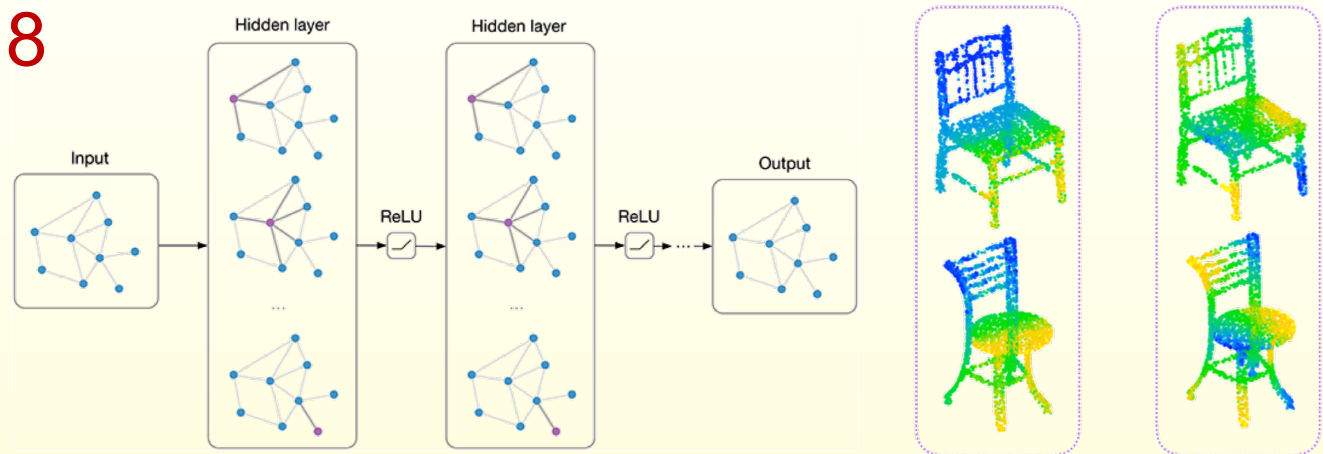
# CS233: Geometric and Topological Data Analysis

## Deep Nets for Graphs and Meshes

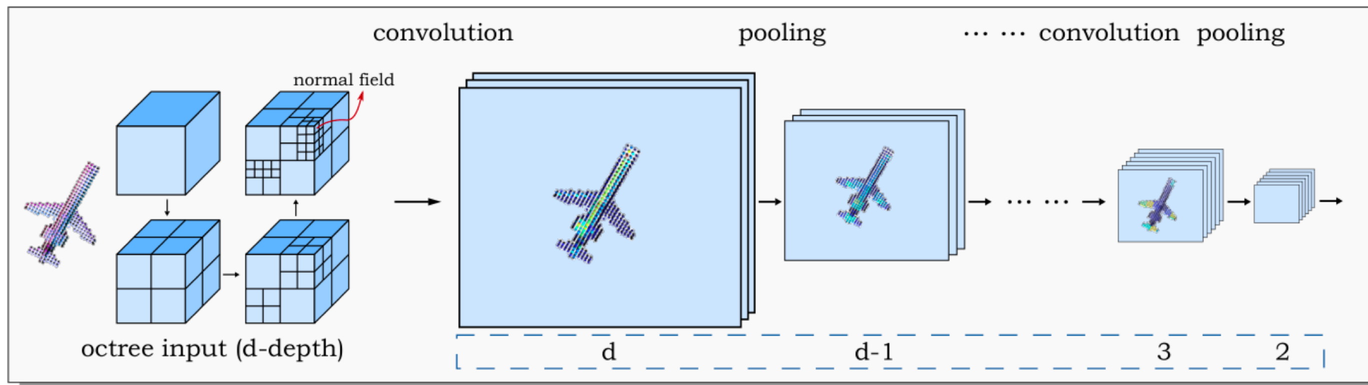
23 May 2018

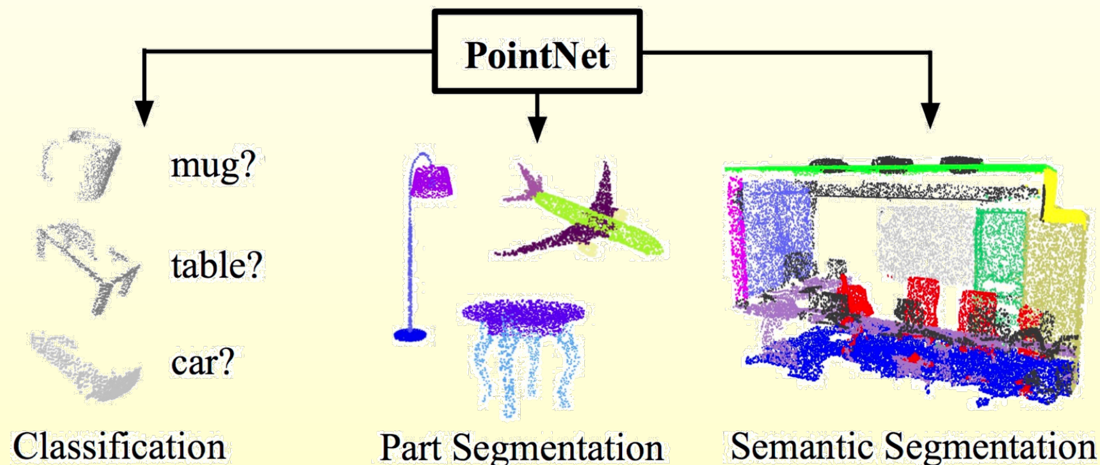



Slides ack: Thomas Kipf, Li (Eric) Yi, Michael Bronstein

# Last Time: Neural Nets for Volumes/Point Sets

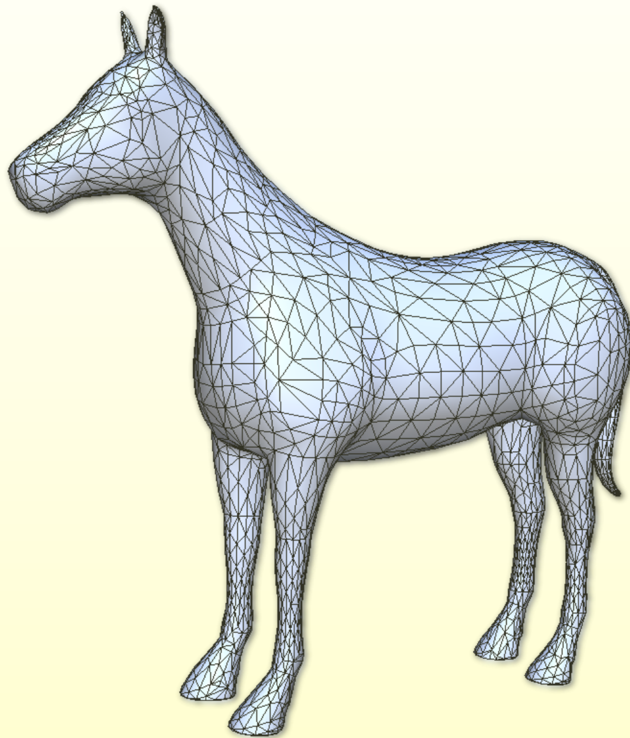
O-CNN

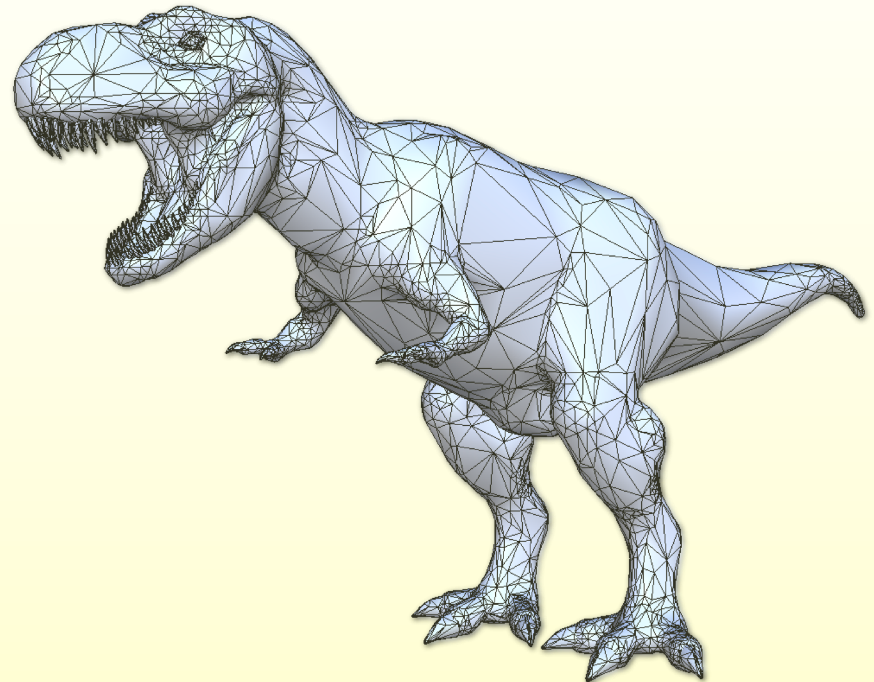


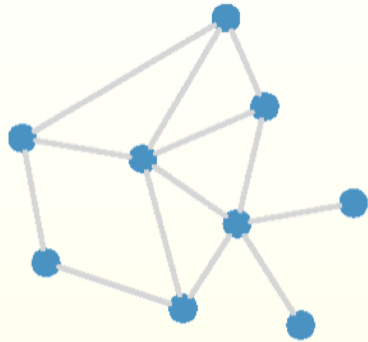
# Graph Representation

Graph representation is more common, and more effective to encode fine details.

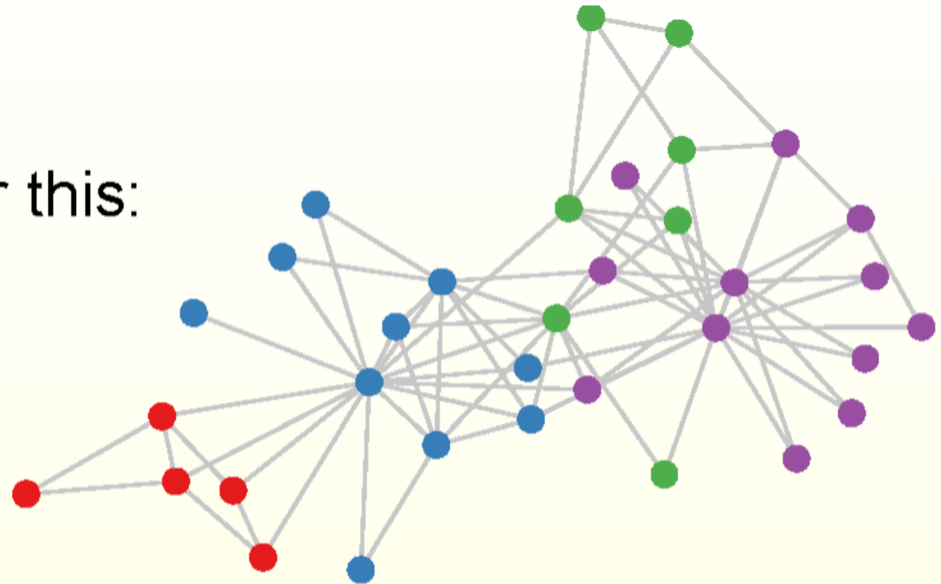




# Graph Representation



or this:



**Real-world examples:**

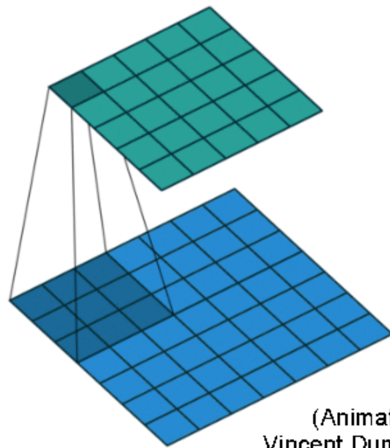
- Social networks
- World-wide-web
- Protein-interaction networks
- Telecommunication networks
- Knowledge graphs
- …

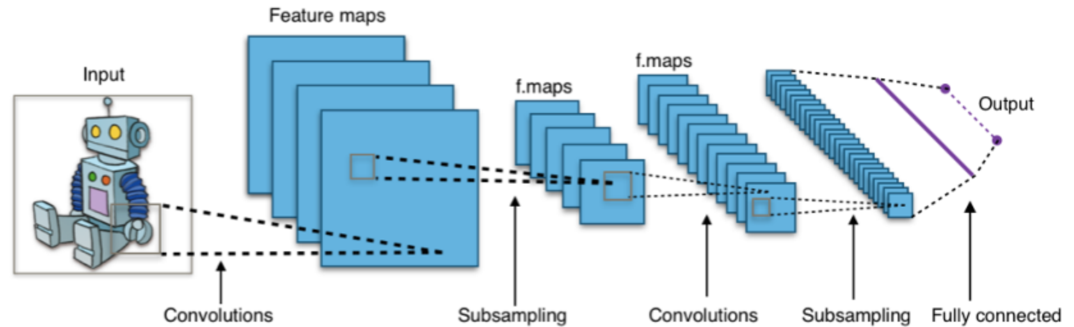
# Graph CNN

# Recap: Deep learning on Euclidean data

**We know how to deal with this:**

Convolutional neural networks (CNNs)


(Animation by Vincent Dumoulin)

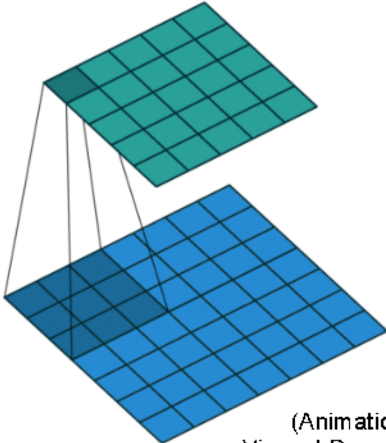


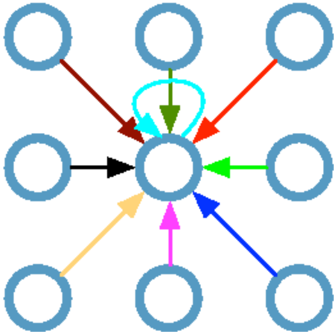
(Source: Wikipedia)

Deep neural nets exploit:
- Translational invariant (weight sharing)

# Convolutional neural networks (on grids)

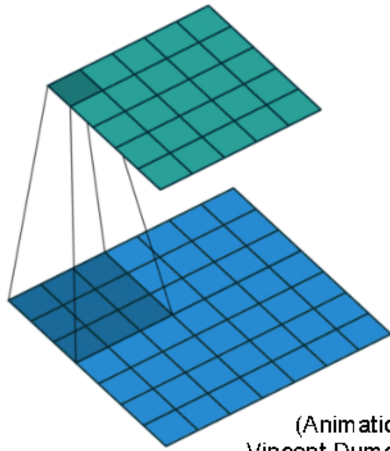
Single CNN layer with 3x3 filter:

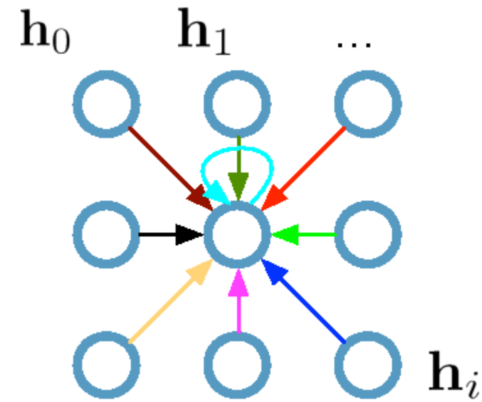


# Convolutional neural networks (on grids)

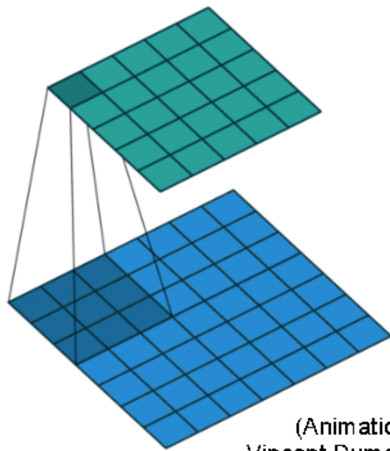
Single CNN layer with 3x3 filter:



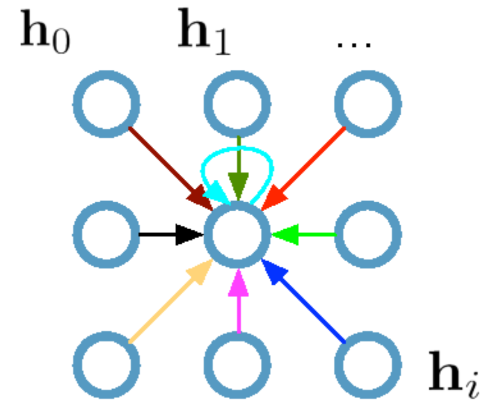



# Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)

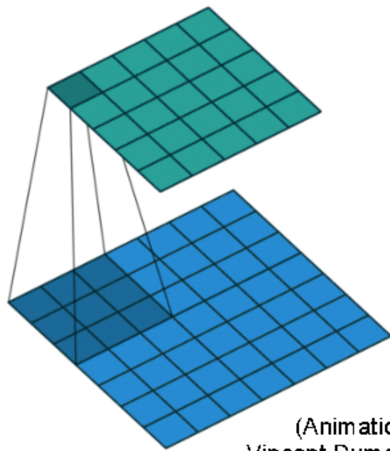



**Update for a single pixel:**

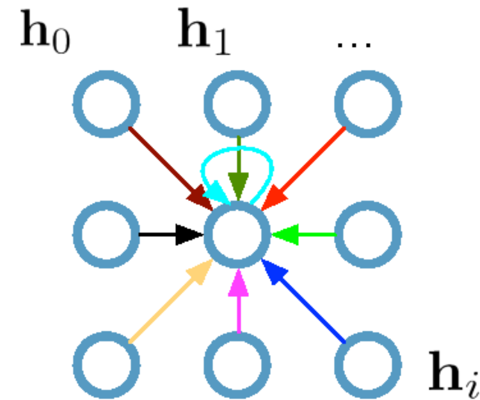
- Transform neighbors individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

# Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



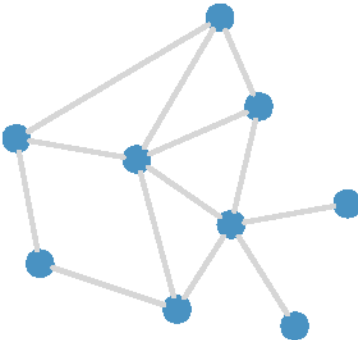

**Update for a single pixel:**

- Transform neighbors individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

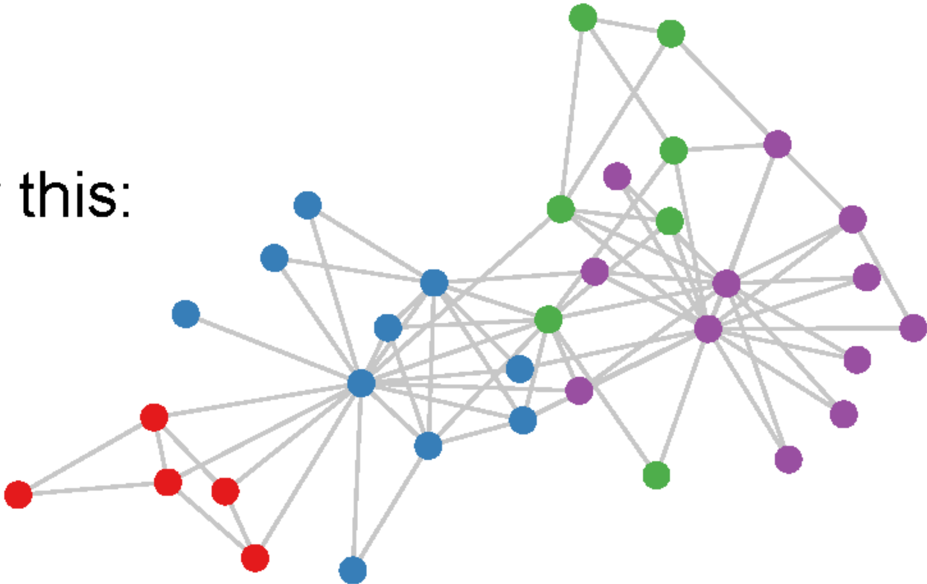
Full update: $\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$

# Graph-structured data

**What if our data looks like this?**

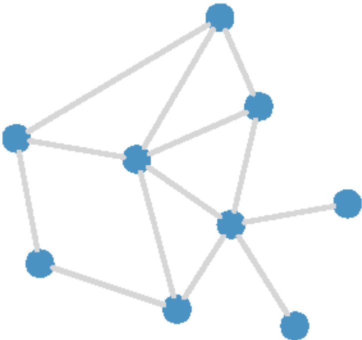


or this:

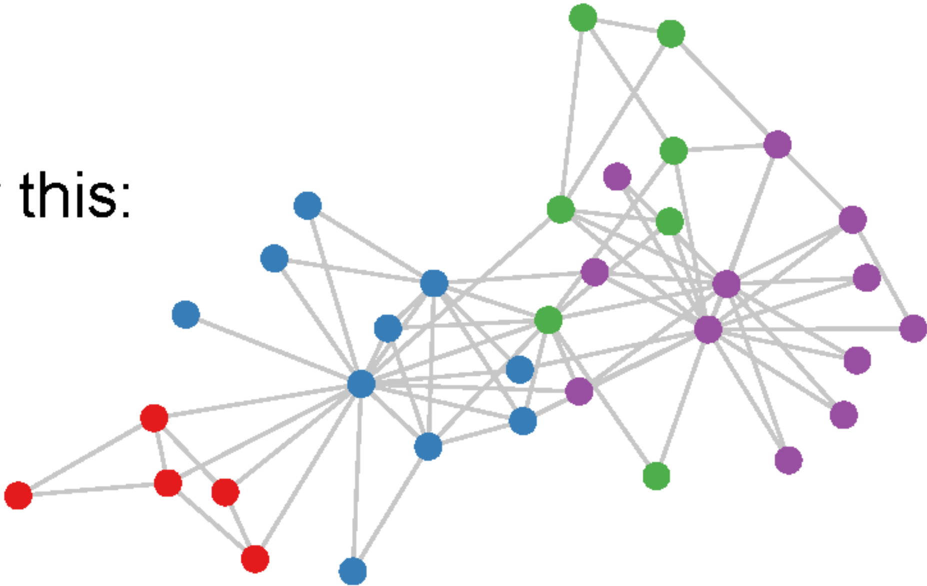


# Graph-structured data

**What if our data looks like this?**

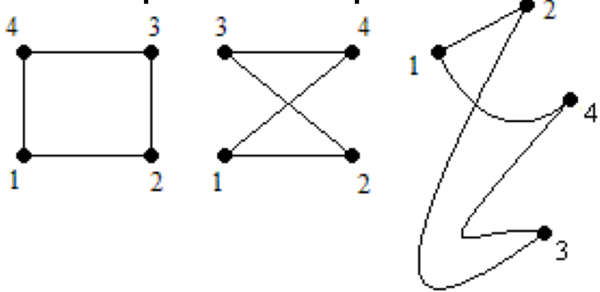


or this:



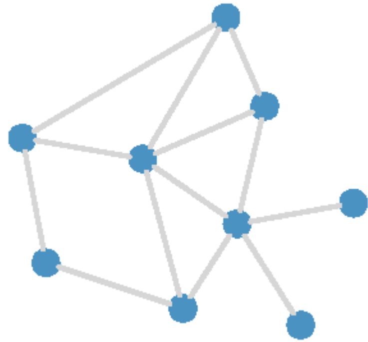
- No order in neighborhood.

