NeeDrop: Self-supervised Shape Representation from Sparse Point Clouds using Needle Dropping — Supplementary Material

Alexandre Boulch[1] Pierre-Alain Langlois[2] Gilles Puy[1] Renaud Marlet[1,2]

[1]Valeo.ai, Paris, France [2]LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France

We provide here complementary information about the paper "NeeDrop: Self-supervised Shape Representation from Sparse Point Clouds using Needle Dropping":

A. We present the network architecture used in our experiments, as well as the training details.

B. We provide a complete derivation of the loss expression defined in the main paper.

C. We analyze the latent space of shape representations through shape generation, shape interpolation, latent space visualization and classification.

D. We study the validity of the needle dropping construction hypothesis.

E. We analyze the impact of two point-sampling strategies: with or without resampling.

G. We compare the quality of our reconstructions with ShapeGF, that generates point clouds rather than meshes.

F. We evaluate the robustness of our trained models to input noise and to variations of numbers of input points.

H. We present additional results on the KITTI dataset, including reconstruction with networks trained on ShapeNet to test the robustness to domain gaps.

I. We give pseudo-code for the loss defined in the paper.

Note: reference numbers for citations used here are not the same as references used in the main paper; they correspond to the bibliography section at the end of this supplementary material.

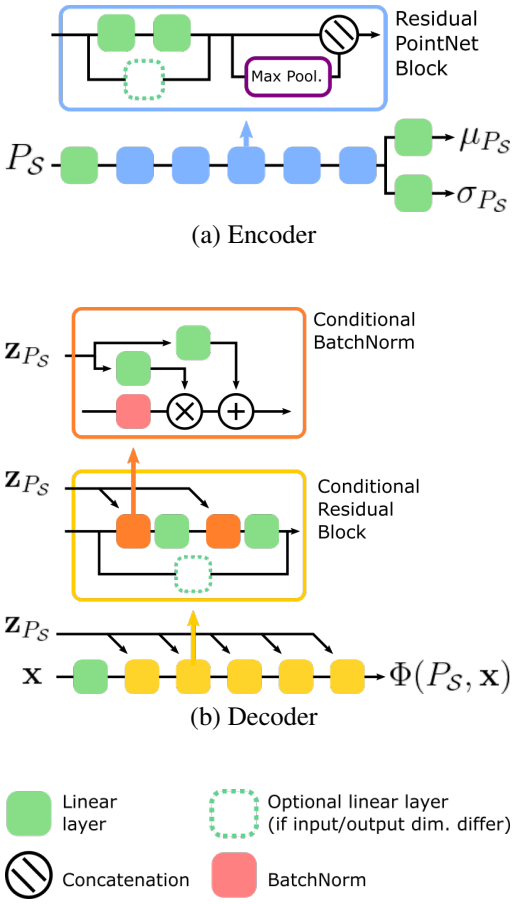


Figure 5. Network used in NeeDrop, similar to [13]

A. Network

In this work, we build on the network from Occupancy Networks [13], which has an encoder-decoder architecture as presented in Figure 5.

Encoder. The encoder, illustrated in Fig. 5(a), is a Residual PointNet [15] composed of 5 residual PointNet blocks, each containing two linear layers and a skip connection.

The encoder is applied point-wise. To gather global information at the output of each residual block, we use a global max-pooling layer (over all points of the point cloud) and concatenate the local feature vector (at point level) and the global feature vector (at point-cloud level) before entering the next block.

As mentioned in Section 3.3.1 of the main paper, our model is trained as a variational auto-encoder. Hence, our encoder outputs two vectors μ_{P_S} and σ_{P_S}, that encode respectively the mean and standard deviation of the normal distribution used for generating the latent code $\mathbf{z}_{P_S}$ at train-

ing time. At test time, we use $\mathbf{z}_{P_S} = \mu_{P_S}$ to be deterministic with respect to the input point cloud.

Decoder. The decoder, illustrated in Figure 5(b), is a conditional multi-layer perceptron (MLP) [6,8]. It is composed of five conditional residual block. Each block is conditioned at batch normalization by the latent code $\mathbf{z}_{P_S}$. The query point, $\mathbf{x} \in \mathbb{R}^3$, is the location at which the occupancy is estimated. Note that the occupancy is estimated with $S(\phi(P_S, \mathbf{x}))$, where S is the sigmoid function, not represented in Figure 5(b).

Parameters. In all our experiments, the latent vector size representing a shape is set to 256. The hidden size of the encoder is also 256, and the latent size of the decoder is set to 512. The batch-norm layer is the usual 1-dimensional batch normalization.

