FFI Forsvarets forskningsinstitutt

Advanced 3D segmentation

Sigmund Rolfsjord

Today's lecture

Different ways to work with 3D data:

- Point clouds
- Grids
- Graphs

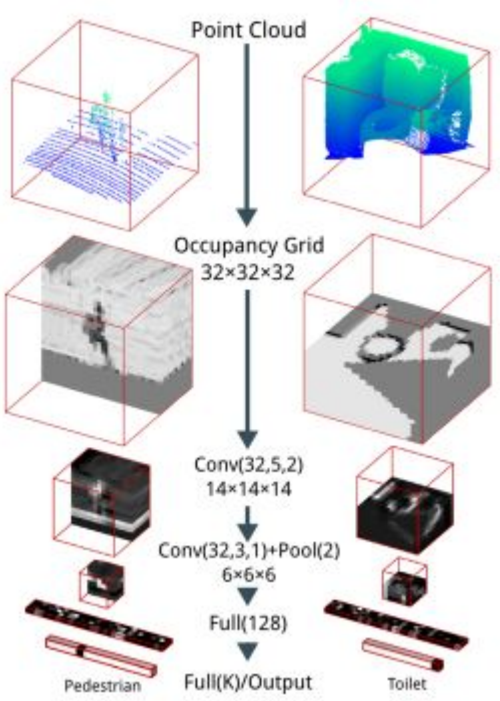
Curriculum:

SEGCloud: Semantic Segmentation of 3D Point Clouds

Multi-view Convolutional Neural Networks for 3D Shape Recognition

Deep Parametric Continuous Convolutional Neural Networks

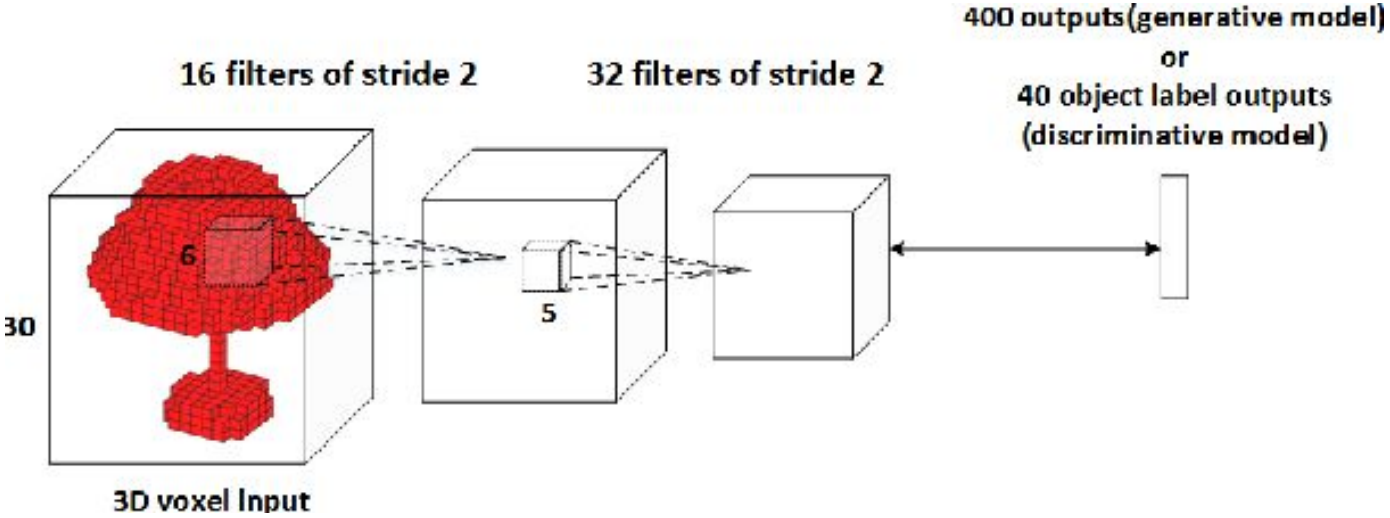
Processing 3D data with deep networks



[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition]

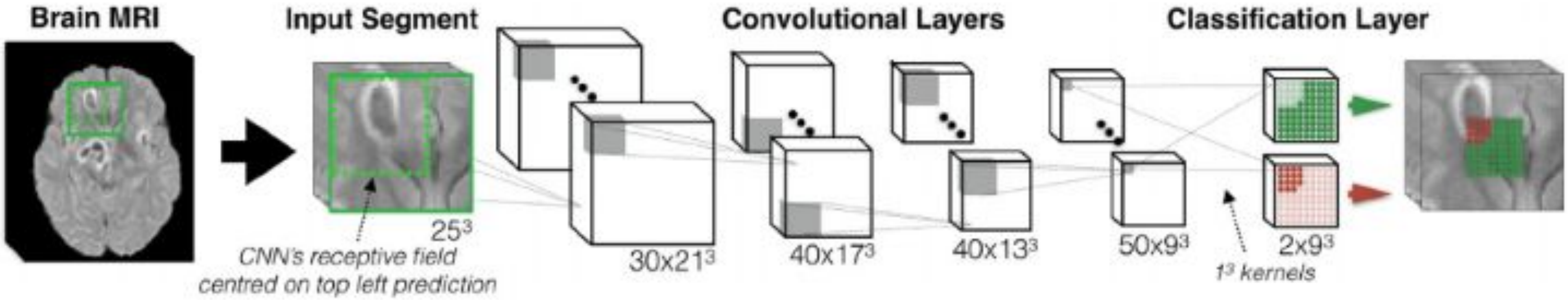
3D convolutions on voxelized data

3D Convolutions



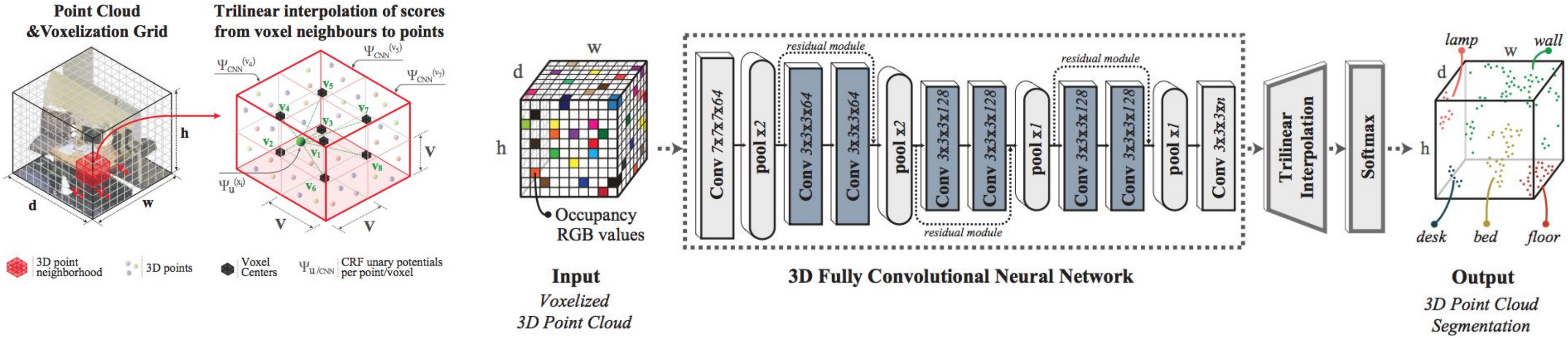
When voxelization works

- Dense images
- Small images



[Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation]

CloudSeg



[SEGCloud: Semantic Segmentation of 3D Point Clouds]

Table 2: **Results on the Large-Scale 3D Indoor Spaces Dataset (S3DIS)**

Method	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter	**mIOU**	**mAcc**
PointNet [53]	88.80	**97.33**	69.80	**0.05**	3.92	**46.26**	10.76	52.61	58.93	40.28	5.85	**26.38**	33.22	41.09	48.98
3D-FCNN-TI(Ours)	**90.17**	96.48	**70.16**	0.00	11.40	33.36	21.12	**76.12**	70.07	57.89	37.46	11.16	**41.61**	47.46	54.91
SEGCloud (Ours)	90.06	96.05	69.86	0.00	**18.37**	38.35	**23.12**	75.89	**70.40**	**58.42**	**40.88**	12.96	41.60	**48.92**	**57.35**

Problems with voxelization

- Memory (1024x1024x1024x1024)
- Lots of zeros
- Field-of-view
- Resolution

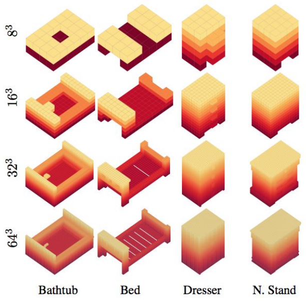
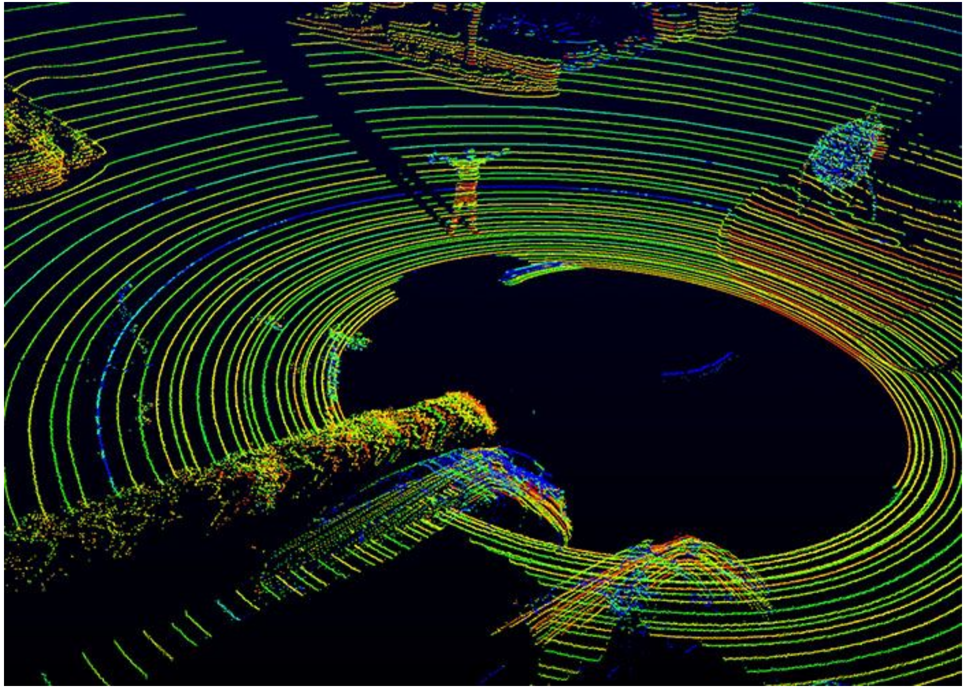


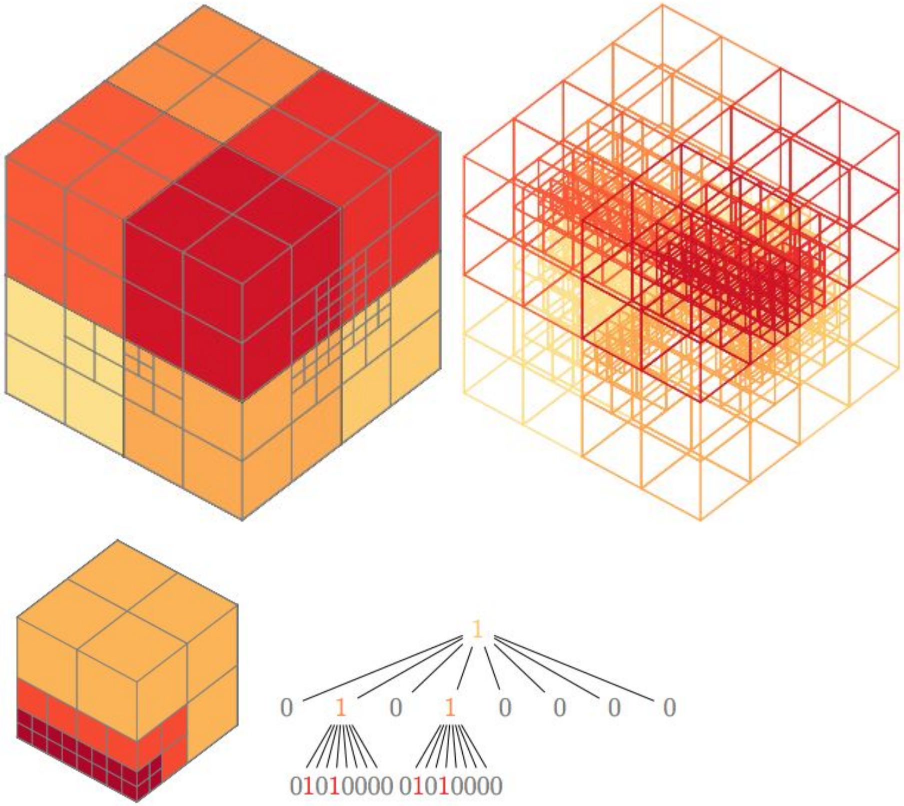
Figure 8: **Voxelized 3D Shapes from ModelNet10.**



OctNets

More memory efficient 3D convolutions for sparse data.

- Irregular grid
- Iteratively split
    - 8 children
    - depth 3



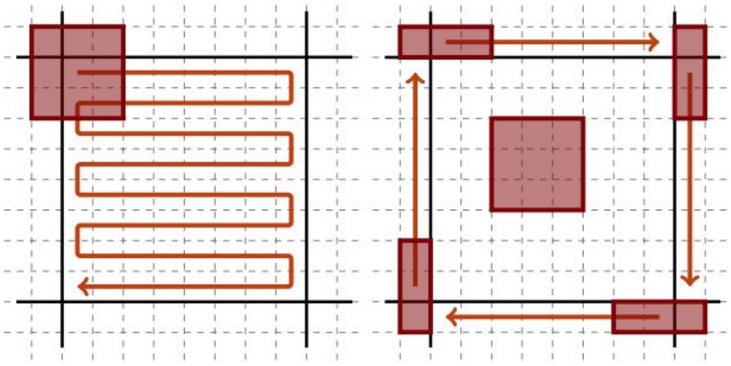
(a) Shallow Octree (b) Bit-Representation

OctNet: Learning Deep 3D Representations at High Resolutions

OctNets

More memory efficient 3D convolutions for sparse data.

