ConvPoint: Continuous convolutions for point cloud processing

Alexandre Boulch

ConvPoint: Continuous Convolutions for Point Cloud Processing

Preprint. Accepted at Computer & Graphics, see final version in journal issue.

Alexandre Boulch
ONERA, Université Paris-Saclay, FR-91123 Palaiseau, France
valeo.ai, Paris, France

Abstract

Point clouds are unstructured and unordered data, as opposed to images. Thus, most machine learning approach developed for image cannot be directly transferred to point clouds. In this paper, we propose a generalization of discrete convolutional neural networks (CNNs) in order to deal with point clouds by replacing discrete kernels by continuous ones. This formulation is simple, allows arbitrary point cloud sizes and can easily be used for designing neural networks similarly to 2D CNNs. We present experimental results with various architectures, highlighting the flexibility of the proposed approach. We obtain competitive results compared to the state-of-the-art on shape classification, part segmentation and semantic segmentation for large-scale point clouds.

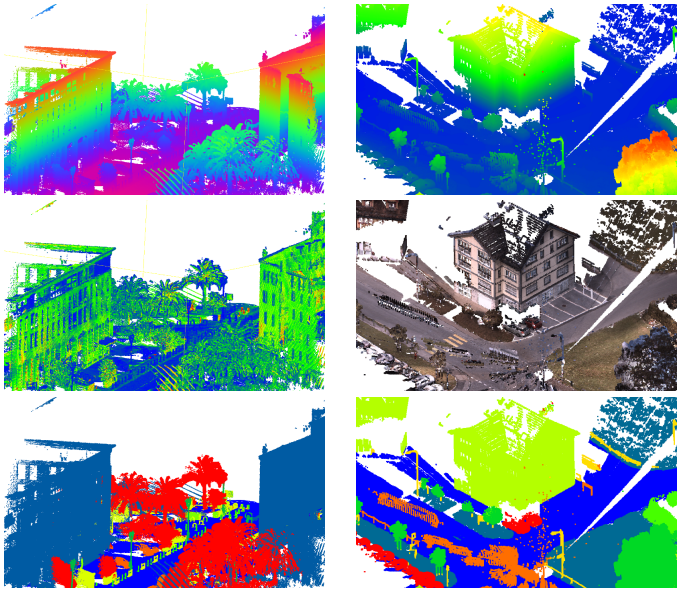
1 Introduction

Point clouds are widely used in varied domains such as historical heritage preservation and autonomous driving. They can be either directly generated using active sensors such as LIDARs or RGB-Depth sensors, or an intermediary product of photogrammetry.

These point sets are sparse samplings of the underlying surface of scenes or objects. With the exception of structured acquisitions (e.g., with LIDARs), point clouds are generally unordered and without spatial structure; they cannot be sampled on a regular grid and processed as image pixels. Moreover, the points may or may not hold colorimetric features. Thus, point-cloud dedicated processing must be able to deal only with the relative positions of the points.

Due to these considerations, methods developed for image processing cannot be used directly on point clouds. In particular, convolutional neural networks (CNNs) have reached state-of-the-art performance in many image processing tasks. They use discrete convolutions which make extensive use the grid structure of the data, which does not exist with point clouds.

Two common approaches to circumvent this problem are: first, to project the points into a space suitable for discrete convolutions, e.g., voxels; second, to reformulate the CNNs to take into account point clouds' unstructured nature.



(a) NPM3D (b) Semantic8

Figure 1: Segmentation results on datasets (a) NPM3D and (b) Semantic8. Point cloud colored according height (top), laser intensity or RGB colors (middle) and predicted segmentation label (bottom).

In this paper, we adopt the second approach and propose a generalization of CNNs for point clouds. The main contributions of this paper are two-fold.

First, we introduce a continuous convolution formulation designed for unstructured data. The continuous convolution is a simple and straightforward extension of the discrete one.

Second, we show that this continuous convolution can be used to build neural networks similarly to its image processing counterpart. We design neural networks using this convolution and a hierarchical data representation structure based on a search tree.

We show that our framework, *ConvPoint*, which extends our work presented in [11], can be applied to various classification and segmentation tasks, including large scale indoor and outdoor semantic segmentation. For each task, we show that ConvPoint is competitive with the state-of-the-art.

The paper is organized as follows: Section 2 presents the related work, Section 3 describes the continuous convolutional formulation, Section 4 is dedicated to the spatial representation of the data, and Section 5 presents the convolution as a layer. Finally, Section 6 shows experiments on different datasets for classification and semantic segmentation of point clouds.

2 Related work

Point cloud processing is a widely discussed topic. We focus here on machine learning techniques for point cloud classification or local attribute estimation.

Most of the early methods use handcrafted features defined using a point and its neighborhood, or a local estimate of the underlying surface around the point. These features describe specific properties of the shape and are designed to be invariant to rigid or non rigid transformations of the shape [5, 14, 24, 33]. Then, classical machine learning algorithms are trained using these descriptors.

In the last years, the release of large annotated point cloud databases has allowed the development of deep neural network methods that can learn both descriptors and decision functions.

The direct adaptation of CNNs developed for image processing is to use 3D convolutions. Methods like [35, 55] apply 3D convolutions on a voxel grid. Even though recent hardware advances enable the use of these networks on larger scenes, they are still time consuming and require a relatively low voxel resolution, which may result in a loss of information and undesirable bias due to grid axis alignment. In order to avoid these drawbacks, [19,20] use sparse convolutional kernels and [30,51] scan the 3D space to focus computation where objects are located.

A second class of algorithms avoids 3D convolutions by creating 2D representations of the shape, applying 2D CNNs and projecting the results back to 3D. This approach has been used for object classification [47], jointly with voxels [39], and for semantic segmentation [12]. One of the main issues of using multi-view frameworks is to design an efficient and robust strategy for choosing viewpoints.

The previous methods are based on 2D or 3D CNNs, creating the need to structure the data (3D grid or 2D image) in order to process it. Recently, work has focused on developing deep learning architectures for non-Euclidean data. This work, referred to as *geometric deep learning* [13], operates on manifolds, graphs, or directly on point clouds.

Neural networks on graphs were pioneered in [42]. Since then, several methods have been developed using the spectral domain from the graph Laplacian [15, 44, 58] or polynomials [18] as well as gated recurrent units for message passing [26, 31].

The first extension of CNNs to manifolds is Geodesic CNN [34], which redefines usual layers such as convolution or max pooling. It is applied to local mesh patches in a polar representation to learn invariant shape features for shape correspondences. In [10], the representation is improved with anisotropic diffusion kernels and the resulting method is not bound to triangular meshes anymore. More recently, MoNet [36] offers a unified formulation of CNNs on manifolds which included [3, 10, 34] and reaches state-of-the-art performance on shape correspondence benchmarks.

These methods have proved to be very efficient but they usually operate on graphs or meshes for surface analysis. However, building a mesh from raw point clouds is a very difficult task and requires in practice priors regarding the surface to be reconstructed [8].

A fourth class of machine learning approaches processes directly the raw point clouds; the method proposed in this paper belongs to this category.

One of the recent breakthroughs in dealing with unstructured data is PointNet [38]. The key idea is to construct a transfer function invariant by permutation of the inputs features, obtained by using the order invariant *max pooling* function. The coordinates of the points are given as input and geometric operations (affine transformations) are obtained with small auxiliary networks. However, it fails to capture local structures. Its improvement, PointNet++ [40], uses a cascade of PointNet networks from local scale to global scale.

Convolutional neural layers are widely used in machine learning for image processing and more generally for data sampled on a regular grid (such as 3D voxels). However, the use of CNNs when data is missing or not sampled on a regular grid is not straightforward. Several works have studied the generalization of such powerful schemes to such cases.

