大连理工大学本科毕业设计（论文）

基于深度学习的艺术风格转化应用

Art style transfer application based on deep learning

学 院（系）：软件学院

专　　业：数字媒体技术

学 生 姓 名：程紫阳

学　　号：201492139

指 导 教 师：张宪超

评 阅 教 师：宗林林

完 成 日 期：2018 年 6 月 5 日

大连理工大学

Dalian University of Technology

摘　要

深度学习这几年的发展使其在很多领域都取得了显著的成果，其中图像处理方面更是如此，特别是卷积神经网络（Convolutional Neural Network, CNN）在图像识别，图像内容语义分割以及图像生成方面的表现尤为出色。本文主要研究的图像艺术风格转化问题同样结合 CNN 作为图像特征的提取器。图像艺术风格转化问题可以被描述为，将一张图像（称为内容图像）的风格转化为给定的风格图像的风格样式，同时尽量保留内容图像的图像内容。这种风格转化可以有绘画风格和真实风格两种，分别采用两种不同的算法进行实现。更为具体的，每一种算法都有快速和慢速两种实现方式。算法可以将任意图像转化为给定艺术绘画作品或者精美摄影图像的风格。

本文结合 VGG19（一种用于图像识别的成熟的 CNN 结构）提取图像特征，采用 Content Loss 和 Style Loss 作为主要的损失函数，利用基于梯度下降法的迭代优化方法对算法进行优化，最终重现了现有算法的成果，并提出了新的模型结构以实现快速真实风格转化。为了让算法运用于应用场景中，开发了一个轻量级的网站。由于结合了多个算法，重现算法的效果不劣于原算法。新模型结构的效果虽没有原算法好，但是实现了快速转化，可达到实时转化的程度，同时达到了端到端（End-to-End）的封装程度，而不用另外进行数据的特殊处理。

关键词：图像风格转化；深度学习；端到端；网站应用

Art style transfer application based on deep learning

Abstract

The development of deep learning over the past few years has made remarkable achievements in many fields, especially in image processing, especially the Convolutional Neural Network (CNN) in image recognition, image content semantic segmentation and images generation. The image art style transfer problem mainly studied in this paper also combines CNN as an image feature extractor. The problem of image art style transfer can be described as transforming the style of an image (called a content image) into the style of a given style image while preserving the image content of the content image as much as possible. This style transfer can have two kinds, painting style and real style, using two different algorithms to achieve. More specifically, each algorithm has two implementations, fast one and slow one. The algorithms can be used to transfer any image into the style of a given artistic painting or beautifully photographed image.

This paper combines VGG19 (a mature CNN structure for image recognition) to extract image features, uses Content Loss and Style Loss as the main loss functions, and uses an iterative optimization method based on gradient descent to optimize the algorithm and eventually reproduces the results of the existing algorithms, and proposes a new model structure to achieve fast real style transfer. In order to apply the algorithm to the application situation, a lightweight website was developed. Due to the combination of multiple algorithms, the effect of the reproduced algorithm is not inferior to the original algorithm. Although the effect of the new model structure is not as good as the original algorithm, it achieves fast transfer and achieves the degree of real-time transfer. At the same time, it achieves an end-to-end encapsulation level without any special data processing.

Key Words：Image Style Transfer；Deep learning；End-to-End；Web application

目　　录

1 绪论

1.1 课题背景

2006 年 Hinton 等人提出深度学习的概念，采用多层神经网络的新模型，并结合反向传播算法进行优化，在多个领域实现了技术突破。深度学习大大提高了不同人工智能（Artificial Intelligence, AI）任务中的技术，并取得了最先进的成果，如对象检测[1]，面部识别[2]，机器翻译[3]等。深层神经网络架构为深度学习提供了解决更复杂的 AI 任务的可能性。因此，除了诸如对象检测，面部识别或机器翻译之类的传统任务，研究人员正在将深度学习扩展到各种不同的应用领域中，例如使用循环神经网络对语音信号进行去噪处理[4]，使用堆栈自动编码器来发现基因表达的聚类模式[5]，使用卷积神经网络进行自动驾驶[6]等。

本文所涉及的图像风格转化任务主要运用卷积神经网络进行算法的实现。卷积神经网络是一种前馈神经网络，一般由多个卷积层、非线性化层、池化层（Pooling）以及最后的全连接层组成。卷积层对输入进行卷积计算，得到的结果经过非线性化层的激活函数（Activation Function），再经过池化层进行下采样（降维），最后输入到全连接层。首先介绍一下卷积操作。假设输入图像是二维矩阵，每个像素值都是输入层的一个神经元，权值也用矩阵来表示，这个权值矩阵叫做卷积核（Kernel），也可以称为滤波器（Filter），卷积核代表了看这个图像时的感受野，从直观上讲，感受野就是视觉感受区域的大小。在卷积神经网络中，感受野的定义是卷积神经网络每一层输出的特征图（Feature Map）上的像素点在原始图像上映射的区域大小。不过卷积核是与输入图片的二维矩阵滑动计算的，这里涉及到了权值共享的问题。计算的过程如图 1.1 所示。

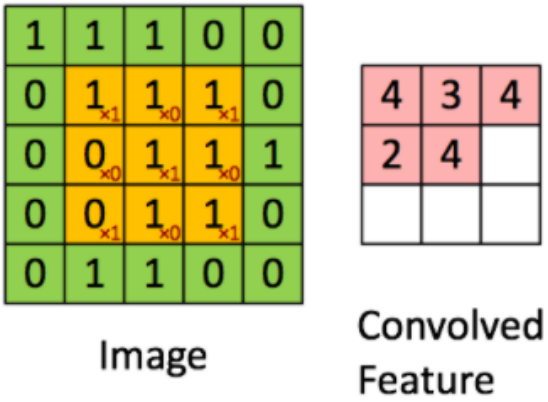

图 1.1 卷积操作

图中黄色部分为 3×3 卷积核，绿色的为 5×5 输入矩阵，黄色部分的卷积计算结果是右边矩阵最后一个 4，卷积核从输入矩阵左上至右下依次进行滑动计算，每次都计算相应位置的乘积再相加，得到卷积后的矩阵（右边矩阵）。每次滑动的一格代表步长（Stride）为 1，也可以为其它值。然后对右面矩阵的值通过激活函数进行非线性化处理。非线性化处理就是将卷积之后的特征图（Feature Map）利用非线性函数计算出新的结果，常用的非线性激活函数有 sigmoid 函数，tanh 函数，relu 函数等，在卷积网络中除了最后层之外，一般都会采用 relu 激活函数，最后一层激活函数的选择与具体执行的任务有关，relu 函数的表达式为$y=\begin{cases}x & if\ \ x>0\\0 & if\ \ x\le 0\end{cases}$，图 1.2 为 relu 函数的图像：

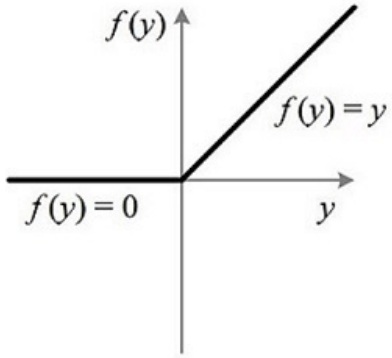

图 1.2　relu 激活函数图像

池化层进行下采样操作，目的是减小特征图大小，池化规模一般为 2×2。常用的池化方法之一是最大池化（Max Pooling），即取 4 个点的最大值，如图 1.3 所示。还可以使用平均池化（Average Pooling），即将最大化操作换成求平均值操作即可。

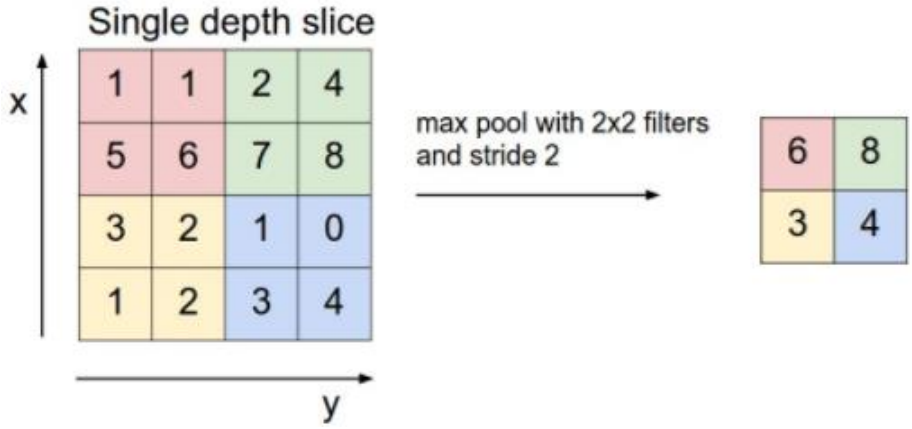

图 1.3　池化操作（这里为最大池化）

卷积神经网络利用这些操作可以不断提取图像在不同层次的特征图，然后用于图像分类等问题。图像风格转化理论上也可以通过多层卷积神经网络来提取图像里面可能含有的一些表示风格或者内容的特征，然后利用这些特征构建合适的损失函数，最后达到风格转化的目的。

Gatys 等人[7,8]就利用卷积神经网络成功分离了图像的内容和风格特征，将图像风格转化问题简化为一个基于卷积特征的优化问题，并得到了可以和著名绘画作品相媲美的图像，成为深度学习图像处理领域的又一大亮点。

1.2 研究现状

第一次使用神经网络进行图像风格转化的方法是由 Gatys， Ecker 和 Bethge 提出的基于卷积神经网络进行图像风格转化的算法[7,8]。深度卷积神经网络可以从图像中提取高级语义信息，Zeiler 和 Fergus 在他们的研究中将深度卷积神经网络不同网络层学习到的图像特征进行可视化[9]，图 1.4 是他们的实验结果。

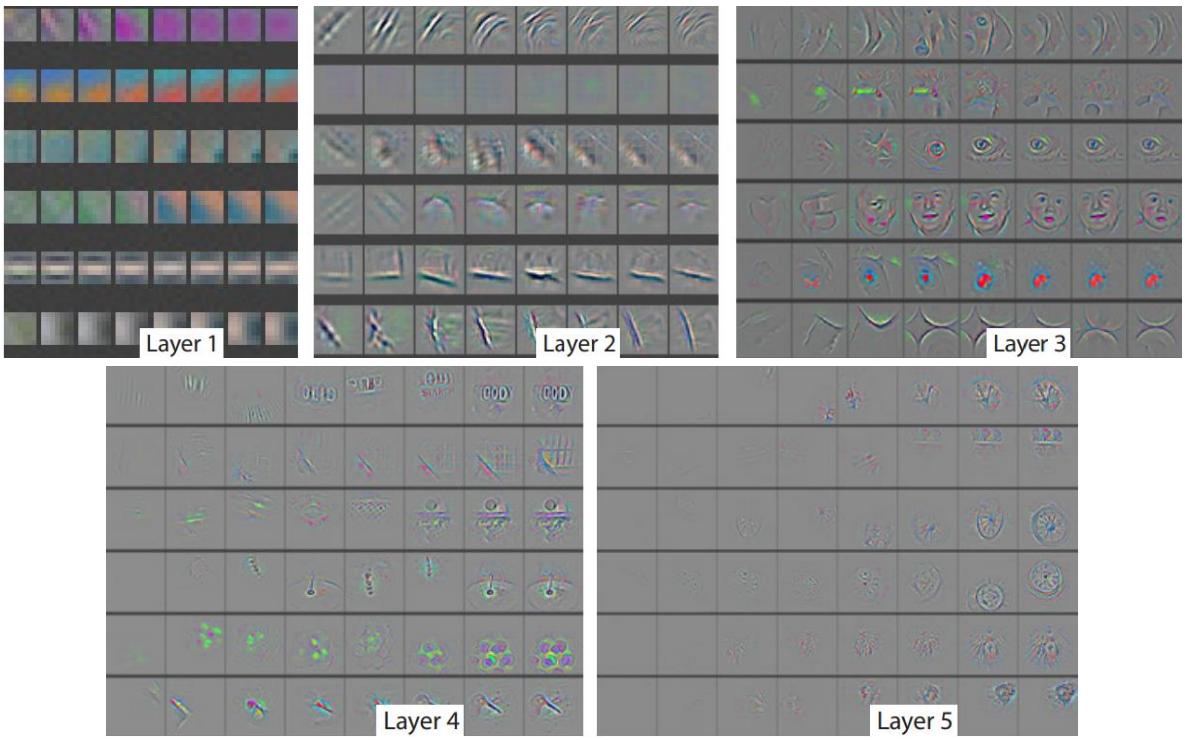

图 1.4　深度卷积神经网络图像特征可视化

可以看出深度卷积神经网络每层都提取到了不同程度的图像特征，而且更加高级，可以表征图像中物体的类别甚至位置等高级信息。因此如果使用成熟的卷积神经网络作

为图像的特征提取器，那么就可以独立提取出图像的内容特征以及风格特征，并将其应用于风格转化，事实上 Gatys 等人就是这么做的，他们提出一种艺术风格转化的神经网络算法，通过来自最先进的卷积神经网络的特征表示来约束风格转化，将转化过程简化为一个单独神经网络内的优化问题，具体的，他们采用 VGG19 网络[10]（一种成熟的用于图像分类的深度卷积神经网络）进行图像特征的提取，并成功将内容特征和风格特征分离，如图 1.5 所示。

利用 VGG19 网络不同层的特征将图像进行重建还原，图的上半部分是风格重建，分别利用‘conv1_1’(a)，‘conv1_1’和‘conv2_1’(b)，‘conv1_1’，‘conv2_1’和‘conv3_1’(c)，‘conv1_1’，‘conv2_1’，‘conv3_1’和‘conv4_1’(d)，‘conv1_1’，‘conv2_1’，‘conv3_1’，‘conv4_1’和‘conv5_1’(e)层重建输入图像的风格，由图可见，越是高层的特征，风格重建的结果就越粗粒度化。下半部分是内容重建，分别利用‘conv1_2’(a)，‘conv2_2’(b)，‘conv3_2’(c)，‘conv4_2’(d)和‘conv5_2’(e)层重建输入图像的内容，由图可见，越是底层的特征，重建的效果就越精细，越不容易变形。故利用‘conv4_2’(d)层作为输入图像的内容特征，因为其既满足精细的内容样式又有一定的微小形变，有利于风格化的艺术效果。利用‘conv1_1’，‘conv2_1’，‘conv3_1’，‘conv4_1’和‘conv5_1’(e)层作为输入图像的风格特征，因为其更加符合风格的粗粒度样式。因此，利用不同卷积层中的特征值作为图像的内容以及风格特征表示，然后在输入图像和内容图像以及输入图像和风格图像的对应特征值之间应用 L_2 损失，不断优化损失使其最小，同时利用梯度信息改变输入图像的像素值，最终得到风格转化之后的图像。

