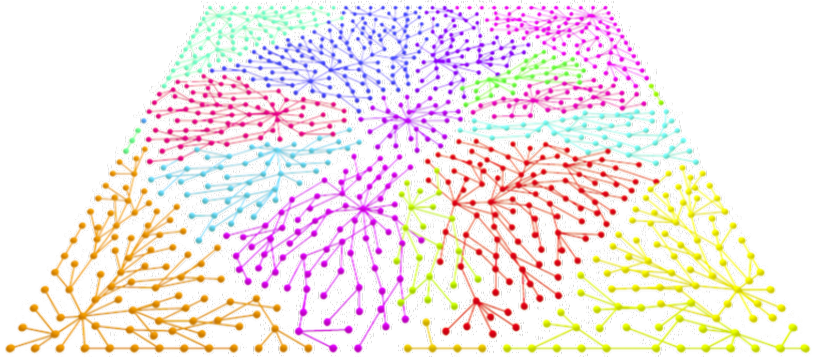
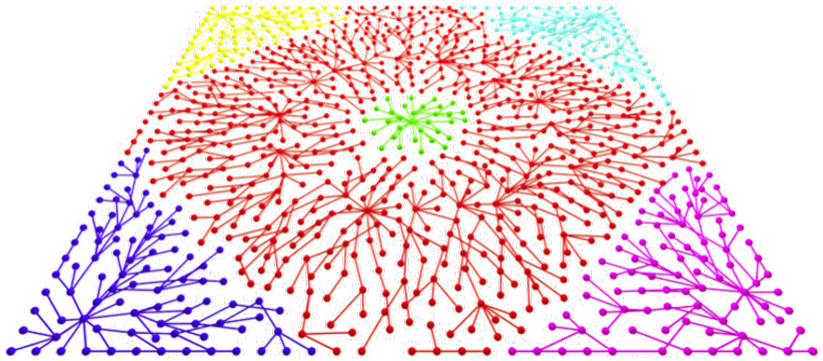
CS233, CME251: Geometric and Topological Data Analysis

Leonidas Guibas
Computer Science Department
Stanford University

Lecture 13
11 May 2022

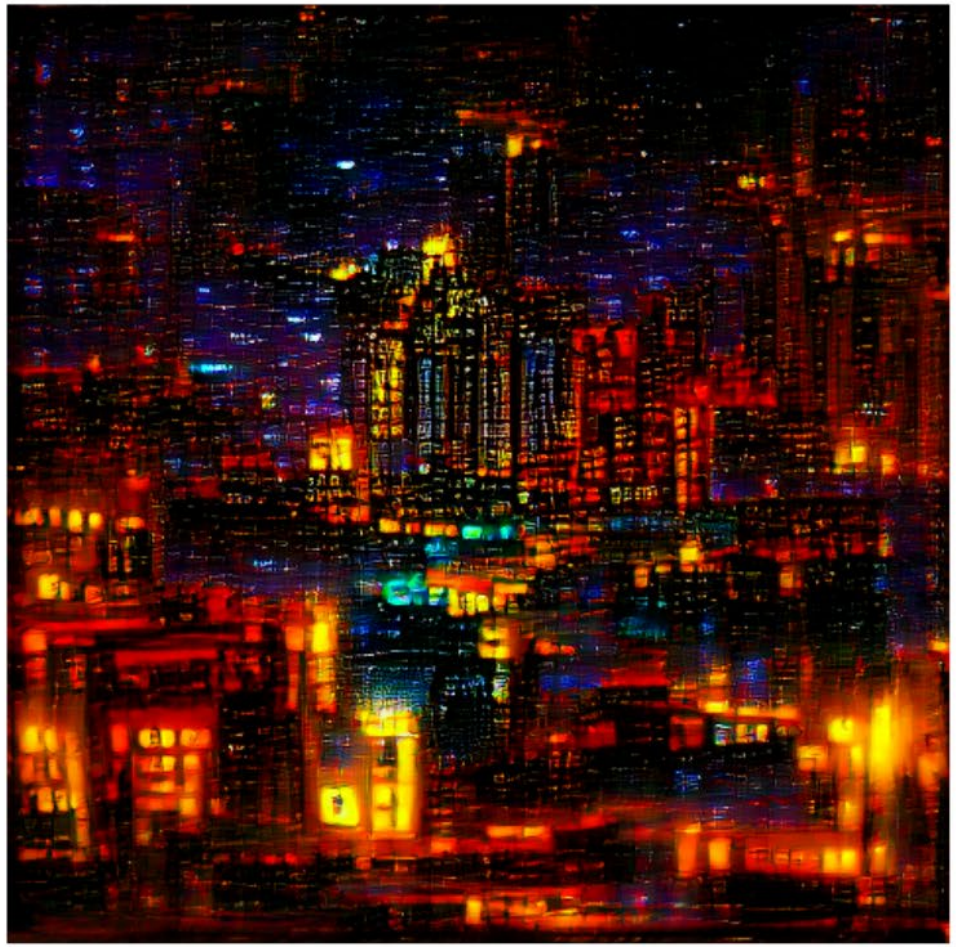




Today: Volumetric and Mesh CNNs for 3D Geometry & Graph Neural Networks

Deep Learning is Awesome!!

An abstract painting of a planet ruled by little castles
Image Source:@RiversHaveWings on Twitter

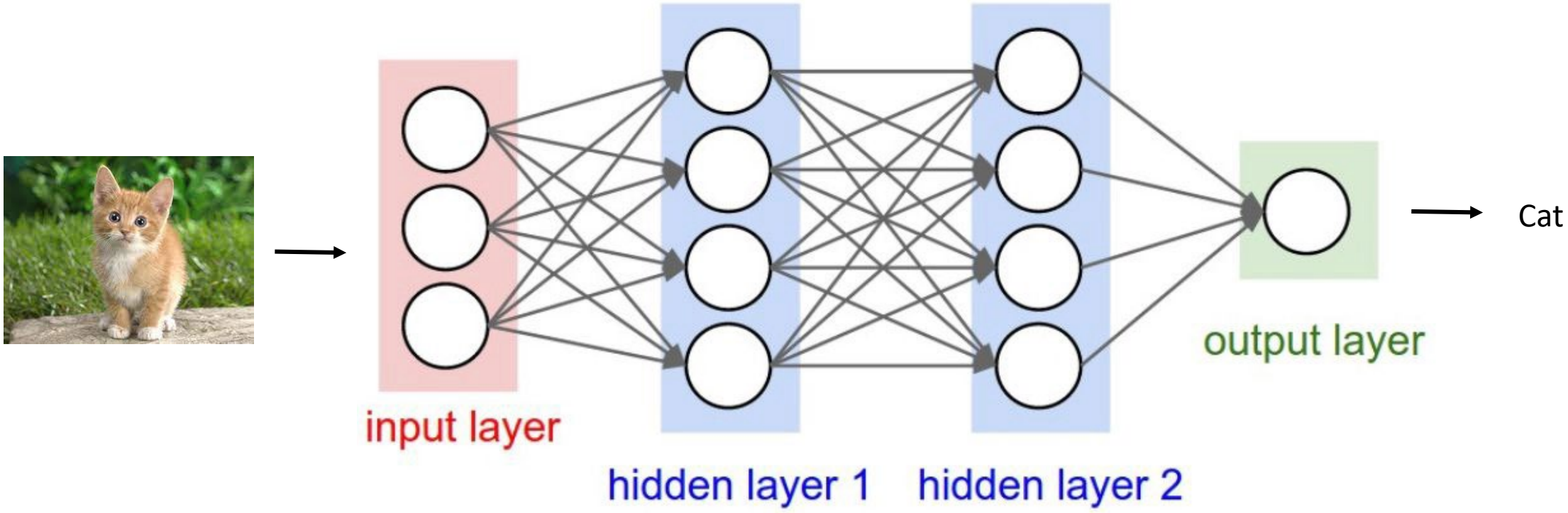


A city scape at night
Image Source:@RiversHaveWings on Twitter

Deep Learning

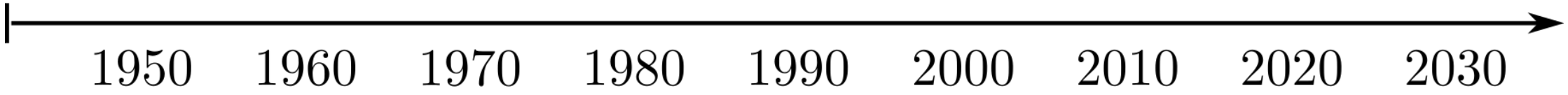
Deep learning allows computational models that are composed of **multiple processing layers to learn representations of data** with **multiple levels of abstraction**.

Deep Learning by Y. LeCun et al. Nature 2015



A Brief History of Deep Learning

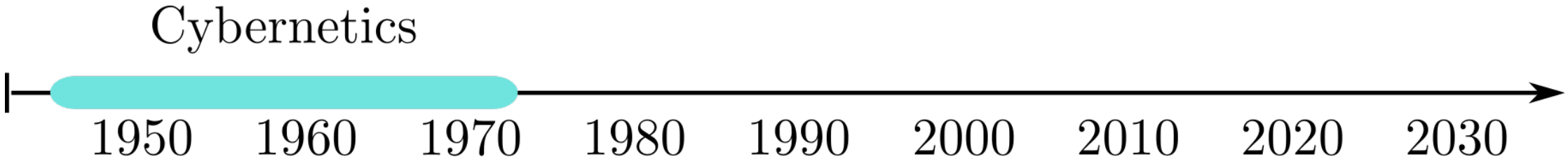
There has been **three waves in the development of Deep Learning**:



A Brief History of Deep Learning

There has been **three waves in the development of Deep Learning**:

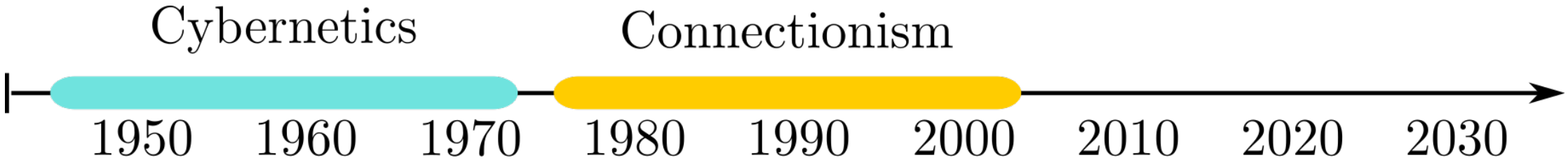
- **1940-1970: Cybernetics**
  - The goal was to utilize **simple rules** inspired by the way the human brain works to **imitate biological learning**



A Brief History of Deep Learning

There has been **three waves in the development of Deep Learning**:

- **1940-1970: Cybernetics**
  - The goal was to utilize **simple rules** inspired by the way the human brain works to **imitate biological learning**
- **1980-2000: Connectionism**
  - Tried to perform more complex tasks by utilizing a large number of simple units

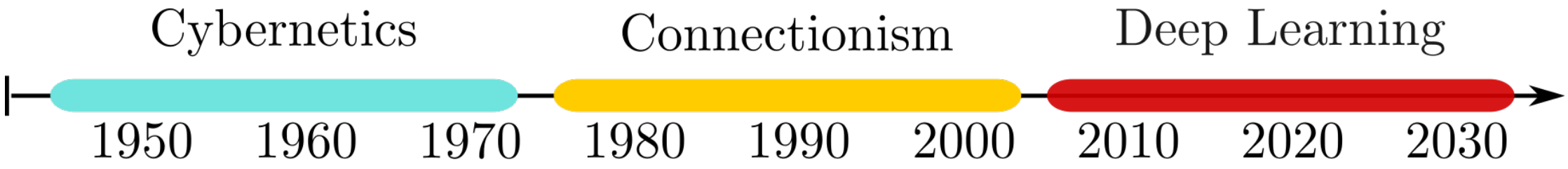


A Brief History of Deep Learning

Slide Credit: Andreas Geiger

There has been **three waves in the development of Deep Learning**:

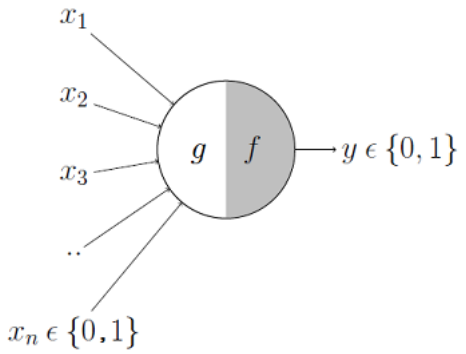
- **1940-1970: Cybernetics**
  - The goal was to utilize **simple rules** inspired by the way the human brain works to **imitate biological learning**
- **1980-2000: Connectionism**
  - Tried to perform more complex tasks by utilizing a large number of simple units
- **2000-Now: Deep Learning**
  - Deeper networks, larger datasets, advances in hardware resulted in state-of-the-art in many areas



A Brief History of Deep Learning

1943: McCulloch-Pitts Neuron

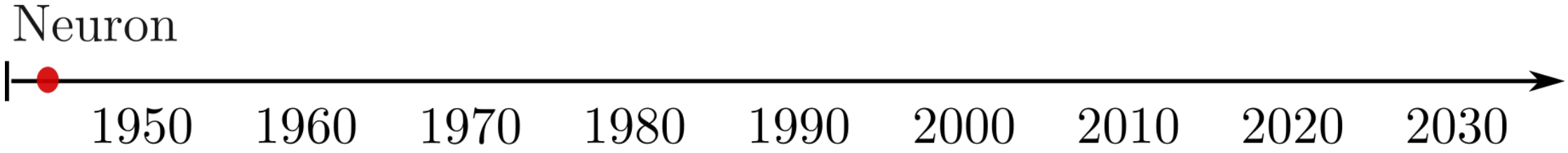
- A neuron can be divided into 2 parts: The first part, ***g*** takes an input performs an aggregation and based on the aggregated value the second part, ***f*** makes a decision.



$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

- No procedure for directly learning the aggregation function ***g.***
- Non-differentiable formulation that cannot be solved with gradient-based optimisation.
- Typically referred to as **Linear Threshold Neuron**.

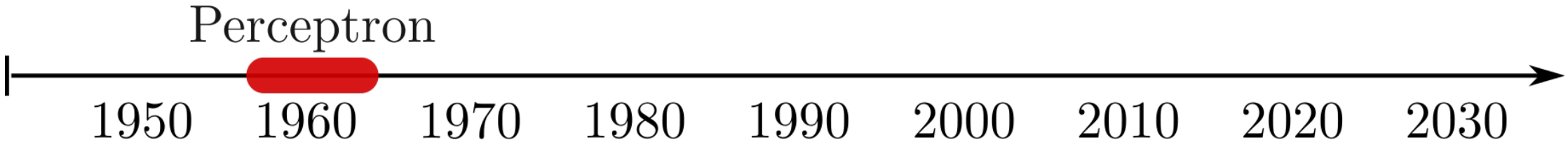


A Brief History of Deep Learning

1958-1962: Rosenblatt's Perceptron

- The first algorithm and the first hardware implementation used to train a single linear threshold neuron.
- Optimization of the perceptron criteric $\mathcal{L}(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T \mathbf{x}_n \, y_n$

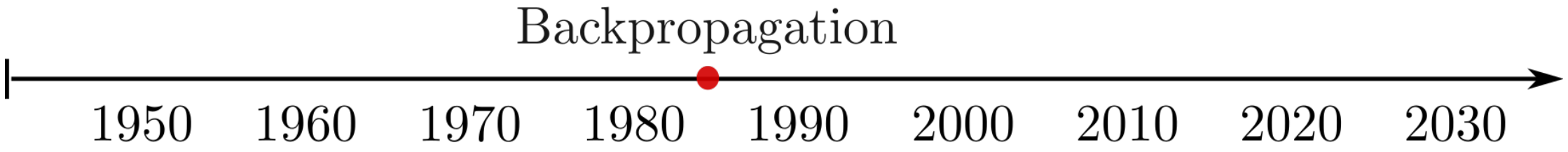
- **Highly Overhyped**: Rosenblatt claimed that the perceptron will lead to computers that walk, talk, see, write, **reproduce and are conscious of their existence**



A Brief History of Deep Learning

1986: Backpropagation Algorithm

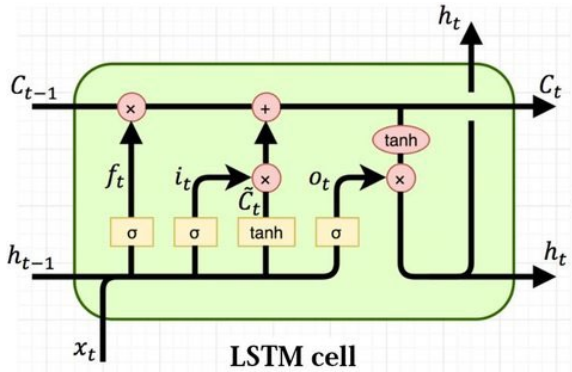
- Efficient calculation of gradients in a deep network wrt. the network weights
- Enables application of gradient based learning to deep networks
- Known since 1961, but the first empirical success was demonstrated in 1986 by Rumelhart and Hinton
- Our main workhorse for learning.



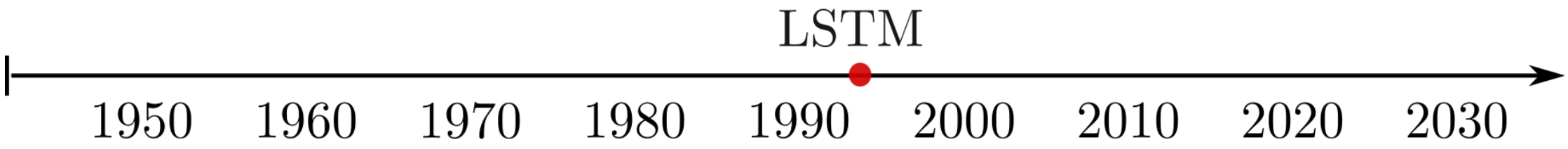
A Brief History of Deep Learning

1997: Long Short-Term Memory

- In 1991, Hochreiter demonstrated t**he problem of vanishing/exploding gradients**. This problem becomes more severe for sequence modelling tasks, where we have variables over long periods of time, such as text.
- Uses feedback loops and forget/keep gate in order to efficiently store information over very long time horizons.
- Has completely revolutionized NLP many years later.



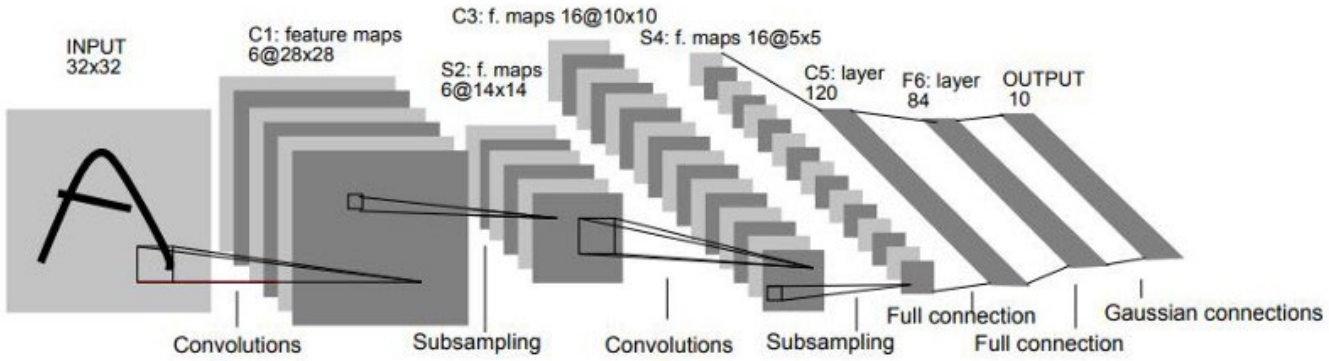
$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

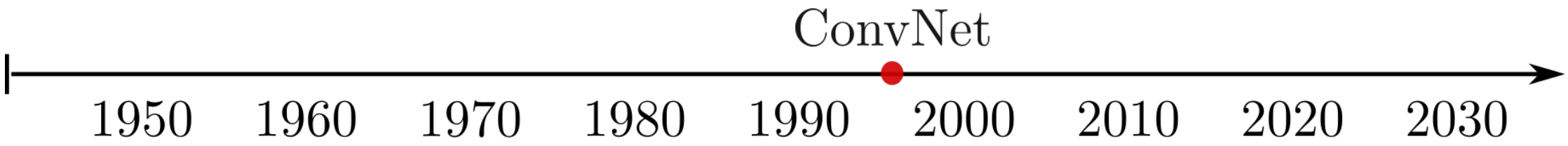


A Brief History of Deep Learning

1998: Convolutional Neural Networks

- Implements spatial invariance via convolution and max pooling
- Weight sharing to reduce parameters
- Tanh/Softmax activations
- Good results on MNIST



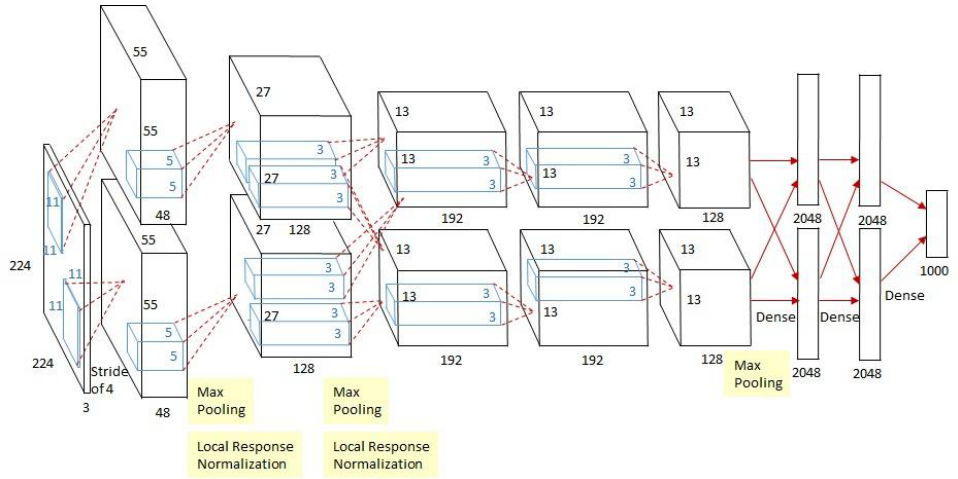


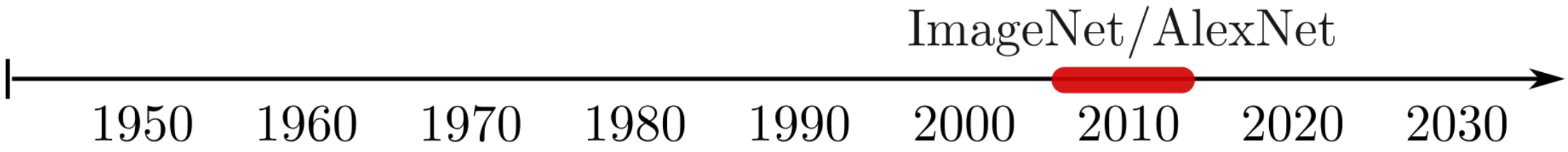
A Brief History of Deep Learning

2009-2012: ImageNet and AlexNet

- **ImageNet:**
  - Recognition benchmark (ILSVRC)
  - 10 million annotated images
  - 1000 Categories

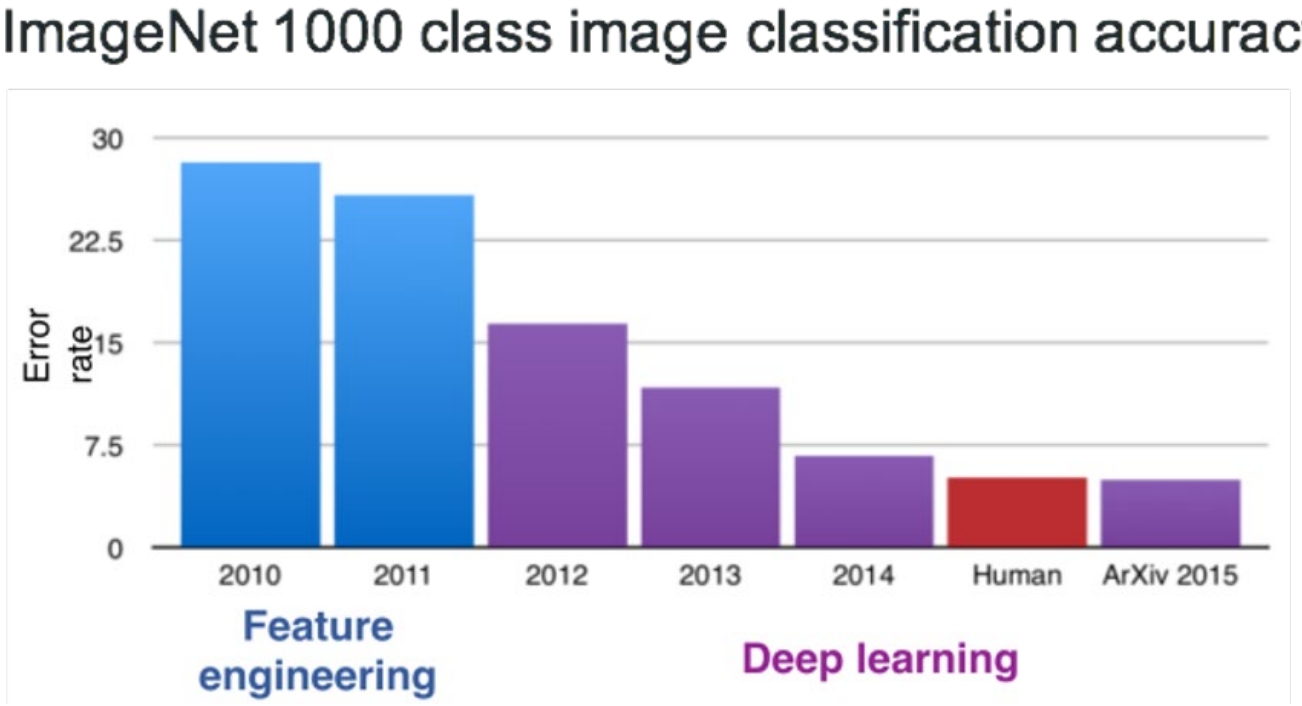
- **AlexNet:**
  - The first neural network to win ILSVRC via GPU training, deep models, data
  - Sparked the deep learning revolution

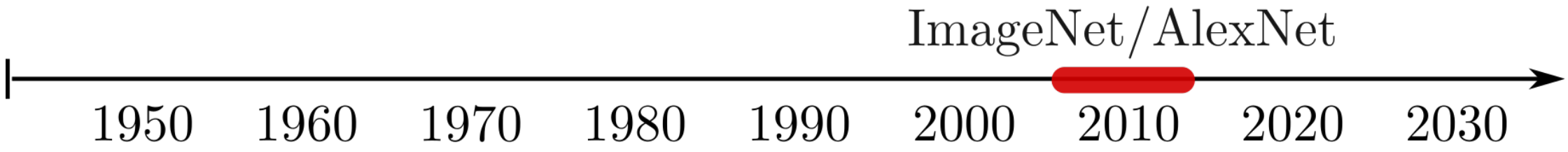




A Brief History of Deep Learning

2009-2012: ImageNet and AlexNet

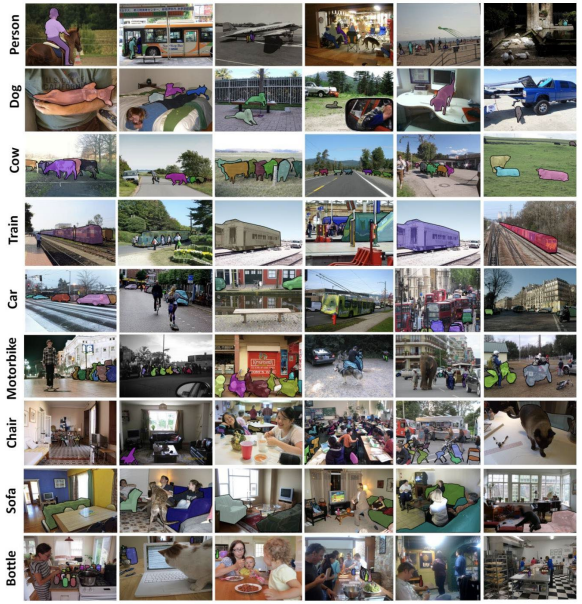




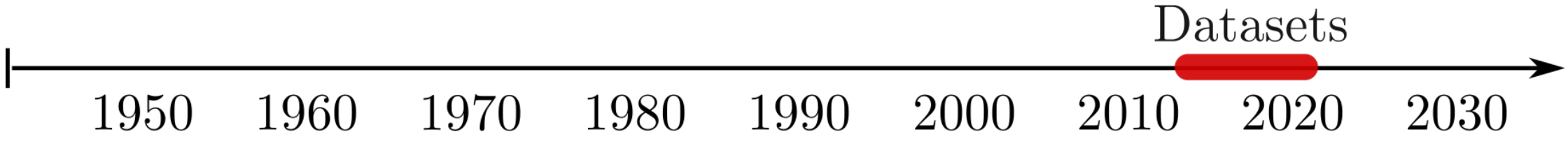
A Brief History of Deep Learning

2012-Now: Golden Age of Datasets

- **PASCAL, MS COCO**: Recognition
- **ShapeNet, ScanNet, Matterport3D**: 3D Reconstruction
- **KITTI, Cityscapes**: Self-driving
- **GLUE**: Language Understanding
- **Visual Genome**: Vision/Language
- **VisualQA**: Question Answering
- ….



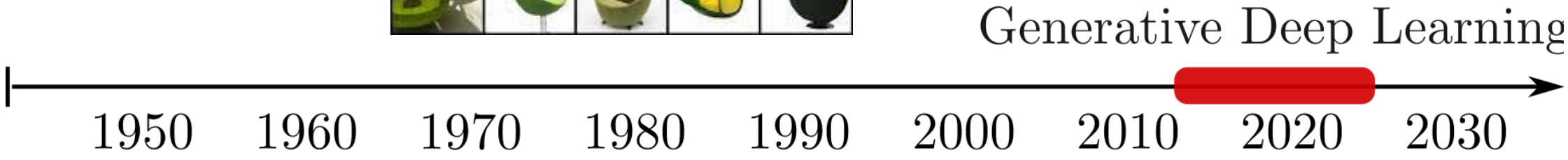
- We also have a lot of Synthetic Datasets:
  - Flying Chairs: Optical Flow Estimation
  - OpenAI Gym: Tool for developing RL algorithms
  - …



A Brief History of Deep Learning

2013-Now: Generative Deep Learning

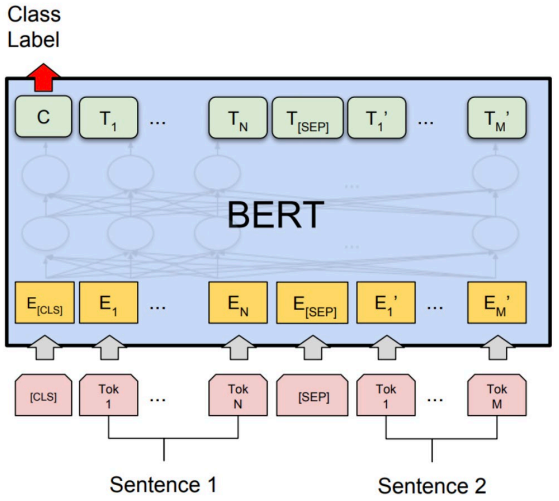
- Variational Autoencoders by Kingma et al. (2013)
- Generative Adversarial Training by Goodfellow et al. (2014)
- RealNVP by Dinh et al. (2016)
- PixelCNN by Oord et al. (2016)
- VQ-VAE by Oord et al. (2017)
- Glow by Kingma et al. (2018)
- ….

