FKAConv: Kernel-Feature Alignment for Point Cloud Convolution

Alexandre Boulch, Gilles Puy and Renaud Marlet

ACCV 2020

Introduction

Image processing

CNNs for image processing

Flexible (usage for multiple tasks)
Efficient
Fast
Easy to setup (e.g., Pytorch)

Introduction

Image processing

CNNs
for image processing

Point cloud processing

~~CNNs for image processing~~

Point clouds:
Invariance by permutation
Density variations
From 1K to 100M points

Introduction

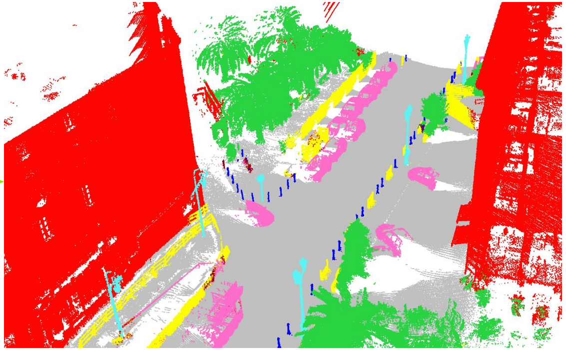
Image processing

CNNs for image processing

Similar formulation taking into account point cloud specificities

Point cloud processing

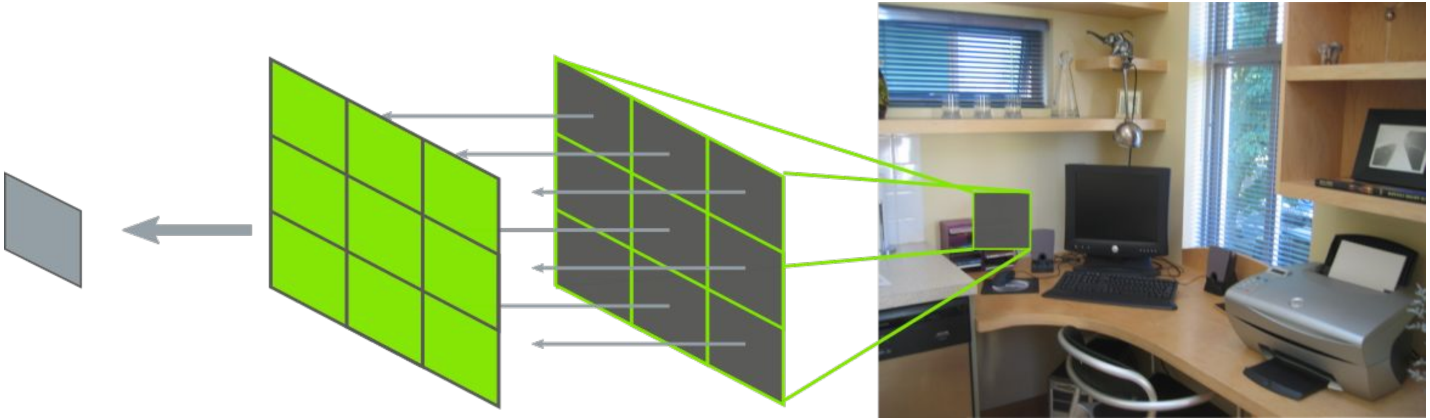
CNNs for point cloud processing



From discrete convolution to convolution for point clouds

Convolution for image processing

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



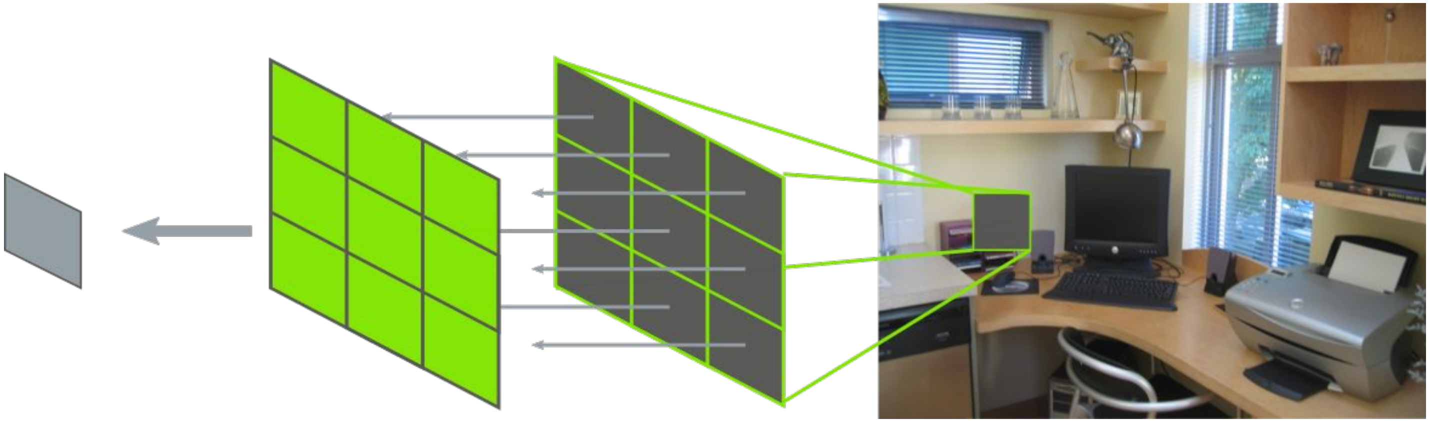
Kernel Features

From discrete convolution to convolution for point clouds

Convolution for image processing

Identity matrix: one to one alignment matrix

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



