Unsupervised Point-Cloud Reconstruction

Luca Barco
Politecnico di Torino
s276072@studenti.polito.it

Stefano Bergia
Politecnico di Torino
s276124@studenti.polito.it

Daniela De Angelis
Politecnico di Torino
s277493@studenti.polito.it

Abstract

Point-Clouds are one of the most powerful and easiest methods to represent 3-D objects. They are unordered, unstructured and eager for precise manual annotation due to many possible sources of noise. For these reasons, classical CNNs cannot deal with raw Point-Clouds and they need some computationally expensive data pre-processing operations. In recent years, some deep learning methods started to deal with raw Point-Cloud data for classification and segmentation tasks with excellent results, like PointNet [4] and DGCNN [5]. The aim of this work is to implement a Point-Cloud Auto-Encoder able to encode the main features and reconstruct as precisely as possible the original input in a totally unsupervised way, basing on these previous works. We also studied how it can be used for solving a Point-Cloud Completion task.

1. Introduction

3-D shape understanding and processing are very challenging tasks for which deep learning methods can provide potential results. The main problems are in terms of representation of such objects: the classical approach consists in considering them as a composition of 2-D surfaces or voxels. It lets the existing deep learning methods work with this kind of data but it requires computationally expensive operations.
Another simpler representation, which is also closer to the raw data from depth sensor like LiDAR scanners, is the so called Point-Cloud. It simply consists in a list of xyz coordinates sampled on the object surface. The more the points we have, the better the resolution of the object is.
If we want to extract features from a 3-D object, a first idea could be to feed raw Point-Cloud data to a CNN.
Unfortunately, the nature of Point-Cloud data poses a great challenge to traditional CNNs. That's because the points in the cloud are permutation-invariant (i.e. if we swap the order of two points the representation doesn't change) and it doesn't exist any kind of implicit relationship between points themselves. CNNs were designed with the inner structure of pixels of an image in mind, meaning that the order and the neighbourhood of each pixel are the fundamental sources for feature extraction. Moreover, there exist infinite Point-Clouds representing the same object. Indeed, a 3-D shape can be viewed as a set of surfaces and, given one of them, we can represent it sampling a certain number of points in infinite ways. A Features Extractor should be invariant to this property.
In recent years, research focused on this topic and some solutions were found in order to manage raw Point-Clouds . In particular, the PointNet and the DGCNN methods achieved excellent results in classification and segmentation tasks by changing some assumptions on convolution operation.
In our work we studied these architectures in order to build an Auto-Encoder. It is able to extract the most important features from the input and to correctly reconstruct the object passing them through a fully connected decoder.
We also studied how this Auto-Encoder could behave if we pass an incomplete Point-Cloud as input and we ask it to reconstruct the missing part. We will refer to this task as *Point-Cloud Completion*. Our code can be found here[1].

2. Related Works

2.1. 3-D Object Reconstruction

Fan *et al*. [2] studied how to construct a 3-D Point-Cloud objects starting from an input image. Since Point-Clouds are un-ordered and un-structured, the Euclidean distance can't be directly used to measure distances. They define two loss functions to solve this issue: the *Chamfer Distance* and the *Earth Mover's Distance*.

2.2. PointNet

Qi *et al*. [4] introduced PointNet: it is one of the most powerful deep learning methods dealing with Point-Clouds. It takes Point-Clouds as input and works for both classification and segmentation tasks. It processes each xyz-point independently and, using a symmetric function and a stack of mlp layers, selects the most important features. This data

[1] [https://github.com/StefanoBergia/Unsupervised_Pointcloud_Reconstruction]

can be passed to the final fully connected part of the network that can use it to solve a classification or a segmentation task.
Since Point-Clouds are geometric objects, the learned representation should be transformation-invariant. For this reason, the network applies some input and feature transformations through transformation matrices (predicted using mini-networks, called T-Net). These transformations are useful to normalize the input data and to align the features from different Point-Clouds.
In particular, the final classification part aggregates the main features with a maxpool operation in order to obtain a unique vector describing the overall Point-Cloud.

2.3. DGCNN

Wang *et al*. [5] proposed another approach with DGCNN.
It starts from the PointNet architecture but, instead of considering isolated points, it builds a *local neighbourhood graph* in order to infer local geometrical structures.
This graph is "dynamic" since it is re-computed at each layer of the network because proximity in the input space is different from proximity in the feature space.
This directed graph is used with an *EdgeConv* operation:

> "It applies a channel-wise symmetric aggregation operation (like sum or max) on the edge features associated with all the edges emanating from each vertex.", Wang *et al*. [5]

The general form of this operation at the i-th vertex is

$$x'_i = \square_{j:(i,j)\in E} h_\theta(x_i, x_j) \quad (1)$$

where E is the set containing all the pairs of points, x_i is the central pixel and $x_j : (i,j) \in E$ the patch around it. h_θ is the edge function and $\square$ is an aggregation operation (e.g. max or sum).
The PointNet can be considered as a particular case of a DGCNN network which considers only global information.

2.4. Point-Cloud Completion

Point-Cloud Completion task is one of the most challenging tasks relying to Point-Clouds. There are several works that have proposed different solutions.
In particular, Huang *et al*. [3] introduced PF-Net. It has an innovative approach: instead of trying to reconstruct the entire Point-Cloud to complete the figure, it tries to predict only the missing part. In this way the information related to the input part are not lost and the network focuses only on the missing points.

3. Methods

In this section we propose our approach, basing on PointNet and DGCNN, to reconstruct an input Point-Cloud using an Auto-Encoder and to solve a Point-Cloud Completion task. In particular, we study how PointNet and DGCNN behave as features extractors, using the produced final features vector to correctly re-construct the input data through a fully connected decoder.
Then, taking inspiration from the PF-Net approach, we force our own Auto-Encoder to reconstruct only the missing part of a cropped input Point-Cloud.
In Section 3.2 we describe the task of building an Auto-Encoder, while in Section 3.3 the Point-Cloud Completion task.

3.1. ShapeNet Dataset

We use the ShapeNet dataset, proposed by Chang *et al*. [1] , to train our model.

> "ShapeNet is a richly-annotated, large-scale repository of shapes represented by 3D CAD models of objects", Chang *et al*. [1]

ShapeNet can be useful to train deep learning methods relying on 3-D data tasks. In particular, we use seven categories: *Airplane, Chair, Table, Lamp, Car, Motorbike, Mug*.

3.2. Auto-Encoder

The aim of our work is to design an Auto-Encoder architecture able to reconstruct as well as possible the input Point-Clouds learning their features in a totally unsupervised manner.

3.2.1 General Architecture

Figure 1 shows in details the entire Architecture we propose.
Starting from this structure, we present two variants that differ regarding the *Features Extractor*: one is based on PointNet (Section 3.2.3) and one is based on DGCNN (Section 3.2.4).

3.2.2 Loss Function

We want the reconstructed Point-Clouds to be as close as possible to the input ones. For this purpose, we need to penalize the distance between input and output data. The loss function in this architecture cannot be an Euclidean Distance (like MSE) because Point-Clouds are un-structured and un-ordered.
Recalling the two proposes of Fan *et al*. [2], we decide to solve this problem using a Chamfer Distance-based loss function.
Given two point sets $P_1, P_2 \subseteq \mathbb{R}^3$, the Chamfer Distance is defined as

$$d_{CD}(P_1, P_2) = \sum_{x\in P_1} \min_{y\in P_2} ||x-y||_2^2 + \sum_{y\in P_2} \min_{x\in P_1} ||x-y||_2^2 \quad (2)$$

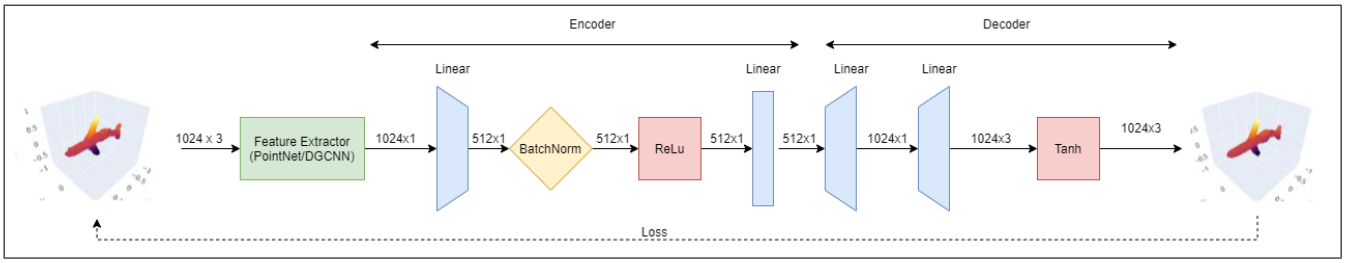

Figure 1. Auto-Encoder Architecture. The network takes a [1024, 3] input Point-Cloud. It is passed to a *Features Extractor* who provides a [1024, 1] output. This vector is forwarded to an *Encoder* who produces a [512, 1] output. Then, the *Fully Connected Decoder* produces a [1024, 3] Point-Cloud.

