Supplementary Material for "PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows"

Guandao Yang[1,2*], Xun Huang[1,2*], Zekun Hao[1,2], Ming-Yu Liu[3], Serge Belongie[1,2], Bharath Hariharan[1]
[1]Cornell University [2]Cornell Tech [3]NVIDIA

A. Overview

In the appendix, we first describe the detailed hyper-parameters and model architectures for our experiments in Section B. We then compare our model with additional baselines to understand the effect of different model components in Section C. Limitations and typical failure cases are discussed in Section D. Finally, additional visualizations of latent space t-SNE, interpolations and flow transformations are presented in Section E, Section F, and Section G respectively.

B. Training details

In this section, we provide details about our network architectures and training hyper-parameters. We will release the code to reproduce our experiments. Please refer to algorithm 1 for the detailed training procedure.

Encoder. The architecture of our encoder follows that of Achlioptas *et al*. [1]. Specifically, we first use 1D Convolution with filter size 128, 128, 256, and 512 to process each point independently and then use max pooling to create a 512-dimension feature as done in PointNet [8]. Such a feature is invariant to the permutation of points due to the max-pooling. Finally, we apply a three-layer MLP with 256 and 128 hidden dimensions to convert the permutation invariant feature to a D_z-dimension one. For the unsupervised representation learning experiment, we set $D_z = 512$ following convention. For all other experiments, D_z is set to 128.

CNF prior. The CNF prior models the distribution $P_\psi(z)$. We follow FFJORD [4]'s released code to use three `concatsquash` layers to model the dynamics f_ψ. A `concatsquash` layer is defined as:

$$\mathrm{CS}(x,t) = (W_x x + b_x)\sigma(W_t t + b_t) + (W_b t + b_b t), \quad (1)$$

where W_x, b_x, W_t, b_t, W_b, and b_b are all trainable parameters and $\sigma(\cdot)$ is the sigmoid function. f_ψ uses three `concatsquash` layers with a hidden dimension 256. Tanh is used as the non-linearity between layers.

We use a Moving Batch Normalization layer to learn the scale of each dimension before and after the CNF, following FFJORD's released code [4]. Specifically, Moving Batch Normalization is defined as

$$\mathrm{MBN}(x) = \frac{x-\mu}{\sigma}\gamma + \beta, \quad (2)$$

where γ and β are trainable parameters, Different from batch normalization proposed by Ioffe and Szegedy [6], μ and σ are running averages of the batch mean and standard deviation. MovingBatchNorm is invertible : $\mathrm{MBN}^{-1}(y) = \frac{y-\beta}{\gamma}\sigma + \mu$. Its log determinant is given as:

$$\log\det\left|\frac{\partial\,\mathrm{MBN}(x)}{\partial x}\right| = \sum_i \log|\gamma_i| - \log|\sigma_i|. \quad (3)$$

CNF decoder. The CNF decoder models the reconstruction likelihood $P_\theta(X|z)$. We extend the `concatsquash` layer to condition on the latent vector z:

$$\begin{aligned}\mathrm{CCS}(x,z,t) = (W_x x + b_x)\sigma(W_{tt}t + W_{tz}z + b_t) \\ + (W_{bt}t + W_{bz}z + b_b t),\end{aligned} \quad (4)$$

where $W_x, W_{tt}, W_{tz}, W_{bt}, W_{bz}, b_t, b_b$ are all learnable parameters. The CNF decoder uses four conditional `concatsquash` layers with a hidden dimension 512 to model the dynamic g_θ. The non-linearity between layers is Tanh. Similar to the CNF prior model, we also add a Moving Batch Normalization layer before and after the CNF. In this case, all 3D points (from different shapes) from a batch are used to compute the batch statistics.

Hyper-parameters. We use an Adam optimizer with an initial learning rate 0.002, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The learning rate decays linearly to 0 starting at the 2000^{th} epoch and ends at the 4000^{th} epoch. We do not use any weight decay. We also learn the integration time t_1 during training by back-propogation [2].

C. Additional comparisons

In this section, we compare our model to more baselines to show the effectiveness of the model design. The first baseline is Neural Statistician (NS) [3], a state-of-the-art generative model for sets. We modify its official code for

*Equal contribution.

