HAL open science

FKAConv: Feature-Kernel Alignment for Point Cloud Convolution

Alexandre Boulch, Gilles Puy, Renaud Marlet

▶ **To cite this version:**

Alexandre Boulch, Gilles Puy, Renaud Marlet. FKAConv: Feature-Kernel Alignment for Point Cloud Convolution. 15th Asian Conference on Computer Vision (ACCV), Nov 2020, Kyoto, Japan. pp.381-399, 10.1007/978-3-030-69525-5_23 . hal-04496450

HAL Id: hal-04496450

https://hal.science/hal-04496450v1

Submitted on 8 Mar 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

FKAConv: Feature-Kernel Alignment for Point Cloud Convolution

Alexandre Boulch
valeo.ai, Paris

Gilles Puy
valeo.ai, Paris

Renaud Marlet
valeo.ai, Paris
LIGM, Ecole des Ponts,
Univ Gustave Eiffel,
CNRS, Marne-la-Vallée, France

arXiv:2004.04462v3 [cs.CV] 24 Nov 2020

Abstract

Recent state-of-the-art methods for point cloud processing are based on the notion of point convolution, for which several approaches have been proposed. In this paper, inspired by discrete convolution in image processing, we provide a formulation to relate and analyze a number of point convolution methods. We also propose our own convolution variant, that separates the estimation of geometry-less kernel weights and their alignment to the spatial support of features. Additionally, we define a point sampling strategy for convolution that is both effective and fast. Finally, using our convolution and sampling strategy, we show competitive results on classification and semantic segmentation benchmarks while being time and memory efficient.

1 Introduction

Convolutional Neural Networks (CNNs) have been a breakthrough in machine learning for image processing [7, 19]. The discrete formulation of convolution allows a very efficient processing of grid-structured data such as images in 2D or videos in 3D. Yet a number of tasks require processing unstructured data such as point clouds, meshes or graphs, with application domains such as autonomous driving, robotics or urban modeling. However discrete convolution does not directly apply to point clouds as 3D points are not usually sampled on a grid.

The most straightforward workaround is to voxelize the 3D space to use discrete CNNs [31]. However, as 3D points are usually sampled on a surface, most of the voxels are empty. For efficient large-scale processing, a sparse formulation is thus required [37, 60]. Other deep learning approaches generalize convolution to less structured data, such as graphs or meshes [5, 41], but applying them to point clouds requires addressing the issue of sensible graph construction first.

Deep-learning techniques that directly process raw data have been developed to overcome the problem of point cloud pre-processing [33, 50]. Just as for structured data, such networks are usually designed as a stack of layers and are optimized using stochastic gradient descent and back-propagation. Key issues when designing these networks include speed and memory efficiency.

In this context, we propose a new convolution method for point cloud processing. It is a mixed discrete-continuous formulation that disentangles the geometry of the convolution kernel and the spatial support

FKAConv code is available on the valeo.ai github: `https://github.com/valeoai/FKAConv`

of the features: using a geometry-less kernel domain, we stick to a discrete convolution scheme, which is efficient and has been successful on grid data; the spatial domain however keeps its continuous flavor, as point clouds are generally sampled on manifolds.

Our contributions are the following: (1) we provide a formulation to relate and analyze existing point convolution methods; (2) we propose a new convolution method (FKAConv) that explicitly separates the estimation of geometry-less kernel weights and their alignment to the spatial support of features; (3) we define a point sampling strategy for convolution that is both efficient and fast; (4) experiments on large-scale datasets for classification and semantic segmentation show we reach the state of the art, while being memory and time efficient.

2 Related work

Projection in 2D. Some methods project the point cloud in a space suitable for using standard discrete CNNs. 2D CNNs have been use for 3D data converted as range images [13, 29] or viewed from virtual viewpoints [4, 21, 44]. As neighboring points in the resulting image can be far away in 3D space, 2D CNNs often fail to capture well 3D relations. 2D CNNs can also be applied locally to point-specific neighborhoods by projecting data on the tangent plane [45]; the result is then highly dependant on the tangent plane estimation. Other approaches use a volumetric data representation, such as voxels [31, 34, 39, 54]. These approaches however suffer from encoding mostly empty volumes, calling for sparsity handling, e.g., with octree-based 3D-CNNs [37] or sparse convolution [11, 12].

Graph convolution, geometric deep learning. Graph Neural Networks (GNNs) [5, 41] extend neural networks to irregular structures (not on a grid), using edges between nodes for message passing [10, 24] or defining convolution in the spectral domain [6, 9, 18]. Point convolution using GNNs requires first explicitly building a graph from the point cloud [36]. To scale to large point clouds, SPG [20] defines a graph over nodes corresponding to point segments. In contrast, our approach directly applies to the raw point cloud, with no predefined relation between points, somehow making point association as part of the method.

MLP processing. PointNet [33] directly processes point coordinates with a multi-layer perceptron (MLP), gathering context information with a permutation-invariant max-pooling. PointNet++ [35] and So-Net [22] reduce the loss of local information due to subsampling with a cascade of MLPs at different scales.

Point convolution. A first line of work considers an explicit spatial location for the kernel, in the same space as the point cloud. Kernel elements can be located on a regular grid (voxels) [16], at the vertices of a polyhedron [47] or randomly sampled and optimized at training [3]. In KPConv [47], an adjustment of the kernel locations may also be predicted at test time to better fit the data.

Another type of approaches models kernel locations implicitly. The kernel can be a family of polynomials like in SpiderCNN [55], or it can be estimated with an MLP, like in PCCN [50], RSConv [27] or PointConv [53]. The weights of the input features are then directly estimated based on the local geometry of points. In contrast, we learn the weights of a discrete kernel and, at inference time, we only estimate the spatial relation between the kernel and input points. PointConv [53] reweights the input features based on local point densities. Our method reaches state-of-the-art performances without the need of such a mechanism.

Finally, PointCNN [23] shares apparent similarities with our work as one of its main components is the estimation of a matrix, that actually differs from ours. Besides, geometric information in [23] is lifted to the feature space and used as additional features. Our work shows it is sufficient to use the geometry only for features-kernel alignment, mimicking the discrete convolution on a regular grid.

Our approach lies in between these lines of work. On the one hand, our kernel weights are explicitly modeled as in [3, 16, 47], which gives a discrete flavor to our method; on the other hand, we estimate a transformation of input points to apply the convolution as in [27, 50, 53], which operates in the continuous domain, avoiding kernel spatialization. The key is that, contrary to fully-continuous approaches that re-estimate at inference time how to weigh given sets of points to operate the convolution, we estimate separately a kernel while learning, and we predict the relation between the kernel and input points while testing. Besides, we perform the convolution with a direct matrix multiplication rather than getting indirectly results from a network output. This separation and the explicit matrix multiplication (outside the network) allows a better learning of kernel weights and spatial relations, without the burden and inaccuracy of estimating their composition, resulting in a time and memory efficient method.

Point sampling. Like PointNet [33], several methods maintain point clouds at full resolution during the whole processing [26, 28, 56]. These methods suffer from a high memory cost, which requires to either limit the input size [33, 56], split the input into parts [26], or use a coarse voxel grid [28]. Other approaches [3, 23, 35], as ours, use an internal sub-sampling of the point cloud. The choice of sampled points forming a good support is a key step for this reduction. *Furthest point sampling* (FPS) [35], where points are chosen iteratively by selecting the furthest point from all the previously picked points at each iteration, yields very good performance but is slow and its performance depends on the initialization. In [56], point sampling is based on a learned attention, which induces a high memory cost. Our sampling strategy, based on the quantization of the 3D space, ensures a good sampling of the space, like FPS, and is fast and memory efficient.

3 A general formulation of point cloud convolution

We base our convolution formulation on the discrete convolution used in image or voxel grid processing. The formulation is general enough to cover a wide range of state-of-the-art convolution methods for point clouds, and to relate them.

3.0.1 Discrete convolution.

Let F be the dimension of the input feature space, d the spatial dimension (e.g., 2 for images, 3 for voxel grids), $\mathbf{K}$ the convolution kernel, and $\mathbf{f}$ the input features. The classical discrete convolution, noted $\mathbf{h}$, is:

$$\mathbf{h}[n] = \sum_{f \in \{1,\dots,F\}} \sum_{m \in \{-M/2,\dots,M/2\}^d} \mathbf{K}_f[m]\, \mathbf{f}_f[n+m], \tag{1}$$

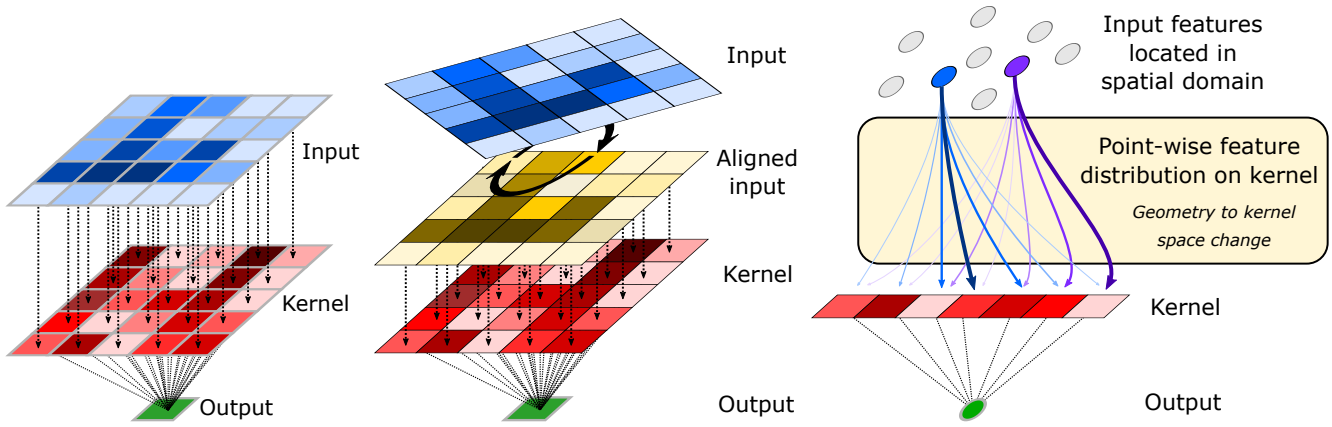
where M^d is the grid kernel size, f indexes the feature space, n is the spatial index, and $\mathbf{K}_f[m]$ and $\mathbf{f}_f[n+m]$ are scalars. Defining vectors $\mathbf{K}_f = (K_f[m],\, m \in \{-M/2, \dots, M/2\}^d\})$ and $\mathbf{f}_f(n) = (\mathbf{f}_f[n+m],\, m \in \{-M/2, \dots, M/2\}^d)$, we can highlight the separation between the kernel space ($\mathbf{K}$) and the feature space ($\mathbf{f}$):

$$\mathbf{h}[n] = \sum_{f \in \{1,\dots,F\}} \underbrace{\mathbf{K}_f^\top}_{\text{Kernel space}} \underbrace{\mathbf{f}_f(n)}_{\text{Feature space}} . \tag{2}$$

The kernel $\mathbf{K}_f$ and the features $\mathbf{f}_f(n)$ are perfectly aligned: the grid index m associates a kernel element $\mathbf{K}_f[m]$ with a single input element $\mathbf{f}_f[n+m]$ (Fig. 1(a)).

