摘 要

在机器学习领域，人类已经教会了机器自动过滤垃圾邮件，下棋、甚至驾驶无人汽车。但是在绘画领域，尤其是艺术创作，计算机还不能很好地模拟人类的艺术行为。人们一直在试图寻找让计算机自动生成艺术图像的方法。

本文基于一种叫做深层神经网络的机器学习算法，设计并实现了一个图像风格转换系统，该系统能够把一张普普通通的图片转换为优美的艺术风格作品。深层神经网络基于对大脑的研究成果，其原理是用计算机实现一个庞大的神经网络，以模仿大脑神经元的运作机制。这种网络算法可以很好地提取出图像的内容特征和风格特征，再进行一些转换运算，便可以转移艺术作品的风格。

本文通过多次的实验测试发现，基于“快速图像风格化算法”实现的图像风格转换系统，对艺术图像的纹理有着良好的学习能力。但是，这种算法在风格特征不明显的图片上表现一般，而且该算法需要大量的计算资源来训练网络模型，实现成本较高。

关键词：神经网络，图片风格，图像转换系统

ABSTRACT

In the field of machine learning, humans have taught the machine to automatically filter spam, play chess, and even drive unmanned cars. But in the field of painting, especially the creation of art, the computer is not a good simulation of human art. People have been trying to find a way for computers to automatically generate art images.

Based on a machine learning algorithm called deep neural network, this paper designs and implements an image style conversion system, which can convert an ordinary picture into a beautiful art style. Deep neural network based on the research results of the brain, the principle is to use a computer to achieve a huge neural network to imitate the operation of the brain neurons mechanism. This network algorithm can be a good way to extract the image of the content characteristics and style features, and then some conversion operations, they can transfer the style of art works.

In this paper, through several experimental tests found that based on "fast image style algorithm" to achieve the image style conversion system, the texture of art images have a good learning ability. However, this algorithm in the style of the image is not obvious on the performance of the general, and the algorithm requires a lot of computing resources to train the network model to achieve higher costs.

KEY WORDS：Neural network, Picture Style, Image conversion system

目 录

第一章 绪论

1.1 研究背景

在机器学习领域，人类已经教会了机器自动过滤垃圾邮件，下象棋，甚至驾驶无人汽车。但是对于艺术创作，尤其是绘画领域，计算机尚且不能很好的模拟人类的这种艺术行为。最近，在物体和人脸识别领域，有一类称为深层卷积神经网络的模型席卷而来。科研人员基于该模型发明了一个有趣的学习算法[1]，该算法使用神经网络来重组图像的内容和风格。

卷积神经网络是一种最有效的处理图像数据的深层网络。卷积神经网络可分为卷积层，池化层，全连接层等“层”结构。每个“层”由一些图像过滤器组成，其中每个图像过滤器可从图像中提取相应的特征。当卷积神经网络被用于“物体识别”方面时，它提取的特征信息，能清晰展现出图像的层次结构。

具有“物体识别”功能的网络，其较低“层”提取的特征信息表示了图片的像素风格，而较高“层”的特征信息表示了图像的抽象内容。若把原图和风格图分别输入的卷积神经网络，便可提取出“原图的内容特征”和“风格图像的风格特征”，以此推导出一个代价函数。最后使用梯度下降算法优化代价函数，就可以得到结合了原图和风格图的艺术图。

1.2 研究目的和意义

人们一直在试图寻找让计算机自动生成艺术图像的方法。本文基于深层卷积神经网络的算法，实现了一个图像风格转换系统，它能够把一张普普通通的图片转换为优美的艺术风格作品。同时，本文设计了“快速图片风格化算法”在云平台下的系统架构方案，以提高系统的负载能力。实现代码见[22]。

第二章 相关技术概述

2.1 人工神经网络

人工神经网络属于机器学习[2]的一个分支。人工神经网络模拟了人类的大脑构造，是一类十分强大的算法，常被用于解决分类和回归问题。这里本文将介绍一种叫做“感知器”人工神经元。

感知器是如何工作的呢？一个感知器接受几个输入，并且通过一定的运算产生相应的输出，如图 2-1 所示。

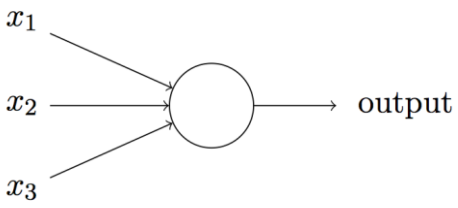

图 2-1 感知器

图 2-1 中的感知器有三个输入，x1，x2，x3。通常可以有更多的输入，输出的计算规则如下：

$$output = \begin{cases} 0 & if \sum_{i=1}^{n} w_i \cdot x_i + b_i \leq 0 \\ 1 & if \sum_{i=1}^{n} w_i \cdot x_i + b_i > 0 \end{cases}$$

其中每个输入都有一个对应的权重 wi 和偏置 bi，对所有的输入进行计算累加，由计算结果所在值的区间得到最终结果。除了最普通的感知器，还有一种经过改进以的 S 型神经元，它的输出计算规则为：

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b_i$$

$$output = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$\sigma(z)$ 的代数曲线见图 2-2，它也被称作 sigmoid 激活函数：

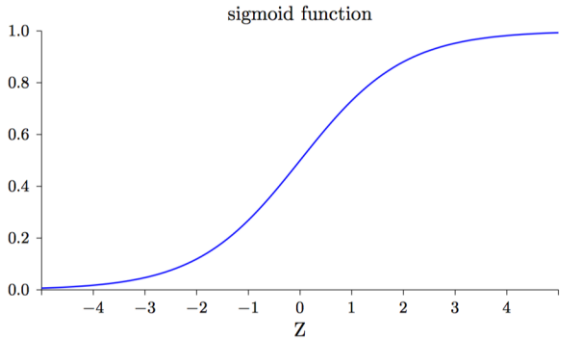

图 2-2 Sigmoid 曲线

S 型神经元改进了感知器输出的平滑性。对于感知器，其输出是一个值域为 0 和 1 的阶跃函数。很明显，S 型神经元可以输出 0 到 1 之间的任何数字，这是非常有用的。这样，S 型神经元的输出便可以表达更多的输出状态，更好的模拟人脑的行为。

尽管如此，只有一个简单的 S 型神经元并不能像人类一样做出复杂的逻辑判断，但用这些“神经元”就能连接成一个复杂的神经网络，并且分成不同的层次：第一列神经元，第二列神经元，以这种方式构成的多层的神经网络便可以进行复杂的决策。其中第二列神经元一般称为隐藏层神经元，而这种连接方式构造的网络又称为全连接网络，如图 2-3 所示。

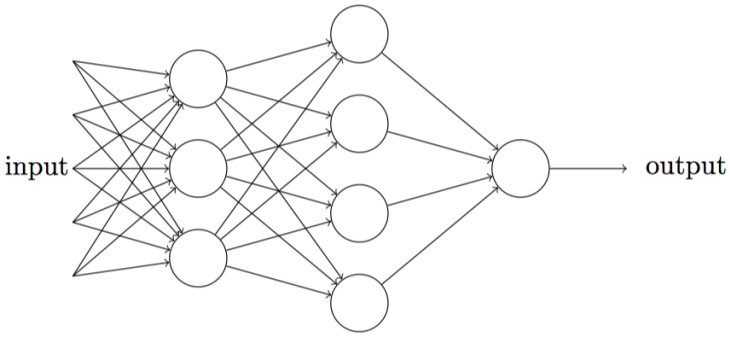

图 2-3 全连接神经网络

有了全连接神经网络，我们还需要确定网络的参数，即的权重 w 和偏置 b。为了量化网络的拟合效果，我们会定义一个误差函数，比如下面这个 MSE 均方误差：

$$C(w,b) \equiv \frac{1}{2n}\sum_{x}\|y(x)-a\|^2$$

其中 y(x) 是网络的输出结果，而 a 是测试数据给出的正确结果，他们之间差的平方和的平均值就是均方误差。我们训练网络的目标即是确定神经元的权重 wi 和偏置 bi，使代价函数最小。

