

華南師範大學

《多媒体技术》课程项目

课 程 项 目 报 告

项 目 题 目：快速图像风格迁移

所 在 学 院：计算机学院

项 目 组 长：张海飞

小 组 成 员：林泳鑫，李志鹏

开 题 时 间：2020 年 5 月 7 日

一、引言

所谓图像风格迁移,是指利用算法学习著名画作的风格,然后再把这种风格应用到另外一张图片上的技术。著名的图像处理应用 Prisma 是利用风格迁移技术,将普通用户的照片自动变换为具有艺术家的风格的图片,所以本项目也拟做一个实现图像风格迁移的程序,来满足人们对娱乐的需求。

二、国内外研究现状

1、国内:

国内对于图像风格迁移这一应用已经比较完善,但由于普通的风格迁移通过输入的内容图和输入的风格图来生成一张融合了内容和风格的新图片往往需要很长的时间,所以国内研究在此基础上,又研发出了快速图像风格迁移,就是将输入内容图片和输入风格图片加载到提前训练好的某特定风格的模型中,就可以快速的生成融合图片,大大减少了整个程序的运行时间,并且效果和普通的图像风格迁移相当。

2、国外:

国外在图像风格迁移这一技术更加的成熟,并且更加深入细致的将图像风格迁移分成了几个方面来处理。目前图像风格化迁移技术在镜头滤镜和字形设计两个领域发展较为成熟。

(1)**镜头滤镜** 用过 prisma 的人都知道只要把照片上传上去然后选择相应的艺术风格,不一会就出来了相应的图片。这其中就运用了图像风格化迁移的技术,如果把图像风格化迁移的技术做成手机应用运用它去照相那么拍出来的照片会跟名画一样很有艺术感,如果提高图像风格化迁移的速度的话也可以用它拍视频或电影如暮光之城

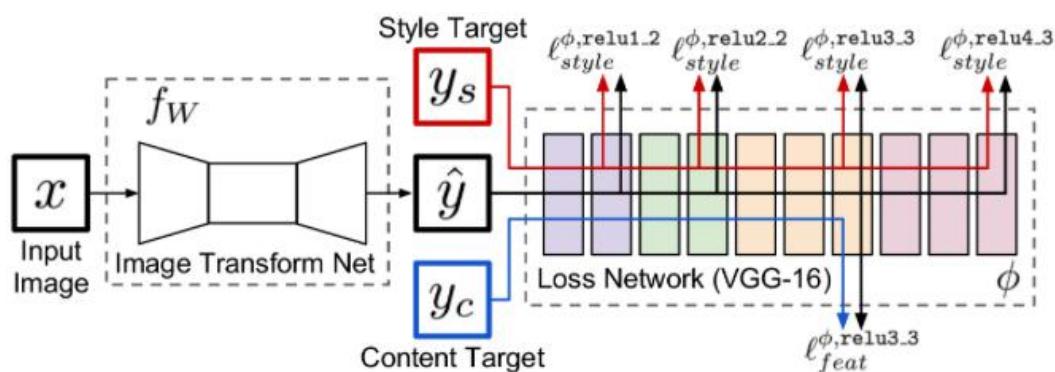
女主角克里斯汀·斯图尔特拍摄的电影《Come swim》中，也利用了神经风格迁移技术将印象派绘画风格和电影画面相融合。他就很好的利用了图像风格化迁移技术从而拍出了很好的效果

(2) **字形设计** 文字是平面设计中很突出的视觉元素。艺术家投入大量时间来设计不同的字形，使得它与其他元素在形状和纹理上相协调。这个过程是需要大量劳动力的，之前艺术家通常只设计标题或注释所需的字形子集，再在需要的地方加以调配，如果字集中没有的话又要重新设计非常麻烦，消耗大量人力财力，但现在可以利用图像风格化迁移技术去提取偏旁与笔触风格再加上需要风格迁移的目标文字输出需要的文字，节省了很多的时间。

