Application of Generative Adversarial Networks in Face Generation

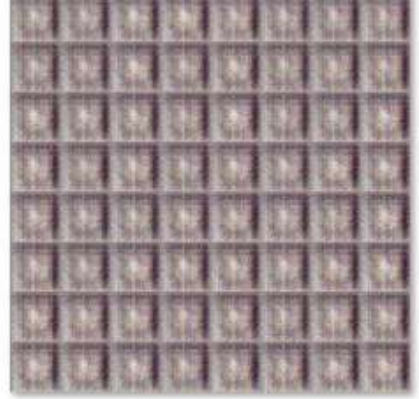
TEAM:爱吃菠萝的西瓜组(PINEAPPLE-LOVING WATERMELON GROUP)

CAPTAIN:张乐晓(LEXIAO ZHANG)

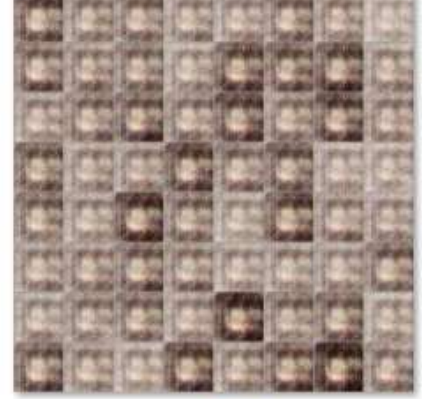
MEMBER:翁芳胜(FANGSHENG WENG) 马佳坚(JIAJIAN MA) 周逸展(YIZHAN ZHOU)


train_00_0000.png
train_00_0050.png
train_00_0100.png
train_00_0150.png
train_00_0200.png
train_00_0250.png
train_00_0300.png
train_00_0350.png
train_00_0400.png
train_00_0450.png
train_00_0500.png
train_00_0550.png
train_00_0600.png
train_00_0650.png
train_00_0700.png
train_00_0750.png
train_01_0000.png
train_01_0050.png
train_01_0100.png
train_01_0150.png
train_01_0200.png
train_01_0250.png
train_01_0300.png
train_01_0350.png
train_01_0400.png
train_01_0450.png
train_01_0500.png
train_01_0550.png
train_01_0600.png
train_01_0650.png
train_01_0700.png
train_01_0750.png
train_02_0000.png
train_02_0050.png
train_02_0100.png
train_02_0150.png
train_02_0200.png
train_02_0250.png
train_02_0300.png
train_02_0350.png
train_02_0400.png
train_02_0450.png
train_02_0500.png
train_02_0550.png
train_02_0600.png
train_02_0650.png
train_02_0700.png
train_02_0750.png
train_03_0000.png
train_03_0050.png
train_03_0100.png
train_03_0150.png
train_03_0200.png
train_03_0250.png
train_03_0300.png
train_03_0350.png
train_03_0400.png
train_03_0450.png
train_03_0500.png
train_03_0550.png
train_03_0600.png
train_03_0650.png
train_03_0700.png
train_03_0750.png
train_04_0000.png
train_04_0050.png
train_04_0100.png
train_04_0150.png
train_04_0200.png
train_04_0250.png
train_04_0300.png
train_04_0350.png
train_04_0400.png
train_04_0450.png
train_04_0500.png
train_04_0550.png
train_04_0600.png
train_04_0650.png
train_04_0700.png
train_04_0750.png
train_05_0000.png
train_05_0050.png
train_05_0100.png
train_05_0150.png
train_05_0200.png
train_05_0250.png
train_05_0300.png
train_05_0350.png
train_05_0400.png
train_05_0450.png
train_05_0500.png
train_05_0550.png
train_05_0600.png
train_05_0650.png
train_05_0700.png
train_05_0750.png
train_06_0000.png
train_06_0050.png
train_06_0100.png
train_06_0150.png
train_06_0200.png
train_06_0250.png
train_06_0300.png
train_06_0350.png
train_06_0400.png
train_06_0450.png
train_06_0500.png
train_06_0550.png
train_06_0600.png
train_06_0650.png
train_06_0700.png
train_06_0750.png
train_07_0000.png
train_07_0050.png
train_07_0100.png
train_07_0150.png
train_07_0200.png
train_07_0250.png
train_07_0300.png
train_07_0350.png
train_07_0
train_07_0
train_07_0
train_07_0
train_07_0
train_07_0
train_07_0
train_07_0
train_08_0
train_08_0
train_08_0
train_08_0
train_08_0
train_08_0
train_08_0