Training procedure. The encoder and decoder are trained end-to-end. In all experiments, we use Adam [12] as optimizer with an initial learning rate set to 10^{-3}. To train with mini-batches, we generate the same number of opposite-side needles (300, the same as the input point cloud size) and the same number of same-side needles (2048, to cover the space around the shape) for each model. The batch size is set to 32.

In practice, we observed a low dependency of the performance to the number of needles (e.g., doubling the size of Q_{opp} and Q_{same}), so we chose these parameters values to permit us to train the model with a middle range GPU NVidia 2080Ti.

Computational cost. Our only additional cost w.r.t. occupancy-supervised methods is the computation, at training time, of nearest neighbors, which is little in our setting. Distance-based methods have a similar "extra" cost as ours.

B. Loss derivation

In the main paper, the final loss and partial derivative expressions, i.e., Eq. (13) and Eq. (14), are given without a complete derivation. We derive here these expressions.

B.1. Training loss

The binary cross entropy reads as

$$\begin{aligned}\text{BCE}(b_{\mathbf{x},\mathbf{y}}, b^0_{\mathbf{x},\mathbf{y}}) = \quad & b^0_{\mathbf{x},\mathbf{y}} \log(b_{\mathbf{x},\mathbf{y}}) \\ & (1 - b^0_{\mathbf{x},\mathbf{y}}) \log(1 - b_{\mathbf{x},\mathbf{y}}),\end{aligned} \tag{15}$$

where, recalling Eq. (4) of the main paper,

$$b_{\mathbf{x},\mathbf{y}} = b_{\mathbf{x}}\, b_{\mathbf{y}} + (1 - b_{\mathbf{x}})(1 - b_{\mathbf{y}}). \tag{16}$$

Subsbituting the expression of $b_{\mathbf{x},\mathbf{y}}$ in Eq. (15) yields

$$\begin{aligned}\text{BCE}(b_{\mathbf{x},\mathbf{y}}, b^0_{\mathbf{x},\mathbf{y}}) = \\ & b^0_{\mathbf{x},\mathbf{y}} \log(b_{\mathbf{x}} b_{\mathbf{y}} + (1 - b_{\mathbf{x}})(1 - b_{\mathbf{y}})) \\ & (1 - b^0_{\mathbf{x},\mathbf{y}}) \log(1 - b_{\mathbf{x}} b_{\mathbf{y}} - (1 - b_{\mathbf{x}})(1 - b_{\mathbf{y}})).\end{aligned} \tag{17}$$

Then, using the fact that $b_{\mathbf{x}} = S(P_S, \mathbf{x})$ and $b_{\mathbf{y}} = S(P_S, \mathbf{y})$, where S is the sigmoid function, we obtain:

$$\begin{aligned}\text{BCE}(b_{\mathbf{x},\mathbf{y}}, b^0_{\mathbf{x},\mathbf{y}}) = \\ & b^0_{\mathbf{x},\mathbf{y}} \log\left(\frac{e^{(P_S,\mathbf{x})} e^{(P_S,\mathbf{y})} + 1}{(e^{(P_S,\mathbf{x})} + 1)(e^{(P_S,\mathbf{y})} + 1)}\right) \\ & (1 - b^0_{\mathbf{x},\mathbf{y}}) \log\left(\frac{e^{(P_S,\mathbf{x})} + e^{(P_S,\mathbf{y})}}{(e^{(P_S,\mathbf{x})} + 1)(e^{(P_S,\mathbf{y})} + 1)}\right).\end{aligned} \tag{18}$$

Finally, we can simplify the expression by noticing that the denominators in the log terms are identical. It yields:

$$\begin{aligned}\text{BCE}(b_{\mathbf{x},\mathbf{y}}, b^0_{\mathbf{x},\mathbf{y}}) &= \log(e^{(P_S,\mathbf{x})} + 1) + \log(e^{(P_S,\mathbf{y})} + 1) \\ & \quad b^0_{\mathbf{x},\mathbf{y}} \log(e^{(P_S,\mathbf{x})} e^{(P_S,\mathbf{y})} + 1) \\ & \quad (1 - b^0_{\mathbf{x},\mathbf{y}}) \log(e^{(P_S,\mathbf{x})} + e^{(P_S,\mathbf{y})}) \\ &= \log(e^{(P_S,\mathbf{x})} + 1) + \log(e^{(P_S,\mathbf{y})} + 1) \\ & \quad b^0_{\mathbf{x},\mathbf{y}} \log(e^{(P_S,\mathbf{x}) + (P_S,\mathbf{y})} + 1) \\ & \quad (1 - b^0_{\mathbf{x},\mathbf{y}}) \log(e^{(P_S,\mathbf{x})} + e^{(P_S,\mathbf{y})}).\end{aligned} \tag{19}$$