二、项目运用的模型和算法

算法：

整个系统由两个神经网络组成，它们在图中由两个虚线框分别标出。左边的是图像生成网络，右边是损失网络。损失网络实际上是 VGGNet，这与原始的风格迁移是一致的。同原始图像风格迁移一样，利用损失网络来定义内容损失、风格损失。这个损失用来训练图像生成网络。图像生成网络的职责是生成某一种风格的图像，它的输入是一个图像，输出同样是一个图像。由于生成图像只需要在网络中计算一遍，所以速度比原始图像风格迁移提高很多。



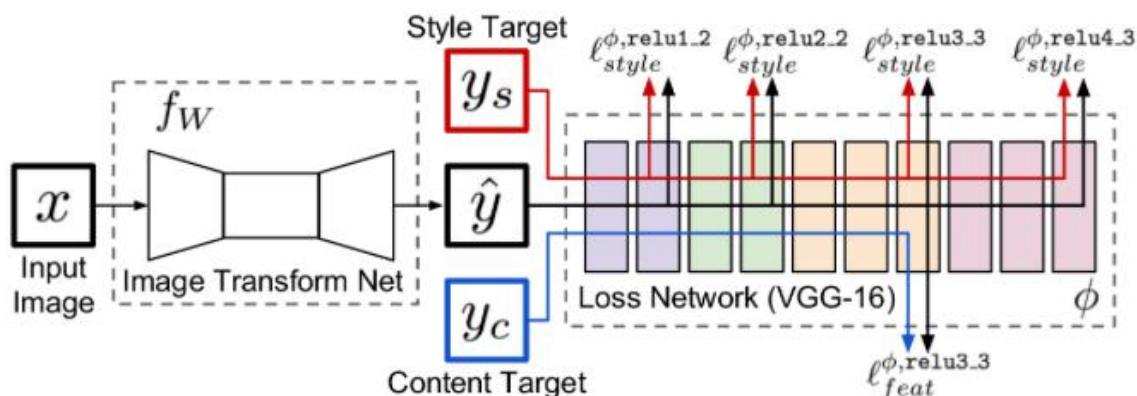
快速图像风格迁移与原始的区别：

简单来说，原始图像风格迁移就是通过 VGG 网络对一张内容照片和一张风格照片进行梯度下降法不断计算损失，让内容损失慢慢变大，风格损失不断变小，通过不停得迭代优化方法生成最适合的图像；而快速风格图像迁移则是利用两个网络框架，一个是 **VGGNet**(我们组用的是 VGG16)，另一个是图像生成网络 **Transformer Net**，对一个数据集进行训练得到模型，再利用训练好的模型以及图像生成网络再进行一次计算就可以生成图像，因此速度会比原始图像风格迁移快很多。

类型	损失定义	是否需要训练新网络	生成图像的方法
原始图像风格迁移	组合内容损失 $L_{content}$ 与风格损失 L_{style}	否。只需要预训练好的 VGGNet	利用损失，通过梯度下降法计算适合的图像
快速图像风格迁移		是。除了预训练好的 VGGNet，还需要训练图像生成网络	利用训练好的图像生成网络之间生成

两者的比较

快速风格迁移具体算法如下：



设输入的图像为 \vec{x} ，经过图像生成网络生成的图像为 \vec{y} 。 \vec{y} 在内容上应该与原始的内容图像 \vec{y}_c 接近，因此可以利用损失网络定义内容损失 $L_{\text{content}}(\vec{y}, \vec{y}_c)$ ，内容损失使用的是 VGG-16 中的 relu3_3 层输出的特征，对应上图中的 $l_{\text{feat}}^{\phi, \text{relu3}_3}$ 。另一方面，我们还希望 \vec{y} 具有目标风格图像 \vec{y}_s 的风格，因此又可以定义一个风格损失 $L_{\text{total}}(\vec{y}, \vec{y}_c, \vec{y}_s)$ 。定义风格损失时使用了 VGG-16 的四个中间层 relu1_2, relu2_2, relu3_3, relu4_3，对应图中的 $l_{\text{style}}^{\phi, \text{relu1}_2}$ ， $l_{\text{style}}^{\phi, \text{relu2}_2}$ ， $l_{\text{style}}^{\phi, \text{relu3}_3}$ ， $l_{\text{style}}^{\phi, \text{relu4}_3}$ 。同样组合这两个损失得到一个总损失 $L_{\text{total}}(\vec{y}, \vec{y}_c, \vec{y}_s)$ 。利用总损失可以训练图像生成网络。训练完成后直接使用图像生成网络生成图像。值得一提的是，在整个训练过程中，一般只固定一种风格 \vec{y}_s ，而内容图像取 \vec{y}_c 和输入 \vec{x} 一样，即 $\vec{y}_s = \vec{x}$ 。