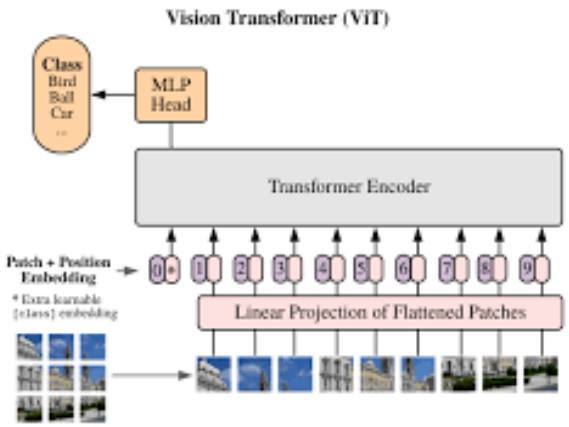


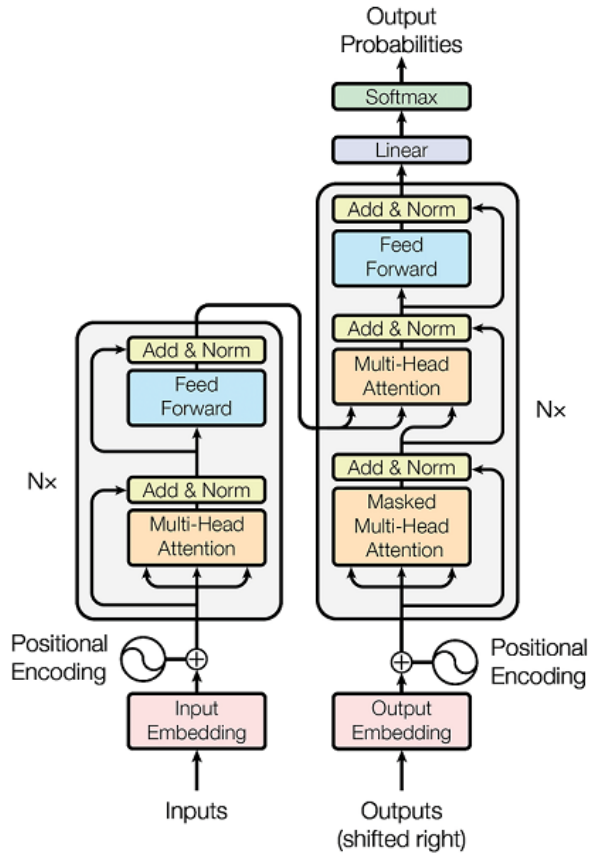
A Brief History of Deep Learning

2017-2018: Transformers and Attention

- Transformer Architectures replace convolutional neural networks **with Attention**
- **BERT:** Allows pre-training language models using unlabeled text
- **ViT:** Transformers can also be used as feature extractors from images







A Brief History of Deep Learning

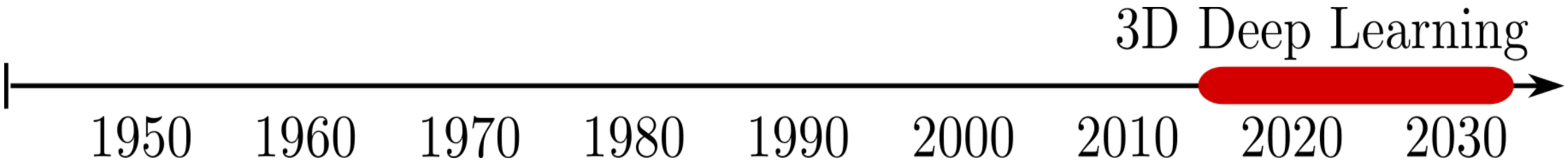
2018: Turing Award is awarded to

- Yoshua Bengio
- Geoffrey Hinton
- Yann LeCun

A Brief History of Deep Learning

2016-Now: 3D Deep Learning

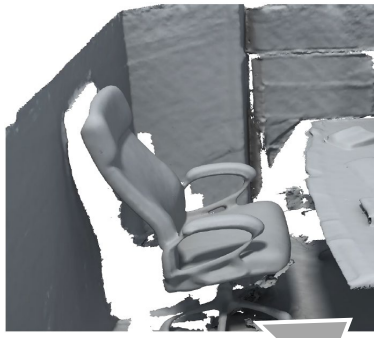
- Learning-based models for outputing 3D representations
- The output representations can be voxels, point clouds, meshes and implicit representations
- We seek to recover the 3D geometry even from a single image
- We try to go beyond geometry and seek to recover materials, light, texture, semantic parts etc.



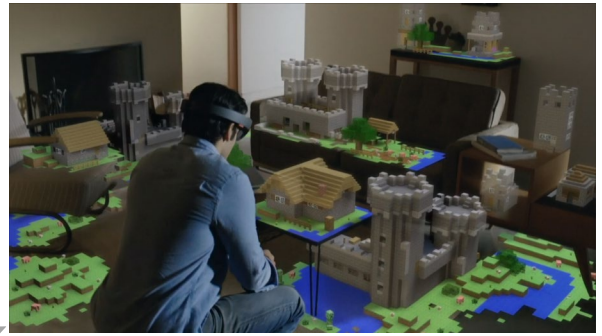
Our world is 3D

Our world is 3D

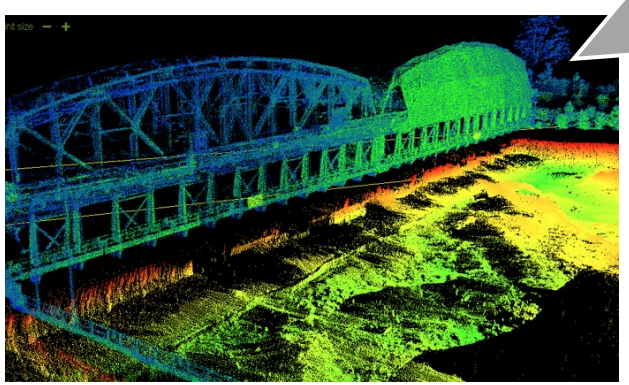
There are multiple 3D Applications



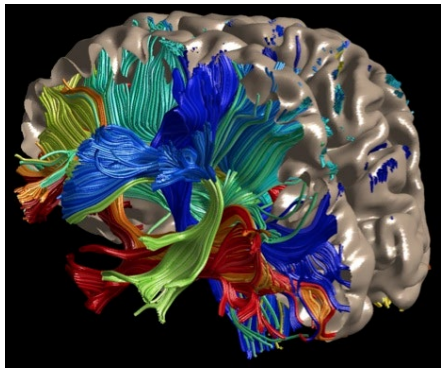
Robotics



Augmented Reality

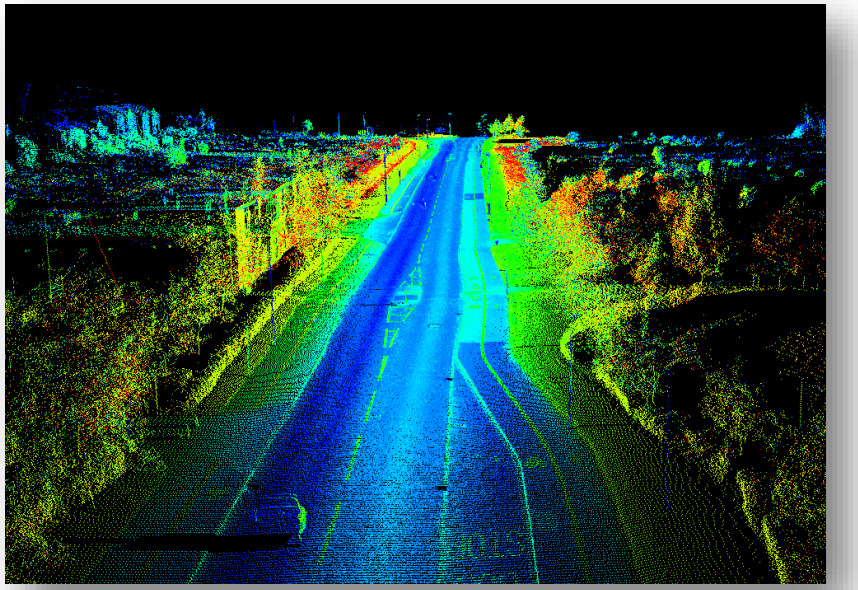


Autonomous driving

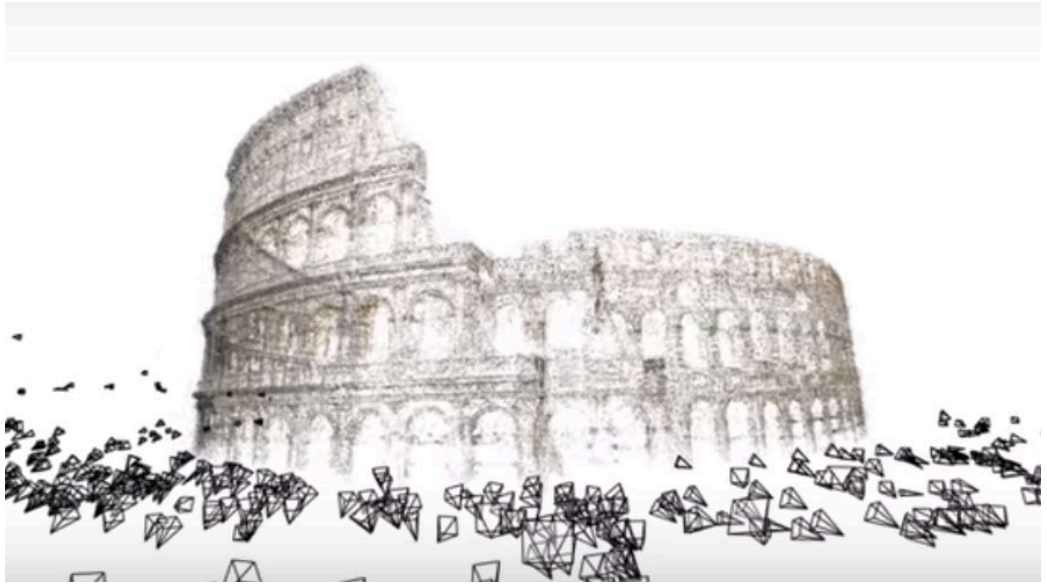


Medical Image Processing

3D Point Clouds from Many Sensors



Lidar point clouds (LizardTech)



Structure from motion (Microsoft)

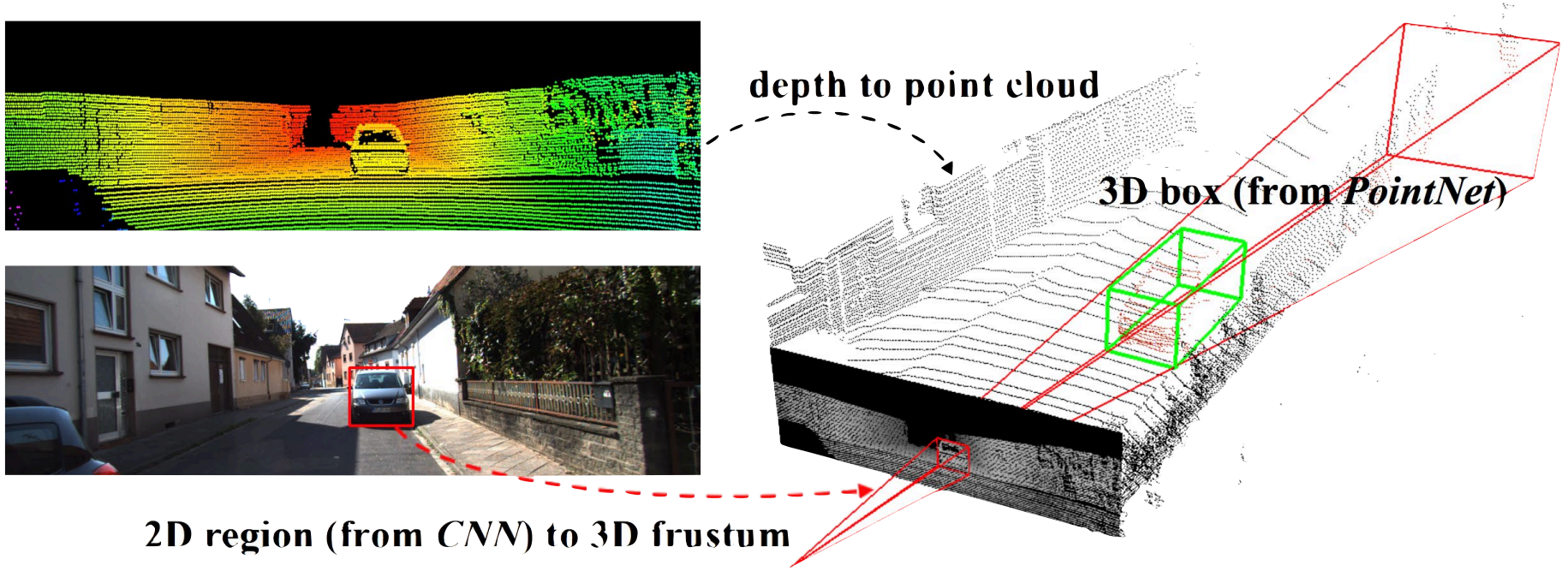
Depth camera (Intel, Microsoft, Google)

Points			
p0	x0	y0	z0
p1	x1	x1	z1
p2	x2	y2	z2
p3	x3	y3	z3
p4	x4	y4	z4
p5	x5	y5	z5
p6	x6	y6	z6
…	…	…	…

In addition, normals, colors, etc.

Understanding in 3D is “Easier”!

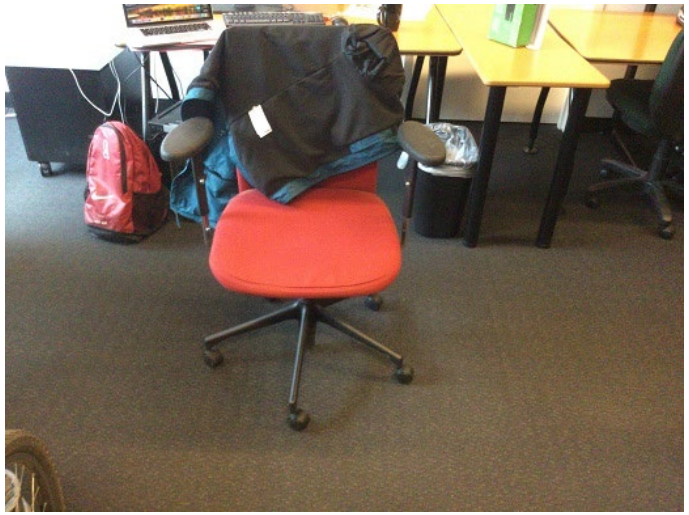
Depth data captured with a laser scan provide a better perception of relative distances to objects



Frustum PointNets for 3D Object Detection from RGB-D Data, Charles Qi et. al.

Understanding in 3D is “Easier”!

Existing 3D scanners even can give us full 3D models! (Not without applying post-processing)



Image/Model Credits: Jingwei Huang

Understanding in 3D is “Easier”!

3D data is less vulnerable to different occlusions, different lighting conditions, etc.

Understanding in 3D is Harder!

3D data is often noisy and of low quality.

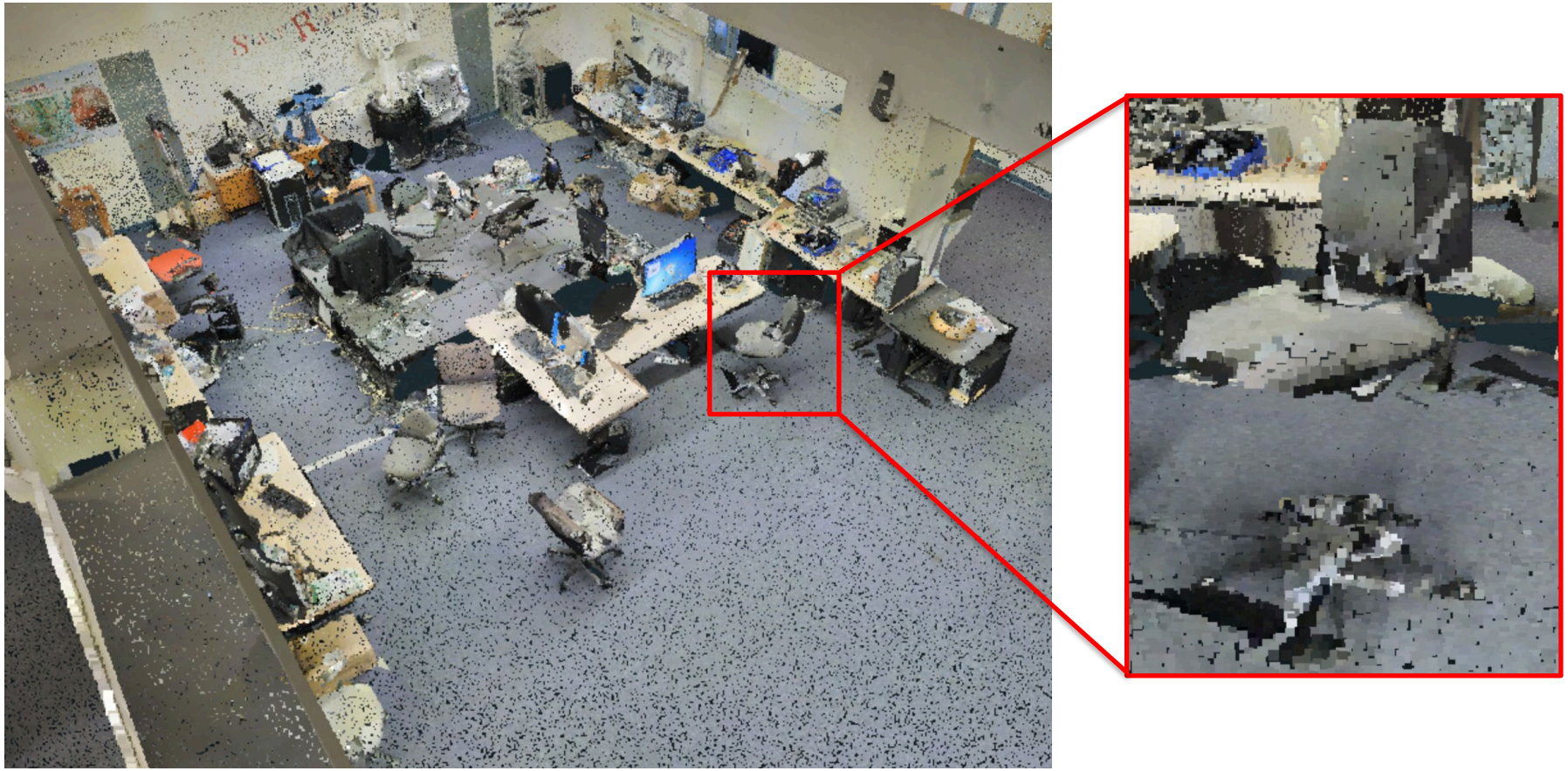


Image Source: http://buildingparser.stanford.edu/dataset.html

Understanding in 3D is Harder!

3D data is mostly incomplete.

Understanding in 3D is Harder!

Compared to 2D images, 3D data is harder to acquire and annotate

- Hard to Scan real-world models
- Hard to design models in CAD softwares

3D Deep Learning Tasks

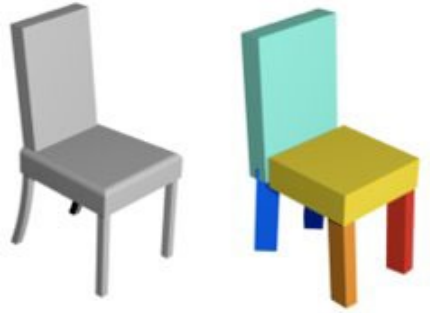
3D Geometrical Analysis

Classification

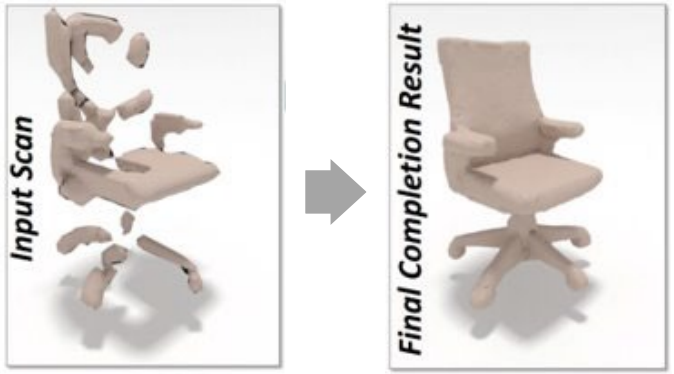
Segmentation

Correspondence

3D Reconstruction



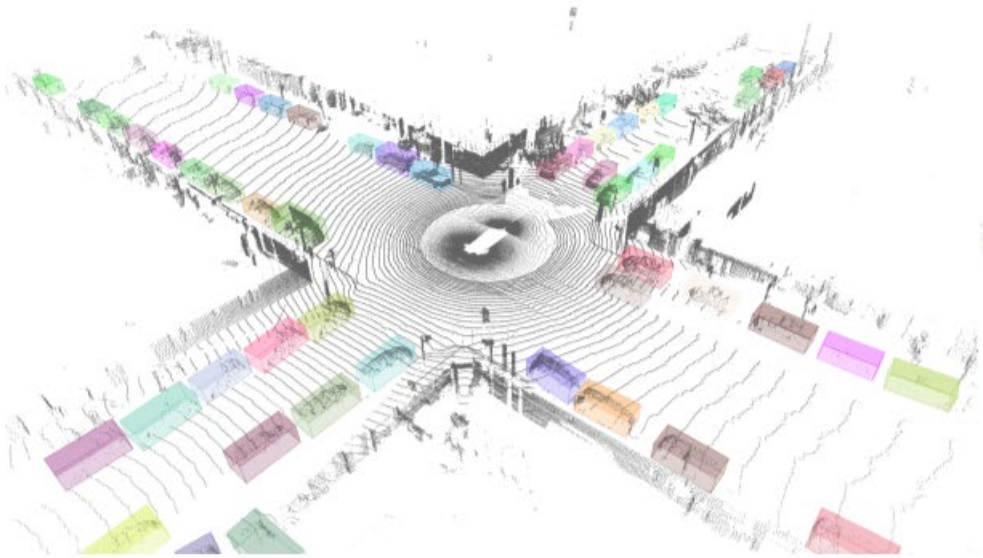
Shape Abstractions

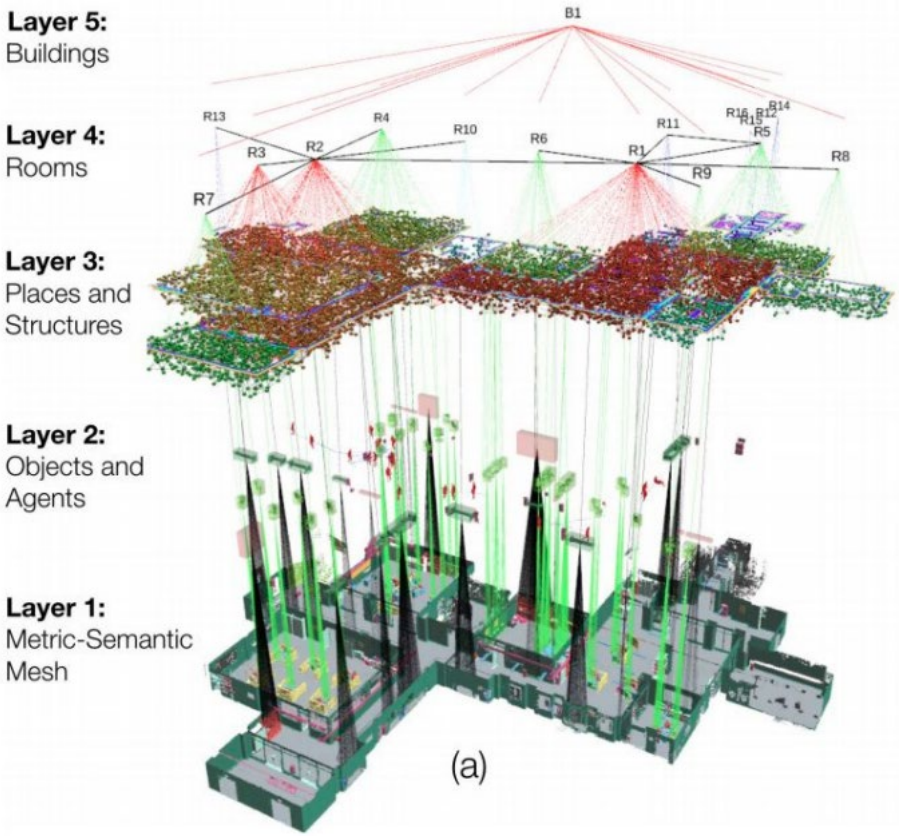


Shape Completion

3D Deep Learning Tasks

3D Scene Understanding



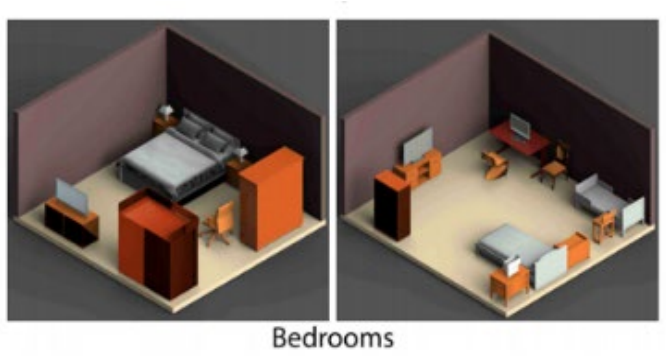


Object Detection / Scene Segmentation

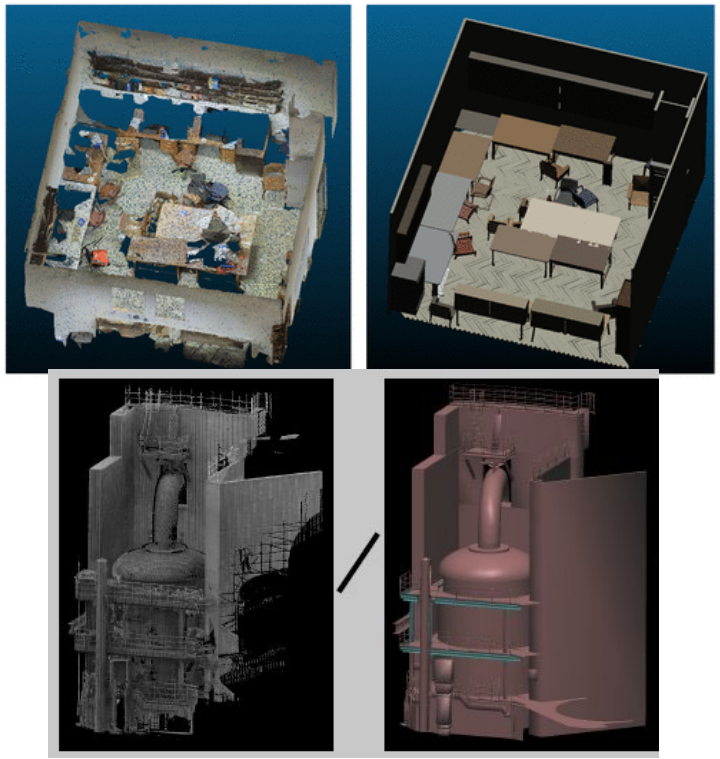
Scene Structure Parsing

3D Deep Learning Tasks

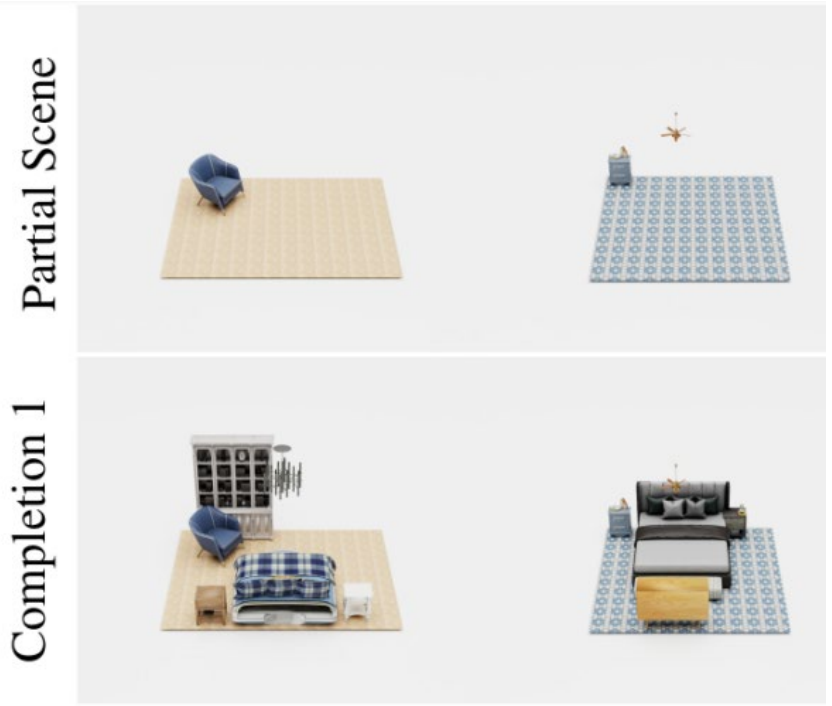
3D Scene Generation and Synthesis



Scene Generation



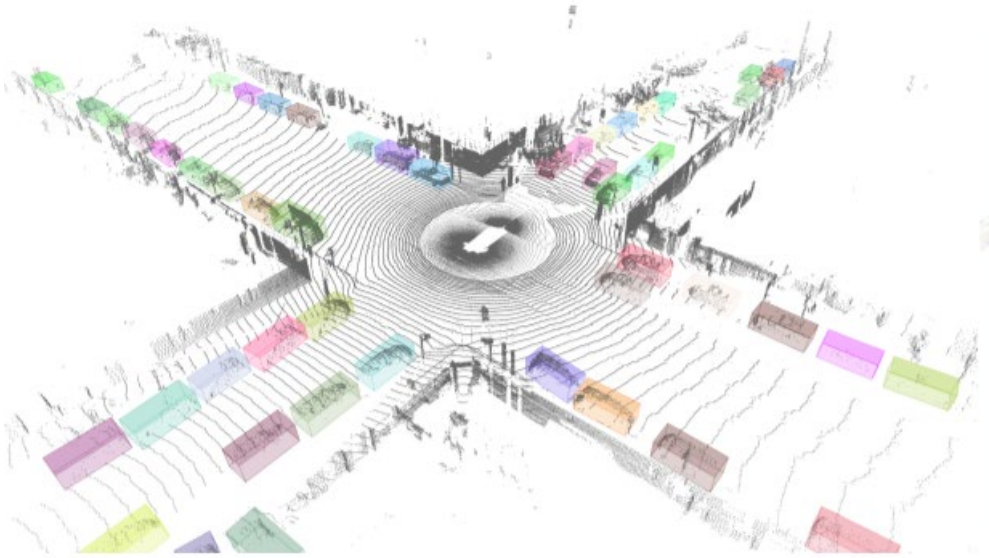
Scene Reconstruction



Scene Completion

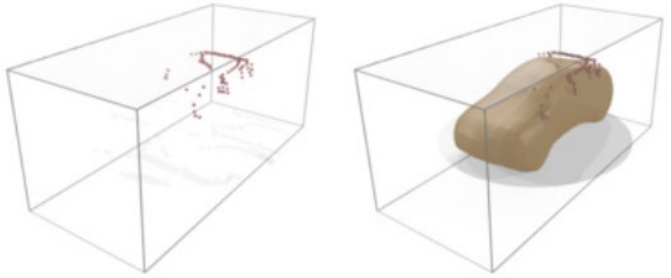
3D Deep Learning Tasks

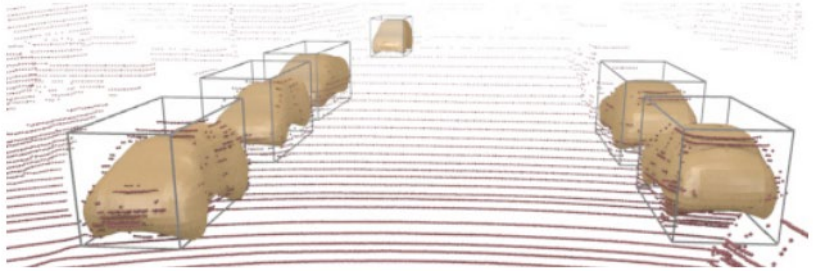
Autonomous Driving Applications



3D Object Detection

Lane Detection

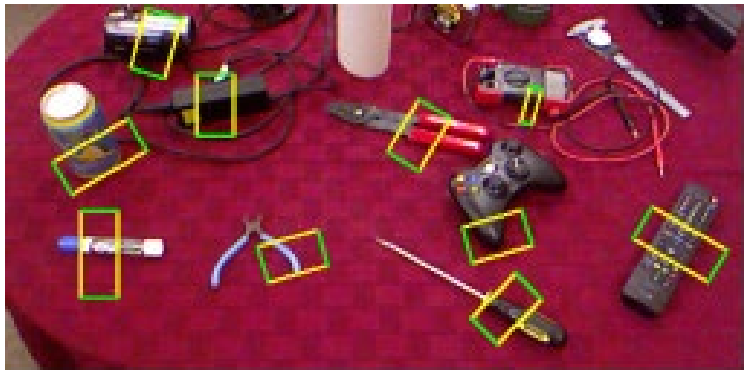




Object Detection/Completion

3D Deep Learning Tasks

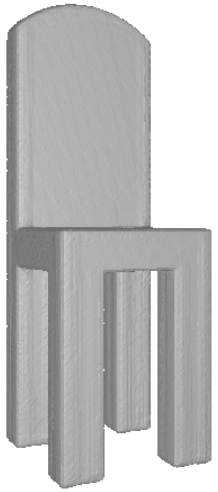
Robotics Applications



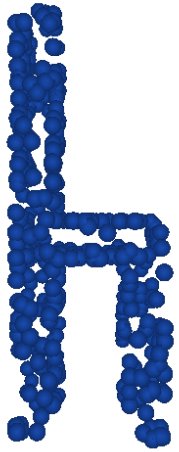
Robot Grasping Affordance Map

Human Interaction Affordance Map

What is the best 3D Representation?

