Advanced Deep Learning Models and Methods for 3D Spatial Data

Giacomo Boracchi, Luca Magri, Simone Melzi, Matteo Matteucci

DEIB, Politecnico di Milano

January 18th, 2024

giacomo.boracchi@polimi.it

https://boracchi.faculty.polimi.it/

Giacomo Boracchi

Mathematician (Università Statale degli Studi di Milano 2004),
PhD in Information Technology (DEIB, Politecnico di Milano 2008)
Associate Professor since 2019 at DEIB (Computer Science), Polimi

Research Interests are mathematical and statistical methods for:

- Image / Signal analysis and processing
- Unsupervised learning, change / anomaly detection

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

Who I am

Mathematician (Università Statale degli Studi di Milano 2012),

PhD in Applied Mathematics (Università Statale degli Studi di Milano 2015)

Researcher (RTD-B) since 2021 at DEIB, Polimi (Computer Science)

My research interests are in:

- Geometric Computer Vision with a special emphasis on 3D reconstruction
- Pattern Recognition and Model Fitting

Teaching: Geometric Computer Vision (PhD, UniTrento, La Sapienza)

luca.magri@polimi.it

https://magrilu.github.io/

Simone Melzi

Mathematician (Università Statale Degli Studi di Milano - 2013)

PhD in Computer Science (Università Degli Studi di Verona - 2018)

Assistant Professor since 2022 at the University of Milano – Bicocca Computer Science

Research interests are geometry processing and machine learning on geometric data

- Non-rigid shape matching and registrations
- Analysis and comaprison of collections of shapes
- Medical imagining and statistical analysis

simone.melzi@unimib.it
https://sites.google.com/site/melzismn

Matteo Matteucci

Matteo Matteucci, PhD

Full Professor

Dept. of Electronics, Information & Bioengineering

Politecnico di Milano

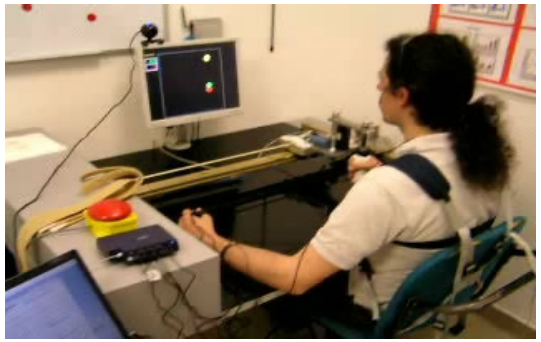
matteo.matteucci@polimi.it

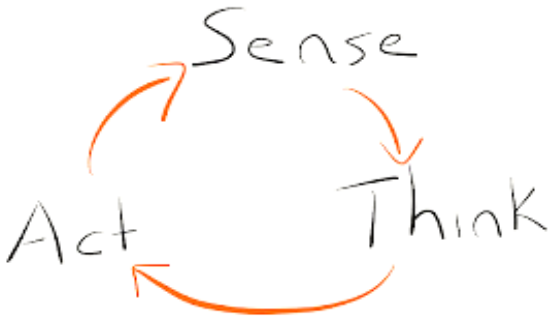
Research interests

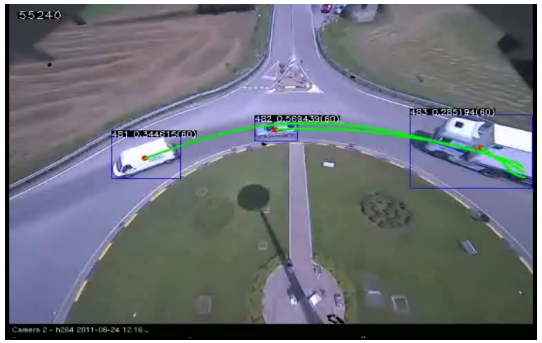
- Robotics & Autonomous Systems
- Machine Learning
- Pattern Recognition
- Computer Vision & Perception

Courses

- Robotics (BS+MS)
- Machine Learning (MS)
- Deep Learning (MS+PhD)
- Uncertainty in AI (MS)







About this Course

Yet another eposode of the ADLMM saga..

Mission and Goal

[…] This course presents recent advances in deep learning that brought **data-driven models** to achieve state-of-the-art performance in **solving 3D vision problems**.

Students will become acquainted with the biggest **challenges of handling 3D data** that are scattered in nature, thus are not suited for traditional filtering operations underpinning convolutional layers.

The course will illustrate the most important layers for handling 3D data, as well as the **neural networks for solving 3D Computer Vision** problems and their **application to Robotics and Computational Geometry**.

This is intended as an advanced course, thus proficiency in neural networks, convolutional neural networks and basic notions of optimization are assumed as pre-requirement to the participants.

The Course Outline

- 17/01/2024, *Deep Learning for Volumetric Data and 3D Point Clouds*, Giacomo Boracchi
- 24/01/2024, *Deep Learning for Depth Estimation*, Luca Magri
- 31/01/2024, *Deep Learning in 3D Non-rigid Shape Registration*, Simone Melzi (Univ. Bicocca)
- **09/02/2024 , *Deep Learning in 3D for Robotics*, Matteo Matteucci**
- 4/02/2024, *Hackaton Session*

This lecture has changed!

Course Logistics

- 17/01/2024 from 14:00 to 18:00 in Sala Conferenze Emilio Gatti Ed. 20
- 24/01/2024 from 14:00 to 18:00 in Sala Conferenze Emilio Gatti Ed. 20
- 31/01/2024 from 14:00 to 18:00 in Sala Conferenze Emilio Gatti Ed. 20
- **09/02/2024 from 14:00 to 18:00 in Room 3.1.7**
- 14/02/2024 from 14:00 to 19:0 in Room 3.1.7

Teaching Organization

Traditional lectures for presenting theory and the most important models will be structured as follows:

- Providing a clear formulation of the addressed problems …
- Provide a unified description of the considered model.
- Provide an overview of applications of the considered model.
- Overview research directions and papers presenting the most important achievements in the literature.

Where possible we will give guided **computer-laboratory sessions**

Hands-on session (the last day), to develop an application in groups

All the lectures will be held in presence in the Leonardo Campus

Course Details for PhD Students:

- Minimum 70% attendance required. In practice you can skip a single lecture
- You have to sign papers we will circulate during the break
- Assessment is pass-or-fail.
- The exam consists in successfully taking part to the hackathon
- During the hackathon you can gather in groups of 2 students and proactively interact with the instructors to demonstrate you are familiar in using these models.

Course Details for MSc Students:

- No attendance requirements, but please fill the form
- Exam grading is 18 – 30L as usual
- Take part to the hackathon, become very familiar with all the materials
- After the hackathon, you need to agree with us a follow up project
- You can gather in groups of 2-3 students

Project for MSc

The project:

You are requested to implement the solution presented in the paper and or extend it (if there is a public implementation available). Apply the solution to a different domain you prefer.

The exam:

- You need to write a paper-like resume presenting your solution / application. Templates will be provided
- There is no exam schedule for PhD courses. We will organize a few presentation days where everybody is welcome to attend.
- The exam will be a discussion on your work and on the most important contents seen during lectures.

Dates: July, September, December

You might want to check previous editions..

2023 Advanced Deep Learning Models And Methods: The Rise of Transformers

2022 Advanced Deep Learning Models And Methods (multiple topics)

2020 Machine Learning For Non-Matrix Data

2019 Advances In Deep Learning With Applications In Text And Image Processing

2018 Image Classification: Modern Approaches

Or some missing item in AN2DL

Credits / Good Resources

This course is inspired by many good sources, some materials are from these.

Hao Su 3D Deep Learning Tutorial CVPR 17 Tutorial

Lecture 17 on 3D Vision from Justin Johnson, Michigan University

"Geometry processing and machine learning for geometric data" PhD Course given in Bicocca by Prof. Melzi and Marin
https://elearning.unimib.it/course/info.php?id=53007

Deep Learning for 3D Point Clouds and Volumetric Data

The Lecture Outline

Outline

Part1: 3D data for Deep Learning

- 3D Data Representations
- 3D Imaging Sensors
- Challenges and Datasets

Part2: Deep Learning on Voxelized Data

- 3D CNN for Classification
- 3D CNN for Segmentation
- Challenges and Advanced Models

Part3: Deep Learning on Point Clouds

- PointNet and Variants
- Point Convolutional Operators

Part4: Demo hands on

Part1: 3D data for Deep Learning

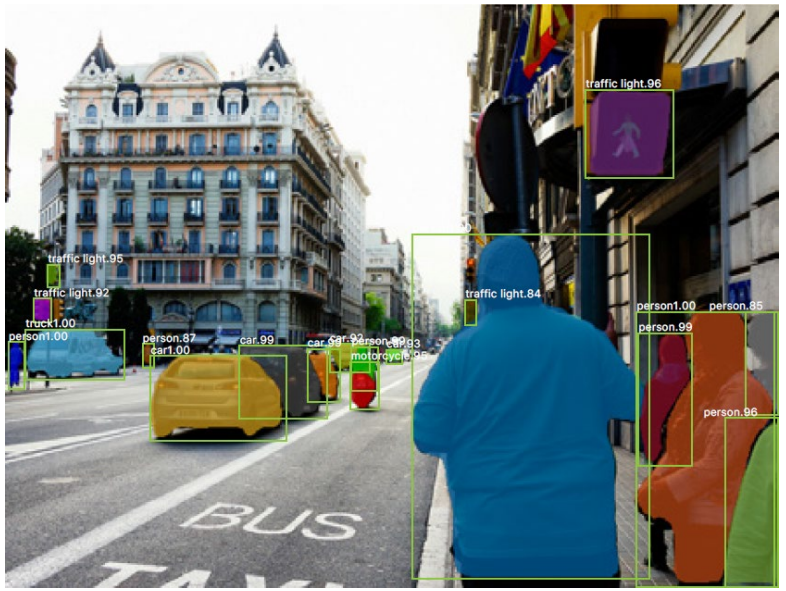
3D Data and Their Representation

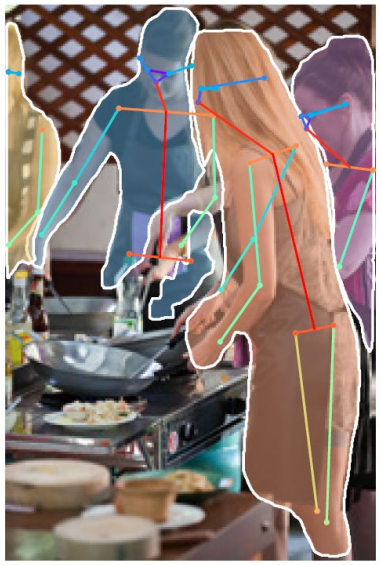
Visual Recognition & Deep Learning

We are used to solving visual recognition problems on images / videos

- Image Classification
- Object Detection
- Instance Segmentation
- Image Generation

You know DL achieves super-human performance in these tasks **when handling 2D images**





Intrinsic Ambiguity od 2D Projections

The 3D world is though more complicated than 2D images.

Unfortunately, there is an intrinsic ambiguity of $3D \rightarrow 2D$ mapping: some information is simply lost.

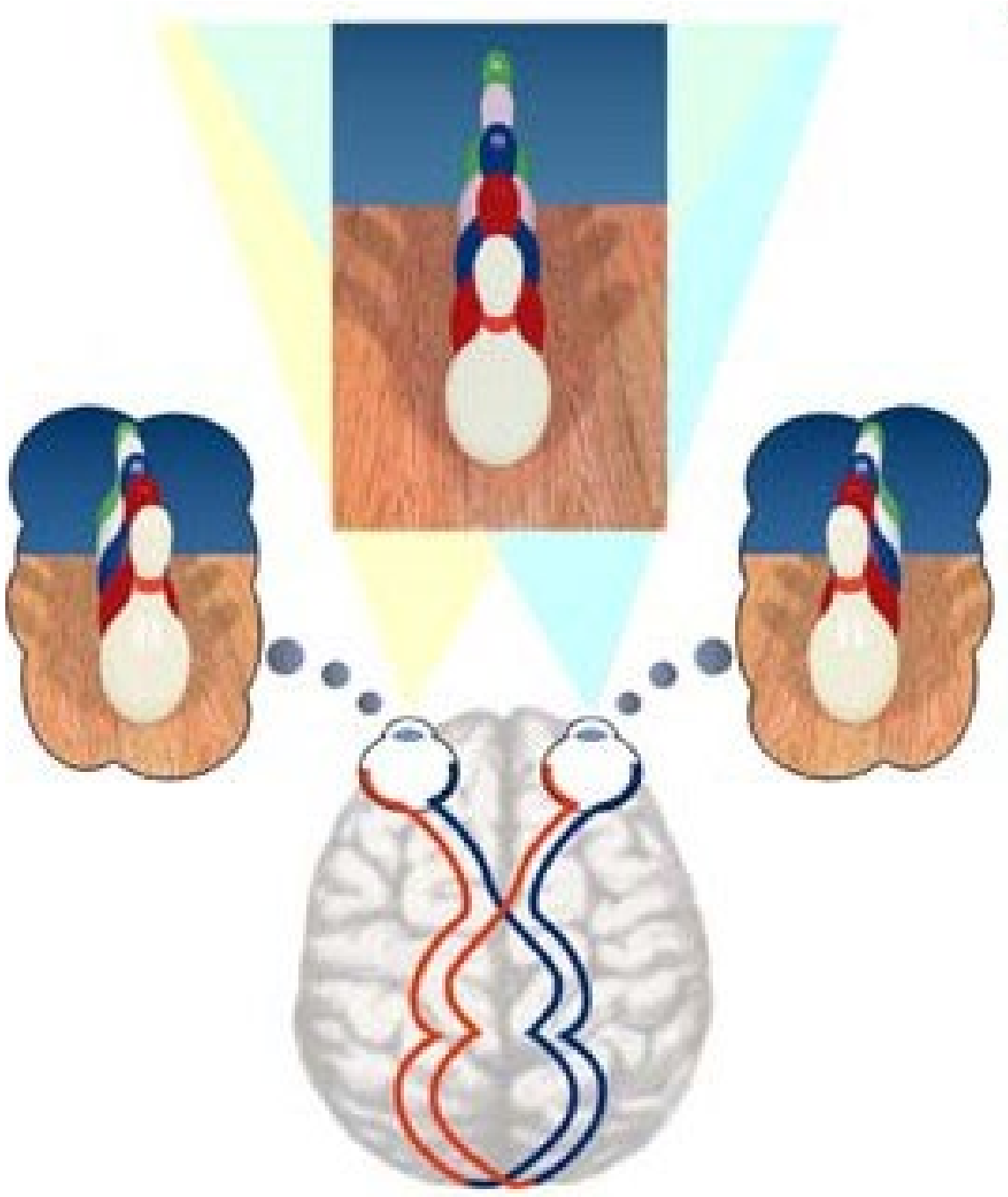
After projection, different depths cannot be distinguished in the image plane.

Courtesy slide S. Lazebnik

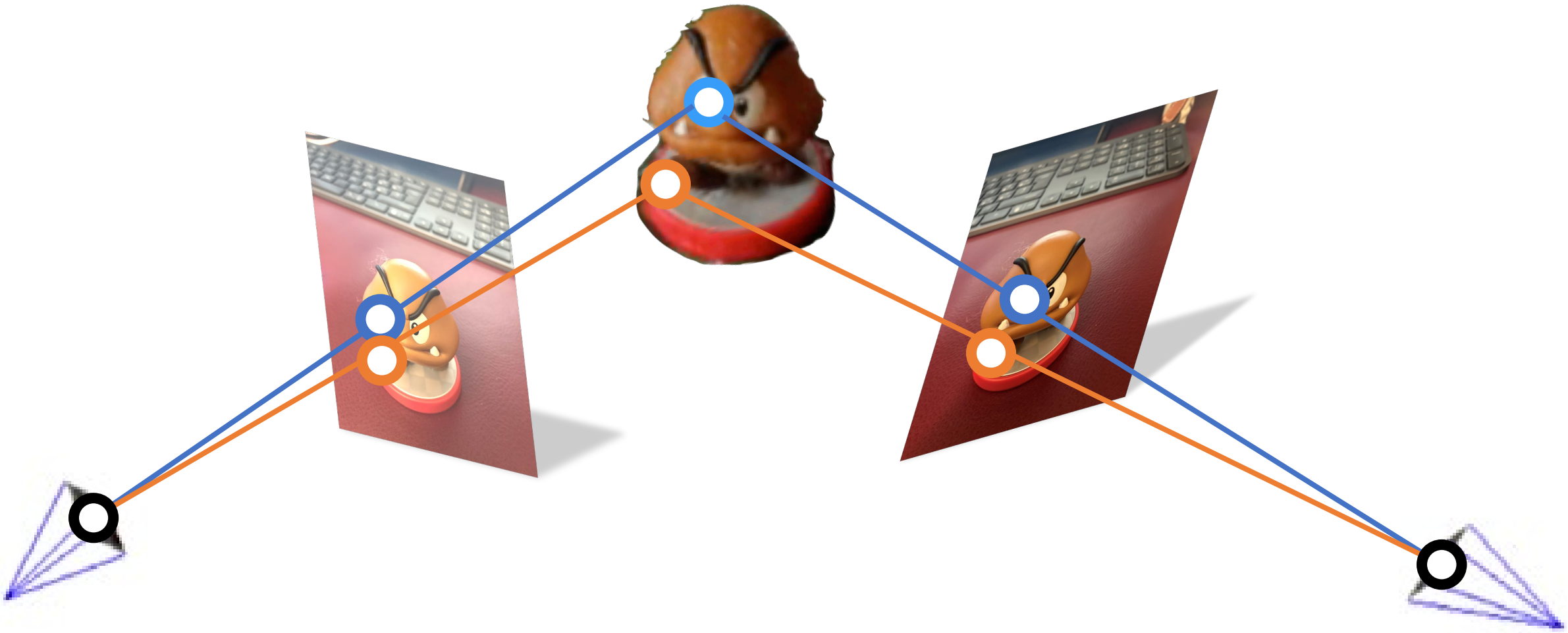
Multiple Views

Multiple views of the same scene help us resolve these potential ambiguities and perceive depths.

In fact, the same scene from a different viewpoint wouldn’t give rise to the same ambiguity.

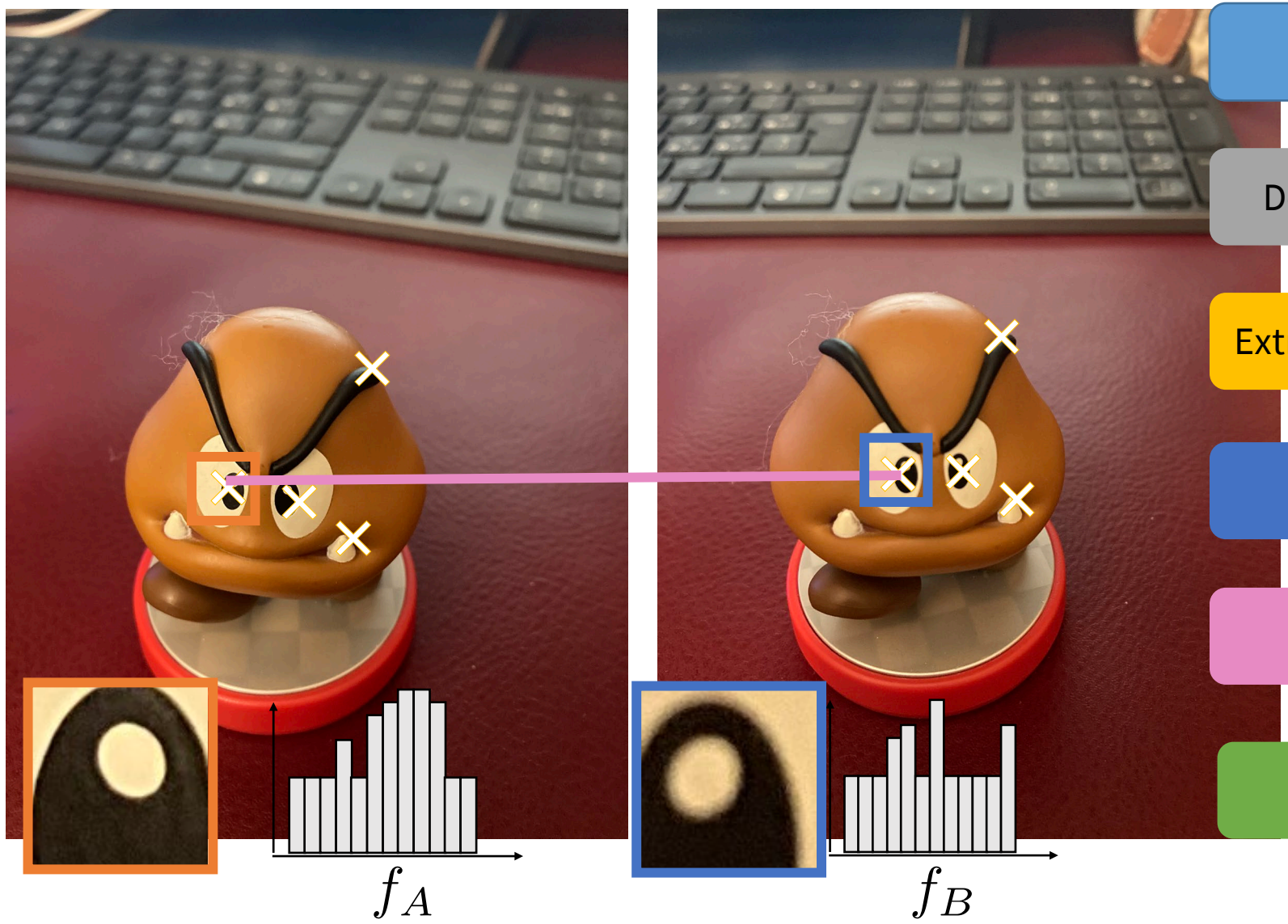


Multi View Geometry



Stereo Vision Systems leverages Calibrated cameras and point correspondences enables 3D reconstruction

Establishing Correspondences is Key!



Find a set of distinctive key points

Define a region around each keypoint

Extract and normalize the region content

Compute a local descriptor for the normalized region

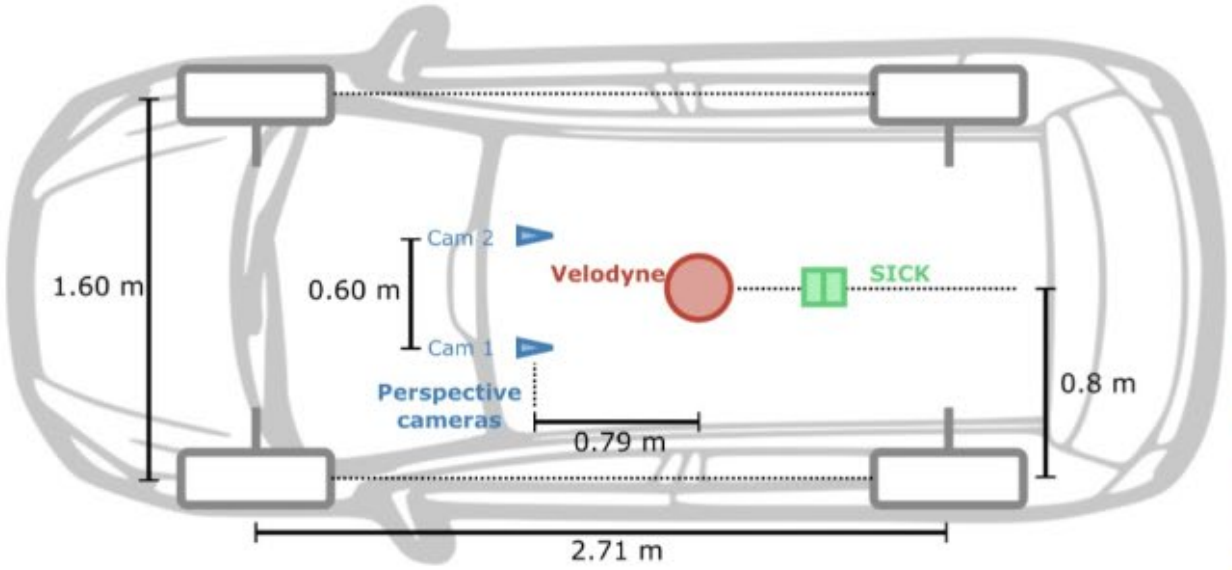
Match local descriptor

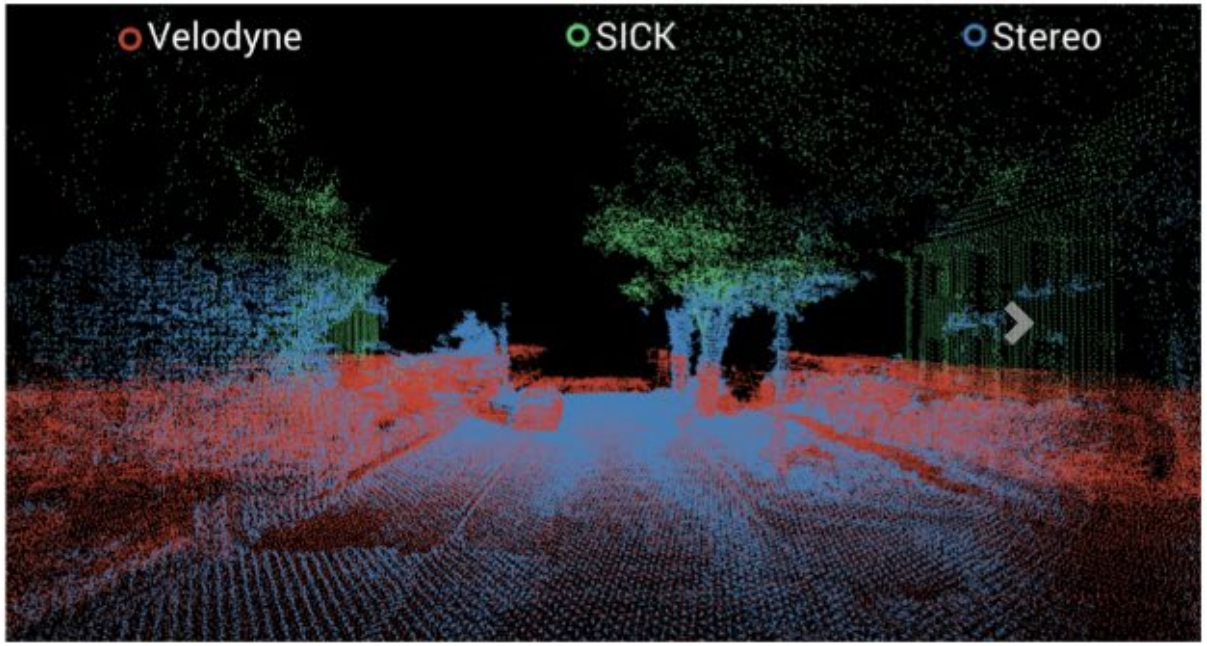
Robust model fitting

3D Data in Applications

Many application domains typically handle 3D data to interpret the real world:

- Medical Imaging (CT scans, MRI)
- Autonomous Driving and UaV (Lidar)
- Augmented Reality / Gaming (this is natively 3D)
- Robotics (require 3D interaction)





https://www.cvlibs.net/datasets/kitti-360/index.php

3D Data Representations

In contrast with images, which always come in an array form, 3D data are encoded in different ways

- Point Clouds
- Depth Maps
- Voxelized Grid
- Meshes
- Implicit Definitions

Different representation feature differenent properties and require **different models**

Point Clouds

This is the canonical form of raw 3D data.

These can be seen as set of points in $\mathbb{R}^d$.

Definition:

$$P \subset \mathbb{R}^d$$

When $d = 3$

- just saving the 3D coorinates of points

When $d > 3$

- When additional information is associated to the point coordinates

These are very popular in autonomous driving

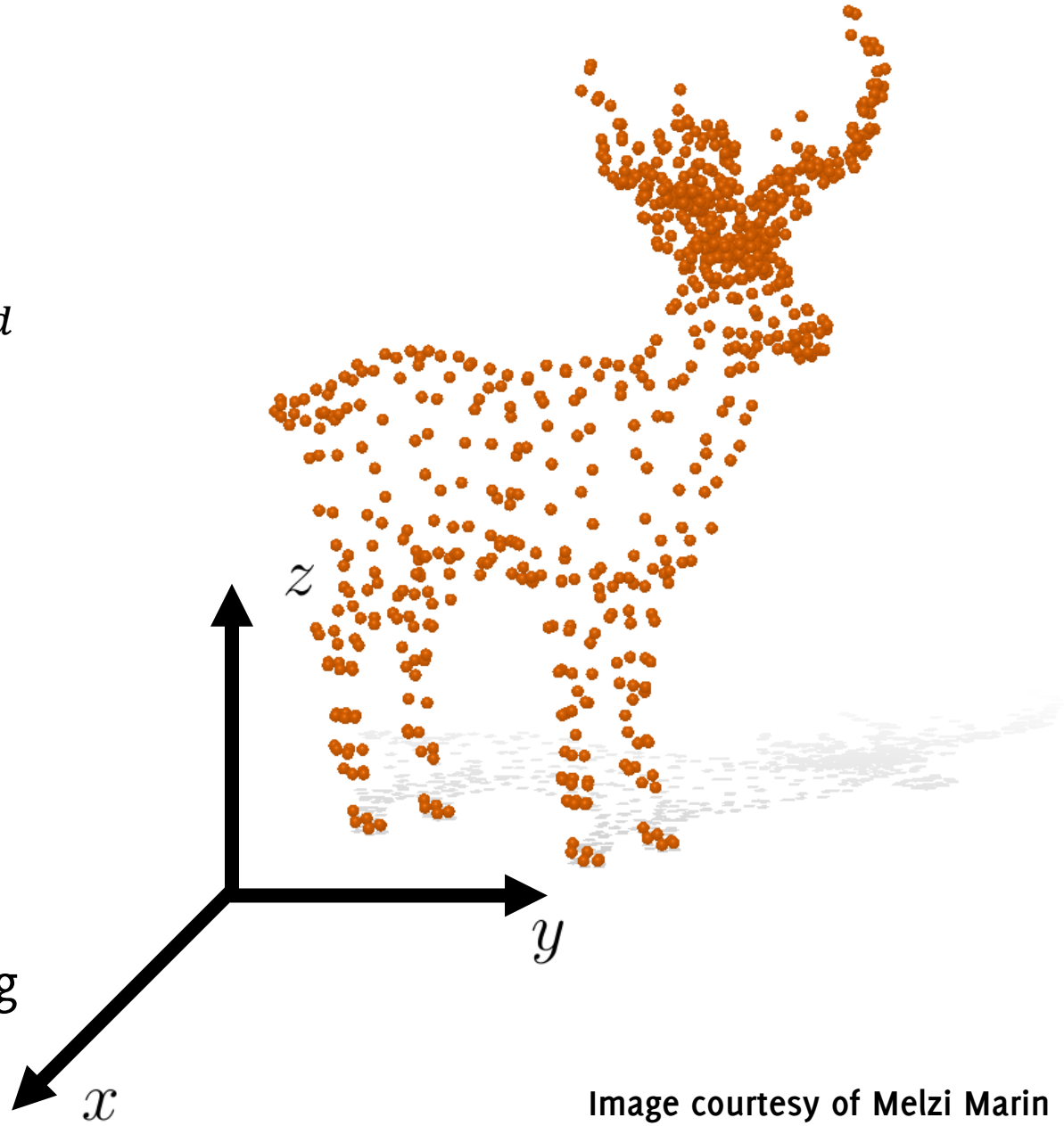


Image courtesy of Melzi Marin

To each 3D point it
to associate additi
information, like th
(RGB triplet), the n
the surface (anoth
normalized triplet)
practice also data-
features

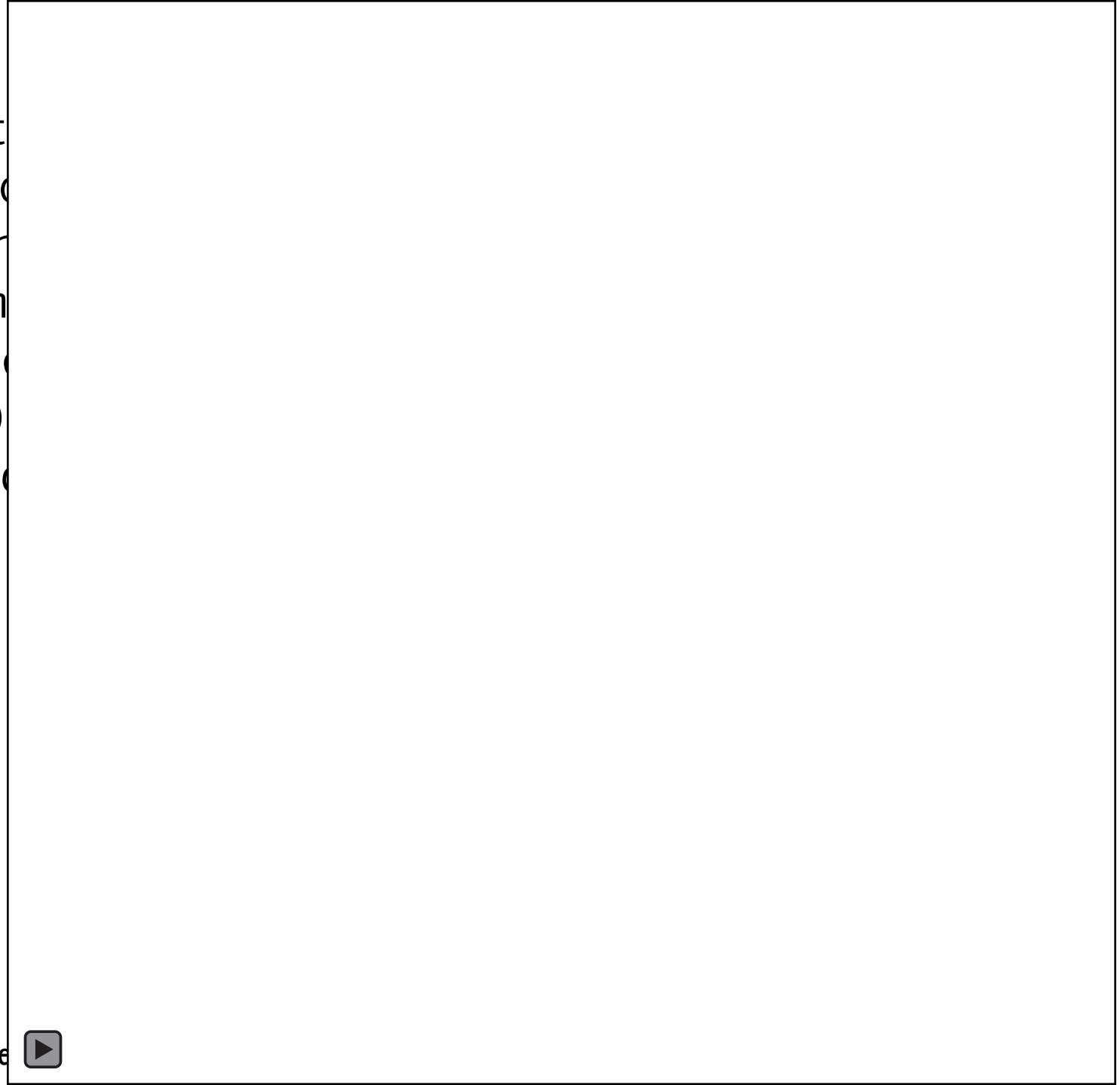


Image Courtesy of Giuseppe Be

A Pointwise point of view:

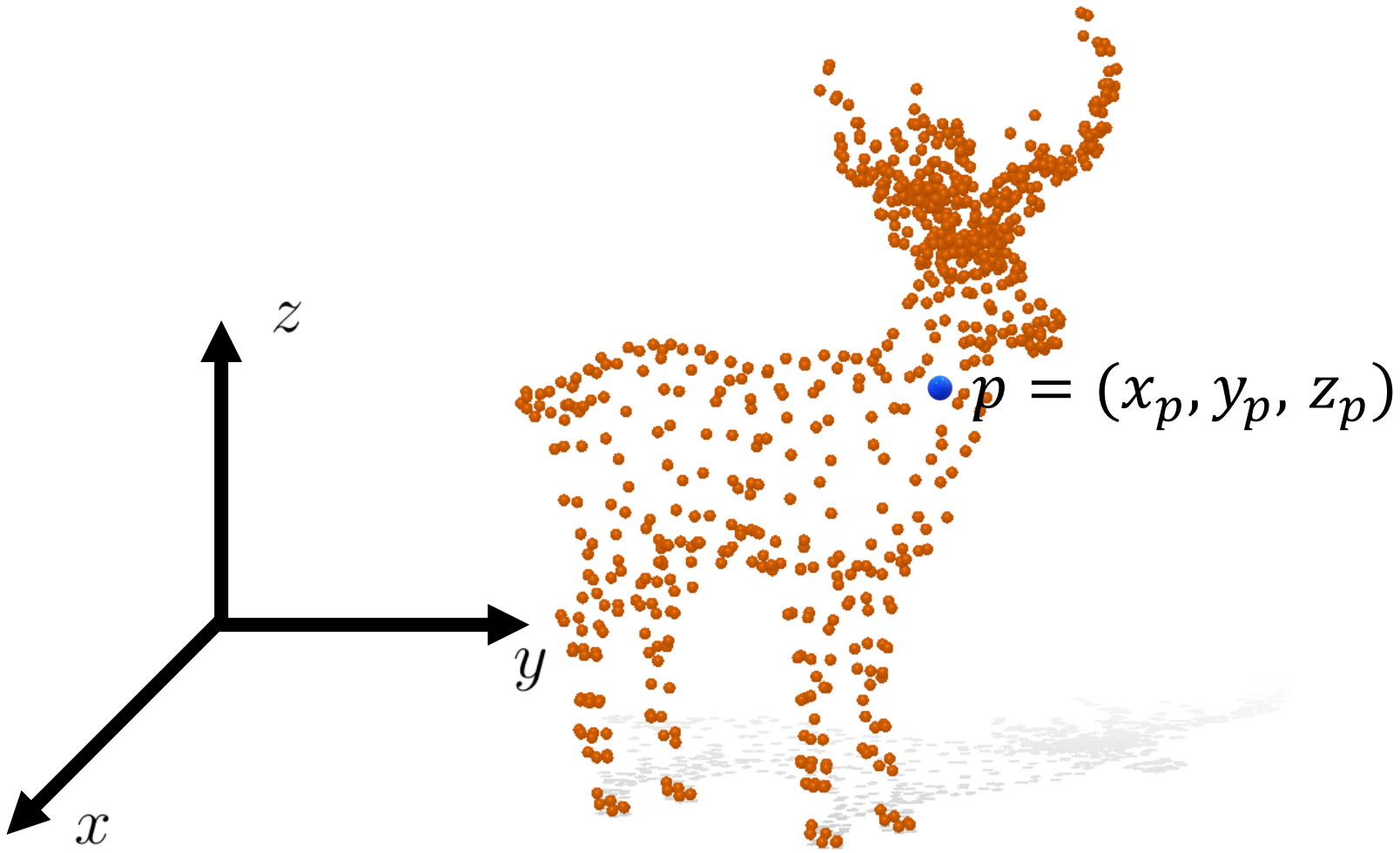


Image courtesy of Melzi Marin

A Pointwise point of view:

…an **arbitrary** reference system x, y, z would change all the input values!

