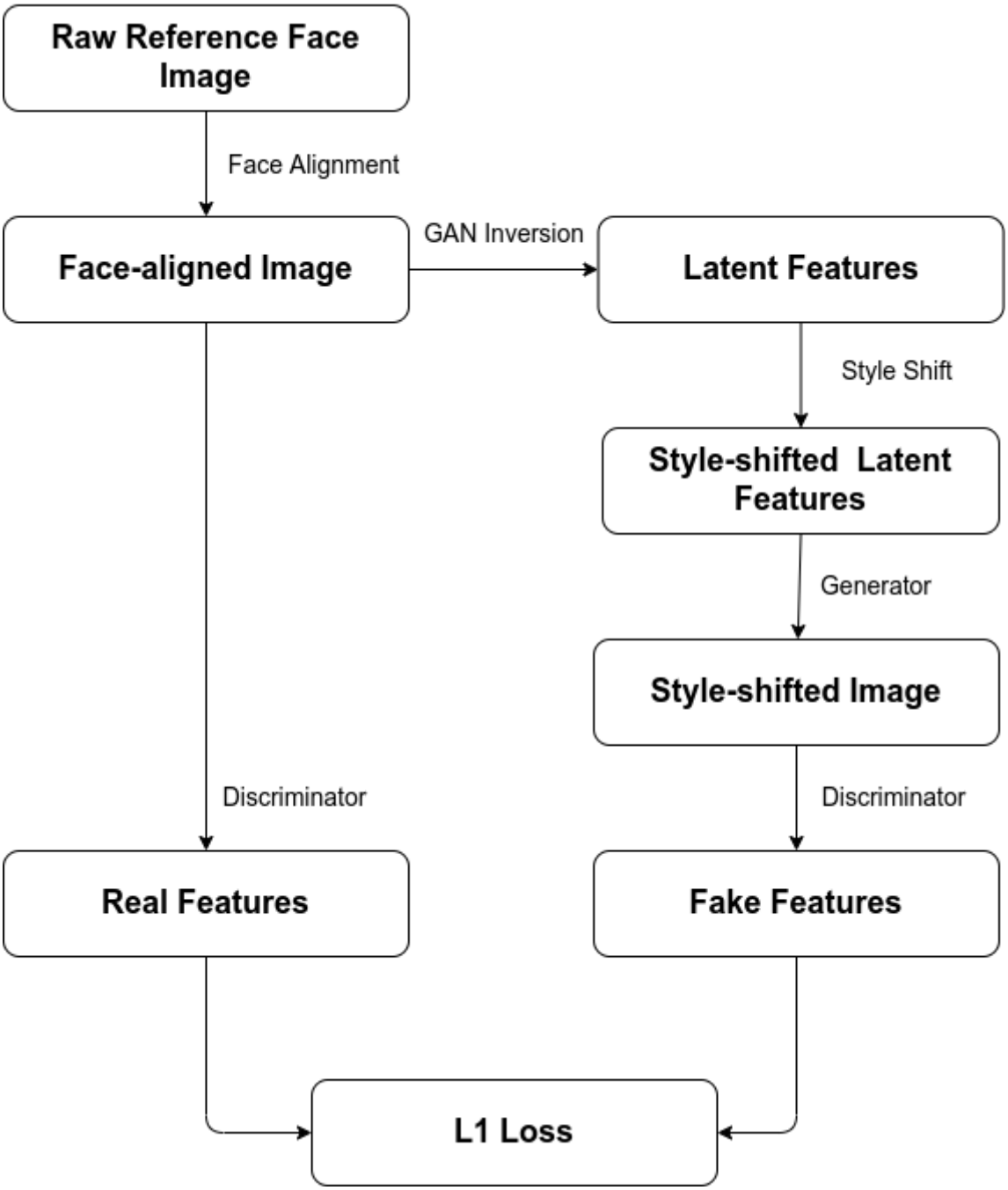
JoJoGAN in Details

Overview

JoJoGAN简单来说就是one-shot learning的fine-tuned版styleGAN，其结构图如下所示：



Face Alignment

face alignment 用于人脸对齐，封装在以下函数调用API内：

```
def align_face(filepath, output_size=1024, transform_size=4096,
enable_padding=True)
```