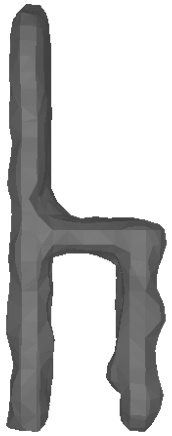


Input Image

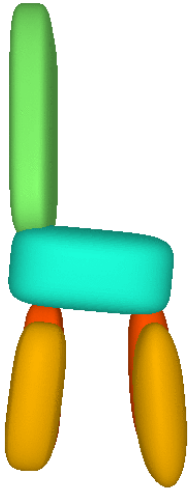


Pointcloud

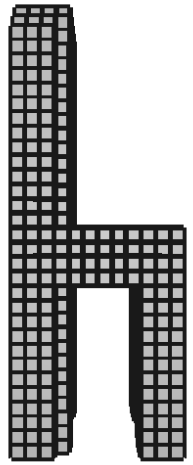
Depth



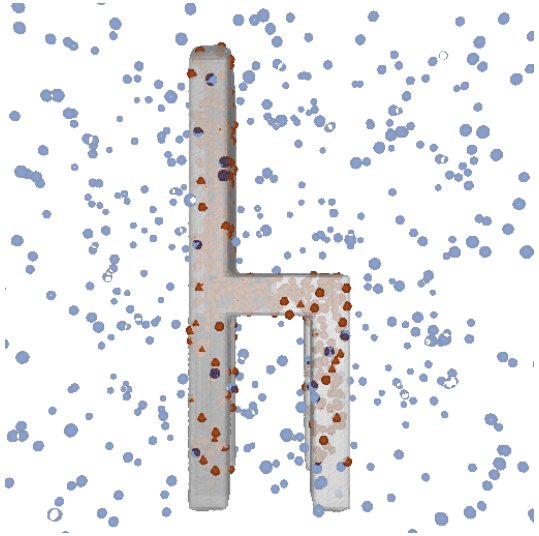
Mesh



Primitives

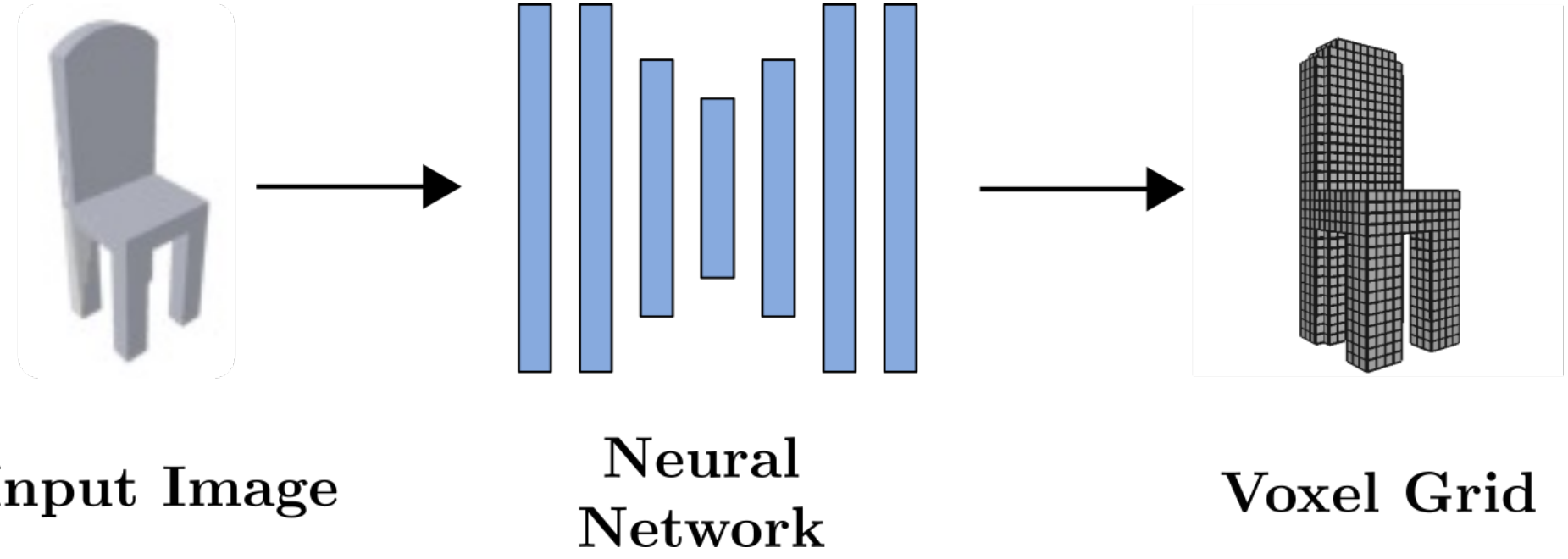


Voxel Grid



Implicit Surface

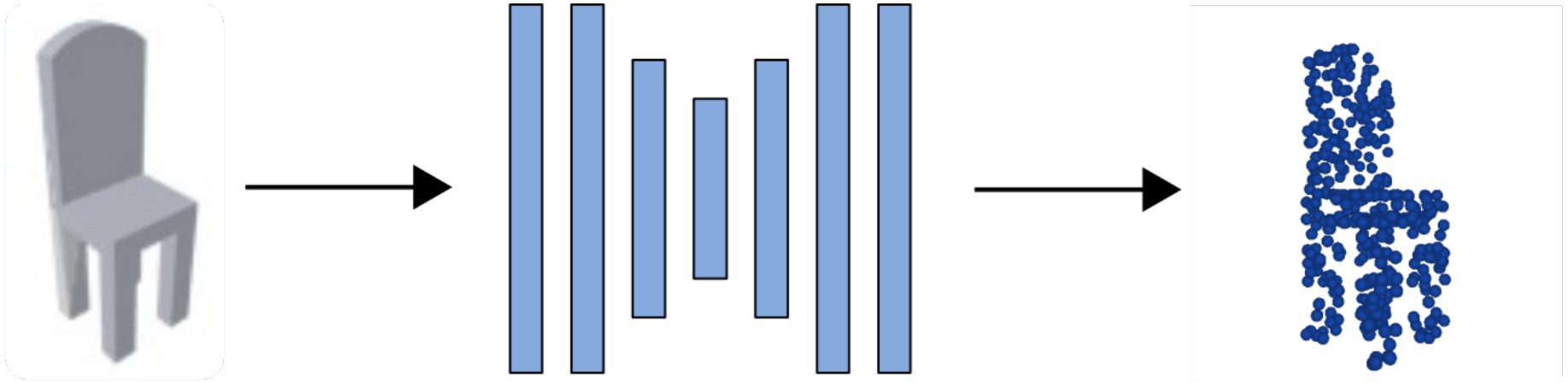
Voxel-based 3D Representations



Discretization of a 3D surface with a voxel grid:

- Can **accurately capture shape details**.
- The **parametrization size** is proportional to the r**econstruction quality**.
- **Cannot** yield **smooth reconstructions**.
- **Cannot** convey **semantic information**.

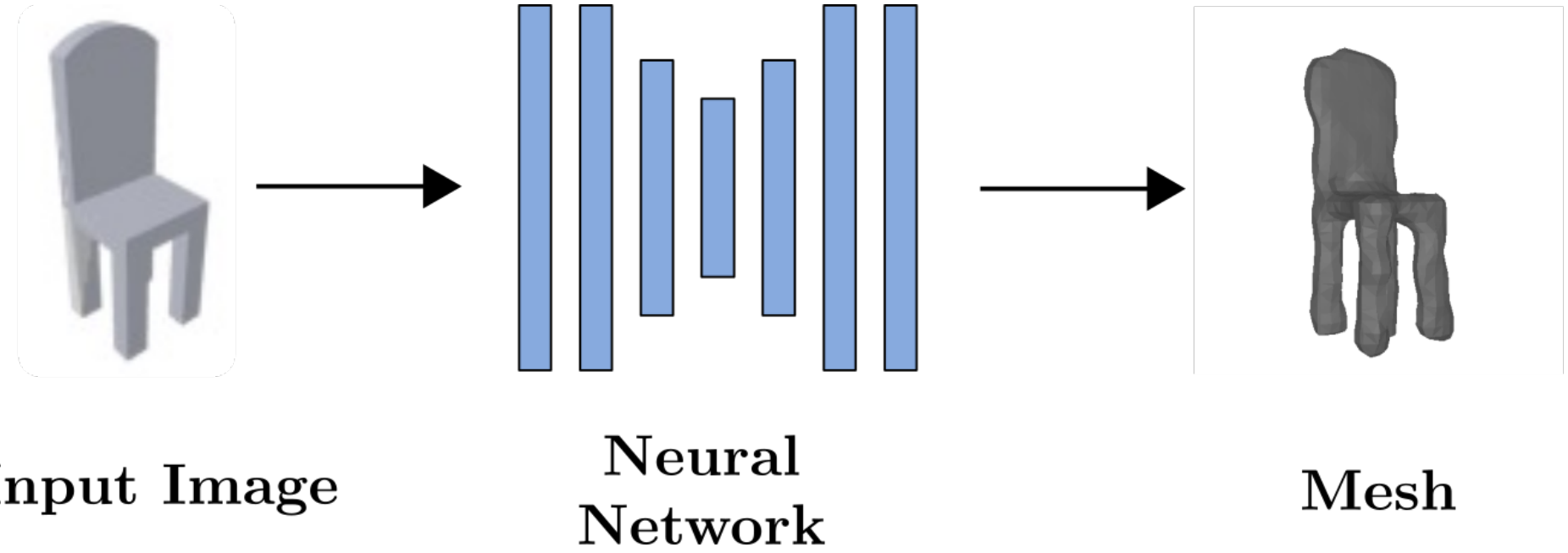
Point-based 3D Representations



Discretization of a 3D surface with 3D points:

- Can **accurately capture shape details**.
- The **parametrization size** is proportional to the r**econstruction quality**.
- **Cannot** yield **smooth reconstructions**.
- **Cannot** convey **semantic information**.
- **Lacks surface connectivity** and assumes a fixed number of points.

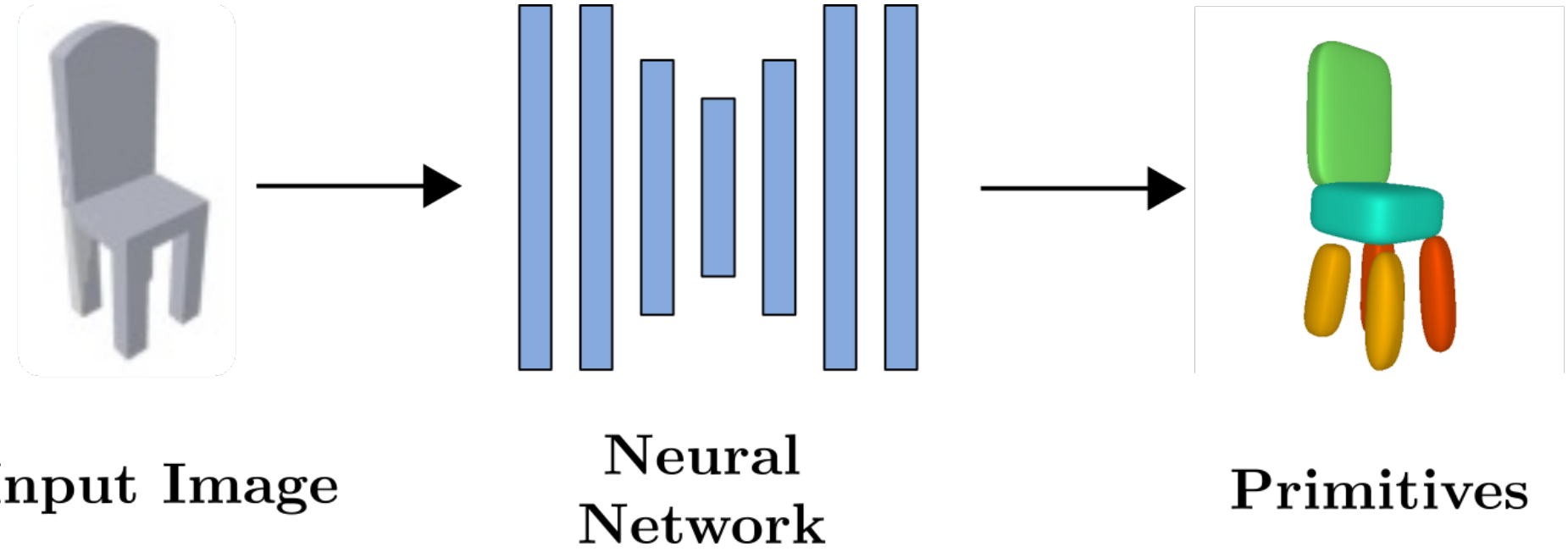
Mesh-based 3D Representations



Discretization of a 3D surface with vertices and faces:

- Can **accurately capture shape details**.
- Yields **smooth reconstructions.**
- Imposes a **large parametrization size.**
- Typically requires **class-specific template topology**.
- **Cannot** convey **semantic information**.

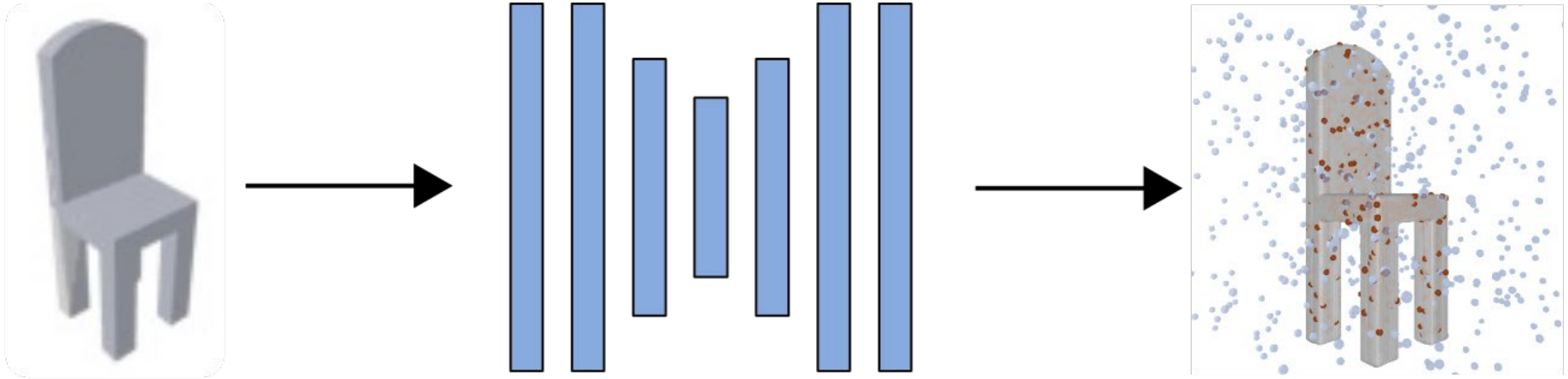
Primitive-based 3D Representations



Discretization of a 3D surface with parts:

- Can **accurately capture shape details**.
- Yields **smooth reconstructions.**
- Imposes a **small parametrization size.**
- Requires post-processing**.**
- Typically **fails to reconstruct fine shape details**.

Implicit-based 3D Representations



No Discretization:

- Can **convey semantic information**.
- Yields **smooth reconstructions.**
- Imposes a **small parametrization size.**
- Ensures **inter-object coherence.**
- **Cannot** convey **semantic information**.

Volumetric Neural Networks

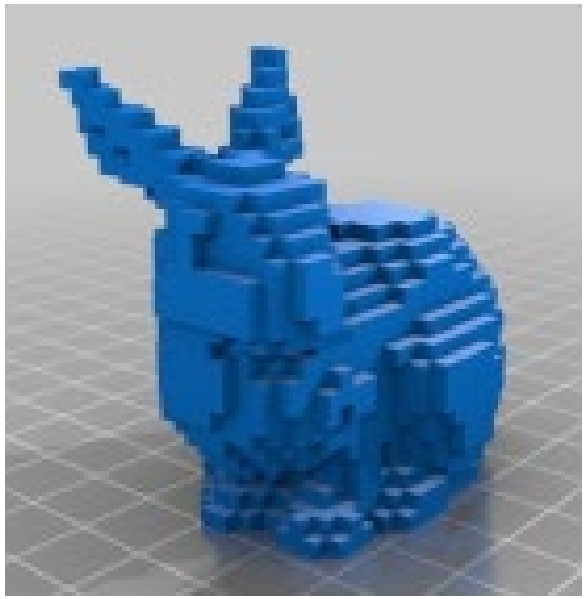
Volumetric Representation: 2D Images

Images: canonical representation with regular data structure

1	44	33	12	20	23	35	14
51	16	40	32	46	48	28	17
29	60	3	63	49	55	36	7
52	22	26	41	38	10	61	53
2	24	19	11	34	43	5	8
57	9	37	42	25	21	27	18
30	56	50	64	4	59	6	13
58	47	45	31	39	15	62	54

Volumetric Representation: 3D Geometry

Volumetric Representations achieve a similar discretization of the 3D space.



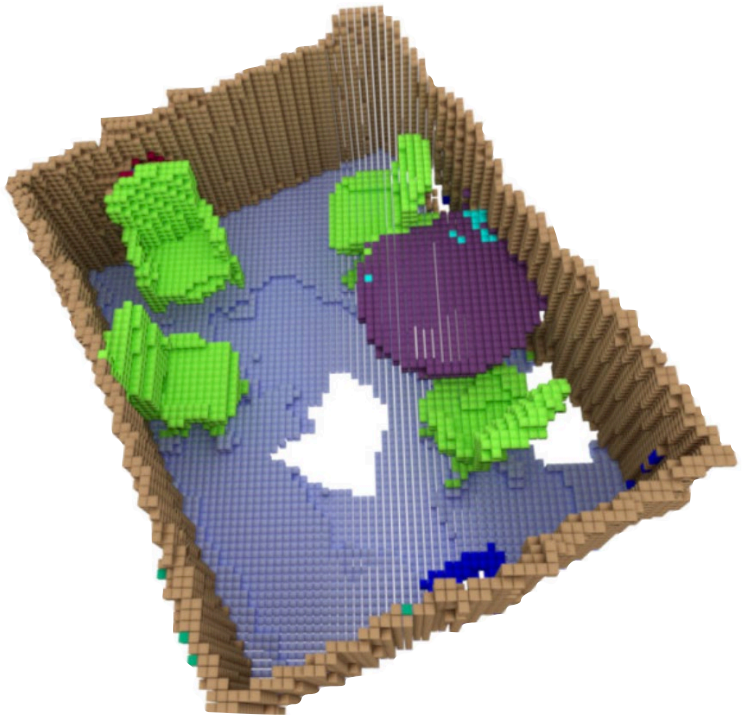
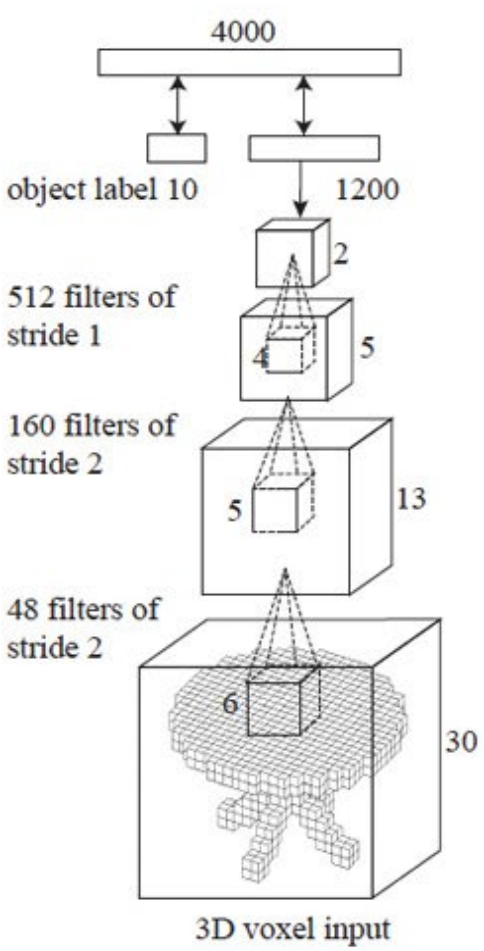
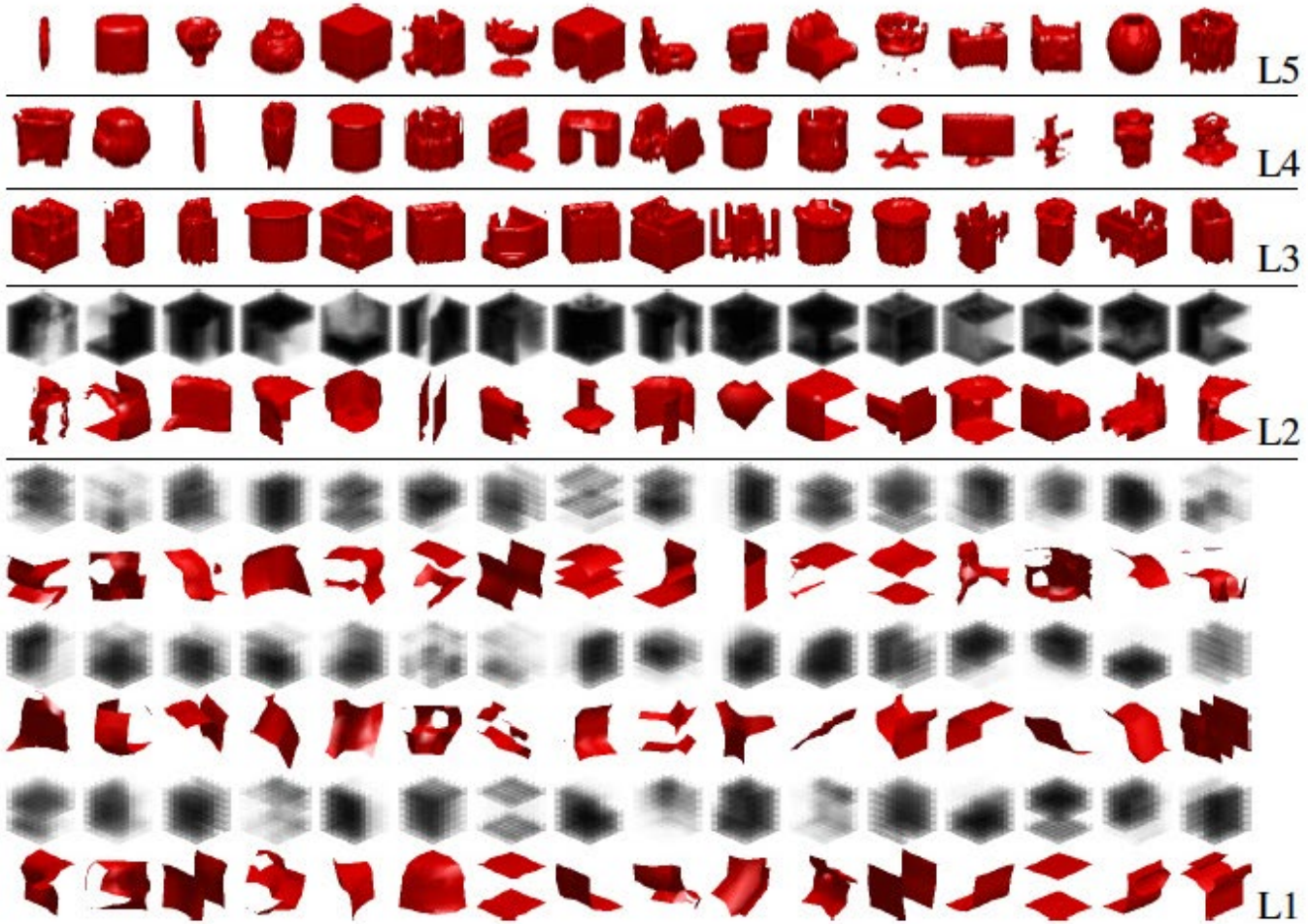


Image Credits: Scannet

Volumetric Deep Learning

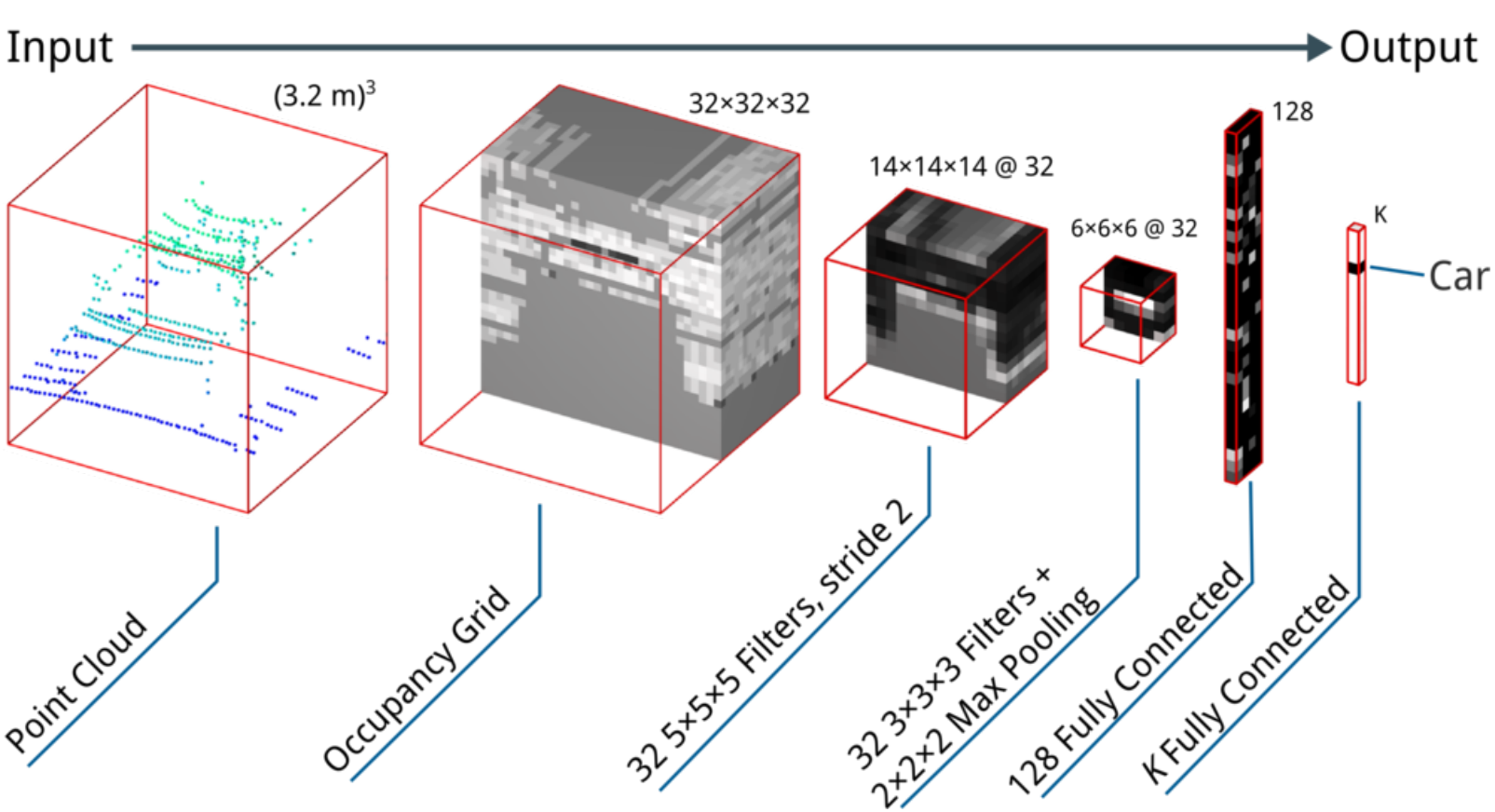


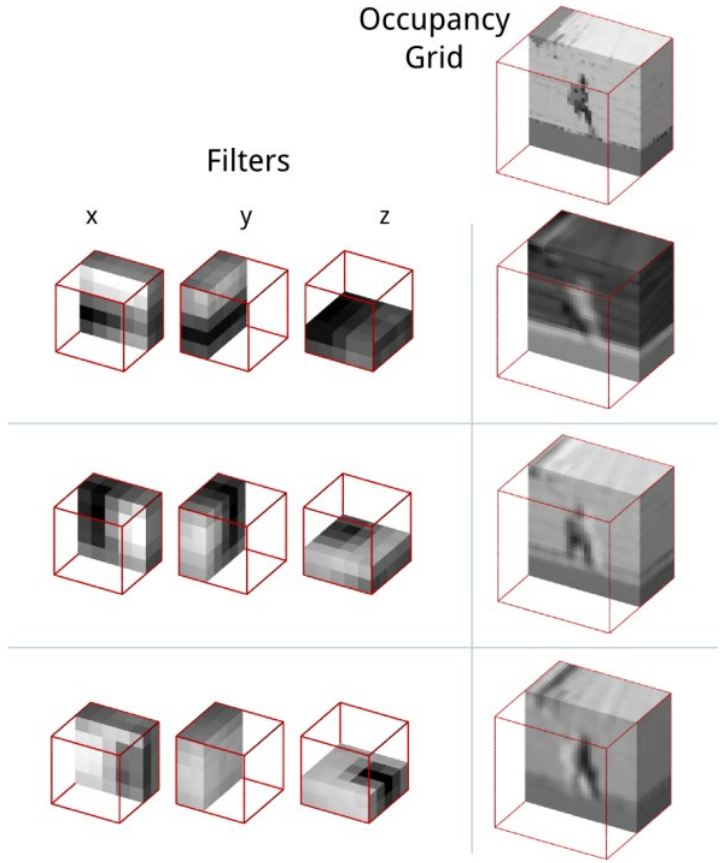


For each neuron, average the top 100 training examples with highest responses

3D ShapeNets: A Deep Representation for Volumetric Shapes, by Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, CVPR 2015