训练模型：

因此，要实现快速风格迁移，就必须得先训练好模型，下面介绍一下我们组是怎么进行模型训练的。对于一张风格图片，为了它能够对不同内容图片有适应性的效果，需要很大的数据集。因此我们先找了一个数据集，一开始数据集只有一万多张

图片，这样在 batch_size 为 4 的情况下只能训练 3000 多次，远远达不到我们要的 10000 次训练，因此我们又重新找了一个有 40000 多张图片的数据集。















里面除了一些人脸图之外，还有各种风景图。

训练时设置学习速率 $\text{learnrate}=0.001$ ，一次训练所选取样本数 $\text{batch_size}=4$ ，总迭代次数=10000，模型保存间隔 $\text{checkpoint_interval}=5000$ 即每 5000 次迭代保存一次模型。

```
C:\WINDOWS\system32\cmd.exe - python train.py --dataset_path E:\Desktop\AI\dataset2 --style_image E:\Desktop\AI\images\styles\XP...
[Epoch 1/1] [Batch 158/10000] [Content: 2823428.25 (2506367.74) Style: 4022046.00 (15298419.32) Total: 6845474.00 (17804
[Epoch 1/1] [Batch 159/10000] [Content: 2834790.25 (2508420.38) Style: 3934927.00 (15227397.49) Total: 6769717.00 (17735
[Epoch 1/1] [Batch 160/10000] [Content: 2900252.00 (2510854.12) Style: 3841391.00 (15156676.95) Total: 6741643.00 (17667
[Epoch 1/1] [Batch 161/10000] [Content: 2932249.50 (2513455.32) Style: 3779927.00 (15086450.10) Total: 6712176.50 (17599
[Epoch 1/1] [Batch 162/10000] [Content: 2797690.50 (2515199.10) Style: 3897538.00 (15017806.47) Total: 6695228.50 (17533
[Epoch 1/1] [Batch 163/10000] [Content: 2756963.50 (2516673.27) Style: 4067534.25 (14951036.52) Total: 6824498.00 (17467
[Epoch 1/1] [Batch 164/10000] [Content: 2960980.00 (2519366.04) Style: 3662526.75 (14882621.31) Total: 6623507.00 (17401
[Epoch 1/1] [Batch 165/10000] [Content: 2877637.00 (2521524.30) Style: 3907926.50 (14816508.69) Total: 6785563.50 (17338
[Epoch 1/1] [Batch 166/10000] [Content: 2821170.50 (2523318.59) Style: 3825243.00 (14750692.72) Total: 6646413.50 (17274
[Epoch 1/1] [Batch 167/10000] [Content: 2884270.75 (2525467.11) Style: 3492875.25 (14683681.91) Total: 6377146.00 (17209
[Epoch 1/1] [Batch 168/10000] [Content: 2854721.00 (2527415.36) Style: 3678009.00 (14618559.58) Total: 6532730.00 (17145
[Epoch 1/1] [Batch 169/10000] [Content: 2811985.50 (2529089.30) Style: 3651903.50 (14554049.84) Total: 6463889.00 (17083
