Unsupervised Point-Cloud Reconstruction

Vittorio Pellegrini
Politecnico di Torino
s284692@studenti.polito.it

Andrea Silvi
Politecnico di Torino
s290312@studenti.polito.it

Fabio Tatti
Politecnico di Torino
s282383@studenti.polito.it

Abstract

Point Cloud processing is a challenging task that has been demonstrated to be solved with good performances exploiting deep learning techniques. In this paper we explore Point Cloud reconstruction combining several existing encoders and decoders. We then try to enrich the unsupervised point completion task by jointly training a supervised segmentation task. We further improve this with Onion-Net, a novel way to extract additional features respecting permutation and geometric transformation invariance.

1. Introduction

Sources such as LiDAR scanners and stereo cameras allowed in recent years to build massive point cloud datasets. This led to increased interest in Geometric Deep Learning. This relatively new field focuses on the task of learning over unordered and unstructured set of points. The nature of point cloud data makes impossible to directly use Computer Vision techniques to Geometric Deep Learning models. A deep learning model that processes point clouds must guarantee:

- Permutation invariance: since the input is a set of points, their order should not impact on the model output;
- Geometric transformation invariance: rigid transformations applied to the input should not alter the output's results.

One of the more challenging tasks in this field is the reconstruction of point clouds leveraging autoencoders. This kind of model firstly encodes the input point cloud into a latent code vector; then this code is passed through a decoder network which tries to reconstruct the shape of the original input, as shown in Fig. 1. Historically there have been proposed few encoders for point clouds. PointNet [5] is one of these. It processes point clouds in an innovative way: the network directly consumes unordered point sets as inputs

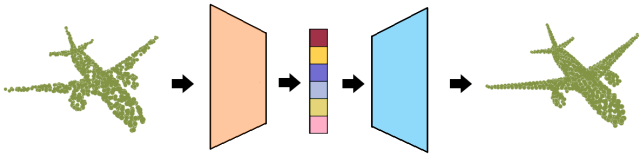

Figure 1: **Autoencoder architecture schema.**

and processes them independently, aggregating the information globally by means of a symmetric function. Unfortunately, this kind of encoder inherently lacks topological information, so designing a model to recover topology can enrich the overall representational power. This is what drives the idea behind DGCNN [7], which exploits Edge Convolution in order to extract features dynamically, incorporating local neighborhood information. In our experiments we use the previous two architectures as encoders, while, for the decoding part, we decide to implement two different versions:

- a fully connected decoder, composed by multiple fully connected layers;
- the Point Pyramid decoder from [4], adapted for reconstruction, which exploits different levels of granularity to better reconstruct the details of the point cloud.

The shape completion task represents an interesting challenge to be solved leveraging deep models: given an occluded point cloud, the architecture aims to recover the overall shape of the original cloud. For this task we propose four different architectures. Firstly we adapt the previous reconstruction framework for completion, taking PointNet as encoder and trying different solutions for the decoder part. Then we take inspiration from PF-Net [4]. The outstanding performances reached by this architecture are partially due to the fact that the model tries to predict only the missing part of the point cloud. In order to reach comparable results, we propose two different variations of PF-Net, but without including their adversarial architecture. In the first proposal we try to extend the network by adding a parallel supervised segmentation task to the unsupervised

shape completion one. Onion-Net is our second proposal: it aims to improve the performances of the previous architecture by extending its latent code. This is done extracting cumulative spherical features for each cropped point cloud, exploiting the segmentation labels predicted by the secondary task. The spherical features extracted by Onion-Net guarantee geometric rotation invariance.

2. Related Work

2.1. Deep Learning on Point Clouds

Following the breakthrough results of convolutional neural networks (CNNs) in vision, there has been strong interest to adapt such methods to geometric data. Pointnet [5] is a MLP architecture that learns how to extract point features from 3D raw point clouds which are then aggregated into a global shape representation. PointNet++ [5] extends on the previous model by hierarchically combining multiple PointNet modules. DGCNN [7] introduces a new operation, EdgeConv, to better capture local geometric features of point clouds while still maintaining permutation invariance. The global representations extracted from these models can be used in many different applications that include shape classification, part segmentation and also reconstruction of the original point cloud.

