Survey of 3D Deep Learning (Robo+R3 Study Group)

Arithmer　R3 Div.　Takashi Nakano

2019/09/12

Self-Introduction

- Takashi Nakano
  - Graduate School
    - Kyoto University
    - Laboratory : Nuclear Theory Group
    - Research : Theoretical Physics, Ph.D. (Science)
      - Phase structure of the universe
      - Theoretical properties of Lattice QCD
      - Phase structure of graphene
  - Former Job
    - KOZO KEIKAKU ENGINEERING Inc.
      - Contract analysis / Technical support / Introduction support by using software of Fluid Dynamics / Powder engineering
  - Current Job
    - Application of machine learning / deep learning to fluid dynamics
      - e.g. https://arithmer.co.jp/2019-12-29-1/
    - Application of machine learning / deep learning to 3D data

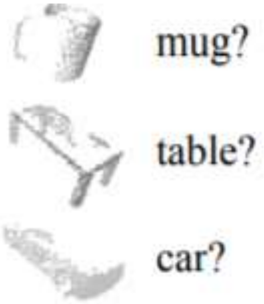
Purpose

- Purpose of this material
  - Overview of 3D deep learning
  - Comparison b/w each method of 3D deep learning
  - Main papers (In this material, I have summarized the material based on following materials and cited papers therein.)
    - E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018
    - M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016

- **Application of 3D Deep Learning**

Classification

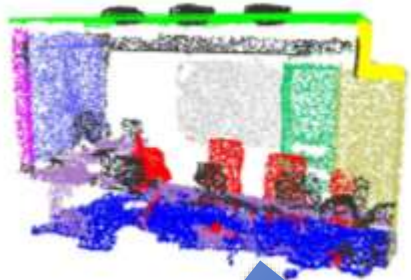
[1]



Segmentation

[1]

[1]



Each label
at each vertex

Correspondence

[2]

same #vertex
at each model

Per-point classification

Retrieval

[1]

Comparison of Global Feature

3D data restoration from 2D images,
Pose Estimation, etc.

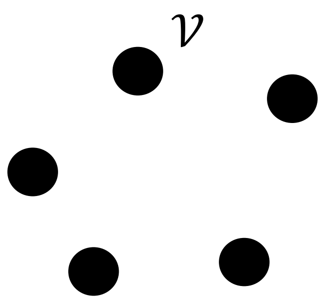
[1] C. R. Qi, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", 2016
[2] J. Masci et al. "Geodesic convolutional neural networks on Riemannian manifolds", 2015

Agenda

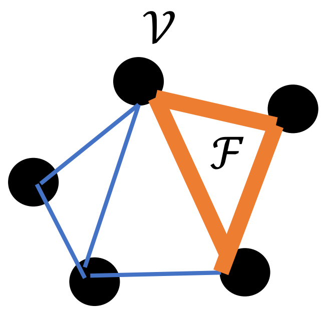
- Methods of 3D Deep Learning
  - Euclidean vs Non-Euclidean
  - Euclidean Method
    - Projections / Multi-View
    - Voxel
  - Non-Euclidean Method
    - Point Cloud / Mesh / Graph
  - Accuracy
  - Dataset / Material
  - Appendix
    - Mesh Generation
    - Laplacian on Graph
    - Correspondence

- **3D Data**

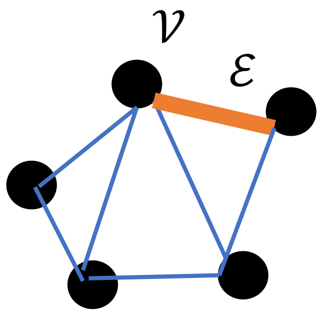
Point Cloud



Mesh



Graph



	Point Cloud	Mesh	Graph
Vertex	○	○	○
Face	-	○	-
Edge	-	-	○

$[[x_0, y_0, z_0], \dots, [x_N, y_N, z_N]]$

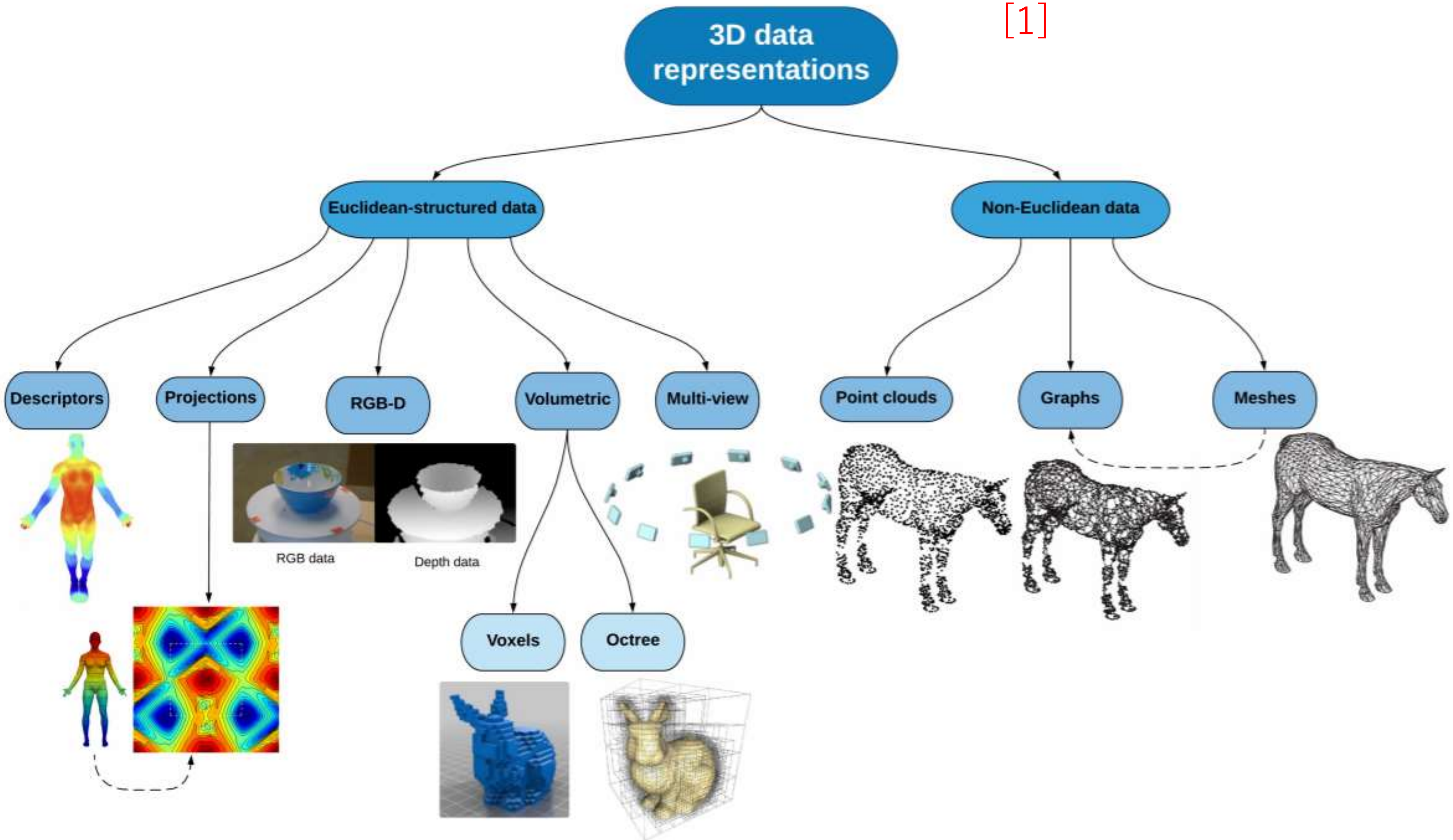
$[[V_{00}, V_{01}, V_{02}], \dots, [V_{F0}, V_{F1}, V_{F2}]]$

$[[V_{00}, V_{01}], [V_{01}, V_{02}], \dots, [V_{E2}, V_{E1}]]$

Representation

- Representation of 3D data

[1]



[1] E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018

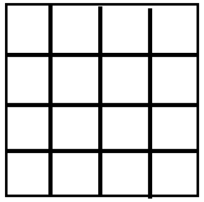
Representation

- **Representation of 3D data**

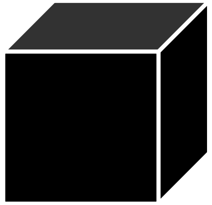
Euclidean

Grid
(Translational invariant)

2D



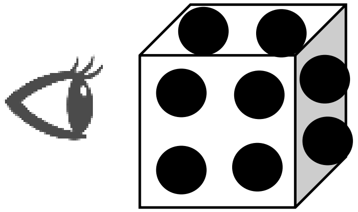
3D



2D CNN → 3D CNN

Global / Extrinsic

Point of view from 3D

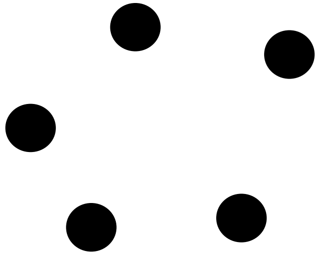


Rigid
Small deformation

[1]

Non-Euclidean

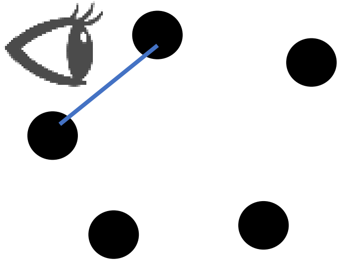
Non-Grid
(not Translational invariant)



→ Non-trivial CNN

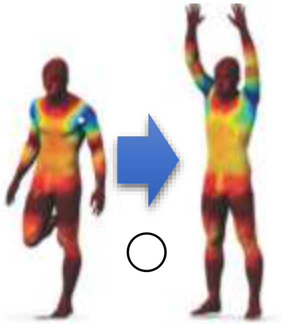
Local / Intrinsic

Point of view from 2D Surface



Non-Rigid
Large deformation

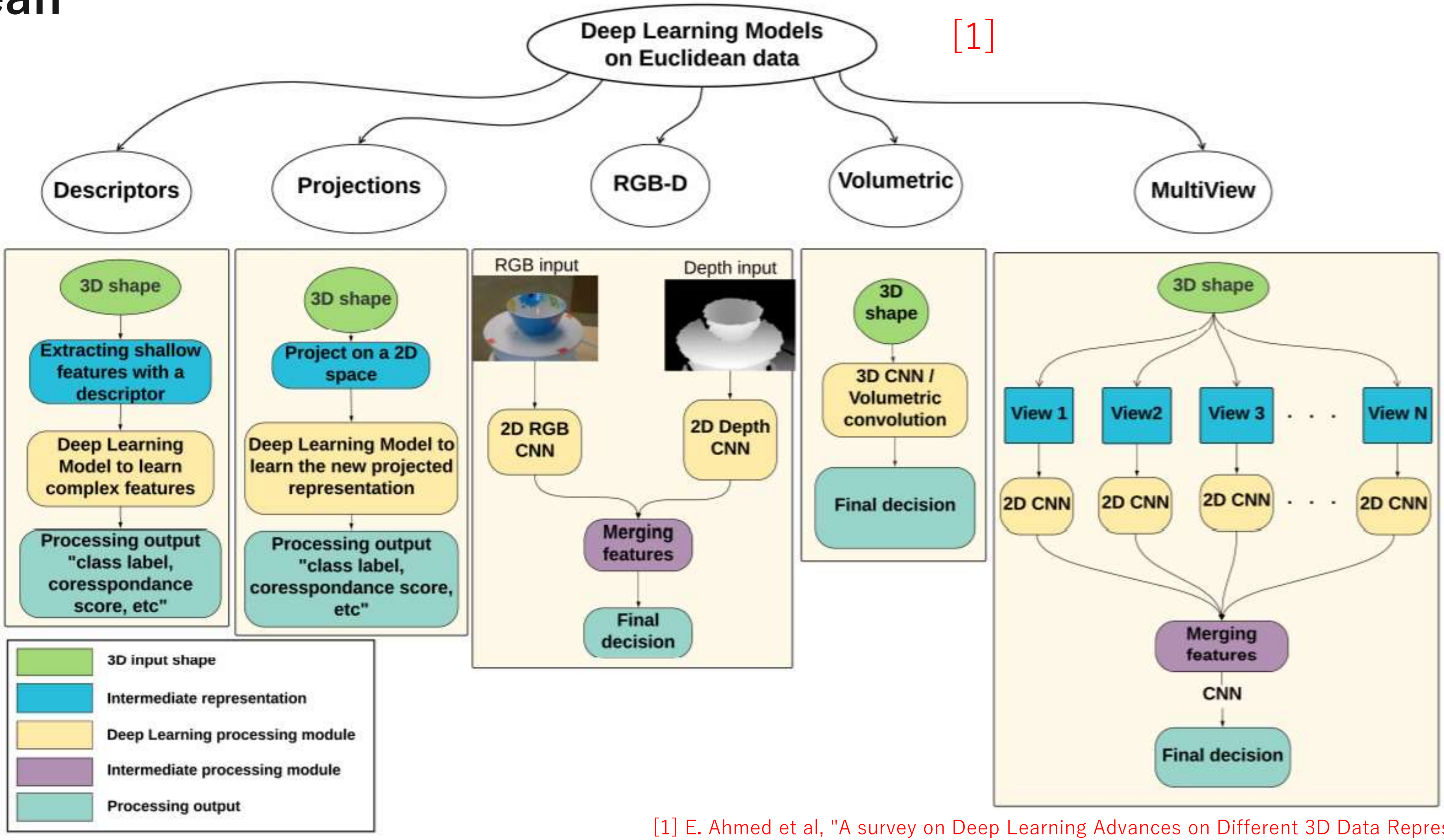
[1]



[1] J. Masci et al. "Geodesic convolutional neural networks on Riemannian manifolds", 2015

Euclidean vs Non-Euclidean

- **Euclidean**



[1] E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018

Euclidean vs Non-Euclidean

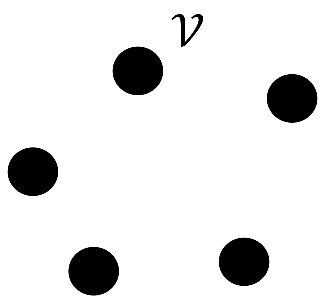
- **Euclidean (detail of feature)**

	Feature	Merit	Demerit
Descriptors	Extraction of 3D topological feature (SHOT, PFH, etc.)	• Can convert as feature • Can each problem	The geometric properties of the shape is lost.
Projections	Projection of 3D to 2D	-	The geometric properties of the shape is lost.
RGB-D	RGB + Depth map	• Can use data from RGB-D sensors (Kinect/realsence) as input	• Need depth map. • Only infer some of the 3D properties based on the depth.
Volumetric	Voxelization	• Expansion of 2D CNN	• Need Large memories. • (grid information) • Need high resolution for detailed shapes. (e.g. segmentation)
Multi-View	2D images from multi-angles	• Highest accuracy in Euclidean method	• Need multi-view images

Euclidean vs Non-Euclidean

- **Non-Euclidean**

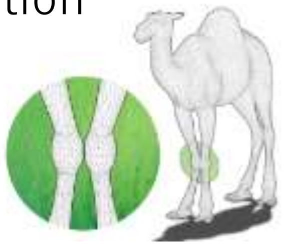
Point Cloud



$$[[x_0, y_0, z_0], [x_1, y_1, z_1]] = [[x_1, y_1, z_1], [x_0, y_0, z_0]]$$

Unordered point cloud

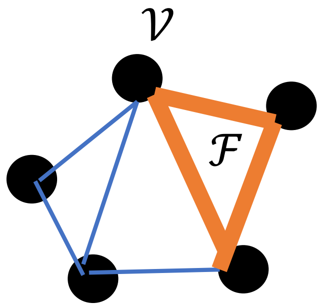
No connected information b/w point cloud