可以使用“梯度下降”的方法来寻找这个最小值，假设 C(w，b)是一个简单的二元函数（即只有一个神经元，正好两个参数 w1，b1），则 z = C（w1，b1）

的三维示意图见图 2-4：

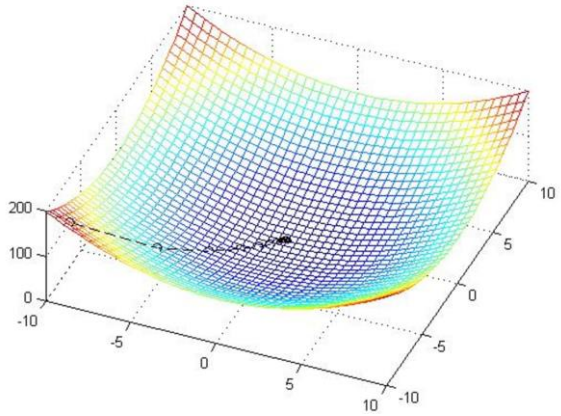

图 2-4 梯度下降过程

通过凸优化的数学知识可以知道：在每一轮训练中，使参数 w1 和 b1 分别减去其偏导数的一定倍数，w1 和 b1 所指向的点会慢慢靠近图 2-4 所示“盆地”的中心点，即代价函数 C(w1，b1)最小的点，这就是梯度下降算法。其数学表达式如下：

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}$$

每迭代一个训练样本，就计算一次网络参数的偏导数，并修正这些参数。经过若干次的迭代修正，就可以获得处于最佳状态的神经网络。在实际的应用中，一般使用反向传播算法[4]来求解偏导，这是因为对于计算机来说，求复杂表达式的偏导是一件很麻烦的事情，而反向传播算法正好解决计算难的问题。

通常我们不会把所有的数据输入进去计算，而是一组一组地输入，这就是 Mini-batch-SGD[3]，或者干脆一次只输入一个数据，即 One-example-SGD，这样做的目的是为了减少计算的复杂度。

2.2 图像的卷积

二维卷积公式是实现图像卷积基本数学原理，如下所示：

$$y(p,q) = \sum_{i=0}^{M} \sum_{j=0}^{N} h(p-i, q-j) u(i,j)$$

假设 u 为 3*3 的卷积核，图像卷积运算的基本步骤为：

1) 卷积核心绕中心旋转 180 度
2) 一边滑动卷积核，一边根据二维卷积公式求和，如图 2-5 所示

3) 将求和结果填入输出矩阵。

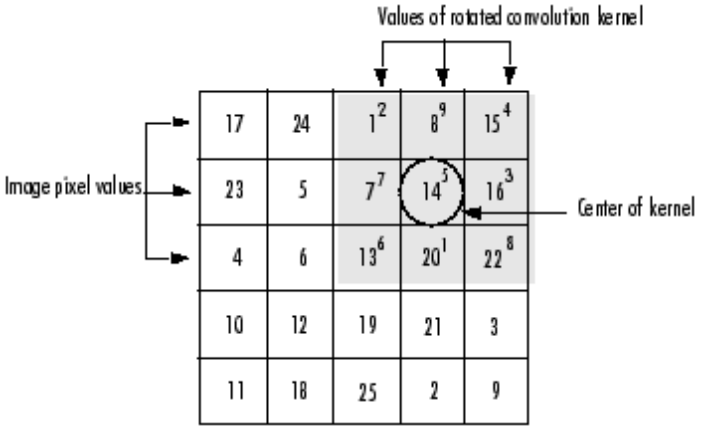

图 2-5 图像的卷积

此外，图像的卷积还可以通过傅里叶变换在频率实现[5]。

不难发现，图像经过卷积运算后边长会减小[卷积核大小 / 2]（设步幅为 1），因此我们一般会对图像边缘进行填充。卷积核滑动的步幅是可以变更的，一般为每次运算滑动 1 格，即步幅为 1。此外，在卷积神经网络中，我们通常忽略卷积核旋转的这个过程。

图像有卷积运算，不难想象也会有相应反卷积运算。图像的卷积运算可以转换为相应的矩阵运算，即 y = C * x，其中 C 是一个特定的稀疏矩阵，x 为输入图像，同时可以得到 $C^T * y = x$，这就是图像的反卷积。本文在“快速艺术风格算法”一章会用“反卷积”实现上采样。

2.3 卷积神经网络

卷积网络也叫卷积神经网络，这种网络结构是 Wiesel 和 Hubel 在研究猫脑皮层中的神经元时发现的，它可以明显的降低深层神经网络的复杂性，所以被广泛运用于计算机视觉领域。“卷积”一词指的是该网络使用了卷积这种数学运算。卷积是一种特殊的线性运算。卷积网络是指那些至少在网络的一层中使用卷积运算来替代一般的矩阵乘法运算的神经网络

卷积神经网络主要可分为卷积层，池化层和全连接层。本文 2.1 节中介绍的人工神经网络只有全连接层，卷积神经网络在其基础上增加了卷积层和池化层。如图 2-6 所示。

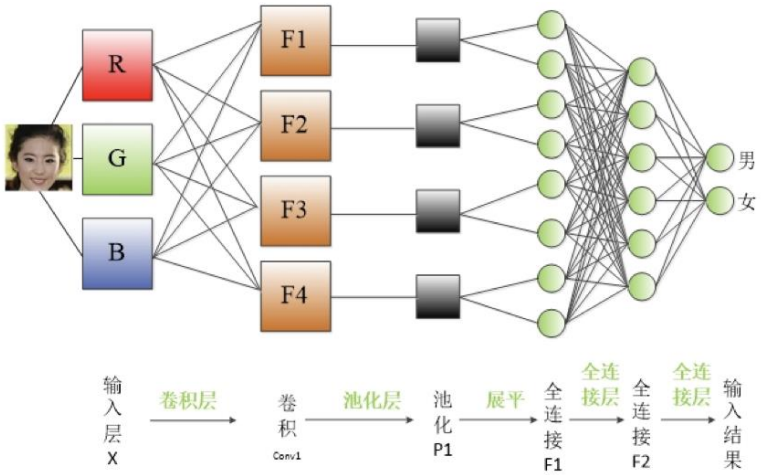

图 2-6 卷积神经网络结构

卷积神经网络中有一个非常重要的概念叫特征图，图像的每个通道就是一副特征图，比如（R，G，B）三个通道。通过卷积层可以把这些通道映射成更多的通道，如图 2-6，（R，G，B）三个特征图通过卷积映射成了四个特征图(F1，F2，F3，F4)。

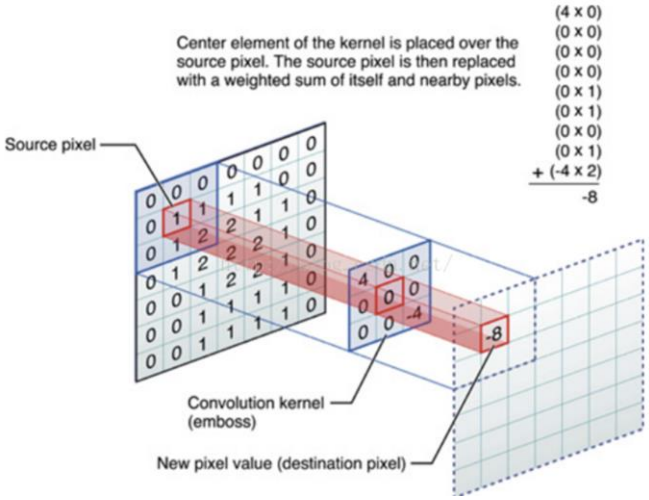

图 2-7 图像卷积映射

图 2-7 即图像的卷积映射。由（R，G，B）三个特征图映射为(F1，F2，F3，F4)四个特征图共需要 3*4 个卷积核，其中的计算规则如下：

$$F_i = W_{i1} * R + W_{i2} * G + W_{i3} * B$$

Wi_1，Wi_2，Wi_3 即 Fi 依赖的三个卷积核，实际上在计算出 Fi 后，还要再加上相应的偏置 bi，并且通过激活函数的转换，才得到卷积的结果。所以卷积层 = 激活函数层(卷积运算 + 偏置)。