- Irregular grid
- Iteratively split
    - 8 children
    - depth 3
- Implementation of 72 bit tree on GPU can be used
- GPU can index and convolve only important locations



(a) Standard Convolution (b) Efficient Convolution

OctNets

- Memory and runtime efficient for larger inputs
- ModelNet10: Resolution is not that important

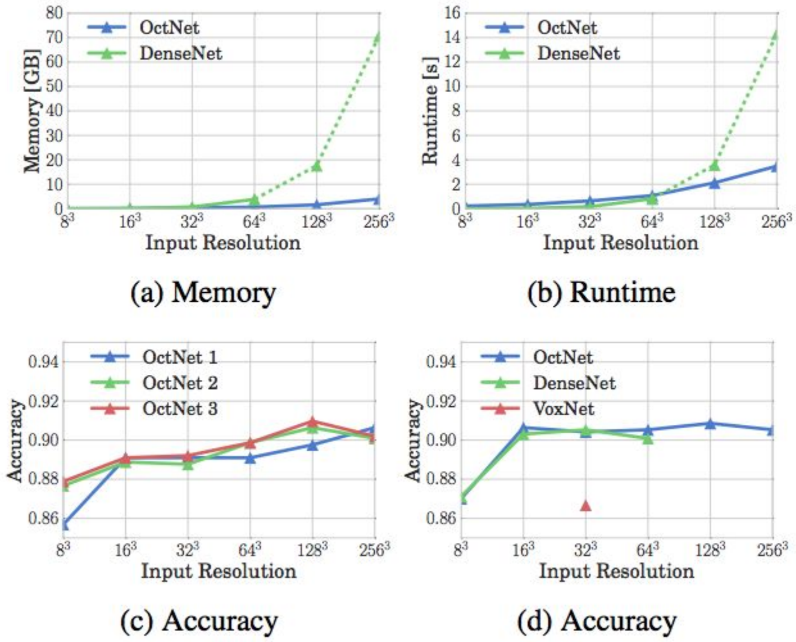


Figure 7: **Results on ModelNet10 Classification Task.**

OctNets

- Memory and runtime efficient for larger inputs
- ModelNet10: Resolution is not that important

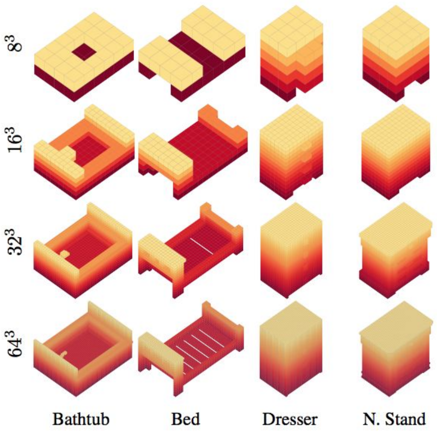


Figure 8: **Voxelized 3D Shapes from ModelNet10.**

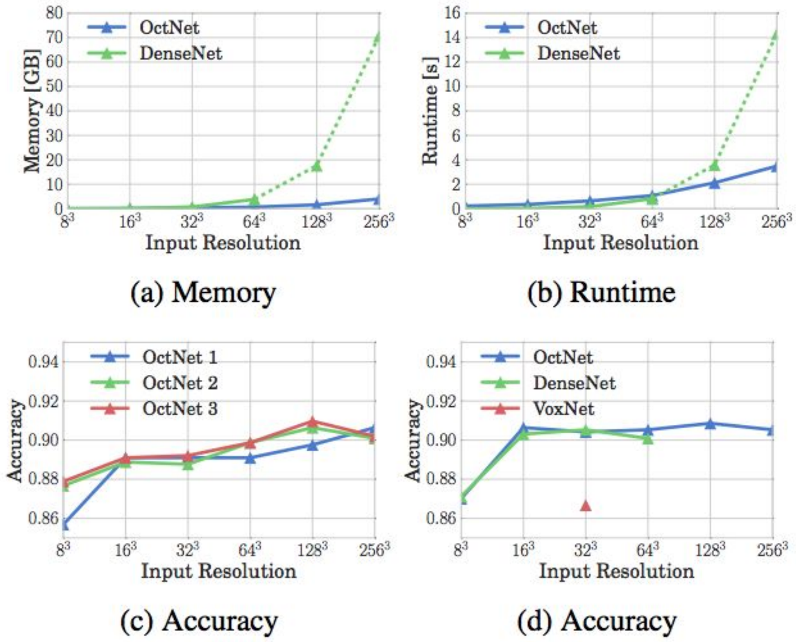


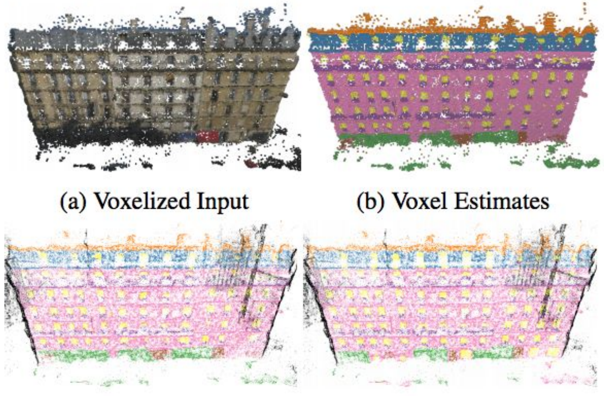
Figure 7: **Results on ModelNet10 Classification Task.**

OctNets

OctNet is efficent on larger relatively sparse point clouds

	Average	Overall	IoU
Riemenschneider et al. [38]	-	-	42.3
Martinovic et al. [29]	-	-	52.2
Gadde et al. [13]	68.5	78.6	54.4
OctNet 64^3	60.0	73.6	45.6
OctNet 128^3	65.3	76.1	50.4
OctNet 256^3	**73.6**	**81.5**	**59.2**

Table 1: **Semantic Segmentation on RueMonge2014.**

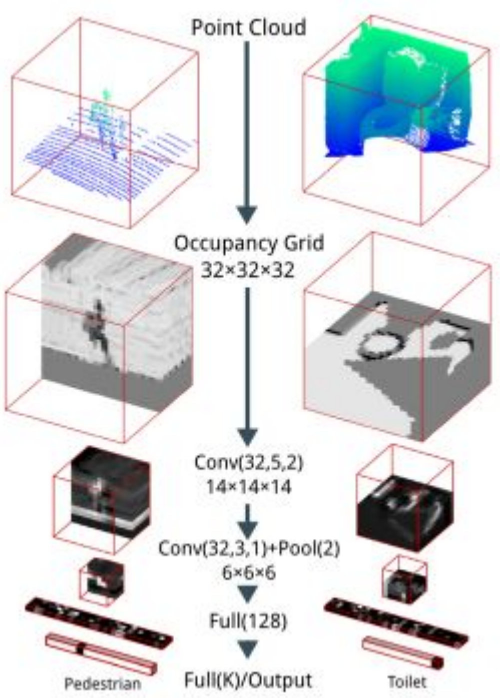


(a) Voxelized Input (b) Voxel Estimates

(c) Estimated Point Cloud (d) Ground Truth Point Cloud

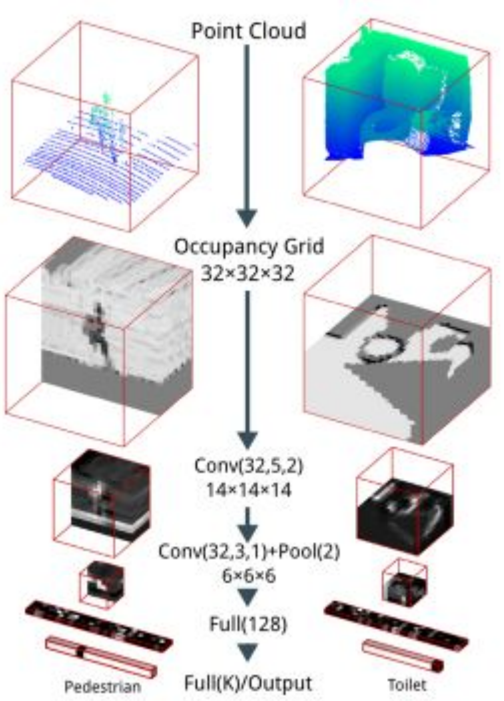
Figure 12: **OctNet** 256^3 **Facade Labeling Results.**

Processing 3D data with deep networks

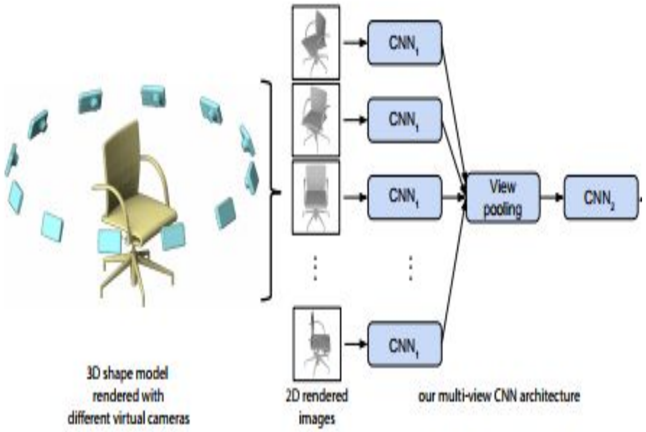


[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition]

Processing 3D data with deep networks



[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition]

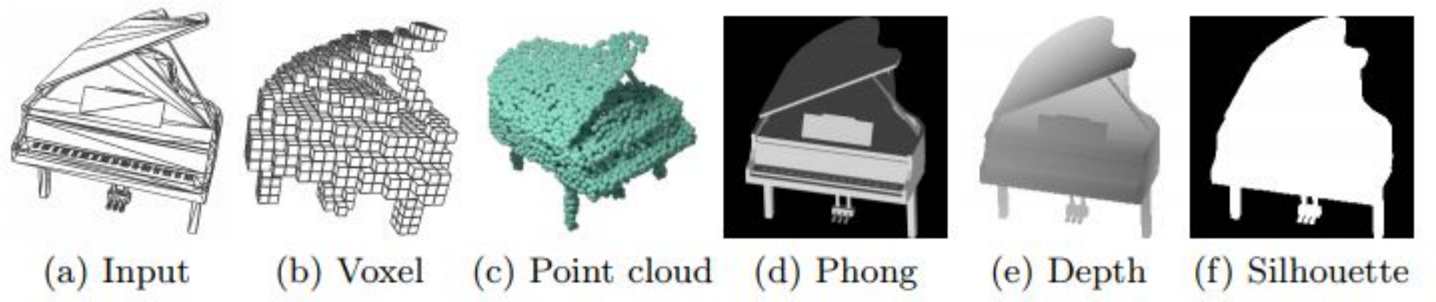


[Multi-view Convolutional Neural Networks for 3D Shape Recognition]

2D convolutions on projections

Multi-View - ShapeNet classification

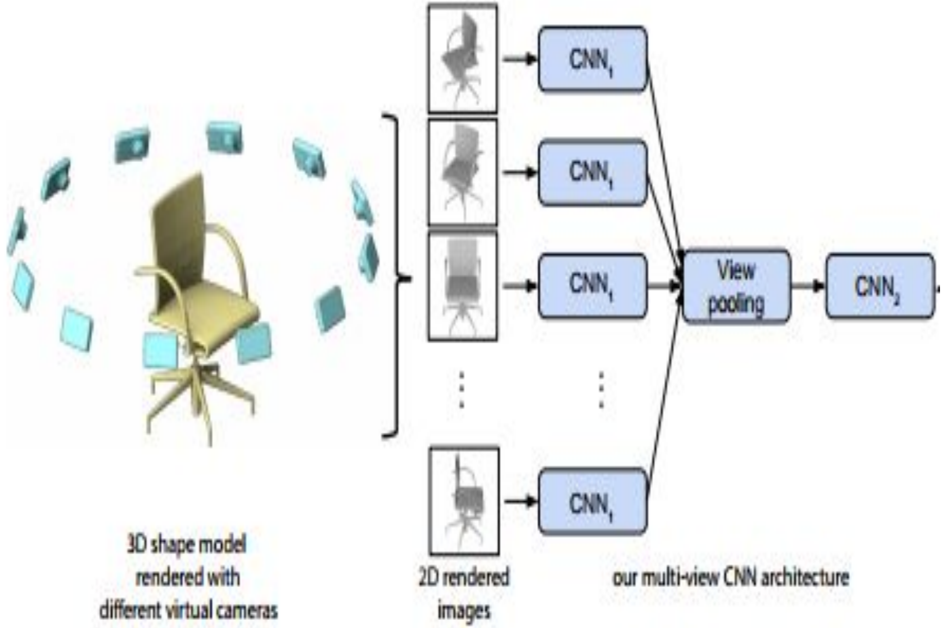
3D models
common objects



www.shapenet.org

A Deeper Look at 3D Shape Classifiers

Multi-View

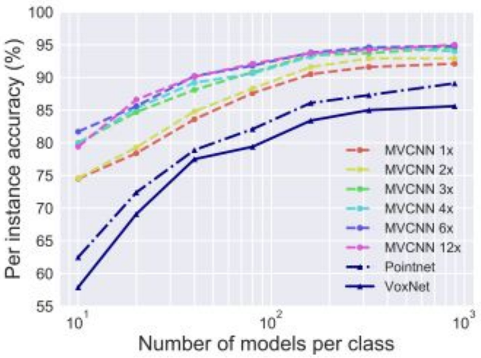


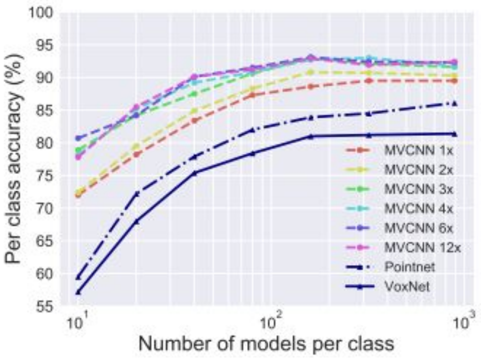
[Multi-view Convolutional Neural Networks for 3D Shape Recognition]

Multi-View

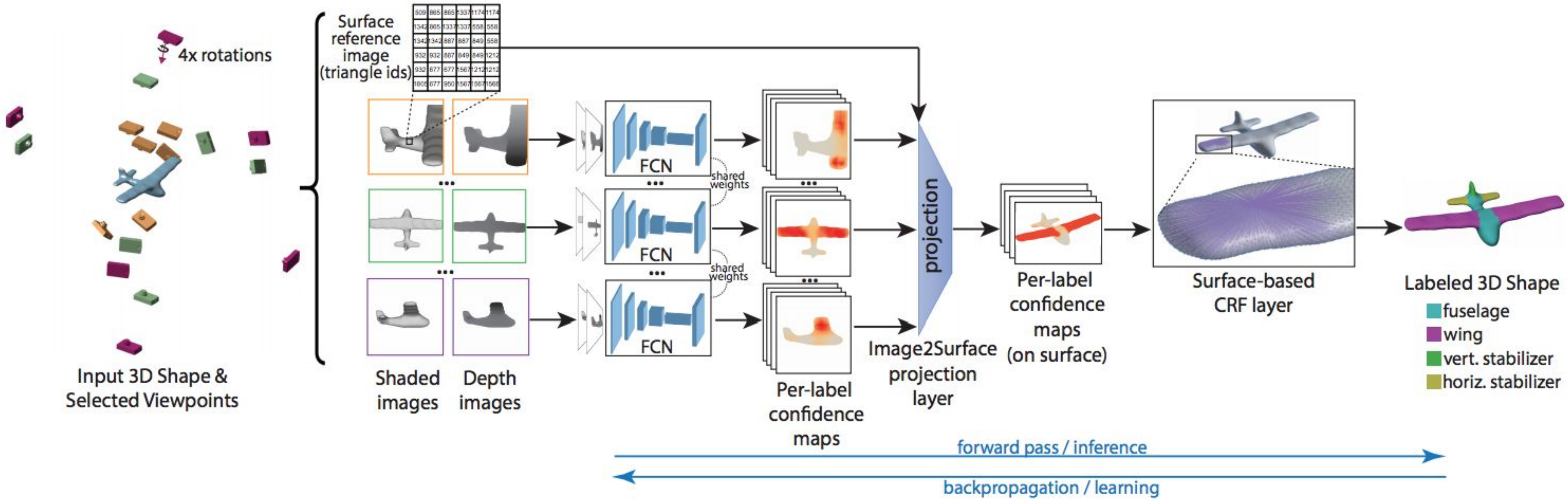
- Simple solution is the best solution
- More views are better, but not by a lot