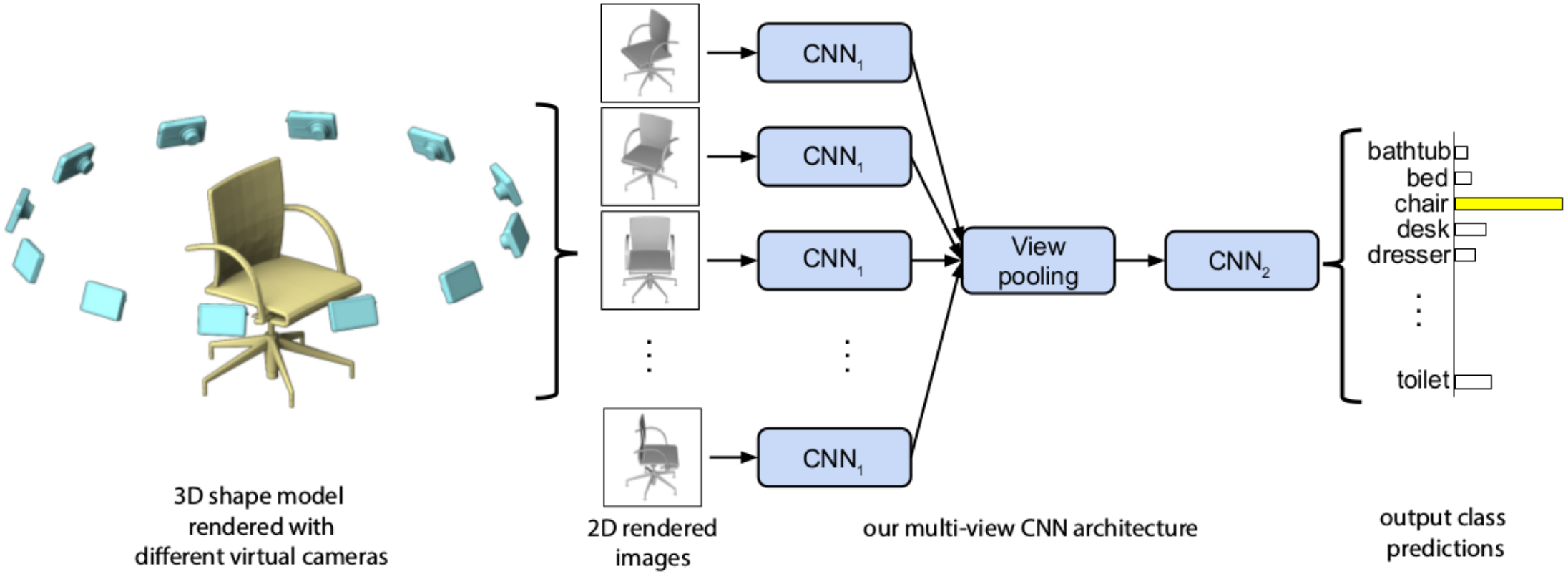
Volumetric Deep Learning: Classification





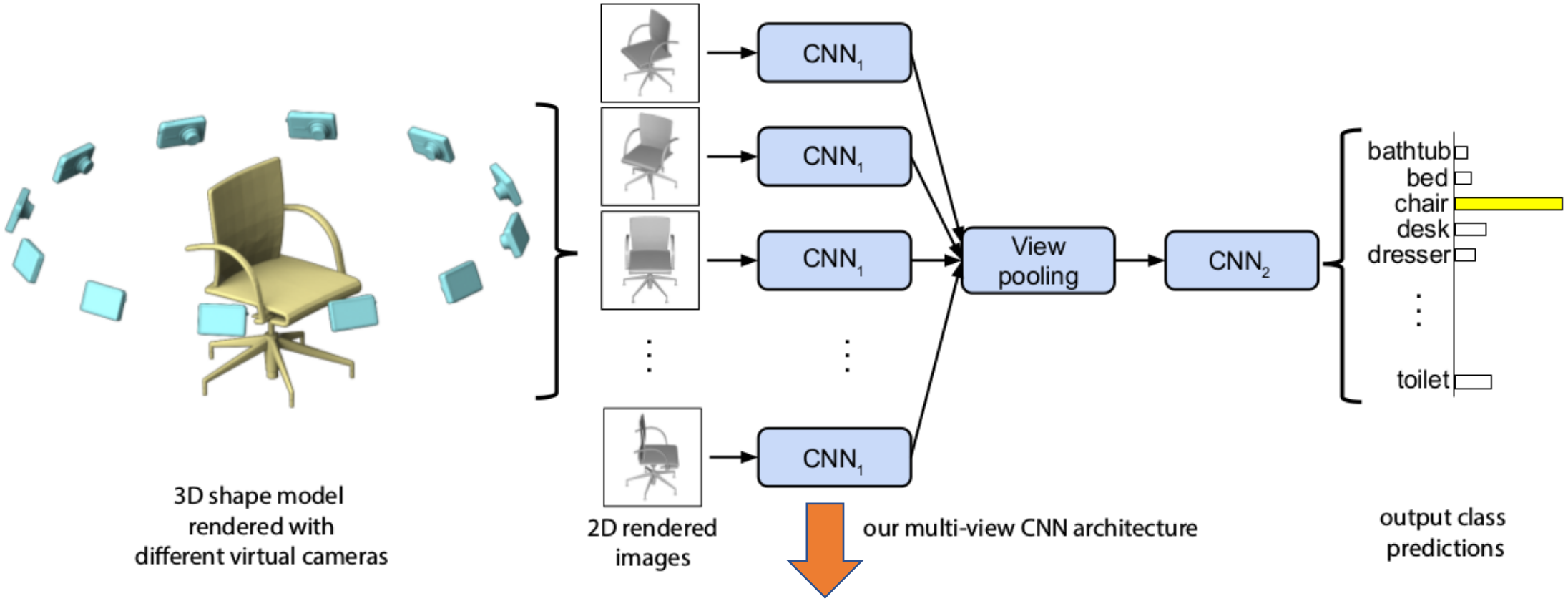
VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition, Daniel Maturana and Sebastian Scherer, IROS 2015

Volumetric Deep Learning: Classification



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

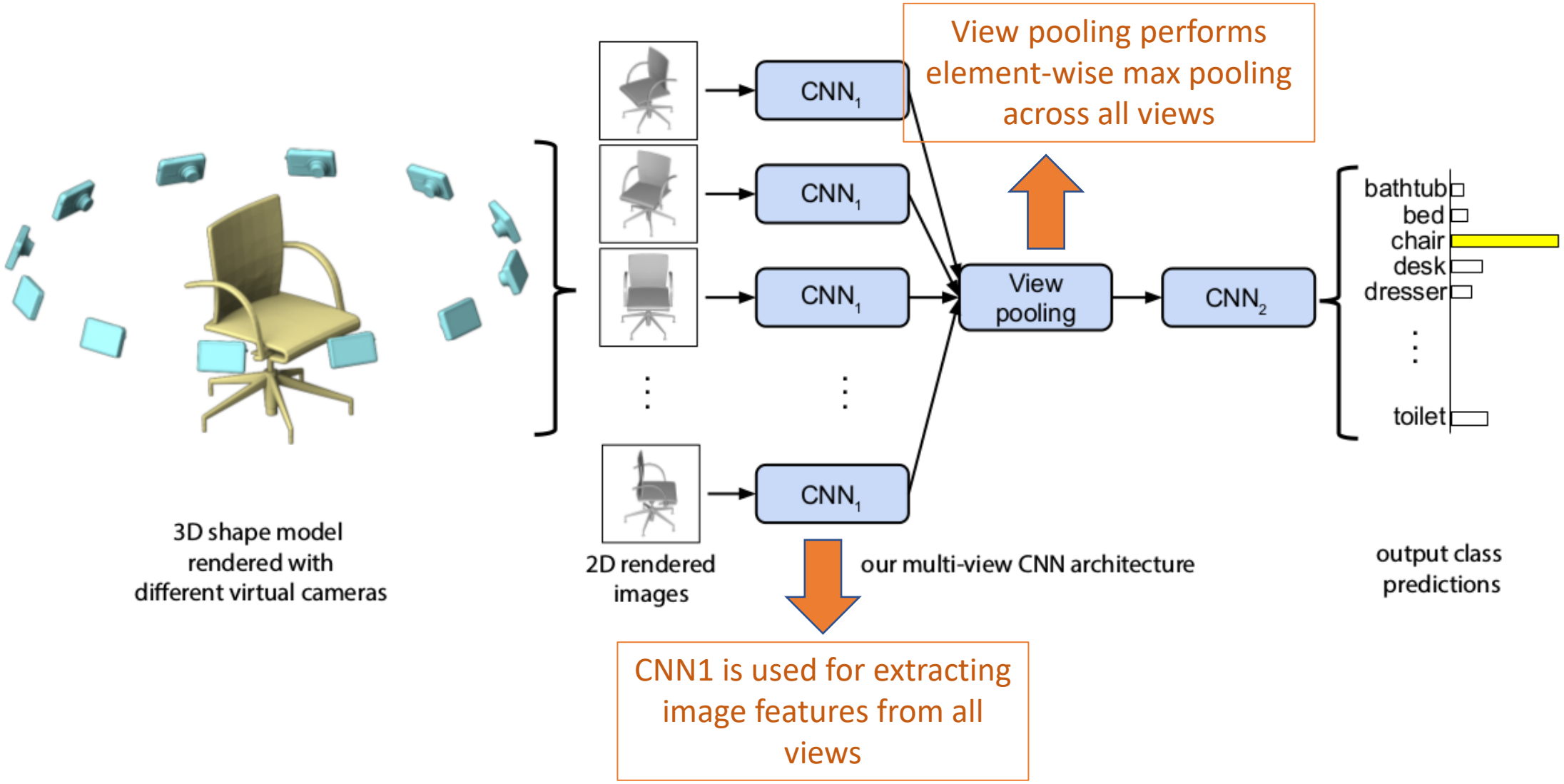
Volumetric Deep Learning: Classification



CNN1 is used for extracting image features from all views

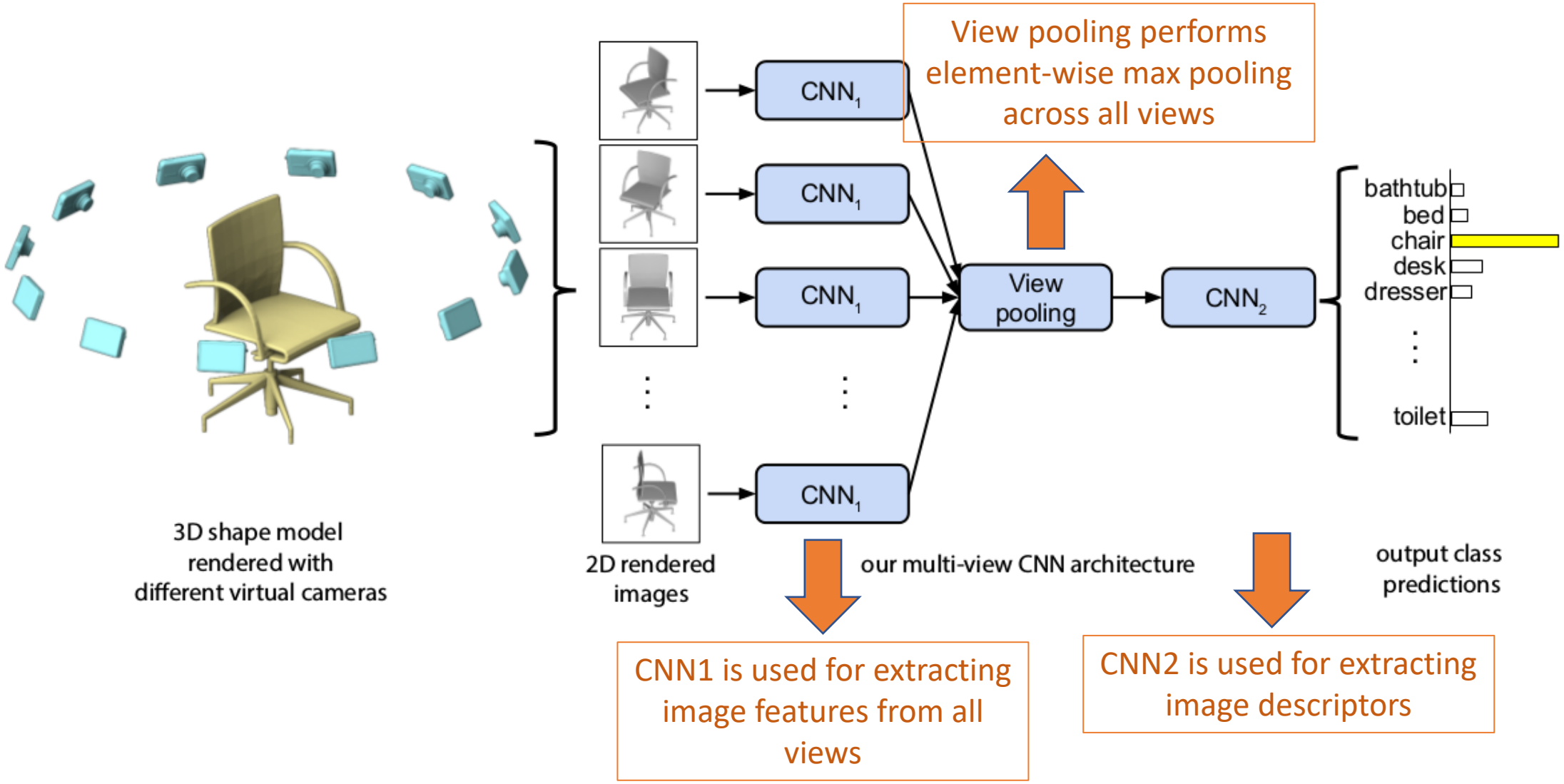
Multi-view Convolutional Neural Networks for 3D Shape Recognition, Hang Su, Subhransu Maji, Evangelos Kalogerakis Erik Learned-Miller, ICCV 2015

Volumetric Deep Learning: Classification



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

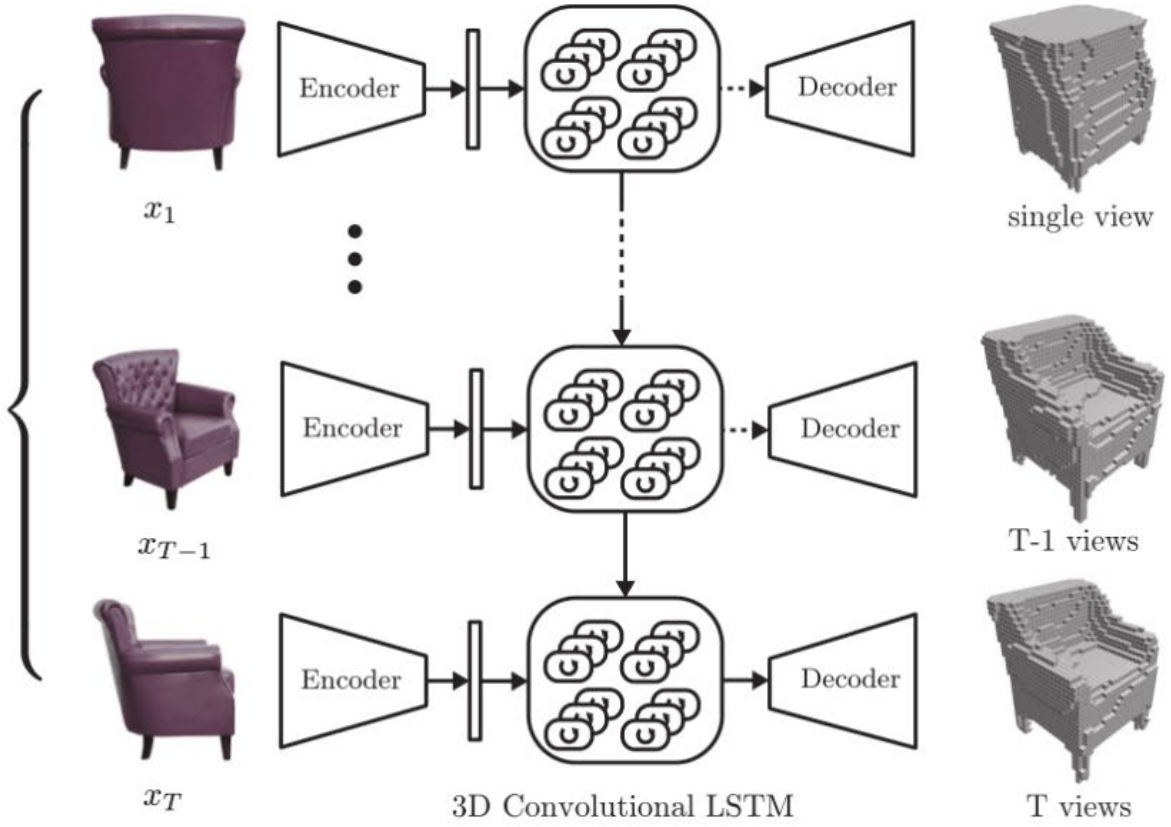
Volumetric Deep Learning: Classification



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

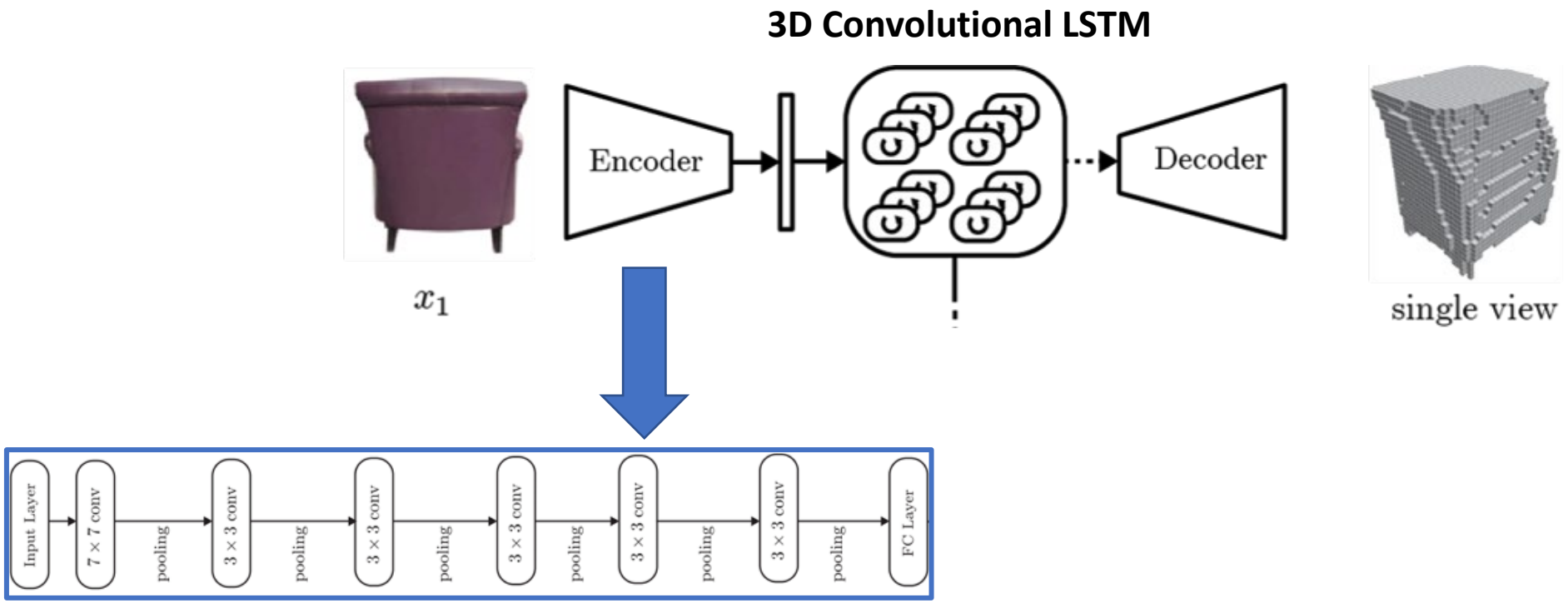
Volumetric Deep Learning: Reconstruction

The network **takes a sequence of images from arbitrary (uncalibrated) viewpoints** as input and **generates voxelized 3D reconstruction** as an output. The reconstruction is incrementally refined as the network sees more views of the object.



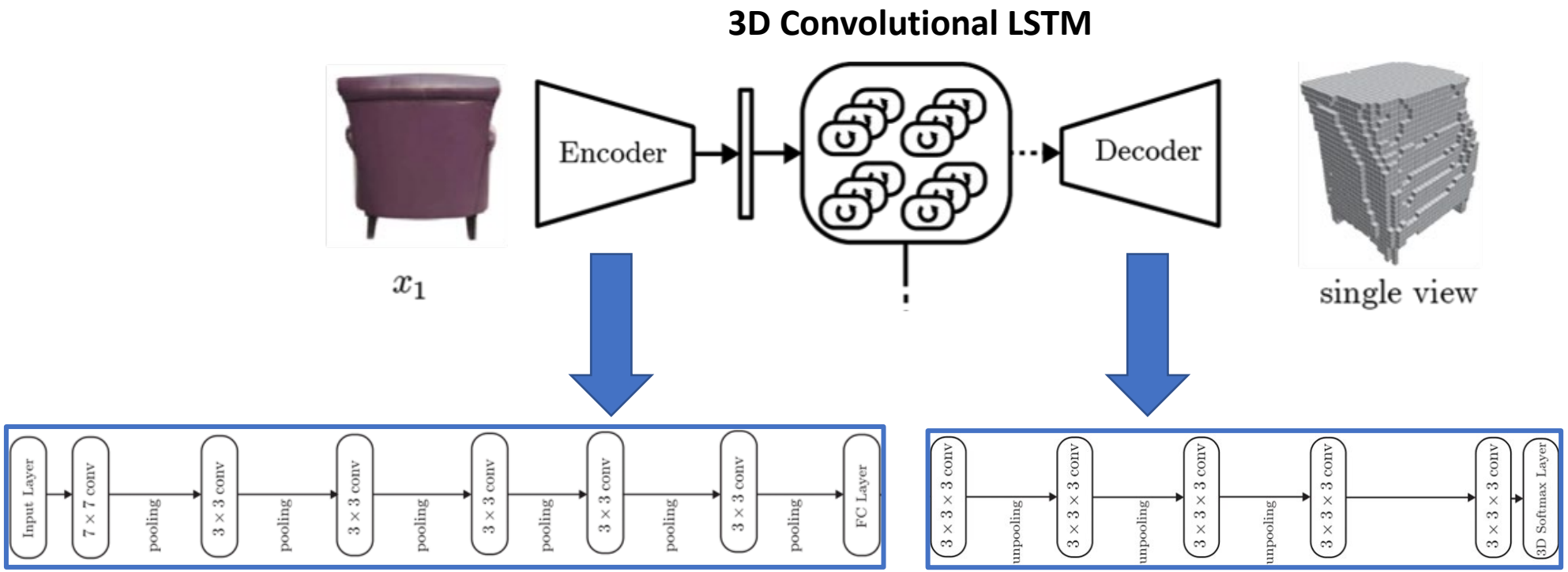
3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction, Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen Silvio Savarese, ECCV 2016

Volumetric Deep Learning: Reconstruction



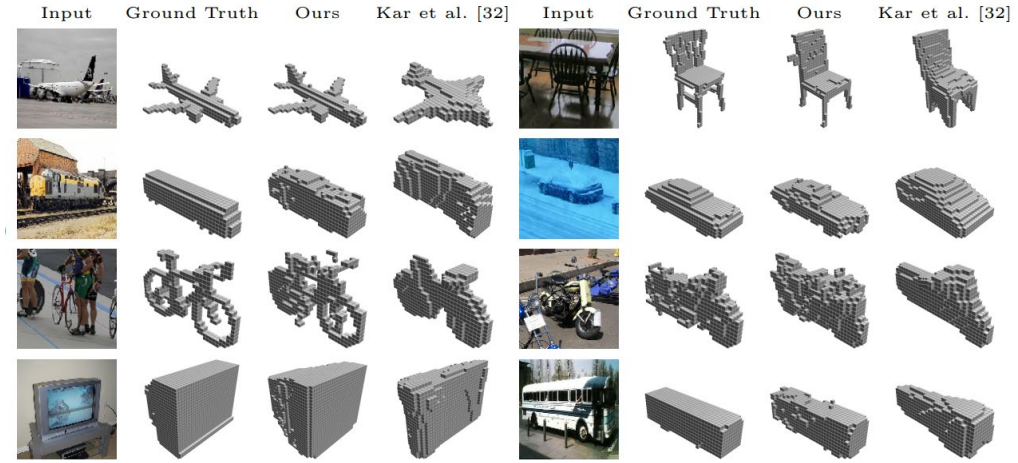
3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction, Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen Silvio Savarese, ECCV 2016

Volumetric Deep Learning: Reconstruction



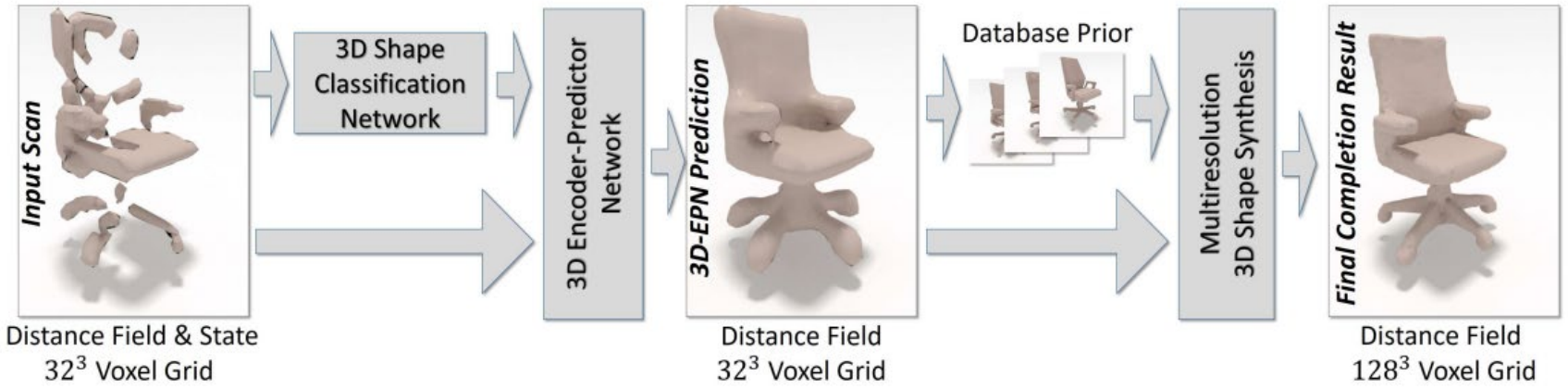
3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction, Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen Silvio Savarese, ECCV 2016

Volumetric Deep Learning: Reconstruction



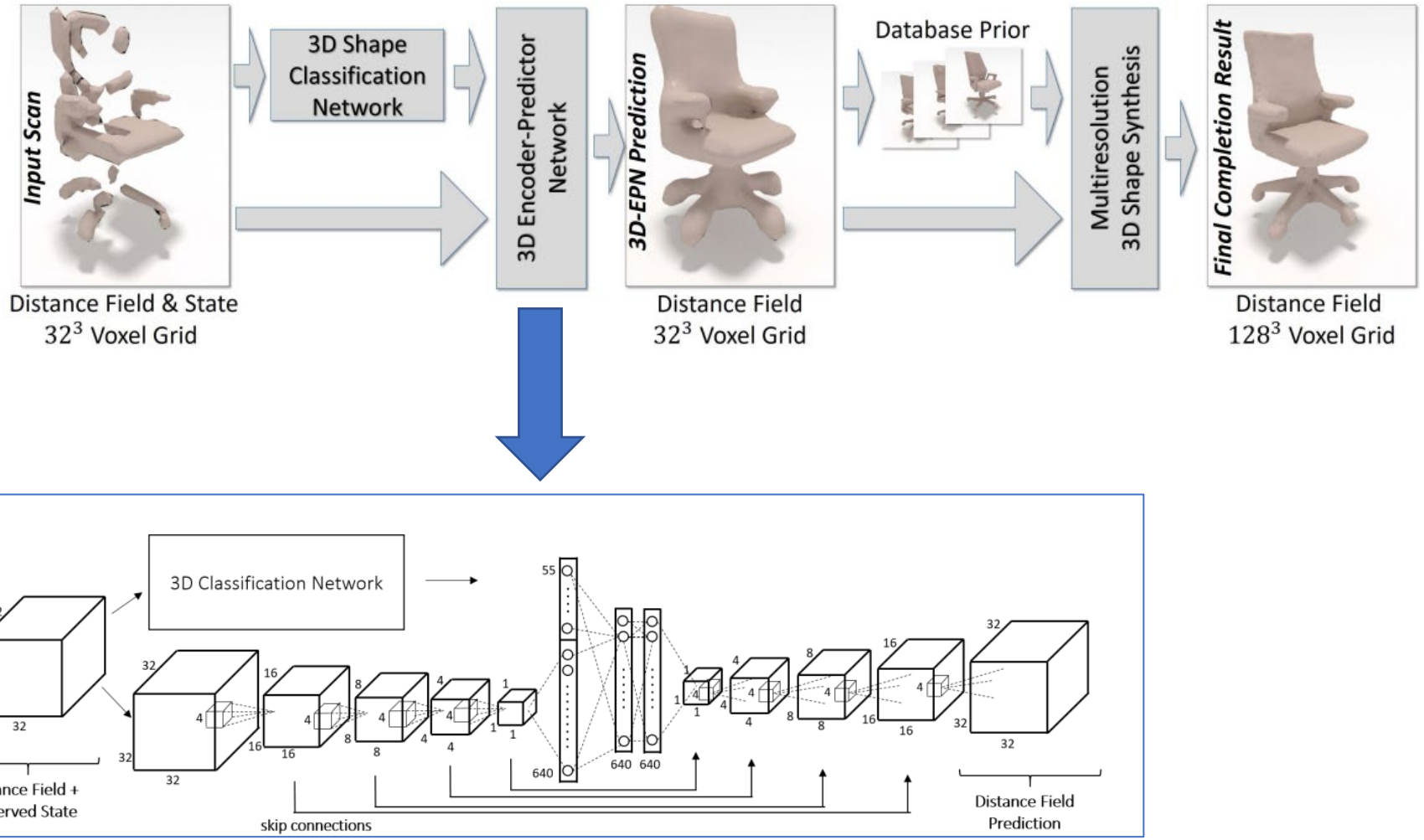
3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction, Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen Silvio Savarese, ECCV 2016

Volumetric Deep Learning: Completion



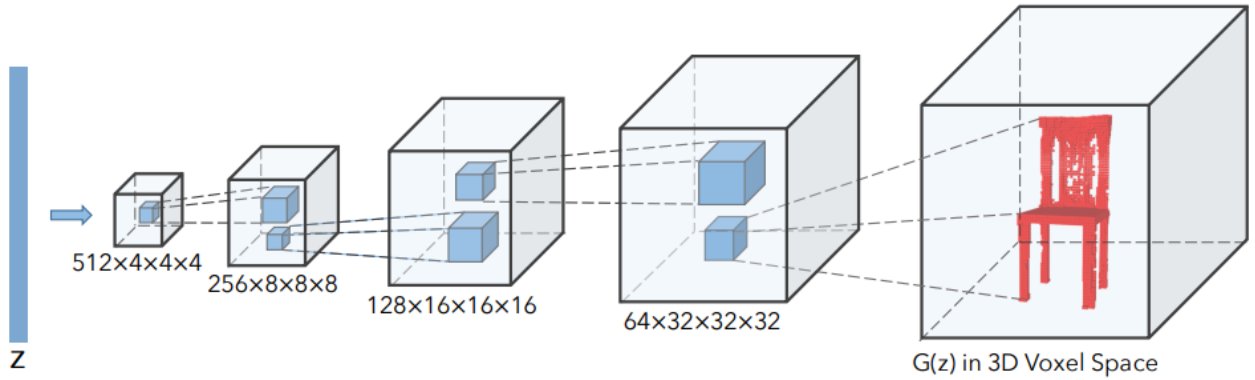
Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis, Angela Dai, Charles Ruizhongtai Qi, Matthias Nießner, CVPR 2017

Volumetric Deep Learning: Completion

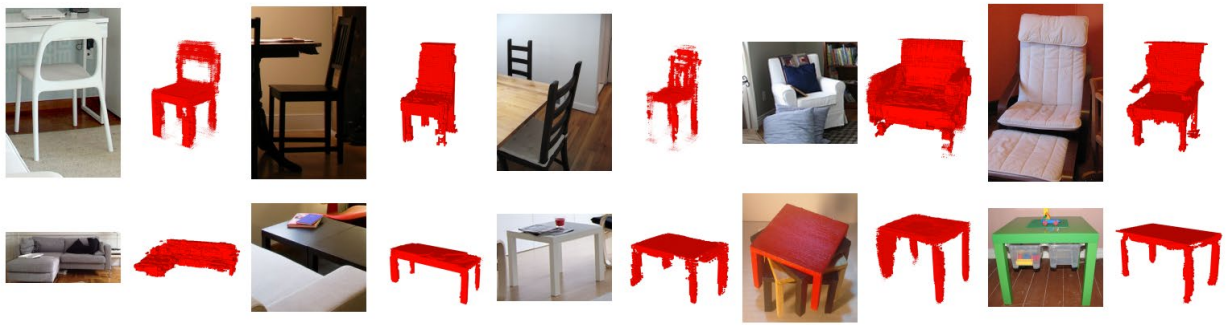


Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis, Angela Dai, Charles Ruizhongtai Qi, Matthias Nießner, CVPR 2017

Volumetric Deep Learning: Generation



Random Samples

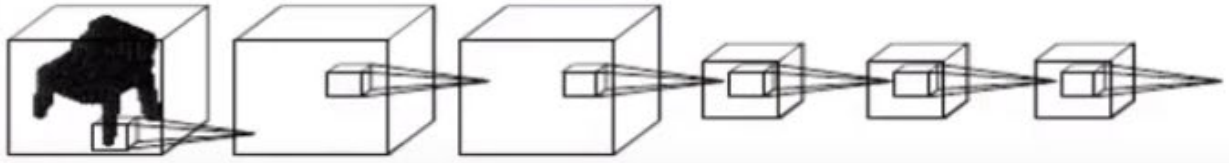


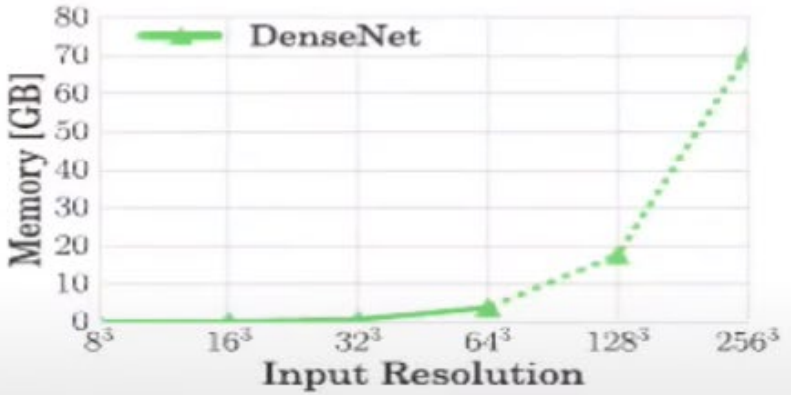
Singe-view 3D Reconstruction

Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling, Jiajun Wu* Chengkai Zhang* Tianfan Xue, William T. Freeman, Joshua B. Tenenbaum, NeurIPS 2016

Issue with Volumetric Representations

Memory requirements increase cubically with respect to the input resolution.

