Geometric Deep Learning

Brown-Bag-Seminar 2019, IOS, Konstanz

Matthias Hermann, HTWG-Konstanz, Institute for Optical Systems
Promotionsstart: 11/2017

Advisors:
Prof. Dr. Georg Umlauf, HTWG-Konstanz, Institute for Optical Systems
Prof. Dr. Matthias O. Franz, HTWG-Konstanz, Institute for Optical Systems
Prof. Dr. Bastian Goldlücke, Universität Konstanz, Computer Vision/Image Analysis

BMBF-FZ: 13N14540

Contents

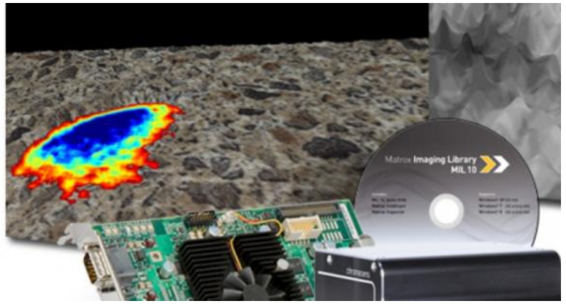
- Why geometric deep learning?
- Limits of traditional Convolutional Neural Networks
- Machine Learning on non-Euclidean domains
  - Meshes a.k.a 2-manifolds
  - General graphs
  - Point clouds a.k.a. Sets
- A Common Framework

A lot of visual data is not flat

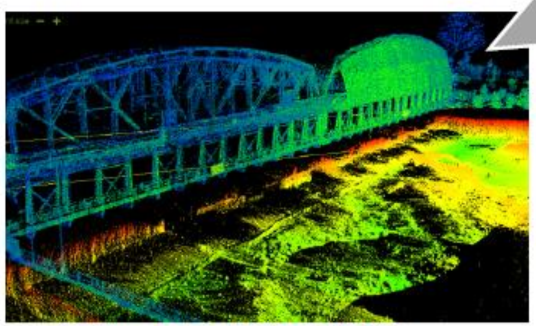
Inpsection

Robotics

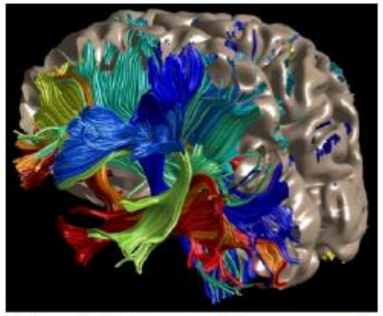
Augmented Reality



Topography



Autonomous driving

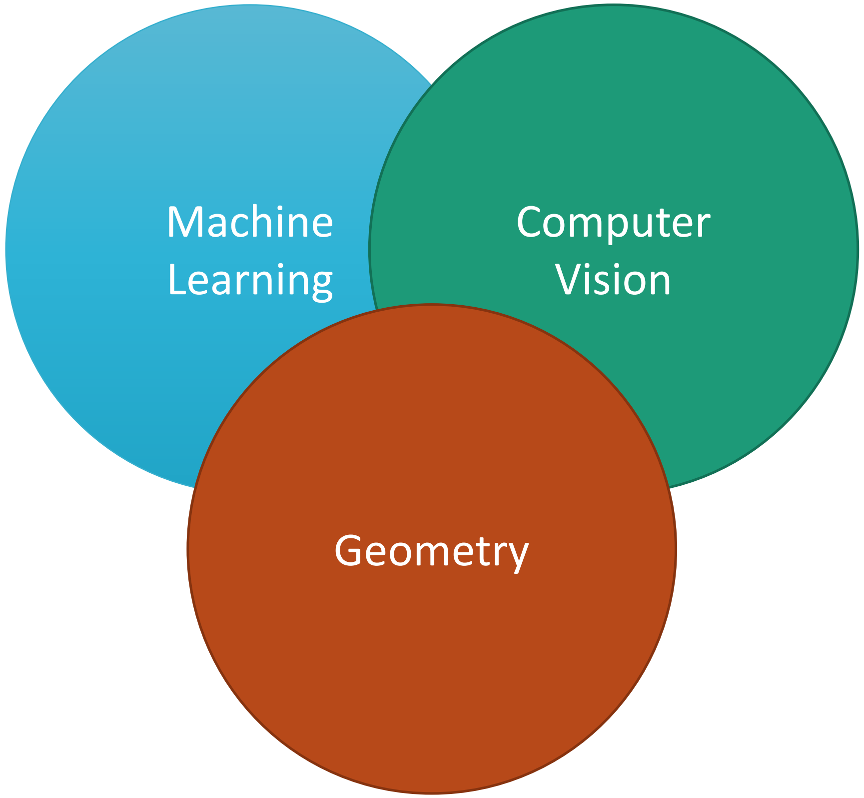


Medical Image Processing

Credits to Hao Su, Stanford 2017

The surge of geometric deep learning

- Started 2015 with big datasets ShapeNet & ModelNet
- Very active due to huge industry interests



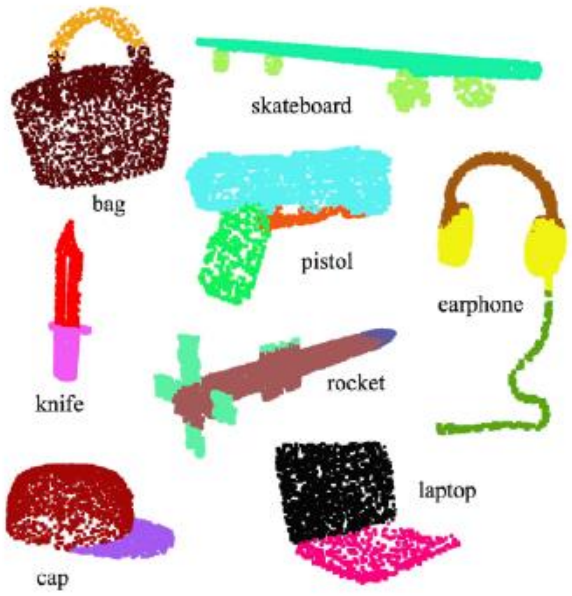
Industries are:

- Robotics
- 3d scanning
- 3d geometric modelling
- Autonomous driving
- Augemented reality
- Virtual reality
- Topography
- Etc.

3d deep learning tasks

3D geometry analysis

Classification



Parsing
(object/scene)

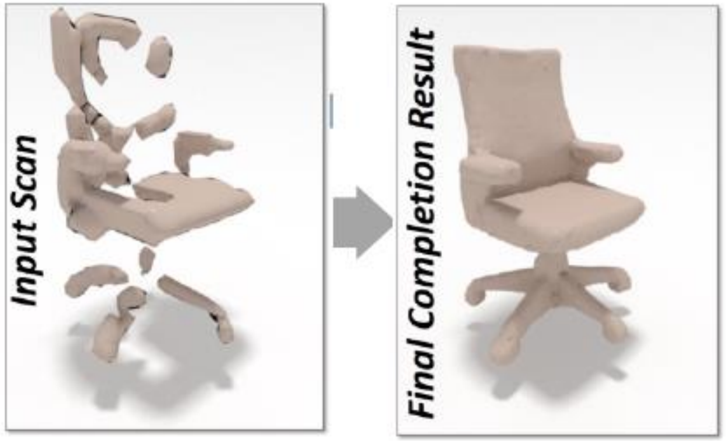
Correspondence

Credits to Hao Su, Stanford 2017

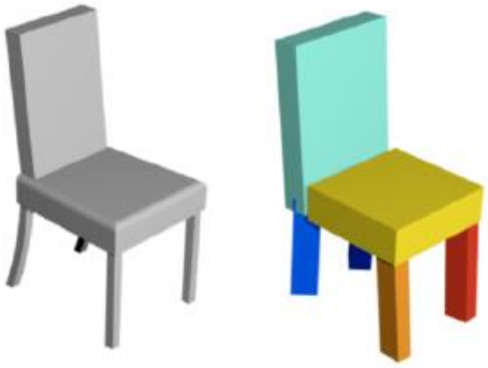
3d deep learning tasks

3D synthesis

Monocular
3D reconstruction



Shape completion



Shape modeling

Credits to Hao Su, Stanford 2017

3d deep learning tasks

3D-assisted image analysis

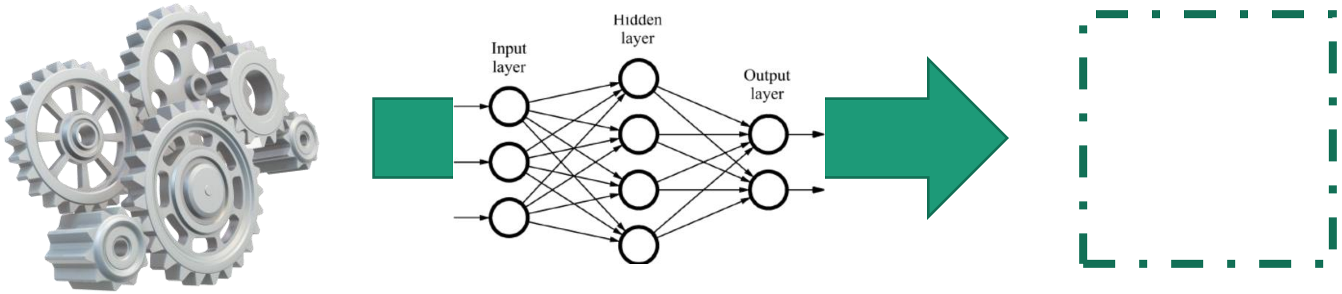
Cross-view image retrieval

Intrinsic decomposition

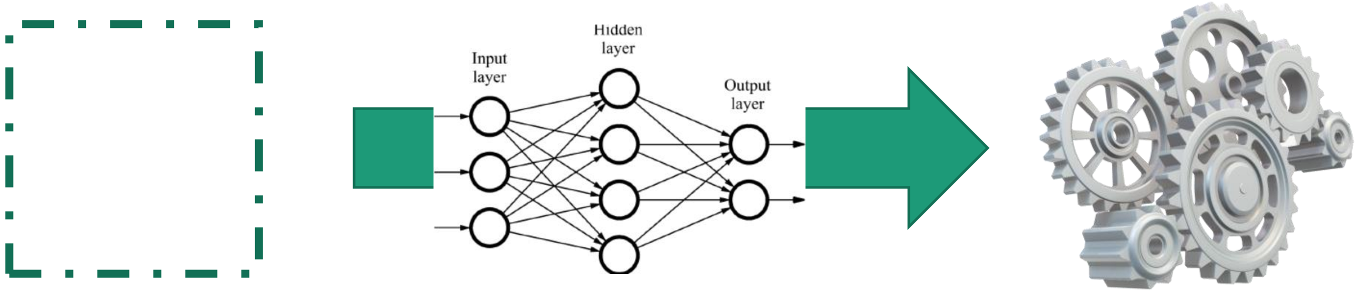
Credits to Hao Su, Stanford 2017

The data vs. the network

Geometry analysis



Geometry synthesis



Convolution Neural Networks. Where is the problem?

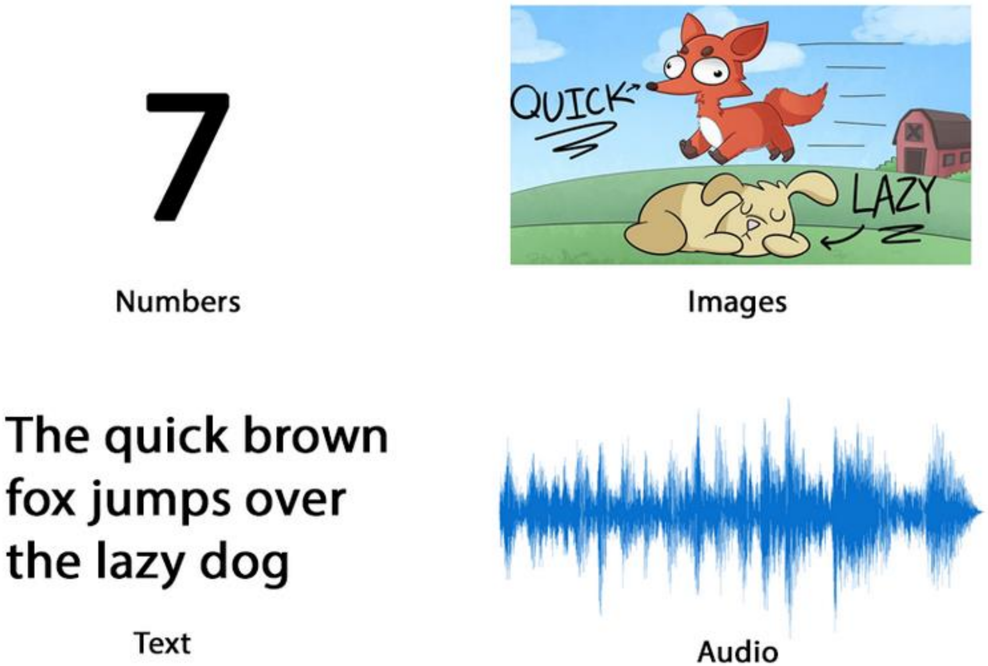
Images have a very easy regular data structure!

- Unique representation
  → easy (e.g. flatten())
- Vector representation
  → easy (e.g. flatten())
- Distance and dot product
  → easy (e.g. $||X - Z||_2$ $or < X, Y >$)
- Functional representation
  → easy (f: $[0,1]^2$ → $\mathbb{R}$)
- Subsampling
  → easy (e.g. X[0::2])

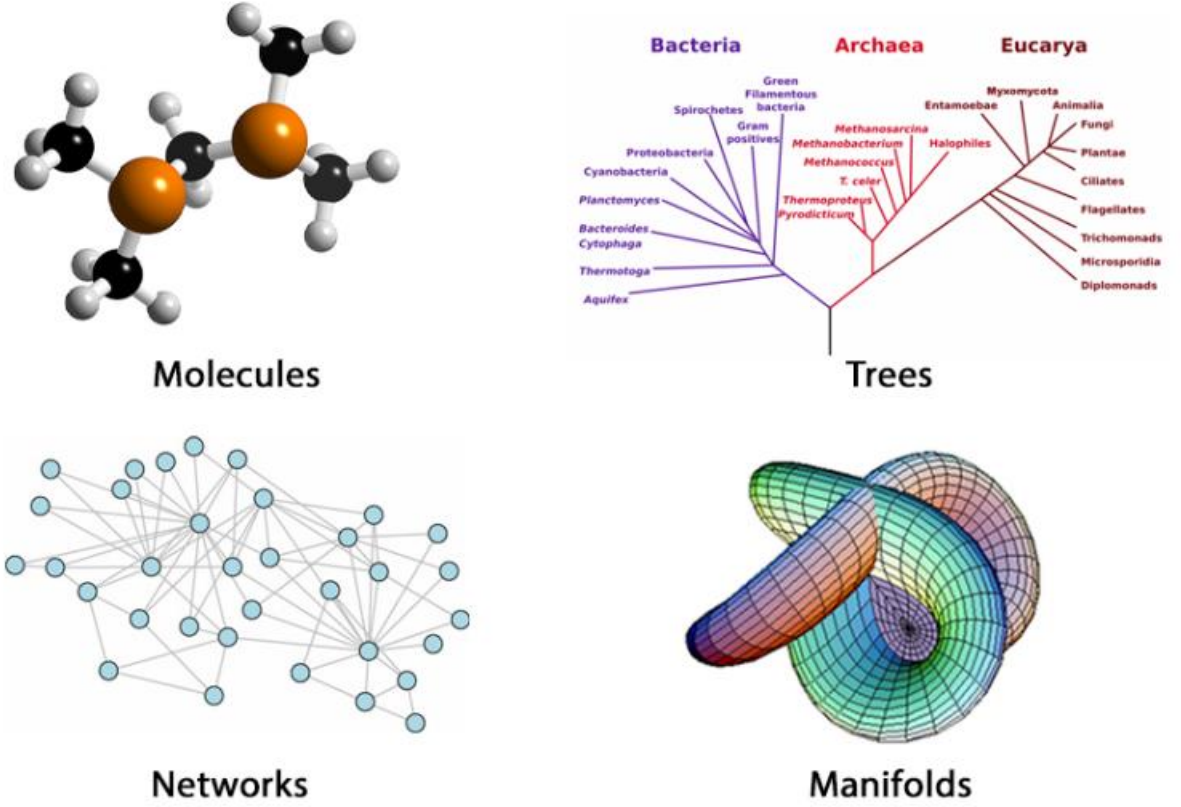
1	44	33	12	20	23	35	14
51	16	40	32	46	48	28	17
29	60	3	63	49	55	36	7
52	22	26	41	38	10	61	53
2	24	19	11	34	43	5	8
57	9	37	42	25	21	27	18
30	56	50	64	4	59	6	13
58	47	45	31	39	15	62	54

Euclidean vs. Non-Euclidean data

Images, text, audio, and others can be treated as Euclidean data (little inductive bias).



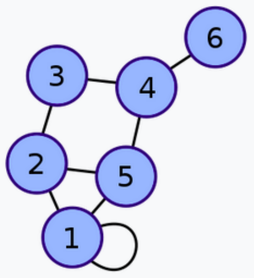
Non-Euclidean data can represent more complex items and concepts (extreme inductive bias).



