Lab 8: 风格迁移

图片风格迁移是指将一幅图像（目标风格图像/style image）的风格特征，融合到另一幅图像（内容图像/content image）中，从而生成一幅新的图像。这幅新图像既保留了内容图像的主要内容，又具有了风格图像的风格特征。

在实现风格迁移的过程中，通常会使用到深度学习技术，比如可以使用循环生成对抗网络（Cycle GANs），或采用卷积神经网络（如VGG19网络），这类网络能够提取图像的特征，并通过优化算法，将风格图像的特征与内容图像的特征进行融合，从而生成新的图像。

核心思想：图像风格迁移技术的核心思想是将内容图像和风格图像的特征进行融合。通过提取并融合两幅图像的特征，生成具有新的风格和内容的图像。

In [1]:
```
# 安装并导入相应的库
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (10,10)
mpl.rcParams['axes.grid'] = False

import numpy as np
from PIL import Image
import time
import functools
```

In [2]:
```
# 导入tensorflow框架
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory as kp_image
from tensorflow.python.keras import models
from tensorflow.python.keras import losses
from tensorflow.python.keras import layers
from tensorflow.python.keras import backend as K
from tensorflow.keras.preprocessing import image
```

举例

将中国地质大学（北京）的校门照片更换为梵高的《星空》风格。

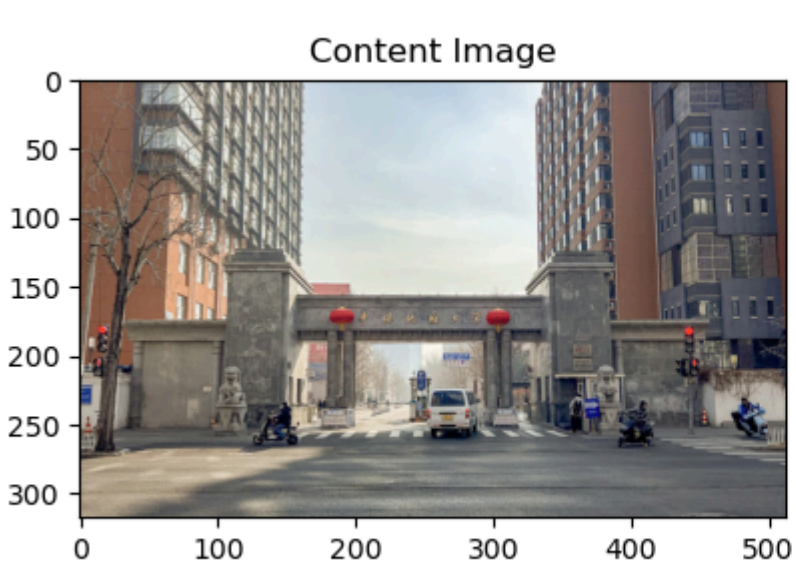
In [3]:
```
# 设置图片路径
content_path = 'lab8_img/campus.jpg' # 需要修改风格的图片
style_path = 'lab8_img/starry_light.jpg'          # 风格的参考图片
```

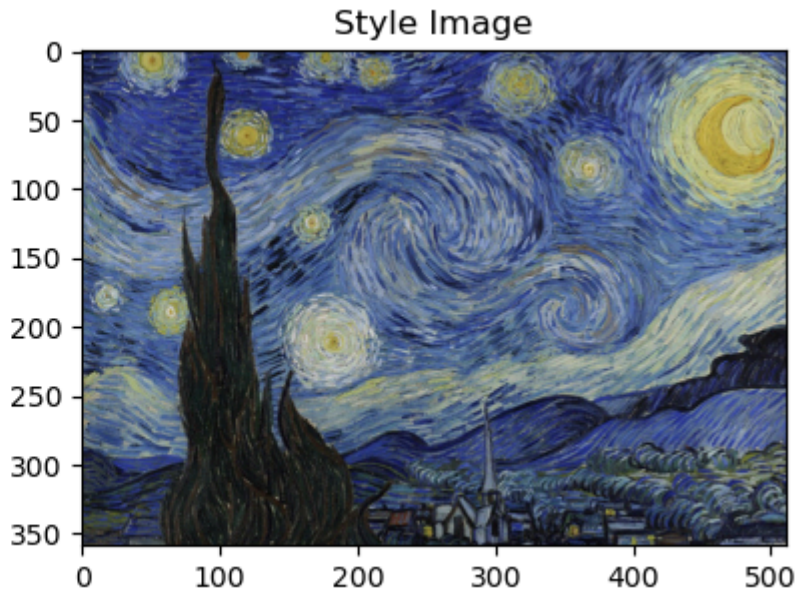
In [4]:
```
def load_img(path_to_img):
    # 重新定义图像的维度
    max_dim = 512
    pic = image.load_img(path_to_img)
    long = max(pic.size)
    scale = max_dim/long
    pic = pic.resize((round(pic.size[0]*scale), round(pic.size[1]*scale)), Image.ANTIALIAS)
    pic = np.asarray(pic)
    # 为图像数组加一个维度，表示批量
    pic = np.expand_dims(pic, axis=0)
    return pic
```