What is the relation between
x'_p, y'_p, z'_p and x_p, y_p, z_p

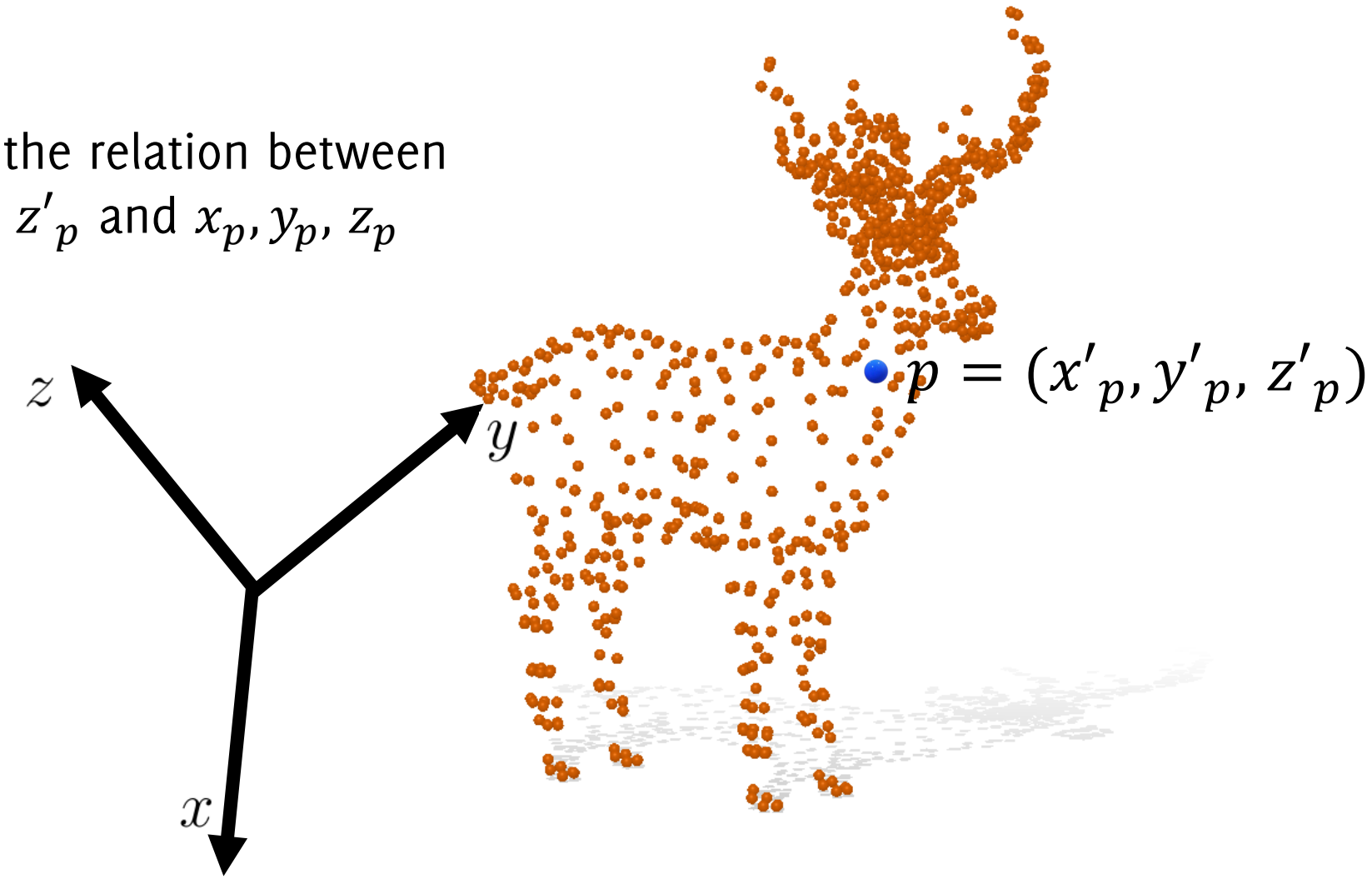
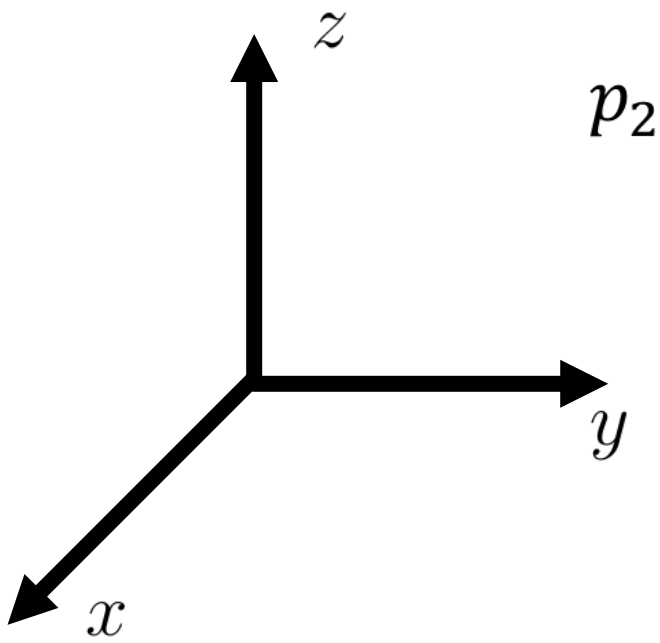
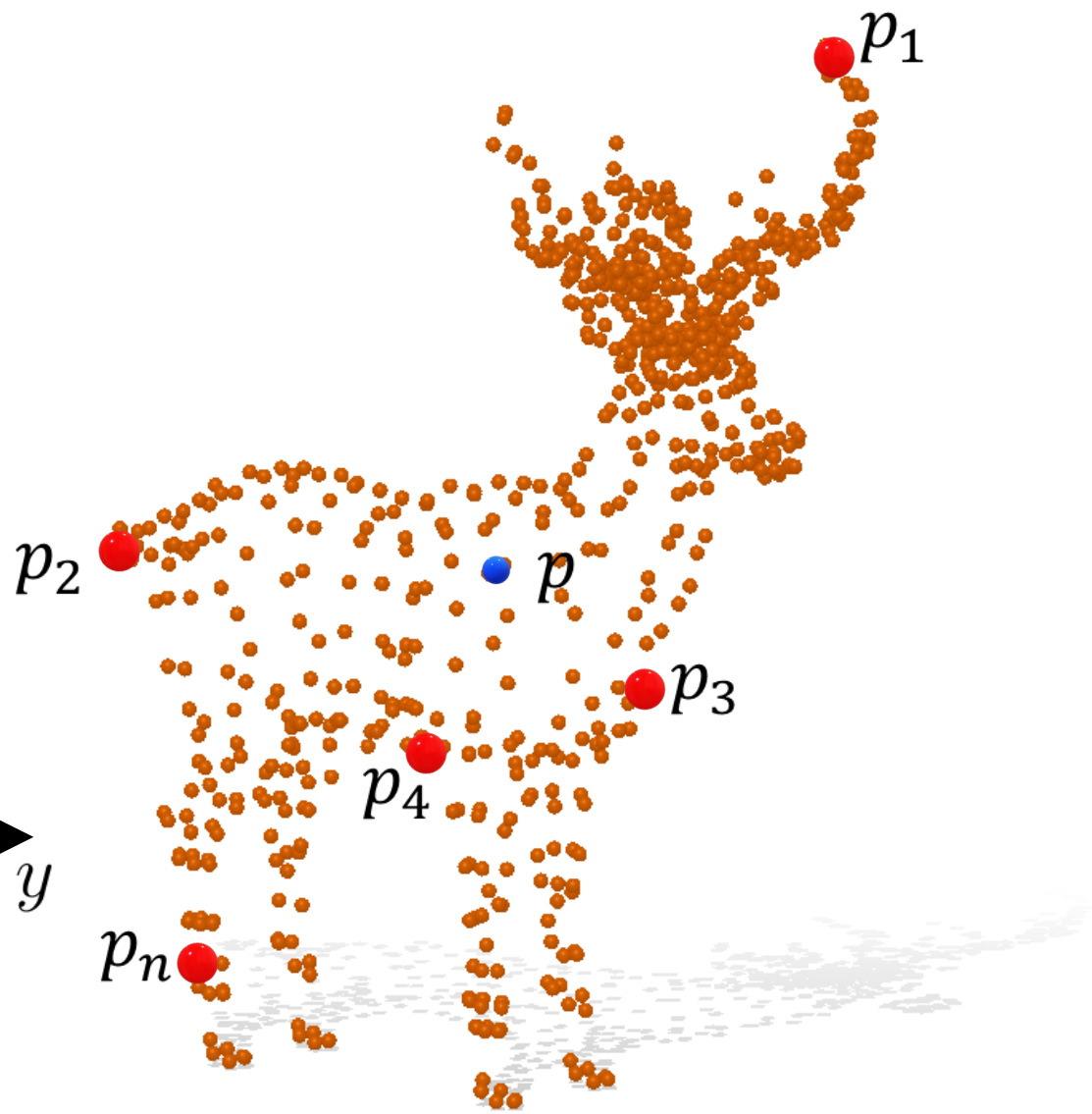


Image courtesy of Melzi Marin

A Matrix point of view:

n points sampled on a surface
+ a fixed reference system





We can order the points $p_1, p_2, p_3, p_4, \cdots, p_n$

Image courtesy of Melzi Marin

A Matrix point of view:

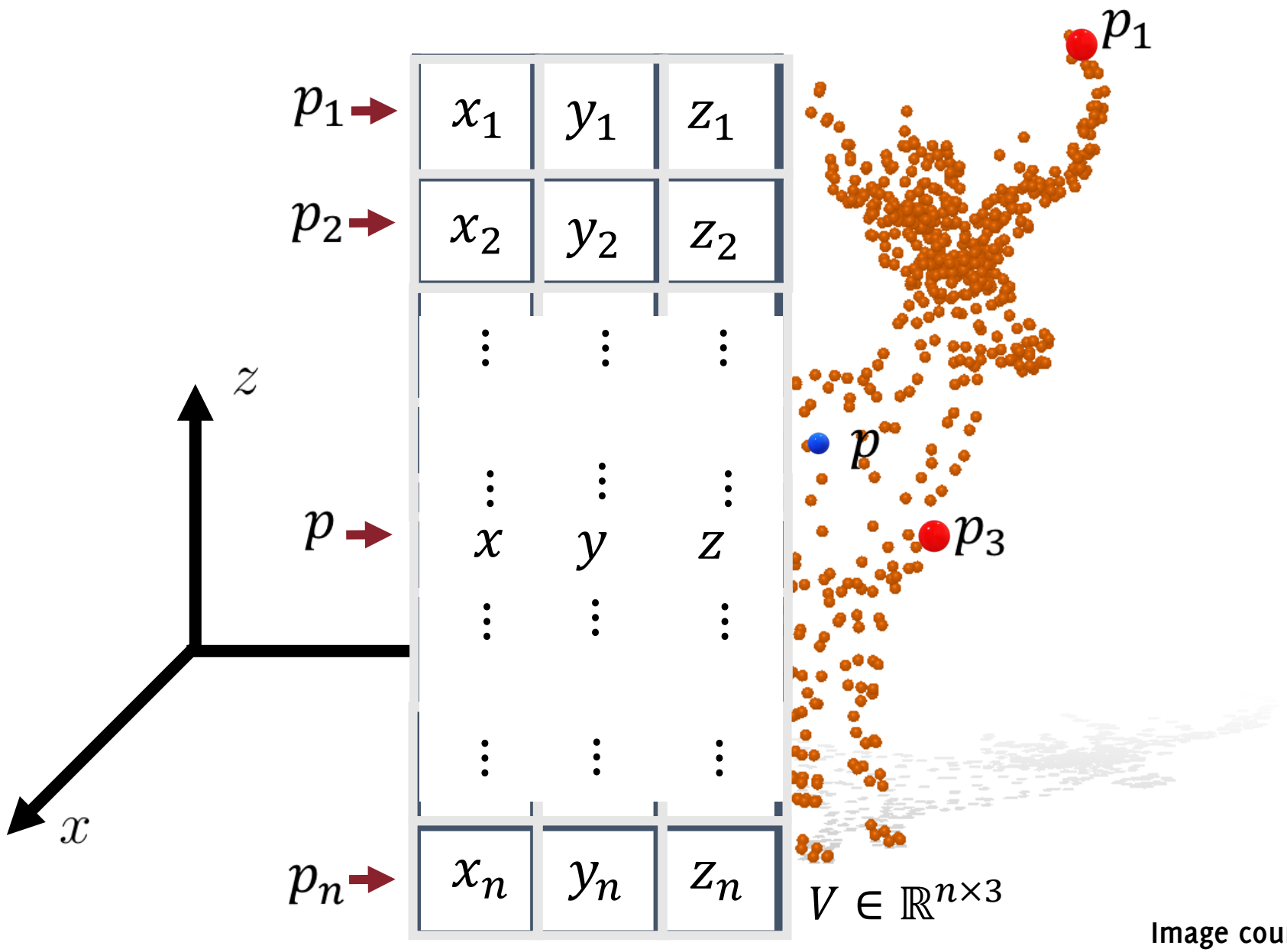
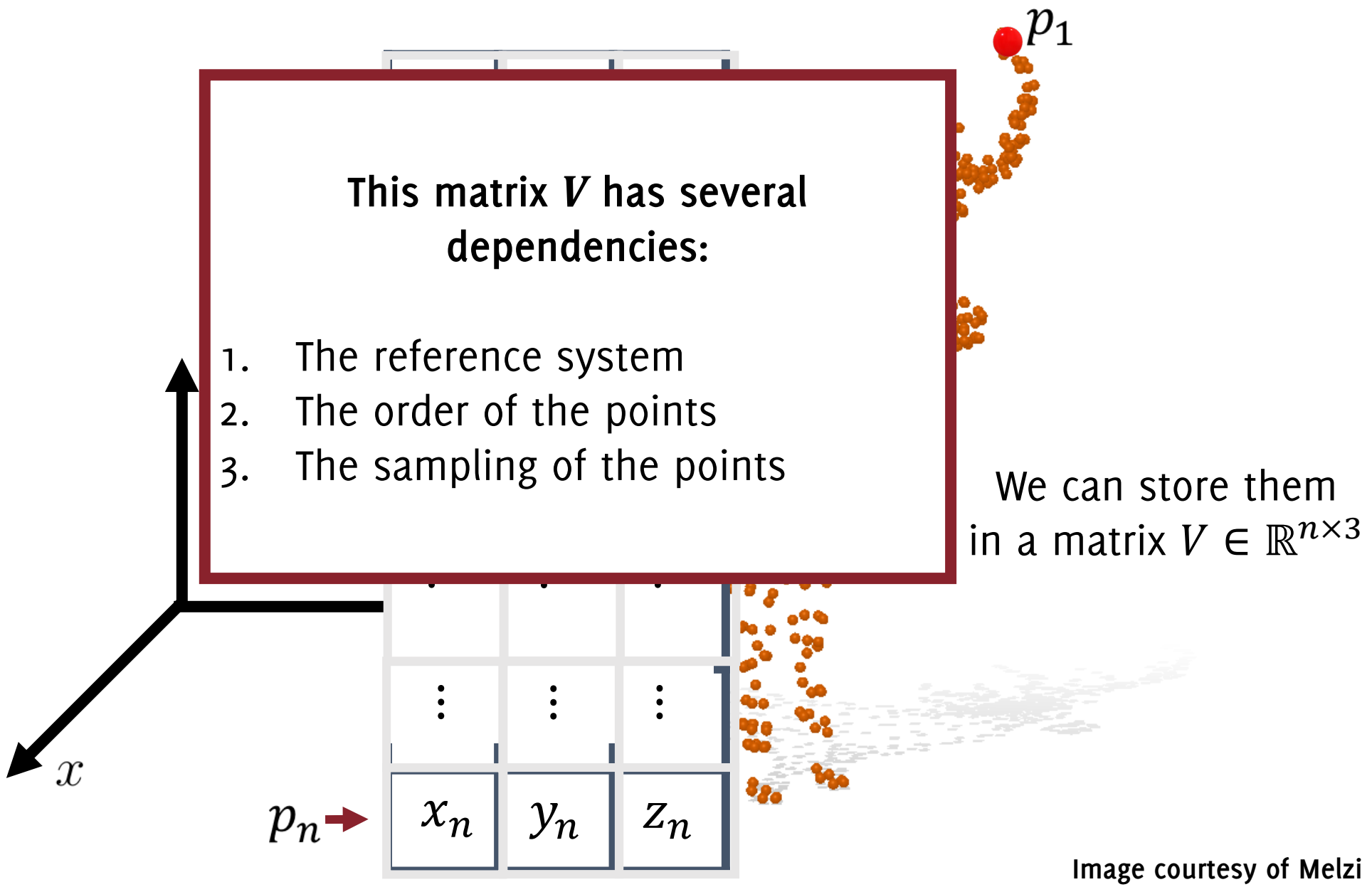


Image courtesy of Melzi Marin

A Matrix point of view:



We can store them in a matrix $V \in \mathbb{R}^{n \times 3}$

Image courtesy of Melzi Marin

Voxels

Represents the 3D word as a discrete grid $N \times N \times N$, primarily containing binary values. In each cell of the grid store whether it is occupied or not

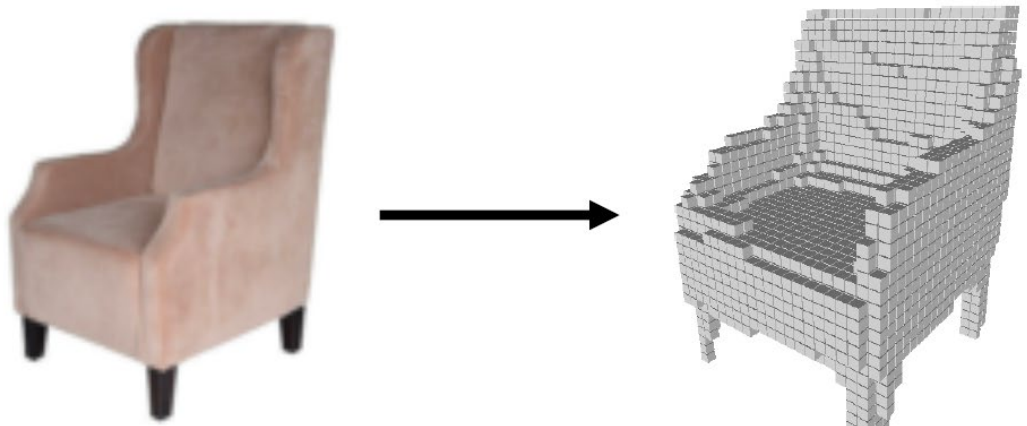
$$V \in \{0,1\}^{N \times N \times N}$$

Very simple representation of the 3D world (Minecraft like)

(+) Conceptually simple: just a 3D grid!

(-) Need high spatial resolution to capture fine structures

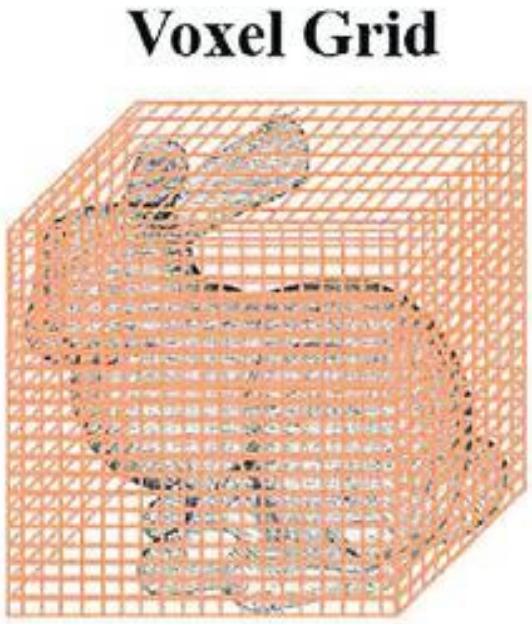
(-) Scaling to high resolutions is nontrivial!

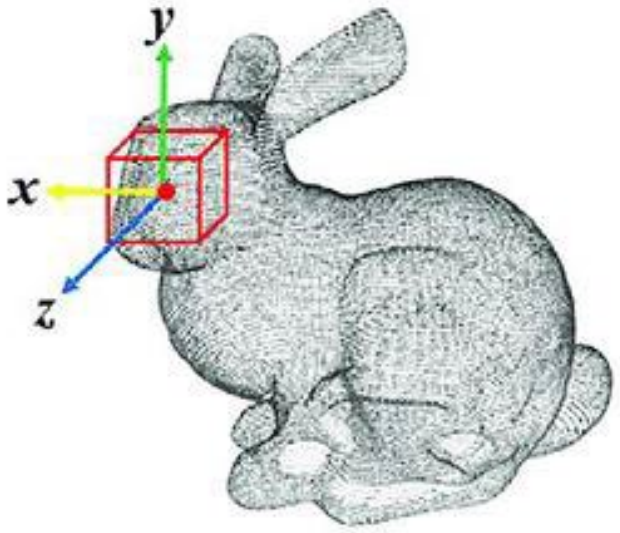


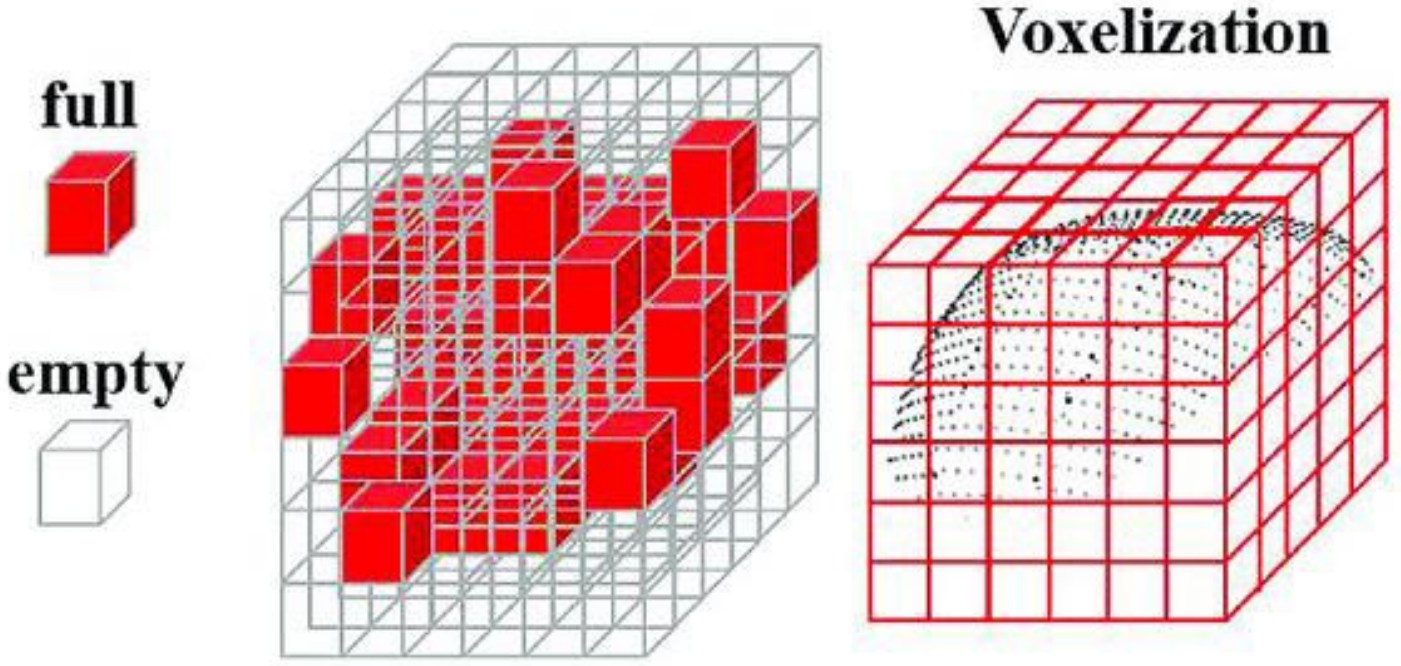
h for Single and Multi-view 3D Object Reconstruction", ECCV 2016

Voxels

Voxels can be obtained by discretizing point clouds, setting 1 whether the voxel area is full or to 0 where this is empty





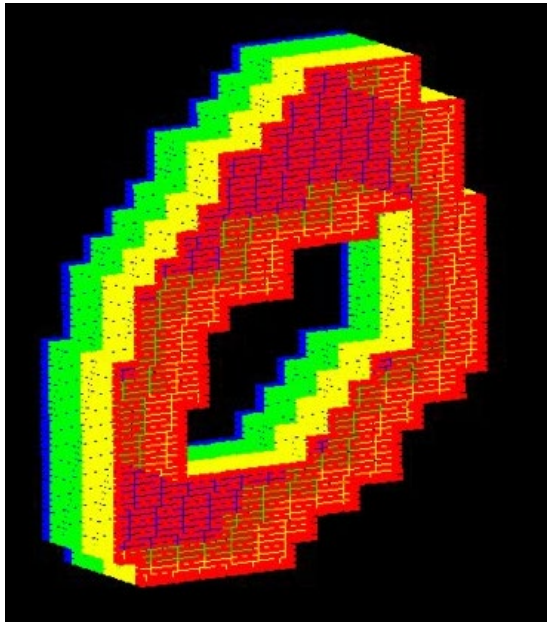


https://www.researchgate.net/figure/Drawing-of-the-LVS-descriptor-with-cubic-volume-definition-voxelization-and-voxel-label_fig3_328129223

Voxels and intensity information

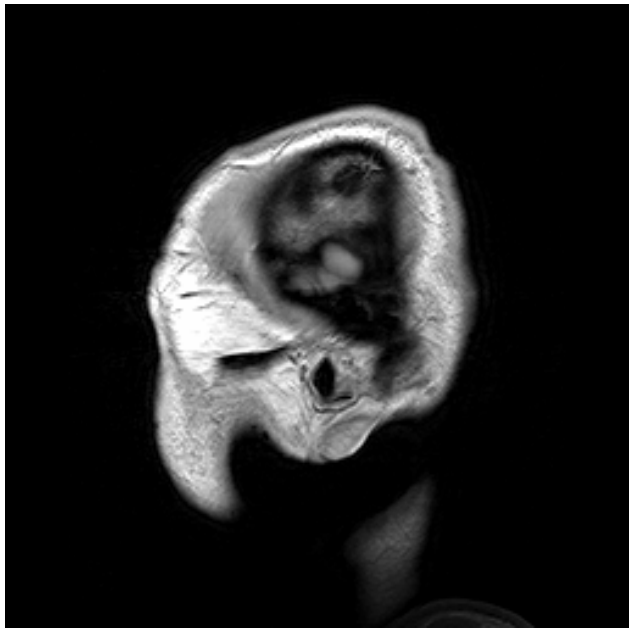
It is also possible to introduce in each voxel more information than $\{0,1\}$

3D mnist Toy dataset



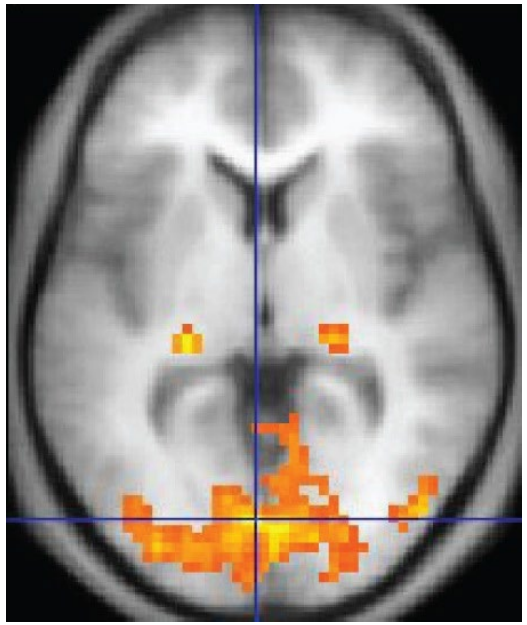
https://www.kaggle.com/datasets/daavoo/3d-mnist

MRI



By Dwayne Reed at English Wikipedia, CC BY-SA 3.0

fMRI

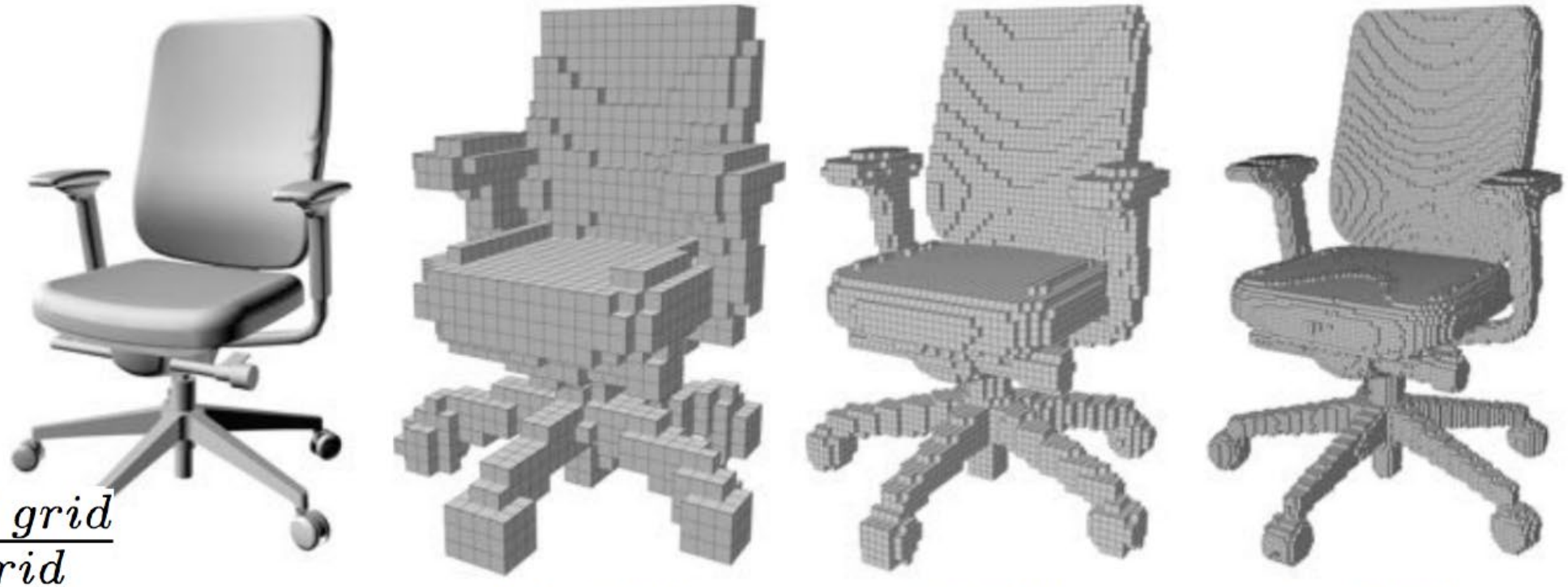


By OpenStax - https://cnx.org/contents/FPtK1zmh@8.25:fEI3C8Ot@10/Preface, CC BY 4.0

MRI with DTI

By Xavier Gigandet et. al. - Gigandet X, Hagmann P, Kurant M, Cammoun L, Meuli R, et al. (2008) Estimating the Confidence Level of White Matter Connections Obtained with MRI Tractography. PLoS ONE 3(12): e4006

Volumetric Data are typically sparse… in particular at high resolutions!



$$\frac{\#occupied\ grid}{\#total\ grid}$$

Occupancy:	10.41%	5.09%	2.41%
Resolution:	32	64	128

Image from CVPR2017 Tutorial from Hao Su

Point Clouds vs Voxels

Pros of Point Clouds:

- **More compact representation** (does not allocate space for empty regions)
- They can «adapt» and be **denser** in regions where there are **finer details**.

Cons of Point Clouds:

- Points **are scattered**, i.e., they do not belong to a grid (and when you project them on a grid you loose the above advantages)
- NN / CNN are meant for vectors / arrays! **We need new architectures**
- Points have size = 0, they do not represent the 3D object, in principle.
- **PC representation depends on the reference axis**
- PC are sets, as such the **order of points is irrelevant**. This is in contrast with CNN, based on the concept of locality (neighbors defined over the grid)

RGB-D Images

These are images captured by raw (possibly low cost) 3D sensors like Microsoft Kinect / Intel Real Sense, Face ID on iPhone.

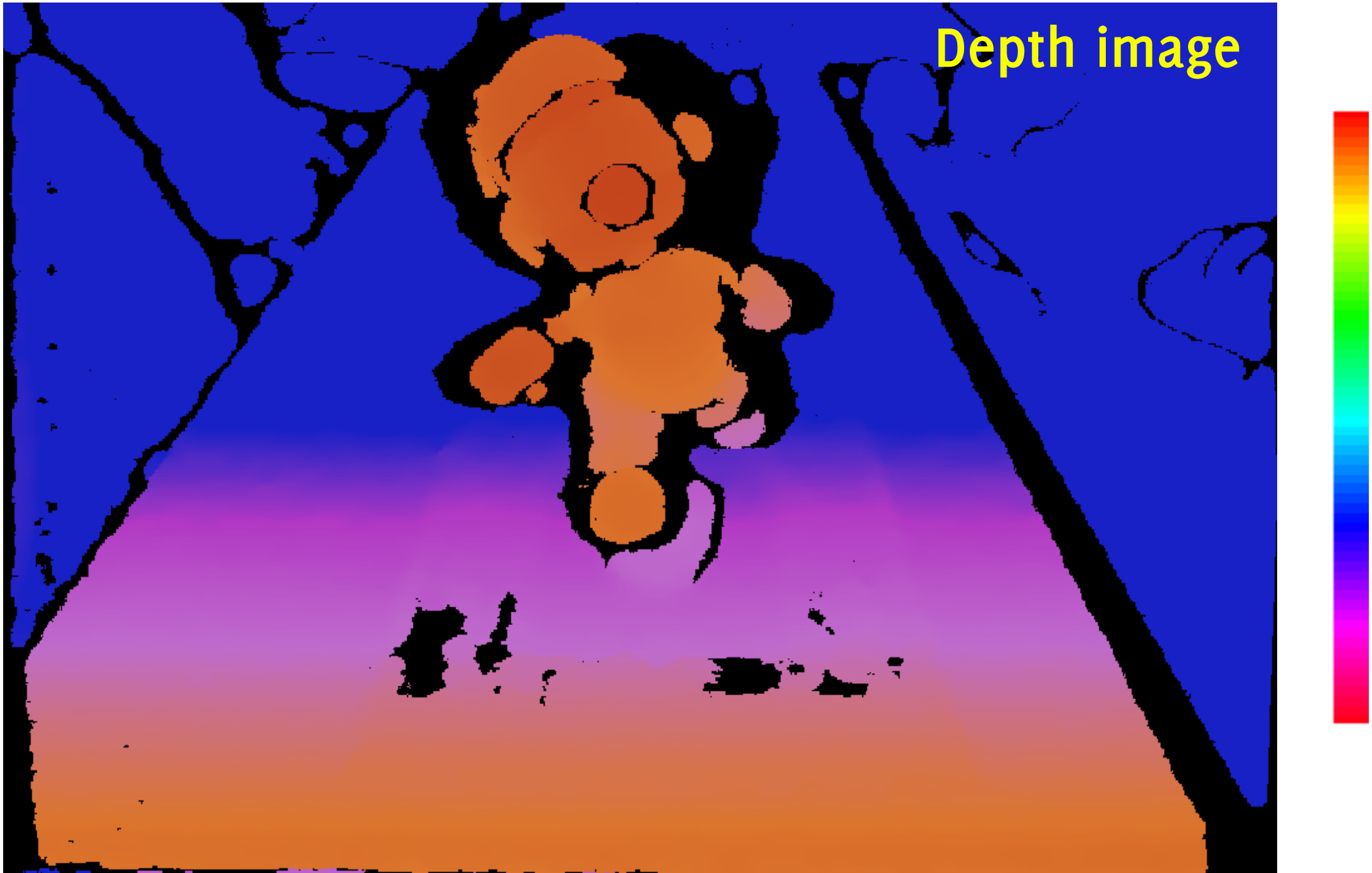
The returned image has 4 values per pixel

- 3 are standard RGB
- 1 is the depth

These are called **2.5D images**, as they are not «fully» 3D data, as they cannot capture occluded objects

Slightly less powerful representation than native 3D data.


RGB


Depth image

RGB-D Images

These images are also called 2.5D images, as they are not «fully» 3D data, since they cannot capture the structure behind occlusions!

Slightly less powerful representation than native 3D data.

We cannot “sense” behind occlusions
(e.g. the table behind the object)

Shadows as well as nonreflecting surfaces give rise to missing values

Implicit Function

We can also represent the 3D object as a function

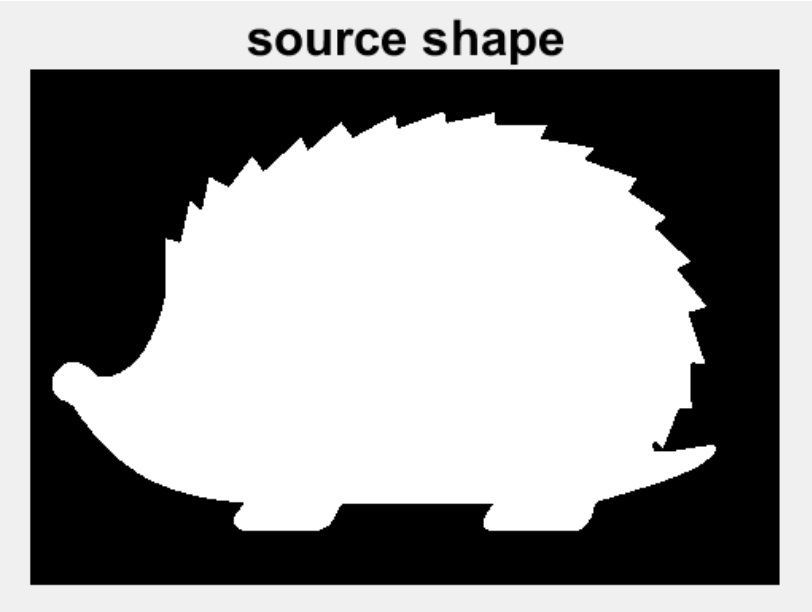
$$f: \mathbb{R}^3 \rightarrow \{0,1\}$$

where $\{0,1\}$ denotes being occupied or not, or

$$f: \mathbb{R}^3 \rightarrow \mathbb{R}$$

This representation identifies a surface as the 0-level set of f

We can also consider the voxelized representation as a discretized version of the implicit function



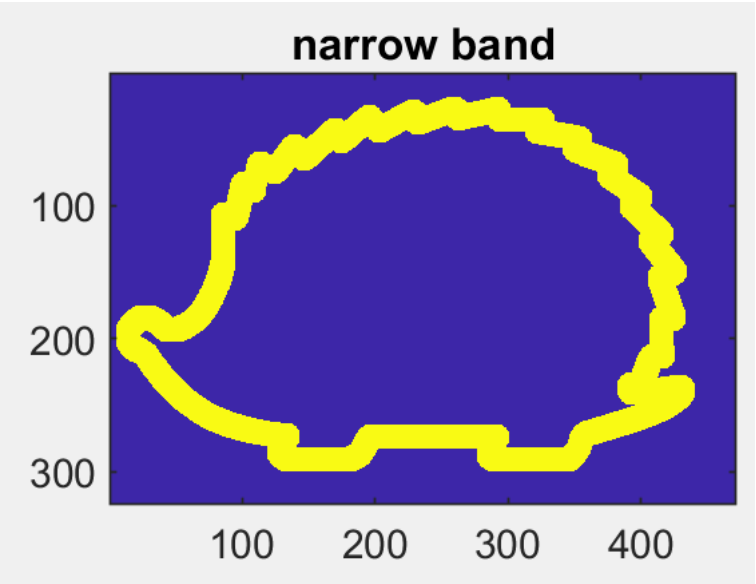


Image courtesy of Melzi Marin

Implicit Function

We can also represent the 3D object as a function

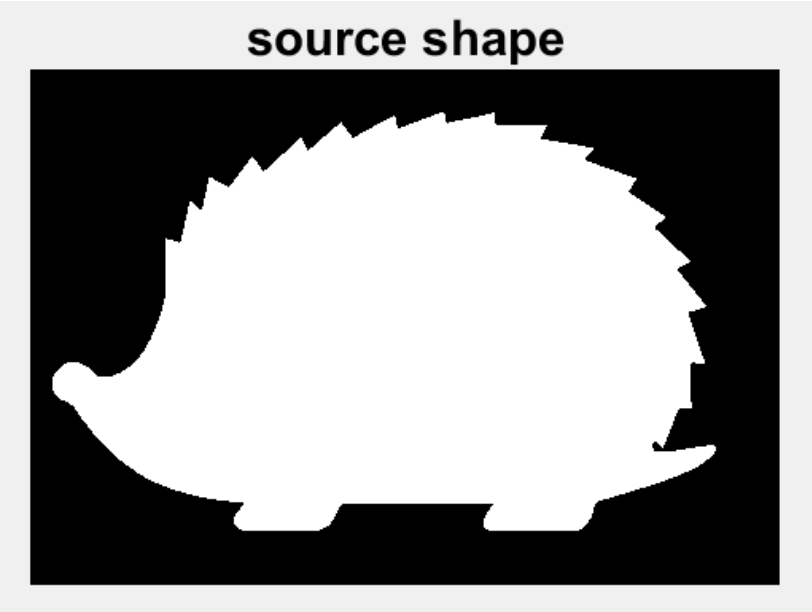
$$f: \mathbb{R}^3 \rightarrow \{0,1\}$$

where $\{0,1\}$ denotes being occupied or not, or

$$f: \mathbb{R}^3 \rightarrow \mathbb{R}$$

This representation identifies a surface as the 0-level set of f

We can also consider the voxelized representation as a discretized version of the implicit function



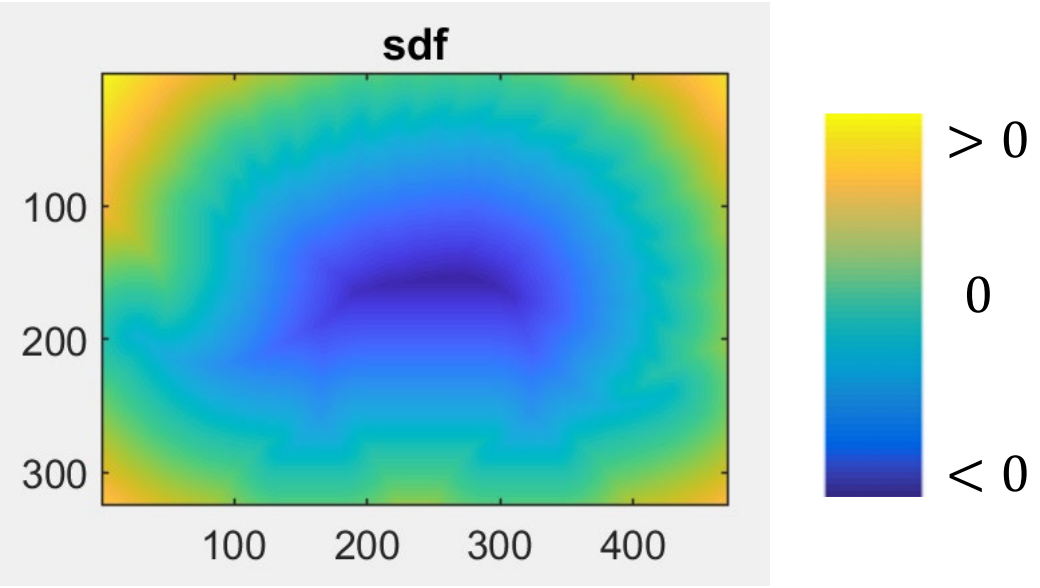


Image courtesy of Melzi Marin

Implicit surfaces

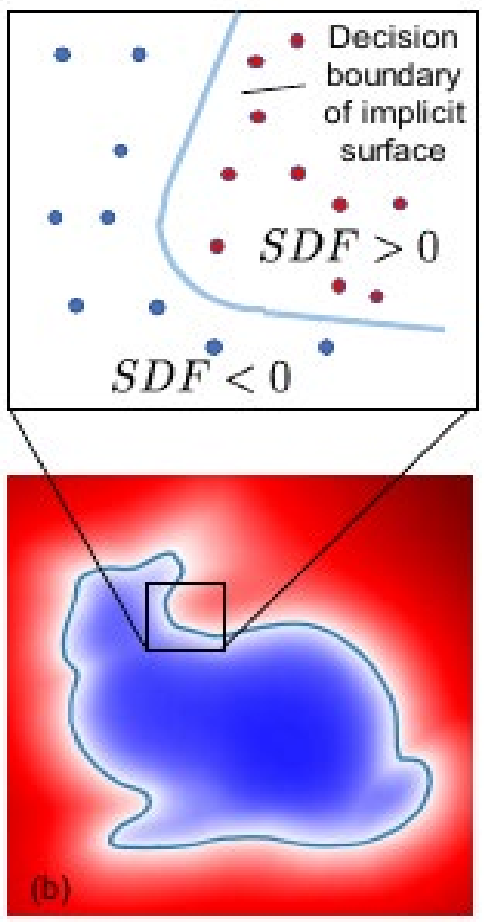


Image from http://people.csail.mit.edu/jstraub/

Signed distance function SDF

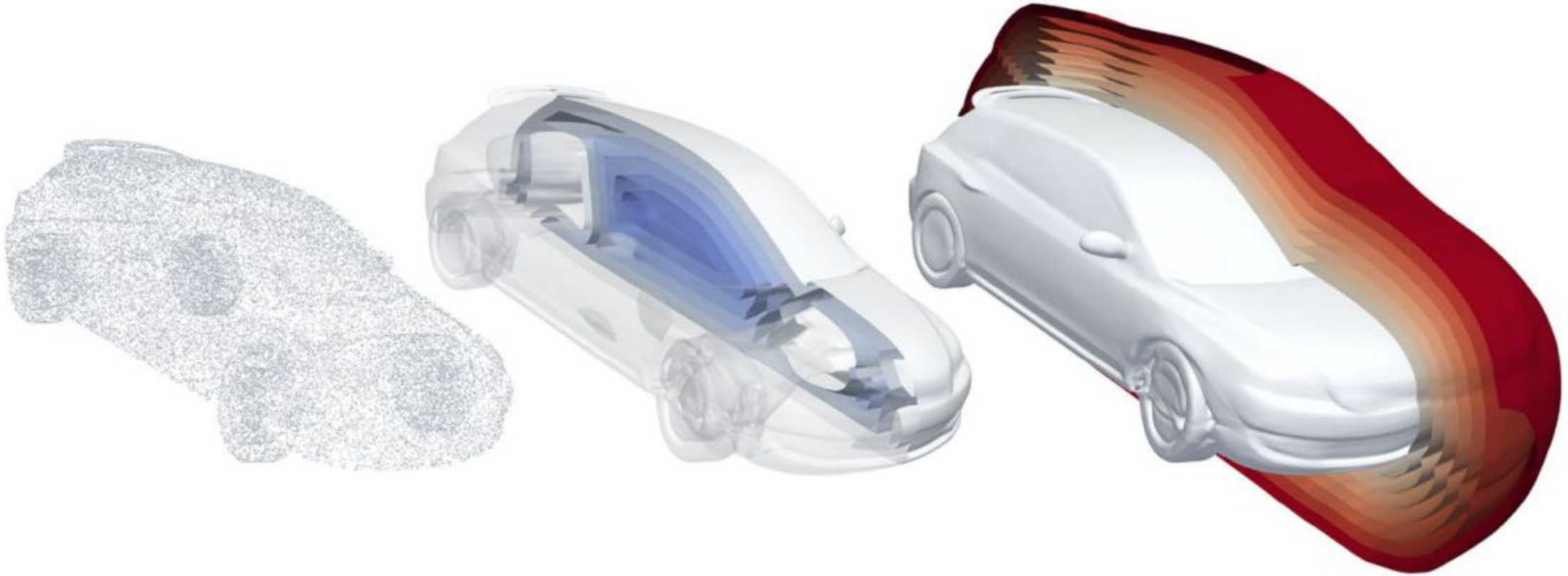


Image from https://arxiv.org/pdf/2002.10099.pdf

Meshes

A very common representation in Computer Graphics.

