数据库融合说明文档

**一、所用数据库**

**1.1 数据来源**

在收集数据的过程中，我们的数据集主要从两方面获取：

**（1）当前各个深度学习数据平台**

**（2）Python爬虫获取**

**1.2 数据类型**

我们共训练两种模型，**图像分类模型**与**目标检测模型**，分别对应不同数据集。

获取的数据主要有两类，分别是水果花卉的图片，以及目标检测所需的.xml标注文件。

**1.3 数据库明细**

以下是我们在各个深度学习数据网站上搜集到的数据

* **ImageNet**（<http://www.image-net.org/>）

ImageNet是我们组最主要来源的数据集，因为ImageNet数据多，且质量较高，而且存在大量目标检测边界框。在本小组的数据当中，约60%以上的图像分类数据与目标检测数据均来自于ImageNet。

* **Oxford 102 Flower**（<https://www.robots.ox.ac.uk/~vgg/data/flowers/102/>）

花卉经典数据集，本数据集包含 102 种花类的图像数据集（，每个类别包含 40—258 张图像。这些图像在比例、姿势以及光照方面有着丰富的变化。我们收集了大量图片，用于图像分类模型。

* **Kaggle中数据集：**

**Fruit Images for Object Detection**（<https://www.kaggle.com/mbkinaci/fruit-images-for-object-detection>）

四种类别，包含多种水果混合图片，收集用于图像分类。

**Flower Recognition**（<https://www.kaggle.com/alxmamaev/flowers-recognition>）

谷歌花卉数据集，五种类别花卉，类别少但质量很高，收集用于图像分类。

* **百度paddle中数据集**：

**PaddleDetection水果目标检测**（<https://aistudio.baidu.com/aistudio/datasetdetail/34327>）

百度的初级目标检测数据，收集用于目标检测。

**水果分类模型**（<https://aistudio.baidu.com/aistudio/datasetdetail/30069>）

水果图像的数据集，包含标签文本，xml信息，收集用于目标检测。

**1.4 爬虫收集**

除此之外，我们还用到了python爬虫，在google图库与百度图库爬共3000张图片左右，作为数据集的补充。

**1.5 数据统计**

**以上图片共有30类，每类图片有1100张，共计33000张。**

图片数据库已经上传到百度网盘当中，关于数据集的下载详见：

链接：<https://pan.baidu.com/s/1zUK0LPZ6SO8tl9ri0GQQdw>

提取码：5q9o

1. **融合数据库所做处理**

**2.1 图像分类数据处理**

**1.图像分类图片格式处理**

图像分类模型需要维度相同的图片作为训练集，一般图片均处理成.jpg格式。所以我们首先将所有图片统一处理为jpg格式，方便后续进行统一的操作。

**2.图像分类图片通道处理**

在爬取或下载的数据图片当中，会存在一部分黑白图片。黑白图片为单通道图片，维度与彩色图片不同。在本步骤里，我们需要查看图片是否均为彩色图片，如果并非彩色图片而是黑白图片，则删除对应图片，保证所有图片均为三通道图片。

**3.数据集划分**

在训练的过程中，我们需要使用到训练集与测试集。我们对数据进行划分，将其以10:1的比例分为测试集和训练集。首先读取对应数据，打乱数据后以对应比例划分训练集与验证集。

**2.2 目标检测数据处理**

**1.目标检测图片数据合并**

首先将下载好的各类目标检测数据进行合并，将图片与标注文件进行匹配，并将所有处理好的数据放入一个文件夹内。

**2.目标检测图片格式处理**

目标检测模型除了图片之外，还需要边界框的数据，这种数据以.xml格式储存。需要将所有的xml相关信息进行汇总，并放入csv列表中。

**3.目标检测图片生成TFRecord数据**

使用Tensorflow训练神经网络时，如果数据较多，直接从硬盘中进行读取效率比较低，所以我们采用 tfrecord数据存储形式。Tensorflow有和tfrecord配套的一些函数，可以加快数据的处理。

* 1. **图像分类数据处理部分代码**

1.图像分类数据格式处理

图像分类模型需要维度相同的图片作为训练集，一般图片均处理成.jpg格式。

1. **from** PIL **import** Image
3. **def** IsValidImage(img\_path):
4. bValid = True
5. **try**:
6. Image.open(img\_path).verify()
7. **except**:
8. bValid = False
9. **return** bValid
11. **def** transimg(img\_path):
12. **if** IsValidImage(img\_path):
13. **try**:
14. str = img\_path.rsplit(".", 1)
15. output\_img\_path = str[0] + ".jpg"
16. im = Image.open(img\_path)
17. im.save(output\_img\_path)
18. **return** True
19. **except**:
20. **return** False
21. **else**:
22. **return** False

