Lecture 12 (Bonus track): Deep Learning for point clouds and graphs

Motivation (overview)

- Autonomous robotics is a huge frontier for AI (maybe the most impactful one?)
- Early successes: autonomous driving prototypes, delivery bots, vacuum cleaners, warehouse robots…
- 3D sensors = big boost
- Deep learning = big boost
- Deep learning + 3D data = 💙 ?

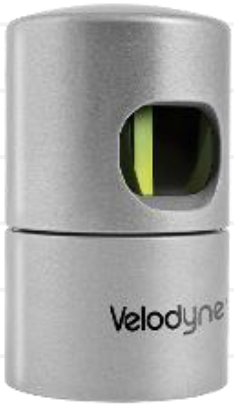
Why point clouds are important?

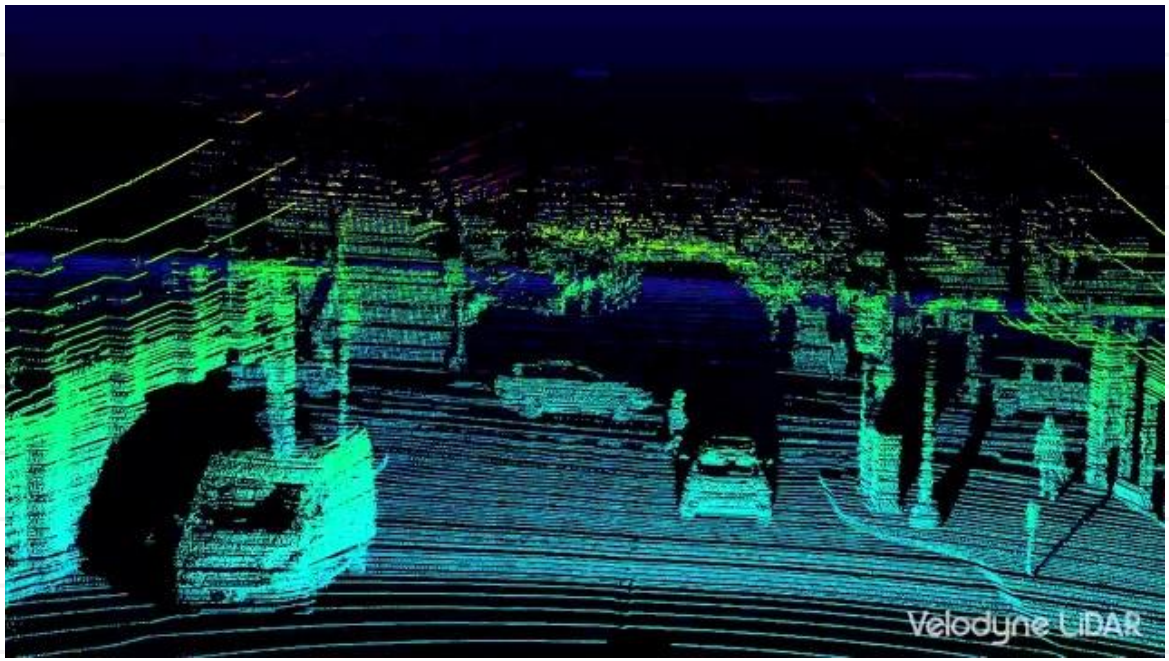
RealSense

Kinect

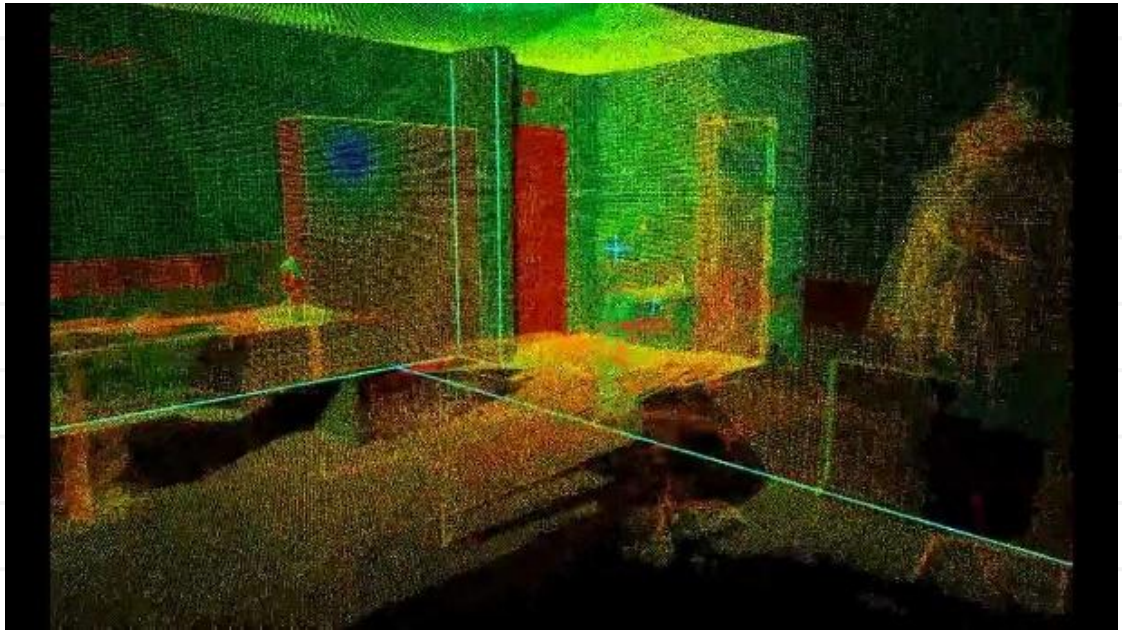
Apple scanner

Why point clouds are important?

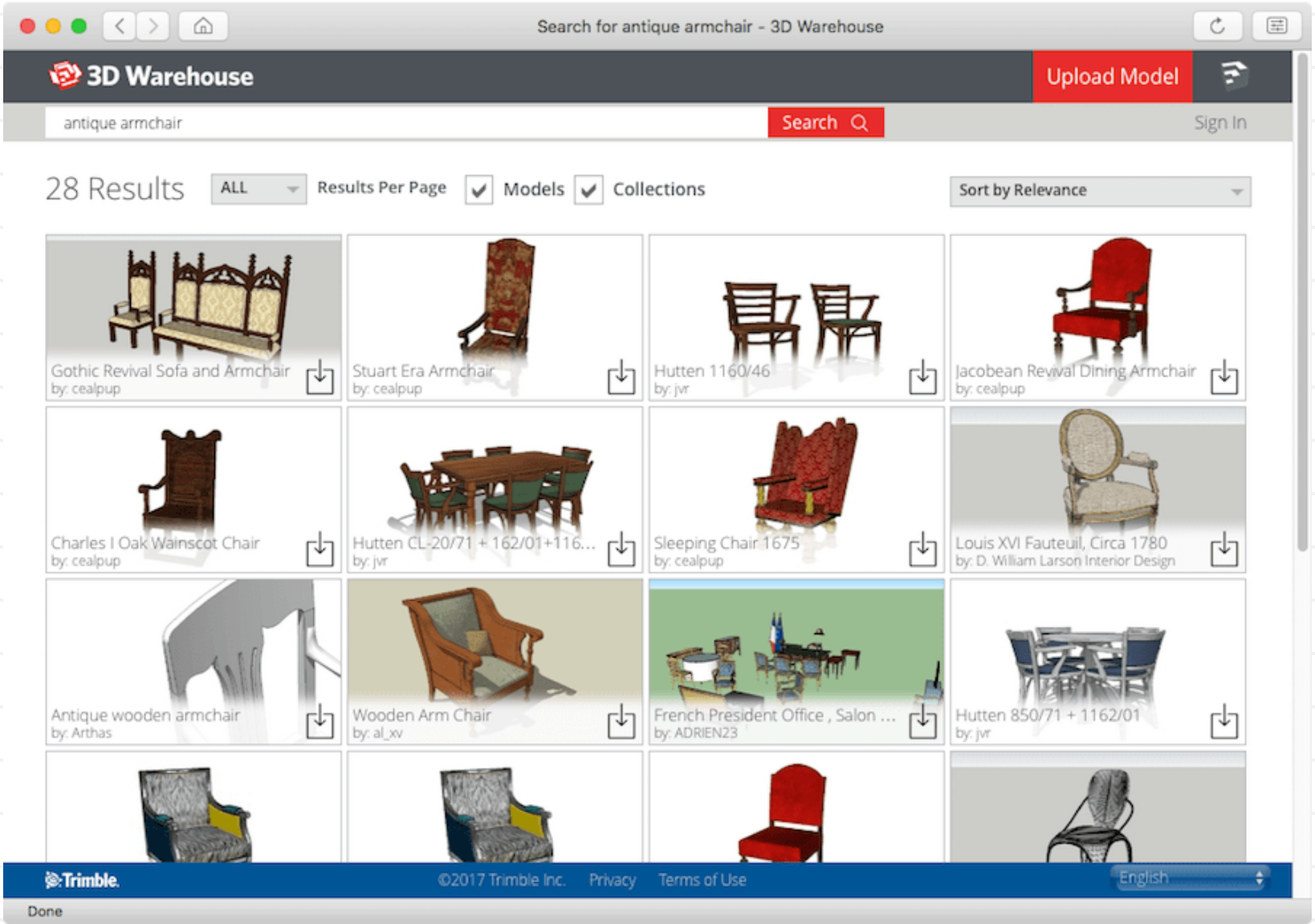




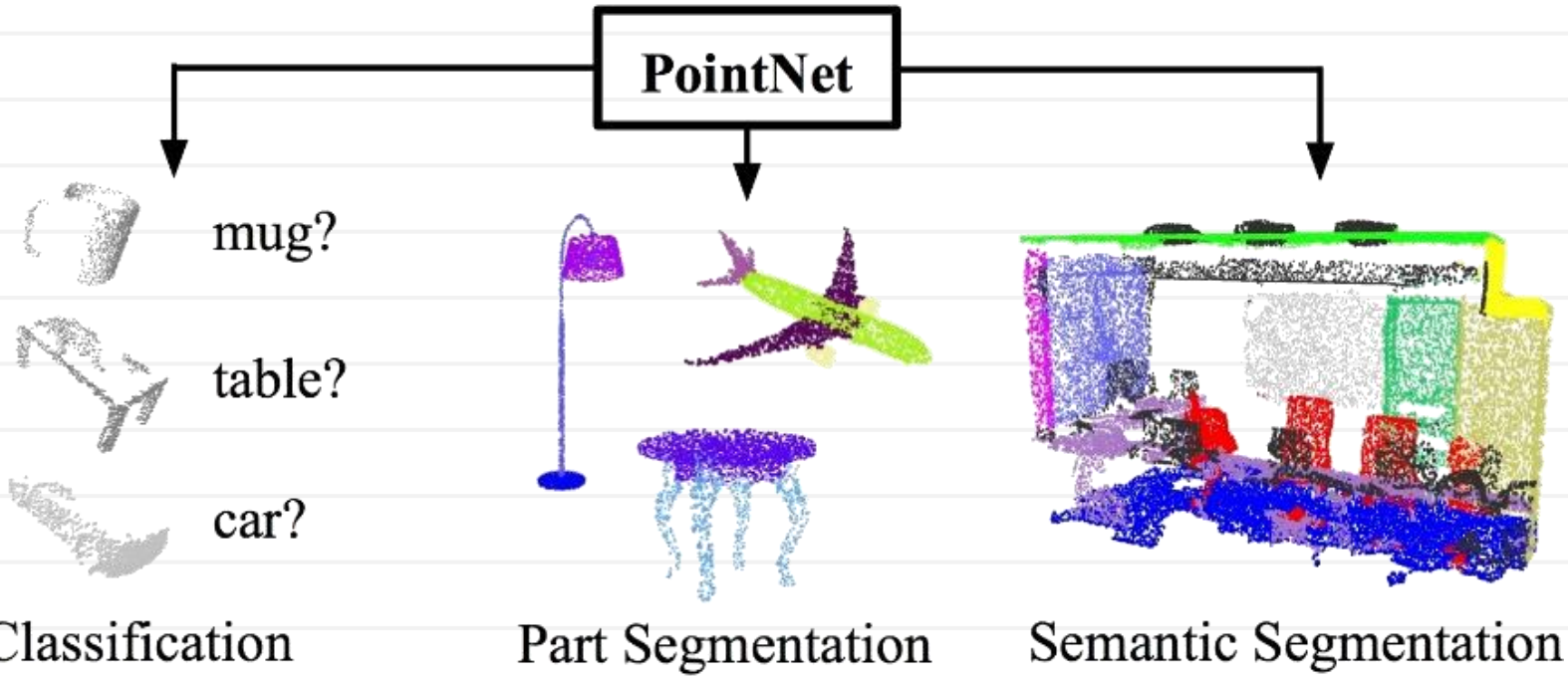
- Dominant sensor for autonomous driving
- Widely used in architecture and urban planning

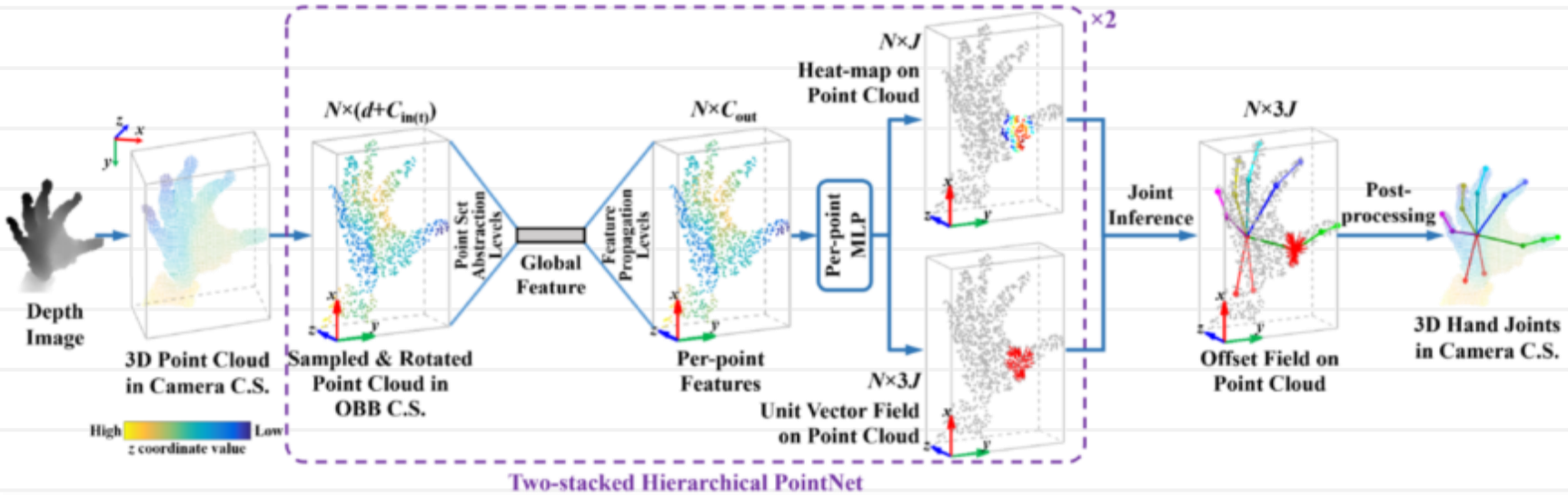


Why point clouds are important?