It is a pair (P, F) where:

- P vertices, a point cloud in 3D
$$P \subset \mathbb{R}^3$$
- E edges set of pair of points connecting vertices
$$E \subset \mathbb{R}^3 \times \mathbb{R}^3$$
- F faces, this is a set of triangular faces over vertices
$$F \subset \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$$

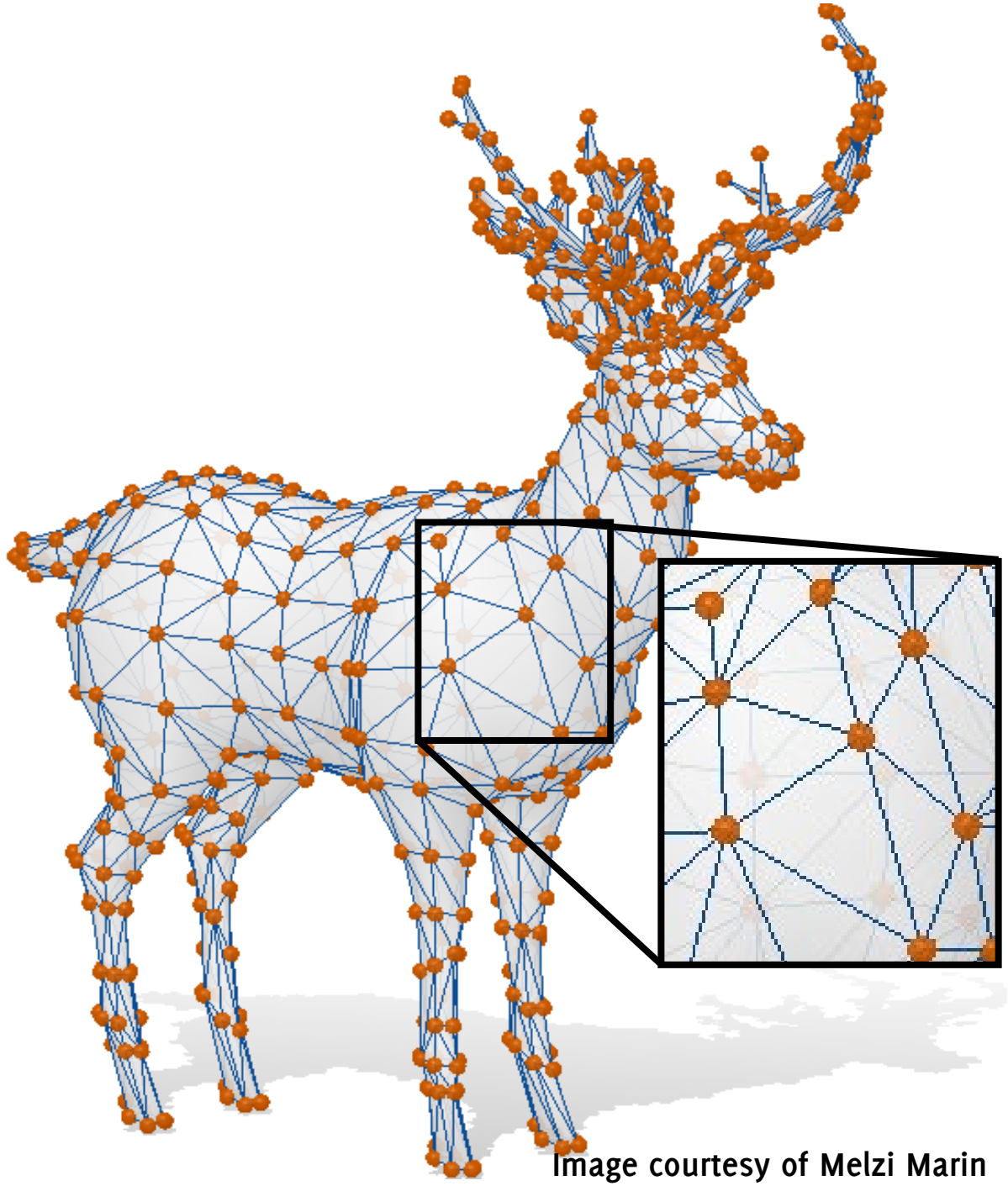


Image courtesy of Melzi Marin

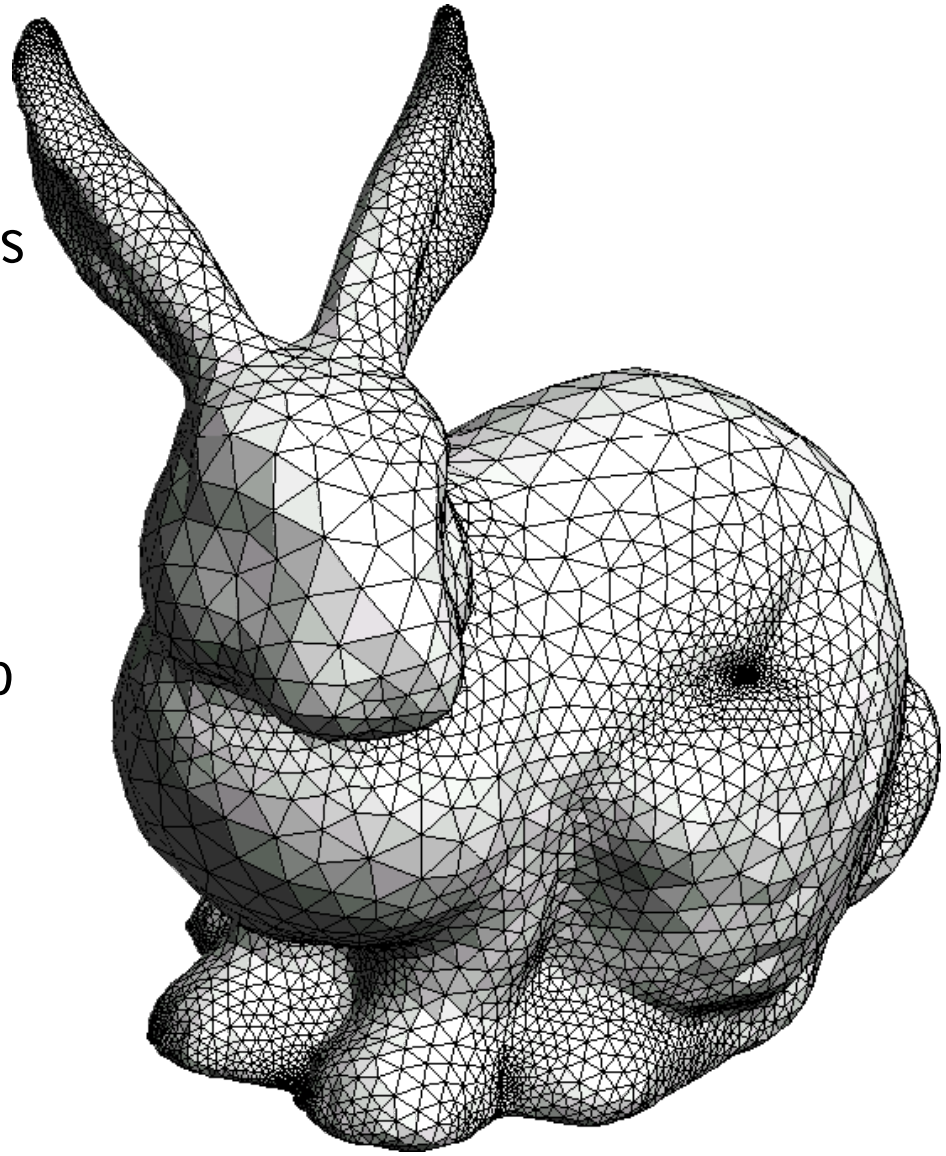
Meshes

Pros:

- Standard in Computer Graphics
- They represents 3D shapes, not just points or voxels
- They are adaptive: very efficient for flat surfaces, can allocate more vertices at fine details
- Can attach data on vertices: RGB colors, texture coordinates, normal vectors, etc.
- Can natively interpolate values on vertices and map to other surfaces.
- They are a special type of graphs

Cons:

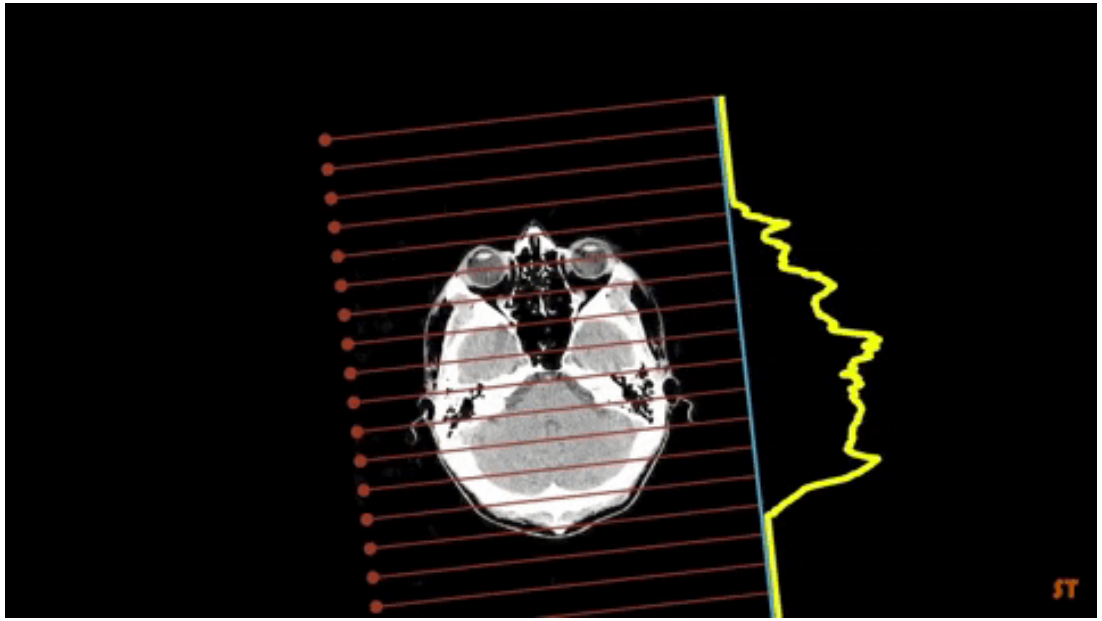
- Nontrivial feeding this structure in NN, they require special NN architecture

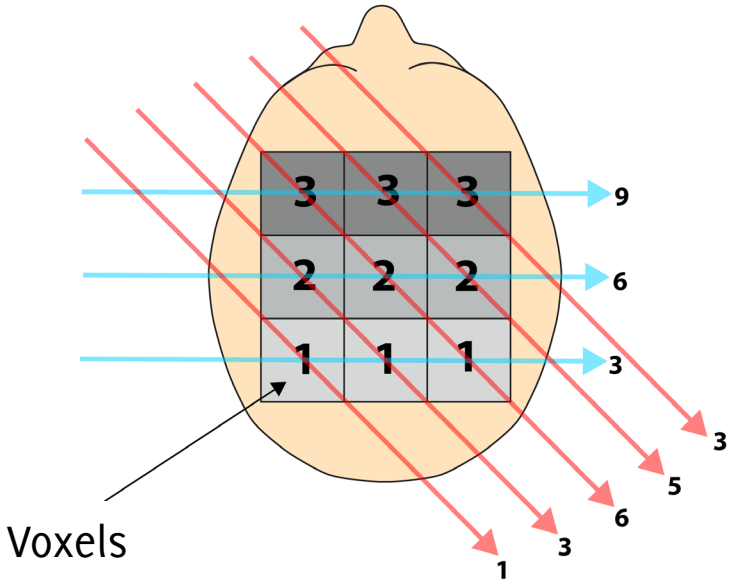


3D Imaging Sensors

Computed Tomography Scan

- **Diagnostic imaging technique** to obtain detailed internal images of the body.
- Multiple X-ray measurements taken from **different angles** are processed on a computer using **tomographic reconstruction** algorithms (such as filtered back projection) to produce cross-sectional images as virtual *slices* of a body.
- Data are discretized using a grid of **voxels** containing the CT number, proportional to the **local value of the density.**

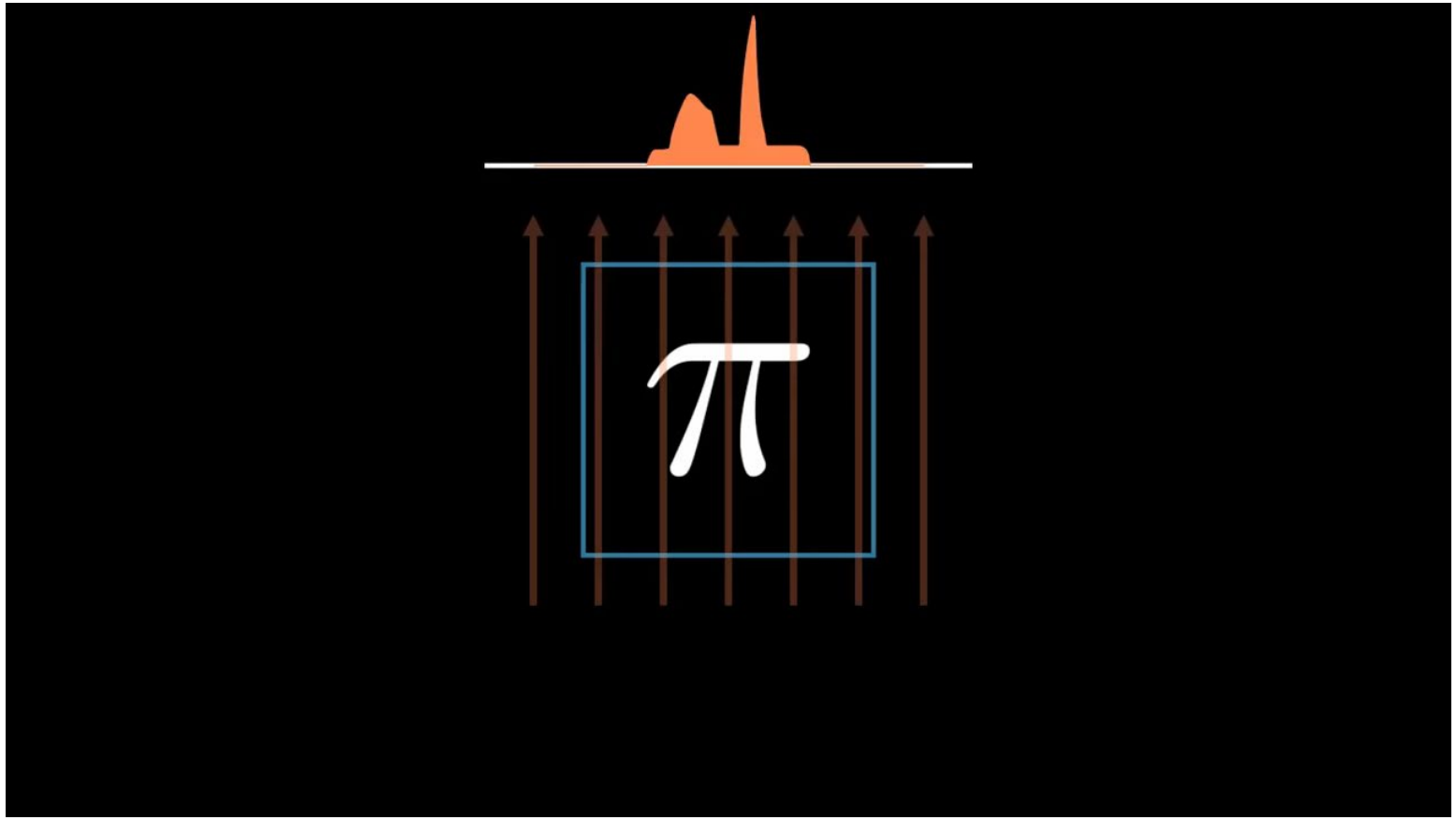




https://ssip2021.riteh.hr/student-projects/p18/#services https://maxfacts.uk/diagnosis/tests/ct-scans

Filtered Back Projection: Intuition

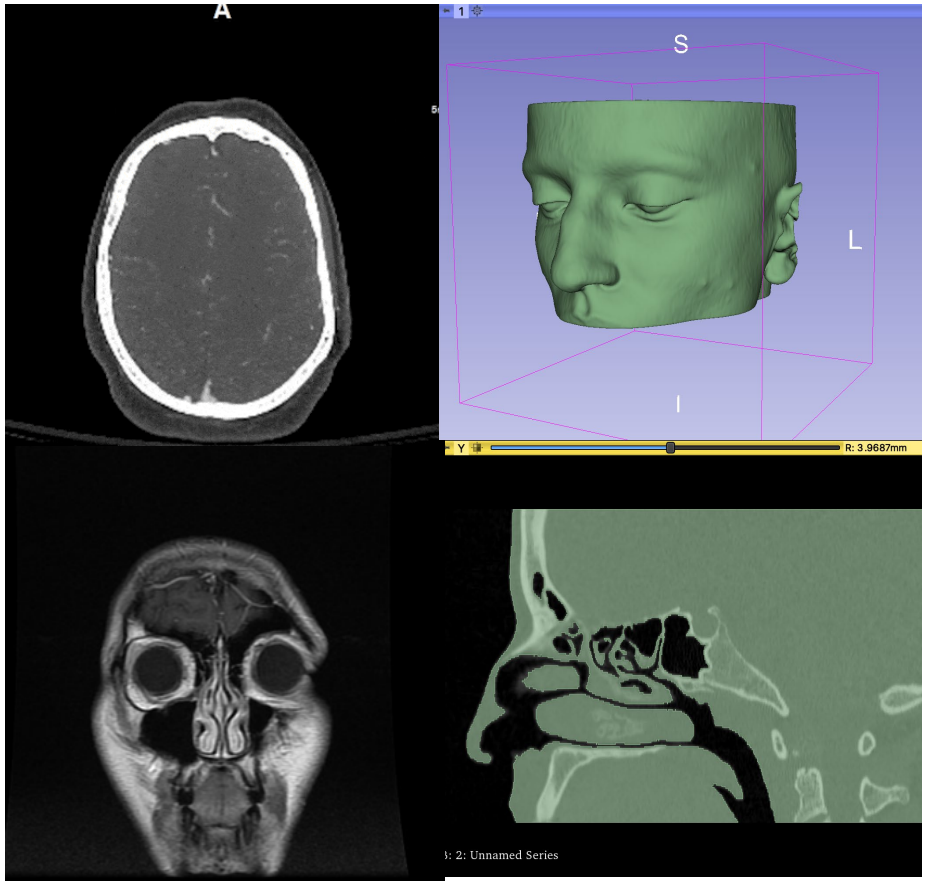
Projections acquired from the CT Scan are i) filtered (high-passed) and ii) projected back to the 2D space. Summing all the filtered backprojection the image is reconstructed



Author Kolibril13 from Wikipedia https://en.wikipedia.org/wiki/File:Tomographic_reconstruction-_Projection,_Back_projection_and_Filtered_back_projection.webm#metadata

Computed Tomography Scan

The *slicing process* in the 3D space returns a reconstructed **3D volume** of the scanned object, which is discretized in a grid of 3D voxels.



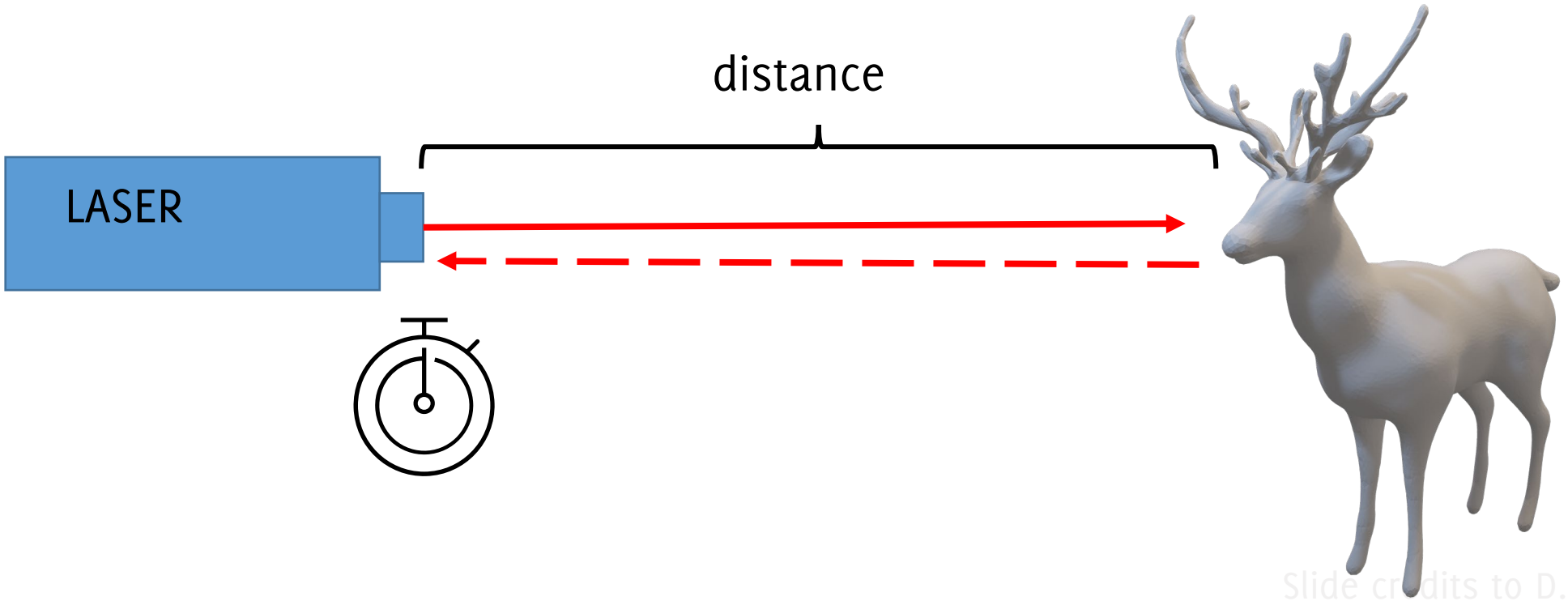
By setting a treshold on the CT number, we can extract 3D surfaces, removing areas where the density (the CT number) is locally low, i.e., empty cavities.

LIDAR ToL / Time of Flight Cameras

Measures the time it takes the laser beam to hit the object and come back

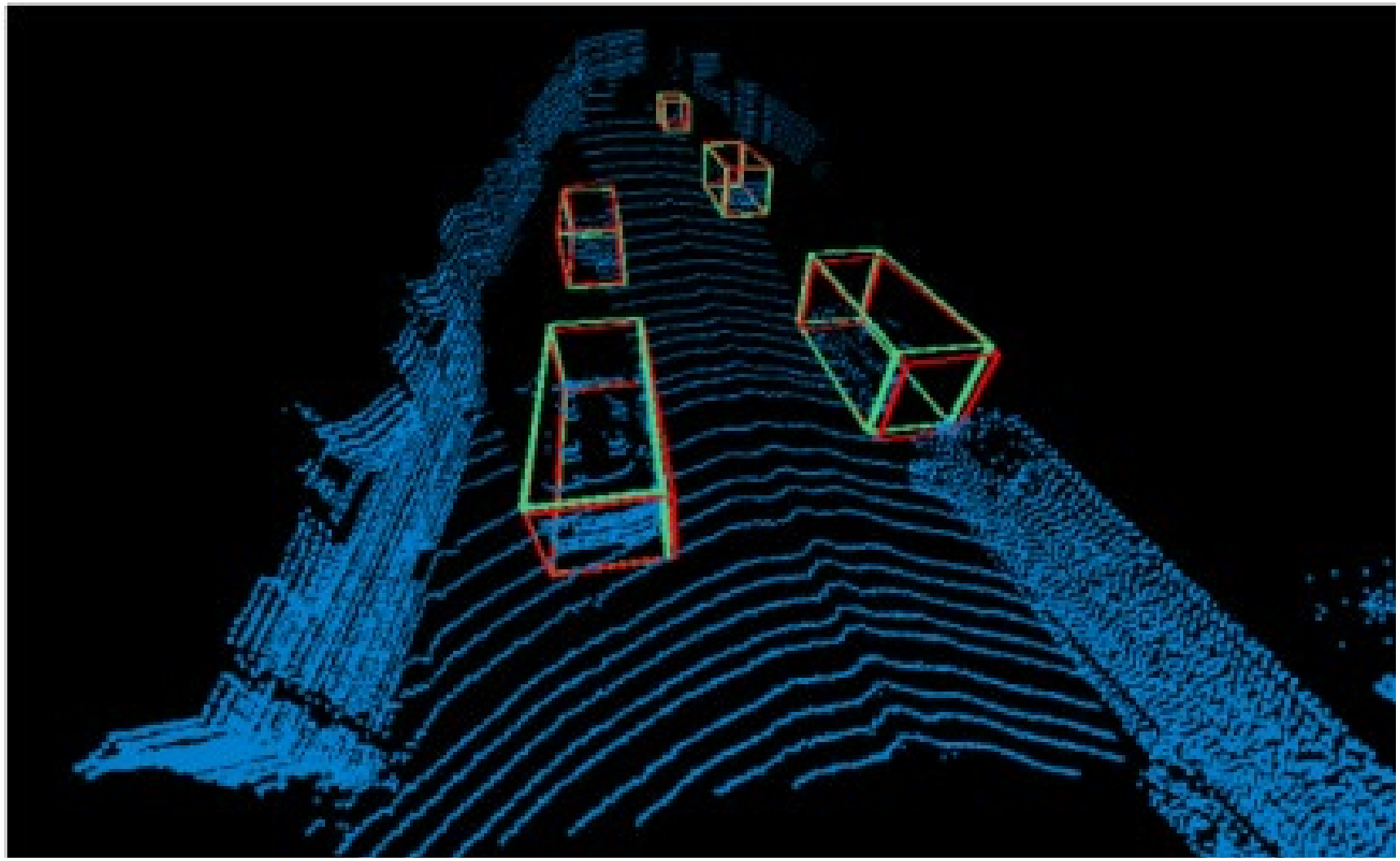
Time of Flight Cameras have an array of sensors to acquire the distance of the whole scene in one shot,

Lidars emit a laser pulse that sweeps the entire scene to reconstruct the depths of the scene one line at a time.

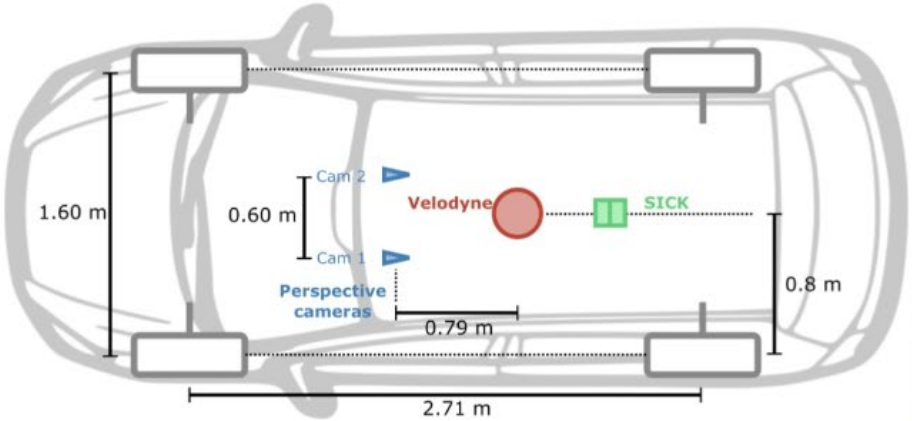


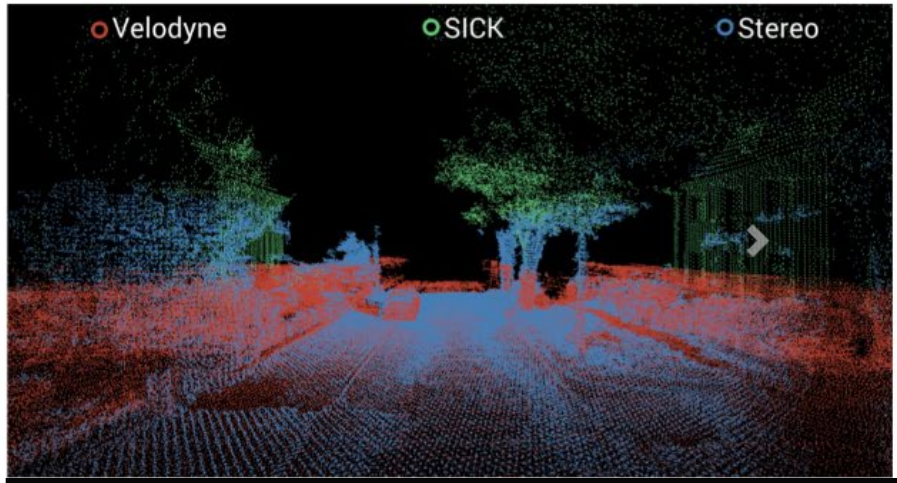
Slide credits to D. Panozzo

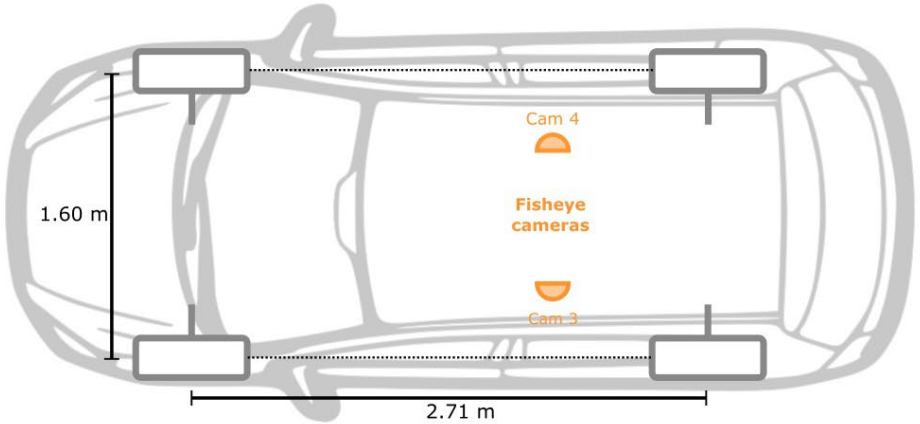
Scanning patterns of Lidar

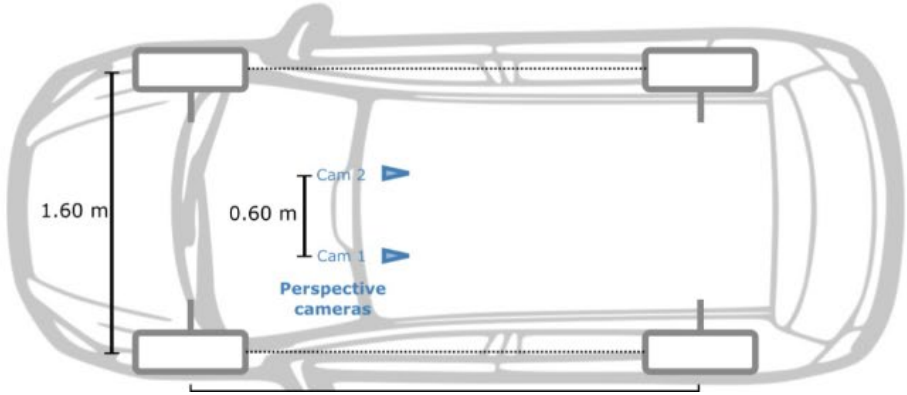


Autonomous vehicles are equipped with multiple vision sensor, including natively 3D ones (LIDAR)







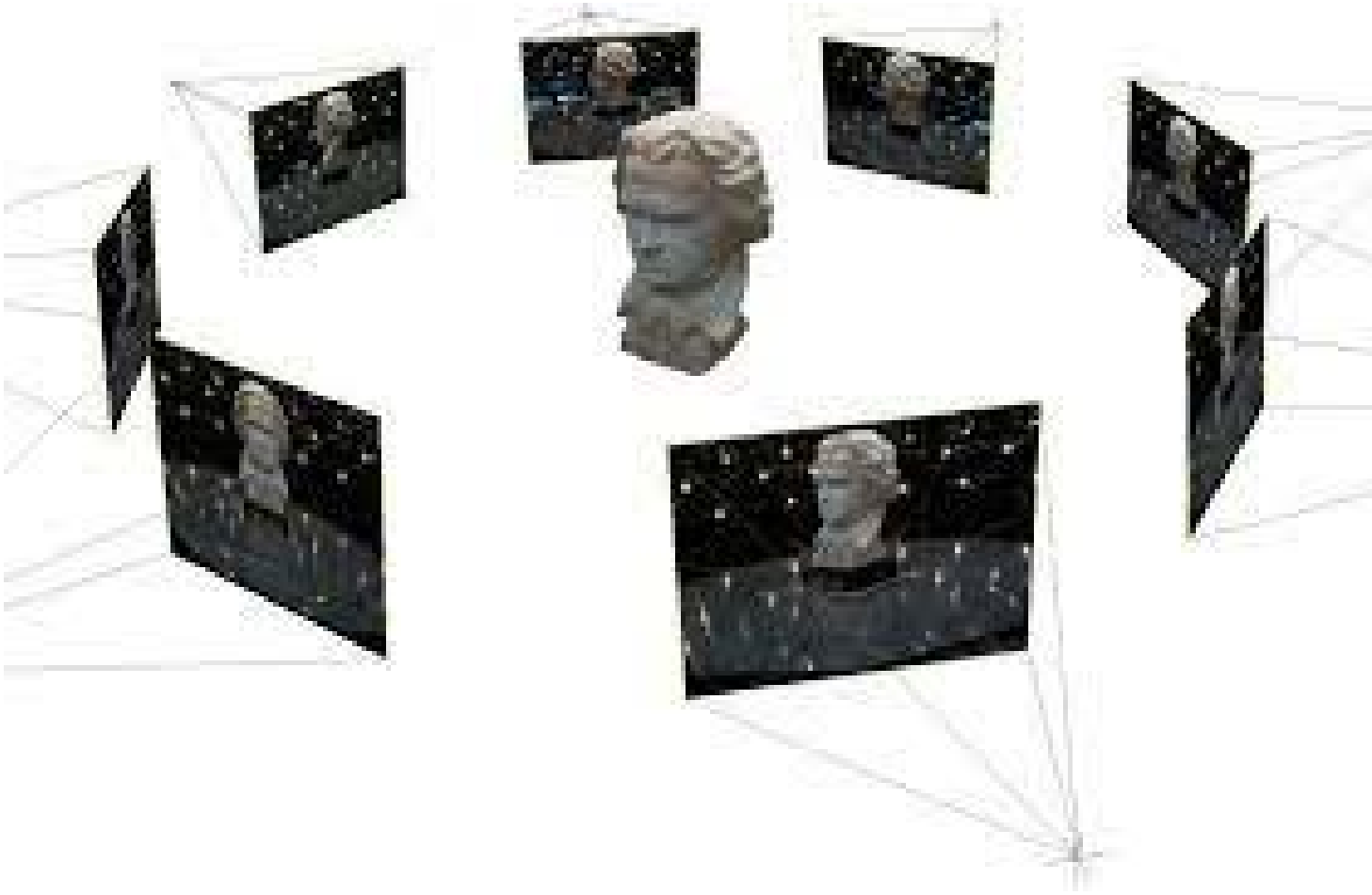


https://www.cvlibs.net/datasets/kitti-360/index.php

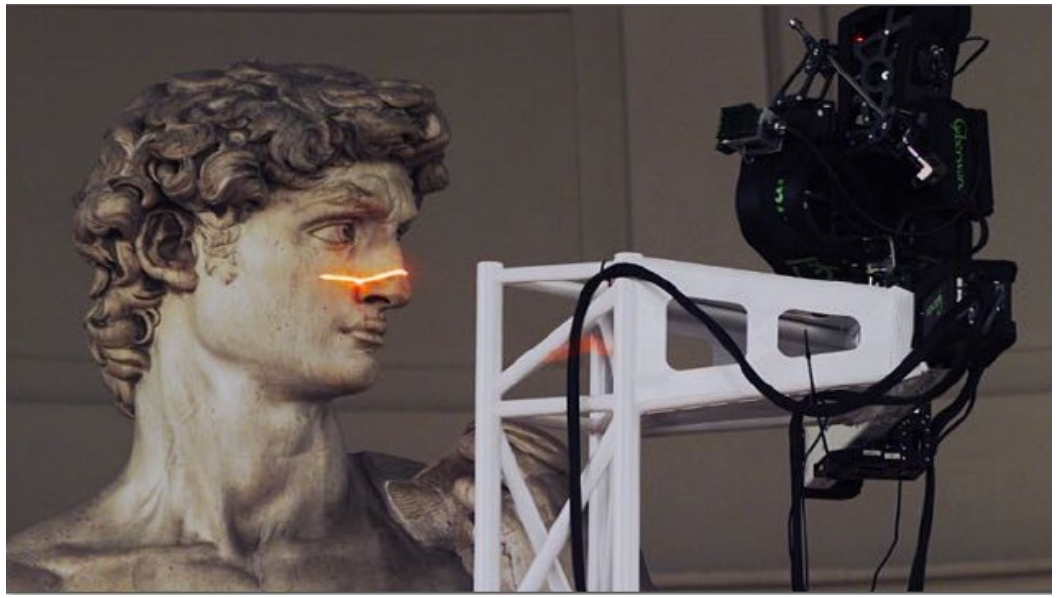
Multiview

A simple camera can be a 3D sensing device as far as there are multiple pictures acquired from different viewpoints.

Ad-hoc software like meta-shape, colmap etc.. can both reconstruct the camera pose (3D location and orientation) and the 3D shape of the depicted object

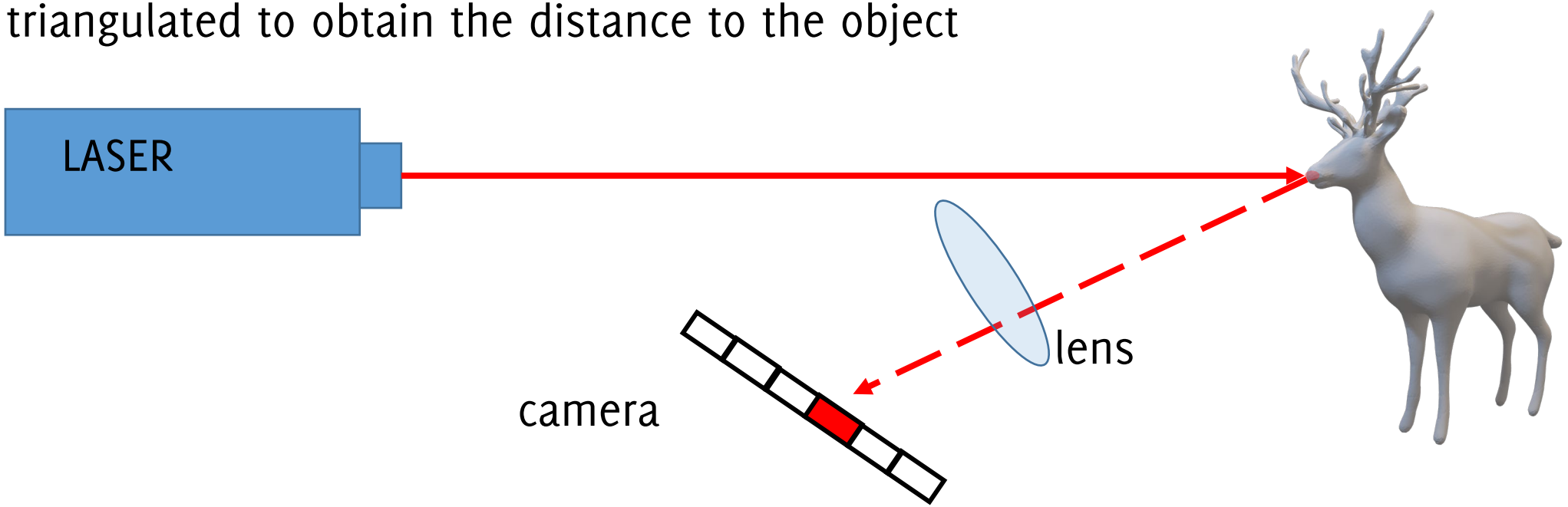


Laser triangulation



- A laser beam
- A camera

1. Laser dot/blade is captured
2. The location of the dot /blade is triangulated to obtain the distance to the object



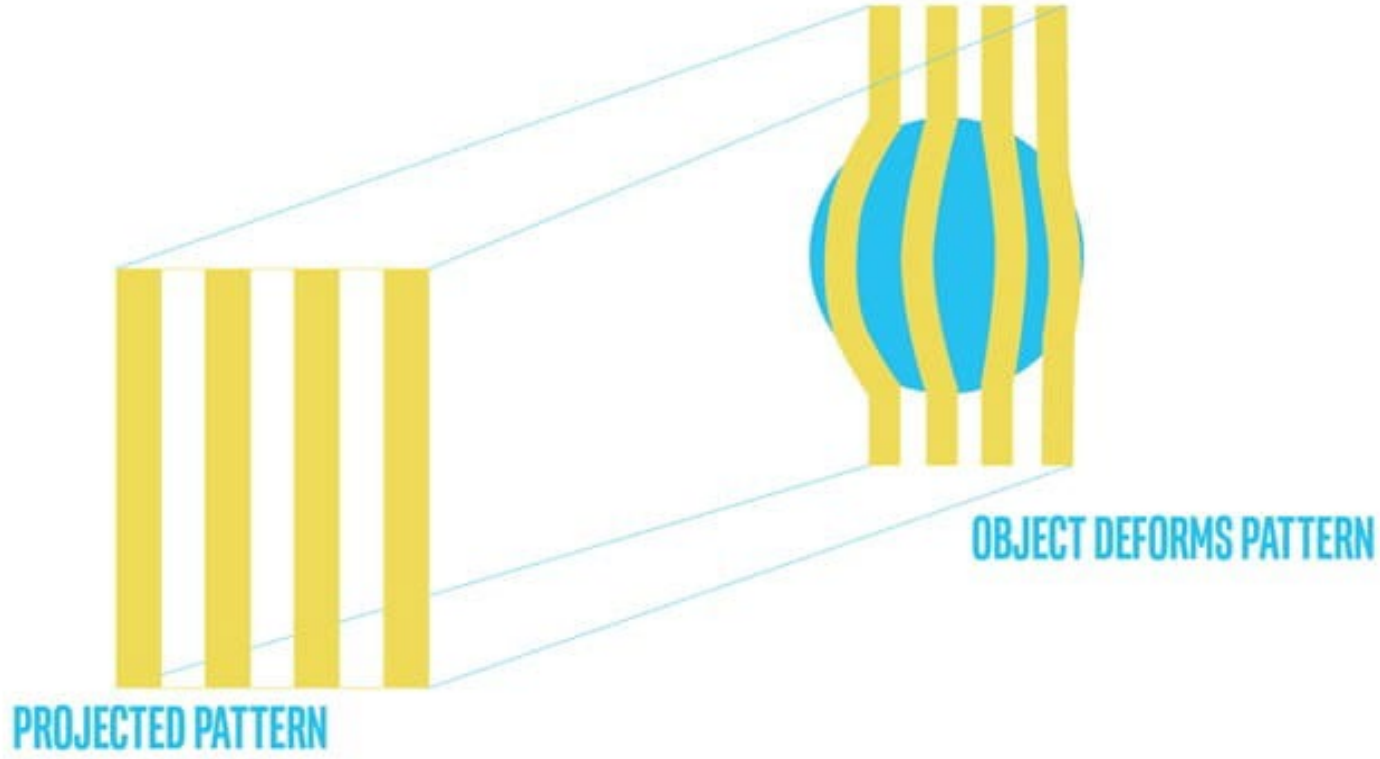
Slide credits to D. Panozzo

Structured light

Active sensors have an emitter that **projects a specific and known pattern of light** and shadows over the 3D scene, like a series of vertical stripes.

The way the pattern deforms depends on the 3D structure of the scene

Comparing the projected and the observed pattern, it is possible to compute the distance of every point of the ball from the camera.



