Image-to-image translation Documentation

2019 年 8 月 8 日

目录

1 我们要做什么？

1.1 I2I(image to image)

广义上，I2I(image to image) 泛指所有满足以下要求的任务：输入为一个图像，输出为符合要求的另一个图像。在两个图像中，有部分信息保持不变，正如无论是用“苹果”还是“apple”来表示，所指代的对象都是相同的事物。当前的 I2I 任务，主要包括风格迁移、图像降噪、超分辨率、语义分割等。本次实验中我们关注的是图像风格迁移及其中间过程。一个典型的图像风格迁移的案例，是将图片中的马识别出来并转换为斑马，或反向进行相同的转换。

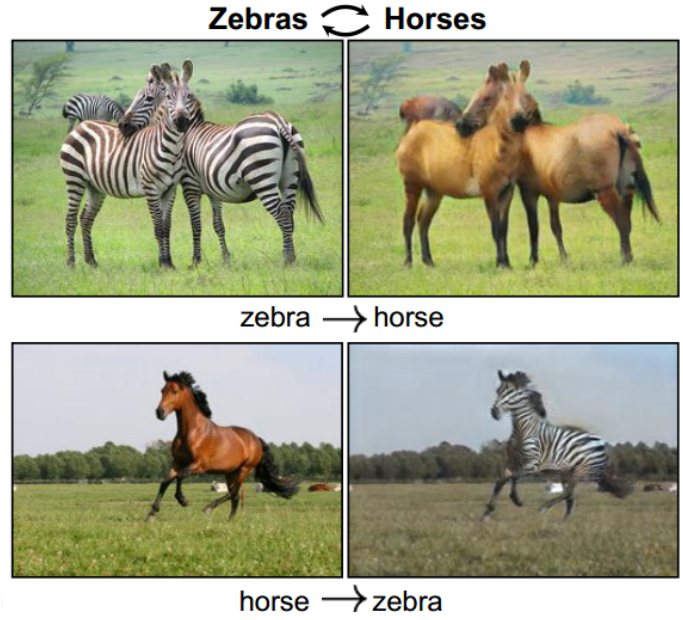

图 1: 风格迁移, 2017

1.2 一些基础的知识

1.2.1 插值算法

既然是从一个状态到另一个状态，一个显然的思路是量化两个状态，再通过数值的中间值反向得到中间状态。获得数值中间值的方法就是插值。简单讲，插值就是根据已知数据点（条件），来预测未知数据点值的方法。最简单的插值就是线性插值，即根据状态 x_1, x_2 和一个表

示中间值相对位置的值 $\varepsilon(\varepsilon \in [0,1])$ 得到中间值 $\varepsilon x_1 + (1-\varepsilon)x_2$。除此之外，常用的插值还有双线性插值、Slerp(spherical linear interpolation) 等方法。

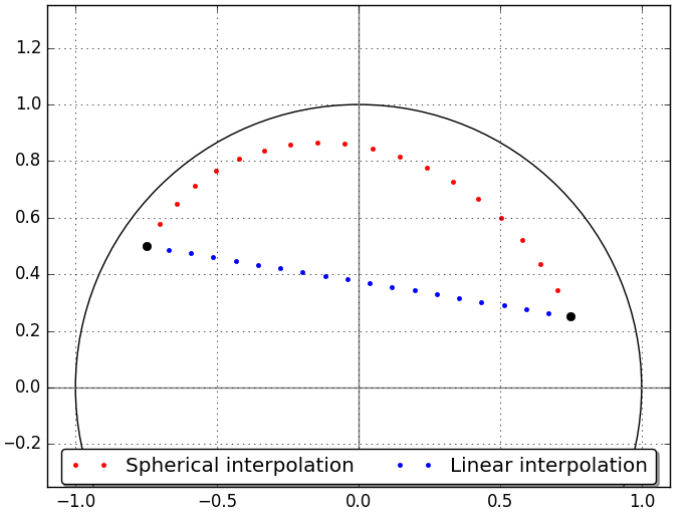

图 2: 常用插值方法，在两个黑色点之间作插值

1.2.2 同态

与插值算法相对应的一个概念是隐空间 (latent space)。正如前文所述，在状态过渡时，我们需要先量化两个状态，再进行插值。任意一个状态 Σ 都能被量化为一个对应向量 t，这些向量 t 的全体就分布在对应的隐空间中。状态间的过渡也可以用以下步骤进行描述：

