第 5 次实验报告

2022111663　刘博飞

2024 年 11 月 24 日

1　实验环境介绍

1. 操作系统

   版本 Windows 11 专业版

   版本号 23H2

   安装日期 2024/10/10

   操作系统版本 22631.4317

   体验 Windows Feature Experience Pack 1000.22700.1041.0

2. GPU 型号

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 560.94                 Driver Version: 560.94         CUDA Version: 12.6     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 3090      WDDM  |   00000000:01:00.0 Off |                  N/A |
| 32%   28C    P8             12W /  350W |       0MiB /  24576MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

图 1: GPU 型号

3. torch 版本

   torch 2.4.1+cu121 pypi_0 pypi

   torchaudio 2.4.1+cu121 pypi_0 pypi

   torchvision 0.19.1 pypi_0 pypi

2 stylegan 原理说明

2.1 生成对抗网络（GAN）概述

StyleGAN 属于 GAN 的一种扩展，GAN 由两部分组成：

- **生成器（Generator）**：负责从噪声中生成假图像。
- **判别器（Discriminator）**：负责判断图像是真实图像还是生成图像。

生成器和判别器通过对抗训练进行优化，生成器的目标是生成足够逼真的图像，以使判别器无法区分真假，而判别器的目标则是尽可能准确地辨别真假图像。

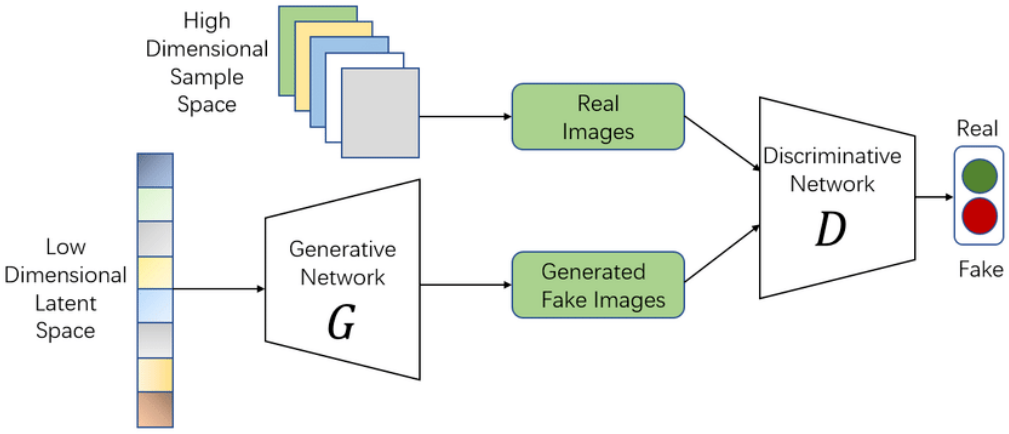


图 2: gan 网络结构

2.2 StyleGAN 的核心思想

StyleGAN 与传统的 GAN 的不同之处在于它引入了“风格化”机制，使得生成的图像可以在多种尺度上进行风格控制，同时保持较高的细节水平。

2.2.1 潜在空间的映射（Mapping Network）

在传统的 GAN 中，生成器通常通过一个固定大小的潜在向量（通常是一个高维的随机噪声向量）来生成图像。而 StyleGAN 在生成器前增加了一个映射网络（Mapping Network），这个网络的作用是将潜在向量 z 映射到一个新的潜在空间 w。

- **映射过程**：潜在向量 z 经过多层全连接层的映射（通常使用 ReLU 激活函数），产生一个新的向量 w。这个向量 w 将作为生成图像的输入。
- **潜在空间**：这个新的空间被称为“风格空间”，它是高维的，比原始潜在向量的维度高(相当于对原始图像信息的解耦)。通过这种方式，网络能够在生成图像时引入更复杂的风格控制。