2.2. 3D Shape Completion

The first learning-based approach in solving this task comes with PCN [9], which implements the Folding operation in order to use a grid to approximate a relative smooth surface and complete the shape. The main drawback of those approaches is the effort wasted to predict not only the missing points, but also the existing ones, since the whole cloud is reconstructed. This is what drives PF-Net's [4] main idea: predict only the missing points, focusing on reconstructing three different levels of granularity in order to obtain more details of the shape. Recent works such as [1] combines supervised and self-supervised learning, in order to jointly learn a shared data representation.

3. Architecture Overview

3.1. Point Cloud Reconstruction

3.1.1 PointNet Autoencoder framework

We implement an autoencoder network composed by a PointNet encoder and a fully connected decoder. The encoder takes n points as input, applies input and feature transformations on each point, and then aggregates point features by max pooling. The max pooling layer acts as a symmetric function to aggregate information from all the points and ensures that the model is invariant to input permutation. It returns a code of fixed length. This is then passed through a fully connected network in order to make the latent code size flexible. The output is then fed to a concatenation of five fully connected layers, followed by a *Tanh* module that returns an output between -1 and 1. We also try to extend the decoder, adding two more linear layers.

3.1.2 DGCNN Autoencoder framework

We change the encoder of the previous architecture to the DGCNN one. Then we try several types of decoders. The DGCNN encoder exploits local geometric structures by constructing a local neighborhood graph and applying convolution-like operations called edge convolutions (EdgeConv) on the edges connecting neighboring pairs of points. The network first constructs the graph as the k-nearest neighbour graph of X. It then defines the edge features as

$$e_{i,j} = h_\Theta(x_i, x_j) = \hat{h}_\Theta(x_i, x_j - x_i), \tag{1}$$

x_i and x_j being two vertices of the neighbourhood. Finally it applies a channel-wise aggregation operation (max and average) on the edge features associated with all the edges emanating from each vertex. The graph is then dynamically updated after each layer of the network. That is, the set of k-nearest neighbors of a point changes from layer to layer of the network and is computed from the sequence of embeddings. Proximity in feature space differs from proximity in the input, leading to nonlocal diffusion of information throughout the point cloud. The first decoder we concatenate is the same proposed at 3.1.1. While a fully-connected decoder is good at predicting the global geometry of a point cloud, it always causes loss of local geometric information since it only uses the final layer to predict the shape. Therefore we decided to leverage a more robust decoder architecture: Point-Pyramid-Decoder (PPD) [4], adapted for the reconstruction task. It predicts primary, secondary, and detailed points from layers of different depth. Primary points and secondary points try to match their corresponding feature points and serve as the skeleton center points to propagate the overall geometry information to the final detailed points. We obtain the primary and secondary ground truth point clouds by applying iterative furthest point sampling (IFPS) [6] to the input.

3.2. Point Cloud Completion

For the resolution of the aforementioned task, we developed four different architectures. The following sections are dedicated to the detailed description of such networks. All make use of the same cropping method, taken from [4].

3.2.1 Naive Point Cloud Reconstruction Network

This architecture is an adaptation of the autoencoder from 3.1.1 for shape completion. It takes as input a cropped point

