# 机器学习最终实验文档——风格迁移实验

**组内成员：**
组长：171250502 曹润泽
组员：171250033 朱令茹　171250614 赵航　161250188 余含章

**文档撰写人：**曹润泽

**任务简介：**

这次实验关于深度学习的风格迁移实验，风格迁移是将一张图片的画风变成另一张图的画风，因此就叫风格迁移。两张图片一个叫做 feature，一个叫做 model，即颜色等特征来自 feature，而线条轮廓特征来自 model。

**方法描述：**

算法采用的是基于优化的方法,即利用神经网络的多层次特性和 Gram 矩阵工具来优化特征。

方法参考：
https://tensorflow.google.cn/tutorials/generative/style_transfer#top_of_page

VGG 模型下载：
链接：https://pan.baidu.com/s/1zceRcQIe7fSNPwu7ifxL6Q
提取码：xyb1

方法简介：
其实很简单，我们首先要载入两张图片，代码中的 load_img 方法加载图片路径，showImage 展示最初的两张图片，即内容图和风格图。

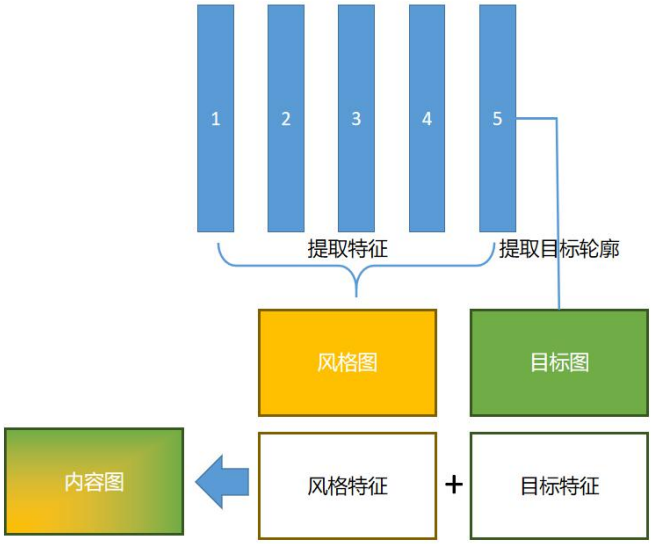
```
def load_img(img_path):
    #这个方法是为了加载图片，因为图片的较长边要被限制在512像素
    max_dim = 512
    image = tf.io.read_file(img_path)
    image = tf.image.decode_image(image,channels = 3)

    # decode_image将PNG编码的图像解码为uint8或uint16张量
    #     channels表示解码图像的期望数量的颜色通道.
    #     接受的值是:
    #     0: 使用PNG编码图像中的通道数量.
    #     1: 输出灰度图像.
    #     3: 输出RGB图像.
    #     4: 输出RGBA图像
    image = tf.image.convert_image_dtype(image, tf.float32)
    # 图片归一化，将image的每个像素的3个intRGB转化为32位浮点数
    # 图片最终应该是384*512*3，然后每一项都是32位浮点数
    shape = tf.cast(tf.shape(image)[:-1], tf.float32)
    # 获取图片尺寸的张量，先将图片的最后一维像素值去掉（变为384*512），再将图片尺寸（本来为int型整数）转化为浮点数（384*512变为384.0*512.0）
    long_dim = max(shape)
    new_shape = tf.cast(shape * max_dim / long_dim, tf.int32)
    image = tf.image.resize(image, new_shape)
    # 这一步将图片进行适当的缩小或放大，使得较长边变为512像素
    image = image[tf.newaxis, :]
    return image
```

Load_img 还包括对图片的一系列预处理，保证两张图片在相关参数上（像素，通道，尺寸）保持一致，否则会出现比较棘手的细节小问题。

ShowImage 则是简单的打印图片方便看到差别。

在获得了两张图片之后就可以对其进行特征提取而进行特征转移。

具体的思路是试用 VGG 神经网络模型对其进行分析,试用模型的中间层来获取图像的内容和风格。很明显，随着网络层数的增加，特征将会越来越接近图形的内容框架。也就是说，前几个层的特征较为低级，体现在边缘和纹理等低级特征，后面的层则代表高级的特征，也就是非常细节的轮廓。而我们的工作即：