Model	Rendering	Full training/test Per class	Full training/test Per instance	80/20 training/test Per class	80/20 training/test Per instance
VGG-M	Shaded from [31]	-	-	89.9	89.9
VGG-M	Shaded from [31] (80×)	-	-	90.1	90.1
VGG-11	Shaded from [31]	-	-	89.1	89.1
VGG-11	Shaded	**92.4**	**95.0**	**92.4**	**92.4**
VGG-11	Depth	89.8	91.6		
VGG-11	Shaded + Depth	94.7	96.2		
VGG-11	Silhouettes	90.7	93.6		
AlexNet	Sphere rendering (20×)	89.7	92.0		
AlexNet-MR	Sphere rendering (20×)	91.4	93.8		



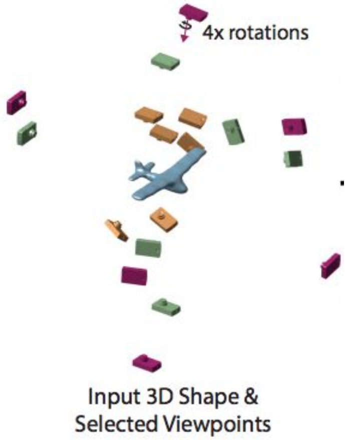


Multi-View - segmentation



[3D Shape Segmentation with Projective Convolutional Networks]

Multi-View - segmentation



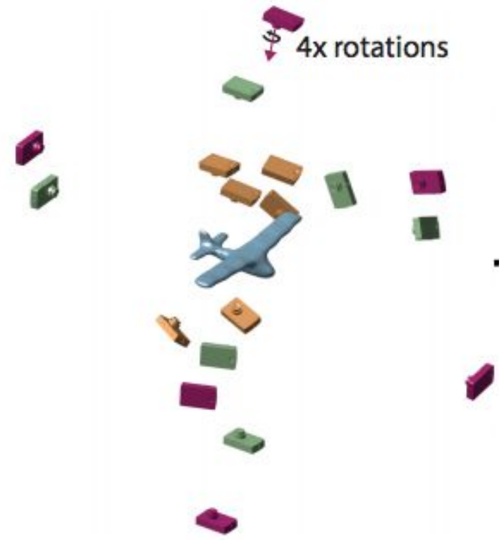
Multi-View - segmentation

Finding viewpoints, by maximising area covered

- Sample surface points (1024)
- Place camera at each surface normal

For each surface normal

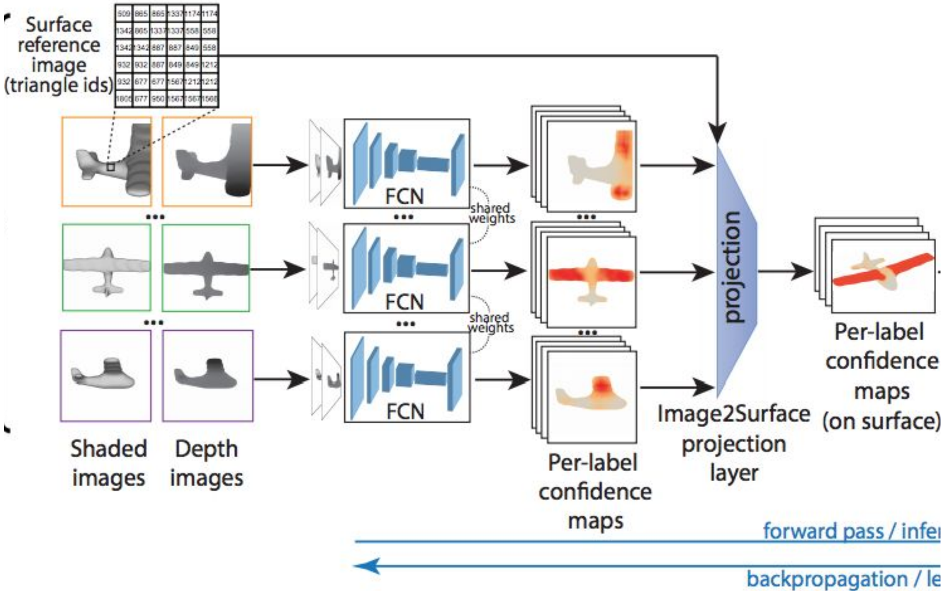
- Rasterize view, and choose rotation with maximally area covered
- Ignore already visible points
- Continue til all surface points are covered



Input 3D Shape & Selected Viewpoints

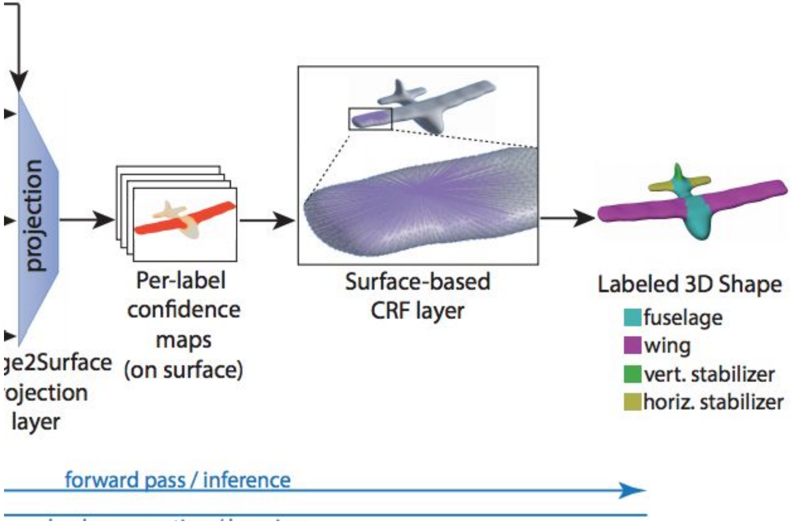
Multi-View - segmentation

- Run depth images through “standard” segmentation networks
- For each view: project/shoot back the segmented labed onto the model
- Average overlapping regions



Multi-View - segmentation

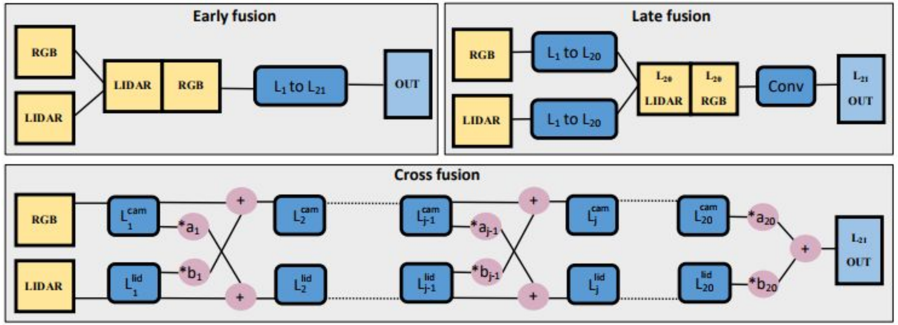
- Run a Conditional Random Field (CRF) over the surface
    - Promotes consistency
    - Makes sure every pixel is labelled
    - Fixes problems due to upsampling
- CRF is **not** in the **curriculum**, but:
    - Loop over neighbouring surfaces
    - Weight angles, distances, and label differences
    - Learns the weights, through backpropagation,



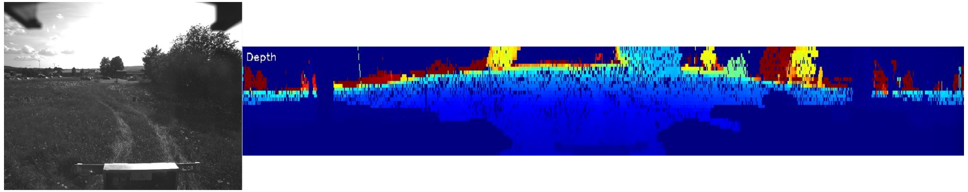
Multi-View / Single-View

Single depth image:

- Depth-rays from one position
- Fusion with image can be a challenge
- Late/cross fusion often best strategy
    - Probably due to alignment issues

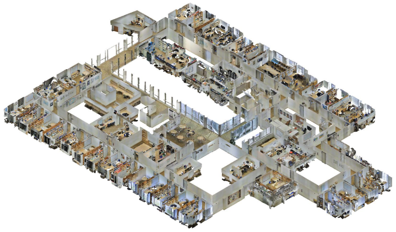


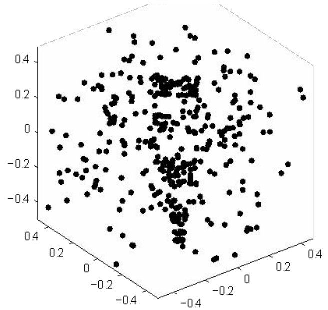
[LIDAR-Camera Fusion for Road Detection Using Fully Convolutional Neural Networks]

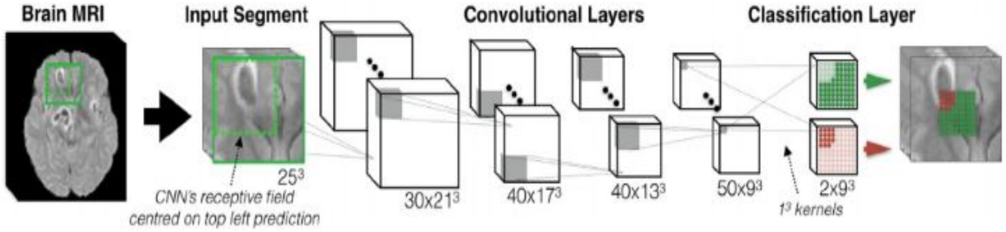


When does multi-view not work?

- Large complex point cloud
  - Hard to choose view-points
- Dense point-cloud
- Noisy/sparse point cloud
  - Convolutions makes, little sense, as the points in your kernel have very different depth.
  - “Randomness” depending on view-point
  - Hard/impossible to train

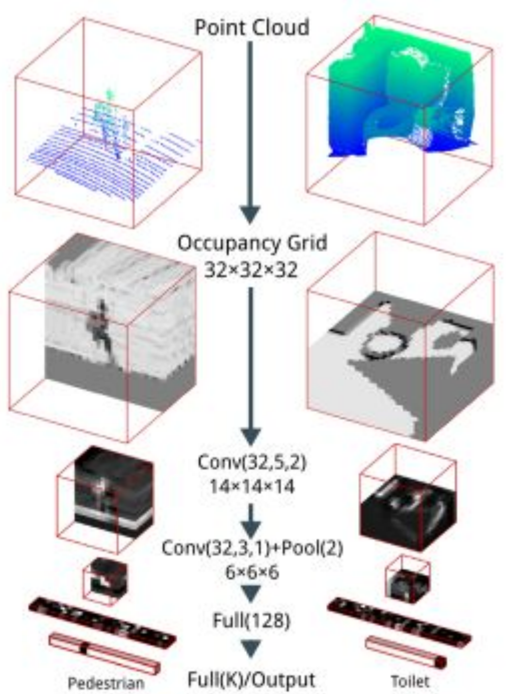




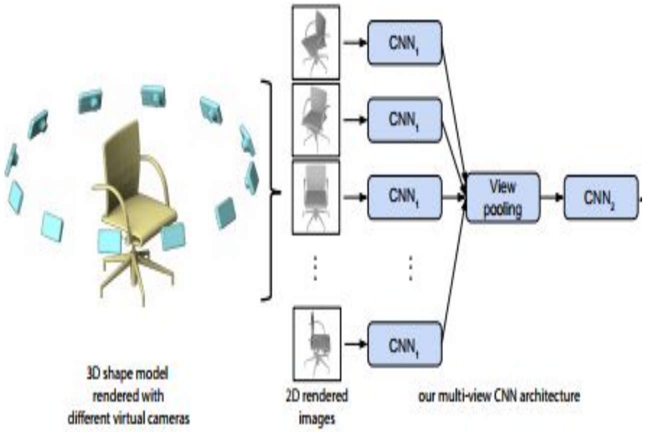


[Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation]

Processing 3D data with deep networks



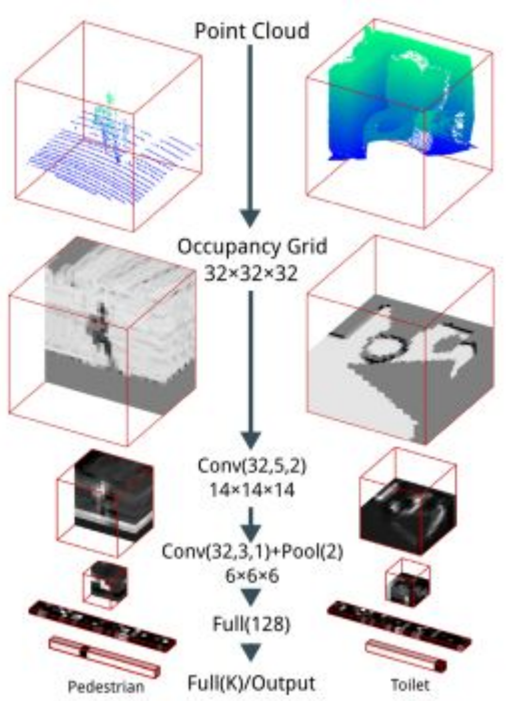
[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition]()



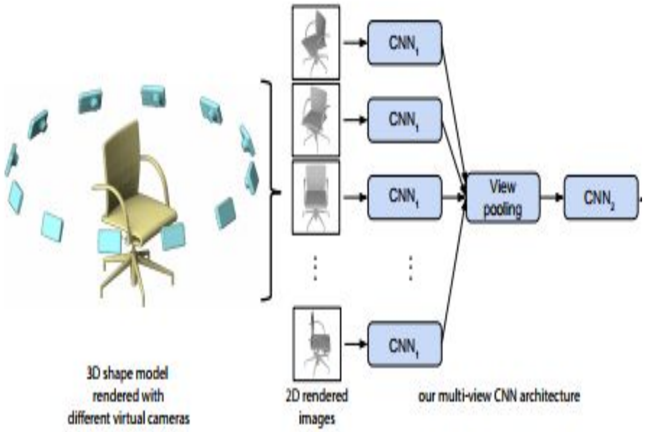
[Multi-view Convolutional Neural Networks for 3D Shape Recognition]()

Processing 3D data with deep networks

[PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation](#)



[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition](#)



