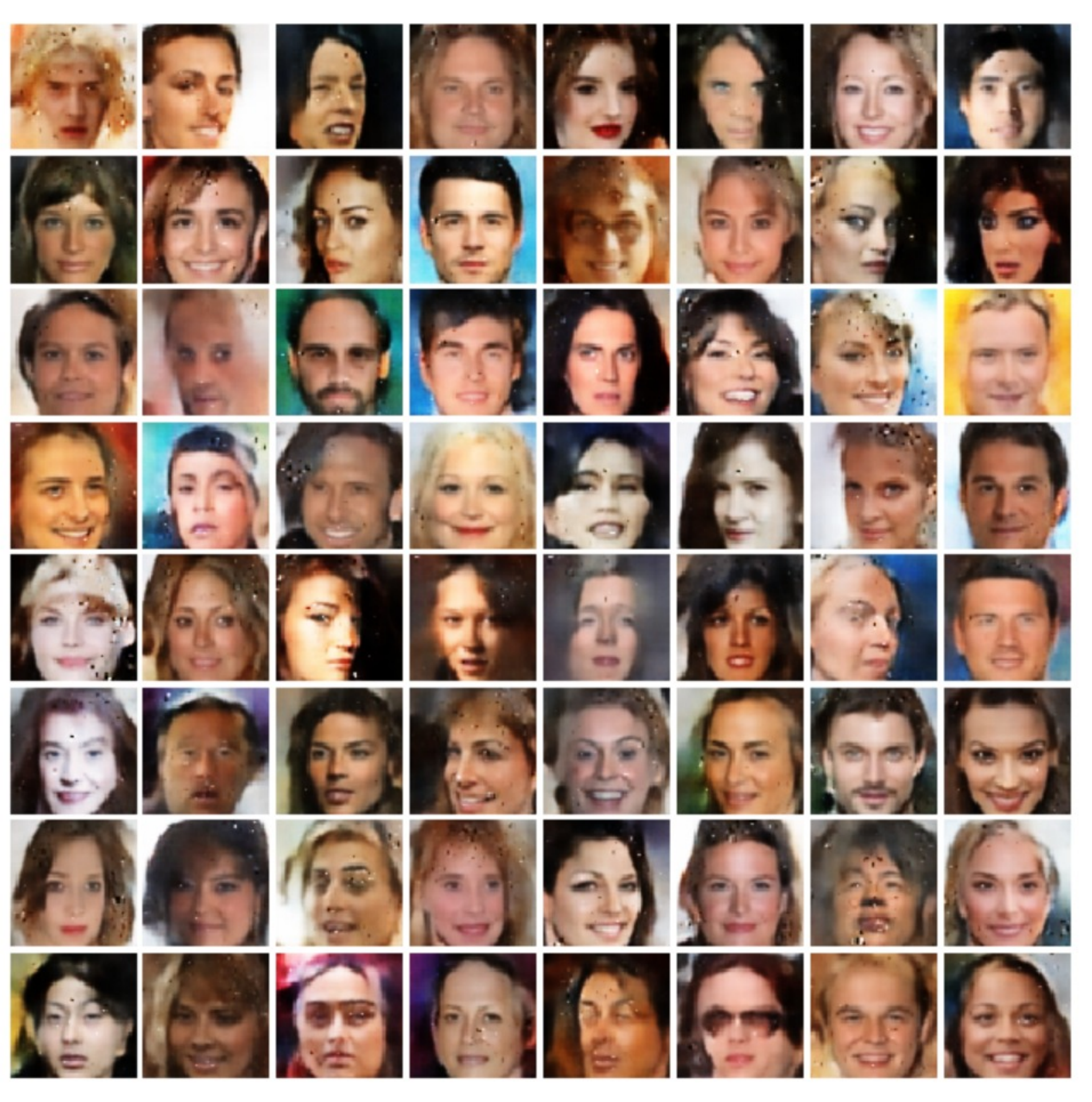
Progressive Growing GANs简介+PyTorch复现

Gapeng

2017-11-15 20:07:01

今天要介绍的文章是NVIDIA投稿ICLR 2018的一篇文章，Progressive Growing of GANs for Improved Quality, Stability, and Variation[1]，姑且称它为PG-GAN。从行文可以看出文章是临时赶出来的，毕竟这么大的实验，用P100都要跑20天，更不用说调参时间了，不过人家在NVIDIA，不缺卡。作者放出了基于lasagna的代码，今天我也会简单解读一下代码。另外，我也在用Pytorch做复现。