Graph isomorphism

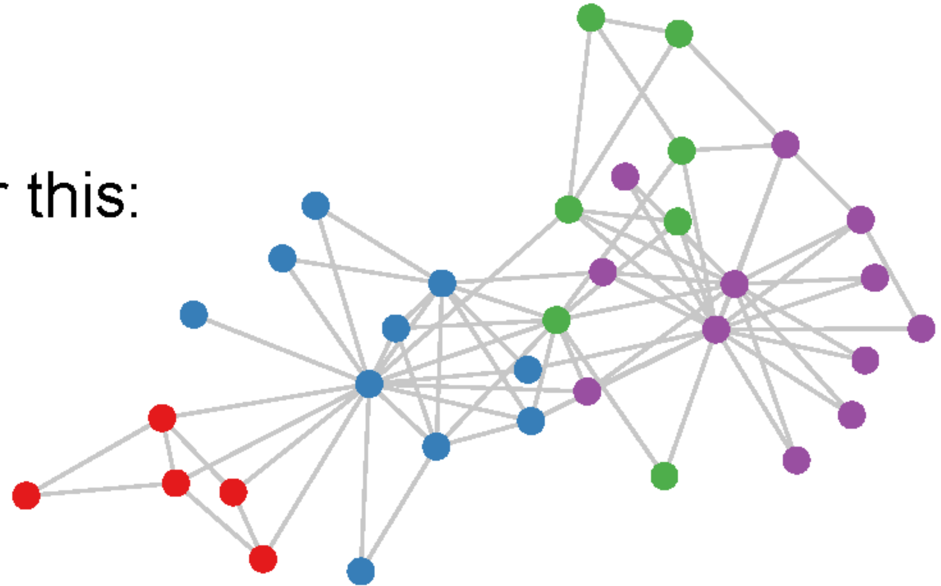


# Graph-structured data

**What if our data looks like this?**



or this:



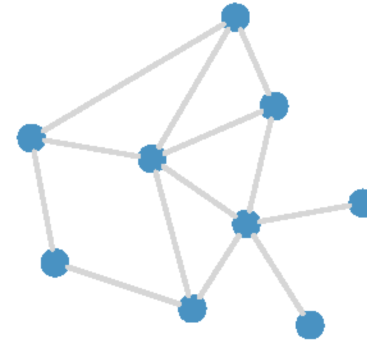
- No order in neighborhood.
- The numbers of neighbors are different.

# Graphs: Definitions

**Graph**: $G = (\mathcal{V}, \mathcal{E})$

$\mathcal{V}$ : Set of nodes $\{v_i\}$, $|\mathcal{V}| = N$

$\mathcal{E}$ : Set of edges $\{(v_i, v_j)\}$



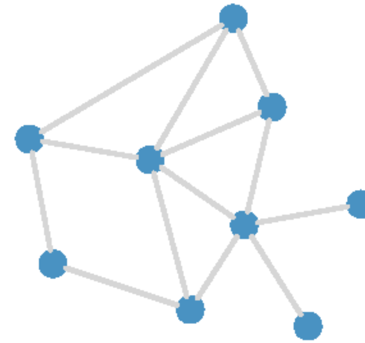
**We can define:**

$\mathbf{A}$ (adjacency matrix): $A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$

*(can also be weighted)*

# Graphs: Definitions

**Graph**: $G = (\mathcal{V}, \mathcal{E})$



$\mathcal{V}$ : Set of nodes $\{v_i\}$, $|\mathcal{V}| = N$

$\mathcal{E}$ : Set of edges $\{(v_i, v_j)\}$

**We can define:**

$\mathbf{A}$ (adjacency matrix): $A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$

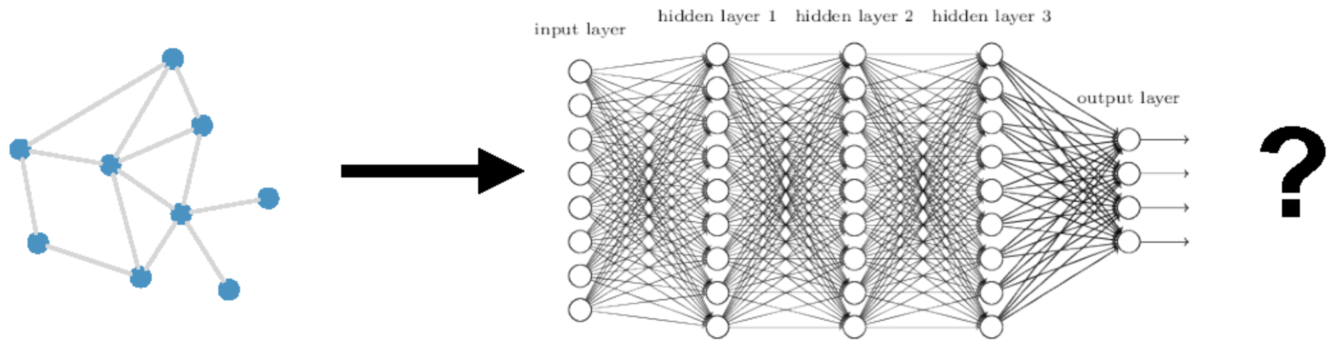
*(can also be weighted)*

**Model wish list:**

- Set of trainable parameters $\{\mathbf{W}^{(l)}\}$
- Trainable in $\mathcal{O}(|\mathcal{E}|)$ time
- Applicable even if the input graph changes

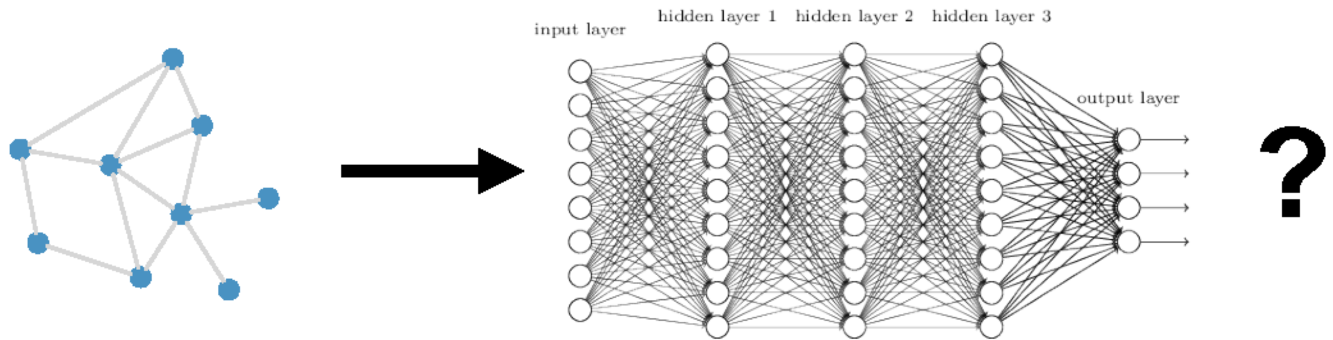
# A naive approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{X}, \mathbf{A}] \in \mathbb{R}^{N \times (N+E)}$
- Feed them into deep (fully connected) neural net
- Done?



# A naive approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{X}, \mathbf{A}] \in \mathbb{R}^{N \times (N+E)}$
- Feed them into deep (fully connected) neural net
- Done?




**Problems:**

- Huge number of parameters $\mathcal{O}(N)$
- Needs to be re-trained if number of nodes changes
- Does not generalize across graphs

# A naive approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{X}, \mathbf{A}] \in \mathbb{R}^{N \times (N+E)}$
- Feed them into deep (fully connected) neural net
- Done?

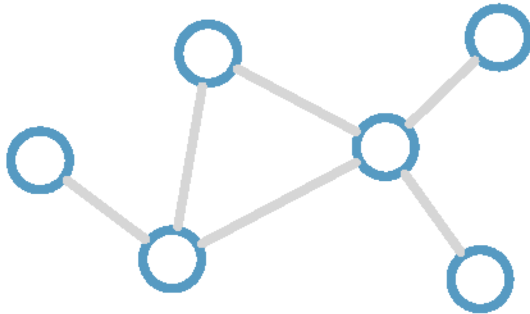
**Problems:**

- Huge number of parameters $\mathcal{O}(N)$
- Needs to be re-trained if number of nodes changes
- Does not generalize across graphs

# CNNs on graphs with spatial filters

(related idea was first proposed in Scarselli et al. 2009)

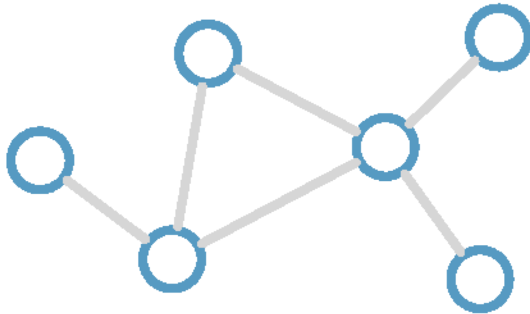
Consider this undirected graph:



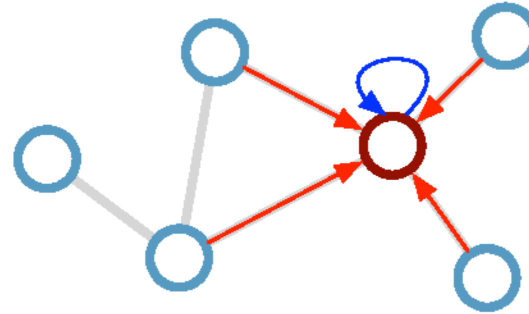
# CNNs on graphs with spatial filters

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:



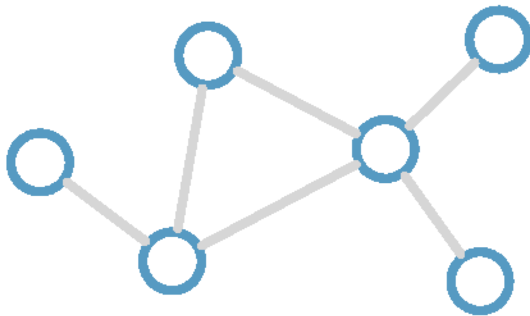
Calculate update for node in red:



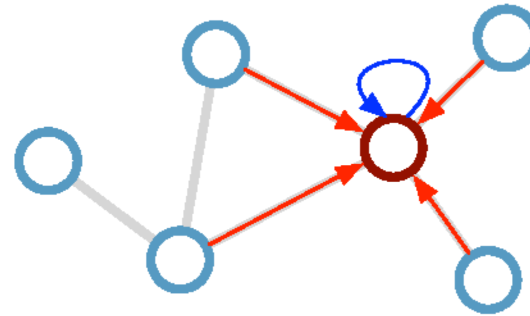
# CNNs on graphs with spatial filters

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:



Calculate update for node in red:



**Update rule:**

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

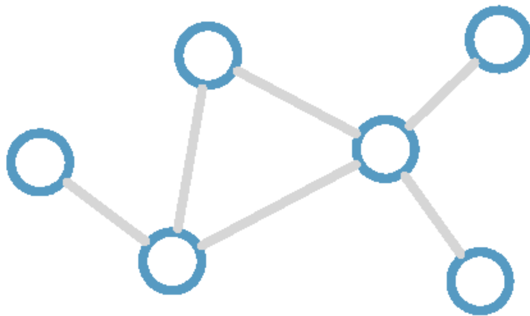
$\mathcal{N}_i$ : neighbor indices

$c_{ij}$ : norm. constant (per edge)

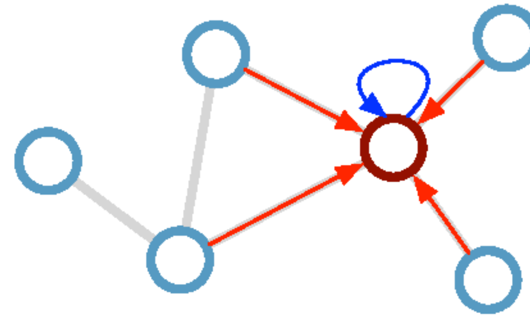
# CNNs on graphs with spatial filters

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:



Calculate update for node in red:



**Update rule:**

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

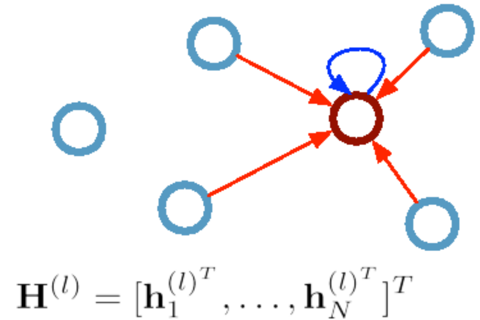
$\mathcal{N}_i$ : neighbor indices

$c_{ij}$ : norm. constant (per edge)

Apply the same transformation matrix for all neighbor nodes.

# Vectorized form

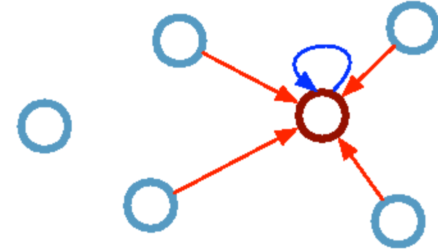
$$\underset{n\times k'}{\mathbf{H}^{(l+1)}} = \sigma\left(\underset{n\times k}{\mathbf{H}^{(l)}}\overset{k\times k'}{\mathbf{W}_0^{(l)}} + \overset{n\times n}{\tilde{\mathbf{A}}}\underset{n\times k}{\mathbf{H}^{(l)}}\overset{k\times k'}{\mathbf{W}_1^{(l)}}\right)$$



$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

# Vectorized form

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$



with $\tilde{A} = D^{-1}A$

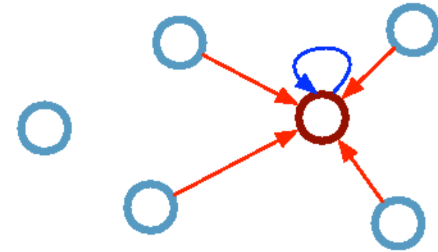
$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

$\boldsymbol{D}$ degree matrix: A diagonal matrix which contains information about the degree of each vertex.

# Vectorized form

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

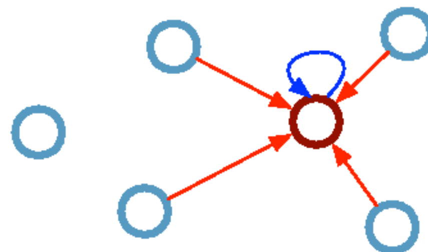
with $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$



$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

# Vectorized form

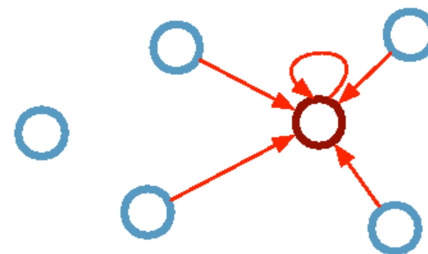
$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$



with $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

Or treat self-connection in the same way:



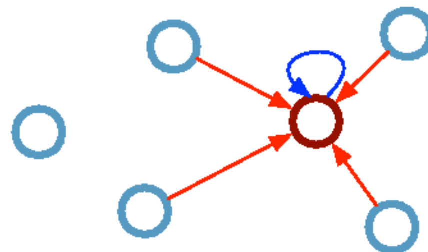
$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

with $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_N)\tilde{\mathbf{D}}^{-\frac{1}{2}}$

$$\tilde{D}_{ii} = \sum_j (A_{ij} + \delta_{ij})$$

# Vectorized form

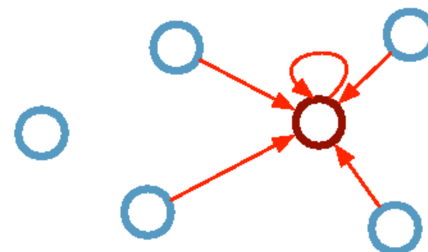
$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$



with $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

Or treat self-connection in the same way:



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

with $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_N)\tilde{\mathbf{D}}^{-\frac{1}{2}}$

$$\tilde{D}_{ii} = \sum_j (A_{ij} + \delta_{ij})$$

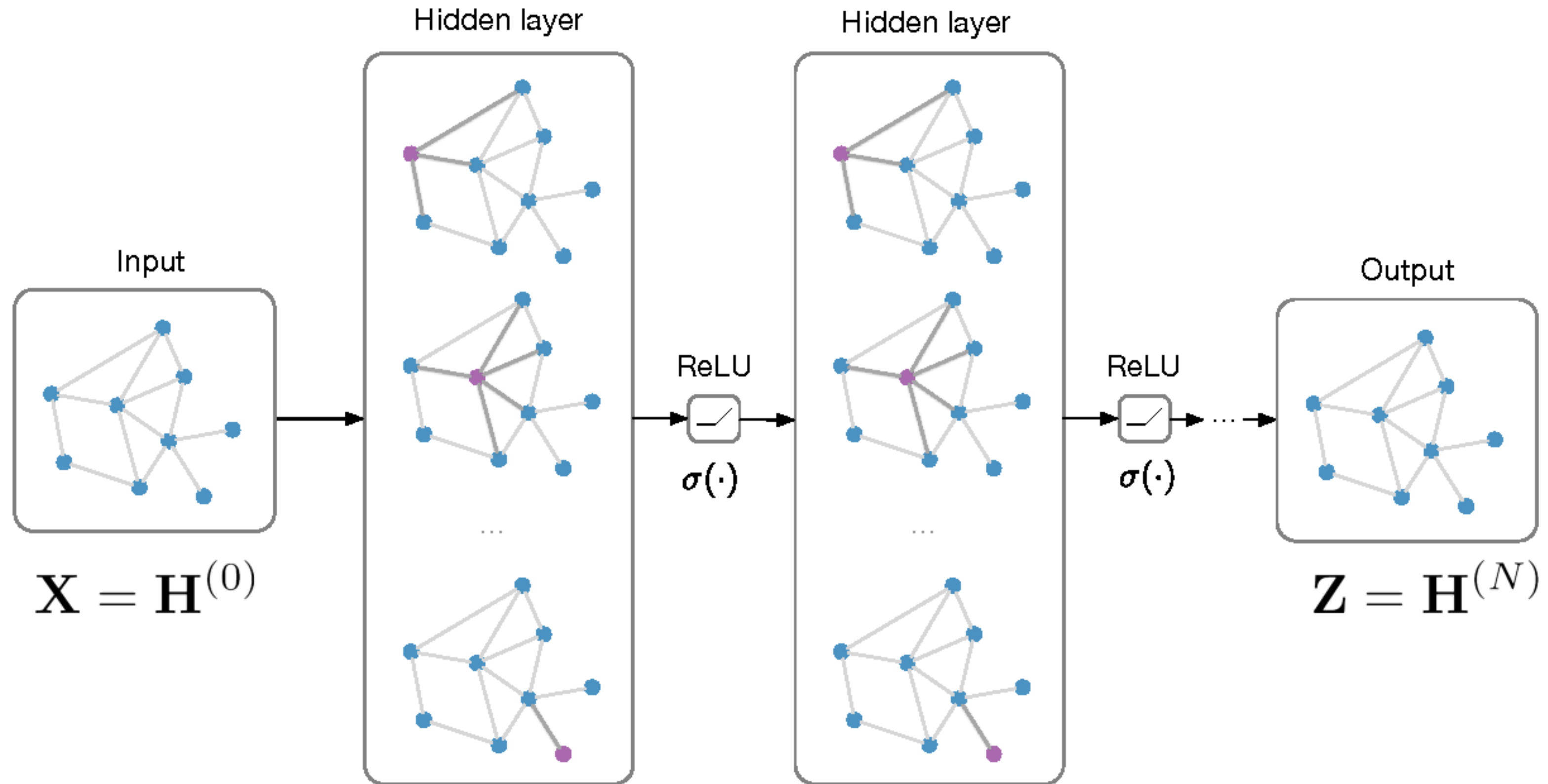
$\mathbf{A}$ is typically **sparse**

➡ We can use sparse matrix multiplications!