如右图所示，即从 VGG 网络模型中间层提取风格图特征，从高级层提取内容图内容轮廓，再将这两者结合，成为我们最终需要的结果。

对于 VGG 网络获取，我们从 github 上进行下载，同时列出各层的名字，方便后面的各个层的处理。

```
#从github上下载vgg19网络，并列出各层的名字
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

print()
for layer in vgg.layers:
    print(layer.name)

# 选择用于表示内容的层，因为底层注重细节，高层注重整体，所以表示内容的话选择中间层最好
content_layers = ['block5_conv2']

# 选择用于表示风格的层
style_layers = ['block1_conv2',
                'block2_conv2',
                'block3_conv2',
                'block4_conv2',
                'block5_conv2']

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

至此我们得到了 VGG 网络的各个层，现在需要将这些层与我们的图片相结合，也就是说对图片进行目标提取。

```
203     '''构建提取器'''
204     extractor = StyleContentModel(style_layers, content_layers)
205     '''获取风格图片的风格层输出和内容图片的内容层输出'''
206     style_targets = extractor(style_image)['style']
207     content_targets = extractor(content_image)['content']
208     # 这两个直接提取出了风格和内容的目标值
```

如上图所示，我们构建了一个提取器来提取风格特征和内容特征，在这个方法中我们用到了 vgg_layers 方法（用于返回单个层，因为要集中对某些层进行处理）。在提取风格特征时使用 get_gram_matrix 方法（用于计算 Gram 矩阵）。对于 Gram 矩阵，事实证明，图像的风格可以通过不同的特征图上的平均值和相关性来描述，通过在每个位置计算特征向量的外积，并在所有位置对该外积进行平均，可以计算出包含此信息的 Gram 矩阵。具体的计算方法如下：

$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

具体的实现都在代码中。

现在我们得到了两张图片和 VGG 网络层的关系，我们需要对其进行特征传输，采用的是梯度下降优化方法。我们通过计算每个图像的输出和目标的均方差来做到这一点，然后取这些损失值的加权和。

```
210     image = tf.Variable(content_image)
211     # 这个是目标图像，首先让他和内容图像形状一样（Variable是用于初始化的函数）
212     opt = tf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
213     # 优化函数，这在train_step方法中被使用到
Transfer.py
215
216     style_weight = 1e-2
217     content_weight = 1e4
218     # 使用两个损失的加权组合来获得总损失，这在style_content_loss方法中被使用到
219     total_variation_weight = 1e8
220     # 上面是高频分量的损
221     showImage(image.read_value())
222
223     '''下面进行一段很长很长的优化'''
224     start = time.time()
225
226     epochs = 10
227     steps_per_epoch = 50
228
229     step = 0
230     for n in range(epochs):
231         for m in range(steps_per_epoch):
232             step += 1
233             train_step(image)
234             print(".", end='')
235         display.clear_output(wait=True)
236         showImage(image.read_value())
237         plt.title("Train step: {}".format(step))
238         plt.show()
239     end = time.time()
240     print("Total time: {:.1f}".format(end - start))
```

前面是对一些细节的处理，对一些参数的初始化，在 225 行之后是核心迭代代码。在这里我们进行了 10 个阶段，每个阶段 50 次迭代。在这里我们起初定义 style_content_loss 方法来计算损失，包括内容损失和样式损失。我们使用定义的的损失的加权组合来获得总损失。

最后在 train_step 中，使用 tf.GradientTape 来更新图像，此时的图像是已经被优化过的提取过特征值的内容图像。

这是最基本的操作，我们还需要减少因此操作产生的大量的高频误差。我们通过正则化图像的高频分量来减少这些高频误差：

```python
'''优化部分，将高频分量损失也算到损失函数中'''
def high_pass_x_y(image):
  x_var = image[:,:,1:,:] - image[:,:,:-1,:]
  y_var = image[:,1:,:,:] - image[:,:-1,:,:]
  return x_var, y_var
```

最后将该高频变量损失加入损失函数，再一并加入训练方法中：

```python
'''优化部分，将高频分量损失也算到损失函数中'''
def total_variation_loss(image):
  x_deltas, y_deltas = high_pass_x_y(image)
  return tf.reduce_mean(x_deltas**2) + tf.reduce_mean(y_deltas**2)

'''训练'''
@tf.function()
def train_step(image):
  with tf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)# 获得损失值
    loss += total_variation_weight * total_variation_loss(image)
```

至此所有的优化都已完成，接下来不断迭代 train_step，通过运行时间来反映指标。

**所使用的的数据：**

风格迁移没有大量数据的训练，也就不存在什么特点以及难点，我们主要是采用了 VGG 模型网络的框架，数据方面只有输入的两张图片。

**实验结果：**

我们的实验只设置了运行时间这一个评价指标，其他的方面很难得出一个指标来反映风格迁移的效果。

将其运行，结果如下：（这是我们一个组员的截图）