Graph representation

Set of points + adjacency matrix + optional vertex attributes

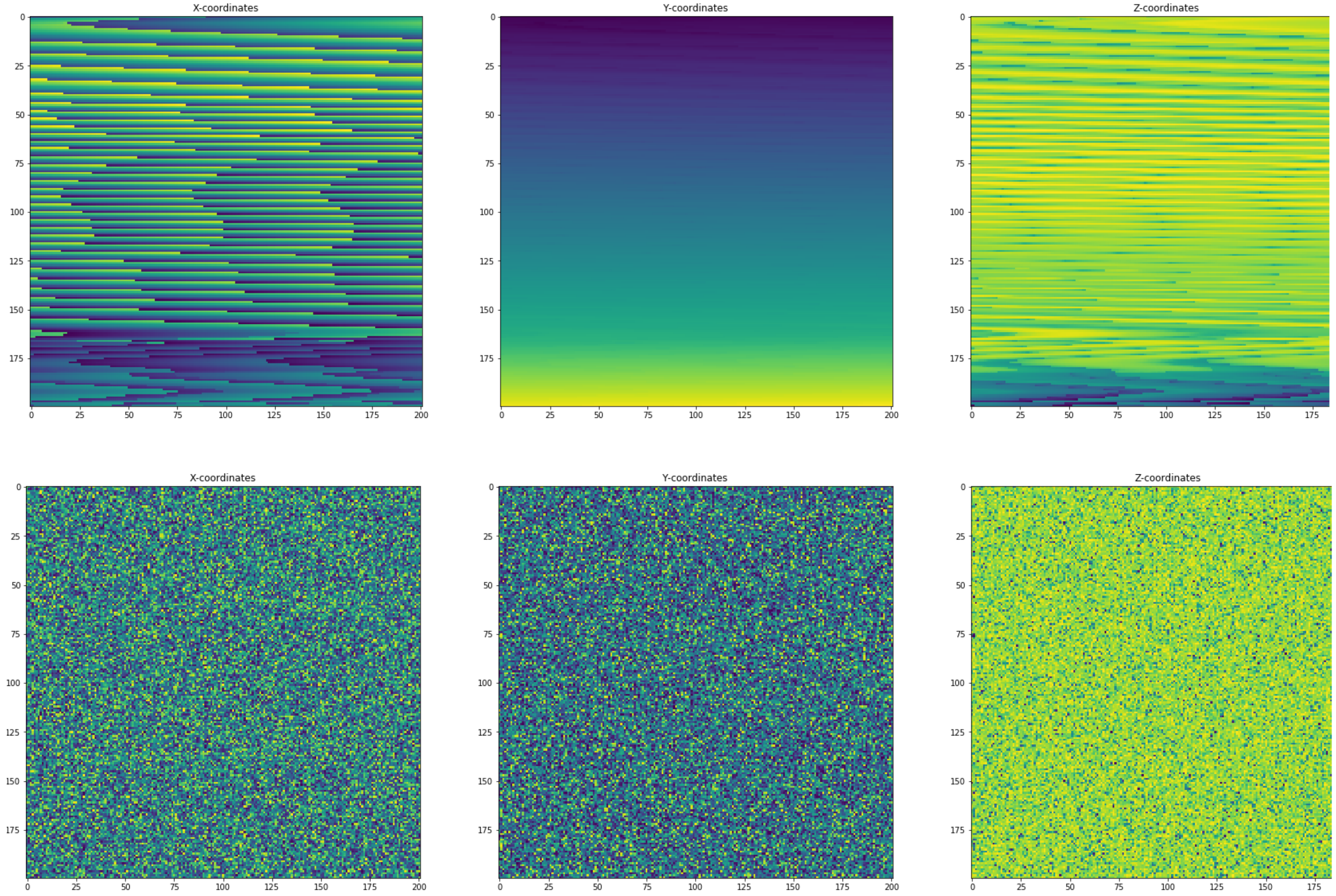
-0.180226841	0.360945118	-1.120304970
-0.180226841	1.559292118	-0.407860970
-0.180226841	1.503191118	0.986935030
-0.180226841	0.360945118	1.29018350
-0.180226841	-0.781300882	0.986935030
-0.180226841	-0.837401882	-0.407860970
-0.180226841	0.360945118	-2.206546970
-0.180226841	2.517950118	-0.917077970
-0.180226841	2.421289118	1.572099030
-0.180226841	-1.699398882	1.572099030
-0.180226841	-1.796059882	-0.917077970

Labeled graph	Adjacency matrix
	$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ Coordinates are 1–6.

-0.180226841	0.360945118	-1.120304970
-0.180226841	1.559292118	-0.407860970
-0.180226841	1.503191118	0.986935030
-0.180226841	0.360945118	1.29018350
-0.180226841	-0.781300882	0.986935030
-0.180226841	-0.837401882	-0.407860970
-0.180226841	0.360945118	-2.206546970
-0.180226841	2.517950118	-0.917077970
-0.180226841	2.421289118	1.572099030
-0.180226841	-1.699398882	1.572099030
-0.180226841	-1.796059882	-0.917077970

Adjacency matrix is either given or induced by metric (e.g. through k-nearest neighbors search)!

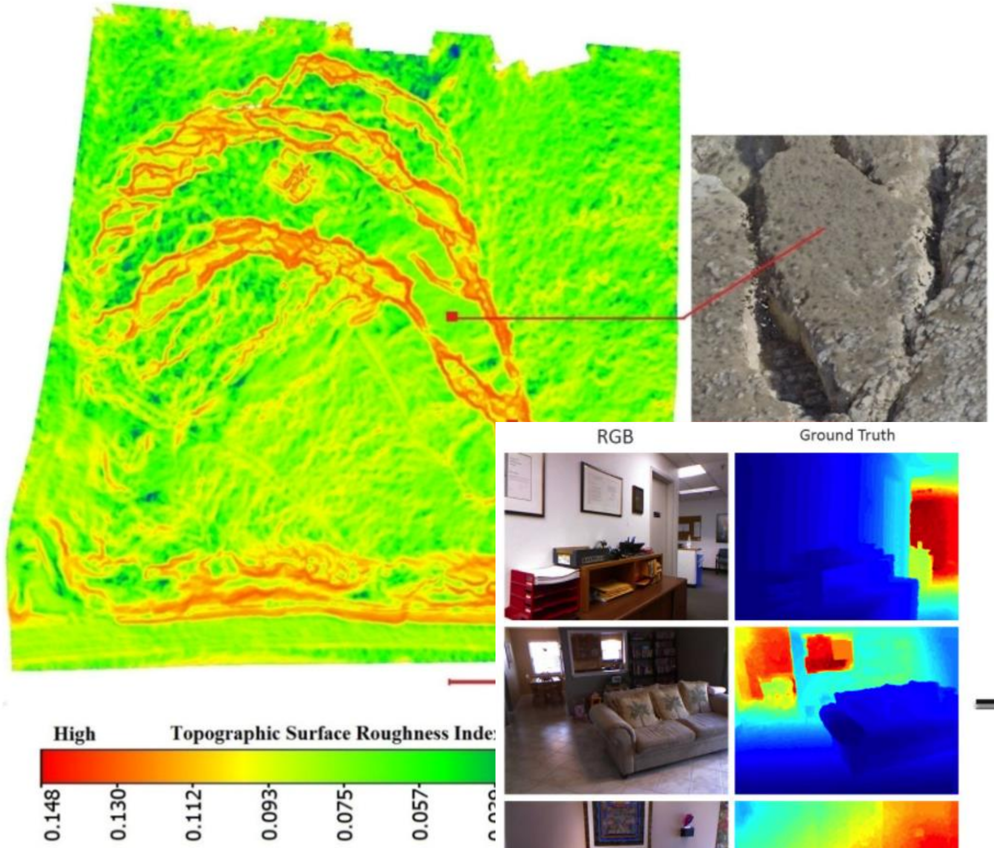
Order matters: Stanford bunny example

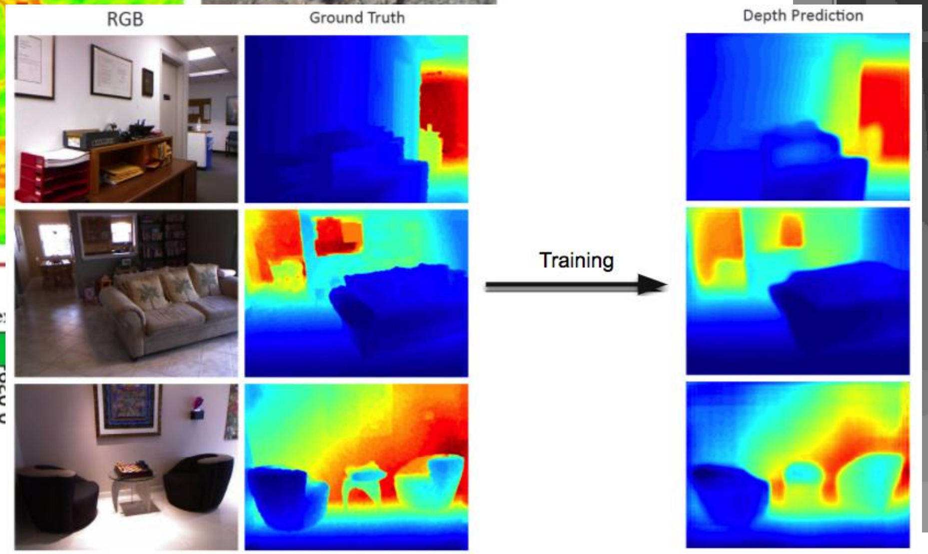


2d coordinate maps of the Stanford bunny in scanning order (top) and arbitrary order (bottom).

In unstructured 3d data order
Is important.

Statistics matters: Topographic and depth maps





Such maps are structured and look like images, but have much rougher local structures and different image statistics compared to natural images.

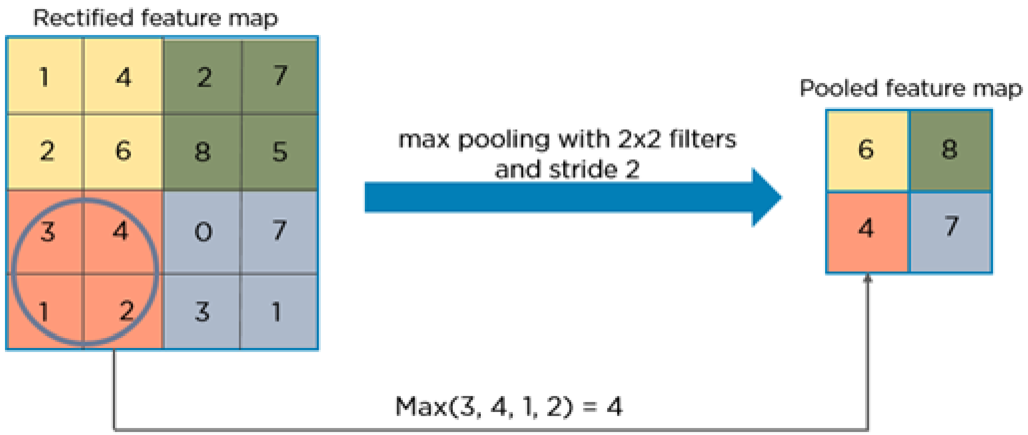
Credits: https://www.mdpi.com/remotesensing/remotesensing-08-00095/

Convolution Neural Networks on grids

Convolution

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

Pooling



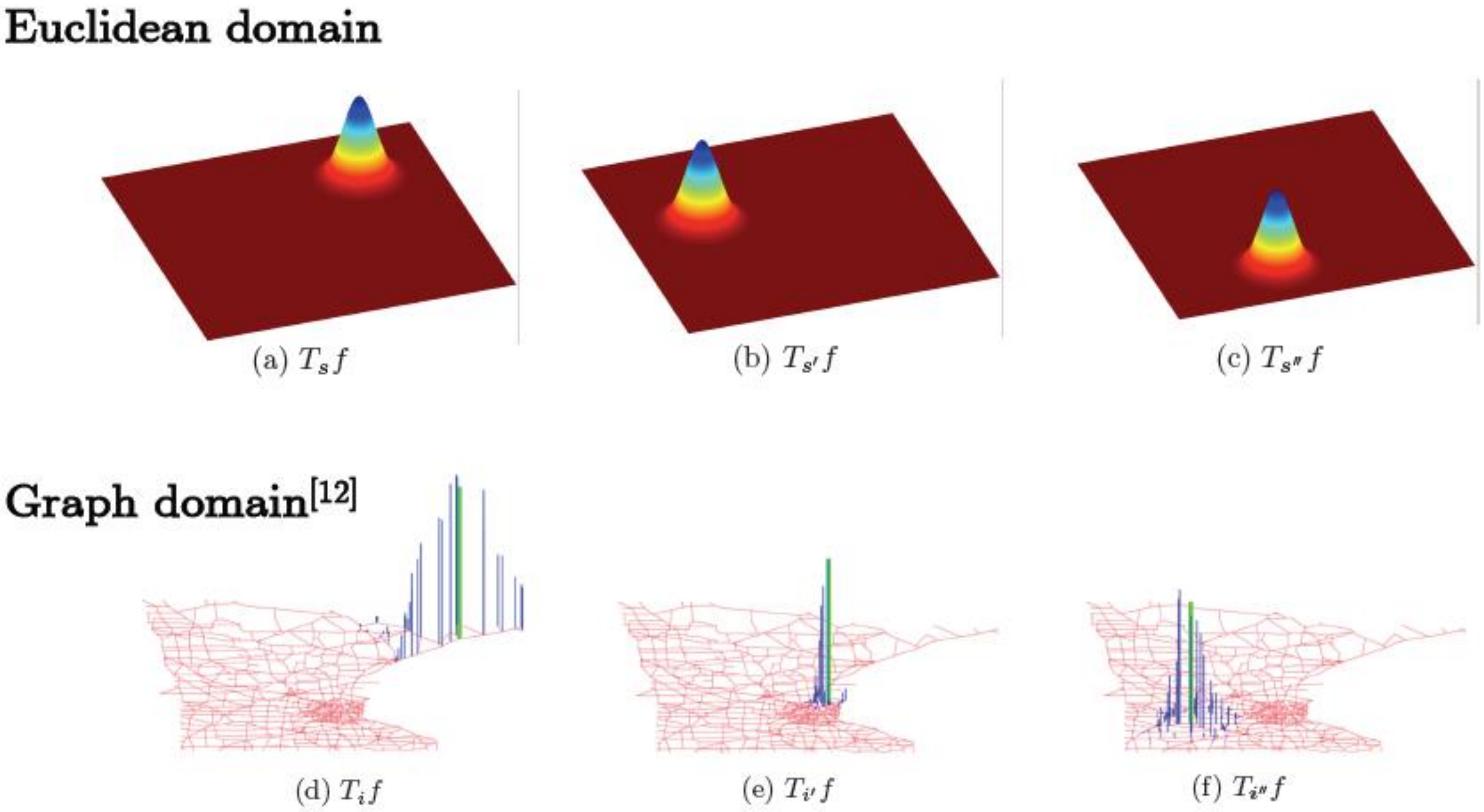
$$(f * g)[x] = \sum_{-M}^{M} f[n-m]g[m]$$

Both operations need an **underlying structure** like defined neighborhoods, directions, order, translations and common vector space!

→ Image are **flat**, i.e. have a flat metric (not curved)

→ Images have a **homogenous topology** (every pixel has the same neighborhood)

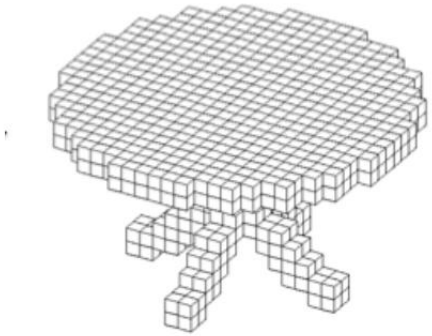
No shift invariance on graphs

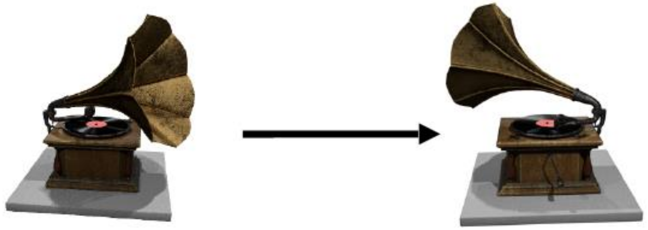


Credits to Shuman et. al., 2016

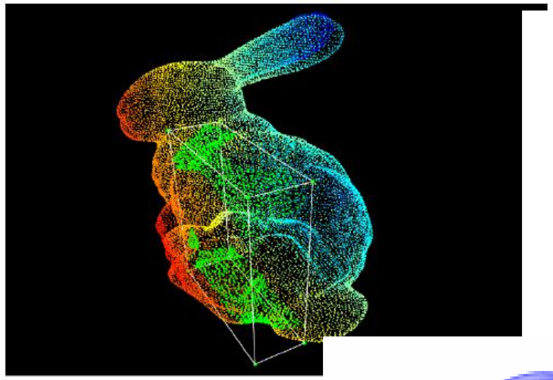
Different 3d data representations

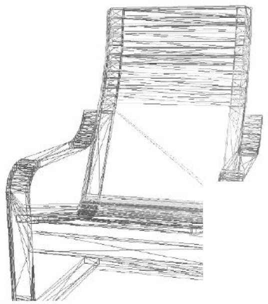
- **Rasterized form (regular)**
  - Multi-view RGB(D) images
  - volumetric

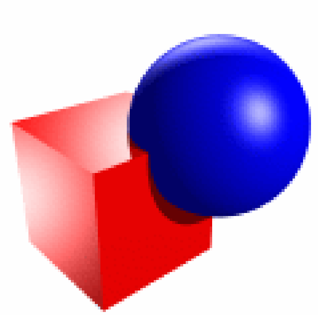


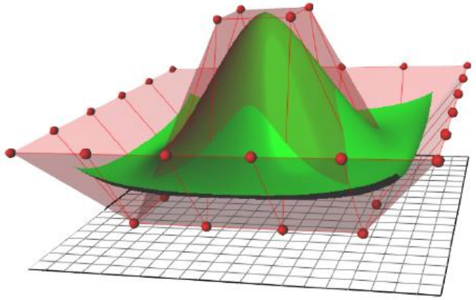


- **Geometric form (irregular)**
  - Polygon mesh / wire frame
  - Point cloud
  - Parametric surfaces
  - Primitive based CAD (CSG)







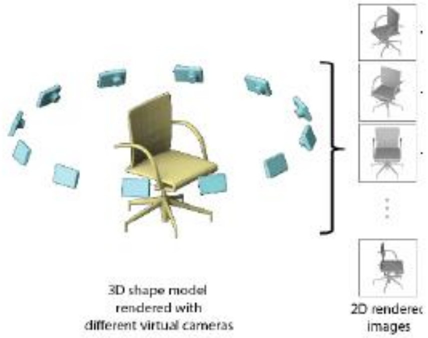


Different 3d data representations

- **Rasterized form (regular)**
  - Multi-view RGB(D) images → Standard convolution and pooling operator
  - volumetric → Discrete 3d convolution and pooling operator

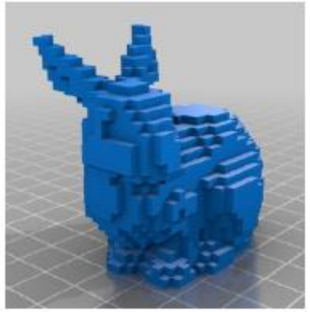
- **Geometric form (irregular)**
  - Polygon mesh / wire frame → e.g. no homogenous neighborhood
  - Point cloud → e.g. no canonical order
  - Parametric surfaces → e.g. no unique parametrization
  - Primitive based CAD (CSG) → e.g. no homogenous neighborhood

Existing 3d learning algorithms



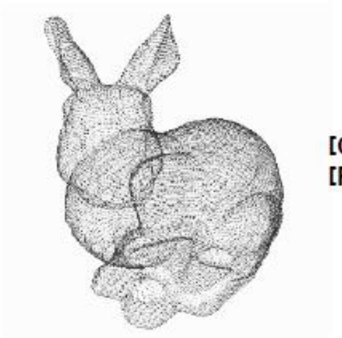
[Su et al. 2015]
[Kalogerakis et al. 2016]
...

