Learning to Sample

Oren Dovrat*
Tel-Aviv University
orendovrat@mail.tau.ac.il

Itai Lang*
Tel-Aviv University
itailang@mail.tau.ac.il

Shai Avidan
Tel-Aviv University
avidan@eng.tau.ac.il

Abstract

Processing large point clouds is a challenging task. Therefore, the data is often sampled to a size that can be processed more easily. The question is how to sample the data? A popular sampling technique is Farthest Point Sampling (FPS). However, FPS is agnostic to a downstream application (classification, retrieval, etc.). The underlying assumption seems to be that minimizing the farthest point distance, as done by FPS, is a good proxy to other objective functions.

We show that it is better to learn how to sample. To do that, we propose a deep network to simplify 3D point clouds. The network, termed S-NET, takes a point cloud and produces a smaller point cloud that is optimized for a particular task. The simplified point cloud is not guaranteed to be a subset of the original point cloud. Therefore, we match it to a subset of the original points in a post-processing step. We contrast our approach with FPS by experimenting on two standard data sets and show significantly better results for a variety of applications. Our code is publicly available[1]

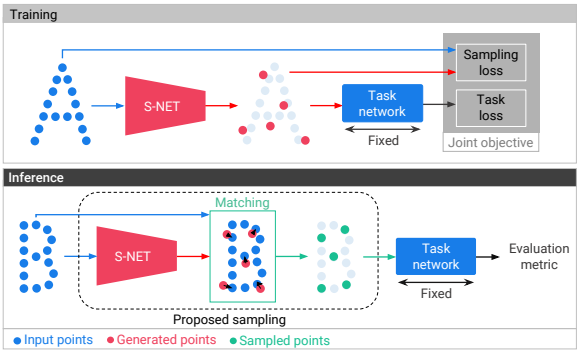


Figure 1. **An illustration of the proposed learned sampling approach.** In the *training* phase, S-NET generates points that are passed to a task network, which was pre-trained and is held fixed. The minimization objective contains the task's loss and a sampling loss. The latter serves as a regularizer and encourages proximity between the input and generated points. At *inference* time, we match the points generated by S-NET with the input point cloud and get a subset of it. Only these points are then fed to the task network for performance evaluation.

1. Introduction

Capturing 3D data is getting easier in recent years and there is a growing number of 3D shape repositories available online. This data can be represented in a variety of ways, including point clouds, multi-view images and voxel grids. A point cloud contains information only about the surface of a 3D object, while a grid based representation also holds data about free space, making the former much more efficient. However, processing a point cloud can be challenging, since it may contain a lot of data points. Reducing the number of points can be beneficial in many aspects, such as reduction of power consumption, computational cost and communication load, to name a few.

One naive approach to reduce the data load is to randomly sample a subset of points. Another approach, which is commonly used in the literature, is Farthest Point Sampling (FPS) [27, 28, 21]. This sampling method takes into account the structure of the point cloud and selects a group of points that are farthest apart from each other [6, 24]. These sampling methods, as well as other approaches in the literature [3, 18], operate according to a non-learned predetermined rule.

In the last few years, deep learning techniques have been applied with great success to point cloud data. Among various applications one can find point cloud classification [27, 28, 21, 42, 30, 36], part segmentation [27, 28, 21, 30, 36, 20], semantic segmentation [27, 21, 36, 34, 32, 12] and retrieval [35, 17]. Other techniques perform point cloud auto-encoding [1, 38, 8], generation [1, 33, 19], completion [1, 2, 41] and up-sampling [40, 39, 43]. Yet a learned point clouds sampling approach, subject to a subsequent task objective, has not been proposed before.

We propose a simplification network, termed S-NET, that is based on the architecture of PointNet [27]. S-NET

[1] https://github.com/orendv/learning_to_sample
*Equal contribution

learns to generate a smaller (simplified) point cloud that is optimized for a downstream task, such as classification, retrieval or reconstruction.

The simplified point cloud must balance two conflicting constraints. On the one hand, we would like it to preserve similarity to the original shape. On the other hand, we wish to optimize it to a subsequent task. We solve this by training the network to generate a set of points that satisfy two objectives: a sampling loss and the task's loss. The sampling loss drives the generated points close to the input point cloud. The task loss ensures that the points are optimal for the task.

An advantages of FPS is that it samples a subset of the original points. In contrast, the simplified point cloud produced by S-NET is not guaranteed to be a subset of the input point cloud. To address this issue, we perform a post-processing step at *inference* time, where we match the generated points with the input point cloud and obtain a subset of it, i.e., a set of sampled points (see Figure 1). Experiments show that better results for several tasks are achieved using our sampled points in comparison to FPS.

Our approach can be thought of as a feature selection mechanism [9, 16]. Each point is a feature of the underlying shape and we seek to select the ones that contribute the most to the task. It can also be interpreted as a form of visual attention [22, 14], focusing the subsequent task network on the significant points.

S-NET is trained to output a fixed sample size, which means that we need to train a different S-NET for every target size. To overcome this limitation, we introduce an extension of S-NET termed ProgressiveNet. ProgressiveNet orders points by importance to the task. This lets the sample size to be chosen at inference time, allowing for a dynamic level-of-detail management, according to the requirements and available resources.

