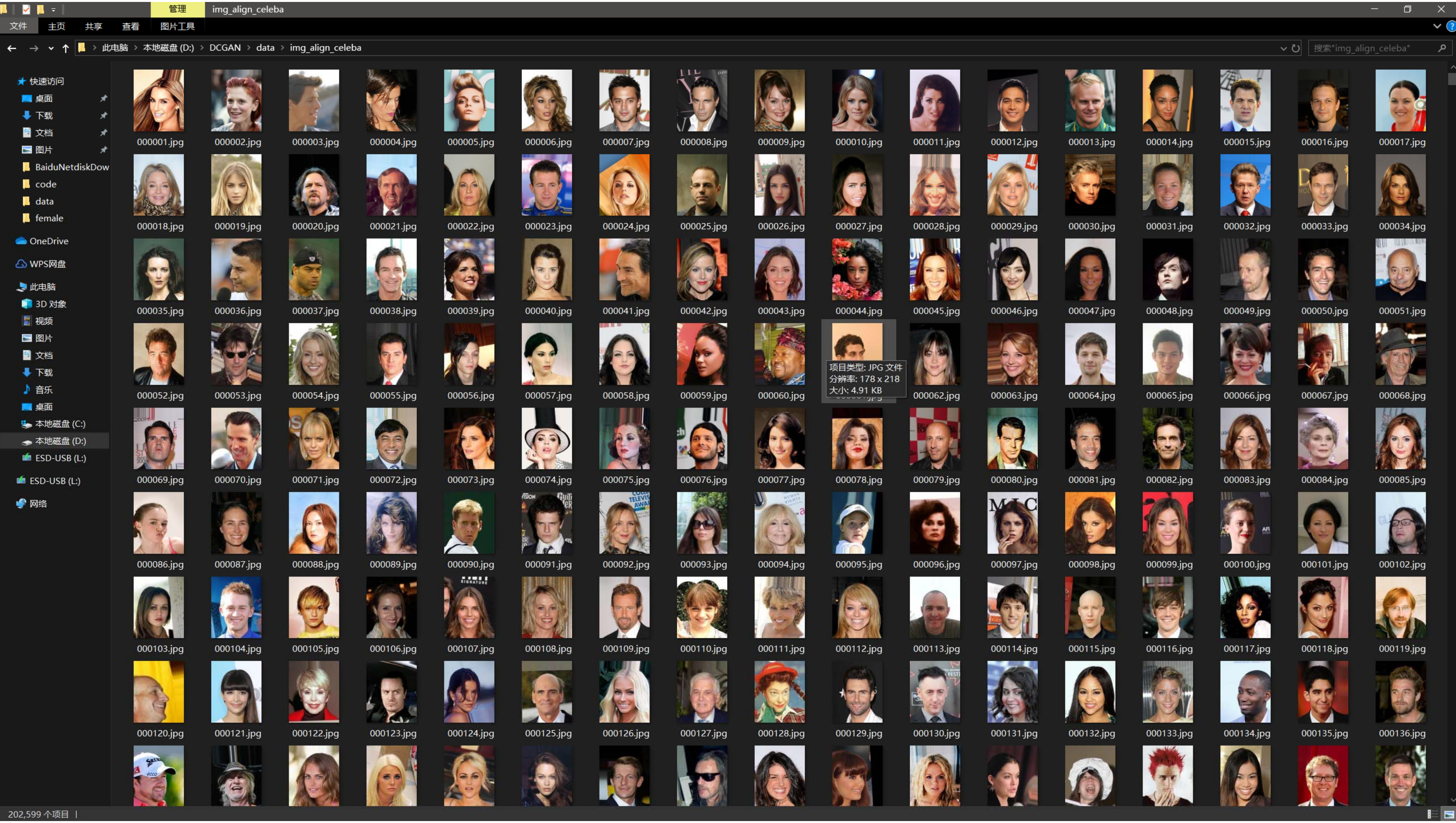
人脸生成——GAN

以CelebA为数据集
通过生成式对抗神经网络
以随机噪声为输入
输出人脸

概　　述

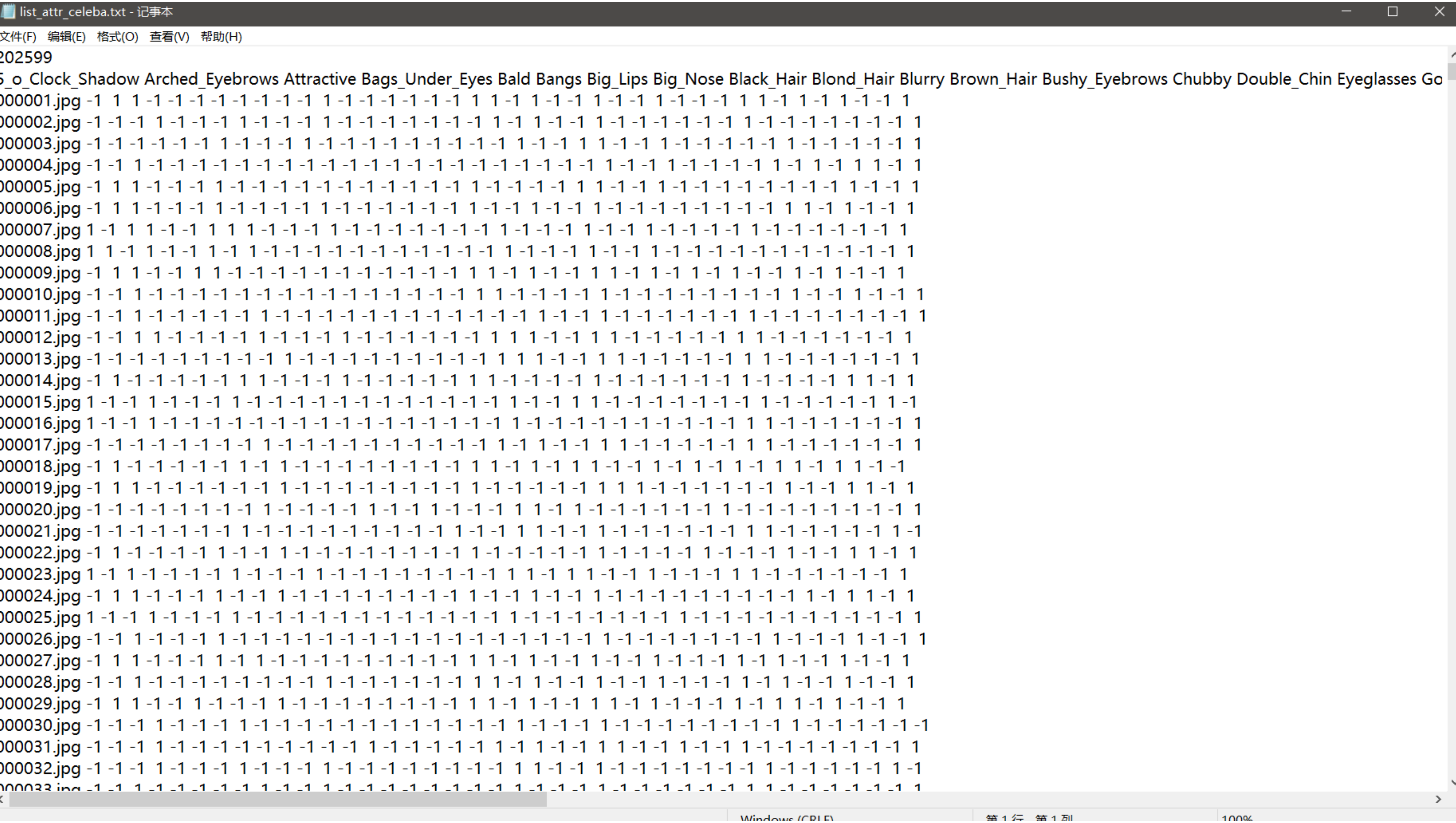
DATASET—CelebA

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations



项目类型: JPG 文件
分辨率: 178 x 218
大小: 4.91 KB

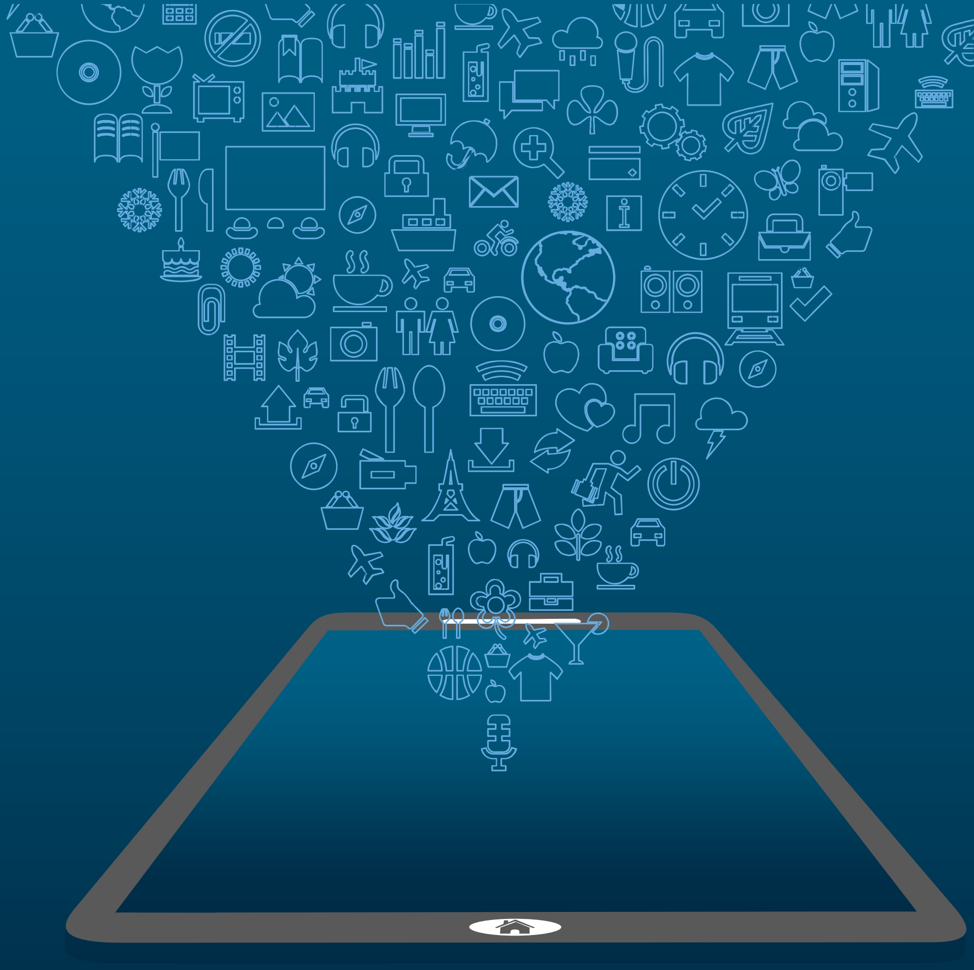
202,599 个项目

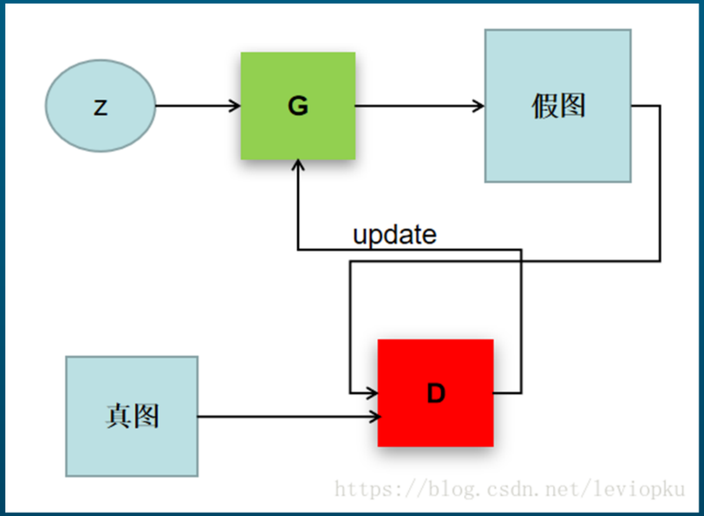


list_attr_celeba.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
202599
5_o_Clock_Shadow Arched_Eyebrows Attractive Bags_Under_Eyes Bald Bangs Big_Lips Big_Nose Black_Hair Blond_Hair Blurry Brown_Hair Bushy_Eyebrows Chubby Double_Chin Eyeglasses Go
000001.jpg -1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1 -1  1 -1  1 -1 -1  1
000002.jpg -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1
000003.jpg -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1
000004.jpg -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1  1 -1  1 -1  1  1 -1  1
000005.jpg -1  1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1
000006.jpg -1  1  1 -1 -1 -1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1
000007.jpg  1 -1  1  1 -1 -1  1  1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1
000008.jpg  1  1 -1  1 -1 -1  1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1
000009.jpg -1  1  1 -1 -1  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1  1 -1  1 -1  1 -1  1 -1 -1  1 -1  1 -1 -1  1