In [5]:
```
def imshow(img, title=None):
    # 显示图片时，需要去掉批量维度
    out = np.squeeze(img, axis=0)
    out = out.astype('uint8')
    plt.imshow(out)
    if title is not None:
        plt.title(title)
    plt.imshow(out)
```

In [6]:
```
# 预览内容图像和目标风格图像

plt.figure(figsize=(10,10))

content = load_img(content_path).astype('uint8')
style = load_img(style_path).astype('uint8')

plt.subplot(1, 2, 1)
imshow(content, 'Content Image')

plt.subplot(1, 2, 2)
imshow(style, 'Style Image')
plt.show()
```

```
/var/folders/j9/_9grtkw11ks6kykr4_bdfqmr0000gn/T/ipykernel_10122/2424415367.py:7: DeprecationWarning: AN
TIALIAS is deprecated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS or Resampling.LANCZOS i
nstead.
  pic = pic.resize((round(pic.size[0]*scale), round(pic.size[1]*scale)), Image.ANTIALIAS)
```





设置神经网络

In [7]:
```
# 用VGG19学习图像的特征，需要对图片进行针对VGG19的预处理，包括各维度的归一化、去均值化处理等
def load_and_process_img(path_to_img):
    img = load_img(path_to_img)
    img = tf.keras.applications.vgg19.preprocess_input(img)
    return img
```

In [8]:
```
def deprocess_img(processed_img):
    #检查图像的尺寸是否正确，如果不是 3 或 4，则会引发错误
    x = processed_img.copy()
    if len(x.shape) == 4:
        x = np.squeeze(x, 0)
    assert len(x.shape) == 3, ("检查输入图片维度")
    if len(x.shape) != 3:
        raise ValueError("维度错误")

    # 预处理时，图像的每个色彩通道做了去均值化处理
    # 这里需要对其进行恢复，即取均值化的逆向操作，这样显示出的图片才更像原来的图片
    # 三个通道的恢复值来自Keras中的VGG预设模型

    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    x = x[:, :, ::-1]

    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

VGG19把串联的3✖ 3卷积层组成了5个堆叠群（block），其中前两个block有两个卷积层，后三个block各有4个卷积层。每个block之间有一个池化层。

对于内容，我们使用block5中的第二个卷积层（block5_conv2）来做输入“图片内容”的特征训练。

对于“目标风格”，我们使用每个block中的第一个卷积层来学习。

In [9]:
```
# 从block5的第二个卷积层的特征图中，学习图片的内容
content_layers = ['block5_conv2']

# 分别用5个block的第一个卷积层学习目标风格
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
               ]

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

In [10]:
```
def get_model():
    """ Creates our model with access to intermediate layers.
    创建模型
    首先采用transfer learning的思路，
    加载已经在ImageNet任务中训练好的vgg_model，但不采用其输出层（重新训练输出层）。
    冻结除了输出层之外的其他所有层次（vgg_model.trainable = False）。

    然后，获取目标风格和图片内容的输出值（遍历上一节中指定的那些关联了图片内容和目标风格的层）。
    然后将这些输出值与VGG19的输入一起使用，以创建可以访问VGG19中间层的新模型，即get_model()返回 Keras 模型。
    该模型输出已训练的 VGG19 模型中的有关目标风格和图片内容的层次。

    返回一个函数，该函数将输出一个图像，目标是：使该图像和相应特征层上的输入内容及风格之间的差异都达到最小化
    """
    # 导入在ImageNet任务里已经训练好的VGG19模型
    vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
    vgg.trainable = False
    # 根据前面定义的图片内容和目标风格的记录层，重新训练输出层
    style_outputs = [vgg.get_layer(name).output for name in style_layers]
    content_outputs = [vgg.get_layer(name).output for name in content_layers]
    model_outputs = style_outputs + content_outputs
    # 创建模型
    return models.Model(vgg.input, model_outputs)
```