[1]

Dependence of noise and density of point cloud

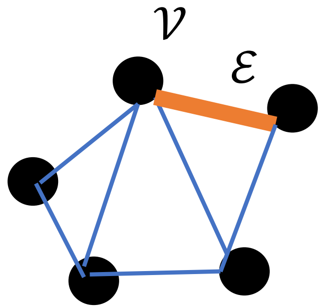
Mesh



Connected information b/w point cloud

Need to convert from point cloud to mesh

Graph



Graph (Vertex, edge)

Need to create graph type

[1] R. Hanocka et al., "MeshCNN: A Network with an Edge", 2018

- **Non-Euclidean (detail of feature)**

	Feature	Merit	Demerit
Point Cloud	• Treat point cloud • Need to keep translational and rotational invariance • Treat unordered point cloud • No connected information b/w point cloud	• Original data is often point cloud. • e.g. scanned data (No CAD data, Terrain data) • Civil engineering, architecture, medical care, fashion	• Treat noise • Dependence of density of point cloud • Complement b/w point cloud • Cannot distinguish b/w close point cloud
Mesh	• Treat mesh data • Connected information b/w point cloud • Convert mesh data to structure for applying CNN	• CAD data • e.g. design in manufacturing • Can keep geometry in few mesh	• Convert point cloud to mesh data
Graph	• Treat mesh as graph • Vertex (node) • Edge (connected information b/w point cloud)	• Same as Mesh	• Create graph type CNN (non-trivial)

Euclidean vs Non-Euclidean

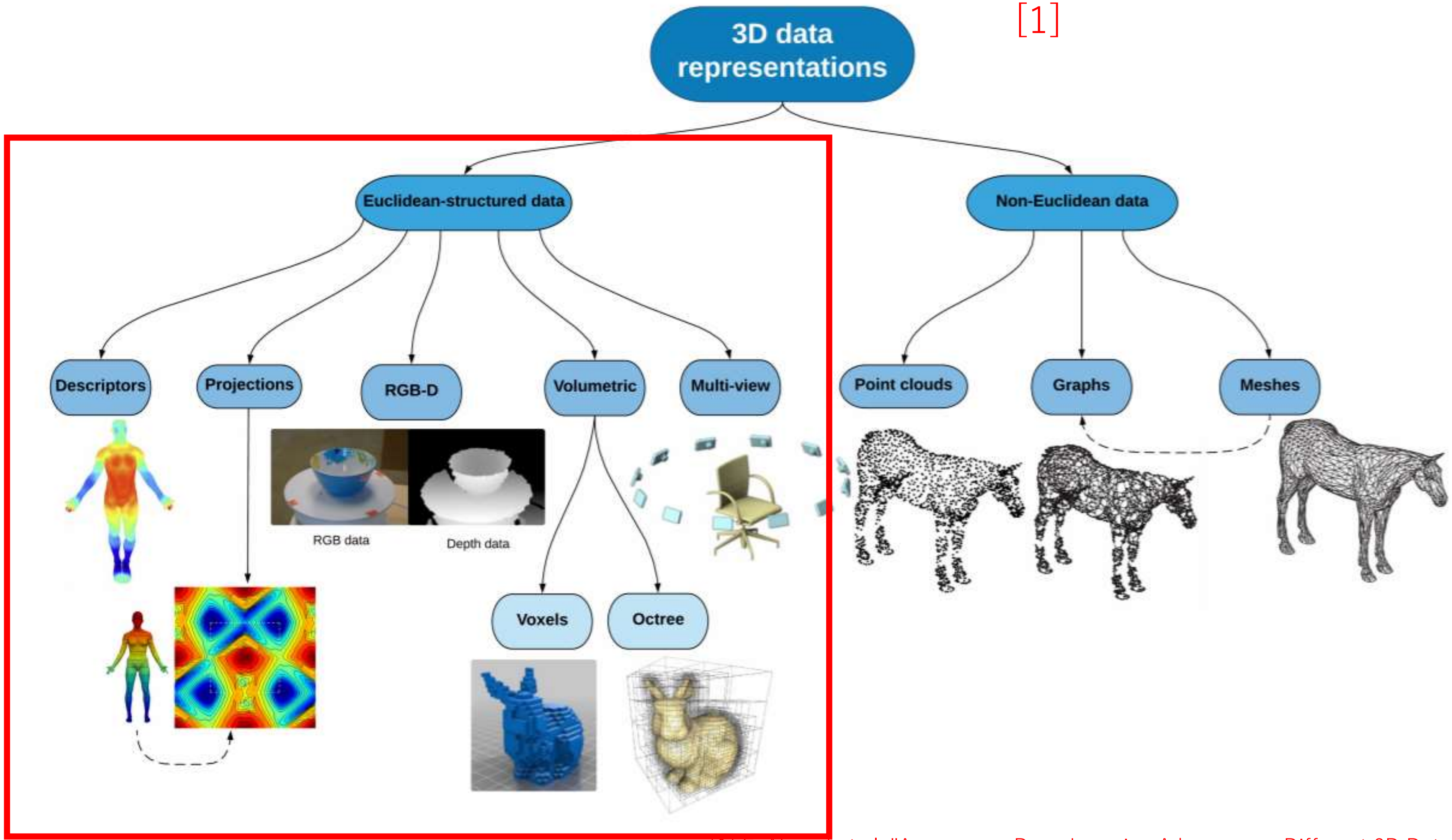
- **Non-Euclidean (detail of feature)**

	Feature	Merit	Demerit
Point Cloud	• Treat point cloud • Need to keep translational and rotational invariance • Treat unordered point cloud • No connected information b/w point cloud	• Original data is often point cloud. • e.g. scanned data (No CAD data, Terrain data) • Civil engineering, architecture, medical care, fashion	• Treat noise • Dependence of density of point cloud • Complement b/w point cloud • Cannot distinguish b/w close point cloud
Mesh	• Treat mesh data • Connected information b/w point cloud • Convert mesh data to structure for applying CNN	• CAD data • e.g. design in manufacturing • Can keep geometry in few mesh	• Convert point cloud to mesh data
Graph	• Treat mesh as graph • Vertex (node) • Edge (connected information b/w point cloud)	• Same as Mesh	• Create graph type CNN (non-trivial)

Euclidean

- Representation of 3D data

[1]



[1] E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018

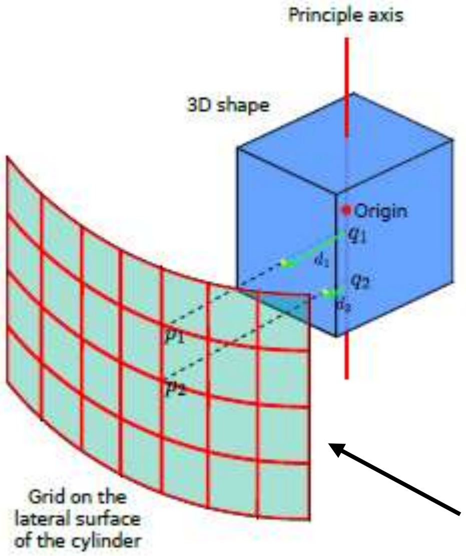
- **Each Euclidean Method (Projections / RGB-D / Volumetric / Multi-View)**

Method	Application	Link
Deep Pano	Classification	Paper
Two-stream CNNs on RGB-D	Classification	Paper
VoxNet	Classification	Paper GitHub(Keras)
MVCNN	Classification Retrieval	Paper GitHub(PyTorch/TensorFlow etc.)

Euclidean

- Deep Pano [1]
  - Projection to Panoramic image
  - Row-wise max-pooling for rotational invariant

[1]



Panoramic image

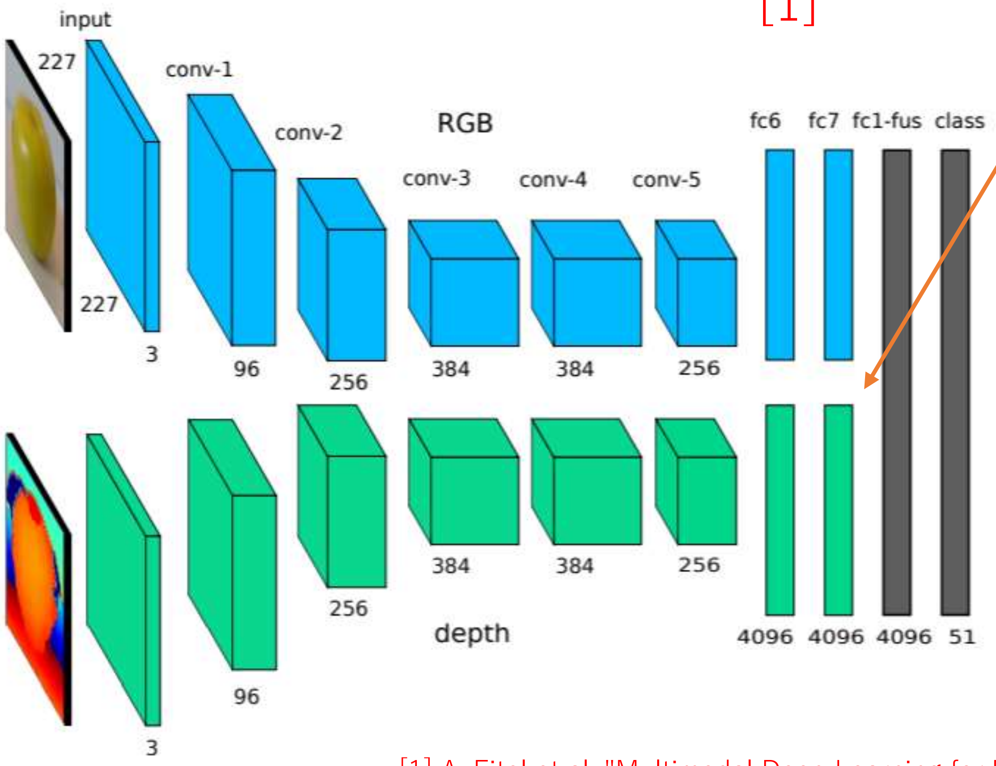
[1] B. Shi et al. "DeepPano: Deep Panoramic Representation for 3D Shape Recognition", 2017

Euclidean

- Two-stream CNNs on RGB-D [1]
  - Concatenate CNN of RGB and CNN of depth map

[1]

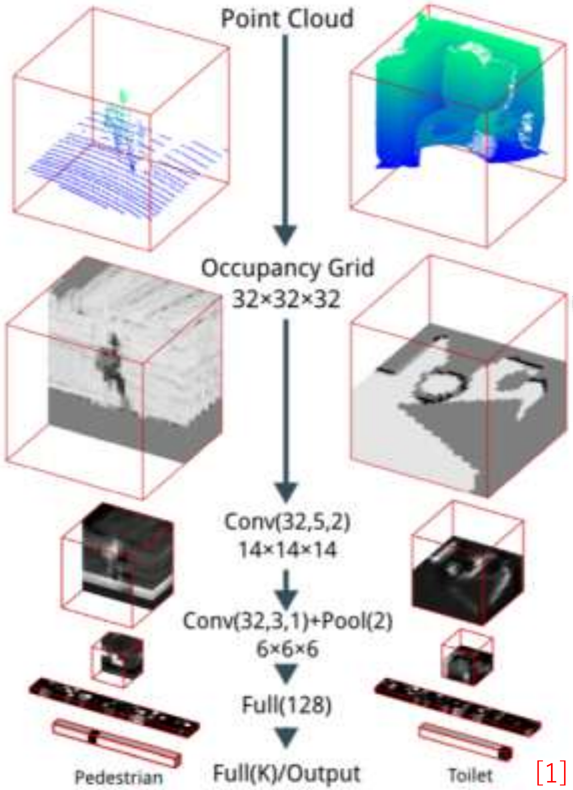
Concatenation

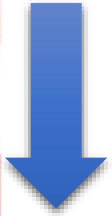


[1] A. Eitel et al. "Multimodal Deep Learning for Robust RGB-D Object Recognition", 2015

- VoxNet [1]
  - Voxelization of 3D point cloud to voxel
  - Not robust for data loss

[1]

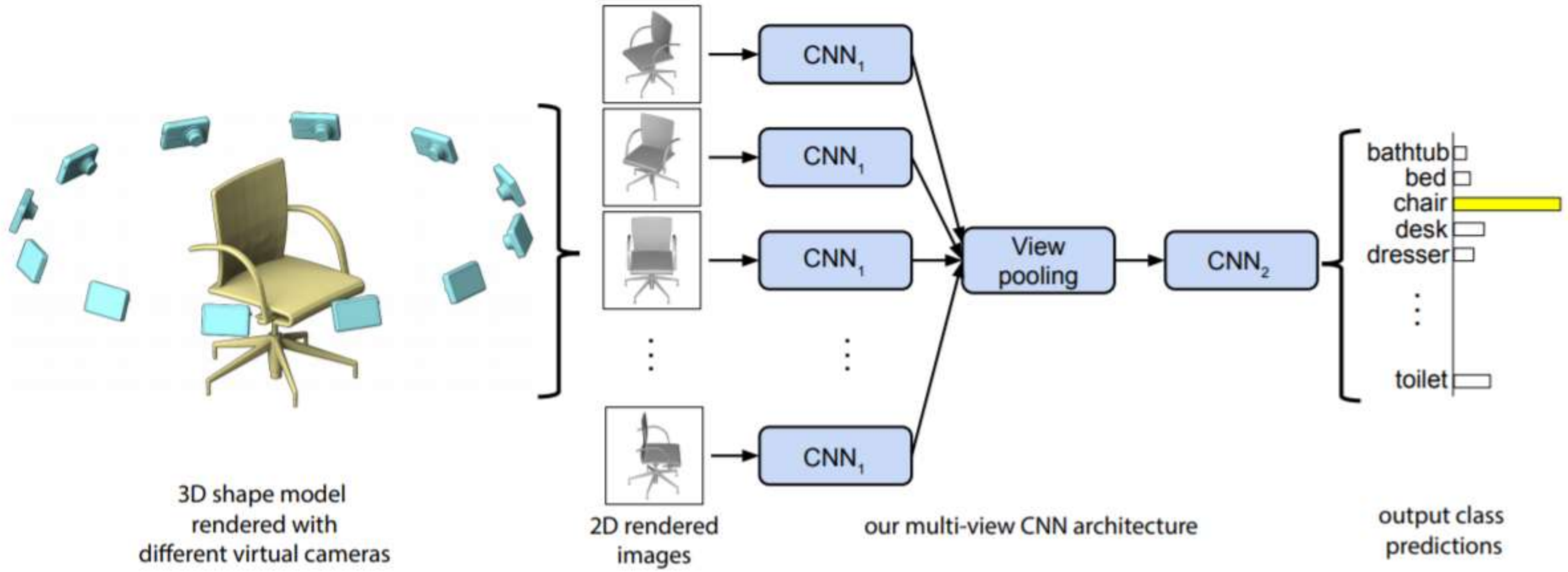




[1] D. Maturana et al. "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition", 2015

- MVCNN [1]
  - Merge CNN of each images

[1]

