基于宽度乘数的风格迁移算法优化

东南大学，06A17108-张欣悦

2019 年 9 月 6 日

目录

摘要

At present, the style transfer algorithm is very mature and can achieve very good effects. And, real-time operation has also been implemented. However, most of the current algorithm design is based on the PC side. If the same model is migrated to a mobile phone or other embedded device, the running speed will become very slow. Based on this problem, this paper proposes a style tans algorithm optimization based on width multiplier. Under the good effect of maintaining the original style migration algorithm, the model is lighter and the speed is real-time.

1 介绍

风格迁移，简单来说，就是生成一张图片，使其具有内容图片的内容，和另一张风格图片的风格的算法（具体效果可见下图 1）。这类问题，由于“风格”的定义过于抽象，传统方法没能较好的解决这个问题。在 2015 年，[1] 利用卷积神经网络可以分别提取图片“内容”和“风格”特征的优势，第一次实现了效果可观的风格迁移。但这个算法不能实时运行。只能每次都接受一个内容图片和风格图片的输入，现场推理得到风格迁移的结果，运行速度较为缓慢，但是不受风格图像的限制，也就是说，内容图片的风格可以转变为任意输入的风格图像。

[1] 中的网络模型如图 2 所示。可以看到，网络的输入为一张白噪声图片，一张内容图像和一张风格图像。在将三张图片都输入进网络之后，在特定的层数分别计算内容损失和风格损失，接着进行反向传播，改变网络中的参数，使内容损失和风格损失的加权和最小。经过几次迭代之后，输入的白噪声图片就会变成兼具内容图片的内容和风格图片的风格的新图片。

2016 年，[2] 发现，[1] 不能实时的原因是，前向和反向传播要多次经过网络模型，所以运算量很大。基于这个发现，[2] 将网络设计为两个部分，一个是前向的网络 (feedforward net)，用于进行风格迁移；另一个是用于计算损失的网络 (loss network)，这个网络可以提取图片的高层语义信息，计算图片的特征，从而计算风格迁移产生的内容损失和风格损失。[2] 的网络模型如图 3 所示。

经过改进的 [2] 可以进行实时的风格迁移算法，但是它不能像 [1] 一样，进行任意风格图片的风格迁移。一张风格图片，只有经过训练之后，才可以进行实时的特定风格的风格迁移。如果需要加入新的风格，则需要重新训练

(a) Original image

(b) image after style transfer

(c) Style image

图 1: Style Transfer Results.(a) is original image and also content image in style transfer. (c) is style image. (b) is result of original image after combining style image's style via style transfer.

整个网络。基于此，[3] 等论文进行了单模型多风格的风格迁移算法的研究，他们在模型中加入了风格库 (style bank)。在风格库中，每一个风格都代表一些卷积滤波器。在训练时，只需要固定住其余部分，只训练卷积滤波器的部分即可。这样的设计使得加入风格变得容易了许多，不再需要从头训练一个神经网络。除此之外，[4], [5], [6] 还进行了实时任意风格的风格迁移算法的研究。

至此，风格迁移算法已经十分成熟了。但是以上所提及的方法均是在 PC 端运行的，如果要将算法部署在手机或者是嵌入式设备上，那么算法会运行的非常缓慢，远远达不到实时的效果。要想让算法在移动设备上运行，要解决的问题主要有两个：模型的存储问题和模型进行预测的速度问题。