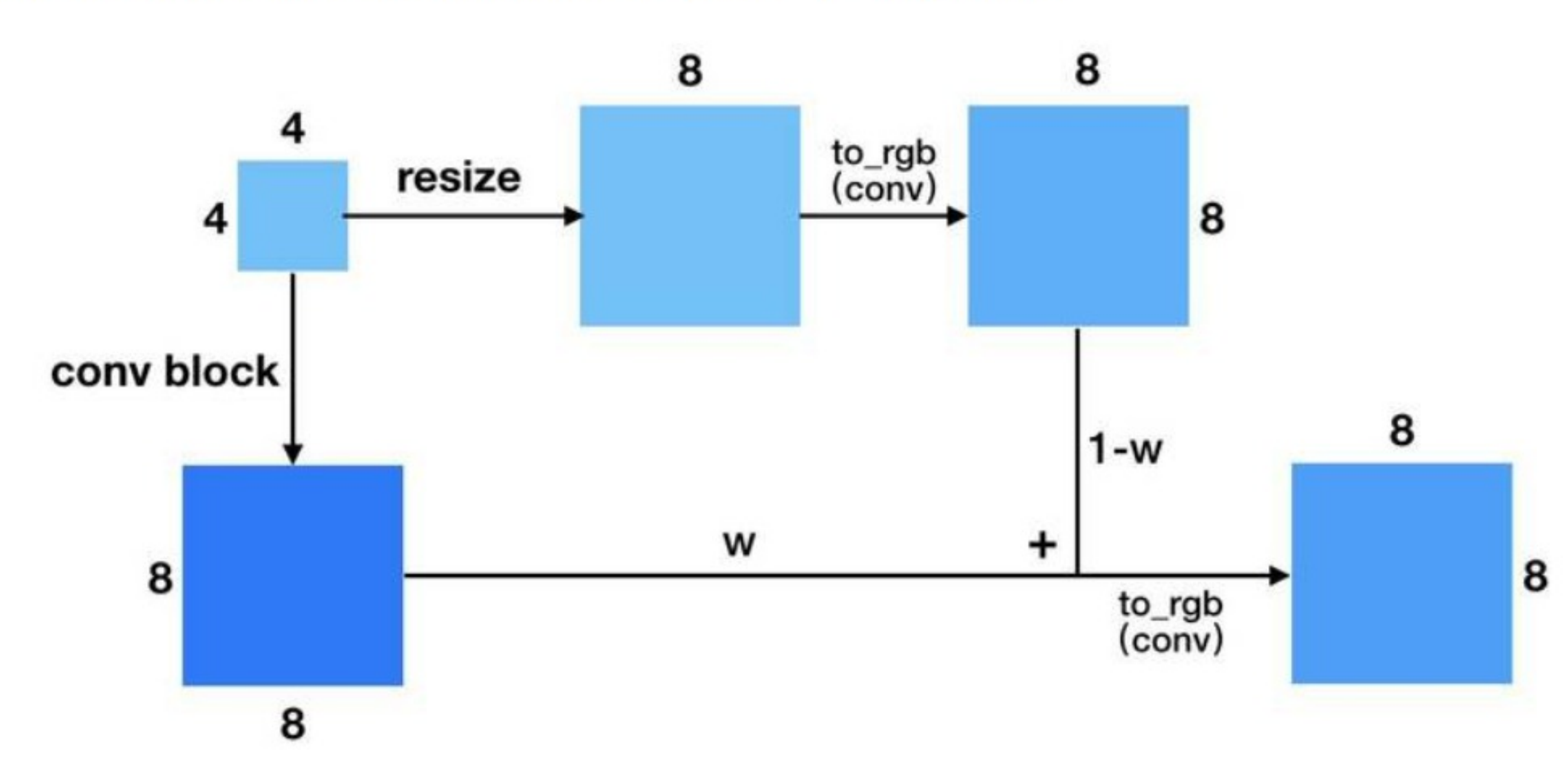
在PG-GAN出来以前，训练高分辨率图像生成的GAN方法主要就是LAPGAN[2]和BEGAN[6]。后者主要是针对人脸的，生成的人脸逼真而不会是鬼脸，这里也提一下，生成鬼脸的原因是Discriminator不再更新，它不能再给予Generator其他指导，Generator找到了一种骗过Discriminator的方法，也就是生成鬼脸，而且很大可能会mode collapse。下图是我用PyTorch做的BEGAN复现，当时没有跑很高的分辨率，但是效果确实比其他GAN好几本没有鬼脸。



