18 JULY 2017 / TENSORFLOW

TensorFlow中的TFRecord文件

背景：最近在学习TensorFlow，需要将自定义图像数据作为训练数据。

标准TensorFlow格式

TensorFlow的训练过程其实就是大量的数据在网络中不断流动的过程，而数据的来源在官方文档[^1]（API r1.2）中介绍了三种方式，分别是：

- Feeding。通过Python直接注入数据。
- Reading from files。从文件读取数据，本文中的TFRecord属于此类方式。
- Preloaded data。将数据以constant或者variable的方式直接存储在运算图中。

当数据量较大时，官方推荐采用标准TensorFlow格式[^2]（Standard TensorFlow format）来存储训练与验证数据，该格式的后缀名为 tfrecord 。官方介绍如下：

> *A TFRecords file represents a sequence of (binary) strings. The format is not random access, so it is suitable for streaming large amounts of data but not suitable if fast sharding or other non-sequential access is desired.*

从介绍不难看出，TFRecord文件适用于大量数据的顺序读取。而这正好是神经网络在训练过程中发生的事情。

如何使用TFRecord文件

对于TFRecord文件的使用，官方给出了两份示例代码，分别展示了如何生成与读取该格式的文件。

生成TFRecord文件

第一份代码 convert_to_records.py [^3]将MNIST里的图像数据转换为了TFRecord格式。仔细研读代码，可以发现TFRecord文件中的图像数据存储在 Feature 下的 image_raw 里。 image_raw 来自于 data_set.images ，而后者又来自 mnist.read_data_sets() 。因此 images 的真身藏在 mnist.py 这个文件里。

mnist.py 并不难找，在Pycharm里按下 ctrl 后单击鼠标左键即可打开源代码。

继续追踪，可以在mnist里发现图像来自 extract_images() 函数。该函数的说明里清晰的写明：

```
Extract the images into a 4D uint8 numpy array [index, y, x, depth].
  Args:
    f: A file object that can be passed into a gzip reader.
  Returns:
    data: A 4D uint8 numpy array [index, y, x, depth].
  Raises:
    ValueError: If the bytestream does not start with 2051.
```

很明显，返回值变量名为 data ，是一个**4D Numpy矩阵**，存储值为 uint8 类型，即图像像素的灰度

数，每个图像通道数。

在获得这个存储着像素灰度值的Numpy矩阵后，使用numpy的 tostring() 函数将其转换为Python bytes格式[^4]，再使用 tf.train.BytesList() 函数封装为 tf.train.BytesList 类，名字为 image_raw 。最后使用 tf.train.Example() 将 image_raw 和其它属性一遍打包，并调用 tf.python_io.TFRecordWriter 将其写入到文件中。

至此，TFRecord文件生成完毕。

可见，**将自定义图像转换为TFRecord的过程本质上是将大量图像的像素灰度值转换为Python bytes，并与其它 Feature 组合在一起，最终拼接成一个文件的过程**。

需要注意的是其它 Feature 的类型不一定必须是BytesList，还可以是Int64List或者FloatList。

读取TFRecord文件

第二份代码 fully_connected_reader.py [1]展示了如何从TFRecord文件中读取数据。

读取数据的函数名为 input() 。函数内部首先通过 tf.train.string_input_producer() 函数读取TFRecord文件，并返回一个 queue ；然后使用 read_and_decode() 读取一份数据，函数内部用 tf.decode_raw() 解析出图像的灰度值，用 tf.cast() 解析出 label 的值。之后通过 tf.train.shuffle_batch() 的方法生成一批用来训练的数据。并最终返回可供训练的 images 和 labels ，并送入 inference 部分进行计算。

1. tf.decode_raw() 解析出的数据是没有 shape 的，因此需要调用 set_shape() 函数来给出tensor的维度。

2. read_and_decode() 函数返回的是单个的数据，但是后边的 tf.train.shuffle_batch() 却能够生成批量数据。
3. 如果需要对图像进行处理的话，需要放在第二项提到的两个函数中间。

其中第2点的原理我暂时没有弄懂。从代码上看 read_and_decode() 返回的是单个数据， shuffle_batch 接收到的也是单个数据，不知道是如何生成批量数据的，猜测与 queue 有关系。

所以，**读取TFRecord文件的本质，就是通过队列的方式依次将数据解码，并按需要进行数据随机化、图像随机化的过程**。

参考

1 Github: fully_connected_reader.py ↩

Yin Guobing

BOE技术研发工程师 ，业余码农，蓝猫铲屎官 。曾独立开发了一款iOS APP并上线 。现居北京，正在为了理想中的生活而奋斗..

Read More

— 尹国冰的博客 —

TensorFlow

基于深度学习的人脸特征点检测 - 生成TFRecord文件

深度学习训练图像太少怎么办

深度学习框架TensorFlow

See all 3 posts →

C++

在Eclipse下编译Dlib

Dlib是一款非常优秀的C++库，尤其是其中的人面部landmark点检测功能，可以在2ms左右的时间检测出面部的68个标记点。

Dlib官方推荐采用CMake的方式来编译，本文描述了在Eclipse下引入Dlib的具体方法。

YIN GUOBING

Dlib编译错误Converting std::__cxx11::string to std::string

在编译Dlib的时候总是无法通过，报错信息包含`Converting std::__cxx11::string to std::string`。引发错误的原因居然是Anaconda。

YIN GUOBING