To avoid problems linked to discrete kernels and sparse data. [17] introduces deformable convolutional kernels able to adapt to the recognition task. In [46], the authors adapt [17] to deal with point clouds. The input signal is interpolated on the convolutional kernel, the convolution is applied, and the output is interpolated back to input shape. The kernel elements' locations are optimized like in [17] and the input points are weighted according to their distance to kernel elements, like in [46]. However, unlike in [46], the our approach is not dependent of a convolution kernel designed on a grid.

In [29], a χ-transform is applied on the point coordinates to create geometrical features to be combined with the input point features. This is a major difference relative to our approach: as in [40], [29] makes use of the input geometry as features. We want the geometry to behave as in discrete convolutions, i.e. weighting the relation between kernel and input. In other words, the geometric aspects are defined in the network structure (convolution strides, pooling layers) and point coordinates (e.g. pixels indices for images) are not an input of the network.

In [52], the authors propose a convolution layer defined as a weighted sum of the input features, the weights being computed by a multi-layer perceptron (MLP). Our approach has points in common with [52] both in concept and implementation, but differs in two ways. First, our approach computes a dense weighting function that takes into account the whole kernel. Second, as in [49], the derived kernel is an explicit set of points associated with weights. However, whereas [49] uses an explicit RBF Gaussian function to correlate input and kernel, we propose to learn this kernel to input relation function with a multi-layer perceptron (MLP).

3 Convolution for point processing

We build our continuous convolutional layer by adapting the discrete convolution formulation used for grid-sampled data such as images.

Notations. In the following sections, d is the dimension of the spatial domain (e.g., 3 for 3D point clouds) and n is the dimension of the features domain (i.e., the dimension of the input features of the convolutional layer). $[\![a, b]\!]$ is an integer sequence from a to b with step 1. The cardinality of a set S is denoted by $|S|$.

3.1 Discrete convolutions

Let $K = \{\mathbf{w}_i, \in \mathbb{R}^n, i \in [\![1, |K|]\!]\}$ be the kernel and $X = \{\mathbf{x}_i, \in \mathbb{R}^n, i \in [\![1, |X|]\!]\}$ be the input. In discrete convolutions, K and X have the same cardinality.

We first consider the case where the elements of K and X range over the same locations (e.g., grid coordinates on an image).

Given a bias β, the output y is:

$$y = \beta + \sum_{i=1}^{|K|} \mathbf{w}_i \mathbf{x}_i = \beta + \sum_{i=1}^{|K|} \sum_{j=1}^{|X|} \mathbf{w}_i \mathbf{x}_j \mathbf{1}(i, j) \tag{1}$$

where $\mathbf{1}(.,.)$ is the indicator function such that $\mathbf{1}(a, b) = 1$ if $a = b$ and 0 otherwise. This expresses a one-to-one relation between kernel elements and input elements.

We now reformulate the previous equation by making explicit the underlying order of sets K and X. Let's consider $K = \{(\mathbf{c}, \mathbf{w})\}$ (resp. $X = \{(\mathbf{p}, \mathbf{x})\}$) where $\mathbf{c}_i \in \mathbb{R}^d$ (resp. $\mathbf{p}_j \in \mathbb{R}^d$) is the spatial location of the kernel element i (resp. j is the spatial location of the j-th point in the input). For convolutions on images, $\mathbf{c}$ and $\mathbf{p}$ denote pixel coordinates in the considered patch. The output is then:

$$y = \beta + \sum_{j=1}^{|X|} \sum_{i=1}^{|K|} \mathbf{w}_i \mathbf{x}_j \mathbf{1}(\mathbf{c}_i, \mathbf{p}_j) \tag{2}$$

3.2 Convolutions on point sets

In this study, we consider that elements of X may have any spatial locations, i.e., we do not assume a grid or other structure on X. In practice, using equation (2) on such an X would result in $\mathbf{1}(\mathbf{c}, \mathbf{p})$ to be zero (almost) all the time as the probability for a random $\mathbf{p}$ to match exactly $\mathbf{c}$ is zero. Thus, using the indicator function is too restrictive to be useful when processing point clouds.

We need a more general function ϕ to establish a relation between the points $\{\mathbf{p}\}$ and the kernel elements $\{\mathbf{c}\}$. We define ϕ as a geometrical weighting function that distributes the input $\mathbf{x}$ onto the kernel.

ϕ must be invariant to permutations of the input points (it is necessary since point clouds are unordered). This is achieved if ϕ is independently applied on each point $\mathbf{p}$. It can also be function of the set kernel points, i.e., $\{\mathbf{c}\}$.

Moreover, to be invariant to a global translation applied on both the kernel and the input cloud, we used relative coordinates with respect to kernel elements, i.e., we apply the ϕ function on $\{\mathbf{p}_j - \mathbf{c}_i\}, i \in [\![1, |K|]\!]$, the set of relative positions of $\mathbf{p}_j$ relatively to kernel elements.

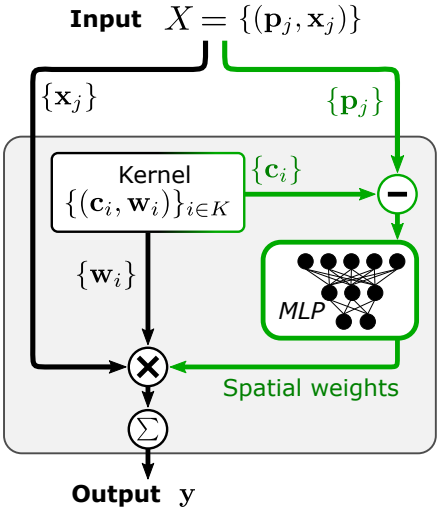


Figure 2: Convolution operation. Black arrows and boxes with black frames are features operations, green ones are spatial operations.

The ϕ function is then a function such that:

$$\phi \colon \mathbb{R}^d \times (\mathbb{R}^d)^{|K|} \longrightarrow \mathbb{R} \tag{3}$$

$$(\mathbf{p}, \{\mathbf{c}\}) \mapsto \phi(\{\mathbf{p} - \mathbf{c}\}) \tag{4}$$

Finally, the convolution operation for point sets is defined by:

$$y = \beta + \frac{1}{|X|} \sum_{j=1}^{|X|} \sum_{i=1}^{|K|} \mathbf{w}_i \, \mathbf{x}_j \, \phi(\{\mathbf{p}_j - \mathbf{c}\}) \tag{5}$$

where we added a normalization according to the input set size for robustness to variation in input size.

In this formulation, the spatial domain and the feature domain are mixed similarly to the discrete formulation. Unlike PointNet [38] and PointNet++ [40], spatial coordinates are not input features. This formulation is closer to [52] where the authors estimate directly the weights of the input features, i.e., the product $\mathbf{w}_i \, \phi(\{\mathbf{p}_j - \mathbf{c}\})$, mixing estimation in the feature space and the geometrical space.

3.3 Weighting function

In practice designing by hand such a ϕ function is not easy. Intuitively, it would decrease with the norm of $\mathbf{p} - \mathbf{c}$. As in [49], Gaussian functions could be used for that, but it would require handcrafted parameters which are difficult to tune. Instead, we choose to learn this function with a simple MLP. Such an approach does not make any specific assumption about the behavior of the function ϕ.

3.4 Convolution

The convolution operation is illustrated in Fig. 2. For a kernel $\{(\mathbf{c}, \mathbf{w})\}$, the spatial part $\{\mathbf{c}\}$ and feature part $\{\mathbf{w}\}$ are processed separately. The black arrows and boxes are for operations in the features space, green ones are for operations in the point

cloud space. Please note that removing the green operations corresponds to the discrete convolution.