➡ Efficient $\mathcal{O}(|\mathcal{E}|)$ implementation in Theano or TensorFlow

# Model architecture (with spatial filters)

**Input**: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$ , preprocessed adjacency matrix $\hat{\mathbf{A}}$




$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

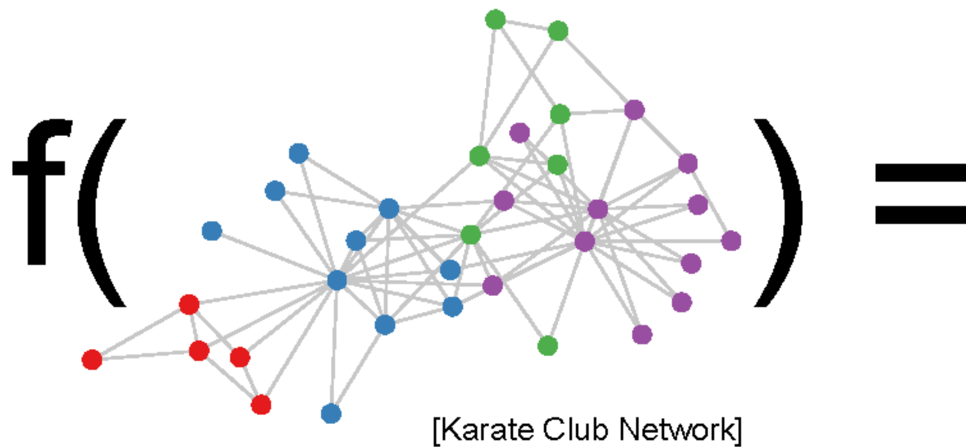
(or more sophisticated filters / basis functions)

[Kipf & Welling, ICLR 2017]

# What does it do? An example.

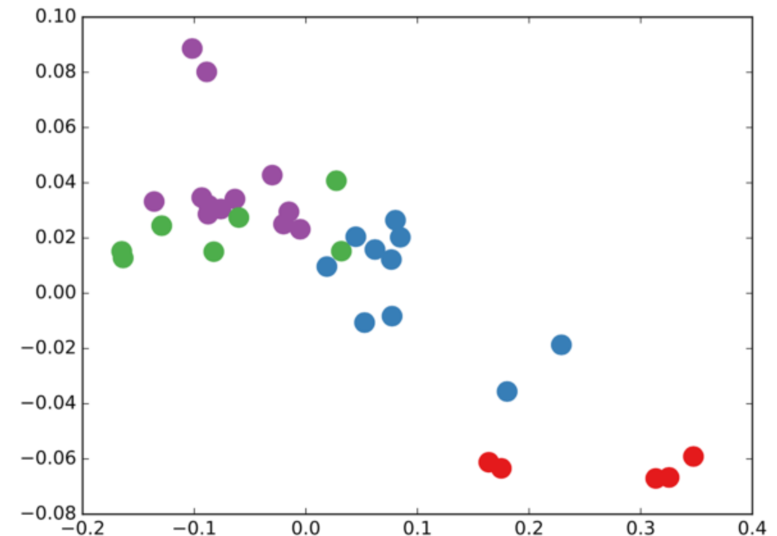
Forward pass through **untrained** 3-layer GCN model

Parameters initialized randomly

f(  ) =

[Karate Club Network]

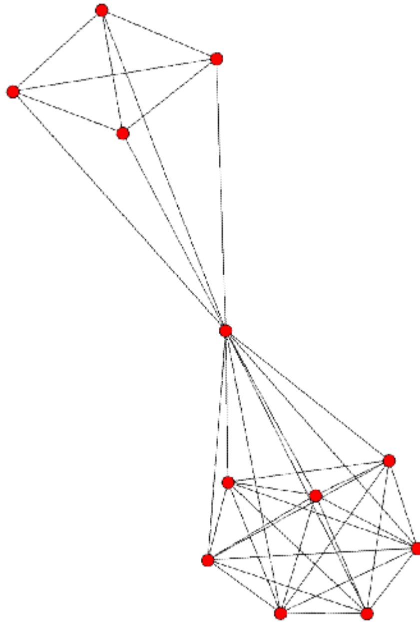
2-dim output per node




**Produces (useful?) random embeddings!**

# Connection to Weisfeiler-Lehman algorithm

A "classical" approach for node feature assignment



---

**Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)**

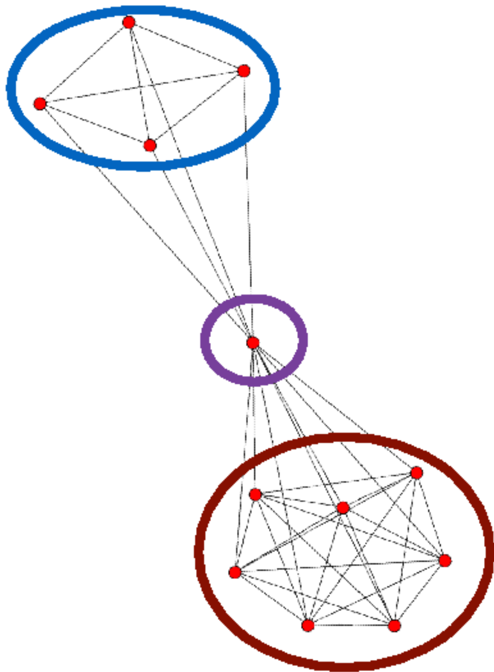
---

**Input**: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
**Output**: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
t $\leftarrow$ 0;
**repeat**
  **for** $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \mathrm{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t + 1$;
**until** *stable node coloring is reached*;

---

# Connection to Weisfeiler-Lehman algorithm

A "classical" approach for node feature assignment



**Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)**

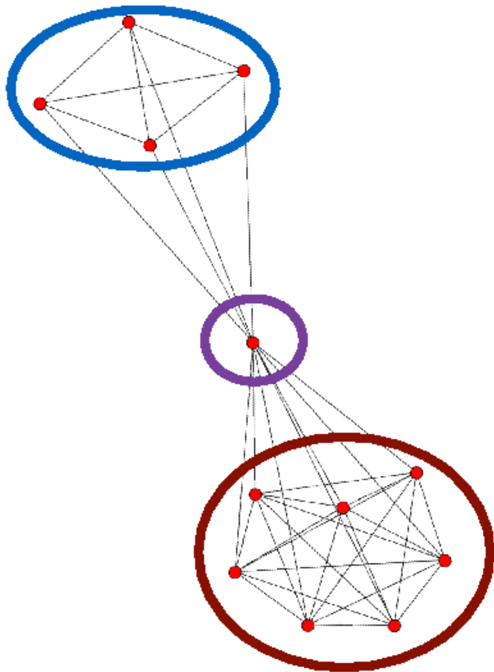
**Input**: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
**Output**: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
t $\leftarrow$ 0;
**repeat**
  **for** $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t + 1$;
**until** *stable node coloring is reached*;

Useful as graph isomorphism check for *most* graphs

(exception: highly regular graphs)

# Connection to Weisfeiler-Lehman algorithm

A "classical" approach for node feature assignment



**Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)**

**Input**: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$

**Output**: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$

t $\leftarrow$ 0;

**repeat**

  **for** $v_i \in \mathcal{V}$ **do**

    ~~$h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;~~

**un**

**GCN:** $$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} \mathbf{W}_1^{(l)}\right)$$

Useful as graph isomorphism check for *most* graphs

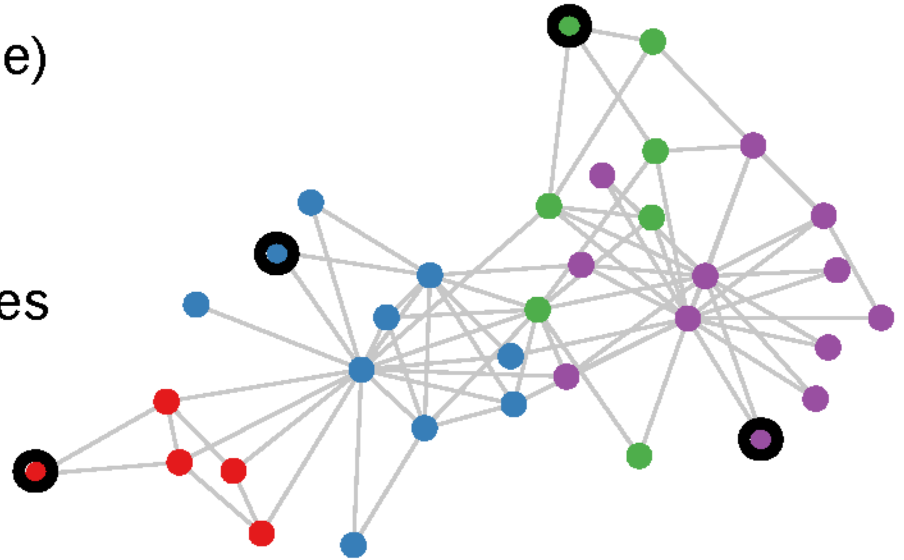
(exception: highly regular graphs)

# Semi-supervised classification on graphs

**Setting:**

Some nodes are labeled (black circle)
All other nodes are unlabeled

**Task:**
Predict node label of unlabeled nodes

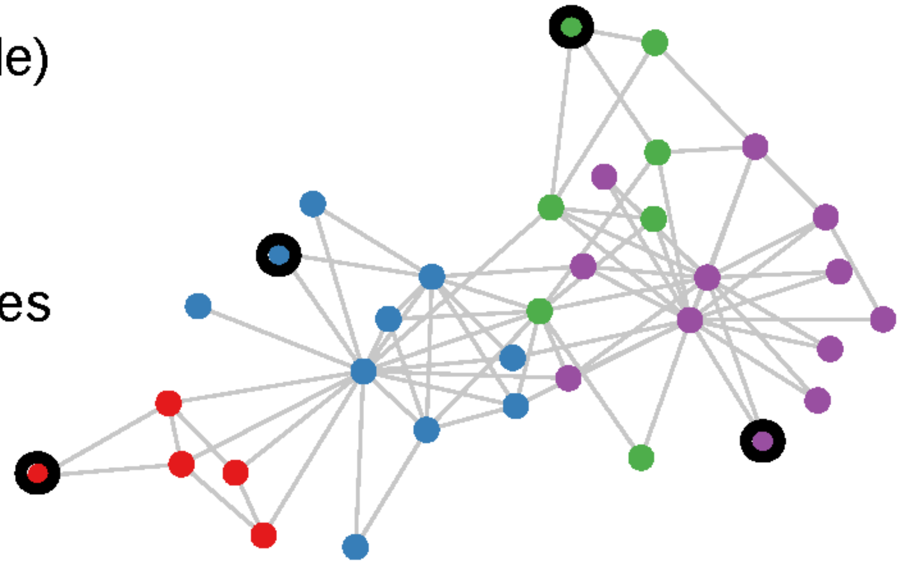


# Semi-supervised classification on graphs

**Setting:**

Some nodes are labeled (black circle)
All other nodes are unlabeled

**Task:**
Predict node label of unlabeled nodes



**Standard approach:**

graph-based regularization (“smoothness constraints”) [Zhu et al., 2003]

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}} \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2$$

assumes: connected nodes likely to share same label

# Semi-supervised classification on graphs

**Embedding-based approaches**

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

**Examples:** DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

# Semi-supervised classification on graphs

**Embedding-based approaches**

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

**Examples:** DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

**Problem**: Embeddings are not optimized for classification!

# Semi-supervised classification on graphs

**Embedding-based approaches**

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

**Examples:** DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

**Problem**: Embeddings are not optimized for classification!

**Idea:** Train graph-based classifier end-to-end using GCN

Evaluate loss on labeled nodes only:

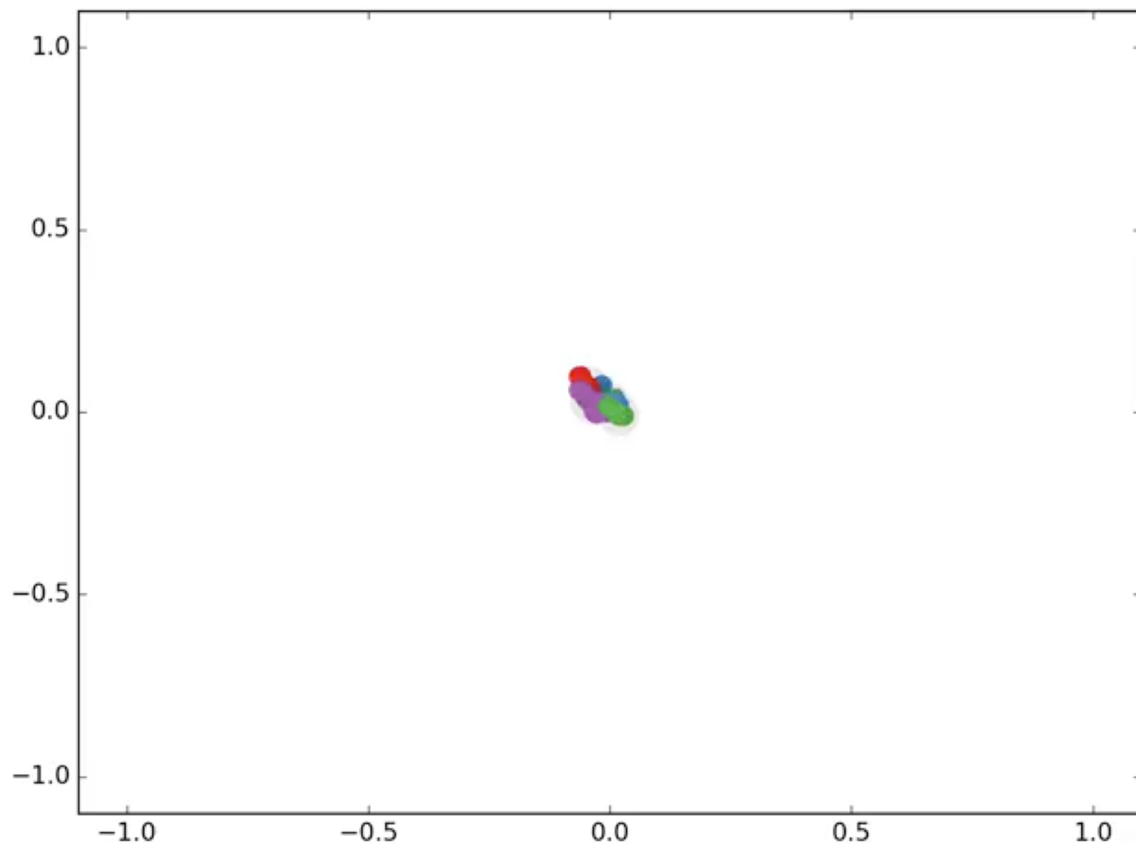
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

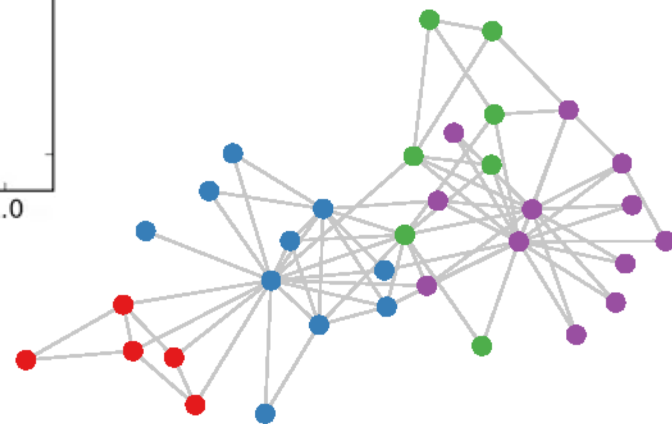
$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

# Toy example (semi-supervised learning)



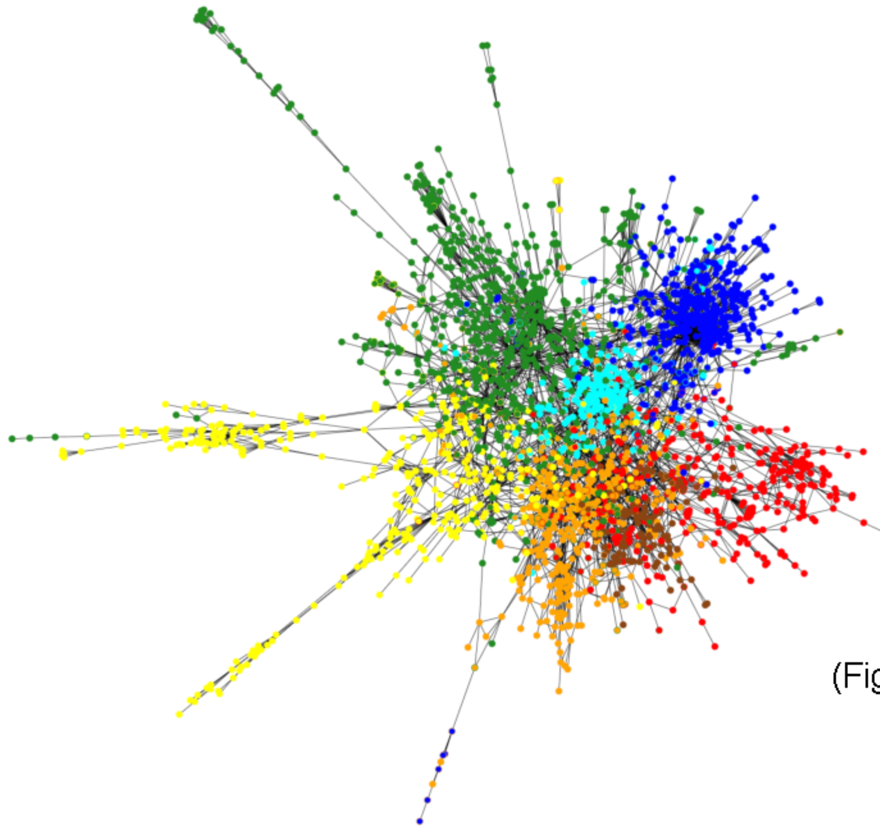



Video also available here:
http://tkipf.github.io/graph-convolutional-networks

# Classification on citation networks

**Input**: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

**Target**: Paper category (e.g. stat.ML, cs.LG, …)



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

# Experiments and results

**Model**: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$

**Dataset statistics**

| Dataset | Type | Nodes | Edges | Classes | Features | Label rate |
|---|---|---|---|---|---|---|
| Citeseer | Citation network | 3,327 | 4,732 | 6 | 3,703 | 0.036 |
| Cora | Citation network | 2,708 | 5,429 | 7 | 1,433 | 0.052 |
| Pubmed | Citation network | 19,717 | 44,338 | 3 | 500 | 0.003 |
| NELL | Knowledge graph | 65,755 | 266,144 | 210 | 5,414 | 0.001 |

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)

# Experiments and results

**Model**: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$

**Dataset statistics**

| Dataset | Type | Nodes | Edges | Classes | Features | Label rate |
|---|---|---|---|---|---|---|
| Citeseer | Citation network | 3,327 | 4,732 | 6 | 3,703 | 0.036 |
| Cora | Citation network | 2,708 | 5,429 | 7 | 1,433 | 0.052 |
| Pubmed | Citation network | 19,717 | 44,338 | 3 | 500 | 0.003 |
| NELL | Knowledge graph | 65,755 | 266,144 | 210 | 5,414 | 0.001 |