池化层和卷积层十分类似，只不过卷积核的功能由“进行像素的加权求和”

运算替换为“选择当前过滤窗口中的最大或者最小的元素”。最大池化示例的见图 2-8。

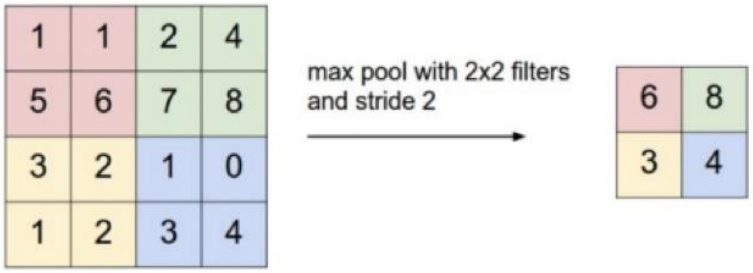

图 2-8 最大池化

卷积神经网络通过卷积层，池化层减少了深层网络中的参数个数。一副 30*30 的 RBG 图像，若由全连接的方式构造神经网络，一共需要 30*30*1000（假设隐藏层有 1000 个神经元） = 900000 个 w 参数。而通过卷积层的方式只需要（假设生成四个特征图，卷积核的大小为 3*3）：3*3*3*4 = 108 个卷积核参数。通过局部感受视野和共享权值得原理，网络的复杂性大大降低，从而优化了神经网络网络的性能。训练全连接的神经网络和卷积神经网络不同之处在于：卷积神经网络被训练的参数是卷积核和偏置 b，而全连接网络训练的是权重 w 和偏置 b。这可以理解为，网络找到了最佳的图像过滤器，提取出抽象信息，简化了全连接层进行图像分类的复杂性。

2.4 VGGNet

VGGNet[8]是一系列非常优秀的用于物体识别深层卷积神经网络，曾经在 ILSVRC-2014 达到了 best-performance。

它由牛津大学的视觉几何小组提出，是 ILSVRC 2014 中分类任务的第二名，其主要内容是证明了使用小卷积核增加网络的深度，可以提高深层神经网络的性能。作者配置了多种不同深度的 VGGNet 进行图片识别效果测试，从而得出这个结论。虽然有些暴力求解的因素，但是这个实验确实是有效的。现在很多预训练（即已经训练好参数的网络）的方法就是基于 VGG 模型，VGG 模型相对其他的网络模型，网络结构比较复杂，导致最终的模型文件有 500 多兆。通常训练一个 VGG 模型要花很多时间和计算资源，还好，网上有很多公开的模型供我们使用，本章节中的“艺术风格的神经算法”就使用了训练好的 VGG19 模型。

VGG19 拥有 19 个 weight layer（最大池化层没有权重参数 weight，所以不是 weight layer），它是 VGGNet 模型中最深的一个，其结构示意见图 2-9：

input (224*224 RGB image)
con3-64(conv1_1) con3-64(conv1_2)
maxpool
con3-128(conv2_1) con3-128(conv2_2)
maxpool
con3-256(conv3_1) con3-256(conv3_2) con3-256(conv3_3) con3-256(conv3_4)
maxpool
con3-512(conv4_1) con3-512(conv4_2) con3-512(conv4_3) con3-512(conv4_4)
maxpool
con3-512(conv5_1) con3-512(conv5_2) con3-512(conv5_3) con3-512(conv5_4)
maxpool
FC-4096
FC-4096
FC-1000
soft-max

图 2-9 VGG19 网络结构

conA-B 表示卷积层，其卷积核的大小为 A*A，输出 B 个特征图。convA_B 表示的是卷积层的编号，FC-A 表示全连接层及其连接的神经元数量为 A。

VGG 网络的输入是 224*224 像素的 RGB 图像，并且采用了 3*3 大小的卷积核（VGG 作者提到这是最小的可以同时采集上下左右周围像素的卷积核了）。这么做的原因是为了减小参数的数量，在加深网络的同时，不至于使网络太过复杂和难以训练。对于 VGG 网络来说，输入的图像必须经过一次预处理，其方法是

减去给定的图片均值像素。

VGG 网络的卷积层连接的是 ReLu 激活函数[6]，该激活函数改进了网络的训练性能。下面给出 ReLu 和 Softmax 的计算公式：

$$ReLu(z) = max(0, z)$$

对于之前提到的 Sigmoid 函数来说，当函数的输入值非常大或者非常小的时候，就会发生梯度消失的问题。这会导致代价函数的偏导数接近 0，所以参数难以得到有效的修正。ReLu 激活函数改善了这个问题，因为它的梯度是一直存在的，如图 2-10 所示。

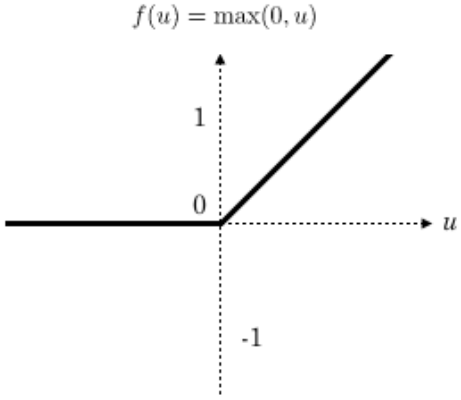

图 2-10 ReLu 激活函数

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

VGG 网络的最后一层使用 Softmax 激活函数将输出转换为表示可能性的概率。假设 VGG 网络的输出为：[0.01，0.05，0.03……**0.5**…0.1]（向量长度为 1000，其中 0.5 为最大概率），则网络识别的结果为 0.5 在向量中的序号，通过该序号就能对应到其表示的事物。

上面公式中的 Zi 表示最后一层网络第 i 个神经元的输出，n 表示最后一层网络的神经元数量。Softmax 函数的本质是进行一次权重值到概率值的转换。

2.5 深度残差网络

在“快速图像风格化算法”中，我们用名为“深度残差网络”[11]来作为“图片转换网络”，这是一种更容易训练的深层神经网络结构。其有效降低了网络的训练难度。

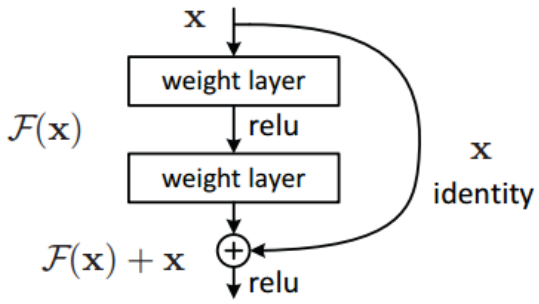

图 2-11 深度残差网络的基本结构

图 2-11 所示为“深度残差网络”的基本结构，其中第二层卷积的输出为 F(x) + x。这里直接把输入 x 拷贝了一份加入第二层的输出中。

下面解释一下这样做的原因：因为反向传播算法中链式求导的缘故，在深层神经网络中，网络参数的偏导会“逐渐消失”。反向传播算法的基本原理是：先用前馈的方式得到网络的输出值 y，然后根据 y 的误差反向一层一层求取浅层网络的误差，根据指定的层误差就可以得到该层的偏导数。假设输出 y 的误差为 0.5，反向传播的过程就好比：0.5*0.3*0.3*0.3…，当到达前几层网络的时候，误差几乎变为 0，于是梯度消失了，网络无法修正。试着计算“深度残差网络”结构的偏导数，如下：

$$\frac{\partial X_L}{\partial X_l} = \frac{\partial X_l + F(X_l, W_l, b_l)}{\partial X_l} = 1 + \frac{\partial F(X_L, W_L, b_L)}{\partial X_L}$$

不难发现，因为导数值 1 的存在，就算网络很深，梯度也不会消失。

第三章 快速图像风格化算法原理

3.1 内容和风格的代价函数

以目前的技术水平，对于任意给定的图片，计算机可以很好地识别出图像的内容。可是图像风格是一类非常抽象的事物，在计算机系统看来，只不过是一些二进制像素，但人眼却可以很轻易地分辨出不同的艺术风格。

