Implicit Motion Function - Supplementary Material

Yue Gao Jiahao Li Lei Chu Yan Lu
Microsoft Research
{yuegao, li.jiahao, lei.chu, yanlu}@microsoft.com

1. Overview

In this supplementary material, we elaborate on the loss functions, implementation details, datasets, additional results on the comparisons with different methods, and more results of free pose and expression editing on wild identities.

2. Loss Functions

Our model is trained using the self-supervised learning pipeline with a reconstruction task.

Pixel-wise Loss $\mathcal{L}_p$. The pixel-wise loss is employed to ensure the synthesis frame $\hat{x}_c$ is similar to the current frame x_c.

$$\mathcal{L}_p = \mathbb{E}[\| \hat{x}_c - x_c \|_1]. \tag{1}$$

Perceptual Loss $\mathcal{L}_v$. Similar to the existing methods [3, 18, 20, 25], we use a pre-trained VGG [8] to guarantee consistency of high level characteristics between the current frame x_c and reconstructed frame $\hat{x}_c$.

$$\mathcal{L}_v = \mathbb{E}[\sum_i \sum_j \| \text{VGG}^j(\hat{x}_c^i) - \text{VGG}^j(x_c^i) \|_1], \tag{2}$$

where i represents that the frame is downsampled i times, and j is the layer index of the VGG. We employ settings consistent with existing methods [3, 18, 20, 25], i.e. $i \in [0, 3]$ and $j \in [0, 4]$.

GAN Loss $\mathcal{L}_G, \mathcal{L}_D$. To make the synthesized frames realistic, we adopt the hinge adversarial loss [11], and two different scale patch discriminator is used for better performance [7].

$$\begin{aligned} \mathcal{L}_G &= -\mathbb{E}[D(\hat{x}_c)], \\ \mathcal{L}_D &= \mathbb{E}[\max(0, 1 - D(x_c) + \max(0, 1 + D(\hat{x}_c)]. \end{aligned} \tag{3}$$

Full Objective Function. The total loss of the generation step is formulated as:

$$L_G = \lambda_p \mathcal{L}_p + \lambda_v \mathcal{L}_v + \lambda_G \mathcal{L}_G, \tag{4}$$

where λ_p, λ_v and λ_G are the weights of loss functions, which equals to 10, 10 and 1, respectively. The loss of the discrimination step is formulated as $L_D = \mathcal{L}_D$. We follow the standard GAN practice [7] during training.

3. Implementation Details

3.1. Model Details

The details of the model structures and sub-modules are shown in Figure 1. Our encoder-decoder framework mainly contains four parts, the *dense feature encoder E_F*, the *latent token encoder E_T*, the *implicit motion function* (IMF) and the *frame decoder D_F*, where the IMF is composed of the *latent token decoder IMF_D* and *implicit motion alignment IMF_A* . The ConvLayer [10] block and StyledConv [10] block are directly adopted from the StyleGAN2-pytorch [16] implementation. The E_T is composed of several ResBlocks [4] and downsample blocks, and a multilayer perceptron (MLP) is appended to the last, to finally obtain the latent token representation. The *latent token decoder IMF_D* is implemented with a StyleGAN2 [10] generator, and the latent token t_c is injected into the layers using the style modulation operation. For the *implicit motion alignment IMF_A* process, it can be formulated as:

$$\begin{aligned} V' &= \text{Attention}(Q, K, V), \\ &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V. \end{aligned} \tag{5}$$

With the aligned values V', we can further refine them using multi-head self-attention and feed-forward network-based Transformer blocks [19]. In this work, we use 4 stacked transformer decoder blocks.

$$\begin{aligned} \text{TransformerBlock}(x) &= \text{FFN}\Big(\text{MultiHeadSA}(x)\Big), \\ \text{head}_i &= \text{Attention}(xW_{Qi}, xW_{Ki}, xW_{Vi}), \\ \text{MultiHeadSA}(x) &= \text{Cat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W_O, \\ \text{FFN}(x) &= \text{GELU}\Big(\text{LN}(x)W_1 + b_1\Big)W_2 + b_2, \\ \text{GELU}(x) &= x \cdot \Phi(x), \end{aligned} \tag{6}$$

where the SA is the self-attention, which takes the output from the previous block, Cat is the concatenation operation, FFN is the feed-forward network, LN is the Layer Normalization [1], GELU [5] is utilized as the activation function and $\Phi(x)$ is the cumulative distribution function for Gaussian distribution.