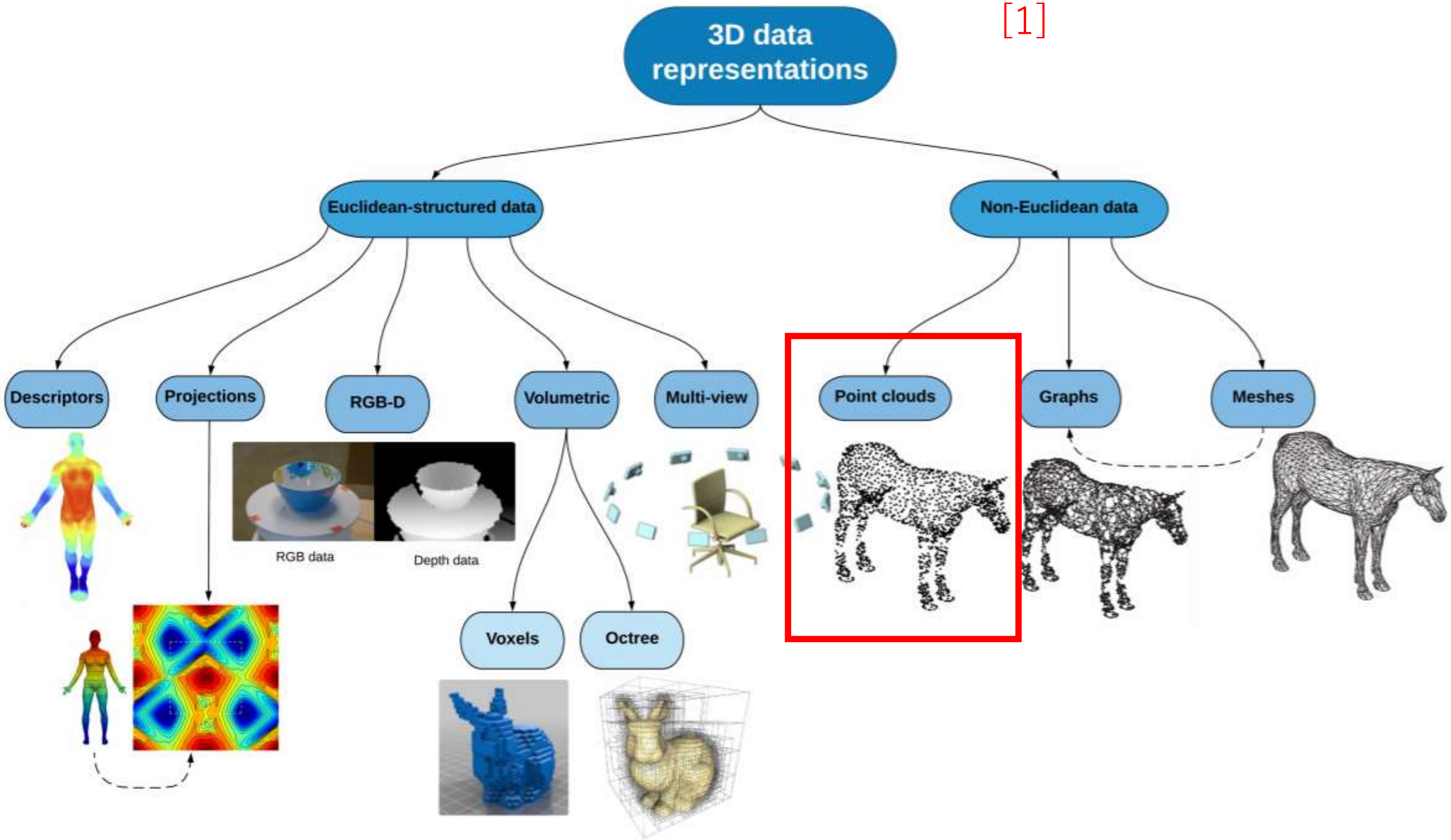


[1] H. Su et al. "Multi-view Convolutional Neural Networks for 3D Shape Recognition", 2015

Non-Euclidean (Point Clouds)

- Representation of 3D data

[1]



[1] E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018

- **Each Non-Euclidean Method (Point Cloud)**

Method	Application	Link
PointNet	Classification Segmentation Retrieval Correspondence	Paper GitHub (TensorFlow)
PointNet++	Classification Segmentation Retrieval Correspondence	Paper GitHub (TensorFlow) PyTorch-geometric (PointConv)
Dynamic Graph CNN (DGCNN)	Classification Segmentation	Paper GitHub (PyTorch/TensorFlow) PyTorch-geometric (DynamicEdgeConv)
PointCNN	Classification Segmentation	Paper GitHub (TensorFlow) PyTorch-geometric (XConv)

※Some equations from following pages are referred to the documents in PyTorch-geometric.
(https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html)
I will explain PyTorch-geometric in later page.

Non-Euclidean (Point Clouds)

- PointNet [1]
  - Treat unordered point cloud by max-pooling
  - Comparison b/w PointNet++
    - Detailed information is lost
    - Cannot treat different density of point cloud

MLP

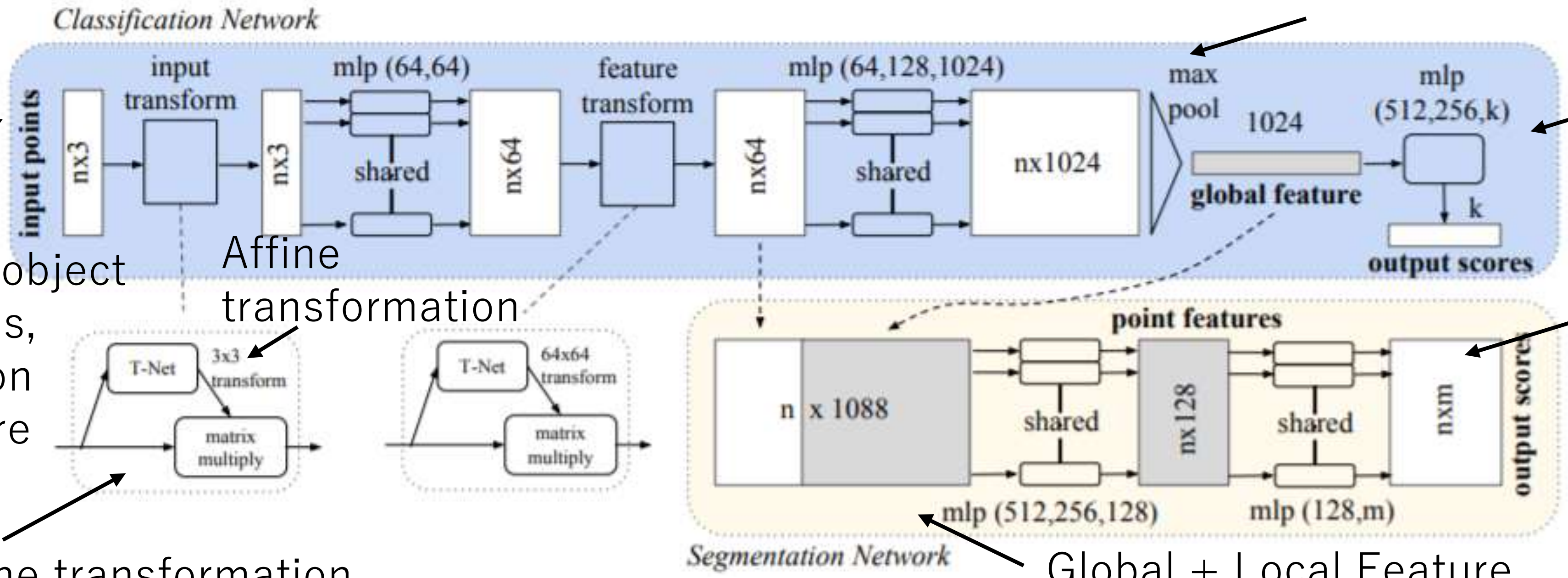
$$f(\{x_1,\cdots,x_n\}) = g(h(x_1),\cdots,h(x_n))$$

Input feature

Max-pooling

Symmetry Function

[1]



Classification

Randomly rotating the object along up-axis, Normalization in unit square

Affine transformation

Part segmentation (Per-point classification)

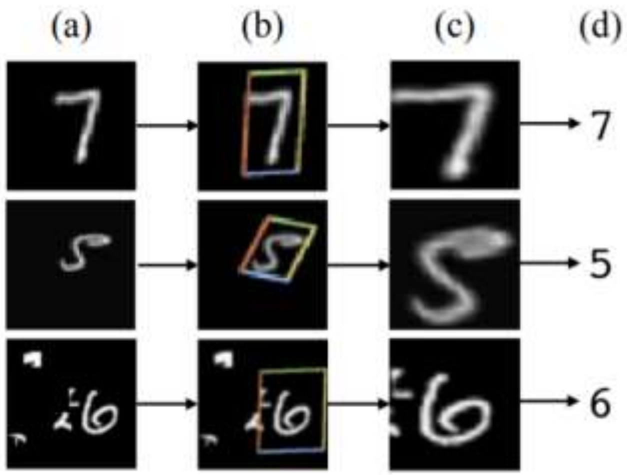
Predict Affine transformation (Transrational, Rotational Invariance) Similar to Spatial Transformer Networks in 2D

Global + Local Feature

[1] C. R. Qi, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", 2016

Non-Euclidean (Point Clouds)

- PointNet
  - T-Net [1]
    - Similar to Spatial Transformer Networks in 2D
    - Spatial Transformer Networks
      - Alignment of image (transformation, rotation, distortion etc.) by spatial transformation
      - Learn affine transformation from input data (not necessarily special data)
      - Can insert this networks at each point b/w networks



[1]

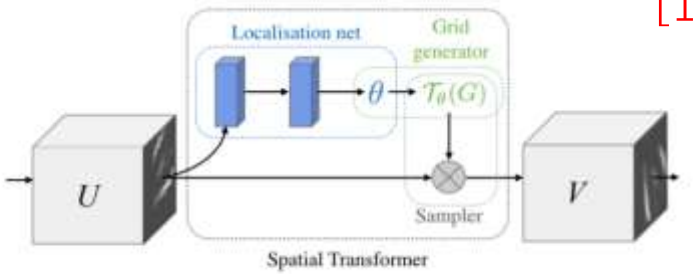
Reference	Contents
Paper	Original Paper
Sample (PyTorch)	Dataset : MNIST

[1] M. Jaderberg et al. "Spatial transformer networks",2015

- PointNet
  - Spatial Transformer Networks
    - Localization net : output parameters θ to transform for input feature map U
      - Combination of Conv, MaxPool, ReLU, FC
      - Output : 2×3
    - Grid generator : create sampling grid by using the parameters
    - Sampler : Output transformed feature map V
      - pixel

Spatial Transformer Networks (2D)

[1]



Input feature map

Output feature map

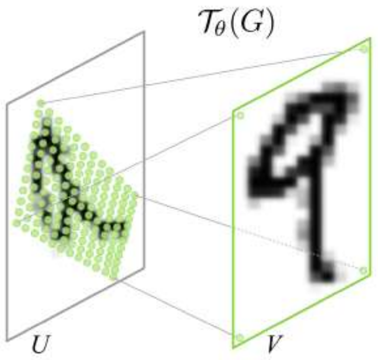
[1]

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = A_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

2×3

Grid generator

[1]



(b)

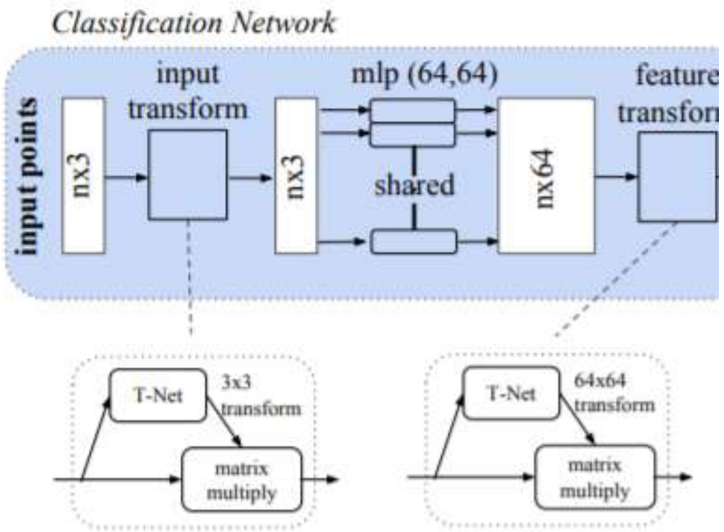
Input map to transformed map

[1] M. Jaderberg et al. "Spatial transformer networks",2015

Non-Euclidean (Point Clouds)

- PointNet
  - T-Net
    - 3D ver. of Spatial Transformer Networks in 2D
    - Not need sampling grid (There are no gird structure in 3D)
      - Directly apply transformation to each point cloud
    - Output parameter
      - 3×3 in first T-Net
      - 64×64 in second T-Net

[1]



T-Net
(input feature : 3)

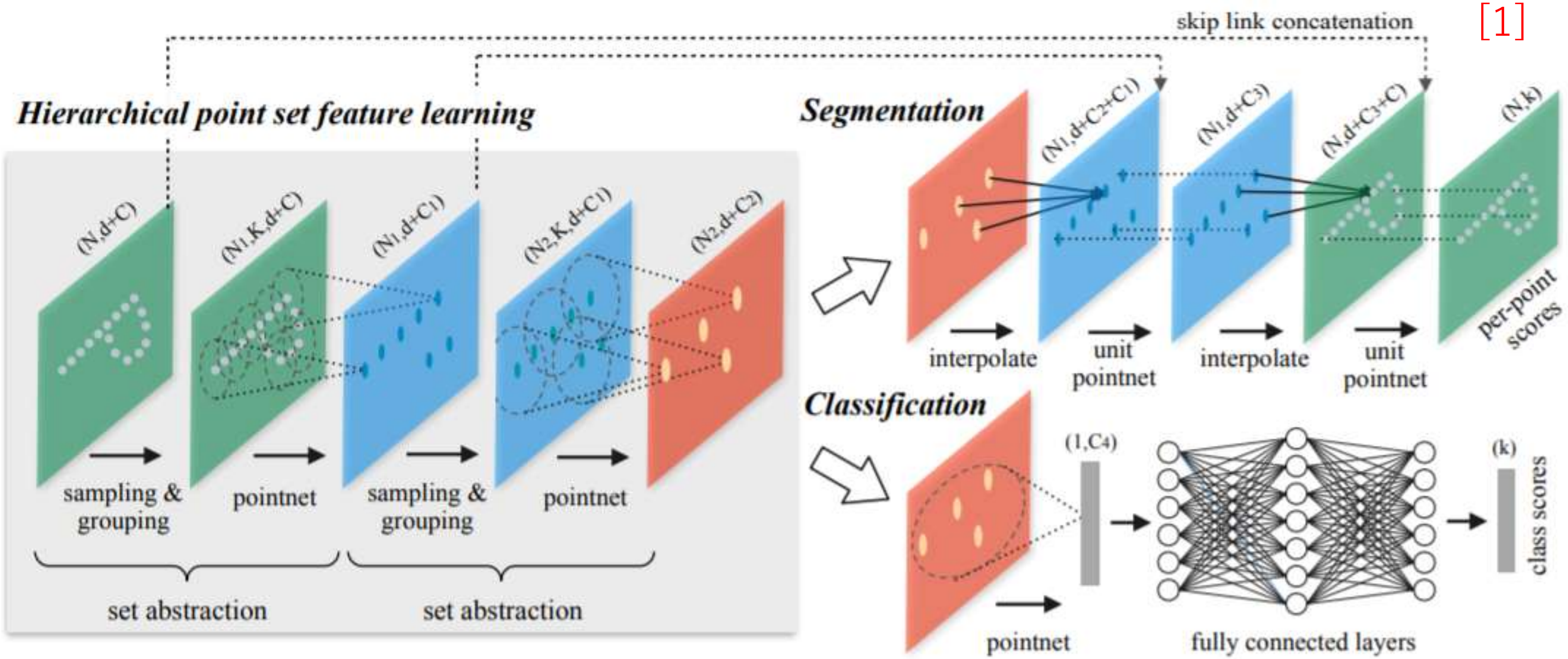
T-Net
(input feature : 64)

