苏剑林OGAN源码解读

目录

网络结构

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 128, 128, 3)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 64)        3136      
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 64, 64, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 128)       131200    
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 128)       512       
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 128)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 256)       524544    
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 256)       1024      
```

```
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 512)         2097664
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 512)         2048
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 1024)        8389632
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 1024)        4096
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 4, 4, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2097280
=================================================================
Total params: 13,251,136
Trainable params: 13,247,296
Non-trainable params: 3,840
_________________________________________________________________
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to

================================================================================
==================
input_2 (InputLayer)            (None, 128)          0


________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 16384)        2113536     input_2[0][0]


________________________________________________________________________________
__________________
reshape_1 (Reshape)             (None, 4, 4, 1024)   0           dense_2[0][0]


________________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 128)          16512       input_2[0][0]


________________________________________________________________________________
__________________
dense_5 (Dense)                 (None, 128)          16512       input_2[0][0]


________________________________________________________________________________
__________________
batch_normalization_5 (BatchNor (None, 4, 4, 1024)   2048        reshape_1[0][0]


________________________________________________________________________________
__________________
dense_4 (Dense)                 (None, 1024)         132096      dense_3[0][0]


________________________________________________________________________________
__________________
```

```
dense_6 (Dense)                (None, 1024)         132096      dense_5[0][0]

__________________________________________________________________________________________________
scale_shift_1 (ScaleShift)     [(None, 4, 4, 1024), 0
batch_normalization_5[0][0]
                                                                dense_4[0][0]

                                                                dense_6[0][0]

__________________________________________________________________________________________________
activation_1 (Activation)      (None, 4, 4, 1024)   0
scale_shift_1[0][0]
__________________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 8, 8, 512)    8389120     activation_1[0]
[0]
__________________________________________________________________________________________________
dense_7 (Dense)                (None, 128)          16512       input_2[0][0]

__________________________________________________________________________________________________
dense_9 (Dense)                (None, 128)          16512       input_2[0][0]

__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 8, 8, 512)    1024
conv2d_transpose_1[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                (None, 512)          66048       dense_7[0][0]

__________________________________________________________________________________________________
dense_10 (Dense)               (None, 512)          66048       dense_9[0][0]

__________________________________________________________________________________________________
scale_shift_2 (ScaleShift)     [(None, 8, 8, 512),  0
batch_normalization_6[0][0]
                                                                dense_8[0][0]

                                                                dense_10[0][0]

__________________________________________________________________________________________________
activation_2 (Activation)      (None, 8, 8, 512)    0
scale_shift_2[0][0]
__________________________________________________________________________________________________
conv2d_transpose_2 (Conv2DTrans (None, 16, 16, 256)  2097408     activation_2[0]
[0]
__________________________________________________________________________________________________
dense_11 (Dense)               (None, 128)          16512       input_2[0][0]
```

```
____________________________________________________________________________________________________
__________________
dense_13 (Dense)                (None, 128)          16512       input_2[0][0]


____________________________________________________________________________________________________
__________________
batch_normalization_7 (BatchNor (None, 16, 16, 256)  512
conv2d_transpose_2[0][0]
____________________________________________________________________________________________________
__________________
dense_12 (Dense)                (None, 256)          33024       dense_11[0][0]


____________________________________________________________________________________________________
__________________
dense_14 (Dense)                (None, 256)          33024       dense_13[0][0]


____________________________________________________________________________________________________
__________________
scale_shift_3 (ScaleShift)      [(None, 16, 16, 256) 0
batch_normalization_7[0][0]
                                                                 dense_12[0][0]

                                                                 dense_14[0][0]


____________________________________________________________________________________________________
__________________
activation_3 (Activation)       (None, 16, 16, 256)  0
scale_shift_3[0][0]
____________________________________________________________________________________________________
__________________
conv2d_transpose_3 (Conv2DTrans (None, 32, 32, 128)  524416      activation_3[0]
[0]
____________________________________________________________________________________________________
__________________
dense_15 (Dense)                (None, 128)          16512       input_2[0][0]


____________________________________________________________________________________________________
__________________
dense_17 (Dense)                (None, 128)          16512       input_2[0][0]


____________________________________________________________________________________________________
__________________
batch_normalization_8 (BatchNor (None, 32, 32, 128)  256
conv2d_transpose_3[0][0]
____________________________________________________________________________________________________
__________________
dense_16 (Dense)                (None, 128)          16512       dense_15[0][0]


____________________________________________________________________________________________________
__________________
dense_18 (Dense)                (None, 128)          16512       dense_17[0][0]