Gatys 等人的方法是第一个利用深度神经网络进行图像风格转化的算法，但是他们的方法速度太慢，每次必须进行一个完整的优化过程才能得到最终结果。在 Gatys 等人的风格迁移方法中，把生成图像的过程当做一个“训练”的过程。每生成一张图像，都相当于要训练一次模型，这可能需要迭代几百几千次。重新训练一个模型相对于执行一个已经训练好的模型来说相当费时。如果把生成图像当做一个“执行”的过程，而不是一个“训练”的过程，那么速度就可以得到有效提升，之后的几个改进算法都采用了这样的思想。文献[11]和文献[12]为每个风格图像训练一个单独的前馈神经网络，达到一个模型完成一种绘画风格转化的效果。文献[13]和文献[14]的风格转化不局限于某一种绘画风格，而是可以做到任意两张图像的风格转化。文献[15]将风格转化从绘画风格迁移到真实图像风格。

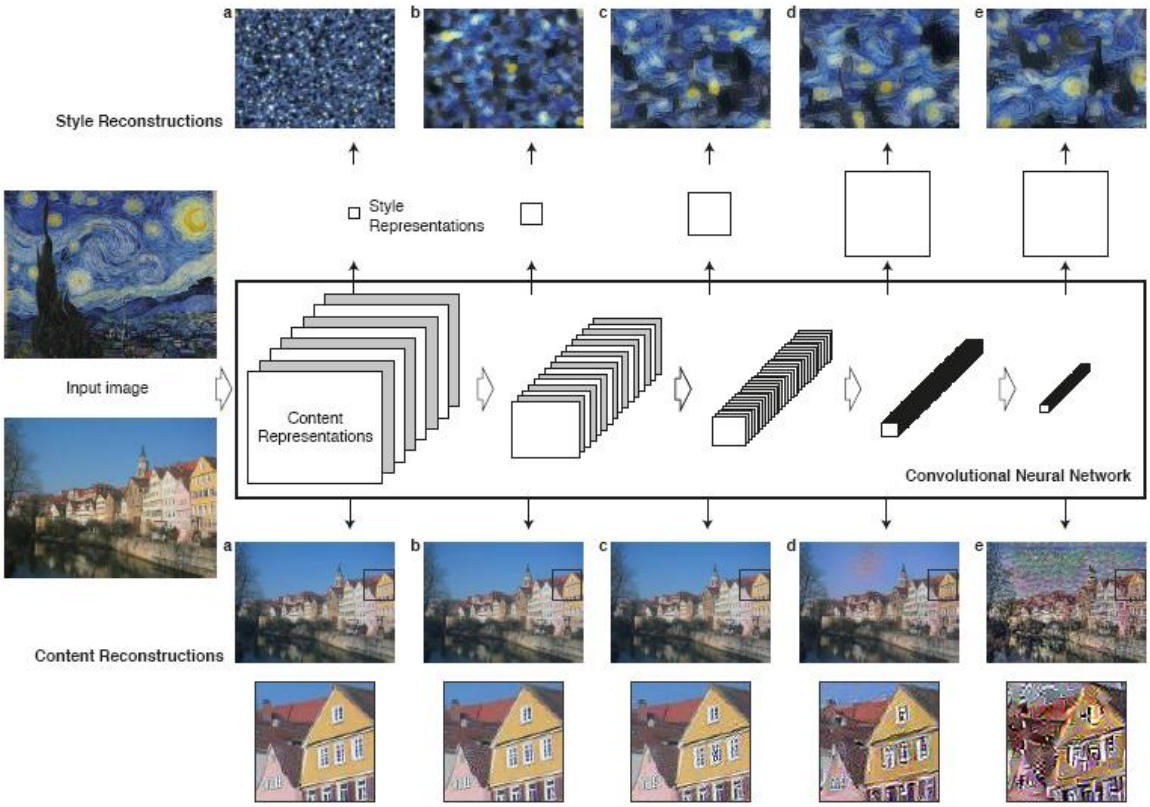

图 1.5　深度卷积神经网络中的图像表示

1.3　本文内容及组织结构

总的来说，可以实现任意两张图像之间风格转化的方法，速度上虽然是最好的，但是生成的图像质量也是最低的，而 Gatys 等人的方法虽然速度最慢，但是生成图像的效果是最好的，而一个模型对应一个风格图像的绘画风格转化方法则是折衷的存在，生成图像的效果较好，同时风格转化的速度很快。

本文旨在结合多种已有的绘画风格转化算法，复现典型的算法结果，其中包括 Gatys 等人的算法[7,8]，以及结合多种快速转化算法的折衷方法，实现 Luan 等人的风格转化算法[15]，并尝试完成快速真实图像风格转化算法。快速真实图像风格转化算法的主要思想是将 Luan 等人[15]的工作和 Johnson 等人[11]的工作相结合，将计算代价放在模型的训练阶段，以此提高算法的测试速度，同时运用实例归一化（Instance Normalization）增强风格转化效果，实现真实图像的快速风格转化。最后，编写简单的网站将算法应用于实践，使风格转化应用达到端到端的封装程度。接下来本文首先介绍使用到的相关技术，从第三章开始逐一介绍本文实现的算法，第 4.2 节介绍提出的简易快速真实图像风格转化算法，最后第五章介绍网站的设计和实现。

2 相关技术

2.1 图像风格转化理论介绍

Gatys 等人将艺术绘画风格转化问题可以看做是纹理转化问题，其关键点在于如何表示以及如何分离图像的内容和风格，传统的方法基本只能通过人工控制数据集的获取方式得到相同图像的不同风格，比如获得不同光照，不同色彩下的相同物体的图像。这种方法效率非常低，而且效果不好。Gatys 等人采用卷积神经网络分别获得图像的内容和风格表示，正如绪论部分介绍的，用于物体识别的成熟的卷积神经网络在不同层产生图像的不同表示，使物体信息沿着处理的层次越来越明显，高层捕捉的是高度概念化的图像表示，而不是精确的像素值，低层捕捉的是原始图像的精确像素值表示。因此假如选择低层作为内容特征表示，那么风格化之后的图像会保留更多的细节，相对的其艺术感不强，因此建议使用处于中等层的卷积层特征作为内容表示。如果只使用高层卷积特征作为风格表示，那么生成的图像缺乏风格图像的细节，因此采用从低层到高层的不同层的多尺度组合作为最终的风格表示，既可以保持整体的风格样式，又有细节的风格刻画。内容表示和风格表示无法完全分开。进行风格转化的时候，通常生成的图像无法同时完全匹配内容和风格两个约束，因此可以调整两者的比重来调整内容和风格的重点。当强调风格的时候，转化的结果是更加纹理化的，强调内容的时候，可以清晰看出具体的内容，但是风格效果不好。

Ulyanov 等人提出纹理网络（Texture Networks）完成图像的风格转化任务[12]，他们的算法将计算负担转移到学习阶段。给定一个纹理的例子，纹理网络可以训练一个紧凑的前馈卷积神经网络用以生成具有任意大小的相同纹理的多个样本，并将艺术风格从给定图像转移到任何其他图像，最后可以生成质量与 Gatys 等人的方法相媲美的图像，但速度要快数百倍。但是纹理网络涉及到多次的上采样操作，然后将上采样的结果作为附加特征通道连接到后面神经网络层的张量中，特征通道最终会从 8 增加到 40，这部分的操作造成了性能上的瓶颈，对进一步提升速度造成了阻碍。

Johnson 等人的方法[11]同样是将计算负担转移到学习阶段，通过训练一个前馈卷积神经网络用以生成转化之后的风格图像，提出使用感知损失函数来训练前馈网络，解决了由 Gatys 等人提出的实时优化问题，与基于优化的方法相比，Johnson 等人的方法提供了类似的定性结果，但速度提高了三个数量级。具体的，Johnson 等人提出的网络结构分为图像转化网络（Image Transform Net）f_w和损失网络（Loss Network）$\emptyset$两部分，其中图像转化网络f_w是个前馈神经网络，由多个卷积层以及残差层[16]组成，作用是将输入图像x转化为风格化之后的图像$\hat{y}$。损失网络$\emptyset$是已经训练完毕的 VGG19 网络，此网

络在整个算法训练过程中不改变，只是用于产生损失函数，$\emptyset$接受$\hat{y}$，结合风格图像y_s和内容图像y_c，定义内容损失l_{feat}，风格损失l_{style}。接下来就是一个优化过程，通过反向传播算法不断改变图像转化网络f_w的权值，使其产生的图像$\hat{y}$同时具有y_s的风格特征和y_c的内容特征，总的来说 Johnson 等人的方法既实现了任意大小图像的风格转化，并且相比较 Gatys 等人的方法速度有非常显著的提升，风格转化的效果也很好，因此本文着重借鉴 Johnson 等人的方法进行修改和实现。

Huang 和 Belongie 的方法首次实现了任意风格的快速转化[14]，与文献[11,12]的方法不同，文献[14]用一个模型就可以实现多种风格图像的风格转化。该方法的核心在于自适应实例规范化（AdaIN）层的使用，它将内容特征的均值和方差与风格特征的均值和方差对齐，最终实现快速风格转化，而不会受到预先定义的风格图像的限制。此外，他们的方法允许灵活的用户控制，如内容风格折衷，多个风格插值，颜色和空间控制等，所有这些都使用单一的前馈神经网络完成。

Chen 和 Schmidt 的工作直接舍弃了训练一个可保存的前馈神经网络的思想，而是运用新的风格交换（Style Swap）技术和逆网络（Inverse Network）结构实现不用多次迭代训练而只需要一次前向传播就能转化图像风格[13]。简单地说，他们利用 VGG19 网络先提取出内容和风格图像的特征，然后在 Style Swap 过程中匹配两者最相近的特征块，并最小化相近特征块的距离，使内容特征和风格特征融合，再由 Inverse Network 将高级图像特征逆卷积为输入图像的大小，此时图像就同时具备内容图像的内容特征和风格图像的风格特征了。这种方法的速度确实非常快，而且也可以实现任意两张图像的风格转化，但是缺点是得到的图像质量不高。

上面提到的算法都是将艺术绘画作品作为风格图像的情况，当风格图像是真实图像，比如风景照片的时候，上述方法都会出现明显的缺点——得到的结果图像都有绘画的变形，而不能保证图像的逼真性。Luan 等人的工作解决了真实图像的风格转化问题[15]。他们的出发点主要有两个，一是如何防止图像中的空间扭曲，二是如何防止不匹配的图像块之间的风格转化而导致的“伪像”，比如在天空中绘制了建筑的纹理。具体的方法是利用局部仿射将从输入到输出的转化限制在颜色空间中，这种方法成功地抑制了形状的扭曲，同时利用图像语义分割产生额外的掩模，将掩模作为附加通道添加到损失函数中，抑制了不相关内容之间的风格转移，使结果更加逼真，最后在广泛的场景中都产生令人满意的真实风格的转化，包括一天中的时间，天气，季节和灯光场景等的转移。Luan 等人的算法效果非常好，但是速度慢，每次需要长时间的优化过程才能完成转化。

值得一提的是这几种算法基本都利用了 Ulyanov 等人发现的小技巧[17]，即使用实例归一化（Instance Normalization）比使用批量归一化（Batch Normalization）生成的图像

的效果要好，尤其是可以缓解批量归一化造成的边缘粗糙和边缘模式奔溃问题，让生成的图像看起来更加平滑，同时一定程度上避免了风格纹理将图像中的物体完全掩盖的现象。

2.2 开发技术介绍

本文涉及的开发技术主要有深度学习技术和网站开发技术。自从深度学习在多个领域实现突破进展之后，越来越多的深度学习框架被设计和开发，供研究和工程人员使用。本文采用 TensorFlow 深度学习框架进行所有算法的实现。TensorFlow 框架自带的 TensorBoard 工具可以很方便地记录训练过程中损失函数的变化，也便于保存训练中产生的临时图像，更重要的是 TensorBoard 可以绘制完整的模型结构，让开发者对模型有非常直观的认识，也便于排查编程问题。本文使用的网站开发技术是 Python 网站开发框架中的 Django 框架。Django 可以让开发者在较短的时间内开发出高可用性的网站应用，同时它自带一个简单的后台管理系统，方便开发者在网站正式部署之前完成对网站生产环境的调试。下面给出硬件环境配置和软件环境配置。

表 2.1 硬件环境配置

硬件	型号
CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHZ
GPU	NVIDIA GeForce GTX 1080 Ti, 10.91G
内存	64G
硬盘	5T

表 2.2 软件环境配置

软件	版本
Ubuntu	Ubuntu 16.04.3 LTS
Python	3.4.5
Anaconda3	4.2.0-Linux-x86_64
TensorFlow-GPU	1.4.0
Tensorboard	0.4.0
Numpy	1.14.2
Django	2.0.4
Scipy	1.0.1
CUDA	7.5.17
Cudnn	6.0.21
Scikit-image	0.13.1

3 艺术绘画风格转化

3.1 慢速转化算法

参考文献[7,8]的工作，完成慢速艺术绘画风格转化算法。算法每次都需要进行一遍训练优化过程，利用不同的两张图像，其中一张作为风格图像，另一张为内容图像，最后需要将内容图像的风格转化为给定的风格图像的风格。

实验使用的卷积神经网络是 VGG19，网络结构共为 19 层，其中 16 个卷积层，5 个池化层，不使用全连接层。采用平均池化代替原论文中的最大池化。每个卷积层都有一个卷积核，卷积核对每层的输入图像进行卷积运算，得到不同层的特征图，作为相应的内容和风格表示，进而根据定义的损失函数进行优化。另外，为了使生成的图像在空间上更加平滑，采用全变分正则化（Total Variation Regularization, TV 正则化）约束生成的图像，文献[7,8]中没有此项。TV 正则化可以用于图像去噪、图像反卷积和图像修复等。支持三种噪音模型：高斯、拉普拉斯和珀松。

3.1.1 内容损失

将一张图像输入到 VGG19 网络中，VGG19 每一层的卷积核对其进行卷积计算，如果某层中有N_l个卷积核，l表示层数，那么该层就有N_l个特征图，而且每个特征图的大小为M_l，其中M_l为特征图的长乘以宽乘以通道数的结果。因此层l中的特征表示可以存储在一个矩阵当中，并用F_l表示：$F_l \in R^{N_l \times M_l}$，其中$F_{ij}^l$是第$l$层中第$i$个卷积核在第$j$个位置的响应（也就是卷积的结果）。设$O$为输出图像（也就是生成的图像），$C$为内容图像，$O^l$和$C^l$分别是输出图像和内容图像在第$l$层的特征表示，定义算法的内容损失为：

$$L_{content}(O, C, l) = 1/2 \sum_{i,j} (O_{i,j}^l - C_{i,j}^l)^2 \qquad (3.1)$$

然后利用基于梯度下降的优化算法进行优化，不断改变输出图像O的像素值，直到其在 VGG19 的某一层产生与内容图像C相同的特征。本文选择 VGG19 的 relu4_2 层作为内容特征层，产生的图像具有很好的艺术效果。

3.1.2 风格损失

与内容特征相对应的，可以定义每层的风格特征，用来计算不同卷积核响应之间的相关性。与内容特征不一样的是，风格特征不直接采用卷积核响应作为特征，而是利用 Gram 矩阵来表示，Gram 矩阵是图像处理中常用的特征相关性表示方法，可以写成$G^l \in R^{N_l \times N_l}$，表示第$l$层的 Gram 矩阵，$N_l$依然是第$l$层的卷积核的个数（也就是产生的特征图的个数），其中G_{ij}^l是第l层向量化之后的第i个特征图F_{ik}^l和第j个特征图F_{jk}^l之间的内积：

