Invertible Neural Skinning (Appendix)

Paper ID 3954

A. Implementation Details

A.1. Sampling Points

Following SNARF, we sample 200K points at every frame of the sequence. Half of these points (100K) are near the mesh (scan) surface, which are obtained by first sampling points on the mesh surface via Poisson disk sampling and followed by displacement with isotropic Gaussian noise (of $\sigma = 0.01$). Remaining half (100K) points are sampled uniformly within a bounding box scaled to 110% of the original bounding box.

A.2. Hyperparameters and Training Details

We trained all our models on a single Tesla V100 GPU for 250 epochs, which took nearly 40 hours on average. We used a learning rate of $1e{-}4$ to train the PINs, while using a learning rate of $1e{-}3$ for remaining modules. We used Adam [5] optimizer, with a linear warmup and no learning rate decay. PyTorch [9] is used for all the experiments. Please refer to Table 1 for full list.

A.3. Metrics

Given set of sampled points $\mathbf{P}$ to be evaluated, we can represent the joint tuple of any point, its ground truth occupancy (which can be either 0 or 1), and predicted occupancy as $(\mathbf{p}_d^i, g^i, h^i) \;\; \forall \;\; \mathbf{p}_d^i \in \mathbf{P}$ respectively. Then Intersection over Union (IoU) can be computed as follows:

$$\text{IoU} = \sum_{\mathbf{p}_d^i \in \mathbf{P}} \frac{g^i \cap h^i}{g^i \cup h^i} \tag{1}$$

To convert predicted probability to binary occupancy, we simply check if it is greater than 0.5. IoU Bounding Box operates with points sampled uniformly in the space, whereas Surface IoU operates with points sampled close to the body as described in Section A.1.

B. Data

CAPE. CAPE originally contains 15 subjects, with each subject wearing 1-6 different types of clothing, and performing 3-74 different actions. On average, it contains nearly 249 frames for every *subject-cloth* pair. Due to high variance as well as high number of *subject-cloth* pairs, we use a subset of CAPE which contains 15 sequences of 13 subjects containing all 8 different types of clothings. A clothing in CAPE is denoted by a joint string of upper and lower body garment, for example, a subject wearing a blazer and pants is annotated as *blazerlong*, and so on.

C. Invertible Neural Network

C.1. Initialization

We found that initializing the Pose-conditioned Invertible Networks (PINs) as identity modules stabilizes training, and allows the LBS network to train better. For this, we initialize the weights and biases of the last layer of the operation maps $\mathbf{m}_r$ and $\mathbf{m}_t$ (shown in Figure 2) as zeros.

C.2. Volume Preservation

Previous works in INNs [2, 3, 6] operating on high-dimensional ($\geq$ 512-d) spaces constrained the Jacobian between input and output to an triangular matrix. This ensured that the Jacobian determinant, used for density modeling, was not expensive to compute. Determinant of a triangular matrix is simply multiplication of its diagonal. However, this prevented these works from using 2D operators such as rotation. We note that such a requirement is unnecessary in our setting, where Jacobian determinant is not needed. Additionally, using rotations also helps to preserve volume between the input and output spaces.

Next, we show that our PINs consisting of only rotations and translations are volume preserving. Note that to show a transform is volume preserving it is sufficient to show that the determinant of the Jacobian of this transform is one. From Equations 5 and 10, we can express the transform represented by a single 2D coupling layer as:

$$\begin{aligned} x' &= x\cos(\gamma_{xy}) - y\sin(\gamma_{xy}) + t_x \\ y' &= x\sin(\gamma_{xy}) + y\cos(\gamma_{xy}) + t_y \\ z' &= z \end{aligned}$$

Then Jacobian of this transform becomes:

$$\mathbf{J} = \begin{bmatrix} \cos(\gamma_{xy}) & -\sin(\gamma_{xy}) & 0 \\ \sin(\gamma_{xy}) & \cos(\gamma_{xy}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

And determinant of the above Jacobian is one, i.e. $|\mathbf{J}| = 1$. Since, our PINs are composed of chaining together such coupling layers described in Equation 13, the overall determinant is also one. Hence volume is preserved within PINs.

