CMSC740
Advanced Computer Graphics

Matthias Zwicker
Fall 2025

Deep learning for geometry processing

Goal: input 3D shapes into neural networks for analysis or modeling tasks

Input 3D shape → Neural network to compute non-linear function → Output

Optimize millions of parameters via gradient descent

Application dependent

Potential operations

- Categorize, classify 3D shapes
- Segment 3D shapes into parts
- Retrieval from databases based on example, text
- Synthesis of new shapes similar to shapes in database
  - Based on suitable low-dimensional shape representation
- Completion of partial data (RGB(D) image(s) to 3D shapes)
- Interactive editing, modeling, deformation (e.g., sketch-based)
- Multi-modal modeling (shapes, text, images; transcoding)

Questions/challenges

- Which 3D shape representation to use as inputs to neural networks?
  - Meshes
  - Point clouds
  - Implicit function on discrete 3D grid (e.g. distance function)
  - Binary values (inside/outside) on 3D grids
  - 2D parameterization (flatten 3D shape to 2D domain)
- How to design neural network architectures to operate on 3D shapes?
- Challenges?
  - 2D surfaces embedded in 3D
  - Non-uniformly sampled

First: supervised learning

Input 3D shape → Neural network to compute non-linear function → Output

- Supervised learning https://en.wikipedia.org/wiki/Supervised_learning
  - Known input-output pairs: assume each shape comes with known ground truth output, called label(s), (per-shape or per-vertex labels)
  - Labels often manually annotated (e.g. shape class, part segmentation, natural language description, etc.)
  - Objective is to optimize network weights to make output as similar as possible to given labels
  - Difference between network output and desired labels evaluated using loss function

Deep learning for shape analysis

Neural network architectures for 3D shapes, three popular examples

- Convolutional networks on 3D shapes
- Pointnet
- Transformers

Note: just a few select example techniques here, many more in recent literature

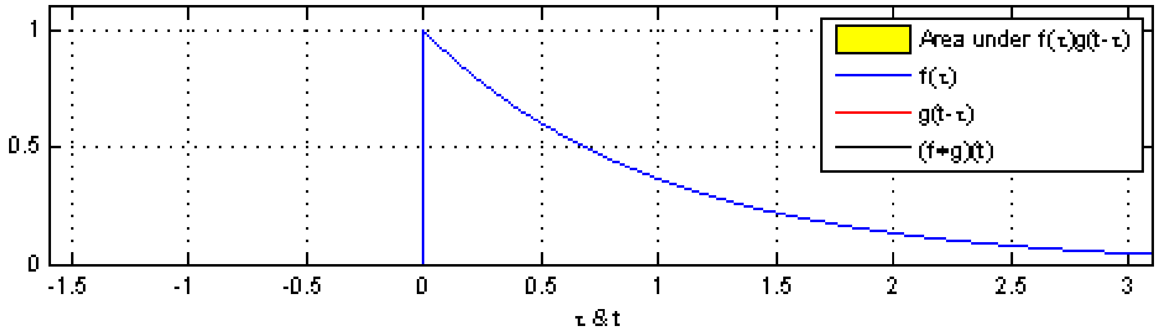
Generalizing Convolutions to 3D Shape Representations

Convolution https://en.wikipedia.org/wiki/Convolution

- 1D continuous example

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$

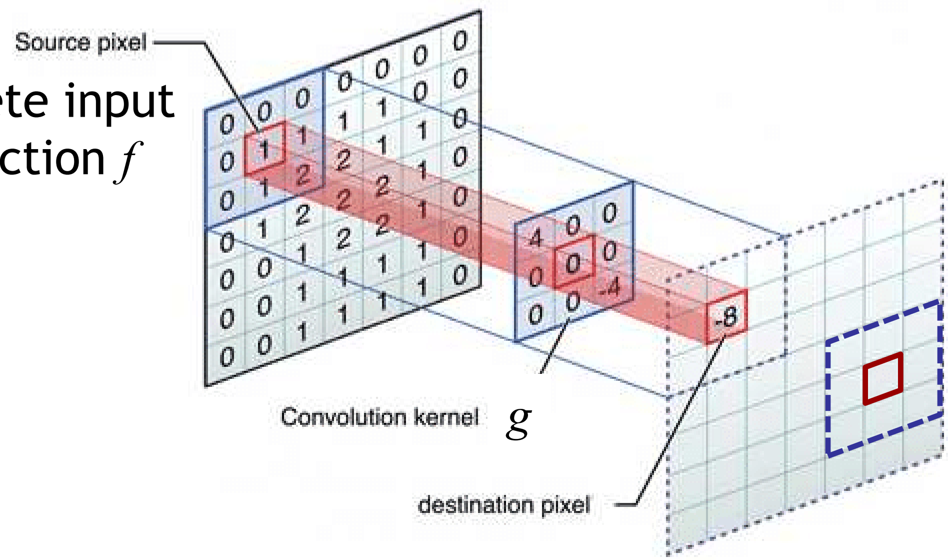
  - Input function $f\colon \mathbf{R} \to \mathbf{R}$
  - Convolution kernel $g\colon \mathbf{R} \to \mathbf{R}$
  - Output is again a function $(f * g)\colon \mathbf{R} \to \mathbf{R}$



Discrete 2D convolution

- Summation instead of integration

Discrete input function f



Shift kernel g to each pixel

Output value $(f * g)$ via elementwise product and summation of f and g at each destination pixel

Relation to convolutional neural networks (CNNs): in CNNs values in convolution kernel g represent network weights that will be trained based on given loss function; result of convolution is input into non-linear activation function

Properties

Assuming finite support kernels (convolution kernel g is non-zero within limited range)

- Sparse, linear (can be represented as multiplication with Toeplitz matrix, https://en.wikipedia.org/wiki/Toeplitz_matrix#Discrete_convolution)
  - Efficient: linear complexity for computation and storage
- Local: output at each point depends on input only over certain region (“receptive field”)
  - By stacking (concatenating) convolutions, can analyze signal at different scales (“levels of abstraction”)
- Translation equivariance (changing order of applying translation and convolution does not change output)
  - Analysis of input is independent of translation of input