[Epoch 1/1] [Batch 170/10000] [Content: 2853578.50 (2530986.90) Style: 3501656.25 (14489415.96) Total: 6355235.00 (17020
[Epoch 1/1] [Batch 171/10000] [Content: 2875777.00 (2532991.49) Style: 3535325.75 (14425729.39) Total: 6411103.00 (16958
[Epoch 1/1] [Batch 172/10000] [Content: 2837809.50 (2534753.45) Style: 3900682.50 (14364890.97) Total: 6738492.00 (16899
[Epoch 1/1] [Batch 173/10000] [Content: 2908521.25 (2536901.38) Style: 3540335.75 (14302680.88) Total: 6448857.00 (16839
[Epoch 1/1] [Batch 174/10000] [Content: 2869374.75 (2538801.38) Style: 3761792.00 (14242447.23) Total: 6631167.00 (16781
[Epoch 1/1] [Batch 175/10000] [Content: 2917794.50 (2540954.75) Style: 3520016.25 (14181524.33) Total: 6437811.00 (16722
[Epoch 1/1] [Batch 176/10000] [Content: 2898195.25 (2542973.06) Style: 3389491.75 (14120552.39) Total: 6287687.00 (16663
[Epoch 1/1] [Batch 177/10000] [Content: 2832651.00 (2544600.47) Style: 3858282.75 (14062899.19) Total: 6690934.00 (16607
[Epoch 1/1] [Batch 178/10000] [Content: 2762241.50 (2545816.34) Style: 3791415.25 (14005516.60) Total: 6553657.00 (16551
[Epoch 1/1] [Batch 179/10000] [Content: 2955512.50 (2548092.43) Style: 3618403.50 (13947810.41) Total: 6573916.00 (16495
[Epoch 1/1] [Batch 180/10000] [Content: 2865211.00 (2549844.46) Style: 3517156.50 (13890182.49) Total: 6382367.50 (16440
[Epoch 1/1] [Batch 181/10000] [Content: 2732890.75 (2550850.21) Style: 3934625.75 (13835481.63) Total: 6667516.50 (16386
[Epoch 1/1] [Batch 182/10000] [Content: 2953886.50 (2553052.60) Style: 3323219.50 (13778037.58) Total: 6277106.00 (16331
[Epoch 1/1] [Batch 183/10000] [Content: 2966357.00 (2555298.82) Style: 3564253.50 (13722527.88) Total: 6530610.50 (16277
[Epoch 1/1] [Batch 184/10000] [Content: 2781669.00 (2556522.44) Style: 3508923.50 (13667319.21) Total: 6290592.50 (16223
[Epoch 1/1] [Batch 185/10000] [Content: 2863143.00 (2558170.94) Style: 3697416.75 (13613717.58) Total: 6560560.00 (16171
[Epoch 1/1] [Batch 186/10000] [Content: 2817659.00 (2559558.57) Style: 3655512.75 (13560465.15) Total: 6473172.00 (16120
023.70)]
```