Structured light

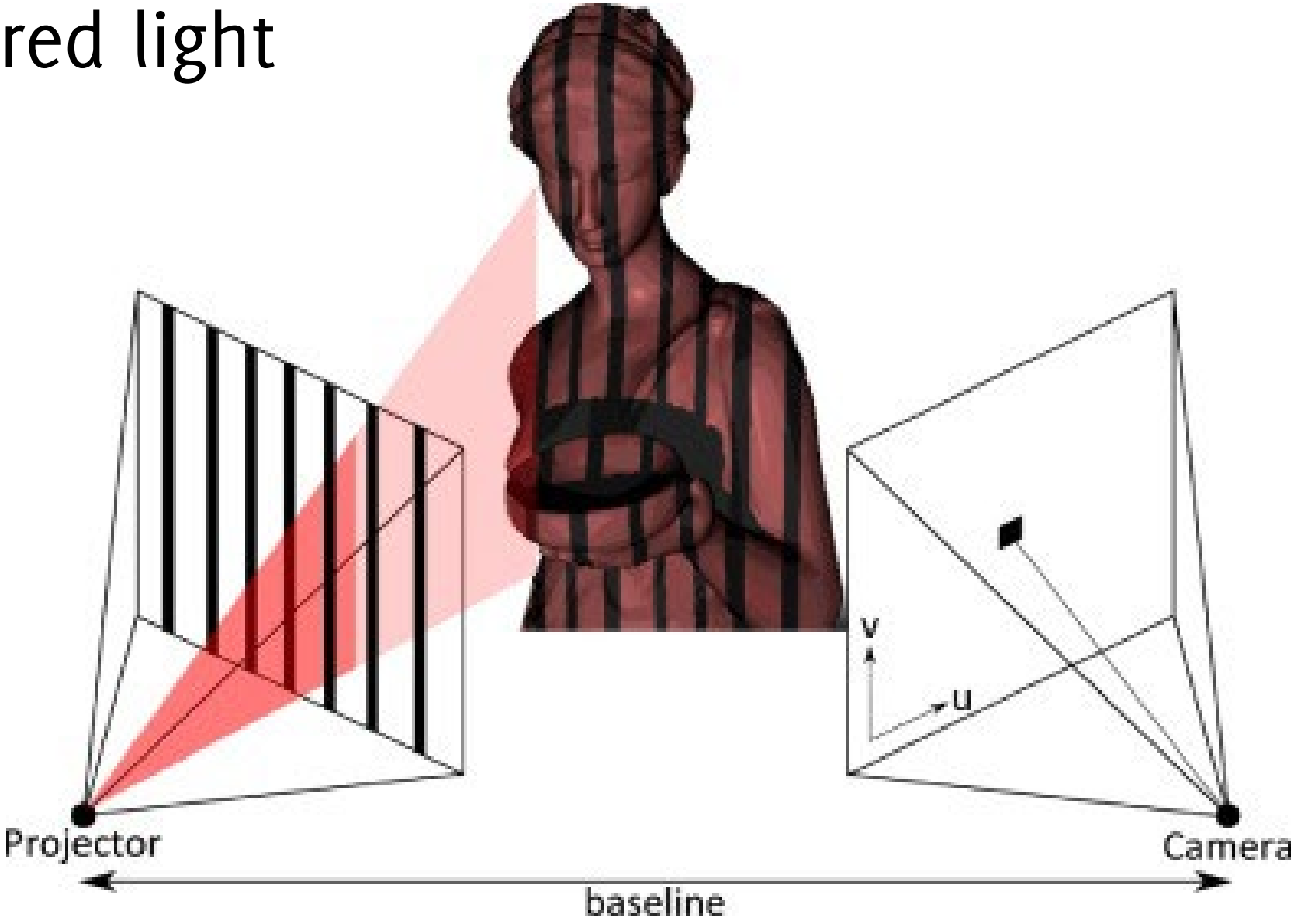
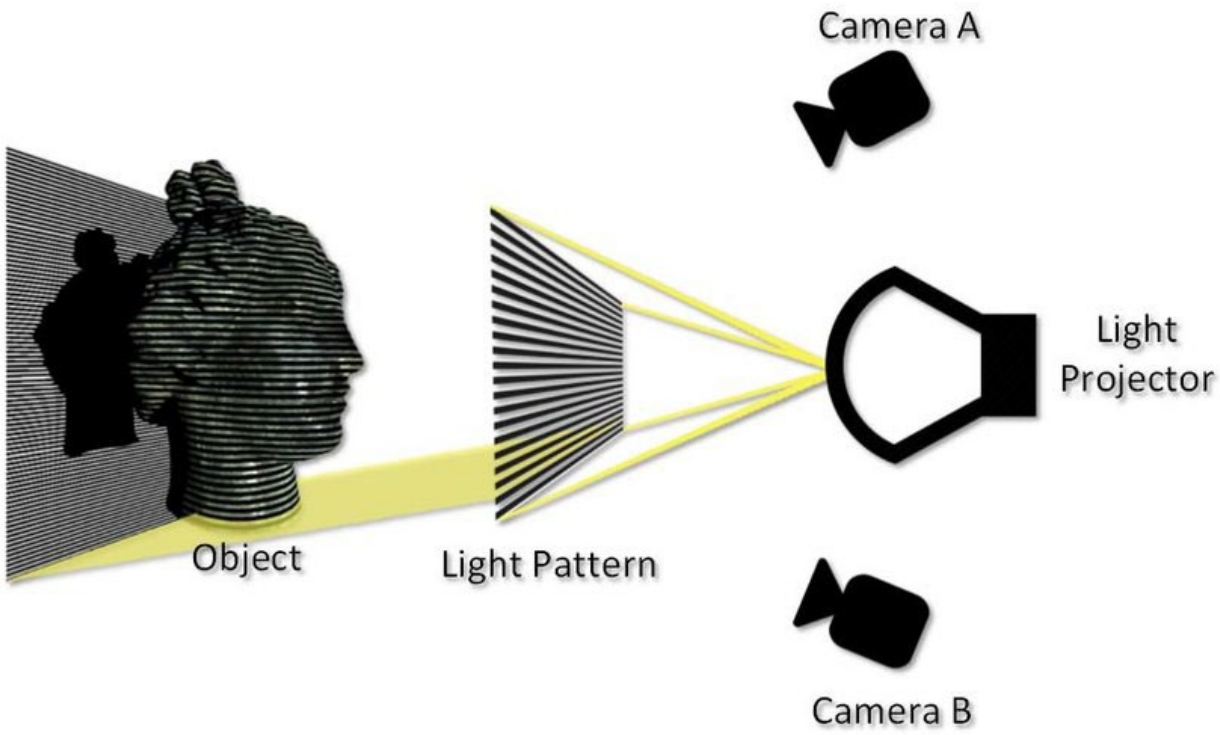


Image from: H. Sarbolandi et al.
«Kinect range sensing: Structured-light versus Time-of-Flight Kinect»

Coded Light

- Two cameras and an emitter (projector) that are accurately synchronized
- Emitter sends a large number of black and white patterns at high frequency
- Each point in the 3D scene is illuminated by a black and white sequence that works as a signature (or as a descriptor) of each 3D point.
- This signature enables creating correspondences between pixels in the two images very easily and accurately.
- Emitter can be non-visible light (e.g. infrared in Kinect)



3D Imaging Datasets

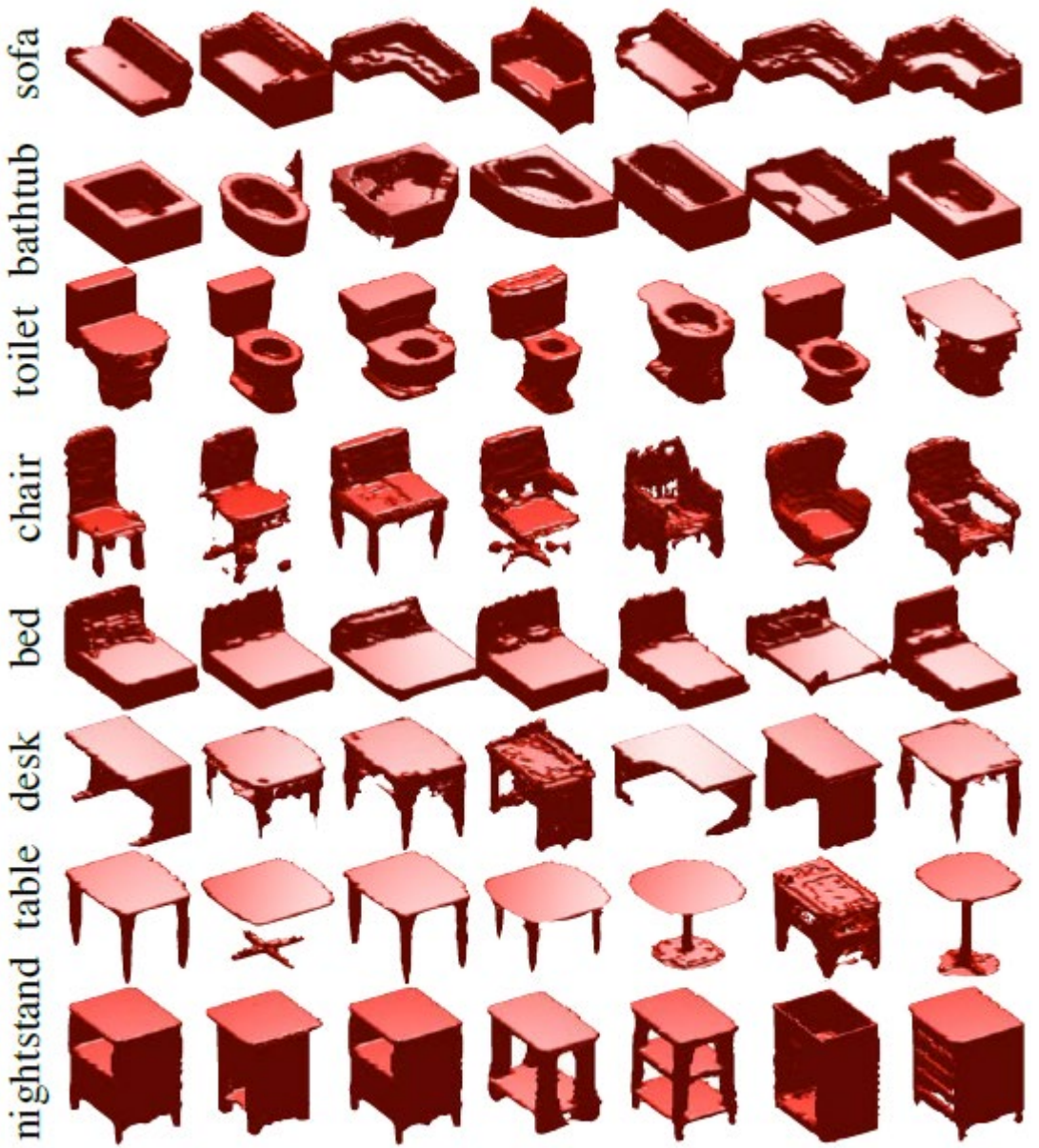
ModelNet

- The first large-scale 3D CAD model dataset.
- Voxels or Point Clouds are obtained by sampling or discretizing from these CAD models.
- Contains 151,128 3D CAD models belonging to 660 unique object categories
- Typically used in 10 or 40 class variants

Wu et al, “3D ShapeNets: A Deep Representation for Volumetric Shapes”, CVPR 2015

ModelNet

- The first large-scale 3D CAD model dataset.
- Voxels or Point Clouds are obtained by sampling or discretizing from these CAD models.
- Contains 151,128 3D CAD models belonging to 660 unique object categories
- Typically used in 10 or 40 class variants



Wu et al, "3D ShapeNets: A Deep Representation for Volumetric Shapes", CVPR 2015

ShapeNet

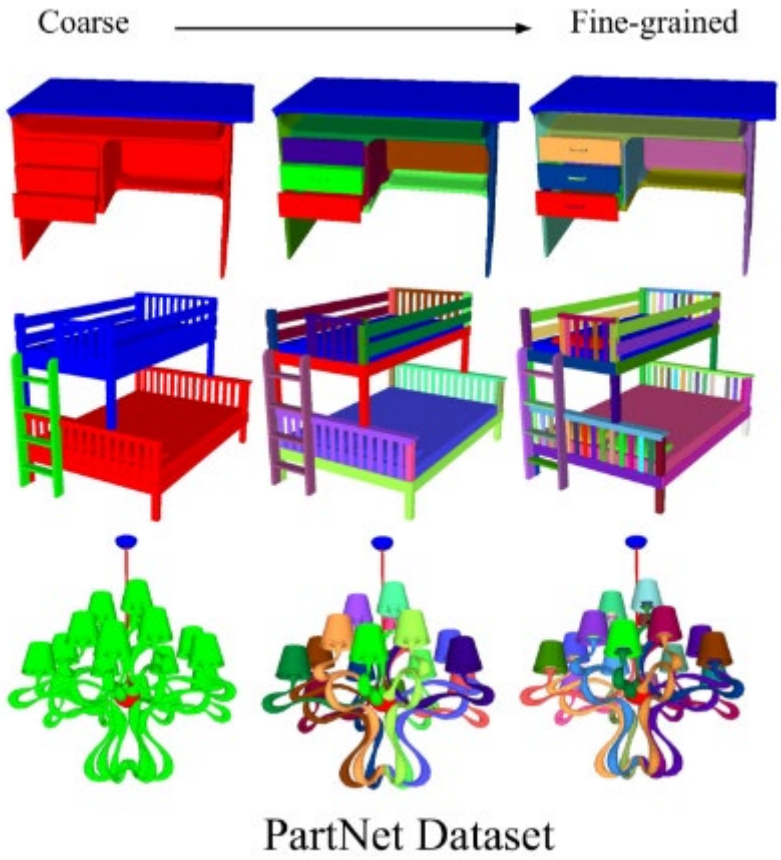
ShapeNet is an ongoing effort to establish a richly-annotated, large-scale dataset of 3D shapes.

- 55 categories,
- 53.100 3D CAD models

Standard split has 13 categories, and ~44k models. Images are also provided (25 render pre model ≈ 1M images)

This dataset absorbs MobileNet, and deals with a multitude of semantic categories and organizes them under the WordNet taxonomy

Includes a ShapeNetPart version, where parts are carefully annotated



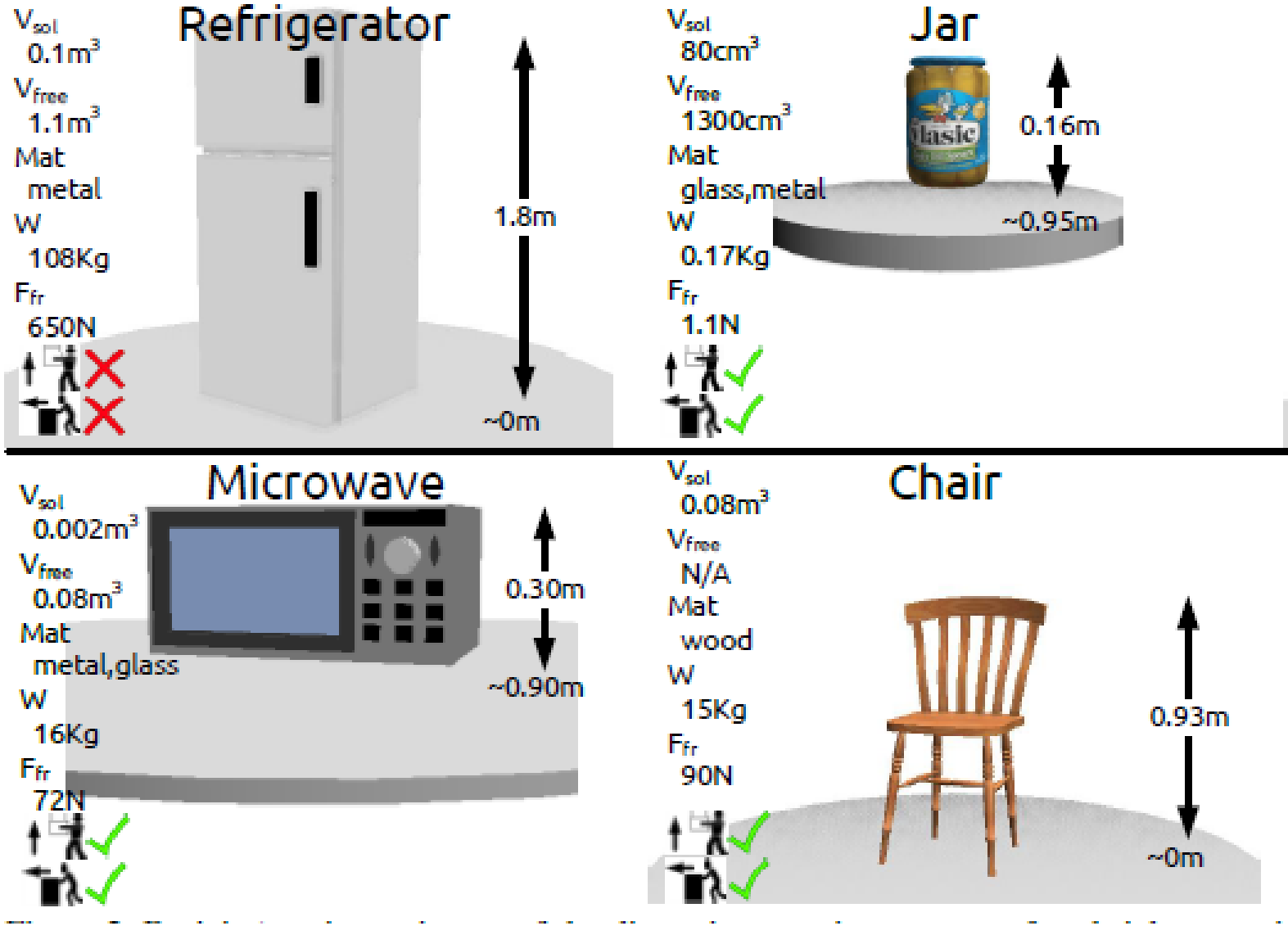
PartNet Dataset

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., ... & Yu, F. (2015). Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012.

ShapeNetSem

12,000 models spread over a broader set of 270 categories.

Models are annotated with real-world dimensions, estimates of their material composition at the category level, and estimates of their total volume and weight.



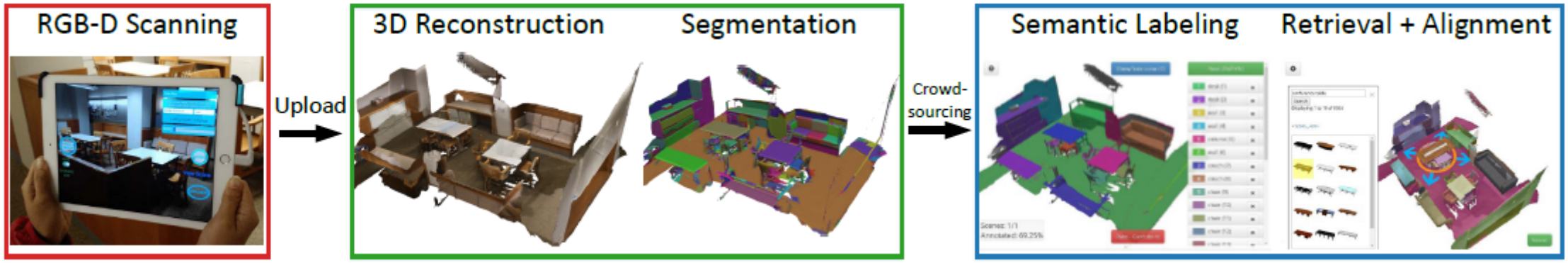
Savva, M., Chang, A. X., & Hanrahan, P. (2015). Semantically-enriched 3D models for common-sense knowledge. CVPR 2015 Workshops

ScanNet

This dataset contains scans from 1513 scenes with 3D camera poses, surface reconstructions, and semantic segmentations

Data are acquired from real world scene by RGB-D cameras, reconstructed in 3D on the cloud and then annotated manually.

3D object classification, semantic voxel labeling, and CAD model retrieval

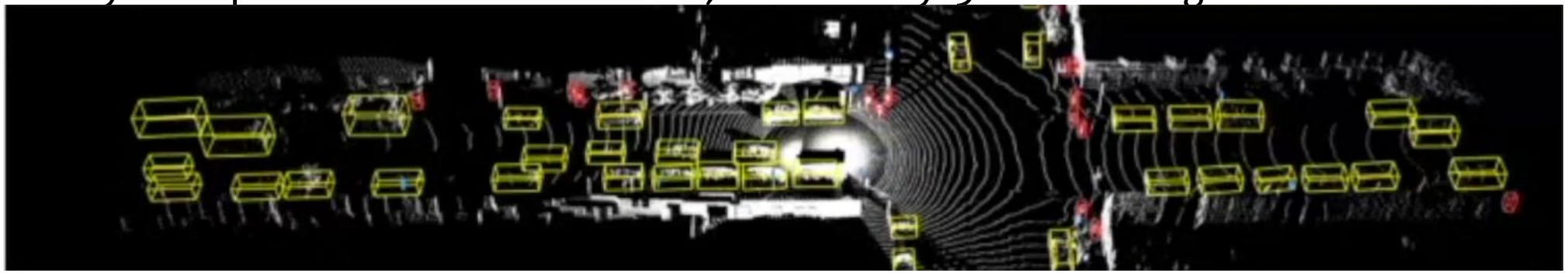


Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., & Nießner, M. Scannet: Richly-annotated 3d reconstructions of indoor scenes CVPR 2017

Datasets for OutDoor Scenes

Kitti 360, Lidar, labeled by 3D Bounding Boxes and semantic segmentation

Waymo Open Dataset: LIDAR data, labeled by 3D bounding boxes



https://www.cvlibs.net/datasets/kitti/

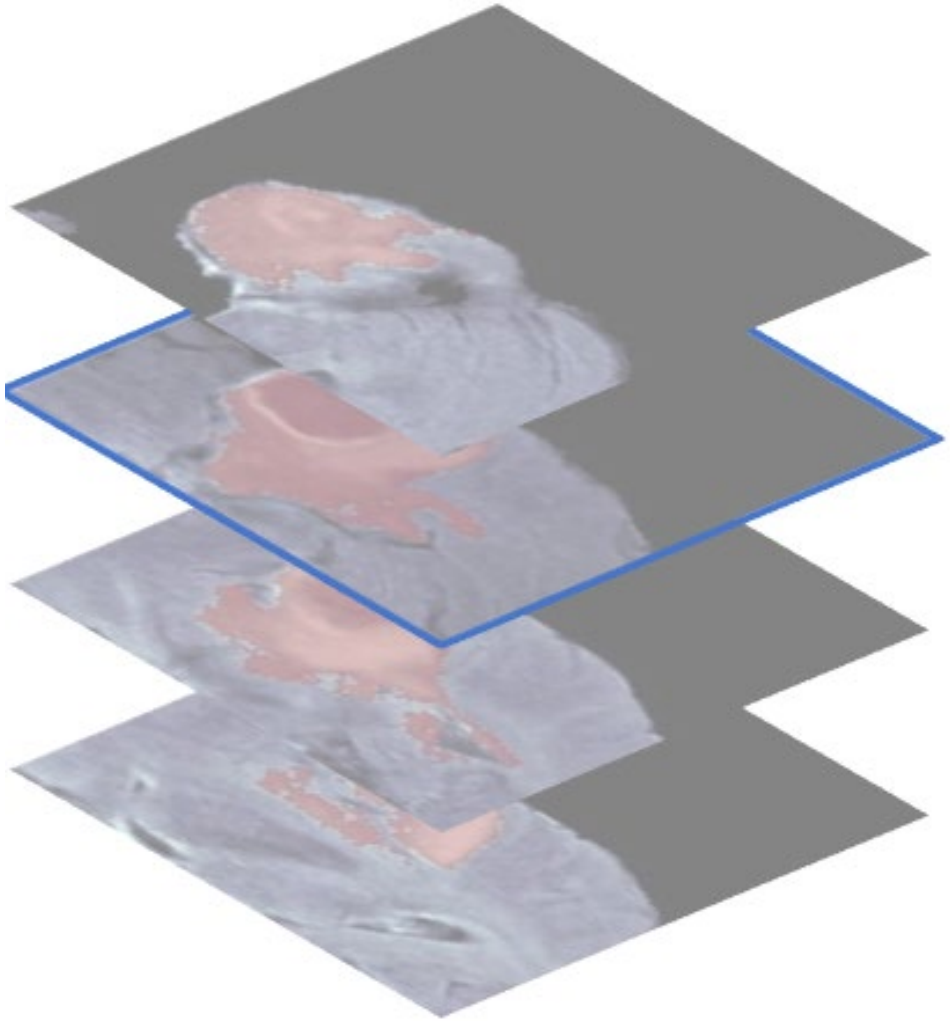
BradTS

BraTS focuses on the segmentation of brain tumors in multimodal magnetic resonance imaging (MRI) scans

There are 369 CT scans with

- spatial resolution $240 \times 240 \times 155$
- 4 channels corresponding to different modalities

There are segmentation labels over 4 classes corresponding to different stages of the tumor

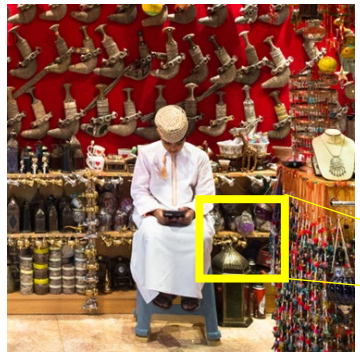


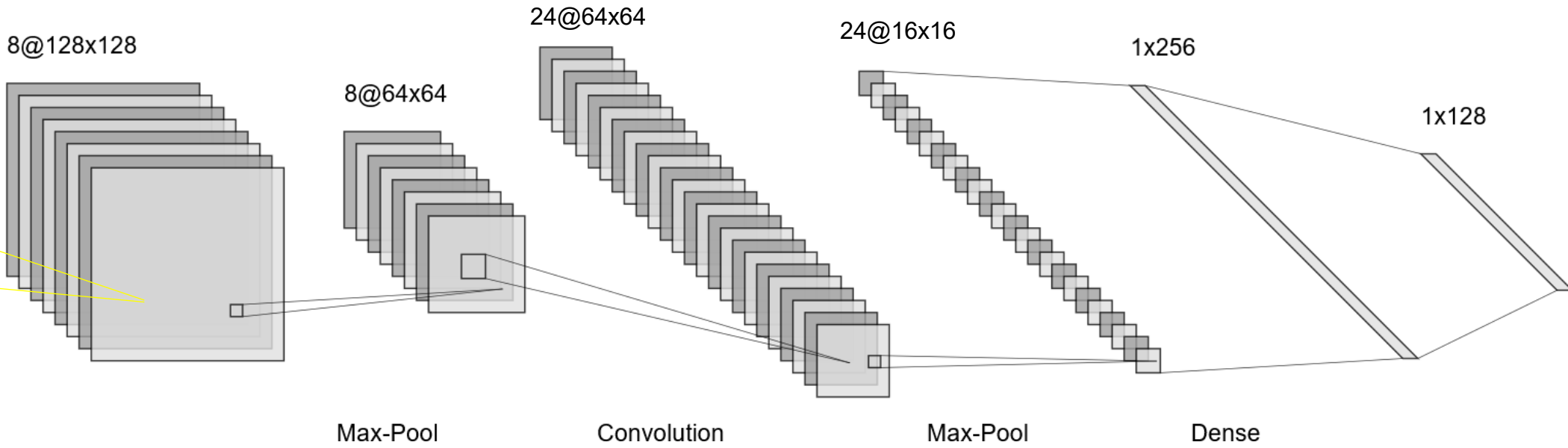
B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, et al. "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)", IEEE Transactions on Medical Imaging 34(10), 1993-2024 (2015)

Part2: Deep Learning on Voxelized 3D Data

The typical architecture of a (2D) CNN

The input of a CNN is an entire image

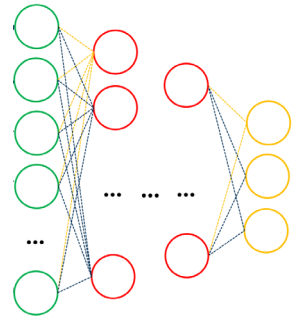




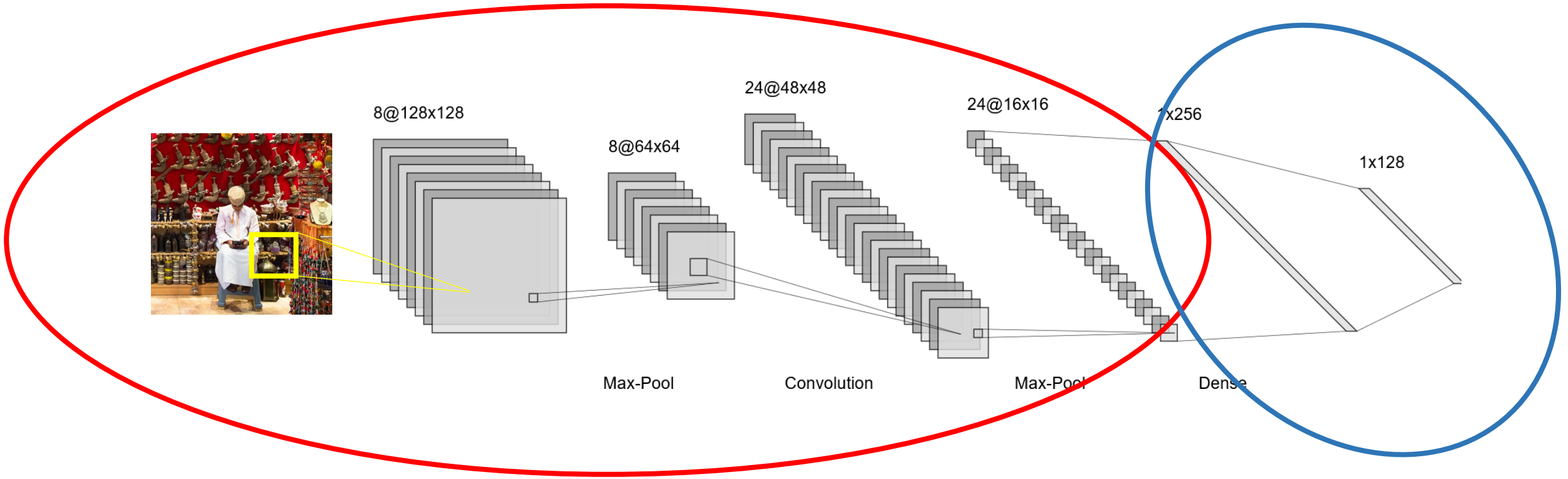
The image gets convolved against many filters

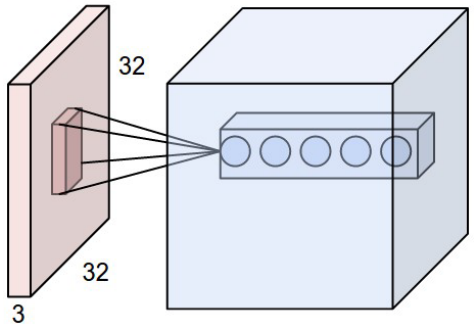
When progressing along the network, the «number of images» or the «number of channels in the images» increases, while the image size decreases

Once the image gets to a vector, this is fed to a traditional neural network

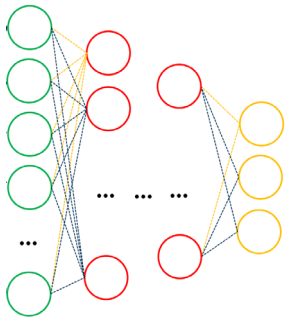


“Locality-Aware” vs “Dense” Processing





Locality-Aware: this is made of convolutional layers, spatial dimension matters.
Give rise to 3D activations



Dense: this is made of MLP layers
This is meant to process 1D vector

By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Locality Aware Part: Convolutional Layers

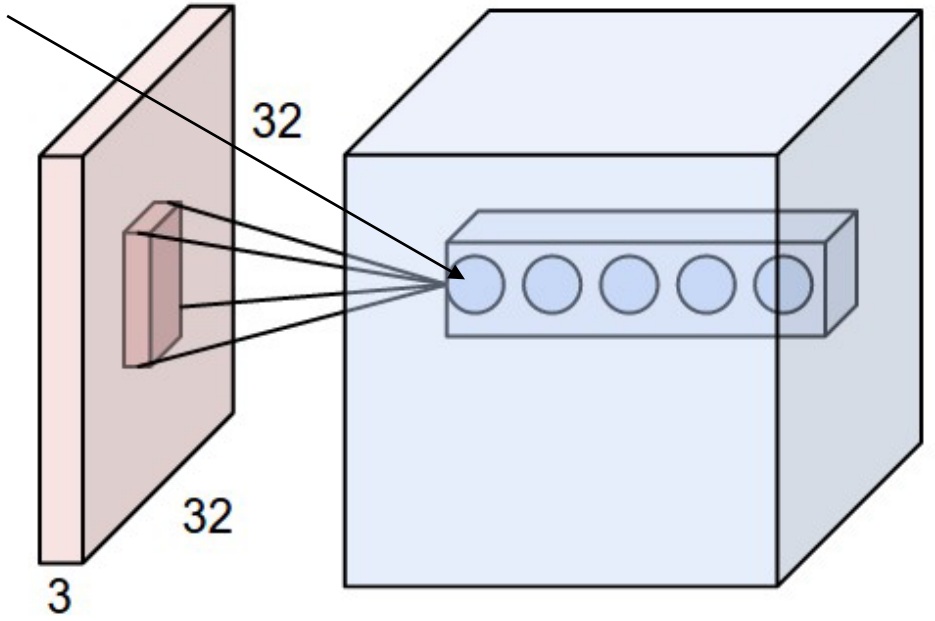
Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input, considering all the channels**

$$a(r,c,1) = \sum_{(u,v)\in U,k} w^1(u,v,k)\, x(r+u, c+v, k) + b^1$$

The parameters of this layer are **called filters.**

The same filter is used through the whole spatial extent of the input



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input, considering all the channels**

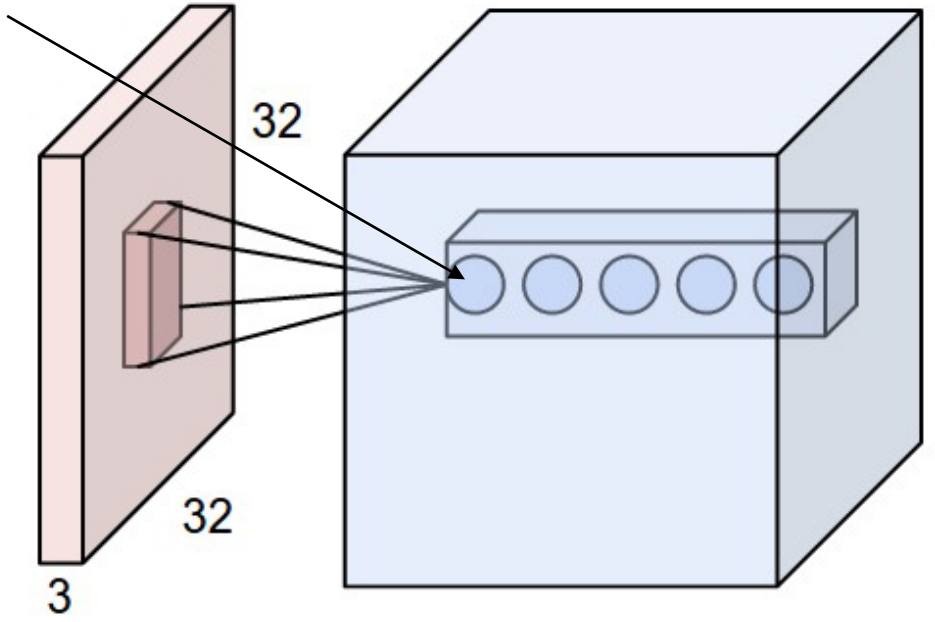
$$a(r, c, 1) = \sum_{(u,v)\in U,k} w^1(u, v, k)\, x(r + u, c + v, k) + b^1$$

Along the **spatial dimension it's a convolution:**

- **local processing:** filters spans a **small neigborhood** U
This is equivalent to **sparse connectivity**

- U needs to be specified, it is a very important attribute of the filter

- It operates in the same way along all the channels
weight sharing.



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

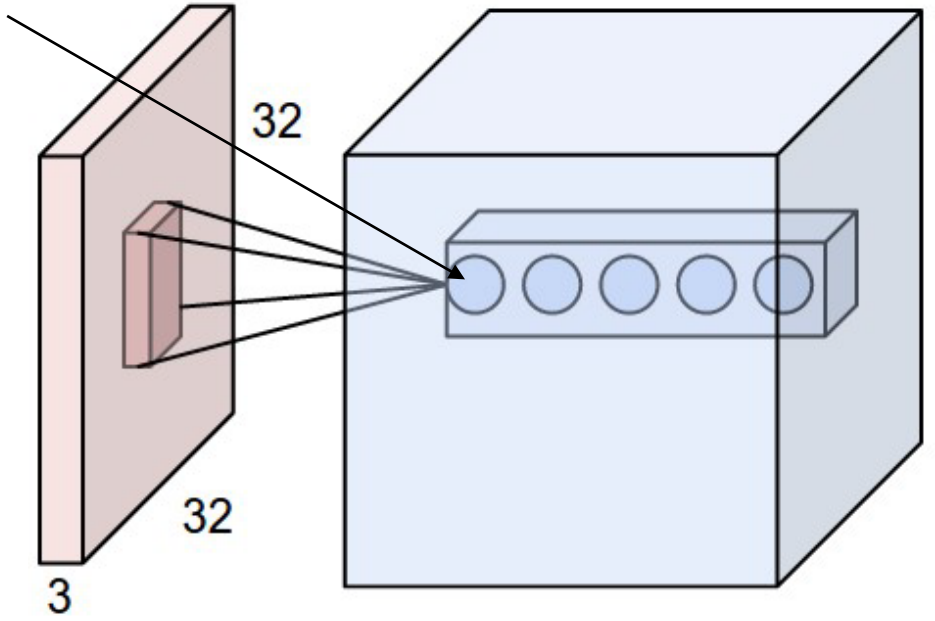
Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input, considering all the channels**

$$a(r,c,1) = \sum_{(u,v)\in U,k} w^1(u,v,k)\, x(r+u, c+v, k) + b^1$$

The **channel dimension**:

- spans the **entire input depth** (no local processing)
- there is no need to specify that in the filter attributes

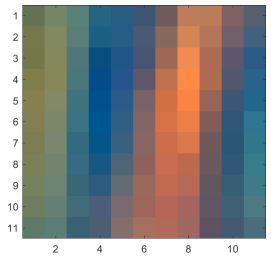


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

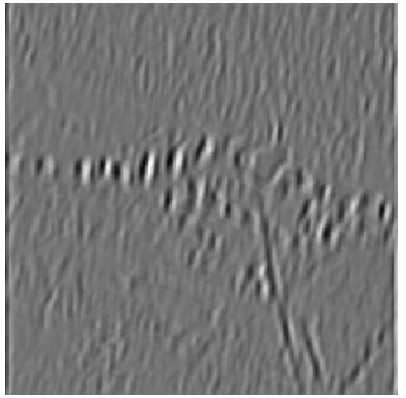
I

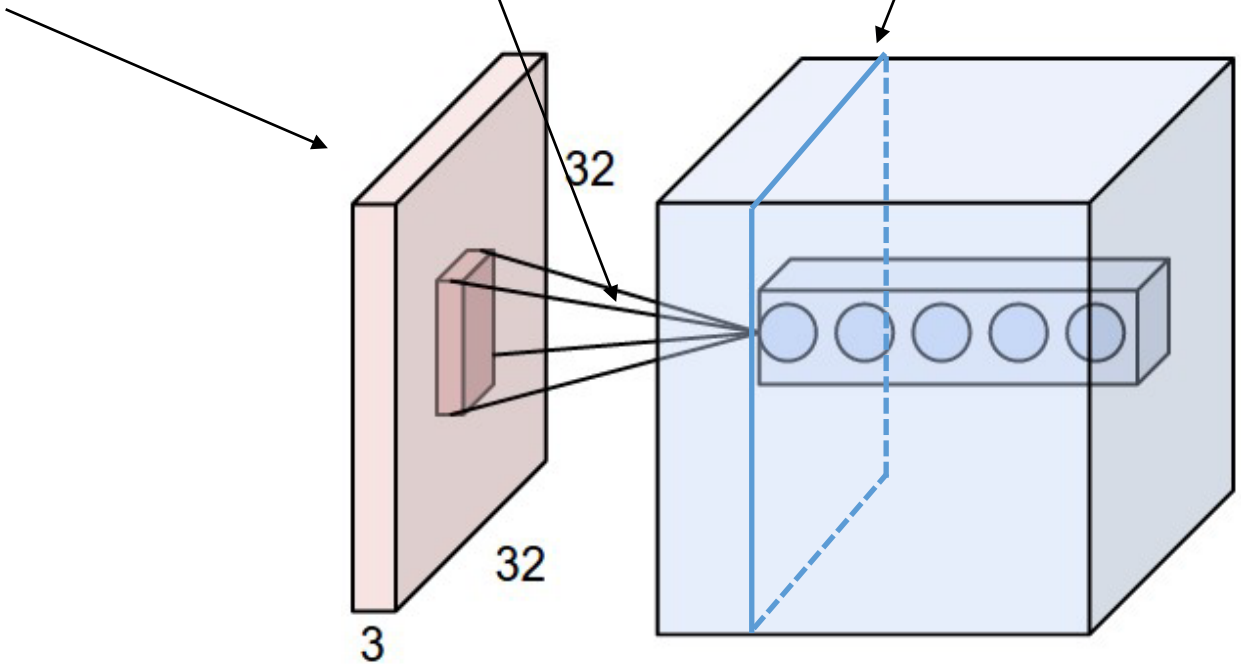
w^1



b^1

$a(:,:,1)$





By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

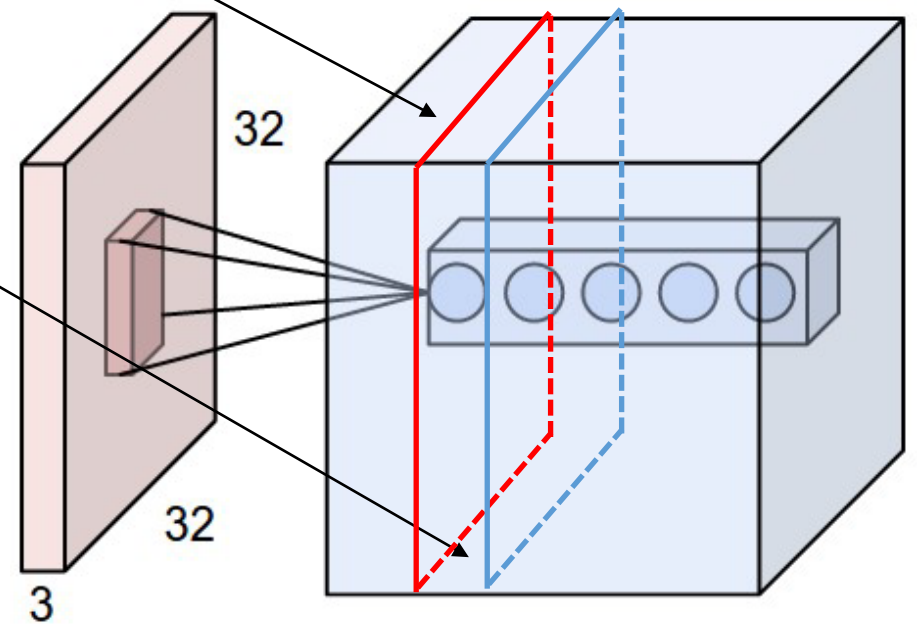
Convolutional Layers

Different filters yield different layers in the output

$$a(r,c,1) = \sum_{(u,v)\in U,k} w^1(u,v,k)\, \boldsymbol{x}(r+u, c+v, k) + b^1$$

$$a(r,c,2) = \sum_{(u,v)\in U,k} w^2(u,v,k)\, \boldsymbol{x}(r+u, c+v, k) + b^2$$

Different filters of the same layer have the same spatial extent

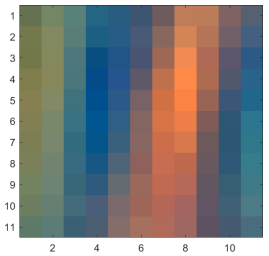


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

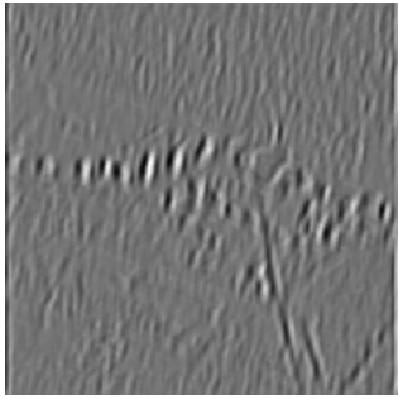
Convolutional Layers

I

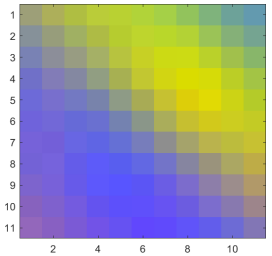
w^1, b^1



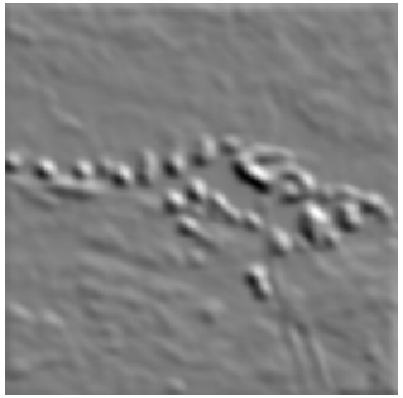
$a(:,:,1)$

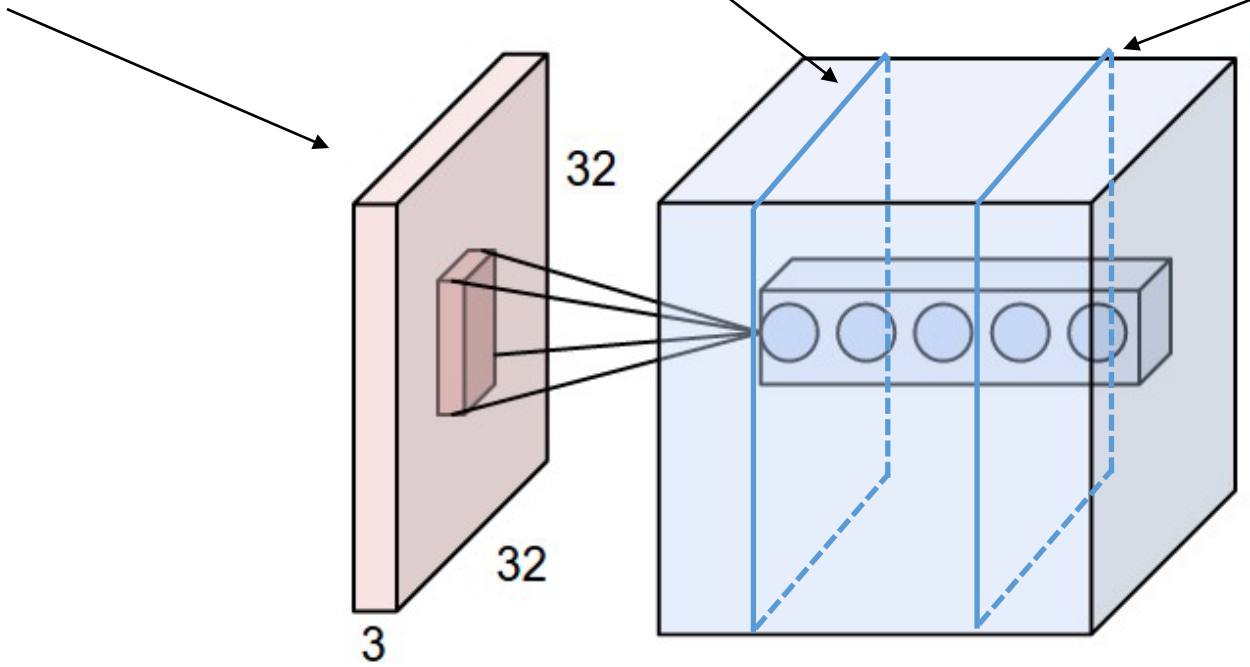


w^n, b^n



$a(:,:,n)$



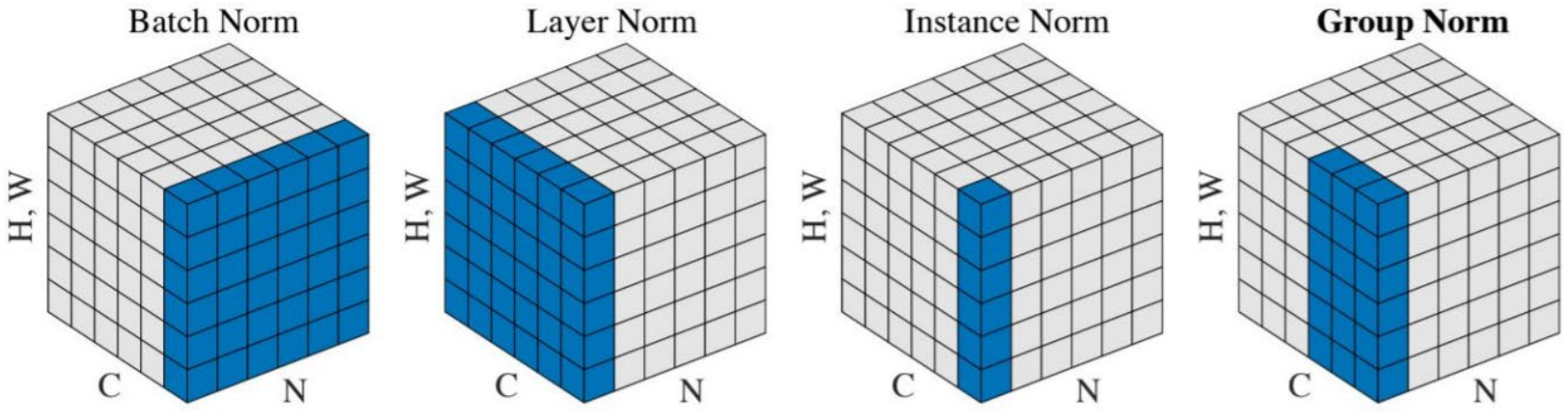


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Pooling Layers in 3D data

Maxpooling: simply affect only the spatial dimension, replacing a 3D subvolume by a number

BatchNorm: treats all the spatial dimensions the same, averaging over the same channel on all the dimension



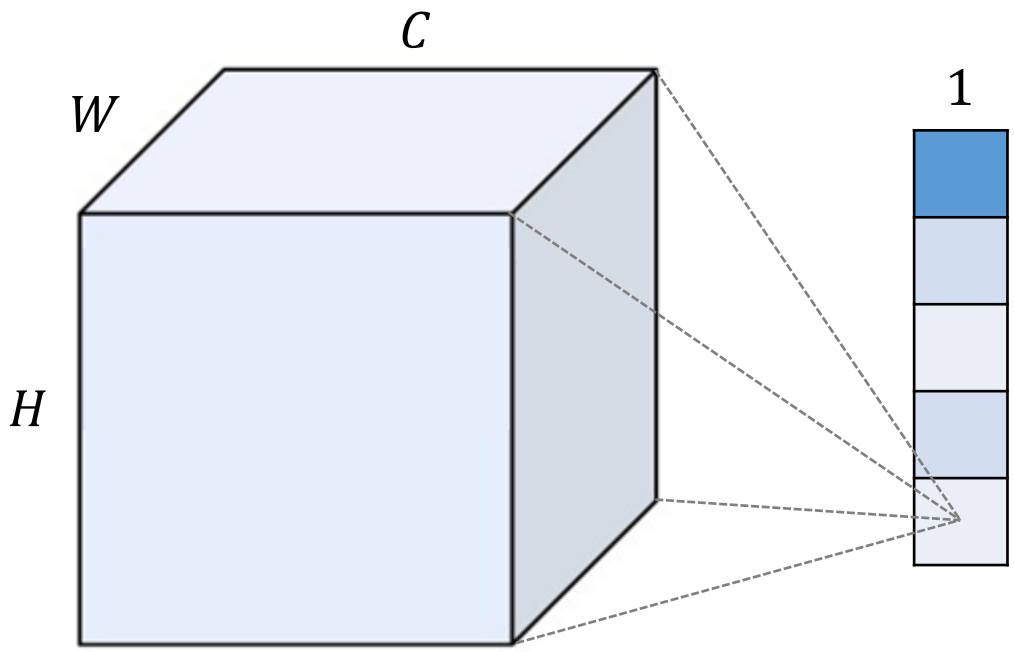
Wu and He, “Group Normalization”, ECCV 2018

Flattening or Global Pooling Layers

Flattening: simply unfold the 3D volume of activations in a vector

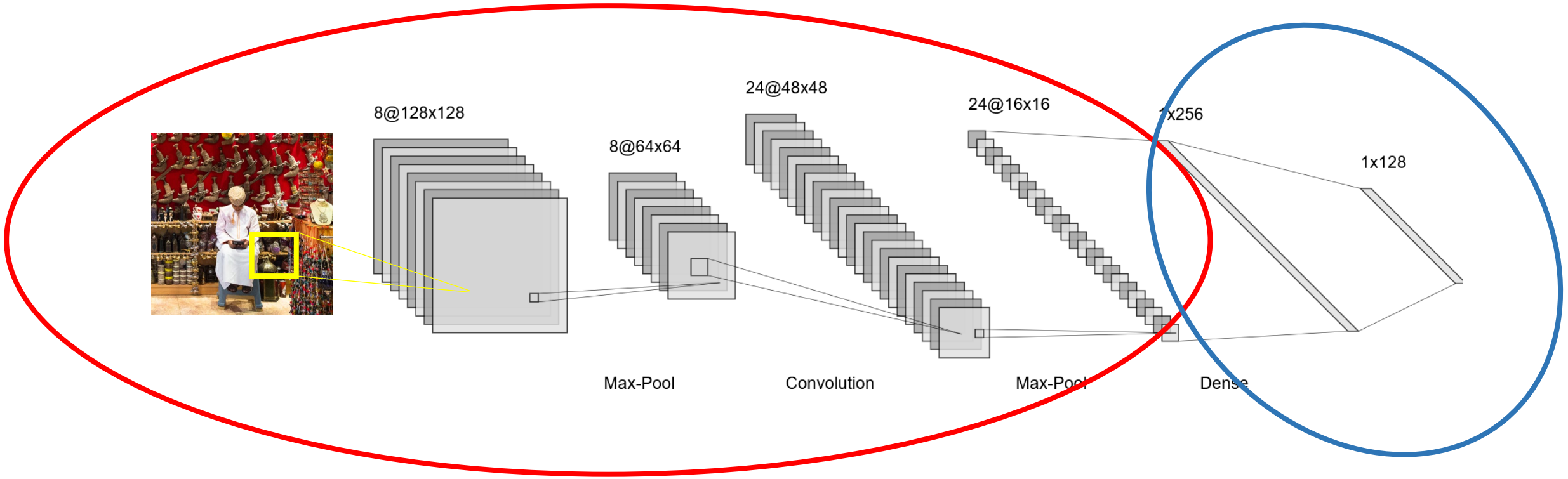
$$A \rightarrow A(:)$$

Global Pooling Layer: Perform a global operation on each channel, **along the spatial components.** Out of each channel, keep a single value. Pooling operations can be the average (GAP), or the maximum (GMP)



$$F_k = \frac{1}{H \cdot W} \sum_{(x,y)} f(x, y, k), k = 1, \dots, C$$

“Locality-Aware” vs “Dense” Processing



How to move from locality-aware to dense part in the CNN?