Table 1: Ablation studies. ↑: the higher the better, ↓: the lower the better. The best scores are highlighted in bold. MMD-CD scores are multiplied by 10^3; MMD-EMD scores are multiplied by 10^2; JSDs are multiplied by 10^2.

Category	Model	# Parameters (M)		JSD (↓)	MMD (↓)		COV (%, ↑)		1-NNA (%, ↓)	
		Full	Gen		CD	EMD	CD	EMD	CD	EMD
Airplane	NS [3]	2.29	1.00	**1.74**	0.655	4.51	7.81	4.51	99.61	99.61
	VAECNF	**1.47**	**0.92**	6.30	0.261	3.35	41.98	46.17	88.64	82.72
	WGAN-CNF	1.75	1.06	4.29	0.254	**3.23**	42.47	**48.40**	75.80	75.68
	PointFlow (ours)	1.61	1.06	4.92	**0.217**	3.24	**46.91**	**48.40**	**75.68**	**75.06**
	Training set	-	-	6.61	0.226	3.08	42.72	49.14	70.62	67.53

generating 2D spatial coordinates of MNIST digits to make it work with 3D point cloud coordinates. We use the same encoder architecture as our model, and use the VAE decoder provided by authors with the input dimension changed from 2 to 3. It differs from our model mainly in 1) using VAEs instead of CNFs to model the reconstruction likelihood, and 2) using a simple Gaussian prior instead of a flow-based one. The second baseline is VAECNF, where we use the CNF to model the reconstruction likelihood but not prior. Specifically, the VAECNF optimizes ELBO in the following form:

$$\mathcal{L}(X;\phi,\theta) = \sum_{x\in X}\left(\log P(G_\theta^{-1}(x;z)) - \int_{t_0}^{t_1}\mathrm{Tr}\left(\frac{\partial g_\theta}{\partial y(t)}\right)dt\right) + D_{KL}(Q_\phi(z|X)||P(z))\,, \tag{5}$$

where $P(z)$ is a standard Gaussian $\mathcal{N}(0,I)$ and D_{KL} is the KL-divergence. As another baseline, we follow l-GAN [1] to train a WGAN [5] in the latent space of our pretrained auto-encoder. Both the discriminator and the generator are MLP with batch normalization between layers. The generator has three layers with hidden dimensions 256. The discriminator has three layers with hidden dimensions 512.

The results are presented in Table 1. Neural Statistician [3] is able to learn the marginal point distribution but fails to learn the correct shape distribution, as it obtains the best marginal JSD but very poor scores according to metrics that measure similarities between shape distributions. Also, using a flexible prior parameterized by a CNF (PointFlow) is better than using a simple Gaussian prior (VAECNF) or a prior learned with a latent GAN (WGAN-CNF) that requires two-stage training.

D. Limitation and failure cases

In this section, we discuss the limitation of our model and present visualizations of difficult cases where our model fails. As mentioned in FFJORD [4], each integration requires evaluating the neural networks modeling the dynamics multiple times. The number of function evaluations tends to increase as the training proceeds since the dynamic becomes more complex and more function evaluations are needed to achieve the same numerical precision. This issue limits our model size and makes the convergence slow. Grathwohl *et al*. indicate that using regularization such as weight decay could alleviate such an issue, but we empirically find that using regularization tends to hurt performance. Future advances in invertible models like CNF might help improve this issue. Typical failure case appears when reconstructing or generating the rare shape or shapes with many thin structures as presented in Figure 1.

Algorithm 1 PointFlow training.

Require: Point cloud encoder Q_ϕ; CNFs G_θ and F_ψ, whose dynamics are defined by g_θ and f_ψ, respectively; Integration time interval $[t_0, t_1]$; Learning rate α; Total number of training iterations T; Data samples X_t.

