生成原始Deep Dream图像——单通道

```
# 定义卷积层、通道数，并取出对应的tensor

name = 'mixed4d_3x3_bottleneck_pre_relu'
# (?, ?, ?, 144)
#name = 'mixed4e_5x5_bottleneck_pre_relu'
#(?, ?, ?, 32)

channel = 139
# 'mixed4d_3x3_bottleneck_pre_relu'共144个通道
# 此处可选任意通道（0~143之间任意整数）进行最大化
layer_output = graph.get_tensor_by_name("import/%s:0" % name)

# 定义噪声图像
img_noise = np.random.uniform(size=(224, 224, 3)) + 100.0

# 调用render_naive函数渲染
render_naive(layer_output[:, :, :, channel], img_noise, iter_n=20)

# 保存并显示图片
im = PIL.Image.open('naive_deepdream.jpg')
im.show()
im.save('naive_single_chn.jpg')
```

生成原始Deep Dream图像——单通道

```
# 渲染函数
def render_naive(t_obj, img0, iter_n=20, step=1.0):
    # t_obj：是layer_output[:, :, :, channel]，即卷积层某个通道的值
    # img0：初始图像（噪声图像）
    # iter_n：迭代次数
    # step：用于控制每次迭代步长，可以看作学习率

    t_score = tf.reduce_mean(t_obj)
    # t_score是t_obj的平均值
    # 由于我们的目标是调整输入图像使卷积层激活值尽可能大
    # 即最大化t_score
    # 为达到此目标，可使用梯度下降
    # 计算t_score对t_input的梯度
    t_grad = tf.gradients(t_score, t_input)[0]

    img = img0.copy()#复制新图像可避免影响原图像的值
    for i in range(iter_n):
        # 在sess中计算梯度，以及当前的t_score
        g, score = sess.run([t_grad, t_score], {t_input: img})
        # 对img应用梯度
        # 首先对梯度进行归一化处理
        g /= g.std() + 1e-8
        # 将正规化处理后的梯度应用在图像上，step用于控制每次迭代步长，此处为1.0
        img += g * step
        #print('score(mean)=%f' % (score))
        print('iter:%d' %(i+1), 'score(mean)=%f' % score)
    # 保存图片
    savearray(img, 'naive_deepdream.jpg')
```

```
# 把一个numpy.ndarray保存成图像文件
def savearray(img_array, img_name):
    scipy.misc.toimage(img_array).save(img_name)
    print('img saved: %s' % img_name)
```

生成原始Deep Dream图像——单通道


```
iter:1 score(mean)=-19.989031
iter:2 score(mean)=-29.381023
iter:3 score(mean)=23.550827
iter:4 score(mean)=97.885620
iter:5 score(mean)=155.194595
iter:6 score(mean)=203.553497
iter:7 score(mean)=272.679962
iter:8 score(mean)=323.798920
iter:9 score(mean)=380.983093
iter:10 score(mean)=431.234680
iter:11 score(mean)=468.452820
iter:12 score(mean)=513.515686
iter:13 score(mean)=552.376221
iter:14 score(mean)=580.254028
iter:15 score(mean)=620.882202
iter:16 score(mean)=652.503235
iter:17 score(mean)=684.732483
iter:18 score(mean)=705.743652
iter:19 score(mean)=738.072571
iter:20 score(mean)=754.676331
```

name = 'mixed4d_3x3_bottleneck_pre_relu'