CNN for 3D voxel grid

Key intuitions:

- The **(three) spatial dimensions need to be handled in the same way**:
  - **Locality** / sparse connectivity,
  - **Translation invariance** / weight sharing
- The **channel dimension** preserves the same meaning
  - Features are always computed by **aggregating all the input channels**
- The GAP and **pooling** layers, **aggregate all the spatial dimensions**, keeping channels separate.

3D CNN have been used for solving classification problems in:

- Voxelized data
- Video Clips, where the temporal dimenison is consdiered as z axis.

Wu et al, "3D ShapeNets: A Deep Representation for Volumetric Shapes", CVPR 2015
Ji, S., Xu, W., Yang, M., & Yu, K. 3D convolutional neural networks for human action recognition, TPAMI 2012
Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. Learning spatiotemporal features with 3d convolutional networks. ICCV 2015
Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks, CVPR2014

3D convolution

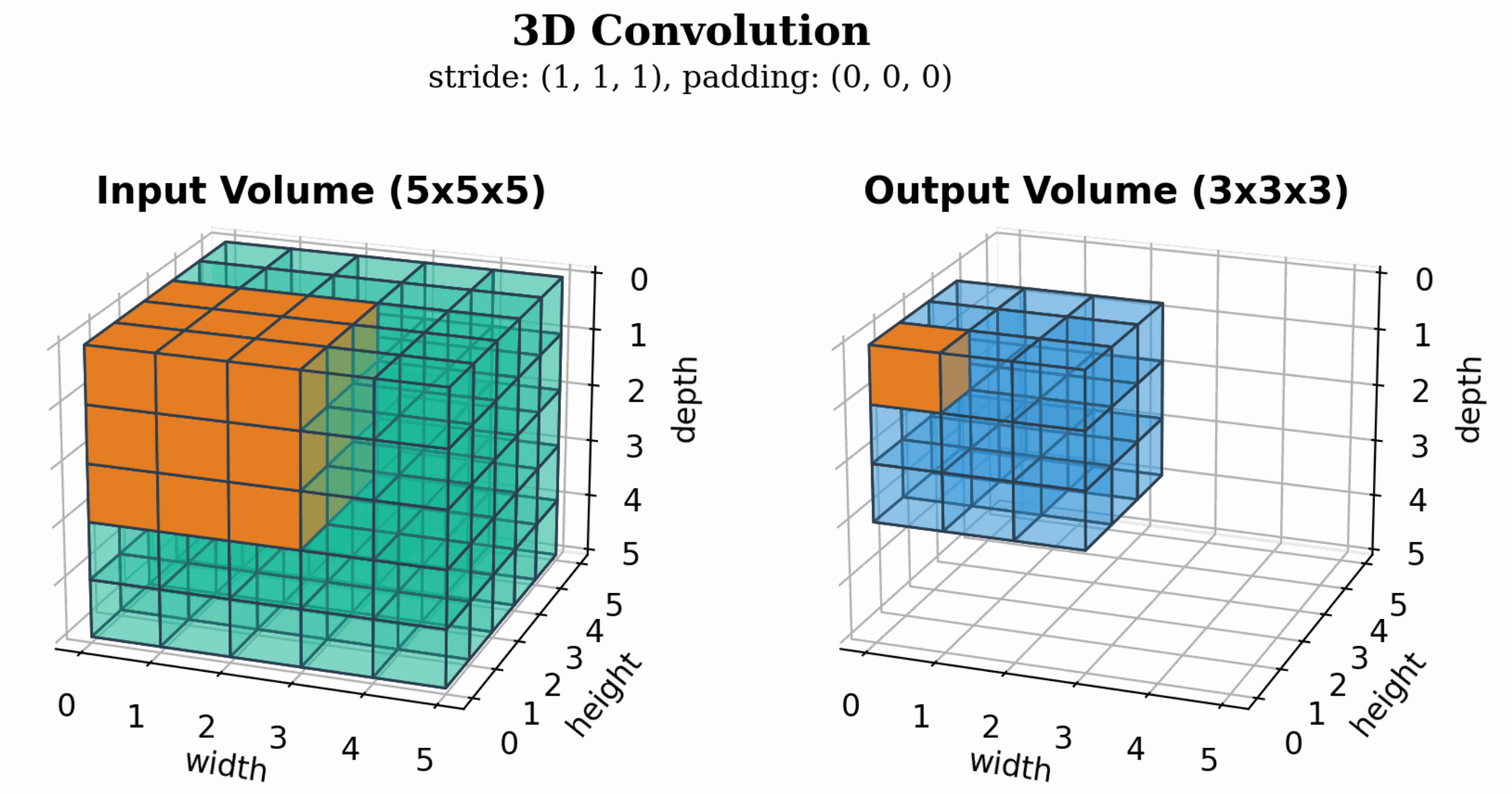


Image Credits Mariusz Wiśniewski

3D convolution

Filter Size?

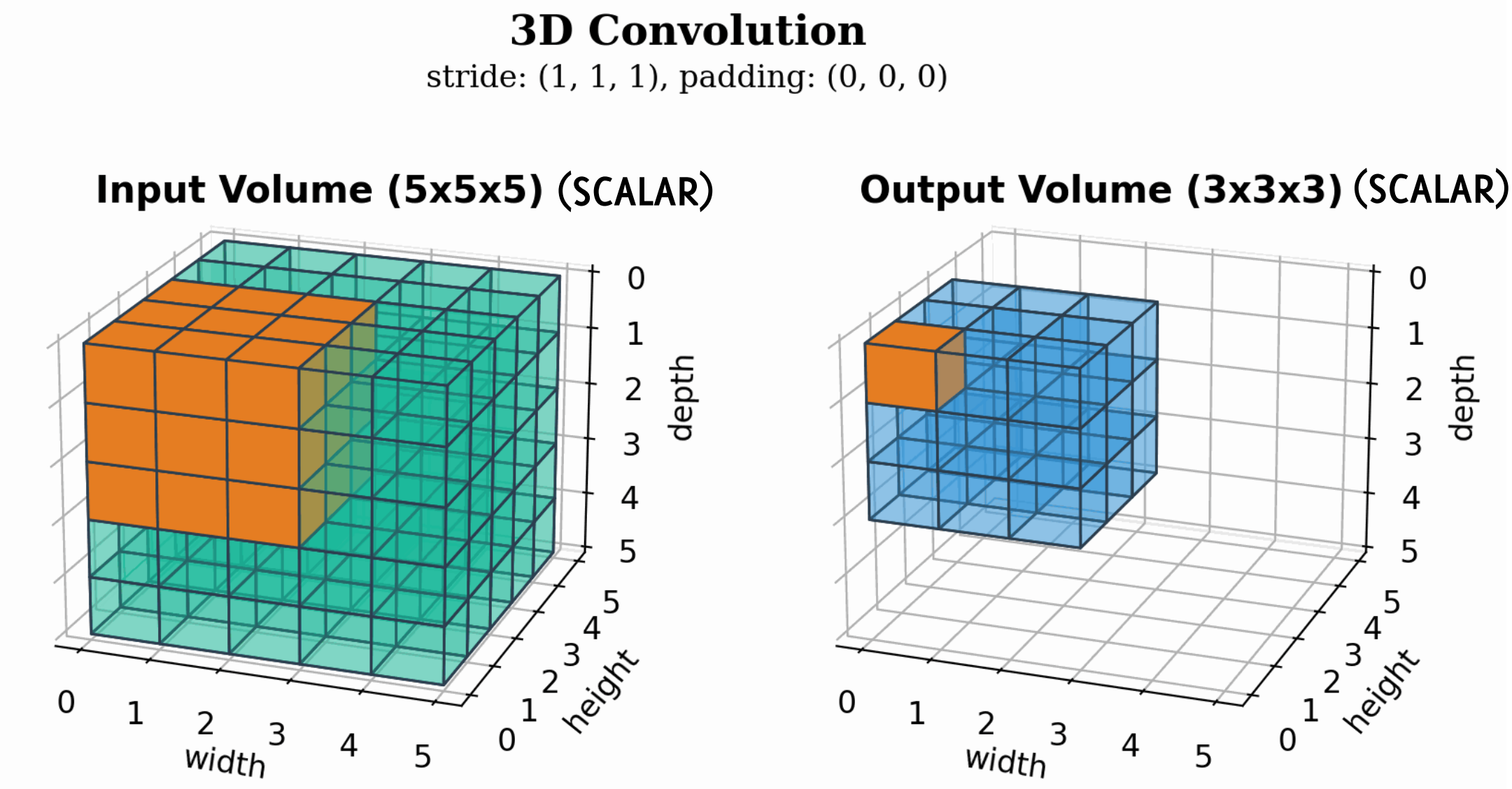


Image Credits Mariusz Wiśniewski

3D convolution

Filter Size? 3x3x3 (+1 bias)

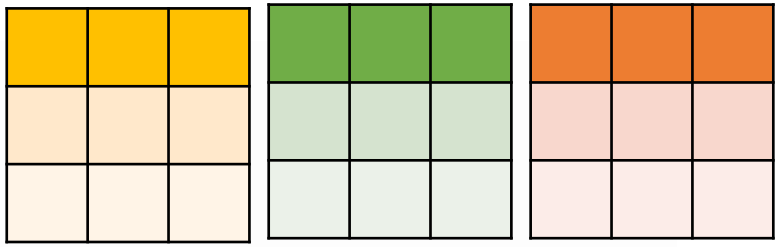
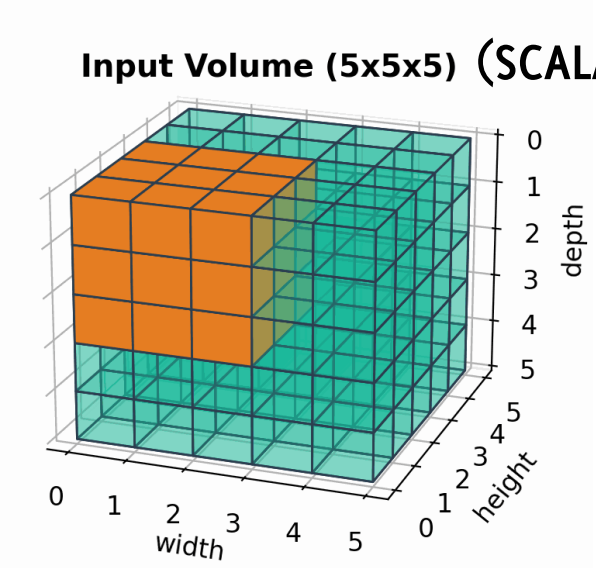


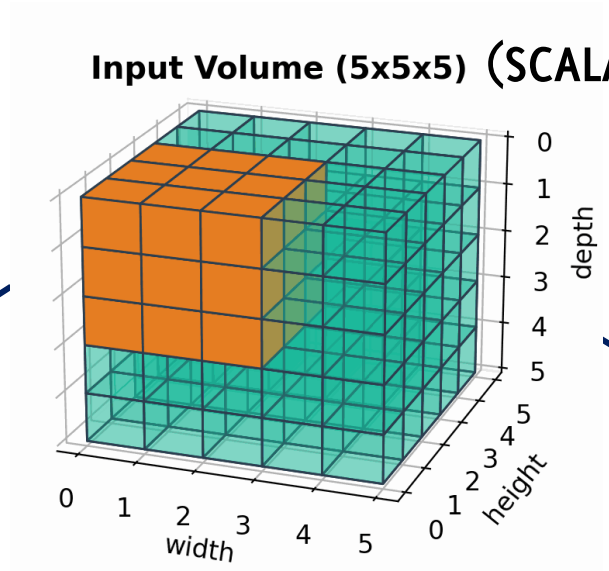
Image Credits Mariusz Wiśniewski

What with 4 filters?



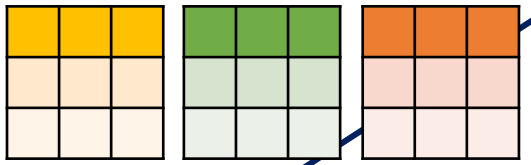
Output size?

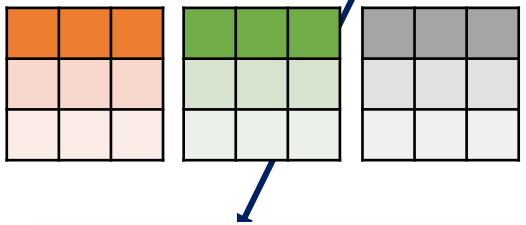
More filters..

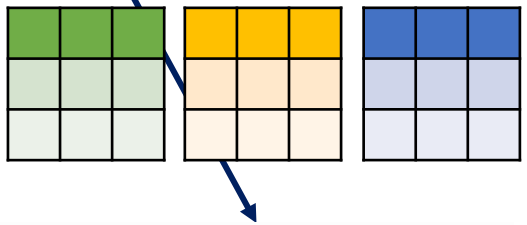


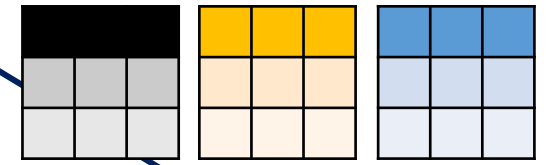
Output size?

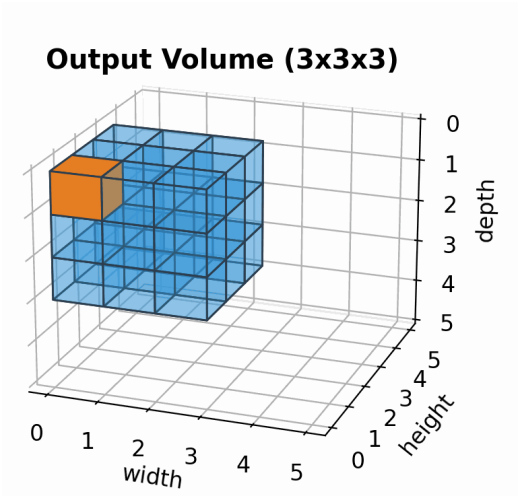
It's a tensor 3x3x3 x4



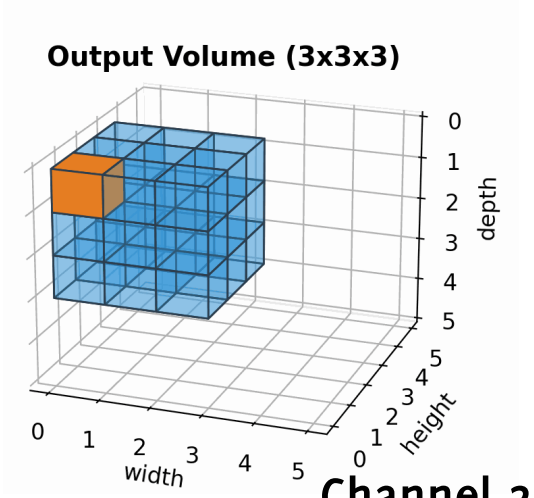




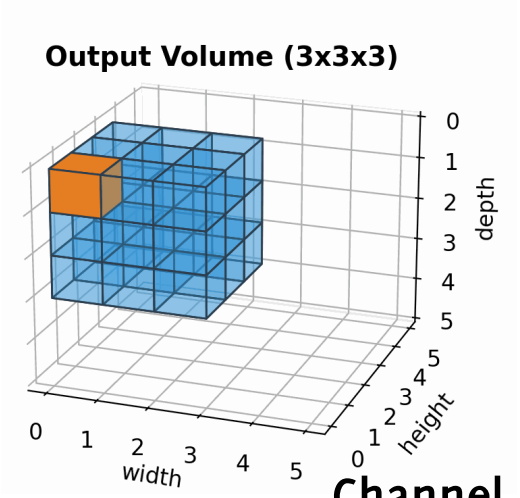




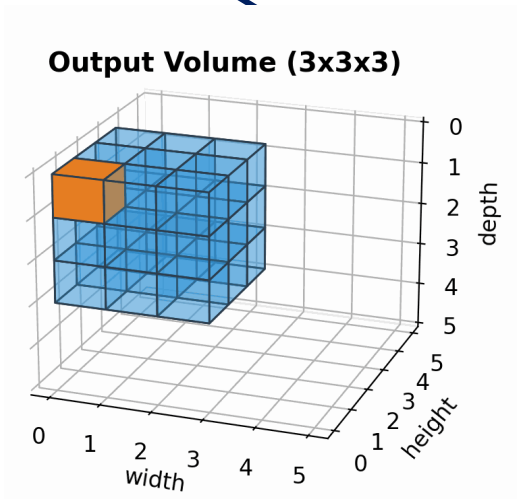
Channel 1



Channel 2



Channel 3

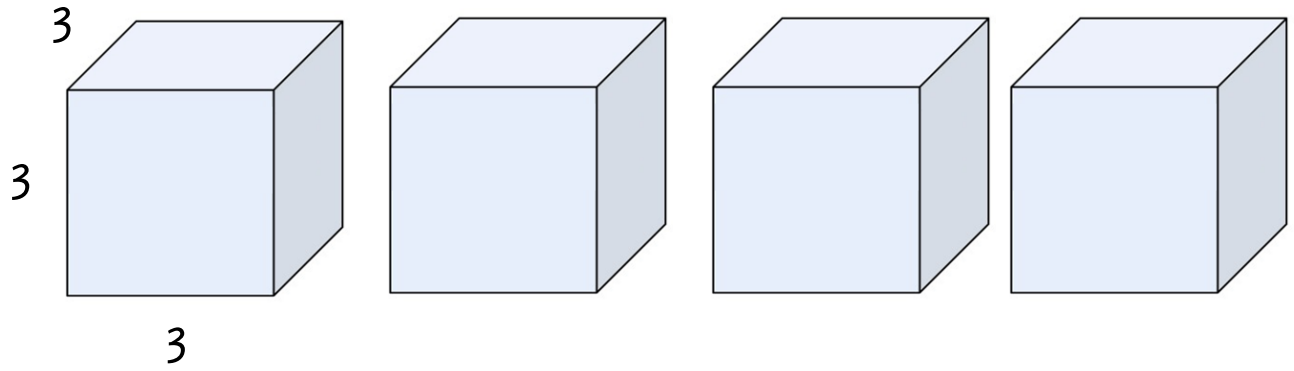


Channel 4

G. Boracchi

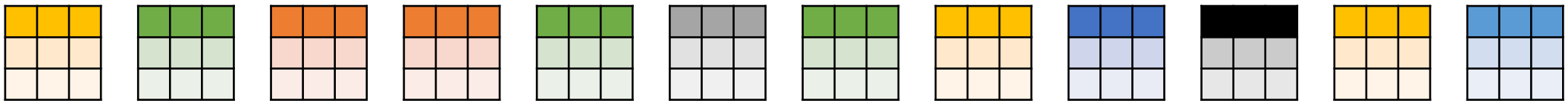
Next Layer

Input: a tensor having size $3 \times 3 \times 3 \times 4$



Add a convolutional layer with 1 Filter $3 \times 3 \times 3$ padding valid

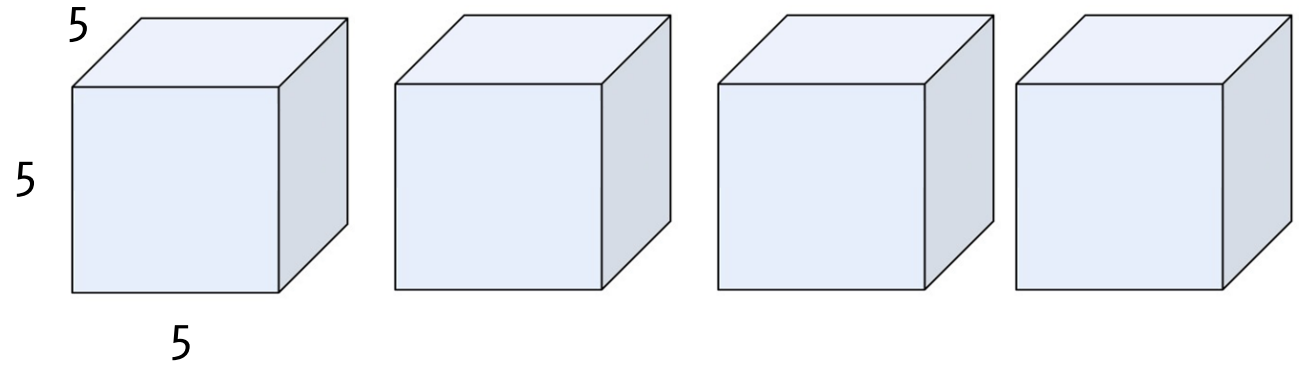
The filter will have 12 channels and mix all the input channels together



The output

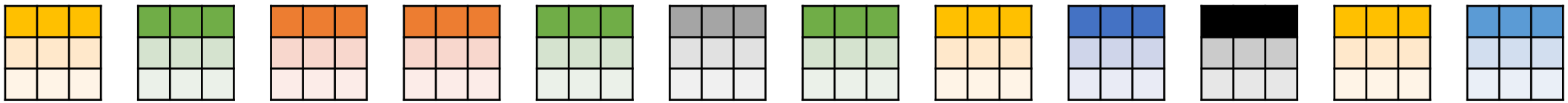
Next Layer

Input: a tensor having size $5 \times 5 \times 5 \times 4$

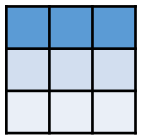


Add a convolutional layer with 1 Filter $3 \times 3 \times 3$ padding valid

The filter will have 12 channels and mix all the input channels together



The output 3x3

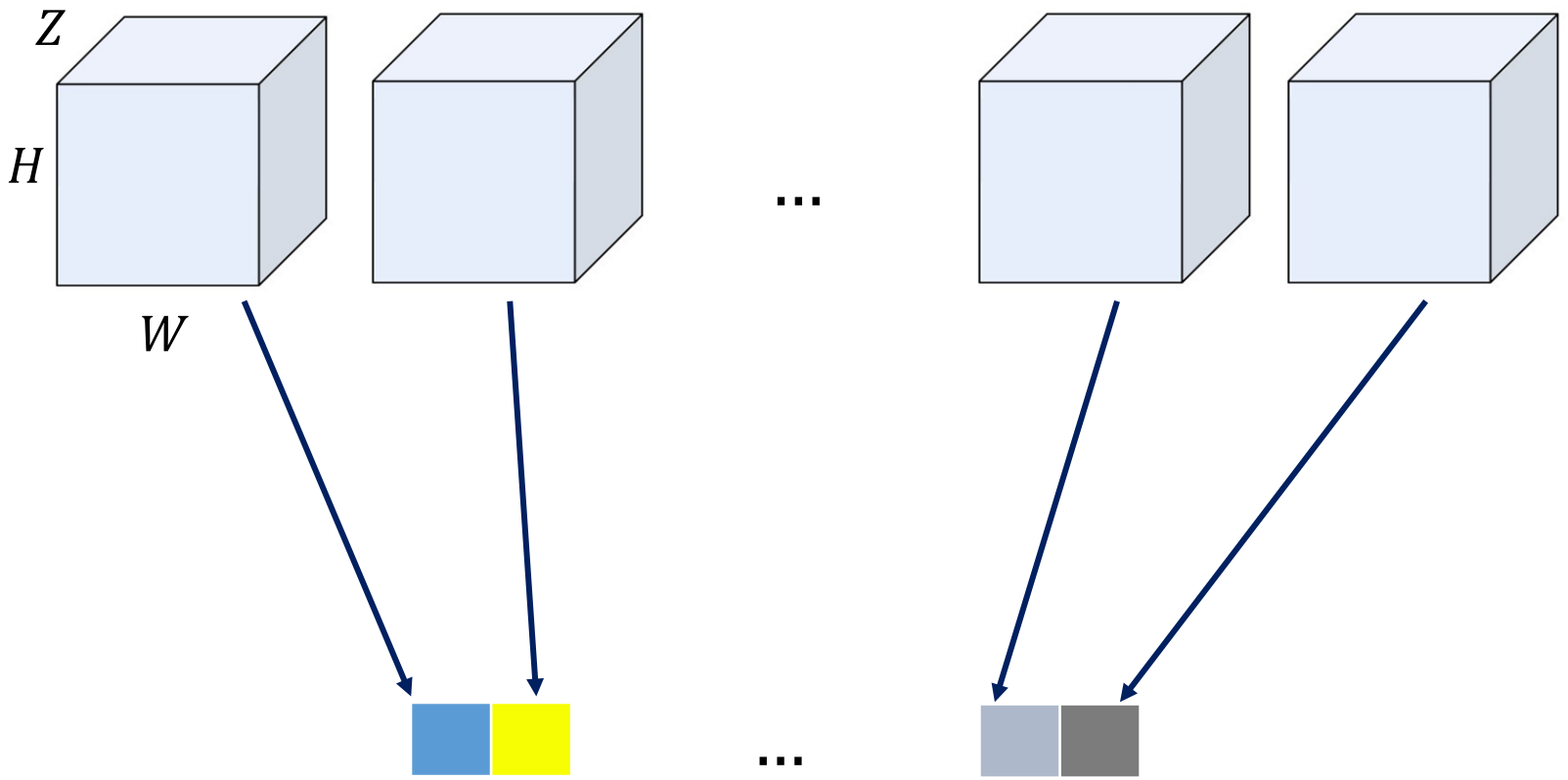


Gloval Average Pooling Layer

Assume you are getting at the last activation as 4 d volume $N \times N \times N \times C$

Gloval Average Pooling Layer

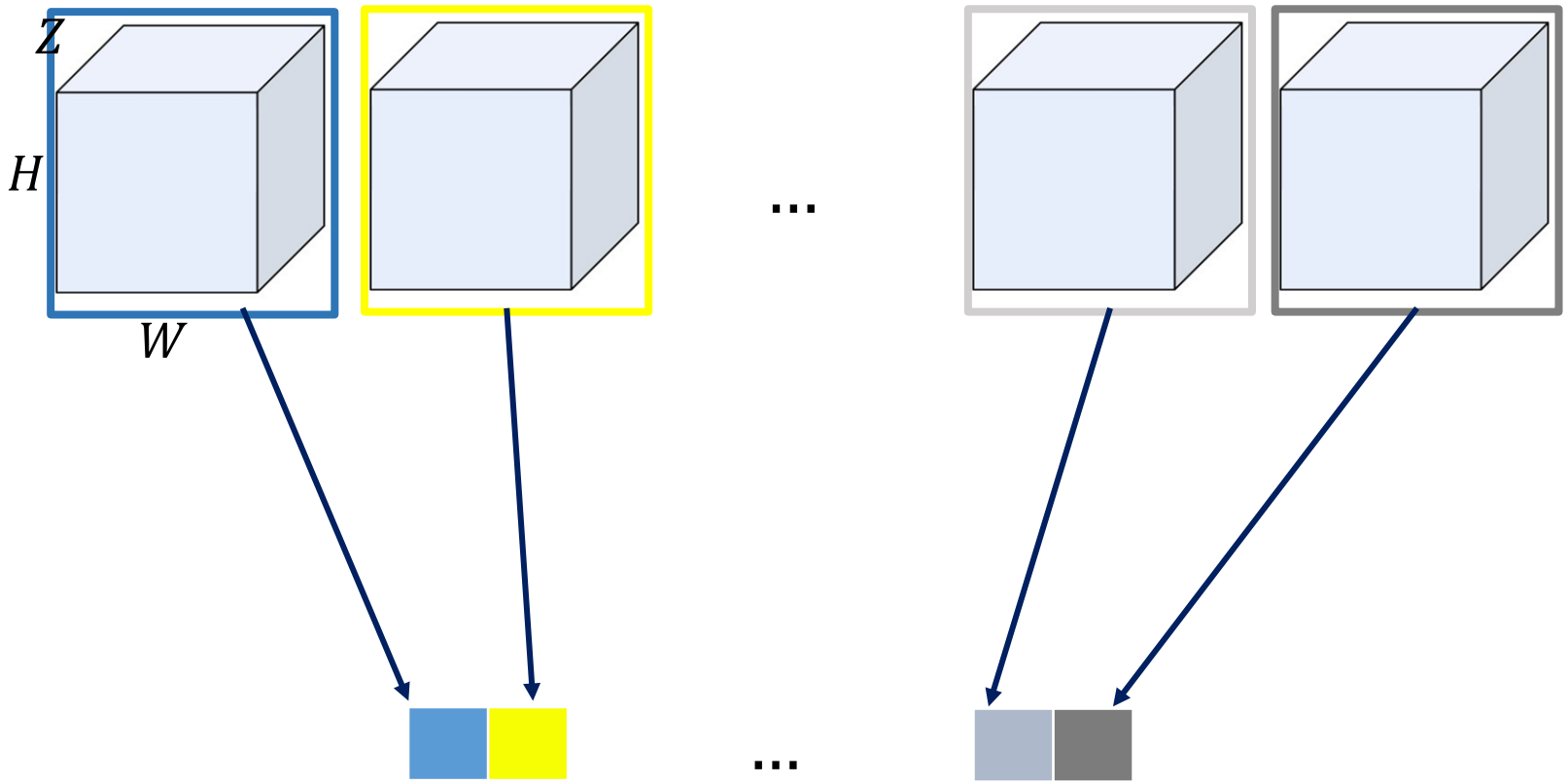
Assume you are getting at the last activation as 4 d volume $N \times N \times N \times C$



$$F_k = \frac{1}{H \cdot W \cdot Z} \sum_{(x,y,z)} f_k(x, y, z), k = 1, \ldots, C$$

Gloval Average Pooling Layer

Assume you are getting at the last activation as 4 d volume $N \times N \times N \times C$



$$F_k = \frac{1}{H \cdot W \cdot Z} \sum_{(x,y,z)} f_k(x, y, z), k = 1, \ldots, C$$

(2+1)D Convolution

In case of videos, the third dimension (time) is not equivalent to the spatial ones

Therefore, it can be convenient to treat this dimension differently

Instead of applying 3D convolution it is possible to perform in cascade

- 2D convolution along the spatial dimension
- 1D convolution along the temporal dimension

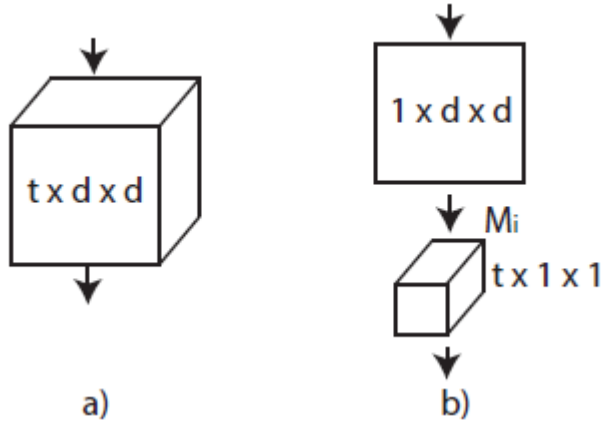
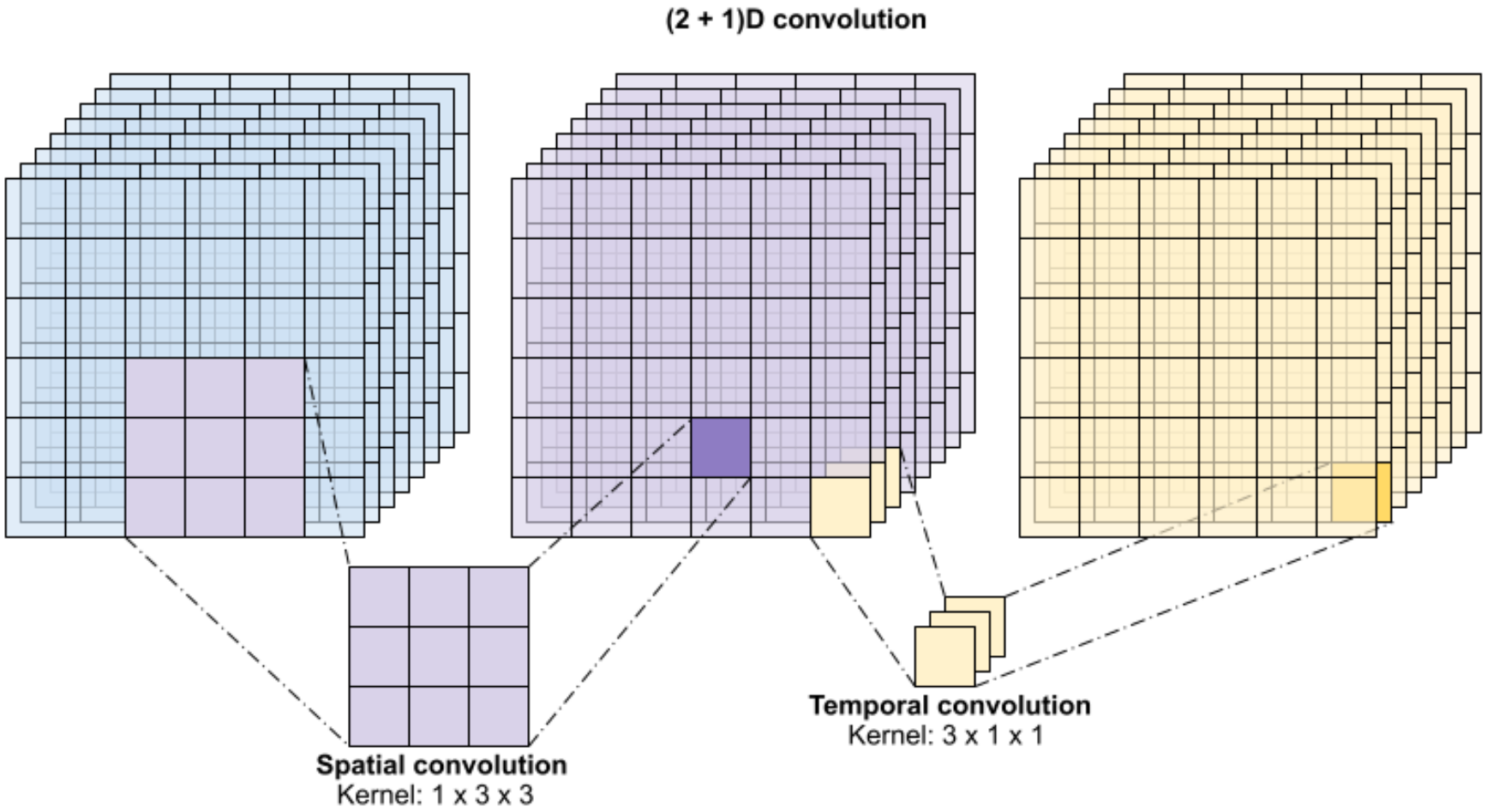


Figure 2. **(2+1)D vs 3D convolution**. The illustration is given for the simplified setting where the input consists of a spatiotemporal volume with a single feature channel. (a) Full 3D convolution is carried out using a filter of size $t \times d \times d$ where t denotes the temporal extent and d is the spatial width and height. (b) A (2+1)D convolutional block splits the computation into a spatial 2D convolution followed by a temporal 1D convolution. We choose the numbers of 2D filters (M_i) so that the number of parameters in our (2+1)D block matches that of the full 3D convolutional block.

Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. A closer look at spatiotemporal convolutions for action recognition. CVPR 2018

(2+1)D Convolution



Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. A closer look at spatiotemporal convolutions for action recognition. CVPR 2018

(2+1)D Convolution

Advantages:

- An additional nonlinearity
- In practice these turn to be easier to optimize

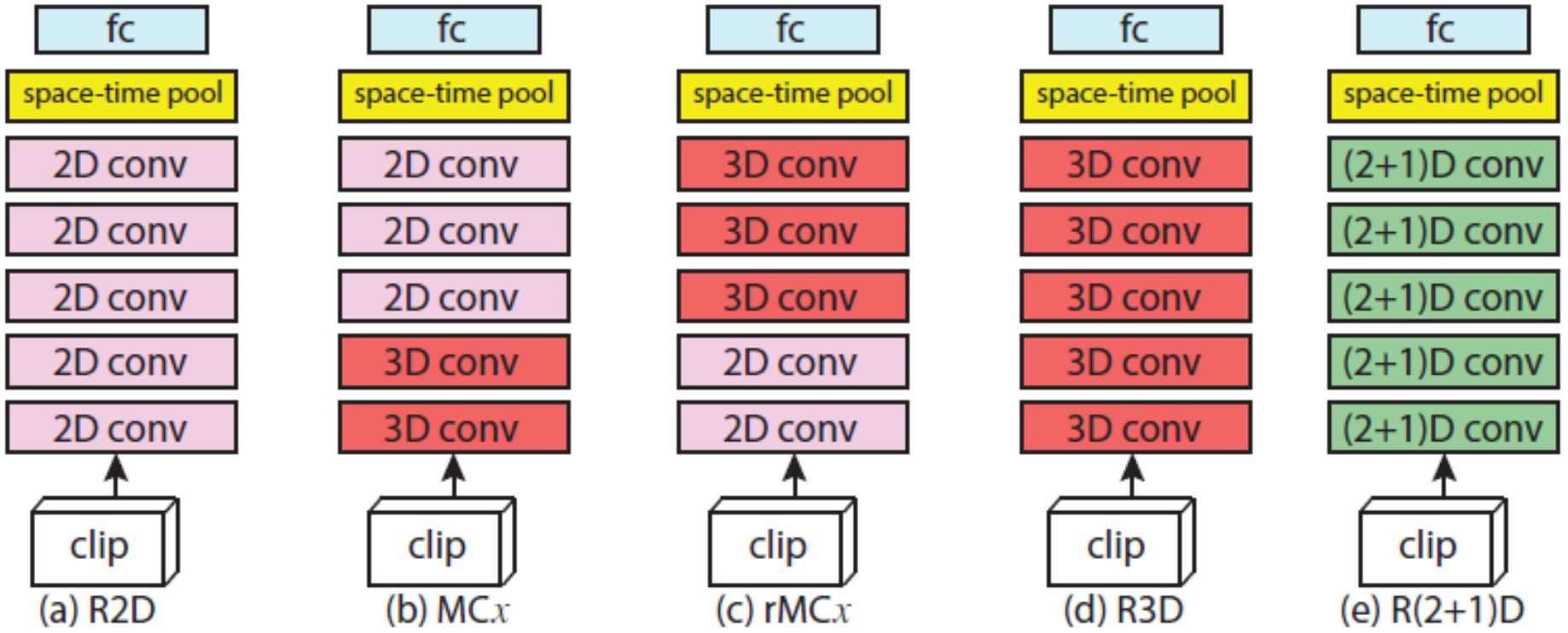
No computational advantages

- (2+1)D conv corresponds to “aggregating terms in the multiply and accumulate expressions”
- MobileNet, leverages a similar principle but shares the same 2D activations for multiple 1D filters… this gives computational advantages

Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. A closer look at spatiotemporal convolutions for action recognition. CVPR 2018

3D Convolution in Videos

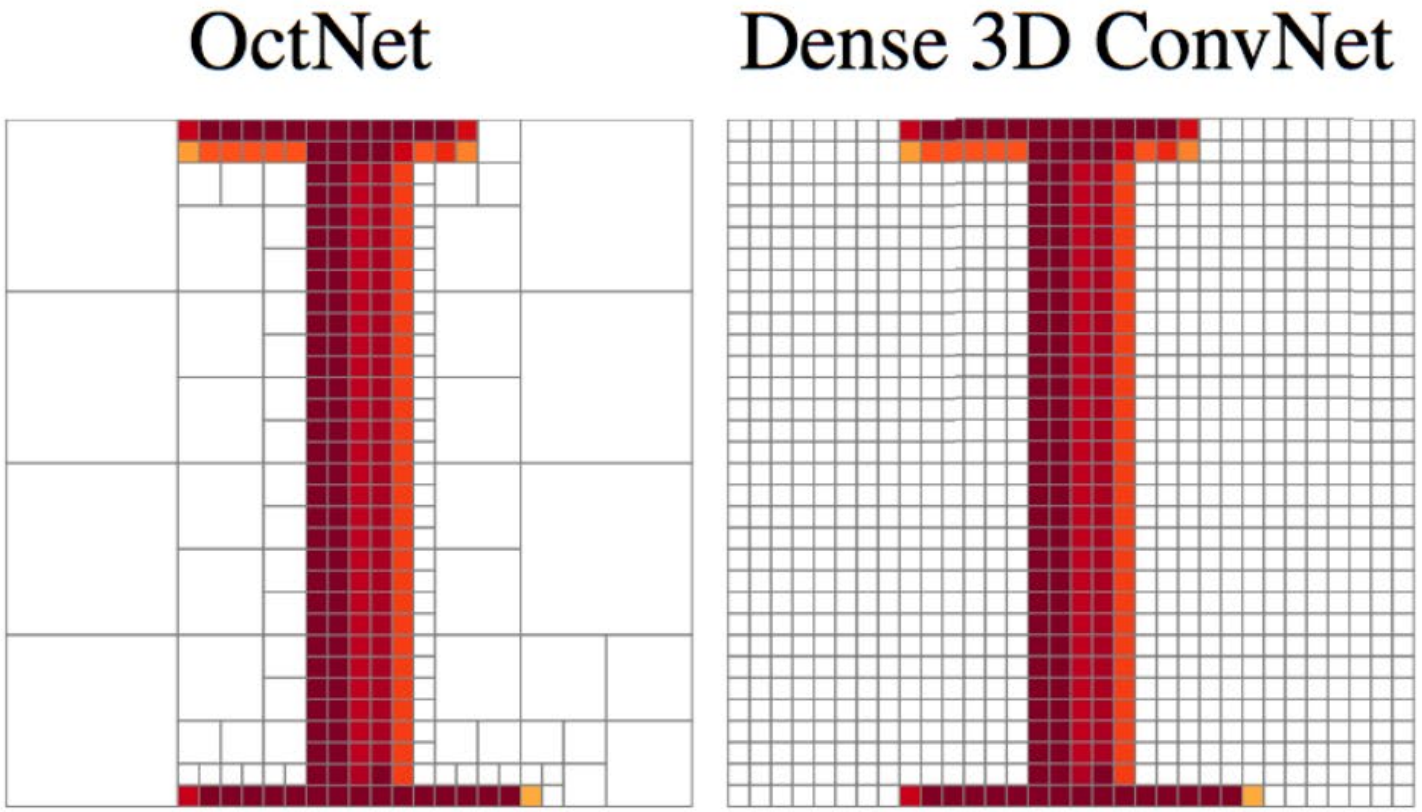
Mixed Convolution: employ conv3D only in the early layers of the network, then conv2D. Early layers model motion via 3D convolutions, and spatial reasoning over mid-level motion features in deep layers



Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. A closer look at spatiotemporal convolutions for action recognition. CVPR 2018

OctTree

Octree are 3D grid structures with adaptive cell size enabling lossless reduction of memory consumption wrt a regular voxel grid.

