tiny YOLO v3做缺陷检测实战

原创 来一包板栗 2018-12-14 21:28:01 16243 收藏 131 版权

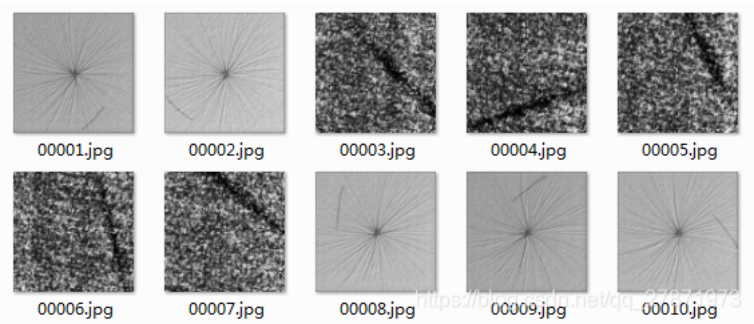
分类专栏： 深度学习 表面缺陷检测 文章标签： 缺陷检测 tiny-yolo v3 图像标注

前言：接触yolo网络是在七月份，当时把yolo检测的论文以及R-CNN系列，SSD等一些论文看了一下，感觉内容很丰富，也尝试了darknet版本的实现，和yolo v3的实现，在网上也有很多关于上面两种的实现，这里就不讲了。九月份用tiny-yolo v3做了一个缺陷检测的实验，效果出乎意料，准确率和召回率“满分”！！过了三个月才想着把以前的实验总结一下，真不应该。下面从头开始说明怎么在自己的数据集上实现tiny-yolo v3，码字不易，给赞啊，代码是在别人的基础上修改的，放到了github上，**感谢加星**：https://github.com/Eatzhy/tiny-yolov3 有问题欢迎交流，会帮忙解决。

1、对图像进行转格式和编队

因为实验使用的缺陷图像为DAGM的数据集，大小为512×512格式为PNG，而tiny-yolo v3输入的格式为jpg，大小为416×416(也有人说tiny网络输入是224×224大小，我们先不管到底哪个，代码在修改的时候是416)，对图像转格式，编队代码如下。

将代码copy下来和缺陷图像文件夹放到一个路径，改一下代码的路径名就ok。转换完之后大概是这个样子（部分图片）：

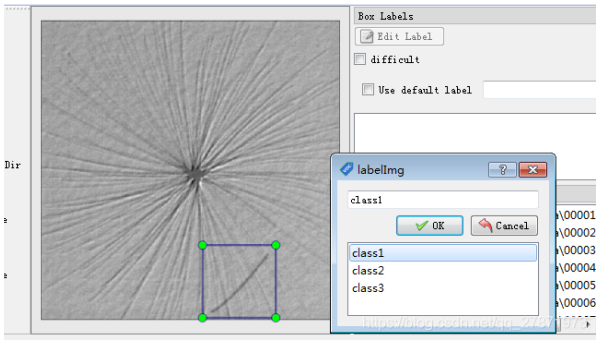



```
# -*- coding: utf-8 -*-

'''
将png转jpg
resize到416
并给图片编队
待转换的图像存放在data下，程序运行后，data下获得的是jpg格式
pre_data存放是png格式
'''
import os
from PIL import Image
import shutil
import sys

#创建一个文件，存放原图
output_dir = 'pre_data'
if not os.path.exists(output_dir):
  os.mkdir(output_dir)

def image2png(dataset_dir,type):
    #转换格式并resize到一定大小
    files = []
    image_list = os.listdir(dataset_dir)
    files = [os.path.join(dataset_dir, _) for _ in image_list]
    for index,png in enumerate(files):
        if index > 100000:
            break
        try:
            sys.stdout.write('\r>>Converting image %d/100000 ' % (index))
            sys.stdout.flush()
            img = Image.open(png)
            img = img.resize((416,416))
            jpg = os.path.splitext(png)[0] + "." + type
            img.save(jpg)
            # 将已经转换的图片移动到指定位置
            shutil.move(png, output_dir)
        except IOError as e:
           print('could not read:',jpg)
```

```
            print('error:',e)
                print('skip it\n')
    sys.stdout.write('Convert Over!\n')
    sys.stdout.flush()

def rename(path):
    #给图片编队函数
    filelist = os.listdir(path)  #获取文件路径
    total_num = len(filelist)  #获取文件长度
    i = 1  #文件从1开始命名
    for item in filelist:
        if item.endswith('.jpg'):
            src = os.path.join(os.path.abspath(path), item)
            #dst = os.path.join(os.path.abspath(path), ''+str(i) + '.jpg')
            dst = os.path.join(os.path.abspath(path), '00' + format(str(i), '0>3s') + '.jpg')
        try:
            os.rename(src, dst)
            print ('converting %s to %s ...' % (src, dst))
            i = i + 1
        except:
            continue
        print ('total %d to rename & converted %d jpgs' % (total_num, i))


if __name__ == "__main__":
  current_dir = os.getcwd()
  print(current_dir)
  data_dir = 'data/'  #待转化图像文件
  image2png(data_dir,'jpg')
  rename(data_dir)
```