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l} \tag{3.2}$$

设O为输出图像（也就是生成的图像），S为风格图像，且$G^{l}[O]$和$G^{l}[S]$分别是输出图像和风格图像在第l层的 Gram 矩阵表示，那么该层的风格损失可以为：

$$E_{l} = 1/(4N_{l}^{2}M_{l}^{2}) \sum_{i,j}(G^{l}[O]_{i,j} - G^{l}[S]_{i,j})^{2} \tag{3.3}$$

所有风格特征层的整体损失就可以写为：

$$L_{style}(O,S) = \sum_{l=0}^{L} w_{l}E_{l} \tag{3.4}$$

其中w_{l}是每层对总的风格损失贡献的权重因子，一般设置为$1/L$即可，L表示风格特征所用的卷积网络的层数。同内容损失一样，风格损失也利用反向传播和随机梯度下降算法不断改变输出图像的像素值，直到输出图像在每个 VGG19 风格层产生的 Gram 矩阵和风格图像产生的相近。算法实现中风格损失使用的 VGG19 层为 relu1_1，relu2_1，relu3_1，relu4_1，relu5_1，权重因子统一设置为 1/5。

3.1.3 风格转化

有了内容损失和风格损失，接下来引入全变分正则化项，就可以定义总损失函数了。全变分正则化在上文已经简单介绍过了，这里给出计算表达式，设O为输出图像，其中O_{ij}表示图像在(i,j)位置的像素值，那么全变分正则化项损失为：

$$L_{TV}(O) = \sum_{i,j}(O_{i+1,j} - O_{ij})^{2} + (O_{i,j+1} - O_{ij})^{2} \tag{3.5}$$

为了产生既符合内容损失又符合风格损失，同时考虑全变分正则化的风格化图像，需要定义总的损失函数。设O为输出图像，C为内容图像，S为风格图像，其中内容图像和风格图像都需要输入到算法当中，而输出图像可以采用随机噪声进行初始化。定义总损失函数为：

$$L_{total}(O,C,S) = \alpha L_{content}(O,C) + \beta L_{style}(O,S) + \gamma L_{TV}(O) \tag{3.6}$$

其中α，β和γ分别为内容，风格以及 TV 损失的权重参数，可以用来调节这三个损失的比重，实现中使用的值分别为 e0，e2，e-1 这三种数量级，具体的值可以根据情况调整。下面给出算法的伪代码：

表 3.1　慢速艺术绘画风格转化算法流程

算法 3.1　慢速艺术绘画风格转化算法	
输入：内容图像C，风格图像S，设定权重参数α，β和γ，设定总迭代次数 max_$iteration$，设定学习速率r，初始化迭代次数$t=0$	
输出：风格化之后的输出图像O	
1	计算内容图像C在 relu4_2 层的特征表示C^{relu4_2}，分别计算风格图像S在 relu1_1，relu2_1，relu3_1，relu4_1，relu5_1 层的 Gram 矩阵$G^l[S]$
2	while 不满足迭代停止条件：
3	计算输出图像O在 relu4_2 层的特征表示O^{relu4_2}，分别计算输出图像O在 relu1_1，relu2_1，relu3_1，relu4_1，relu5_1 层的 Gram 矩阵$G^l[O]$
4	计算损失函数值：
5	$L_{total}(O,C,S)=\alpha L_{content}(O,C)+\beta L_{style}(O,S)+\gamma L_{TV}(O)$
6	利用 Adam 优化器优化算法：
7	$Adam(L_{total}(O,C,S))$
8	更新输出图像O的像素值：
9	$Update(O_{i,j})$
10	更新迭代次数：$t=t+1$
11	end while

3.1.4　慢速算法实验结果

算法使用 Adam 优化方法[18]进行迭代优化，在众多基于梯度下降的优化算法中 Adam 算法在深度学习领域表现最稳定，而且效果很好。这里将学习速率设置为 1。实际操作中输出图像O的初始化使用的不是随机噪声，而是随机噪声和内容图像的加权混合，具体为噪声 0.6，内容图像 0.4，这样相比单纯的随机噪声可以减少总的迭代次数。算法使用 TensorFlow 深度学习框架实现，在单个 NVIDIA GeForce 1080 GPU 上运行 28 分钟左右，共迭代 8000 次，并得到与文献[7,8]类似的效果图，下面展示一些实验结果。

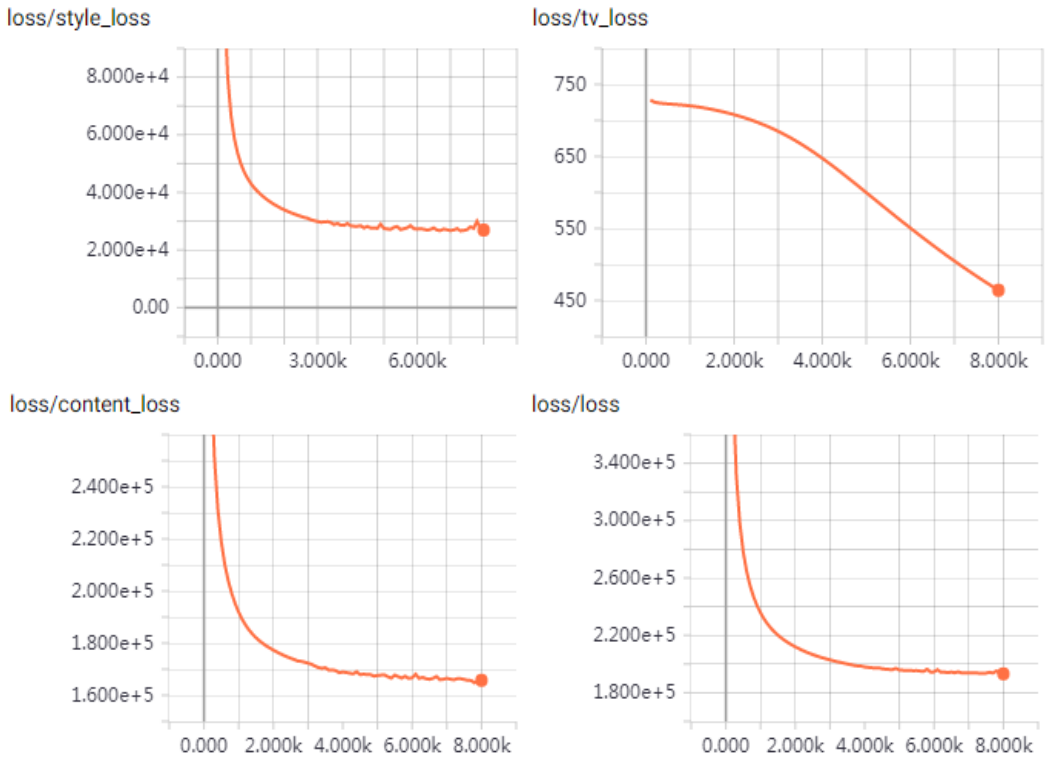

图 3.1　训练损失函数

图 3.1 是算法运行过程中损失函数的变化曲线，分别展示了风格损失，TV 损失，内容损失和总损失的曲线。可以看出所有损失值的抖动都非常小，基本都是随着迭代次数逐渐单调降低，但是前期降幅比较大，后期降幅很小，说明像素值在前期快速变为最优值附近，后期算法进行像素值的微调，使总损失缓慢地下降。

图 3.2 是算法运行的结果图，其中左上角的图像是风格转化的原图，也就是内容图像。下方是风格图像和对应的输出图像，可以看出算法的输出图像兼具风格图像的风格和内容图像的内容，体现在建筑的纹理和天空的纹理上，同时又保留了内容图像的细节，特别是天空的风格纹理较少，因为原内容图像的天空就是比较空旷的。

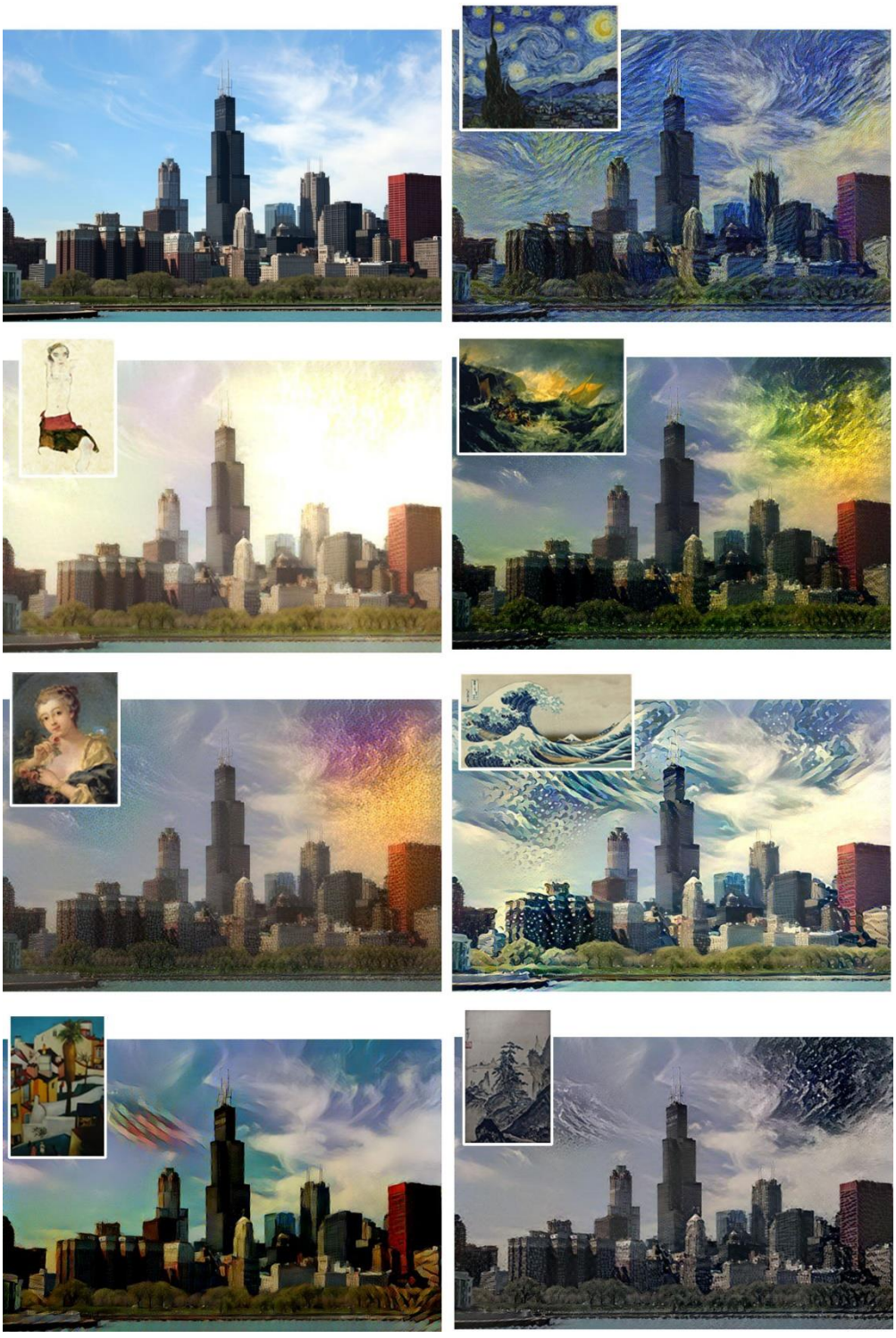

图 3.2　风格图和相应的生成图像

3.2 快速转化算法

参考文献[11][12][16]的工作，实现快速艺术绘画风格转化算法。为了解决慢速艺术绘画风格转化的速度问题，实现实时风格转化，本算法采用前馈神经网络将计算代价放在训练阶段，当训练完成之后，只需要一次前向传播即可完成风格化，避免了重复的迭代训练过程，并且损失函数几乎没有什么变化，只是增加了一个新的前馈神经网络结构，得到的风格化图像和慢速方法相比具有类似的风格质量，但是算法速度得到巨大提升。整个算法可以描述为图 3.3：

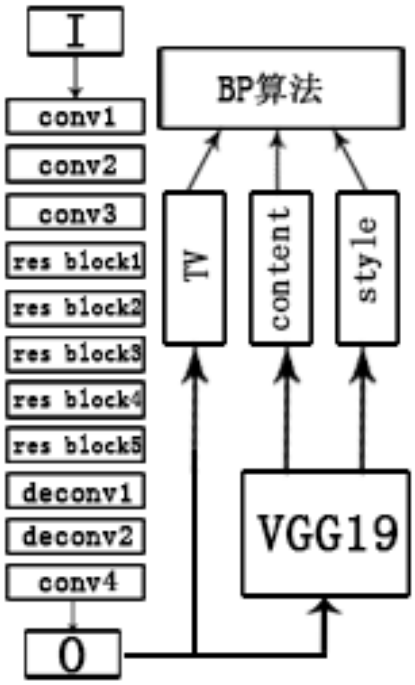

图 3.3　算法结构图

其中I为输入图像，即内容图像C的副本，输入前馈神经网络，经过网络中的 3 个卷积层，5 个残差块，2 个反卷积层以及最后的非线性层的处理之后，得到和输入图像同样尺寸的输出图像O。接着将输出图像O输入 VGG19 网络产生内容和风格特征，用于计算内容和风格损失，再结合 TV 损失项共同组成总损失函数，最后利用反向传播算法，进行迭代优化，但是优化过程改变的不是直接的像素值，而是前馈神经网络中定义的参数，通过优化网络中的参数，使其前向传播之后得到的输出图像O同时具有内容图像的内容特点和风格图像的风格特点。

3.2.1 前馈神经网络结构

本文的前馈神经网络结构参考文献[19]和文献[11]，具体参见上图，其中不使用任何的池化层，而是使用卷积和反卷积层来达到下采样以及上采样的目的，网络除了卷积层和反卷积层之外，还使用了 5 个残差块（Residual Block）。除了最后的输出层采用 tanh 激活函数用来保证输出像素值在 0 到 255 之间，其余层进行卷积操作之后都使用 relu 非线性激活函数，第一层和最后一层卷积层使用 9×9 大小的卷积核，其余层采用 3×3 大小的卷积核。此外每个卷积层之后都使用实例归一化而不是常用的批量归一化[20]的方法，具体原因在绪论部分也有讲，即可以一定程度上消除边缘模式崩坏现象，使生成的图像质量更高。批量归一化的工作是通过调整批量归一化中定义的两个参数 beta 和 gamma，将每一个批次的所有图像归一化到标准高斯分布，但是这容易使一个单独的图像被批次中其他的图像平均其统计量，这样产生的图像往往会被风格特征主导，图像中的物体会被严重风格化，因此需要将每一张图像单独作为一个个体进行归一化，才能达到更好的风格化效果，而且实例归一化在训练和测试阶段都可以使用，不像批量归一化只能在训练阶段使用，故采用实例归一化。

残差块的结构如图 3.4 所示，残差结构中的捷径连接（Shortcut Connection）部分将输入直接无损地向网络的深层传递，从而一定程度上避免了梯度消失现象，可以使网络变得更深，同时将原始输入和卷积之后的输出相结合构成新的输出，这可以在一定程度上防止图像的模式崩坏现象，因为在大多数情况下输出图像应该和输入图像共享结构，提高生成图像的质量。

