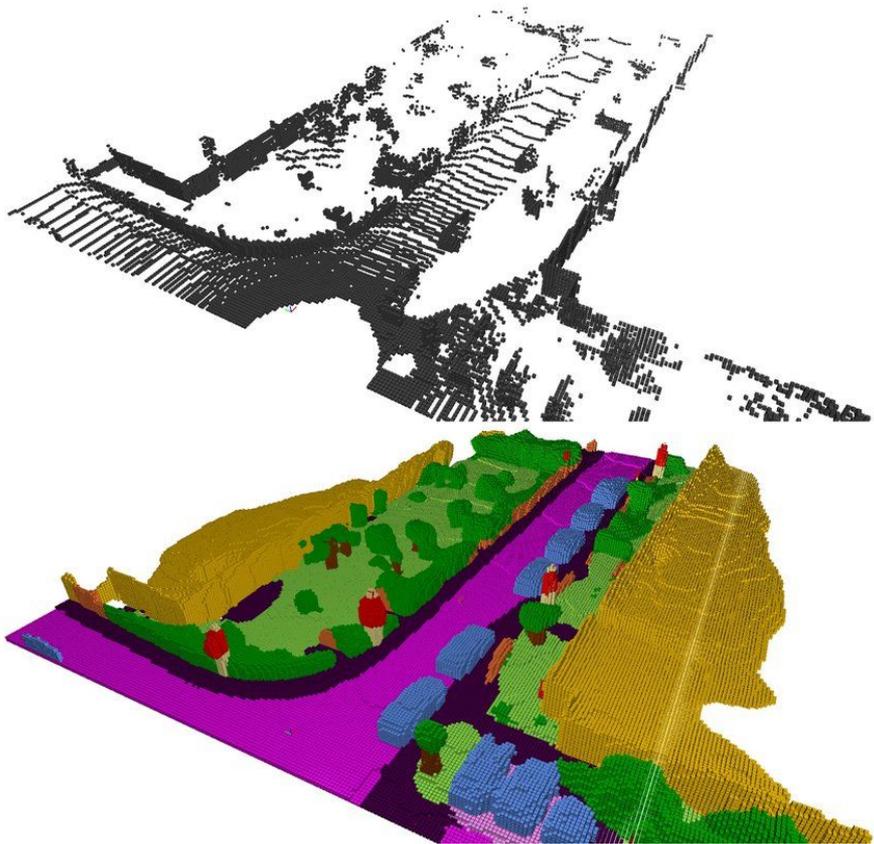


# Nuages de Points et Modélisation 3D

6 - Machine learning III  
From convolution to transformers

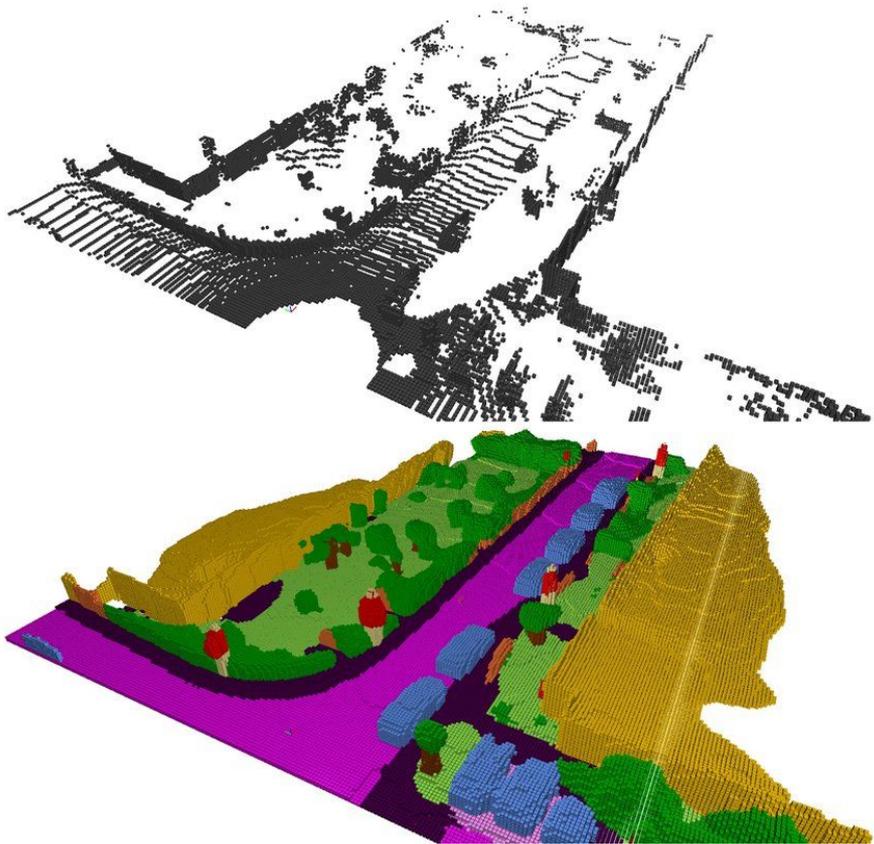
# Overview

- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers



# Overview

- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers

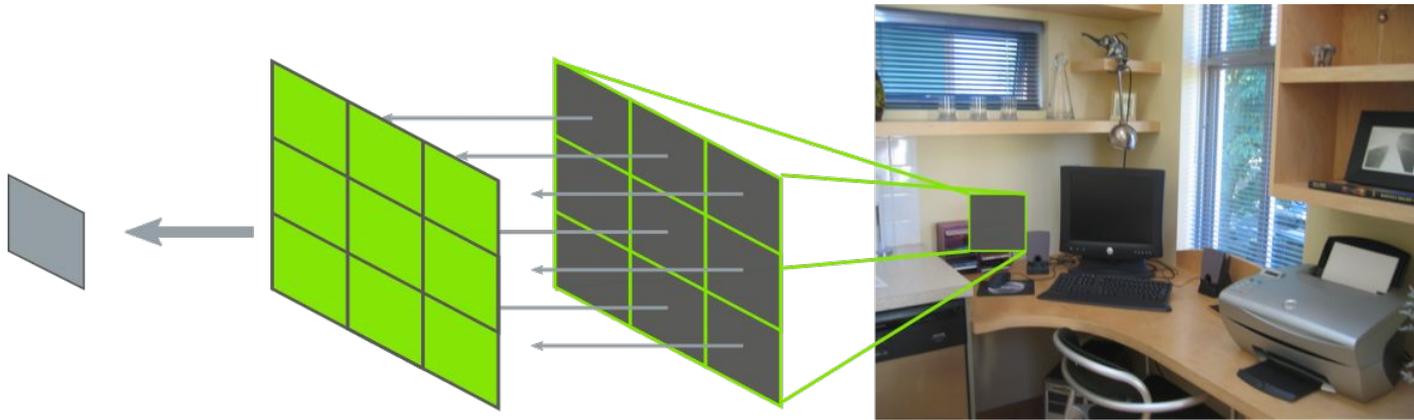


# Convolution on images

Convolution on images

Convolution for image processing

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

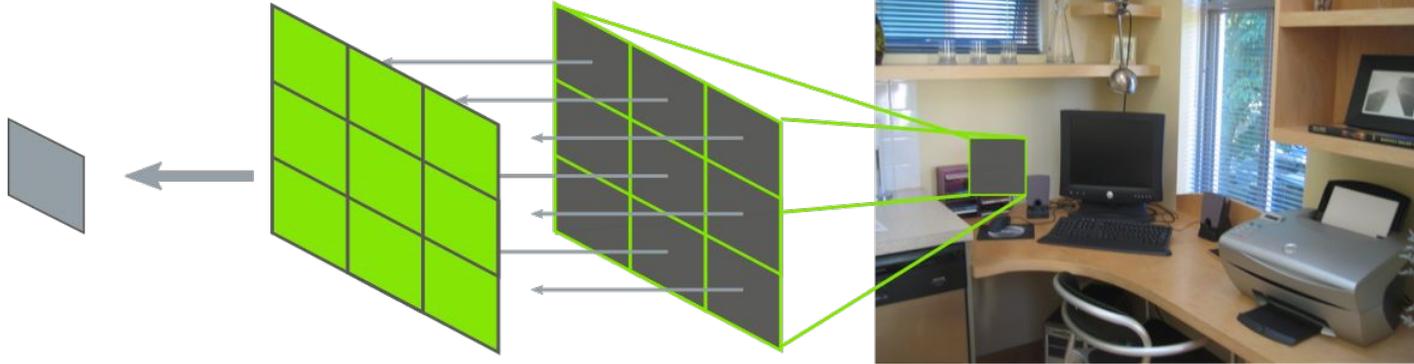


# Convolution on images

Convolution on images

Convolution for image processing

$$\mathbf{h}[n] = \sum_{f \in \{1, \dots, C\}} \mathbf{K}_f^T \mathbf{f}_f(n)$$

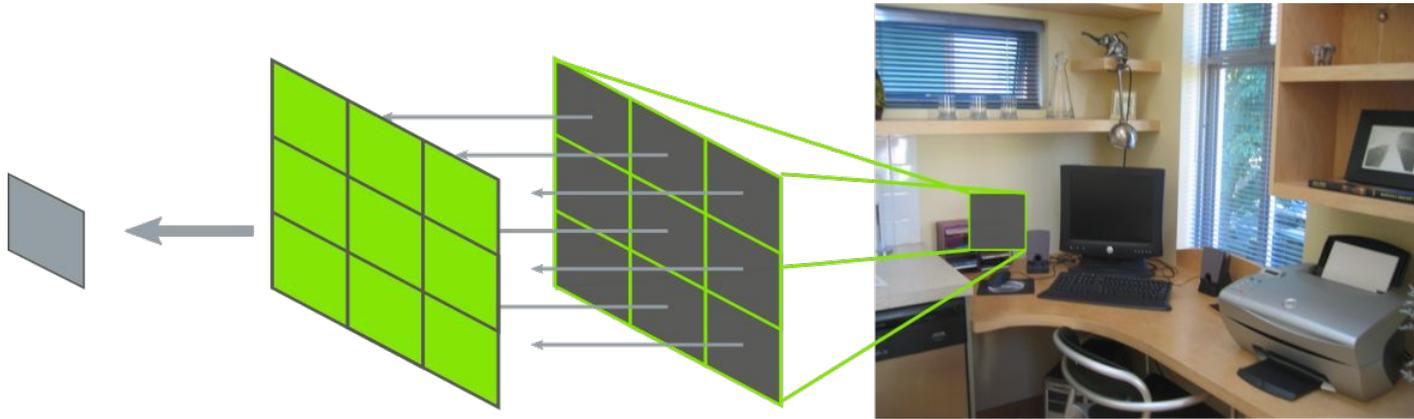


# Convolution on images

Convolution on images

Convolution for image processing

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



# Convolution on images

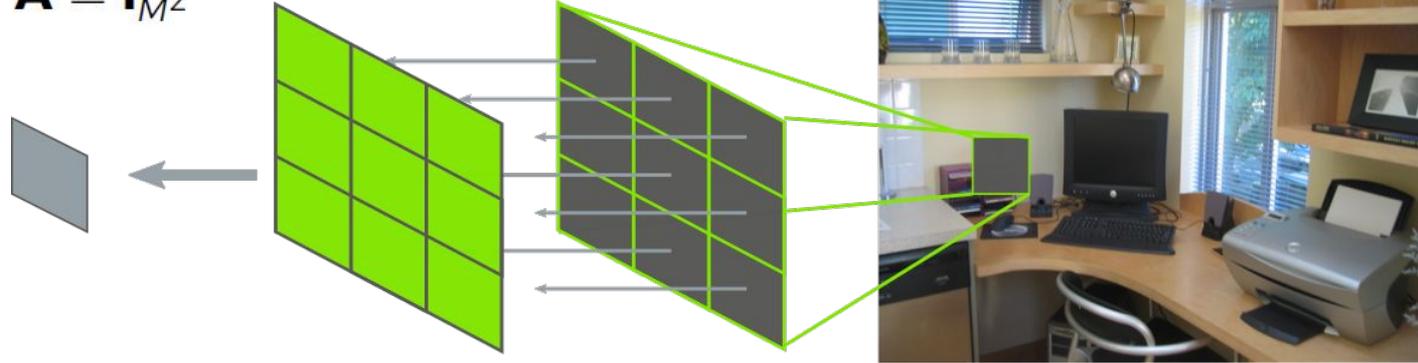
Convolution on images

Convolution for image processing

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

With  $\mathbf{A}$  the alignment matrix

Image processing:  $\mathbf{A} = \mathbf{I}_{M^2}$



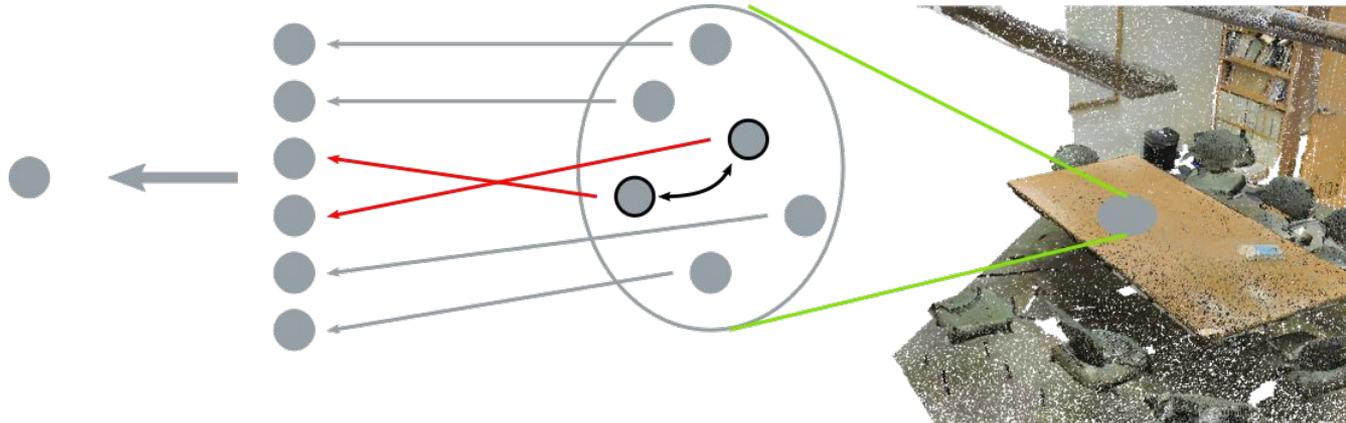
# Convolution for points

Convolution for points

Apply the same formula on a small set of points:

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

Problem:  $\mathbf{A}$  is not permutation invariant



# Convolution on points

Convolution on points

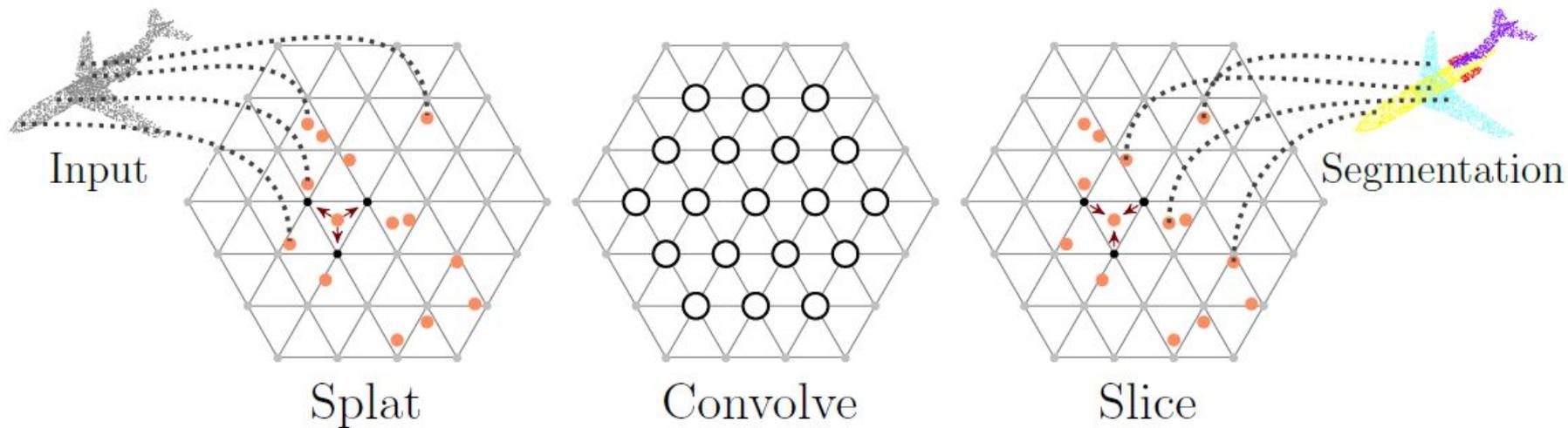
**A** must be estimated from the neighborhood  $\mathcal{N}$  of  $n$ :

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

# SplatNet

SplatNet

**Estimation of  $\mathbf{A}$ :** Interpolation of the features on a regular grid (barycentric

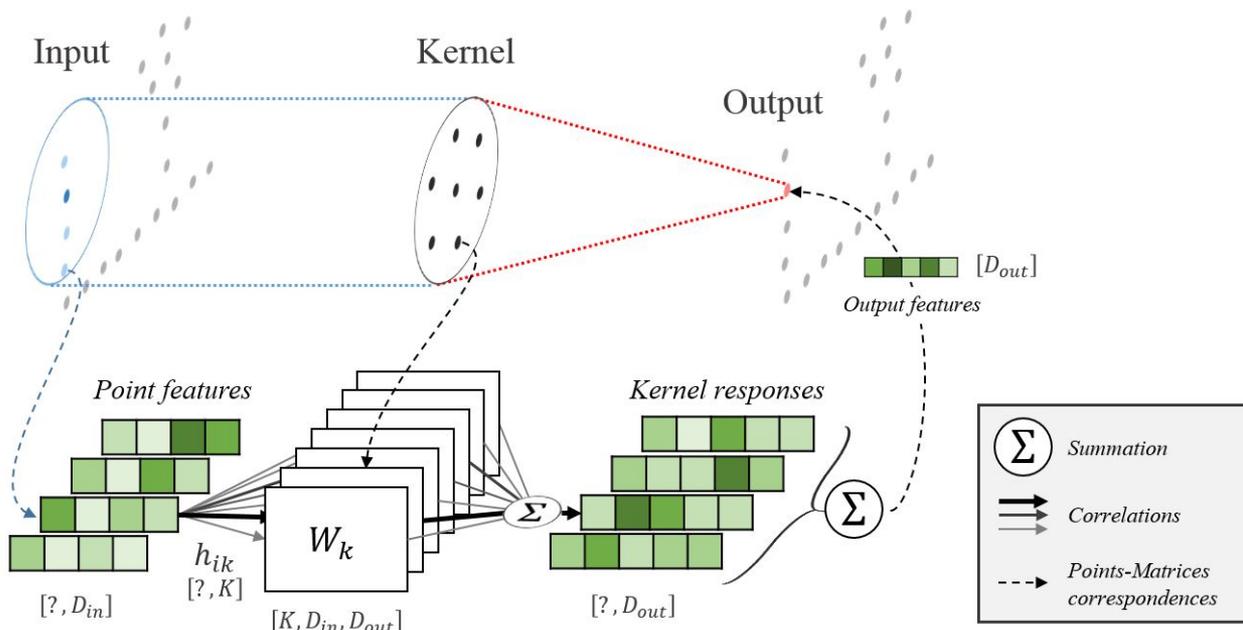


# KPConv

KPConv

$$a_{i,j} = \alpha(y_i, \hat{x}_j) = \max \left( 0, 1 - \frac{\|y_i - \hat{x}_j\|}{\sigma} \right)$$

**Estimation of A:** Create kernel locations in space, weighted interpolation to all kernel location based on distance.



# ConvPoint

## ConvPoint

Estimation of A: Create kernel locations in space, weighted interpolation learned with MLP.

$$a_{i,j} = a(y_i, \hat{x}_j) = \text{MLP}(y_i - \hat{x}_j)$$

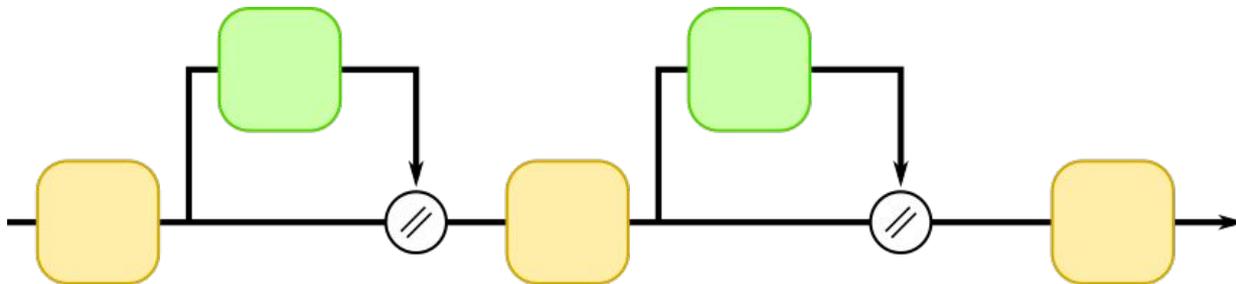
Optimization of both MLP weights and kernel point positions.

# FKACnv

FKACnv

**Estimation of A:** Direct estimation of A using a mini-PointNet.

$$a_{i,j} = a_i(\hat{x}_j) = \text{MLP}_i(\hat{x}_j, \{\hat{x}_k\}_k) \approx \text{PointNet}(\{\hat{x}_k\}_k)$$



# Neighborhood search

Neighborhood search

Convolution is a local operation.

- K-nearest neighbors search
- Ball search

# K-nearest neighbors search

K-nearest neighbors search

Let  $q$  be the support point (center of the neighborhood):

$$\operatorname{argtop}\text{-}K_{\mathbf{p} \in P} \{-\|\mathbf{p} - \mathbf{q}\|\}$$

## Pros:

- All neighborhoods have the same cardinal
- Relatively fast

## Cons:

- Neighborhoods scales vary

# Ball search

## Ball search

Let  $q$  be the support point (center of the neighborhood):

$$\{\mathbf{p} \in P, s.t. \|\mathbf{p} - \mathbf{q}\| < r\}$$

with  $r$  the ball radius.

### Pros:

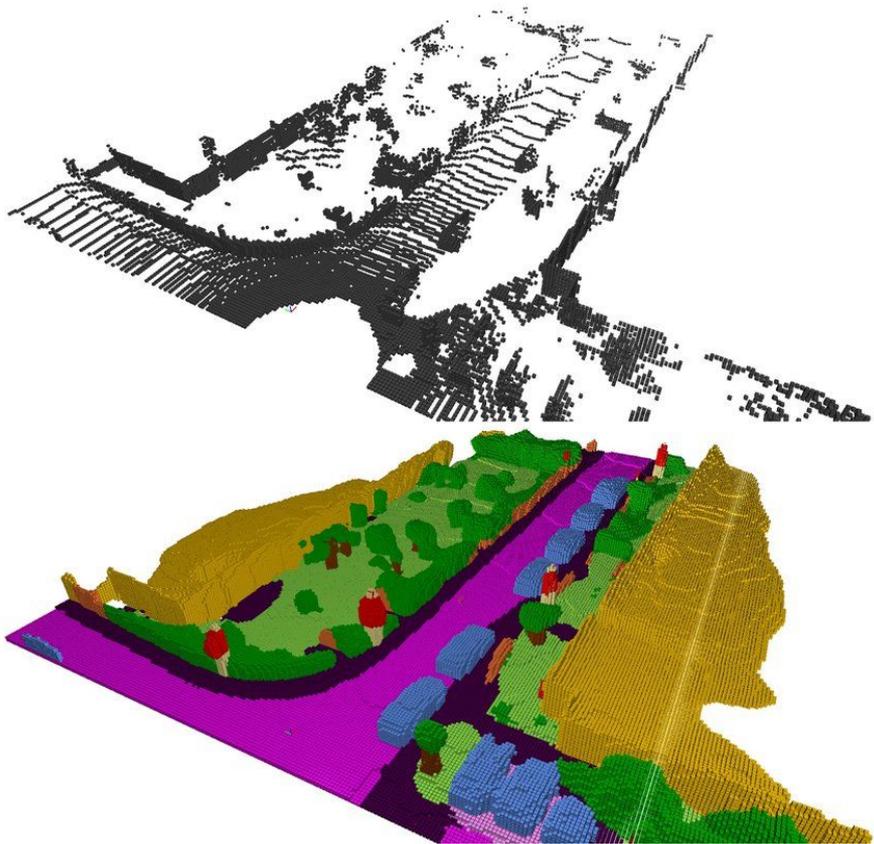
- All neighborhoods have the same scale

### Cons:

- Neighborhoods cardinals (number of points) vary
- Usually slower than K-nn

# Overview

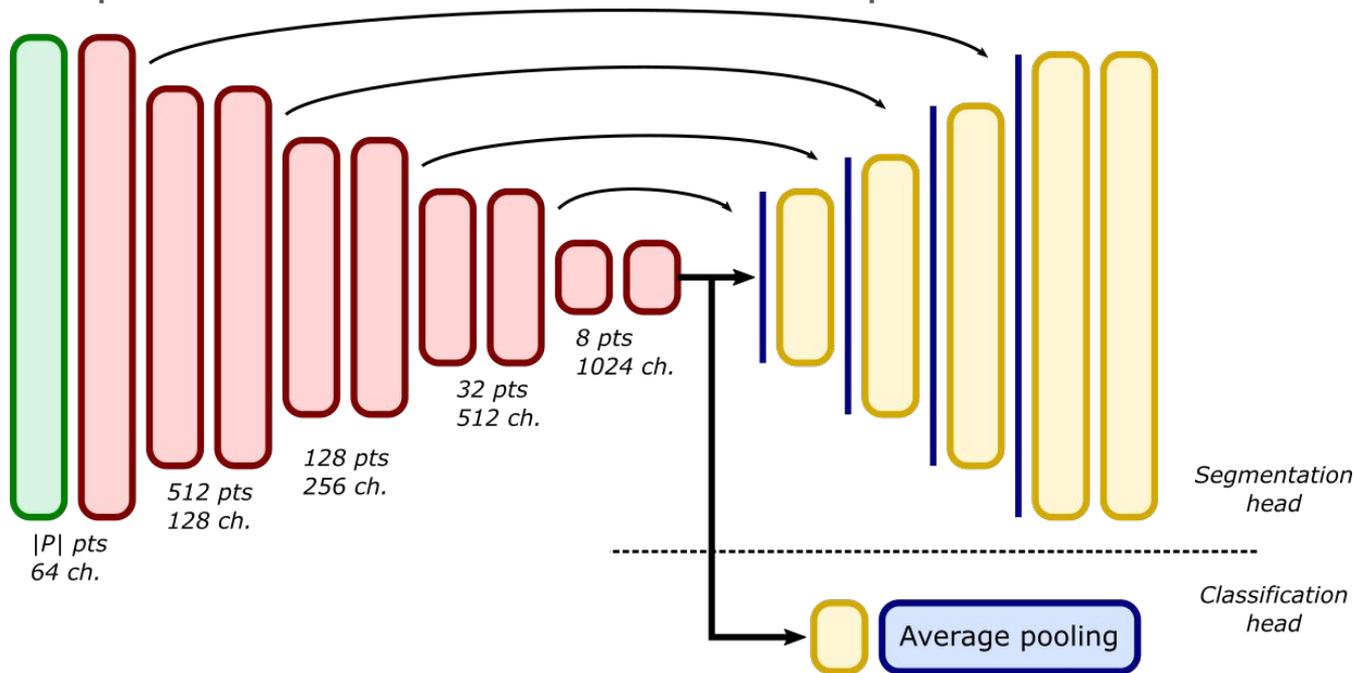
- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers



# Progressive dimension reduction

Progressive dimension reduction

What is the equivalent of stride for convolution on points ?