The proposed sampling approach is applied to three different tasks: point cloud classification, retrieval and reconstruction. We compare our approach with common non-data driven methods: random sampling and FPS. For the first task we show better classification accuracy; in the second we show improved retrieval results; and in the last we get a lower reconstruction error. To summarize, our key contributions are:

- A task-specific data-driven sampling approach for point clouds;
- A Progressive sampling method that orders points according to their relevance for the task;
- Improved performance for point cloud classification, retrieval and reconstruction with sampled point clouds.

2. Related work

Point cloud simplification and sampling Several techniques for either point cloud simplification [25, 23] or sampling [15, 5] have been proposed in the literature. Pauly *et al*. [25] presented and analyzed several simplification methods for point-sampled surfaces, including: clustering methods, iterative simplification and particle simulation. The simplified point set, resulting from these algorithms, was not restricted to be a subset of the original one. Farthest point sampling was adopted in the work of Moenning and Dodgson [23] as a means to simplify point clouds of geometric shapes, in a uniform as well as feature-sensitive manner.

Katz and Tal [15] suggested a view dependent algorithm to reduce the number of points. They used hidden-point removal and target-point occlusion operators in order to improve a human comprehension of the sampled point set. Recently, Chen *et al*. [5] employed graph-based filters to extract per point features. Points that preserve specific information are likely to be selected by a their sampling strategy. The desired information is assumed to be beneficial to a subsequent application.

The above sampling approaches aim to optimize a variety of sampling objectives. However, they do not consider *directly* the objective of the task to be followed.

Progressive simplification In a seminal paper, Hoppe [11] proposed a technique for progressive mesh simplification. In each step of his method, one edge is collapsed such that minimal geometric distortion is introduced.

A recent work by Hanocka *et al*. [10] suggested a neural network that performs task-driven mesh simplification. Their network relies on the edges between the mesh vertices. This information is not available for point clouds.

Several researchers studied the topic of point set compression [26, 13, 29]. An octree data structure was used for progressive encoding of the point cloud. The objective of the compression process was low distortion error.

Deep learning on point sets The pioneering work of Qi *et al*. [27] presented PointNet, the first neural network that operates directly on unordered point cloud data. They constructed their network from per-point multi-layer perceptrons, a symmetric pooling operation and several fully connected layers. PointNet was employed for classification and segmentation tasks and showed impressive results. For assessing the applicability of PointNet for reduced number of input points, they used random sampling and FPS. In our work, we suggest a data-driven sampling approach, that improves the classification performance with sampled sets in comparison to these sampling methods.

Later on, Qi *et al*. extended their network architecture for hierarchical feature learning [28]. In the training phase, centroid points for local feature aggregation were selected by FPS. Similar to their previous work [27], FPS was used for evaluating the ability of their network to operate on fewer input points.

Li *et al*. [20] suggested to learn the centroid points for feature aggregation by a self-organizing map (SOM). They used feature propagation between points and SOM nodes and showed improved results for point cloud classification and part segmentation. The SOM was optimized separately, as a pre-processing step.

Building on the work of Qi *et al*. [27], Achlioptas *et al*. [1] developed autoencoders and generative adversarial networks for point clouds. Instead of shape class or per-point label, the output of their network was a set of 3D points. In this work we apply our sampling approach for the task of point cloud reconstruction with the autoencoder proposed by Achlioptas *et al*.

Several researchers tackled the problem of point cloud consolidation. Yu *et al*. [40] extracted point clouds from geodesic patches of mesh models. They randomly sampled the point sets and trained a network to reconstruct the original patch points. Their follow-up work [39] incorporated edge information to improve the reconstruction accuracy. Zhang *et al*. [43] studied the influence of sampling strategy on point cloud up-sampling. They used Monte-Carlo random sampling and curvature based sampling. Their network was trained to produce a fixed size point cloud from its sample. In contrast to these works, our study focuses on the down-sampling strategy.

3. Method

Problem statement Given a point set $P = \{p_i \in \mathbb{R}^3, i = 1, \ldots, n\}$, a sample size $k \leq n$ and a task network T, find a subset S^* of k points that minimizes the task network's objective function f:

$$S^* = \underset{S}{\operatorname{argmin}} f(T(S)), \quad S \subset P, \quad |S| = k \leq n. \tag{1}$$

This problem poses a challenge, as sampling might seems akin to pooling, yet in pooling the pooled value is propagated forward, so the gradient with respect to it can be calculated. Discrete sampling, however, is like "arg-pooling", where the propagated value cannot be updated incrementally. As a result, a sampling operation cannot be trained directly. Therefore, we propose a two-step process: first, we employ a neural network, i.e., S-NET, to generate a set of points. Second, we match the *generated* points with the input point cloud to obtain a subset of its points, i.e., the *sampled* points. Figure 1 illustrates the process.

The input to S-NET is a set of n 3D coordinates, namely points, representing a 3D shape. The output of S-NET is k generated points. S-NET is followed by a task network. The task network is pre-trained on an input of n points, to perform a given task on the point cloud (i.e., classification, retrieval or reconstruction). It is kept fixed during training and testing of S-NET. This ensures that sampling is being optimized to the task, rather than the task being optimized to an arbitrary sampling.