train_00_0000.png

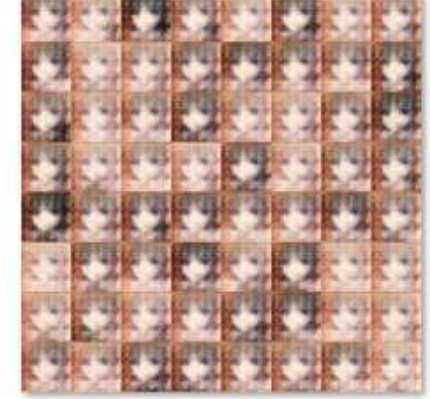


train_00_0050.png

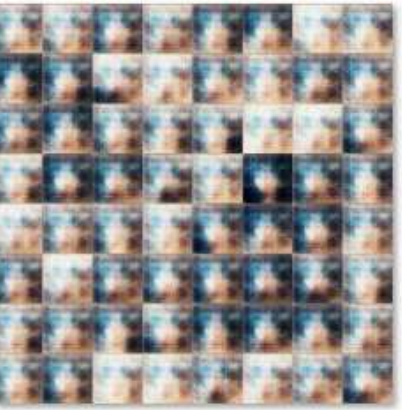


train_00_0100.png

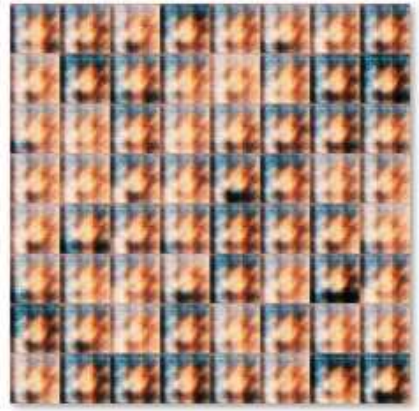
train_00_0150.png



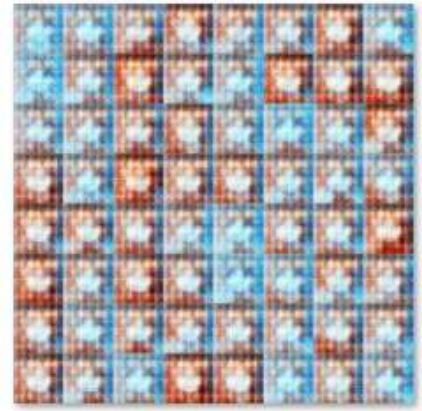
train_00_0200.png



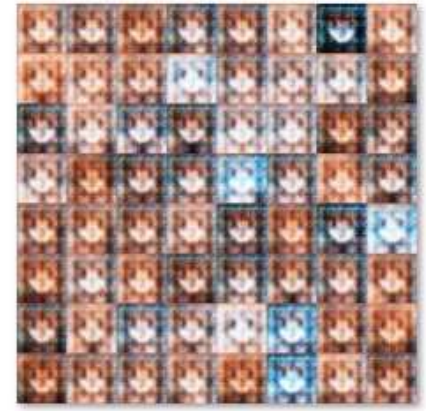
train_00_0250.png



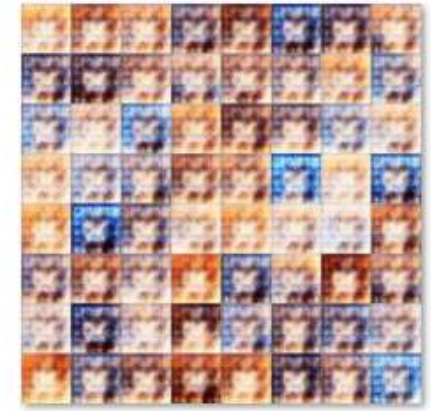
train_00_0300.png



train_00_0350.png



train_00_0400.png



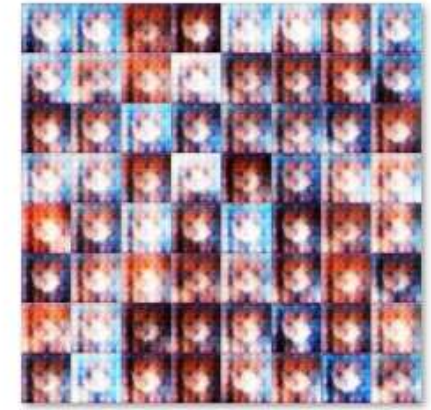
train_00_0450.png

train_00_0500.png

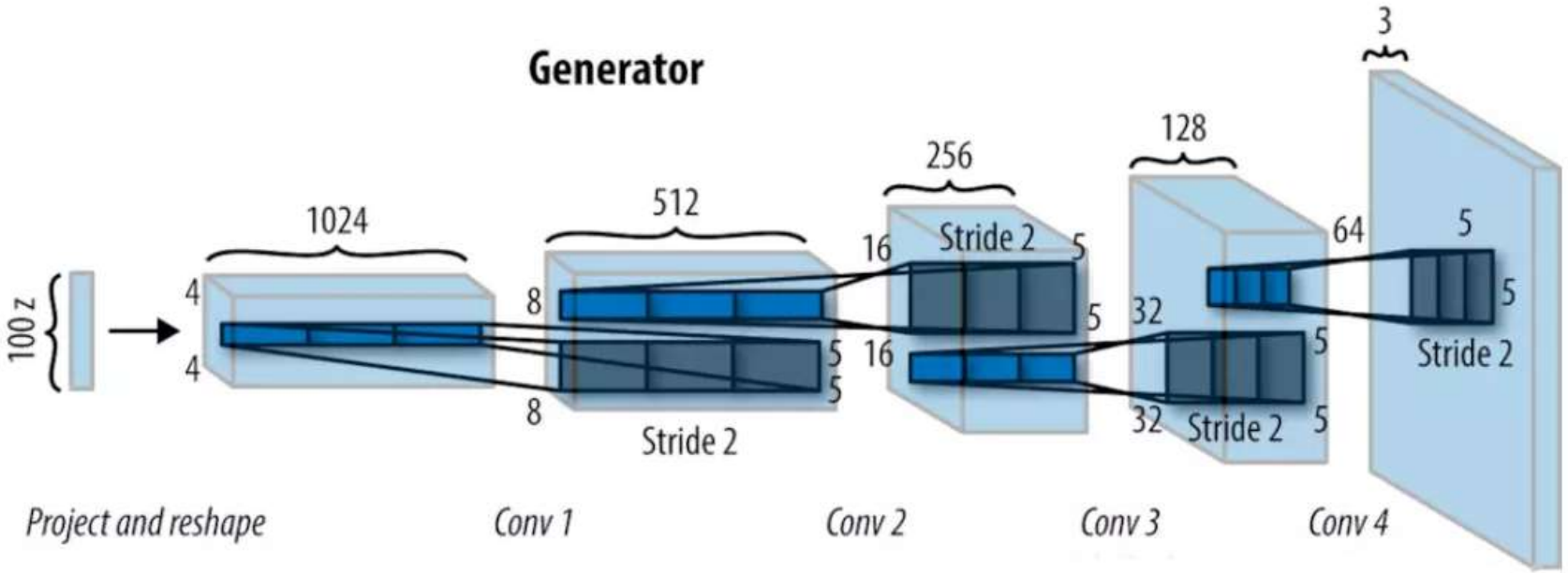
train_00_0550.png

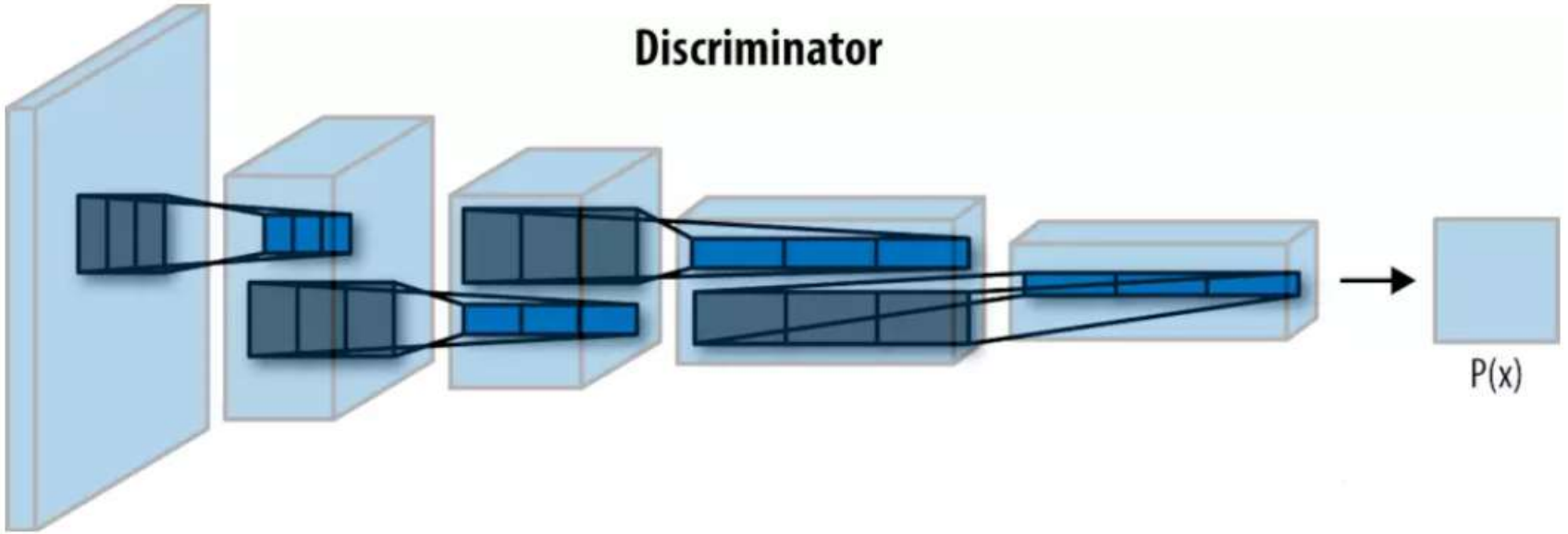
train_00_0600.png

train_00_0650.png



train_00_0700.png


Generator
100 z
4
4
1024
8
8
512
Stride 2
16
16
256
Stride 2
5
5
32
32
128
Stride 2
64
3
5
5
Stride 2