因为深层神经网络的实质就是找出更复杂，更深层的信息。所以可以用深层卷积神经网络来提取图像的风格。根据 Gatys 等人提出的算法[1]，训练好的 VGG 网络可以提取出图像的风格特征。

研究人员将图像输入 VGG 网络，然后用 VGG 网络提取的图像特征信息来重构原图像。他们发现利用较底层网络重构的图片几乎完美，而在网络的较高层，详细的像素信息已经丢失，只保留了图片的轮廓信息。这可以认为高层次提取的特征只包含图像的内容信息，而低层次提取的特征更好地保留了图片的像素风格。所以可以认为 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 为风格提取层，而 conv4_2 为内容提取层。

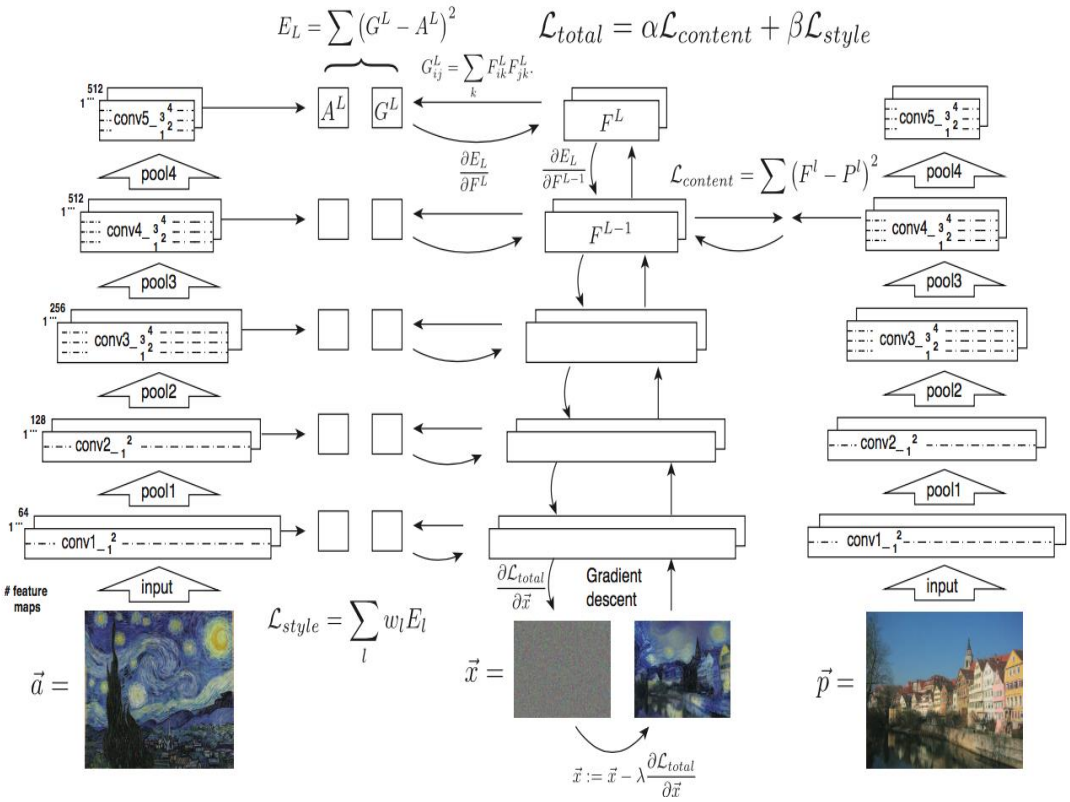

图 3-1 内容和风格代价函数的计算

可由内容提取层的特征信息可以构造内容代价函数，如图 3-1 所示。

$$\mathcal{L}_{content}(\vec{p},\vec{x},l)=\frac{1}{2}\sum_{i,j}\left(F_{ij}^{l}-P_{ij}^{l}\right)^{2}.$$

$$\frac{\partial\mathcal{L}_{content}}{\partial F_{ij}^{l}}=\begin{cases}\left(F^{l}-P^{l}\right)_{ij} & \text{if } F_{ij}^{l}>0\\ 0 & \text{if } F_{ij}^{l}<0.\end{cases}$$

F_{ij}^{l} 中的 l 代表 VGG 网络的 conv4_2 层，F_{ij} 是生成图在该卷积层的输出值，P_{ij} 是原图在该层的输出值。第二个公式是内容代价函数的偏导，可以用反向传播算法求出该偏导数。

$$G_{ij}^{l}=\sum_{k}F_{ik}^{l}F_{jk}^{l}.$$

该算法不使用像素值来描述风格，而是用 gram 矩阵表示图片的风格特征。$G_{ij}(l)$是被向量化第 l 层特征通道 i 和 j 的内积。这可以理解为，对图片 i 通道的像素和 j 通道的像素求内积，放大了它们之间的共同特性，所谓的共同特性即像素的风格。

$$E_{l}=\frac{1}{4N_{l}^{2}M_{l}^{2}}\sum_{i,j}\left(G_{ij}^{l}-A_{ij}^{l}\right)^{2}$$

$$\mathcal{L}_{style}(\vec{a},\vec{x})=\sum_{l=0}^{L}w_{l}E_{l}$$

E_l 表示第 l 层的风格代价，其中 A_l 是风格图在第 l 层的 gram 矩阵，G_l 是生成图在第 l 层的矩阵。N，M 表示相应的缩放系数。每一层的代价函数累加就得到了总体的风格代价函数。

风格代价函数和内容代价函数以一定比例相加，得到的整体代价函数。

$$\mathcal{L}_{total}(\vec{p},\vec{a},\vec{x})=\alpha\mathcal{L}_{content}(\vec{p},\vec{x})+\beta\mathcal{L}_{style}(\vec{a},\vec{x})$$

得到整体代价函数后，只要用“梯度下降算法”不断优化代价 L_{total}，便能使生成图 x 在轮廓上越来越接近原图 p，同时又在像素上逼近风格图 a。遗憾的是，这个算法的运行速度不是十分理想。实验表明，一张 500*500 像素的图片，在 Ubuntu 16.04 + Tesla K40 平台上进行风格迁移训练，需要近一个小时的时间。

3.2 生成对抗网络

3.1 节中提到的算法依赖于对生成图 x 的迭代修正，因此算法的效率十分低下。Johnson 等科研人员发现[9]，可以用一个前馈网络代替这个过程，这就是快速图像艺术风格化算法[9]的原理。

这是通过“生成对抗网络”[10]实现的，所谓的生成对抗是让两个不同的网络相互对抗。他们中的一个叫做生成网络，它是优化算法训练的主体，主要功能是生成结果。而另一个叫判别器网络，它可以监督生成器网络的生成结果，并给予一个评价，这个评价可以用作代价函数。

该算法用一个“Image Transform Net”（图像转换网络）作为生成器网络，它的作用是把输入图像转换为特定的风格，而用 2.4 节提到的 VGGNet 作为判别器网络。其结构如图 3-2 所示。

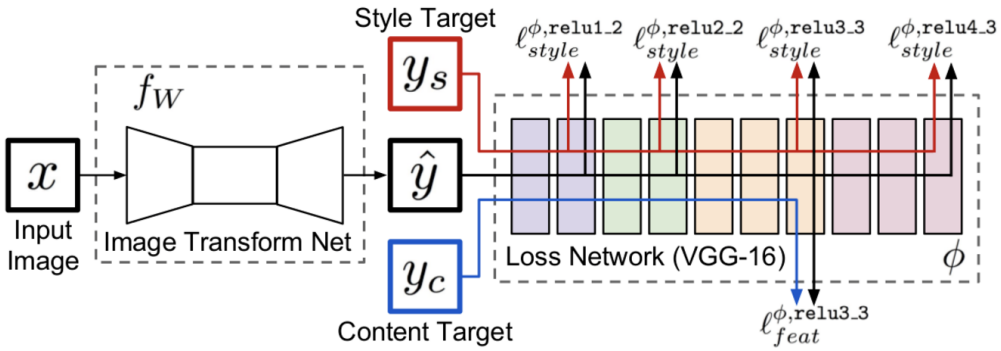

图 3-2 快速图片风格化算法原理

图像转网络的训练流程为：

1) 图像 x 通过生成器网络生成输出图像 $y^{\wedge}$
2) 生成图 $y^{\wedge}$、风格图 y_s 和内容图 y_c 分别进入判别器网络，VGGNet 提取其相应的特征信息计算出“代价”
3) 根据代价修正图像转换网络。