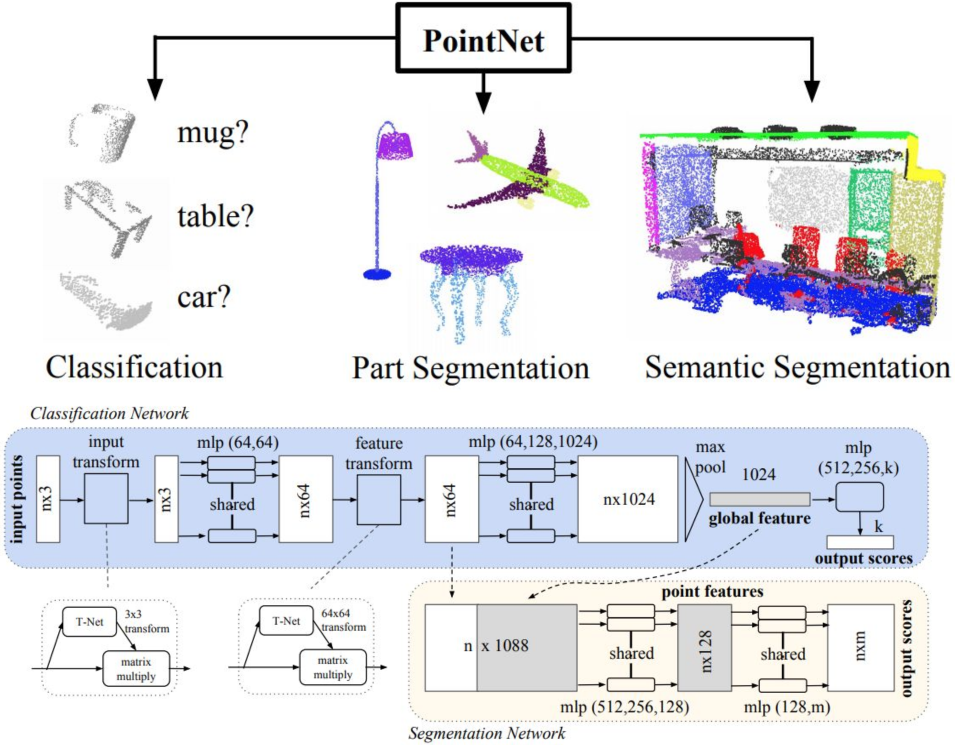
[Multi-view Convolutional Neural Networks for 3D Shape Recognition](#)

FFI

Direct point cloud processing

PointNet

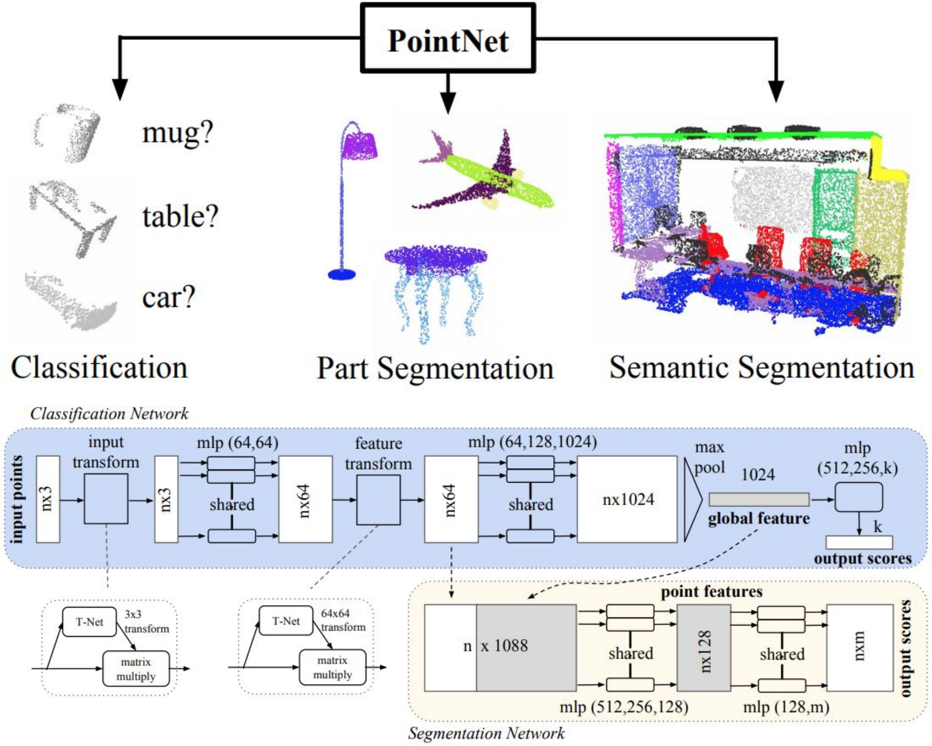
- Learning directly on point clouds
- No direct local information
    - Perhaps only global?
    - Ignoring similar points



[PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation]()

PointNet

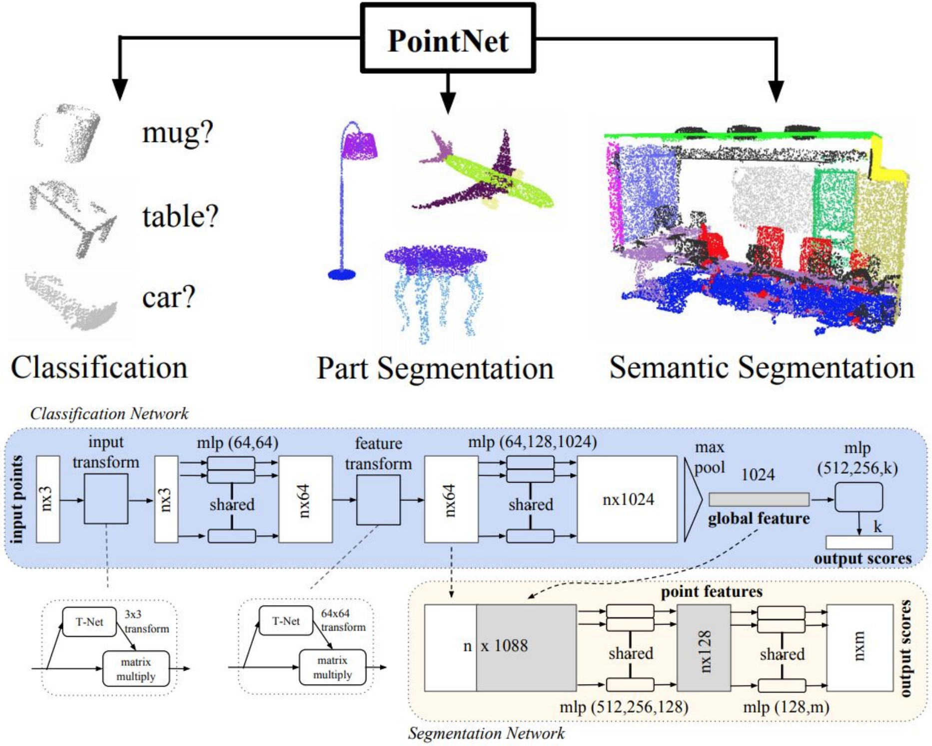
1. Transforms each point into high dimension (1024) with same transform.
2. Aggregates with per-channel max-pool
3. Uses aggregate to find new transform and and run transform
4. Then run per point neural nett
5. Repeat for *n* layers
6. Finally aggregate again with maxpool
7. Run fully-connected layer on aggregated results



PointNet

Why does this work? (speculations):

- Forced to choose “a few” important points
- Transform based on the kind of points have been seen



PointNet https://github.com/charlesq34/pointnet/blob/master/models/pointnet_cls.py

```
net = tf_util.conv2d(input_image, 64, [1,3],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv1', bn_decay=bn_decay)
net = tf_util.conv2d(net, 64, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv2', bn_decay=bn_decay)

with tf.variable_scope('transform_net2') as sc:
    transform = feature_transform_net(net, is_training, bn_decay, K=64)
end_points['transform'] = transform
net_transformed = tf.matmul(tf.squeeze(net, axis=[2]), transform)
net_transformed = tf.expand_dims(net_transformed, [2])
```

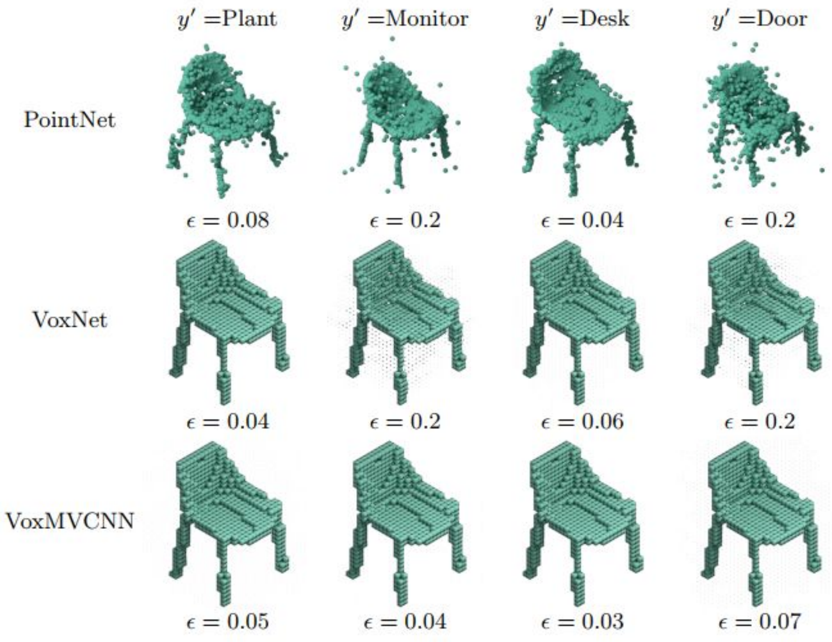
```
net = tf_util.conv2d(net_transformed, 64, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv3', bn_decay=bn_decay)
net = tf_util.conv2d(net, 128, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv4', bn_decay=bn_decay)
net = tf_util.conv2d(net, 1024, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv5', bn_decay=bn_decay)

# Symmetric function: max pooling
net = tf_util.max_pool2d(net, [num_point,1],
                         padding='VALID', scope='maxpool')

net = tf.reshape(net, [batch_size, -1])
net = tf_util.fully_connected(net, 512, bn=True, is_training=is_training,
                              scope='fc1', bn_decay=bn_decay)
```

PointNet

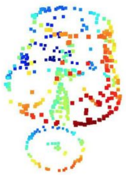
Adverserial robustness:

- With aggregation based on max-pool it may not rely on all points (max 1024 for each transform)
- Small changes will not have much effect
- Robust to deformation and noise

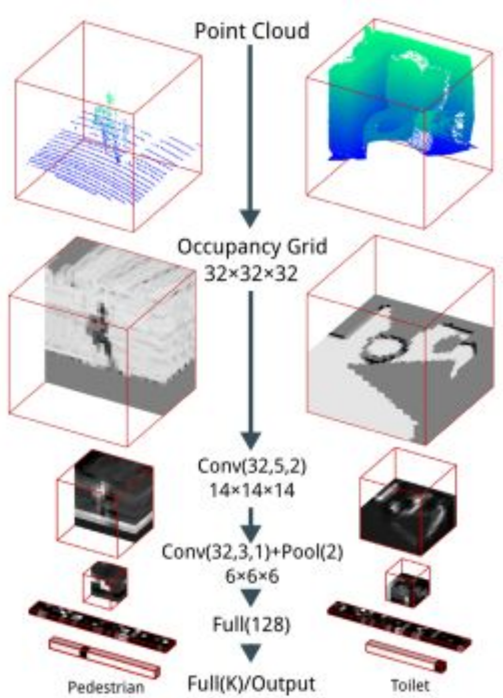
- Not good at detecting small details



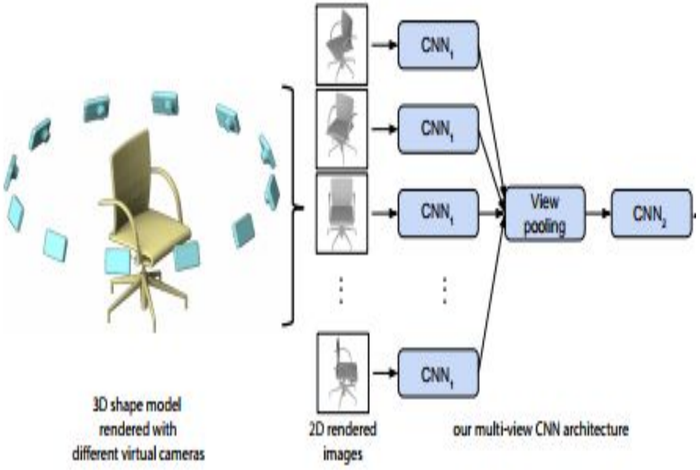
Processing 3D data with deep networks



[PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation](#)



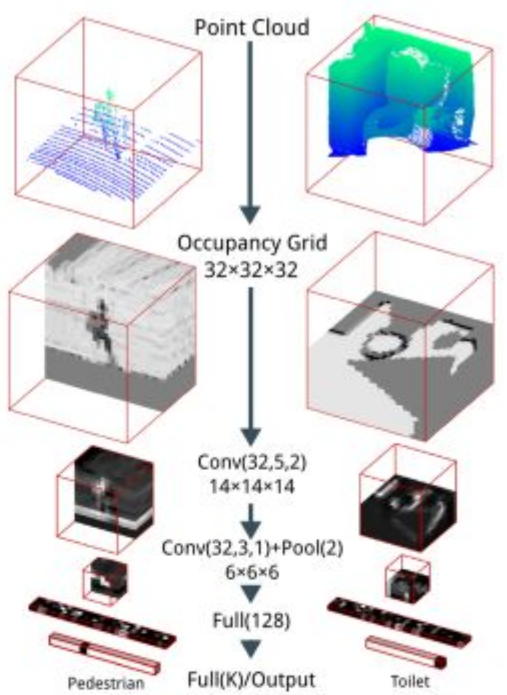
[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition](#)



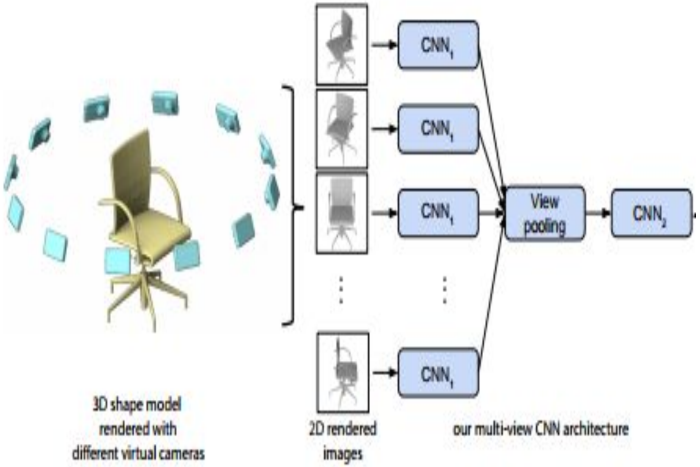
[Multi-view Convolutional Neural Networks for 3D Shape Recognition](#)

Processing 3D data with deep networks

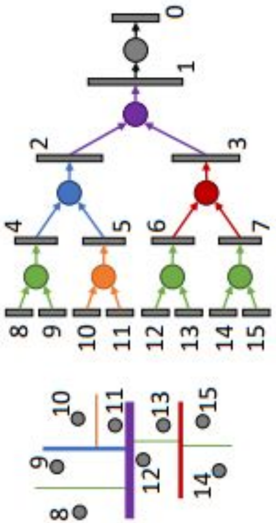
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation



VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition



Multi-view Convolutional Neural Networks for 3D Shape Recognition

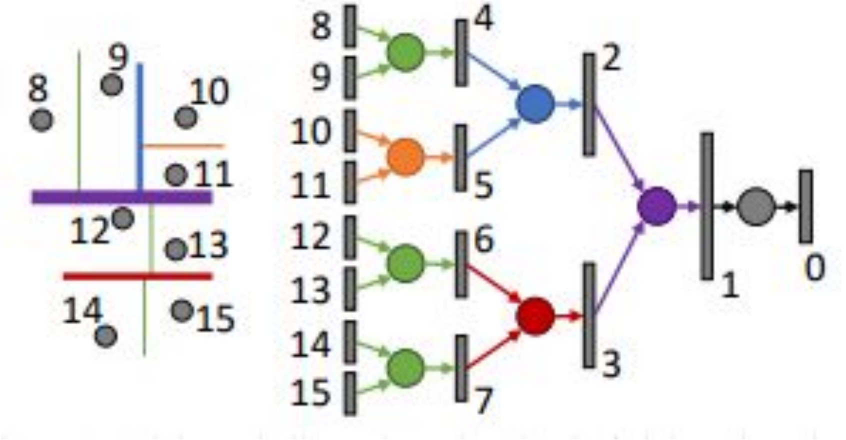


Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models

Abstraction of convolutions

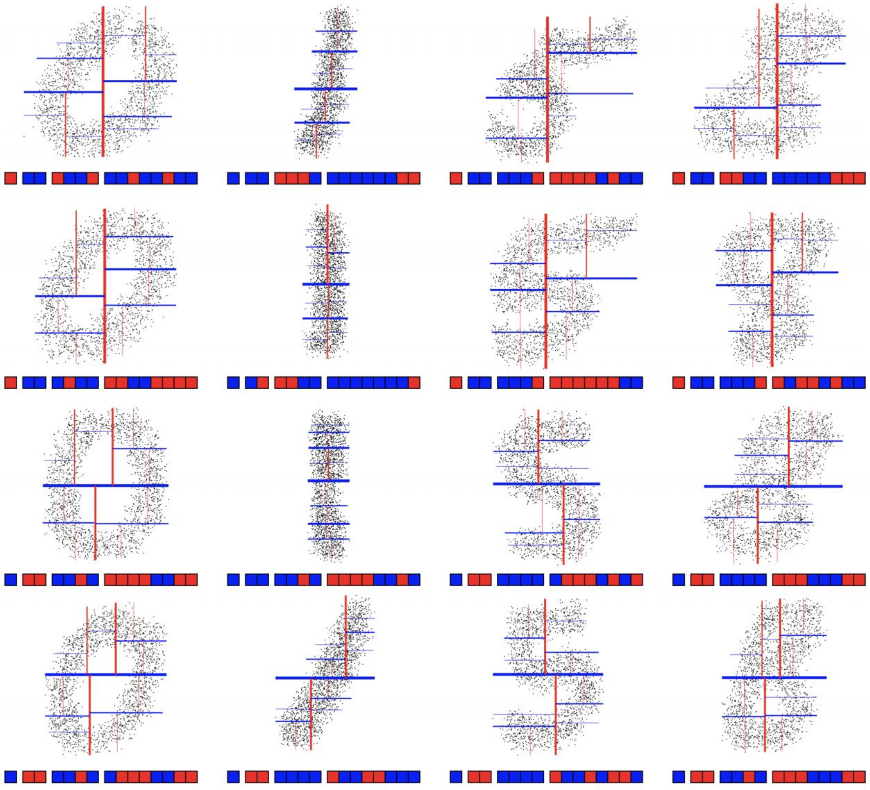
Kd-networks

“Convolutions” over sets



Kd-networks

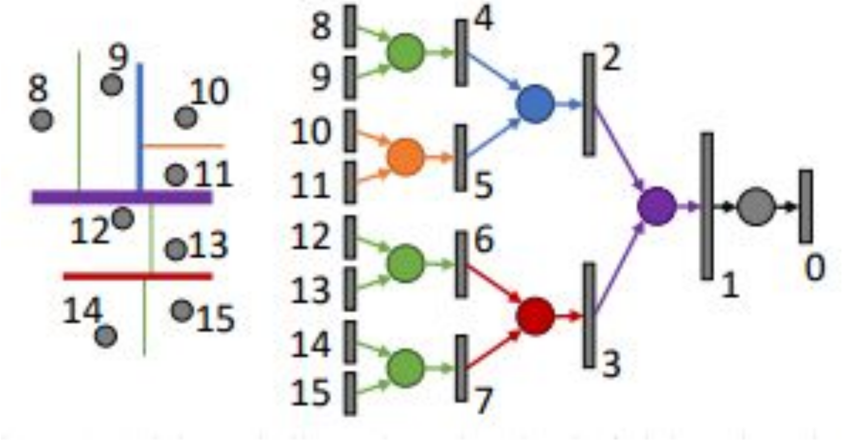
- Fixed number of points $N = 2^D$
- 3D points {x, y, z}
- Split along widest axis
- Choose split to divide data set in two



Kd-networks

- Each node have a representation vector:

$$\mathbf{v}_i = \begin{cases} \phi(W_{\mathbf{x}}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{\mathbf{x}}^{l_i}), & \text{if } d_i = \mathbf{x} \\ \phi(W_{\mathbf{y}}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{\mathbf{y}}^{l_i}), & \text{if } d_i = \mathbf{y} \\ \phi(W_{\mathbf{z}}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{\mathbf{z}}^{l_i}), & \text{if } d_i = \mathbf{z} \end{cases}$$



Final layer is a fully connected layers

Shared weights for nodes splitting along same dimension at same level.

Not shared for left and right node.

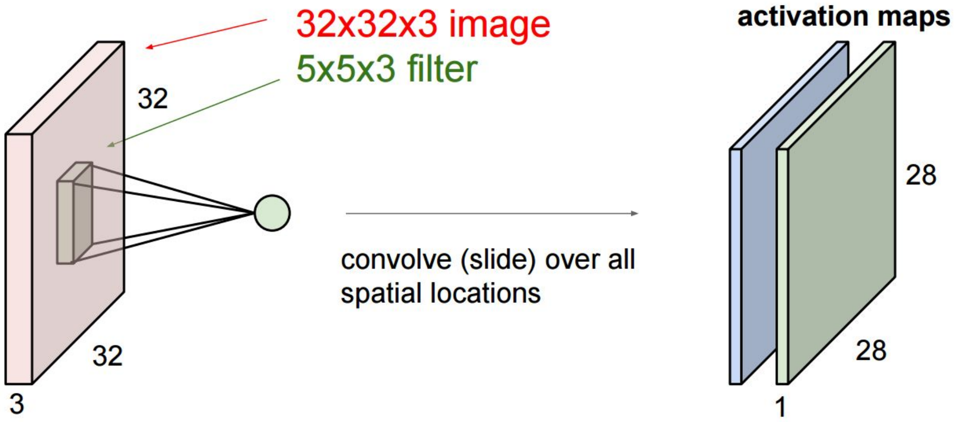
Kd-networks

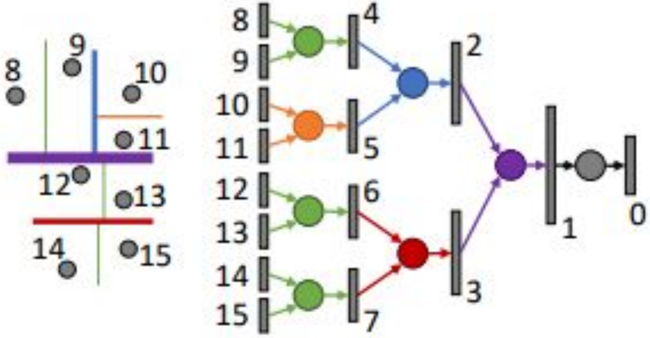
Convolutions over sets

Running kernel over neighbours in group.

Shared weights for nodes splitting along same dimension at same level.

Not shared for left and right node

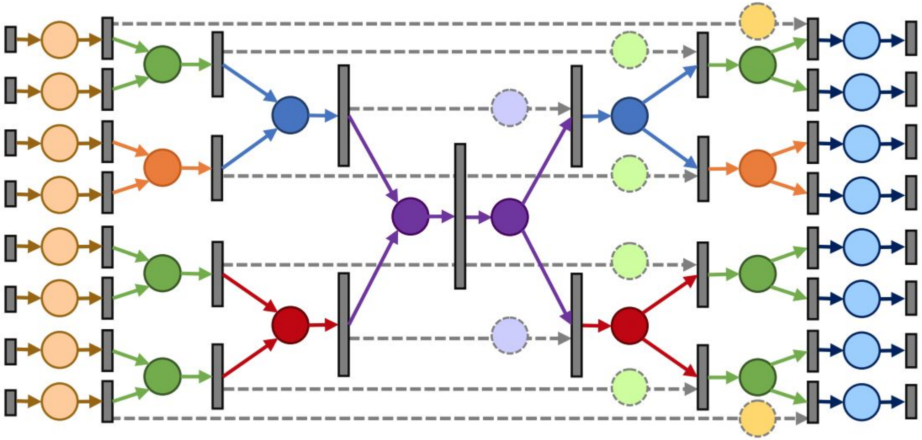




Kd-networks - segmentation

- One different weight matrix for each direction
- Shared between nodes, depending on split direction
- Skip-connection matrix shared between all nodes in a layer

- Final result: Use {x, y, z} from corresponding input nodes



$$\tilde{\mathbf{v}}_{c_1(i)} = \phi([\tilde{W}^{l_i}_{d_{c_1(i)}} \tilde{\mathbf{v}}_i + \tilde{\mathbf{b}}^{l_i}_{d_{c_1(i)}}; S^{l_i} \mathbf{v}_{c_1(i)} + \mathbf{t}^{l_i}])$$

$$\tilde{\mathbf{v}}_{c_2(i)} = \phi([\tilde{W}^{l_i}_{d_{c_2(i)}} \tilde{\mathbf{v}}_i + \tilde{\mathbf{b}}^{l_i}_{d_{c_2(i)}}; S^{l_i} \mathbf{v}_{c_2(i)} + \mathbf{t}^{l_i}])$$

Kd-networks - results

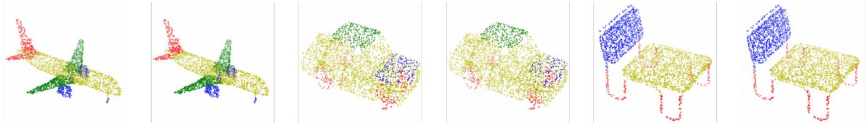
- Slightly worse than Multi-View on 3D model classification
- More flexible: can be used on sparse point clouds etc.

Classification

ModelNet	10-class		40-class	
Accuracy averaging	class	instance	class	instance
3DShapeNets [36]	83.5	-	77.3	-
MVCNN [31]	-	-	90.1	-
FusionNet [12]	-	93.1	-	90.8
VRN Single [4]	-	93.6	-	91.3
MVCNN [21]	-	-	89.7	92.0
PointNet [20]	-	-	86.2	89.2
OctNet [23]	90.1	90.9	83.8	86.5
ECC [29]	90.0	90.8	83.2	87.4
Kd-Net (depth 10)	92.8	93.3	86.3	90.6
Kd-Net (depth 15)	93.5	94.0	88.5	91.8
VRN Ensemble [4]	-	97.1	-	95.5
MVCNN-MultiRes [21]	-	-	91.4	93.8

Segmentation

	mean	aero plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skate board	table
Yi [37]	81.4	81.0	78.4	77.7	75.7	87.6	61.9	92.0	85.4	82.5	95.7	70.6	91.9	85.9	53.1	69.8	75.3
3DCNN [20]	79.4	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6	74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
PointNet [20]	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
Kd-network	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3

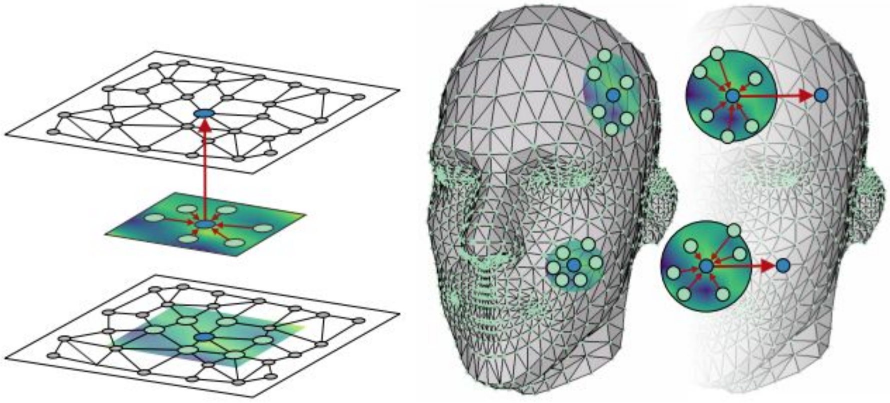


Graph Convolutional operators

Based on [Geometric deep learning on graphs and manifolds using mixture model CNNs]

Generalising convolutions to irregular graphs, with **two base concepts**

- Parametric kernel function
- Pseudo-coordinates



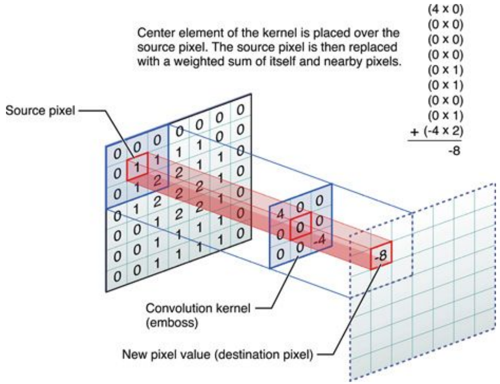
(a) Filtering of graphs (b) Filtering of meshes

SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels

Graph convolutions - parametric kernel

Basic CNN weight function *w(x, y):*

Look-up-table for neighbouring directions {dx=1, dy=0}, {dx=0, dy=0}, etc.



[Apple: performing convolution operations]

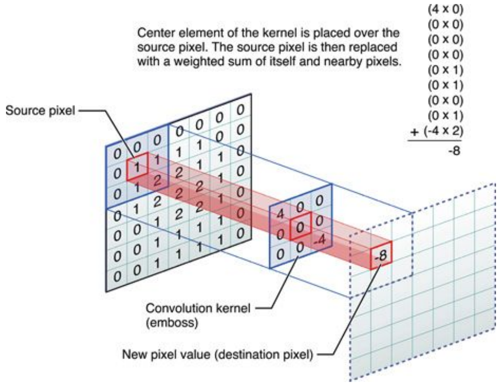
Graph convolutions - parametric kernel

Basic CNN weight function *w(x, y):*

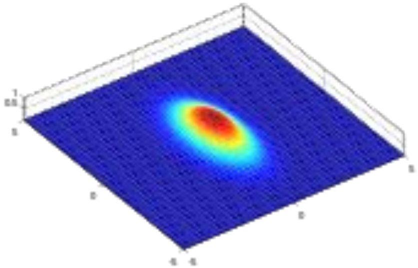
Look-up-table for neighbouring directions {dx=1, dy=0}, {dx=0, dy=0}, etc.

Parametric kernel function *w(x, y)*:

Continuous function for coordinates in relation to center



[Apple: performing convolution operations]



Graph convolutions - parametric kernel

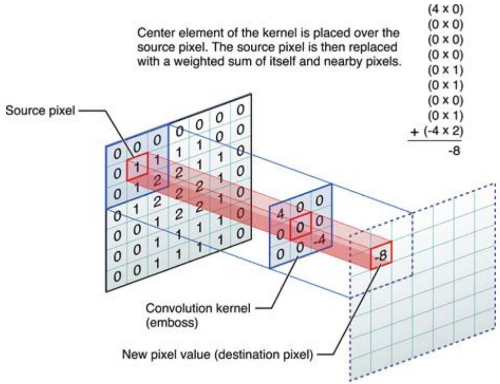
Basic CNN weight function *w(x, y):*

Look-up-table for neighbouring directions {dx=1, dy=0}, {dx=0, dy=0}, etc.