For each point $x \in P_1$, it finds the nearest neighbour $y \in P_2$ and sums the squared distance between them. The same operation is done in the opposite direction (for each point $y \in P_2$, it finds the nearest neighbour $x \in P_1$ and sums the squared distance between them) and the two results are summed together.
Even if d_{CD} is not a distance function in the strict sense (since triangle inequality does not hold), we can call it "distance" because it's a non-negative function defined on point set pairs. Furthermore, we have a pseudo-symmetric behaviour because the distances are computed along two opposite directions.
In particular, given P_1 as the input Point-Cloud, P_2 as the reconstructed Point-Cloud, N as the number of points of P_1 (that is also equal to the number of points of P_2) and w as a weighting factor, the loss function L can be defined as following, recalling the Equation (2) :

$$L(P_1, P_2) = \frac{w}{2N} \cdot d_{CD}(P_1, P_2) \tag{3}$$

w is necessary because the raw values of the Chamfer Distance are too small.
For the whole work we set w=100.

3.2.3 PointNet-based Features Extractor

We used a PointNet-based Features Extractor as first configuration. We started from the architecture proposed by Qi *et al*. [4].
We have a [1024, 3] input Point-Cloud (i.e. a list of 1024 xyz-coordinates). We apply a transformation to it using a 3x3 matrix predicted using a T-Net. The output of this step has the same shape of the input ([1024, 3]).
Then we apply a shared MLP (with Batch Normalization and ReLU activation function) to obtain a [1024, 64] output.
The resulting vector of shape [1024, 64] is passed to a sequence of two MLP (with the same characteristics of the previous one). They produce [1024, 128] and [1024, 1024] output vectors, respectively.
At this point, a symmetric aggregation is done through a MaxPool operation, obtaining a [1024, 1] vector encoding the overall Point-Cloud.
We use the T-Net to perform an alignment of the input data in order to be sure that all of them are oriented in the same way.

3.2.4 DGCNN-based Features Extractor

We tried a second configuration using a DGCNN-based Features Extractor.
In particular, we adapt the architecture proposed by Wang *et al*. [5] in which the main features of a [1024, 3] input Point-Cloud are encoded at different scales and then they are summarized in a unique [1024, 1] vector. In this way, the obtained information regards also the local neighbourhood of points. This is done through the sequential computation of dynamic graphs and relative EdgeConv operation. In this way, four tensors are extracted: x1= [64, 1024]; x2 = [64, 1024]; x3 = [64, 1024]; x4 = [128, 1024]. They represents the features at each EdgeConv layer. In our work, we change the dimensions of the last two tensors from [64, 1024] and [128, 1024] to [128, 1024] and [256, 1024], respectively.
To summarize these tensors, they are concatenated into a unique output [512, 1024], that is passed to a MLP and to a Max Pooling layer that produces a [1024, 1] vector containing the representation of the main features.
The steps to obtain one of the tensors specified above of dimension [F, 1024] (i.e. F in x1, x2, x3 and x4 is 64, 64, 128, 256, respectively) are the following: given an input Point-Cloud of size [M, 1024], generate a *dynamic graph* of dimension [2M, 1024, K]. The K parameter represents the number of nearest points to consider in order to build the graph and in our work we set $K = 20$.
The graph is passed to a Convolutional layer that produces a [F, 1024, K] output.
The latter is passed to a Max Pooling layer that provides a [F, 1024] tensor.
About the EdgeConv operation, for the *edge function* h_Θ we choose the fifth proposal of Wang *et al*. [5]:

$$h_\Theta(x_j, x_i) = \bar{h_\Theta}\ (x_i, x_j - x_i) \tag{4}$$

because it combines local and global information. In particular we choose the e'_{ijm} operator implemented as a shared MLP.

$$e'_{ijm} = ReLU(\theta_m \cdot (x_j - x_i) + \phi_m \cdot x_i) \quad (5)$$

We choose the max operation as aggregation function

$$x'_{im} = \max_{j:(i,j)\in\epsilon} e'_{ijm} \quad (6)$$

where $\Theta(\theta_1, ..., \theta_M, \phi_1, ..\phi_M)$ is the matrix containing the parameters of the MLP (weights and biases)

3.3. Point-Cloud Completion

Once we have designed our Auto-Encoder, we try to use it in a Point-Cloud Completion task.
We aim to complete a Point-Cloud with a missing region (e.g. an Airplane without a wing). To solve this task we take inspiration from the PF-Net work of Huang Qi *et al*. [3] in which the network tries to reconstruct only the missing part instead of the whole Point-Cloud.

3.3.1 Cropping method and Data Augmentation

Since in this work we use the ShapeNet dataset, we decided to generate by ourselves incomplete Point-Clouds from seven viewpoints.
In order to remove points, we select the **256 nearest neighbours** of an "observation" point (so called *viewpoint*). Furthermore, we decide to perform a sort of "Data Augmentation" choosing a list of viewpoints in order to obtain several cropped Point-Clouds from one input Point-Cloud.
Since the Point-Clouds are all inside the unitary cube $((-1,-1,-1),(1,1,1))$, we use these seven viewpoints: [(1,0,0), (-1,0,0), (0,0,0), (0,1,0), (0,-1,0), (0,0,1), (0,0,-1)]. They correspond to the origin of the axes and to the six centers of the faces of the cube.
Thus, we generate seven samples for each input Point-Cloud.

3.3.2 Architecture and Training

Since for this task we use a "data augmentation" mechanism, the hardware limits took us to choose our Auto-Encoder with the PointNet-based Features Extractor instead of the DGCNN one, in fact it is less complex (it doesn't have the overhead of the computation of the graph).
The only difference with respect to the architecture shown in Section 3.2.1 is in terms of dimension of the input data that contain less points because of the cropping.
Figure 2 shows the detailed architecture.
To train the model we feed it with the cropped Point-Cloud and we compute the loss on the missing part only. Indeed, we obtain two Point-Clouds when we crop the input one:

- *Cropped Point-Cloud* (size: [768, 3]): it contains the partial input Point-Cloud.
- *Missing-Part Point-Cloud* (size: [256, 3]): it contains the 256 points missing part.

We will refer to these names in the following explanation.
Starting from [1024,3] Point-Cloud, seven pairs *[Cropped Point-Cloud, Missing-Part Point-Cloud]* are generated. Then, only one at a time, the Cropped Point-Clouds are passed to the Auto-Encoder Network, that tries to infer only the missing 256 points using an *ad-hoc* Loss Function.

3.3.3 Loss Function

The output of the Auto-Encoder is the reconstruction of the Missing-Part Point-Cloud. We want:

- to penalize the Chamfer Distance between the original Missing-Part Point-Cloud and the reconstructed one
- to keep the global shape homogeneous taking into consideration both Missing and Cropped parts.

Thus, defining:

- P_{gt}: Groundtruth Point-Cloud (size: [1024, 3]).
- P_m: Missing-Part Point-Cloud (size: [256, 3]).
- P_c: Cropped Point-Cloud (size: [768, 3]).
- P_{rm}: Reconstructed Missing-Part Point-Cloud (size: [256, 3]).
- $P_r = P_c + P_{rm}$: Reconstructed Point-Cloud given by concatenation on P_{rm} and P_c (size: [1024, 3]).
- $L(P_1, P_2)$ as the mean Chamfer Distance between Point-Clouds P_1 and P_2 computed using Equation (3).

we can define a loss function, called **Matching Loss function** L_{PCC}:

$$L_{PCC} = L(P_{rm}, P_m) + r \cdot (L(P_r, P_{gt})) \quad (7)$$

where r is a parameter used to find a balance between the two terms reducing the effect of the second one. Indeed, with a large value, the network will reconstruct the whole object (instead of only the missing part) because the order of magnitude of the second term is higher than the first one. In our work it is set to $r = 0.01$.

4. Experiments

In this section we describe our experiments, that were done using the Google Colaboratory tool, which provides us random NVIDIA GPUs with memory of either 16GB or 8GB. The entire training phase was done using an ADAM scheduler. We use a hold-out validation technique on the ShapeNet Dataset, which is splitted in three sets: *Training* (72% of the data), *Validation* (11% of the data) and *Test* (17% of the data).
The entire list of our attempts and further charts can be found in the Appendix.