1. 从起始状态映射至隐空间；
2. 在隐空间中进行过渡（插值）；
3. 从隐空间映射至最终状态。

显然，状态与隐空间中有意义的点之间是一一对应的关系，在拓扑上，这种关系也被称为同态。

1.2.3 卷积

如果直接使用插值，常常出现我们不想看到的一种情况：如果将一个人的脸转换为另一个人，在图像里两张人脸有略微的不对齐时，中间过程里会出现若隐若现的四个眼睛！这是由于我们的插值算法无法辨别出两张图片里五官的位置。显然，我们希望我们的程序能“聪明”地识别出图像中内容的特征信息。

这些特征信息有什么特点呢？首先，像五官这样的信息在图片里往往是连续的像素点，此外，眼睛、头发等特征的颜色往往与皮肤的颜色有很大区别。因此，我们可以将图片中相邻一片像素点的颜色值（RGB 值）作为一个可能的特征。卷积正是进行了这样的操作。在卷积过程中，我们取一个较小的矩阵，例如 3×3，然后赋予这个矩阵每个点一个值。随后，我们将这个矩阵叠在图片上，将每一对重叠的数字相乘以后求和，就得到了卷积的值。而上述的矩阵，被我们称为卷积核。

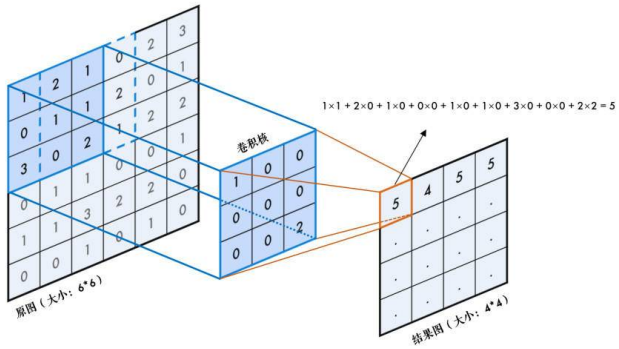

图 3: 卷积

卷积可以提取出抽象的特征，通常，我们认为一次或连续多次优秀的卷积，能够将输入的内容映射至前文所述的，包含所有可以表示特征的向量的隐空间里。

1.2.4 神经网络

在计算机领域，神经网络往往是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应。

神经网络中最基本的成分是神经元模型，即上述的“简单单元”。在神经网络中，每个神经元与其它神经元相连，当它“兴奋”时，就会向相连的神经元发送信号；当某个神经元接受

到的信号超过一个“阈值”时，它就会被激活，即“兴奋”起来。这一模型也被称为“M-P 神经元模型”。

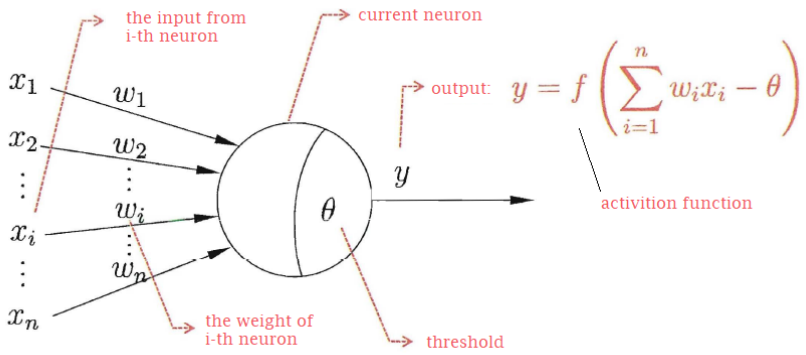

图 4: M-P 神经元模型，周志华《机器学习》

理想的激活函数应当形如阶跃函数，即在输入大于某个值，神经元完全兴奋，否则完全不兴奋。然而，由于阶跃函数有不连续、不光滑等性质，因此实际常使用一些连续的，形状与阶跃函数相似的函数作为激活函数，如 Sigmoid 函数。由于它将较大范围内变化的输入值挤压到 $[0, 1]$ 输出值范围内，因此有时也称其为“挤压函数”。常用的激活函数还包括 ReLU 函数及其衍生等。