channel = 139

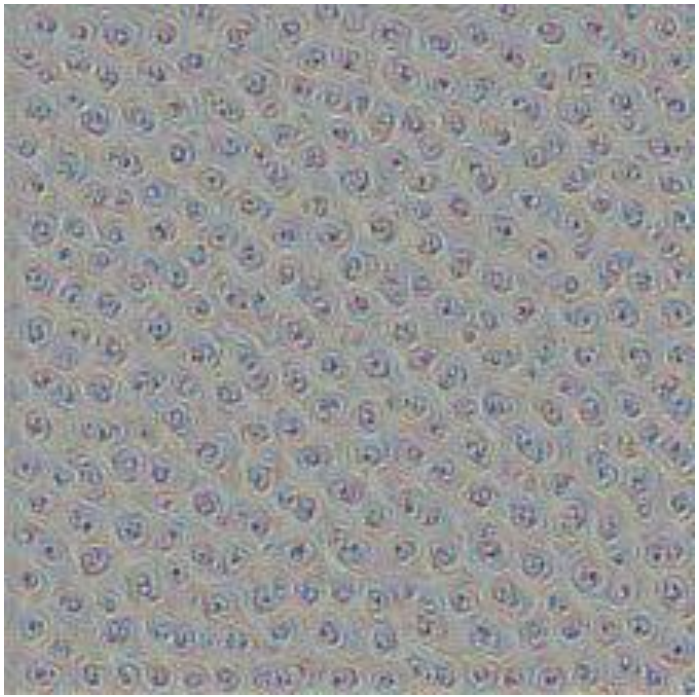
较低层单通道卷积特征生成DeepDream图像

```
# 定义卷积层、通道数，并取出对应的tensor
name3 = 'mixed3a_3x3_bottleneck_pre_relu'
layer_output = graph.get_tensor_by_name("import/%s:0" % name3)
print('shape of %s: %s' % (name3, str(graph.get_tensor_by_name('import/' + name3 + ':0').get_shape())))
```

```
# 定义噪声图像
img_noise = np.random.uniform(size=(224, 224, 3)) + 100.0

# 调用render_naive函数渲染
channel = 86 # (?, ?, ?, 96)
render_naive(layer_output[:, :, :, channel], img_noise, iter_n=20)

# 保存并显示图片
im = PIL.Image.open('naive_deepdream.jpg')
im.show()
im.save('shallow_single_chn.jpg')
```

较低层单通道卷积特征生成DeepDream图像

iter:1 score(mean)=5.855525
iter:2 score(mean)=47.340935
iter:3 score(mean)=132.637970
iter:4 score(mean)=202.955002
iter:5 score(mean)=246.161850
iter:6 score(mean)=277.414490
iter:7 score(mean)=298.887115
iter:8 score(mean)=315.585632
iter:9 score(mean)=328.799225
iter:10 score(mean)=338.754059
iter:11 score(mean)=347.754059
iter:12 score(mean)=354.990845
iter:13 score(mean)=361.499847
iter:14 score(mean)=366.884796
iter:15 score(mean)=371.788971
iter:16 score(mean)=376.163422
iter:17 score(mean)=380.038452
iter:18 score(mean)=383.628143
iter:19 score(mean)=386.749908
iter:20 score(mean)=389.645813



name3 = 'mixed3a_3x3_bottleneck_pre_relu'