**Classification results (accuracy)**

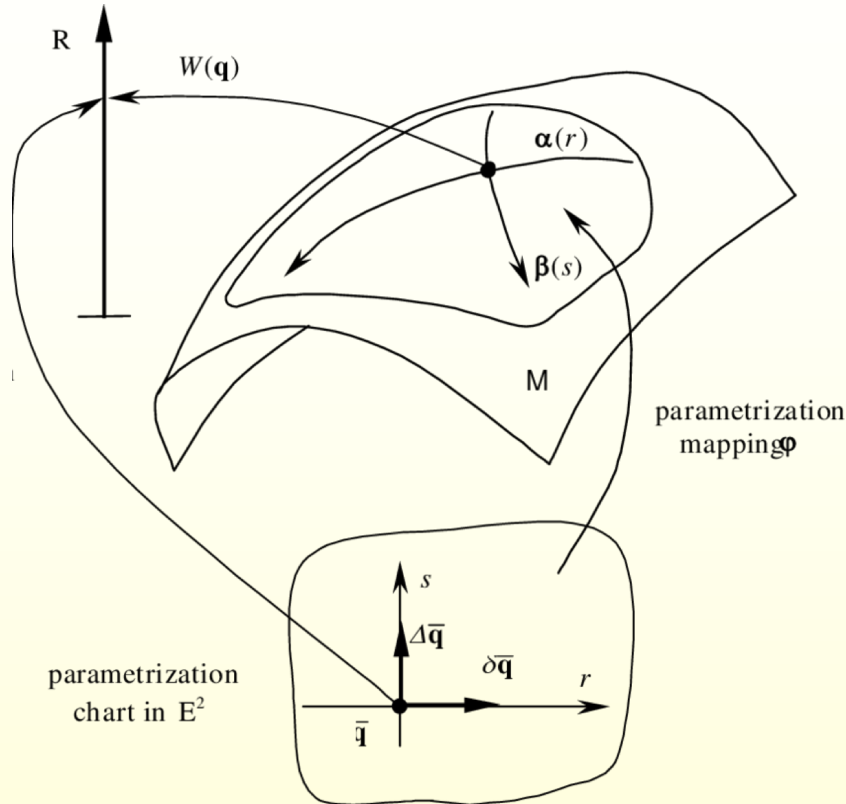
no input features → LP [27], DeepWalk [18]

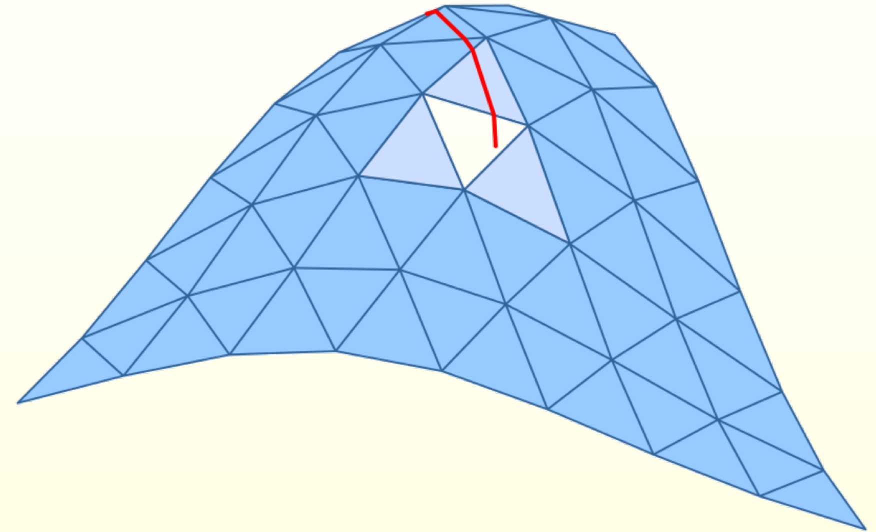
| Method | Citeseer | Cora | Pubmed | NELL |
|---|---|---|---|---|
| ManiReg [3] | 60.1 | 59.5 | 70.7 | 21.8 |
| SemiEmb [24] | 59.6 | 59.0 | 71.1 | 26.7 |
| LP [27] | 45.3 | 68.0 | 63.0 | 26.5 |
| DeepWalk [18] | 43.2 | 67.2 | 65.3 | 58.1 |
| Planetoid* [25] | 64.7 (26s) | 75.7 (13s) | 77.2 (25s) | 61.9 (185s) |
| **GCN** (this paper) | **70.3** (7s) | **81.5** (4s) | **79.0** (38s) | **66.0** (48s) |
| GCN (rand. splits) | $67.9 \pm 0.5$ | $80.1 \pm 0.5$ | $78.9 \pm 0.7$ | $58.4 \pm 1.7$ |

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)

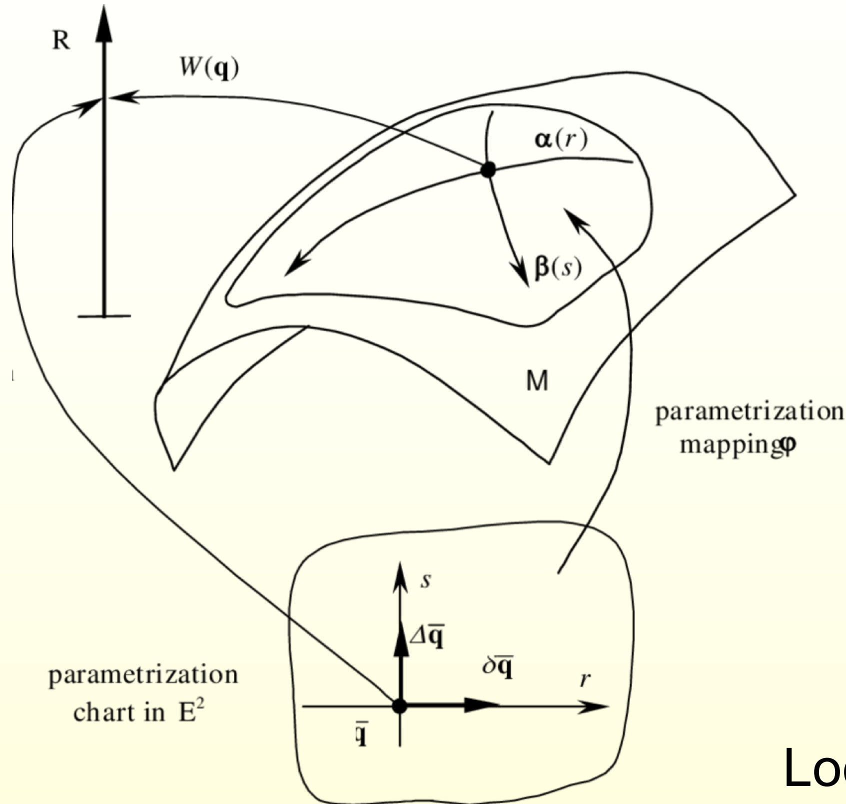
# Geodesic CNN

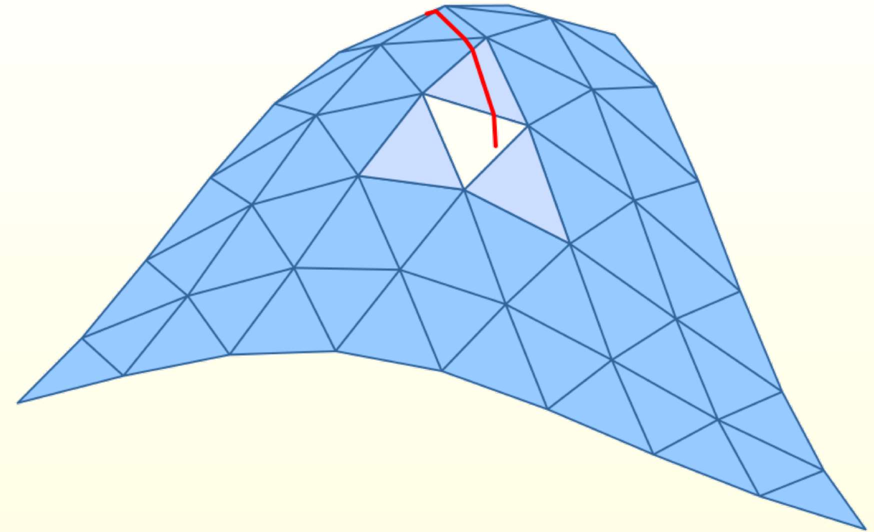
# Convolution on Manifold



# Convolution on Manifold

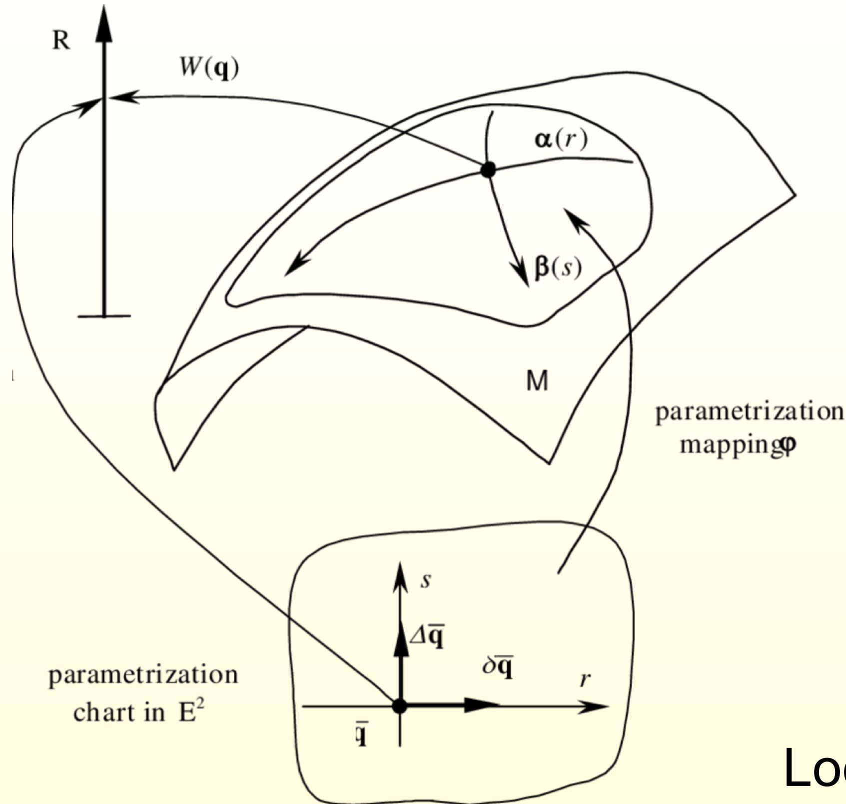


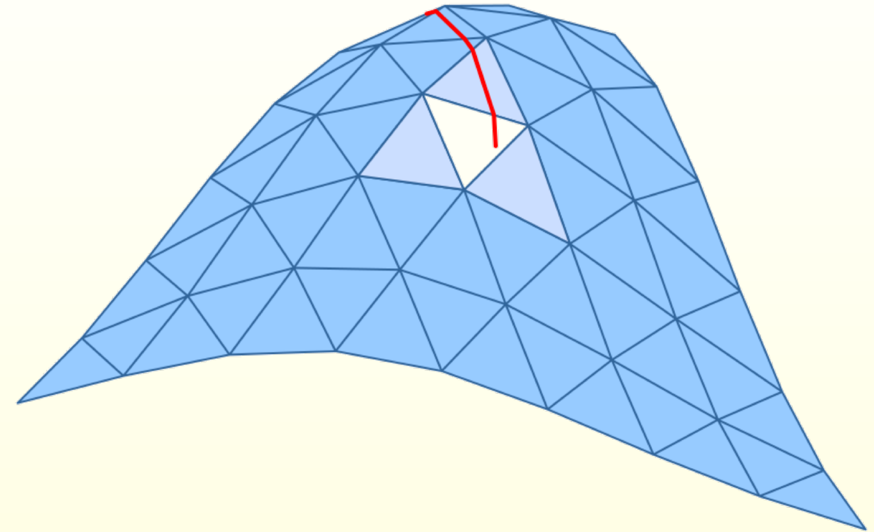



Locally plane

- Can define orientations (circular order).

# Convolution on Manifold

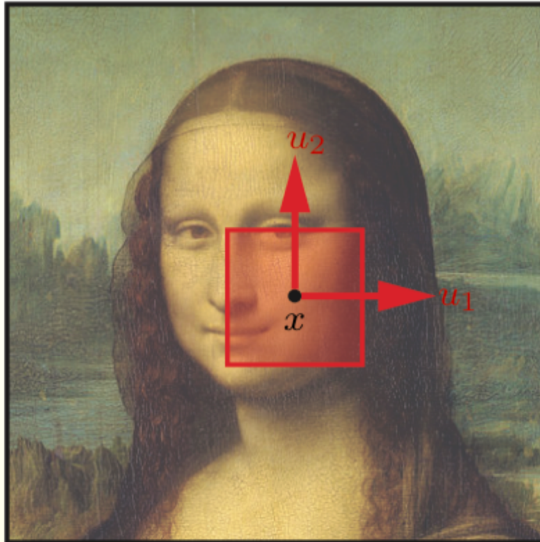


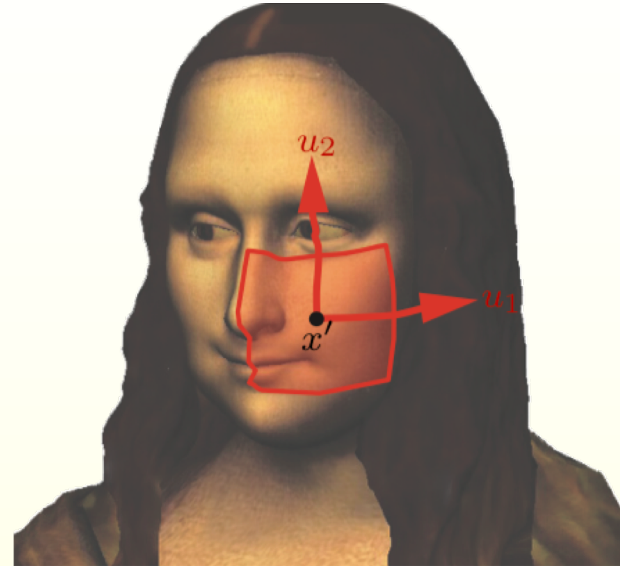
Locally plane

- Can define orientations
- Can compute distances on the surface.

# Patch Operator




Image




Manifold

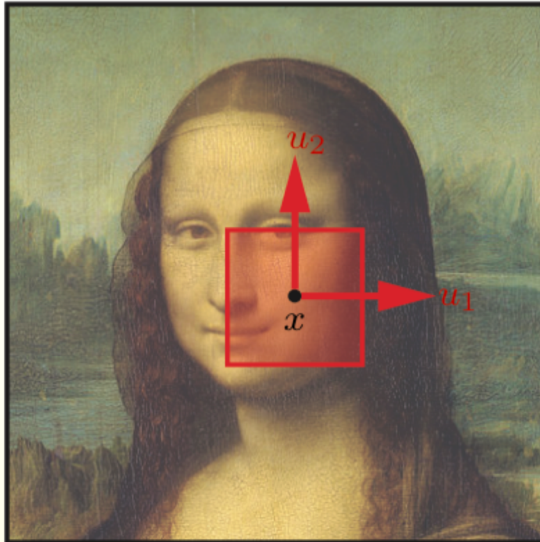
- Local system of coordinates: bijection $\varsigma_x : B_{\rho_0}(x) \to [0,1]^2$
- Patch operator $\mathcal{D} : L^2(\mathcal{X}) \to L^2([0,1]^2)$ mapping $f$ around $x$

$$(\mathcal{D}(x)f)(\mathbf{u}) = (f \circ \varsigma_x^{-1})(\mathbf{u})$$

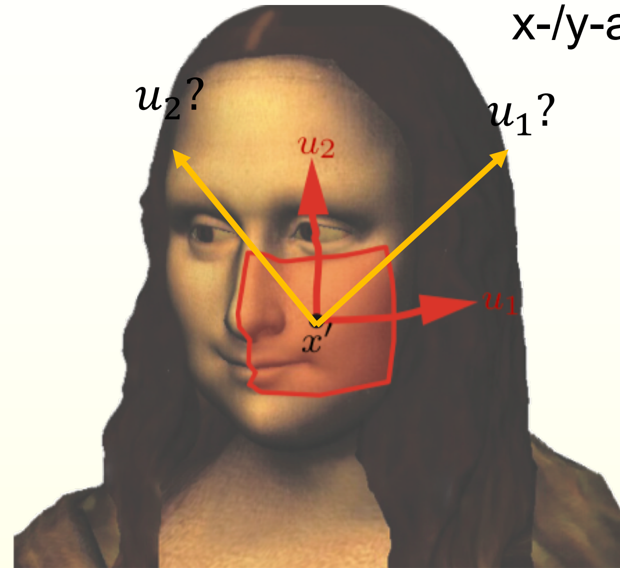
Boscaini et al. 2015

# Patch Operator

How to specify x-/y-axis?




Image




Manifold

- Local system of coordinates: bijection $\varsigma_x : B_{\rho_0}(x) \to [0,1]^2$
- Patch operator $\mathcal{D} : L^2(\mathcal{X}) \to L^2([0,1]^2)$ mapping $f$ around $x$

$$(\mathcal{D}(x)f)(\mathbf{u}) = (f \circ \varsigma_x^{-1})(\mathbf{u})$$

Boscaini et al. 2015

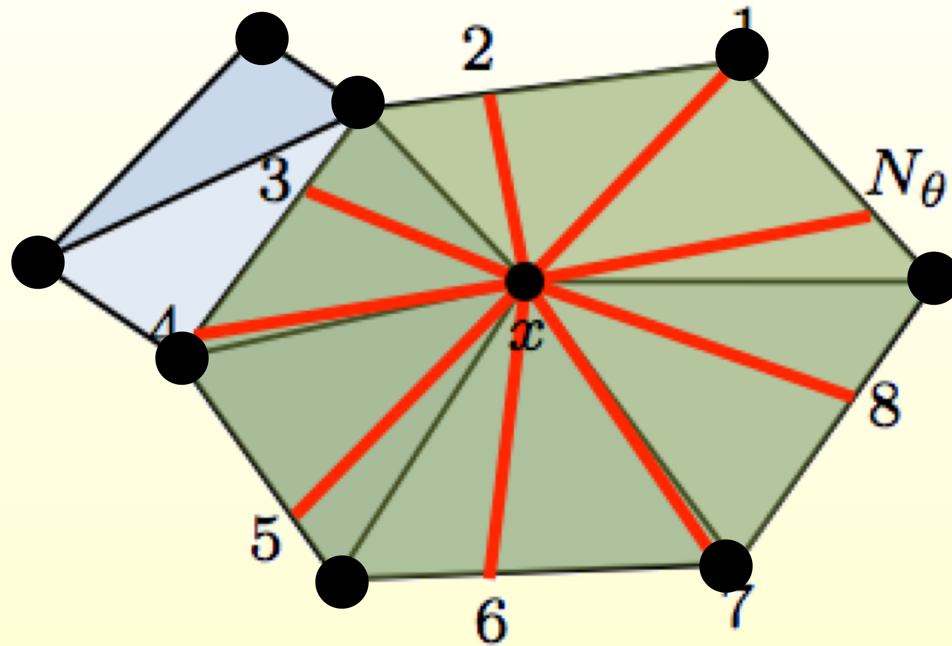
# Geodesic CNN

- Constructing convolution kernels:
  - Local system of geodesic ***polar*** coordinate.
  - Radial coordinate $\rho$ - geodesic distance (truncated)
  - Angular coordinate $\theta$ - direction of geodesics (origin choice)

Jonathan Masci et al 2015

# Geodesic CNN

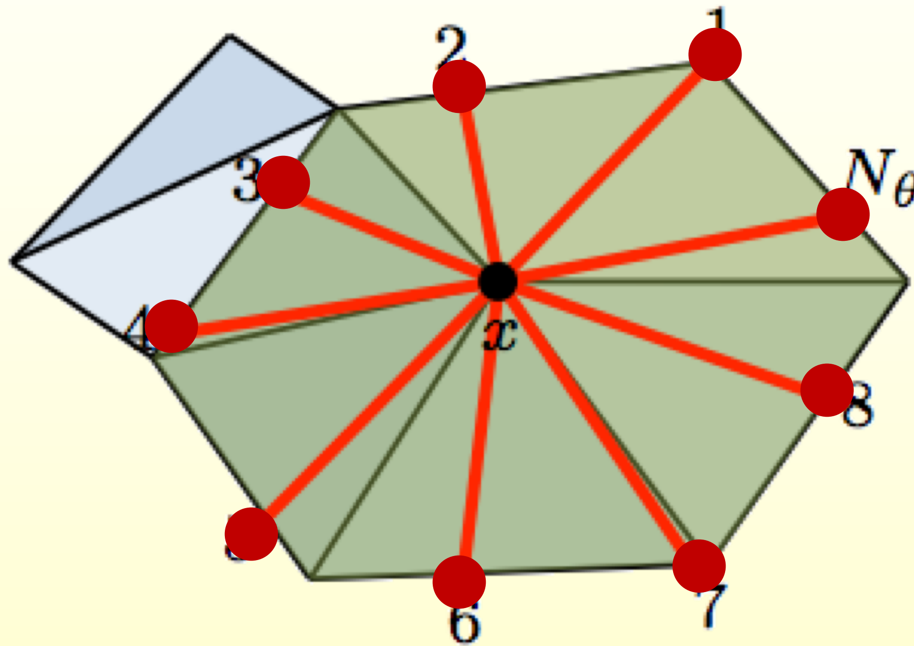
Output: features on *vertices*




Jonathan Masci et al 2015

# Geodesic CNN

Input: features on *polar grid*

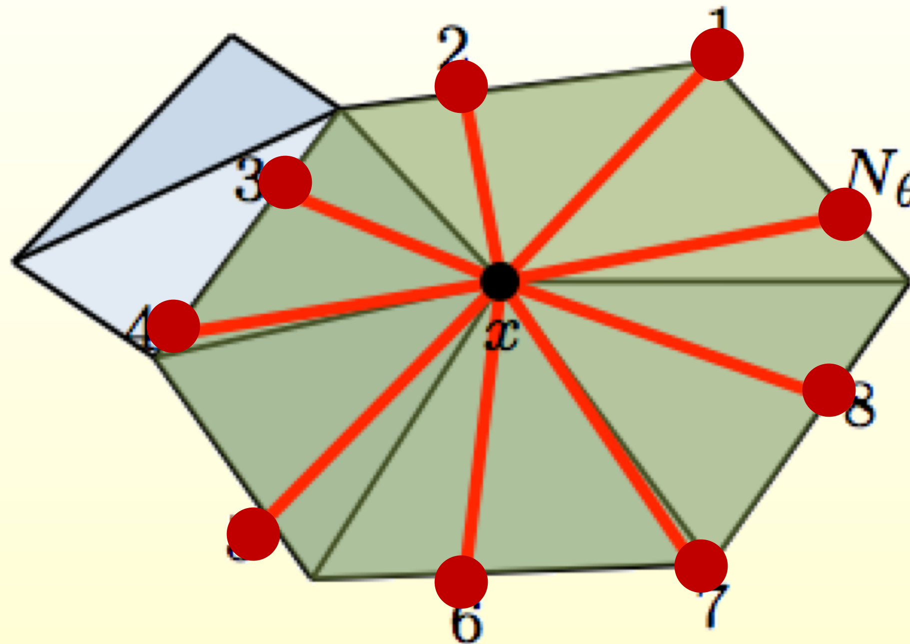



Jonathan Masci et al 2015

# Geodesic CNN

Input: features on *polar grid*

*Approximate from vertex features.*




Jonathan Masci et al 2015

# Geodesic CNN

$$(D(x)f)(\rho,\theta) = \frac{\int_X v_\rho(x,\xi)v_\theta(x,\xi)f(\xi)d\xi}{\int_X v_\rho(x,\xi)v_\theta(x,\xi)d\xi}$$

