

# 中山大學



Computer Vision

By Mindspore

题 目 : 艺术家画作风格迁移

授课教师 : 梁小丹

组 别 : 1

日 期 : 2023/1/1

# 艺术家画作风格迁移

孙广岩 (20354242) 宋昕帅 (20354241) 李云飞 (20354068)  
刘丁烨 (20354226) 李懿展 (20354064)

中山大学，智能工程学院

**摘要：**图像风格迁移可以将一张图像的内容和另一张图像的风格结合起来，得到一张新的图像。这种技术可以用来将艺术作品的风格迁移到照片或者将照片的内容迁移到某种艺术风格。本次作业的目的是通过深度学习模型实现图像风格迁移。我们尝试了多种模型来对自然图像转梵高风格进行实验，最终，我们得到了能够实现高质量图像风格迁移的模型。

**关键词：**AIGC；GAN；Style Transfer；Domain Adaptation

## 1 Introduction

图像风格转移是将一个图像的风格转移到另一个图像的内容上的过程，创造一个新的图像，将原始图像的内容与参考图像的风格相结合。这种技术有广泛的应用，包括创造艺术，增强照片，以及为营销和设计目的提高图像的视觉吸引力。风格转换的过程涉及使用机器学习技术，特别是神经网络，来分析输入图像的内容和风格，并生成一个新的图像，将第一张图像的内容与第二张图像的风格相结合。风格转换的过去一个流行方法是使用卷积神经网络（CNN）。CNN是一种深度学习模型，特别适合于图像分析任务，它们通过分离内容和风格表征并优化这些表征的组合来创建最终的输出图像，已经成功地用于风格转移。最近几年，GAN模型的火热也在风格转移这个领域火爆了起来，有许多模型使用GAN的方法实现了非常好的效果，而我们认为风格转换是个非常有趣的图像生成领域的应用，所以选择了这个方向来作为我们的选题。

我们认为相较于去花费全部精力去钻研一个非常新的模型把它从Pytorch改成Mindspore，我们更应该首先了解尝试多个模型来对整个风格迁移领域有一个比较整体的认知，这样对于我们之后的学习和研究更加有益。我们主要以发表的时间顺序尝试了多个模型来进行探索，其中有用到Mindspore框架的，也有Pytorch框架的。我们会在每个模型上面标注，如果使用了Torch就是(T)，如果使用了Mindspore就是(M)。

我们最终提交的文件结构如下，注意原图和转换图里面的文件展示了每个模型的10张结果，我们在最后计算了所有图片的FID分数来作为量化指标，如果把全部图片都打包就太大了，所以我们每个模型就展示了10张：

```
|— 第一组_第二次作业.pdf          # 报告  
|— CNN                            # CNN模型Notebook  
|— ArbitraryStyleTransfer         # ArbitraryStyleTransfer代码  
|— CycleGAN                       # CycleGAN代码  
|— AdaAttN                         # AdaAttN代码  
|— U-GAT-IT                        # U-GAT-IT代码  
|— orignal_pic                     # 原图  
|— transferred_pic                 # 转换图
```

## 2 Dataset

非常巧的是我们在当时在确定这个选题之后，我们在Mindspore官网看到最近有一个昇思AI挑战赛-艺术家画作风格迁移，他们提供了一个有着自然图像和梵高风格图像的这样一个数据集，而且整个数据集的大小不算很大，所以我们觉得这是一个我们拿来练手的一个非常好的一个数据集，所以之后我们的实验将基于这个数据来进行尝试，具体来说，我们的训练数据集如下：

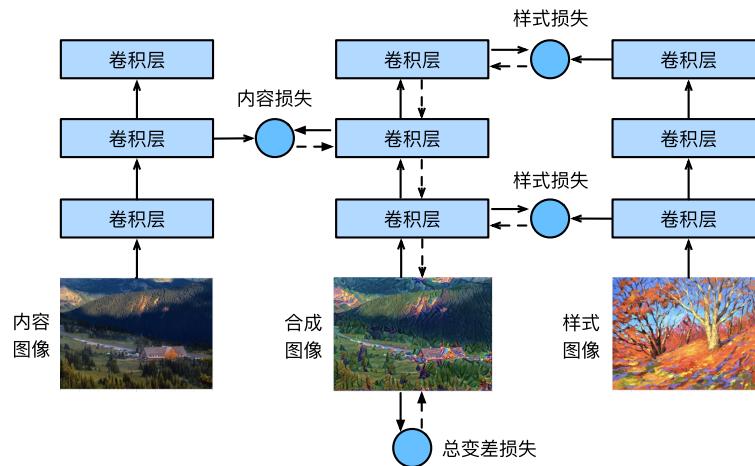
image_size	自然图像数量	梵高画像数量
(256, 256)	2991	400

## 3 Model

### 3.1 CNN (M)

#### 3.1.1 模型介绍

为了能够使得生成图片拥有风格图片的风格特色，实现风格迁移，我们使用CNN分别提取原内容图像、风格图像与生成图像的不同层次特征，并利用感知损失实现对生成图片的迭代优化。



基于卷积神经网络的风格迁移。实线箭头和虚线箭头分别表示前向传播和反向传播

生成图像是风格迁移过程中唯一需要更新的变量，即风格迁移所需迭代的模型参数。如上图所示，我们选择一个预训练的卷积神经网络来抽取图像的特征，其中的模型参数在训练中无须更新。这个深度卷积神经网络凭借多个层逐级抽取图像的特征，我们可以选择其中某些层的输出作为内容特征或风格特征。

在这里，我们使用基于ImageNet2012预训练的经典卷积神经网络[Vgg16](#)来作为特征提取网络。通过提取神经网络中不同深度卷积层的输出实现不同层次的特征提取效果，用以计算感知损失。

接下来，通过前向传播（实线箭头方向）计算风格迁移的损失函数，并通过反向传播（虚线箭头方向）迭代模型参数，即不断更新合成图像。感知损失函数由3部分组成：

1. 内容损失使合成图像与内容图像在内容特征上接近；
2. 风格损失使合成图像与风格图像在风格特征上接近；
3. 全变分损失则有助于减少合成图像中的噪点。

即：

$$\text{Perceptual Loss} = \alpha * \text{Content Loss} + \beta * \text{Style Loss} + \gamma * \text{Total variation Loss} \quad (1)$$

$\alpha$ 、 $\beta$ 、 $\gamma$ 用于控制内容损失、风格损失和全变分损失在感知损失中所占权重。其中：

$$\text{Content Loss} = \text{MSE}(\text{Content}_X, \text{Content}_Y) \quad (2)$$

$\text{Content}_X, \text{Content}_Y$ 分别为内容图像、生成图像所提取的内容特征。

$$\text{Style Loss} = \text{MSE}[\text{Gram}(\text{Style}_X), \text{Gram}(\text{Style}_Y)] \quad (3)$$

$\text{Style}_X, \text{Style}_Y$ 分别为风格图像、生成图像所提取的风格特征； $\text{Gram}$ 用于求取其格拉姆矩阵。

$$\text{Total variation Loss} = \sum_{i,j} |Y_{i,j} - Y_{i+1,j}| + |Y_{i,j} - Y_{i,j+1}| \quad (4)$$

$Y_{i,j}$ 为生成图像在*i, j*处的值。

所有代码都在Notebook里面，还是比较好的懂的，这里就不赘述了，这是我使用的超参数：

image_size	batch_size	learning rate	epoch
(256, 256)	/	0.2	500

最后，当模型训练结束时，输出风格迁移的模型参数，即得到最终的合成图像，下面是我们训练得到的图像，当然，这种模型还是有局限性的，他每次想要转换一张图片都需要重新训练，所以还是比较麻烦的



## 3.2 Arbitrary Style Transfer(M)

### 3.2.1 模型介绍

此模型的最早由Golnaz Ghiasi等人提出，为了融合神经算法在艺术风格上的灵活性和快速风格传递网络的快速性，使用内容/风格的图像对实现实时的风格转换，作者用学习从一个风格图片预测传统规范化参数的方式，建立了传统的多风格图像规范化(normalization)方法。模型可以无监督训练，并且生成的图像非常平滑，包含了许多原始图像中的重要信息。