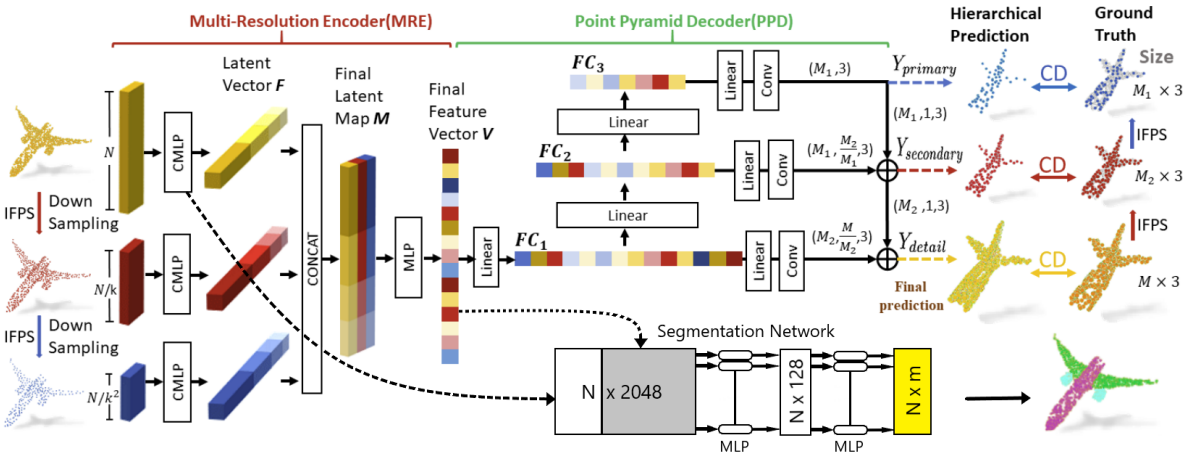

Figure 2: **The Multi-Task architecture**. The incomplete point cloud (yellow) is sampled into different scales (red and blue) through IFPS and all three are given as input. From the first CMLP we obtain the points features, that are then concatenated to the final feature vector and given as input to the segmentation network. PPD also predicts the missing point cloud.

cloud and returns as output the overall shape that is then compared with the complete original one.

3.2.2 PointNet Based Multi-Task Network

As the next step we keep PointNet as the encoder while we implement a two-branches architecture composed by PPD [4] to solve the main unsupervised completion task, and a fully connected decoder that solves the secondary supervised part segmentation task. The idea is that the two tasks can complement each others thus learning more robust features [1], leading to better performances. Here we train the network by predicting only the missing part of the incomplete shape, like in PF-Net. For the part segmentation branch we provide as input the concatenation of the global latent code with the local point features from the encoder.

3.2.3 PF-Net Based Multi-Task Network

Subsequently, we modify the previous network by changing the encoder with PF-Net's Multi-Resolution-Encoder (MRE), as shown in Fig. 2. Combined Multi-Layer Perception (CMLP) is a feature extractor that concatenates latent vectors from different layers in order to extract both low and high-level features. The architecture exploits three CMLP modules: each receives as input the point cloud either in its full form or downsampled using IFPS. Finally the results from the three modules are concatenated into a single feature vector. This multi-scale vector contains both local and global features, enhancing the ability of the network to extract semantic and geometric information. The point features necessary for the segmentation task are obtained by concatenating the global latent vector with local features taken from the CMLP module that processes the full point cloud.

3.2.4 Onion-Net

The previous architecture is extended by inserting an additional module. The idea is to expand the latent code passed to the decoder of shape completion, by using the labels predicted by the segmentation decoder. In order to ensure geometric transformation invariance, we decided to split the incomplete point cloud in several nested spheres centered at (0, 0, 0). For each sphere the frequency of each segmentation class is computed, considering as labels the predictions of the secondary task. The resulting $N_{spheres}$ vectors are then concatenated, obtaining a single vector of length $N_{spheres} * N_{classes}$, where $N_{classes}$ represents the total number of segmentation classes of all shapes used at training time. In order to enhance internal relations between segmentation class frequencies, the aforementioned vector is passed through a fully connected network composed by 3 layers. As last step, the output of the FC is concatenated to the output code of the MRE and then passed to the PPD for the completion task. The extension is reported in Fig. 3.

3.3. Loss Functions

In both the reconstruction and completion tasks we use the Chamfer Distance (CD), defined in [3], as loss function:

$$d_{CD}(S1, S2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} \|x-y\|_2^2 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} \|x-y\|_2^2 \quad (2)$$

It takes as input two unordered sets $S_1, S_2 \subseteq \mathbb{R}^3$ (the ground truth and the predicted output) and for each point