具体算法实现如下：

1. 使用**dlib**的**get_landmark**函数得到人脸图片的**landmark**

   get_landmark函数使用的**predictor**为pre_trained的**dlibshape_predictor_68_face_landmarks**

2. 从**landmark**中抽取脸部关键特征位置向量并计算关键属性

   脸部关键特征位置向量包括左右眼向量，鼻，嘴，眉，眼，鼻孔向量等，关键属性包括左眼中心点，右眼中心点，双眼中间点，嘴左端点，右端点，中心点，嘴眼距离等

3. 根据关键属性计算裁剪矩形位置大小

4. 根据所确定的裁剪矩形作放缩，裁剪，填充，变形

GAN Inversion

GAN Inversion是指根据图片生成lantent feature的过程，也就是GAN的逆过程。JoJoGAN采用e4e Projection作GAN Inversion，关于e4e Projection有以下介绍：

> Official Implementation of "Designing an Encoder for StyleGAN Image Manipulation" paper for both training and evaluation. The e4e encoder is specifically designed to complement existing image manipulation techniques performed over StyleGAN's latent space.

Style Shift

Style Shift通过在lantent feature的特定维度添加随机噪声实现，核心代码如下所示：

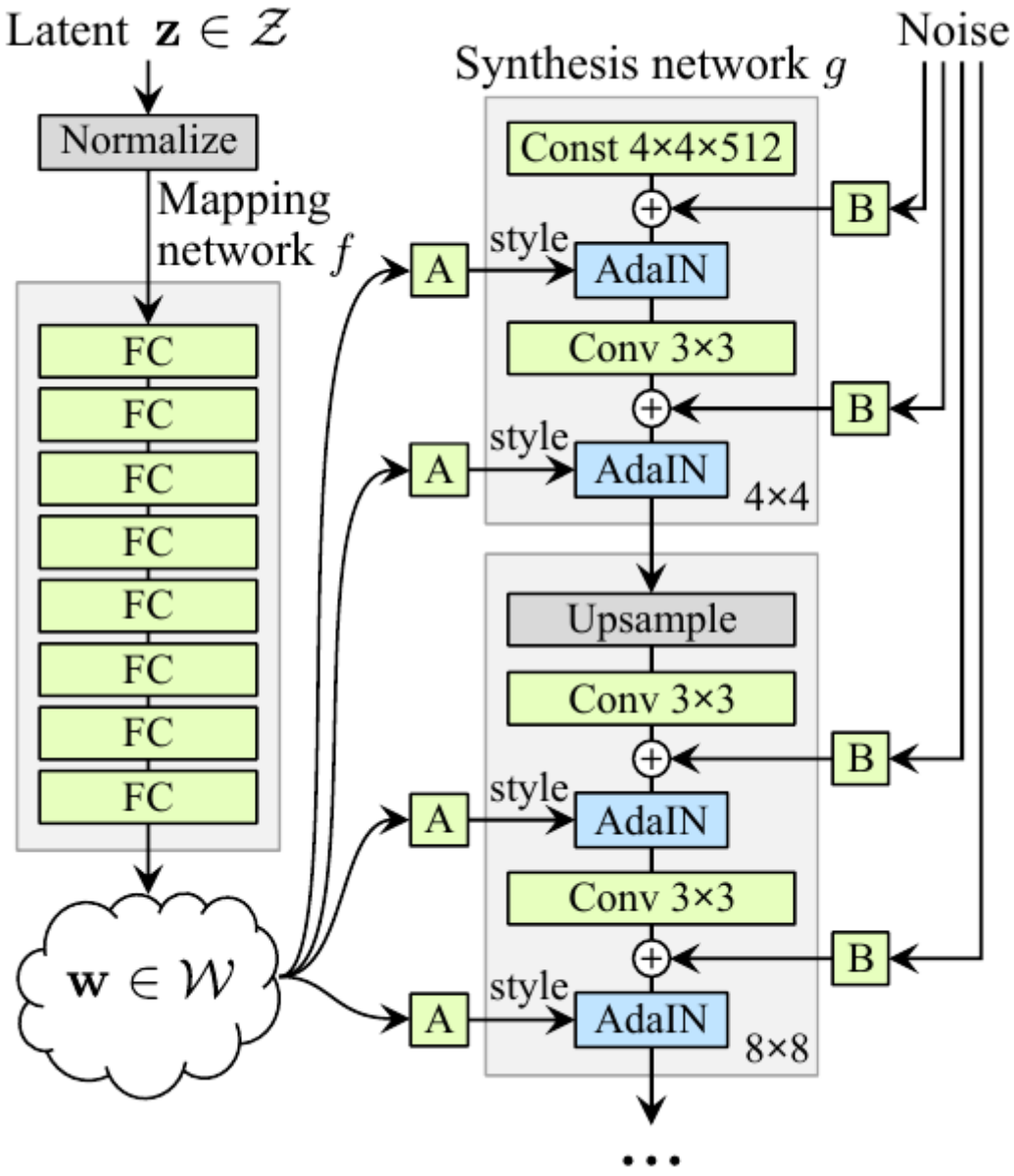
```
idx_swap = list(range(7, generator.n_latent))
 mean_w = generator.get_latent(torch.randn([latents.size(0), latent_dim]).to(
        device)).unsqueeze(1).repeat(1, generator.n_latent, 1)
    in_latent = latents.clone()
    in_latent[:, idx_swap] = alpha * latents[:, idx_swap] + \
        (1 - alpha) * mean_w[:, idx_swap]
```