000010.jpg -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1  1
000011.jpg -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1
000012.jpg -1 -1  1  1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1
000013.jpg -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1
000014.jpg -1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1  1  1 -1  1
000015.jpg  1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1
000016.jpg  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1
000017.jpg -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1
000018.jpg -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1  1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1  1 -1 -1
000019.jpg -1  1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1  1 -1 -1  1  1 -1  1
000020.jpg -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1
000021.jpg -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1  1 -1
000022.jpg -1  1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1  1 -1  1
000023.jpg  1 -1  1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1  1  1 -1 -1  1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1
000024.jpg -1  1  1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1  1  1 -1  1
000025.jpg  1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1
000026.jpg -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1
000027.jpg -1  1  1 -1 -1 -1  1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1
000028.jpg -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1
000029.jpg -1  1  1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1  1 -1  1 -1 -1 -1  1 -1  1  1 -1  1 -1 -1  1
000030.jpg -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1
000031.jpg -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1
000032.jpg -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1
000033.jpg -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1 -1  1
```

Model Introduce



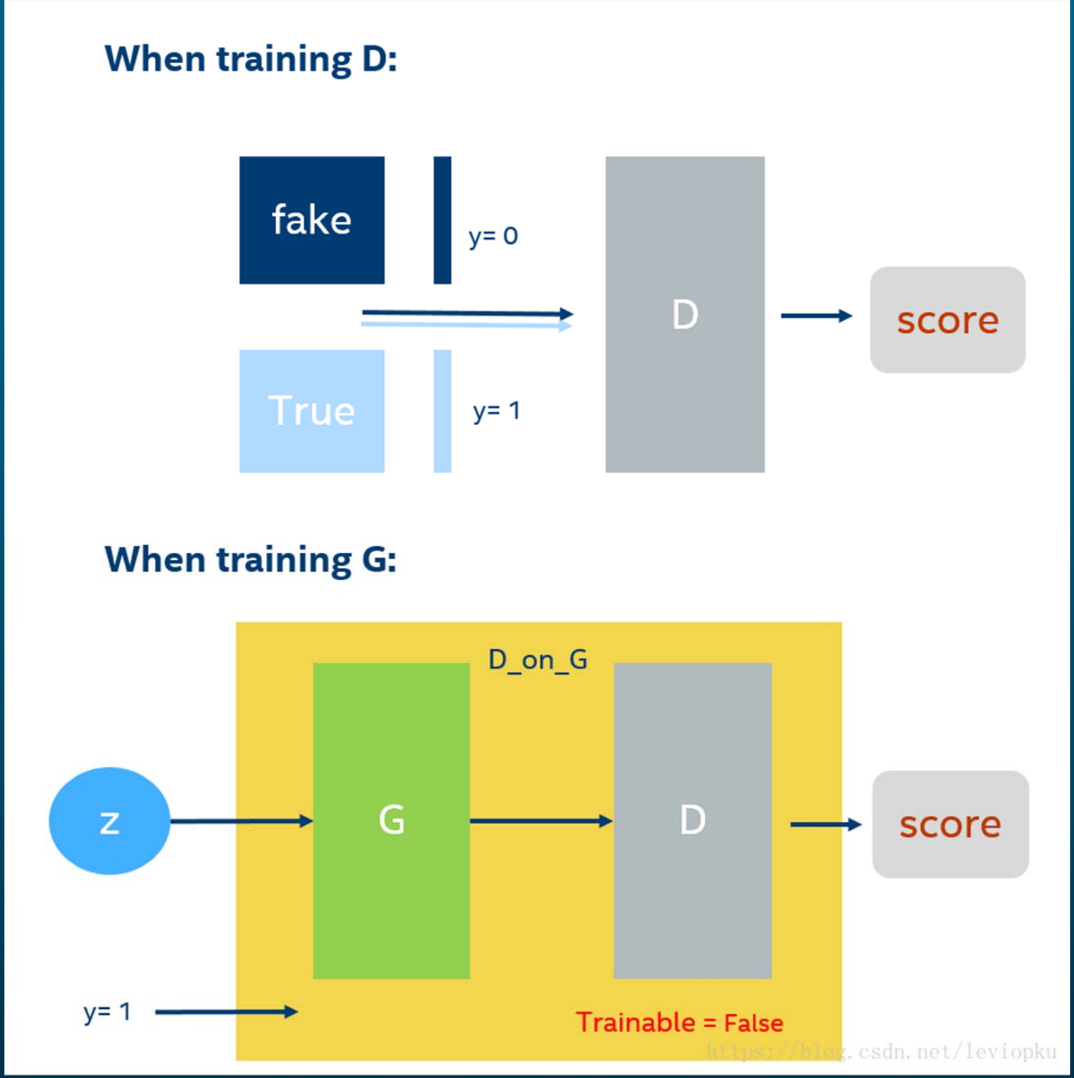


G是generator，生成器: 负责凭空捏造数据出来

D是discriminator，判别器: 负责判断数据是不是真数据

是两个网络的博弈过程。

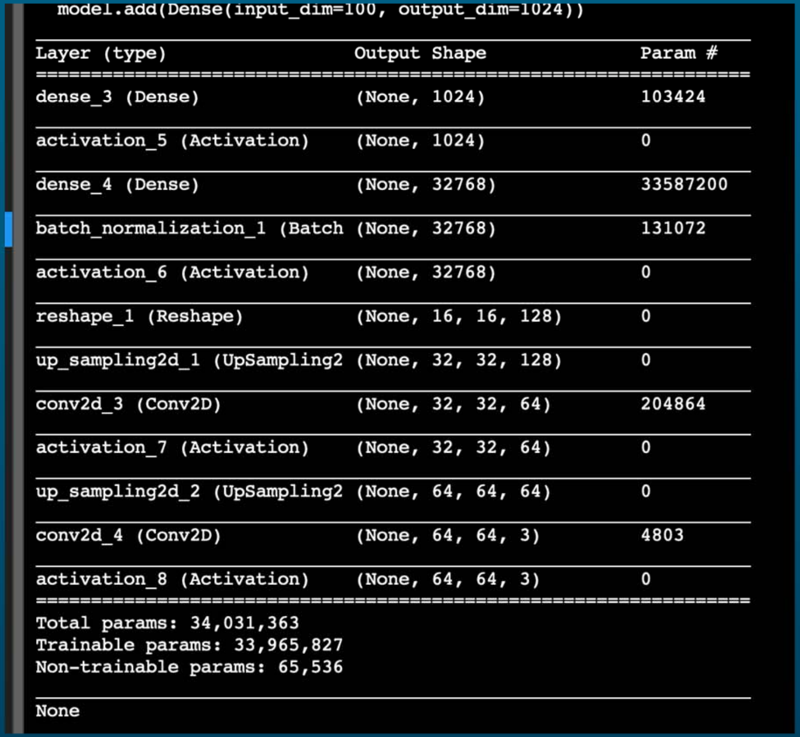
z是随机噪声(就是随机生成的一些数，也就是GAN生成图像的源头)。D通过真图和假图的数据(相当于天然label)，进行一个二分类神经网络训练。G根据一串随机数就可以捏造一个“假图像”出来，用这些假图去欺骗D，D负责辨别这是真图还是假图，会给出一个score。比如，G生成了一张图，在D这里得分很高，那证明G是很成功的；如果D能有效区分真假图，则G的效果还不太好，需要调整参数。GAN就是这么一个博弈的过程。



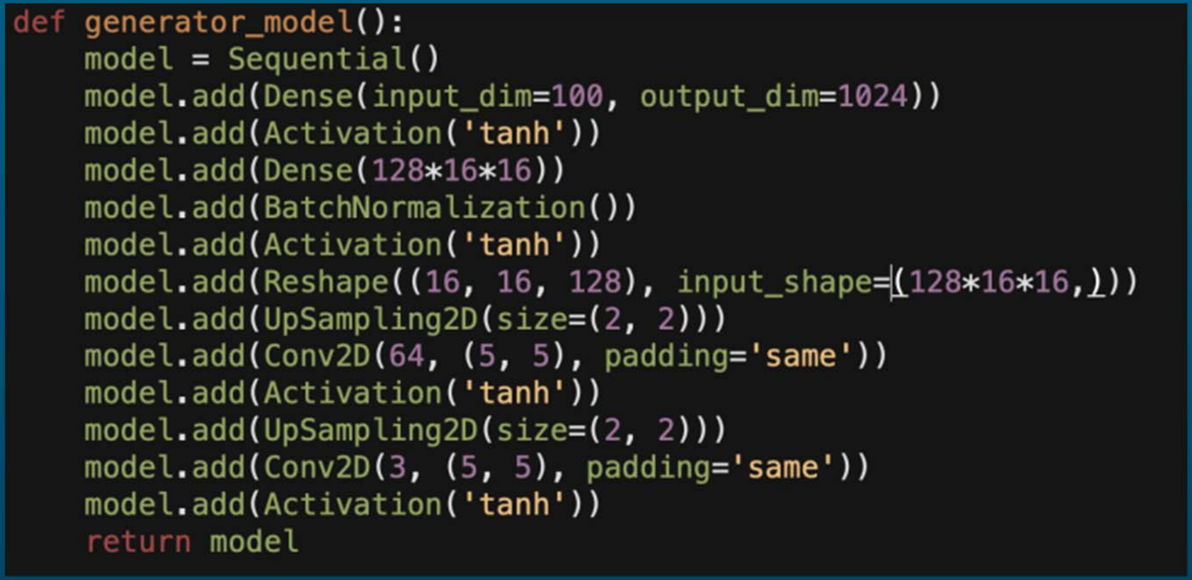
GAN的训练在同一轮梯度反传的过程中可以细分为2步，先训练D在训练G；注意不是等所有的D训练好以后，才开始训练G，因为D的训练也需要上一轮梯度反传中G的输出值作为输入。

当训练D的时候，上一轮G产生的图片，和真实图片，直接拼接在一起，作为x。然后根据，按顺序摆放0和1，假图对应0，真图对应1。然后就可以通过，x输入生成一个score（从0到1之间的数），通过score和y组成的损失函数，就可以进行梯度反传了。（我在图片上举的例子是batch = 1，len(y)=2*batch，训练时通常可以取较大的batch)

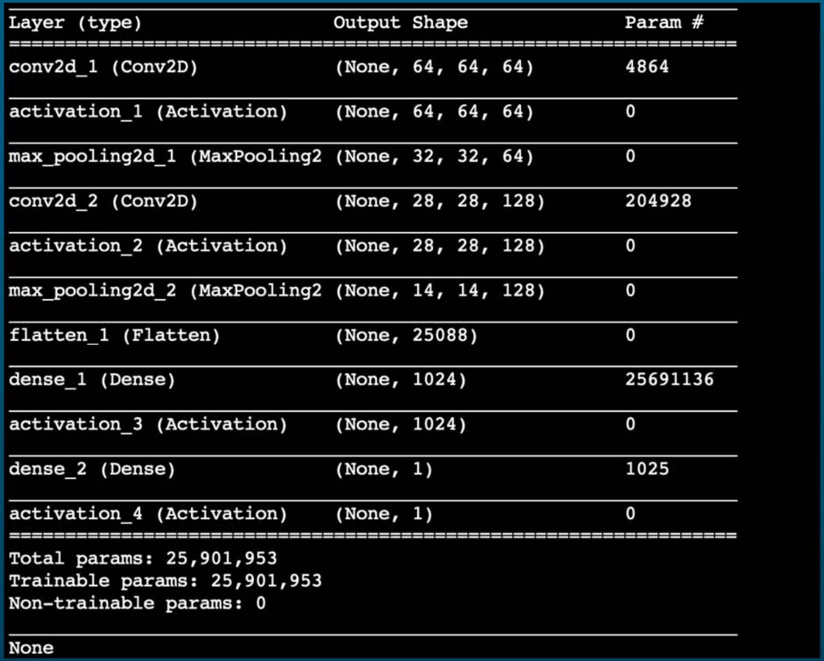
当训练G的时候， 需要把G和D当作一个整体，我在这里取名叫做’D_on_G’。这个整体(下面简称DG系统)的输出仍然是score。输入一组随机向量，就可以在G生成一张图，通过D对生成的这张图进行打分，这就是DG系统的前向过程。score=1就是DG系统需要优化的目标，score和y=1之间的差异可以组成损失函数，然后可以反向传播梯度。注意，这里的D的参数是不可训练的。这样就能保证G的训练是符合D的打分标准的。这就好比：如果你参加考试，你别指望能改变老师的评分标准