在有 GPU 的情况下，完整训练完一个模型大概要 3 个小时，我们组总共训练了 9 个

模型。

 Chengbao_5000.pth	2020/6/14 1:54	PTH 文件	6,573 KB
 Chengbao_10000.pth	2020/6/14 3:02	PTH 文件	6,573 KB
 cuphead_10000.pth	2020/6/7 13:26	PTH 文件	6,572 KB
 Dongm_10000.pth	2020/6/9 10:18	PTH 文件	6,573 KB
 Fangao_5000.pth	2020/6/11 1:28	PTH 文件	6,573 KB
 Fangao_10000.pth	2020/6/11 2:32	PTH 文件	6,573 KB
 Mon_5000.pth	2020/6/9 15:25	PTH 文件	6,573 KB
 mosaic_10000.pth	2020/6/7 14:29	PTH 文件	6,572 KB
 Nahan_10000.pth	2020/6/9 0:52	PTH 文件	6,573 KB
 starry_night_10000.pth	2020/6/7 14:09	PTH 文件	6,572 KB
 XPY_5000.pth	2020/6/14 12:20	PTH 文件	6,573 KB
 XPY_10000.pth	2020/6/14 13:29	PTH 文件	6,573 KB

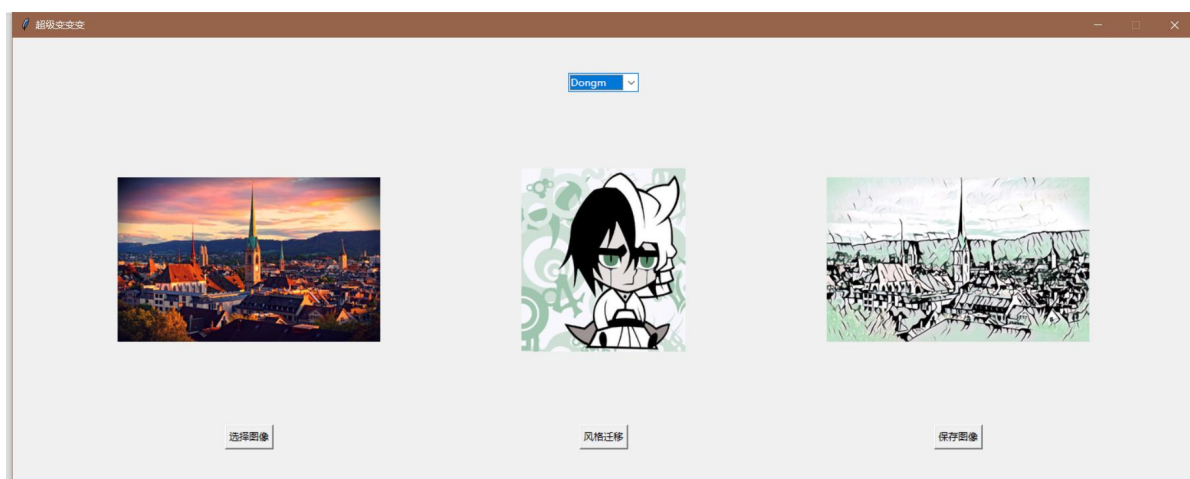
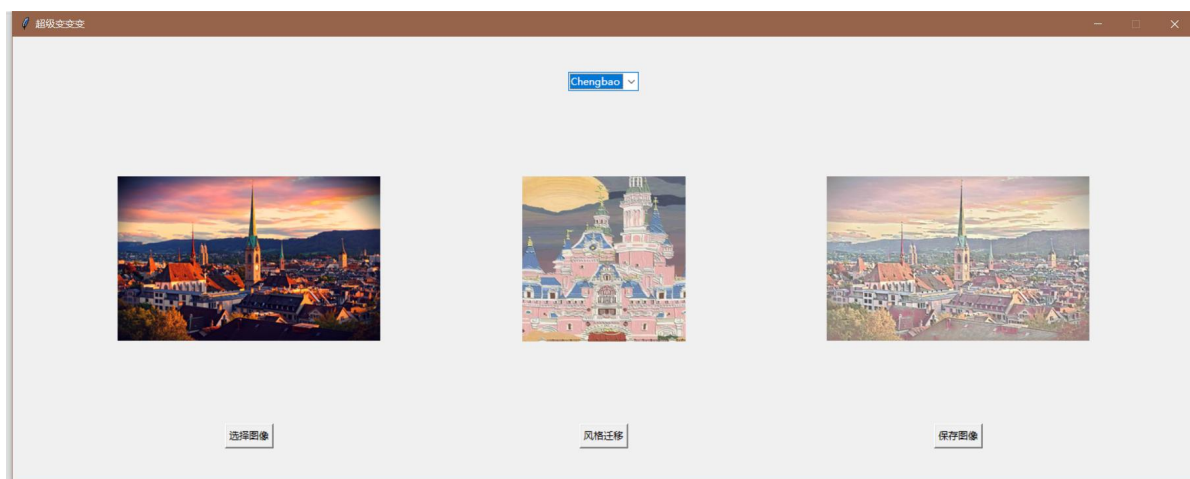
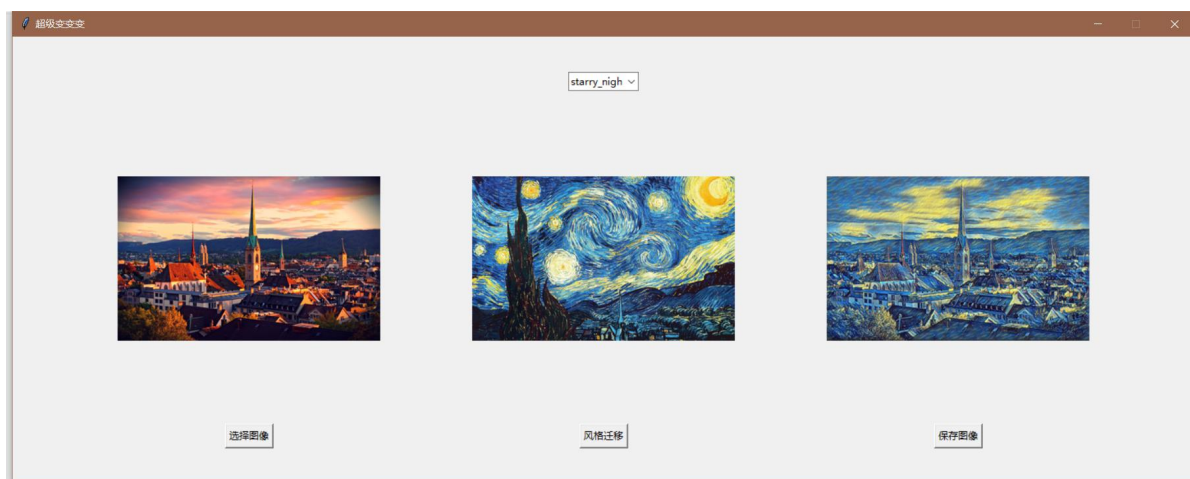
有了这些模型，再去生成一张图片，在有 **GPU** 的情况下大概只要 5 秒钟，用 **CPU** 也才大概只要 30 秒的时间。