2.2.2 风格化过程

风格化的核心在于将潜在空间向量 w 与生成器的不同层级的卷积层进行结合，允许不同的层在不同的空间尺度上控制图像的风格。

- **风格化应用**：在 StyleGAN 中，每一层的卷积运算都有一个对应的“风格”向量，这个向量来自于映射网络输出的 w 向量。每一层的卷积权重会通过这些风格向量进行“条件化”调整，使得图像的不同层级可以有不同的风格特征。
- **影响尺度**：风格向量的引入让不同层对生成图像的不同部分（如大体形状、细节纹理、颜色等）产生影响。高层的风格控制图像的宏观结构（如大致形状、布局等），低层的风格则控制细节（如纹理、边缘、颜色等）。

2.2.3 噪声注入（Noise Injection）

除了风格向量，StyleGAN 还通过在生成过程中注入噪声来增加生成图像的多样性。噪声向量被加到生成图像的不同层次，尤其是在低频区域，以增加细节并改善生成图像的细腻度。

2.3 StyleGAN 的网络结构

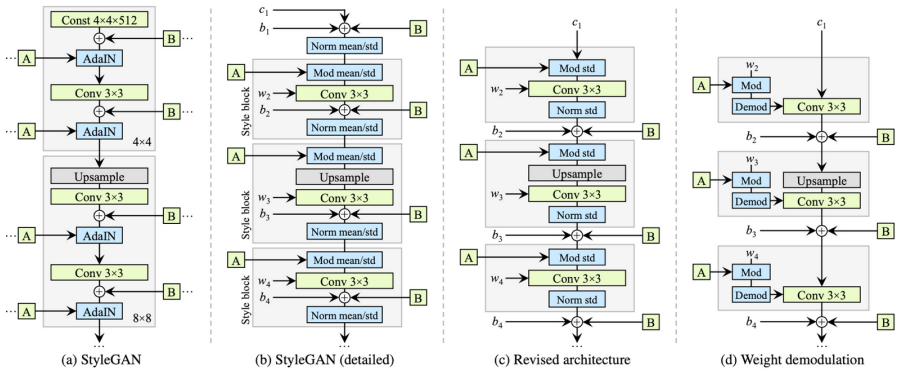


图 3: stylegan 网络架构

StyleGAN 的生成网络由多个关键组件构成，包括以下几个部分：

2.3.1 Mapping Network（映射网络）

映射网络将潜在向量 z 映射到风格空间 w，通常由多层全连接网络构成。每一层的输出都会被正则化并传递到下一层，最终输出一个新的向量 w，该向量控制生成器各层的风格化过程。

2.3.2 合成网络（Synthesis Network）

合成网络的作用是根据风格向量 w 和噪声向量 n 来生成图像。合成网络通常由多个卷积层组成，并在每一层引入风格向量进行权重控制。每个卷积层不仅接收来自前一层的输出，还会从 w 向量中获取风格信息。

生成过程从一个低分辨率的图像开始，逐渐通过多个反卷积层提高分辨率，直到最终生成高分辨率的图像。

2.3.3 判别器

判别器的结构类似于传统的卷积神经网络，包括多层卷积层和池化层。它的任务是判断输入的图像是来自真实分布还是来自生成器。在 StyleGAN 中，判别器不仅仅是一个简单的二分类网络，还包含一些细节提取机制，以提高判别能力。

2.3.4 自适应归一化（Adaptive Instance Normalization, AdaIN）

为了实现风格化的效果,StyleGAN 在生成过程中对每一层使用了自适应实例归一化（AdaIN）。AdaIN 的核心思想是，通过给每一层的卷积操作引入来自风格向量的控制信号，来动态调整生成图像的色彩、纹理等特征。这种方法让网络能够在不同尺度上控制图像风格的变化。

