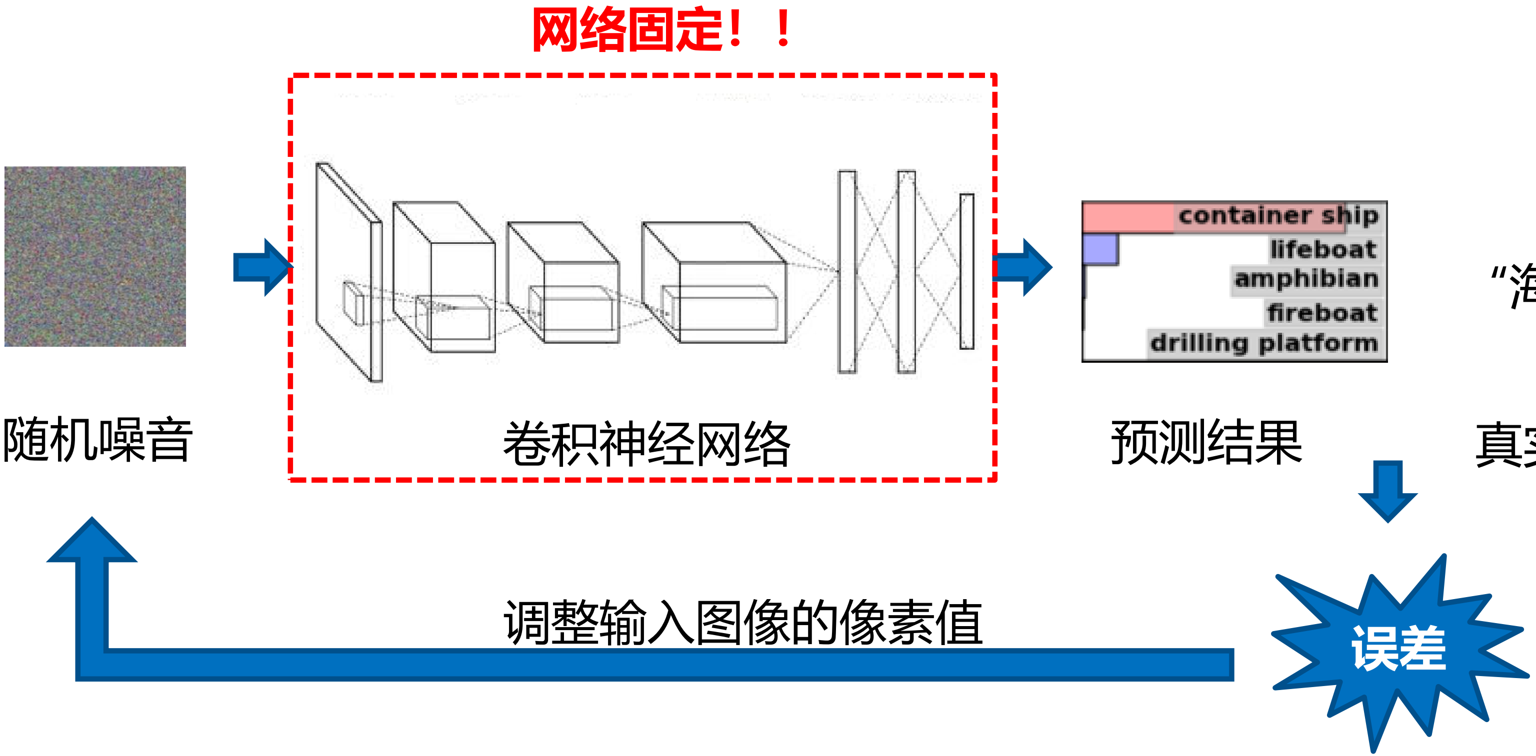
Deep Dream：
理解深度神经网络结构及应用

Deep Dream 技术原理



Deep Dream 初体验

噪音图像起点单层网络单通道

TensorFlow2实践

导入相关库

导入库函数

```
import tensorflow as tf
print("TF version:", tf.__version__)

#检测TensorFlow是否支持GPU
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
```

```
TF version: 2.0.0
GPU is NOT AVAILABLE
```

```
import numpy as np

import IPython.display as display
import PIL.Image

from tensorflow.keras.preprocessing import image
```

定义相关函数

定义图像标准化函数

```
# 图像标准化
def normalize_image(img):
    img = 255*(img + 1.0)/2.0
    return tf.cast(img, tf.uint8)
```

定义图像可视化函数

```
# 图像可视化
def show_image(img):
    display.display(PIL.Image.fromarray(np.array(img)))
```