Multi-view



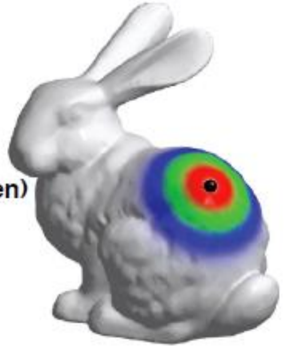
[Maturana et al. 2015]
[Wu et al. 2015] (GAN)
[Qi et al. 2016]
[Liu et al. 2016]
[Wang et al. 2017] (O-Net)
[Tatarchenko et al. 2017] (OGN)
...

Volumetric



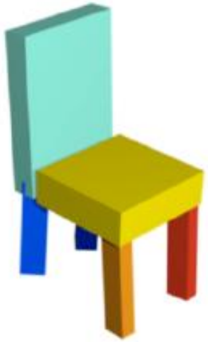
[Qi et al. 2017] (PointNet)
[Fan et al. 2017] (PointSetGen)

Point cloud



[Defferard et al. 2016]
[Henaff et al. 2015]
[Yi et al. 2017] (SyncSpecCNN)
...

Mesh (Graph CNN)



[Tulsiani et al. 2017]
[Li et al. 2017] (GRASS)

Part assembly

Deep Learning on 3d meshes

- *Math heavy approach, will be a standard deep learning tool, soon –*

The math ingredients of meshes

Manifolds

Sparse data structures

Differential geometry

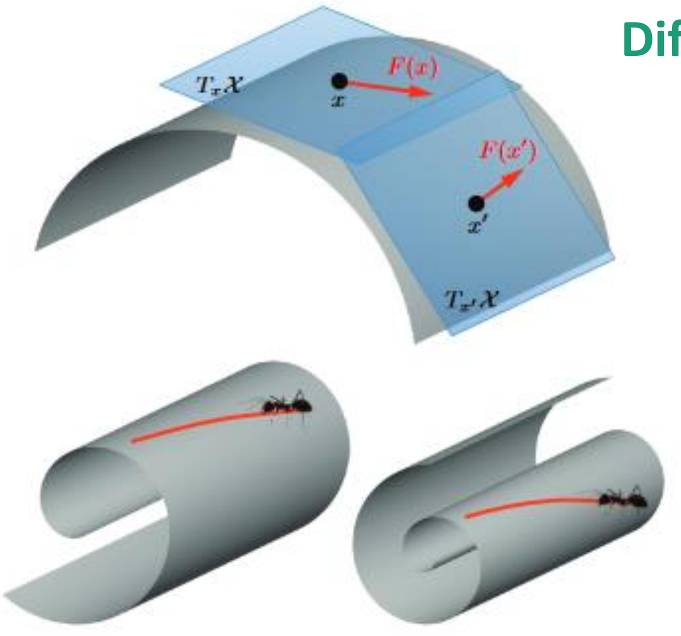
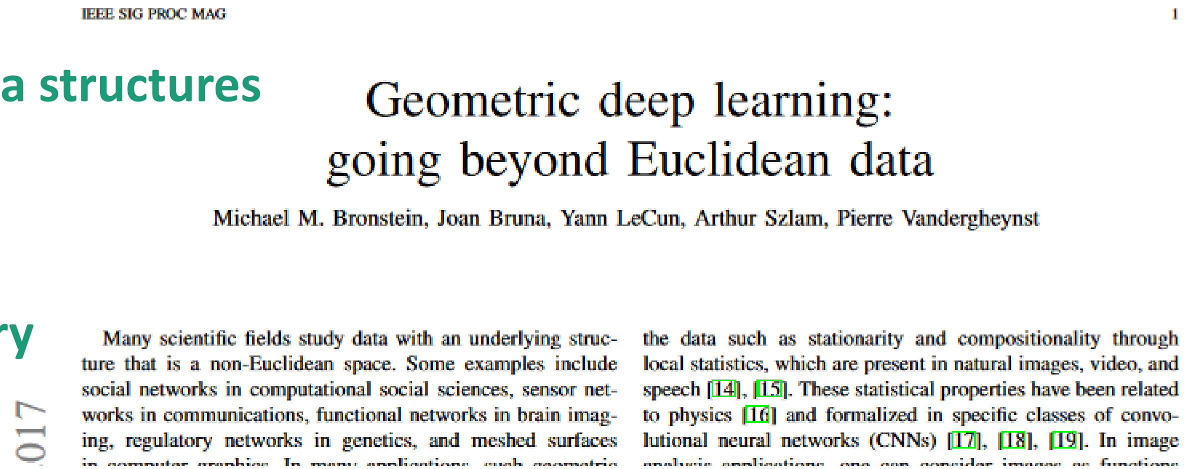


Fig. 1. Top: tangent space and tangent vectors on a two-dimensional manifold (surface). Bottom: Examples of isometric deformations.



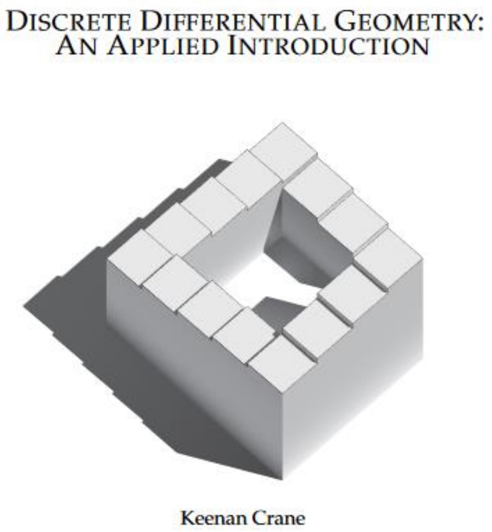
IEEE SIG PROC MAG 1

Geometric deep learning: going beyond Euclidean data

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

Many scientific fields study data with an underlying structure that is a non-Euclidean space. Some examples include social networks in computational social sciences, sensor networks in communications, functional networks in brain imaging, regulatory networks in genetics, and meshed surfaces

the data such as stationarity and compositionality through local statistics, which are present in natural images, video, and speech [14], [15]. These statistical properties have been related to physics [16] and formalized in specific classes of convolutional neural networks (CNNs) [17], [18], [19]. In image



DISCRETE DIFFERENTIAL GEOMETRY: AN APPLIED INTRODUCTION

Keenan Crane

Differential topology

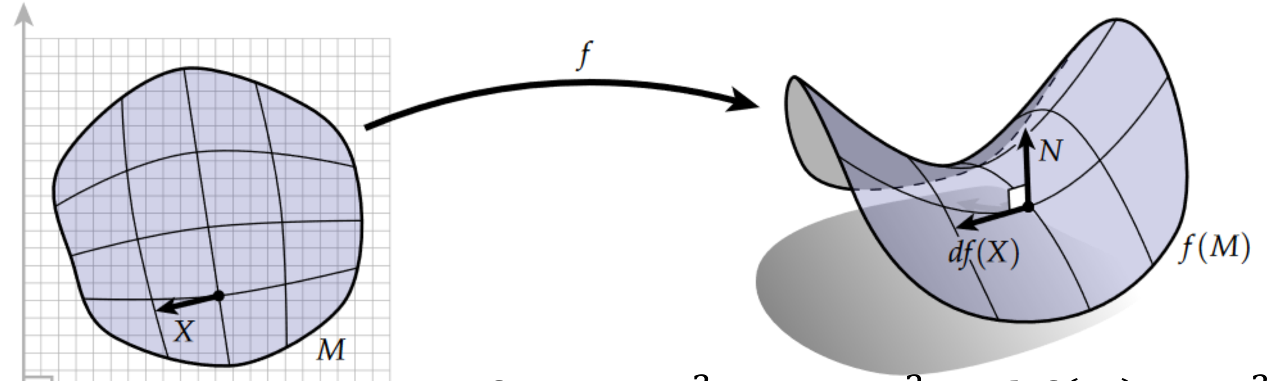
Laplacian

Graph theory

Credits to Michael Bronstein et. al., 2016 and Keenan Crane, 2019

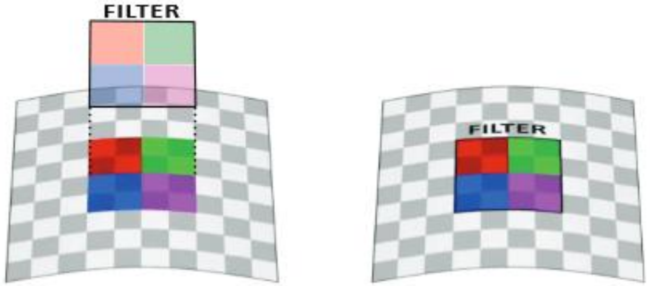
Three strategies to define a convolution neural network on meshes

- RNNs (more like a brute force approach)
- Conduct convolution **on a parametrization** (typically 2d) of a mesh/graph (typically 3d)



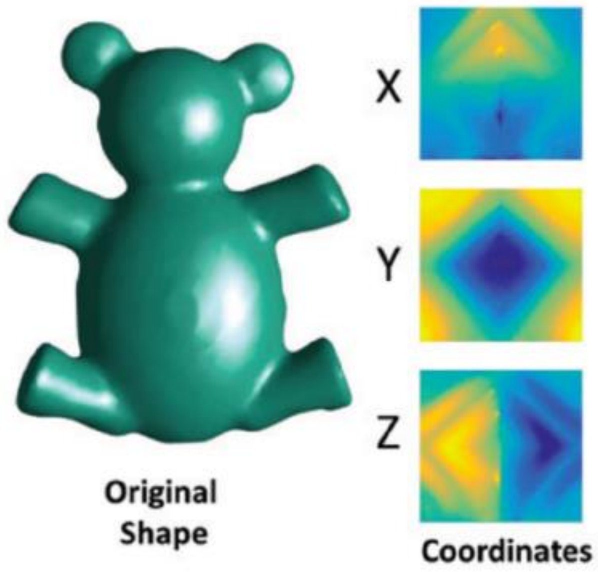
$$f: M \to \mathbb{R}^3, M \supset \mathbb{R}^2 \text{ and } f(M) \supset \mathbb{R}^3$$

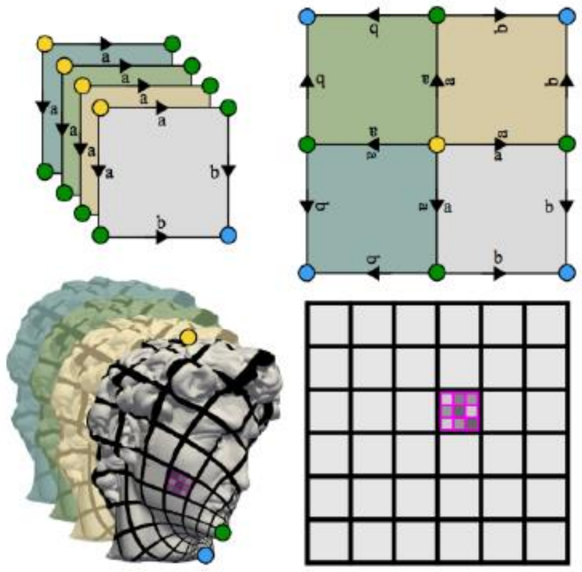
- Conduct convolution **on the mesh**



Bringing 3d into Euclidean plane and proceed with traditional techniques

- Map curved 3D surfaces to 2D Euclidean plane



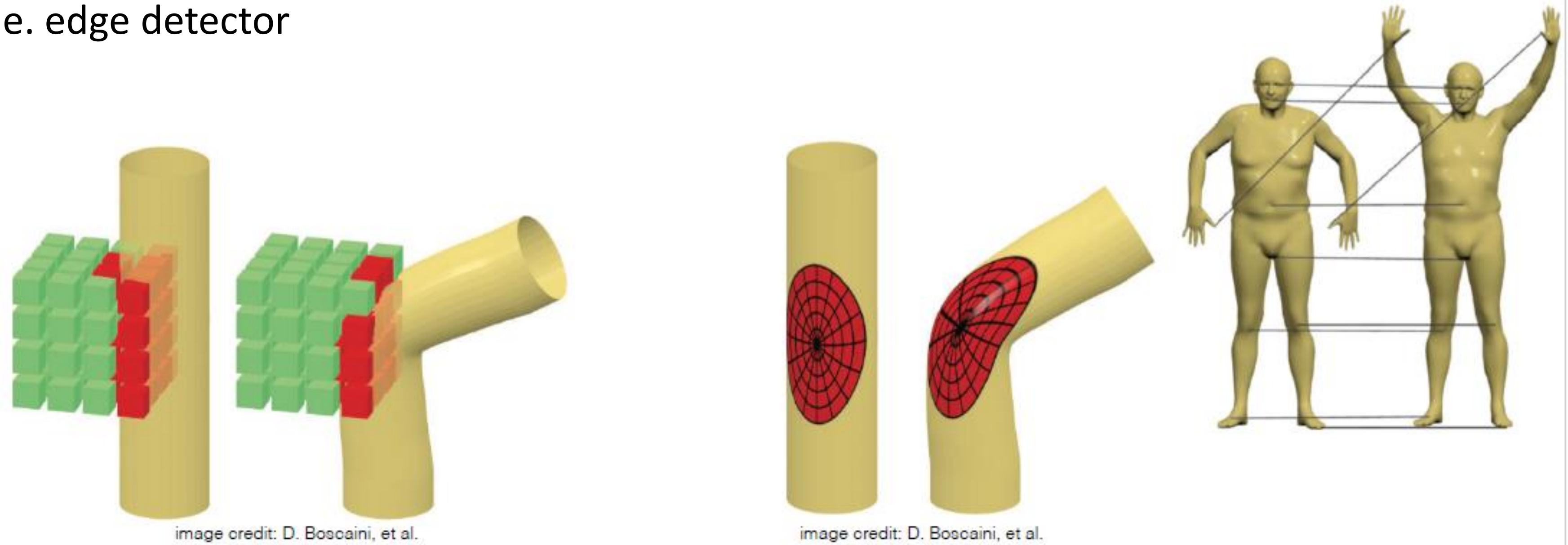


Ayan Sinha, Jing Bai, Karthik Ramani
"Deep Learning 3D Shape Surfaces Using Geometry Images"
ECCV2016

Maron et al.
"Convolutional Neural Networks on Surfaces via Seamless Toric Covers'
SIGGRAPH2017

Desired properties for convolution without parametrization

- Translation invariant filters, i.e. weight sharing
- Localized, i.e. edge detector