Kernel

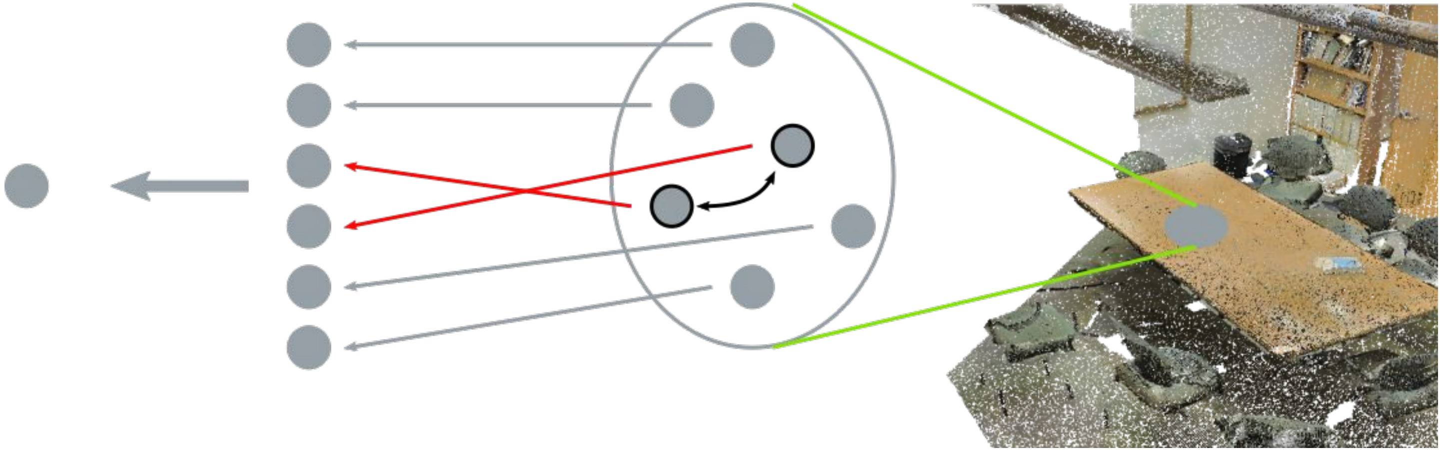
Features

valeo.ai

From discrete convolution to convolution for point clouds

Toward convolution for point clouds

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



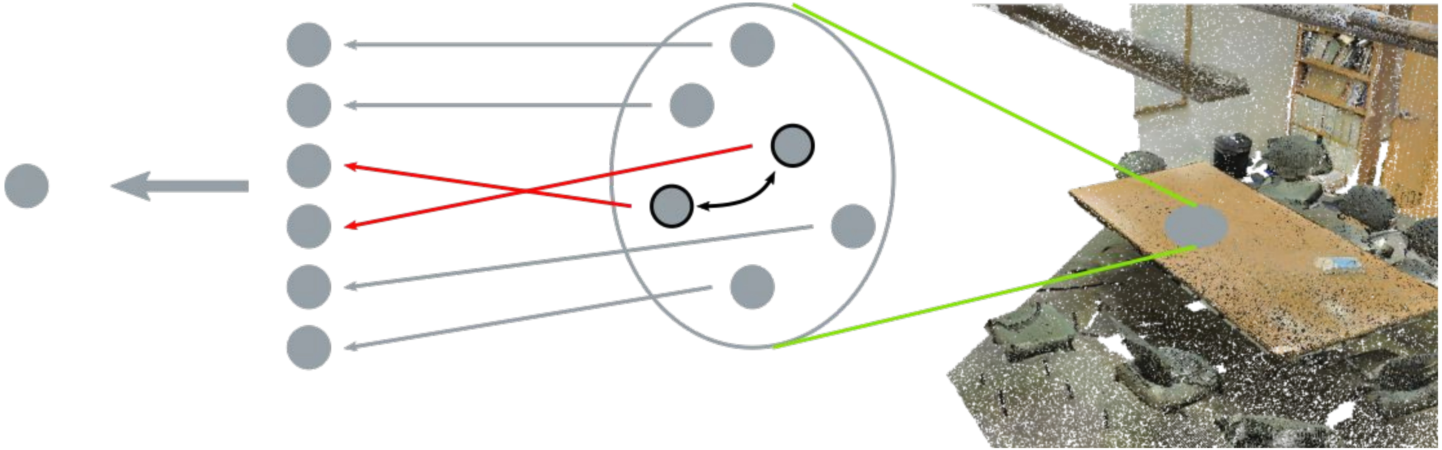
Permutation of two points in the input ⇒ different result

From discrete convolution to convolution for point clouds

Toward convolution for point clouds

Identity matrix:
⇒ no invariance to permutation

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



Permutation of two points in the input ⇒ different result

From discrete convolution to convolution for point clouds

Toward convolution for point clouds

$$\mathbf{h}[n] = \sum_{f\in\{1,\ldots,F\}} \underbrace{\mathbf{K}_f^\top}_{\text{Kernel space}} \cancel{\boxed{I}} \underbrace{\mathbf{f}_f(n)}_{\text{Feature space}}$$

Need for a more general matrix

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

Alignment matrix prediction

Toward convolution for point clouds

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

Alignment matrix $\mathbf{A}$