____________________________________________________________________________________________________
__________________
scale_shift_4 (ScaleShift)      [(None, 32, 32, 128) 0
batch_normalization_8[0][0]
                                                                 dense_16[0][0]
```

```
                                                                 dense_18[0][0]

________________________________________________________________________________
__________________
activation_4 (Activation)       (None, 32, 32, 128)  0
scale_shift_4[0][0]
________________________________________________________________________________
__________________
conv2d_transpose_4 (Conv2DTrans (None, 64, 64, 64)   131136      activation_4[0]
[0]
________________________________________________________________________________
__________________
dense_19 (Dense)                (None, 128)          16512       input_2[0][0]

________________________________________________________________________________
__________________
dense_21 (Dense)                (None, 128)          16512       input_2[0][0]

________________________________________________________________________________
__________________
batch_normalization_9 (BatchNor (None, 64, 64, 64)   128
conv2d_transpose_4[0][0]
________________________________________________________________________________
__________________
dense_20 (Dense)                (None, 64)           8256        dense_19[0][0]

________________________________________________________________________________
__________________
dense_22 (Dense)                (None, 64)           8256        dense_21[0][0]

________________________________________________________________________________
__________________
scale_shift_5 (ScaleShift)      [(None, 64, 64, 64), 0
batch_normalization_9[0][0]

                                                                 dense_20[0][0]

                                                                 dense_22[0][0]

________________________________________________________________________________
__________________
activation_5 (Activation)       (None, 64, 64, 64)   0
scale_shift_5[0][0]
________________________________________________________________________________
__________________
conv2d_transpose_5 (Conv2DTrans (None, 128, 128, 3)  3075        activation_5[0]
[0]
________________________________________________________________________________
__________________
activation_6 (Activation)       (None, 128, 128, 3)  0
conv2d_transpose_5[0][0]
================================================================================
==================
Total params: 13,939,651
Trainable params: 13,935,683
Non-trainable params: 3,968
________________________________________________________________________________
__________________
```

```
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to

====================================================================================================
input_4 (InputLayer)            (None, 128)          0

____________________________________________________________________________________________________
model_2 (Model)                 (None, 128, 128, 3)  13939651    input_4[0][0]

____________________________________________________________________________________________________
input_3 (InputLayer)            (None, 128, 128, 3)  0

____________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 128, 128, 3)  0           model_2[1][0]

____________________________________________________________________________________________________
model_1 (Model)                 (None, 128)          13251136    input_3[0][0]

                                                                 model_2[1][0]

                                                                 lambda_1[0][0]

====================================================================================================
Total params: 27,190,787
Trainable params: 27,182,979
Non-trainable params: 7,808
____________________________________________________________________________________________________
```

源码阅读

读取数据集(glob、shuffle)

```
imgs = glob.glob('D:\\Storage\\datasets\\lfw\\*\\*.jpg')
np.random.shuffle(imgs)
```

老苏使用了glob.glob的方式读取数据。

glob

读取到的imgs其实只是每一个图片的路径

```
['D:\\Storage\\datasets\\lfw\\Aaron_Eckhart\\Aaron_Eckhart_0001.jpg',
'D:\\Storage\\datasets\\lfw\\Aaron_Guiel\\Aaron_Guiel_0001.jpg'....]
```

后序，实际读入python对象的时候，我们需要借助另外的函数(misc.imread、misc.imresize)

shuffle

从后序使用了np.random.shuffle将读入的图片路径打散，以下附上shuffle的用法

```
arr = np.arange(10)
np.random.shuffle(arr)
arr
```

```
[1 7 5 2 9 4 3 6 0 8]
```

```
arr = np.arange(9).reshape((3, 3))
np.random.shuffle(arr)
arr
```

```
array([[3, 4, 5],
       [6, 7, 8],
       [0, 1, 2]])
```

可以看出shuffle可以只打乱numpy数组的第一维

misc

```
img_dim = 128


def imread(f):
    x = misc.imread(f, mode='RGB')
    print(x.shape)
    x = misc.imresize(x, (img_dim, img_dim))
    x = x.astype(np.float32)
    return x / 255 * 2 - 1



imgs = glob.glob('D:\\Storage\\datasets\\lfw\\*\\*.jpg')
f = imgs[0]
data = imread(f)
print(data.shape)
```

```
(250, 250, 3)
(128, 128, 3)
```

使用misc.imread 以三通道的方式读取图片，f就是传入图片的路径。

可见最初的图片大小是250x250x3，但是使用imresize之后，可以统一大小为128x128x3

数据的预处理

img_dim 处理为 num_layers(卷积层、逆卷积层个数),max_num_channels(单个卷积层和逆卷积层的卷积输出通道数)

