Course Notes: Deep Learning for Visual Computing

Peter Wonka

October 24, 2021

Contents

1 Point Cloud Processing

1.1 TODO

- Add a figure showing the scalar functions in pointnet (figure in the appendix)
- Add an explanation of GCNNs
- Add an explanation of spatial transformers
- Add an example calculation for PointNet
- Add Literature as href

1.2 Overview

- Point cloud

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1^\intercal \\ \mathbf{p}_2^\intercal \\ \vdots \\ \mathbf{p}_N^\intercal \end{bmatrix} \in \mathbb{R}^{N \times 3}$$

- Per-point features

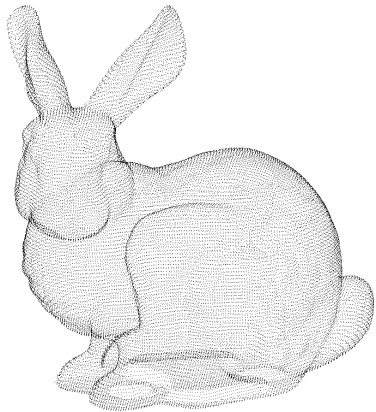
$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\intercal \\ \mathbf{x}_2^\intercal \\ \vdots \\ \mathbf{x}_N^\intercal \end{bmatrix} \in \mathbb{R}^{N \times D}$$

- N - number of points in a point cloud
- $\mathbf{p}_i$ - the spatial location of point i

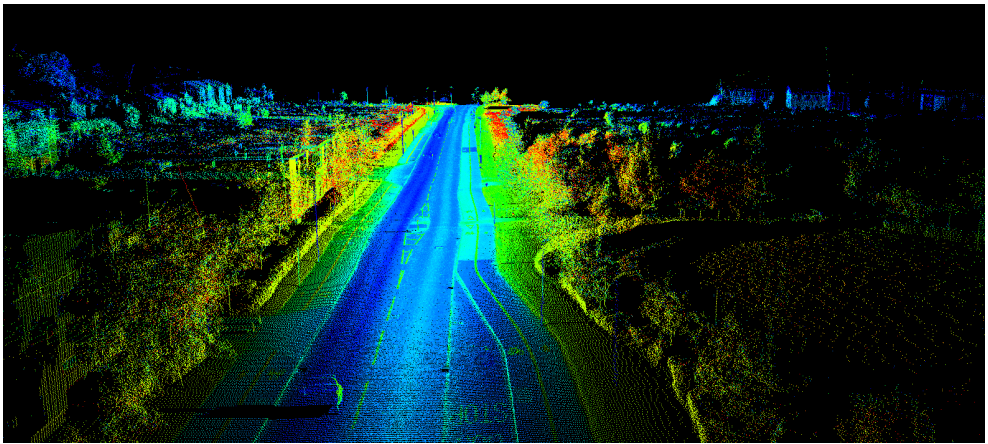
- $\mathbf{x}_i$ - the feature vector of point i
  - e.g. color, normal, curvature, tangent vector, material properties, ...
- D - the dimension of the feature vector

1.3 Examples

- Point cloud for a single object



- Point cloud for a scene



1.4 Permutation Invariance

- How to order a set of points?
  - Point cloud is given as unordered set
  - No easy way to order a point cloud to preserve spatial neighborhoods
    - 1D, 2D: does not preserve all neighbors
    - 3D: memory consuming

- **Permutation Invariance:** network should give the same output for every permutation of the input point cloud
- Original point cloud:

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$

- Point cloud reordered by a permutation:

$$\{\mathbf{x}_{\pi_1}, \mathbf{x}_{\pi_2}, \ldots, \mathbf{x}_{\pi_N}\}$$

- Permutation can be defined by a bijective function from the index set onto itself:

$$\pi : \{1, 2, \ldots, N\} \to \{1, 2, \ldots, N\}$$

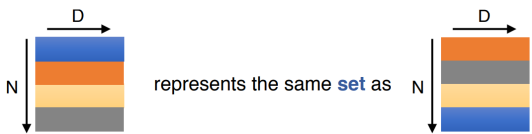
- Example:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 2 & 1 & 4 \end{pmatrix} \quad \text{e.g.} \quad \pi(3) = 2, \pi(5) = 4$$

- A **symmetric function** is permutation invariant
  - For all permutations π:

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = f(\mathbf{x}_{\pi_1}, \mathbf{x}_{\pi_2}, \ldots, \mathbf{x}_{\pi_N}) \tag{1.1}$$

  - Visualization (each point of dimension D is visualized by a different color)
- Example functions that are symmetric: **max, min, sum, average, softmax, product**



1.5 Example Symmetric Function Computation

- Example point cloud (4 points in $\mathbb{R}^3$):

$$\boldsymbol{P} = \begin{pmatrix} 1.0 & 2.0 & 0.3 \\ 1.0 & 2.2 & 0.5 \\ 0.2 & 1.5 & 1.5 \\ 1.2 & 0.5 & 0.5 \end{pmatrix}$$

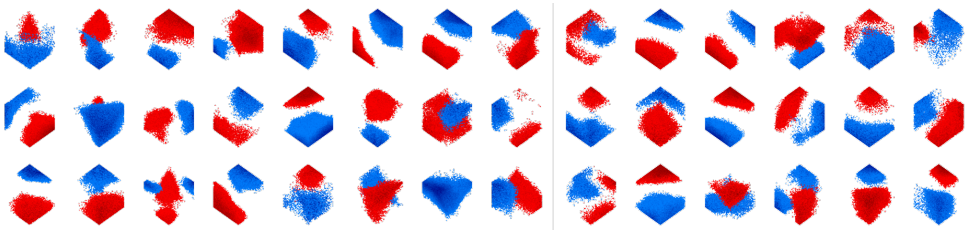
- Many point networks compute symmetric functions column-wise (per dimension), e.g.:

$$\max(\boldsymbol{P}) = \max(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = (1.2, 2.2, 1.5)$$

- Notation: we write $\max(\mathbf{h}_1, \ldots, \mathbf{h}_n)$ or $\max\{\mathbf{h}_i\}$ to denote the maximum computation per dimension
  - the output has as many dimensions as each of the vectors $\mathbf{h}_i$

1.6 What can an MLP compute from a single point?

- Many networks use an MLP to process each point separately
- What can an MLP compute for a single point?