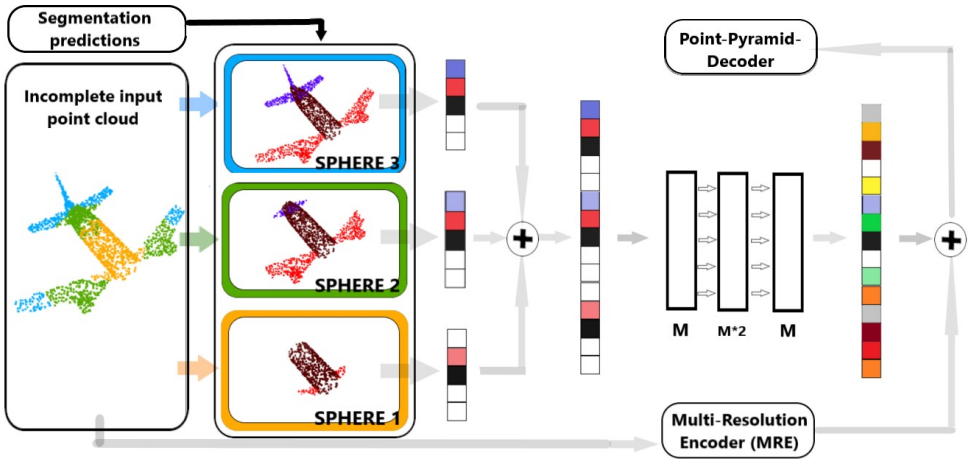

Figure 3: **The Onion-Net architecture**. In the above example, the incomplete point cloud is split in 3 nested spheres. For each of them the frequency of each segmentation class is evaluated, considering the labels predicted by the segmentation decoder. The concatenated feature vector (of length $M = N_{spheres} * N_{classes}$) is then passed through a fully connected network. Its output is concatenated to the MRE output and passed to PPD, for the shape completion task.

the algorithm finds the nearest neighbour in the other set and sums the squared distances up. When using PPD we also implement the multi-stage completion loss from [4], which calculates the CD between the three predicted outputs and their respective ground truth point cloud subsampled accordingly:

$$L_{com} = d_{CD1}(Y_{detail}, Y_{gt}) + \alpha\, d_{CD2}(Y_{primary}, Y_{gt}^{'}) + 2\alpha\, d_{CD3}(Y_{secondary}, Y_{gt}^{''}) \tag{3}$$

In the multitask architectures for the segmentation task we use point-wise cross entropy as the loss. The overall loss becomes

$$L_{joint} = L_{completion} + \gamma\, L_{segmentation} \tag{4}$$

4. Experiments

4.1. Data Generation

To train our models, we use 7 categories of different objects in the benchmark dataset ShapeNetPart [8]. The total number of shapes sums up to 14450 (12124 for training and 2426 for testing). Note that our training dataset consists of the union of validation and training splits from ShapeNetPart. All input point cloud data is centered at the origin and normalized to [-1, 1]. For Point Cloud Reconstruction, 1024 points are randomly sampled with replacement from the input data. For Completion, each shape is uniformly sampled to 2048 points. The incomplete point cloud is then obtained by randomly choosing a view-point as a center and removing the closest 512 points (25% of the input). We also test the generalization capabilities of our models on unseen classes. For Reconstruction we use 13 new categories from ShapeNet-Core [2], while for Completion we use the remaining 9 classes that we have not trained on from ShapeNetPart.

4.2. Evaluation metric

We evaluate our methods using the same metric of [4]. It contains two indexes. The first one, Pred $\rightarrow$ GT (prediction to ground truth) error, computes the average squared distance from each prediction point to its closest in ground truth, and indicates how dissimilar the prediction is from the original. The second one, GT $\rightarrow$ Pred error, is computed swapping prediction and ground truth and indicates how much the original surface is covered by the shape of the prediction. We use the mean between these two indexes for reconstruction, and we refer to it as Chamfer Mean (CM).