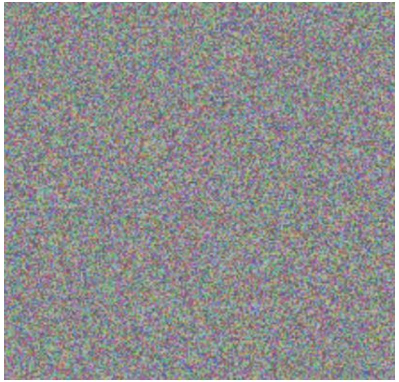
定义保存图像文件函数

```
# 保存图像文件
def save_image(img, file_name):
    PIL.Image.fromarray(np.array(img)).save(file_name)
```

产生噪音图像

产生噪音起点图像文件

```
# 定义图像噪声
img_noise = np.random.uniform(size=(300, 300, 3)) + 100.0
img_noise = img_noise.astype(np.float32) # dtype转换成float32
show_image(normalize_image(img_noise))
```



构建模型

预训练模型的加载

tf.keras.applications中包括了多种预训练的经典深度模型

```
base_model =
        tf.keras.applications.InceptionV3(include_top=False,
                                          weights='imagenet')
```

导入ImageNet数据集的图像识别预训练InceptionV3模型

去掉顶层，这样能接受新的训练数据shape

查看InceptionV3模型


```
base_model.summary()
```

```
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, None, None, 7 0           activation_87[0][0]
                                                                 activation_88[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, None, None, 7 0           activation_91[0][0]
                                                                 activation_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, None, None, 1 0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, None, None, 2 0           activation_85[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_93[0][0]
==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
```

Inception模型

- InceptionV3模型的架构相当大，有'mixed0'到'mixed10'这样11层。
- 使用不同的层会产生不同的图像，较深的层对较高级的特征(如眼睛和脸)有响应，而较早的层对较简单的特征(如边缘、形状和纹理)有响应。
- 可以自由地尝试选择的层，更深的层(具有更高索引的层)需要更长的时间来训练，因为梯度计算地更深入。

选择卷积层和通道

DeepDream的主要想法是选择一个卷积层的某个通道或者卷积层（也可以是多个网络层），改变图像像素（与训练分类器最大的区别），来最大化选中层或通道的激活值。

确定需要最大化激活的卷积层

```
# 最大限度地激活这些层的指定的层
layer_names='conv2d_85'

layers = base_model.get_layer(layer_names).output
```

```
layers
```

```
0]: <tf.Tensor 'conv2d_85/Identity:0' shape=(None, None, None, 320) dtype=float32>
```

创建特征提取模型

创建特征提取模型

```python
# 创建特征提取模型
dream_model = tf.keras.Model(inputs=base_model.input, outputs=layers)
```

tf.keras.Model(inputs=base_model.input, outputs=layers)

查看所创建的模型

查看模型摘要

```
dream_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None, None,  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, None, None, 3 864         input_1[0][0]
__________________________________________________________________________________________________

__________________________________________________________________________________________________
activation_84 (Activation)      (None, None, None, 1 0           batch_normalization_84[0][0]
__________________________________________________________________________________________________
mixed9 (Concatenate)            (None, None, None, 2 0           activation_76[0][0]
                                                                 mixed9_0[0][0]
                                                                 concatenate[0][0]
                                                                 activation_84[0][0]
__________________________________________________________________________________________________
conv2d_85 (Conv2D)              (None, None, None, 3 655360      mixed9[0][0]
==================================================================================================
Total params: 16,378,080
Trainable params: 16,350,176
Non-trainable params: 27,904
```

计算损失

损失是选中层的通道输出

```
def calc_loss(img, model):

    channel=13 # 选定第13通道

    # 对图像做变形，由（300, 300, 3）扩展为（1，300，300，3）
    img = tf.expand_dims(img, axis=0)

    # 图像通过模型前向传播得到计算结果
    layer_activations = model(img)

    # 取选中通道的值
    act = layer_activations[:,:,:,channel]

    # 选中通道的输出结果求均值
    loss = tf.math.reduce_mean(act)

    return  loss
```

图像预处理——增加维度

- 使用的图像数据格式通常是（height,width,channel），只能表示一张图像；
- 而Inception模型要求的输入格式却是（batch，height， width, channel），即同时将多张图像送入网络

tf.expand_dims(input, axis, name=None)

Returns:

A Tensor. Has the same type as input. Contains the same data as input, but its shape has an additional dimension of size 1 added.