“代价”的可以理解为判别器网络给生成器打的分数，它被用来评价“生成器网络”的转换效果的好坏。所以可以这可以通过修正“图像转换网络”实现，使得 VGGNet 对“图像转换网络”的评价越来越好。经过若干次迭代，一个“图像转换网络”就训练好了。对于一个训练好的“图像转换网络”来说，它已经记住了艺术家的绘画方式，并能够快速地将任意图片转换为固有的风格。

此外，快速艺术风格算法还增加了一个全变差代价，用于优化图片的平滑性。于是，新的代价函数 = （生成图的风格特征 – 风格图的风格特征） + （生成图的内容特征 – 原图的内容特征）+ （生成图整体的全变差）。

3.3 图像转换网络

input (RGB image)
//下采样 con9-32(stride 1) con3-64(stride 2) con3-128(stride 2)
//残差卷积层 res3-128 res3-128 res3-128 res3-128 res3-128
//反卷积 decon3-64(stride 2) decon3-32(stride 2) con9-3(stride 1)
tanh

图 3-3 图像转换网络接结构

图像转换网络的结构参考了论文[12]，它利用了两个步长为 2 的卷积去下采样，接着连接了几个残差层，然后是两个相对应的上采样卷积层。这样的上采样，下采样过程会带来一些额外的好处，如图 3-3 所示。

首先，图像经过下采样之后，会让我们的卷积运算次数变少。于是，我们可以计算更大的网络而不用更换硬件设备。这就充分利用了计算机的性能资源。第二个好处是通过下采样，有效感受视野的大小变大了。

所有的非残差卷积层都跟着一个 Batch-Normalization[13] 层和 Relu 激活函数层，最后面的输出层除外。最后一层使用 Tanh 函数来限制图像的像素在 [0，255] 的范围内。Batch-Normalization 是非常重要的一个优化技巧，本文将在 3.4 节中详细说明。

3.4 Batch-Normalization

通常情况下，研究人员在输入训练数据之前会对数据进行预处理，比如减去

均值，甚至进行白化处理[17]，这样做是为了加快网络的收敛速度。在图 3-4 中，图 a 中的数据都分布在第一象限，设方程 y = W*x + b。通常随机初始化的参数是 0 均值的，所以拟合线会 y 从原点附近经过，如图 b 所示。这就需要多次训练才能接近目标线——紫色直线。若对数据进行白化预处理（使数据分散），不难发现初始的拟合线和目标线会很接近，如图 d。于是，网络收敛速度大大加快。

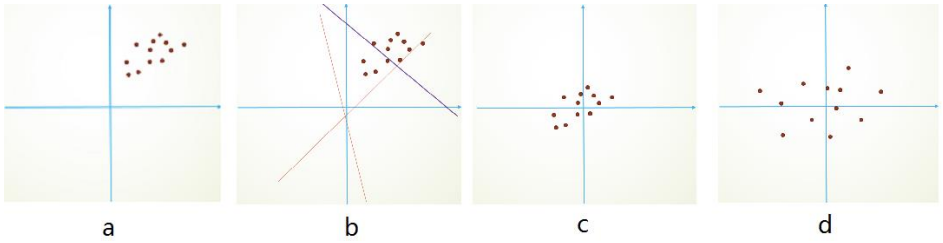

图 3-4 数据的 Normalization

3.3 节中提到，图像转换网络中的非残差卷基层都跟着一个 Batch-Normalization。Batch-Normalization[13] 就是一种用来解决上述问题（Covariate Shift 问题）的有效方法，它比白化的复杂度要低一些。

Batch-Normalization 不仅针对输入层数据进行 Normalization，而且对神经网络内层的输出值也进行归一化。这是因为随着数据在网络中的流动，数据在网络内层分布状态也是变化的（这被称为 Internal Covariate Shift）。于是，我们可以在每层神经网络后面，进行一次 Normalization，让数据保持一个较好地分布状态，从而提升网络的训练效率。从本质上来说，Batch-Normalization 是在解决逐渐消失的梯度问题。因为在每层神经元后面，Batch-Normalization 拉开了输出值的距离，所以梯度值又出现了。

Batch-Normalization 具体处理方法是：

1) 对于某一个批次的数据，让它们减去自身整体的均值，
2) 然后除以其整体的标准差。
3) 最后，再加入一定的变换系数和偏移就可以了。

其数学表达过程如下：

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

很明显在前三步的计算中，数据就已经被拉回了标准正态分布，但最后一步却增加了一个系数和偏置。这是因为，如果去掉这个步骤，神经网络每一层的分布都会是一样的，这就好比，深层神经网络其实只有一层，因为它们的分布都是一样的，深层网络好像失去了它原本的意义。所以这个系数的出现是为了让这些分布变得不一样，从而保留深层神经网络的本质意义。当然，这些系数和偏置的出现，也使得网络变得更加难训练。

3.5 Instance-Normalization

Batch-Normalization 对网络的优化效果非常好，但最后本文还是没有选择使用它优化图像转换网络。Dmitry Ulyanov 等人在实验中发现使用 Batch-Normalization 生成的图像转换的质量不是十分理想。他尝试用 Instance-Normalization[14]替换了 Batch-Normalization，发现图像生成质量有了明显的提高。如图 3-5，左侧为使用 Batch-Normalization 的效果图，右侧为使用 Instance-Normalization 的效果图

图 3-5 Normalization 优化效果对比图

Instance-Normalization 的计算方式和 Batch-Normalization 有些不同，它只针对单个数据进行计算均值和标准差，而不是批量数据，并且没有缩放系数和偏移。其数学表达式如下：

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2.$$

不难发现，Instance-Normalization 去掉了 Batch-Normalization 中缩放系数和偏移。这是因为对于“图片生成类网络”来说，Normalization 的过程仅仅破坏了图片的对比度、色彩信息，却不会影响图像的内容特征。Batch-Normalization 为了减轻对比度、色彩受到的拉伸，不得不在最后一步加入了一个系数和偏置。巧合的是，“快速艺术图片风格化算法”本就不需要原图的对比度、色彩信息，它只需要原图的轮廓信息就够了。于是，Instance-Normalization 直接地去掉了 Batch-Normalization 最后一步的缩放系数和偏移。因为减少了训练缩放系数和偏移的所需要的计算代价，网络的训练效果得到了进一步提升。

第四章 基于 Tensorflow 的算法设计与实现

4.1 算法系统设计

本文的第三章已经详细论述了“快速艺术风格化算法”的原理，要将这个算法快速可靠地应用于实际的产品中，选择一个可靠的开发框架是十分重要的。这一章将阐述如何使用 Tensorflow 来实现快速风格转换算法。有关 TensorFlow 的具 API 定义，可参考官方教程[18]。

本文实现的算法设计思路如下：

1) 读取训练图、风格图
2) 训练图输入“图像转换网络”得到生成图
3) 训练图、风格图、生成图输入“VGGNet”得到代价
4) 根据代价，用 Adam 优化器优化“图像转换网络”
5) 循环训练，直到网络收敛，保存模型文件
6) 读取模型文件，生成效果图