B.2. Gradient

The gradient that backpropagates in is $\partial\text{BCE}/\partial(\mathbf{x}) + \partial\text{BCE}/\partial(\mathbf{y})$. As $\mathbf{x}$ and $\mathbf{y}$ are interchangeable in the loss expression, we derive only the $\mathbf{x}$-term. We have:

$$\begin{aligned}\frac{\partial\text{BCE}(b_{\mathbf{x},\mathbf{y}}, b^0_{\mathbf{x},\mathbf{y}})}{\partial(\mathbf{x})} &= S(-(\mathbf{x}))\, b^0_{\mathbf{x},\mathbf{y}}\, S(-(\mathbf{x}) + (\mathbf{y})) \\ & \quad (1 - b^0_{\mathbf{x},\mathbf{y}})\, S(-(\mathbf{x}) - (\mathbf{y})),\end{aligned} \tag{20}$$

where we used the facts that

$$\frac{\partial \log(e^a + 1)}{\partial a} = \frac{e^a}{e^a + 1} = S(a), \tag{21}$$

$$\frac{\partial \log(e^{a+b} + 1)}{\partial a} = \frac{e^{a+b}}{e^{a+b} + 1} = S(a + b), \tag{22}$$

and

$$\frac{\partial \log(e^a + e^b + 1)}{\partial a} = \frac{e^a}{e^a + e^b} = \frac{e^{a-b}}{e^{a-b} + 1} = S(a - b). \tag{23}$$

The gradient expression can be further simplified by exploiting the fact that $b^0_{\mathbf{x},\mathbf{y}} = 1$ when $(\mathbf{x},\mathbf{y}) \in Q_{\text{same}}$, and $b^0_{\mathbf{x},\mathbf{y}} = 0$ when $(\mathbf{x},\mathbf{y}) \in Q_{\text{opp}}$.

C. Latent space analysis

In this section, we propose an analysis of the latent space through an evaluation of the generation capacity of the network, the possibility of interpolating in the latent space between two shapes, as well as a visualization of the latent space itself over a multi-class dataset.

C.1. Shape generation

We use the same network as in Occupancy Networks [13], which uses a variational auto-encoder formulation. As in previous works, we can use our trained network for shape generation. In our network, the encoder outputs a latent vector $\mathbf{z}_S$ representing the shape; the decoder outputs the probability of occupancy given the latent code and the spatial coordinates of a point at which to evaluate the occupancy. We add a latent regularization term similar to [13] to the training loss so that the latent space allows generation and interpolation between shapes. For a complete overview of variational auto-encoders, one could refer to [7].

Instead of generating a latent vector from P_S, we pick a random $\mathbf{z}_S$ and reconstruct the shape based on this latent vector. A few generation results are shown in Figure 6.

Generation for a model trained on a single category. We show in Figure 6(a) some shapes generated with a model trained only on cars. The latent code $\mathbf{z}_S$ is generated according to a normal law where the mean and standard deviation are computed over the train set. We observe that our NeeDrop model is able to generate plausible shapes, including details such as wing mirrors or bumpers.

Conditional generation. In Figure 6(b), we show a few conditional generation results for several classes of the ShapeNet dataset. For conditional generation, we follow the same procedure as in the previous section, except for the normal distribution parameters that are evaluated on the train set restricted to the desired category.

C.2. Shape interpolation

To illustrate the consistency of NeeDrop's latent space, we show here that it can be used to interpolate smoothly between two shapes via a simple linear interpolation between the corresponding latent codes: no spurious artifacts or intermediate shapes unrelated to the end-point shapes appear on the linear path from one latent code to another.

In Figure 7, we present interpolations between three couples of shapes. For each row of the figure, we linearly interpolate the latent vectors with a fixed step and show the reconstructions. Please note that each row corresponds to different object categories but a single network is used for all of them, trained on all categories of ShapeNet.

(a) Random generation with a model trained only on cars

(b) Random generation with a model trained on all categories

Figure 6. Shape generation with NeeDrop using latent vectors randomly sampled according to the target class distribution.

