --method 2 --pathList
F:/yuncong/yuncong_data/Mall_train.json,
F:/yuncong/yuncong_data/our_train.json,
F:/yuncong/yuncong_data/Part_A_train.json,
F:/yuncong/yuncong_data/Part_B_train.json
```

3. json文件的作用一览表：

All_data.json	All_evaluate(test).json	All_train.json	Mall_train.json	Part_A_train.json	Part_B_train.json	our_train.json
所有图片的集合	所有图片的evaluate集合	所有图片的train集合	Mall图片集合	Part_A_train图片集合	Part_B_train图片集合	our_train图片集合

读代码的一些知识补充

1.

epoch 将所有训练数据集跑一次

iter 将batchsize跑一次

batchsize 用来更新参数的一次

max_iteration 网络的最大迭代次数

例如：假如有1280000张图片，batchsize=256，则1个epoch需要1280000/256=5000次iteration ,它的max-iteration=450000，则共有450000/5000=90个epoch

2.

jitter

在训练时使用scale jittering进行数据增强能有效降低错误率。关于scale jittering的介绍，参考了大牛的讲解，简单来说，就是crop size是固定的，而image size是随机可变的。举例来说，比如把crop size固定在224×224，而image的短边可以按比例缩放到[256, 512]区间的某个随机数值，然后随机偏移裁剪个224×224的图像区域

3.

yield的功能类似于return，但是不同之处在于它返回的是生成器。

```
# 通过`yield`来创建生成器
def func():
    for i in xrange(10);
        yield i

# 通过列表来创建生成器
[i for i in xrange(10)]
```

```
# 调用如下
>>> f = func()
>>> f # 此时生成器还没有运行
<generator object func at 0x7fe01a853820>
>>> f.next() # 当i=0时，遇到yield关键字，直接返回
0
>>> f.next() # 继续上一次执行的位置，进入下一层循环
1
...
>>> f.next()
9
```

参考

https://www.cnblogs.com/coder2012/p/4990834.html

12.28

1. 出现的问题：跑Part_B_train.json时，图片全部生成完毕，但是报错，说x1<x2这个问题，我觉得可能是json的文件末尾有类似空行这种问题。还有可能是网络对json进行resize的时候，将坐标resize出错了，明天定位一下这个错。并解决掉。
2. 第二个要做的是，把数据传到服务器上，把所有数据跑一边，然后用测试数据进行测试。有一个要做的事就是搞清楚没有groundtruth的图片该怎么写json，加到evaluate里头去。
3. model生成的json文件都是经过缩放过的label，需要写一个Python把这个json转换会原来图片的label坐标。
4. 跑一遍数据，看懂源码。

工作记录：

1. 将图片resize到640*480，保存到文件夹后，在跑一次数据集。明天看数据结果。

2.读源码：

Googlenet部分，采用的是直接load一个Googlenet的feature output， 具体的做法是直接以图片为输入，加载Googlenet运行之后的*832*维特征向量

修改思路：

1.首先加载googlenet的图片的大小有规定吗？如果有（我觉得一般就是这里规定了图片的输入）则需要看一下怎么修改这个load googlenet feature的地方

2. 如果没有，是不是可以在这里加一层修改feature size使之能够成为lstm的合法输入？

