



Stay Hungry. Stay Foolish.

TensorFlow 的 TFRecord 和 QueueRunner 简介

📁 TensorFlow | 💬 6 | 👁 1420

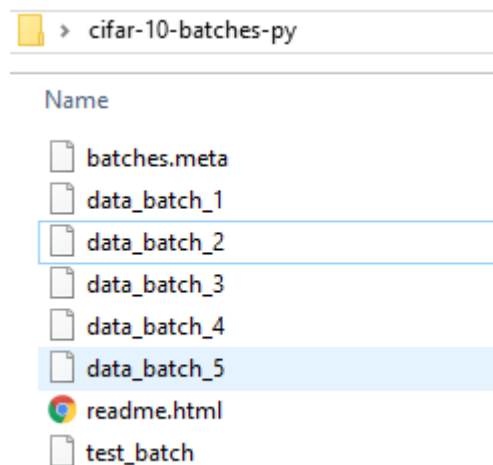
通常我们下载的数据集都是以压缩文件的格式存在，解压后会有多个文件夹，像 `train`，`test`，`val` 等等。而文件也有可能多达数万或者数百万个。这种形式的数据集不但读取复杂、慢，而且占用磁盘空间。这时二进制的格式文件的优点便显现出来了。我们可以把数据集存储为一个**二进制文件**，这样就没有了 `train`，`test`，`val` 等等的文件夹。更重要的是，这些数据只会占据一块内存（Block of Memory），而不需要一个一个单独加载文件。因此使用**二进制文件**效率更高。

你以为 TensorFlow 都为你封装好二进制文件文件的读写、解析方式了吗？是的，都封装好了~本文就是介绍如何将数据转换为 TFRecord 格式。

CIFAR-10 数据集

本文以 CIFAR-10 数据集为例，什么是 CIFAR-10 数据集？看这儿 => [图像数据集](#) ~

假设你已经有了以下数据：



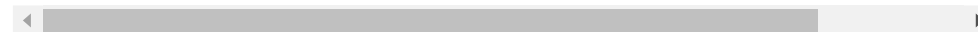
写，保存为 TFRecord 格式

定义的一些常量：

```
_NUM_TRAIN_FILES = 5
# The height and width of each image.
_IMAGE_SIZE = 32
# The names of the classes.
_CLASS_NAMES = [
    'airplane',
    'automobile',
    'bird',
    'cat',
    'deer',
    'dog',
    'frog',
    'horse',
    'ship',
    'truck',
]
```

这里我们创建两个 split 文件，分别存储 train 和 test 需要的数据：

```
dataset_dir = 'data'
if not tf.gfile.Exists(dataset_dir):
    tf.gfile.MakeDirs(dataset_dir)
training_filename = _get_output_filename(dataset_dir,
testing_filename = _get_output_filename(dataset_dir,
```



`_get_output_filename` 函数用来生成文件名：

```
def _get_output_filename(dataset_dir, split_name):
    """Creates the output filename.
    Args:
        dataset_dir: The dataset directory where the data is stored.
        split_name: The name of the train/test split.
    Returns:
        An absolute file path.
    """
    return '%s/cifar10_%s.tfrecord' % (dataset_dir, split_name)
```

然后，处理训练数据：

```
# First, process the training data:
with tf.python_io.TFRecordWriter(training_filename) as writer:
    offset = 0
    for i in range(_NUM_TRAIN_FILES):
        filename = os.path.join('./cifar-10-batches-py',
                                'data_batch_%05d' % (i + 1))
        offset = _add_to_tfrecord(filename, tfrecord_iterator(filename,
                                                                reader_queue))
```

