通过 Tensorflow 实现 Neural Style-Transfer

源码索引：https://codeload.github.com/anishathalye/neural-style/zip/master

实现环境： TensorFlow

NumPy

SciPy

Pillow

Win 10 系统 cmd 模块

Pycharm 调试

一、实现原理

1、卷积神经网络（CNN）

卷积神经网络是深层神经网络中处理图像最强大的一个类别。卷积神经网络由一层层小的计算单元（神经元）组成，可以以前馈的方式分层地处理视觉上的信息（图 1）。每一层中的计算单元（神经元）可以被理解为是对过滤图像信息的收集，也就是说，每一个神经元都会从输入的图像中抽取某个特征。

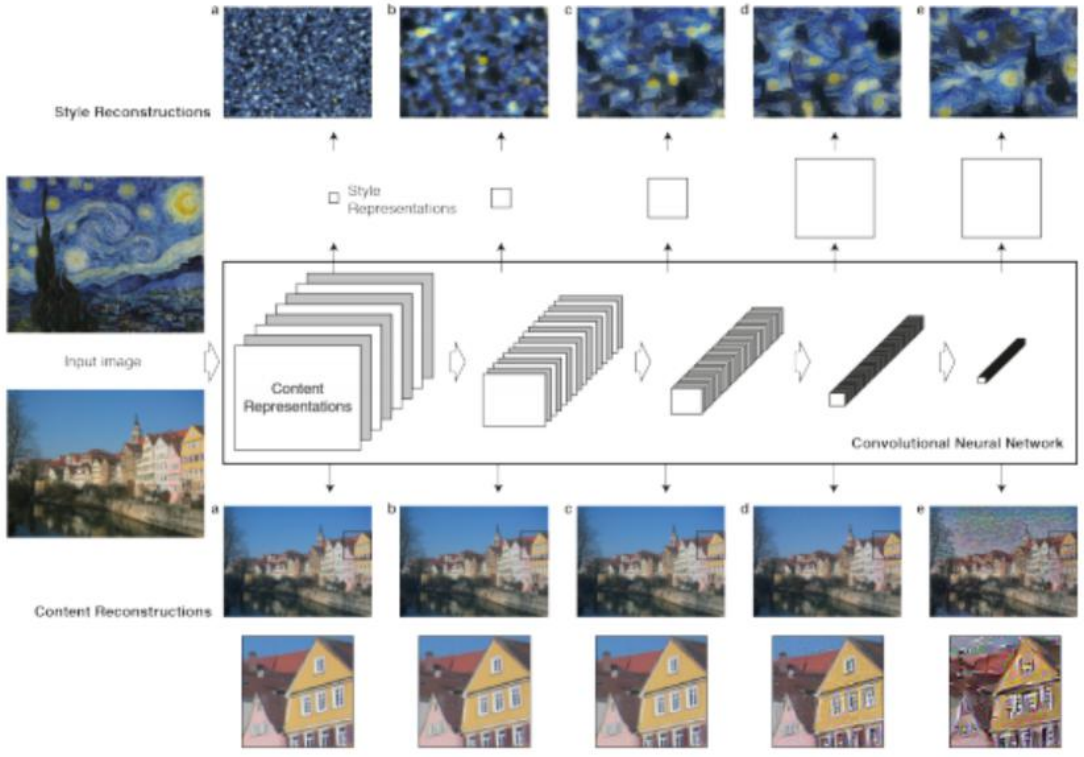


图 1 卷积神经网络分层抽取图片信息

即总结为如下过程：

a.每个卷积层前一层输出的一组特征图层是内容表征（**Content**）；

b.神经网络某一个是有多个神经元组成，每个神经元输出的是一个特征图层。

c.神经网络层输出的一组特征图层，使他们两两求相关性，该相关性就是风格表征（**Style**）。

输入一张图片，会在卷积神经网的各层以一系列过滤后的图像表示。随着层级的一层一层处理，过滤后的图片会通过向下取样的方式不断减小（比如通过池化层）。这使得每层神经网的神经元数量会原来越小。（也就是层越深，因为经过了池化层，单个特征图层会越来越小，于是每层中的神经元数量也会越来越少）。

