# 基于VGG16的图像画风迁移设计

# 背景问题和数据描述

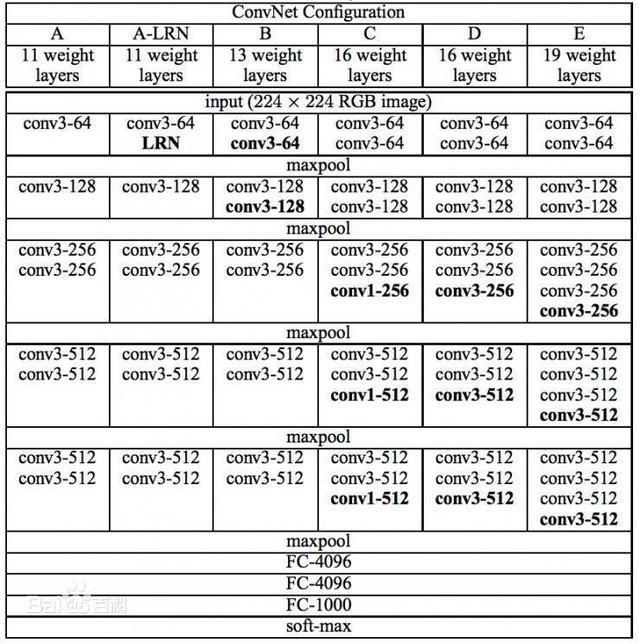
如何要把一个图像的风格变成另一种风格更是难以定义的问题。对于程序员，特别是对于机器学习方面的程序员来说，这种模糊的定义简直就是噩梦。到底怎么把一个说都说不清的东西变成一个可执行的程序，是困扰了很多图像风格迁移方面的研究者的问题。事实上在深度学习算法出现之前，对画风迁移方面的研究进展相当缓慢

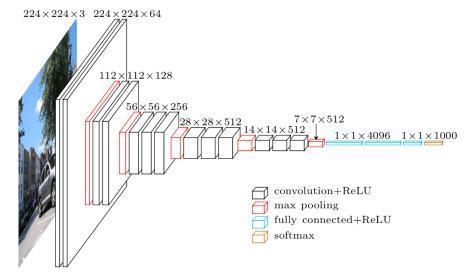
卷积神经网络当时最出名的一个物体识别网络之一叫做VGG16，每一层神经网络都会利用上一层的输出来进一步提取图像中更为复杂的特征，直至能够用来识别物体为止，VGG16在物体识别方面的精度甩了之前的算法一大截，因此之后的画风迁移方面的研究也基本都以VGG16为基础进行。

1. **解决思路和具体方案**
   1. **VGG16模型简介：**

VGG16模型是一种成熟的卷积神经网络模型，在这个网络中包含16个卷积层和全连接层。这是一个很大的网络，总共包含1.38亿个参数。但VGG-16结构并不复杂，这点非常优秀，而且这种网络结构很规整，都是几个卷积层后面跟着可以压缩图像大小的池化层，池化层缩小图像的高度和宽度。同时，卷积层的过滤器数量变化存在一定的规律，这种相对一致的网络结构对学习者很有吸引力，而它的主要缺点是需要训练的特征数量非常巨大。