Moreover, as in the case of discrete convolutions, our convolutional layer is made of several kernels (corresponding to the number of output channel). The output of the convolution is thus a vector $\mathbf{y}$.

3.5 Parameters and optimization on kernel element location

To set the location of the kernel elements, we randomly sample the locations $\mathbf{c} \in \{\mathbf{c}\}$ in the unit sphere and consider them as parameters. As ϕ is differentiable with respect to its input, $\{\mathbf{c}\}$ can be optimized.

At training time, both parameters $\{\mathbf{w}\}$ and $\{\mathbf{c}\}$ of K as well as the MLP parameters are optimized using gradient descent.

3.6 Properties

Permutation invariance. As stated in [38], operators on point clouds must be invariant unser the permutation of points. In the general case, the points are not ordered. In our formulation, y is a sum over the input points. Thus, permutations of the points have no effect on the results.

Translation invariance. As the geometric relations between the points and the kernel elements are relative, i.e., $(\{\mathbf{p} - \mathbf{c}\})$, applying a global translation to the point cloud does not change the results.

Insensitivity to the scale of the input point cloud. Many point clouds, such as photogrammetric point clouds, have no metric scale information. Moreover the scale may vary from one point cloud to another inside a dataset. In order to make the convolution robust to the input scale, the input geometric points $\{\mathbf{p}\}$ are normalized to fit in the unit ball.

Reduced sensibility to the size of input point cloud. Dividing $\mathbf{y}$ by $|X|$ makes the output less sensitive to input size. For instance, using "$2X$" as input does not change the result.

4 Hierarchical representation and neighborhood computation

Let P be an input point cloud. The convolution as described in the previous section is a local operation on a subset X of P. It operates a projection of P on an output point cloud $\{\mathbf{q}\}$ and computes the features $\{\mathbf{y}\}$ associated with each point of $Q = \{\mathbf{q}, \mathbf{y}\}$.

Depending on the cardinality of Q, we have three possible behaviors for the convolution layer:

(a) $|P| = |Q|$. In this case the cardinality of the output is the same as the intput. In the discrete framework it corresponds to the convolution without stride.

(b) $|P| > |Q|$. This includes the particular case of $\{\mathbf{p}\} \supset \{\mathbf{q}\}$. The convolution operates a spatial size reduction. The parallel with the discrete framework is a convolution with stride bigger than one.

(c) $|P| < |Q|$. This includes the particular case of $\{\mathbf{p}\} \subset \{\mathbf{q}\}$. The convolutional layer produces an upsampled version of the input. It is similar to the up-convolutional layer in discrete pipelines.

Computation of $\{\mathbf{q}\}$ $\{\mathbf{q}\}$ can either be given as an input, or computed from $\{\mathbf{p}\}$. For the second case, a possible strategy is to randomly pick points in $\{\mathbf{p}\}$ to generate $\{\mathbf{q}\}$. However, it is possible to pick several times the same point and some points of $\{\mathbf{p}\}$ may not be in the neighborhoods of the points of $\{\mathbf{q}\}$. An alternative is proposed in [40] using a furthest-point sampling strategy. This is very efficient and ensures a spatially uniform sampling, however it requires to compute all mutual distances in the point cloud.

We propose an intermediate solution. For each point, we memorize how many times it has been selected. We pick the next point in the set of points with the lower number of selection. Each time a point $\mathbf{q}$ is selected, its score is increased by 100. The score of the points in its neighborhood are increased by 1. The points of $\{\mathbf{q}\}$ are iteratively picked until the specified number of points is reached. Using a higher score for the points in $\{\mathbf{q}\}$ ensures that they will not be chosen anymore, except if all points have been selected once.

Neighborhood computation All k-nearest neighbor search are computed using a kd-tree built with $\{\mathbf{p}\}$.

5 Convolutional layer

The convolutional layer is presented on Fig. 3. It is composed of the two previously described operations (point selection and the convolution itself). The inputs are P and optionally $\{\mathbf{q}\}$. If $\{\mathbf{q}\}$ is not provided, it is selected as a subset of P following the procedure described in the previous section. First, for each point of $\{\mathbf{q}\}$, local neighborhoods in P are computed using a k-d tree. Then, for each of these subsets, we apply the convolution operation, creating the output features. Finally, the output Q is the union of the pairs $\{(\mathbf{q}, \mathbf{y})\}$.

Parameters The parameters of the convolutional layers are very similar to discrete convolution parameters in the most deep learning frameworks.

- **Number of output channels** (C): it is the number of convolutional kernels used in the layer. It defines the output feature dimension, i.e., the dimension of $\mathbf{y}$.
- **Size of the output point cloud** ($|Q|$): it is the number of points that are passed to the next layer.
- **Kernel size** ($|K|$): it is the number of kernel elements used for the convolution computation.
- **Neighborhood size** (k): it is the number of points in $\{\mathbf{p}\}$ to consider for each point in $\{\mathbf{q}\}$.

6 Experiments

The following section is dedicated to experiments and comparison with state-of-the-art methods. As the spatial structure generation (selection of the output point cloud for each layer) is a stochastic process, two runs through the network may lead to different outputs. In the following, we aggregate the results of multiple runs by averaging the outputs. The number of runs is then referred to as the number of spatial samplings. In the

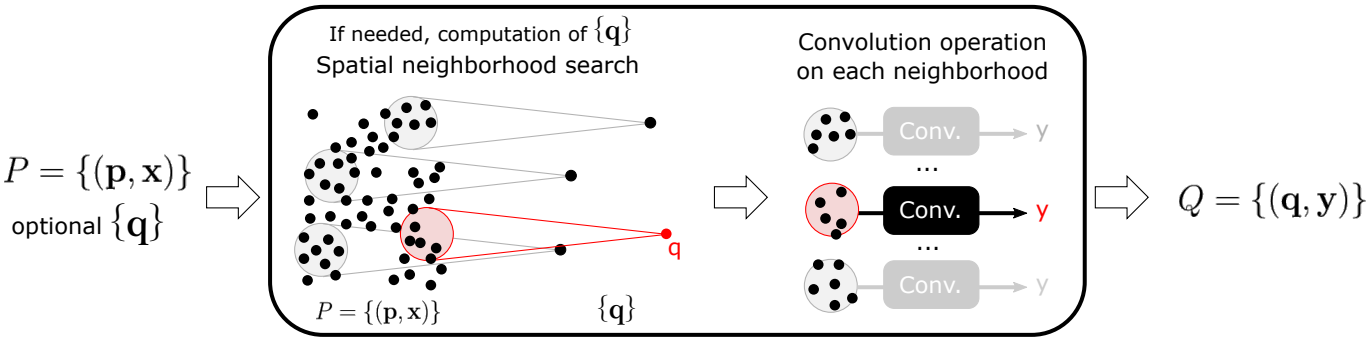
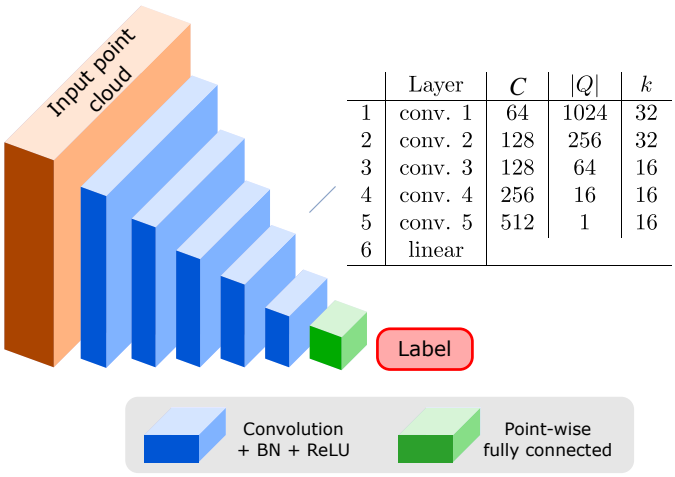


Figure 3: Convolutional layer, composed of two steps: the spatial structure computation (selecting $\{q\}$ if needed by computing the local neighborhoods of each q) and the convolution operation itself on each neighborhood, as defined in figure 2.