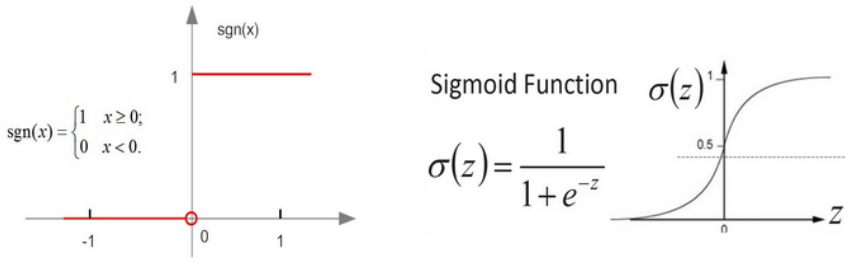

图 5: 激活函数，左为阶跃函数，右为 Sigmoid 函数

将许多个这样的神经元按照一定的层次结构连接起来，就得到了神经网络。若神经元分属不同层次，每一层都只接收前一层的信号，并将信号传递至后一层，则称其为前馈神经网络。前馈神经网络有许多优秀的模型，通常我们说“神经网络”时，指代的就是前馈神经网络。前

馈神经网络中，在输入层与输出层之间的神经元层次，被称为隐层或隐含层。

$$y = (f^3(f^2(f^1(x)))$$

$$f^1(x) = \sigma(W^1 f^{1\,-\,1}(.))$$

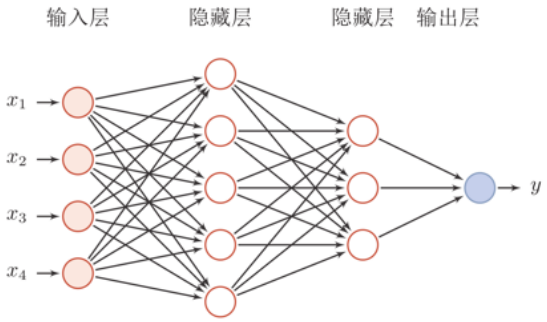

图 6: 多层前馈神经网络

根据 Hornik 和 Cybenko 提出的万能近似定理，一个前馈神经网络中，如果具有线性输出层和至少一层具有“挤压”性质的隐层，则只要这个神经网络有足够多的隐藏单元，就能以任意精度来近似任意一个从有限维空间到有限维空间的 Borel 可测函数。因此，从理论上，神经网络可以很好地拟合得到几乎所有目标函数。

1.2.5 误差逆传播与梯度下降

如何对神经网络中繁多的参数进行更新呢？目前常用的方法被称为误差逆传播算法，常用于多层前馈神经网络。一个简单的神经网络如上图所示，包含输入层、隐藏层和输出层。假设输入层共有 d 个结点，用 $\S_1, x_2 ... x_d$，隐层上共有 q 个结点，编号为 $\lfloor_1, b_2 ... b_q$，其阈值为 $\gamma_1, \gamma_2 ... \gamma_q$。输出层共有 l 个结点，依次为 $\dagger_1, y_2 ... y_l$，其阈值为 $\theta_1, \theta_2 ... \theta_l$。不失一般性，假设每一层都是全连接层，激活函数为 Sigmoid 函数。

为了方便书写，假设输入层至隐层的连接权为 $\sqsubset_{\rangle j}, i = 1, 2 ... dj = 1, 2 ... q$，隐层至输出层的连接权为 $\sqsupset_{\rangle j}, i = 1, 2 ... qj = 1, 2 ... l$，且假设隐层的输入为 $\alpha_i i = 1, 2, ... q$，输出层的输入为 $\beta_i i = 1, 2, ... l$。

对于一个训练例 $(\S_k, \dagger_k)$，神经网络的输出为 $\hat{\dagger_k} = (\hat{\dagger_1^k}, \hat{\dagger_2^k} ... \hat{\dagger_l^k})$。则网络的均方误差为

$$\mathrm{E}_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{\dagger_j^k} - \dagger_j^k)^2$$

任意参数 $\sqsubset$ 的更新估计式为：$\sqsubset \leftarrow \sqsubset + \Delta\sqsubset$

下面以隐层到输出层的连接权 $\sqsupseteq_{\langle j}$ 为例描述这一算法：

BP 算法基于梯度下降策略，以目标的负梯度方向对参数进行调整。对于一个给定的学习率 η（用于衡量每次梯度下降的程度，在 0 至 1 之间），有

$$\Delta\sqsupseteq_{\langle j} = -\eta\frac{\partial \mathrm{E}_k}{\partial \sqsupseteq_{\langle j}}$$

注意到 $\sqsupseteq_{\langle j}$ 先影响道第 | 个输出层神经元的输入值 β_j，再影响到其输出值 $\hat{\dagger_j^k}$，然后影响到 E_k，有