Invariance to permutation
⇒ estimated based in inputs

Alignment matrix prediction

Toward convolution for point clouds

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

Alignment matrix $\mathbf{A}$
Invariance to permutation
⇒ estimated based in inputs

SplatNet [43]
- Kernel on grid
- Nearest neighbor interpolation

KPConv [47]
- Kernel elements on geodesic ball
- $\mathbf{A}$ computed according to distances from input to kernel

ConvPoint [3]
- Kernel elements randomly initialized and optimized
- $\mathbf{A}$ computed with a learnable function

References as in the paper.

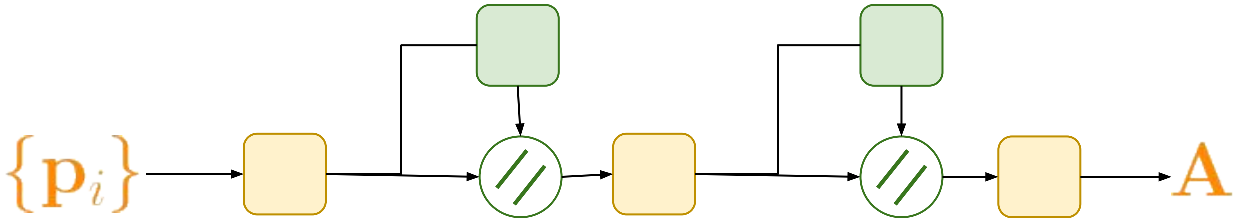
From discrete convolution to convolution for point clouds

Implicit formulation of the kernel location

Estimation of A using a point-wise MLP with context aggregation

⇒ **invariance to point permutation**

$$\mathbf{A} = \phi(\mathbf{p}_i, \{\mathbf{p}_i\})$$



Point-wise linear

Max-Pooling

Concatenation

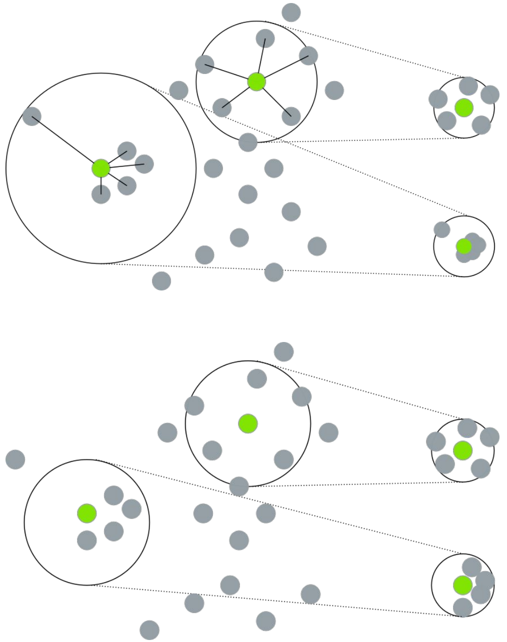
Adaptive normalization of support point neighborhoods

Convolutions operates on local neighborhoods around support point.