Project and reshape
Conv 1
Conv 2
Conv 3
Conv 4


Discriminator
P(x)

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

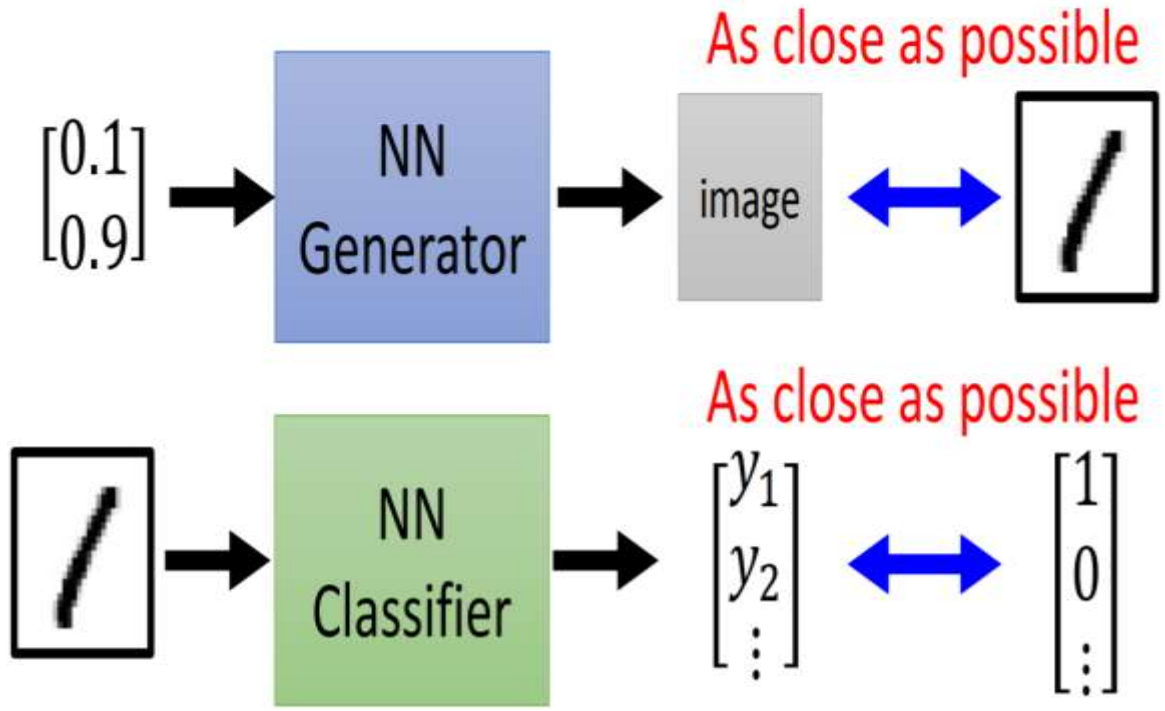
  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



```
def detect_faces(image_path):

        image=cv2.imread(image_path)
        image_grey=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

        faces = FACE_CASCADE.detectMultiScale(image_grey,scaleFactor=1.16,minNeighbors=5,minSize=(25,25),flags=0)

        for x,y,w,h in faces:
            sub_img=image[y-10:y+h+10,x-10:x+w+10]
            sub_img=cv2.resize(image,(48,48))
            os.chdir("C:\\Users\\MYDELL\\Face-Detect\\Extracted")

cv2.imwrite(str(randint(0,10000))+".jpg",sub_img)
            os.chdir("C:\\Users\\MYDELL\\kunkun")
```

```
def dowmloadPicture(html, keyword):
    global num
    # t =0
    pic_url = re.findall('"objURL":"(.*?)",', html, re.S)  # 先利用正则表达式找到图片url
    print('找到关键词:' + keyword + '的图片，即将开始下载图片...')
    for each in pic_url:
        print('正在下载第' + str(num + 1) + '张图片，图片地址:' + str(each))
        try:
            if each is not None:
                pic = requests.get(each, timeout=7)
            else:
                continue
        except BaseException:
            print('错误，当前图片无法下载')
            continue
        else:
            string = file + r'\\' + keyword + '_' + str(num) + '.jpg'
            fp = open(string, 'wb')
            fp.write(pic.content)
            fp.close()
            num += 1
        if num >= numPicture:
            return
```