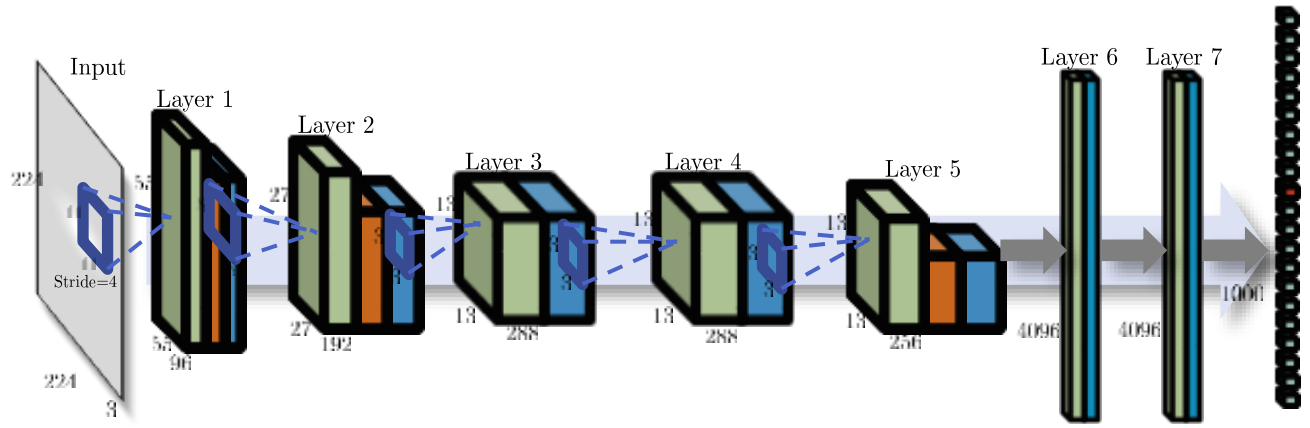


Recognition tasks



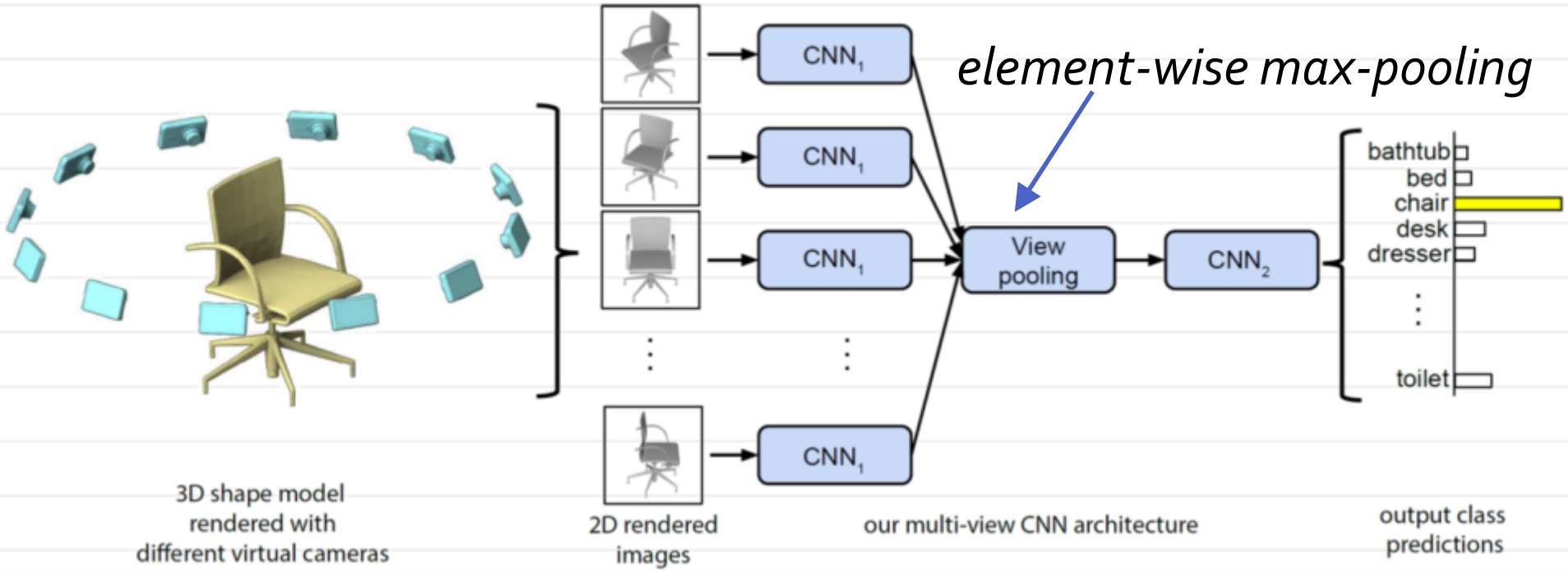


General motivation



This is ConvNet
It does recognition very well
Be like ConvNet

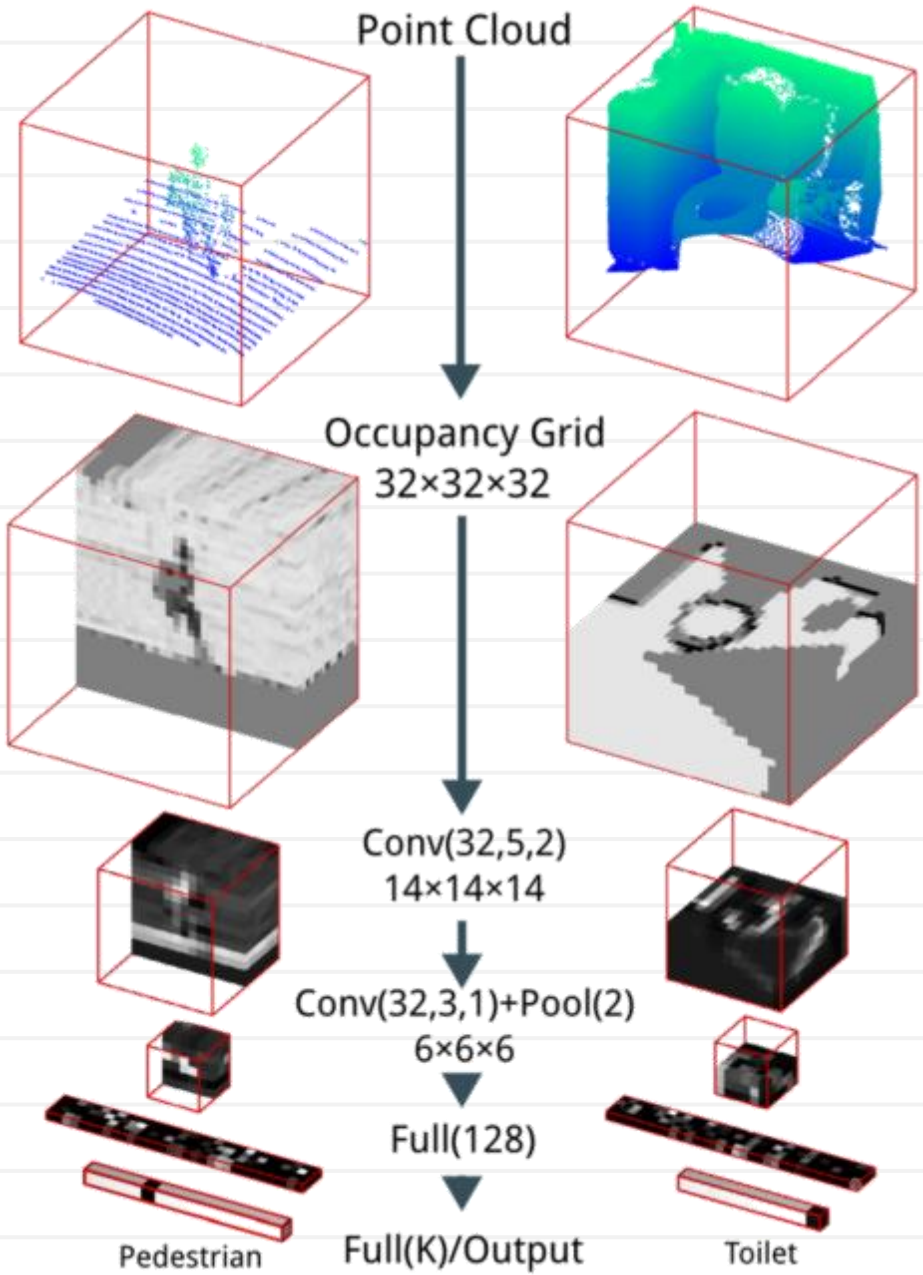
Multi-view ConvNets



- Take an object (point cloud or mesh)
- Render it to 2D images
- Apply ConvNets

[Su et al. CVPR15]

3D volumetric CNNs

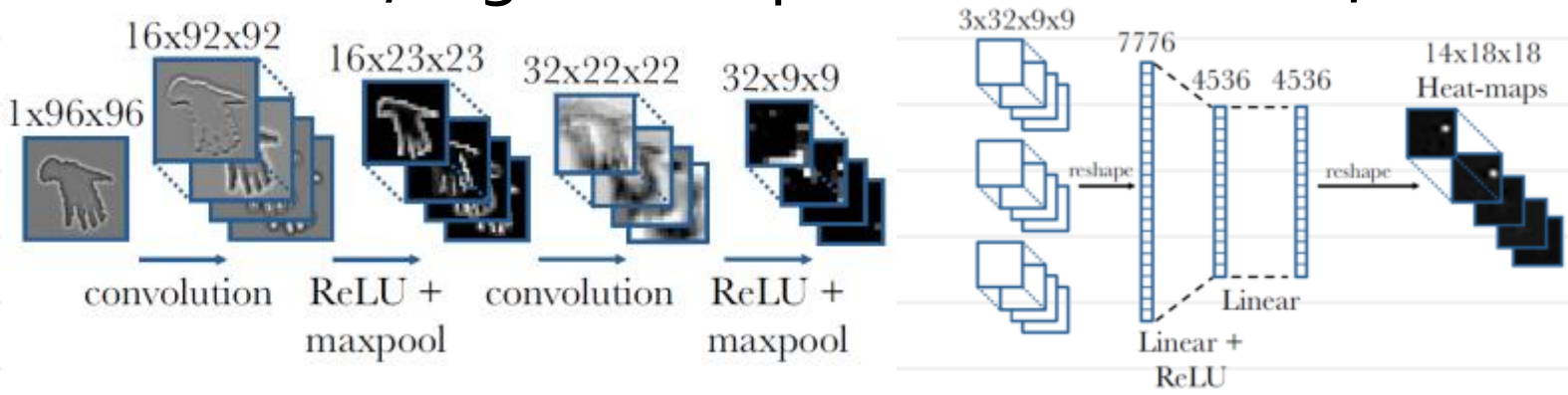


- Straightforward generalization to 3D
- Resolution is very limited

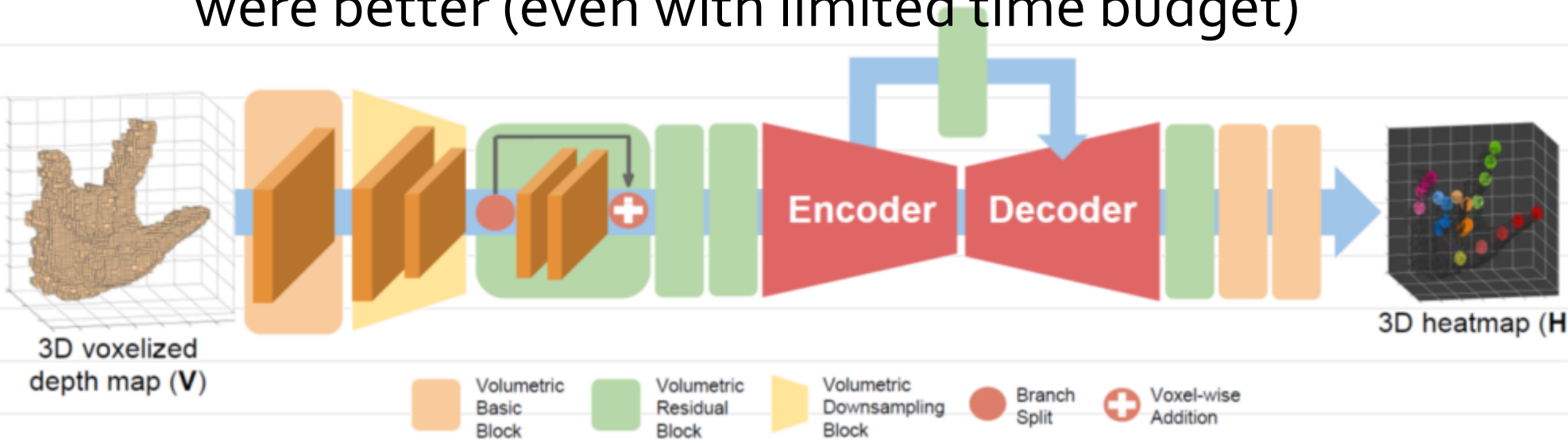
[Maturana IROS15]

V2V hand tracking

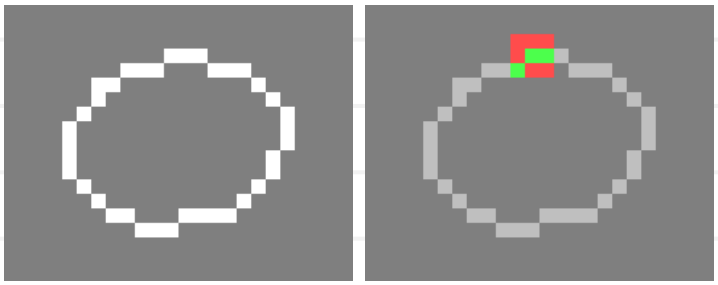
- Before RGBD->hand was handled by 2D ConvNets, e.g. [Thompson SIGGRAPH14]:



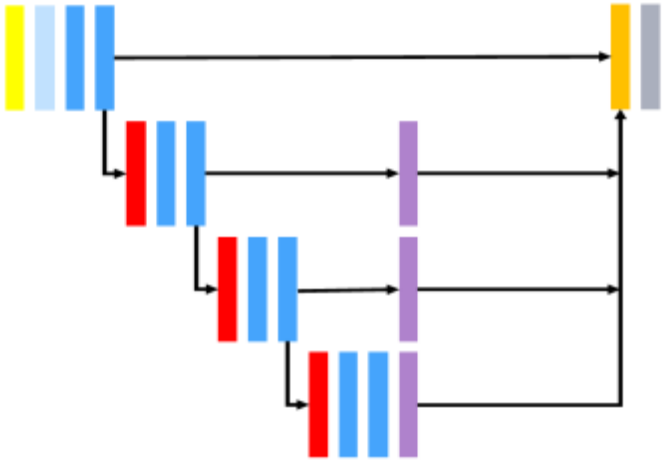
- [Moon et al. CVPR18] showed that 3D ConvNets were better (even with limited time budget)



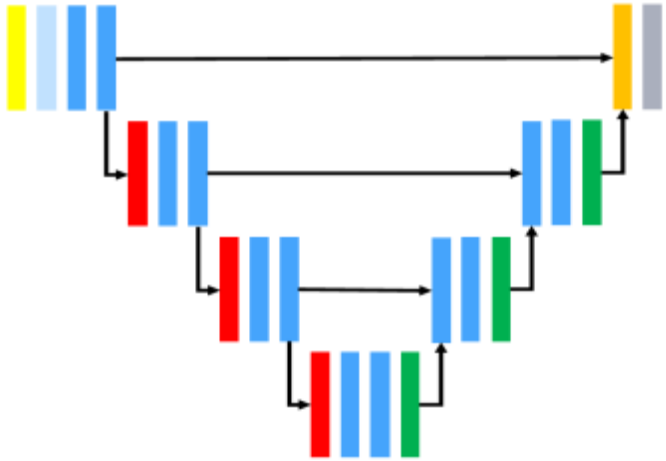
Submanifold-Sparse ConvNets



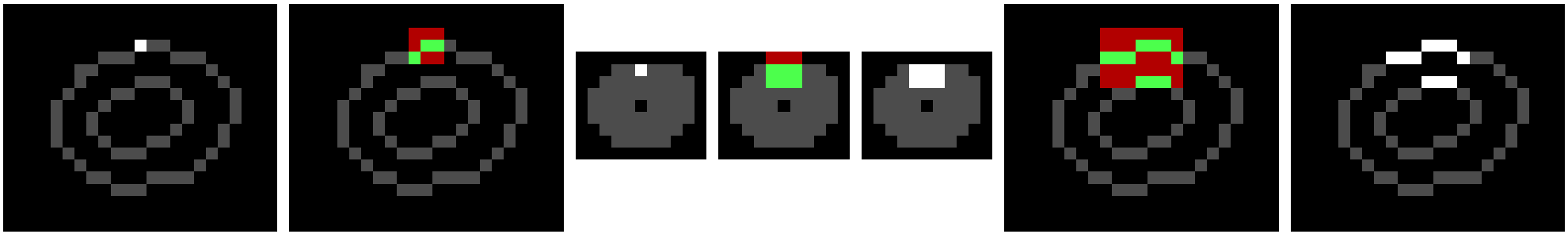
Idea: use sparse format, keep only voxels that were initially occupied



(a) Submanifold sparse FCN.



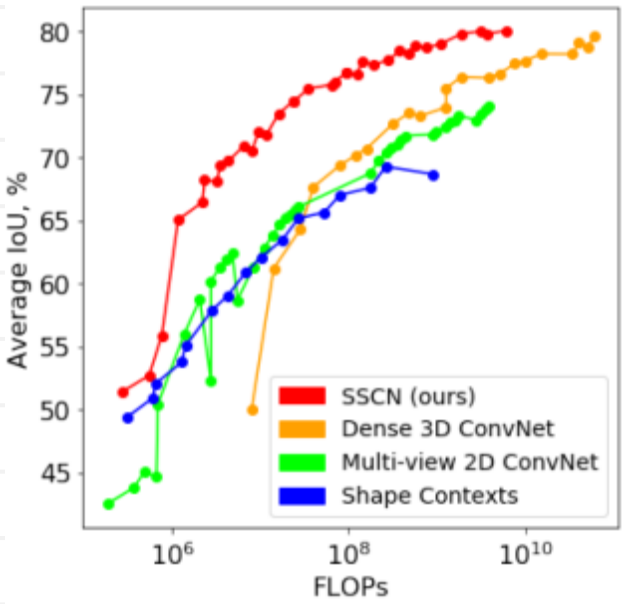
(b) Submanifold sparse U-Net.