```
img_dim = 128
z_dim = 128
batch_size = 64

num_layers = int(np.log2(img_dim)) - 3
max_num_channels = img_dim * 8
f_size = img_dim // 2 ** (num_layers + 1)
```

np.log2 是对numpy数组元素进行取对数的操作

```
>>> x = np.array([0, 1, 2, 2**4])
>>> np.log2(x)
array([-Inf,   0.,   1.,   4.])
```

num_layers

$$NumLayer = (int)(log_2(ImageDim)) - 3$$

num_layers 与 img_dim呈现正相关，只要图片的维度越高，num_layers数值越大，而num_layers本身可能代表的是网络的层数。

在这里img_dim = 128， 则 num_layers = 7 - 3 = 4

---

1. 作用于f_size
2. 作用于解码器的卷积层数
3. 作用于生成器的卷积层数

max_num_channels

$$MaxNumChannels = ImageDim * 8$$

在这里max_num_channels同样与img_dim正相关。就等于img_dim * 8

此处的max_num_channels = 1024

---

1. 作用于生成器的每一个逆卷积层的输出通道
2. 作用于解码器的每一个卷积层的输出通道

f_size

$$Fsize = ImageDim/2^{Numlayers+1}$$

f_size 在此处等于 128/(32) = 4

f_size 是生成器初始层参数个数的重要参数

```
z_in = Input(shape=(z_dim,))
z = z_in

z = Dense(f_size ** 2 * max_num_channels,
          kernel_initializer=RandomNormal(0, 0.02))(z)
z = Reshape((f_size, f_size, max_num_channels))(z)
```

$$zDim-> f^2 * channels-> (f, f, channels)$$

编码器

编码器的代码

```
# 编码器
x_in = Input(shape=(img_dim, img_dim, 3))
x = x_in

for i in range(num_layers + 1):
    num_channels = max_num_channels // 2 ** (num_layers - i)
    x = Conv2D(num_channels,
               (4, 4),
               strides=(2, 2),
               padding='same',
               kernel_initializer=RandomNormal(0, 0.02))(x)
    if i > 0:
        x = BatchNormalization()(x)
    x = LeakyReLU(0.2)(x)

x = Flatten()(x)
x = Dense(z_dim,
          kernel_initializer=RandomNormal(0, 0.02))(x)

e_model = Model(x_in, x)
e_model.summary()
```

$$numChannels = maxNumChannels/2^{numLayers-i}$$

```
num_channels = max_num_channels // 2 ** (num_layers - i)
```

num_channels

num_channels的取值变化。首先i的取值是 [0,1,....,layers]

那么numLayers - i 的取值是[layers,layers - 1, ... ,1 , 0]

注意的是此处num_layers= 4

所以我们可以推出numChannels的取值是 [64,128,256,512,1024]

数据形状

我们发现这样使用最少的代码来构建不断压缩的卷积网络。图片的大小经过该五层网络后，则按如下规律变化

$$128->64->32->16->8->4$$

无论卷积核大小为多少，**只要strides = 2，padding选择 same。**

那么就可以实现**每次大小缩减一半。**

生成器

生成器的代码

```
# 生成器
```

```
z_in = Input(shape=(z_dim,))
z = z_in

z = Dense(f_size ** 2 * max_num_channels,
          kernel_initializer=RandomNormal(0, 0.02))(z)
z = Reshape((f_size, f_size, max_num_channels))(z)
z = SelfModulatedBatchNormalization(z, z_in)
z = Activation('relu')(z)

for i in range(num_layers):
    num_channels = max_num_channels // 2 ** (i + 1)
    z = Conv2DTranspose(num_channels,
                        (4, 4),
                        strides=(2, 2),
                        padding='same',
                        kernel_initializer=RandomNormal(0, 0.02))(z)
    z = SelfModulatedBatchNormalization(z, z_in)
    z = Activation('relu')(z)

z = Conv2DTranspose(3,
                    (4, 4),
                    strides=(2, 2),
                    padding='same',
                    kernel_initializer=RandomNormal(0, 0.02))(z)
z = Activation('tanh')(z)

g_model = Model(z_in, z)
g_model.summary()
```

值得注意的是生成器在最开始的维度就被拓展为了

$$维度 = fSize^2 * maxNumChannels$$

i 的取值范围是[0,1,2,3]

num_channels的取值范围是[512,256,128,64]

初始维度是4 x 4 x 1024， 使用逆卷积，对应上之前的num_channels

维度变化为