**其各层结构如下：**





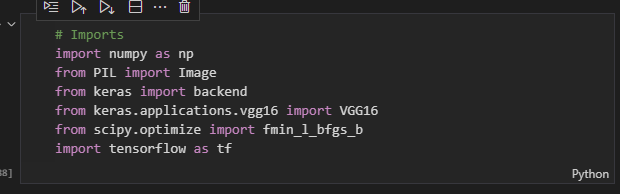
**2.2 程序流程图**

图示

描述已自动生成

1. **实现步骤和代码**
2. 导入必要的包，图像的存储上这里选择使用的是PTL用于存储图像的长宽和RGB数值

此处实现画风迁移的基础包体为TensorFlow，核心算法为VGG16，前面的章节已详细阐述此算法的原理



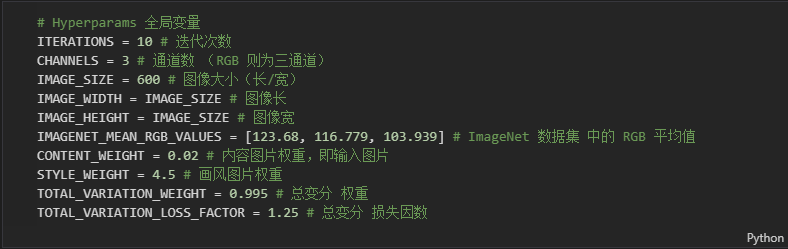
1. 定义需要用到的全局变量

ITERATIONS：考虑到我们三人设备的性能受限，此处仅迭代10次

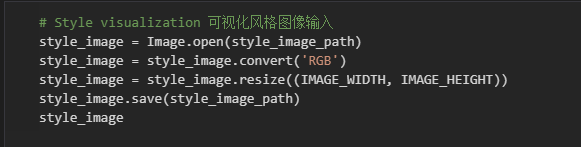
CANNELS：通道数，对于全真彩色图像一般含有RGB三个通道，若图像包含不透明度（A）通道则需修改为4

IMAGE\_MEAN\_RGB\_VALUES：定义了给定数据值中rgb三通道各自的平均值

CONTENT\_WEIGHT和STYLE\_WEIGHT定义了内容图片和画风图片的权重

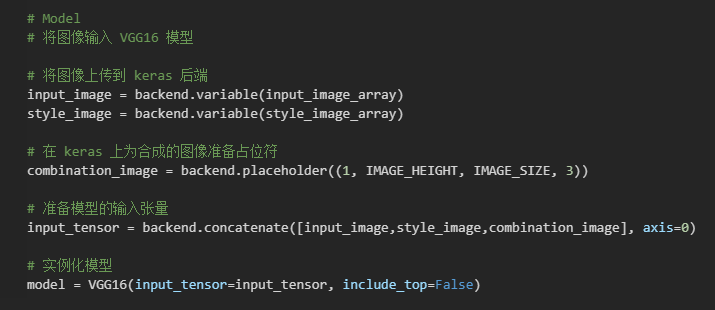


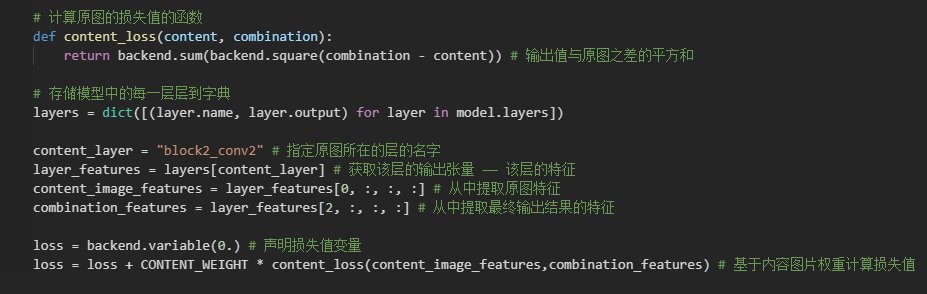
* 1. 输入内容图片，画风图片和导出图片的路径，并创建一个预览窗口可视化输入数据



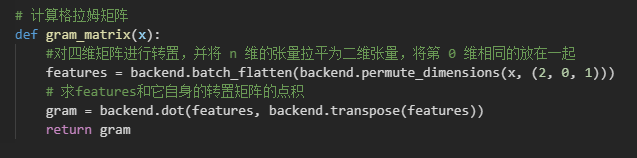
* 1. 由于PIL中对图像数据的存储是按照[B,G,R]的格式进行存储的，因此我们需要将先前的数据进行Normalize标准化处理，并将[R,G,B]统一转化为[B,G,R]格式