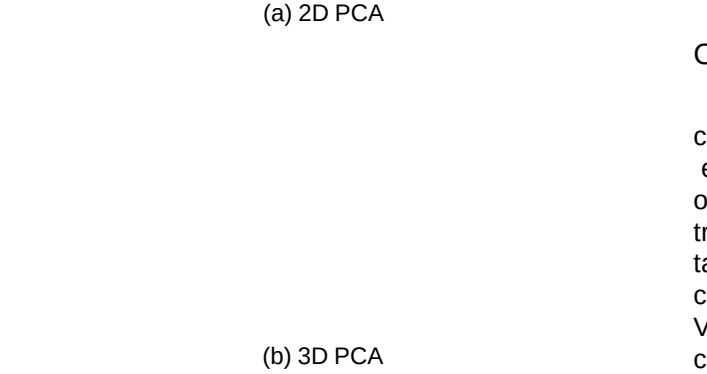
We observe good interpolations between the shapes, with a smooth transformation from one shape to the other and without unrelated shapes appearing along the path (e.g., a chair interpolated between cars). Each interpolated shape seems a plausible realization of the category it belongs to.

C.3. Latent space visualization

In order to further study the latent space of the multi-category ShapeNet models, we perform a Principal Component Analysis (PCA) of sampled latent vectors with 2 or 3 principal dimensions. Corresponding visualizations are displayed on Figure 8. We represent each category with a different color to emphasize the natural clustering of the latent vectors for each shape category.

We can see clusters of shapes from the same category, sharing common geometric features (e.g., wings for planes).

Figure 7. Linear interpolation in NeeDrop's latent space between two shapes (shape 1 and shape 2) of the test set.



(a) 2D PCA

(b) 3D PCA

Figure 8. Principal component analysis of the latent space of the network trained on ShapeNet parts, with one color per category.

As the clusters are mostly separable (one model category per cluster) and somehow convex, any linear interpolation between two shapes of the same class will likely remain in the cluster. Figure 8 visually accounts for the success of the previously presented interpolations.

Method	Overall accuracy
SVM	94.5%
MLP	93.2%
Random Forest	93.3%

Table 2. Accuracy of classi ers learned on the latent representations of ShapeNet parts.

C.4. Classi cation based on latent vectors

In order to further investigate the separability of the classes in the latent space, we train three simple classi- ers on the latent vectors from the train set and evaluate on the test set. Please note that for ShapeNet, we use the train/test split used usually for part segmentation. It contains 13 very different model categories, from planes to chairs. We experimented with three classi ers: a Support Vector Machine (SVM) [5], 1-hidden-layer multilayer perceptron (MLP) and a Random Forest [10]).

The results are presented in Table 2. All classi ers perform more or less equally well, with more than 93% overall accuracy. Besides, the parameters of the classi ers were not tuned for this experiment; we used the default parameters provided by the Scikit-Learn library [14]. These results validate the fact that the different shape categories are mostly separable in the latent space, as can be seen in Figure 8.

D. Needle dropping analysis

As described in the main paper, the loss function is composed of two terms L_{opp} and L_{same}. The former, computed on the set Q_{opp} of “opposite-side” needles, aims at setting

different labels on the different sides of the surface. The latter, computed on the set Q_{same} of "same-side" needles, enforces consistent labels inside and outside the shape.

The construction of valid Q_{opp} and Q_{same} is critical in our approach. Ideally, these two sets of needles should contain only "good" needles, i.e., needles for which their end-points are indeed on opposite sides of the surface for Q_{opp}, and on the same side of the surface for Q_{same}. In practice, as the method is fully self-supervised, such objective is difficult to reach and each set contains wrong needles.

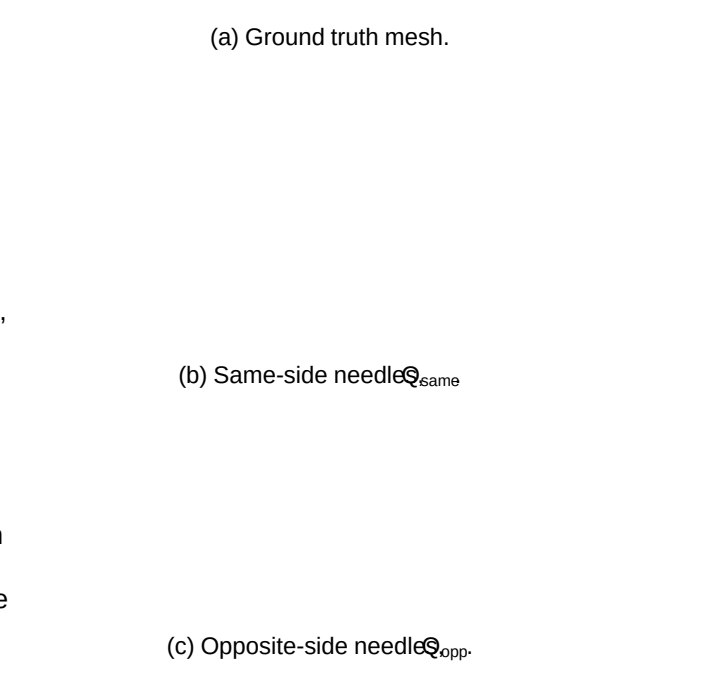
D.1. Needle error localization

Figure 9 presents a random draw of Q_{opp} and Q_{same} on two shapes, for the default value of h. Correct needles are green and wrong needles are red.

