FKAConv: Kernel-Feature Alignment for Point Cloud Convolution

Alexandre Boulch, Gilles Puy and Renaud Marlet

ACCV 2020
Introduction

Image processing

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

CNNs for image processing
Introduction

Image processing

Point cloud processing

CNNs for image processing

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

Image processing

Point cloud processing

CNNs for image processing

Similar formulation taking into account point cloud specificities

CNNs for point cloud processing
From discrete convolution to convolution for point clouds

Convolution for image processing

\[ h[n] = \sum_{f \in \{1, \ldots, F\}} K_f^\top \begin{pmatrix} f_f(n) \end{pmatrix} \]

Kernel space Feature space

Features

Kernel
From discrete convolution to convolution for point clouds

Convolution for image processing

\[ h[n] = \sum_{f \in \{1, \ldots, F\}} \mathbf{K}_f^\top \mathbf{I} \mathbf{f}_f(n) \]

Identity matrix: one to one alignment matrix
From discrete convolution to convolution for point clouds

Toward convolution for point clouds

\[ h[n] = \sum_{f \in \{1, \ldots, F\}} K_f^\top \begin{bmatrix} I \\ f_f(n) \end{bmatrix} \]

Kernel space  Feature space

Permutation of two points in the input ⇒ different result
From discrete convolution to convolution for point clouds

Toward convolution for point clouds

$$h[n] = \sum_{f \in \{1, \ldots, F\}} K^T_f \times \mathcal{F}_f(n)$$

Identity matrix: ⇒ no invariance to permutation

Permutation of two points in the input ⇒ different result
Toward convolution for point clouds

\[ h[n] = \sum_{f \in \{1, \ldots, F\}} \mathbf{K}^\top_f \mathbf{f}_f(n) \]

Kernel space \quad Feature space

**Need for a more general matrix**
Alignment matrix prediction

Toward convolution for point clouds

\[ h[n] = \sum_{f \in \{1, \ldots, F\}} K_f^T A f_f(n) \]

Kernel space
Feature space

Alignment matrix \( A \)
Invariance to permutation
⇒ estimated based on inputs
Alignment matrix prediction

Toward convolution for point clouds

$$h[n] = \sum_{f \in \{1, \ldots, F\}} K_f^T A f_f(n)$$

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

**Alignment matrix** $A$
Invariance to permutation
⇒ estimated based in inputs

**KPConv [47]**
- Kernel elements on geodesic ball
- $A$ computed according to distances from input to kernel

**ConvPoint [3]**
- Kernel elements randomly initialized and optimized
- $A$ 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

$$A = \phi(p_i, \{p_i\})$$

- 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)
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

**Input**

**On all points:** space quantization and unique per quantile point selection

**objective:** 20 support points

9 support points selected (●)
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 reached

**Input**

On all points: space quantization and unique per quantile point selection

Cardinality not reached

9 support points selected

14 new support points selected

Cardinality exceeded
random discard of 3 points from last selection step

**Outputs**

Support points

**Objective:** 20 support points
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

\[ v = \frac{\text{diag}}{\sqrt{|Q|}} \]

Number of support points

Diagonal of bounding box

Model based on a simple case (planar surface) and validated on experimental data
Network
Experimental results: S3DIS

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Search</th>
<th>IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointnet [31]</td>
<td>Knn</td>
<td>47.6</td>
</tr>
<tr>
<td>RSNet [17]</td>
<td>-</td>
<td>56.5</td>
</tr>
<tr>
<td>PCCN [48]</td>
<td>-</td>
<td>58.3</td>
</tr>
<tr>
<td>SPGraph [20]</td>
<td>Super pt.</td>
<td>62.1</td>
</tr>
<tr>
<td>PointCNN [23]</td>
<td>Knn</td>
<td>65.4</td>
</tr>
<tr>
<td>PointWeb [56]</td>
<td>Knn</td>
<td>66.7</td>
</tr>
<tr>
<td>ShellNet [55]</td>
<td>Knn</td>
<td>66.8</td>
</tr>
<tr>
<td>ConvPoint [3]</td>
<td>Knn</td>
<td>68.2</td>
</tr>
<tr>
<td>KPConv [45]</td>
<td>Radius</td>
<td>70.6</td>
</tr>
<tr>
<td>FKACConv (Ours fusion)</td>
<td>Knn</td>
<td>68.4</td>
</tr>
<tr>
<td>Rank</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

References as in the paper.
Experimental results: Semantic8

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Av.</th>
<th>OA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IoU</td>
<td></td>
</tr>
<tr>
<td>TML-PC [30]</td>
<td>39.1</td>
<td>74.5</td>
</tr>
<tr>
<td>TMLC-MS [15]</td>
<td>49.4</td>
<td>85.0</td>
</tr>
<tr>
<td>PointNet++ [33]</td>
<td>63.1</td>
<td>85.7</td>
</tr>
<tr>
<td>EdgeConv [8]</td>
<td>64.4</td>
<td>89.6</td>
</tr>
<tr>
<td>Snapchat [4]</td>
<td>67.4</td>
<td>91.0</td>
</tr>
<tr>
<td>PointGAN [28]</td>
<td>69.5</td>
<td>92.1</td>
</tr>
<tr>
<td>FPCR [46]</td>
<td>72.0</td>
<td>90.6</td>
</tr>
<tr>
<td>SPGraph [20]</td>
<td>76.2</td>
<td>92.9</td>
</tr>
<tr>
<td>ConvPoint [3]</td>
<td>76.5</td>
<td>93.4</td>
</tr>
<tr>
<td>FKACConv* (ours fusion)</td>
<td>74.6</td>
<td>94.1</td>
</tr>
<tr>
<td>Rank</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

References as in the paper.
Experimental results: NPM3D

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Av.IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF MSSF [44]</td>
<td>56.3</td>
</tr>
<tr>
<td>MS3 DVS [37]</td>
<td>66.9</td>
</tr>
<tr>
<td>HDGCN [25]</td>
<td>68.3</td>
</tr>
<tr>
<td>ConvPoint [3]</td>
<td>75.9</td>
</tr>
<tr>
<td>KPConv [45]</td>
<td>82.0</td>
</tr>
<tr>
<td>FKACConv (ours fusion)</td>
<td>82.7</td>
</tr>
<tr>
<td>Rank</td>
<td>1</td>
</tr>
</tbody>
</table>

References as in the paper.
Conclusion

FKACConv: 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 and efficient point-cloud sampling

Code available at

https://github.com/valeoai/FKACConv

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

https://github.com/valeoai/LightConvPoint