计算损失

In [11]:
```
# 计算“图片内容”部分的损失
def get_content_loss(base_content, target):
    return tf.reduce_mean(tf.square(base_content - target))
```

In [12]:
```
def gram_matrix(input_tensor):
    # We make the image channels first
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram / tf.cast(n, tf.float32)

# 计算“目标风格”部分的损失
def get_style_loss(base_style, gram_target):
    """输入两个图片维度（高、宽、深）"""
    # 高、宽、深度（即上一层的卷积核数）
    # 我们根据特征图的大小和滤波器的数量，来调整特定层的损失值
    height, width, channels = base_style.get_shape().as_list()
    gram_style = gram_matrix(base_style)

    return tf.reduce_mean(tf.square(gram_style - gram_target)) # / (4. * (channels ** 2) * (width * heigh
```

In [13]:
```python
def get_feature_representations(model, content_path, style_path):
    """辅助函数，用于计算内容和风格特征的中间表示

    该函数将直接加载内容图像和风格图像的路径，并做好预处理。然后将它们送入神经网络，获得中间层的输出。

    传入参数:
      model: 使用的模型
      content_path: 内容图像的路径
      style_path: 目标风格图像的路径

    返回:
      风格特征图和内容特征图
    """
    # 导入图像
    content_image = load_and_process_img(content_path)
    style_image = load_and_process_img(style_path)

    # 把图像送入神经网络进行训练
    style_outputs = model(style_image)
    content_outputs = model(content_image)


    # 获得风格特征图和内容的特征图
    style_features = [style_layer[0] for style_layer in style_outputs[:num_style_layers]]
    content_features = [content_layer[0] for content_layer in content_outputs[num_style_layers:]]
    return style_features, content_features
```

In [14]:
```python
def compute_loss(model, loss_weights, init_image, gram_style_features, content_features):
    """计算损失函数.

    传入参数:
      model: 所用的模型
      loss_weights: 目标风格和内容图像都会产生损失，但是我们可以调整它们之间的权重
        （风格的权重，内容的权重，以及总权重）
      init_image: 输入的原始图像。优化过程就是在更新这幅图像，也就是说，计算损失的梯度会被应用于该图像。
     gram_style_features: 与前面定义的目标风格层对应的预计算格矩阵。
      content_features: 从前面定义的内容层（第五个block的第2层）中计算的输出结果。

    返回:
      风格的权重，内容的权重，以及总权重
    """
    style_weight, content_weight = loss_weights

    # 通过模型生成初始图像。这将为我们提供所有相关层的图像内容和目标风格的特征表示。
    model_outputs = model(init_image)

    style_output_features = model_outputs[:num_style_layers]
    content_output_features = model_outputs[num_style_layers:]

    style_score = 0
    content_score = 0

    # 累计所有层的风格损失。在这个例子里，我们对每层的赋予相同的权重。
    weight_per_style_layer = 1.0 / float(num_style_layers)
    for target_style, comb_style in zip(gram_style_features, style_output_features):
        style_score += weight_per_style_layer * get_style_loss(comb_style[0], target_style)

    # 在所有层上累计内容损失
    weight_per_content_layer = 1.0 / float(num_content_layers)
    for target_content, comb_content in zip(content_features, content_output_features):
        content_score += weight_per_content_layer* get_content_loss(comb_content[0], target_content)

    style_score *= style_weight
    content_score *= content_weight

    # 获得总损失
    loss = style_score + content_score
    return loss, style_score, content_score
```