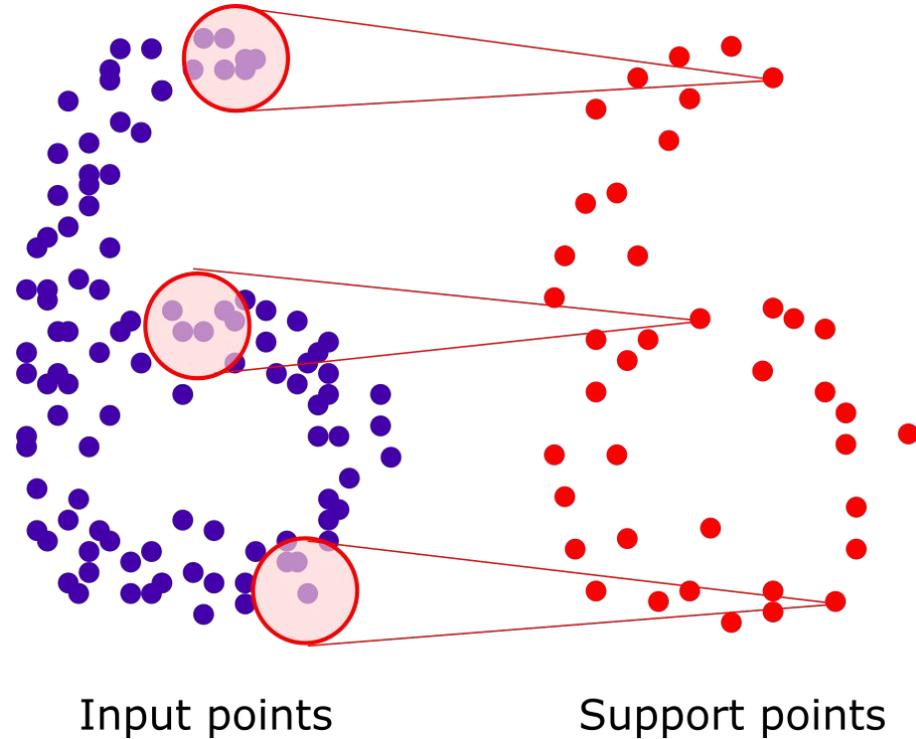


# Support point sampling

Support point sampling

Q (Support points), points used as neighborhood centers for the convolution operation.

Usually Q is a subset of P



# Random sampling

## Random sampling

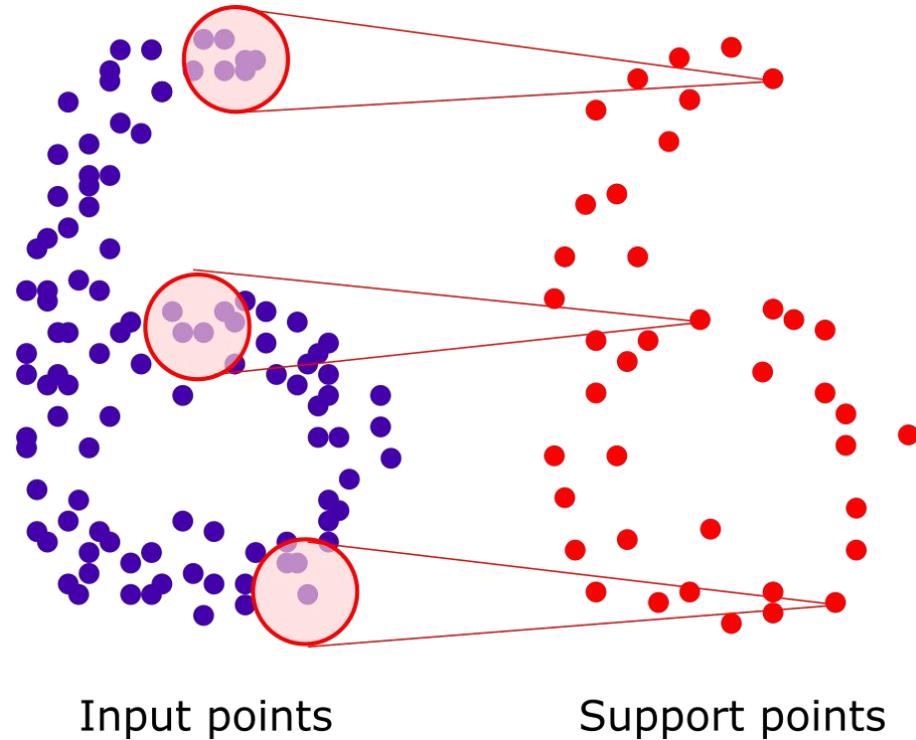
Uniform selection of the input points.

### Pros:

- simple and fast.

### Cons:

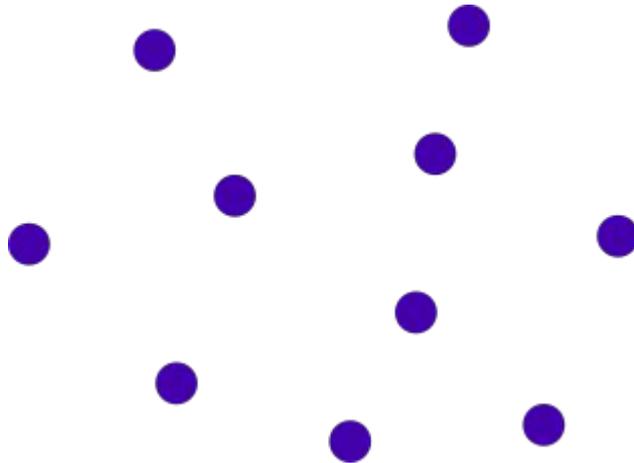
- loss of geometric information on area with low density or extreme points.



# Furthest point sampling

Furthest point sampling

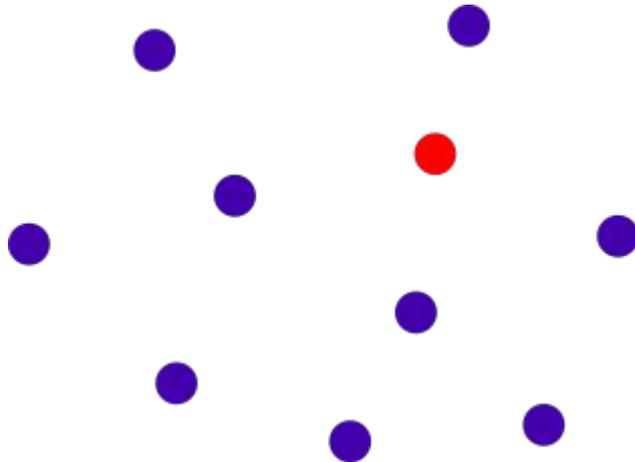
Introduced in PointNet++: iteratively select the further point from the previously selected.



# Furthest point sampling

Furthest point sampling

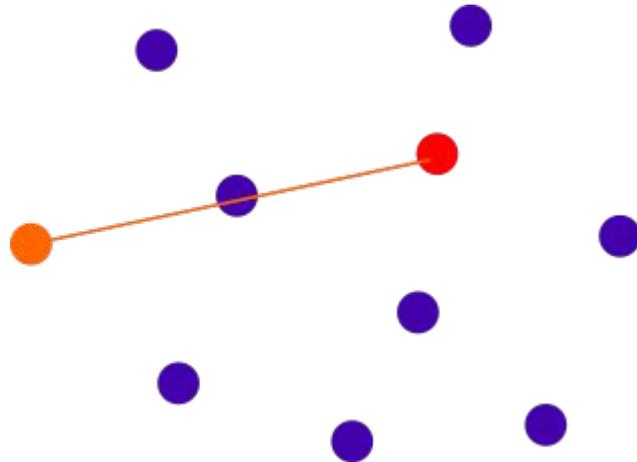
Introduced in PointNet++: iteratively select the further point from the previously selected.



# Furthest point sampling

Furthest point sampling

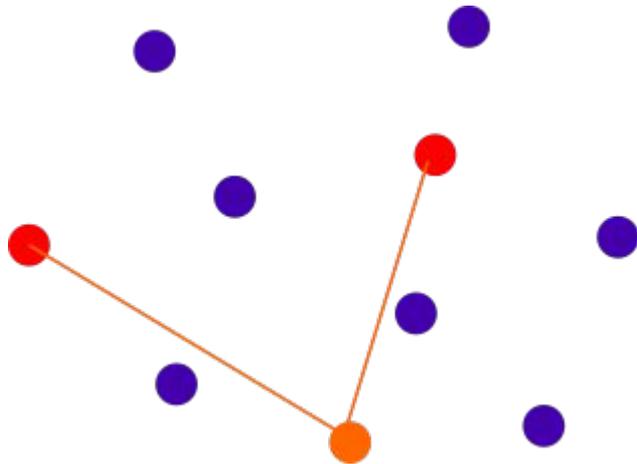
Introduced in PointNet++: iteratively select the further point from the previously selected.



# Furthest point sampling

Furthest point sampling

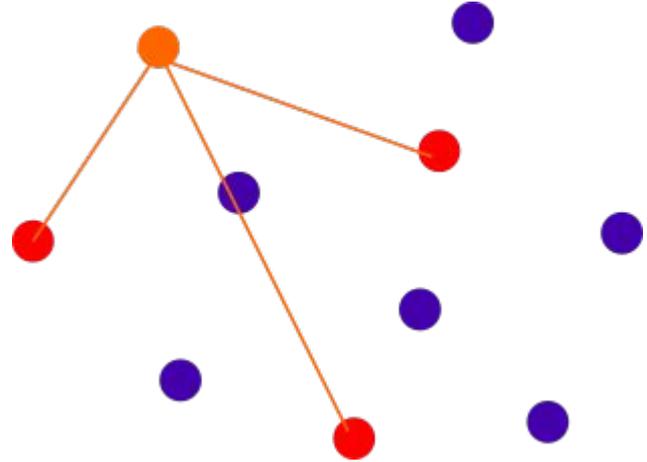
Introduced in PointNet++: iteratively select the further point from the previously selected.



# Furthest point sampling

Furthest point sampling

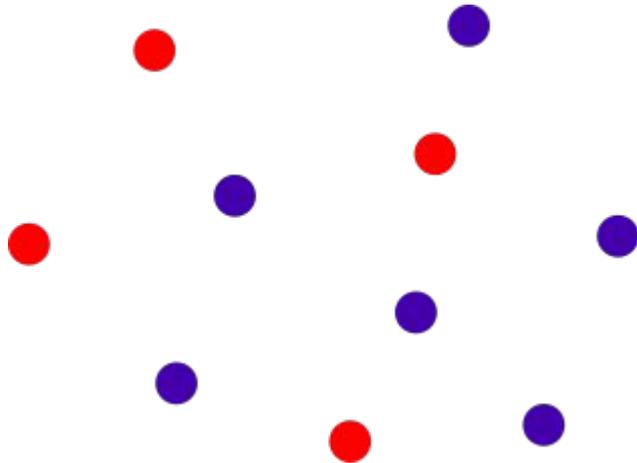
Introduced in PointNet++: iteratively select the further point from the previously selected.



# Furthest point sampling

Furthest point sampling

Introduced in PointNet++: iteratively select the further point from the previously selected.



# Voxel-grid sampling

## Voxel-grid sampling

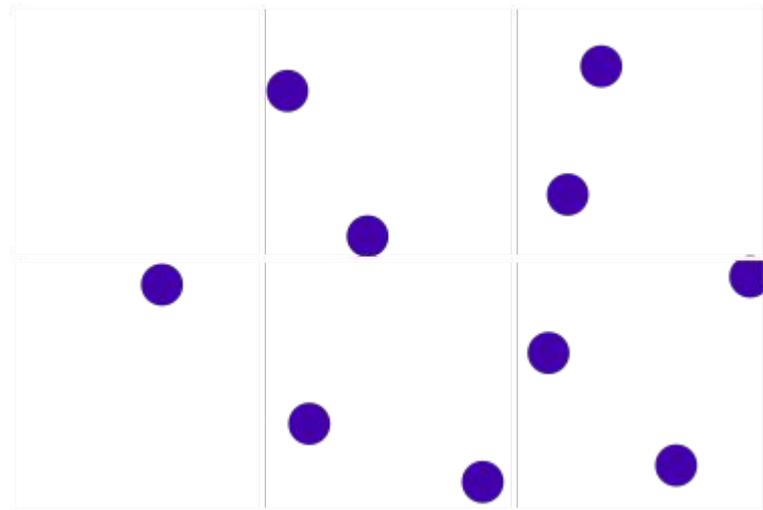
Apply a voxel pooling: select a point in each voxel.

