**实战二：手把手教你图像风格迁移**

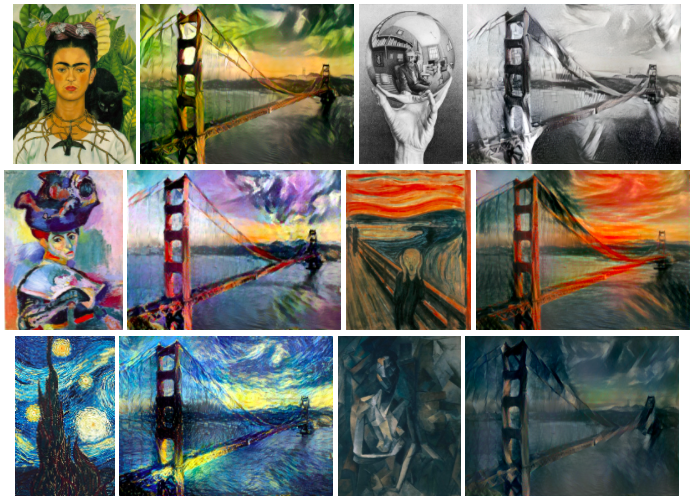
**一、简介**

图像风格迁移是指，将一幅内容图的内容，和一幅或多幅风格图融合在一起，从而生成一些有意思的图片。

有兴趣的可以看一下外文文献 [Leon A. Gatys' paper, A Neural Algorithm of Artistic Style](https://arxiv.org/abs/1508.06576" \t "https://colab.research.google.com/drive/_blank)

<https://arxiv.org/abs/1508.06576>（ps访问需外网，可参考后面的操作）

例子：



**二、方法一TensorFlow实现图像风格迁移**

**1. 准备**

安装包：

pip install numpy scipy tensorflow keras

图片准备：

准备好内容图片以及风格图片



**2. 原理**

为了将风格图的风格和内容图的内容进行融合，所生成的图片，在内容应当尽可能接近内容图，在风格中应当尽可能接近风格图。

因此需要定义内容损失函数和风格损失函数，经过加权后作为总的损失函数。

实现步骤如下：

a.随机产生一张图片

b.在每轮迭代中，根据总的损失函数，调整图片的像素值

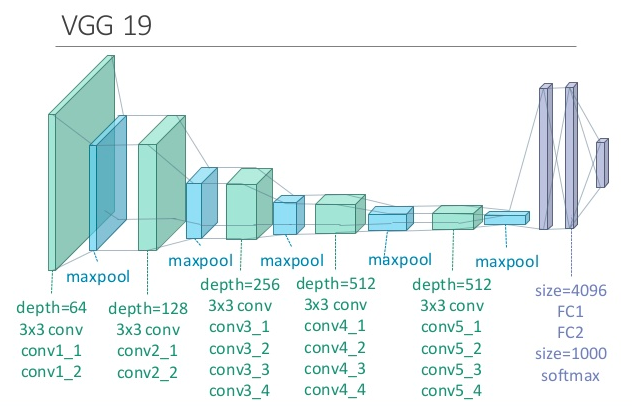
c.经过多轮迭代，得到优化后的图片

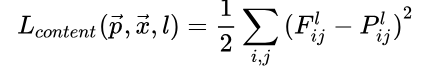
**3.内容损失函数**

两张图片在内容中相似，不能仅仅靠简单的纯像素比较

CNN具有抽象和理解图像的能力，因此可以考虑将各个卷积层的输出作为图像的内容。

以VGG19为例，其中包括了多个卷积层、池化层，以及最后的全连接层。



这里使用conv4\_2的输出作为图像的内容表示，定义内容损失函数如下

**4.风格损失函数**

风格是一个很难说清楚的概念，可能是笔触、纹理、结构、布局、用色等等。

这里使用卷积层各个特征图之间的互相关作为图像的风格，以conv1\_1为例

共包含64个特征图即feature map，或者说图像的深度、通道的个数

每个特征图都是对上一层输出的一种理解，可以类比成64个人对同一幅画的不同理解

这些人可能分别偏好印象派、现代主义、超现实主义、表现主义等不同风格

当图像是某一种风格时，可能这一部分人很欣赏，但那一部分人不喜欢

当图像是另一种风格时，可能这一部分人不喜欢，但那一部分人很欣赏

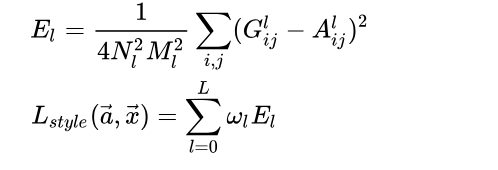
64个人之间理解的差异，可以用特征图的互相关表示，这里使用Gram矩阵计算互相关