	Layer	C	$\lvert Q\rvert$	k
1	conv. 1	64	1024	32
2	conv. 2	128	256	32
3	conv. 3	128	64	16
4	conv. 4	256	16	16
5	conv. 5	512	1	16
6	linear			

Figure 4: Classification network. It is composed of five convolution layer (blue blocks) with progressive point cloud size reduction ($|Q|$) and a final fully connected layer (green block). For all layers $|K| = 16$.

folowing tables, it correspond to the number between parentheses (for classification and part segmentation). The influence of this number is discussed in section 6.3.1.

6.1 Classification

The first experiments are dedicated to points cloud classification. We experimented on both 2D and 3D point cloud datasets.

Network The network is described in Fig. 4. It is composed of five convolutions that progressively reduce the point cloud size to one single point, while increasing the number of channels. The features associated to this point are the inputs of a linear layer. This architecture is very similar to the ones that can be used for image processing (e.g., LeNet [27]).

2D classification: MNIST The 2D experiment is done on the MNIST dataset. This is a dataset for the classification of gray scale handwritten digits. The point cloud $P = \{(\mathbf{p}, \mathbf{x})\}$ is built from the images, using pixel coordinates as point coordinates and, thus, is sampled on a grid. We build two variants of the dataset: first, point clouds are built with the whole image and the features associated with each point is the grey level ($\{\mathbf{x}\} = \{\text{grey level}\}$); second, only the black points are considered ($\{\mathbf{x}\} = \{1\}$).

Table 1: Shape classification. Overall accuracy (OA %) and class average accuracy (AA, %).

(a) MNIST

Methods	OA
Image-based methods	
LeNet [27]	99.20
NiN [32]	99.53
Point-based methods	
PointNet++ [40]	99.49
PointCNN [29]	99.54
Ours - Gray levels (16)	**99.62**
Ours - Black points (16)	99.49

(b) ModelNet 40

Methods	OA	AA
Mesh or voxels		
Subvolume [39]	89.2	
MVCNN [47]	90.1	
Points		
DGCNN [54]	92.2	**90.2**
PointNet [38]	89.2	86.2
PointNet++ [40]	90.7	
PointCNN [29]	92.2	88.1
KPConv [49]	**92.9**	
Ours - 1024 pts (16)	91.8	88.5
Ours - 2048 pts (16)	92.5	89.6

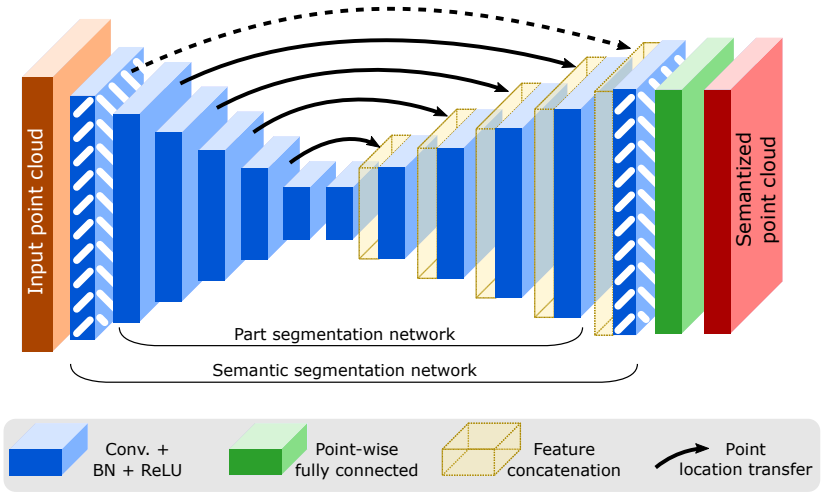
Results are presented in table 1(a). We compare with both image CNNs (LeNet [27] and Network in Network [32]) and point-based methods (PointNet++ [40] and PointCNN [29]). Scores, averaged over 16 spatial samplings, are competitive with other methods. More interestingly, we do not observe a great difference between the two variants (grayscale points or black points only). In the *Gray levels* experiment (whole image), the framework is able to learn from the color value only as the points do not hold shape-related information. On the contrary, in the *Black points only*, it learns from geometry only, which is a common case for point cloud.

3D classification: ModelNet40 We also experimented on 3D classification on the ModelNet40 dataset. This dataset is a set of meshes from 40 various classes (planes, cars, chairs, tables...). We generated point clouds by randomly sampling points on the triangular faces of the meshes. In our experiments, we use an input size of either 1024 or 2048 points for training. Table 1(b) presents the results. As for 2D classification, we are competitive with the state of the art concerning point-based classification approaches.

6.2 Segmentation

6.2.1 Segmentation network

The segmentation network is presented on Fig. 5. It has an encoder-decoder structure, similar to U-Net, a stack of convolu-



	Layer	C	$\lvert Q\rvert$	k
(0)	conv. 0	64	$\lvert P\rvert$	16
(1)	conv. 1	64	2048	16
2	conv. 2	64	1024	16
3	conv. 3	64	256	16
4	conv. 4	128	64	8
5	conv. 5	128	16	8
6	conv. 6	128	8	4
7	(de)conv. 5	128	16	4
8	(de)conv. 4	128	64	4
9	(de)conv. 3	64	256	4
10	(de)conv. 2	64	1024	4
11	(de)conv. 1	64	2048	8
(12)	(de)conv. 0	64	$\lvert P\rvert$	8
13	linear			

Note: striped convolution (layer 0, 1 and 12) are only in semantic segmentation network.

Figure 5: Segmentation networks. It is made of an encoder (progressive reduction of the point cloud size) followed by a decoder (to get back to the original point cloud size). Skip connections (black arrows) allow information to flow directly from encoder to decoder: they provide point locations for decoder's convolutions and encoder's features are concatenated with decoder's ones at corresponding scale.

tions that reduces the cardinality of the point cloud as an encoder and a symmetrical stack of convolution as a decoder with skip connections. In the decoder, the points used for upsampling are the same as the points in the encoder at the corresponding layer. Following the U-Net architerture, the features from the encoder and from the decoder are concatenated at the input of the convolutional layers of the decoder. Finally, the last layer is a point-wise linear layer used to generate an output dimension corresponding to the number of classes.

The network comes in two variants, i.e., with two different numbers of layers. The part segmentation network (plain colors, in Fig. 5) is used with ShapeNet for part segmentation. The second network, used for large-scale segmentation, is the same network with three added convolutions (hatched layers in Fig. 5). It is a larger network; its only purpose is to deal with larger input point clouds.

For both versions of the network, we add a dropout layer between the last convolution and the linear layer. At training time, the probability of an element to be set to zero is 0.5.