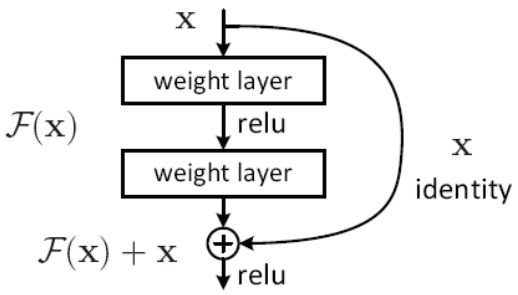

图 3.4 残差块结构

前馈网络在训练阶段，为了加快训练速度和数据统一，输入图像*I*都转化成大小为 3×256×256 的彩色图片。由于前馈神经网络是全卷积的，即整个网络没有全连接层，并且保持了输入输出的维度大小相等，因此尽管在训练的时候，图像大小为 3×256×256，但训练结束后，网络可以接受任意大小和分辨率的彩色图像。对于一个输入图像，首先

采用步长为 2 的卷积层进行下采样操作，接着是几个残差块，残差块之中使用的是两层步长为 1，卷积核为 3×3 的卷积层，因此不改变图像大小，最后是相应的反卷积层，将图像上采样，从而得到和输入图像I同等大小的输出图像O。值得一提的是本文的残差块和文献[11]中采用的不一样，每次残差操作不会丢失 4 像素的图像大小，因此不用对输入图像进行 padding 操作，一定程度上也避免了边缘的模式崩溃。整个前馈神经网络的参数以及相应的特征图大小如表 3.2 所示：

表 3.2　前馈神经网络的结构以及特征图大小

网络层	特征图大小
输入层	3×256×256
32×9×9 卷积层，stride 步长 1	32×256×256
64×3×3 卷积层，stride 步长 2	64×128×128
128×3×3 卷积层，stride 步长 2	128×64×64
残差层，128 个卷积核	128×64×64
残差层，128 个卷积核	128×64×64
残差层，128 个卷积核	128×64×64
残差层，128 个卷积核	128×64×64
残差层，128 个卷积核	128×64×64
64×3×3 卷积层，stride 步长 1/2	64×128×128
32×3×3 卷积层，stride 步长 1/2	32×256×256
3×9×9 卷积层，stride 步长 1	3×256×256

3.2.2　损失函数及训练

设内容图像C和前馈神经网络的输出图像O在 VGG19 网络的第l层的特征分别为C^l和O^l，其本质是一个大小为$N_l \times H_l \times W_l$的特征图，其中$N_l$，$H_l$，$W_l$分别是特征图的通道个数（也就是卷积核的个数），高度和宽度大小，定义内容特征损失为两图像在 relu4_2 层的特征图的欧氏距离：

$$L_{content}(O,C,l) = 1/(N_l H_l W_l)\|O^l - C^l\|_2^2 \tag{3.7}$$

风格特征损失约束图像O保留风格图像S的颜色，纹理以及图案模式。采用 Gram 矩阵表达对风格的刻画，Gram 矩阵的定义已在 3.1 节介绍过，此处不再赘述，第l层的 Gram 矩阵G^l大小为$N_l \times N_l$，定义总的风格特征损失为多层 Gram 矩阵的欧式距离和：

$$L_{style}(O,S) = \sum_{l=0}^{L}(1/(N_l H_l W_l))\|G^l[O] - G^l[S]\|_2^2 \tag{3.8}$$

具体在训练中采用 relu1_1，relu2_1，relu3_1，relu4_1，relu5_1 这几层 VGG19 网络层进行风格损失的计算，并且由于 Gram 矩阵的大小只和特征图的通道个数有关，和具

体的输入图像大小无关，故即使不一样尺寸大小的图像也能够很好的并行计算 Gram 矩阵的值。值得一提的是尽管 Gram 矩阵在图像风格转化中被频繁使用，但是很少有人解释原因，文献[21]的工作发现匹配两张图像的 Gram 矩阵，其实数学上严格等价于极小化这两张图像卷积特征图的二维 poly kernel 的 MMD 距离。其中，MMD 距离是用来通过从两个分布中采样的样本来衡量两个分布之间的差异的一种度量。所以本质上，风格特征损失做的事情就是将输出图像O的卷积特征图分布和风格图像S的分布进行匹配，从而达到保留风格样式的目的。

最后，算法同样采用全变分正则化项，对前馈神经网络产生的输出图像O进行降噪处理。算法最终最小化的目标和慢速艺术绘画风格转化算法的类似，即由内容损失，风格损失和 TV 损失加权加和在一起组成。

算法中使用的内容，风格以及 TV 损失的权重参数分别为 e0，e2，e2 这三种数量级，具体的值可以根据情况调整。算法采用 Adam 优化方法进行迭代优化。为每个风格图像训练一个图像转化模型（即前馈神经网络），模型训练时输入一个特定的风格图像S，用于产生风格特征。采用 Microsoft COCO dataset 图像数据集，具体为 train2017 数据集作为训练数据集，也就是内容图像，将训练图像都缩放尺寸为 3×256×256，然后输入到前馈神经网络中，产生相应的输出图像O，利用O和S计算损失大小。整个算法在 train2017 数据集上进行两个 epoch，每次输入的图像批次大小（batchsize）为 4，学习速率为 0.001，由于只进行了两个 epoch，不会产生过拟合现象，因此网络中不使用 dropout 和 weight-decay 等防止过拟合技巧，最后将训练完成的前馈神经网络模型保存。由于前馈神经网络是全卷积的，因此测试阶段可以使用该模型将任意大小的输入图像转化为风格图像S的风格样式。训练过的前馈神经网络可以意识到图像的语义内容，一种解释为 VGG19 网络本身经过训练就已经有对人和动物等物体的特定特征了，而这些对象（比如人和动物）在风格转化的训练集中就存在，因此最后得到的前馈神经网络也就保持了对人和动物等物体的对象特征。下面给出算法伪代码：

表 3.3　快速艺术绘画风格转化算法流程

算法 3.2　快速艺术绘画风格转化算法	
输入：训练图像集（也就是内容图像C的集合）$Data$，风格图像S，设定权重参数α，β和γ，设定总迭代次数max_epoch，设定学习速率r，设定批量大小$batch_size = 4$，初始化迭代次数$t = 0$	
输出：风格化之后的输出图像O，训练完成之后的前馈神经网络Net	
1	首先计算风格图像S在 relu1_1，relu2_1，relu3_1，relu4_1，relu5_1 层的 Gram 矩阵$G^l[S]$
2	while 不满足迭代停止条件：
3	从$Data$中选择$batch_size$个内容图像C，计算内容图像C在 relu4_2 层的特征表示C^{relu4_2}
4	将内容图像C输入前馈神经网络中产生输出图像O：
5	$$O = Net(C)$$
6	计算输出图像O在 relu4_2 层的特征表示O^{relu4_2}，分别计算输出图像O在 relu1_1，relu2_1，relu3_1，relu4_1，relu5_1 层的 Gram 矩阵$G^l[O]$
7	计算损失函数值：
8	$$L_{total}(O,C,S) = \alpha L_{content}(O,C) + \beta L_{style}(O,S) + \gamma L_{TV}(O)$$
9	利用 Adam 优化器优化算法：
10	$$Adam(L_{total}(O,C,S))$$
11	更新前馈神经网络Net的权值参数：
12	$$Update(W_{Net})$$
13	更新迭代次数：$t = t + 1$
14	end while

3.2.3　快速算法实验结果

算法使用 TensorFlow 深度学习框架实现。单个模型的训练在单个 NVIDIA GeForce 1080 GPU 上进行约 7 个小时，保存最后的训练模型。实验得到与已有的算法类似的效果图，下面展示一些实验结果。

图 3.5 是单个模型训练期间的损失函数变化，包括风格损失，TV 损失，内容损失和总损失。从曲线图中可以看出和慢速转化算法相比，快速算法损失值的抖动非常严重，尤其是内容损失值最为剧烈。产生抖动的原因是模型训练期间每次迭代使用不一样的内容图像，故产生不一样的内容特征，而不像慢速算法中内容图像是固定的，内容特征也

是固定的。同时快速算法优化的是前馈神经网络中的参数，不是直接优化输出图像的像素值，由于参数数量多，敏感度高，因此优化起来比较困难，容易产生抖动。随着迭代次数的增加，虽然抖动一直都有，但是大体上是不断下降的。

图 3.6 是算法运行的输出图像。其中左上角的图像为内容图像，也就是输入到前馈神经网络中的输入图像，下方为不同的风格图像和相应的输出图像。快速转化算法为每个风格图像单独训练一个模型，因此上述实验共训练了 7 个模型。由输出图像可以看出模型成功地将内容图像风格化了，而且效果非常明显，和慢速算法相比，快速算法产生的输出图像的风格化力度更大，甚至有些输出图像的内容被严重弱化，而且天空出现了很明显的风格纹理。所以快速算法产生的输出图像更具艺术特色。

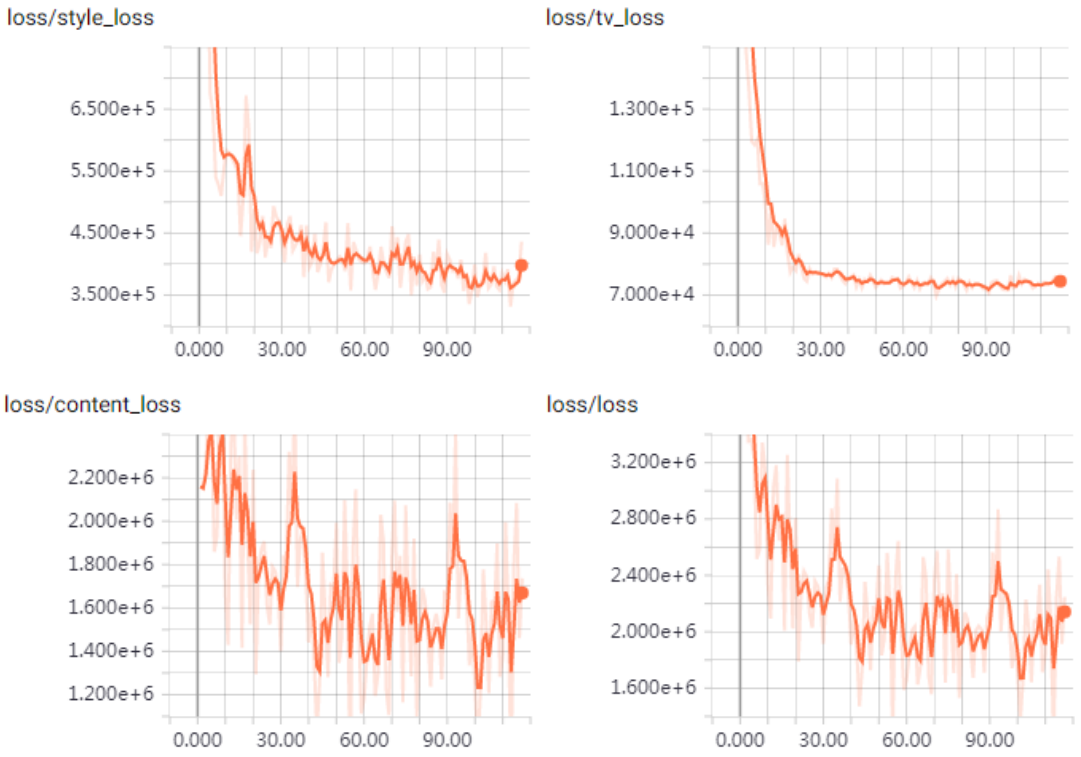

图 3.5　训练损失函数

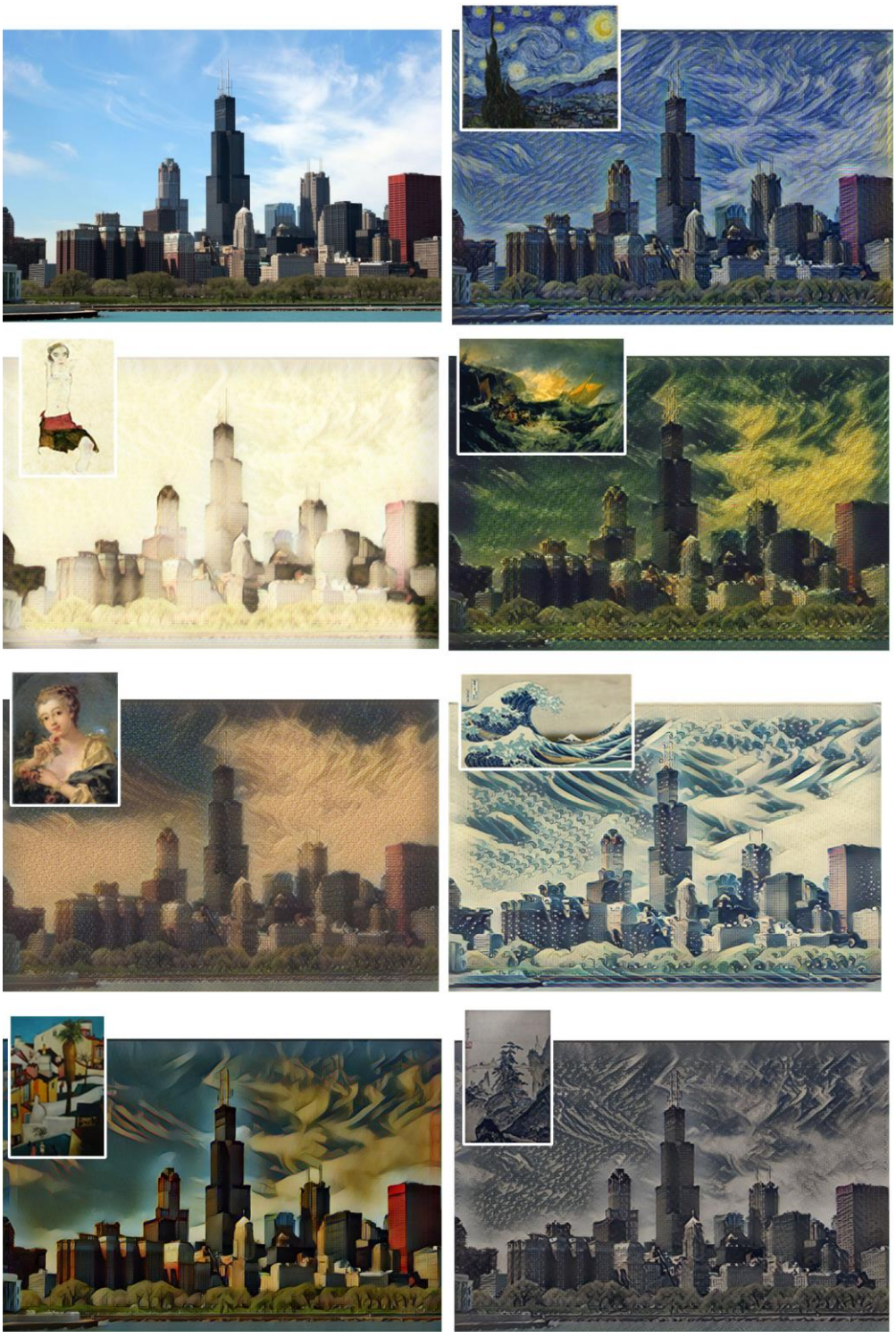

图 3.6　风格图和相应的生成图像

4　真实图像风格转化

4.1　慢速转化算法

上文提到的算法都是针对绘画的图像风格转化，每次生成的结果虽然既有内容图像的内容样式，也有风格图像的风格样式，但是图像会严重扭曲，就算风格图像和内容图像都是真实照片，生成的图像依然会变成扭曲的绘画。本节为了解决真实图像（照片）的风格转化问题，保持图像的真实性，参考文献[15]实现慢速真实图像风格转化算法。

