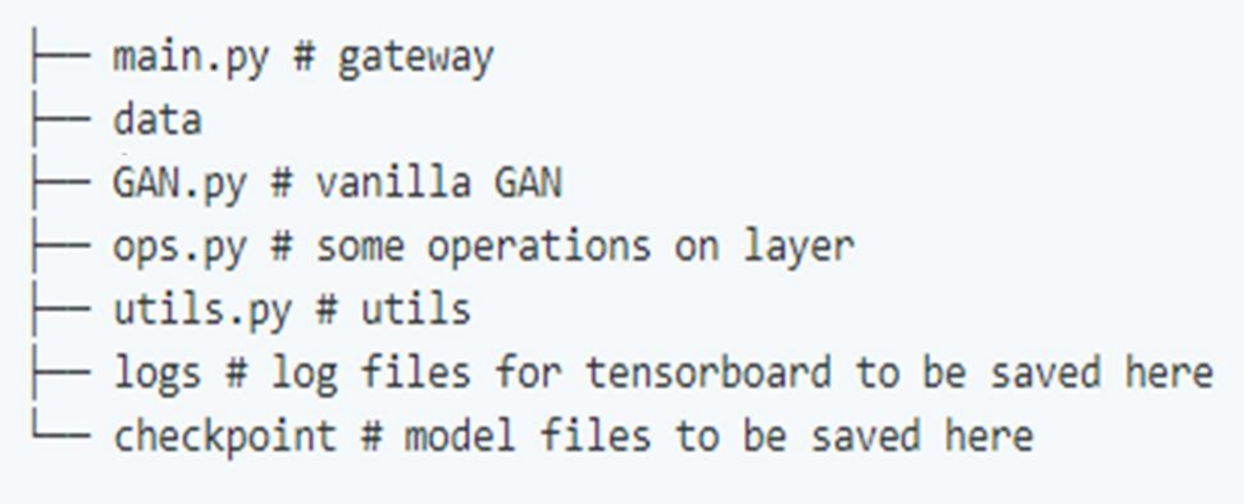
项目实践(1)：
利用GAN生成fashion-mnist图像

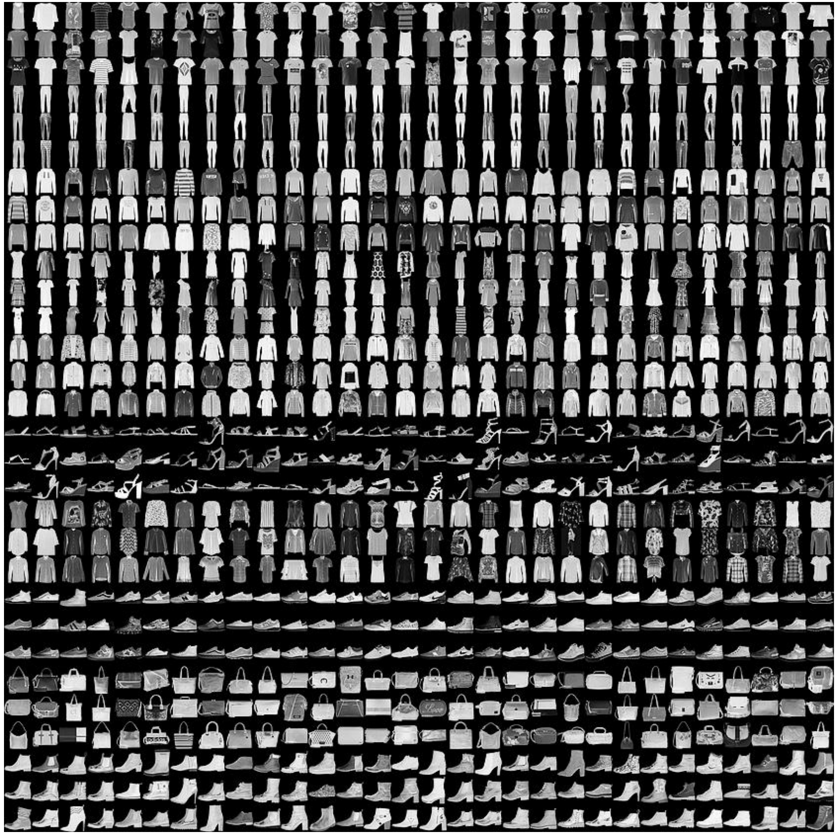
文件夹结构



```
├── main.py # gateway
├── data
├── GAN.py # vanilla GAN
├── ops.py # some operations on layer
├── utils.py # utils
├── logs # log files for tensorboard to be saved here
└── checkpoint # model files to be saved here
```

fashion-mnist 数据集

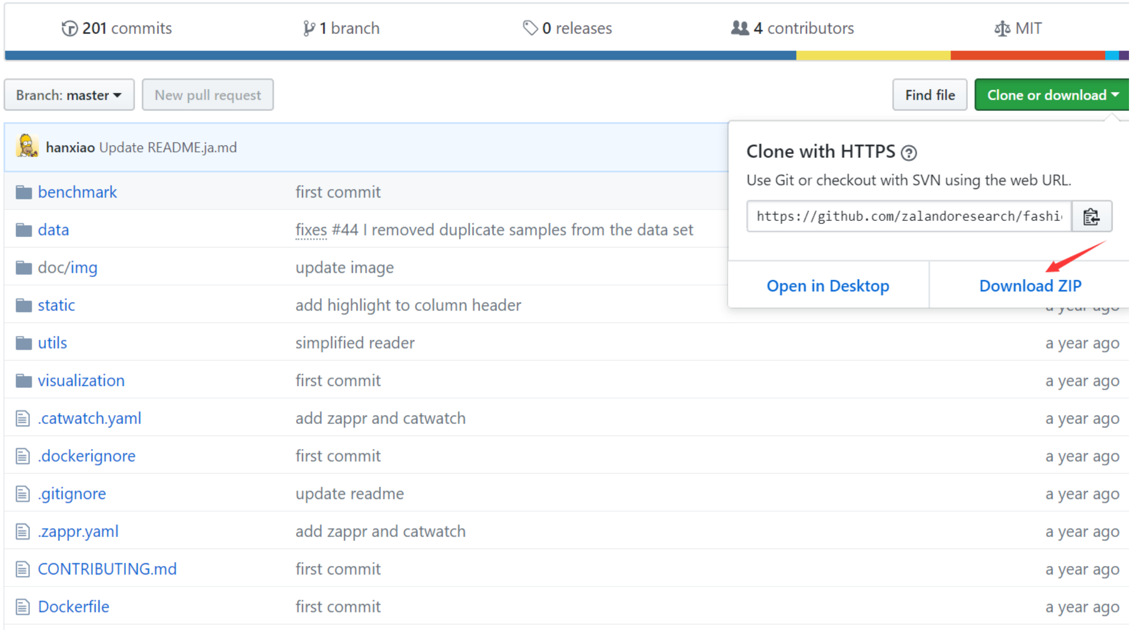
- consisting of a training set of 60,000 examples and a test set of 10,000 examples.
- Each example is a 28x28 grayscale image, associated with a label from 10 classes. (T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot)
- serving as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms.



https://github.com/zalandoresearch/fashion-mnist

fashion-mnist 数据集下载方法

https://github.com/zalandoresearch/fashion-mnist



201 commits · 1 branch · 0 releases · 4 contributors · MIT

Branch: master · New pull request · Find file · Clone or download

hanxiao Update README.ja.md

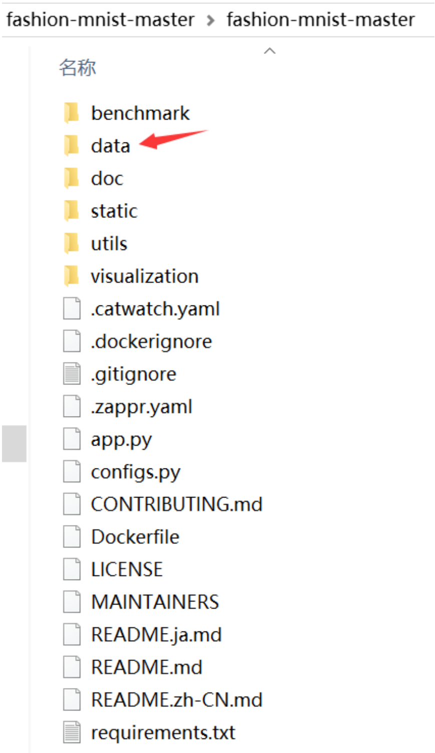
benchmark	first commit	
data	fixes #44 I removed duplicate samples from the data set	
doc/img	update image	
static	add highlight to column header	
utils	simplified reader	a year ago
visualization	first commit	a year ago
.catwatch.yaml	add zappr and catwatch	a year ago
.dockerignore	first commit	a year ago
.gitignore	update readme	a year ago
.zappr.yaml	add zappr and catwatch	a year ago
CONTRIBUTING.md	first commit	a year ago
Dockerfile	first commit	a year ago