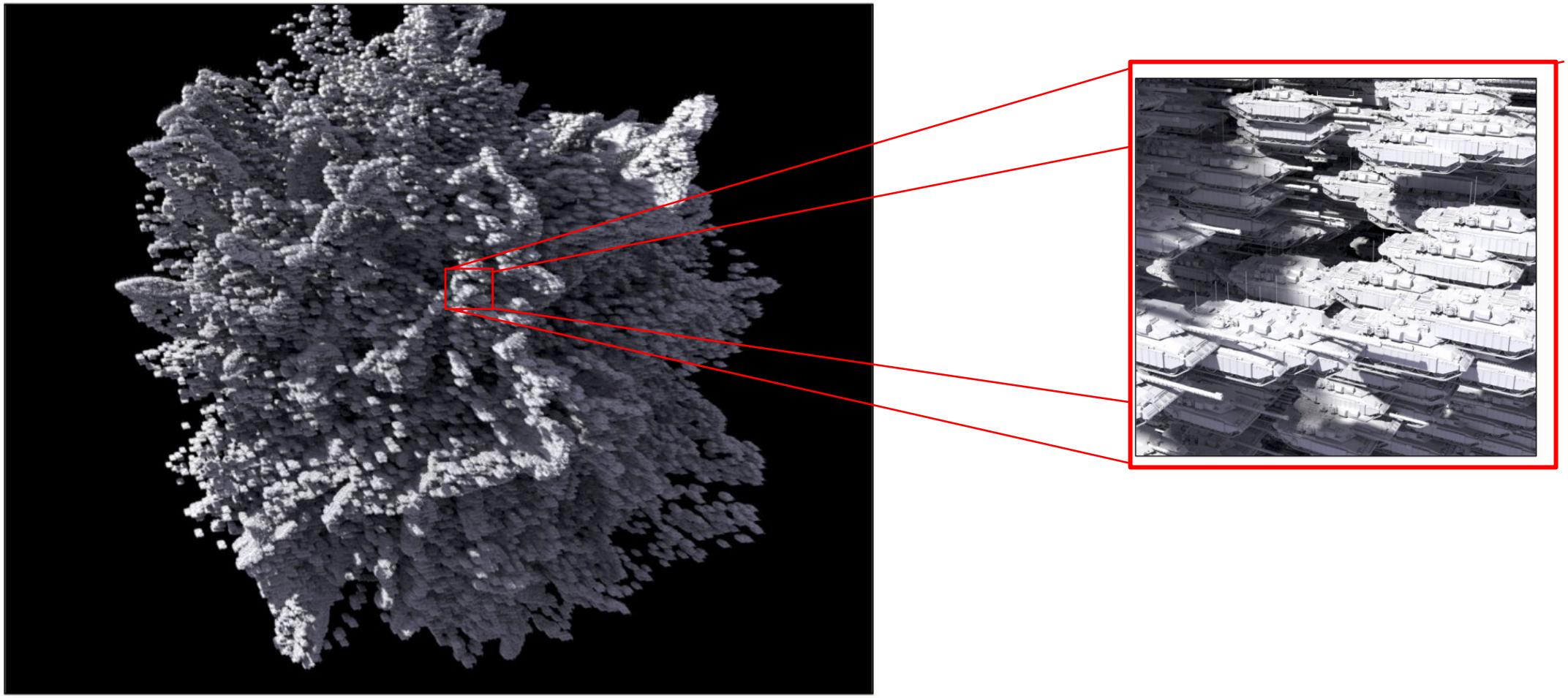




OctNet: Learning Deep 3D Representations at High Resolutions, Gernot Riegler and Ali Osman Ulusoy and Andreas Geiger, CVPR 2017

Efficient Data Structures for Volumetric Data

Why we need Efficient Data Structures?

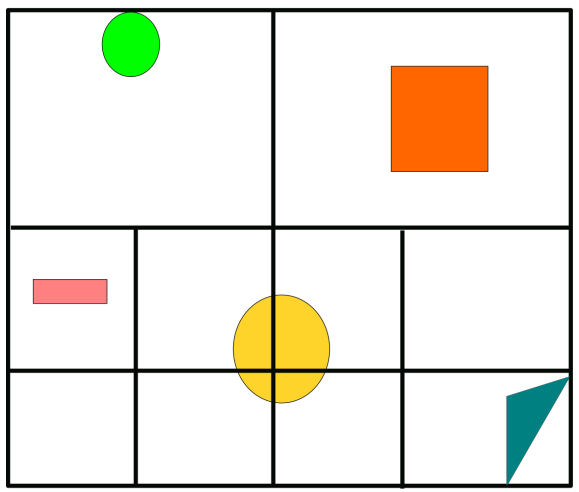


This mesh consists of 245 billion polygonal meshes

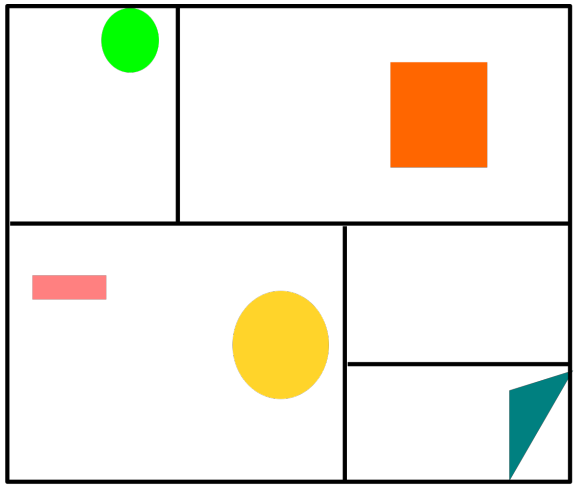
Efficient Data Structures for Volumetric Data

There are various data structures for efficiently partitioning the 3D space such as:

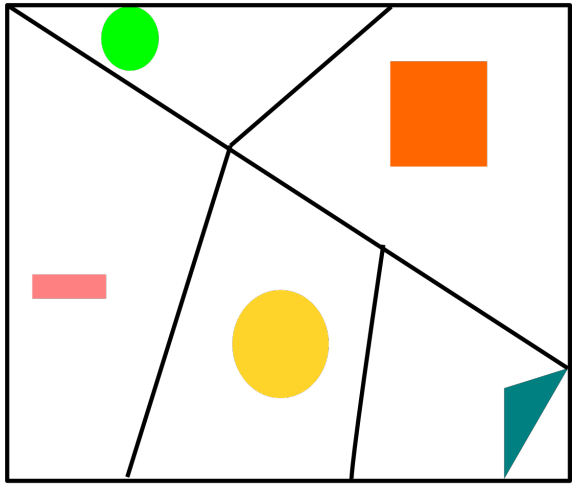
- Octrees
- Kd-trees
- BSP-trees



Octree



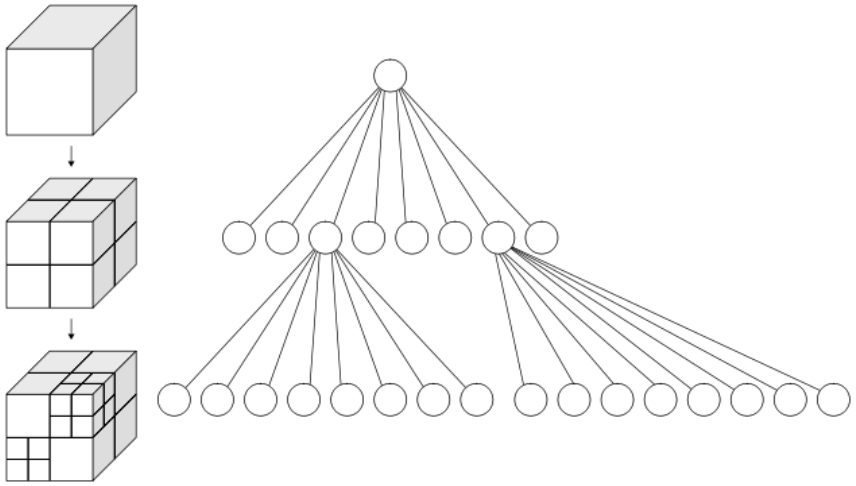
kd-tree

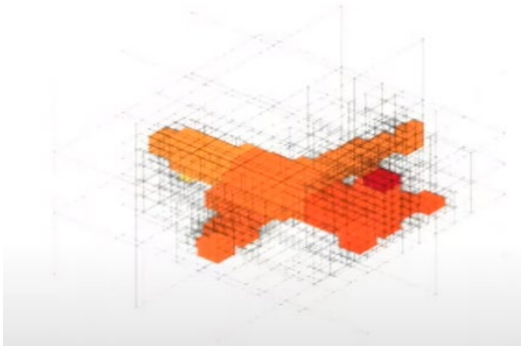


bsp-tree

Octrees

An **octree is a tree data structure** in which **each internal node has exactly eight children**. Octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants



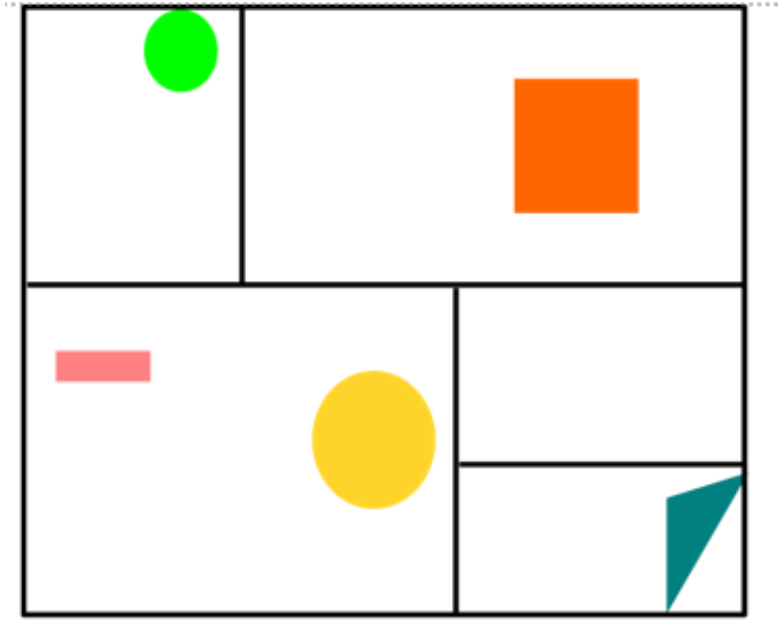


An octree partitions the 3D space by recursively subdividing it into octants. **By subdividing only the cells which contain relevant information** (e.g., cells crossing a surface boundary or cells containing one or more 3D points) **storage can be allocated adaptively**. Densely populated regions are modeled with high accuracy (i.e., using small cells) while empty regions are summarized by large cells in the octree.

k-d tree

A k-d tree (k-dimensional tree) is a space partitioning data structure for organizing points in a k-dimensional space.

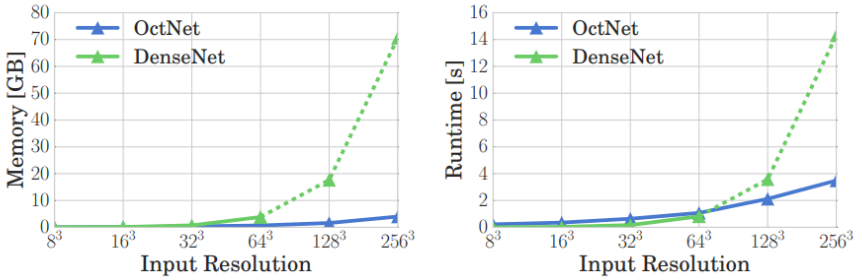
- Start by computing the bounding box of the scene
- **Recursively split each cell using an axis-aligned plane**
- Continue until termination criterion, e.g. maximum depth of minimum number of objects in each cell.



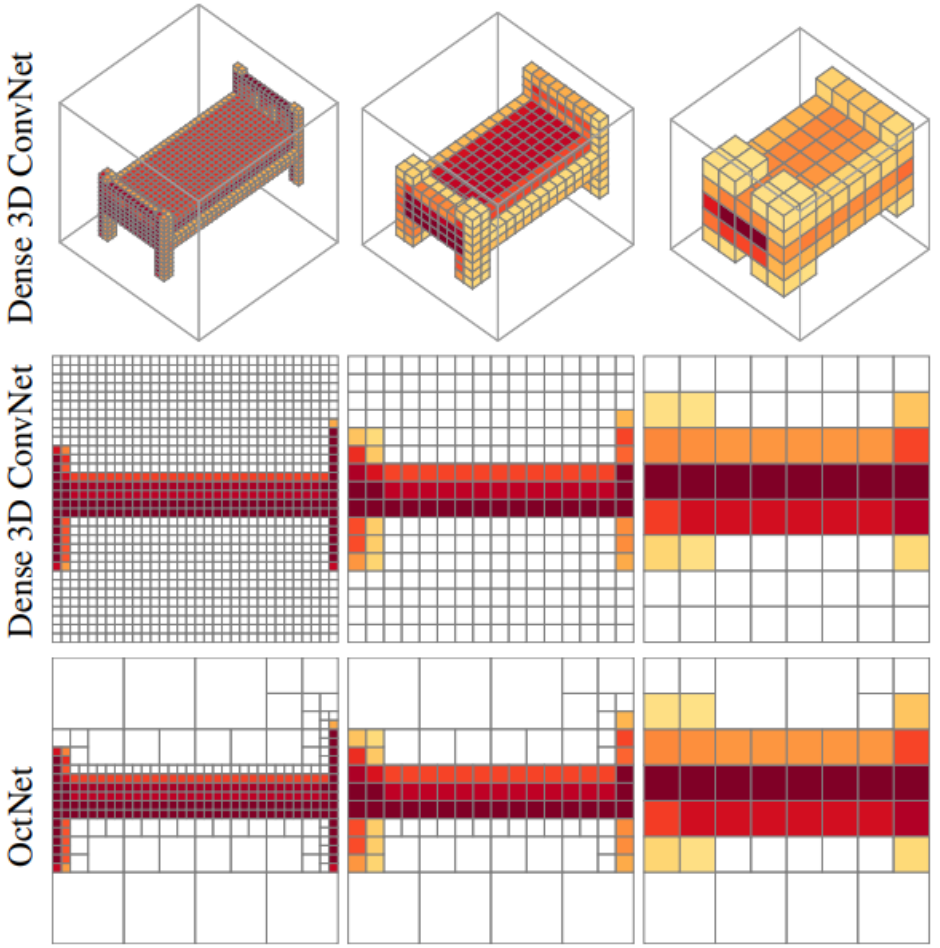
kd-tree

Volumetric Deep Learning with Octrees

- **OctNet is a 3D convolutional network** that exploits this sparsity property, by hierarchically partitioning the 3D space into a set of unbalanced octrees.
- Instead of representing the entire high resolution 3D input with a single unbalanced octree, they propose to use multiple shallow octrees of maximal depth.
- Shallow octrees can be efficiently encoded using bit-strings
- Using these representation, they define various convolutional layers, mooling and unpooling layers,



(a) Memory (b) Runtime

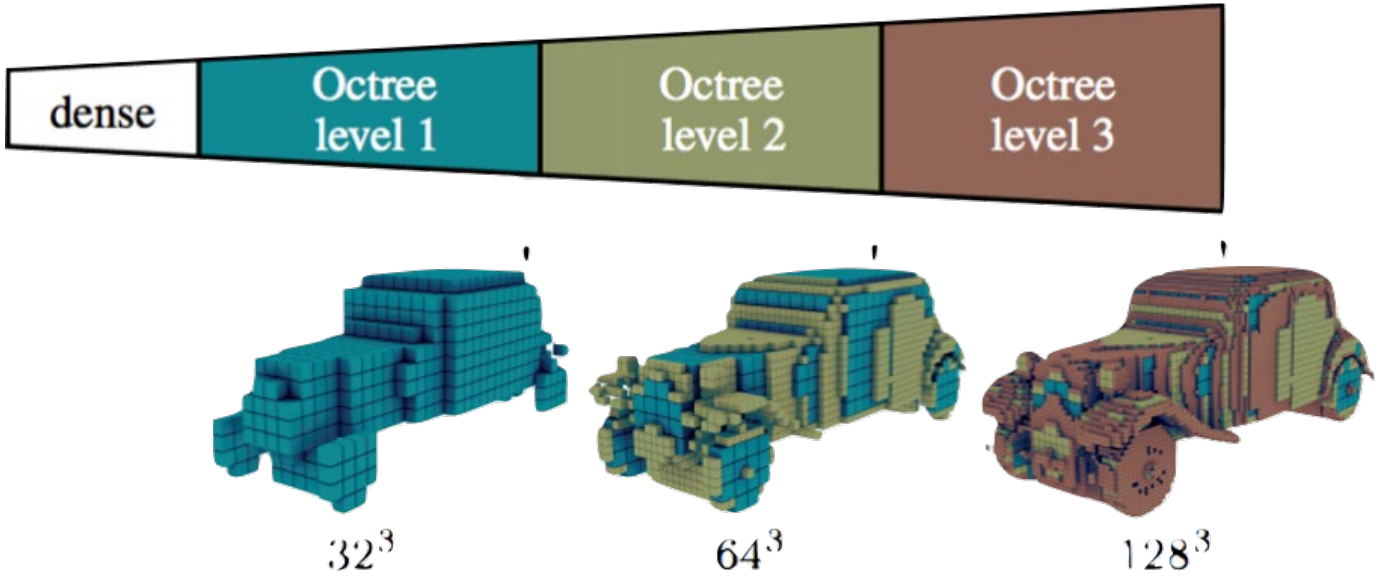


(a) Layer 1: 32^3 (b) Layer 2: 16^3 (c) Layer 3: 8^3

OctNet: Learning Deep 3D Representations at High Resolutions, Gernot Riegler and Ali Osman Ulusoy and Andreas Geiger, CVPR 2017

Volumetric Deep Learning with Octrees

The proposed model **represents its volumetric output as an octree**. Initially a rough low-resolution structure is estimated, which is then gradually refined to a desired high resolution. At each level only a sparse set of spatial locations is predicted.



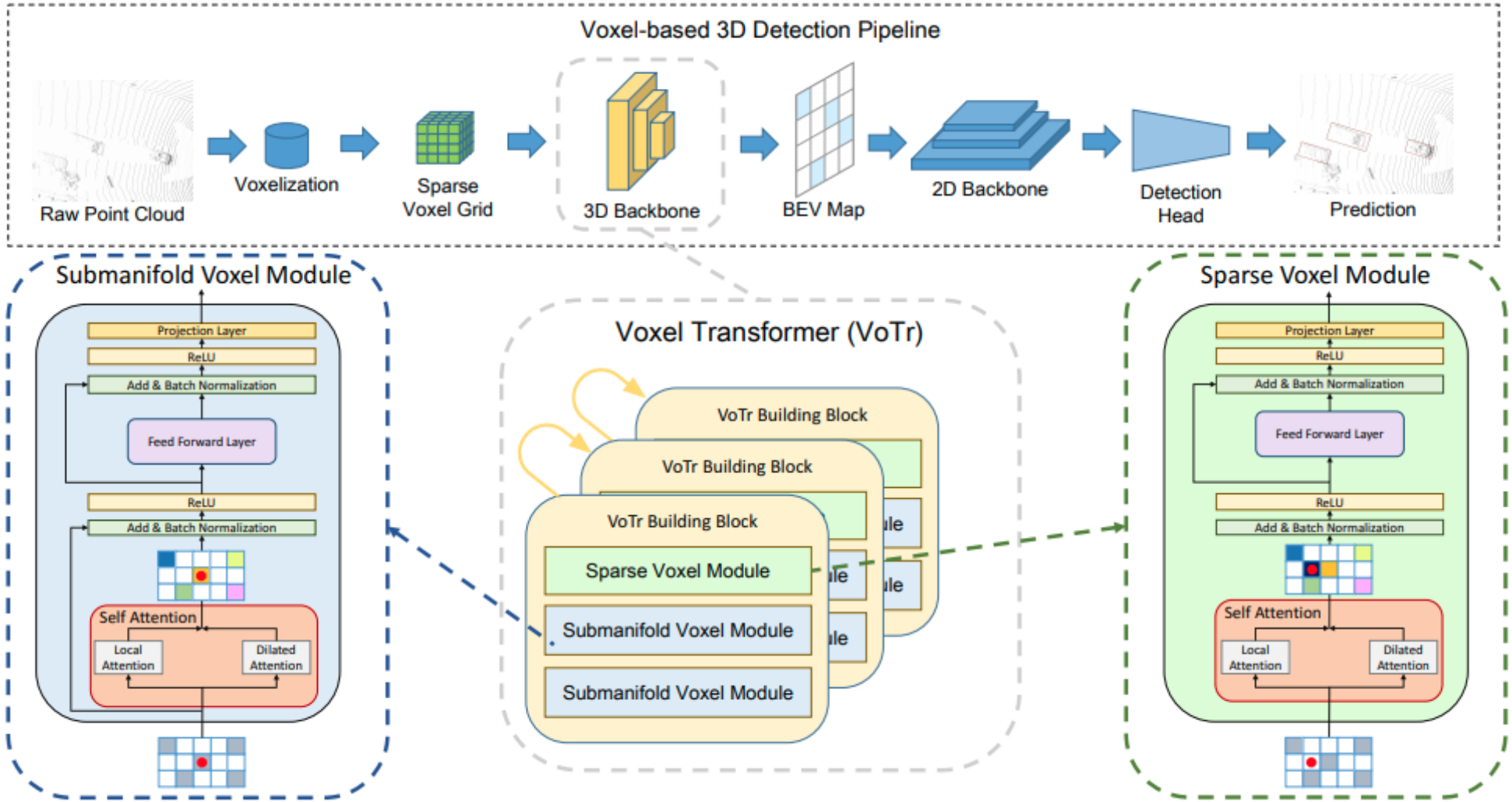
Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs, Maxim Tatarchenko, Alexey Dosovitskiy, Thomas Brox, ICCV 2017

Volumetric Deep Learning with Octrees

- **Advantages:**
  - Address the O(N^3) Storage / Computation Explosion
  - Allow 3D CNNs to work with Larger N --> better performance

- **Disadvantages:**
  - Still have computations over empty voxels
  - Not very flexible given the specific data structure (e.g. the octree)
  - Hard to implement

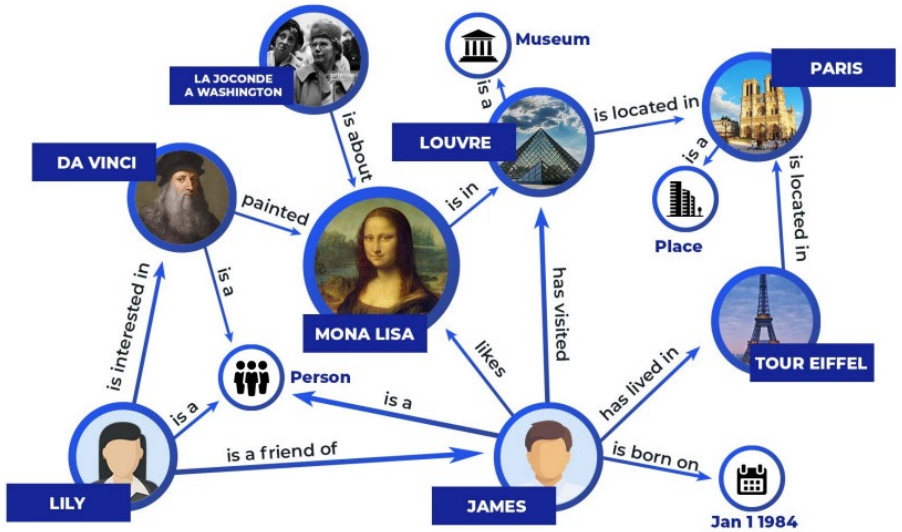
VoTr: Voxel Transformer

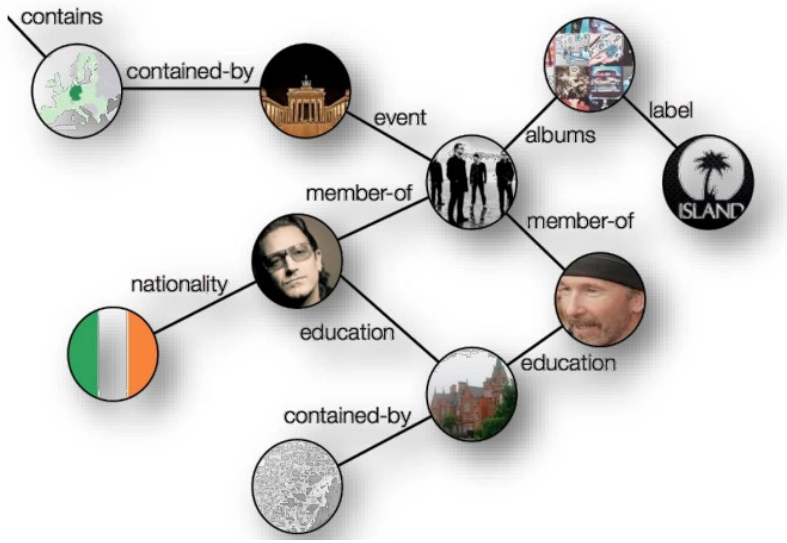


Voxel Transformer for 3D Object Detection, *Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng*, ICCV 2021

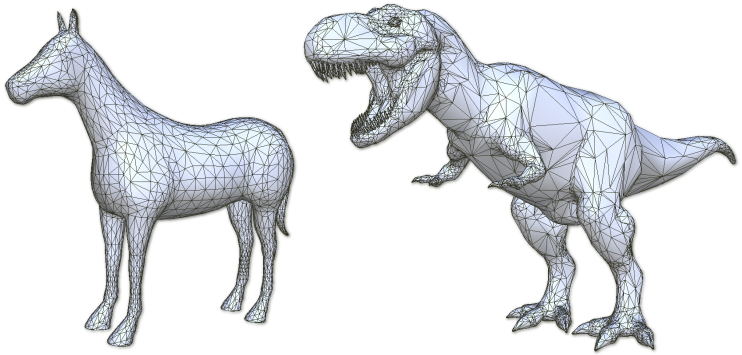
Graph Convolutional Neural Networks & Geodesic Convolutional Neural Networks

Graph Representations



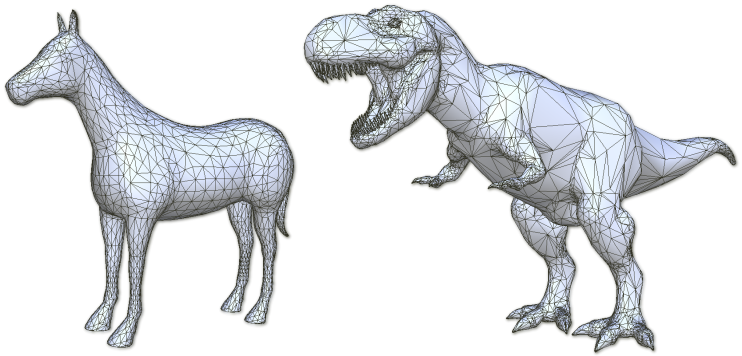


Graph Representations

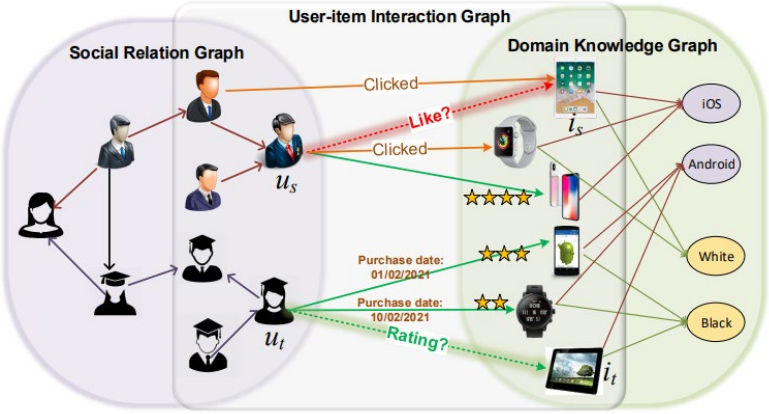


3D Meshes represent the object's geometry using a graph.

