Supplementary Material for Learning Neural Representation of Camera Pose with Matrix Representation of Pose Shift via View Synthesis

1. Training details

In this section, we describe the details about the structure of our neural networks and the hyperparameters we use in the experiments. The main differences among the network structures we use on different datasets depend on: (i) the size of the image we are dealing with: the larger image needs more blocks; (ii) the complexity of the scenes. For 7Scenes and Gibson rooms dataset, the scenes are highly complex. Therefore we apply instance normalization to multiple layers, which is dependent on scenes, besides the vector representation of the scene at the top layer. For the GQN rooms dataset, which includes a huge amount of scenes, we employ an encoder to calculate the scene representations online. We use Adam[5] as optimizer for all the experiments with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate for each setting is introduced in each later section.

1.1. GQN rooms dataset

Generative experiment. Since this dataset contains a huge amount of scenes, and each scene only has few images, we encode the scene representations online instead of learning an individual vector representation for each scene. The encoder structure is shown in Figure 4a. Specifically, the encoder encodes the scenes as a scene vector that is fed to the top layer of the decoder, and it also encodes the parameters of instance normalization [3] that is applied to the multiple layers of the generator. Following [2], to summarize information across multiple images of the same scene, we sum up the encoded vectors and parameters of these images. The decoder structure is shown in Figure 4b. We discretize the square space into 20×20 grids and learn a position vector at each grid. Similarly, we discretize the orientation into 36 grids (10° per grid) and learn an orientation vector at each grid. The training takes about four days on a single Titan RTX GPU.

We train the model for one million iterations. At each iteration, we randomly sample 30 scenes, each containing ten images. We use the first six images of each scene to encode the scene representation and concatenate it with the other three images' pose representations. We use the concatenated representations to reconstruct the three images. We leave the last image for testing. For the rotation loss, we randomly sample 4000 pairs of poses for each iteration. The learning rate of the pose representations and matrix representations of camera movements is 0.01, and the learning rate for the encoder, decoder, and scene representations is 0.0001. Here, we update all the learnable parameters together. We set λ_1 as 0.05, λ_2, λ_3 as 100 and λ_4 as 0.8.

For the baseline GQN network, we also train the model for one million steps. At each step, we feed in a batch of 64 scenes. The other parameters follow the original implementation.

Inference experiment. We show the inference model structure in Figure 4c. Like the generative experiment, we use an encoder to encode the scene and the parameters of instance normalization online. The encoder structure is the same as the encoder used in the generation task, except that we do not encode a vector representation at the top layer but encode another set of instance norm parameters $(\gamma 4, \beta 4)$. We set the learning rate as 0.0001 for all the parameters. We train the inference model for 100,000 steps. At each iteration, we feed in 30 scenes. For this dataset, we use the homoscedastic uncertainty method proposed in [4] to automatically tune the weight between pose prediction loss of position and orientation. We set the initial guess for logarithmic weight of position loss as $S_{\text{pos}} = -\log 20$ and the initial guess for logarithmic weight of orientation loss as $S_{\text{ori}} = -\log 5$ (so that $\exp(-S_{\text{pos}}) = 20$ and $\exp(-S_{\text{ori}}) = 5$). We use the same inference model structure for baseline models, except that we add another fully-connected (FC) layer with size 196 to these models to make sure that they have approximately the same amount of parameters as the model trained on our representations. We also train these models for 100,000 iterations with the same batch size. We tune the learning rate for each baseline model to make a fair comparison and use the same automatically weight tuning method for the two quaternions-related baselines. The initial guess for the logarithm weight of position loss is set to 0.0, and the one of orientation loss is set to -3.0 as suggested by [4]. For our model and each of the baseline models, the training takes about 5 hours on a single

Titan RTX GPU.

1.2. ShapeNet car

Generative experiment. This dataset contains 2151 different cars. The heads of the cars are aligned to the same orientation, and the background is blank. Given the simplicity of this dataset, we do not use instance normalization. The vector representation of scenes is of 128 dimensions, and we learn a separate vector representation for each scene instead of obtaining by an encoder. The structure of the generator model is shown in Figure 5a. For our pose representation system, we discretize the orientation for 0° to 360° into 36 grids and learn individual orientation vectors at each grid.

For each scene, we randomly sample 50 pairs of images for each scene as the training set and leave the others as the test set. The camera poses of the two images in each pair is close to each other, so that the change from one to another can be approximated by Taylor expansion of the matrix Lie groups as discussed in section 3.3, which means that we can apply the camera poses of the two images to the rotation loss. We train our model for 160,000 iterations, i.e., 1500 epochs. We randomly sample 20 scenes at each iteration, and for each instance, we sample 10 images (5 pairs). For the rotation loss, we randomly sample additional 200 pairs of camera poses to compute the loss. The learning rate is set to 0.0001. We set λ_1 as 0.05 and λ_2 as 50. We iteratively update the decoder for one time and pose representation system for three times at each iteration. The training takes about four days on a single Titan RTX GPU.