- Let's assume the MLP takes a 3D point as input and maps it to a 1024D feature space:

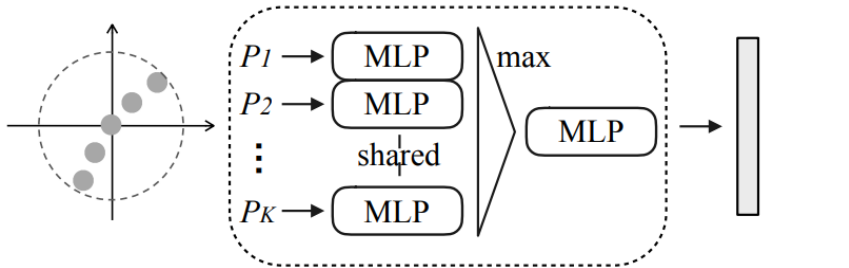
$$h(\mathbf{p}) = (h_1(\mathbf{p}), h_2(\mathbf{p}), \ldots, h_{1024}(\mathbf{p})) \tag{1.2}$$

$$h_i(\mathbf{p}) : \mathbb{R}^3 \rightarrow \mathbb{R} \tag{1.3}$$

- Each of the 1024D can approximate an arbitrary function, best imagined as 3D scalar field
- The MLP basically evaluates the 1024 functions using the input point

1.6.1 PointNet

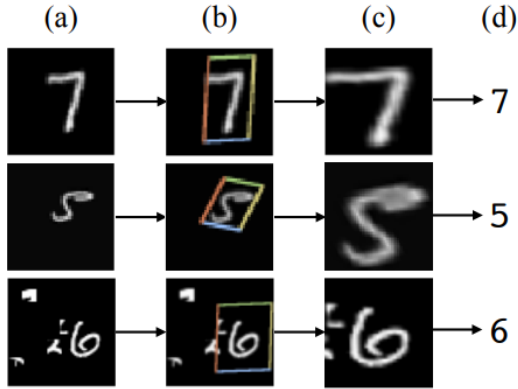
- Literature: PointNet (CVPR 2017)
- Ideas:
  - Use symmetric functions
  - $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \max\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$
  - $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \text{sum}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$
- Basic PointNet architecture (also called Vanilla PoinNet)
  - $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \gamma\left(\max_{i=1,2,\ldots,N}\{h(\mathbf{x}_i)\}\right) = \gamma\left(\max\left(h(\mathbf{x}_1), \ldots, h(\mathbf{x}_N)\right)\right.$
  - $h : \mathbb{R}^D \mapsto \mathbb{R}^{D'}$ - modeled by an MLP, e.g. $D = 3$ and $D' = 1024$
  - $\max : \mathbb{R}^{N \times D'} \mapsto \mathbb{R}^{D'}$ - maximum per dimension
  - $\gamma : \mathbb{R}^{D'} \mapsto \mathbb{R}^{D''}$ - MLP
  - PointNet encodes a function f that is a symmetric function



- Theoretical analysis,
  - Universal Approximation Theorem
  - A Hausdorff continuous symmetric function f can be arbitrarily approximated by PointNet,

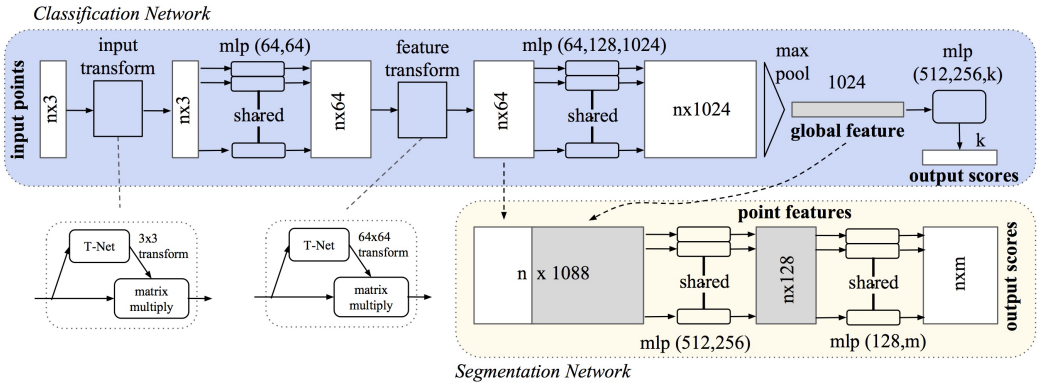
$$\left| f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) - \gamma\left(\max_{i=1,2,\ldots,N} \{h(\mathbf{x}_i)\} \right) \right| \leq \epsilon$$

- Align point cloud to a canonical space,
  - A basic PointNet network
  - Input - $\mathbf{X} \in \mathbb{R}^{N \times D}$
  - Output - $D \times D$ transform matrix $\mathbf{T}$
  - Aligned point cloud - $\mathbf{XT}$
  - 2D Spatial Transformer example -



- Advantages,
  - Can handle an anbitrary number of points in a single data sample
  - Basic PointNet modules can be stacked

- Can be used in multiple applications, e.g. classification and segmentation
- Can be used as point cloud feature extraction network (back-bone network in many point cloud papers)



- Qualitative results for part segmentation.