4.3. Point Cloud Reconstruction Results

For the PointNet-based autoencoder we note that the deeper decoder does not show improvements with respect

Category	PointNet-AE	DGCNN-AE	Samples
Airplane	1.011 / 0.997	0.768 / **0.765**	2349
Car	2.467 / 2.691	**2.144** / 2.326	740
Chair	2.112 / 2.161	**1.406** / 1.510	3054
Lamp	3.485 / 4.400	**2.019** / 3.014	1261
Motorbike	1.829 / 1.949	**1.568** / 1.865	151
Mug	3.711 / 5.386	**3.017** / 4.081	146
Table	2.311 / 2.454	**1.653** / 1.749	4423
Mean	2.418 / 2.863	**1.796** / 2.187	-

Table 1: **Point cloud reconstruction results.** The shown numbers are the Chamfer Mean between ground truth and prediction, scaled by 1000. We present it as [CM of model trained on all classes / CM of model trained on that single class]. We report the number of training samples for each class.

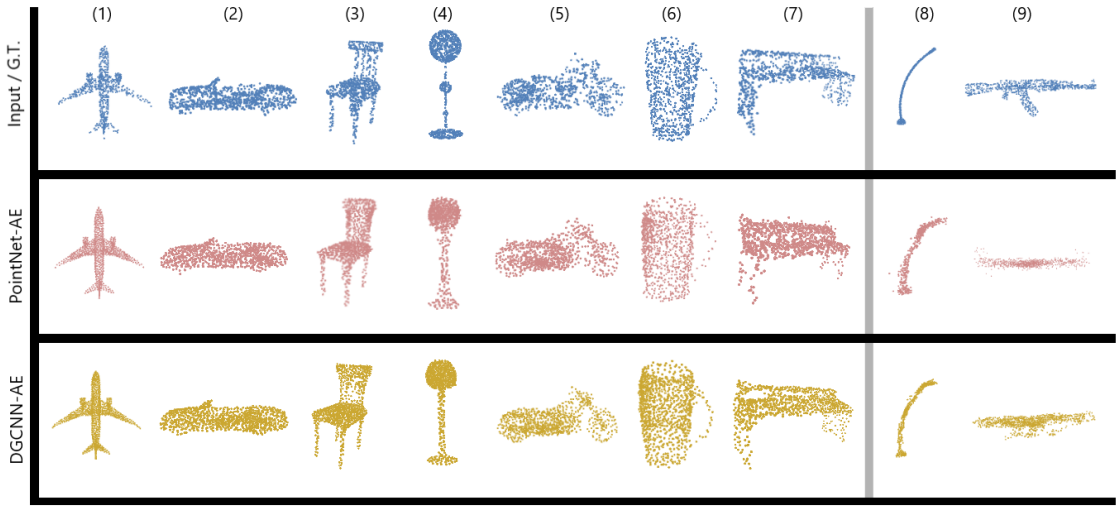

Figure 4: **Qualitative reconstruction results**. The first seven columns are results of categories used during training. The last two columns are results of novel categories. In (2)(6)(7), the DGCNN-based AE is able to reconstruct the geometric structure of the windshield, the mug handle and the table legs. It still misses some very specific details like in (4), still reconstructing the overall shape better than PointNet-AE. It is also able to generalize better on unseen categories (8)(9).

Category	PointNet-AE	DGCNN-AE
Basket	10.713	4.532
Bicycle	4.054	2.891
Bookshelf	5.968	4.284
Bottle	3.905	2.182
Bowl	17.130	5.546
Clock	6.953	3.908
Helmet	9.491	5.101
Microphone	7.088	3.057
Microwave	6.045	3.846
Piano	7.768	4.576
Rifle	2.224	1.276
Telephone	5.070	2.745
Watercraft	3.137	2.061
Mean	6.972	3.539

Table 2: **Reconstruction results on novel categories.** The numbers shown are the Chamfer mean between the original shape and the autoencoder outputs, scaled by 1000.

to the simpler one, deciding to use the latter. We train both autoencoders (PointNet, DGCNN) on single classes. Then we train them with all 7 classes and test separately on each class. The results are shown in Table 1. For each model we also try different sizes of the latent code: this analysis is reported in Supplementary Material B. We also show the results of our two multiclass models in terms of generalization capabilities on unseen classes in Table 2. Considering our DGCNN-based autoencoder, we observe that the results for unseen classes are comparable to the ones obtained on the training classes, showing its satisfactory generalization capabilities. Some qualitative results of the reconstructed shapes are shown in Fig. 4.