不同的风格会导致差异化的互相关结果

Gram矩阵的计算如下，如果有64个特征图，那么Gram矩阵的大小便是64 × 64，第i行第j列的值表示第i个特征图和第j个特征图之间的互相关，用内积计算

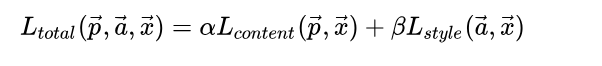


风格损失函数定义如下，对多个卷积层的风格表示差异进行加权



**5.损失函数**

总的损失函数即内容损失函数和风格损失函数的加权，不同的权重会导致不同的迁移效果。



**6.TensorFlow代码实现**

# -\*- coding: utf-8 -\*-

# 加载库

import tensorflow as tf

import numpy as np

import scipy.io

import scipy.misc

import os

import time

def the\_current\_time():

print(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(int(time.time()))))

# 定义一些变量

CONTENT\_IMG = 'content.jpg'

STYLE\_IMG = 'style5.jpg'

OUTPUT\_DIR = 'neural\_style\_transfer\_tensorflow/'

if not os.path.exists(OUTPUT\_DIR):

os.mkdir(OUTPUT\_DIR)

IMAGE\_W = 800

IMAGE\_H = 600

COLOR\_C = 3

NOISE\_RATIO = 0.7

BETA = 5

ALPHA = 100

VGG\_MODEL = 'imagenet-vgg-verydeep-19.mat'

MEAN\_VALUES = np.array([123.68, 116.779, 103.939]).reshape((1, 1, 1, 3))

# 加载VGG19模型

def load\_vgg\_model(path):

'''

Details of the VGG19 model:

- 0 is conv1\_1 (3, 3, 3, 64)

- 1 is relu

- 2 is conv1\_2 (3, 3, 64, 64)

- 3 is relu

- 4 is maxpool

- 5 is conv2\_1 (3, 3, 64, 128)

- 6 is relu

- 7 is conv2\_2 (3, 3, 128, 128)

- 8 is relu

- 9 is maxpool

- 10 is conv3\_1 (3, 3, 128, 256)

- 11 is relu

- 12 is conv3\_2 (3, 3, 256, 256)

- 13 is relu

- 14 is conv3\_3 (3, 3, 256, 256)

- 15 is relu

- 16 is conv3\_4 (3, 3, 256, 256)

- 17 is relu

- 18 is maxpool

- 19 is conv4\_1 (3, 3, 256, 512)

- 20 is relu

- 21 is conv4\_2 (3, 3, 512, 512)

- 22 is relu

- 23 is conv4\_3 (3, 3, 512, 512)

- 24 is relu

- 25 is conv4\_4 (3, 3, 512, 512)

- 26 is relu

- 27 is maxpool

- 28 is conv5\_1 (3, 3, 512, 512)

- 29 is relu

- 30 is conv5\_2 (3, 3, 512, 512)

- 31 is relu

- 32 is conv5\_3 (3, 3, 512, 512)

- 33 is relu

- 34 is conv5\_4 (3, 3, 512, 512)

- 35 is relu

- 36 is maxpool

- 37 is fullyconnected (7, 7, 512, 4096)

- 38 is relu

- 39 is fullyconnected (1, 1, 4096, 4096)

- 40 is relu

- 41 is fullyconnected (1, 1, 4096, 1000)

- 42 is softmax

'''

vgg = scipy.io.loadmat(path)

vgg\_layers = vgg['layers']

def \_weights(layer, expected\_layer\_name):

W = vgg\_layers[0][layer][0][0][2][0][0]

b = vgg\_layers[0][layer][0][0][2][0][1]

layer\_name = vgg\_layers[0][layer][0][0][0][0]

assert layer\_name == expected\_layer\_name

return W, b

def \_conv2d\_relu(prev\_layer, layer, layer\_name):

W, b = \_weights(layer, layer\_name)