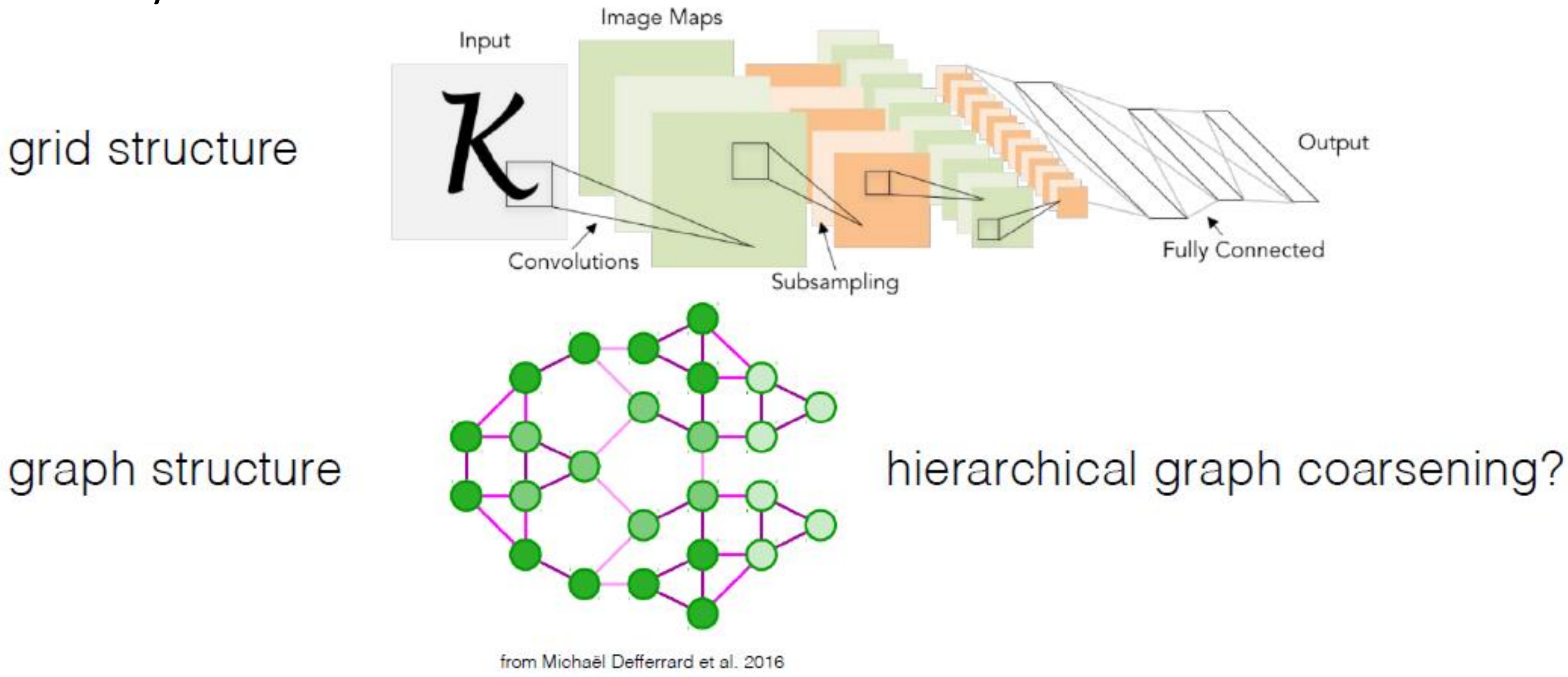


convolutional along spatial coordinates

convolutional considering underlying geometry

More inductive bias, please

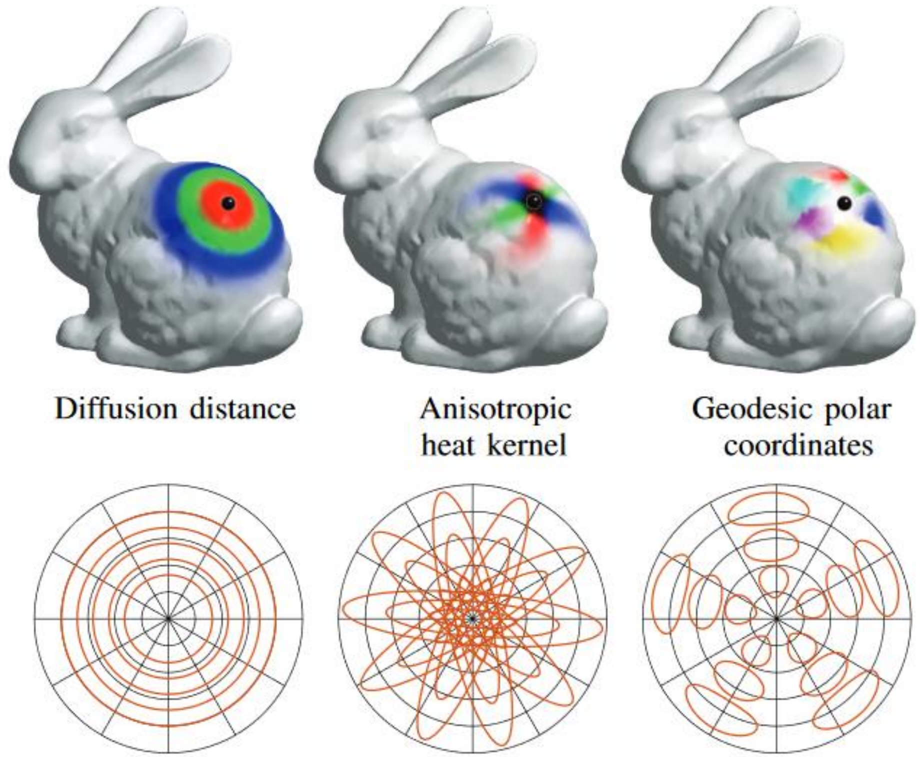
- Receptive fields
- Multi-scale analysis

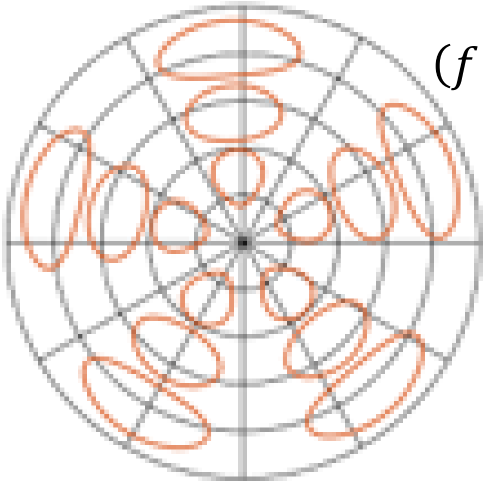


Credits to Michael Deferrard et. al., 2016

Geometry approach: Geodesic CNN

- Local system of geodesic polar coordinate
- Extract a small patch at each point x
- Compute response with a trainable patch-like filter





$$(f * g)[x] = \sum_{i}^{angles} \sum_{j}^{rings} g_{ij} D_{ij}(x) f$$

One weight g for all i*j basis functions
In a local point specific coordinate system

Credits to Jonathan Masci et. al., 2015

Geometry approach: Geodesic CNN

- Direct encoding of the differential geometry
- The radius of the geodesic patches must be sufficiently small to acquire a topological disk
- No effective pooling, purely relying on convolutions to increase receptive field
- Slow because of huge tensors because of local of coordinate frames
- Limited to rotation invariant filters or curvature aligned filters

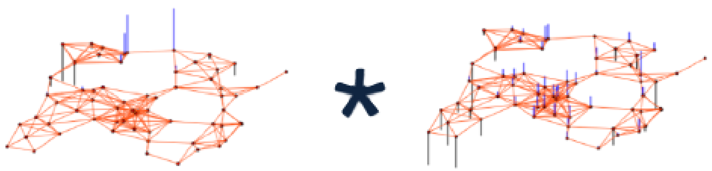
Credits to Jonathan Masci et. al., 2015

Signal approach: Spectral CNN

Generalized convolution of $f, g \in L^2(X)$ can be defined by analogy

$$(f \star g)(x) = \underbrace{\sum_{k\geq 1} \underbrace{\langle f, \phi_k \rangle_{L^2(X)} \langle g, \phi_k \rangle_{L^2(X)}}_{\text{product in the Fourier domain}} \phi_k(x)}_{\text{inverse Fourier transform}}$$

Generalized convolution allows spectral filtering!



The Laplace operator tells us something about curvature!
>> We can compute Eigenfunctions of the Laplacian

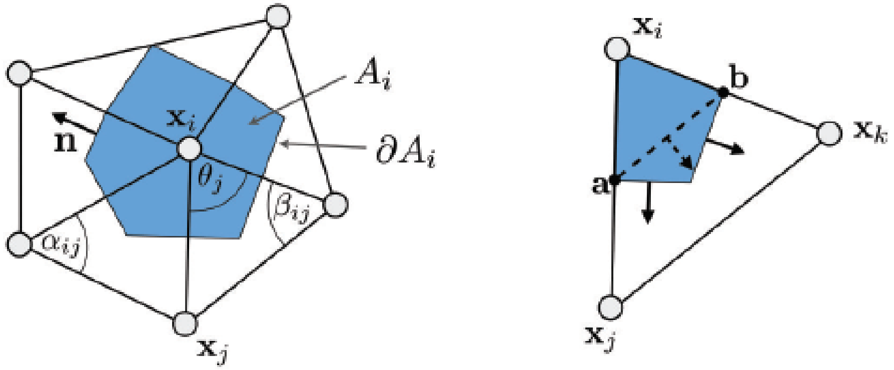
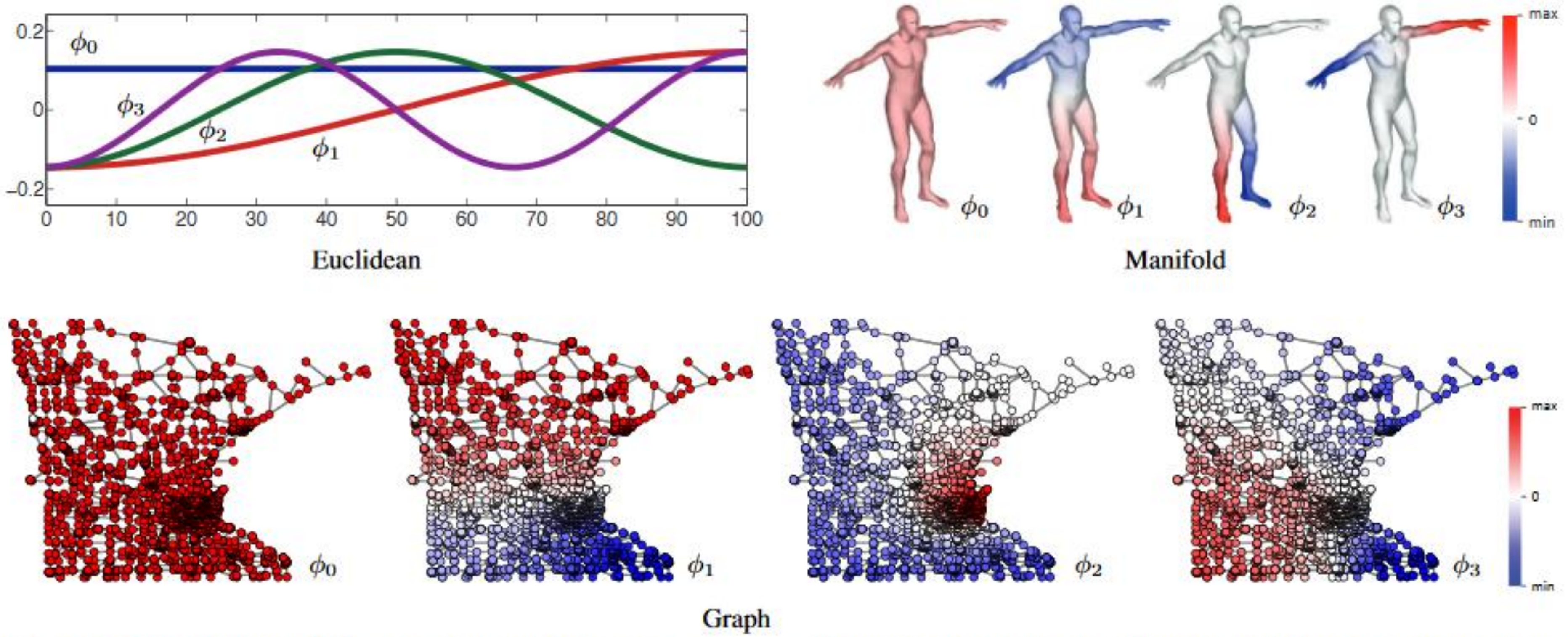


Figure 3.10. Illustration of the quantities used in the derivation of the discrete Laplace-Beltrami operator and discrete Gaussian curvature operator.

Credits to Michael Bronstein et. al., 2016

Signal approach: Spectral CNN



[FIGS3] Example of the first four Laplacian eigenfunctions $\phi_0, \ldots, \phi_3$ on a Euclidean domain (1D line, top left) and non-Euclidean domains (human shape modeled as a 2D manifold, top right; and Minnesota road graph, bottom). In the Euclidean case, the result is the standard Fourier basis comprising sinusoids of increasing frequency. In all cases, the eigenfunction ϕ_0 corresponding to zero eigenvalue is constant ('DC').

Credits to Michael Bronstein et. al., 2016

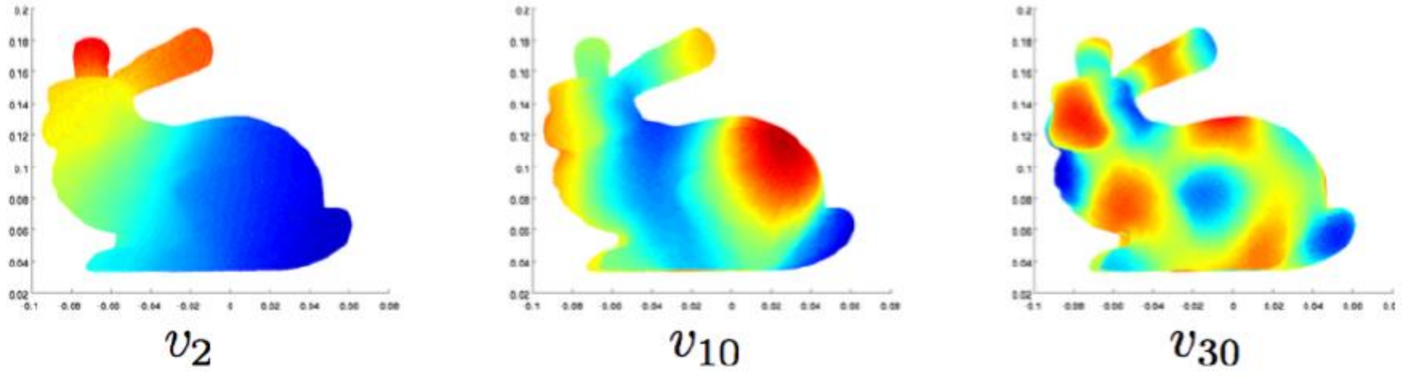
Signal approach: Spectral CNN

Mesh basis: Eigenfunctions of the Laplace-Beltrami-Operator Δ

Define the filter function g as a function of Laplace-Beltrami-Operator s a Δ

$g_{\boldsymbol{\alpha}}(\boldsymbol{\Delta}) \quad = \quad \boldsymbol{\Phi} g_{\boldsymbol{\alpha}}(\boldsymbol{\Lambda}) \boldsymbol{\Phi}^{\top}$ (Eigenspace of Graph)

$g_{\boldsymbol{\alpha}}(\lambda) = \sum_{j=0}^{r-1} \alpha_j \lambda^j$ (Function of Eigenvalues)



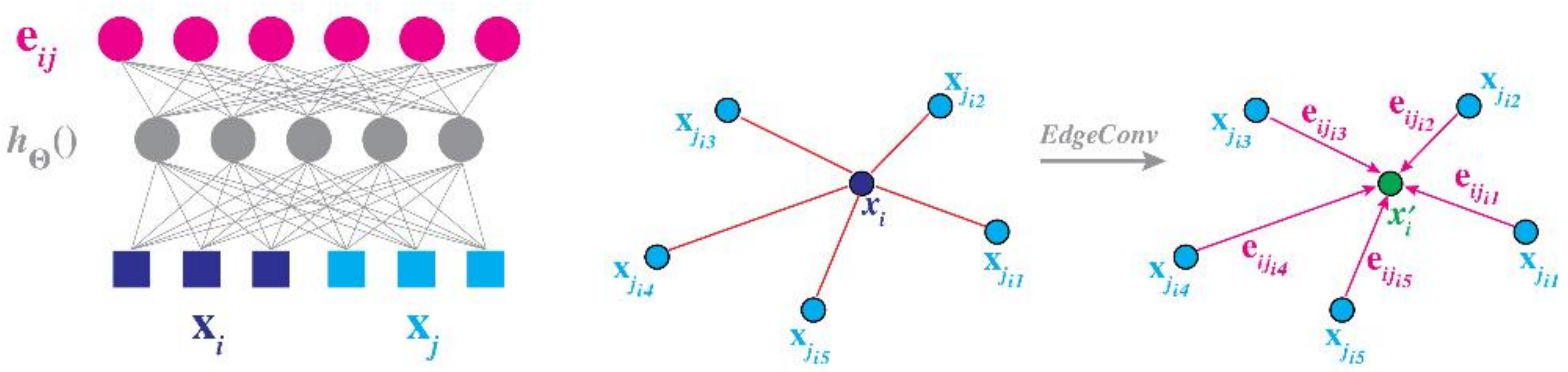
Credits to Mario Botsch et. al., 2010

Signal approach: Spectral CNN

- Filters are exactly localized in ***r***-hops support
- O(1) parameters per layer
- No computation of ϕ, ϕT ⇒ O(***n***) computational complexity
- Stable under coefficients perturbation
- Filters are basis-dependent ⇒ does not generalize across graphs, i.e. Eigenfunctions are Laplacian-specific and therefore graph specific.

