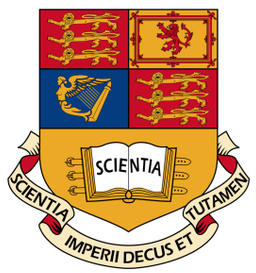
# 3D Deep Learning

Anastasia Dubrovina
Lyft (previously Stanford)

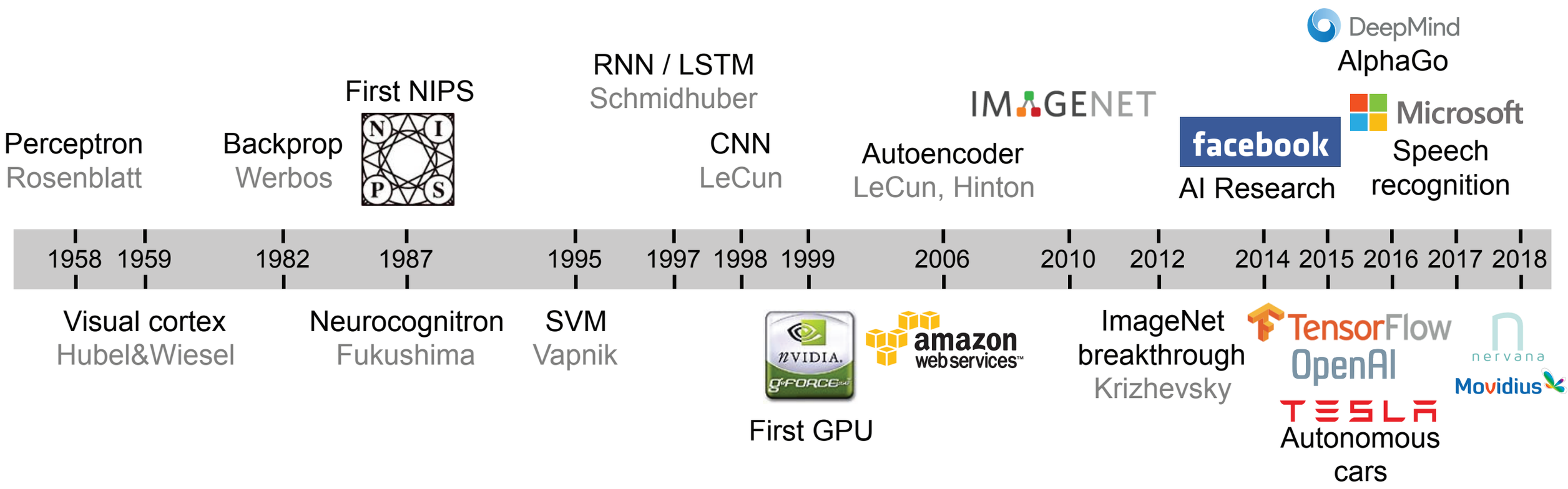
Giorgos Bouritsas
Imperial College London


Imperial College London

SGP Graduate School
6 July 2019, Milan

# Deep Learning: Introduction



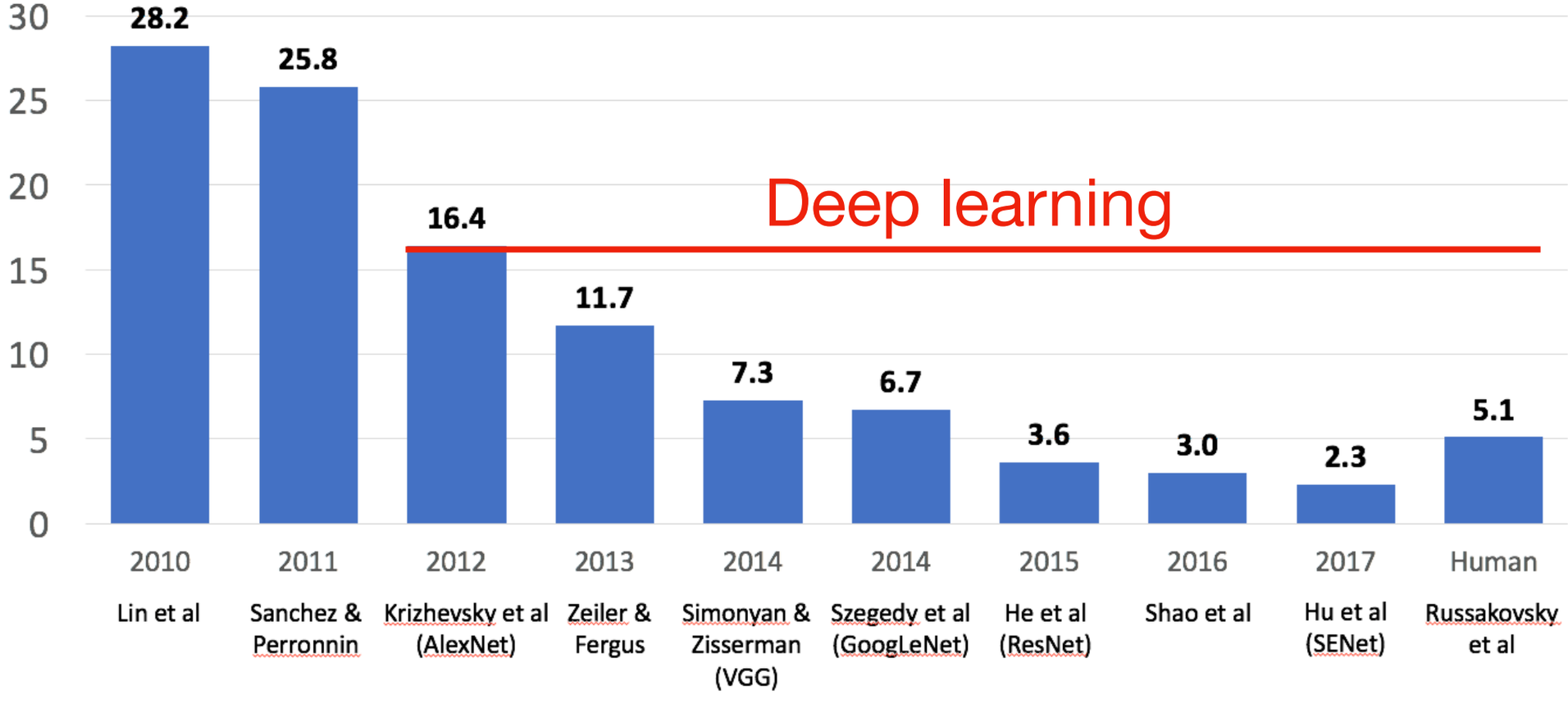

Slide: M. Bronstein

# Breakthrough in image recognition

IMAGENET **Large Scale Visual Recognition Challenge**

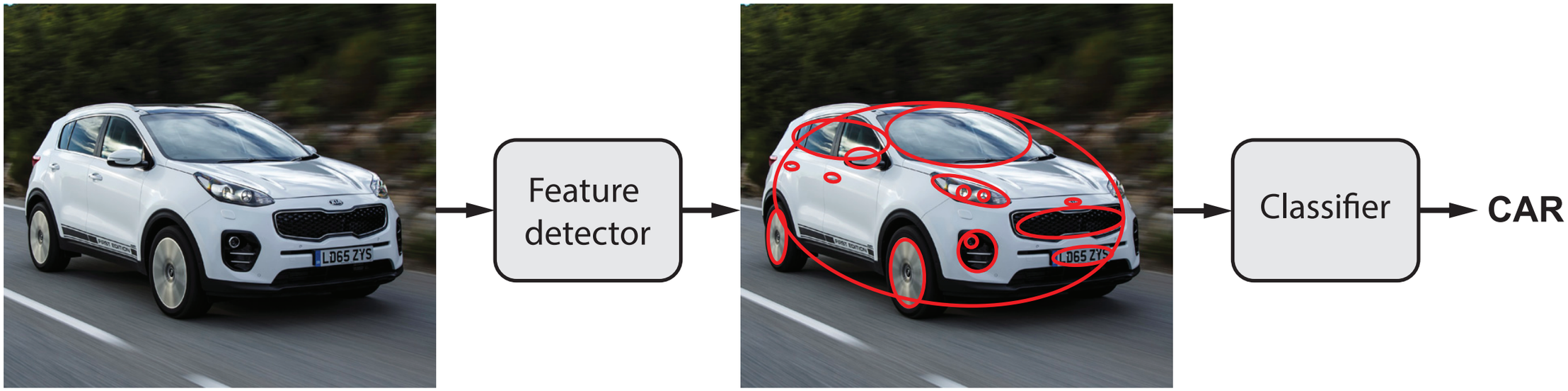
1,000 object classes

1,431,167 images




Slide: CS231n@Stanford

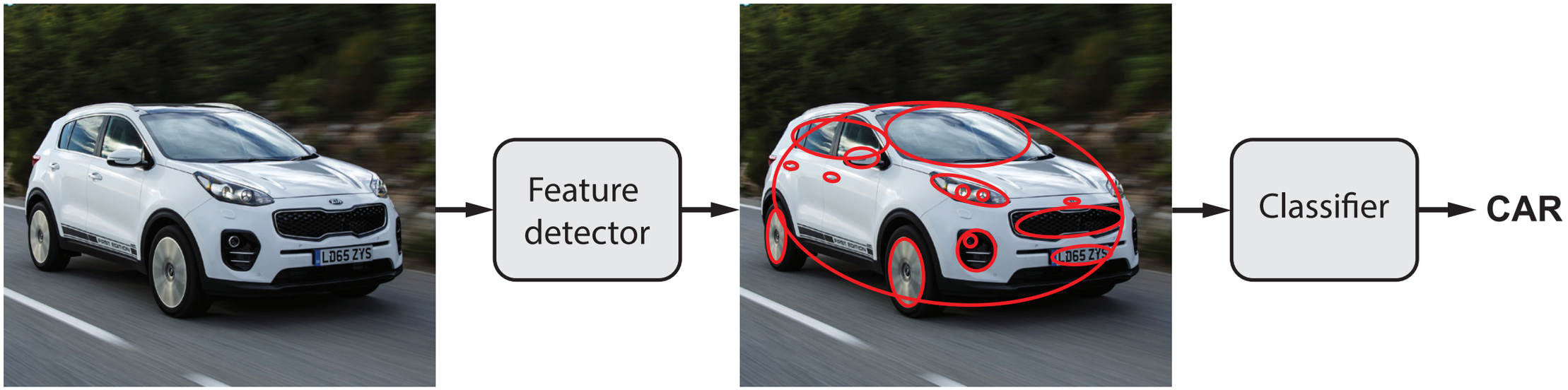
# Handcrafted vs Learned features



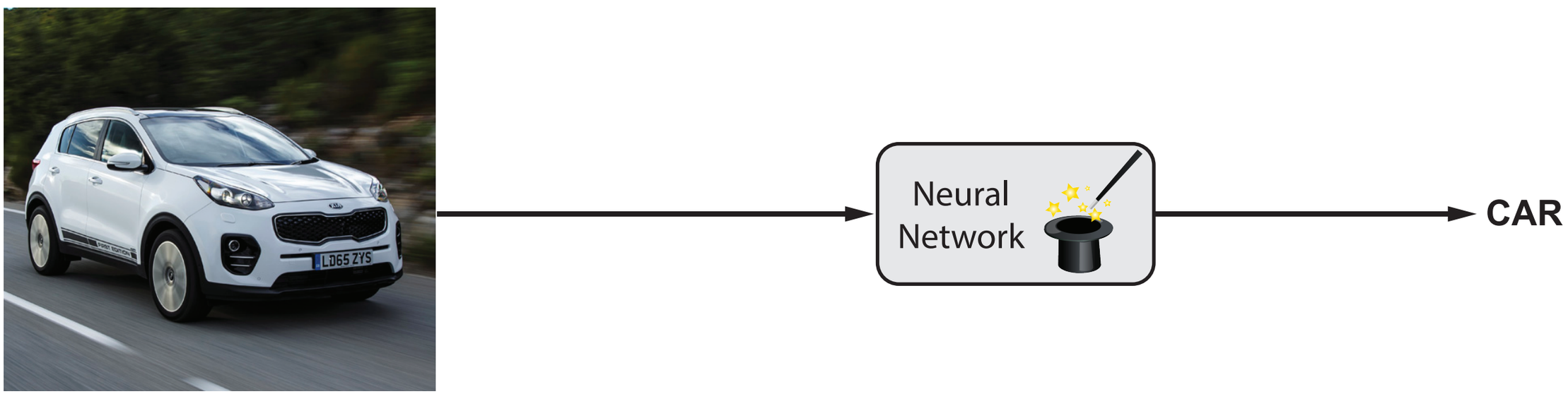

Classical computer vision: hand-crafted features (e.g. SIFT) +
simple classifier (e.g. SVM)

Slide: M. Bronstein

# Handcrafted vs Learned features




Classical computer vision: hand-crafted features (e.g. SIFT) +
simple classifier (e.g. SVM)




Modern computer vision: data-driven end-to-end systems

Slide: M. Bronstein

# Convolutional Neural Networks (CNN)

1998
LeCun et al.

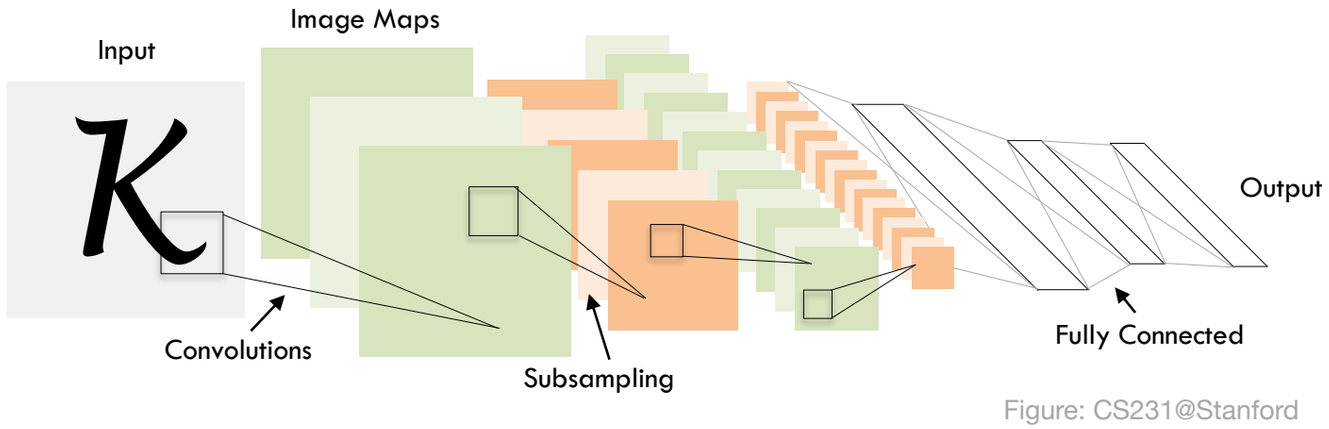
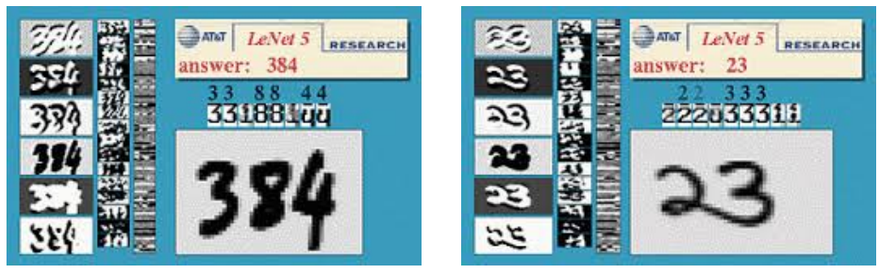
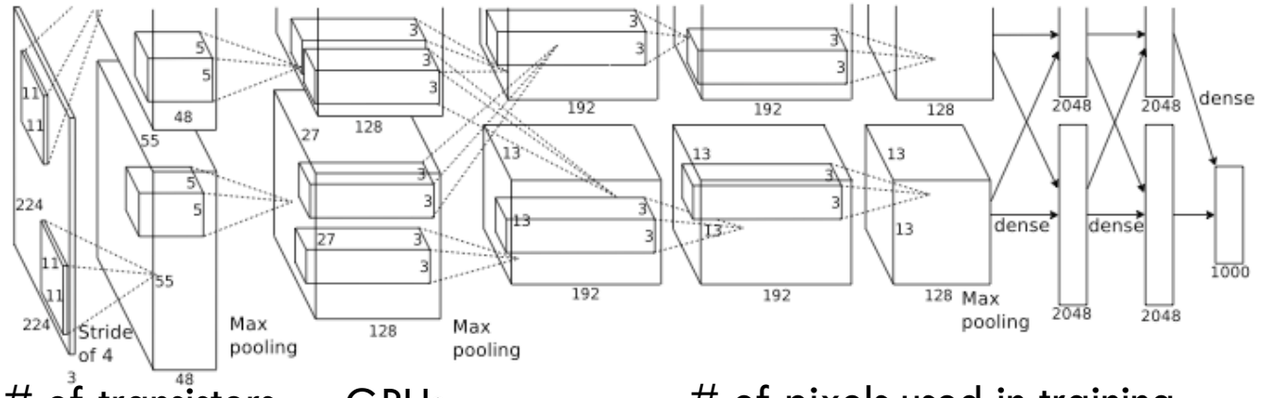



Figure: CS231@Stanford




- 3 convolutional + 1 fully connected layers
- 1M parameters
- Trained on MNISK 70K
- CPU-based
- *tanh* non-linearity

2012
Krizhevsky et al.




# of transistors GPUs # of pixels used in training

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012.

- 5 convolutional + 3 fully connected layers
- 60M parameters
- Trained on ImageNet 1.5M
- GPU-based
- ReLU, Dropout

Credit: CS231n@Stanford, M. Bronstein

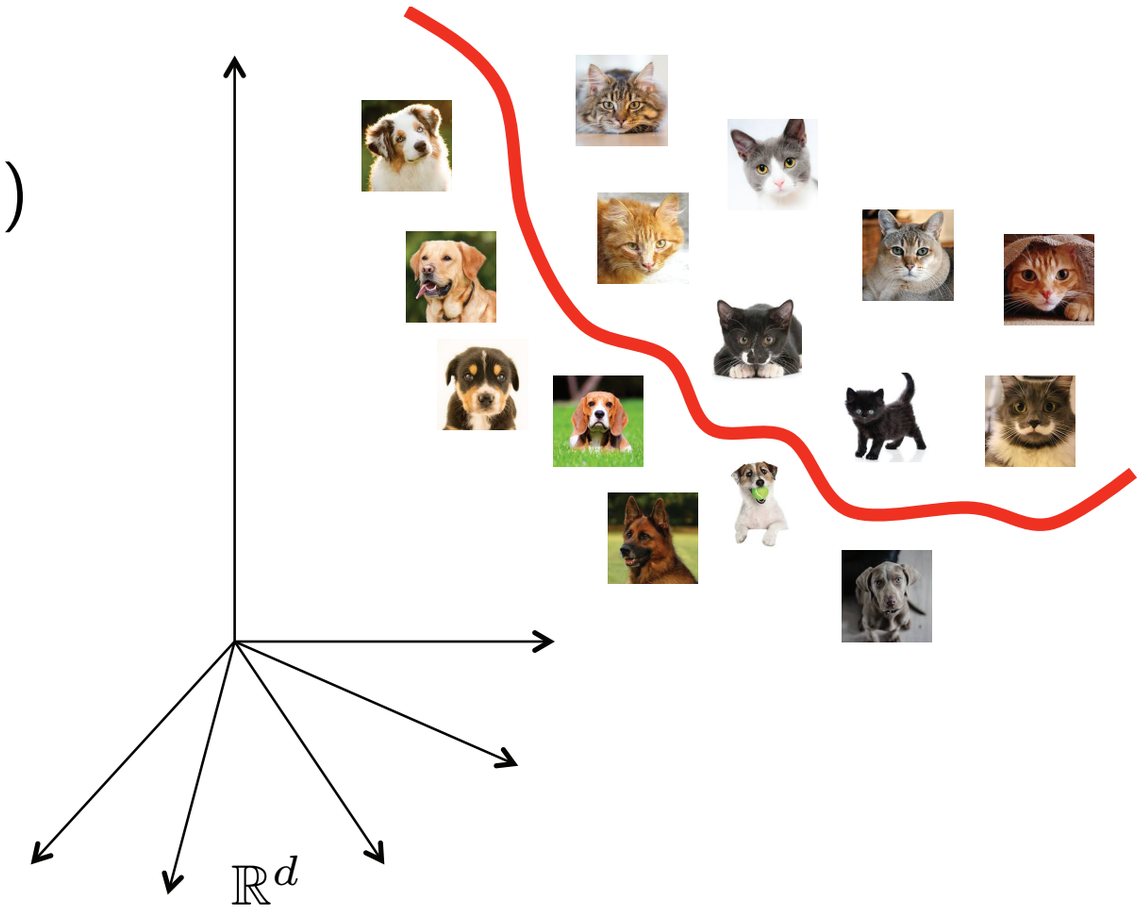
# Basics of deep learning

# Supevised learning: classification example

- Data vectors $\mathbf{x} \in \mathbb{R}^d$
  (e.g. for 512×512 images $d \approx 10^5$)
- Unknown classification functional
  $f : \mathbb{R}^d \to \{1, \ldots, L\}$ in $L$ classes
- Training set

$$S = \{(\mathbf{x}_i \in \mathbb{R}^d,\, y_i = f(\mathbf{x}_i))\}_{i=1}^{T}$$

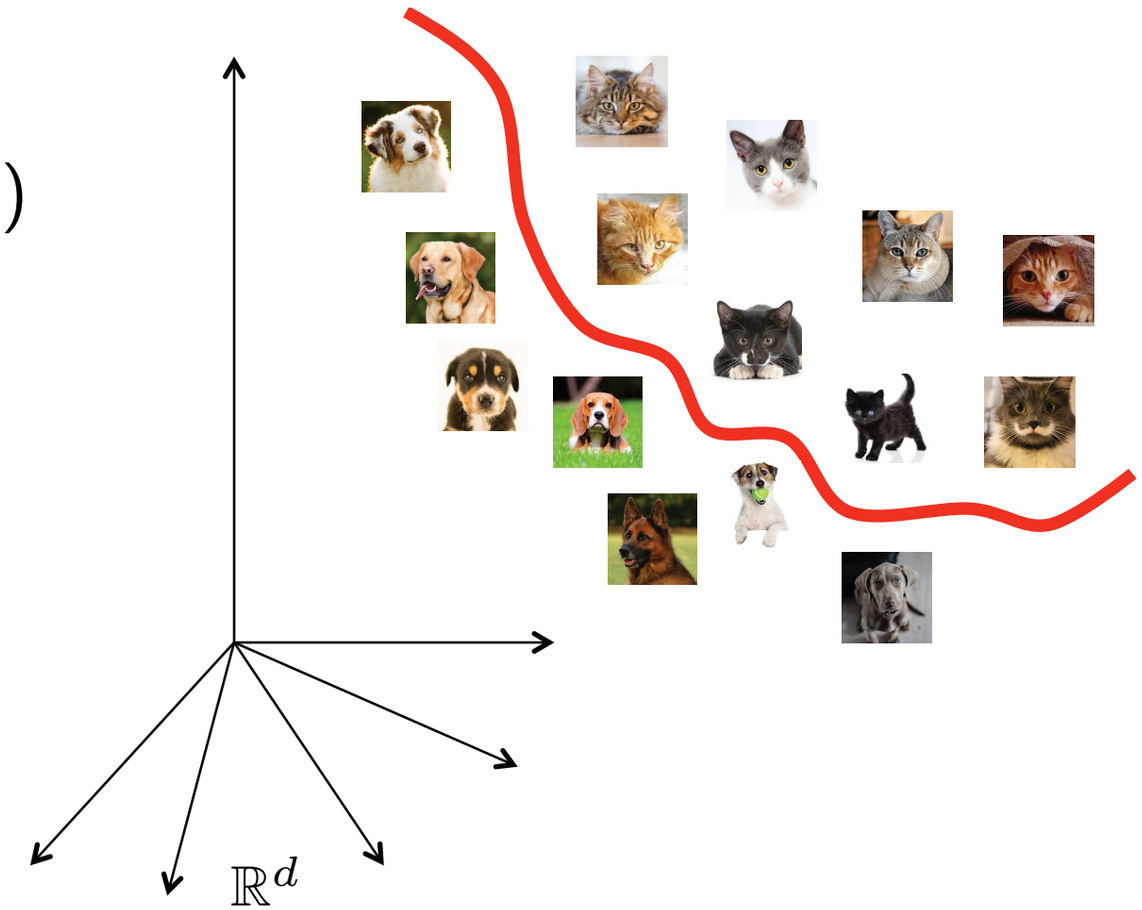
- Parametric model $f_{\mathbf{\Theta}}$ of $f$



# Supevised learning: classification example

- Data vectors $\mathbf{x} \in \mathbb{R}^d$
  (e.g. for 512×512 images $d \approx 10^5$)
- Unknown classification functional
  $f : \mathbb{R}^d \to \{1, \ldots, L\}$ in $L$ classes
- Training set

$$S = \{(\mathbf{x}_i \in \mathbb{R}^d,\, y_i = f(\mathbf{x}_i))\}_{i=1}^{T}$$



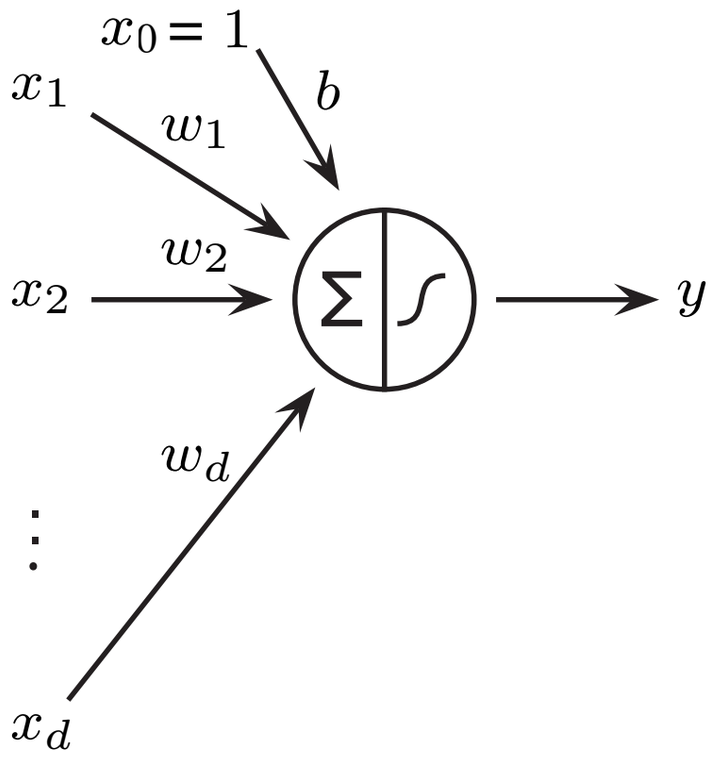

- Parametric model $f_{\boldsymbol{\Theta}}$ of $f$

Supervised learning: find optimal model parameters by minimizing the loss $\ell$ on the training set

$$\boldsymbol{\Theta}^* = \operatorname*{argmin}_{\boldsymbol{\Theta}} \sum_{i=1}^{T} \ell(f_{\boldsymbol{\Theta}}(\mathbf{x}_i), y_i)$$

Slide: M. Bronstein

# Simplest neural network: perceptron




Linear layer $$y = \xi\left(\sum_{\ell=1}^{d} w_\ell x_\ell + b\right) = \xi\left(\mathbf{w}^\top \mathbf{x}\right)$$

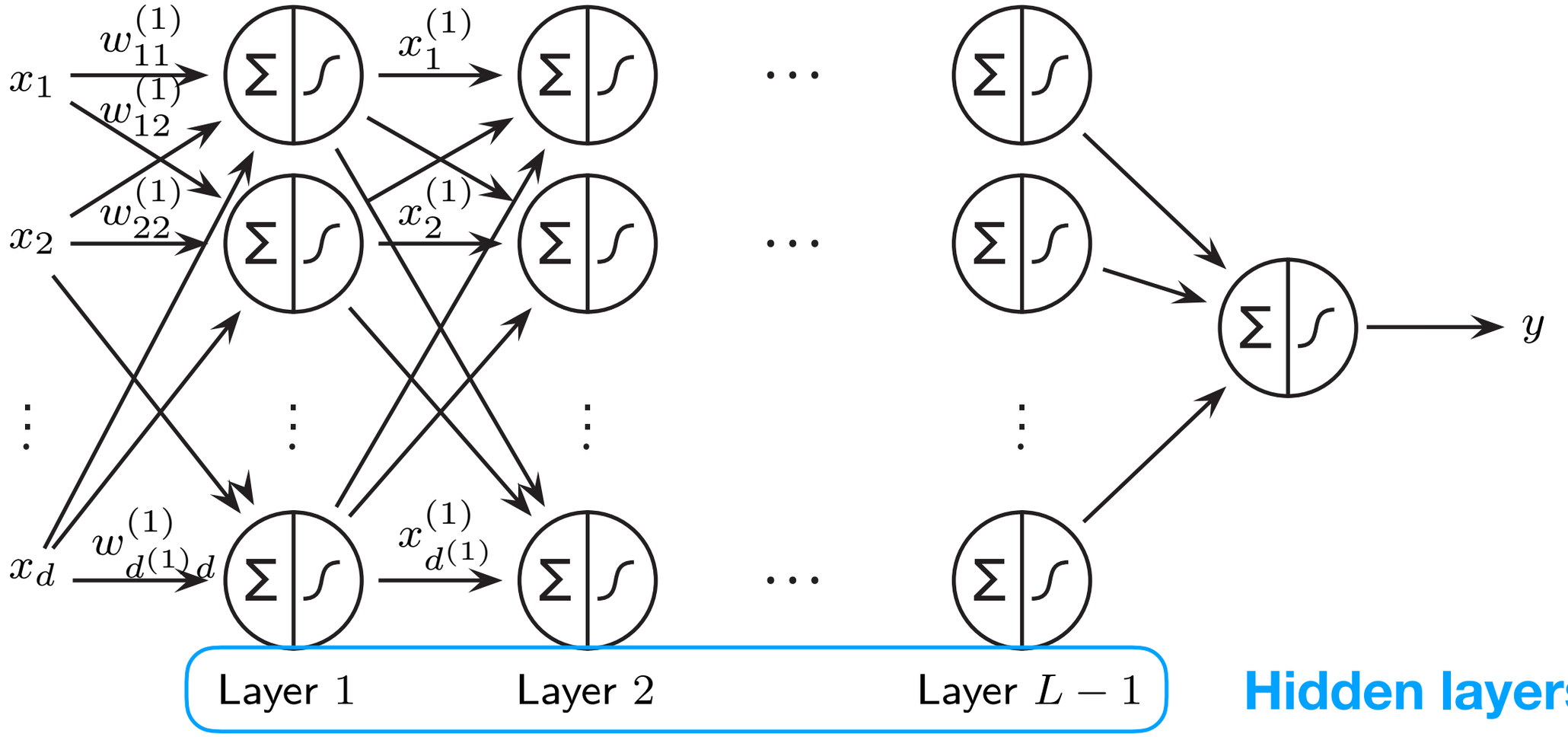
Activation, e.g. $\xi(x) = \tanh(x)$

Parameters layer weights $\mathbf{w} = (b, w_1, \ldots, w_d)$, including bias

Rosenblatt 1957

Slide: M. Bronstein

# Multi-layer fully connected neural network




**Hidden layers**

Linear layer $\quad \mathbf{x}^{(l+1)} = \xi\left(\mathbf{W}^{(l+1)}\mathbf{x}^{(l)}\right)$

Activation, e.g. $\quad \xi(x) = \tanh(x)$

Parameters $\quad$ layer weights $\mathbf{W}^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}}$

Slide: M. Bronstein

# Setting the number of layers and their sizes

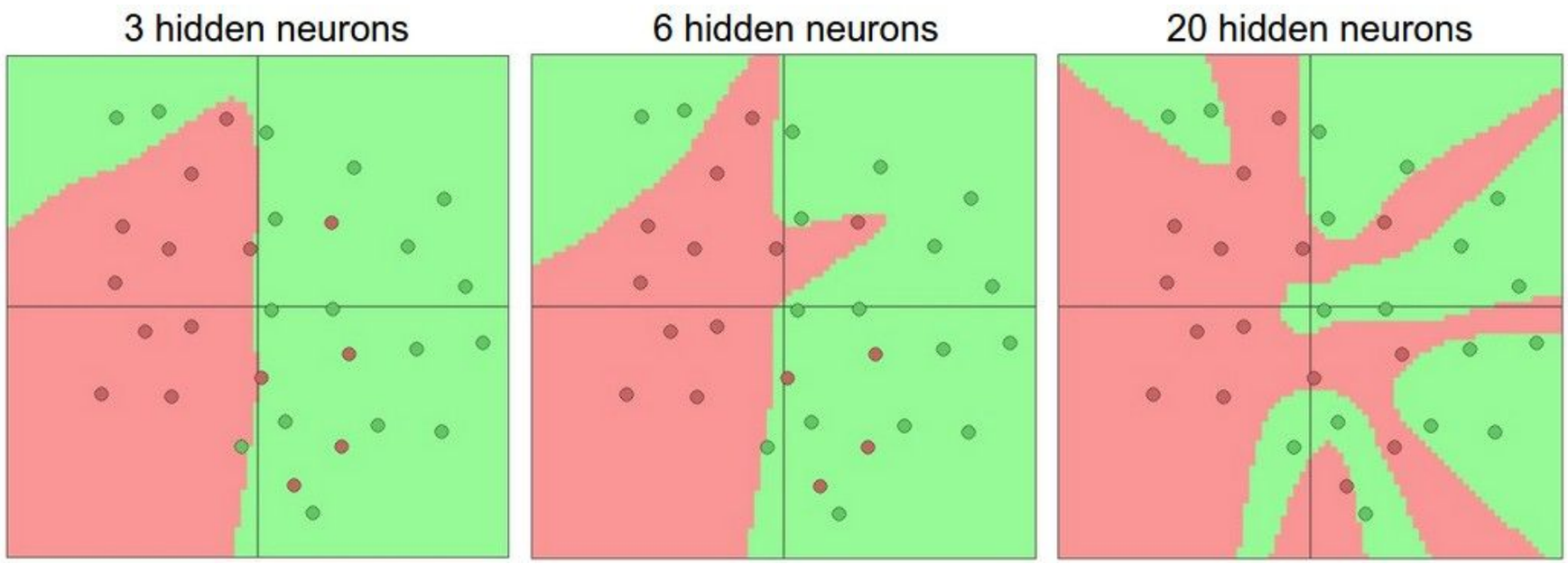



Image: CS231n@Stanford

# Setting the number of layers and their sizes

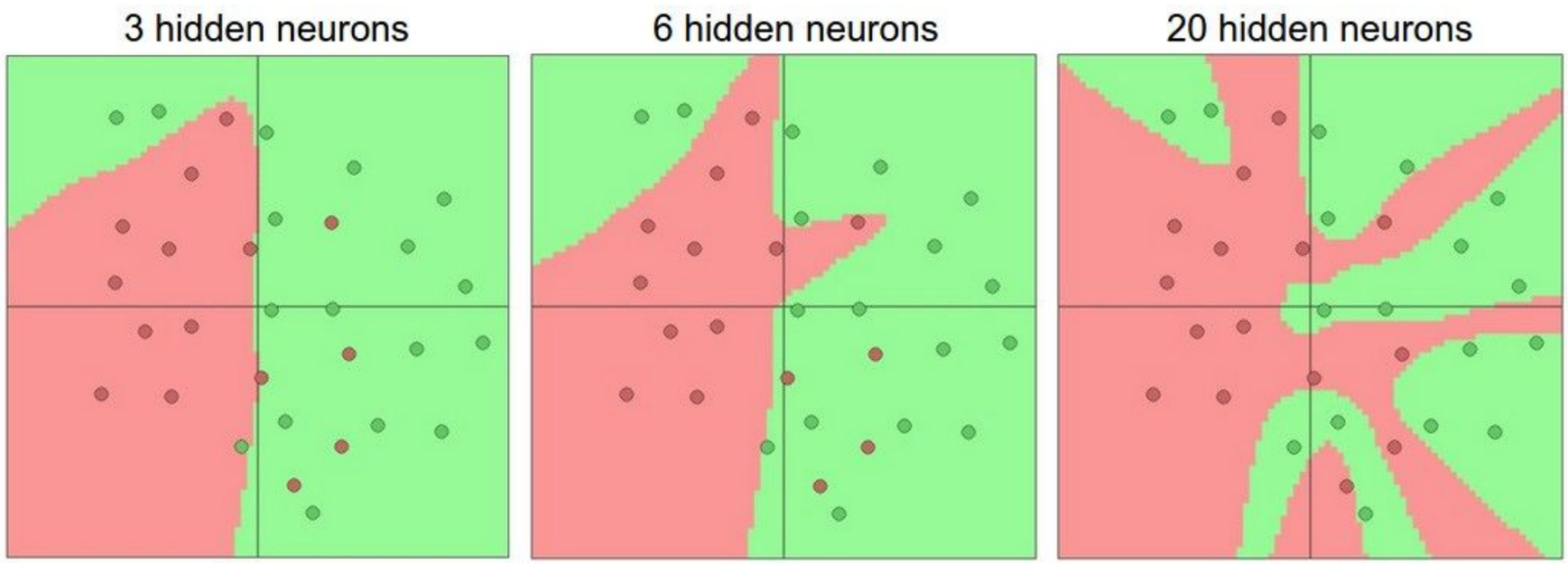



Image: CS231n@Stanford

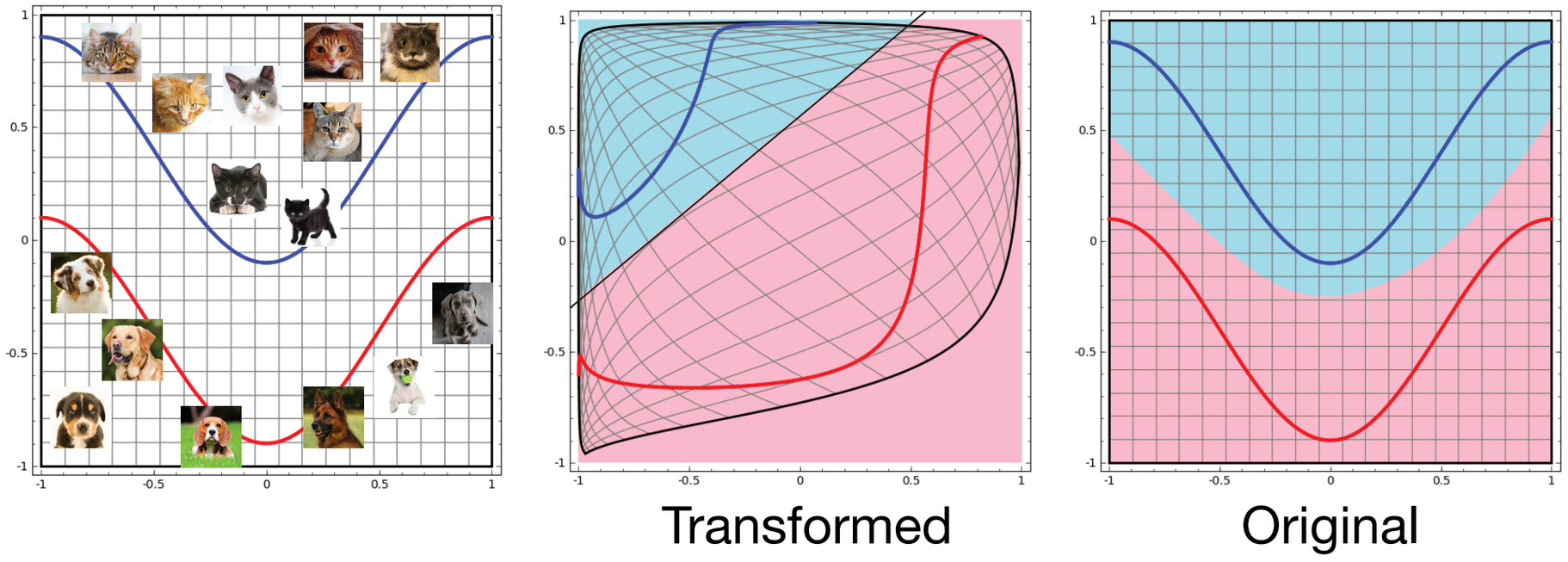
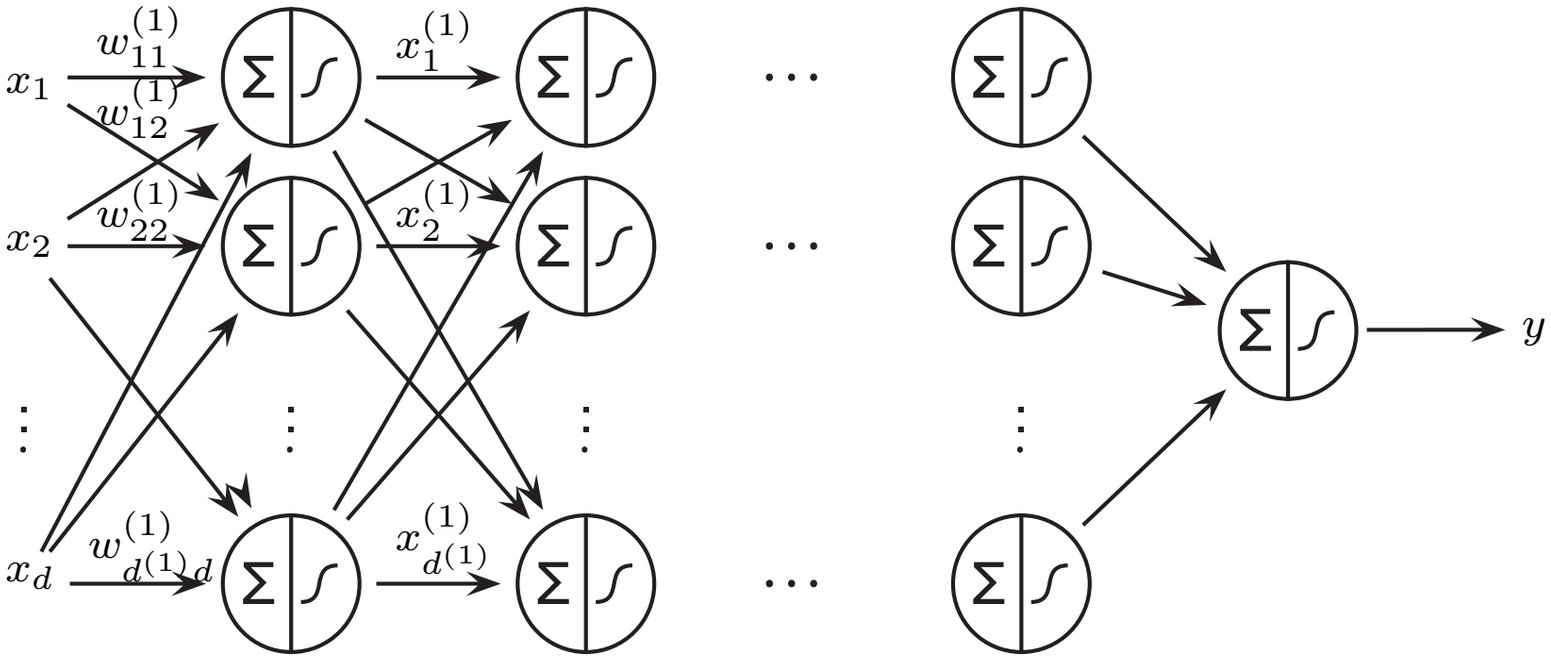
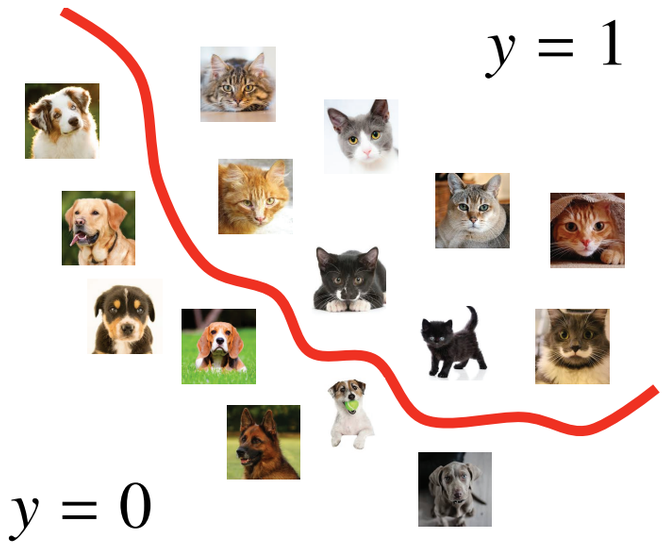



Image: Christopher Olah

# Neural network training: Backpropagation

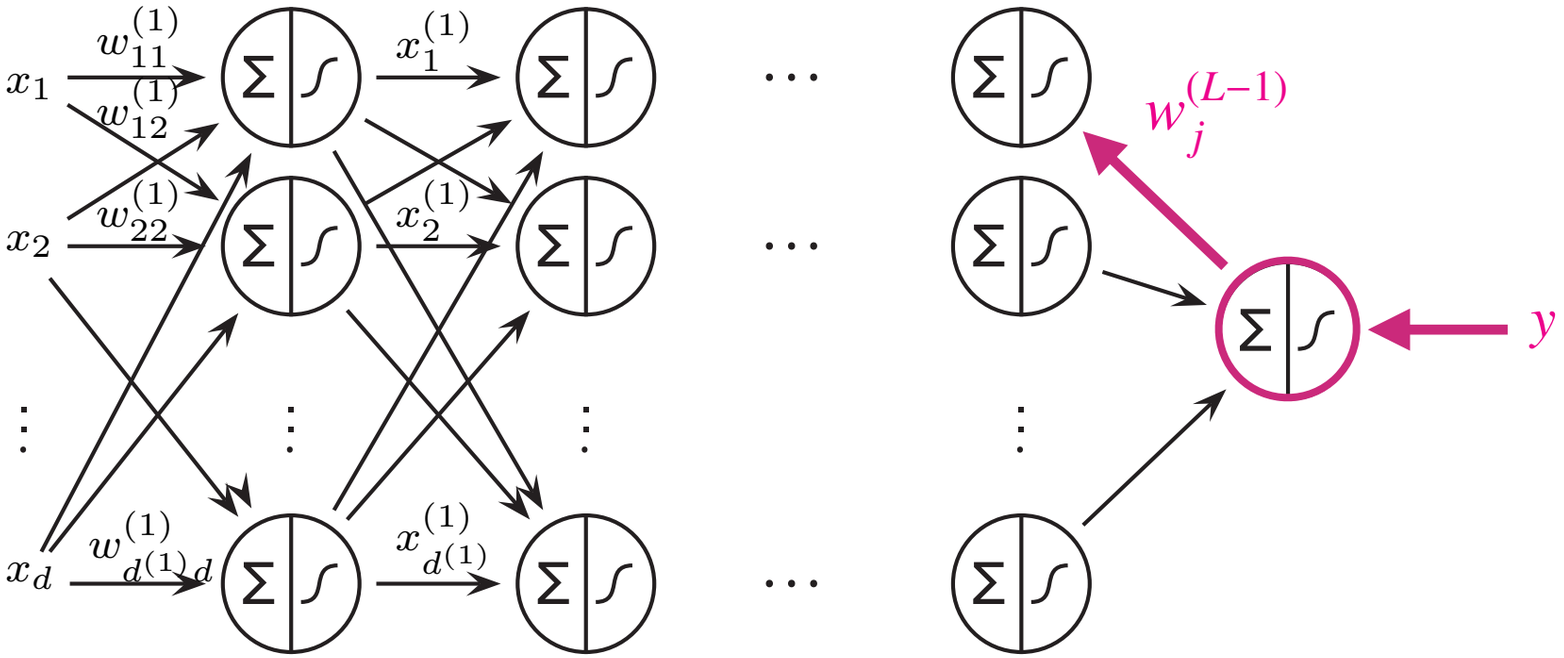



$$\mathbf{\Theta}^* = \underset{\mathbf{\Theta}}{\operatorname{argmin}} E, \quad \text{where} \;\; E = \sum_{i=1}^{T} \ell(f_{\mathbf{\Theta}}(\mathbf{x}_i), y_i)$$

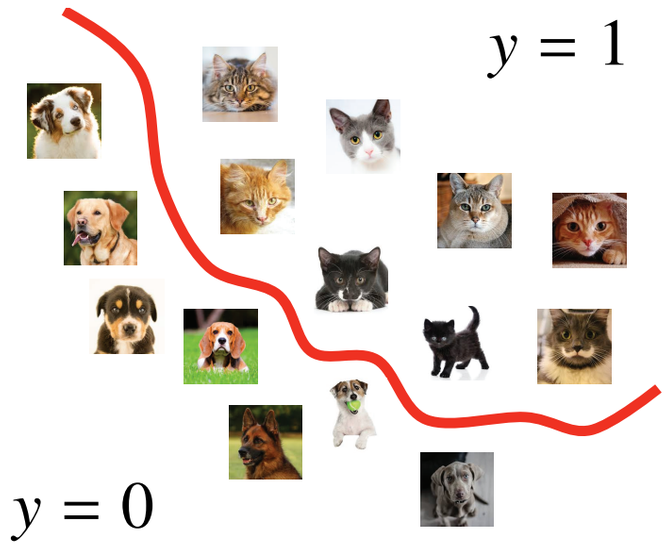
Image credit: M. Bronstein

# Neural network training: Backpropagation









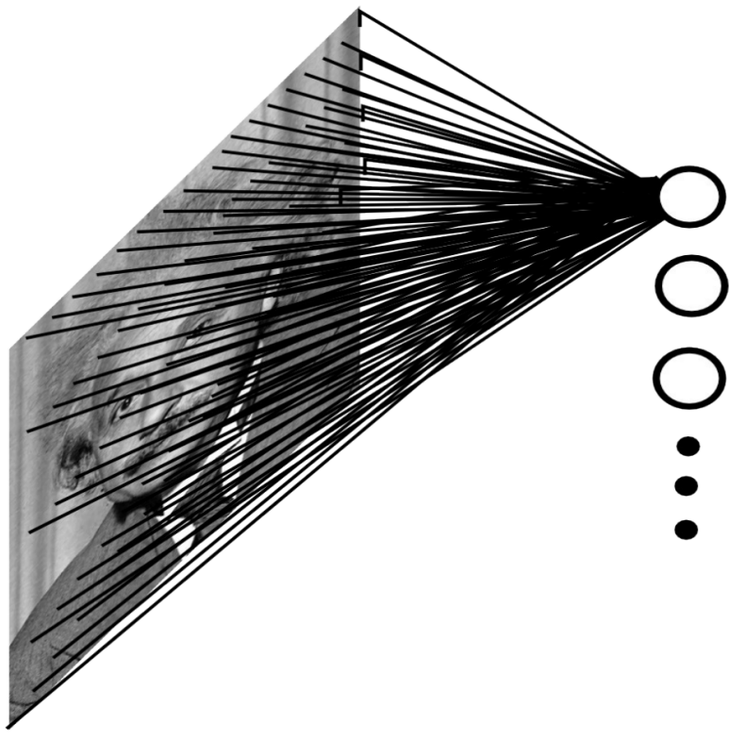

$$\mathbf{\Theta}^* = \underset{\mathbf{\Theta}}{\operatorname{argmin}} E, \quad \text{where} \quad E = \sum_{i=1}^{T} \ell(f_{\mathbf{\Theta}}(\mathbf{x}_i), y_i)$$

Chain rule: $$\frac{\delta E}{\delta w_j^{(L-1)}} = \frac{\delta E}{\delta y} \frac{\delta y}{\delta w_j^{(L-1)}}$$

Image credit: M. Bronstein

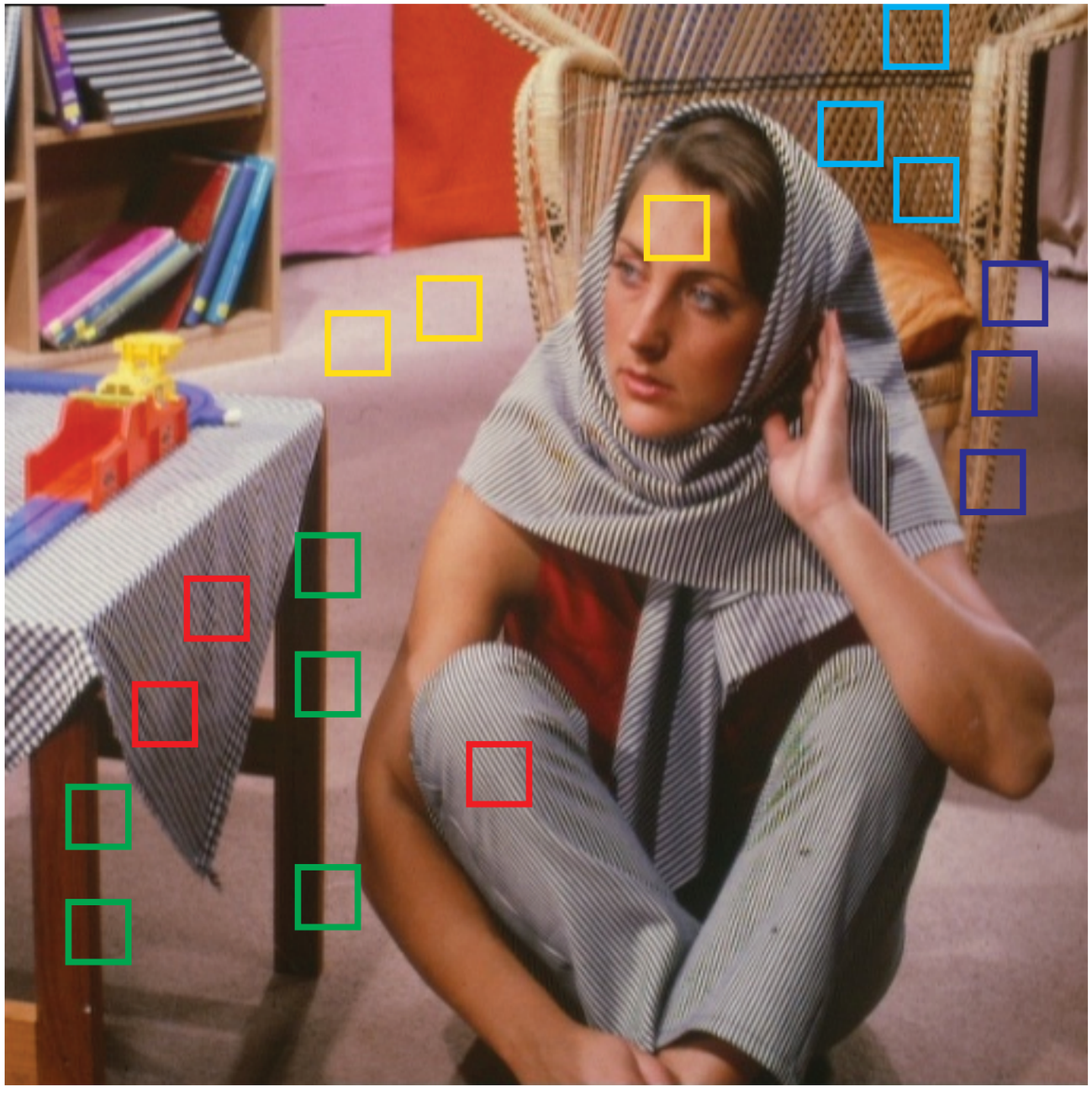
# Fully connected neural networks

- **Example: 200x200 image**
  - Fully-connected, 400,000 hidden units = 16 billion parameters