总体来说，图片风格转换的模型可以大致分为两种方法：有参数和无参数。早期的无参数方法聚焦于从一个像素或一小片区域“生长”出视觉图像，而有参数方法则把注意力放在建立有参数模型，使得结果匹配某个区域的视觉模式。早期的有参数方法主要是通过大规模的线性过滤器得到物体边缘的统计特征，近些年则逐渐转为通过图像分类器收集物体中央的视觉特征，后者被证明有更好的视觉效果。当网络层数加深，还需要加入另一个约束——保留原来图像的内容。根本上，两个条件存在一定的互斥性，既要保留原有图像的又要转变图像风格，一般的思想会建立两个网络：特征提取和风格转换。这样大大减少了训练时的计算量，利于收敛，但是也削减了灵活性：必须为每一种新的绘画风格搭建和训练一个网络。模型的能力被限制在有限的几个特征中，而对于ArbitraryStyleTransfer模型来说，它不仅可以快速运行，还从大量数据中得到生成从未见过的风格图片，并且允许更大范围的艺术风格的探索。

模型的输入是两个：原图，想要转换的风格画作，输出为把原图转换为对应风格图像。基于两个重要的原理：当两个图片的深层特征的欧氏距离越小，两个图片的内容越相近；两个图片的浅层统计特征越相近，二者的风格就越相近。优化的方程式可以写为：

$$\min_x \mathcal{L}_c(x, c) + \lambda_s \mathcal{L}_s(x, s) \quad (5)$$

其中 $\mathcal{L}_c$ 代表内容损失， $\mathcal{L}_s$ 代表风格损失。在同一个图片分类神经网络中，我们定义较浅层的图片S和较深层的图片C，那么根据上述原理，定义内容和风格损失：

$$\mathcal{L}_s(x, s) = \sum_{i \in S} \frac{1}{n_i} \|\mathcal{G}[f_i(x)] - \mathcal{G}[f_i(s)]\|_F^2 \quad (6)$$

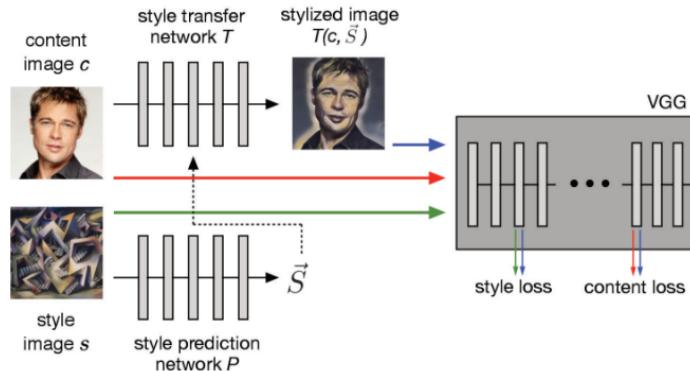
$$\mathcal{L}_c(x, c) = \sum_{i \in C} \frac{1}{n_j} \|f_j(x) - f_j(c)\|_2^2 \quad (7)$$

其中 $L_i()$ 代表了网络的第*i*层激活函数。正如上文所说的，以往的迭代式地将一张图片逐渐升级，直到符合某一种视觉特征，不仅非常消耗时间，而且只能局限于某几种风格。我们另外搭建了一个风格转换网络，用的就是简单的带有encoder/decoder的CNN，风格传输网络的参数通过使用摄影图像语料库作为内容来最小化上面的损失函数来训练。由此产生的网络可能会在艺术上更快地渲染图像，但必须为每种绘画风格学习单独的网络。

然而，为每一种风格训练一次网络仍然非常繁琐，所以在原有的基础上我们就想着保留一些东西，比如encoder/decoder；但在激活层的正则化参数上根据不同的风格而改变。也即经典的一个归一化公式中加入两个可训练参数：

$$\tilde{z} = \gamma_s \left( \frac{z - \mu}{\sigma} \right) + \beta_s \quad (8)$$

$\gamma$ 和 $\beta$ 就是归一化中可训练的参数。总的来说，整个模型的结构可以用下图来概括：



### 3.2.2 训练/测试代码讲解与超参数设置

首先将数据集读入，统一reshape成同样的大小，然后进行预处理。我们的预处理方法包含两个：随机裁剪和随机翻转，这是为了增强模型的泛用性和提取到一些翻转不敏感的特征，最后用`getitem()`函数生成总体数据集。接着就是通过采样的方法，从总体中取出一些部分来作为训练数据集，具体方法就是通过随机选取下标来实现，用mindspore的`GeneratorDataset()`函数来生成可训练的数据集。

模型的前向结构如下：

```
def construct(self, content_img, style_feat):
    style_vector = self.style_prediction_network(style_feat)
    stylized_img = self.style_transfer_network(content_img, style_vector)
    return stylized_img
```

如前文所述，模型主要分为了两个部分，分别介绍预测网络和转换网络：

```
class style_prediction_network(nn.Cell):
    def construct(self, style_feat):
        style_vector = style_feat.mean(axis=3).mean(axis=2)
        style_vector = self.flatten(style_vector)
        style_vector = self.fc(style_vector)
        return style_vector
```

主要的结构就是两个全连接层。

转换网络是基于卷积层以及残差连接的。残差连接ResidualBlockWIthStyle可以参考resnet，主要目的就是防止随着网络层数增加出现信息遗忘以及梯度消失的问题，而将提取到的风格style通过残差的方式直接加进去，就是最直观也最简单粗暴的方法，然后`UPsampleConvReluwithStyle()`主要就是使用了线性插值，是图像的经典处理方法，对于提升视觉效果有很大提升：（代码比较长就只展示了前向代码，res1, 2, 3, 4, 5就是运行了res1,res2,res3,res4,res5函数，输入都是x和style\_vector,输出都是x, upconv同理）

```
def construct(self, content, style_vector):
    """
    construct"""
    x = self.conv1(content)
    x = self.conv3(self.conv2(x))
    x = self.res1, 2, 3, 4, 5(x, style_vector)
    x = self.upconv1, 2, 3(x, style_vector)
    x = self.tanh(x)
    return x
```

损失函数正如第一部分的图和公式所表述，需要分别得到style loss和content loss，则要在vgg网络中间层的结果拿出来计算。至于具体的计算两个loss，可以简单使用mse\_loss即可。

下面是训练使用的超参数设置：

image_size	batch_size	learning rate	epoch
(256, 256)	16	1e-4	100

使用如下命令就可以来进行训练：

```
bash ./scripts/run_standalone_train.sh ascend 0 [CONTENT_PATH] [STYLE_PATH] [CKPT_PATH]
```

我截取了部分的训练日志：

```
INFO:root:Using MoXing-v2.0.1.rc0.ffd1c0c8-ffd1c0c8
INFO:root:Using OBS-Python-SDK-3.20.9.1
ModelArt
INFO:root:pid: None. 1000/2991
INFO:root:pid: None. 2000/2991
train content size: 2991 train style size: 400
content: ./cache/con
style: ./cache/sty
load vgg16_conv1
load vgg16_conv2
load vgg16_conv3
load vgg16_conv4
load inceptionv3
start training style transfer:
training 1 epoch:
2022-12-25 13:28:43 epoch 0: 0/187, loss=607.0
2022-12-25 13:28:47 epoch 0: 1/187, loss=613.0
2022-12-25 13:28:47 epoch 0: 2/187, loss=967.5
2022-12-25 13:28:47 epoch 0: 3/187, loss=647.5
2022-12-25 13:28:48 epoch 0: 4/187, loss=737.0
2022-12-25 13:28:48 epoch 0: 5/187, loss=708.5
2022-12-25 13:28:48 epoch 0: 6/187, loss=624.0
2022-12-25 13:28:49 epoch 0: 7/187, loss=640.0
2022-12-25 13:28:49 epoch 0: 8/187, loss=513.0
...
...
...
022-12-25 14:59:49 epoch 99: 178/187, loss=134.2
2022-12-25 14:59:49 epoch 99: 179/187, loss=151.2
2022-12-25 14:59:49 epoch 99: 180/187, loss=122.3
```

```
2022-12-25 14:59:49 epoch 99: 181/187, loss=130.9
2022-12-25 14:59:50 epoch 99: 182/187, loss=128.6
2022-12-25 14:59:50 epoch 99: 183/187, loss=127.06
2022-12-25 14:59:50 epoch 99: 184/187, loss=121.8
2022-12-25 14:59:51 epoch 99: 185/187, loss=123.5
2022-12-25 14:59:51 epoch 99: 186/187, loss=129.8
per step needs time:279ms
per epoch style_loss:132.53475936
100/100 epoch finished
```