即依次读取 `data_batch_?` 文件，调用 `_add_to_tfrecord` 将其保存为 TFRecord 格式。

```

def _add_to_tfrecord(filename, tfrecord_writer, offset):
    """Loads data from the cifar10 pickle files and writes it to the TFRecord.
    Args:
        filename: The filename of the cifar10 pickle file.
        tfrecord_writer: The TFRecord writer to use for writing.
        offset: An offset into the absolute number of images.
    Returns:
        The new offset.
    """
    with tf.gfile.Open(filename, 'rb') as f:
        data = pickle.load(f, encoding='bytes')
        images = data[b'data']
        num_images = images.shape[0]

    images = images.reshape((num_images, 3, 32, 32))
    labels = data[b'labels']
    with tf.Graph().as_default():
        image_placeholder = tf.placeholder(tf.uint8)
        encoded_image = tf.image.encode_png(image_placeholder)
        with tf.Session() as sess:
            for j in range(num_images):
                sys.stdout.write('\r>> Reading file [%d]' % j)
                sys.stdout.flush()
                image = np.squeeze(images[j]).transpose(2, 1, 0)
                label = labels[j]
                png_string = sess.run(encoded_image, {image_placeholder: image})
                example = image_to_tfexample(png_string, label)
                tfrecord_writer.write(example.SerializeToString())
    return offset + num_images

```

```
tfrecord_writer.write(example.SerializeFromString(encoded_image))  
return offset + num_images
```

因为 CIFAR-10 数据集的图片是 `10000x3072 numpy array` 格式的，因此需要 `reshape` 为 `tf.image.encode_png` 需要的格式：`[height, width, channels]`。`tf.image.encode_png` 返回编码后的字符串，然后还需要保存图片的宽高、格式信息。调用 `image_to_tfexample` 将这些数据保存到 `tf.train.Example` 中：

```
def image_to_tfexample(image_data, image_format, height, width):  
    return tf.train.Example(features=tf.train.Features({  
        'image/encoded': tf.train.BytesList(value=[image_data]),  
        'image/format': tf.train.BytesList(value=[image_format]),  
        'image/class/label': tf.train.Int64List(value=[label]),  
        'image/height': tf.train.Int64List(value=[height]),  
        'image/width': tf.train.Int64List(value=[width])  
    }))
```

TensorFlow 会将数据转换为 `tf.train.Example` Protobuf 对象，`Example` 包含 `Features`，`Features` 包含一个 `dict`，来区分不同的 `Feature`。`Feature` 可以包含 `FloatList`，`ByteList` 或者 `Int64List`。注意这里的 key，`image/encoded`，`image/format` 等，是可以随便定义的，这里是 TensorFlow 默认的图片数据集的 key，我们一般采取 TensorFlow 默认的值。

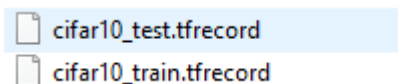
有了 `example`，我们将其转换为字符串写入到文件就完成了整个 TFRecord 格式文件的制作。

```
tfrecord_writer.write(example.SerializeToString())
```

同理，制作测试数据集：

```
# Next, process the testing data:  
with tf.python_io.TFRecordWriter(testing_filename) as  
    filename = os.path.join('./cifar-10-batches-py',  
        _add_to_tfrecord(filename, tfrecord_writer)
```

最后，我们会得到两个文件：



cifar10_test.tfrecord
cifar10_train.tfrecord

这就是最后的 TFRecord 格式文件，二进制文件。

读

最简单的就是直接读取：

```

reconstructed_images = []
record_iterator = tf.python_io.tf_record_iterator(pat
for string_iterator in record_iterator:
    example = tf.train.Example()
    example.ParseFromString(string_iterator)
    height = example.features.feature['image/height'].i
    width = example.features.feature['image/width'].i
    png_string = example.features.feature['image/encod
    label = example.features.feature['image/class/label
    with tf.Session() as sess:
        image_placeholder = tf.placeholder(dtype=tf.s
        decoded_img = tf.image.decode_png(image_place
        reconstructed_img = sess.run(decoded_img, feed
        reconstructed_images.append((reconstructed_img, l

```

其实就是“写”的逆过程。生成一个 `Example`，分析读取的字符串，然后从 `features` 中根据 key 获取相应的对象即可。图片的话我们使用 `tf.image.decode_png` 解码，即 `tf.image.encode_png` 逆过程。