Two common strategies for neighborhood computation:

- **K-nearest neighbors**
  - Fast
  - Loss of scale information
  - Influence of outliers
- **Radius search**
  - Slower for large scenes
  - Different sizes of neighborhoods

    ⇒ memory consuming strategy

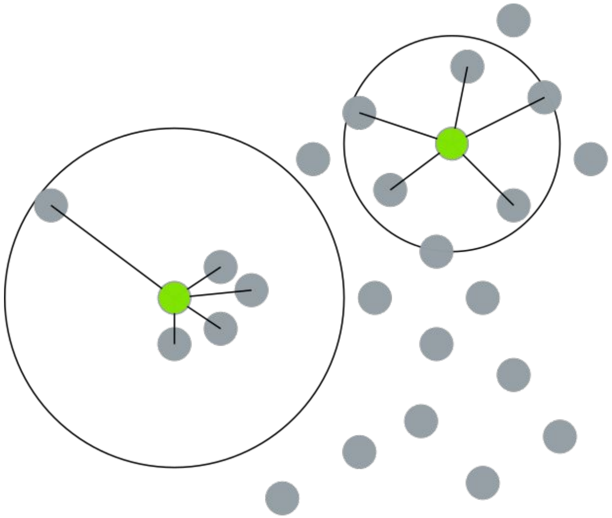


Adaptive normalization of support point neighborhoods

Convolutions operates on local neighborhoods around support points

Adaptive normalization of support point neighborhoods

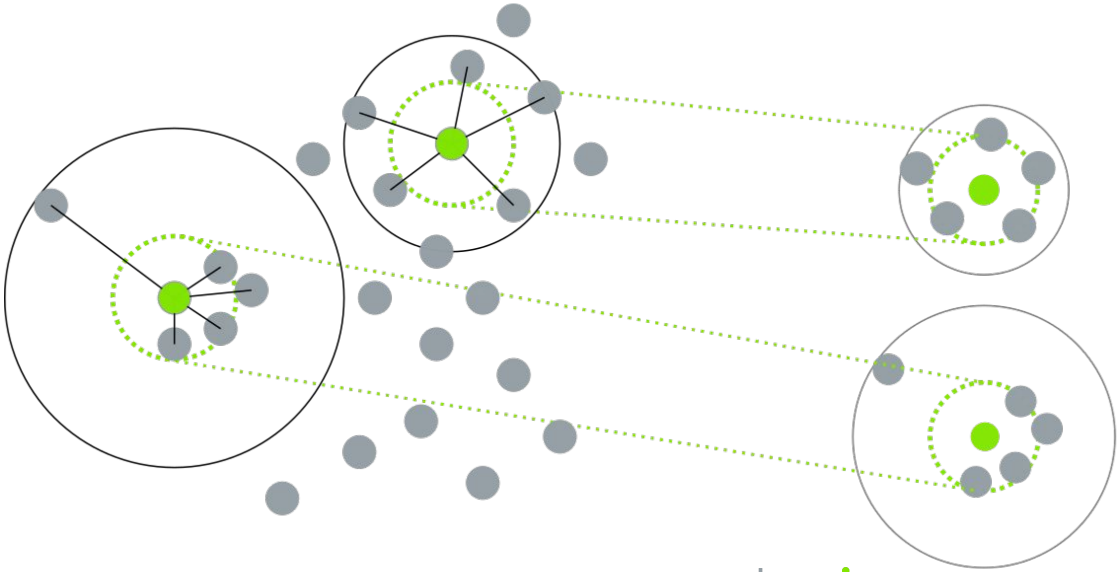
- Use **K-nearest neighbor** search (large scenes: usually faster than radius search)



valeo.ai

Adaptive normalization of support point neighborhoods

- Use **K-nearest neighbors** search (large scenes: usually faster than radius search)
- Normalize using average neighborhood radius ⇒ **scale information preserved**



Adaptive normalization of support point neighborhoods

- Use **K-nearest neighbors** search (large scenes: usually faster than radius search)
- Normalize using average neighborhood radius ⇒ **scale information preserved**
- Learn to **weight influence of outliers** according to distance to support point