$\frac{\partial \mathrm{E}_k}{\partial \sqsupseteq_{\langle j}} = \frac{\partial \mathrm{E}_k}{\partial \hat{\dagger_j^k}} \cdot \frac{\partial \hat{\dagger_j^k}}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial \sqsupseteq_{\langle j}}$

由定义有 $\frac{\partial \beta_j}{\partial \sqsupseteq_{\langle j}} = \lfloor_h$

Sigmoid 函数在任意位置的导数均存在且可以被解出（其余连续的激活函数均满足这一要求）。有

$$\}_j = -\frac{\partial \mathrm{E}_k}{\partial \hat{\dagger_j^k}} \cdot \frac{\partial \hat{\dagger_j^k}}{\partial \beta_j} = -(\hat{\dagger_j^k} - \dagger_j^k) \cdot \{'(\beta_j - \theta_j)$$

则 $\sqsupseteq_{\langle j}$ 的更新公式为

$$\Delta\sqsupseteq_{\langle j} = \eta\}_j\lfloor_h$$

1.2.6 GAN

GAN（生成式对抗网络）当中，两个神经网络相互竞争，生成网络从隐空间中随机取样作为输入，其输出结果需要尽量模仿训练集中的真实样本。判别网络的输入则为真实样本或生成网络的输出，其目的是将生成网络的输出从真实样本中尽可能分辨出来。而生成网络则要尽可能地欺骗判别网络。两个网络相互对抗、不断调整参数，最终目的是使判别网络无法判断生成网络的输出结果是否真实。进一步细分，生成网络还可以分为三部分：提取特征（Encode），中间过程和翻译特征（Decode）。

2 最初的尝试：直接使用插值

当图片完全对齐时，使用线性插值的结果如下（最左、最右侧的图片分别是起始、目标图片，下同）：

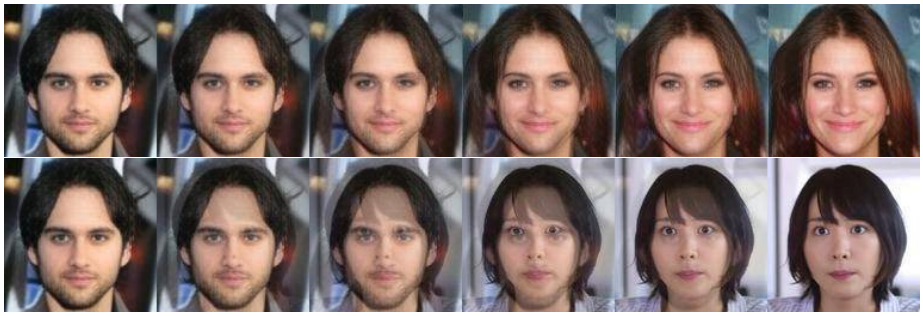

图 7: 线性插值

看起来还不错，可这是由于我们的初始、目标图片的五官位置几乎完全一致。一旦图片里人的五官有不对齐，就会变成这样：

中间过程里，生成的人脸甚至出现了四只模糊的眼睛！显然，这个方法是不够普适的，我们无法保证所有目标图像的重要信息都被存储在同一块区域里，它们可能出现在相似却不完全重合的位置。因此，我们此前介绍的卷积就该发挥作用了。

3 使用神经网络：CycleGAN

3.1 为什么使用 CycleGAN？

正如上文所述，传统的插值算法无法得到图片中具体内容的特征。为此，我们可以引入卷积网络。为了得到较为优秀的卷积核的值，可以使用神经网络进行训练。

既然我们的中间图片是否优秀依赖于图片具体内容的特征，假设我们已经知道了我们的起始图片和目标图片的属性（为了避免混淆，我们将卷积提取到的内容称为特征，将具有具体意义的，例如眼睛、头发颜色等，称为属性），如果属性足够丰富，那么我们只要将我们起始属性与目标属性不同的那部分改变为目标属性，就实现了图片的转换与部分中间过程。例如，将一个在笑的男人变为一个没有笑的女人，我们可以先生成一个在笑的女人，再将这张生成的照片进一步转换为没有笑的女人即可。