W = tf.constant(W)

b = tf.constant(np.reshape(b, (b.size)))

return tf.nn.relu(tf.nn.conv2d(prev\_layer, filter=W, strides=[1, 1, 1, 1], padding='SAME') + b)

def \_avgpool(prev\_layer):

return tf.nn.avg\_pool(prev\_layer, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

graph = {}

graph['input'] = tf.Variable(np.zeros((1, IMAGE\_H, IMAGE\_W, COLOR\_C)), dtype='float32')

graph['conv1\_1'] = \_conv2d\_relu(graph['input'], 0, 'conv1\_1')

graph['conv1\_2'] = \_conv2d\_relu(graph['conv1\_1'], 2, 'conv1\_2')

graph['avgpool1'] = \_avgpool(graph['conv1\_2'])

graph['conv2\_1'] = \_conv2d\_relu(graph['avgpool1'], 5, 'conv2\_1')

graph['conv2\_2'] = \_conv2d\_relu(graph['conv2\_1'], 7, 'conv2\_2')

graph['avgpool2'] = \_avgpool(graph['conv2\_2'])

graph['conv3\_1'] = \_conv2d\_relu(graph['avgpool2'], 10, 'conv3\_1')

graph['conv3\_2'] = \_conv2d\_relu(graph['conv3\_1'], 12, 'conv3\_2')

graph['conv3\_3'] = \_conv2d\_relu(graph['conv3\_2'], 14, 'conv3\_3')

graph['conv3\_4'] = \_conv2d\_relu(graph['conv3\_3'], 16, 'conv3\_4')

graph['avgpool3'] = \_avgpool(graph['conv3\_4'])

graph['conv4\_1'] = \_conv2d\_relu(graph['avgpool3'], 19, 'conv4\_1')

graph['conv4\_2'] = \_conv2d\_relu(graph['conv4\_1'], 21, 'conv4\_2')

graph['conv4\_3'] = \_conv2d\_relu(graph['conv4\_2'], 23, 'conv4\_3')

graph['conv4\_4'] = \_conv2d\_relu(graph['conv4\_3'], 25, 'conv4\_4')

graph['avgpool4'] = \_avgpool(graph['conv4\_4'])

graph['conv5\_1'] = \_conv2d\_relu(graph['avgpool4'], 28, 'conv5\_1')

graph['conv5\_2'] = \_conv2d\_relu(graph['conv5\_1'], 30, 'conv5\_2')

graph['conv5\_3'] = \_conv2d\_relu(graph['conv5\_2'], 32, 'conv5\_3')

graph['conv5\_4'] = \_conv2d\_relu(graph['conv5\_3'], 34, 'conv5\_4')

graph['avgpool5'] = \_avgpool(graph['conv5\_4'])

return graph

# 定义内容损失函数

def content\_loss\_func(sess, model):

def \_content\_loss(p, x):

N = p.shape[3]

M = p.shape[1] \* p.shape[2]

return (1 / (4 \* N \* M)) \* tf.reduce\_sum(tf.pow(x - p, 2))

return \_content\_loss(sess.run(model['conv4\_2']), model['conv4\_2'])

# 定义风格损失函数

STYLE\_LAYERS = [('conv1\_1', 0.5), ('conv2\_1', 1.0), ('conv3\_1', 1.5), ('conv4\_1', 3.0), ('conv5\_1', 4.0)]

def style\_loss\_func(sess, model):

def \_gram\_matrix(F, N, M):

Ft = tf.reshape(F, (M, N))

return tf.matmul(tf.transpose(Ft), Ft)

def \_style\_loss(a, x):

N = a.shape[3]

M = a.shape[1] \* a.shape[2]

A = \_gram\_matrix(a, N, M)

G = \_gram\_matrix(x, N, M)

return (1 / (4 \* N \*\* 2 \* M \*\* 2)) \* tf.reduce\_sum(tf.pow(G - A, 2))

return sum([\_style\_loss(sess.run(model[layer\_name]), model[layer\_name]) \* w for layer\_name, w in STYLE\_LAYERS])

# 随机产生一张初始照片

def generate\_noise\_image(content\_image, noise\_ratio=NOISE\_RATIO):

noise\_image = np.random.uniform(-20, 20, (1, IMAGE\_H, IMAGE\_W, COLOR\_C)).astype('float32')

input\_image = noise\_image \* noise\_ratio + content\_image \* (1 - noise\_ratio)

return input\_image

# 加载图片

def load\_image(path):

image = scipy.misc.imread(path)

image = scipy.misc.imresize(image, (IMAGE\_H, IMAGE\_W))

image = np.reshape(image, ((1, ) + image.shape))

image = image - MEAN\_VALUES

return image

# 保存图片

def save\_image(path, image):

image = image + MEAN\_VALUES

image = image[0]

image = np.clip(image, 0, 255).astype('uint8')

scipy.misc.imsave(path, image)

