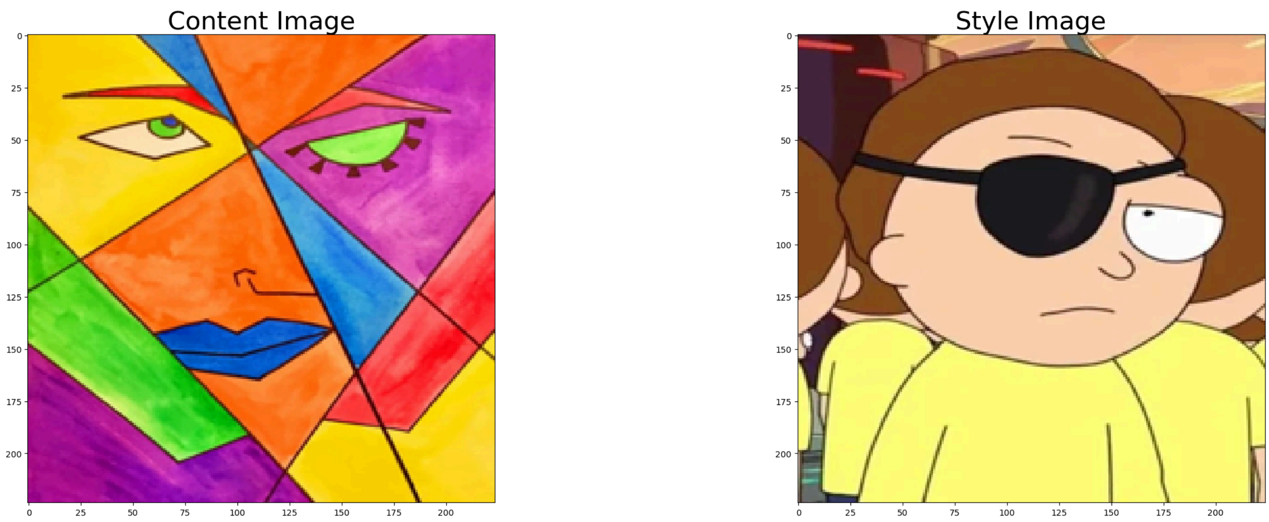
In [1]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torchsummary import summary
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```python
# 加载风格图片
img0 = Image.open("C:/Users/dgq/Pictures/Saved Pictures/abstract-art-by-pablo-picasso.jpg").resize([224,224])
style_img = torch.Tensor(np.array(img0)/255.) # 将图片像素标准化并转化为张量形式
style_img = style_img.permute(2,0,1).unsqueeze(dim = 0) # 将图片尺寸转化为(B,C,H,W)

# 加载内容图片
img1 = Image.open("C:/Users/dgq/Pictures/Saved Pictures/evilmorty_s1_8.webp").resize([224,224])
content_img = torch.Tensor(np.array(img1)/255.) # 将图片像素标准化并转化为张量形式
content_img = content_img.permute(2,0,1).unsqueeze(dim = 0) # 将图片尺寸转化为(B,C,H,W)
```

In [70]:
```python
fig,ax=plt.subplots(1,2) # 切割画板为一行两列个
fig.set_figheight(10) # 确定画板高度
fig.set_figwidth(30) # 确定画板宽度
ax[0].imshow(img0) # 呈现风格图片和内容图片
ax[0].set_title('Content Image', fontsize = 30)
ax[1].imshow(img1)
ax[1].set_title('Style Image', fontsize = 30)
```

Out[70]:
```
Text(0.5, 1.0, 'Style Image')
```



In [61]:
```python
class LossNet(nn.Module):
    def __init__(self, backbone): # 需要传入backbone模型信息
        super(LossNet, self).__init__() # 继承nn.Module的信息
        self.select = ["8", "17", "26"] # 选择这些层的输出为特征比对对象
        self.feature_detector = backbone.features # 提取backbone的特征提取层
        for p in self.parameters(): # 循环模型里的每一个参数
            p.requires_grad = False # 将参数设置为不计算梯度

    def forward(self, x):
        features = [] # 储存提取出的特征
        for name, layer in self.feature_detector.named_children(): # 循环模型内容的编号和网络层
            x = layer(x) # 将x通过网络层计算
            if name in self.select: # 如果是比对对象的输出层
                features.append(x) # 将其加入特征列表
        return features # 返回特征列表
```

In [20]:
```python
vgg19 = torchvision.models.vgg19(pretrained=True) # 加载预训练好的vgg19模型
```