第一，存储问题。数百层网络有着大量的权值参数，保存大量权值参数

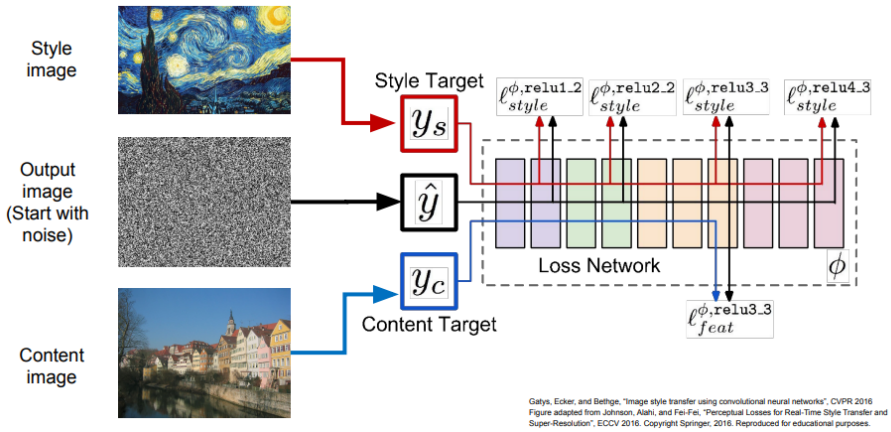

图 2: Net Model in [1]

对设备的内存要求很高；

第二，速度问题。在实际应用中，往往是毫秒级别，为了达到实际应用标准，要么提高处理器性能（短期内得不到提高），要么减少计算量。

对于这个问题，我们提出的解决方案是设计轻量型的神经网络：将 [7] 中提出的宽度乘数 (Width Multiplier) 应用到风格迁移的模型上，在保证了风格迁移的效果没有随着网络结构的减小而变差的基础上，实现了在嵌入式设备和手机上模型小型化、运行实时化。

2 相关的工作

想要减轻网络的运算量，主要有两种方式：一个是在训练结束后对模型进行压缩和剪枝；另外一种是设计出小型的神经网络，再在此基础上进行训练。本文采用了 [7] 中使用的宽度乘数 (Width Multiplier)，设计了一个神经网络，从而达到减小模型大小、加快模型运算速度的目的。

小型的神经网络，比如 [7],[8],[9],[10] 都是对卷积方式进行了改变。比如 [7] 中使用的是深度可分离卷积 (Depthwise Separable Convolution)，还加入了两种超参数用来调节网络的大小，分别是：宽度乘数 (Width Multiplier) 和分辨率乘数 (Resolution Multiplier)。这种卷积方式可以在感受野相同的情况下，减少了卷积的参数。并且两个超参数的加入使得网络中间层的深度可调节、图片输入大小可调节，增加了训练的灵活性。从而提高了运行的速

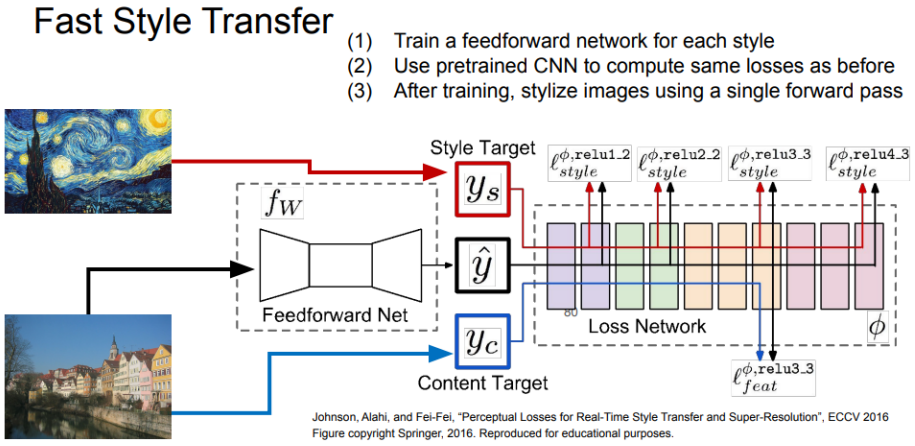

图 3: Net Model in [2]