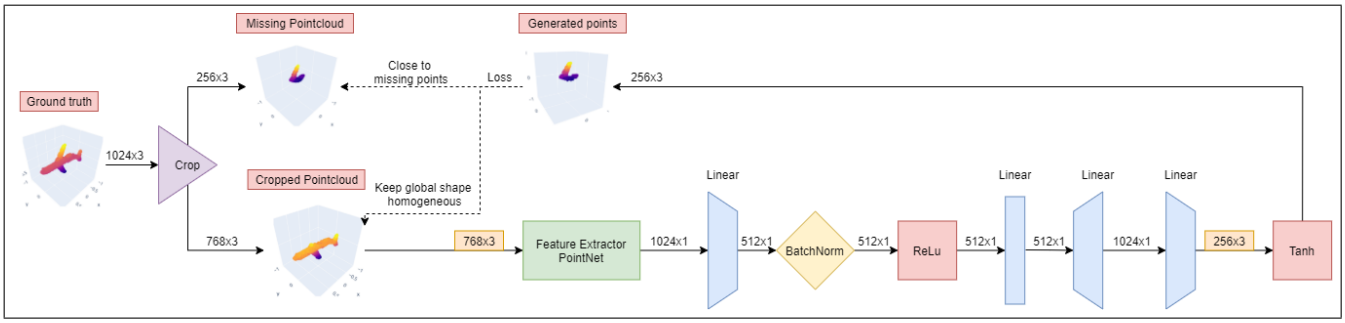

Figure 2. Point-Cloud Completion complete Architecture.

Methods	PNet_AE_128	PNet_AE_512
Encoder	1024-512-256-128	1024-512
Decoder	128-256-512-1024-1024*3	512-1024-1024*3
LR	0,001	0,001
Epoch	40	30
gamma	0,25	0,25
StepSize	20	25
BatchSize	32	32
Testing Loss	0.250	**0.227**

Table 1. PointNet based Auto-Encoder Architectures. The models are trained and tested only on category: *Table*.

Methods	DGCNN_AE_192	DGCNN_AE_512
Encoder	Max-1024-768-192	Max-1024-512
Decoder	192-768-1024-1024*3	512-1024-1024*3
k	20	20
LR	0,001	0,001
Epoch	30	30
gamma	0,5	0,25
StepSize	5	25
BatchSize	12	12
Testing Loss	0.230	**0.228**

Table 2. DGCNN based Auto-Encoder Architectures. The models are trained and tested only on category: *Table*.

4.1. Auto-Encoder

Starting from our structure explained in section 3.2.1 (Figure 1), our experiments are focused on:

- the *Features Extractor*: one based on PointNet and one based on DGCNN
- the *Encoder* and *Decoder* stack: we try different architectures (both symmetrical and asymmetrical) varying also the dimension N of the embedded vector produced by the Encoder.

In Table 1 we present two different PoinNet-based Architectures: *PNet_AE_128* and *PNet_AE_512*.
In Table 2 we present two different DGCNN-based Architectures: *DGCNN_AE_192* and *DGCNN_AE_512*.
The last number indicates the dimension of the embedded vector produced by the Encoder. We performed the hyperparameters tuning using data belonging to only *Table* category of the ShapeNet Dataset.
The reported models are the ones with the best performance among all our attempts. The entire list can be found in the Appendix.
Once we found these models, we trained them considering 7 classes of the ShapeNet Dataset: *Airplane, Chair, Table, Lamp, Car, Motorbike, Mug*. Thus, we considered three scenarios: *Train on all 7-classes jointly*, *Train on single classes* and *Testing on novel categories*.

4.1.1 Train on all 7-classes jointly

In this scenario, the training dataset contains all the data belonging to the *Airplane, Chair, Table, Lamp, Car, Motorbike, Mug* classes.
Considering Table 1 and 2, we choose to train the models with the best performance: *PNet_AE_512, DGCNN_AE_512*. We trained them on 60 epochs. Table 3 shows the results for this scenario in terms of Testing Loss.

4.1.2 Train on single classes

In this scenario, we trained the architectures on data belonging to a single one of the *Airplane, Chair, Table, Lamp, Car, Motorbike, Mug* classes, one at a time. Considering Table 1 and 2, we choose to train the models with the best performance: *PNet_AE_512, DGCNN_AE_512*. We trained them on 30 epochs. Table 4 shows the results for this scenario in terms of Testing Loss.

4.1.3 Testing on novel categories

In this scenario, we tested only two of the models already trained for all 7-classes with novel categories never seen before. The chosen models are the ones with the best performance: *PointNet_AE_512* and *DGCNN_AE_512*.

	Category	PNet_AE_512	DGCNN_AE_512
Known	Airplane	**0.100**	0.105
	Chair	0.191	**0.190**
	Table	**0.207**	0.219
	Lamp	0.301	**0.272**
	Car	**0.245**	0.256
	Motorbike	**0.186**	0.193
	Mug	0.381	**0.364**
	Avg	*0.230*	***0.228***
Similar	Basket	0.711	**0.606**
	Bicycle	0.408	**0.399**
	Bowl	1.072	**0.747**
	Helmet	0.902	**0.732**
	Microphone	1.635	**0.694**
	Rifle	0.201	**0.197**
	Watercraft	0.261	**0.259**
	Avg	*0.741*	***0.516***
Dissimilar	Bookshelf	0.576	**0.551**
	Bottle	0.330	**0.307**
	Clock	0.702	**0.562**
	Microwave	**0.494**	0.517
	Pianoforte	0.732	**0.631**
	Telephone	0.494	**0.424**
	Avg	*0.584*	***0.499***

Table 3. Results for Auto-Encoder with training on all 7 known classes jointly and testing on novel categories classes (both similar and dissimilar).

Category	PtNet_AE_512	DGCNN_AE_512
Airplane	0,113	**0,099**
Chair	0,221	**0,197**
Table	**0,227**	0,228
Lamp	**0,387**	0,499
Car	0,294	**0,254**
Motorbike	**0,225**	0,236
Mug	**0,582**	0,585
Avg	***0,293***	*0,300*

Table 4. Results for Auto-Encoder with training on single classes.

The data belong to:

- **Similar categories**: *Basket, Bicycle, Bowl, Helmet, Microphone, Rifle, Watercraft*
- **Dissimilar categories**: *Bookshelf, Bottle, Clock, Microwave, Pianoforte, Telephone*

Table 3 shows the results on both similar and dissimilar categories.

4.2. Point-cloud Completion

Regarding the Point-Cloud Completion task, we choose to train only the *PNet_AE_512* architecture, as already said in Section 3.3.2, for 30 epochs. Our experiments are focused on the definition of the loss function.
In particular, we tried two different formulations:

- **Vanilla Loss**: Use the loss function defined by equation (3);
- **Matching Loss**: The loss function already defined in Section 3.3.3, equation (7).

	Category	PNet_AE_512	
		Vanilla Loss Model	Matching Loss Model
Known	Airplane	0.103	**0.102**
	Chair	0.181	**0.167**
	Table	**0.180**	0.191
	Lamp	**0.592**	0.639
	Car	**0.181**	0.249
	Motorbike	**0.130**	0.372
	Mug	**0.283**	0.422
	Avg	***0.236***	*0.306*
Similar	Basket	0.264	**0.249**
	Bicycle	**0.155**	0.159
	Bowl	0.474	**0.416**
	Helmet	**0.287**	**0.287**
	Microphone	2.426	**2.168**
	Rifle	0.152	**0.141**
	Watercraft	**0.087**	0.094
	Avg	*0,549*	***0.502***
Dissimilar	Bookshelf	0.239	**0.226**
	Bottle	**0.174**	0.184
	Clock	**0.257**	0.281
	Microwave	0.251	**0.242**
	Pianoforte	**0.168**	0.188
	Telephone	0.252	**0.219**
	Avg	***0,223***	*0.224*

Table 5. Testing Loss results for Point-Cloud Completion (training on all 7 known classes jointly using Vanilla Loss and Matching Loss). Testing Loss: mean Chamfer Distance only on the missing part.

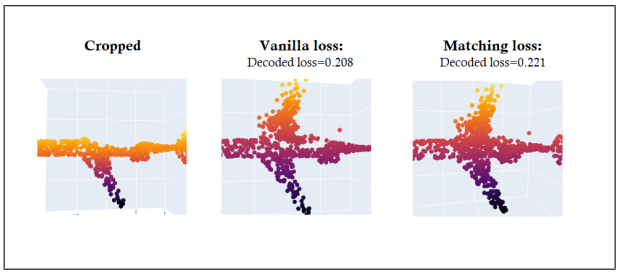

Figure 3. Point-Cloud Completion: Reconstruction of an airplane's top wing with different models.