Credits to Jonathan Masci et. al., 2015

Graph approach: Graph CNN

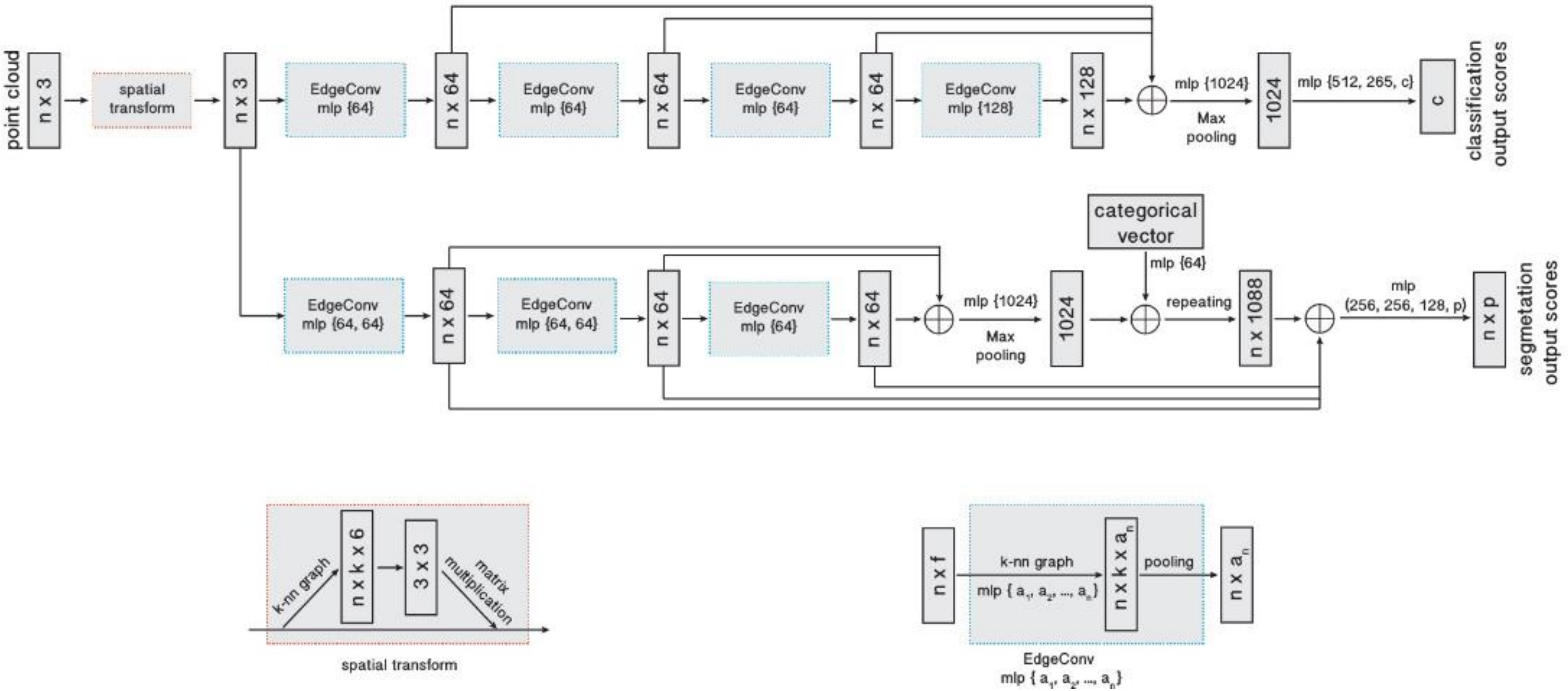


- **Minimal inner** structure (no fixed indexing of the nodes required)
- **Localized** (only neighbors are considered)
- **Weight sharing** (convolution-like operations)
- **Graph topology** independent

$$x_i = f\_gnn(\{x_j : j \rightarrow i\})$$

Credits to Michael Bronstein et. al., 2018

Graph approach: Graph CNN



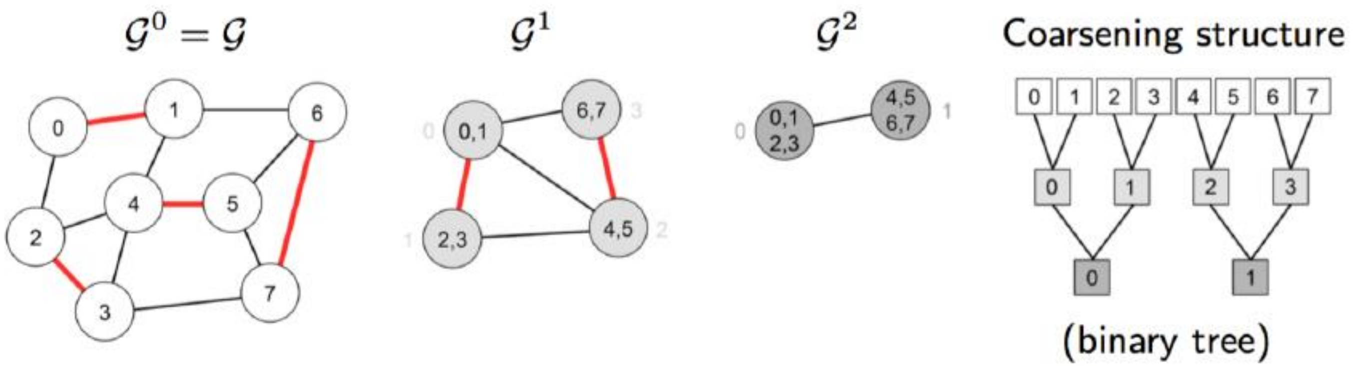
Credits to Michael Bronstein et. al., 2018

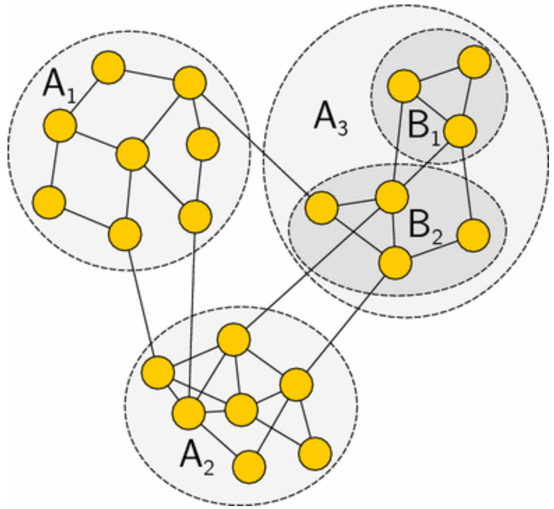
Graph approach: Graph CNN

- Generalizes well to changing graph topologies
- Unified framework
- Slow k-nearest neighbor searches
- Only pairwise relationships and no assumption about being locally flat

Graclus, the typical pooling layer

- Graph downsampling == graph coarsening == graph pooling == graph partitioning. Decompose Graph into meaningful clusters.
- Graph partitioning is NP hard → Use Graclus approximation





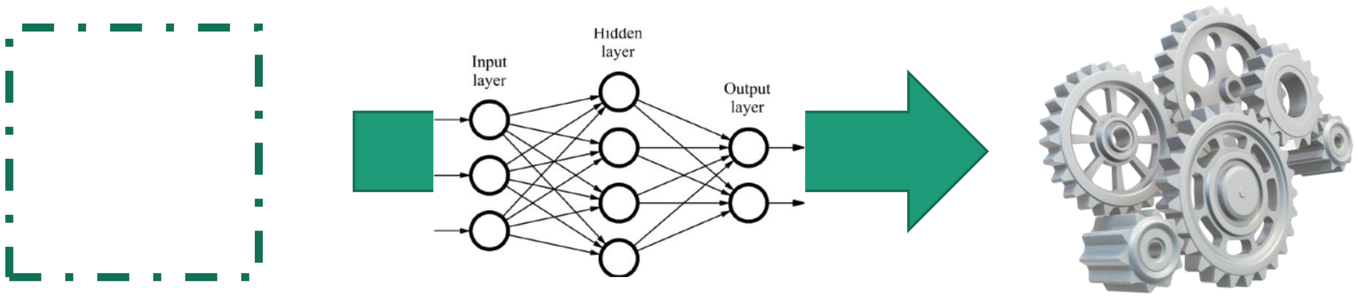
Credits to Dhillon, Guan, Kulis 2007 and Defferrard, Bresson, Vandergheynst 2016

Techniques can be easily generalized to general graphs

3D shape graph social network molecules

Open issues with mesh based representation

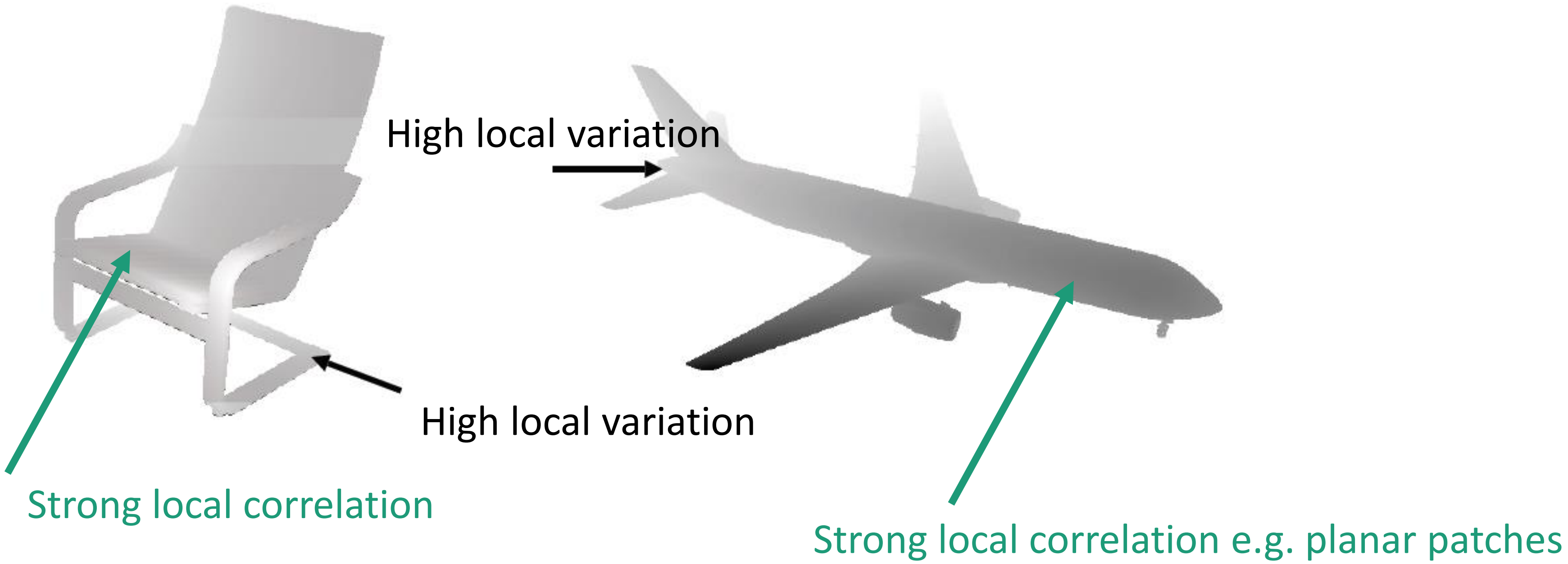
- Mesh as network output is difficult as topology may be variable
- Not clear how to generate shapes with topology variation
- No unique parametrization available, we need to match graphs in order to compute loss function!



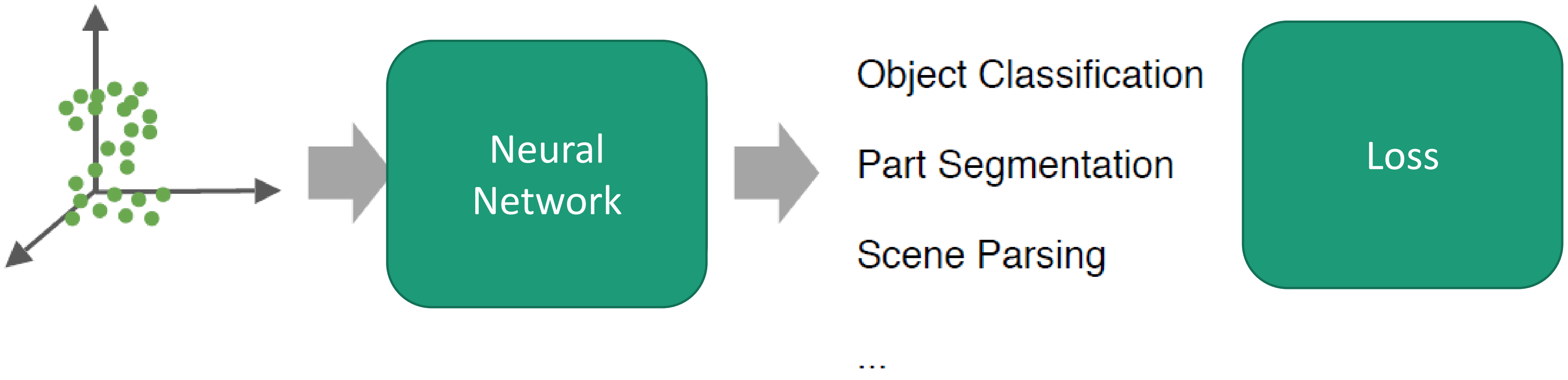
Deep Learning on point clouds

- *The computer scientists’ approach: theory follows implementation –*

Statistics of geometry



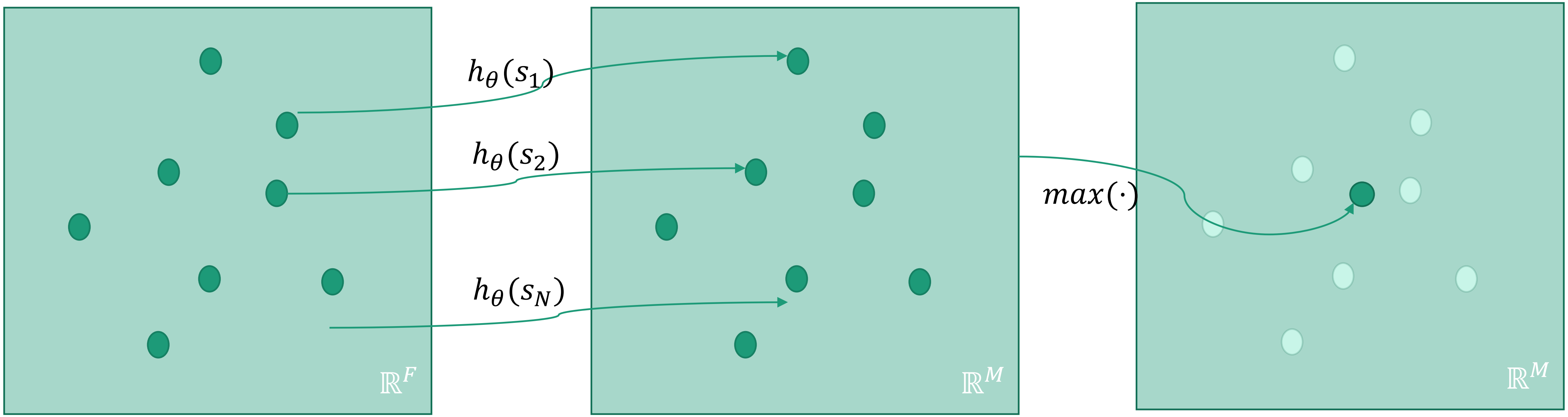
The desired pipeline



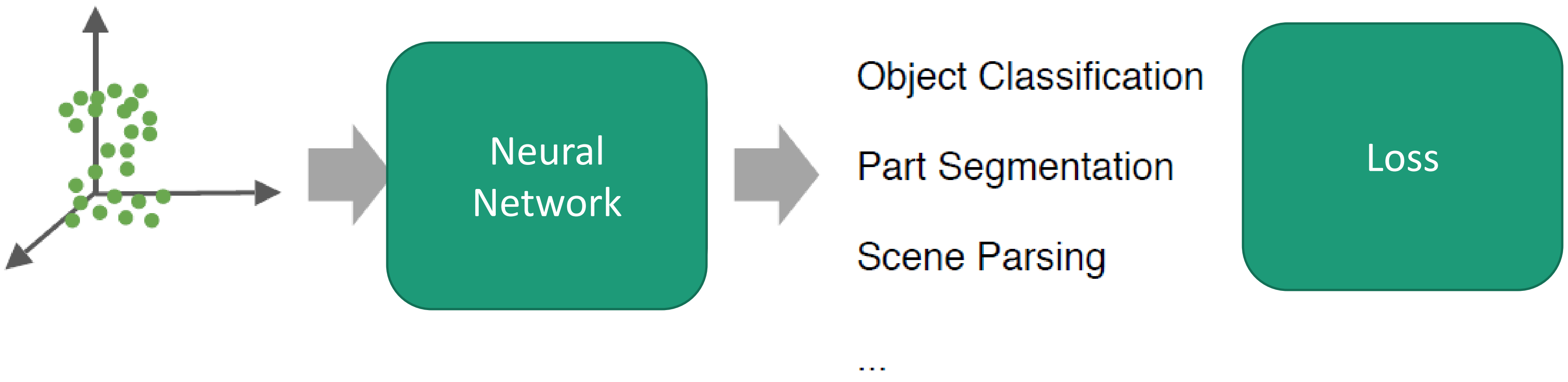
Natural questions arise:
- How to order input points?
- How to induce that nearby points are correlated
- Which loss functions can I use?