At the training phase, the generated points are fed to the task network. The points are optimized to the task at hand by minimizing the task loss. We use an additional sampling regularization loss term, that encourages each of the generated points to be close to one of the input points and forces the generated points to spread over the input cloud.

At inference, the generated points are matched with the input point cloud in order to obtain a subset of it. These are the sampled points, the final output of our process. These points are passed through the task network and its performance is evaluated.

We present two sampling versions: S-NET and ProgressiveNet. In the first version (Figure 1), we train a different sampling network per sample size. In the second one (Figure 2), we train one network that can be used to produce any sample size smaller than the input size.

3.1. S-NET

The architecture of S-NET follows that of Qi *et al*. [27]. The input points undergo a set of 1×1 convolution layers, resulting in a per point feature vector. Then, a symmetric feature-wise max pooling operation is used to obtain a global feature vector. Finally, we use several fully-connected layers. The output of the last layer is the set of generated points.

Let us denote the generated point set as G and the input point set as P. We construct a sampling regularization loss, composed out of three terms:

$$L_f(G, P) = \frac{1}{|G|} \sum_{g \in G} \min_{p \in P} ||g - p||_2^2 \tag{2}$$

$$L_m(G, P) = \max_{g \in G} \min_{p \in P} ||g - p||_2^2 \tag{3}$$

$$L_b(G, P) = \frac{1}{|P|} \sum_{p \in P} \min_{g \in G} ||p - g||_2^2. \tag{4}$$

L_f and L_m keeps the points in G close to those in P, in the average and worst case, respectively. This is designed to encourage tight matches in the following matching process. We found that mixing average and maximum operations speeds up convergence. L_b ensures that the generated points are well spread over the input points, decreasing the number of collisions in the matching process. The sampling regularization loss is a weighted sum of these three terms:

$$\begin{aligned} L_s(G, P) = L_f(G, P) + \beta L_m(G, P) \\ +(\gamma + \delta|G|)L_b(G, P). \end{aligned} \tag{5}$$

Note that this is a generalization of the Chamfer distance [7], achieved when $\beta = 0$, $\gamma = 1$ and $\delta = 0$.

In addition, we denote L_{task} as the task network loss. The total S-NET loss is:

$$L^{S-NET}(G, P) = L_{task}(G) + \alpha L_s(G, P) \tag{6}$$

where α controls the regularization trade-off. The output of S-NET is a $k \times 3$ matrix, where k is the sample size, and we train separately for each k.

3.2. Matching

The generated points G are not guaranteed to be a subset of the input points P. In order to get a subset of the input points, we match the generated points to the input point cloud.

A widely used approach for matching two point sets is the Earth Mover's Distance (EMD) [1, 41, 43, 7]. EMD finds a bijection between the sets that minimizes the average distance of corresponding points, while the point sets are required to have the same size. In our case, however, G and P are of different size.

We examine two matching methods. The first adapts EMD to uneven point sets. The second is based on nearest neighbour (NN) matching. Here we describe the latter, which yielded better results. The reader is referred to the supplementary material for details about the other matching method.

In NN-based matching, each point $x \in G$ is replaced with its closest euclidean corresponding point $y^* \in P$:

$$y^* = \underset{y \in P}{\operatorname{argmin}} ||x - y||_2. \tag{7}$$

Since several points in G might be closest to the same point in P, the number of unique sampled points might be smaller than the requested sample size. Therefore, we remove duplicate points and get an initial sampled set. Then, we complete this set, up to the size of G, by running farthest point sampling (FPS) [28], where in each step we add a point from P that is farthest from the current sampled set.

The matching process is only applied at inference time, as the final step of inference. During training, the generated points are processed by the task network as-is, since the matching is not differentiable and cannot propagate the task loss back to S-NET .

3.3. ProgressiveNet: sampling as ordering

S-NET is trained to sample the points to a single, predefined, sample size. If more than one sample size is required, more than one S-NET needs to be trained. But what if we want to train one network that can produce any sample size, i.e., sample the input at any sampling ratio? To this end we present ProgressiveNet. ProgressiveNet is trained to take a point cloud of a given size and return a point cloud of the same size, consisting of the same points. But, while the points of the input are arbitrarily ordered, the points of the output are ordered by their relevance to the task. This allows sampling to any sample size: to get a sample of size k, we simply take the first k points of the output point cloud of ProgressiveNet and discard the rest. The architecture of ProgressiveNet is the same as S-NET , with the last fully connected layer size equal to the input point cloud size (has $3n$ elements).

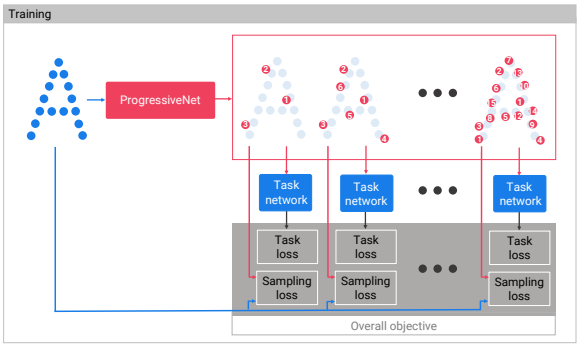


Figure 2. **ProgressiveNet training.** The generated points are divided into groups of increasing size, such that each group is a subset of the following larger group. Each group has corresponding task and sampling losses. The overall objective is the sum of the per-group losses. The task network was pre-trained and is kept fixed.