```
8 x 8 x 512 -> 16 x 16 x 256 -> 32 x 32 x 128 -> 64 x 64 x 64
```

最后经过一次转置

```
64 x 64 x 64 -> 128 x 128 x 3
```

整合模型（生成器和编码器）

模型整合部分

```
# 整合模型
x_in = Input(shape=(img_dim, img_dim, 3))
z_in = Input(shape=(z_dim,))

x_real = x_in
```

```
x_fake = g_model(z_in)
x_fake_ng = Lambda(K.stop_gradient)(x_fake)

z_real = e_model(x_real)
z_fake = e_model(x_fake)
z_fake_ng = e_model(x_fake_ng)

train_model = Model([x_in, z_in],
                    [z_real, z_fake, z_fake_ng])

z_real_mean = K.mean(z_real, 1, keepdims=True)
z_fake_mean = K.mean(z_fake, 1, keepdims=True)
z_fake_ng_mean = K.mean(z_fake_ng, 1, keepdims=True)
```

这里需要理解一下x_fake_ng，这是x_fake求梯度得到的结果。

keras.backend.stop_gradient可以求的所传入参数的梯度

至于使用Lambda层， 不太清楚不使用Lambda层会有什么后果。

z_real 与 z_fake 都是 (? , 128) 形状，而z_real_mean 是 (?, 1)的形状

损失函数部分

```
t1_loss = z_real_mean - z_fake_ng_mean
t2_loss = z_fake_mean - z_fake_ng_mean
z_corr = correlation(z_in, z_fake)
qp_loss = 0.25 * t1_loss[:, 0] ** 2 / K.mean((x_real - x_fake_ng) ** 2, axis=[1,
2, 3])

train_model.add_loss(K.mean(t1_loss + t2_loss - 0.5 * z_corr) + K.mean(qp_loss))
train_model.compile(optimizer=RMSprop(1e-4, 0.99))
train_model.metrics_names.append('t_loss')
train_model.metrics_tensors.append(K.mean(t1_loss))
train_model.metrics_names.append('z_corr')
train_model.metrics_tensors.append(K.mean(z_corr))
```

$$z_{in}\text{经过}model_g\text{再经过}model_z\text{得到}z_{fake}\text{。我们需要比较}z_{in}\text{与}z_{fake}\text{的}correlation$$

```
def correlation(x, y):
    x = x - K.mean(x, 1, keepdims=True)
    y = y - K.mean(y, 1, keepdims=True)
    x = K.l2_normalize(x, 1)
    y = K.l2_normalize(y, 1)
    return K.sum(x * y, 1, keepdims=True)
```

其中l2_normalize函数的作用如下

$$x = [1, 2, 3]\text{对其求}l2 - normalize\text{的结果是}:[\frac{1}{1^2+2^2+3^2}, \frac{2}{1^2+2^2+3^2}, \frac{3}{1^2+2^2+3^2}]$$

其运算意义，只知道最终这个是一个相关性

t1_loss 本来就是(?, 1)形状的tensor

$$ave[xReal - (xFake)']^2$$

训练模型

创建generator

```
主要是__iter__方法返回迭代器
__len__方法返回：训练集训练完毕（N个batch）需要多少步
```

```
class img_generator:
    def __init__(self, imgs, mode='gan', batch_size=64):
        self.imgs = imgs
        self.batch_size = batch_size
        self.mode = mode
        if len(imgs) % batch_size == 0:
            self.steps = len(imgs) // batch_size
        else:
            self.steps = len(imgs) // batch_size + 1

    def __len__(self):
        return self.steps

    def __iter__(self):
        X = []
        while True:
            np.random.shuffle(self.imgs)
            for i, f in enumerate(self.imgs):
                X.append(imread(f, self.mode))
                if len(X) == self.batch_size or i == len(self.imgs) - 1:
                    X = np.array(X)
                    if self.mode == 'gan':
                        Z = np.random.randn(len(X), z_dim)
                        yield [X, Z], None
                    elif self.mode == 'fid':
                        yield X
                    X = []
```

使用fit_generator

```
img_data = img_generator(imgs, 'gan', batch_size)
train_model.fit_generator(img_data.__iter__(),
                          steps_per_epoch=len(img_data),
                          epochs=1000,
                          callbacks=[trainer])
```

注意

这里注意，在'gan'模式下，generator 返回的迭代器是一个（ [X , Z] , None ）形式的返回值

```
train_model = Model([x_in, z_in],
                    [z_real, z_fake, z_fake_ng])
```

这里对应的是规划训练模型的时候，使用了这个结构