Unsupervised Geodesic-preserved Generative Adversarial Networks for Unconstrained 3D Pose Transfer

Haoyu Chen[1] Hao Tang[2] Henglin Shi[1] Wei Peng[1] Nicu Sebe[3] Guoying Zhao[1,*]
[1]CMVS, University of Oulu [2]Computer Vision Lab, ETH Zurich [3]DISI, University of Trento
{chen.haoyu, henglin.shi, wei.peng, guoying.zhao}@oulu.fi
hao.tang@vision.ee.ethz.ch nicu.sebe@unitn.it

The supplementary material includes technical details (this file) as well as the source code (.zip file) attached. The IEP-GAN architecture (Sec. 1) and implementing and training details (Sec. 2) as described in the main manuscript are all provided.

1. IEP-GAN Network Architectures

Our IEP-GAN framework consists of four main parts: a 3D mesh encoder, a generator, a global discriminator and an extrinsic discriminator. We first introduce each component in the IEP-GAN, then the whole model will be introduced.

1.1. 3D Encoders

The architecture of encoders is presented in Table 1. The encoder is derived from classical PointNet model [4] while we replace the batch normalization layers into Instance normalization as introduced in [5]. The input of the encoder is a 3d mesh with V vertices. The output is the latent codes of pose and shape information which are from the layers of Index "Pose code" and "shape code".

1.2. 3D Generator

The network architecture of the generator/decoder is presented in Table 2. It contains several SPAdaINResBlock as well as InstanceNorm block introduced from [6] specifically for 3D style transfer task. The detailed structure of a SPAdaIN ResBlock and an InstanceNorm block are presented in Table 3 and Table 4. The generator is used to take the shape code and pose code from the encoder and generate a new 3D mesh.

1.3. Global discriminator

The network architecture of the global discriminator is presented in Table 5. The design of the global discriminator is asymmetry of the generator except that we stack two linear connections to the last layer of the discriminator to produce the prediction of real or fake meshes.

*Corresponding Author. Code is available: https://github.com/mikecheninoulu/Unsupervised_IEPGAN

Table 1: 3D encoder architecture. "N" stands for batch size and "V" stands for vertex number. The first parameter of conv1d is the kernel size, the second is the stride size. The same as below. Pose code and shape code are obtained separately from the encoder. Note that the shape code is different for the task of disentanglement and pose transfer task.

Index	Inputs	Operation	Output Shape
(1)	3D mesh	Input mesh	N×3×V
Shape code (Pose transfer)	(1)	-	N×3×V
(2)	(1)	conv1d (1 × 1, 1)	N×64×V
(3)	(2)	Instance Norm, Relu	N×64×V
(4)	(3)	conv1d (1 × 1, 1)	N×128×V
(5)	(4)	Instance Norm, Relu	N×128×V
(6)	(5)	conv1d (1 × 1, 1)	N×512×V
(7)	(6)	Instance Norm, Relu	N×512×V
(8)	(7)	conv1d (1 × 1, 1)	N×512×V
Pose code	(8)	Instance Norm, Relu	N×512×V
(9)	(7)	conv1d (1 × 1, 1)	N×1024×V
(10)	(9)	Instance Norm, Relu	N×1024×V
(12)	(11)	conv1d (1 × 1, 1)	N×2048×V
Shape code (Disentanglement)	(12)	Instance Norm, Relu	N×2048×V

Table 2: 3D generator architecture. It takes the shape code and pose code to generate corresponding meshes with given latent code information.

Index	Inputs	Operation	Output Shape
(1)	Pose code	-	N×256×V
(2)	Shape code	-	N×6890×V
(3)	(1)	conv1d (1 × 1, 1)	N×128×V
(4)	(3)(2)	SPAdaIN ResBlock 1	N×128×V
(5)	(5)	conv1d (1 × 1, 1)	N×512×V
(6)	(5)(2)	SPAdaIN ResBlock 2	N×512×V
(7)	(6)	conv1d (1 × 1, 1)	N×256×V
(8)	(7)(2)	SPAdaIN ResBlock 3	N×256×V
(9)	(8)	conv1d (1 × 1, 1)	N×3×V
(10)	(9)	Tanh	N×3×V

Table 3: SPAdaIN ResBlock architecture. C stands for the channel number for each layer, which varies according to the layer sizes in the full generator structure introduced in Table. 2.

Index	Inputs	Operation	Output Shape
(1)	Shape code	-	N×C×V
(2)	Pose code	-	N×6890×V
(14)	(1)(2)	InstanceNorm block	N×C×V
(15)	(14)	conv1d(1 × 1, 1), Relu	N×C×V
(16)	(14)(15)	InstanceNorm block	N×C×V
(17)	(16)	conv1d(1 × 1, 1), Relu	N×C×V
(18)	(12)(13)	InstanceNorm block	N×C×V
(19)	(18)	conv1d(1 × 1, 1), Relu	N×C×V
(20)	(17)(19)	Add	N×C×V

Table 4: InstanceNorm block architecture. It is used to normalize the input data without repealing the instance information which is effective in style transfer learning task.