向tensor中插入1个维度，插入位置就是参数代表的位置（维度从0开始）

定义图像优化过程

通过梯度上升进行图像调整，该图像会越来越多地“激活”模型中的指定层和通道的信息

```python
# 定义图像优化过程函数
def render_deepdream(model, img, steps=100, step_size=0.01, verbose=1):
    for n in tf.range(steps):
        with tf.GradientTape() as tape:
            # 对img进行梯度变换
            tape.watch(img)
            loss = calc_loss(img, model)

        #计算损失相对于输入图像像素的梯度
        gradients = tape.gradient(loss, img)

        # 归一化梯度值
        gradients /= tf.math.reduce_std(gradients) + 1e-8

        # 在梯度上升中，损失值越来越大，因此可以直接添加损失值到图像中，因为它们的shape相同
        img = img + gradients*step_size
        img = tf.clip_by_value(img, -1, 1)

        # 输出过程提示信息
        if (verbose ==1):
            if ((n+1)%10==0):
                print ("Step {}/{}, loss {}".format(n+1,steps, loss))

    return img
```

Deep Dream 应用实例

数据预处理

```
# 定义图像噪声
img_noise = np.random.uniform(size=(300, 300, 3)) + 100.0
img_noise = img_noise.astype(np.float32) # dtype转换成float32
show_image(normalize_image(img_noise))
```

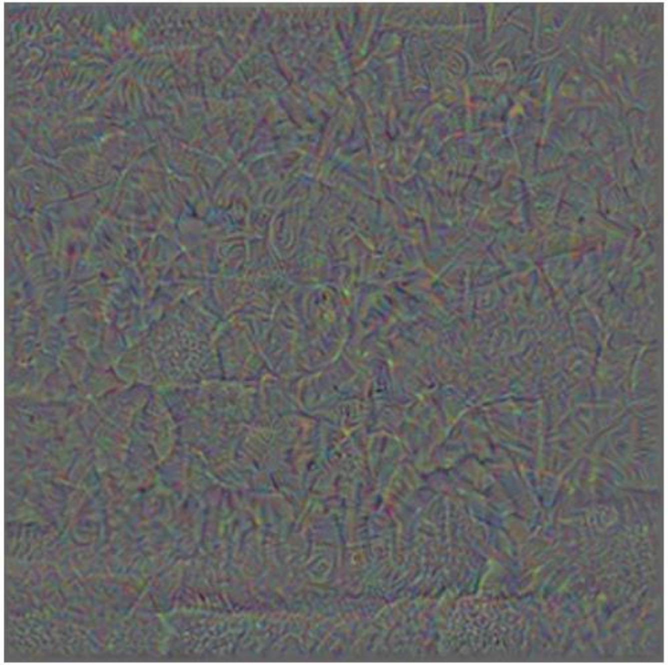
```
img = tf.keras.applications.inception_v3.preprocess_input(img_noise)
img = tf.convert_to_tensor(img)
```

开始做梦（应用Deep Dream）

```
import time

start = time.time()
print("开始做梦......")

# 调用优化过程
dream_img = render_deepdream(dream_model, img, steps=100, step_size=0.01)

end = time.time()
end-start
print("梦醒时分......")

# 标准化图像
dream_img = normalize_image(dream_img)

# 显示结果图像
show_image(dream_img)

# 保存结果图像
file_name = 'deepdream_{}.jpg'.format(layer_names)
save_image(dream_img,file_name)
print("梦境已保存为:{}! ".format(file_name))
```

Deep Dream结果

```
开始做梦......
Step 10/100, loss 0.7494049072265625
Step 20/100, loss 1.0840781927108765
Step 30/100, loss 1.4459736347198486
Step 40/100, loss 1.7884564399719238
Step 50/100, loss 2.075512647628784
Step 60/100, loss 2.4572386741638184
Step 70/100, loss 2.6662683486938477
Step 80/100, loss 2.9690115451812744
Step 90/100, loss 3.1516478061676025
Step 100/100, loss 3.3239903450012207
梦醒时分...... 耗时: 107.61676144599915
```



梦境已保存为:deepdream_conv2d_85.jpg!