4.4. Point Cloud Completion Results

We now present the comparison between the results of our four shape completion models and two representative baselines, such as PCN [9] and PF-Net [4]. **Note** that these last two models are trained using the same data generation method as ours, but on 13 categories of ShapeNetPart (6 more than ours), with the total number of shapes being 14473 (11705 for training and 2768 for testing). Table 3 shows the results. Our multi-task models considerably outperforms our baseline (the Naive shape completion framework). Since our multi-task models predict only the missing shape, as PF-Net, it does not change the original partial shape. To make a fairer comparison, we also discuss the errors on the missing region only in Supplementary Material C. These results show that the supervised part segmentation task actually helps the unsupervised completion task making the latent code more general and representative of the missing point cloud. In Supplementary Material C we further present additional analysis. Some qualitative results of the completed shapes are shown in Fig. 5.

5. Conclusions

We have presented an intuitive solution for shape completion, by adding a part segmentation task. This secondary task allows to learn more general and robust features, leading to an improvement in the shape completion results. Moreover, we have shown that with Onion-Net we can extend the latent code of the primary task by exploiting the secondary task output.

Category	PCN	Naive	PF-Net	PointNet M-T	PF-Net M-T	Onion-Net
Airplane	0.800 / 0.800	0.784 / 0.653	0.263 / 0.238	0.250 / 0.197	0.222 / 0.189	**0.210** / **0.161**
Car	2.324 / 1.738	1.578 /1.682	0.599 / 0.424	0.429 / 0.462	**0.365** / **0.407**	0.371 / 0.411
Chair	1.592 / 1.538	1.701 / 1.407	0.487 / 0.427	0.386 / 0.382	0.374 / 0.308	**0.295** / **0.292**
Lamp	2.757 / 2.003	2.860 / 3.024	1.037 / **0.640**	0.528 / 0.909	0.539 / 0.646	**0.462** / 0.711
Motorbike	1.699 / 1.459	1.365 / 1.338	0.522 / 0.389	0.392 / 0.451	**0.296** / 0.363	0.301 / **0.346**
Mug	2.893 / 2.821	2.661 / 2.804	0.745 / 0.739	0.662 / 0.573	0.559 / **0.514**	**0.534** / 0.550
Table	1.604 / 1.790	1.939 / 1.467	0.525 / 0.404	0.386 / 0.392	0.340 / 0.331	**0.337** / **0.327**
Mean	1.953 / 1.736	1.841 / 1.768	0.597 / 0.466	0.433 / 0.481	0.385 / **0.395**	**0.359** / 0.400

Table 3: **Point Cloud Completion results of the overall point cloud.** The training data of our four models consists of 7 categories of different objects, while PF-Net and PCN are trained on 13 categories. The numbers shown are [Pred → GT error / GT → Pred error], scaled by 1000. We compute those metrics considering as ground truth and prediction the overall point cloud.