Credit: Yann LeCun

# Stationarity and self-similarity

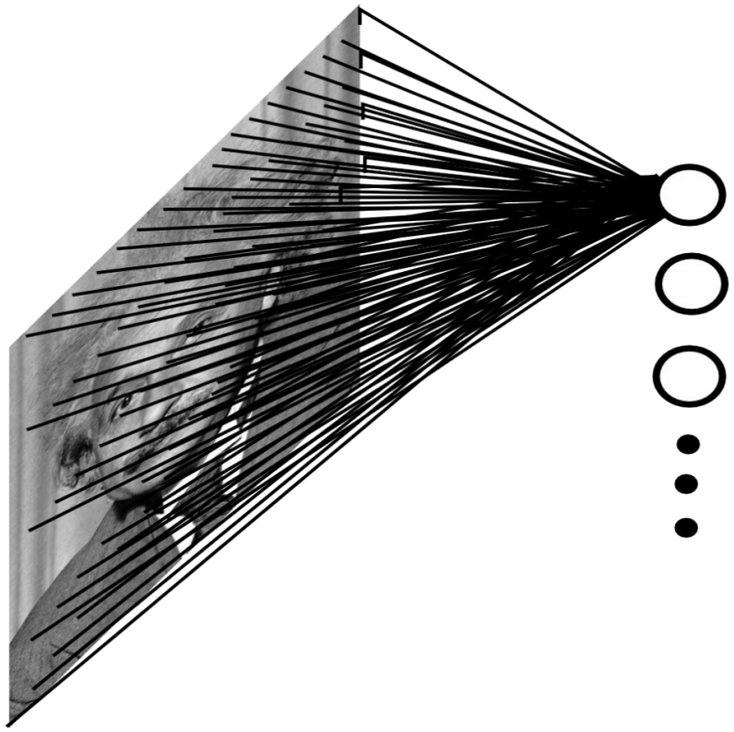


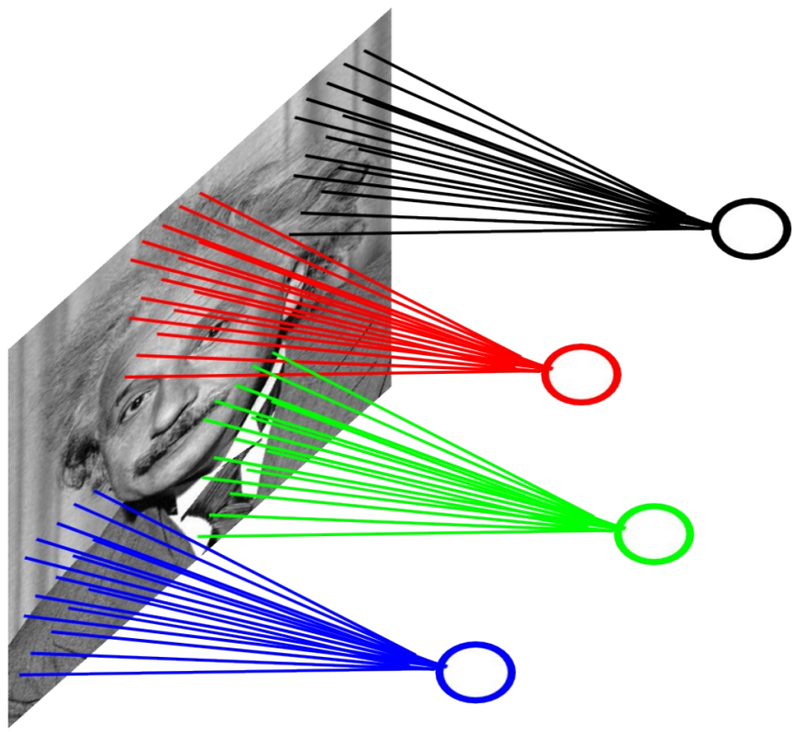
Data is self-similar across the domain

Slide: M. Bronstein

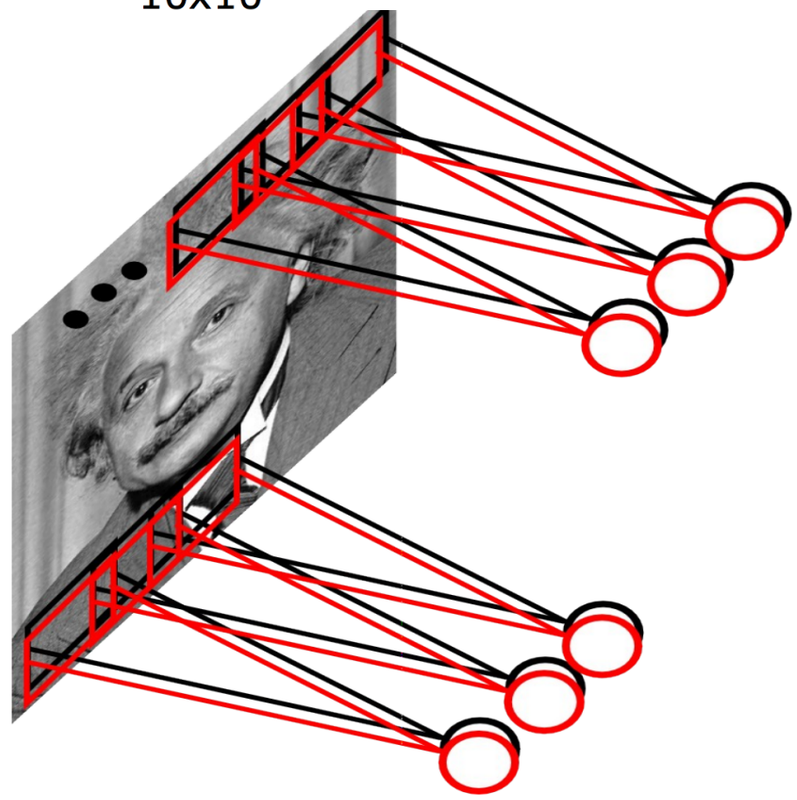
# Convolutional neural networks

- **Example: 200x200 image**
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



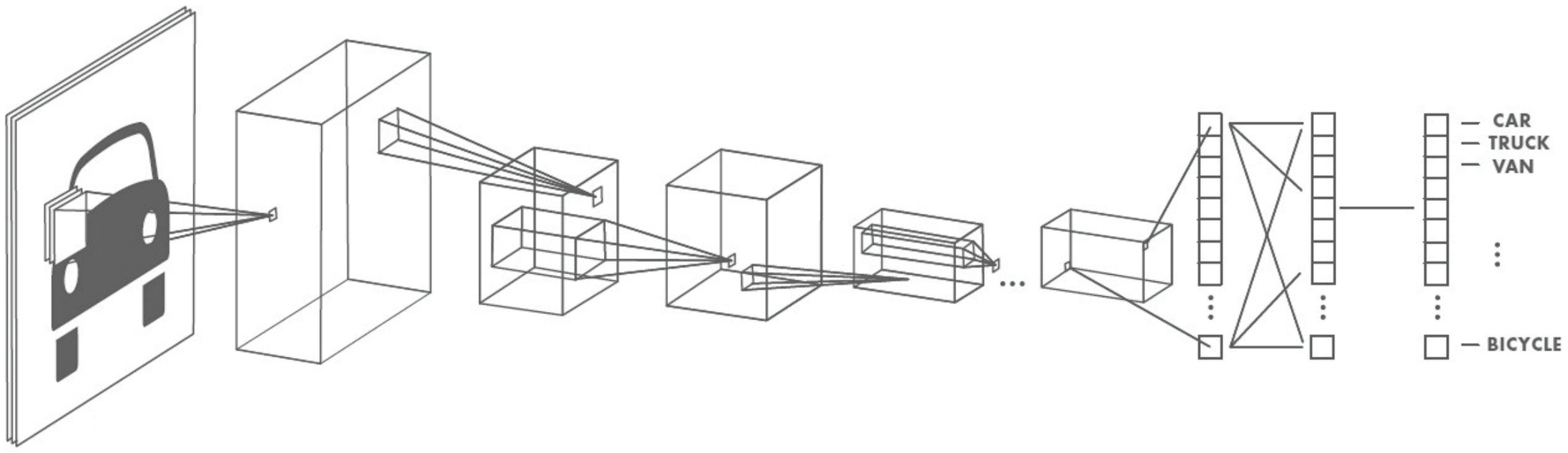


- **Example: 200x200 image**
  - 400,000 hidden units with 10x10 fields = 1000 params
  - 10 feature maps of size 200x200, 10 filters of size 10x10



Credit: Yann LeCun

# Key properties of CNN




- ☺ Convolutional filters **(Translation invariance+Self-similarity)**
- ☺ Multiple layers **(Compositionality)**
- ☺ Filters localized in space **(Locality)**
- ☺ $\mathcal{O}(1)$ parameters per filter (independent of input image size $n$)
- ☺ $\mathcal{O}(n)$ complexity per layer (filtering done in the spatial domain)
- ☺ $\mathcal{O}(\log n)$ layers in classification tasks

LeCun et al. 1989

Slide: M. Bronstein

# Image-based deep network evolution

## IMAGENET Large Scale Visual Recognition Challenge

### Year 2010

NEC-UIUC

Dense descriptor grid: HOG, LBP

Coding: local coordinate, super-vector

Pooling, SPM

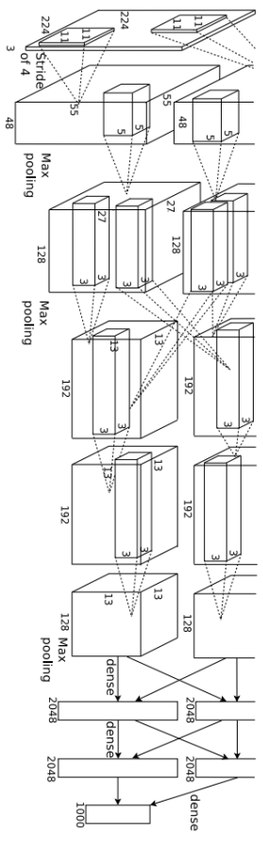
Linear SVM

[Lin CVPR 2011]



### Year 2012

SuperVision

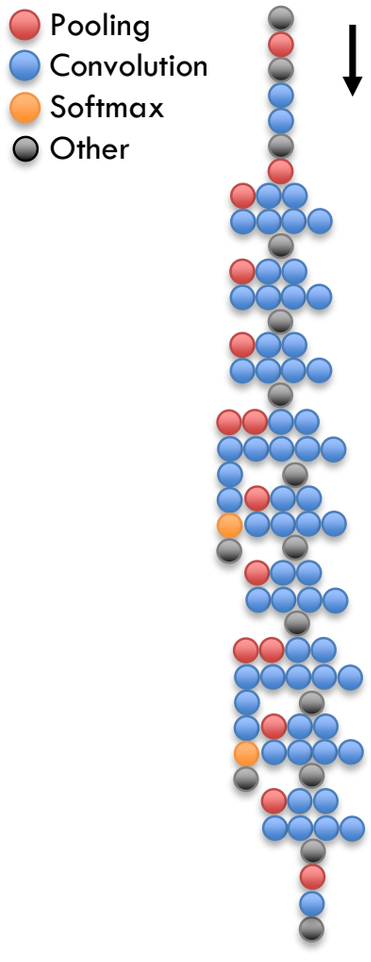



[Krizhevsky NIPS 2012]



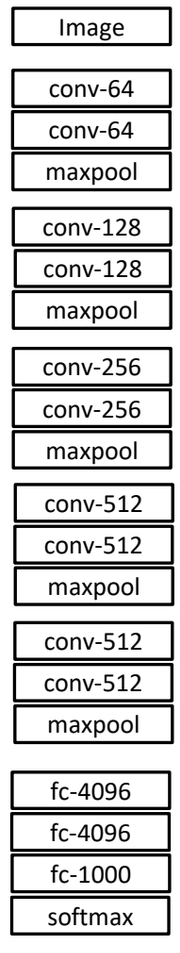
### Year 2014

GoogLeNet




[Szegedy arxiv 2014]

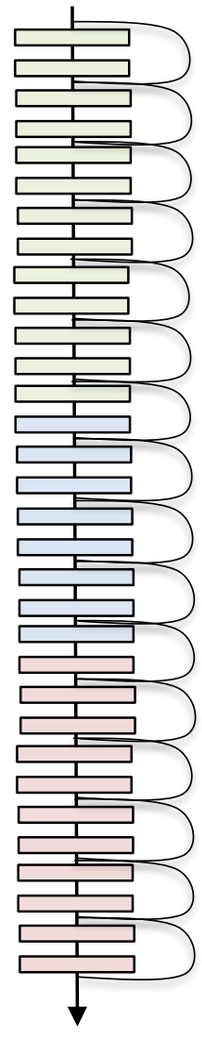
VGG




[Simonyan arxiv 2014]

### Year 2015

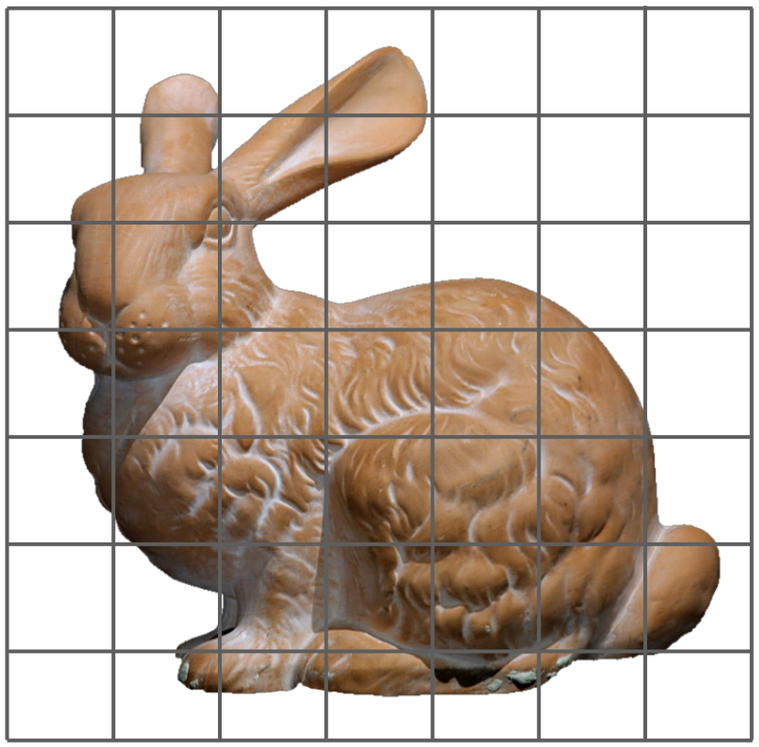
MSRA

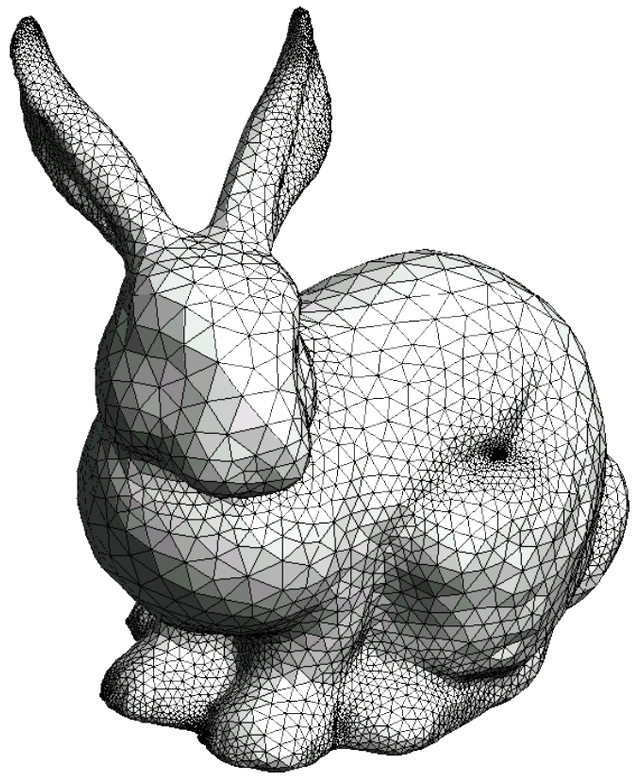


[He ICCV 2015]

Slide: CS231n @ Stanford

# Deep learning for 2D vs. 3D data





Deep Neural network

?

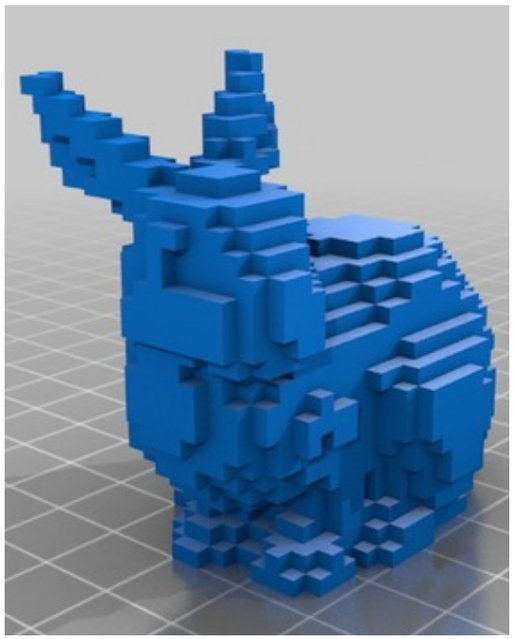
Deep Neural network

“Stanford bunny”

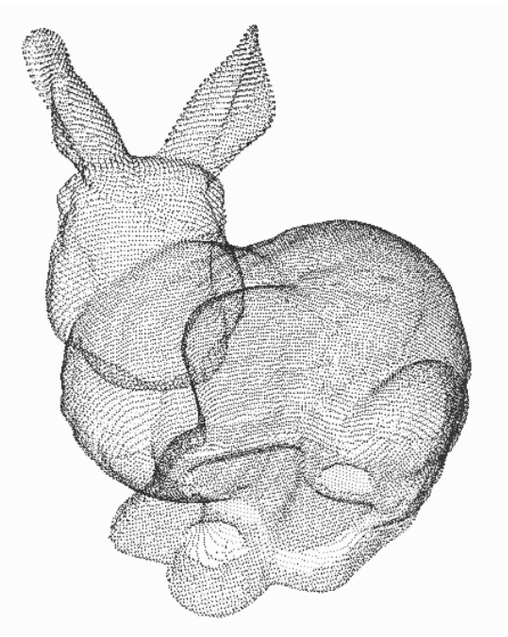
“Stanford bunny”

# 3D Representations

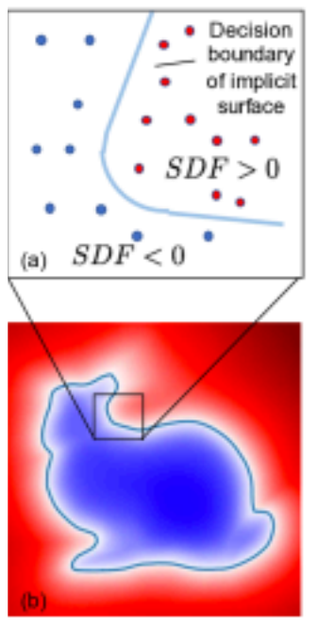
Image-based



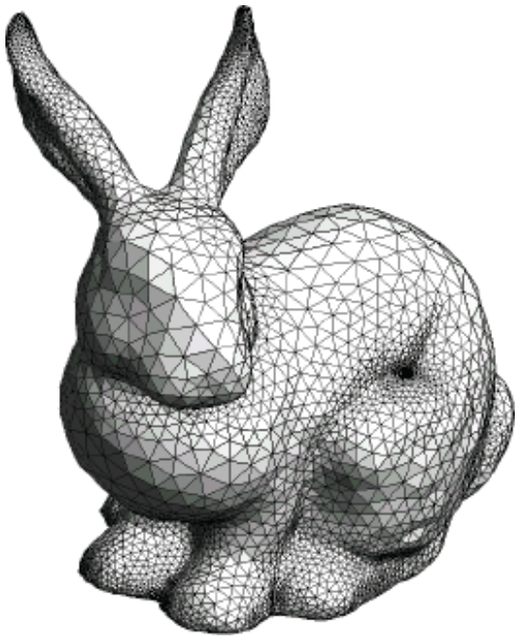
Volumetric



Point Cloud




Signed distance-based



Mesh-based

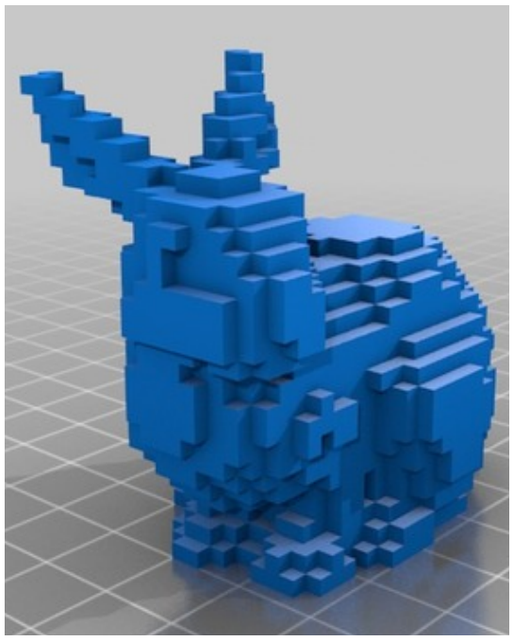
Credit: Michael Bronstein, Charles R. Qi, Jeong J. Park

# Course Outline

- [x] Deep learning: short introduction
- [x] 3D representations
  - [ ] Deep learning for image and voxel-based representations
  - [ ] Deep learning on point clouds
  - [ ] Deep learning implicit representations
  - [ ] Deep learning on graphs and meshes
- [ ] Frameworks, datasets, relates courses

# 3D Representations

**Image-based**



**Volumetric**



Point Cloud



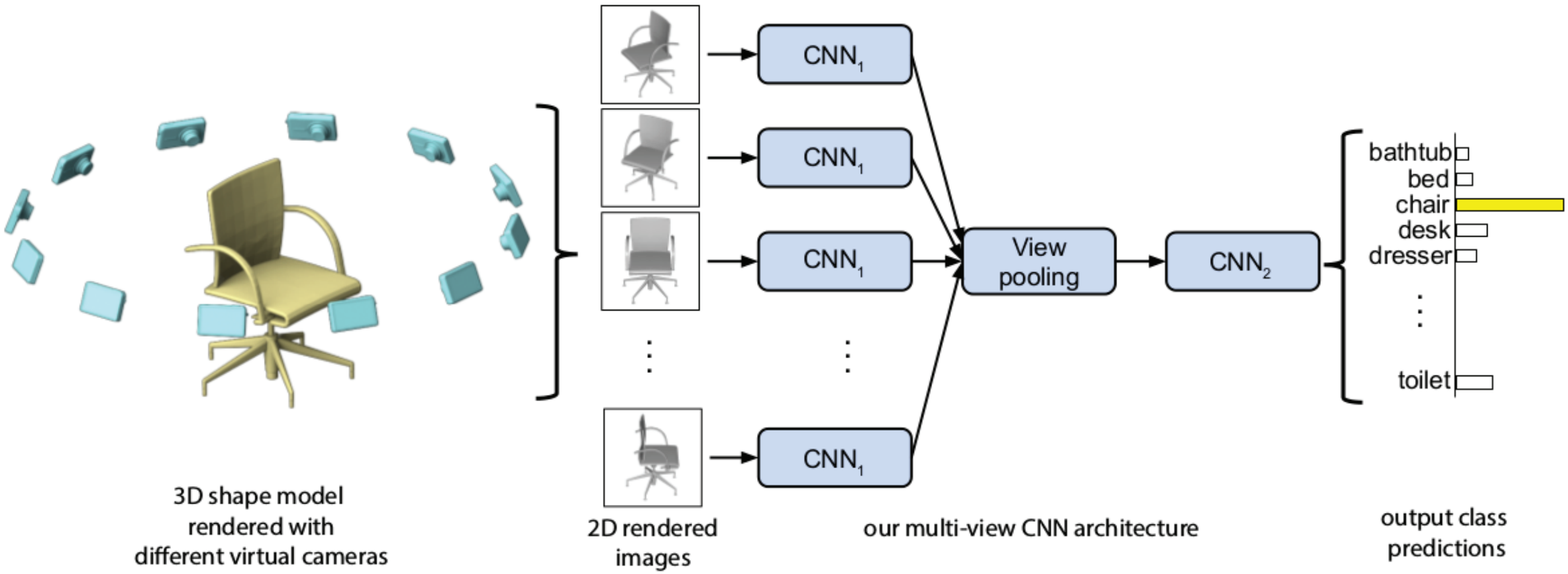

Signed distance-based



Mesh-based

Credit: Michael Bronstein, Charles R. Qi, Jeong J. Park

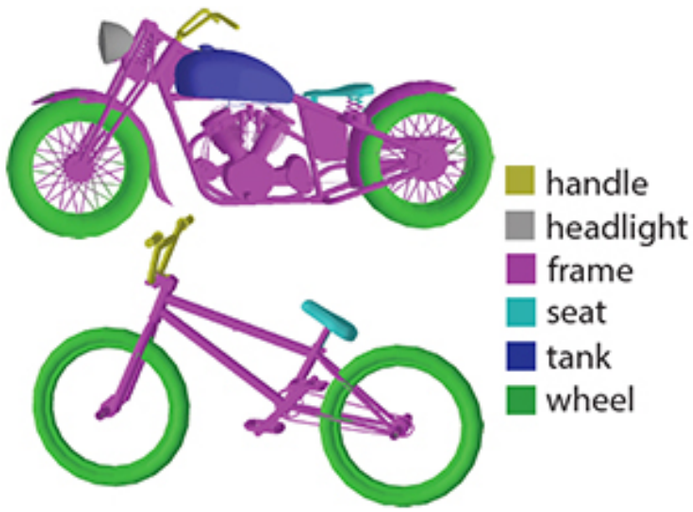
# Image-based representations: Rendered Views




**Multi-view CNN**

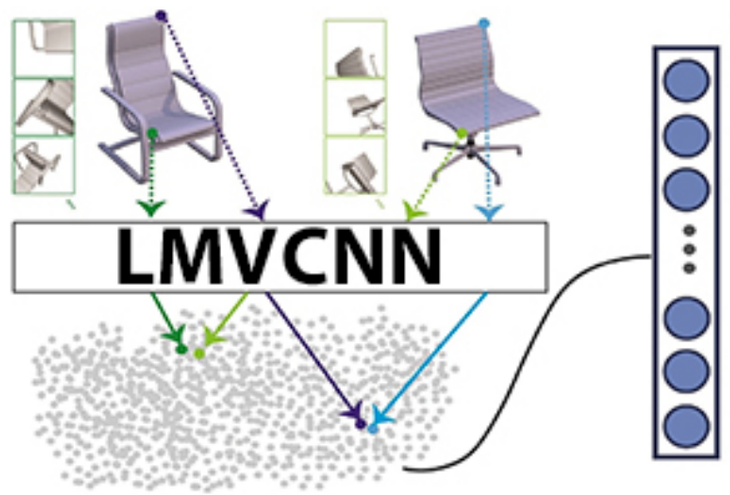
Hang Su et al., ICCV 2015

# Image-based representations: Rendered Views



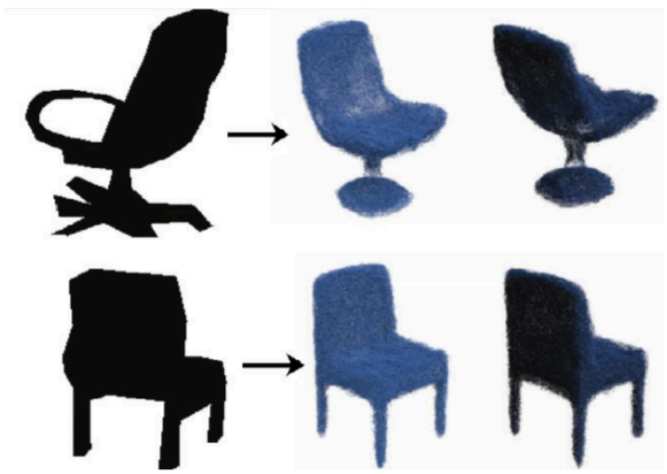

**3D shape segmentation**

E. Kalogerakis et al., CVPR 2017




**Local shape descriptor learning**

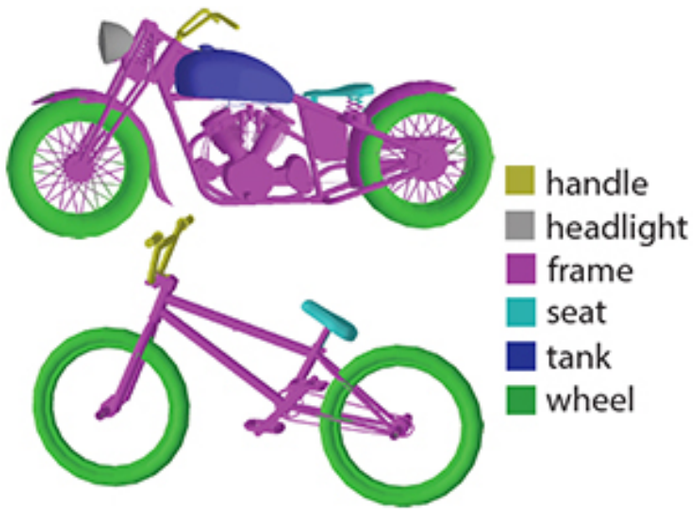
Haibin Huang et al., TOG 2018



**3D shape synthesis via silhouettes**

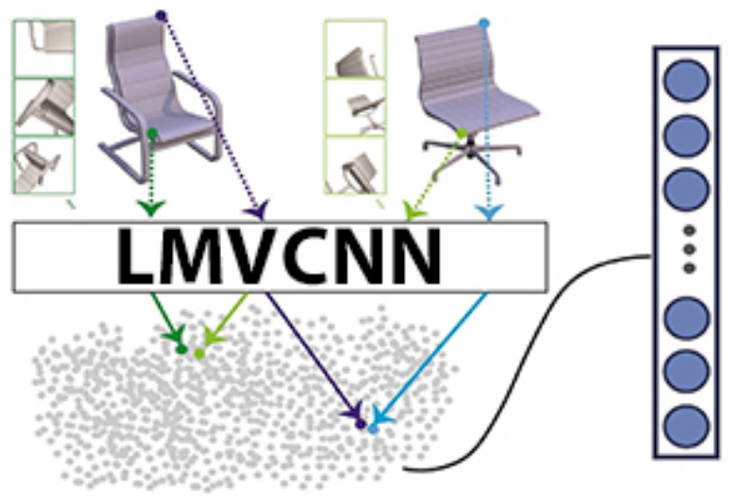
A. Soltani et al., CVPR 2017

# Image-based representations: Rendered Views



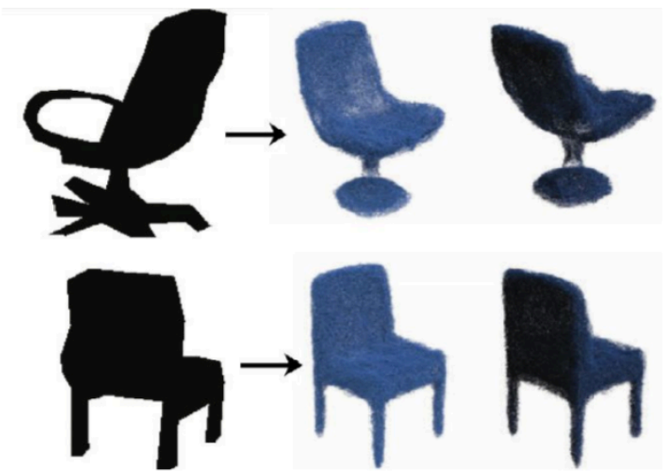

**3D shape segmentation**

E. Kalogerakis et al., CVPR 2017




**Local shape descriptor learning**

Haibin Huang et al., TOG 2018



**3D shape synthesis via silhouettes**

A. Soltani et al., CVPR 2017

✅ Simple, re-use standard components of CNNs

✅ Efficient, good results

⛔ Memory

⛔ Not geometric

⛔ No invariance

# Image-based representations: Mapping to flat domain

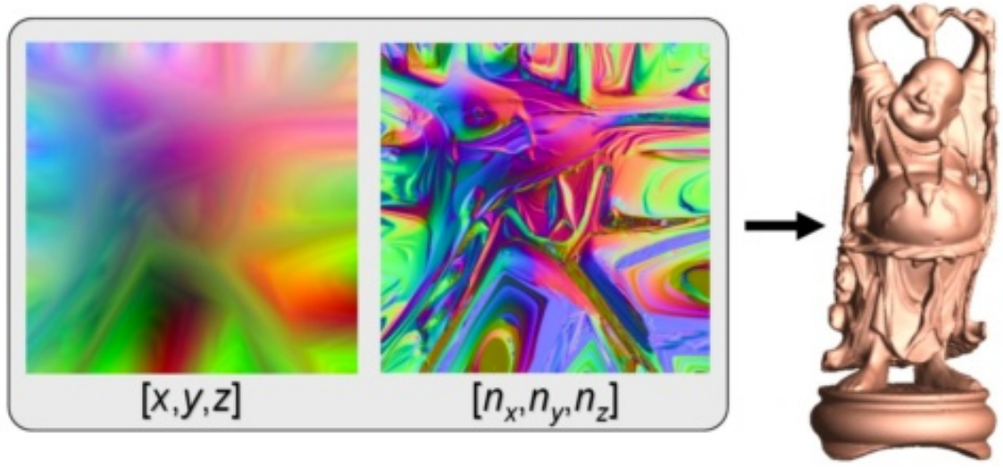



Image: H.Hoppe

**Deep learning 3D shapes using geometry images**

A. Sinha et al., ECCV 2016

# Image-based representations: Mapping to flat domain

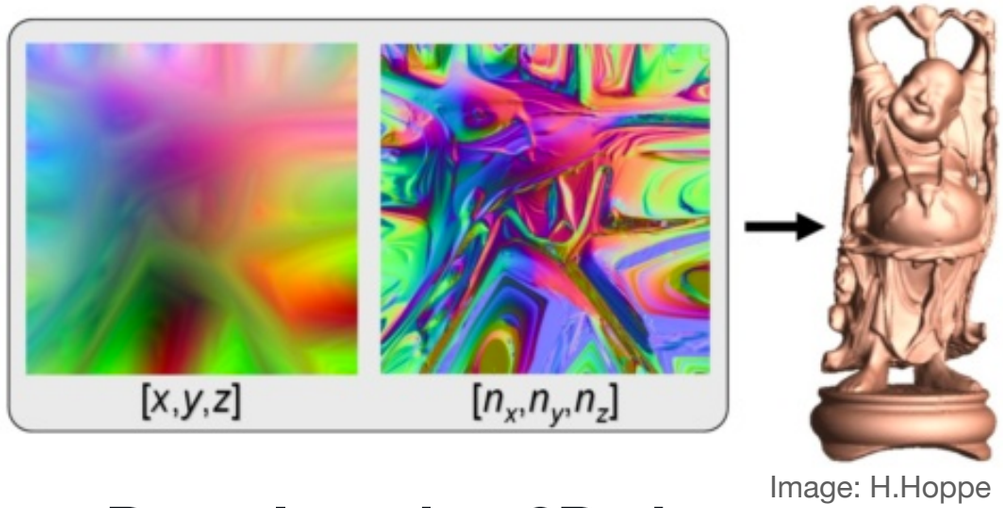
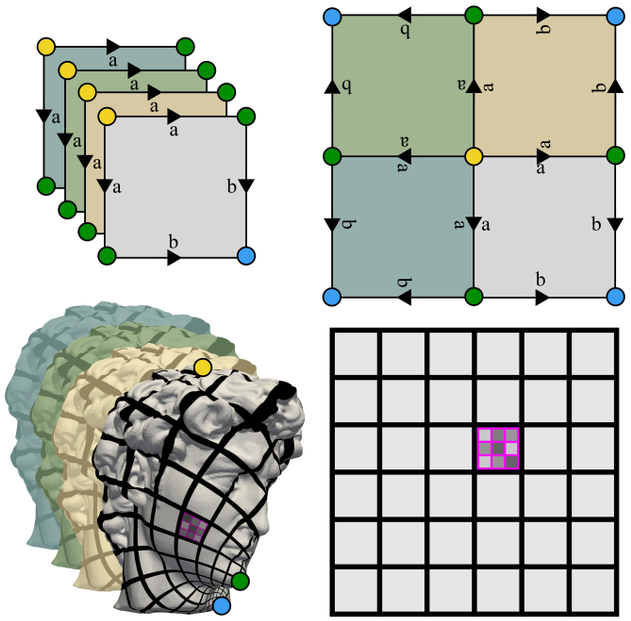



Image: H.Hoppe

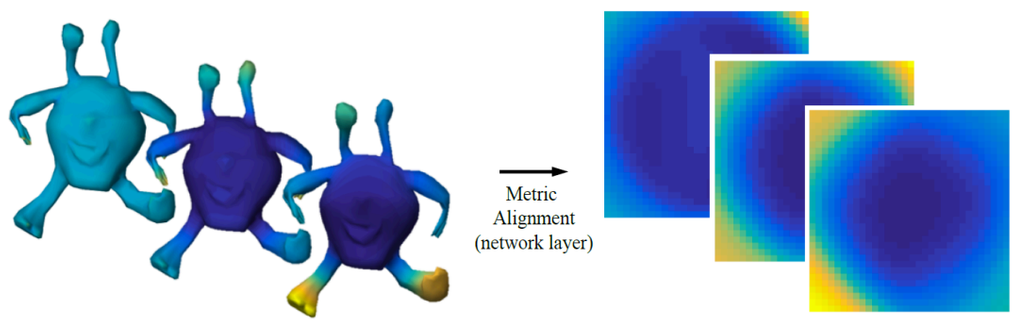
**Deep learning 3D shapes using geometry images**

A. Sinha et al., ECCV 2016



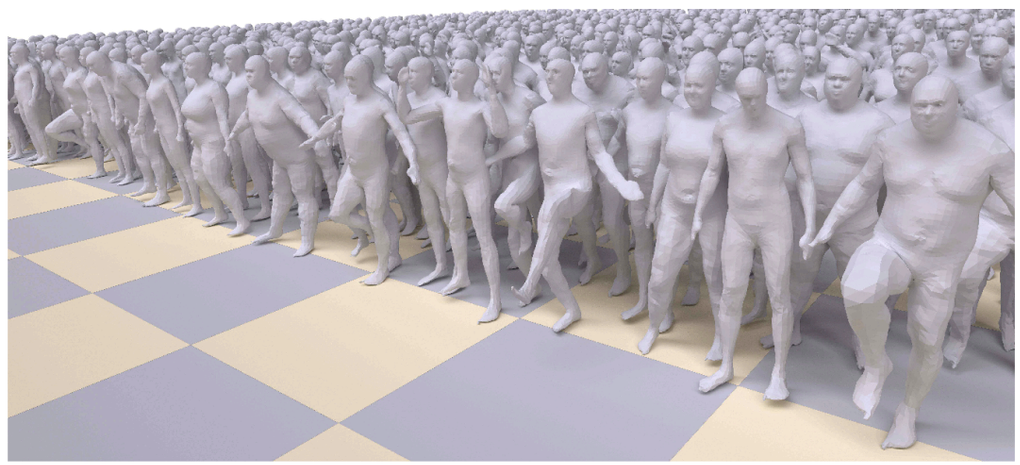
**Seamless Toric Covers**

H. Maron et al., SIGGRAPH 2017




**Gromov–Wasserstein CNN**

D. Ezuz et al., SGP 2017



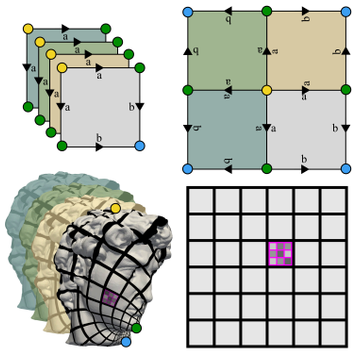
**Multi-chart Generative Surface Modeling**

H. Ben-Hamu et al., SIGGRAPH Asia 2018

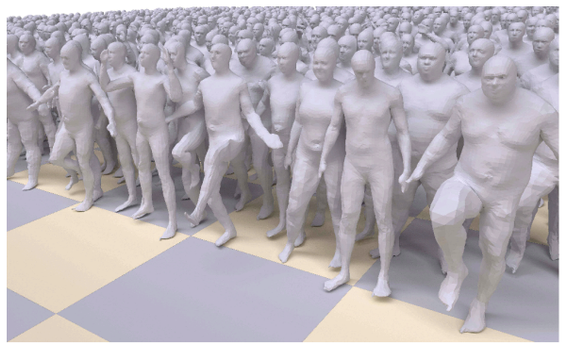
# Image-based representations: Mapping to flat domain


Image: H.Hoppe

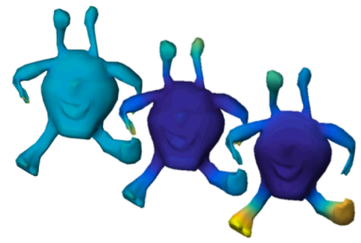
**Deep learning 3D shapes using geometry images**
A. Sinha et al., ECCV 2016



**Seamless Toric Covers**
H. Maron et al., SIGGRAPH 2017



**Multi-chart Generative Surface Modeling**
H. Ben-Hamu et al., SIGGRAPH Asia 2018



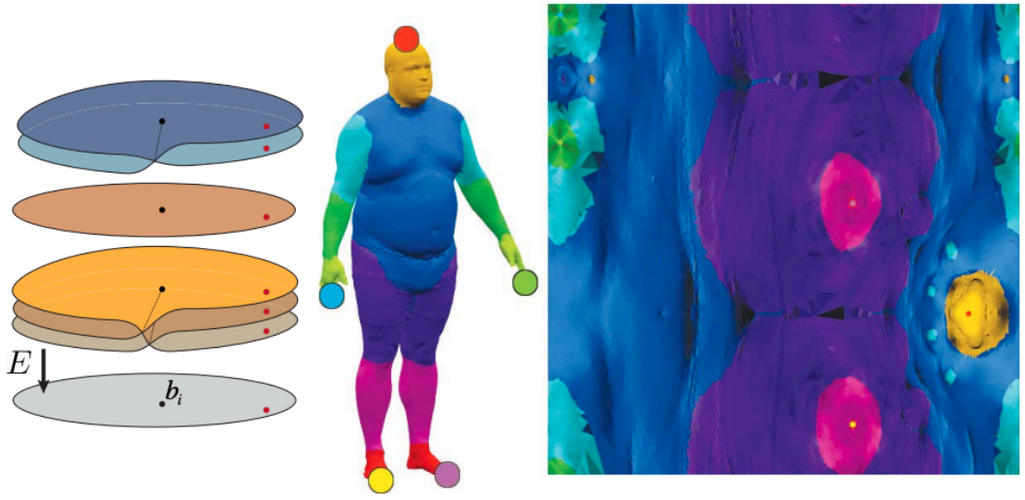
**Gromov–Wasserstein CNN**
D. Ezuz et al., SGP 2017

✅ Efficient

✅ Deformation invariant*

⛔ Mapping distortion

⛔ Genus-zero surfaces



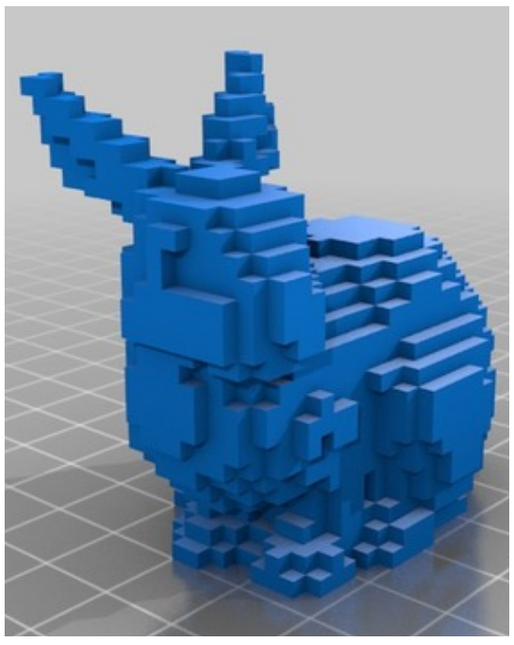

**Surface Networks via General Covers**
N. Hain et al., arXiv 2019

*If the mapping is deformation invariant

# 3D Representations



Image-based



**Volumetric**



Point Cloud



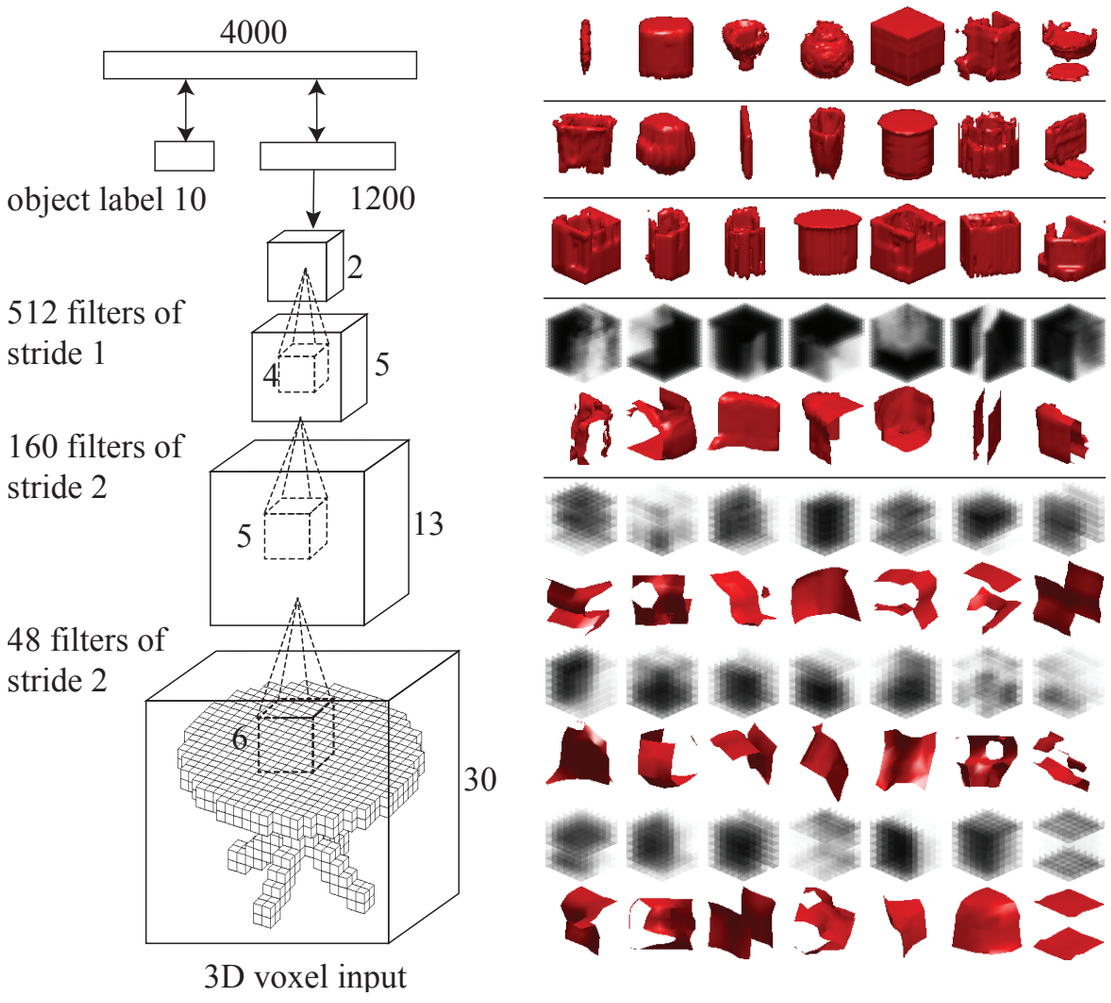

Signed distance-based



Mesh-based

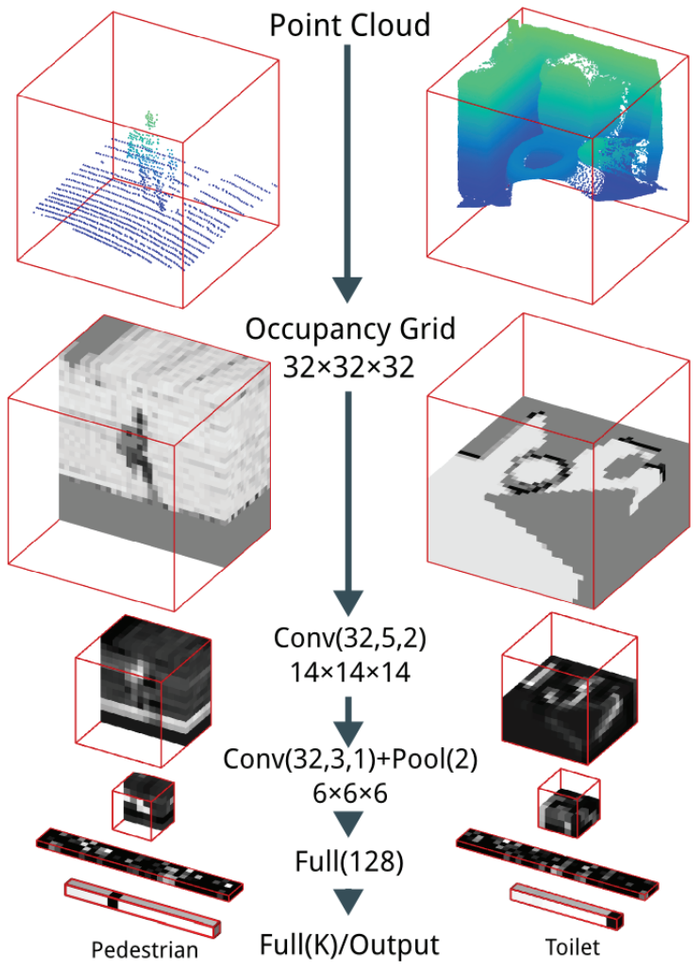
Credit: Michael Bronstein, Charles R. Qi, Jeong J. Park

# Volumetric Representations




**3D ShapeNets**

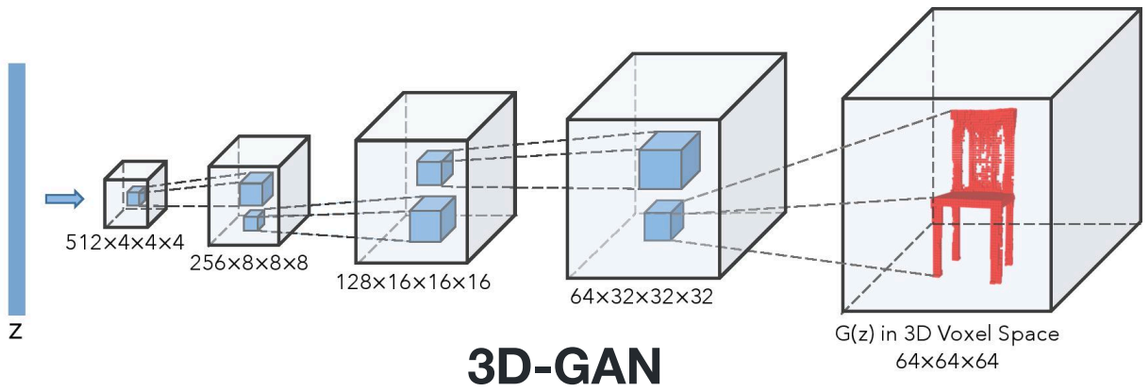
Z. Wu et al., CVPR 2015




**VoxNet**

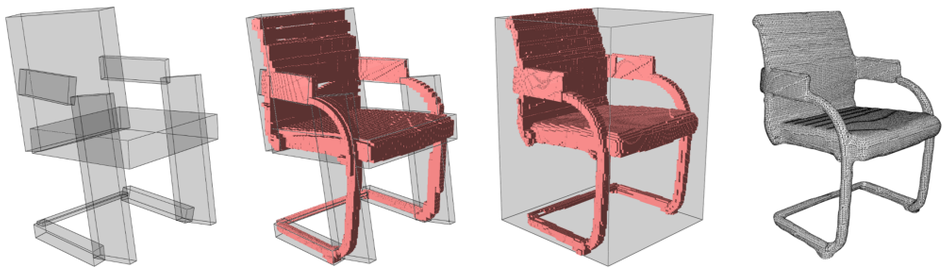
Maturana and Scherer, IROS 2015

# Volumetric Representations



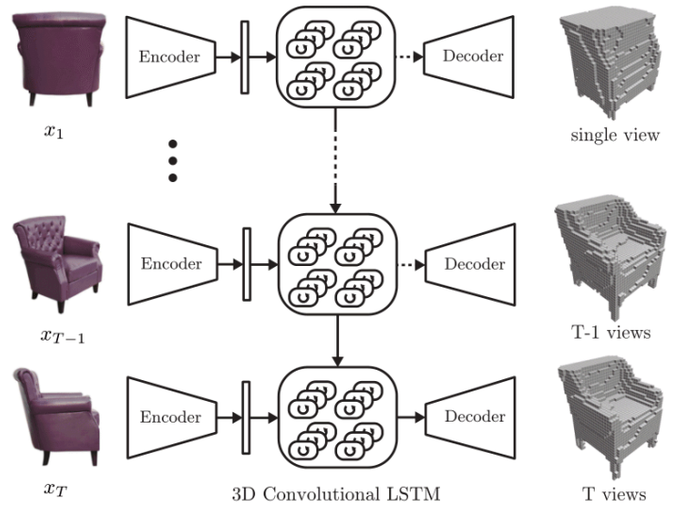

**3D-GAN**

J. Wu et al., NIPS 2016



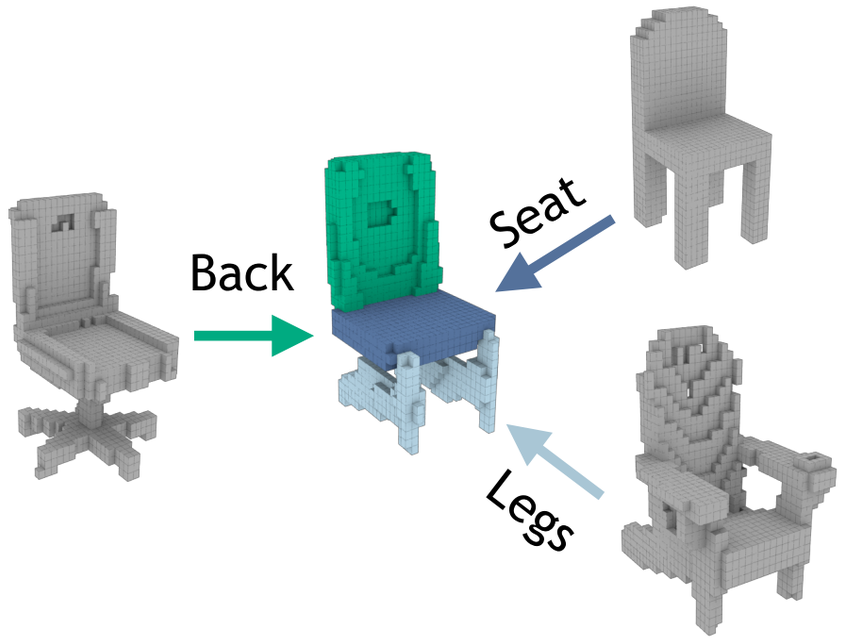
**GRASS: Generative Recursive Autoencoders**