Deep Dream 体验V2

噪音图像起点单层网络多通道

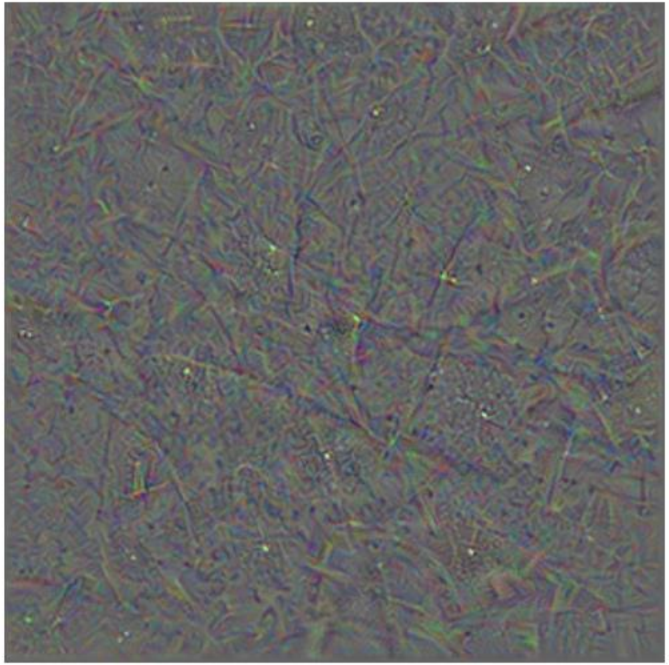
TensorFlow2实践

计算损失（多个通道总和）

```python
def calc_loss(img, model):

    channels=[13,139] # 选定通道列表

    # 对图像做变形，由（300, 300, 3）扩展为（1, 300, 300, 3）
    img = tf.expand_dims(img, axis=0)

    # 图像通过模型前向传播得到计算结果
    layer_activations = model(img)

    losses = [] # 计算选中每通道的计算结果均值
    for cn in channels:
        act = layer_activations[:,:,:,cn]
        loss = tf.math.reduce_mean(act)
        losses.append(loss)

    # 返回选中每通道的计算结果和
    return  tf.reduce_sum(losses)
```

Deep Dream结果

```
开始做梦......
Step 10/100, loss 1.237304925918579
Step 20/100, loss 1.9009684324264526
Step 30/100, loss 3.0373148918151855
Step 40/100, loss 3.6903765201568604
Step 50/100, loss 4.196420669555664
Step 60/100, loss 4.757476806640625
Step 70/100, loss 4.808797359466553
Step 80/100, loss 5.468280792236328
Step 90/100, loss 5.647038459777832
Step 100/100, loss 5.871610164642334
梦醒时分...... 耗时: 107.4575674533844
```



梦境已保存为:deepdream_conv2d_85.jpg!

Deep Dream 体验V3

噪音图像起点多层网络全通道

TensorFlow2实践

选择卷积层和通道

确定需要最大化激活的卷积层

```
#最大限度地激活这些层的指定的层
layer_names=['mixed3','mixed5']

layers = [base_model.get_layer(name).output for name in layer_names]
```

```
layers
```

10]:
```
[<tf.Tensor 'mixed3/Identity:0' shape=(None, None, None, 768) dtype=float32>,
 <tf.Tensor 'mixed5/Identity:0' shape=(None, None, None, 768) dtype=float32>]
```

计算损失（多层总和）

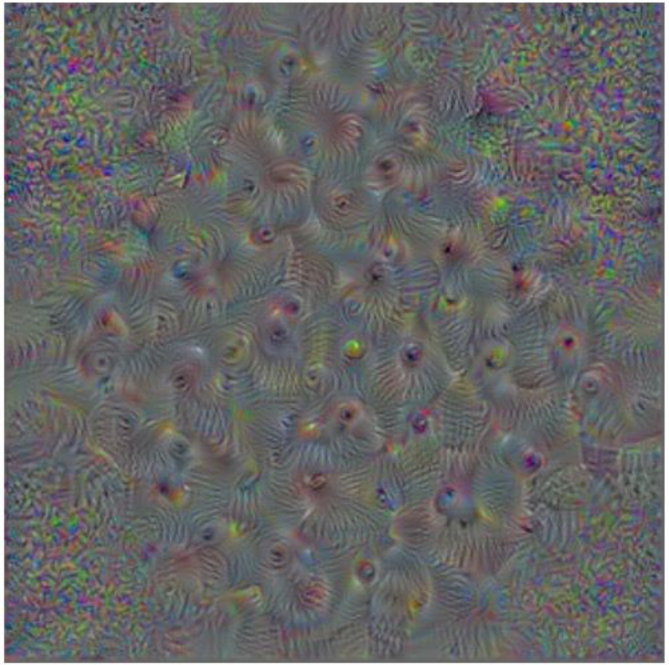
计算损失值

- 损失是选中层激活函数输出的总和，损失在每一层均归一化过，因此每一层对最终结果的影响差距不会很大。
- 通常，损失是希望通过梯度下降最小化的东西，但是在DeepDream中，需要将它最大化。

```
def calc_loss(img, model):

    # 对图像做变形，由（300, 300, 3）扩展为（1，300，300，3）
    img = tf.expand_dims(img, axis=0)

    # 图像通过模型前向传播得到计算结果
    layer_activations = model(img)

    losses = [] # 计算选中每层的计算结果均值
    for act in layer_activations:
        loss = tf.math.reduce_mean(act)
        losses.append(loss)

    # 返回选中每通道的计算结果和
    return  tf.reduce_sum(losses)
```