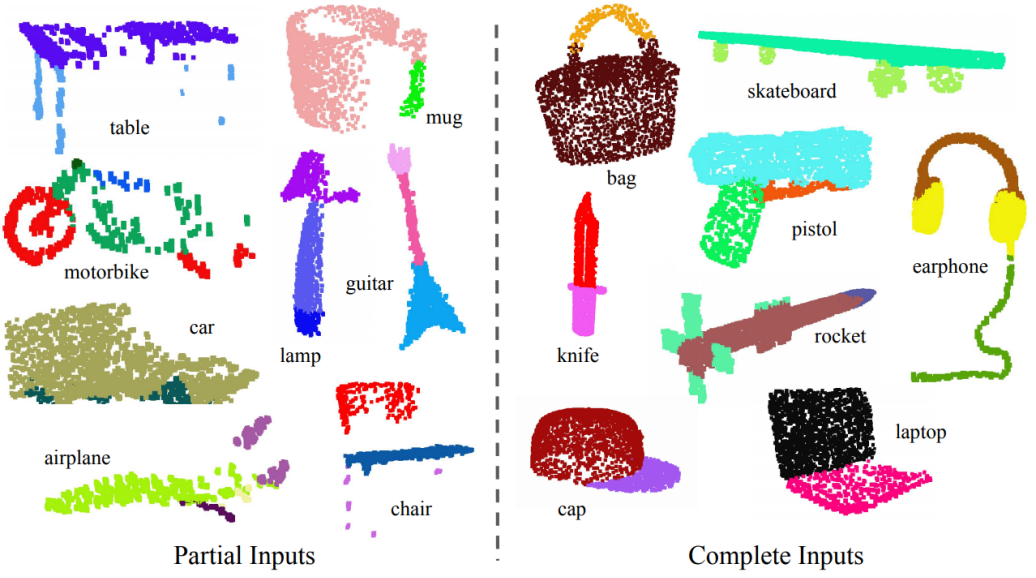
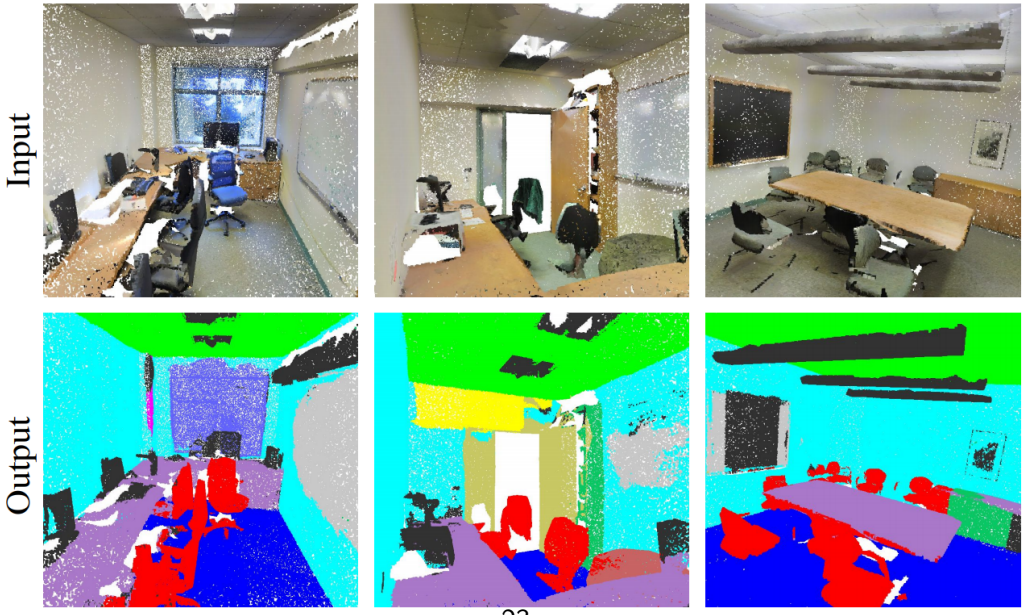


table
mug
motorbike
guitar
car
lamp
airplane
chair
Partial Inputs
bag
skateboard
pistol
earphone
knife
rocket
laptop
cap
Complete Inputs

- Qualitative results for semantic segmentation.


Input
Output

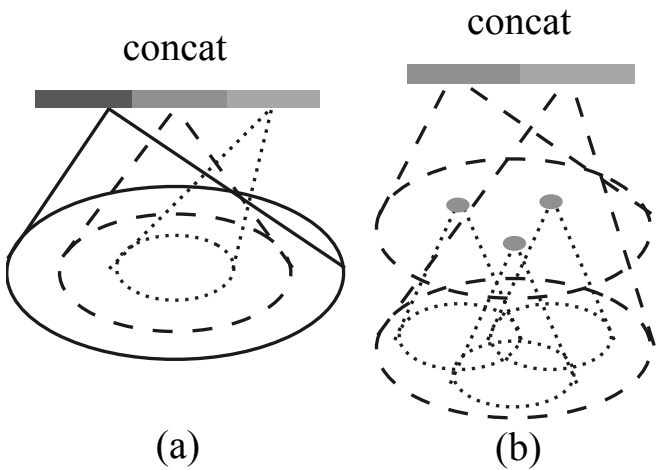
1.7 Software Libraries

- Torch Points 3D

1.8 Selected Deep Learning Papers for Point Clouds

1.8.1 PointNet++

- Literature: PointNet++ (NIPS 2017)
- Ideas:
  - Exploiting local structure by introducing pooling to PointNet



- Set abstraction - partition a point cloud into overlapping local regions
  - **Sampling layer**: sample M points among N points

- Input - original point cloud $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$
- Output - sampled point cloud $\{\mathbf{x}_m : m \in \mathcal{M}\}$ where $\mathcal{M} \subset \{1, 2, \ldots, N\}$
- Farthest point sampling (FPS)

- **Grouping layer**: find K nearest neighbors for sampled points
  - Input - original point cloud $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$, features $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, and sampled index set $\{m : m \in \mathcal{M}\}$
  - Output - $\forall m \in \mathcal{M}$, neighborhood point sets $\mathcal{N}_m$, of which every point is a neighborhood of point m in Euclidean space.
  - Ball query or k-nearest neighbor

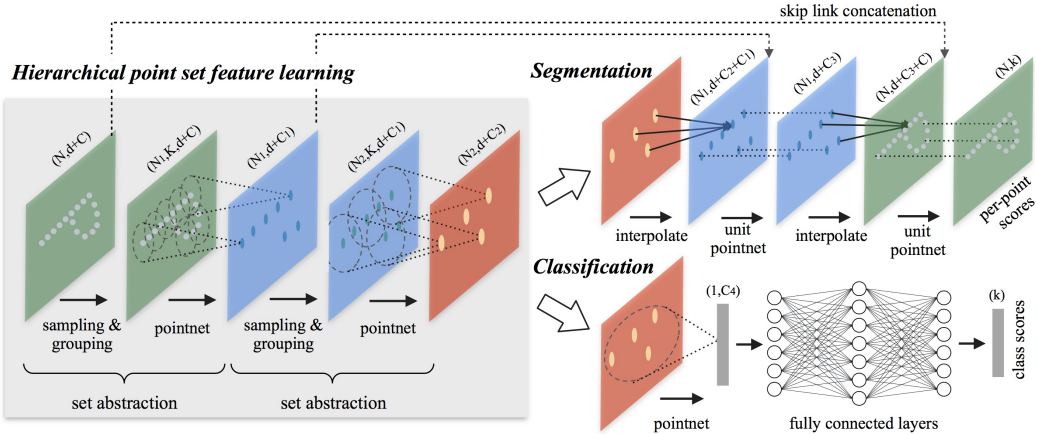
- Comparison with PointNet
  - PointNet,