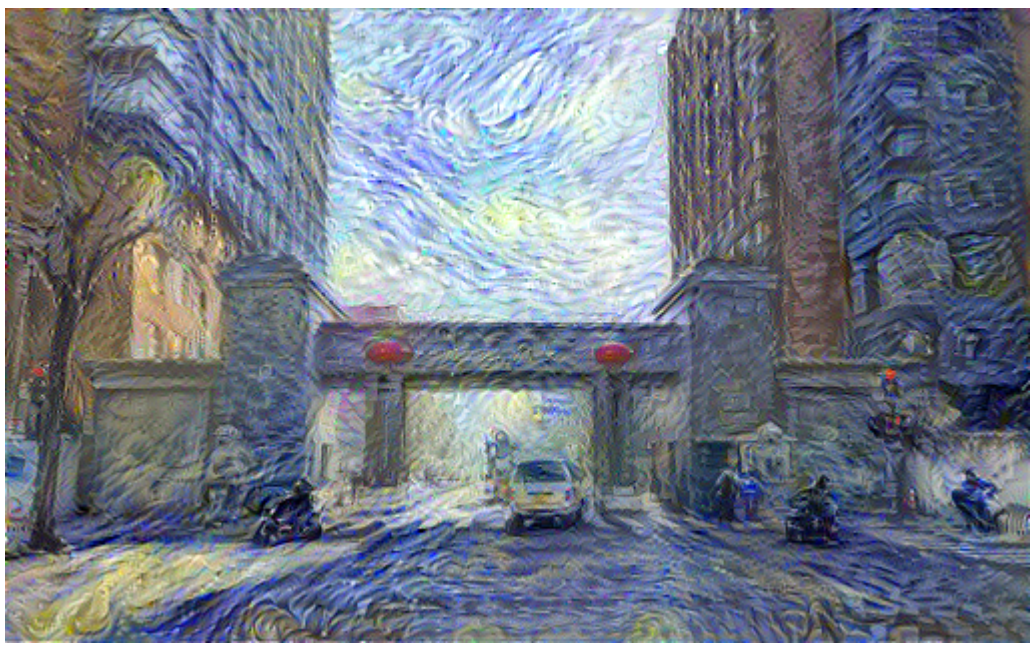
梯度及优化

In [15]:
```python
# 计算梯度
def compute_grads(cfg):
    with tf.GradientTape() as tape:
        all_loss = compute_loss(**cfg)
    total_loss = all_loss[0]
    return tape.gradient(total_loss, cfg['init_image']), all_loss
```

In [16]:
```
# 训练模型并显示结果
import IPython.display

def run_style_transfer(content_path,
                       style_path,
                       num_iterations=1000,
                       content_weight=1000,
                       style_weight=0.01):
    # 我们只训练输出层，对于其它层我们只获得和目标风格和内容有关的损失值，不训练它们的参数，所以把trainable设为false
    model = get_model()
    for layer in model.layers:
        layer.trainable = False

    # 从指定的中间层获取风格和内容特征表征
    style_features, content_features = get_feature_representations(model, content_path, style_path)
    gram_style_features = [gram_matrix(style_feature) for style_feature in style_features]

    # 设定初始图片
    init_image = load_and_process_img(content_path)
    init_image = tf.Variable(init_image, dtype=tf.float32)
    # 利用Adam做优化
    opt = tf.optimizers.Adam(learning_rate=5, epsilon=1e-1)

    # 显示中间过程的图像
    iter_count = 1

    # 存下最好的结果（对应最小的损失值）
    best_loss, best_img = float('inf'), None

    # 设定configuration
    loss_weights = (style_weight, content_weight)
    cfg = {
      'model': model,
      'loss_weights': loss_weights,
      'init_image': init_image,
      'gram_style_features': gram_style_features,
      'content_features': content_features
    }


    start_time = time.time()
    global_start = time.time()

    norm_means = np.array([103.939, 116.779, 123.68]) # 来自Keras中VGG19模型的每一个色彩通道的均值
    # 去均值处理
    min_vals = -norm_means
    max_vals = 255 - norm_means

    imgs = []
    for i in range(num_iterations):
        grads, all_loss = compute_grads(cfg)
        loss, style_score, content_score = all_loss
        opt.apply_gradients([(grads, init_image)])
        clipped = tf.clip_by_value(init_image, min_vals, max_vals)
        init_image.assign(clipped)
        end_time = time.time()

        if loss < best_loss:
          # 更新损失值到当前的最佳损失
            best_loss = loss
            best_img = deprocess_img(init_image.numpy())


        plot_img = init_image.numpy()
        plot_img = deprocess_img(plot_img)
        imgs.append(plot_img)
        IPython.display.clear_output(wait=True)
        IPython.display.display_png(Image.fromarray(plot_img))
        print('迭代次数: {}'.format(i+1))
        print('总损失: {:.4e}, '
            '目标风格损失: {:.4e}, '
            '图片内容损失: {:.4e}, '
            '耗时: {:.4f}s'.format(loss, style_score, content_score, time.time() - start_time))
    print('总耗时: {:.4f}s'.format(time.time() - global_start))


    return best_img, best_loss
```