Jun Li et al., SIGGRAPH 2017




**3D Recurrent Reconstruction**

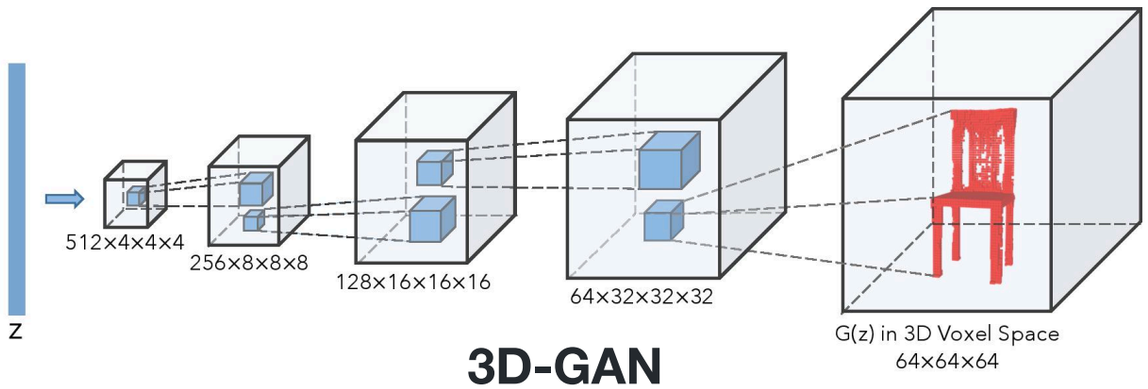
C. Choy et al., ECCV 2016




**Decomposer-Composer Network**

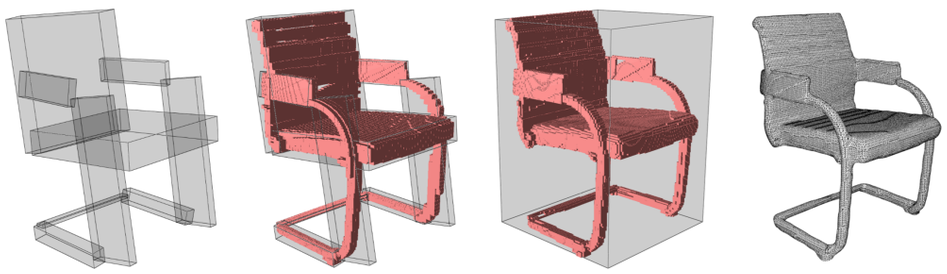
Dubrovina et al., ICCV 2019

# Volumetric Representations




**3D-GAN**

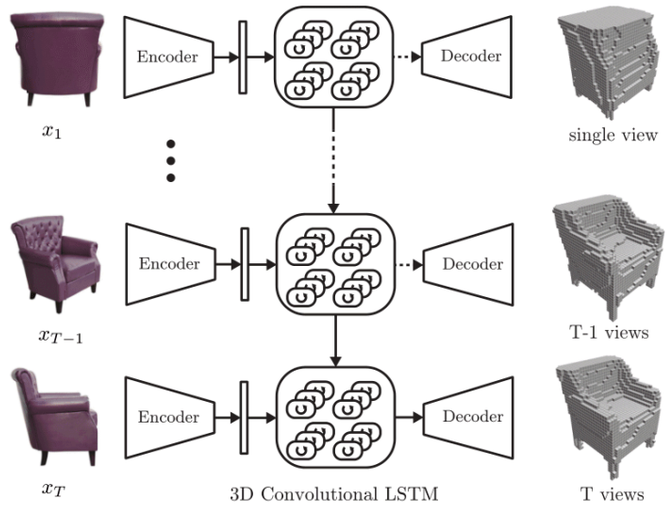
J. Wu et al., NIPS 2016



**GRASS: Generative Recursive Autoencoders**

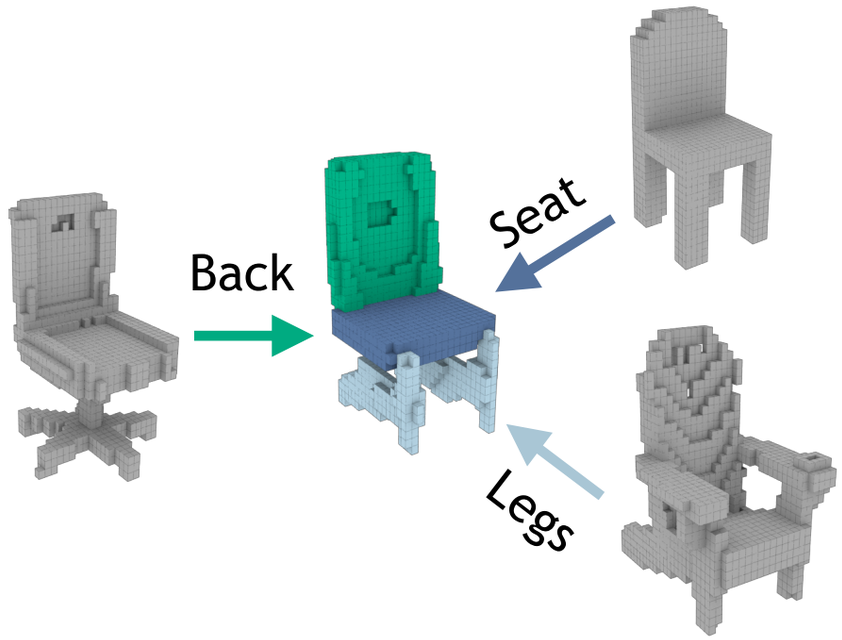
Jun Li et al., SIGGRAPH 2017

- ✅ Simple
- ⛔ Coarse
- ⛔ Memory
- ⛔ No invariance




**3D Recurrent Reconstruction**

C. Choy et al., ECCV 2016

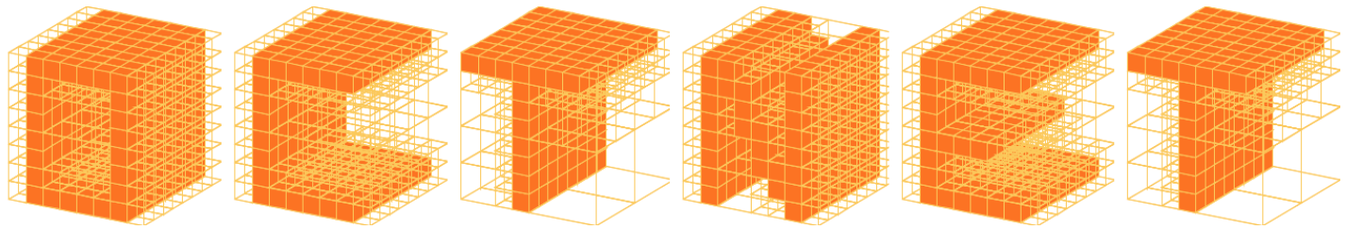



**Decomposer-Composer Network**

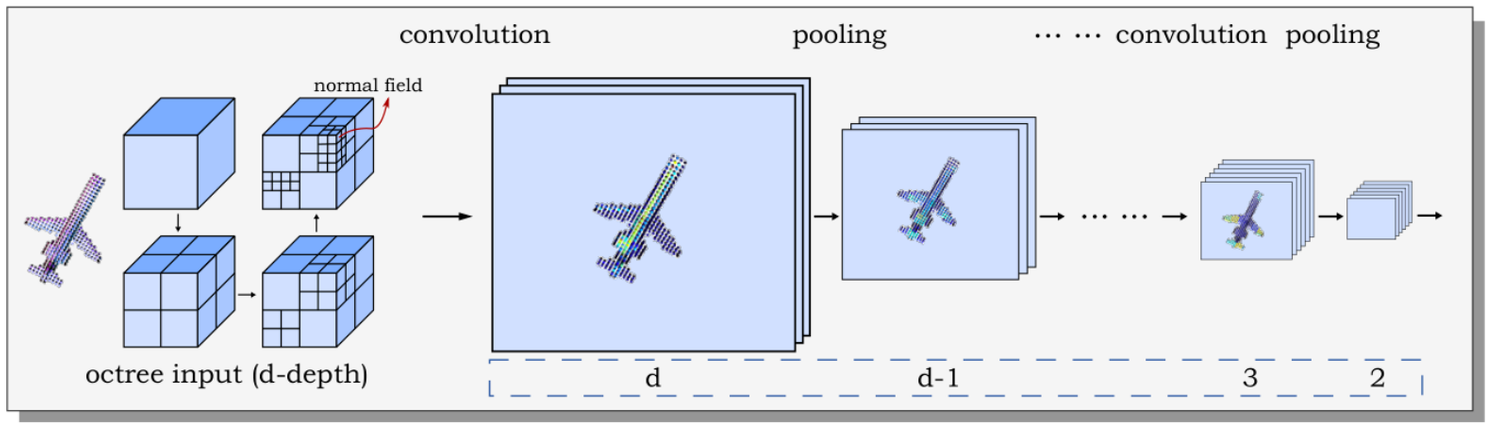
Dubrovina et al., ICCV 2019

# Volumetric Representations: Octree based-networks

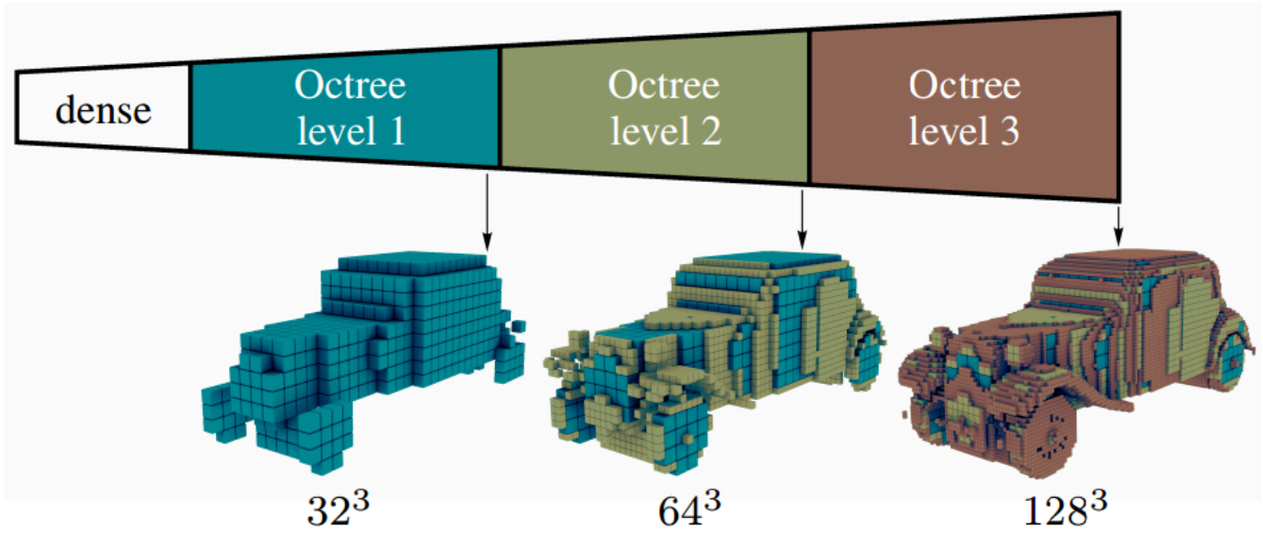
**OctNet**



**O-CNN**

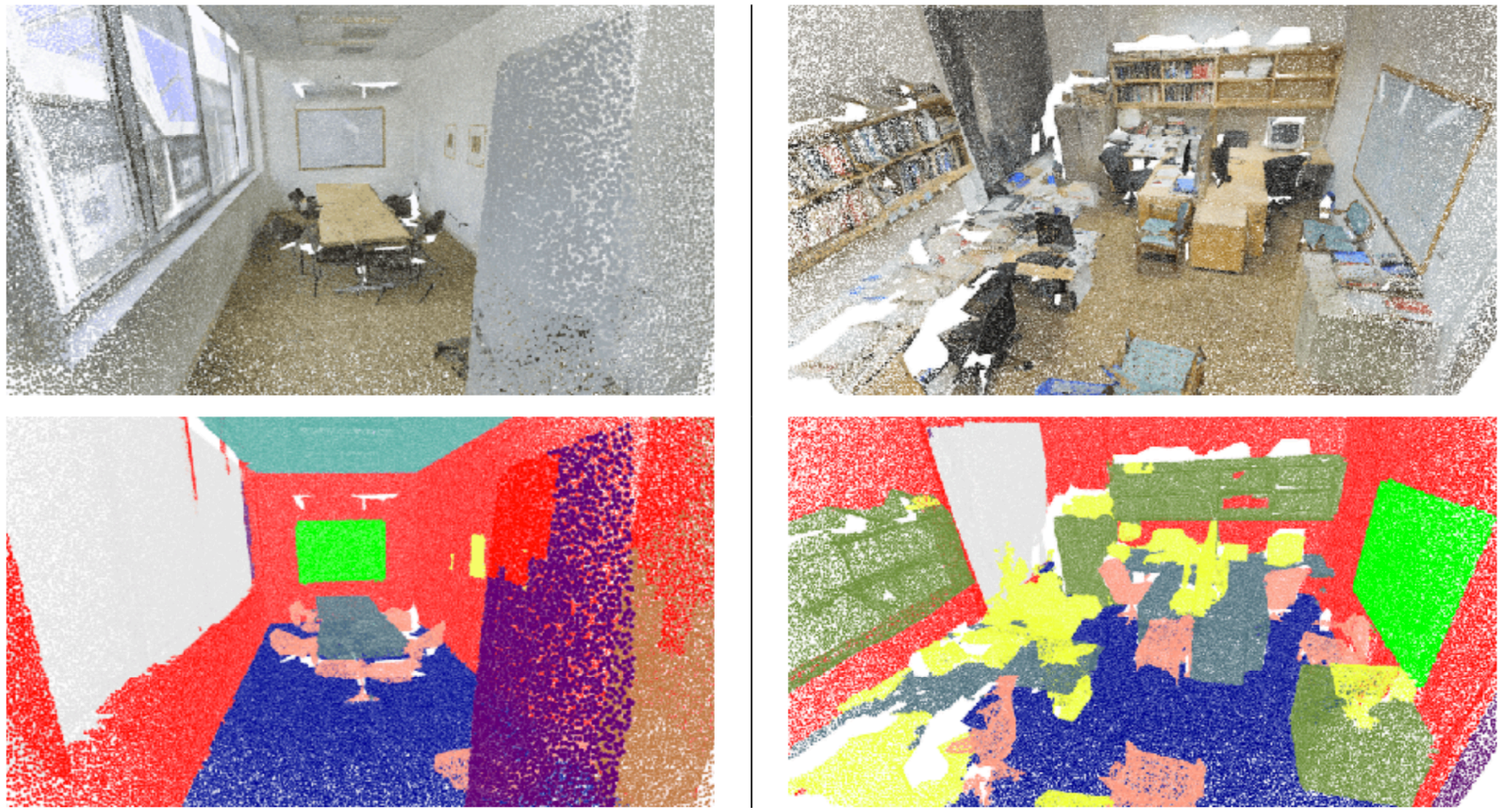



**Octree Generating Networks**

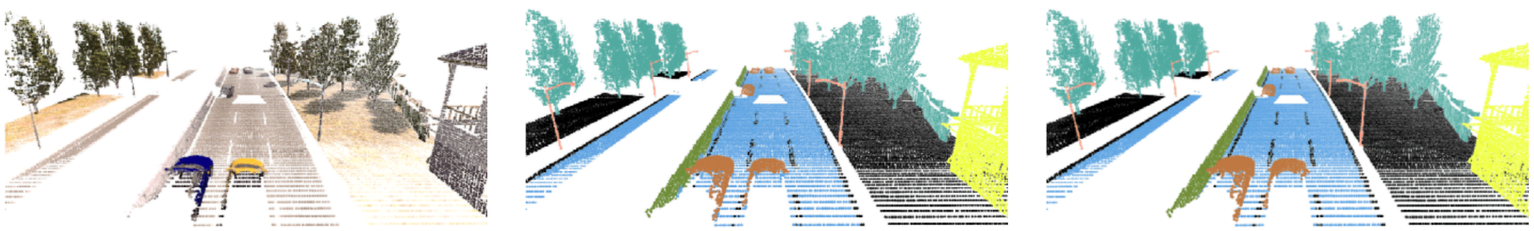



M. Tatarchenko et al. [ICCV 2017], G. Riegler [CVPR 2017], P.-S. Wang et al. [SIGGRAPH 2017]

# Volumetric Representations: Sparse convolutions



Input RGB pointcloud | Prediction result | Ground-truth label



**4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Network**

Choy et al., CVPR 2019

# Course Outline

- [x] Deep learning: short introduction
- [x] 3D representations
  - [x] Deep learning for image and voxel-based representations
  - [ ] Deep learning on point clouds
  - [ ] Deep learning implicit representations
  - [ ] Deep learning on graphs and meshes
- [ ] Frameworks, datasets, relates courses

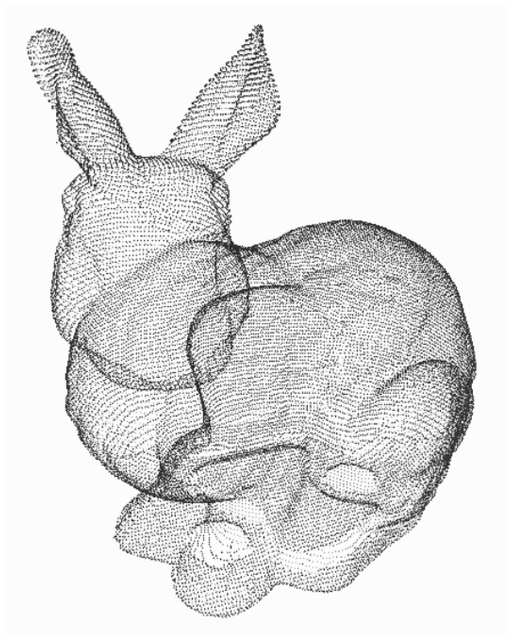
# 3D Representations



Image-based



Volumetric



**Point Cloud**




Signed distance-based

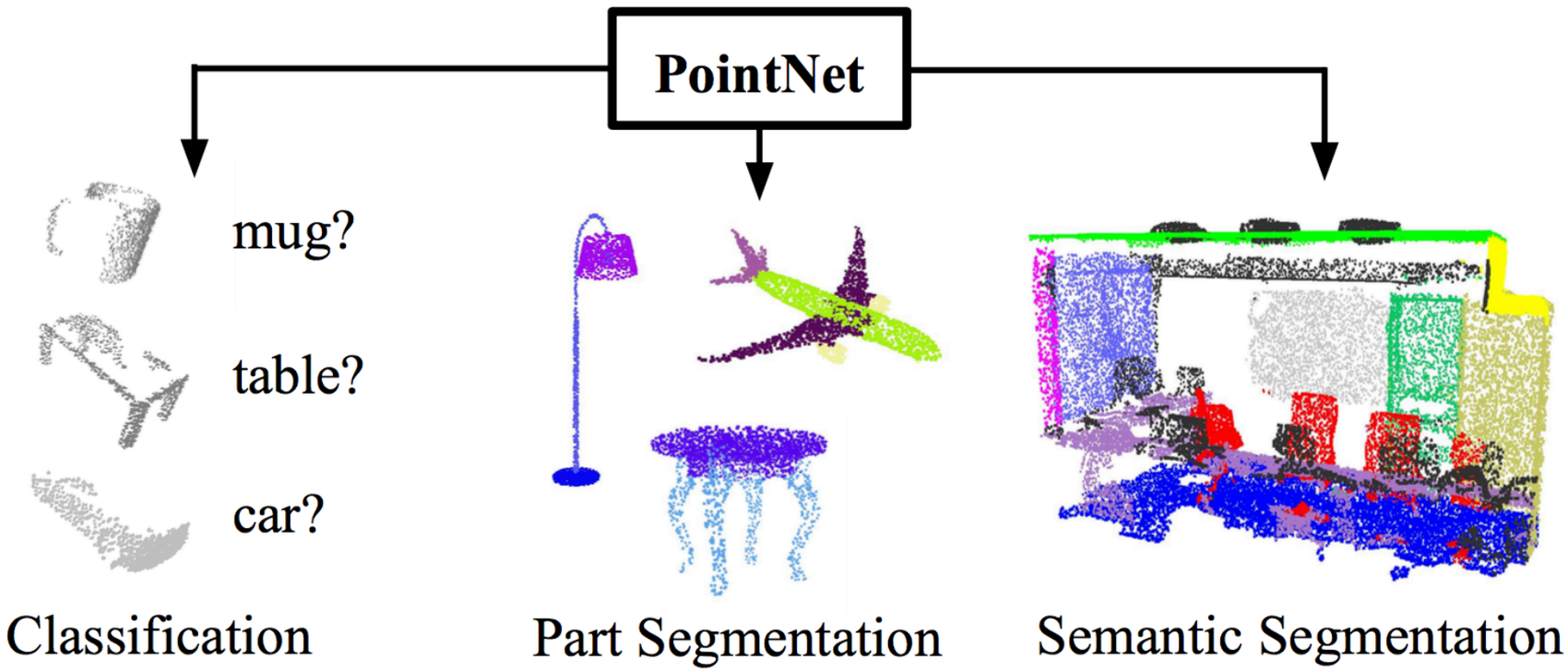


Mesh-based

Credit: Michael Bronstein, Charles R. Qi, Jeong J. Park

# PointNet

- **End-to-end learning** for irregular point data
- **Unified** framework for various tasks

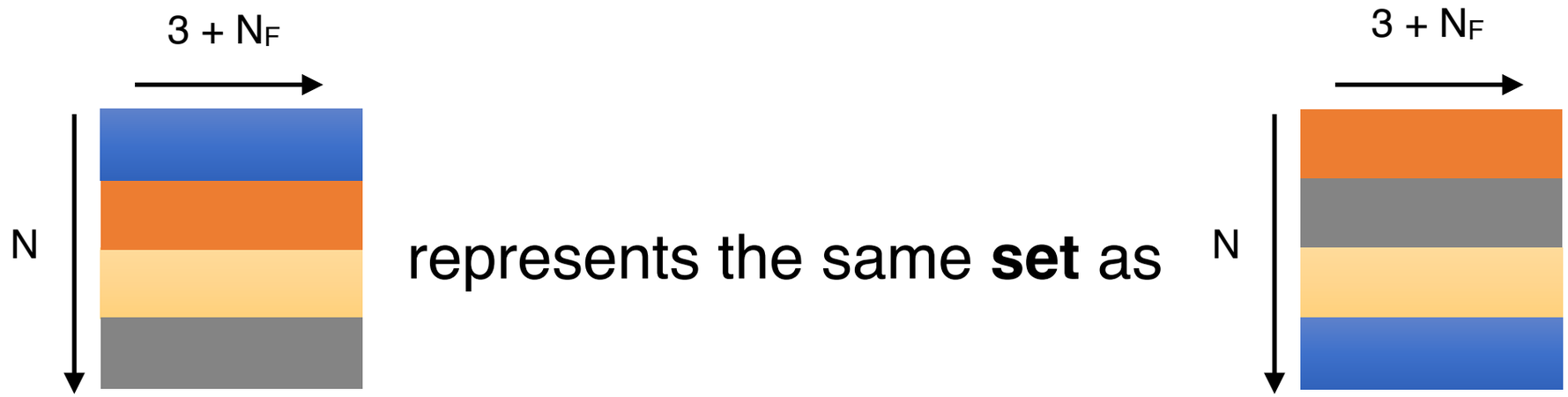



PointNet, Qi et al., CVPR 2017

# PointNet

- **Permutation invariance (equivariance)**

  Point cloud is a set of *unordered* points



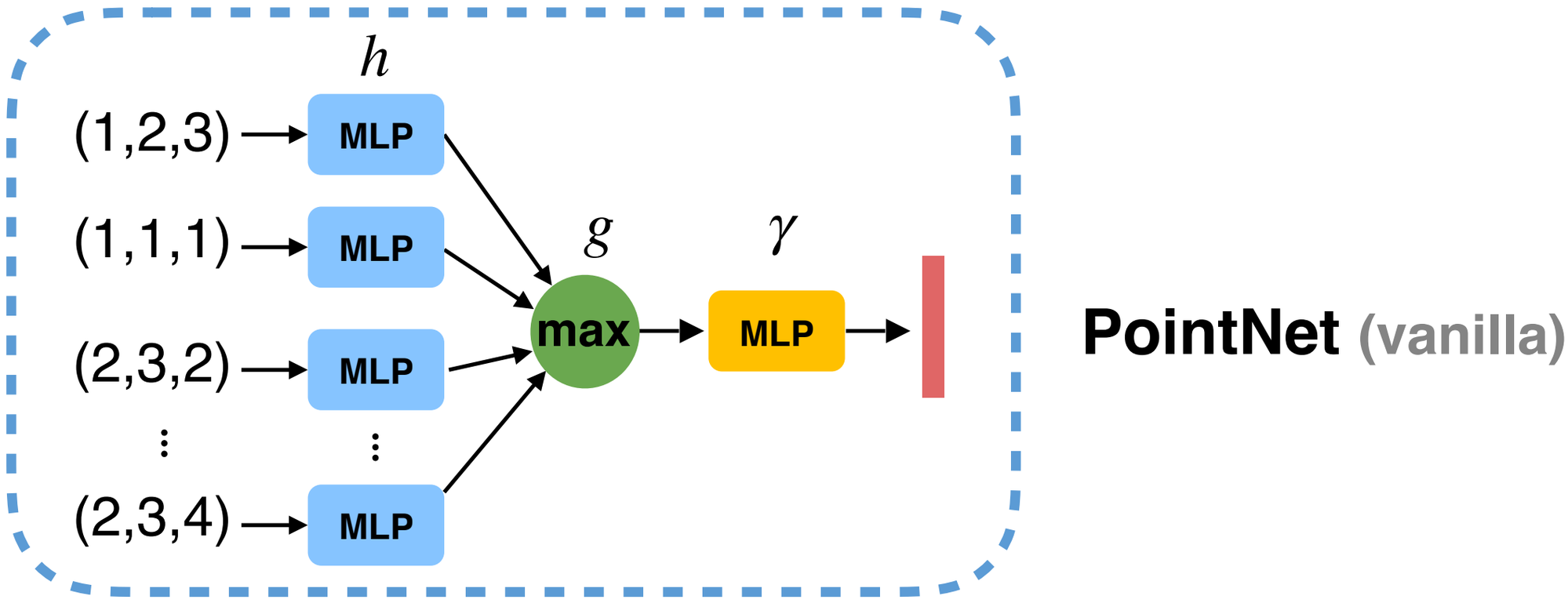

- **(Partial) rigid transformation invariance**

  Point cloud rotations should not alter classification results

Slide: Charles R. Qi

# PointNet

- **Permutation Invariance**

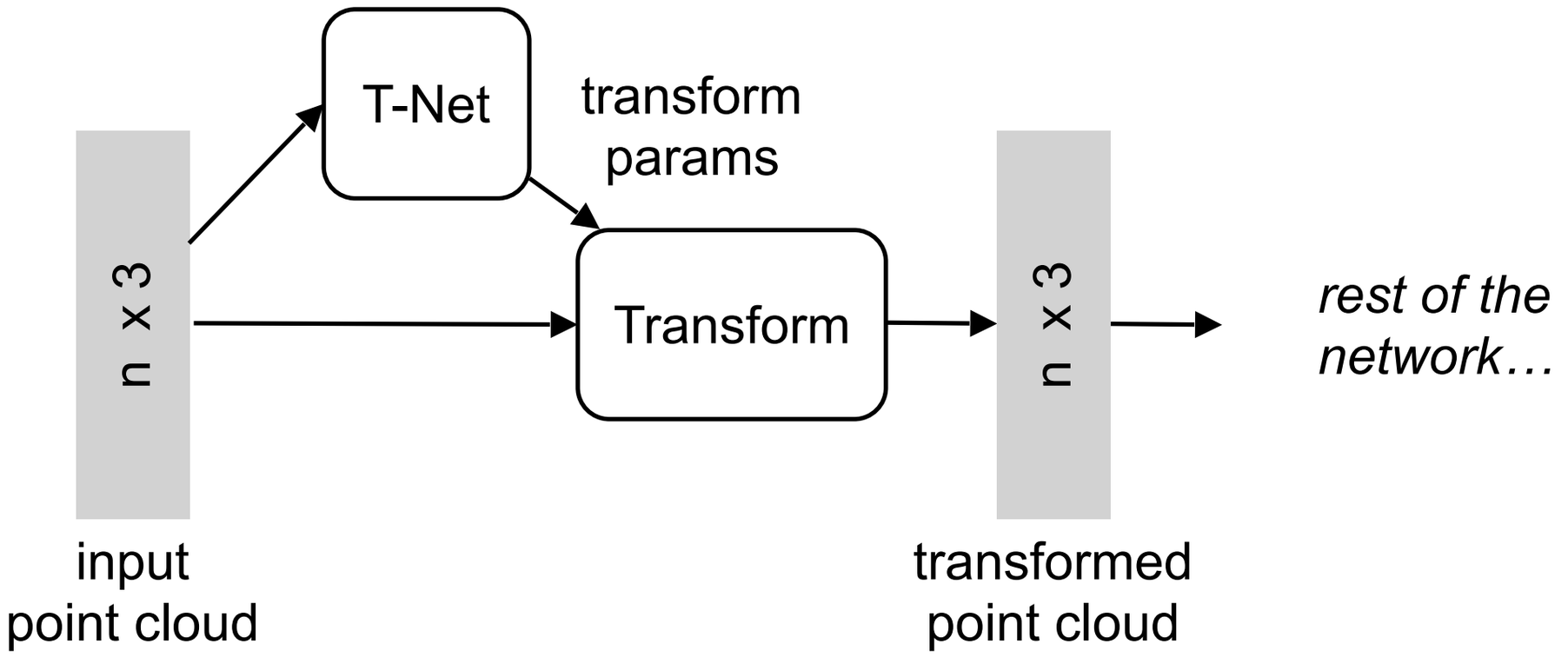



$g\,(h(x_1), h(x_2), ..., h(x_n))$ is symmetric (w.r.t. order of input points)

Slide: Charles R. Qi

# PointNet

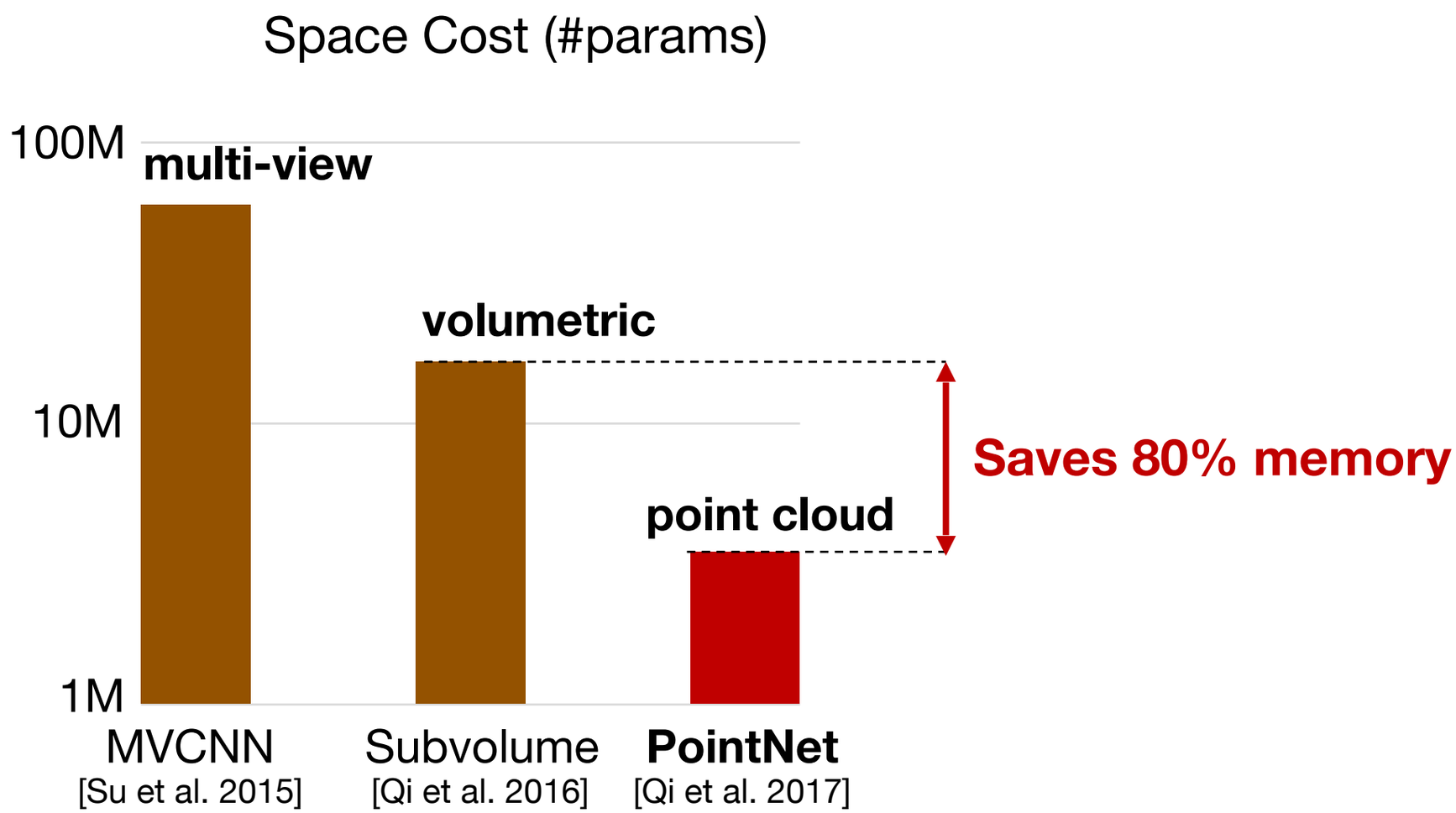
- **Transformation Invariance**




Slide: Charles R. Qi

# PointNet

- Memory requirement decrease

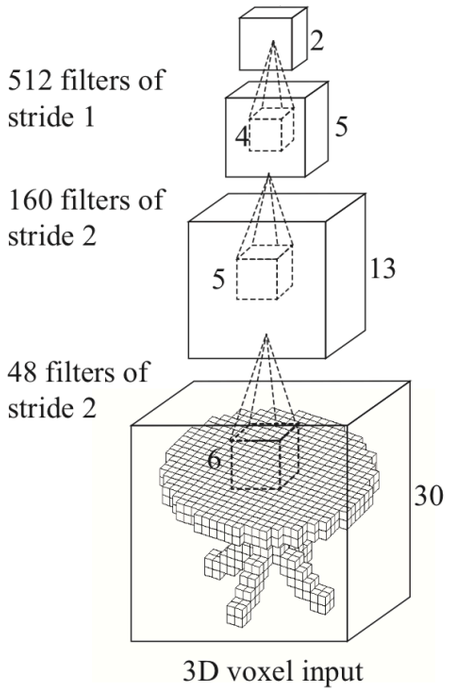



- Comparable decrease in computation time

Slide: Charles R. Qi

# PointNet limitations

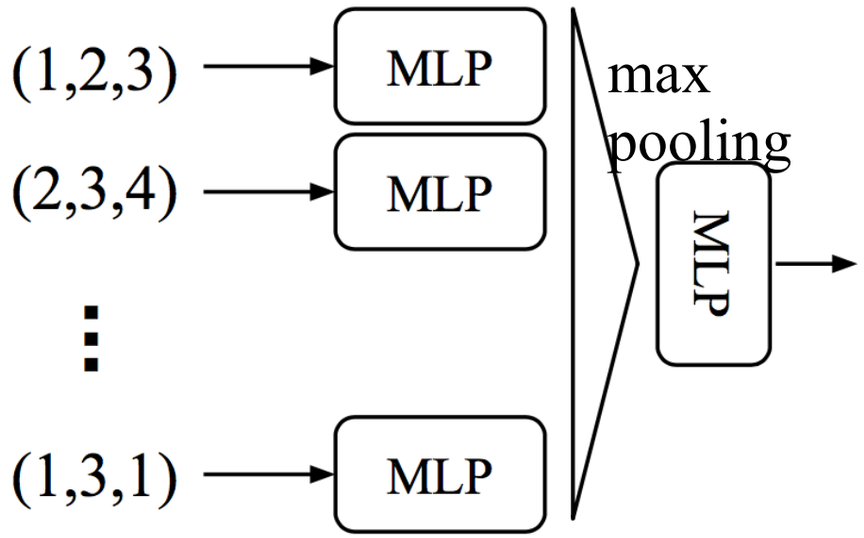
**Hierarchical feature learning**
multiple levels of abstraction




3D CNN [Wu et al.2015]

**v.s.**

**Global feature learning**
either **one** point or **all** points

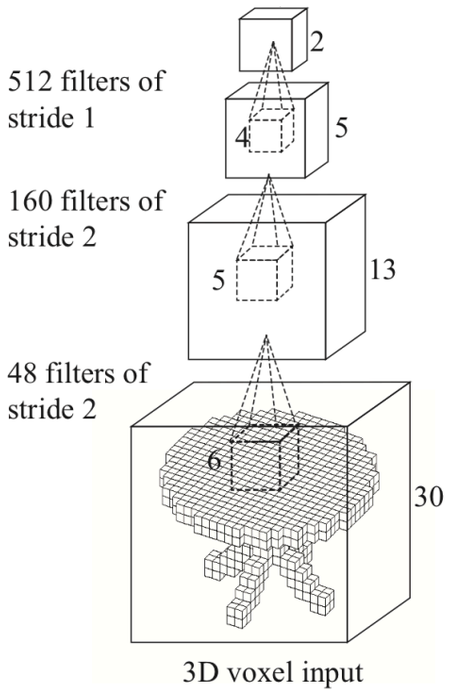



PointNet (vanilla) [Qi et al.2017]

Slide: Charles R. Qi

# PointNet limitations

**Hierarchical feature learning**
multiple levels of abstraction




3D CNN [Wu et al.2015]

**v.s.**

**Global feature learning**
either **one** point or **all** points

**No local context**

**Limited translation invariance**

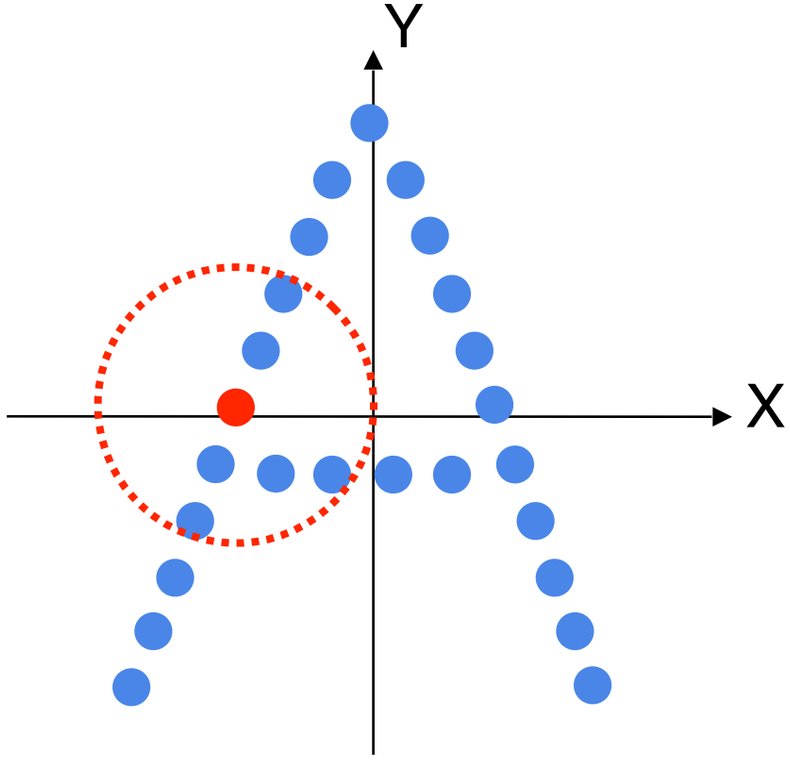
PointNet (vanilla) [Qi et al.2017]

Slide: Charles R. Qi

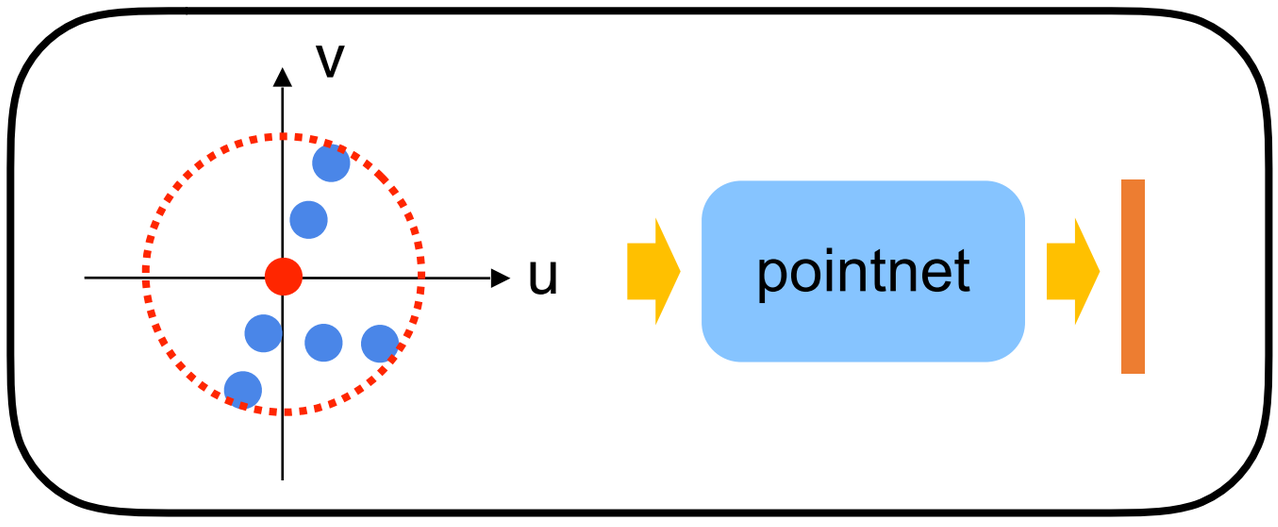
# PointNet++

Basic idea: Apply pointnet at local regions.

- ✓ Hierarchical feature learning
- ✓ Translation invariant
- ✓ Permutation invariant



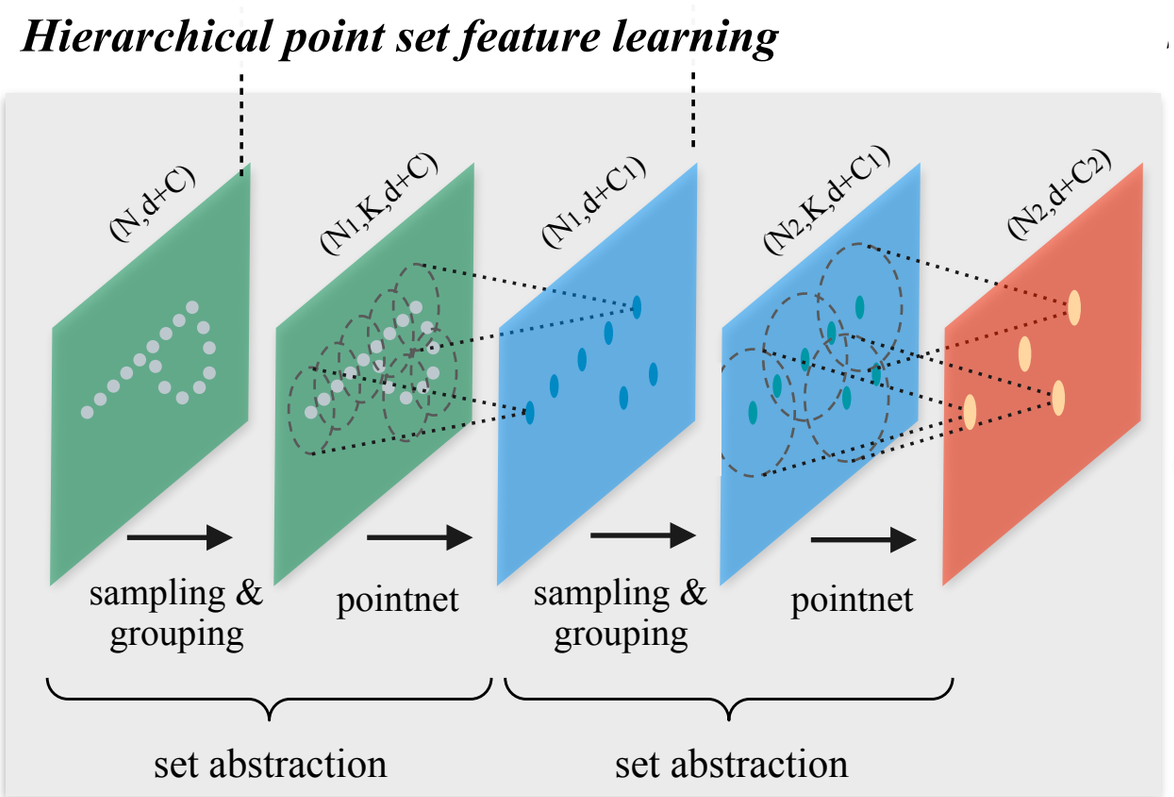

N points in (X,Y)

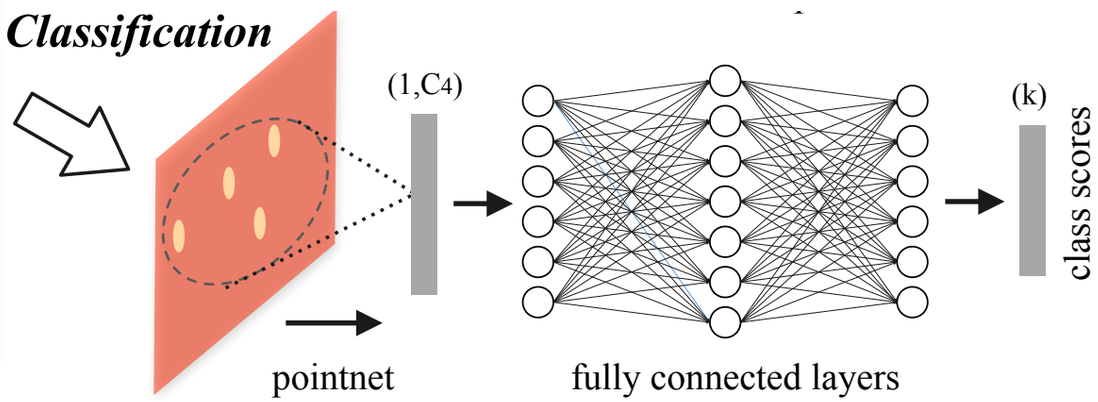



k points in local coordinates (u,v)

PointNet++, Qi et al., NeurIPS, 2018

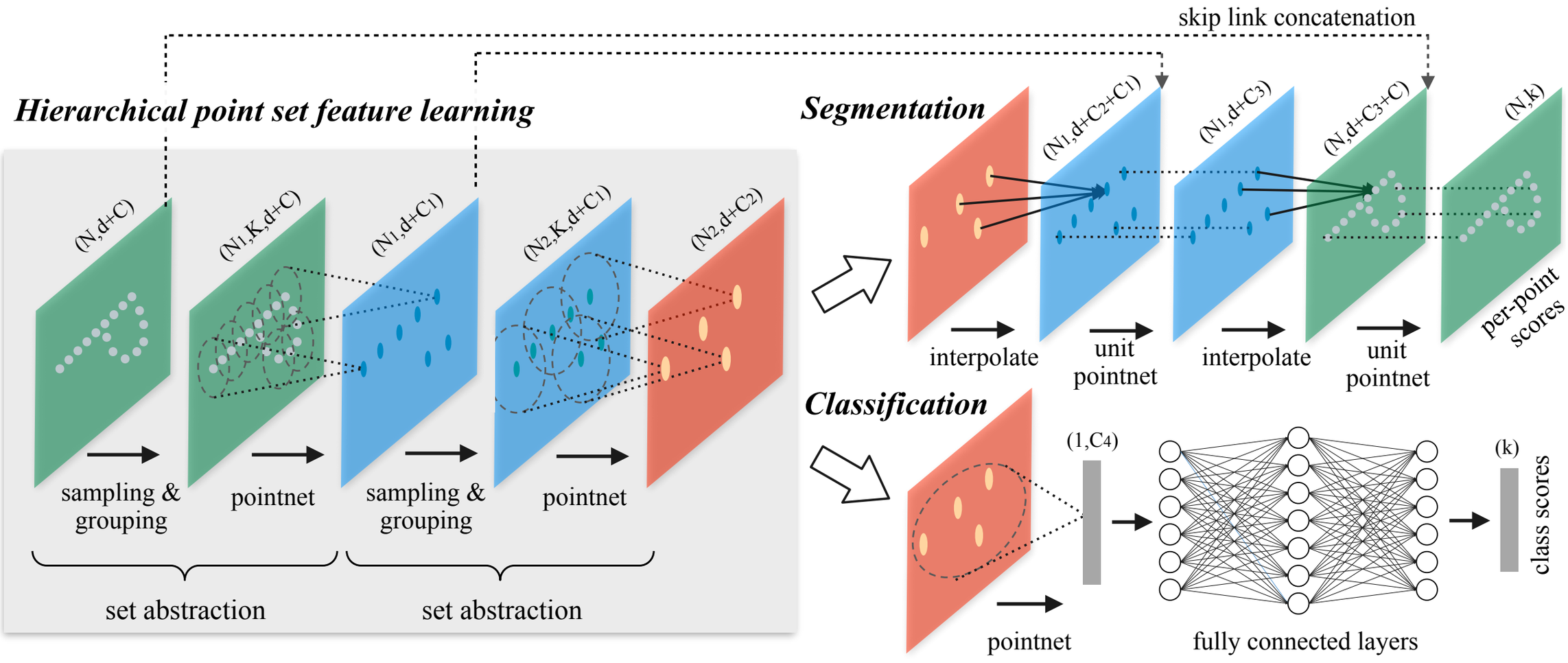
# PointNet++: Complete architecture




Slide: Charles R. Qi

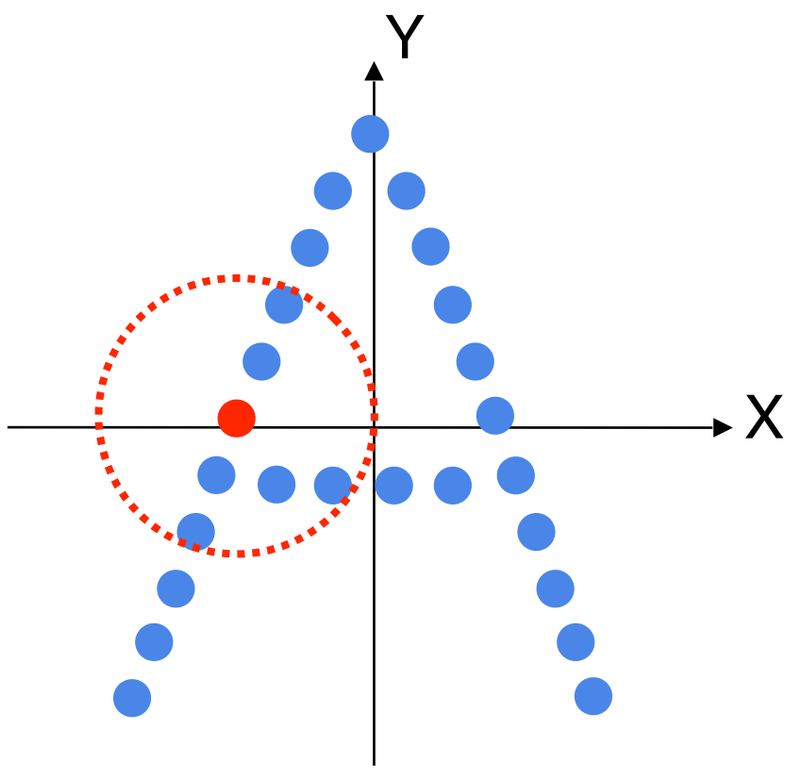
# PointNet++: Complete architecture



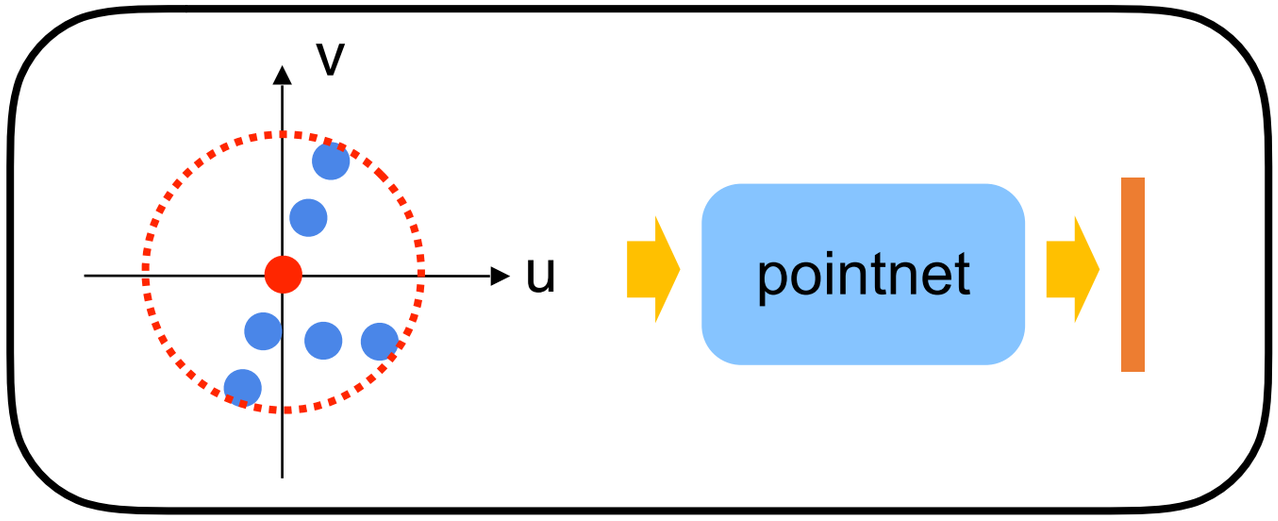

Slide: Charles R. Qi

# Dynamic Graph CNN for Learning on Point Clouds (DGCNN)

- Capture local geometric features of point clouds with EdgeConvolution (EdgeConv)



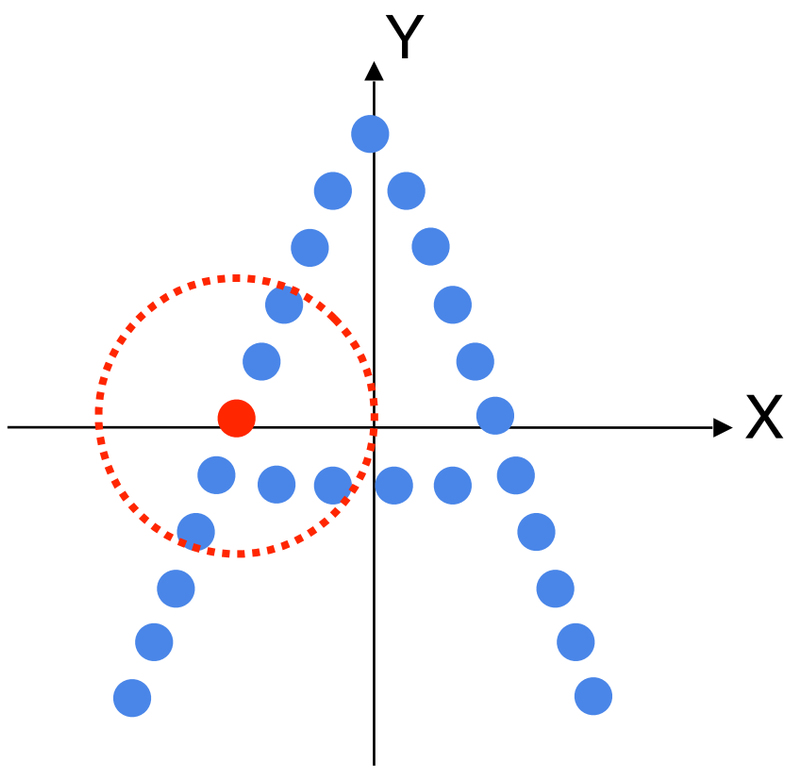

N points in (X,Y)



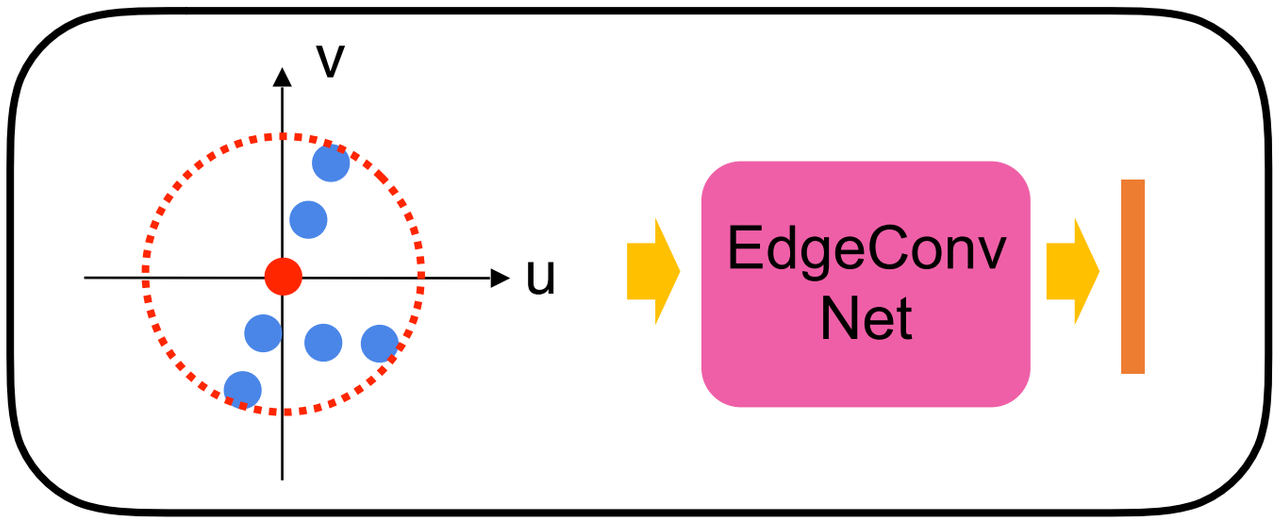

k points in local coordinates (u,v)

# Dynamic Graph CNN for Learning on Point Clouds (DGCNN)

- Capture local geometric features of point clouds with EdgeConvolution (EdgeConv)



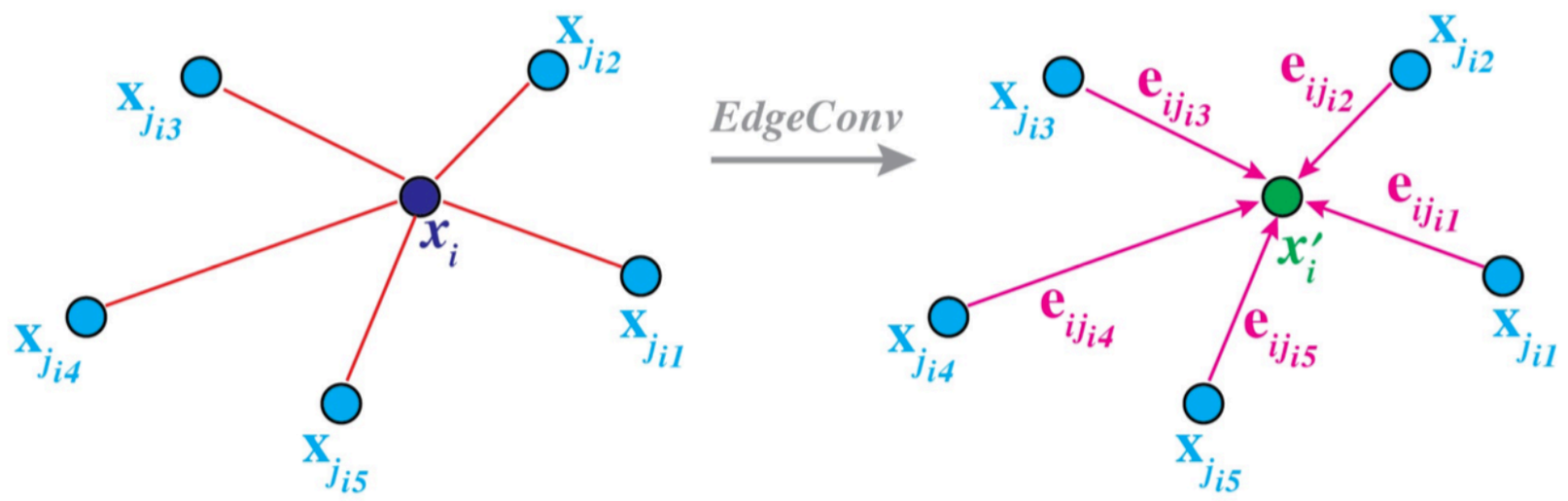

N points in (X,Y)




k points in local coordinates (u,v)

# Dynamic Graph CNN for Learning on Point Clouds (DGCNN)

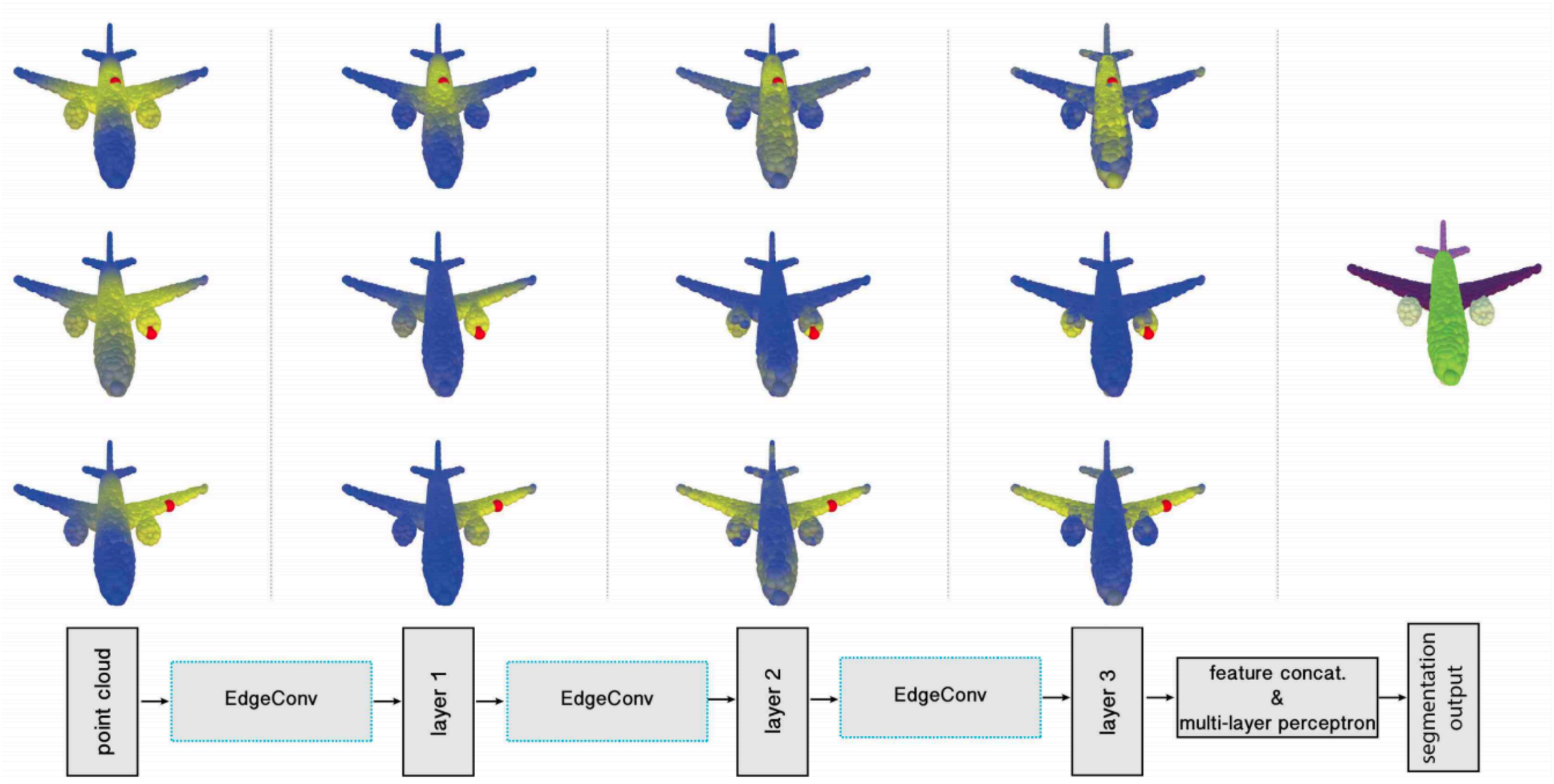
- Capture local geometric features of point clouds / features with EdgeConvolution (EdgeConv)




- **Edge convolution operator** is defined as

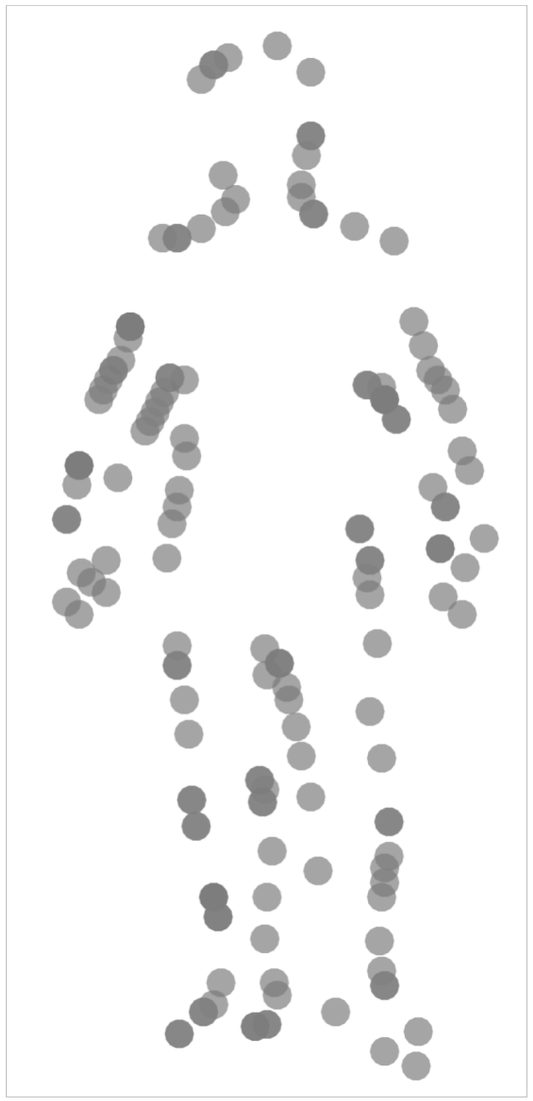
$$h_\theta(x_i, x_j) = h_\theta(x_i, x_j - x_i)$$

# Dynamic Graph CNN for Learning on Point Clouds (DGCNN)



# Point Convolutional Neural Networks by Extension Operators

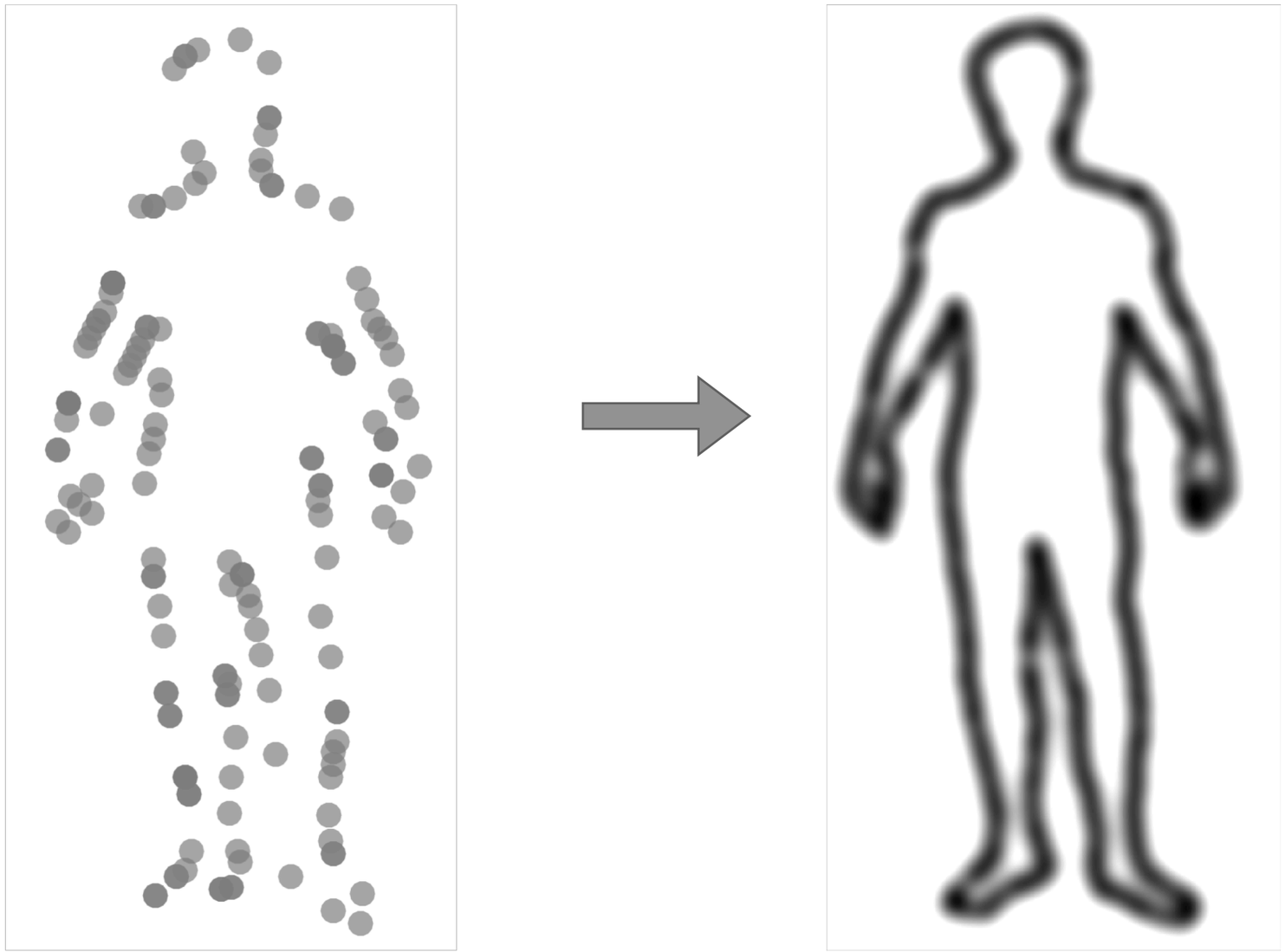
- A different approach for point cloud processing



Credit: Matan Atzmon et al.