```
148             cnn = tf_concat(3, (cnn_deconv, cnn[:, :, :, 256:]))
149 #到时候要改
150     elif H['avg_pool_size'] > 1: #5>1 进这里
151         pool_size = H['avg_pool_size'] # 5
152         cnn1 = cnn[:, :, :, :700] #832为700维
153         cnn2 = cnn[:, :, :, 700:] #832为701到832维
154         cnn2 = tf.nn.avg_pool(cnn2, ksize=[1, pool_size, pool_size, 1],# 1,5,5,1
155                               strides=[1, 1, 1, 1], padding='SAME') #padding = same的方式用来填充最后一个pooling不够的情况
156         cnn = tf_concat(3, [cnn1, cnn2])#在第三位上对数据进行叠加
157
158     cnn = tf.reshape(cnn,
159                      [H['batch_size'] * H['grid_width'] * H['grid_height'], H['later_feat_channels']])#1*20*15*832
160     initializer = tf.random_uniform_initializer(-0.1, 0.1)
```

12.29

1. 搞清楚Google net的结构和输入输出
2. 想办法可视化网络结构(tensor board)
3. 重点其实是在修改Google net的前后输入输出
4. output 的 json 转换成比赛要求的格式
5. 所有数据跑一遍，看效果（但是数据集传不上去）

工作记录：

1.

> scp指令的规则：
> scp -p xxxxx sourceuser@sourcehost:/path/to/source/file destinationuser@destinationhost:/path/to/destination/

从服务器下载文件

```
scp -P 11111 zhouwenhui@ictvr.ml:~/AI/head_Detection/TensorBox/image_resize.py /Users/tangye/Desktop/tangye/
```

上传文件到服务器

```
scp -P 11111 /Users/tangye/Desktop/tangye/hhh.ipynb zhouwenhui@ictvr.ml:~/AI/head_Detection/TensorBox/
```

3. 服务器连接：

> 新Server（Titan XP × 4）
> 内网访问：ssh xxx@10.26.0.189
> 外网访问：ssh xxx@ictvrml -p 11112
>
> 旧Server（1080Ti × 4）
> 内网访问：ssh xxx@10.26.0.188
> 外网访问：ssh xxx@ictvr.ml -p 11111

4. 解压指令：tar:

```
压缩： tar -czvf file.tar.gz ./head_Detection
解压： tar -xzvf file.tar.gz
```

5. .bashrc文件用于配置用于设置linux上一些工具的用户环境配置（例如快捷键的设置等等，此类文件很多），放在~/bashrc中。
6. cv2显示图片，并将矩形标注绘制在图片上

```
img = cv2.imread(img_path)
img = cv2.resize(img, (640, 480), interpolation=cv2.INTER_CUBIC)
cv2.namedWindow('Image')
# 输入参数分别为图像、左上角坐标、右下角坐标、颜色数组、粗细
cv2.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (0, 0,
255), 1)
cv2.imwrite('qq_big.jpg', aimg)
cv2.imshow('image', aimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. 完成了模型输出数据到原始数据的转换，完成boxes的绘制，并保存：

> 原模型输出的图片大小全部都是[640,480]，因此图片的boxes部分对应的是该尺寸下的boxes，因此需要将其转换成原始图片的大小，修改相应的boxes。
> 将boxes框直接绘制在图片上，可用来后期看模型绘制的结果。

8. 配置服务器运行的环境：

> cuda: 9.1，cuDNN: 7.1.3，tensorflow: 1.8.0

---

linux上直接下载google drive文件专题

首先将**单个文件**上传到google drive中（注意单个文件，可以打包成zip），然后右键该文件，选择“获取共享链接”，即可得到如下的链接，我们记录下它的id部分：

> https://drive.google.com/open?id=**16Toya_aBhHuEWSUiAa_C92Q3cc9fK3ie**

（一）若你有root权限，执行以下指令：
下面指令意思是上**moeclub.org**网站上下载gdlink.sh文件，并将其命名为/usr/local/bin/gdlink (-qO)，这条操作需要root权限，同时使得第二条指令，可以直接使用gdlink来执行gdlink.sh。随后的chmod 将gdlink执行的权利赋给了所有用户。

```
wget --no-check-certificate -qO /usr/local/bin/gdlink
'https://moeclub.org/attachment/LinuxShell/gdlink.sh' && chmod a+x
/usr/local/bin/gdlink
```

这条指令引号内部即为**上面链接的id，**filename为你希望保存成的文件名，即用gdlink.sh这个脚本完成了将google drive 转为直链接，使用wget即可获得资源。

```
gdlink '18v_9Bet11B3ZZ3GqWCOa3yTuY2Lzzaav' |xargs -n1 wget -c -O
./filename
```