output

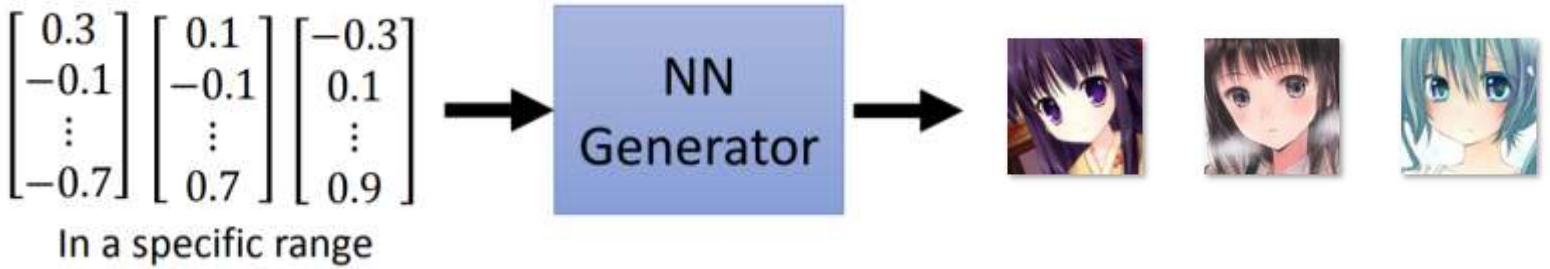
```
'''图像预处理'''
    image_decode = tf.cast(image_decode, tf.float32)/127.5-1

    '''生成batch图像'''
    # 随机获得batch_size大小的图像和label
    images, labels = tf.train.shuffle_batch([image_decode, label],
                                            batch_size=BATCH_SIZE,
                                            num_threads=1,
                                            capacity=10000 + 3 * BATCH_SIZE，# 队列最大容量
                                            min_after_dequeue=1000)
    return images, labels
```

```
def batch_from_tfr(image_height=IMAGE_HEIGHT,
                   image_width=IMAGE_WIDTH,
                   image_depth=IMAGE_DEPTH):
    """从TFR文件读取batch数据"""

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

    '''读取TFR数据并还原为uint8的图片'''
    file_names = glob.glob(os.path.join(TFR_PATH, '{0}.tfrecords_*_of_*')
                           .format(IMAGE_PATH.split('/')[-1]))
    filename_queue = tf.train.string_input_producer(file_names, num_epochs=NUM_EPOCHS,
shuffle=True)

    reader = tf.TFRecordReader()
    _, serialized_example = reader.read(filename_queue)
    features = tf.parse_single_example(
        serialized_example,
        features={
            'image': tf.FixedLenFeature([], tf.string),
            'label': tf.FixedLenFeature([], tf.int64)
        })
    image = features['image']
    image_decode = tf.decode_raw(image, tf.uint8)
    # 解码会变为一维数组，所以这里设定shape时需要设定为一维数组
    image_decode.set_shape([image_height * image_width * image_depth])
    image_decode = tf.reshape(image_decode, [image_height, image_width, image_depth])
    label = tf.cast(features['label'], tf.int32)
```

```
./Data_Set/kunkun//7723.jpg 0
./Data_Set/kunkun//6861.jpg 0
./Data_Set/kunkun//9318.jpg 0
./Data_Set/kunkun//8150.jpg 0
./Data_Set/kunkun//6122.jpg 0
./Data_Set/kunkun//8550.jpg 0
./Data_Set/kunkun//3634.jpg 0
./Data_Set/kunkun//7722.jpg 0
./Data_Set/kunkun//5300.jpg 0
./Data_Set/kunkun//6793.jpg 0
./Data_Set/kunkun//6029.jpg 0
./Data_Set/kunkun//956.jpg 0
./Data_Set/kunkun//7787.jpg 0
./Data_Set/kunkun//5251.jpg 0
./Data_Set/kunkun//8983.jpg 0
./Data_Set/kunkun//3902.jpg 0
./Data_Set/kunkun//1327.jpg 0
./Data_Set/kunkun//8034.jpg 0
./Data_Set/kunkun//4327.jpg 0
./Data_Set/kunkun//9436.jpg 0
./Data_Set/kunkun//770.jpg 0
./Data_Set/kunkun//6452.jpg 0
./Data_Set/kunkun//4081.jpg 0
./Data_Set/kunkun//2365.jpg 0
./Data_Set/kunkun//2368.jpg 0
./Data_Set/kunkun//7831.jpg 0
./Data_Set/kunkun//3621.jpg 0
./Data_Set/kunkun//2442.jpg 0
./Data_Set/kunkun//9865.jpg 0
./Data_Set/kunkun//5697.jpg 0
./Data_Set/kunkun//9390.jpg 0
./Data_Set/kunkun//7367.jpg 0
./Data_Set/kunkun//6076.jpg 0
```