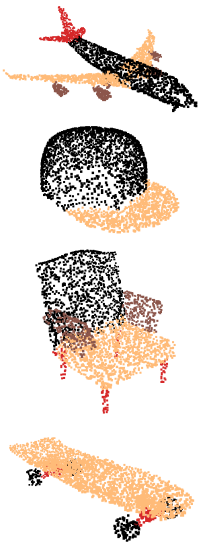
6.2.2 Part segmentation

Given a point cloud, the part segmentation task is to recognize the different constitutive parts of the shape. In practice, it is a semantic segmentation at shape level. We use the Shapenet [57] dataset. It is composed of 16680 models belonging to 16 shape categories and split in train/test sets. Each category is annotated with 2 to 6 part labels, totalling 50 part classes. As in [29], we consider the part annotation problem as a 50-class semantic segmentation problem. We use a cross-entropy loss for each category. The scores are computed at shape level.

We use the semantic segmentation network from Fig. 5. The provided point clouds have various sizes, we randomly select 2500 points (possibly with duplication if the point cloud size is lower than 2500) and predict the labels for each input point. The points do not have color features so we set the input features to one (this is the same as for the MNIST experiment with black points only). As all points may not have been selected for labeling, the class of an unlabeled point is given by the label of its nearest neighbor.

Table 2: ShapeNet

Method	mcIoU	mIoU
SyncSpecCNN [58]	82.0	84.7
Pd-Network [25]	82.7	85.5
3DmFV-Net [7]	81.0	84.3
PointNet [38]	80.4	83.7
PointNet++ [40]	81.9	85.1
SubSparseCN [19]	83.3	86.0
SPLATNet [46]	83.7	85.4
SpiderCNN [56]	81.7	85.3
SO-Net [28]	81.0	84.9
PCNN [4]	81.8	85.1
KCNet [43]	82.2	83.7
SpecGCN [50]	-	85.4
RSNet [23]	81.4	84.9
DGCNN [54]	82.3	85.1
SGPN [53]	82.8	85.8
PointCNN [29]	84.6	86.1
KPConv [49]	**85.1**	**86.4**
ConvPoint (16)	83.4	85.8
rank	*4*	*4*



The results are presented in table 2. The scores are the mean class intersection over union (mcIoU) and instance average intersection over union (mIoU). Similarly to classification, we aggregate the scores of multiple runs through the network. Our framework is also competitive with the state-of-the-art methods as we rank among the top five methods for both mcIoU and mIou.

6.2.3 Semantic segmentation

We now consider semantic segmentation for large-scale point clouds. As opposed to part segmentation, point clouds are now sampled in multi-object scenes outdoors and indoors.

Datasets We use three datasets, one with indoor scenes, two with outdoors acquisitions.

The *Stanford 2D-3D-Semantics* dataset [2] (S3DIS) is an indoor point cloud dataset for semantic segmentation. It is composed of six scenes, each corresponding to an office building floor. The points are labeled according to 13 classes: 6 building elements classes (floor, ceiling...), 6 office equipment classes (tables, chairs...) and a "stuff" class regrouping all the small equipment (computers, screens...) and rare items. For each scene considered as a test set, we train on the other five.

The *Semantic8* [21] outdoor dataset is composed of 30 ground lidar scenes: 15 for training and 15 for evaluation. Each scene is generated from one lidar acquisition and the point cloud size ranges from 16 million to 430 million points. 8 classes are manually annotated.

Finally, the *Paris-Lille 3D dataset (NPM3D)* [41] has been acquired using a Mobile Laser System. The training set contains four scenes taken from the two cites, totalizing 38 million points. The 3 tests scenes, with 10 million points each, were acquired on two other cities. The annotations correspond to 10 coarse classes from buildings to pedestrian.

For both Semantic8 and NPM3D, test labels are unknown and evaluated on an online server.

Learning and prediction strategies As the scenes may be large, up to hundred of millions of points, the whole point clouds cannot be directly given as input to the network.

At training time, we randomly select points in the considered point cloud, and extract all the points in an infinite vertical column centered on this point, the column section is 2 meters wide for indoor scenes and 8 meters wide for outdoor scenes (Fig. 6).

During testing, we compute a 2D occupancy pixel map with "pixel" size 0.1 meters for indoor scenes and 0.5 meters for outdoor scenes by projecting vertically on the horizontal plane. Then, we considered each occupied cell as a center for a column (same size as for training).

For each column, we randomly select 8192 points which we feed as input to the network. Finally, the output scores are aggregated (summed) at point level and points not seen by the network receive the label of their nearest neighbor.

Improving efficiency with fusion of two networks learned on different modalities First, the objective is to evaluate the influence of the color information as an input feature.

We trained two networks, one with color information (*RGB*) and one without (*NoColor*). Let $P = \{(\mathbf{p}, \mathbf{x})\}$ be the input point cloud. In the first case, *RGB*, the input features is the color information, i.e., $\{\mathbf{x}\} = \{(r, g, b)\}$, r, g, b being the red, green and blue values associated with each point. For the *NoColor* model, input features do not hold color information: $\{\mathbf{x}\} = \{1\}$.

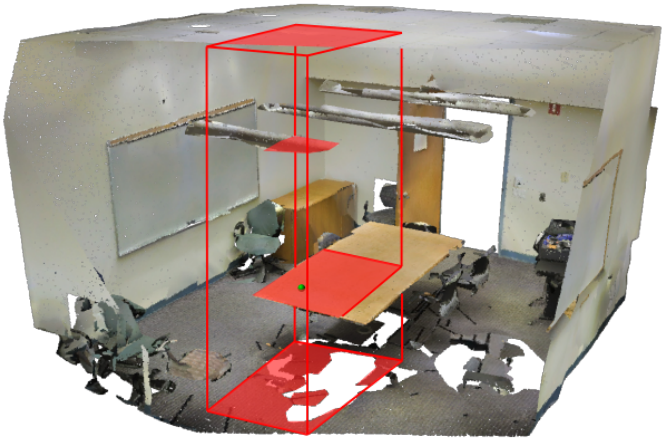


Figure 6: Point set selection for semantic segmentation. A point is selected in the original point cloud (green point) and an infinite vertical column centered around this point is defined (red frame) whose points are used for the network input (red points).

Experimental results are presented in table 4. We consider the line *ConvPoint - RGB* and *ConvPoint - NoColor* of table 4(a) (S3DIS) and 4(b)(Semantic8).

Intuitively, the *RGB* network should outperform the other model on all categories: the *RGB* point cloud is the *NoColor* point cloud with additional color information.

However, according to the scores, the intuition is only partly verified. As an example in table 4(a), the *NoColor* model is much more efficient than the *RGB* model on the *column* class, mainly due to color confusion as walls and columns have most of the time the same color. To our understanding, when color is provided, the stochastic optimization process follows the easiest way to reach a training optimum, thus, giving too much importance to color information. In contrast, the *NoColor* generates different features, based only on the geometry, and is more discriminating on some classes.

Now looking at table 4(b), we observe similar performances for both models. As these models use different inputs, they likely focus on different features of the scene. It should thus be possible to exploit this complementarity to improve the scores.

Model fusion To show the interest of fusing models, we chose to use a residual fusion module [6, 12]. This approach have proven to produce good results for networks with different input modalities. Moreover, one advantage is that the two segmentation networks are first trained separately, the fusion module being trained afterward. This training process makes it possible to first, reuse the geometry only model if the RGB information is not available and second, to train with limited resources (see implementation details in section 7).

The fusion module is presented in Fig. 7. The features before the fully-connected layer are concatenated, becoming the input of two convolutions and a point-wise linear layer. The outputs of both segmentation networks and the residual module are then added to form the final output. At training time, we add a dropout layer between the last convolution and the linear layer.