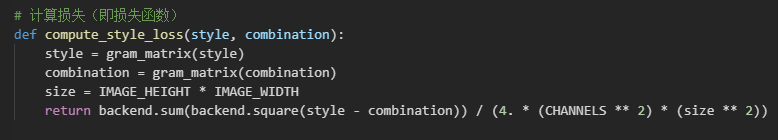


4. 将标准化之后的数据输入VGG16模型，上传至keras后端进行一系列准备，最后实例化VGG16模型

5. 计算原图损失值函数，提取原图像中每一层当中的图片特征，最后基于全局变量中赋予的内容图片的权重值计算损失值

6. 计算格拉姆矩阵和损失函数

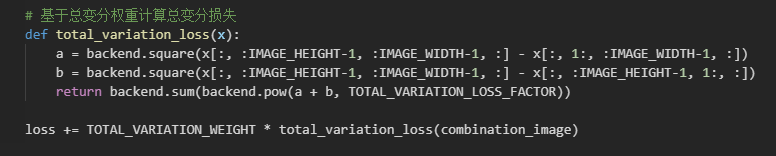


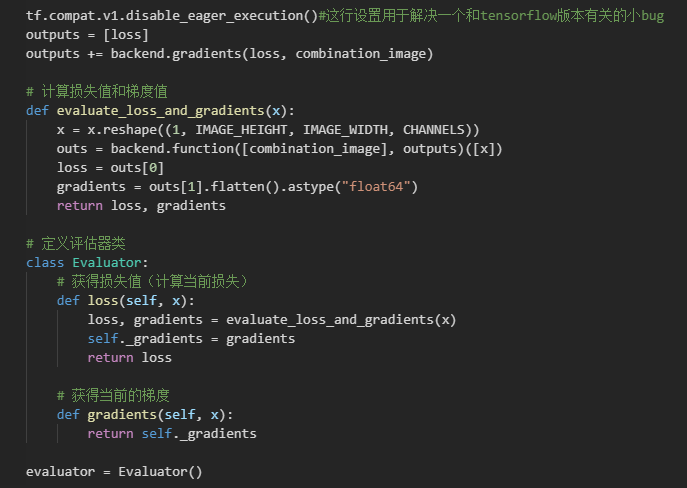


7. 声明画风迁移的层并循环输入每一层的参数，基于之前定义的全局变量权重计算每一层的损失值

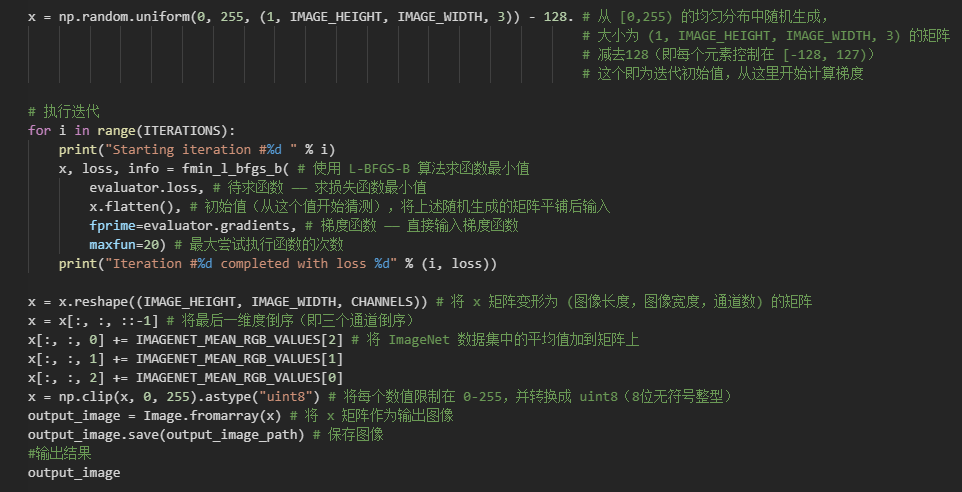


8. 计算基于总变分权重计算总的变分损失



9.定义计算损失值和梯度值的函数，并创建一个评估器class用于获得当前的损失值和梯度值 

10. 定义一个从[-128, 127)的均匀分布中随机生成的大小为(1, IMAGE\_HEIGHT, IMAGE\_WIDTH, 3)的矩阵作为seed，并根据计算出的梯度逐步计算迭代，梯度逐步下降的同时满足损失值最小化，生成最终的图像并保存

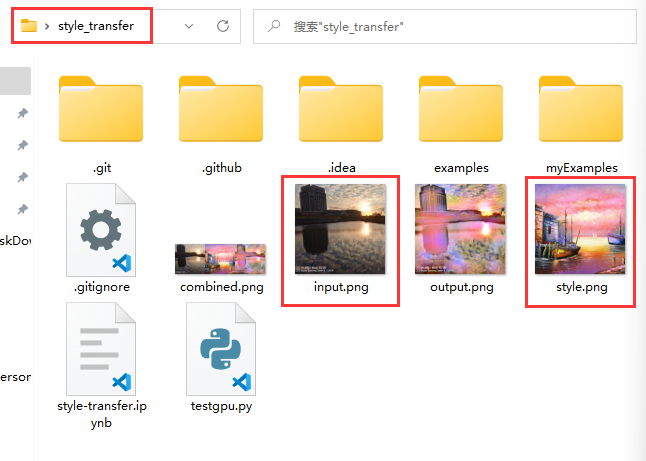


11. 将内容图片，风格图片和最终成果进行拼合，窗口可视化展示画风迁移的结果

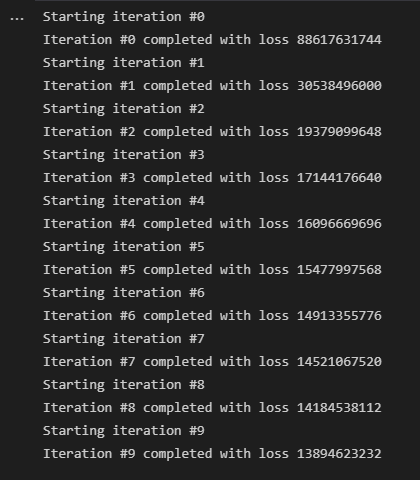