We trained the model using these loss functions with the *Airplane* data. In Table 5 we report the values of the Testing Loss for both the models trained using Vanilla Loss and Matching Loss. The **Testing Loss** is computed **as the mean Chamfer Distance only on the reconstructed missing part**. The model trained using *Vanilla Loss* provides comparable results to the one trained with *Matching Loss*.
In Figure 3 there is the output of the reconstruction of an airplane's top wing. We can observe an interesting thing: for the model trained using the *Vanilla Loss*, the reconstruction is not good in terms of cohesion of the whole image. The *Reconstructed Missing-Part* and the *Cropped Part* are clearly separated. The model trained with *Matching Loss*, instead, provides a better result for this aspect. Thus, considering these two observations, we decide to use the *Matching Loss* to train the model.

5. Discussion

5.1. AutoEncoder

5.1.1 Training in PointNet based Auto-Encoder

The results obtained considering both training on all 7-classes jointly and on single classes are quite good, as we can see from Tables 3 and 4. In most of the cases, the network can reconstruct at least a general shape of the object. One area in which our model doesn't work so good is the reconstruction of fine details.
We find that our architecture is better for those categories which have a low variance of features (i.e. airplanes), while it performs really bad for categories with many different shapes and unique details (i.e. lamps). We believe that this is due to the fact that the PointNet tends to learn a sort of *"average shape"* of the object. Indeed, our model exploits the latter whenever it encounters features that it is not able to recognize (because it hasn't seen them enough times).
One interesting fact we can observe is that the training done on all 7-classes gives us better results than the one done on single classes.
It could seem strange because, in theory, the model trained on a single category should work better with data belonging to the same category.
A possible interpretation, as mentioned above, can be that the *"average shape"* is actually detrimental for the reconstruction of finer details. By training on all classes jointly we force the network to learn how to recognize more basic *sub-shapes*, because this *"average shape"* doesn't resemble any category.
Doing so, we have two effects: since these *sub-spaces* are simpler, they can be reused on multiple categories and, in the meanwhile, we have much more data for training.
For example, Figure 4 represents a particular table with a shelf. In the case of training on single classes, in particular with a model trained only on tables, the shelf is not present. Instead, looking at the reconstruction done by the model trained on all 7-classes, we have the partial reconstruction of the shelf. We believe that the model has learned this pattern from the Chair category.
Furthermore, looking at the table in Figure 7, we can observe that a Point-Net based Auto-Encoder tends to place points in regions with an high density of them. We think that this is the cause of the loss of detail in the output, since the less dense regions of points correspond to the one containing the finest details of the image (in the case of Figure 7, these regions are the bars under the table and its legs).

5.1.2 Training in DGCNN based Auto-Encoder

Looking at Tables 3 and 4, the results obtained for all 7-classes and for single classes point out the same observations done for the PointNet case.

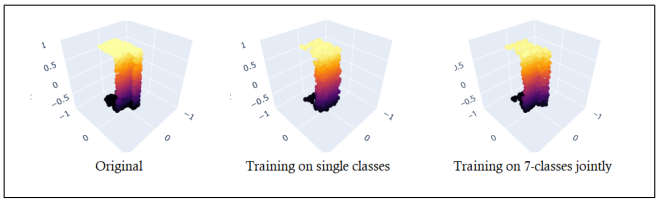

Figure 4. Auto-Encoder: comparison between training on 7-classes jointly vs on single classes.

We can observe that the PointNet-based architecture and the DGCNN-based one have comparable performance. This is different from our expectations, because the DGCNN generally works better in classification tasks as demonstrated by Wang *et al*. [5] since it is able to recognize fine details of the images. It seems that our decoder is not able to use this additional information to reconstruct a more precise object.

5.1.3 Testing on novel categories

Looking at Tables 3, the performance on *similar* and *dissimilar* categories are hard to evaluate in terms of average loss because we have very different results in different categories. Overall, we can make the same observation as above regarding the fact that the network has learned some basic feature for all the categories it has seen and tries to reconstruct the new objects using this knowledge. This sometimes has good results (i.e. basket, bookshelves), sometimes not (i.e. microphone and pianoforte). Further examples are in the Appendix.
For *similar categories*, we can observe that the worst result (microphone) corresponds to the train class that already had poor performance (lamp).

5.2. Point-cloud Completion

Point-Cloud Reconstruction works quite well: it's able to recognize and to complete the missing part. Of course, all the previous shortcomings of the PointNet architecture still applies. In particular, it has an hard time understanding and replicating the concept of *symmetry*.
For example, looking at Figure 3, we can observe that the reconstructed top wing is not exactly symmetrical to the other one. Despite this, it was able to recognize the correct shape of the wing, and this is a good result for us.
For novel categories, the performance drops not mainly in terms of computed loss but in terms of fine grained details quality. The network is able to fill the missing part but, in most of the cases, it completely loses track of the general structure (see Figure 5). This happens due to the fact that the model is trained on the known categories, as we have already seen for the testing on novel categories task for the Auto-Encoder. This problem gets worse when it is focusing only on recreating one part. There are some exceptions

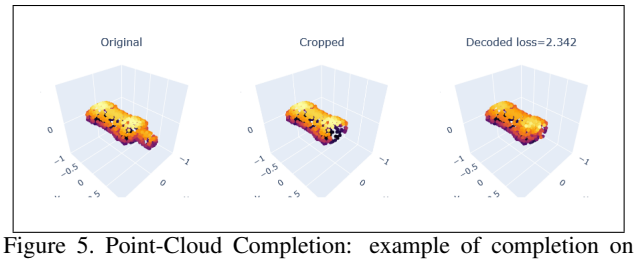

Figure 5. Point-Cloud Completion: example of completion on novel categories - Dissimilar : *Bottle*.

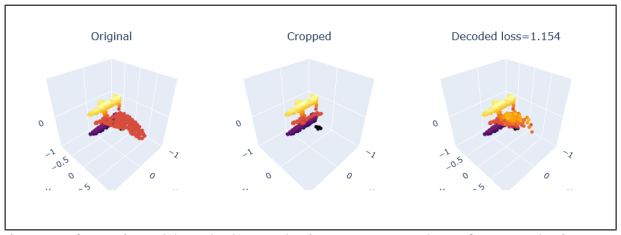

Figure 6. Point-Cloud Completion: example of completion on novel categories - Similar : *Watercraft*.

when the missing part is comparable to some others missing parts present in the training set. As we can see in Figure 6, the sail of the watercraft, which is similar to the wing of a plane, is partially reconstructed even if the object belongs to a novel category.
It seems that the network loses the concepts of "similarity" and "dissimilarity" in terms of the whole objects' categories but it retains them on a lower level of objects' components. This is due to the fact that training Point-Clouds are cropped, so all the training categories are actually "dissimilar" despite having redundant components that can be recognized and reproduced. This explains why we have a lower loss on "dissimilar" categories for Point-Cloud Reconstruction task with respect to the "similar" ones.

6. Final observations and future works

Small Encoder layers The architectures *PointNet_AE_128* and *DGCNN_AE_192* in Tables 1 and 2, despite having a very small embedded vector's size (128 and 192, respectively), are able to reach very good results in reconstruction task. This means that both PointNet and DGCNN are able to extract the main features very memory-efficiently. Working on this property, it could be possible to realize a network that could be able to compress a Point-Cloud.

Limits of Chamfer Distance Loss function Basing on our observations, we believe that Chamfer Distance is not enough to guide the training process. In particular, it has issues when we have regions of different densities in our Point-Cloud. Indeed, it incentives the network to have a negative bias regarding the number of points to put in low-density regions, which usually represent finer object's details. This is due to the *pseudo-symmetric* behaviour of the Chamfer Distance.
In other words, assigning a point to an high-density region will always result with a low Chamfer Distance value for that point. On the other hand, assigning a point to a low-density region and making even a little displacement with respect to the correct position, will result in an higher Chamfer Distance value.
Indeed, looking at Figure 7, we can see that most of the points are based on the flat region of the table and a very low number of points on the legs and bars (the lowest amount to minimize the Chamfer Distance relying to those parts of the Point-Cloud).
Thus, to solve this problem, we should add another term to the loss function in order to enforce the network to keep the local density of the training data.

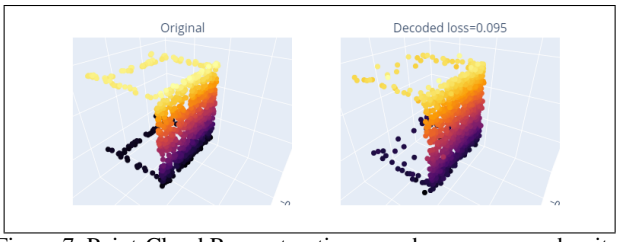

Figure 7. Point-Cloud Reconstruction: non-homogeneous density.