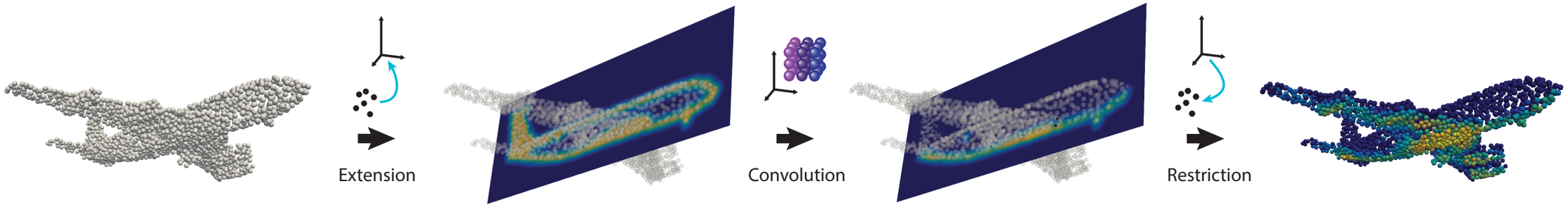
# Point Convolutional Neural Networks by Extension Operators

- A different approach for point cloud processing
  - extend the samples to a continuous function
  - utilize it to perform continuous convolution



Credit: Matan Atzmon et al.

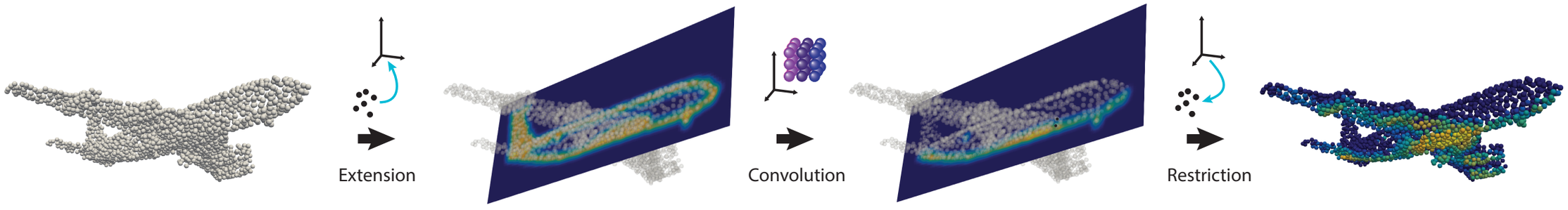
# Point Convolutional Neural Networks by Extension Operators




$$O_X = \mathcal{R}_X \circ O \circ \mathcal{E}_X$$

Credit: Matan Atzmon et al.

# Point Convolutional Neural Networks by Extension Operators




$$O_X = \mathcal{R}_X \circ O \circ \mathcal{E}_X$$

- Extension operator $\mathcal{E}_X$

$$\mathcal{E}_X[f](x) = \sum_i f_{ij} \ell_i(x),$$

$$\ell_i(x) = c\omega_i \Phi(|x - x_i|)$$

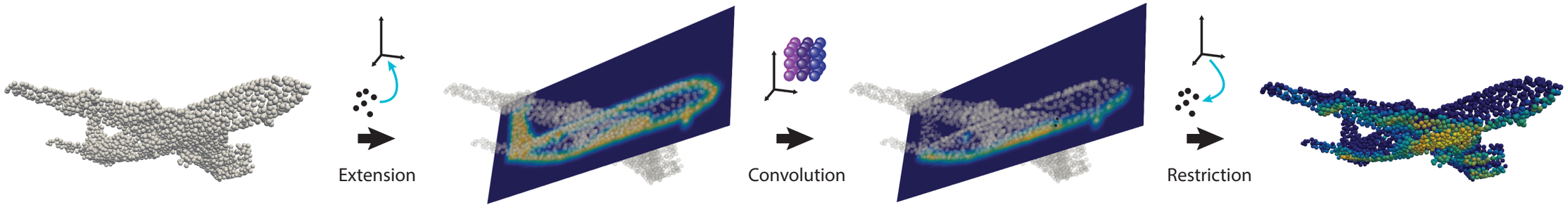
Gaussian RBF

Voronoi Diagram

* Weights inverse proportional to point density

Credit: Matan Atzmon et al.

# Point Convolutional Neural Networks by Extension Operators




$$O_X = \mathcal{R}_X \circ O \circ \mathcal{E}_X$$

- Continuous convolutional kernels

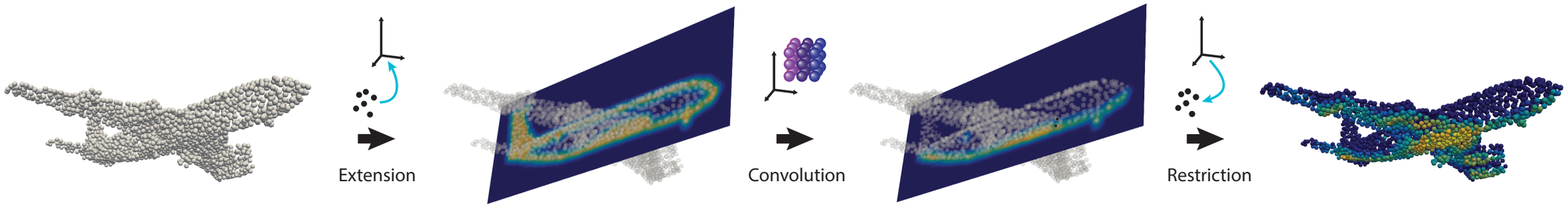
$$k(x) = \sum_{l=1}^{L} k_l \, \Phi(x - T_l)$$

Learnable parameters

Kernel layout
(can also be learnt)

Credit: Matan Atzmon et al.

# Point Convolutional Neural Networks by Extension Operators



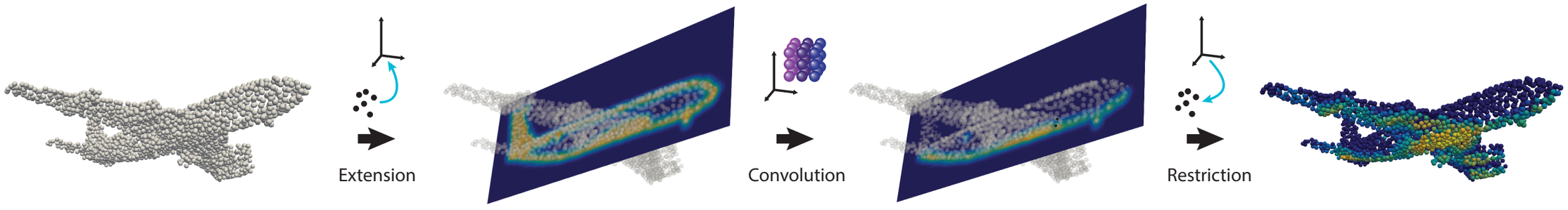

$$O_X = \mathcal{R}_X \circ O \circ \mathcal{E}_X$$

- Extension + convolution: closed form solution

$$\mathcal{E}_X * k(x) = \sum_{i,l} \alpha_i k_l \, \Phi(x - x_i - T_l)$$

Credit: Matan Atzmon et al.

# Point Convolutional Neural Networks by Extension Operators




$$O_X = \mathcal{R}_X \circ O \circ \mathcal{E}_X$$

- Extension + convolution: closed form solution

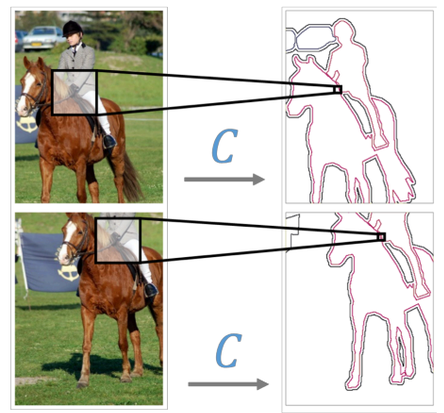
$$\mathcal{E}_X * k(x) = \sum_{i,l} \alpha_i k_l \, \Phi(x - x_i - T_l)$$

- Restriction by sampling

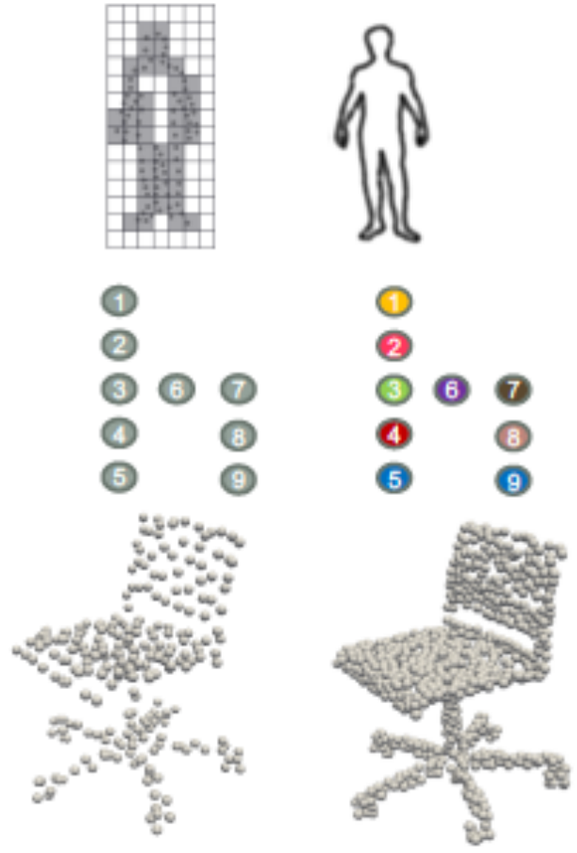
Credit: Matan Atzmon et al.

# Point Convolutional Neural Networks by Extension Operators

- Image convolution properties
  - Simplicity (Sparse+Linear)
  - **Translation equivariance** and locality

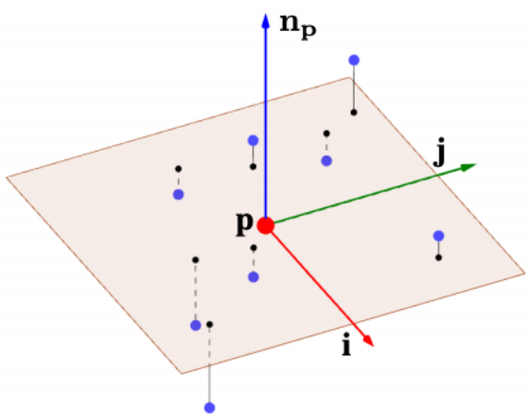


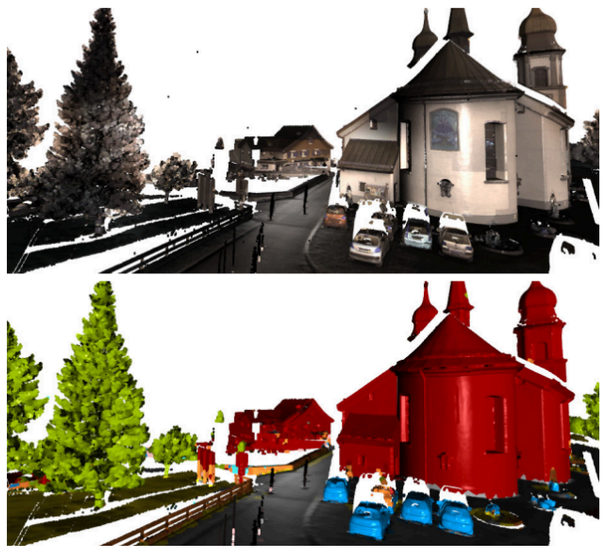

- Point convolution properties
  - Operate intrinsically
  - Order equivariance
  - **Robustness to sampling**



Credit: Matan Atzmon et al.

# Surface-based Convolution



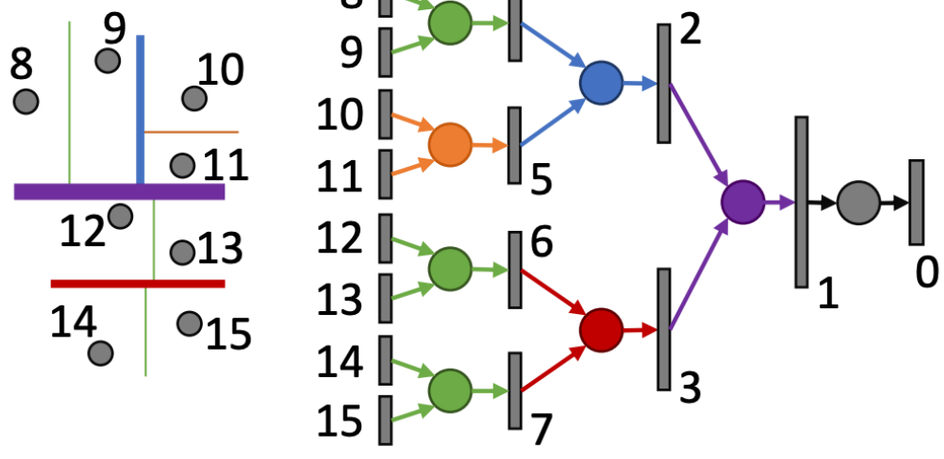



**Tangent Convolutions**
[Tatarchenko et al. 2018]

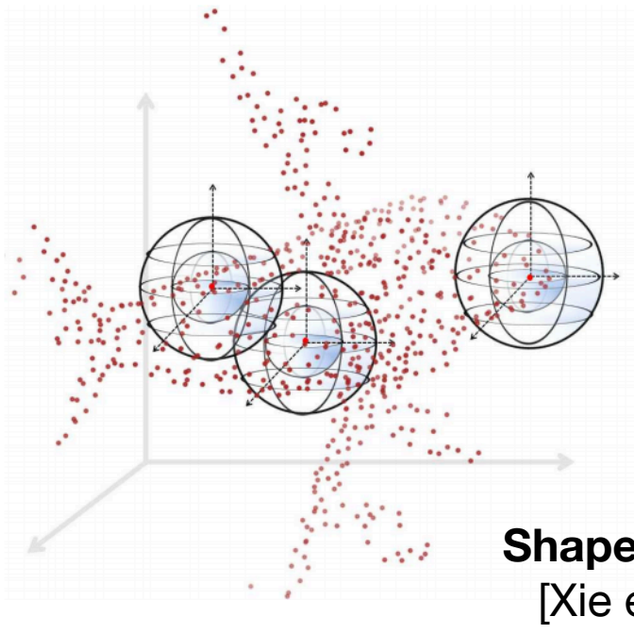

Surface Convolution

**SurfConv**
[Chu et al. 2018]

# Classic Representations + NN



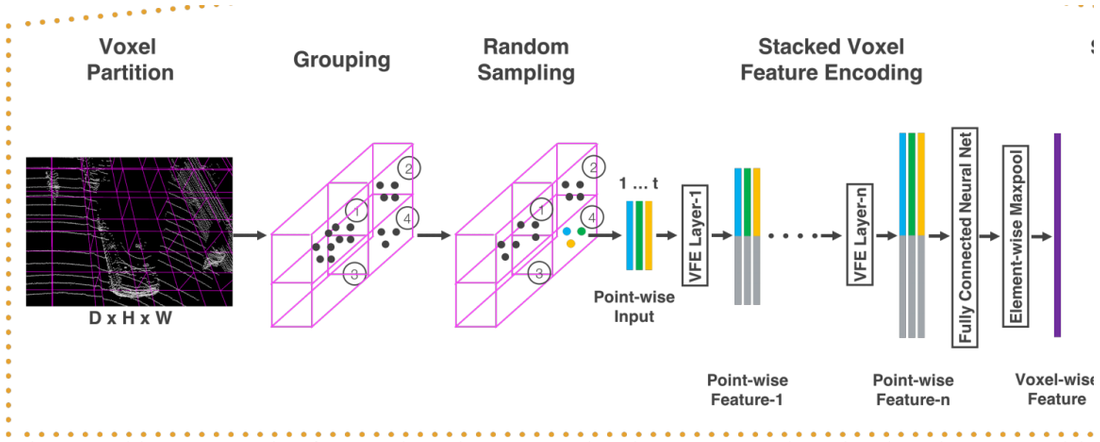

**Kd-Network**
[Klokov et al. 2017]



**ShapeContextNet**
[Xie et al. 2018]

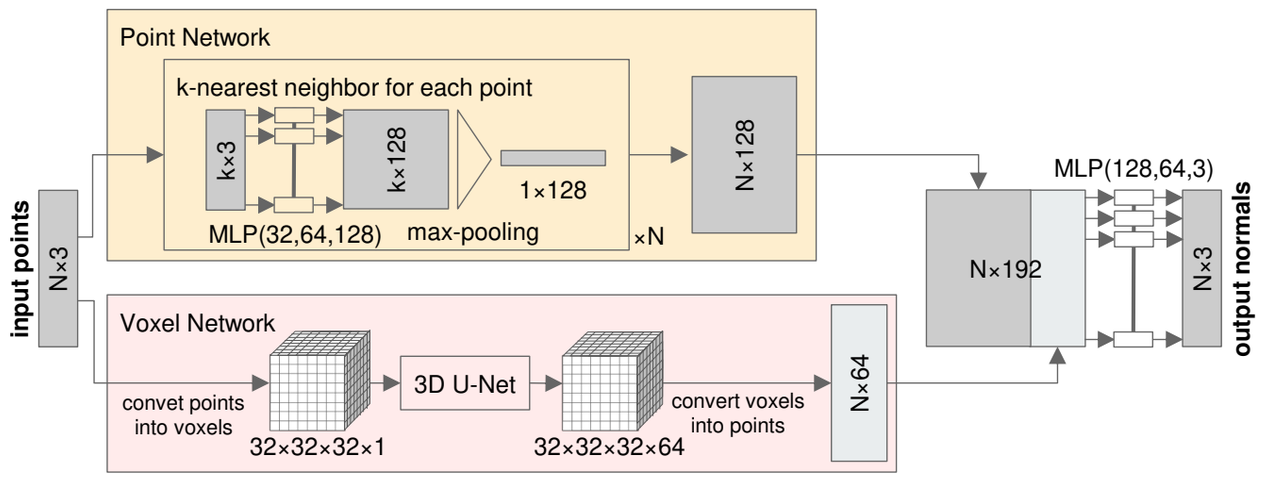
Slide: Charles R. Qi

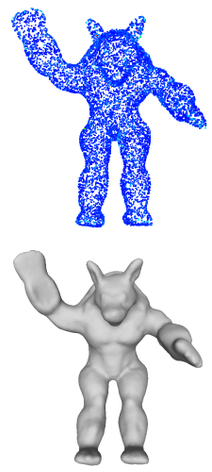
# Hybrid Networks: Grids + Points

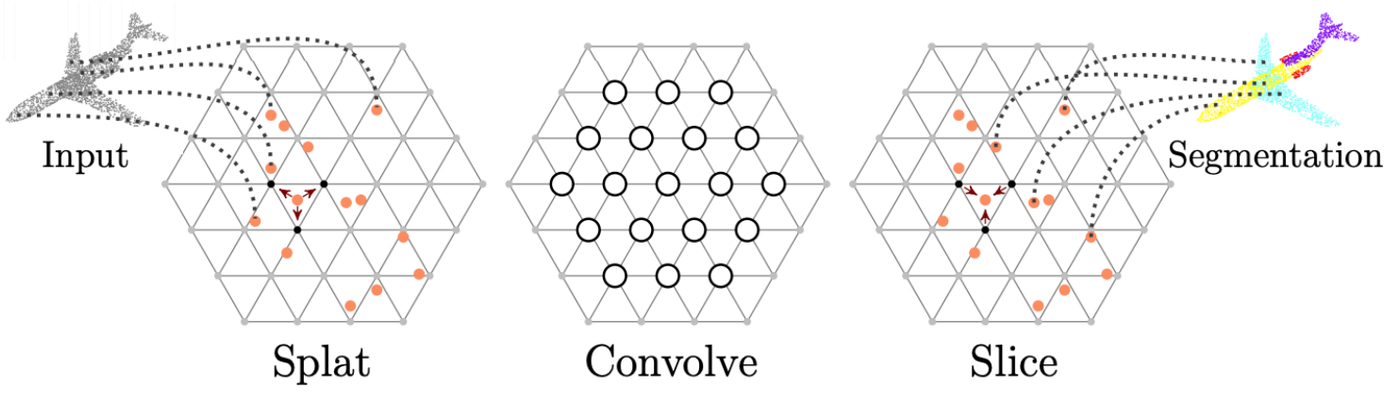



**VoxelNet**
[Zhou et al. 2018]

**Normal estimation**
[Hashimoti and Saito, CVPR 2019]



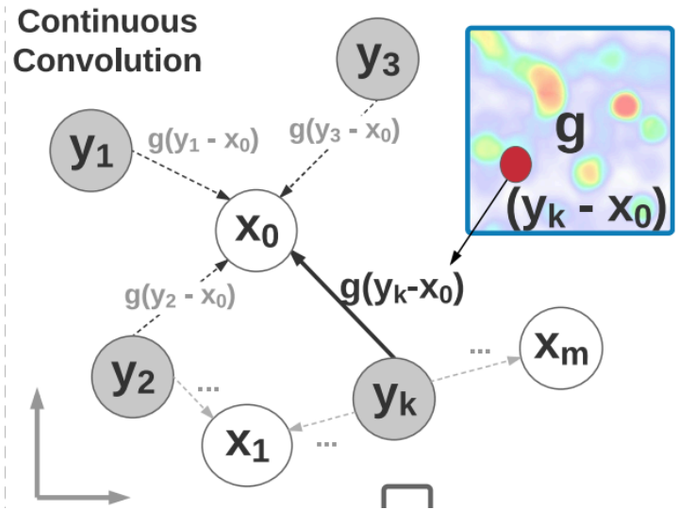





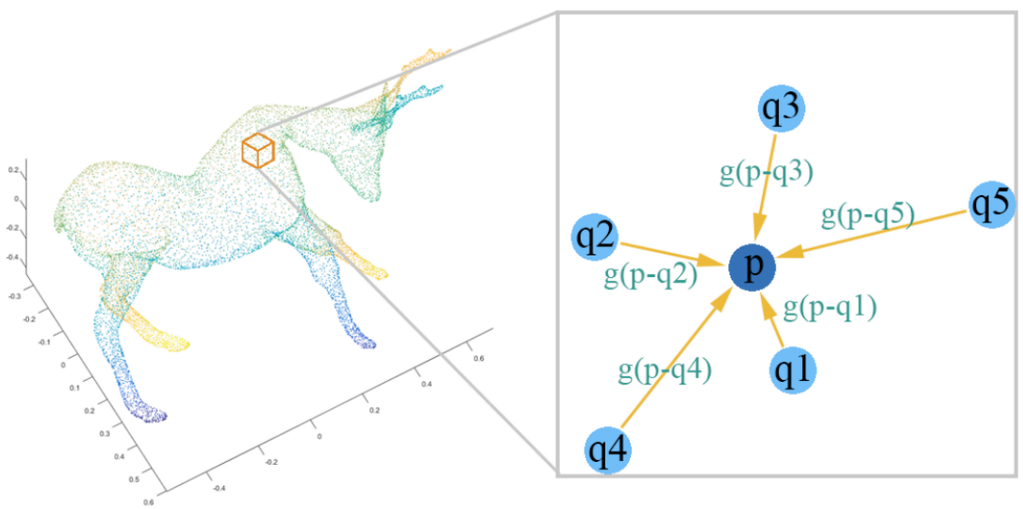

**SPLATNet**
[Su et al. 2018]

Credit: Charles R. Qi

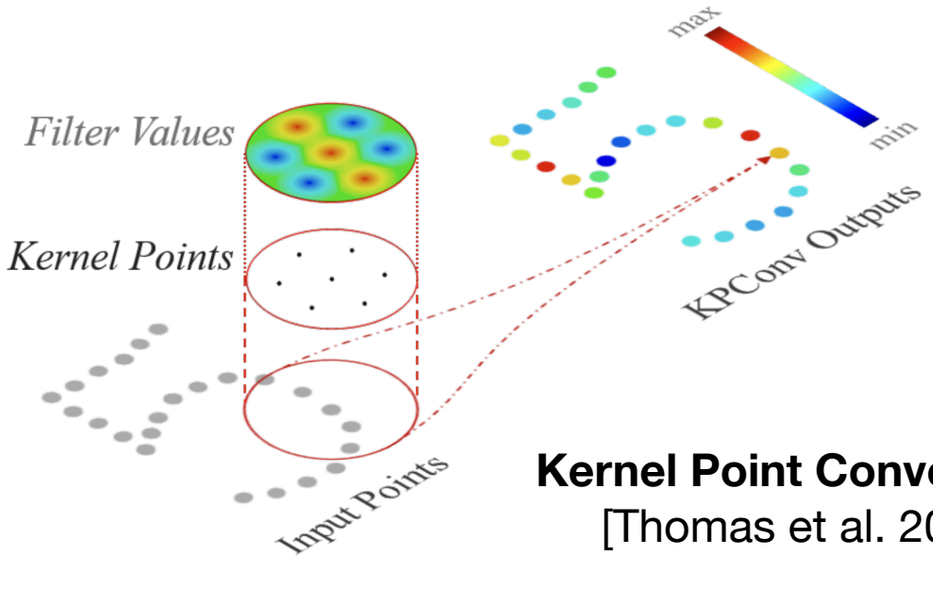
# Point Cloud Convolutions



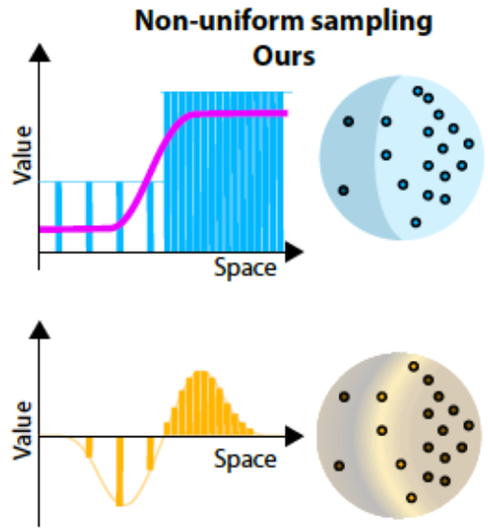

**Deep Parametric Continuous Convolution**
[Wang et al. 2018]




**SpiderCNN**
[Xu et al. 2018]



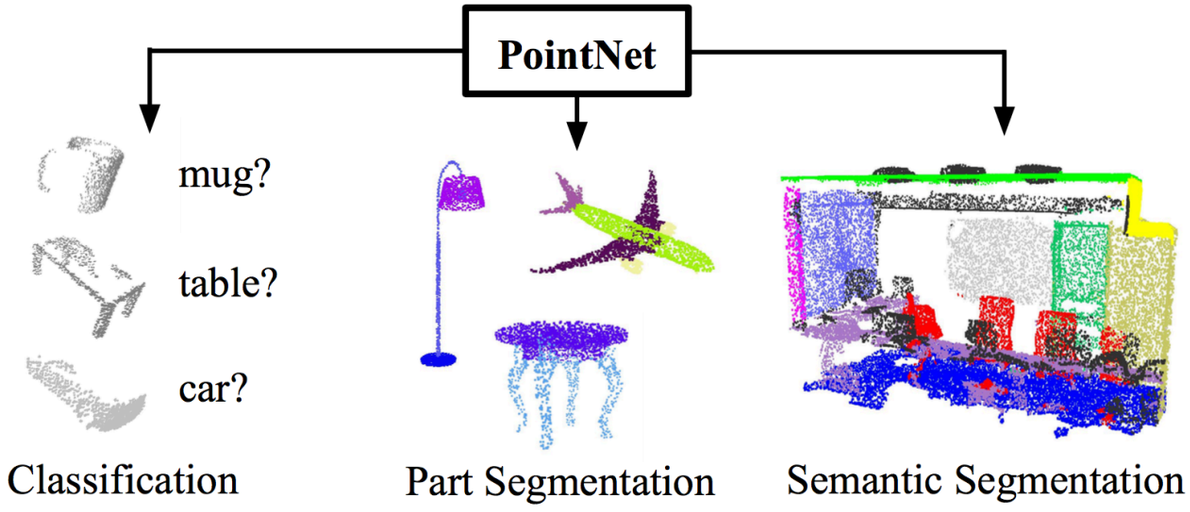

**Kernel Point Convolution**
[Thomas et al. 2019]

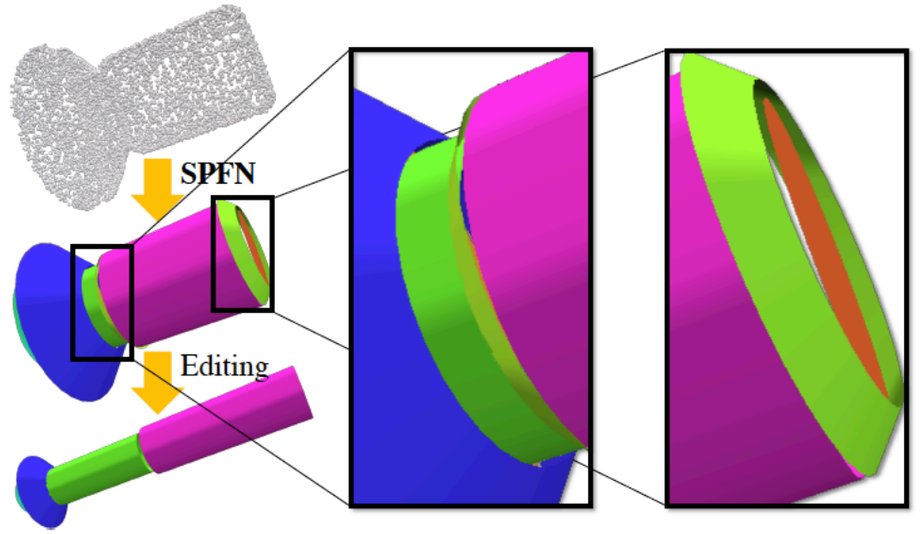



**Monte-Carlo Convolution**
[Hermosilla et al. 2018]

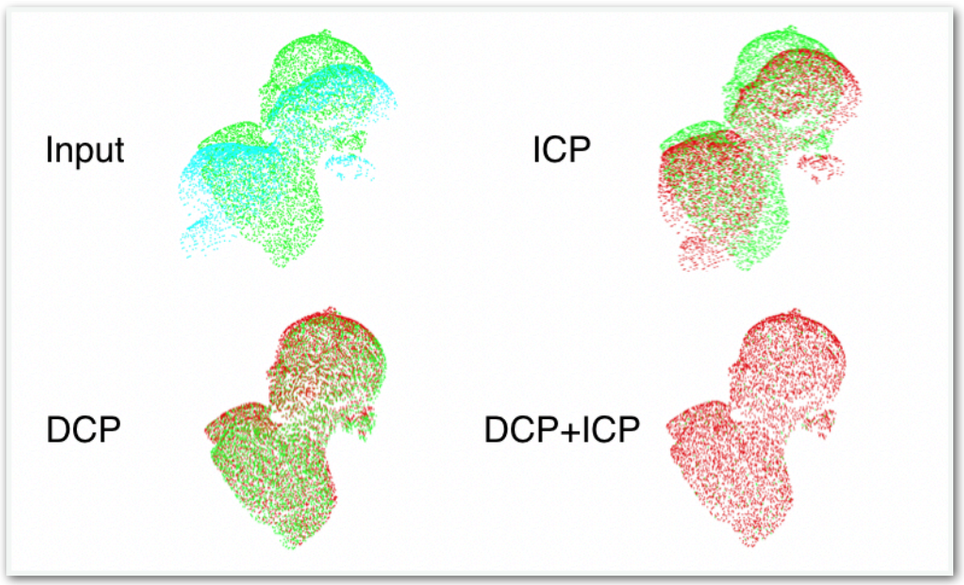
Slide: Charles R. Qi

# Point cloud deep learning: Applications

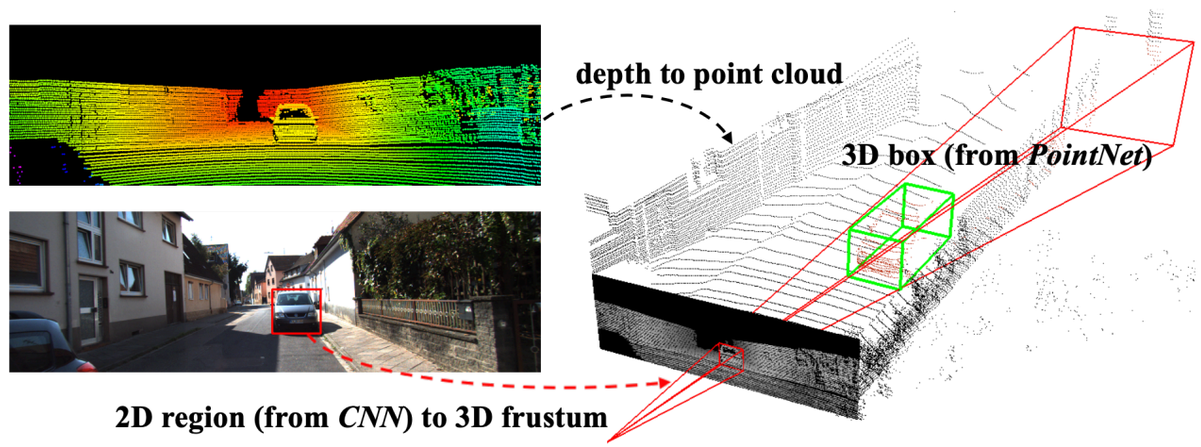



**Primitive fitting**



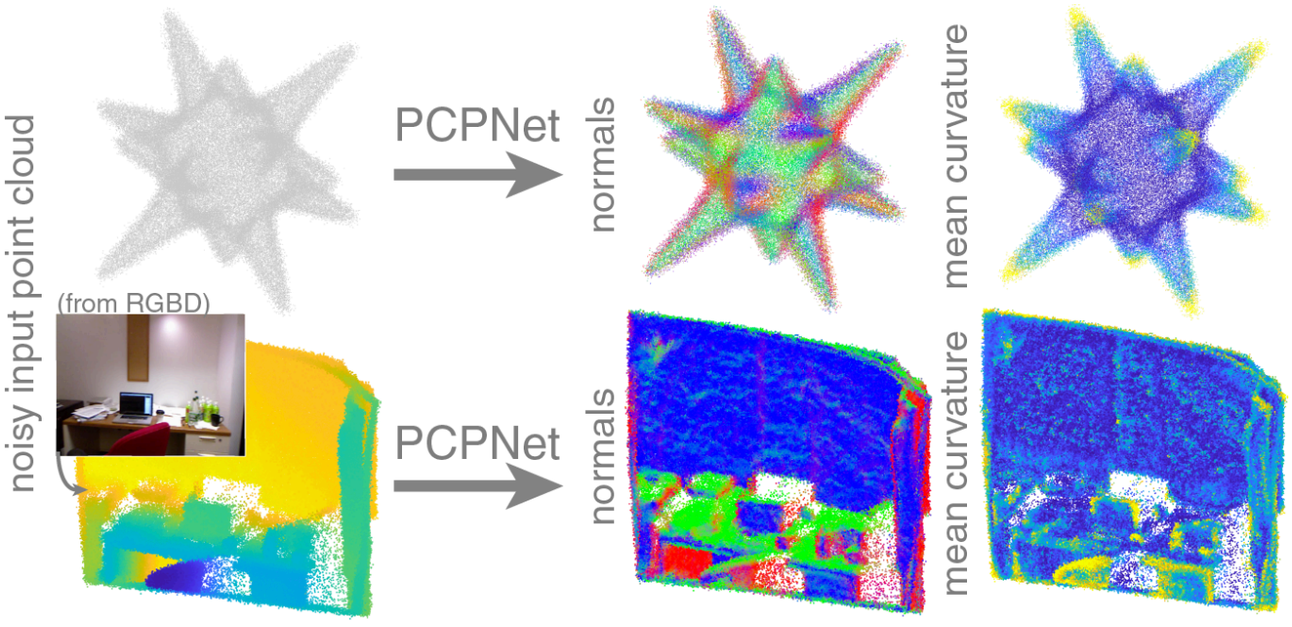

**Point cloud alignment**

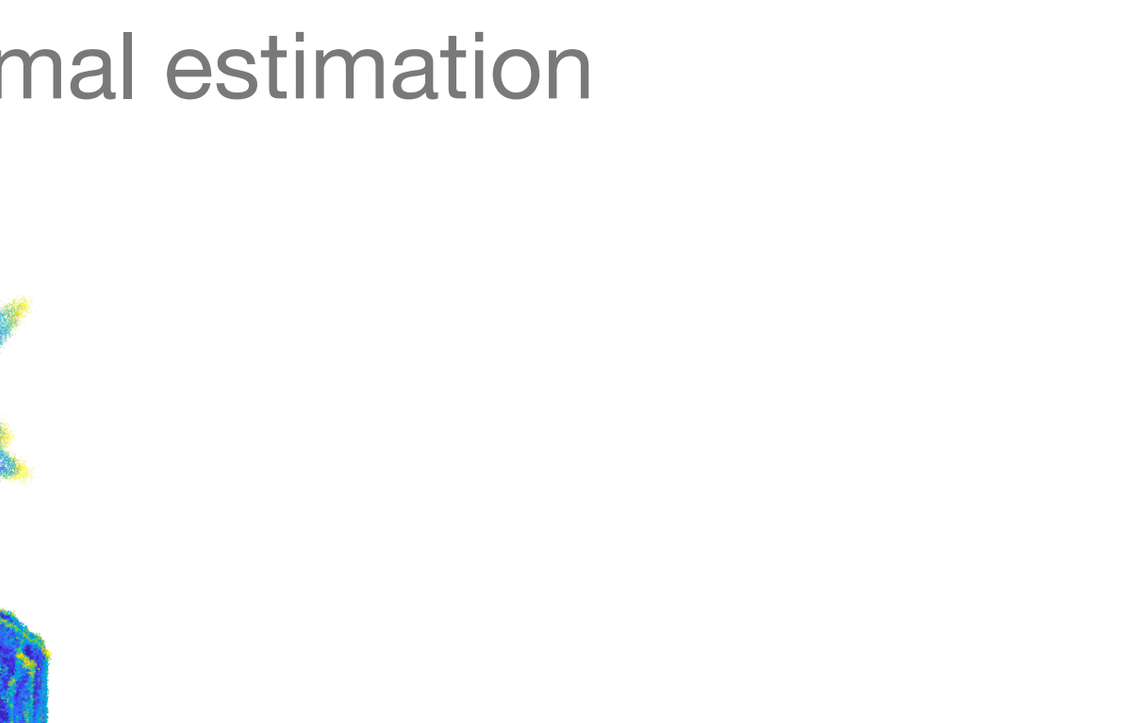



**Object detection in lidar scans**

Qi et al. [CVPR 2017], Li et al. [CVPR 2019], Wang and Solomon [arXiv], Qi et al. [CVPR 2018]

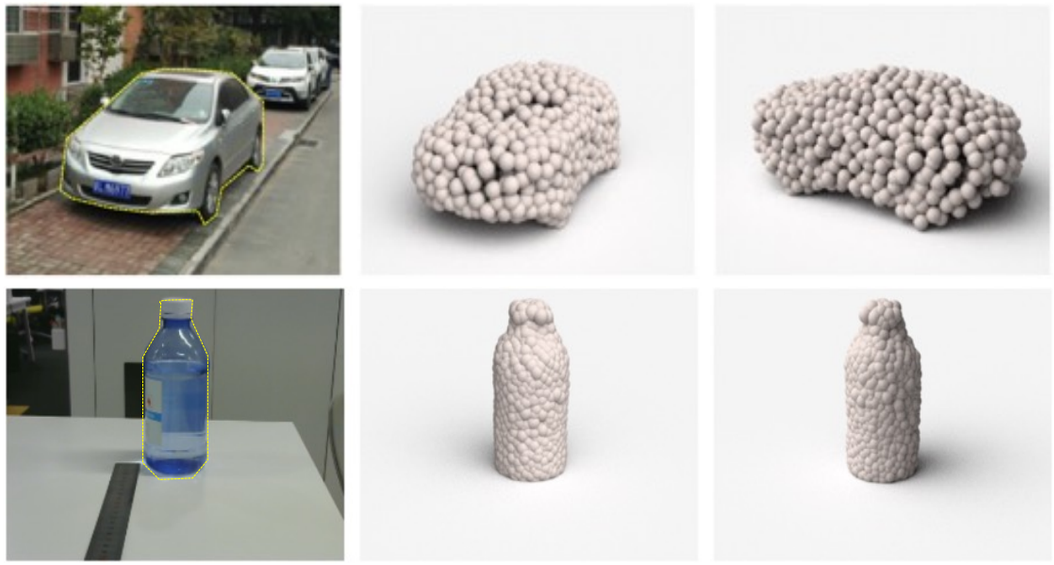
# Point cloud deep learning: Normal estimation



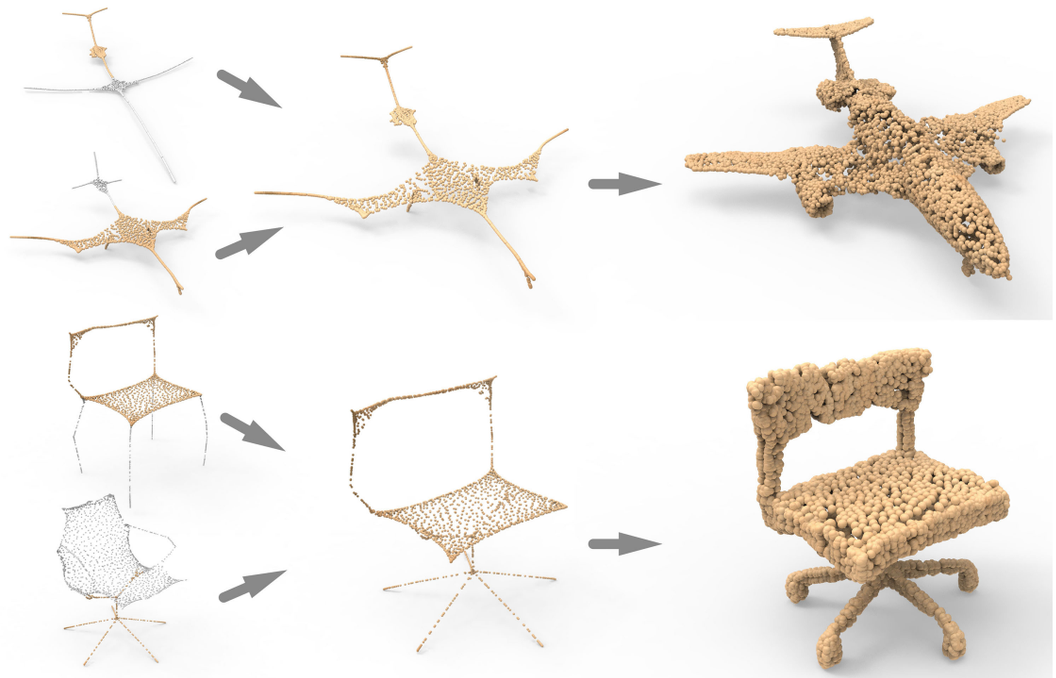



Guerrero et al. [Eurographics 2018], Boulch and Marlet [SGP 2016], Hashimoto and Saito [CVPR 2019]

# Point cloud deep learning applications: generative models

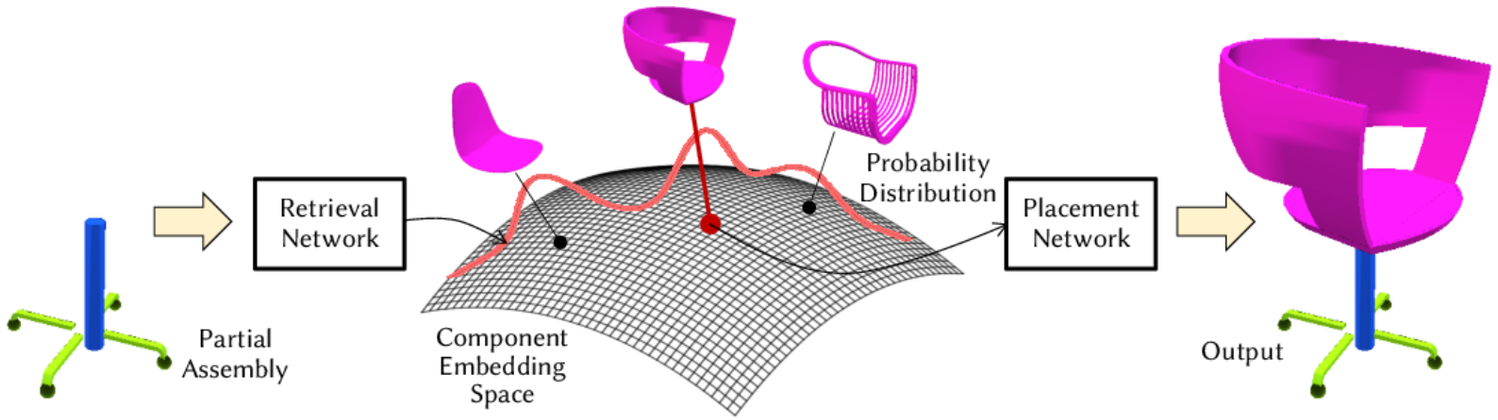


Input Reconstructed 3D point cloud

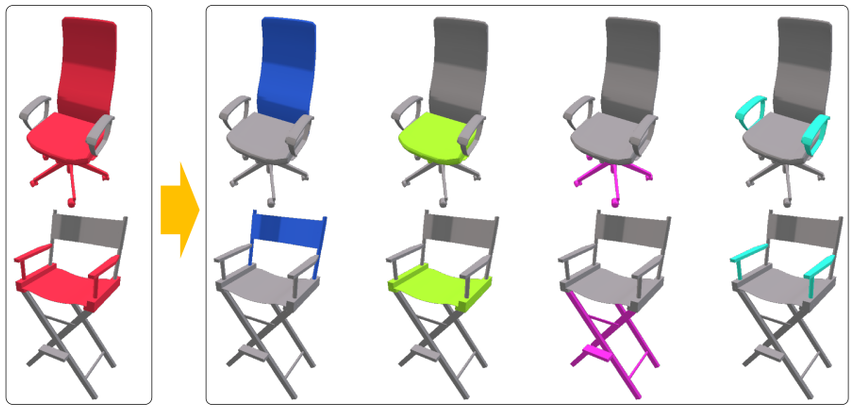



Fan et al. [CVPR 2017], Yin et al. [SIGGRAPH 2018], Achlioptas et al. [ICML 2018], Groueix et al. [CVPR 2018]

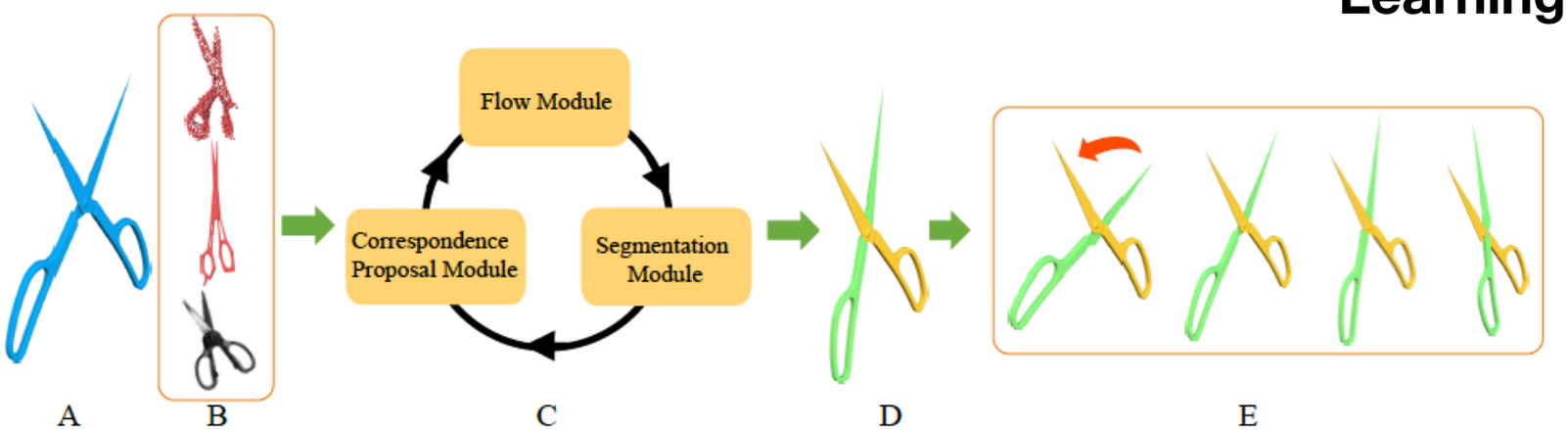
# Point cloud deep learning applications: proxy for mesh




**Shape completion**

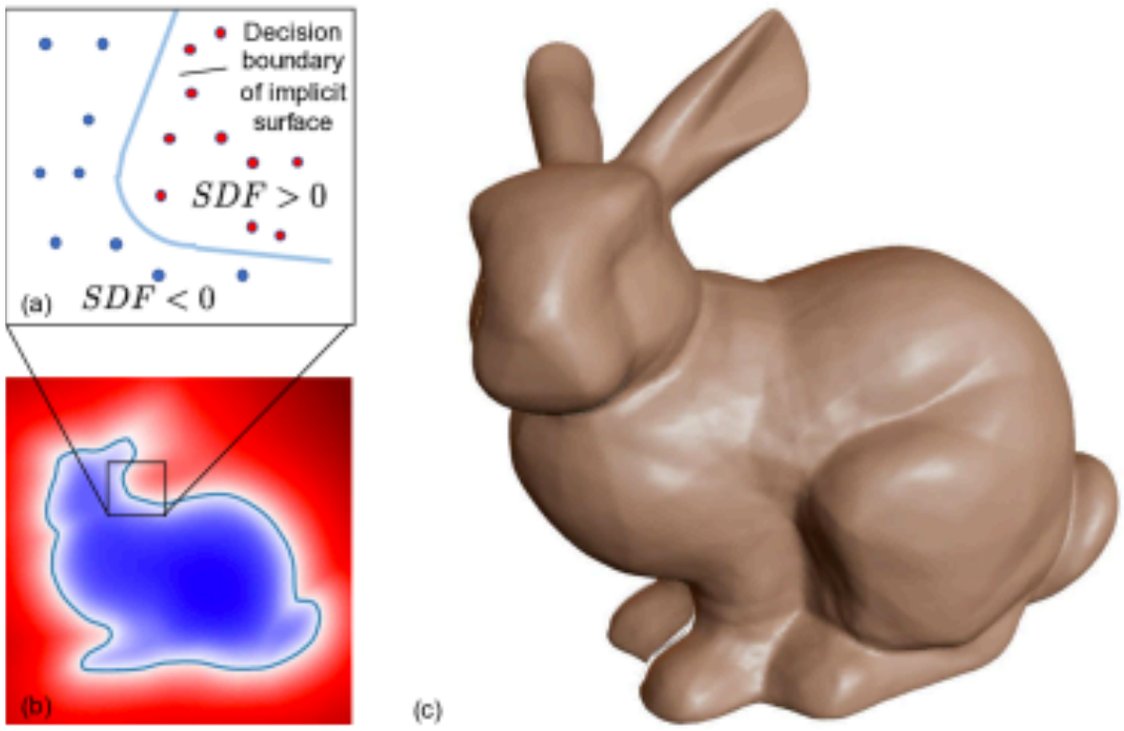


**Learning Consistent Semantic Structures**




**Part Induction from Articulated Object Pairs**

Sung et al. [SIGGRAPH Asia 2017], Li et al. [SIGGRAPH Asia 2018], Sung et al. [NeurIPS 2018]