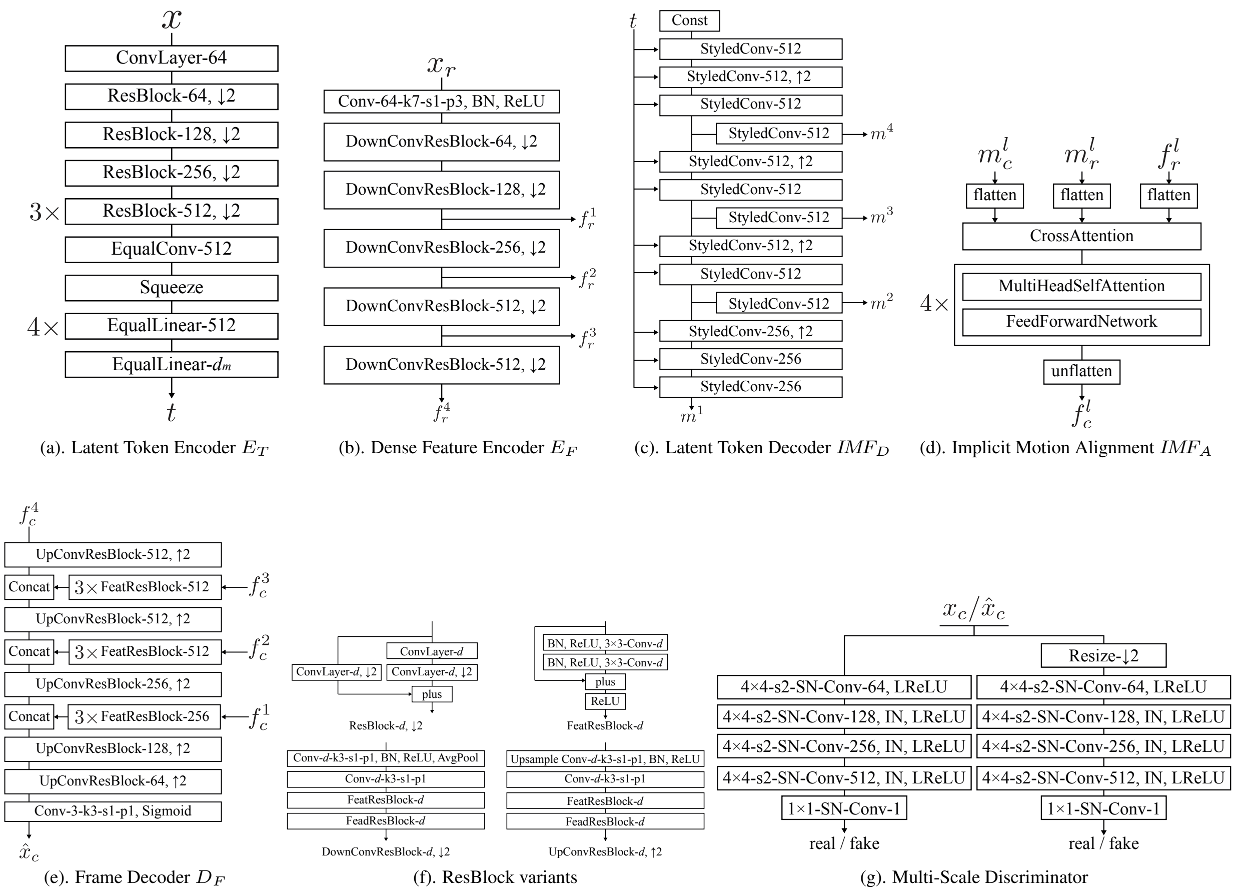

Figure 1. The detailed architectures of components in our model.

3.2. Datasets Details

Two talking head datasets, *i.e.*, CelebV-HQ [26], and VFHQ [23], are used in this paper. Apart from facial datasets, we also utilize three general datasets, *i.e.*, Flower, Wavecloth, and Foliage. For more details about these three datasets, please contact us. The resolution of frames is resized to 256×256 for all the experiments.

CelebV-HQ. The CelebV-HQ provides more than 35K video clips with diverse appearances, actions, and expressions, involving more than 15K identities.

VFHQ. The VFHQ dataset is mainly constructed for video face super-resolution, which contains over 16K high-fidelity clips of diverse interview scenarios, providing the highest frame quality among these datasets.

For VFHQ, we follow the approaches used in [23] to split the training and validation sets respectively, and report the performance of our model on the validation sets. For CelebV-HQ, we randomly select 500 videos for validation, as the official validation split is not provided.

GeneralVideo. We adopt a large-scale text-video dataset. Please contact us for the detail information of the dataset. We use the words "flower", "wavecloth" and "foliage" to filter the captions to obtain the sub-datasets Flower, Wavecloth, and Foliage. Flower sub-dataset contains 50,837 training videos and 89 validation videos. Wavecloth sub-dataset contains 47,707 training videos and 23 validation videos. Foliage sub-dataset contains 1,003 training videos and 118 validation videos. For the foliage experiments, we first pretrain all the methods on the flower dataset, and then fine-tune on the foliage dataset.

3.3. Optimization

The codebase for all these experiments is built upon PyTorch [14]. The Adam [12] optimizer is adopted with $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and the learning rate policy is set to 2×10^{-4}. The batch size is 64 over 8×32G NVIDIA Tesla V100 GPUs, in which 8 training samples will be