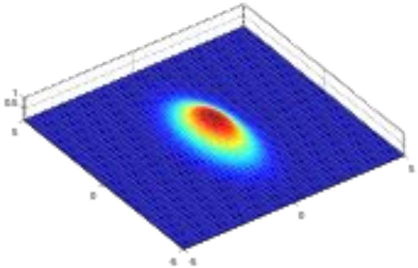
Parametric kernel function *w(x, y)*:

Continuous function for coordinates in relation to center:

$$w_j(\mathbf{u}) = \exp(-\tfrac{1}{2}(\mathbf{u}-\boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\mathbf{u}-\boldsymbol{\mu}_j)),$$

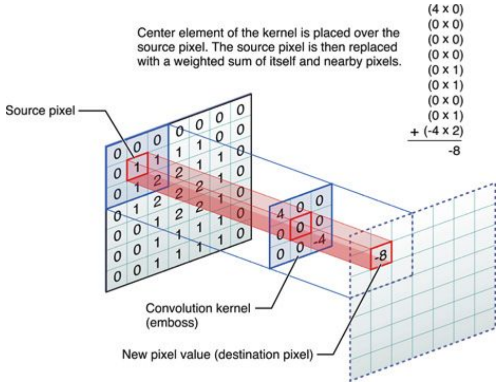


[Apple: performing convolution operations](#)



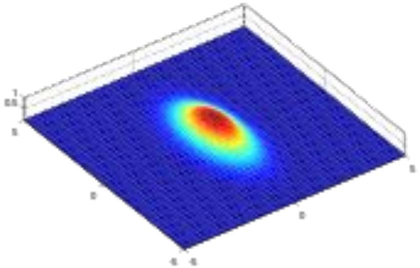
Graph convolutions - parametric kernel

Instead of learning w(x, y) directly, you learn the parameters of the function, e.g. $\mathbf{\Sigma}$ and $\boldsymbol{\mu}$. Any position is “legal”, and give some weight.



Apple: performing convolution operations

$$w_j(\mathbf{u}) = \exp(-\tfrac{1}{2}(\mathbf{u} - \boldsymbol{\mu}_j)^\top \mathbf{\Sigma}_j^{-1}(\mathbf{u} - \boldsymbol{\mu}_j)),$$



Graph convolutions - Pseudo-coordinates

“Real” coordinates may be arbitrary and not very meaningful or to high dimensional.

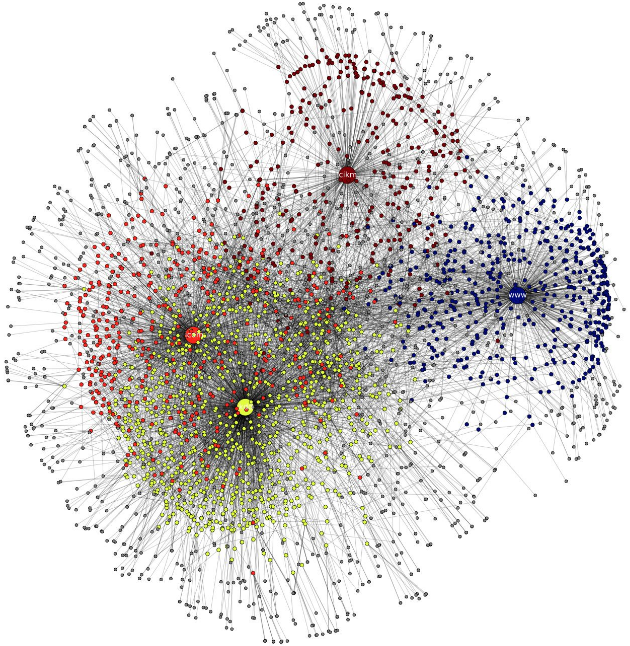


Image from:
https://gisellezeno.com/tag/graphs.html

Graph convolutions - Pseudo-coordinates

Method	Pseudo-coordinates	$\mathbf{u}(x,y)$	Weight function $w_j(\mathbf{u}), j=1,\ldots,J$
CNN [23]	Local Euclidean	$\mathbf{x}(x,y)=\mathbf{x}(y)-\mathbf{x}(x)$	$\delta(\mathbf{u}-\bar{\mathbf{u}}_j)$
GCNN [26]	Local polar geodesic	$\rho(x,y),\theta(x,y)$	$\exp(-\frac{1}{2}(\mathbf{u}-\bar{\mathbf{u}}_j)^\top \begin{pmatrix}\bar{\sigma}_\rho^2 & \\ & \bar{\sigma}_\theta^2\end{pmatrix}^{-1}(\mathbf{u}-\bar{\mathbf{u}}_j))$
ACNN [7]	Local polar geodesic	$\rho(x,y),\theta(x,y)$	$\exp(-\frac{1}{2}\mathbf{u}^\top \mathbf{R}_{\bar{\theta}_j}\begin{pmatrix}\bar{\alpha} & \\ & 1\end{pmatrix}\mathbf{R}_{\bar{\theta}_j}^\top\mathbf{u})$
GCN [21]	Vertex degree	$\deg(x),\deg(y)$	$\left(1-\lvert 1-\frac{1}{\sqrt{u_1}}\rvert\right)\left(1-\lvert 1-\frac{1}{\sqrt{u_2}}\rvert\right)$
DCNN [3]	Transition probability in r hops	$p^0(x,y),\ldots,p^{r-1}(x,y)$	$\mathrm{id}(u_j)$

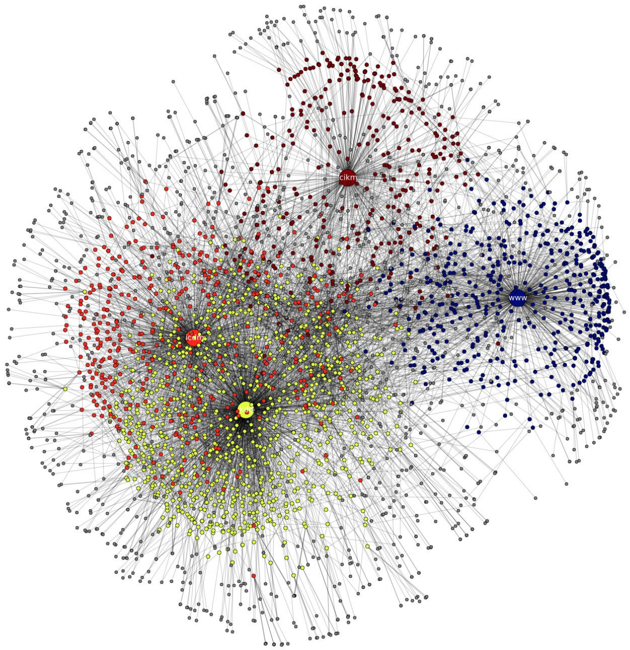
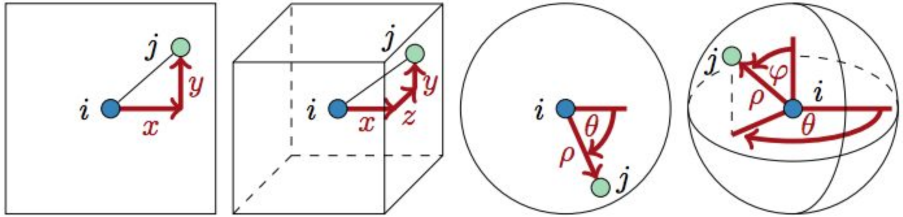


Image from:
https://gisellezeno.com/tag/graphs.html

Graph convolutions - Pseudo-coordinates

“Real” coordinates may be arbitrary and not very meaningful or to high dimensional.

Method	Pseudo-coordinates	$\mathbf{u}(x,y)$	Weight function $w_j(\mathbf{u}), j=1,\dots,J$
CNN [23]	Local Euclidean	$\mathbf{x}(x,y)=\mathbf{x}(y)-\mathbf{x}(x)$	$\delta(\mathbf{u}-\bar{\mathbf{u}}_j)$
GCNN [26]	Local polar geodesic	$\rho(x,y),\theta(x,y)$	$\exp(-\frac{1}{2}(\mathbf{u}-\bar{\mathbf{u}}_j)^\top \begin{pmatrix}\bar{\sigma}_\rho^2 & \\ & \bar{\sigma}_\theta^2\end{pmatrix}^{-1}(\mathbf{u}-\bar{\mathbf{u}}_j))$
ACNN [7]	Local polar geodesic	$\rho(x,y),\theta(x,y)$	$\exp(-\frac{1}{2}\mathbf{u}^\top \mathbf{R}_{\bar{\theta}_j}\begin{pmatrix}\bar{\alpha} & \\ & 1\end{pmatrix}\mathbf{R}_{\bar{\theta}_j}^\top \mathbf{u})$
GCN [21]	Vertex degree	$\deg(x),\deg(y)$	$\left(1-\lvert 1-\frac{1}{\sqrt{u_1}}\rvert\right)\left(1-\lvert 1-\frac{1}{\sqrt{u_2}}\rvert\right)$
DCNN [3]	Transition probability in r hops	$p^0(x,y),\dots,p^{r-1}(x,y)$	$\mathrm{id}(u_j)$



$\mathbf{u}(i,j)=(x,y)$ $\mathbf{u}(i,j)=(x,y,z)$ $\mathbf{u}(i,j)=(\rho,\theta)$ $\mathbf{u}(i,j)=(\rho,\theta,\varphi)$

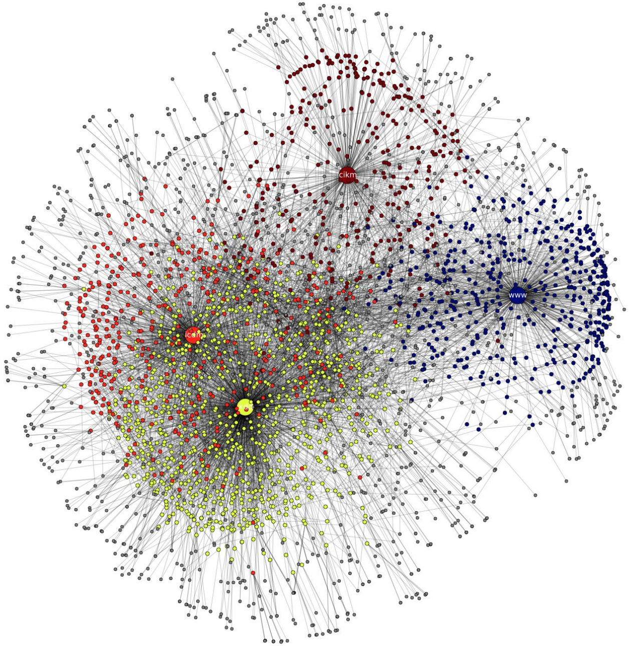


Image from:
https://gisellezeno.com/tag/graphs.html

Graph convolutions - MNIST

- In the first example pixels are on a regular grid, same for all images
- Polar representations of the coordinates are used

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

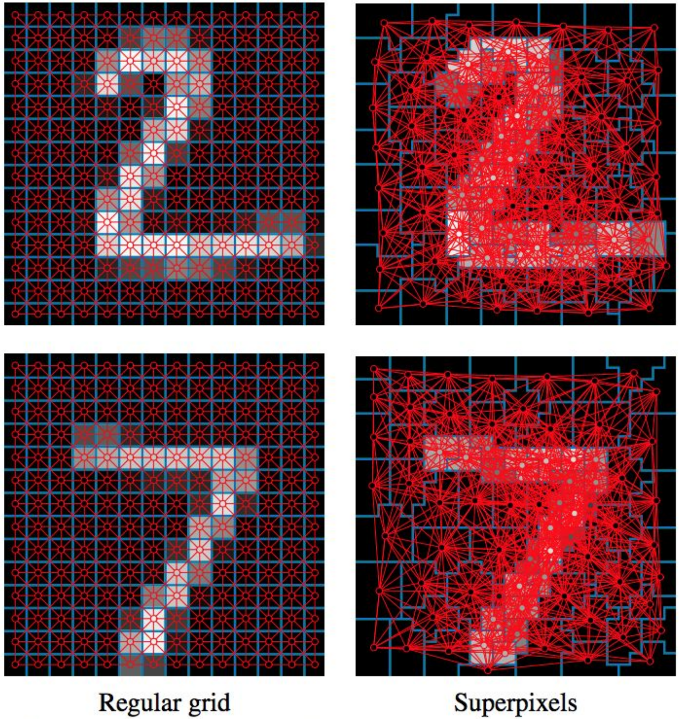


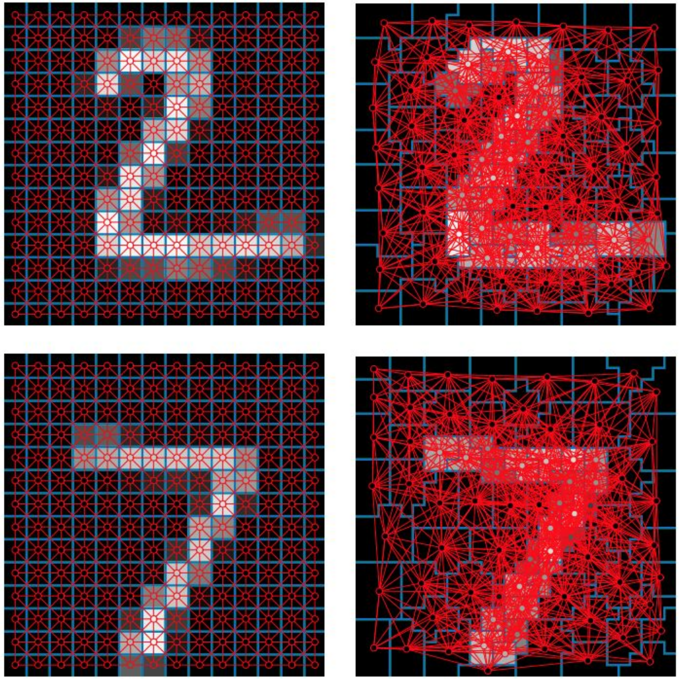
Figure 2. Representation of images as graphs. Left: regular grid (the graph is fixed for all images). Right: graph of superpixel adjacency (different for each image). Vertices are shown as red circles, edges as red lines.

Graph convolutions - MNIST

- In the first example pixels are on a regular grid, same for all images
- Polar representations of the coordinates are used

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

- Second example use an superpixel algorithm
- Different superpixels for each image
- Still polar representations are used



Regular grid Superpixels

Figure 2. Representation of images as graphs. Left: regular grid (the graph is fixed for all images). Right: graph of superpixel adjacency (different for each image). Vertices are shown as red circles, edges as red lines.