3.0.2 Point convolution.

To generalize this discrete convolution to point clouds, we first consider a hypothetical misalignment between the feature and kernel spaces, assuming the feature grid is rotated with respect to the kernel grid (Fig. 1(b)),



(a) Discrete conv. aligned (b) Discrete conv. misaligned (c) FKAConv point convolution

Figure 1: Kernel-input alignment for grid inputs (a,b) and point clouds (c).

thus obfuscating the correspondence between kernel elements and feature elements. Yet, provided that the rotation matrix $\mathbf{A} \in \mathbb{R}^{M^d} \times \mathbb{R}^{M^d}$ is known, the correspondences can be recovered by rotating the support points of features:

$$\mathbf{h}[n] = \sum_{f \in \{1,\dots,F\}} \mathbf{K}_f^\top \mathbf{A}\, \mathbf{f}_f(n). \tag{3}$$

This equation actually holds in a more general setting, with an arbitrary linear transformation between the feature space and the kernel space; $\mathbf{A}$ is then the *alignment matrix* that associates the feature values to the kernel elements.

The discrete convolution on a regular grid becomes a particular case of Equation (3), with $\mathbf{A} = \mathbf{I}_{M^d}$, the identity matrix. In the case of a point cloud, $\mathbf{f}_f(n)$ is the feature associated to the point at spatial location n, typically computed on a neighborhood $\mathbf{N}[n]$. These features are generally not grid-aligned. But Equation (3) can still apply, provided we can estimate an alignment matrix $\mathbf{A}$ that distributes each input point onto the kernel elements (Fig. 1(c)).

In this context, a fixed matrix $\mathbf{A}$ is suboptimal as it cannot cope well with both a regular grid ($\mathbf{A} = \mathbf{I}_{M^d}$) and arbitrary point configurations in a point cloud. $\mathbf{A}$ thus has to be a function of the input points, which in practice have to be limited to neighbors $\mathbf{N}[n]$ at location n. The convolution becomes:

$$\mathbf{h}[n] = \sum_{f \in \{1,\dots,F\}} \mathbf{K}_f^\top \mathbf{A}(\mathbf{N}[n])\, \mathbf{f}_f(n). \tag{4}$$

It is a mixed discrete-continuous formulation: $\mathbf{K}_f$ and $\mathbf{f}_f(n)$ have a discrete support and continuous values, while $\mathbf{A}(\mathbf{N}[n])$ provides a continuous mapping.

3.0.3 Analysis of exiting methods.

This formulation happens to be generic enough to describe a range of existing methods for point convolution [3, 23, 43, 47, 50].

Using spatial kernel points. The most common approach to discrete convolution on a point cloud assigns a spatial point to each kernel element. The distribution of features on kernel elements is then based on the distance between kernel points and points in $\mathbf{N}[n]$, corresponding to an association matrix $\mathbf{A}$ invariant by rotation. A simple method would be to assign the features to the nearest kernel point, but it is unstable as a small perturbation in the point position may result in a different kernel point attribution. A workaround is to

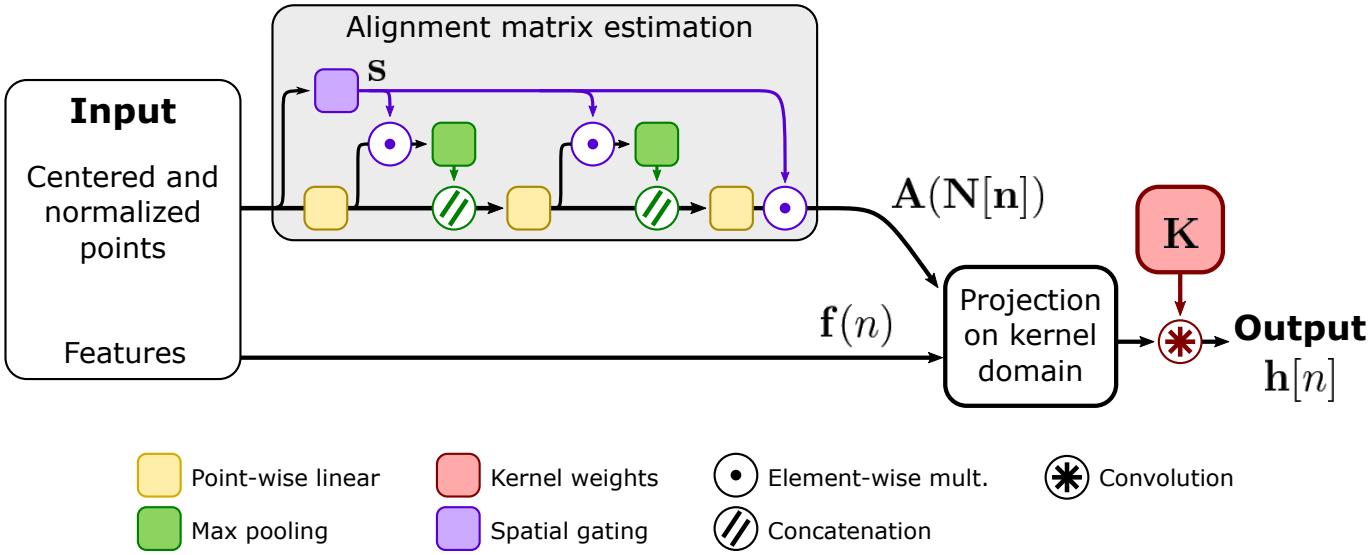


Figure 2: FKAConv convolutional layer.

distribute the input points to several close kernel points. In SplatNet [43], an interpolation distribute points onto the kernel space. However, this handcrafted assignment is arbitrary and heavily relies on the geometry of kernel points. KPConv [47] chooses to distribute the input points over all the neighboring kernel points, with a weight inversely proportional to their distance to kernel points. Moreover, KPConv allows deformable kernels, for which local shifts of kernel points are estimated, offering more adaptation to input points. Yet, this handcrafted distribution is still arbitrary and still relies on the geometry of kernel points. ConvPoint [3] randomly samples the kernel points, and their position is learned along with an assignment function $\mathbf{A}$, with an MLP is applied to the kernel points represented in the coordinate system centered on the input points. All these methods [3, 43, 47] raise the issue of defining and optimizing the position of kernel points.

Feature combination and geometry lifting. In PointCNN [23], geometric information is extracted with an MLP_δ, parameterized by δ, and concatenated with the input features to create mixed spectral-geometric features. The summands in Equation (4) become $\mathbf{K}^\top \mathbf{A}(\mathbf{N}[n])[\mathbf{f}_f(n), \text{MLP}_\delta(\mathbf{N}[n])]$.

Joint estimation of $\mathbf{K}^\top \mathbf{A}(\mathbf{N}[n])$. In fully implicit approaches [3, 47, 50], MLPs are used to directly estimate the weights $\mathbf{W}(n)$ to apply to input features $\mathbf{f}_f(n)$, i.e., not separating $\mathbf{W}(n)$ into a product $\mathbf{K}^\top \times \mathbf{A}(\mathbf{N}[n])$, and thus mixing estimations in the spatial and feature domains.

Kernel separation. As acknowledged by the authors of PCCN themselves [50], a direct estimation of $\mathbf{W}(n) = \mathbf{K}^\top \mathbf{A}(\mathbf{N}[n])$ is too computationally expensive to be used in practice. Instead, they resort to an implementation which falls into our formulation: for N_{out} output channels, they consider N_{out} parallel convolution layers with a size-1 kernel, corresponding to using a different $\mathbf{A}$ for each filter. In PointCNN [23], the extra features generated by the geometry lifting induce a larger kernel (1/4 more weights with default parameters [23]), thus an increased memory footprint. To overcome this issue, [23] also chooses to factorize the kernel as the product of two smaller matrices. Notice that this kernel separation trick can be implemented in any method that uses explicit kernel weights. Although we could as well, we do not need to resort to that trick in our method.

4 Our method: estimating a feature-kernel alignment

Our own convolution method is also based on Equation (4). However, contrary to preceding approaches, we do not use kernel points. Instead, we estimate a soft alignment matrix $\mathbf{A}$ based on the coordinates of neighboring points in $\mathbf{N}[n]$. Our convolutional layer is illustrated on Figure 2.

Neighborhood normalization. To be globally invariant to translation, all coordinates of the points of $\mathbf{N}[n]$ are expressed in the local coordinates system of n. This is particularly important for scene segmentation: the network should behave the same way for similar objects at different spatial locations. Please notice that it is not the case in RSConv [27] where absolute coordinates are used, making it appropriate for shape analysis, not for scene processing.

$\mathbf{N}[n]$ is typically defined as the k-nearest neighbors (k-NNs) of n, or as all points in a ball around n. Both definitions have pros and cons. Using k-NNs is relatively fast, but the radius of the encompassing ball is (potentially highly) variable. As observed in [47], it may degrade spatial consistency compared to using a ball with a fixed radius. But searching within a radius is slower and yields (potentially widely) different numbers of neighbors, requiring strategies to deal with variable sizes, e.g., large tensor sizes and size tracking as in [47].

We propose an intermediate approach based on the k nearest neighbors, with a form a rescaling. As opposed to [3,23,35], we do not normalize the neighborhood to a unit sphere regardless of its actual size in the original space. We estimate a normalization radius r_t of the neighborhood at the layer level using the exponential moving average of Eq. (9) computed at training time, where t is the update step, m is a momentum parameter and $\hat{r}_t$ is the average neighborhood radius of the current batch. Let $\mathbf{q}$ be the support point associated to n, and $\mathbf{p}_i$ the i-th point of $\mathbf{N}[n]$. The points $(\hat{\mathbf{p}}_i)_i$ actually used for the estimating $\mathbf{A}$ are the points $(\mathbf{p}_i)_i$ centered and normalized using $\mathbf{q}$ and r_t as follows:

$$r_t = \hat{r}_t * m + r_{t-1} * (1 - m), \tag{5}$$

$$\hat{\mathbf{p}}_i = (\mathbf{p}_i - \mathbf{q})/r_t. \tag{6}$$

At inference time, this normalization ensures that all neighborhood are processed at the same scale while on average, neighborhoods are mapped to the unit ball.

Gating mechanism on distance to support point. While solving the problem of the neighborhood scale, this normalization strategy does not prevent points far away from the support point (the neighborhood center) to influence negatively the result. One could use hard-thresholding on the distance based on the estimated normalization radius r to filter these points, but this approach may cut too much information from the neighborhood, particularly in the case of high variance in neighborhood radii. Instead, we propose a gating mechanism to reduce, if needed, the effect of such faraway points. Given $(\hat{\mathbf{p}}_i)_i$ as defined in Equation (10), the spatial gate weight $\mathbf{s} = (s_i)_i$ satisfies

$$s_i = \sigma(\beta - \alpha||\hat{\mathbf{p}}_i||_2), \tag{7}$$

where $\sigma(\cdot)$ is the sigmoid function, β is the cutoff distance (50% of the maximal value) and α parametrize the slope of the transition between 0 (points filtered out) and 1 (points kept). Both α and β are learned layer-wise.