图 4-1 给出了详细的流程

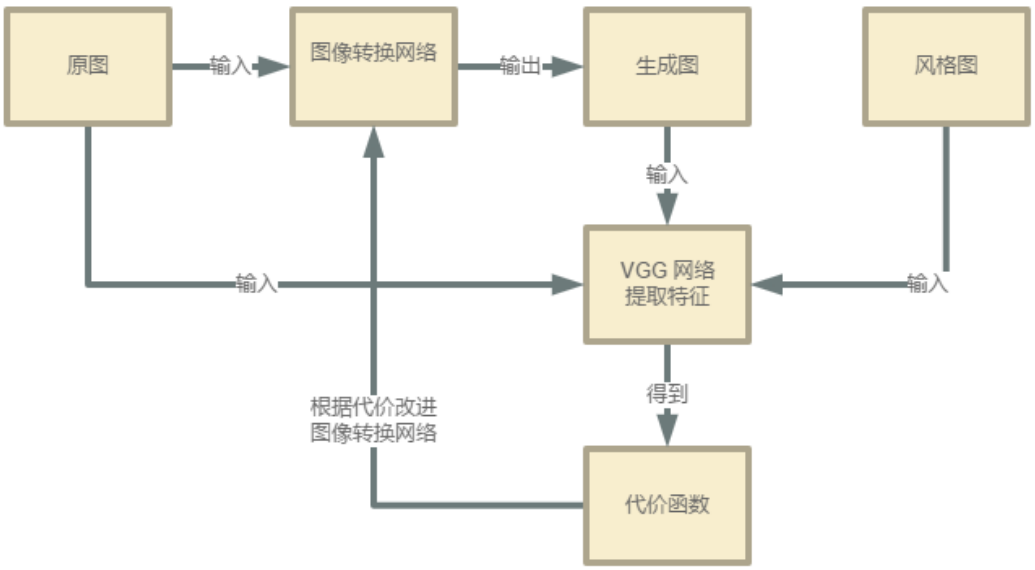

图 4-1 模型训练流程图

下面给出算法主要模块的实现分析，其他模块可参考系统源码[22]。

4.2 实现深层神经网络

本文使用预训练的 VGG19 模型作为“对抗网络”。

1) 先从 MatConvNet 获取 MATLAB 格式的 VGG19 模型文件[19]。
2) 然后使用 Scipy 库读取该模型文件。
3) 再结合 TensorFlow 的图系统便可构建好 VGG19Net。

在代码 4-1 中，net 函数返回深层神经网络每一层的引用，通过 net[“层名”]可获取指定层的输出。此外 Tensorflow-Slim 也提供了类似的 VGGNet 的模型文件和模型读取源码，可以在 GiHub 中直接下载使用。

```
def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', ......
    )

    data = scipy.io.loadmat(data_path)
    weights = data['layers'][0]

    net = {}
    current = input_image
        for i, name in enumerate(layers):
                kind = name[:4]
                if kind == 'conv':
                    kernels, bias = weights[i][0][0][0][0]
                    # matconvnet: weights are [width, height, in_channe
ls, out_channels]
                    # tensorflow: weights are [height, width, in_channe
ls, 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)
                net[name] = current

    return net
```

代码 4-1 VGG19 网络

代码 4-2 是本文实现的“图像转换网络”。tf.nn.relu 是 TensorFlow 中实现的 ReLu 激活函，conv2d，residual，instance_norm 是本文实现的卷积层函数，可在系统源码[22]中找到定义。y = (deconv3 + 1) * 127.5 是将 tanh 函数的输出转换为图像灰度范围。

```
def net(image, training):
    # Less border effects when padding a little before passing through
..
    image = tf.pad(image, [[0, 0], [10, 10], [10, 10], [0, 0]], mode='R
EFLECT')

        conv1 = tf.nn.relu(instance_norm(conv2d(image, 3, 32, 9, 1)))
        conv2 = tf.nn.relu(instance_norm(conv2d(conv1, 32, 64, 3, 2)))

        conv3 = tf.nn.relu(instance_norm(conv2d(conv2, 64, 128, 3, 2)))

        res1 = residual(conv3, 128, 3, 1)
        res2 = residual(res1, 128, 3, 1)
        res3 = residual(res2, 128, 3, 1)
        res4 = residual(res3, 128, 3, 1)
        res5 = residual(res4, 128, 3, 1)
        deconv1 = tf.nn.relu(instance_norm(resize_conv2d(res5, 128, 64,
 3, 2, training)))
        deconv2 = tf.nn.relu(instance_norm(resize_conv2d(deconv1, 64, 3
2, 3, 2, training)))
        deconv3 = tf.nn.tanh(instance_norm(conv2d(deconv2, 32, 3, 9, 1)
))

    y = (deconv3 + 1) * 127.5

    # Remove border effect reducing padding.
    height = tf.shape(y)[1]
    width = tf.shape(y)[2]
    y = tf.slice(y, [0, 10, 10, 0], tf.stack([-1, height - 20, width -
20, -1]))

    return y
```

代码 4-2 图像转换网络

4.3 实现代价函数

在 3.1 节中本文阐述了代价函数的数学原理，其中计算风格代价函数的基本步骤是：

1) 使用 VGG19 网络中的 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 层提取的特征信息计算 gram 矩阵。
2) 利用生成图和风格图的 gram 矩阵逐差求和得到风格代价。

本文把原图和风格图一次性输入 VGG19 网络，同时得到它们在网络每一层的特征信息，然后用 tf.split 分割特征信息，计算出相应的 gram 矩阵，最后用 tf.nn.l2_loss 求出二次代价。如代码 4-3 所示。

```
# 计算 gram 矩阵
def gram(layer):
    shape = tf.shape(layer)
    num_images = shape[0]
    width = shape[1]
    height = shape[2]
    num_filters = shape[3]
    filters = tf.reshape(layer, tf.stack([num_images, -1, num_filters]))
)
    grams = tf.matmul(filters, filters, transpose_a=True) / \
        tf.to_float(width * height * num_filters)

    return grams

# 计算风格损失
def style_loss(net, style_features_t, style_layers):
    style_loss = 0
    for style_gram, layer in zip(style_features_t, style_layers):
        generated_images, _ = tf.split(net[layer], 2, 0)
        size = tf.size(generated_images)
        layer_style_loss = tf.nn.l2_loss(
            gram(generated_images) - style_gram) * 2 / tf.to_float(size
)
        style_loss += layer_style_loss
    return style_loss
```

代码 4-3 风格代价函数

内容代价函数和风格代价函数的计算方法大同小异，只是由 gram 矩阵逐差变为特征图直接逐差，这里不再给出具体分析。最后可以得到整体的代价函数见

代码 4-4。

```
1.  total_loss = FLAGS.STYLE_WEIGHT * style_loss + FLAGS.CONTENT_WEIGHT * c
ontent_loss + \
2.              FLAGS.TV_WEIGHT * loss.total_variation_loss(generated)
```

代码 4-4 整体代价函数

代码 4-4 中的 total_variation_loss 是为了保证图像像素的平滑性而增添的代价，其详细实现可以参考系统源码[22]。

4.4 文件读取队列

本文使用的 COCO dataset 2014 数据集共有 8 万多张图片，几乎不能一次加载到内存中。TensorFlow 中的 string_input_producer 的文件队列提供了边训练边读取的功能，可以用该文件队列方便的读取大量图片。

在代码 4-5 中，tf.image.decode_png 是 TensorFlow 提供的图片解码工具，tf.train.batch 可指定 Mini-batch-SGD 的 batch 大小。

```
1.  def image(n, size, path, epochs=2, shuffle=True, crop=True):
2.
3.      filenames = [join(path, f) for f in listdir(path) if isfile(join(pa
th, f))]
4.      if not shuffle:
5.          filenames = sorted(filenames)
6.
7.      png = filenames[0].lower().endswith('png') # If first file is a png
, assume they all are
8.
9.      filename_queue = tf.train.string_input_producer(filenames, shuffle=
shuffle, num_epochs=epochs)
10.     reader = tf.WholeFileReader()
11.     _, img_bytes = reader.read(filename_queue)
12.     image = tf.image.decode_png(img_bytes, channels=3) if png else tf.i
mage.decode_jpeg(img_bytes, channels=3)
13.
14.     processed_image = preprocess(image, size)
15.     return tf.train.batch([processed_image], n, dynamic_pad=True)
```