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

https://github.com/zalandoresearch/fashi

Open in Desktop · Download ZIP



fashion-mnist-master › fashion-mnist-master

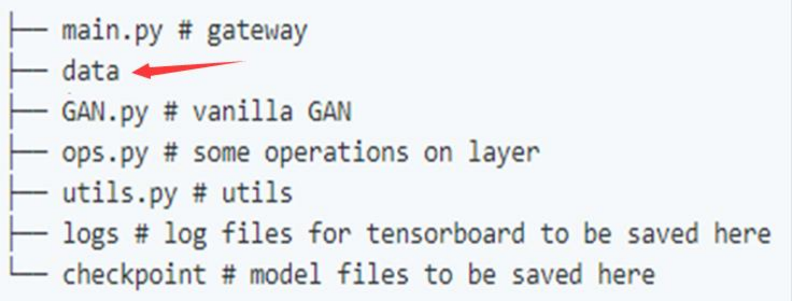
名称

- benchmark
- data
- doc
- static
- utils
- visualization
- .catwatch.yaml
- .dockerignore
- .gitignore
- .zappr.yaml
- app.py
- configs.py
- CONTRIBUTING.md
- Dockerfile
- LICENSE
- MAINTAINERS
- README.ja.md
- README.md
- README.zh-CN.md
- requirements.txt

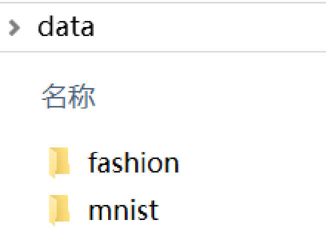
把解压出来的data目录拷贝的下载的源代码目录中

fashion-mnist 数据集安置

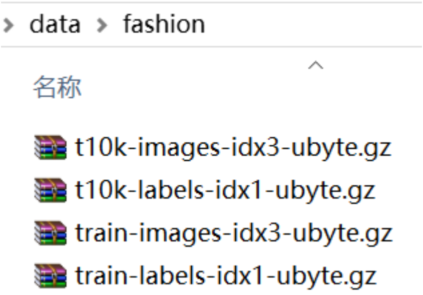
把解压出来的data目录拷贝的下载的源代码目录中



```
├── main.py # gateway
├── data
├── GAN.py # vanilla GAN
├── ops.py # some operations on layer
├── utils.py # utils
├── logs # log files for tensorboard to be saved here
└── checkpoint # model files to be saved here
```

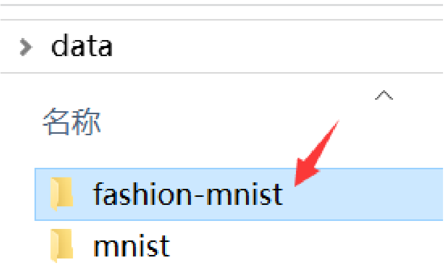


```
> data
名称
fashion
mnist
```



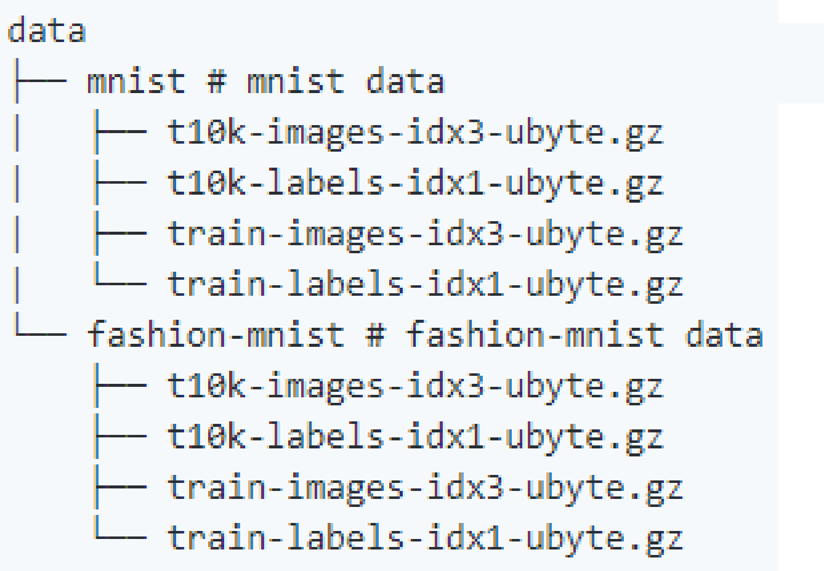
```
> data > fashion
名称
t10k-images-idx3-ubyte.gz
t10k-labels-idx1-ubyte.gz
train-images-idx3-ubyte.gz
train-labels-idx1-ubyte.gz
```

把文件夹fashion改为fashion-mnist
和代码中的数据目录保持一致

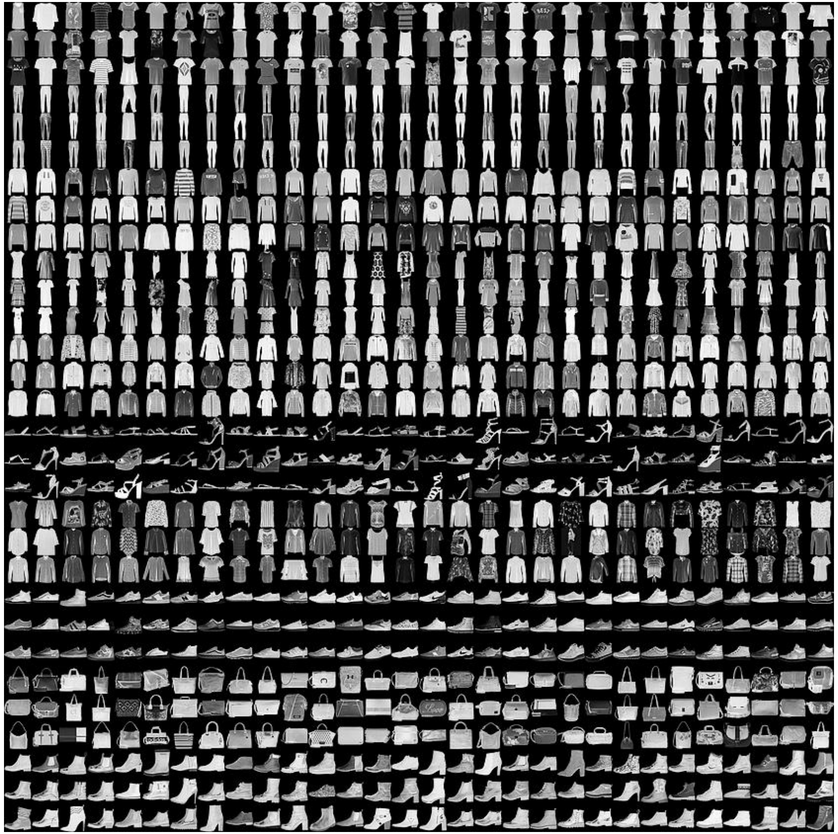


```
> data
名称
fashion-mnist
mnist
```

fashion-mnist 数据集

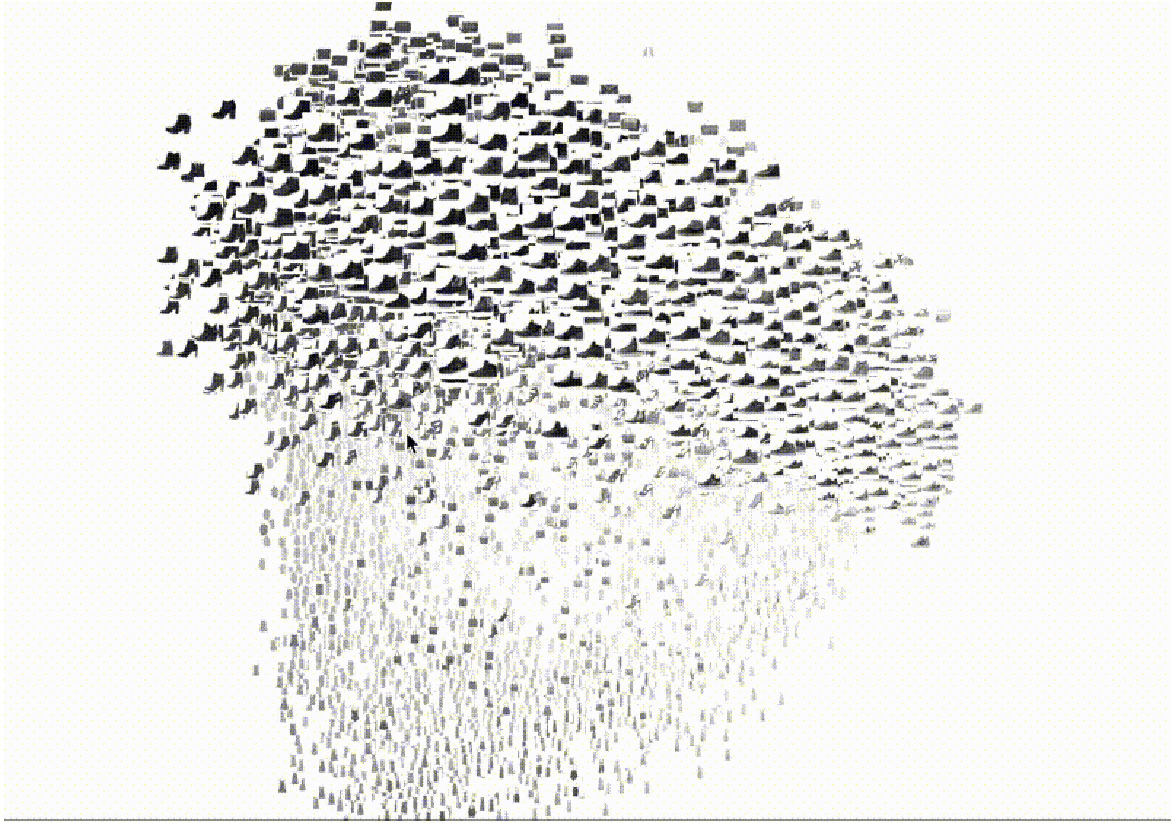


```
data
├── mnist # mnist data
│   ├── t10k-images-idx3-ubyte.gz
│   ├── t10k-labels-idx1-ubyte.gz
│   ├── train-images-idx3-ubyte.gz
│   └── train-labels-idx1-ubyte.gz
└── fashion-mnist # fashion-mnist data
    ├── t10k-images-idx3-ubyte.gz
    ├── t10k-labels-idx1-ubyte.gz
    ├── train-images-idx3-ubyte.gz
    └── train-labels-idx1-ubyte.gz
```