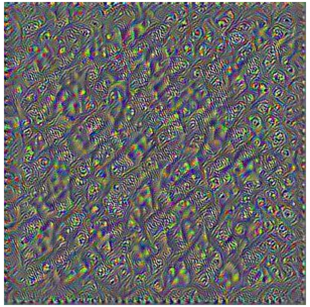
Deep Dream结果

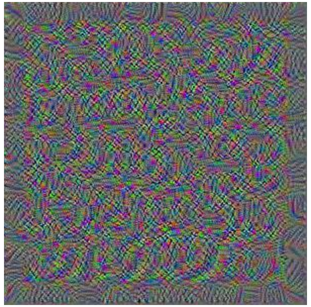

```
开始做梦......
Step 10/100, loss 1.0956816673278809
Step 20/100, loss 1.4184706211090088
Step 30/100, loss 1.6393194198608398
Step 40/100, loss 1.8129205703735352
Step 50/100, loss 1.9650282859802246
Step 60/100, loss 2.069246292114258
Step 70/100, loss 2.179379940032959
Step 80/100, loss 2.2747039794921875
Step 90/100, loss 2.35669207572937
Step 100/100, loss 2.4287261962890625
梦醒时分...... 耗时: 74.7091658115387
```

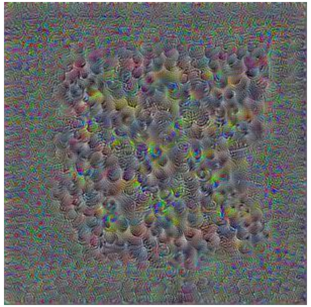


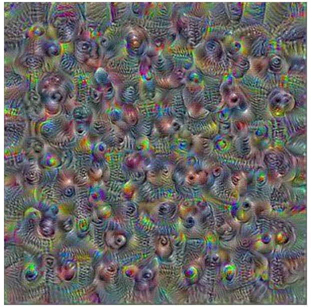
梦境已保存为:deepdream_['mixed3', 'mixed5'].jpg!