For the baseline GQN model, we trained the model for 500,000 steps. At each step, we randomly sample a batch of 36 scenes. We randomly sample 15 images for each scene to infer the scene representation and another image as the reconstruction target. We use the same train-test split as our model for each scene here.

Inference experiment Since the head direction for each car is aligned to the same direction, the pose regression task should follow the same rule across different scenes. Thus, we do not include scene-related parameters in our inference model. The structure of our inference model is shown in Figure 5b. For each scene, we randomly sample 250 images as the training set and the rest 250 images as the test set. We train our model and all the baseline models for 500 epochs. At each iteration, we use 10 scenes, and we randomly sample 20 images from each scene. The learning rate is set to 0.001. We simply set the weights of prediction losses of the two rotation vectors as 1.0 without further automatic tuning. For each baseline representation, we use the same inference model structure and add another fully-connected (FC) layer with size 256. We tune the learning rate carefully to make a fair comparison, and we use the automatic weight tuning method for the two quaternions-related baseline methods. The initial guess for the logarithmic weight of orientation loss is set to -3.0 as suggested by [4]. The training for our model and each of the baseline models takes about 8 hours on a single Titan RTX GPU.

1.3. Gibson rooms dataset

Generative experiment. This dataset contains complex scenes. We apply instance normalization at multiple layers, which is dependent on the scene. The structure is shown in Figure 6a. The scene vector representation is of 768 dimensions, and the dimensions of instance normalizations are summarized in Figure 6a. We discretize the 2m $\times$ 2m square space into 40 $\times$ 40 grids. We discretize the two orientation angles into grids so that each grid is 10°.

For each scene, we randomly sample half of the data as the training set and the rest as the test set. We train our model for 500k steps. At each iteration, we randomly choose four scenes. For each scene, we randomly sample 50 images. For the rotation loss, we randomly sample another 3000 pairs of poses. We use a learning rate of 0.0001 for training the generator and a learning rate of 0.01 for the pose representation. We iteratively update the generator parameters for one time and update the pose representation two times at each iteration. We set λ_1 as 0.01, λ_2, λ_3 as 100 and λ_4 as 0.8. The training takes about five days on a single Titan RTX GPU.

For the baseline GQN model, we train the model for 500k steps. At each iteration, we randomly sample and predict 36 images. To predict each image, we randomly pick 15 images from the same scene to infer the scene representations.

Inference experiment. The inference structure is shown in Figure 6b. We trained the inference model for 25000 steps for both our representation and the baseline representations. At each step, we randomly sample 4 scenes with 50 images from each scene. The learning rate for the model with our representation is 0.001. For this dataset, we find that simply set the weight of position prediction loss as 20 and set the weight of orientation prediction loss as 10 is good enough. So we do not employ the automatic weight tuning mechanism here. We tuned the learning rate for each baseline model, and we applied the homoscedastic uncertainty method to tune the weight for the quaternions-related representations automatically. The initial guess for the logarithm weight of position loss is set to 0.0, and the one of orientation loss is set to -3.0. For each baseline representation, we use the same inference model structure and add another fully-connected (FC) layer with size 192.The initial guess follows [4]. For our model and each of the baseline models, the training takes about 5.5 hours on a single Titan RTX GPU.

1.4. 7Scenes

Generative experiment. For this dataset, we use the same generator structure as for the Gibson Room dataset (see Figure 6a). Since this dataset contains less data than the Gibson Room dataset, we set the dimension of the scene vector representation to 96. We discretize the whole region (4m $\times$ 1.5m $\times$ 3m) into grids so that each grid is of 0.1m $\times$ 0.1m $\times$ 0.1m. The orientation is discretized into grids so that each grid is of 10°.

We update the model for 100,000 steps. At each step, we randomly sample 16 pairs of images from each scene, and we randomly sample 3000 extra pairs of poses to estimate the rotation loss. We use the learning rate 0.0001 for the generator and 0.001 for the pose representation system. We iteratively update the generator for one time and pose representation system for two times at each iteration. We set λ_1 as 0.009 and λ_2 as 50. The training takes about one day on a single Titan RTX GPU.