First, we observe that a majority of the needles are green, which validates the default value used for h: $r(p) = d_p = 3$, where d_p is the distance between a point p to its closest point in P.

Second, the number of errors on Q_{same} appears to be approximately the same on both 3D models.

Third, we observe more errors in Q_{opp} on the plane model than on the car model. One possible explanation is that Q_{opp} is sensitive to the curvature of the surface to be estimated. To validate this assumption, we compute the curvature of the ground-truth surface, as illustrated on Figure 10(a), and compare it to the concentration of error for Q_{opp}, as shown in Figure 10(b). These concentrations are obtained by aggregating a large number of random pickings of Q_{opp}. Each wrong needle votes for the closest vertex of the surface mesh and a Gaussian filter is applied for smooth visualization. We may observe that the errors for Q_{opp} indeed concentrate in high-curvature regions.



(a) Ground truth mesh.

(b) Same-side needles Q_{same}

(c) Opposite-side needles Q_{opp}.

Figure 9. Needle dropping visualization for Q_{same} (b) and Q_{opp} (c) for two given shapes (a). Green needles belong to correct set; red needles are in the incorrect set and should have been placed in the other set according to the ground-truth mesh (if it could have been known at training time).

D.2. Needle error statistics

The construction of Q_{opp} is based on the hypothesis that the surface is locally planar. At the scale of observation (h), this assumption holds in mostly planar parts of S (0 curvature), but it is erroneous in high-curvature areas. A seemingly good solution would be to reduce h such that we observe the surface at a scale where all neighborhoods of points are planar. In Table 3, we use different values of h and compute the rate of correct needles for Q_{opp} and Q_{same}.

As expected, the smaller the scale h, the better Q_{opp}. But on the other hand, reducing the value of h simultaneously decreases the rate of good needles in Q_{same}. This confirms that one should find a trade-off when setting h.

An illustration of this phenomenon on a 2D case is proposed in Figure 11. As explained in Section 3.2.2 of the paper, when constructing Q_{same} we first pick points in space and then build needles with the points of $P_{same} \cup P_{opp}$. We consider 8 cases, with two h values (large value on the first row and small value on the second row), two configurations

(a) Curvature (b) Q_{opp} error

Figure 10. Visualization of the surface curvature (a) and of the frequency of incorrectly classified needles, estimated over several random draw of Q_{same} (b).

of needles (different orientations, first and second column) and two cases, with and without a point from P_{same} to build a needle (Figure 11(a) and (b)). In the latter case, in the illustration, the point from P_{same} is placed at distance h from the shape.

The colored areas (except those in red) represent the

Category	Multiplier ($\alpha_h(p) = \frac{d_p}{3}$)	Opposite-side	Same-side
Planes	2	70.2%	92.7%
	1	83.9%	92.4%
	0.5	91.8%	92.0%
	0.1	98.2%	91.6%
	0.01	99.8%	91.5%
Cars	2	76.0%	90.6%
	1	82.9%	90.7%
	0.5	88.8%	90.3%
	0.1	97.1%	89.7%
	0.01	99.7%	89.5%

Table 3. Rate of good needles for Q_{opp} and Q_{same} as a function of α_h, when constructed on planes and on ShapeNet cars.

Voronoï cells around each point, inside which a sampled point of P_{same} yields a good needle. On the contrary, if a new point of P_{same} falls in the red area, then the constructed needle falls on the opposite side of the surface and produces a bad needle for Q_{same}. In all these configurations, reducing α_h (i.e., going from first row to second row) leads to larger red areas, thus increasing the probability of wrong needles in Q_{same}. It corroborates the observation made on ShapeNet in Table 3.

(a) No point belonging to P_{same}

(b) A point belonging to P_{same} is also present.

Figure 11. Influence of α_h on the probability of picking a wrong needle for Q_{same}: (a) with only points from P_{opp}, and (b) with points from P_{opp} and a point from P_{same} ($P_{space} = P_{same} \cup P_{opp}$ is disjoint from P_S.)