Goal

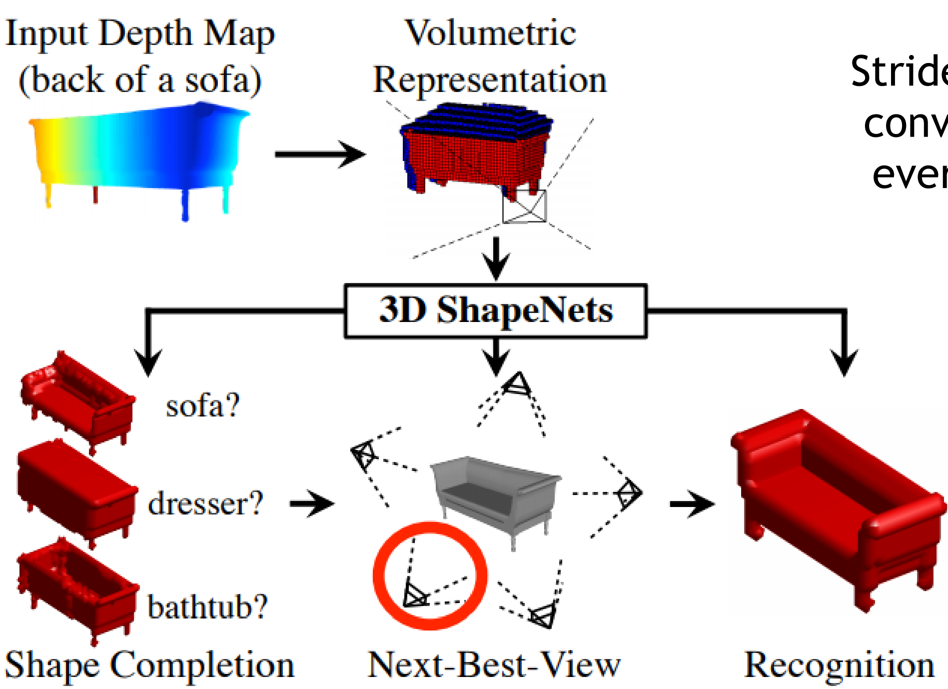
- Preserve properties for convolution on 3D shapes

3D voxel grids

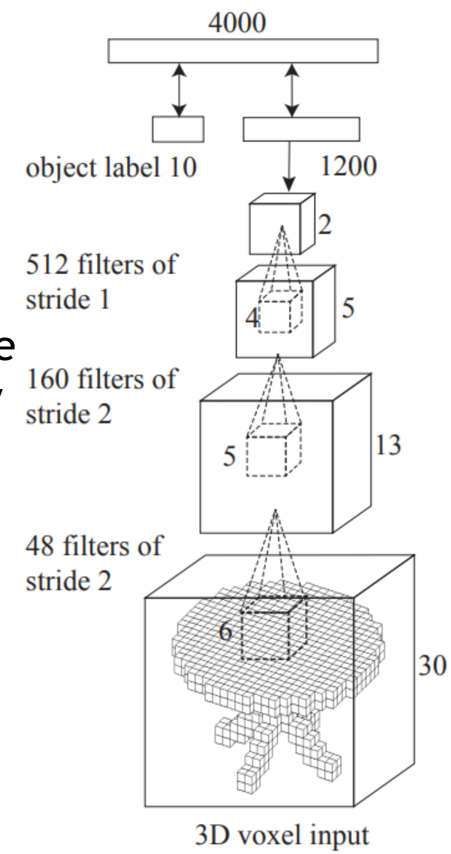
- Represent shapes discretized on 3D grids
  - Similar to raster images, but with 3D voxels instead of 2D pixels
- Alternatives
  - Binary inside/outside values per voxel
  - Implicit function
    - E.g., signed distance function
- Advantage
  - Performing convolution (in 3D) straightforward
- Disadvantage
  - Extra dimension required to represent data (surfaces are 2D, but voxel grids are 3D)
  - Memory, computation overhead
  - Workaround: hierarchical data structures (e.g., octrees, https://arxiv.org/abs/1712.01537)
  - Output depends on shape orientation (not rotation invariant)

Binary voxel grid

- 3D ShapeNets: A Deep Representation for Volumetric Shapes, CVPR 2015
- Trained as deep belief network



Stride n: evaluate convolution only every-nth voxel



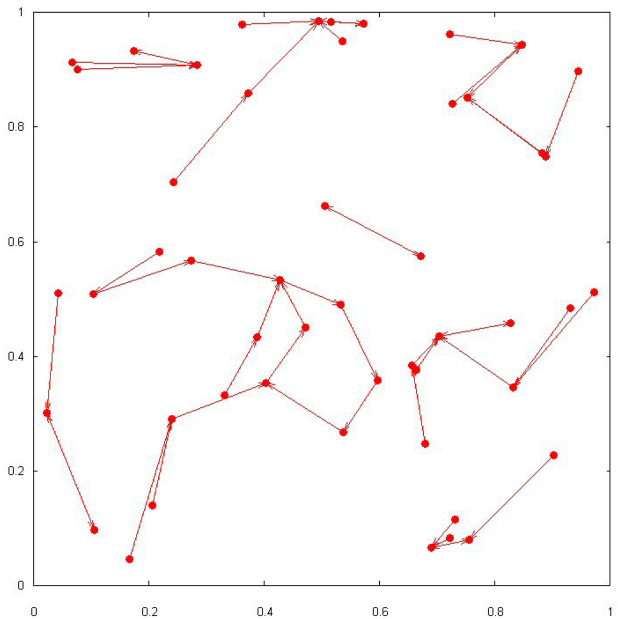
3D convolutions on uniform grid

https://3dshapenets.cs.princeton.edu/

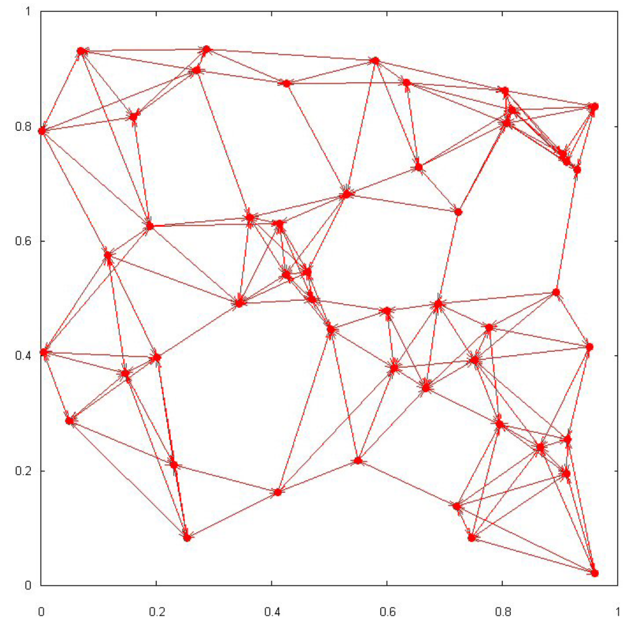
Convolutions on Point Clouds

Graph convolutional networks

- Meshes are graphs (nodes/vertices connected by edges)
- Point clouds can be represented as graphs too, using k-nearest neighbor (k-nn) graph
  https://en.wikipedia.org/wiki/Nearest_neighbor_graph