To train ProgressiveNet (Figure 2) we define a set of sizes $C_s = \{2^1, 2^2, ..., 2^{log_2(n)}\}$. For each size $c \in C_s$ we compute a task loss term and a sampling regularization loss term, such that the total ProgressiveNet's loss becomes:

$$L^{ProgressiveNet}(G, P) = \sum_{c \in C_s} L^{S-NET}(G_c, P)) \tag{8}$$

where L^{S-NET} is the loss term as defined in equation 6 and G_c are the first c points from the points generated by ProgressiveNet (the first $3c$ elements of the output layer). This loss function defines a nested structure of point subsets. For example, the first subset, consisting of two points, is used in all terms $L^{S-NET}(G_c, P)$, for $c \geq 2$. Because this subset is used in all terms, it is contained in all larger subsets. Similarly, the first 4 points define the second subset, that includes the first subset and is part of all larger subsets.

Under this training process, the first k generated points (for any k) are optimized to be suitable both for the task at hand as a stand-alone set and for integration with their preceding points to create a larger set that will improve results on the given task. This makes sure that a point that is more important for the task will appear earlier in the generated point set, while extra points will give diminishing marginal utility, resulting in a a task-oriented progressive decomposition of the point cloud [31].

At inference, the generated points (ProgressiveNet's output layer) are matched with the input point cloud using the same matching process we used in Section 3.2 for S-NET. We transfer the order of the generated points to their matched points. To obtain a specific sample size k, we take the first k sampled points.

4. Results

We applied S-NET to three tasks: point set classification [27], retrieval and reconstruction [1]. For classification and retrieval we adopt the ModelNet40 [37] point cloud data provided by Qi *et al*. [27] and PointNet [27] as the task network. For reconstruction we employ ShapeNet Core55 repository [4] and the point set autoencoder of Achlioptas *et al*. [1].

Random sampling and FPS are employed as alternative non-data driven sampling approaches for comparison with our suggested approach. In order to make a fair comparison, we only use S-NET points after the matching process, so both our sampled points and the alternative approaches' points are subsets of the input point cloud. Further experimental details can be found in the supplementary material.

4.1. Classification

We report instance classification results on the ModelNet40 data set, adopting the official train-test split. We employed the full version of PointNet as the task network for S-NET and the vanilla version of PointNet for ProgressiveNet (to save training time), except where otherwise noted.

S-NET Figure 3 shows the classification accuracy of PointNet, when trained on the complete data (1024 points per point cloud) and tested on samples of different size. We compare several different sampling methods: random sampling with uniform distribution over the input points, FPS and S-NET. We trained 10 different S-NETs, for sample sizes of $k \in \{2, 4, 8, \ldots, 1024\}$ points. The sampling ratio is defined as $n/k = 1024/k$. We observe that the classification accuracy using S-NET's sampled points is equal or better than that of using FPS's points for any sampling ratio, with a margin of up to 34.2% (for sampling ratio 32).

Table 1 shows that the accuracy of PointNet is also higher when *training* it on the points sampled by S-NET. Each of the numbers in this table represents a different PointNet, each trained and tested on point clouds of a specific size that was sampled with a different sampling method. We see, for example, that PointNet that was both trained and tested on point clouds of just 16 points, achieved 85.6% accuracy, when using S-NET, compared to only 76.7% with FPS. This shows that S-NET is not overfitted to the specific instance of the classifier it was trained with. Instead, it samples the points in a way that makes the sampled set easy to classify, creating a strong distinction between different classes in the data.

ProgressiveNet In Figure 4 we compare the accuracy of PointNet vanilla on points sampled by S-NET (trained with PointNet vanilla, in this case) to those sampled by ProgressiveNet. The evaluation was done for all sample sizes in the range $[2, 1024]$. S-NET results are from 10 different S-NETs, each trained for a specific sample size, for sizes of $k \in \{2, 4, 8, \ldots, 1024\}$ points. For the sample sizes in between those values, we took the S-NET that was trained for a lower sample size and then completed with FPS to the required size. For example, to get a sample of size 48, we took the points sampled by S-NET that was trained to sample 32 points, and then made 16 steps of FPS. The Progressive results are from one ProgressiveNet with output size 1024, that was trained with classification and sampling loss terms for sizes $C_s = \{2, 4, 8, \ldots, 1024\}$.

We observe that S-NET has better performance for the sample sizes it was trained for, while ProgressiveNet performs better for any sample size in between. This shows the advantage of ProgressiveNet, which orders the points by priority, so that the accuracy is approximately monotonically increasing in the sample size.

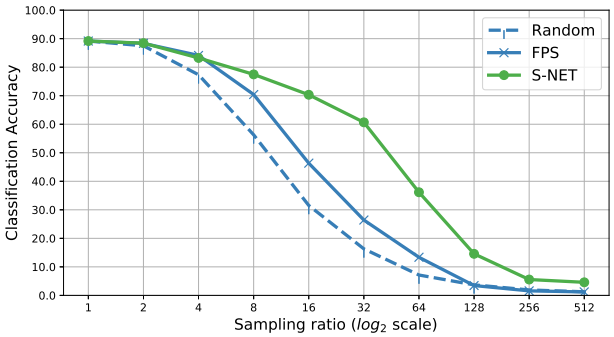


Figure 3. **S-NET for classification.** PointNet was trained on complete point clouds (1024 points) and evaluated on sampled point clouds of the test set using different sampling methods: random, FPS, and S-NET. The accuracy using S-NET is evidently higher.