现在，我们的目标变为将具有某一属性（如“在笑”）的照片，转换为不具有该属性的照片，同时保证其余内容尽可能不改变。而这正是 CycleGAN 所擅长的。

3.2 CycleGAN 是什么？

CycleGAN 用于解决在两个域之间相互转换的问题，正如其名字所示，它的结构是一对对偶的 GAN。

假设我们想要将域 A 与域 B 上的内容相互转换（例如在上文中，域 A 是“在笑的人”，而域 B 是“不在笑的人”），那么将会有一个生成网络负责将域 A 上的内容变为域 B 上的内容，一个判别网络负责鉴定当前内容是否在域 B 上，抑或是生成网络制造的错误图片。对称地，还有一个从 B 至 A 的生成网络和判别是否是 A 上天然图片的判别网络。

为了让这两个 GAN 可以相互帮助对方进行修正，CycleGAN 引入了重构误差，也就是将域 A 上的内容通过生成网络映射至域 B 后，再通过另一个生成网络重新映射回 A，然后再利用对应的判别网络进行鉴别，并修改两个生成网络的参数以尽量迷惑鉴别网络。对于 B 上的内容，有一套对称的步骤。

3.3 一些额外的小技巧

神经网络的结构采用了 ResNet 模型。ResNet 模型最初被用于解决神经网络层数加深至一定数量，性能降低、计算速度不足的问题。在 ResNet 模型中，较深层的输入由两部分构成，其一为上一层的输出信息（被称为残差），其二为多层以前的输出信息（称为恒等项），如下图所示：

实际上，这相当于将学习的目标从被提取的特征本身，变为了被提取特征与输入值的差，正因此，输出信息才会被称为“残差”。在进行反向梯度传播时，这一网络只有在极少情况下才会造成残差项与恒等项的梯度相同（恒等项梯度仅由激活函数决定，与一个常数项类似），反向传播梯度消失的情况。因此理论上，如果网络已经到达最优，继续加深网络，残差项的影响将会变为 0，仅剩恒等项。即深度过高时，网络的结构相当于保持不变，仍然处于最优状态。