3-NN graph



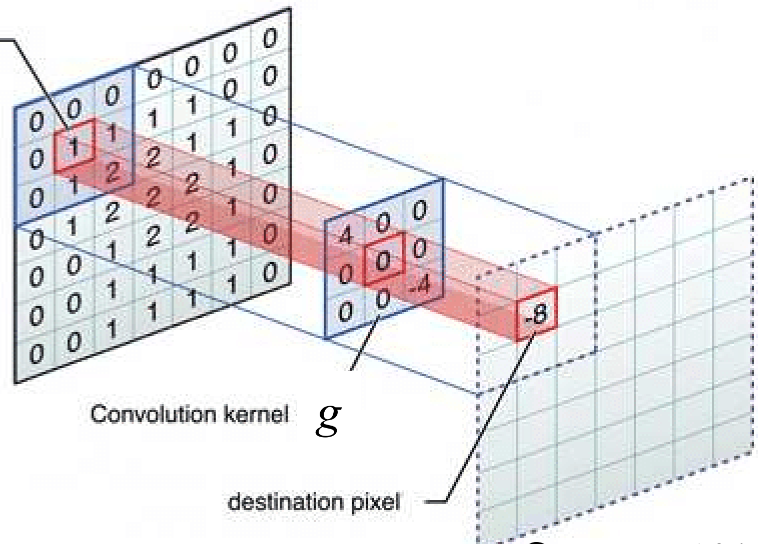
5-NN graph

http://www.maths.dur.ac.uk/users/andrew.wade/research/graphs.html

Generalize convolution to graphs

- Discrete 2D convolution

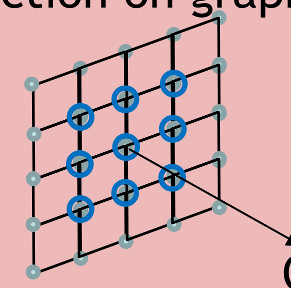
Discrete input function f



Output $(f * g)$

Interpret as operation on graph

Function f as function on graph



Output $(f * g)$

Kernel g

Each value in kernel associated with one **graph neighbor** o

Graph structure used to define convolution kernels on neighbors in graph (instead of spatial neighbors)

CNNs on point clouds

- "PointCNN: Convolution On $\mathcal{X}$-Transformed Points", Li et al., NeurIPS 2018
  https://arxiv.org/pdf/1801.07791.pdf
  - Goal: generalize convolution to irregularly sampled point clouds
- Convolution on k-nearest neighbor graph of point cloud
  - Issues with naïve approach: not clear how to associate convolution kernel values to neighbors in k-nn graph (which weight in kernel for which neighbor)
- $\mathcal{X}$-convolution: learn a transformation of k-nearest neighbors that should introduce invariance to ordering, placement of neighbors
  - Called $\mathcal{X}$-transformation matrix

$\mathcal{X}$-convolution

- Evaluation point p
- k-nearest neighbor points $\mathbf{P}$
- Features $\mathbf{F}$ of neighbor points
- Learnable $\mathcal{X}$-transformation matrix $\mathcal{X}$
- Learnable (discrete) convolution kernel $\mathbf{K}$

ALGORITHM 1: $\mathcal{X}$-Conv Operator

Input : $\mathbf{K}$, p, $\mathbf{P}$, $\mathbf{F}$

Output : $\mathbf{F}_p$ ▷ Features "projected", or "aggregated", into representative point p

$\mathbf{P}' \leftarrow \mathbf{P} - p$ ▷ Move $\mathbf{P}$ to local coordinate system of p

$\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$ ▷ **Individually** lift each point into C_δ dimensional space

$\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$ ▷ Concatenate $\mathbf{F}_\delta$ and $\mathbf{F}$, $\mathbf{F}_*$ is a $K \times (C_\delta + C_1)$ matrix

$\mathcal{X} \leftarrow MLP(\mathbf{P}')$ ▷ Learn the $K \times K$ $\mathcal{X}$-transformation matrix

$\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F}_*$ ▷ Weight and permute $\mathbf{F}_*$ with the learnt $\mathcal{X}$

$\mathbf{F}_p \leftarrow \mathrm{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$ ▷ Finally, typical convolution between $\mathbf{K}$ and $\mathbf{F}_\mathcal{X}$

https://arxiv.org/pdf/1801.07791.pdf

$\mathcal{X}$-convolution

- Shape classification results, ShapeNet40

	ModelNet40				ScanNet	
	Pre-aligned		Unaligned			
	mA	OA	mA	OA	mA	OA
Flex-Convolution [14]	-	90.2	-	-	-	-
KCNet [42]	-	91	-	-	-	-
Kd-Net [22]	88.5	90.6 (91.8 w/ P32768)	-	-	-	-
SO-Net [27]	-	90.7 (93.4 w/ PN5000)	-	-	-	-
3DmFV-Net [4]	-	91.4 (91.6 w/ P2048)	-	-	-	-
PCNN [3]	-	92.3	-	-	-	-
PointNet [33]	-	-	86.2	89.2	-	-
PointNet++ [35]	-	-	-	90.7 (91.9 w/ PN5000)	-	76.1
SpecGCN [46]	-	-	-	91.5 (92.1 w/ PN2048)	-	-
SpiderCNN [53]	-	-	-	- (92.4 w/ PN1024)	-	-
DGCNN [50]	-	-	**90.2**	**92.2**	-	-
PointCNN	**88.8**	**92.5**	88.1	**92.2**	55.7	**79.7**

Ablation study

	PointCNN	w/o $\mathcal{X}$	w/o $\mathcal{X}$-W	w/o $\mathcal{X}$-D
Core Layers	$\mathcal{X}$-Conv×4	Conv×4	Conv×4	Conv×5
# Parameter	0.6M	0.54M	0.63M	0.61M
Accuracy (%)	**92.2**	90.7	90.8	90.7

https://arxiv.org/pdf/1801.07791.pdf

Note

- Many other techniques available to apply convolutions to surfaces (point clouds, meshes)
  - Using spectral graph convolution https://arxiv.org/abs/1609.02907
  - Using diffusion on surfaces, which can represent radially geodesic convolution https://arxiv.org/abs/2012.00888
  - As a concatenation of continuous reconstruction, continuous convolution and sampling on point clouds https://dl.acm.org/doi/10.1145/3197517.3201301
- Etc.