#	Hyperparameters	Value	#	Hyperparameters	Value
1	No. of Parameters in INS	1.80M	2	No. of Coupling Layers in $\mathbf{H}_d$/$\mathbf{H}_c$	18
3	No. of Parameters in PIN $\mathbf{H}_d$/$\mathbf{H}_c$	0.41M	4	No. of Parameters in Occupancy Network $\mathbf{O}$	0.46M
5	No. of Parameters in LBS Network	53K	6	No. of Parameters in Bone Encoder	0.46M
7	Pose Embedding Dimension	120	8	Space Embedding Dimension	120
9	Space and Pose Embedding	240	10	PIN input and output Dimension	3
11	Number of epochs	250	12	Optimizer	Adam
13	Batch size (DFAUST/CAPE)	12/8	14	Learning rate (INNs/Rest)	1e-4/1e-3
15	Warm-up learning rate factor	0.2	16	Warm-up iterations	2400
17	No. of points per batch	60000	18	Gradient clipping (L-2 Norm)	4.0

Table 1. Hyperparameters and Training configuration to train INS.

D. Gradients

Training INS requires calculating gradients of the Binary Cross Entropoy (BCE) loss $\mathcal{L}_{bce}$ (Equation 18), with respect to all the components. Let the weights of PINs $\mathbf{H}_c$, $\mathbf{H}_d$, the LBS network $\mathbf{w}_{lbs}$, and the Occupancy network $\mathbf{O}$ be denoted with σ_c, σ_d, σ_{lbs}, and σ_o respectively. Backpropagating through the occupancy network $\mathbf{O}$ and the PIN $\mathbf{H}_c$ is straightforward:

$$\frac{\partial \mathcal{L}_{bce}}{\partial \sigma_o} = \frac{\partial \mathcal{L}_{bce}}{\partial o} \cdot \frac{\partial o}{\partial \mathbf{O}(\mathbf{p}_c)} \cdot \frac{\partial \mathbf{O}(\mathbf{p}_c)}{\partial \sigma_o} \tag{3}$$

$$\frac{\partial \mathcal{L}_{bce}}{\partial \sigma_c} = \frac{\partial \mathcal{L}_{bce}}{\partial \mathbf{O}(\mathbf{p}_c)} \cdot \frac{\partial \mathbf{O}(\mathbf{p}_c)}{\partial \mathbf{H}_c(\mathbf{q}_c^*)} \cdot \frac{\partial \mathbf{H}_c(\mathbf{q}_c^*)}{\partial \sigma_c} \tag{4}$$

where o is the predicted occupancy. While the gradients for LBS network $\mathbf{w}_{lbs}$ and second PIN $\mathbf{H}_d$ are:

$$\frac{\partial \mathcal{L}_{bce}}{\partial \sigma_{lbs}} = \frac{\partial \mathcal{L}_{bce}}{\partial \mathbf{H}_c(\mathbf{q}_c^*)} \cdot \frac{\partial \mathbf{H}_c(\mathbf{q}_c^*)}{\partial \mathbf{q}_c^*} \cdot \frac{\partial \mathbf{q}_c^*}{\partial \sigma_{lbs}} \tag{5}$$

$$\frac{\partial \mathcal{L}_{bce}}{\partial \sigma_d} = \frac{\partial \mathcal{L}_{bce}}{\partial \mathbf{q}_c^*} \cdot \frac{\partial \mathbf{q}_c^*}{\partial \mathbf{H}_d(\mathbf{p}_d^t)} \cdot \frac{\partial \mathbf{H}_d(\mathbf{p}_d^t)}{\partial \sigma_d} \tag{6}$$

where $\mathbf{q}_c^*$ is the root of the Equation 17, and $\mathbf{p}_d^t$ is the input point. Pytorch's automatic differentiation can handle the gradients in Equations 22 and 23. However, to obtain gradients w.r.t. $\mathbf{q}_c^*$implicit differentiation is required, similar to SNARF:

$$\begin{aligned} & \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t) - \mathbf{p}_d^t = \mathbf{0} \\ \Leftrightarrow \; & \frac{\partial \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t)}{\partial \sigma_{lbs}} + \frac{\partial \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t)}{\partial \mathbf{q}_c^*} \cdot \frac{\partial \mathbf{q}_c^*}{\partial \sigma_{lbs}} = \mathbf{0} \\ \Leftrightarrow \; & \frac{\partial \mathbf{q}_c^*}{\partial \sigma_{lbs}} = -\left(\frac{\partial \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t)}{\partial \mathbf{q}_c^*}\right)^{-1} \cdot \frac{\partial \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t)}{\partial \sigma_{lbs}} \end{aligned} \tag{7}$$

And we can find gradients of $\mathbf{q}_c^*$ with respect to $\mathbf{q}_d^t$ as follows:

$$\begin{aligned} & \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t) - \mathbf{p}_d^t = \mathbf{0} \\ \Leftrightarrow \; & \frac{\partial \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t)}{\partial \mathbf{q}_c^*} \cdot \frac{\partial \mathbf{q}_c^*}{\partial \mathbf{H}_d(\mathbf{p}_d^t)} + \mathbf{1} = \mathbf{0} \\ \Leftrightarrow \; & \frac{\partial \mathbf{q}_c^*}{\partial \mathbf{H}_d(\mathbf{p}_d^t)} = -\left(\frac{\partial \mathbf{lbs}(\mathbf{q}_c^*, \boldsymbol{\theta}^t)}{\partial \mathbf{q}_c^*}\right)^{-1} \end{aligned} \tag{8}$$

E. Miscellaneous Failed Experiments

In order to help with a future research in this direction we list several ideas that have been tried in our project, which however did not improve performance.

E.1. Invertible Residual Layers

Idea. Beyond the invertible space-splitting layers, we also experimented with using invertible residual layers [1, 4]. These layers operate by limiting the Lipschitz constant of the residual branch, which has be less than one in order to guarantee invertibility. Inversion of these layers can be achieved using a fixed-point iteration method, with convergence rate exponential in the number of iterations.

Outcome. We tried chaining residual layers with coupling layers alternately, and also placing them in the start and end of the invertible networks. However, these setups performed close or worse than without using residual layers, and were slower due to the expensive inversion pass.

E.2. Coupling Layers with Scales

Idea. Previous works CaDeX [7], and NeuralParts [8] utilized invertible networks with scale and translation operations, following the architecture proposed in RealNVP [3]. Such a transform can be represented as:

$$\begin{aligned} x' &= x\exp(s_x) + t_x \\ y' &= y\exp(s_y) + +t_y \\ z' &= z \end{aligned}$$

Outcome. When using these layers in our experiments we encountered the following difficulties:

- **Unstable Training.** Floating point overflows occurred frequently during training due to the exponential scaling term. Even after carefully tuned gradient clipping and learning rate schedules, we encountered frequent experiment failures.

- **Squashing Effect.** Since the scaling operator can lead to very large outputs from INN, generally a sigmoid squashing layer is used at the end to restrict the input to a fixed range that matches output distribution. However, due to this sigmoid layer, the INN can no longer be initialized as an Identity layer, even when all the rotations are identity and translations are zero. This leads to a squashing artefacts in outputs.

- **Non-volume Preserving.** The Jacobian of these layers [7] with scaling is not one, previously derived Equation 2. Due to this additional regularization is needed for training.

E.3. Pose-conditioned 3D rotation and translation layers

Idea. We tried learning pose-conditioned global 3D rotation and translation layers. We implemented it similar to the coupling layers to predict rotation and translation parameters, but without any space conditioning.

Outcome. We did not find significant gains using this, and decided against using them in the final version as they had a big memory footprint. These layers often rotate the canonical space creating issues during mesh extraction.

F. Visuals

We place all the qualitative results in the video file called `qual-video.mp4` *of the supplementary material, and discuss its contents below.*

[Video Part-1] Pose-varying deformations in INS. In the first part of the video, we visualize the deformations introduced by PINs $\mathbf{H}_c$ and $\mathbf{H}_d$ under varying target poses (shown in top right). Deformations introduced by $\mathbf{H}_c$ are shown in the top-middle part, whereas those introduced by $\mathbf{H}_d$ are shown in the bottom-left part shaded in green. We demonstrate that INS is able to handle complex deformations of clothing across poses.