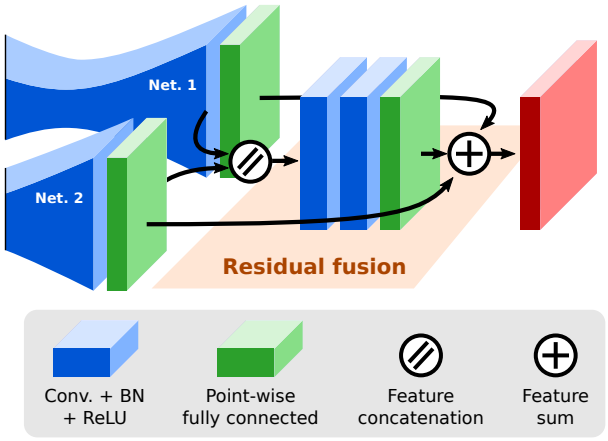


Figure 7: Segmentation networks: residual fusion. The input of the fusion module is the concatenation of the features collected before the fully-connected layer in networks 1 and 2. Its output is added as a correction to the sum of the outputs of networks 1 and 2.

In table 4(a) and (b), the results of segmentation with fusion is reported at line *ConvPoint - fusion*. As expected, the fusion increases the segmentation scores with respect to *RGB* and *NoColor* alone.

On the S3DIS dataset (table 4(a)), the fusion module obtains a better score on 10 out 13 categories and the average intersection over union is increased by 3.5%. It is also interesting to note that, on categories for which the fusion is not better than with one single modality or the other, the score is close to the best mono-mode model. In other words, the fusion often improves the performance and in any case does not degrade it.

On the Semantic8 dataset (table 4(b)), we observe the same behavior. The gain is particularly high on the artefact class, which is one of the most difficult class: both mono-mode models reach 43-44% while the fusion model reaches 61%. It validates the fact that both *RGB* and *NoColor* models learn different features and that the fusion module is not only a sum of activations, but can select the best of both modalities.

For comparison with official benchmarks, we use the fusion model.

Large-scale datasets: comparison with the state of the art Table 4 presents also, for comparison, the scores obtained by other methods in the literature. Our approach is competitive with the state of the art on the three datasets.

On S3DIS, we place second behind KPConv [49] in term of average intersection over union (mIoU), while being first on several categories. It can be noted that approaches sharing concepts with ours, such as PCCN [52] or PointCNN [29], do not perform as well as ours.

On Semantic8, we report the state of the benchmark leaderboard at the time of article writing (for entries that are not anonymous). PointNet++ has two entries in the benchmark, we only reported the best one. Our convolutional network for segmentation places first before Superpoint Graph (SPG) [26] and SnapNet [12]. It differs greatly from SPG, which relies on a pre-segmentation of the point cloud, and SnapNet, which uses a 2D segmentation network on virtual pictures of the scene to produce segments. We surpass the PointNet++ by 13% on the average IoU. We perform particularly well on car and artifacts detection where other methods, except for SPG get relatively low results.

Finally on NPM3D Paris-Lille dataset (table 4(c)), we also report the official benchmark at the time the paper was written. Based on the average IoU, we place second surpassed only by KPConv [49]. Our approach is the best or second best for 6 out of 9 categories. The second place is explained mostly by the relatively low score on pedestrian and trash cans. These are particularly difficult classes, due to their variability and the low number of instances in the train set.

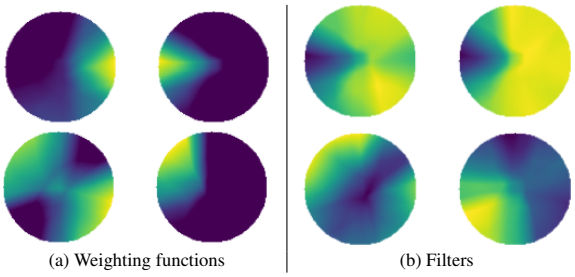


Figure 8: Weighting function and filters for the first convolution layer of the classification model trained of MNIST.

Table 3: Influence of the spatial sampling number, ShapeNet dataset.

Spatial samplings	1	2	4	8	16
mIoU	83.9	84.8	85.4	85.7	85.8

6.3 Emprical properties of the convolutional layer.

Filter visualization Fig. 8 presents a visualization of some characteristics of the first convolutional layer of the 2D classification model trained on MNIST.

Fig. 8(a) shows the weighting function associated with four of the sixty-four kernel elements of the first convolutional layer.

These weights are the output of the MLP function. As expected, their nature varies a lot depending on the kernel element, underlying different regions of interest for each kernel element.

Fig. 8(b) shows the resulting convolutional filters. These are computed using the previously presented weighting functions, multiplied by the weights of each kernel element ($\mathbf{w}$) and summed over the kernel elements. As with discrete CNNs, we observe that the network has learned various filters, with different orientations and shapes.

6.3.1 Influence of random selection of output points

The strategy for selecting points of the output $\{\mathbf{q}\}$ is stochastic at each layer, i.e., for a fixed input, two runs through the same layer may lead to two different $\{\mathbf{q}\}$'s. Therefore, the outputs $\{\mathbf{y}\}$'s may also be different, and so may be the predicted labels. To increase robustness, we aggregate several outputs computed with the same network and the same input point cloud. This is referred to as the number of sampling (from 1 to 16) in table 3. We observe an improvement of the performances with the number of spatial sampling. In practice, we only use up

Table 4: Large scene semantic segmentation benchmarks.
Best score, *second best* and third best.

(a) S3DIS

Method	OA	mAcc	mIoU	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
Pointnet [38]	78.5	66.2	47.6	88.0	88.7	69.3	42.4	23.1	47.5	51.6	42.0	54.1	38.2	9.6	29.4	35.2
RSNet [23]	-	66.5	56.5	92.5	92.8	78.6	32.8	34.4	51.6	68.1	60.1	59.7	50.2	16.4	44.9	52.0
PCCN [52]	-	67.0	58.3	92.3	96.2	75.9	0.27	6.0	**69.5**	63.5	65.6	66.9	*68.9*	47.3	*59.1*	46.2
SPGraph [26]	85.5	73.0	62.1	89.9	95.1	76.4	62.8	47.1	55.3	68.4	73.5	69.2	63.2	45.9	8.7	52.9
PointCNN [29]	*88.1*	*75.6*	65.4	94.8	*97.3*	75.8	*63.3*	*51.7*	58.4	57.2	71.6	69.1	39.1	*61.2*	52.2	58.6
KPConv [49]	-	**79.1**	**70.6**	93.6	92.4	**83.1**	**63.9**	**54.3**	*66.1*	**76.6**	57.8	64.0	**69.3**	**74.9**	**61.3**	60.3
ConvPoint - RGB	87.9	-	64.7	**95.1**	**97.7**	80.0	44.7	17.7	62.9	67.8	*74.5*	*70.5*	61.0	47.6	57.3	*63.5*
ConvPoint - NoColor	85.2	-	62.6	92.8	94.2	76.7	43.0	43.8	51.2	63.1	71.0	68.9	61.3	56.7	36.8	54.7
ConvPoint - Fusion	**88.8**	-	*68.2*	*95.0*	*97.3*	*81.7*	47.1	34.6	63.2	*73.2*	**75.3**	**71.8**	64.9	59.2	57.6	**65.0**

(b) Semantic8

Method	mIoU	OA	Man made	Natural	High veg.	Low veg.	Buildings	Hard scape	Artefacts	Cars
TML-PC [37]	0.391	0.745	0.804	0.661	0.423	0.412	0.647	0.124	0.000	0.058
TMLC-MS [22]	0.494	0.850	0.911	0.695	0.328	0.216	0.876	0.259	0.113	0.553
PointNet++ [40]	0.631	0.857	0.819	0.781	0.643	0.517	0.759	0.364	0.437	0.726
SnapNet [12]	0.674	0.910	0.896	0.795	0.748	0.561	0.909	0.365	0.343	0.772
SPGraph [26]	*0.762*	0.929	0.915	0.756	**0.783**	*0.717*	0.944	**0.568**	*0.529*	**0.884**
ConvPoint - RGB	0.750	**0.938**	**0.934**	**0.847**	0.758	0.706	0.950	*0.474*	0.432	*0.902*
ConvPoint - NoColor	0.726	0.927	0.918	0.788	0.748	0.646	**0.962**	0.451	0.442	0.856
ConvPoint - Fusion	**0.765**	*0.934*	*0.921*	*0.806*	*0.760*	**0.719**	*0.956*	0.473	**0.611**	0.877