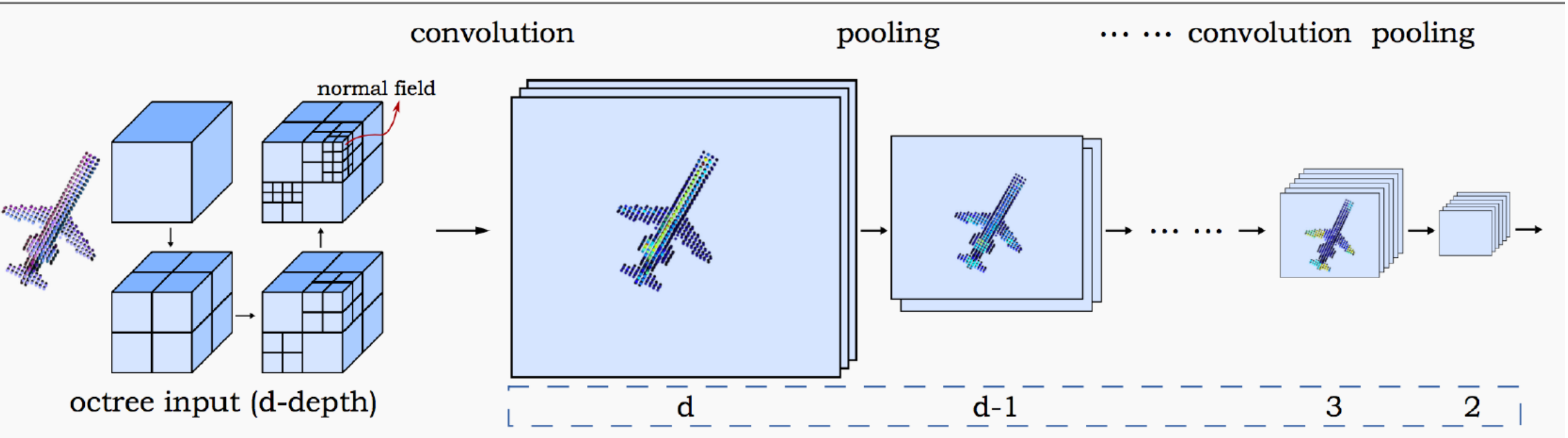


Riegler, G., Osman Ulusoy, A., & Geiger, A. "Octnet: Learning deep 3d representations at high resolutions" CVPR 2017

OctTree

Octree are 3D grid structures with adaptive cell size enabling lossless reduction of memory consumption wrt a regular voxel grid.

However, these networks lack flexibility as their kernels are constrained to use $3^3 = 27$ or $5^3 = 125$ voxels.



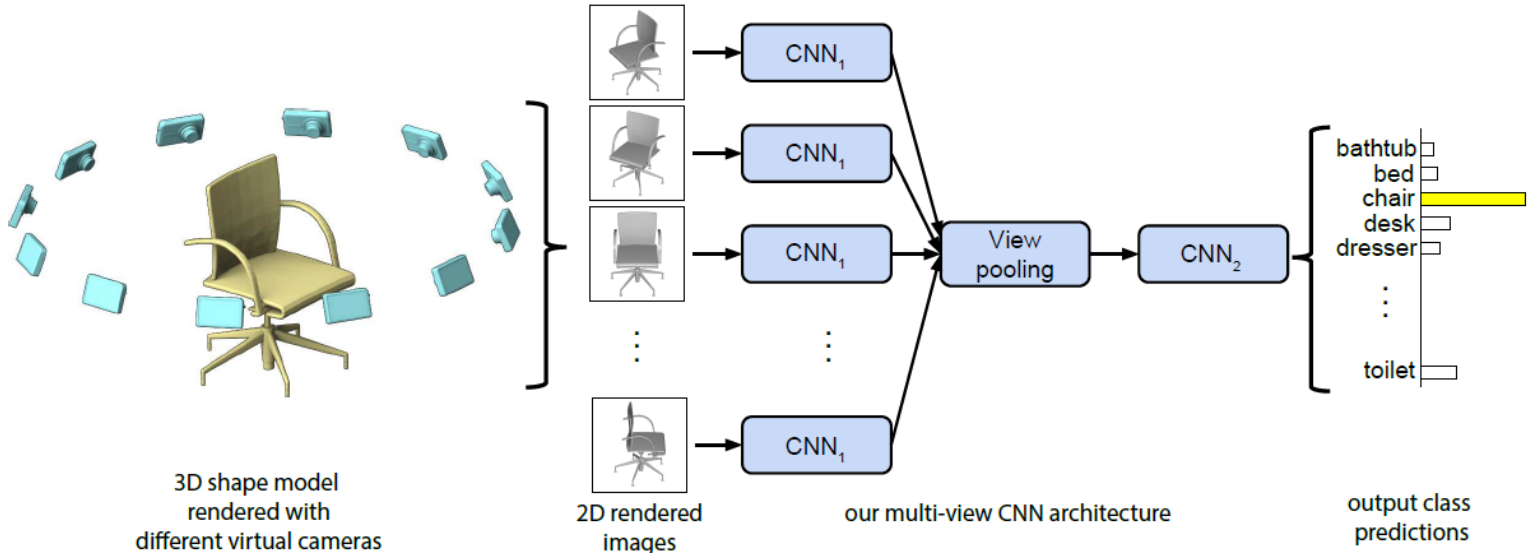
Multi-View CNN

Recognize 3D shapes from a **collection of their rendered views** several 2D images.

No need to use large input size, better training on (smaller) images. Can leverage pre-trained models and datasets for images.

Seem related to “test time augmentation”, but learns how to aggregate a fixed number of **views rendered from virtual cameras in pre-defined locations** (e.g. 30 degrees shift above). **View pooling layer by maxpooling elements across the view.**

They also test “traditional” CV features + SVM classification approach.



H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition", ICCV 2015

Part3: Deep Learning on Point Clouds

Point Net & Deep Sets

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

Charles R. Qi* Hao Su* Kaichun Mo Leonidas J. Guibas
Stanford University

Abstract

Point cloud is an important type of geometric data structure. Due to its irregular format, most researchers transform such data to regular 3D voxel grids or collections of images. This, however, renders data unnecessarily voluminous and causes issues. In this paper, we design a novel type of neural network that directly consumes point clouds, which well respects the permutation invariance of points in the input. Our network, named PointNet, provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing. Though simple, PointNet is highly efficient and effective. Empirically, it shows strong performance on

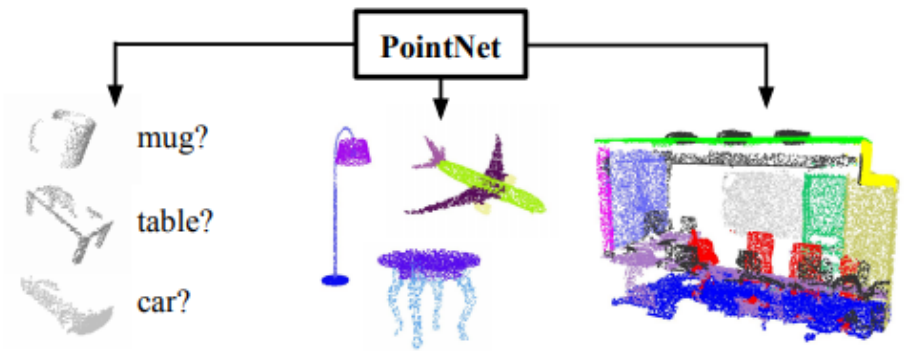


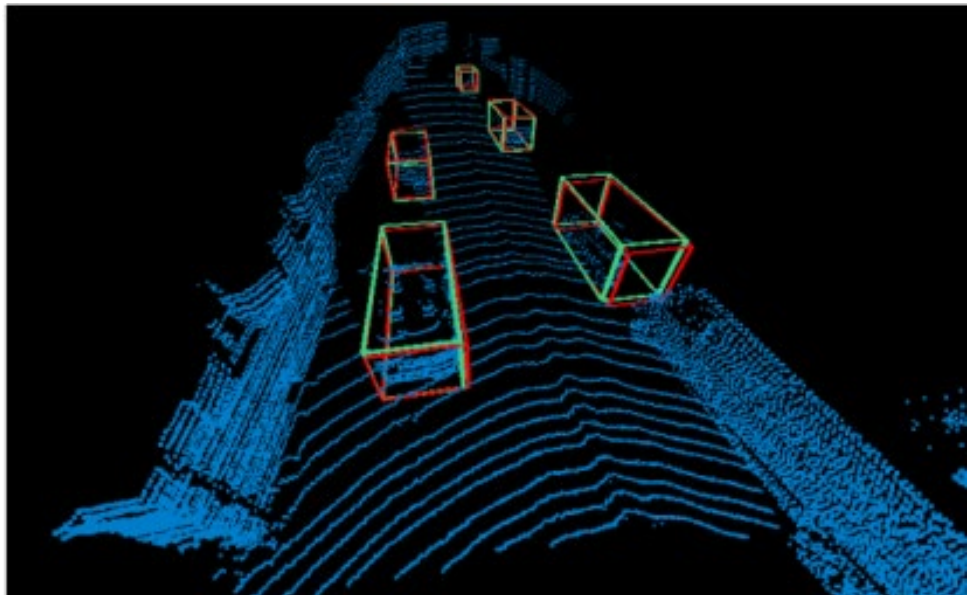
Figure 1. **Applications of PointNet.** We propose a novel deep net architecture that consumes raw point cloud (set of points) without voxelization or rendering. It is a unified architecture that learns both global and local point features, providing a simple, efficient and effective approach for a number of 3D recognition tasks.

The Limitations of Voxelization

Lidars and other 3D scanning devices provide **sparse 3D outputs**

Moving to voxels or multiple images to perform deep learning is **suboptimal**:
- data becomes incredibly **larger**
- introduce artifacts and **approximations**
- we **loose intrinsic invariances** of the 3D sets

Point Clouds need to be treated as sets of 3D points:
- these do not admit duplicates,
- do not have a specific order,
- need to be invariant to rigid rotations
- relative position matters, as they are embedded in a metric space



PointNet

This is a network meant to process sets of 3D points

- We do not want the order of points in the PC to matter!
  - ➔We want the network to perform only operations that are invariant to the order of points.
- We do not want the chosen reference system to matter!
  - ➔The semantic labeling of a point cloud must be invariant if the point cloud undergoes rigid transformations. Therefore, the learnt representation by our point set need to be invariant to these transformations.

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet and DL on Sets

Three options for handling unordered set of points:

1) sort input into a canonical order;

2) treat the input as a sequence to train an RNN, but augment the training data by all kinds of permutations;

3) use a simple symmetric function to aggregate information from each point.

PointNet

The input

	3		
N	x_1	y_1	z_1
	x_2	y_2	z_2
	x_N	y_N	z_N

Qi et al, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”, CVPR 2017

PointNet

The input

N × 6

x_1	y_1	z_1	r_1	g_1	b_1
x_2	y_2	z_2	r_2	g_2	b_2
x_N	y_N	z_N	r_N	g_N	b_N

Qi et al, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”, CVPR 2017

PointNet

The input

N × d

x_1	y_1	z_1	f_1	...	f_1^M
x_2	y_2	z_2	f_2	...	f_2^M
...	...	...			
x_N	y_N	z_N	f_N	...	f_N^M

Qi et al, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”, CVPR 2017

PointNet

The input

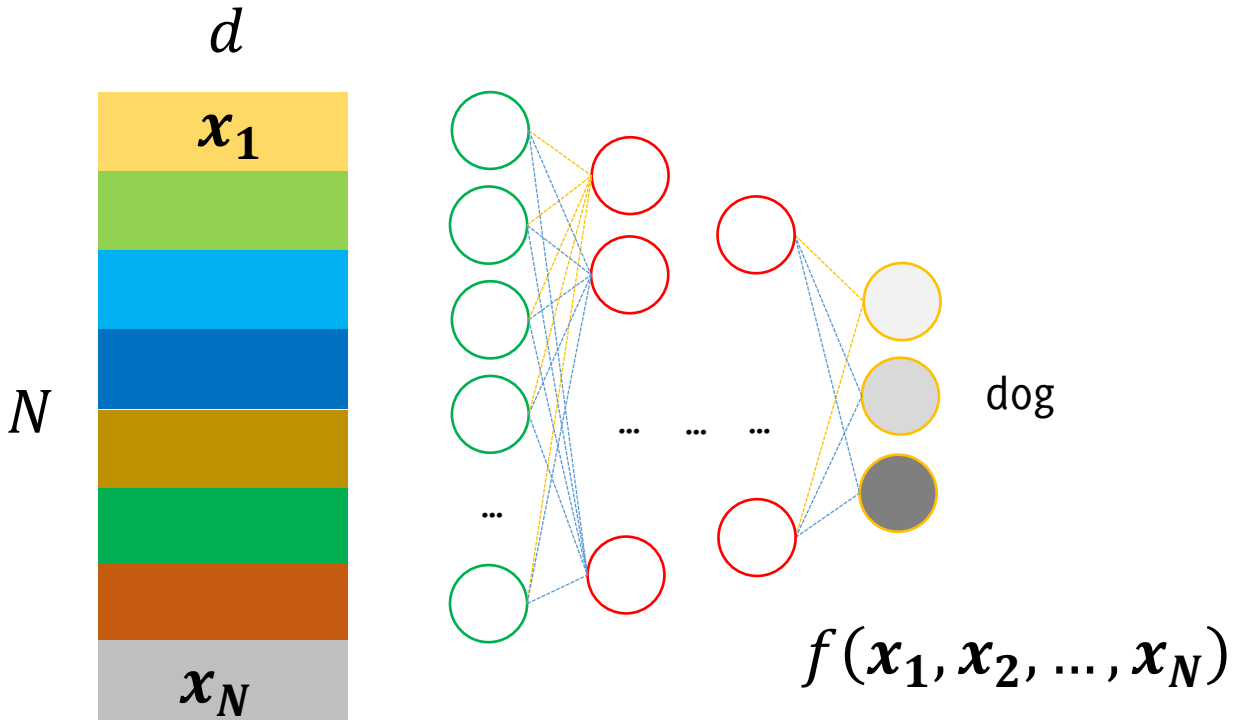
N × d

x_1	y_1	z_1	f_1	…	f_1^M
x_2	y_2	z_2	f_2	…	f_2^M
…	…	…			
x_N	y_N	z_N	f_N	…	f_N^M

Qi et al, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”, CVPR 2017

PointNet

The input does not change when changing the order of points by a permutation function π, so should do the output

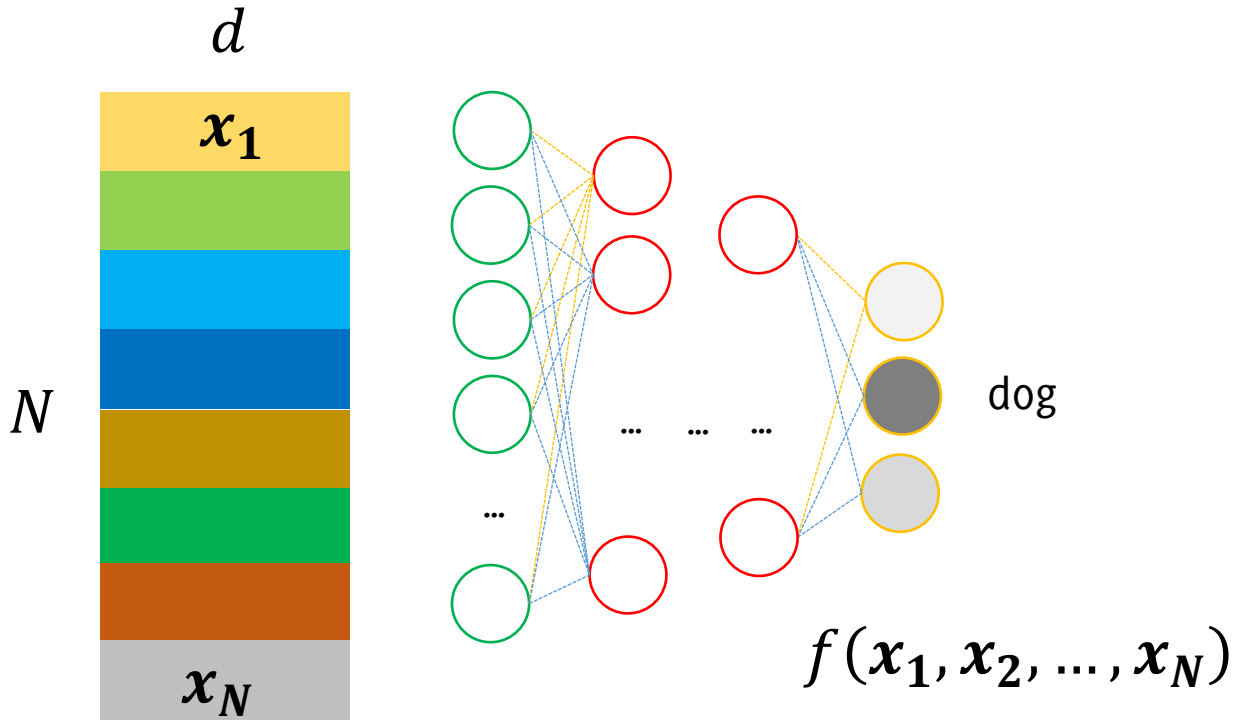


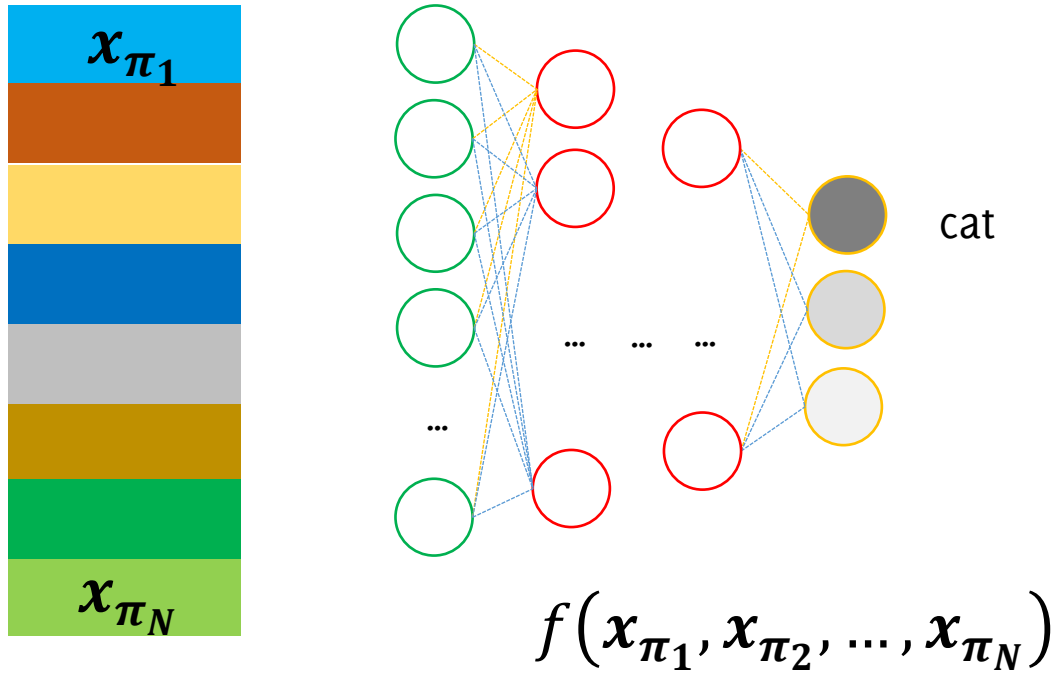
$(x_1, x_2, \ldots, x_N)$

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet

The input does not change when changing the order of points by a permutation function π, so should do the output





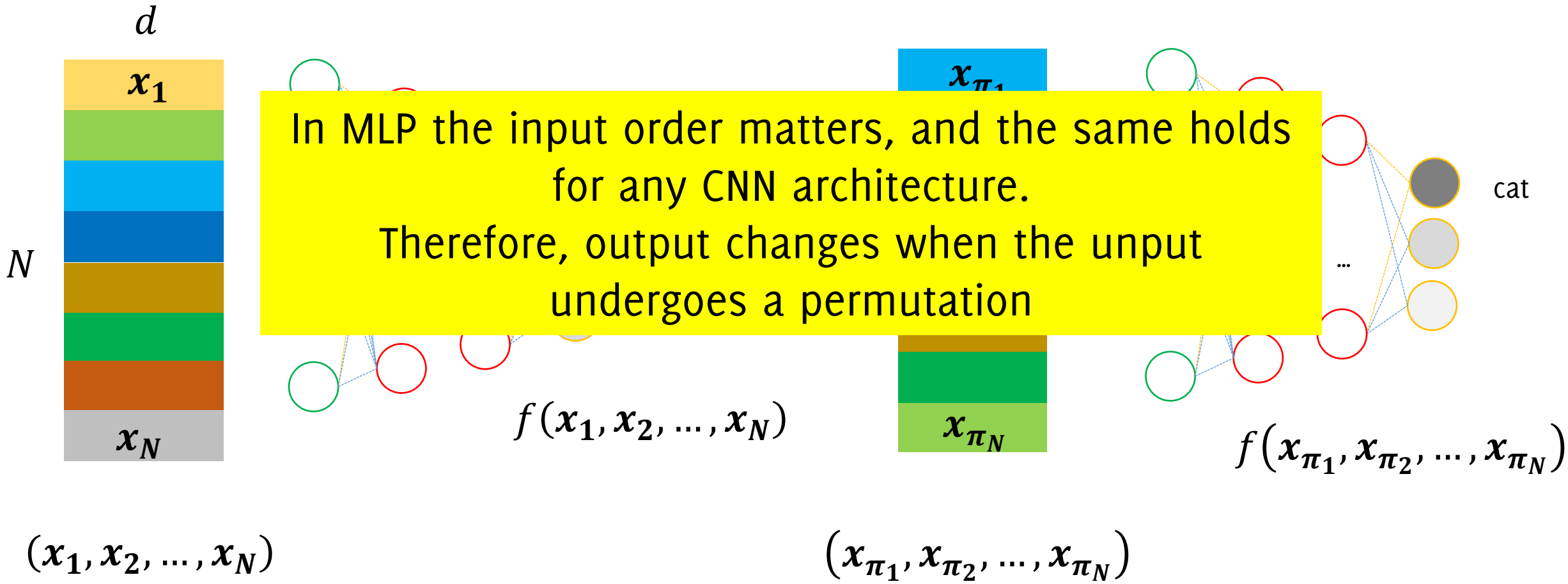
$(x_1, x_2, \dots, x_N)$

$(x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_N})$

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet

The input does not change when changing the order of points by a permutation function π, so should do the output



Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

Permutation Invariant Networks

The “ideal” classification function

$$k\colon 2^{\mathbb{R}^N} \to \mathbb{R}^K$$

Need to be a symmetric function, namely invariant to the input order.

Example of invariant functions:

- Aggregation functions (maximum, minimum, average)
- Any function taking a single input

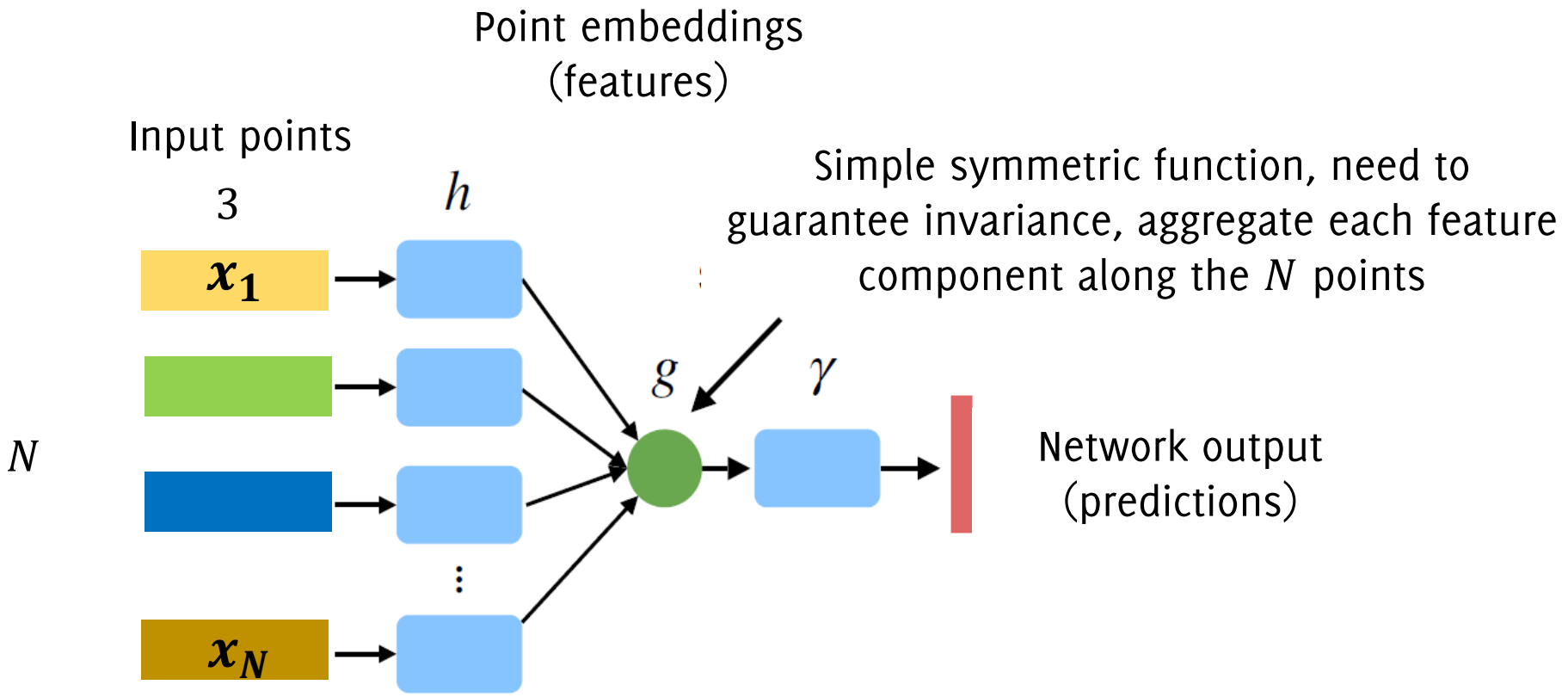
PointNet decision: can approximate our classification network by the following symmetric function

$$f(x_1, \dots, x_N) \approx \gamma \circ g\big(h(x_1), \dots, h(x_N)\big)$$

where:

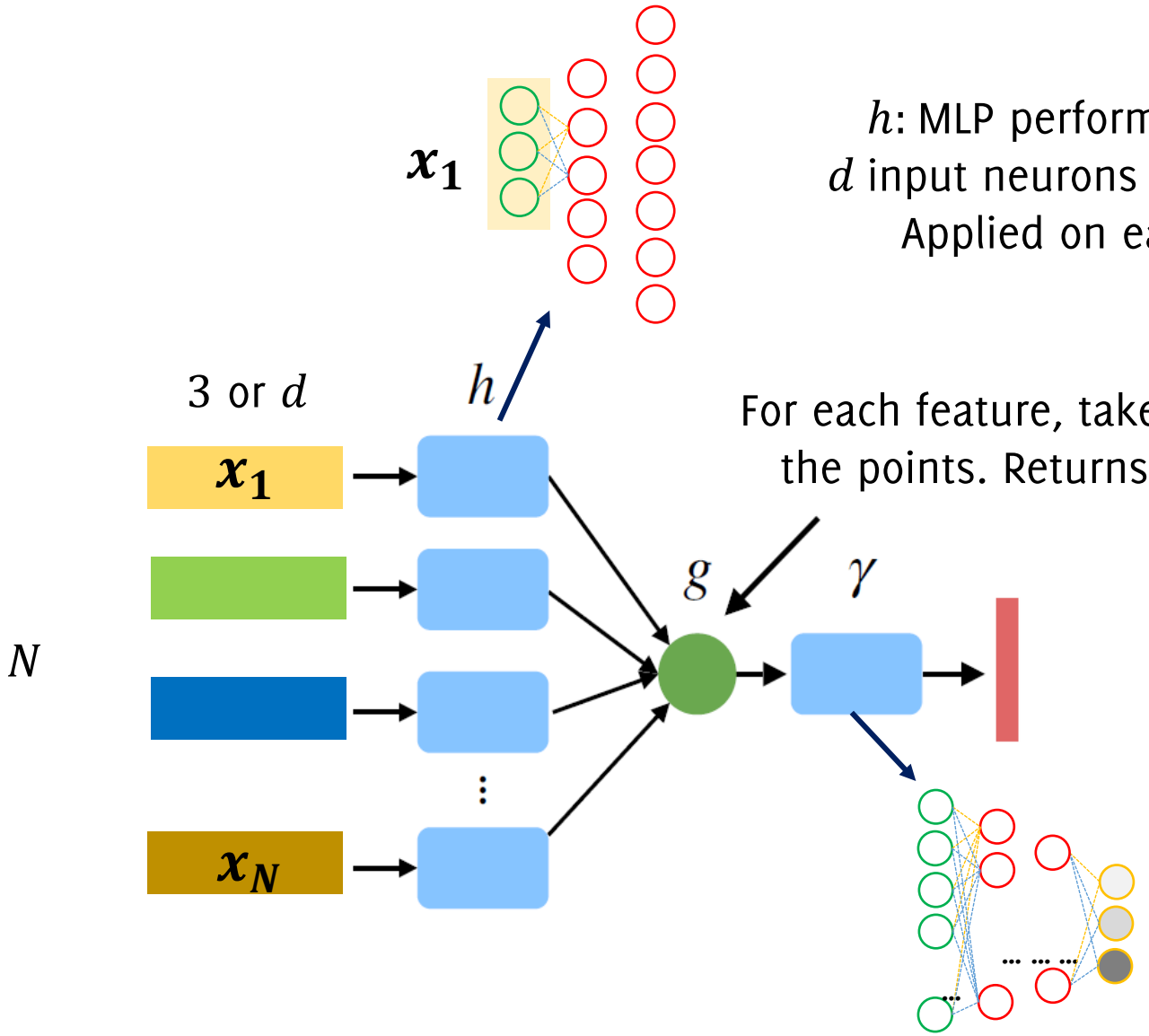
- h, γ are MLP taking as input a single vector
- g an aggregation function

PointNet Vanilla (1-layer PointNet)



Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet Vanilla (1-layer PointNet)



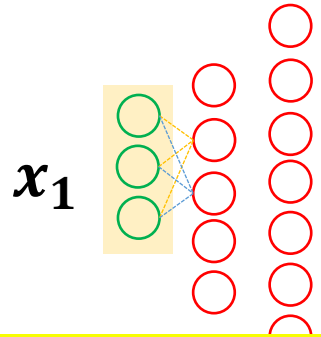
h: MLP performing Point Embedding
d input neurons and D output neurons.
Applied on each of the N points

For each feature, take the maximum along all the points. Returns D-dimensional vector

γ: MLP performing classification of aggregated features from the PC.
Takes D inputs and outputs K, number of classes/predictions

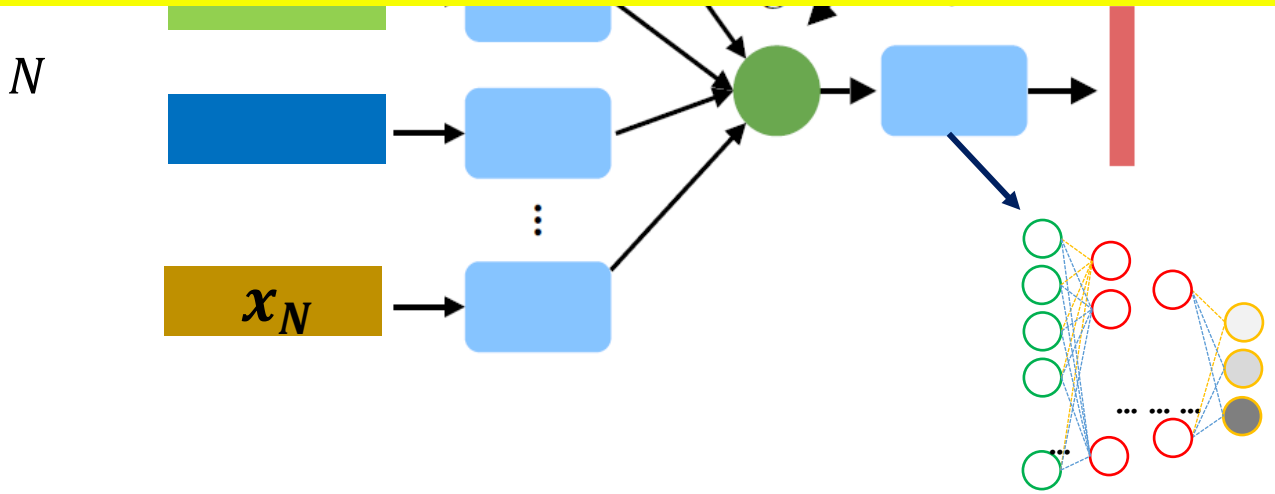
Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet Vanilla (1-layer PointNet)



h: MLP performing Point Embedding
d input neurons and D output neurons.
Applied on each of the N points

PointNet by design can process an arbitrary number of points. This is apparent from the architecture and is a desiderata when workingon point clouds

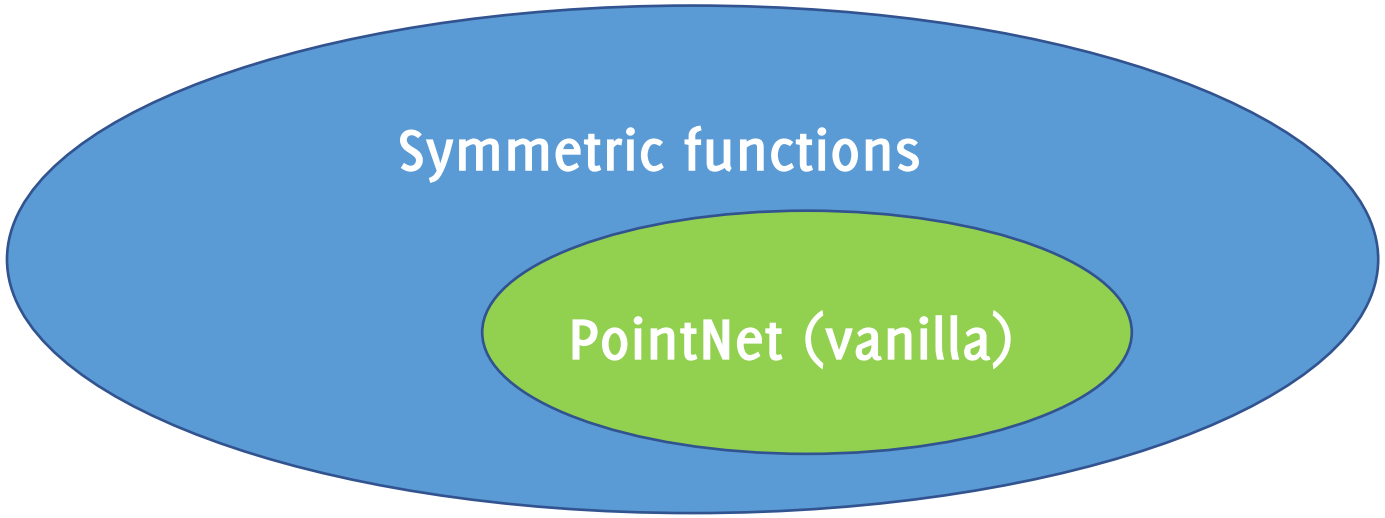


γ: MLP performing classification of aggregated features from the PC.
Takes D inputs and outputs K, number of classes/predictions

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

Theorem

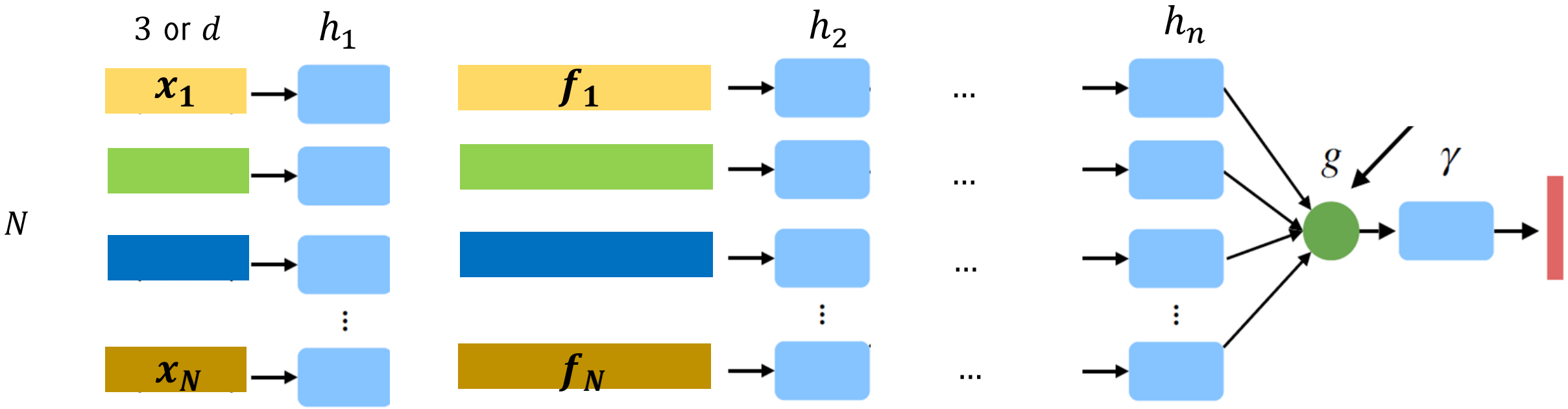
In the paper a theoretical results about which symmetric functions can be approximated by PointNet is reported.



How to make PointNet Deep?

Aggregate multiple point-wise embedding layers.

These always operate on each point separately, to preserve symmetry



Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet

This is a network meant to process sets of 3D points

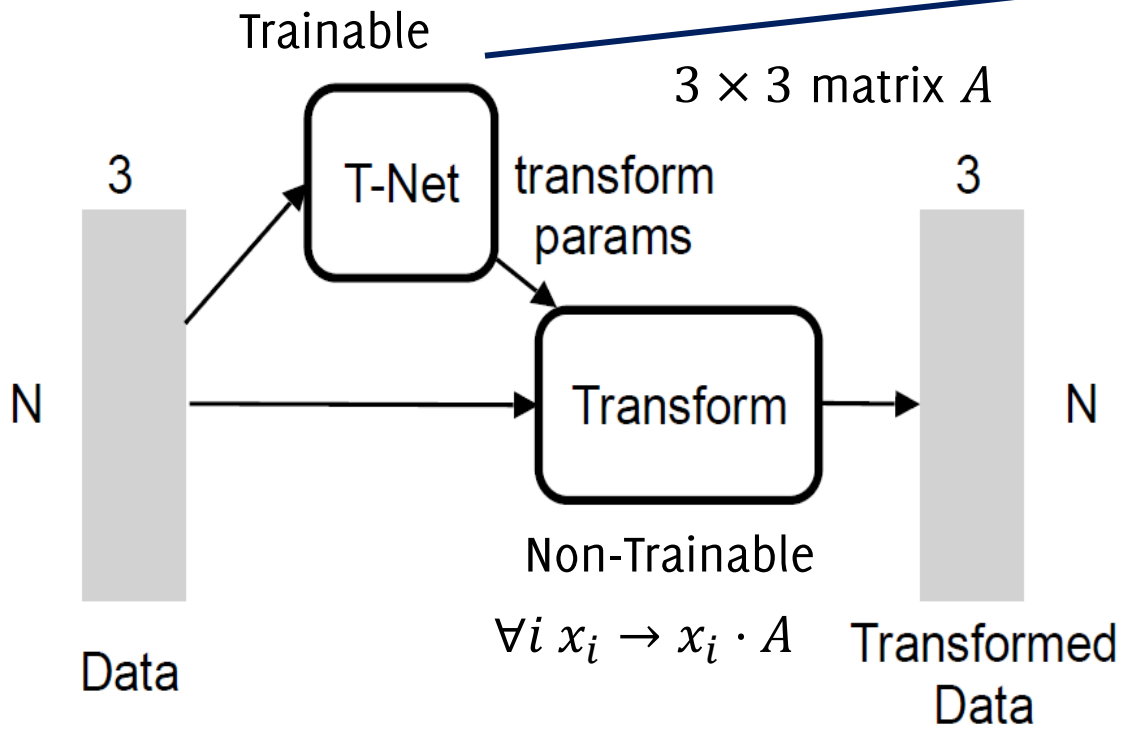
- We do not want the order of points in the PC to matter!
  - ➔ We want the network to perform only operations that are invariant to the order of points.
- We do not want the chosen reference system to matter!
  - ➔ The semantic labeling of a point cloud must be invariant if the point cloud undergoes rigid transformations. Therefore, the learnt representation by our point set need to be invariant to these transformations.