PG-GAN能够稳定地训练生成高分辨率的GAN。我们来看一下PG-GAN跟别的GAN不同在哪里。

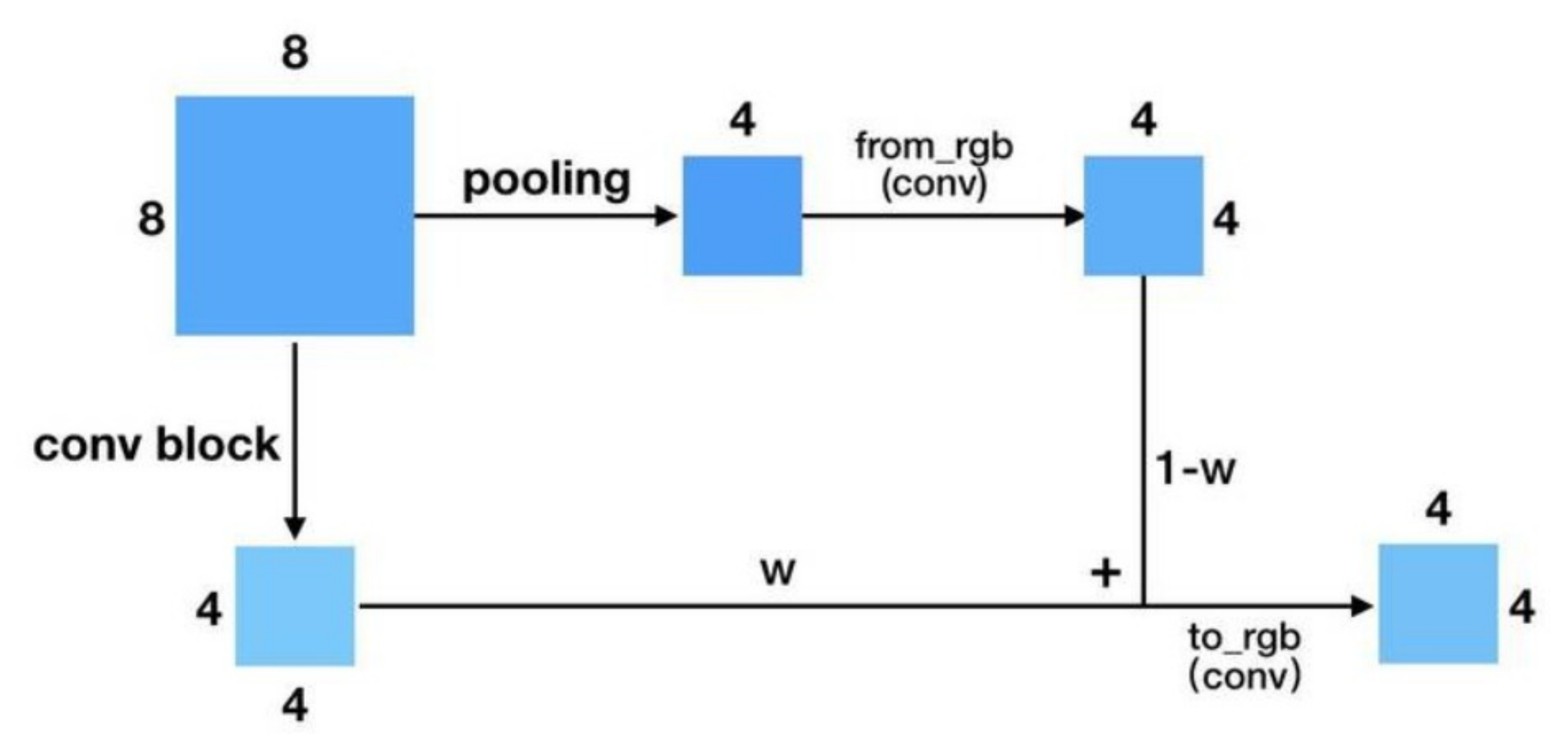
1. 训练方式¶

作者采用progressive growing的训练方式，先训一个小分辨率的图像生成，训好了之后再逐步过渡到更高分辨率的图像。然后稳定训练当前分辨率，再逐步过渡到下一个更高的分辨率。



如上图所示。更具体点来说，当处于fade in（或者说progressive growing）阶段的时候，上一分辨率(4x4)会通过resize+conv操作得到跟下一分辨率(8x8)同样大小的输出，然后两部分做加权，再通过to_rgb操作得到最终的输出。这样做的一个好处是它可以充分利用上个分辨率训练的结果，通过缓慢的过渡(w逐渐增大)，使得训练生成下一分辨率的网络更加稳定。