# 调用以上函数并训练模型

the\_current\_time()

with tf.Session() as sess:

content\_image = load\_image(CONTENT\_IMG)

style\_image = load\_image(STYLE\_IMG)

model = load\_vgg\_model(VGG\_MODEL)

input\_image = generate\_noise\_image(content\_image)

sess.run(tf.global\_variables\_initializer())

sess.run(model['input'].assign(content\_image))

content\_loss = content\_loss\_func(sess, model)

sess.run(model['input'].assign(style\_image))

style\_loss = style\_loss\_func(sess, model)

total\_loss = BETA \* content\_loss + ALPHA \* style\_loss

optimizer = tf.train.AdamOptimizer(2.0)

train = optimizer.minimize(total\_loss)

sess.run(tf.global\_variables\_initializer())

sess.run(model['input'].assign(input\_image))

ITERATIONS = 2000

for i in range(ITERATIONS):

sess.run(train)

if i % 100 == 0:

output\_image = sess.run(model['input'])

the\_current\_time()

print('Iteration %d' % i)

print('Cost: ', sess.run(total\_loss))

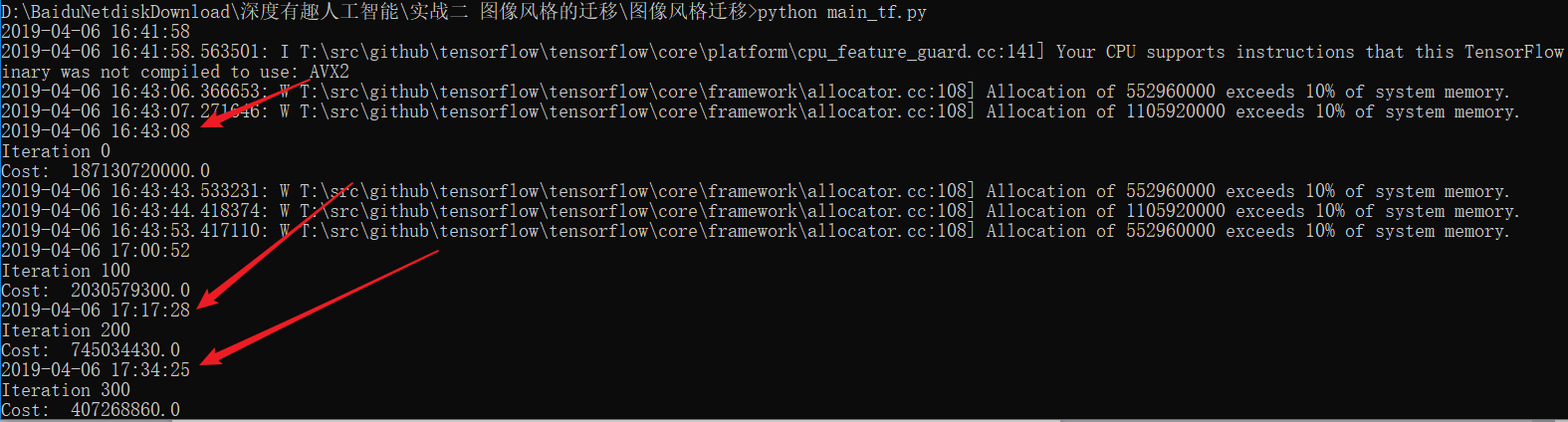
save\_image(os.path.join(OUTPUT\_DIR, 'output\_%d.jpg' % i), output\_image)