https://github.com/zalandoresearch/fashion-mnist

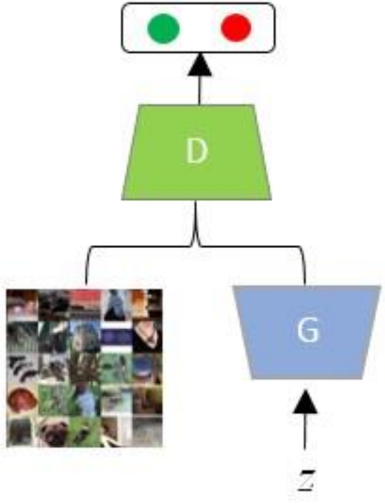
fashion-mnist 数据集



https://github.com/zalandoresearch/fashion-mnist

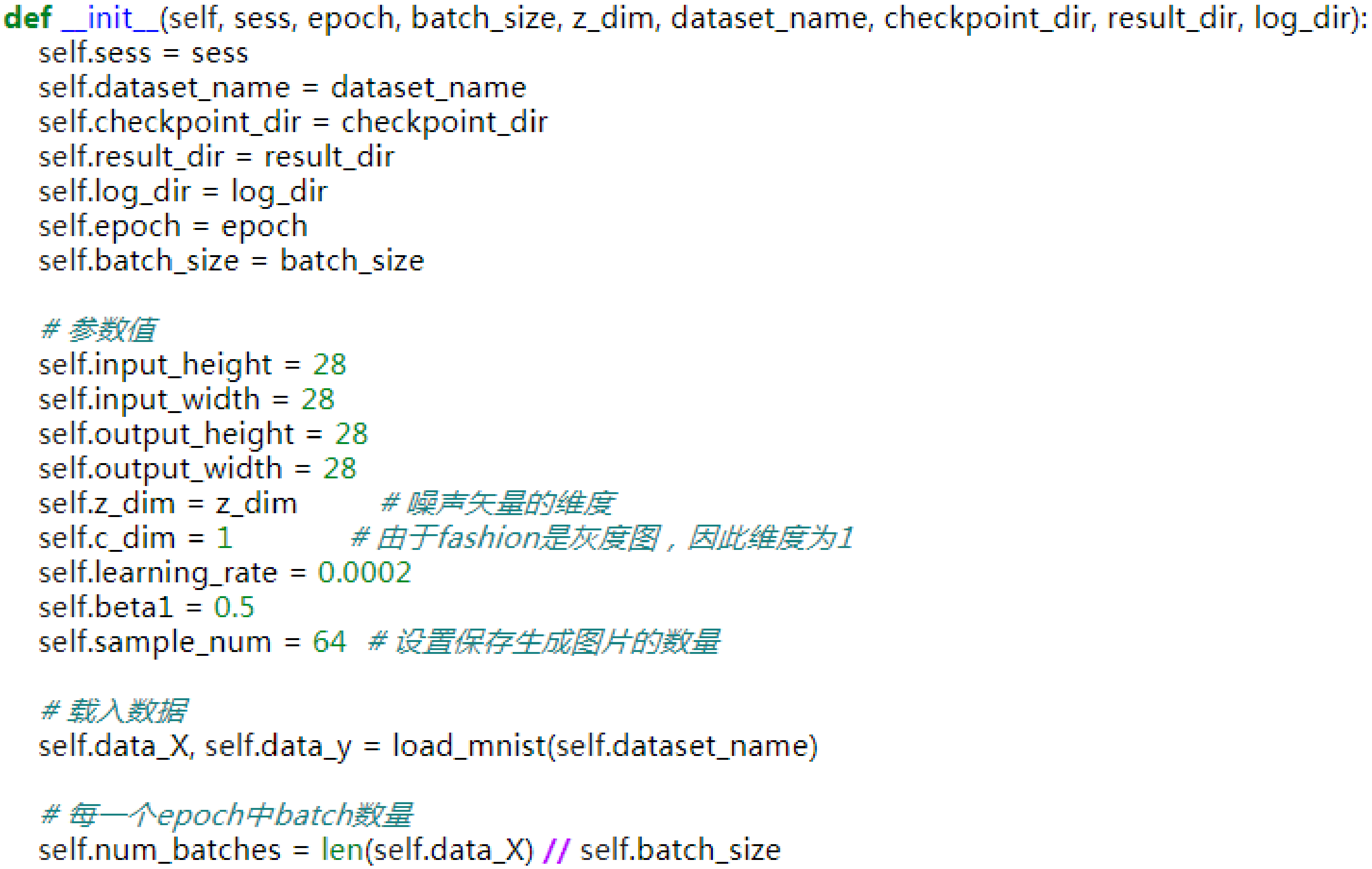
GAN的Tensorflow实现(GAN.py)

```
class GAN(object):

    """ 对实例的属性进行初始化 """
    def __init__(self, sess, epoch, batch_size, z_dim, dataset_name, checkpoint_dir, result_dir, log_dir):

    """ 搭建判别器 """
    def discriminator(self, x, is_training=True, reuse=False):

    """ 搭建生成器 """
    def generator(self, z, is_training=True, reuse=False):

    """ 构建模型 """
    def build_model(self):

    """ 执行训练 """
    def train(self):

    """ 定义功能函数 """
    def visualize_results(self, epoch):

    def model_dir(self):

    def save(self, checkpoint_dir, step):

    def load(self, checkpoint_dir):
```



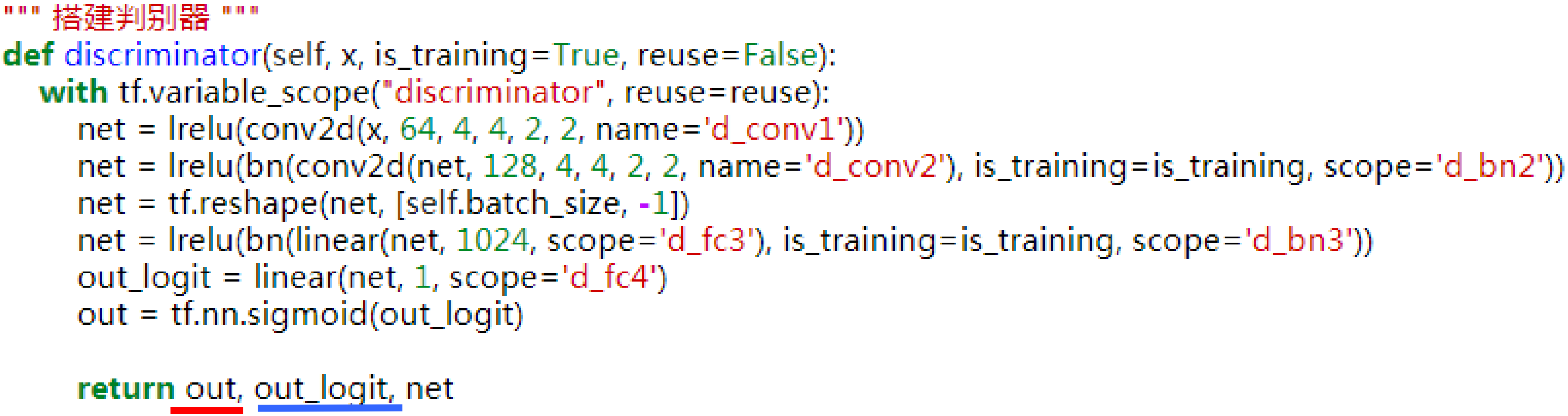
GAN

对实例的属性进行初始化



```
def __init__(self, sess, epoch, batch_size, z_dim, dataset_name, checkpoint_dir, result_dir, log_dir):
    self.sess = sess
    self.dataset_name = dataset_name
    self.checkpoint_dir = checkpoint_dir
    self.result_dir = result_dir
    self.log_dir = log_dir
    self.epoch = epoch
    self.batch_size = batch_size

    # 参数值
    self.input_height = 28
    self.input_width = 28
    self.output_height = 28
    self.output_width = 28
    self.z_dim = z_dim         # 噪声矢量的维度
    self.c_dim = 1             # 由于fashion是灰度图，因此维度为1
    self.learning_rate = 0.0002
    self.beta1 = 0.5
    self.sample_num = 64  # 设置保存生成图片的数量

    # 载入数据
    self.data_X, self.data_y = load_mnist(self.dataset_name)

    # 每一个epoch中batch数量
    self.num_batches = len(self.data_X) // self.batch_size
```