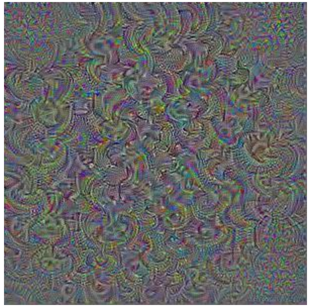
结果对比 – 噪音起点

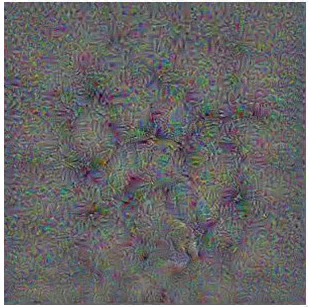

mixed0

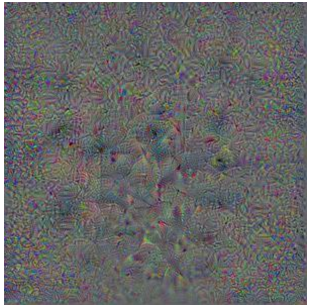

mixed1

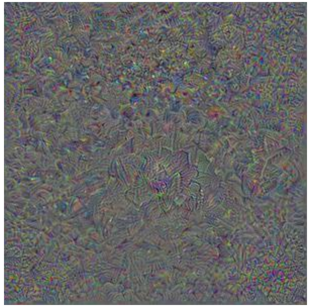

mixed2

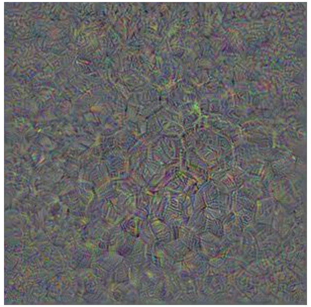

mixed3

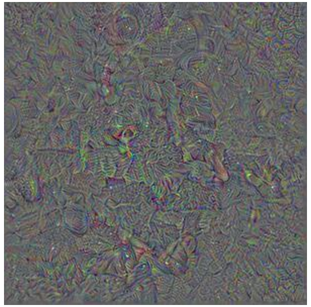

mixed4


mixed5

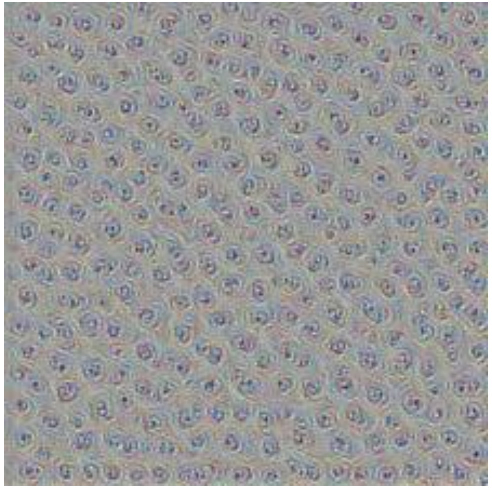

mixed6

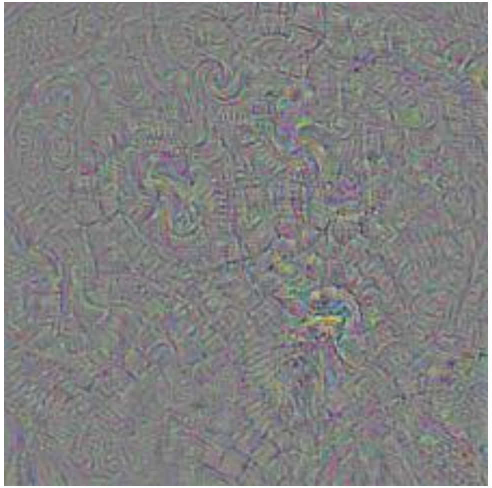

mixed7


mixed8


mixed9

深浅梦境对比





- 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

ZHEJIANG UNIVERSITY CITY COLLEGE

浙江大学城市学院

Deep Dream 体验V4

背景图像起点多层网络全通道

TensorFlow2实践

定义相关函数

定义读取图像文件函数，可以设置图像最大尺寸

```python
# 定义读取指定图像文件函数，可以设置图像最大尺寸
def read_image(file_name, max_dim=None):
    # image_path = "./ZUCC.jpg"
    img = PIL.Image.open(file_name)
    if max_dim:
        img.thumbnail((max_dim, max_dim))
    return np.array(img)
```

读取待处理图像文件

```
# 读取待处理图像
image_file = "ZUCC.jpg"

original_img = read_image(image_file, max_dim=500)

show_image(original_img)
```

选择卷积层和通道

确定需要最大化激活的卷积层

```
#最大限度地激活这些层的指定的层
layer_names=['mixed3','mixed5']

layers = [base_model.get_layer(name).output for name in layer_names]
```

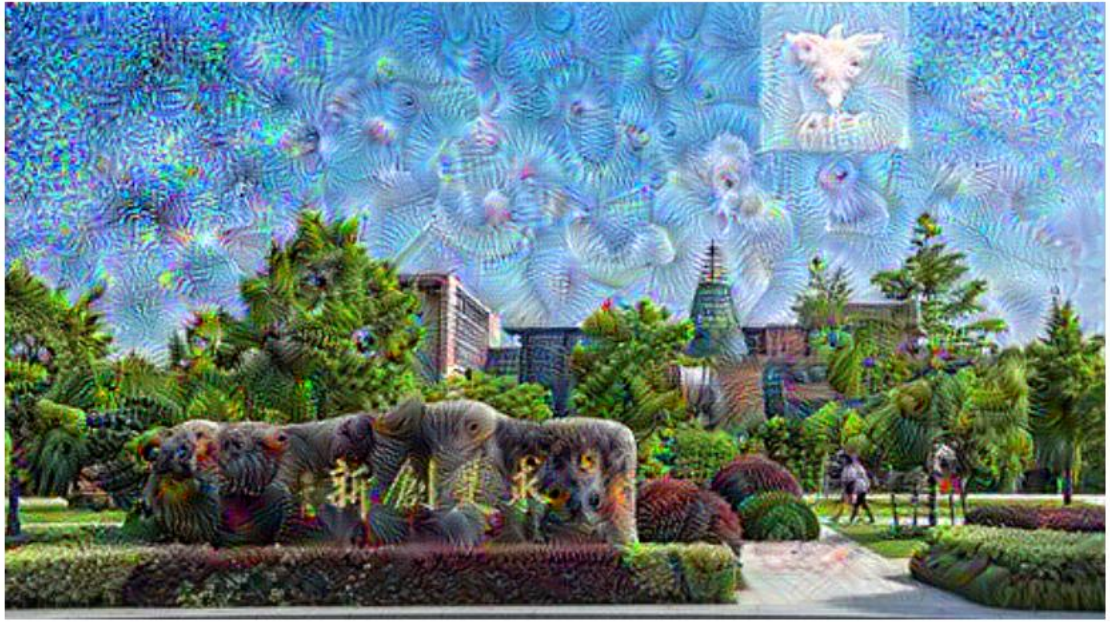
```
layers
```

10]:
```
[<tf.Tensor 'mixed3/Identity:0' shape=(None, None, None, 768) dtype=float32>,
 <tf.Tensor 'mixed5/Identity:0' shape=(None, None, None, 768) dtype=float32>]
```

Deep Dream 结果

浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

```
Step 190/200, loss 2.6629419326782227
Step 200/200, loss 2.6972901821136475
梦醒时分...... 耗时： 229.09420895576477
```



```
梦境已保存为:dream_['mixed3', 'mixed5'].jpg!
```