[1] C. R. Qi, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", 2016

Non-Euclidean (Point Clouds)

- PointNet++ [1]
  - Comparison b/w PointNet
    - Detailed information is kept
    - Can treat different density of point cloud

[1]

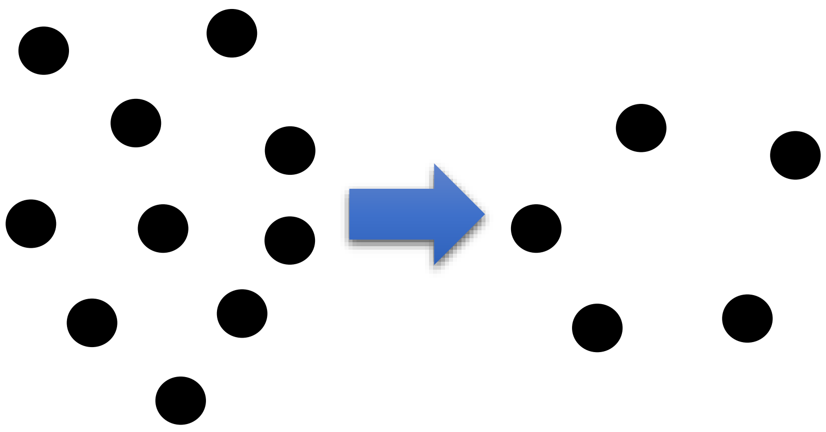


Concatenation of multi-resolution

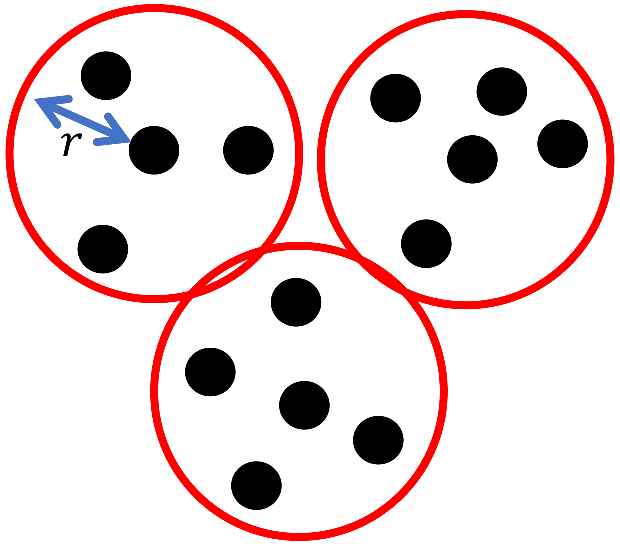
[1] C. R. Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", 2017

- PointNet++
  - Set abstraction
    - Grouping in one scale + feature extraction
      - Sampling Layer : Extraction of sampling points by farthest point sampling (FPS)
      - Grouping Layer : Grouping points around sampling points
      - PointNet Layer : Applying PointNet

Sampling Layer



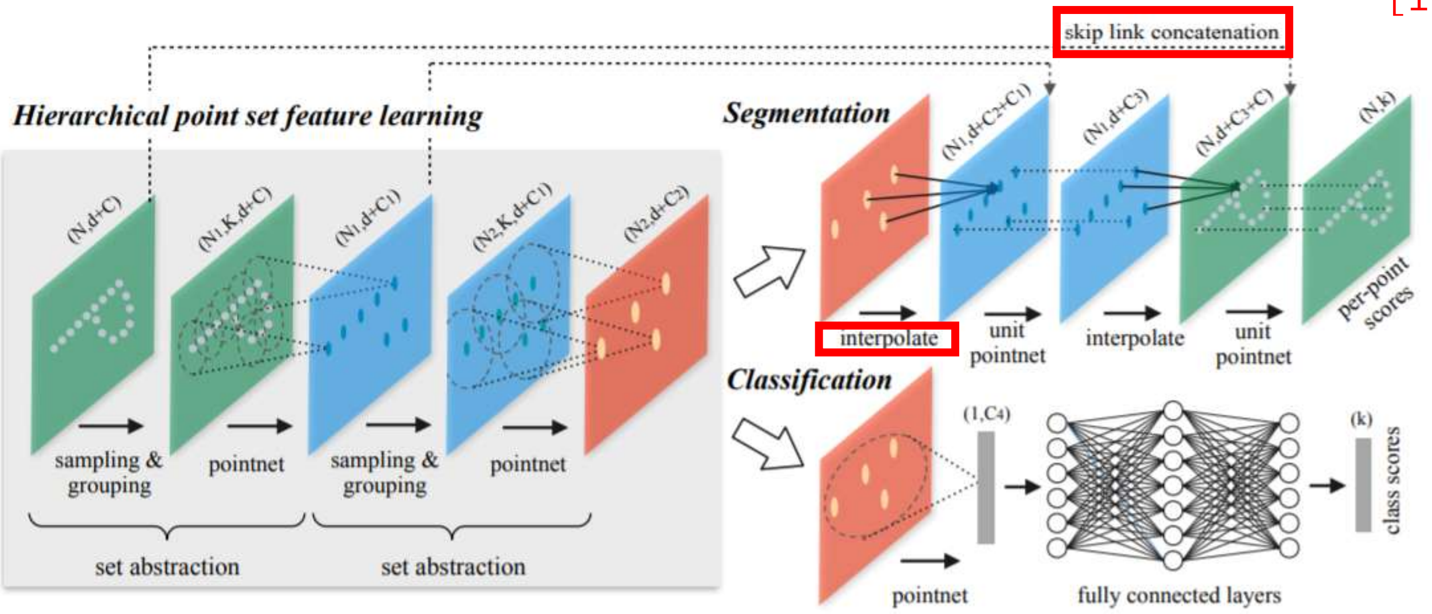
Grouping Layer



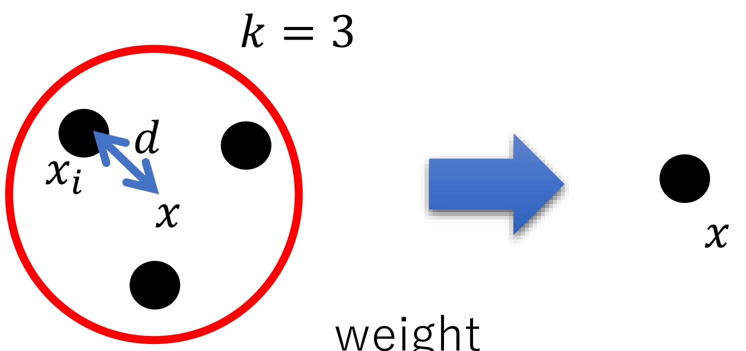
Non-Euclidean (Point Clouds)

- PointNet++
  - Point Feature Propagation for segmentation
    - Interpolation : interpolation from k neighbor points
    - Concatenation

[1]



Interpolation



weight

feature

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \qquad w_i(x) = \frac{1}{d(x, x_i)^2}$$

Inverse of distance

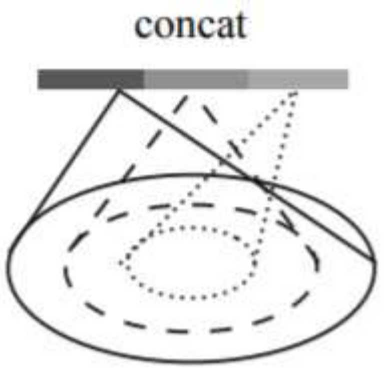
[1] C. R. Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", 2017

Non-Euclidean (Point Clouds)

- PointNet++
  - Single scale grouping
  - Multi scale/resolution grouping
    - Combination of features from different scales
    - Robust for non-uniform sampling density
    - Modifying architecture in set abstraction level

multi-scale grouping (MSG)

[1]

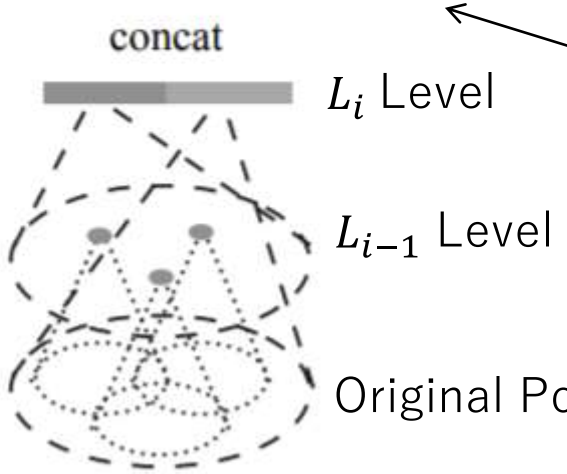


High computational cost

multi-resolution grouping (MRG)

Recommendation

[1]



Concatenation of information of multi-resolution

[1] C. R. Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", 2017

Non-Euclidean (Point Clouds)

- PointNet++
  - **Detail of architecture**
  - **Note: #vertex is fixed**

Set abstraction level $SA(K, r, [\ell_1, \cdots, \ell_d])$

#vertex radius Pointnet (#FC:d)

Global Set abstraction level $SA([\ell_1, \cdots, \ell_d])$

#FC:d

Convert single vector by maxpooling

Fully Connected $FC(\ell, dp)$

Ratio of dropout

Channel

Feature Propagation $FP(\ell_1, \cdots, \ell_d)$

#FC:d

Architecture for classification
and part segmentation of ModelNet
using single scale grouping

$\#input\ vertex: 1024$

$1024/2$ $512/4$

$\boxed{SA(512, 0.2, [64, 64, 128]) \rightarrow SA(128, 0.2, [64, 64, 128]) \rightarrow SA([256,512,1024])}$

Same in cls. and seg.

$\rightarrow FC(512,0.5) \rightarrow FC(256,0.5) \rightarrow FC(K)$

class

For classification

$\rightarrow FP(256, 256) \rightarrow FP(256,128) \rightarrow FP(128,128,128,128, K)$

For part segmentation

per point segmentation

Non-Euclidean (Point Clouds)

- PointNet++
  - Detail of architecture
  - Note: #vertex is fixed

Architecture classification of ModelNet using **multi-resolution grouping (MRG)**

$\#input\ vertex: 1024$

$SA(512, 0.2, [64, 64, 128]) \to SA(64, 0.4, [128, 128, 256])$

Concat.

$SA(512, 0.4, [64, 128, 256])$

$SA([256,512,1024])$

Concat.

$SA([64, 128, 256,512])$

$\to FC(512,0.5) \to FC(256,0.5) \to FC(K)$

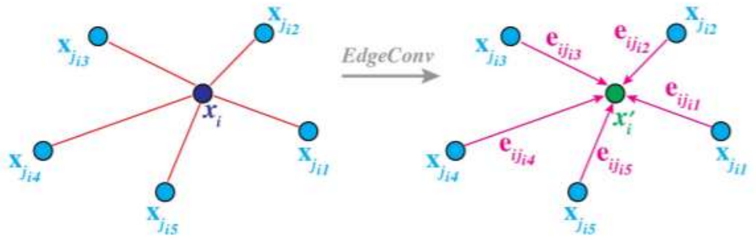
Same as single scale grouping

class

Non-Euclidean (Point Clouds)

- Dynamic Graph CNN (DGCNN) [1]
  - PointNet + w/ Edge Conv.
  - Edge Conv.
    - Create local edge structure dynamically (not fixed in each layer)

Edge Conv. [1]

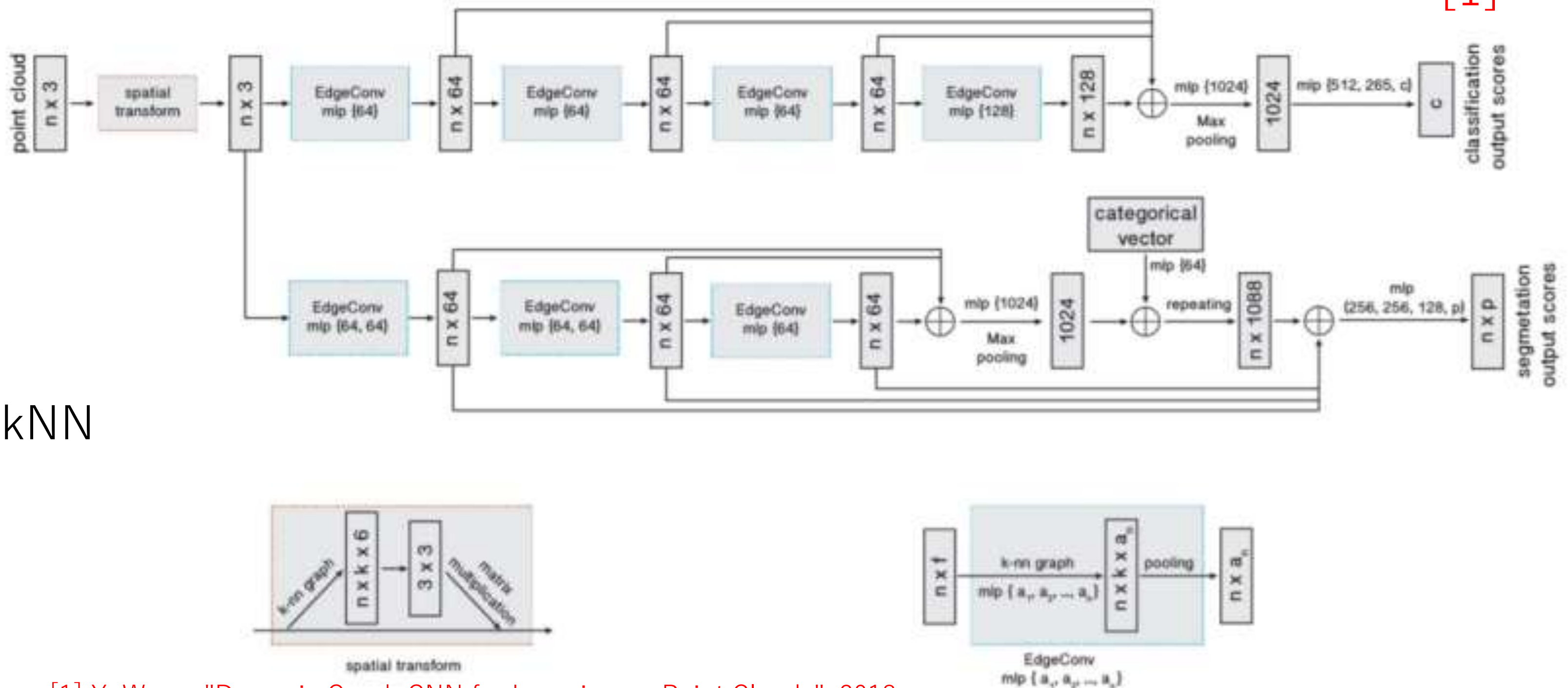


Search neighbors in feature space by kNN

$$x_i' = \sum_{j \in N(i)} h_\Theta(x_i, x_j - x_i)$$

global local

PointNet+ w/ Edge Conv. [1]