channel = 86

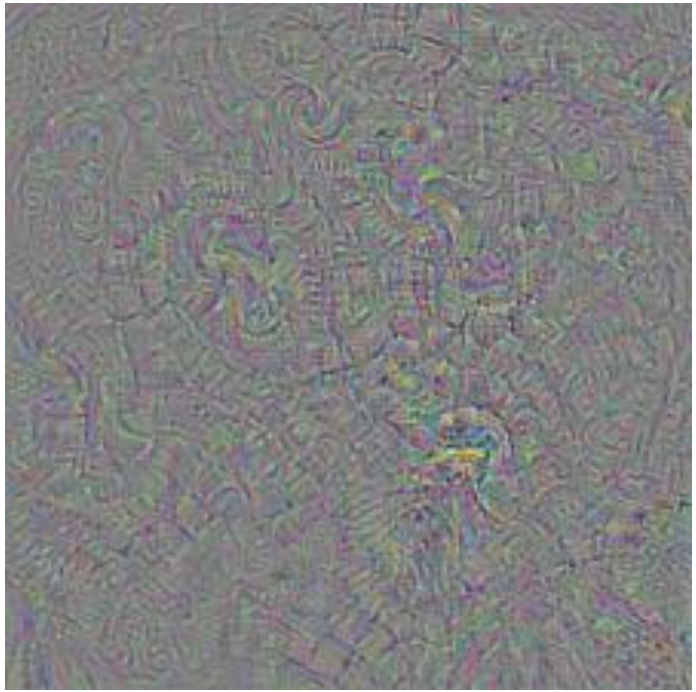
高层单通道卷积特征生成DeepDream图像

```
# 定义卷积层、通道数，并取出对应的tensor
name4 = 'mixed5b_5x5_pre_relu'
layer_output = graph.get_tensor_by_name("import/%s:0" % name4)
print('shape of %s: %s' % (name4, str(graph.get_tensor_by_name('import/' + name4 + ':0').get_shape())))
```

```
# 定义噪声图像
img_noise = np.random.uniform(size=(224, 224, 3)) + 100.0

# 调用render_naive函数渲染
channel =118 # (?, ?, ?, 128)
render_naive(layer_output[:, :, :, channel], img_noise, iter_n=20)

# 保存并显示图片
im = PIL.Image.open('naive_deepdream.jpg')
im.show()
im.save('deep_single_chn.jpg')
```

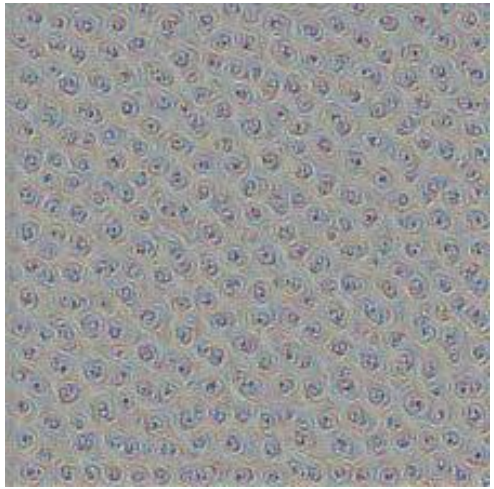
高层单通道卷积特征生成DeepDream图像

iter:1 score(mean)=-7.724879
iter:2 score(mean)=-7.390279
iter:3 score(mean)=-3.476621
iter:4 score(mean)=2.926053
iter:5 score(mean)=16.754850
iter:6 score(mean)=25.444139
iter:7 score(mean)=38.472725
iter:8 score(mean)=46.330528
iter:9 score(mean)=49.195690
iter:10 score(mean)=60.904064
iter:11 score(mean)=68.480019
iter:12 score(mean)=74.273743
iter:13 score(mean)=80.612503
iter:14 score(mean)=91.333069
iter:15 score(mean)=98.545242
iter:16 score(mean)=108.033852
iter:17 score(mean)=117.799294
iter:18 score(mean)=129.255615
iter:19 score(mean)=126.681732
iter:20 score(mean)=133.212936



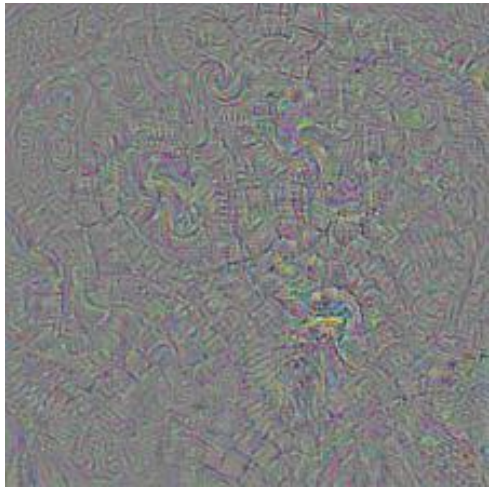
name4 = 'mixed5b_5x5_pre_relu'
channel = 118