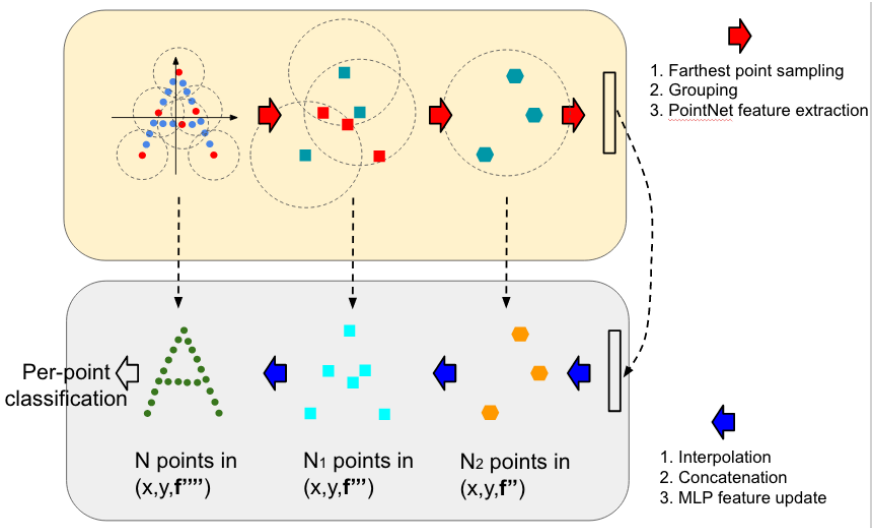
$$\mathbf{x}'_i = h(\mathbf{x_i}), \quad \forall i \in \{1, 2, \cdots, N\}$$

- PointNet++,

$$\mathbf{x}'_m = \max_{j \in \mathcal{N}_m \cup m} h(\mathbf{x}_j, \mathbf{p}_j - \mathbf{p}_m), \quad \forall m \in \mathcal{M} \subset \{1, 2, \cdots, N\},$$

where $h : \mathbb{R}^{D+3} \mapsto \mathbb{R}^{D'}$

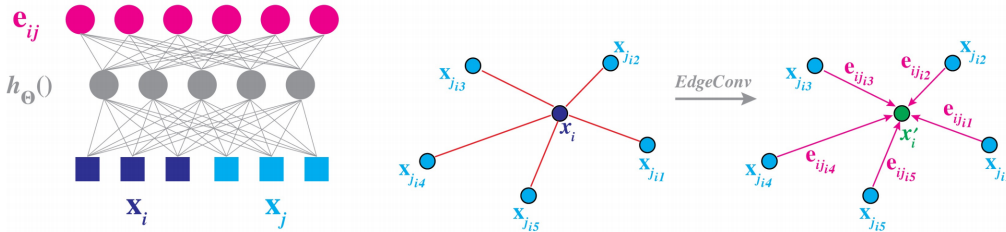



1. Farthest point sampling
2. Grouping
3. PointNet feature extraction
Per-point classification
N points in (x,y,f'''')
N1 points in (x,y,f''')
N2 points in (x,y,f'')
1. Interpolation
2. Concatenation
3. MLP feature update

1.8.2 Dynamic Graphc CNNs (DGCNN)

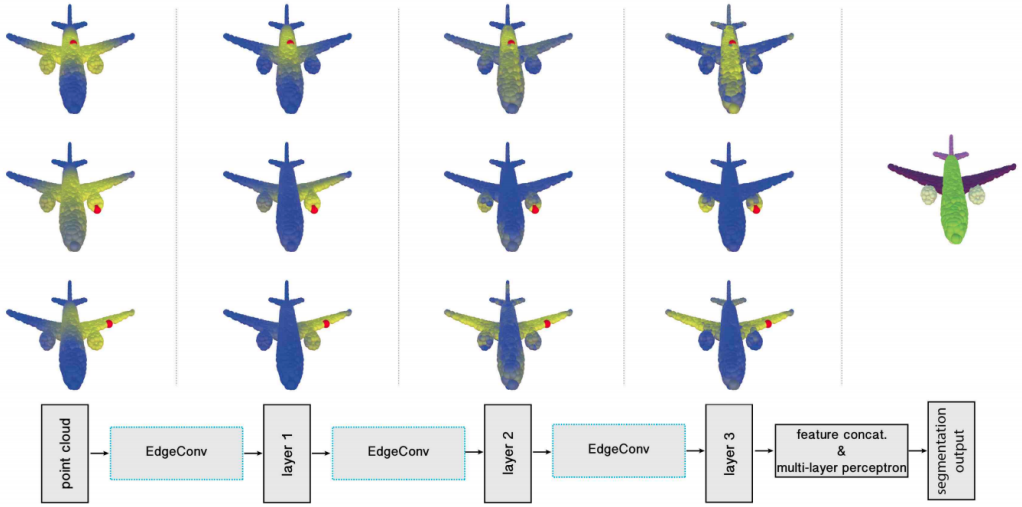
- Literature: DGCNN (TOG 2019)
- Ideas:
  - Building a dynamic graph in feature space
  - Exploiting local relative features
- Dynamic graph
  - Input: points (features) $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$
    - x_i can be coordinates, coordinates + other features, or intermediate features in the network
  - Output: neighborhood point sets $\mathcal{N}_i$ of point i in feature space

- Edge convolution:
  - $\mathbf{e_{ij}} = h_\Theta(\mathbf{x_i}, \mathbf{x_j}), \quad \forall j \in \mathsf{N}_i$
  - $h_\Theta : \mathbb{R}^{D+D} \mapsto \mathbb{R}^{D'}$ - MLP



- Multiple special cases are discussed in the paper, e.g. $\mathrm{h}_\Theta(\mathbf{x_i}, \mathbf{x_j} - \mathbf{x_i})$
- Aggregation: x'_i is computed by (weighted) sum or max pooling of the edge features

- Comparison with PointNet++
  - Distances are calculated in (high-dimensional) feature space instead of 3D Euclidian space
  - Update features of all points, not only sampled points


point cloud
EdgeConv
layer 1
EdgeConv
layer 2
EdgeConv
layer 3
feature concat.
&
multi-layer perceptron
segmentation
output

1.8.3 FeastNet Setup

- Literature: FeaStNet (CVPR 2018)
- Consider a traditional CONV-layer that a tensor of size (W, H, C_{in}) as input and outputs tensor (W, H, C_{out})
  - Normally we visualize convolutions per layer and the layers are then added together
- But now let's imagine that the CONV-layer is a 1×1 convolution
  - Each input pixel $p(w, h)$ of dimension C_{in} is multiplied with a matrix $\boldsymbol{W}$ to give output pixel of dimension C_{out}
  - $W \in \mathbb{R}^{C_{out} \times C_{in}}$
- Now let's imagine that the CONV-layer is a 3×3 convolution
  - We could actually imagine it as a sum of 9 different 1×1 convolutions
  - Each 1×1 convolution is given by a matrix $\boldsymbol{W}_m, 1 \leq m \leq 9$

- We can now write the convolution as

$$p'(w,h) = bias + \sum_{m=1}^{9} \boldsymbol{W}_m p(w + offx(m), h + offy(m)) \tag{1.4}$$

- To generalize the convolution we can do two things
  - For each neighboring pixel we compute all feature transformations $\boldsymbol{W}_m$ instead of only one
  - We can use an **attention module** to compute a weight for each of the matrices $\boldsymbol{W}$

1.8.4 FeastNet Details

- FeaStConv
  - $$\mathbf{x}'_i = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \sum_{l=1}^{L} q_l(\mathbf{x}_i, \mathbf{x}_j) h_l(\mathbf{x}_j), \quad \forall i \in [1, 2, \ldots, N]$$
  - $h_l : \mathbb{R}^D \mapsto \mathbb{R}^{D'}$ - fully connected layer
  - $q_l = \text{softmax}_l(\mathbf{u}_l^\intercal(\mathbf{x}_i - \mathbf{x}_j) + c_l) : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$
- Comparison with EdgeConv
  - Learn multiple transformations $(h_l(\cdot))$ instead of one $(h(\cdot))$
  - Combine transformed features with attention $q_l(\mathbf{x_i}, \mathbf{x}_j)$.