PointNet
(for Point Cloud Processing)

PointNet https://arxiv.org/pdf/1612.00593.pdf

- Many operations on point clouds, including classification, compute set function (invariant to order of points) of the point cloud
  - Point cloud with n points $\{x_1, \ldots, x_n\}$
  - Goal: compute function $f(\{x_1, \ldots, x_n\}) =$ "class label"
- Challenges: function f should be
  - Invariant to ordering of points
  - Invariant to spatial transformation (rotation, translation) points

PointNet: order invariance

- Key idea: use approximation

$$f(\{x_1, \ldots, x_n\}) \approx g(h(x_1), \ldots, h(x_n))$$

  where

  - h is function of single point to higher dimensional feature vector (dimensionality of points N=3, K>>N)

$$h : \mathbb{R}^N \rightarrow \mathbb{R}^K$$

  - g is symmetric function (invariant to order of n input features) https://en.wikipedia.org/wiki/Symmetric_function

$$g : \underbrace{\mathbb{R}^K \times \cdots \times \mathbb{R}^K}_{n} \rightarrow \mathbb{R}$$

PointNet: order invariance

- Proof (see paper): arbitrary continuous set function f can be approximated to any accuracy ε if K is sufficiently large

$$\left| f(S) - \overbrace{\gamma\Big(\underset{x_i \in S}{MAX}}^{\text{Symmetric function } g} \{h(x_i)\}\Big) \right| < \epsilon$$

Arbitrary continuous set function

Function $\mathbf{R}^K \to \mathbf{R}$

Maximum of each feature element over all points

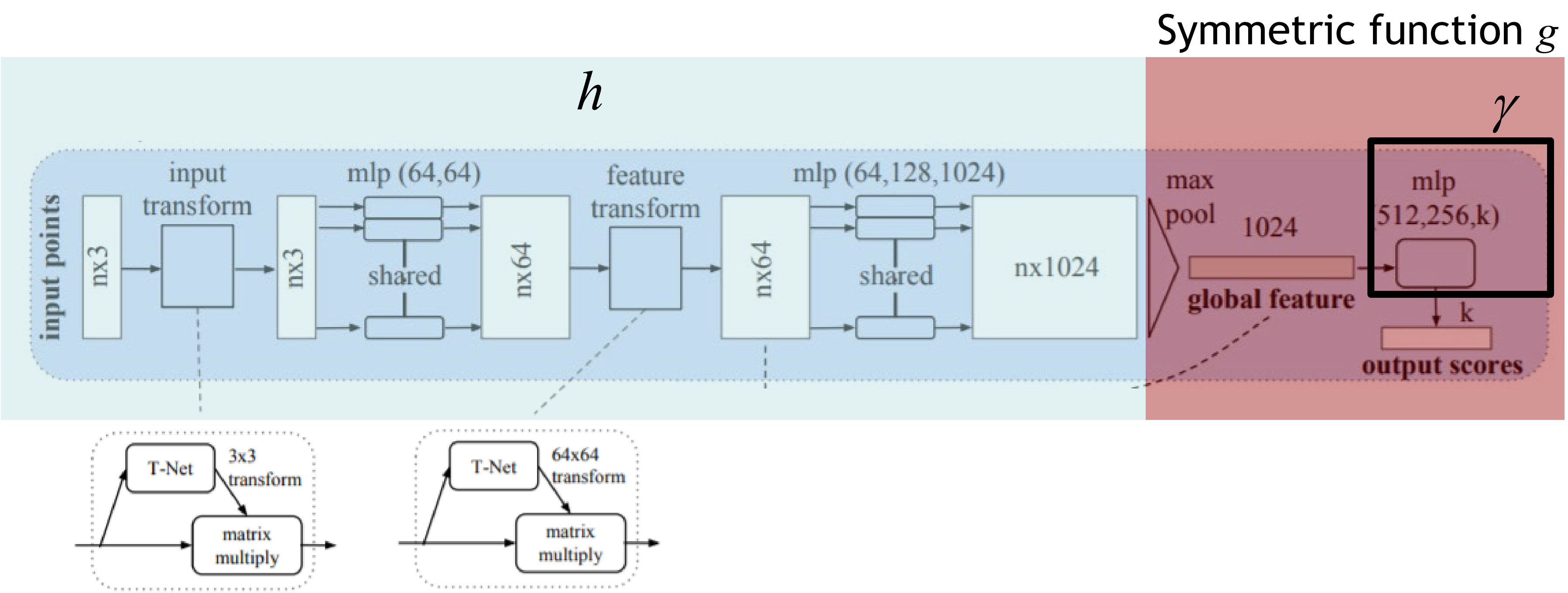
K-dim. feature vector of input point x_i

PointNet: invariance to spatial transformation

- Train a spatial transformer network
  https://arxiv.org/abs/1506.02025
  - Neural network that produces spatial transformation as its output
  - Here: just rotation
  - Different from transformers to implement attention
- Intuition: spatial transformer rotates point cloud into a “canonic” pose
  - All objects of a certain class will be oriented the same way
  - Cars: x-y is ground plane, x axis points to right of car, y to front, z up