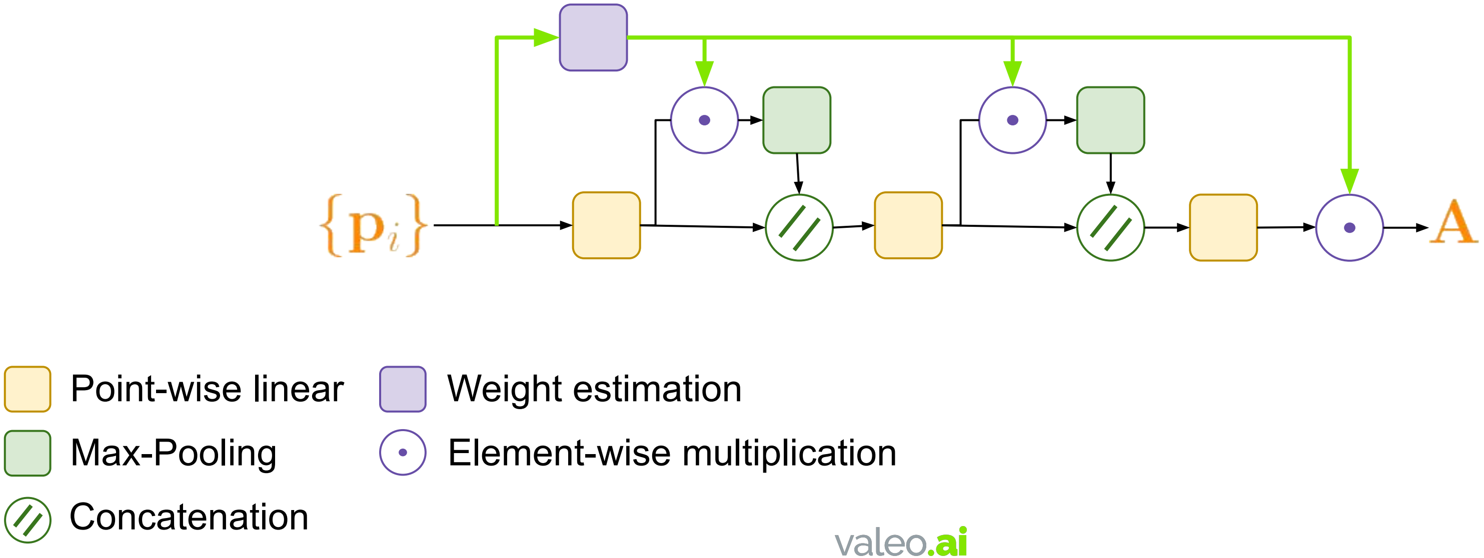
Adaptive normalization of support point neighborhoods

Implicit formulation of the kernel location

Estimation of A using a point-wise MLP with context aggregation.