[1] Y. Wang, "Dynamic Graph CNN for Learning on Point Clouds", 2018

Non-Euclidean (Point Clouds)

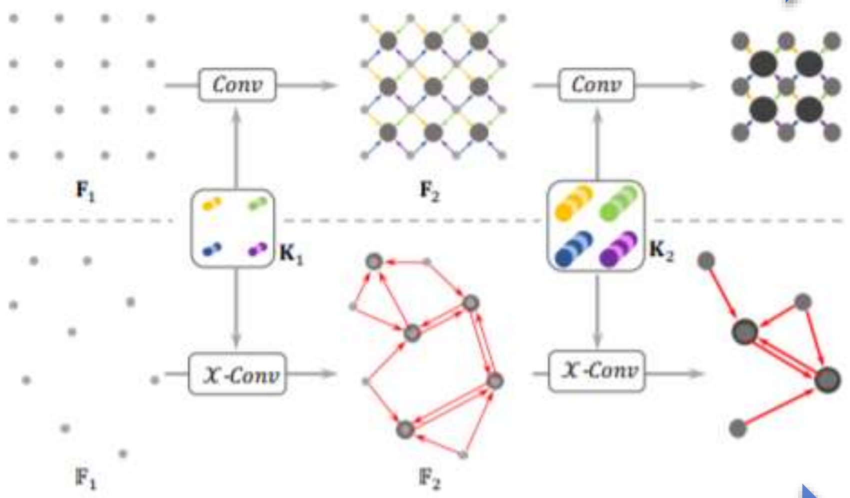
- **PointCNN [1]**
  - **Downsampling information from neighborhoods into fewer representative points**

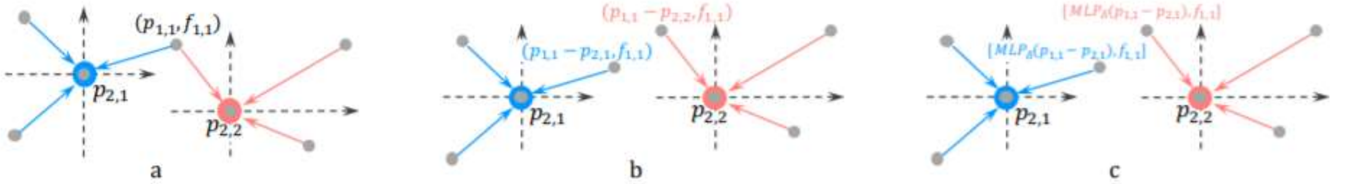
X-Conv.

[1]

Lower resolution, deeper channels

[1]





Input feature

$$x_i' = Conv\left(\boldsymbol{K}, \left[\gamma_\Theta(\boldsymbol{P}_i - \boldsymbol{p}_i) \times [h_\Theta(\boldsymbol{P}_i - \boldsymbol{p}_i), \boldsymbol{x}_i]\right]\right)$$

Kernel

Concatenation

MLP applied individually on each point like PointNet

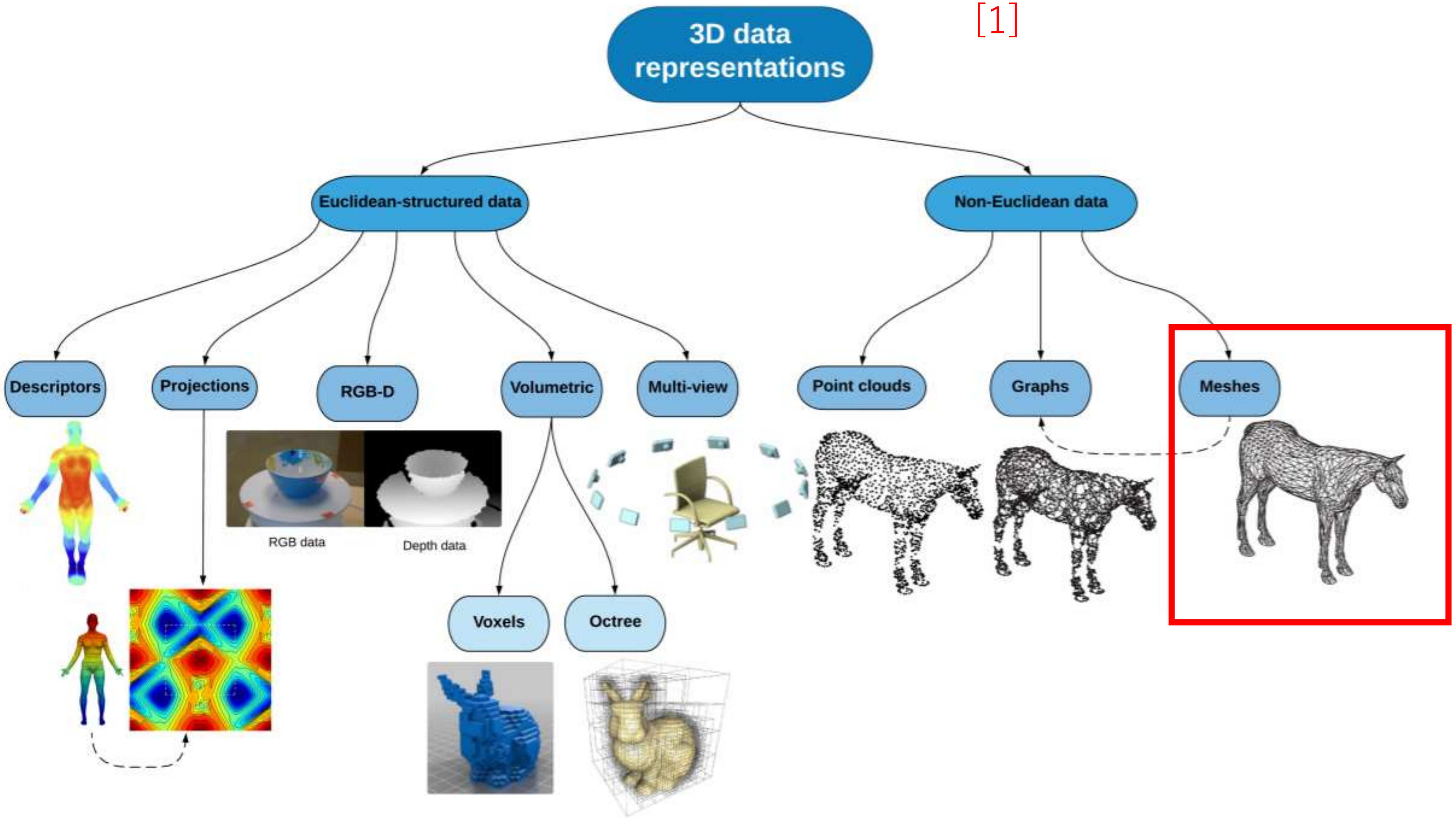
Decreasing #representative points, deeper channels

[1] Y. Li et al. "PointCNN: Convolution On X-Transformed Points", 2018

Non-Euclidean (Mesh)

- Representation of 3D data

[1]



[1] E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018

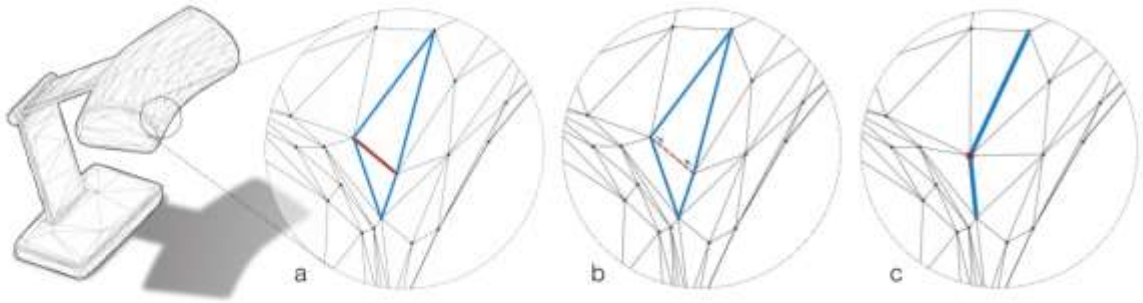
- **Each Non-Euclidean Method (Mesh)**

Method	Application	Link
MeshCNN	Classification Segmentation	Paper GitHub (PyTorch)
MeshNet	Classification	Paper GitHub (PyTorch)

Non-Euclidean (Mesh)

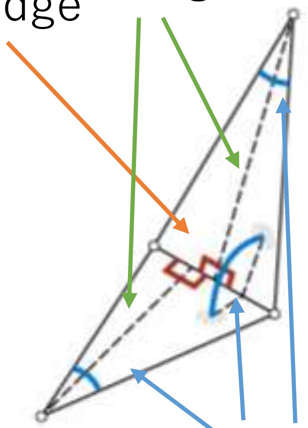
- MeshCNN [1]
  - Edge collapse by pooling
  - Can apply only the manifold mesh

Edge collapse by pooling [1]



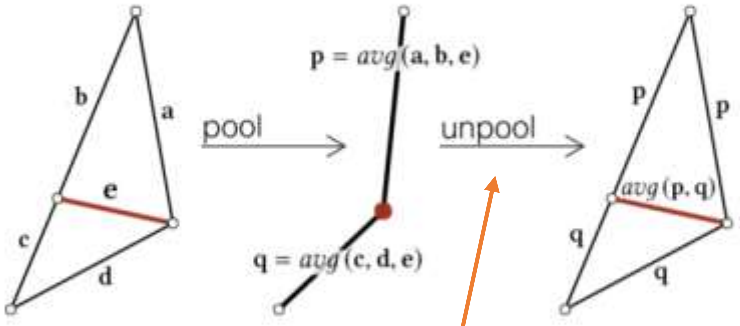
Input feature [1]

Edge
Length



Angle

Pooling / Unpooling [1]

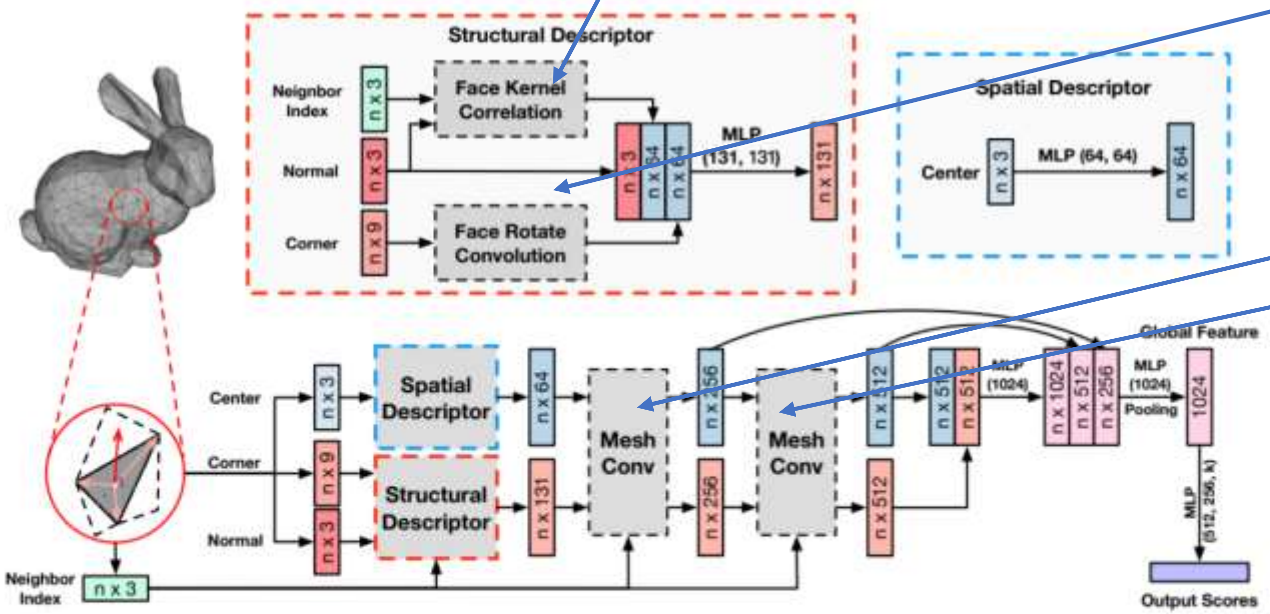


use in segmentation

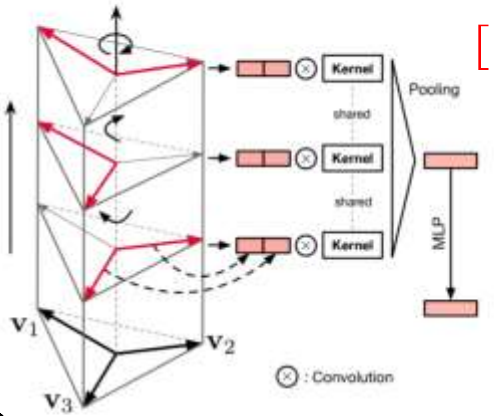
[1] R. Hanocka et al., "MeshCNN: A Network with an Edge", 2018

- MeshNet
  - Input feature
    - Center, corner, normal, neighbor index

Information of neighborhood of face

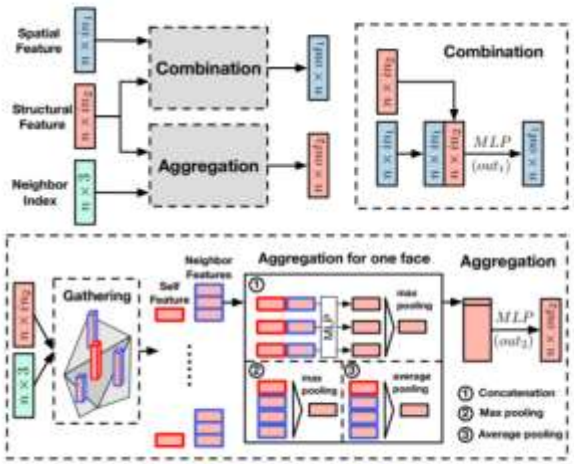


[1]



[1]

Mesh Conv.
(Combination + Aggregation)