使用下面的命令可以进行测试：

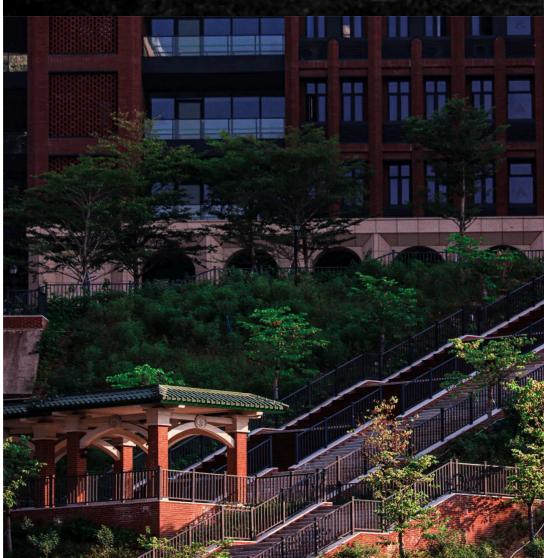
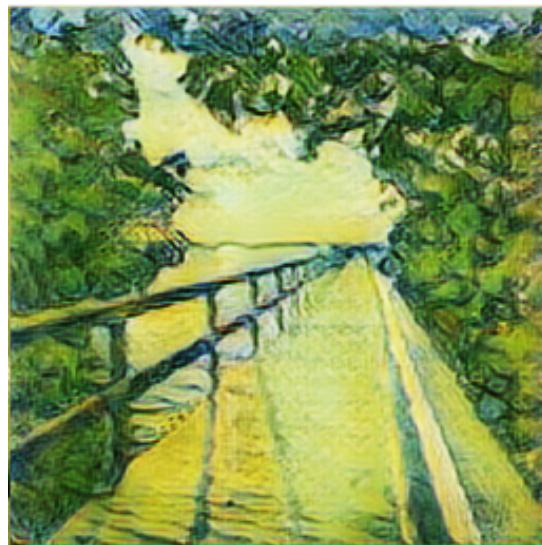
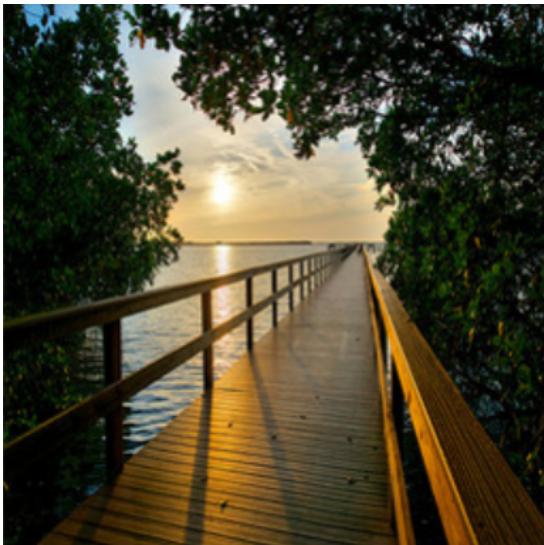
```
bash ./scripts/run_eval.sh Ascend 0 /home/ma-user/work/test-final /home/ma-user/work /test-
style /cache/ckpt/inceptionv3.ckpt /cache/out/style_transfer_model_0100.ckpt
```

这个是测试的输出日志：

```
start
test content size: 1000 test style size: 1
load inceptionv3
[WARNING] ME(596347:281473834822208,MainProcess):2023-01-08-17:26:43.270.199
[mindspore/train/serialization.py:674] For 'load_param_into_net', remove parameter
transfer_net.style_prediction_network.fc.weight's prefix name: transfer_net., continue to load
it to net parameter style_prediction_network.fc.weight.
=====starting test=====
per step needs time:169ms
test content size: 1000 test style size: 1
load inceptionv3
[WARNING] ME(596347:281473834822208,MainProcess):2023-01-08-17:29:36.161.423
[mindspore/train/serialization.py:674] For 'load_param_into_net', remove parameter
transfer_net.style_prediction_network.fc.weight's prefix name: transfer_net., continue to load
it to net parameter style_prediction_network.fc.weight.
=====starting interpolation test=====
per step needs time:390ms
finish
```

### 3.2.3 图片结果展示

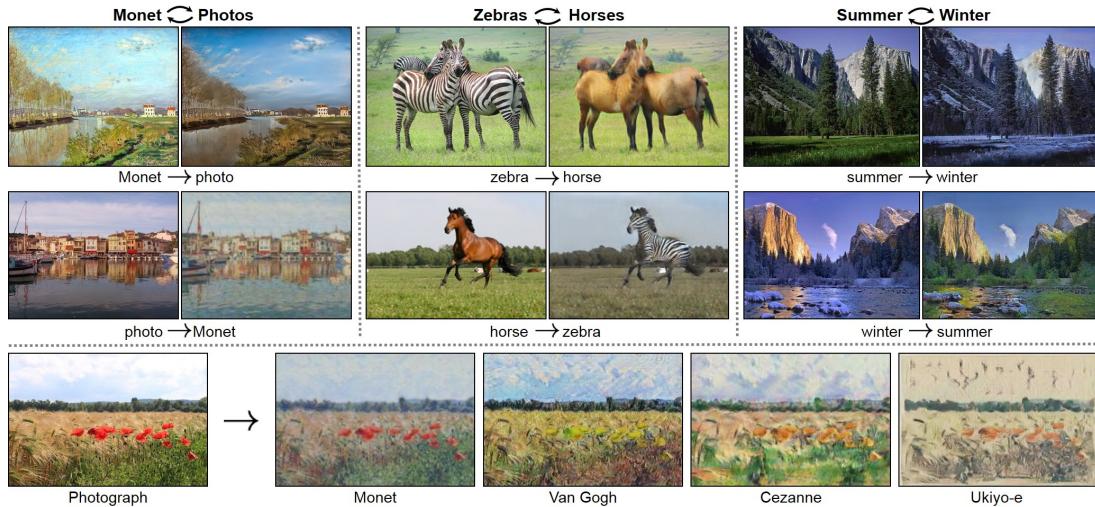
可以看到模型可以很好地把原图转换为想要的风格，我们此次使用的风格数据集是梵高的画作，转换出的图片从线条和颜色上都是非常接近风格图像的，而其中的主要内容也仍然得到了保留。总的来说，模型的效果比较优秀，并且具有快速、泛用性强的特点，并且可以规模化，使其具有很多向的风格转换能力。



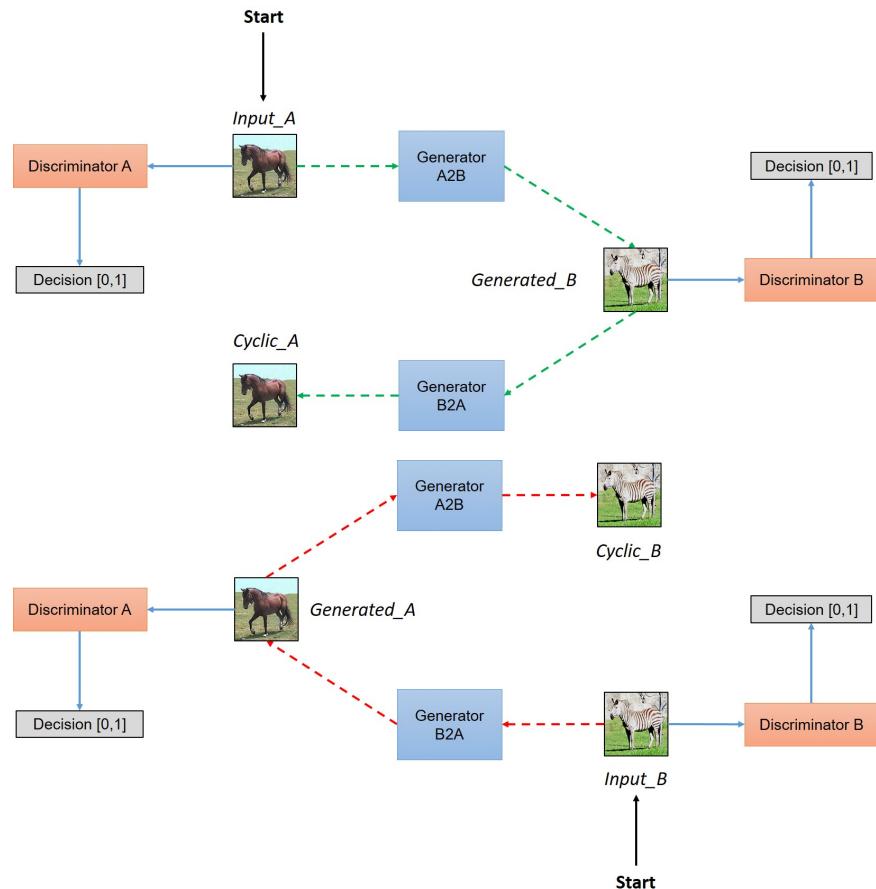
### 3.3 CycleGAN(M)