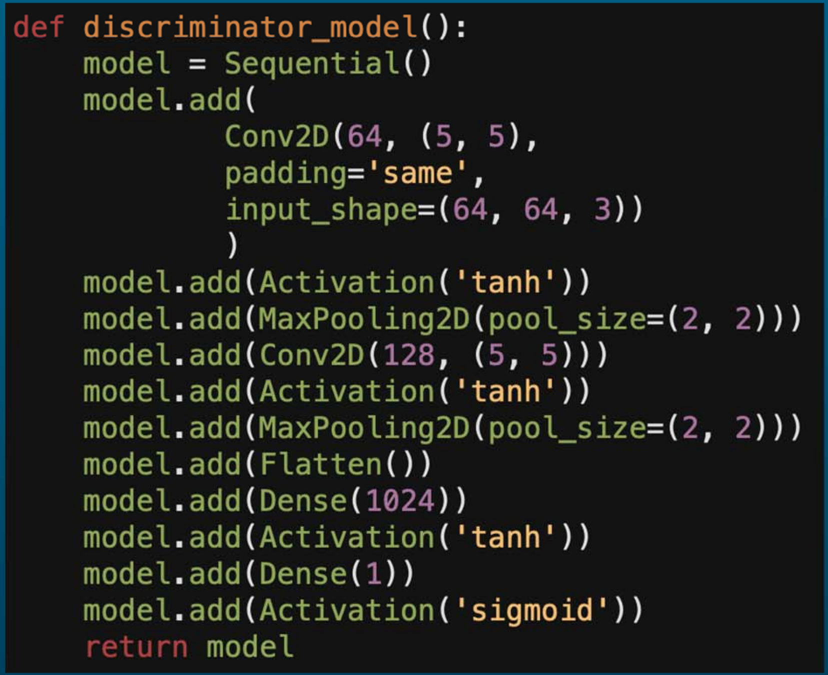
```
  model.add(Dense(input_dim=100, output_dim=1024))
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 1024)              103424
_________________________________________________________________
activation_5 (Activation)    (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 32768)             33587200
_________________________________________________________________
batch_normalization_1 (Batch (None, 32768)             131072
_________________________________________________________________
activation_6 (Activation)    (None, 32768)             0
_________________________________________________________________
reshape_1 (Reshape)          (None, 16, 16, 128)       0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        204864
_________________________________________________________________
activation_7 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 3)         4803
_________________________________________________________________
activation_8 (Activation)    (None, 64, 64, 3)         0
=================================================================
Total params: 34,031,363
Trainable params: 33,965,827
Non-trainable params: 65,536
_________________________________________________________________
None