for $t = 1, 2, \ldots, T$ do **do**
  $\mu, \sigma \leftarrow Q_\phi(X_t)$ {d is the dimension of μ}
  $\mathcal{L}_{ent} = \frac{d}{2}(1 + \ln(2\pi)) + \sum_{i=1}^{d}\ln\sigma_i$
  $z \leftarrow \epsilon \odot \sigma + \mu$ {Reparameterization.}
  $w \leftarrow F_\psi^{-1}(z)$
  $\mathcal{L}_{prior} = \log\mathcal{N}(w;0,I) - \int_{t_0}^{t_1}\mathrm{Tr}\left(\frac{\partial f_\psi(w(t))}{\partial w(t)}\right)dt$
  $L \leftarrow 0$
  for $x_i \in X_t$ do **do**
    $y_i \leftarrow G_\theta^{-1}(x_i;z)$
    $L_i \leftarrow \log\mathcal{N}(y_i;0,I) - \int_{t_0}^{t_1}\mathrm{Tr}\left(\frac{\partial g_\theta(y_i(t))}{\partial y_i(t)}\right)dt$
    $L \leftarrow L + L_i$
  end for
  $\mathcal{L}_{recon} = \frac{L}{|X_t|}$
  $\mathcal{L} = \mathcal{L}_{recon} + \mathcal{L}_{prior} + \mathcal{L}_{ent}$
  $\phi, \psi, \theta \leftarrow Adam(\mathcal{L}, \phi, \psi, \theta)$
end for
return Q_ϕ, G_θ, F_ψ

E. Latent space visualizations

We provide visualization of the sampled latent vectors $z \in \mathbb{R}^{128}$ in Figure 2. We sample 1000 latent vectors and run t-SNE [7] to visualize these latent vectors in 2D. Shapes with similar styles are close in the latent space.

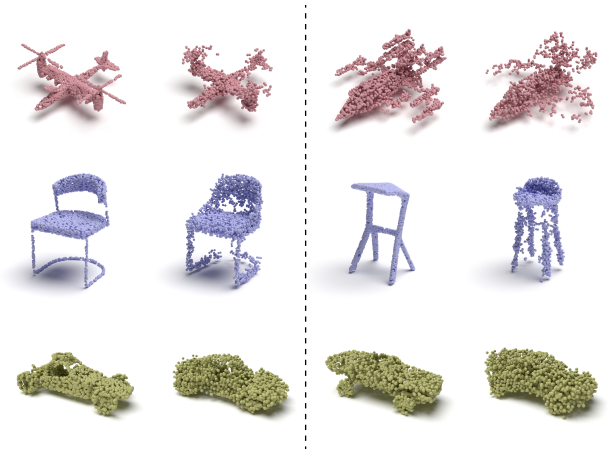

Figure 1: Difficult cases for our model. Rare shapes or shapes that contain many thin structures are usually hard to reconstruct in high quality.

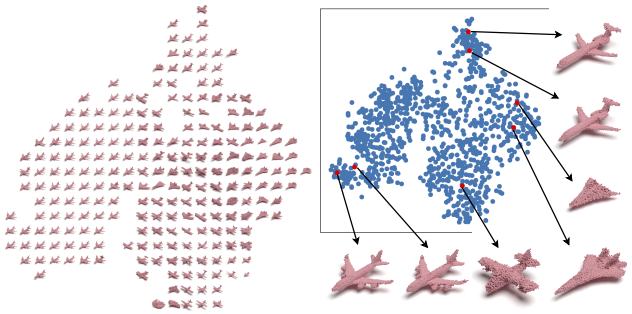

Figure 2: Visualization of latent space.

F. Interpolation

In this section, we present interpolation between two different shapes using our model. For two shapes X_1 and X_2, we first compute the mean of the posterior distribution using $Q_\theta(z|X)$. Let μ_1 and μ_2 be the means of the posterior distribution for X_1 and X_2 respectively. We use μ_1 and μ_2 as the latent representation for these two shapes. We then use the inverse prior flow F_ψ^{-1} to transform μ_1 and μ_2 back to the prior space. Let $w_1 = F_\psi^{-1}(\mu_1)$ and $w_2 = F_\psi^{-1}(\mu_2)$ be the corresponding vectors for μ_1 and μ_2 in the prior space. We use spherical interpolation between w_1 and w_2 to retrieve a series of vectors w_i. For each w_i, we use the CNF prior F_ψ and the CNF decoder G_θ to generate the corresponding shape X_i. Figure 3 contains examples of the interpolation.