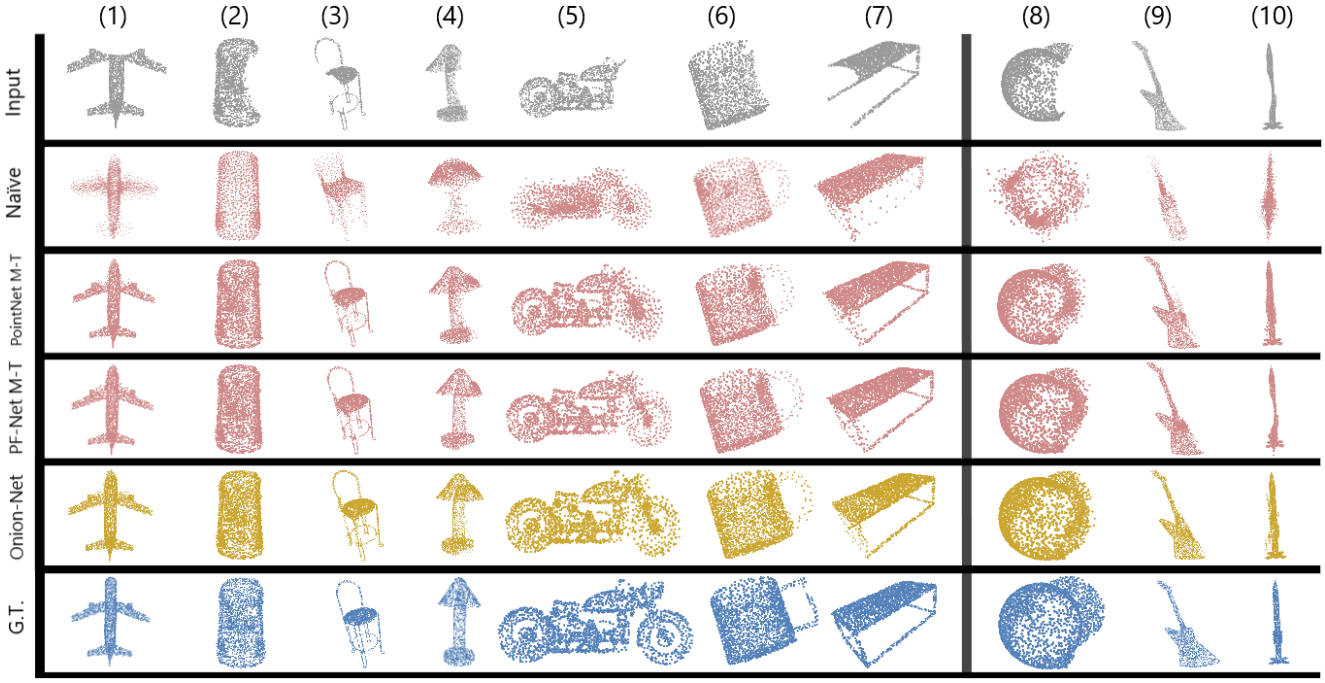

Figure 5: **Qualitative completion results**. The first seven columns are results of categories used during training. The last three columns are results of novel categories. In (3)(7), Onion-Net is able to reconstruct the geometric structure of the chair and the legs of the table with reduced noise. In (1)(4)(5) we see a better reconstruction of the overall silhouette. Still, it misses some specific details such as the squared handle of the mug (6). At the same time we obtain similar results to PointNet M-T on unseen categories (8)(9)(10).

References

[1] Antonio Alliegro, Davide Boscaini, and Tatiana Tommasi. Joint supervised and self-supervised learning for 3d real-world challenges, 2020.

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015.

[3] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image, 2016.

[4] Zitian Huang, Yikuan Yu, Jiawen Xu, Feng Ni, and Xinyi Le. Pf-net: Point fractal network for 3d point cloud completion, 2020.

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

[6] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.

[7] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2019.

[8] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

[9] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network, 2019.

Supplementary material

A. Implementation Details

We implement our models in PyTorch. All of our reconstruction frameworks are trained by using ADAM optimizer with an initial learning rate of 0.0001 and a StepLR scheduler that decays the learning rate by 0.5 every 20 epochs. For the DGCNN encoder we use 10 as the number of nearest neighbours to be considered in the EdgeConv operation. We train all PointNet-based autoencoders for 200 epochs and all DGCNN-based autoencoders for 150 epochs. Our first two completion frameworks are trained by using the same hyperparameters listed above, while the last two are trained by using the same hyperparameters as [4], since we use their network as our skeleton. We fine-tune the number of spheres of Onion-Net considering three values (5, 10, 15) and we end up choosing 15. All of our completion models are trained for 130 epochs. For the multi-stage loss (3) we use the same increasing weights as PF-Net, with α being 0.01 for the first 30 epochs, 0.05 up to the 80th epoch and 0.1 for the remaining 50 epochs. We weight the segmentation loss from (4) by using $\gamma = 0.6$.

B. Reconstruction results

More reconstruction results are reported in this section. In Table 4 we show the latent code sizes of our reconstruction models that resulted in the best performances.

