Summary

#CV 经典网络

在经典网络中包含 AlexNet、VGGNet 和 ResNet

AlexNet 算是一种开头，不过在之后的领域中不算常用

VGGNet 和 ResNet 则是在 AlexNet 的基础之上进行了拓展，良好的性能让它们成为了图像领域的 BackBone

> BackBone 就是领域的骨架或者说基石

它们都采用了更深的网络来进行图像识别，不过网络越深就意味着训练的难度也会上升，为了更好的训练效果，它们分别采取了不同的策略

- 在 VGGNet 中，利用较小的卷积核（3by3）来替代较大的卷积核（7by7），虽然卷积核的感受野小了，不过可以有效地减少参数数量，另外，连续堆叠的 `3*3` 卷积核在最终的效果上等效于一个 `5*5` 的卷积核
- 在 ResNet 中，利用了残差连接来避免难以优化（梯度消失）的问题，具体来说就是在一个残差块中，将输入加在卷积层的输出上，需要注意的是将输入转换为可以加在输出上的形式也有不同的策略。至于为什么采用残差方式，从反向传播的角度来看似乎是可以让梯度跨层传播从而避免梯度衰减的问题

> 这两种设计方法在之后的论文中很常用

从代码实现的层面来看，我们必须学会如何有效地抽象出网络中相同的部分，从而有效地避免编写冗余代码（如果你直接看别人的源码的话直接忽略就好了）

```python
from torch import nn

class ResBlock(nn.Module):
```

```
    def __init__(self, ...):
        self.conv...
        self.activation...
        self.norm...

    def forward(self, x):
        identity = x
        ...
        return out + iendtity

class ResNet(nn.Module):
    def __init__(self, ...):
        self.entity = nn.Sequential(
            ResBlock(...),
            ResBlock(...),
            ...
            ResBlock(...),
        )

    def forward(self, x):
        return self.entity(x)
```

风格迁移

在风格迁移中包含 Neural Algorithm、AdaIN 和 Perceptual Loss，它们应该仅仅算是风格迁移中的一小部分，且它们之间的关系也是递进式的

在 Neural Algorithm 也算是利用深度神经网络来实现风格迁移的开山之作了吧。这篇论文的思想也很简单，就是利用预训练好的 VGGNet 在进行图像识别时中间层所抽取的特征图来计算损失，优化的对象是初始为噪声的一张图像（也就是像素），图像的风格从读取风格图像的 VGGNet 的低层特征图中抽取，图像的内容从读取风格图像的 VGGNet 中的高层特征图中抽取，在将被优化的图像丢到 VGGNet 中，利用各层的特征图和相应的风格特征或内容特征以 Gram 矩阵的形式来计算它们之间的损失，从而优化图像

> 不过 Neural Algorithm 有一个很致命的缺陷就是合成一张图像用的时间太久了，效果还是蛮不错的

在 Perceptual Loss 中，作者训练了一个 U-Net 来实现风格迁移，然后损失的计算也采用类似于 Neural Algorithm 的方法，用一个预训练的 VGGNet 来抽取内容、风格以及由网络生成的图像的特征图，再利用 Gram 矩阵来计算它们之间的差距作为损失从而优化网络参数。简单地说

就是从 Nerual Algorithm 的优化一张图像转换为了优化一个用来生成图像的网络。

> 虽然在训练好一个迁移网络之后，生成一张图像的时间确实快了不少，不过每个网络仅可以用来迁移一个风格图像，当我们想迁移另一种风格时，就必须再训练一个网络了

在 AdaIN 中，采用 Encoder-Decoder 的架构，Encoder 分别读取内容图像和风格图像，将它们的高层特征图利用 AdaIN 整合在一起（确切地说是将内容特征所在的分布迁移到风格特征所在的分布上），再利用解码器将整合起来的特征图解码为风格迁移之后的图像。至于损失则沿用了 Neual Algorithm 的方式，不过内容损失的来自于整合之后的特征图，所以说优化的对象是 Decoder，而 Encoder 始终是预训练好的 VGGNet，值得注意的是，用来抽取特征图的网络也是 Encoder。