[Video Part-2] Baseline Comparison. In the second part of the video, we compare our method INS against all the five baselines discussed in Section 4.2 of the main paper. While both the LBS baselines, and SNARF-NC suffers from artifacts, we see that INS performs much better than other methods.

[Video Part-3] INS Ablations. In the third part of the video, we visualize results from various ablations reported in Section 4.4 of the main paper. Here, we find that removing SIREN leads to an overly smooth surface, and removing the LBS network makes it harder for the network to learn limb movements correctly.

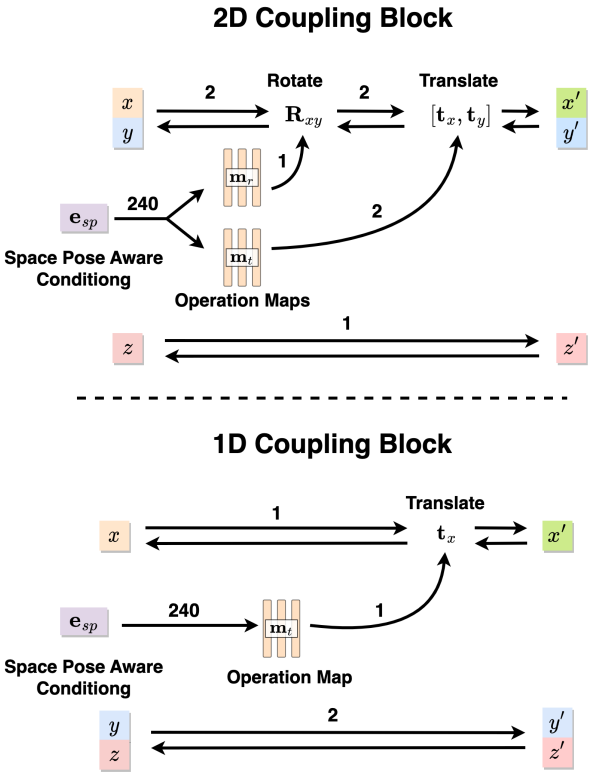

Figure 1. **1D and 2D Coupling Layers.** We show comparison between both types of layers used in PINs. The bidirectional arrows show invertible computations.

[Video Part-4] Texture Propagation. As INS can preserve correspondences across poses, it becomes possible to propagate mesh attributes such as texture across various time frames. We conducted an experiment to test this, where we applied texture to the pose-independent canonical mesh. Next, we propagated this texture through the INS network. We show the results of this experiment in the fourth (and last) part of the video. We found that the applied texture deformed realistically like clothing, while being consistent across all frames, and was free of jittering.

To contrast and compare with the above experiment, we conducted similar texture propagation using SNARF. Since, SNARF decodes a separate mesh at each time-step, we color this mesh using the same scheme for coloring INS canonical mesh above. Propagating this texture through the LBS block, we find that it frequently leads to jittery artefacts as the texture overflows across semantically different parts. For example, the texture patch E4 applied to the blazer in a particular frame, overflows onto pants in another frame, and so on.

F.1. 1D and 2D Coupling Layers

We visualize both 1D and 2D coupling layers together in Figure 1 for better understanding. In 1D case the space-pose aware conditioning gets conditioned on the 2D input, which helps to improve expressiveness. In the 2D case, we can make edits on an entire 2D plane conditioned on the 1D input. We find out that these blocks provide complementary benefits, thus we utilize both of them in the final architecture.

References

[1] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019. 2

[2] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 1

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 1, 2

[4] Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018. 2

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2015. 1

[6] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018. 1

[7] Jiahui Lei and Kostas Daniilidis. Cadex: Learning canonical deformation coordinate space for dynamic surface representation via neural homeomorphism. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2, 3

[8] Despoina Paschalidou, Angelos Katharopoulos, Andreas Geiger, and Sanja Fidler. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3204–3215, 2021. 2

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 1