Estimation of $\mathbf{A}$. As underlined in [33], a point cloud is invariant by point permutation: changing the order of points should not change the point cloud properties. Hence, the product $\mathbf{A}\ \mathbf{f}_f(n)$ must be invariant by permutation of the inputs. This can be achieved by estimating independently each line $\mathbf{A}_j$ of the matrix $\mathbf{A}$ using only the corresponding point $\mathbf{p}_j \in \mathbf{N}[n]$, with an MLP shared accross all points. But it does not

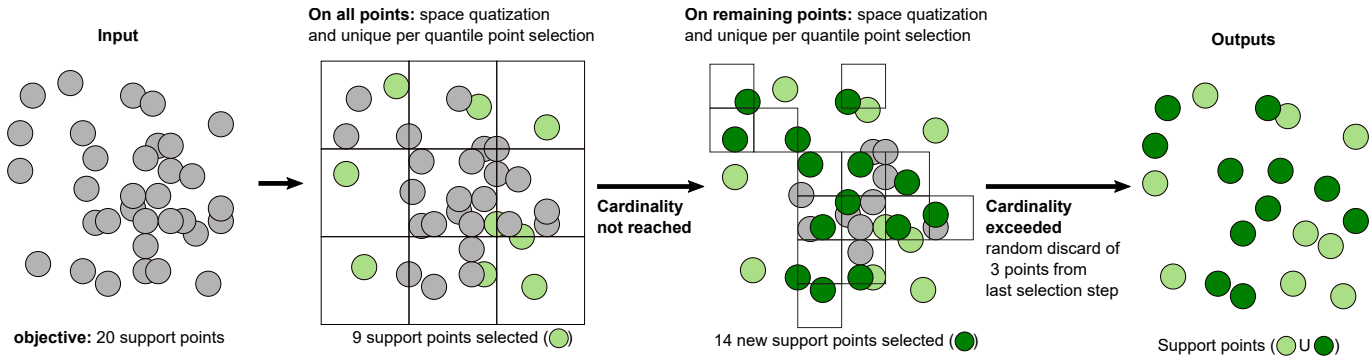


Figure 3: Point sampling with space quantization.

take the neighborhood into account, and thus may ignore useful information such as the local normal or curvature. To address point permutation invariance, PointNet [33] uses a max-pooling operation. Likewise, we use a three-layer point-wise MLP with max-pooling after the first two layers. To reduce the influence of outliers, max-pooling inputs are weighted with $\mathbf{s}$. The output is then concatenated to the point-wise features and given as input to the next fully-connected layer. This series of computations is illustrated in Figure 2 in the block called "alignment matrix estimation."

5 Efficient point sampling with space quantization

Networks architectures for point cloud processing operates at full resolution through the entire network [50], or have an encoder/decoder structure [3, 23] similar to networks used in image processing, e.g., U-Net [38]. While the former maintain a maximum of information through the network, the later are usually faster as convolutions are applied to smaller point sets. However, decreasing the size of the point cloud requires to select *the support points*, i.e., the points at the center of the neighborhoods used in the convolution.

PointNet++ [35] introduces farthest point sampling, an iterative sampling procedure where the next sampled point is the farthest from the already picked points. The main advantage of this sampling is to ensure a somewhat spatially uniform distribution which favors extreme points (e.g., at wing extremities for planes) as they are usually important for shape recognition. However, it requires to keep track of distances between all pairs of points, which is costly and increases the computation time, in particular when dealing with large point clouds.

In ConvPoint [3], the point-picking strategy only takes into account seen and unseen points, without distance consideration. Points are randomly picked among points that were not previously seen (picked points and points in the neighborhood of these points). While being much faster than farthest point sampling, it appears to be less efficient (see experiments in Section 6). In particular, the sampling is dependent of the neighborhood size: the sampling is done outside the neighborhoods of the previously picked support points. A very small neighborhood size reduces the method to a pure random sampling.

Space quantization. We propose an alternative approach that ensures a better sampling than [3] while being much faster than [35]. The procedure is illustrated on Fig. 3. We discretize the space using a regular voxel grid. Each point is associated to the grid cube it falls in. In each non-empty grid cube, one point is selected. We continue with the non-selected points and a voxel size divided by two, and repeat the process until the desired number of sampled points is reached or exceeded. In the later case, some points selected at the last iteration are discarded at random to reduce the cardinality of Q, the set selected points.

Quantization step estimation. Our approach is voxel-size dependent. On the one hand, a coarse grid leads to many iterations in the selection procedure, at the expense of computation time. On the other hand, a fine grid reduces to random sampling. Finding the optimal voxel size could be achieved using a exhaustive search (for $|Q|$ filled voxels at a single quantization step), but it is very slow. Instead, we propose to estimate the voxel size via a rule of thumb derived by considering a simple configuration where a plane is intersecting a cube of unit length divided by a voxel grid of size $a \times a \times a$. If the plane is axis aligned, it intersects a^2 voxels. A sensible sampling would pick a support point in each intersected voxel, *i.e.*, $|Q| = a^2$. This indicates that letting the length $v = 1/a$ of a voxel be proportional to $1/\sqrt{|Q|}$ is a reasonable choice for the voxel size. We found experimentally that choosing the diagonal length of the bounding box of the point cloud, denoted hereafter by diag, as factor of proportionality is usually a good choice (see Section 6.2). The voxel size is thus set to

$$v = \mathrm{diag}/\sqrt{|Q|}. \tag{8}$$

6 Experiments

In this section, we evaluate our convolutional layer on shape classification, part segmentation and semantic segmentation, reaching the state of the art regarding task metrics while being efficient regarding computation time and memory usage.

Network architectures. In our experiments, we use a simple yet effective residual network for classification and semantic segmentation. We mimic the architecture of [47], except that ours is designed for k-NN convolution, i.e., we do not need to add phantom points and features to equalize the size of data tensor due to a variable number of points in radius search. The network has an encoder-decoder structure. The encoder is composed of an alternation of residual blocks maintaining the resolution and residual blocks with down-sampling. The decoder is a stack of fully-connected and nearest-neighbor upsampling layers. The classification network is the encoder of the previously described network followed by a global average pooling. For large scale semantic segmentation, we use either input modality dropout [47] or dual network fusion [3], as indicated in tables.

Experimental setup. Our formulation (and code) allows a variable input size, but in order to use optimization with mini-batches, with train the networks with fixed input sizes. As every operations of FKAConv are differentiable, all parameters are optimized via gradient descent (including the spatial gating parameters α's and β's). Finally, we use a standard cross-entropy loss.

6.1 Benchmark results

Shape classification. The classification task is evaluated on ModelNet40 [54]. As the spatial pooling process is stochastic, multiple predictions with the same point cloud might lead to different outcomes. We aggregate 16 predictions for each point cloud and select the most predicted shape (we use a similar approach for part segmentation). On the classification task (Table 1(a)), we present average (and best) results over five runs. For fair comparison, we train with 1024 (resp. 2048) points. We rank first (resp. second) among the method trained with 1024 (resp. 2048) points. We mainly observe that increasing the number points of reduces the standard deviation of the performances.

Part segmentation. On ShapeNet [57], the network is trained with 2048 input points and 50 outputs (one for each part). The loss and scores are computed per object category (16 object categories with 2- to 6-part labels). The results are presented in Table 1(b). We rank among the best methods: top-2 or top-5 depending on the metric used, i.e., mean class intersection over union (mcIoU) or instance average intersection over union (mIoU); we are only 0.3 point mcIoU and 0.7 point mIoU behind the best method. It is interesting

Table 1: Classification and part segmentation benchmarks.

(a) ModelNet40

Methods	Num. points	OA	AA
Mesh or voxels			
Subvolume [34]	-	89.2	-
MVCNN [44]	-	90.1	-
Points			
DGCNN [52]	1024	92.2	**90.2**
PointNet [33]	1024	89.2	86.2
PointNet++ [35]	1024	90.7	-
PointCNN [23]	1024	92.2	88.1
ConvPoint [3]	2048	92.5	89.6
KPConv [47]	2048	**92.9**	-
Ours		Average±std. (best run)	
FKAConv	1024	92.3±0.2 (92.5)	89.6±0.3 (89.9)
	2048	92.5±0.1 (92.5)	89.5±0.1 (89.7)

(b) ShapeNet

Method	mcIoU	mIoU
PointNet++ [35]	81.9	85.1
SubSparseCN [12]	83.3	86.0
SPLATNet [43]	83.7	85.4
SpiderCNN [55]	81.7	85.3
SO-Net [22]	81.0	84.9
PCNN [2]	81.8	85.1
KCNet [42]	82.2	83.7
SpecGCN [49]	-	85.4
RSNet [17]	81.4	84.9
DGCNN [52]	82.3	85.1
SGPN [51]	82.8	85.8
PointCNN [23]	84.6	86.1
ConvPoint [3]	83.4	85.8
KPConv [47]	**85.1**	**86.4**
FKAConv (Ours)	84.8	85.7

to notice that we are as good as or better than several methods for which the convolution falls into our formalism, such as ConvPoint [3] or SPLATNet [43].

Semantic segmentation Three datasets are used for semantic segmentation corresponding to three different use cases. S3DIS [1] is an indoor dataset acquired with an RGBD camera. The evaluation is done using a 6-fold cross validation. NPM3D [40] is an outdoor dataset acquired in four sites using a lidar-equipped car. Finally, Semantic8 [14] contains 30 lidar scenes acquired statically. NPM3D and Semantic8 are datasets with hidden test labels. Scores in the tables are reported from the official evaluation servers.

We use 8192 input points but, as subsampling the whole scene produces a significant loss of information, we select instead points in vertical pillars with a square footprint of 2 m for S3DIS, and 8 m for NPM3D and Semantic8. The center point of the pillar is selected randomly at training time and using a sliding window at test time. If a point is seen several times, the prediction scores are summed and the most probable class is selected afterward.

The results are presented in Fig. 4 and Table 2. We use S3DIS (Table 2(a)) to study the impact of the training strategy. As underlined in [3, 47], direct learning with colored points yields a model relying too much on color information, at the expense of geometric information. We train three models. The first is the baseline model trained with color information, the second uses color dropout as in [47], and the third is a dual model with a fusion module [3]. We observe that fusion gives the best results. In practice, the model trained with modality dropout tends to select one of the two modalities, either color or geometry, depending on what modality gives the best results. On the contrary, the fusion technique uses two networks each trained with a different modality, resulting in a lot larger network, but ensuring that the information of both modalities is taken into account.