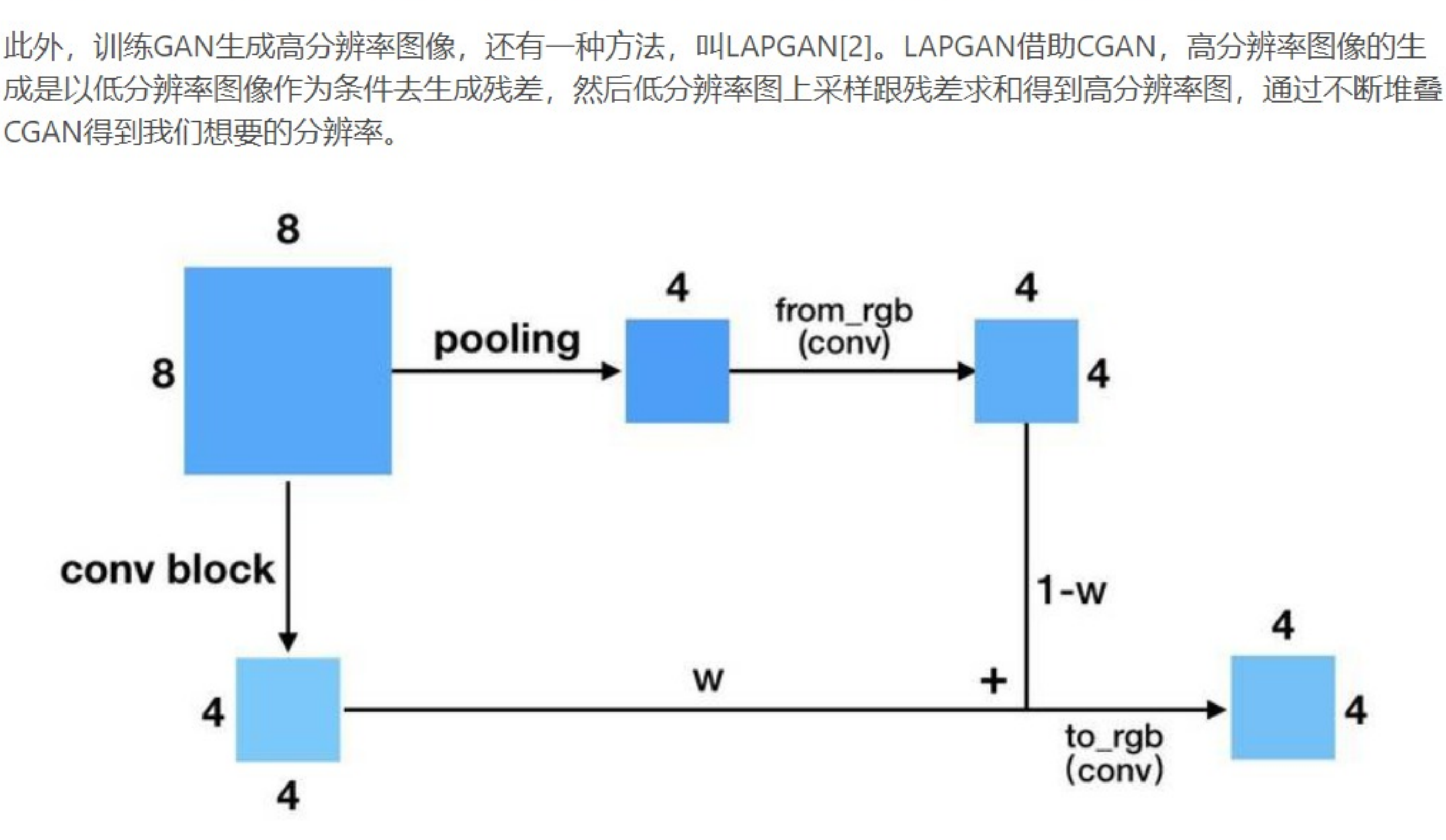
上面展示的是Generator的growing阶段。下图是Discriminator的growing，它跟Generator的类似，差别在于一个是上采样，一个是下采样。这里就不再赘述。



不难想象，网络在growing的时候，如果不引入progressive(fade in)，那么有可能因为比较差的初始化，导致原来训练的进度功亏一篑，模型不得不从新开始学习，如此一来就没有充分利用以前学习的成果，甚至还可能误导。我们知道GAN的训练不稳定，这样的突变有时候是致命的。所以fade in对训练的稳定性来说至关重要。

说到growing的训练方式，我们很容易想到autoencoder也有一种类似的训练方式：先训各一层的encoder和decoder，训好了以后再过渡到训练各两层的encoder和decoder，这样的好处是避免梯度消失，导致离loss太远的层更新不够充分。PG-GAN的做法可以说是这种autoencoder训练方式在GAN训练上的应用。

此外，训练GAN生成高分辨率图像，还有一种方法，叫LAPGAN[2]。LAPGAN借助CGAN，高分辨率图像的生成是以低分辨率图像作为条件去生成残差，然后低分辨率图上采样跟残差求和得到高分辨率图，通过不断堆叠CGAN得到我们想要的分辨率。



LAPGAN是多个CGAN堆叠一起训练，当然可以拆分成分阶段训练，但是它们本质上是不同的，LAPGAN学的是残差，而PG-GAN存在stabilize训练阶段，学的不是残差，而直接是图像。

作者在代码中设计了一个LODSelectLayer来实现progressive growing。对于Generator，每一层插入一个LODSelectLayer，它实际上就是一个输出分支，实现在特定层的输出。从代码来看，作者应该是这样训练的(参见这里的train_gan函数)，先构建4x4分辨率的网络，训练，然后把网络存出去。再构建8x8分辨率的网络，导入原来4x4的参数，然后训fade in，再训stabilize，再存出去。我在复现的时候，根据文章的意思，修改了LODSelectLayer层，因为pytorch是动态图，能够很方便地写if-else逻辑语句。

借助这种growing的方式，PG-GAN的效果超级好。另外，我认为这种progressive growing的方法比较适合GAN的训练，GAN训练不稳定可以通过growing的方式可以缓解。不只是在噪声生成图像的任务中可以这么做，在其他用到GAN的任务中都可以引入这种训练方式。我打算将progressive growing引入到CycleGAN中，希望能够得到更好的结果。