度，减小了网络的参数量。[8] 中的卷积思想来源于 inception network[11]，分为两个部分，一个是压缩 (squeeze)，另一个是扩张 (expand)，扩张层中分别使用了 1*1 大小和 3*3 大小的卷积核，然后将两个卷积结果级联在一起。而 [9] 是先通过分组卷积 (group convolution)，再通过打乱各个通道使不同的分组之间有信息的交换，然后再通过分组卷积。[10] 中的卷积方式分别借鉴了 [11] 中的 inception 思想和 [8] 中提到的深度卷积 (depthwise network) 思想。

除了改变卷积方式之外，还有另一种优化方法是基于网络中的参数。比如降低网络中的数字精度 [13]，从 float16 下降到 float8，做半精度的训练。另外，还有二值神经网络 [12]，也就是网络中只含有 +1 和-1。很显然，这样对数字优化之后的网络效率会有显著提高。与之类似，量化神经网络 (Quantization Neural Network) 的主要目的是裁剪掉数据的冗余精度，原本 32 位精度的浮点数由 “1 8 23” 的结构构成，裁剪的方法是根据预训练得到的全精度神经网络模型中的数据分布，分别对阶码和位数的长度进行适当的减少。实验证明，对于大部分的任务来说，6 位比特或者 8 位比特的数据已经能够保证足够好的测试准确率。

3 基于宽度乘积的风格迁移算法

3.1 网络结构

本文采用的网络结构是基于 [2] 的网络结构进行改进而得到的。图 3 中有两个网络，一个是前置的风格转换网络，另一个是基于物体分类预训练的 VGG 神经网络。由于 VGG 网络的作用只是提取图像特征，方便计算各种损失，所以 VGG 神经网络在测试时是不需要的，我们只需要前置的风格转换网络就可以进行风格迁移了。因此，我们的优化只关注前置的前向网络。[2] 中原本的前向网络结构如图 4.

Layer	Activation size
Input	$3 \times 256 \times 256$
Reflection Padding (40×40)	$3 \times 336 \times 336$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 336 \times 336$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 168 \times 168$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 84 \times 84$
Residual block, 128 filters	$128 \times 80 \times 80$
Residual block, 128 filters	$128 \times 76 \times 76$
Residual block, 128 filters	$128 \times 72 \times 72$
Residual block, 128 filters	$128 \times 68 \times 68$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

图 4: Feedforward Network in [2]

我们在网络中加入了一个可调节的宽度乘数 α，即对表格中的卷积层的通道数做出改变。比如，网络中第一层卷积是 9*9*32 的卷积核，前两个 3 分别代表卷积核的长和高，第三个数字 32 代表卷积的通道数。我们将卷积的通道数乘 α，即变为 9*9*(32α)。对所有的卷积核都进行相同的操作后，就得到了我们需要的网络。

3.2 损失函数

3.2.1 内容损失

内容损失的定义式如下：

$$L_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2}\sum_{i,j}(F_{ij}^{l} - P_{ij}^{l})^2$$

其中，F^l 表示的是将内容图像 $\vec{p}$ 输入到网络内第 l 层的提取出来的特征矩阵，F_{ij}^{l} 表示在层 l 的特征矩阵中在第 i 个滤波器的位置 j 的响应值。相应的，P^l 代表的是将生成的图像 $\vec{x}$ 输入网络在第 l 层产生的特征矩阵。内容损失，就是对两个特征矩阵做均方差。

3.2.2 风格损失

下面是风格损失的定义：

$$L_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

其中，$\vec{a}$ 表示风格图片，$\vec{x}$ 表示生成的图片，w_l 是在第 l 层算出的风格损失的权重，E_l 表示在第 l 层算出的风格损失。所以风格损失的总表达式就是一个对各个层算出的风格损失的加权和。而 E_l 的定义如下：

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{ij}(G_{ij}^{l} - A_{ij}^{l})^2$$