Implementation

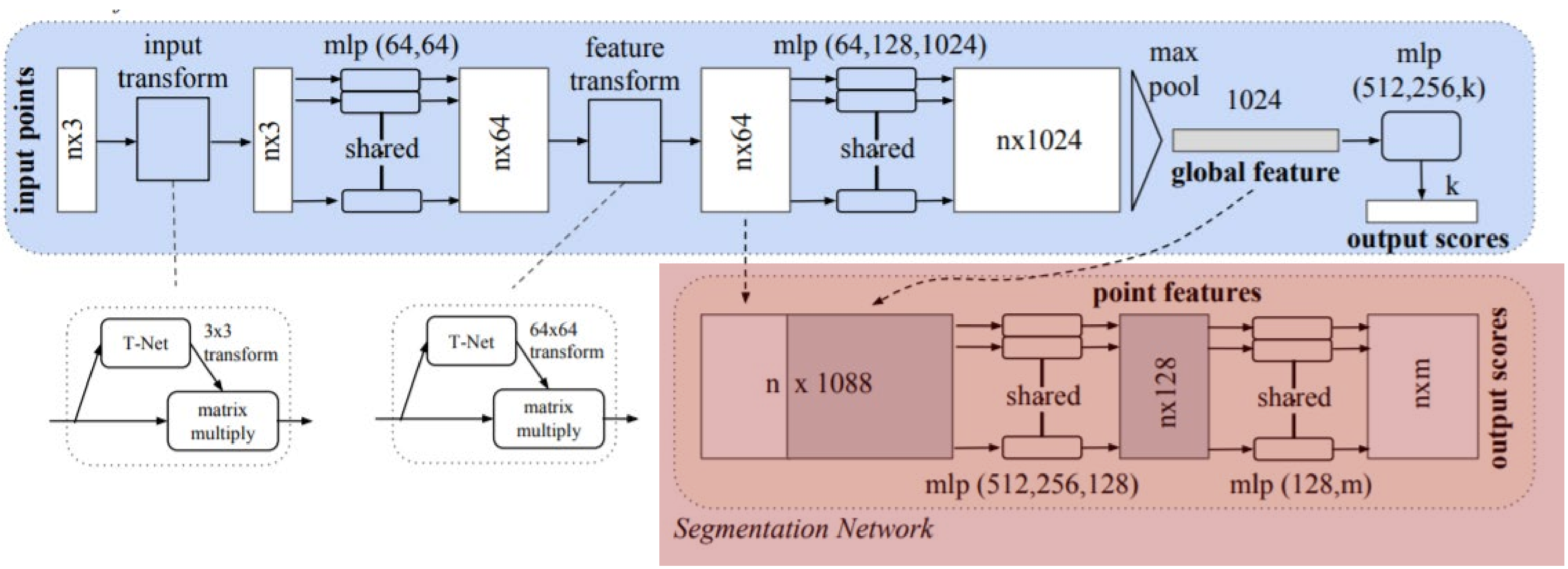
- h and γ implemented using multilayer perceptrons (MLPs), here feature dimension $K = 1024$
- Two spatial transformers (one for point cloud, one for features) for better performance



Spatial transformers

Implementation

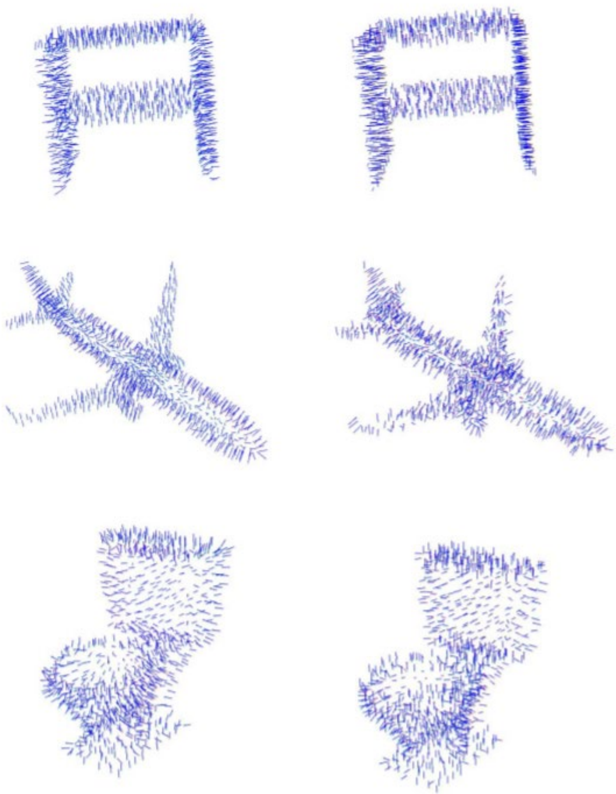
- Concatenate point features and global features to produce per-point output
  - E.g. for segmentation, normal estimation



Results

Segmentation (per-point label)

Prediction Ground truth



Normal estimation

CNNs vs PointNet

Aspect	PointNet	CNNs (on voxels)
Input Representation	Raw points	Voxel grid / 2D projections
Data Efficiency	High (uses fewer points)	Low (dense grid, many empty voxels)
Permutation Invariance	Yes	No (grid-structured input)
Spatial Locality	Weak (no local neighborhoods)	Strong (local receptive fields)
Computation	Efficient (sparse ops)	Expensive (dense convolutions)
Memory Usage	Low	High
Detail Capture	Global only	Good for local structures

Notes:

- PointNet++ includes hierarchy/spatial locality https://github.com/charlesq34/pointnet2
- CNN architectures can be built without voxelization (e.g. PointCNN, see earlier slides)

Transformers
(for 3D Point Cloud Processing)

Point transformer

- "Point Transformer", ICCV 2021
  https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer
- Transformer architecture applied to point clouds
  - "Attention mechanism" to differentially weight different parts of data
  - First developed for natural language processing (NLP) https://arxiv.org/abs/1706.03762

Transformers

- Architecture consisting of self-attention layers
  - Input: set of feature vectors (“tokens”) in context window
  - Output: set (usually same size) of transformed feature vectors
- Self-attention captures all pairwise relationships between features within context window
  - Computes each output feature i using weighted sum of all input features j, with pairwise weights to model contribution of input j to output i
- Advantages
  - Consider all pairwise relations in context window using learned weighting (attention: give more weight to more relevant elements in context window, ignore others)
  - Enables parallel instead of recursive processing as in previous architectures (RNN, LSTM)
  - State of the art performance for many applications
- Disadvantage: complexity quadratic in size of context window
  - Unless sparse attention techniques are used

Transformers

- NLP: word embedding vectors as input features/tokens https://arxiv.org/abs/1706.03762
  - Encoder/decoder architecture for sequence processing
- Computer vision: input tokens (features) computed for image patches instead of per pixel
  - Vision transformer https://arxiv.org/abs/2010.11929 https://en.wikipedia.org/wiki/Vision_transformer
  - Pixel-wise features intractable due to quadratic complexity
- Here: transformers for point clouds https://github.com/POSTECH-CVLab/point-transformer

Scalar dot-product attention layer

https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

- Input features x_j, output features y_i

Pairwise scalar weight for feature i, j

$$\mathbf{y}_i = \sum_{\mathbf{x}_j \in \mathcal{X}} \underbrace{\qquad\qquad\qquad}_{} \alpha(\mathbf{x}_j)$$

- Context window X

Scalar dot-product attention layer

https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

- Input features x_j, output features y_i

$$\mathbf{y}_i = \sum_{\mathbf{x}_j \in \mathcal{X}} \overbrace{\rho\left(\underbrace{\varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)}_{\substack{\text{Dot product, scalar} \\ \text{attention weight for} \\ \text{input } j \text{ on output } i}} + \delta\right)}^{\substack{\text{Pairwise scalar weight for} \\ \text{feature } i,\, j}} \alpha(\mathbf{x}_j)$$