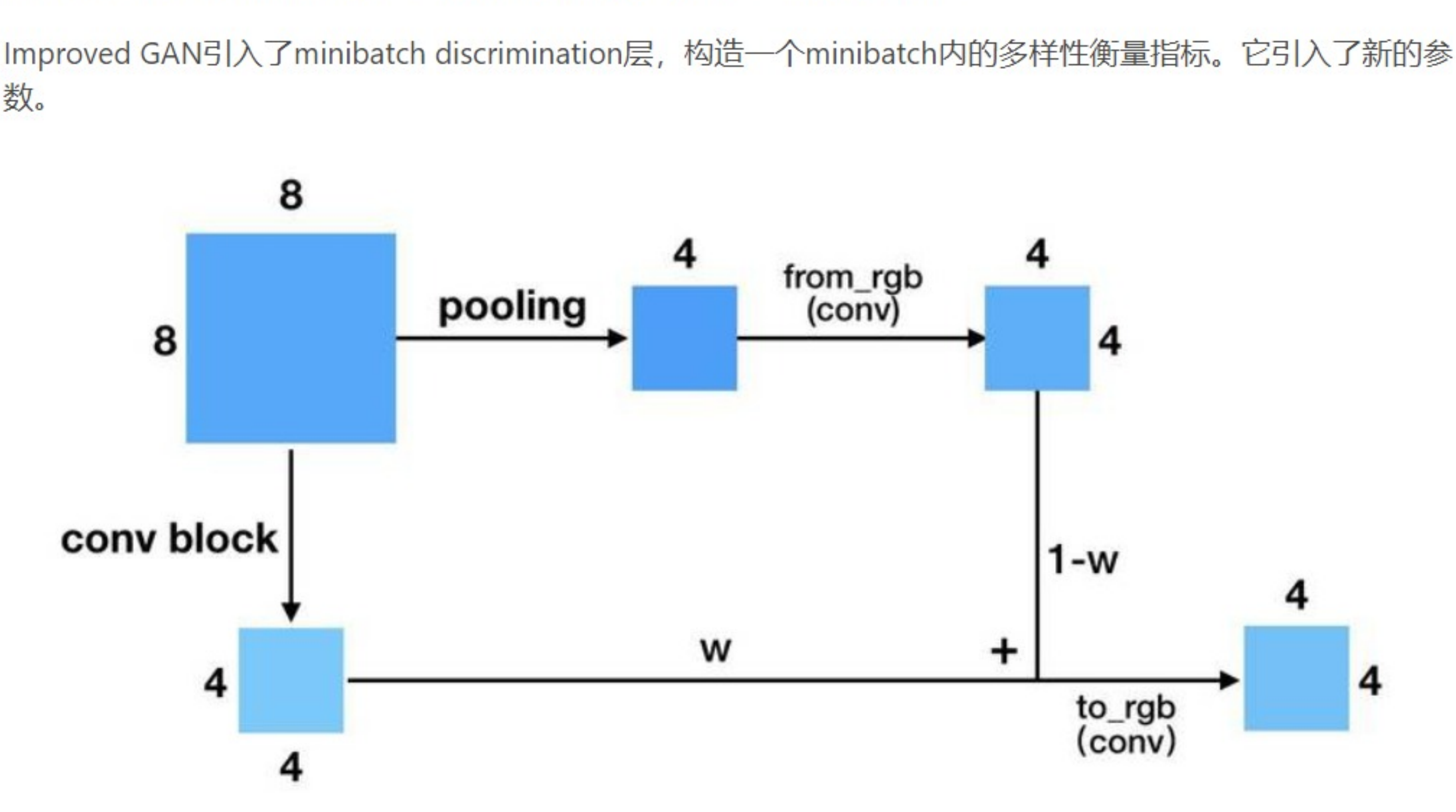
2. 增加生成多样性¶

增加生成样本的多样性有两种可行的方法：通过loss让网络自己调整、通过设计判别多样性的特征人为引导。

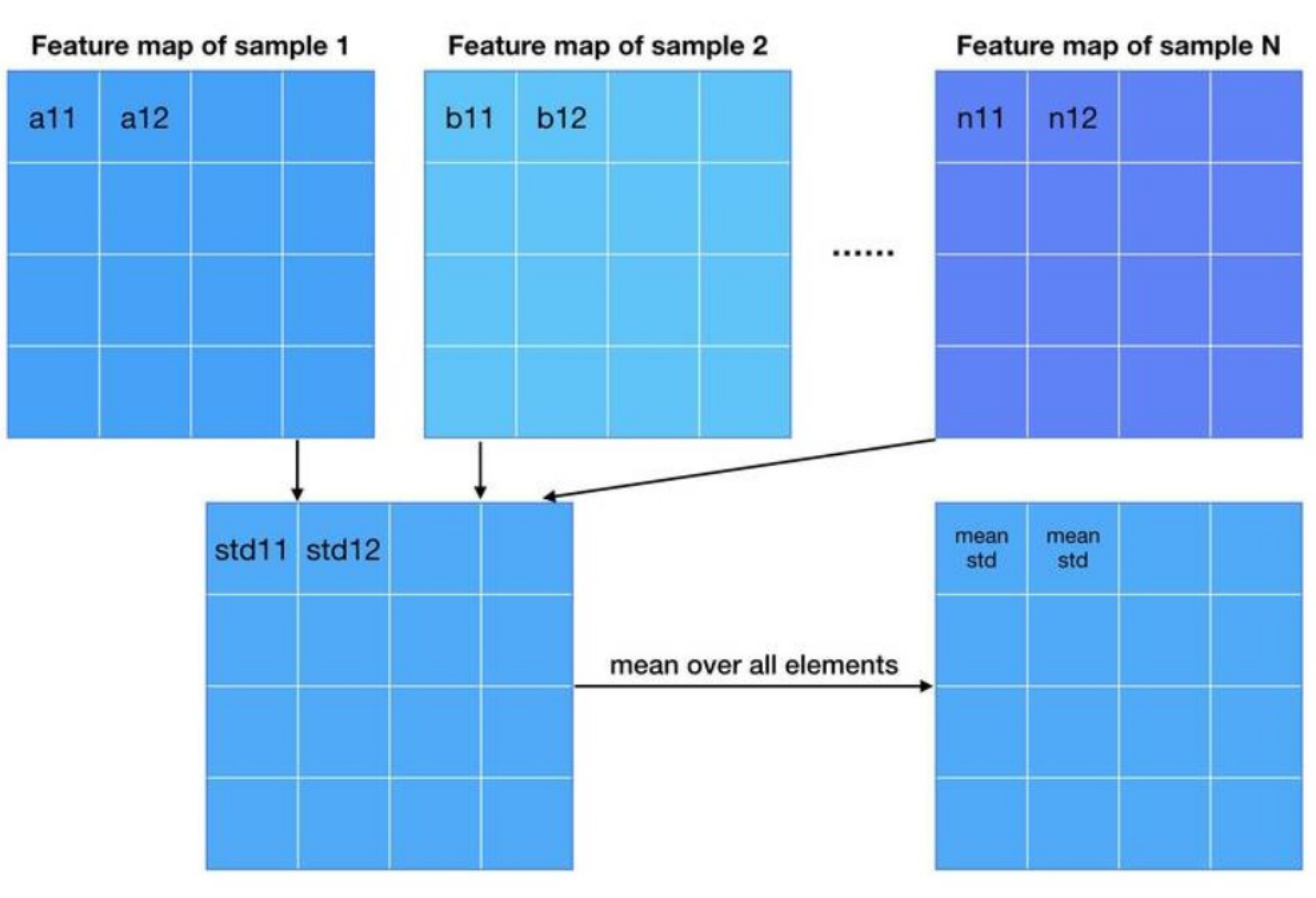
WGAN属于前者，它采用更好的分布距离的估计(Wasserstein distance)。模型收敛意味着生成的分布和真实分布一致，能够有多样性的保证。PG-GAN则属于后者。

作者沿用improved GAN的思路，通过人为地给Discriminator构造判别多样性的特征来引导Generator生成更多样的样本。Discriminator能探测到mode collapse是否产生了，一旦产生，Generator的loss就会增大，通过优化Generator就会往远离mode collapse的方向走，而不是一头栽进坑里。