#### 3.3.1 模型介绍

CycleGAN发表于ICCV2017，它可以将真实场景变为艺术优化，将马变成斑马，将夏天变成冬天：



整个模型的结构如下：



在一个配对数据集中，每张图像，比如说imgA，都被手动映射到目标域的一些图像，比如说imgB，这样它们就有了各种特征。这些特征可用于将图像（imgA/imgB）映射到其相应的映射对应物（imgB/imgA）。基本上，配对是为了使输入和输出共享一些共同的特征。这种映射定义了一个图像从一个领域到另一个领域的有意义的转变。因此，当我们有配对的数据集时，生成器必须从DA域中获取一个输入，例如输入A，并将这个图像映射到一个输出图像，例如genB，这个图像必须接近其映射的对应物。但在非配对数据集中，我们没有这种奢侈，没有预先定义的有意义的转换，我们可以学习，所以，我们将创造它。我们需要确保输入的图像和生成的图像之间有一些有意义的关系。所以，作者试图通过以下方式来实现这一点：生成器将把域DA中的输入图像（inputA）映射到目标域DB中的一些图像，但为了确保这些图像之间存在有意义的关系，它们必须共享一些特征，这些特征可以用来把这个输出图像映射回输入图像，所以必须有另一个生成器能够把这个输出图像映射回原始输入。所以，你可以看到这个条件在输入A和基因B之间定义了一个有意义的映射。

简而言之，该模型的工作原理是将域DA的输入图像送入我们的第一个生成器GeneratorA→B，其工作是将域DA的给定图像转换为目标域DB的图像。然后，这个新生成的图像被送入另一个生成器GeneratorB→A，该生成器将其转换为来自原域DA的图像，即CyclicA（想想自动编码器，只不过我们的潜在空间是Dt）。正如我们在上一段所讨论的，这个输出图像必须接近原始输入图像，以定义一个有意义的映射，这在未配对数据集中是不存在的。

正如你在上图中看到的，两个输入被送入每个鉴别器（一个是对应于该领域的原始图像，另一个是通过生成器生成的图像），鉴别器的工作是区分它们，这样鉴别器就能对抗对手（在这种情况下是生成器）并拒绝它生成的图像。而生成器希望确保这些图像被鉴别器接受，所以它将尝试生成与DB类中的原始图像非常接近的图像。事实上，生成器和鉴别器实际上是在进行一场博弈，当生成器的分布与所需分布相同时，其纳什均衡就实现了。

### 3.3.2 训练/测试代码讲解与超参数设置

本次我们尝试使用了Mindspore官方的大模型体验平台来进行训练和推理。

点击“创建训练实例”，填写相应的信息（训练名称，代码目录，启动文件，数据集，输出路径，超参数）：

你可以通过表单方式创建训练实例，详情请参考 [表单方式-创建训练实例](#)

* 训练名称	train1
* 训练平台	ModelArts
* 代码目录	train/
* 启动文件	train.py
* 框架版本	MindSpore
* 计算资源	GPU:1*NVIDIA-V100(32GB) CPU:8核 64GB 3200 GB
描述	请输入描述

\* 训练平台 ModelArts

\* 代码目录 train/ 选择

\* 启动文件 train.py 选择

\* 框架版本 MindSpore mindspore\_1.3.0-cuda\_10.1-py\_3.7-ubuntu\_1804-

\* 计算资源 GPU:1\*NVIDIA-V100(32GB)|CPU:8核 64GB 3200 GB

描述 请输入描述

---

输入模型 ①

\* 引用参数 G\_A\_ckpt

\* 参数值 MindSpore / CycleGAN\_model / G\_A\_19.ckpt

\* 引用参数 G\_B\_ckpt

\* 参数值 MindSpore / CycleGAN\_amodel / G\_B\_19.ckpt

\* 引用参数 D\_A\_ckpt

\* 参数值 MindSpore / CycleGAN\_amodel / D\_A\_19.ckpt

\* 引用参数 G\_B\_ckpt

\* 参数值 MindSpore / CycleGAN\_amodel / D\_B\_19.ckpt

④ 添加训练输入模型

输入数据集 ①

\* 引用参数 data\_path

\* 参数值 drizzleyk / CycleGAN\_image / vangogh/

④ 添加训练输入数据集

训练输出  若你要支持训练输出，需在训练代码中指定参数名为output\_path，并将训练生成的文件保存在该参数路径下。

自定义评估  默认为标准评估，打开此按钮则支持自定义评估，且需在训练代码中指定解析参数名为aim\_repo，并将aim生成的仓库保存在该参数路径下。

超参 ①

max_epoch	= 50
save_checkpoint_epochs	= 5
platform	= CPU
image_size	= 256
need_dropout	= False
load_ckpt	= True

④ 添加超参

环境变量 ① ④ 添加环境变量

**创建**

训练名称 train1

```
=====start training=====
Epoch[1][100/400] step cost: 11434.25 ms, G_loss: 0.09, D_loss:0.21, loss_G_A: 0.89, loss_G_B: 0.72, loss_C_A: 1.25, loss_C_B: 1.18, loss_idt_A: 0.46, loss_idt_B: 0.58
Epoch[1][100/400] step cost: 11328.26 ms, G_loss: 5.19, D_loss:0.30, loss_G_A: 0.23, loss_G_B: 0.54, loss_C_A: 1.48, loss_C_B: 1.58, loss_idt_A: 0.65, loss_idt_B: 0.73
Epoch[1][100/400] step cost: 11349.05 ms, G_loss: 4.89, D_loss:0.41, loss_G_A: 0.53, loss_G_B: 0.30, loss_C_A: 0.98, loss_C_B: 1.46, loss_idt_A: 0.92, loss_idt_B: 0.69
Epoch[1][100/400] step cost: 11336.56 ms, G_loss: 5.17, D_loss:0.25, loss_G_A: 0.88, loss_G_B: 0.48, loss_C_A: 1.51, loss_C_B: 0.91, loss_idt_A: 0.84, loss_idt_B: 0.47
Epoch [1] total cost: 4544926.45 ms, per step: 11362.32 ms, G_loss: 5.18, D_loss: 0.30
[WARNING] ME(480:39763949885248, MainProcess):2022-12-23-16:51:02.780.633
[mindspore/dataset/engine/datasets_user_defined.py:699]GeneratorDataset num_parallel_workers: 8 is too large which maybe cause a lot of memory occupation (>85%) or out of memory(OOM) during multi process running.
Therefore, it is recommended to reduce num_parallel_workers to 1 or smaller.
Epoch[2][100/400] step cost: 12074.50 ms, G_loss: 5.01, D_loss:0.14, loss_G_A: 0.67, loss_G_B: 0.79, loss_C_A: 1.33, loss_C_B: 1.15, loss_idt_A: 0.50, loss_idt_B: 0.58
Epoch[2][200/400] step cost: 11764.18 ms, G_loss: 4.88, D_loss:0.47, loss_G_A: 0.25, loss_G_B: 0.44, loss_C_A: 1.42, loss_C_B: 1.53, loss_idt_A: 0.56, loss_idt_B: 0.67
Epoch[2][300/400] step cost: 11773.09 ms, G_loss: 5.20, D_loss:0.34, loss_G_A: 0.57, loss_G_B: 0.46, loss_C_A: 1.00, loss_C_B: 1.80, loss_idt_A: 0.68, loss_idt_B: 0.69
Epoch[2][400/400] step cost: 11747.83 ms, G_loss: 6.01, D_loss:0.33, loss_G_A: 0.69, loss_G_B: 0.45, loss_C_A: 1.54, loss_C_B: 1.96, loss_idt_A: 0.88, loss_idt_B: 0.50
Epoch [2] total cost: 4735996.22 ms, per step: 11639.99 ms, G_loss: 5.08, D_loss: 0.28
[WARNING] ME(480:39763949885248, MainProcess):2022-12-23-18:09:58.775.908
[mindspore/dataset/engine/datasets_user_defined.py:699]GeneratorDataset num_parallel_workers: 8 is too large
```