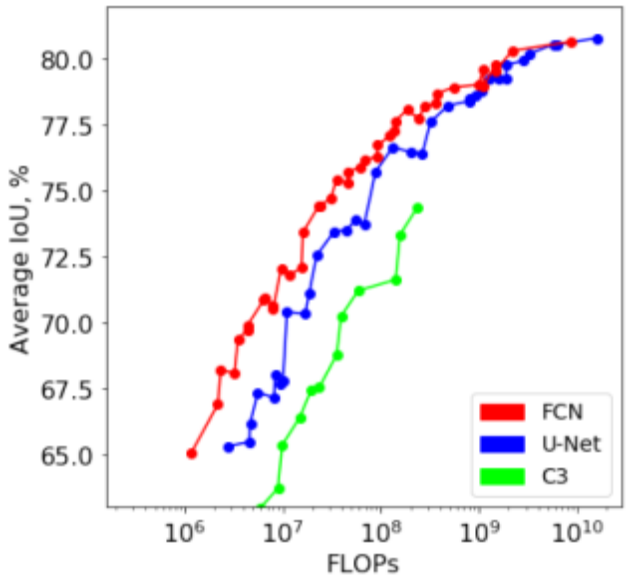


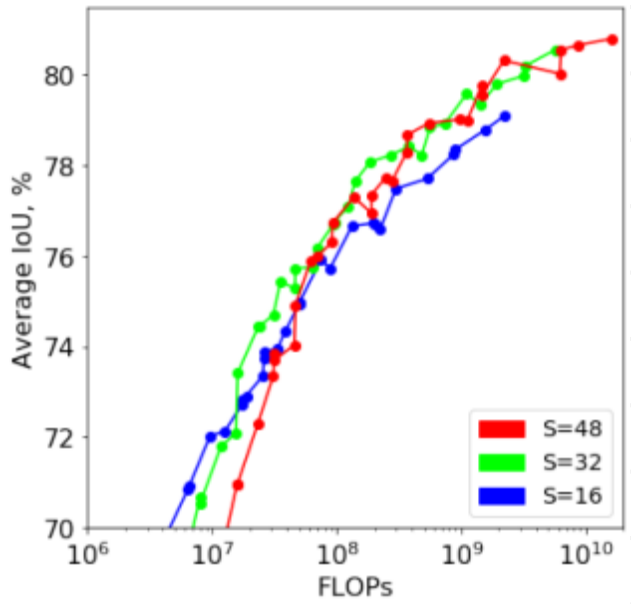
[Graham et al. CVPR18]

Submanifold-Sparse ConvNets

ShapeNet benchmarks:
(note: FLOPs != runtime)



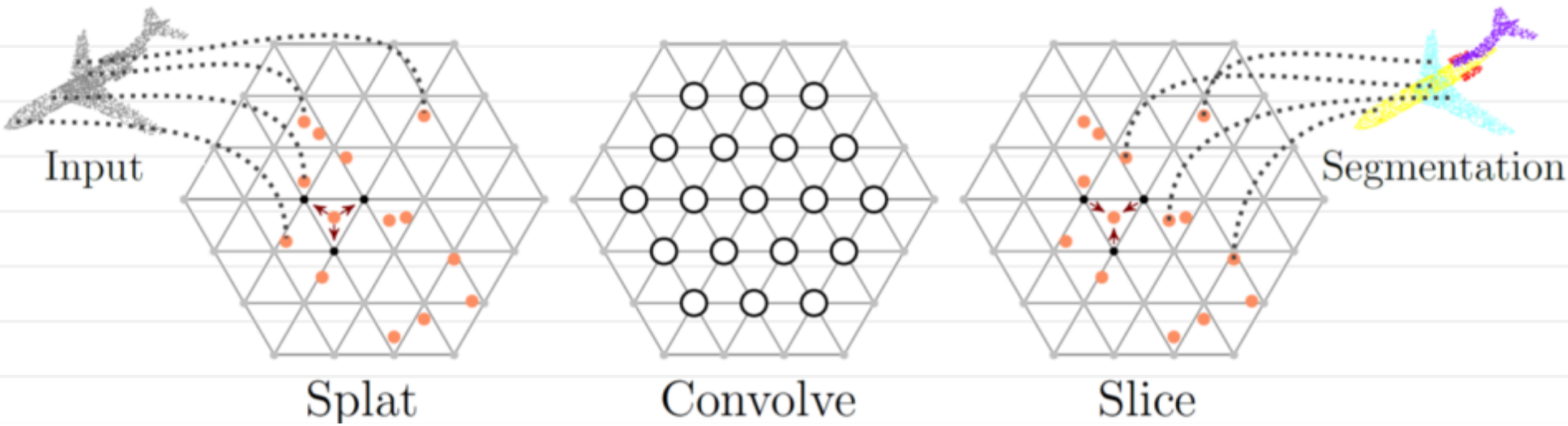




Other related idea: use octrees to store data (much more tricky)

[Graham et al. CVPR18]

SplatNet



$$\hat{F}_c = S_{slice} B_{conv} S_{splat} F_c$$

- Permutohedral lattices are more efficient than rectangular grids in high dimensions (not really used in this work)
- SOTA performance (last year) across number of tasks

[Su et al. CVPR18]

Kd-trees: recap

function node = *BuildKdTree*(M)

if |*M*| > 1

a = axis with the biggest variance

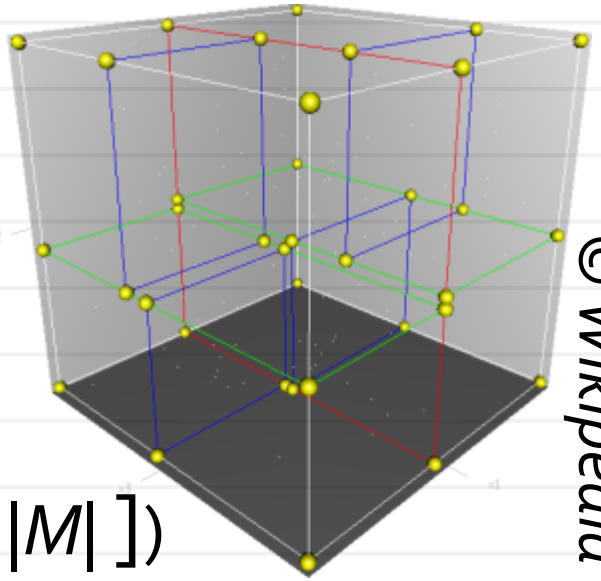
M = *sort* (*M*, axis = *a*)

node.child1 = *BuildKdTree*(*M*[1:|*M*|/2])

node.child2 = *BuildKdTree*(*M*[|*M*|/2+1:|*M*|])

return node

else return *CreateLeaf* (*M*[0])

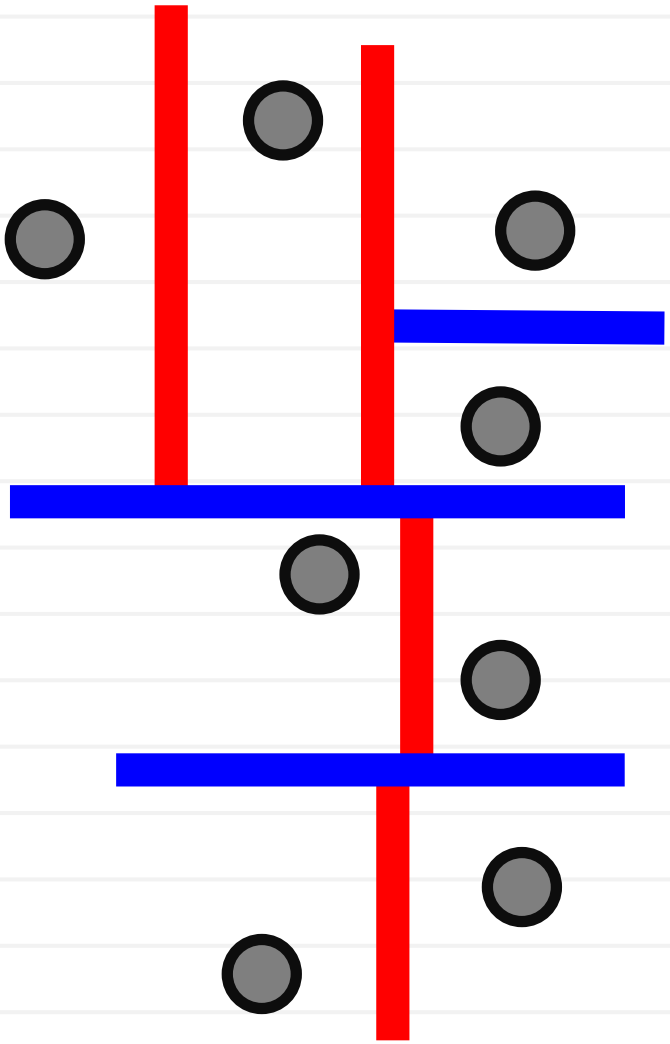


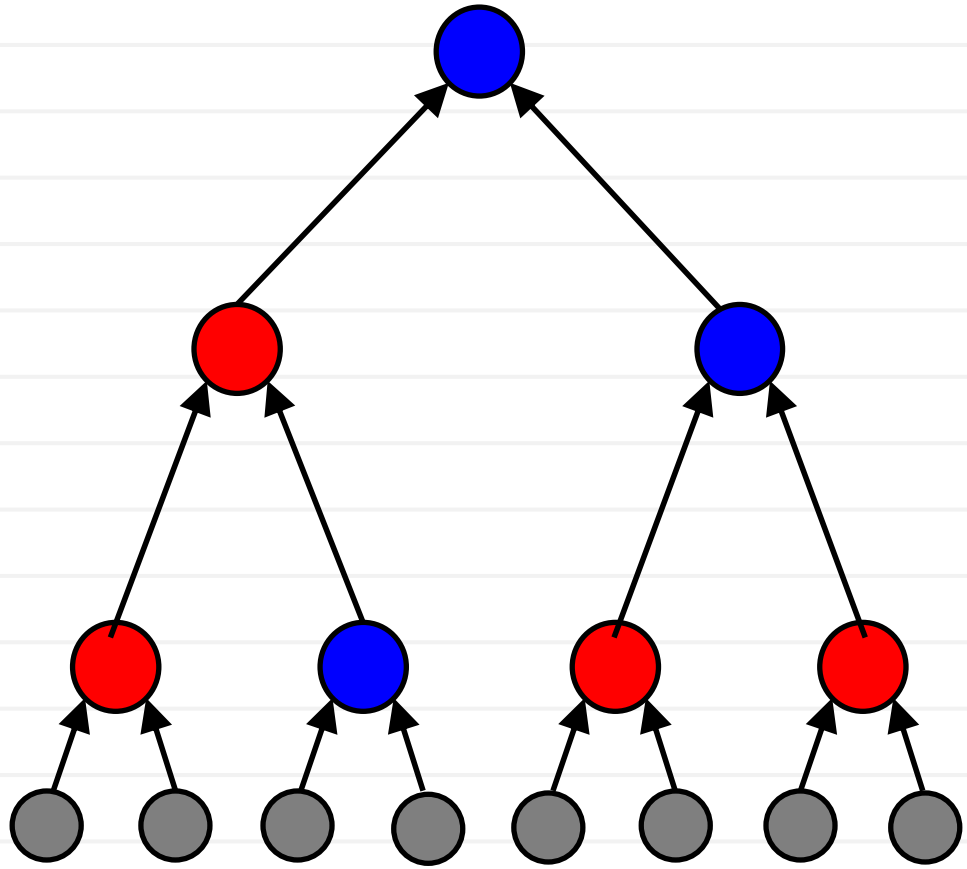
© *Wikipedia*

Conventional use:

- Approximate nearest neighbor (alternate tree descent and backtracking)
- Ray tracing

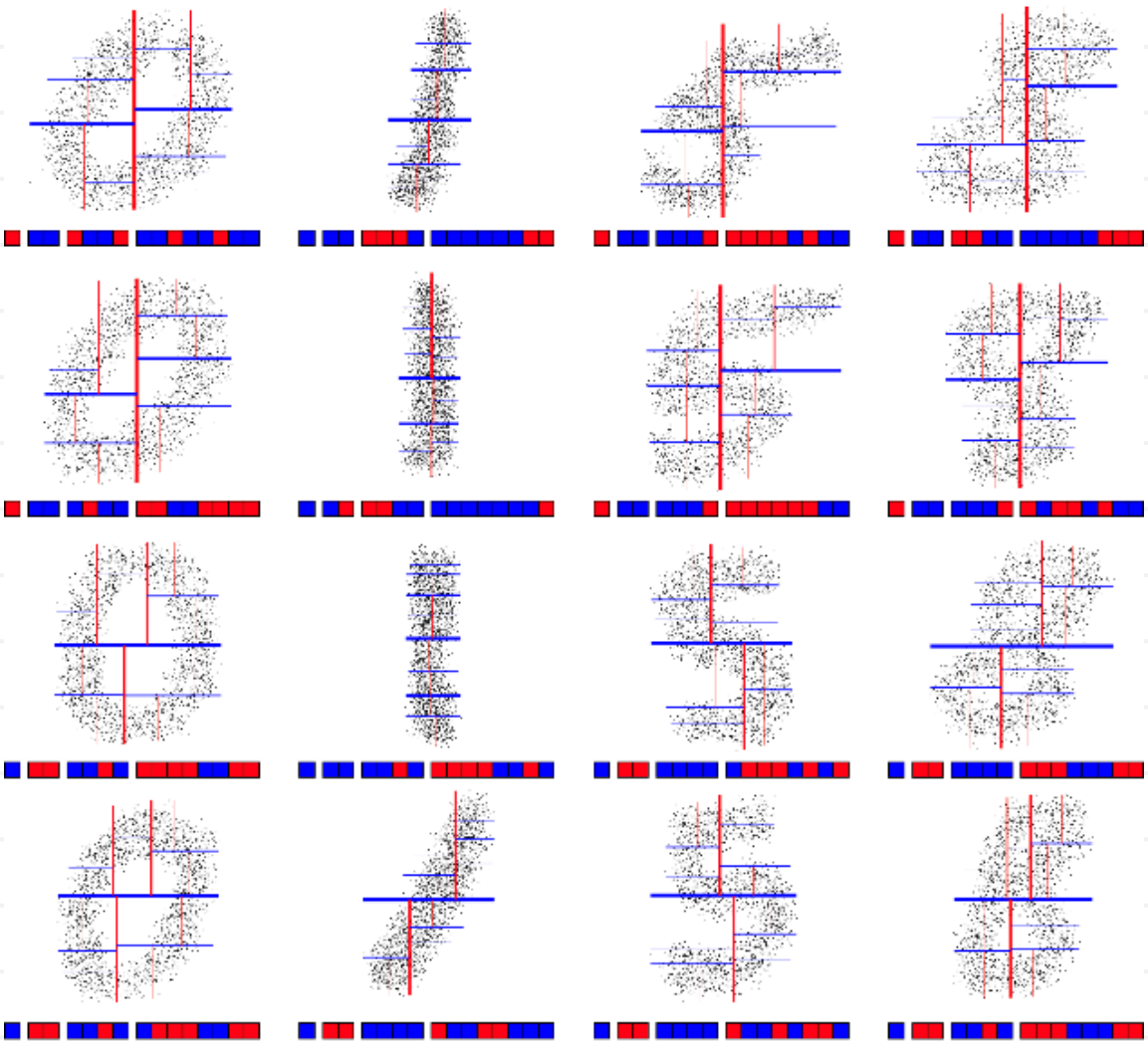
Kd-tree: 2D example





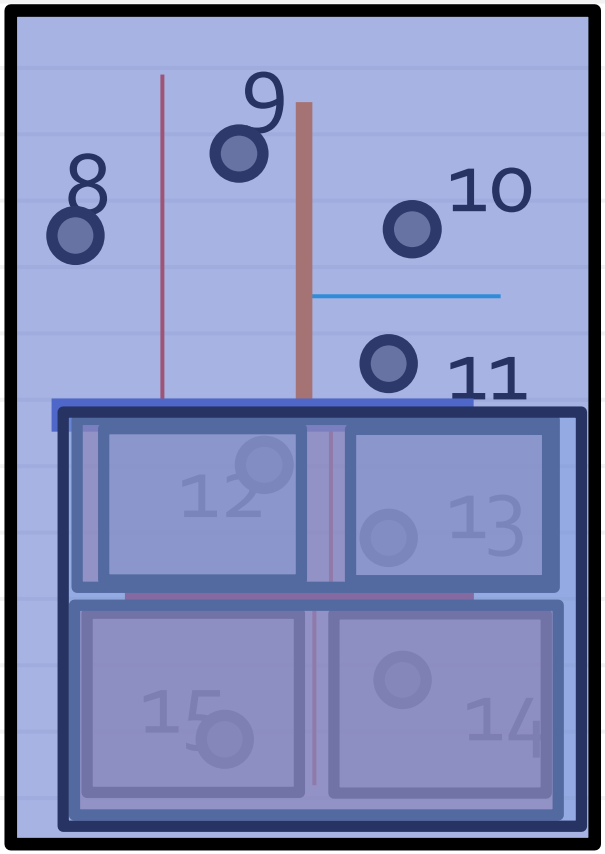
Leaves = original points

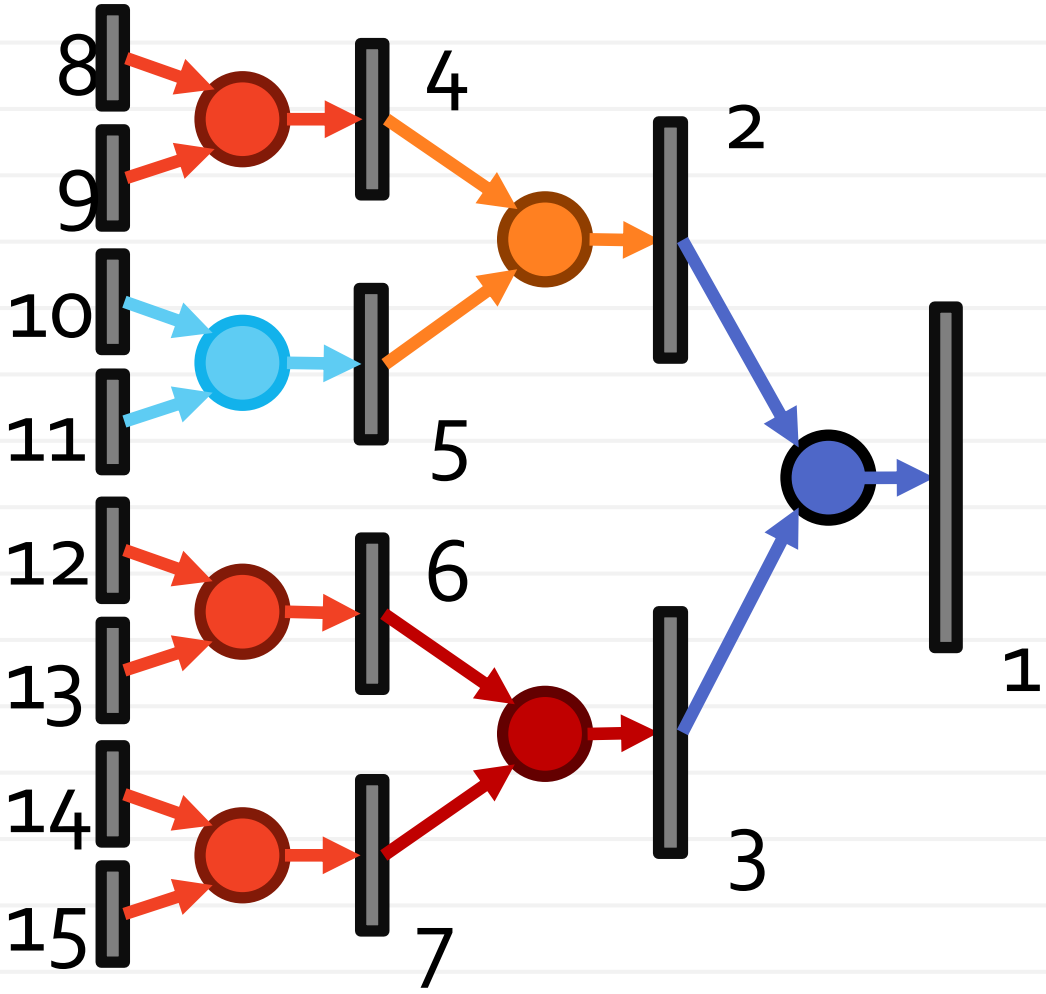
Kd-tree for MNIST-dots



- 2D point clouds obtained from non-zero pixels in MNIST
- Kd-trees contain information about shapes
- Naïve kd-based classification: 82.4%