Graph Representations

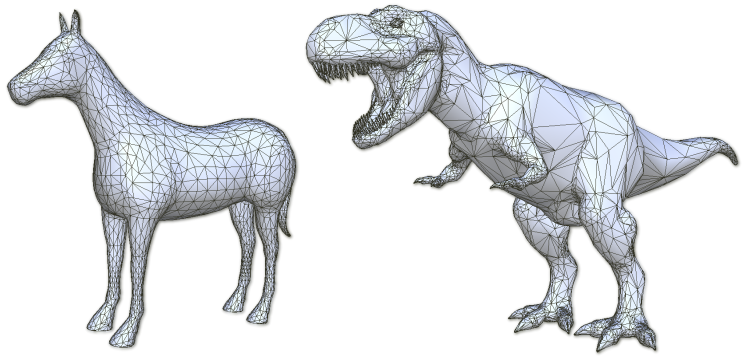


3D Meshes represent the object's geometry using a graph.

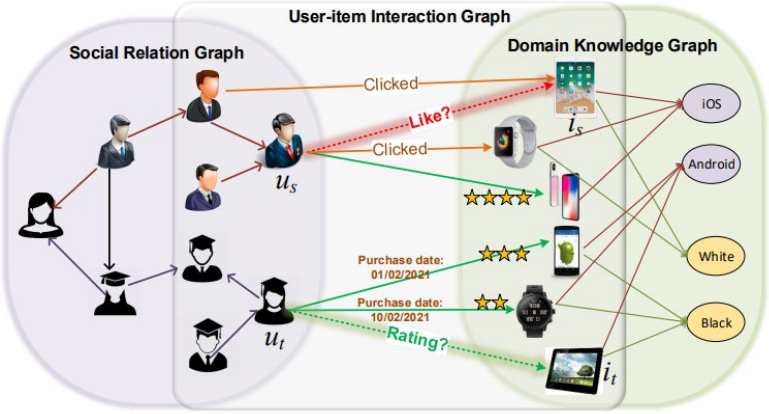


Graphs are used for performing **recommendations**.

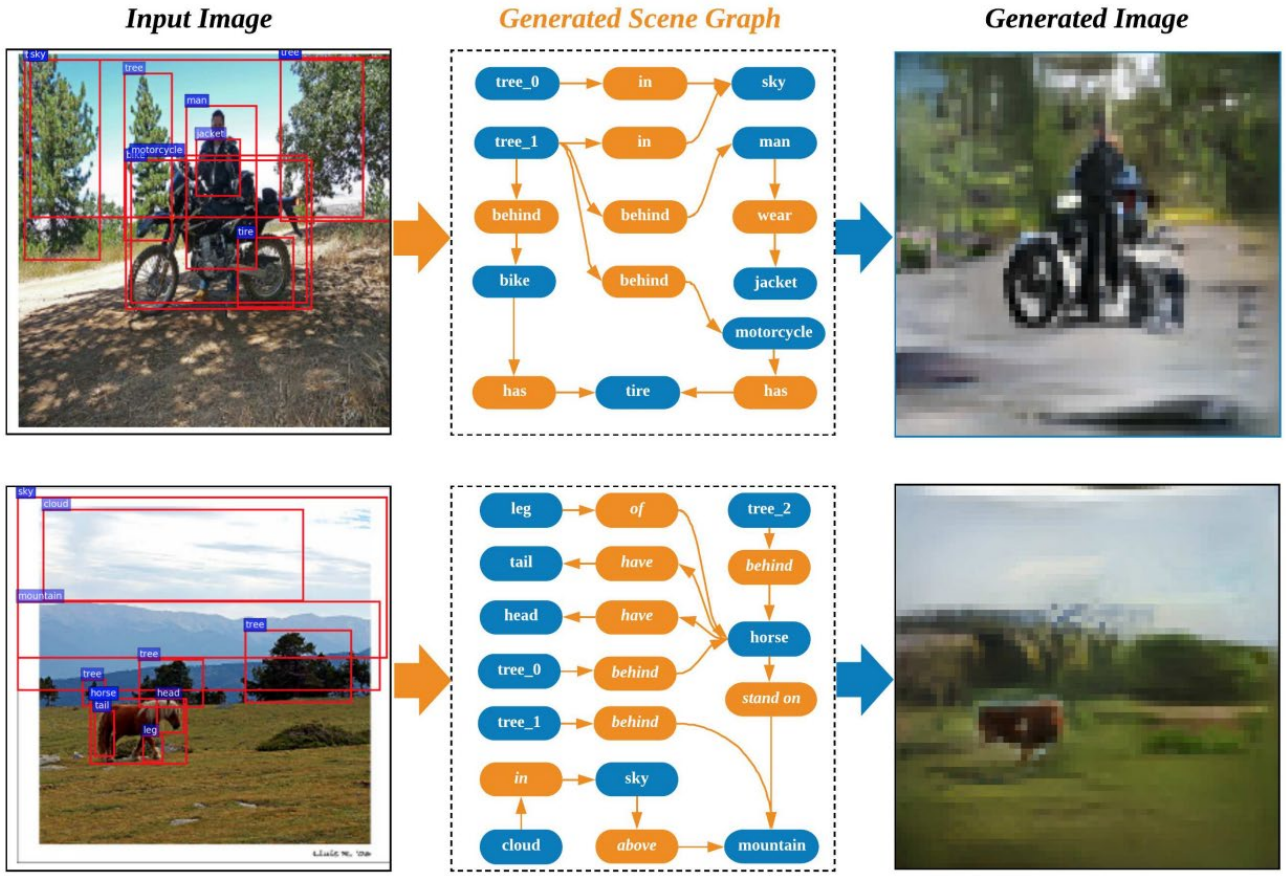
Graph Representations



3D Meshes represent the object's geometry using a graph.

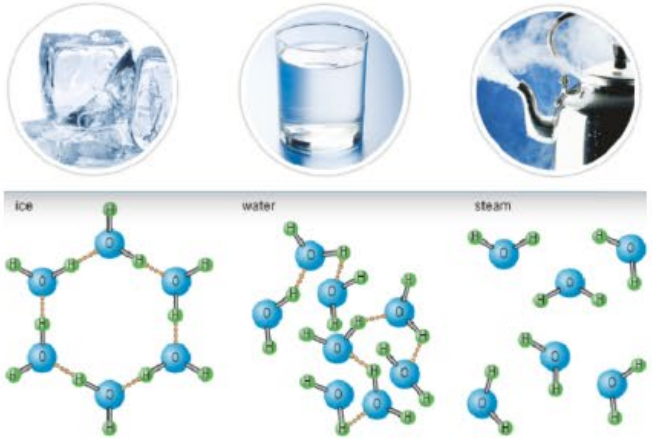


Graphs are used for performing **recommendations**.

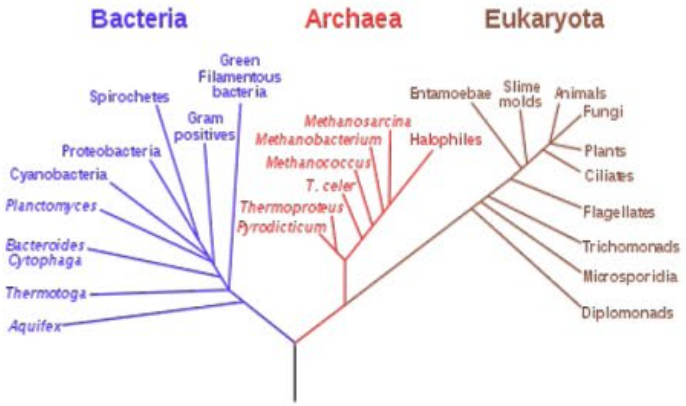


Scene graphs can be used for generating new and describing existing scenes.

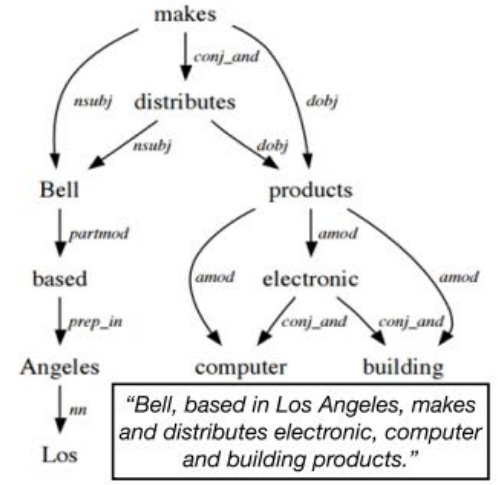
Why do we need Graph Neural Networks?



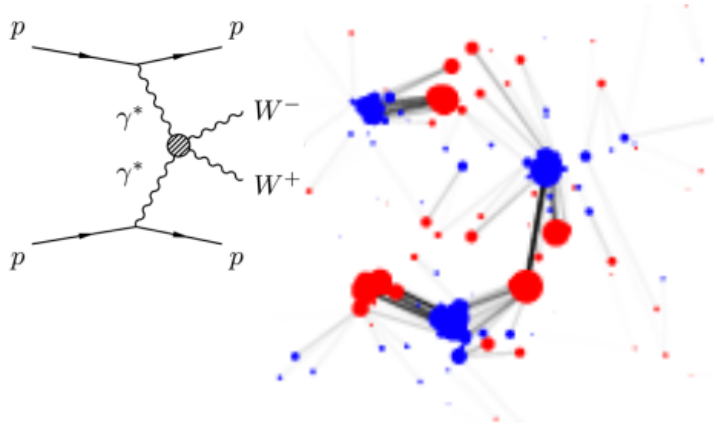
Molecules Predictions



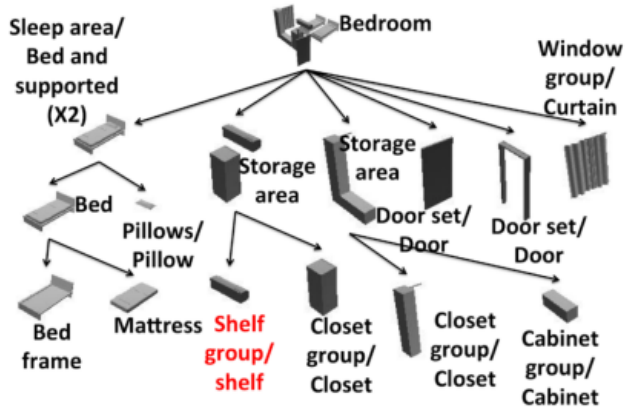
Biological Species Categorizations



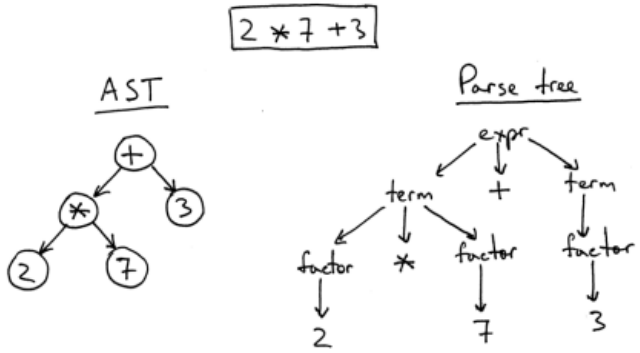
Natural Language Processing



Sub-atomic particles



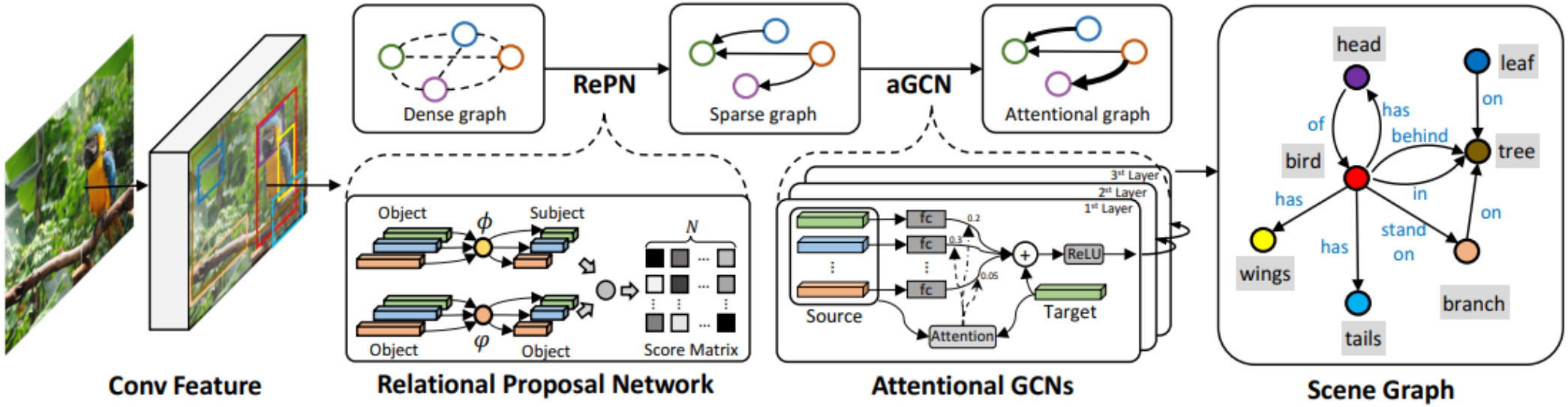
Everyday scenes



Code analysis

Slide Credit: Peter Battaglia

Scene Graph Generation



Goal: Infer the graph that captures relationships between objects in the image

- Starting from an image, we run an o**bject detector** and we **get object proposals.**
- Initially, we start from a complete graph, which is subsequently, pruned based on how related two nodes in the graph (objects in the scene) are.
- Using the Attentional GCNs, they try to infer a rich representation that can also reason about the relationships between objects in the scene.

Graph R-CNN for Scene Graph Generation, Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra and Devi Parikh**,** ECCV 2018

Deep Learning on Graphs and Manifolds

Deep Learning on Graphs:

- Spectral CNN, by Bruna et al.
- Smooth Spectral CNN, by Henaff et al.
- Chebysev Spectral CNN, by Defferrard et al.
- Graph Convolutional Network (GCN) by Kipf et al.
- Diffusion CNN (DCNN) by Atwood and Towsley et al.

- **Deep Learning on Manifolds:**
  - Geodesic CNN (GCNN) by Masci et al.
  - Anisotropic CNN by Boscanni (ACNN) et al.
  - Mixture model CNNs (MoNet) by Monti et al.

A Comprehensive Survey on Graph Neural Networks, Zonghan Wu, Shirui Pan, Member, IEEE, Fengwen Chen, Guodong Long, Chengqi Zhang

Deep Learning on Graphs and Manifolds

Deep Learning on Graphs:

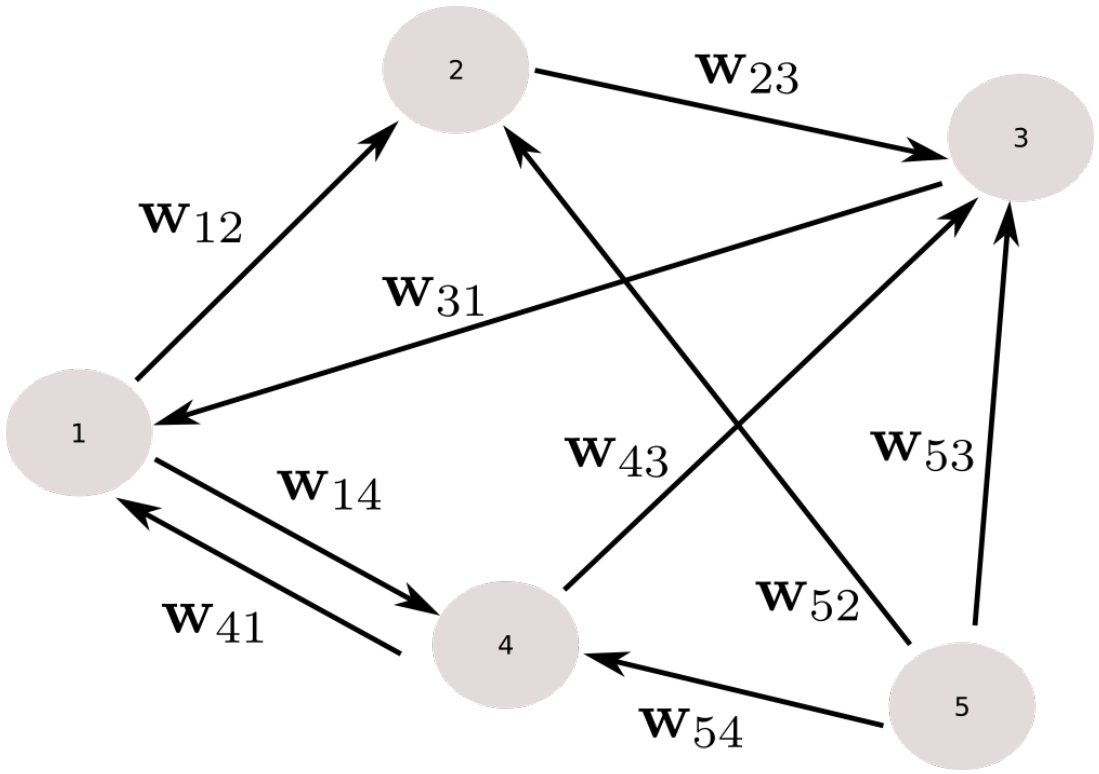
- Spectral CNN, by Bruna et al.
- Smooth Spectral CNN, by Henaff et al.
- Chebysev Spectral CNN, by Defferrard et al.
- Graph Convolutional Network (GCN) by Kipf et al.
- Diffusion CNN (DCNN) by Atwood and Towsley et al.

- **Deep Learning on Manifolds:**
  - Geodesic CNN (GCNN) by Masci et al.
  - Anisotropic CNN by Boscanni (ACNN) et al.
  - Mixture model CNNs (MoNet) by Monti et al.

A Comprehensive Survey on Graph Neural Networks, Zonghan Wu, Shirui Pan, Member, IEEE, Fengwen Chen, Guodong Long, Chengqi Zhang

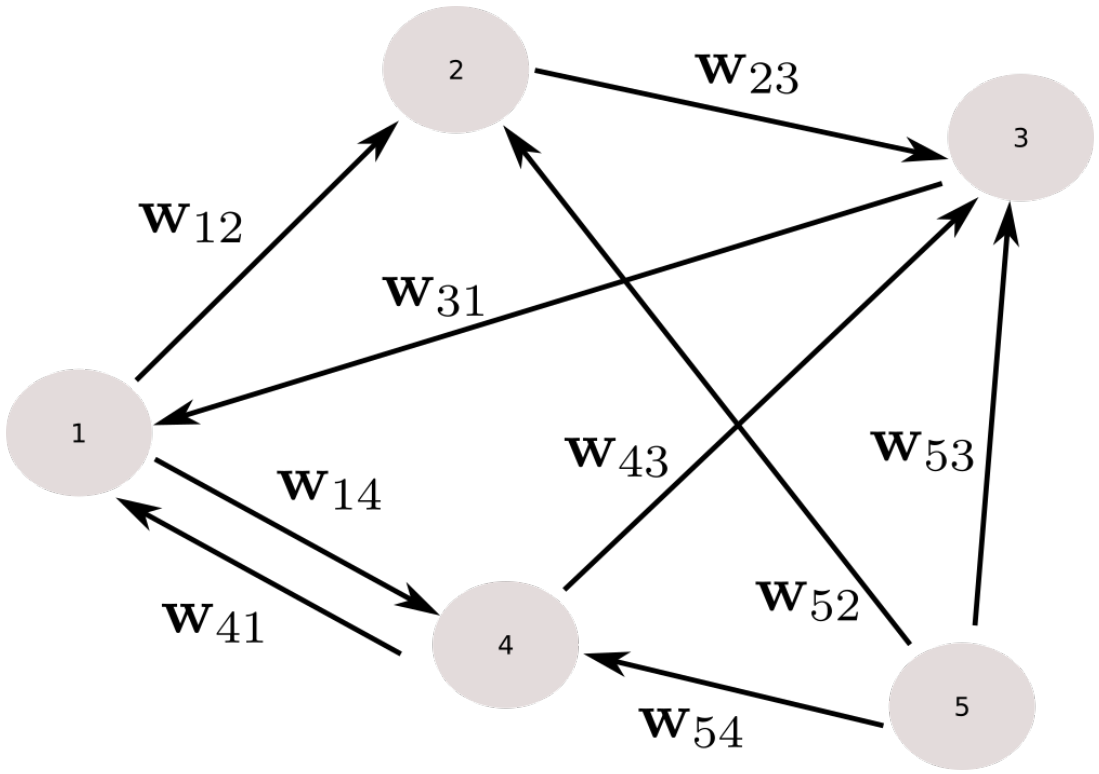
Quick Graph Recap

Directed Graphs



A **directed graph** is a graph, where the edges have a direction. Typically, **a directed graph is represented as G = {V, E, W}**, where V are the vertices, E are the edges and W are the weights:

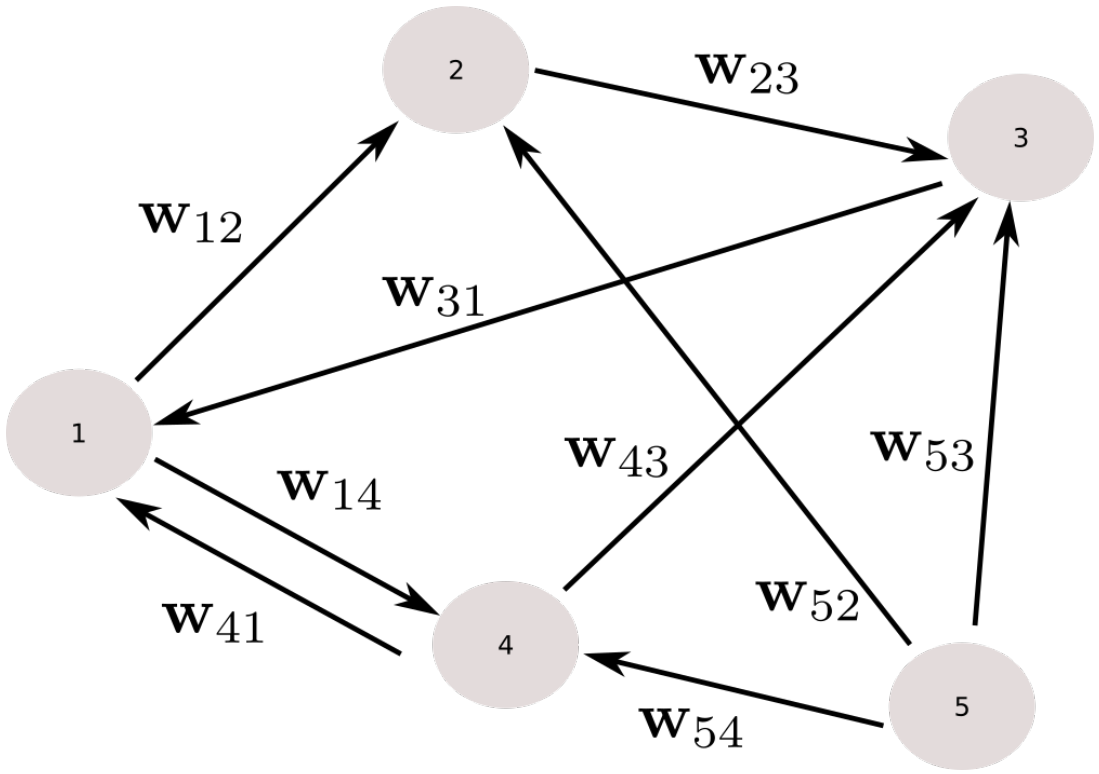
Directed Graphs



A **directed graph** is a graph, where the edges have a direction. Typically, **a directed graph is represented as G = {V, E, W}**, where V are the vertices, E are the edges and W are the weights:

- **The vertices or nodes** are defined as the set of N elements in the graph.

Directed Graphs

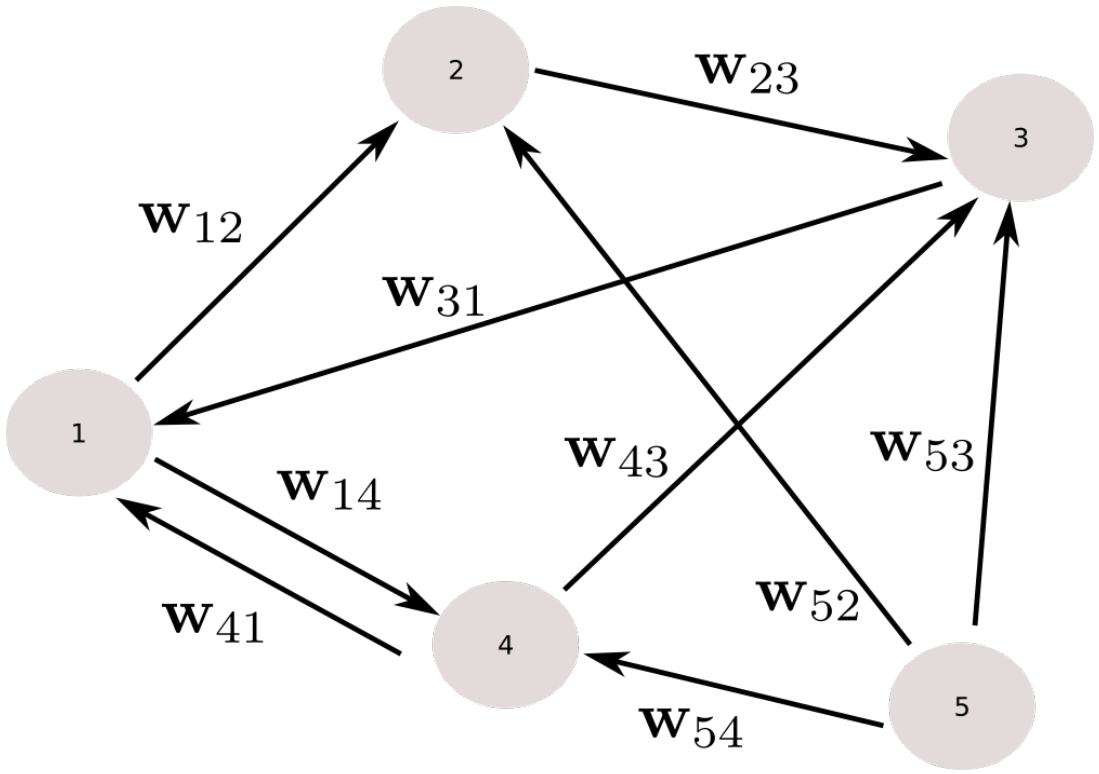


A **directed graph** is a graph, where the edges have a direction. Typically, **a directed graph is represented as G = {V, E, W}**, where V are the vertices, E are the edges and W are the weights:

- **The vertices or nodes** are defined as the set of N elements in the graph.
- **The edges** are defined as the ordered pairs of nodes in the graph. If there is an edge between the node i and node j, it means that i is influenced by j.

The (i,j) edge is different from the (j,i) edge!!!

Directed Graphs

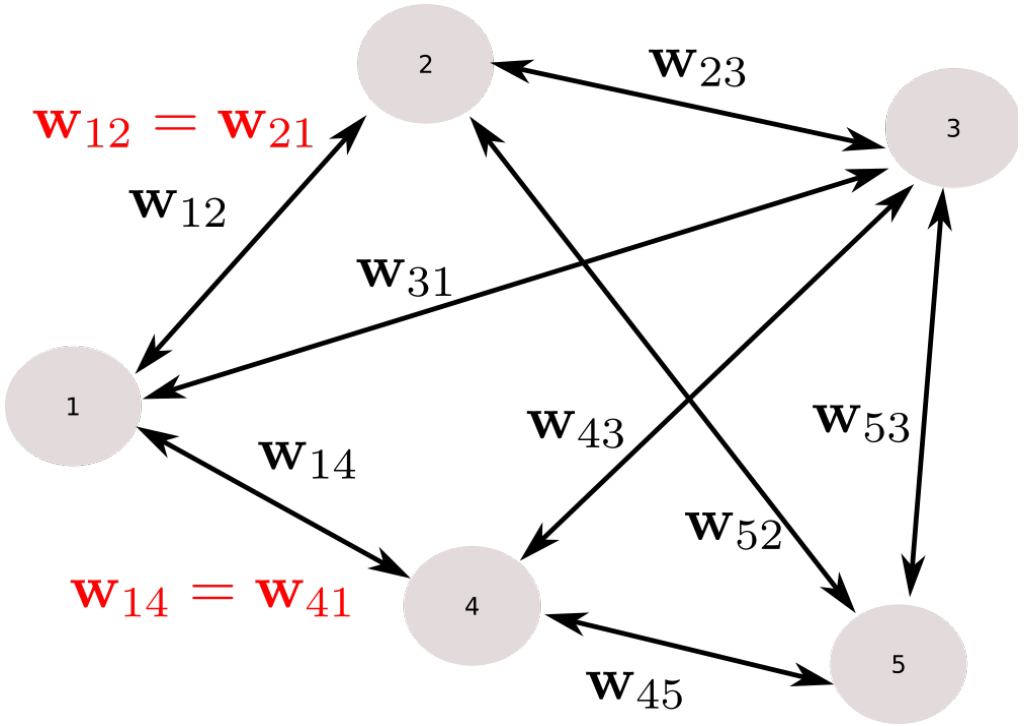


The weights on the (i,j) edge is different from the weights on the (j,i) edge !!!

A **directed graph** is a graph, where the edges have a direction. Typically, **a directed graph is represented as G = {V, E, W}**, where V are the vertices, E are the edges and W are the weights:

- **The vertices or nodes** are defined as the set of N elements in the graph.
- **The edges** are defined as the ordered pairs of nodes in the graph. If there is an edge between the node i and node j, it means that i is influenced by j.
- **The weights** are numbers associated with the edges and determine the strength of the influence between two nodes connected with an edge.

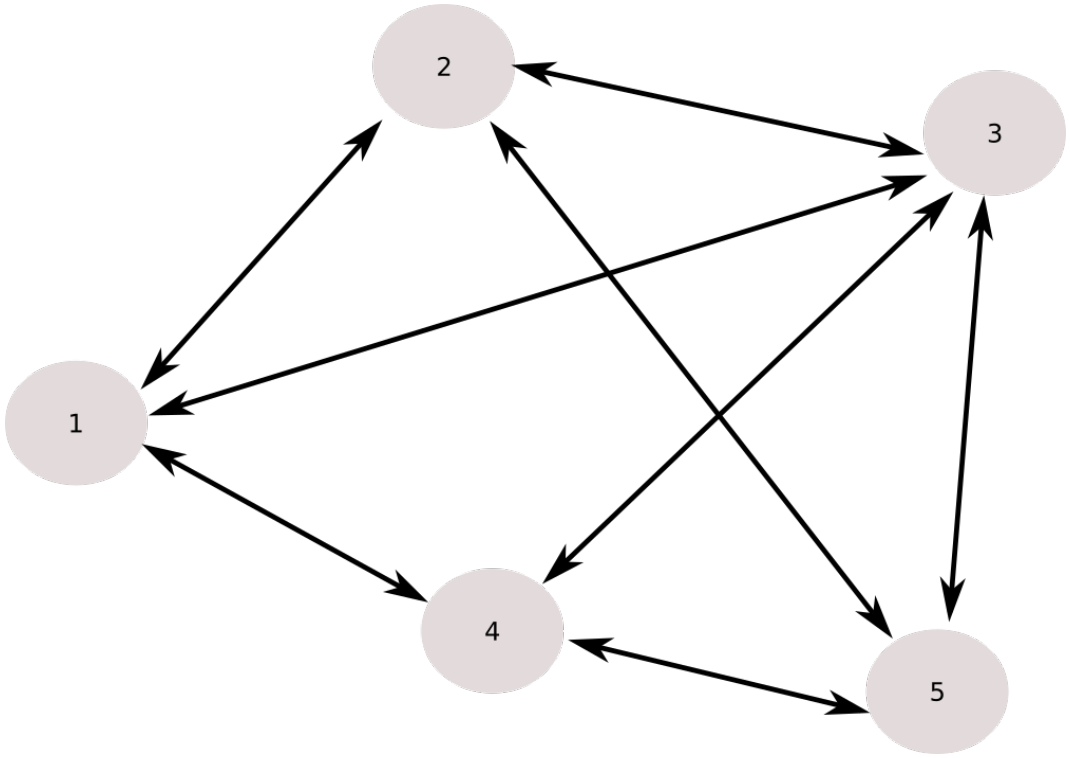
Undirected Graphs



An **undirected graph** is a graph, whose edges do not have a direction. Again, it is typically, **represented as G = {V, E, W}**, where V are the vertices, E are the edges and W are the weights.