训练详情

创建时间	2022-12-23
运行状态	运行中
运行时长	0:38:07
AI引擎	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64
计算节点个数	1
规格	modelarts.p3large.public
日本文件	train.log
训练输出	训练中

评估

训练日志可视化, 请按照输入框参数范围, 目前只支持LossMonitor, 详情请参考文档, 更多的参数评估请选用自定义评估

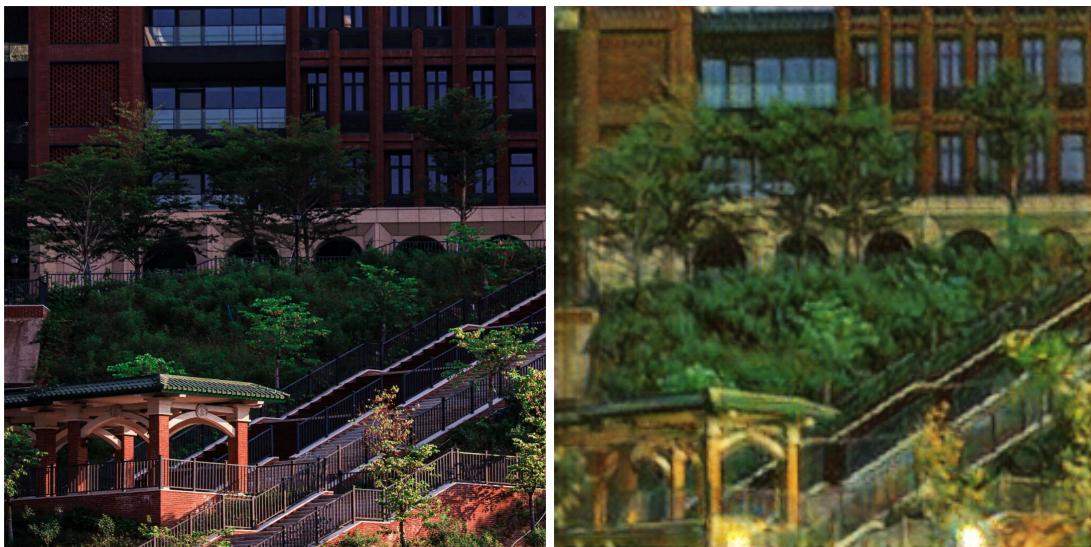
learning\_rate [请输入例0.01, 0.02, 0.03的字段]

启动推理可以得到一个可视化界面:



### 3.3.3 图片结果展示

其中可以选择两种迁移方向, 分别是真实转梵高和梵高转真实, 下面是几组我们尝试的结果, 其中左边的为输入图片:





下面是将梵高风格转为真实风格的尝试：

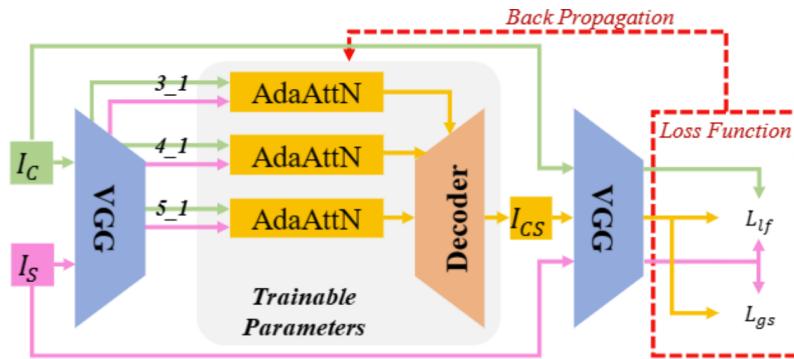


可以看到从梵高转到真实风格的人脸效果比较差，风景的话相对好一些，这个主要我们认为是训练数据中真实场景都是风景照，并没有真实场景中的人脸照，所以导致整个效果比较差。

## 3.4 AdaAttN(T)

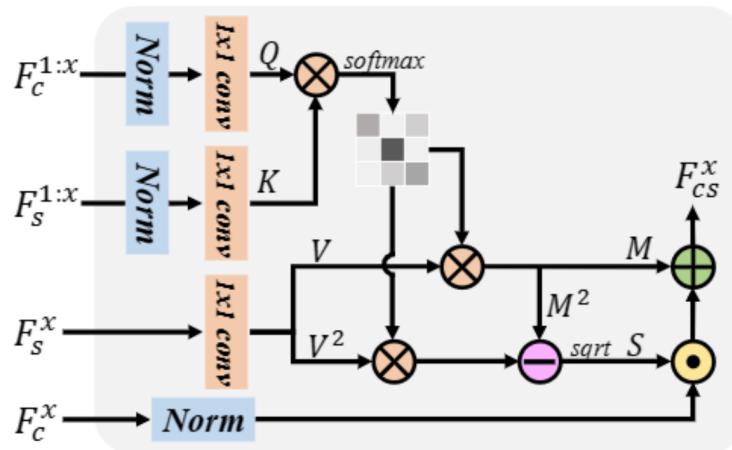
### 3.4.1 模型介绍

自适应注意归一化Adaptive Attention Normalization (AdaAttN) 方法在图像风格迁移中应用空间广阔。严格意义上，AdaAttN模块在整体网络结构中仅包括一小部分，结合运用注意力机制与统计归一化，在原有的任意风格迁移的网络中解决已有方法无法解决的图像局部失真问题。AdaAttN整体风格迁移网络结构如下：



如图所示，网络整体框架旨在利用内容图像 $l_c$ 以及风格图像 $l_s$ 合成风格化图像 $l_{cs}$ ，同时呈现 $l_c$ 的内容以及 $l_s$ 的风格。在网络的具体搭建上，使用预先训练好的VGG-19网络作为编码器，从 $l_c$ 以及 $l_s$ 输入中分别提取出不同深度的特征（如图中的ReLU3\_1，ReLU4\_1，ReLU5\_1输出）。而后，网络集成了AdaAttN模块用于接收上述的不同深度特征，经过自适应注意归一化处理，将内容与风格特征整合。进一步将当前层（深度）的特征与其前一层的下采样特征连接起来，输入至解码器中，其中解码器设置为VGG的对称结构，将得到的特征处理后输出风格化图像 $l_{cs}$ 。至此已经得到目标图像，然而在训练阶段，为了得到损失，还需要将 $l_c$ 、 $l_s$ 和 $l_{cs}$ 再一次经过VGG层提取特征，分别计算得到全局风格损失 $L_{gs}$ 以及局部特征损失 $L_{lf}$ ，二者加权得到整体损失，反向传播至AdaAttN-解码器结构中用于训练参数。

多级AdaAttN模块具体结构如下：



总结而言，AdaAttN工作流程为：(1) 计算具有从浅层到深层的内容和风格特征的注意力分布；(2) 计算风格特征的加权均值和标准差分布；(3) 对内容特征进行自适应归一化，以实现逐点特征分布对齐。注： $F_c^x$ 表示从编码器 ReLUx\_1层输出的内容特征，其他同理。