- Context window X
- Feature transformations φ, ψ, α (linear or MLP), also called queries, keys, values
- Position encoding δ
  - Depends on i and j, e.g. $\theta(\mathbf{p}_i\text{-}\mathbf{p}_j)$, MLP θ, point positions $\mathbf{p}_i$
  - Attention layer is order independent, if not for positional encoding
- Normalization ρ (softmax)

Vector attention layer

https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

- Vector-valued relation function β (e.g. subtraction) instead of dot product

$$\mathbf{y}_i = \sum_{\mathbf{x}_j \in \mathcal{X}} \overbrace{\qquad\qquad\qquad\qquad}^{\text{Pairwise weight \textbf{vector} for feature } i,\, j} \odot \alpha(\mathbf{x}_j)$$

Element-wise multiplication

Vector attention layer

https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

- Vector-valued relation function β (e.g. subtraction) instead of dot product
- Vector-valued mapping function γ (e.g. MLP), instead of dot product

Pairwise weight **vector** for feature *i, j*

$$\mathbf{y}_i = \sum_{\mathbf{x}_j \in \mathcal{X}} \overbrace{\rho\Big(\boxed{\gamma\big(\beta(\varphi(\mathbf{x}_i), \psi(\mathbf{x}_j)\big)} + \delta\Big)}\odot \alpha(\mathbf{x}_j)$$

Vector of weights for each element in input *j* on output *i*

Element-wise multiplication

Application to point clouds

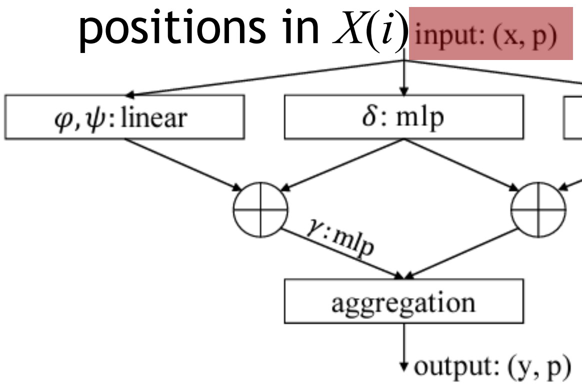
- Transformer layers as described above can be applied in many applications (NLP/LLMs, vision transformers, etc.)
- Application to point clouds
  - Here: technique described in “Point Transformer”, ICCV 2021
    https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

Point transformer layer

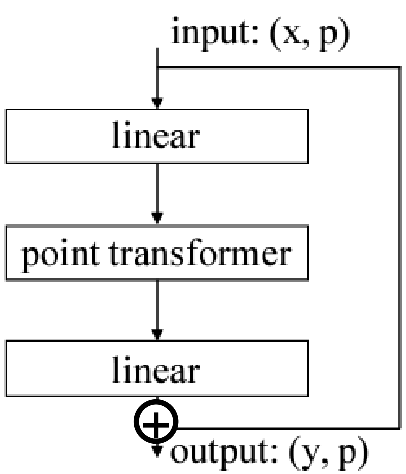
$$\mathbf{y}_i = \sum_{\mathbf{x}_j \in \mathcal{X}(i)} \rho\big(\gamma(\varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j) + \delta)\big) \odot \big(\alpha(\mathbf{x}_j) + \delta\big)$$

- Applied locally to k-nearest neighborhood $X(i)$
  - MLP mapping function γ, linear feature transformations φ, ψ, α
  - MLP position encoding $\delta = \theta(\mathbf{p}_i - \mathbf{p}_j)$

Features and positions in $X(i)$



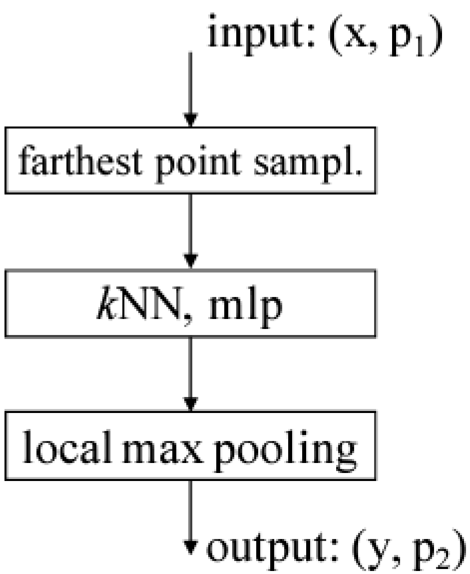
Point transformer layer



Point transformer block, linear pre-, post-processing, residual connection

Downsampling layer

- Operations to change cardinality of point set

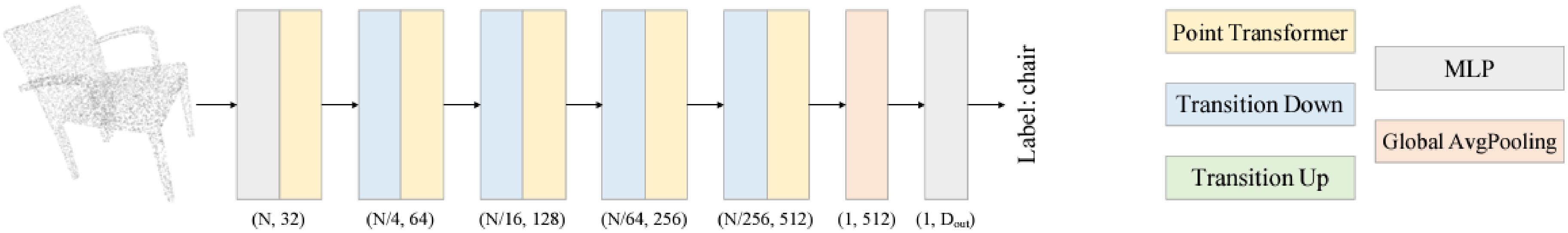


Farthest point sampling: Greedy algorithm, selecting next sample as input point farthest from existing samples, until desired cardinality reached

Downsampling pointset p_1 to pointset p_2
Feature aggregation in *knn*-neighborhood (k=16) using elementwise local max pooling