Our network is second on S3DIS, first on NPM3D and third on Semantic8. On S3DIS, it is the best approach for 3 out of 13 categories and it performs well on the remaining ones. We are only outperformed by KPConv, which is based on radius search. On NPM3D, we reach an average intersection over union (av. IoU) of 82.7, which is 0.7 point above the second best method. On Semantic8, we place third according to average IoU, and first on overall accuracy among the published and arXiv methods. We obtain the best scores in 3 out 8 categories (the top-3 for 6 categories out of 8). More interestingly, we exceed the scores of ConvPoint [3] on 5 categories. The only downside is the very low score on the category of artefacts. One possible explanation could be that the architecture used in this paper (the residual network) is not suitable to learn a reject class (the artefact class is mainly all the points that do not belong to the 7 other classes, i.e., pedestrians but also

Table 2: Semantic segmentation benchmarks.

(a) S3DIS

Method	Search	IoU	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
Pointnet [33]	Knn	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 [17]	-	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 [50]	-	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 [20]	Super pt.	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 [23]	Knn	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
PointWeb [59]	Knn	66.7	93.5	94.2	80.8	52.4	41.3	64.9	68.1	71.4	67.1	50.3	62.7	**62.2**	58.5
ShellNet [58]	Knn	66.8	90.2	93.6	79.9	60.4	44.1	64.9	52.9	71.6	84.7	53.8	64.6	48.6	59.4
ConvPoint [3]	Knn	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
KPConv [47]	Radius	**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
FKAConv (Ours *RGB only*)	Knn	64.9	94.0	97.8	80.5	38.5	48.5	49.8	68.0	79.4	70.7	48.4	43.7	62.9	61.4
FKAConv (Ours *RGB drop.*)	Knn	66.6	94.4	97.8	81.5	38.7	43.3	56.4	71.6	80.2	71.8	63.5	54.1	50.6	62.5
FKAConv (Ours *fusion*)	Knn	68.4	94.5	**98.0**	82.9	41.0	46.0	57.8	74.1	**77.7**	71.7	65.0	60.3	55.0	**65.5**
Rank		2	3	1	2	7	4	6	2	1	2	3	4	5	1

(b) NPM3D

Method	Av.IoU	Ground	Building	Pole	Bollard	Trash can	Barrier	Pedestrian	Car	Natural
RF MSSF [46]	56.3	99.3	88.6	47.8	67.3	2.3	27.1	20.6	74.8	78.8
MS3 DVS [39]	66.9	99.0	94.8	52.4	38.1	36.0	49.3	52.6	91.3	88.6
HDGCN [25]	68.3	99.4	93.0	67.7	75.7	25.7	44.7	37.1	81.9	89.6
ConvPoint [3]	75.9	99.5	95.1	71.6	88.7	46.7	52.9	53.5	89.4	85.4
KPConv [47]	82.0	99.5	94.0	71.3	83.1	**78.7**	47.7	**78.2**	94.4	91.4
FKAConv (ours *fusion*)	**82.7**	**99.6**	**98.1**	**77.2**	**91.1**	64.7	**66.5**	58.1	**95.6**	**93.9**
Rank	1	1	1	1	1	2	1	2	1	1

Note: We report here only the published methods at the time of writing.

(c) Semantic3D

Method	Av. IoU	OA	Man made	Nat.	High veg.	Low veg.	Build.	Hard scape	Art.	Cars
TML-PC [32]	39.1	74.5	80.4	66.1	42.3	41.2	64.7	12.4	0.	5.8
TMLC-MS [15]	49.4	85.0	91.1	69.5	32.8	21.6	87.6	25.9	11.3	55.3
PointNet++ [35]	63.1	85.7	81.9	78.1	64.3	51.7	75.9	36.4	43.7	72.6
EdgeConv [8]	64.4	89.6	91.1	69.5	65.0	56.0	89.7	30.0	43..8	69.7
SnapNet [4]	67.4	91.0	89.6	79.5	74.8	56.1	90.9	36.5	34.3	77.2
PointGCR [30]	69.5	92.1	93.8	80.0	64.4	66.4	93.2	39.2	34.3	85.3
FPCR [48]	72.0	90.6	86.4	70.3	69.5	68.0	96.9	43.4	52.3	89.5
SPGraph [20]	76.2	92.9	91.5	75.6	**78.3**	71.7	94.4	**56.8**	52.9	88.4
ConvPoint [3]	**76.5**	93.4	92.1	80.6	76.0	**71.9**	**95.6**	47.3	**61.1**	87.7
FKAConv* (ours fusion)	74.6	**94.1**	**94.7**	**85.2**	77.4	70.4	94.0	52.9	29.4	**92.6**
Rank	3	1	1	1	2	3	5	2	9	1

Note: We report here only the published methods at the time of writing.
**In the official benchmark, the entry corresponding to our method is called LightConvPoint, which refers to the framework used for our implementation.*

scanning outliers). It is future work to train the ConvPoint network with our convolution layer to support this hypothesis.

6.2 Support point sampling: discretization parameter.

The rule of thumb in Equation (8) was derived in a simplistic case: a point cloud sampled from an axis aligned plane crossing a regular voxel grid. In practice, planar surfaces are very common, particularly in semantic segmentation (walls, floors, etc.), but are not a good model for most of the object of the scenes (chairs, cars, vegetation, *etc.*). To validate Eq. (8), we compute the optimal quantization parameter (i.e., the parameter with the largest value leading to the desired number of support points in a single quantization) computed using a dichotomic search on the parameter space and compare it to the derived expression. Figure 5 presents the results of the experiment. For each point cloud, the optimal voxel size is represented by a semi-transparent disk (blue for ShapeNet, orange for S3DIS) and can be compared to the derived expression (red curve). In our setting, a curve under the colored disks is not desired; it is an over-quantization. We prefer a curve above these disks, possibly leading to extra iterations, but not affecting performance. We observe that Equation (8) provides a good estimate of the voxel size, especially for S3DIS which is a dataset containing a lot of planes.

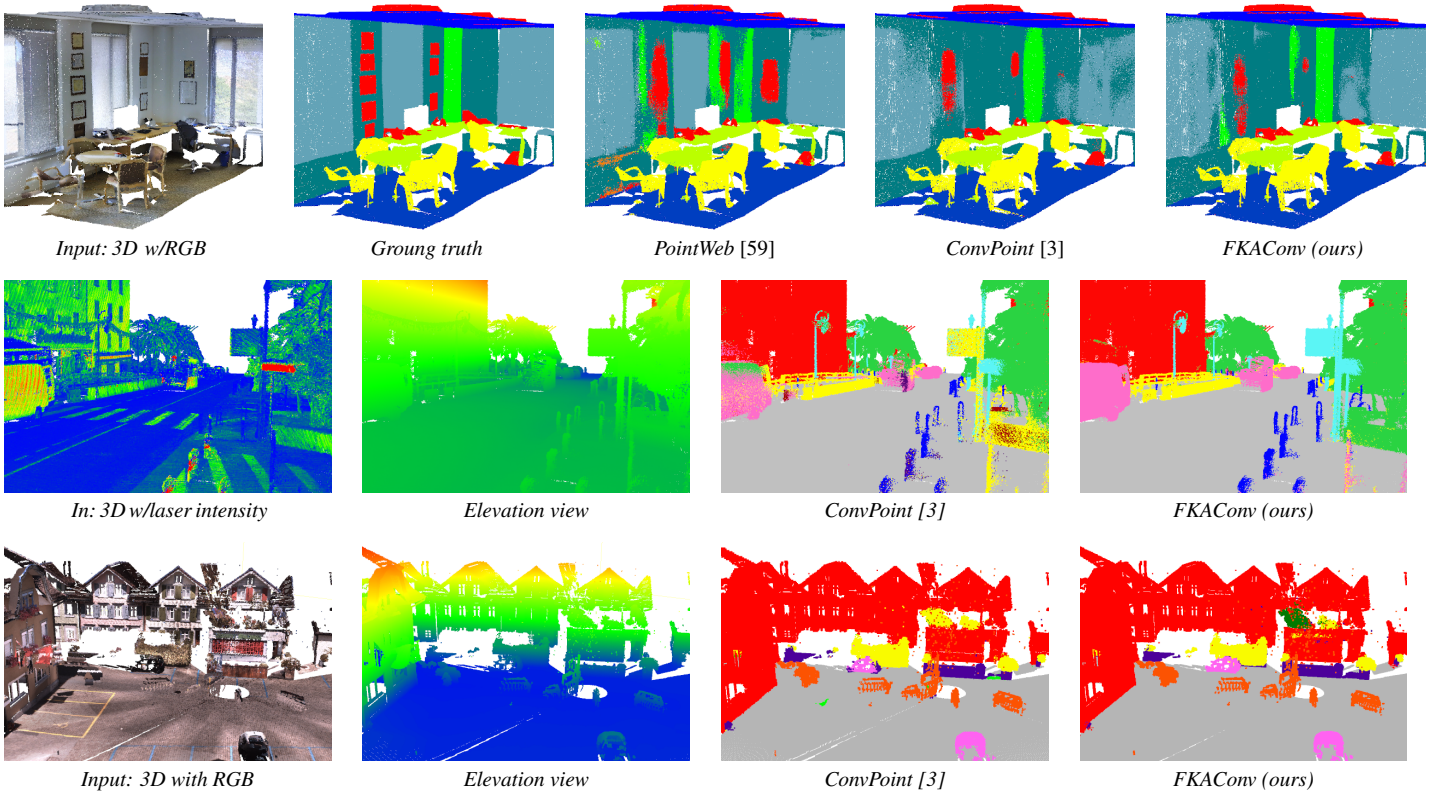


Figure 4: Visual results of semantic segmentation: S3DIS (1st row), NPM3D (2nd row) and Semantic3D (3rd row). Ground truth of test data publicly unavailable for last two.

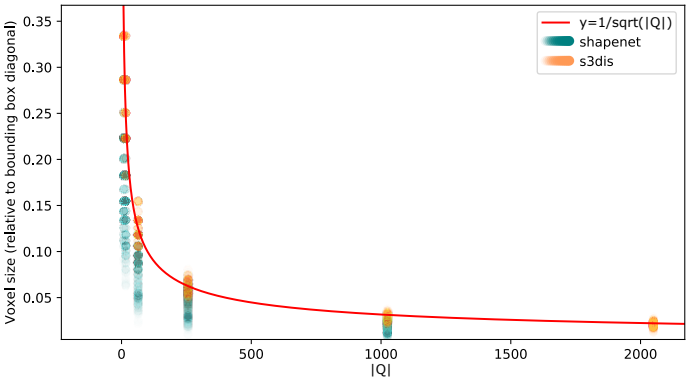


Figure 5: Empirical validation of voxel size estimation: ShapeNet (blue), S3DIS (orange). Each dot is the empirical optimal voxel size obtained by dichotomic search. The red line is the voxel size defined as the inverse square root of the number of support points.

For ShapeNet, we observe a higher variance, due to the great variability of shapes. Because of numerous objects that cannot be modeled well by planes in ShapeNet, we slightly overestimate the voxel size, leading only to one spurious iteration, which only slightly slows down the operation.

6.3 Support point sampling: computation times.

To assess our sampling approach, we run two experiments. First, in Table 3(a), we compare the sampling time as a function of the size of the input point cloud. The number of support points is half the input point cloud size, and the number of neighbors is 16. The scores are averaged over 5000 random points clouds sampled

Table 3: Computation time and memory consumption.

(a) Computation times for different sampling strategies.