2.4 风格化如何生效

- **层级控制**：StyleGAN 通过在不同的层级控制图像的风格。较高层的风格影响图像的整体结构和大体布局，而较低层的风格则控制细节（例如，面部特征、头发的纹理等）。
- **风格向量的逐层应用**：映射网络将潜在空间向量 z 映射到一个风格空间向量 w，然后这些风格向量被逐层应用于生成器中的卷积层。通过这种方式，不同层级的风格向量会影响图像的不同特征，且这些特征是连续可控的。
- **控制细节与大局**：风格向量使得生成器能够在更细致的尺度上控制生成图像的外观。例如，通过调整风格向量，生成器能够生成不同风格的面部图像，改变面部的表情、肤色、头发样式等。

3 clip + stylegan 架构实现及训练过程

3.1 网络架构

本实验采用了 clip 模型的图像编码器 (image encoder) 对输入图像进行编码，进而得到了图像的嵌入向量 (embedding), 这个 embedding 的维度是 (,1024).

```
model, preprocess = load_pretrained_CLIP()
my_face = preprocess(Image.open('my_face.jpg')).unsqueeze(0).cuda()
features = model.encode_image(my_face).cuda()
features = features / features.norm(dim=-1, keepdim=True)  # 归一化
generated_latent = bridge(features.float())
generated_latent.cuda()
```

由于 stylegan 的输入是 512 维，因此需要训练一个线性全连接网络 (FeatureToLatent), 将 clip 编码的 embedding 投影到 stylegan 的输入 (latent), 这个线性全连接网络要求可以将 clip 编码的信息与 stylegan 的输入信息建立正确的联系，FeatureToLatent 实现如下：

```
class FeatureToLatent(nn.Module):
def __init__(self, input_dim, latent_dim):
    super(FeatureToLatent, self).__init__()
    self.fc1 = nn.Linear(input_dim, 1024)
    self.fc2 = nn.Linear(1024, 1024)
    self.fc3 = nn.Linear(1024, 1024)
    self.fc4 = nn.Linear(1024, latent_dim)

def forward(self, x):
    x = torch.relu(self.fc1(x))
    x = torch.relu(self.fc2(x))
    x = torch.relu(self.fc3(x))
    return self.fc4(x)
```

最后，由 stylegan 接收潜在向量并生成图像。

```
G = load_pretrained_GAN(model_path)
img_data = G(generated_latent, c)
```

3.2 训练过程细节

- **损失函数：**由于 FeatureToLatent 的训练目标是建立 clip 输出与 stylegan 输入的正确关系，因此将损失定义为输入图像与输出图像的相似度，具体来说就是输入图像与输出图像 RGB 信息的相似度。
- **模型保存：**将训练好的线性层保存为 pth 文件，以便之后实验的使用。
- **训练参数：**Adam 优化器，学习率 1e-5，训练步数 200.

4 实验结果

4.1 线性层训练结果

线性层损失变化如下所示：

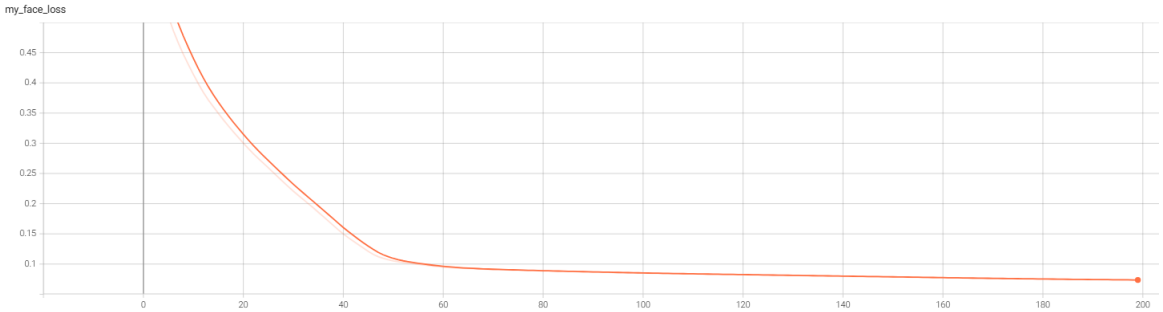


图 4: 损失变化

保存了原输入图像，首次生成图像以及最后一张生成图像如下所示：

(a) 原图　　(b) 第一张图　　(c) 最后一张图

可以看到生成效果并不理想，但是也能看出线性层网络确实也在发挥作用，例如，原图的皮肤颜色较白，头部偏向朝右等，这些特征在线性层的训练过程中都体现在了生成图像的变化中。