三、实验结果分析

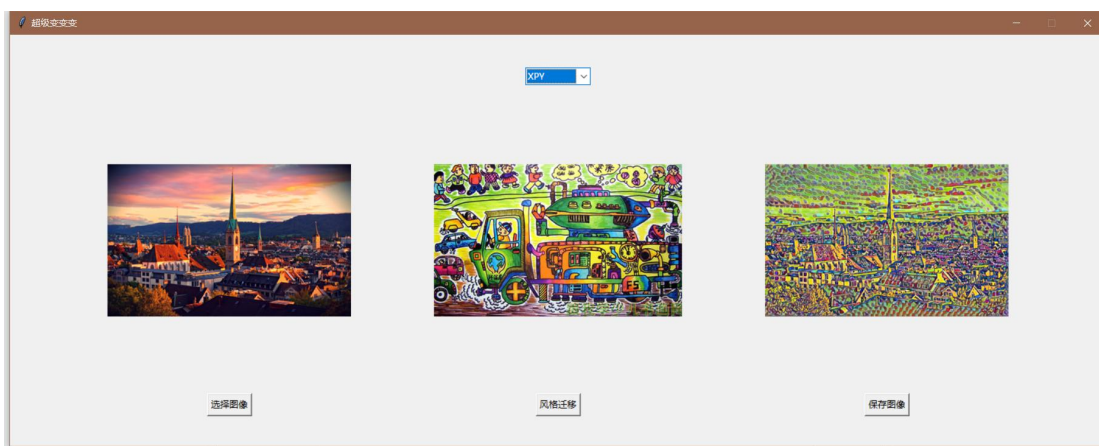
我们组总共训练了 9 个模型风格，下面是华师校门在不同风格下的效果。



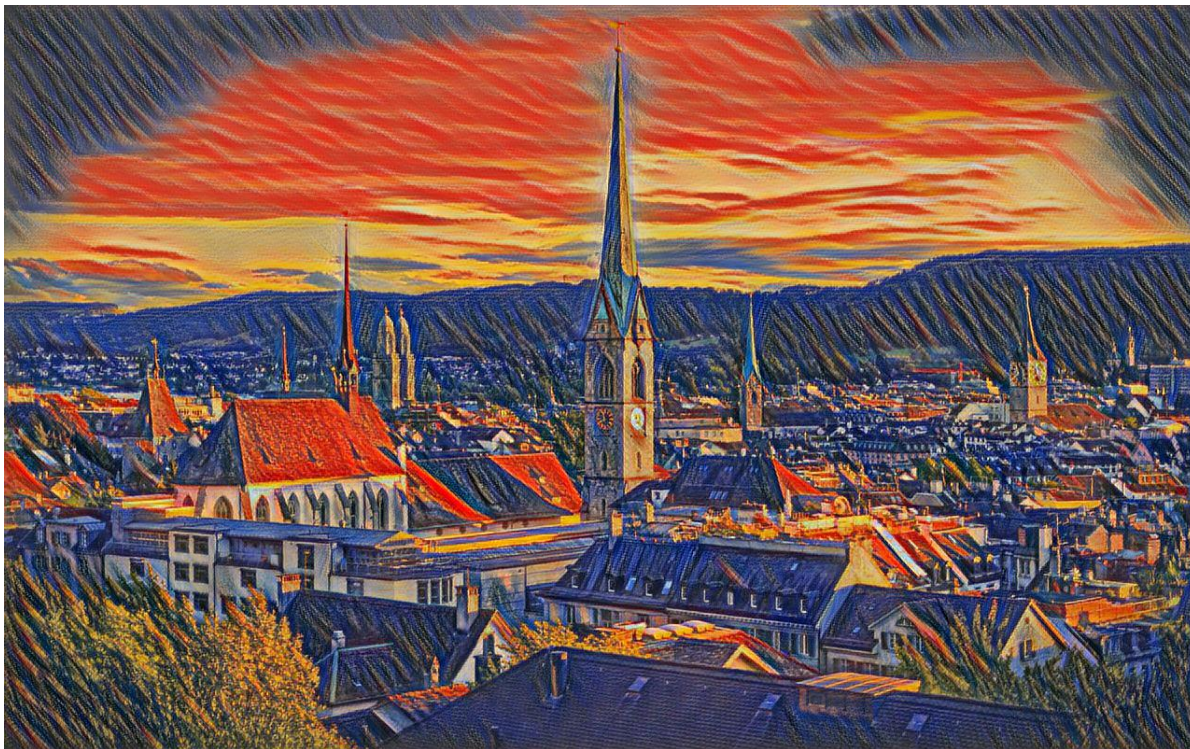
然后这是我们组做的一个简单的界面，可以选择自己的一张图片，并且选择其中的一个风格，点击迁移就可以在右侧生成出图片，点击保存就可以将风格迁移后的图片保存到特定路径。