7. Conclusions

With our studies, we showed that both PointNet and DGCNN networks are good starting points to build a working Point-Cloud Auto-Encoder, especially learning the main features of the object in a small embedding space. Of course our work can be improved in order to preserve also the finer details. Future effort should focus on improving the loss function preserving both density and symmetry for Point-Cloud Reconstruction and completion.

References

[1] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[2] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. *CoRR*, abs/1612.00603, 2016.

[3] Z. Huang, Y. Yu, J. Xu, F. Ni, and X. Le. Pf-net: Point fractal network for 3d point cloud completion, 2020.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[5] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

8. Appendix

In this section, we report:

- The list of our attempts to perform the hyper-parameters tuning of the Auto-Encoder architecture. The lists of the architectures are in Tables 6 and 7. Tables 8 and 9 contain the results in terms of Validation Loss and Testing Loss, computed using the loss function defined in equation 3 on the Validation and Test part of the ShapeNet Dataset, respectively. This first step of our work is done considering only data belonging to *Table* category. Then, we train four architectures with the best performance (two for Point-Net and two for DGCNN) on all 7-classes jointly and on single classes. The results are in Table 10 and 11.
- The plots of Training Loss for *PointNet_AE_512* and *DGCNN_AE_512* in the case of training on single classes (Figure 8; Figure 9) and on all 7-classes jointly (Figure 10).
- Examples of output for Point-Cloud Reconstruction using *PointNet_AE_512* and *DGCNN_AE_512* architectures trained on all 7-classes and on single classes (Figure 11; Figure 12), and testing on novel categories (Figure 13; Figure 14; Figure 15; Figure 16).
- Examples of output for Point-Cloud Completion (Figure 17; Figure 18; Figure 19)

Methods	**Encoder Layers**	**Decoder Layers**	**LR**	**Epoch**	**γ**	**StepSize**	**BatchSize**
PNet_AE_100_v1	1024-896-768-512-256-100	100-128-256-512-1024-1024*3	0,001	30	0,5	10	32
PNet_AE_100_v2	1024-768-256-100	100-128-256-512-1024-1024*3	0,001	30	0,5	15	32
PNet_AE_128	1024-512-256-128	128-256-512-1024-1024*3	0,001	40	0,25	20	32
PNet_AE_128_v2	1024-768-256-128	128-256-768-1024-1024*3	0,001	40	0,25	20	32
PNet_AE_192	1024-768-256-192	192-256-768-1024-1024*3	0,001	40	0,5	20	32
PNet_AE_512	1024-512	512-1024-1024*3	0,001	30	0,25	25	64
PtNet_AE_noEnc	No Enc	1024-1024-1024*3	0,001	30	0,5	15	64
PNet_AE_skipC	1024-512-256-128	(128+1024)-2048-2048-2048*3	0,001	30	0,1	20	32
PNet_AE_C64	No Enc-Conv(64,64) in PNet	1024-1024-1024*3	0,001	30	0,1	10	64

Table 6. AutoEncoder: PointNet-based architectures. The *PNet_AE_N* architectures have N as the lower dimension of the Encoder and differ regarding the stack of Encoder's and Decoder's layers. The *PNet_AE_noEnc* doesn't have the Encoder. The *PNet_AE_skipC* has a further connection between the output of the Feature Extractor and the input of the Decoder. The *PNet_AE_C64* contains a further convolutional layer in the PointNet Features Extractor.

Methods	**Encoder Layers**	**Decoder Layers**	**LR**	**Epoch**	**γ**	**StepSize**	**BatchSize**	**k**
DGCNN_AE_192	Max - 1024-768-192	192-768-1024-1024*3	0,001	30	0,5	5	12	20
DGCNN_AE_256	Max - 1024-768-256	256-768-1024-1024*3	0,001	30	0,5	5	12	20
DGCNN_AE_512	Max - 1024-512	512-1024-1024*3	0,001	30	0,25	25	12	20
DGCNN_AE_768	Max - 1024-768	768-1024-1024*3	0,001	30	0,25	25	12	20
DGCNN_AE_noEnc	Max - noEnc	1024-1024-1024*3	0,001	30	0,5	20	12	20
DGCNN_AE_noEnc_v2	Max+Avg - noEnc	2048-2048-1024-1024-1024*3	0,001	30	0,5	20	12	20

Table 7. DGCNN-based AutoEncoder Architectures. The *DGCNN_AE_N* architectures have N as the lower dimension of the Encoder and differ regarding the stack of Encoder's and Decoder's layers. The *DGCNN_AE_noEnc* architectures don't have the Encoder.

Methods	**Validation Loss**	**Test Loss**
PNet_AE_100_v1	0,250	0,332
PNet_AE_100_v2	0,215	0,285
PNet_AE_128	**0,188**	**0,250**
PNet_AE_128_v2	0,196	0,263
PNet_AE_192	0,198	0,260
PNet_AE_512	**0,175**	**0,227**
PNet_AE_noEnc	0,210	0,277
PNet_AE_skipC	0,215	0,287
PNet_AE_C64	0,344	0,422

Table 8. Hyper-parameters tuning for PointNet-based architectures.

Methods	**Validation Loss**	**Test Loss**
DGCNN_AE_192	**0,188**	**0,250**
DGCNN_AE_512	**0,175**	**0,228**
DGCNN_AE_768	0,177	0,232
DGCNN_AE_noEnc	0,180	0,240
DGCNN_AE_256	0,178	0,238
DGCNN_AE_noEnc_v2	0,186	0,247

Table 9. Hyper-parameters tuning for DGCNN-based architectures.

Methods	Testing Loss (10^2)							
	Airplane	Chair	Table	Lamp	Car	Motorbike	Mug	Avg
PointNet_AE_128	0,109	0,217	0,237	0,329	0,257	0,203	0,396	0,250
PointNet_AE_512	**0,100**	0,191	**0,207**	0,301	**0,245**	**0,186**	0,381	0,230
DGCNN_AE_192	0,109	0,205	0,234	0,316	0,258	0,189	0,376	0,241
DGCNN_AE_512	0,105	**0,190**	0,219	**0,272**	0,256	0,193	**0,364**	**0,228**

Table 10. Auto-Encoder: Training of the best architectures on all 7-classes jointly.

Methods	Testing Loss (10^2)							
	Airplane	Chair	Table	Lamp	Car	Motorbike	Mug	Avg
PointNet_AE_128	0,116	0,264	0,250	0,443	0,273	0,216	0,538	0,300
PointNet_AE_512	0,113	0,221	**0,227**	**0,387**	0,294	**0,225**	**0,582**	**0,293**
DGCNN_AE_192	0,099	0,212	0,250	0,440	0,241	0,207	0,436	0,269
DGCNN_AE_512	**0,099**	**0,197**	0,228	0,499	**0,254**	0,236	0,585	0,300

Table 11. Auto-Encoder: Training of the best architectures on single classes.

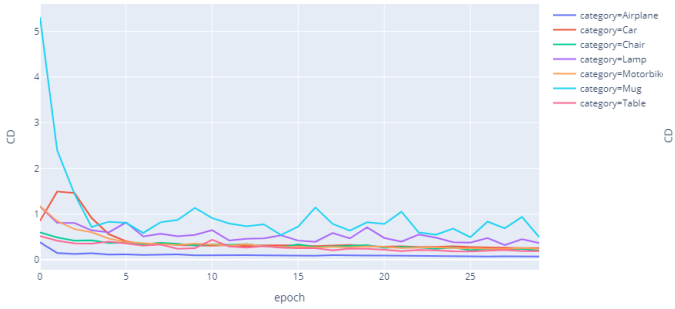

Figure 8. *PointNet_AE_512*: Plot of Training Loss on single classes.

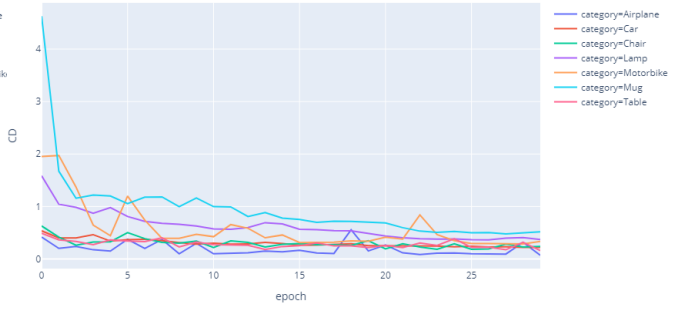

Figure 9. *DGCNN_AE_512*: Plot of Training Loss on single classes.