Inference experiment. For the inference model, we use the same structure proposed in [1], *i.e.*, we use a pre-trained ResNet34 as the basic feature extractor. We learn a separate module containing several FC layers on the top of the extracted features to predict each pose vectors. Following [1], we train an individual inference model for each scene. We use learning rate 0.00005 and train the model of each scene for 60 epochs. To isolate the effect of different representations, we use PoseNet as the model for all the representations, without other techniques such as adding pair losses or unlabeled data. We consider these techniques to be orthogonal to the improvement in pose representation. We employ the automatic weight tuning method as [1] to tune the weight between the three position vectors and three orientation vectors. We set the initial guess for the logarithm weight of three position vectors' losses the initial guess for logarithm weight of three orientation vectors' losses as -3.0. We employ 0.7 color jitter as data augmentation and remove the dropout in the final FC layer. Our model takes about 3.7 hours for training all the 7 scenes on a single Titan RTX GPU. For the baseline model, we use the released code of [1] and we use the default setting with python 3.6 and torch 1.2.0, which trains the models on each scene for 300 epochs with a learning rate of 0.0001. It takes about 13 hours to train the baseline models on the entire 7 scenes on a single Titan RTX GPU.

2. Additional training results

2.1. Generative results

We show more novel view synthesis results for GQN rooms, ShapeNet car and Gibson rooms in Figures 7, 8, 9.

2.2. Reconstructed image under different noise magnitude

In Figure 1, we show the reconstructed images at different noise levels using our model with learned camera pose representation and GQN (which uses predefined low dimensional sinusoidal function to represent rotation). We can see that our model can reconstruct image with correct pose even with high noise while the poses in the reconstructed images of GQN model change a lot as noise increases. This agrees with our observations from the psnr curves and further prove that our learned camera pose representation is more robust to noise.

2.3. Learning the camera pose representation by a fully connected neural network

As a comparison, we replace our proposed camera pose representation by a fully connected neural network on ShapeNet car dataset. Specifically, we encode each angle by a 2-layer fully-connected neural network. The first layer has a length of 128 the second layer has a length of 96 (which is same to our embedding). We use leaky relu as the activation function. As shown in Figure 2, this embedding is also a high-dimensional one but it doesn't has the translation invariance [6] as in our learned representation. Figure 3 shows the PSNR over the magnitude of noise added to representations. The representation using a fully connected neural network works better than the plain low dimension embedding used in GQN in terms of robustness to noise. But it still performs worse than our design, which is regulated by the rotation loss. As for the camera pose estimation, using the representation from a fully connected neural network gives a testing error of 3.63°, which is lower than the results of all the other hand designed representations but still higher than the result of our design (with testing error 2.85°). The results show that learning a high-dimenstional representation is better than the low-dimensional hand designed ones and enforcing the translation invariance using rotation loss can further improve the results.

References

[1] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Geometry-aware learning of maps for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2018. 3, 7

[2] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 1

[3] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceed-*

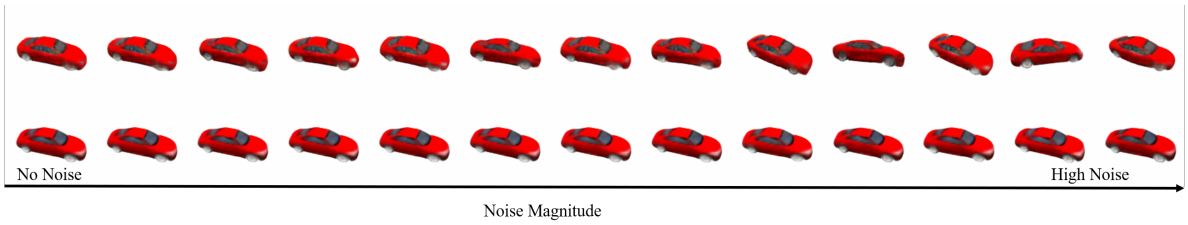


Figure 1: Reconstructed image at different noise levels. Top: Results from GQN (which uses low dimensional $\sin(\alpha)$, $\cos(\alpha)$ to represent angles); Bottom: Our results with learned camera pose representation. From left to right: Gradually increasing the noise added to the camera pose representation.

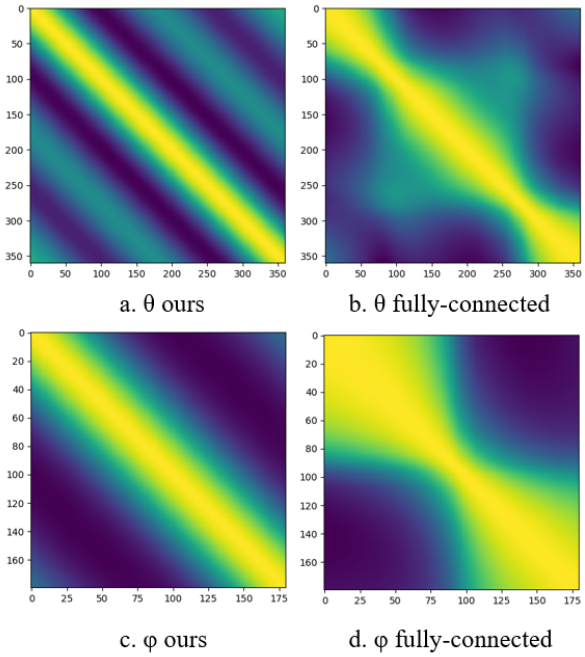


Figure 2: Inner product of the learned camera pose representation. Each figure shows the inner product matrix between the camera pose representation at different positions (check [6] for more details). *Left*: our camera pose representation; *Right*: Camera pose representation from a fully connected neural network. *Top*: Embedding for angle θ; *Bottom*: Embedding for angle ϕ.