```



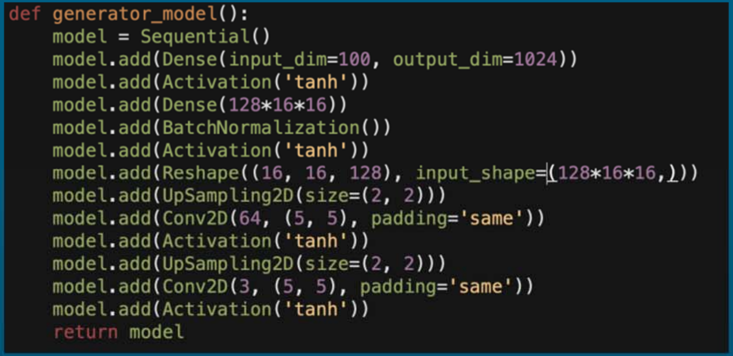
```python
def generator_model():
    model = Sequential()
    model.add(Dense(input_dim=100, output_dim=1024))
    model.add(Activation('tanh'))
    model.add(Dense(128*16*16))
    model.add(BatchNormalization())
    model.add(Activation('tanh'))
    model.add(Reshape((16, 16, 128), input_shape=(128*16*16,)))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(64, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(3, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    return model
```



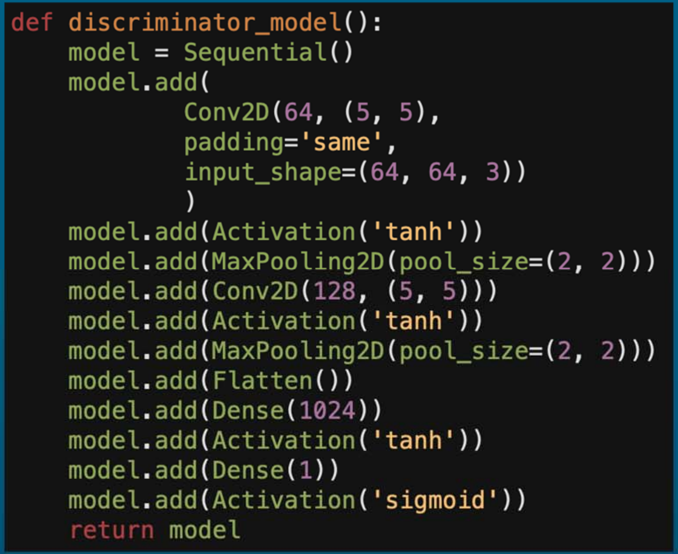
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 64)        4864      
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 64)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       204928    
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 128)       0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              25691136  
_________________________________________________________________
activation_3 (Activation)    (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 1025      
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0         
=================================================================
Total params: 25,901,953
Trainable params: 25,901,953
Non-trainable params: 0
_________________________________________________________________
None
```