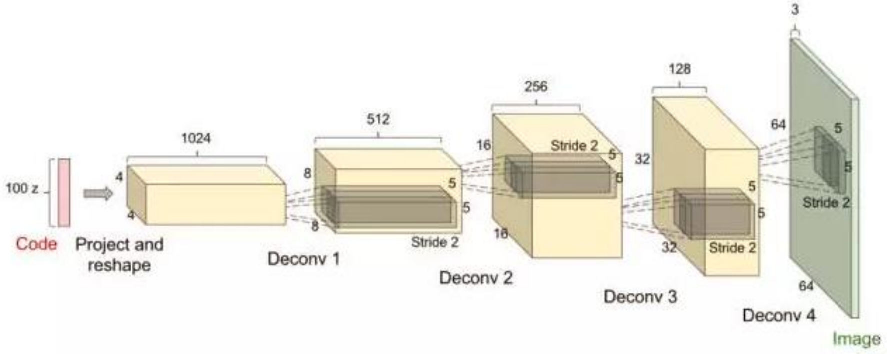
判别器*D*的网络结构



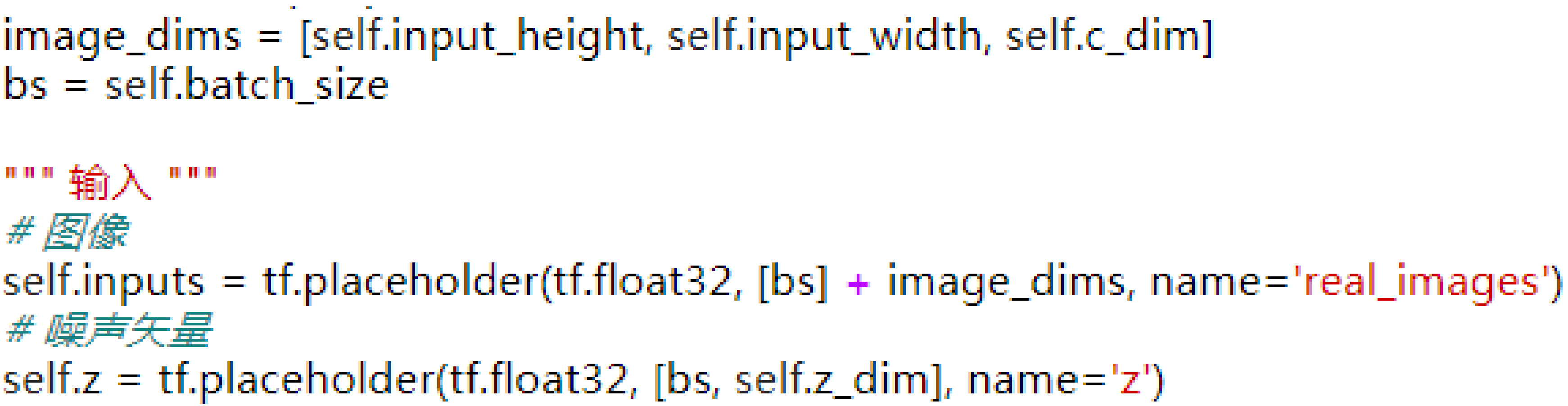
```
""" 搭建判别器 """
def discriminator(self, x, is_training=True, reuse=False):
    with tf.variable_scope("discriminator", reuse=reuse):
        net = lrelu(conv2d(x, 64, 4, 4, 2, 2, name='d_conv1'))
        net = lrelu(bn(conv2d(net, 128, 4, 4, 2, 2, name='d_conv2'), is_training=is_training, scope='d_bn2'))
        net = tf.reshape(net, [self.batch_size, -1])
        net = lrelu(bn(linear(net, 1024, scope='d_fc3'), is_training=is_training, scope='d_bn3'))
        out_logit = linear(net, 1, scope='d_fc4')
        out = tf.nn.sigmoid(out_logit)

        return out, out_logit, net
```

生成器G的网络结构

```
""" 搭建生成器 """
def generator(self, z, is_training=True, reuse=False):
    with tf.variable_scope("generator", reuse=reuse):
        net = tf.nn.relu(bn(linear(z, 1024, scope='g_fc1'), is_training=is_training, scope='g_bn1'))
        net = tf.nn.relu(bn(linear(net, 128 * 7 * 7, scope='g_fc2'), is_training=is_training, scope='g_bn2'))
        net = tf.reshape(net, [self.batch_size, 7, 7, 128])
        net = tf.nn.relu(
            bn(deconv2d(net, [self.batch_size, 14, 14, 64], 4, 4, 2, 2, name='g_dc3'), is_training=is_training,
               scope='g_bn3'))
        out = tf.nn.sigmoid(deconv2d(net, [self.batch_size, 28, 28, 1], 4, 4, 2, 2, name='g_dc4'))

        return out
```

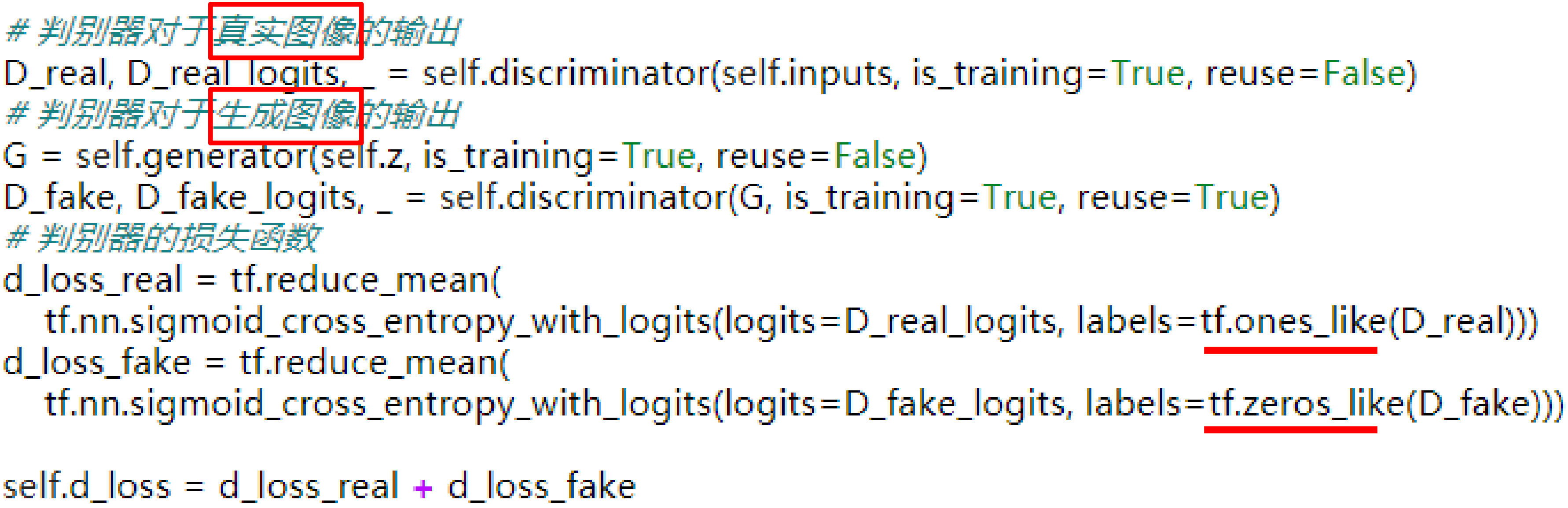


定义输入



```
image_dims = [self.input_height, self.input_width, self.c_dim]
bs = self.batch_size

""" 输入 """
# 图像
self.inputs = tf.placeholder(tf.float32, [bs] + image_dims, name='real_images')
# 噪声矢量
self.z = tf.placeholder(tf.float32, [bs, self.z_dim], name='z')
```

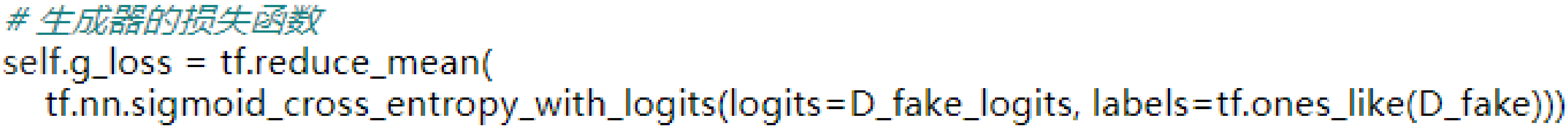
构建判别器*D*的损失函数



```
# 判别器对于真实图像的输出
D_real, D_real_logits, _ = self.discriminator(self.inputs, is_training=True, reuse=False)
# 判别器对于生成图像的输出
G = self.generator(self.z, is_training=True, reuse=False)
D_fake, D_fake_logits, _ = self.discriminator(G, is_training=True, reuse=True)
# 判别器的损失函数
d_loss_real = tf.reduce_mean(
   tf.nn.sigmoid_cross_entropy_with_logits(logits=D_real_logits, labels=tf.ones_like(D_real)))
d_loss_fake = tf.reduce_mean(
   tf.nn.sigmoid_cross_entropy_with_logits(logits=D_fake_logits, labels=tf.zeros_like(D_fake)))

self.d_loss = d_loss_real + d_loss_fake
```