除了 ResNet 以外，还有一个常用模型，这就是 U-Net 模型。该模型结构如下所示：

U-net 模型中的网络是对称的结构，将第 i 层的输出加入到倒数第 i 层的输入中。这一模型融合了不同尺度的特征，同时能保证上采样恢复的特征不会过于粗糙。

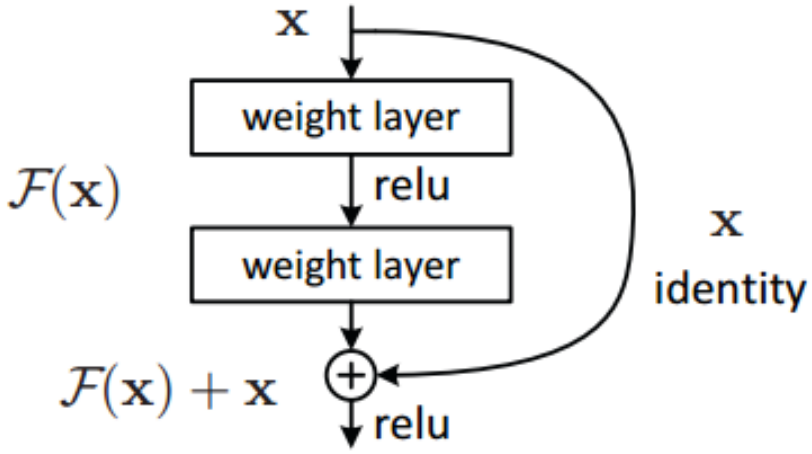

图 8: ResNet 模型。$\mathcal{F}$(x) 是残差项，而 x 是恒等项

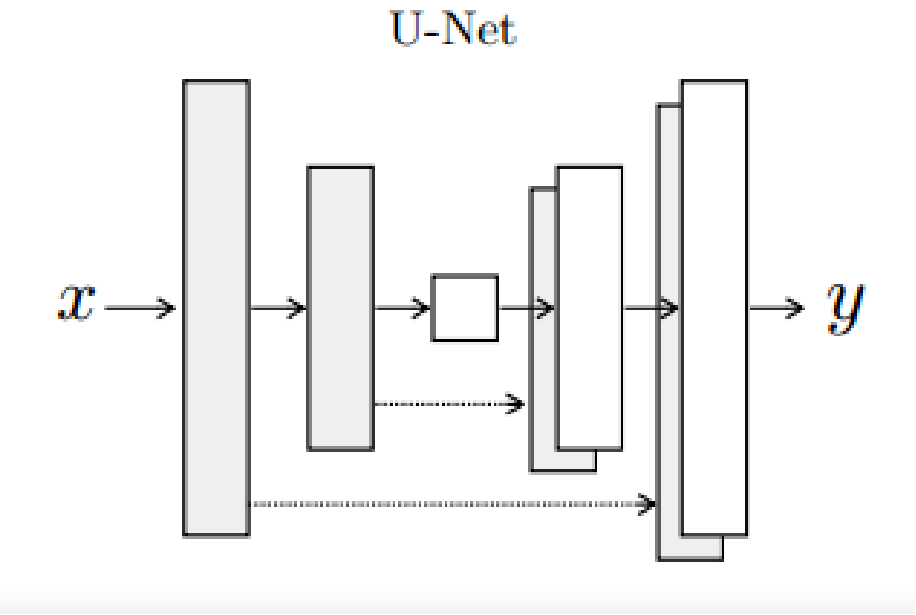

图 9: U-net 模型，前半部分被称为下采样，后半部分被称为上采样

3.4 结果如何？

3.5 CycleGAN 的问题

首先，这样的转换依赖于图片中的内容的属性，因此需要手动对图片的属性进行标注。当然，我们也可以手动训练一个网络来对图片中的属性进行标注。

此外，正如结果所示，在图片属性不够丰富时，我们转换的最终结果与我们期望的结果并不相似。然而，当图片属性过多，训练这些网络的开销又会变得非常庞大，且图片的转换过程变得非常漫长。

最后一个问题是，我们其实并不能获得所有有意义的中间状态。例如，当我们将一个在大笑的男人转换为不在笑的女人时，在“在大笑的女人 → 不在笑的女人”这一过程里，我们不能获得位于其间的“在微笑的女人”。

4 新的思路：引入一个同态

4.1 为什么需要同态？

将生成网络细分为三个部分：提取特征、中间过程和翻译特征。只要我们能在提取特征后数据所在的隐空间里找到一个有效、连续的插值，上面的 CycleGAN 的问题就得到了解决。

在实践中我们发现，如果在隐空间上作均匀的线性变化，对应图像上特定属性的变化并不均匀。相反，它往往在某一段区间上突变。如下图所示，我们希望将人脸的笑容隐去，并控制其间笑容逐渐淡化的程度（如第二行所示），但此前的方法并不能完成我们的目标（如第一行所示），图片在 c 与 d 之间有较大的变化，却在其余部分几乎不变。

因此,在隐空间上进行插值时,线性函数效果不佳,需要一个非线性的函数。同时,CycleGAN 中低效的问题仍然无法解决。

综上所述，我们的新算法应当让卷积得到的特征尽可能表示图像中的所有属性，并且使用神经网络来拟合一个个插值。尽管隐空间上的插值无法使用线性插值，但现实属性的变换与过渡，是可以使用线性插值来完成的。如果我们能让现实属性的过渡与隐空间上的插值形成同态的关系，就可以通过现实属性的过渡来纠正隐空间上神经网络拟合插值函数的误差。

图 10

4.2 如何实现同态？

在神经网络中，生成器被分为三个部分：提取特征、插值、翻译特征。其中提取特征与翻译特征的过程，相当于实现图片与隐空间之间的相互映射。插值部分包括了多个并列的神经网络，每一个网络对应了一个有现实意义的属性。插值的最终结果等于每个神经网络结果的加和。由于插值结果与现实属性是同态的，而现实中属性的过渡可以用线性的变换，故可以直接对插值结果均匀变换，来调整某个属性的变化程度。

为了校正用于插值的网络，需要额外引入一个网络 $\mathcal{A}'(\cdot)$ ，其作用是拟合得到一个函数，可以将插值结果映射至现实属性，从而得到对应的误差。

总的来说，我们的模型的结构如下：

图 11

训练时，读入两组图片 $\mathrm{I}m_a$、$\mathrm{I}m_b$ 和它们自身的一些属性 z_a、z_b。图片首先通过提取特征的网络 $\mathcal{E}$，得到对应的特征矩阵 (将起始图片和目标图片的特征分别记为 f_a 和 f_b)。

接下来我们将两个特征矩阵通过插值器 $\mathcal{I}$ 进行插值，得到 $\mathrm{F}_{\rangle nterp}$。插值器是一组结构相同的神经网络，插值的方法如下：

$$\mathrm{F}_{\rangle nterp} = \mathrm{F}_a + \sum_{i=1}^{c} \sqsubseteq_i \cdot \mathcal{T}_i(\mathrm{F}_b - \mathrm{F}_a)$$