生成原始Deep Dream图像



'mixed3a_3x3_bottleneck_pre_relu'

'mixed4d_3x3_bottleneck_pre_relu'



'mixed5b_5x5_pre_relu'

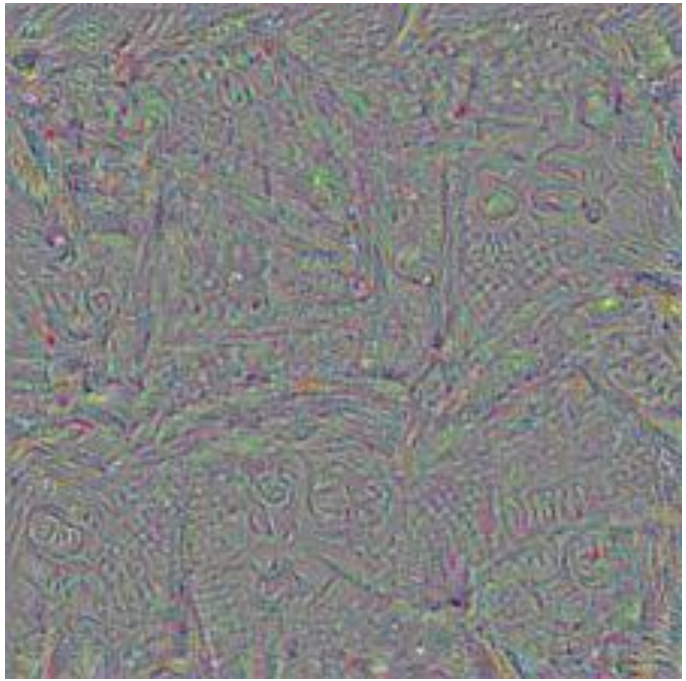
- 通过最大化某一通道的平均值能够得到有意义的图像
- 浅层 → 高层： 越来越抽象
- 单通道 → 多通道 → 所有通道

生成原始Deep Dream图像——所有通道

```
# 定义卷积层，并取出对应的tensor
name = 'mixed4d_3x3_bottleneck_pre_relu'
layer_output= graph.get_tensor_by_name("import/%s:0" % name)

# 定义噪声图像
img_noise = np.random.uniform(size=(224, 224, 3)) + 100.0

# 调用render_naive函数渲染
render_naive(layer_output, img_noise, iter_n=20) # 不指定特定通道，即表示利用所有通道特征
# 单通道时：layer_output[:, :, :, channel]

# 保存并显示图片
im = PIL.Image.open('naive_deepdream.jpg')
#im = PIL.Image.open('deepdream.jpg')
im.show()
im.save('all_chn.jpg')
```

生成原始Deep Dream图像——所有通道

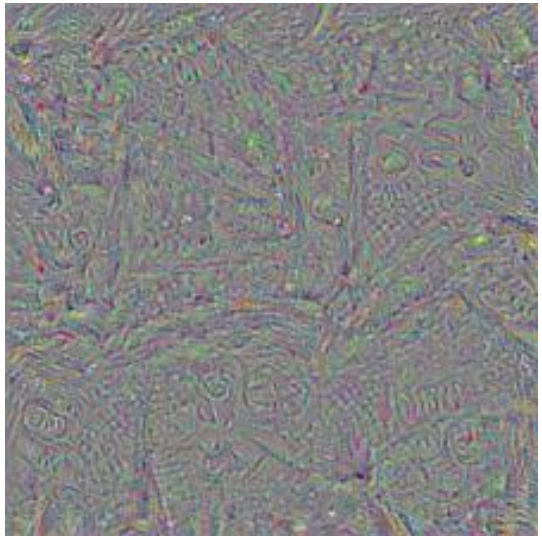
```
iter:1 score(mean)=-6.765324
iter:2 score(mean)=-10.231122
iter:3 score(mean)=-5.484193
iter:4 score(mean)=0.036023
iter:5 score(mean)=5.580604
iter:6 score(mean)=11.821063
iter:7 score(mean)=15.174915
iter:8 score(mean)=19.919533
iter:9 score(mean)=21.887245
iter:10 score(mean)=26.302950
iter:11 score(mean)=28.987417
iter:12 score(mean)=32.422749
iter:13 score(mean)=36.112301
iter:14 score(mean)=38.974667
iter:15 score(mean)=42.095287
iter:16 score(mean)=44.010132
iter:17 score(mean)=46.449688
iter:18 score(mean)=47.703903
iter:19 score(mean)=51.044811
iter:20 score(mean)=53.114281
```