**7.结果展示**

（1）运行

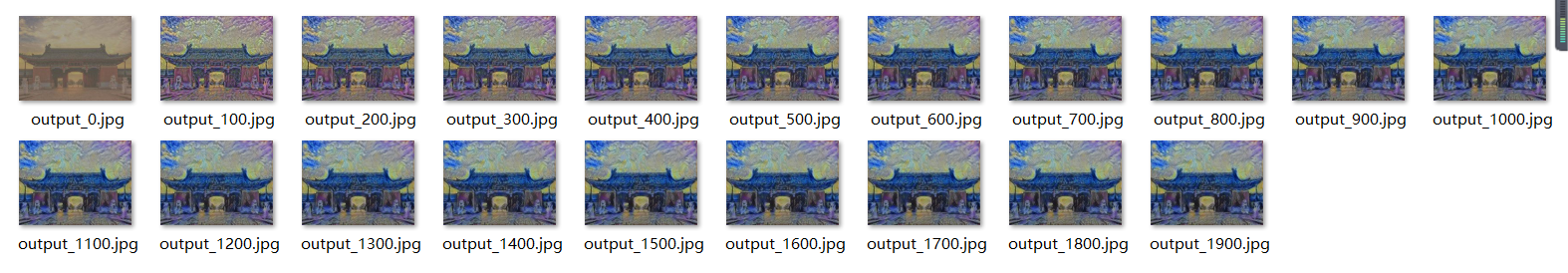
python main\_tf

（2）运行过程



可以看出在本机的cpu上跑非常慢，很慢，，，，，

（3）结果展示



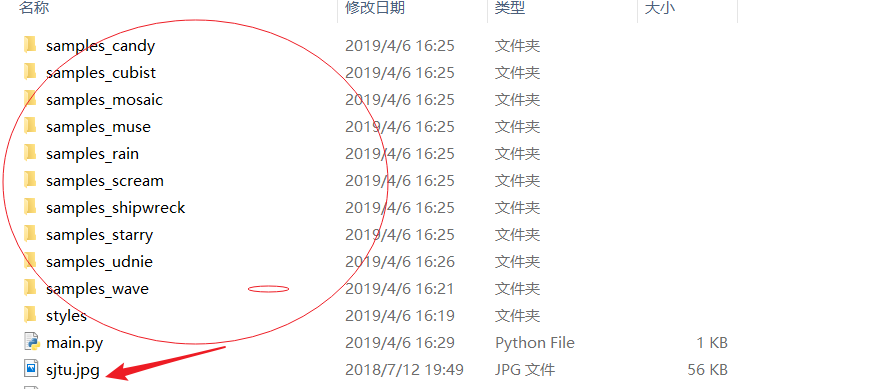
**三、方法二TensorFlow实现快速图像风格迁移（Fast Neural Style Transfer）**

**1.简介**

使用TensorFlow实现快速图像风格迁移（Fast Neural Style Transfer）

**2.准备**

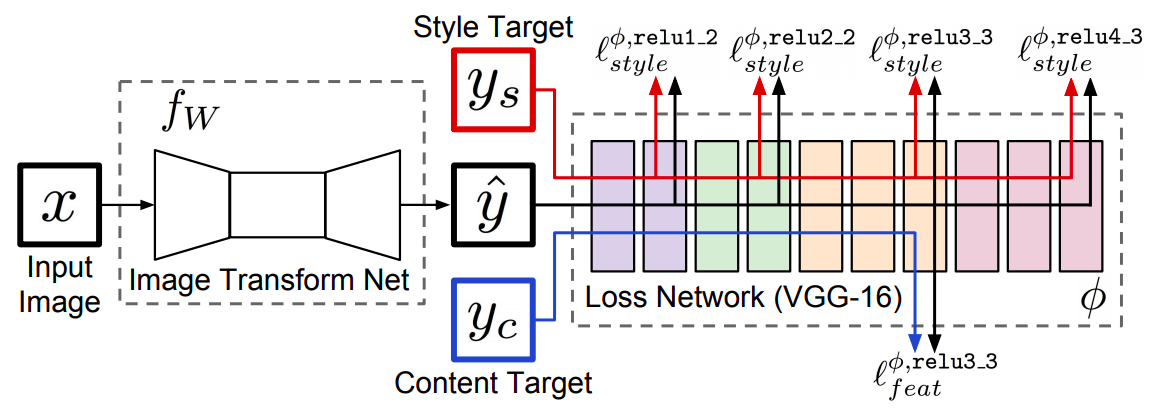
风格图及原图



**3.原理**

在之前介绍的图像风格迁移中，可以根据内容图片和风格图片优化输入图片，使得内容损失函数和风格损失函数尽可能小。

使用快速图像风格迁移可大大缩短生成一张迁移图片所需的时间，其模型结构如下，包括转换网络和损失网络。



风格图片是固定的，而内容图片是可变的输入，因此以上模型用于将任意图片快速转换为指定风格的图片

转换网络：参数需要训练，将内容图片转换成迁移图片

损失网络：计算迁移图片和风格图片之间的风格损失，以及迁移图片和原始内容图片之间的内容损失。

经过训练后，转换网络所生成的迁移图片，在内容和输入的内容图片相似，在风格上和指定的风格图片相似

进行推断时，仅使用转换网络，输入内容图片，即可得到对应的迁移图片

如果有多个风格图片，对每个风格分别训练一个模型即可。

**4.实现代码**

# -\*- coding: utf-8 -\*-

import tensorflow as tf

import numpy as np

from imageio import imread, imsave

import os

import time

def the\_current\_time():