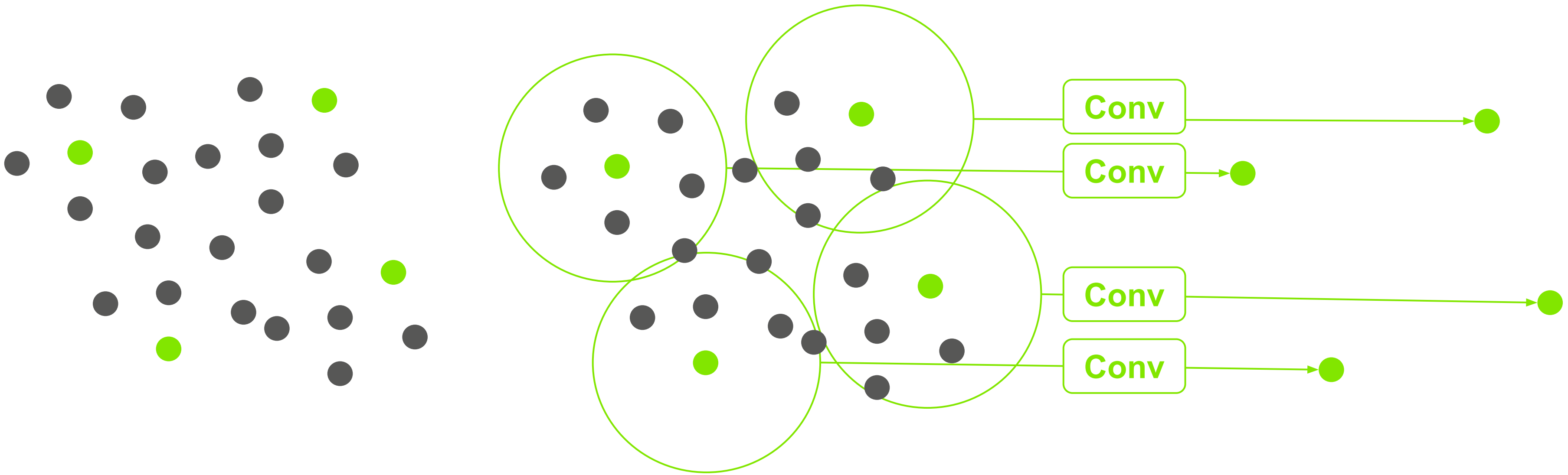
⇒ **invariance to point permutation**

⇒ **reduced influence of outliers**



Quantized sampling

Reduction of point cloud size through the network (grid data: convolution with stride)



Select a given number of support points

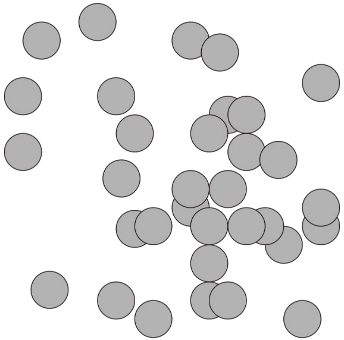
Compute neighborhoods of support points

Apply convolutional layer

Common approach: **Furthest Point Sampling [35]** → slow (requires to maintain distance maps)