Radial weight

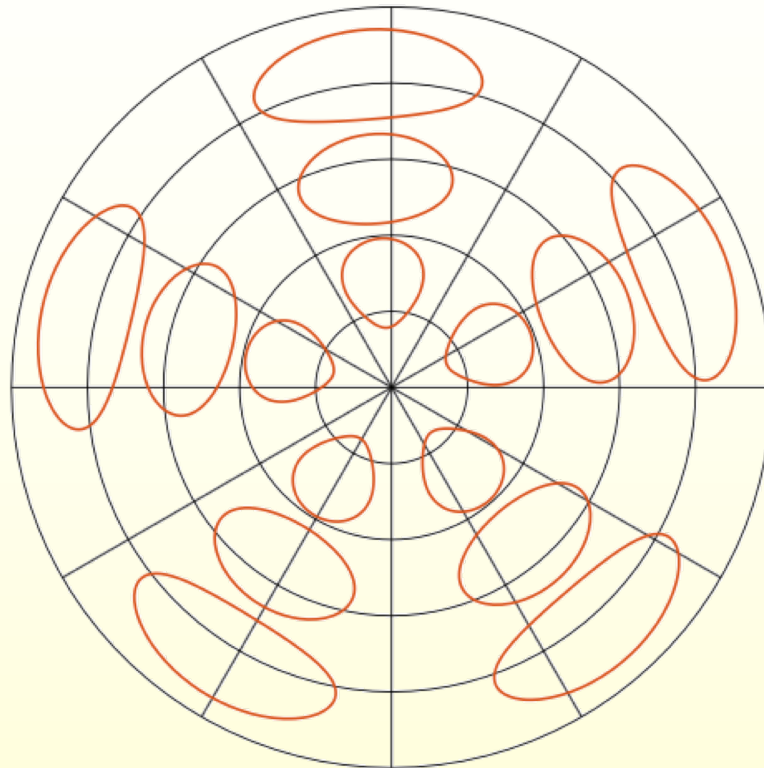
$$v_\rho(x,\xi) \propto e^{-(d_X(x,\xi)-\rho)^2/\sigma_\rho^2}$$

Angular weight

$$v_\theta(x,\xi) \propto e^{-d_X^2(\Gamma(x,\theta),\xi)/\sigma_\theta^2}$$

Jonathan Masci et al 2015

# Geodesic CNN

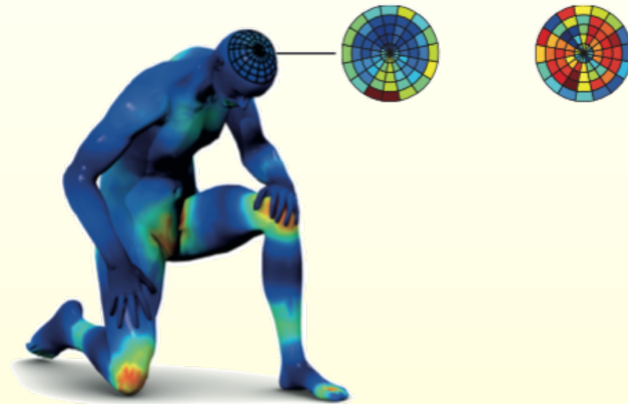


Weighting functions of the geodesic polar patch operator shown in $(\rho, \theta)$ coordinates (contours mark the $\frac{1}{2}$-level set)

# Geodesic CNN

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

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

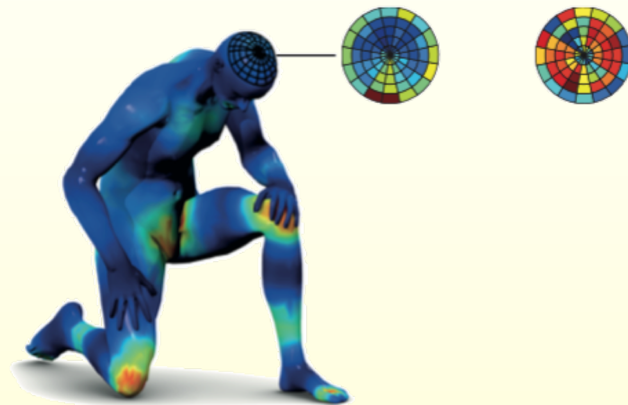


Jonathan Masci et al 2015

# Geodesic CNN

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

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



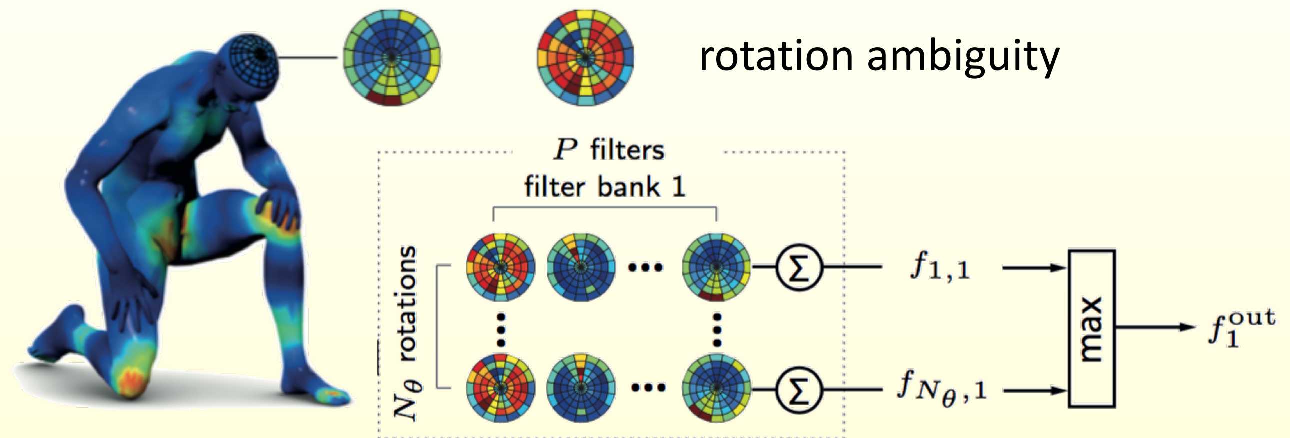
rotation ambiguity

Jonathan Masci et al 2015

# Geodesic CNN

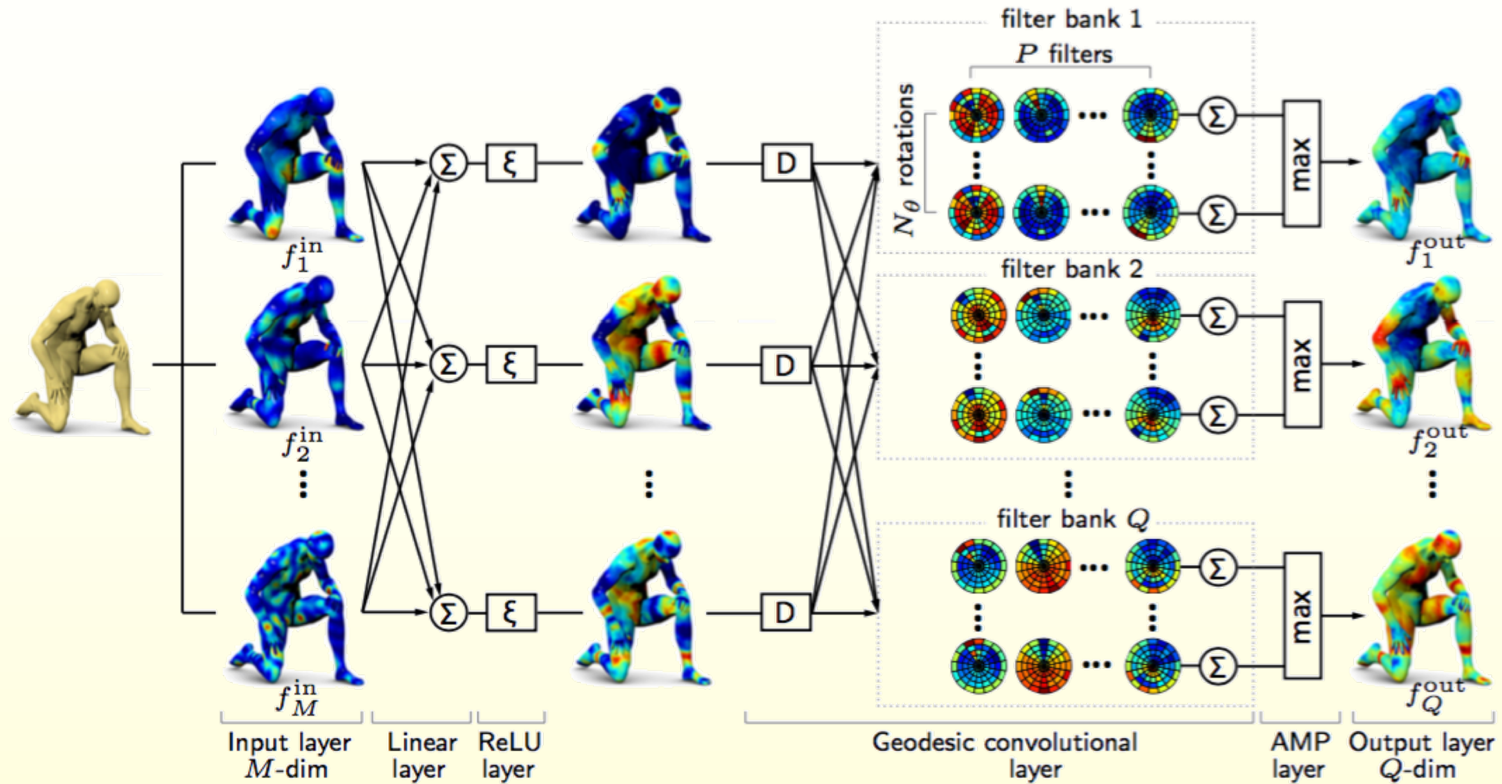
- Geodesic convolution = apply filter $a$ to patches extracted from $f \in L^2(X)$ in local geodesic polar coordinates

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




Jonathan Masci et al 2015

# Geodesic CNN



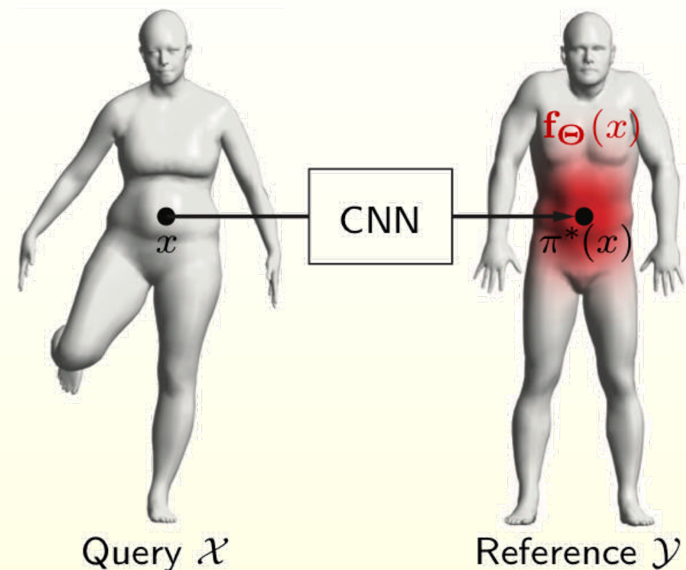

Jonathan Masci et al 2015

# Learning deformation-invariant correspondence

- Groundtruth correspondence $\pi^* : \mathcal{X} \to \mathcal{Y}$ from query shape $\mathcal{X}$ to some reference shape $\mathcal{Y}$ (discretized with $n$ vertices)
- Correspondence = label each query vertex $x$ as reference vertex $y$
- Net output at $x$ after softmax layer

$$\mathbf{f}_{\boldsymbol{\Theta}}(x) = (f_{\boldsymbol{\Theta},1}(x), \ldots, f_{\boldsymbol{\Theta},n}(x))$$

$=$ probability distribution on $\mathcal{Y}$




Minimize on training set the cross entropy between groundtruth correspondence and output probability distribution w.r.t. net parameters $\boldsymbol{\Theta}$

$$\min_{\boldsymbol{\Theta}} \sum_{x} H(\boldsymbol{\delta}_{\pi*(x)}, \mathbf{f}_{\boldsymbol{\Theta}}(x))$$

# Blended Intrinsic Map

Pointwise correspondence error (geodesic distance from groundtruth)

Kim, Lipman, Funkhouser 2011

# Geodesic CNN

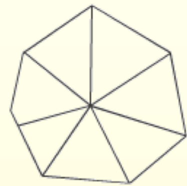
Pointwise correspondence error (geodesic distance from groundtruth)



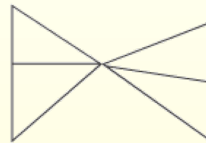
Masci et al. 2015

# Geodesic CNN

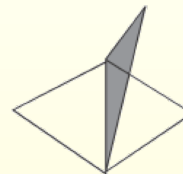
- Issues:
  - Requires lots of preprocessing.
  - Requires to have clean two-manifold meshes.



Manifold



Non-Manifold



Non-Manifold

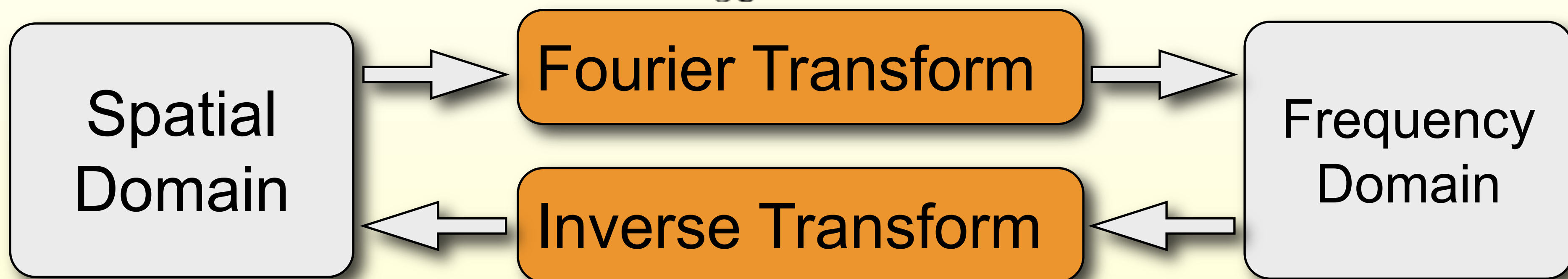
# Spectral CNN

# Fourier analysis

- Inner product for $L_2$ function space $\langle f, g \rangle := \int_{-\infty}^{\infty} f(x)\,\overline{g(x)}\,\mathrm{d}x$
- Orthonormal basis : complex “waves”

$$e_u(x) \; := \; \mathrm{e}^{\mathrm{i}2\pi ux} \; = \; \cos(2\pi ux) - \mathrm{i}\sin(2\pi ux)$$

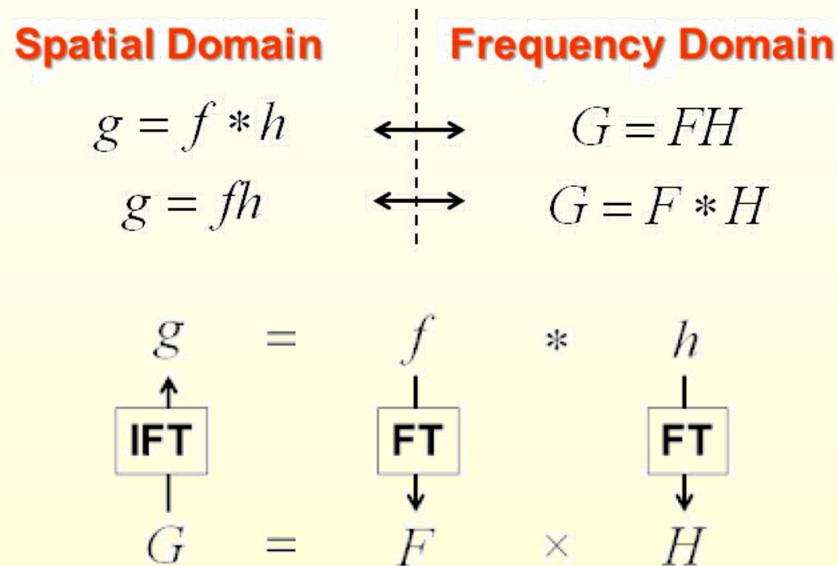
$$F(\omega) = \int_{-\infty}^{\infty} f(x)\mathrm{e}^{-2\pi i\omega x}\,dx$$




$$f(x) = \int_{-\infty}^{\infty} F(\omega)\mathrm{e}^{2\pi i\omega x}\,d\omega$$

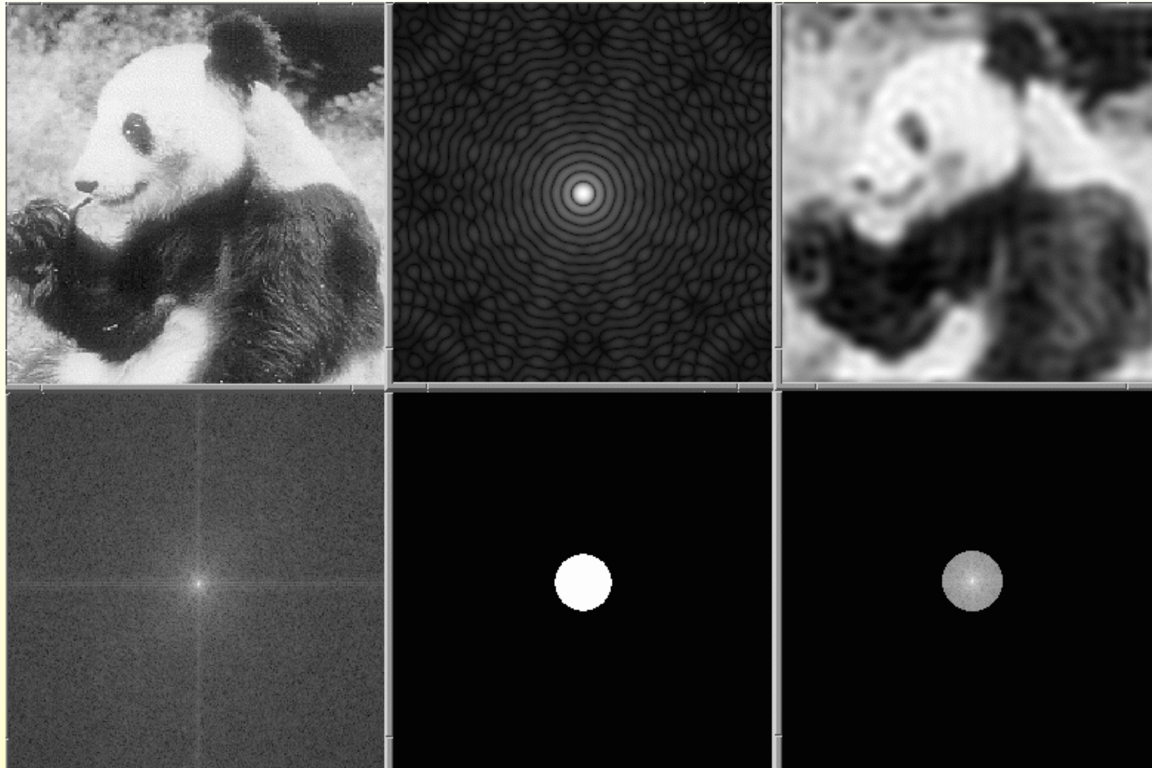
# Convolution Theorem

Convolution in one domain (e.g., space domain) equals point-wise multiplication in the other domain (e.g., frequency domain).