其中，$\mathcal{T}_i$ 代表第 i 个属性的插值函数，是一个简单的神经网络，$\sqsubseteq_i$ 代表了第 i 个属性的

变化程度，在 0 和 1 之间，越靠近 0，表示这一属性的变化程度越小。这一插值的原理还会在下文中插值部分进一步解释。

在插值后，我们将得到的特征 $\mathrm{F}_{\rangle nterp}$ 通过另一个神经网络 $\mathcal{D}$ 来实现翻译特征的过程，以得到最终的图片。

为了对每一个神经网络的参数进行更新，还额外引入了一些训练中的过程：

首先，生成的内容需要通过一个神经网络 $\mathcal{D}is$ 进行鉴别，我们采用了 WGAN，但鉴别的对象并不是生成的图片，而是插值得到的特征 $\hat{\mathrm{F}}$。（这一特征会与图片中直接提取出的特征 F 进行比较）为了让训练更稳定，还使用了梯度惩罚。

$$\min_{\mathcal{D}is}\mathcal{L}_{\mathrm{G}AN_{\mathcal{D}is}}=\mathbb{E}_{\mathbb{P}_{\rangle nterp}}[\mathcal{D}is(\hat{\mathrm{F}})]-\mathbb{E}_{\mathbb{P}_{\nabla eal}}[\mathcal{D}is(\mathrm{F})]+\lambda_{\}p}\mathcal{L}_{\}p}(4.1)$$

$$\min_{\mathcal{E},I}\mathcal{L}_{\mathrm{G}AN_{\mathcal{E},I}}=\mathbb{E}_{\mathbb{P}_{\nabla eal}}[\mathcal{D}is(\mathrm{F})]-\mathbb{E}_{\mathbb{P}_{\rangle nterp}}[\mathcal{D}is(\hat{\mathrm{F}})](4.2)$$

神经网络 $\mathcal{D}is$ 的参数也通过这一距离来更新，体现了网络中对抗的成分。

对于生成器中提取和翻译特征的部分 $\mathcal{E}$、$\mathcal{D}$，额外增加一个步骤来更新其参数：将提取出的特征不通过插值器，直接进行翻译。得到的图片与原图的差距即可被用于更新这两个网络的参数。实践表明，计算距离时直接将两张图片各像素点的 RGB 值相减的效果较差，因此，我们引入一个已训练好的、在判别图片差距这一模型上表现优秀的网络 VGG 来计算两张图片的差距：

$$\min_{\mathcal{D}}\mathcal{L}_{\mathcal{D}}=\mathbb{E}(||\Phi_3(\mathcal{D}(\mathrm{F}))-\Phi_3(\mathcal{I}m)||^2)(4.3)$$

$$\min_{\mathcal{E}}\mathcal{L}_{\nabla econ}=\mathbb{E}(||\Phi_3(\mathcal{D}(\mathrm{F}))-\Phi_3(\mathcal{I}m)||^2)(4.4)$$

VGG 网络给出图片之间差距的方法实质上是通过卷积提取各个局部的特征，从而对图片的每个小块的距离进行比较。这一原理与提取特征的 $\mathcal{E}$ 的目标很相似。因此，我们可以用 VGG 中更深层的部分来更新 $\mathcal{E}$ 的参数：

$$\min_{\mathcal{E},I}\mathcal{L}_{\mathrm{K}G}=\mathbb{E}_{\mathbb{P}_{\nabla eal}}||\mathcal{P}[\mathcal{E}(\mathrm{I}m)]-\Phi_5(\mathrm{I}m)||^2(4.5)$$

其中，网络 $\mathcal{P}$ 只含有一个卷积核大小为 $1*1$ 的卷积层，目的仅在于让两个被比较对象的形状相同。

现在，除了插值以外的两个部分的参数可以得到充分的更新了。$\mathcal{D}$ 的更新如公式 4.3 所示，$\mathcal{E}$ 的更新通过以下公式：

$$\mathcal{L}_{\mathcal{E}} = \lambda_{\mathrm{GAN}_{\mathcal{E}}} \mathcal{L}_{\mathrm{GAN}_{\mathcal{E},I}} + \lambda_{\nabla econ} \mathcal{L}_{\nabla econ} + \lambda_{\mathrm{KG}} L_{\mathrm{KG}} (4.6)$$