（二）若你没有root权限：
浏览器输入下面网址，下载gdlink.sh文件：

> https://moeclub.org/attachment/LinuxShell/gdlink.sh

执行下面语句，给所有用户授权：

```
chmod a+x ./gdlink.sh
```

执行下面指令，与上面左右相同，手动bash，执行gdlink.sh：

```
bash ./gdlink.sh '18v_9Bet11B3ZZ3GqWCOa3yTuY2Lzzaav' |xargs -n1 wget -c -O ./filename
```

12.30

1. 把数据全部传到服务器上面去
2. 试着可视化一下网络，顺便学习一下
3. 跑一下网络

历史遗留问题：

训练好的model跑Part_B_train.json时，图片全部生成完毕，但是报错，说x1<x2这个问题

工作记录：

1. 将图片resize到640*480，保存到文件夹后，在跑一次数据集。明天看数据结果。

2019.01.02

1. 试着对网络进行可视化，学习网络结果
2. 将训练好的模型拿到测试数据上看看跑出的效果

历史遗留问题：

训练好的model跑Part_B_train.json时，图片全部生成完毕，但是报错，说x1<x2这个问题：

解答：这个问题在于有些**rect矩形框**存在矩形的左上角的点和右下角的点是同一个位置，导致height和weight均变为0。

工作记录：

1. 将所有数据用模型跑了一遍，发现检测效果还是不错的。
2. 将竞赛给的测试数据整理成:All_data_test.json文件。

```
[{"image_path":path/to/image/xx.jpg,"rects":[]},{}...]
```

3. 用所有的训练数据：**All_train.json**，与**All_data_test.json**共同组成训练模型，进行模型的训练。今天晚上跑完这个模型，明天跑一下evaluate.py看一下结果。

2019.01.03

1. 把训练好的模型evaluate.py一下，然后把json修改成竞赛要求的格式。

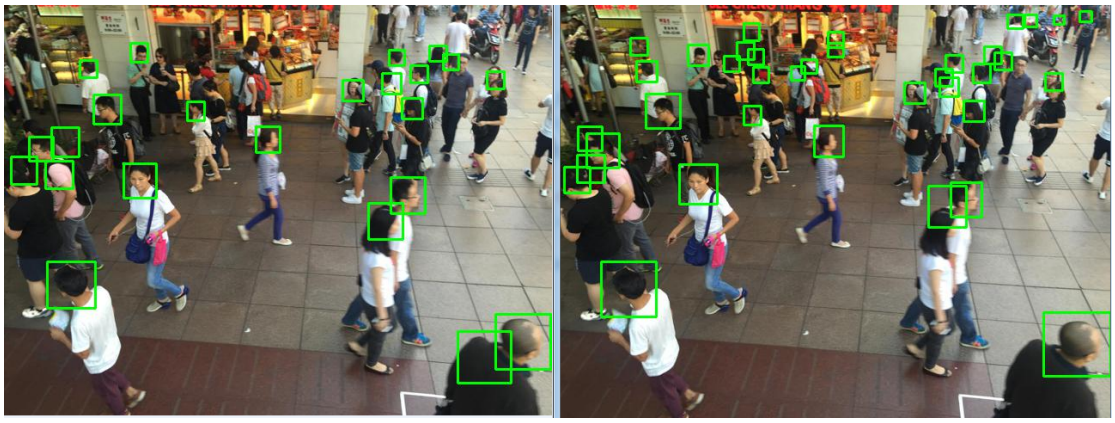
2. 如果效果不好，我觉得可以提高一下confs的大小。
3. 试着选一个训练次数的样本，使得效果最好。
4. 试着对网络进行可视化，学习网络结构。

工作记录：

1. 将所有数据用模型跑一边
2. 发现数据中有些图片是灰度图，因此通道数只有1，需要把它修改成三通道的图片，只需要在每一个通道上放相同的灰度值即可。

```
img = cv2.imread('img.jpg')
src_RGB = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
```