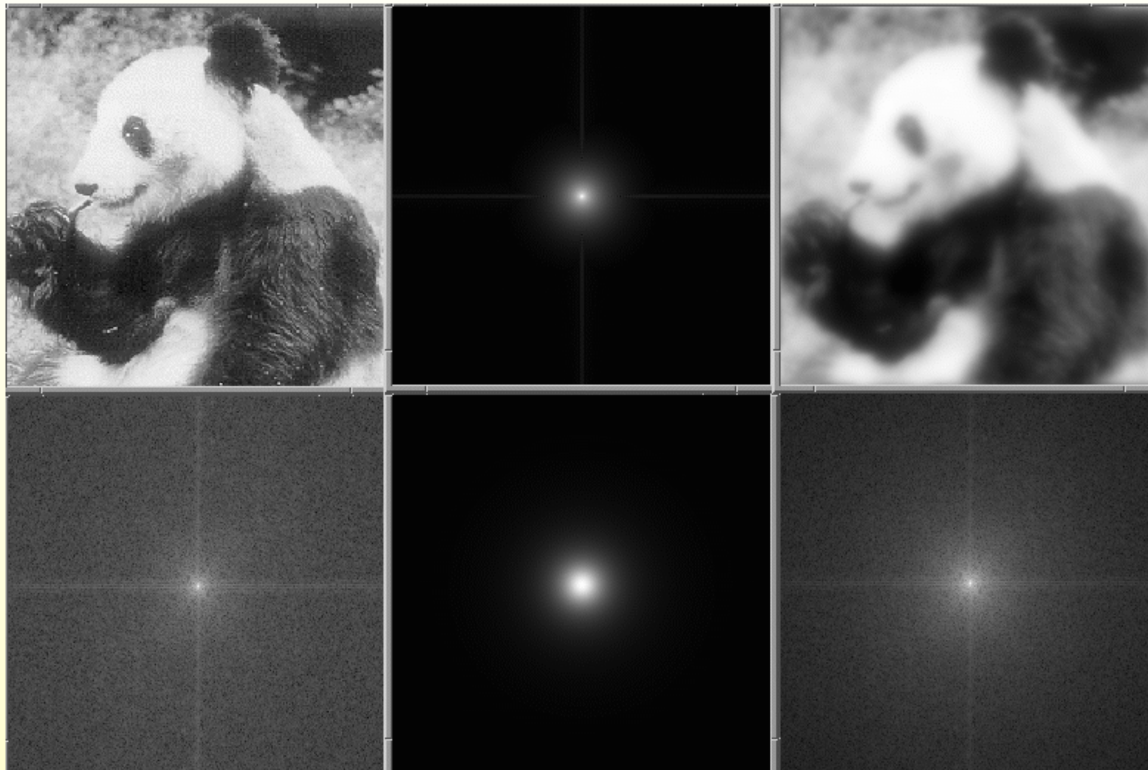
# Convolution Theorem

- Low-frequency filter

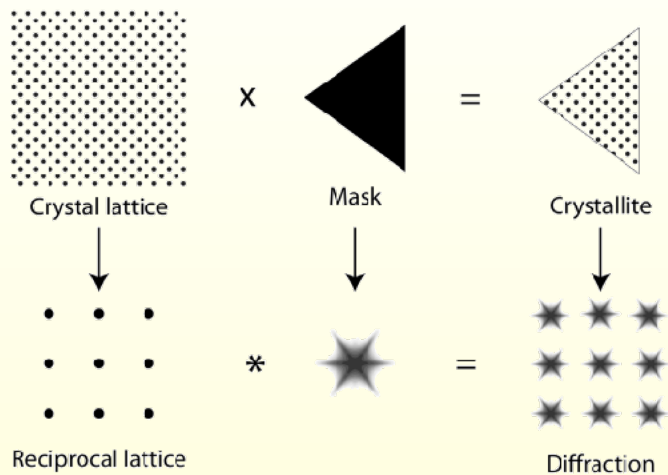


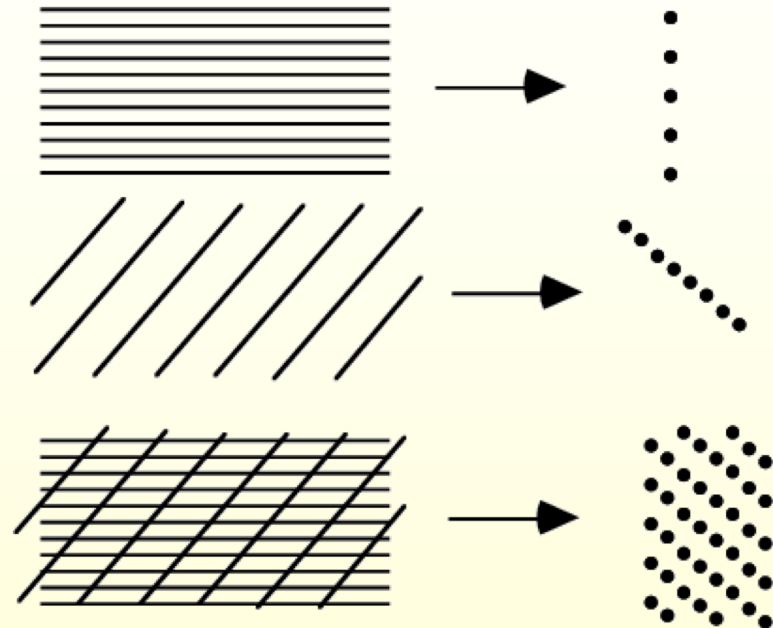
# Convolution Theorem

- Low-frequency filter



# Convolution Theorem



# Convolution Theorem

Convolution of two vectors $\mathbf{f} = (f_1, \ldots, f_n)^\top$ and $\mathbf{g} = (g_1, \ldots, g_n)^\top$

$$\mathbf{f} \star \mathbf{g} = \underbrace{\begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix}}_{\text{circulant matrix}} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

$n \times n$ square matrix

1-D Euclidean Space Proof

# Convolution Theorem

Convolution of two vectors $\mathbf{f} = (f_1, \ldots, f_n)^\top$ and $\mathbf{g} = (g_1, \ldots, g_n)^\top$

$$\mathbf{f} \star \mathbf{g} = \begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

$$= \mathbf{\Phi} \begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix} \mathbf{\Phi}^\top \mathbf{f}$$

- *All circulant matrix have the same eigenvectors.*
- *Laplace operator is a circulant matrix.*

1-D Euclidean Space Proof

# Convolution Theorem

Convolution of two vectors $\mathbf{f} = (f_1, \ldots, f_n)^\top$ and $\mathbf{g} = (g_1, \ldots, g_n)^\top$

$$\mathbf{f} \star \mathbf{g} = \begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

$$= \boldsymbol{\Phi} \begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix} \boldsymbol{\Phi}^\top \mathbf{f}$$

1-D Euclidean Space Proof

# Convolution Theorem

Convolution of two vectors $\mathbf{f} = (f_1, \ldots, f_n)^\top$ and $\mathbf{g} = (g_1, \ldots, g_n)^\top$

$$\mathbf{f} \star \mathbf{g} = \begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

$$= \mathbf{\Phi} \begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix} \begin{bmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{bmatrix}$$

1-D Euclidean Space Proof

# Convolution Theorem

Convolution of two vectors $\mathbf{f} = (f_1, \ldots, f_n)^\top$ and $\mathbf{g} = (g_1, \ldots, g_n)^\top$

$$\mathbf{f} \star \mathbf{g} = \begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

$$= \mathbf{\Phi} \begin{bmatrix} \hat{f}_1 \cdot \hat{g}_1 \\ \vdots \\ \hat{f}_n \cdot \hat{g}_n \end{bmatrix}$$

1-D Euclidean Space Proof

# Convolution Theorem

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

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

# Spectral Transform on Mesh Surface

- The second derivative of sine/cosine function is the same with itself (except for the coefficient).

$$\frac{d^2}{dx^2} = \sin(\omega x) = -\omega^2 \sin(\omega x)$$
$$\frac{d^2}{dx^2} = \cos(\omega x) = -\omega^2 \cos(\omega x)$$

- Laplacian is a second derivative operator.

$$-\Delta V^k = \lambda^k V^k$$

# Convolution Theorem

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

$$f \star g = \sum_{k \geq 1} \langle f, \phi_k \rangle_{L^2(\mathcal{X})} \langle g, \phi_k \rangle_{L^2(\mathcal{X})} \, \phi_k$$

In matrix-vector notation

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

$$\mathbf{f} \star \mathbf{g} = \underbrace{\mathbf{\Phi} \, \mathrm{diag}(\hat{g}_1, \ldots, \hat{g}_n) \mathbf{\Phi}^\top}_{\mathbf{G}} \, \mathbf{f}$$

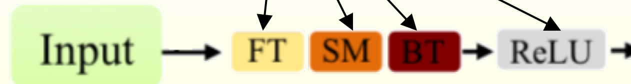
Spectral transform *diagonalizes* the convolution operator.

# Spectral CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{g}_l = \xi\left(\sum_{l'=1}^{p} \mathbf{\Phi}\hat{\mathbf{W}}_{l,l'}\mathbf{\Phi}^\top \mathbf{f}_{l'}\right) \qquad \begin{array}{l} l = 1,\ldots,q \\ l' = 1,\ldots,p \end{array}$$

where $\hat{\mathbf{W}}_{l,l} = n \times n$ diagonal matrix of filter coefficients




Bruna et al. 2014

# Spectral CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{g}_l = \xi\left(\sum_{l'=1}^{p} \mathbf{\Phi}\hat{\mathbf{W}}_{l,l'}\mathbf{\Phi}^\top \mathbf{f}_{l'}\right) \quad \begin{array}{l} l = 1,\ldots,q \\ l' = 1,\ldots,p \end{array}$$

where $\hat{\mathbf{W}}_{l,l} = n \times n$ diagonal matrix of filter coefficients

☹ $\mathcal{O}(n)$ parameters per layer

Bruna et al. 2014

# Spectral CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{g}_l = \xi\left(\sum_{l'=1}^{p} \mathbf{\Phi}\hat{\mathbf{W}}_{l,l'}\mathbf{\Phi}^\top \mathbf{f}_{l'}\right) \quad \begin{array}{l} l = 1,\ldots,q \\ l' = 1,\ldots,p \end{array}$$

where $\hat{\mathbf{W}}_{l,l} = n \times n$ diagonal matrix of filter coefficients

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

Bruna et al. 2014

# Spectral CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{g}_l = \xi \left( \sum_{l'=1}^{p} \mathbf{\Phi} \hat{\mathbf{W}}_{l,l'} \mathbf{\Phi}^\top \mathbf{f}_{l'} \right) \quad \begin{array}{l} l = 1, \ldots, q \\ l' = 1, \ldots, p \end{array}$$

where $\hat{\mathbf{W}}_{l,l} = n \times n$ diagonal matrix of filter coefficients

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



Bruna et al. 2014

# Spectral CNN

Convolutional layer expressed in the spectral domain

$$\mathbf{g}_l = \xi \left( \sum_{l'=1}^{p} \mathbf{\Phi} \hat{\mathbf{W}}_{l,l'} \mathbf{\Phi}^\top \mathbf{f}_{l'} \right) \quad \begin{array}{l} l = 1, \ldots, q \\ l' = 1, \ldots, p \end{array}$$

where $\hat{\mathbf{W}}_{l,l} = n \times n$ diagonal matrix of filter coefficients

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

Bruna et al. 2014

# Spectral CNN

Function $f$

Filtered function $\tilde{f}$

Same function,
same filter,
another shape

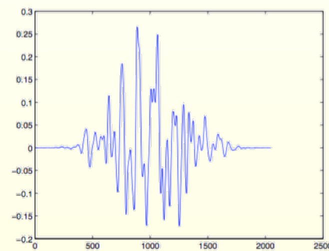
image from Jonathan Masci et al

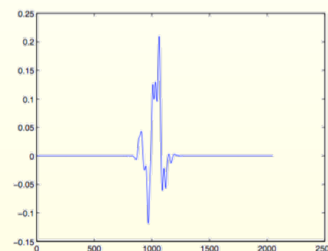
# Spectral CNN

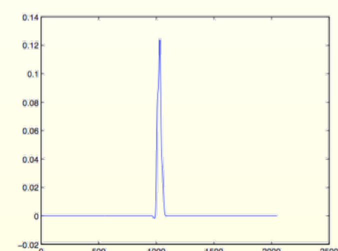
- Observation:
  In Fourier analysis, smoothness and sparsity are dual notions.

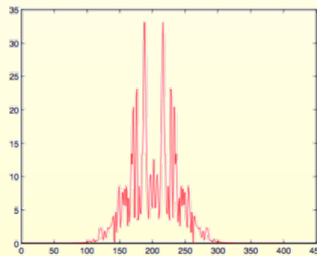
$x$ fast decay

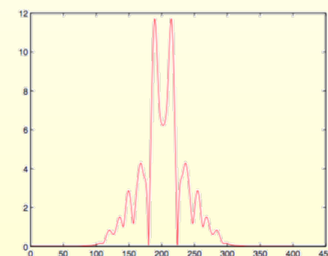
$\widehat{x}$ smooth

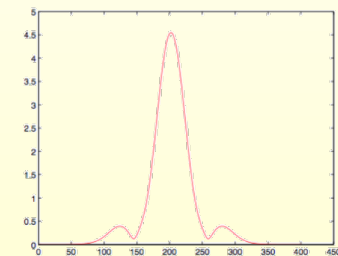












Joan Bruna et al. 2013

# Spectral CNN

In the Euclidean setting (by Parseval's identity)

$$\int_{-\infty}^{+\infty} |x|^{2k} |f(x)|^2 dx = \int_{-\infty}^{+\infty} \left| \frac{\partial^k \hat{f}(\omega)}{\partial \omega^k} \right|^2 d\omega$$

$\Rightarrow$ Localization in space = smoothness in frequency domain

Parametrize the filter using a smooth spectral transfer function $\tau(\lambda)$

Bruna et al. 2014; Henaff, Bruna, LeCun 2015

# Spectral CNN

In the Euclidean setting (by Parseval's identity)

$$\int_{-\infty}^{+\infty} |x|^{2k} |f(x)|^2 dx = \int_{-\infty}^{+\infty} \left| \frac{\partial^k \hat{f}(\omega)}{\partial \omega^k} \right|^2 d\omega$$

$\Rightarrow$ Localization in space = smoothness in frequency domain

Parametrize the filter using a smooth spectral transfer function $\tau(\lambda)$

E.g. Cubic-spline parameterization

$$\mathrm{Diag}(W) = \begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 2 & 1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{r-1} \\ \alpha_r \end{bmatrix}$$

$(r \ll n)$ $\quad B: (n \times r)$ $\quad \alpha: (r \times 1)$

Bruna et al. 2014; Henaff, Bruna, LeCun 2015

# Spectral CNN

$$\tau_\alpha = \Phi W \Phi^T$$

$$= \Phi \mathrm{Diag}(B\alpha) \Phi^T$$

$$= \sum_{i=0}^{r} \alpha_i (\Phi \mathrm{Diag}(b_i) \Phi^T)$$

$$= \sum_{i=0}^{r} \alpha_i \Delta_i$$

# Spectral CNN

Represent spectral transfer function as a polynomial or order $r$

$$\tau_{\boldsymbol{\alpha}}(\boldsymbol{\Delta}) = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j$$

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

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

Defferrard, Bresson, Vandergheynst 2016

# Spectral CNN

Represent spectral transfer function as a polynomial or order $r$

$$\tau_{\boldsymbol{\alpha}}(\boldsymbol{\Delta}) = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j$$

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

- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ Filters have guaranteed $r$-hops support

Defferrard, Bresson, Vandergheynst 2016

# Spectral CNN

Represent spectral transfer function as a polynomial or order $r$

$$\tau_{\boldsymbol{\alpha}}(\boldsymbol{\Delta}) = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j$$

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

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

Defferrard, Bresson, Vandergheynst 2016

# Spectral CNN

Represent spectral transfer function as a polynomial or order $r$

$$\tau_{\boldsymbol{\alpha}}(\boldsymbol{\Delta}) = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j$$

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

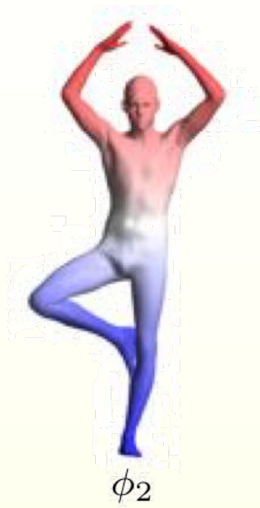
- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ Filters have guaranteed $r$-hops support
- ☺ No explicit computation of $\boldsymbol{\Phi}^\top, \boldsymbol{\Phi} \Rightarrow \mathcal{O}(nr)$ complexity
- ☹ Does not generalize across domains

Defferrard, Bresson, Vandergheynst 2016

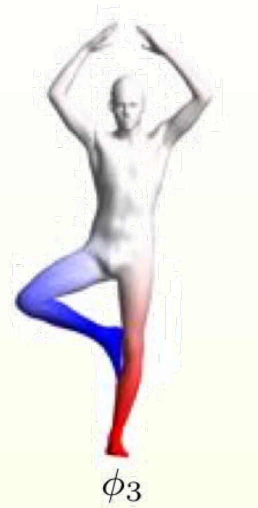
# Filtering in Different Bases

**Eigenmode switching**:
Perturbation theory predicts that when eigenvalues move close to each other, the corresponding eigenvectors may switch order.

# Filtering in Different Bases



$\phi_2$



$\phi_3$

$\phi_{10}$

$\phi_{15}$

$\phi_{20}$

$\psi_2$

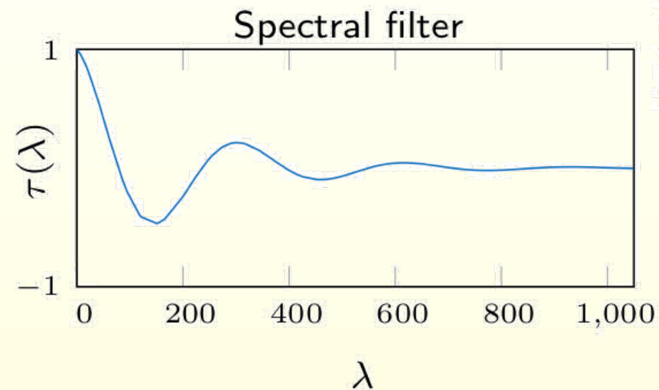
$\psi_3$

$\psi_{10}$

$\psi_{15}$

$\psi_{20}$

# Filtering in Different Bases



$\mathbf{\Phi}\tau(\mathbf{\Lambda_\Phi})\mathbf{\Phi}^\top\boldsymbol{\delta}_0$

$\mathbf{\Psi}\tau(\mathbf{\Lambda_\Psi})\mathbf{\Psi}^\top\boldsymbol{\delta}_0$

Apply spectral filter $\tau(\lambda)$ in different bases $\mathbf{\Phi}$ and $\mathbf{\Psi}$

$\Rightarrow$ **different results!**

# Bases Synchronization

$\Phi \tau(\Lambda_{\Phi}) \Phi^{\top} \delta_0$

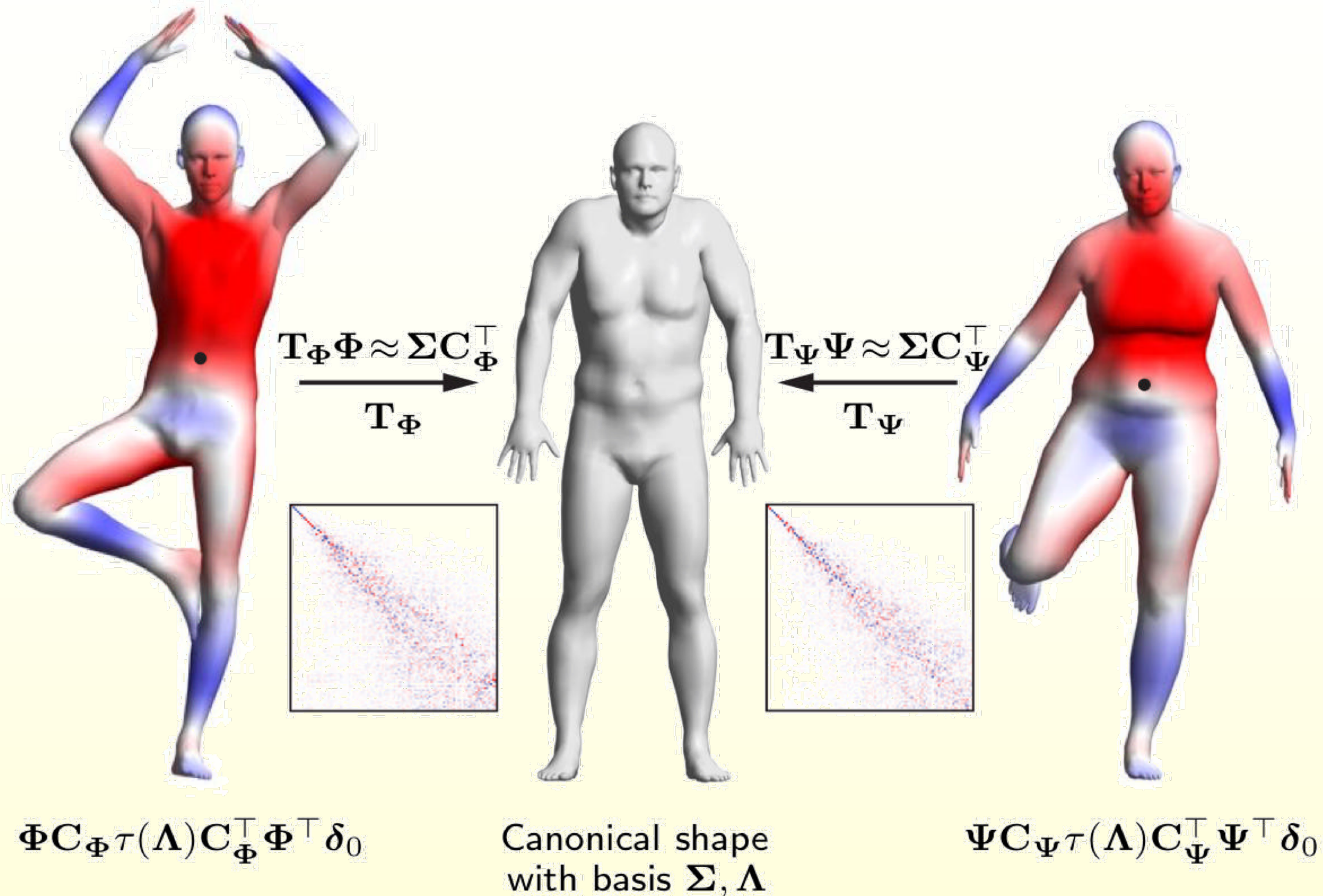
Canonical shape
with basis $\Sigma, \Lambda$

$\Psi \tau(\Lambda_{\Psi}) \Psi^{\top} \delta_0$

Apply spectral filter $\tau(\lambda)$ in different bases $\Phi$ and $\Psi$

$\Rightarrow$ **different results!**

# Bases Synchronization




Apply spectral filter $\tau(\lambda)$ in synchronized bases $\Phi \mathbf{C}_\Phi$ and $\Psi \mathbf{C}_\Psi$

$\Rightarrow$ similar results!