读取后可以直接来显示：

```

plt.imshow(reconstructed_images[0][0])
plt.title(_CLASS_NAMES[reconstructed_images[0][1]])
plt.show()

```


这种方法比较直接，直接从文件读取并分析，但是如果数据较多就会你比较慢，而且没有考虑分布式、队列、多线程的问题。我们还可以使用文件队列来读取。

队列

```
# first construct a queue containing a list of filenames
# this lets a user split up there dataset in multiple
# size down
filename_queue = tf.train.string_input_producer(['./c
# Unlike the TFRecordWriter, the TFRecordReader is sync
reader = tf.TFRecordReader()
_, serialized_example = reader.read(filename_queue)
features = tf.parse_single_example(serialized_example,
    {'image/encoded': tf.FixedLenFeature([], tf.string),
     'image/format': tf.FixedLenFeature([], tf.string),
     'image/height': tf.FixedLenFeature([], tf.int64),
     'image/width': tf.FixedLenFeature([], tf.int64),
     'image/class/label': tf.FixedLenFeature([], tf.int64)},
    {}))
image = tf.image.decode_png(features['image/encoded'],
image = tf.image.resize_image_with_crop_or_pad(image,
label = features['image/class/label']
init_op = tf.group(tf.global_variables_initializer(),
with tf.Session() as sess:
    sess.run(init_op)
    tf.train.start_queue_runners()
    # grab examples back.
    # first example from file
    image_val_1, label_val_1 = sess.run([image, label])
    # second example from file
    image_val_2, label_val_2 = sess.run([image, label])
    print(image_val_1, label_val_1)
    print(image_val_2, label_val_2)
```

```
plt.imshow(image_val_1)
plt.title(_CLASS_NAMES[label_val_1])
plt.show()
```

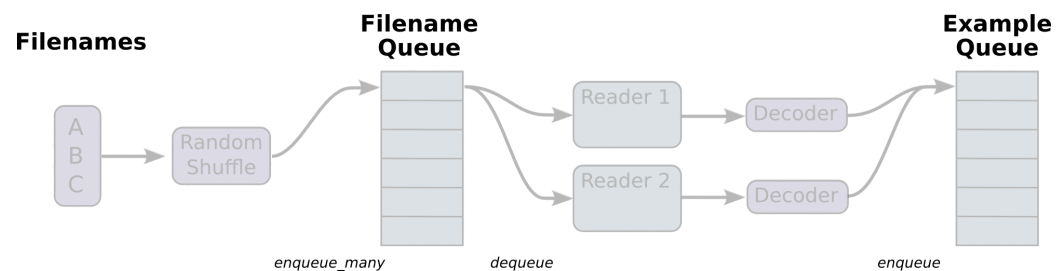
首先定义我们的文件名队列 `filename_queue`，包含一个文件名列表，这样我们可以把大的文件分成多个小的文件，保证单个文件不会太大，本例只有一个文件。然后用 `TFRecordReader` 读取。TensorFlow 的 `graphs` 包含一些状态变量，允许 `TFRecordReader` 记住 `tfrecord` 读到了哪儿，下次从哪儿读起，因此我们需要 `sess.run(init_op)` 来初始化这些状态。与 `tf.python_io.tf_record_iterator` 不同的是 `TFRecordReader` 总是作用在文件名（`filename_queue`）队列上，它会弹出一个文件名读取数据，直到 `tfrecord` 为空，然后读取下一个文件名对应的文件。

如何生成文件名队列呢，这时我们需要 `QueueRunners` 来做。`QueueRunners` 其实就是一个线程，使用 `session` 不断执行入队操作，TensorFlow 已经封装好了 `tf.train.QueueRunner` 对象。但是大部分时间 `QueueRunner` 只是底层操作，我们不会直接操作它，本例使用 `tf.train.string_input_producer` 生成。

此时，需要发送信号让 TensorFlow 开起线程，执行 `QueueRunners`，否则，代码将会永远阻塞，等待数据入队。因此需要执行 `tf.train.start_queue_runners()`，此行代码执行完会立即创建线程。注意，必须在 `initialization 运算符（sess.run(init_op)）` 执行之后调用。