笔者尝试了增加训练步数，但是效果不佳，从损失变化图可以看出模型训练已经趋于收敛，即模型性能已经达到上限。因此笔者认为，如果想要进一步提升线性层的映射能力，需要对转换网络的结构进行更加复杂设计与改进，然而鉴于实验的实现时间有限，笔者将使用该线性层网络进行接下来的实验。

4.2 对于风格空间的编辑

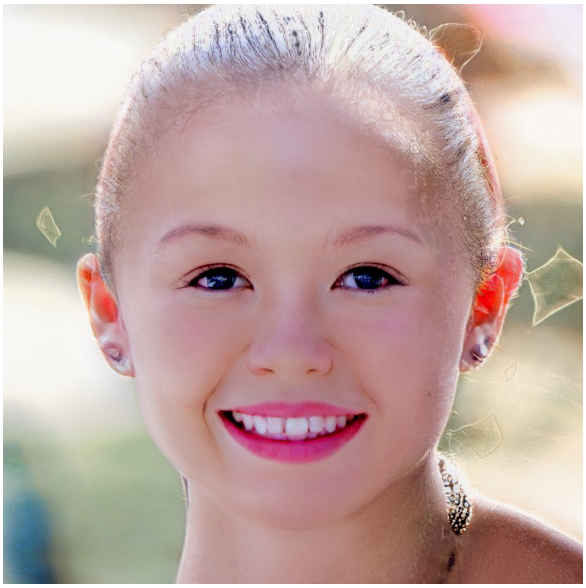
接下来使用训练并保存好的线性层网络进行 stylegan 风格变化的实验。为了改变 stylegan 的风格空间，需要将 generater 的 mapping 层和 synthesis 层拆分，并在之间加入风格空间变化方向。

```
w = G.mapping(generated_latent,None)  # 映射到风格空间
offset_1_4 = torch.randn(1, 4, 512).cuda()  # 对应1-4层的偏移量
w[:, 0:4, :] += offset_1_4*0.03  # 对1-4层添加偏移量
img_data = G.synthesis(w)
```

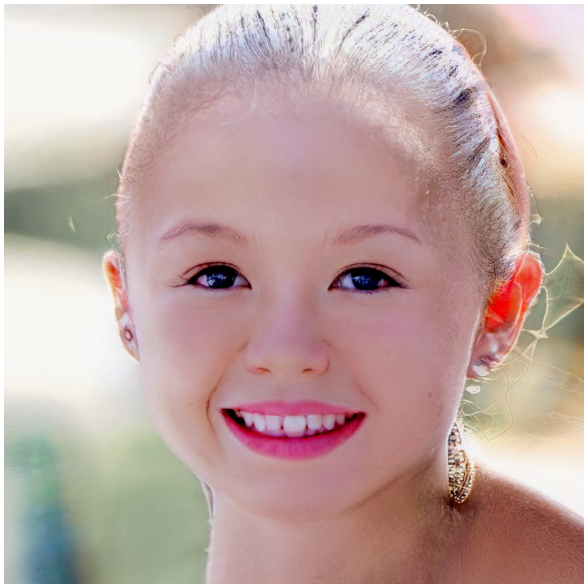
stylegan 设计的风格空间形状为 [batch,16,512]，因此一共有 16 个层次的风格可以调整，本实验依次调整了 1-4，4-8，8-12，12-16 层级的风格空间，调整步长为 0.03，调整步数为 50 步，offset 的具体实现如下所示：

```
offset_1_4 = torch.randn(1, 4, 512).cuda()  # 对应1-4层的偏移量
offset_4_8 = torch.randn(1, 4, 512).cuda()  # 对应4-8层的偏移量
offset_8_12 = torch.randn(1, 4, 512).cuda()  # 对应8-12层的偏移量
offset_12_16 = torch.randn(1, 4, 512).cuda()  # 对应12-16层的偏移量
w[:, 0:4, :] += offset_1_4*0.03  # 对1-4层添加偏移量
w[:, 4:8, :] += offset_4_8*0.03  # 对4-8层添加偏移量
w[:, 8:12, :] += offset_8_12*0.03  # 对8-12层添加偏移量
w[:, 12:16, :] += offset_12_16*0.03  # 对12-16层添加偏移量
```