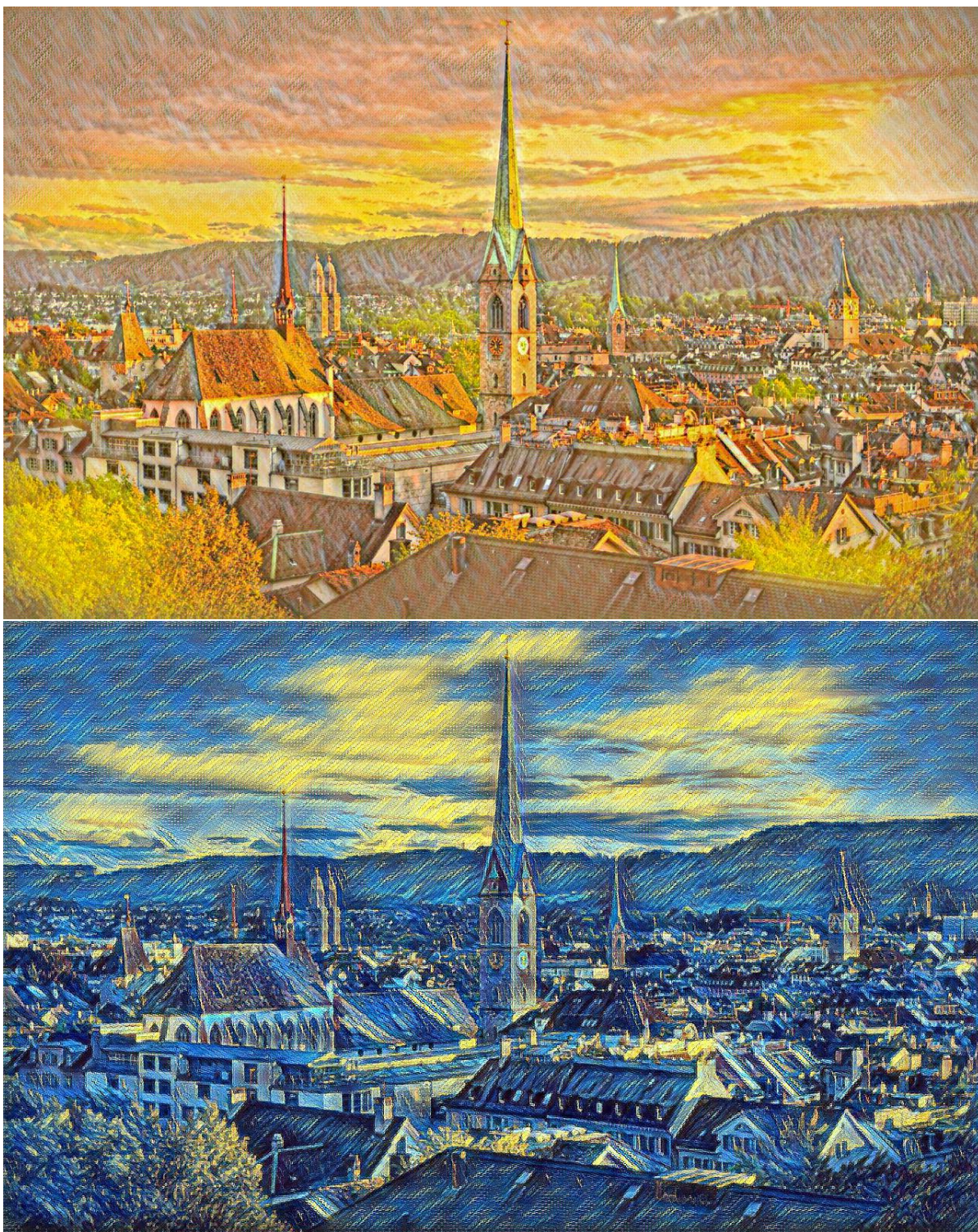


有些图的迁移效果并不是很好，可能是风格图像选得不好，比如下面这张：



迁移效果比较好的图





五、结论

从上述的训练结果,可以观察到大部分的图片的风格迁移的结果还是令人比较满意

的，不过在对于每一种风格图片进行模型训练的时候，训练的时间往往比较久，而且生成的图像还不够平滑。因此能找到一种既简单快速，又能保证图像的生成质量的方法来实现快速图像风格迁移也是有必要的。

本项目采用的是 Gatys 等人发现的预训练 VGG16 模型，虽然这是一个优秀的卷积神经网络模型，在特征提取方面很出色，但是它是一个重量级的模型，存在**体积庞大**以及**计算量巨大**的问题，并且这个 VGG16 模型最初设计出来不是专门为图像风格迁移的，而是用来提取图片中的特征从而进行分类。

所以要摆脱对预训练 VGG16 模型的依赖，设计出一个对于快速图像风格迁移的更加精小有效的**特征提取器**推动基于深度学习的图像风格迁移进一步发展的的重要途径。

六、参考文献

[1]MartinLwx,图像风格前移 (Pytorch)

<https://www.cnblogs.com/MartinLwx/p/10572466.html>

[2]《基于深度学习的图像风格迁移研究综述》，陈淑环，韦玉科，徐乐，董晓华，温坤哲，广东工业大学 计算机学院, 广州

[3]基于 PyTorch 的深度学习入门教程（八）——/details/78849250 图像风格迁移, 雁回晴空, <https://blog.csdn.net/zzlyw/article>

[4]<https://github.com/eriklindernoren/Fast-Neural-Style-Transfer>

[5]深度学习框架 PyTorch 快速开发与实战, 邢梦来, 王硕, 孙洋洋, 电子工业出版社

[6]神经网络与深度学习应用实战,刘凡平, 电子工业出版社