训练模型

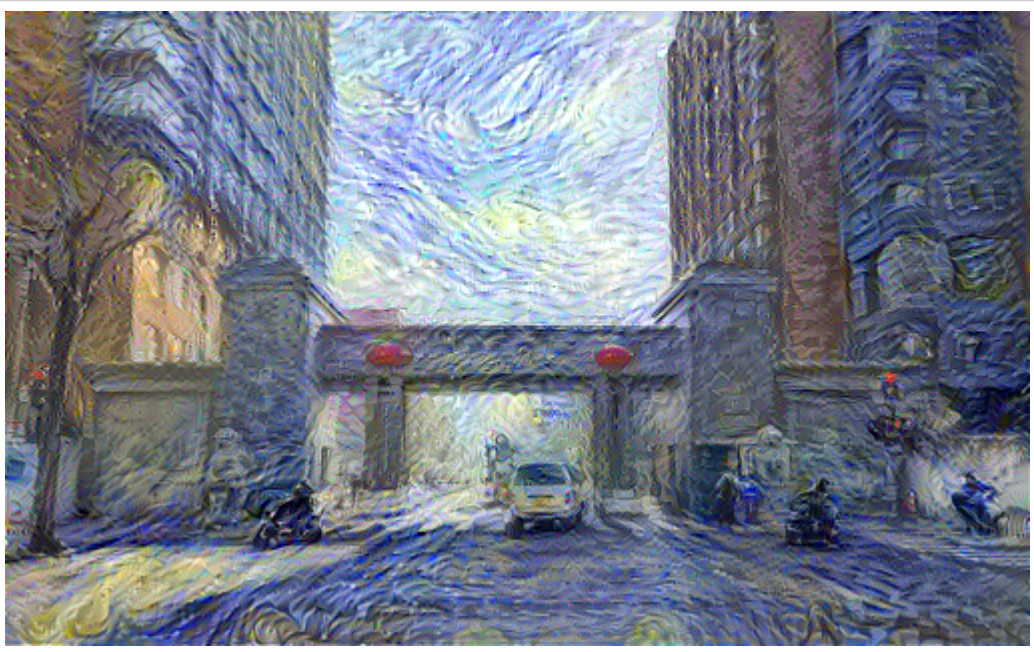
In [17]:
```
# 开始训练
best, best_loss = run_style_transfer(content_path, # 内容图片的路径
                                     style_path, # 目标风格图片的路径
                                     num_iterations=1000, # 迭代次数
                                     content_weight=1000, # 内容权重
                                     style_weight=0.01) # 风格权重
```



```
迭代次数: 1000
总损失: 8.1367e+05, 目标风格损失: 4.0787e+05, 图片内容损失: 4.0580e+05, 耗时: 1133.7979s
总耗时: 1133.7979s
```

In [18]:
```
# 提取最佳结果
Image.fromarray(best)
```

Out[18]:

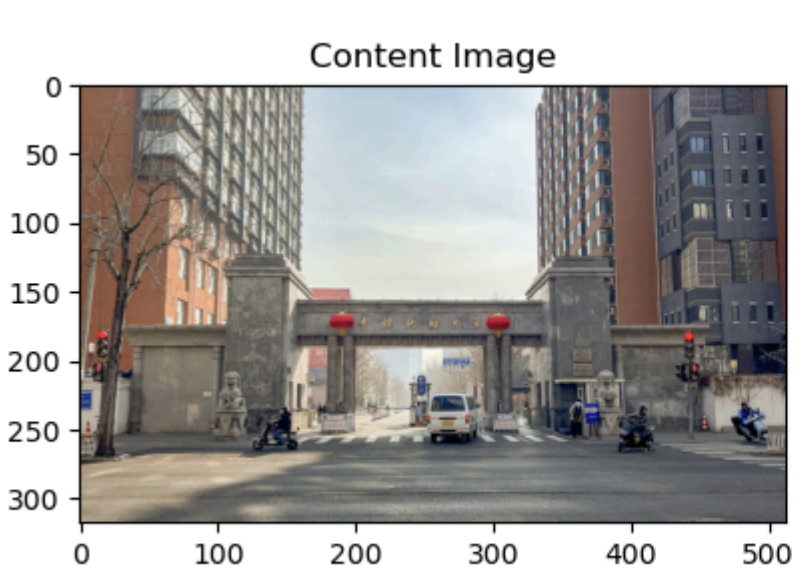


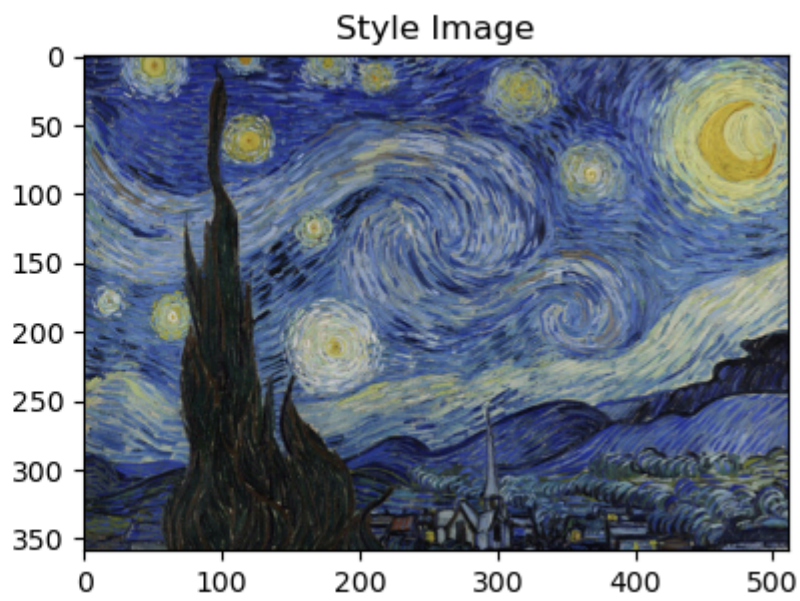
展示结果

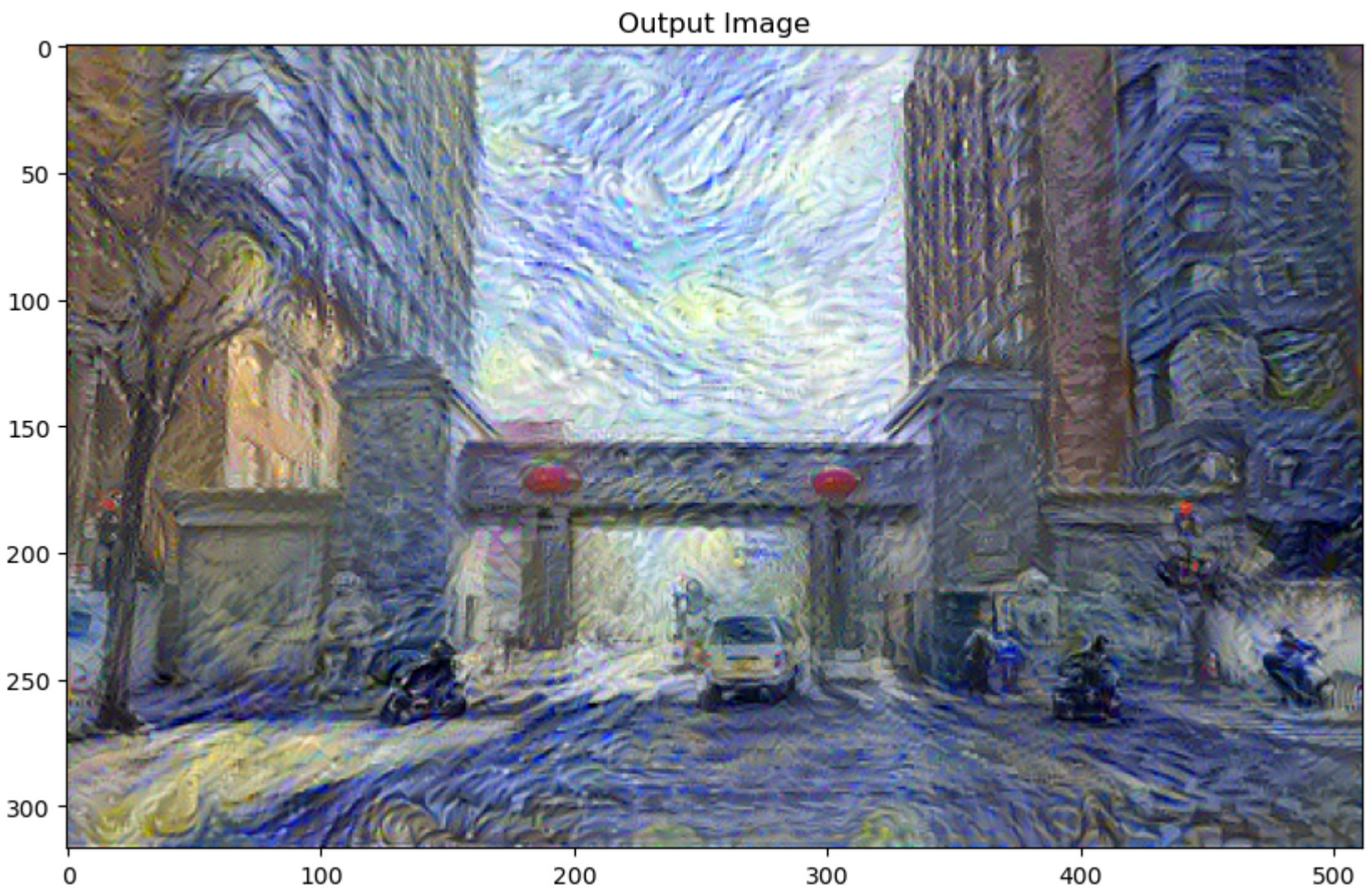
In [19]:
```
# 系统化展示结果
def show_results(best_img, content_path, style_path, show_large_final=True):
    plt.figure(figsize=(10, 5))
    content = load_img(content_path)
    style = load_img(style_path)

    plt.subplot(1, 2, 1)
    imshow(content, 'Content Image')

    plt.subplot(1, 2, 2)
    imshow(style, 'Style Image')

    if show_large_final:
        plt.figure(figsize=(10, 10))

        plt.imshow(best_img)
        plt.title('Output Image')
        plt.show()
```

In [20]:
```
show_results(best, content_path, style_path)
```

```