AdaIN 可以迁移任意类型的风格，而不用再训练一个新的网络，就其原因就是 AdaIN 的功劳了，它将风格迁移这一任务利用 AdaIN 解决了，而不再需要劳驾网络在训练时来学习如何实现此功能，从而解放了网络，让网络可以专注于解码的任务。

AdaIN 的功能性不言而喻，所以之后经常被用来作为其它网络的某个部分来实现风格迁移的功能

> AdaIN 包含了 Neural Algorithm 的风格转化功能，也可以训练一个网络很快的实现此功能，最关键的还是任意风格的转换

实际上，仅从这 3 篇论文就可以窥探到科技发展的方向和规律。从代码实现的角度来看，复现这 3 篇用了大约 20 天左右，大约 15 天左右全都用在了 DeBUG 上

Nerual Algorithm 还好，这篇的复现我主要参考了 PyTorch 官网教程中的例子，在官网教程的实现中以动态构建网络的方式来抽取特征不同类型的特征，参考代码一步步写下来的话复现还是很简单的

卡就卡在了后两篇上，通过上面的论述不难发现每个网络中都有抽取特征计算损失的成分，所以不出意外的后 2 篇我都用了动态构建网络的方式来计算损失，但是不论怎么调试结果都是不对的，最后还是参考源码，采用控制变量法找出了原因所在，以下是结论

- 在 Perceptual Loss 中动态构建网络本身不存在问题，问题在于图像像素值没有处理好的原因，在利用 Pytorch 自带的 Transforms 的 ToTensor 方法的话，像素值是会自动映射到 -1 到 1 的，所以生成的图像像素值也是在 -1 到 1 的，所以生成之后的图像必须将它们映射

回原来的 0 到 255 才可以正确的显示，所以如何是处理的图像的话，应当多注意一下像素值的范围

- 在 AdaIN 中就是动态构建网络本身的问题了，因为我发现用静态的网络来抽取特征时，网络是可以正确生成图像的，为了找出为什么动态构建网络不可行的原因，我甚至利用某个库绘制了它们（只有动态和静态网络的区别）的梯度传播图，发现它们是一模一样的，所以我也不知道为什么动态构建网络的问题究竟在哪里了。总之凡是抽取特征的部分尽量避免用动态构建的网络就是了

在这一部分的代码实现方面，我们必须学会如何应用 Pytorch 的灵活性——不影响输入和输出格式的前提下，网络中间层的输出格式并不一定必须等于下一层的输入格式，你可以输出你想要的任何数据，不论是在形式上还是数量上，而在传递到下一层的输入也同样是可以选择的。总之，只要最后能反向传播起来，你中间干啥都行

生成对抗

在生成对抗部分包含 GAN、DCGAN、StyleGAN 和 StyleGAN2

GAN 算是“最出圈”的生成式网络了，在 GAN 中包含两个子网络，分别是生成器（Generator）和判别器（Discriminator）。生成器（一个神经网络）用来将一个来自简单且已知分布（往往是标准正太分布）的样本数据转换至另一个复杂且未知（无法形式化表示出来）的分布中；判别器（一个神经网络）用来分辨某个样本数据来自于数据集还是由生成器生成的。通常情况下，我们希望生成器所转换出来的分布和数据集中数据所在的分布是一致的（最起码得是相似的）。

GAN 的训练损失从公式上来看也是比较矛盾的

$$G^* = arg\min_G(\max_D V(G,D))$$

$$V(D,G) = E_{y\sim P_{data}}[logD(y)] + E_{y\sim P_G}[log(1-D(y))]$$

不过从这个公式上我们可以更好的理解 GAN 的原理：我们想让 Generator 最小化 P_G 和 P_{data} 的差距，而此差距由 Discriminator 相关的优化问题计算出来，此优化问题即如何提升 Discriminator 分辨真假（分类）的能力