E. Point sampling at training time

In the main paper, for comparison purposes, we followed the same training procedure as in the previous papers, i.e., sampling new points on the surface of each object at each epoch. Therefore, over a long training period of N_e epochs, the overall information originating from each shape cannot be strictly considered to be limited to a set of 300 points. Yet, it is not equivalent either to training with $300 N_e$ points because the information comes in small fractions of 300 points per shape, that have to be processed at a time. It impacts for instance the distance to the nearest neighbor, which is very different when computed on 300 points or on $300 N_e$ points.

Here, we perform another set of experiments with fixed points for each training shape (same points at each epoch) to investigate the consequences of a truly limited amount of information on each shape at training time. In this setting, each shape is sampled once and for all with 300 points.

Results on the three datasets used in the main paper are presented in Table 4. Conclusions are threefold. First, we notice that the training is stable, no divergence is observed. Second, fixed-points training reaches almost the same performances as training with point re-sampling. Third, the size of the training dataset has an impact on the robustness to sampling procedure: on the car subset, the average Chamfer distance increases by 10^{-5}, while on all ShapeNet performances are nearly identical with both point sampling strategies.

F. Robustness to noise and density variations

In the main paper, as in previous works, we experimented without noise except for the multi-category ShapeNet experiment where we used a very low level of noise equal to the level used in [13]. In this section, we are interested in studying further how a model behaves when the number of input points and the level of noise differ from the values used in the training dataset.

On Figure 13, we present a qualitative robustness study of the predictions of a network trained on the car subset of ShapeNet, with 300 input points and no noise. With this single network, and for the same test shape, we predict the surface from various point cloud samplings: from no noise to a high level of noise, and from the "native" point cloud

Point sampling	Chamfer ℓ_2 # Mean	Median
Resampled	1.703	1.109
Fixed	2.461	1.897

(a) ShapeNet cars, closed meshes [16], results 10^{-4}, cf. [3].

Point sampling	IoU "	Cha. ℓ_1 #
Resampled	0.666	0.112
Fixed	0.669	0.106

(b) ShapeNet subset of [4], all classes, results 10^{-1}, cf. [13].

Sampling	Chamfer ℓ_2 # 5%	50%	95%
Resampled	0.269	0.433	1.149
Fixed	0.202	0.526	1.322

(c) DFaust, results 10^{-3}, cf. [1].

Table 4. Performance of NeeDrop on complete point clouds with two different point sampling procedures at training time: 300 points newly sampled at each epoch on each shape (resampled) / same 300 points sampled on each shape for all epochs (xed).

size used at training time (300 points) to a higher resolution (up to 10k points).

First, without noise, our model is globally insensitive to the number of points. This is due to the PointNet structure in the encoder: the information being gathered using global max-pooling, no neighborhood or density notions are involved in the latent vector predictions.

Second, for the same small number of points, the effect of noise is limited. We observe a progressive loss of details and an in ation of the shape, with hallucinated surfaces appearing inside the actual car shape. This is a behavior somehow similar to what we can be observed when looking at the convex envelop of the point cloud with added noise.

Finally, the conjunction of noise and of an increased number of points worsens a lot the quality of the latent representations, and thus of the predicted shape. The point-wise information aggregation of the encoder, which ensures robustness to the number of points when no noise is added, is conversely the source of non-robustness to noise: no local averaging effect is then possible. In addition, as the network has been trained without noise, each point is considered meaningful, thus leading to a wrong latent vector and a resulting surface that is very complex, with a lot of folds. This con guration is particularly challenging for the networks as it differs a lot from the training conditions.

A solution to palliate this degradation could be to add noise and use variable point cloud sizes at training time, or to pre-process the point cloud with a denoising and outlier removal algorithm [9].

(a) ShapeGF

(b) NeeDrop

Figure 12. Comparing the point cloud generated by ShapeGF to a point cloud (of the same size) sampled on our generated mesh.

G. Comparing to ShapeGF

To illustrate why we wrote in Section 4.1 that ShapeGF [2] yields “noisy” points (cf. Fig. 3(a)), we scanned as many points on our generated surface, and zoomed on a slice of the cabin and on an engine (see Fig. 12). With ShapeGF, the aircraft engines are somehow blurry and not well localized, and the two surfaces of thin volumes, such as wings, cannot be told apart.

H. Reconstructing KITTI shapes

Figure 14 presents more results in the same setting as Figure 3(c) in the main paper. AtlasNet is trained on the KITTI point clouds.