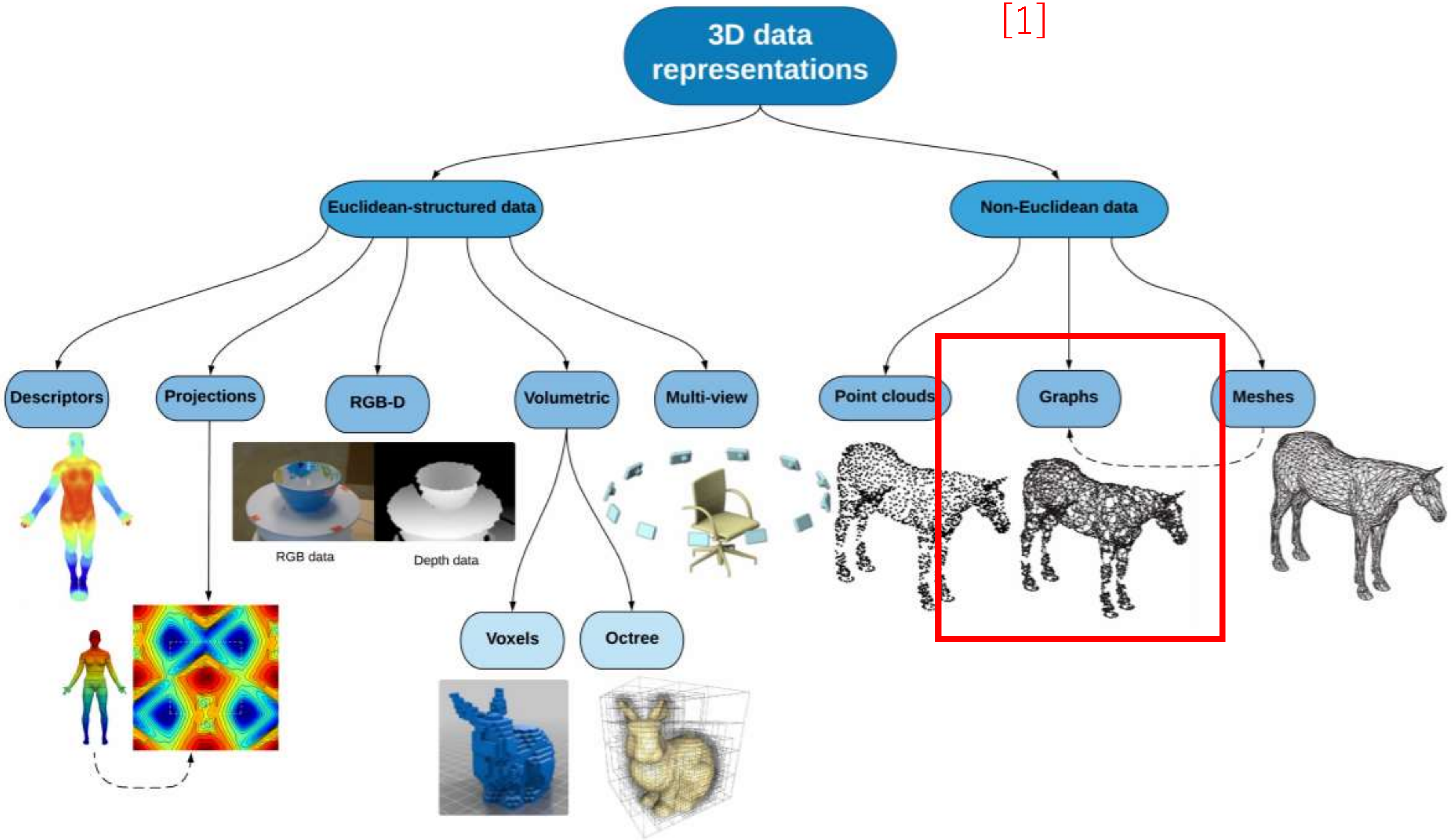
[1]

[1] Y. Feng et al. "MeshNet: Mesh Neural Network for 3D Shape Representation", 2018

Non-Euclidean (Graph)

- Representation of 3D data

[1]

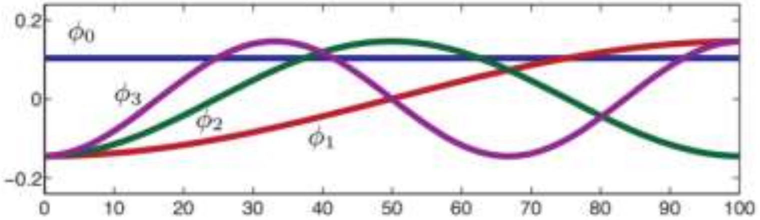


[1] E. Ahmed et al, "A survey on Deep Learning Advances on Different 3D Data Representations", 2018

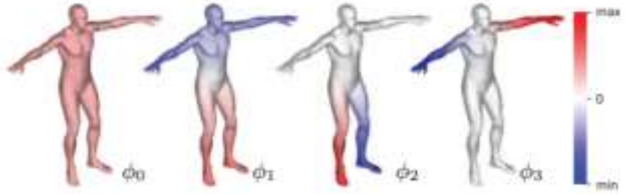
- **Each Non-Euclidean Method (Graph)**
  - **Spectral / Spatial Method**

Spectral

Euclidean (1D) [1]



Non-Euclidean (Manifold) [1]

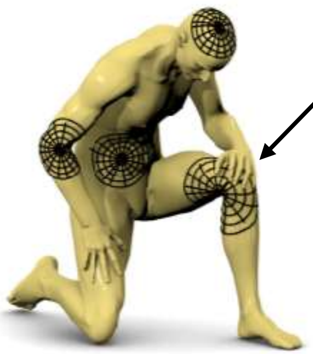


$$\phi_i = e^{i\omega x}, \lambda_i = \omega^2 \xrightarrow{\Delta\phi_i = \lambda_i\phi_i}$$

Generalization of Fourier basis

Spatial

Local coordinate

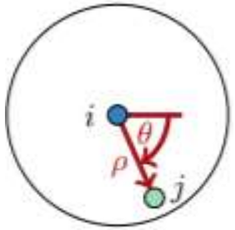


[2]

Convolution

$$(f * g)(x) = \sum_{j=1}^{J} g_j D_j(x) f$$

[3]



$\mathbf{u}(i,j) = (\rho,\theta)$

Patch Operator

$$D_j(x)f = \sum_{y \in N(x)} \omega_j(\mathbf{u}(x,y))f(y)$$

Pseudo-coordinate

[1] M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016
[2] J. Masci et al. "Geodesic convolutional neural networks on Riemannian manifolds", 2015
[3] M. Fey et al. "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels", 2017

Non-Euclidean (Graph)

- **Each Non-Euclidean Method (Graph)**
  - **Spatial method is more useful than spectral method.**

Method	Structure	Feature
Spectral	• Fourier basis in manifold • Laplacian eigenvalue/eigenvector	• Spectral filter coefficients is base dependent in some method • No locality in some method • High computational cost
Spatial	• Create local coordinate • Patch operator + Conv.	• Locality • Efficient computational cost

※Some equations from following pages are referred to the documents in PyTorch-geometric.
(https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html)
I will explain PyTorch-geometric in later page.

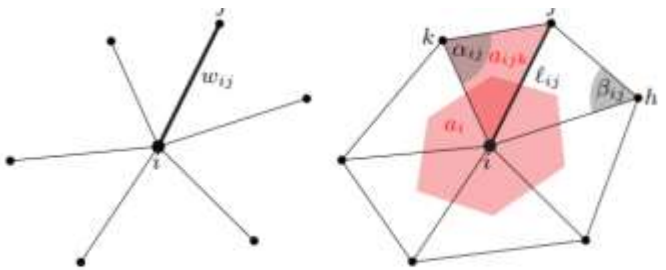
- **Each Non-Euclidean Method (Graph)**
  - **Spectral, Spectral free**

Method	Method	Application	Link
Spectral CNN	Spectral	Graph	Paper
Chebyshev Spectral CNN (ChebNet)	Spectral free	Graph	Paper GitHub (TensorFlow) PyTorch-geometric (ChebConv)
Graph Convolutional Network (GCN)	Spectral free	Graph	Paper PyTorch-geometric (GCNConv)
Graph Neural Network (GNN)	Spectral free	Graph	Paper

Non-Euclidean (Graph)

- Spectral CNN [2]
  - cannot use different shape
    - Spectral filter coefficients is base dependent
  - High computational cost
  - No locality

[1]



Laplacian $(\Delta f)_i \propto \sum_{(i,j)\in\mathcal{E}} \omega_{ij}(f_i - f_j)$

$$\boldsymbol{f}_\ell^{out} = \xi\left(\sum_{\ell'=1}^{p} \boldsymbol{\Phi}_k \widehat{\boldsymbol{G}}_{\ell,\ell'} \boldsymbol{\Phi}_k^T \boldsymbol{f}_{\ell'}^{in}\right)$$

ReLU

Laplacian eigenvector

Different shape
-> different basis -> different result

[1]

Domain	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{Y}$
Basis	$\mathbf{\Phi}$	$\mathbf{\Phi}$	$\mathbf{\Psi}$
Signal	$\mathbf{f}$	$\mathbf{\Phi W \Phi^\top f}$	$\mathbf{\Psi W \Psi^\top f}$

[1] M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016
[2] J. Bruna et al. "Spectral Networks and Locally Connected Networks on Graphs", 2013

Non-Euclidean (Graph)

- Chebyshev Spectral CNN (ChebNet) [1]
  - Not calculate Laplacian eigenvectors directly
  - Locality (K hops)
  - Approximate filter as polynomial
- Graph Convolutional Network (GCN) [2]
  - Special ver. of ChebNet $(K = 2)$

$$X' = \sum_{k=0}^{K-1} Z^{(k)} \cdot \Theta^{(k)}$$

$$Z^{(0)} = X$$

$$Z^{(1)} = \hat{L} \cdot X$$

$$Z^{(k)} = 2 \cdot \hat{L} \cdot Z^{(k-1)} - Z^{(k-2)}$$

$\hat{L}$:scaled and normalized Laplacian

[1] M. Defferrard et al. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering", 2016
[2] T. N. Kipf et al. "Semi-Supervised Classification with Graph Convolutional Networks", 2016

- **Each Non-Euclidean Method (Graph)**
  - **Charting**

	Application	Link
Geodesic CNN	Mesh Shape retrieval / correspondence	Paper
Anisotropic CNN	Mesh / point cloud Shape correspondence	Paper
MoNet	Graph / mesh / point cloud Shape correspondence	Paper PyTorch-geometric (GMMConv)
SplineCNN	Graph / Mesh Classification Shape correspondence	Paper GitHub (PyTorch) PyTorch-geometric (SplineConv)
FeaStNet	Graph / Mesh Shape correspondence Segmentation	Paper PyTorch-geometric (FeaStConv)

Non-Euclidean (Graph)

- **Geodesic CNN (GCNN)[1] ⊂ Anisotropic CNN (ACNN)[2] ⊂ MoNet [3]**

Convolution

$$(f * g)(x) = \sum_{j=1}^{J} g_j D_j(x) f$$

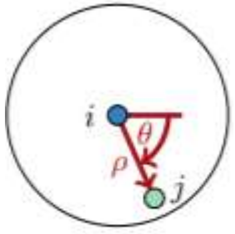
Patch Operator

$$D_j(x) f = \sum_{y \in N(x)} \omega_j(\boldsymbol{u}(x, y)) f(y)$$

Pseudo-coordinate

[1]

[4]



$\mathbf{u}(i, j) = (\rho, \theta)$

GCNN $\omega_j(\boldsymbol{u}) = \exp\left(-\frac{1}{2}(\boldsymbol{u} - \bar{u}_j)^{\mathrm{T}} \begin{pmatrix} \bar{\sigma_\rho}^2 & 0 \\ 0 & \bar{\sigma_\theta}^2 \end{pmatrix} (\boldsymbol{u} - \bar{u}_j)\right)$

covariance (radius, angle direction)

ACNN $\omega_j(\boldsymbol{u}) = \exp\left(-\frac{1}{2}\boldsymbol{u}^T \boldsymbol{R}_{\bar{\theta}_j} \begin{pmatrix} \bar{\alpha} & 0 \\ 0 & 1 \end{pmatrix} \boldsymbol{R}_{\bar{\theta}_j}^T \boldsymbol{u}\right)$

Rotation of θ to the maximum curvature direction

The degree of anisotropy

MoNet $\omega_j(\boldsymbol{u}) = \exp\left(-\frac{1}{2}(\boldsymbol{u} - \mu_j)^{\mathrm{T}} \Sigma_{\mathrm{j}}^{-1} (\boldsymbol{u} - \mu_j)\right)$

Learning parameters

[1] J. Masci et al. "Geodesic convolutional neural networks on Riemannian manifolds", 2015
[2] D Boscaini et al. "Learning shape correspondence with anisotropic convolutional neural networks", 2016
[3] F. Monti et al. "Geometric deep learning on graphs and manifolds using mixture model CNNs", 2016
[4] M. Fey et al. "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels", 2017

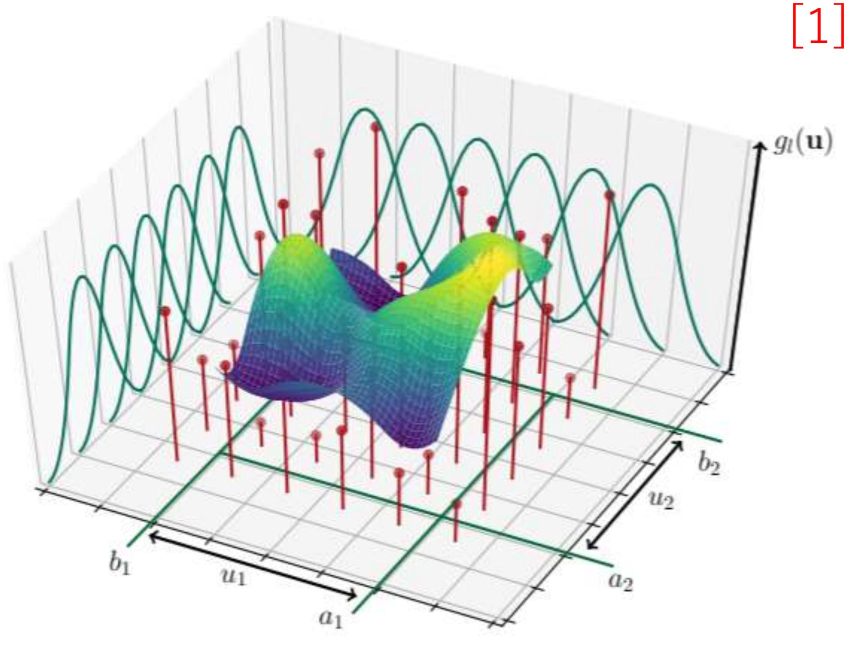
Non-Euclidean (Graph)

- Geodesic CNN (GCNN)
  - Create local coordinate
  - Do not verify the meaningful chart (need to create small radius chart)
- Anisotropic CNN (ACNN)
  - Fourier basis is based on anisotropic heat diffusion eq.
- MoNet
  - Learn filter as parametric kernel
  - Generalization of geodesic CNN and anisotropic CNN

Non-Euclidean (Graph)

- SplineCNN [1]
  - Filter based on B-spline function
  - Efficient computational cost

[1]



$$\boldsymbol{x}_i{}' = \frac{1}{|N(i)|}\sum_{j\in N(i)} \boldsymbol{x}_i \cdot h_{\boldsymbol{\Theta}}(\boldsymbol{e}_{i,j})$$

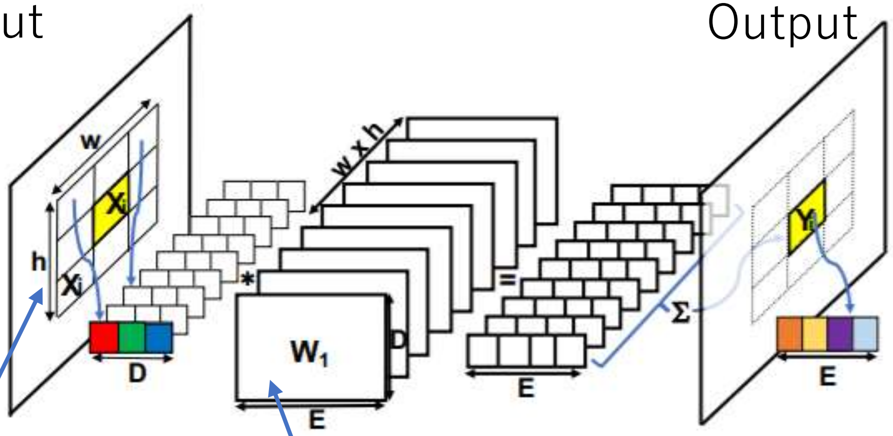
Weighted B-Spline basis

[1] M. Fey et al. "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels", 2017

- FeaStNet [1]
  - Dynamically determine relation b/w filter weight and local graph of a node

Euclidean [1]

Input
Output



Filter
(e.g. $M = 3 \times 3 = 9$)