#Sampled points	Random	FPS	S-NET
1024	89.2	89.2	89.2
512	88.2	**88.3**	87.8
256	86.6	88.1	**88.3**
128	86.2	87.9	**88.6**
64	81.5	86.1	**87.7**
32	77.0	82.2	**87.3**
16	65.8	76.7	**85.6**
8	45.8	61.6	**83.6**
4	26.9	35.2	**73.4**
2	16.6	18.3	**53.0**

Table 1. **Training PointNet classifier on sampled point clouds.** We sample the train and test data with different sampling methods: random sampling, FPS, and S-NET. Afterwards, PointNet is trained and evaluated on the sampled data. Applying S-NET allows good classification even with minimal data to train on. We conclude that S-NET transforms the data into a more separable representation.

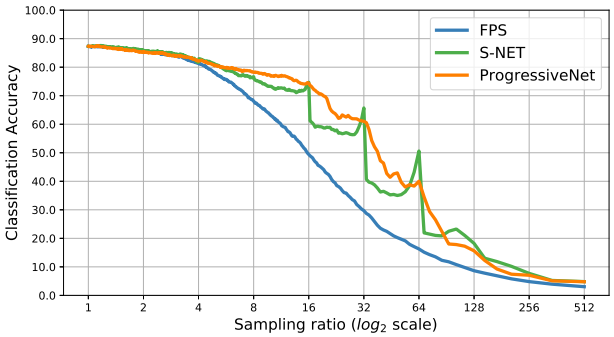


Figure 4. **ProgressiveNet vs. S-NET.** PointNet vanilla was trained on complete point clouds (1024 points) and evaluated on every sample size k in the range $[2, 1024]$. The sampling ratio is defined as $1024/k$. We compared three different sampling methods: FPS, S-NET and ProgressiveNet. S-NET is better for the sample sizes it was trained for, while ProgressiveNet performs better for sample sizes in between.

Scalability by S-NET S-NET assumes that the task network is already trained. This will cause a problem in case of very large point clouds. We show that it is possible to train the task network on FPS-sampled point clouds and use that to train S-NET . The resulting S-NET can then be used to re-train the task network to a higher accuracy. Please see Figure 5 for an illustration of the proposed training and testing procedure.

The following small scale simulation demonstrate this: We used FPS to sample ModelNet40 to $k = 32$ points per point cloud, and trained PointNet on those sampled points. This is our baseline. The accuracy of the baseline on the test set is 82.2%. When we fed this trained PointNet with point clouds of size 1024, the accuracy was just 46.6%. We then trained S-NET , using the 32-sized point clouds training set, and the baseline PointNet as the task network. Then we sampled ModleNet40 again to size $k = 32$, this time using S-NET. Finally, we trained PointNet on the points sampled by S-NET. The accuracy of the re-trained PointNet on the test set improved to 86.0%. Employing S-NET allows us to improve PointNet's accuracy without training on larger point clouds.

Time and space considerations The time complexity of PointNet-like networks is dominated by their per-point convolution layers, and thus is strongly dependent on the size of the input point cloud. The space complexity of S-NET is a linear function of its output size k. S-NET offers a trade-off between space (number of parameters) and inference time (number of floating point operations). For example, cascading S-NET that samples a point cloud of 1024 to 16 points with a following PointNet reduces inference time by over 90% compared to running PointNet on the complete point cloud, with only 5% increase in space. See full details in the supplementary material.

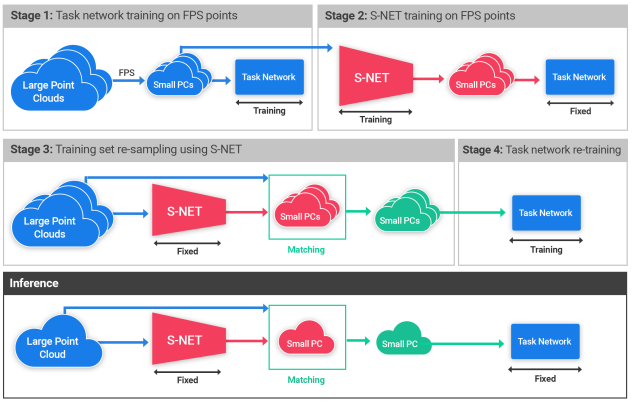


Figure 5. **Scalability by S-NET.** Illustration of the proposed training and inference procedure on large point clouds. In stage 1 we use FPS to sample the point clouds to a trainable size and use the sampled point clouds to train the task network. In stage 2 we train S-NET on the sampled point clouds, employing the fixed task network. In stage 3 we apply S-NET to re-sample the large point clouds and in stage 4 we train the task network again, this time on S-NET's points. At inference time, we use S-NET to sample a large point cloud to the size the task network was trained for.