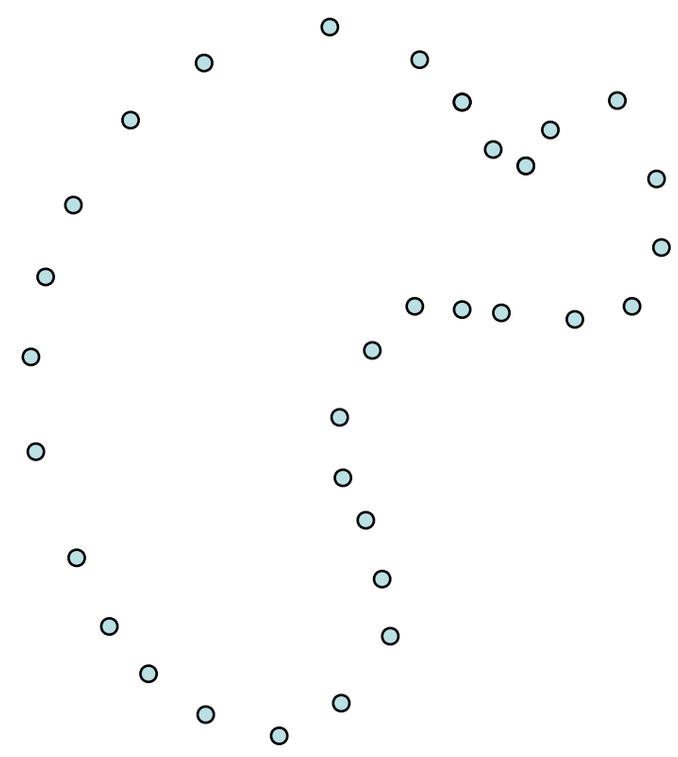



# Deep Learning Implicit Representations

Image credit: J. Park

# Implicit Function Representations

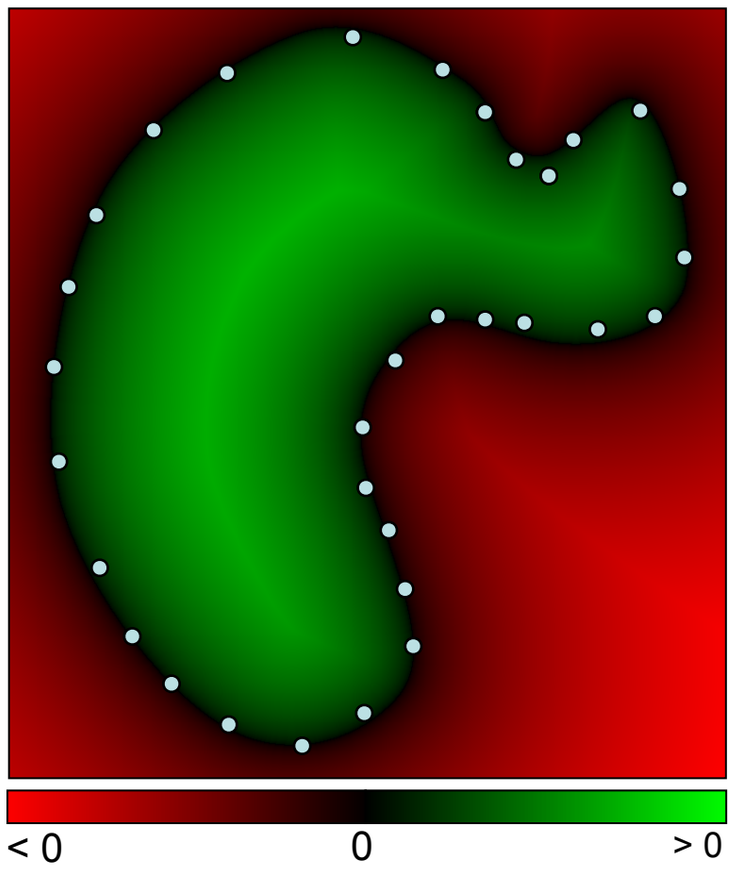
- Define a function $f : R^3 \rightarrow R$ with value < 0 outside the shape and > 0 inside



Credit: O. Diamanti, CS468

# Implicit Function Representation

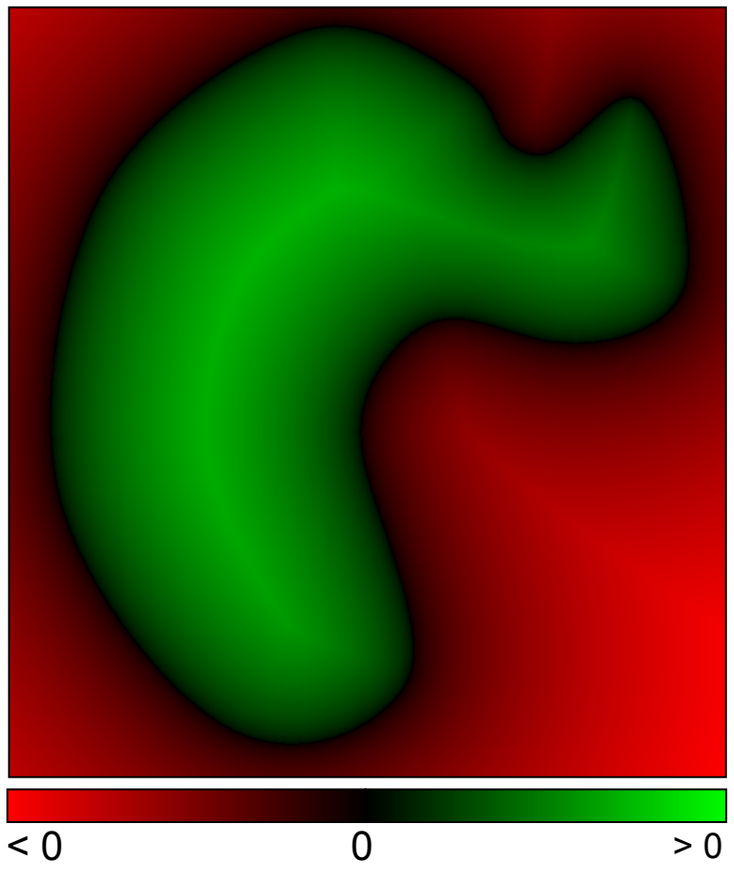
- Define a function $f : R^3 \rightarrow R$ with value < 0 outside the shape and > 0 inside




Credit: O. Diamanti, CS468

# Implicit Function Representation

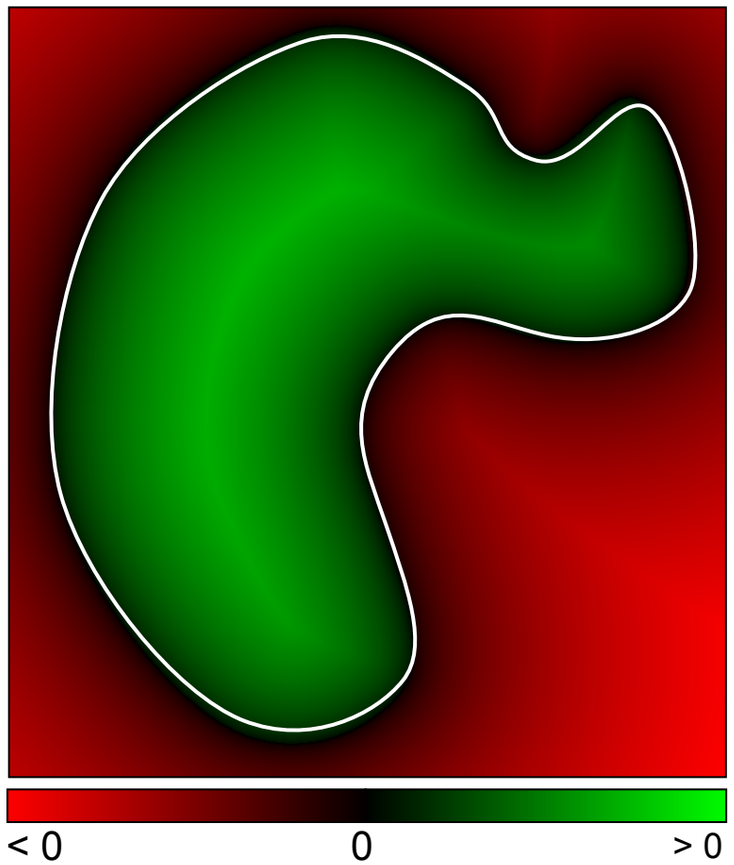
- Define a function $f : R^3 \rightarrow R$ with value < 0 outside the shape and > 0 inside




Credit: O. Diamanti, CS468

# Implicit Function Representation

- Define a function $f : R^3 \rightarrow R$ with value < 0 outside the shape and > 0 inside
- Extract the zero-set $\{x : f(x) = 0\}$




Credit: O. Diamanti, CS468

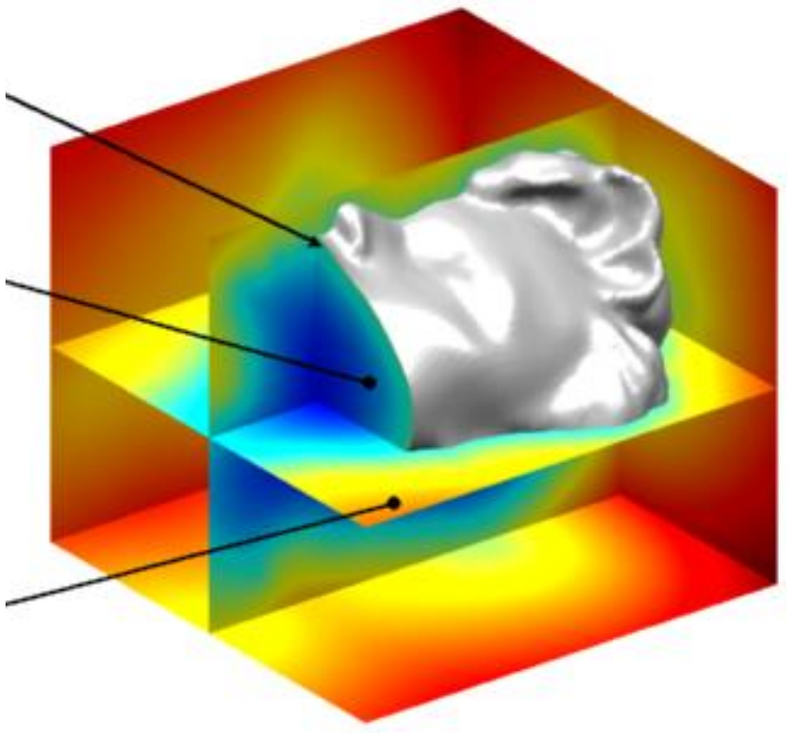
# Extract the surface

- Given an implicit representation, create a triangle mesh that approximates the surface: Marching cubes

$F(\mathbf{x}) = 0$ → surface

$F(\mathbf{x}) < 0$ → inside

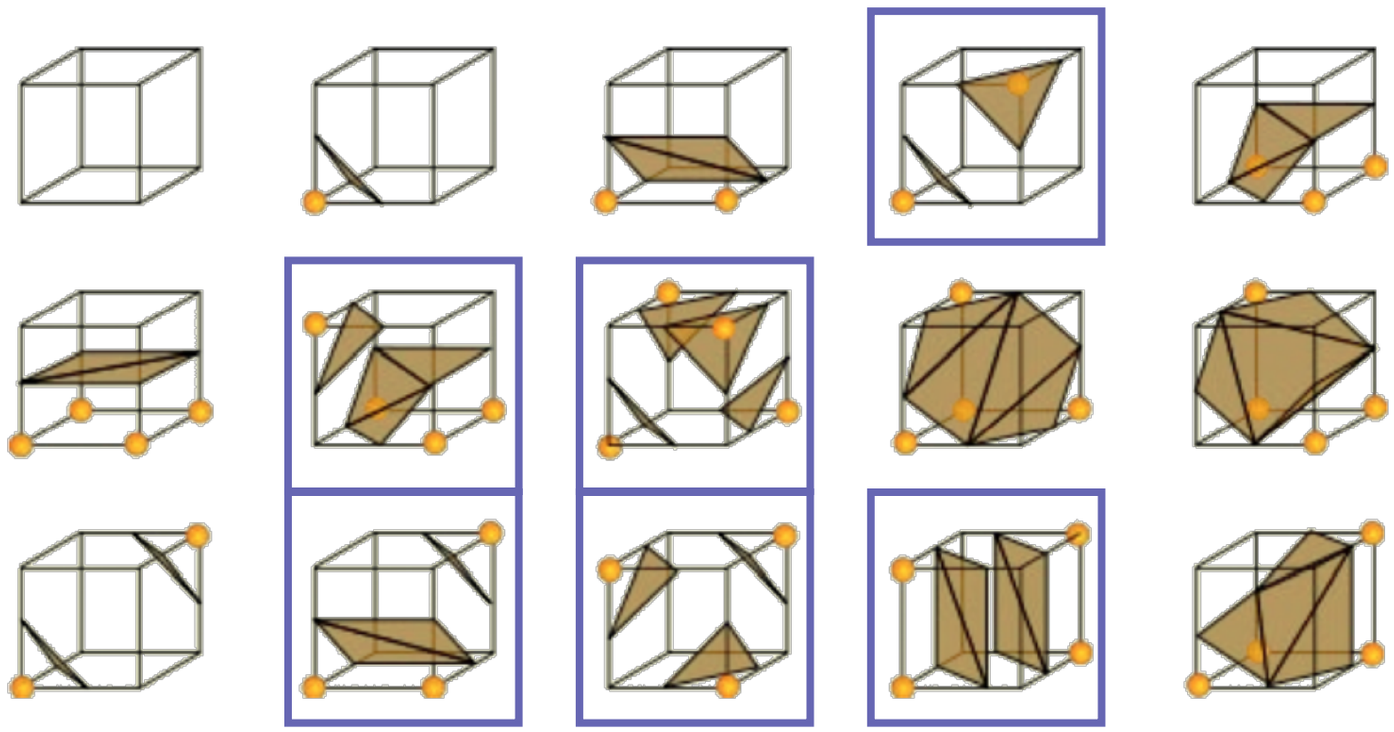
$F(\mathbf{x}) > 0$ → outside



Credit: O. Diamanti, CS468

# Extract the surface: Marching Cubes

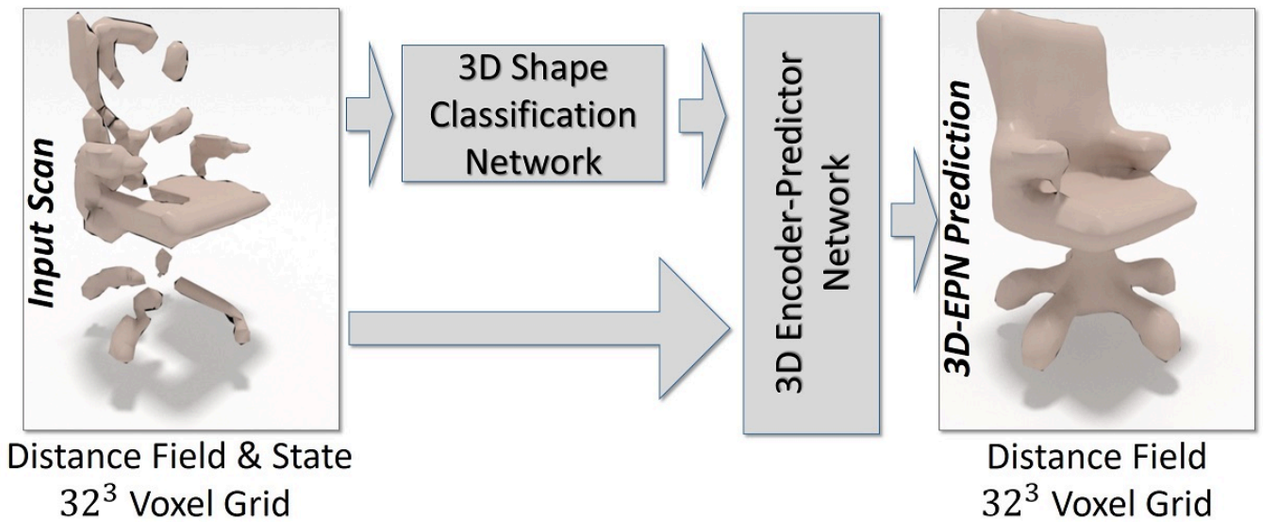
- 256 different cases - 15 after symmetries, 6 ambiguous cases
- More subsampling rules → 33 unique cases



the 15 cases

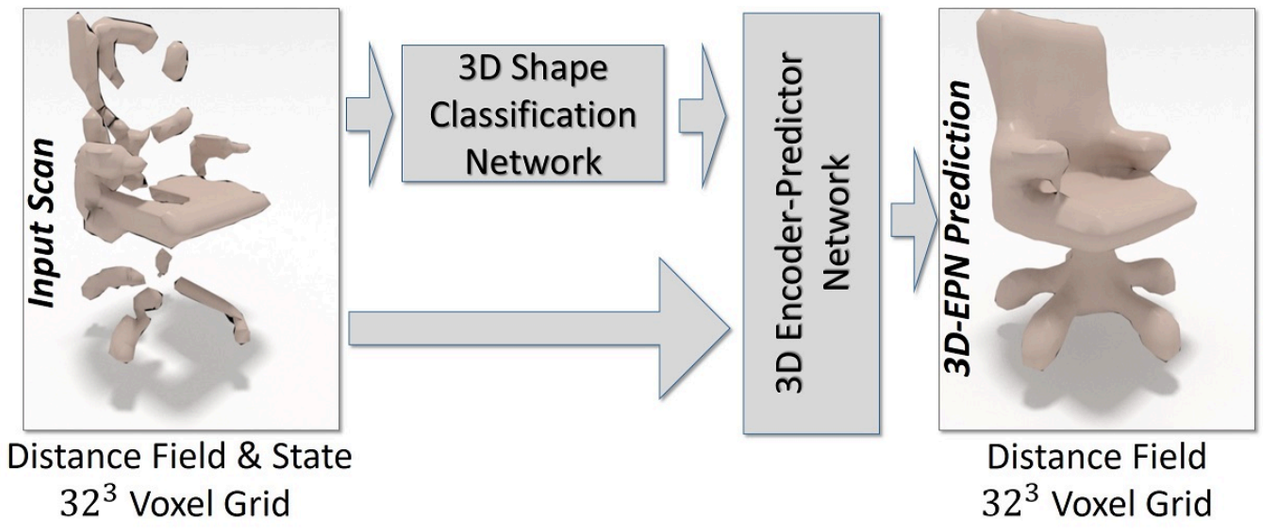
Credit: O. Diamanti, CS468

# Shape Completion using 3D-Encoder-Predictor




Dai et al. [CVPR 2017]

# Shape Completion using 3D-Encoder-Predictor



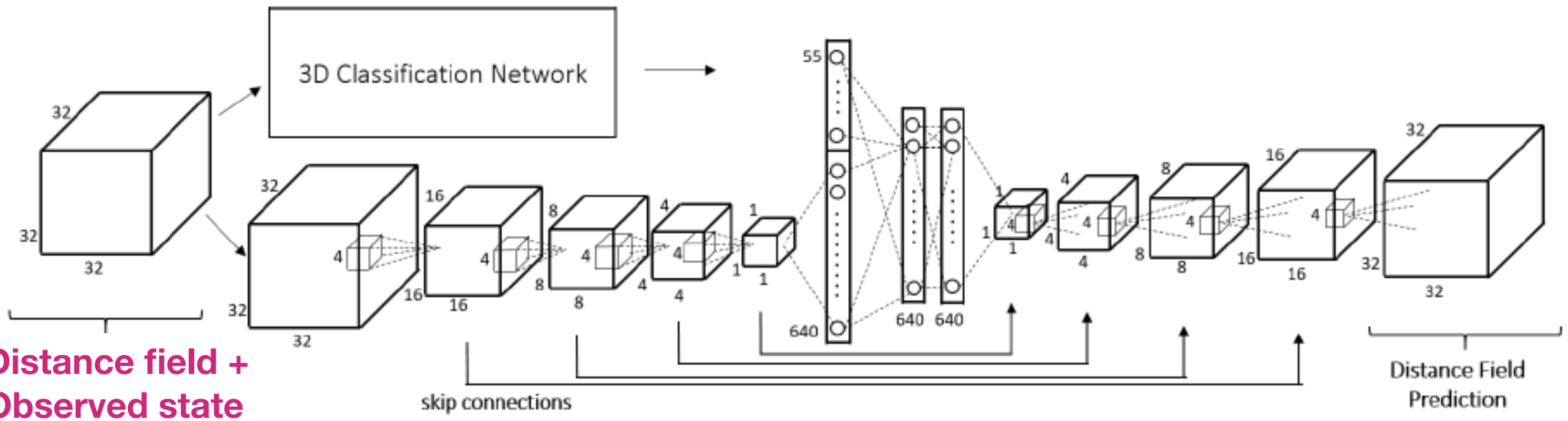
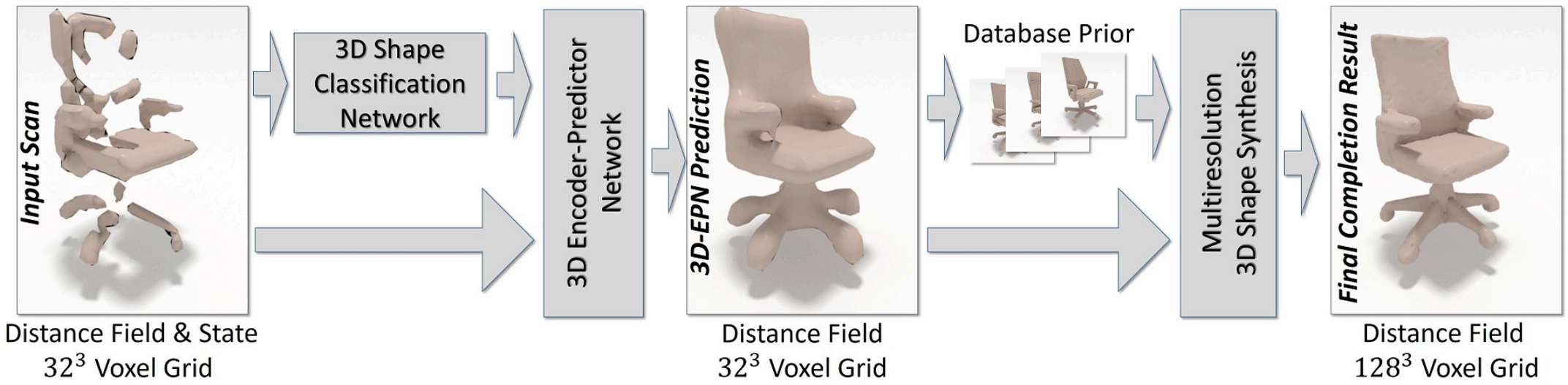




Figure 1: Network architecture of our 3D Encoder-Predictor Network.

Dai et al. [CVPR 2017]

# Shape Completion using 3D-Encoder-Predictor



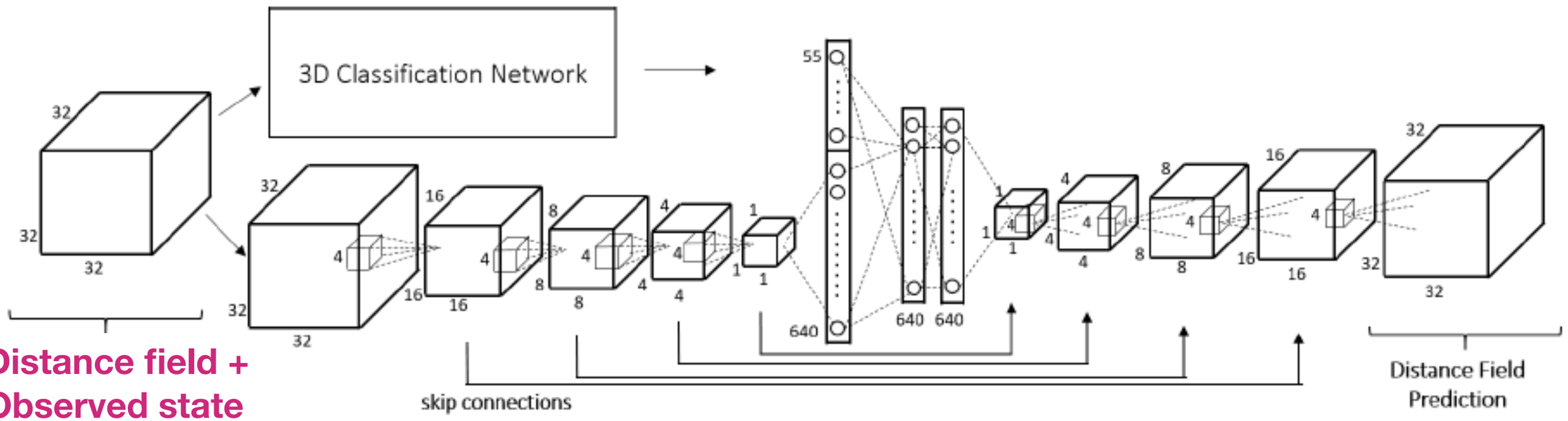
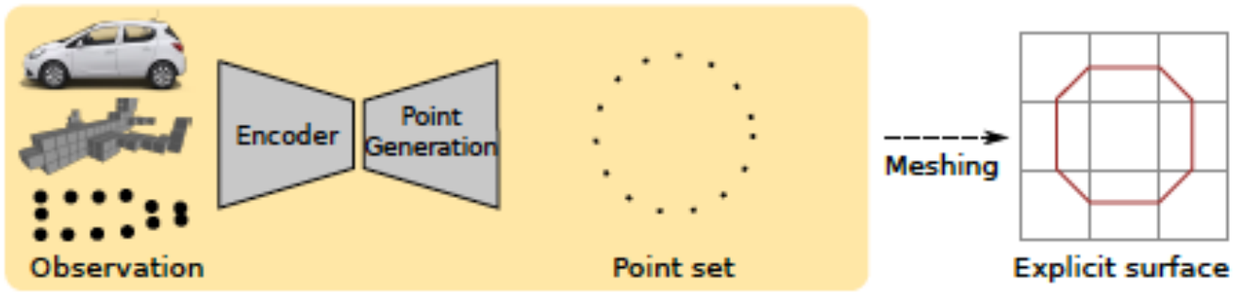




Figure 1: Network architecture of our 3D Encoder-Predictor Network.

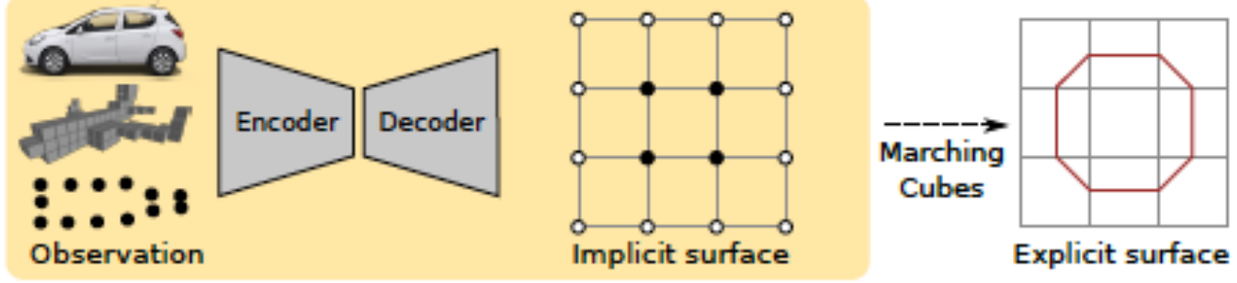
Dai et al. [CVPR 2017]

# Deep Marching Cubes

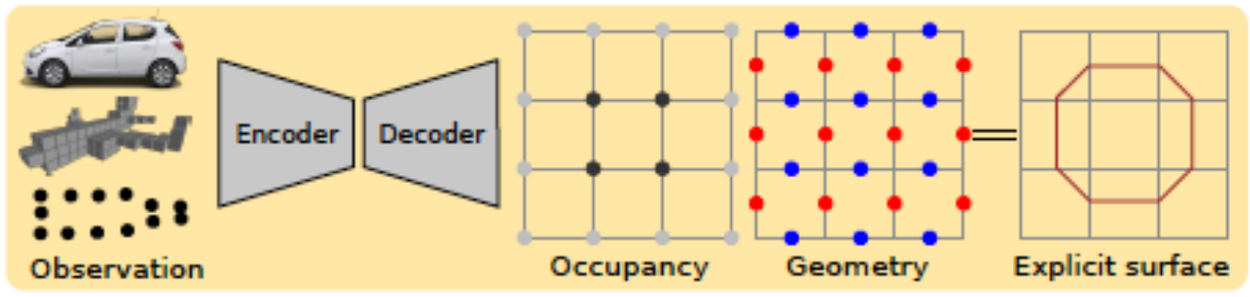
- Implicit surface prediction




(a) **Sparse Point Prediction** (e.g., [12])




(b) **Implicit Surface Prediction** (e.g., [35, 45])

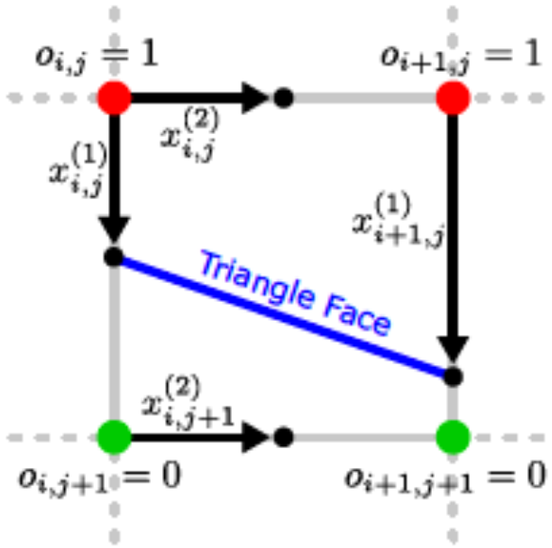



(c) **Explicit Surface Prediction** (ours)

Liao et al. [CVPR 2018]

# Deep Marching Cubes

- **Proposed alternative differentiable Marching Cubes**

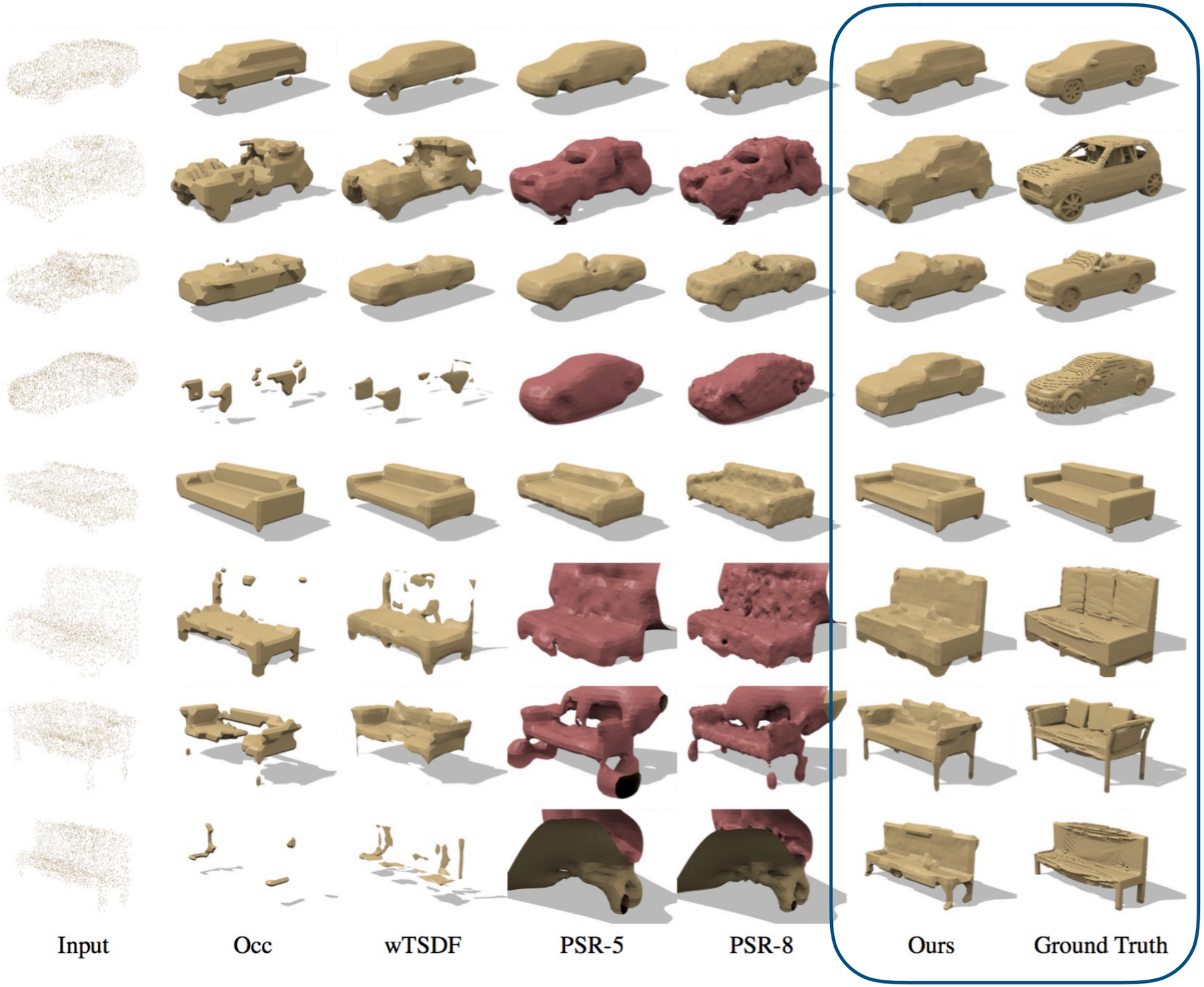



(b) Differentiable MC

- Per-vertex, occupancy probabilities **O** instead of signed distance
- Vertex displacements **X** to specify triangle vertices
- Defined differentiable distribution over meshes, used for back-propagation

Liao et al. [CVPR 2018]

# Deep Marching Cubes




Liao et al. [CVPR 2018]

# Learning decision boundary of occupancy classifier [CVPR 2019]

- “Occupancy Networks: Learning 3D Reconstruction in Function Space”, by Mescheder et al.
- “Learning Implicit Fields for Generative Shape Modeling”, by Chen and Zhang
- “DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation”, by Park et al.

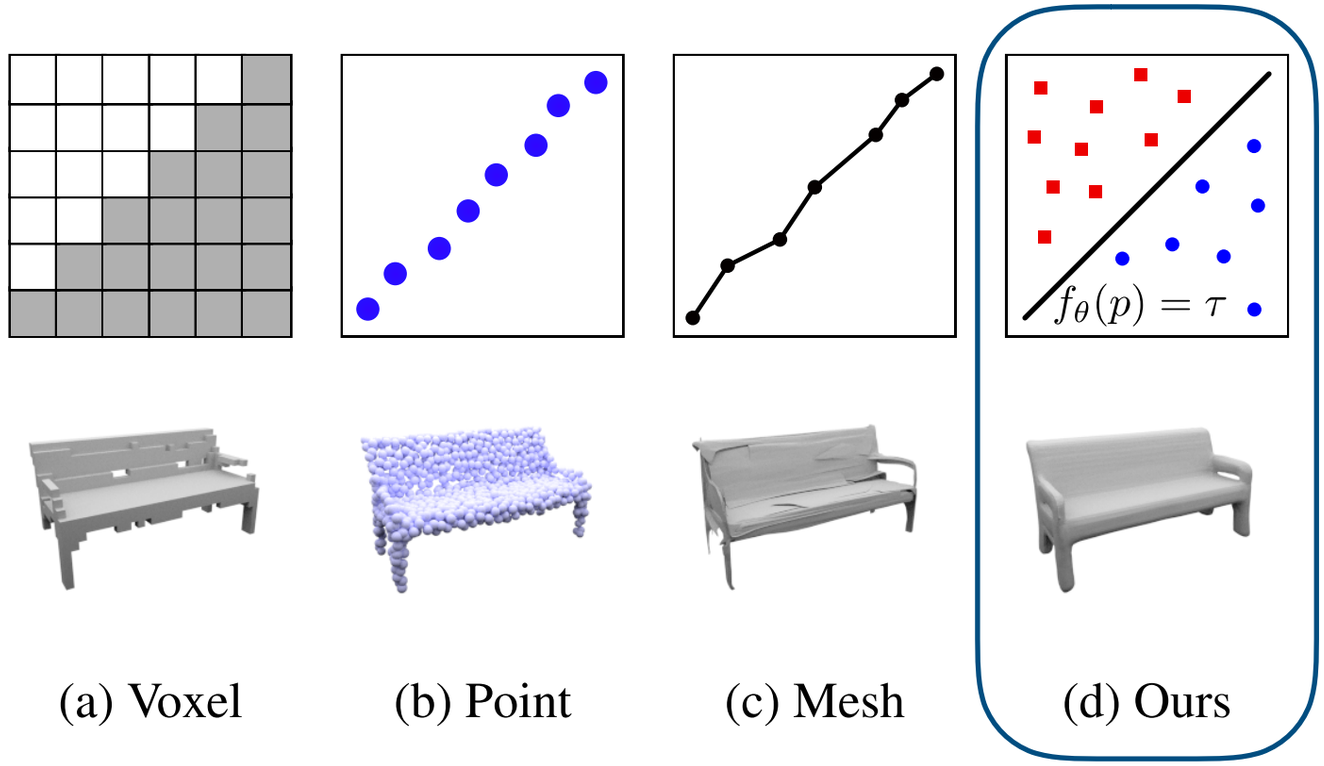
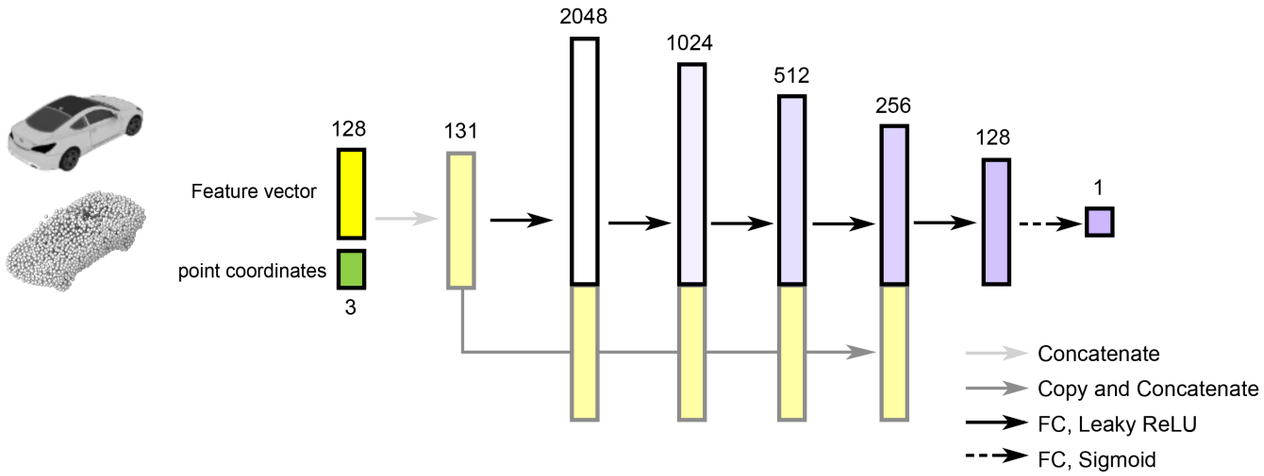


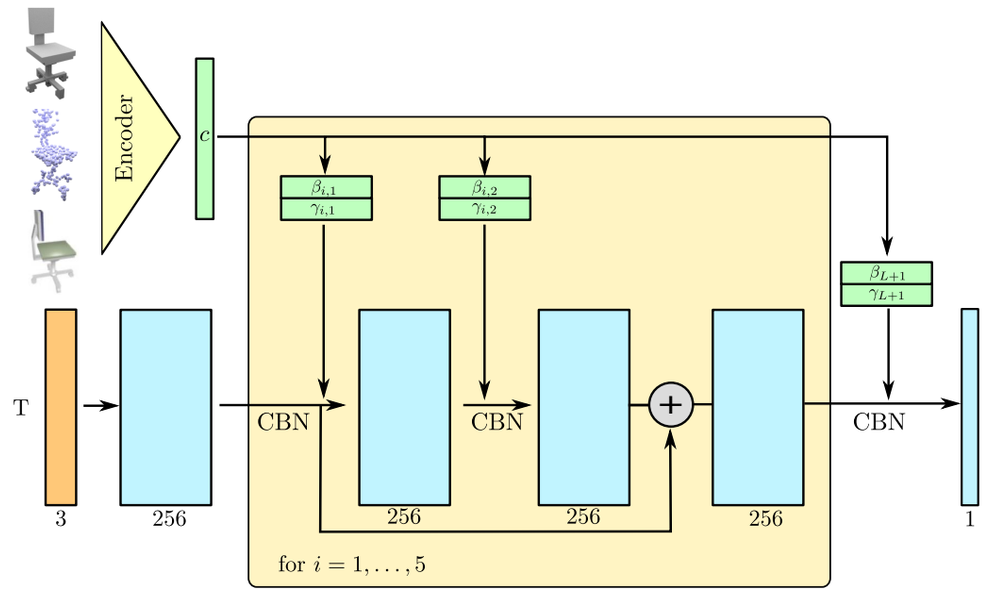

Image credit: L. Mescheder

# Learning decision boundary of occupancy classifier [CVPR 2019]

**Implicit field learning** [Chen and Zhang]

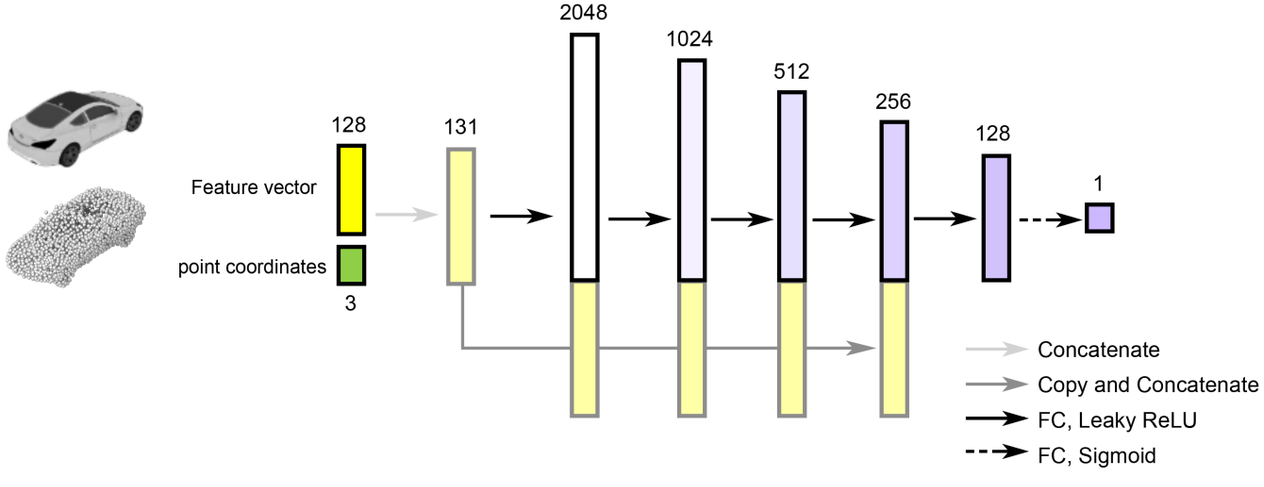



**Occupancy networks** [Mescheder et al.]



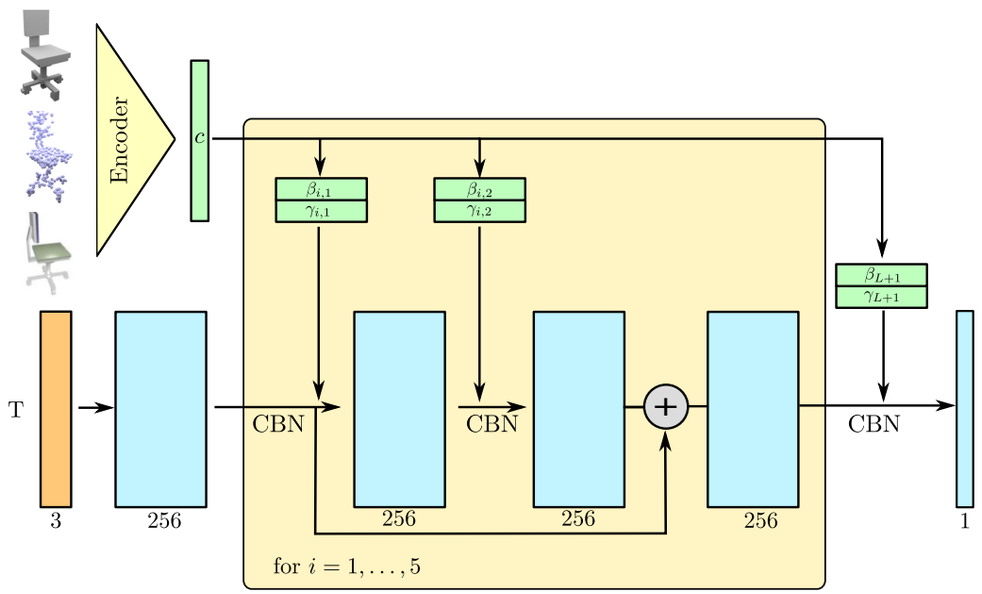
# Learning decision boundary of occupancy classifier [CVPR 2019]

**Implicit field learning** [Chen and Zhang]




- Predict point occupancy
- Supervised training using ground truth occupancy
- Use multi-resolution surface extraction

**Occupancy networks** [Mescheder et al.]



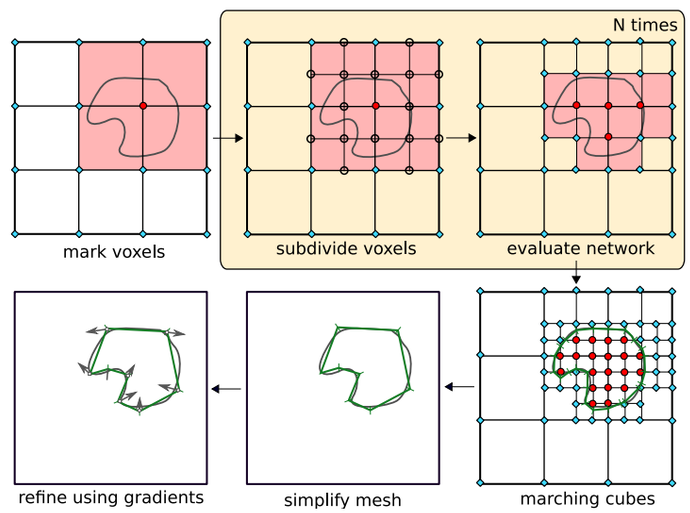
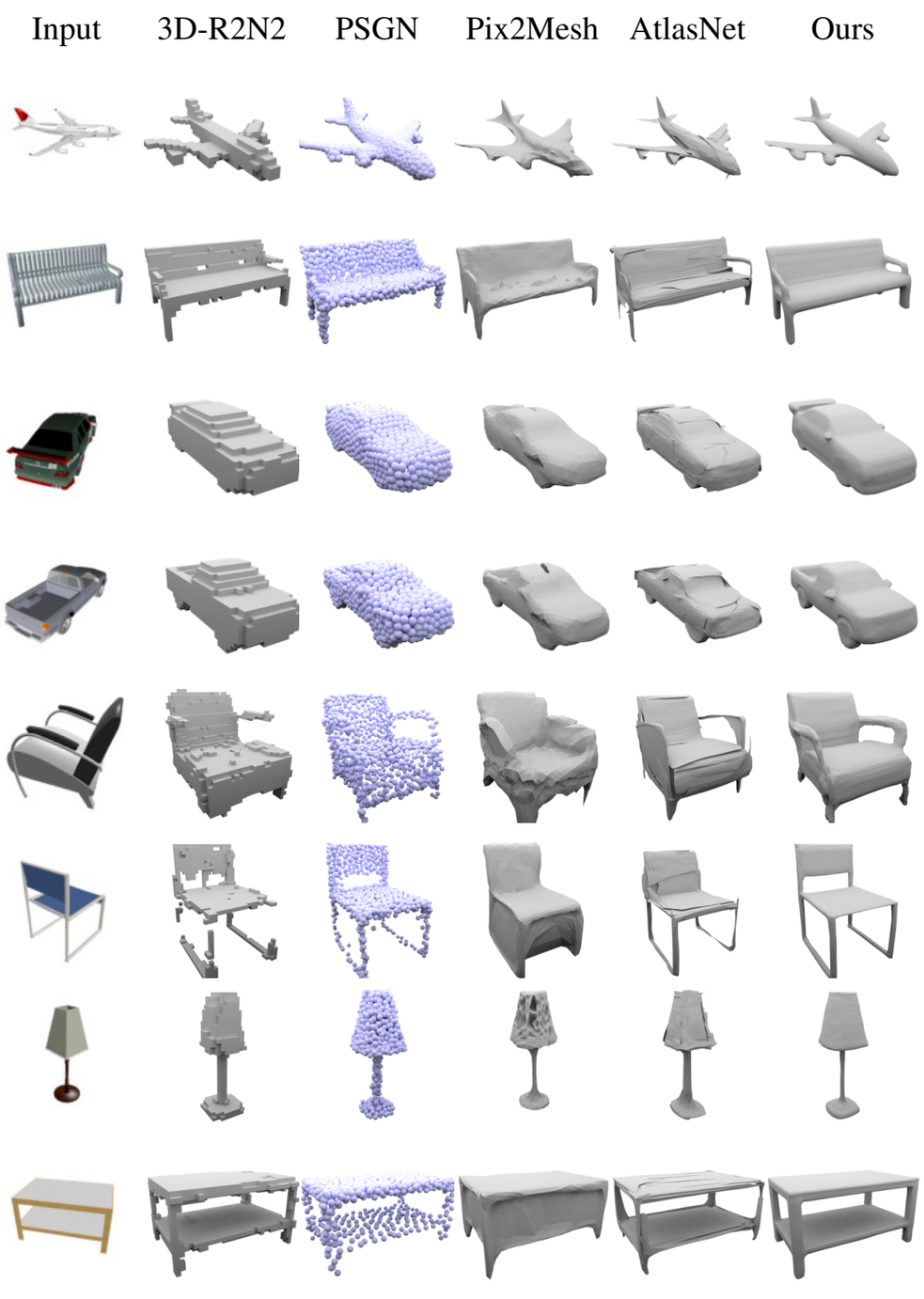
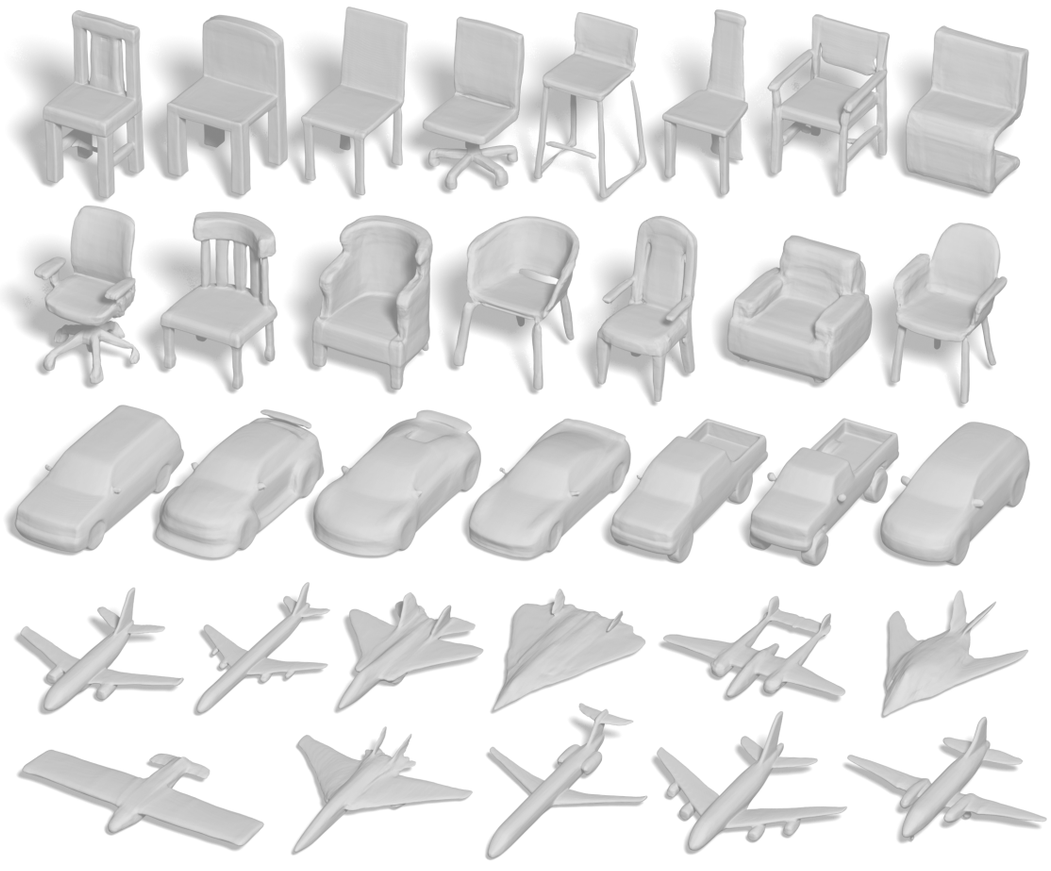




Image: L. Mescheder

# Learning decision boundary of occupancy classifier [CVPR 2019]




**Occupancy networks** [Mescheder et al.]



**Implicit field learning** [Chen and Zhang]

# Mesh-based representation



Image-based



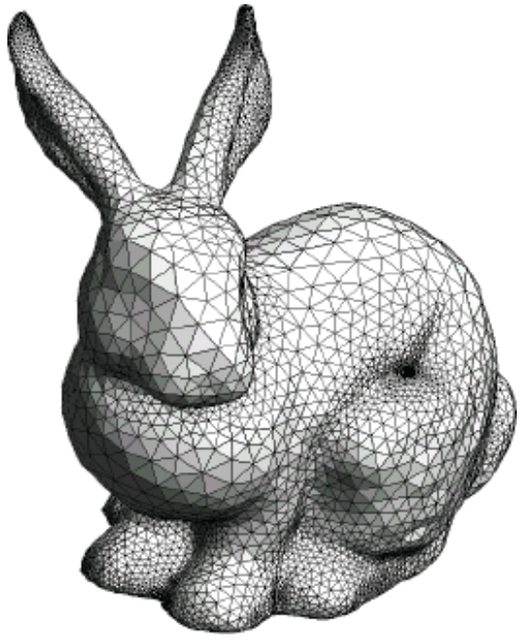
Volumetric



Point Cloud




Signed distance-based



**Mesh-based**

Credit: Michael Bronstein, Charles R. Qi, Jeong J. Park

# Pros and Cons of Mesh representations

☺ Accurate approximations of the continuous surface

# Pros and Cons of Mesh representations

- ☺ Accurate approximations of the continuous surface
- ☺ Compact: Only the surface is encoded (contrary to volumetric methods)
- ☺ Flexible: Only a handful of points can represent large approximately planar surfaces

# Pros and Cons of Mesh representations

- ☺ Accurate approximations of the continuous surface
- ☺ Compact: Only the surface is encoded (contrary to volumetric methods)
- ☺ Flexible: Only a handful of points can represent large approximately planar surfaces
- ☺ No post processing needed to render a continuous object

# Pros and Cons of Mesh representations

- ☺ Accurate approximations of the continuous surface
- ☺ Compact: Only the surface is encoded (contrary to volumetric methods)
- ☺ Flexible: Only a handful of points can represent large approximately planar surfaces
- ☺ No post processing needed to render a continuous object
- ☺ Invariance can be built-in (e.g invariance to isometries by operating on the metric)

# Pros and Cons of Mesh representations

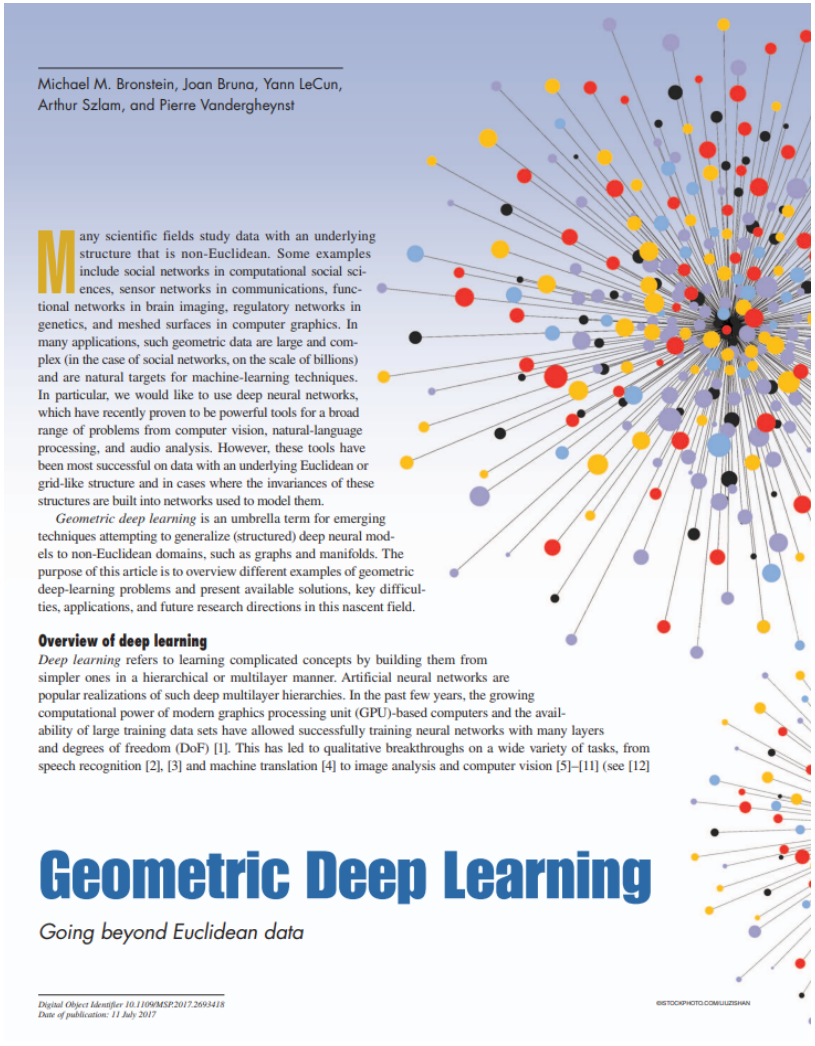
- ☺ Accurate approximations of the continuous surface
- ☺ Compact: Only the surface is encoded (contrary to volumetric methods)
- ☺ Flexible: Only a handful of points can represent large approximately planar surfaces
- ☺ No post processing needed to render a continuous object
- ☺ Invariance can be built-in (e.g invariance to isometries by operating on the metric)
- ☹ **Non-euclidean operators needed**

# Pros and Cons of Mesh representations

- ☺ Accurate approximations of the continuous surface
- ☺ Compact: Only the surface is encoded (contrary to volumetric methods)
- ☺ Flexible: Only a handful of points can represent large approximately planar surfaces
- ☺ No post processing needed to render a continuous object
- ☺ Invariance can be built-in (e.g invariance to isometries by operating on the metric)
- ☹ **Non-euclidean operators needed**

## Birth of Geometric Deep Learning

# What is Geometric Deep Learning?



Michael M. Bronstein, Joan Bruna, Yann LeCun,
Arthur Szlam, and Pierre Vandergheynst

Many scientific fields study data with an underlying structure that is non-Euclidean. Some examples include social networks in computational social sciences, sensor networks in communications, functional networks in brain imaging, regulatory networks in genetics, and meshed surfaces in computer graphics. In many applications, such geometric data are large and complex (in the case of social networks, on the scale of billions) and are natural targets for machine-learning techniques. In particular, we would like to use deep neural networks, which have recently proven to be powerful tools for a broad range of problems from computer vision, natural-language processing, and audio analysis. However, these tools have been most successful on data with an underlying Euclidean or grid-like structure and in cases where the invariances of these structures are built into networks used to model them.