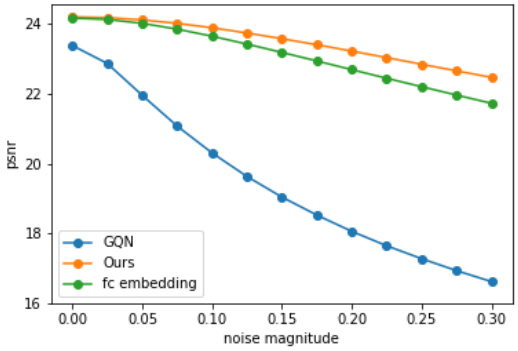


Figure 3: PSNR over magnitude of noise added to representations, learned from ShapeNet car dataset.

ings of the IEEE International Conference on Computer Vision, pages 1501–1510, 2017. 1

[4] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5974–5983, 2017. 1, 2

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[6] Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, and Jakob Grue Simonsen. On position embeddings in bert. In *International Conference on Learning Representations*, 2021. 3, 4

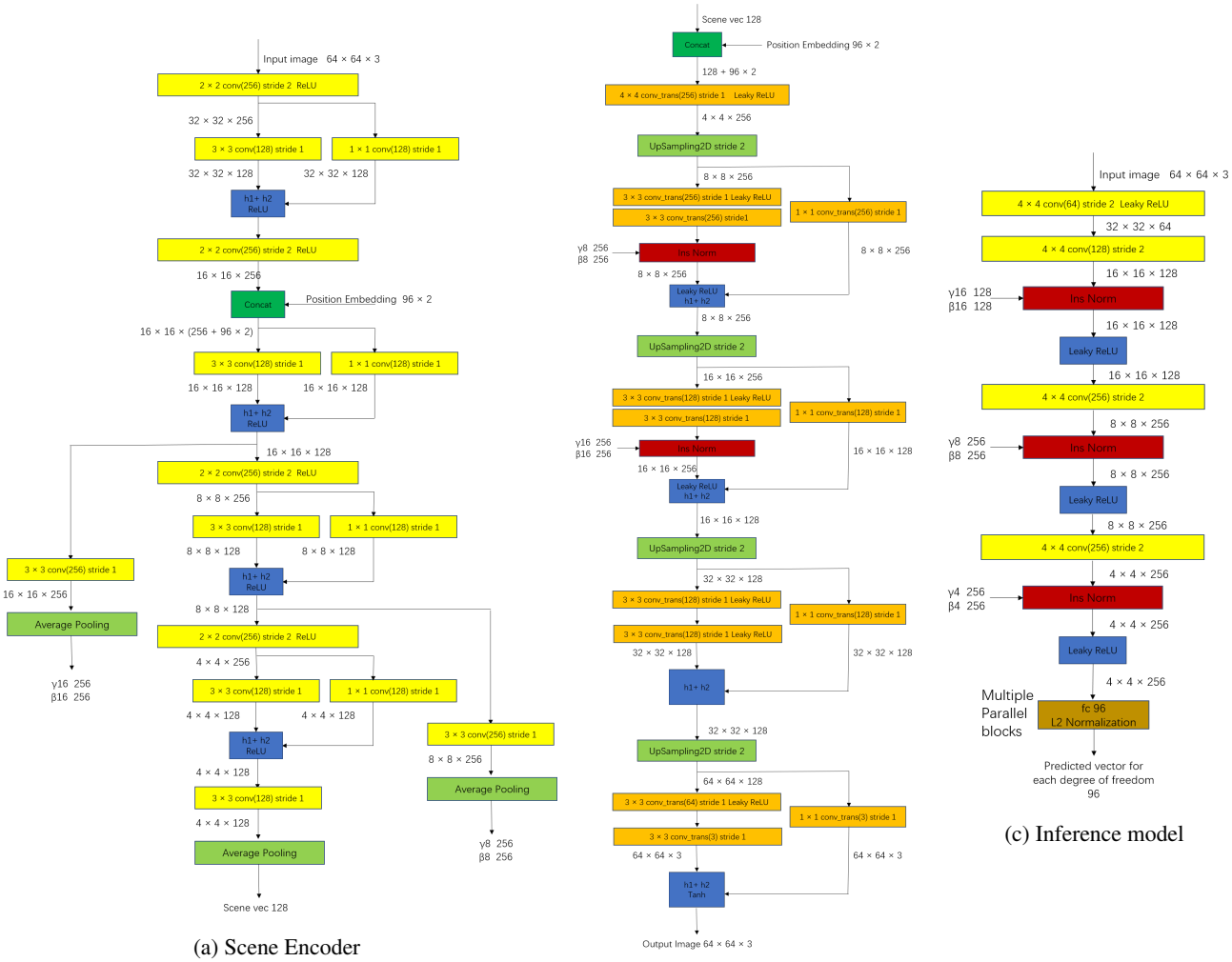
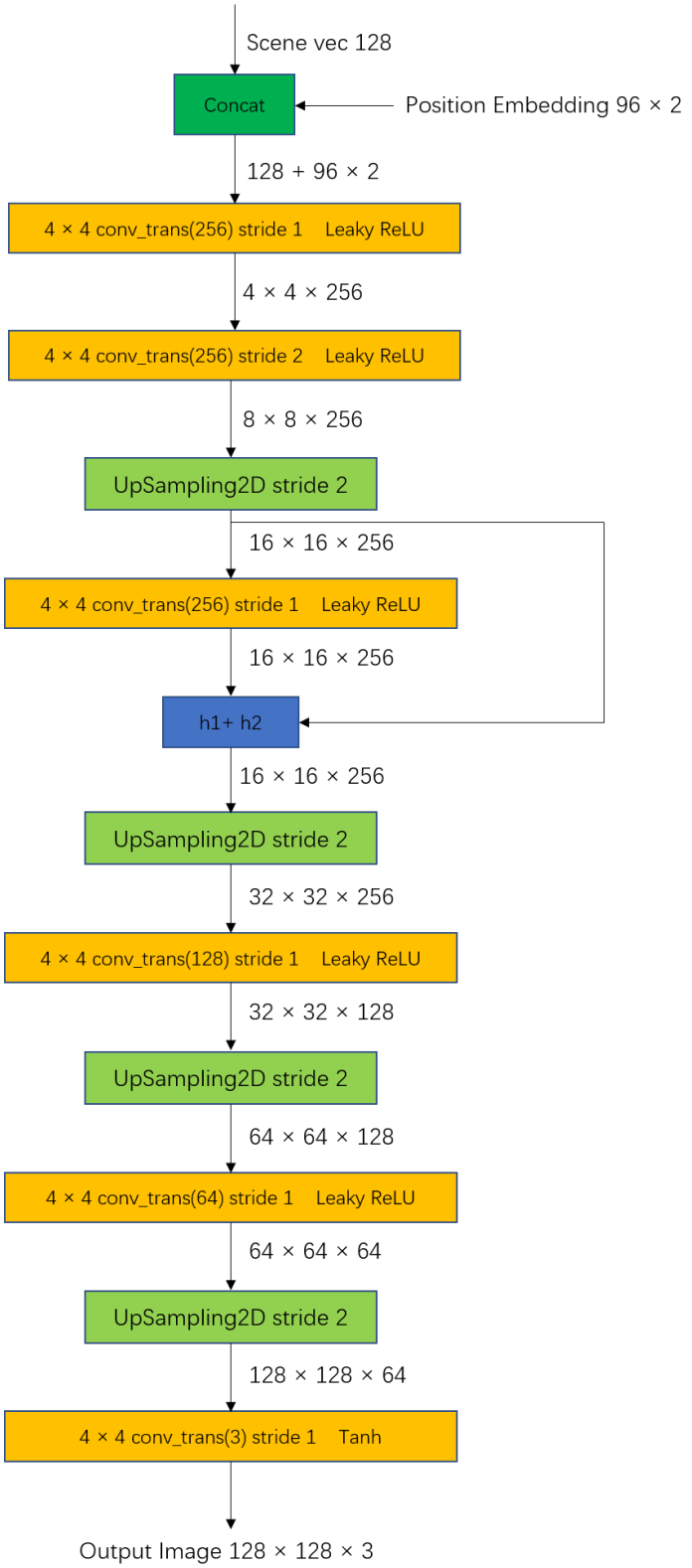
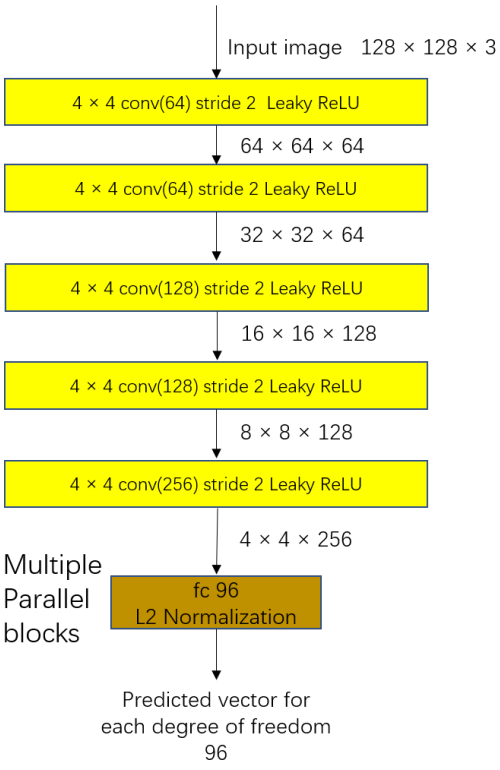