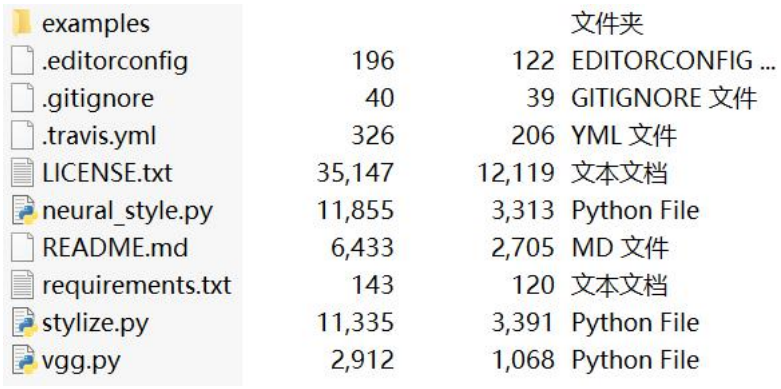
2、内容重塑和风格重塑

A、根据图 1 可以看出，每层的输出结果通过重塑入图层而逐渐展现不同的图像信息；

B、在原始的 CNN 表征之上，建立了一个新的特征空间(feature space)，这个特征空空间捕获了输入图像的风格。风格的表征计算了在 CNN 的不同层级间不用特征之间的相似性。通过在 CNN 隐层的不同的子集上建立起来的风格的表征，重构输入图像的风格。如此，便创造了与输入图像一致的风格而丢弃了全局的内容。

（可以将这种过程理解为两个不同图形的风格和内容的融合与重塑）

二、实现过程



examples			文件夹
.editorconfig	196	122	EDITORCONFIG ...
.gitignore	40	39	GITIGNORE 文件
.travis.yml	326	206	YML 文件
LICENSE.txt	35,147	12,119	文本文档
neural_style.py	11,855	3,313	Python File
README.md	6,433	2,705	MD 文件
requirements.txt	143	120	文本文档
stylize.py	11,335	3,391	Python File
vgg.py	2,912	1,068	Python File

图 2 源码包截图

Tensor Flow 的源码主要包含了三个代码文件：neural_style.py, stylize.py 和 vgg.py。

1、neural_style.py

外部接口函数，定义了函数的主要参数以及部分参数的默认值，包含对图像的读取和存贮，对输入图像进行 resize，权值分配等操作，并将参数以及 resize 的图片传入 stylize.py 中。

2、Stylize.py

Style-Transfer 实现的核心代码，包含了训练、优化等过程。

3、Vgg.py

定义了网络模型以及相关的运算。

三、程序运行和调试

（由于之前基本没有接触过 Python 及具体实现过程，导致我在初期实现代码时经常一

头雾水，陷入“看不懂—查资料—看不懂”的恶性循环之中…）

1、VGG-19

在正式开始运行调试之前，不得不提实现图像处理时经常用到的神经网络模型——VGG-19；

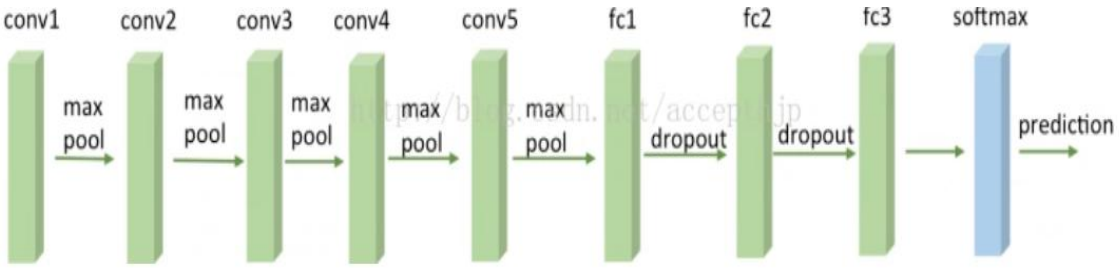


图 3　VGG-19 模型的基本结构

前几层为卷积和 maxpool 的交替，每个卷积包含多个卷积层，最后面再紧跟三个全连接层。具体而言，第一个卷积包含 2 个卷积层，第二个卷积包含 2 个卷积层，第三个卷积包含 4 个卷基层，第四个卷积包含 4 个卷积层，第五个卷积包含 4 个卷基层，所以一共有 16 个卷积层，加上 3 个全连接层，一共 19 层，因此称为 VGG-19 模型。

该模型大小约五百 M，需要从官方网站下载，下载完成后拖入“Profile Project”，以便于程序的调用和数据导入（“vgg.py”模块）：

2、vgg.py 模块

既然介绍完 VGG-19 模型，不妨直接开始理解与该模型相关联的程序”vgg.py”模块吧；我们要实现的 Neural Style 只依赖于 VGG-19 的卷积层，故在程序中引用如下：

```
VGG19_LAYERS = (
    'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

    'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

    'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
    'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

    'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
    'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

    'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
    'relu5_3', 'conv5_4', 'relu5_4'
)
```