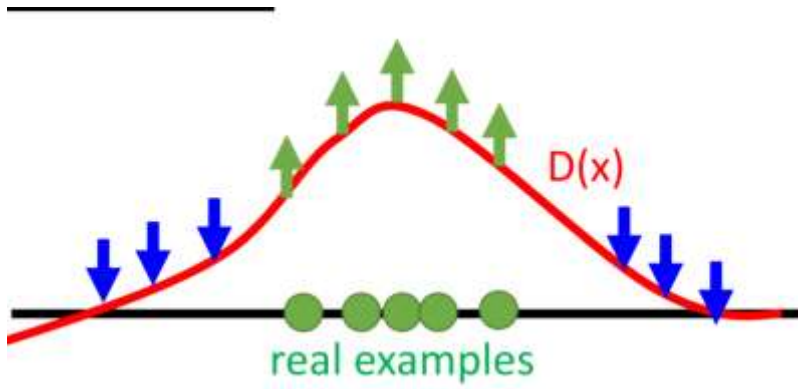
generator

```
def generator(self, z, train=True):
    """生成器"""
    with tf.variable_scope("generator") as scope:
        if not train:
            scope.reuse_variables()

        s_h, s_w = self.input_height, self.input_width
        s_h2, s_w2 = conv_out_size_same(s_h, 2), conv_out_size_same(s_w, 2)
        s_h4, s_w4 = conv_out_size_same(s_h2, 2), conv_out_size_same(s_w2, 2)
        s_h8, s_w8 = conv_out_size_same(s_h4, 2), conv_out_size_same(s_w4, 2)
        s_h16, s_w16 = conv_out_size_same(s_h8, 2), conv_out_size_same(s_w8, 2)

        z_ = linear(
            z, self.gf_dim * 8 * s_h16 * s_w16, scope='g_h0_lin')

        h0 = tf.reshape(z_, [-1, s_h16, s_w16, self.gf_dim * 8])
        h0 = tf.nn.relu(batch_normal(h0, train=train, scope='g_bn0'))

        h1 = deconv2d(h0, [self.batch_size, s_h8, s_w8, self.gf_dim * 4],
scope='g_h1')
        h1 = tf.nn.relu(batch_normal(h1, train=train, scope='g_bn1'))
        h2 = deconv2d(h1, [self.batch_size, s_h4, s_w4, self.gf_dim * 2], scope='g_h2')
        h2 = tf.nn.relu(batch_normal(h2, train=train, scope='g_bn2'))

        h3 = deconv2d(h2, [self.batch_size, s_h2, s_w2, self.gf_dim * 1],
scope='g_h3')
        h3 = tf.nn.relu(batch_normal(h3, train=train, scope='g_bn3'))

        h4 = deconv2d(h3, [self.batch_size, s_h, s_w, self.c_dim], scope='g_h4')

        return tf.nn.tanh(h4)
```