(c) NPM3D

Name	mIoU	Ground	Building	Pole	Bollard	Trash can	Barrier	Pedestrian	Car	Natural
MSRRNetCRF	65.8	99.0	**98.2**	45.8	15.5	64.8	**54.8**	29.9	**95.0**	89.5
EdConvE	66.4	99.2	91.0	41.3	50.8	*65.9*	38.2	49.9	77.8	83.8
RFMSSF	56.3	99.3	88.6	47.8	67.3	2.3	27.1	20.6	74.8	78.8
MS3DVS	66.9	99.0	94.8	52.4	38.1	36.0	49.3	52.6	91.3	88.6
HDGCN	68.3	99.4	93.0	67.7	75.7	25.7	44.7	37.1	81.9	*89.6*
KP-FCNN [49]	**82.0**	**99.5**	94.0	*71.3*	*83.1*	**78.7**	47.7	**78.2**	*94.4*	**91.4**
ConvPoint - Fusion	*75.9*	**99.5**	*95.1*	**71.6**	**88.7**	46.7	*52.9*	*53.5*	89.4	85.4

to 16 samplings because a larger number does not significantly improve the scores.

This procedure shares similarities with the approaches used for image processing for test set augmentation such as image crops [45, 48]. The main difference resides in the fact that the output variation is not a result of input variation but is inherent to the network, i.e., the network is not deterministic.

6.3.2 Robustness to point cloud size and neighborhood size

In order to evaluate the robustness to test conditions that are different from the training ones, we propose two experiments. As stated in section 3, the definition of the convolutional layer does not require a fixed input size. However for a gain in performance and time, we trained the networks with minibatches, fixing the input size at training. We evaluate the influence of the input size at inference on the performance of a model trained with fix input size. Results are presented in Fig. 9 for the ModelNet40 classification dataset. Each curve (from blue to red) is an instance of the classification network, trained with 16, 32, ..., 2048 input points. The black dots are the scores for each model at their native input size. The dashed curve describe a theoretical model performing as well as the best model for each input size. Please note that the horizontal scale is a log scale and that each step is doubling the number of input points. A first observation is that almost each model performs the best at its native input size and that very few points are needed on ModelNet40 to reach decent performances: with 32 points, the performance already reaches to 85%. Besides, the larger the training size is, the more robust to size variation the model becomes. While the model trained on 32 points see its performance drop by 25% with $\pm 50\%$ points, the model trained with 2048 points still reaches 82% (a drop of 10%) with only 512 points (4 times less).

Table 5: ModelNet40 classification scores with a variation of neighborhood sizes k at test time.

(a) *First layer (default $k = 32$)*

/2	/1.5	$\times 1$	$\times 1.5$	$\times 2$
91.0	92.2	92.5	91.6	90.2

(b) *4^{th} layer (default $k = 16$)*

/2	/1.5	$\times 1$	$\times 1.5$	$\times 2$
90.9	91.2	92.5	91.9	91.4

Note: scores are computed with a 16-element spatial structure. The model was trained with the default configuration.

Second, in our formulation, the neighborhood size $|X|$ for each convolution layer remains a variable parameter after training, i.e., it is possible to change the neighborhood size at test time. It is the reason why, in equation (5), the normalizing weight $1/|X|$ (average with respect to input size) has been added to be robust to neighborhood size variation. In addition to the robustness provided by averaging, the variation of k somehow simulates a density variation of the points for the layer. We evaluate the robustness to such variations in table 5, on the classification dataset ModelNet40, with a single model trained with the default configuration (see Fig. 4). We report the impact of k on the first layer (table 5(a)) and on the fourth layer (table 5(b)). As expected, even though the best score is reached for the default k, the layer is robust to a high variation of k values: the overall accuracy loss is lower than 2% when k is 2 times larger or smaller. A particularly interesting feature is that the first layer, which extract local features, is more robust to a decreasing k, making the features more local than a increasing k. It is the opposite for the fourth layer: global features are more robust to an increase of k than a decreasing k.

Table 6: Training and inference timings.

(a) Comparison with PointCNN [29] on ModelNet40 and ShapeNet, computed with *Config. 1*

Dataset	Point num.	Batch size	PointCNN Training (epoch)	PointCNN Test	ConvPoint Training (epoch)	ConvPoint Test
ModelNet40	1024	32	58 s	8 s	42 s	7 s
		128	51 s	7 s	35 s	7 s
	2048	32	80 s	11 s	53 s	9 s
		128	-		45 s	8 s
ShapeNet	2048	4	1320 s	120 s	255 s	41 s
		8	-		185 s	37 s
		64	-		116 s	29 s

(b) Comparison between large segmentation network and fusion architecture, computed on the S3DIS dataset. Timings are given in milliseconds.

Batch size	1	2	4	8	16	32
Segmentation network						
Config. 1	93	69	46	40	39	38
Config. 2	155	95	89	81	-	-
Fusion architecture						
Config. 1	230	170	110	103	101	-
Config. 2	418	282	232	-	-	-

Config. 1: Middle-end configuration. Intel Xeon E5-1620 v3, 3.50 GHz, 8 cores, 8 threads + Nvidia GTX 1070 8Gb
Config. 2: Low-end configuration. Intel Core i7-5500U+, 2.40GHz, 2 cores, 4 threads + Nvidia GTX 960M 2Gb
The symbol "-" corresponds to setups (batch size and number of points) that exceed GPU memory.

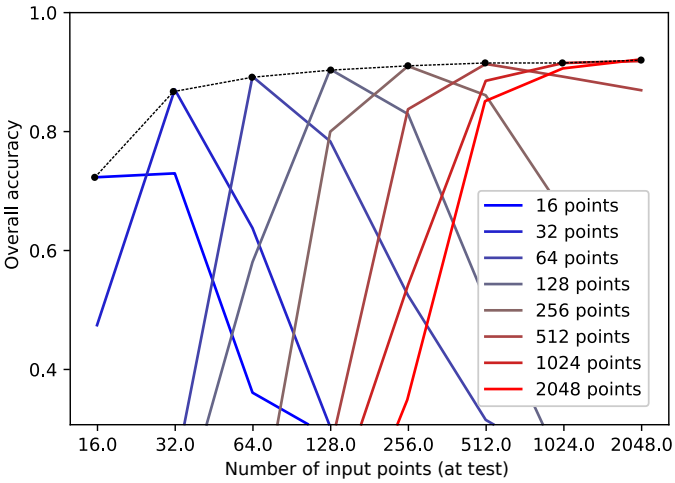


Figure 9: Influence of a varying number of input points at training time versus at test time on ModelNet40. Each colored curve shows the performance of a function of model trained with a given input point cloud size when tested on different numbers of input points.

7 Computation times and implementation details

In our experiments, the convolutional pipeline was implemented using Pytorch [1]. The neighborhood computation are done with NanoFLANN [9] and parallelized over CPU cores using OpenMP [16].

Table 6 presents the computation times. We consider two hardware configurations: a desktop workstation (Config. 1) and a low-end configuration, i.e., a gaming laptop (Config. 2). For instance, Config. 2 could correspond to a hardware specification for embedded devices.