图 4　读取我们需要使用的 VGG-19 模块

该数据包含很多信息，我们需要的信息是每层神经网络的 kernels 和 bias。

kernels 的获取方式是 data['layers'][0][第 i 层][0][0][0][0][0]，形状为[width, height, in_channels, out_channels]：

```
def net_preloaded(weights, input_image, pooling):
    net = {}
    current = input_image
    for i, name in enumerate(VGG19_LAYERS):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channels, out_channels]
            # tensorflow: weights are [height, width, in_channels, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
            bias = bias.reshape(-1)
            current = _conv_layer(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current)
        elif kind == 'pool':
            current = _pool_layer(current, pooling)
        net[name] = current

    assert len(net) == len(VGG19_LAYERS)
    return net
```

图 5　获取 kernels

bias 的获取方式是 data['layers'][0][第 i 层][0][0][0][0][0]，形状为[1,out_channels]：

```
def _conv_layer(input, weights, bias):
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1, 1, 1, 1),
            padding='SAME')
    return tf.nn.bias_add(conv, bias)
```

图 6　获取 bias

3、neural_style.py 模块

在该模块，可以看到定义了很多参数和接口：

```
# default arguments
CONTENT_WEIGHT = 5e0
CONTENT_WEIGHT_BLEND = 1
STYLE_WEIGHT = 5e2
TV_WEIGHT = 1e2
STYLE_LAYER_WEIGHT_EXP = 1
LEARNING_RATE = 1e1
BETA1 = 0.9
BETA2 = 0.999
EPSILON = 1e-08
STYLE_SCALE = 1.0
ITERATIONS = 1000
VGG_PATH = 'imagenet-vgg-verydeep-19.mat'
POOLING = 'max'
```

图 7　定义参数

（定义接口代码过长，在此不再展示，可移步源码:>）

4、stylize.py 模块

迭代层数函数：

```
def stylize(network, initial, initial_noiseblend, content, styles, preserve_colors, iterations,
        content_weight, content_weight_blend, style_weight, style_layer_weight_exp, style_blend_weights,
        learning_rate, beta1, beta2, epsilon, pooling,
        print_iterations=None, checkpoint_iterations=None):
```

图 8

计算内容图层（content feature）：

```
    g = tf.Graph()
    with g.as_default(), g.device('/cpu:0'), tf.Session() as sess:
        image = tf.placeholder('float', shape=shape)
        net = vgg.net_preloaded(vgg_weights, image, pooling)
        content_pre = np.array([vgg.preprocess(content, vgg_mean_pixel)])
        for layer in CONTENT_LAYERS:
            content_features[layer] = net[layer].eval(feed_dict={image: content_pre})
```

图 9

计算风格图层（style feature）：

```
for i in range(len(styles)):
    g = tf.Graph()
    with g.as_default(), g.device('/cpu:0'), tf.Session() as sess:
        image = tf.placeholder('float', shape=style_shapes[i])
        net = vgg.net_preloaded(vgg_weights, image, pooling)
        style_pre = np.array([vgg.preprocess(styles[i], vgg_mean_pixel)])
        for layer in STYLE_LAYERS:
            features = net[layer].eval(feed_dict={image: style_pre})
            features = np.reshape(features, (-1, features.shape[3]))
            gram = np.matmul(features.T, features) / features.size
            style_features[i][layer] = gram
```

图 10

5、程序运行过程

A、将“imagenet-vgg-verydeep-19.mat”导入目录；

B、电脑“开始”，进入 Cmd 命令提示符，进入 Tensor flow 模式：

```
命令提示符
Microsoft Windows [版本 10.0.17763.1217]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\microapple>activate tensorflow
```

图 11 “activate tensorflow”

C、进入源码所在目录：

```
(tensorflow) C:\Users\microapple>cd Desktop

(tensorflow) C:\Users\microapple\Desktop>cd Neural-Style-master\Neural-Style-master

(tensorflow) C:\Users\microapple\Desktop\Neural-Style-master\Neural-Style-master>
```

图 12

D、run the commend ： python neural_style.py --content examples/cat.jpg --styles examples/2-style1.jpg --output y-output.jpg

E、等待结果

（耗时很长）与示例结果相似：

图 13 原图片

图 14 Style-Transfer 结果

四、遇到的问题与解决方法

1、运行环境的搭建

在不了解 Python 程序，甚至不常使用 Github 的菜鸟级别的人，光是对程序运行环境的搭建就让我非常头痛：最开始我选择的代码是比较复杂的，对实现环境要求很高（需要 LINUX 命令、或者要求下载的数据包太大）。所以我选择了容易实现，相对不太复杂的 deep-learning 程序，即 Style-Transfer。

了解到 Anaconda 环境，下载完成后，在其子模块“Anaconda Prompt”下载 Tensor Flow 等环境。

当我按照步骤“pip install TensorFlow”后，我就束手无策了：程序如何运行？在哪里运行？于是我下载了 Pycharm 成功打开了源码目录，并能够调试和察看相关程序。