```
d:\Users\dgq\anaconda3\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sin
ce 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
d:\Users\dgq\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=VGG
19_Weights.IMAGENET1K_V1`. You can also use `weights=VGG19_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to C:\Users\dgq/.cache\torch\hub\checkpoints\vgg19-dcbb9e9d.pt
h
100%|██████████| 548M/548M [02:26<00:00, 3.93MB/s]
```

In [63]:
```python
lossnet = LossNet(vgg19).to("cuda").eval() # 将backbone设置为vgg19，并标注进行模型预测，而非训练
summary(lossnet, input_size = (3, 224, 224)) # 总结模型参数
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
           Conv2d-17          [-1, 256, 56, 56]         590,080
             ReLU-18          [-1, 256, 56, 56]               0
        MaxPool2d-19          [-1, 256, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       1,180,160
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
           Conv2d-24          [-1, 512, 28, 28]       2,359,808
             ReLU-25          [-1, 512, 28, 28]               0
           Conv2d-26          [-1, 512, 28, 28]       2,359,808
             ReLU-27          [-1, 512, 28, 28]               0
        MaxPool2d-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
           Conv2d-31          [-1, 512, 14, 14]       2,359,808
             ReLU-32          [-1, 512, 14, 14]               0
           Conv2d-33          [-1, 512, 14, 14]       2,359,808
             ReLU-34          [-1, 512, 14, 14]               0
           Conv2d-35          [-1, 512, 14, 14]       2,359,808
             ReLU-36          [-1, 512, 14, 14]               0
        MaxPool2d-37            [-1, 512, 7, 7]               0
================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 238.30
Params size (MB): 76.39
Estimated Total Size (MB): 315.26
----------------------------------------------------------------
```

In [12]:

```
def gram_matrix(x): # 求解gram矩阵
    (b, ch, h, w) = x.size() # 提取x的形状信息
    features = x.view(b, ch, h*w) # 将高度和宽度拉直
    features_t = features.transpose(1, 2) # 转置矩阵
    gram = features.bmm(features_t) / (ch*h*w) # 计算gran矩阵
    return gram
```

In [58]:

```
def train(model, optimizer, input_img, content_features, style_gram, max_T = 10000): # 模型训练
    style_weight = 1e7 # 风格损失的权重
    content_weight = 1 # 内容损失的权重

    for t in range(max_T): # 循环训练
        optimizer.zero_grad() # 清空优化器梯度
        features = model(input_img) # 得到代训练图片经过模型的输出
        features_gram = [gram_matrix(x) for x in features] # 代训连图片的gram矩阵
        content_loss = F.mse_loss(content_features[1], features[1]) * content_weight # 计算输入和内容第二个特征的吗mse作为内容损失
        style_loss = 0
        for gram1, gram2 in zip(style_gram, features_gram): # 对风格gram和代训连的gram计算每一对矩阵之间的mse作为风格损失
            style_loss += F.mse_loss(gram1, gram2) * style_weight

        loss = content_loss + style_loss # 将两个损失加总作为总损失
        loss.backward() # 反向传播
        optimizer.step() # 梯度下降优化

        if (t+1) % 500 == 0: # 训练过程的结果输出
            print('Step {}: Total Loss: {:4f} - Style Loss: {:4f} - Content Loss: {:4f}'.format(
                t+1, loss.item(), style_loss.item(), content_loss.item()))
```

In [64]:

```
content_features = lossnet(content_img.cuda()) # 得到内容的特征列表
print(len(content_features)) # 打印内容特征列表的长度
print(content_features[0].shape) # 打印第一个元素的大小
style_features = lossnet(style_img.cuda()) # 得到风格的特征列表
style_gram = [gram_matrix(x) for x in style_features] # 计算风格的特征列表里每一个元素的gram矩阵
print(len(style_features)) # 打印风格特征列表的长度
print(style_features[0].shape) # 打印第一个元素的大小
print(len(style_gram)) # 打印风格的特征gram矩阵的列表的长度
print(style_gram[0].shape) # 打印第一个元素的大小
input_img = content_img.clone() # 将内容照片克隆作为输入图片
input_img = input_img.cuda() # 将输入传入GPU
input_img.requires_grad = True # 要求对输入图片的像素计算梯度，对其进行训练
optimizer = torch.optim.Adam([input_img], lr = 0.001) # 选定优化器为Adam
```

```
3
torch.Size([1, 128, 112, 112])
3
torch.Size([1, 128, 112, 112])
3
torch.Size([1, 128, 128])
```

In [65]:

```
train(lossnet, optimizer, input_img, content_features, style_gram) # 对输入图片进行训练
```

```
Step 500: Total Loss: 18.750172 - Style Loss: 13.782828 - Content Loss: 4.967344
Step 1000: Total Loss: 13.766439 - Style Loss: 8.636961 - Content Loss: 5.129478
Step 1500: Total Loss: 11.825233 - Style Loss: 6.669051 - Content Loss: 5.156183
Step 2000: Total Loss: 10.738632 - Style Loss: 5.588904 - Content Loss: 5.149728
Step 2500: Total Loss: 10.019345 - Style Loss: 4.896452 - Content Loss: 5.122893
Step 3000: Total Loss: 9.506028 - Style Loss: 4.414025 - Content Loss: 5.092003
Step 3500: Total Loss: 9.129368 - Style Loss: 4.068510 - Content Loss: 5.060858
Step 4000: Total Loss: 8.838967 - Style Loss: 3.806567 - Content Loss: 5.032400
Step 4500: Total Loss: 8.599688 - Style Loss: 3.595339 - Content Loss: 5.004349
Step 5000: Total Loss: 8.401045 - Style Loss: 3.423503 - Content Loss: 4.977541
Step 5500: Total Loss: 8.231367 - Style Loss: 3.277014 - Content Loss: 4.954354
Step 6000: Total Loss: 8.092190 - Style Loss: 3.158337 - Content Loss: 4.933853
Step 6500: Total Loss: 7.979589 - Style Loss: 3.067094 - Content Loss: 4.912495
Step 7000: Total Loss: 7.882589 - Style Loss: 2.987272 - Content Loss: 4.895318
Step 7500: Total Loss: 7.804623 - Style Loss: 2.925180 - Content Loss: 4.879443
Step 8000: Total Loss: 7.743200 - Style Loss: 2.875688 - Content Loss: 4.867513
Step 8500: Total Loss: 7.688539 - Style Loss: 2.832666 - Content Loss: 4.855873
Step 9000: Total Loss: 7.643973 - Style Loss: 2.798095 - Content Loss: 4.845879
Step 9500: Total Loss: 7.604075 - Style Loss: 2.770035 - Content Loss: 4.834041
Step 10000: Total Loss: 7.570479 - Style Loss: 2.743149 - Content Loss: 4.827330
```

In [68]:

```
result = input_img.data.squeeze(dim = 0).permute(1,2,0) # 获取结果，调整结果格式
fig,ax=plt.subplots(1,3) # 切割画板为一行三列
fig.set_figheight(10) # 确定画板高度
fig.set_figwidth(30) # 确定画板宽度
# 呈现三张图片
ax[0].imshow(img0)
ax[0].set_title('Content Image', fontsize = 30)
ax[1].imshow(img1)
ax[1].set_title('Style Image', fontsize = 30)
ax[2].imshow(result.cpu())
ax[2].set_title('Transferred Image', fontsize = 30)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Out[68]:

```
Text(0.5, 1.0, 'Transferred Image')
```