Simple approach

- $f(S) = g(\{h(s_1), h(s_2), \dots, h(s_N)\}),$
  $with\ feature\ map\ h\colon \mathbb{R}^F \to \mathbb{R}^M, symmetric\ g\colon 2^X \to \mathbb{R}\ and\ S \supseteq \mathbb{R}^D$



The desired pipeline



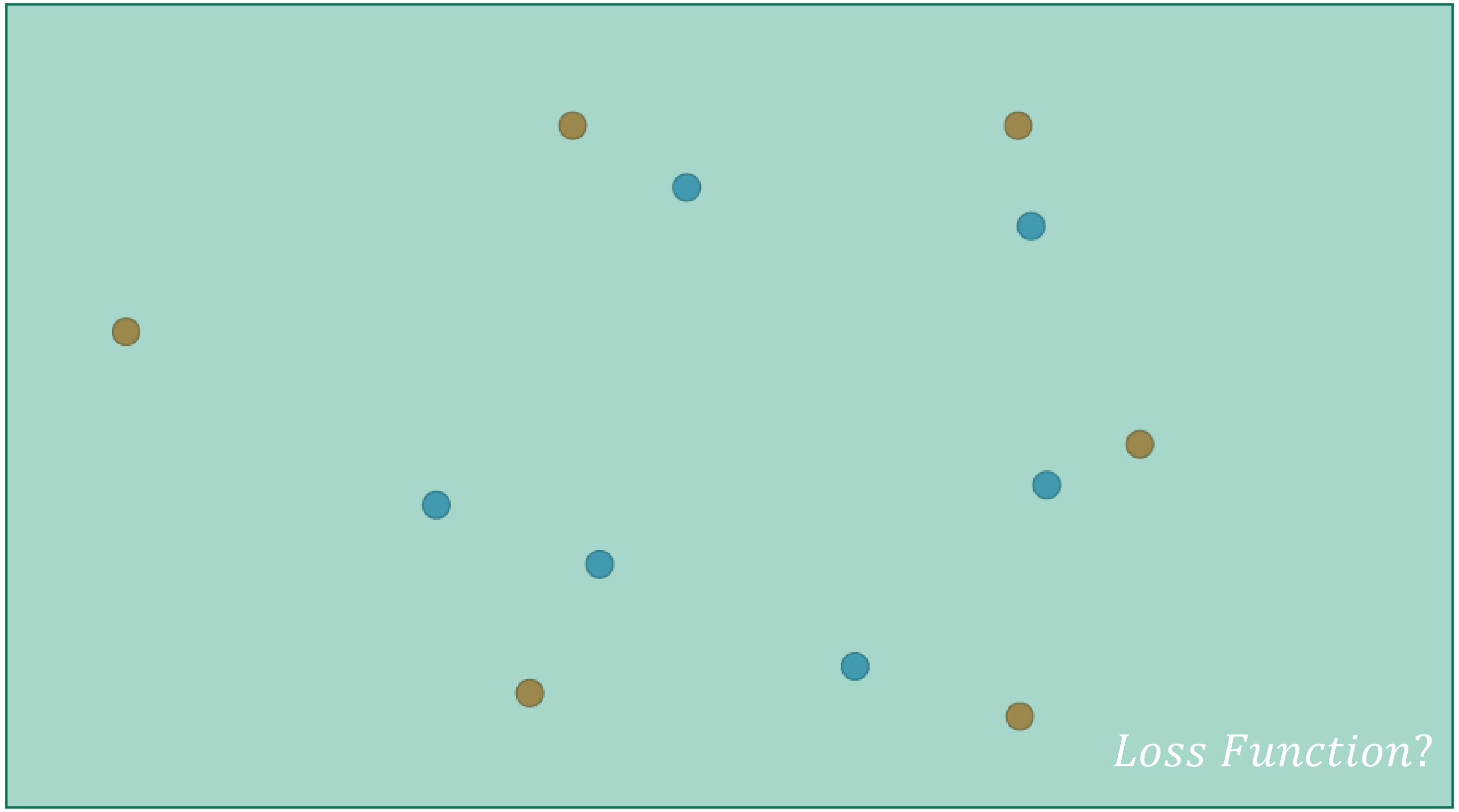
Natural questions arise:

- How to order input points? → **Doesn't matter, feature map h() gets applied individually**
- How to induce that nearby points are correlated → **Learned from data**
- Which loss functions can I use? → **g() yields a vector, standard losses for classification, etc.**

→ What about segmentation, deconvolution, predicting points?

Example semantic segmentation

Correspondence problem when predicting point clouds

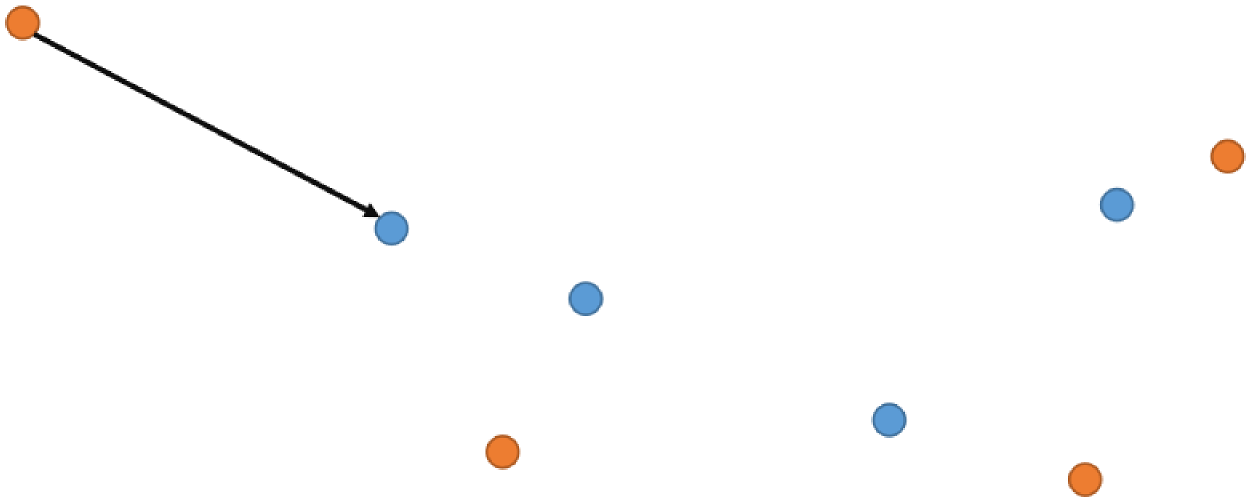


Given two sets of points, measure their discrepancy

Typical distances between sets

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

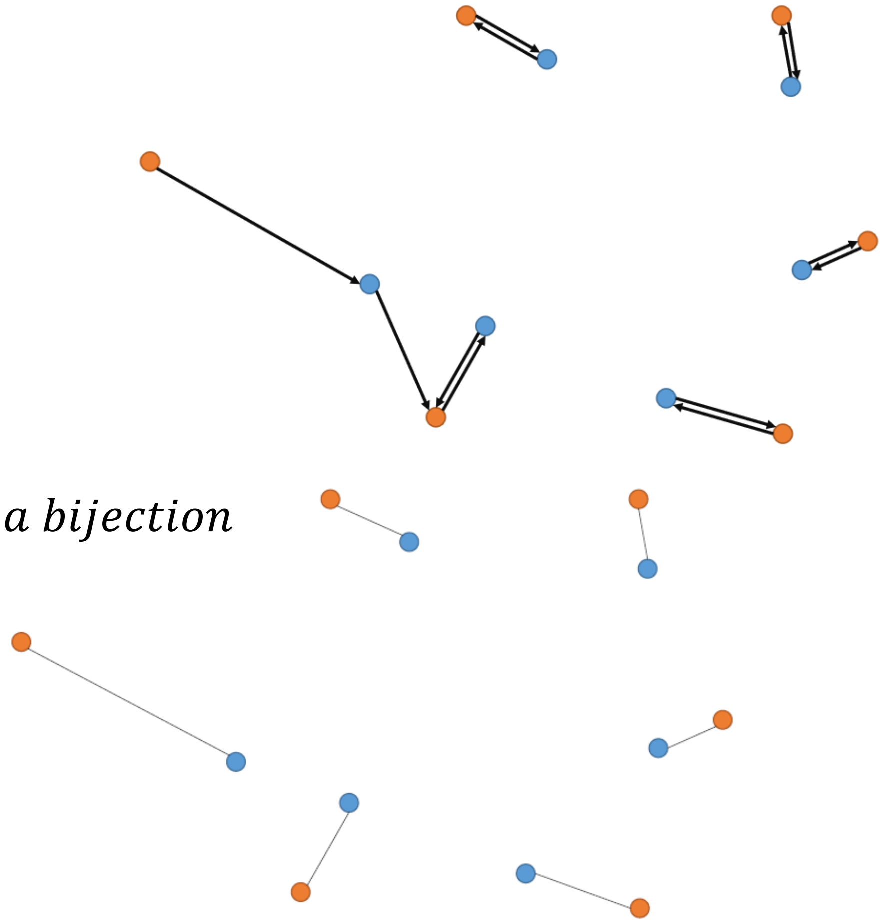
Not very robust!



Typical distances between sets

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

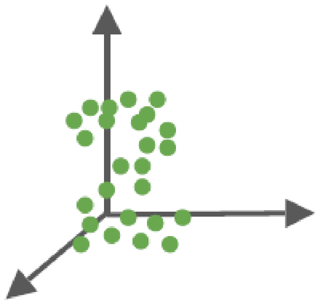
$$d_{EarthMover}(S_1, S_2) = \min_{\phi: x_1 \to x_2} \sum_{x \in S_1} ||x - \phi(x)||_2^2 \ with\ \phi: S_1 \to S_2\ is\ a\ bijection$$



Simple function of coordinates:

- In general positions, the correspondence is unique
- With infinitesimal movement, the correspondence does not change
- **Conclusion: differentiable almost everywhere**

The desired pipeline for point predictions



Neural Network

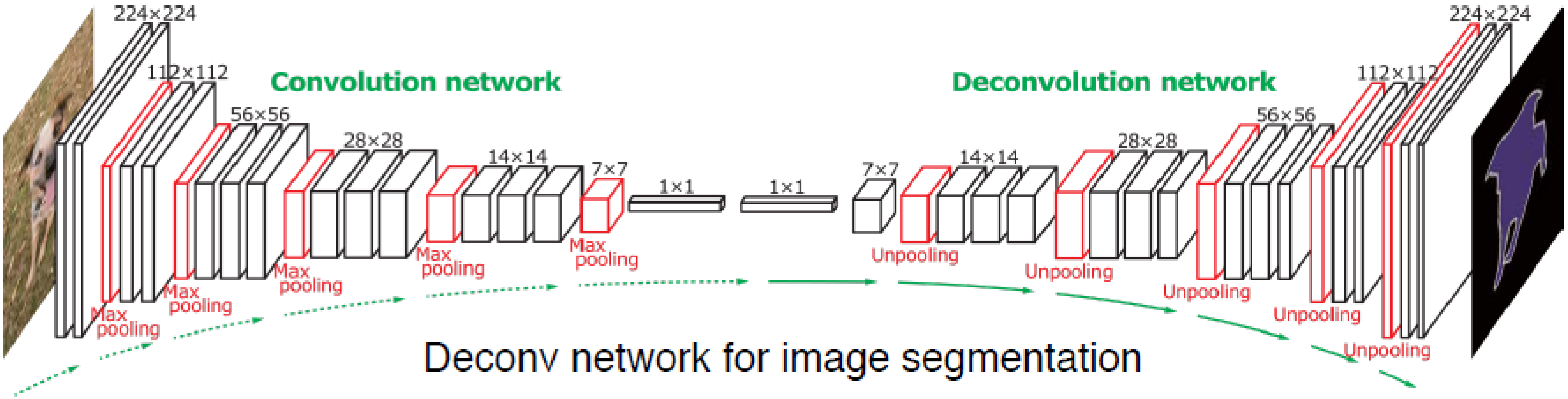
$\{ (x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_N, y_N, z_N) \}$

Loss on sets

...

→ We want to predict points in space! How to implement devonvolution?

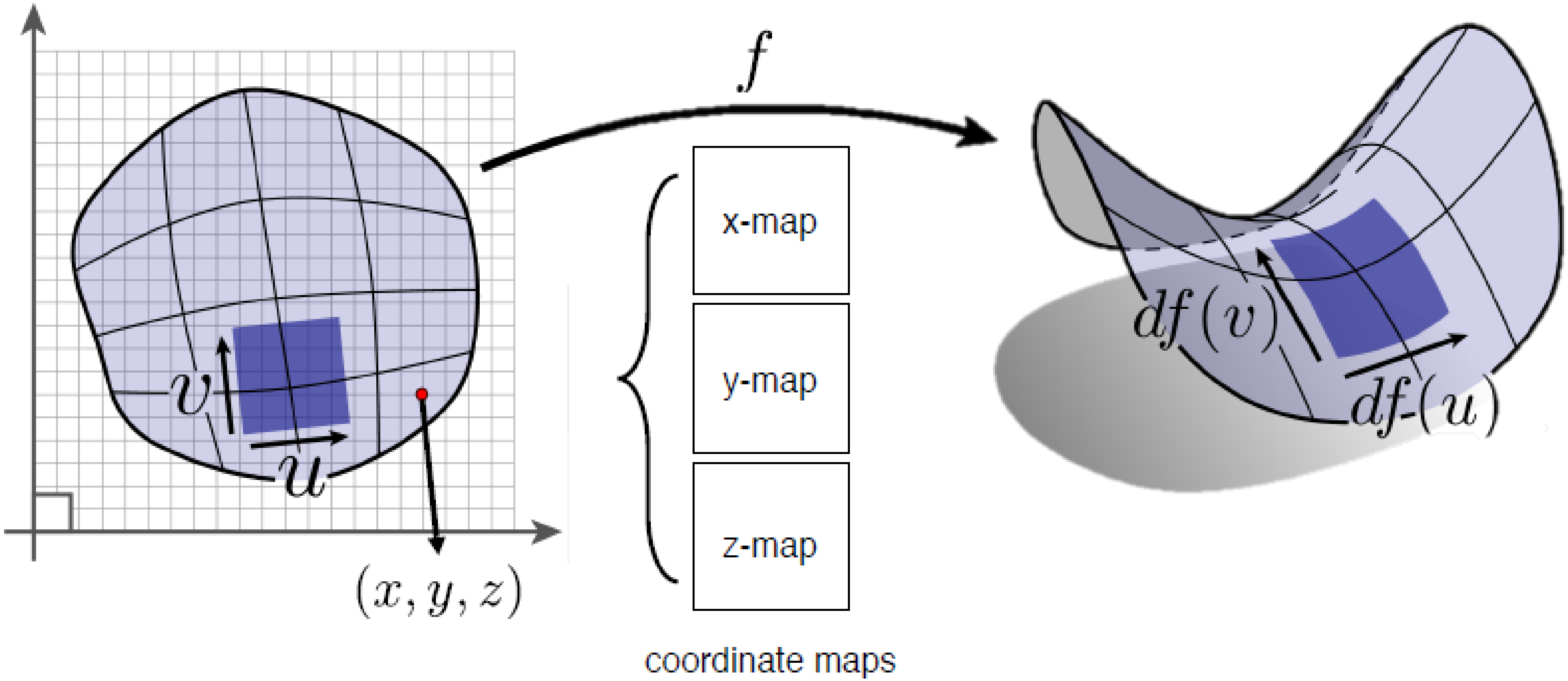
Recap Image Segmentation with DeconvNet



Credit: FCNN, Long et al.

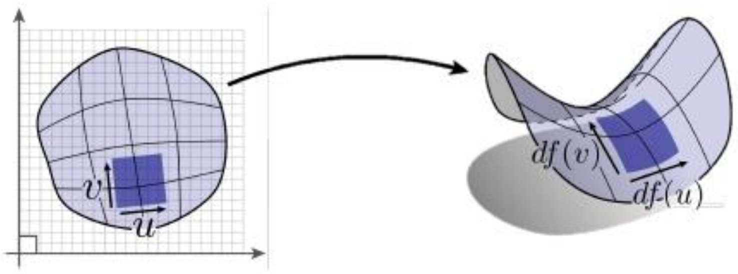
Observation: Parametrization looks like image deconvolution

Surface parametrization (2D↔3D mapping)