Table 6(a) is a comparison with PointCNN [29] which was reported as the fastest in [29] among other methods. We used the code version available one the official PointCNN repository at the time of submission, and used the recommended network architecture. For both framework, we ran experiments on the *Config. 1* computer. On the ModelNet40 classification dataset, our model is about 30% faster than PointCNN for training, but inference times are similar. The difference is more significant on the ShapeNet segmentation dataset. For a batch size of 4, our segmentation framework is more than 5 times faster for training, and 3 times faster at test time.

Moreover, we also show that our ConvPoint is more memory efficient than the implementation of PointCNN. The "-" symbol in table 6 indicates batch sizes / numbers of points configurations that exceed the GPU memory. For example, we can train the classification model with 2048 points and batch size of 128, which is not possible with PointCNN (on an Nvidia GTX 1070 GPU). The same goes on ShapeNet where we can train with a batch size of 64 while PointCNN already uses too much GPU memory with batch size 8.

In table 6(b), we report timings for the large-scale segmentation network and the fusion architecture, with a point cloud size of 8192. Note that for training this network we used NVidia GTX 1080Ti GPU. These timings represent the inference time only (neighborhood computation and convolutions), not data loading. We first observe that even the fusion architecture (two segmentation networks and a fusion module) can be run on the small configuration. Second, as for CNN with images, we benefit from using batches which reduce per point cloud computation time. Finally, our implementation is efficient given that for the segmentation network, we are able to process from 100,000 points (Config. 2) to 200,000 points (Config. 1) per second.

8 Discussion and limitations

Convolutional layer First, our convolution design is agnostic to the object scales, due to neighborhood normalization to the unit ball. It is of interest for non metric data such as CAD models or photogrammetric point clouds where scales are not always available. On the contrary, in metric scans, object sizes are valuable information (e.g., humans have almost all the time similar sizes) but removing the normalization would cause the kernel and the input points to have different volumes.

Second, an alternative is to use a fixed-radius neighborhood instead of a fix number of neighbors. As pointed out in [49], the resulting features would be more related to geometry and less to sampling. However, the actual code optimization to speed up computation such as batch training would be inapplicable due to a variable number of neighbors in a batch.

Input features Another perspective is to explore the use of precomputed features as inputs. In this study, we only use raw data for network inputs: RGB colors when available, or all features set to 1 otherwise. In the future, we will work on feeding the networks with extra features such as normals or curvatures.

Network architecture Finally, we proposed two networks architectures that are widely inspired from computer vision models. It would be interesting to explore further variations of network architectures. As our formulation generalizes the discrete convolution, it is possible to transpose more CNN architectures, such as residual networks.

9 Conclusion

In this paper, we presented a new CNN framework for point cloud processing. The proposed formulation is a generalization of the discrete convolution for sparse and unstructured data. It is flexible and computationally efficient, which it makes it possible to build various network architectures for classification, part segmentation and large-scale semantic segmentation. Through several experiments on various benchmarks, real and simulated, we have shown that our method is efficient and at state of the art.

References

[1] P. Adam, G. Sam, C. Soumith, C. Gregory, Y. Edward, D. Zachary, L. Zeming, D. Alban, A. Luca, and L. Adam. Automatic differentiation in pytorch. In *Proceedings of Neural Information Processing Systems*, 2017.

[2] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017.

[3] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[4] M. Atzmon, H. Maron, and Y. Lipman. Point Convolutional Neural Networks by Extension Operators. *arXiv preprint arXiv:1803.10091*, 2018.

[5] M. Aubry, U. Schlickewei, and D. Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1626–1633. IEEE, 2011.

[6] N. Audebert, B. Le Saux, and S. Lefèvre. Semantic Segmentation of Earth Observation Data Using Multimodal and Multi-scale Deep Networks. In *ACCV*, Taipei, Taiwan, Nov. 2016.

[7] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer. 3D Point Cloud Classification and Segmentation using 3D Modified Fisher Vector Representation for Convolutional Neural Networks. *arXiv preprint arXiv:1711.08241*, 2017.

[8] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[9] J. L. Blanco and P. K. Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. `https://github.com/jlblancoc/nanoflann`, 2014.

[10] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016.

[11] A. Boulch. Generalizing discrete convolutions for unstructured point clouds. In *Eurographics Workshop on 3D Object Retrieval (3DOR)*, 2019.

[12] A. Boulch, J. Guerry, B. Le Saux, and N. Audebert. SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks. *Computers & Graphics*, 2017.

[13] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[14] M. M. Bronstein and I. Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1704–1711. IEEE, 2010.

[15] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun. Spectral networks and locally connected networks on graphs. pages http–openreview, 2014.

[16] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, Jan. 1998.

[17] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. pages 764–773, 2017.

[18] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[19] B. Graham, M. Engelcke, and L. van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018.

[20] B. Graham and L. van der Maaten. Submanifold Sparse Convolutional Networks. *arXiv preprint arXiv:1706.01307*, 2017.

[21] T. Hackel, N. Savinov, L. Ladicky, J. Wegner, K. Schindler, and M. Pollefeys. Smeantic3D.net: A new large scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume IV-1-W1, pages 91–98, 2017.

[22] T. Hackel, J. D. Wegner, and K. Schindler. Fast semantic segmentation of 3D point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3), 2016.

[23] Q. Huang, W. Wang, and U. Neumann. Recurrent Slice Networks for 3D Segmentation of Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2626–2635, 2018.

[24] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 1999.

[25] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3D point cloud models. In *IEEE International Conference on Computer Vision (ICCV)*, pages 863–872. IEEE, 2017.

[26] L. Landrieu and M. Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] J. Li, B. M. Chen, and G. Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[29] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution On X-Transformed Points. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 828–838. Curran Associates, Inc., 2018.

[30] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. FPNN: Field probing neural networks for 3D data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016.

[31] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[32] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[33] H. Ling and D. W. Jacobs. Shape classification using the inner-distance. *IEEE transactions on pattern analysis and machine intelligence*, 29(2):286–299, 2007.

[34] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[35] D. Maturana and S. Scherer. Voxnet: A 3D convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[36] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[37] J. A. Montoya-Zegarra, J. D. Wegner, L. Ladickỳ, and K. Schindler. Mind the gap: modeling local and global context in (road) networks. In *German Conference on Pattern Recognition*, pages 212–223. Springer, 2014.

[38] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[39] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view CNNs for object classification on 3D data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[40] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5105–5114, 2017.

[41] X. Roynard, J.-E. Deschaud, and F. Goulette. Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, 2018.

[42] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[43] Y. Shen, C. Feng, Y. Yang, and D. Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 4, 2018.

[44] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] H. Su, V. Jampani, D. Sun, S. Maji, V. Kalogerakis, M.-H. Yang, and J. Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. *arXiv preprint arXiv:1802.08275*, 2018.

[47] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[48] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[49] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *arXiv preprint arXiv:1904.08889*, 2019.

[50] C. Wang, B. Samari, and K. Siddiqi. Local spectral graph convolution for point set feature learning. pages 52–66, 2018.

[51] D. Z. Wang and I. Posner. Voting for Voting in Online Point Cloud Object Detection. In *Robotics: Science and Systems*, volume 1, page 5, 2015.

[52] S. Wang, S. Suo, W.-C. M. A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

[53] W. Wang, R. Yu, Q. Huang, and U. Neumann. SGPN: Similarity group proposal network for 3D point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018.

[54] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[55] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[56] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. pages 87–102, 2018.

[57] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, L. Guibas, et al. A scalable active framework for region annotation in 3D shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016.

[58] L. Yi, H. Su, X. Guo, and L. J. Guibas. SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.