Improved GAN引入了minibatch discrimination层，构造一个minibatch内的多样性衡量指标。它引入了新的参数。



而PG-GAN不引入新的参数，利用特征的标准差作为衡量标准。



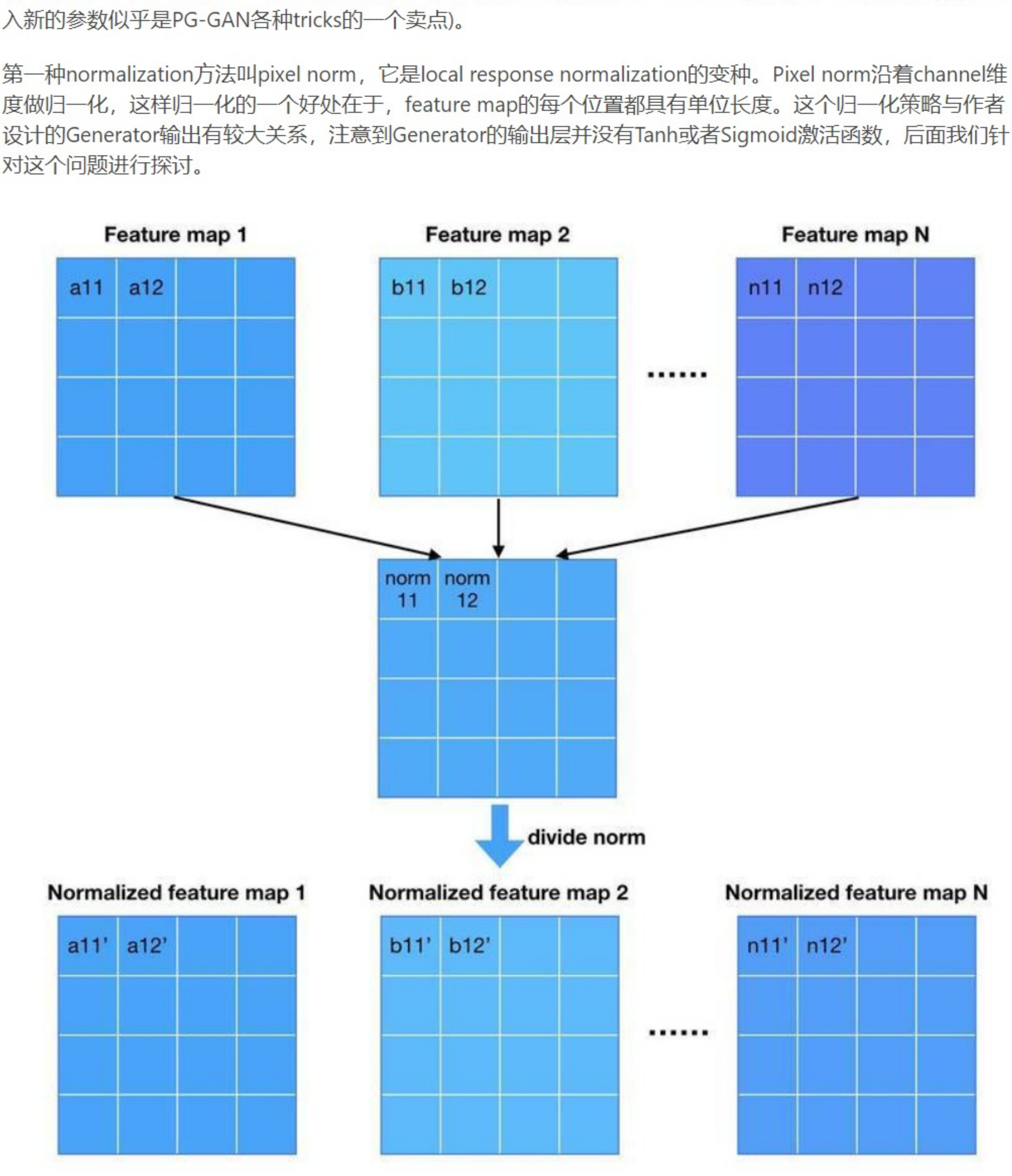
这里啰嗦地说明上面那张图做了什么。我们有N个样本的feature maps(为了画图方便，不妨假设每个样本只有一个feature map)，我们对每个空间位置求标准差，用numpy的std函数来说就是沿着样本的维度求std。这样就得到一张新的feature map(如果样本的feature map不止一个，那么这样构造得到的feature map数量应该是一致的)，接着feature map求平均得到一个数。这个过程简单来说就是求mean std，作者把这个数复制成一张feature map的大小，跟原来的feature map拼在一起送给Discriminator。

从作者放出来的代码来看，这对应averaging=“all”的情况。作者还尝试了其他的统计量：“spatial”，“gpool”，“flat”等。它们的主要差别在于沿着哪些维度求标准差。至于它们的作用，等我的代码复现完成了会做一个测试。估计作者调参发现“all”的效果最好。

3. Normalization¶

从DCGAN[3]开始，GAN的网络使用batch(or instance) normalization几乎成为惯例。使用batch norm可以增加训练的稳定性，大大减少了中途崩掉的情况。作者采用了两种新的normalization方法，不引入新的参数(不引入新的参数似乎是PG-GAN各种tricks的一个卖点)。

第一种normalization方法叫pixel norm，它是local response normalization的变种。Pixel norm沿着channel维度做归一化，这样归一化的一个好处在于，feature map的每个位置都具有单位长度。这个归一化策略与作者设计的Generator输出有较大关系，注意到Generator的输出层并没有Tanh或者Sigmoid激活函数，后面我们针对这个问题进行探讨。



第二种normalization方法跟凯明大神的初始化方法[4]挂钩。He的初始化方法能够确保网络初始化的时候，随机初始化的参数不会大幅度地改变输入信号的强度。

$$\frac{1}{2}n_l\mathrm{Var}[w_l]=1\ (w_l:\text{weights of layer } l;\ n_l:\text{number of weights at layer } l)$$

根据这个式子，我们可以推导出网络每一层的参数应该怎样初始化。可以参考pytorch提供的接口。

作者走得比这个要远一点，他不只是初始化的时候对参数做了调整，而是动态调整。初始化采用标准高斯分布，但是每次迭代都会对weights按照上面的式子做归一化。作者argue这样的归一化的好处在于它不用再担心参数的scale问题，起到均衡学习率的作用(euqalized learning rate)。

4. 有针对性地给样本加噪声¶

通过给真实样本加噪声能够起到均衡Generator和Discriminator的作用，起到缓解mode collapse的作用，这一点在WGAN的前传中就已经提到[5]。尽管使用LSGAN会比原始的GAN更容易训练，然而它在Discriminator的输出接近1的适合，梯度就消失，不能给Generator起到引导作用。针对D趋近1的这种特性，作者提出了下面这种添加噪声的方式