Method	Sampling time (ms)			ShapeNet
	1k pts	5k pts	10k pts	(mIoU)
Random	1.66	8.6	18.6	84.4
(baseline)	(-60%)	(-89%)	(-94%)	
ConvPoint [3]	2.60	25.4	88.2	84.6
	(-37%)	(-68%)	(-71%)	
Farthest [35]	4.12	79.8	310.2	84.7
	(-)	(-)	(-)	
FKAConv sampling	1.93	10.3	20.4	84.6
	(-53%)	(-87%)	(-93%)	

(time for n inputs points, $n/2$ support points, 16 neighbors, averaged over 5000 iterations).

(b) Time and memory consumption for a segmentation network, with 8192 points.

Convolution Layer	Training		Test	
	Time (ms)	Memory (GB)	Time (ms)	Memory (GB)
ConvPoint [3]	85.7	10.1	65	2.9
ConvPoint*	**12.2**	**4.3**	4.29	1.6
PointCNN* [23]	33.6	3.5	6.23	1.7
PCCN* [50]	31.1	4.9	10.2	2.3
PCCN** (bs4)	64.2	6.4	19.7	2.6
FKAConv (Ours)	19.1	5.6	**4.9**	**1.4**

*: reimplemented in our framework.
**: original formulation without separation trick, differs from code used for experiments in [50].

in a cube. We also report the ShapeNet scores to relate the performance and the computation times. We compare our sampling strategy with farthest point sampling [35], with iterative neighborhood rejection [3] and with a random baseline. As farthest point sampling [35] is the reference of several state-of-the-art methods, we give the gain relatively to this method in percentage. Our quantized sampling is almost as fast as random sampling and much more efficient than farthest point sampling. In fact, our sampling has almost a linear complexity, compared to farthest point sampling, that has a quadratic complexity.

6.4 Inference time and memory consumption

We present in Table 3(b) the performance of our convolution layer and compare it to other convolutional layers. All computation times and memory usage are given for the segmentation network architecture and for one point cloud. The measures were done with 8192 points in each point cloud and a batch size of 16 (except for PCCN** for which the batch size is reduce to 4 to fit in the 11 GB GPU memory). Computational times are given per point cloud in milliseconds, and memory usage is reported in gigabytes.

We observe that our computation times at inference are very similar to those of ConvPoint [3], which is expected as it falls into our same general formulation. The same would probably be observed for a k-NN version of the KPConv [47]. Then, we remark that PointCNN [23] and PCCN [50] are up to twice slower for inference. PCCN uses the separable kernel trick to improve memory performance (cf. Section 3.0.3). In this form, it is similar to N_{out} (N_{out} being the number of filters of the layer) parallel instances of our layer with one kernel element, i.e., it is equivalent to estimating a different $\mathbf{A}$ for each $f \in \{1, \dots, F\}$. We also report in Table 3(b) the performance for PCCN**, which is the the purely continuous convolution described in PCCN [50], but without the separable kernel trick.

6.5 Filter visualization

Our method FKAConv was derived from the discrete convolution on regular grids. The behavior of our 3D filters should thus be comparable to their 2D counterparts. In Figure 6, we present the outputs of early and deep filters for the classification network on ModelNet40. For easier visualization, the features at coarse scales (high level / deep features) have been upsampled at the full point-cloud resolution. We notice that early layers produce features based on surface orientation. This is consistent with the small receptive field of early layers, that yields fine-scale features. On the contrary, deep layers produces shape-related features detecting objects parts, such as people heads or airplane bodies.

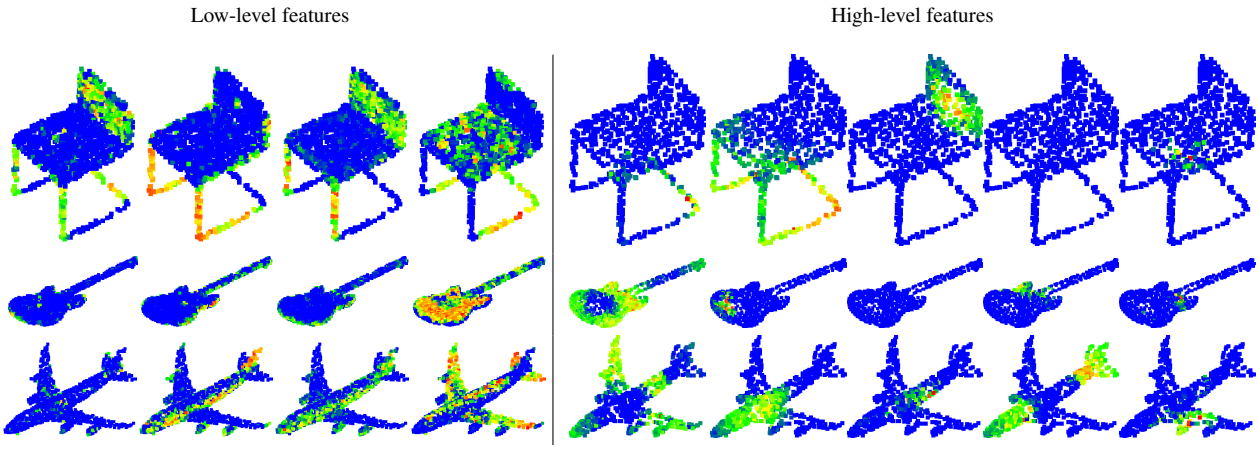


Figure 6: FKAConv filter response for different input shapes on ModelNet40. Low-level features are extracted from the first layer (4 filters), and high-level features from the fourth layer (5 filters). The colormap represents the filter response for the shape, from blue (low response) to red (high response).

7 Conclusion

We presented a formulation for convolution on point clouds that unifies a range of existing convolutional layers and suggests a new point convolution approach. The core of the method is the estimation of an alignment matrix between the input points and the kernel. We also introduced an alternative point sampling strategy to farthest point sampling by using a progressive voxelization of the input space. While being almost as efficient as farthest point sampling, it is nearly as fast as random sampling. With these conceptually simple and easy to implement ideas, we obtained state-of-the-art results on several classification and semantic segmentation benchmarks among methods based on k-NN search, while being among the fastest and most memory-efficient approaches.

References

[1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D semantic parsing of large-scale indoor spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1534–1543, 2016.

[2] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *SIGRAPH, ACM Transaction on Graphics (TOG)*, 37(4), 2018.

[3] A. Boulch. ConvPoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24 – 34, 2020.

[4] A. Boulch, J. Guerry, B. Le Saux, and N. Audebert. SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks. *Computers & Graphics*, 71:189–198, 2018.

[5] 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.

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representation (ICLR)*, 2014.

[7] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1237–1242, 2011.

[8] J. Contreras and J. Denzler. Edge-convolution point net for semantic segmentation of large-scale point clouds. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 5236–5239, 2019.

[9] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3844–3852, 2016.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

[11] B. Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014.

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

[13] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In *European Conference on Computer Vision (ECCV)*, pages 345–360. Springer, 2014.

[14] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys. Semantic3D.net: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS Annals)*, volume IV-1-W1, pages 91–98, 2017.

[15] 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 (ISPRS Annals)*, 3(3), 2016.

[16] B.-S. Hua, M.-K. Tran, and S.-K. Yeung. Pointwise convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 984–993, 2018.

[17] Q. Huang, W. Wang, and U. Neumann. Recurrent slice networks for 3D segmentation of point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2626–2635, 2018.

[18] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 2017.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012.

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

[21] F. J. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. S. Khan, and M. Felsberg. Deep projective 3D semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 95–107, 2017.

[22] J. Li, B. M. Chen, and G. Hee Lee. SO-Net: Self-organizing network for point cloud analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9397–9406, 2018.

[23] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution on X-transformed points. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 820–830, 2018.

[24] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[25] Z. Liang, M. Yang, L. Deng, C. Wang, and B. Wang. Hierarchical depthwise graph convolutional neural network for 3D semantic segmentation of point clouds. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 8152–8158, 2019.

[26] J. Liu, B. Ni, C. Li, J. Yang, and Q. Tian. Dynamic points agglomeration for hierarchical point sets learning. In *IEEE International Conference on Computer Vision (ICCV)*, pages 7546–7555, 2019.

[27] Y. Liu, B. Fan, S. Xiang, and C. Pan. Relation-shape convolutional neural network for point cloud analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8895–8904, 2019.

[28] Z. Liu, H. Tang, Y. Lin, and S. Han. Point-voxel CNN for efficient 3D deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 965–975, 2019.

[29] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[30] Y. Ma, Y. Guo, H. Liu, Y. Lei, and G. Wen. Global context reasoning for semantic segmentation of 3D point clouds. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2931–2940, 2020.

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

[32] 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 (GCPR)*, pages 212–223. Springer, 2014.

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

[34] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5648–5656, 2016.

[35] 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 (NeurIPS)*, pages 5105–5114, 2017.

[36] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3D graph neural networks for RGDB semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5199–5208, 2017.

[37] G. Riegler, A. Osman Ulusoy, and A. Geiger. OctNet: Learning deep 3D representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3577–3586, 2017.

[38] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241. Springer, 2015.

[39] X. Roynard, J.-E. Deschaud, and F. Goulette. Classification of point cloud scenes with multiscale voxel deep network. *arXiv preprint arXiv:1804.03583*, 2018.

[40] 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. *International Journal of Robotics Research (IJRR)*, 37(6):545–557, 2018.

[41] 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.

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

[43] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. SPLATNet: Sparse lattice networks for point cloud processing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2530–2539, 2018.

[44] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[45] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou. Tangent convolutions for dense prediction in 3D. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3887–3896, 2018.

[46] H. Thomas, F. Goulette, J.-E. Deschaud, and B. Marcotegui. Semantic classification of 3D point clouds with multiscale spherical neighborhoods. In *IEEE International Conference on 3D Vision (3DV)*, pages 390–398, 2018.

[47] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. KPConv: Flexible and deformable convolution for point clouds. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[48] G. Truong, S. Z. Gilani, S. M. S. Islam, and D. Suter. Fast point cloud registration using semantic segmentation. In *IEEE Digital Image Computing: Techniques and Applications (DICTA)*, 2019.

[49] C. Wang, B. Samari, and K. Siddiqi. Local spectral graph convolution for point set feature learning. In *European conference on computer vision (ECCV)*, pages 52–66, 2018.

[50] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2589–2597, 2018.

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

[52] 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.

[53] W. Wu, Z. Qi, and L. Fuxin. PointConv: Deep convolutional networks on 3D point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9621–9630, 2019.

[54] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[55] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao. SpiderCNN: Deep learning on point sets with parameterized convolutional filters. In *European Conference on Computer Vision (ECCV)*, pages 87–102, 2018.

[56] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian. Modeling point clouds with self-attention and Gumbel subset sampling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3323–3332, 2019.

[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] Z. Zhang, B.-S. Hua, and S.-K. Yeung. ShellNet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1607–1616, 2019.

[59] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia. PointWeb: Enhancing local neighborhood features for point cloud processing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5565–5573, 2019.