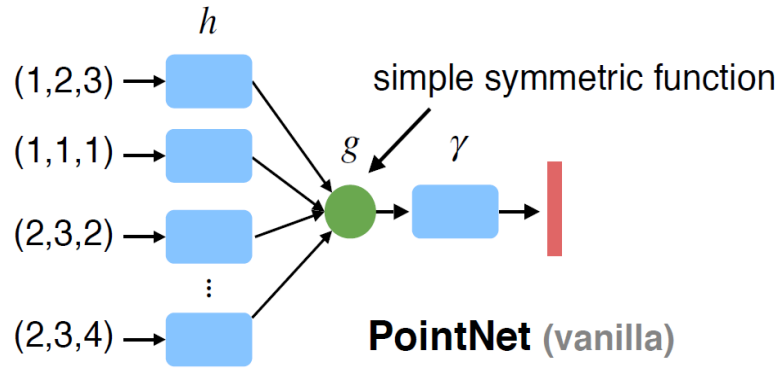
Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

Transformation Network

Idea behind transformation network

- learn a transformation to align all the input points to a canonical space!
- apply the same principle to feature space





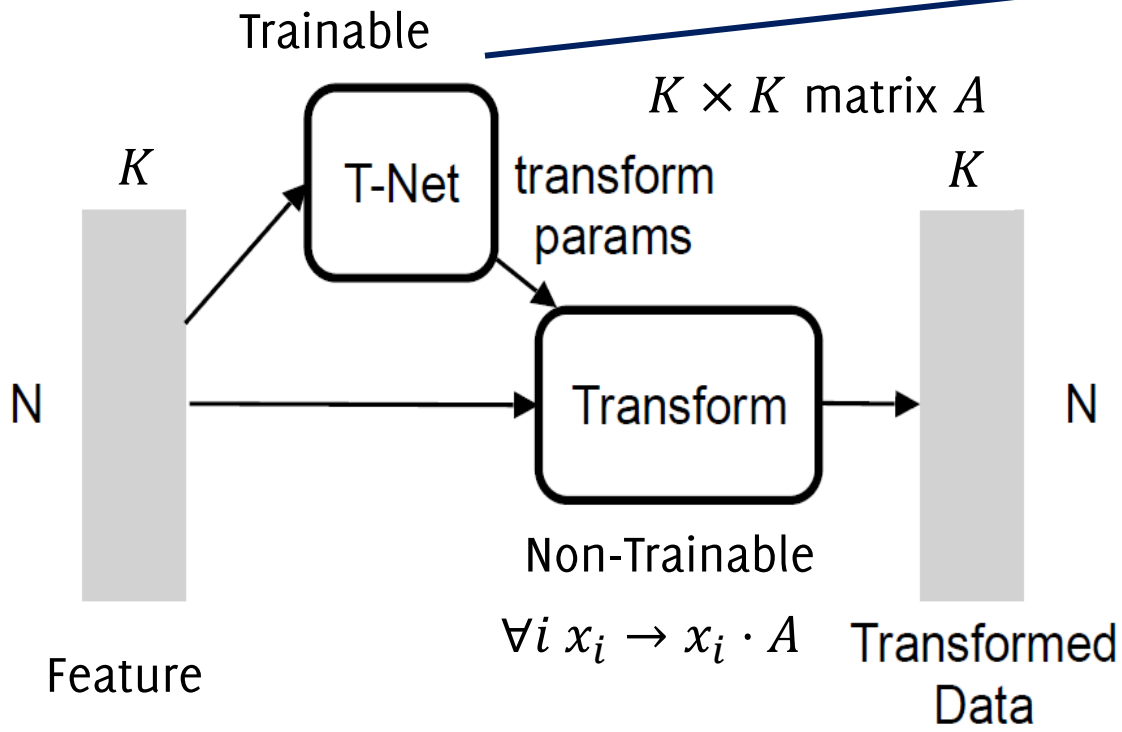
This has to be a symmetric function,
It's a vanilla PointNet trained for regression with 9 outputs

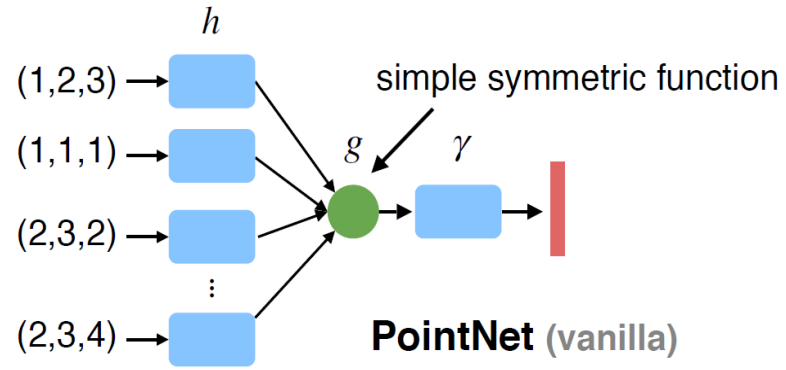
Transformation Network

Idea behind transformation network

- learn a transformation to align all the input points to a canonical space!

- apply the same principle to feature space





This network solves a more difficult learning problem since feature dimension is larger than inputs ($D \gg d$)

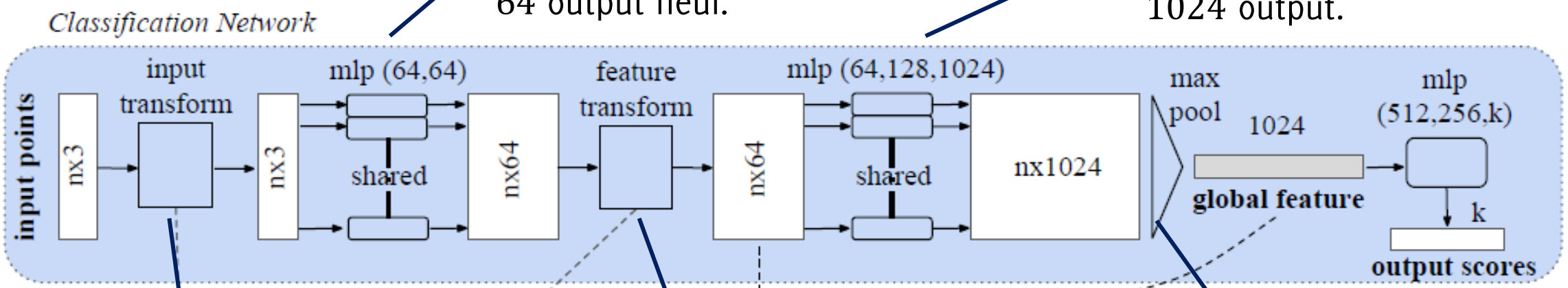
A regularization term added to the loss to get a better conditioned matrix (close to a rotation)

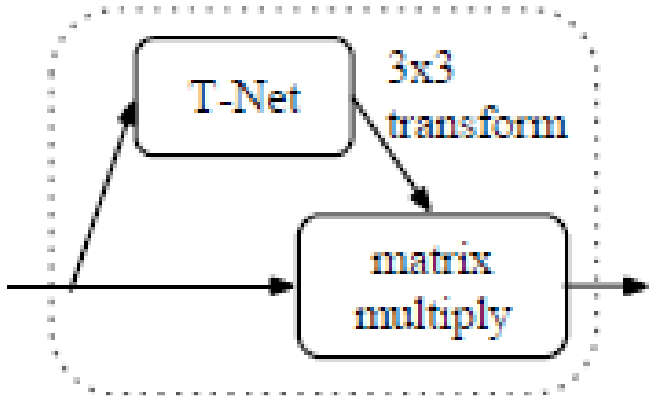
$$\mathcal{R}(A) = \|I - A^T \cdot A\|_F$$

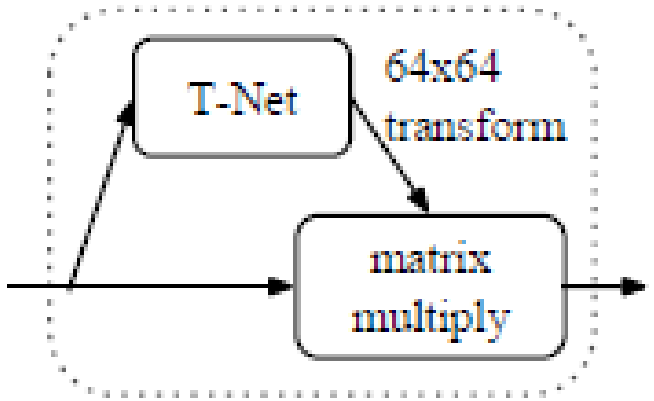
PointNet Classification Architecture

3 input neur.,
64 hidden neur.,
64 output neur.

64 inputs,
64 hidden,
128 hidden,
1024 output.







g: maxpooling along the points for each component.
This is still, the **only operation mixing points.**

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet Segmentation Architecture

Conflicting goals of segmentation: you need to combine locality (we need prediction on each point) and global information (we need to interpret semantic)

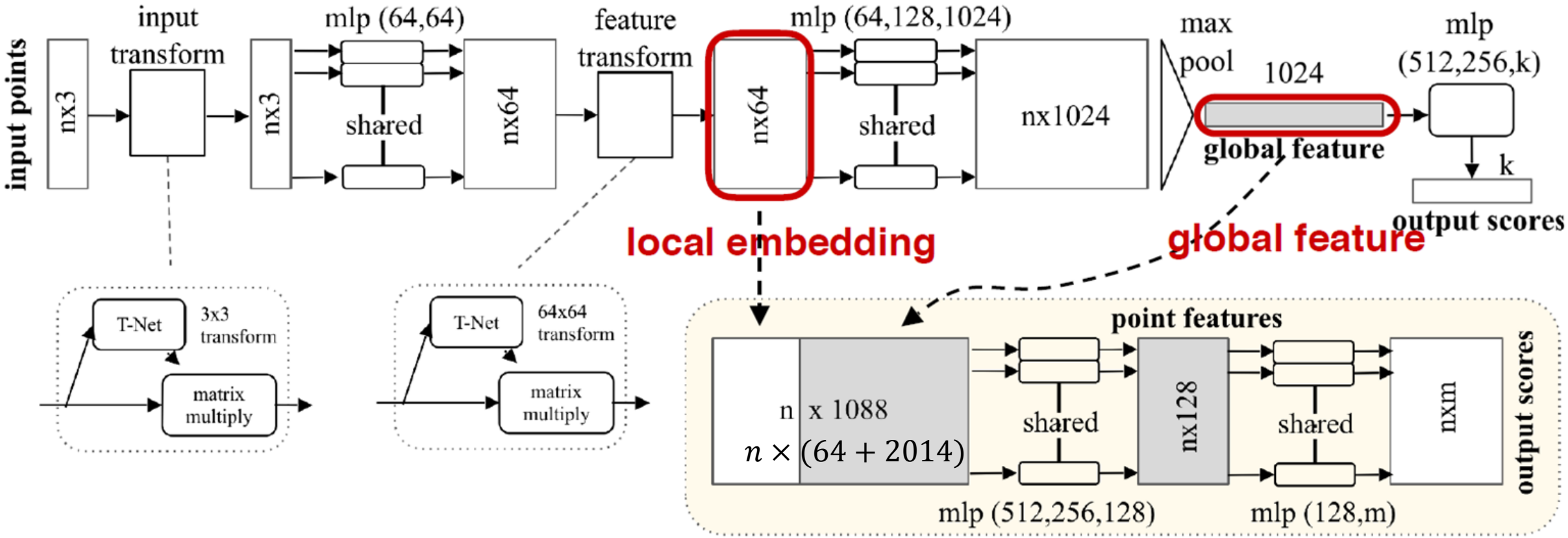
Idea:

- Compute first a global embedding from the input point cloud,
- Feed it back to each point features by concatenation,
- Process the augmented input with a new MLP combining local/global features
- Achieve dense prediction on each point

Point feature (and predictions) are now aware of both the local and global information.

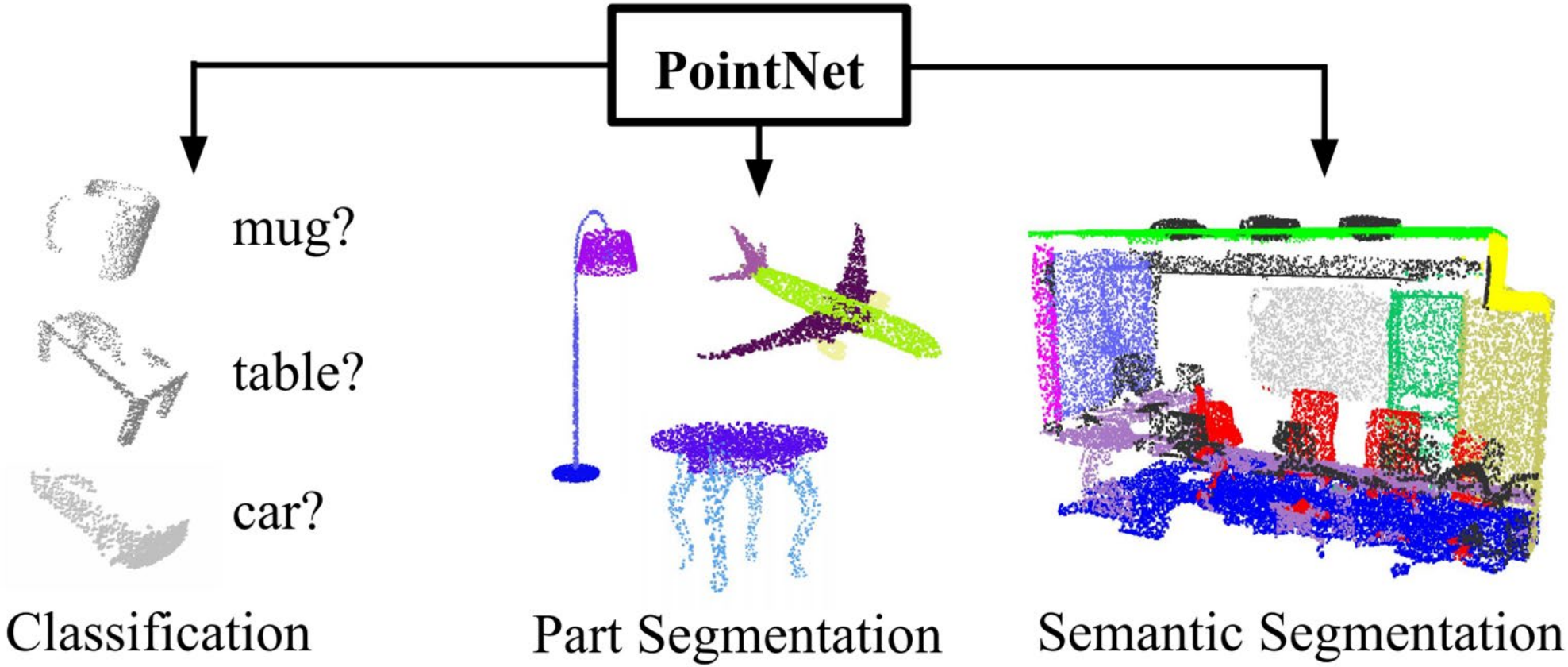
PointNet Segmentation Architecture

The architecture partially overlaps with PointNet for Classification



Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet Applications



Assign part category label (e.g. chair leg, cup handle) to each point or face.

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

Classification Performance

PointNet is **much faster and more accurate / on par** than state of the art deep learning **models based on voxelized** representation.

With only fully connected layer and max pooling, our net gains a ***strong lead in inference speed*** *and can be* ***easily parallelized in CPU*** *as well*

	input	#views	accuracy avg. class	accuracy overall
SPH [11]	mesh	-	68.2	-
3DShapeNets [25]	volume	1	77.3	84.7
VoxNet [15]	volume	12	83.0	85.9
Subvolume [16]	volume	20	86.0	**89.2**
LFD [25]	image	10	75.5	-
MVCNN [20]	image	80	**90.1**	-
Ours baseline	point	-	72.6	77.4
Ours PointNet	point	1	86.2	**89.2**

Table 1. **Classification results on ModelNet40.** Our net achieves state-of-the-art among deep nets on 3D input.

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

Ability to Process 3D PC with missing data

Less than 2% accuracy drop with 50% missing data

This emphasizes that PC can have different sizes and Point-Net based architecture **can natively process sets with different inputs**

dataset: ModelNet40; metric: 40-class classification accuracy (%)

Deep Sets

Manzil Zaheer[1,2], **Satwik Kottur**[1], **Siamak Ravanbhakhsh**[1],
Barnabás Póczos[1], **Ruslan Salakhutdinov**[1], **Alexander J Smola**[1,2]
[1] Carnegie Mellon University [2] Amazon Web Services
{manzilz,skottur,mravanba,bapoczos,rsalakhu,smola}@cs.cmu.edu

Abstract

We study the problem of designing models for machine learning tasks defined on *sets*. In contrast to traditional approach of operating on fixed dimensional vectors, we consider objective functions defined on sets that are invariant to permutations. Such problems are widespread, ranging from estimation of population statistics [1], to anomaly detection in piezometer data of embankment dams [2], to cosmology [3, 4]. Our main theorem characterizes the permutation invariant functions and provides a family of functions to which any permutation invariant objective function must belong. This family of functions has a special structure which enables us to design a deep network architecture that can operate on sets and which can be deployed on a variety of scenarios including both unsupervised and supervised learning tasks. We also derive the necessary and sufficient conditions for permutation equivariance in deep models. We demonstrate the applicability of our method on population statistic estimation, point cloud classification, set expansion, and outlier detection.

Deep Sets

More general perspective: designing ML models for tasks where inputs and possibly outputs are sets.

Addresses more general problems: population statistic estimation, point cloud classification, set expansion (retrieval), and outlier detection.

Get to a very similar theoretical results

Theorem 2 *A function $f(X)$ operating on a set X having elements from a countable universe, is a valid set function,* i.e., ***invariant*** *to the permutation of instances in X, iff it can be decomposed in the form $\rho\left(\sum_{x \in X} \phi(x)\right)$, for suitable transformations ϕ and ρ.*

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. NeurIPS

Deep Sets

The structure of permutation invariant functions

$$\rho\left(\sum_{x\in X}\phi(x)\right)$$

We replace ρ and ϕ by universal approximators (e.g. MLP) to be learned.

1. Each input point x is individually mapped via a MLP ϕ to its own representation
2. All the representations of the sets are summed
3. Another MLP γ is used to infer predictions

The key is to add up all representations and then apply nonlinear transformations

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. NeurIPS

PointNet Limitations

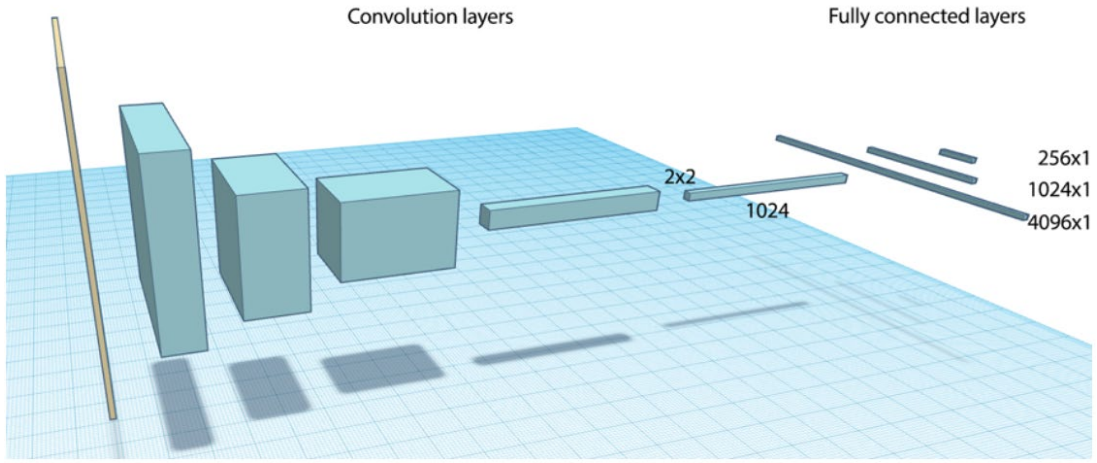
By design PointNet **does not capture local structures**, while locality in the inputs matters since points live in a metric space.

Global features depend on absolute coordinates, difficult to generalize to unseen settings

Local patterns are important to increase the power of deep NN on visual data

- in recognizing fine-grained patterns
- generalizing to complex scenes

Neurons along the network depth can grasp details at different resolution, as their receptive field increases (and features are richer with more nonlinearities)



Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017

PointNet++

PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space

Charles R. Qi **Li Yi** **Hao Su** **Leonidas J. Guibas**
Stanford University

Abstract

Few prior works study deep learning on point sets. PointNet [20] is a pioneer in this direction. However, by design PointNet does not capture local structures induced by the metric space points live in, limiting its ability to recognize fine-grained patterns and generalizability to complex scenes. In this work, we introduce a hierarchical neural network that applies PointNet recursively on a nested partitioning of the input point set. By exploiting metric space distances, our network is able to learn local features with increasing contextual scales. With further observation that point sets are usually sampled with varying densities, which results in greatly decreased performance for networks trained on uniform densities, we propose novel set learning layers to adaptively combine features from multiple scales. Experiments show that our network called PointNet++ is able to learn deep point set features efficiently and robustly. In particular, results significantly better than state-of-the-art have been obtained on challenging benchmarks of 3D point clouds.

Qi, C. R., Yi, L., Su, H., & Guibas, L. J. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". NIPS 2017

PointNet++

A hierarchical neural network that processes a subset of **points sampled in a metric space in a hierarchical** manner.

Idea:

- **Group points** into nonoverlapping regions **(neighborhoods)** by **considering 3D distances.**
- **Extract features in** small **neighborhoods.**
- Produce **higher-level features by aggregating local features** in larger units.

Issues:

- how to **define neighborhoods** from a point set?
- how to **extract local features** from point clouds?

Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

PointNet++

A hierarchical neural network that processes a subset of **points sampled in a metric space in a hierarchical** manner.

Idea:

- **Group points** into nonoverlapping regions **(neighborhoods)** by **considering 3D distances.**
- **Extract features in** small **neighborhoods.**
- Produce **higher-level features by aggregating local features** in larger units.

Issues:

- how to **define neighborhoods** from a point set?
- how to **extract local features** from point clouds?

-> will see soon

-> using PointNet as a building block, of course…

Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

Local Processing in PointNet++

PointNet++: a hierarchical stack of **set abstraction levels** that group points and progressively abstract higher-level feature from larger local regions

Set abstraction level:

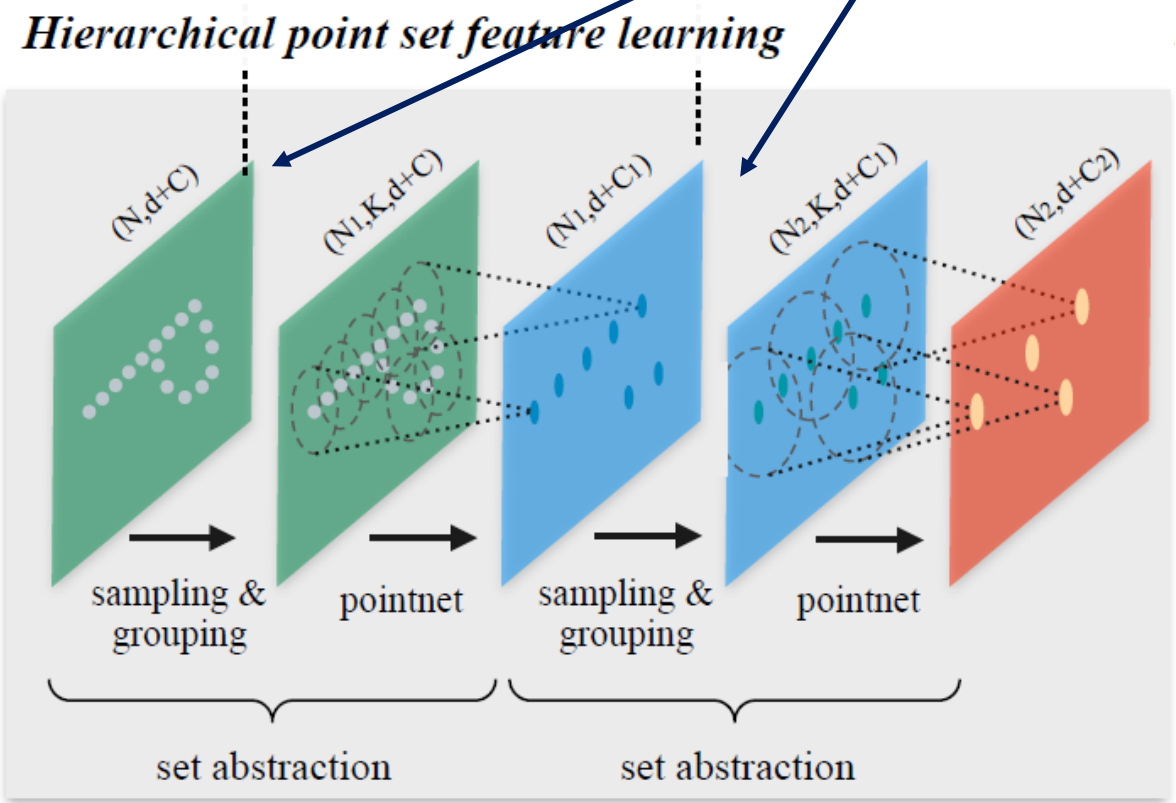
1. **Sample points in the input** at random (anchor points) or using an algorithm (FPS).
2. **Introduce local neighborhoods** around each anchor:
    - Balls, characterized by center and radius (slow)
    - k-NN neighboroods (faster)
3. From each local neighbor we **extract a single local feature using pointNet**. This reduces the point set cardinality. **Preserve the coordinates of anchors** in the output

By concatenating a few set abstraction levels, as the PC progresses along the network, fewer points remain and features are associated to larger regions.

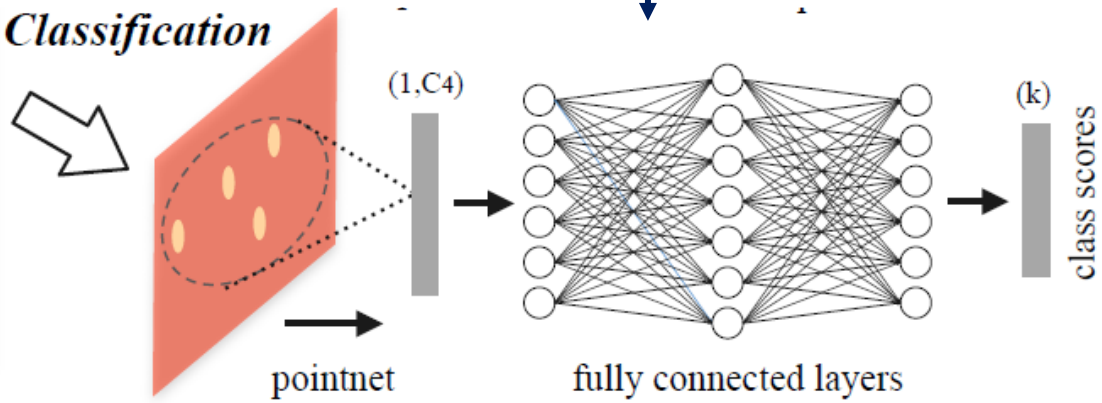
Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

Classification in PointNet++

The sampling and grouping is new to PointNet



This classification block is based on PointNet, followed by a MLP classification head



Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

Set Abstraction levels

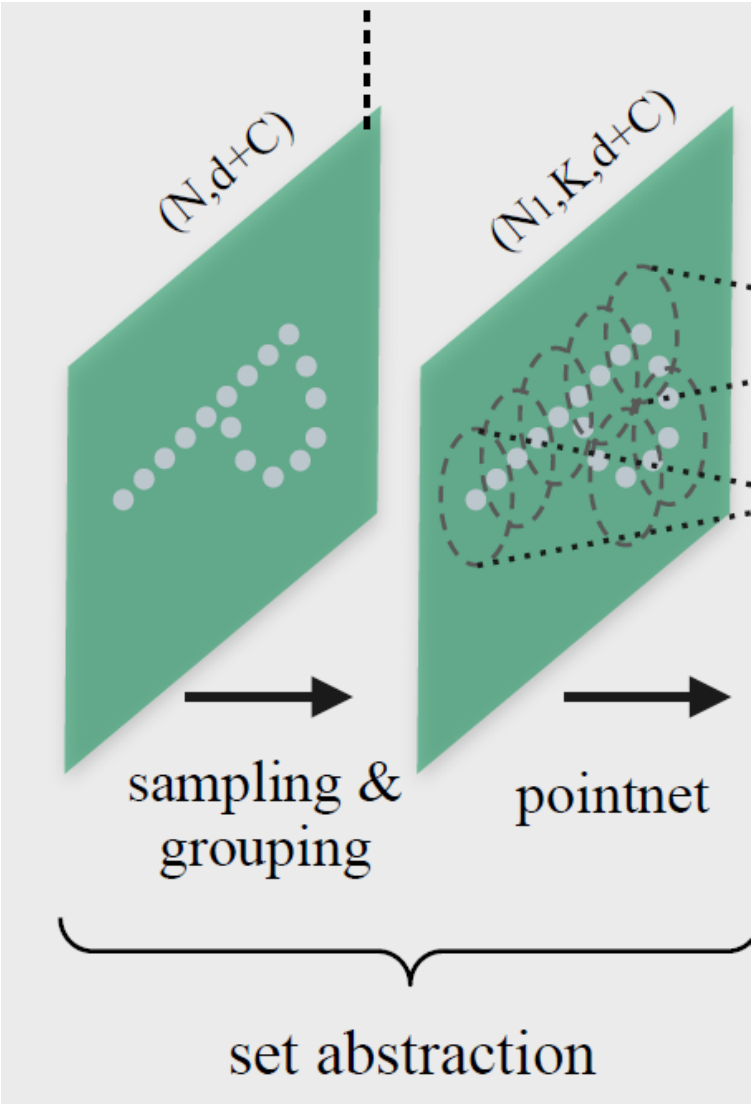
Set abstraction level: it takes as input a set of points to abstract a new set with fewer points and richer features:

- *Sampling layer,*
- *Grouping layer,*
- *PointNet layer.*

Input of Set Abstraction level: N points, $d + C$ values each.

Output of Set Abstraction level: N' points, $d + C'$ values each. **Preserve the original coordinates of anchor points**

Typically points decrease ($N \geq N'$)
while features increase ($C \leq C'$)

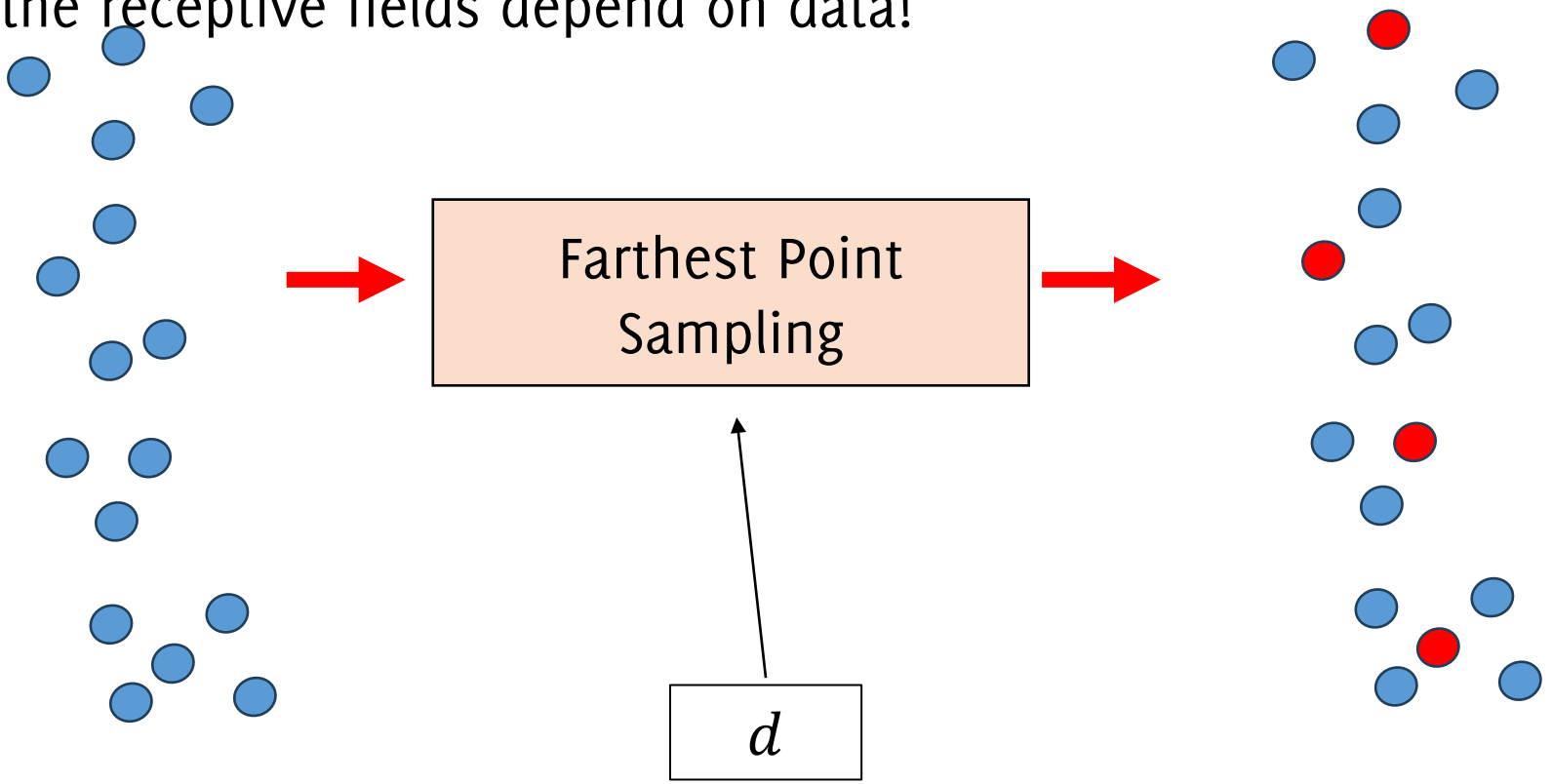


Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

Sampling layer

Farthest Point Sampling (FPS): Given $\{x_1, \dots, x_N\}$ select a set of query points $\{x_{i_1}, \dots, x_{i_N,}\}$ such that x_{i_j} is the most distant point from $\{x_{i_1}, \dots, x_{i_{j-1}}\}$

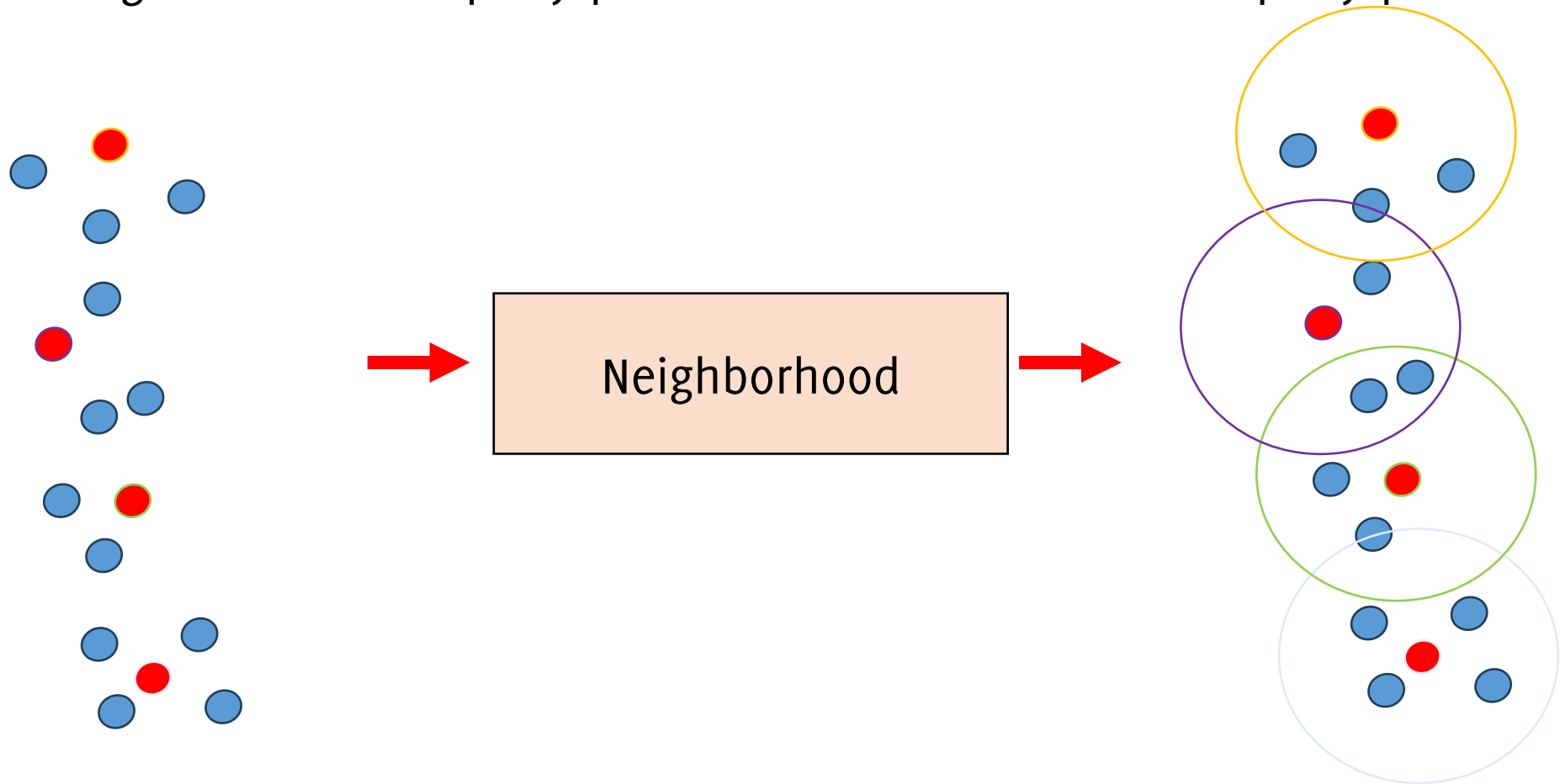
FPS: Better than random search, NN training converges faster.
Moreover, the receptive fields depend on data!



Slide arranged from Melzi and Marin

Grouping layer

Extract neighbors around query points. Balls centered around query points



N' Query points

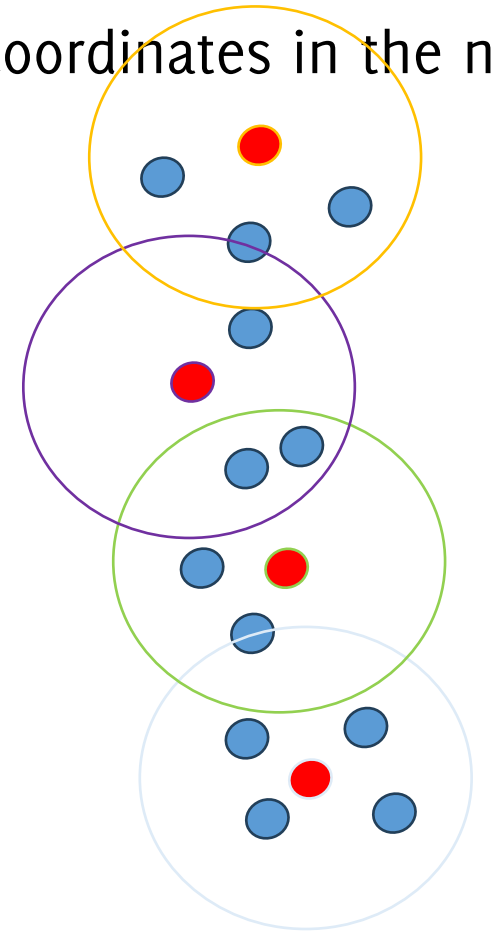
N' neighbors of K points maximum each

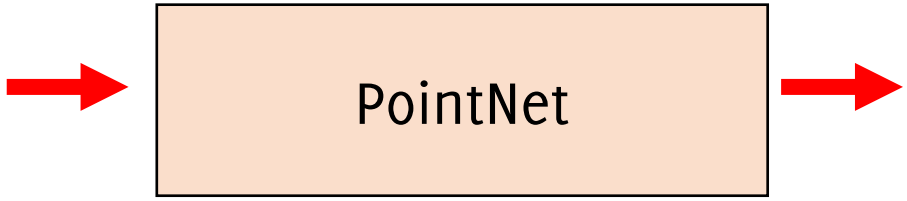
$N' \times K \times (d + C)$. Neighborhood can overlap

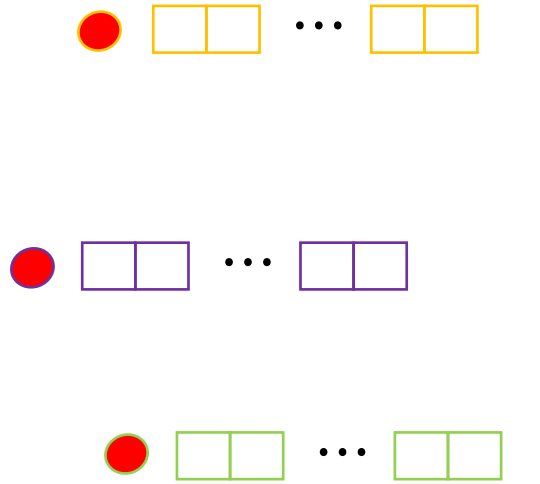
Slide arranged from Melzi and Marin

PoinNet layer

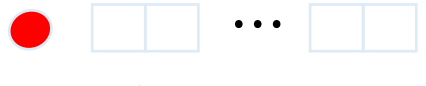
Runs PointNet on each ball. PointNet can handle a varying number of inputs.
Coordinates in the neighbor are expressed relative to the center.