过程中引入的网络 $\mathcal{D}is$、$\mathcal{P}$ 的更新分别通过公式 4.1 与 4.5 所示。

现在我们来看插值的部分。正如上文所述，我们引入了一个网络 $\mathcal{A}'(\cdot)$，模拟从插值得到的矩阵 $\mathcal{T}(\mathrm{F}_b - \mathrm{F}_a)$ 到属性 z（与图片一同读入，随插值改变）的同态。注意 $\mathcal{T}$ 实际上是一组网络 $\mathcal{T}_1 ... T_c$。对两张图片的属性，可以直接使用线性插值，即

$$\mathrm{z}_{\rangle nterp} = \mathrm{z}_a + \sum_{i=1}^{c} \sqsubseteq_i \cdot (\mathrm{z}_b - \mathrm{z}_a)$$

上文中，两个特征的差通过插值器后得到的矩阵可以直接与代表插值程度的系数相乘，正是由于插值器得到矩阵与属性的同态，和属性可以进行上述线性插值。通过这一同态，我们可以更新插值网络的参数了：

$$\min_{\mathcal{I}} \mathcal{L}_{\mathrm{F}_{hom}} = \mathbb{E}[-\mathrm{z}_{\rangle nterp} \log(\mathcal{A}'(\mathrm{F}_{\rangle nterp}))] (4.7)$$

网络 $\mathcal{A}'(\cdot)$ 的误差（被称为同态差距）也可以通过类似的方式更新：

$$\min_{\mathcal{A}'(\cdot)} \mathcal{L}_{\mathcal{A}'(\cdot)} = \mathbb{E}[-\mathrm{z}_a \log(\mathcal{A}'(\mathrm{F}_{\rangle nterp}))] (4.8)$$

尽管将 z_a 与 $\mathcal{A}'(\mathrm{F}_a)$ 比较更符合逻辑，但实际实践中，这会让插值器效果变差，因此其意义被改为让插值器尽量减少变化的内容，从而让修改特定属性的插值器不改变其余内容。

此外，还可以将插值强度设为最高，即令插值器将 F_a 彻底转换为 F_b，然后用插值结果与 F_b 比较，也可以用于更新插值器:

$$\mathcal{L}_{\mathcal{I}_{\sqcup otal}} = ||\mathrm{F}_{\rangle nterp, v=1} - \mathrm{F}_b||^2$$

综上所述，插值过程的参数更新为：

$$\mathcal{L}_{\mathcal{I}} = \lambda_{GAN_{\mathcal{I}}}\mathcal{L}_{GAN_{\mathcal{E},I}} + \lambda_{\mathcal{I}_{\langle om}}\mathcal{L}_{\mathrm{F}_{\langle om}} + \lambda_{\mathcal{I}_{\sqcup otal}}\mathcal{L}_{\mathcal{I}_{\sqcup otal}} \quad (4.9)$$

过程中引入的网络 $\mathcal{A}'(\cdot)$ 的更新方法如公式 4.8 所示。

4.3 其它技巧

注意到训练的主要目标是获得一个较好的插值函数，因此我们的判别器被用于判定插值的结果，而非翻译特征后输出的图片。而 $\mathcal{A}'(\cdot)$ 同样是用于计算插值后结果与目标属性之间误差的交叉熵。因此，这两个网络具有一定的相似性。为了加快模型的训练速度，我们令 $\mathcal{A}'(\cdot)$ 与 $\mathcal{D}is$ 的前几层共用，只在最后一层有所区别。

4.4 最终成果

4.5 问题与改进

在训练过程中，我们向测试集中加入了一些额外的图片，发现在训练相同时间时的效果远弱于数据集中自带的测试集。我们认为，这是由于数据集中的头像位置几乎重合，因此我们加入的图片即使稍有位置的差异，也会造成效果不佳的问题。于是我们在训练过程中，手动为图像增加了位置的变化等干扰。

此外，人脸及其表情本身具有较好的连续性，但很多数据集并没有这一优势。因此，当我们将这一模型应用于一些图像变化连续性弱的领域时，得到的结果并不令人满意，这是这一网络没有考虑到的问题。

最终训练得到的模型中，输入图片背景为白色时，输出图片背景会变灰。我们认为这可能是由于网络的层次过深，导致提取与翻译特征中每个点都与整张图片中大部分都相关，从而让原本为白色的部分也受到了其余位置的颜色的影响。

5 总结