### 3.4.2 训练/测试代码讲解与超参数设置

下面是训练使用的超参数设置：

image_size	batch_size	learning rate	epoch
(512, 512)	8	2e-3	5

使用如下命令就可以来进行训练：

```
python -m visdom.server  
bash train_adaattn.sh
```

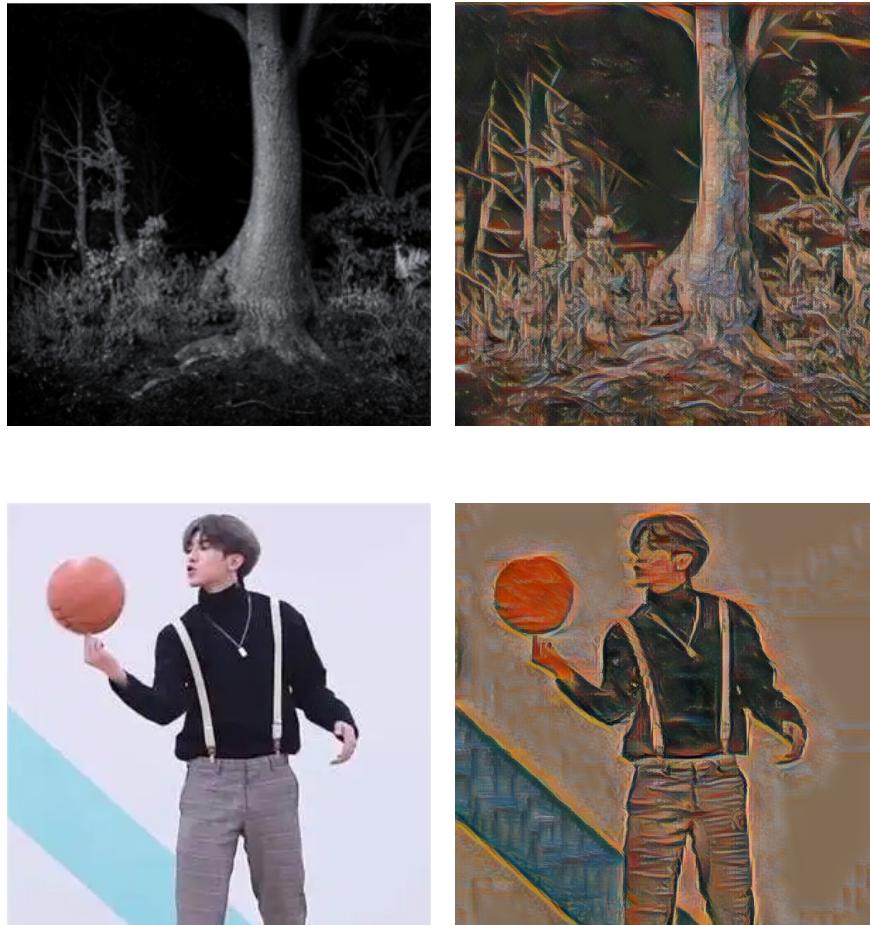
```
pool_size: 50  
preprocess: resize_and_crop  
save_by_iter: False  
save_epoch_freq: 5  
save_latest_freq: 5000  
serial_batches: False  
shallow_layer: True  
skip_connection_3: True  
style_path: datasets/styles  
suffix:  
update_html_freq: 1000  
verbose: False  
End -----  
/usr/local/lib/python3.8/dist-packages/torchvision/transforms/transforms.py:332: UserWarning: Argument interpolation should be of type InterpolationMode instead of int. Please, use InterpolationMode enum.  
warnings.warn([training] dataset [UnalignedDataset] was created  
The number of training samples = 40964  
initialize network with normal  
initialize network with normal  
initialize network with normal  
model [AdaAttnModel] was created  
----- Networks initialized -----  
[Network decoder] Total number of parameters : 4.095 M  
[Network transformer] Total number of parameters : 9.067 M  
[Network adaattn_3] Total number of parameters : 0.468 M  
-----  
/usr/local/lib/python3.8/dist-packages/torch/nn/functional.py:788: UserWarning: Note that order of the arguments: ceil_mode and return_indices will change to match the args list in nn.MaxPool2d in a future release.  
warnings.warn("Note that order of the arguments: ceil_mode and return_indices will change"  
saving the latest model (epoch 1, total_iters 40960)  
End of epoch 1 / 5 Time Taken: 1054 sec  
learning rate 0.0002000 -> 0.0002000
```

测试命令：

```
bash test_adaattn.sh
```

```
isTrain: False  
load_iter: 0  
load_ratio: 1.0  
load_size: 512  
max_dataset_size: Inf  
model: adaattn  
n_layers_D: 3  
name: AdaAttn  
nDf: 64  
netD: basic  
netG: resnet_9blocks  
nGf: 64  
D:\python3.8\lib\site-packages\torchvision\transforms\transforms.py:332: UserWarning: Argument 'interpolation' of type int is deprecated since 0.13 and will be removed in 0.15. Please use InterpolationMode enum.  
warnings.warn([test] dataset [UnalignedDataset] was created  
initialize network with normal  
initialize network with normal  
initialize network with normal  
model [AdaAttnModel] was created  
loading the model from ./checkpoints/AdaAttn/latest_net_decoder.pth  
loading the model from ./checkpoints/AdaAttn/latest_net_transformer.pth  
loading the model from ./checkpoints/AdaAttn/latest_net_adaattn_3.pth  
----- Networks initialized -----  
[Network decoder] Total number of parameters : 4.095 M  
[Network transformer] Total number of parameters : 9.067 M  
[Network adaattn_3] Total number of parameters : 0.468 M  
-----  
creating web directory: ./results/adaattn/test/latest  
processing (0000)-th image.. ['20200428220829_1.png']  
processing (0005)-th image.. ['20200428220829_5.jpg']
```

### 3.4.3 图片结果展示



## 3.5 U-GAT-IT(T)

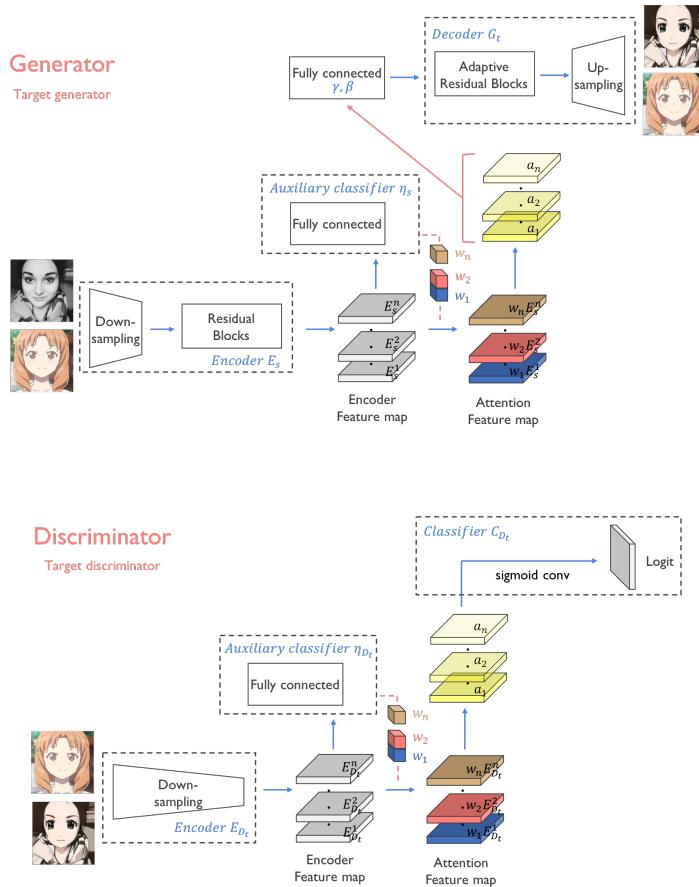
### 3.5.1 模型介绍

U-GAT-IT是一种新的无监督图像到图像转换方法，以端到端的方式结合新的注意力模块和新的可学习归一化函数。与CycleGan相似，它同样使用了gan loss, cycle loss和idt loss。不同的是，U-GAT-IT在Generator和Discriminator中添加了一个分类器，并为训练分类器设计了损失函数cam loss。同时，它的注意力模块和AdaLIN 函数帮助模型实现更好的风格迁移效果。