Figure 4: Network structures for GQN rooms dataset. γ and β denote the parameters of instance normalization.

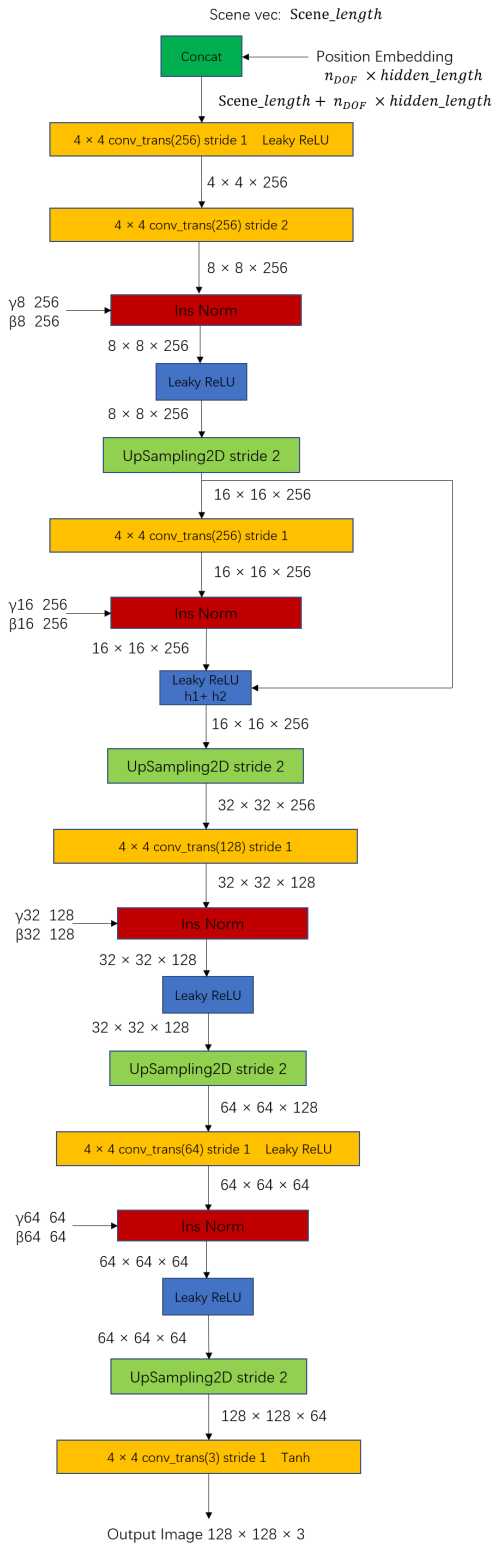


(a) Decoder

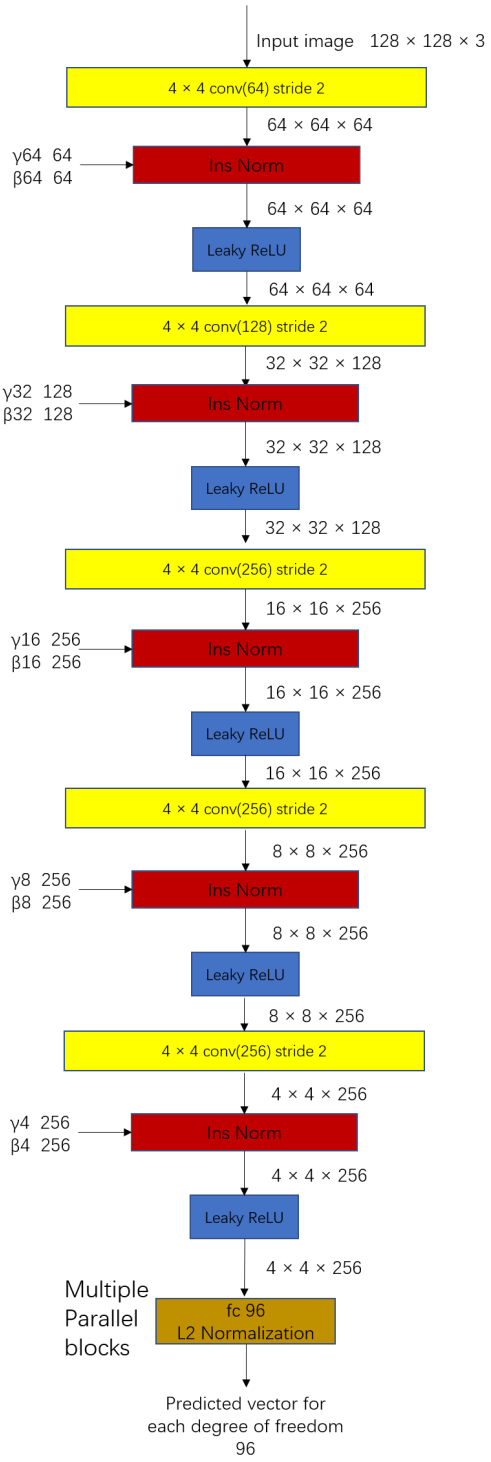


(b) Inference

Figure 5: Network structures for ShapeNet car dataset.