其中，G_l 和 A_l 分别表示的是风格图像和生成的图像在第 l 层计算出的 Gram 矩阵。相应的风格损失就是对两个 Gram 矩阵做均方差。

注意这里使用了 Gram 矩阵来定义图片的风格。一张图片在神经网络层的推演过程中，我们取第 l 层来计算这张图片的 Gram 矩阵。假设图片的大小为 'w*h*c'，w 和 h 分别表示长度和高度，而 c 表示的是通道数。Gram 矩阵表示的就是任意两个通道之间的相关度。这张图片的 Gram 矩阵定义为：

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}}\sum_{j=1}^{n_w^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

3.2.3 总损失函数

综合内容和风格损失，总损失函数如下：

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

其中，α 和 β 分别代表内容和风格损失的权重，在训练的过程中，我们可以通过调整 α 和 β 的值，调整生成图像的内容和风格的占比，从而达到不同的输出效果。

如上图所示，我们在 VGG 网络的 $conv4_2$ 层计算内容损失，在 VGG 网络的 $conv1_1, conv2_1, conv3_1, conv4_1$ 和 $conv5_1$ 层取出特征矩阵，计算风格损失。

3.3 训练方法

在训练时，前向网络随机初始化，VGG 网络是由分类任务预训练而来的网络，它的参数固定不动。在每一次迭代中，特定图片的风格图像和任意的内容图像传入前向网络，生成一张目标图 (target image)，即既保留内容图片的内容，又含有风格图像的风格的图片。接着，目标图和内容图都传输进 VGG 网络进行图片特征的抓取，然后如上所述计算内容损失和风格损失。

接下来，进行梯度反向传播，即改变前向网络中的参数，使得总损失最小。在这样一次又一次的迭代过程中，前向网络中的参数不断改变，最终收敛到一个可以产生可观的结果的值。

在前向推理阶段，我们只需要将一张内容图片传进前向网络，就可以得到一张进行了风格迁移之后的结果了。

4 实验结果

4.1 对不同 alpha 的测试

宽度乘数中的 α 是可调节的，当 $\alpha = 1$ 时，该网络就和原网络的大小一样。当 $\alpha = 0.5$ 时，通道数就降为原来的一半。所以为了测试不同的 α 值对算法的影响，我们将 α 设为不同的值，可视化其内容损失和风格损失的变化。结果如图 5：

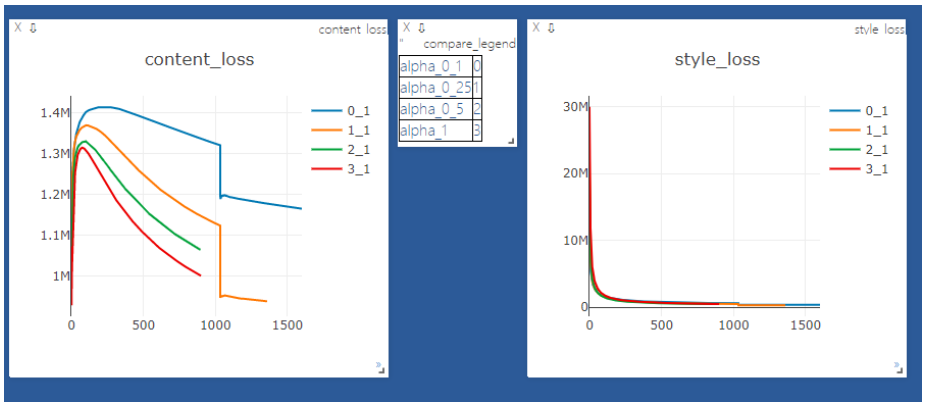

图 5: Style loss and Content loss with different alpha(0.1,0.25, 0.5, 1)

可见风格损失差别不大，内容损失虽然刚开始差别较大，但是随着迭代次数的增加，内容损失也有猛然下降的情况出现。在训练了一段时间后，$\alpha = 0.1$ 可视化风格迁移效果如图 6：