代码 4-5 图片文件读取队列

4.5 训练和结果分析

在代码 4-6 中，variable_to_train 指定被训练的变量——即图像转换网络，这里使用了 Adam 优化器对网络进行训练，这是一种梯度下降算法的增强版。

```
# 被训练的变量
variable_to_train = []
        for variable in tf.trainable_variables():
            if not variable.name.startswith('vgg19'):
                variable_to_train.append(variable)

# 指定训练器
train_op = tf.train.AdamOptimizer(FLAGS.LEARNING_RATE).minimize(
      total_loss, global_step=global_step,var_list=variable_to_train)

# 迭代训练
try:
    while not coord.should_stop():
        sess.run([train_op])

    except tf.errors.OutOfRangeError:
        saver.save(sess,  FLAGS.MODEL_PATH + '-done')
        tf.logging.info('Done training -- epoch limit reached')
    finally:
        coord.request_stop()
```

代码 4-6 网络的训练

本文使用附录中的 COCO dataset 2014[20]的图像集来训练“图像转换网络”。Mini-batch-SGD 的训练尺寸为 4，内容和风格的比例在 1：10 左右，全变差误差系数为 $1e^{-6}$。实际的效果见图 4-2。

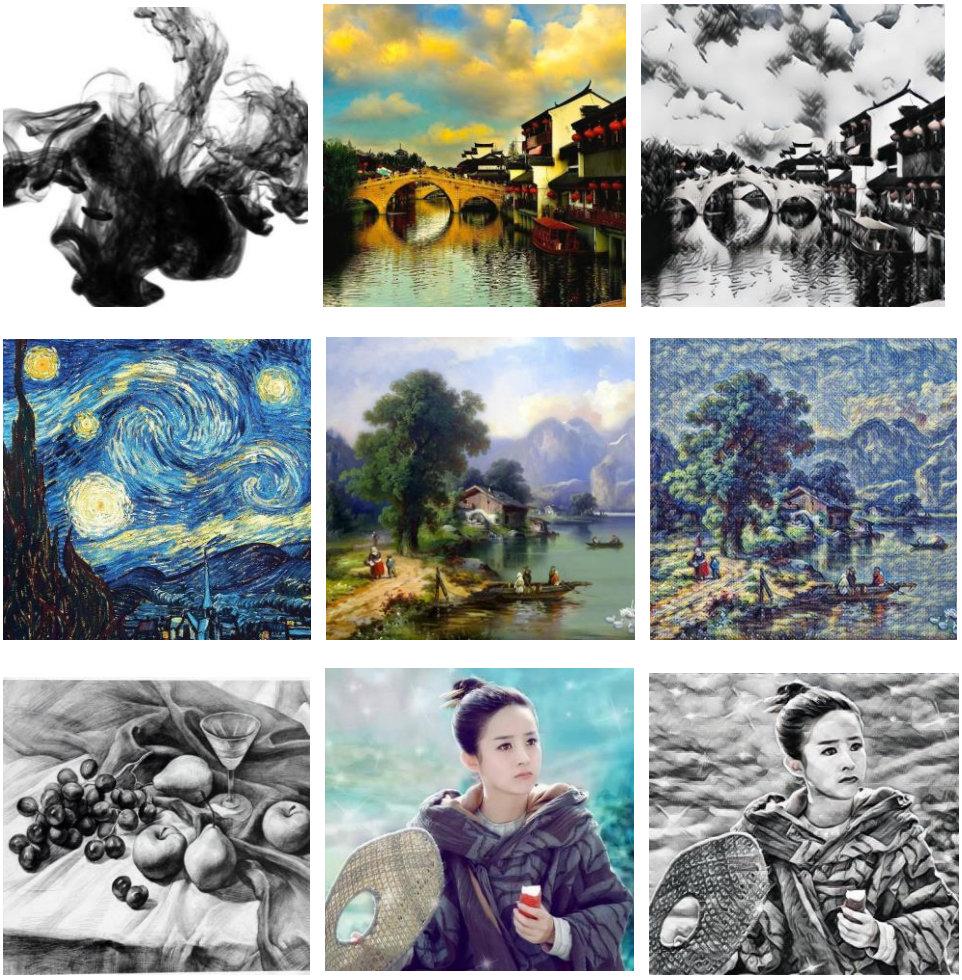

图 4-2 风格转换效果

通过多次的实验我们发现，“快速图片艺术风格算法”对纹理有着良好的学习能力，所以建议使用该算法学习风格特征比较明显的艺术图。本文尝试过基于“新海诚动漫风格”训练图像转换网络，最终效果不是很理想。

第五章 云平台下的系统设计与实现

5.1 基于 Flask 的后端设计

Flask 是一个灵活的 Web 应用框架。因为其使用简单的核心和小巧的身姿，深受开发者的喜爱。与此同时，使用和深度学习工具 Tensorflow 一致的开发语言可以有效降低开发成本。

最小的 Flask 应用示例：

```
1.  from flask import Flask
2.  app = Flask(__name__)
3.
4.  @app.route('/')
5.  def hello_world():
6.      return 'Hello World!'
7.
8.  if __name__ == '__main__':
9.      app.run()
```

本文基于 Flask 实现了图片风格迁移系统的后台接口，其接口设计如下

http://域名/index
http://域名/help
http://域名/transform

其中 /index 向客户端返回用户界面，/help 返回滤镜系统提供的模型类型。而/transform 接受一个 base64 编码的图片数据，模型代号及邮箱地址，即图片风格转换请求，其 JSON 格式为：

{ email: 邮箱地址,
filename: 图片文件名,
image: base64 编码的图片数据,
model: 模型代号 }

用户不仅可以通过 index 接口直接访问网页界面，还可以基于 help 和 transform 接口实现自己的客户端，比如安卓和 iOS 上的客户端。

5.2 分布式任务队列

在 transform 接口的实现中，不仅要考虑输入数据的验证和对恶意请求的检测，更要考虑系统的负载能力。图片转换是比较消耗计算资源的任务，如果同时有多个人对系统发起转换请求，必然会导致系统负担过大，从而导致服务器崩溃。

可以用一个分布式的异步队列来解决这个问题。异步任务队列，就是操作系统原理中经常提到的“生产者消费者模型”。Flask 进程作为生产者提交图片转换任务，然后任务队列进程作为消费者从缓存中读取并消耗这个任务，如图 5-1 所示。本文使用的是 Celery 分布式异步任务队列，它不提供消息中间件，这里使用 Redis 作为其消息代理。

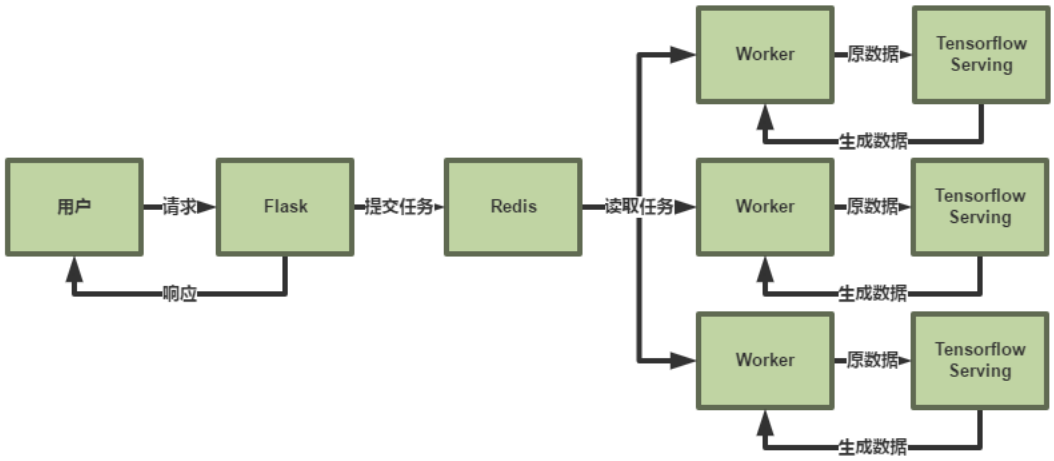

图 5-1 分布式任务队列

使用 Celery 提供的库函数，可以方便的实现异步任务请求。因此，用户无需等待后台转换结束就可以关闭网页。异步任务队列会在转换任务结束后，自动把图片发送到用户指定的邮箱。同时，用户也可以访问 Flask 相应的接口查询任务转换的进度。

如图 5-1 所示，本文使用高性能 Tensorflow Serving[21]搭建图片转换服务系统，以保证前馈网络的运算性能。（附录中提供的源码简化了具体实现，省去了 TensorFlow Serving）

