**Machine Learning Guru (../../../../index.php)**

Machine Learning and Computer Vision tutorials using open source packages.

**Sections**



# How to write into and read from a TFRecords file in TensorFlow

In this post, you will learn how to save a large amount of data (images) into a single TFRecords format file and load it batch-wise to train your network in tensorflow.

## Introduction

In the previous post (../hdf5/hdf5.html) we explained the benefits of saving a large dataset in a single HDF5 file. In this post we will learn how to convert our data into the Tensorflow standard format, called TFRecords. When we are training a deep network, we have two options to feed the data into out Tensorflow program: loading the data using pure python code at each step and feed it into a computaion graph or use an input pipeline which takes a list of filenames (any supported format), shuffle them (optional), create a file queue, read, and decode the data. However, TFRecords is the recommended file format for Tensorflow.

In this post, we load, resize and save all the images inside the train folder of the well-known Dogs vs. Cats (https://www.kaggle.com/c/dogs-vs-cats) data set into a single TFRecords file and then load and plot a couple of them as samples. To follow the rest of this post you need to download the train part of the Dogs vs. Cats data set.

## List images and their labels

First, we need to list all images and label them. We give each cat image a label = 0 and each dog image a label = 1. The following code list all images, give them proper labels, and then shuffle the data. We also divide the data set into three train (%60), validation (%20), and test parts (%20).

List images and label them

```
from random import shuffle
import glob
shuffle_data = True  # shuffle the addresses before saving
cat_dog_train_path = 'Cat vs Dog/train/*.jpg'

# read addresses and labels from the 'train' folder
addrs = glob.glob(cat_dog_train_path)
labels = [0 if 'cat' in addr else 1 for addr in addrs]  # 0 = Cat, 1 = Dog

# to shuffle data
if shuffle_data:
    c = list(zip(addrs, labels))
    shuffle(c)
    addrs, labels = zip(*c)

# Divide the hata into 60% train, 20% validation, and 20% test
train_addrs = addrs[0:int(0.6*len(addrs))]
train_labels = labels[0:int(0.6*len(labels))]

val_addrs = addrs[int(0.6*len(addrs)):int(0.8*len(addrs))]
val_labels = labels[int(0.6*len(addrs)):int(0.8*len(addrs))]

test_addrs = addrs[int(0.8*len(addrs)):]
test_labels = labels[int(0.8*len(labels)):]
```

## Create a TFRecords file

First we need to load the image and convert it to the data type (float32 in this example) in which we want to save the data into a TFRecords file. Let's write a function which take an image address, load, resize, and return the image in proper data type:

A function to Load images

```
def load_image(addr):
    # read an image and resize to (224, 224)
    # cv2 load images as BGR, convert it to RGB
    img = cv2.imread(addr)
    img = cv2.resize(img, (224, 224), interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)
    return img
```

Before we can store the data into a TFRecords file, we should stuff it in a protocol buffer called **Example**. Then, we serialize the protocol buffer to a string and write it to a TFRecords file. Example protocol buffer contains **Features**. Feature is a protocol to describe the data and could have three types: bytes, float, and int64. In summary, to store your data you need to follow these steps:

- Open a TFRecords file using `tf.python_io.TFRecordWriter`
- Convert your data into the proper data type of the feature using `tf.train.Int64List`, `tf.train.BytesList`, or `tf.train.FloatList`
- Create a feature using `tf.train.Feature` and pass the converted data to it
- Create an Example protocol buffer using `tf.train.Example` and pass the feature to it
- Serialize the Example to string using `example.SerializeToString()`
- Write the serialized example to TFRecords file using `writer.write`

We are going to use the following two functions to create features (Functions are from this Tensorflow Tutorial (https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/how_tos/reading_data/convert_to_records.py))

Convert data to features

```
def _int64_feature(value):
  return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))


def _bytes_feature(value):
  return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))
```

Now let's store the training data to the TFRecords file.

Write data into a TFRecords file

```
train_filename = 'train.tfrecords'  # address to save the TFRecords file

# open the TFRecords file
writer = tf.python_io.TFRecordWriter(train_filename)

for i in range(len(train_addrs)):
    # print how many images are saved every 1000 images
    if not i % 1000:
        print 'Train data: {}/{}'.format(i, len(train_addrs))
        sys.stdout.flush()

    # Load the image
    img = load_image(train_addrs[i])

    label = train_labels[i]

    # Create a feature
    feature = {'train/label': _int64_feature(label),
               'train/image': _bytes_feature(tf.compat.as_bytes(img.tostring()))}

    # Create an example protocol buffer
```

```
    example = tf.train.Example(features=tf.train.Features(feature=feature))

    # Serialize to string and write on the file
    writer.write(example.SerializeToString())

writer.close()
sys.stdout.flush()
```

and finaly we close the file using: `writer.close()`. Similarly we write the validation and test data to two other TFRecords files.