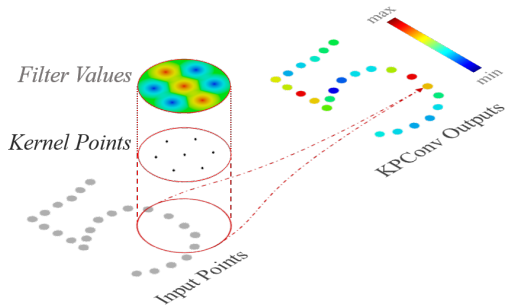
1.8.5 PPFNet

- Literature: PPFNet, CVPR 2018
- Rotation-invariant Point Pair Features (PPF)
  - PPF is computed for a pair of points
  - $\psi_{i,j} = (\mathbf{d}_{j,i}, \angle(\mathbf{n}_i, \mathbf{d}_{j,i}), \angle(\mathbf{n}_j, \mathbf{d}_{j,i}), \angle(\mathbf{n}_i, \mathbf{n}_j))$
  - $\mathbf{n}_i$ - surface normal of point i
  - $\mathbf{p}_i$ - 3D position of point i
  - $\mathbf{d}_{j,i} = \mathbf{p}_j - \mathbf{p}_i$
  - $\angle(\mathbf{x}, \mathbf{y})$ is the angle between two vectors
  - $\psi : \mathbb{R}^{3+3+3+3} \mapsto \mathbb{R}^6$
-
- PPFConv

- $$\mathbf{x}_i' = \max_{j \in \mathcal{N}_i \cup i} \{h(\mathbf{x}_j, \mathbf{n}_j, \psi_{i,j})\}, \quad \forall i \in [1, 2, \ldots, N]$$
- $h : \mathbb{R}^{D+6} \mapsto \mathbb{R}^{D'}$ - MLP

1.8.6 KPConv

- Literature: Thomas et al., KPConv: Flexible and Deformable Convolution for Point Clouds, ICCV 2019



- General **point convolution** (computed per point $\mathbf{x}$):

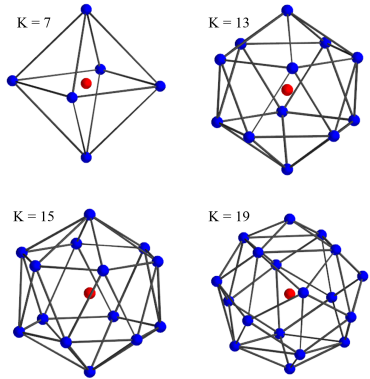
$$(\boldsymbol{F} * g)(\mathbf{x}) = \sum_{\mathbf{x_i} \in N_x} g(\mathbf{x_i} - \mathbf{x})\mathbf{f_i} \tag{1.5}$$

  - N: number of points
  - Points: $\boldsymbol{P} \in \mathbb{R}^{N \times 3}$; a point is denoted as $\mathbf{x_i}$
  - Features: $\boldsymbol{F} \in \mathbb{R}^{N \times D}$; a feature vector is $\mathbf{f_i}$
  - Neighborhood: $N_x = \{\mathbf{x_i} \in \boldsymbol{P} | \quad \|\mathbf{x_i} - \mathbf{x}\| < r\}$
    - r: radius of the sphere

- Idea: Define the kernel function g using K kernel points
- Define all neighboring points in coordinates centered on $\mathbf{x}$

$$\mathbf{y_i} = \mathbf{x_i} - \mathbf{x} \tag{1.6}$$

  - K kernel points: $\tilde{\mathbf{x}}_\mathbf{k}$ (locations computed by optimization)



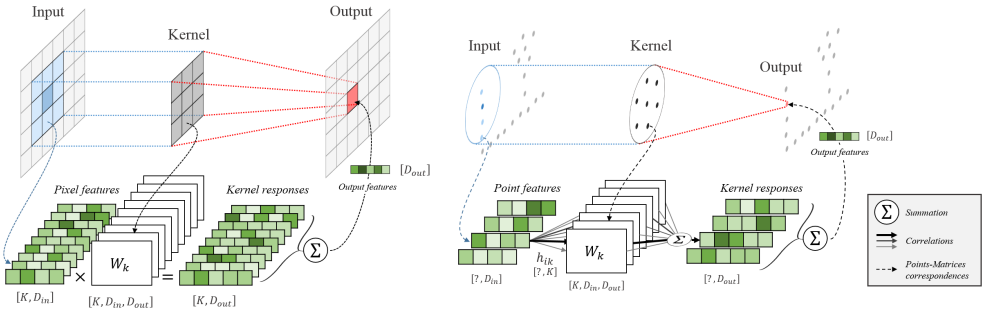
- Weight matrices: $\boldsymbol{W_k} \in \mathbf{R}^{D_{in} \times D_{out}}$
  - Input feature dimension: D_{in}
  - Output feature dimension: D_{out}
- Kernel function:

$$g(\mathbf{y_i}) = \sum_k h(\mathbf{y_i}, \tilde{\mathbf{x}}_\mathbf{k}) \boldsymbol{W_k} \tag{1.7}$$

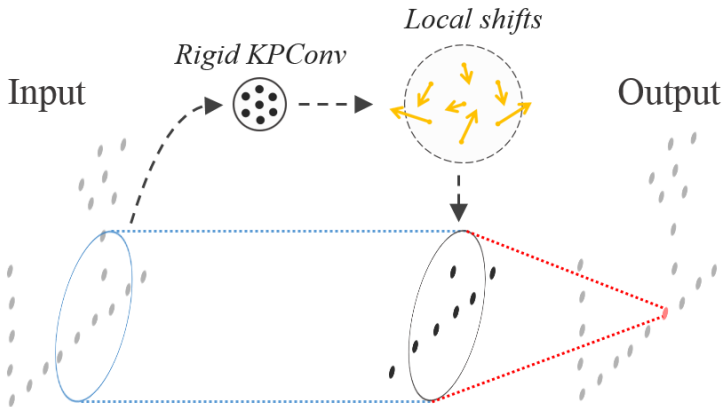
- Correlation function:

$$h(\mathbf{y_i}, \tilde{\mathbf{x}}_\mathbf{k}) = \max\left(0, 1 - \frac{\|\mathbf{y_i} - \tilde{\mathbf{x}}_\mathbf{k}\|}{\sigma}\right) \tag{1.8}$$