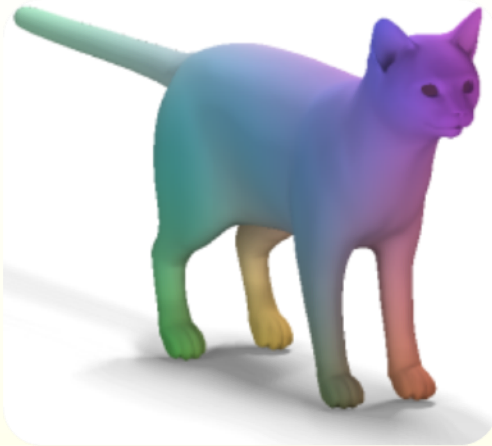
Yi et al. 2017

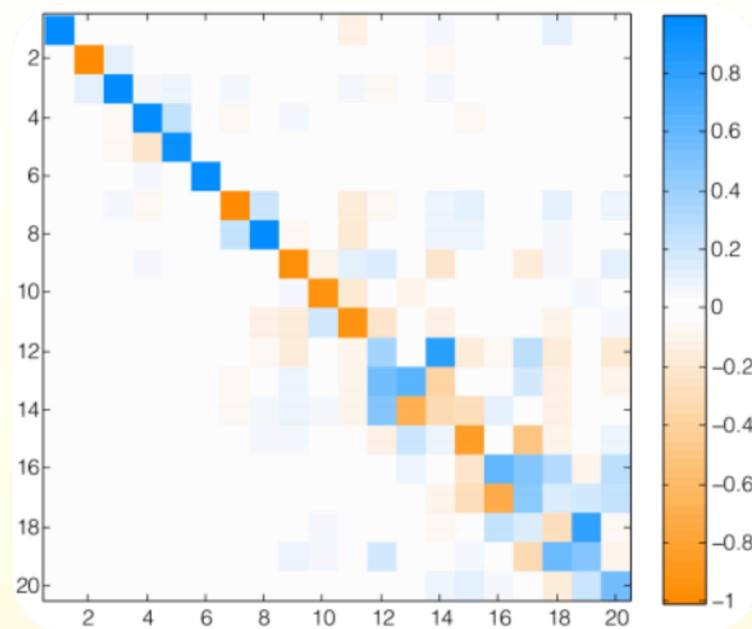
# Bases Synchronization

| 1 | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | | | | | | |
| | | 1 | | | | | |
| | | | 1 | | | | |
| | | | | | 1 | | |
| | | | | 1 | | | |
| | | | | | | 1 | |
| | | | | | | | ⋱ |

Permutation Matrix

# Bases Synchronization





Generalize to *Fuzzy* Permutation Matrix
– called **Functional Map**

# Functional Map (Intro)

Shape $A$

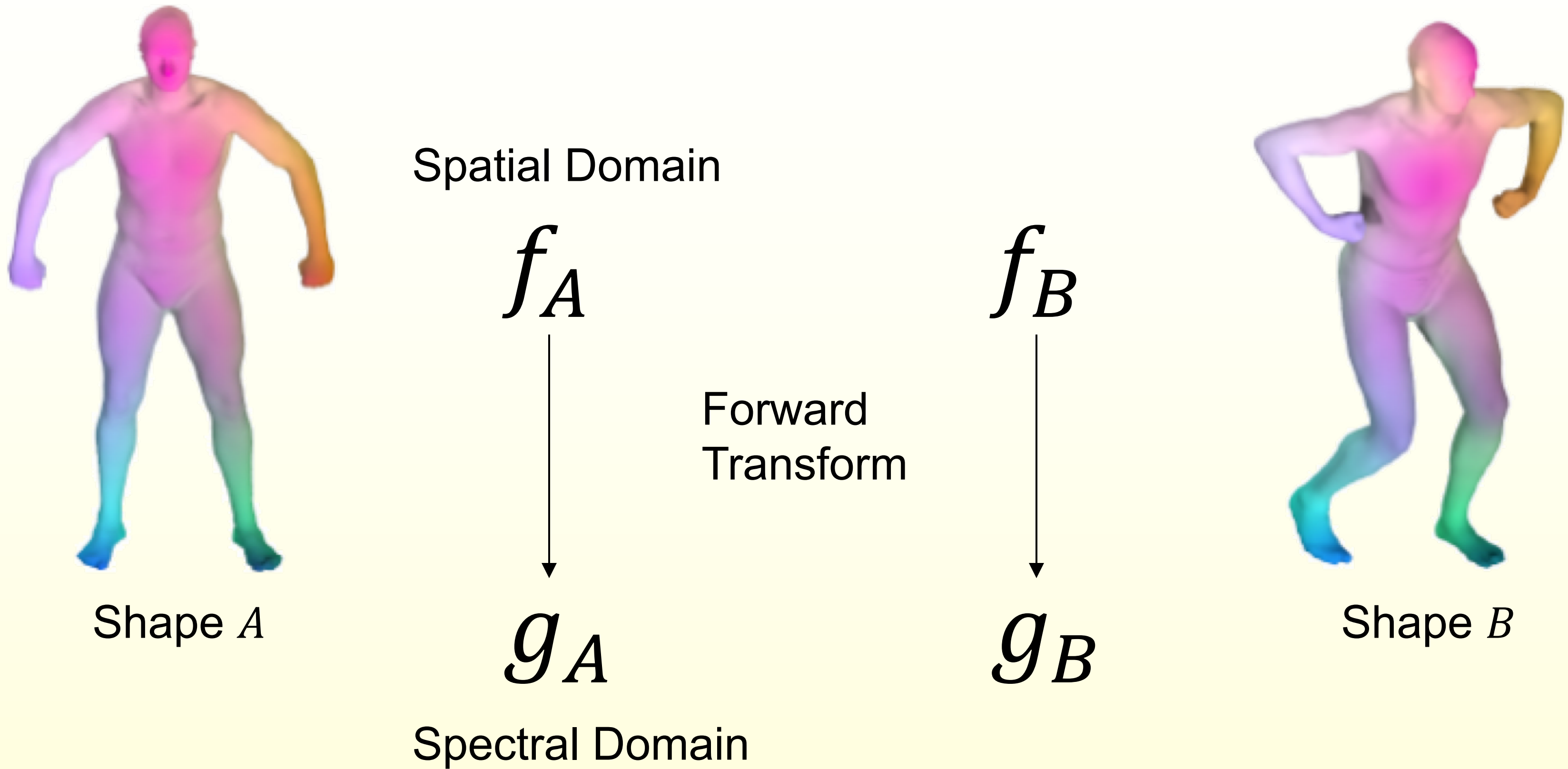
Spatial Domain

$f_A$

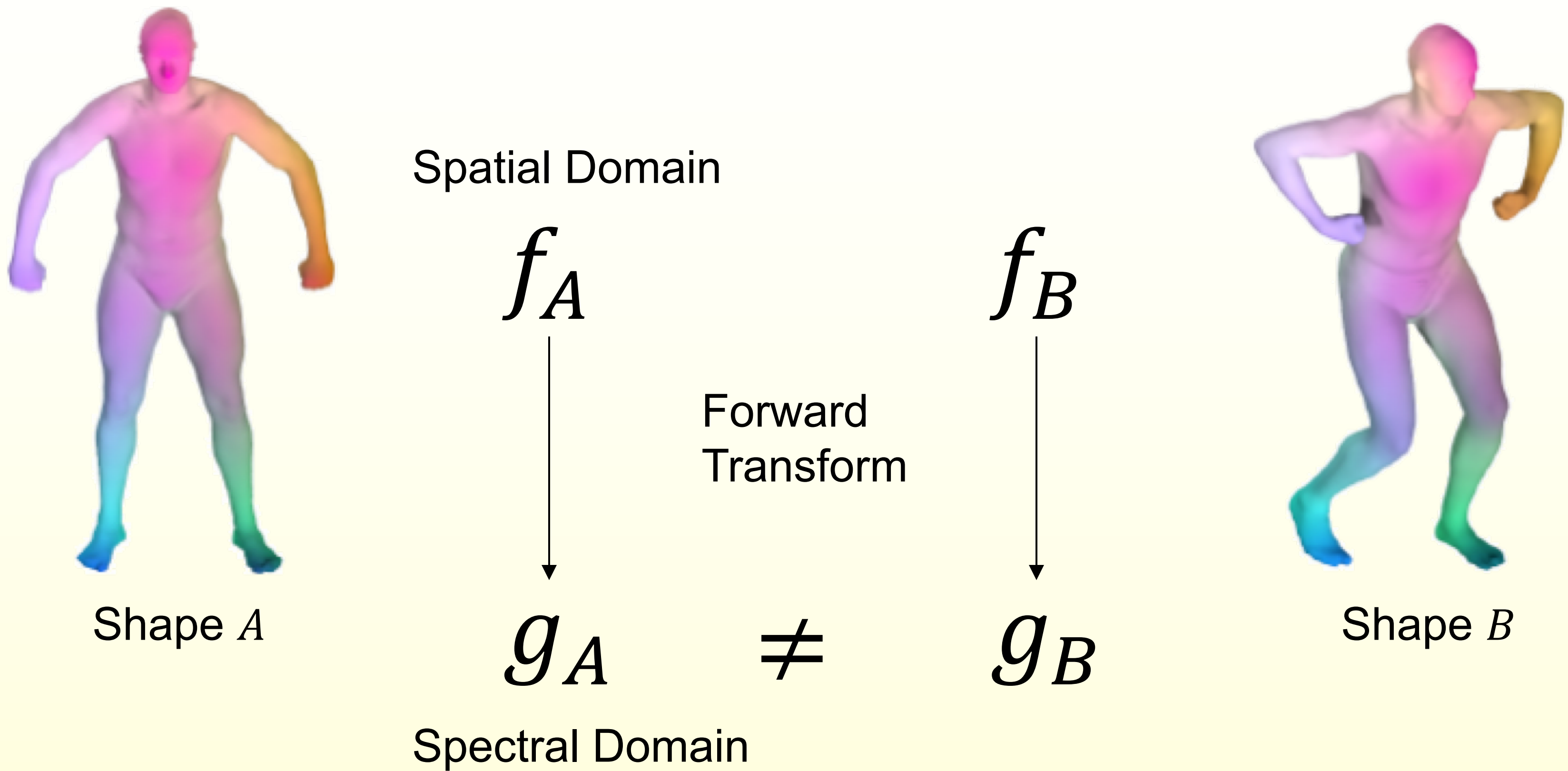
$f_B$

Shape $B$

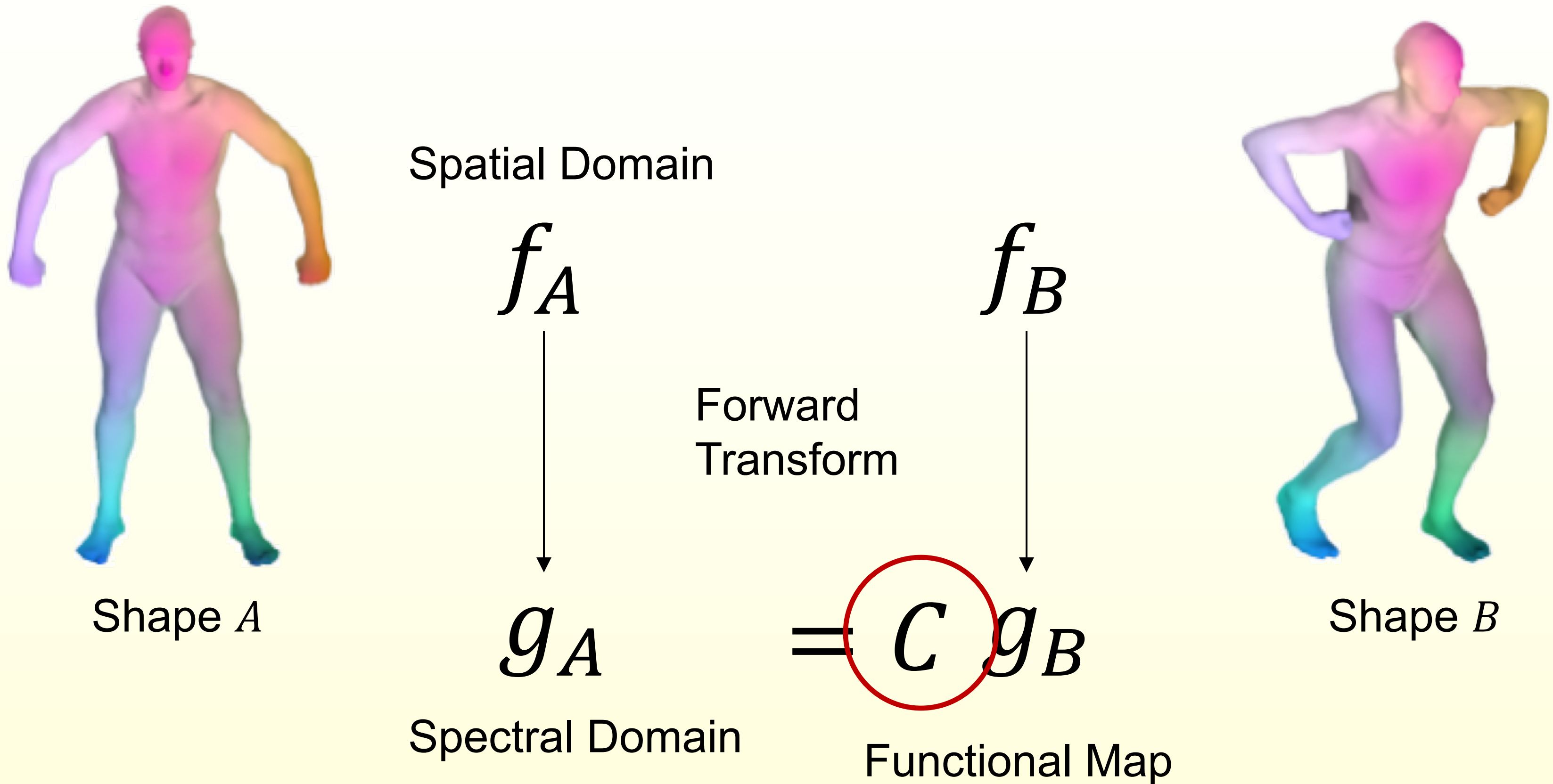
# Functional Map (Intro)



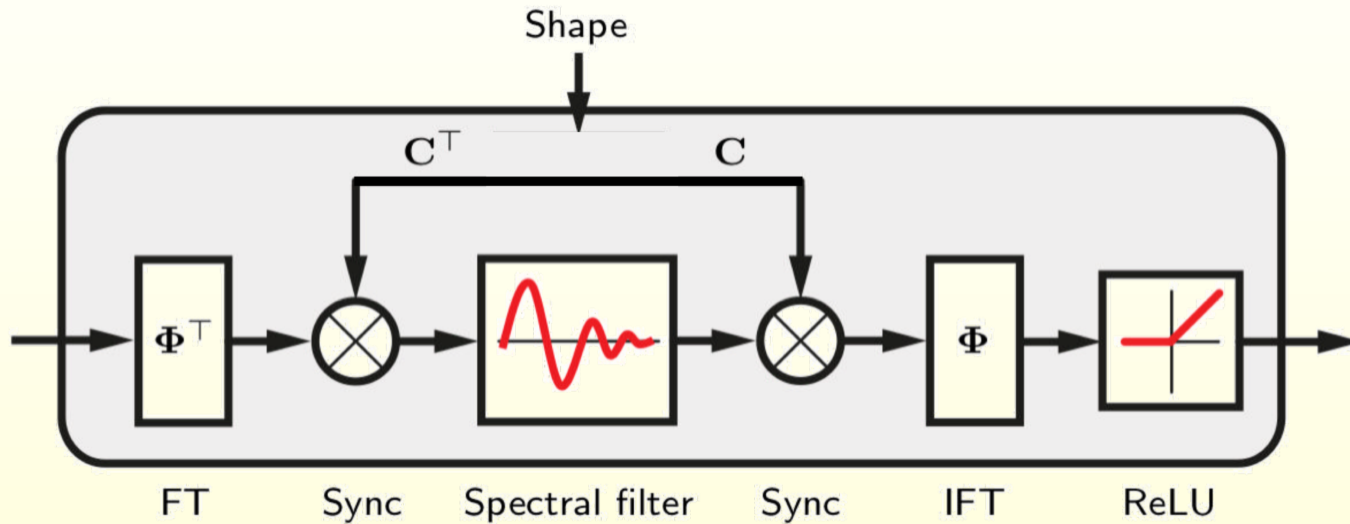
# Functional Map (Intro)



# Functional Map (Intro)

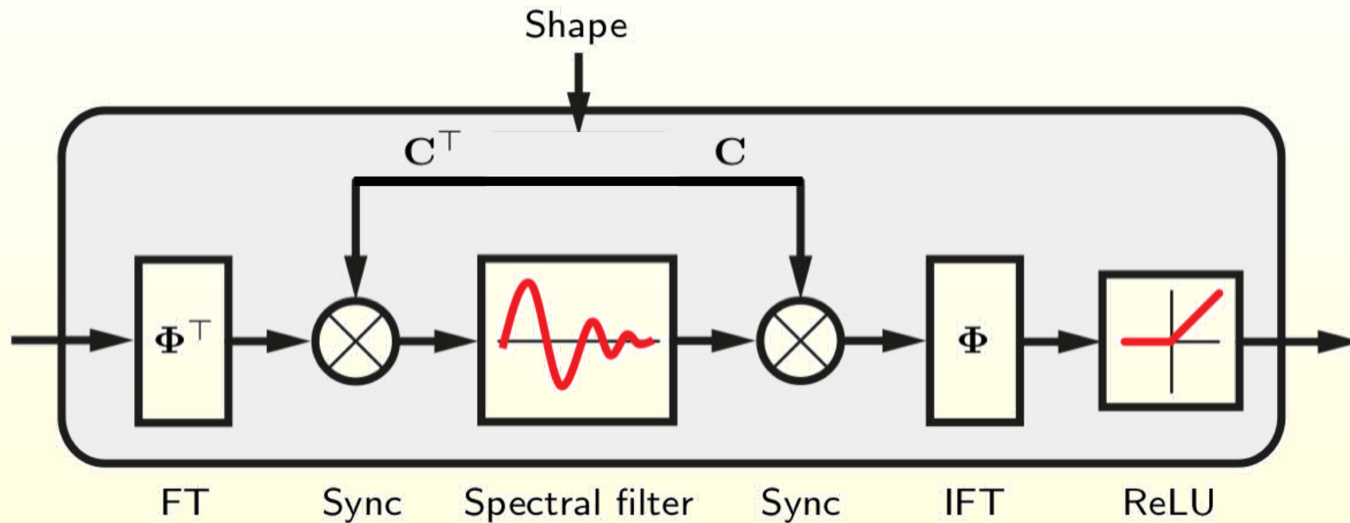


# SyncSpecCNN



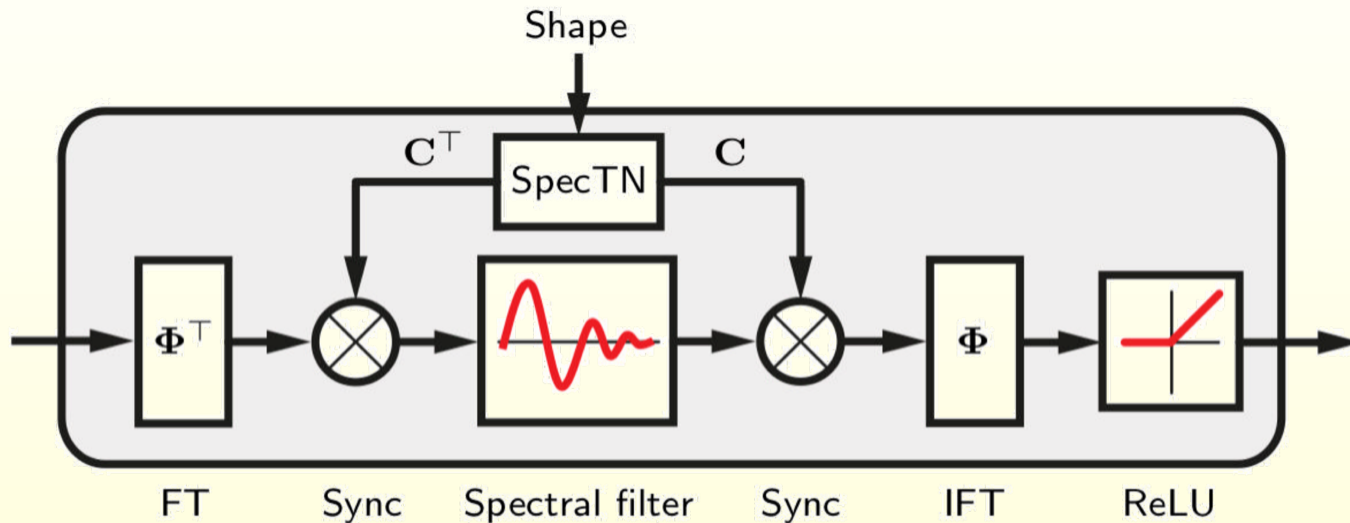
# SyncSpecCNN

- What if no mapping is provided?