Approximate sampling Up to this point we applied the matching post-processing step to compare ourselves directly with FPS. However, there might be settings where the k output points do not have to be a subset of the original point cloud. In such cases, we can use S-NET's generated points directly, forgoing the matching step. One can use either the generated or the sampled points. A third alternative is to interpolate between the two, i.e., using points that are up to ϵ away from the original input points. This is done by interpolating each generated point with its matched sampled point, to get a third point on the line between them, no more than ϵ away from the sampled point. Figure 6 shows that PointNet's accuracy is higher when feeding it the generated points. For point clouds normalized to the unit sphere, we find that choosing $\epsilon = 0.05$ results in classification accuracy that is about mid-way between the accuracy on the sampled and generated points. Note that ϵ is set at inference time.

Critical set sampling The critical set, as defined by Qi *et al*. [27], is the set of points that contributed to the max pooled features. A plausible alternative to our method might be to sample the critical points that contribute the most features. We tried this approach and found it viable only at small sampling ratios. See full details in the supplementary material.

4.2. Retrieval

We now show that employing S-NET instead of FPS leads to better retrieval results. Specifically, we took the S-NET that was trained with PointNet for classification as the task network, without re-training it. We sampled the points

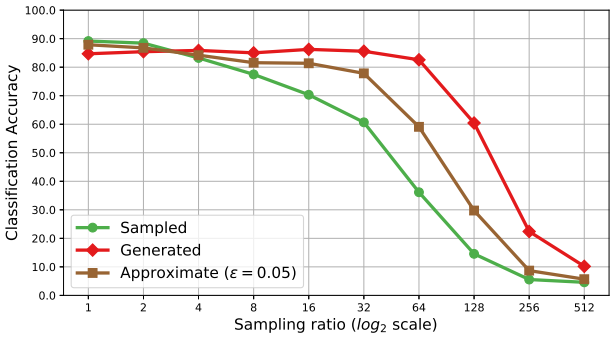


Figure 6. **Approximate sampling.** PointNet was trained on complete point clouds (1024 points) and evaluated on point clouds of different sizes. We use S-NET's generated and sampled points, as well as a third set of points, where each point is an interpolation between the generated and sampled point, bounded to be no more than $\epsilon = 0.05$ away from an original input point. Approximate sampling enables higher accuracy, when deviating from original points is not a concern.

Sampling ratio	FPS mAP	S-NET mAP
1	71.3	71.3
2	**70.1**	69.8
4	**65.7**	64.8
8	58.3	**60.4**
16	49.4	**59.0**
32	37.7	**59.0**
64	27.4	**54.5**

Table 2. **Point cloud retrieval.** We took the same S-NET that was trained with PointNet classifier as the task network and applied it as the sampling method for retrieval. The shape descriptor was PointNet's activations of the layer before the last, with L_2 as the distance metric. We measured the macro mean Average Precision (mAP) for different sampling ratios and methods. S-NET performs better than FPS for large sampling ratios and is almost insensitive to the sampling ratio.

that are fed to PointNet and used its penultimate layer as a shape descriptor. Retrieval was done based on L_2 distance on this shape descriptor. We repeated the experiment with every shape in the ModelNet40 test set serving as a query shape. The experiment was repeated with different sample sizes for both S-NET and FPS.

Table 2 summarizes the mean Average Precision (mAP) for different sampling ratios and sampling methods. We see that S-NET performs much better for sampling ratios larger than 4 and is not very sensitive to the sampling ratio. Figure 7 presents the precision-recall curve for the original data and for the sampled data using FPS and S-NET, sampling to k=32 points per shape. We observe significantly better precision when applying S-NET across all recall values.

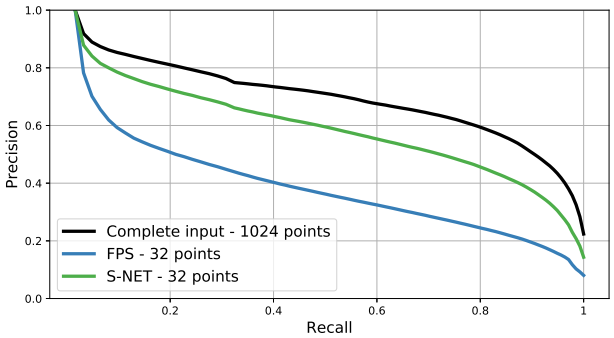


Figure 7. **Precision-Recall curve for point cloud retrieval.** We compare FPS and S-NET when sampling 32 points, as well as using the complete 1024 points data. Significantly better precision is achieved when using S-NET compared to FPS, across all recall values.

4.3. Reconstruction

We next learn to sample with an autoencoder as the task network. We used the ShapeNet Core55 point cloud data provided by Achlioptas *et al*. [1], as well as their autoencoder and trained it on four shape classes: table, car, chair and airplane. These classes have the most available models. Each shape class is split into 85%-5%-10% for train-validation-test sets. The autoencoder was trained to receive and reconstruct point clouds of 2048 points.

The reconstruction error of the autoencoder is measured by the Chamfer distance [1]. In order to compare different sampling methods, we use Normalized Reconstruction Error (NRE). That is, we reconstruct the complete point set from a subset of points and from the complete point set and take the reconstruction error ratio between the two. We trained several S-NETs with the following sample sizes: $k \in \{16, 32, \ldots, 2048\}$, and a single ProgressiveNet with loss terms for the same sizes. As an alternative sampling approach we used FPS.