G. More flow transformation

Figure 4 presents more examples of flow transformations from the Gaussian prior to different shapes.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018. 1, 2

[2] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018. 1

[3] Harrison A Edwards and Amos J. Storkey. Towards a neural statistician. In *ICLR*, 2017. 1, 2

[4] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *ICLR*, 2019. 1, 2

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017. 2

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 1

[7] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 2

[8] 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

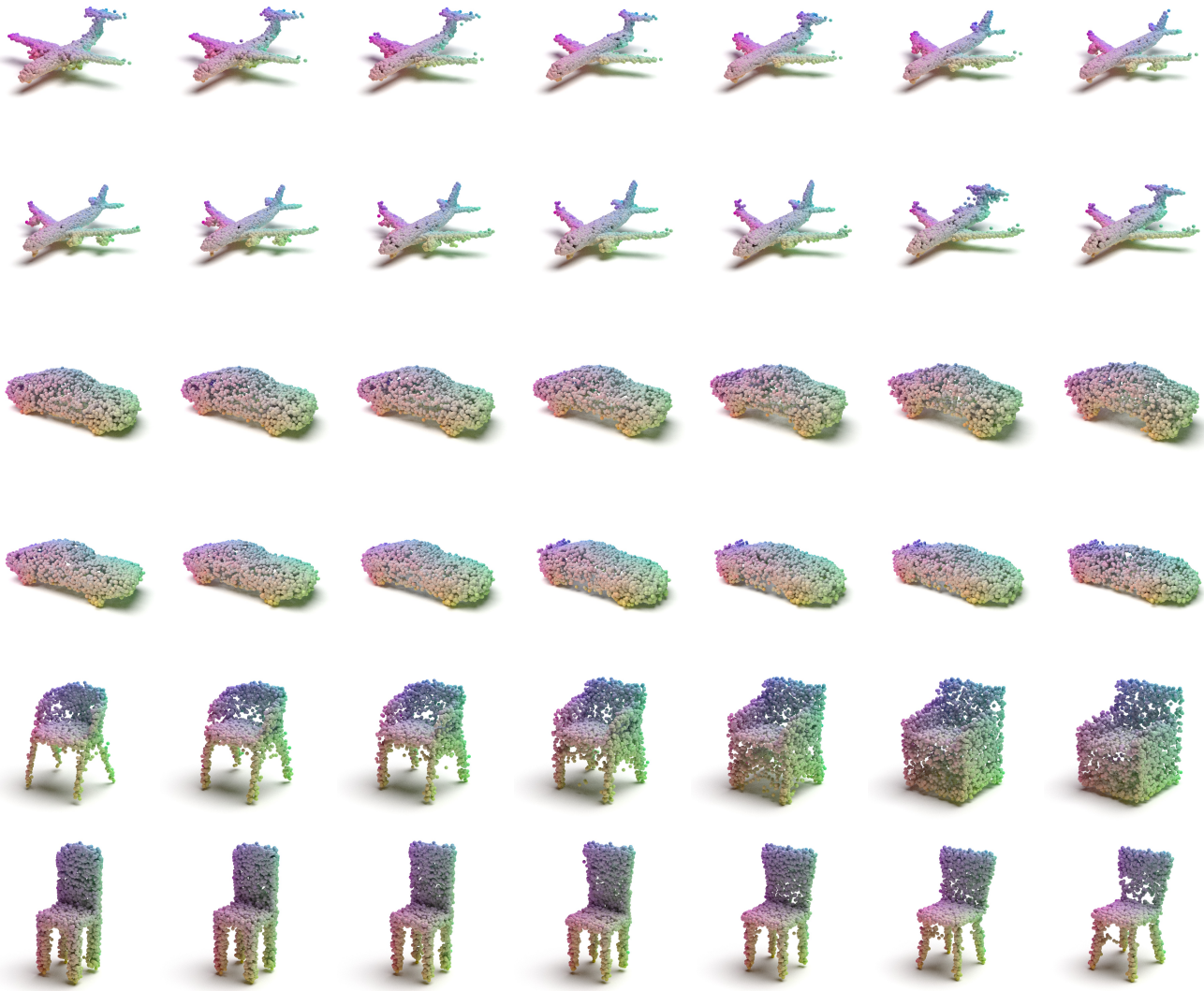

Figure 3: Feature space interpolation. The left-most and the right-most shapes are sampled from scratch. The shapes in between are generated by interpolating the two shapes in the prior space.

Figure 4: Additional visualizations on the process of transforming prior to point cloud.