图 4.1 展示了用慢速艺术绘画风格转化算法对真实照片进行转化的结果。其中上方是风格图像和内容图像，下方是风格化之后的输出图像。可以看出，尽管风格图像和内容图像都是真实照片，算法产生的输出图像仍然会被扭曲，形成绘画般的质感，最明显的是天空和建筑部分，天空中出现了纹理的横向扭曲，建筑中的窗户和边缘都有不同程度的形状扭曲，窗户之间的界限不清晰。

图 4.1　被扭曲的真实图像

为了抑制图像的扭曲，将从输出图像（也就是生成的图像）O到风格图像S的变换约束在色彩空间的局部仿射变换中，只变换颜色，抑制形状的变化，并且将这个约束利用

拉普拉斯抠图矩阵表示成一个完全可微的惩罚项，在实现的时候可以直接计算该项的梯度，并添加到反向传播的梯度流中，作为仿射变换惩罚。导致图像失真的另一个原因是变换过程中由于输出图像O和风格图像S的内容不同而导致的不相关内容不在预期区域内的变换，比如，内容图像C的天空区域比较少，而风格图像S的天空区域比较多，风格变换可能会忽略掉内容上的差异而导致风格图像S的天空区域“溢出”到输出图像O的其他部分。为了解决这个问题，算法采用图像的语义分割技术，对输出图像O和风格图像S进行语义分割，得到图像的语义掩模（Mask），然后限制图像的风格转化只在有相同语义标签的区域之间进行，比如风格图像S中的天空和输出图像O中的天空之间的转化，S中的河流和O中的河流之间的转化，S中的车辆和O中的车辆之间的转化等。有了这两项约束，通过迭代优化算法，就可以实现输出图像O的真实图像风格转化，并且生成的图像具有高保真的特点。下面详细介绍一下损失函数的计算。

4.1.1 内容损失

设输出图像O和内容图像C在 VGG19 网络的第l层的特征表示分别为O^l和C^l，其本质是一个大小为$N_l \times H_l \times W_l$的特征图，其中$N_l$，$H_l$，$W_l$分别是特征图的通道个数，高度和宽度大小，那么内容损失可以表示为：

$$L_{content}(O, C, l) = 1/(2N_l H_l W_l)\|O^l - C^l\|_2^2 \tag{4.1}$$

实验中算法使用 conv4_2 层作为内容特征的计算层。上文算法使用的是 relu4_2 层，relu4_2 层是将 conv4_2 层经过 relu 函数激活之后得到的。在实验中，发现采用未经过激活函数的层作为损失函数的计算层，其效果要优于经过激活之后的层，因此慢速真实图像风格转化算法的内容损失和风格损失都采用未经过激活的 VGG19 层来计算。

4.1.2 风格损失

为了抑制输出图像O和风格图像S中不相关内容不在预期区域内的变换，采取语义分割增强风格损失函数。艺术绘画风格算法的做法是，利用输出图像和风格图像的 VGG19 特征计算 Gram 矩阵，该 Gram 矩阵隐式地对特征的分布进行编码，然后采用欧式距离和作为风格损失。但是这种计算方法限制了 Gram 矩阵适应语义内容变化的能力，对整幅图像计算 Gram 矩阵，使其无法包含不同语义的对应关系，不加约束地进行风格转化，导致区域“溢出”。为了约束转化在对应语义的区域之间进行，采用文献[22]的全卷积语义分割网络，产生内容图像C和风格图像S的语义标签，比如：建筑，天空，河流，人像和窗户等，设产生的语义分割掩模为$A[C]$和$A[S]$，在计算 Gram 矩阵之前，将掩模和特征表示相乘，得到掩模之后的特征表示，利用掩模之后的特征表示计算 Gram 矩阵，最后

采用欧氏距离和计算输出图像和风格图像的 Gram 矩阵的距离，并作为风格损失。设O为输出图像（也就是生成的图像），S为风格图像，C为内容图像，且$F^l[O]$和$F^l[S]$分别是输出图像和风格图像在 VGG19 网络第l层的特征表示，其实质是大小为$N_l \times H_l \times W_l$的特征图，设$A^{l,k}[C]$和$A^{l,k}[S]$分别是内容图像$C$和风格图像$S$在 VGG19 网络第$l$层第$k$个通道的掩模，在 VGG19 卷积神经网络的每一层，利用平均池化对原始掩模$A[C]$和$A[S]$进行下采样操作，得到与特征表示$F^l[O]$和$F^l[S]$相同尺寸的$A^{l,k}[C]$和$A^{l,k}[S]$，然后将下采样掩模和特征图进行逐元素相乘，得到掩模之后的特征表示：

$$F^{l,k}[O] = F^l[O]A^{l,k}[C], F^{l,k}[S] = F^l[S]A^{l,k}[S] \tag{4.2}$$

接着利用向量化之后的$F^{l,k}[O]$和$F^{l,k}[S]$计算对应k通道的 Gram 矩阵，Gram 矩阵的大小为$N_l \times N_l$：

$$G^{l,k}[O] = F^{l,k}[O]F^{l,k}[O], G^{l,k}[S] = F^{l,k}[S]F^{l,k}[S] \tag{4.3}$$

有了输出图像和风格图像各自l层k通道的 Gram 矩阵，利用 Gram 矩阵的欧氏距离作为损失函数，将掩模中所有通道的 Gram 矩阵相加，就得到l层的风格损失：

$$E_l^+ = \sum_{k=1}^{K} 1/(2N_l^2) \sum_{i,j} (G^{l,k}[O]_{i,j} - G^{l,k}[S]_{i,j})^2 \tag{4.4}$$

其中K表示掩模通道的总个数。所有风格特征层的整体损失就可以写为：

$$L_{style}^+(O,S) = \sum_{l=0}^{L} w_l E_l^+ \tag{4.5}$$

其中w_l是每层对总的风格损失贡献的权重因子，一般设置为$1/L$即可，L表示风格特征所用的卷积网络的层数。具体实现中使用 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 这几个 VGG19 网络层进行风格损失的计算，每层需要将掩模的所有通道加和，语义分割网络能够产生 151 个类别的掩模，为了加快计算速度，实际操作中将 151 个类分成了 10 组，共产生 10 个掩模通道，也就是$K = 10$。

4.1.3 仿射变换惩罚

为了抑制图像的扭曲，约束图像风格转化只在颜色空间进行，算法采用仿射变换惩罚项，这个惩罚项直接作用在输出图像O上。首先，应该认识到这样一个事实，内容图像C本身就是真实照片，输出图像O的目的是保持内容图像的内容，同时拥有风格图像的风格，在转化的过程中由于算法的缺陷，导致内容图像的形状发生了扭曲，造成了不真实的感觉，因此，要抑制图像的扭曲，最简单的方法就是添加一个正则化惩罚项，约束内容在转化的过程中不被扭曲即可。直觉上，可以考虑使用边缘检测区块。RGB 通道的仿射合并会生成一组变量，但边缘不会移动，因为它在所有通道上都在相同的位置。文

献[23]的拉普拉斯抠图算法实现了如何利用颜色空间进行抠图，论文在内容图像C的 RGB 通道的局部仿射组合上构建拉普拉斯抠图算法。算法的关键在于矩阵M_C（可称为 Matting 矩阵），M_C刻画了图像的颜色边缘信息，且这个矩阵只依赖于内容图像C（具体可以参考论文的细节，注意给定一个内容图像C有N个像素，M_C的大小是$N \times N$），这里将输出图像O在通道f的向量化格式表示为$V_f[O]$，大小为$N \times 1$，给出下面的仿射变换惩罚项约束输出图像O只在颜色空间改变：

$$P_{affine}(O,C) = 2M_C V_f[O] \tag{4.6}$$

在算法实现的时候，每次迭代优化直接计算上面惩罚项的值，然后添加到优化器的梯度序列中，利用得到的总梯度进行反向传播计算像素更新量即可。

4.1.4 风格转化

真实图像风格转化算法同样采用 TV 损失来防止图像的失真，对输出图像O进行简单的修复。所以总损失函数可以定义为：

$$L_{total}(O,C,S) = \alpha L_{content}(O,C) + \beta L^{+}_{style}(O,S) + \gamma L_{TV}(O) + \varphi P_{affine}(O,C) \tag{4.7}$$

注意，上式中的仿射变换惩罚项只是为了公式的统一，正如上文提到的，在实际操作中将直接计算惩罚项的值，添加到损失函数的梯度流中，然后利用得到的总梯度进行优化即可。α，β，γ，φ分别是各项的权重系数，本文在实验时分别设置为 e0，e2，e-3，e4 这四种数量级。输出图像O的初始化和慢速艺术绘画风格转化算法的类似，不采用单纯的随机噪声，而是随机噪声和内容图像C的加权混合，具体为噪声 0.6，内容图像 0.4。使用的语义分割网络[22]采用数据集 MIT Scene Parsing 进行训练，该数据集共包括 151 个语义类别，语义分割网络同样是全卷积的，即训练完成之后，输入任意大小的图像，可以得到相同大小的掩模图像，掩模图像是灰度图，只有一个通道，每个位置的像素值为 0 到 150 之间，表示 151 个类别。本文直接使用 GitHub 上 Tianxiao Zhao 和 easyt0re 的工作（https://github.com/DeepSegment/FCN-GoogLeNet），简单修改之后作为掩模生成器。在运行慢速真实图像风格转化算法之前，将输入算法的内容图像C和风格图像S先经过掩模生成器生成相应的掩模$A[C]$和$A[S]$，然后将内容图像，风格图像连同他们各自的掩模一起输入风格转化算法，最后进行迭代更新完成真实图像的风格转化，得出与文献[15]类似的效果图。但是具体实现中有一些和文献[15]的工作不同的地方，首先文献[15]提供的算法没有掩模生成器，只提供了有限的几个图像掩模，本节将掩模生成器涵盖在整个算法流程中，因此实现了端到端的真实风格转化，尽管算法的性能和掩模的质量有很大关系，但实验发现，采用自动的掩模生成器完全可以生成与文献[15]类似的效果图。其次，文献

[15]的工作使用了 CUDA 图像处理库中光滑局部仿射（Smooth Local Affine）功能对结果图进行光滑处理，而本节经过对风格损失函数的细微调整，在不使用光滑局部仿射的情况下，达到了和文献[15]类似的效果。下面给出算法伪代码：

表 4.1　慢速真实图像风格转化算法流程

算法 4.1　慢速真实图像风格转化算法
输入：内容图像C，风格图像S，设定权重参数α，β，γ，φ，设定总迭代次数max_$iteration$，设定学习速率r，初始化迭代次数$t=0$
输出：风格化之后的图像O
利用掩模生成器 Mask 生成内容图像C和风格图像S的掩模：
$$A[C]=Mask(C),\ A[S]=Mask(S)$$
计算内容图像C的 Matting 矩阵：
$$M_C$$
计算内容图像C在 conv4_2 层的特征表示C^{conv4_2}，结合掩模计算风格图像S在 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 层的 Gram 矩阵$G^l[S]$
while 不满足迭代停止条件：
计算输出图像O在 conv4_2 层的特征表示O^{conv4_2}，结合掩模计算输出图像O在 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 层的 Gram 矩阵$G^l[O]$
计算除了仿射变换惩罚项之外的总损失函数值：
$$L_{total}(O,C,S)^- = \alpha L_{content}(O,C)+\beta L^+_{style}(O,S)+\gamma L_{TV}(O)$$
利用 Adam 优化器计算$L_{total}(O,C,S)^-$关于输出图像O的梯度：
$$Grad^- = Gradient_{Adam}(L_{total}(O,C,S)^-)$$
直接计算仿射变换惩罚项的值并添加到总梯度中：
$$Grad = Grad^- + \varphi P_{affine}(O,C)$$
利用总梯度更新输出图像O的像素值：
$$Update(O_{i,j})$$
更新迭代次数：$t=t+1$
end while

4.1.5 慢速算法实验结果

算法使用 TensorFlow 深度学习框架实现，使用 Adam 优化方法进行迭代优化，学习速率设置为 1，在单个 NVIDIA GeForce 1080 GPU 上运行 1 小时 30 分钟左右，共迭代 8000 次。下面展示一些算法的结果。

图 4.2 是算法在迭代优化过程中损失函数的变化情况，其中包括内容损失，风格损失，TV 损失和总损失，但不包括仿射变换惩罚项。因为在算法实现时会直接计算仿射变换惩罚项并添加到总梯度流中，然后利用梯度改变输出图像的像素值，使总损失 VGGNetLoss 持续减小。另外，慢速真实图像风格转化算法的损失函数曲线相对平稳，不会出现快速算法中的剧烈抖动，因为算法的内容图像和风格图像在每一轮迭代中都是不变的，每轮优化都只需要计算梯度方向然后进行梯度下降即可，而快速算法的内容图像在每一轮迭代中都是不一样的，重新计算的内容图像特征导致损失函数的剧烈抖动，但是总损失大体上依然是不断下降的。