- The edges are symmetric, namely if edge $(i, j) \in E$ then also $(j, i) \in E$
- The weights are symmetric, namely $\mathbf{w}_{ij} = \mathbf{w}_{ji}$ for all $(i, j) \in E$

Unweighted Graphs



An **unweighted graph** is a graph that does not have weights associated with its edges.

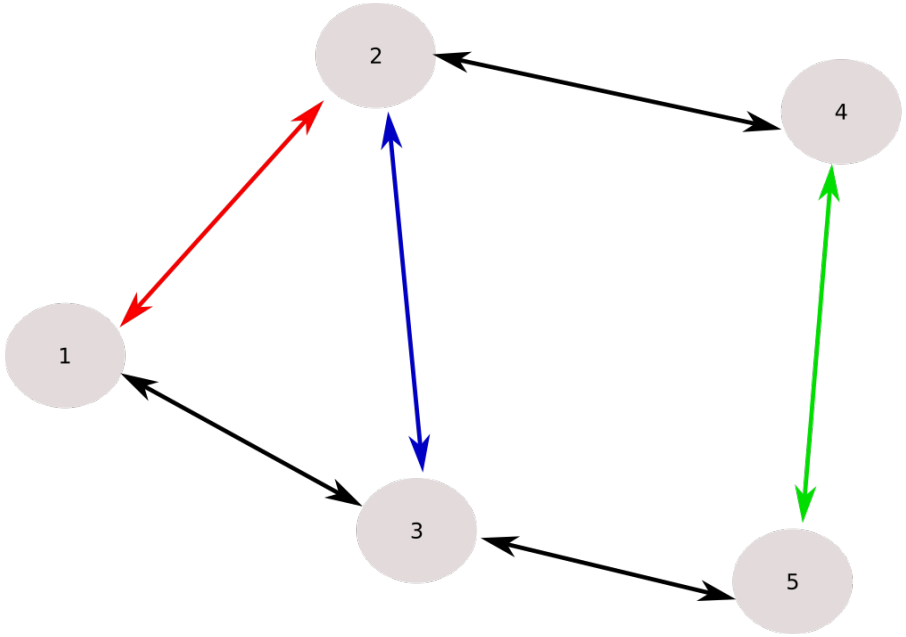
- Equivalently, we **typically say that all weights are one**, namely

$$\mathbf{w}_{ij} = 1 \forall (i,j) \in E$$

- The edge structure informs us about the proximity between two nodes, however, we do not know **how close they really are**.
- Unweighted graphs can be either directed or undirected.

Graphs as Adjacency Matrices

Starting from **an unweighted, undirected graph**



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

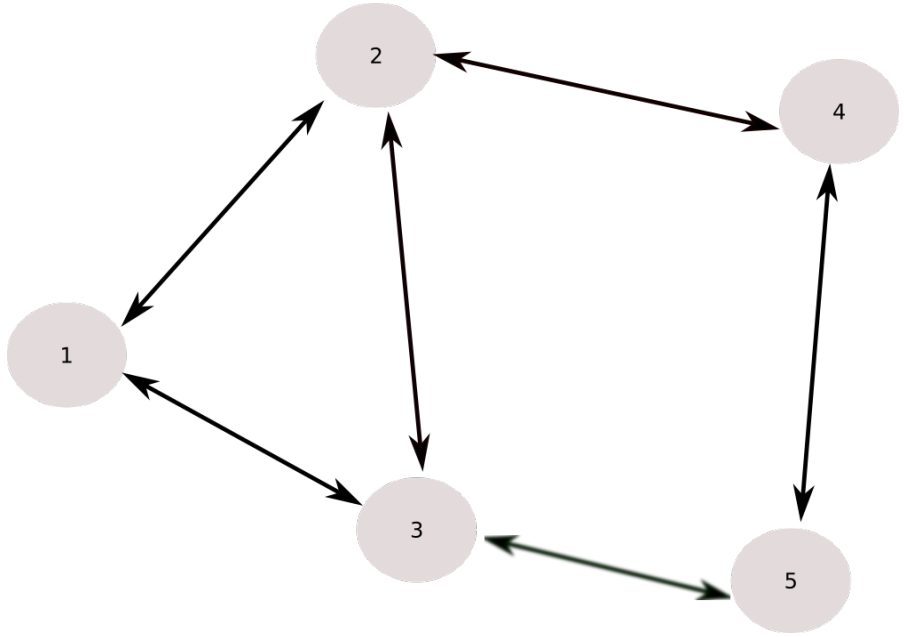
The **adjacency matrix of a graph** is the **sparse matrix A**, with nonzero entries between nodes that are connected with an edge, namely:

$$A_{ij} = \mathbf{w}_{ij} \text{ for all} (i,j) \in E$$

If the graph is symmetric, then also the adjacency matrix is symmetric, namely $A = A^T$

Degree Matrix of a Graph

Starting from **an unweighted, undirected graph,** the **degree of the graph** can be defined as

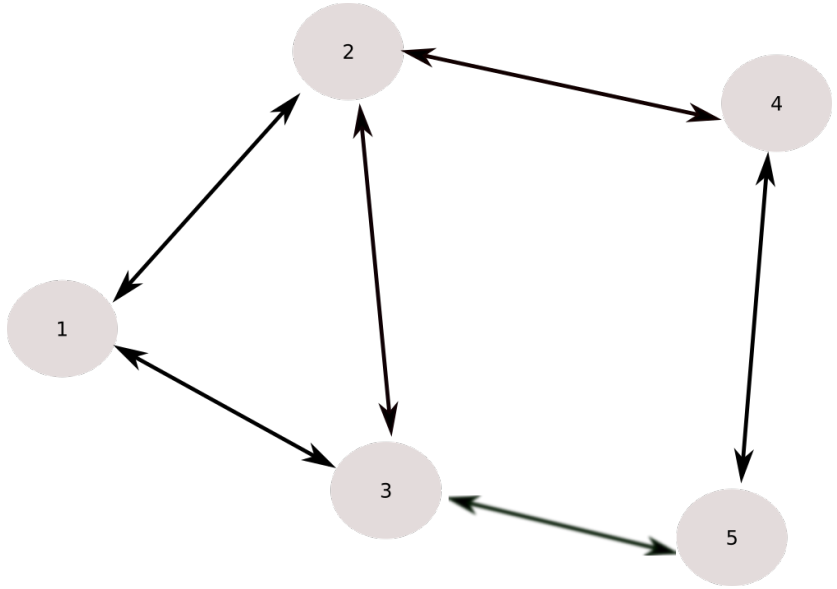


- The **neighborhood N(i) of a node i** is the set of nodes that are influenced by it, namely

$$N(i) = \{j \mid (i, j) \in E\}$$

Degree Matrix of a Graph

Starting from **an unweighted, undirected graph,** the **degree of the graph** can be defined as



- The **neighborhood N(i) of a node i** is the set of nodes that are influenced by it, namely

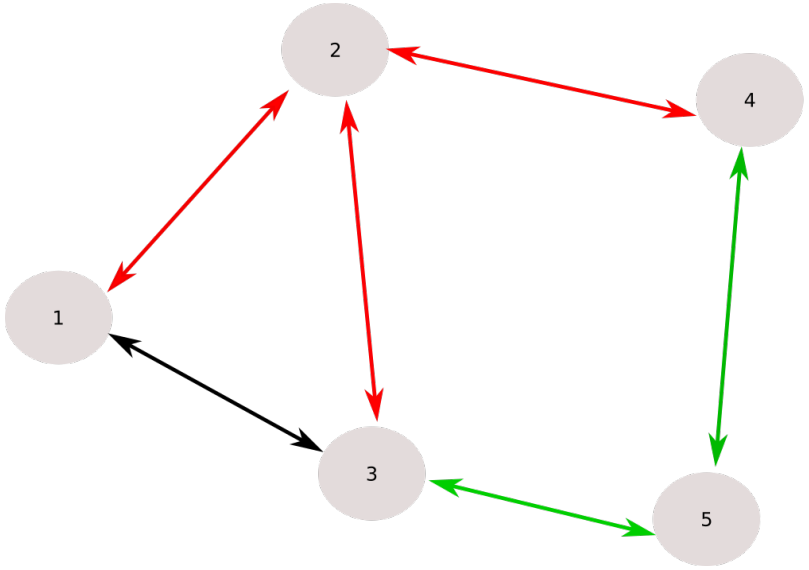
$$N(i) = \{j \mid (i, j) \in E\}$$

- The **degree of a node** is the sum of the weights of its incident edges, namely

$$d_i = \sum_{j \in N(i)} \mathbf{w}_{ij}$$

Degree Matrix of a Graph

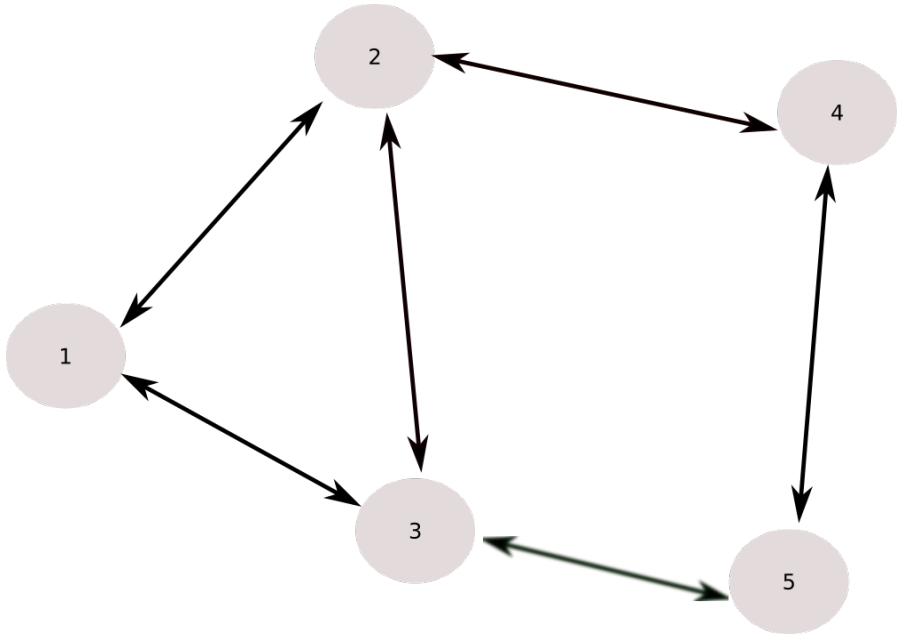
Starting from **an unweighted, undirected graph,** the **degree of the graph** can be defined as



$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

- The **degree matrix D** is a **diagonal matrix of degrees**, namely $D_{ii} = d_i$

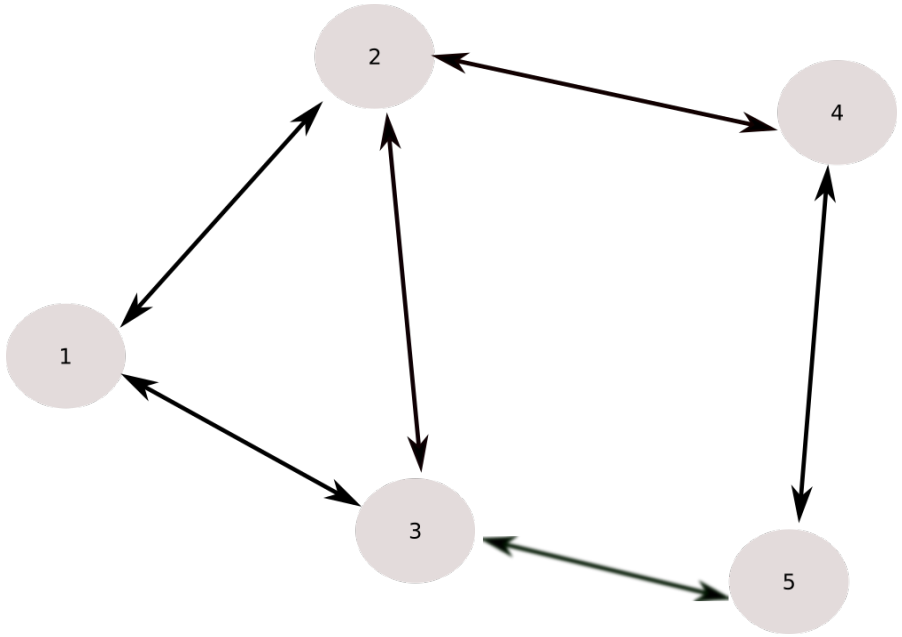
Laplacian Matrix



$$\mathbf{L} = \underbrace{\begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}}_{\text{Degree}} - \underbrace{\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}}_{\text{Adjacency}} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

- The **Laplacian matrix L** of a graph with adjacency matrix **A** is defined as L = D – A

Laplacian Matrix



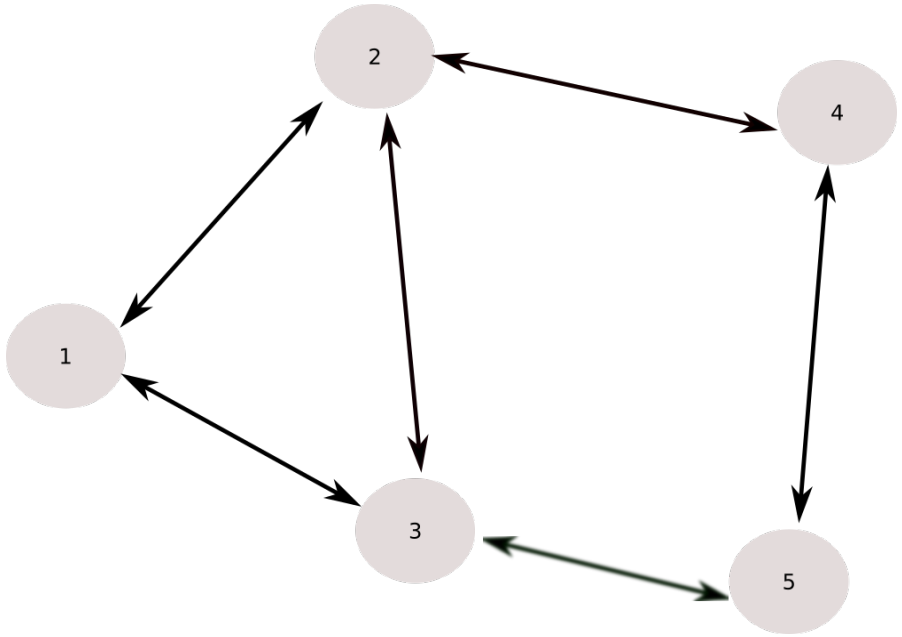
$$\mathbf{L} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

Adjacency

- The **Laplacian matrix L** of a graph with adjacency matrix **A** is defined as L = D – A
- The off diagonal entries of the Laplacian matrix are simply:

$$L_{ij} = -A_{ij} = -\mathbf{w}_{ij}$$

Laplacian Matrix



Degree

$$\mathbf{L} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

- The **Laplacian matrix L** of a graph with adjacency matrix **A** is defined as L = D – A
- The off diagonal entries of the Laplacian matrix are simply:

$$L_{ij} = -A_{ij} = -\mathbf{w}_{ij}$$

- The diagonal entries of the Laplacian matrix are simply $L_{ii} = di = \sum_{j \in N(i)} \mathbf{w}_{ij}$

Normalized Adjacency and Laplacian Matrix

- The **normalized adjacency matrix** expresses weights **relative to the degree** of the matrix

$$\bar{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad \text{,namely} \quad \bar{A}_{ij} = \frac{\mathbf{w}_i j}{\sqrt{(didj)}}$$

Normalized Adjacency and Laplacian Matrix

- The **normalized adjacency matrix** expresses weights **relative to the degree** of the matrix

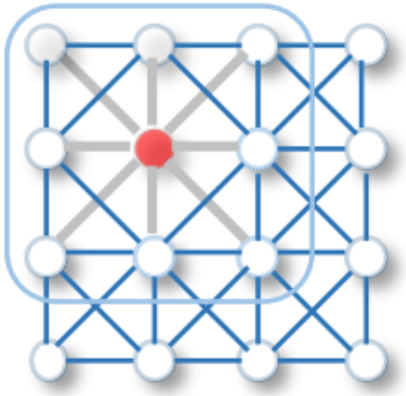
$$\bar{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad \text{,namely} \quad \bar{A}_{ij} = \frac{\mathbf{w}_i j}{\sqrt{(didj)}}$$

- The **normalized Laplacian matrix** is similarly defined as:

$$\bar{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = D^{-\frac{1}{2}} (D - A) D^{-\frac{1}{2}} = I - \bar{A}$$

Graph Convolutional Networks

The goal of the **Graph Convolutional Networks (GCNs) is to apply a convolution over a graph**. Instead of having a 2-D array as input, GCN takes a graph as an input.



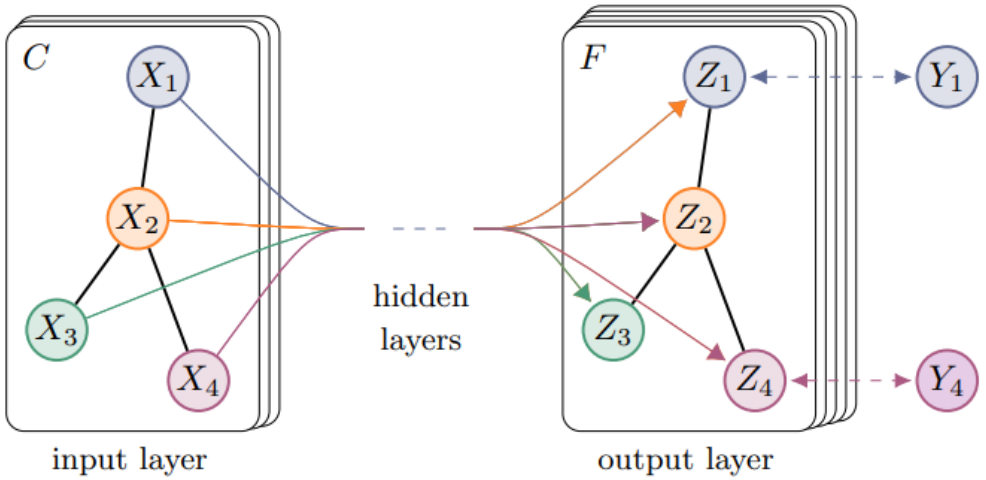
(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

A Comprehensive Survey on Graph Neural Networks, Zonghan Wu, Shirui Pan, Member, IEEE, Fengwen Chen, Guodong Long, Chengqi Zhang

Graph Convolutional Networks

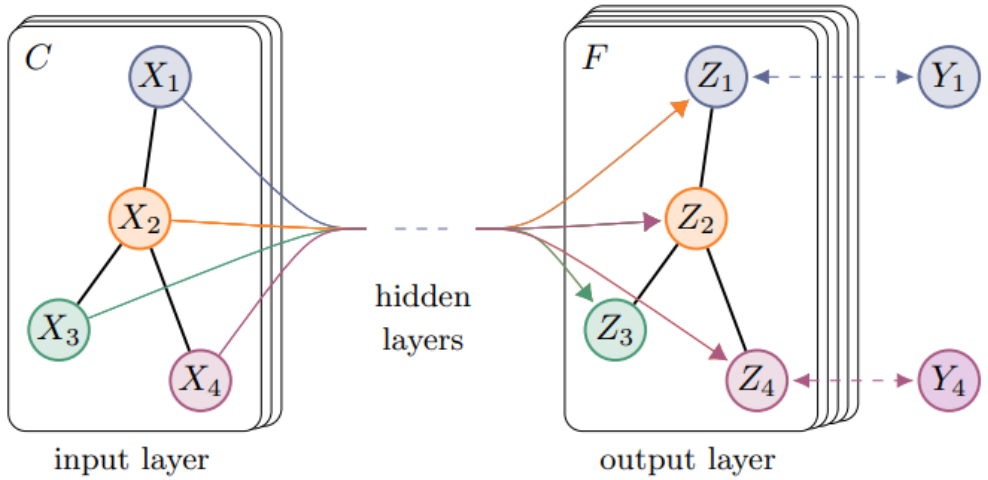
Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

Graph Convolutional Networks

Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



To this end we define a neural network layer as

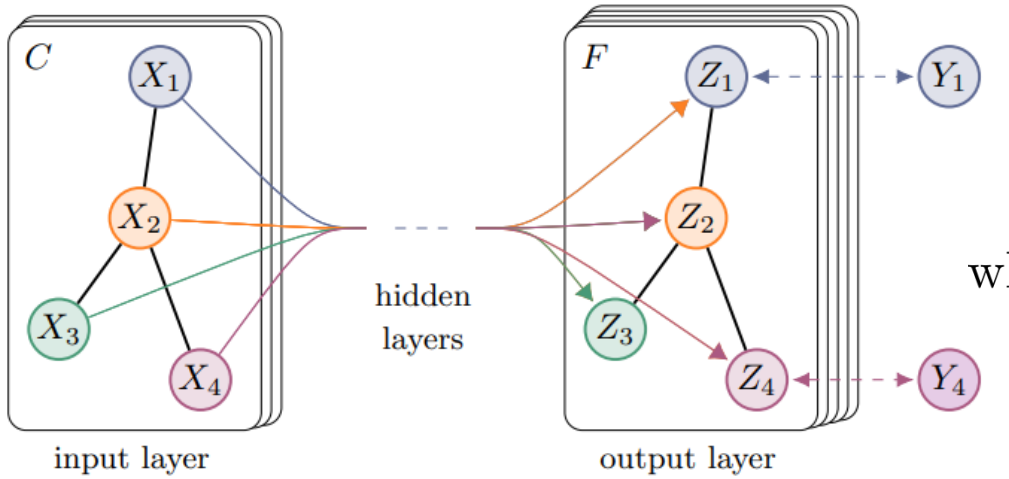
$$H^{(l+1)} = f(H^{(l)}, A)$$

where $H^0 = X$ and $H^{(l)} = Z$ and L is the number of layers

Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

Graph Convolutional Networks

Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



$$H^{(l+1)} = f(H^{(l)}, A)$$

where $H^0 = X$ and $H^{(l)} = Z$ and L is the number of layers

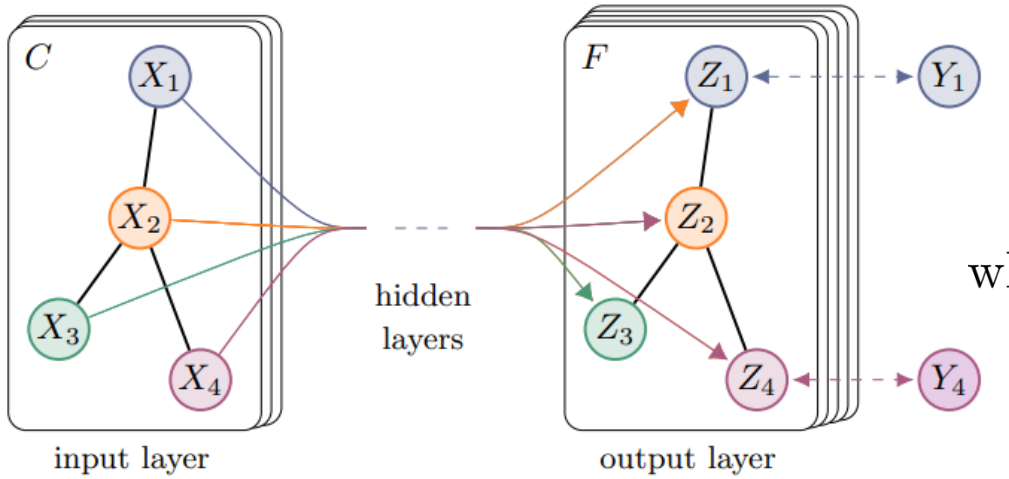
Let as assume the following very simple form of a layer

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

Graph Convolutional Networks

Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



$$H^{(l+1)} = f(H^{(l)}, A)$$

where $H^0 = X$ and $H^{(l)} = Z$ and L is the number of layers

Let as assume the following very simple form of a layer

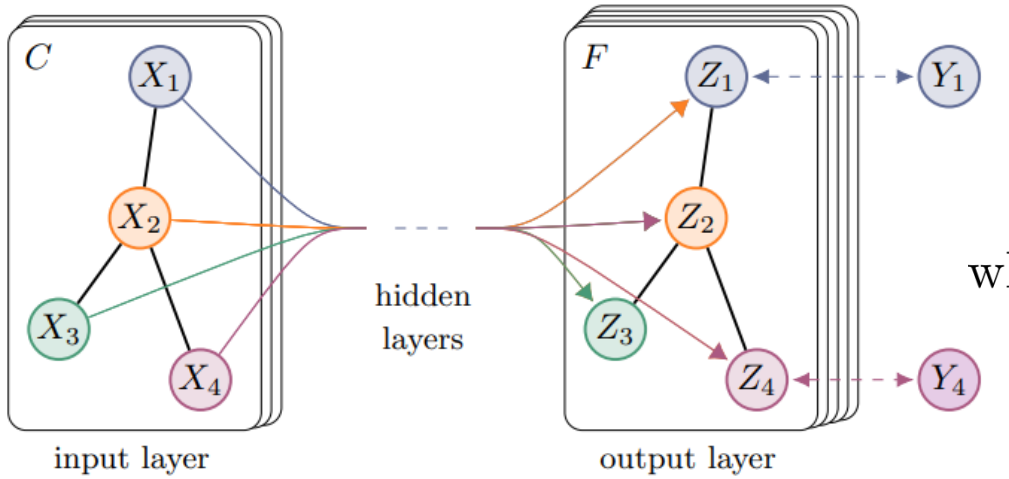
$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

Non linear activation function

Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

Graph Convolutional Networks

Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



$$H^{(l+1)} = f(H^{(l)}, A)$$

where $H^0 = X$ and $H^{(l)} = Z$ and L is the number of layers

Let as assume the following very simple form of a layer

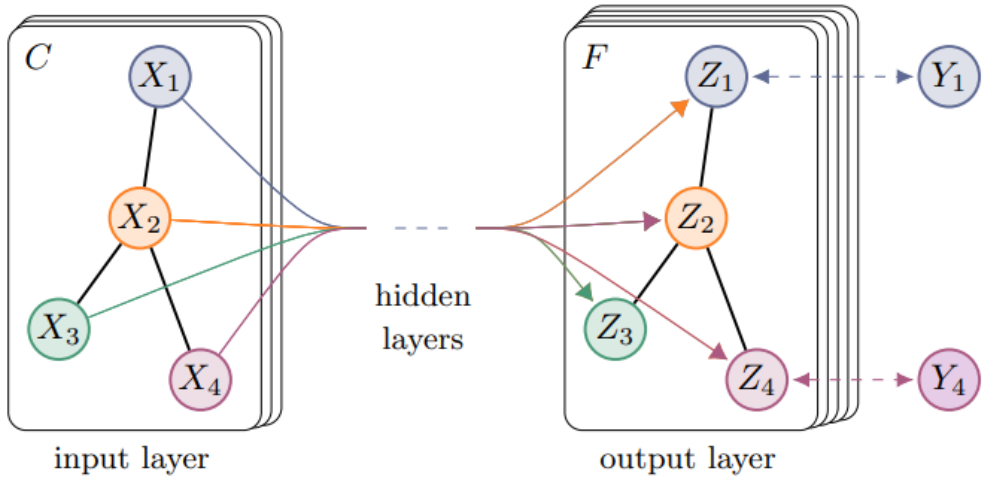
$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

Weight matrix

Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

Graph Convolutional Networks

Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



Let as assume the following very simple form of a layer

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

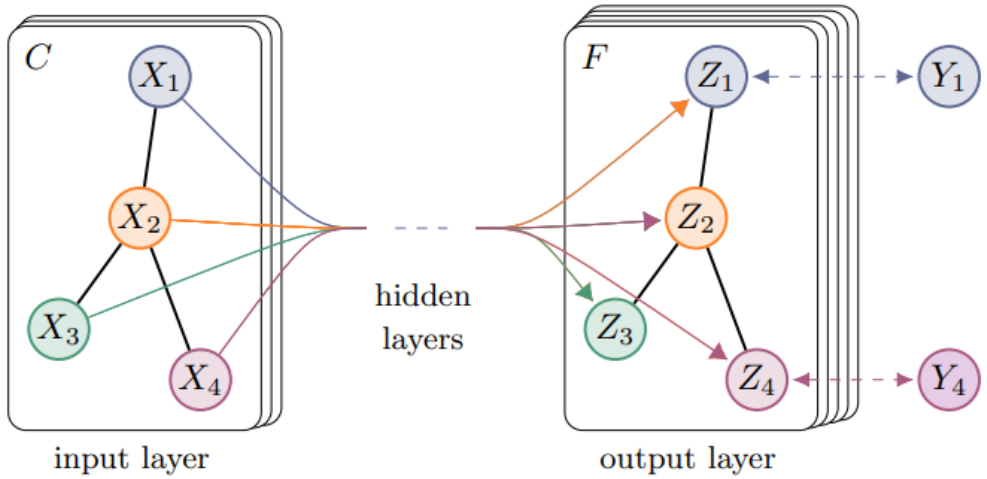
- **Multiplying with A, means to sum up all the feature vectors for all neighboring nodes** but not the node itself (unless there are self loops in the graph. **To enforce self-loops in the graph, we replace A with A+I**

$$f(H^{(l)}, A) = \sigma((A+I)H^{(l)}W^{(l)})$$

Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

Graph Convolutional Networks

Starting **from a graph G = (V, E, X)**, where **V are the N nodes**, **E are the edges** and **X is the NxD feature matrix** for very node in the graph **and a representation of its graph structure** in a matrix form (such as the adjacency matrix) we want to learn **a node-level representation Z**, defined as a NxF feature matrix, where F are the output features per node.