### Pros:

- fast

### Cons:

- voxel size is extra parameter, may lead to variable number of points



# Voxel-grid sampling

## Voxel-grid sampling

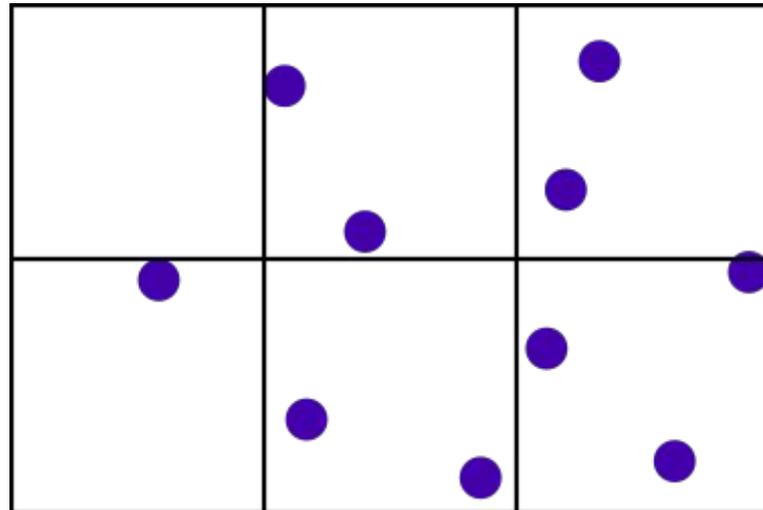
Apply a voxel pooling: select a point in each voxel.

### Pros:

- fast

### Cons:

- voxel size is extra parameter, may lead to variable number of points



# Voxel-grid sampling

## Voxel-grid sampling

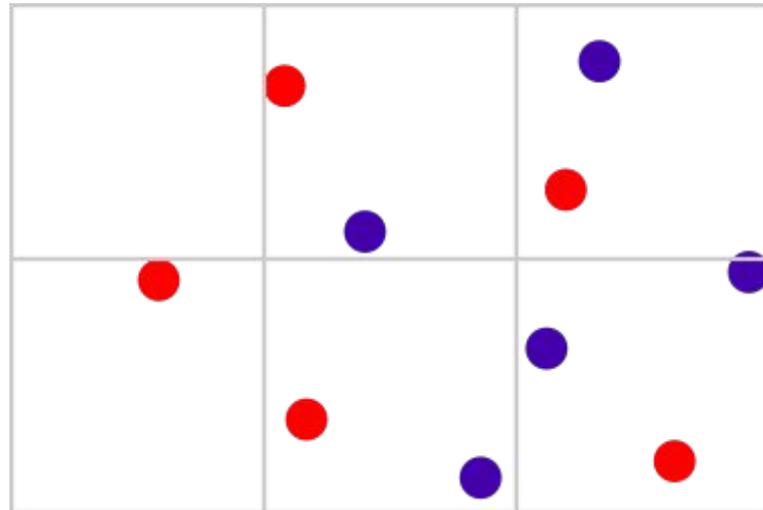
Apply a voxel pooling: select a point in each voxel.

### Pros:

- fast

### Cons:

- voxel size is extra parameter, may lead to variable number of points



# Voxel-grid sampling

## Voxel-grid sampling

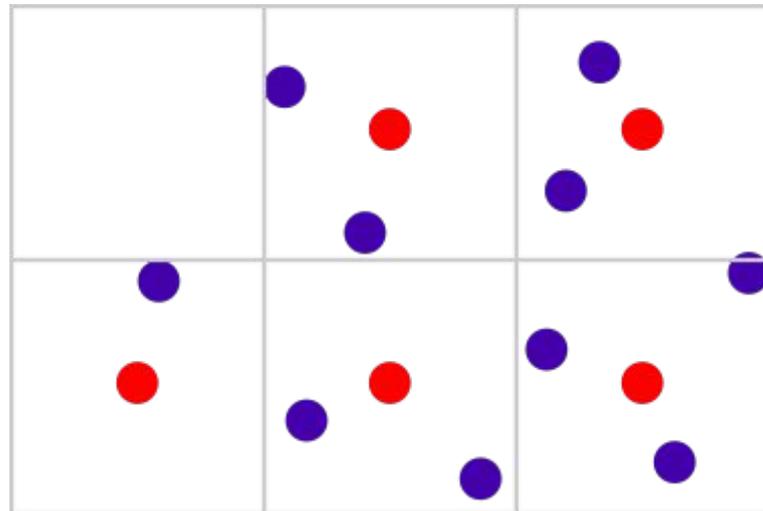
Apply a voxel pooling: select a point in each voxel.

### Pros:

- fast

### Cons:

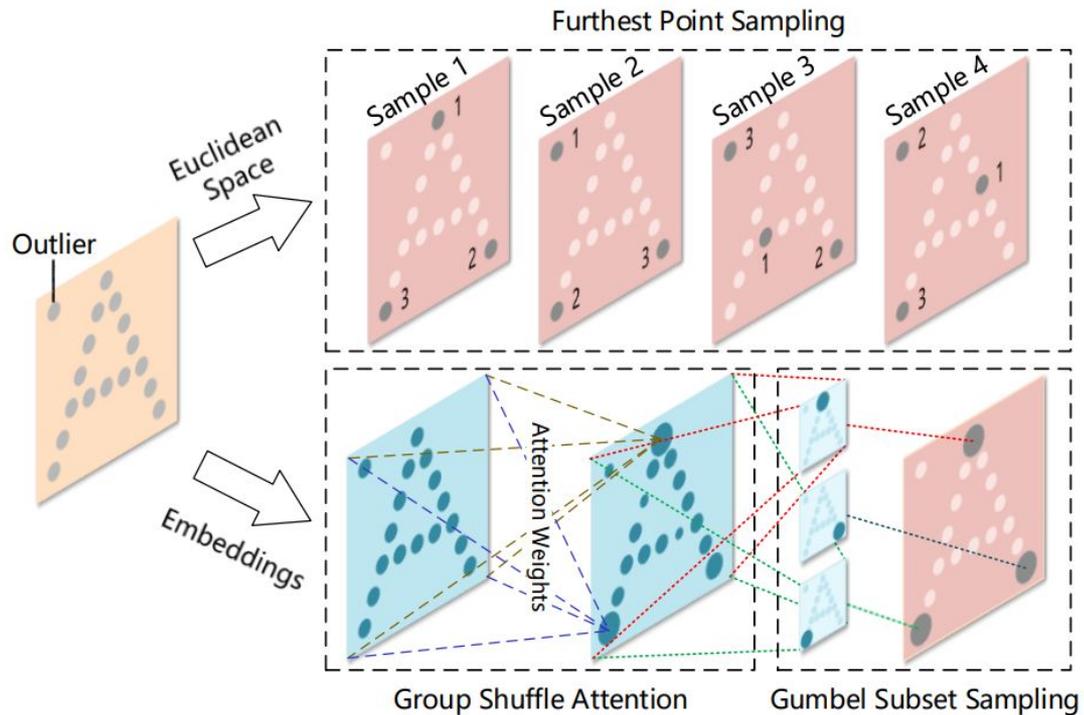
- voxel size is extra parameter, may lead to variable number of points



# Attention pooling

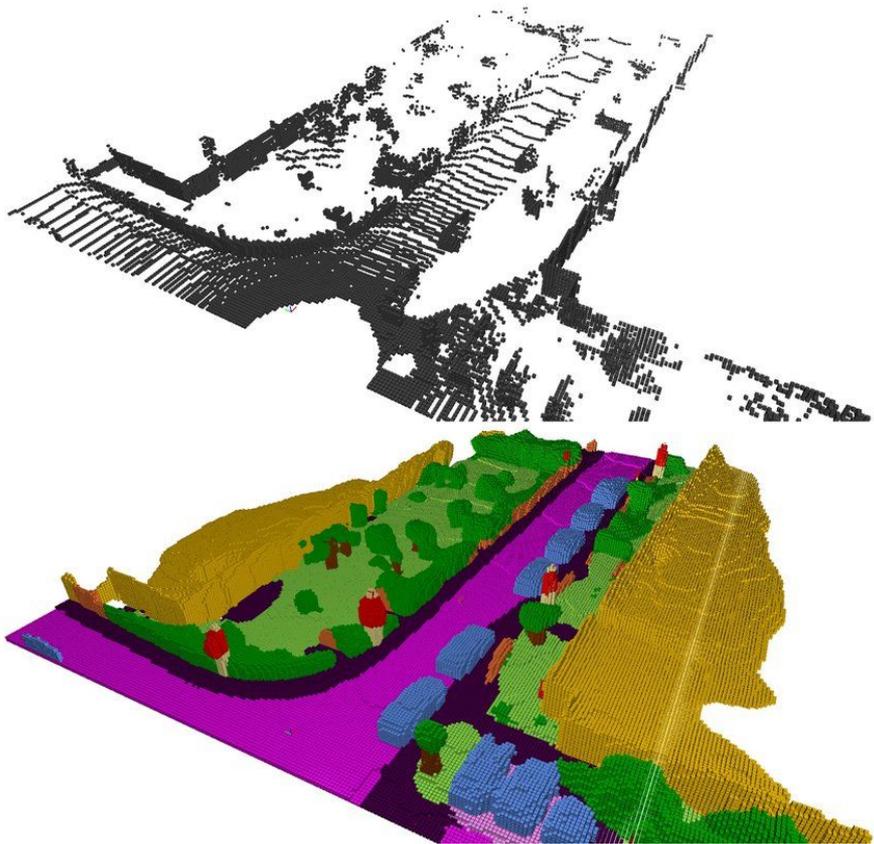
## Attention pooling

Learned attention on the points for outlier robustness.



# Overview

- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers



# 3D grid convolution

## II. Voxels

3D convolution for an grid patch centered on  $n$ :

$$\mathbf{h}[n] = \sum_{f \in \{1, \dots, C\}} \sum_{m \in \{-M/2, \dots, M/2\}^3} \mathbf{K}_f[m] \mathbf{f}_f[n + m]$$

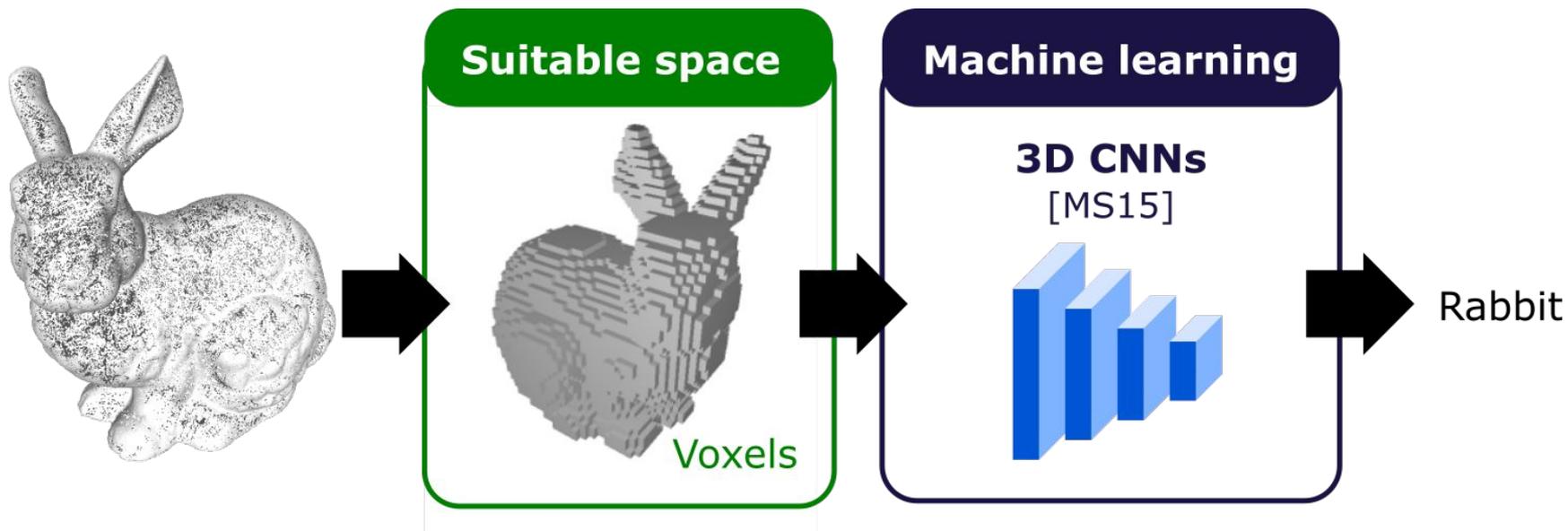
$\mathbf{f}$ : input features

$\mathbf{K}$ : convolution kernel

*How to represent the scene as a 3D grid?*

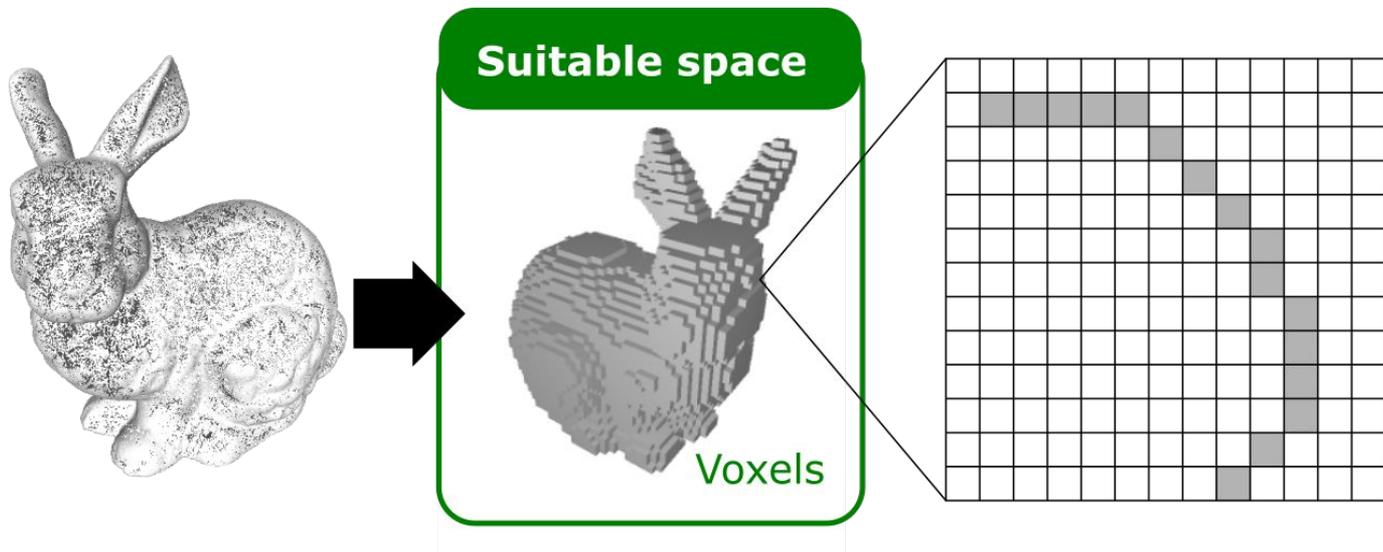
# 3D projections (voxels)