- Visualization of KPConv vs. regular Conv layer



- Adapt point locations per point
- Use a fixed point location KPConv layer to predict offsets

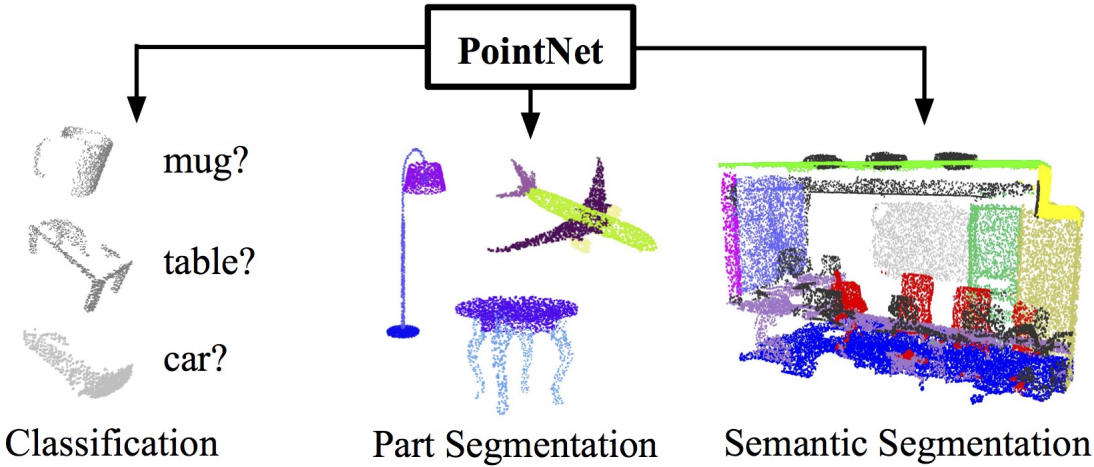


1.9 Point Cloud Applications

1.9.1 Classification and Segmentation

PointNet (CVPR 2017)

- Classification
  - Output - Global max pooling
  - Loss function - Cross Entropy
- Segmentation
  - Output - Shared MLP over all points
  - Loss Function - Per-point Cross Entropy


PointNet
mug?
table?
car?
Classification
Part Segmentation
Semantic Segmentation

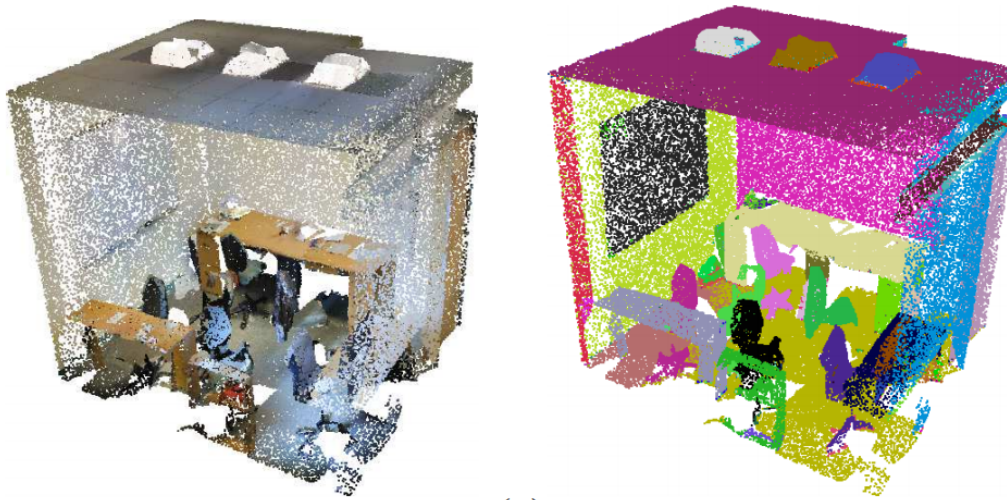
PointNet++ (NIPS 2017)

- Classification
  - Output - Global max pooling
  - Loss function - Cross Entropy
- Segmentation
  - Architecture - U-Net-like structure
  - Upsampling - Interpolation with inverse distance weighted average based on k nearest neighbors
  - Output - Shared MLP over all points
  - Loss Function - Per-point Cross Entropy

1.9.2 Instance Segmentation

SGPN (CVPR 2018)

- Goal - Infer per-point class labels and instance IDs at the same time
- Backbone - PointNet and PointNet++
- Idea - Pairwise similarity
  - Compute an embedding of all points
  - Cluster in embedding space using Euclidean distance

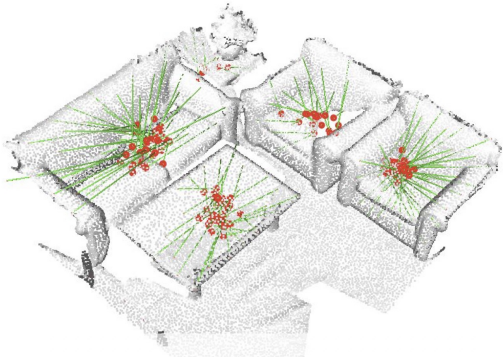


(a)

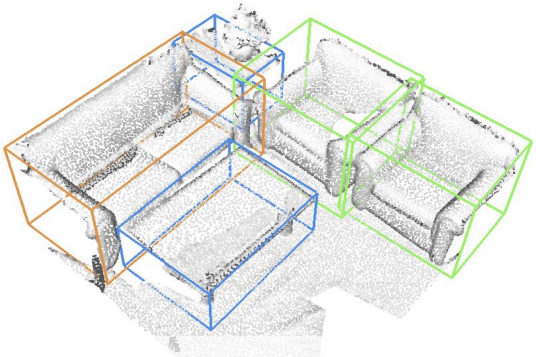
1.9.3 Detection

Deep Hough Voting (ICCV 2019)