Classification architecture

- (N,M): point set cardinality N, feature size M

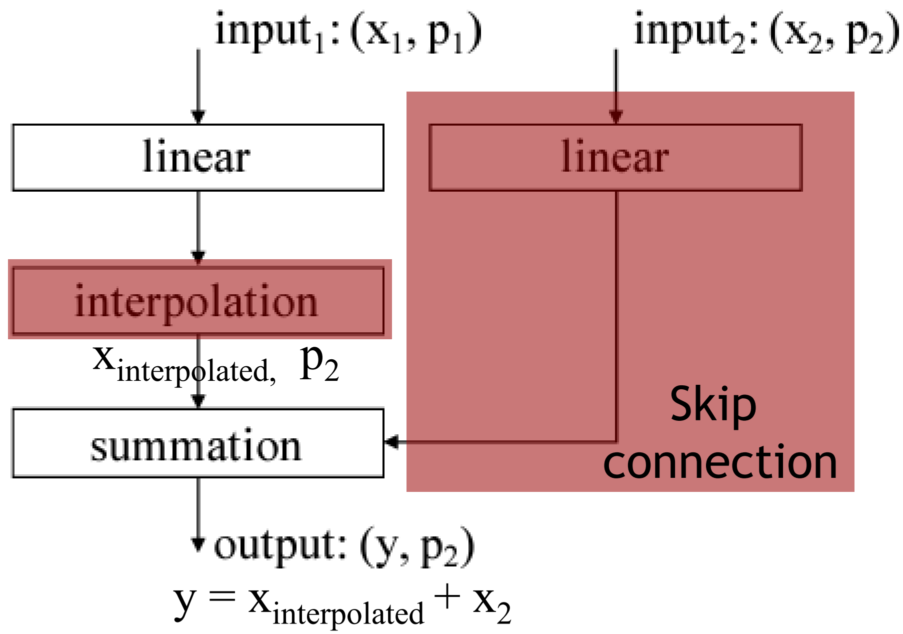


Upsampling layer

Low-res pointset p_1, features x_1

High-res pointset p_2 , features x_2

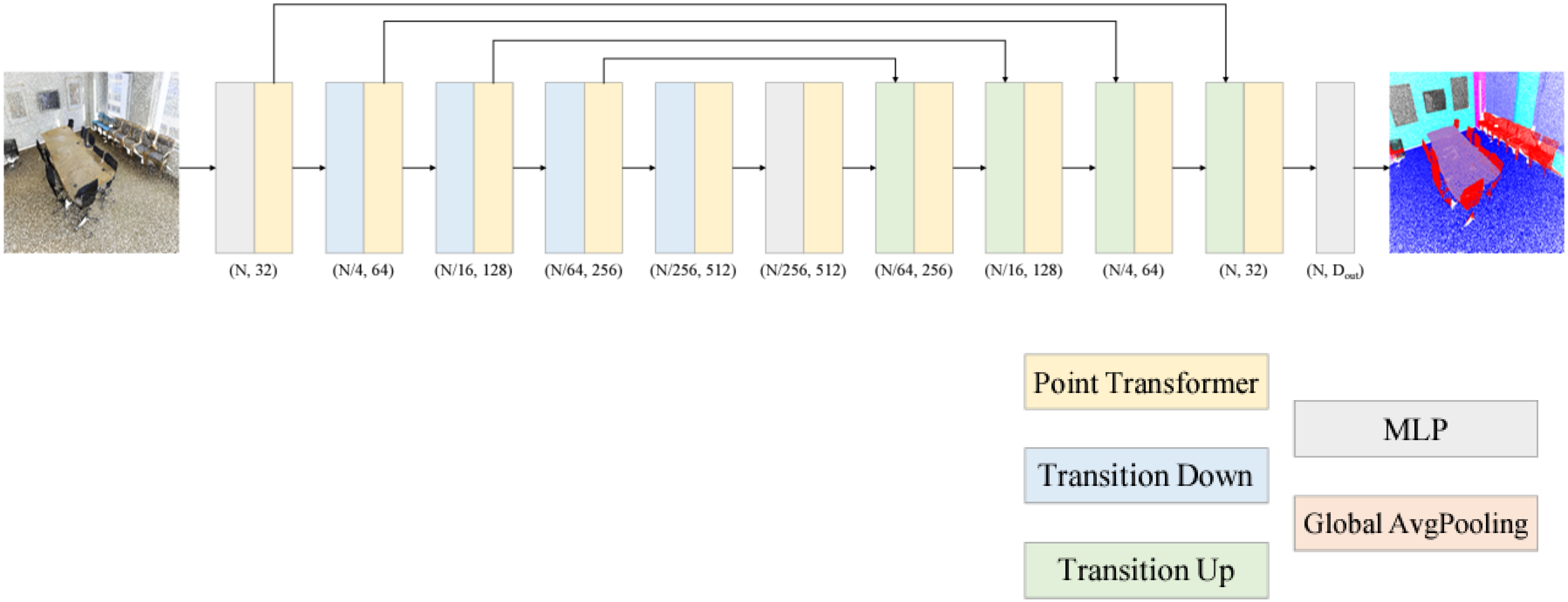
Linear interpolation of low-res features x_1 at locations of high-res points p_2