从代码实现的角度来看的话，GAN 本身有一点古老了，所以我用代码实现了 DCGAN，和 GAN 从概念上差别不大，可以理解为采用了卷积层的 GAN。具体的代码参考了 Pytorch 官方示例的 DCGAN 实现。GAN 本身就比较难训练起来，而 PyTorch 官方的 DCGAN 示例中应用到了很多值得学习的 Tricks，是很不错的入门级学习材料

- 标准化
  - 将 Discriminator 的输入标准化至 -1 到 1 之间
  - 在 Generator 的最后一层利用 `Tanh` 激活
- 目标函数
  - 若 Generator 的目标是最小化 $log(1 - D(G(z)))$ 将很容易出现梯度消失
  - 所以我们将 Generator 的目标调整为最大化 $log(D(G(z))$
- 采样
  - 从高斯分布中采样，而不是均匀分布
  - 在噪声之间插值时应采用大圆弧（great circle）的方式而不是简单的直线
- 批归一化
  - 在训练 Discriminator 时，之干不同的真和伪数据构造不同的小批量
  - 在批归一化不可用时，采用实例归一化
- 梯度
  - 避免稀疏梯度 `ReLU` 和 `MaxPool`
  - 在 Generator 和 Discriminator 中采用 `LeakyReLU`
  - 通过 `AvgPool` 或 `Conv2d+Stride` 实现下采样
  - 通过 `PixelShuffle` 或 `ConvTranspose2d+Stride` 实现上采样
- 优化器
  - 在 Discriminator 中应用 SGD

- 在 Generator 中应用 `Adam`

- 模型
  - 在 Generator 的某些层中以 `Dropout` 的形式补充额外的噪声

StyleGAN 和 StyleGAN2 是基于 GAN 的人脸图像生成网络。

在 StyleGAN 中，判别器还是比较常规的，关键之处在于生成器 。不同于以往的生成器，StyleGAN 中的生成器由 2 个子网络组成，分别是映射网络和合成网络。不能在训练还是推理阶段，生成器的工作机理是这样的，首先随机生成一个样本（隐编码），将它丢到映射网络（8层全连接网络）中得到一个特征编码；然后由合成网络从一个固定分辨率的噪声图像开始逐步上采样到目标分辨率大小。在上采样的过程中，将特征编码利用一个简单的网络转换为合适的形式，再利用 AdaIN 嵌入到某个特征图上，此外，还生成随机的噪声并同样利用一个简单的网络转换为合适的形式嵌入到特征图中。特征编码的嵌入自然是为了让我们有能力控制生成器生成图像的过程，而噪声的嵌入则是为了人为地制造一些随机性在里面。

每个人脸都对应一个独一无二的特征编码，如果我们将两个特征编码以某种组合方式嵌入到合成网络的不同层级中，那么最终生成的人脸将同时拥有两个人的特征的，而倾向性则依赖于两种特征嵌入的位置，此即为 StyleMixing

> 除此之外，StyleGAN 中还有新的衡量插值质量的方法以及 FFHQ 的介绍，感觉暂时用不上就每有仔细了解了

StyleGAN2 则是为了解决 StyleGAN 生成伪影的问题以及提升生成图像的质量，作者发现产生伪影的“罪魁祸首”就是 AdaIN 中关于均值和方差在特征图上的标准化操作，为了解决这个问题呢，在合成网络中有如下修改：

1. 对于 AdaIN 来说不再有均值的参与，只有方差的参与
2. 将偏差和噪声的嵌入移到了 StyleBlock 之外

老实说，上述第 2 点我认为在里面在外面都一样，不再讨论了，着重说明第 1 点。如果应用 AdaIN 时只有方差参与的话，那就相当于在特征图除以了标准方差（解调制）之后，特征图乘了某个因子（调制）呗。那我们是否可以直接将这个应用在特征图上的因子应用在卷积层的权重上呢，如果我们将权重也看作乘在特征图上的一个因子，最终的效果应该是相同的，另外，理所应当的我们也必须在权重上除以相应的标准差来模拟解调制的过程。如此一来，我们就将之于特征图的调制/解调制过程转移到

了之于卷积层权重的调制/解调制过程。最后的效果和原来相比也是蛮优雅的。

> 除了优雅之外，还有一个原因，我不是很理解，作者在原论文中提到...