*Geometric deep learning* is an umbrella term for emerging techniques attempting to generalize (structured) deep neural models to non-Euclidean domains, such as graphs and manifolds. The purpose of this article is to overview different examples of geometric deep-learning problems and present available solutions, key difficulties, applications, and future research directions in this nascent field.

**Overview of deep learning**

*Deep learning* refers to learning complicated concepts by building them from simpler ones in a hierarchical or multilayer manner. Artificial neural networks are popular realizations of such deep multilayer hierarchies. In the past few years, the growing computational power of modern graphics processing unit (GPU)-based computers and the availability of large training data sets have allowed successfully training neural networks with many layers and degrees of freedom (DoF) [1]. This has led to qualitative breakthroughs on a wide variety of tasks, from speech recognition [2], [3] and machine translation [4] to image analysis and computer vision [5]–[11] (see [12]

# Geometric Deep Learning

*Going beyond Euclidean data*

*Digital Object Identifier 10.1109/MSP.2017.2693418*
*Date of publication: 11 July 2017*





Bronstein et al., SPM 2017

# What is Geometric Deep Learning?

## Relational inductive biases, deep learning, and graph networks

**Peter W. Battaglia**[1]*, **Jessica B. Hamrick**[1], **Victor Bapst**[1], **Alvaro Sanchez-Gonzalez**[1], **Vinicius Zambaldi**[1], **Mateusz Malinowski**[1], **Andrea Tacchetti**[1], **David Raposo**[1], **Adam Santoro**[1], **Ryan Faulkner**[1], **Caglar Gulcehre**[1], **Francis Song**[1], **Andrew Ballard**[1], **Justin Gilmer**[2], **George Dahl**[2], **Ashish Vaswani**[2], **Kelsey Allen**[3], **Charles Nash**[4], **Victoria Langston**[1], **Chris Dyer**[1], **Nicolas Heess**[1], **Daan Wierstra**[1], **Pushmeet Kohli**[1], **Matt Botvinick**[1], **Oriol Vinyals**[1], **Yujia Li**[1], **Razvan Pascanu**[1]

[1]DeepMind; [2]Google Brain; [3]MIT; [4]University of Edinburgh

**Abstract**

Artificial intelligence (AI) has undergone a renaissance recently, making major progress in key domains such as vision, language, control, and decision-making. This has been due, in part, to cheap data and cheap compute resources, which have fit the natural strengths of deep learning. However, many defining characteristics of human intelligence, which developed under much different pressures, remain out of reach for current approaches. In particular, generalizing beyond one's experiences—a hallmark of human intelligence from infancy—remains a formidable challenge for modern AI.

The following is part position paper, part review, and part unification. We argue that combinatorial generalization must be a top priority for AI to achieve human-like abilities, and that structured representations and computations are key to realizing this objective. Just as biology uses nature and nurture cooperatively, we reject the false choice between "hand-engineering" and "end-to-end" learning, and instead advocate for an approach which benefits from their complementary strengths. We explore how using *relational inductive biases* within deep learning architectures can facilitate learning about entities, relations, and rules for composing them. We present a new building block for the AI toolkit with a strong relational inductive bias—the *graph network*—which generalizes and extends various approaches for neural networks that operate on graphs, and provides a straightforward interface for manipulating structured knowledge and producing structured behaviors. We discuss how graph networks can support relational reasoning and combinatorial generalization, laying the foundation for more sophisticated, interpretable, and flexible patterns of reasoning. As a companion to this paper, we have also released an open-source software library for building graph networks, with demonstrations of how to use them in practice.

Battaglia et al.,arxiv 2018

# What is Geometric Deep Learning?

## Goal:

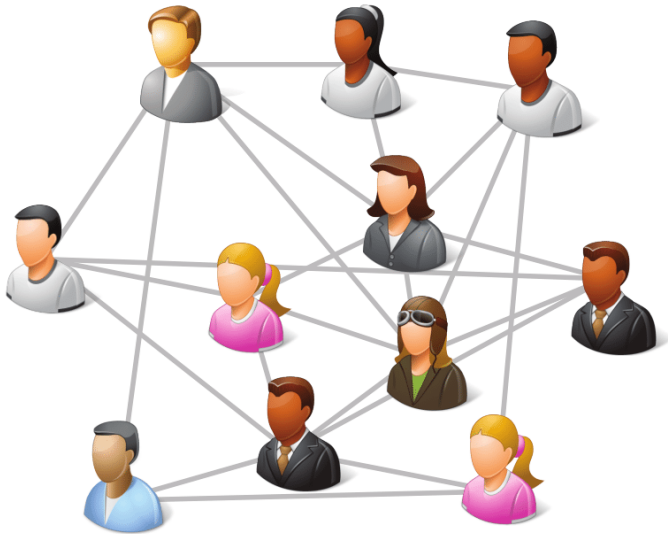
- Design learnable operators
- Optimise them w.r.t a specific task

## How?

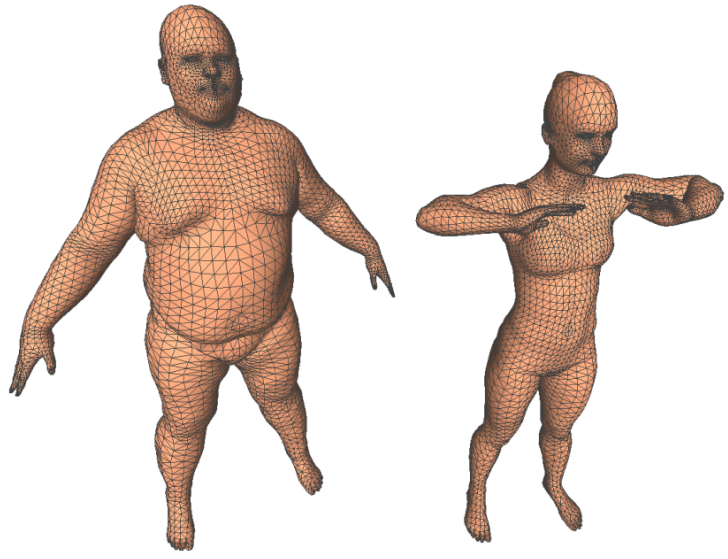
- Incorporate appropriate inductive biases related to the data structure and the task
- Encode priors and desired properties

# Inductive bias of the structure of the data: Domain

## Fixed Domain



Social network
(fixed graph)

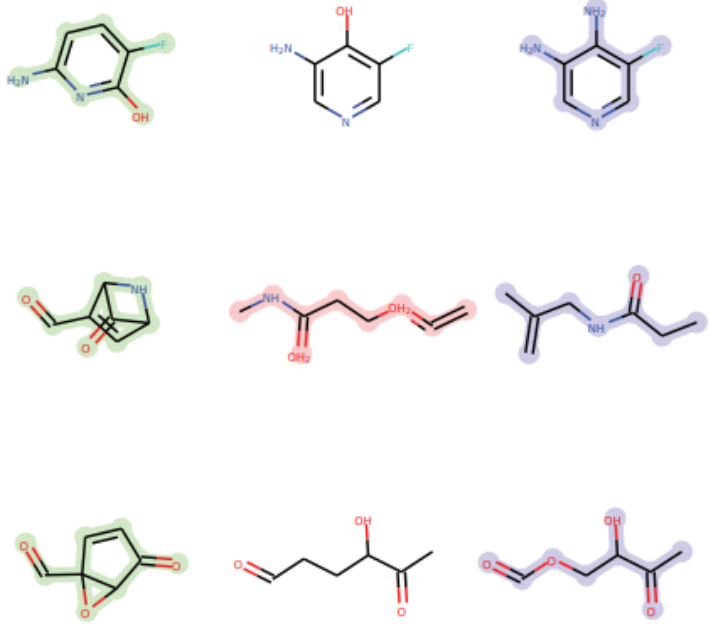


registered 3D meshes

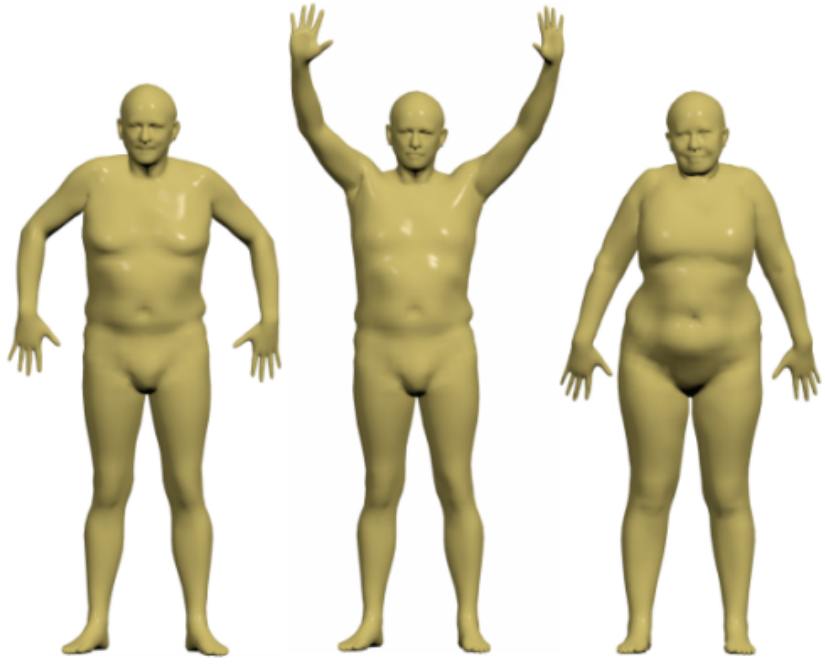
Bogo et al., CVPR 2017

# Inductive bias of the structure of the data: Domain

## Different Domains



Molecules

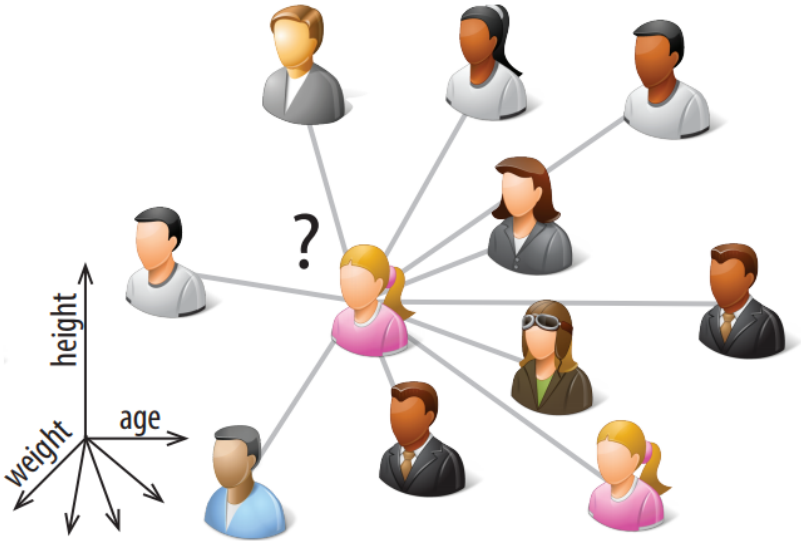


3D meshes with different connectivity

Simonovsky and Komodakis, ICANN 2019

# Inductive bias of the structure of the data: Domain

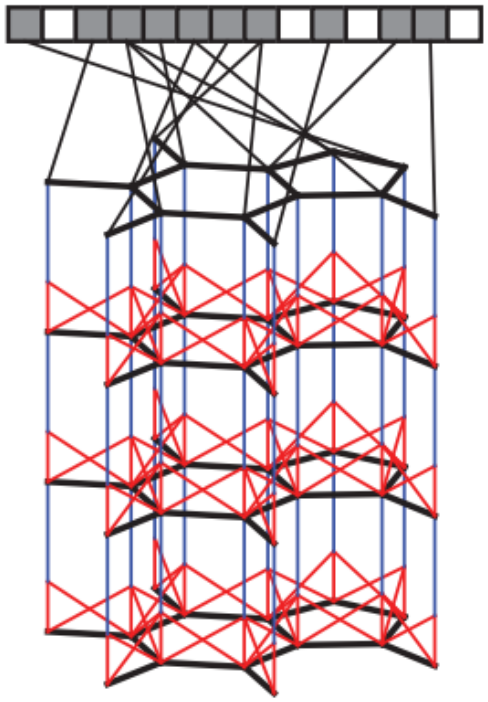
## Unknown Domain(s)



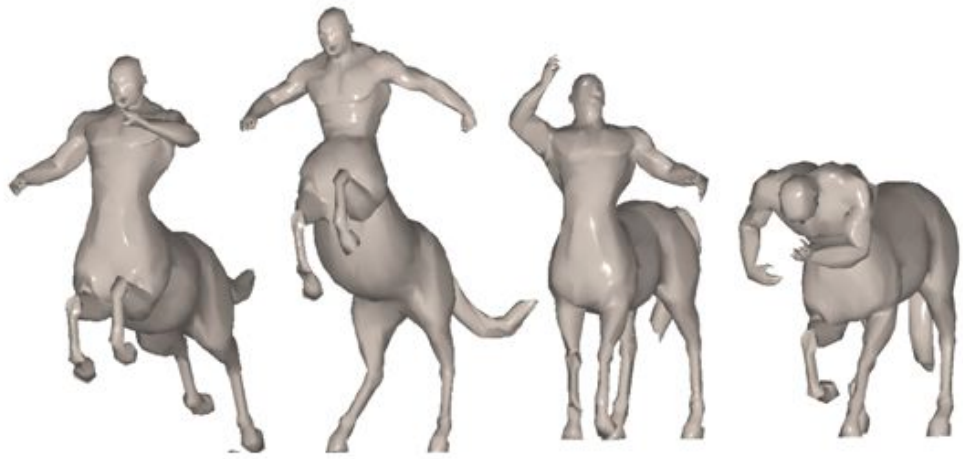

Social network with unknown connectivity
(graph metric has to be learnt)

3D point clouds
(no triangulation)

# Inductive bias of the task: Graph classification



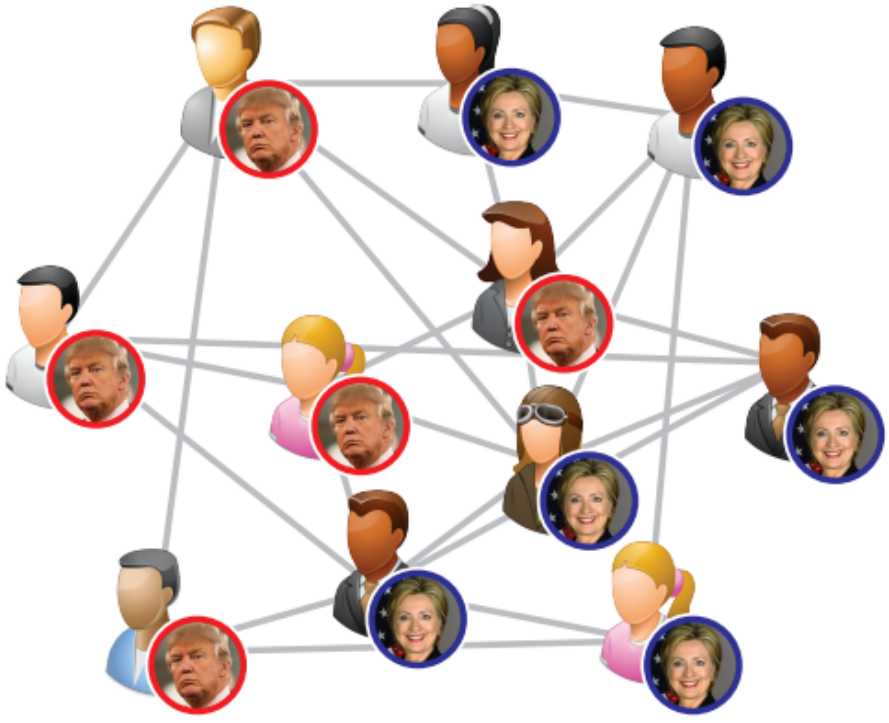
Molecule classification



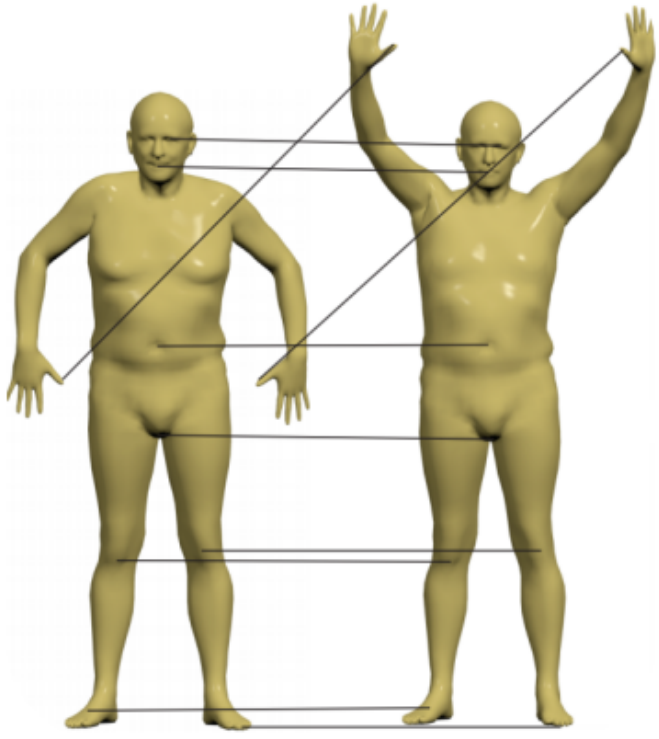
Shape retrieval

Duvenaud et al., NIPS 2015

# Inductive bias of the task: Vertex classification

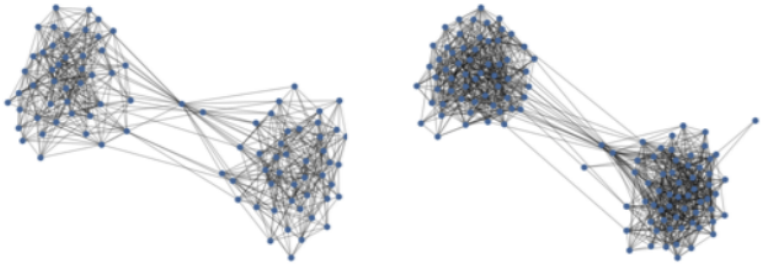


Community detection

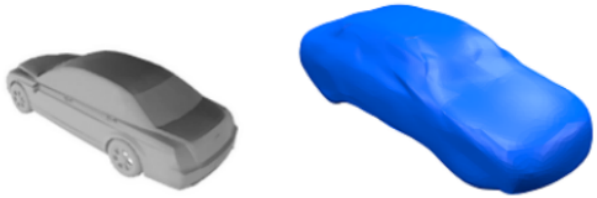


Shape correspondences

# Inductive bias of the task: Graph synthesis



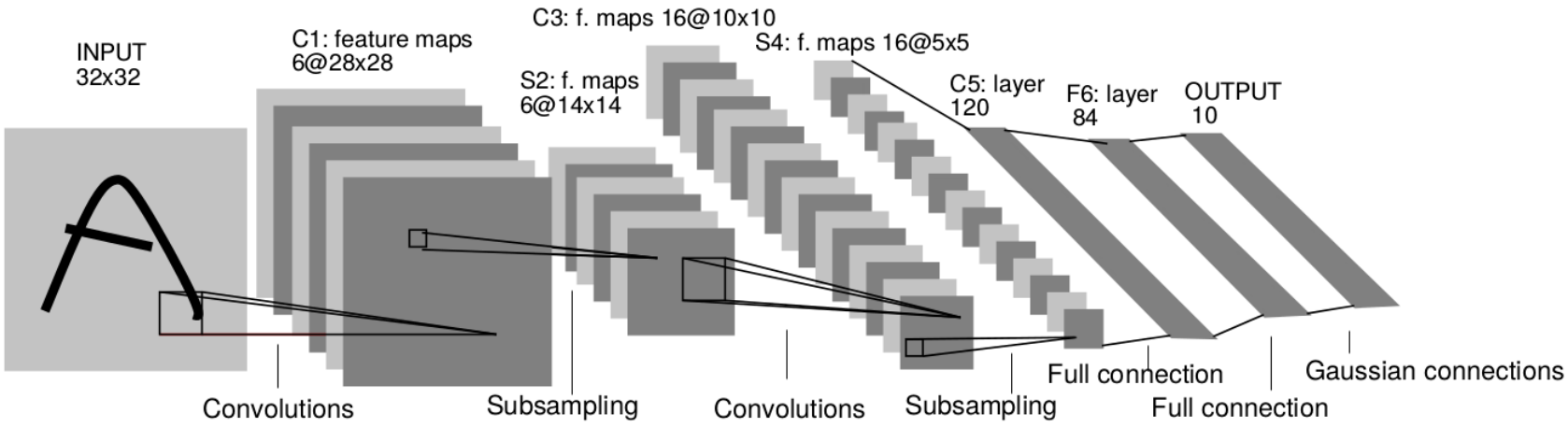
Topology generation



Mesh synthesis
(topology and signal)

You et al., ICML 2018, Smith et al., ICML 2019

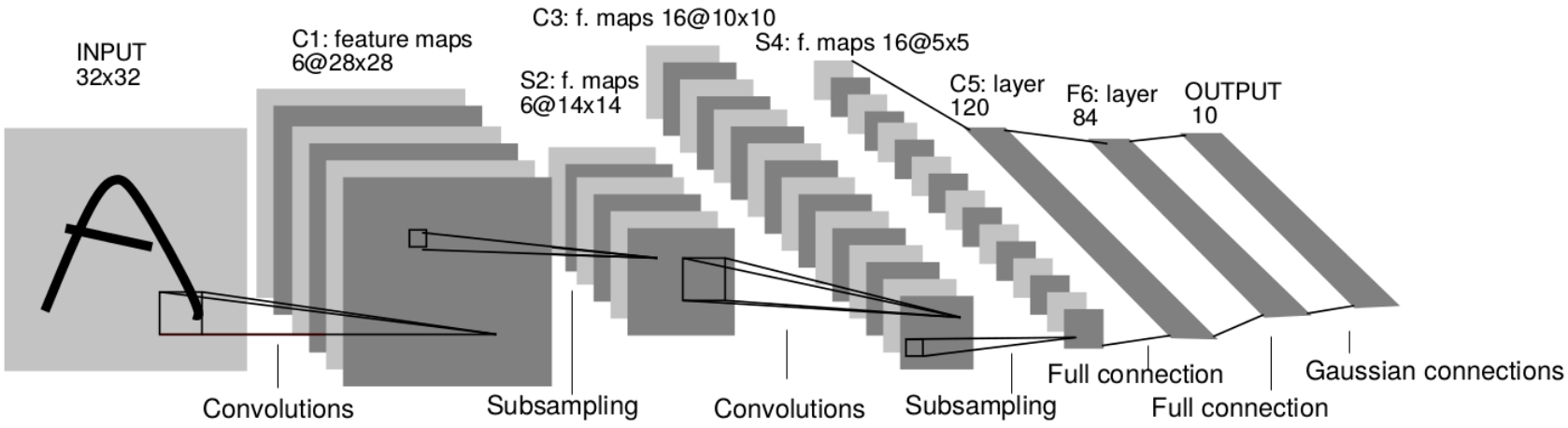
# Priors and desired properties: Revisiting CNNs




☺ Stationarity **(Convolutions)**

LeCun et al. 1989

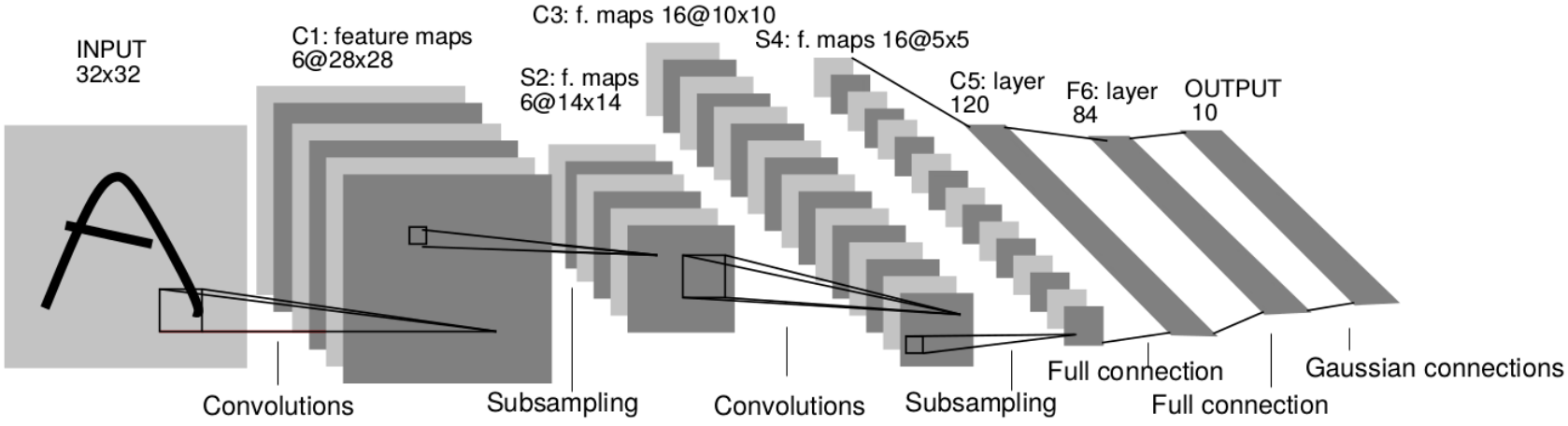
# Priors and desired properties: Revisiting CNNs




- ☺ Stationarity **(Convolutions)**
- ☺ Compositionality **(Spatial localisation of filters + Hierarhical structure)**

LeCun et al. 1989

# Priors and desired properties: Revisiting CNNs




- ☺ Stationarity **(Convolutions)**
- ☺ Compositionality **(Spatial localisation of filters + Hierarhical structure)**
- ☺ $\mathcal{O}(1)$ parameters per filter (independent of input image size $n$)
- ☺ $\mathcal{O}(n)$ complexity per layer (filtering done in the spatial domain)
- ☺ $\mathcal{O}(\log n)$ layers in classification tasks

LeCun et al. 1989

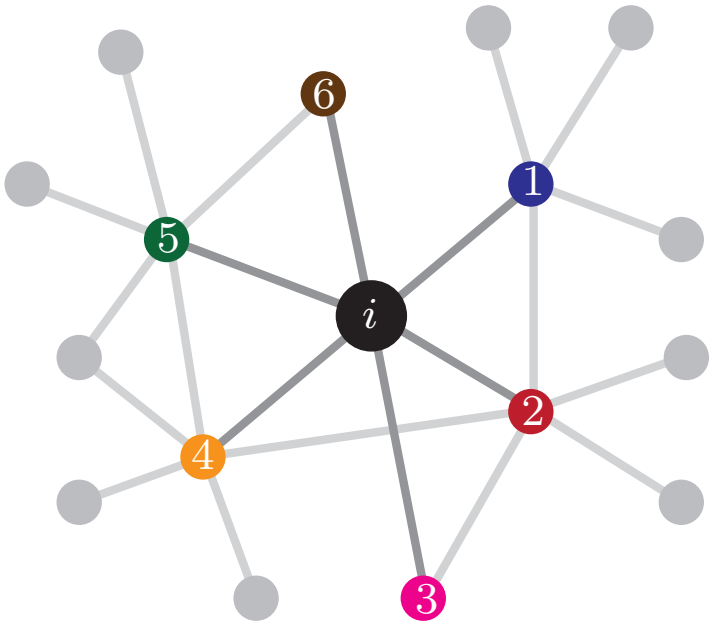
# Priors and desired properties: Going non-euclidean

- Do this properties hold for non-euclidean domains?
- **Assumption:** Non-Euclidean data are locally stationary and manifest hierarchical structures

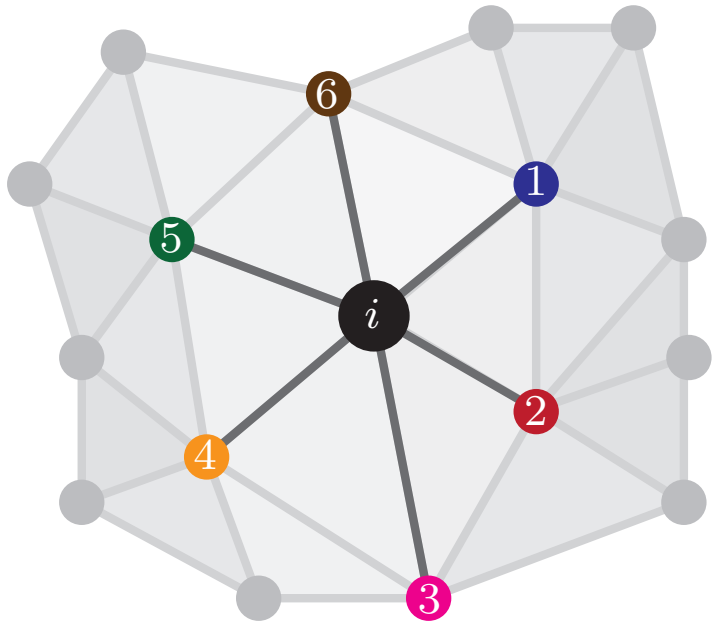
# Challenges

- How to extend euclidean operators, such as convolution and pooling, to non-euclidean-domains?
  1. Permutation invariance: Graph structured data do not admit a global ordering

# How to achieve permutation invariance?

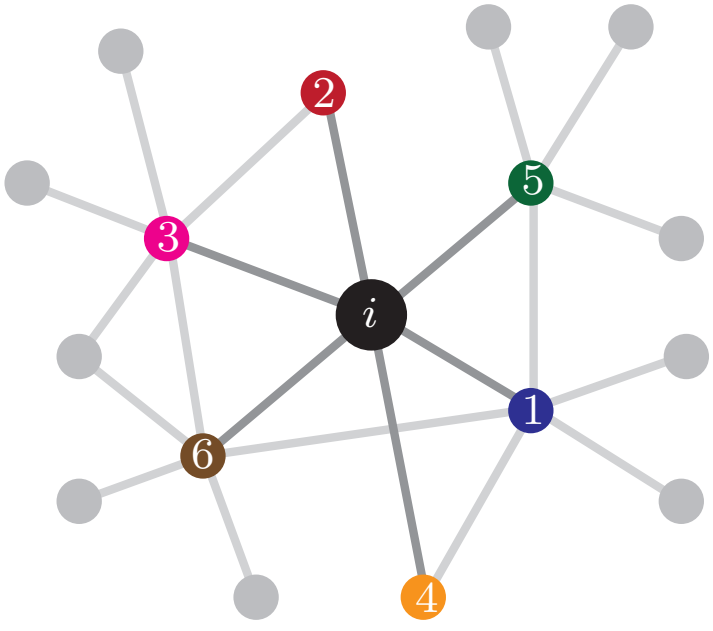



Graph (permutation)

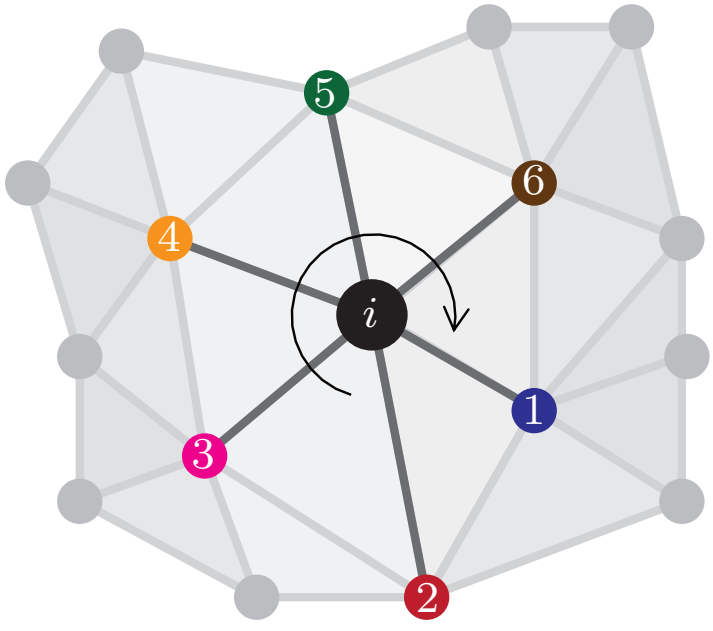



Mesh (rotation)

# How to achieve permutation invariance?




Graph (permutation)

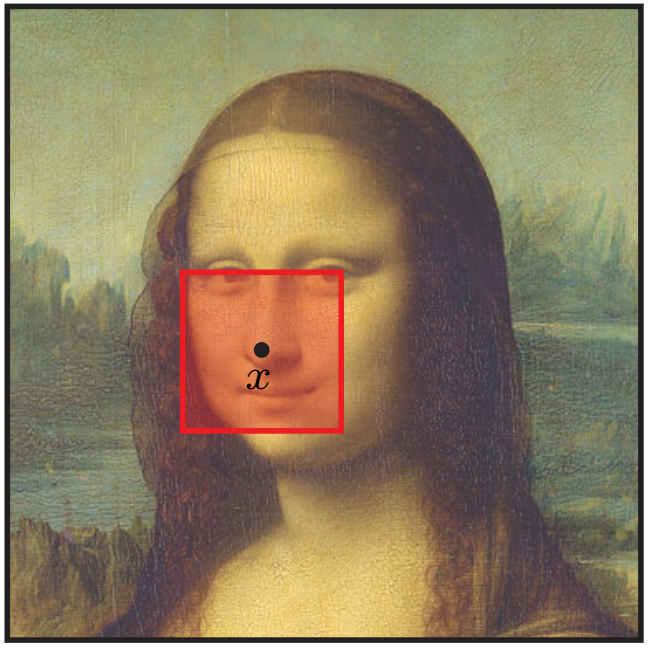



Mesh (rotation)

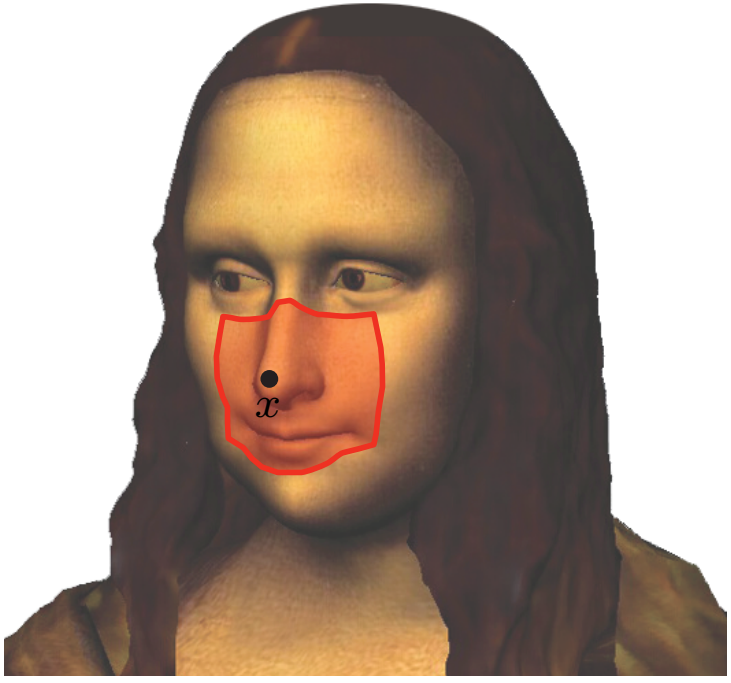
# Challenges

- How to extend euclidean operators, such as convolution and pooling, to non-euclidean-domains?
  1. Permutation invariance: Graph structured data do not admit a global ordering
  2. Transferability of the filters across local neighbourhoods (analogous to translation)

# How to transfer filters across the same non-euclidean domain?

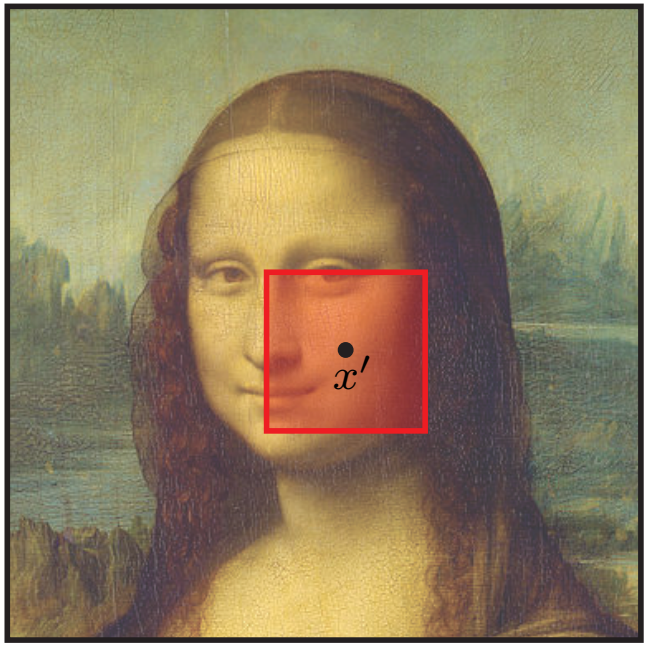



Euclidean

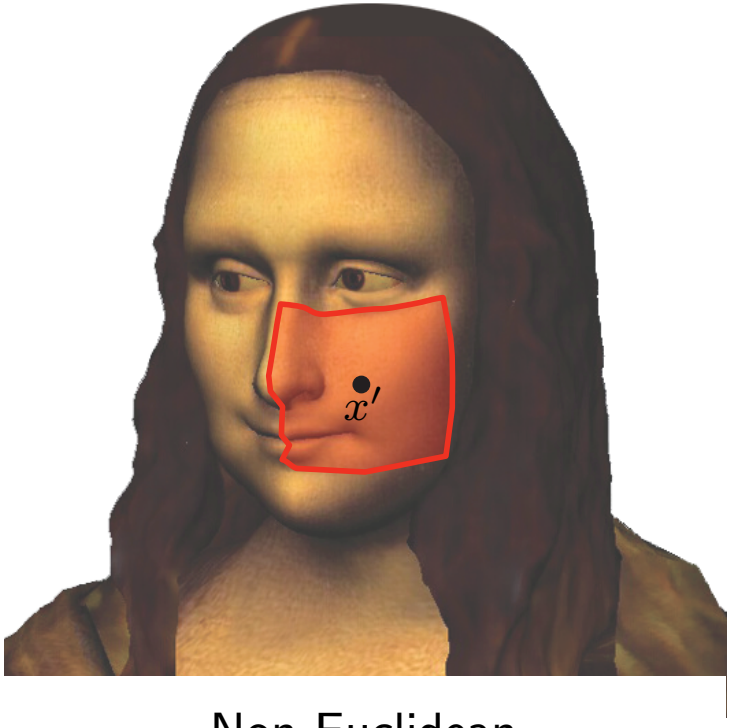



Non-Euclidean

# How to transfer filters across the same non-euclidean domain?




Euclidean

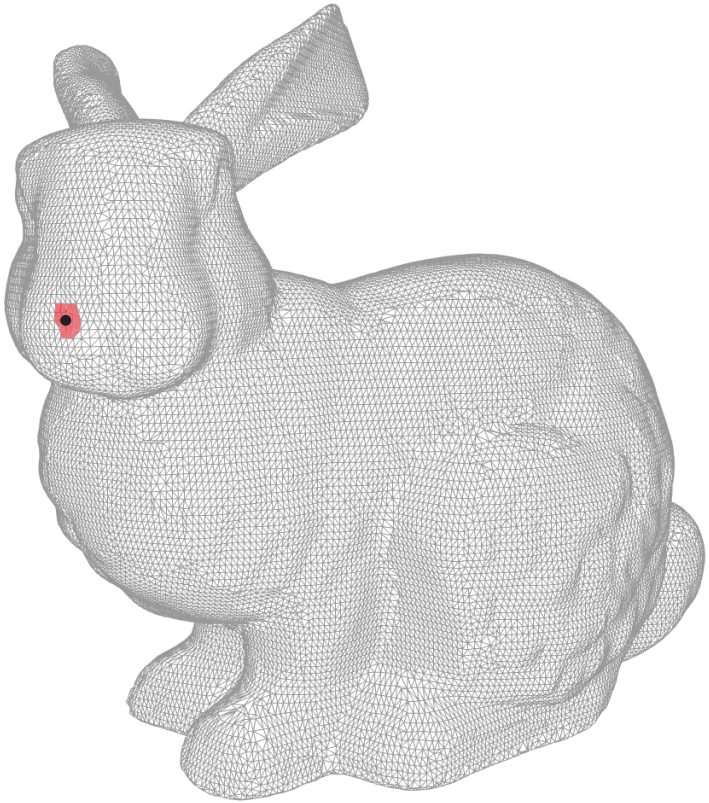



Non-Euclidean

# Challenges

- How to extend euclidean operators, such as convolution and pooling, to non-euclidean-domains?
  1. Permutation invariance: Graph structured data do not admit a global ordering
  2. Transferability of the filters across local neighbourhoods (analogous to translation)
  3. Transferability of the filters when dealing with multiple graphs

# How to transfer filters across different domains?



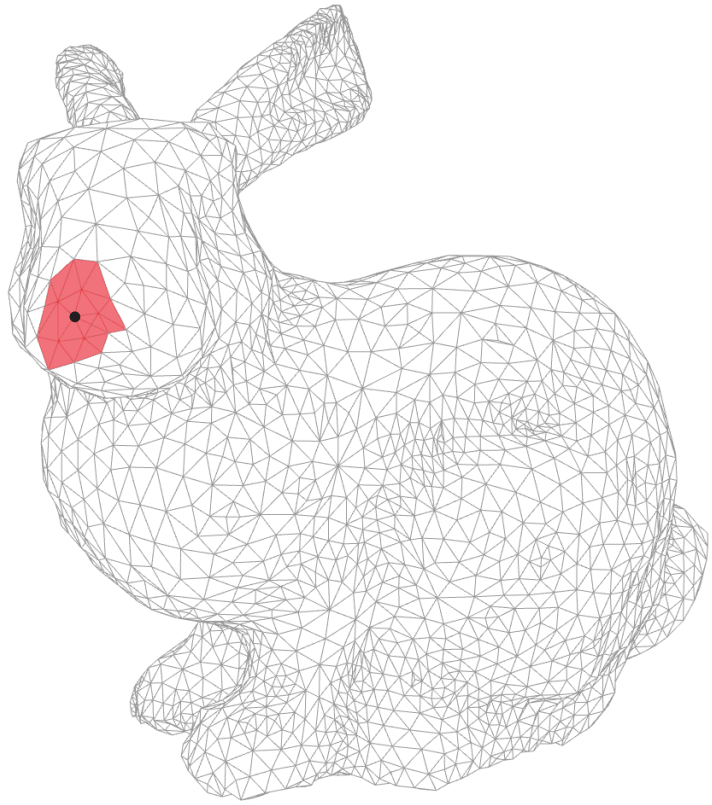


Image analogy: Can we use the same filters for low and high resolution images?

# Challenges

- How to extend euclidean operators, such as convolution and pooling, to non-euclidean-domains?
  1. Permutation invariance: Graph structured data do not admit a global ordering
  2. Transferability of the filters across local neighbourhoods (analogous to translation)
  3. Transferability of the filters when dealing with multiple graphs
  4. Transferability of the filters when dealing with multiple graphs
- Scalability: How to make them fast (and ideally parallelizable)?

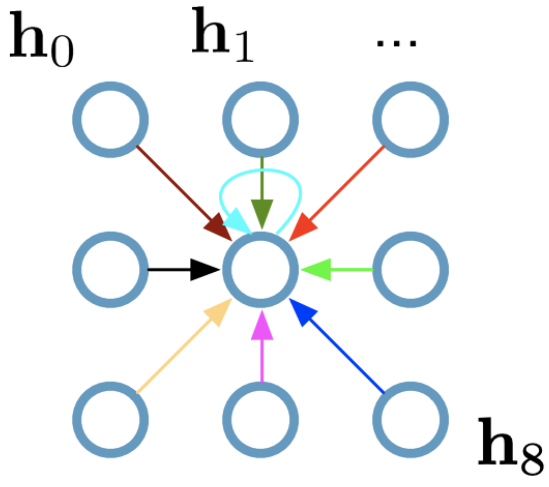
# Different Perspectives: Graph Convolutions vs Message Passing

- Graph Convolutions

Spectral

Patch Operator based (aka Spatial)

- Message Passing

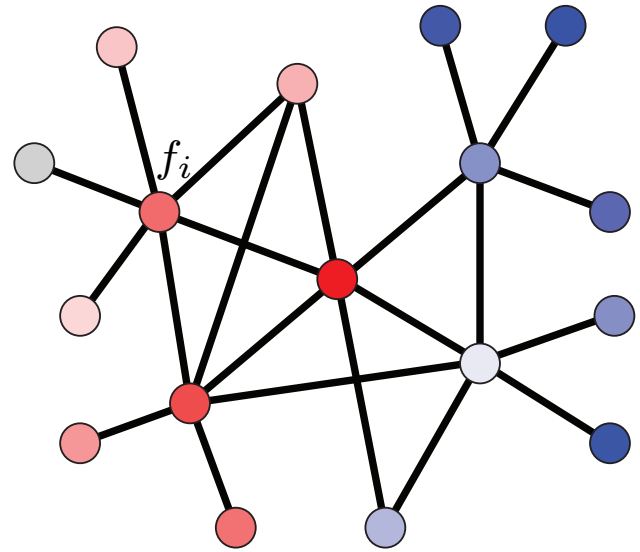


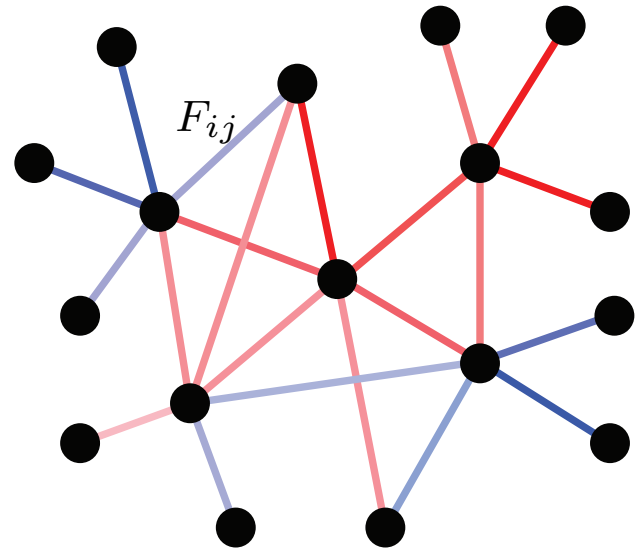

(figure by Thomas Kipf)

- They boil down to the same thing!

# Graphs: notations and basics

- Weighted undirected graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$ with vertices $\mathcal{V} = \{1, \ldots, n\}$, edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.
- Edge weights $w_{ij} \geq 0$ for $(i, j) \in \mathcal{E}$ and Vertex weights $a_i > 0$ for $i \in \mathcal{V}$
- Functions over the vertices $L^2(\mathcal{V}) = \{f : \mathcal{V} \to \mathbb{R}\}$ represented as vectors $\mathbf{f} = (f_1, \ldots, f_n)$
- Functions over the edges $L^2(\mathcal{E}) = \{F : \mathcal{E} \to \mathbb{R}\}$



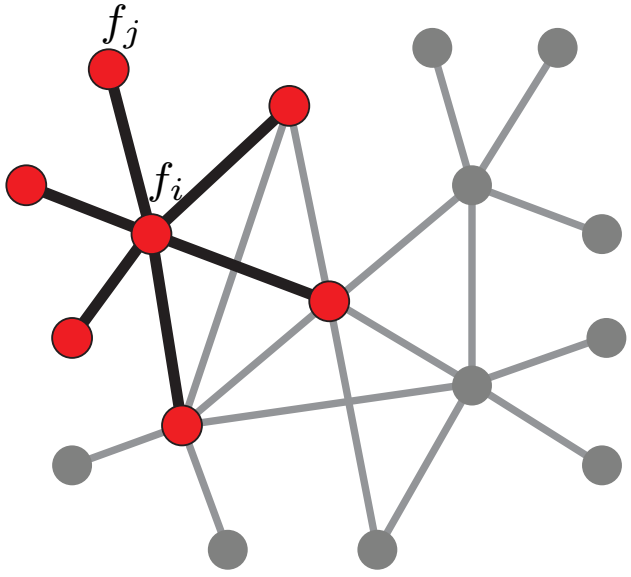



# Graphs: notations and basics

- Laplacian $\Delta : L^2(\mathcal{V}) \to L^2(\mathcal{V})$

$$= f_i \frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{ij} - \sum_{j:(i,j)\in\mathcal{E}} w_{ij} f_j$$

difference between $f$ and its local average

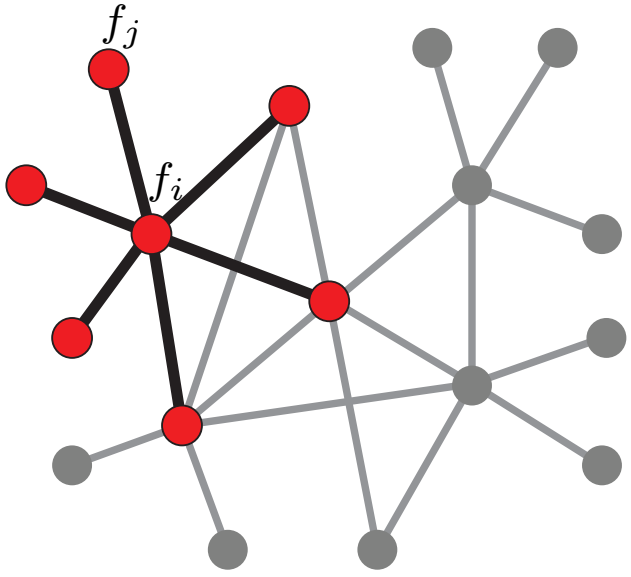


# Graphs: notations and basics

- Laplacian $\Delta : L^2(\mathcal{V}) \to L^2(\mathcal{V})$

$$= \quad f_i \frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{ij} - \sum_{j:(i,j)\in\mathcal{E}} w_{ij} f_j$$

difference between $f$ and its local average



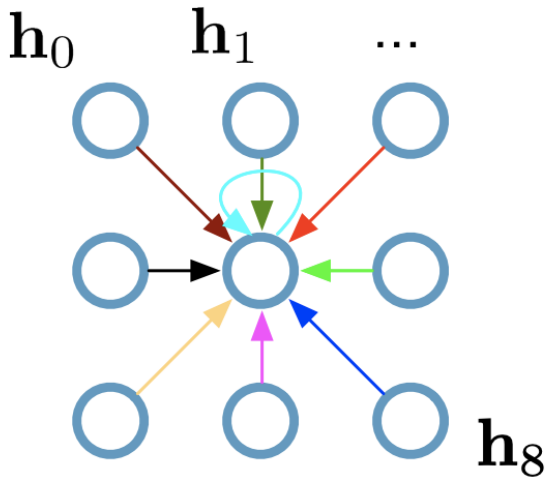

- Represented as a positive semi-definite $n \times n$ matrix $\mathbf{\Delta} = \mathbf{A}^{-1}(\mathbf{D} - \mathbf{W})$ where $\mathbf{W} = (w_{ij})$, $\mathbf{A} = diag(a_1, a_2 \dots \ a_n)$ and $\mathbf{D} = \mathrm{diag}(\sum_{j\neq i} w_{ij})$.
  Symmetric Normalized Laplacian: $\mathbf{\Delta} = \boldsymbol{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$

# Different Perspectives: Graph Convolutions vs Message Passing

- Graph Convolutions

**Spectral** Patch Operator based (aka Spatial)

- Message Passing



# Revisiting Euclidean Convolution

- Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function
$$(f \star g)(x) = \int_{-\pi}^{\pi} f(x')g(x - x')dx'$$
- Traditional convolution needs to translate the kernel $g$ across different locations of the domain.
- The notion of translation in a non-euclidean domain is elusive.
- Convolution theorem: "The fourier transform of the convolution between two functions is the dot product of their fourier coefficients"
$$(f \star g) = \mathcal{F}^{-1}\{\mathcal{F}(f) \cdot \mathcal{F}(g)\}$$
- Fourier transform on a non-euclidean domain?

# Convolution Theorem

- Fourier basis in non-euclidean domains: eigenvectors of the Laplacian $\mathbf{\Phi} =$ i.e.

$$\mathbf{\Delta}\Phi = \mathbf{\Phi}\mathbf{\Lambda}$$

- Convolution theorem: "The fourier transform of the convolution between two functions is the dot product of their fourier coefficients"

$$(f \star g) = \mathcal{F}^{-1}\{\mathcal{F}(f) \cdot \mathcal{F}(g)\}$$

- Spectral convolution of $\mathbf{f}, \mathbf{g} \in L^2(\mathcal{V})$ can be defined by analogy

$$\mathbf{f} \star \mathbf{g} = \sum_{k \geq 1} \langle \mathbf{f}, \boldsymbol{\phi}_k \rangle_{L^2(\mathcal{V})} \langle \mathbf{g}, \boldsymbol{\phi}_k \rangle_{L^2(\mathcal{V})} \boldsymbol{\phi}_k$$

# Convolution Theorem

- Fourier basis in non-euclidean domains: eigenvectors of the Laplacian $\mathbf{\Phi} =$ i.e.

$$\mathbf{\Delta}\Phi = \mathbf{\Phi}\mathbf{\Lambda}$$

- Convolution theorem: "The fourier transform of the convolution between two functions is the dot product of their fourier coefficients"

$$(f \star g) = \mathcal{F}^{-1}\{\mathcal{F}(f) \cdot \mathcal{F}(g)\}$$

- Spectral convolution of $\mathbf{f}, \mathbf{g} \in L^2(\mathcal{V})$ can be defined by analogy

$$\mathbf{f} \star \mathbf{g} = \sum_{k \geq 1} \underbrace{\langle \mathbf{f}, \boldsymbol{\phi}_k \rangle_{L^2(\mathcal{V})} \langle \mathbf{g}, \boldsymbol{\phi}_k \rangle_{L^2(\mathcal{V})}}_{\text{product in the Fourier domain}} \boldsymbol{\phi}_k$$

# Convolution Theorem

- Fourier basis in non-euclidean domains: eigenvectors of the Laplacian $\mathbf{\Phi} =$ i.e.

$$\mathbf{\Delta}\Phi = \mathbf{\Phi}\mathbf{\Lambda}$$

- Convolution theorem: "The fourier transform of the convolution between two functions is the dot product of their fourier coefficients"

$$(f \star g) = \mathcal{F}^{-1}\{\mathcal{F}(f) \cdot \mathcal{F}(g)\}$$

- Spectral convolution of $\mathbf{f}, \mathbf{g} \in L^2(\mathcal{V})$ can be defined by analogy

$$\mathbf{f} \star \mathbf{g} = \underbrace{\sum_{k \geq 1} \underbrace{\langle \mathbf{f}, \boldsymbol{\phi}_k \rangle_{L^2(\mathcal{V})} \langle \mathbf{g}, \boldsymbol{\phi}_k \rangle_{L^2(\mathcal{V})}}_{\text{product in the Fourier domain}} \boldsymbol{\phi}_k}_{\text{inverse Fourier transform}}$$

# Convolution Theorem

- In matrix-vector notation:

$$\mathbf{f} \star \mathbf{g} = \underbrace{\mathbf{\Phi}}_{\text{inverse Fourier}} \cdot \underbrace{(\mathbf{\Phi}^\top \mathbf{g}) \circ (\mathbf{\Phi}^\top \mathbf{f})}_{\text{product in the Fourier domain}}$$

$$\mathbf{f} \star \mathbf{g} = \mathbf{\Phi}\, \boldsymbol{G} (\mathbf{\Phi}^\top \mathbf{f})$$

where $\boldsymbol{G} = diag(\hat{g_1}, \hat{g_2} \ldots \hat{g_n})$, the fourier coefficients of $g$

# Convolution Theorem

- In matrix-vector notation:

$$\mathbf{f} \star \mathbf{g} = \underbrace{\mathbf{\Phi}}_{\text{inverse Fourier}} \cdot \underbrace{(\mathbf{\Phi}^\top \mathbf{g}) \circ (\mathbf{\Phi}^\top \mathbf{f})}_{\text{product in the Fourier domain}}$$

$$\mathbf{f} \star \mathbf{g} = \mathbf{\Phi}\, \boldsymbol{G}(\mathbf{\Phi}^\top \mathbf{f})$$

  where $\boldsymbol{G} = diag(\hat{g_1}, \hat{g_2} \ldots \hat{g_n})$, the fourier coefficients of $g$

- $\boldsymbol{G}$ can be learned!