[60] Y. Zhou and O. Tuzel. VoxelNet: End-to-end learning for point cloud based 3D object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4490–4499, 2018.

We present complementary information about the FKAConv paper. In Section A, we detail the network architectures used for classification and semantic segmentation. In Section B, we briefly describe the datasets used for experiments. In Section C, we discuss the use of a learned normalization of the support point neighborhoods. Finally, in Section D, we provide more qualitative results on the semantic segmentation test datasets.

A Network details

As mentioned in Section 5.1 of the paper, the networks we use in our experiments are based on the residual network architecture in KPConv [47], for which we replace the convolution by FKAConv and the neighborhood construction by our point sampling with space quantization. We detail here the main components.

A.0.1 Residual block.

The residual block in Fig. 7 (a) is the main module of our networks. This block is made of an FKAConv layer placed between two linear layers. The residual connection has one *optional* linear layer and one *optional* max-pooling layer. The *optional* linear layer is used only when the number of input channels is different from the number of output channels, and the *optional* max-pooling layer is used if the cardinality of the support points is different from the cardinality of the input points.

A.0.2 Classification and segmentation networks.

The two networks for these tasks are presented in Fig. 7 (b). They share the same encoder structure, i.e., a FKAConv layer and 9 residual modules with a progressive reduction of the point cloud size.

- The classification network has an extra point-wise fully-connected layer (or unary convolutional) with its output dimension equal to the number of classes. The final prediction is done by averaging the scores of the 8 final support points.

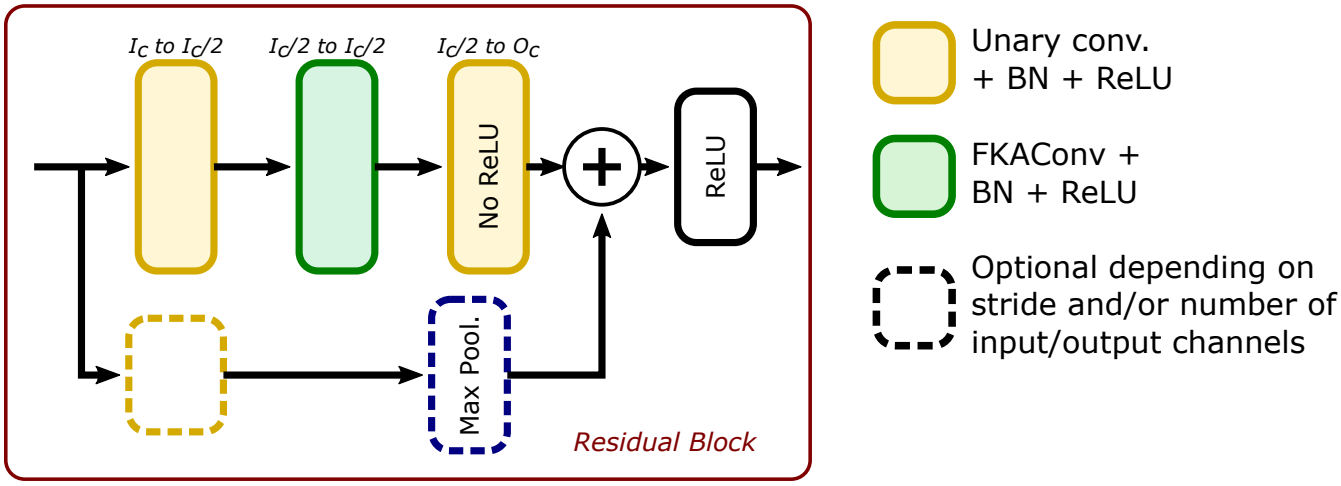
- The segmentation network has an encoder-decoder structure. The decoder is a stack of 5 unary layers with, nearest-neighbor up-sampling between each layer. We use skip connections from the encoder to the decoder: the target points for up-sampling are the support points at the corresponding scale in the encoder, and the features from the encoder and the decoder are concatenated at each scale.

A.0.3 Fusion network.

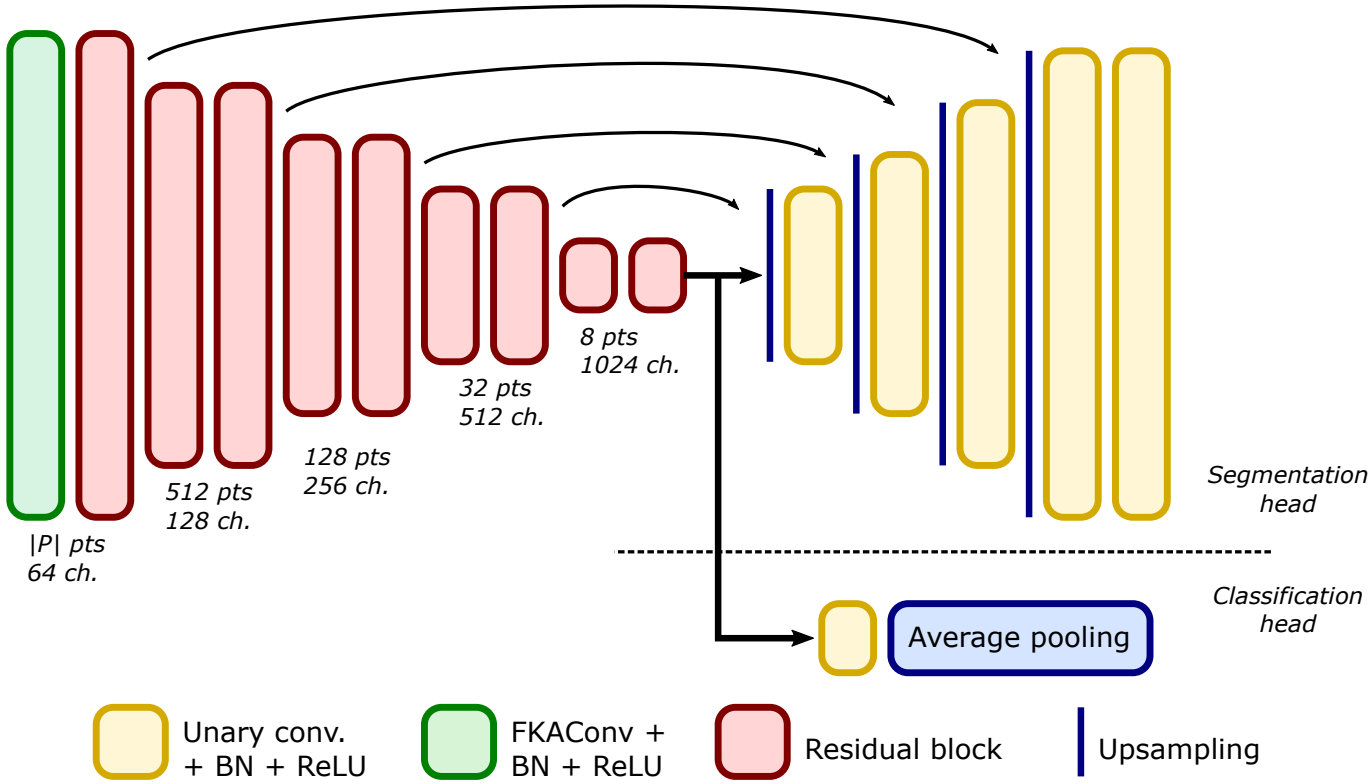
The fusion module presented in Fig. 7 (c) is identical to the module from the official repository of ConvPoint [3], except that it uses our proposed convolution. It is made of 3 layers: 2 FKAConv layers and 1 unary layer. The features from the penultimate layer of both segmentation networks are concatenated and given as input to the first FKAConv layer. The output of the second FKAConv layer is then concatenated with the predictions of the two segmentation networks and given to the unary layer.

A.0.4 Parameters of the convolution.

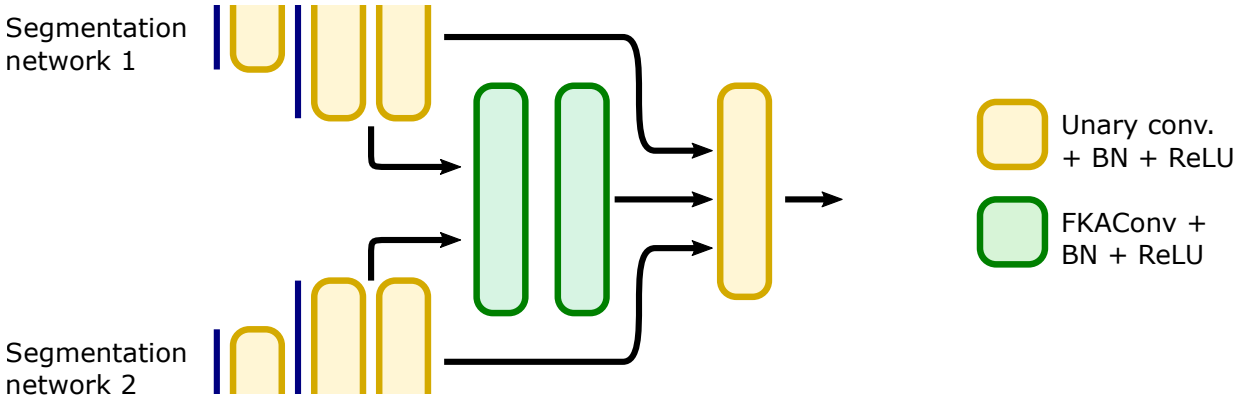
In order to keep the setup simple, we use the same parameters for all FKAConv layers. The neighborhood size of the support points is fixed to 16 and we use 16 kernels. In the encoder, each odd residual block but the first reduces the number of support points from $|P|$ to 512, 128, 32, and finally 8.



(a) Residual block. I_c (resp. O_c) is the number of input (resp. output) channels.



(b) Classification and segmentation networks.



(c) Fusion module.

Figure 7: Network architectures used for semantic segmentation and classification.

B Datasets

We evaluate our convolution on three different tasks: object classification, part segmentation and semantic segmentation.

B.0.1 Classification

is evaluated on ModelNet40 [54]. It contains 12,311 point clouds sampled from CAD models of 40 different categories. 9843 shapes are used for training, 2468 for testing.

B.0.2 Part segmentation

is done on Shapenet [57]. It is composed of 16 object category, each category being annotated with 2 to 6 part labels. As a pre-processing, all models are first normalized to the unit sphere. In our implementation, the network has 50 outputs (one for each part) and the loss and scores are computed per object category.

B.0.3 Semantic segmentation

is evaluated on 3 different indoor and outdoor datasets: S3DIS [1], NPM3D [40] and Semantic8 [14].

- S3DISS [1] is a subset of the 2D-3D-S dataset for semantic segmentation of building interiors. The data are acquired over 6 building floors with an RGBD camera. Each points is annotated with one of 13 labels: 12 semantic labels (floors, tables, chairs, etc.) and 1 label for a "clutter" class, mostly including office supplies. The evaluation is done using a 6-fold cross validation.
- NPM3D [40] is a lidar dataset for large-scale outdoor semantic segmentation. Points were acquired in 4 sites using a car equipped with lidar. 10 classes of urban entities are labeled, such as impervious surface, poll or pedestrian.
- Semantic8 [14] is the main dataset in the Semantic3D benchmark suite. It contains 30 lidar scenes, 15 for training and 15 for testing. Over 4 billion points are labeled with 8 classes such as building, vegetation and car, and a challenging class for scanning artifacts. The test set is particularly difficult as it covers several diverse scenes such as city streets, villages or old castles.