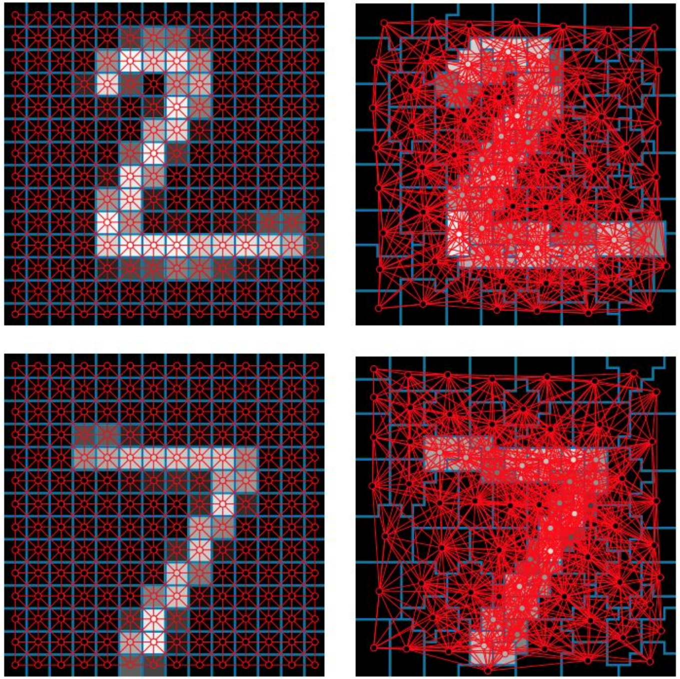
Graph convolutions - MNIST

- In the first example pixels are on a regular grid, same for all images
- Polar representations of the coordinates are used

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

- Second example use an superpixel algorithm
- Different superpixels for each image
- Still polar representations are used

Dataset	LeNet5 [23]	ChebNet [13]	**MoNet**
*Full grid	99.33%	99.14%	99.19%
*$\frac{1}{4}$ grid	98.59%	97.70%	98.16%
300 Superpixels	-	88.05%	**97.30%**
150 Superpixels	-	80.94%	**96.75%**
75 Superpixels	-	75.62%	**91.11%**

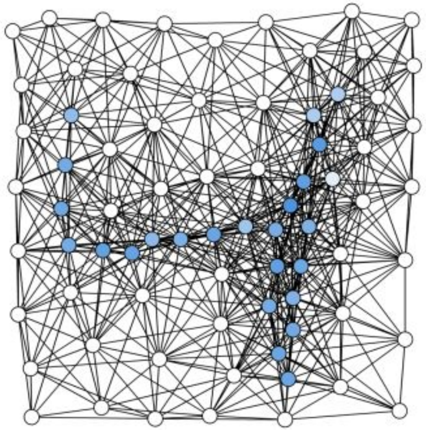


Regular grid Superpixels

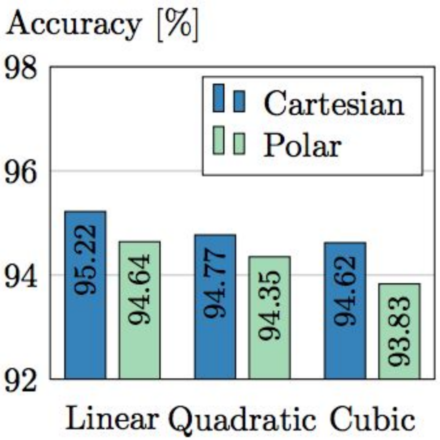
Figure 2. Representation of images as graphs. Left: regular grid (the graph is fixed for all images). Right: graph of superpixel adjacency (different for each image). Vertices are shown as red circles, edges as red lines.

Graph convolutions - MNIST

- A later study suggest that the **pseudo-coordinates** are less important, at least for 2D and 3D applications
- The difference is that they used B-Spline kernels, instead of gaussian



(a) MNIST superpixels example



(b) Classification accuracy

Dataset	LeNet5 [14]	MoNet [18]	**SplineCNN**
Grid	**99.33%**	99.19%	99.22%
Superpixels	–	91.11%	**95.22%**

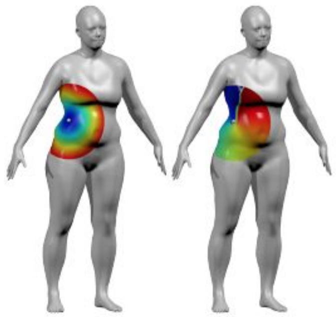
[SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels]

Graph convolutions - Surface/manifold correspondances

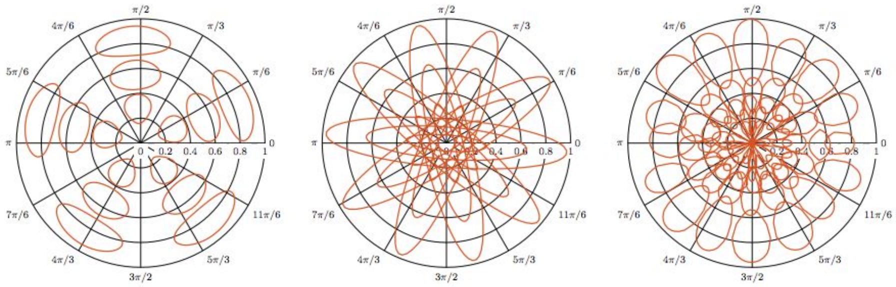
Graph convolutions - Surface/manifold

- Using spherical coordinates
- Weighting the neighbourhood with gaussian kernels
- Use histogram of local normal vectors as input (SHOT)
- Correspond to moving kernel along surface of the model

- Multiple layers work similar to regular CNN. Only swap out representation and keep position (coordinates)

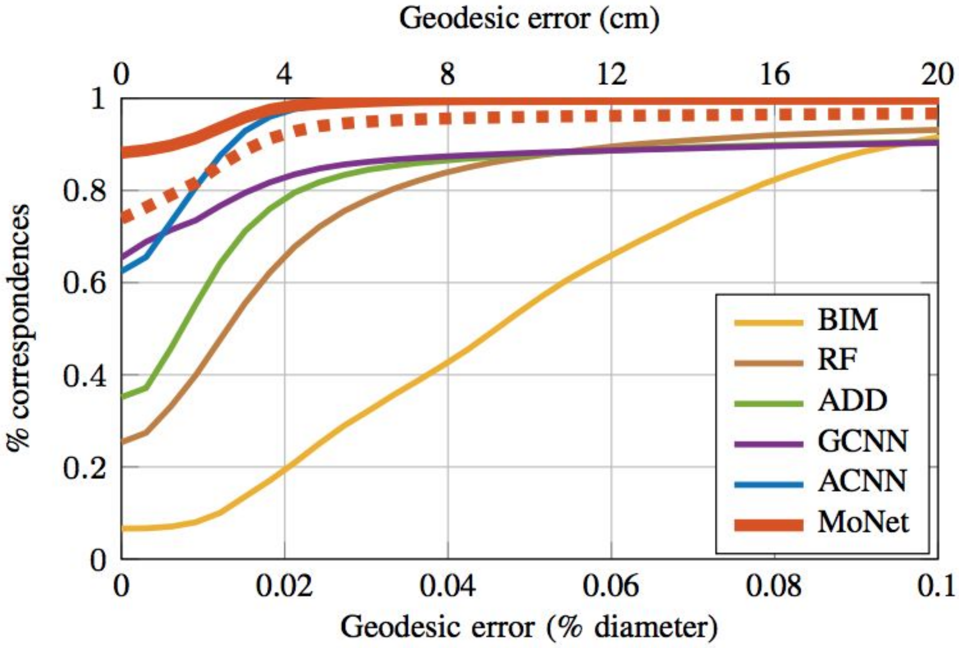


Polar coordinates ρ, θ



GCNN ACNN MoNet

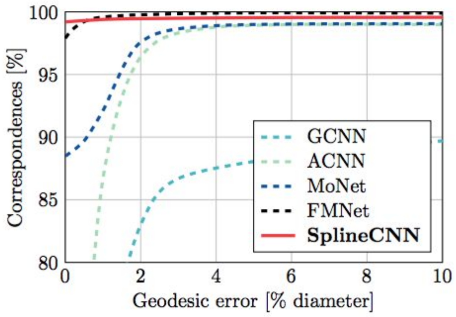
Graph convolutions - Surface/manifold



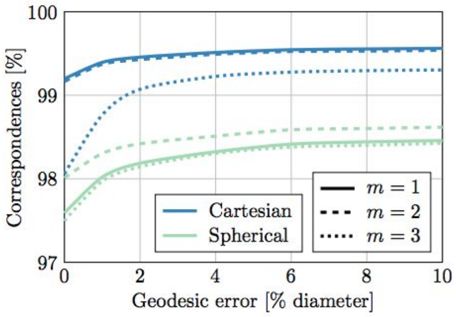
Graph convolutions - Surface/manifold

Spline kernel function and cartesian coordinates seems to work better here as well.

In this example they did not use the SHOT descriptors.



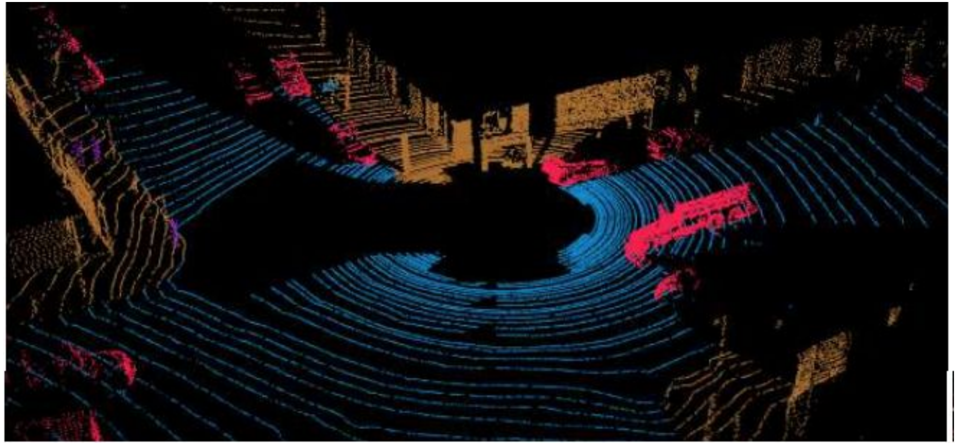
(a) Results of SplineCNN and other methods



(b) Results for different SplineCNNs

Graph convolutions on point clouds

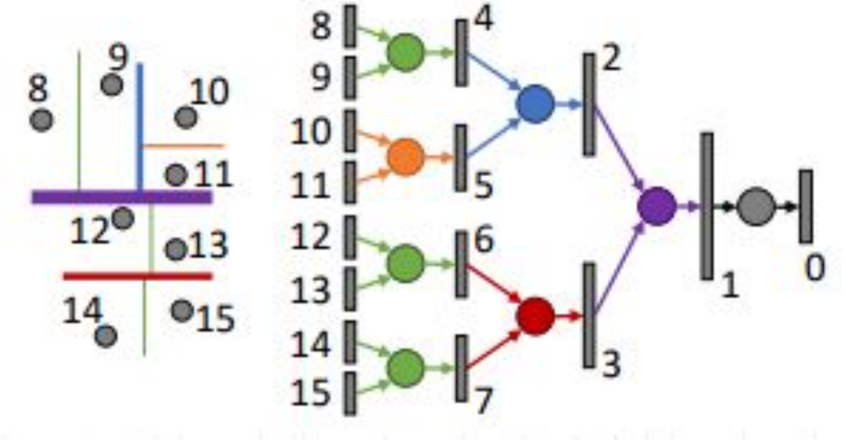
- The graph convolutional methods all have a defined neighbourhood
- How can we use graph convolutional methods without one.



[Deep Parametric Continuous Convolutional Neural Networks]

Graph convolutions on point clouds

A recent article from Uber [Deep Parametric Continuous Convolutional Neural Networks](). Used a combination of Kd-network and graph convolutions.



Graph convolutions on point clouds

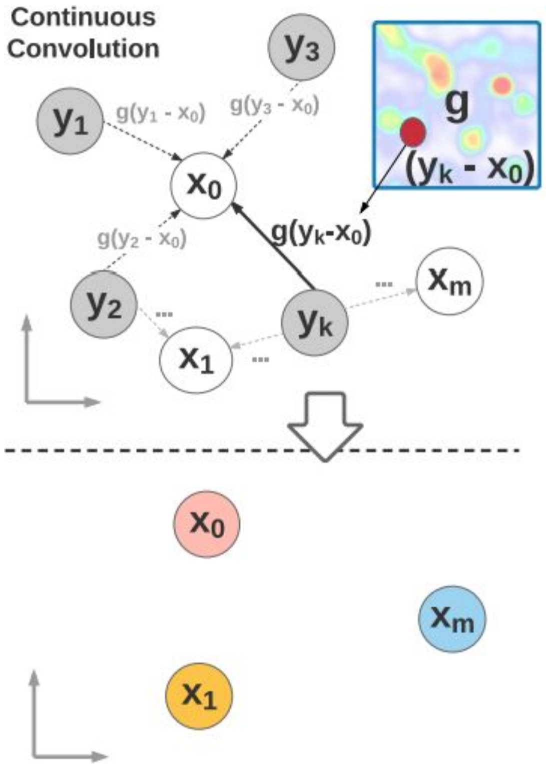
They used continuous kernels.

$$h_{k,i} = \sum_{d}^{F} \sum_{j}^{N} g_{d,k}(\mathbf{y}_i - \mathbf{x}_j) f_{d,j}$$

Over the nearest neighbours in a Kd-tree.