## II. Voxels



# Memory

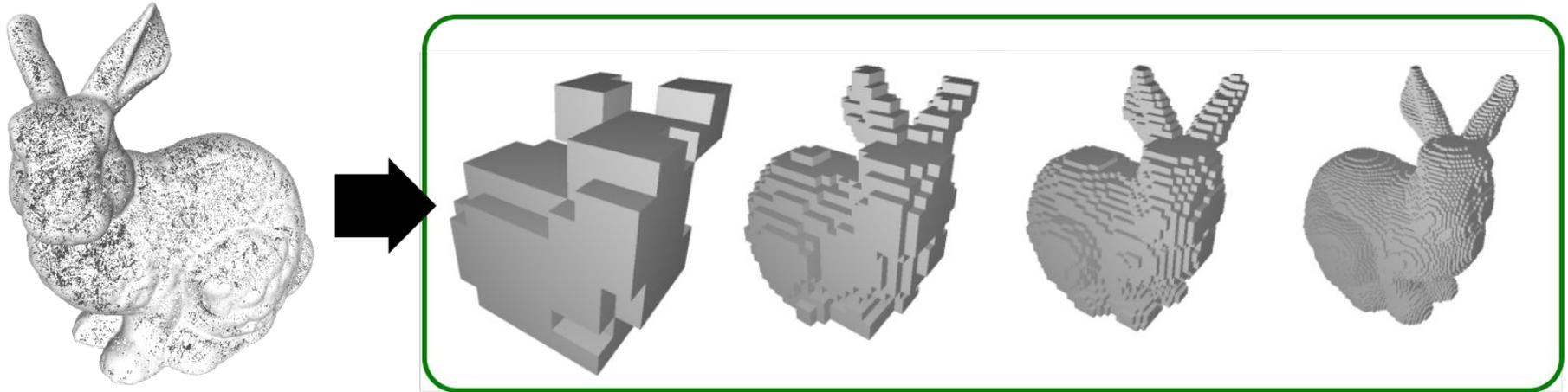
## II. Voxels



Point clouds sampled on surfaces are very sparse.  
We mostly encode empty voxels!

# Memory vs representation power

## II. Voxels



Memory efficiency vs information loss

# Are voxels doomed?

## II. Voxels

Voxels are OK for small scenes:

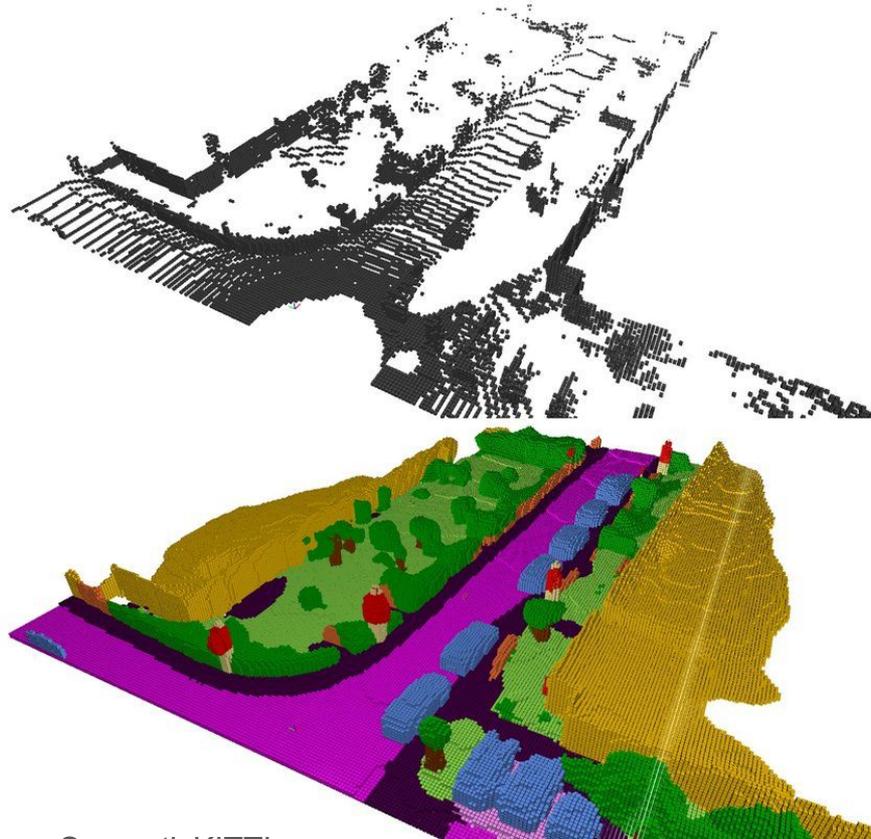
### Shapes:

→  $32 \times 32 \times 32 = 32768$  voxels

### Scenes:

→  $[100\text{m}, 100\text{m}, 10\text{m}]$ , vox 0.05:  
800M voxels

While for a lidar point cloud only  
~150k voxels are filled (0.02%)

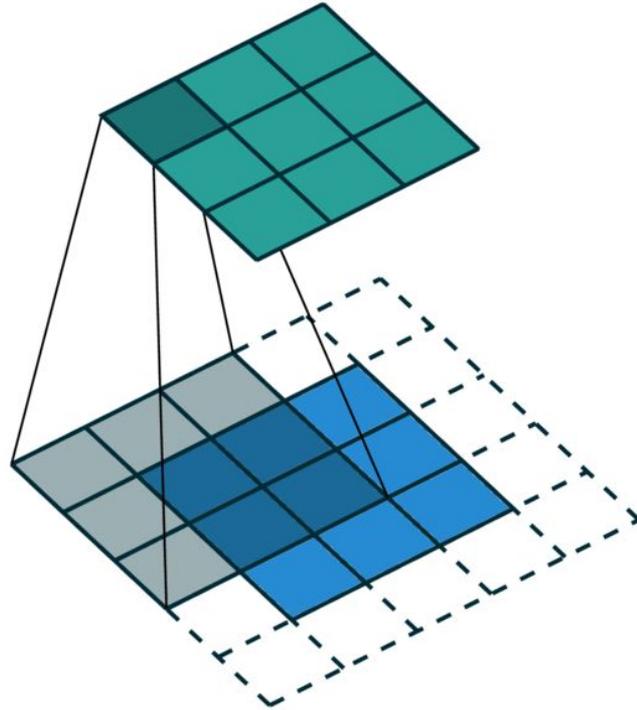


SemanticKITTI

# Idea?

## II. Voxels

Look at the functioning of the convolution for dense input



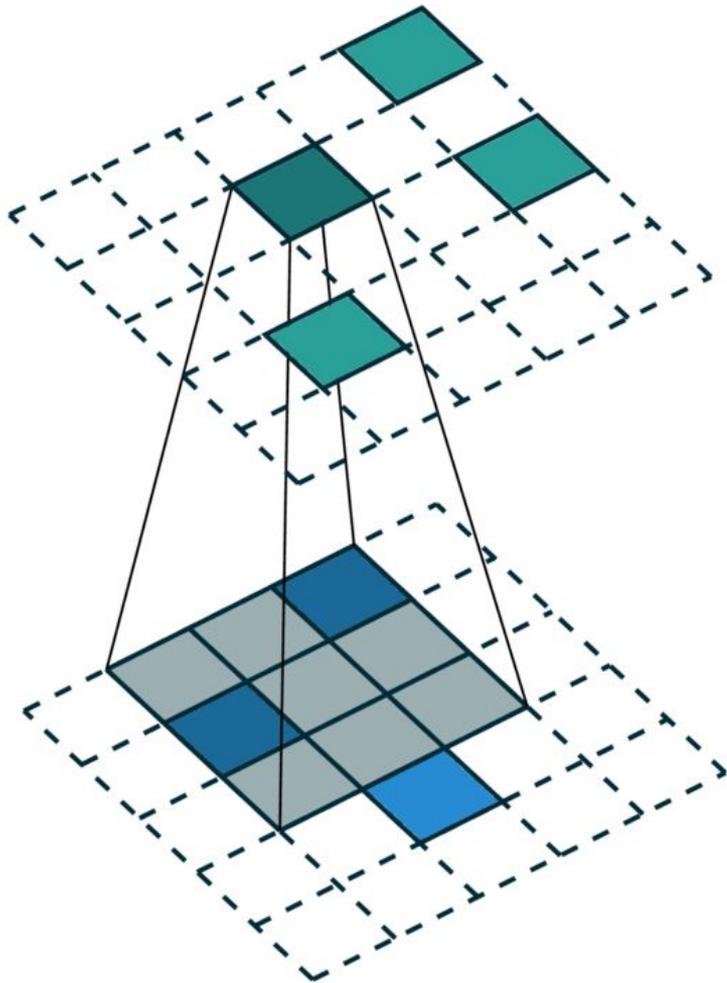
# Idea?

## II. Voxels

Look at the functioning of the convolution for dense input

Mimic the behavior only at point location

→ sparse convolution



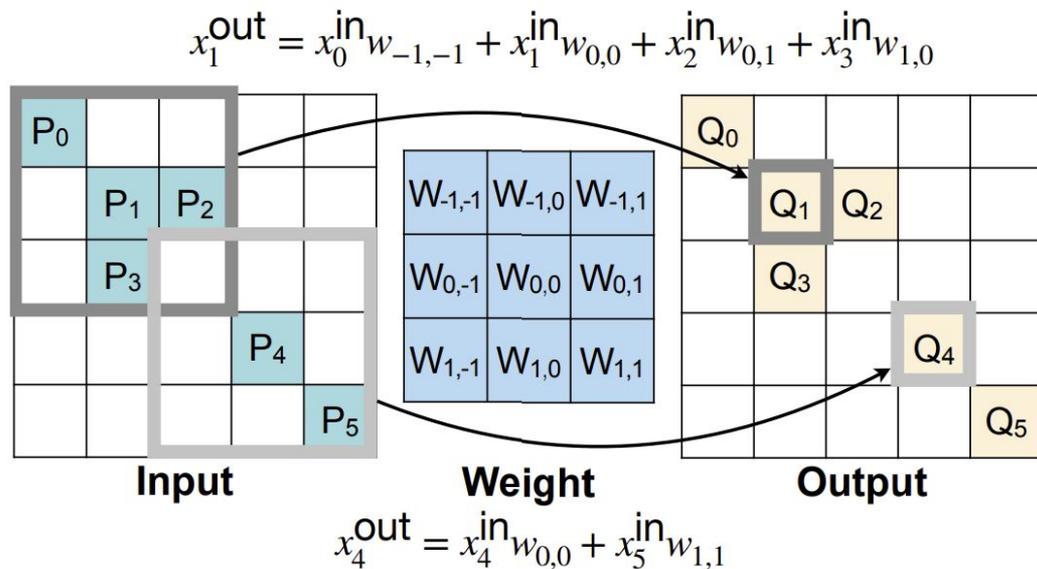
# Sparse convolutions

## II. Voxels

Look at the functioning of the convolution for dense input

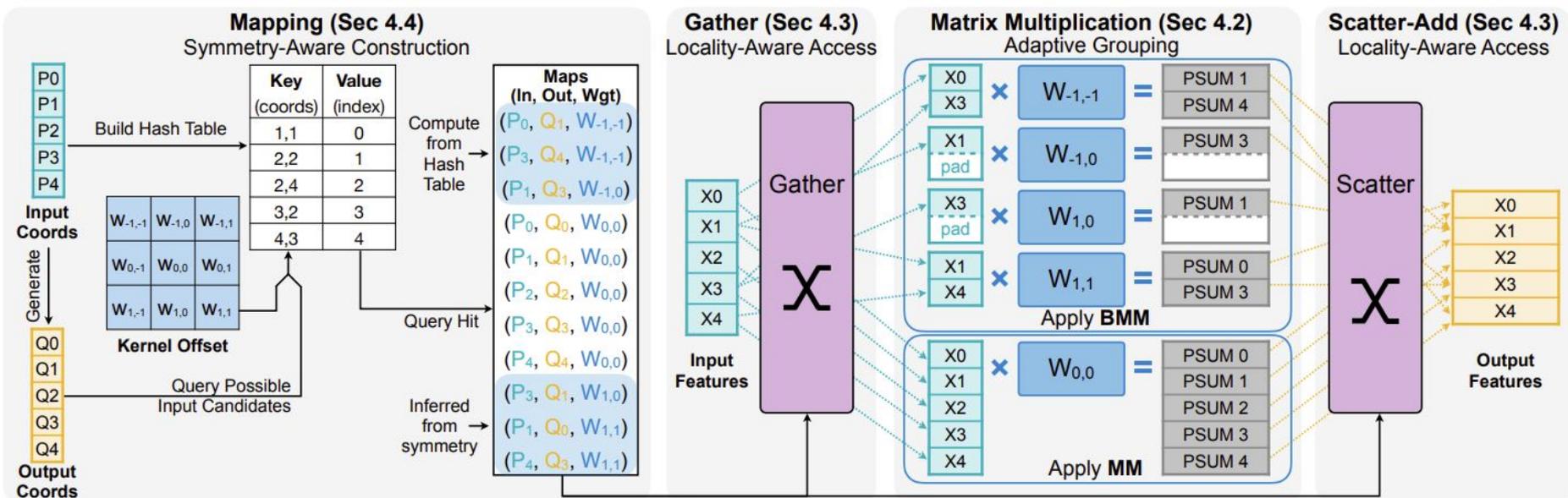
Mimic the behavior only at point location

→ sparse convolution



# Sparse convolutions

## II. Voxels



# Alternative: sparse convolutions

## II. Voxels

Use sparse convolution for memory saving: do not code the empty cells.