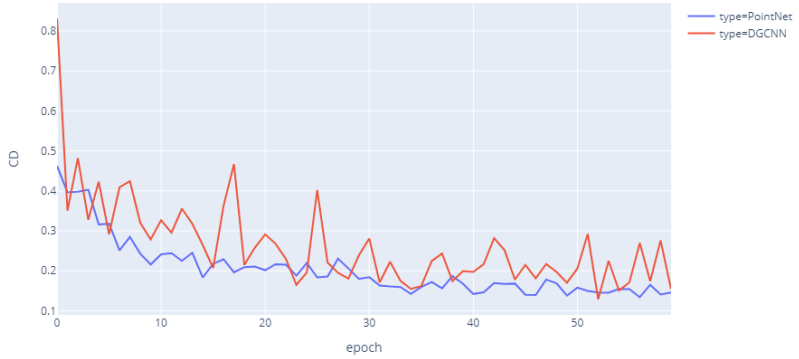

Figure 10. *PointNet_AE_512* and *DGCNN_AE_512*: Plot of Training Loss on all 7-classes jointly.

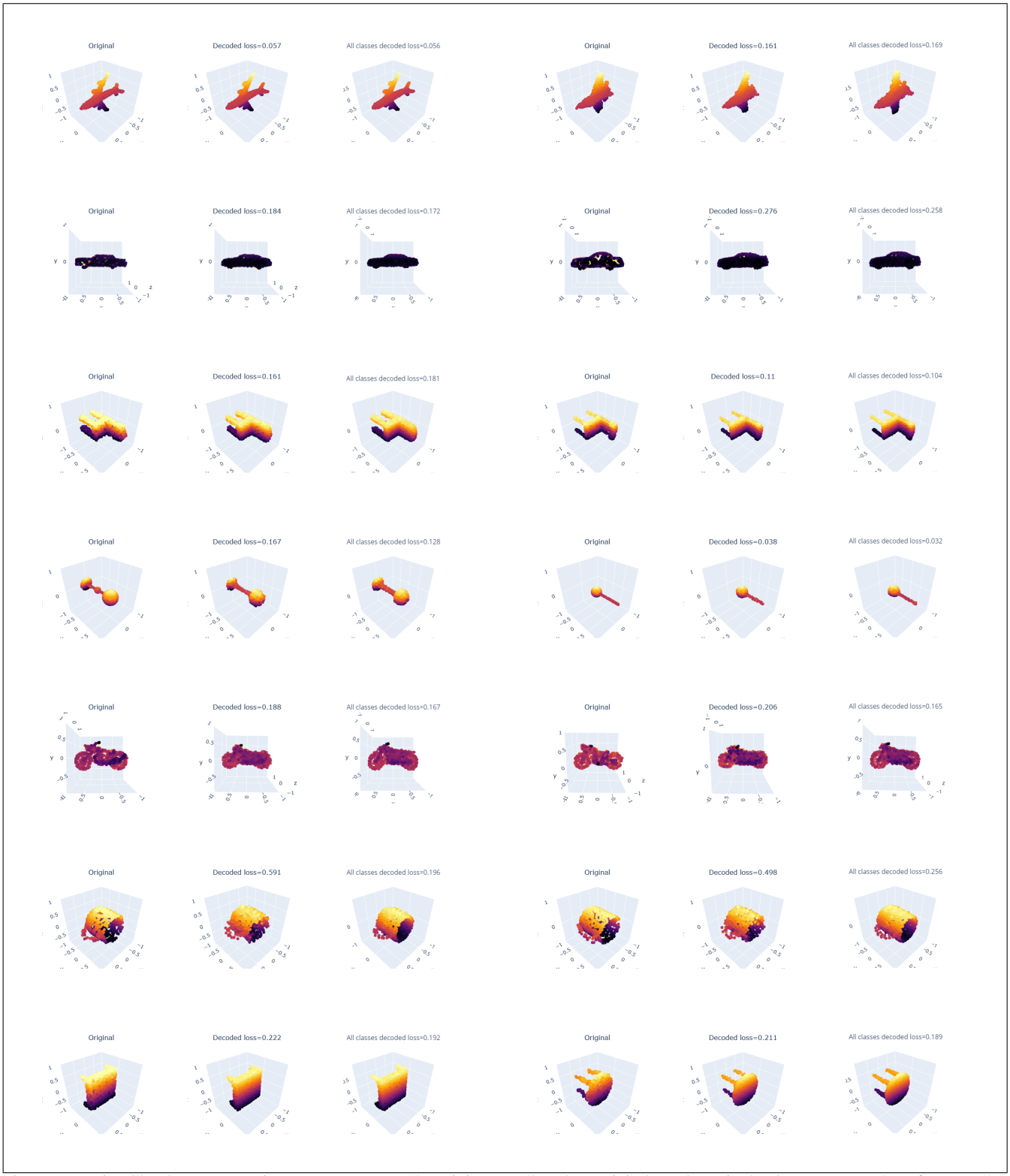

Figure 11. Point-Cloud Reconstruction: *PointNet_AE_512*, training on all 7-classes jointly and on single classes. Examples of output (2 per categories, for each row). Each example is in the form [Input, training on single classes, training on all 7-classes jointly]. In the order: *Airplane, Car, Chair, Lamp, Motorbike, Mug, Table.*

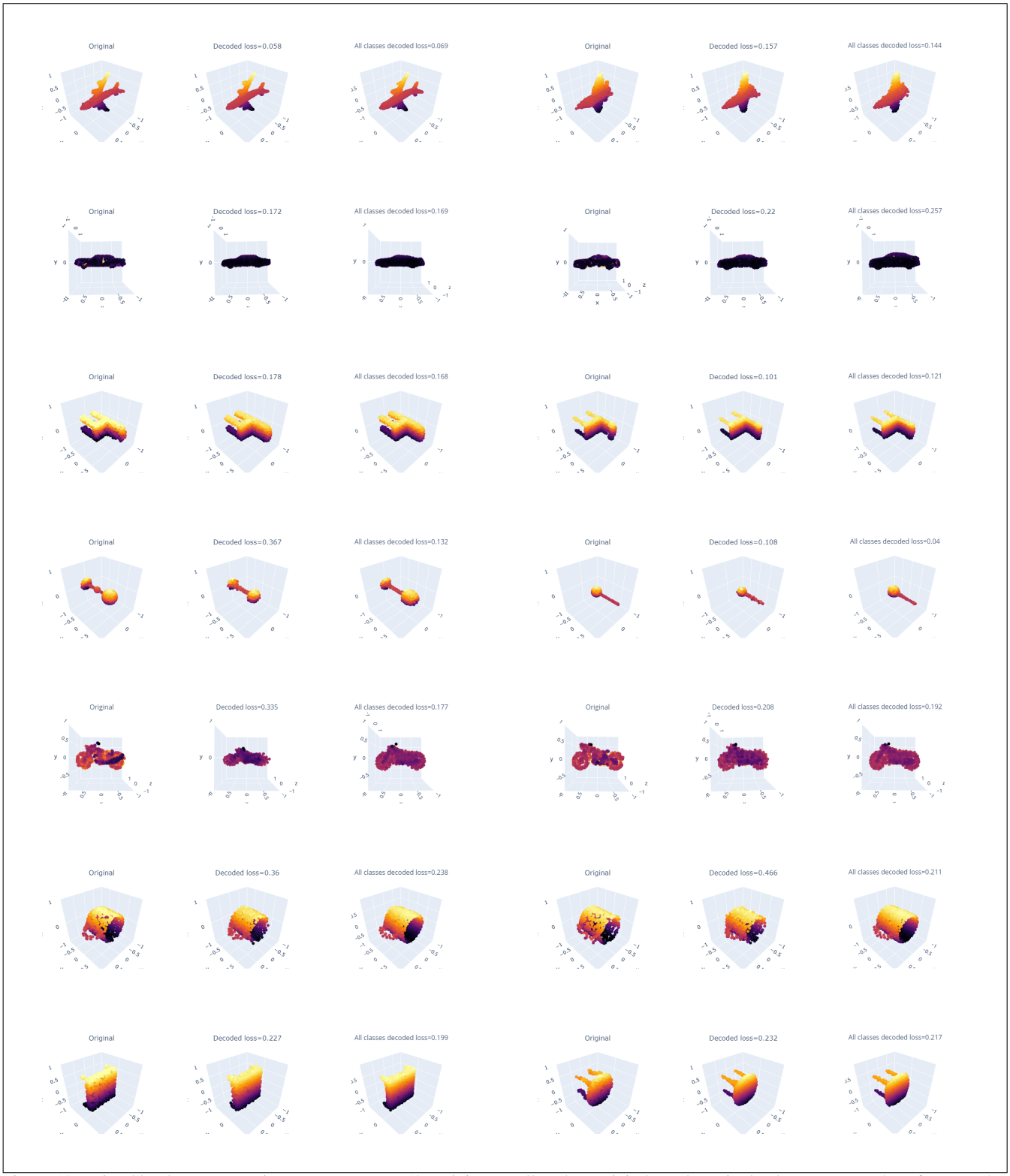

Figure 12. Point-Cloud Reconstruction: *DGCNN_AE_512*, training on all 7-classes jointly and on single classes. Examples of output (2 per categories, for each row). Each example is in the form [Input, training on single classes, training on all 7-classes jointly]. In the order: *Airplane, Car, Chair, Lamp, Motorbike, Mug, Table.*