print(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(int(time.time()))))

style = 'udnie'

model = 'samples\_%s' % style

content\_image = 'sjtu.jpg'

result\_image = 'sjtu\_%s.jpg' % style

X\_image = imread(content\_image)

sess = tf.Session()

sess.run(tf.global\_variables\_initializer())

saver = tf.train.import\_meta\_graph(os.path.join(model, 'fast\_style\_transfer.meta'))

saver.restore(sess, tf.train.latest\_checkpoint(model))

graph = tf.get\_default\_graph()

X = graph.get\_tensor\_by\_name('X:0')

g = graph.get\_tensor\_by\_name('transformer/g:0')

the\_current\_time()

gen\_img = sess.run(g, feed\_dict={X: [X\_image]})[0]

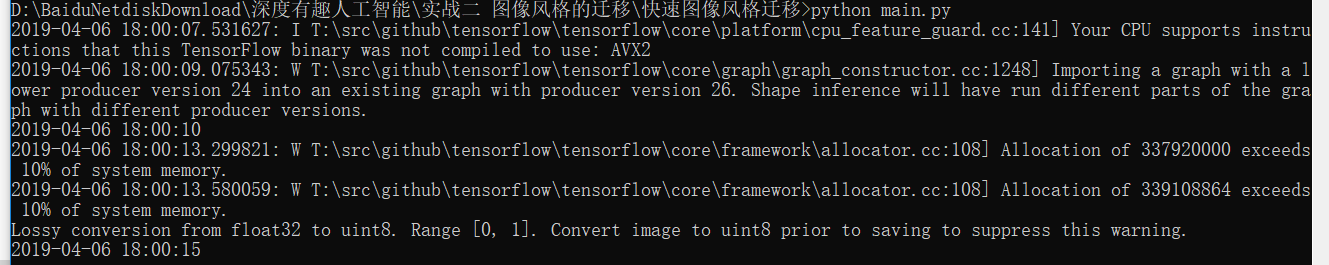
gen\_img = np.clip(gen\_img, 0, 255) / 255.

imsave(result\_image, gen\_img)

the\_current\_time()