name = 'mixed4e_5x5_bottleneck_pre_relu'
channel = all

生成原始Deep Dream图像——所有通道

Channel=139



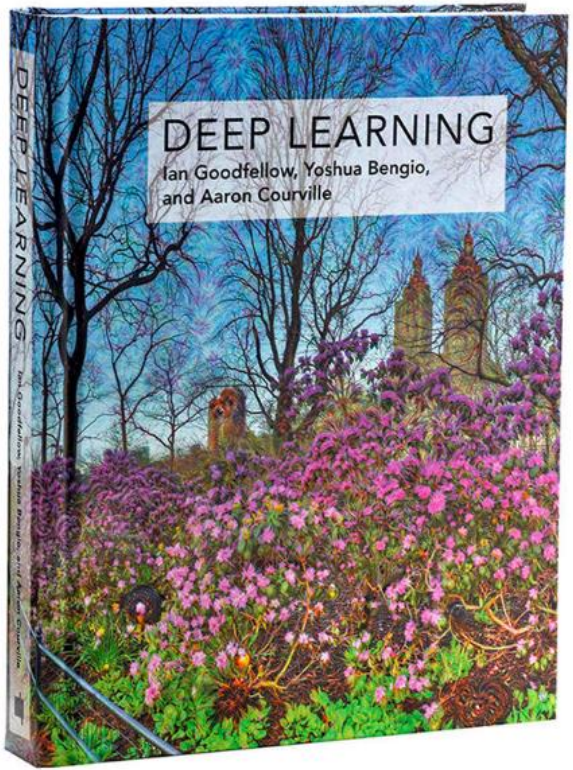
Channel: all

'mixed4d_3x3_bottleneck_pre_relu'