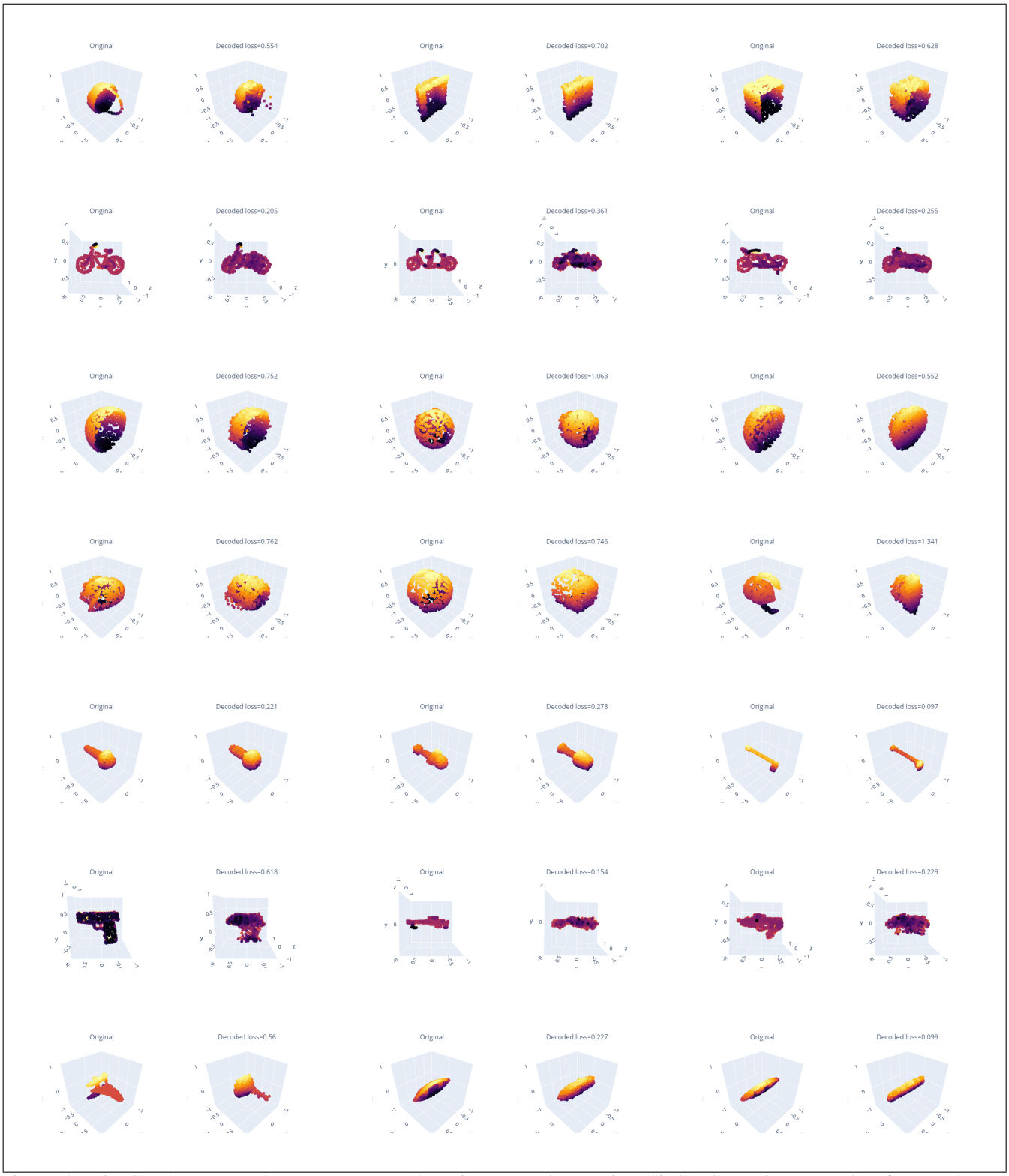

Figure 13. Point-Cloud Reconstruction: *PointNet_AE_512*, testing on novel categories - Similar Categories. Examples of output (3 per categories, for each row). In the order: *Basket, Bicycle, Bowl, Helmet, Microphone, Rifle, Watercraft.*

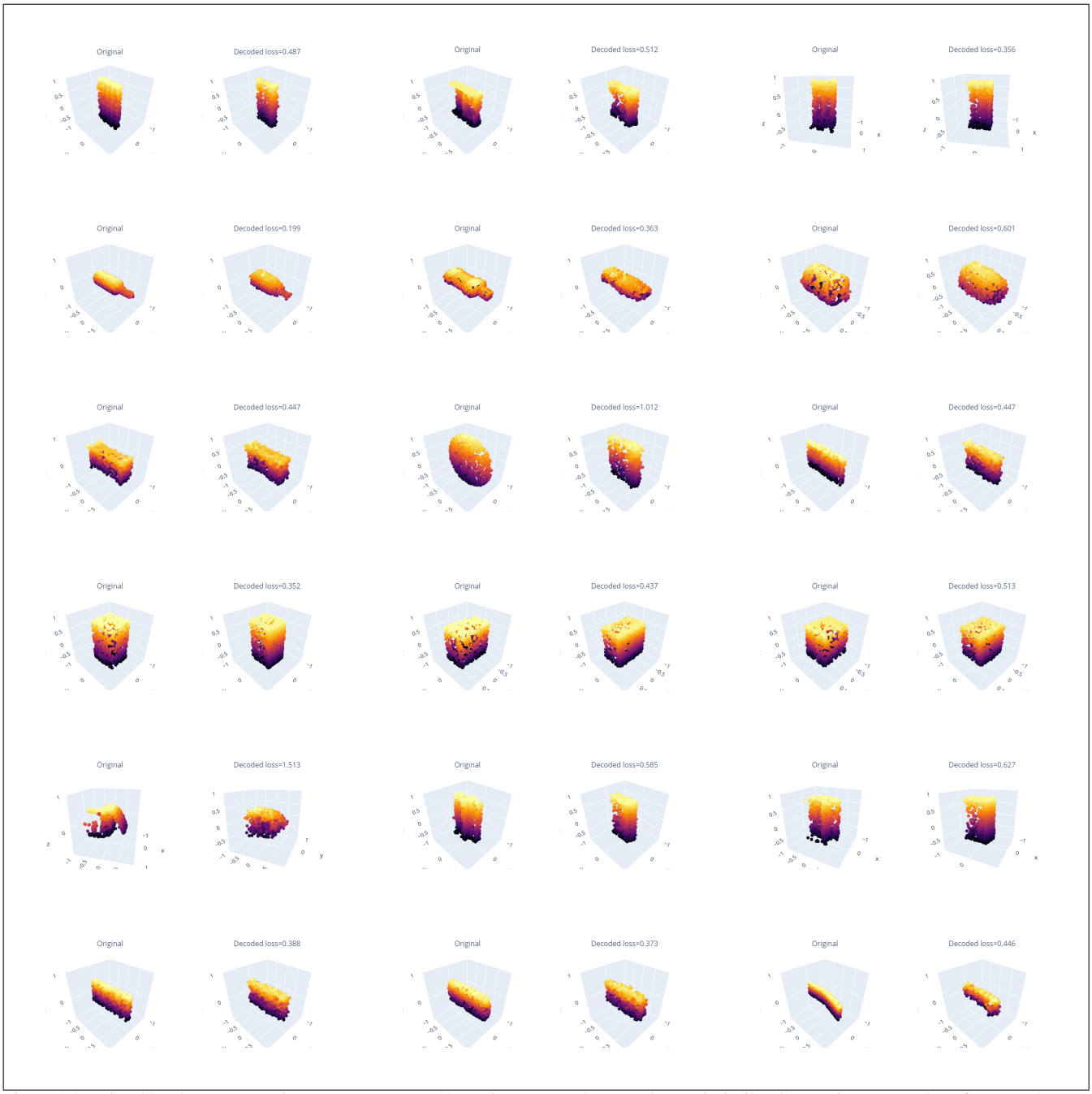

Figure 14. Point-Cloud Reconstruction: *PointNet_AE_512*, testing on novel categories - Dissimilar Categories. Examples of output (3 per categories, for each row). In the order: *Bookshelf, Bottle, Clock, Microwave, Pianoforte, Telephone.*