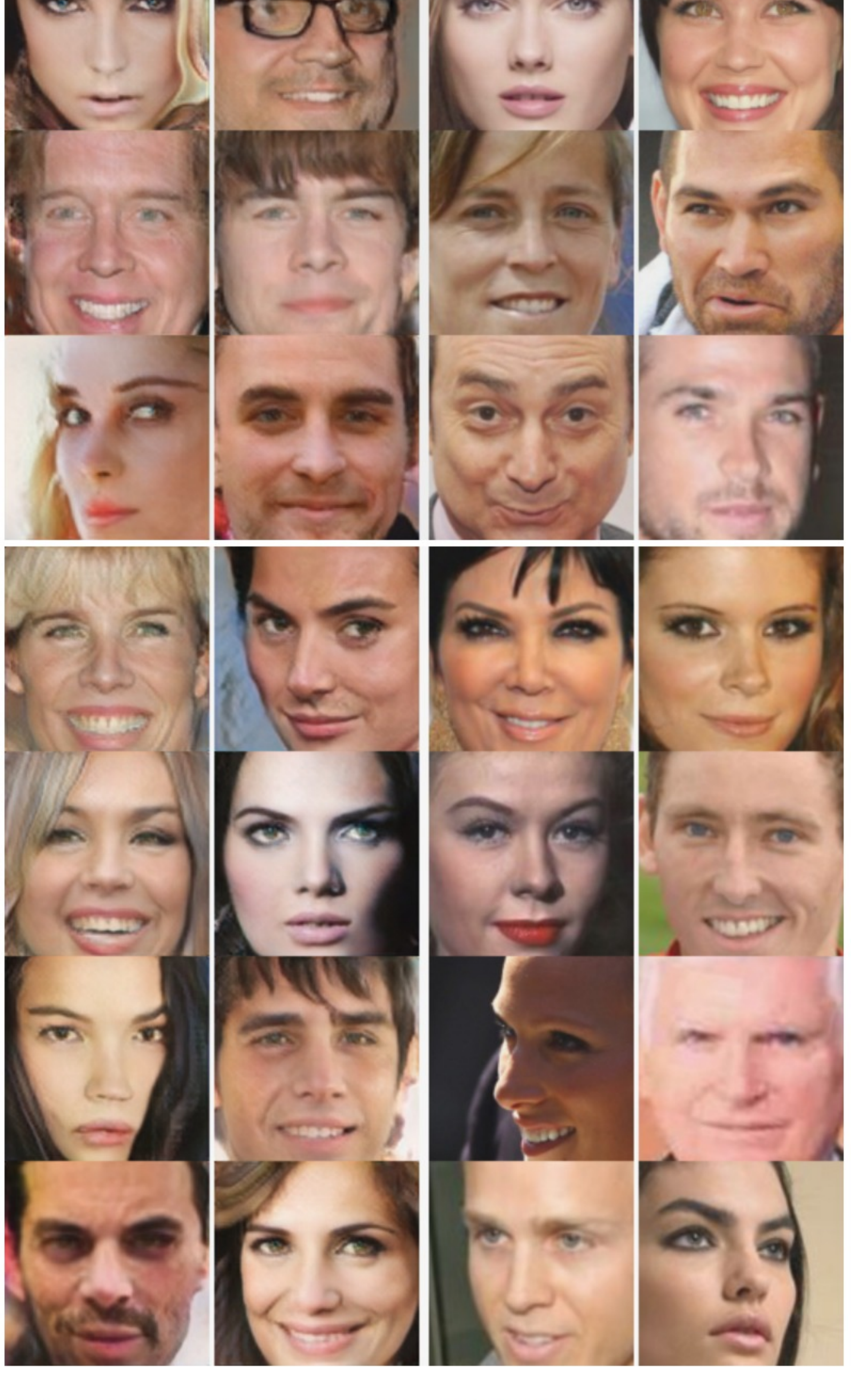
$$\text{noise strength}=0.2*\max(0,d_t-0.5)^2$$

其中，$d_t=0.9*d_{t-1}+0.1*d$，d_t, d分别为第t次迭代判别器输出的修正值、第t-1次迭代真样本的判别器输出。

从式子可以看出，当真样本的判别器输出越接近1的时候，噪声强度就越大，而输出太小(<=0.5)的时候，不引入噪声，这是因为0.5是LSGAN收敛时，D的合理输出(无法判断真假样本)，而小于0.5意味着D的能力太弱。

文章还有其他很多tricks，有些tricks不是作者提出的，如Layer norm，还有一些比较细微的tricks，比如每个分辨率训练好做sample的时候学习率怎么decay，每个分辨率的训练迭代多少次等等，我们就不再详细展开。具体可以参见官方代码，也可以看我复现的代码。

目前复现的结果还在跑，现在训练到了128x128分辨率的fade in阶段，放一张当前的结果图，左边2列是生成的图，右边2列是真实样本。256x256分辨率的结果很快就会出来了。



官方Lasagna代码：tkarras/progressive_growing_of_gans

我的PyTorch复现：github-pengge/PyTorch-progressive_growing_of_gans