Index	Inputs	Operation	Output Shape
(1)	Pose code	-	N×C×V
(2)	(1)	Instance Norm	N×C×V
(3)	Shape code	-	N×6890×V
(4)	(3)	conv1d (1 × 1, 1)	N×C×V
(5)	(3)	conv1d (1 × 1, 1)	N×C×V
(6)	(4)(2)	Multiply	N×C×V
(7)	(6)(5)	Add	N×C×V

Table 5: 3D discriminator architecture. It takes a generated mesh as input and predict if the mesh is a fake one or not.

Index	Inputs	Operation	Output Shape
(1)	Generated mesh	-	N×3×V
(3)	(1)	conv1d (1 × 1, 1)	N×256×V
(4)	(3)(2)	SPAdaIN ResBlock 1	N×256×V
(5)	(5)	conv1d (1 × 1, 1)	N×512×V
(6)	(5)(2)	SPAdaIN ResBlock 2	N×512×V
(7)	(6)	conv1d (1 × 1, 1)	N×1024×V
(8)	(7)(2)	SPAdaIN ResBlock 3	N×1024×V
(9)	(8)	AdaptiveMaxPool2d	N×4×512
(10)	(9)	Flatten	N×2048
(11)	(10)	Linear	N×1024
(12)	(11)	Linear	N×1

1.4. Laplacian co-occurrence discriminator

The structure of the Laplacian co-occurrence discriminator is presented in Table 6. The design is similar to the global discriminator but the co-occurrence one is a multi-head structure that takes both generated meshes and reference meshes. The Laplacians of the meshes are obtained by implementing classical Laplacian transform of the original meshes with random iterations between 1 to 100.

Table 6: Laplacian co-occurrence architecture. It takes several Laplacians as input and predict if there are in the same pattern or not. Fake and Ref Laplacian stand for Laplacian from generated pose and real pose mesh. n stands for the reference mesh number.

Index	Inputs	Operation	Output Shape
(1)	Fake Laplacian	-	N×3×V
(2)	(1)	conv1d (1 × 1, 1)	N×64×V
(3)	(2)	Instance Norm, Relu	N×64×V
(4)	(3)	conv1d (1 × 1, 1)	N×128×V
(5)	(4)	Instance Norm, Relu	N×128×V
(6)	(5)	conv1d (1 × 1, 1)	N×256×V
(7)	(6)	Instance Norm, Relu	N×256×V
(8)	(7)	conv1d (1 × 1, 1)	N×256×V
(9)	(8)	Instance Norm, Relu	N×256×V
(10)	Ref Laplacian	-	N×3×V
(11)	(10)	conv1d (1 × 1, 1)	N×n×64×V
(12)	(11)	Instance Norm, Relu	N×n×64×V
(13)	(12)	conv1d (1 × 1, 1)	N×n×128×V
(14)	(13)	Instance Norm, Relu	N×n×128×V
(15)	(14)	conv1d (1 × 1, 1)	N×n×256×V
(16)	(15)	Instance Norm, Relu	N×n×256×V
(17)	(16)	conv1d (1 × 1, 1)	N×n×256×V
(18)	(17)	Instance Norm, Relu	N×n×256×V
(19)	(18)	Mean	N×256×V
(20)	(19)(9)	Concatenate	N×512×V
(21)	(20)	AdaptiveMaxPool2d	N×16×16
(22)	(21)	Linear	N×1024
(23)	(22)	Linear	N×1

2. Training Settings

The IEP-GAN is implemented in PyTorch [3]. The hardware environment for training is a remote server with a single NVIDIA Tesla V100, 32GB. The rum time testing is conducted on a local PC with a single GPU NVIDIA GTX 1080Ti, CPU Intel Core i7. We train our networks for 40,000 epochs with a learning rate of 0.00005 and Adam optimizer. The batch size is fixed as 4 for all the settings. Training time is around 26-30 hours. The reference Laplacian number is set as three. The vertex number of FAUST and DFAUST datasets are kept unchanged as 6,890 without down-sampling. The vertex number for local regional geodesic calculation is set as 400. For each training iteration, two local regions will be adaptively sampled.

Training GAN is extremely hard and easily encounters the generator collapses issue or degenerate meshes, thus we design the training of the IEP-GAN with three stages. For the first 2×10^4 iterations, only reconstruction is conducted to stabilize the GAN and avoid local minima, where the pair of reconstruction adversarial loss is used with raw SMPL meshes from [6]. From iterations 2×10^4 to 3×10^4, the pose transfer learning starts with the swapping loss and extrinsic preservation loss added on [1] or [2]. The intrinsic preservation loss will be added after 3×10^4 iterations.

References

[1] Federica Bogo, Javier Romero, Matthew Loper, and Michael J Black. Faust: Dataset and evaluation for 3d mesh registration. In *CVPR*, 2014. 2

[2] Federica Bogo, Javier Romero, Gerard Pons-Moll, and Michael J Black. Dynamic faust: Registering human bodies in motion. In *CVPR*, 2017. 2

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 2

[4] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 1

[5] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 1

[6] Jiashun Wang, Chao Wen, Yanwei Fu, Haitao Lin, Tianyun Zou, Xiangyang Xue, and Yinda Zhang. Neural pose transfer by spatially adaptive instance normalization. In *CVPR*, 2020. 1, 2