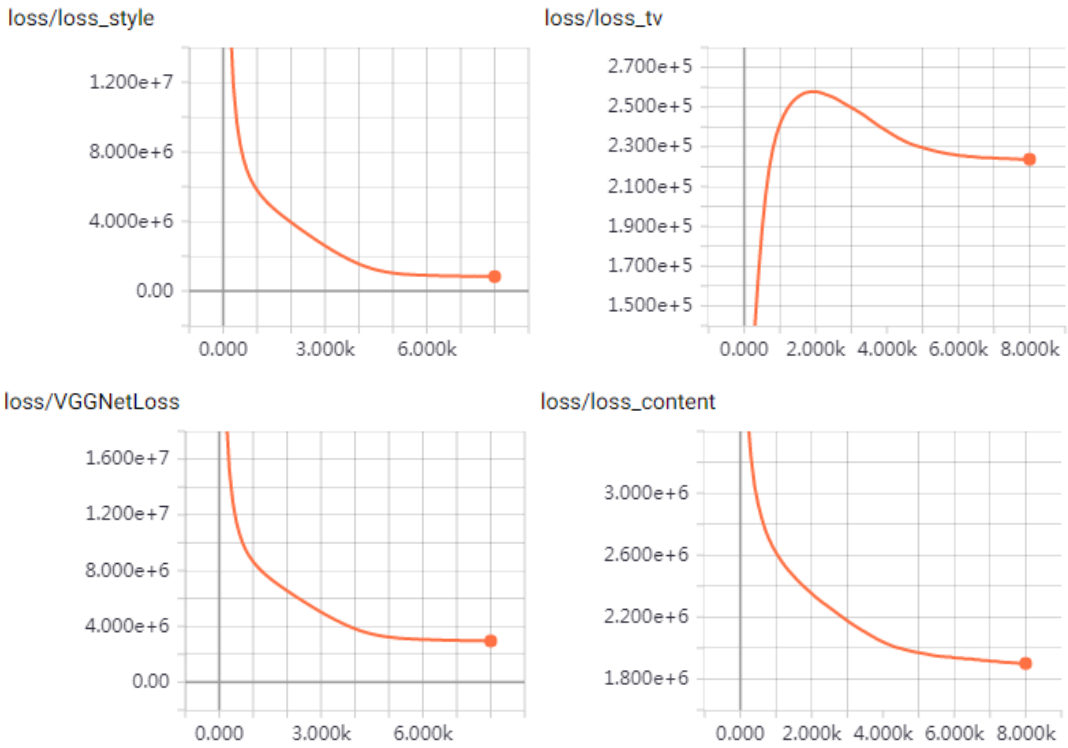

图 4.2　训练损失函数

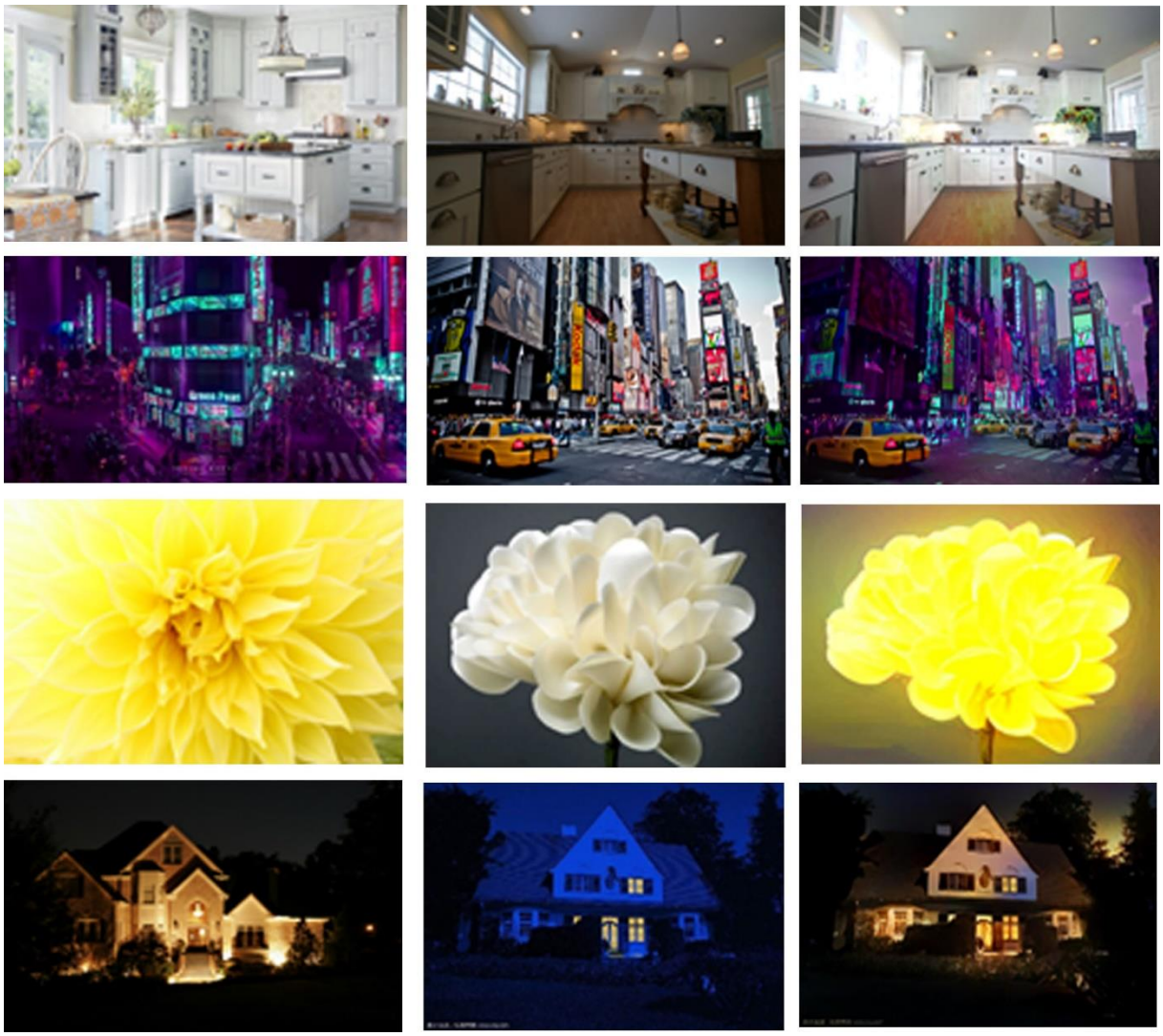

图 4.3　风格图和相应的生成图像

图 4.3 是算法使用不同风格图像的运行结果。共分为四组，每组第一个图像为风格图像，后面图像依次为内容图像和输出图像，可以看出利用慢速真实图像风格转化算法最大限度地保留了图像的真实感，同时将风格图像的风格转移到了内容图像上。以第一组实验为例子，风格图像是一张明亮的室内照片，经过算法的运行，成功地将下面两张光线很暗的室内照片转化为了明亮的室内照片，同时保持图像的逼真性，将房间的窗户明亮化，通过增强光线效果，到达风格化的目的，自然美观，没有出现图 4.1 中的窗户以及边缘扭曲现象，图像看起来具有照片质感。

4.2 快速转化算法

与慢速艺术绘画风格转化算法类似，慢速真实图像风格转化算法每次进行风格转化时，都需要进行一个完整的优化训练过程，不断改变生成图像O的像素值，进行 8000 次左右的迭代之后，才能得到最终的结果，因此速度慢。为了提高图像生成的速度，本节尝试提出快速真实图像风格转化算法。并且慢速真实图像风格转化算法的仿射变换惩罚项的可解释性差，本节简单修改仿射变换惩罚项使其适应快速算法，并提高该项的可解释性。算法基本思想是，结合快速艺术绘画风格转化算法的前馈神经网络，将计算代价放在训练时期，将输入图像I通过前馈神经网络的转化，得到输出图像O，然后参考慢速真实图像风格转化算法的损失函数，构建对输出图像O和内容图像C的内容损失，输出图像O和风格图像S的风格损失，内容图像C对输出图像O的仿射变换约束损失以及最后的 TV 损失，利用损失函数和优化算法，对前馈神经网络中的参数进行梯度下降优化，最后得到能够以风格图像S的样式风格化任意输入图像I的前馈神经网络，并将网络以模型的形式保存，一旦保存完毕，以后每次风格转化只需要在前馈神经网络中进行一次前向传播即可。由实验结果可知，快速真实图像风格转化算法将速度提高了三个数量级，但是生成图像O的效果不好，这将是下一步工作的重点。

4.2.1 前馈神经网络结构

与快速艺术绘画风格转化算法类似，这里也采用类似的前馈神经网络结构，即网络中不使用池化层和全连接层，是全卷积的，输入图像I先在卷积层中进行下采样，然后通过 5 个残差块，每个残差块由两层卷积层和一个捷径连接组成，而且残差块操作不改变图像的大小，最后由反卷积层对图像上采样，输出一个与输入图像I同等大小的输出图像O，具体的网络结构可以参见快速艺术绘画风格转化算法。特别的，在实验中发现生成的图像边缘经常有明显的模式崩坏现象，而且 TV 损失不易下降，生成图像O得不到有效的修复，因此网络训练阶段，在前馈神经网络的最后增加了一层，该层的作用是将网络输出的像素值和原始的输入图像I的像素值进行加权混合，设前馈神经网络最后层的输出像素值为x^o，输入前馈神经网络的输入图像I的像素值为x^i，增加的层可以表示为：

$$x^{final} = \alpha x^o + (1-\alpha)x^i \tag{4.8}$$

其中α为混合的权重，实验中设$\alpha = 0.85$，将最后的x^{final}作为训练阶段前馈神经网络的输出，即输出图像O。当模型训练结束，进行测试的时候，将增加的加权混合层去掉，直接使用x^o作为输出图像O，实验证明，这样做有效地降低了 TV 损失值，一定程度上抑制了生成图像O边缘崩坏的现象。和快速艺术绘画风格转化算法类似，在训练阶

段，将输入图像*I*都重缩放为 3×256×256 的彩色图片，由于网络是全卷积的，在测试阶段，允许输入图像*I*为任意大小。下面给出网络定义的核心代码：

```
import tensorflow as tf
from ImageTransfer.fast_art import transfer_ops as ops
# transfer net define
def net(image, name='net'):
    """
    :param image: the original image with scaled [0, 1]
    :param name: the name scope name
    :return: the transfered image with scaled [0, 255]
    """
    with tf.name_scope(name):
        conv1 = ops.conv_layer(image, 32, 9, 1, 'conv1')
        conv2 = ops.conv_layer(conv1, 64, 3, 2, 'conv2')
        conv3 = ops.conv_layer(conv2, 128, 3, 2, 'conv3')
        res_block1 = ops.residual_block(conv3, 'res_block1', 128, 3, 1)
        res_block2 = ops.residual_block(res_block1, 'res_block2', 128, 3, 1)
        res_block3 = ops.residual_block(res_block2, 'res_block3', 128, 3, 1)
        res_block4 = ops.residual_block(res_block3, 'res_block4', 128, 3, 1)
        res_block5 = ops.residual_block(res_block4, 'res_block5', 128, 3, 1)
        deconv1 = ops.conv_tranpose_layer(res_block5, 64, 3, 2, 'deconv1')
        deconv2 = ops.conv_tranpose_layer(deconv1, 32, 3, 2, 'deconv2')
        conv4 = ops.conv_layer(deconv2, 3, 9, 1, 'conv4', instance_norm=True, relu=False)
        pre_output = (tf.nn.tanh(conv4) + 1) * 127.5 # [0-255]
        output = pre_output * 0.85 + image * 255.0 * 0.15
        return output
```

4.2.2 损失函数及训练

损失函数同样分为四部分：内容损失，风格损失，仿射变换损失和 TV 损失。其中内容损失、风格损失和 TV 损失与慢速真实图像风格转化算法基本一致，具体可参考第 4.1 节的内容。不同点在于，这里的内容损失、风格损失和 TV 损失都是向量化的（Vectorized），即支持数组编程（Array Programming），这样的好处是当使用批梯度下降算法优化训练时，可以大大提高训练速度，节省训练时间。4.1 节的损失函数是非向

量化的，因此这里需要进行改写。为了抑制风格化过程中图像形变的产生，需要将风格化限制在颜色空间，参考文献[15]的工作，利用文献[23]的拉普拉斯抠图算法产生的 Matting 矩阵，完成仿射变换损失的构建。需要注意的是，不能直接将文献[15]的仿射变换损失迁移过来，因为文献[15]中需要优化的参数实际上就是输出图像O的像素值，因此可以针对输出图像O的像素值单独计算仿射变换惩罚，并添加到梯度流中，反向传播的时候，可以利用总的梯度值直接对输出图像O的像素值更改，进而一步一步风格化图像。快速真实图像风格转化算法中需要优化的参数不是输出图像O的像素值，而是定义在前馈神经网络中的参数，因此无法直接对输出图像O的像素值求梯度，且直接对前馈神经网络中参数求梯度复杂度过高，故需要修改仿射变换损失，使其适应对网络参数的优化。设内容图像为C，且有N个像素，按照文献[7]的方法计算内容图像C的 Matting 矩阵M_C，且M_C的大小是$N \times N$，这里将输出图像O在通道f的向量化格式表示为$V_f[O]$，大小为$N \times 1$，将内容图像C在通道f的向量化格式表示为$V_f[C]$，大小也为$N \times 1$，并且定义下面的仿射变换损失：

$$L^{+}_{affine}(O,C) = (1/N)\sum_{f=0}^{3}\left\|M_C V_f[O] - M_C V_f[C]\right\|_2^2 \qquad (4.9)$$

由于M_C包含了内容图像的颜色空间约束，为了抑制形变，应该使输出图像O和内容图像C的颜色约束尽量一致，而直接利用内容图像C和输出图像O的 Matting 矩阵作为欧氏距离损失是不合理的，因为算法前期输出图像O的像素几乎可以说是随机生成的，在随机像素上计算 Matting 矩阵没有意义，故使用内容图像C的 Matting 矩阵M_C和图像的矩阵乘积作为欧氏距离损失，这就是上述仿射变换的含义。

最后总的损失可以定义为：

$$L_{total}(O,C,S) = \alpha L_{content}(O,C) + \beta L^{+}_{style}(O,S) + \gamma L_{TV}(O) + \varphi L^{+}_{affine}(O,C) \qquad (4.10)$$

其中α，β，γ，φ分别是各个损失项的权重系数，本文在实验时使用的系数分别为 e1，e2，e-3，e4 这四种数量级。训练期间，将所有图像都缩放为 3×256×256，训练完成后模型可以用于任意大小图像的风格转化。关于训练，还有几点需要说明。首先，仿射变换损失在每次迭代的时候都需要重新计算新的 Matting 矩阵，且 Matting 矩阵的计算是相对耗时的，导致算法的训练速度非常缓慢，这也是之后改进工作的重点。其次，算法不能继续采用 Microsoft COCO dataset 图像数据集作为训练数据集（也就是内容图像C），因为快速真实图像风格转化算法与慢速真实图像转化算法一样，需要计算图像的掩模，根据掩模的语义类别定义风格损失，这就要求风格图像S和内容图像C尽量拥有相同的掩模语义类别，而 Microsoft COCO dataset 图像数据集不满足这个要求，故不适合