Normalized reconstruction error Figure 8 presents the NRE as a function of the sampling ratio, where the sampling ratio is defined as $n/k = 2048/k$. We compare FPS with S-NET and ProgressiveNet. For small sampling ratios, the NRE for our sampled points is similar to that of FPS. However, as the sampling ratio increases, our sampling methods outperform FPS. For example, at sampling ratio of 32, the NRE of FPS is a little over 2, while the NRE of S-NET and ProgressiveNet is about 1.5 and 1.75, respecitvely. S-NET achieves lower NRE than ProgressiveNet, since the former was optimized separately per sampling ratio, resulting in improved reconstruction performance. We learn to sample points from unseen shapes that enable lower reconstruction error.

Sample and reconstruction visualization Figure 9 compares the reconstruction result from the entire point cloud,

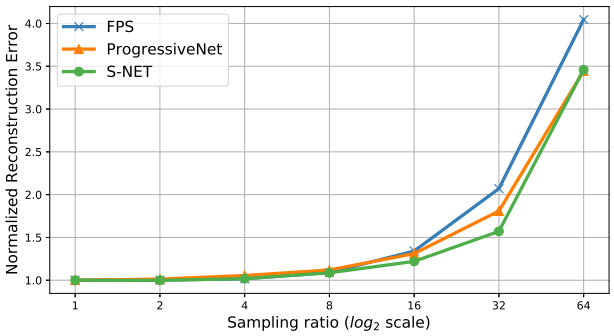


Figure 8. **Normalized Reconstruction Error (NRE).** We trained an autoencoder on complete point clouds of 2048 points, and evaluated the reconstruction error from sampled point clouds with FPS, S-NET and ProgressiveNet on the test split. Up to a sampling ratio of 8, the error for S-NET and ProgressiveNet is on par with FPS. However, at higher sampling ratios S-NET and ProgressiveNet achieves lower error.

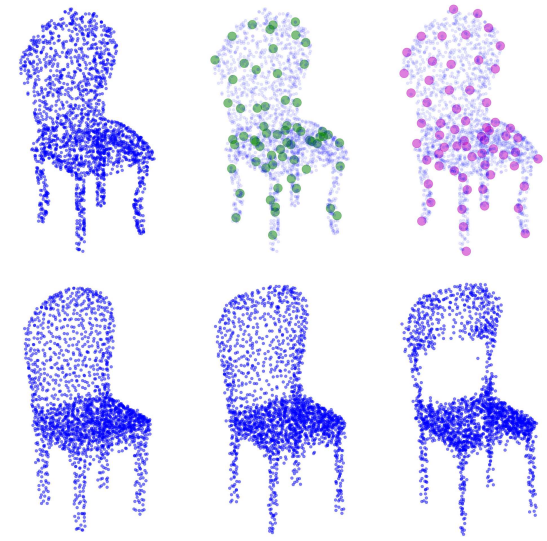
to that from a sample size of 64 points, where the samples are produced by either S-NET or FPS. The reconstruction quality when employing S-NET is higher than that of using FPS and approaches that of using the entire point cloud. Interestingly, the points sampled by S-NET are *non-uniformly* distributed, as opposed to the more uniform distribution of the points sampled by FPS.

Adversarial simplification In this proof of concept we show how to trick the autoencoder. We simplify a point cloud to be visually similar to one class but reconstructed by the autoencoder to a shape from a *different* class. We train S-NET with a single pair of input and target shapes, where the input is a shape from one class and the target is a shape from another class. The sampling loss was between the input and the points generated by S-NET. The reconstruction loss was between the target shape and the reconstructed one. Figure 10 shows the result of turning an airplane into a car.

5. Conclusions

We presented a method that learns how to sample a point cloud that is optimized for a downstream task. The method consists of a simplifying network, S-NET, followed by a post-processing matching step. We also suggested a network, termed ProgressiveNet, for ordering a point cloud according to the contribution of each point to the task. The resulting methods outperform FPS in sampling points for several tasks: classification, retrieval and reconstruction of point clouds.

The proposed method is general and can produce a small point cloud that consists of points that are not necessarily part of the original input shape. The output point cloud minimizes a geometric error with respect to the input point



Complete Input NRE = 1.00 | S-NET NRE = 1.11 | FPS NRE = 2.38

Figure 9. **Point cloud reconstruction.** NRE stands for Normalized Reconstruction Error. Top row: complete input point cloud of 2048 points, input with 64 S-NET sampled points (in Green), input with 64 FPS points (in Magenta). The sampled and FPS points are enlarged for visualization purpose. Bottom row: reconstructed point cloud from the input and from the corresponding sample. The reconstructed point cloud from S-NET's sampled points is visually more similar to the input and has lower reconstruction error.