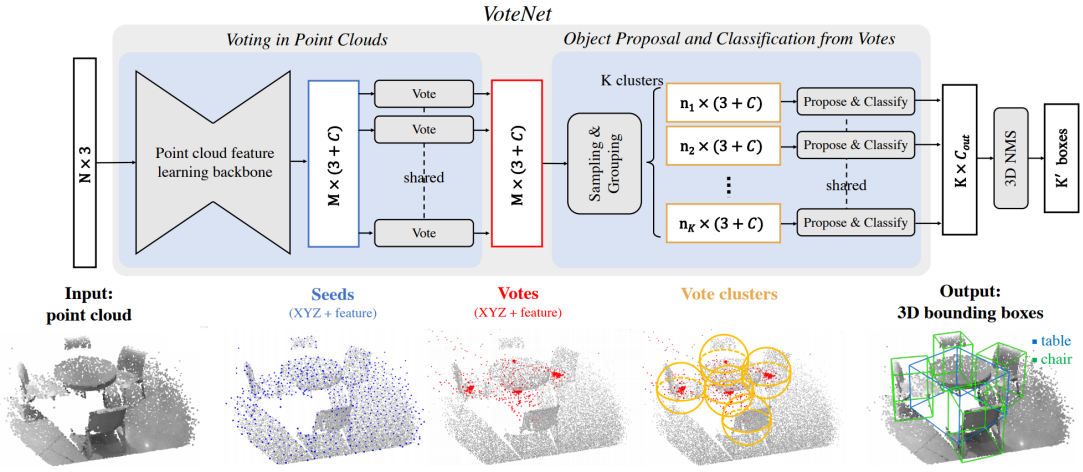
- Backbone - PointNet++
- Architecture
  - Subsample seed points, a subset of M points
  - Generate M votes
  - Subsample a subset of K votes using farthest point sampling and group them into proposals
  - Classify K proposals



Voting from input point clouds



3D object detection output

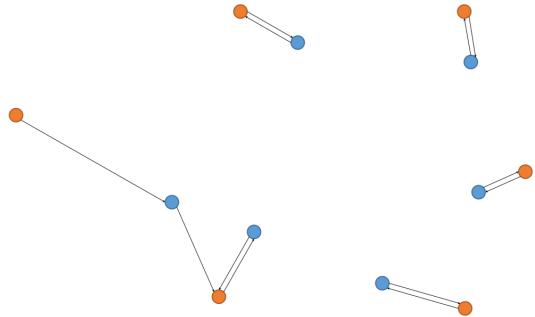

VoteNet
Voting in Point Clouds
Object Proposal and Classification from Votes
N × 3
Point cloud feature learning backbone
M × (3 + C)
Vote
Vote
shared
Vote
M × (3 + C)
K clusters
Sampling & Grouping
n1 × (3 + C)
n2 × (3 + C)
nK × (3 + C)
Propose & Classify
Propose & Classify
shared
Propose & Classify
K × Cout
3D NMS
K' boxes
Input: point cloud
Seeds (XYZ + feature)
Votes (XYZ + feature)
Vote clusters
Output: 3D bounding boxes
table
chair

1.10 Generation

PointOutNet (CVPR 2017)

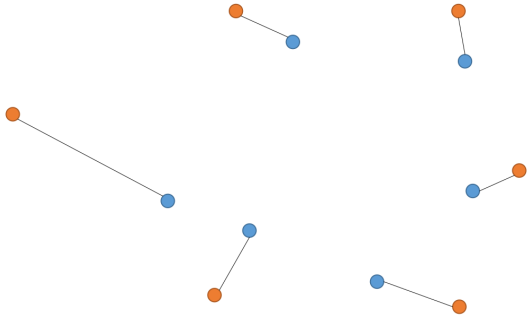
- Goal - Convert an image to its point cloud representation
- Idea - Conditional Variational Autoencoder
- Loss function
  - Chamfer Loss:

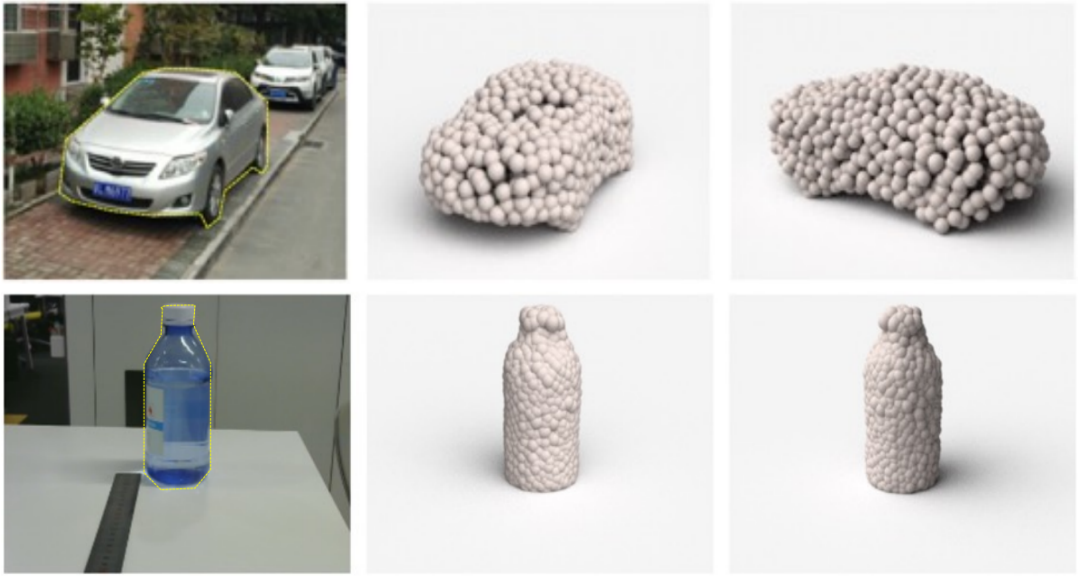
$$CD(\mathbf{X},\mathbf{Y}) = \sum_{\mathbf{x}\in\mathbf{X}} \min_{\mathbf{y}\in\mathbf{Y}} \left\|\mathbf{x}-\mathbf{y}\right\|_2^2 + \sum_{\mathbf{y}\in\mathbf{Y}} \min_{\mathbf{x}\in\mathbf{X}} \left\|\mathbf{x}-\mathbf{y}\right\|_2^2$$



- Earth mover's distance:

$$EMD(\mathbf{X},\mathbf{Y}) = \min_{\phi:\mathbf{X}\rightarrow\mathbf{Y}} \sum_{\mathbf{x}\in\mathbf{X}} \left\|\mathbf{x}-\phi(\mathbf{x})\right\|_2^2$$