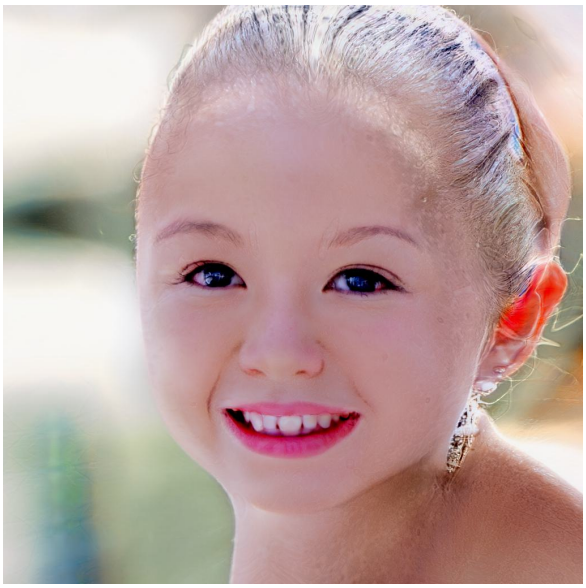
4.2.1 1-4 层调整结果



(d) step0



(e) step15



(f) step30

(g) step45

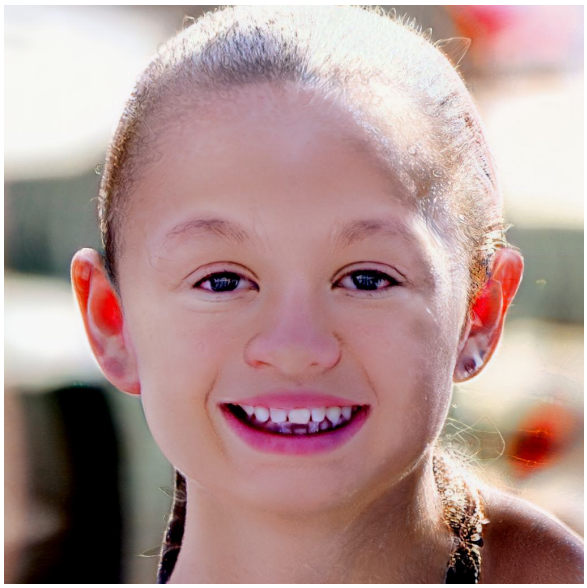
可以看到调整 1-4 层时，生成图像的脸部朝向发生了改变，头发纹理和颜色也有了细微变化。

