CycleGAN 图像风格迁移

这个文档里会列出代码里最重要的一些部分进行讲解，全部代码分为四个文件：

- `build_data.py` 将图片数据集预处理tfrecord的格式，为了方便读取，统一输入方式。
- `cyclegan.py` 定义模型的计算和训练。
- `layers.py` 一些基础计算操作的定义（例如卷积层、线性变换等）
- `reader.py` 定义如何从tfrecord中读取数据输入到模型

代码着重讲解`cyclegan.py`中的这两个部分两部分: (1)计算图构建 `build_model`; (2)模型训练 `train`。

计算图构建

判别器D

判别器使用5层卷积，都是 4x4 的卷积，其中前4层步长为2并且连接batch normalization 和 leaky ReLU，最后一层是步长为1的卷积。

```
def D(_input, name="D", is_training=True):
  with tf.variable_scope(name, reuse=tf.AUTO_REUSE):
    c64 = layers.c4s2_k(_input, 64, "c64")
    c128 = layers.c4s2_k(c64, 128, "c128")
    c256 = layers.c4s2_k(c128, 256, "c256")
    c512 = layers.c4s2_k(c256, 512, "c512")
    out = layers._conv(c512, 1, "out", stride=1)
  return out
```

生成器G

这里的G与DCGAN中的G不同，不是从噪音生成图片，而是将一个图片作为输入，生成不同风格的另一个图片。

使用三层卷积之后，连接多层ResNet（残差网络），然后再连接两层反卷积和一层卷积：

```
def G(_input, name="G", is_training=True):
  with tf.variable_scope(name, reuse=tf.AUTO_REUSE):
    c7s1_64 = layers.c7s1_k(_input, 64, is_training=is_training,
name='c7s1_64')
    c3s2_128 = layers.c3s2_k(c7s1_64, 128, is_training=is_training,
name='c3s2_128')
    c3s2_256 = layers.c3s2_k(c3s2_128, 128, is_training=is_training,
name='c3s2_256')
    res = layers.n_res_blocks(c3s2_256, is_training=is_training, n=6)
    dc3s2_128 = layers.dc3s2_k(res, 128, is_training=is_training,
name="dc3s2_128")
    dc3s2_64 = layers.dc3s2_k(dc3s2_128, 64, is_training=is_training,
```

```
name="dc3s2_64")
    out = layers.c7s1_k(dc3s2_64, 3, activation="tanh", name="output")
    return out
```

CycleGAN模型构建

输入节点使用reader.py提供好的输入节点：

```
X_reader = Reader(hp.X, name='X',
  image_size=hp.image_size, batch_size=hp.batch_size)
Y_reader = Reader(hp.Y, name='Y',
  image_size=hp.image_size, batch_size=hp.batch_size)
x = X_reader.feed()
y = Y_reader.feed()
```

定义生成器和判别器，注意模型中有两对生成器和判别器，因此需要用不用的name加以区分：

```
_G = partial(G, name="G", is_training=is_training)
_F = partial(G, name="F", is_training=is_training)
_D_X = partial(D, name="D_X", is_training=is_training)
_D_Y = partial(D, name="D_Y", is_training=is_training)
```

计算目标函数值：

```
cycle_loss = cycle_consistency_loss(_G, _F, x, y, hp)

# X -> Y
fake_y = _G(x)
G_gan_loss = generator_loss(_D_Y, fake_y)
G_loss =  G_gan_loss + cycle_loss
D_Y_loss = discriminator_loss(_D_Y, y, fake_y)

# Y -> X
fake_x = _F(y)
F_gan_loss = generator_loss(_D_X, fake_x)
F_loss = F_gan_loss + cycle_loss
D_X_loss = discriminator_loss(_D_X, x, fake_x)
```

构建摘要节点:

```
tf.summary.scalar('loss/G', G_gan_loss)
tf.summary.scalar('loss/D_Y', D_Y_loss)
tf.summary.scalar('loss/F', F_gan_loss)
tf.summary.scalar('loss/D_X', D_X_loss)
```

```
tf.summary.scalar('loss/cycle', cycle_loss)

tf.summary.image('X/generated', batch_convert2int(_G(x)))
tf.summary.image('X/reconstruction', batch_convert2int(_F(_G(x))))
tf.summary.image('Y/generated', batch_convert2int(_F(y)))
tf.summary.image('Y/reconstruction', batch_convert2int(_G(_F(y))))
```

最后返回更新操作：`return optimize(G_loss, D_Y_loss, F_loss, D_X_loss, hp)` optimize函数定义如下：

```
def optimize(G_loss, D_Y_loss, F_loss, D_X_loss, hp):
  # 根据目标函数（loss），  参数来获取参数的更新操作
  def make_optimizer(loss, variables, name='Adam'):
    """ Adam optimizer with learning rate 0.0002 for the first 100k steps
(~100 epochs)
        and a linearly decaying rate that goes to zero over the next 100k
steps
    """
    global_step = tf.Variable(0, trainable=False)
    starter_learning_rate = hp.learning_rate
    end_learning_rate = 0.0
    start_decay_step = 100000
    decay_steps = 100000
    beta1 = hp.beta1
    learning_rate = (
      tf.where(
        tf.greater_equal(global_step, start_decay_step),
        tf.train.polynomial_decay(
          starter_learning_rate,
          global_step-start_decay_step,
          decay_steps, end_learning_rate,
          power=1.0),
        starter_learning_rate
      ))
    tf.summary.scalar('learning_rate/{}'.format(name), learning_rate)

    learning_step = (
        tf.train.AdamOptimizer(learning_rate, beta1=beta1, name=name)
                .minimize(loss, global_step=global_step,
var_list=variables)
    )
    return learning_step

  # 获取两个判别器和两个生成器的参数
  G_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope="G")
  F_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope="F")
  DX_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
scope="D_X")
  DY_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
scope="D_Y")
```

```
  G_optimizer = make_optimizer(G_loss, G_vars, name='Adam_G')
  D_Y_optimizer = make_optimizer(D_Y_loss, DY_vars, name='Adam_D_Y')
  F_optimizer =  make_optimizer(F_loss, F_vars, name='Adam_F')
  D_X_optimizer = make_optimizer(D_X_loss, DX_vars, name='Adam_D_X')

  # 将所有更新节点合并为一个新节点，方便计算
  with tf.control_dependencies([G_optimizer, D_Y_optimizer, F_optimizer,
D_X_optimizer]):
    return tf.no_op(name='optimizers')
```

模型训练

模型训练与DCGAN类似，也是不断迭代地执行（sess.run）一个参数优化的操作optim_op，然后写入摘要。

```
while not coord.should_stop():
  _, summary = sess.run([optim_op, summary_op])
  writer.add_summary(summary, step)
  writer.flush()
  step += 1
```