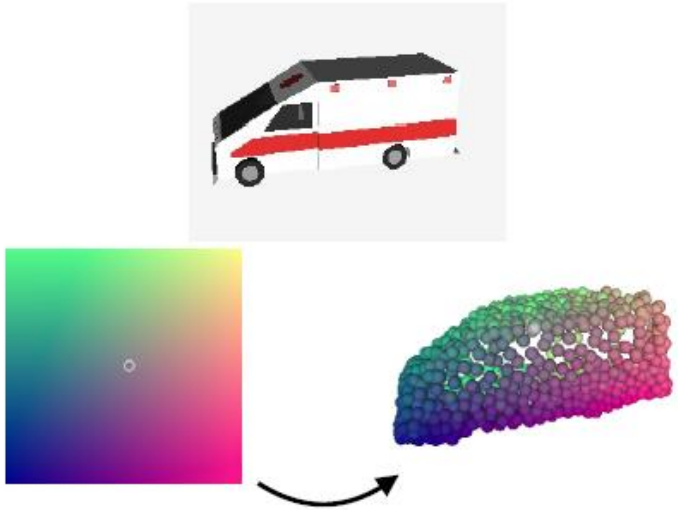


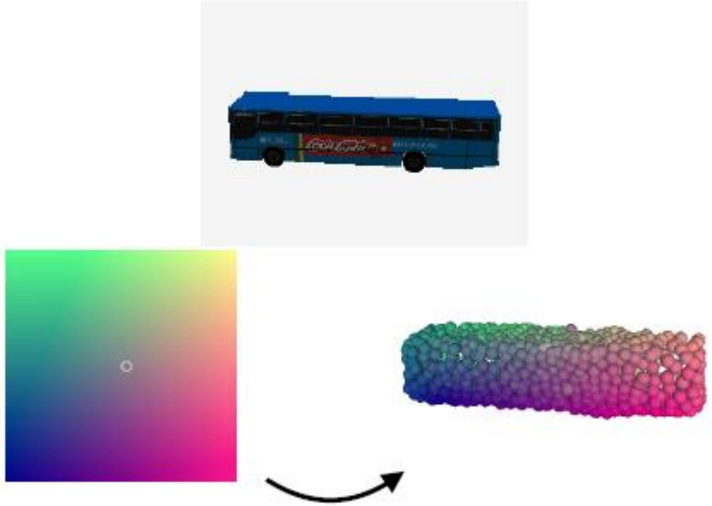
Credits Keenan Crane, 2019

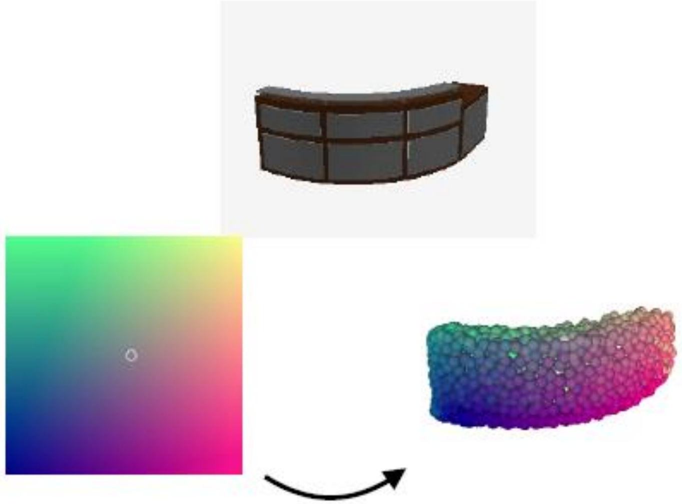
Example Smooth Point Cloud Prediction



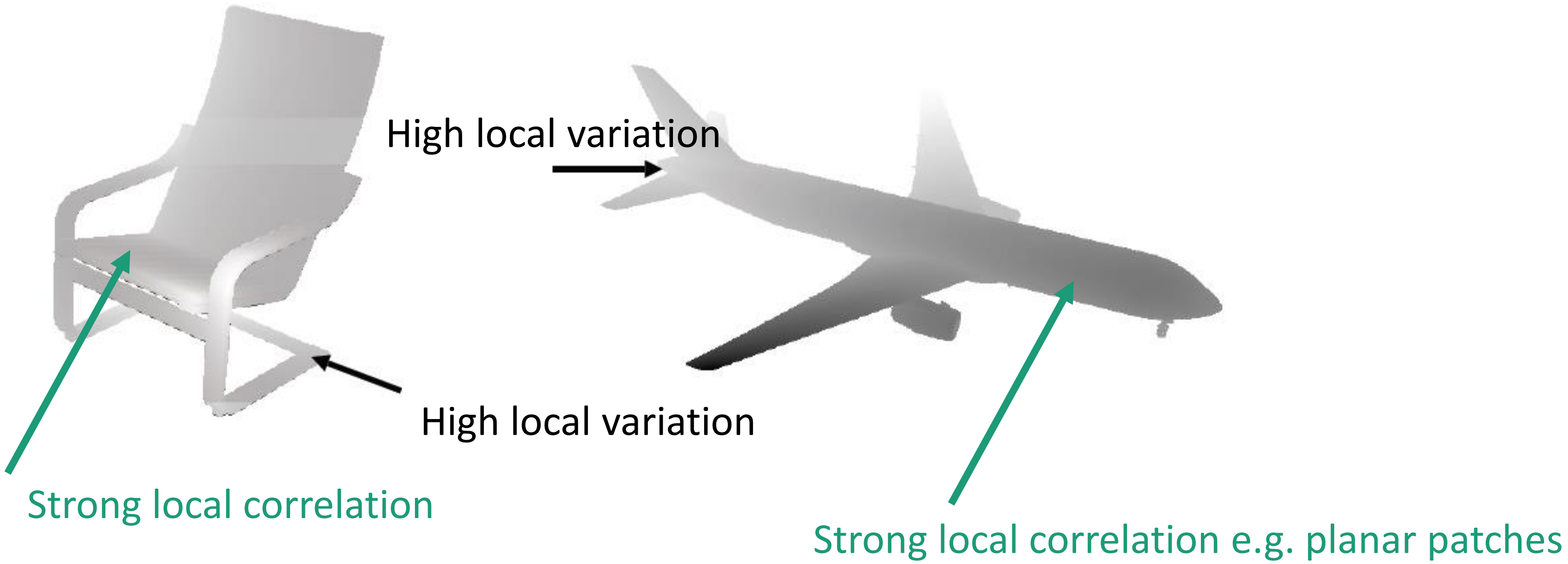
Network outputs are coordinate maps (x, y, z) !



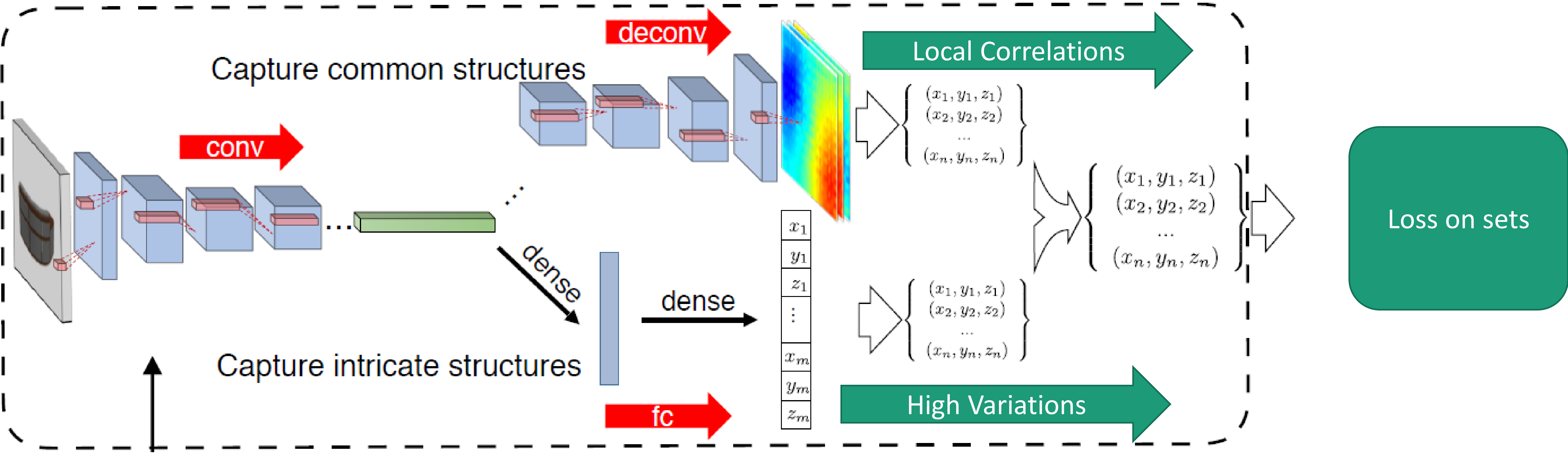




Recap: Statistics of geometry



Full example architecture of a point network



Credit Hao Su, 2017

Sharp and Smooth structures

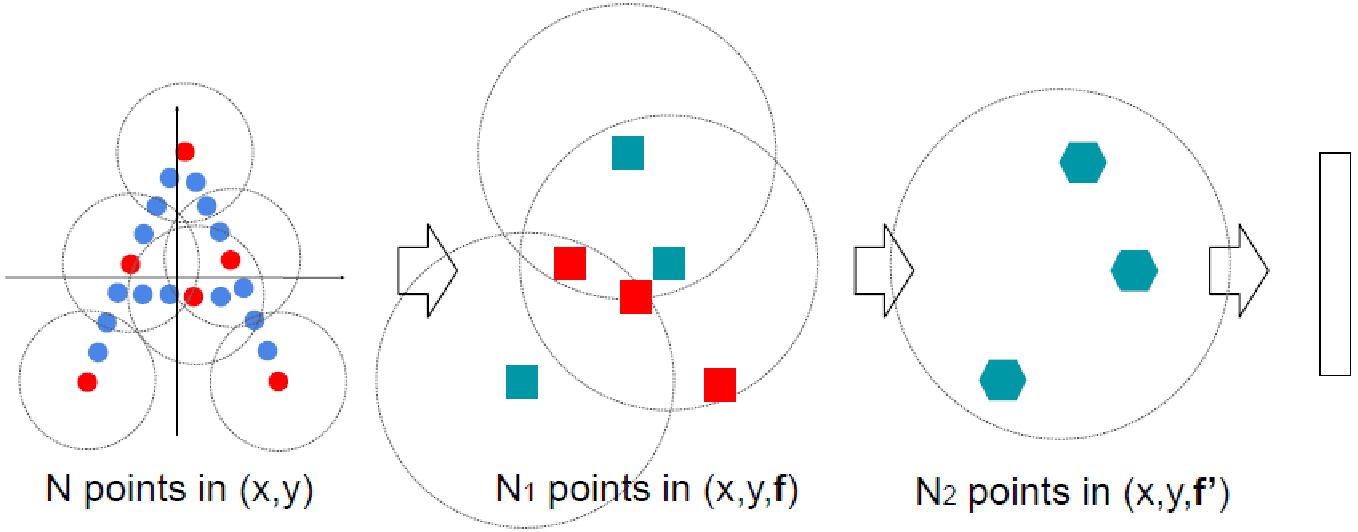
CVPR '17, Point Set Generation

Example Shape Completion from RGB-D

RGBD map (input) | $90°$ view of input | output: completed point cloud

Credit Hao Su, 2017

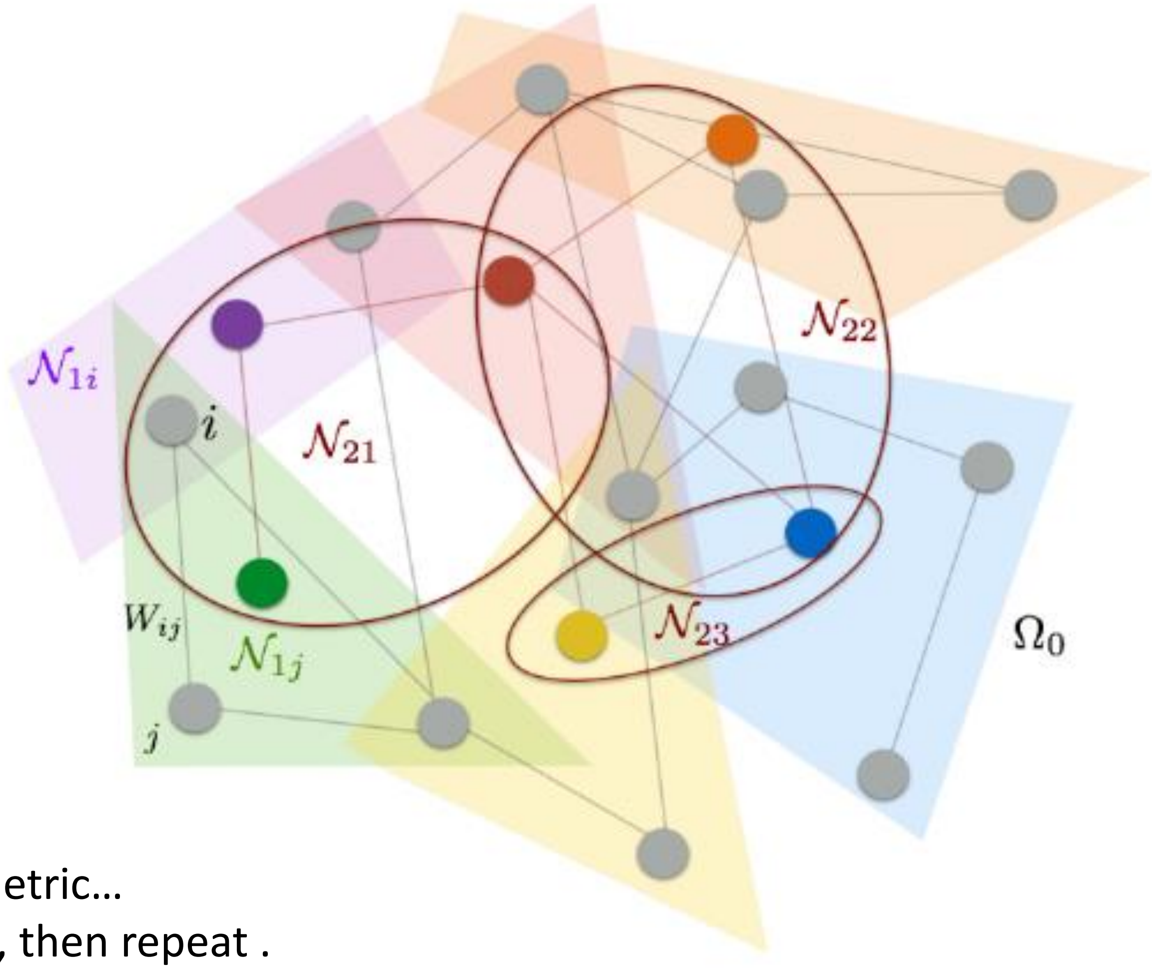
Farthest point sampling (FPS), the typical pooling layer



Common Framework

- *Everything is a graph -*

Comparing to Graph CNN



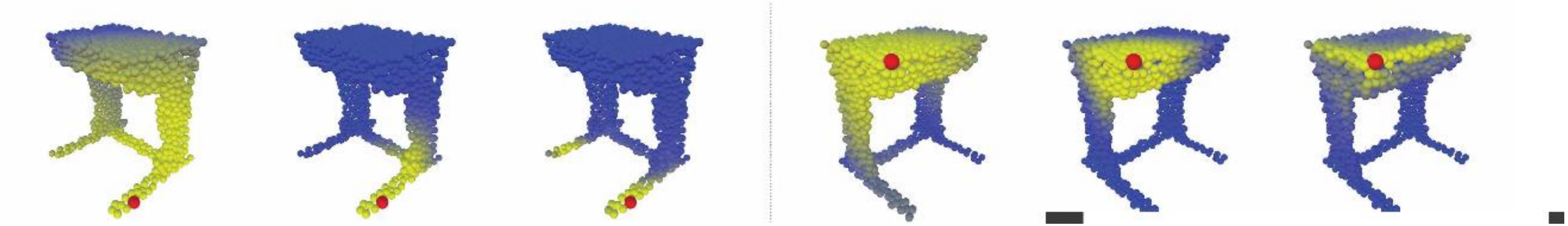
Very similar to Graph CNN with euclidean metric…
…Local feature extraction, graph coarsening, then repeat .

Joan Bruna et. al., 2014

Graph CNN

	Aggregation	Edge Function	Learnable parameters
PointNet [Qi et al. 2017b]	—	$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_i)$	Θ
PointNet++ [Qi et al. 2017c]	max	$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_j)$	Θ
MoNet [Monti et al. 2017a]	$\sum$	$h_{\theta_m, w_n}(\mathbf{x}_i, \mathbf{x}_j) = \theta_m \cdot (\mathbf{x}_j \odot g_{w_n}(u(\mathbf{x}_i, \mathbf{x}_j)))$	w_n, θ_m
PCNN [Atzmon et al. 2018]	$\sum$	$h_{\theta_m}(\mathbf{x}_i, \mathbf{x}_j) = (\theta_m \cdot \mathbf{x}_j) g(u(\mathbf{x}_i, \mathbf{x}_j))$	θ_m

Table 1. Comparison to existing methods. The per-point weight w_i in [Atzmon et al. 2018] effectively is computed in the first layer and could be carried onward as an extra feature; we omit this for simplicity.



Credits to Michael Bronstein et. al., 2018

Example in PyTorch

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        nn = Seq(Lin(coord_dims, 64), ReLU(), Lin(64, 64))
        self.conv1 = PointConv(local_nn=nn)

        nn = Seq(Lin(coord_dims + 64, 128), ReLU(), Lin(128, 128))
        self.conv2 = PointConv(local_nn=nn)

        self.lin2 = Lin(128, 256)
        self.lin3 = Lin(256, num_classes)

    def forward(self, data):
        pos, batch = data.pos, data.batch

        edge_index = radius_graph(pos, r=0.2, batch=batch)
        x = F.relu(self.conv1(None, pos, edge_index))

        idx = fps(pos, batch, ratio=0.5)
        x, pos, batch = x[idx], pos[idx], batch[idx]

        edge_index = radius_graph(pos, r=0.2, batch=batch)
        x = F.relu(self.conv2(x, pos, edge_index))
        x = global_max_pool(x, batch)
        x = F.relu(self.lin2(x))
        x = self.lin3(x)
        return F.log_softmax(x, dim=-1)

model = Net()
optimizer = torch.optim.SGD(model.parameters(), lr=lrate, momentum=0.95)
loss = (lambda x, y: F.nll_loss(F.log_softmax(x, dim=1), y))