```
.....................................................
this epoch's time: 272 second which contains 60 steps

.....................................................
this epoch's time: 267 second which contains 60 steps

.....................................................
this epoch's time: 269 second which contains 60 steps

.....................................................
this epoch's time: 268 second which contains 60 steps

.....................................................
this epoch's time: 271 second which contains 60 steps

.....................................................
this epoch's time: 274 second which contains 60 steps

.....................................................
this epoch's time: 272 second which contains 60 steps
```

我们打印的是一个 epoch 的时间，平均算下来每一步需要花费 4.5 秒时间。

时间还算可以。

这是运行的情况，最终风格迁移的输出图存放在了“抽象海龟”和“浦东梵高”文件夹里，这里稍微分析一下情况：

这个海龟的结果说实话，我觉得效果还是不好，结果上的风格确实和风格图有几分相似，可是差距还是很大，这可能和我理解的风格大相庭径。可能海龟原图的线条划分较为明显，所以颜色的布局就比较杂乱。我觉得海龟头上那片海应该用纯色就可以了，虽然说风格图的颜色也很错乱，但是海龟的既然颜色比较单一那么结果的图也应该单一一下比较好。这可能是风格迁移的弊端，也是机器学习的难点。

相比而言，浦东梵高的结果显得比较好，

可能这张风格图的色彩比较好处理，与轮廓特征结合的比较好。

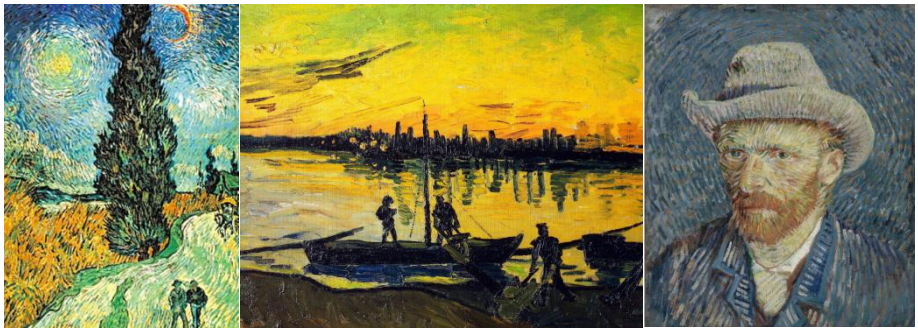
所以说，除了运行时间以外，我们就通过实际结果来判断效果好坏了。综上分析来看，感觉风格迁移这个工作的效果因人而异，因为风格是个比较抽象的东西，说不准。

**实验分析：**

本次实验采用基于优化的方法，因为基于优化比较简单，通俗易懂，操作起来容易。主要是因为我们对于机器学习了解颇潜，很多知识都是第一次听说，所以就采用最基本的方式了，至少还能够理解大概流程。

结果中也提到，这种方法通过提取中间层来获取风格特征，再将这种特征应用于高级层的内容轮廓上，大部分场景上是合适的。但是有一个问题，我们应用这种方法时，默认了将风格定义成“颜色”的特征，通俗的理解，就是将某些像素点替代，当然可能更复杂。

然而风格不仅仅是为颜色为基础的，这样对于梵高的画作是可以的，例如：



很明显，梵高成名就是因为他独特的色彩运用，所以说他的风格就是色彩方面，非常符合风格迁移的定义。

可是并不是所有的风格都是这样，可以看看毕加索的画作：

也很明显，毕加索的风格不只是在颜色上，也存在于线条的夸张运用。如果单单将毕加索的画与实物图结合，是看不出风格所在的。

为了证实这一点，我们单独做了两张图片的风格迁移：

这肯定不是我想得到的结果。我们只是提取了颜色的特征，而毕加索这幅画的灵魂却没有被提取出来。也就是线条之间的关系没有被体现出来。

因为我认为在风格提取这一块，不仅仅要将颜色特征提取出来，还要将线条的关系也展示出来。但是这又引入一个问题，因为梵高的画作就是颜色，他就不包含线条的特征。所以我觉得风格迁移的技术的选择得看具体是什么风格，不同的风格具有的特征也不一样。

这同时也是风格迁移所具有的的弊端，也就是没有统一的标准。