至此，我们可以方便的把算法部署到分布式的云平台上，提供给用户优质的图片转换服务。

5.3 用户界面

本文基于Bootstrap前端框架和AJAX技术[16]实现了简洁无刷新的网页界面，实际效果见图5-2。页面初始化时会向 /help 接口请求 model 列表及相应的示例图片，然后异步加载和渲染这些图片。对于图片的上传，本文采用侦听页面 drag 事件的方法实现了拖拽上传功能，以改进用户的使用体验。

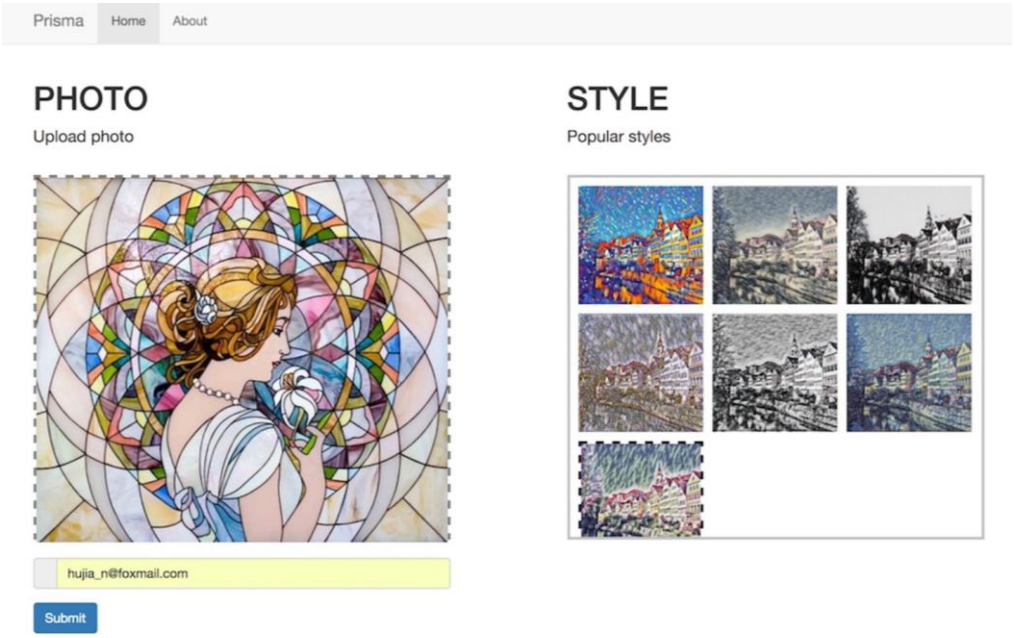

图5-2 网页界面

5.4 测试

1) 修改配置文件 default_config.py，填入自己的邮箱服务器帐号和地址
2) 安装好 Redis 的最新稳定版本，并将服务监听地址写入配置文件 default_config.py。然后启动 Redis，如图5-3所示。

```
➜  src git:(master) ✗ ./redis-server
1172:C 27 May 11:31:43.662 # Warning: no config file specified, using the defau
lt config. In order to specify a config file use ./redis-server /path/to/redis.
conf
1172:M 27 May 11:31:43.664 * Increased maximum number of open files to 10032 (i
t was originally set to 4864).
```

图5-3 启动 Redis

3) 用 pip 命令安装好 Celery 分布式异步任务队列，并启动 Celery，如图 5-4 所示

图 5-4 启动 Celery

4) 用 pip 命令安装好 Flask，然后启动 Flask，如图 5-5 所示

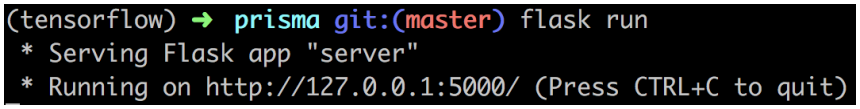

图 5-5 启动 Flask

5) 打开 localhost:5000 页面，拖入要上传的文件，并填入自己的邮箱信息，提交请求。稍等一会，便可以收到系统发来的转换好的图片，如图 5-6 所示。

图 5-6 收到转换好的图片

参考文献

[1] Gatys L A, Ecker A S, Bethge M. A Neural Algorithm of Artistic Style[J]. Computer Science, 2015.

[2] 机器学习. [EB/OL]. 2016.12.15. https://zh.wikipedia.org/wiki/机器学习

[3] Konečný J, Liu J, Richtárik P, et al. Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting[J]. Selected Topics in Signal Processing IEEE Journal of, 2014, 10(2):242-255.

[4] Hecht-Nielsen R. Theory of the backpropagation neural network[C]. Neural Networks for Perception. Harcourt Brace & Co. 1992:593-605 vol.1.

[5] 黄金平, 张正炳. 卷积定理与傅里叶变换性质及其应用的关系探讨[J]. 长江大学学报(自科版), 2016, 13(19):29-31.

[6] Nair V, Hinton G E. Rectified Linear Units Improve Restricted Boltzmann Machines[C]. International Conference on Machine Learning. DBLP, 2010:807-814.

[7] Lawrence S, Giles C L, Tsoi A C, et al. Face recognition: a convolutional neural-network approach[J]. IEEE Transactions on Neural Networks, 1997, 8(1):98.

[8] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

[9] Johnson J, Alahi A, Li F F. Perceptual Losses for Real-Time Style Transfer and Super-Resolution[J]. 2016.

[10] Goodfellow I J, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[C]. International Conference on Neural Information Processing Systems. MIT Press, 2014:2672-2680.

[11] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[C]. Computer Vision and Pattern Recognition. IEEE, 2015:770-778.

[12] Radford A, Metz L, Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[J]. Computer Science, 2015.

[13] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network

Training by Reducing Internal Covariate Shift[J]. Computer Science, 2015.

[14] Ulyanov D, Vedaldi A, Lempitsky V. Instance Normalization: The Missing Ingredient for Fast Stylization[J]. 2016.

[15] Abadi M, Barham P, Chen J, et al. TensorFlow: A system for large-scale machine learning[J]. 2016.

[16] Galhardas H, Florescu D, Shasha D, et al. AJAX:an extensible data cleaning tool[C]. ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, Usa. DBLP, 2000:590.

[17] 主成分分析和白化. [EB/OL]. 2013.4.10. http://ufldl.stanford.edu/wiki/index.php/UFLDL 教程

[18] Getting Started With TensorFlow. [EB/OL]. 2017.6.1. https://www.tensorflow.org/get_started/

[19] VGG-19 pre-trained model. [EB/OL]. 2017.6.1. http://www.vlfeat.org/matconvnet/models/beta16/imagenet-vgg-verydeep-19.mat

[20] COCO dataset 2014. [EB/OL]. 2017.6.1. http://msvocds.blob.core.windows.net/coco2014/train2014.zip

[21] TensorFlow Serving. [EB/OL]. 2017.6.1. https://www.tensorflow.org/deploy/tfserve

[22] Prisma. [EB/OL]. 2017.6.3. https://github.com/hijkzzz/prisma

致 谢

这次的毕业论文设计是在我的指导老师——陆伟老师亲切关怀和悉心指导下完成的。从毕业设计选题到设计完成，陆老师给予了我耐心指导与细心关怀，有了陆老师耐心指导与细心关怀我才不会在设计的过程中迷失方向，失去前进动力，陆老师有严肃的科学态度，严谨的治学精神和精益求精的工作作风，这些都是我所需要学习的，感谢陆伟老师给予了我这样一个学习机会，谢谢！此外，还要感谢 14011301 班的巫健同学无私地分享了他的深度学习平台远程账号，感谢！

毕业设计小结

毕业论文是本科学习阶段一次非常难得的理论与实际相结合的机会，在这次毕业设计中，我比较完整的实现了一个深度学习算法系统，摆脱了单纯的理论知识学习状态。这不仅锻炼了我的综合运用所学的专业基础知识，分析实际工程问题的能力，同时也提高我查阅文献资料以及代码编写的能力。最后，希望这篇文章能给喜欢图像风格迁移和深度学习的同学带来一定的参考价值。