具体来说，保留前七个维度不变，在剩余所有维度上添加正态噪声。

Generator

generator即为styleGAN网络结构，简单来说如下所述：

1. 判断输入是否为latent，若不是，即代表原图，则过几层线性层，如是，跳过。
2. 过一层StyleConv
3. 过一层UpSample生成RGB图片
4. 过styleGAN核心结构，即StyleConv和随机噪声混合层，其中作残差连接。具体结构如下图所示：



Discriminator

discriminator实现为几个ResBlock的线性堆叠，每个ResBlock实现如下：

1. 先过两个ConvLayer
2. 再与一个ConvLayer作残差连接

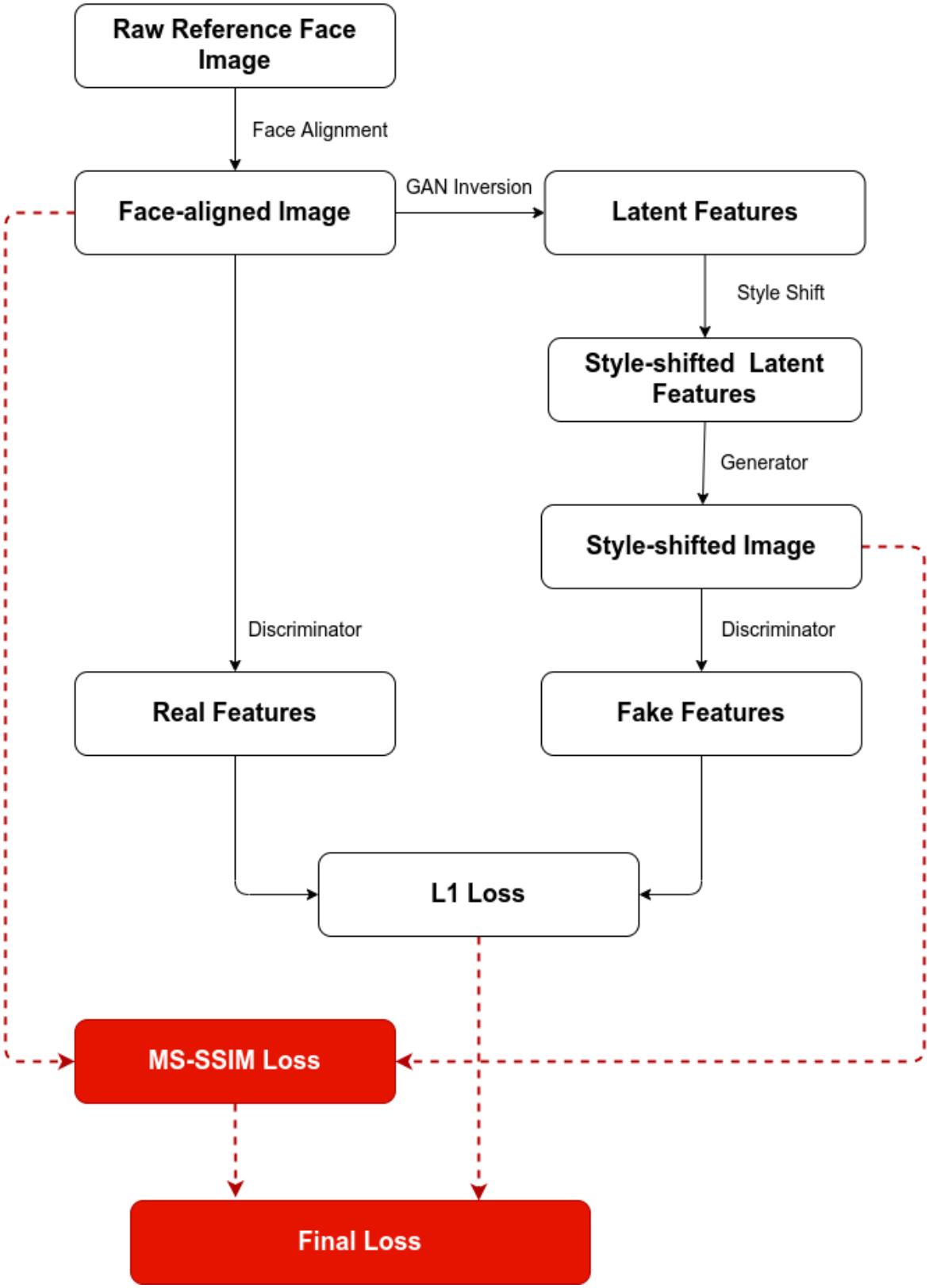
> 残差连接公式：$out = (out + skip)/\sqrt{2}$

LPIPS LOSS

JoJoGAN论文中表述Loss为**LPIPS Loss**，在代码中实现即为**discriminator**和**L1 Loss**的结合，核心思想即为使用深度CNN卷出来的feature直接输入神经网络，训练输出得到差异指标。关于**LPIPS Loss**出自论文[The Unreaonable Effectiveness of Deep Features as a Perceptual Metric]

Optimization

优化后的神经网络结构如下所示：



MS-SSIM Loss

Multi-Scale Structural Similarity (**MS-SSIM**)用于衡量两张图片的相似度，其公式如下所示：

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = [l_M(\mathbf{x},\mathbf{y})]^{\alpha_M} \cdot \prod_{j=1}^{M} [c_j(\mathbf{x},\mathbf{y})]^{\beta_j} [s_j(\mathbf{x},\mathbf{y})]^{\gamma_j}$$

将经典的**MS-SSIM Loss**与新兴的 **Perceptual Loss**相结合，可以提高模型对图片差异特征的捕获能力，优化模型的风格迁移与图片生成。

> 代码中使用pytorch-msssim提供的**MS-SSIM Loss**