优化1

存在问题

上面生成的图像有以下几个问题：

1. 输出有噪声
2. 图像分辨率低
3. 输出的特征模式都一样

方案：

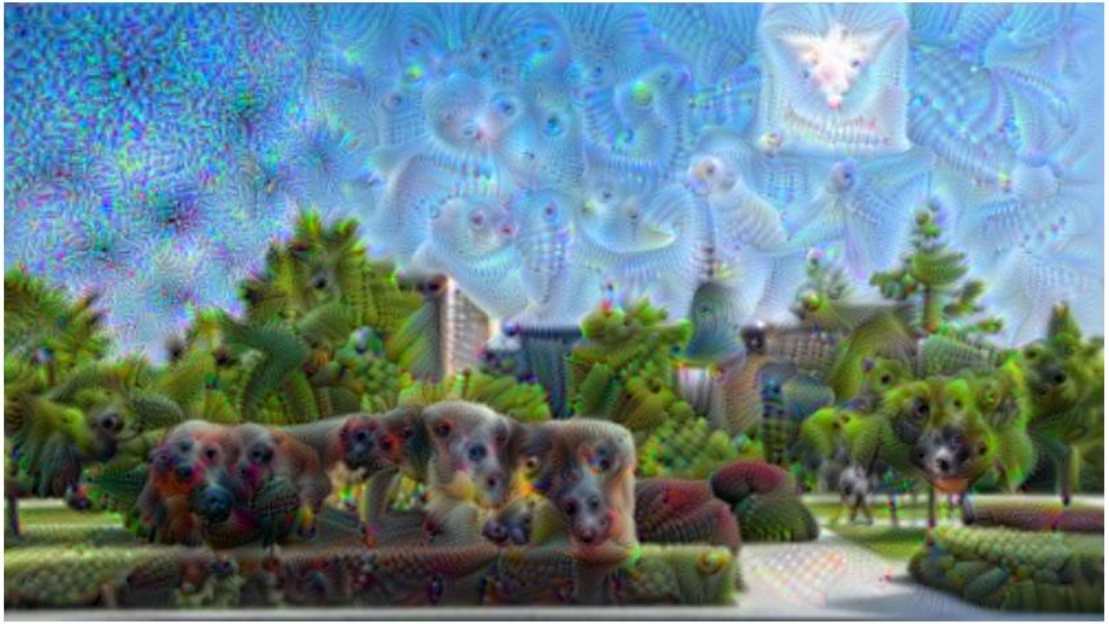
可以在不同比例的图上使用梯度上升来解决这些问题，并在小比例上图生成的结果合并到到更大比例的图上。

不同比例迭代进行


```
import time
start = time.time()

OCTAVE_SCALE = 1.30

img = tf.keras.applications.inception_v3.preprocess_input(original_img)
img = tf.convert_to_tensor(img)

initial_shape = tf.shape(img)[:-1]

# 从小到大比例进行多次优化过程
for octave in range(-2, 3):
    new_size = tf.cast(tf.convert_to_tensor(initial_shape), tf.float32)*(OCTAVE_SCALE**octave)
    img = tf.image.resize(img, tf.cast(new_size, tf.int32))

    img = render_deepdream(dream_model, img, steps=30, step_size=0.01)

# 把图调整回原始大小
img = tf.image.resize(img, initial_shape)

# 标准化图像
img = normalize_image(img)

show_image(img)

end = time.time()
print('耗时：',end-start)
```

Deep Dream 结果

Step 20/30, loss 1.6665358543395996
Step 30/30, loss 1.840975046157837



耗时： 228.23297953605652
图片已保存为:dream_octave_['mixed3', 'mixed5'].jpg!

优化2

存在问题

如果单幅图像尺寸过大，执行梯度计算所需的时间和内存也会随之增加，有的机器可能无法支持

方案：

　　可以将图像拆分为多个小图块计算梯度，最后将其拼合起来，得到最终图像。

定义图像切分移动函数

定义图像移动函数

```
def random_roll(img, maxroll=512):
  # 随机移动图像
    shift = tf.random.uniform(shape=[2], minval=-maxroll, maxval=maxroll, dtype=tf.int32)
    print(shift)
    shift_down, shift_right = shift[0],shift[1]
    print(shift_down, shift_right)
    img_rolled = tf.roll(tf.roll(img, shift_right, axis=1), shift_down, axis=0)
    return shift_down, shift_right, img_rolled
```

图像切分移动案例

```
shift_down, shift_right, img_rolled = random_roll(np.array(original_img), 512)
print(shift_down, shift_right)
show_image(img_rolled)
```

```
tf.Tensor([-396  456], shape=(2,), dtype=int32)
tf.Tensor(-396, shape=(), dtype=int32) tf.Tensor(456, shape=(), dtype=int32)
tf.Tensor(-396, shape=(), dtype=int32) tf.Tensor(456, shape=(), dtype=int32)
```