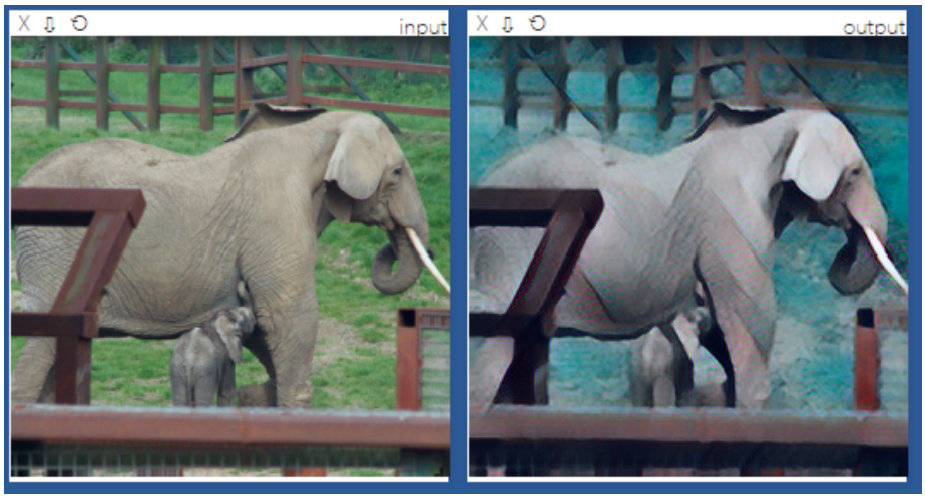

图 6: Style transfer results when $\alpha = 0.1$

综合速度和效果考虑，我们最终选择使 $\alpha = 0.1$。

4.2 与直接优化网络方法的对比

我们还用我们的方法与直接修改网络的方法进行了对比。我们通过观察原始网络中的参数分布，发现残差层 (residual block) 的参数量很大。所以我们将卷积层的最后一层和反卷积层中的第一层去掉，然后将残差层的参数改为 64，即 64 filters。这样修改以后，网络的参数量同样下降很多。我们将原始网络、修改之后的网络和基于宽度乘积的网络在 jetson nano 开发板上进行了对比，测试结果如下表：

表 1: Compare between two different changed networks.

Network	**Total tensors**	**Total mems**	**FPS in jetson nano**
Original Network	1679241	6.41M	5.09
Changed 64res network	201027	792.00k	4.06
Network with width multiplier	16497	93.00k	21.82

可以很直观的看到，虽然修改的网络参数数量下降了许多，但是帧率 (即运行速度) 提高并不明显。但是基于宽度乘数的网络不仅参数数量下降许多，帧率也有显著提高。

5 总结

基于宽度乘数的风格迁移网络，可以在保持原有风格迁移效果的基础上，将网络的参数数量减少，速度也显著提升。可见宽度乘数这种算法是有效并且高效的。

6 参考文献

1. Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576 (2015)

2. J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. arXiv preprint arXiv:1603.08155, 2016.

3. Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua.

StyleBank: An Explicit Representation for Neural Image Style Transfer. arXiv preprint arXiv:1703.09210, 2017.

4. Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu and Ming-Hsuan Yang. Universal Style Transfer via Feature Transforms. arXiv preprint arXiv:1705.08086v2,2017.

5. Tian Qi Chen and Mark Schmidt. Fast Patch-based Style Transfer of Arbitrary Style. arXiv preprint arXiv:1612.04337v1, 2016.

6. Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network.arXiv preprint arXiv:1705.06830v2, 2017.

7. Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.arXiv preprint arXiv: 1704.04861v1,2017.

8. F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. arXiv preprint arXiv:1602.07360, 2016.

9. Zhang X, Zhou X, Lin M, et al. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices[J]. 2017.

10. F. Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357v2, 2016

11. inception

12. Courbariaux M, Hubara I, Soudry D, et al. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1[J]. 2016.

13. Hubara I, Courbariaux M, Soudry D, et al. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations[J]. 2016.

14. S. Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. ICLR, 2016.