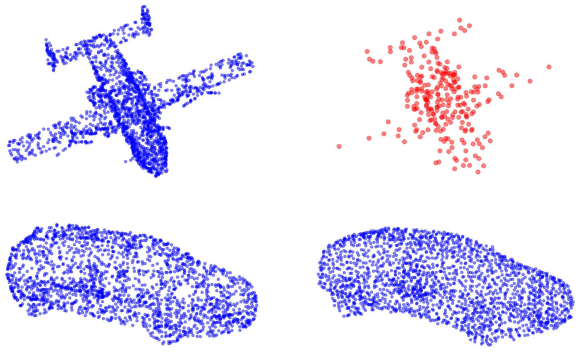


Figure 10. **Adversarial simplification.** Top row: input shape (in Blue) and 256 generated points (in Red). Bottom row: target shape and reconstruction from the generated points. While the simplified point cloud resembles the input airplane shape, it is reconstructed to a *completely different* shape - a car!

cloud while optimizing the objective function of a downstream task. We have shown that learning to sample can improve results and be used in conjunction with various applications.

Acknowledgments: Parts of this research were supported by ISF grant 1917/15.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J Guibas. Learning Representations and Generative Models For 3D Point Clouds. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[2] Shubham Agrawal and Swaminathan Gurumurthy. High Fidelity Semantic Shape Completion for Point Clouds using Latent Optimization. *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.

[3] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point Set Surfaces. *Proceedings of IEEE Visualization Conference*, pages 21–28, 2001.

[4] 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. *arXiv preprint arXiv:1512.03012*, 2015.

[5] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovaevi. Fast Resampling of Three-Dimensional Point Clouds via Graphs. *IEEE Transactions on Signal Processing*, 66:666–681, 2018.

[6] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Y. Yehoshua Zeevi. The Farthest Point Strategy for Progressive Image Sampling. *IEEE Transactions on Image Processing*, 6:1305–1315, 1997.

[7] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[9] Isabelle Guyon and Andr Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, pages 1157–1182, 2003.

[10] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN: A Network with an Edge. *arXiv preprint arXiv:1809.05910*, 2018.

[11] Hugues Hoppe. Progressive Meshes. *Proceedings of the ACM Special Interest Group on Computer Graphics (SIGGRAPH)*, pages 99–108, 1996.

[12] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise Convolutional Neural Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] Yan Huang, Jingliang Peng, C.-C. Jay Kuo, and M. Gopi. Octree-Based Progressive Geometry Coding of Point Clouds. *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 103–110, 2006.

[14] Ba Jimmy, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[15] Sagi Katz and Ayellet Tal. Improving the Visual Comprehension of Point Sets. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 121–128, 2013.

[16] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, pages 273–324, 1997.

[17] Zhenzhong Kuang, Jun Yu, Jianping Fan, and Min Tan. Deep Point Convolutional Approach for 3D Model Retrieval. *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, 2018.

[18] Linsen Lars. Point Cloud Representation. *Technical Report, Faculty of Computer Science, University of Karlsruhe*, 2001.

[19] Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point Cloud GAN. *arXiv preprint arXiv:1810.05795*, 2018.

[20] Jiaxin Li, Ben M Chen, and Gim Hee Lee. SO-Net: Self-Organizing Network for Point Cloud Analysis. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[21] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2018.

[22] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent Models of Visual Attention. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2014.

[23] Carsten Moenning and Neil A. Dodgson. A new point cloud simplification algorithm. *Proceedings of the IASTED International Conference on Visualization, Imaging and Image Processing (VIIP)*, 2003.

[24] Carsten Moenning and Neil A. Dodgson. Fast Marching farthest point sampling. *Eurographics Poster Presentation*, 2003.

[25] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient Simplification of Point-Sampled Surfaces. *Proceedings of IEEE Visualization Conference*, 2002.

[26] Jingliang Peng and C.-C. Jay Kuo. Octree-Based Progressive Geometry Encoder. *Proceedings of SPIE*, pages 301–311, 2003.

[27] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[28] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

[29] Ruwen Schnabel and Reinhard Klein. Octree-based Point-Cloud Compression. *Proceedings of the Eurographics Symposium on Point-Based Graphic*, 2006.

[30] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[31] Jag Mohan Singh and P.J. Narayanan. Progressive Decomposition of Point Clouds Without Local Planes. *Computer Vision, Graphics and Image Processing*, pages 364–375, 2006.

[32] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2530–2539, 2018.

[33] Yongbin Sun, Yue Wang, Ziwei Liu, Joshua E. Siegel, and Sanjay E. Sarma. PointGrow: Autoregressively Learned Point Cloud Generation with Self-Attention. *arXiv preprint arXiv:1810.05591*, 2018.

[34] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. SEGCloud: Semantic Segmentation of 3D Point Clouds. *Proceedings of the International Conference on 3D Vision (3DV)*, 2017.

[35] Mikaela Angelina Uy and Gim Hee Lee. PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[36] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[37] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[38] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[39] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen Or, and Pheng Ann Heng. EC-Net: an Edge-aware Point set Consolidation Network. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[40] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng Ann Heng. PU-Net: Point Cloud Upsampling Network. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[41] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. PCN: Point Completion Network. *Proceedings of the International Conference on 3D Vision (3DV)*, 2018.

[42] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabs Pczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep Sets. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

[43] Wentai Zhang, Haoliang Jiang, Zhangsihao Yang, Soji Yamakawa, Kenji Shimada, and Levent Burak Kara. Data-driven Upsampling of Point Clouds. *arXiv preprint arXiv:1807.02740*, 2018.