# Spectral graph CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}} \mathbf{\Phi}\mathbf{G}_{i,j}\mathbf{\Phi}^\top \mathbf{F}_i\right) \qquad \begin{array}{l} i = 1, \ldots, d_{in} \\ j = 1, \ldots, d_{out} \end{array}$$

where $\xi$ a non-linearity, $\mathbf{G}_{i,j} = n \times n$ diagonal matrix of filter coefficients for the input dimension $i$ and output dimension $j$.

Bruna et al., ICLR 2014

# Spectral graph CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}} \mathbf{\Phi}\mathbf{G}_{i,j}\mathbf{\Phi}^\top \mathbf{F}_i\right) \qquad \begin{array}{l} i = 1, \ldots, d_{in} \\ j = 1, \ldots, d_{out} \end{array}$$

where $\xi$ a non-linearity, $\mathbf{G}_{i,j} = n \times n$ diagonal matrix of filter coefficients for the input dimension $i$ and output dimension $j$.

☹ No guarantee of spatial localization of filters

Bruna et al., ICLR 2014

# Spectral graph CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}} \mathbf{\Phi}\mathbf{G}_{i,j}\mathbf{\Phi}^\top \mathbf{F}_i\right) \qquad \begin{array}{l} i = 1, \ldots, d_{in} \\ j = 1, \ldots, d_{out} \end{array}$$

where $\xi$ a non-linearity, $\mathbf{G}_{i,j} = n \times n$ diagonal matrix of filter coefficients for the input dimension $i$ and output dimension $j$.

☹ No guarantee of spatial localization of filters
☹ $\mathcal{O}(n)$ parameters per layer

Bruna et al., ICLR 2014

# Spectral graph CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}} \mathbf{\Phi}\mathbf{G}_{i,j}\mathbf{\Phi}^\top \mathbf{F}_i\right) \qquad \begin{array}{l} i = 1, \ldots, d_{in} \\ j = 1, \ldots, d_{out} \end{array}$$

where $\xi$ a non-linearity, $\mathbf{G}_{i,j} = n \times n$ diagonal matrix of filter coefficients for the input dimension $i$ and output dimension $j$.

- ☹ No guarantee of spatial localization of filters
- ☹ $\mathcal{O}(n)$ parameters per layer
- ☹ $\mathcal{O}(n^2)$ computation of forward and inverse Fourier transforms $\mathbf{\Phi}^\top, \mathbf{\Phi}$ (no FFT on graphs)

Bruna et al., ICLR 2014

# Spectral graph CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}} \mathbf{\Phi}\mathbf{G}_{i,j}\mathbf{\Phi}^\top \mathbf{F}_i\right) \qquad \begin{array}{l} i = 1, \ldots, d_{in} \\ j = 1, \ldots, d_{out} \end{array}$$

where $\xi$ a non-linearity, $\mathbf{G}_{i,j} = n \times n$ diagonal matrix of filter coefficients for the input dimension $i$ and output dimension $j$.

- ☹ No guarantee of spatial localization of filters
- ☹ $\mathcal{O}(n)$ parameters per layer
- ☹ $\mathcal{O}(n^2)$ computation of forward and inverse Fourier transforms $\mathbf{\Phi}^\top, \mathbf{\Phi}$ (no FFT on graphs)
- ☹ Filters are basis-dependent $\Rightarrow$ does not generalize across graphs!

Bruna et al., ICLR 2014

# ChebNet: Spectral graph CNN with polynomial filters

Parametrise filter $\boldsymbol{G}$, as a polynomial of the eigenvalue matrix

$$\boldsymbol{G} = \sum_{k=0}^{r} \alpha_k \boldsymbol{\Lambda}_m^k$$

where $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_r)^\top$ is the vector of filter parameters

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Parametrise filter $\boldsymbol{G}$, as a polynomial of the eigenvalue matrix

$$\boldsymbol{G} = \sum_{k=0}^{r} \alpha_k \boldsymbol{\Lambda}_m^k$$

where $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_r)^\top$ is the vector of filter parameters

Now the convolution becomes

$$\mathbf{f} \star \mathbf{g} = \boldsymbol{\Phi} \sum_{k=0}^{r} a_k \boldsymbol{\Lambda}^k \boldsymbol{\Phi}^\top \mathbf{f}$$

$$\mathbf{f} \star \mathbf{g} = \sum_{k=0}^{r} a_k \boldsymbol{\Delta}^k \mathbf{f}$$

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Convolutional layer with Chebyshev Polynomial filters (stable under perturbations of coefficients)

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}}\sum_{k=0}^{r} a_{i,j,k}\, T_k(\mathbf{\Delta})\mathbf{F}_i\right) \qquad \begin{array}{l} i = 1,\ldots,d_{in} \\ j = 1,\ldots,d_{out} \end{array}$$

where $T_k$ is the Chebyshev polynomial of order $k$.

☺ Filters have guaranteed $r$-hops support

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Convolutional layer with Chebyshev Polynomial filters (stable under perturbations of coefficients)

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}}\sum_{k=0}^{r} a_{i,j,k}\, T_k(\mathbf{\Delta})\mathbf{F}_i\right) \qquad \begin{array}{l} i = 1,\ldots,d_{in} \\ j = 1,\ldots,d_{out} \end{array}$$

where $T_k$ is the Chebyshev polynomial of order $k$.

☺ Filters have guaranteed $r$-hops support

☺ $\mathcal{O}(1)$ parameters per layer

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Convolutional layer with Chebyshev Polynomial filters (stable under perturbations of coefficients)

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}}\sum_{k=0}^{r} a_{i,j,k}\, T_k(\mathbf{\Delta})\mathbf{F}_i\right) \qquad \begin{array}{l} i = 1,\ldots,d_{in} \\ j = 1,\ldots,d_{out} \end{array}$$

where $T_k$ is the Chebyshev polynomial of order $k$.

- ☺ Filters have guaranteed $r$-hops support
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ No explicit computation of $\mathbf{\Phi}^\top, \mathbf{\Phi} \Rightarrow \mathcal{O}(|\mathcal{E}))$ computational complexity

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Convolutional layer with Chebyshev Polynomial filters (stable under perturbations of coefficients)

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}}\sum_{k=0}^{r} a_{i,j,k}\, T_k(\mathbf{\Delta})\mathbf{F}_i\right) \qquad \begin{matrix} i = 1, \ldots, d_{in} \\ j = 1, \ldots, d_{out} \end{matrix}$$

where $T_k$ is the Chebyshev polynomial of order $k$.

- ☺ Filters have guaranteed $r$-hops support
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ No explicit computation of $\mathbf{\Phi}^\top, \mathbf{\Phi} \Rightarrow \mathcal{O}(n)$ computational complexity (assuming sparsely-connected graph)

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Convolutional layer with Chebyshev Polynomial filters (stable under perturbations of coefficients)

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}}\sum_{k=0}^{r} a_{i,j,k}\, T_k(\mathbf{\Delta})\mathbf{F}_i\right) \qquad \begin{matrix} i = 1,\dots,d_{in} \\ j = 1,\dots,d_{out} \end{matrix}$$

where $T_k$ is the Chebyshev polynomial of order $k$.

- ☺ Filters have guaranteed $r$-hops support
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ No explicit computation of $\mathbf{\Phi}^\top, \mathbf{\Phi} \Rightarrow \mathcal{O}(n)$ computational complexity (assuming sparsely-connected graph)
- ☹ Isotropic kernels

Defferrard, Bresson and Vandergheynst, NIPS 2016

# ChebNet: Spectral graph CNN with polynomial filters

Convolutional layer with Chebyshev Polynomial filters (stable under perturbations of coefficients)

$$\mathbf{F'}_j = \xi\left(\sum_{i=1}^{d_{in}}\sum_{k=0}^{r} a_{i,j,k}\, T_k(\mathbf{\Delta})\mathbf{F}_i\right) \qquad \begin{matrix} i = 1,\ldots,d_{in} \\ j = 1,\ldots,d_{out} \end{matrix}$$

where $T_k$ is the Chebyshev polynomial of order $k$.

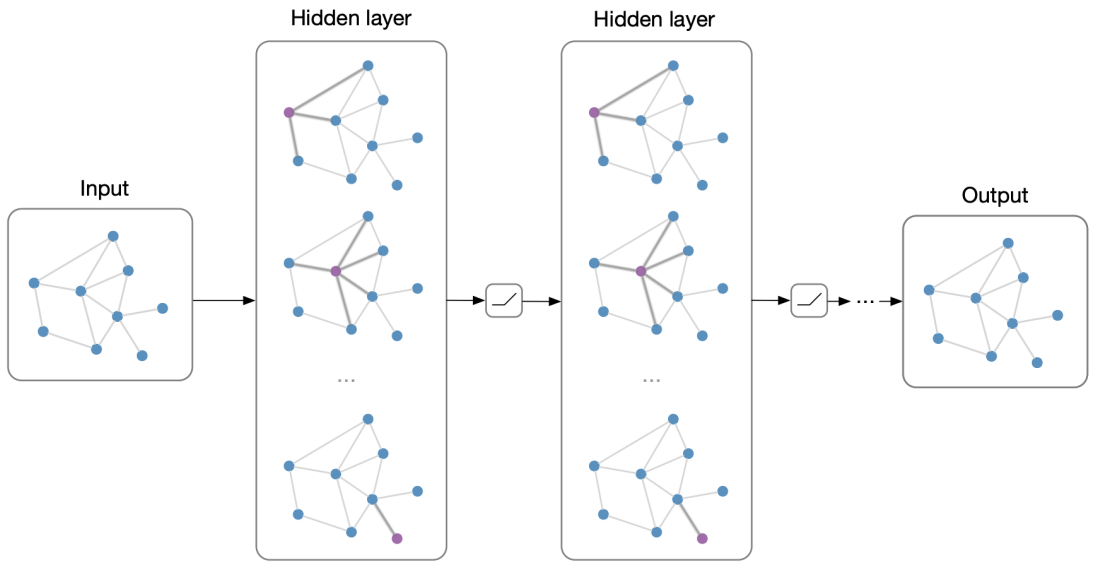
- ☺ Filters have guaranteed $r$-hops support
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ No explicit computation of $\mathbf{\Phi}^\top, \mathbf{\Phi} \Rightarrow \mathcal{O}(n)$ computational complexity (assuming sparsely-connected graph)
- ☹ Isotropic kernels
- ☹ Domain dependent (different Laplacian for each graph)

Defferrard, Bresson and Vandergheynst, NIPS 2016

# Graph Convolutional Network (GCN): Simplified ChebNet - Going deeper

- First order polynomial
- More layers are preferred over larger respective fields
- Stack multiple layers. First actually "deep" architecture on graphs

$$\mathbf{F}' = \xi\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-1/2}\mathbf{F}\boldsymbol{G}\right)$$




(figure by Thomas Kipf)

Kipf and Welling, ICLR 2017

# Example: citation networks

GCN: First state-of-the art GraphNN for graph classification tasks

| **Method** | **Cora**[1] | **PubMed**[2] |
|---|---|---|
| Manifold Regularization[3] | 59.5% | 70.7% |
| Semidefinite Embedding[4] | 59.0% | 71.1% |
| Label Propagation[5] | 68.0% | 63.0% |
| DeepWalk[6] | 67.2% | 65.3% |
| Planetoid[7] | 75.7% | 77.2% |
| **Graph Convolutional Net**[8] | **81.59%** | **78.72%** |

Classification accuracy of different methods on citation network datasets

Monti et al. 2016; data: [1,2]Sen et al. 2008; methods: [3]Belkin et al. 2006; [4]Weston et al. 2012; [5]Zhu et al. 2003; [6]Perozzi et al. 2014; [7]Yang et al. 2016; [8]Kipf, Welling 2016

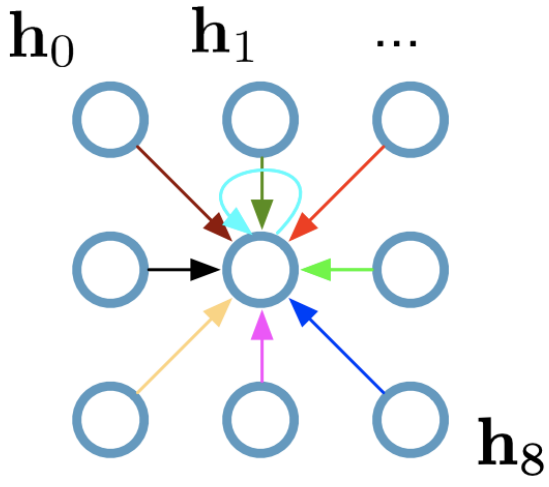
# Different Perspectives: Graph Convolutions vs Message Passing

- Graph Convolutions

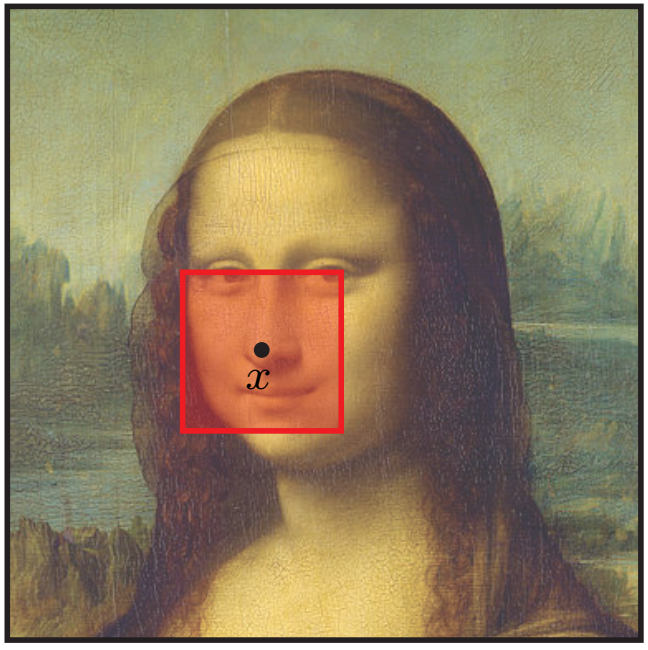
Spectral

**Patch Operator based (aka Spatial)**

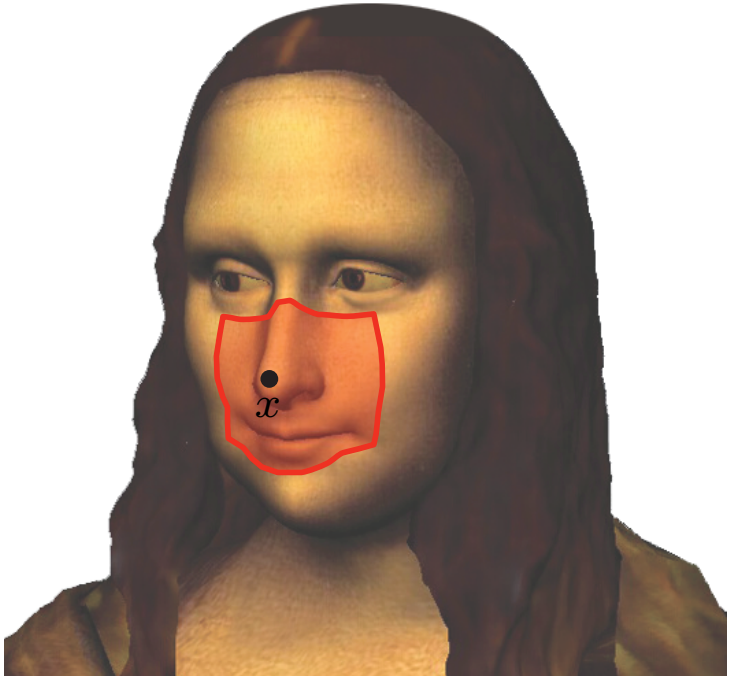
- Message Passing



# How to transfer filters across the same non-euclidean domain?

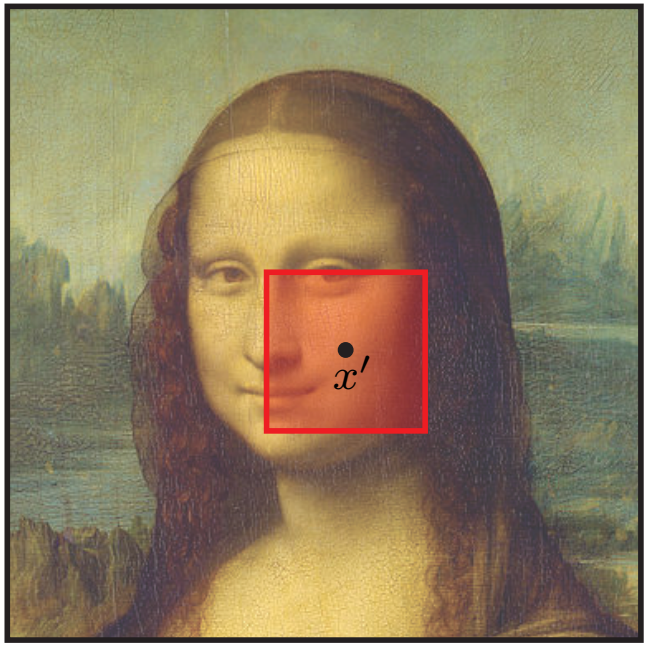



Euclidean

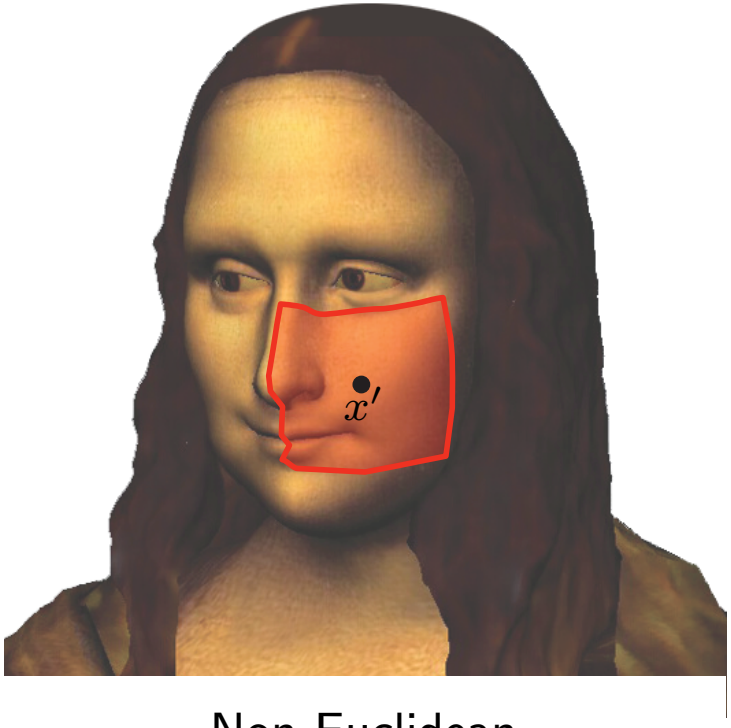



Non-Euclidean

# How to transfer filters across the same non-euclidean domain?




Euclidean

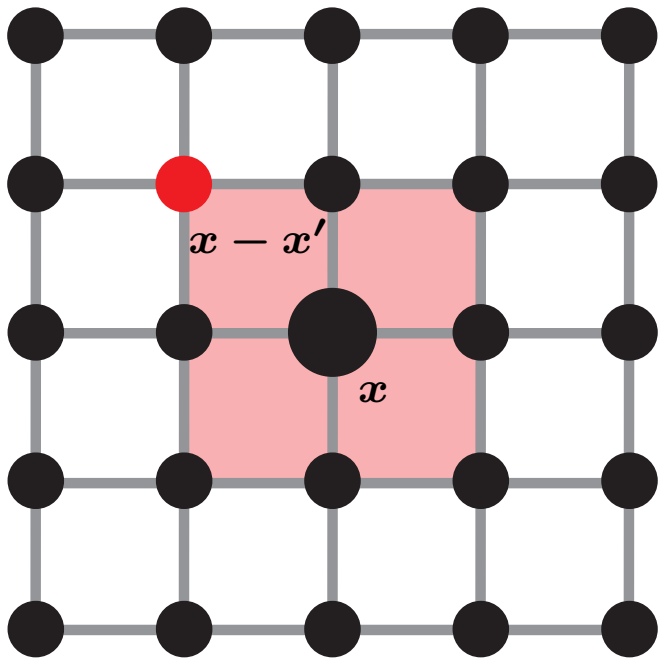



Non-Euclidean

# Revisiting CNNs in the spatial domain: Patch operators

- Recall the definition of convolution on a 2D grid:

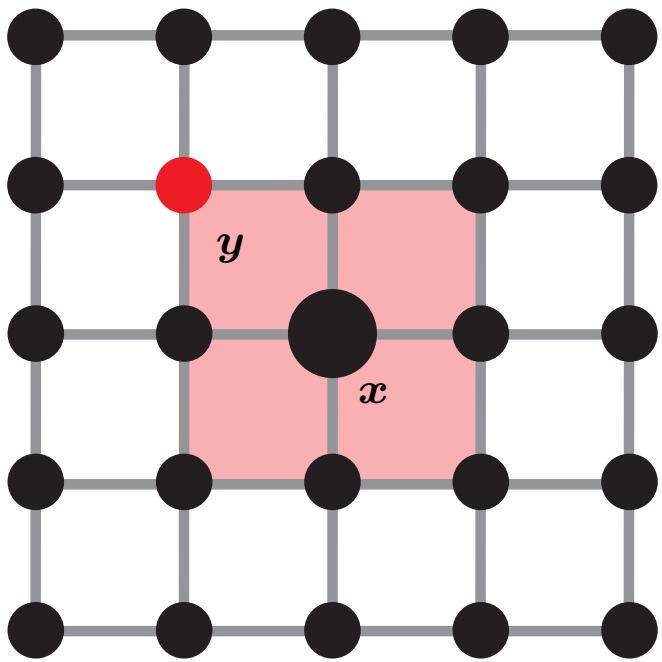
$$(f \star g)(\boldsymbol{x}) \quad = \sum_{\boldsymbol{x'} \in supp(g)} g(\boldsymbol{x'}) f(\boldsymbol{x} - \boldsymbol{x'})$$



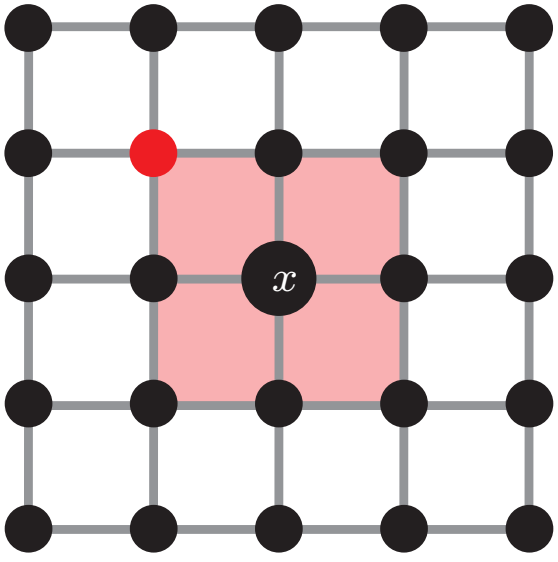
# Revisiting CNNs in the spatial domain: Patch operators

- Patch Operator: This amount to mapping each filter parameter $g(\boldsymbol{x'})$ to one value of the function $f(\boldsymbol{y})$: $(\mathcal{D}(\boldsymbol{x})f)(\boldsymbol{x'}) = f(\boldsymbol{y})$

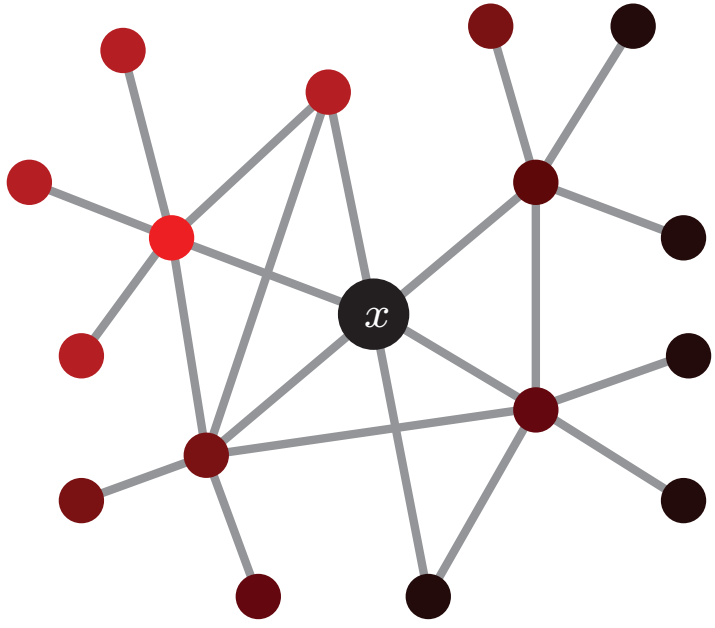
$$(f \star g)(\boldsymbol{x}) \quad = \sum_{\boldsymbol{x'} \in supp(g)} g(\boldsymbol{x'})(\mathcal{D}(\boldsymbol{x})f)(\boldsymbol{x'})$$



# Patch operator on Non-euclidean domains




$w_1(x, \cdot)$ on a grid

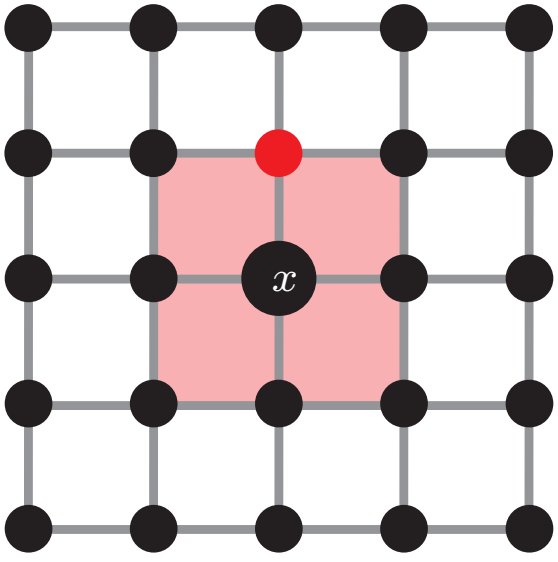



$w_1(x, \cdot)$ on a graph

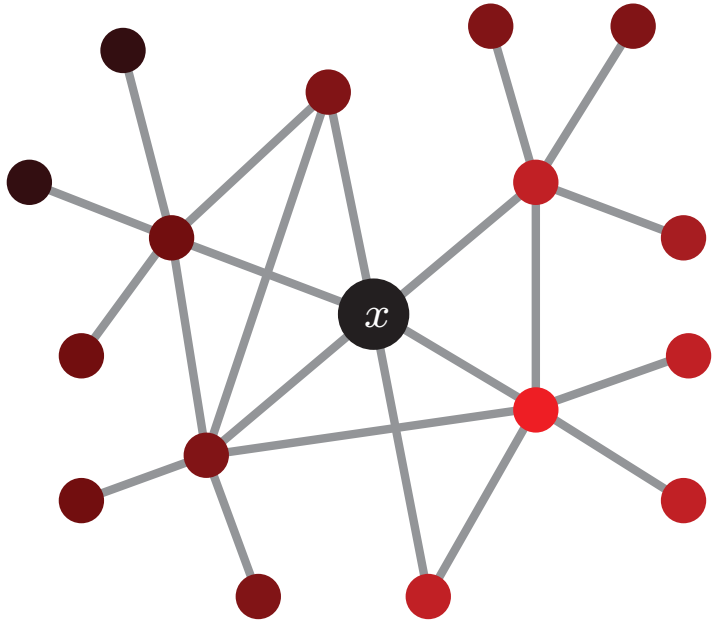
Instead of having a "1-1" mapping between patches and filter parameters, define $K$ "generalized patches" as follows:

- Define a weighting function for each patch $w_k(x, y)$, assigning weights to a pair of vertices $x, y$.

# Patch operator on Non-euclidean domains




$w_2(x, \cdot)$ on a grid




$w_2(x, \cdot)$ on a graph

Instead of having a "1-1" mapping between patches and filter parameters, define $K$ "generalized patches" as follows:

- Define a weighting function for each patch $w_k(x, y)$, assigning weights to a pair of vertices $x, y$.
- $D_k(x)(f) = \sum_{y \in \mathcal{N}(x)} w_k(x, y) f(y)$

Masci*, Boscaini*, et al., ICCVW-3DRR 2015

# Patch Operator based convolution

Patch Operator based convolution of $f \in L^2(\mathcal{X})$ with discrete filter $g = (g_1, \ldots, g_K)$

$$(f \star g)(x) \quad = \quad \sum_{k=1}^{K} g_k \mathcal{D}_k(x) f$$

Matrix-vector notation

$$\mathbf{f} \star \mathbf{g} = \mathbf{g}^\top (\mathcal{D}\mathbf{f})$$

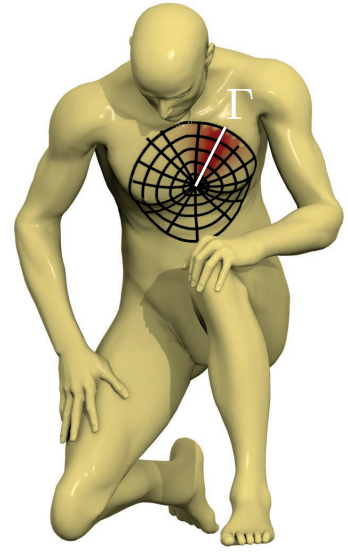
where $\mathbf{g} = (g_1, \ldots, g_K)^\top$ is the filter and $\mathcal{D}\mathbf{f}$ is an $n \times K$ matrix containing patches evaluated at each point as rows.

# Geodesic CNN

$$(\mathcal{D}(x)f)(\rho,\theta) \quad = \quad \int_{\mathcal{X}} \underbrace{w_\rho(x,x')w_\theta(x,x')}_{w_{\rho\theta}(x,x')} f(x')\,dx'$$


Radial weight

$$w_\rho(x,x') \propto e^{-(d_{\mathcal{X}}(x,x')-\rho)^2/\sigma_\rho^2}$$




Angular weight

$$w_\theta(x,x') \propto e^{-d^2_{\mathcal{X}}(\Gamma_\theta(x),x')/\sigma_\theta^2}$$

Kokkinos et al., CVPR 2012

# Geodesic CNN (GCNN)

Convolutional layer expressed in the spatial domain using geodesic polar patch operator + angular max pooling to solve rotational ambiguity

$$\boldsymbol{F'}(x) = \max_{\Delta\theta} \; \xi\left( \sum_{\rho,\theta} (\mathcal{D}(x)\boldsymbol{F})(\rho,\theta) \; \boldsymbol{G}(\rho, \theta + \Delta\theta) \right)$$

$\boldsymbol{G}(\rho,\theta) \in \mathbb{R}^{d_{in} \times d_{out}}$ the learnable parameters for each patch

# Geodesic CNN (GCNN)

Convolutional layer expressed in the spatial domain using geodesic polar patch operator + angular max pooling to solve rotational ambiguity

$$\boldsymbol{F'}(x) = \max_{\Delta\theta} \; \xi\left( \sum_{\rho,\theta} (\mathcal{D}(x)\boldsymbol{F})(\rho,\theta) \; \boldsymbol{G}(\rho, \theta + \Delta\theta) \right)$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity

Masci*, Boscaini*, et al., ICCVW-3DRR 2015

# Geodesic CNN (GCNN)

Convolutional layer expressed in the spatial domain using geodesic polar patch operator + angular max pooling to solve rotational ambiguity

$$\boldsymbol{F'}(x) = \max_{\Delta\theta} \; \xi\left( \sum_{\rho,\theta} (\mathcal{D}(x)\boldsymbol{F})(\rho,\theta) \; \boldsymbol{G}(\rho, \theta + \Delta\theta) \right)$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent (patches defined locally on the continuous domain)

Masci*, Boscaini*, et al., ICCVW-3DRR 2015

# Geodesic CNN (GCNN)

Convolutional layer expressed in the spatial domain using geodesic polar patch operator + angular max pooling to solve rotational ambiguity

$$\boldsymbol{F'}(x) = \max_{\Delta\theta} \; \xi\left( \sum_{\rho,\theta} (\mathcal{D}(x)\boldsymbol{F})(\rho,\theta) \; \boldsymbol{G}(\rho, \theta + \Delta\theta) \right)$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent (patches defined locally on the continuous domain)
- ☹ Expensive pre-computation of patches
- ☹ Handcrafted patch operators - applicable only on shapes
- ☹ Rotation ambiguity (faced by angular max-pooling here)

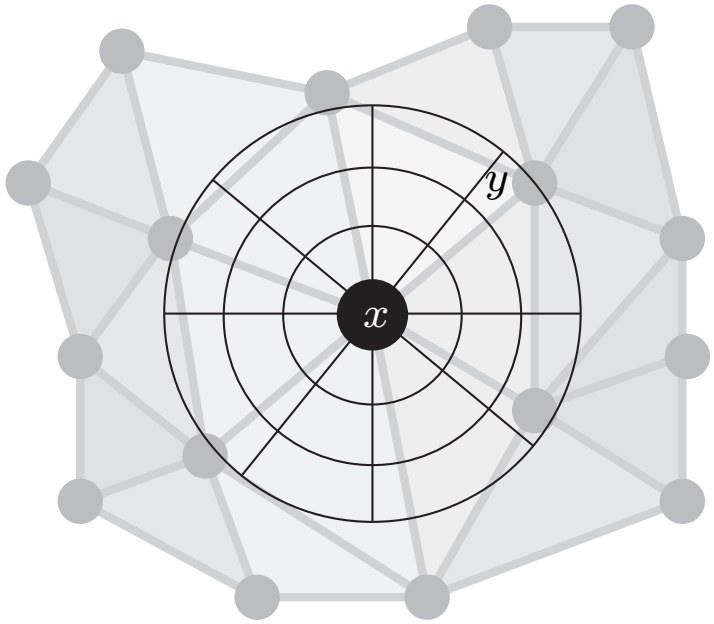
Masci*, Boscaini*, et al., ICCVW-3DRR 2015

# Mixture Model Networks: Learnable patch operator

- Define Local system of coordinates $\mathbf{u}(x, y)$ around $x$ (e.g. geodesic polar)

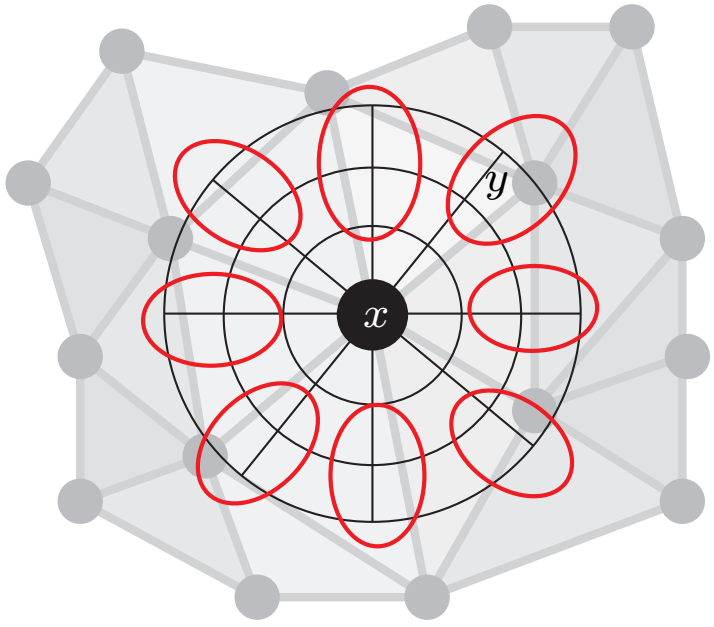
- Learnable weights:
  $w_1(\mathbf{u}), \dots, w_K(\mathbf{u})$ functions of $\mathbf{u}$, e.g. Gaussians

  $$w_k = \exp\left(-(\mathbf{u} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{u} - \boldsymbol{\mu}_k)\right)$$




Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks: Learnable patch operator




- Define Local system of coordinates $\mathbf{u}(x, y)$ around $x$ (e.g. geodesic polar)

- Learnable weights: $w_1(\mathbf{u}), \dots, w_K(\mathbf{u})$ functions of $\mathbf{u}$, e.g. Gaussians

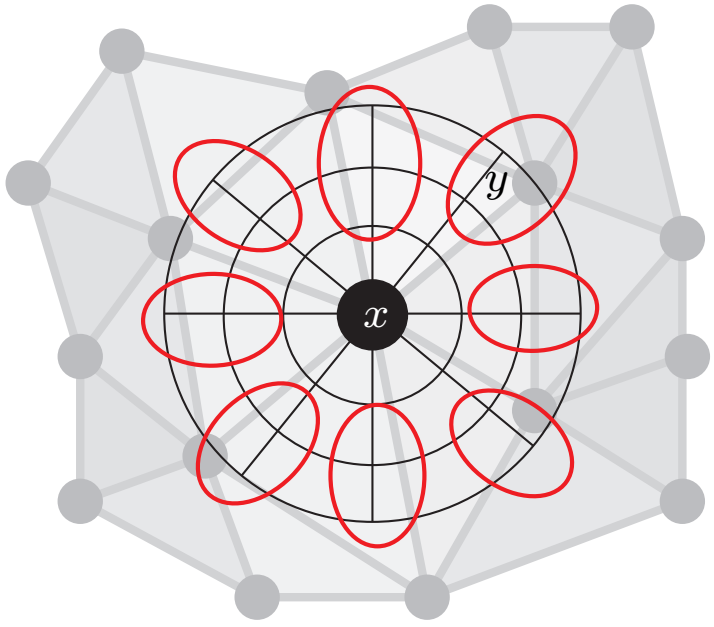
$$w_k = \exp\left(-(\mathbf{u} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{u} - \boldsymbol{\mu}_k)\right)$$

- Now the convolution with filter $g$ becomes:

$$(f \star g)(x) = \sum_{k=1}^{K} g_k \sum_{y \in \mathcal{N}(x)} w_k(\mathbf{u}(x, y)) f(y)$$

Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks: Learnable patch operator



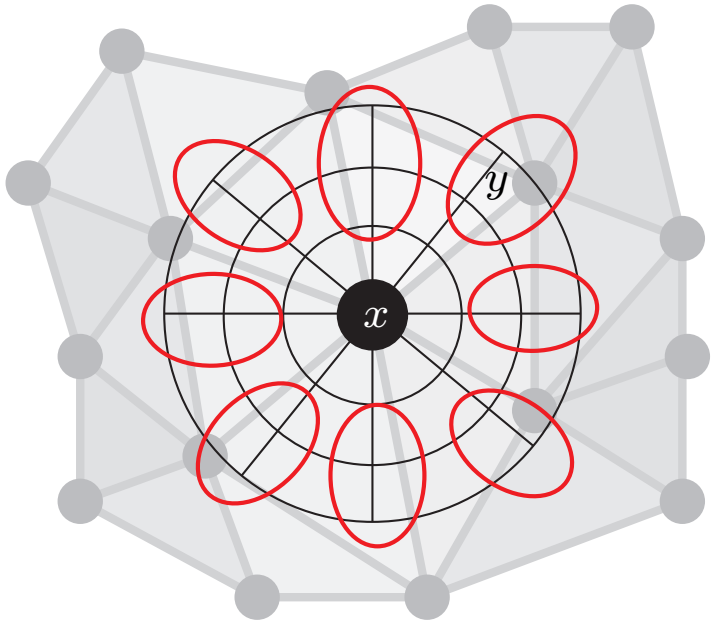

- Define Local system of coordinates $\mathbf{u}(x, y)$ around $x$ (e.g. geodesic polar)

- Learnable weights:
  $w_1(\mathbf{u}), \dots, w_K(\mathbf{u})$ functions of $\mathbf{u}$,
  e.g. Gaussians
  $$w_k = \exp\left(-(\mathbf{u} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{u} - \boldsymbol{\mu}_k)\right)$$

- Now the convolution with filter $g$ becomes:
  $$(f \star g)(x) = \sum_{k=1}^{K} g_k \underbrace{\sum_{y \in \mathcal{N}(x)} w_k(\mathbf{u}(x, y)) f(y)}_{\text{patch operator}}$$

Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks: Learnable patch operator




- Define Local system of coordinates $\mathbf{u}(x, y)$ around $x$ (e.g. geodesic polar)

- Learnable weights: $w_1(\mathbf{u}), \dots, w_K(\mathbf{u})$ functions of $\mathbf{u}$, e.g. Gaussians

$$w_k = \exp\left(-(\mathbf{u} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{u} - \boldsymbol{\mu}_k)\right)$$

- Now the convolution with filter $g$ becomes:

$$(f \star g)(x) = \sum_{y \in \mathcal{N}(x)} \underbrace{\sum_{k=1}^{K} g_k w_{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k}(\mathbf{u}(x, y))}_{\text{Gaussian mixture}} f(y)$$

Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks (MoNet)

Convolutional layer expressed in the spatial domain

$$\boldsymbol{F'}(x) = \xi\left(\sum_{y\in\mathcal{N}(x)}\sum_{k=1}^{K} w_k(\mathbf{u}(x,y))\boldsymbol{F}(y)\boldsymbol{G}_k\right), \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity

Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks (MoNet)

Convolutional layer expressed in the spatial domain

$$\boldsymbol{F'}(x) = \xi\left(\sum_{y\in\mathcal{N}(x)}\sum_{k=1}^{K} w_k(\mathbf{u}(x,y))\boldsymbol{F}(y)\boldsymbol{G}_k\right), \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent (patches defined locally on the continuous domain)

Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks (MoNet)

Convolutional layer expressed in the spatial domain

$$\boldsymbol{F'}(x) = \xi\left(\sum_{y\in\mathcal{N}(x)}\sum_{k=1}^{K} w_k(\mathbf{u}(x,y))\boldsymbol{F}(y)\boldsymbol{G}_k\right), \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent (patches defined locally on the continuous domain)
- ☺ Learnable patch operators - applicable on general graphs

Monti*, Boscaini*, et al., CVPR 2017

# Mixture Model Networks (MoNet)

Convolutional layer expressed in the spatial domain

$$\boldsymbol{F'}(x) = \xi \left( \sum_{y \in \mathcal{N}(x)} \sum_{k=1}^{K} w_k(\mathbf{u}(x,y)) \boldsymbol{F}(y) \boldsymbol{G}_k \right) , \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in} \times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow \mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent (patches defined locally on the continuous domain)
- ☺ Learnable patch operators - applicable on general graphs
- ☹ Hancrafted pseudo-coordinates and expensive pre-computation
- ☹ Orientation ambiguity for meshes (if the pseudo-coordinates are defined on the input space)

Monti*, Boscaini*, et al., CVPR 2017

# MoNet as generalization of previous methods

| **Method** | **Coordinates** $u(x,y)$ | **Weight function** $w_{\mathbf{\Theta}}(\mathbf{u})$ |
|---|---|---|
| CNN[1] | $\mathbf{u}(x') - \mathbf{u}(x)$ | $\delta(\mathbf{u}-\mathbf{v})$<br>fixed parameters $\mathbf{\Theta} = \mathbf{v}$ |
| GCN[2] | $\deg(x), \deg(x')$ | $\left(1 - \lvert 1 - \frac{1}{\sqrt{u_1}}\rvert\right)\left(1 - \lvert 1 - \frac{1}{\sqrt{u_2}}\rvert\right)$ |
| GCNN[3] | $\rho(x,x'), \theta(x,x')$ | $\exp\left(-\frac{1}{2}(\mathbf{u}-\mathbf{v})^\top \begin{pmatrix} \sigma_\rho^2 & \\ & \sigma_\theta^2 \end{pmatrix}^{-1} (\mathbf{u}-\mathbf{v})\right)$<br>fixed parameters $\mathbf{\Theta} = (\mathbf{v}, \sigma_\rho, \sigma_\theta)$ |
| ACNN[4] | $\rho(x,x'), \theta(x,x')$ | $\exp\left(-t\mathbf{u}^\top \mathbf{R}_\varphi \begin{pmatrix} \alpha & \\ & 1 \end{pmatrix} \mathbf{R}_\varphi^\top \mathbf{u}\right)$<br>fixed parameters $\mathbf{\Theta} = (\alpha, \varphi, t)$ |
| MoNet[5] | $\rho(x,x'), \theta(x,x')$ | $\exp\left(-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{u}-\boldsymbol{\mu})\right)$<br>learnable parameters $\mathbf{\Theta} = (\boldsymbol{\mu}, \mathbf{\Sigma})$ |

Some CNN models can be considered as particular settings of MoNet with weighting functions of different form

Methods: [1]LeCun et al. 1998; [2]Kipf, Welling 2016; [3]Masci et al. 2015; [4]Boscaini et al. 2016; [5]Monti et al. 2016

# Learn the Patch Operator in the feature space

- Engineering the pseudo-coordinates requires domain knowledge.

# Learn the Patch Operator in the feature space

- Engineering the pseudo-coordinates requires domain knowledge.
- Solution: Learn the weighting functions $w_k$ directly from the feature space:

$$w_k(x, y) = w_k(\boldsymbol{F}(x), \boldsymbol{F}(y))$$

Verma et al., CVPR 2018, Veličković et al., ICLR 2018

# Learn the Patch Operator in the feature space

- Engineering the pseudo-coordinates requires domain knowledge.
- Solution: Learn the weighting functions $w_k$ directly from the feature space:

$$w_k(x, y) = w_k(\boldsymbol{F}(x), \boldsymbol{F}(y))$$

- FeastNet: $w_k(x, y) = softmax_{k \in \{0 \dots K\}}(\boldsymbol{a_k}^T [\boldsymbol{F}(x) || \boldsymbol{F}(y)] + c_k)$

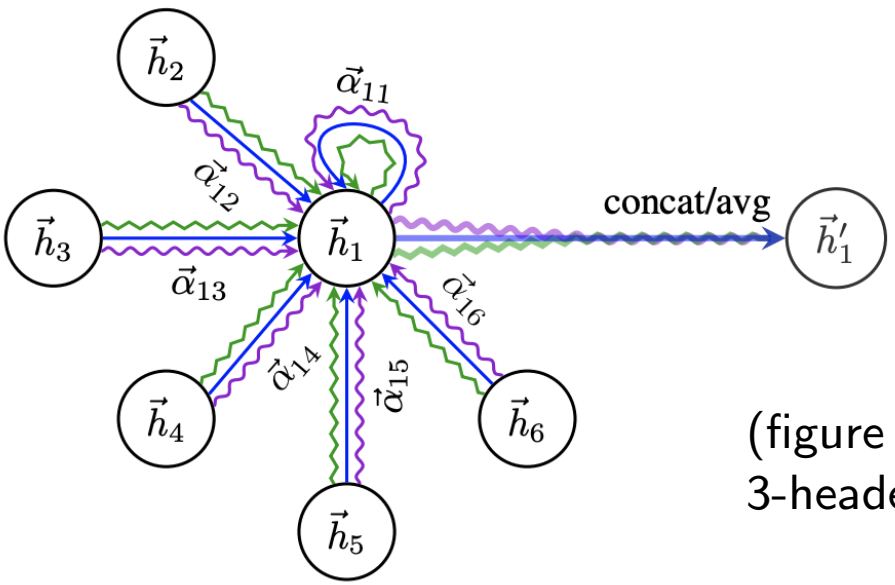
Verma et al., CVPR 2018, Veličković et al., ICLR 2018

# Learn the Patch Operator in the feature space

- Engineering the pseudo-coordinates requires domain knowledge.
- Solution: Learn the weighting functions $w_k$ directly from the feature space:

$$w_k(x, y) = w_k(\boldsymbol{F}(x), \boldsymbol{F}(y))$$

- FeastNet: $w_k(x, y) = softmax_{k \in \{0 \dots K\}}(\boldsymbol{a_k}^T[\boldsymbol{F}(x)||\boldsymbol{F}(y)] + c_k)$
- Graph Attention:
  $w_k(x, y) = softmax_{y \in \mathcal{N}(x)}(LeakyReLU(\boldsymbol{a_k}^T[\boldsymbol{G}_k\boldsymbol{F}(x)||\boldsymbol{G}_k\boldsymbol{F}(y)]))$




(figure by Petar Veličković:
3-headed graph attention mechanism)

Verma et al., CVPR 2018, Veličković et al., ICLR 2018

# Feature-Steered Graph Convolutions (FeastNet)

FeastNet Convolutional layer

$$\boldsymbol{F'}(x) = \xi \left( \sum_{y \in \mathcal{N}(x)} \sum_{k=1}^{K} w_k(\boldsymbol{F}(x), \boldsymbol{F}(y)) \; \boldsymbol{F}(y) \boldsymbol{G}_k \right) , \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in} \times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent

Verma et al., CVPR 2018

# Feature-Steered Graph Convolutions (FeastNet)

FeastNet Convolutional layer

$$\boldsymbol{F'}(x) = \xi \left( \sum_{y \in \mathcal{N}(x)} \sum_{k=1}^{K} w_k(\boldsymbol{F}(x), \boldsymbol{F}(y))\ \boldsymbol{F}(y) \boldsymbol{G}_k \right) , \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in} \times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow \mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent
- ☺ Learnable weight functions - no engineering needed

Verma et al., CVPR 2018

# Feature-Steered Graph Convolutions (FeastNet)

FeastNet Convolutional layer

$$\boldsymbol{F'}(x) = \xi \left( \sum_{y \in \mathcal{N}(x)} \sum_{k=1}^{K} w_k(\boldsymbol{F}(x), \boldsymbol{F}(y))\ \boldsymbol{F}(y)\boldsymbol{G}_k \right) , \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in} \times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent
- ☺ Learnable weight functions - no engineering needed
- ☹ No guarantee that the weighting functions will have a small support $\Rightarrow$ possibly all the vertices might contribute in the calculation of a patch $k$ $\Rightarrow$ Harder to optimize.
- ☹ Geometry agnostic - potentially sensitive to remeshing

Verma et al., CVPR 2018

# Graph Attention Networks (GAT)

GAT layer

$$\boldsymbol{F'}(x) = \xi\left(\Big\|_{k=1}^{K} \sum_{y\in\mathcal{N}(x)} w_k(\boldsymbol{F}(x),\boldsymbol{F}(y))\ \boldsymbol{F}(y)\boldsymbol{G}_k\right),\quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Domain independent
- ☺ Learnable weight functions - no engineering needed
- ☺ Easier to optimize due to the concatenation of the patches and the softmax normalization of the weights across each neighborhood
- ☹ Geometry agnostic - potentially sensitive to remeshing

Veličković et al., ICLR 2018

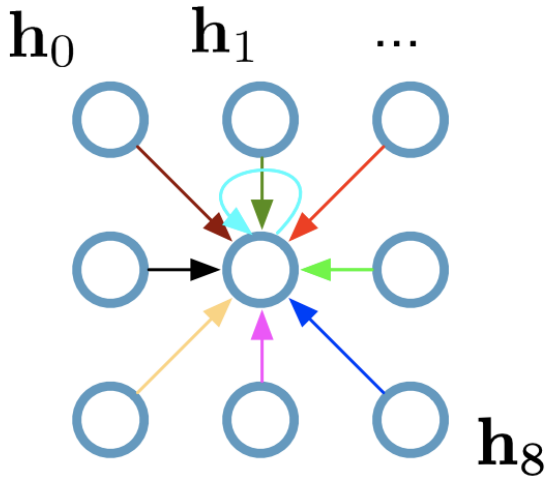
# Different Perspectives: Graph Convolutions vs Message Passing

- Graph Convolutions

Spectral

Patch Operator based (aka Spatial)

- **Message Passing**

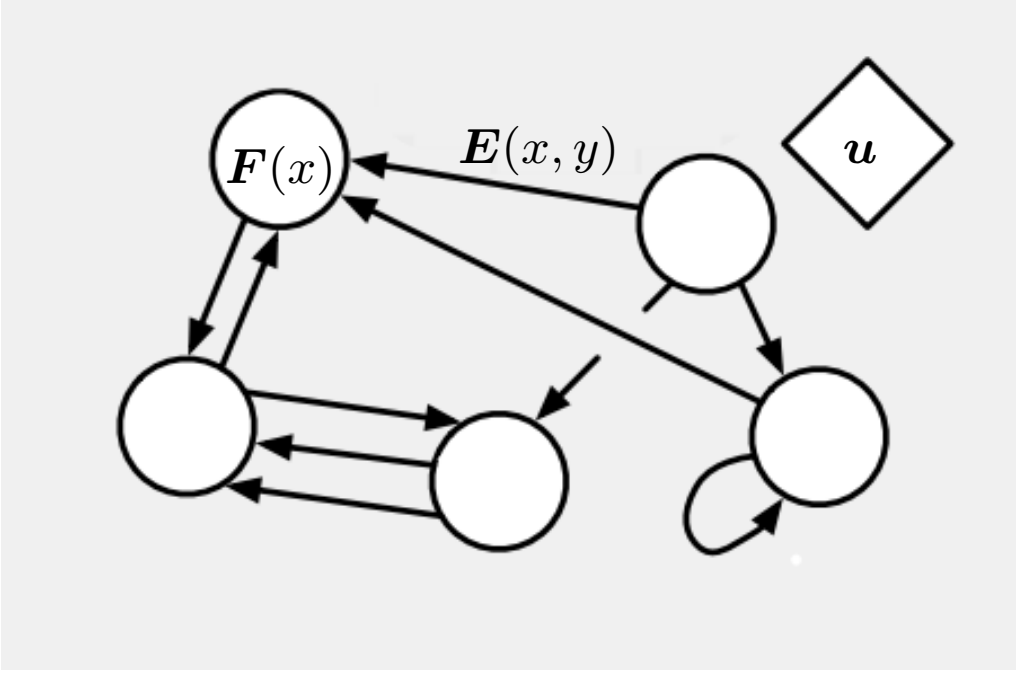


# The Message Passing paradigm

- Node features are learned by exchanging information with neighbouring nodes (originally introduced in the first Graph Neural Network, Scarselli, Gori et al. 2009)
- Draws inspiration from traditional diffusion processes in graphs (e.g. random walks)
- The message passing operation is repeated for a certain amount of steps $\Rightarrow$ similarly to Recurrent Neural Networks (RNNs)
- **By unrolling the time-steps, this framework is equivalent to the feed-forward networks described so far**.