Quantized sampling

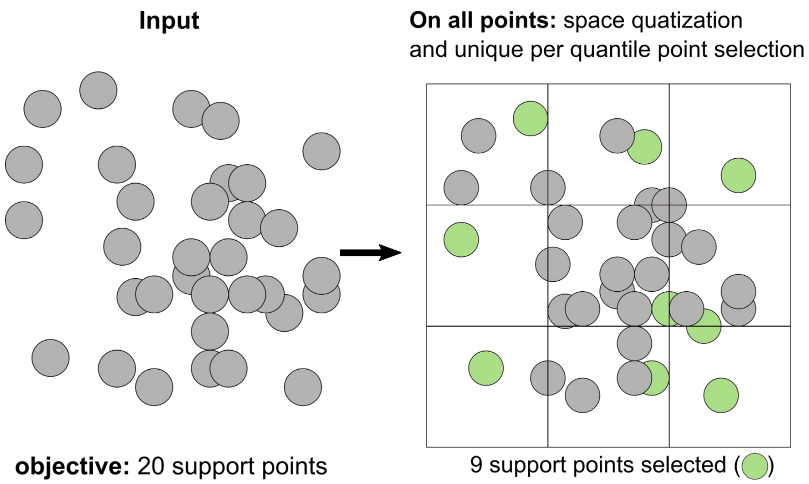
Input



objective: 20 support points

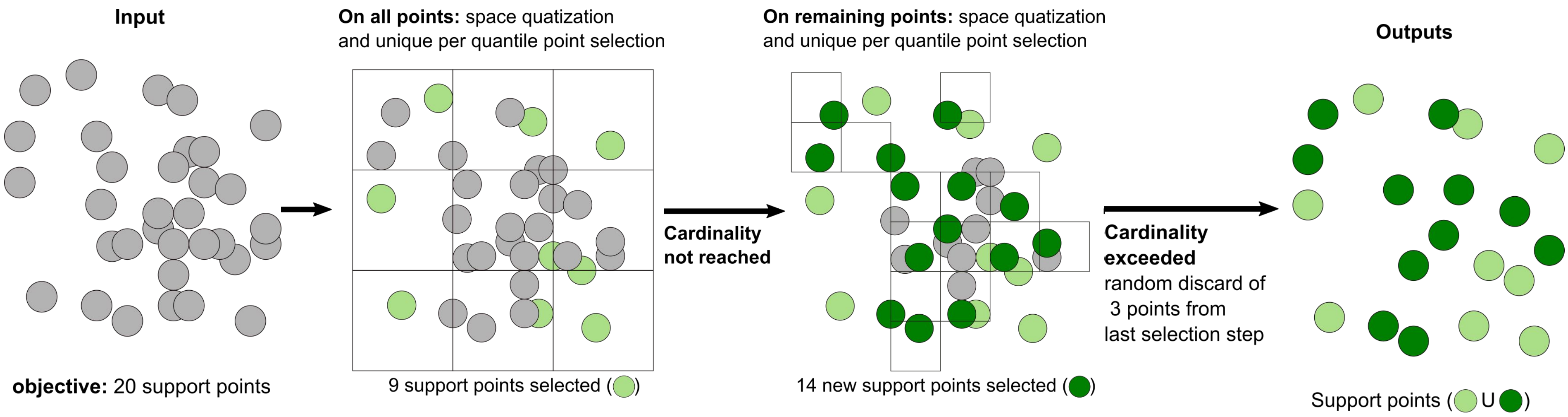
Quantized sampling

1. Quantization of the space
2. Select one point in each voxel



Quantized sampling

1. Quantization of the space
2. Select one point in each voxel
3. Reduce voxel size and iterate until the number of support points is reach



valeo.ai

Quantized sampling

Fast sampling ⇒ A good initial voxel size

Objective: get almost all support points at first iteration without over-voxelization

Voxel size is estimated at point-cloud level

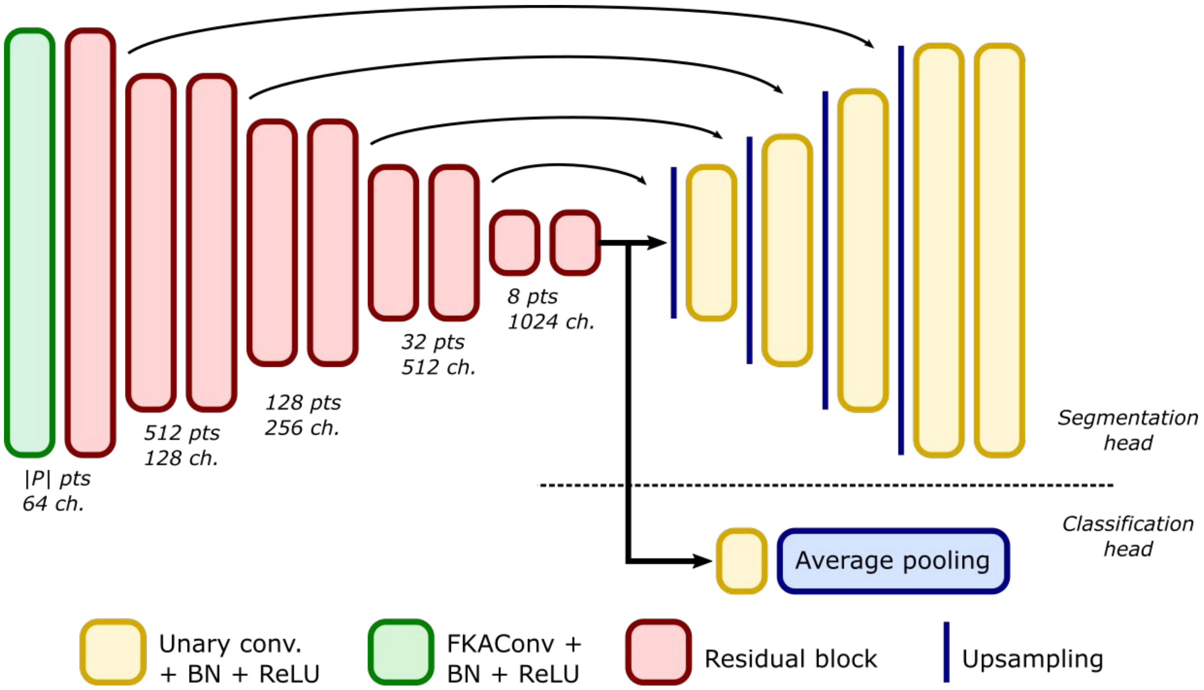
$$v = \text{diag}/\sqrt{|Q|}$$

Number of support points

Diagonal of bounding box

Model based on a simple case (planar surface) and validated on experimental data

Network

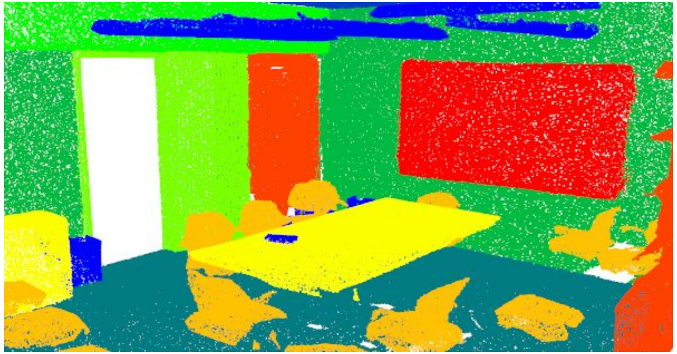


|P| pts
64 ch.
512 pts
128 ch.
128 pts
256 ch.
32 pts
512 ch.
8 pts
1024 ch.
Segmentation
head
Classification
head
Average pooling
Unary conv.
+ BN + ReLU
FKAConv +
BN + ReLU
Residual block
Upsampling

