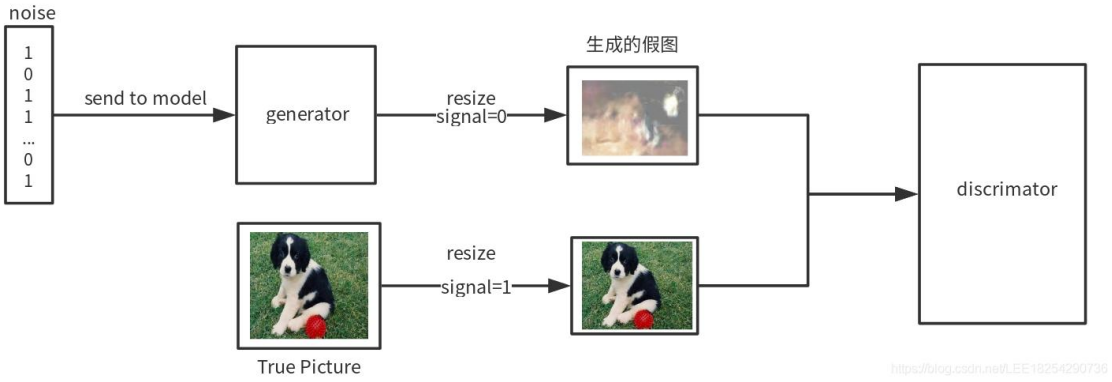
小白理解 GAN 网络

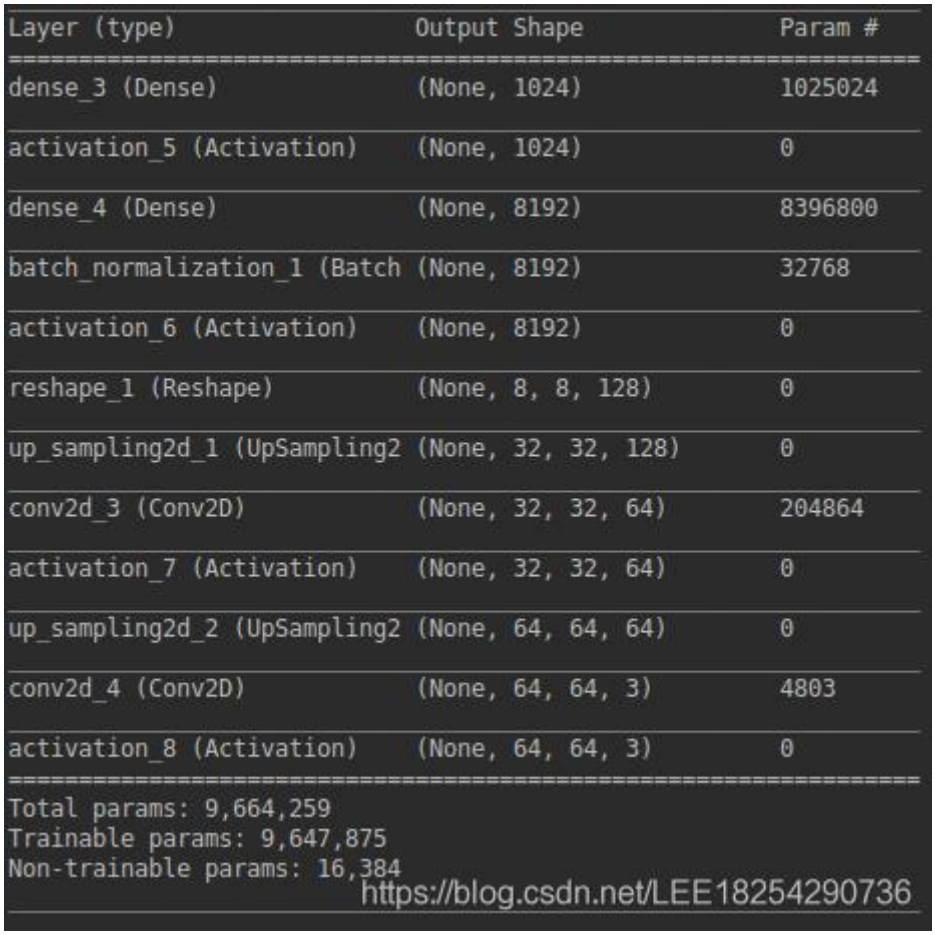
从一个小白的方式理解 GAN 网络（生成对抗网络），可以认为是一个造假机器，造出来的东西跟真的一样，下面开始讲如何造假：（主要讲解 GAN 代码，代码很简单）

我们首先以造小狗的假图片为例。

首先需要一个生成小狗图片的模型，我们称之为 generator，还有一个判断小狗图片是否是真假的判别模型 discrimator，



首先输入一个 1000 维的噪声，然后送入生成器，生成器的具体结构如下所示（不看也可以，看完全篇回来再看也一样）：



Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 1024)	1025024
activation_5 (Activation)	(None, 1024)	0
dense_4 (Dense)	(None, 8192)	8396800
batch_normalization_1 (Batch	(None, 8192)	32768
activation_6 (Activation)	(None, 8192)	0
reshape_1 (Reshape)	(None, 8, 8, 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: 9,664,259
Trainable params: 9,647,875
Non-trainable params: 16,384