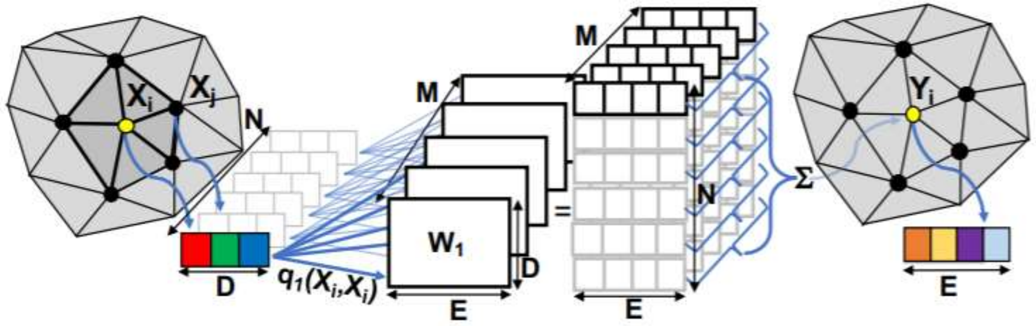
Weight

$$\boldsymbol{x}_i{}' = \sum_{m=1}^{M} \boldsymbol{W}_m \boldsymbol{x}_{n(m,i)}$$

E output feature
D input feature
pixel

FeaStNet [1]

Input
Output



#neighbor
(e.g. $N = 6$)

$$\boldsymbol{x}_i{}' = \frac{1}{|N(i)|} \sum_{j \in N(i)} \sum_{m=1}^{M} q_m(\boldsymbol{x}_i, \boldsymbol{x}_j) \boldsymbol{W}_m \boldsymbol{x}_j$$

$$q_m(\boldsymbol{x}_i, \boldsymbol{x}_j) = softmax_j(\boldsymbol{u}_m^T(\boldsymbol{x}_i - \boldsymbol{x}_j) + c_m)$$

[1] N Verma et al. "FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis", 2017

Non-Euclidean (Graph)

- PyTorch-geometric
  - https://github.com/rusty1s/pytorch_geometric
  - Library based on PyTorch
  - For point cloud, mesh (not only graph)
  - Include Point cloud, graph-type approach code
    - PointNet++, DGCNN, PointCNN
    - ChebNet, GCN, MoNet, SplineCNN, FeaStNet
  - Easy to get the famous sample data and transform same data format
    - ModelNet, ShapeNet, etc.
  - Many example and benchmark

Accuracy

- **Accuracy (Classification)**
  - **around 90% in any method (except VoxNet)**

Method	"ModelNet40" Overall Acc. [%] / Mean class Acc. [%]	"SHREC" Overall Acc. [%]
VoxNet	85.9 / 83.0	–
MVCNN	– / 90.1	**96.09**
PointNet	89.2 / 86.0	–
PointNet++	90.7 / –	–
DGCNN	**92.9 / 90.2**	–
PointCNN	92.2 / 88.1	–
MeshNet	– / 91.9	–
MeshCNN	– / –	91.0

※Please refer the detail in each paper (mentioned in each page)

- **Accuracy (Segmentation)**

Method	Part segmentation "ShapeNet" mIoU (mean per-class part-averaged IoU) [%]	Part segmentation "ScanNet" Acc. [%]	Part segmentation "COSEG" Acc. [%]	Scene segmentation "S3DIS" Acc. [%] / mIoU [%]	Human body segmentation "including SCAPE, FAUST etc." Acc. [%]
PointNet	80.4 (57.9 – 95.3)	73.9	54.4 – 91.5	78.6 / –	90.77
PointNet++	81.9 (58.7 – 95.3)	84.5	79.1 – 98.9	– / –	–
DGCNN	82.3 (**63.5** – **95.7**)	–	–	**84.1** / 56.1	–
PointCNN	**84.6**	**85.1**	–	– / **65.39**	–
MeshCNN	–	–	**97.56** – **99.63**	– / –	**92.30**
FeaStNet	81.5	–	–	– / –	–

※Please refer the detail in each paper (mentioned in each page)

Dataset

• 3D Dataset

	Contents	Data Format	Purpose	PyTorch-geometric
ModelNet10/40	3D CAD Model (10 or 40 classes)	Mesh (.OFF)	Classification	ModelNet
ShapeNet	3D Shape	Point Cloud (.pts)	Segmentation	ShapeNet
ScanNet	Indoor Scan Data	Mesh (.ply)	Segmentation	-
S3DIS (original, .h5)	Indoor Scan Data	Point Cloud	Segmentation	S3DIS

ScanNet：registration required
S3DIS : registration required (for original)

• **3D Dataset**

	Contents	Data Format	Purpose	PyTorch-geometric
SHREC	many type for each contest	-	Retrieval	-
SHREC2016	Animal, Human (Part Data)	Mesh (.OFF)	Correspondence	SHREC2016
TOSCA	Animal, Human	Mesh (same #vertices at each category, separate file of vertices and triangles)	Correspondence	TOSCA
PCPNet	3D Shape	Point Cloud (.xyz) (Including normal, curvature files.)	Estimation of local shape (Normal, curvature)	PCPNet
FAUST	Human body	Mesh	Correspondence	FAUST

FAUST(Note) : registration required

- **Material of 3D deep learning (3D / point cloud)**

Paper	Comment
A survey on Deep Learning Advances on Different 3D Data Representations	• Review of 3D Deep Learning • Easier to read it • Written from point of view about Euclidean and Non-Euclidean method
Paperwithcode	• Paper w/ code about 3D
Point Cloud Deep Learning Survey Ver. 2	• Deep learning for point cloud • Survey of many papers

- **Material of 3D deep learning (graph)**

Paper	Comment
Geometric deep learning: going beyond Euclidean data	• Review of geometric deep learning
Geometric Deep Learning	• summary of paper and code about geometric deep learning
Geometric Deep Learning on Graphs and Manifolds (NIPS2017)	• Presentation (youtube) about geometric deep learning

Summary

- There are many methods of 3D deep learning.
- Two main method
  - Euclidean vs Non-Euclidean
  - Euclidean Method
    - Projections / Multi-View / Voxel
  - Non-Euclidean Method
    - Point Cloud / Mesh / Graph
- Each method have merit and demerit.
  - We need to choose the better method for each data type and application.
- The research about 3D deep learning is growing.

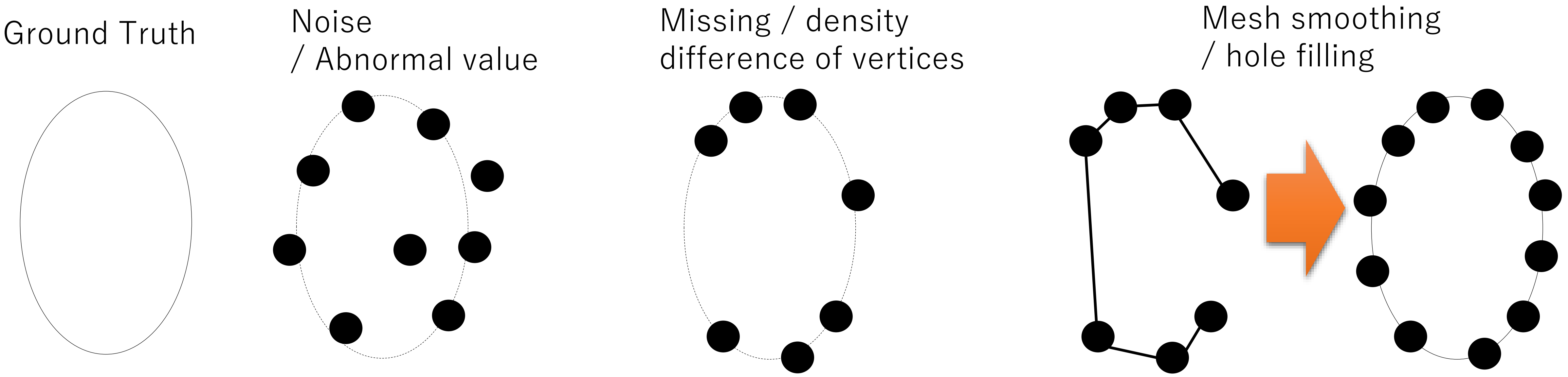
- Appendix
  - Mesh Generation
  - Laplacian on Graph
  - Correspondence

- Mesh Generation
  - In this material, I have summarized these materials.

Link	Contents
点群面張り（精密工学会）	• Surface reconstruction
メッシュ処理（精密工学会）	• Mesh processing
CV勉強会@関東発表資料　点群再構成に関するサーベイ	• Survey of point cloud reconstruction

- **Difficulty of Mesh Generation**

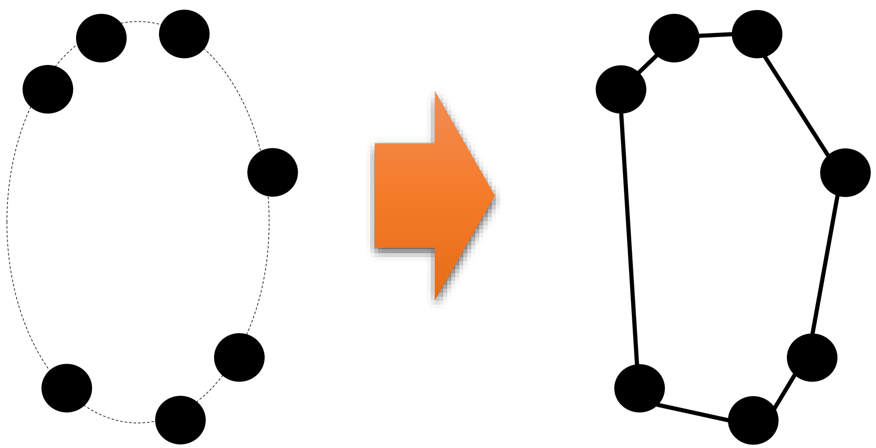
Processing	Difficulty
Pre-processing	Reduction of Noise / Missing / Abnormal value / density difference of vertices
Post-processing	Mesh smoothing / hole filling



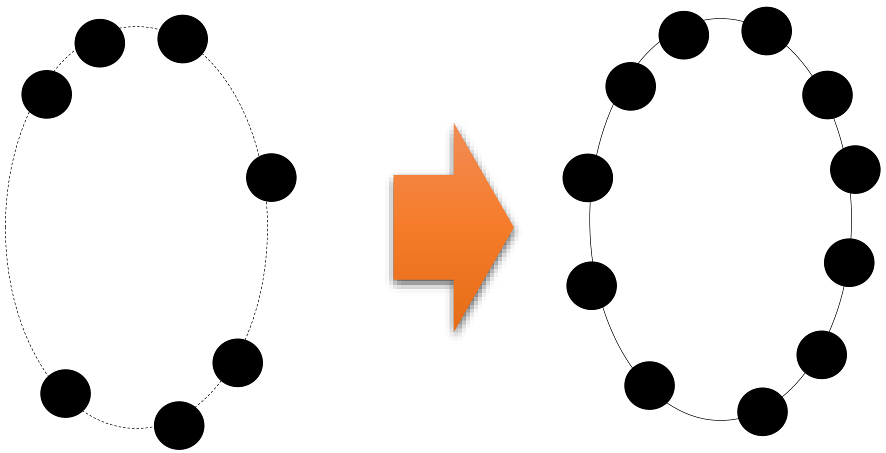
- **Kinds of Mesh Generation**

Kind	Feature	Classification of the method
Direct Triangulation	Direct mesh generation form point cloud	Explicit method
Surface Smoothness	Smooth surface mesh from point cloud	Implicit method

Direct Triangulation



Surface Smoothness



Appendix : Mesh Generation

- Classification of the method
  - In general, it is easier to use the implicit method, since there are noise of point cloud.

Classification of the method	Information to use	Influence of noise and density of vertices	Guarantee of accuracy
Explicit method	Vertices	Large (error of vertices = error of meshes)	◎
Implicit method	Meshes based on isosurface of function fields which is calculated from vertices	Small	○

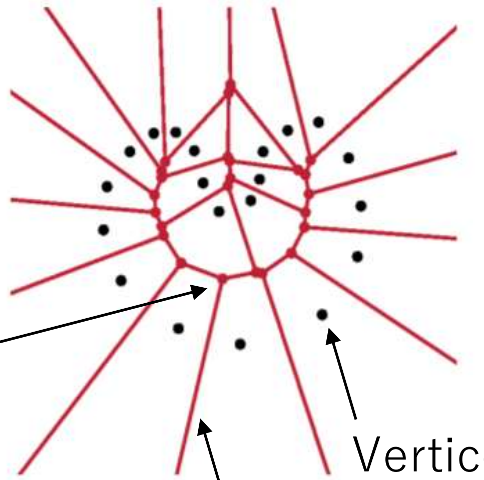
- **Kinds of Mesh Generation (Detail)**
  - **Direct Triangulation (example of built-in function in MeshLab)**

Method	Feature
Voronoi-Based Surface Reconstruction	Creation of Delaunay diagram adding the vertices using Voronoi diagram
Ball-Pivoting Algorithm	Roll the ball over the point cloud and generate mesh from the point cloud located within a certain distance

Appendix : Mesh Generation

- Voronoi-Based Surface Reconstruction
  - Voronoi diagram
    - Region divided by the bisector of each vertices (in 2D)
  - Delaunay triangulation
    - Triangulation by connection of vertices

Example of 2D



Voronoi Vertices (V)

bisector

Vertices (S)

[1]

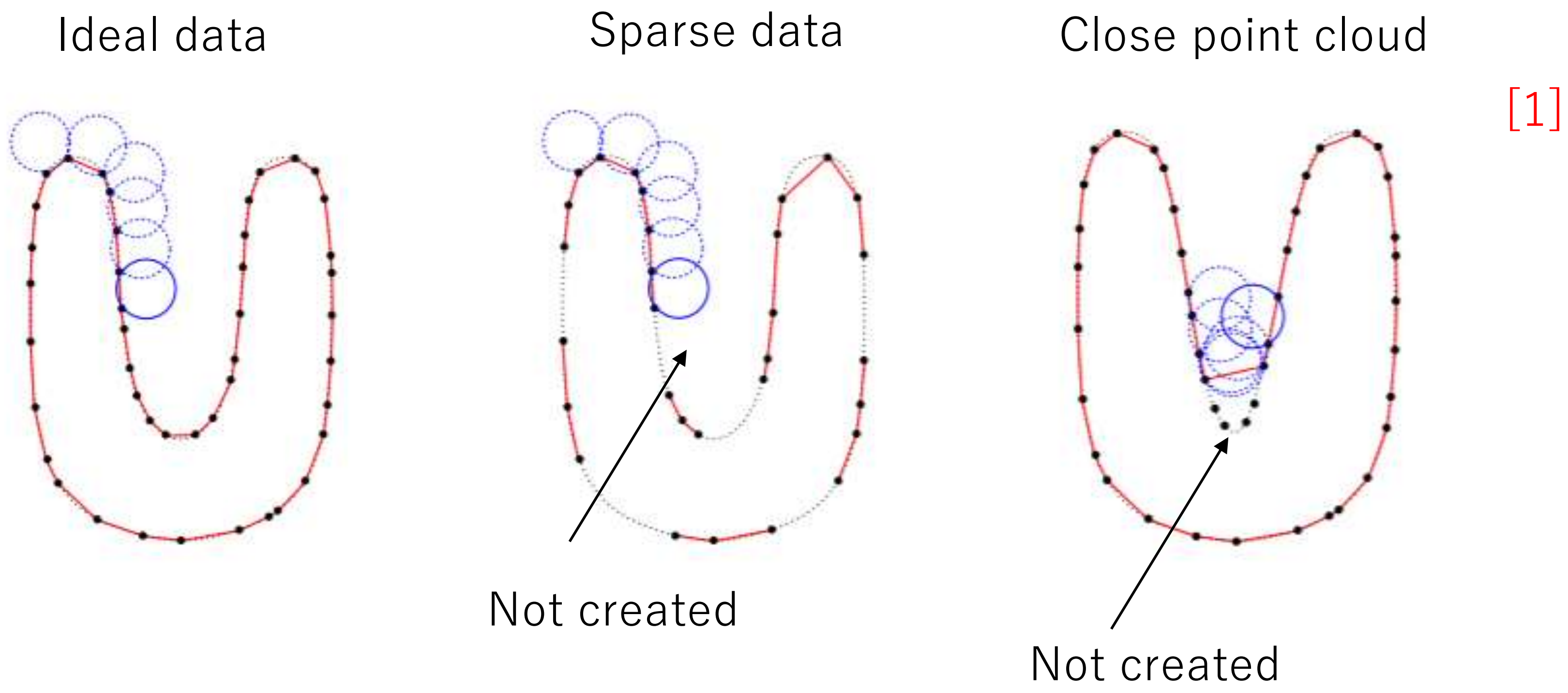
Surface (black line)