- 浅层 → 高层
- 单通道 → 所有通道

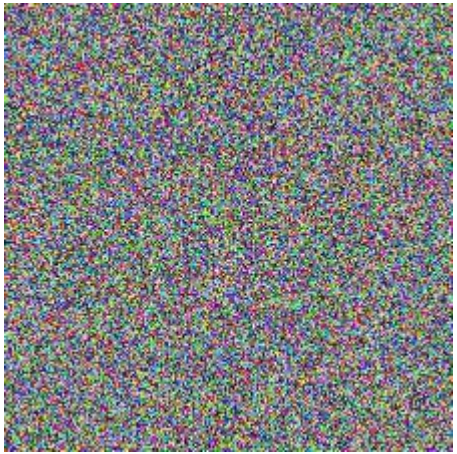
以背景图像为起点生成Deep Dream图像

以背景图像为起点生成Deep Dream图像



以背景图像为起点生成Deep Dream图像

- 以噪声图像为起点



- 以背景图像为起点

以背景图像为起点生成Deep Dream图像

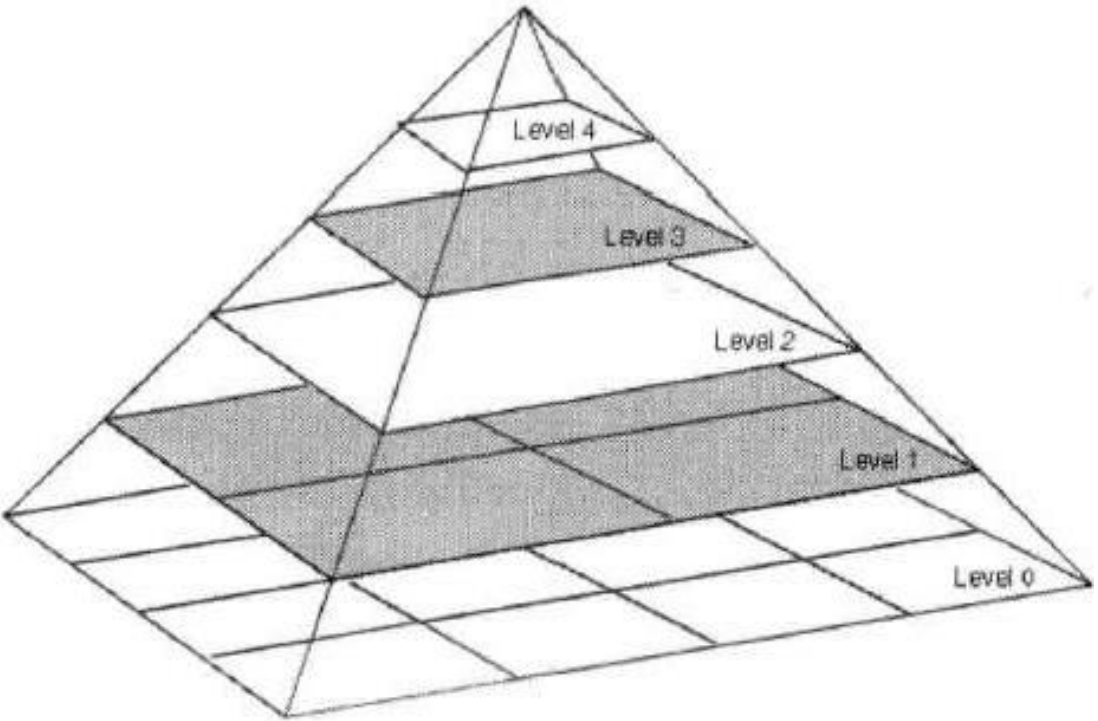

```
# 定义卷积层、并取出对应的tensor
name = 'mixed4c'
layer_output= graph.get_tensor_by_name("import/%s:0" % name)
print(layer_output)
```

```
shape of mixed4c: (?, ?, ?, 512)
Tensor("import/mixed4c:0", shape=(?, ?, ?, 512), dtype=float32, device=/device:CPU:0)
```

```
# 用一张背景图像（而不是随机噪音图像）作为起点对图像进行优化
img_test=PIL.Image.open('mountain.jpg') # img_noise = np.random.uniform(size=(224, 224, 3)) + 100.0

# 调用render_naive函数渲染
render_naive(layer_output, img_noise, iter_n=100) # 不指定特定通道，即表示利用所有通道特征

# 保存并显示图片
im = PIL.Image.open('deepdream.jpg')
im.show()
im.save('mountain_naive.jpg')
```

以背景图像为起点生成Deep Dream图像

提高生成图像的质量



图像的拉普拉斯金字塔分解

- **高频成分**：（level1, level2）图像中灰度、颜色、明度变化比较大的地方，如边缘、细节部分
- **低频成分**：（level3, level4）图像中变化不大的地方，如大块色块、整体风格