作为训练数据，实现中利用简单的 Python 爬虫爬取百度图库的图片，搜索关键字可以为风格图像S的语义类别，为每个风格图像S爬取约 12000 张相应的内容图像，作为训练数据集。每个模型在训练数据集上进行 3 个 epoch，利用学习速率为 0.001 的 Adam 算法进行优化。这是对快速真实图像风格转化算法的第一次尝试，从实验结果来看，生成的图像效果并不理想，还是有明显的形变痕迹。下面给出损失函数定义的核心代码：

```
def content_loss(const_layer, var_layer, weight):
    b, h, w, c = [i.value for i in var_layer.shape]
    loss = weight * (2 * tf.nn.l2_loss(var_layer - const_layer) / tf.to_float(b * h * w * c))
    return loss

def cal_style_loss(CNN_structure, const_layers, var_layers, content_segs, style_segs_para,
weight):
    loss_styles = []
    layer_index = 0
    style_segs = style_segs_para
    _, content_seg_height, content_seg_width, _ = content_segs[0].shape
    _, style_seg_height, style_seg_width, _ = style_segs[0].shape
    for layer_name in CNN_structure:
        layer_name = layer_name[layer_name.find("/") + 1:]
        # down sampling segmentation
        if "pool" in layer_name:
            content_seg_width,content_seg_height=int(math.ceil(content_seg_width/2)),in
            t(math.ceil(content_seg_height/2))
            style_seg_width, style_seg_height = int(math.ceil(style_seg_width /
            2)),int(math.ceil(style_seg_height / 2))
            for i in range(len(content_segs)):
                content_segs[i] = tf.image.resize_bilinear(content_segs[i],
                tf.constant((content_seg_height, content_seg_width)))

                style_segs[i] = tf.image.resize_bilinear(style_segs[i],
                tf.constant((style_seg_height, style_seg_width)))
        elif "conv" in layer_name:
            for i in range(len(content_segs)):
```

```
                content_segs[i] = tf.nn.avg_pool(tf.pad(content_segs[i], [[0, 0], [1, 1], [1,
                1], [0, 0]], "CONSTANT"), ksize=[1, 3, 3, 1], strides=[1, 1, 1, 1],
                padding='VALID')

                style_segs[i] = tf.nn.avg_pool(tf.pad(style_segs[i], [[0, 0], [1, 1], [1, 1],
                [0, 0]], "CONSTANT"), ksize=[1, 3, 3, 1], strides=[1, 1, 1, 1],
                padding='VALID')
        layer_name = layer_name[:layer_name.find("/")]
        if layer_name in var_layers[layer_index].name[var_layers[layer_index] .name.
        find("/") + 1:]:
            print("Setting up style layer: <{}>".format(layer_name))
            const_layer = const_layers[layer_index]
            var_layer = var_layers[layer_index]
            layer_index = layer_index + 1
            layer_style_loss = 0.0
            for content_seg, style_seg in zip(content_segs, style_segs):
                gram_matrix_const = gram_matrix(tf.multiply(const_layer, style_seg))
                gram_matrix_var = gram_matrix(tf.multiply(var_layer, content_seg))
                content_mask_mean = tf.reduce_mean(content_seg)
                b, h, w = [i.value for i in gram_matrix_var.shape]
                diff_style_sum = 2 * tf.nn.l2_loss(gram_matrix_var - gram_matrix_const)
                / tf.to_float(h * w * b) * content_mask_mean
                layer_style_loss += diff_style_sum
            loss_styles.append(layer_style_loss * weight)
    return loss_styles

def single_affine_loss(content_input_one, output_one, M, weight):
    affine_loss = 0.0
    for v_o, v_c in zip(tf.unstack(tf.squeeze(output_one), axis=-1),tf.unstack
    (tf.squeeze(content_input_one), axis=-1)):
        v_o = tf.reshape(v_o, [IMAGE_SIZE * IMAGE_SIZE])
        v_c = tf.reshape(v_c, [IMAGE_SIZE * IMAGE_SIZE])
        o_matrix = tf.sparse_tensor_dense_matmul(M, tf.expand_dims(v_o, -1))
        c_matrix = tf.sparse_tensor_dense_matmul(M, tf.expand_dims(v_c, -1))
        affine_loss += weight * 2 * tf.nn.l2_loss(o_matrix - c_matrix)
    affine_loss = affine_loss / tf.to_float(IMAGE_SIZE * IMAGE_SIZE)
```

```
    return affine_loss

def total_variation_loss(output, weight):
    tv_loss = tf.reduce_sum((output[:, :-1, :-1, :] - output[:, :-1, 1:, :]) *
                            (output[:, :-1, :-1, :] - output[:, :-1, 1:, :]) +
                            (output[:, :-1, :-1, :] - output[:, 1:, :-1, :]) *
                  (output[:, :-1, :-1, :] - output[:, 1:, :-1, :])) / tf.to_float(2.0 * BATCH_SIZE)
    return tv_loss * weight
```

下面给出训练过程的核心代码：

```
saver = tf.train.Saver()
with tf.name_scope('train'):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        # merge summary
        merged = tf.summary.merge_all()
        # choose dir
        writer = tf.summary.FileWriter(tf_board_path, sess.graph)
        for epoch in range(NUM_EPOCHS):
            num_examples = len(content_image_list)
            num_segs = len(segmentation_image_list)
            if num_examples != num_segs:
                print('num_examples!=num_segs')
                return
            iterations = 0
            start_time = time.time()
            while iterations * BATCH_SIZE < num_examples:
                curr = iterations * BATCH_SIZE
                x_batch = np.zeros(batch_shape, dtype=np.float32)
                seg_batch = np.zeros(seg_shape, dtype=np.float32)
                for j, img_p in enumerate(content_image_list[curr:curr + BATCH_SIZE]):
                    try:
                        x_batch[j] = utils.load_image_resize(img_p, height=
                        IMAGE_SIZE, width=IMAGE_SIZE)
                    except ValueError:
```

```
                print('read image error just ignore it', iterations)
            for j, img_seg in enumerate(segmentation_image_list[curr:curr
            + BATCH_SIZE]):
                try:
                    seg_batch[j] = np.array(utils.load_image_resize(img_seg, height
                    =IMAGE_SIZE, width=IMAGE_SIZE))
                except ValueError:
                    print('read seg error just ignore it', iterations)
            M_list = []
            for i in range(BATCH_SIZE):
                M = tf.to_float(getLaplacian(np.squeeze(x_batch[i])))
                M_list.append(M)
            content_masks_list = []
            content_masks_list = load_seg(seg_batch * 255.0)
            iterations += 1
            # optimize
            sess.run(train_op, feed_dict={content_input: x_batch})
    print('train done')
    # save sess
    saver.save(sess, save_path)
    print('save done')
```

下面给出算法伪代码：

表 4.2　快速真实图像风格转化算法流程

算法 4.2　快速真实图像风格转化算法	
输入：训练图像集（也就是内容图像C的集合）$Data$，风格图像S，设定权重参数α，β，γ和φ，设定总迭代次数max_epoch，设定学习速率r，设定批量大小$batch_size = 4$，初始化迭代次数$t = 0$	
输出：风格化之后的输出图像O，训练完成之后的前馈神经网络Net	
1	利用掩模生成器 Mask 生成所有训练图像集的掩模：
2	$$A[Data] = Mask(Data)$$
3	利用掩模生成器 Mask 生成风格图像S的掩模：
4	$$A[S] = Mask(S)$$
5	结合掩模$A[S]$计算风格图像S在 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 层的 Gram 矩阵$G^l[S]$
6	while 不满足迭代停止条件：
7	从$Data$中选择$batch_size$个内容图像C，计算内容图像C的 Matting 矩阵：
8	$$M_C$$
9	计算内容图像C在 conv4_2 层的特征表示C^{conv4_2}
10	将内容图像C输入前馈神经网络中产生输出图像O：
11	$$O = Net(C)$$
12	计算输出图像O在 conv4_2 层的特征表示O^{conv4_2}，从掩模$A[Data]$中选择出内容图像对应的掩模$A[C]$，并利用$A[C]$计算输出图像O在 conv1_1，conv2_1，conv3_1，conv4_1，conv5_1 层的 Gram 矩阵$G^l[O]$
13	计算损失函数值：
14	$$L_{total}(O,C,S) = \alpha L_{content}(O,C) + \beta L^{+}_{style}(O,S) + \gamma L_{TV}(O) + \varphi L^{+}_{affine}(O,C)$$
15	利用 Adam 优化器优化算法：
16	$$Adam(L_{total}(O,C,S))$$
17	更新前馈神经网络Net的权值参数：
18	$$Update(W_{Net})$$
19	更新迭代次数：$t = t + 1$
20	end while

4.2.3 快速算法实验结果

算法使用 TensorFlow 深度学习框架实现，在单个 NVIDIA GeForce 1080 GPU 上训练约 68 个小时，保存最后的训练模型。在模型上进行一次前向传播即可得到风格化之后的图像。下面展示一些实验结果。

图 4.4 是单一模型在训练期间的损失曲线。与快速艺术绘画风格转化算法一样，快速真实图像风格转化算法的内容损失曲线也是抖动最剧烈的，同时其余的几个损失值也存在不同程度的抖动，抖动的理由和快速艺术绘画风格转化算法的类似，这里不再赘述。值得注意的是，快速真实图像风格转化算法给出了仿射变换损失的曲线，因为算法优化的是前馈神经网络中的参数，不是直接优化输出图像的像素值，这和慢速真实图像风格转化算法是不同的，因此不用直接计算梯度。算法优化的目的是降低总损失 VGGNetLoss 的值，随着迭代次数的增加，总损失值整体是抖动下降的，最后逐渐收敛。

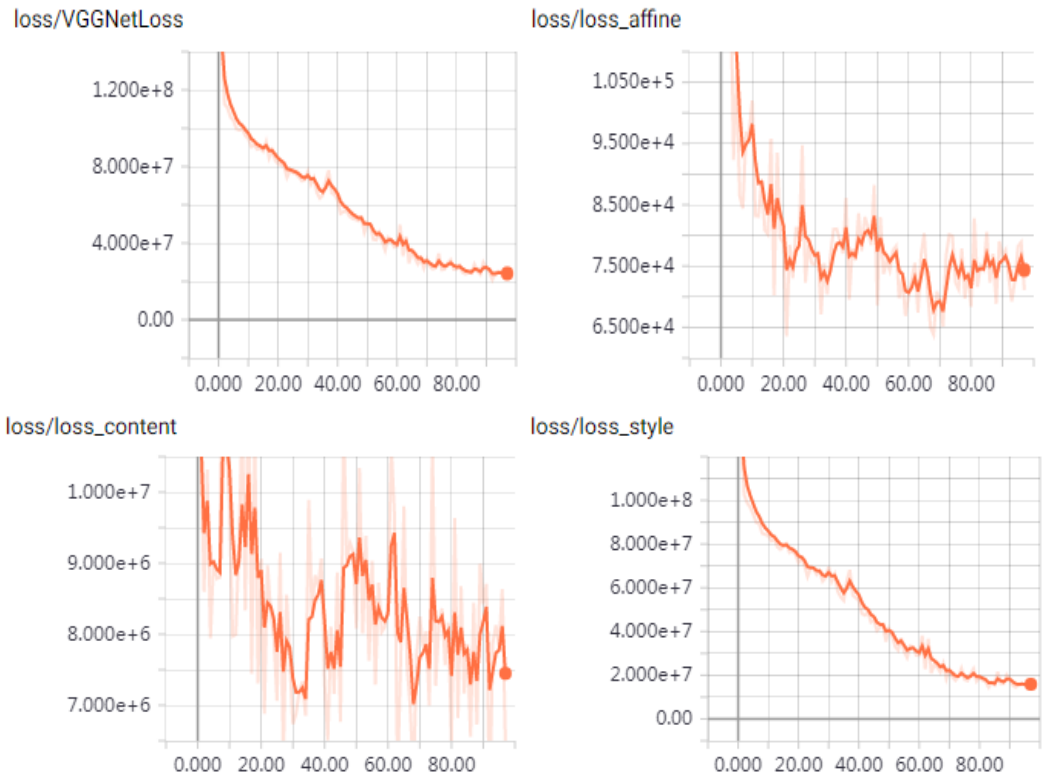

图 4.4 训练损失函数

图 4.5　风格图和相应的生成图像

图 4.5 是快速真实图像风格转化算法的实验结果图，共三组实验，分别对应三个不同的风格图像和三个模型。每组实验结果第一张为风格图像，后面的图像为内容图像和风格化之后的图像，可以看到算法将原始图像成功风格化，同时一定程度上保留了内容图像的内容，但是缺点同样很明显，输出图像的分辨率太低，图像细节没有得到保留，有的内容图像被严重风格化，产生的输出图像非常不自然，比如实验一中最后一张图像，整幅图都呈现出亮白色，这显然不符合真实照片的质感。

5 网站设计

为了实现端到端的算法应用，简单搭建一个网站平台，提供风格转化服务。网站功能主要有三个：在线风格转化服务，自定义风格转化服务，图片分享服务。在线风格转化服务提供已经训练好的快速艺术绘画风格转化模型和快速真实图像风格转化模型，支持用户上传特定的内容图像，然后根据用户选择的风格图像类型，为用户在线风格化内容图像，并将风格化之后的图像返回。自定义风格转化服务使用慢速艺术绘画风格转化算法和慢速真实图像风格转化算法实现，用户可以上传特定的内容图像和特定的风格图像，然后选择艺术绘画风格算法或者真实图像风格转化算法进行风格转化，由于算法的时间较长，故不能支持在线转化，而是在算法运行结束之后，向用户发送邮件，将结果以附件的方式返回。图片分享服务就是支持简单的图片上传和下载功能，展示上传到服务器的图像，并提供任意图像的下载服务。图 5.1 为网站的数据流图：

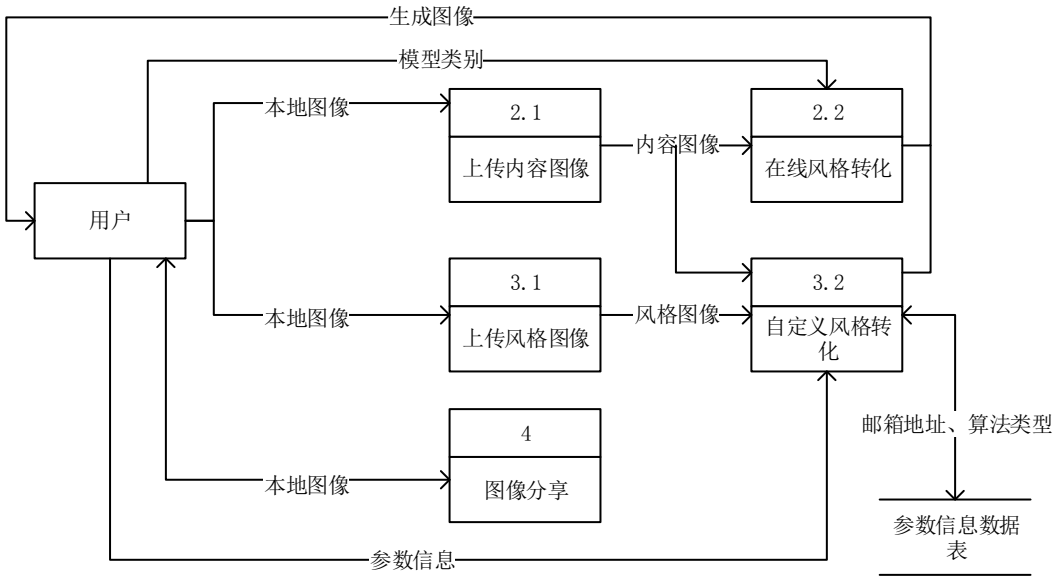

图 5.1 网站数据流图

5.1 前端设计

网站前端用到的技术主要有 Bootstrap，JavaScript，Ajax 和 HTML。Bootstrap 和 HTML 主要用于网站静态页面的设计。JavaScript，Ajax 主要用于动态更新页面以及和服务器进行数据交互。网站主要由四个 Bootstrap 模块构成，体现为四个<section>标签，分别代表主页，在线风格转化页，自定义风格转化页和图片分享页。所有页面的左侧都