定义分块计算的梯度函数

```
# 求梯度
def get_tiled_gradients(model, img, tile_size=150):
    shift_down, shift_right, img_rolled = random_roll(img, tile_size)

    # 初始化梯度为0
    gradients = tf.zeros_like(img_rolled)

    # 产生分块坐标列表
    xs = tf.range(0, img_rolled.shape[0], tile_size)
    ys = tf.range(0, img_rolled.shape[1], tile_size)

    for x in xs:
        for y in ys:
            #计算该图块的梯度
            with tf.GradientTape() as tape:
                tape.watch(img_rolled)

                #从图像中提取该图块，最后一块大小会按实际提取
                img_tile = img_rolled[x:x+tile_size, y:y+tile_size]

                loss = calc_loss(img_tile, model)

                #更新图像的梯度
                gradients = gradients + tape.gradient(loss, img_rolled)

    # 将图块放回原来的位置
    gradients = tf.roll(tf.roll(gradients, -shift_right, axis=1), -shift_down, axis=0)

    # 归一化梯度
    gradients /= tf.math.reduce_std(gradients) + 1e-8

    return gradients
```

定义优化后的Deep Dream函数

定义优化后的Deep Dream函数

```
def render_deepdream_with_octaves(model, img, steps_per_octave=100, step_size=0.01,
                                  octaves=range(-2,3), octave_scale=1.3):

    initial_shape = img.shape[:-1]

    for octave in octaves:
        new_size = tf.cast(tf.convert_to_tensor(initial_shape), tf.float32)*(octave_scale**octave)
        img = tf.image.resize(img, tf.cast(new_size, tf.int32))

        for step in range(steps_per_octave):
            gradients = get_tiled_gradients(model, img)
            img = img + gradients*step_size
            img = tf.clip_by_value(img, -1, 1)

            if ((step+1)%10==0):
                print ("Octave {},Step {}".format(octave, step+1))

    img = tf.image.resize(img, initial_shape)
    result = normalize_image(img)
    return result
```

应用Deep Dream

应用Deep Dream

```
import time
start = time.time()

print("开始做梦......")

img = tf.keras.applications.inception_v3.preprocess_input(original_img)
img = tf.convert_to_tensor(img)

img = render_deepdream_with_octaves(dream_model, img, steps_per_octave=50, step_size=0.01,
                                    octaves=range(-2,3), octave_scale=1.3)

show_image(img)

end = time.time()
end-start

print("梦醒时分......")

file_name = 'dream_tile_octave_{}.jpg'.format(layer_names)
save_image(img,file_name)
print("图片已保存为:{}!".format(file_name))
```

结果

```
Octave 2,Step 30
Octave 2,Step 40
Octave 2,Step 50
```

```
梦醒时分......
图片已保存为:dream_tile_octave_['mixed3', 'mixed5'].jpg!
```

结果对比 – 背景图起点

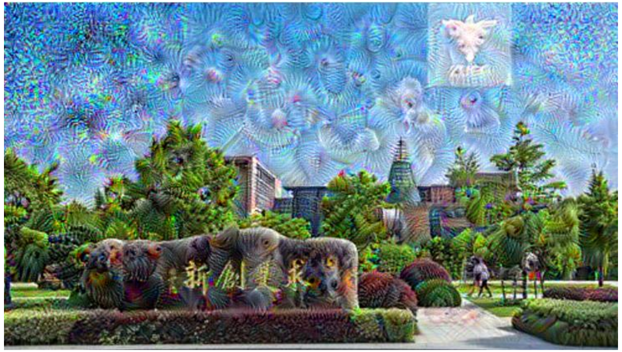
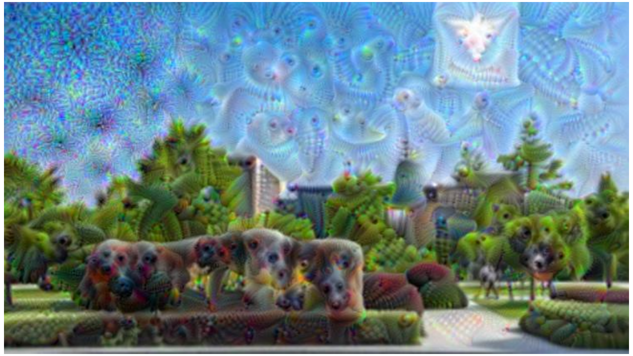
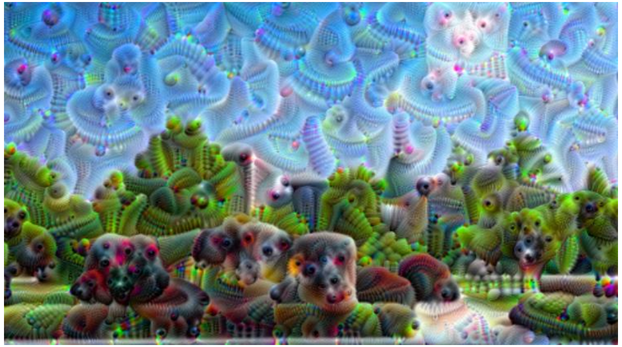


图0 | 图1

图2 | 图3





经典Deep Dream图像