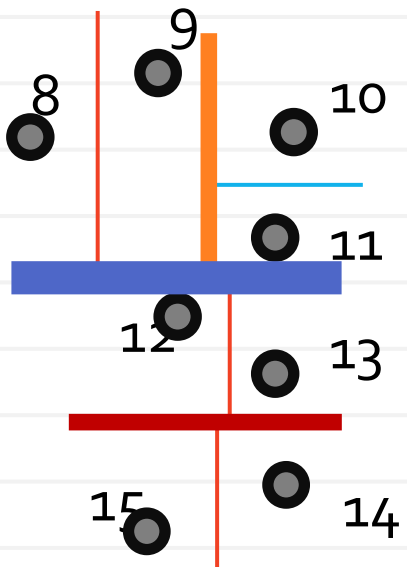
Hierarchical features from a Kd-tree

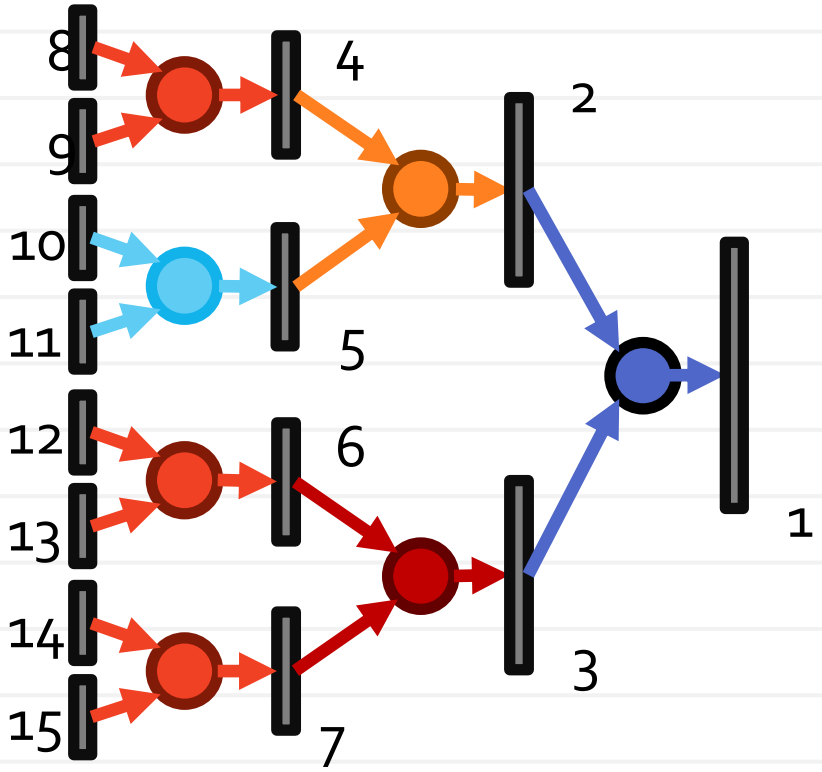




$$v_7 = \phi(W_7[v_{14}; v_{15}] + b_7)$$
$$v_6 = \phi(W_6[v_{12}; v_{13}] + b_6)$$
$$v_3 = \phi(W_3[v_6; v_7] + b_3)$$

Hierarchical features from a Kd-tree



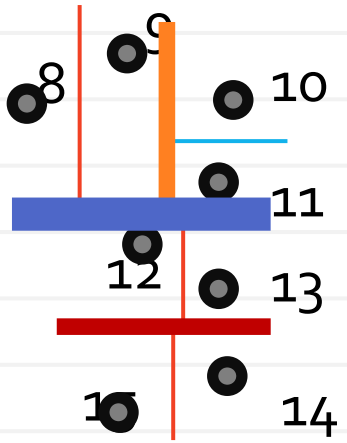


$$v_7 = \phi(W_7[v_{14}; v_{15}] + b_7)$$

$$v_6 = \phi(W_6[v_{12}; v_{13}] + b_6)$$

Kd-network "equation"

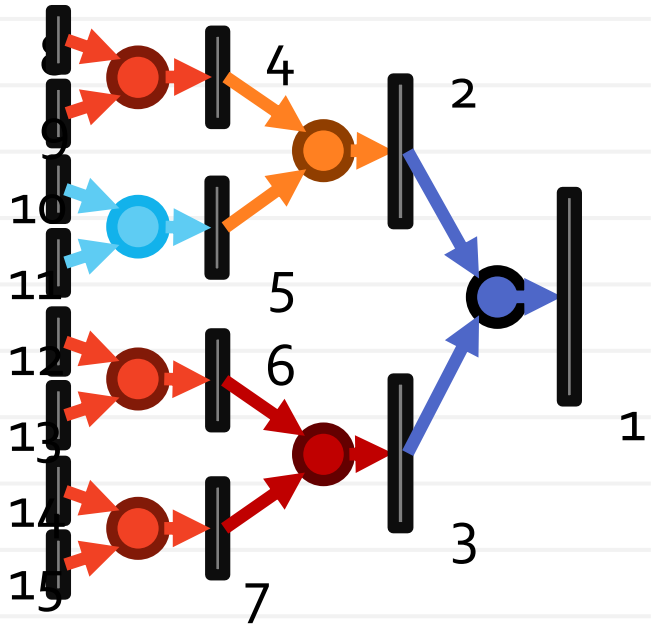
$$v_i = \begin{cases} \phi\left(W_x^{l_i}[v_{2i}; v_{2i+1}] + b_x^{l_i}\right) & \text{if } d_i = \text{'x'} \\ \phi\left(W_y^{l_i}[v_{2i}; v_{2i+1}] + b_y^{l_i}\right) & \text{if } d_i = \text{'y'} \\ \phi\left(W_z^{l_i}[v_{2i}; v_{2i+1}] + b_z^{l_i}\right) & \text{if } d_i = \text{'z'} \end{cases}$$



$$v_i = \phi\left(W_{s_i}^{l_i}[v_{2i}; v_{2i+1}] + b_{s_i}^{l_i}\right)$$

ConvNet analogies:

- Localized linear operators
- Stationarity
- Bottom-up processing

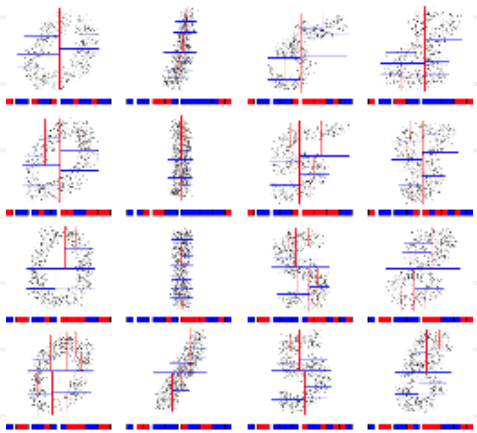


Recap: Kd-network classifier

1. *Build a Kd-tree*
2. *Run recursively bottom-up:*

$$v_i = \phi\left(W_{s_i}^{l_i}[v_{2i}; v_{2i+1}] + b_{s_i}^{l_i}\right)$$

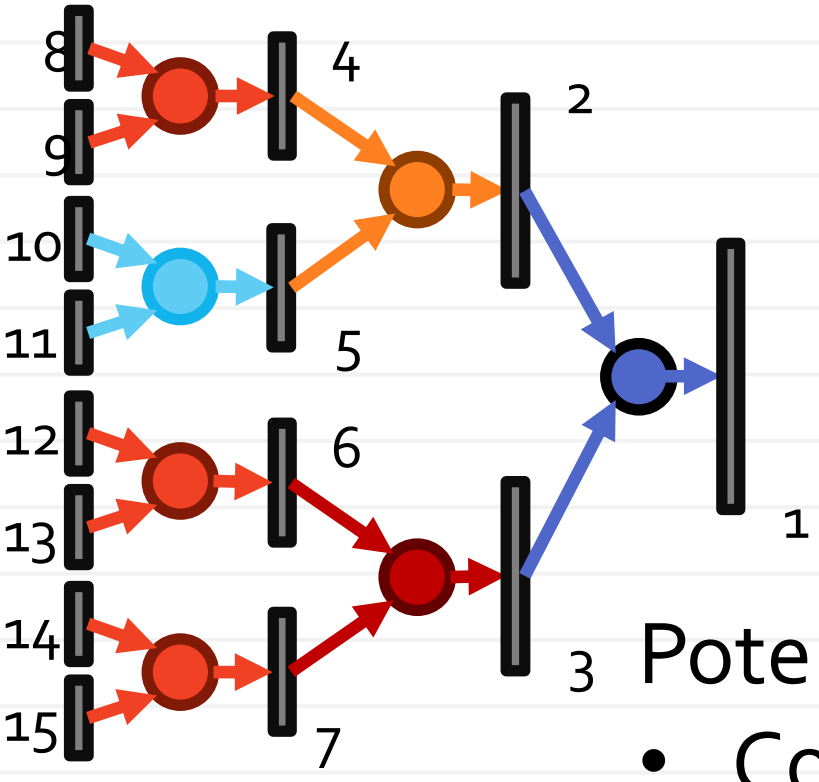
3. *Output* $p(y|x) = softmax\left(W_0 v_1 + b_0\right)$



Learnable parameters

$$\left\{\begin{array}{c} \{W_0; b_0\} \\ \{W_x^1; b_x^1; W_y^1; b_y^1; W_z^1; b_z^1\} \\ \{W_x^2; b_x^2; W_y^2; b_y^2; W_z^2; b_z^2\} \\ \\ \{W_x^L; b_x^L; W_y^L; b_y^L; W_z^L; b_z^L\} \end{array}\right\}$$

Kd-network: leaf representations



$[x, y, R, G, B]$

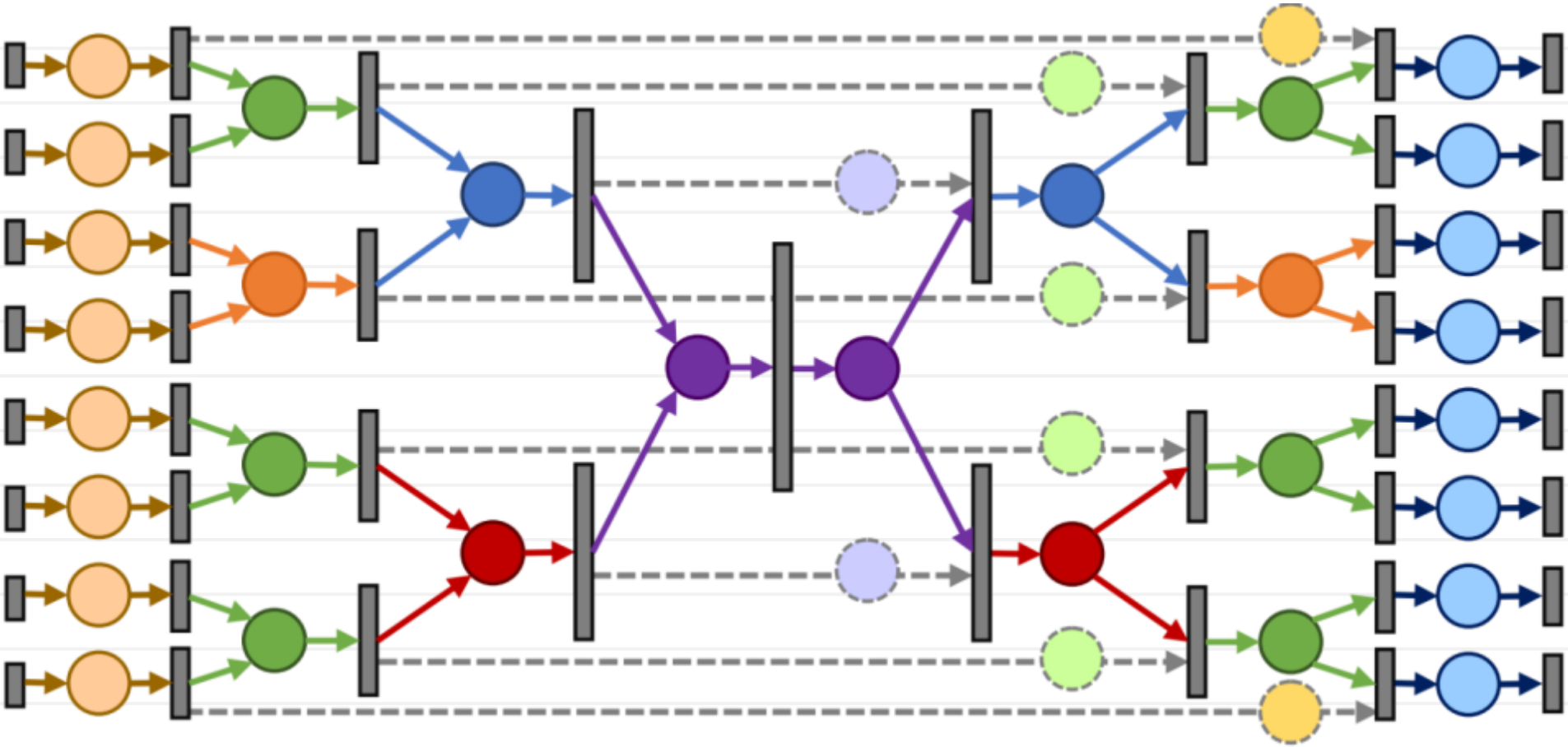
Potential options:

- Coordinates
- Color (R,G,B)
- Conv features
- Normal vector
- ..or nothing

[Klokov ICCV17]

Kd-Nets for semantic segmentation

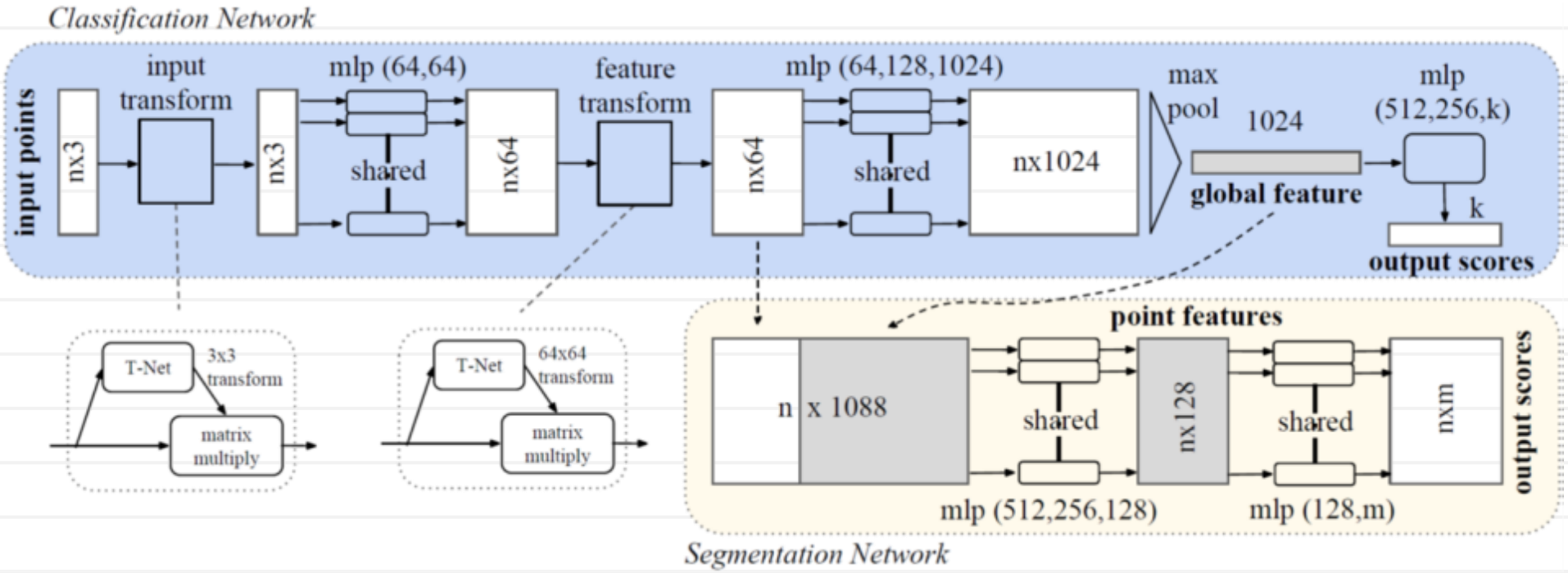
"Kd-Unet":



[Klokov ICCV17]