2、图像分类数据通道处理

在爬取或下载的数据图片当中，会存在一部分黑白图片。黑白图片为单通道图片，维度与彩色图片不同。在本步骤里，我们需要查看图片是否均为彩色图片，如果并非彩色图片而是黑白图片，则删除对应图片，保证所有图片均为三通道图片。

1. # 已处理过格式的图片
2. path = './fruit\_flower/'
4. #逐个文件夹遍历
5. **for** item1 **in** os.listdir(path):
6. path1 = path + item1 + '/'
7. **print**(item1)
8. **for** item2 **in** os.listdir(path1):
9. path2 = path1 + item2
10. # 判断是否格式错误
11. **if** os.path.getsize(path2) == 0:
12. **print**(path2)
13. **else**:
14. img = Image.open(path2)
15. # 判断是否为彩色图片
16. **if** np.shape(img\_array.shape)[0] != 3:
17. **print**('p', path2)
18. os.remove(path2)
    1. **图像分类数据处理部分代码**

1.目标检测图片数据合并

首先将下载好的各类目标检测数据进行合并，将图片与标注文件进行匹配，并将所有处理好的数据放入一个文件夹内。

1. # 初始数据文件夹爱
2. path = './newimg/flower/'
3. # 新数据文件夹
4. newimg = './newimg/combine/'
5. t = 0
7. # 判断是否存在文件夹
8. **if** **not** os.path.exists(newimg):
9. os.mkdir(newimg)
10. **if** os.listdir(newimg):
11. shutil.rmtree(newimg)
12. os.mkdir(newimg)
14. # 逐个文件夹进行遍历
15. **for** items1 **in** os.listdir(path):
16. path1 = ori1 + items1 + '/'
17. i = 0
18. **for** items2 **in** os.listdir(path1):
19. i += 1
20. **print**(int((i/2)+1))
21. **for** p **in** range(1, int((i/2)+1)):
22. shutil.copy(path1 + str(p) + '.jpg', newimg)
23. shutil.copy(path1 + str(p) + '.xml', newimg)
24. os.rename(newimg + str(p) + '.jpg', newimg + 'img' + str(t) + '.jpg')
25. os.rename(newimg + str(p) + '.xml', newimg + 'img' + str(t) + '.xml')
26. t += 1

2.目标检测数据生成TFRecord数据

目标检测模型除了图片之外，还需要边界框的数据，这种数据以.xml格式储存。需要将所有的xml相关信息进行汇总，并放入csv列表中，方便TFRecord数据的生成。

1. **import** xml.etree.ElementTree as ET
3. **def** xml\_to\_csv(path):
4. xml\_list = []
5. **for** xml\_file **in** glob.glob(path + '/\*.xml'):
6. tree = ET.parse(xml\_file)
7. root = tree.getroot()
8. **for** member **in** root.findall('object'):
9. value = (root.find('filename').text,
10. int(root.find('size')[0].text),
11. int(root.find('size')[1].text),
12. member[0].text,
13. int(member[4][0].text),
14. int(member[4][1].text),
15. int(member[4][2].text),
16. int(member[4][3].text)
17. )
18. xml\_list.append(value)
19. column\_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
20. xml\_df = pd.DataFrame(xml\_list, columns=column\_name)
21. **return** xml\_df
23. **for** folder **in** ['train','test']:
24. image\_path = os.path.join(os.getcwd(), ('images/' + folder))
25. xml\_df = xml\_to\_csv(image\_path)
26. xml\_df.to\_csv(('images/' + folder + '\_labels.csv'), index=None)
27. **print**('Successfully converted xml to csv.')

**三、MongoDB中储存情况**

* **首先引入新的程序集“MongoDB.GridFS.dll”，使用“GridFS”方式进行操作。**

我们知道MongoDB的文档结构为BJSON格式，支持保存二进制格式的数据。但是由于一个BJSON的最大长度不能超过4M，所以我们使用samus驱动提供的“GridFS”方式来支持图片数据的存储。"

* **将图片数据与图片属性数据分开保存。**

用chunks来保存图片数据，files保存属性数据，一个图片file可能对应多个chunks，每个chunk的内存大小固定（16M），若图片数据大于chunk，则分为多个chunk保存，用同一个ObjectID关联，下载时自动将多个chunk合并为图片数据。

* **图片具体存储情况**

图片共有30类，其中花卉与水果各有15类。

每类图片包含1000张训练集，100张测试集，数据集总共有33000张图片。