As kernels they used neural networks, that took distance in input point, as input, and outputs a weight value for that position.

$$g(\mathbf{z};\theta) = MLP(\mathbf{z};\theta)$$



Graph convolutions on point clouds

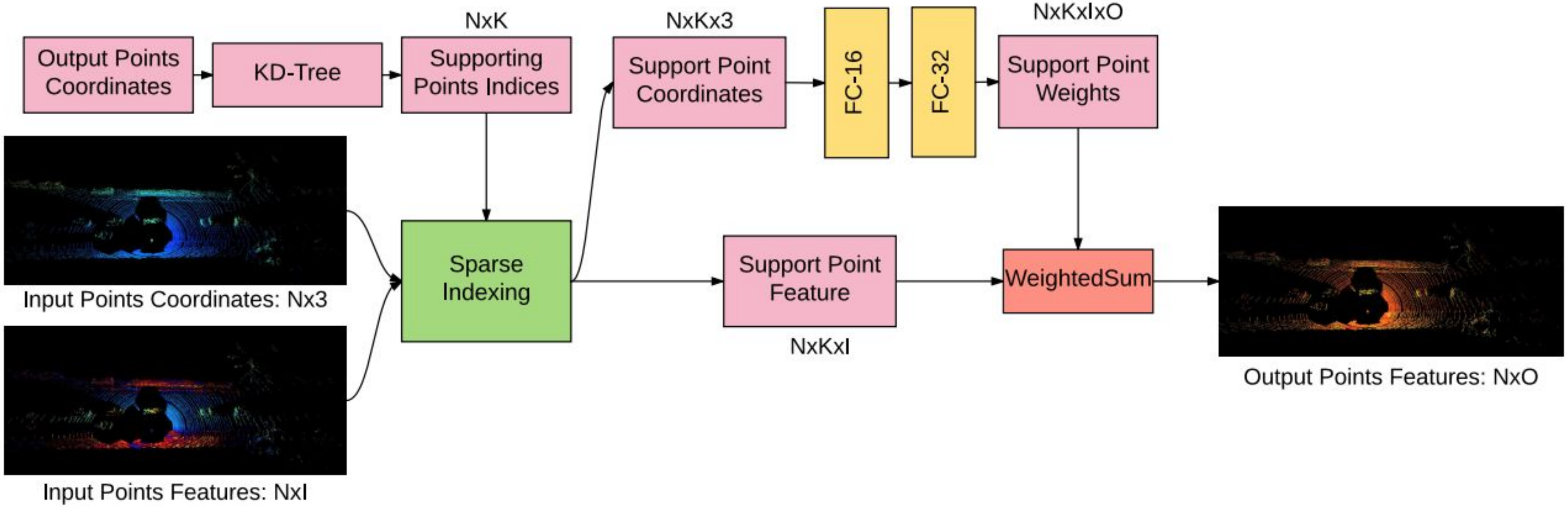
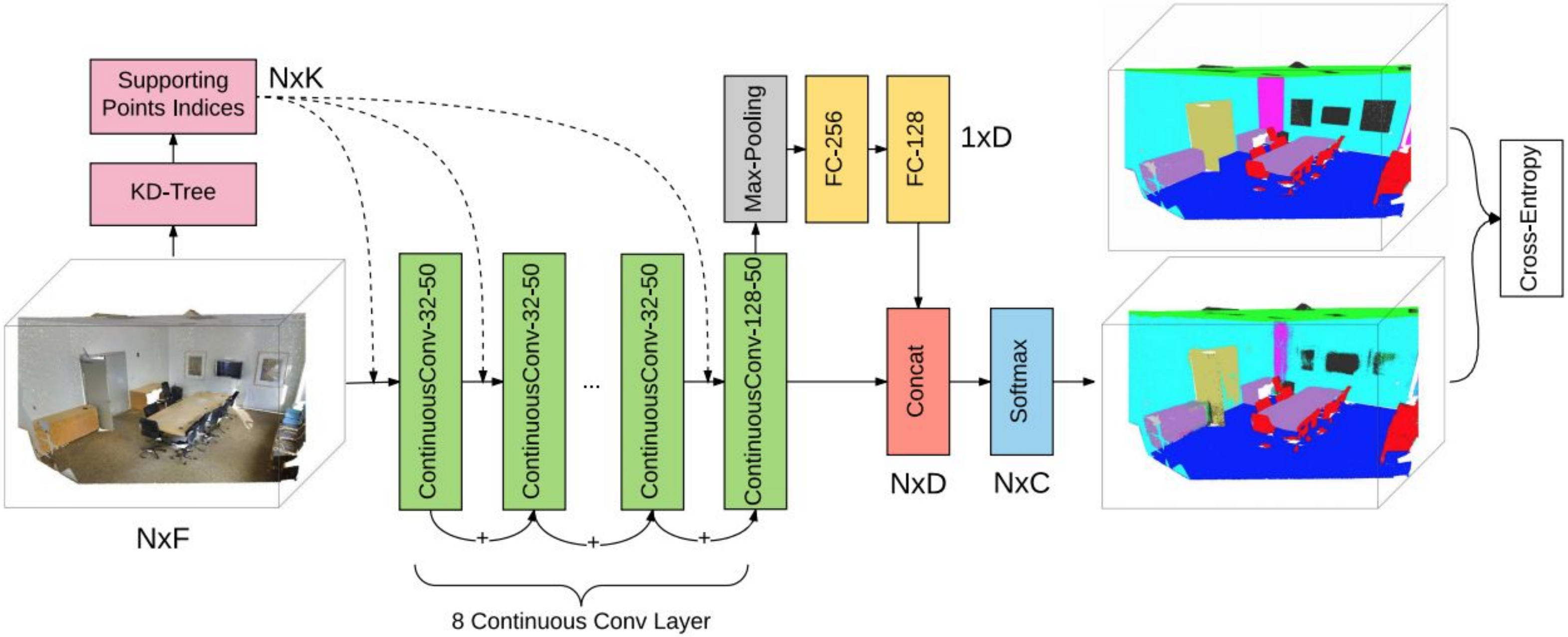
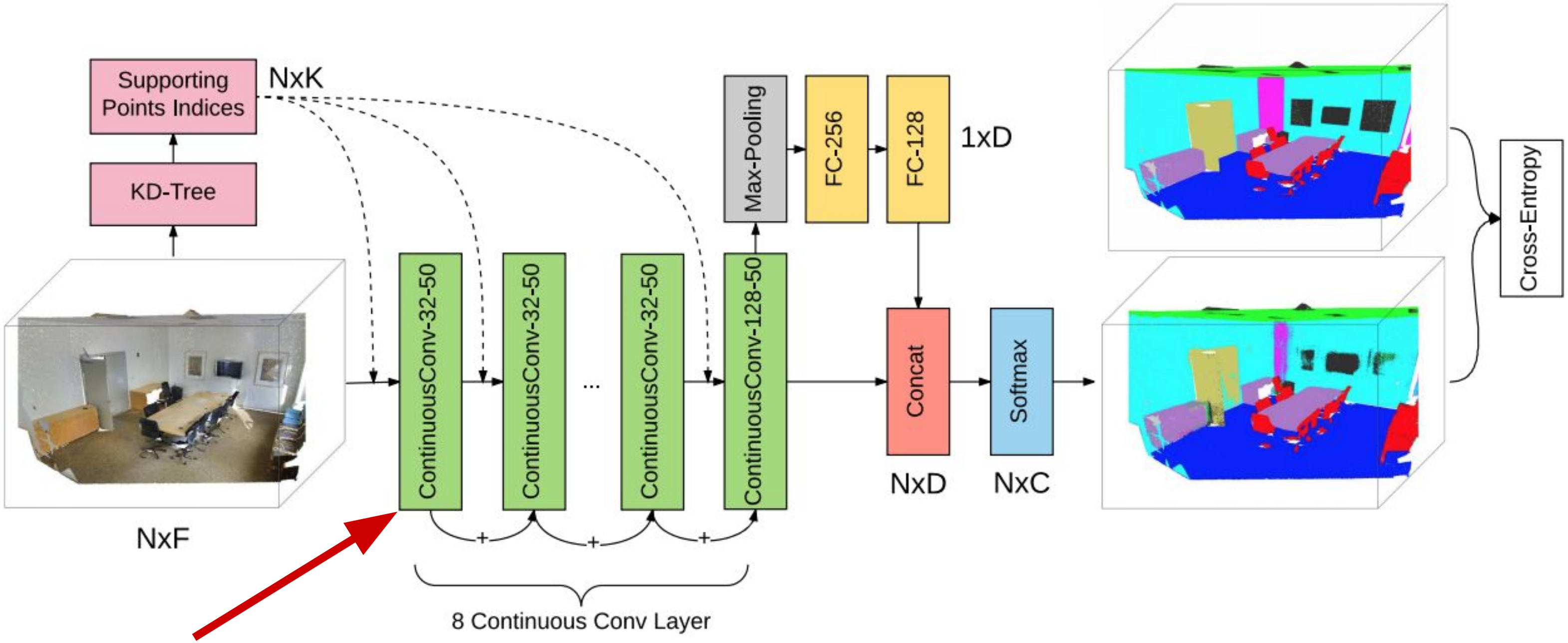


Figure 2: Detailed Computation Block for the Parametric Continuous Convolution Layer.

Graph convolutions on point clouds



Graph convolutions on point clouds



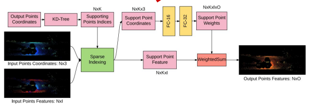
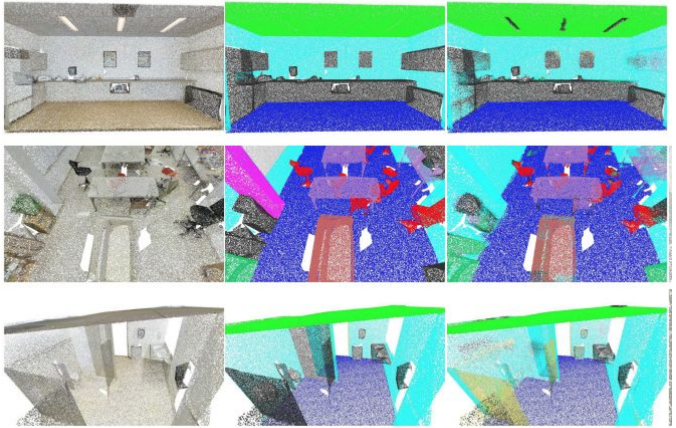


Figure 2: Detailed Computation Block for the Parametric Continuous Convolution Layer.

Graph convolutions on point clouds

State-of-art as far as I know on 3DISD

Deep nets take 33ms and KD-Tree takes 28ms on Xeon E5 and GTX 1080 Ti. OBS! Point cloud size not clear



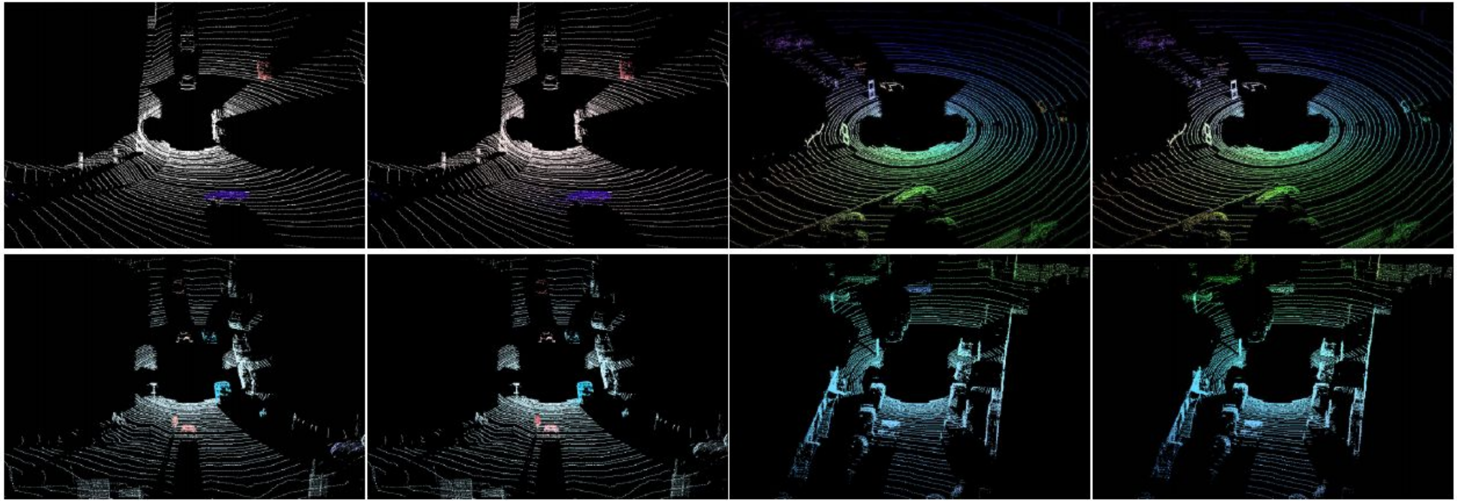
Input Ground Truth Ours PCCN

Method	mIOU	mAcc	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
PointNet [20]	41.09	48.98	88.80	**97.33**	69.80	0.05	3.92	46.26	10.76	52.61	58.93	40.28	5.85	26.38	33.22
3D-FCN-TI [28]	47.46	54.91	90.17	96.48	70.16	0.00	11.40	33.36	21.12	**76.12**	70.07	57.89	37.46	11.16	41.61
SEGCloud [28]	48.92	57.35	90.06	96.05	69.86	0.00	**18.37**	38.35	23.12	75.89	**70.40**	**58.42**	40.88	12.96	41.60
Ours PCCN	**58.27**	**67.01**	**92.26**	96.20	**75.89**	**0.27**	5.98	**69.49**	**63.45**	66.87	65.63	47.28	**68.91**	**59.10**	**46.22**

Table 1: Semantic Segmentation Results on Stanford Large-Scale 3D Indoor Scene Dataset

Graph convolutions on point clouds

Also good results on ego-motion and movement of other objects.

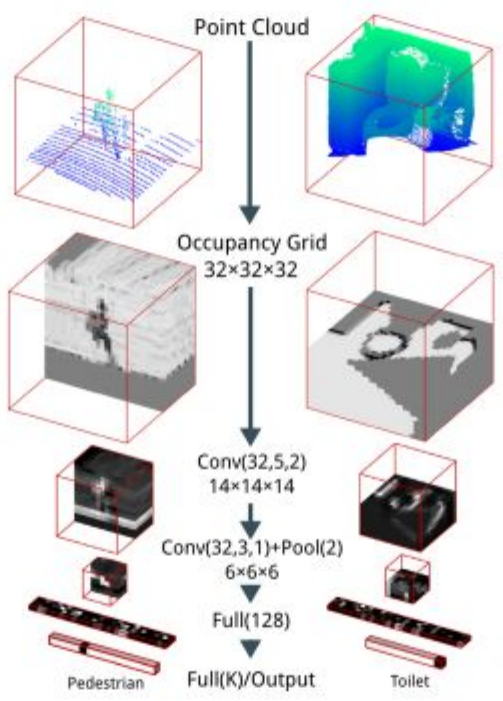


Method	EPE (cm)	Outlier$\%_{10}$	Outlier$\%_{20}$
3D-FCN	8.161	25.92%	7.12 %
Ours 3D-FCN+PCCN	**7.810**	**19.84%**	**5.97%**

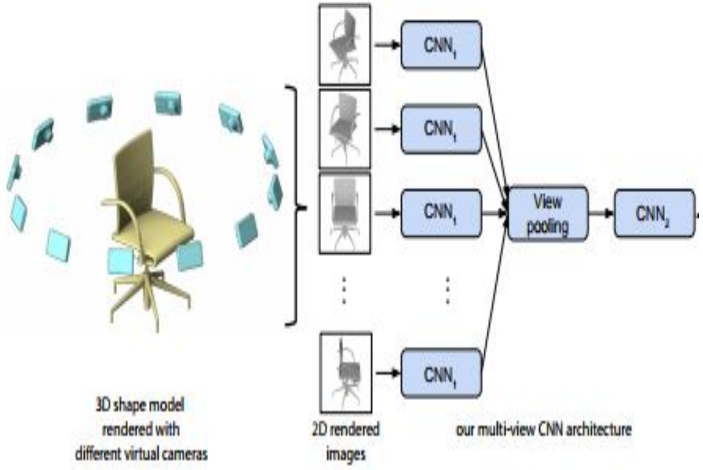
Table 3: Lidar Flow Results on Driving Scenes Dataset

Summary

Summary

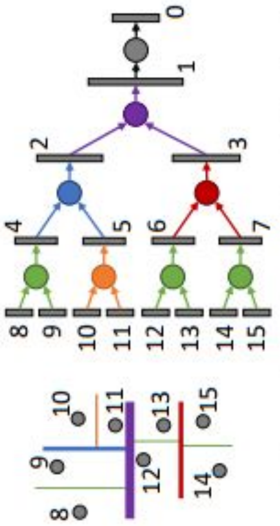


[VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition]()



[Multi-view Convolutional Neural Networks for 3D Shape Recognition]()

[PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation]()



[Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models]()

Summary

3D Segmentation:

- For dense data
- Small grids
- Resolution not important

Multi-view:

- Single objects
- Clear surfaces
- Obvious view angles

Direct point-cloud:

- Global patterns
- Noisy data

Convolution abstractions:

- Surface segmentation
- Sparse data
- Defined graph with logical edges