PointNet

Key idea: use only permutation invariant operators

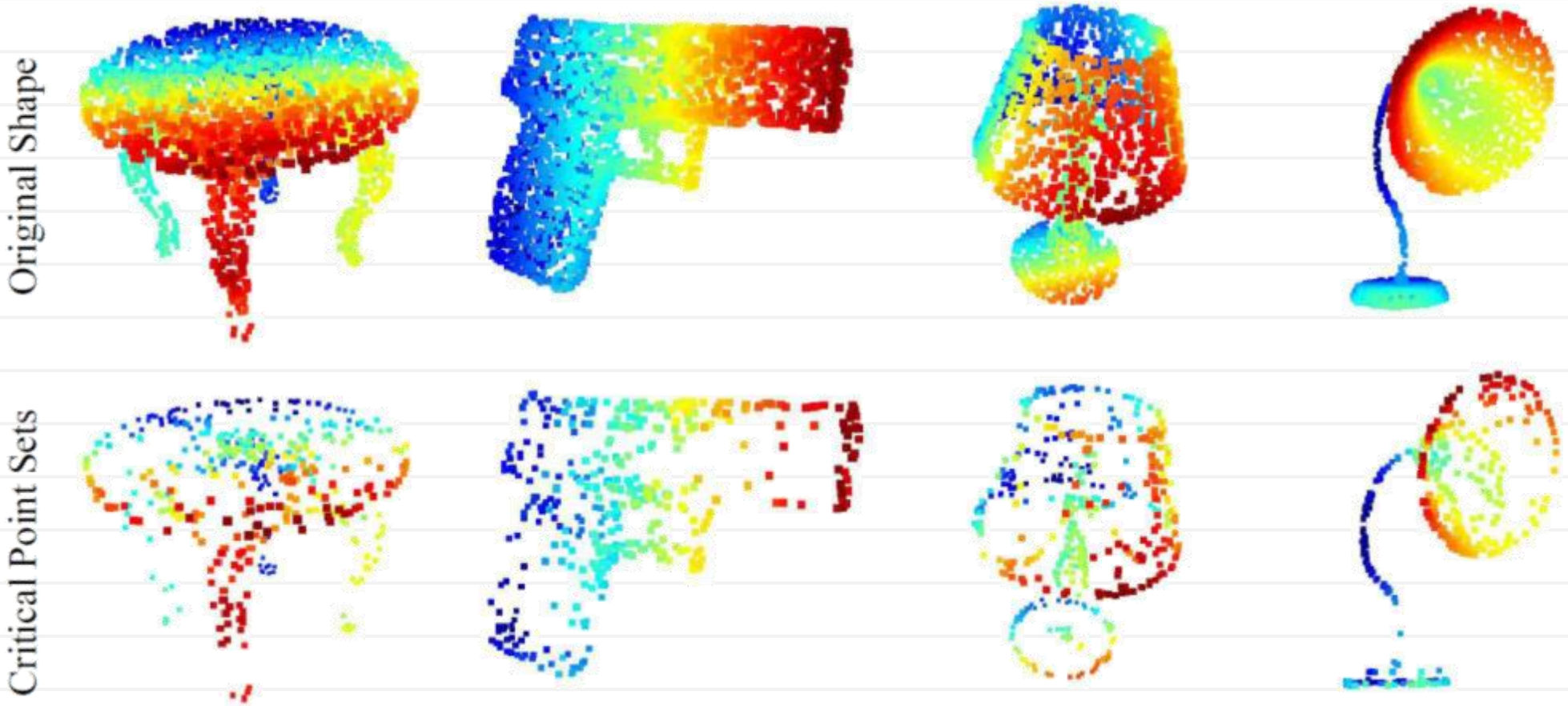


T-Net = 1x1 MLP + max-pool over points + MLP

Overall: efficient architecture with good performance

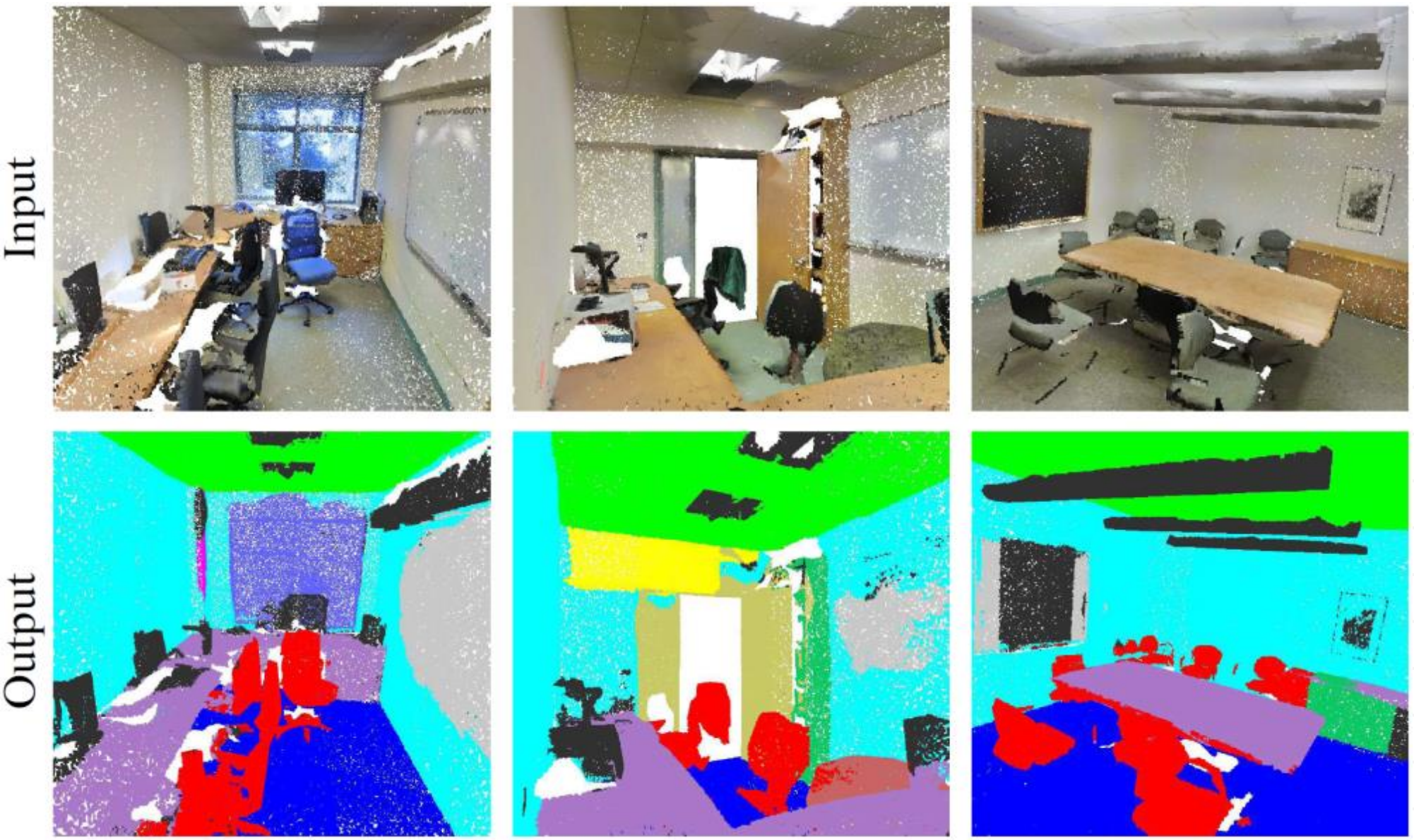
[Qi et al. CVPR17]

PointNet results: critical sets



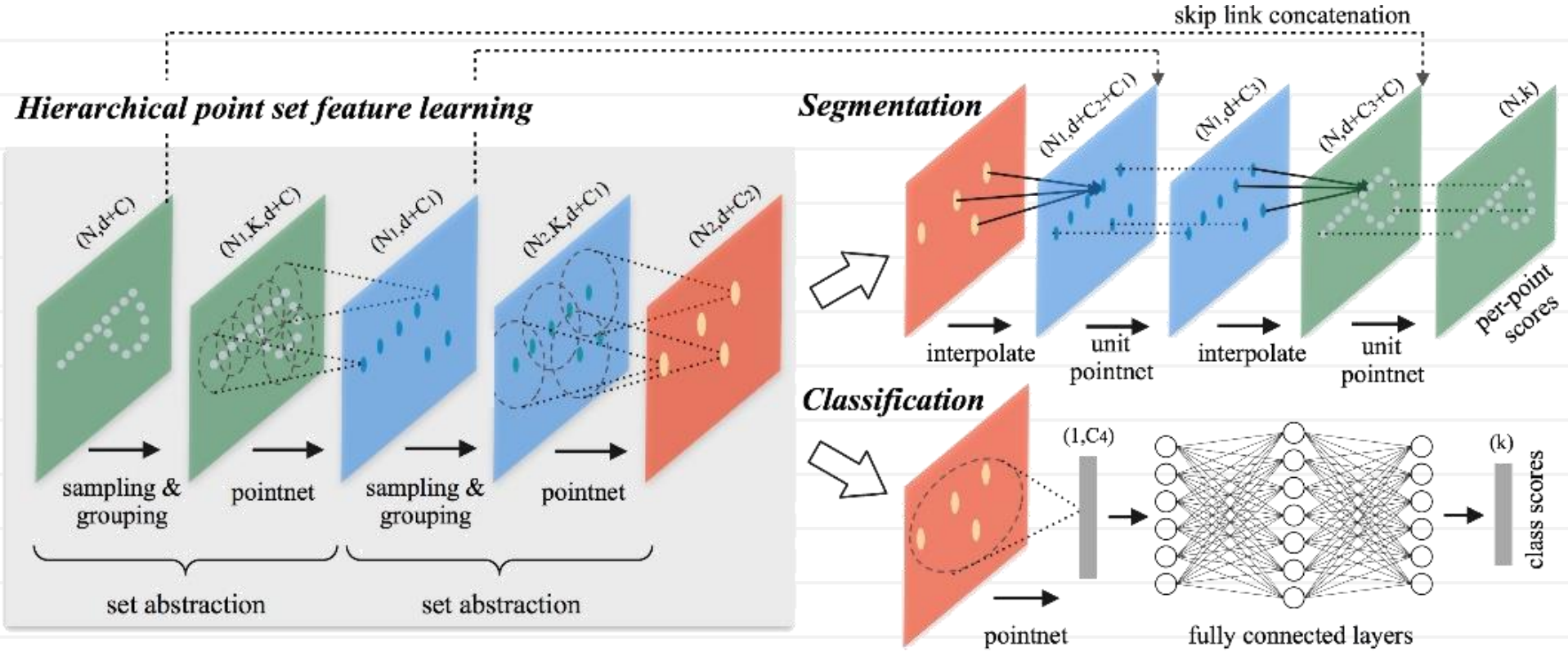
[Qi et al. CVPR17]

PointNet results



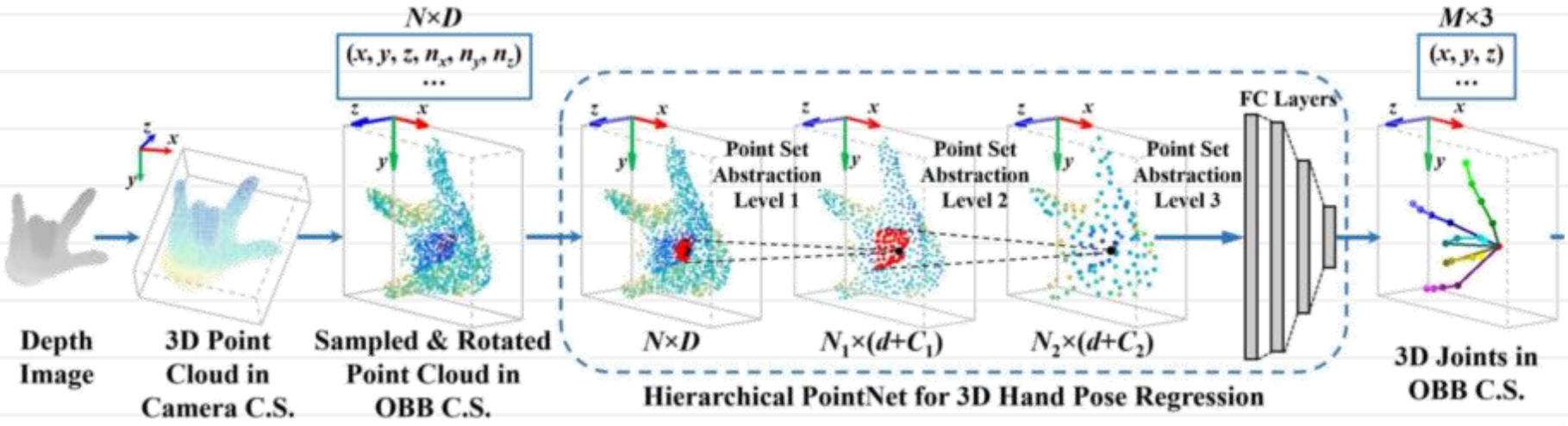
[Qi et al. CVPR17]

PointNet++

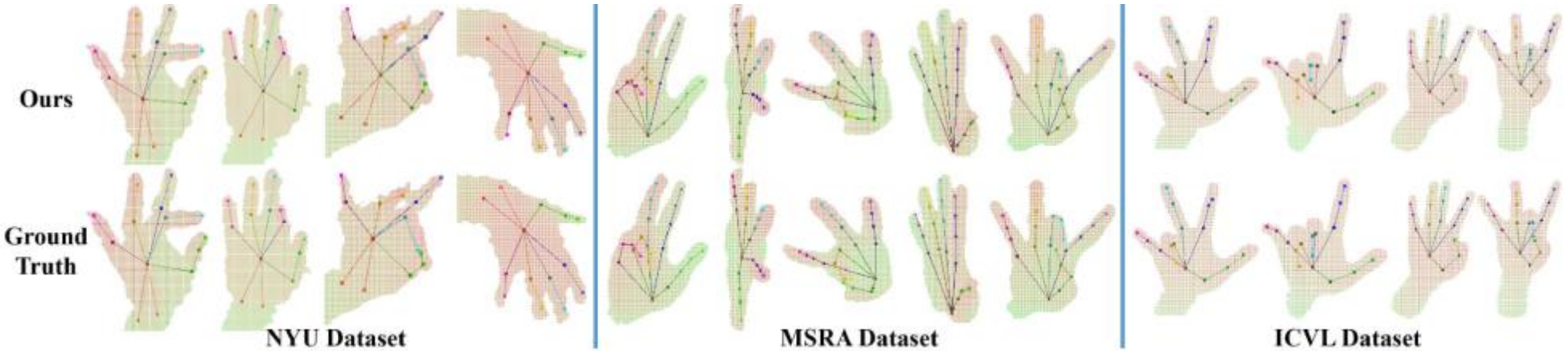


- Replaces global pooling with localized grouping and pooling
- Improved performance over PointNet (but slower)

Hand PointNet

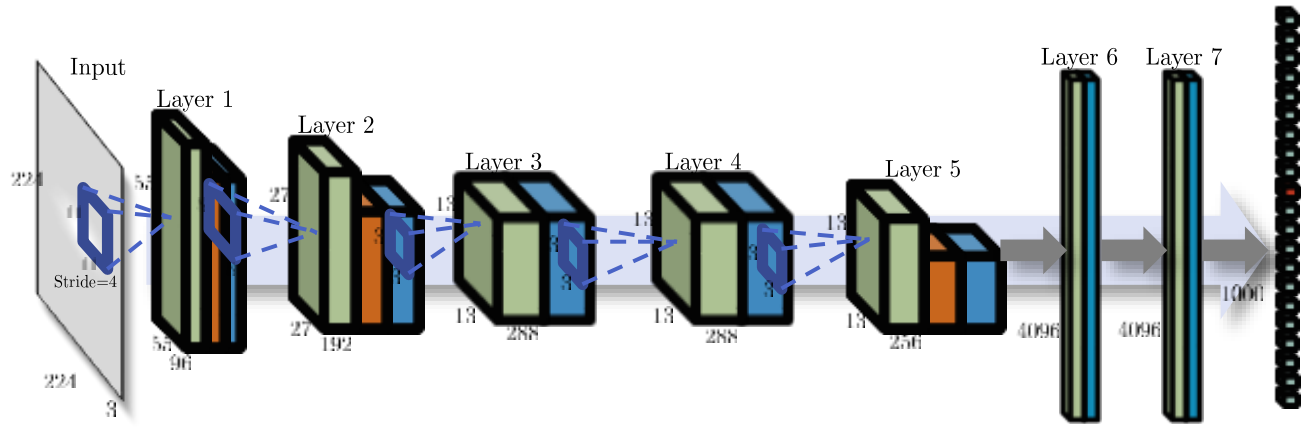


- Uses PointNet++ for hand keypoint estimation



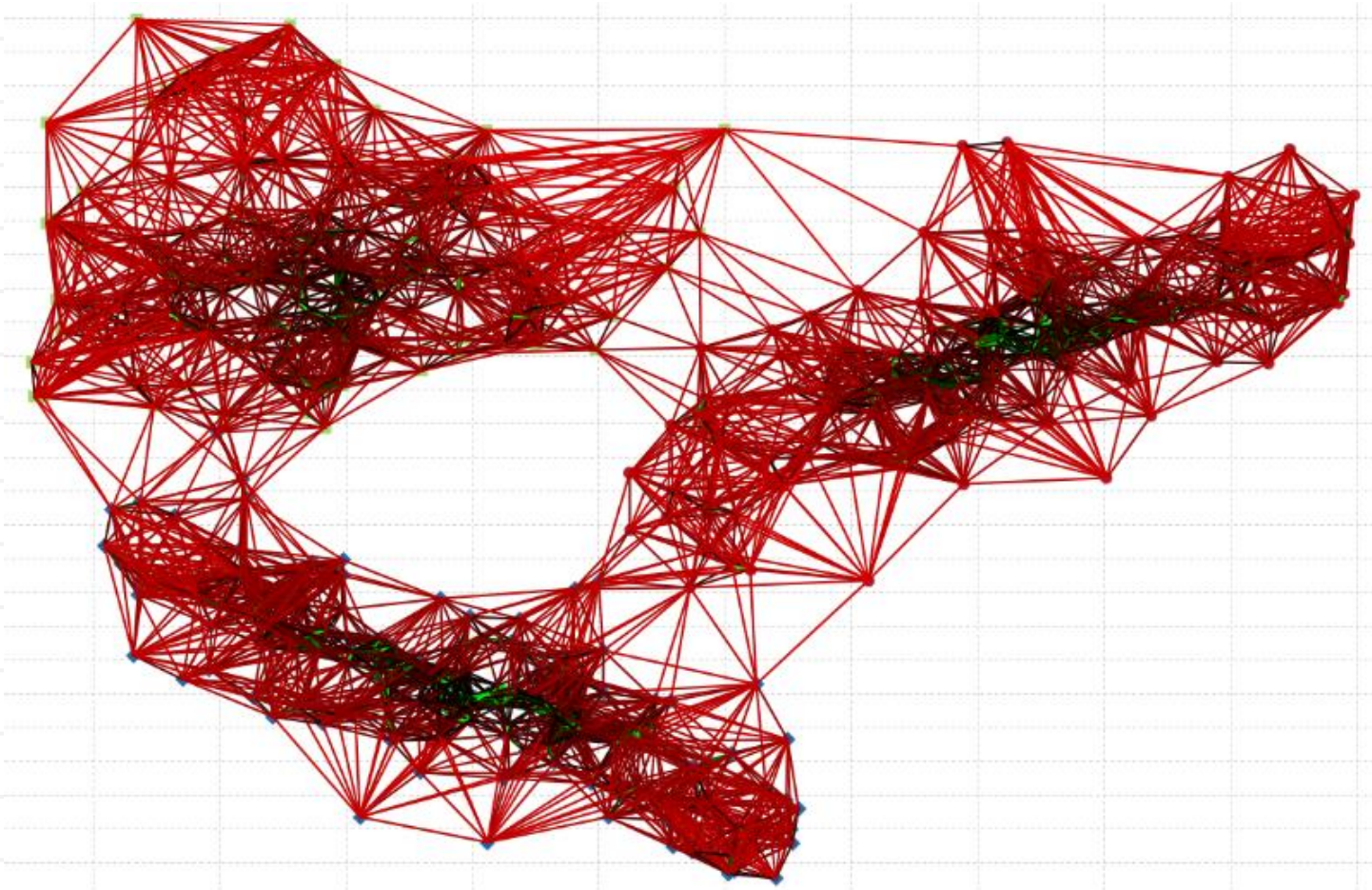
[Ge et al. CVPR18]