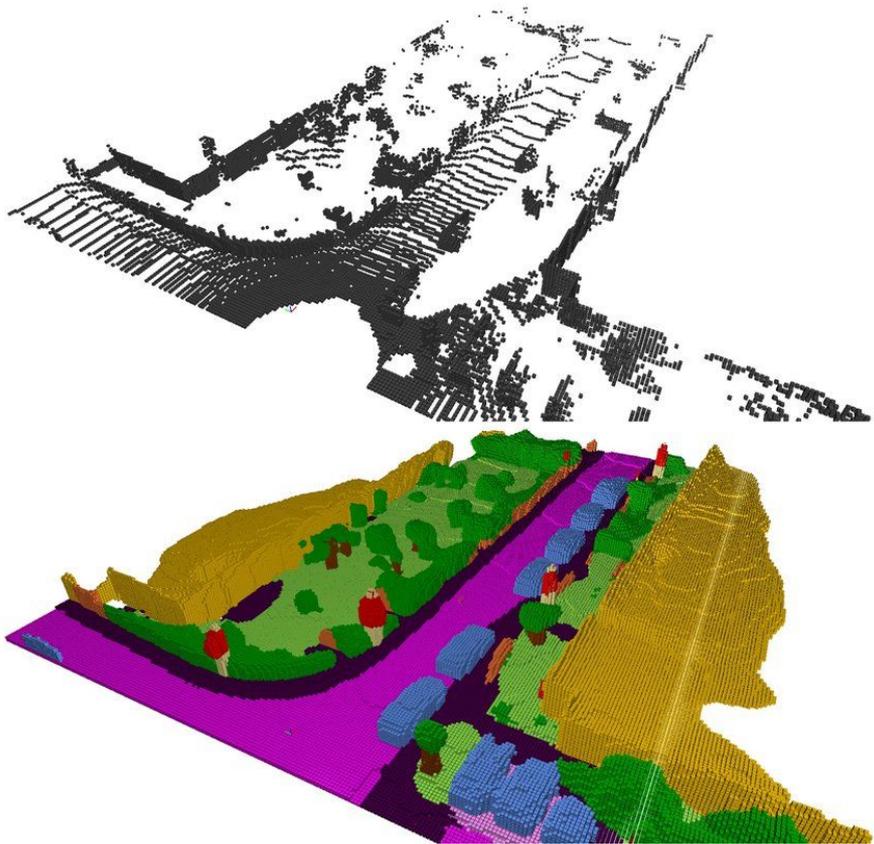
- Minkowski engine (Nvidia)
- SparseConvNet (Facebook)
- Torchsparse
- Spconv

Drawback: slower than dense convolution, extensive use of CPUs.

Only available for Nvidia hardware

# Overview

- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers

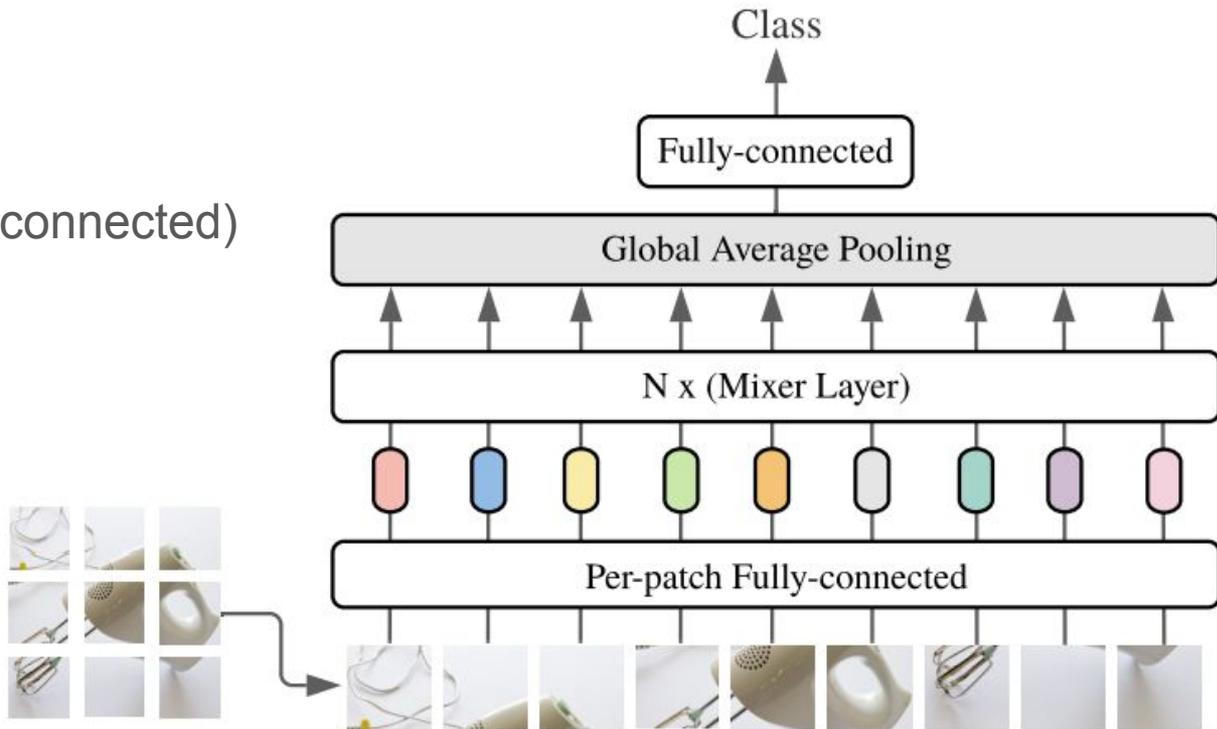


# MLP-Mixer

## III-A Mixers

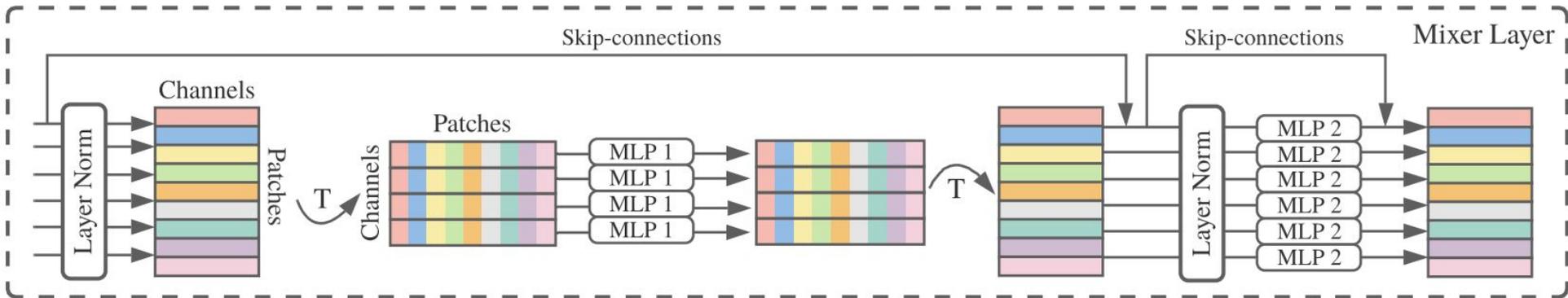
### Image backbone

- Patchification
- Patch encoding (fully connected)
- N x Mixer Layer
- Global pooling
- Classification head



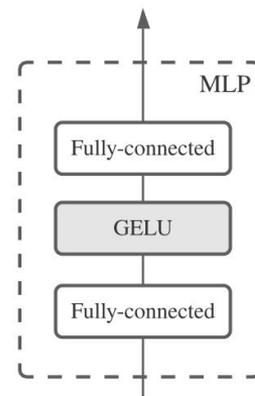
# MLP-Mixer

## III-A Mixers



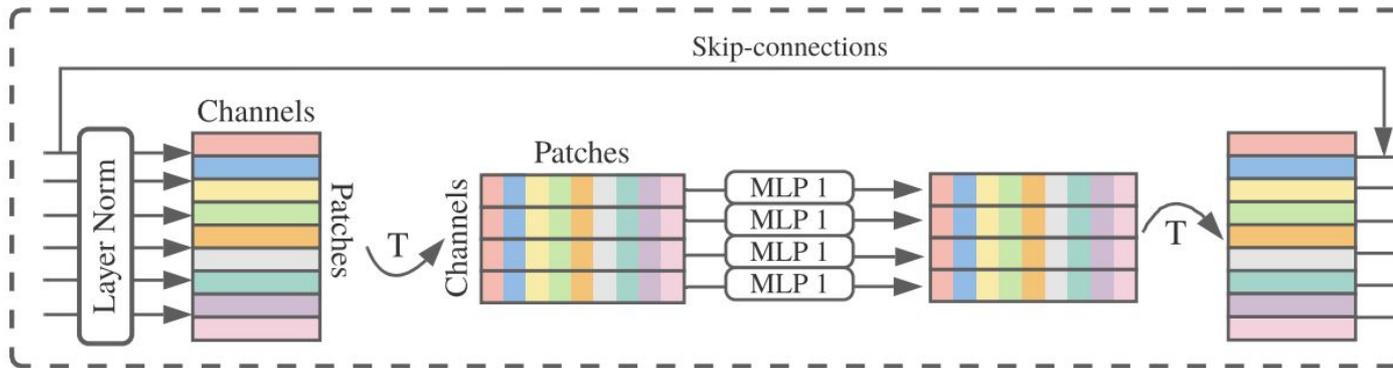
Two sub-blocks:

- **Spatial Mixing:** mixes the patch per channel
- **Spectral Mixing:** mixes the channels per patch



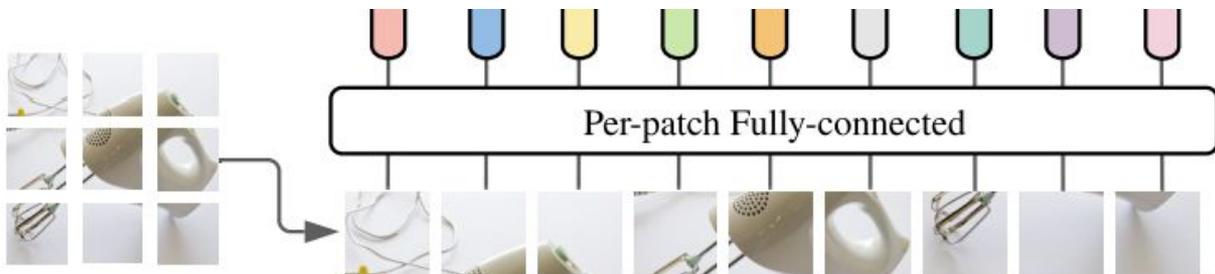
# MLP-Mixer

## III-A Mixers



Why does it work?