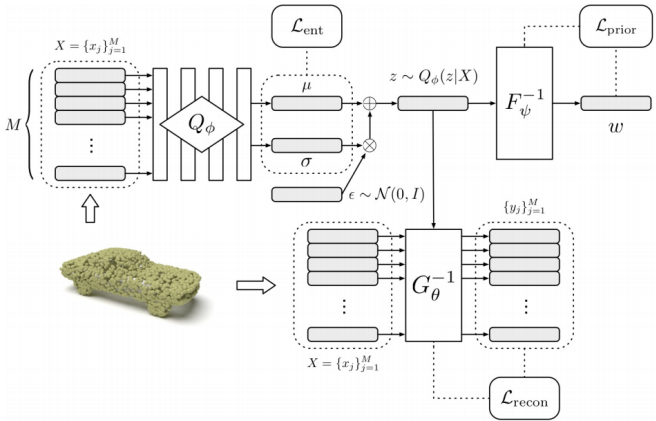


Input

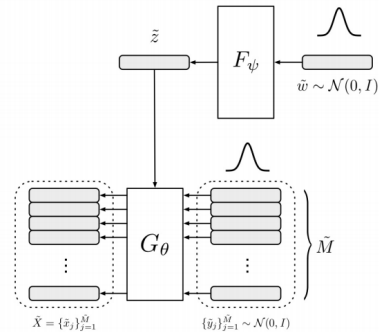
Reconstructed 3D point cloud

PointFlow (ICCV 2019)

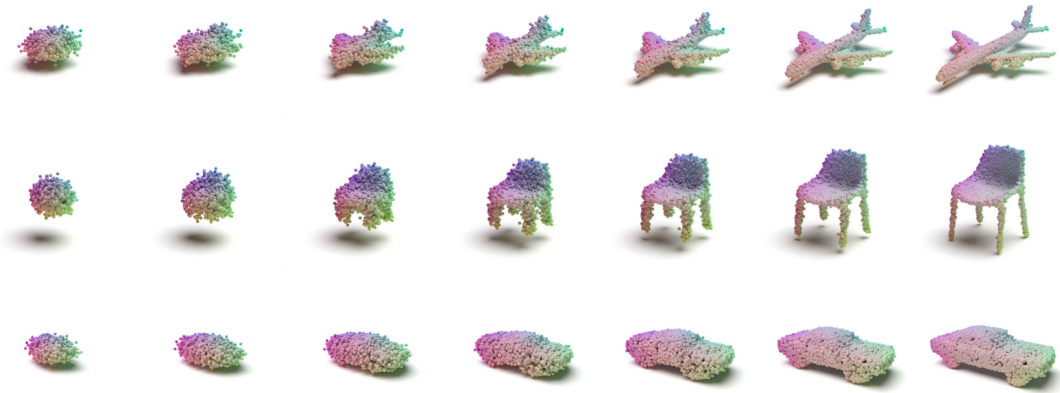
- Goal - Train a network can be used to generate point clouds
- Idea - Normalizing flows (ICML 2015)



(a) Training (Auto-encoding)



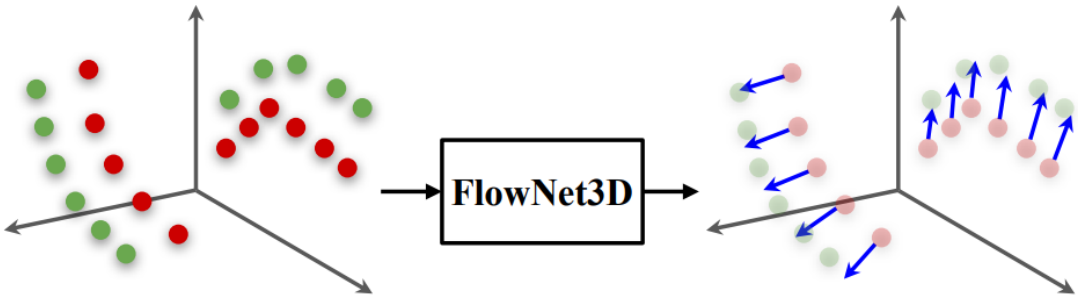
(a) Test (Sampling)

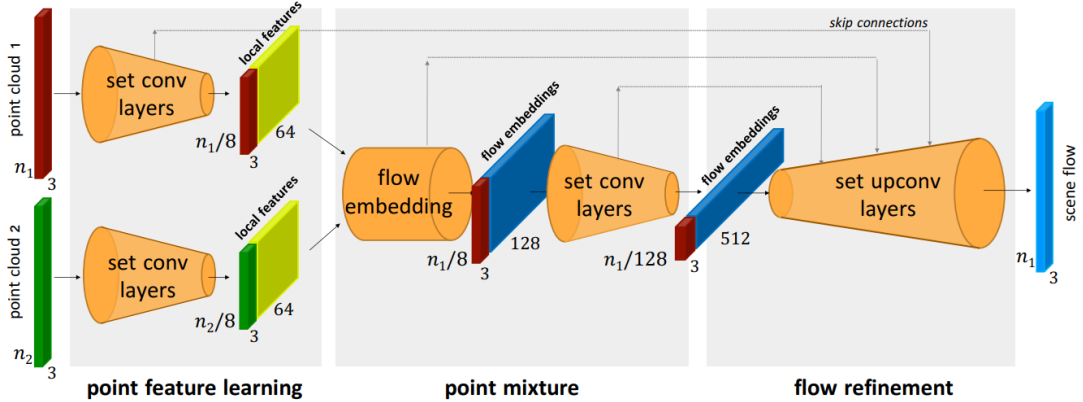


1.11 Motion Estimation

FlowNet3D (CVPR 2019)

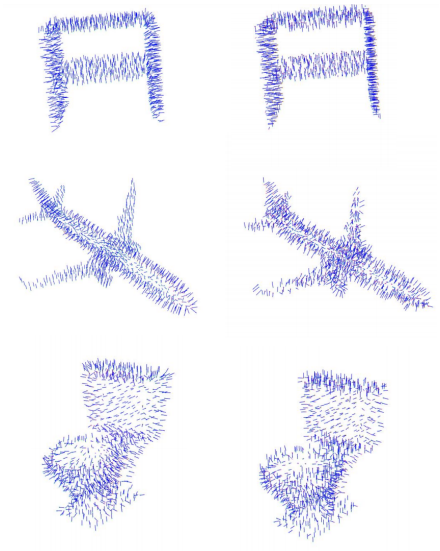
- Goal - Estimate scene flow directly from point clouds
- Backbone - PointNet++
- Input - Two frames of point clouds
- Outpu - Scene flow


FlowNet3D
point cloud 1: $n_1 \times 3$
point cloud 2: $n_2 \times 3$
scene flow: $n_1 \times 3$


point cloud 1
n_1
3
point cloud 2
n_2
3
set conv layers
set conv layers
local features
$n_1/8$
64
3
local features
$n_2/8$
64
3
point feature learning
flow embedding
$n_1/8$
flow embeddings
128
3
set conv layers
$n_1/128$
flow embeddings
512
3
point mixture
skip connections
set upconv layers
n_1
3
scene flow
flow refinement

1.11.1 Normal Estimation

- Predict a normal vector for each point



Prediction Ground-truth