General motivation



This is ConvNet
It does recognition very well
Be like ConvNet

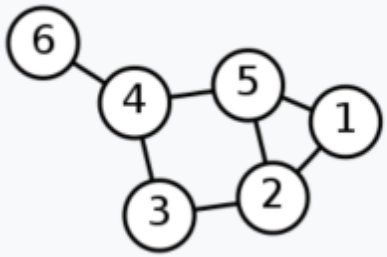
Point clouds as graphs



- k-nearest neighbor graphs
- hierarchical clustering graph
- ….

Graph Laplacian

Graph Laplacian is a linear operator on graph function that measures the difference between the value at each vertex and adjacent neighbors

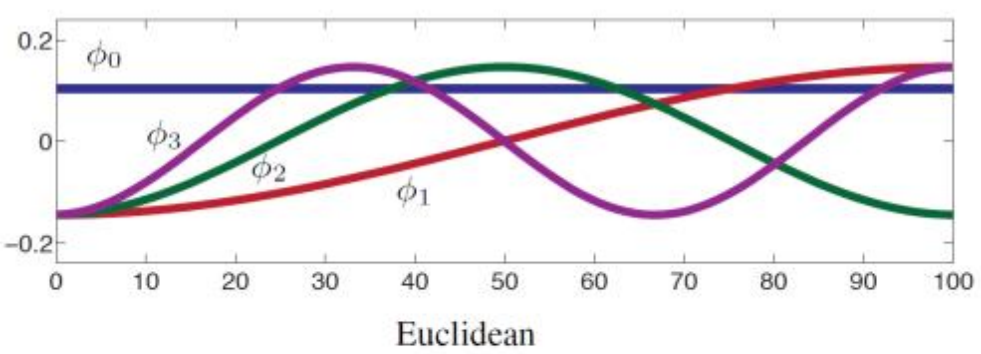
Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

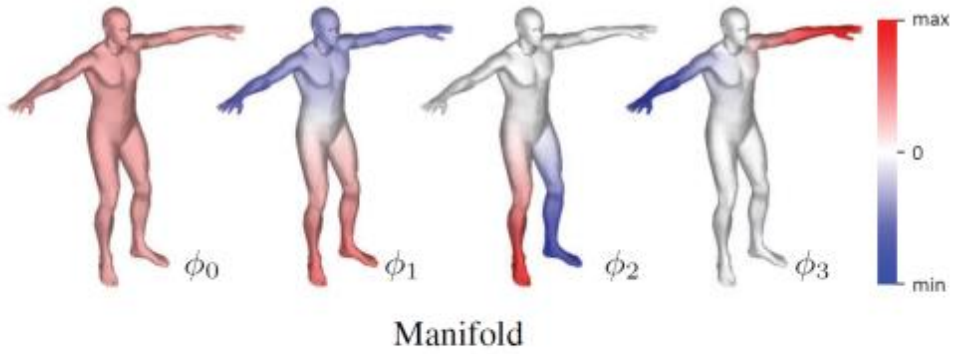
$$L_{\text{unnorm}} = D - A$$

$$L = I - D^{-1/2} A D^{-1/2}$$

- If *x* is a signal (function) on a graph, then *Lx* measures at vertex *i* measures the diff between x_i and the weighted average of the neighbors
- Laplacian measures pointwise smoothness

Eigenbasis of Laplacian





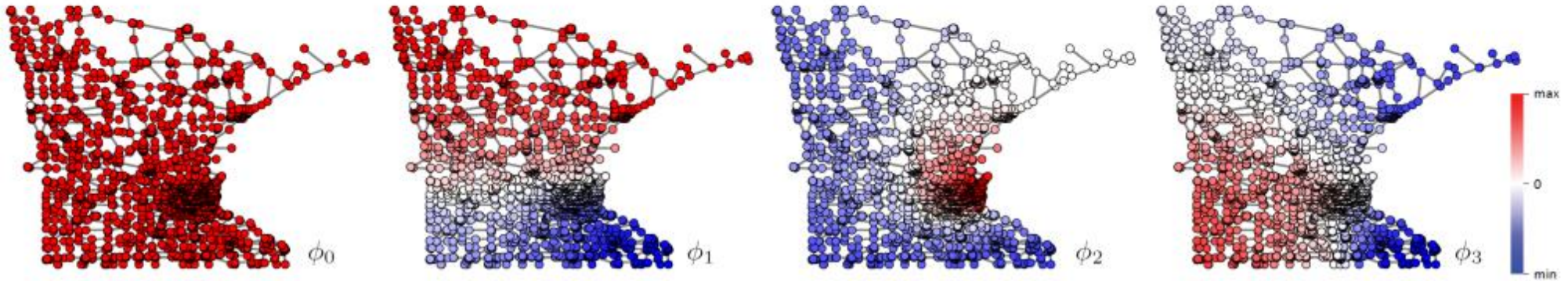


Image source: [Bronstein et al. 2017]

- For regular grids, eigenvectors are sine waves
- Projection of a signal onto the eigenbasis of the Laplacian generalizes Fourier transform to graphs

Eigenbasis of Laplacian

Eigenvectors of the Laplacian: $\Delta\psi_n = \lambda_n\psi_n$

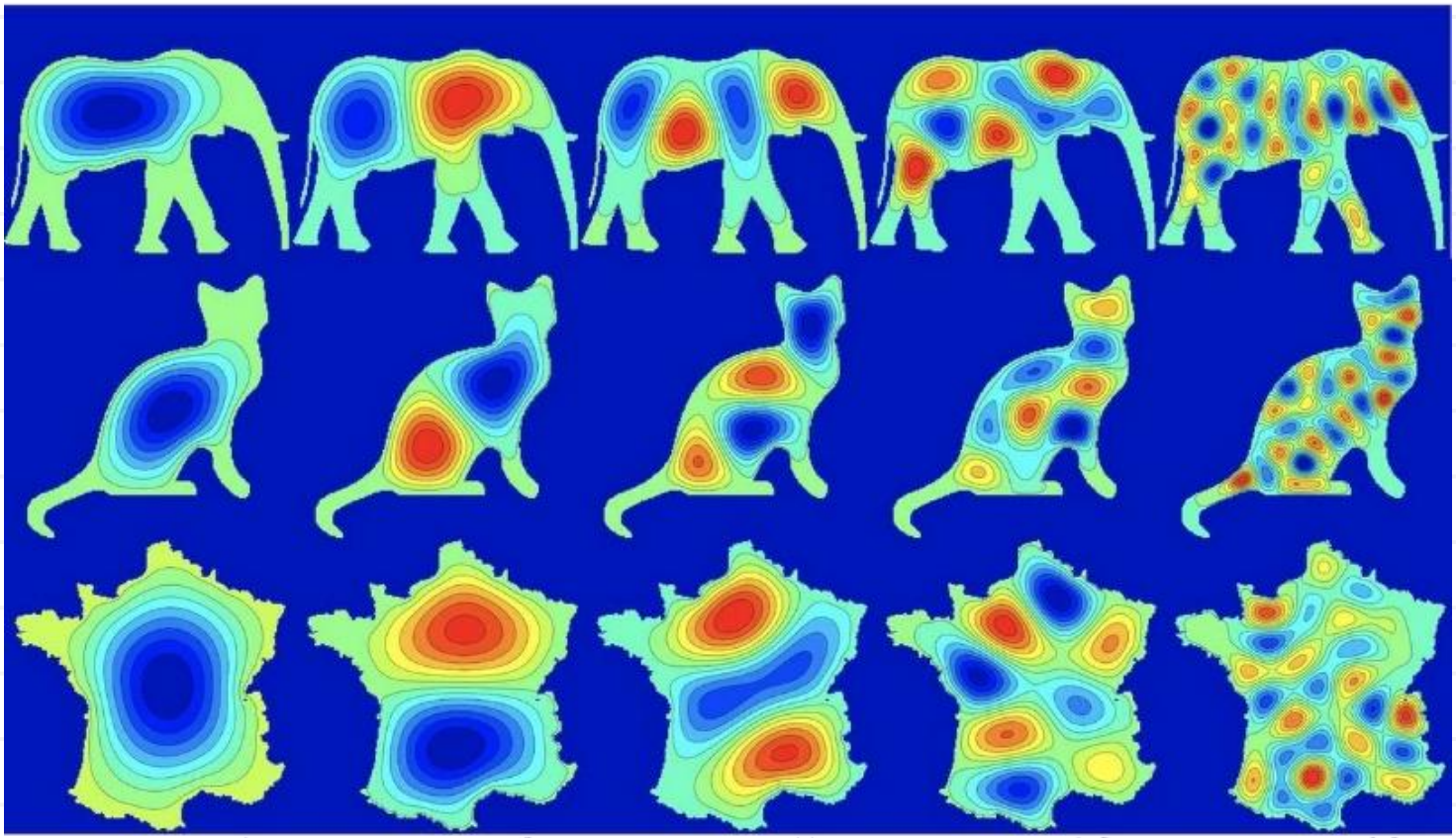


Image source: Gabriel Peyre

- For regular grids, eigenvectors are sine functions
- Projection of a signal onto the eigenbasis of the Laplacian hence generalizes Fourier transform to graphs

Graph-convolutional neural networks

- Projection of a signal on the eigenbasis of the Laplacian generalizes Fourier transform to graphs
- Kernel convolution is generalized as:

inverse graph Fourier transform

$$g_\theta \star x = U \operatorname{diag}(\theta) U^T x$$

Convolution parameters

graph Fourier transform

Neat, but:

- Not localized
- O(n) learnable parameters
- Computationally heavy (U is dense)
- Poor generalization to new graphs

Same story with standard convolutions (but we can impose restriction in the spatial domain)

[Bruna ICLR14]

Graph-convolutional neural networks

$$g_\theta \star x = U \operatorname{diag}(\theta) U^T x$$

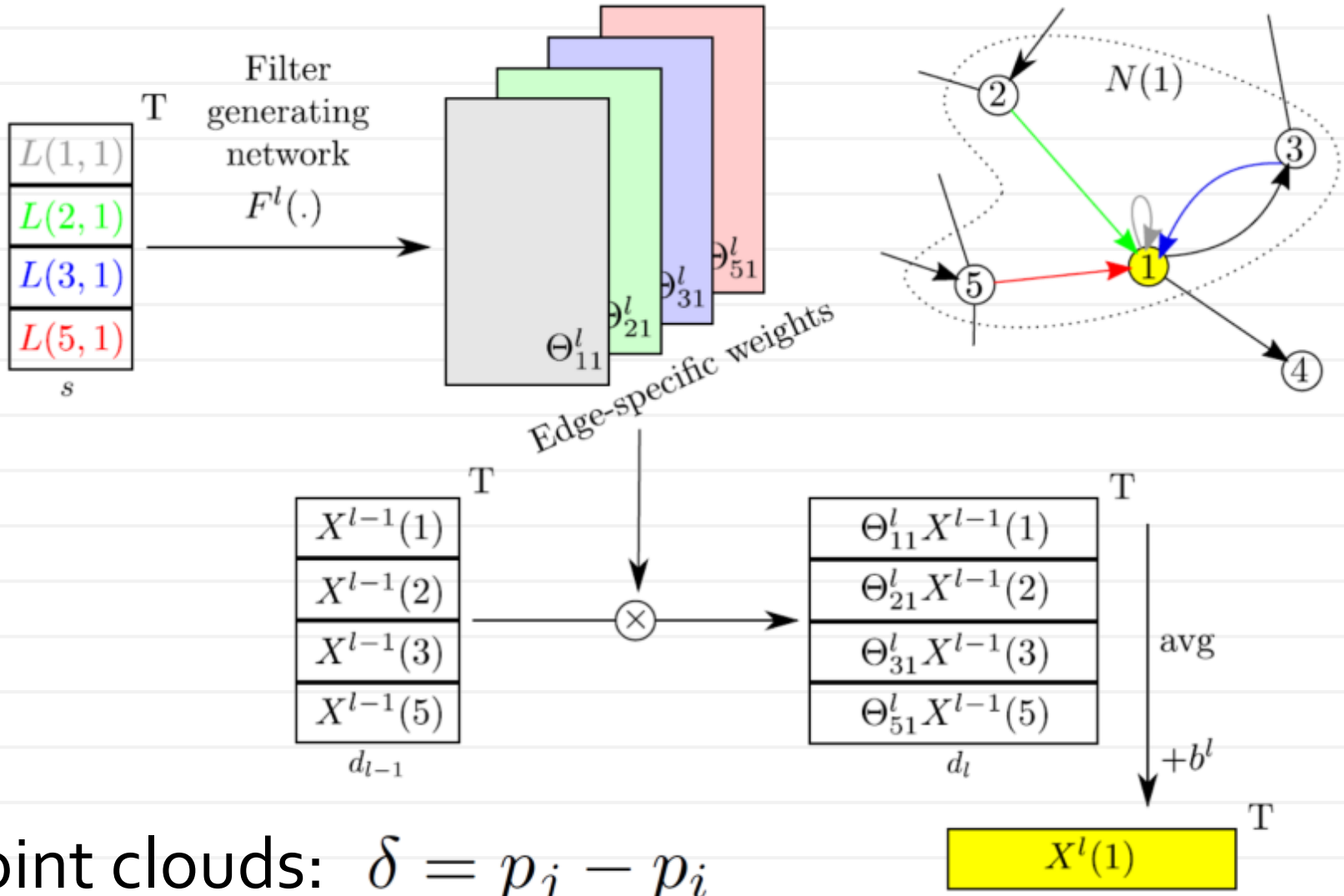
- Laplacian is a convolution: $Lx = U \Lambda U^T x$
- Degrees of the Laplacian are convolutions:

$$L^k x = (U \Lambda U^T)^k x = U \Lambda^k U^T x$$

- Laplacian and its degrees are k-localized
- Idea: restrict considered filters to polynomials of degree K of the Laplacian (loose analogy: restricting spatial extent of filters in normal convolutions)
- The new operator is localized, has O(K) learnable parameters (K << n), is efficient (no Fourier), generalizes reasonably well to new graphs
- Problem: not so expressive

[Defferrard et al. NIPS16]