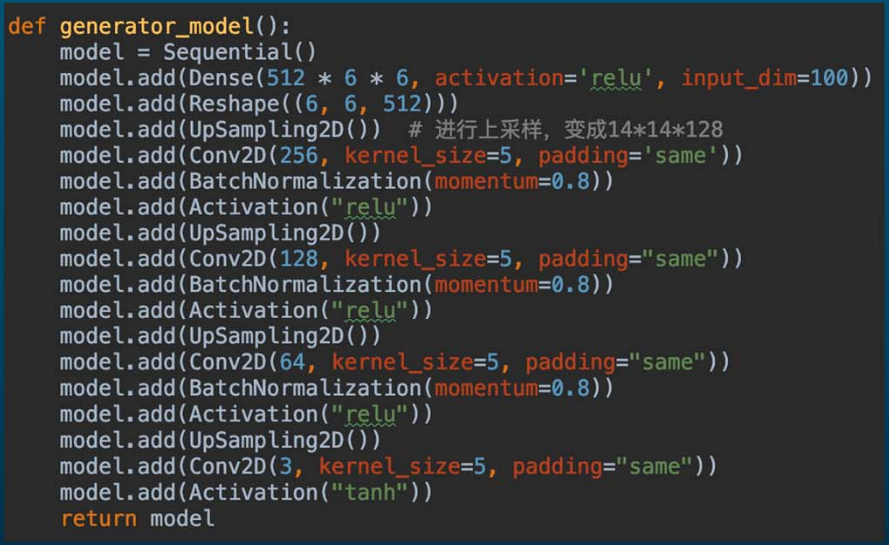
```python
def discriminator_model():
    model = Sequential()
    model.add(
            Conv2D(64, (5, 5),
            padding='same',
            input_shape=(64, 64, 3))
            )
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (5, 5)))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation('tanh'))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    return model
```