`tf.parse_single_example` 根据我们定义的 `features` 数据格式解析。最终，`image_val_1` 就是图片数据集中的一张图片，shape 为 `(32, 32, 3)`。


队列读取的流程图如下：



Batch

上例中，我们获得的 `image` 和 `label` 都是单个的 `Example` 对象，代表数据集中的一条数据。训练的时候不可能单条数据训练，如何生成 batches？

```
images_batch, labels_batch = tf.train.shuffle_batch(  
    [image, label], batch_size=128,  
    capacity=2000,  
    min_after_dequeue=1000)  
  
with tf.Session() as sess:  
    sess.run(init_op)  
    tf.train.start_queue_runners()  
    labels, images = sess.run([labels_batch, images_t  
    print(labels.shape)
```



这里我们使用 `tf.train.shuffle_batch` 将单个的 `image` 和 `label` Example 对象生成 batches。`tf.train.shuffle_batch` 实际上构建了另一种

`QueueRunner`，`RandomShuffleQueue`。`RandomShuffleQueue` 将单个的 `image` 和 `label` 累积成队列，直到包含 `batch_size + min_after_dequeue` 个。然后随机选择 `batch_size` 条数据返回，因此 `shuffle_batch` 的返回值实际上是 `RandomShuffleQueue` 执行 `dequeue_many` 的返回值。

如果 tensor 的形状为 `[x, y, z]`，`shuffle_batch` 返回的对应的 tensor 形状为 `[batch_size, x, y, z]`，本例 `labels` 和 `images` 形状分别为 `(128,)` 和 `(128, 32, 32, 3)`。

DatasetDataProvider

如果我们使用 `tf.contrib.slim` , 我们可以将读取过程封装的更优雅。

定义我们的数据集 `cifar10.py` , 具体怎么定义相信看了以上代码, 下面的代码不用解释也能看懂了吧~

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import tensorflow as tf

slim = tf.contrib.slim

_FILE_PATTERN = 'cifar10_%s.tfrecord'

SPLITS_TO_SIZES = {'train': 50000, 'test': 10000}

_NUM_CLASSES = 10

_ITEMS_TO_DESCRIPTIONS = {
    'image': 'A [32 x 32 x 3] color image.',
    'label': 'A single integer between 0 and 9',
}

def get_split(split_name, dataset_dir, file_pattern=None, reader=None):
    """Gets a dataset tuple with instructions for reading.

    Args:
        split_name: A train/test split name.
        dataset_dir: The base directory of the dataset.
        file_pattern: The file pattern to use when matching files.
        reader: A reader to use.

    It is assumed that the pattern contains a '%s'
    name can be inserted.
    """
```

```

    reader: The TensorFlow reader type.
Returns:
    A `Dataset` namedtuple.
Raises:
    ValueError: if `split_name` is not a valid tra
"""
if split_name not in SPLITS_TO_SIZES:
    raise ValueError('split name %s was not recog

if not file_pattern:
    file_pattern = _FILE_PATTERN
file_pattern = os.path.join(dataset_dir, file_pat

# Allowing None in the signature so that dataset_
if not reader:
    reader = tf.TFRecordReader

keys_to_features = {
    'image/encoded': tf.FixedLenFeature(), tf.st
    'image/format': tf.FixedLenFeature(), tf.str
    'image/class/label': tf.FixedLenFeature(
        [], tf.int64, default_value=tf.zeros([],

}

items_to_handlers = {
    'image': slim.tfexample_decoder.Image(shape=[
    'label': slim.tfexample_decoder.Tensor('image
}

```