- 判别器的目的是尽量正确判别输入数据是真实数据还是来自生成器！

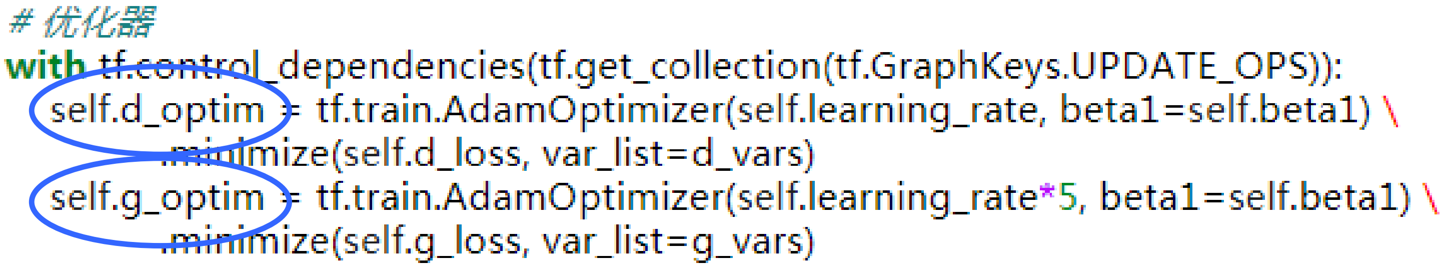
构建生成器*G*的损失函数



```
# 生成器的损失函数
self.g_loss = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(logits=D_fake_logits, labels=tf.ones_like(D_fake)))
```

- 生成器的目的是尽量去学习真实的数据分布，使得生成样本能够以假乱真！

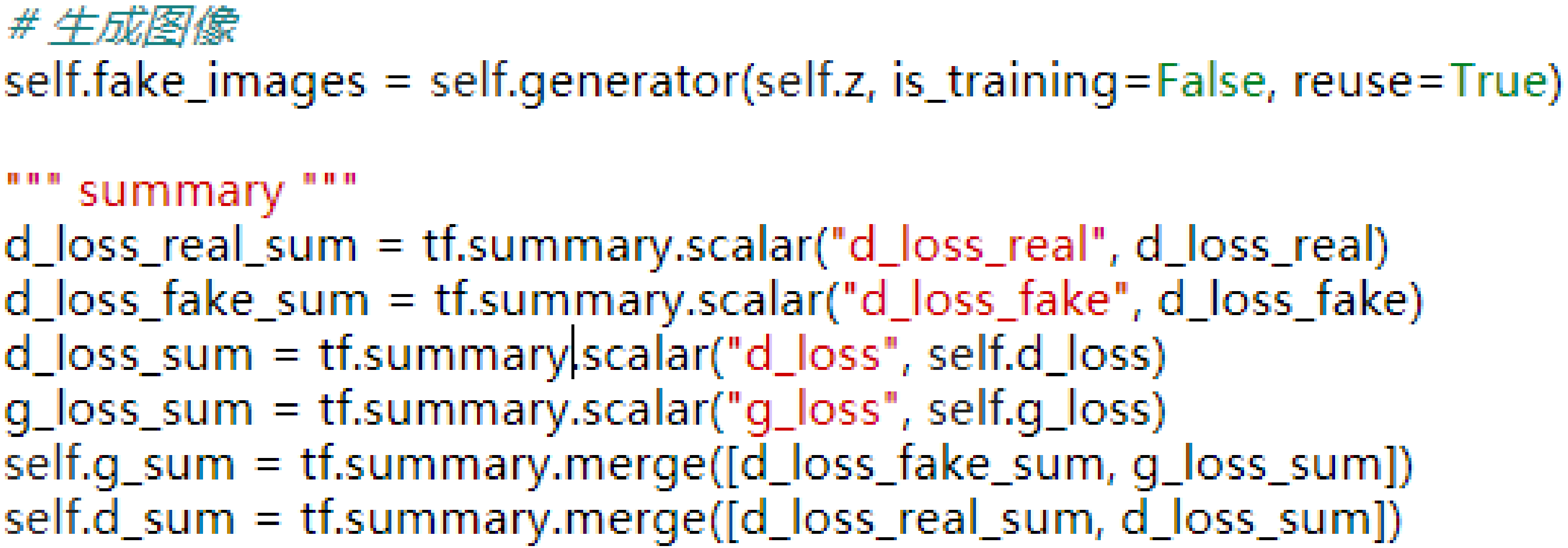
训练



```
# 优化器
with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
  self.d_optim = tf.train.AdamOptimizer(self.learning_rate, beta1=self.beta1) \
            .minimize(self.d_loss, var_list=d_vars)
  self.g_optim = tf.train.AdamOptimizer(self.learning_rate*5, beta1=self.beta1) \
            .minimize(self.g_loss, var_list=g_vars)
```

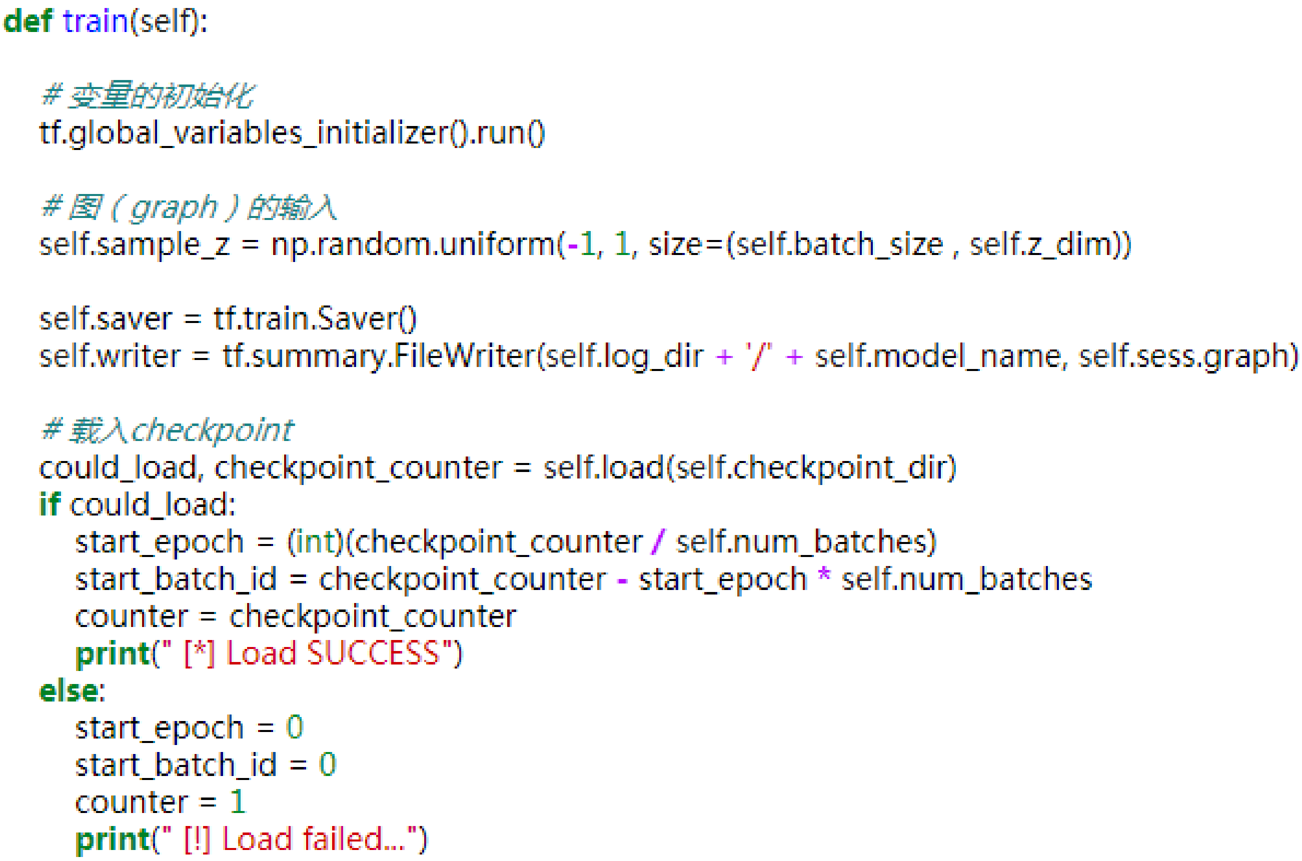
- D和G交替优化：在上面的步骤中，每对D的参数更新1次，便接着对G的参数更新1次；
- 有时还可以对D的参数更新K次后再对G的参数更新1次