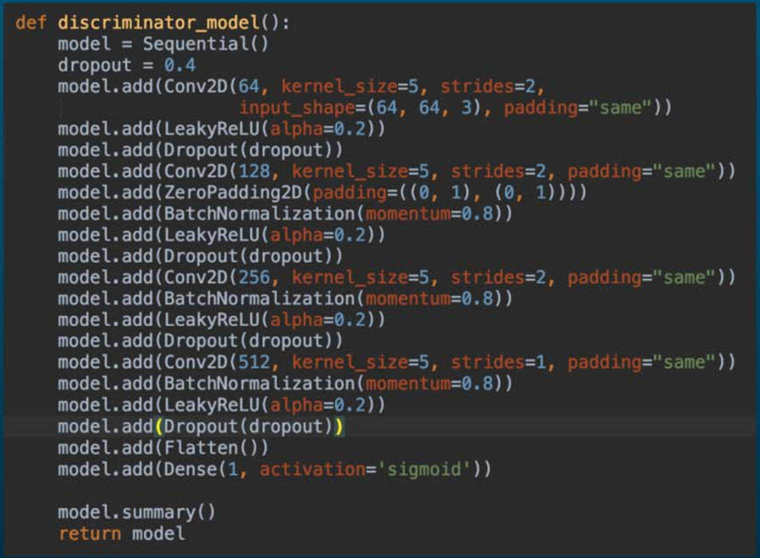
```python
def generator_model():
    model = Sequential()
    model.add(Dense(input_dim=100, output_dim=1024))
    model.add(Activation('tanh'))
    model.add(Dense(128*16*16))
    model.add(BatchNormalization())
    model.add(Activation('tanh'))
    model.add(Reshape((16, 16, 128), input_shape=(128*16*16,)))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(64, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(3, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    return model
```



```python
def discriminator_model():
    model = Sequential()
    model.add(
            Conv2D(64, (5, 5),
            padding='same',
            input_shape=(64, 64, 3))
            )
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (5, 5)))
    model.add(Activation('tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation('tanh'))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    return model
```