作为U-GAT-IT的核心：

- 注意力模块根据辅助分类器获得的注意力图，引导模型专注于区分源域和目标域的更重要的区域（帮助模型知道在哪里进行密集转换）。与之前无法处理域之间几何变化的基于注意力的方法不同，U-GAT-IT可以转换需要整体变化的图像和需要大形状变化的图像。
- AdaLIN 函数帮助注意力模型灵活控制形状和纹理的变化量，而无需修改模型架构或超参数。

U-GAT-IT 分为生成器和判别器两部分，结构几乎一致。生成器比判别器多了AdaLIN算法实现的Decoder模块。如下图：



The model architecture of U-GAT-IT. The detailed notations are described in Section Model

上图描述了U-GAT-IT网络结构。以生成器为例，输入图像通过Encoder编码阶段（下采样+残差模块）得到特征图，然后添加一个辅助分类引入Attention机制通过特征图的最大池化，经过全连接层输出一个节点的预测，然后将这个全连接层的参数和特征图相乘从而得到Attention的特征图。最后经过Decoder模块得到输出图像。

模型的目标是训练一个函数 $G_{s \rightarrow t}$ ，该函数使用从每个域中抽取未配对的样本将图像从源域 $X_s$ 映射到目标域 $X_t$ ：

- 该框架由两个生成器 $G_{s \rightarrow t}$ 和 $G_{t \rightarrow s}$ 以及两个鉴别器 $D_s$ 和 $D_t$ 组成；
- 将注意力模块集成到生成器和鉴别器中；
- 判别器中的注意力模块引导生成器关注对生成逼真图像至关重要的区域；
- 生成器中的注意力模块关注与其他域不同的区域（判别器注意力模块已经引导生成器聚焦了一个域，那么生成器的注意力模块则聚焦其它的域）。

如上文提到的，模型一共包含四个损失函数：

- 对抗损失： $L_{lsgan}^{s \rightarrow t} = (\mathbb{E}_{x \sim X_t}[(D_t(x))^2] + \mathbb{E}_{x \sim X_s}[(1 - D_t(G_{s \rightarrow t}(x)))^2])$ ，保证风格迁移图像的分布与目标图像分布相匹配；
- 循环损失： $L_{cycle}^{s \rightarrow t} = E_{x \sim X_s}[||x - G_{t \rightarrow s}(G_{s \rightarrow t}(x))||_1]$ ，保证一个图像 $x \in X_s$ ，在从 $X_s$ 到 $X_t$ ， $X_t$ 到 $X_s$ 一系列转化后，该图像能成功的转化回原始域；
- 一致性损失： $L_{identity}^{s \rightarrow t} = E_{x \sim X_t}[||x - G_{s \rightarrow t}(x)||_1]$ ，保证输入图像与输出图像的颜色分布相似，给定一个图像

$x \in X_t$ , 在使用  $G_{s \rightarrow t}$  翻译之后, 图像不应该改变;

- 分类激活映射损失:  $L_{cam}^{s \rightarrow t} = -(E_{x \sim X_s}[\log(\eta_s(x))] + E_{x \sim X_t}[\log(1 - \eta_s(x))])$ , 辅助分类器  $\eta_s$  和  $\eta_{D_t}$  带来的损失。  
 $L_{cam}^{D_t} = E_{x \sim X_t}[(\eta_{D_t}(x))^2] + E_{x \sim X_s}[(1 - \eta_{D_t}(G_{s \rightarrow t}(x)))^2]$

最后, 联合训练编码器、解码器、判别器和辅助分类器以优化最终目标函数:

$$\min_{G_{s \rightarrow t}, G_{t \rightarrow s}, \eta_s, \eta_t} \max_{D_s, D_t, \eta_{D_s}, \eta_{D_t}} \lambda_1 L_{lsgan} + \lambda_2 L_{cycle} + \lambda_3 L_{identity} + \lambda_4 L_{cam} \quad (9)$$

其中,  $\lambda_1 = 1$ ,  $\lambda_2 = 10$ ,  $\lambda_3 = 10$ ,  $\lambda_4 = 1000$ ,  $L_{lsgan} = L_{lsgan}^{s \rightarrow t} + L_{lsgan}^{t \rightarrow s}$ ,  $L_{cycle} = L_{cycle}^{s \rightarrow t} + L_{cycle}^{t \rightarrow s}$ ,  
 $L_{identity} = L_{identity}^{s \rightarrow t} + L_{identity}^{t \rightarrow s}$ ,  $L_{cam} = L_{cam}^{s \rightarrow t} + L_{cam}^{t \rightarrow s}$ 。

### 3.5.2 训练/测试代码讲解与超参数设置

由于具体网络代码过于复杂, 不便展示, 这里对模型训练代码进行讲解:

首先是Generator与Discriminator训练封装部分:

```
class TrainOneStepCellGen(nn.Cell):
    """Encapsulation class of UGATIT generator network training."""
    def __init__(self, generator, optimizer):
        super().__init__()
        self.generator = generator
        self.opt = optimizer
        self.weights = optimizer.parameters
        self.grad = ops.GradOperation(get_by_list=True)

    def construct(self, real_A, real_B):
        weights = self.weights
        loss = self.generator(real_A, real_B)
        grads = self.grad(self.generator, weights)(real_A, real_B)
        self.opt(grads)
        return loss

class TrainOneStepCellDis(nn.Cell):
    """Encapsulation class of UGATIT discriminator network training."""
    def __init__(self, discriminator, optimizer):
        super().__init__()
        self.discriminator = discriminator
        self.opt = optimizer
        self.weights = optimizer.parameters
        self.grad = ops.GradOperation(get_by_list=True)

    def construct(self, real_A, real_B):
        weights = self.weights
```

```

loss = self.discriminator(real_A, real_B)
grads = self.grad(self.discriminator, weights)(real_A, real_B)
self.opt(grads)
return loss

```

这里输入的real\_A, real\_B分别是内容图片与风格图片。模块初始化输入的generator与discriminator分别为输出对应损失的计算网络(NetWithLoss)，而optimizer为载入了对应模型各个网络参数的优化器。具体可以看到训练过程如下：

```

real_A = A["sample"]
real_B = B["sample"]

# train step

# Define network with loss
G_loss_cell = GenWithLossCell(self.genA2B, self.genB2A, self.disGA, self.disGB, self.disLA,
self.disLB, self.adv_weight, self.cycle_weight, self.identity_weight, self.cam_weight)
D_loss_cell = DisWithLossCell(self.genA2B, self.genB2A, self.disGA, self.disGB, self.disLA,
self.disLB, self.adv_weight)

# Define One Step Train
G_trainOneStep = TrainOneStepCellGen(G_loss_cell, self.G_optim)
D_trainOneStep = TrainOneStepCellDis(D_loss_cell, self.D_optim)

# Train
G_trainOneStep.set_train(True)
D_trainOneStep.set_train(True)

G_loss = G_trainOneStep(real_A, real_B)
D_loss = D_trainOneStep(real_A, real_B)

```

其中“A”与“B”为对应数据集，`GenWithLossCell` 与 `DisWithLossCell` 即之前提到的计算网络。这其中涉及到六个更为具体的网络：`self.genA2B`，`self.genB2A`，`self.disGA`，`self.disGB`，`self.disLA`，`self.disLB`。其中 `self.genA2B` 与 `self.genB2A` 为内容图像->风格图像与风格图像->内容图像的生成器，它们完全相同，区别只在于输入不同；`self.disGA`、`self.disGB`、`self.disLA`、`self.disLB` 为对应生成图片的判别器，对应模型介绍部分的 $D_s$ 和 $D_t$ 。与生成器类似，它们的网络结构均完全相同。两个计算网络最终输出的Generator\_loss与Discriminator\_loss为模型介绍部分几个损失的加权求和。

对于具体的代码，UGATIT部分代码中，network.py包含了生成器与判别器在内的所有具体网络；dataset.py用于构建数据集；而UGATIT.py则是整个模型的主体部分，包含整体构架与训练测试流程；main.py主要涉及启动代码以及参数配置。