1. **处理结果和分析**

准备好对应的内容图片和风格图片，放置于项目根目录下并命名为input.png和style.png。倘若需要处理的内容图片和风格图片不位于项目根目录下，则参数中对应图片名称应输入图片所在的绝对路径。



输入参数后，命令行会持续输出每一次迭代的层数和对应的损失值：



迭代时巨大的GPU压力：

图片包含 图形用户界面

描述已自动生成

当执行至全局参数所设定的最大迭代次数后，输出命名为output.png的画风迁移后的图片，并窗口展示内容图片，风格图片和最终成果：



至此画风迁移算法执行完毕，做画风迁移对设备的GPU性能还是有较高要求的，如果使用CPU硬算的话，整套算法的运行时间会长上不少。

**下面附上我们小组图像迁移后的一些完成品：**

最初我们使用现实照片作为内容图片，但画风迁移之后效果不是很理想：水中的倒影

中度可信度描述已自动生成

后面我们发现这套模型对于线稿/素描一类的内容图片有着更好的识别效果，对原先图片的特征也能得到很好的保留，因此后面我们做的一些画风迁移的方向转为了对原先线稿/素描图片的风格化上色处理，最终成果也确实能够很好的提取风格图片当中的颜色特征来对内容图片进行上色：

卡通人物

低可信度描述已自动生成

下面的这两组迁移结果证明了，对风格图片的选取上应尽可能只选取特征信息丰富的部分，会得到更好的迁移结果，模型对颜色特征值的提取也能够更少的受到风格图片中大片留白的干扰：