Category	PointNet-AE	DGCNN-AE
Airplane	128	1024
Car	128	128
Chair	512	512
Lamp	512	128
Motorbike	128	128
Mug	512	1024
Table	1024	512
Multiclass	1024	1024

Table 4: **Latent code sizes.** For the single class models, we try as sizes 128, 512 and 1024, while for the multiclass ones 512, 1024 and 1536.

C. Completion results

More completion results are reported in this section. In Table 5 we show the results of our models considering only the error between the output cloud and the cloud missing from the input. We notice that Onion-Net is able to achieve better results in terms of the error between the prediction and the ground truth, while the PF-Net Multi-Task network is slightly better on average at covering the original shape with its prediction. In Table 6 we show our models generalization capabilities. It is worth noting that the PointNet-based network reaches on average better results than the other two models, possibly because PointNet focuses more on the global shape than on the local details. Table 7 shows the three multi-task models performances in terms of accuracy for the part segmentation task.

Category	PCN	PF-Net	PointNet M-T	PF-Net M-T	Onion-Net
Airplane	5.060 / 1.243	1.091 / 1.070	1.077 / 0.849	**0.955** / 0.797	1.031 / **0.773**
Car	2.741 / 2.123	2.489 / 1.839	1.827 / 1.905	**1.556** / **1.683**	1.564 / 1.702
Chair	3.952 / 2.301	2.074 / 1.824	1.697 / 1.602	1.604 / 1.281	**1.302** / **1.212**
Lamp	11.61 / 7.139	5.122 / 3.460	3.488 / 5.046	2.778 / **2.876**	**2.440** / 3.050
Motorbike	4.962 / 1.922	2.206 / 1.775	1.704 / 1.873	**1.290** / 1.516	1.368 / **1.439**
Mug	3.590 / 3.591	3.138 / 3.238	2.849 / 2.320	2.364 / **2.081**	**2.269** / 2.227
Table	2.503 / 2.452	2.235 / 1.934	1.809 / 1.717	1.635 / 1.461	**1.555** / **1.403**
Mean	4.917 / 3.019	2.622 / 2.163	2.064 / 2.187	1.740 / **1.671**	**1.647** / 1.687

Table 5: **Point Cloud Completion results of the missing point cloud.** Our three models are trained on 7 categories of different objects, while PF-Net and PCN are trained on 13 categories. The numbers shown are [Pred $\rightarrow$ GT error / GT $\rightarrow$ Pred error], scaled by 1000. We compute those metrics considering as ground truth and prediction only the missing points of the point cloud.

Category	PointNet M-T	PF-Net M-T	Onion-Net
Bag	1.253 / 1.252	1.126 / 1.120	**0.932** / **0.929**
Cap	**1.226** / 2.992	1.775 / **2.726**	1.801 / 2.819
Earphone	**1.041** / **5.459**	2.810 / 6.595	2.479 / 7.372
Guitar	0.445 / **0.202**	0.420 / 0.245	**0.371** / 0.212
Knife	0.338 / 0.253	0.336 / **0.158**	**0.238** / 0.179
Laptop	0.606 / 0.549	0.505 / 0.408	**0.425** / **0.366**
Pistol	1.391 / **0.459**	**1.118** / 0.584	2.487 / 0.537
Rocket	0.256 / 0.392	**0.218** / 0.339	0.259 / **0.312**
Skateboard	0.783 / 0.428	**0.680** / **0.337**	0.950 / 0.462
Mean	**0.815** / **1.332**	0.999 / 1.390	1.105 / 1.465

Table 6: **Point Cloud Completion results on unseen categories.** The numbers shown are [Pred $\rightarrow$ GT error / GT $\rightarrow$ Pred error], scaled by 1000. We compute those metrics considering as ground truth and prediction the complete point cloud, like in Table 3.

Categories	PointNet M-T	PF-Net M-T	Onion-Net
Airplane	0.898	**0.909**	0.907
Car	0.895	**0.908**	0.907
Chair	0.933	0.936	**0.938**
Lamp	**0.871**	0.862	0.862
Motorbike	0.834	0.855	**0.860**
Mug	0.950	0.963	**0.988**
Table	**0.944**	0.941	0.941
Mean	0.904	0.911	**0.915**

Table 7: **Part segmentation accuracy results.**