图像的拉普拉斯金字塔分解

```
def render_deepdream(t_obj, img0,
                     iter_n=10, step=1.5, octave_n=4, octave_scale=1.4):
    t_score = tf.reduce_mean(t_obj)
    t_grad = tf.gradients(t_score, t_input)[0]
    img = img0.copy()

    # 将图像进行金字塔分解
    # 从而分为高频、低频部分
    octaves = []
    for i in range(octave_n - 1):
        hw = img.shape[:2]
        lo = resize(img, np.int32(np.float32(hw) / octave_scale))
        hi = img - resize(lo, hw)
        img = lo
        octaves.append(hi)

    # 首先生成低频的图像，再依次放大并加上高频
    for octave in range(octave_n):
        if octave > 0:
            hi = octaves[-octave]
            img = resize(img, hi.shape[:2]) + hi
        for i in range(iter_n):
            g = calc_grad_tiled(img, t_grad)
            img += g * (step / (np.abs(g).mean() + 1e-7))

    img = img.clip(0, 255)
    savearray(img, 'mountain_deepdream.jpg')
    im = PIL.Image.open('mountain_deepdream.jpg').show()
```

生成更大尺寸的图像


```python
# 原始图像尺寸可能很大，从而导致内存耗尽问题
# 每次只对 tile_size * tile_size 大小的图像计算梯度，避免内存问题
def calc_grad_tiled(img, t_grad, tile_size=512):
    sz = tile_size
    h, w = img.shape[:2]
    sx, sy = np.random.randint(sz, size=2)
    img_shift = np.roll(np.roll(img, sx, 1), sy, 0)  # 先在行上做整体移动，再在列上做整体移动
    grad = np.zeros_like(img)
    for y in range(0, max(h - sz // 2, sz), sz):
        for x in range(0, max(w - sz // 2, sz), sz):
            sub = img_shift[y:y + sz, x:x + sz]
            g = sess.run(t_grad, {t_input: sub})
            grad[y:y + sz, x:x + sz] = g
    return np.roll(np.roll(grad, -sx, 1), -sy, 0)
```

以背景图像为起点生成Deep Dream图像

导入库与Inception模型

```
from __future__ import print_function
import os
from io import BytesIO
import numpy as np
from functools import partial
import PIL.Image
import scipy.misc
import tensorflow as tf


graph = tf.Graph()
model_fn = 'tensorflow_inception_graph.pb'
sess = tf.InteractiveSession(graph=graph)
with tf.gfile.FastGFile(model_fn, 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
t_input = tf.placeholder(np.float32, name='input')
imagenet_mean = 117.0
t_preprocessed = tf.expand_dims(t_input - imagenet_mean, 0)
tf.import_graph_def(graph_def, {'input': t_preprocessed})
```

以背景图像为起点生成Deep Dream图像

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

定义相关函数

```
# 保存图像
def savearray(img_array, img_name):
    scipy.misc.toimage(img_array).save(img_name)
    print('img saved: %s' % img_name)

# 将图像放大ratio倍
def resize_ratio(img, ratio):
    min = img.min()
    max = img.max()
    img = (img - min) / (max - min) * 255
    img = np.float32(scipy.misc.imresize(img, ratio))
    img = img / 255 * (max - min) + min
    return img

# 调整图像尺寸
def resize(img, hw):
    min = img.min()
    max = img.max()
    img = (img - min) / (max - min) * 255
    img = np.float32(scipy.misc.imresize(img, hw))
    img = img / 255 * (max - min) + min
    return img
```

```
# 原始图像尺寸可能很大，从而导致内存耗尽问题
# 每次只对 tile_size * tile_size 大小的图像计算梯度，避免内存问题
def calc_grad_tiled(img, t_grad, tile_size=512):
```

```
def render_deepdream(t_obj, img0,
                     iter_n=10, step=1.5, octave_n=4, octave_scale=1.4):
```

以背景图像为起点生成Deep Dream图像

主程序

```
name = 'mixed4c'
layer_output = graph.get_tensor_by_name("import/%s:0" % name)

img0 = PIL.Image.open('mountain.jpg')
img0 = np.float32(img0)
render_deepdream(tf.square(layer_output), img0)
```

提高生成图像的质量