除了这个之外，作者提到类似于 ProgressiveGAN 的训练方式也有伪影产生，原因是细节的强局部表现力太过火了，所以作者利用了跳跃连接和残差连接的方式替换了此方法（这时我们就可以采用正常的训练方法来训练 StyleGAN 了），同时，判别器生成的图像也是由不同分辨率的卷积层所产生的特征图转换为 RGB 图像之后累加的结果，判别器也类似，不再赘述。通过实验的结果来看，在生成器中使用跳跃连接的方式效果最好，而判别器中则是残差连接的效果最好

注意力机制

在注意力机制部分包含 Attention 和 ViTransformer。Attention 机制原本是用来解决自然语言处理相关的问题的，不过随着大牛们的开发，Attention 机制也被广泛地应用在其它各种领域，比如说 ViTransformer 就是用在图像领域的

不是搞自然语言处理的，所以就简单说明一下 Attention 机制中 Attention 块内部的处理流程以及底层的原理。对于以组织为一系列向量的输入序列，分别乘上 3 个可学习矩阵 Q、K 和 V，得到和每个向量相关的查询（qi）、键（ki）以及值（vi），让每个向量的查询分别乘上包括自己在内的其它所有向量的键来得到相关性系数（alpha），相关性系数也必须以类似于 Softmax 方式标准化一下（赋予权性），再将标准化之后相关性系数乘上各自相应的向量的值再求和就可以得到每个向量分别考虑了全部向量之后的结果。每个向量相应的结果都必须再经过 MLP 处理一下才是每个 Attention 块最终的输出结果。必须注意的是在注意力机制和 MLP 之间都有残差连接的处理

> 在序列中的每个向量上嵌入位置编码可以让注意力机制在处理时考虑到顺序的因素

在某些时候，输入序列中的相关性或许体现在不同的方面，此时唯一的一组查询、键和值或许不可以很好地表征向量之间的关系，所以之于每个序列，我们可以同时有多个相互独立的查询、键和值，也就是说有多个相互独立的可学习矩阵组，每个矩阵组构造一组查询、键和值。如此以来，每个向量将同时拥有多个相关的输出结果，此时我们可将它们连接起来得到一个较长的向量，之后再和某个可学习的转化矩阵相乘得到最终的结果

从函数的角度去说明的话，就是自注意力机制所表示的函数集合本身就是大于 CNN 所可以表示的函数集合，所以自注意力机制的函数拟合能力肯定是比 CNN 强很多的，不过也就意味着自注意力机制比 CNN 更难训练，需要更多的数据集才有可能找到不错的结果。具体来说，CNN 中的卷积核所考虑的范围往往是规定大小的方块（也就是卷积核），所以相关性总是局限在特定的范围之内，而之于考虑所有向量自注意力机制来说，考虑哪些范围是由模型自己学习到的，而且不存在范围的限制

为了确保自己理解了注意力机制，用代码实现一下是最好的检验方式，不过不是搞自然语言处理的，原始的 Transformer 是不好写了，所以就尝试实现了一下 ViTransformer

ViTransformer 是自注意力机制在图像领域的应用吧，任务的话还是用来分类图像，大致思路就是将一张图片划分为不重叠的 Patch（利用一个特定大小核步长的卷积核处理一下，得到的特征图就是每个 Patch 了），再将每个 Patch 调整成向量的形式，拼接一个最后用来分类的 Class Token，再给所有的向量嵌入位置编码然后送到多个多头注意力机制块中处理即可。我们希望图片用于分类的所有信息都存储在 Class Token 所对应的最终结果上，所以我们最后取它的输出结果，丢到一个 MLP 里面类似于以往的方式分类就可以了