The gure also shows the transfer capacity of our model between two datasets: training on ShapeNet and testing on KITTI. We actually provide transfer results for two models trained on the ShapeNet dataset: a rst model trained with complete shapes (model from the main paper), and a second model trained with partial point clouds. For the latter model, at training time, we generate partial point clouds on the y using the visibility operator of [11] to simulate visibility from a single viewpoint. As there exists a symmetry plane (vertical, from front to rear of the car) for most cars, we exploit this symmetry at training time, by concatenating the sampled point cloud and its mirror-image symmetrized version. Please note that although it helps densifying and completing the training point clouds, it does not guarantee a complete coverage of the whole shape. In particular, cars that are seen from the front are not completed in the back and conversely. Besides, the underside of cars remain unsampled as their are never seen by the lidar.

The domain gaps include: (1) going from a training on points uniformly sampled on perfect shapes without noise to a training on points captured by an actual lidar, and (2) go-

ing from a training on complete shapes to a training on partial shapes (i.e., point cloud not covering the whole shape).

We observe that all models transfer well to the KITTI dataset, producing plausible reconstructions.

However, contrary to our intuition, we also observe that the model trained on partial views does not perform better than the model trained on the complete point clouds. To our understanding, this is mainly due to the fact a complete point cloud carries much more information than a partial one. The full-model latent space is therefore more guided and constrained by the whole geometry of a car (front and rear should exists, with four wheels for each model, etc.), compared to a space learned only on partial point clouds.

Details of the training for the KITTI experiment. The object-speci c point clouds for cars and pedestrians are extracted from the lidar scans using the 3D bounding boxes provided for the KITTI 3D detection challenge. Training point clouds are generated using the training 3D boxes. The shapes presented in the paper and in the supplementary material are not generated from points seen at training time; they are generated using point clouds extracted from validation 3D boxes.

At training time, as well as test time, we sample 300 points inside a given 3D bounding box. If the box contains less than 300 points, some points are re-sampled to reach the desired number of points. In practice, duplicating points (in case there is less than 300 points) has no particular effect on the predicted latent vector, beyond the fact that there is less information to rely on. Indeed, the backbone (a residual PointNet) processes the points independently, and operates feature aggregation with max-pooling operations, which are invariant to redundancy.

I. Pseudo-code of the loss

Listing 1 provides the pseudo-code of the NeeDrop loss for Python with a Pytorch function implementation style. This function implements equations (13) and (14) from the paper. The only extra step needed is the clamping of the inputs in the forward function to prevent the logarithm to produce in nite values. Please note that this operation is only used in the forward function, and that it has no in uence on the backward function.

References

[1] Matan Atzmon and Yaron Lipman. SAL: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2565–2574, 2020.

[2] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III*, pages 364–381. Springer, 2020.

[3] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, December 2020.

[4] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[6] Harm de Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron Courville. Modulating early visual processing by language. In *31st International Conference on Advances in Neural Information Processing Systems (NeurIPS)*, pages 6597–6607, 2017.

[7] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[8] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. In *International Conference on Learning Representations (ICLR)*, 2017.

[9] Xian-Feng Han, Jesse S Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, and Liping Xiao. A review of algorithms for filtering the 3D point cloud. *Signal Processing: Image Communication*, 57:103–112, 2017.

[10] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 278–282. IEEE, 1995.

[11] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. In *ACM SIGGRAPH 2007 papers*, pages 24–es. 2007.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[13] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4470, 2019.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. DISN: Deep implicit surface network for high-quality single-view 3D reconstruction. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

```
class NeeDropLoss(Function):
  # input0, input1: network output logit (without sigmoid) for first and second needle end point
  # target: needle label for the loss to be computed with 1 for same-side, 0 for opposite-side

  def forward(ctx, input0, input1, target):

    ctx.save_for_backward(input0, input1, target) # save for backward

    # clamp to avoid infinite value in logarithm
    input0, input1 = input0.clamp(-10,10), input1.clamp(-10,10)

    e0, e1 = exp(input0), exp(input1)

    loss = log(1+e0) + log(1+e1)
    mask = (target==1) # term specific to label 1 (same-side)
    loss[mask] = loss[mask] - log(1+e0*e1)[mask]
    mask = (target==0) # term specific to label 0 (opposite-side)
    loss[mask] = loss[mask] - log(e0+e1)[mask]
    return loss

  def backward(ctx, grad_output):

    input0, input1, target = ctx.saved_tensors # get saved input and targets

    # gradient for input_0
    grad_input0 = sigmoid(input0)
    mask = (target==1) # term specific to label 1 (same-side)
    grad_input0[mask] = grad_input0[mask] - sigmoid(input0+input1)[mask]
    mask = (target==0) # term specific to label 0 (opposite-side)
    grad_input0[mask] = grad_input0[mask] - sigmoid(input0-input1)[mask]
    grad_input0 = grad_input0 * grad_output

     # gradient for input_1
    grad_input1 = ... # exact same operations as for grad_input0 with input1

    return grad_input0, grad_input1, None
```