3. 以下是训练了50000与500000次的模型的训练结果。



500000次训练的样本细节更丰富，但是也有相当多的错误。因此需要做一下对比，找出最好的训练模型。

4. 下午将生成的json数据恢复到原始大小后，转成txt格式，上传竞赛页面。
5. github tensorbox:https://github.com/russell91/tensorbox

2019.01.04

1. 提交了一版结果，发现效果很不好，虽然提交的版本很粗糙。感觉可以调整的地方有几点：
   a. 人头很多的图片检测的效果很差，仅仅能检测出少部分的头。
   b. 由于比赛网站每天只能提交一次结果，提交结果之前先画出来，分析结果。
   c. 调整conf_threshold，即边框的置信度，控制边框的多少。
   d. 提交结果为（x,y,w,h,confs），其中x,y,w,h均为整数，w = h，可以在提交的时候做一下调整。
   e. 选一个训练次数刚好的样本，使得最后的效果最好。
2. 试着对网络进行可视化，学习网络结构。
3. 明天试着调整一下输出的结果，看看能不能提升一下结果：思路就是对比图片，

工作记录：

1. 将所有数据用模型跑一遍。

2. 我觉得实验结果不应该这么差，需要对跑出来的json和图片进行一下对比。

做完实验后发现evaluation出来的label（json文件）就是还原回原图大小的坐标点，因此没有错

2019.01.06

工作记录：

1. 在AI/head_Detection/TensorBox/tensorboard/ 里面做了一个可视化实验, 首先把要画图的部分用一个namescope 包起来，然后用tf.summary.FileWriter存在指定目录底下。最好是与.py文件同级的logs文件夹底下。

```
import tensorflow as tf

with tf.name_scope('graph') as scope:
     matrix1 = tf.constant([[3., 3.]],name ='matrix1')  #1 row by 2
column
     matrix2 = tf.constant([[2.],[2.]],name ='matrix2') # 2 row by 1
column
     product = tf.matmul(matrix1, matrix2,name='product')

sess = tf.Session()

writer = tf.summary.FileWriter("logs/", sess.graph) #第一个参数指定生成文
件的目>录。

init = tf.global_variables_initializer()

sess.run(init)
```

然后在logs所在的目录下运行

```
tensorboard --logdir=logs
```

由于用的是服务器，在本地终端输入

```
ssh zhouwh@ictvr.ml -p 11112 -N -L localhost:6006:localhost:6006
```