生成图像及状态可视化



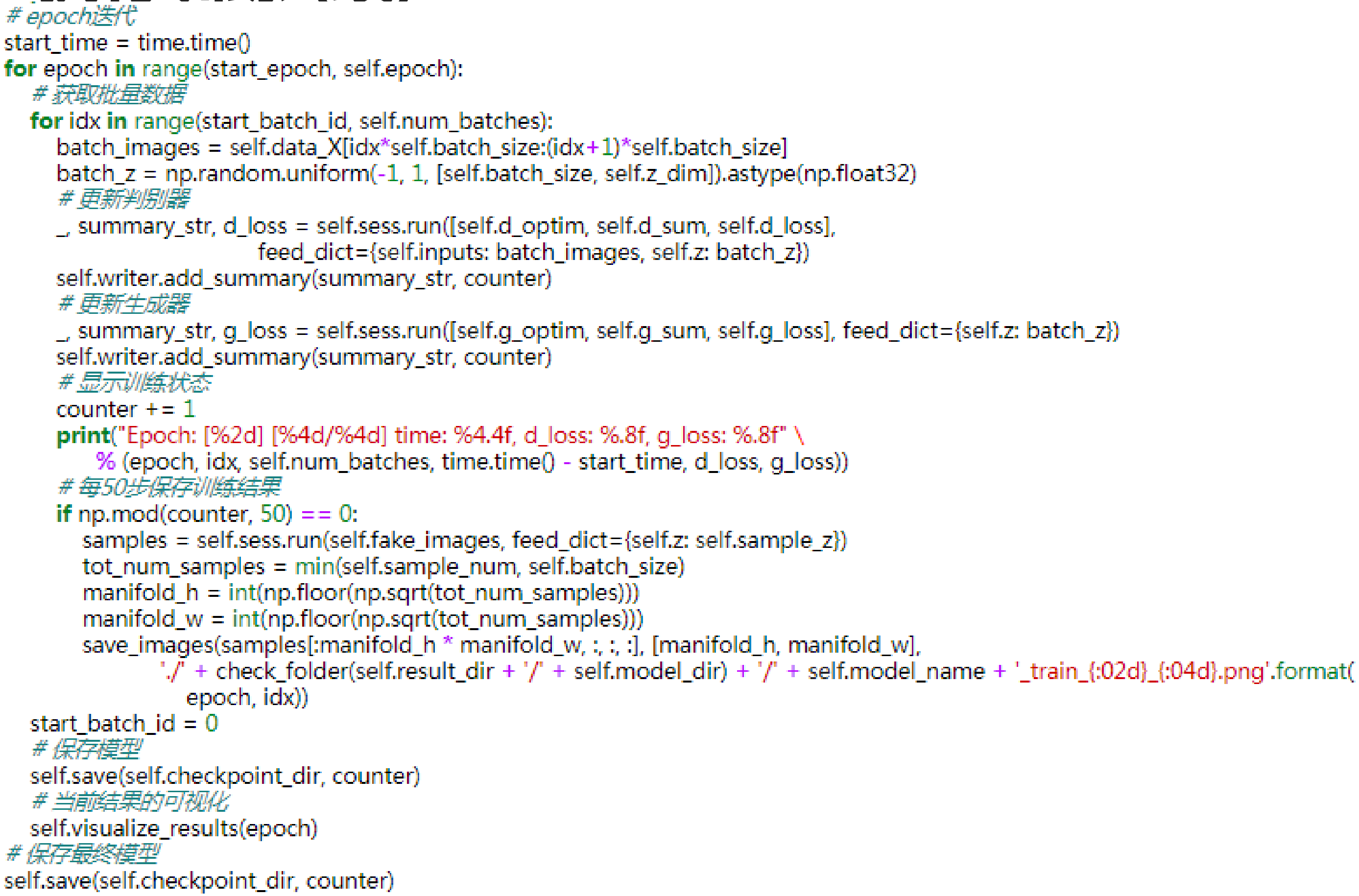
```
# 生成图像
self.fake_images = self.generator(self.z, is_training=False, reuse=True)

""" summary """
d_loss_real_sum = tf.summary.scalar("d_loss_real", d_loss_real)
d_loss_fake_sum = tf.summary.scalar("d_loss_fake", d_loss_fake)
d_loss_sum = tf.summary.scalar("d_loss", self.d_loss)
g_loss_sum = tf.summary.scalar("g_loss", self.g_loss)
self.g_sum = tf.summary.merge([d_loss_fake_sum, g_loss_sum])
self.d_sum = tf.summary.merge([d_loss_real_sum, d_loss_sum])
```

执行训练



```
def train(self):

    # 变量的初始化
    tf.global_variables_initializer().run()

    # 图（graph）的输入
    self.sample_z = np.random.uniform(-1, 1, size=(self.batch_size , self.z_dim))

    self.saver = tf.train.Saver()
    self.writer = tf.summary.FileWriter(self.log_dir + '/' + self.model_name, self.sess.graph)

    # 载入checkpoint
    could_load, checkpoint_counter = self.load(self.checkpoint_dir)
    if could_load:
        start_epoch = (int)(checkpoint_counter / self.num_batches)
        start_batch_id = checkpoint_counter - start_epoch * self.num_batches
        counter = checkpoint_counter
        print(" [*] Load SUCCESS")
    else:
        start_epoch = 0
        start_batch_id = 0
        counter = 1
        print(" [!] Load failed...")
```

执行训练（续）

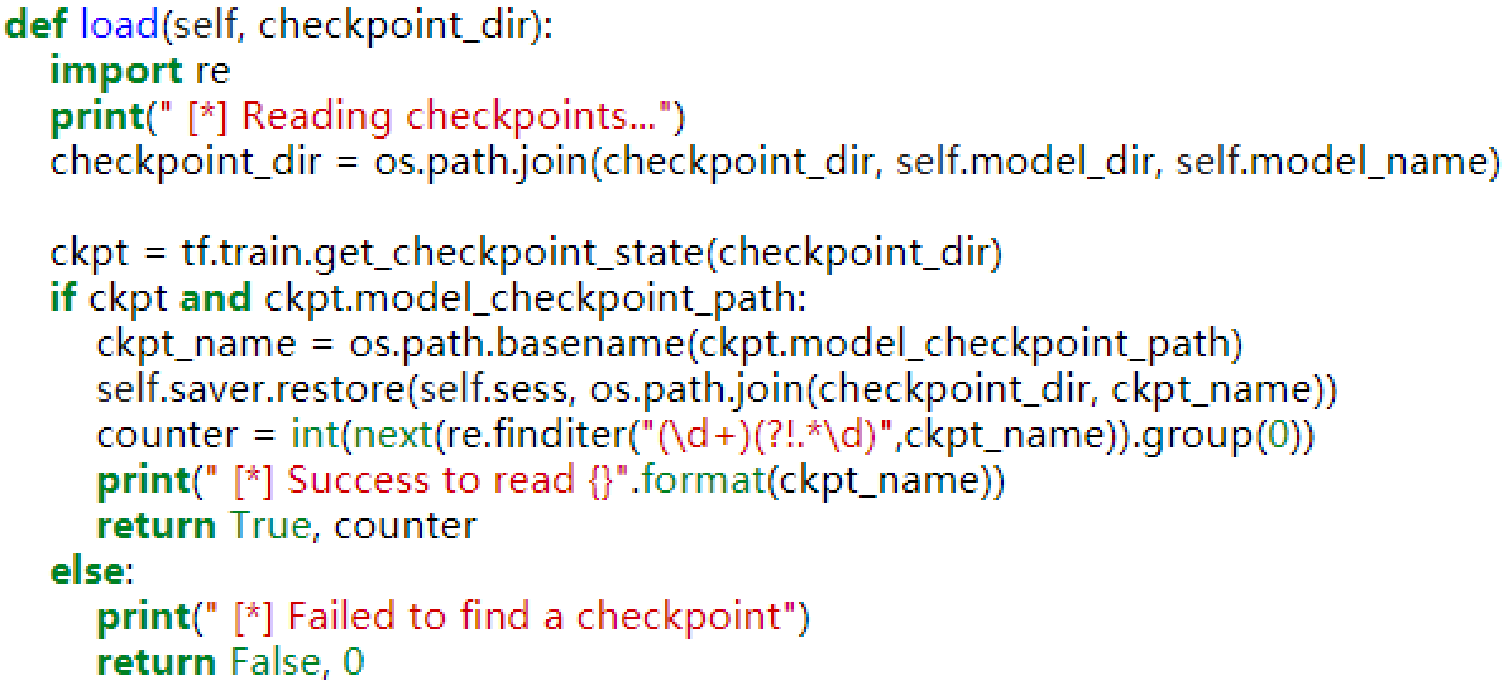


```
# epoch迭代
start_time = time.time()
for epoch in range(start_epoch, self.epoch):
    # 获取批量数据
    for idx in range(start_batch_id, self.num_batches):
        batch_images = self.data_X[idx*self.batch_size:(idx+1)*self.batch_size]
        batch_z = np.random.uniform(-1, 1, [self.batch_size, self.z_dim]).astype(np.float32)
        # 更新判别器
        _, summary_str, d_loss = self.sess.run([self.d_optim, self.d_sum, self.d_loss],
                                              feed_dict={self.inputs: batch_images, self.z: batch_z})
        self.writer.add_summary(summary_str, counter)
        # 更新生成器
        _, summary_str, g_loss = self.sess.run([self.g_optim, self.g_sum, self.g_loss], feed_dict={self.z: batch_z})
        self.writer.add_summary(summary_str, counter)
        # 显示训练状态
        counter += 1
        print("Epoch: [%2d] [%4d/%4d] time: %4.4f, d_loss: %.8f, g_loss: %.8f" \
              % (epoch, idx, self.num_batches, time.time() - start_time, d_loss, g_loss))
        # 每50步保存训练结果
        if np.mod(counter, 50) == 0:
            samples = self.sess.run(self.fake_images, feed_dict={self.z: self.sample_z})
            tot_num_samples = min(self.sample_num, self.batch_size)
            manifold_h = int(np.floor(np.sqrt(tot_num_samples)))
            manifold_w = int(np.floor(np.sqrt(tot_num_samples)))
            save_images(samples[:manifold_h * manifold_w, :, :, :], [manifold_h, manifold_w],
                        './' + check_folder(self.result_dir + '/' + self.model_dir) + '/' + self.model_name + '_train_{:02d}_{:04d}.png'.format(
                            epoch, idx))
    start_batch_id = 0
    # 保存模型
    self.save(self.checkpoint_dir, counter)
    # 当前结果的可视化
    self.visualize_results(epoch)
# 保存最终模型
self.save(self.checkpoint_dir, counter)
```