4.2.2 4-8 层调整结果

(h) step0

(i) step15

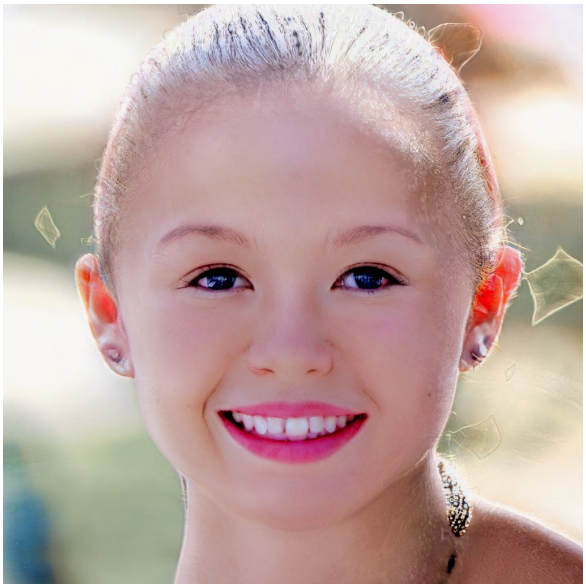
(j) step30



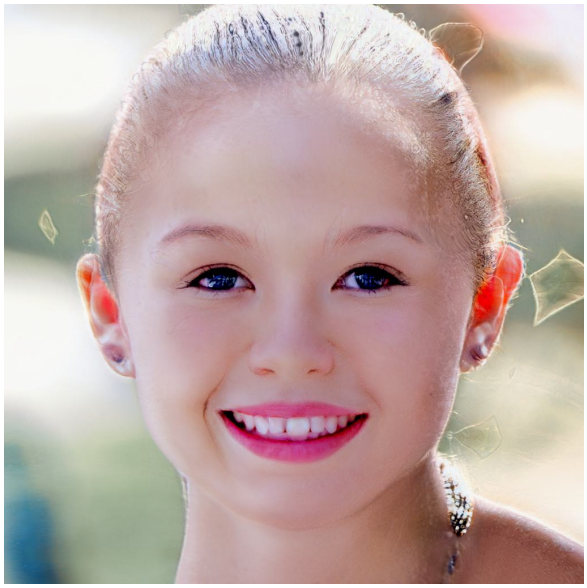
(k) step45

可以看到调整 4-8 层时，生成图像的脸部特征有了变化，似乎增加了褶皱的深度，眼球的位置也发生了变化。

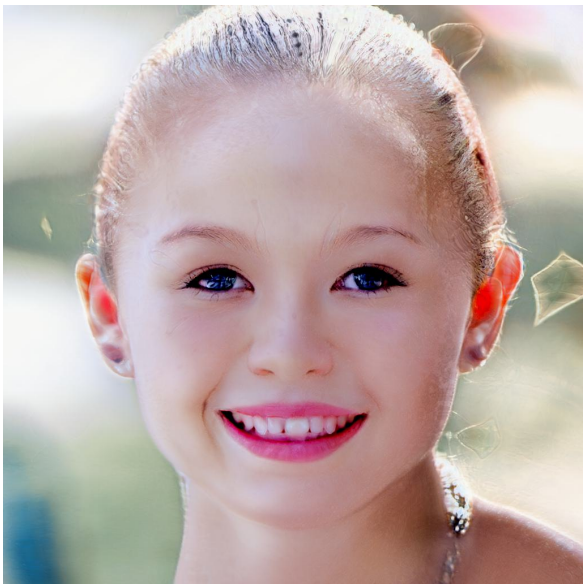
4.2.3 8-12 层调整结果



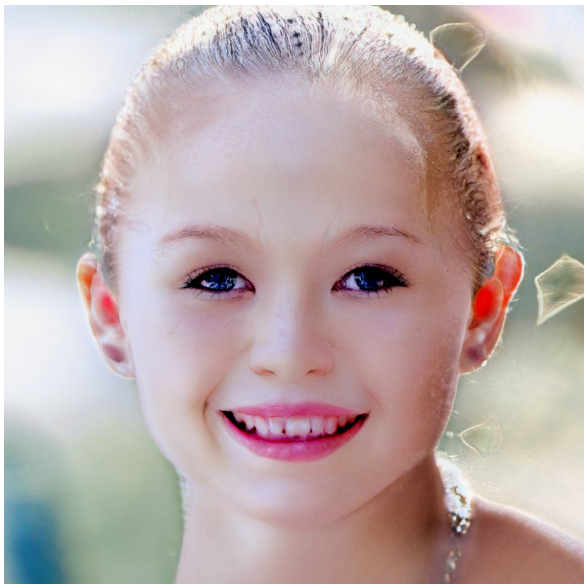
(l) step0



(m) step15