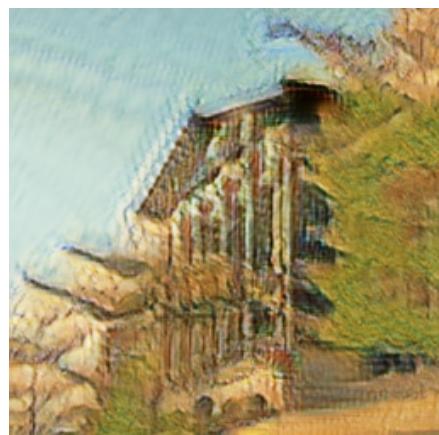
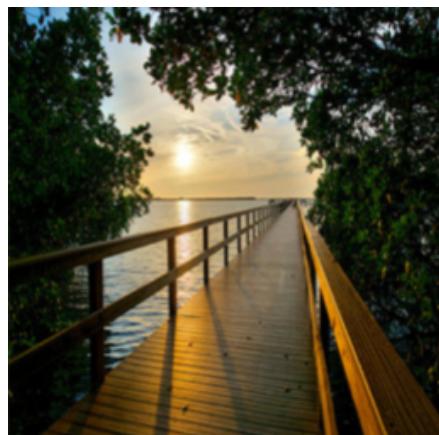
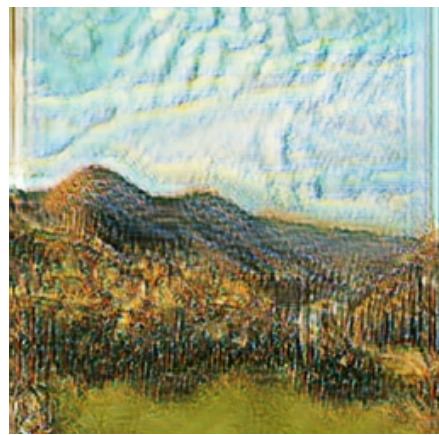
下面是训练用到的超参数配置：

image_size	batch_size	learning rate	epoch
(256, 256)	2	1e-4	100000

由于模型过于庞大，即使是在简化状态下，也只能支持batch\_size为2的训练。如果在硬件条件许可下需要更复杂的模型，可以在main.py将“`--light`”部分参数默认设为`False`。

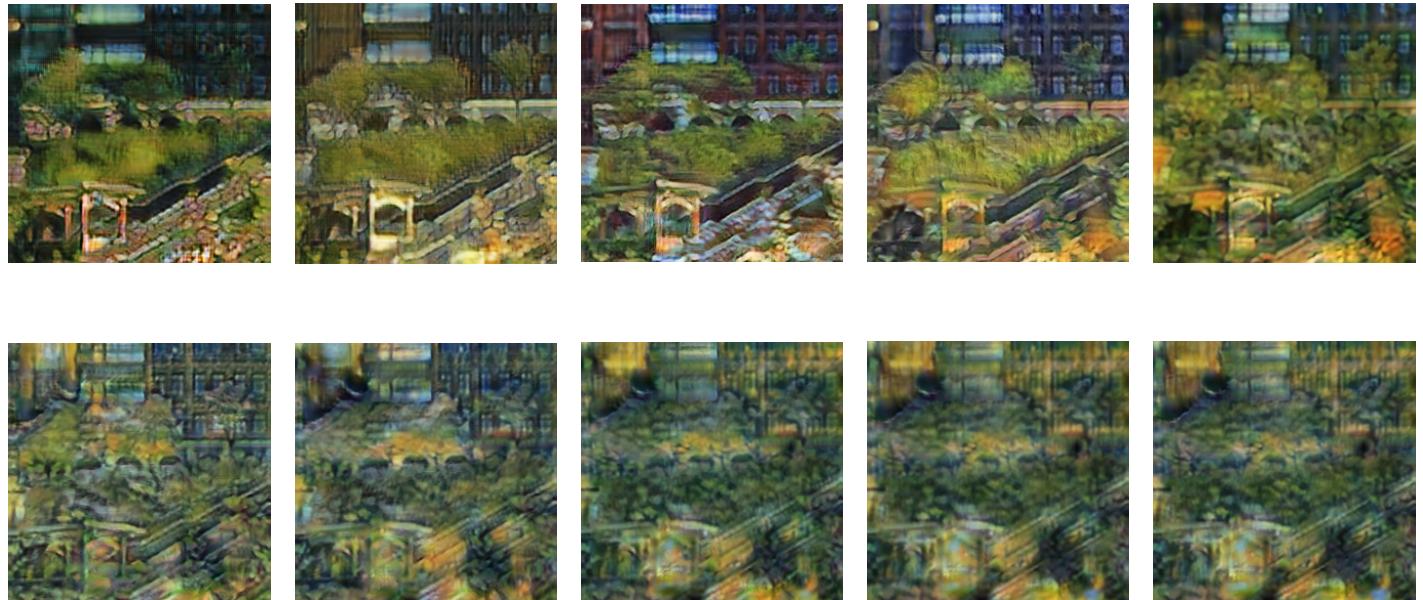
### 3.5.3 图片结果展示

得到一组测试结果如下：



这是epoch=20000时生成的图片。可以看到，模型成功将内容图像转换到了风格图像对应的风格。需要注意的是，即使是同一个风格的风格图像，其不同的色调等特征，同样也会影响到生成图像的具体风格。UGATIT的注意力模块在此体现在对内容图像特征的抓取。例如上图第二组中，生成图像同样很好地展现了日落、黄昏的特征。但略微遗憾的是，上述生成图像均较为模糊。虽然这也是梵高风格的部分特色，但不够细腻的图像仍然对观感有一定影响。

同时可以看到当epoch分别为10000、20000、30000、40000、50000、60000、70000、80000、90000、100000时保存模型对同一张内容图像的风格迁移效果：



可以看到随着训练加深，生成图像在细节上的内容丢失越来越多，风格化越来越明显。但当epoch在50000以后，生成图像综合质量明显变差，到epoch=10000时生成图像几乎变成一团揉乱的色彩。出现这种情况的主要原因在于训练数据集过小。由于资源限制，模型训练数据集仅为50张图像，这导致模型没能更深层次地学习到风格特征。

此外，训练数据集的内容对不同类型的风格迁移效果同样有很大影响。如此次训练内容图像均为风景画，这使得测试时人物图像的风格迁移效果较差。

训练过程中包含热力图在内部分图像以及测试生成图像均保存在[UGATIT/result](#) 中。

## 4 总体展示

### 4.1 模型结果对比



### 4.2 量化结果

我们采用经典的FID (Fréchet Inception Distance) , 计算了真实图片和风格图片在 feature 层面的距离, FID越小代表生成效果越好, 计算公式如下:

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g + (\Sigma_r + \Sigma_g)^{\frac{1}{2}}) \quad (10)$$

测试数据集地址为[link](#), 一共有1000张图片。

可以直接安装pytorch-fid来得到FID的分：

```
pip install pytorch-fid
```

使用如下命令即可：

```
python -m pytorch_fid path1 path2
```

结果如下：

CNN	Arbitrary	CycleGAN	AdaAttN	U-GAT-IT
369.4580	223.5860	203,4569	327.0946	351.7662

## 5 Conclusion

综上所述，我们从最一开始的CNN开始，之后学习了多种模型来实现图像梵高风格的转换，对整个的风格转换这个领域有了更加深刻的理解，对于我们之后不管是进行这方面的研究或者进行其他生成任务，视觉任务都会有所启发。

小组分工：

- 孙广岩：CNN, CycleGAN, 整体报告整合
- 宋昕帅：CNN, U-GAT-IT
- 李懿展：ArbitraryStyleTransfer
- 李云飞：AdaAttN
- 刘丁烨：AdaAttN

参考资料：

- [1] [梵高数据集](#)
- [2] [使用卷积神经网络进行风格迁移](#)
- [3] [Exploring the structure of a real-time, arbitrary neural artistic stylization network](#)
- [4] [ArbitraryStyleTransfer](#)
- [5] [Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks](#)
- [6] [CycleGANhttps://arxiv.org/abs/1907.10830\)](#)
- [7] [CycleGAN Explained](#)
- [8] [AdaAttN: Revisit Attention Mechanism in Arbitrary Neural Style Transfer](#)
- [9] [AdaAttN](#)
- [10] [Unsupervised Generative Attentional Networks with Adaptive Layer-Instance Normalization for Image-to-Image Translation](#)
- [11] [U-GAT-IT](#)