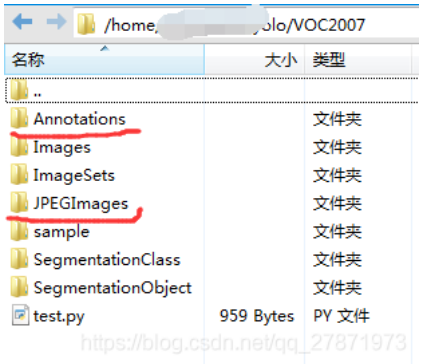
2、对缺陷图像集做标注

图像标注比较消耗人的耐力和专注力，废话不多话。这次缺陷检测实验使用的图库为：DAGM 2007的数据集中的其中三类，图像标注推荐使用 labelImg软件，关于这个软件的安装和使用参考我的一篇博文：labelImg的安装和使用，内容很详细，如果出问题，请留言。标注的过程大概是这样：



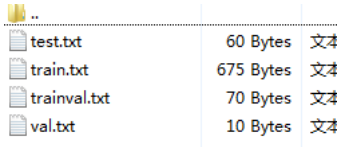
3、在github上下载tiny-yolo v3工程

将标注后的图片和xml文档分别放到tiny-yolo v3文件下的VOC2007文件下的JPEGImages文件和Annotation文件下。



4、编译程序运行

使用TensorFlow的编译器Spyder运行VOC2007文件下的test.py程序，会在VOC2007/ImageSets/Main下生产如下几个txt文档，就是对图片路径按照比例分成训练，验证，测试集，测试集用不上。



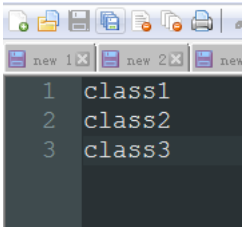
5、

对tiny-yolo v3下的voc_annotation.py进行简单修改，并运行。**在tiny-yolo v3下会生成几个txt文档，比如：2007_train.txt**，这时候我们将前面的**2007_**删掉。因为训练的缺陷图像为三类，voc_annotation.py修改如下：

```
import xml.etree.ElementTree as ET
from os import getcwd

sets=[('2007', 'train'), ('2007', 'val'), ('2007', 'test')]

classes = ["class1", "class2", "class3"]


def convert_annotation(year, image_id, list_file):
    in_file = open('VOC%s/Annotations/%s.xml'%(year, image_id))
    tree=ET.parse(in_file)
    root = tree.getroot()

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult)==1:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (int(xmlbox.find('xmin').text), int(xmlbox.find('ymin').text), int(xmlbox.find('xmax').text), int(xmlbox.fir
        list_file.write(" " + ",".join([str(a) for a in b]) + ',' + str(cls_id))

wd = getcwd()

for year, image_set in sets:
    image_ids = open('VOC%s/ImageSets/Main/%s.txt'%(year, image_set)).read().strip().split()
    list_file = open('%s_%s.txt'%(year, image_set), 'w')
    for image_id in image_ids:
        list_file.write('%s/VOC%s/JPEGImages/%s.jpg'%(wd, year, image_id))
        convert_annotation(year, image_id, list_file)
        list_file.write('\n')
    list_file.close()
```

更新补充：

写的时候忘记了一步，还有一处需要修改。路径 yolo/model_data下的voc_classes.txt也要修改，因为我们这次做的三个类别class1,class2,class3的检测，所以修改如下：



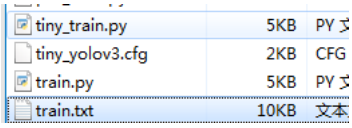
如果要对其他检测，需要把三个类别名字改成其他的，注意字符的分行，建议使用notepad++修改，因为python在判定类别的时候使用的是len()函数。

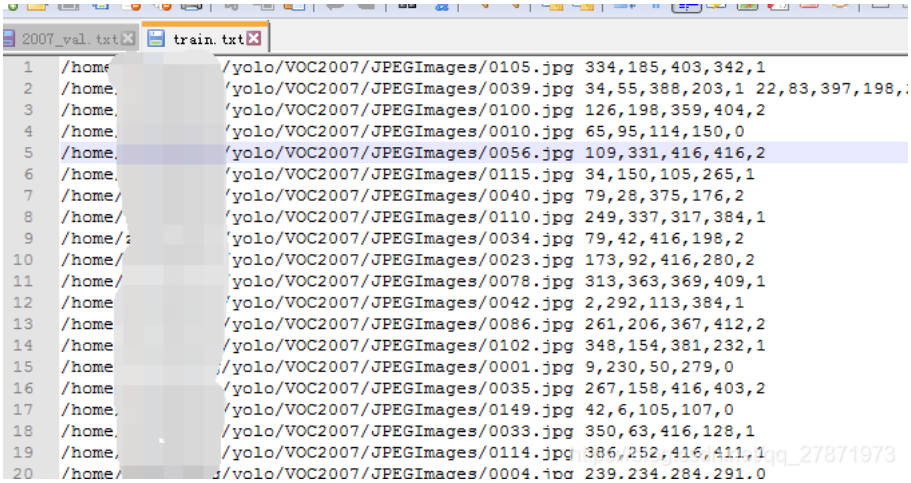
5.1、前面不是说过了吗，在划分验证，测试，训练集时，训练和测试集在实际程序运行中根本没有用上，因为在程序tiny-train.py中有一个函数已经说明了问题，如下图，代码中的annotation_path输入的是train.txt，然后对其中的图片重新划分训练和测试集。总数×val_split就是训练图片数量。