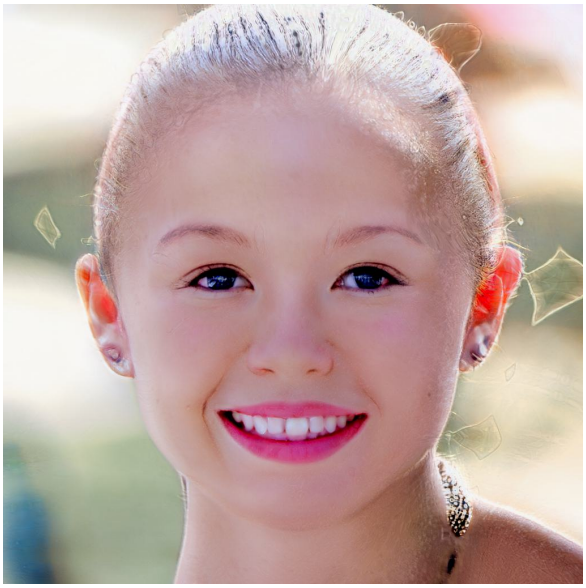
(n) step30



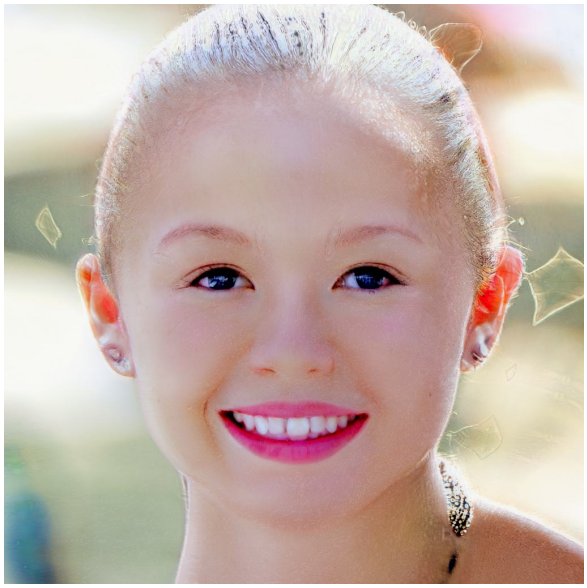
(o) step45

可以看到调整 4-12 层时，生成图像的肤色变白了。

4.2.4 12-16 层调整结果

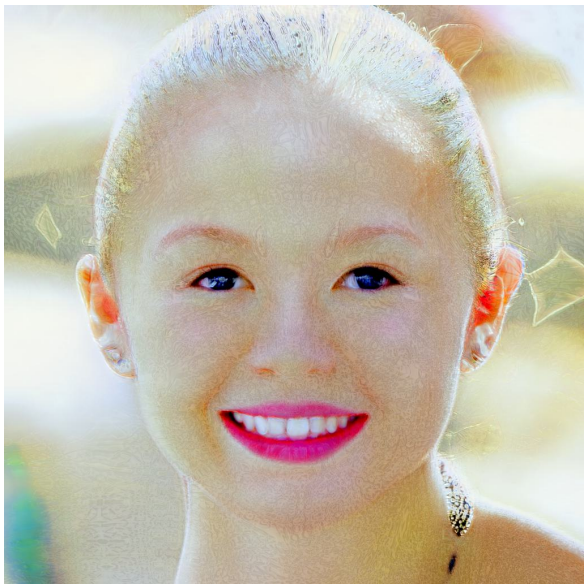


(p) step0



(q) step15

(r) step30



(s) step45

可以看到调整 12-16 层时，生成图像的背景颜色整体都发生了变化。

ps: 不要一直盯着图片，本人写报告的时候恐怖谷效应犯了，起了一身鸡皮疙瘩。。。