定义功能函数

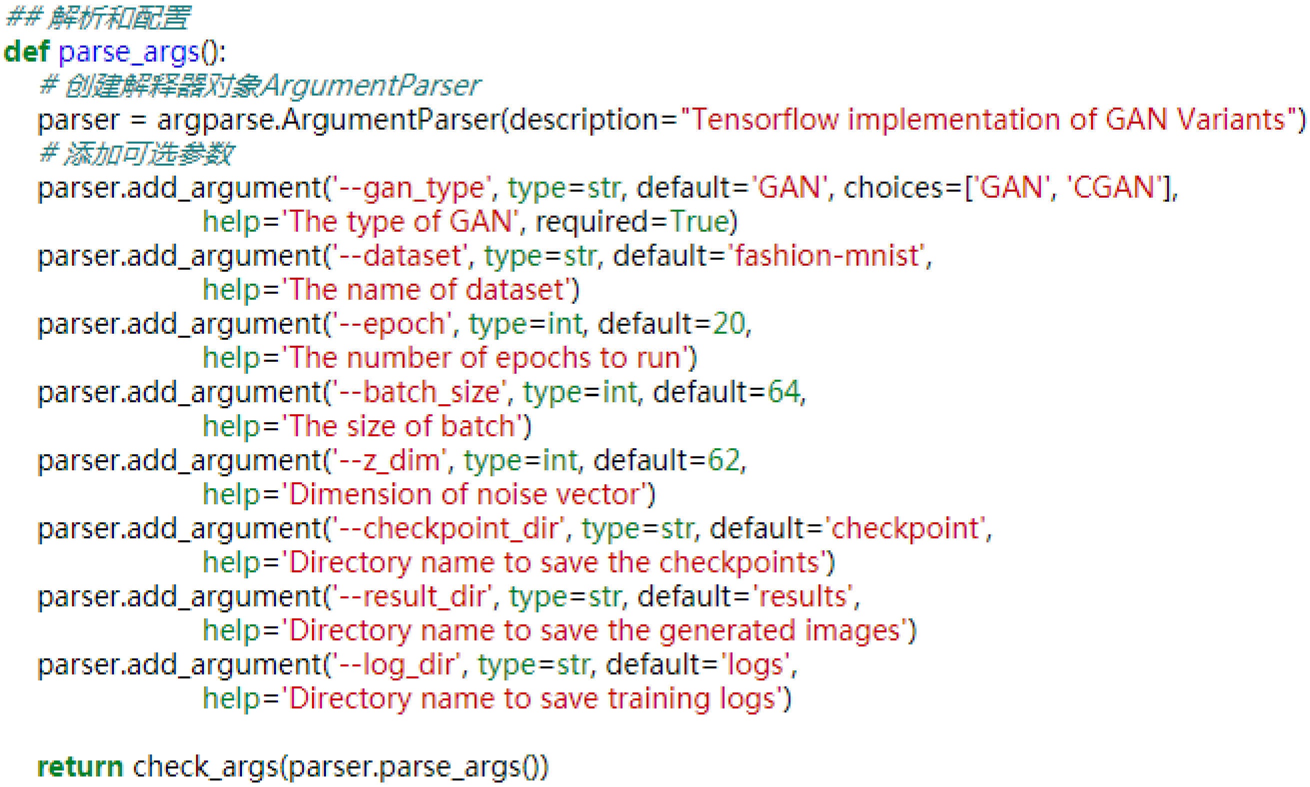
```
""" 定义功能函数 """
def visualize_results(self, epoch):
    tot_num_samples = min(self.sample_num, self.batch_size)
    image_frame_dim = int(np.floor(np.sqrt(tot_num_samples)))

    z_sample = np.random.uniform(-1, 1, size=(self.batch_size, self.z_dim))
    samples = self.sess.run(self.fake_images, feed_dict={self.z: z_sample})

    save_images(samples[:image_frame_dim * image_frame_dim, :, :, :], [image_frame_dim, image_frame_dim],
                check_folder(self.result_dir + '/' + self.model_dir) + '/' + self.model_name + '_epoch%03d' % epoch + '_test_all_classes.png')

@property
def model_dir(self):
    return "{}_{}_{}_{}".format(
        self.model_name, self.dataset_name,
        self.batch_size, self.z_dim)

def save(self, checkpoint_dir, step):
    checkpoint_dir = os.path.join(checkpoint_dir, self.model_dir, self.model_name)

    if not os.path.exists(checkpoint_dir):
        os.makedirs(checkpoint_dir)

    self.saver.save(self.sess,os.path.join(checkpoint_dir, self.model_name+'.model'), global_step=step)
```

定义功能函数（续）



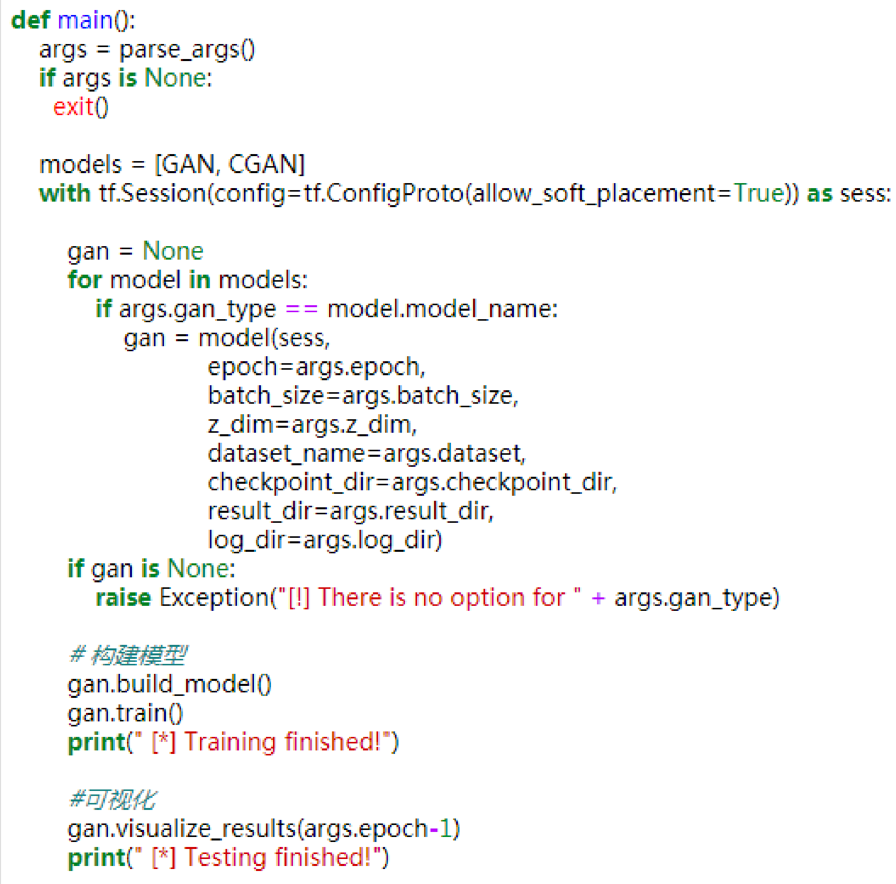
```
def load(self, checkpoint_dir):
    import re
    print(" [*] Reading checkpoints...")
    checkpoint_dir = os.path.join(checkpoint_dir, self.model_dir, self.model_name)

    ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
    if ckpt and ckpt.model_checkpoint_path:
        ckpt_name = os.path.basename(ckpt.model_checkpoint_path)
        self.saver.restore(self.sess, os.path.join(checkpoint_dir, ckpt_name))
        counter = int(next(re.finditer("(\d+)(?!.*\d)",ckpt_name)).group(0))
        print(" [*] Success to read {}".format(ckpt_name))
        return True, counter
    else:
        print(" [*] Failed to find a checkpoint")
        return False, 0
```

main.py



```
## 解析和配置
def parse_args():
    # 创建解释器对象ArgumentParser
    parser = argparse.ArgumentParser(description="Tensorflow implementation of GAN Variants")
    # 添加可选参数
    parser.add_argument('--gan_type', type=str, default='GAN', choices=['GAN', 'CGAN'],
                        help='The type of GAN', required=True)
    parser.add_argument('--dataset', type=str, default='fashion-mnist',
                        help='The name of dataset')
    parser.add_argument('--epoch', type=int, default=20,
                        help='The number of epochs to run')
    parser.add_argument('--batch_size', type=int, default=64,
                        help='The size of batch')
    parser.add_argument('--z_dim', type=int, default=62,
                        help='Dimension of noise vector')
    parser.add_argument('--checkpoint_dir', type=str, default='checkpoint',
                        help='Directory name to save the checkpoints')
    parser.add_argument('--result_dir', type=str, default='results',
                        help='Directory name to save the generated images')
    parser.add_argument('--log_dir', type=str, default='logs',
                        help='Directory name to save training logs')

    return check_args(parser.parse_args())
```