Patches are always in the same order



Incompatible with point clouds

# PointMixer

## III-A Mixers

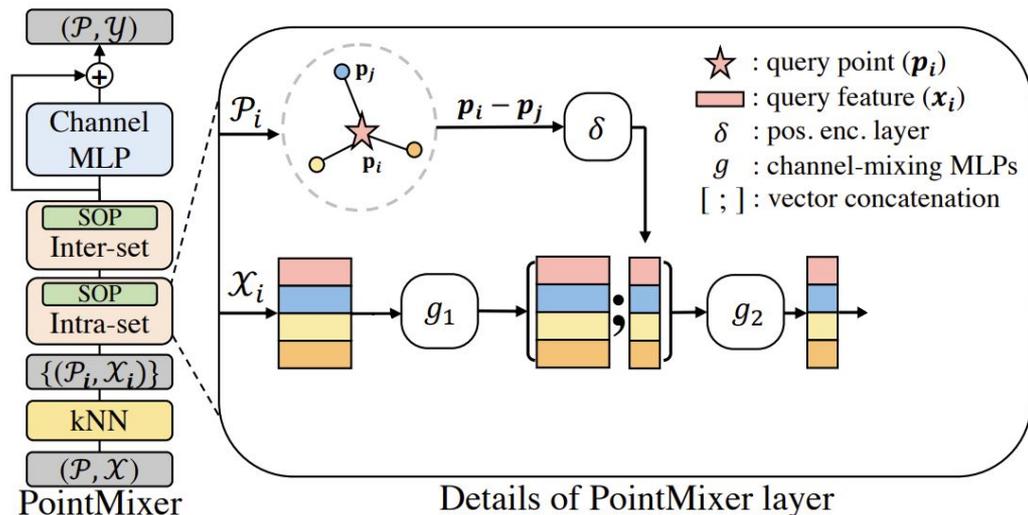
Neighborhood:  $\mathcal{X}_i = \{\mathbf{x}_j\}$

1. Predict a score vector

$$\mathbf{s} = [s_1, \dots, s_K] \quad \mathbf{s} \in \mathbb{R}^K$$

With

$$s_j = g_2 \left( [g_1(\mathbf{x}_j); \delta(\mathbf{p}_i - \mathbf{p}_j)] \right)$$



# PointMixer

## III-A Mixers

Neighborhood:  $\mathcal{X}_i = \{\mathbf{x}_j\}$

1. Predict a score vector

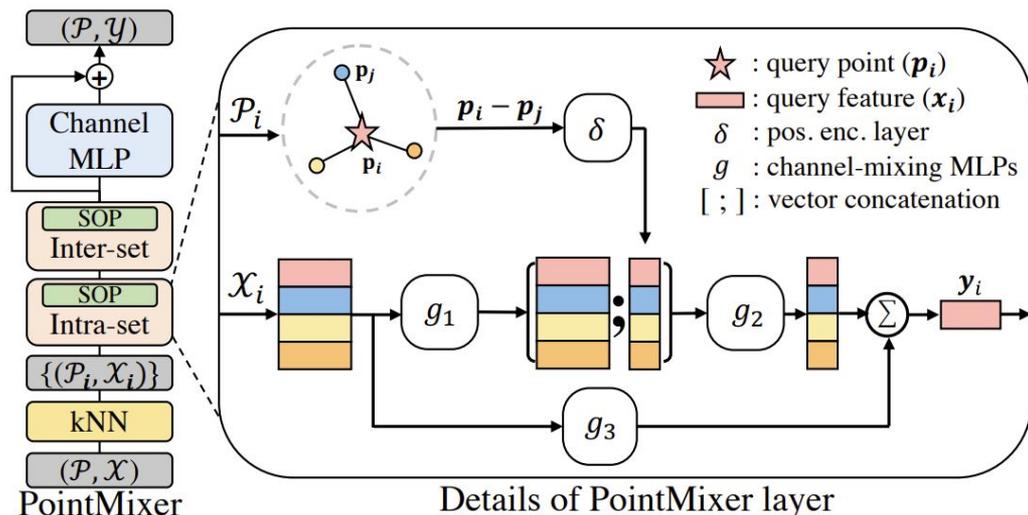
$$\mathbf{s} = [s_1, \dots, s_K] \quad \mathbf{s} \in \mathbb{R}^K$$

With

$$s_j = g_2 \left( [g_1(\mathbf{x}_j); \delta(\mathbf{p}_i - \mathbf{p}_j)] \right)$$

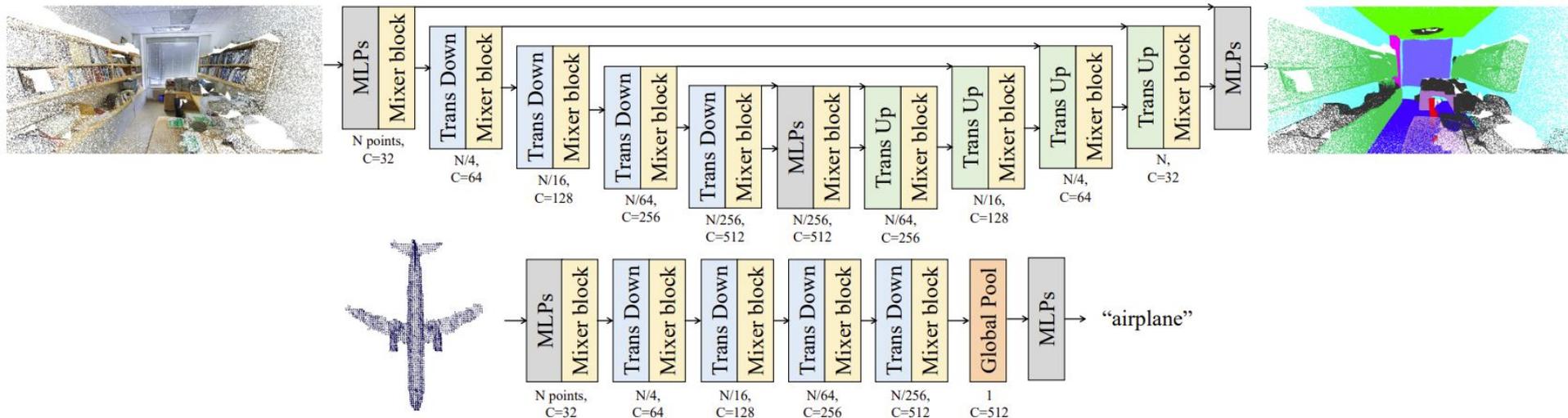
2. Use the scores to weight the features

$$\mathbf{y}_i = \sum_{j \in \mathcal{M}_i} \text{softmax}(s_j) \odot g_3(\mathbf{x}_j),$$



# PointMixer

## III-A Mixers

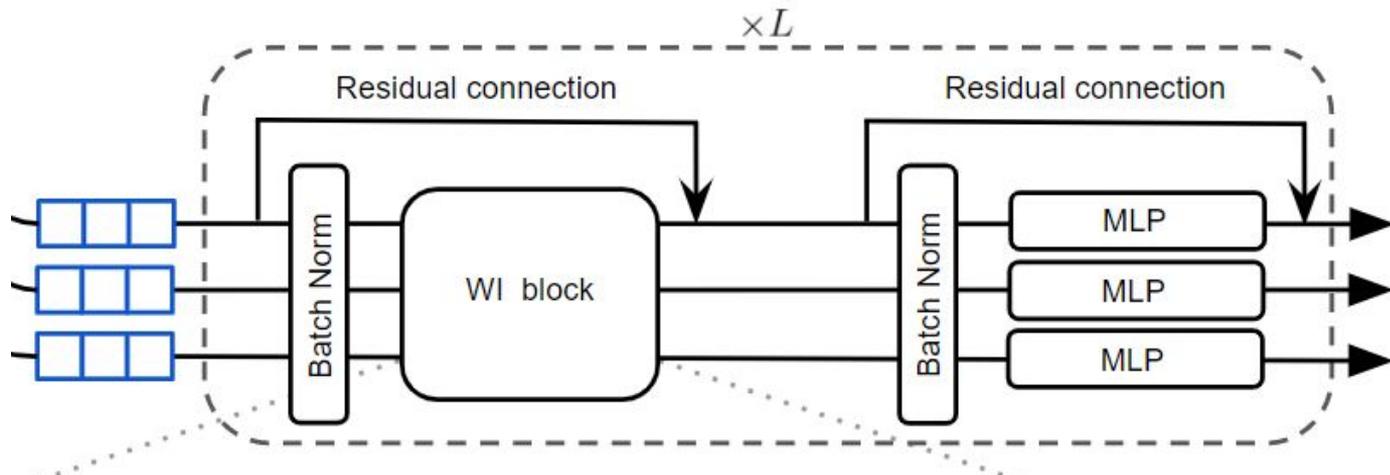


(a) PointMixer network for the dense prediction tasks (top) and the classification task (down).

**Architecture:** U-Net (closer to convolutional architectures than MLP-Mixers)

# WaffleIron

## III-A Mixers

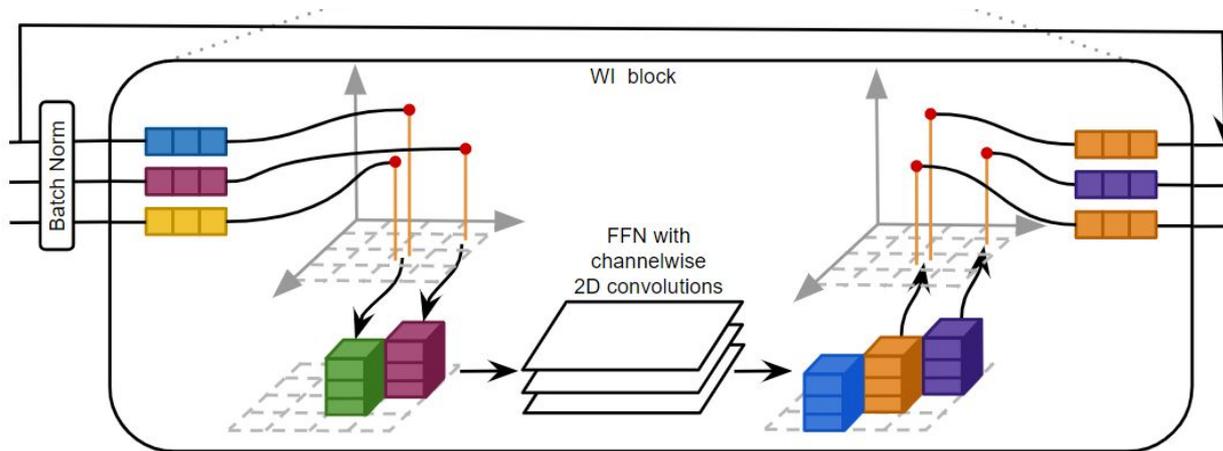


Architecture similar to MLP-Mixer:

- Spatial mixing (WI block)
- Channel mixing (MLP)

# WaffleIron

## III-A Mixers

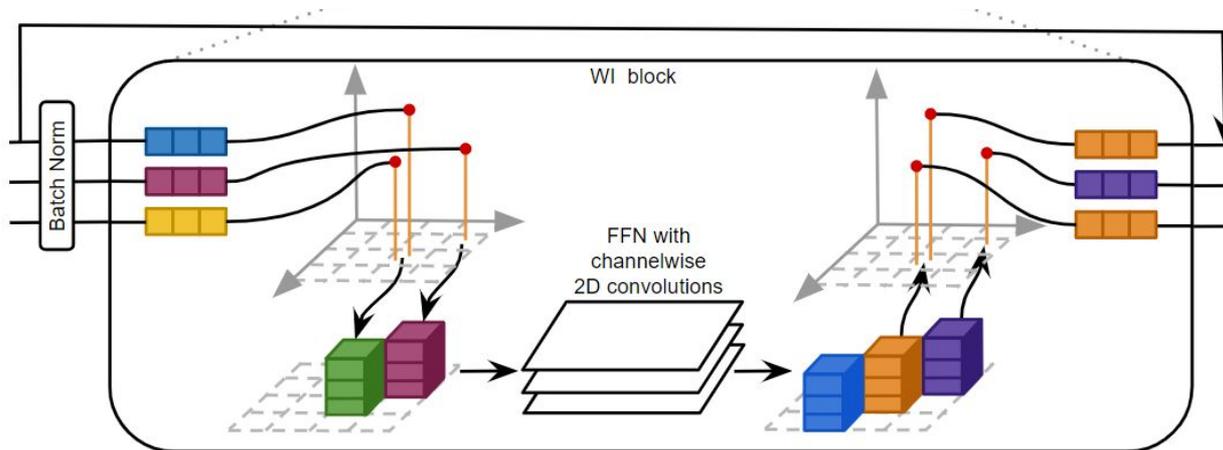


## Spatial mixing:

- Project on a plane → makes it order invariant
- Apply convolutions
- Un-project to planes

# WaffleIron

## III-A Mixers

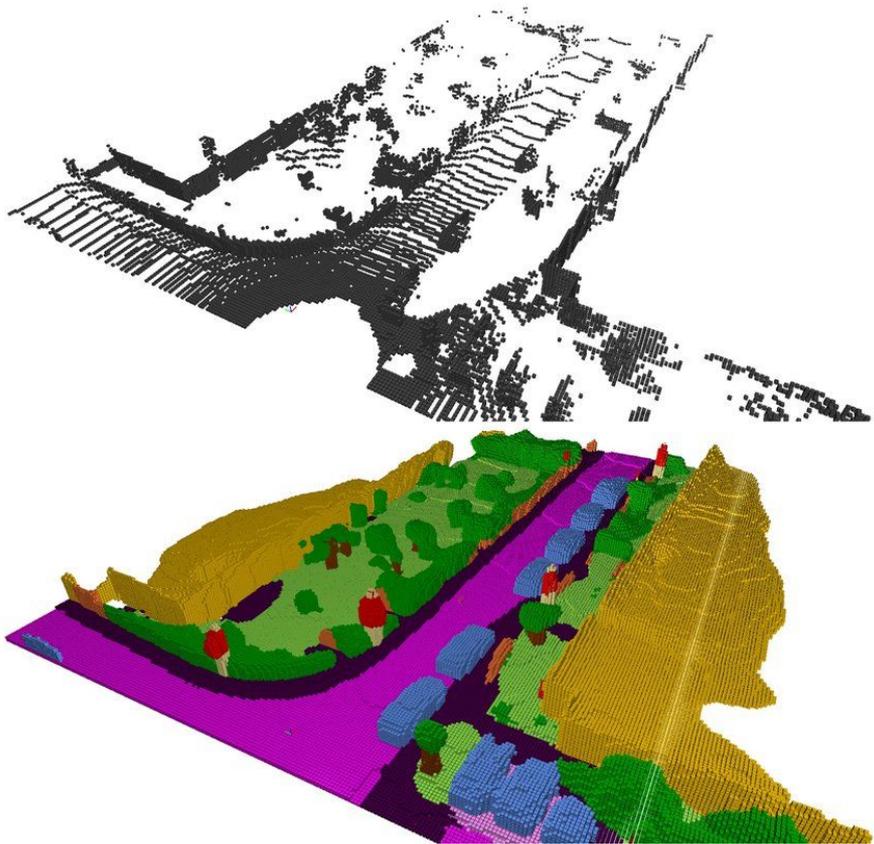


### Advantage:

Do not rely on SparseConv → can be used on any hardware / any deep learning framework