C K-nearest neighbors search with learned neighborhood normalization

As described in Section 4 of the paper, we normalize the size of point neighborhoods. After recalling the principle of this normalization, we analyze here its effect empirically, for different datasets and for different layers in our networks.

To normalize the neighborhoods, we estimate an average neighborhood radius r_t using an exponential average of the actual neighborhood radius $\hat{r}_t$ seen at training time in the successive batches indexed by t (see Equation (9) below where m is the momentum parameter). The point coordinates $(\mathbf{p}_i)_{1\leq i\leq k}$ of the k nearest neighbors of a support point $\mathbf{q}$ are then centered and normalized as specified in Equation (10), yielding normalized points $(\hat{\mathbf{p}}_i)_{1\leq i\leq k}$:

$$r_t = \hat{r}_t * m + r_{t-1} * (1-m), \tag{9}$$

$$\hat{\mathbf{p}}_i = (\mathbf{p}_i - \mathbf{q})/r_t. \tag{10}$$

We also proposed a gating mechanism to reduce, if needed, the negative effect of faraway points. Given the centered and normalized points $(\hat{\mathbf{p}}_i)_i$ computed above, the spatial gate weight $\mathbf{s} = (s_i)_i$ satisfies:

$$s_i = \sigma(\beta - \alpha||\hat{\mathbf{p}}_i||_2), \tag{11}$$

where $\sigma(\cdot)$ is the sigmoid function, and α, β are parameters to learn.

C.0.1 Average neighborhood radius r_t.

First, we study the estimated neighborhood size r_t at each layer of the encoder after training. This size is the averaged radius of the smallest sphere centered on the support point and encompassing the 16 nearest neighbors. We present in Fig. 8 (a) the evolution of this radius as a function of the layer's depth, for different datasets.

We observe that this radius is directly linked to the size of the bounding box of the input point cloud in the 3D space, i.e., to the size of the point cloud pillars (vertical infinite cylinders of diameter 8 meters for Semantic8 and NPM3D, and 2 meters for S3DIS) or to the size of the ShapeNet's CAD models.

C.0.2 Weighting function.

In Figs. 8 (b-c), we plot the values of parameters α and β as a function of the depth of the FKAConv layers. First, we observe that these parameters, which are optimized with the network, take values that are different from the initial values that are set at the training initialization, i.e., $\alpha = 1$ and $\beta = 1$. Second, the values of α and β are similar after training on Semantic8 and NPM3D. This could be expected as both datasets share common characteristics: outdoor urban scenes, same pillar size. Third, we observe the same global variations of the curves on all datasets: α tends to decrease with the depth while β tends to increase. As a result, the transition of the gating function becomes wider in deeper layers, i.e., deeper layers tend to take into account more far away points.

We illustrate this phenomenon on Figs. 8 (d-e), where we represent the weighting function after optimization for the first and for the seventh FKAConv layer of the network. For comparison purpose, we normalize the curves by setting the weight to 1 at distance 0. The black curve is the initial function, before optimization.

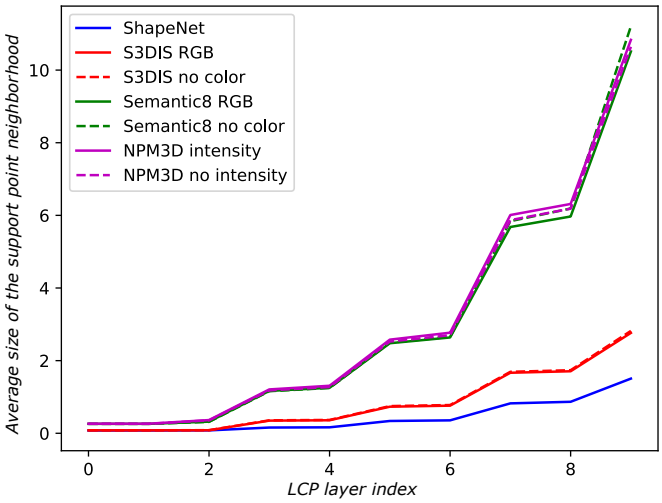
At the first layer, only the neighboring points that are very close to the support point are taken into consideration to estimate matrix $\mathbf{A}$. We hypothesize that, in the absence of noise in data, a small neighborhood is sufficient to estimate local geometric features.

On the contrary, in the 7th layer, all the neighboring points are taken into consideration to compute $\mathbf{A}$. At this stage, the number of support points is small and each point carries features that are discriminating for the task. The network considers all available information, including points that are faraway from the support points.

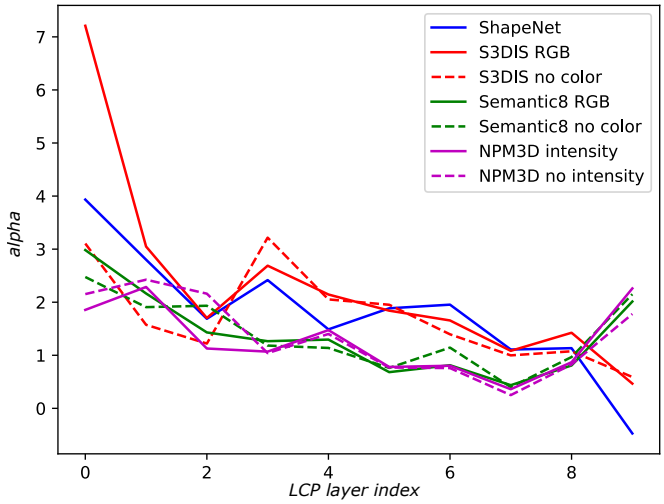
C.0.3 Influence on performance.

To quantify the impact of the neighborhood normalization and gating mechanism, we trained a segmentation network using five different configurations.

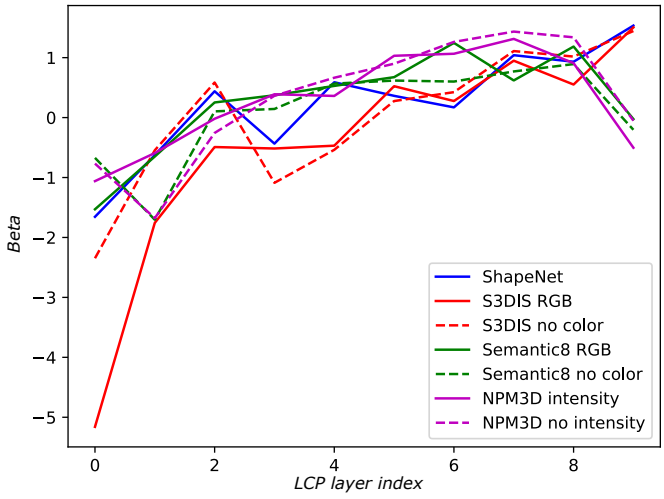
- *Baseline (no normalization).* We do not normalize the neighborhood coordinates. The network sees different neighborhood sizes.
- *Baseline (normalization to the unit ball).* Each neighborhood is normalized into the unit sphere, regardless of its original size.
- *Learned normalization + fixed gating at r.* The radius used for normalization is estimated using the proposed exponential moving average. The gating mechanism is replaced by hard-thresholding: features of all points outside the ball with the learned radius r are set to zero.



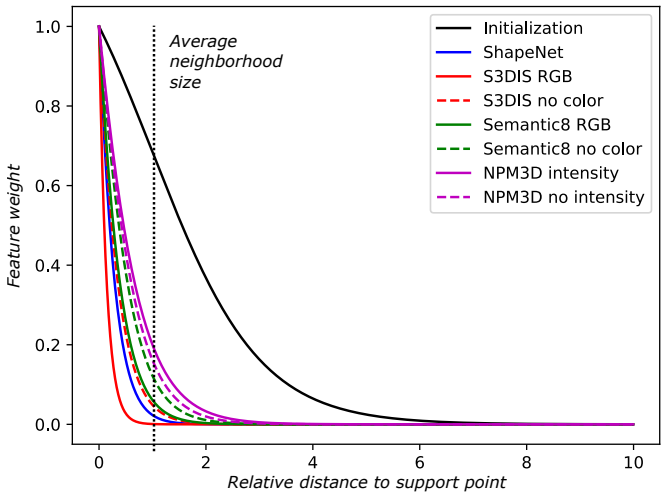
(a) Average neighborhood radius at each layer.



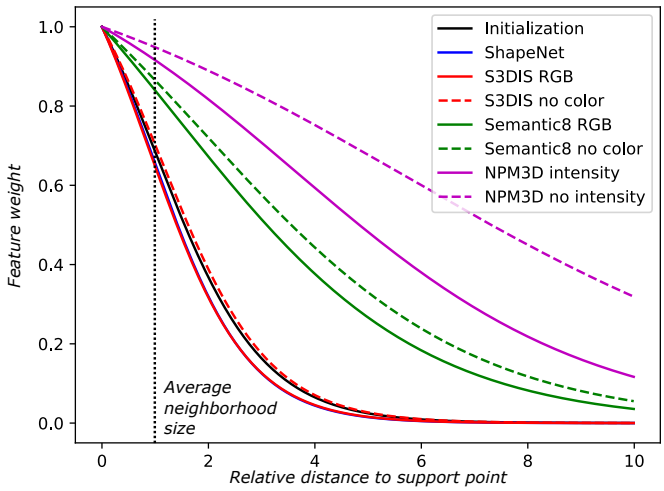
(b) α learned at each layer.



(c) β learned at each layer.



(d) for the first FKAConv layer.



(e) for the 7th residual block.

Influence of the distance to support point on feature weight

Figure 8: Behavior of the learned neighborhood normalization and feature weighting across the segmentation network for various datasets.

- *Learned normalization + fixed gating at* $2r$. Same as above but using a radius of $2r$ for hard-thresholding.
- *Our approach*. The radius used for normalization and the gating function are learned as proposed.

Table 4: Impact of the neighborhood normalization and gating on ShapeNet.

Method	mcIoU	mIoU
Baseline (no normalization)	84.3	85.3
Baseline (normalization to unit ball)	84.6	85.6
Learned normalization + fixed gating		
$s_i = 1$ if $d(\mathbf{p}, \mathbf{q}) < r_t$, and $s_i = 0$ otherwise.	83.8	84.8
$s_i = 1$ if $d(\mathbf{p}, \mathbf{q}) < 2r_t$, and $s_i = 0$ otherwise.	84.0	85.2
($\mathbf{p}$ *denotes a point in the neighborhood of* $\mathbf{q}$*, and* r_t *is the estimated radius in Equation (1).)*		
Our method (learned normalization and gating)	**84.8**	**85.7**

The results, reported in Table 4, show a slight improvement using our approach with respect to the baselines: the mean class intersection over union (mcIoU) increases by 0.2 point and the instance average intersection over union (mIoU) by 0.1 point. This gain may seem small, but it is significant on this dataset as the performance are close to saturation in the leaderboard.