有一个可关闭的导航栏，利用 JavaScript 实现导航栏的动画效果，导航栏的四个按钮分别实现四个页面的跳转。下面简单介绍一下各页面的设计。

主页设计非常简单，主体为两个 模块，分别给出网站的名字和风格转化算法原理的简单介绍，并提供一个按钮用于下载算法的相关论文，具体样式可见图 5.2。

在线风格转化页提供的功能主要有三个：提供用户上传内容图像的接口，提供用户选择已经训练好的模型的接口和提供用户查看返回结果的接口。用户上传内容图像由一个自定义的按钮实现，如图 5.3 左侧的按钮，用户点击按钮，会触发一个 onclick()函数，该函数会触发 JavaScript 的文件上传逻辑，打开文件选择界面，用户选择图片后会在按钮处显示。右侧为一个下拉菜单，分为两组：绘画风格和现实风格。绘画风格中展示的是服务器上已经训练好的快速艺术绘画风格转化算法的模型名称，现实风格中展示的是快速真实图像风格转化算法的模型名称。用户选择任意一个模型后，页面会将相应的风格图像展示出来，如图 5.5 所示，点击下面的风格转化按钮，服务器就开始执行风格转化过程了。当进度条完成后，页面将展示服务器返回的输出图像，如图 5.6。

图 5.2　网站主界面（左侧是导航）

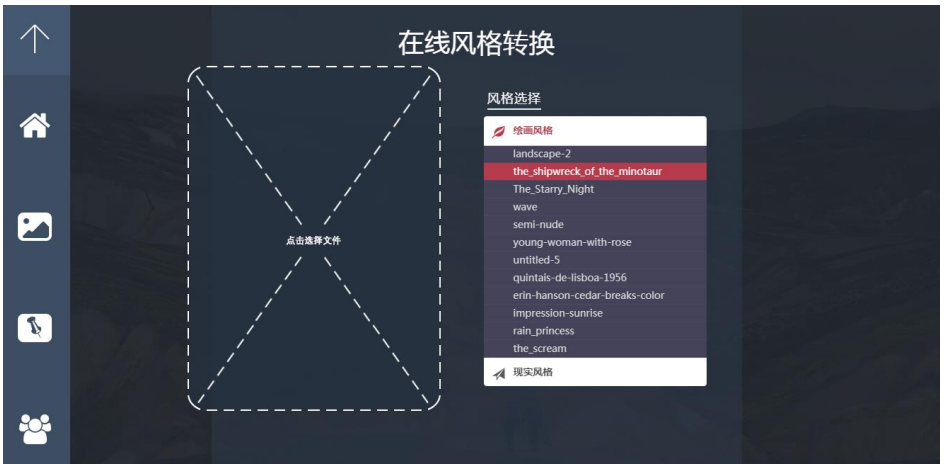

图 5.3　在线风格转化界面

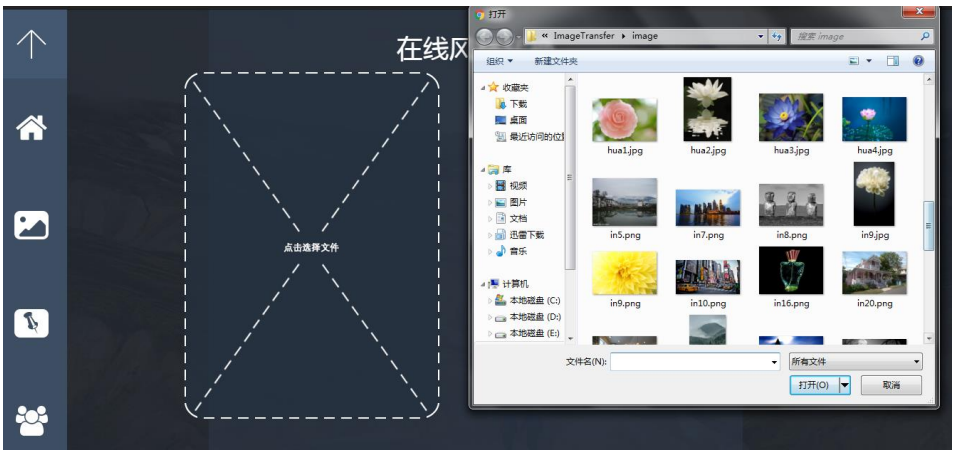

图 5.4　文件选择界面

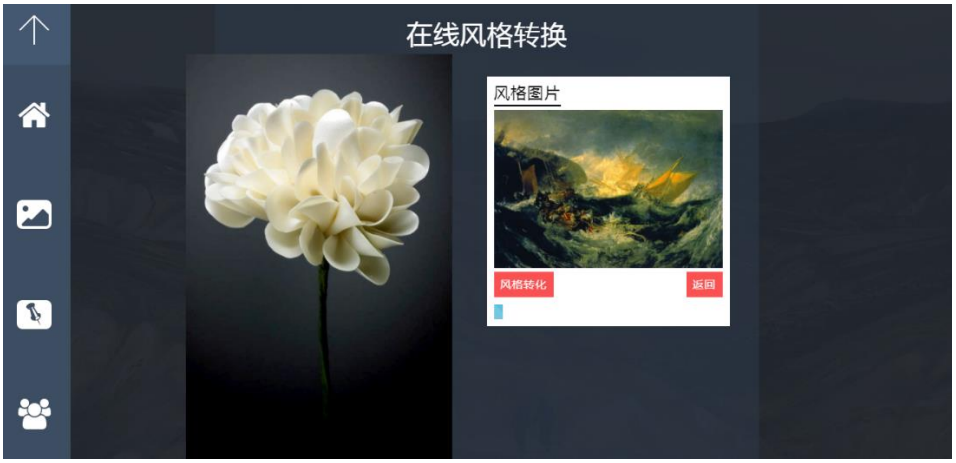

图 5.5　在线风格转化过程

图 5.6　返回结果图像界面

自定义风格转化页实现用户自定义风格转化功能，对应的服务器算法为慢速艺术绘画风格转化算法和慢速真实图像风格转化算法。如图 5.7 所示，页面两侧为两个自定义的文件上传按钮，分别上传内容图像和风格图像，并将选择的图像显示在两个按钮的相应位置。下一步页面将引导用户选择转化算法的类型，分为艺术绘画转化算法和真实图像转化算法，如图 5.8 所示。最后用户需要填写邮箱信息，用于将结果图像以邮件形式返回给用户，如图 5.9 所示。

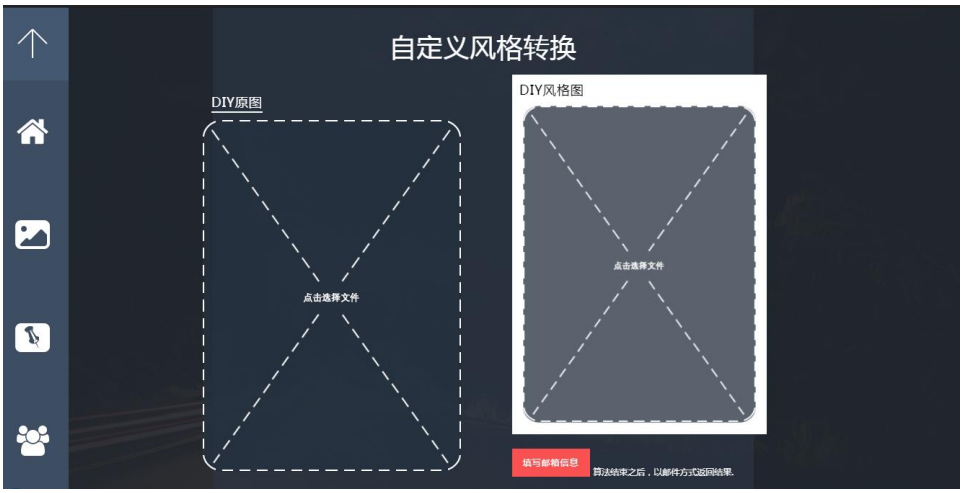

图 5.7　自定义风格转化界面

图 5.8　算法类型选择界面

图 5.9　邮箱填写界面

图 5.10　图片分享界面

图片分享页实现用户上传和下载图片，如图 5.10。页面主体是一个简单的 3D 相册，相册展示服务器上已有的分享图像，下方的两个按钮实现上传和下载功能。

5.2 后台设计

网站后台使用 Python 的 Django 框架实现。Django 采用 MTV 的开发模式，即模型 M，模板 T 和视图 V。模型 M 用于数据库操作，即定义数据表并提供数据的增删查改操作。模板 T 用于定义静态页面，并且 Django 的模板功能提供了良好的工具，可以很方便地将 Django 的特性嵌入到 HTML 页面中，支持 Django 和 HTML 的混合编程。视图 V 用于处理前端发来的请求，并将相应的结果返回。Django 的 URL 映射也非常简单，只需要对文件 urls.py 进行简单的配置即可，将用到的 url 绑定到对应的视图处理函数，那么用户对 url 的访问和请求就会自动交由视图函数处理了。感兴趣的可以访问如下地址进行网站的使用：http://210.30.97.228:8000/ImageTransfer/home/。

结　论

本文首先回顾了图像风格转化算法的发展历史，Gatys 等人[7][8]的工作是利用深度学习进行图像风格转化的第一次成功尝试，他们发现成熟的深度卷积网络可以提取到图像的高级语义信息，特别是可以将图像的内容和风格特征进行分离，进而产生一个朴素的想法，利用分离得到的内容和风格特征构造简单的欧氏距离损失，将图像风格转化问题简化为优化问题。Ulyanov 等人[12]提出纹理网络（Texture Networks）实现快速风格转化算法，利用多次上采样产生多个附加通道，叠加到后续的卷积层中，以此提高纹理生成的质量，损失函数部分利用预训练好的卷积网络实现。该方法可以生成类似 Gatys 等人的图像效果，缺点在于多次上采样使计算复杂度上升。Johnson 等人的方法[11]同样是将计算负担转移到学习阶段，提出一个卷积前馈神经网络将输入图像转化为风格化之后的输出图像，损失函数同样采用预训练的 VGG19 卷积神经网络，通过不断优化前馈神经将网络的参数，到达训练网络的目的，最后只需要一次前向传播即可完成风格转化。Huang 和 Belongie 的方法首次实现了任意风格的快速风格转化[14]，该方法的核心在于自适应实例规范化（AdaIN）层的使用。算法不会受到预先定义的风格图像的限制，甚至可以在训练的过程中改变风格图像。Chen 和 Schmidt 的工作[13]运用新的风格交换（Style Swap）技术和逆网络（Inverse Network）结构实现只需要一次前向传播就能转化图像风格，算法没有耗时的训练过程，达到了目前最快的风格转化速度，但是生成的图像效果比较一般。Luan 等人[15]的工作成功地将风格转化算法运用于真实图像，利用仿射变换损失约束转化只发生在颜色空间，利用语义分割风格损失加强图像的真实感，生成高逼真高质量的图像。

从艺术绘画风格转化算法到真实图像风格转化算法，研究者不断对图像风格转化问题提出新的方法和理论，推进该领域的发展，本文在复现前人优秀的实验结果的基础上，尝试提出快速真实图像风格转化算法，解决真实图像风格转化的速度问题，具体方法结合了 Johnson 等人[11]和 Luan 等人[15]的工作，利用前馈神经网络将计算负担转移到训练阶段，利用仿射变换损失和语义分割风格损失保持生成图像的真实性。尽管结果并不令人满意，但是在研究和实现过程中，包括复现前人算法的过程中，随着对研究问题的不断深入，锻炼了研究方法，实践了研究技巧。同时对图像风格转化领域进行了新的尝试，在未来的工作中，我将继续坚持对该问题的研究，期望可以高质量地解决真实图像风格转化问题。最后开发了简单的网站平台，利用实现的风格转化算法提供风格转化服务。

参 考 文 献

[1] He, Kaiming, Gkioxari, G., Dollár, P., et al. Mask R-CNN [C]. IEEE International Conference on Computer Vision, Venice, Italy, 2017: 2980-2988.

[2] Bulat, A. and Tzimiropoulos, G. How Far are We from Solving the 2D & 3D Face Alignment Problem? (and a Dataset of 230, 000 3D Facial Landmarks) [C]. IEEE International Conference on Computer Vision, Venice, Italy, 2017: 1021-1030.

[3] Vaswani, A., Shazeer, N., Parmar, N., et al. Attention is All you Need [C]. Neural Information Processing Systems, Long Beach, CA, USA, 2017: 6000-6010.

[4] Wu Jie, Huang D.-Y., Xie Lei, et al. Denoising Recurrent Neural Network for Deep Bidirectional LSTM Based Voice Conversion [C]. Interspeech, Stockholm, Sweden, 2017: 3379-3383.

[5] Hongzhu Cui, Chong Zhou, Xinyu Dai, et al. Boosting Gene Expression Clustering with System-Wide Biological Information: A Robust Autoencoder Approach. bioRxiv, 2017: 214122.

[6] Bojarski, M., Testa, D. D., Dworakowski, D., et al. End to End Learning for Self-Driving Cars. arXiv preprint 2016: 1604.07316.

[7] Gatys L. A., Ecker A. S. and Bethge M. A Neural Algorithm of Artistic Style [J]. The Computing Research Repository, 2015: 1508.06576.

[8] Gatys L. A., Ecker A. S. and Bethge M. Image Style Transfer Using Convolutional Neural Networks [C]. Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 2016: 2414-2423.

[9] Zeiler M. D. and Fergus R. Visualizing and Understanding Convolutional Networks [C]. European Conference on Computer Vision, Zurich, Switzerland, 2014: (1)818-833.

[10]Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv preprint, 2014:1409.1556.

[11]Johnson J., Alahi A. and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution [J]. The Computing Research Repository, 2016: 1603.08155.

[12]Ulyanov D., Lebedev V., Vedaldi A., et al. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images [J]. The Computing Research Repository, 2016: 1603.03417.

[13]Chen Tian Qi and Schmidt M. Fast Patch-based Style Transfer of Arbitrary Style [J]. The Computing Research Repository, 2016: 1612.04337.

[14]Huang Xun and Belongie S. J. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization [J]. The Computing Research Repository, 2017: 1703.06868.

[15]Luan Fujun, Paris S., Shechtman E., et al. Deep Photo Style Transfer [C]. Computer Vision and Pattern Recognition, Honolulu, HI, USA, 2017: 6997-7005.

[16]He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep Residual Learning for Image Recognition [C]. Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 2016: 770-778.

[17]Ulyanov D., Vedaldi A. and Lempitsky V. S. Instance Normalization: The Missing Ingredient for Fast Stylization [J]. The Computing Research Repository, 2016: 1607.08022.

[18]Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization[J]. Computing Research Repository, 2014:1412.6980.

[19]Radford, A., Metz, L. and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint 2015: 1511.06434.

[20]Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [C]. International Conference on Machine Learning, Lille, France, 2015: 448-456.

[21]Li, Yanghao, Naiyan Wang, Jiaying Liu, et al. Demystifying neural style transfer. arXiv preprint 2017:1701.01036.

[22]Long J., Shelhamer E. and Darrell T. Fully convolutional networks for semantic segmentation [C]. Computer Vision and Pattern Recognition, Boston, MA, USA, 2015: 3431-3440.

[23]Levin A., Lischinski D. and Weiss Y. A Closed-Form Solution to Natural Image Matting [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2008, 30(2): 228-242.

致　　谢

在算法调研和理论学习阶段，导师给予了我很大的帮助，所以首先要感谢老师。在算法实现阶段，主要是自己完成的，网站实现阶段，收到了同学的帮助和提醒，在这里也要表示感谢。