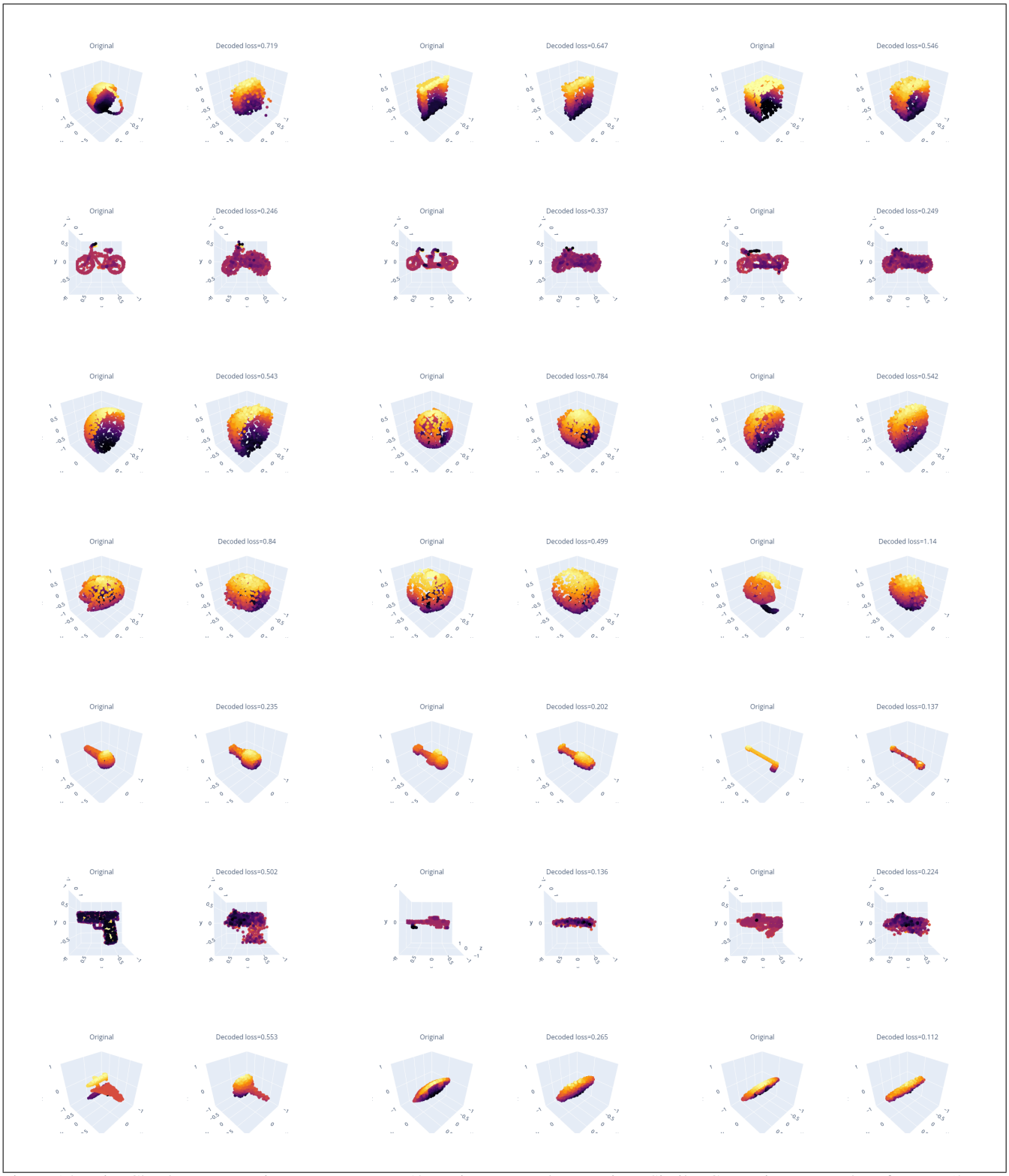

Figure 15. Point-Cloud Reconstruction: *DGCNN_AE_512*, testing on novel categories - Similar Categories. Examples of output (3 per categories, for each row). In the order: *Basket, Bicycle, Bowl, Helmet, Microphone, Rifle, Watercraft.*

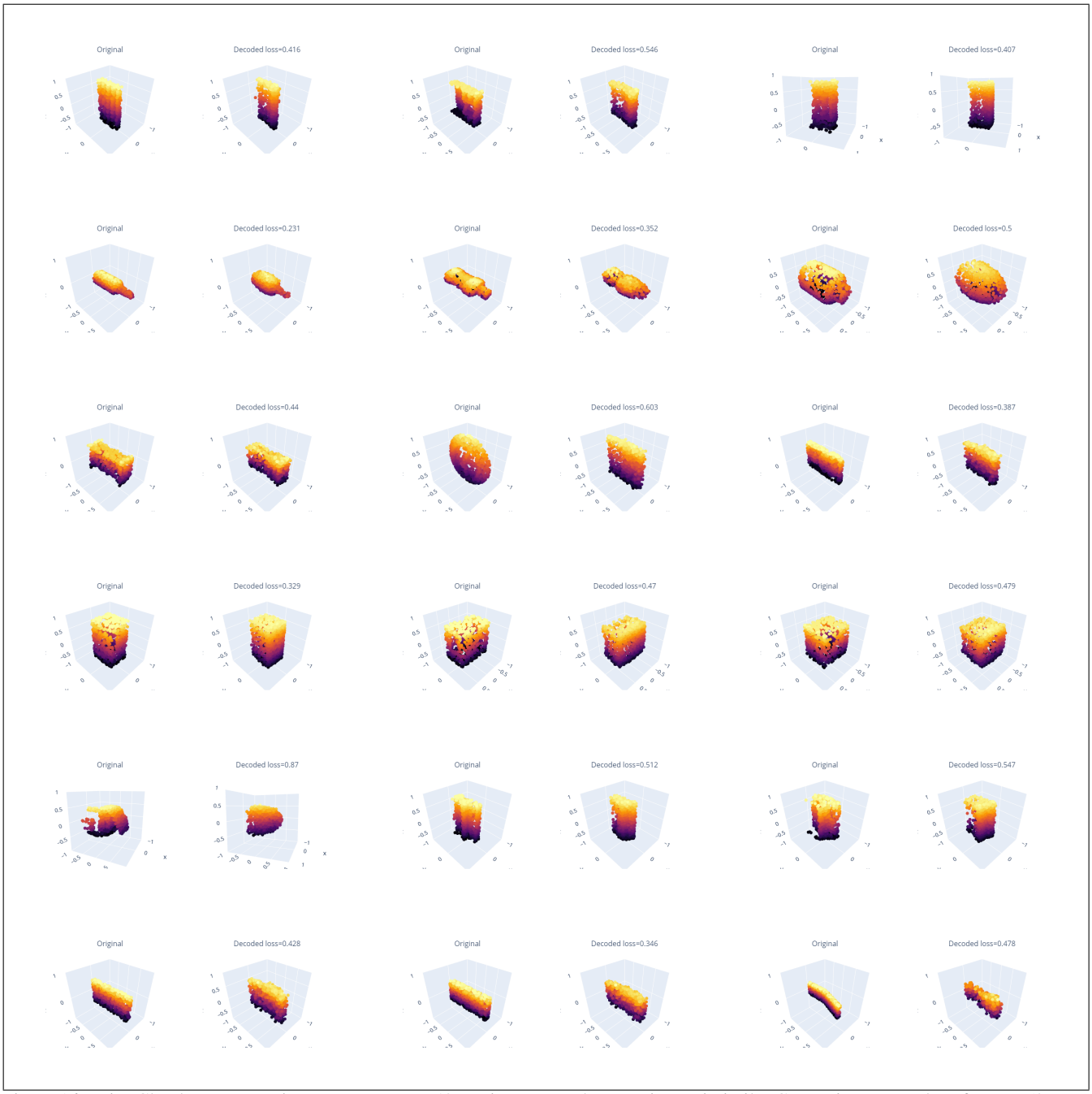

Figure 16. Point-Cloud Reconstruction: *DGCNN_AE_512*, testing on novel categories - Dissimilar Categories. Examples of output (3 per categories, for each row). In the order: *Bookshelf, Bottle, Clock, Microwave, Pianoforte, Telephone.*

Figure 17. Point-Cloud Completion. Examples of output. Two input Point-Clouds that are cropped in 7 different ways using the viewpoints defined in Section 3.3.1

Figure 18. Point-Cloud Completion on similar categories: examples of output.

Figure 19. Point-Cloud Completion on dissimilar categories: examples of output.