所以我们使用Notepad++打开test.txt和val.txt，将其中的路径复制到train.txt中，最后只保留train.txt，如下：

```python
#函数定义
def train(model, annotation_path, input_shape, anchors, num_classes, log_dir='logs/'):
    model.compile(optimizer='adam', loss={
        'yolo_loss': lambda y_true, y_pred: y_pred})
    #记录所有训练过程，每隔一定步数记录最大值
    tensorboard = TensorBoard(log_dir=log_dir)
    checkpoint = ModelCheckpoint(log_dir + "best_weights.h5",
                                 monitor="val_loss",
                                 mode='min',
                                 save_weights_only=True,
                                 save_best_only=True,
                                 verbose=1,
                                 period=1)

    callback_lists=[tensorboard,checkpoint]
    batch_size = 16
    val_split = 0.05
    with open(annotation_path) as f:
        lines = f.readlines()
    np.random.shuffle(lines)

    num_val = int(len(lines)*val_split)
    num_train = len(lines) - num_val
    print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))

    model.fit_generator(data_generator_wrap(lines[:num_train], batch_size, input_shape, anchors, num_classes),
            steps_per_epoch=max(1, num_train//batch_size),
            validation_data=data_generator_wrap(lines[num_train:], batch_size, input_shape, anchors, num_classes)
            validation_steps=max(1, num_val//batch_size),
            epochs=3000, #迭代的步数
            initial_epoch=0, callbacks=callback_lists, verbose=1)
    model.save_weights(log_dir + 'tiny-trained_weights.h5')
```


图1 tiny-train.py





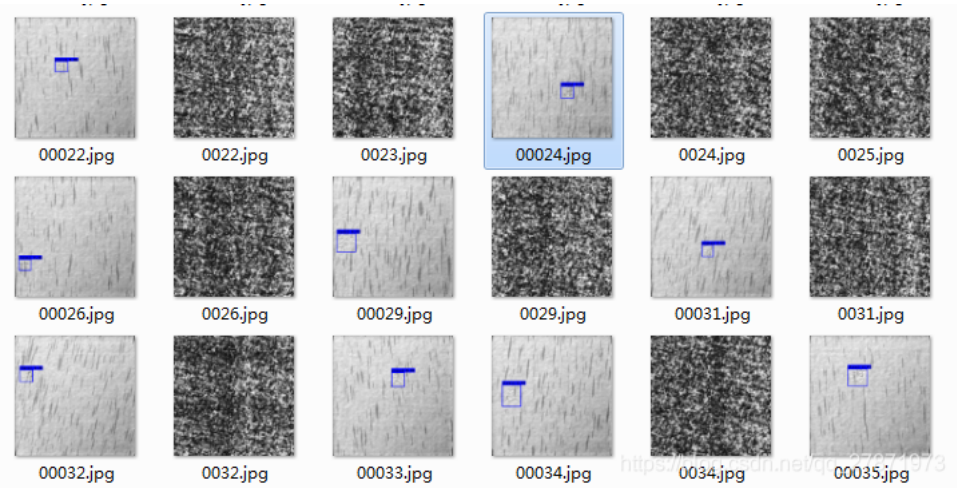
复制路径

6、准备训练

然后对一些代码文件进行修改，就可以开始训练了，这里就不一一叙述具体的修改了。直接在博客前言的github中下载修改好的代码即可，欢迎加星，有问题的话请提问，会协助解决。

训练使用：tiny-train.py，批量测试使用：yolo-test-batch.py，将待测试的图像放在VOC2007文件下的Images文件中。测试结果会在VOC2007下的SegmentationClass文件中。**待测图像使用步骤1进行转格式和编队、resize。**放一张训练过程的图和测试结果图：

```
Epoch 4/3000
8/8 [==============================] - 6s 769ms/step - loss: 85.4440 - val_loss: 151.5055

Epoch 00004: val_loss did not improve from 135.77103
Epoch 5/3000
8/8 [==============================] - 6s 744ms/step - loss: 58.3442 - val_loss: 114.8700

Epoch 00005: val_loss improved from 135.77103 to 114.86998, saving model to logs/best_weights
.h5
```



程序是在ubantu下跑的，当然windows系统也可以，只是文件文件需要修改，码字不易，欢迎交流、点赞，给github加星。

当然，稍微修改也可以测试YOLO v3，和是否加载预权重的程序，以及对其中的特征网络等部件修剪，博主都做了。