```
decoder = slim.tfexample_decoder.TFExampleDecoder(
    keys_to_features, items_to_handlers)

labels_to_names = None
if has_labels(dataset_dir):
    labels_to_names = read_label_file(dataset_dir)

return slim.dataset.Dataset(
    data_sources=file_pattern,
    reader=reader,
    decoder=decoder,
    num_samples=SPLITS_TO_SIZES[split_name],
    items_to_descriptions=_ITEMS_TO_DESCRIPTIONS,
    num_classes=_NUM_CLASSES,
    labels_to_names=labels_to_names)

def has_labels(dataset_dir, filename='labels.txt'):
    """Specifies whether or not the dataset directory
    Args:
        dataset_dir: The directory in which the labels
        filename: The filename where the class names are
    Returns:
        `True` if the labels file exists and `False` otherwise
    """
    return tf.gfile.Exists(os.path.join(dataset_dir,

def read_label_file(dataset_dir, filename='labels.txt'):
    """Reads the labels file and returns a mapping from
    Args:
```

```
dataset_dir: The directory in which the labels
filename: The filename where the class names are
Returns:
    A map from a label (integer) to class name.
"""
labels_filename = os.path.join(dataset_dir, filename)
with tf.gfile.Open(labels_filename, 'rb') as f:
    lines = f.read().decode()
lines = lines.split('\n')
lines = filter(None, lines)

labels_to_class_names = {}
for line in lines:
    index = line.index(':')
    labels_to_class_names[int(line[:index])] = line[index+1:]
return labels_to_class_names
```

读取的话就非常简单了：

```
dataset = cifar10.get_split('train', DATA_DIR)
provider = slim.dataset_data_provider.DatasetDataProvider
[image, label] = provider.get(['image', 'label'])
```

以上所有代码都可以在 tensorflow/model 仓库下的 [slim](#) 中找到~

总结

流程：

1. 生成 TFRecord 格式文件
2. 定义 record reader 分析 TFRecord 文件
3. 定义 batcher
4. 构建网络模型
5. 初始化所有运算符
6. 开始 queue runners.
7. 训练 loop

参考

1. [Tfrecords Guide](#)
2. [TensorFlow Data Input \(Part 1\): Placeholders, Protobufs & Queues](#)
3. [关于tensorflow 的数据读取线程管理QueueRunner](#)

-- END

写的不错，帮助到了您，赞助一下主机费~

赞赏

版权声明: ©自由转载-非商用-非衍生-保持署名 ([创意共享3.0许可证](#))

创建日期: 2017年10月31日

修改日期: 2017年12月13日

文章标签:

Machine Learning

TensorFlow

推荐阅读:



[TensorFlow 和 NumPy 的 Broadcasting...](#) 2017-10-27



lkq

2017-11-04 17:45

#1

支持一下

回复



lkq

2017-11-04 17:45

#2

this is test

回复



ShuaiZhao

2017-12-12 19:42

#3

```
plt.imshow(reconstructed_images[0][0])  
plt.title(_CLASS_NAMES[reconstructed_images[0][0]])
```

第二个应该是

```
plt.title(_CLASS_NAMES[reconstructed_images[0][0]])
```

吧

再有就是不是很理解

```
images = images.reshape((num_images, 3, 32, 32))
```

为什么不直接reshape为(num_images, 32, 32, 3)呢？后面也不用装置。测试了一下发现是可行的。

最后谢谢您的教程了，棒！

回复



lufficc 博主

2017-12-13 13:06

#4

的确应该为

```
plt.title(_CLASS_NAMES[reconstructed_images[0]  
[1]]), 已修改, 多谢~~
```

CIFAR-10 存储的图片数据格式为：

10000x3072 numpy array，每一行是一个 32x32 彩色图片。每 1024 个元素依次代表 R G B，像素值没有归一化（取值为 0-255）。

直接reshape为(num_images, 32, 32, 3)，怕是把图片结构破坏了

回复



ShuaiZhao

2017-12-15 16:35

#5

@lufficc

谢谢~~.

是的。我的实验确实有些问题。后来仔细去看了确实和博主所说一样。numpy的维度机制和matlab不一样，用matlab的理解带入了。

贴上测试

```
a = [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]  
b = reshape(a, [2, 3, 3])
```

```
b(:, :, 1) =
```

1	3	5
2	4	6

```
b(:, :, 2) =
```

1	3	5
2	4	6

```
b(:, :, 3) =
```

1	3	5
2	4	6