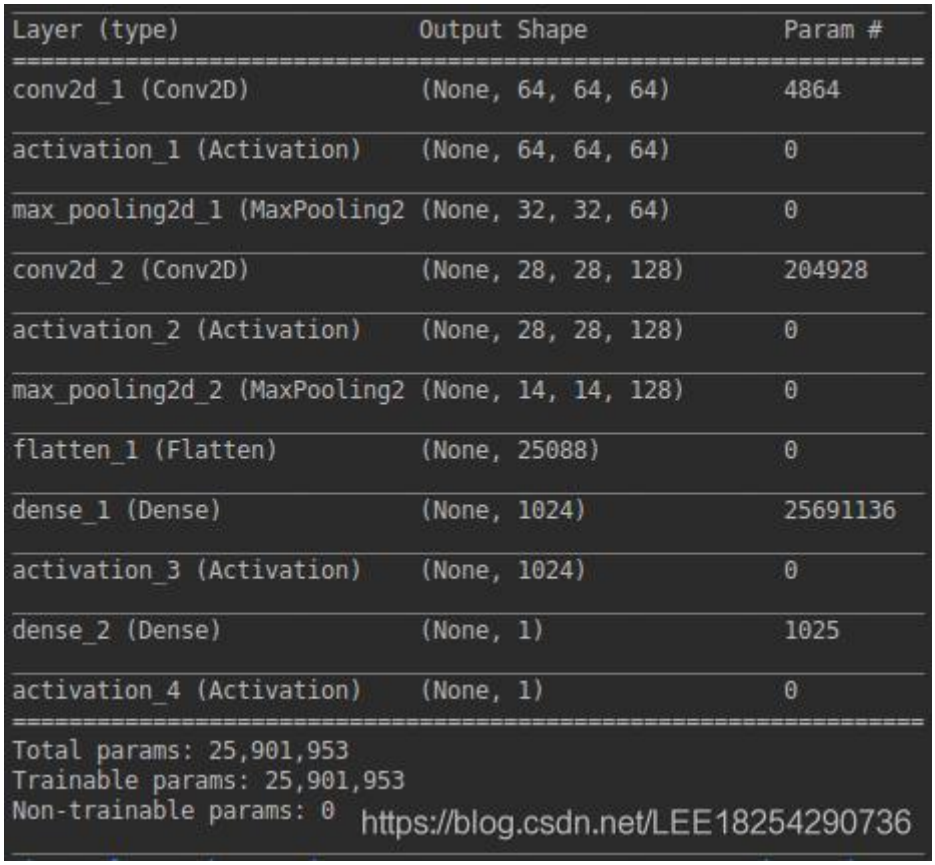
其实比较简单，代码如下所示：

```
def generator_model():
    model = Sequential()
    model.add(Dense(input_dim=1000, output_dim=1024))
    model.add(Activation('tanh'))
    model.add(Dense(128 * 8 * 8))
    model.add(BatchNormalization())
```

```
    model.add(Activation('tanh'))

    model.add(Reshape((8, 8, 128), input_shape=(8 * 8 * 128,)))

    model.add(UpSampling2D(size=(4, 4)))

    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
```

生成器接受一个 1000 维的随机生成的数组，然后输出一个 64×64×3 通道的图片数据。输出就是一个图片。不必太过深究，输入是 1000 个随机数字，输出是一张图片。

下面再看判别器代码与结构：



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

代码如下所示：

```
.

.       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'))

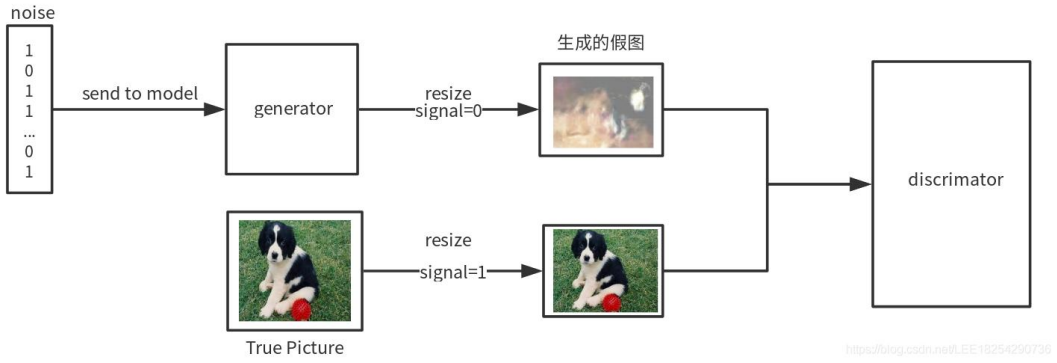
. model.add(Dense(1))

. model.add(Activation('sigmoid'))

. return model

输入是 64，64，3 的图片，输出是一个数 1 或者 0,代表图片是否是狗。

下面根据代码讲具体操作：



把真图与假图。进行拼接，然后打上标签，真图标签是 1,假图标签是 0，送入训练的网络。