/var/folders/j9/_9grtkw11ks6kykr4_bdfqmr0000gn/T/ipykernel_10122/2424415367.py:7: DeprecationWarning: AN
TIALIAS is deprecated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS or Resampling.LANCZOS i
nstead.
  pic = pic.resize((round(pic.size[0]*scale), round(pic.size[1]*scale)), Image.ANTIALIAS)
```

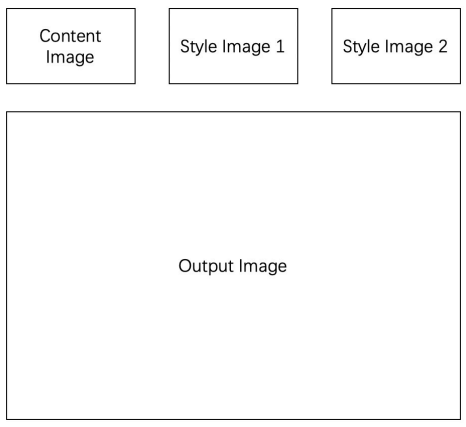






任务

将lab8_img文件夹中的世界地图（world_map.jpg）与**任意两幅不同的艺术作品**的风格进行融合。将最终结果输出为以下格式：



提示：可以先迁移一种风格，再融入第二种风格，通过调整content和style的权重来达到最好效果.

In [ ]:
```
# 在这里完成任务，可以使用额外的cells
```

提交方式

本次作业有1个任务。完成后将ipynb命名为“Lab8+姓名+学号.ipynb”，将导出的PDF命名为“Lab8+姓名+学号.pdf”，并将上述两个文件提交到学习通作业模块的相应位置。另外请在学习通的相应模块填入最终的output image以及所选的**两幅**风格图像。请独立完成练习，截止时间：2024年6月24日23:59，因涉及到期末登分，本次作业提前发放，不允许超时。