从代码实现的角度来看，也不是很难写，就是比较容易被变来变去的维度搞晕，不过在草稿纸上画一下，搞清楚每一次处理的维度变化也是很容易的，有一个值得提到的技巧就是利用 Chunk 可以少写很多循环（具体的我也有一点忘了，太长时间没碰了）

点云

分类

在点云分类包含 PointNet 和 PointNet++，它们应该算是点云处理领域的 BackBone

PointNet 实际上也没什么好说的，本质上还是按照深度学习（一维卷积、共享参数）那一套去处理三维点云数据，另外，为了适应点云数据的特征（无序性和稳定性等），在 PointNet 中在利用若干个小型网络去转换点云数据（实现对齐的同时确保稳定性），利用最大池化函数（从每个孤立的点中）提取全局特征，利用其对称性来避免点云数据无序性带来的影响

PointNet++ 是 PointNet 的迭代版本，和之前相比着重考虑了点云数据的局部结构，主要的方法就是类似于 CNN 中的卷积核在图像上的应用。由于点云数据的特殊性，在 PointNet++ 中，一个集合抽象层级（Set Abstraction Levels）相当于 CNN 中的一个卷积层，在集合抽象层级中

包含采样层、聚集层和简单的 PointNet，采样层和聚集层合起来做的事情就是用最远点采样确定整个点云数据中合适的“卷积核中心”位置，然后以每个卷积核中心为球心利用球采样方法来采样周围的固定数量的点，之后再利用 PointNet 对每个区域（卷积核中心及其周围的点）编码。剩余的就跟一个卷积网络差不多了，就不再赘述了

生成

SPGAN 从功能和架构上给我的感觉其实很像 StyleGAN，不过根据点云数据自身的特征来设计网络的话结构上还是有一些不同的。在 Generator 部分，网络的输入有两个，分别是全局先验球（S，以矩阵的形式来表示），和以特定的方式附着了隐编码（z）的先验矩阵

> 感觉就像是 StyleGAN 上生成网络上 4 * 4 大小的特定输入以及隐编码 z

然后利用一个 MLP（Feature Embedding）来处理先验矩阵，得到类似于 StyleGAN 中的特征编码 w，然后再让这个 w 经过（可能有多个）各自独立的 MLP（Style Embedding）来处理得到可以用于 AdaIN 进行特征迁移的两个矩阵，这一条线就类似于 StyleGAN 中的映射网络吧

之于全局先验球（S）这一条线来说，类似于 StyleGAN 中的生成网络，网络中采用了 Graph Attention Module 来处理全局先验球，每经过一个 GAM 处理之后，就利用 AdaIN 嵌入特征。最后在以类似于 PointNet 中的处理方式捕获全局特征等，最后生成一个和全局先验球相同点个数的目标点云

> 和 StyleGAN 的不同之处可能在于 StyleGAN 依赖于上采样生成，而这个则依赖于将先验球上的每个点移动到特定的位置来生成

Discriminator 感觉也没什么，不过就是除了整个形状的分数之外，还有一个可选的（作者源代码里面这部分是默认关闭的）和每个点相关的分数。至于特征提取的过程则是以 PointNet++ 为 BackBone 的提取方式了。还有一个就是损失也不太一样了，不过对于 GAN 来说，不论损失函数的形式是怎么样的，最终的目的都是一样的，所以就不再赘述了

还有一个是 Point-StyleGAN，有了前面的铺垫，理解起来也很简单，网络的整体架构基本上和 StyleGAN 差不多，不过在 Discriminator 部分，可能是为了模拟图像逐步下采样的过程，所以有了很多个 Discriminator。另外损失的话有两部分组成，其中一个是和 SPGAN 一样的（没有和每个点相关的分数），还有一个是和 KL 散度相关的...