discriminator

```
def discriminator(self, image, reuse=False):
    with tf.variable_scope("discriminator", reuse=reuse) as scope:

        h0 = lrelu(conv2d(image, self.df_dim, scope='d_h0_conv'))
        h1 = lrelu(batch_normal(conv2d(h0, self.df_dim * 2, scope='d_h1_conv'), scope='d_bn1'))
        h2 = lrelu(batch_normal(conv2d(h1, self.df_dim * 4, scope='d_h2_conv'), scope='d_bn2'))
        h3 = lrelu(batch_normal(conv2d(h2, self.df_dim * 8, scope='d_h3_conv'), scope='d_bn3'))
        h4 = linear(tf.reshape(h3, [self.batch_size, -1]), 1, scope='d_h4_lin')

        return tf.nn.sigmoid(h4), h4
```

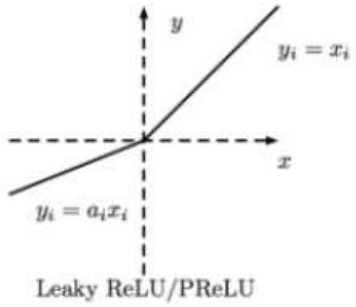


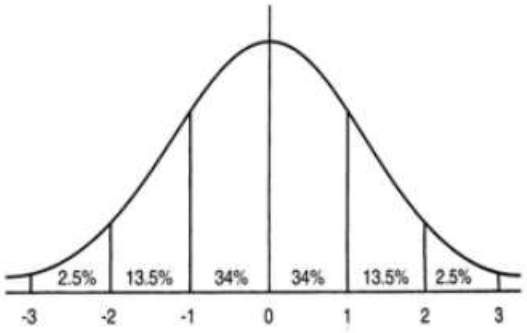
Features

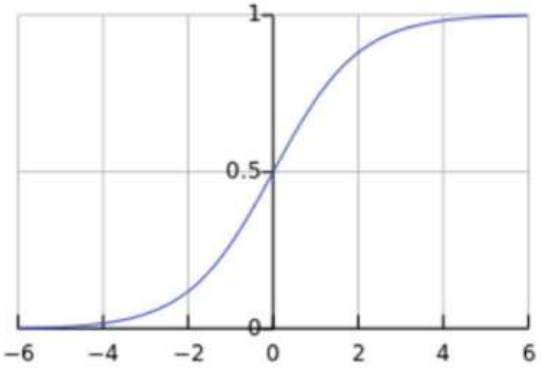