Experimental results: S3DIS

- **2nd on S3DIS**
  - 1st K-nn-based method
  - 1st on 3/15 categories

Method	Search	IoU
Pointnet [31]	Knn	47.6
RSNet [17]	-	56.5
PCCN [48]	-	58.3
SPGraph [20]	Super pt.	62.1
PointCNN [23]	Knn	65.4
PointWeb [56]	Knn	66.7
ShellNet [55]	Knn	66.8
ConvPoint [3]	Knn	68.2
KPConv [45]	Radius	**70.6**
FKAConv (Ours *fusion*)	Knn	68.4
Rank		2

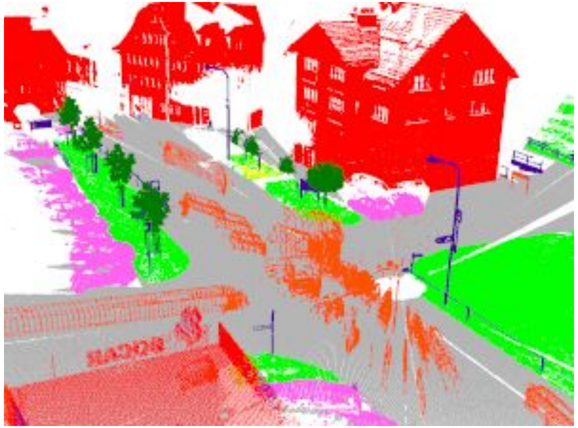


References as in the paper.

Experimental results: Semantic8

- **2nd on Semantic8**
  - 1st on 3/8 classes

Method	Av. IoU	OA
TML-PC [30]	39.1	74.5
TMLC-MS [15]	49.4	85.0
PointNet++ [33]	63.1	85.7
EdgeConv [8]	64.4	89.6
SnapNet [4]	67.4	91.0
PointGCR [28]	69.5	92.1
FPCR [46]	72.0	90.6
SPGraph [20]	76.2	92.9
ConvPoint [3]	**76.5**	93.4
FKAConv* (ours fusion)	74.6	**94.1**
Rank	3	1

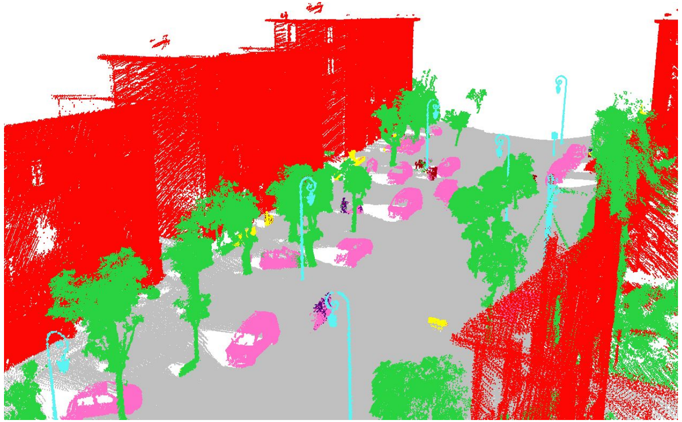


References as in the paper.

valeo.ai

Experimental results: NPM3D

- **1st on NPM3D**
  - 1st on 7/9 classes



Method	Av.IoU
RF MSSF [44]	56.3
MS3 DVS [37]	66.9
HDGCN [25]	68.3
ConvPoint [3]	75.9
KPConv [45]	82.0
FKAConv (ours *fusion*)	**82.7**
Rank	1

References as in the paper.

Conclusion

FKAConv: Feature-Kernel Alignment for Point Cloud Convolution

- A simple **formulation** of convolution for point cloud using an **alignment matrix**
- An **adaptive normalization** using an *average radius* and a learned *outlier filter*
- A **quantized sampling**: a fast an efficient point-cloud sampling

Code available at

https://github.com/valeoai/FKAConv

using LightConvPoint, a library for convolution on points (PyTorch):

https://github.com/valeoai/LightConvPoint