[1]

Delaunay triangulation of S and V (black + red line)

[1] N. Amenta et al. "A New Voronoi-Based Surface Reconstruction Algorithm", 1998

- **Ball-Pivoting Algorithm**



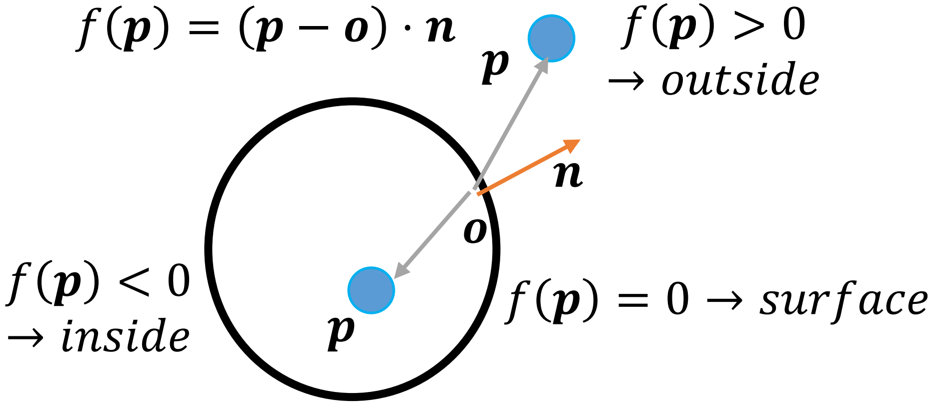
[1] F Bernardini et al. "The Ball-Pivoting Algorithm for Surface Reconstruction", 1999

- **Kinds of Mesh Generation (Detail)**
  - **Surface Smoothness (example of built-in function in MeshLab)**

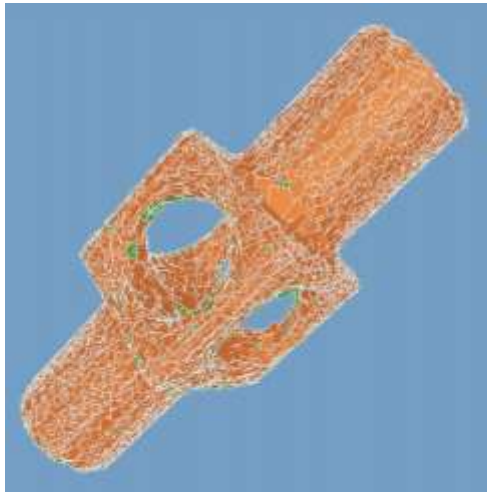
Method	Feature
Signed distance function + Marching Cubes	Creation of Signed distance function by using the distance b/w vertices and surface + Mesh generation by using Marching Cubes
Screened Poisson surface reconstruction (Poisson surface reconstruction)	Distinguish b/w inside and outside of surface by using Poisson eq.

Appendix : Mesh Generation

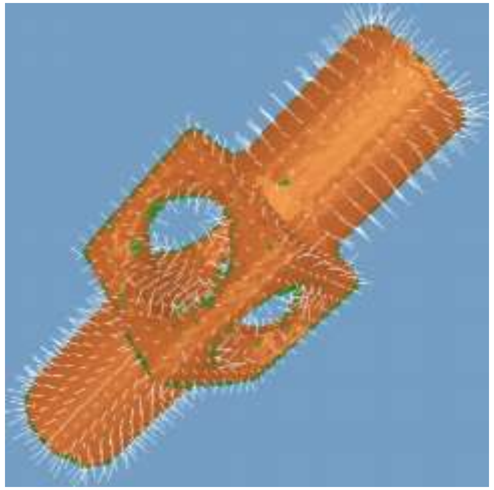
- **Signed distance function + Marching Cubes**



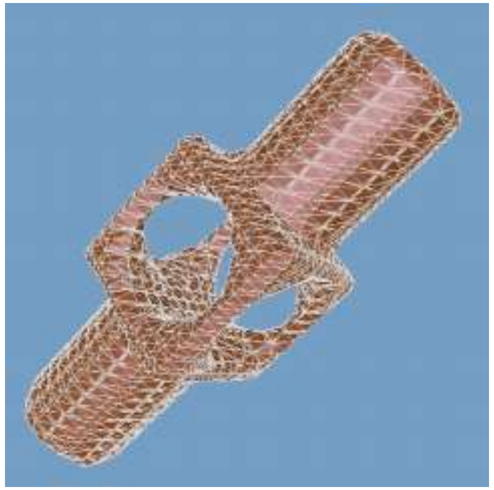
Oriented tangent planes



Estimated signed distance



Output of
modified marching cubes



[1]

[1] H. Hoppe et al. "Surface Reconstruction from Unorganized Points", 1992

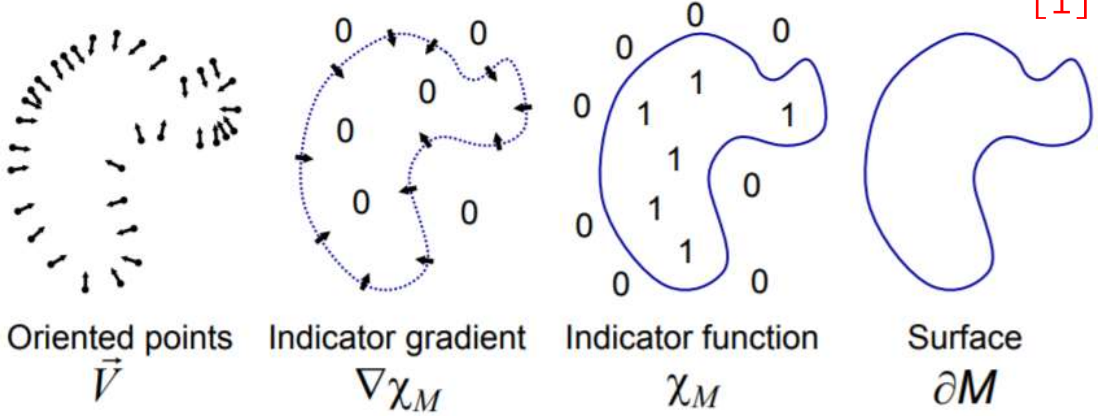
- **Screened Poisson surface reconstruction**
  - **get Indicator Function by solving the Poisson eq.**

Poisson eq.

$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \boldsymbol{V}$$

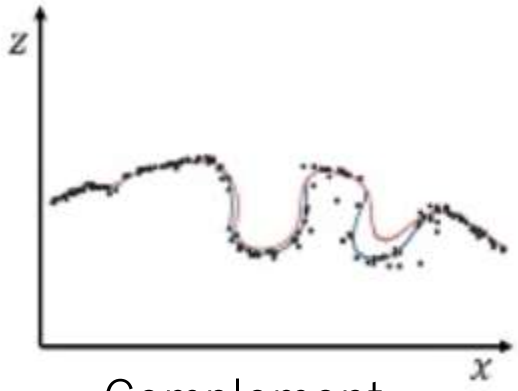
Poisson surface reconstruction

[1]



Screened Poisson
surface reconstruction

[2]



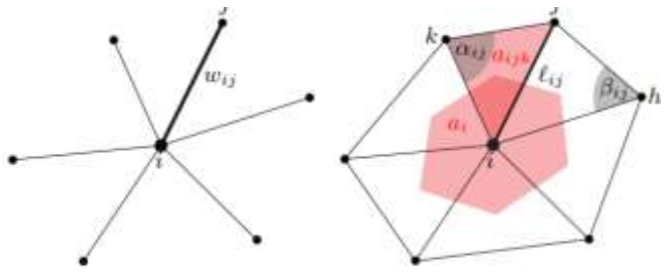
Complement
b/w point cloud

[1] M. Kazhdan et al. "Poisson Surface Reconstruction", 2006
[2] M. Kazhdan et al. "Screened Poisson Surface Reconstruction", 2013

Appendix : Laplacian on Graph

- **Laplacian on Graph [1]**

[1]



Graph (undirected) $(\mathcal{V},\mathcal{E})$

weight

$\mathcal{V}=\{1,\cdots,n\}$ a_i

$\mathcal{E}\subseteq \mathcal{V}\times\mathcal{V}$ ω_{ij}

$f:\mathcal{V}\to\mathbb{R}, F:\mathcal{E}\to\mathbb{R}$

Grad. $(\nabla f)_{ij}=f_i-f_j$

div. $(div\ F)_i=\dfrac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}}\omega_{ij}F_{ij}$

Laplacian $(\Delta f)_i=\dfrac{1}{a_i}\sum_{(i,j)\in\mathcal{E}}\omega_{ij}(f_i-f_j)$

$\Delta\equiv -div\,\nabla$

→Laplacian eigenvalues $\lambda>0$

Mesh

$$\omega_{ij}=\frac{-\ell_{ij}^2+\ell_{jk}^2+\ell_{ki}^2}{8a_{ijk}}+\frac{-\ell_{ij}^2+\ell_{jh}^2+\ell_{hi}^2}{8a_{ijh}}=\frac{1}{2}(\cot\alpha_{ij}+\cot\beta_{ij})$$

$$a_i=\frac{1}{3}\sum_{jk:(i,j,k)\in\mathcal{F}}a_{ijk}$$

$$a_{ijk}=\left[s_{ijk}(s_{ijk}-\ell_{ij})(s_{ijk}-\ell_{jk})(s_{ijk}-\ell_{ki})\right]^{1/2}$$

$$s_{ijk}=\frac{1}{2}(a_{ij}+a_{jk}+a_{ki})$$

[1] M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016

• Laplacian on Graph [1]

Laplacian (as matrix)

$$\Delta \boldsymbol{f} = \boldsymbol{A}^{-1}(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{f}$$

$$\boldsymbol{W} = (\omega_{ij})$$

$$\boldsymbol{A} = diag(a_1, \cdots, a_n)$$

$$\boldsymbol{D} = diag\left(\sum_{j:j\neq i} \omega_{ij}\right)$$

$$\boldsymbol{f} = (f_1, \cdots, f_n)^T$$

Laplacian	Δ	Condition
Unnormalized graph Laplacian	$\Delta = \boldsymbol{D} - \boldsymbol{W}$	$A = I$
Normalized Symmetry Laplacian	$\Delta = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{W}\boldsymbol{D}^{\frac{1}{2}}$	$A = D$ + Normalization
Random walk Laplacian	$\Delta = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W}$	$A = D$

[1] M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016

- **Laplacian on Graph [1]**
  - **Convolution**

Conv.

$$(f * g)(x) = \sum_{i \geq 0} \hat{f}_i \hat{g}_i \phi_i(x)$$

Matrix

$$\boldsymbol{f} * \boldsymbol{g} = \boldsymbol{\Phi} diag(\hat{g}) \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{f}$$

$$\boldsymbol{f} = (f_1, \cdots, f_n)^T$$

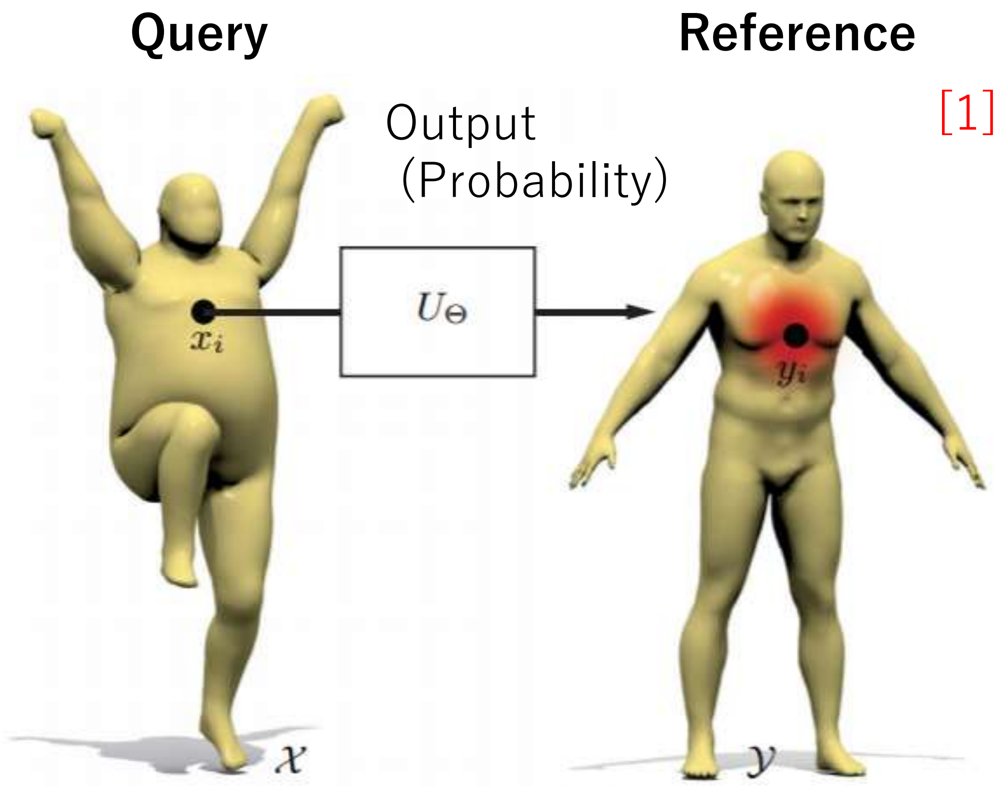
$$\hat{\boldsymbol{g}} = (\hat{g}_1, \cdots, \hat{g}_{\boldsymbol{n}})$$

$$\boldsymbol{\Phi} = (\phi_1, \cdots, \phi_n)$$

[1] M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016

Appendix :Correspondence

- **Correspondence [1]**



Each query vertex has labels as all reference vertices

input x_i ⟷ label $(y_1, \cdots, y_N)$

One-hot vector

Output (Probability) $(p_1, p_2, \cdots, p_N)$

Correct Label $(1, 0, \cdots, 0)$

→ Loss

[1] M. M. Bronstein et al., "Geometric deep learning: going beyond Euclidean data", 2016

Arithmer 株式会社
〒106-6040
東京都港区六本木一丁目６番１号 泉ガーデンタワー 38/40F(受付)
03-5579-6683
https://arithmer.co.jp/

Arithmer