Listing 1. Needrop loss pseudo-code (PyTorch style)

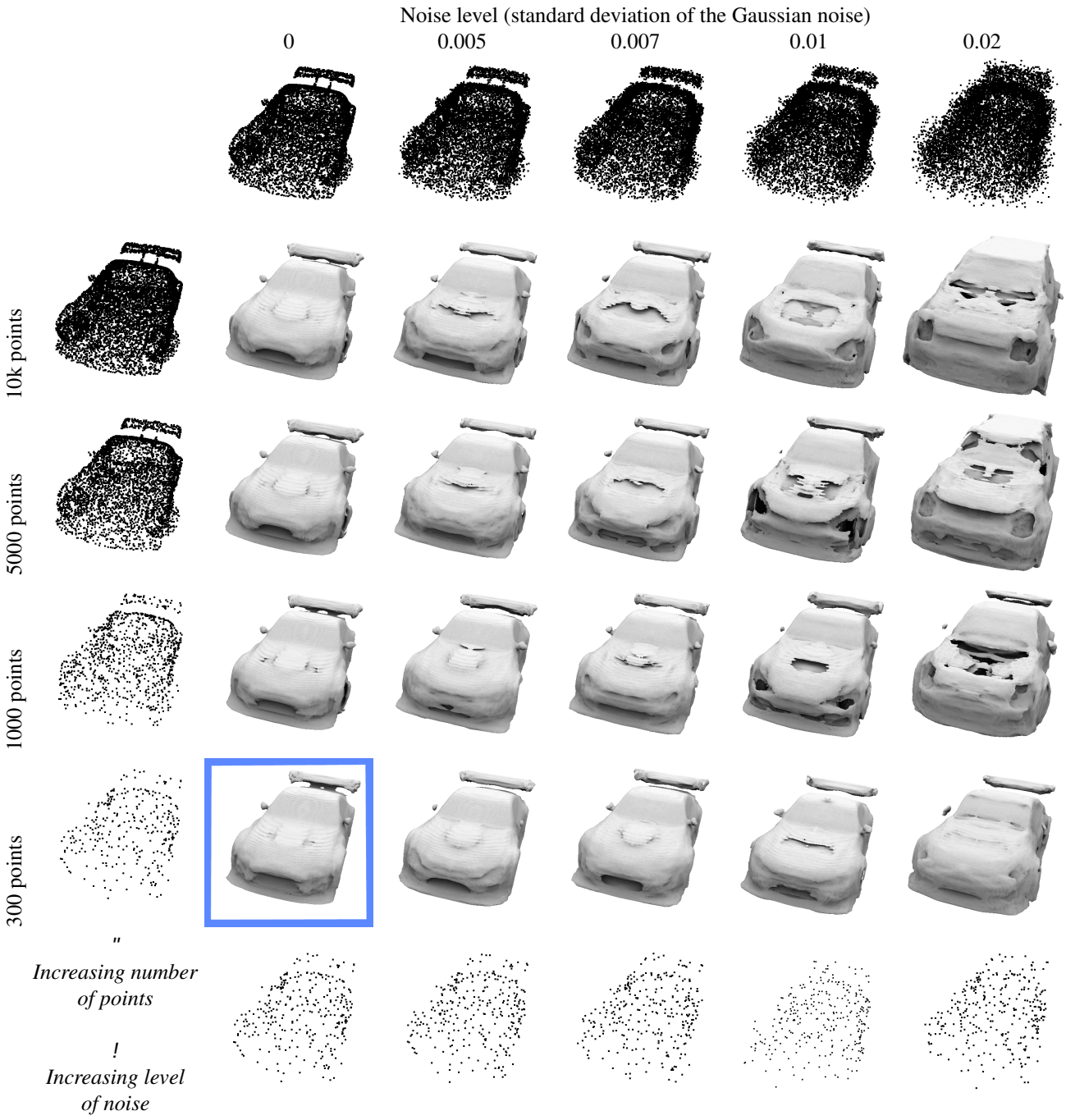


Figure 13. Robustness of a single network to a varying number of points and a varying level of input noise. The framed reconstruction corresponds to the number of points and to the level of points used at training time; all predictions have been made with the same network.