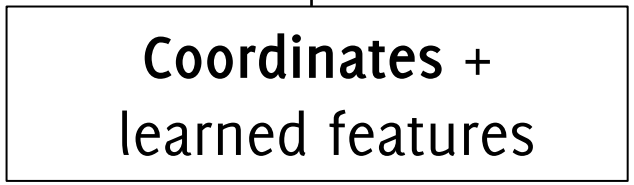






One feature vector (size $d + C'$) per query point N'



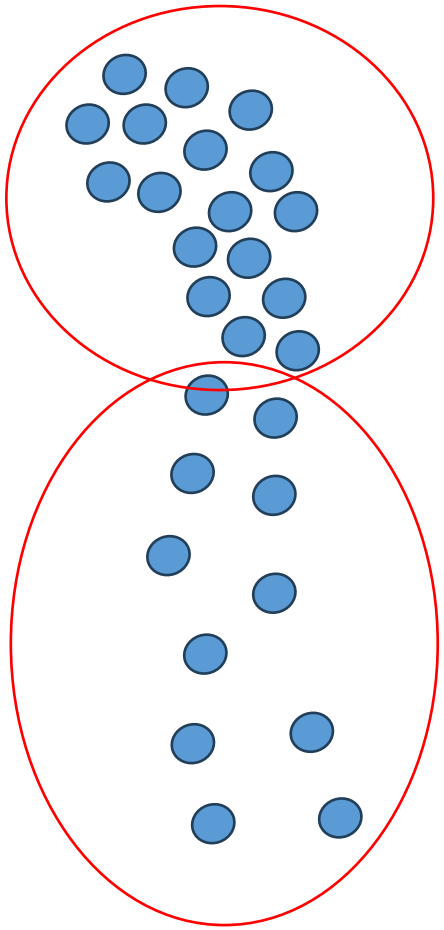


N' neighbors of K points maximum
$N' \times K \times (d + C)$

Slide arranged from Credits Melzi and Marin

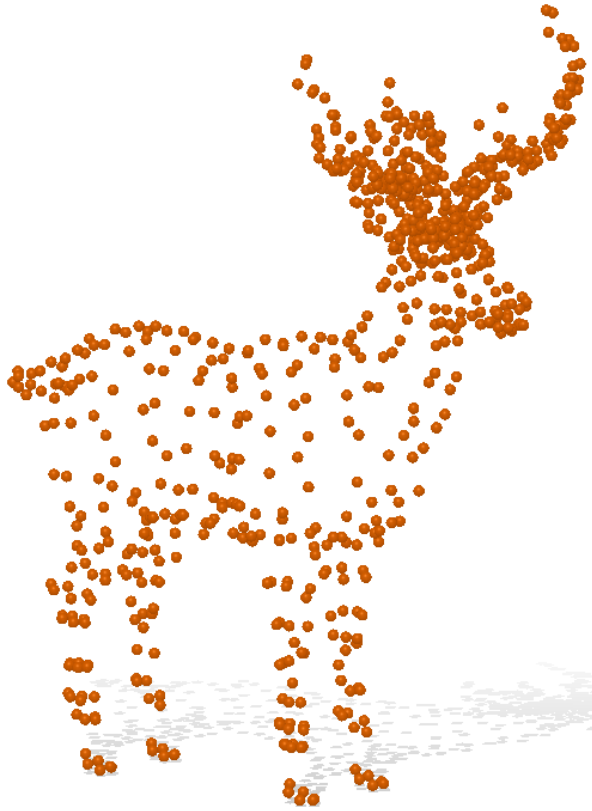
Different Densities

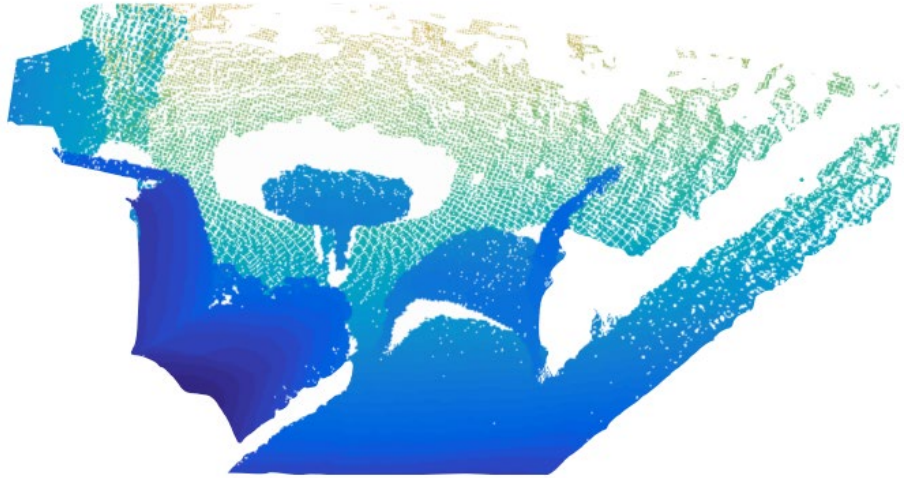
Real world point clouds are not uniform



dense

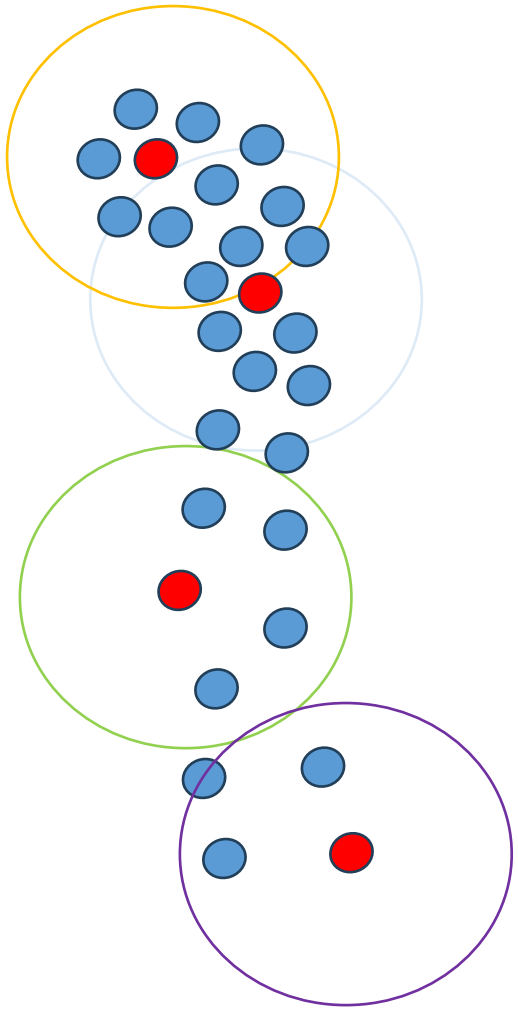
sparse

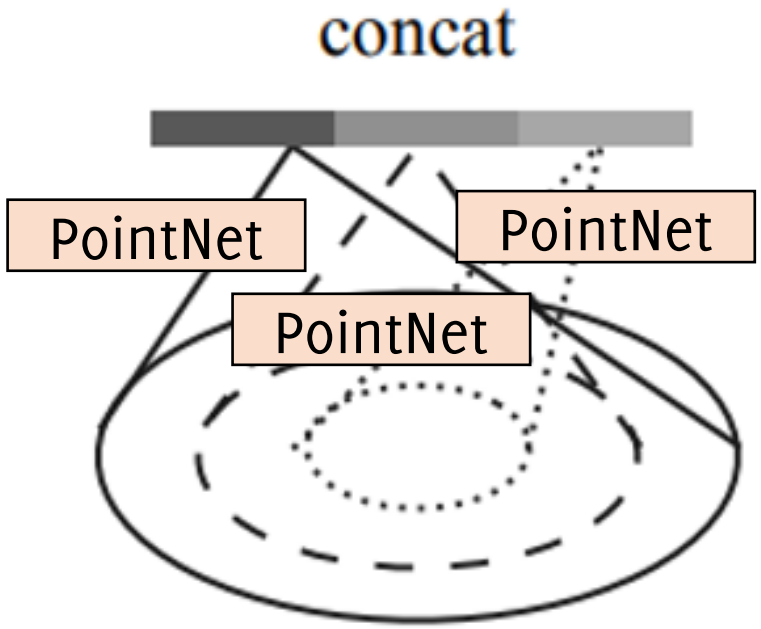




Dense adaptive layer: Multi-scale grouping (MSG)

The radius for grouping is fixed by multiple groups are selected per anchor.
Each group undergoes a PointNet layer and the output gets concatenated



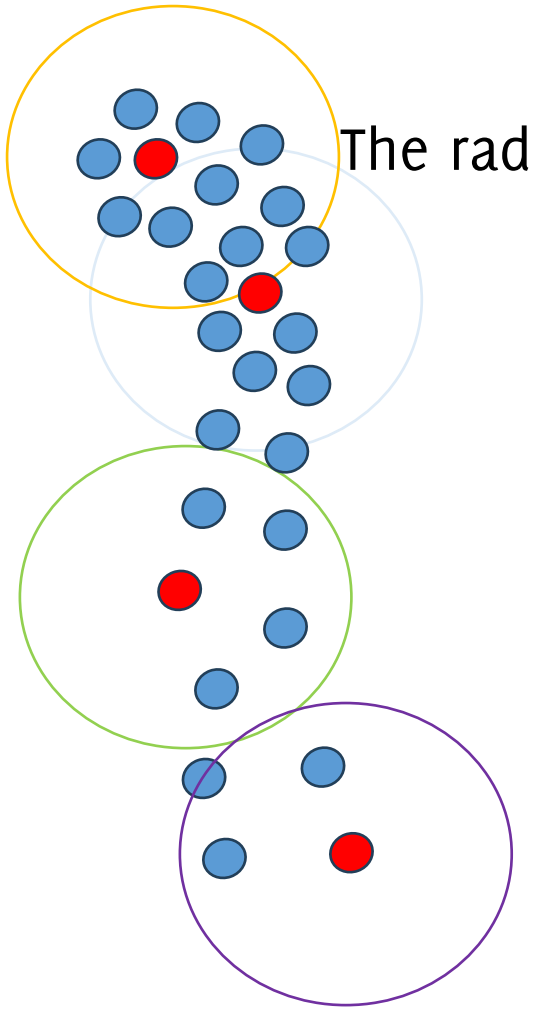


+ Random input dropout during training to show the network different densities

Not efficient: Large neighborhood for all centroids where to run PointNet

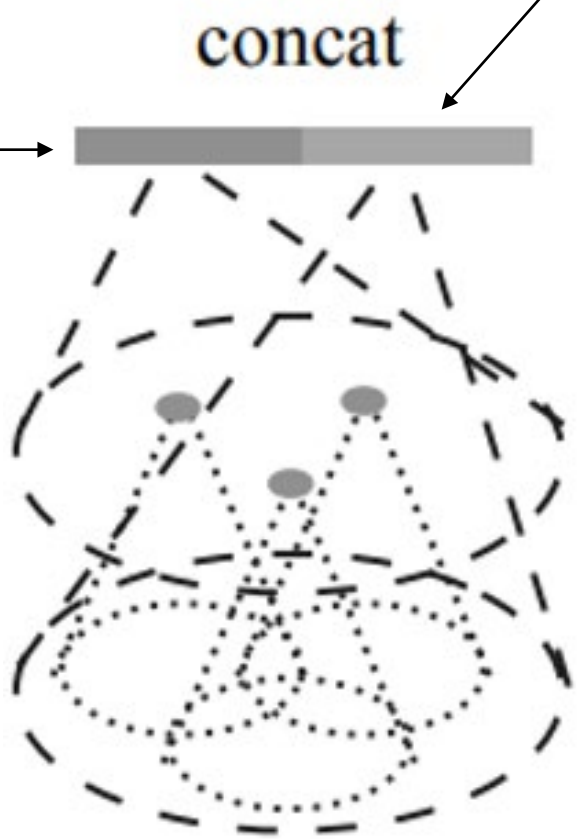
Dense adaptive layer: Multi-resolution grouping (MRG)

Concatenate features from the previous layers in two ways



The radius is fixed

Standard grouping of features from the last level



Apply PointNet to all the points that fall in the previous neighborhood

Segmentation in PointNet++

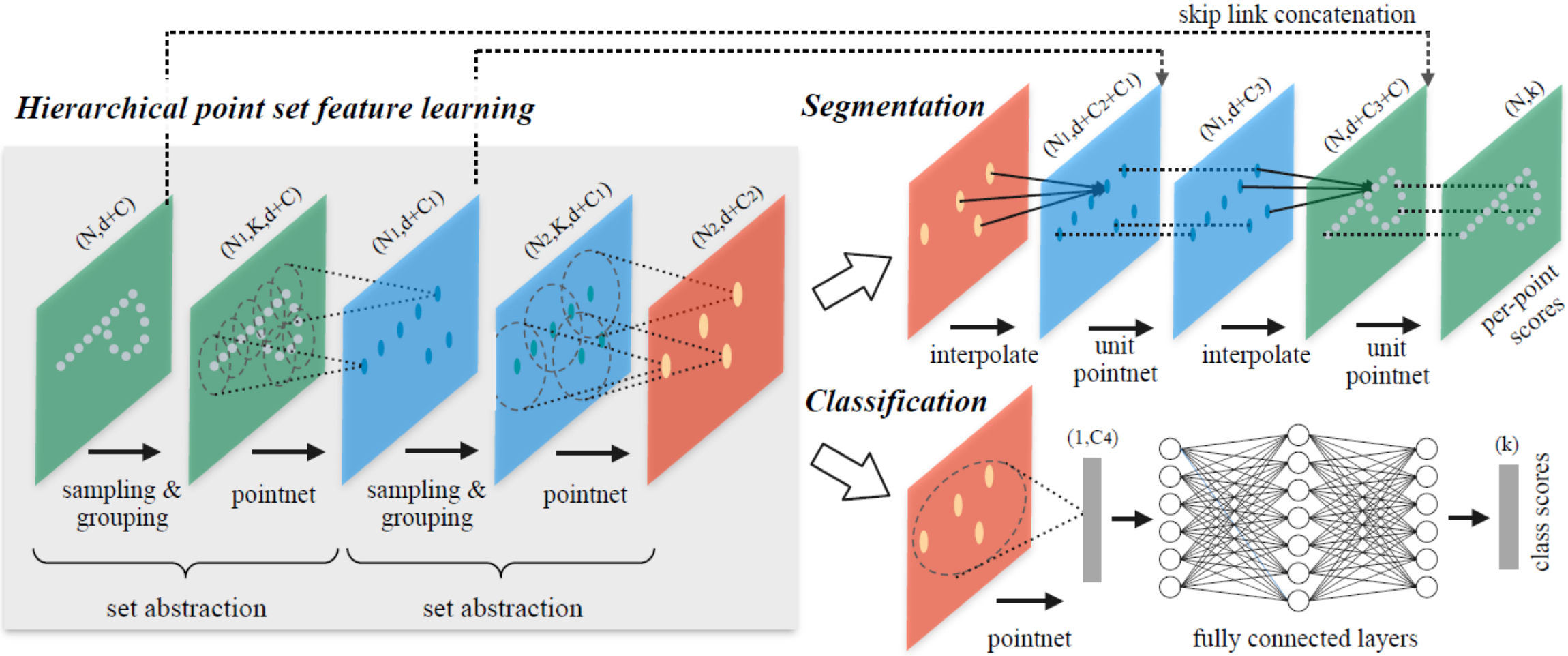
Issue when using PointNet++ for Segmentation: **the abstraction level discards points,** while we need dense predictions.

Design an “expanding path” of the network, to recover the discarded points in the contractive path (as in U-Net)

The issue is what feature to assign to “recovered” points?

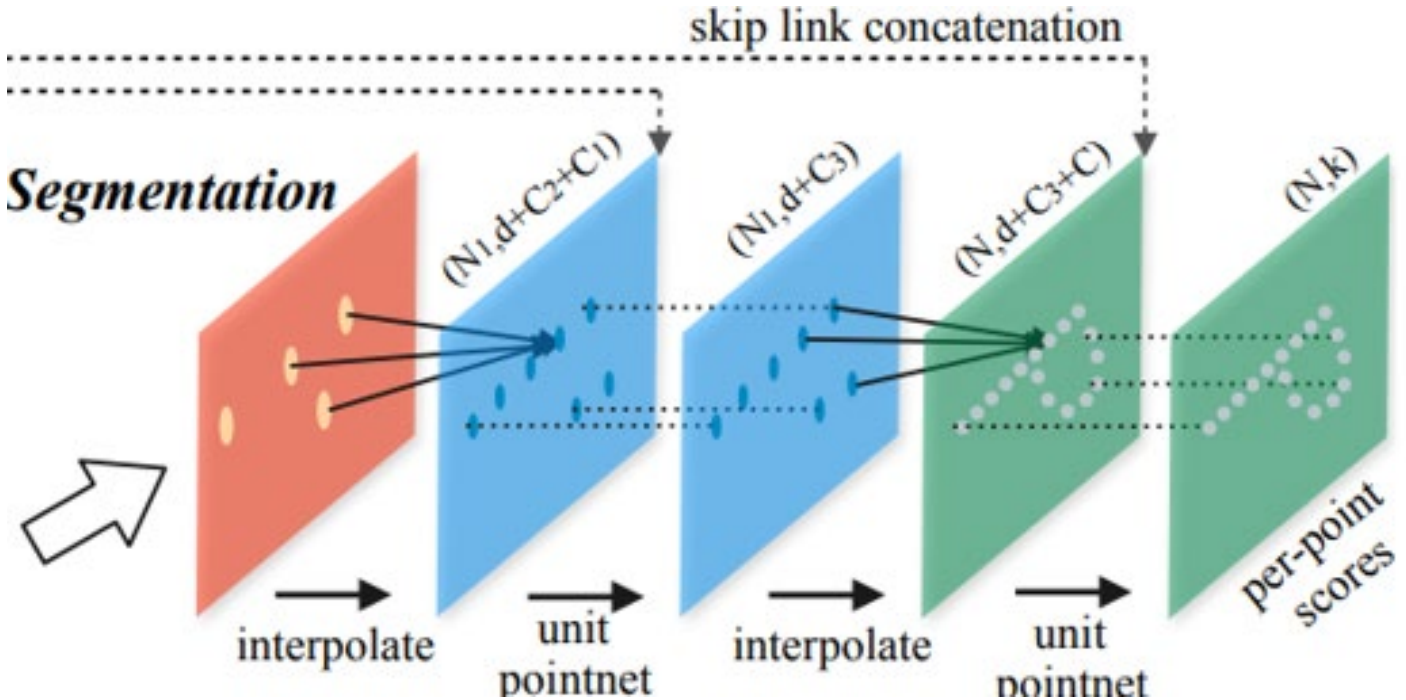
- **We recover coordinates** of the points from the contractive path via skip connections (retrieve the same original location before downsampling).
- **We propagate features from the** $(i-1)$ **layer to layer** i via linear interpolation of features extracted from the previous layer.

Segmentation in PointNet++



Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

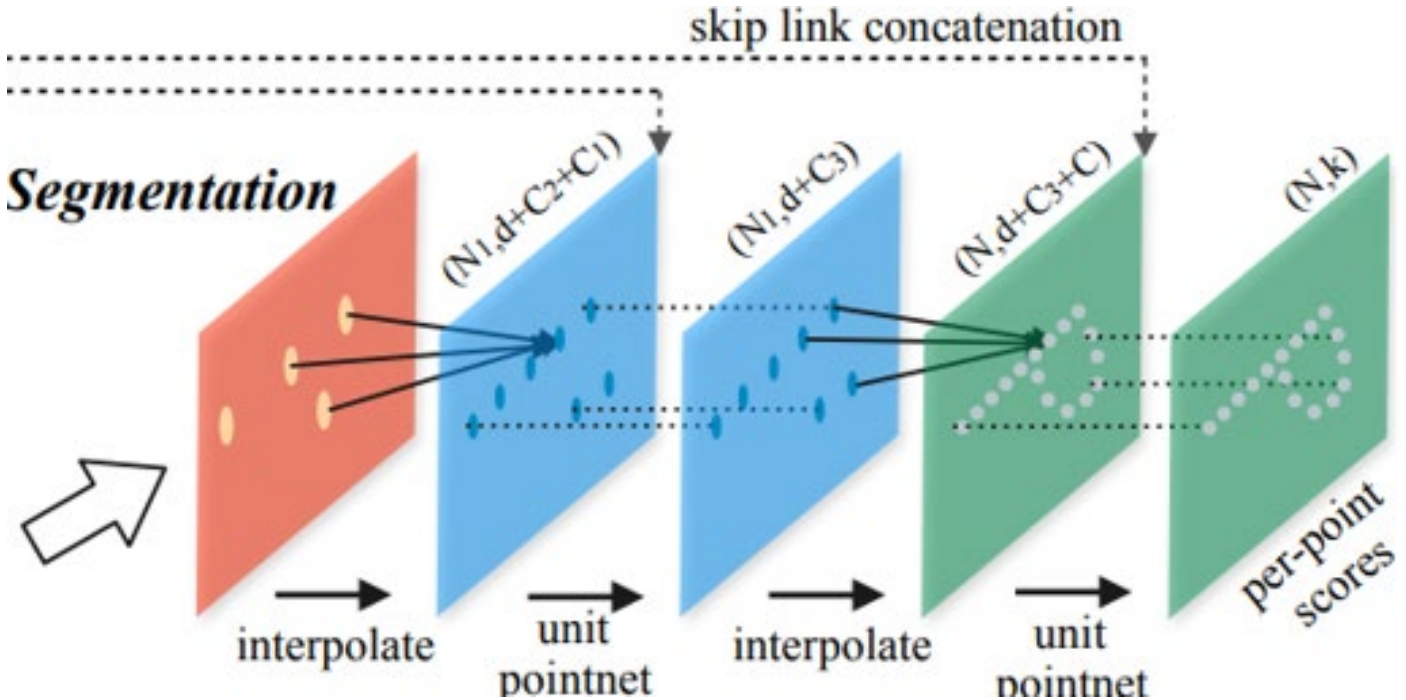
Interpolation



$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \quad \text{where} \quad w_i(x) = \frac{1}{d(x, x_i)^p}, \; j = 1, ..., C$$

x_i are the k nearest points ($k = 3$) and $f_i^{(j)}$ are its feature in the j-th dimension ($p = 2$)

Interpolation



Unit Pointnet (like a 1×1 conv layer) to learn the best merging strategy for these features (always point-wise)

Iterate the process until recovering features from all the original points

Point Convolutional Operators

Locality in PointNet and PointNet++

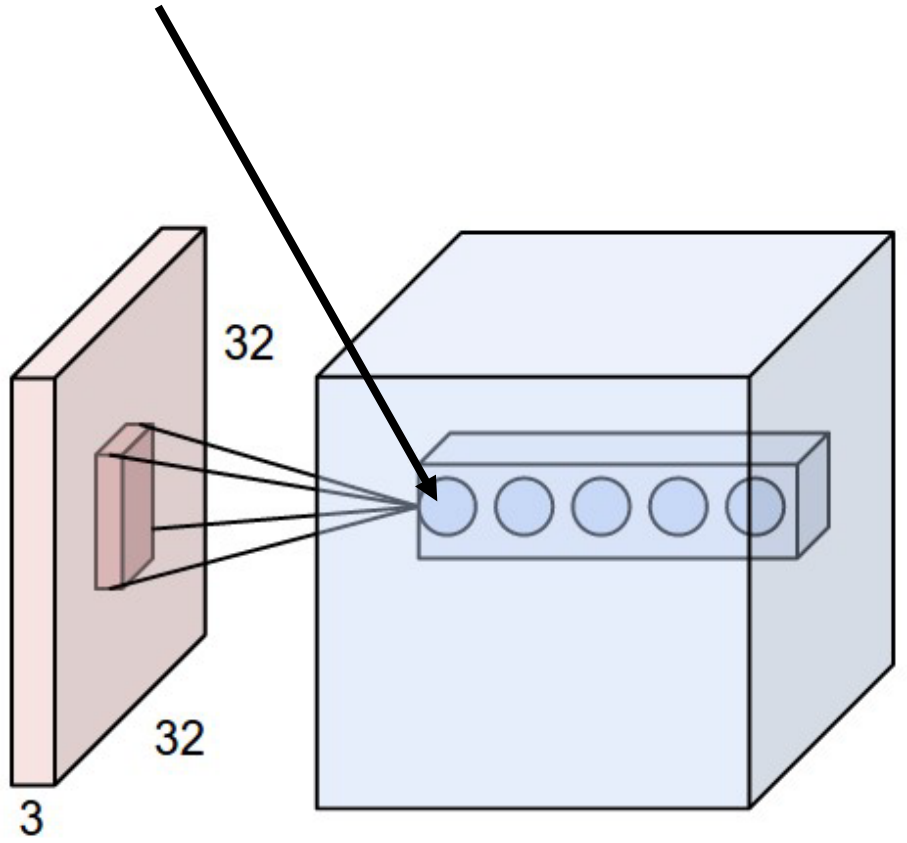
PointNet (and PointNet++) process each point from a PC (or from a subset of PC) independently, without considering the relative position of points.

$$\sum_{(u,v)\in U,k} w^1(u,v,k)\, x(r+u,c+v,k) + b^1$$

In a 2D convolution, locality is described by indexes. In a PC, locality is described by the values of point coordinates.

Rmk: Convolution can be defined on scattered points, as long as we can define the kernel g in any location

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) f_i$$



A channel-wise view of conv2D in (deep) CNN

Look at this expression "channel wise" for the $j-th$ filter

$$\sum_{(u,v)\in U}\sum_{k} w^{j}(u,v,k)\; x(r+u,c+v,k)+b^{1}$$

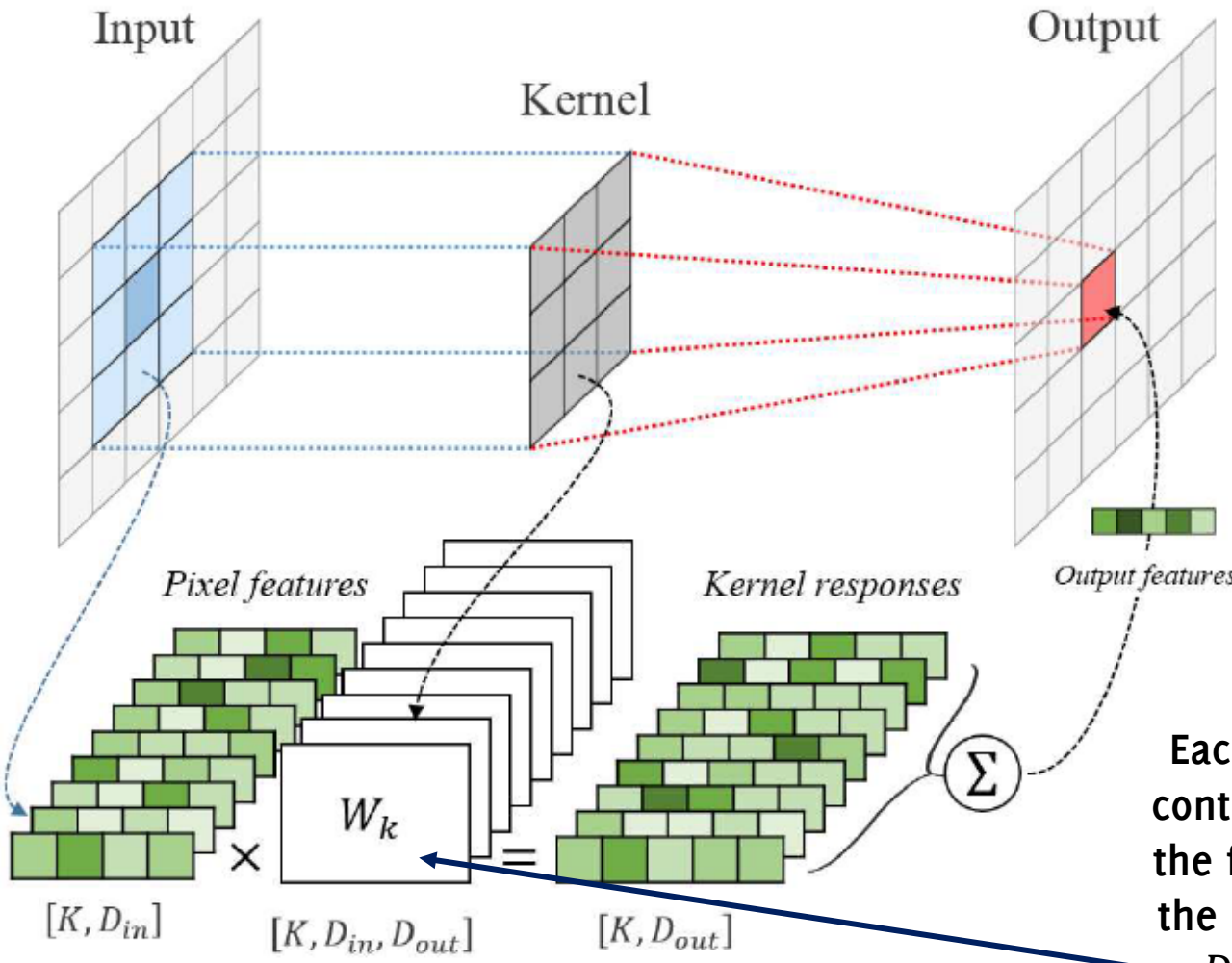
You can write the inner sum as an inner product

$$\sum_{(u,v)\in U} \boldsymbol{x}(r+u,c+v,:)^{\mathrm{T}}\cdot\ \boldsymbol{w^{j}}\ (u,v,:)+b^{1}$$

Where $\boldsymbol{x}(r+u,c+v,:)^{\mathrm{T}}$ and $\boldsymbol{w^{j}}(u,v,:)$ are vectors of $D_{in}\times 1$ the number of input channels

When you have D_{out} filters, then $\boldsymbol{W}(u,v,:)$ is a matrix of size $D_{in}\times D_{out}$

A channel-wise view of conv2D in (deep) CNN



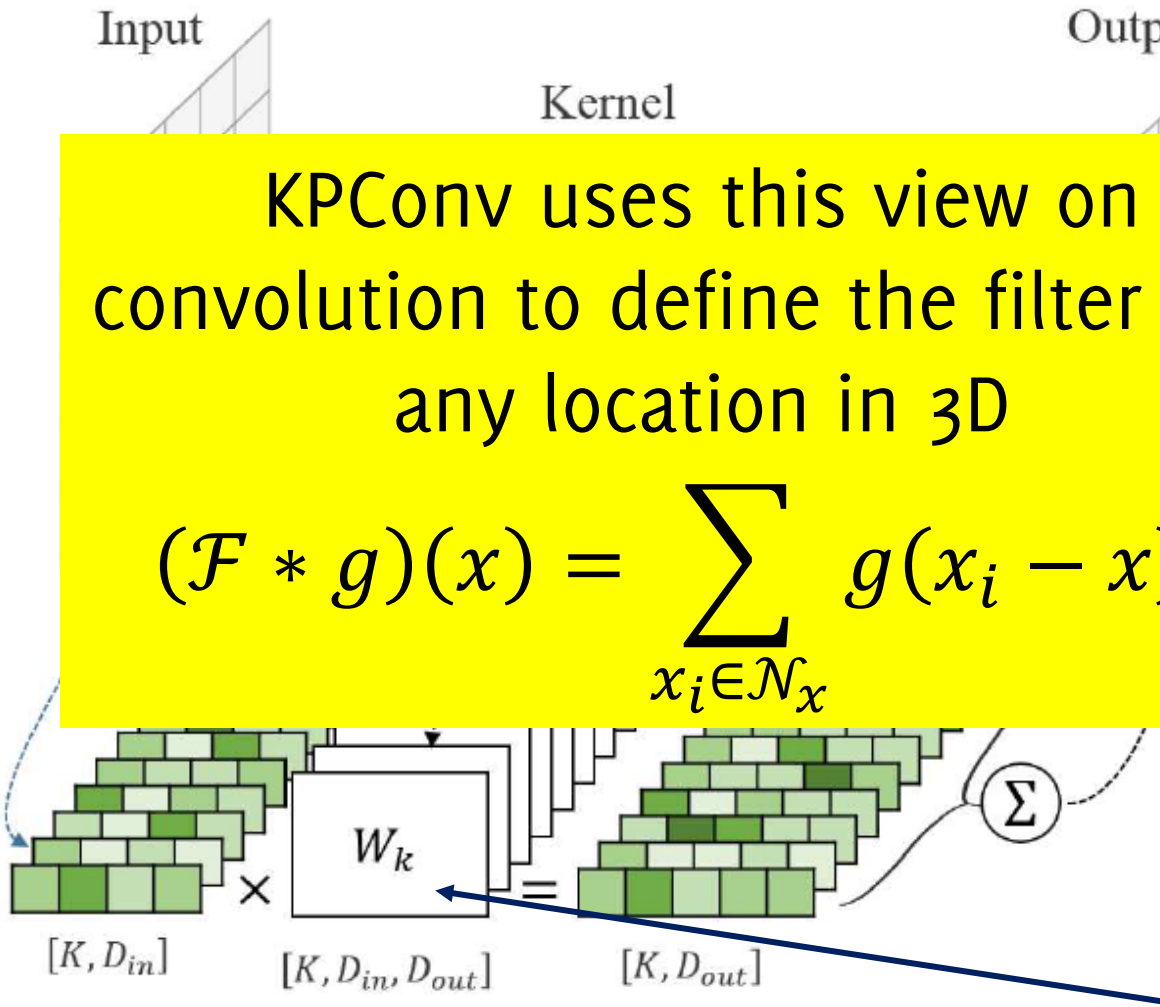
These vectors are then summed all over the neighbor ($K = 9$). The remaining D_{out} dimensional vector is the feature at the central pixel

These are channels (i.e. $D_{in} = 4$ features) associated to the $K = 9$ pixel locations in the neighbor (in blue)

Each column of the $K = 9$ matrices contains the weights for transforming the feature vector of a single point in the neighbor. Since we are extracting D_{out} features, we have a matrix (corresponding to using D_{out} filters)

Thomas H, Qi CR, Deschaud JE, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. ICCV 2019

A channel-wise view of conv2D in (deep) CNN



KPConv uses this view on convolution to define the filter g at any location in 3D

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) f_i$$

These vectors are then summed all over the neighbor ($K = 9$). The maining D_{out} dimensional ector is the feature at the central pixel

These are channels (i.e. $D_{in} = 4$ features) associated to the $K = 9$ pixel locations in the neighbor (in blue)

Each column of the $K = 9$ matrices contains the weights for transforming the feature vector of a single point in the neighbor. Since we are extracting D_{out} features, we have a matrix (corresponding to using D_{out} filters)

Thomas H, Qi CR, Deschaud JE, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. ICCV 2019

KPConv

$\mathcal{N}_x = \left\{ x_i \in \mathcal{P} \text{ s.t. } \left|\left| x_i - x \right|\right|_2 < r \right\}$

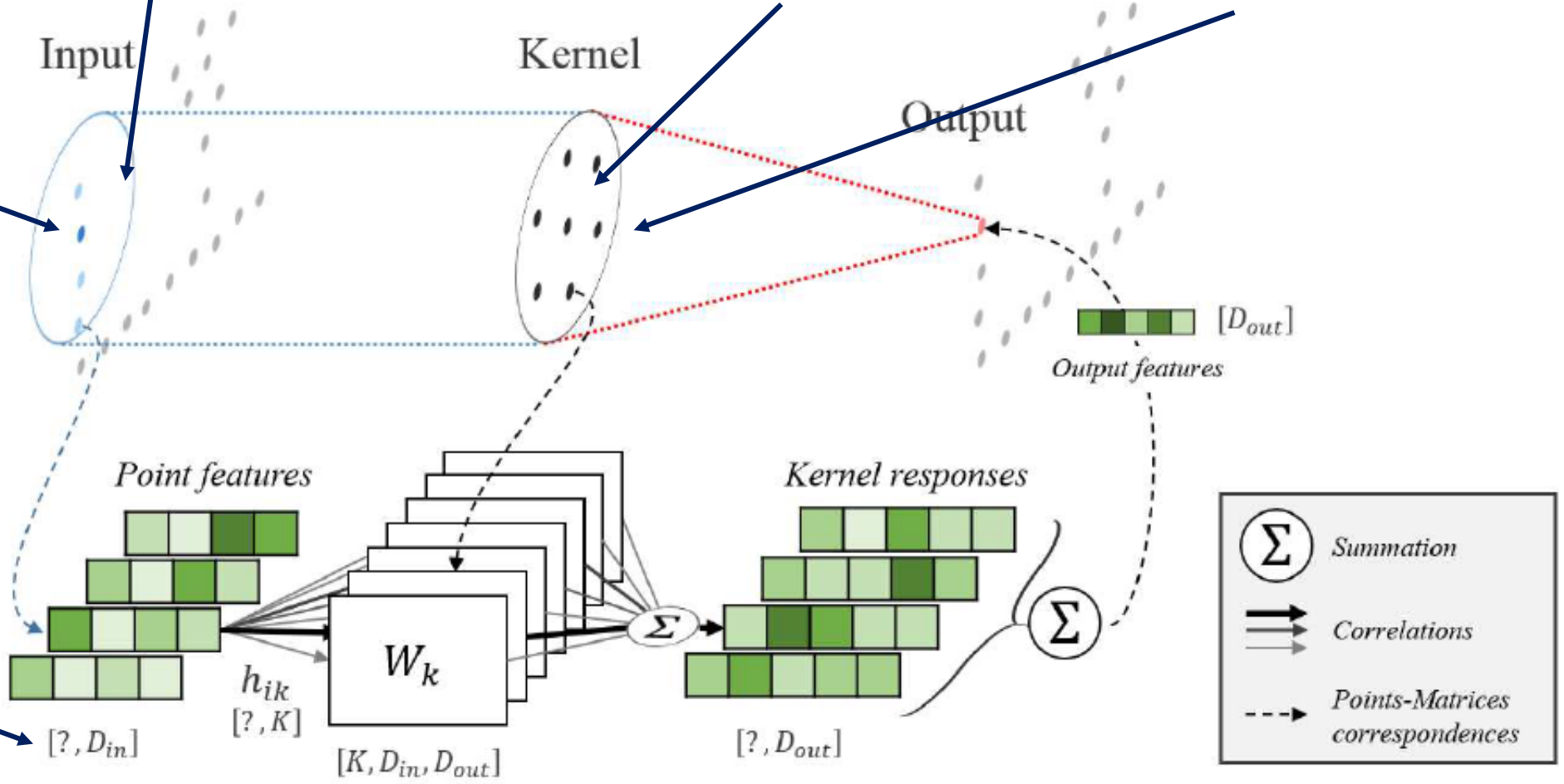
The neighbor of x

$K = 7$ Kernel points $\tilde{x}_i$
The filter is defined at these locations, which are fixed w.r.t. the center x

$\mathcal{B}_r = \left\{ y \in \mathbb{R}^3 \text{ s.t. } \left|\left| y \right|\right|_2 < r \right\}$

The centered neighbor where g is defined.

x point where to compute the output



These are features ($D_{in} = 4$) associated to the varying number of pixels in the neighbor

Thomas H, Qi CR, Deschaud JE, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. ICCV 2019

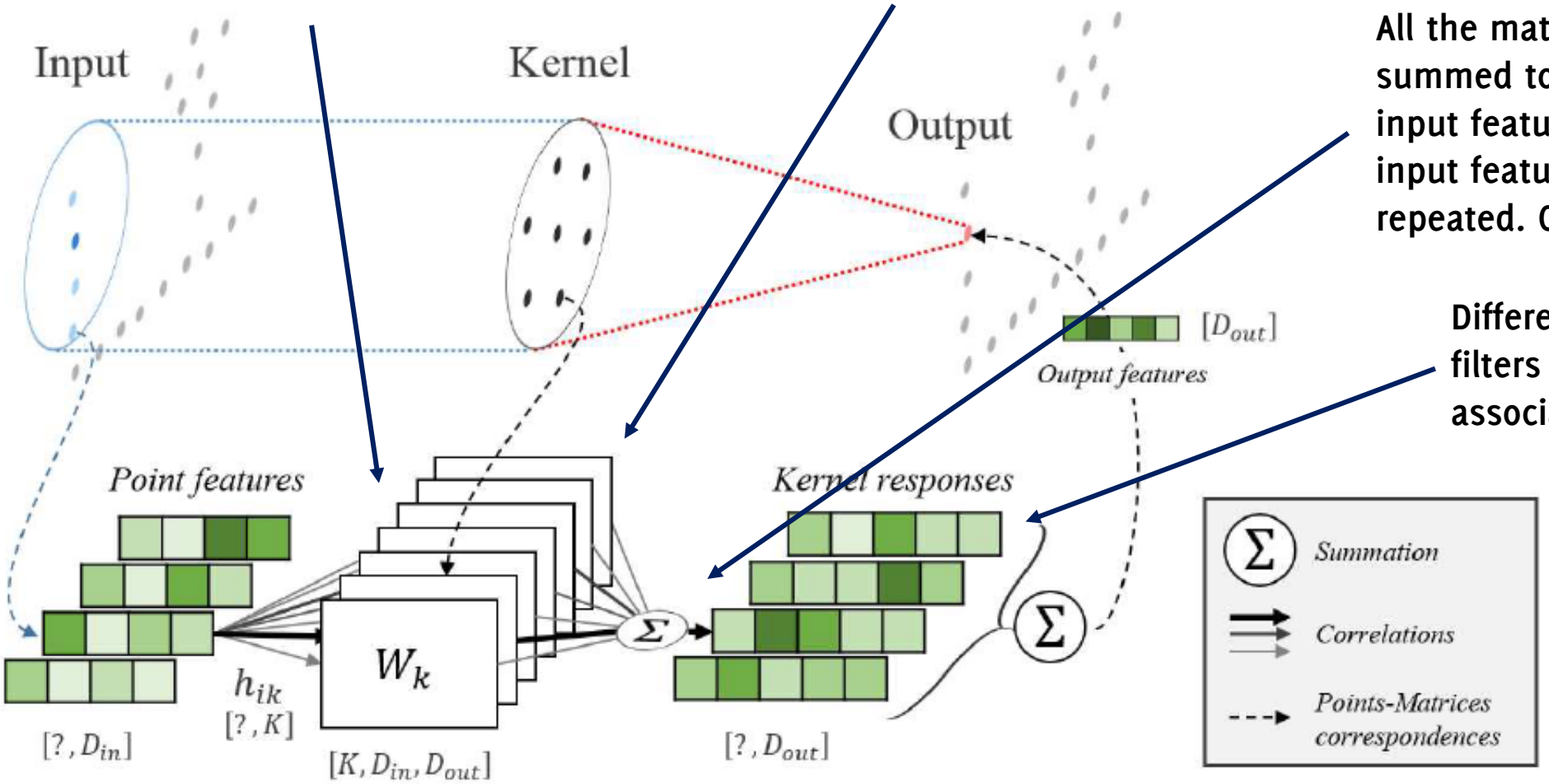
KPConv (cnt)

Each of the $K = 7$ kernel point is associated to a matrix.

For each point y_i, we compute the filter function

$$g(y_i) = \sum h(y_i, \tilde{x}_k) W_k$$

As a linear combination of all the matrices. Weights $h(y_i, \tilde{x}_k)$ encode the Euclidean distance between y_i and the kernel points.



All the matrices are scaled and summed together, multiplied to the input feature. When changing the input feature, the procedure is repeated. Only as the weights change

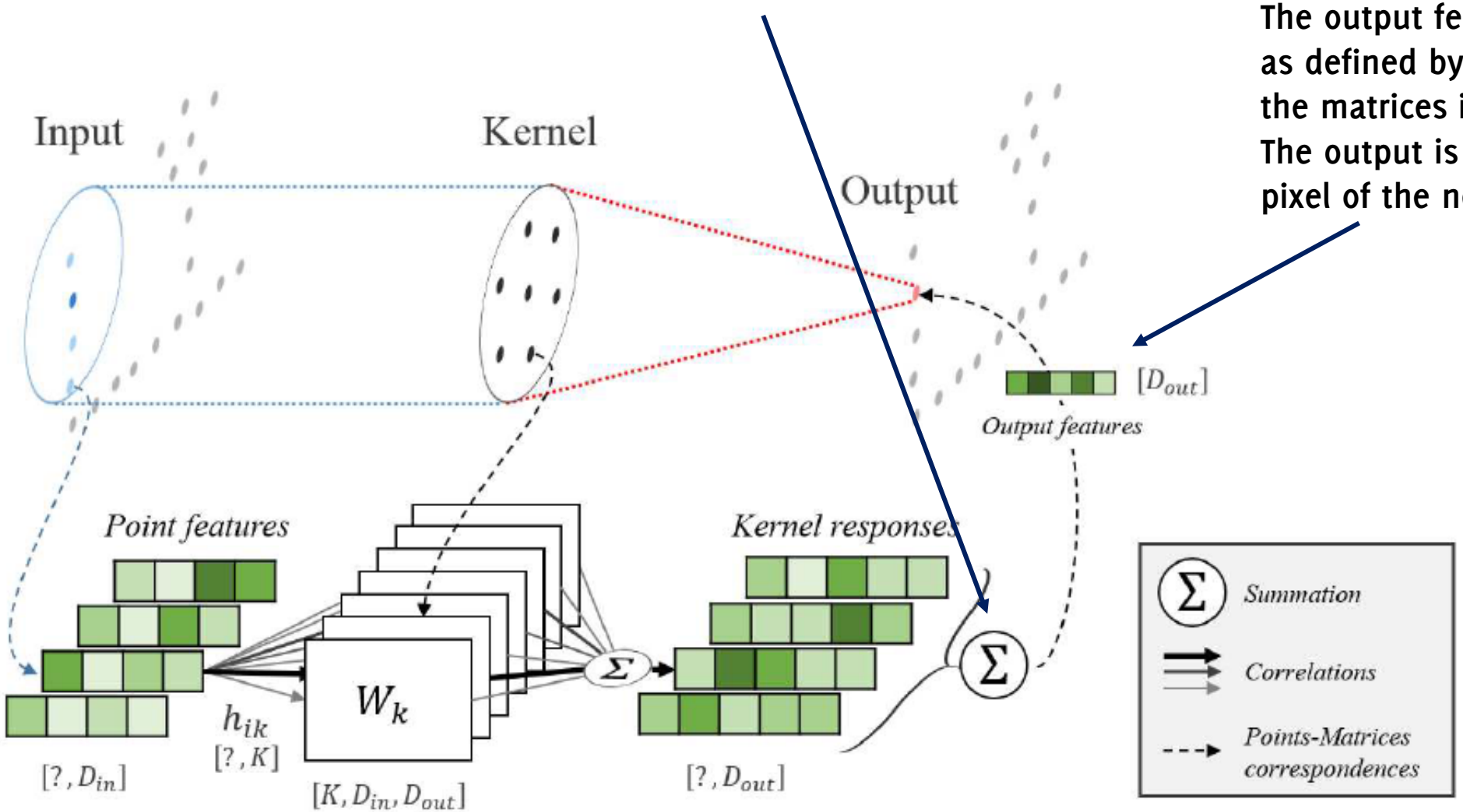
Different f_i have different filters $g(y_i)$ as they are associated to different locations y_i

Thomas H, Qi CR, Deschaud JE, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. ICCV 2019

KPConv (cnt)

There is an output feature vector for each input feature. These are all summed up to provide the layer output

The output feature is D_{out} dimensional, as defined by the number of columns of the matrices in the filter.
The output is associated to the central pixel of the negihborhood



Thomas H, Qi CR, Deschaud JE, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. ICCV 2019

KPConv

Input points $\mathcal{P} \in \mathbb{R}^{N \times 3}$ and their associated D-dimensional features $\mathcal{F} \in \mathbb{R}^{N \times D}$

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) f_i$$

Where g is the kernel, x is the output point,

$$\mathcal{N}_x = \left\{ x_i \in \mathcal{P} \text{ s.t.} \left|\left| x_i - x \right|\right|_2 < r \right\}$$

x_i is a point and f_i is the corresponding feature.

$\mathcal{N}_x$ is defined from a fixed radius ensures robustness to varying densities

Thomas H, Qi CR, Deschaud JE, Marcotegui B, Goulette F, Guibas LJ. Kpconv: Flexible and deformable convolution for point clouds. ICCV 2019

KPConv

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) f_i$$

The Kernel g is defined as a linear combination of inner products against a learnable set of K matrices

$$g(y_i) = \sum_{k \leq K} h(y_i, \tilde{x}_k) W_k$$

Matrices W_k are associated to K anchor points $\{\tilde{x}_k\}$ expressed in a relative position w.r.t. the center

The weights depend on the distance function h of inputs to the center

$$h(y_i, \tilde{x}_k) = \max\left(0, 1 - \frac{||y_i - \tilde{x}_k||}{\sigma}\right)$$

Link to Colab Sessions

https://drive.google.com/drive/folders/1GgRDVk4fTJ5CTfrcv1mAUS3fgIV152QC?usp=drive_link

Concluding Remarks

Concluding Remarks

On the one hand, truly 3D data representations offer several benefits

- They are compact and very informative

On the other hand, traditional ML models and NN (in particular CNN) are meant for «matrix data». Brute force attempts (voxelization / projection) are

- suboptimal in terms of information loss and space requirments
- with the increase of truly 3D dataset, they won't be anylonger convenient

3D data are ubiquitous and going to be increasingly used in diverse fields

3D data have opened (and will open) new challenges for modern Deep Learning and Comptuer Vision research