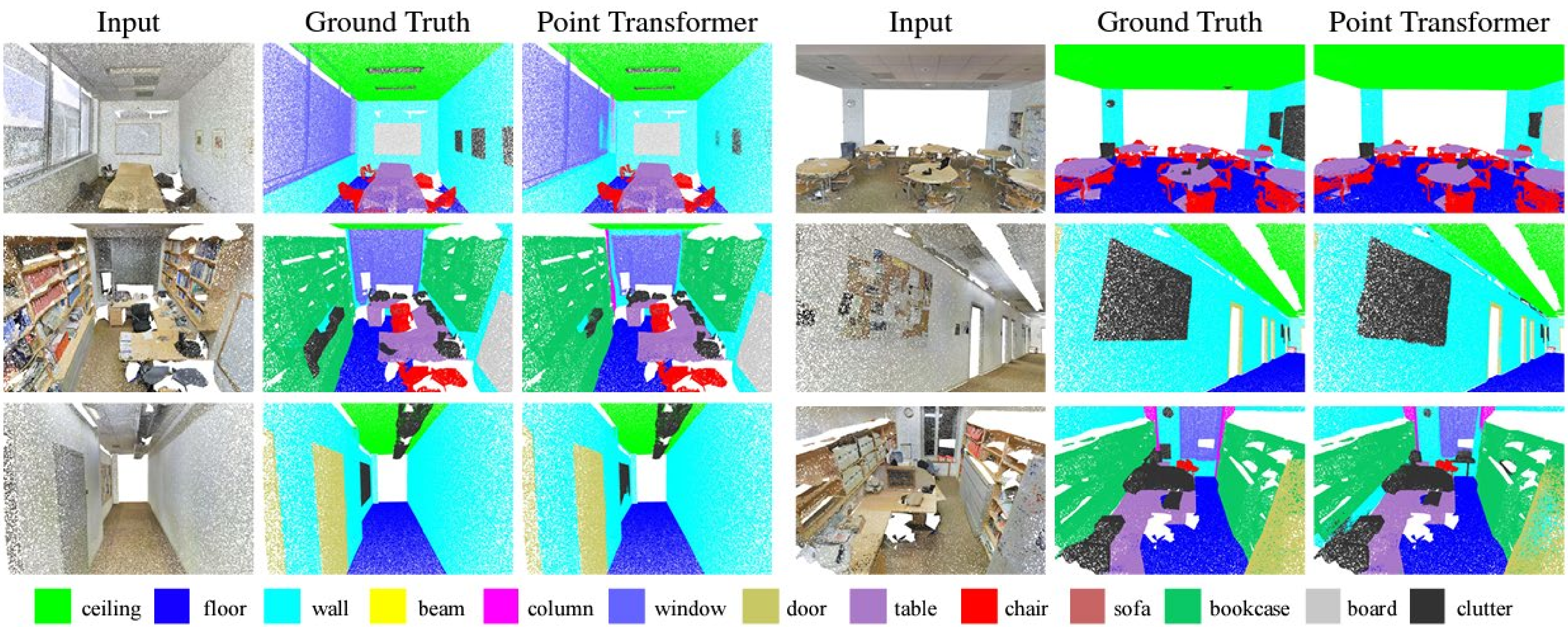
Upsampling pointset p_1 to pointset p_2
Output: features y at high-res points p_2

Semantic segmentation architecture

U-net architechture with skip connections



Semantic segmentation results



https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

Semantic segmentation results

- S3DIS dataset, 271 rooms, 13 categories
- Mean classwise intersection over union (mIoU), mean of classwise accuracy (mAcc), overall pointwise accuracy (OA)
- PointNet: MLP-based; SegCloud: voxel-based; SPGraph: graph-based, PAT: attention-based; MinkowskiNet: sparse convolution, KPConv: continuous convolution
- Number of parameters
  - Point Transformer: 4.9M; KPConv: 14.9M; SparseConv: 30.1M

Method	OA	mAcc	mIoU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [25]	–	49.0	41.1	88.8	97.3	69.8	0.1	3.9	46.3	10.8	59.0	52.6	5.9	40.3	26.4	33.2
SegCloud [36]	–	57.4	48.9	90.1	96.1	69.9	0.0	18.4	38.4	23.1	70.4	75.9	40.9	58.4	13.0	41.6
TangentConv [35]	–	62.2	52.6	90.5	97.7	74.0	0.0	20.7	39.0	31.3	77.5	69.4	57.3	38.5	48.8	39.8
PointCNN [20]	85.9	63.9	57.3	92.3	98.2	79.4	0.0	17.6	22.8	62.1	74.4	80.6	31.7	66.7	62.1	56.7
SPGraph [15]	86.4	66.5	58.0	89.4	96.9	78.1	0.0	42.8	48.9	61.6	84.7	75.4	69.8	52.6	2.1	52.2
PCCN [42]	–	67.0	58.3	92.3	96.2	75.9	0.3	6.0	69.5	63.5	66.9	65.6	47.3	68.9	59.1	46.2
PAT [50]	–	70.8	60.1	93.0	98.5	72.3	1.0	41.5	85.1	38.2	57.7	83.6	48.1	67.0	61.3	33.6
PointWeb [55]	87.0	66.6	60.3	92.0	98.5	79.4	0.0	21.1	59.7	34.8	76.3	88.3	46.9	69.3	64.9	52.5
HPEIN [13]	87.2	68.3	61.9	91.5	98.2	81.4	0.0	23.3	65.3	40.0	75.5	87.7	58.5	67.8	65.6	49.4
MinkowskiNet [37]	–	71.7	65.4	91.8	98.7	86.2	0.0	34.1	48.9	62.4	81.6	89.8	47.2	74.9	74.4	58.6
KPConv [37]	–	72.8	67.1	92.8	97.3	82.4	0.0	23.9	58.0	69.0	81.5	91.0	75.4	75.3	66.7	58.9
PointTransformer	**90.8**	**76.5**	**70.4**	94.0	98.5	86.3	0.0	38.0	63.4	74.3	89.1	82.4	74.3	80.2	76.0	59.3

https://arxiv.org/abs/2012.09164, https://github.com/POSTECH-CVLab/point-transformer

Ablation studies

k	mIoU	mAcc	OA
4	59.6	66.0	86.0
8	67.7	73.8	89.9
16	**70.4**	**76.5**	**90.8**
32	68.3	75.0	89.8
64	67.7	74.1	89.9

Table 5. Ablation study: number of neighbors k in the definition of local neighborhoods.

Operator	mIoU	mAcc	OA
MLP	61.7	68.6	87.1
MLP+pooling	63.7	71.0	87.8
scalar attention	64.6	71.9	88.4
vector attention	**70.4**	**76.5**	**90.8**

Table 7. Ablation study: form of self-attention operator.

Pos. encoding	mIoU	mAcc	OA
none	64.6	71.9	88.2
absolute	66.5	73.2	88.9
relative	**70.4**	**76.5**	**90.8**
relative for attention	67.0	73.0	89.3
relative for feature	68.7	74.4	90.4

Table 6. Ablation study: position encoding.

Transformers vs. CNNs

Concept	CNN	Transformer
Basic operation	Convolution (weighted sum using learned, but fixed convolution kernel weights)	Self-Attention (weighted sum using data-dependent attention weights)
Input structure	Grid (pixels, voxels, etc.)	Tokens (words, image patches, etc.)
Parameter sharing	Shared kernel weights	Shared feature transformations χ, ψ, α (projection matrices)
Locality	Local receptive field (convolution kernel with fixed size)	Global (any token can attend to/weight any other)
Inductive bias	Translation invariance, locality	Context learning, position-awareness via position embeddings δ
Computation	Linear in convolution kernel size	Quadratic in token count in context window (unless sparse/efficient)

Transformers pros/cons

- Pros
  - Self-attention captures long-range and local dependencies (depending on context window size)
  - Can learn complex relationships
- Cons
  - Quadratic complexity (unless sparse attention is used)
  - Large data and compute requirements for training
- Overall: most powerful architecture for many applications