# Overview

- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers



# Transformers

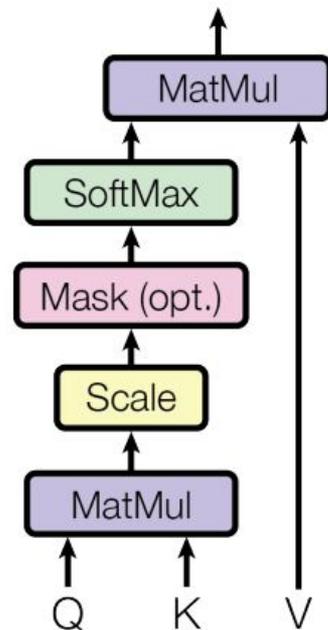
## III-B Transformers

Attention as defined for transformers:

- Base block of all recent architectures (LLMs, VLM, ViTs...)
- Order invariant by design

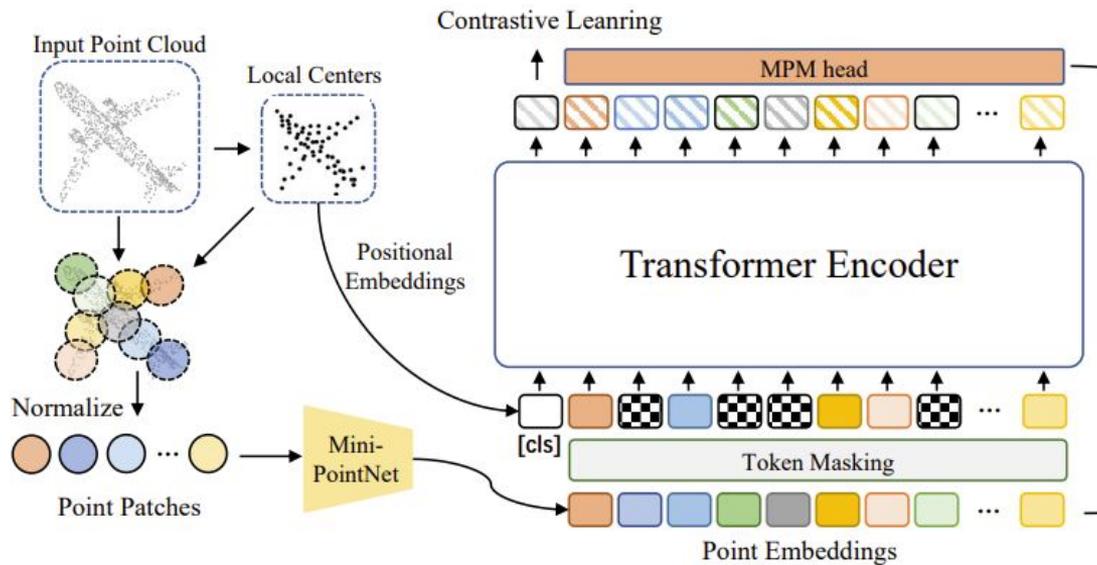
→ Suitable for point clouds

## Scaled Dot-Product Attention



# PointBert

## III-B Transformers



## Transformer architecture

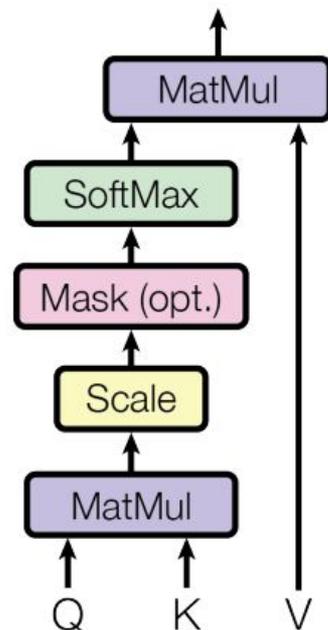
# Transformers

## III-B Transformers

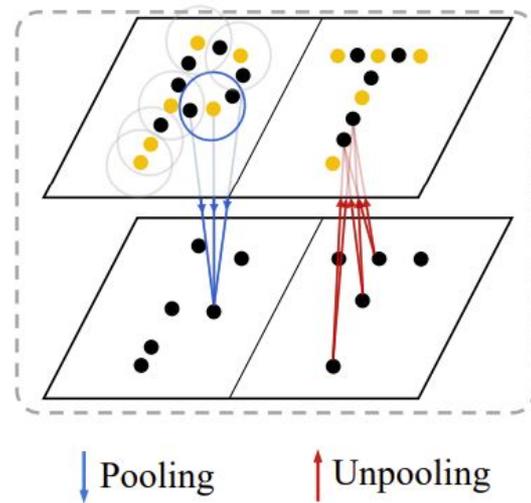
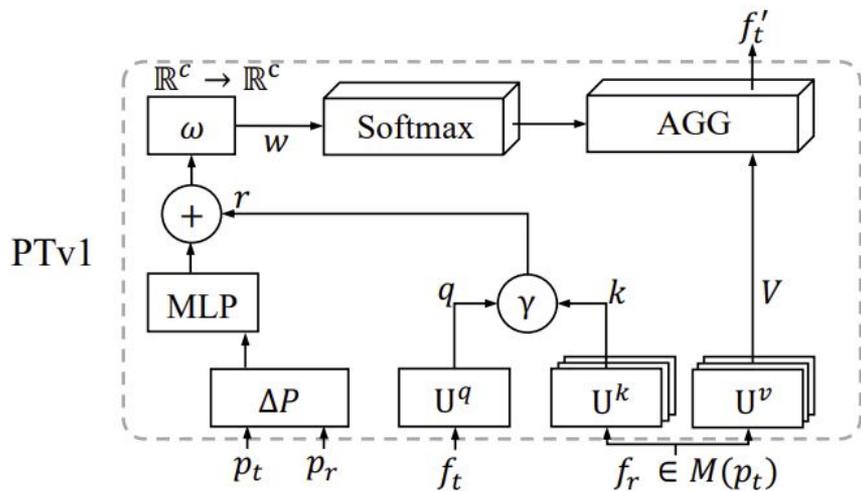
### Difficulties

- Attention scales quadratically in memory (naive implementation)
  - Efficient attention, linear depending on the number of queries / keys / values
- Point clouds are large
  - attention matrix resolution may be under the float precision

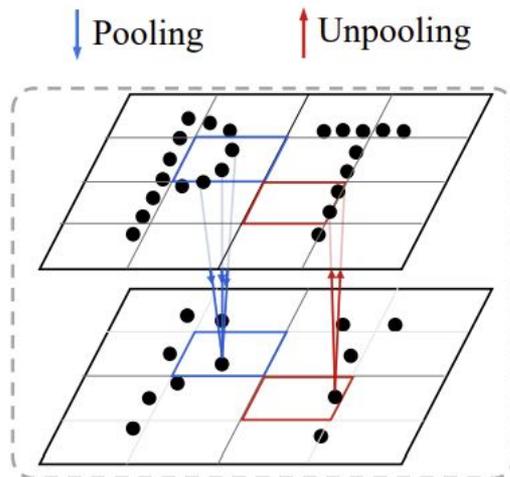
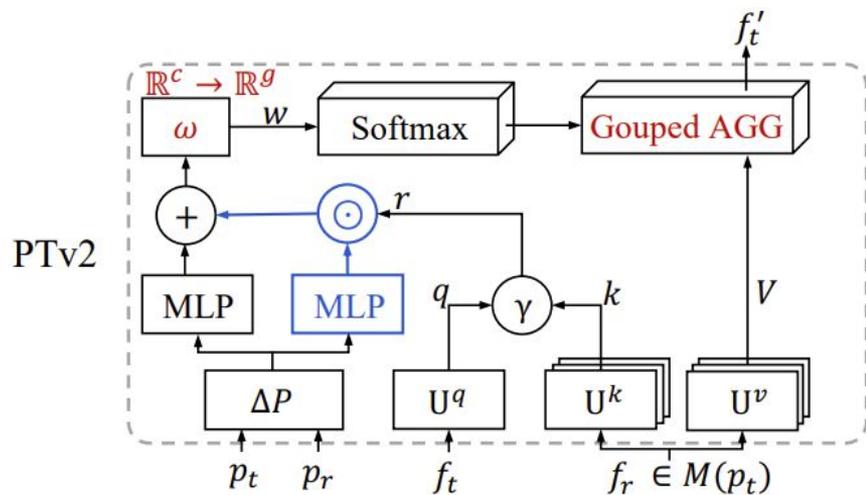
### Scaled Dot-Product Attention



# PointTransformer v1



# PointTransformer v2



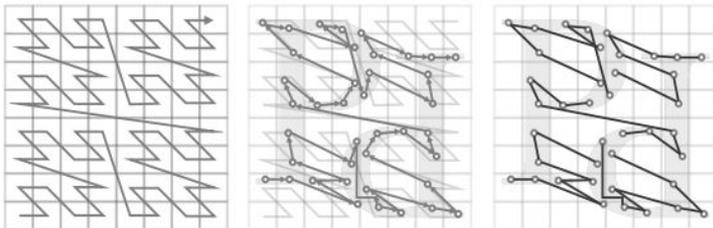
# PointTransformer v3

U-Net architectures

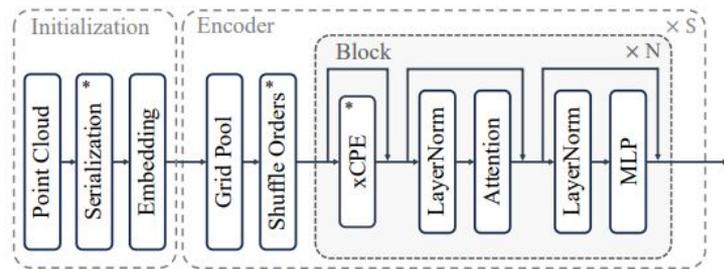
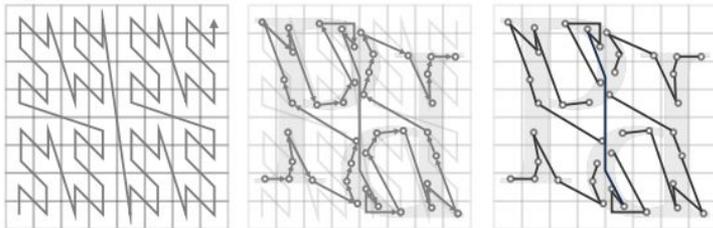
Neighborhood defined by space filling curves

Attention on multiple scales

(a) Z-order



(c) Trans Z-order





(a) Breakdown of trainable parameters



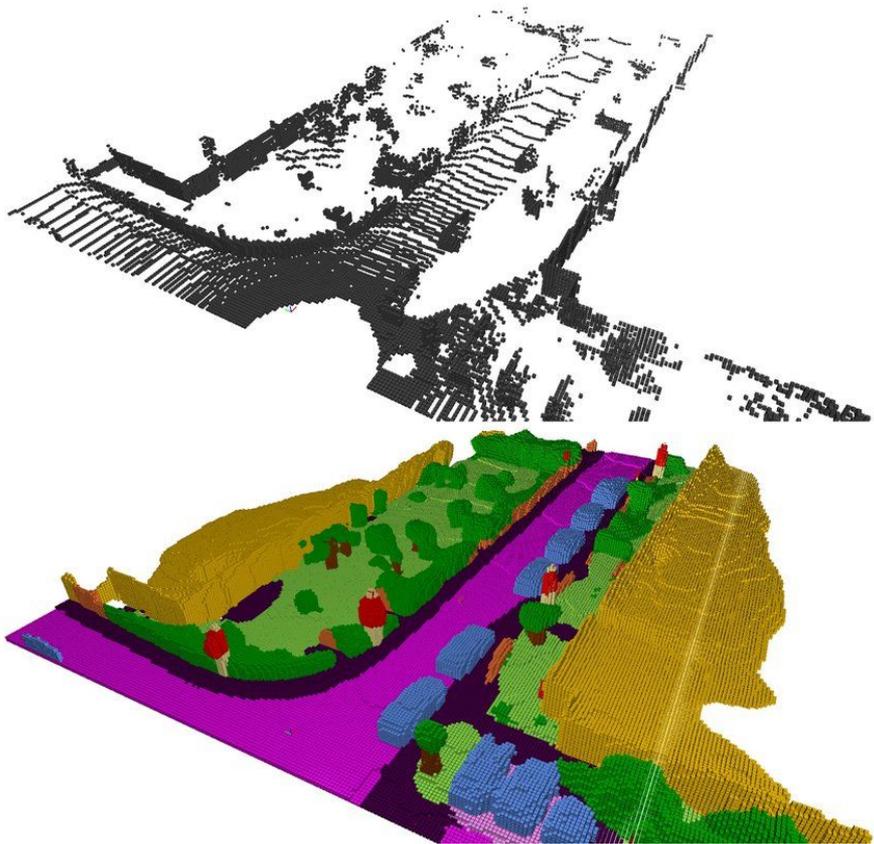


(a) Breakdown of trainable parameters



# Overview

- I. Convolutions on points
  - A. Formulation
  - B. Sampling
- II. Voxels
- III. Mixers and Transformers
  - A. Mixers
  - B. Transformers
- IV. Conclusion



# Conclusion

## Efficient architectures

- MinkUNet (for everything)
- PTv3 (flexible, sometimes hard to train)
- WaffleIron (outdoor lidar)

## Practical sessions

- WaffleIron for part segmentation