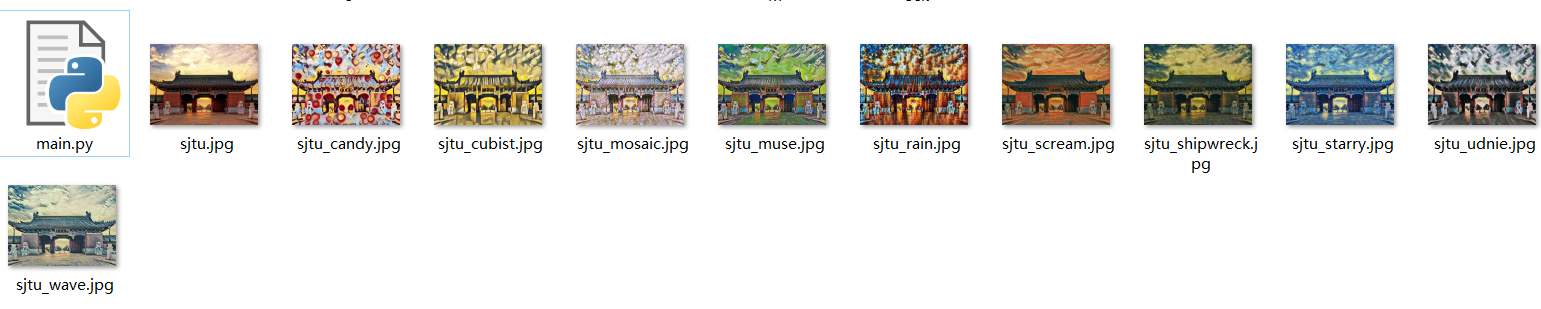
**5.运行结果展示**

（1）过程展示



速度很快，在本机CPU中5s左右就完成了

（2）结果展示



**四、方法三TensorFlow实现图像风格迁移**

前面都是在CPU上跑的程序，可我就想试试在GPU上跑，怎么办？怎么办？

那么方法来了，谷歌提供了Google Colob可以免费试用GPU来跑代码，那么赶快来试试吧！！！

**1.外网访问**

谷歌访问助手破解版，无需设置推广网站的地址

<https://github.com/haotian-wang/google-access-helper>

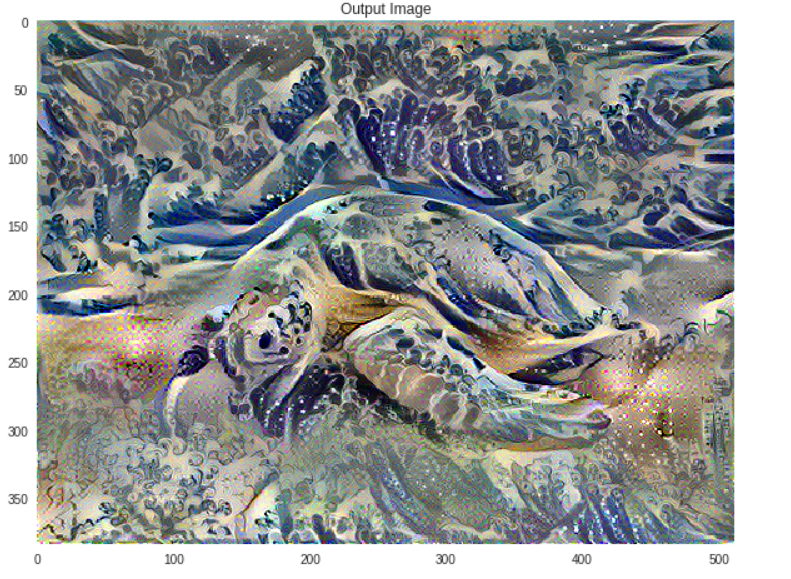
**2.Google Colab免费GPU服务器使用教程**

参考一个博主提供的文章

<https://blog.csdn.net/cocoaqin/article/details/79184540>

**3.例子**





在此过程中，我们按照下列步骤中进行

a.可视化数据

b.基本预处理/准备数据

c.设置损失功能

d.创建模型

e.优化损失功能

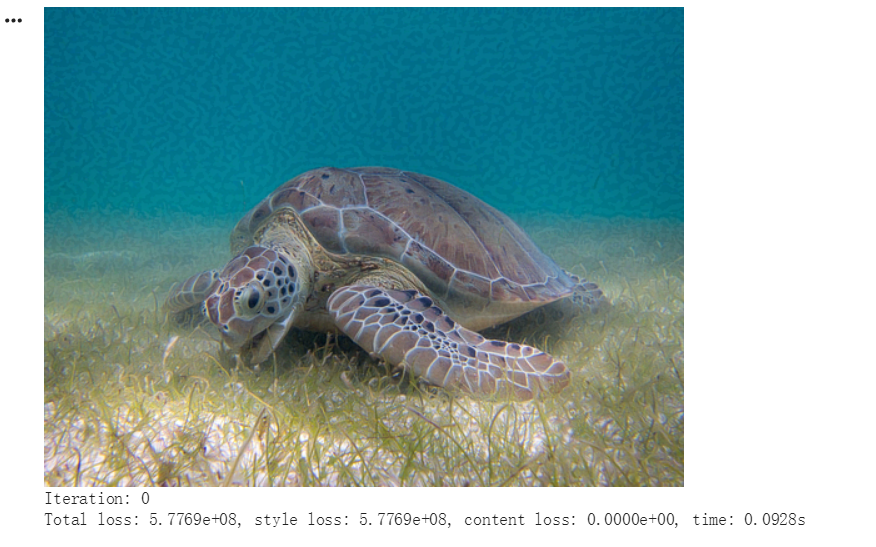
**4.代码展示**

（1）下载图片到文件中

（2）运行代码

**5.结果展示**

**（1）过程展示**

****

**（2）最终结果**

****

**五、项目地址**

链接：https://pan.baidu.com/s/1MydlfdfLRYUigZVhKwj7kw

提取码：utmt

复制这段内容后打开百度网盘手机App，操作更方便哦