```
# 随机生成的 1000 维的噪声

noise = np.random.uniform(-1, 1, size=(BATCH_SIZE, 1000))



# X_train 是训练的图片数据，这里取出一个 batchsize 的图片用于训练，这个是真图（64 张）

image_batch = X_train[index * BATCH_SIZE:(index + 1) * BATCH_SIZE]
```

```
# 这里是经过生成器生成的假图

generated_images = generator_model.predict(noise, verbose=0)



# 将真图与假图进行拼接

X = np.concatenate((image_batch, generated_images))



# 与 X 对应的标签，前 64 张图为真，标签是 1，后 64 张图是假图，标签为 0

y = [1] * BATCH_SIZE + [0] * BATCH_SIZE



# 把真图与假图的拼接训练数据 1 送入判别器进行训练判别器的准确度

d_loss = discriminator_model.train_on_batch(X, y)
```

这里要是看不明白的话可以结合别人的讲解结合来看。

在这里训练好之后，判别器的精度会不断提高。

下面是重头戏了，也是 GAN 网络的核心：

```
def generator_containing_discriminator(g, d):
```

```
    model = Sequential()

    model.add(g)

    # 判别器参数不进行修改

    d.trainable = False

    model.add(d)

    return model
```

他的网络结构如下所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_2 (Sequential)    (None, 64, 64, 3)         9664259
_________________________________________________________________
sequential_1 (Sequential)    (None, 1)                 25901953
=================================================================
Total params: 35,566,212
Trainable params: 9,647,875
Non-trainable params: 25,918,337
_________________________________________________________________
```

这个模型有生成器与判别器组成：看代码，这个模型上半部分是生成网络，下半部分是判别网络，生成网络首先生成假图，然后送入判别网络中进行判断，这里有一个 d.trainable=False，意思是，只调整生成器，判别的的参数不做更改。简直巧妙。

然后我们来看如何训练生成网络，这一块也是核心区域：

```
# 训练一个 batchsize 里面的数据
for index in range(int(X_train.shape[0]/BATCH_SIZE)):

    # 产生随机噪声
    noise = np.random.uniform(-1, 1, size=(BATCH_SIZE, 1000))

    # 这里面都是真图片
    image_batch = X_train[index*BATCH_SIZE:(index+1)*BATCH_SIZE]

    # 这里产生假图片
    generated_images = g.predict(noise, verbose=0)

    # 将真图片与假图片拼接在一起
    X = np.concatenate((image_batch, generated_images))

    # 前 64 张图片标签为 1,即真图，后 64 张照片为假图
    y = [1] * BATCH_SIZE + [0] * BATCH_SIZE

    # 对于判别器进行训练，不断提高判别器的识别精度
    d_loss = d.train_on_batch(X, y)

    # 再次产生随机噪声
```

```
            noise = np.random.uniform(-1, 1, (BATCH_SIZE, 1000))


            # 设置判别器的参数不可调整
            d.trainable = False


            # ××××××××××××××××××××××××××××××××××××××××××××××××××××
××××××××××××
            # 在此我们送入噪声，并认为这些噪声是真实的标签
            g_loss = generator_containing_discriminator.train_on_batch(noise, [1] *
BATCH_SIZE)
            # ××××××××××××××××××××××××××××××××××××××××××××××××××××
××××××××××××


            # 此时设置判别器可以被训练，参数可以被修改
            d.trainable = True


            # 打印损失值
            print("batch %d d_loss : %s, g_loss : %f" % (index, d_loss, g_loss))
```

重点在于这句代码

```
g_loss = generator_containing_discriminator.train_on_batch(noise, [1] *
BATCH_SIZE)
```

首先这个网络模型（定义在上面），先传入生成器中，然后生成器生成图片之后，把图片传入判别器中，标签此刻传入的是 1,真实的图片，但实际上是假图，此刻判别器就会判断为假图，然后模型就会不断调整生成器参数，此刻的判别器的参数被设置为为不可调整，d.trainable=False，所以为了不断降低 loss 值，模型就会一直调整生成器的参数，直到判别器认为这是真图。此刻判别器与生成器达到了一个平衡。也就是说生成器产生的假图，判别器已经分辨不出来了。所以继续迭代，提高判别器精度，如此往复循环，直到生成连人都辨别不了的图片。

最后我训练了大概 65 轮，实际上生成比较真实的狗的图片我估计可能上千轮了，当然不同的网络结构，所需要的迭代次数也不一样。我这个因为太费时间，就跑了大概，可以看出大概有个狗模样。这个是训练了 65 轮之后的效果：

以上就是全部的内容了。

https://github.com/jensleeGit/Kaggle_self_use/tree/master/Generative%20Dog%20Images