在浏览器里面输入：

```
localhost:6006
```

就能显示
我们的model

2019.01.09

faster Rcnn

https://github.com/jwyang/faster-rcnn.pytorch

trainval_net.py

238-248定义了网络结构，之前的都是在处理数据和传入一些超参数。

268 行定义了 optimizer

295 行定义了是否使用tensorboard

现在只要看懂238-248，注意rpn_loss_cls是怎么来的就行

2019.01.10

现在要看我们的model只要

cd到

```
/AI/head_Detection/TensorBox/
```

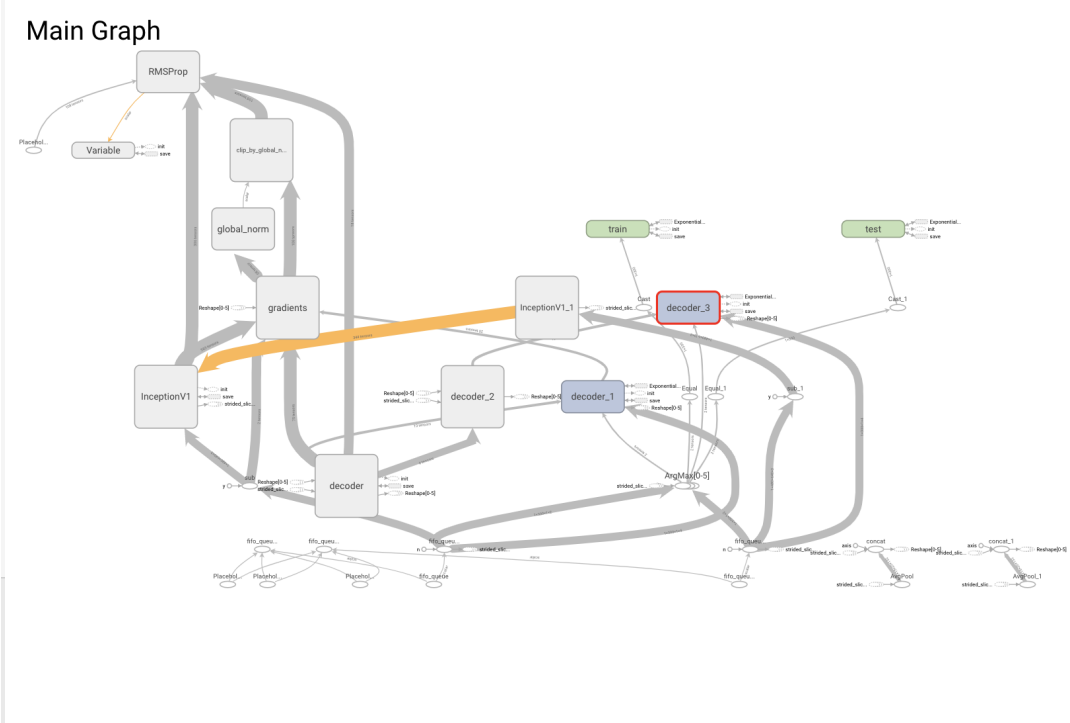
输入

```
tensorboard --logdir output
```

在本地输入

```
ssh zhouwh@ictvr.ml -p 11112 -N -L localhost:6006:localhost:6006
```

就行



2019.01.12

1. 学习一下tensorboard

工作记录：

tensorbard：

【参考视频】：https://www.youtube.com/watch?v=eBbEDRsCmv4

【参考代码】：https://github.com/larsaurdal/TF_MNIST_Demo/blob/master/TF_MNIST_demo.ipynb

- **作用：**用来可视化模型训练的计算图结构，以及实时绘制一些变量形成一个表格，如loss,accuracy的值。
- **tensorboard工作方式：**从硬盘读取模型记录的文件，并在tensorboard中可视化。将模型数据写入磁盘的函数如下，他可写入任何你想到在tensorboard中可视化的数据：

```
tf.summary.FileWriter(n)
```

将数据写入tensorboard的方法：

```
writer = tf.summary.FileWriter('path/to/logdir')#创建一个filewriter对象
writer.add_graph(sess.graph) # 给tensorboard添加一个图sess
```

本地代码启动tensorboard，在命令行输入:

```
tensorboard --logdir path/to/output/log
```

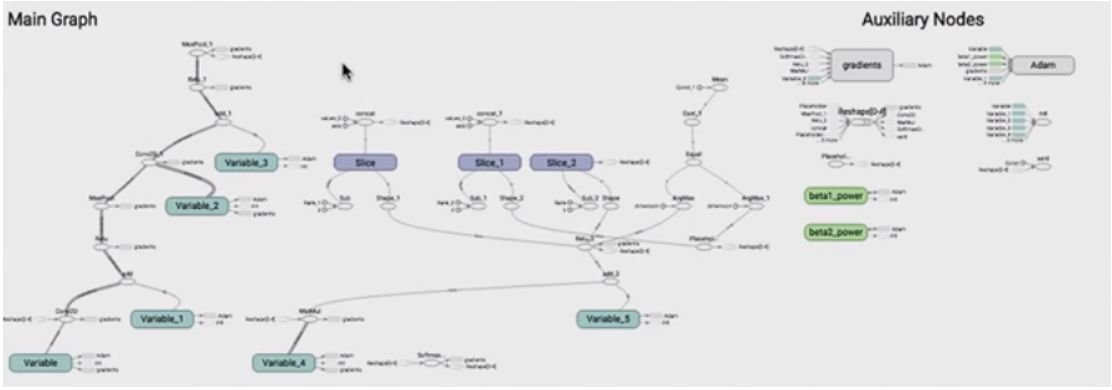
若在服务器执行代码，本地启动tensorboard：

```
ssh zhouwh@ictvr.ml -p 11112 -N -L localhost:6006:localhost:6006
```