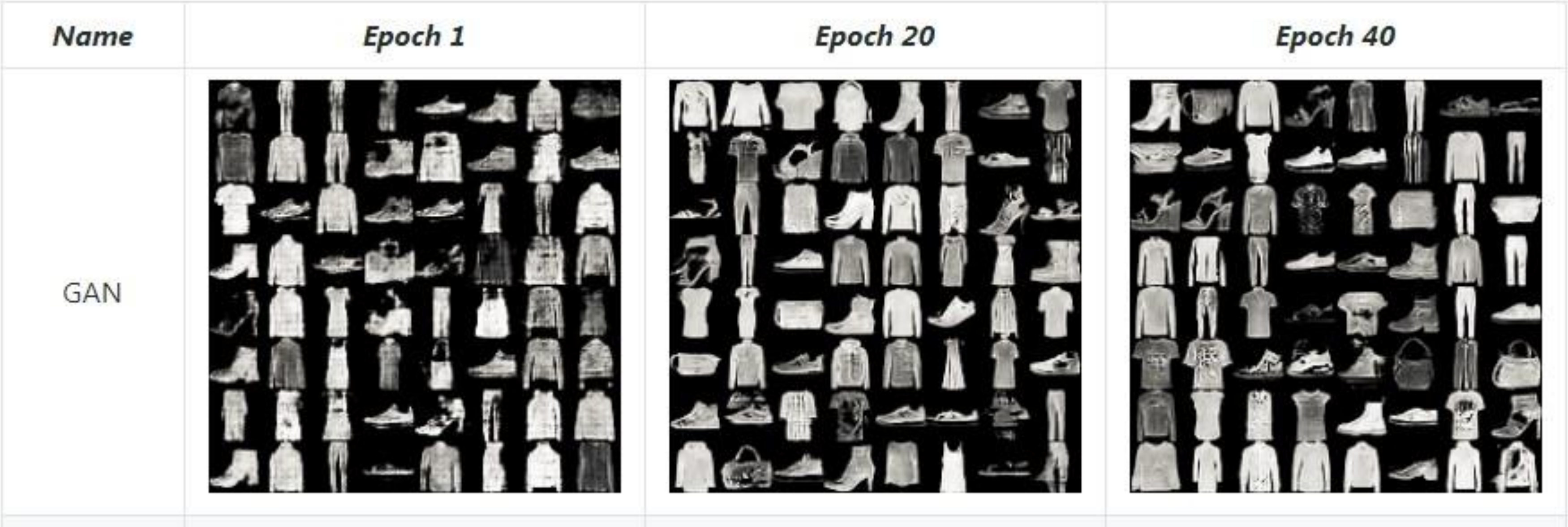
main.py （续）



```
def main():
    args = parse_args()
    if args is None:
      exit()

    models = [GAN, CGAN]
    with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:

        gan = None
        for model in models:
            if args.gan_type == model.model_name:
                gan = model(sess,
                            epoch=args.epoch,
                            batch_size=args.batch_size,
                            z_dim=args.z_dim,
                            dataset_name=args.dataset,
                            checkpoint_dir=args.checkpoint_dir,
                            result_dir=args.result_dir,
                            log_dir=args.log_dir)
        if gan is None:
            raise Exception("[!] There is no option for " + args.gan_type)

        # 构建模型
        gan.build_model()
        gan.train()
        print(" [*] Training finished!")

        #可视化
        gan.visualize_results(args.epoch-1)
        print(" [*] Testing finished!")
```

运行

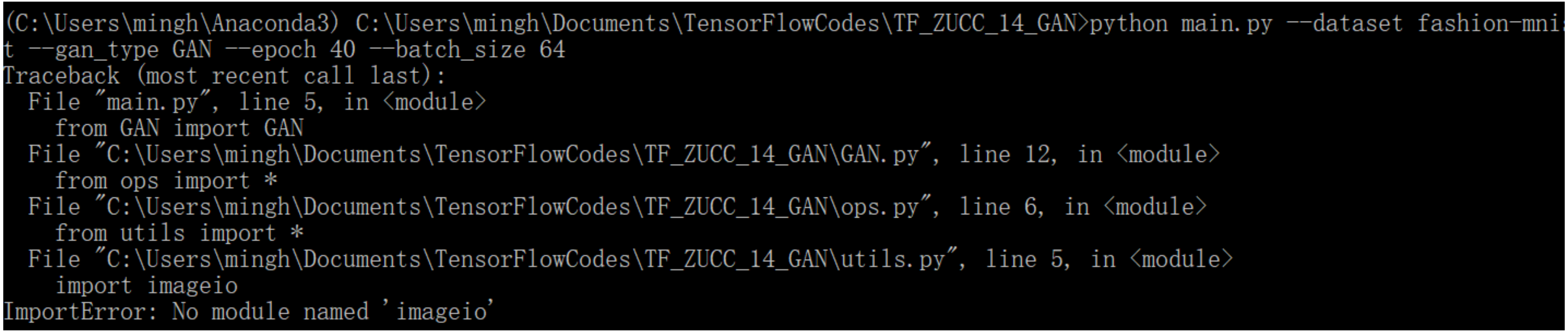
```
python main.py --dataset fashion-mnist --gan_type GAN --epoch 40 --batch_size 64
```



Name	*Epoch 1*	*Epoch 20*	*Epoch 40*
GAN			

如果运行报错……

根据出错提示进行相应处理，比如…



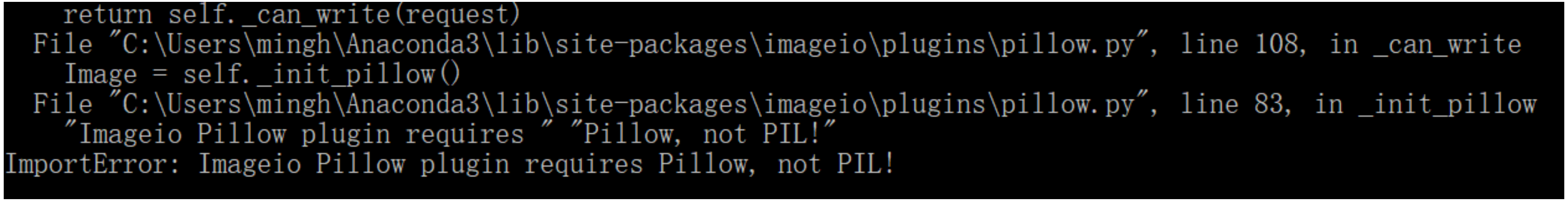
```
(C:\Users\mingh\Anaconda3) C:\Users\mingh\Documents\TensorFlowCodes\TF_ZUCC_14_GAN>python main.py --dataset fashion-mni
t --gan_type GAN --epoch 40 --batch_size 64
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    from GAN import GAN
  File "C:\Users\mingh\Documents\TensorFlowCodes\TF_ZUCC_14_GAN\GAN.py", line 12, in <module>
    from ops import *
  File "C:\Users\mingh\Documents\TensorFlowCodes\TF_ZUCC_14_GAN\ops.py", line 6, in <module>
    from utils import *
  File "C:\Users\mingh\Documents\TensorFlowCodes\TF_ZUCC_14_GAN\utils.py", line 5, in <module>
    import imageio
ImportError: No module named 'imageio'
```

解决方案：安装imageio库

```
conda install imageio
```

如果运行报错……

根据出错提示进行相应处理，比如…



```
    return self._can_write(request)
  File "C:\Users\mingh\Anaconda3\lib\site-packages\imageio\plugins\pillow.py", line 108, in _can_write
    Image = self._init_pillow()
  File "C:\Users\mingh\Anaconda3\lib\site-packages\imageio\plugins\pillow.py", line 83, in _init_pillow
    "Imageio Pillow plugin requires " "Pillow, not PIL!"
ImportError: Imageio Pillow plugin requires Pillow, not PIL!
```

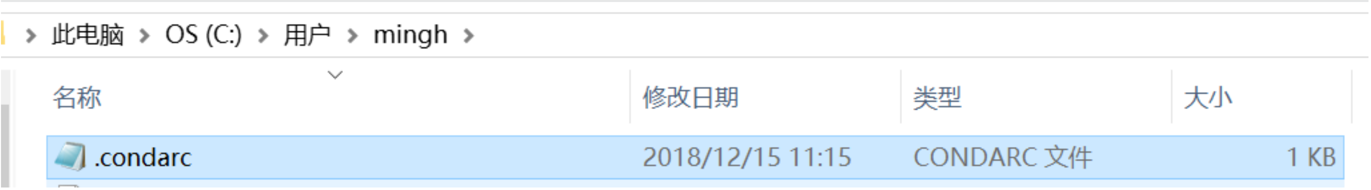
解决方案：更新pillow库

conda upgrade pillow

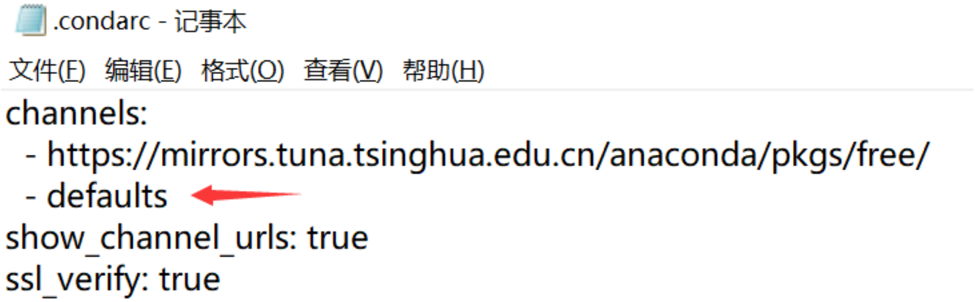
如果运行报错……

如果提示找不到要下载的库，可能是前面设定的清华映像站没有该库

解决方案：找到用户目录底下找到 **.condarc** 文件，打开编辑，加上defaults下载通道



名称	修改日期	类型	大小
.condarc	2018/12/15 11:15	CONDARC 文件	1 KB



```
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
  - defaults
show_channel_urls: true
ssl_verify: true
```