Dynamic edge conditioned filters

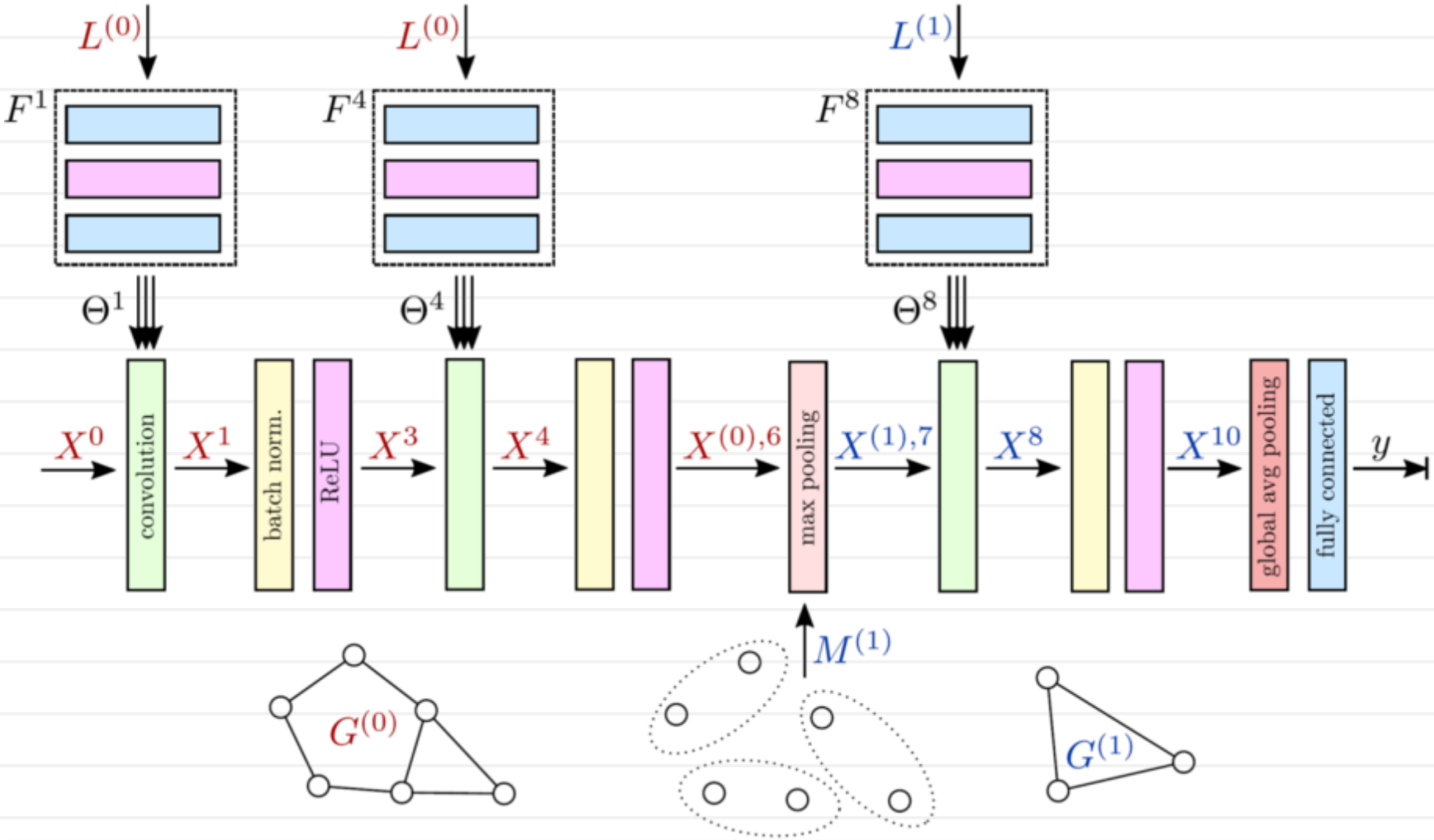


For point clouds: $\delta = p_j - p_i$

$$L(j,i) = (\delta_x, \delta_y, \delta_z, ||\delta||, \arccos \delta_z/||\delta||, \arctan \delta_y/\delta_x)$$

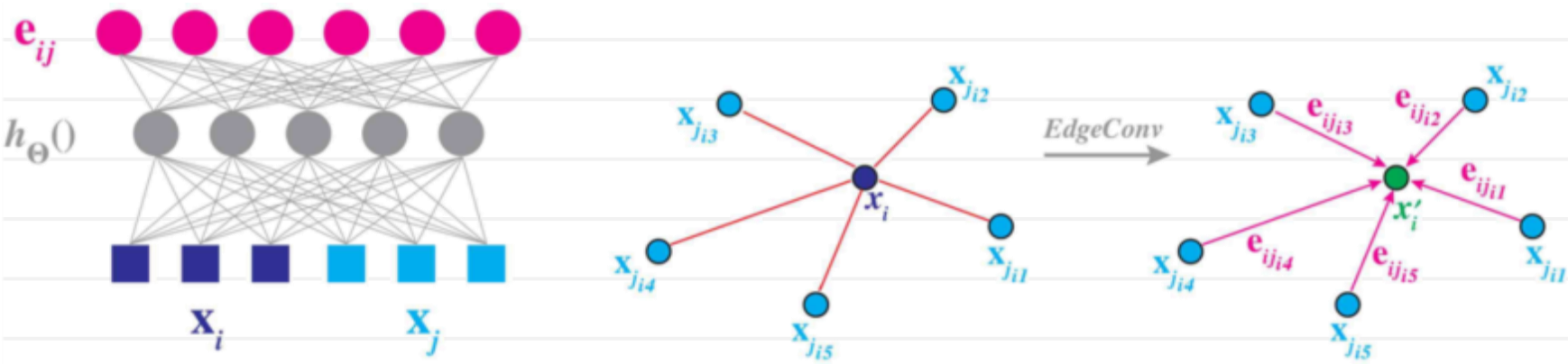
[Simonovsky CVPR17]

Dynamic edge conditioned filters



[Simonovsky CVPR17]

Dynamic Graph CNN



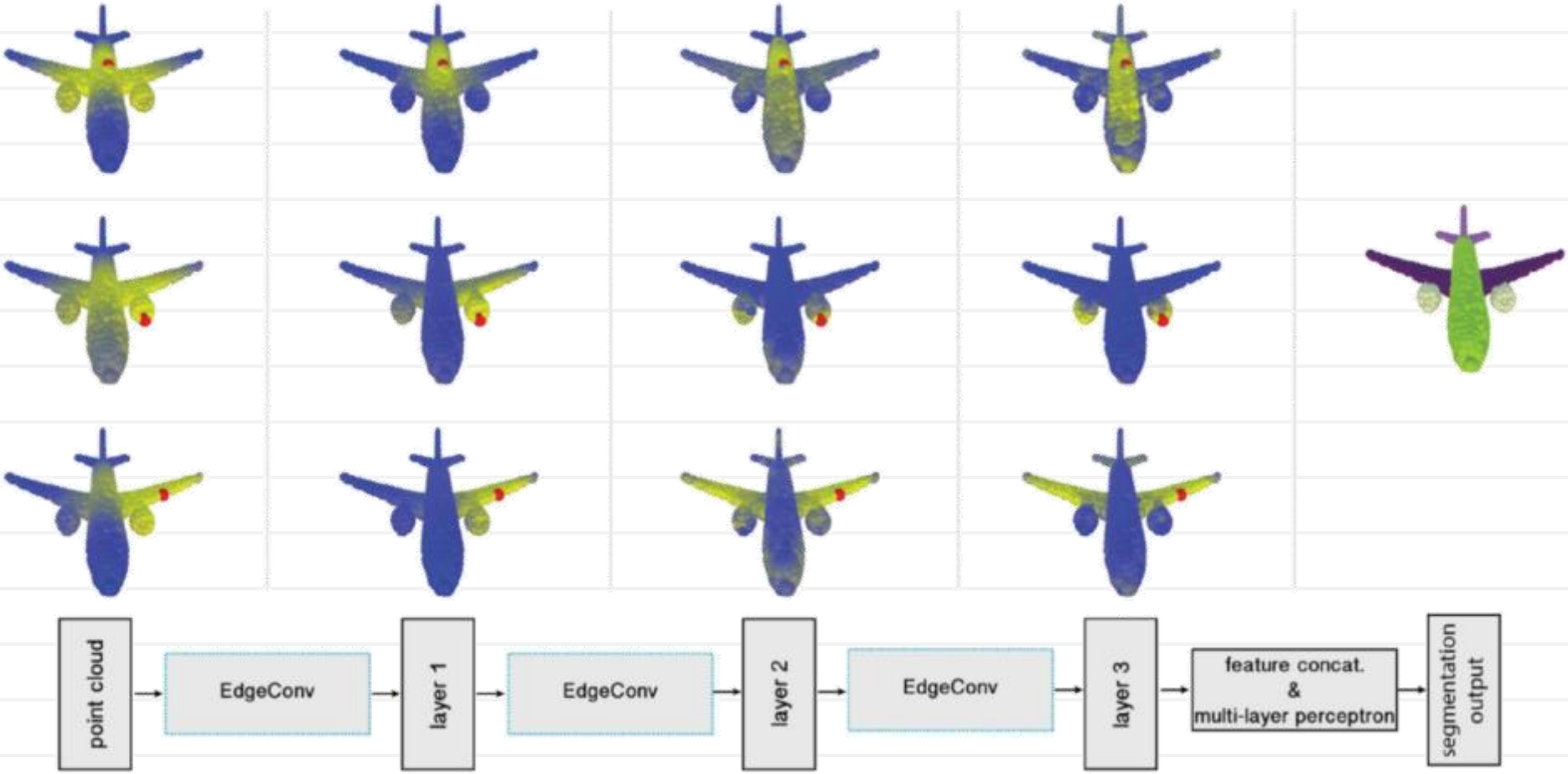
$$x'_i = \square_{j:(i,j)\in\mathcal{E}} h_{\Theta}(x_i, x_j)$$

□ = sum or max

Very similar to [Simonovsky], but the graph is recomputed after each convolution

[Wang et al. CVPR18]

Dynamic Graph CNN



[Wang et al. CVPR18]

Dynamic graph CNN

ModelNet 40 benchmark:

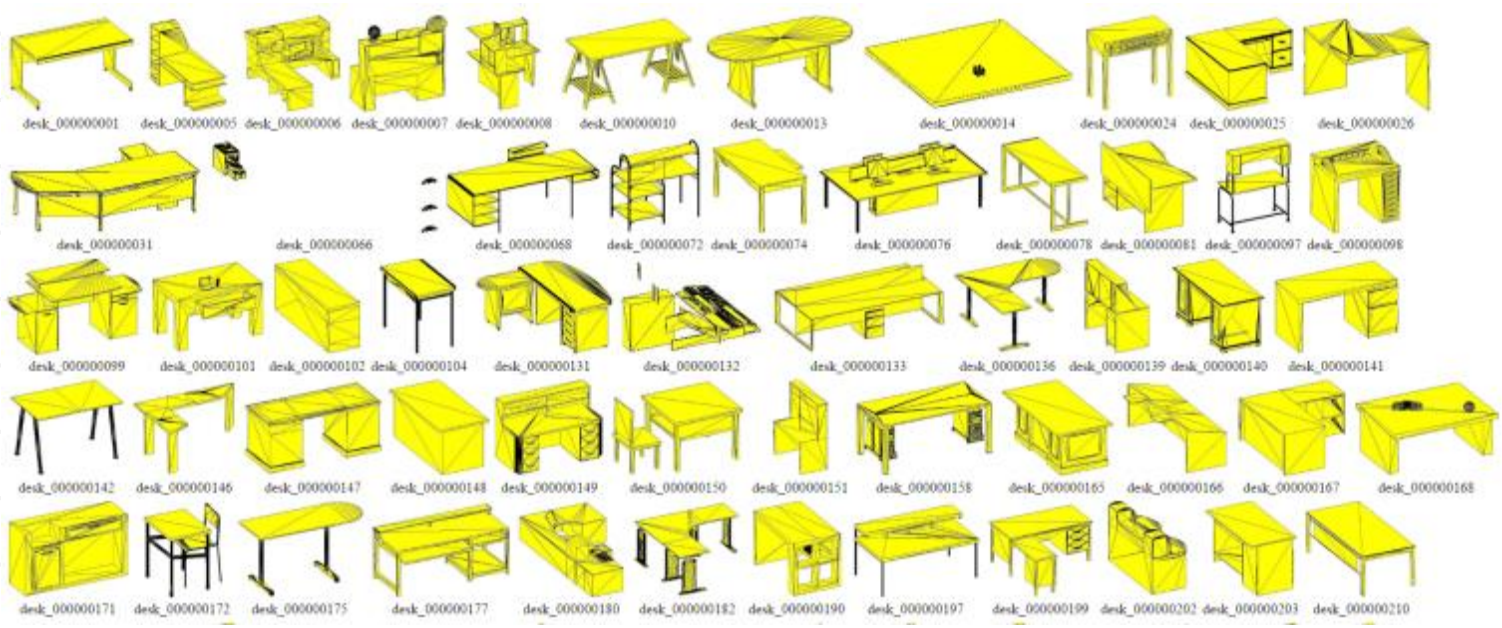
	Mean Class Accuracy	Overall Accuracy
3DShapeNets [54]	77.3	84.7
VoxNet [30]	83.0	85.9
Subvolume [35]	86.0	89.2
ECC [45]	83.2	87.4
PointNet [34]	86.0	89.2
PointNet++ [36]	-	90.7
Kd-Net (depth 10) [20]	-	90.6
Kd-Net (depth 15) [20]	-	91.8
Ours (baseline)	88.8	91.2
Ours	**90.2**	**92.2**

[Wang et al. CVPR18]

ModelNet: bathtubs

ModelNet: desks and tables

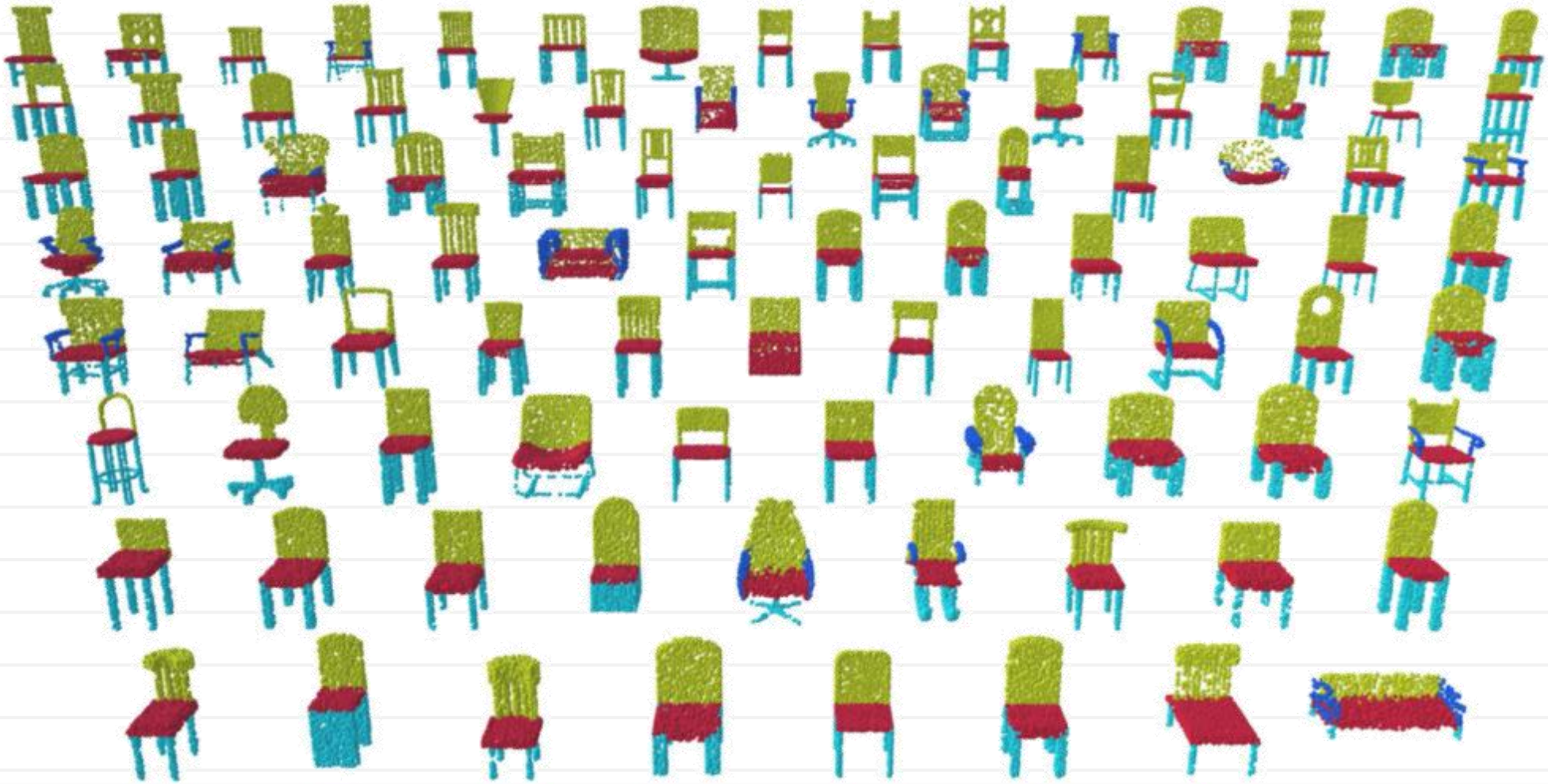
"desk"



"table"

Dynamic graph CNN

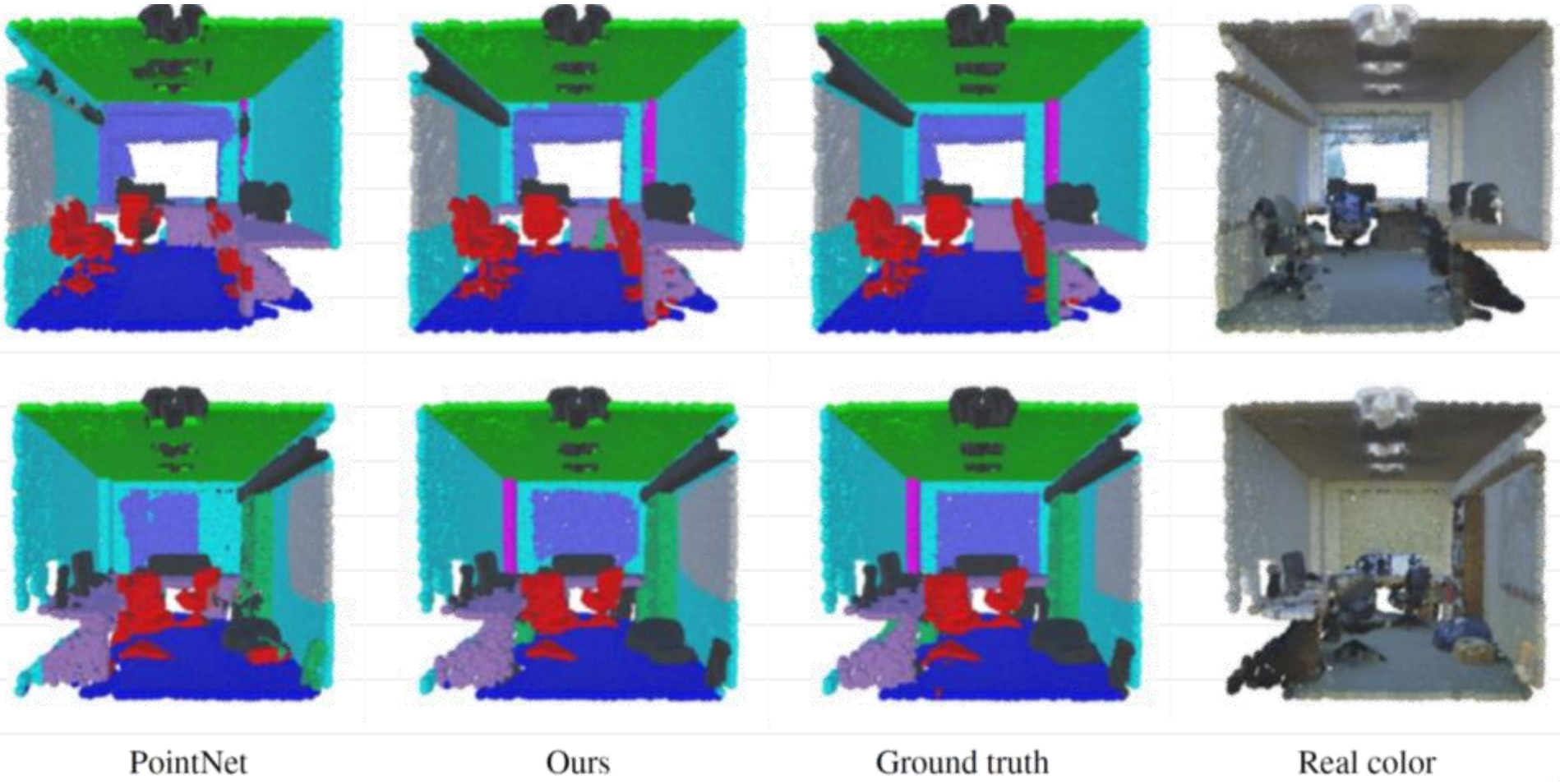
ShapeNet part segmentation benchmark:



[Wang et al. CVPR18]

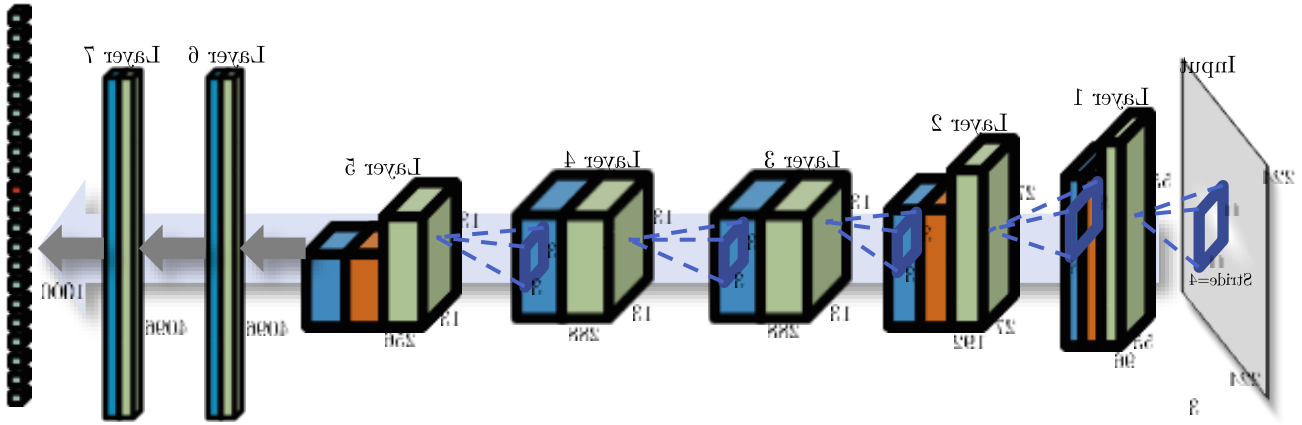
Dynamic graph CNN

Semantic segmentation results:



[Wang et al. CVPR18]

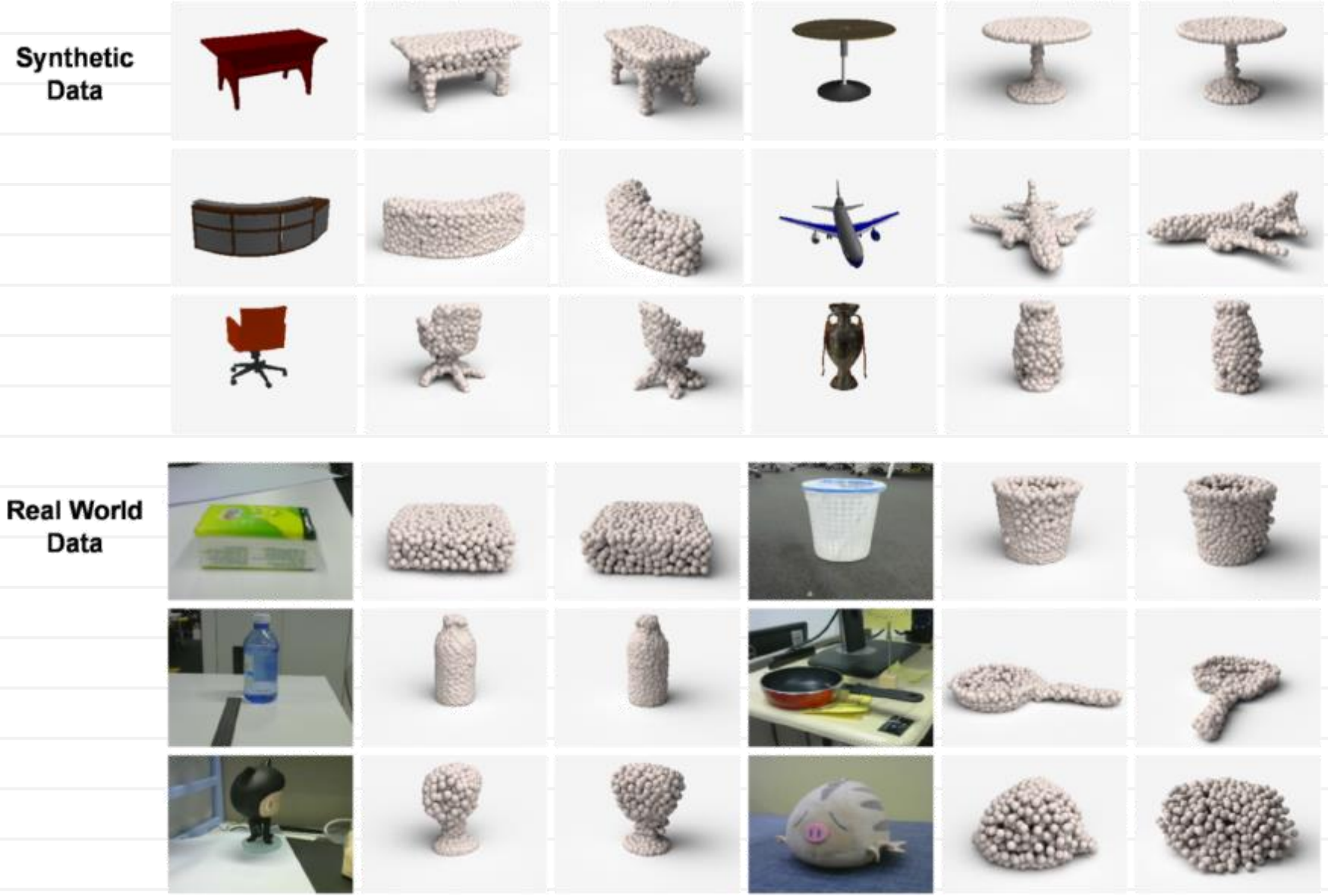
General motivation



This is ConvNet
It does generation very well
Be like ConvNet

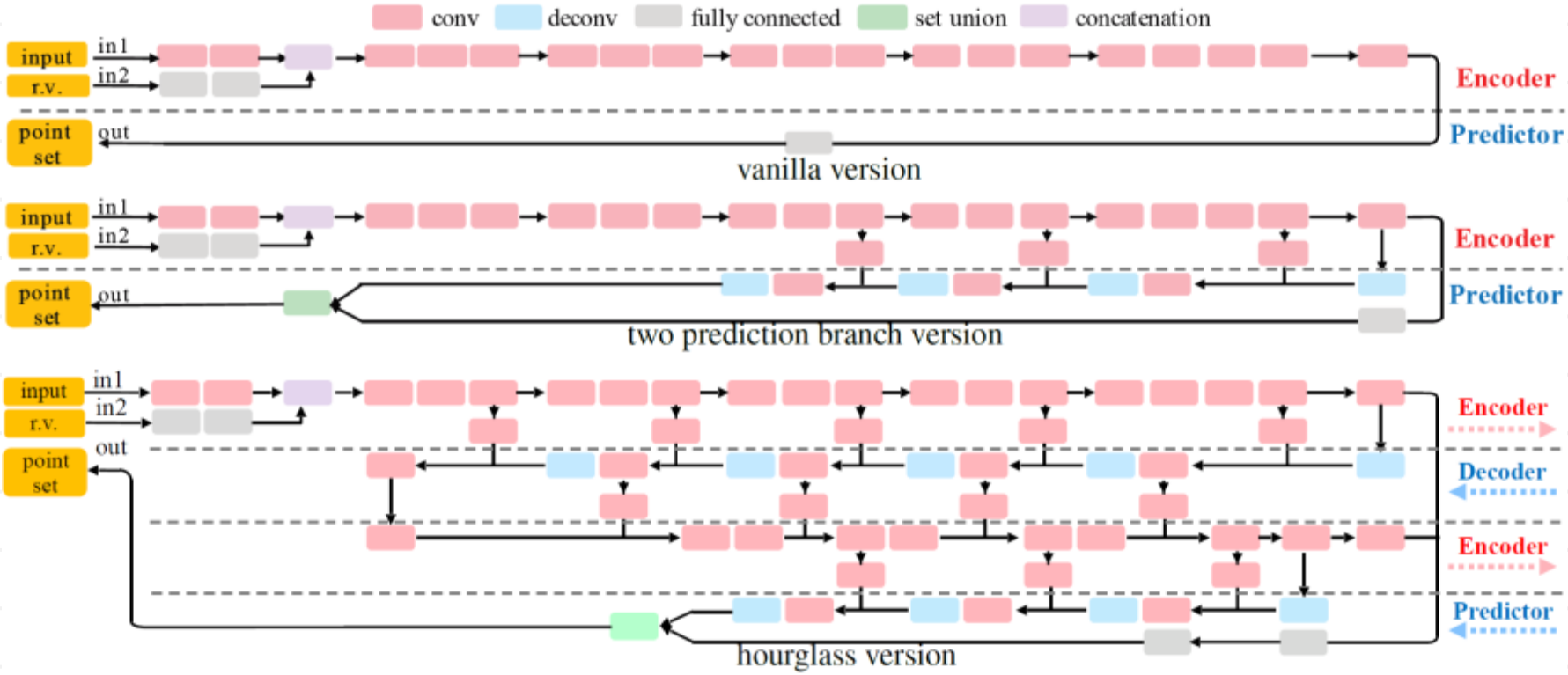
Point Set Generation Networks

Task: given an image, output point cloud for the object:



[Fan,Su,Guibas, CVPR17]

Point Set Generation Networks



- Branch 1: raw point coordinates
- Branch 2: point coordinates packed into 2D XYZ maps

[Fan,Su,Guibas, CVPR17]

Point Set Generation Networks

- Branch 1: raw point coordinates
- Branch 2: point coordinates packed into 2D XYZ maps

[Fan,Su,Guibas, CVPR17]

Point Set Generation Networks: losses

Chamfer distance (CD):

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

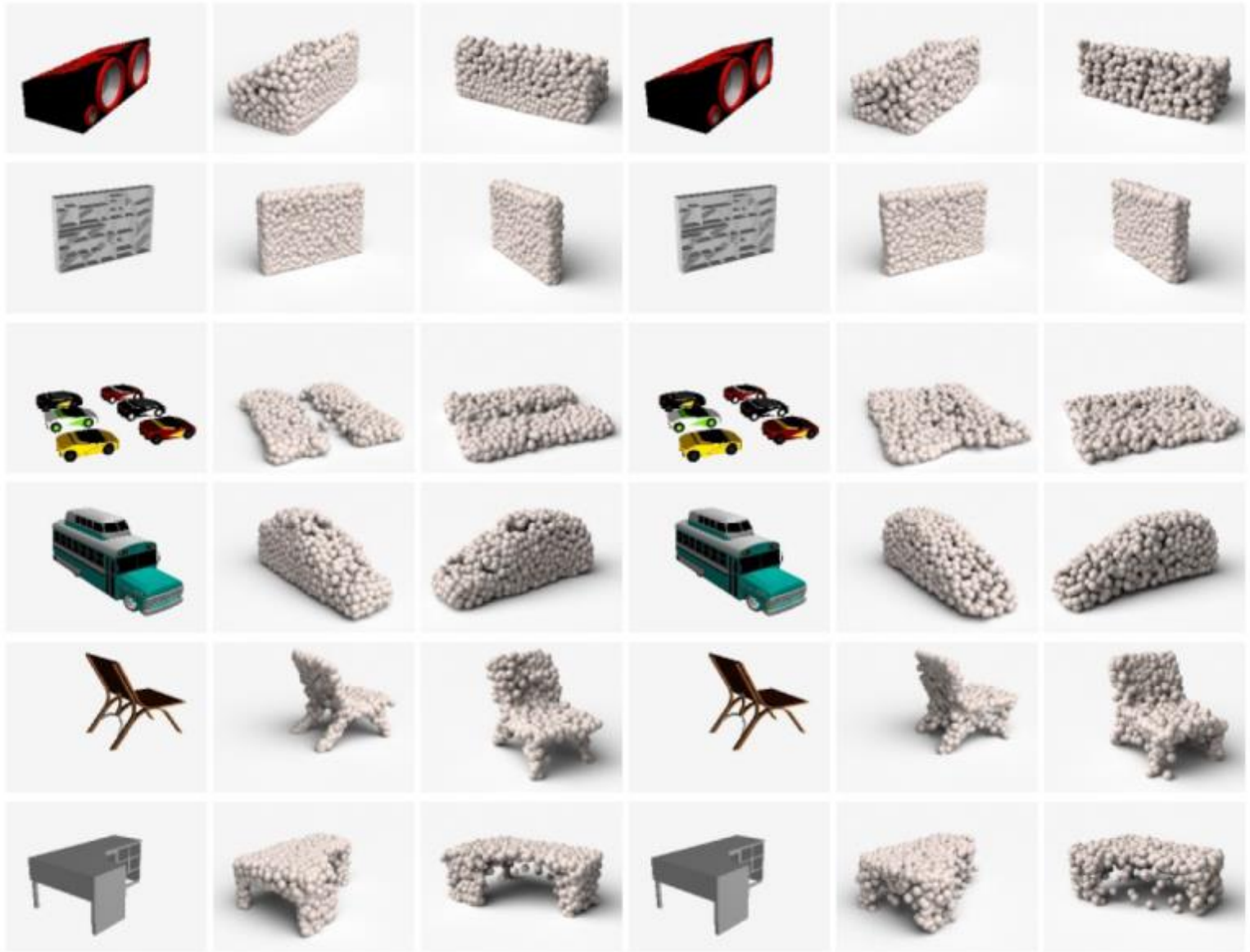
Earth-mover distance (EMD):

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2$$

bijection

[Fan,Su,Guibas, CVPR17]

Point Set Generation Networks: losses



Chamfer distance (CD) Earth-mover distance (EMD)

[Fan,Su,Guibas, CVPR17]

Bibliography

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas:
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. CVPR 2017: 77-85

Charles Ruizhongtai Qi, Li Yi, Hao Su, Leonidas J. Guibas:
PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. NIPS 2017: 5105-5114

Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik G. Learned-Miller:
Multi-view Convolutional Neural Networks for 3D Shape Recognition. ICCV 2015: 945-953

Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, Jan Kautz:
SPLATNet: Sparse Lattice Networks for Point Cloud Processing. CVPR 2018: 2530-2539

Daniel Maturana, Sebastian Scherer:
VoxNet: A 3D Convolutional Neural Network for real-time object recognition. IROS 2015: 922-928

Jonathan Tompson, Murphy Stein, Yann LeCun, Ken Perlin:
Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks. ACM Trans. Graph. 33(5): 169:1-169:10 (2014)

Gyeongsik Moon, Ju Yong Chang, Kyoung Mu Lee:
V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation From a Single Depth Map. CVPR 2018: 5079-5088

Bibliography

Roman Klokov, Victor S. Lempitsky:
Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. ICCV2017: 863-872

Benjamin Graham, Martin Engelcke, Laurens van der Maaten:
3D Semantic Segmentation With Submanifold Sparse Convolutional Networks. CVPR2018: 9224-9232

Liuhao Ge, Yujun Cai, Junwu Weng, Junsong Yuan:
Hand PointNet: 3D Hand Pose Estimation Using Point Sets. CVPR 2018: 8417-8426

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, Pierre Vandergheynst:
Geometric Deep Learning: Going beyond Euclidean data. IEEE Signal Process. Mag. 34(4): 18-42 (2017)

Joan Bruna, Wojciech Zaremba, Arthur Szlam, Yann LeCun:
Spectral Networks and Locally Connected Networks on Graphs. ICLR 2014

Michaël Defferrard, Xavier Bresson, Pierre Vandergheynst:
Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. NIPS2016: 3837-3845

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, Justin M. Solomon:
Dynamic Graph CNN for Learning on Point Clouds. CoRR abs/1801.07829 (2018)

Martin Simonovsky, Nikos Komodakis:
Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. CVPR 2017: 29-38

Haoqiang Fan, Hao Su, Leonidas J. Guibas:
A Point Set Generation Network for 3D Object Reconstruction from a Single Image. CVPR 2017: 2463-2471