- Loss function

Batch Normalization

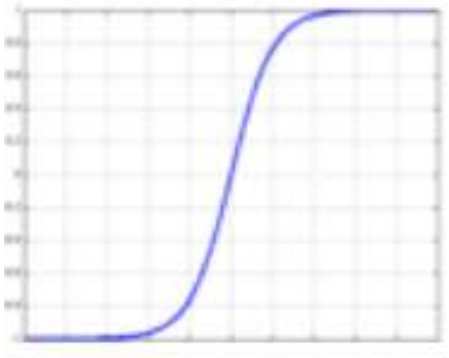
sigmoid







Tanh

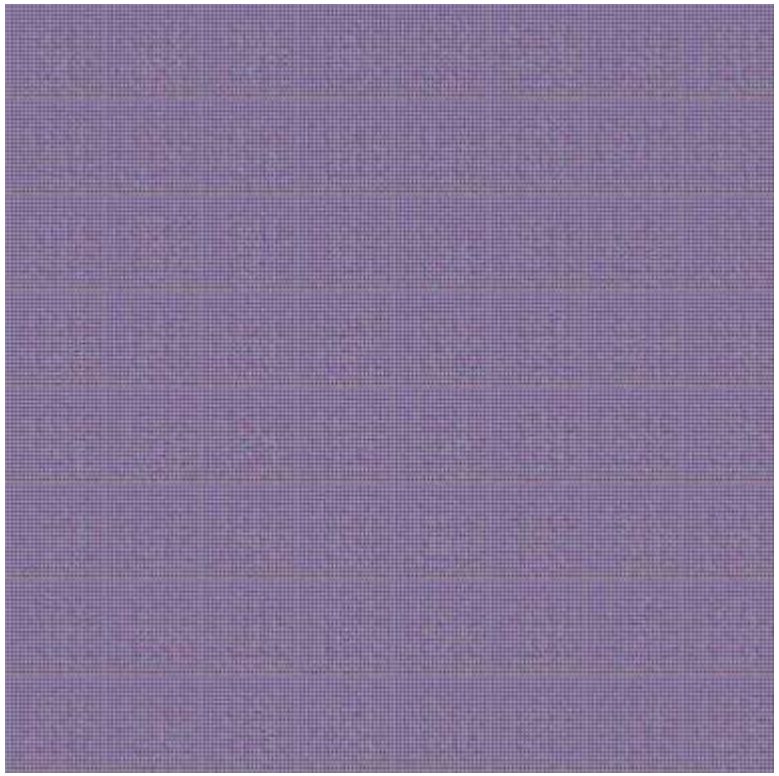


AdamOptimizer

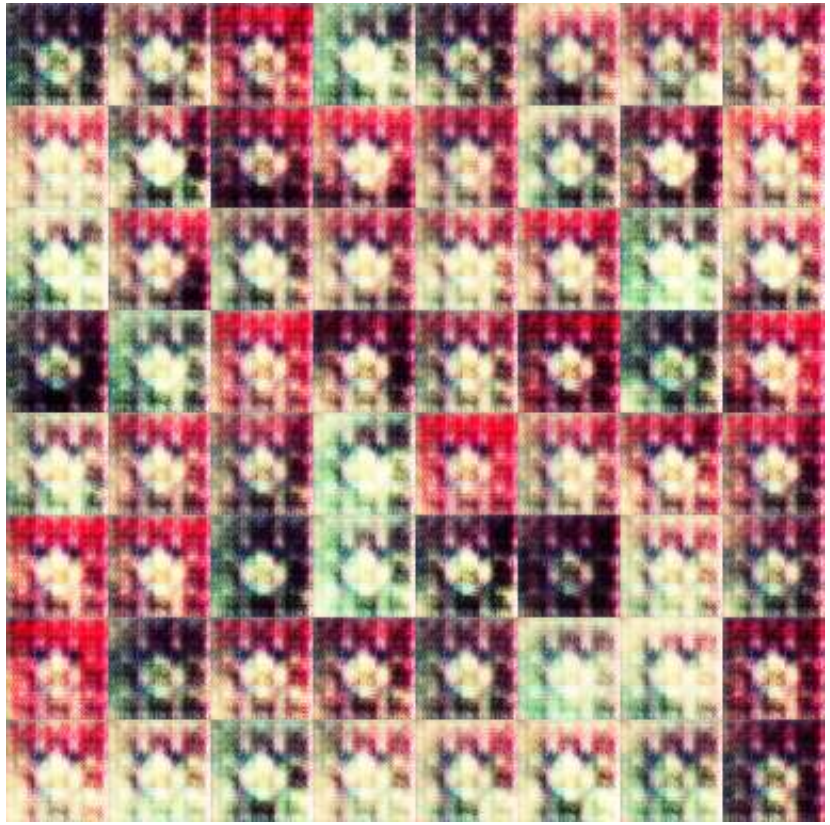
No pooling

Training d(k Times) g(one Time)

Result



0 epoch



1epoch

10 epoch

20 epoch

40 epoch

50 epoch

Conclusion

- 1.Beacuse of lacing datasets(only 1000 images),output is not clear.
- 2.Training G net is not easy. Overfitting bothers us.
- 3. Training costs a lot of computing resources and time.(we cost 10 hours)
- 4.generally speaking，Gan is one thing which need plenty of data and computing resources.
- In future，we think deeper network will be trained to capture semantic feature and generate high-quality images.