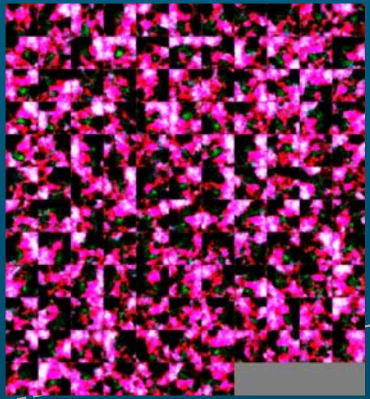


```python
def generator_model():
    model = Sequential()
    model.add(Dense(512 * 6 * 6, activation='relu', input_dim=100))
    model.add(Reshape((6, 6, 512)))
    model.add(UpSampling2D())  # 进行上采样，变成14*14*128
    model.add(Conv2D(256, kernel_size=5, padding='same'))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))
    model.add(UpSampling2D())
    model.add(Conv2D(128, kernel_size=5, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))
    model.add(UpSampling2D())
    model.add(Conv2D(64, kernel_size=5, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))
    model.add(UpSampling2D())
    model.add(Conv2D(3, kernel_size=5, padding="same"))
    model.add(Activation("tanh"))
    return model
```



```python
def discriminator_model():
    model = Sequential()
    dropout = 0.4
    model.add(Conv2D(64, kernel_size=5, strides=2,
                     input_shape=(64, 64, 3), padding="same"))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(dropout))
    model.add(Conv2D(128, kernel_size=5, strides=2, padding="same"))
    model.add(ZeroPadding2D(padding=((0, 1), (0, 1))))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(dropout))
    model.add(Conv2D(256, kernel_size=5, strides=2, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(dropout))
    model.add(Conv2D(512, kernel_size=5, strides=1, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(dropout))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    model.summary()
    return model
```



epoch0-200

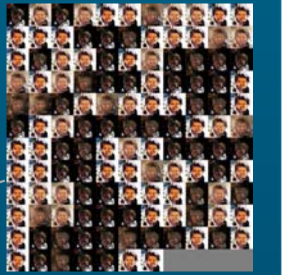
彩色的不可名状图

epoch1-400

有了人脸的轮廓但并不清晰

epoch560-200

更加细节



epoch560-400

感谢观看！