在浏览器里面输入如下指令即可显示：

```
localhost:6006
```

一个原生的graph长这个样子，难以理解，需要对图进行清理，给graph赋予结构。



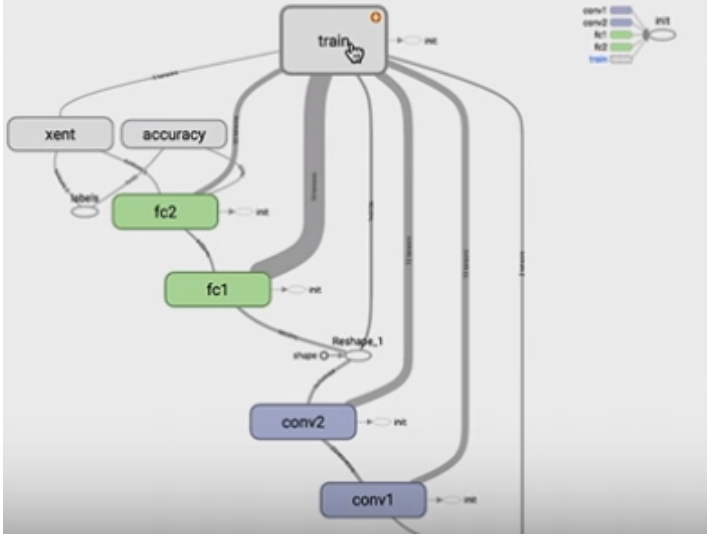
tensorboard命名系统如下：

```
Node Names    # 给每个操作或变量赋名
Name Scopes   #命名空间（可以在不同命名范围内给相同操作起相同名称）
```

例如：

```
def conv_layer(input,channels_in,channel_out,name="conv"):
    with tf.name_scope(name):  ##创建了一个命名空间,给命名分组
        w = tf.Variable(tf.zeros([5,5,channels_in,channels_out]),
            name='W')#权重名w起名为W
        b = tf.Variable(tf.zeros([channels_out]),name = "B")#bias起名B
         ....
```

清理后的图结构如下：



以上是计算图的结构可视化部分，接下来介绍tensorboard可视化训练过程中产生的数据。

- **向硬盘中写入数据：**

tensorboard通过一个summary函数，summary函数可以将你graph中定义的张量输出到协议缓冲区（序列化数据）从而写入硬盘，以下有几种不同的summary函数：

```
tf.summary.scalar #标量summary，写下单一数值，可用来构造折线图
tf.summary.histogram #柱状图，附加上变量可以显示变量的分布，如权重w
tf.summary.image   #用来显示图片，例如生成的图片
tf.summary.audio   #记录声音
```

例如：

```
tf.summary.scalar('cross_entropy',xent)
tf.summary.scalar('input',x_image,3)
tf.summary.histogram('weights',w)
```

如何执行这些summary呢，可以一个一个执行，或合并起来一起执行：

```
merged_summary = tf.summary.merge_all()#将把所有数据一次性存起来
...
for i in range(2000):
    batch = mnist.train.next_batch(100)
    if i%5 == 0: #每五次循环记录一下batch[0]和batch[1]
        s = sess.run(merged_summary,feed_dict={x:batch[0],y:batch[1]})
        writer.add_summary(s,i)
    ....
```

tensorboard同时可以对多个模型进行对比，只要我们在启动的时候将logdir指向log的父目录，该目录下有多个模型，启动tensorboard在命令行输入:

```
tensorboard --logdir path/to/output/
```

embedding visualing:该功能可以将数据分布展示成三维分布，非常酷炫。