# Graph Network

- Most general framework so far.
- Let $\mathcal{G}$ a graph with vertex signals $\boldsymbol{F}(x)$ and edge signals $\boldsymbol{E}(x, y)$ and $\boldsymbol{u}$ a global signal associated




Battaglia et al., arxiv 2018

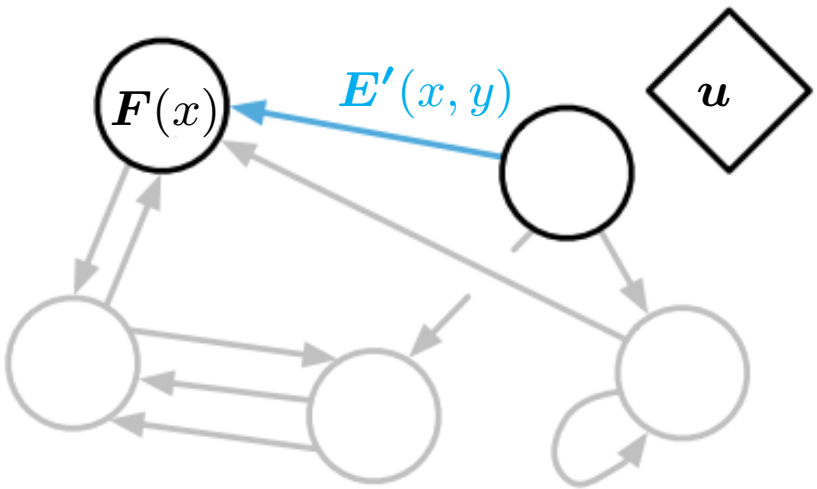
# Graph Network algorithm

1. The edge attributes are updated by the edge update function $\boldsymbol{\phi}^{\mathcal{E}}(\cdot)$.

$$\boldsymbol{\acute{E}}(x,y) = \boldsymbol{\phi}^{\mathcal{E}}(\boldsymbol{E}(x,y), \boldsymbol{F}(x), \boldsymbol{F}(y), \boldsymbol{u})$$

In MoNet:

$$\boldsymbol{\acute{E}}(x,y) = \sum_{k=1}^{K} w_k(\boldsymbol{u}(x,y))\boldsymbol{F}(y)\boldsymbol{G_k}$$



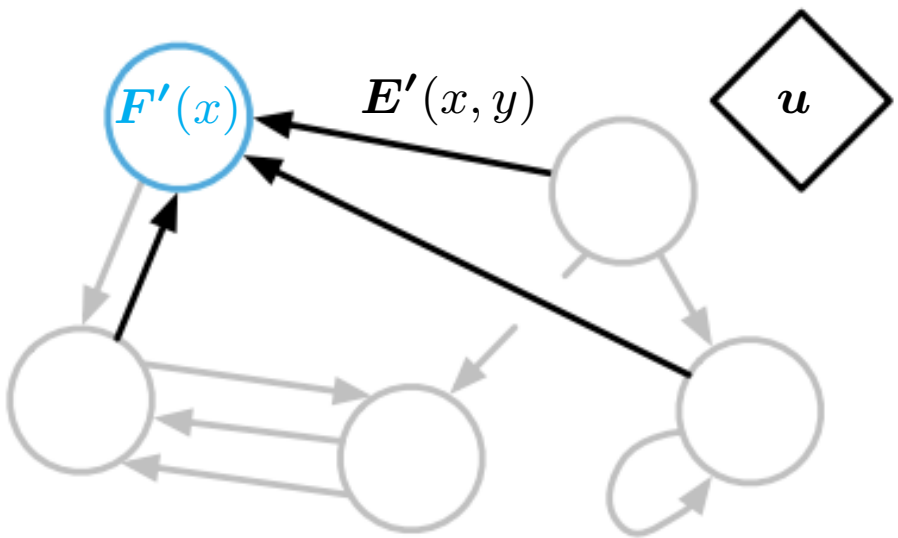

Battaglia et al., arxiv 2018

# Graph Network algorithm

2. The edge attributes of all the edges associated with a vertex are aggregated by a permutation invariant aggregation function.

$$\bar{\boldsymbol{E}}(x) = \boldsymbol{\rho}^{\mathcal{E}\to\mathcal{V}}(\{\acute{\boldsymbol{E}}(x,y)\}_{\{y\in\mathcal{N}(x)\}}) \tag{1}$$

In MoNet: $\bar{\boldsymbol{E}}(x) = \sum_{y\in\mathcal{N}(x)} \acute{\boldsymbol{E}}(x,y)$




3. The node attributes are updated by the node update function $\boldsymbol{\phi}^{\mathcal{V}}(\cdot)$.

$$\acute{\boldsymbol{F}}(x) = \boldsymbol{\phi}^{\mathcal{V}}(\bar{\boldsymbol{E}}(x), \boldsymbol{F}(x), \boldsymbol{u}) \tag{2}$$

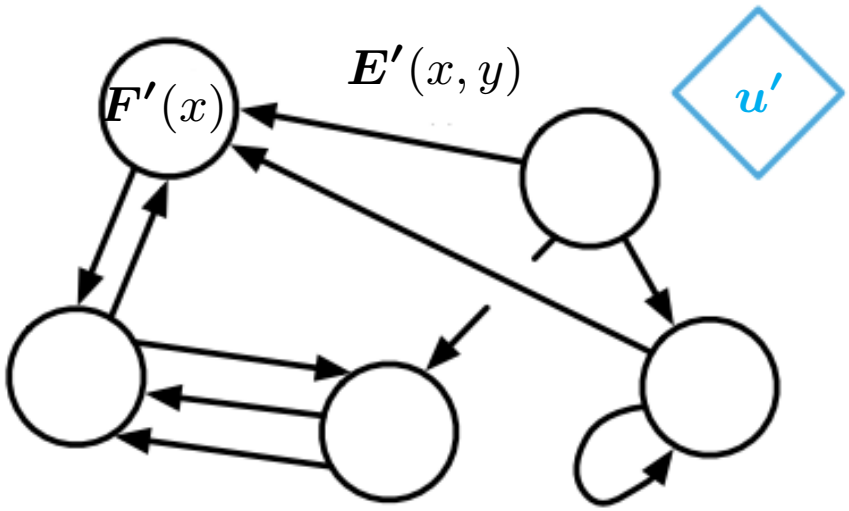
In MoNet: $\acute{\boldsymbol{F}}(x) = \xi(\bar{\boldsymbol{E}}(x))$

Battaglia et al., arxiv 2018

# Graph Network algorithm

4. All the edge attributes and all the node attributes are aggregated by a pair of permutation invariant aggregation functions.

$$\hat{\boldsymbol{E}} = \boldsymbol{\rho}^{\mathcal{E}\to\boldsymbol{u}}(\{\acute{\boldsymbol{E}}(x,y)\}_{\{(x,y)\in\mathcal{E}\}})$$
$$\hat{\boldsymbol{F}} = \boldsymbol{\rho}^{\mathcal{V}\to\boldsymbol{u}}(\{\acute{\boldsymbol{F}}(x)\}_{\{x\in\mathcal{V}\}})$$

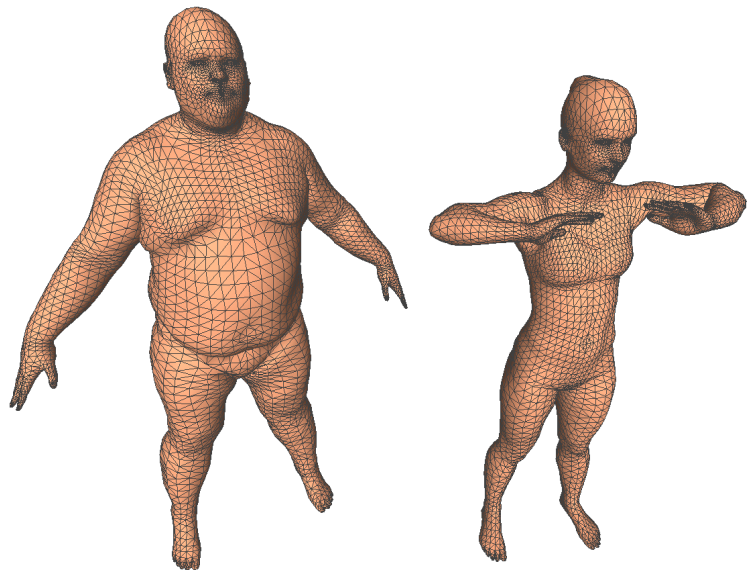



5. Finally the global attribute is updated.

$$\acute{\boldsymbol{u}} = \boldsymbol{\phi}^{\boldsymbol{u}}(\hat{\boldsymbol{E}}, \hat{\boldsymbol{V}}, \boldsymbol{u}) \tag{3}$$

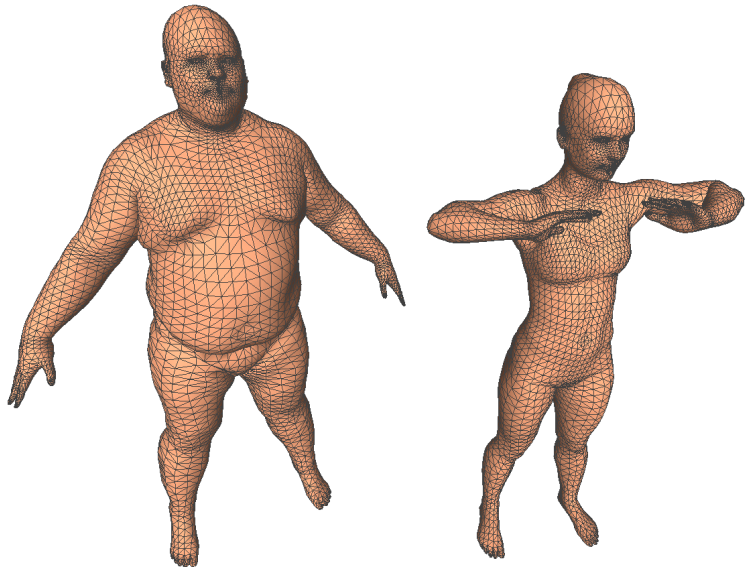
Battaglia et al., arxiv 2018

# Inductive bias of the domain: Fixed connectivity



- How does existing work model the connectivity of the graph?

# Inductive bias of the domain: Fixed connectivity



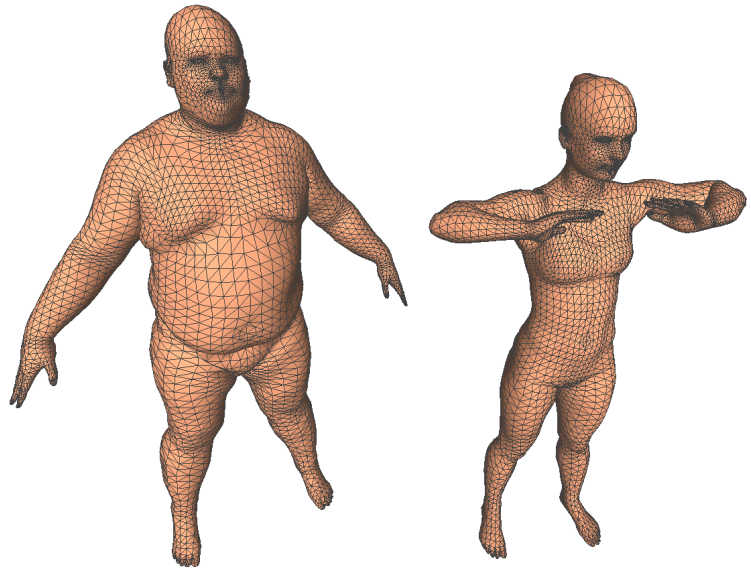
- How does existing work model the connectivity of the graph?
- Spectral methods: Through the graph Laplacian (or its eigen-decomposition)

$$\mathbf{f} \star \mathbf{g} = \sum_{k=0}^{r} a_k \mathbf{\Delta}^k \mathbf{f}$$

☺ Different signal values on the same node always undergo the same transformation.

☹ Isotropic Kernels

# Inductive bias of the domain: Fixed connectivity



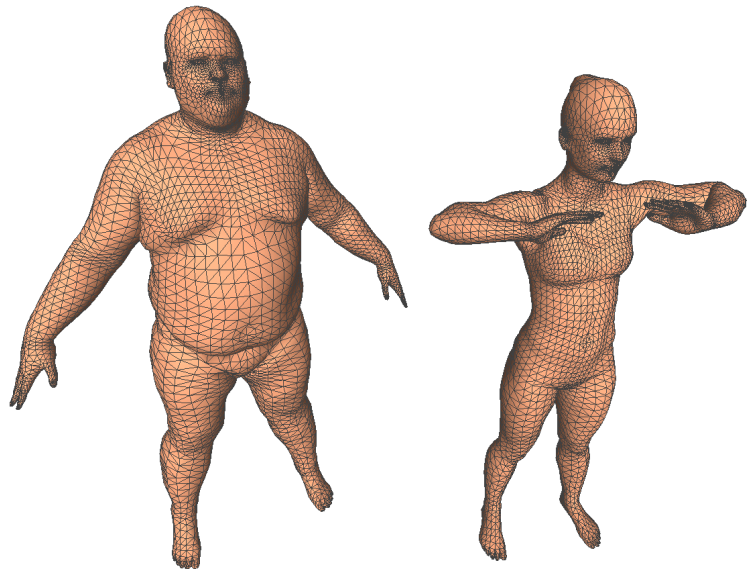
- How does existing work model the connectivity of the graph?
- Patch-based:

$$(f \star g)(x) = \sum_{y \in \mathcal{N}(x)} \sum_{k=1}^{K} g_k w_k(\mathbf{u}(x,y)) f(y)$$

☺ Anisotropic Kernels

☹ Connectivity prior not explicitly modelled: Different signals values on the same node undergo different transformations (depending on the signal values themselves)

# Inductive bias of the domain: Fixed connectivity



- Can we have anisotropic kernels and explicit modelling of the connectivity at the same time?
- Reformulate the patch operator to depend only on the connectivity.
- "Hard" assignments between nodes and parameters (easier to optimise).

# Ordering-Based Graph Convolutions

- **Solution: locally order the vertices**!

# Ordering-Based Graph Convolutions

- **Solution: locally order the vertices**!
- For kernels equal to the maximum number of neighbours $K = max(|\mathcal{N}(x)|)$:

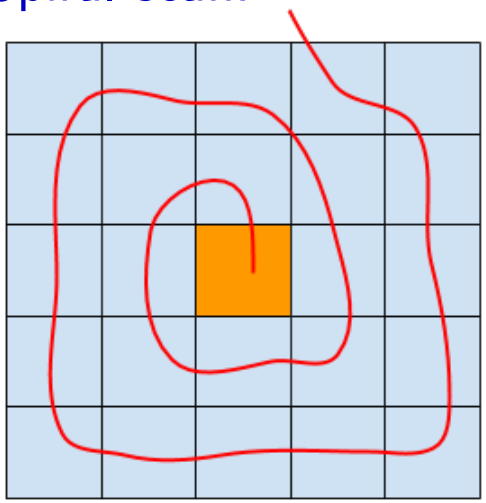
$$(f \star g)(\boldsymbol{x}) = \sum_{k=1}^{|\mathcal{N}(x)|} g_k f(x_k).$$

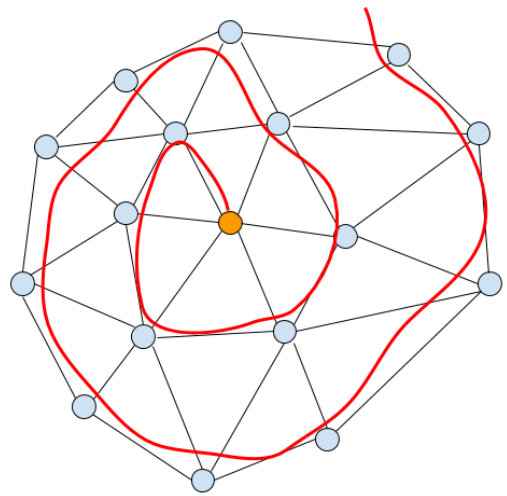
  where $\mathcal{N}(x) = \{x_1, \ldots, x_{|\mathcal{N}(x)|}\}$ the neighbourhood of $x$ (inc. $x$) ordered in some fixed way.
- The above formulation is equivalent with traditional convolution, after choosing a consistent ordering.

Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

# How to define the local ordering: Spiral Convolutions

- How can we make sure that the ordering is also consistent across different vertices of the graph?
- Harder problem for general graphs. For meshes, the ambiguity falls again into different rotations.
- Spiral scan:





Lim et al., ECCVW 2018, Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

# Ordering-Based Graph Convolutions

$$\boldsymbol{F'}(x) = \xi\left(\sum_{k=1}^{|\mathcal{N}(x)|} \boldsymbol{F}(x_k)\boldsymbol{G}_k\right), \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels

Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

# Ordering-Based Graph Convolutions

$$\boldsymbol{F'}(x) = \xi\left(\sum_{k=1}^{|\mathcal{N}(x)|} \boldsymbol{F}(x_k)\boldsymbol{G}_k\right), \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Lightweight, fast & easier to optimise
- ☺ Similar to traditional convolutions $\Rightarrow$ practices for traditional CNNs can be directly transferred (e.g. dilated convolutions)
- ☺ Connectivity and geometry aware

Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

# Ordering-Based Graph Convolutions

$$\boldsymbol{F'}(x) = \xi\left(\sum_{k=1}^{|\mathcal{N}(x)|} \boldsymbol{F}(x_k)\boldsymbol{G}_k\right), \quad \boldsymbol{G}_k \in \mathbb{R}^{d_{in}\times d_{out}}$$

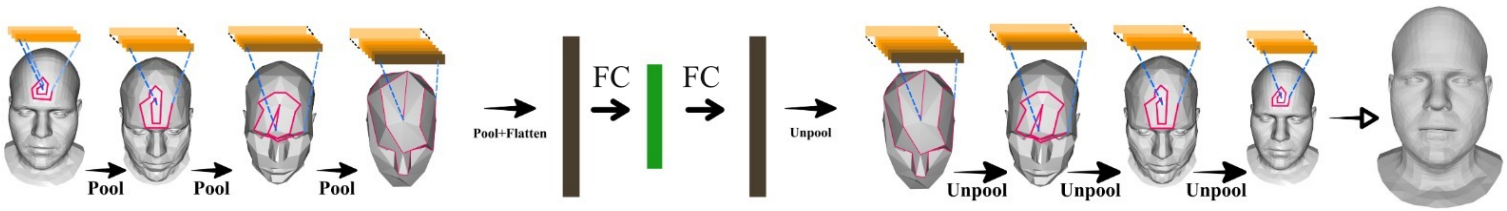
- ☺ Spatially-localized filters
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ All operations are local $\Rightarrow$ $\mathcal{O}(n)$ computational complexity
- ☺ Anisotropic Kernels
- ☺ Lightweight, fast & easier to optimise
- ☺ Similar to traditional convolutions $\Rightarrow$ practices for traditional CNNs can be directly transferred (e.g. dilated convolutions)
- ☺ Connectivity and geometry aware
- ☹ Ordering needs to be engineered

Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

1. Introduction to Geometric Deep Learning
2. Graphs: Fundamentals
3. Graph Convolutions
   - Spectral Approaches
   - Patch-based approaches
4. Message Passing
5. Ordering-Based Graph Convolutions
6. Applications: Generative models for 3D shapes

# Representation Learning for 3D meshes of fixed topology

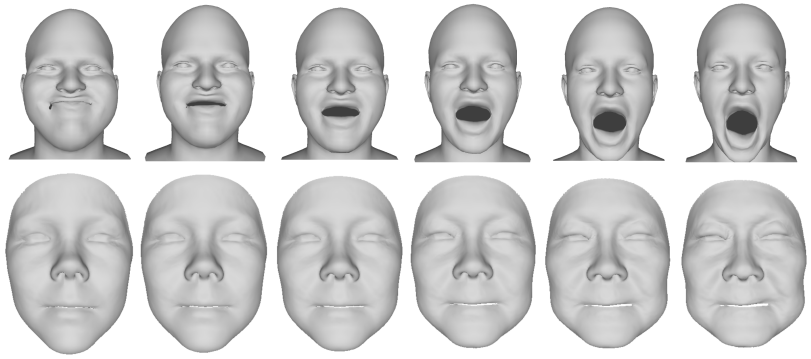
- Autoencoder architecture
- Spiral Convolutions
- Hierarchical structure



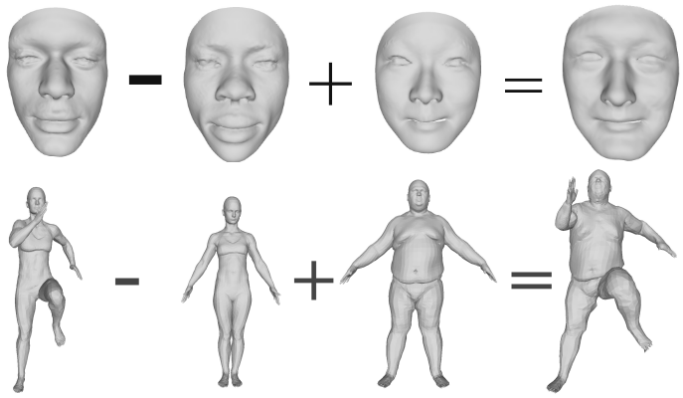

Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

# Vector space Arithmetics

- Interpolation



- Analogies




Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

# 3D shape Generation

- Wasserstein GAN architecture

Bouritsas*, Bokhnyak* et al., ICCV 2019, ICLRW 2019

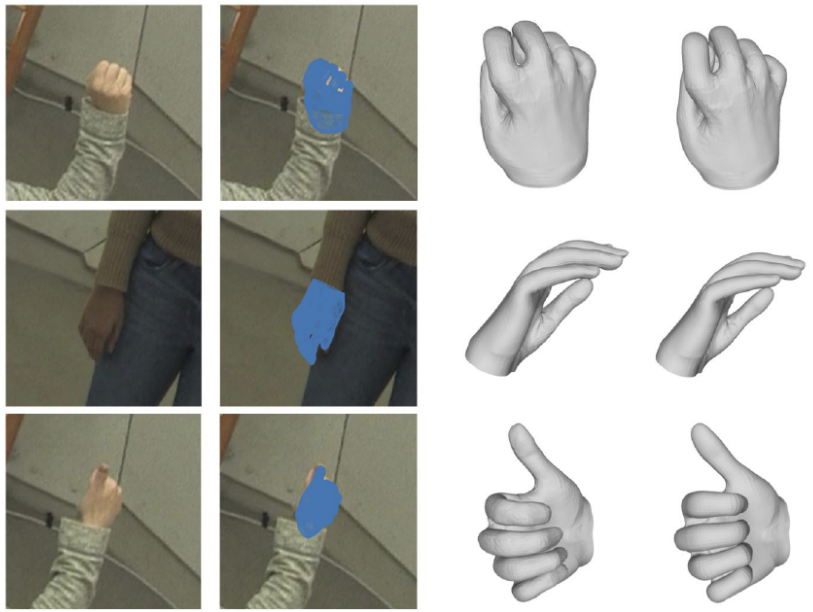
# Shape Completion (fixed toplogy)

Litany et al., CVPR 2018

# Facial expression Generation

Ranjan et al., ECCV 2018

# 3D Hand and Human Body Reconstruction




Kulon et al., BMVC 2019, Kolotouros et. al, CVPR 2019

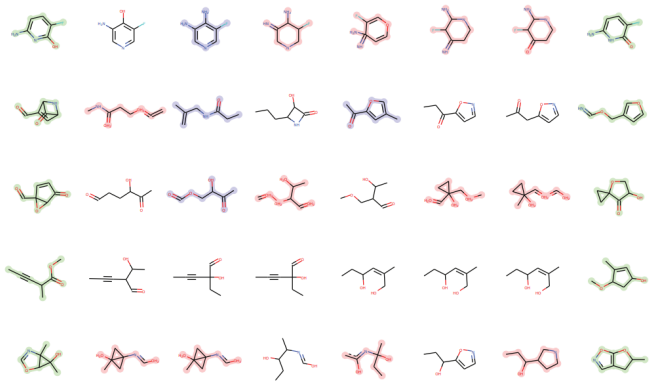
# Arbitrary topology 3D shape generation: Relatively unexplored

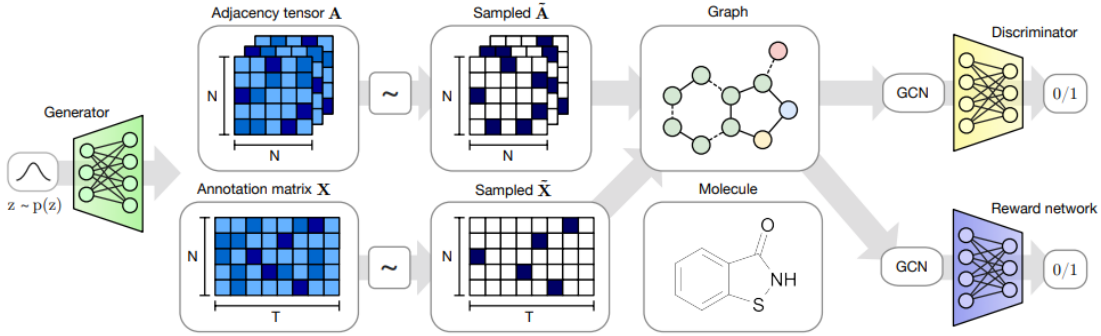

Range scan to 3D mesh (only up to 100 vertices)

Zero genus shape generation

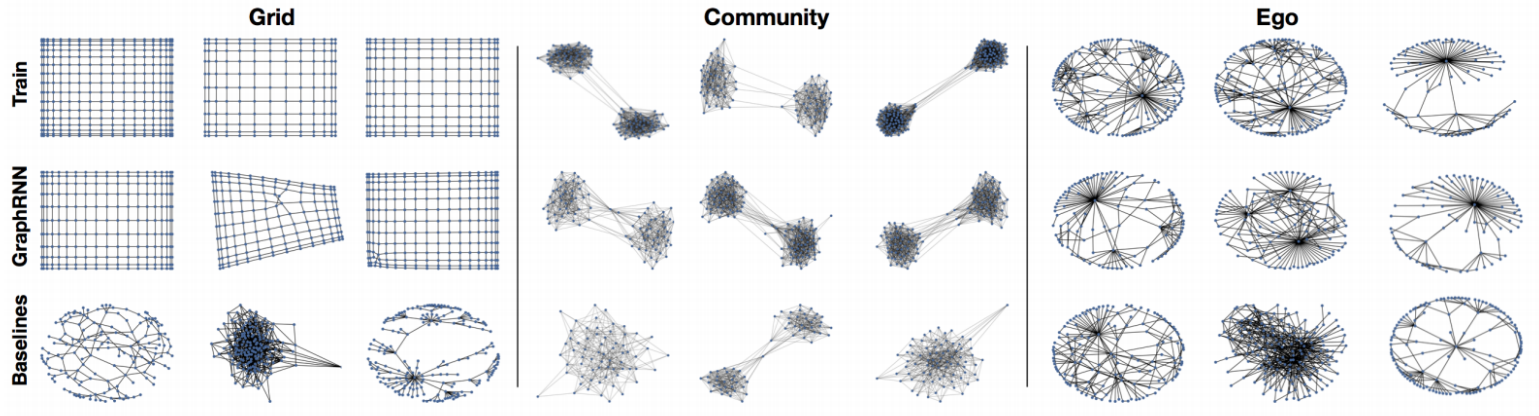
Dai and Nießner, CVPR 2019, Smith et al., ICML 2019

# Can we draw insipration from methods on arbitrary Graphs?





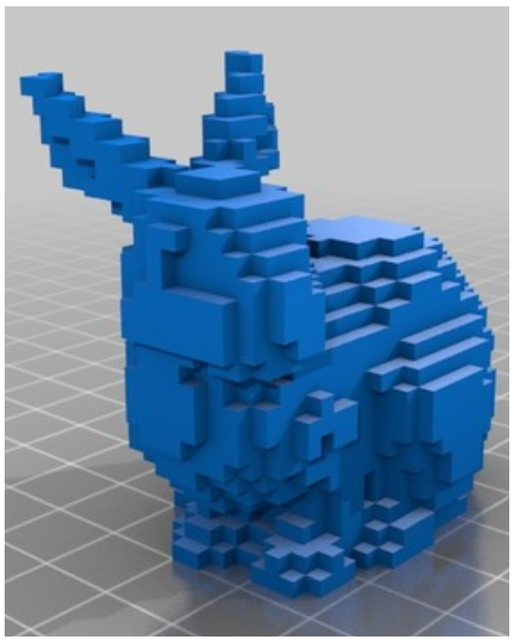

Molecule generation

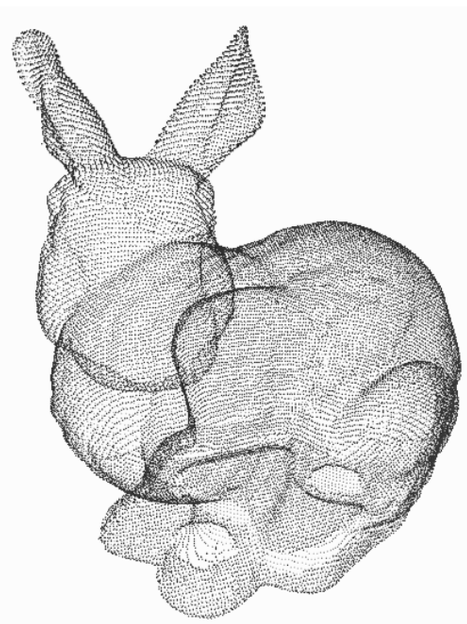


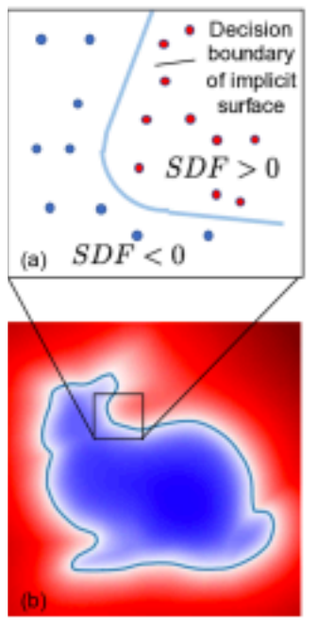

Topology Generation

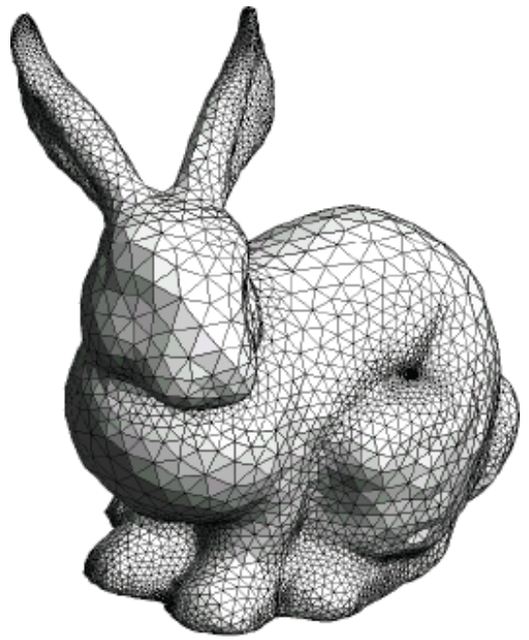
Simonovsky and Komodakis, ICANN 2019, De Cao and Kipf, ICMLW 2018, You et al., ICML 2018

# Summary: Deep Learning on 3D Data







| Image-based | Volumetric | Point Cloud | Signed distance-based | Mesh-based |
|---|---|---|---|---|
| ✅ Simple | ✅ Simple | ✅ Efficient | ✅ Efficient | ✅ Efficient |
| ✅ Good results | ⛔ Coarse | ✅ Simple | ✅ Simple | ✅ Accurate |
| ⛔ Not geometric | ⛔ Memory | ✅ Accurate | ✅ Accurate | ✅ Deform. invariant |
| ⛔ No invariance | ⛔ No invariance | ⛔ No invariance | ⛔ No invariance | ⛔ Custom layers |

Credit: Michael Bronstein, Charles R. Qi, Jeong J. Park

# Deep Learning Frameworks

# Datasets

# Additional 3D Deep Learning tutorials

# Deep Learning Frameworks

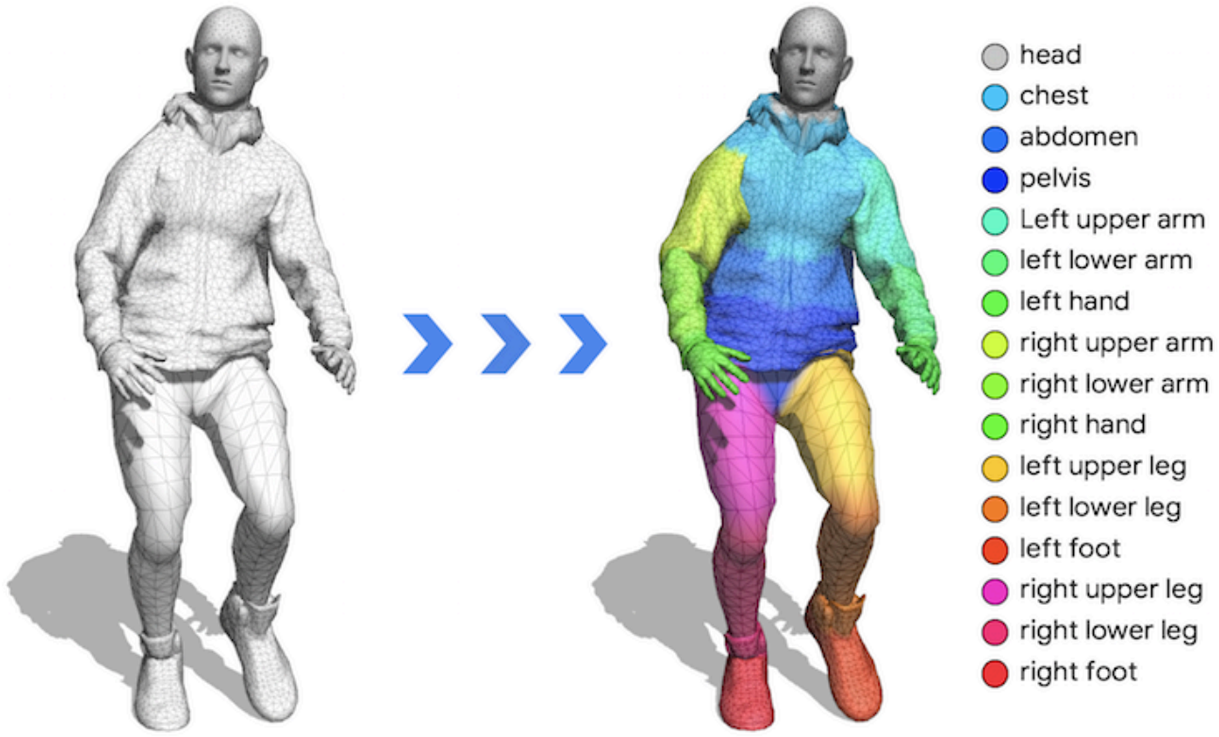
- Tensorflow (Google)
- PyTorch (Facebook), Torch
- Caffe (Berkeley) and Caffe v2 (merged with PyTorch)
- CNTK (Microsoft)
- MatConvNet (Oxford)
- Keras (high level Python API)
- Theano (University of Montreal)
- Matlab

# Deep Learning Frameworks

## Introducing TensorFlow Graphics: Computer Graphics Meets Deep Learning

TensorFlow in TensorFlow Follow

May 9 · 5 min read




https://medium.com/tensorflow/introducing-tensorflow-graphics-computer-graphics-meets-deep-learning-c8e3877b7668

# SGP 2019: Keynotes

**Yaron Lipman**
(Weizmann Institute of Science)

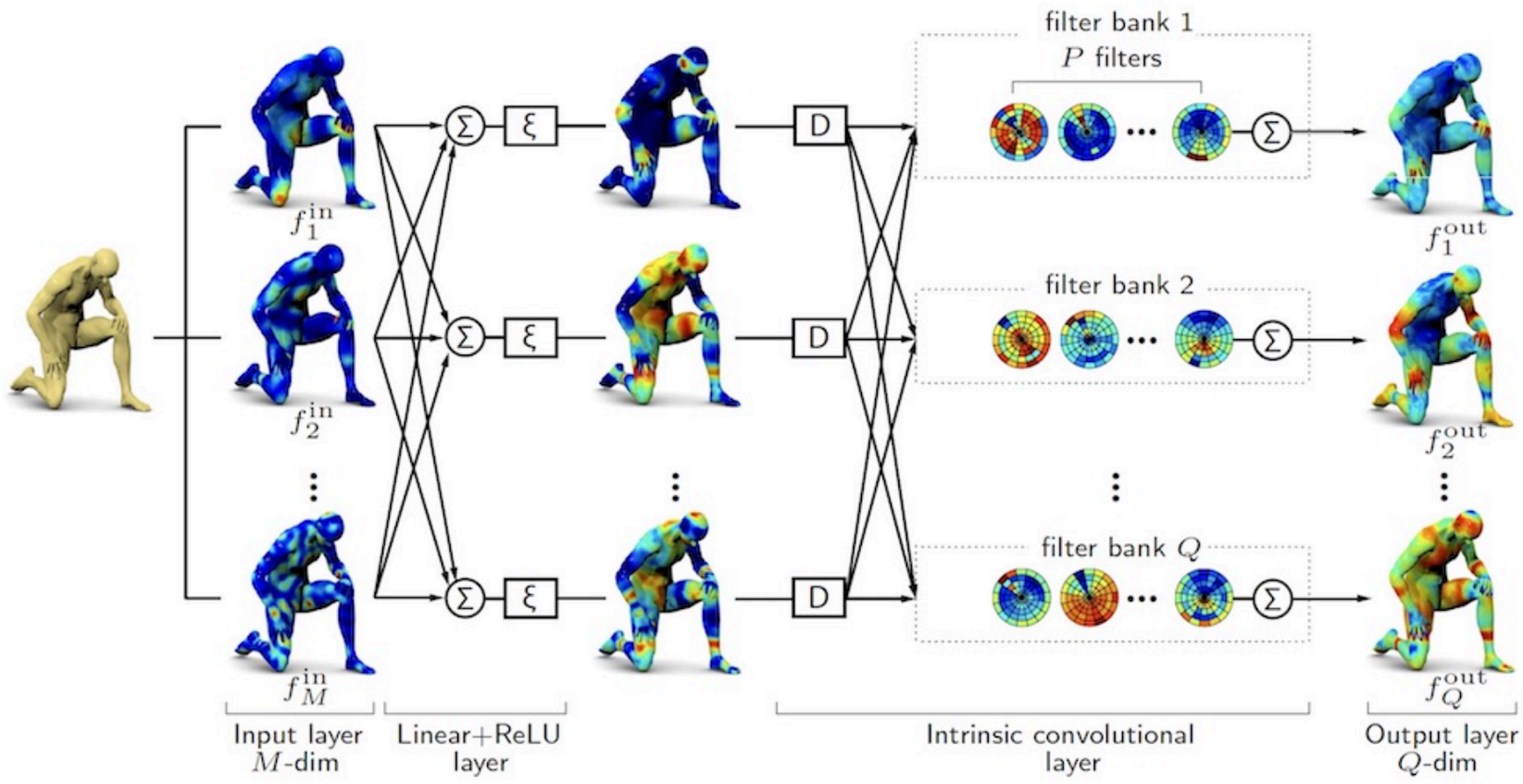
**Deep Learning Irregular Data**
Large part of the recent success of applying neural networks to image data is attributed to the restriction of the networks to translation-invariant functions without compromising their expressive power. In this talk we discuss how to adapt this basic paradigm of neural networks to irregular data including graphs and hyper-graphs. We characterize the symmetries of irregular data, construct linear layers that respect this symmetry, and discuss expressiveness of the resulting networks. We will conclude by introducing a simple model for learning graph data that has better expressive power than existing graph neural networks.

**Hao (Richard) Zhang**
(Simon Fraser University)

**Can Machines Learn to Generate 3D Shapes?**
Computer-aided geometric modeling is about synthesis and creation by computing machinery. Early success has been obtained on training deep neural networks for speech and image syntheses, while similar attempts on learning generative models for 3D shapes are met with difficult challenges. In this talk, I will highlight the representation, data, and output challenges we must tackle and how my research has shaped itself to address them. In particular, I argue that the ultimate goal of 3D shape generation is not for the shapes to look right; they need to serve their intended (e.g., functional) purpose with the right part connection, arrangements, and geometry. Hence, I advocate the use of structural representations of 3D shapes and show our latest work on training machines to learn one such representation and an ensuing generative model. At last, I will venture into creative modeling, perhaps a new territory in machine intelligence and ask: can machines learn to generate creative contents?

# Related courses

## Geometric Deep Learning




SGP Graduate School 2017, 2018

http://school.geometryprocessing.org/

# Related courses

## Learning Generative Models of 3D Structures

Eurographics 2019 Tutorial

https://3dstructgen.github.io/

# Related courses

## CreativeAI:
## Deep Learning for Graphics

Siggraph 2019, Eurographics 2018, 2019, Siggraph Asia 2018

http://geometry.cs.ucl.ac.uk/creativeai/

http://geometry.cs.ucl.ac.uk/dl4g/

# Related courses

## Geometric Deep Learning on Graphs and Manifolds

NIPS 2017 Tutorial

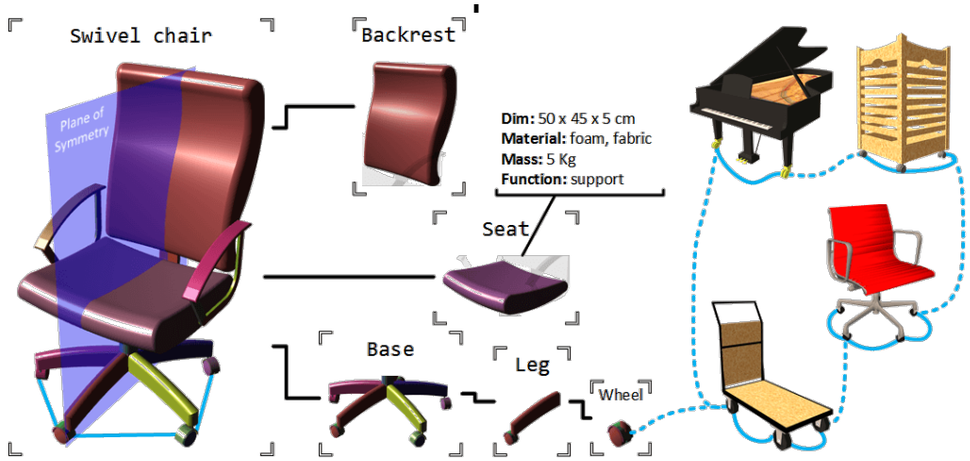
http://www.geometricdeeplearning.com

# Datasets for 3D Deep Learning


**ModelNet (2015)**

ModelNet10: 4899 models, 10 categories

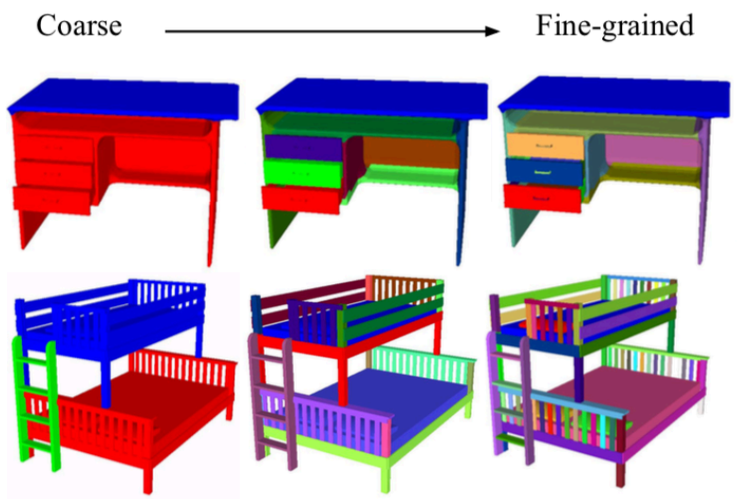
ModelNet40: 12311 models, 40 categories




**ShapeNet (2015)**

3Million+ models and 4K+ categories

ShapeNetCore: 51300 models, 55 categories



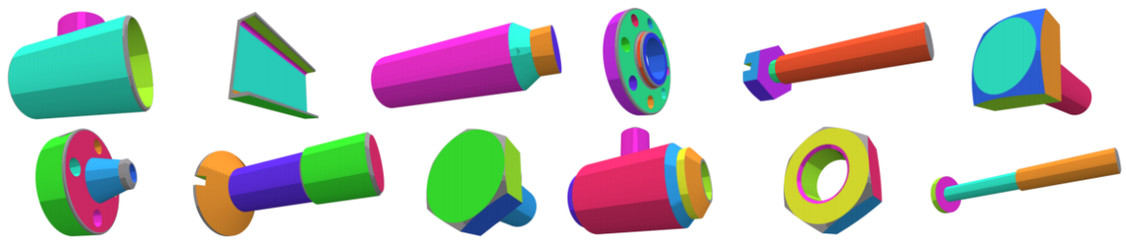

**PartNet (2019)**

A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding

# Datasets for 3D Deep Learning

**ABC (2018)**

A Big CAD Model Dataset
For Geometric Deep Learning

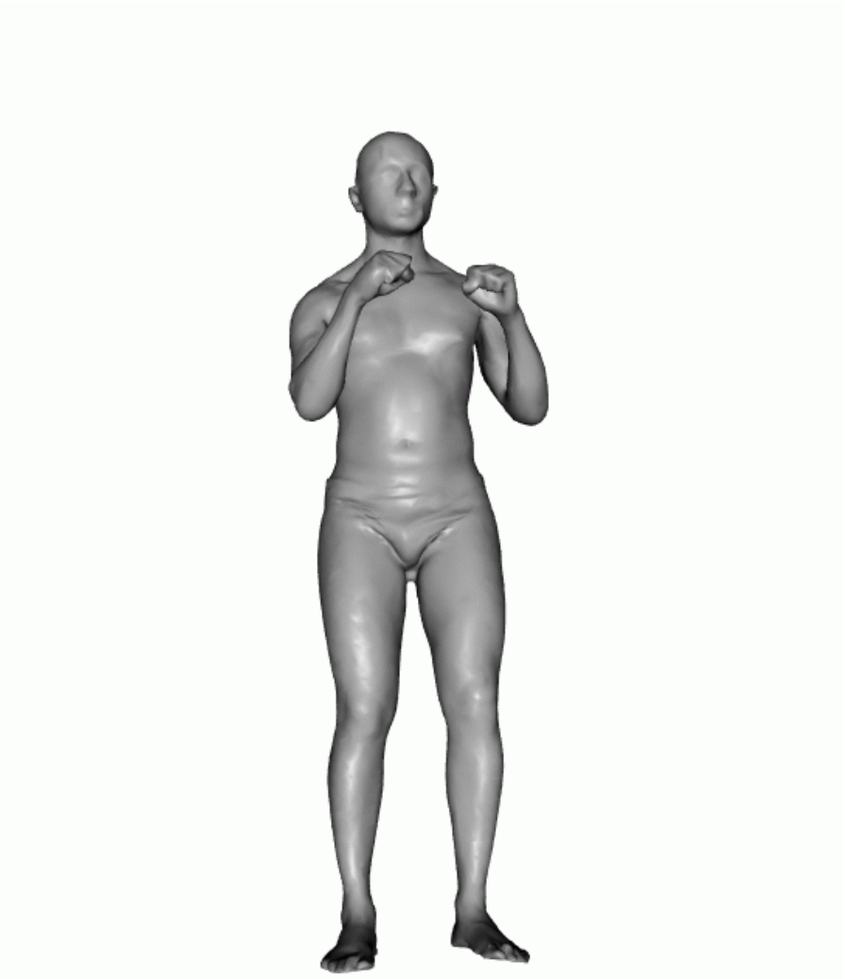


**TraceParts (2019)**

3D dataset of mechanical components

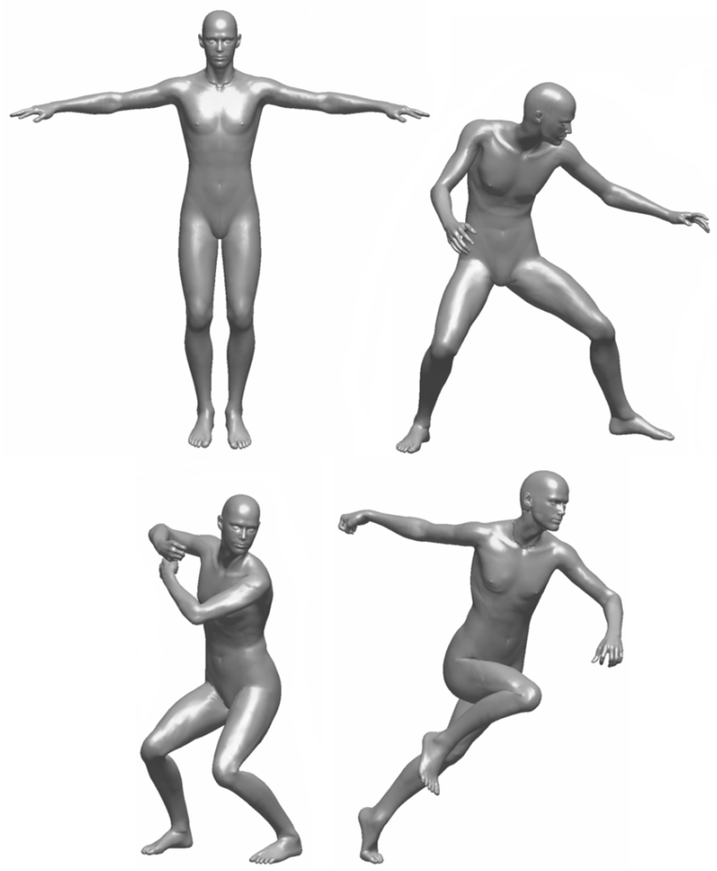
- 3D Machine Learning Github repository
  - https://github.com/timzhang642/3D-Machine-Learning

# Datasets for 3D Deep Learning



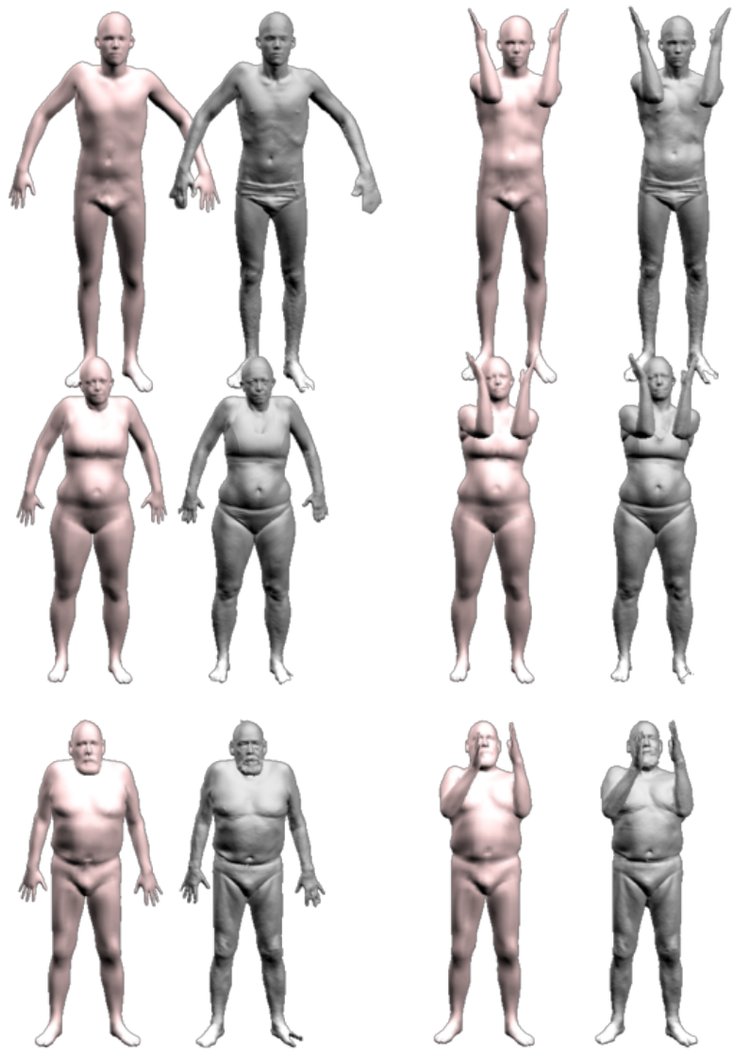
## SCAPE (Stanford)

- 71 human meshes
- same person, different poses
- 12.5K vertices



## TOSCA (Technion)

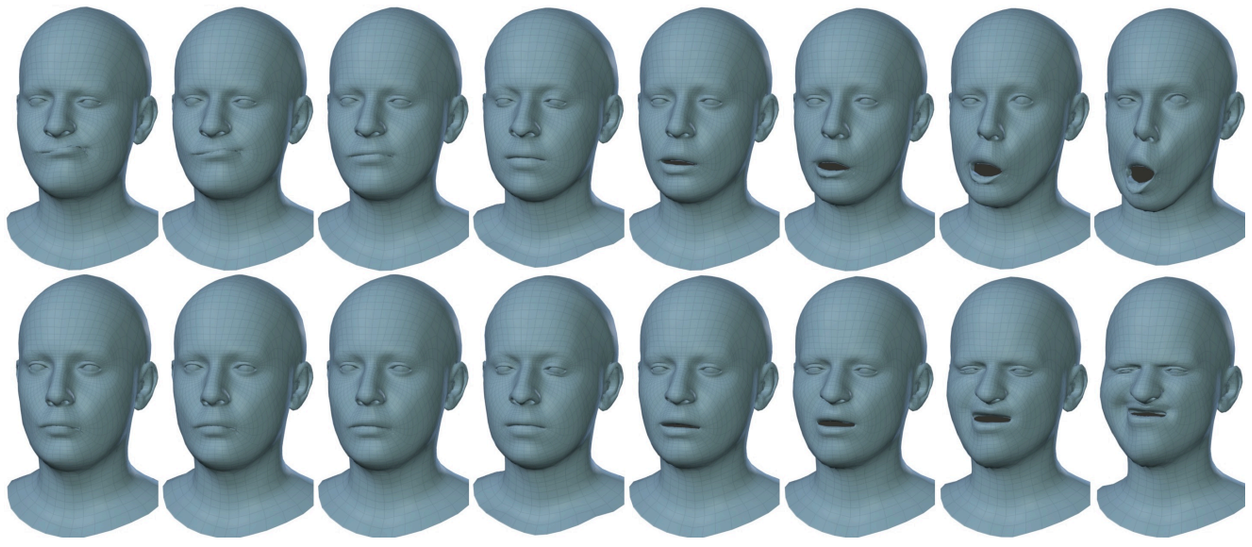
- 80 objects, 9 categories
- human and animal shapes
- 3K-50K vertices



## FAUST (MPI)

- 300 human meshes
- 10 objects in 30 poses
- ~7K vertices

# Datasets for 3D Deep Learning

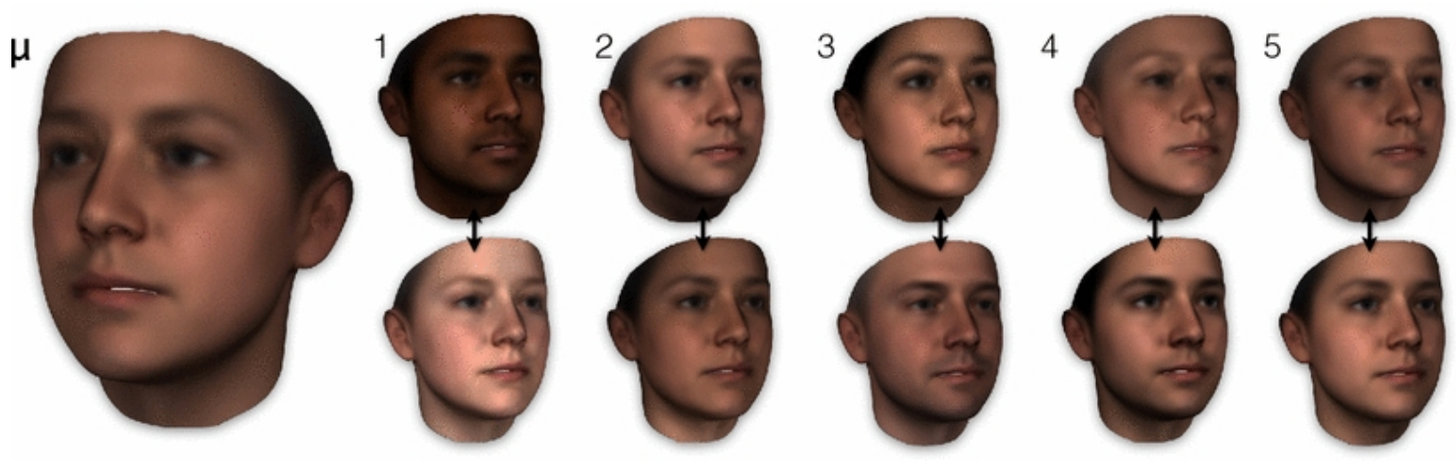


## COMA

Generating 3D faces using Convolutional Mesh Autoencoders

Anurag Ranjan, Timo Bolkart, Soubhik Sanyal and Michael J Black

EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV) 2018, MUNICH, GERMANY

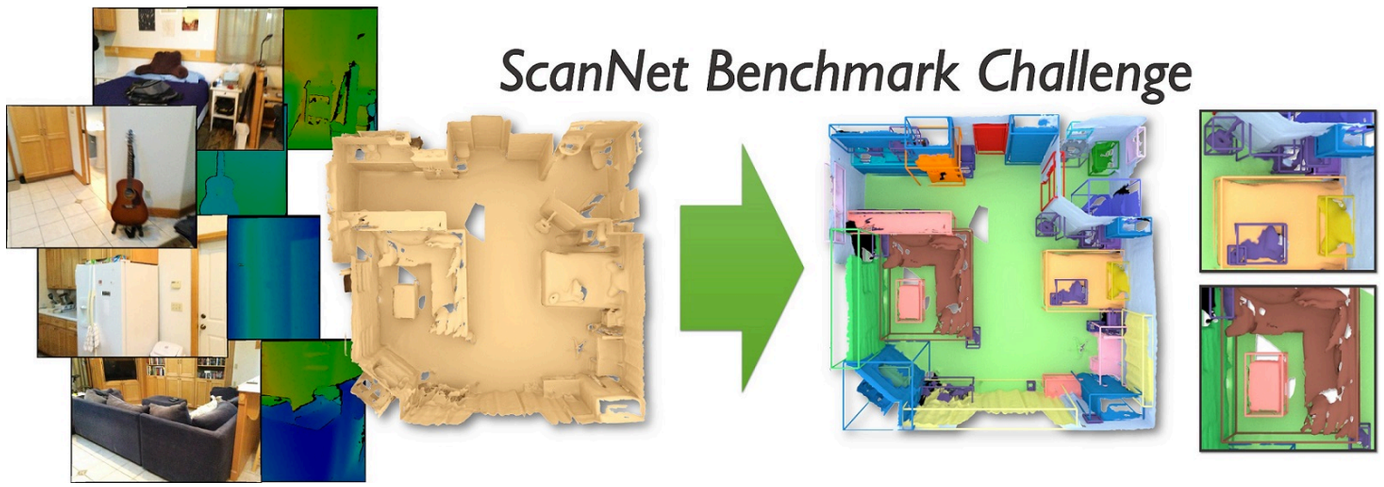



## Large Scale 3D Morphable Models

Booth et al., IJCV 2017

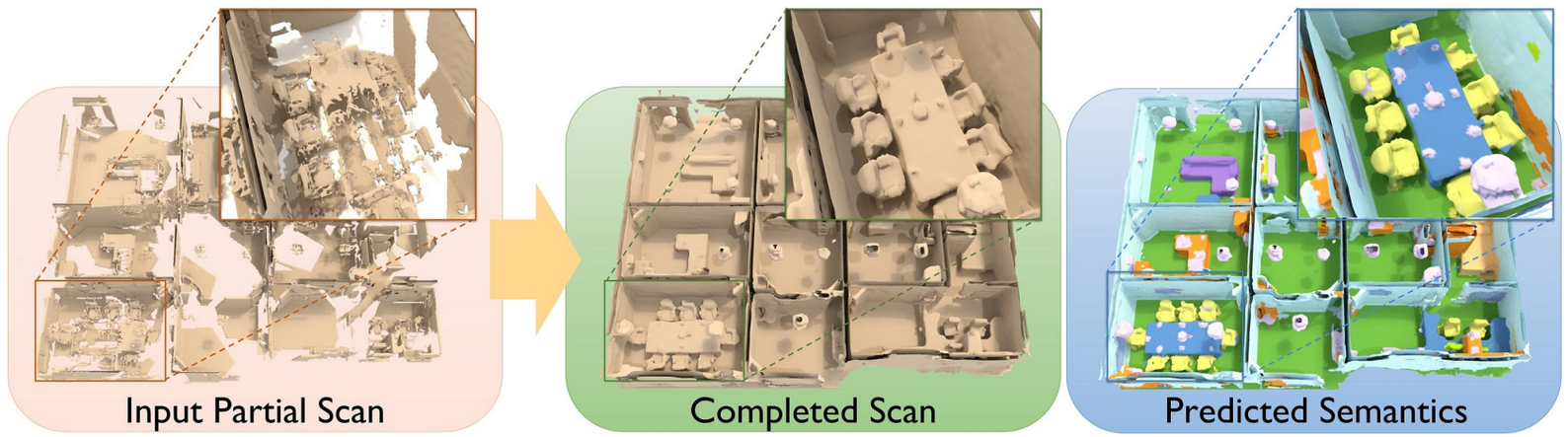
# Datasets for scene understanding

**ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes**




**ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D Scans**

Angela Dai[1,3,5] Daniel Ritchie[2] Martin Bokeloh[3] Scott Reed[4] Jürgen Sturm[3] Matthias Nießner[5]
[1]Stanford University [2]Brown University [3]Google [4]DeepMind [5]Technical University of Munich




Dai et al. [CVPR 2017], Dai et al. [CVPR 2018]