卡通人物

中度可信度描述已自动生成

卡通人物

描述已自动生成

下面这组图证实了，选用色彩上色相较为接近的（同一色或近似色）风格图片，会比使用色相差异较大（对比色或三原色，例如红配绿，红配蓝等，绿配蓝等对比较大的配色）的风格图片，能得到更好的画风迁移效果：

卡通人物

中度可信度描述已自动生成

卡通人物

低可信度描述已自动生成

# 答辩问答环节

Q：具体怎么工作的？

A：通过将格拉姆矩阵（gram matrix）应用于卷积神经网络各层能够捕获该层的样式，所以，如果从填充了随机噪声的图像开始，对其进行优化使得网络各层的格拉姆矩阵与目标图像的格拉姆矩阵相匹配，那么不难理解，生成的图像将会模仿目标图像的风格。

可以定义一个style损失函数，计算两组激活输出值各自减去格拉姆矩阵之后计算平方误差。

在原始图像和风格图像上进行训练，将两幅图片输入VGG16的卷积神经网络，对每个卷积层计算上述style损失并多层累加，对损失使用lbfgs进行优化（它需要梯度值和损失值进行优化）。

Q：为什么不直接匹配所有层的激活值？而使用格拉姆矩阵？

A：因为使用格拉姆矩阵，相当于通过计算给定层的每个激活值与其他激活值的乘积，这样我们获得了神经元之间的相关性。这些相关性可以理解为图像风格的编码，因为他们衡量了激活值的分布情况，而不是激活值本身（风格本身）。

P.S. 这也带来了几个问题。一个就是零值问题，在任一被乘数为零的情况下，一个向量和自己的转置的点积都会是零，模型无法在零值处识别相关性。由于零值频繁出现，可以在执行点积前为特征值添加一个小的差量delta，delta取-1即可。还有一个问题就是，我们计算了所有激活值的格拉姆矩阵，难道不是应该针对像素通道计算吗？事实上，我们为每个像素的通道都计算了格拉姆矩阵，然后观察它们在整个图像上的相关性，这样做提供了一个捷径：可以计算通道均值并将其用作格拉姆矩阵，这会帮助获得一幅平均风格的图像，更有普适性。

Q: 将总变分损失（total variation loss）纳入的意义？

A：总变分损失要求网络时刻检查相邻像素的差异，否则，图像会趋于像素化且更加不平缓。某种程度上，这种方法与用于持续检查层权重与层输出的正则化过程非常类似，整体效果相当于在输出像素上添加了一个略微模糊的滤镜。（本质上，这是一种模糊化的方法）该部分的结果是将最后一个组成成分添加到了损失函数中，使得图像整体更像内容图像而不是风格图像。这里所做的，就是有效优化生成图像，使得上层的激活值对应图像内容，下层的激活值对应图像风格，也就是网络底层对应图像风格，网络高层对应图像内容，通过这种方式实现图像风格转换。（实际上这部分并没有做得很好，因此最终我们还是将这个模型更多用于线稿上色）

# 个人工作总结

在制作本次大作业之前，之前因为自己的个人需求，需要对图片当中的色彩进行简化处理，因而有接触过OpenCV方面的基础知识和python当中有关图像数据的存储方法，也了解过近似色替换方面的一些基础算法知识，能够将原图逐像素近似替换为给定色彩列表中的颜色，但受限于个人知识一直无法做到进一步的优化。

这一次学习人工智能课程，尤其是了解到一些关于画风迁移方面的知识之后，在B同学的帮助下成功实现了基于给定风格图片的图像近似色替换，虽然我们的模型在面对现实摄影照片时的效果依旧不够完善，但意外地在绘画线稿与素描作品上取得了非常棒的迁移效果，也算是本次大作业制作过程中的意外惊喜。最后感谢B同学的鼎力相助。