from ummon import *
with Logger(loglevel=20, logdir='', log_batch_interval=1) as lg:
    trn = ClassificationTrainer(lg, model, loss, optimizer)
    trn.fit(train_loader, epochs=100)
```

Graph CNN

- Practical applicable, easy to understand, fast, works well
- Unified framework, easy to implement
- Models only pairwise correlations
- Not using curvature information
- Set theoretic approach
- Not Riemannian

Theorem:

A Hausdorff continuous symmetric function $f : 2^{\mathcal{X}} \to \mathbb{R}$ can be arbitrarily approximated by PointNet.

$$\left| f(S) - \underbrace{\gamma \left(\underset{x_i \in S}{\mathrm{MAX}} \{h(x_i)\} \right)}_{\text{PointNet (vanilla)}} \right| < \epsilon$$

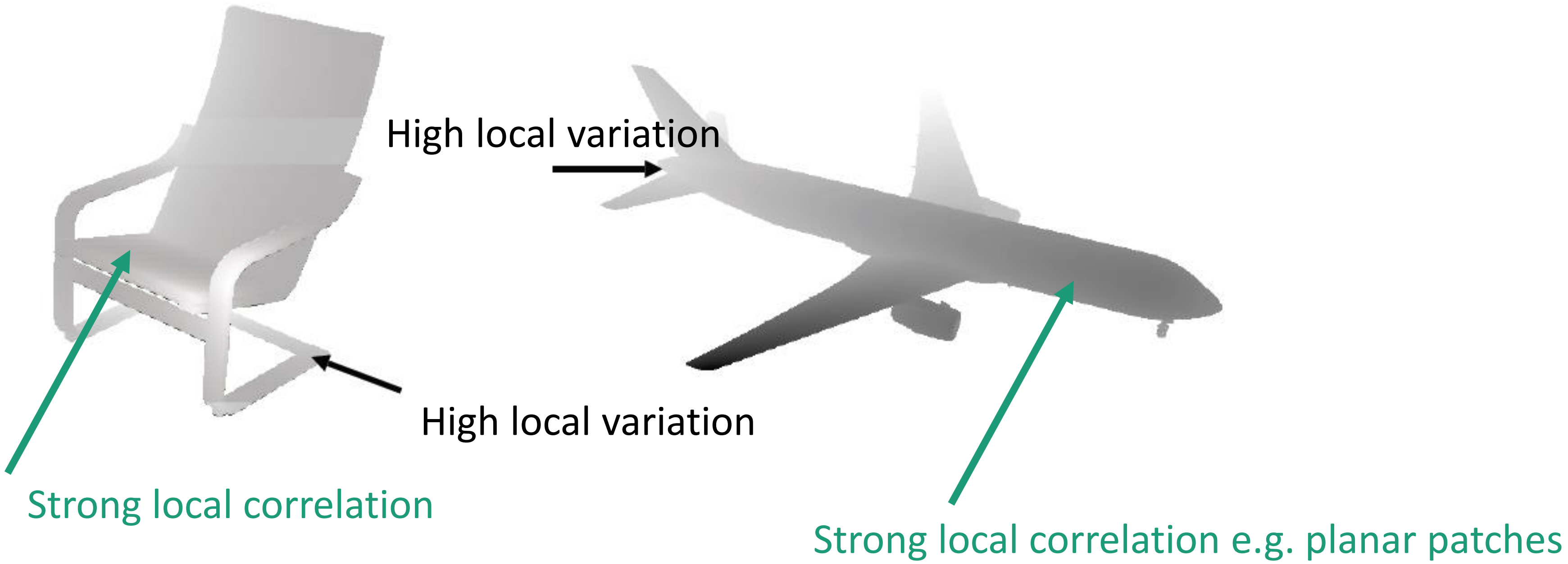
$S \subseteq \mathbb{R}^d$,

PointNet (vanilla)

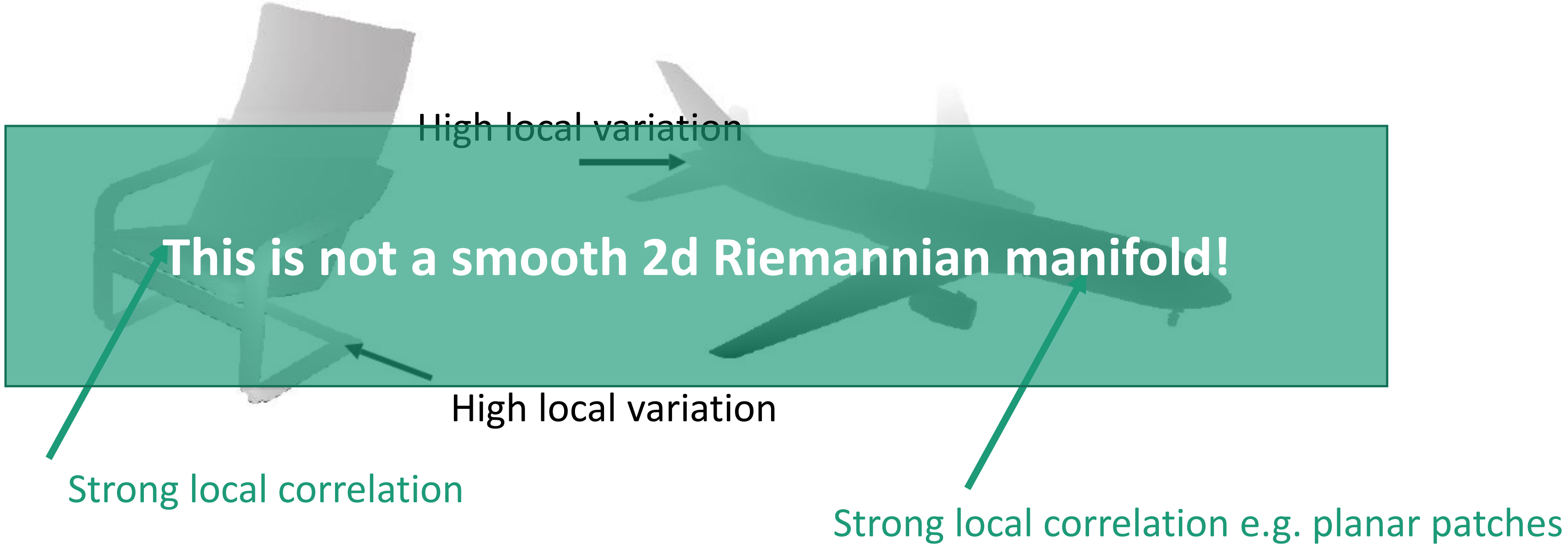
Cool, but only half the story!

- *Carl Friedrich says-*

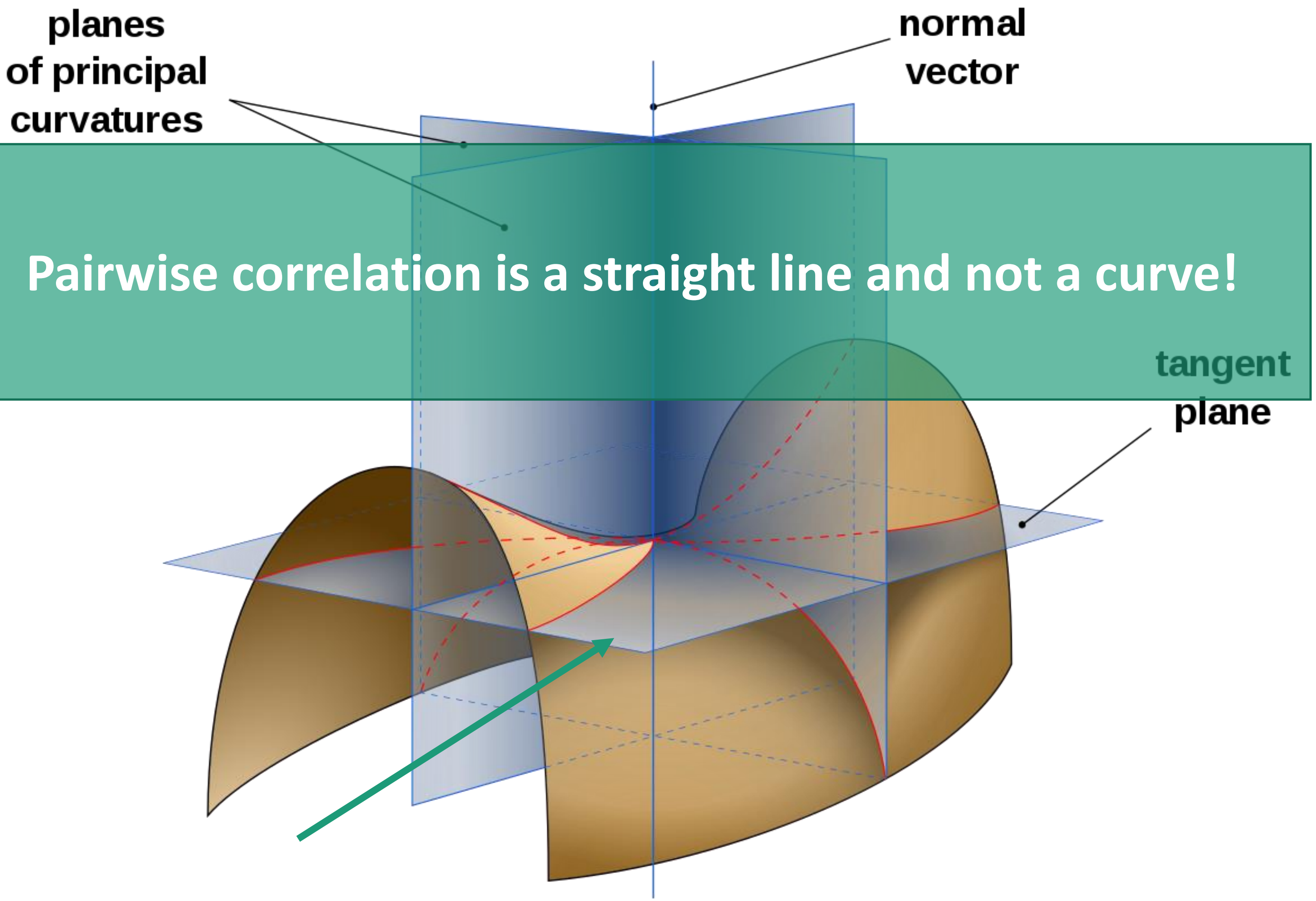
Recap: Statistics of geometry



Recap: Statistics of geometry

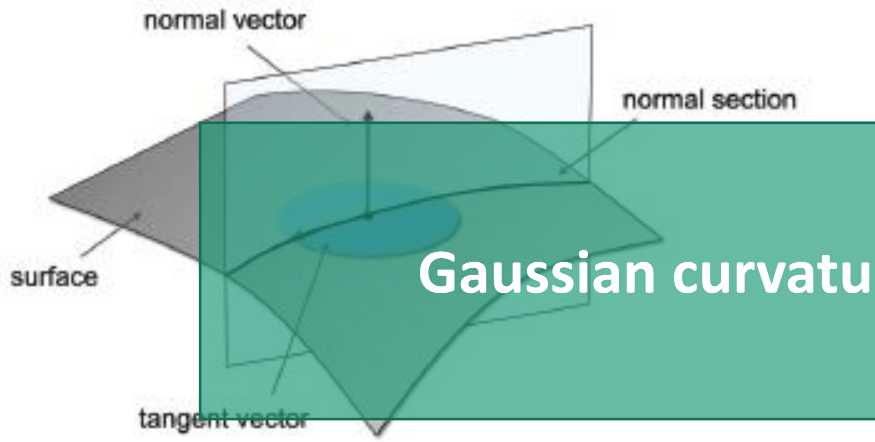


Why is it not a 2d Riemannian manifold?

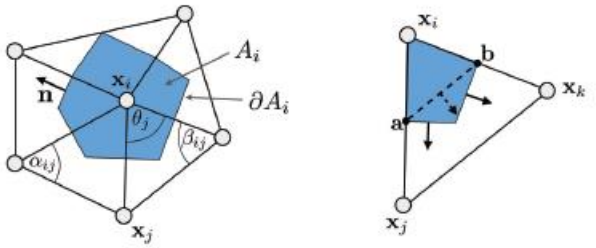


Pairwise correlation is a straight line and not a curve!

Next time, integrating curvature!



Gaussian curvature is an intrinsic property!



$$\Delta f = div\nabla f = f_{uu} + f_{vv}$$

$$K = \frac{< (\nabla_{e2}\nabla_{e1} - \nabla_{e1}\nabla_{e2})e_1, e_2 >}{\det g}$$

And, topological algebra does Deep Learning, too

Gauge Equivariant Convolutional Networks and the Icosahedral CNN

Taco S. Cohen [*1] Maurice Weiler [*2] Berkay Kicanaoglu [*2] Max Welling [1]

This does only work for scalar functions, what about wind directions?

Abstract

The principle of *equivariance to symmetry transformations* enables a theoretically grounded approach to neural network architecture design. Equivariant networks have shown excellent performance and data efficiency on vision and medical imaging problems that exhibit symmetries. Here we show how this principle can be extended beyond global symmetries to local gauge transformations. This enables the development of a very general class of convolutional neural networks on manifolds that depend only on the intrinsic geometry, and which includes many popular methods from equivariant and geometric deep learning.

We implement gauge equivariant CNNs for sig-

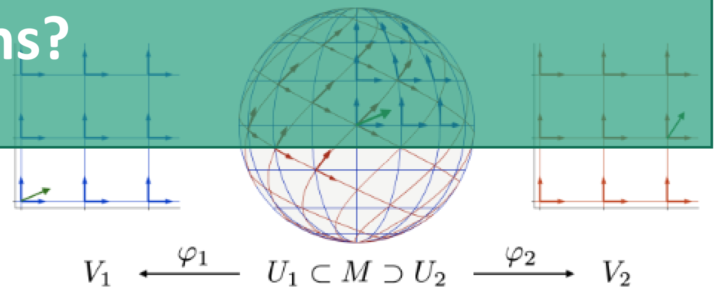
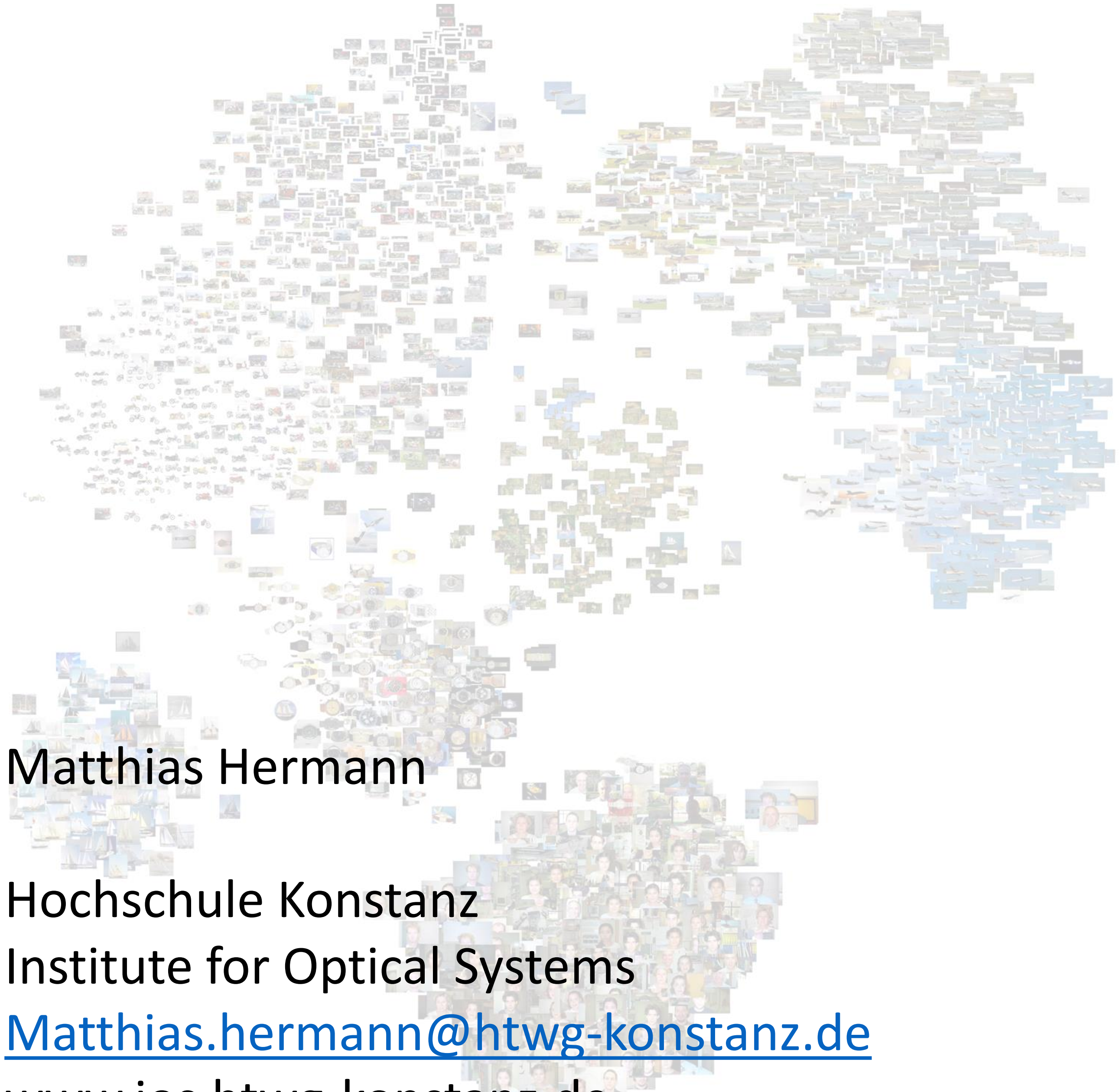


Figure 1. A gauge is a smoothly varying choice of tangent frame on a subset U of a manifold M. A gauge is needed to represent geometrical quantities such as convolutional filters and feature maps (i.e. fields), but the choice of gauge is ultimately arbitrary. Hence, the network should be equivariant to gauge transformations, such as the change between red and blue gauge pictured here.



Thanks for your attention!

Matthias Hermann

Hochschule Konstanz
Institute for Optical Systems
Matthias.hermann@htwg-konstanz.de
www.ios.htwg-konstanz.de

INSTITUT FÜR OPTISCHE SYSTEME

Literature

- *H. Bay; T. Tuytelaars & L. van Gool (2006).* ["SURF: Speeded Up Robust Features"]. *Proceedings of the 9th European Conference on Computer Vision, Springer LNCS volume 3951, part 1. pp. 404–417.*
- Kim, M. S., Chen, Y. R., Cho, B. K., Chao, K., Yang, C. C., Lefcourt, A. M., & Chan, D. (2007). Hyperspectral reflectance and fluorescence line-scan imaging for online defect and fecal contamination inspection of apples. *Sensing and Instrumentation for Food Quality and Safety*, *1*(3), 151.
- Tsai, I. S., & Hu, M. C. (1996). Automatic inspection of fabric defects using an artificial neural network technique. *Textile Research Journal*, *66*(7), 474-482.
- Xie, X. (2008). A review of recent advances in surface defect detection using texture analysis techniques. *ELCVIA: electronic letters on computer vision and image analysis*, *7*(3), 1-22.
- Aiger, D., & Talbot, H. (2012). The phase only transform for unsupervised surface defect detection. In *Emerging Topics In Computer Vision And Its Applications* (pp. 215-232).
- Zimek, A., Schubert, E., & Kriegel, H. P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *5*(5), 363-387.
- Marimont, R.B.; Shapiro, M.B. (1979). "Nearest Neighbour Searches and the Curse of Dimensionality". *IMA J Appl Math*. **24**(1): 59–70. doi:10.1093/imamat/24.1.59.
- Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001, January). On the surprising behavior of distance metrics in high dimensional space. In International conference on database theory (pp. 420-434). Springer, Berlin, Heidelberg.
- Raj, B. (2017). Introduction to Deep Learning. *Carnegie Mellon's School of Computer Science.*