We also observe that the learned gating is an important factor of the success of our approach. Fixed gating leads to a performance drop (see Table 4). This is due to the fact that hard-thresholding suppresses too much information, particularly in the late stages of the network where the neighborhoods are less regular and more subject to size variation.

D Qualitative results of segmentation

Finally, we provide more visual results illustrating our semantic segmentation predictions on datasets NPM3D (Fig. 9), Semantic8 (Fig. 10) and S3DIS (Fig. 11).

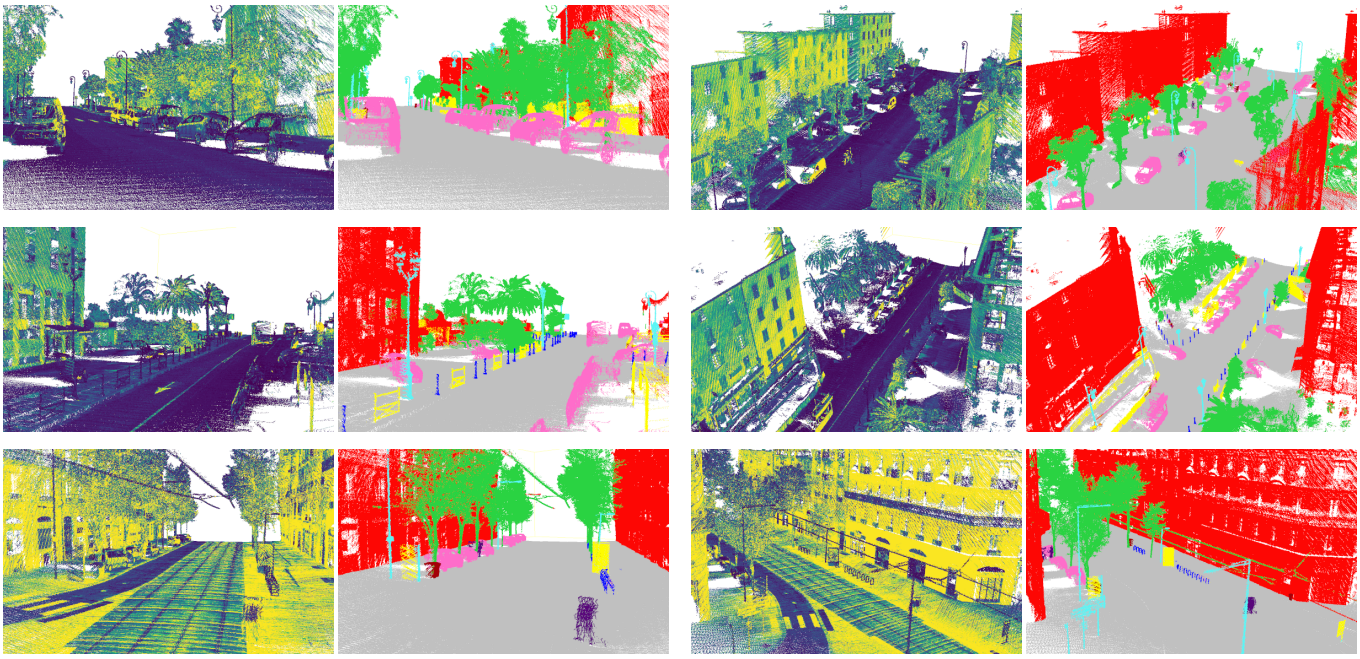


Figure 9: Visual results of our predictions on the test scenes of the NPM3D dataset. The input data is colored according to lidar intensity, from blue (low intensity) to red (high intensity) through green, yellow and orange. Colored, predicted, semantic segments are immediately on the right to lidar images. The segmentation results are obtained with the fusion model (intensity + geometry).

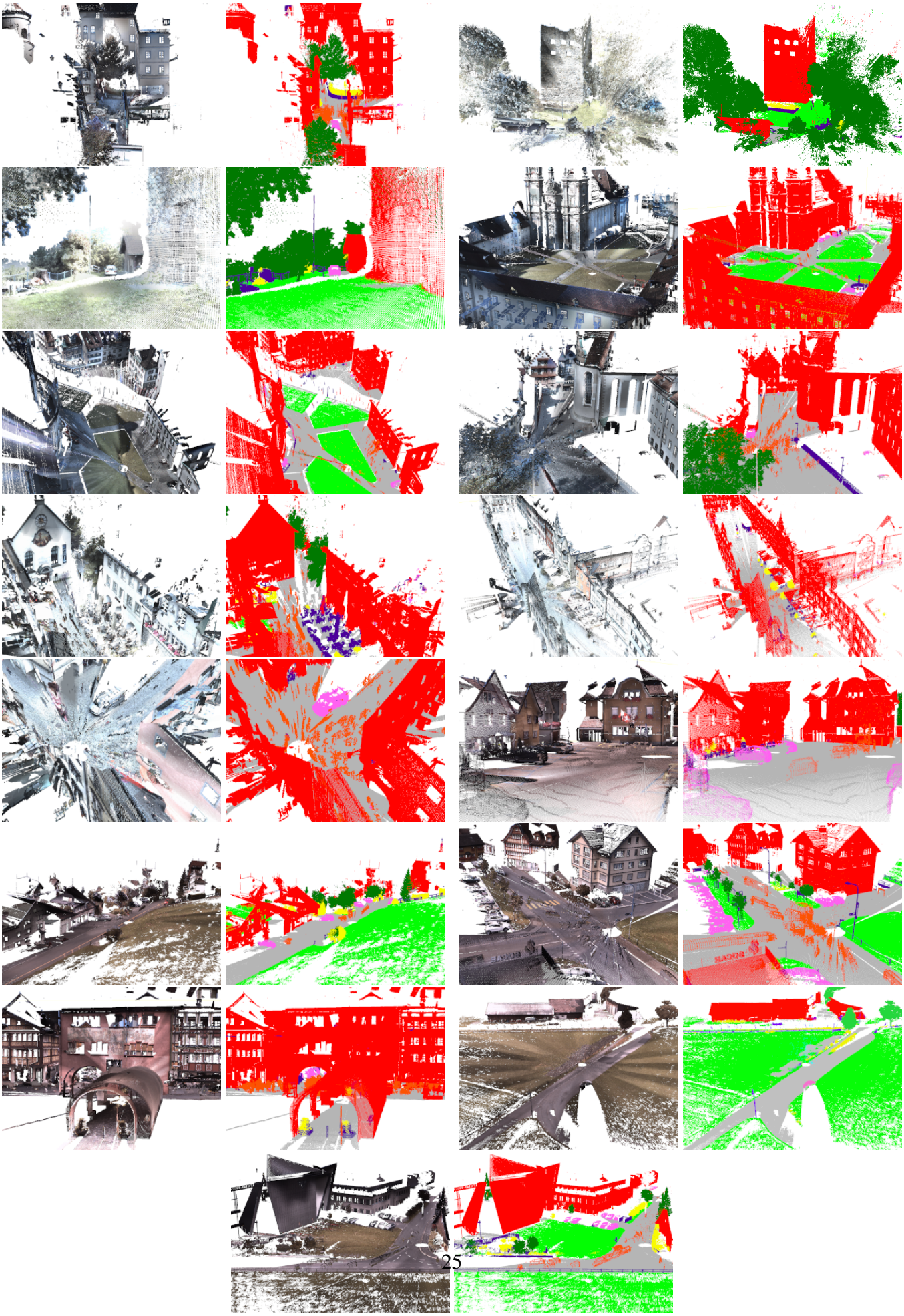


Figure 10: Visual results of our predictions on the 15 test scenes of the Semantic8 dataset. The segmentation

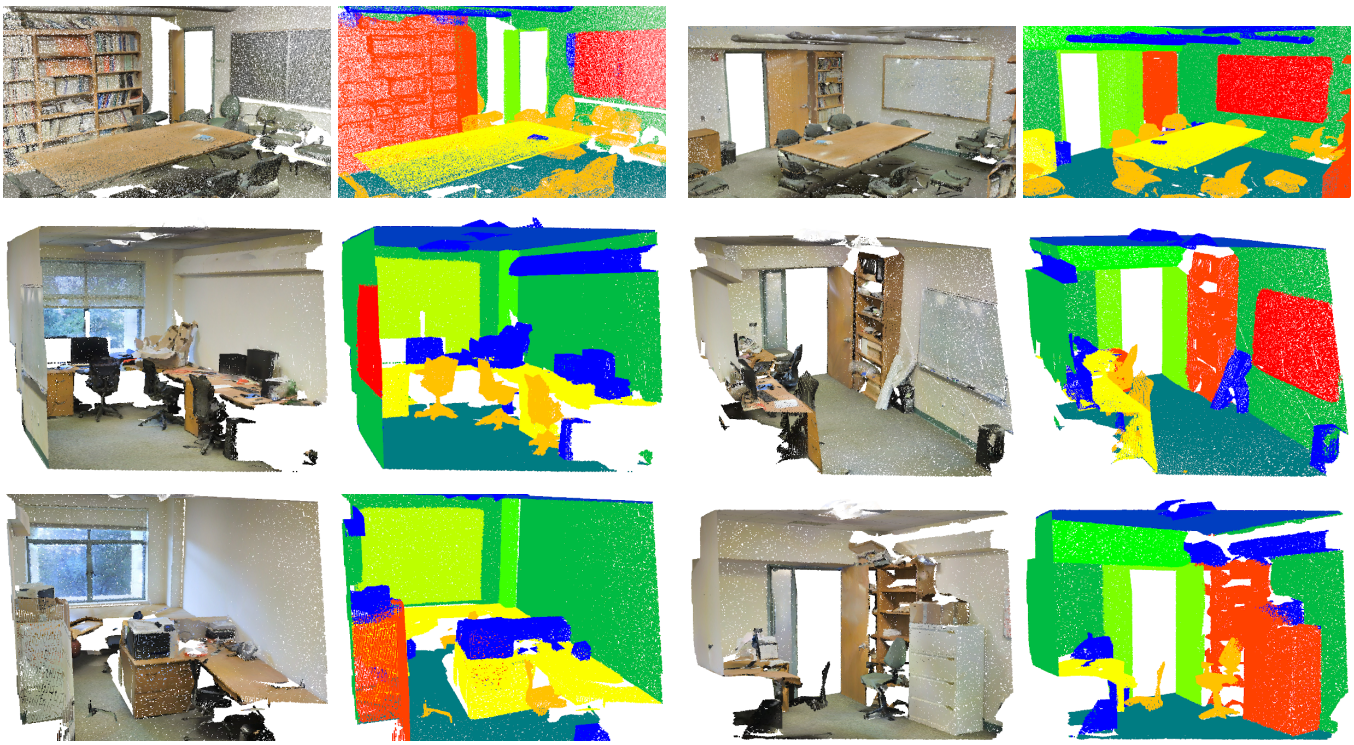


Figure 11: Visual results of our predictions on the S3DIS dataset obtained with the fusion model (RGB + geometry).