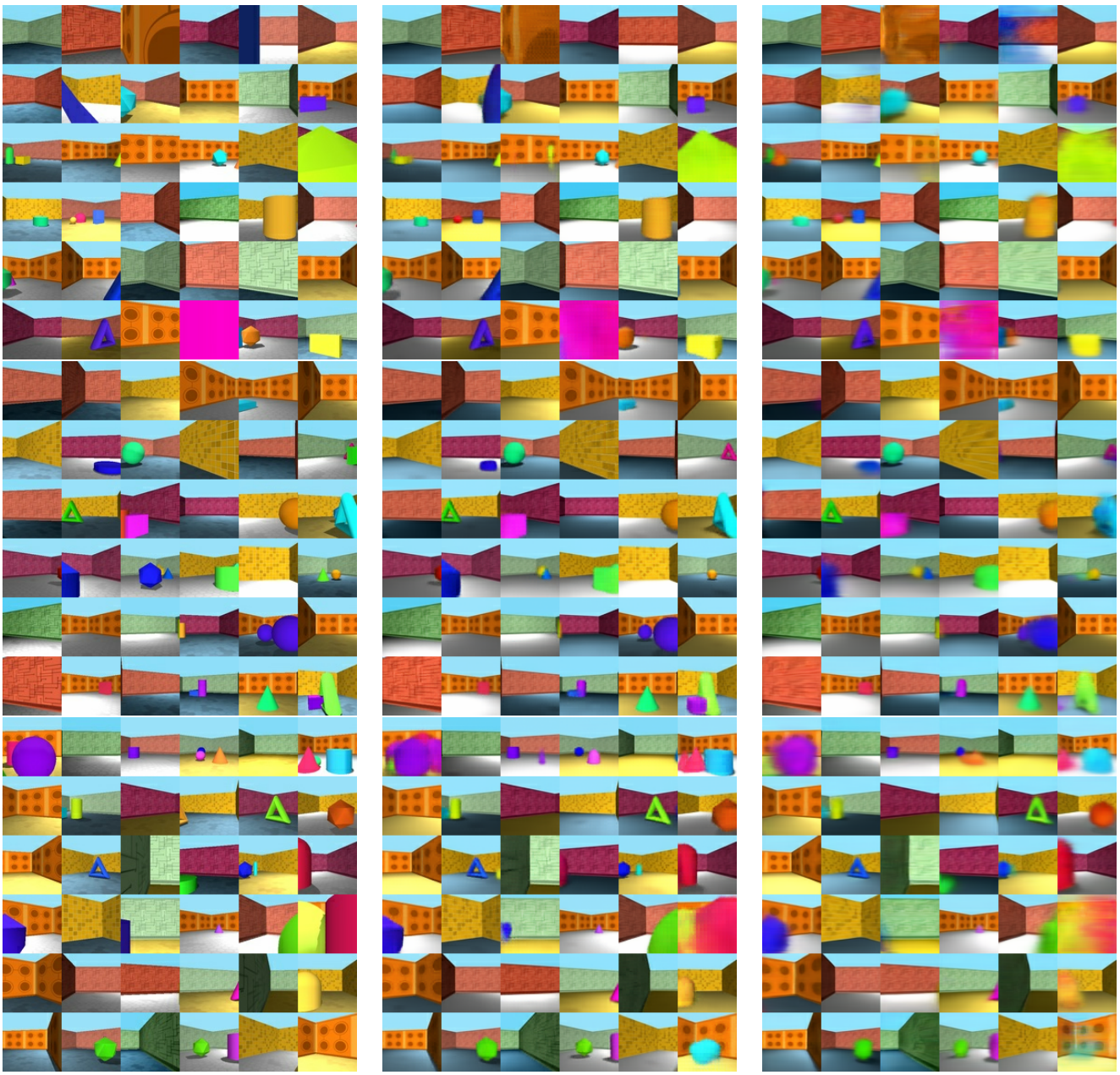


(a) Decoder



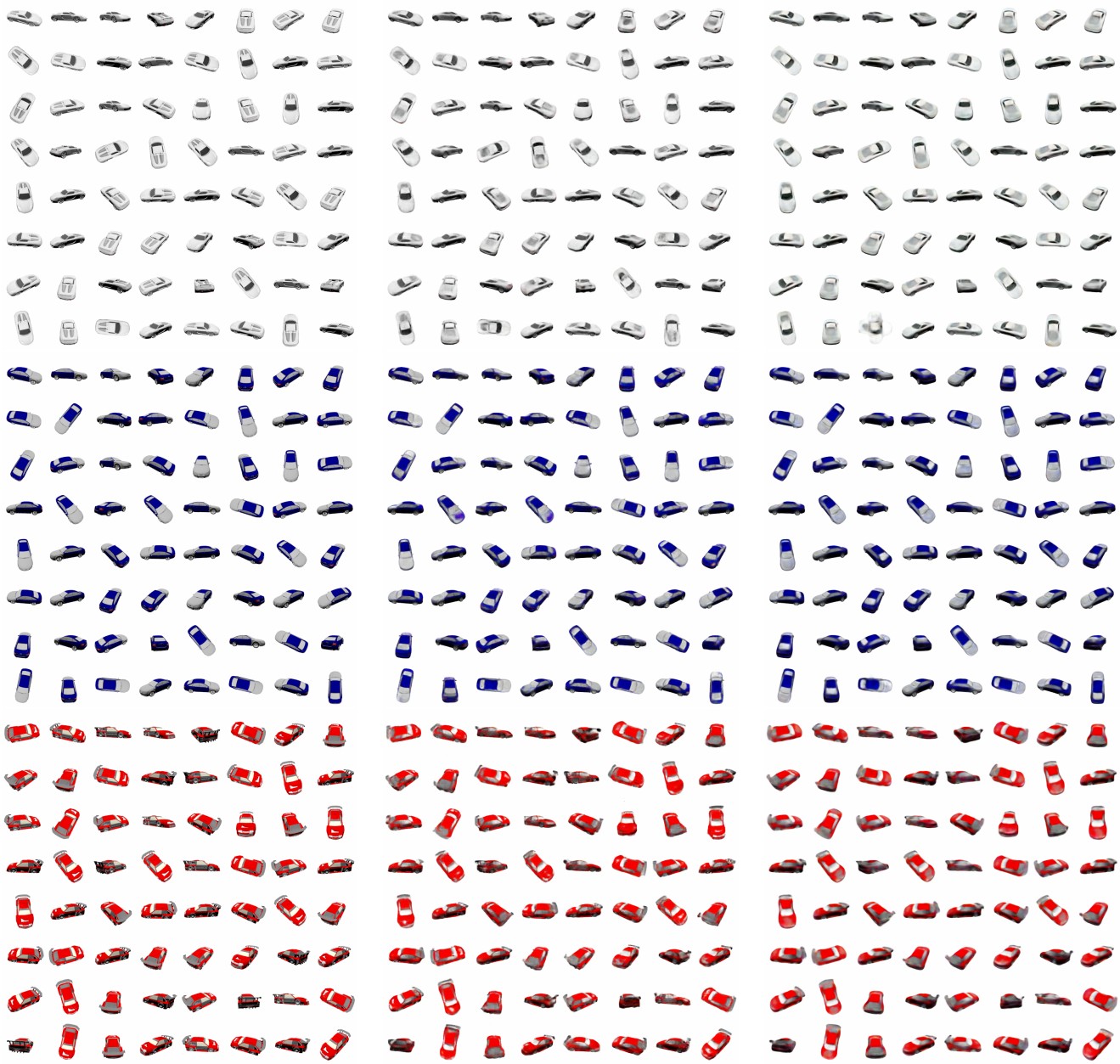
(b) Inference

Figure 6: Network structures for Gibson rooms dataset. γ and β denote the parameters of instance normalization. The 7Scenes dataset shares the same decoder structure with Gibson room dataset while its inference model is the same as [1].



(a) Ground Truth (b) GQN (c) Ours

Figure 7: Additional novel view synthesis results on GQN rooms dataset.



(a) Ground Truth (b) GQN (c) Ours

Figure 8: Additional novel view synthesis results on ShapeNet v2 car dataset.

(a) Ground Truth (b) GQN (c) Ours

Figure 9: Additional novel view synthesis results on Gibson rooms dataset.