Let as assume the following very simple form of a layer

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

- **Multiplying with A, means to sum up all the feature vectors for all neighboring nodes** but not the node itself (unless there are self loops in the graph. **To enforce self-loops in the graph, we replace A with A+I**

$$f(H^{(l)}, A) = \sigma((A+I)H^{(l)}W^{(l)})$$

- Instead of directly using A+I, we normalize it

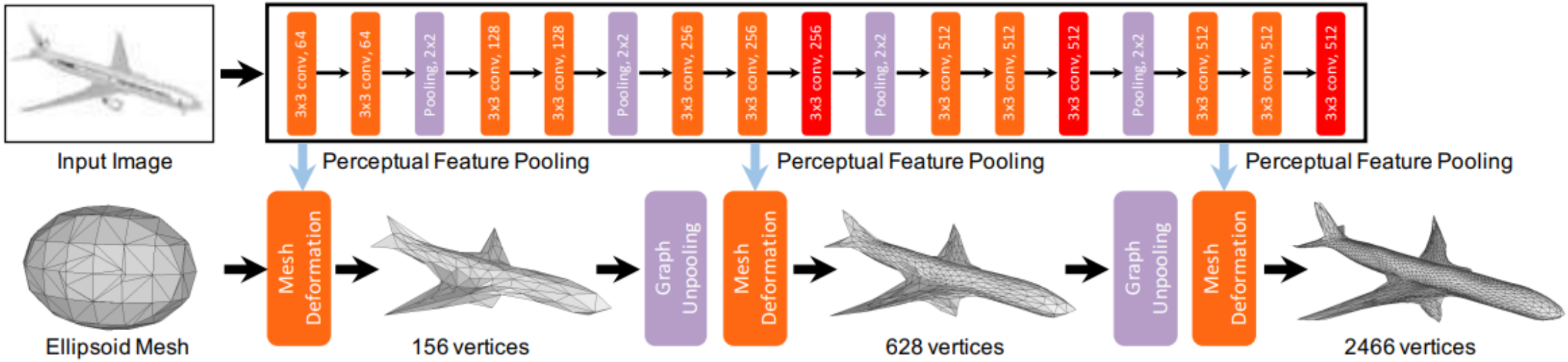
$$f(H^{(l)}, A) = \sigma(D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

where $\hat{A} = A + I$

Semi-supervised Classification with Graph Convolutional Networks, Thomas Kipf and Max Welling, ICLR 2017

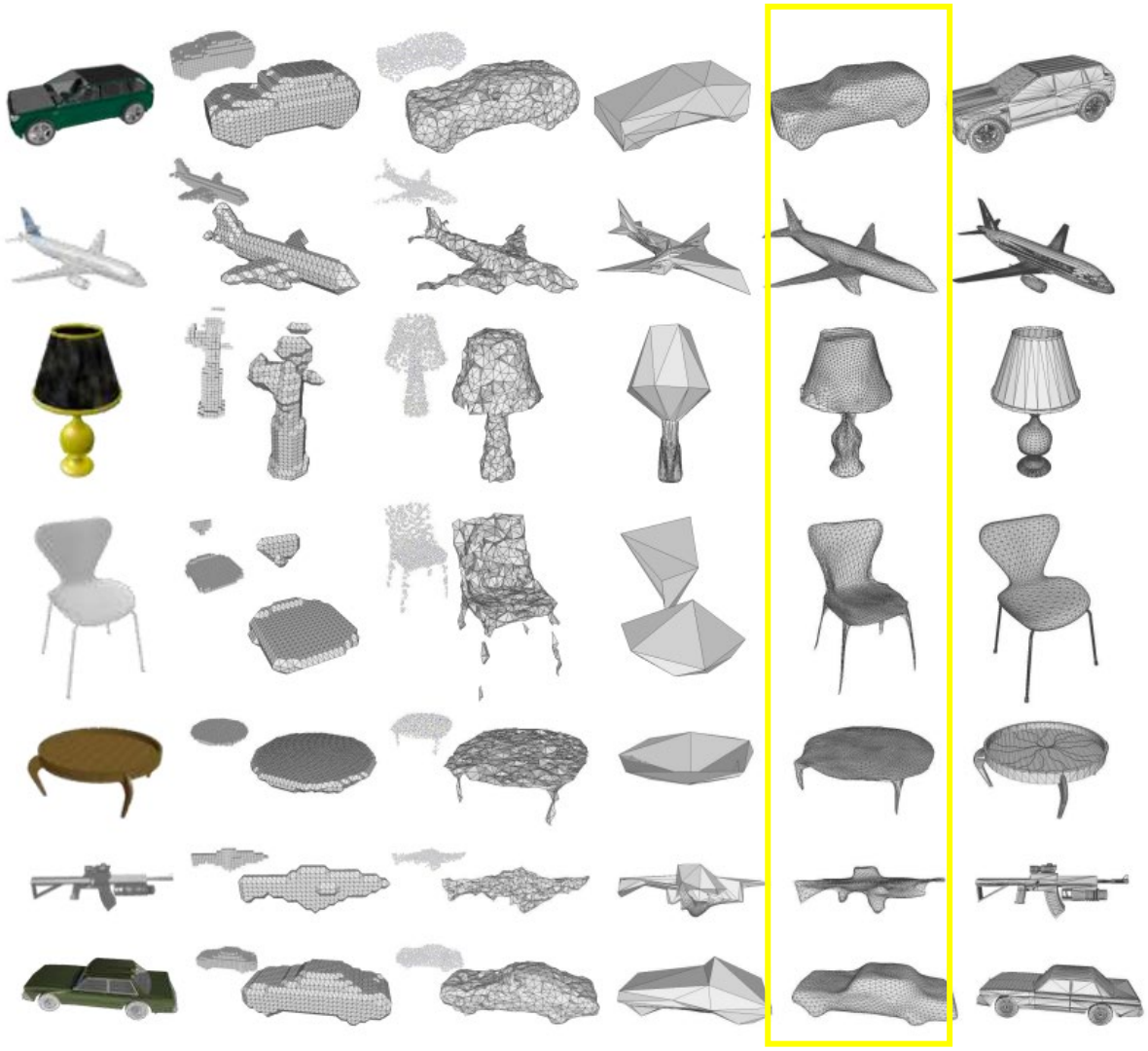
Graph Convolutional Networks for 3D

Goal: Train a deep neural network that would be able to reconstruct a single image as a 3D mesh. The key idea is that **a mesh can be defined as the deformation of an ellipsoid**. The **deformation is parametrized with a GCN**.



Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images, Nanyang Wang , Yinda Zhang2, Zhuwen Li3, Yanwei Fu4 , Wei Liu5 , Yu-Gang Jiang1, ECCV 2018

Graph Convolutional Networks for 3D



Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images, Nanyang Wang , Yinda Zhang2, Zhuwen Li3, Yanwei Fu4 , Wei Liu5 , Yu-Gang Jiang1, ECCV 2018

Convolution in Euclidean Space

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function

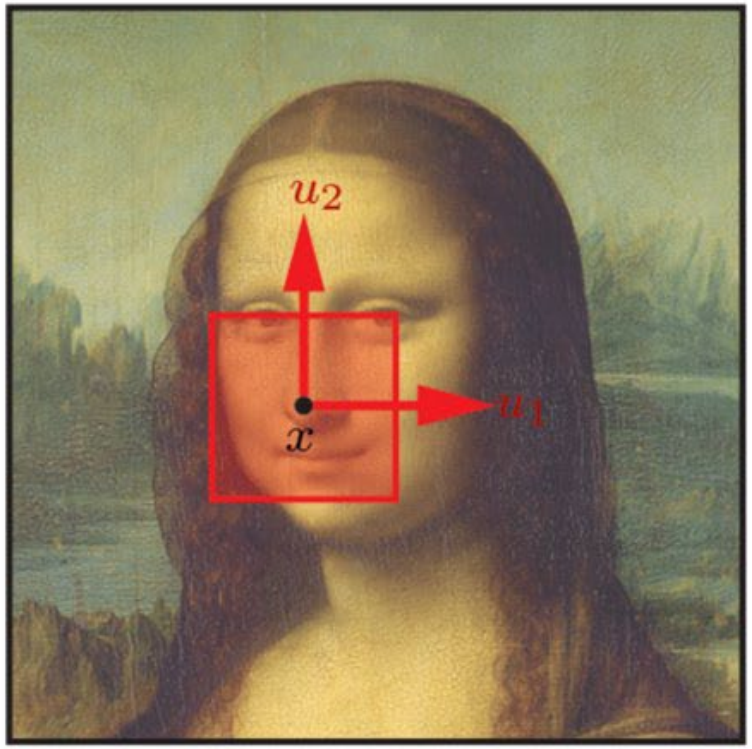
$$(f \star g)(x) = \int_{-\pi}^{\pi} f(x')g(x - x')dx'$$

- Shift-invariance: $f(x - x_0) \star g(x) = (f \star g)(x - x_0)$
- Convolution theorem: Fourier transform diagonalizes the convolution operator $\Rightarrow$ convolution can be computed in the Fourier domain as

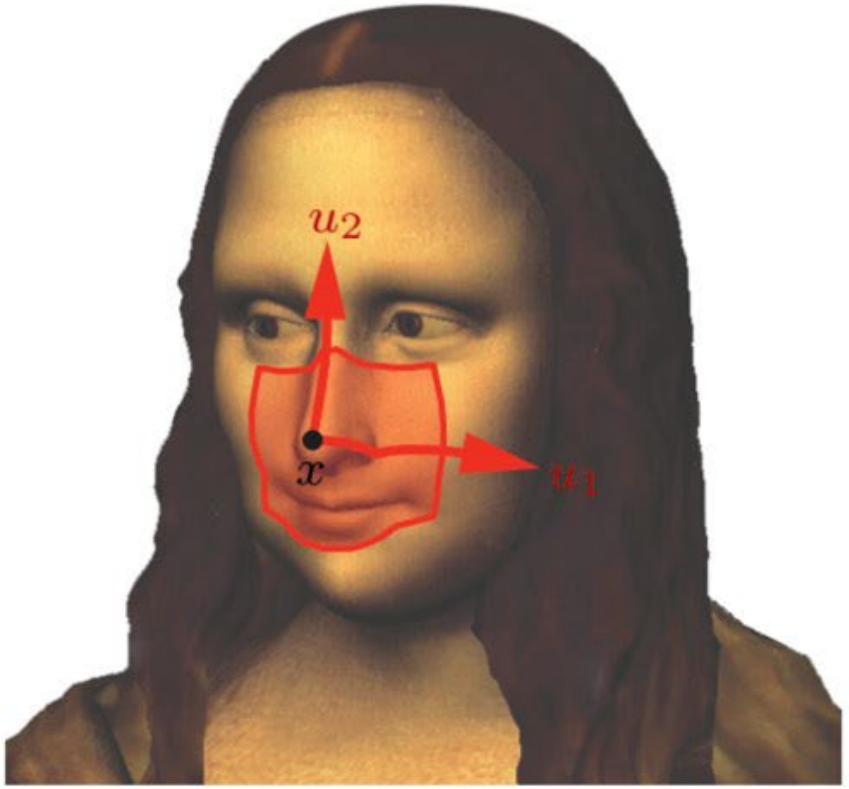
$$\widehat{(f \star g)} = \hat{f} \cdot \hat{g}$$

Convolution in non-Euclidean Space

To be able to use convolutions in non-Euclidean spaces, we need to extract patches on the manifold.



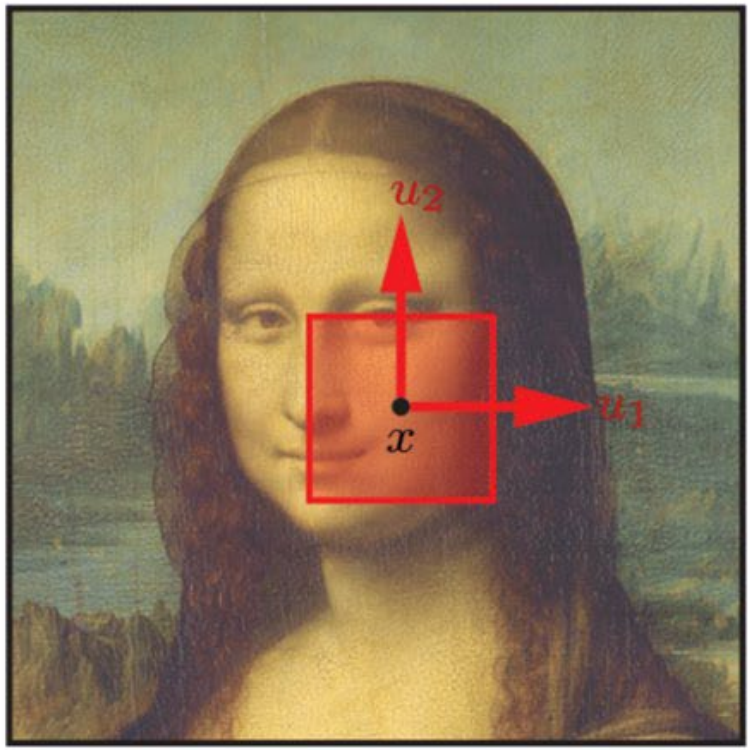
Image



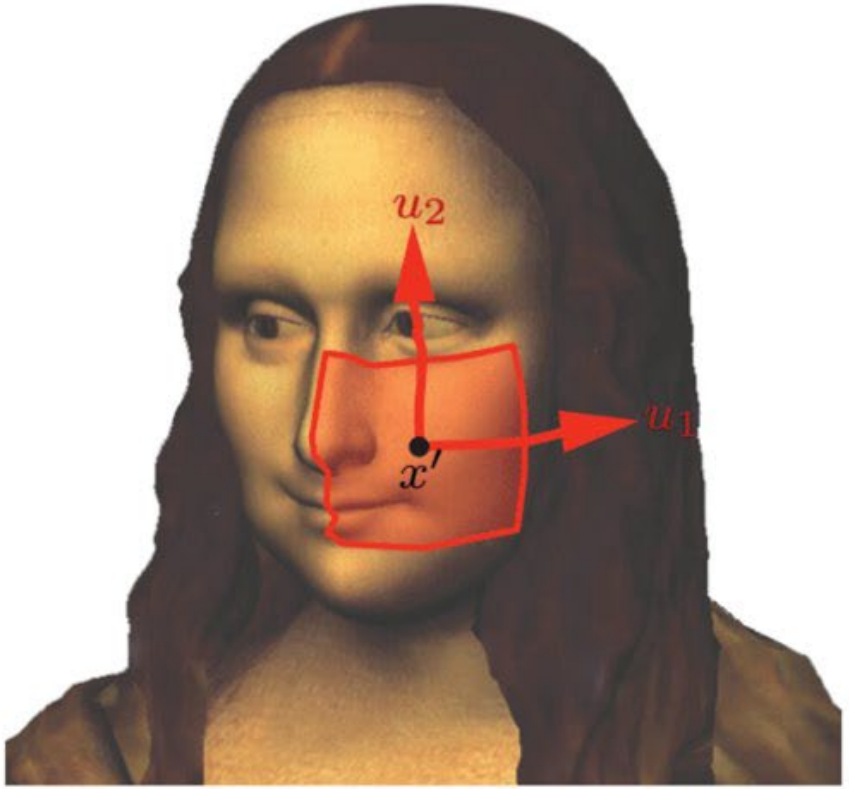
Manifold

Convolution in non-Euclidean Space

To be able to use convolutions in non-Euclidean spaces, we need to extract patches on the manifold.



Image



Manifold

Geodesic Convolutional Neural Networks

The key idea is to **employ a local system of geodesic polar coordinates** constructed at around a point x. **The radial coordinates** is constructed as p-level sets $\{x' : d_x(x, x') = p\}$ of the geodesic distance (shortest path) for every $p \in [0, p_0]$ where p_0 is the radius of the geodesic disc

Examples of Local Geodesic Patches

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Geodesic Convolutional Neural Networks

The key idea is to **employ a local system of geodesic polar coordinates** constructed at around a point x. **The radial coordinates** is constructed as p-level sets $\{x' : d_x(x, x') = p\}$ of the geodesic distance (shortest path) for every $p \in [0, p_0]$ where p_0 is the radius of the geodesic disc

Examples of Local Geodesic Patches

We can define the **bijective map from the manifold into the local geodesic polar coordinates (ρ,θ)** around point x as:

$$\Omega(x) : B_{p_0}(x) \rightarrow [0, p_0] \times [0, 2\pi]$$

Points on the manifold around x !!!

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Geodesic Convolutional Neural Networks

The key idea is to **employ a local system of geodesic polar coordinates** constructed at around a point x. **The radial coordinates** is constructed as p-level sets $\{x' : d_x(x, x') = p\}$ of the geodesic distance (shortest path) for every $p \in [0, p_0]$ where p_0 is the radius of the geodesic disc

Examples of Local Geodesic Patches

We can define the **bijective map from the manifold into the local geodesic polar coordinates (ρ,θ)** around point x as:

$$\Omega(x) : B_{p_0}(x) \rightarrow [0, p_0] \times [0, 2\pi]$$

We also define $(D(x)f)(p, \theta) = (f \circ \Omega^{-1}(x))(p, \theta)$

where D(x) is called the **patch operator** that is used for **mapping the values of the function** f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Patch Operator

The patch operator maps the values of the function f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

$$(D(x)f)(p,\theta) = \int_{\mathcal{X}} w_{p,\theta}(x,y) f(y) dy$$

Examples of Local Geodesic Patches

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Patch Operator

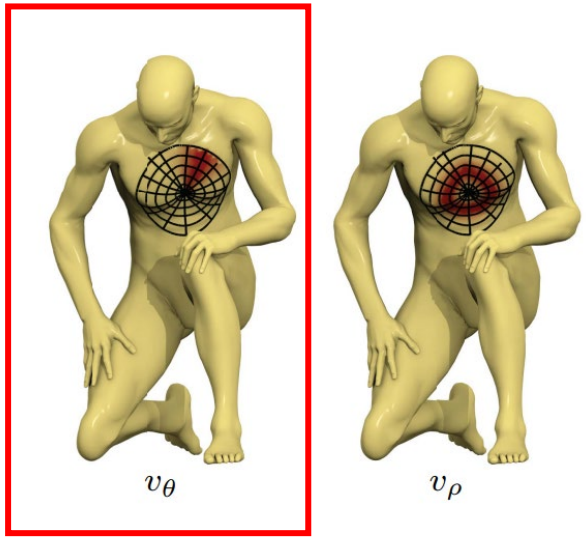
The patch operator maps the values of the function f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

$$(D(x)f)(p,\theta) = \int_{\mathcal{X}} w_{p,\theta}(x,y) f(y) dy$$

Examples of Local Geodesic Patches

$$w_{p,\theta} = \frac{v_p(x,x') v_\theta(x,x')}{\int_{\mathcal{X}} v_p(x,x') v_\theta(x,x') dx'}$$

Angular Weights



Both the Radial and the Angular weights are Gaussians.

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Patch Operator

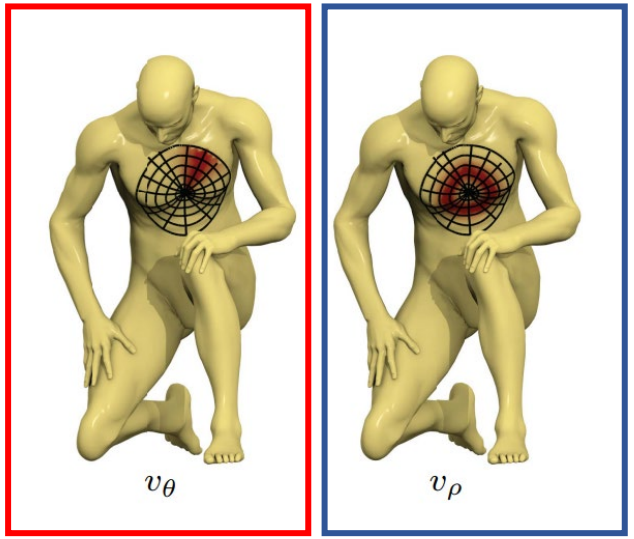
The patch operator maps the values of the function f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

$$(D(x)f)(p,\theta) = \int_{\mathcal{X}} w_{p,\theta}(x,y) f(y) dy$$

Examples of Local Geodesic Patches

$$w_{p,\theta} = \frac{v_p(x,x')\, v_\theta(x,x')}{\int_{\mathcal{X}} v_p(x,x')\, v_\theta(x,x')\, dx'}$$

Angular Weights



Radial Weights

Both the Radial and the Angular weights are Gaussians.

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Patch Operator

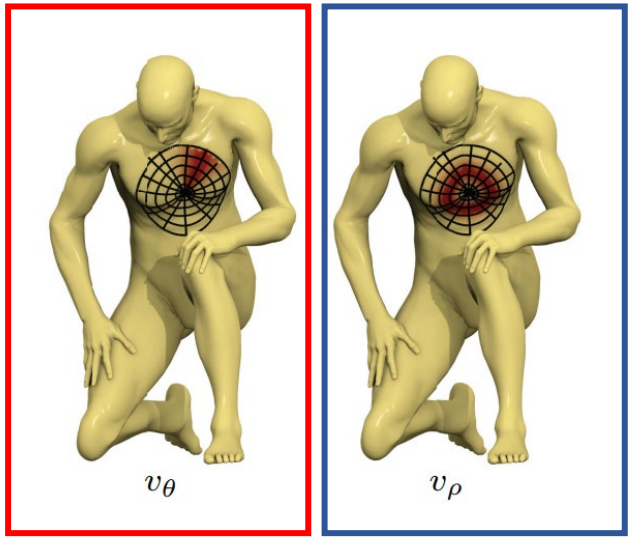
The patch operator maps the values of the function f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

$$(D(x)f)(p,\theta) = \int_{\mathcal{X}} w_{p,\theta}(x,y) f(y) dy$$

Examples of Local Geodesic Patches

$$w_{p,\theta} = \frac{v_p(x,x')\, v_\theta(x,x')}{\int_{\mathcal{X}} v_p(x,x')\, v_\theta(x,x') dx'}$$

Angular Weights



Radial Weights

Both the Radial and the Angular weights are Gaussians.

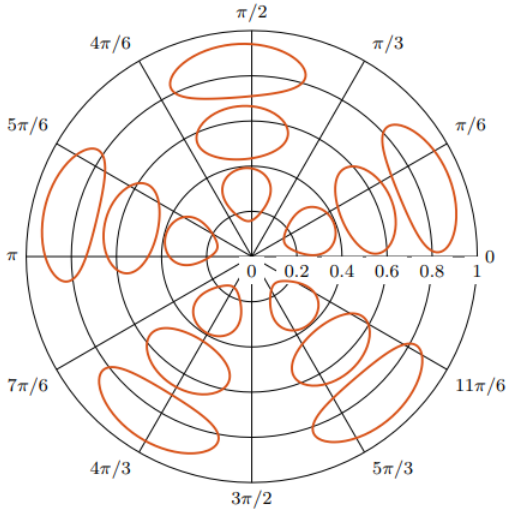
Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Patch Operator

The patch operator maps the values of the function f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

$$(D(x)f)(p,\theta) = \int_{\mathcal{X}} w_{p,\theta}(x,y) f(y) dy$$

$$w_{p,\theta} = \frac{v_p(x,x')v_\theta(x,x')}{\int_{\mathcal{X}} v_p(x,x')v_\theta(x,x')dx'}$$



Examples of Local Geodesic Patches

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Geodesic Convolutional Neural Networks

The patch operator maps the values of the function f at a neighborhood of the point x into the local polar coordinates (ρ,θ).

$$(D(x)f)(p,\theta) = \int_{\mathcal{X}} w_{p,\theta}(x,y)f(y)dy$$

The **Geodesic Convolution** can be thought of as matching a template g(ρ, θ) with the extracted patch at each point

$$(f\star g)(x) = \max_{\Delta\theta\in[0,2\pi)} \int_0^{2\pi}\int_0^{\rho_{\max}} g(\rho,\theta+\Delta\theta)(D(x)f)(\rho,\theta)d\rho d\theta,$$

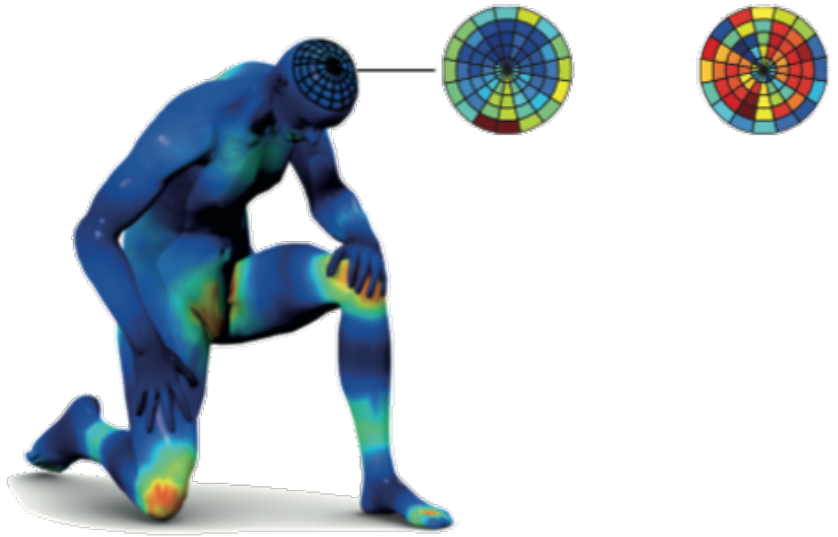
where **the maximum is taken over all possible rotations** of the template in order to **resolve the origin ambiguity in the angular coordinate.**

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Geodesic Convolutional Neural Networks

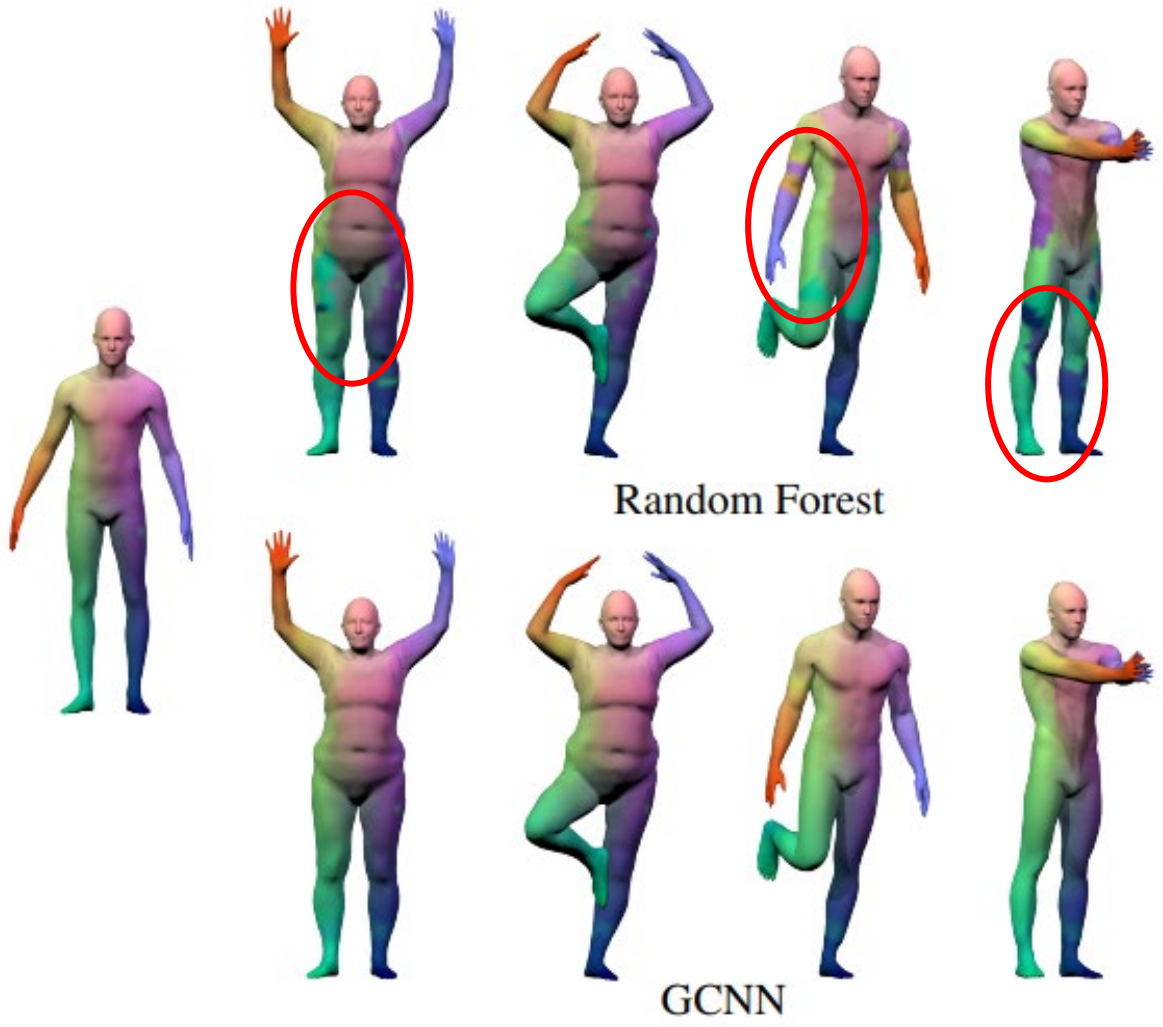
- Geodesic convolution = apply filter a to patches extracted from $f \in L^2(X)$ in local geodesic polar coordinates

$$(f \star a)(x) = \sum_{\theta,r} \underbrace{(D(x)f)(r,\theta)}\ \underbrace{a(\theta,r)}$$



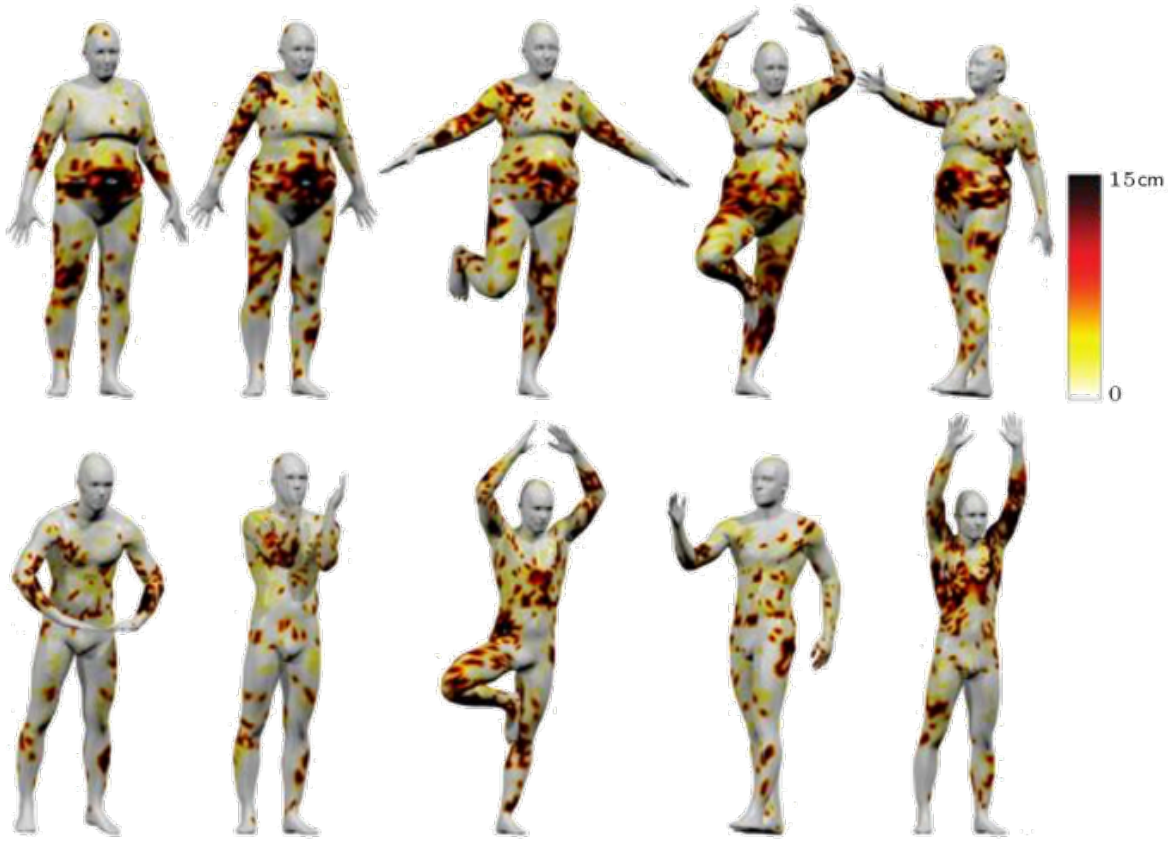
Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

GCNN for Learning: Shape Correspondences



Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

GCNN for Learning: Shape Matching



Pointwise correspondence error (geodesic distance from groundtruth)

Geodesic convolutional neural networks on Riemannian manifolds, Jonathan Masci, Davide Boscaini, Michael M. Bronstein, Pierre Vandergheynst, ICCV 2015

Conclusions

- Geometric Deep Learning is widely used for various tasks such as:
  - 3D Reconstruction
  - Completion
  - Scene Understanding
  - Shape completion
  - Shape parsing
  - …

- Using graph convolutional neural networks we can easily implement a network that can operate on graphs using two simple tricks:
  - We make sure to account for self loops
  - Instead of using the adjacency matrix, we use the normalized adjacency matrix
  - Additional Material on Graph Neural Networks: **https://gnn.seas.upenn.edu/lectures/lecture-1/**

That’s All