Write validation and test data into a TFRecords file

```
# open the TFRecords file
val_filename = 'val.tfrecords'  # address to save the TFRecords file
writer = tf.python_io.TFRecordWriter(val_filename)

for i in range(len(val_addrs)):
    # print how many images are saved every 1000 images
    if not i % 1000:
        print 'Val data: {}/{}'.format(i, len(val_addrs))
        sys.stdout.flush()

    # Load the image
    img = load_image(val_addrs[i])

    label = val_labels[i]

    # Create a feature
    feature = {'val/label': _int64_feature(label),
               'val/image': _bytes_feature(tf.compat.as_bytes(img.tostring()))}

    # Create an example protocol buffer
    example = tf.train.Example(features=tf.train.Features(feature=feature))

    # Serialize to string and write on the file
    writer.write(example.SerializeToString())

writer.close()
sys.stdout.flush()

# open the TFRecords file
test_filename = 'test.tfrecords'  # address to save the TFRecords file
writer = tf.python_io.TFRecordWriter(test_filename)

for i in range(len(test_addrs)):
    # print how many images are saved every 1000 images
    if not i % 1000:
        print 'Test data: {}/{}'.format(i, len(test_addrs))
        sys.stdout.flush()

    # Load the image
    img = load_image(test_addrs[i])

    label = test_labels[i]

    # Create a feature
    feature = {'test/label': _int64_feature(label),
               'test/image': _bytes_feature(tf.compat.as_bytes(img.tostring()))}

    # Create an example protocol buffer
    example = tf.train.Example(features=tf.train.Features(feature=feature))

    # Serialize to string and write on the file
    writer.write(example.SerializeToString())

writer.close()
sys.stdout.flush()
```

## Read the TFRecords file

It's time to learn how to read data from the TFRecords file. To do so, we load the data from the train data in batchs of an arbitrary size and plot images of the 5 batchs. We also check the label of each image. To read from files in tensorflow, you need to do the following steps:

- **Create a list of filenames:** In our case we only have a single file `data_path = 'train.tfrecords'`. Therefore, our list is gonna be like this: `[data_path]`
- **Create a queue to hold filenames:** To do so, we use `tf.train.string_input_producer` tf.train.string_input_producer function which hold filenames in a FIFO queue. it gets the list of filnames. It also has some optional arguments including `num_epochs` which indicates the number of

epoch you want to to load the data and `shuffle` which indicates whether to suffle the filenames in the list or not. It is set to **True** by default.

- **Define a reader:** For files of TFRecords we need to define a TFRecordReader with `reader = tf.TFRecordReader()`. Now, the reader returns the next record using: `reader.read(filename_queue)`
- **Define a decoder:** A decoder is needed to decode the record read by the reader. In case of using TFRecords files the decoder should be `tf.parse_single_example`. it takes a serialized Example and a dictionary which maps feature keys to FixedLenFeature or VarLenFeature values and returns a dictionary which maps feature keys to Tensor values: `features = tf.parse_single_example(serialized_example, features=feature)`
- **Convert the data from string back to the numbers:** `tf.decode_raw(bytes, out_type)` takes a Tensor of type string and convert it to type `out_type`. However, for labels which have not been converted to string, we just need to cast them using `tf.cast(x, dtype)`
- **Reshape data into its original shape:** You should reshape the data (image) into it's original shape before serialization using `image = tf.reshape(image, [224, 224, 3])`
- **Preprocessing:** if you want to do any preprocessing you should do it now.
- **Batching:** Another queue is needed to create batches from the examples. You can create the batch queue using `tf.train.shuffle_batch([image, label], batch_size=10, capacity=30, num_threads=1, min_after_dequeue=10)` where `capacity` is the maximum size of queue, `min_after_dequeue` is the minimum size of queue after dequeue, and `num_threads` is the number of threads enqueuing examples. Using more than one thread, it comes up with a faster reading. The first argument in a list of tensors which you want to create batches from.

Read a TFRecords file

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

data_path = 'train.tfrecords'  # address to save the hdf5 file

with tf.Session() as sess:
    feature = {'train/image': tf.FixedLenFeature([], tf.string),
               'train/label': tf.FixedLenFeature([], tf.int64)}

    # Create a list of filenames and pass it to a queue
    filename_queue = tf.train.string_input_producer([data_path], num_epochs=1)

    # Define a reader and read the next record
    reader = tf.TFRecordReader()
    _, serialized_example = reader.read(filename_queue)

    # Decode the record read by the reader
    features = tf.parse_single_example(serialized_example, features=feature)

    # Convert the image data from string back to the numbers
    image = tf.decode_raw(features['train/image'], tf.float32)

    # Cast label data into int32
    label = tf.cast(features['train/label'], tf.int32)

    # Reshape image data into the original shape
    image = tf.reshape(image, [224, 224, 3])

    # Any preprocessing here ...

    # Creates batches by randomly shuffling tensors
    images, labels = tf.train.shuffle_batch([image, label], batch_size=10, capacity=30, num_threads=
```

- **Initialize all global and local variables**
- **Filing the example queue:** Some functions of `tf.train` such as `tf.train.shuffle_batch` add `tf.train.QueueRunner` objects to your graph. Each of these objects hold a list of enqueue op for a queue to run in a thread. Therefore, to fill a queue you need to call `tf.train.start_queue_runners` which starts threades for all the queue runners in the graph. However, to manage these threads you need a `tf.train.Coordinator` to terminate the threads at the proper time.

Everything is ready. Now you can read a batch and plot all batch images and labels. Do not forget to stop the threads (by stopping the cordinator) when you are done with your reading process.

Read a TFRecords file

```
    # Initialize all global and local variables
    init_op = tf.group(tf.global_variables_initializer(), tf.local_variables_initializer())
    sess.run(init_op)

    # Create a coordinator and run all QueueRunner objects
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)

    for batch_index in range(5):
        img, lbl = sess.run([images, labels])

        img = img.astype(np.uint8)

        for j in range(6):
            plt.subplot(2, 3, j+1)
            plt.imshow(img[j, ...])
            plt.title('cat' if lbl[j]==0 else 'dog')

        plt.show()

    # Stop the threads
    coord.request_stop()

    # Wait for threads to stop
    coord.join(threads)
    sess.close()
```

You can download the codes of this post in our Github (TFRecords.html) page.

Go Top

**Related Posts:**