之后我才得知其实在 cmd 命令符里面就可以运行。

2、代码调试中遇到的问题（error）

由于计算机的版本、实现环境版本的不同，我第一手拿到的程序并不是直接就能在“README”的协助下就跑通的。过程中遇到的问题很多：

A、“error：the following arguments are required：--content --styles --output ”

这个问题是我在 Pycharm 上跑程序遇到的，着实令我困扰了很久...通过搜索引擎得到的原因是“符号'-'的问题”，我照做解决方案后，依然报错——我意识到每个人遇到的问题都可能是不同的。经过排查和跟其他人遇到问题的情形相比较，原来是因为计算机不知道“--content”、“--styles”、“--output”所在的目录，或者说不知道它们对应什么。

明白原因后，我立刻找到了解决方案：

①、在 Pycharm 界面，依次点击 Run>Edit Configurations，进入如下界面：

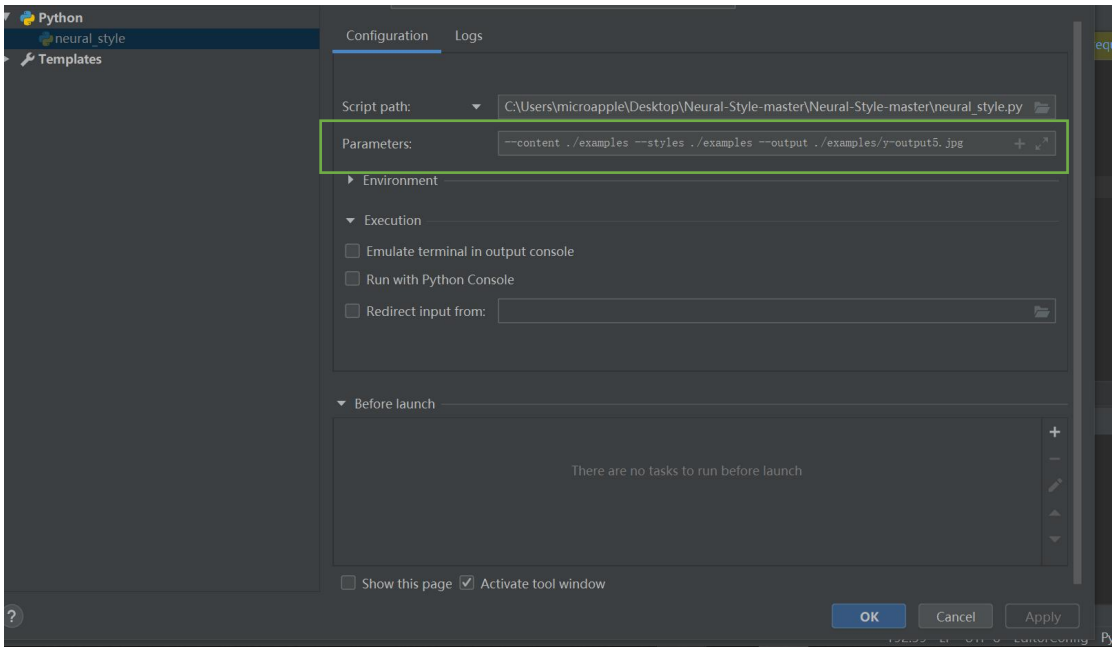


图 15 Edit Configurations 界面

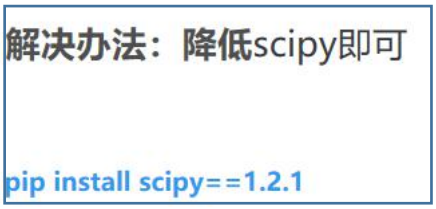
在参数栏设定好相关参数位置即可；

②、也可以在程序运行界面下的“Terminal”栏，设定相关参数。也可以很好的解决问题。

③、或者选择不在 Pycharm 界面运行，直接在 cmd 栏输入“python neural_style.py --content examples/cat.jpg --styles examples/2-style1.jpg --output y-output.jpg”，同样可以设定参数。

B、遇到报错“AttributeError: module 'scipy.misc' has no attribute 'imread'”

这个问题我通过搜索引擎找到了解决方案：



报错的原因是我的 Scipy 模块版本过高，使得程序运行环境与其不兼容导致的。因此在我手动进入“Anaconda Prompt”下载其较低的版本后，解决了问题。

五、参考资料

1、源码索引：https://codeload.github.com/anishathalye/neural-style/zip/master

2、图像风格迁移（Neural Style）简史 https://zhuanlan.zhihu.com/p/26746283

3、TensorFlow 实战：Neural Style：https://segmentfault.com/a/1190000009820053

4、【Paper 翻译】A Neural Algorithm Artistic Style：

http://blog.csdn.net/sinat_33761963/article/details/53521292