# SyncSpecCNN

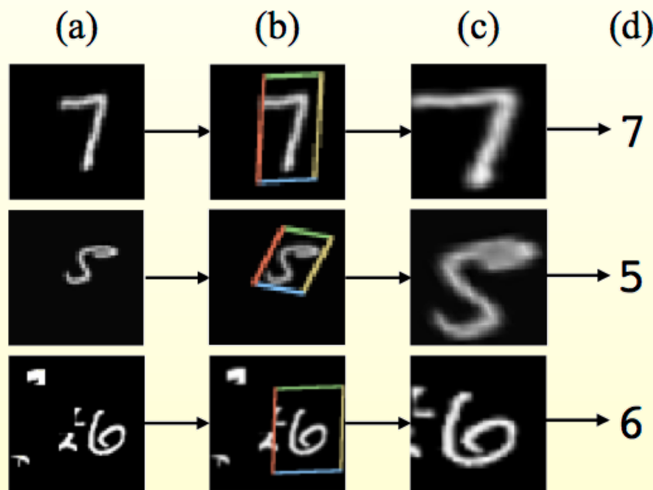
- What if no mapping is provided?
  : Let the neural network synchronize them!

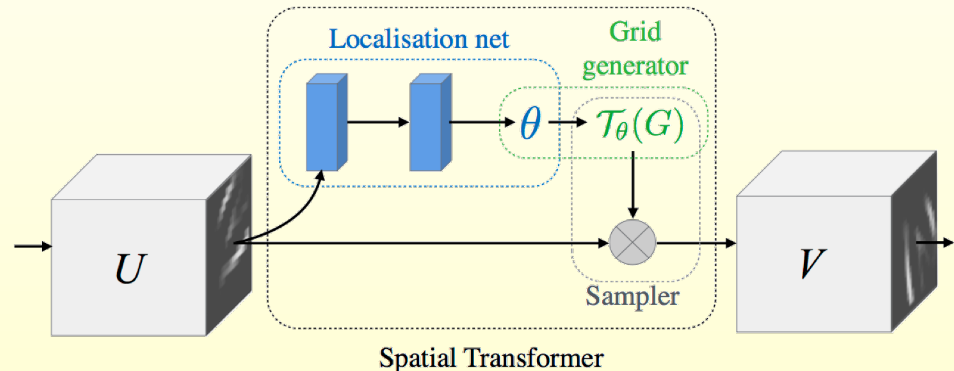



Convolutional filter of a Spectral Transformer Network

# SyncSpecCNN

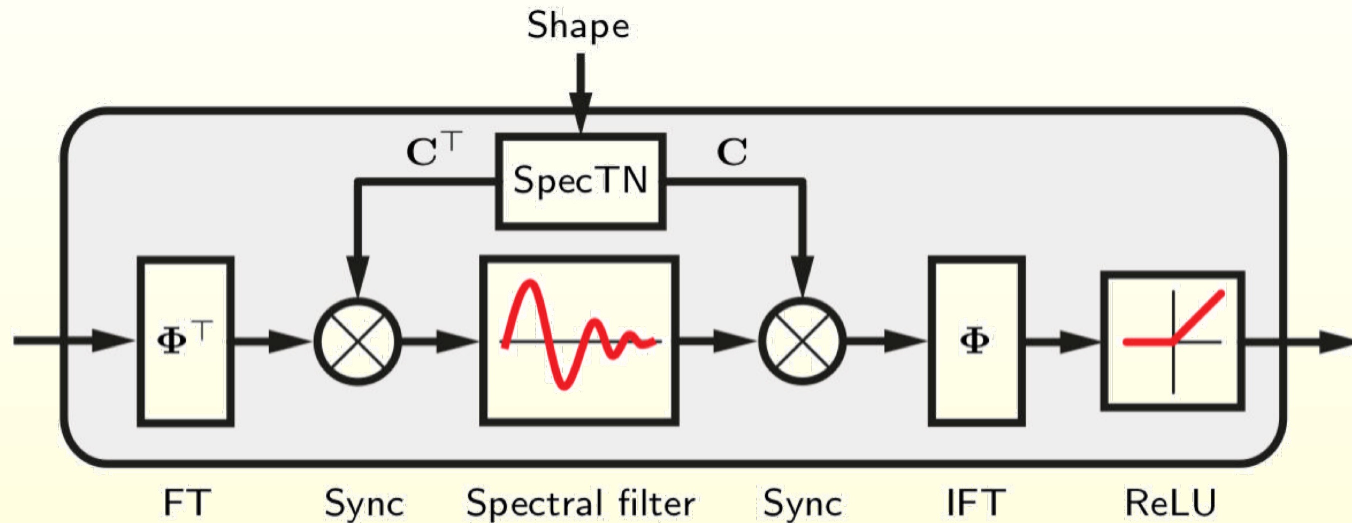
- Spatial Transformer Network [Jaderberg et al., 2015]
  - Reduce the variation of input data.
  - Can be plugged into any architecture.



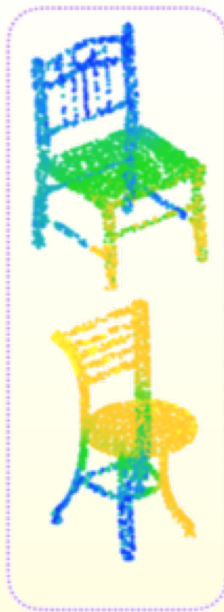
# SyncSpecCNN

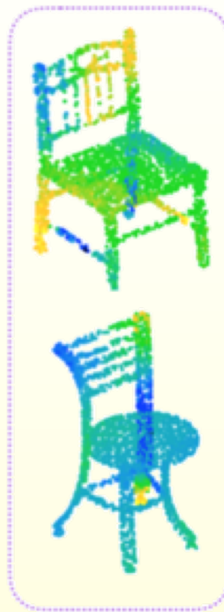
- *Spectral* Transformer Network




Convolutional filter of a Spectral Transformer Network

# Synchronization visualization



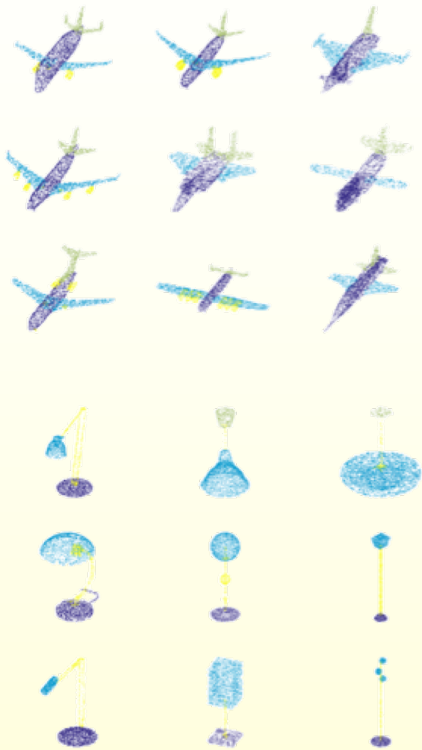


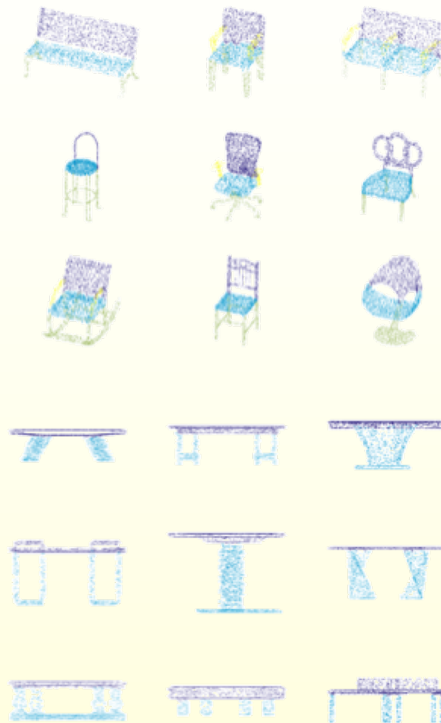
before synchronization

after synchronization

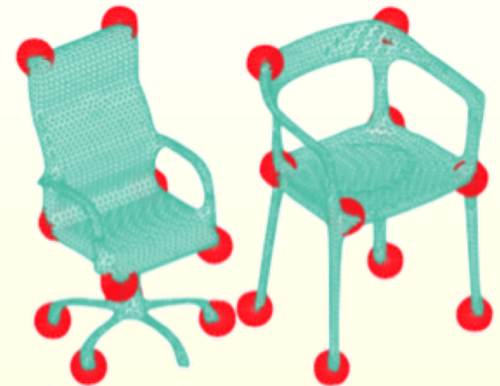
Li Yi et al. 2017

# SyncSpecCNN





part segmentation



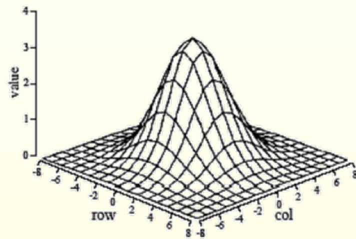
Li Yi et al. 2017

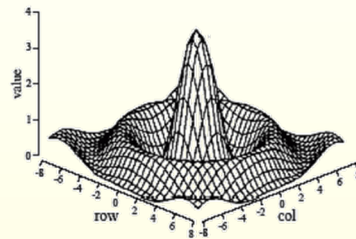
key point prediction

# Anisotropic CNN

# Homogeneous Diffusion

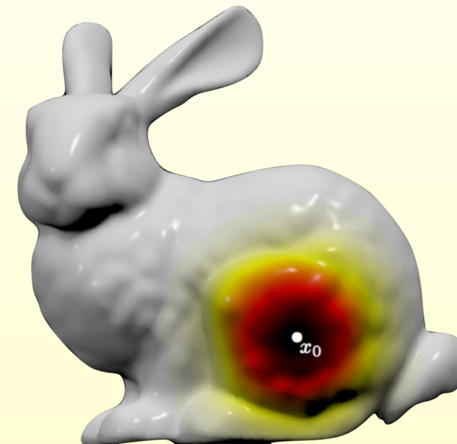
- Isotropic filters are less descriptive, in analogy to circular filters in image CNN




circular filters

edge filters

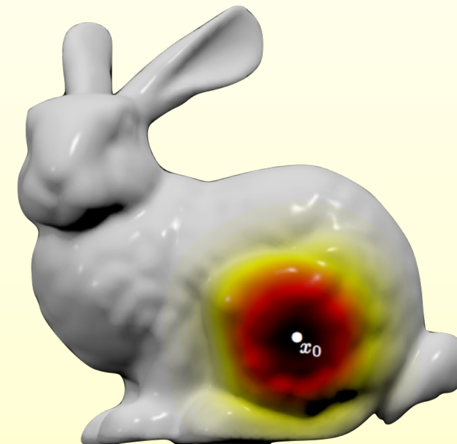


# Homogeneous Diffusion

- Heat kernel - isotropic diffusion

$$f_t(x) = -c\Delta f(x)$$

$c =$ thermal diffusivity constant describing heat conduction properties of the material (diffusion speed is equal everywhere)



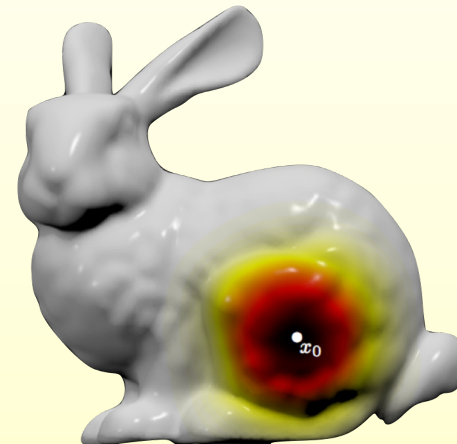

Davide Boscaini et al. 2016

# Homogeneous Diffusion

- Heat kernel - isotropic diffusion

$$f_t(x) = -\mathrm{div}_X(c\nabla_X f(x))$$

$c =$ **thermal diffusivity constant** describing heat conduction properties of the material (diffusion speed is equal everywhere)



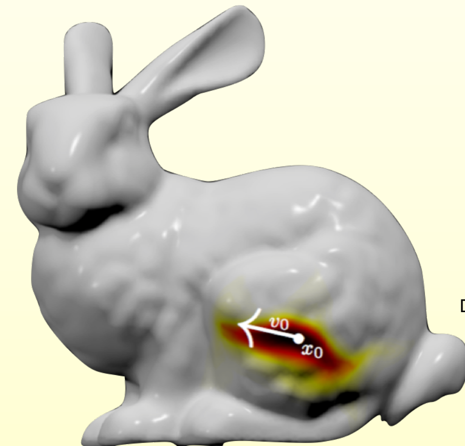

Davide Boscaini et al. 2016

# Anisotropic Diffusion

- Heat kernel - anisotropic diffusion

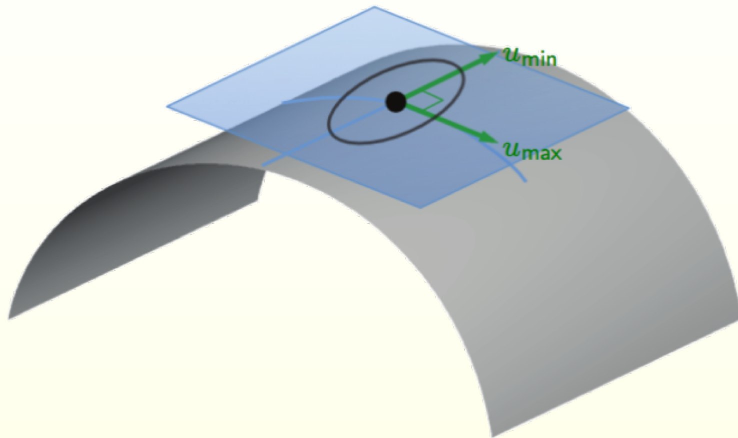
$$f_t(x) = -\mathrm{div}_X(A(x)\nabla_X f(x))$$

$A(x)$ = **heat conductivity tensor** describing heat conduction properties of the material (diffusion speed is position + direction dependent)




Davide Boscaini et al. 2016

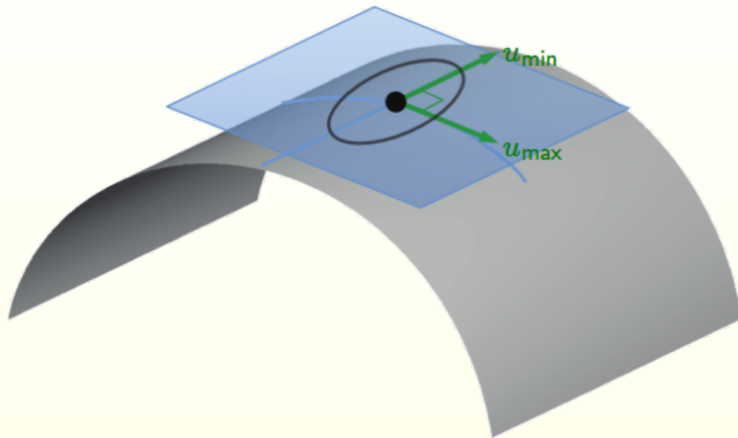
# Anisotropic Diffusion

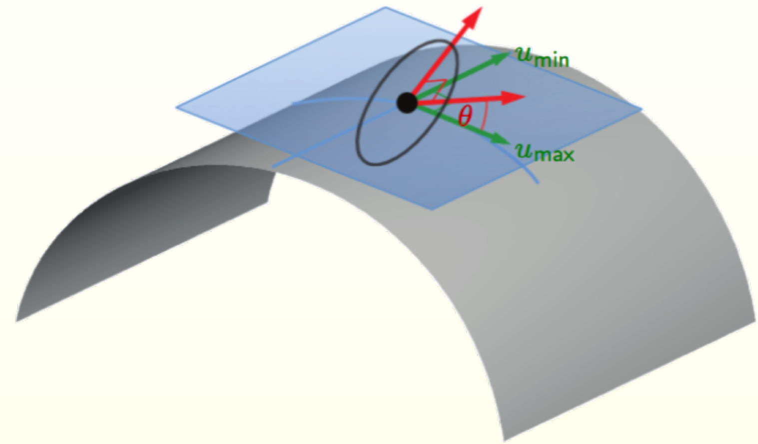



$$f_t(x) = -\mathrm{div}_X \left( \begin{bmatrix} \alpha & \\ & 1 \end{bmatrix} \nabla_X f(x) \right)$$

Davide Boscaini et al. 2016

# Anisotropic Diffusion



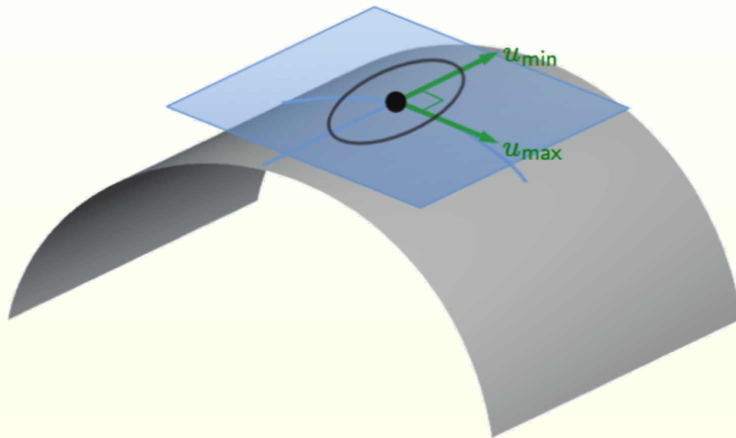



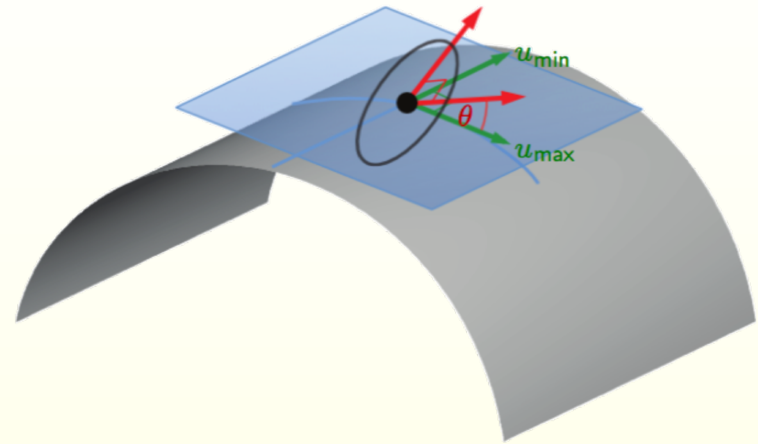

$$f_t(x) = -\mathrm{div}_X \left( \begin{bmatrix} \alpha & \\ & 1 \end{bmatrix} \nabla_X f(x) \right)$$

$$f_t(x) = -\mathrm{div}_X \left( \underbrace{R_\theta \begin{bmatrix} \alpha & \\ & 1 \end{bmatrix} R_\theta^\top}_{A_{\alpha\theta}(x)} \nabla_X f(x) \right)$$

Davide Boscaini et al. 2016

# Anisotropic Diffusion




$$f_t(x) = -\text{div}_X\left(\begin{bmatrix}\alpha & \\ & 1\end{bmatrix}\nabla_X f(x)\right)$$

$$f_t(x) = -\text{div}_X\left(\underbrace{R_\theta\begin{bmatrix}\alpha & \\ & 1\end{bmatrix}R_\theta^\top}_{A_{\alpha\theta}(x)}\nabla_X f(x)\right)$$

- **Anisotropic Laplacian** $\Delta_{\alpha\theta} f(x) = \text{div}_X\left(A_{\alpha\theta}(x)\nabla_X f(x)\right)$
- $\theta =$ orientation w.r.t. max curvature direction
- $\alpha =$ 'elongation'

Davide Boscaini et al. 2016

# Blended Intrinsic Map

Pointwise correspondence error (geodesic distance from groundtruth)

Kim, Lipman, Funkhouser 2011

# Geodesic CNN

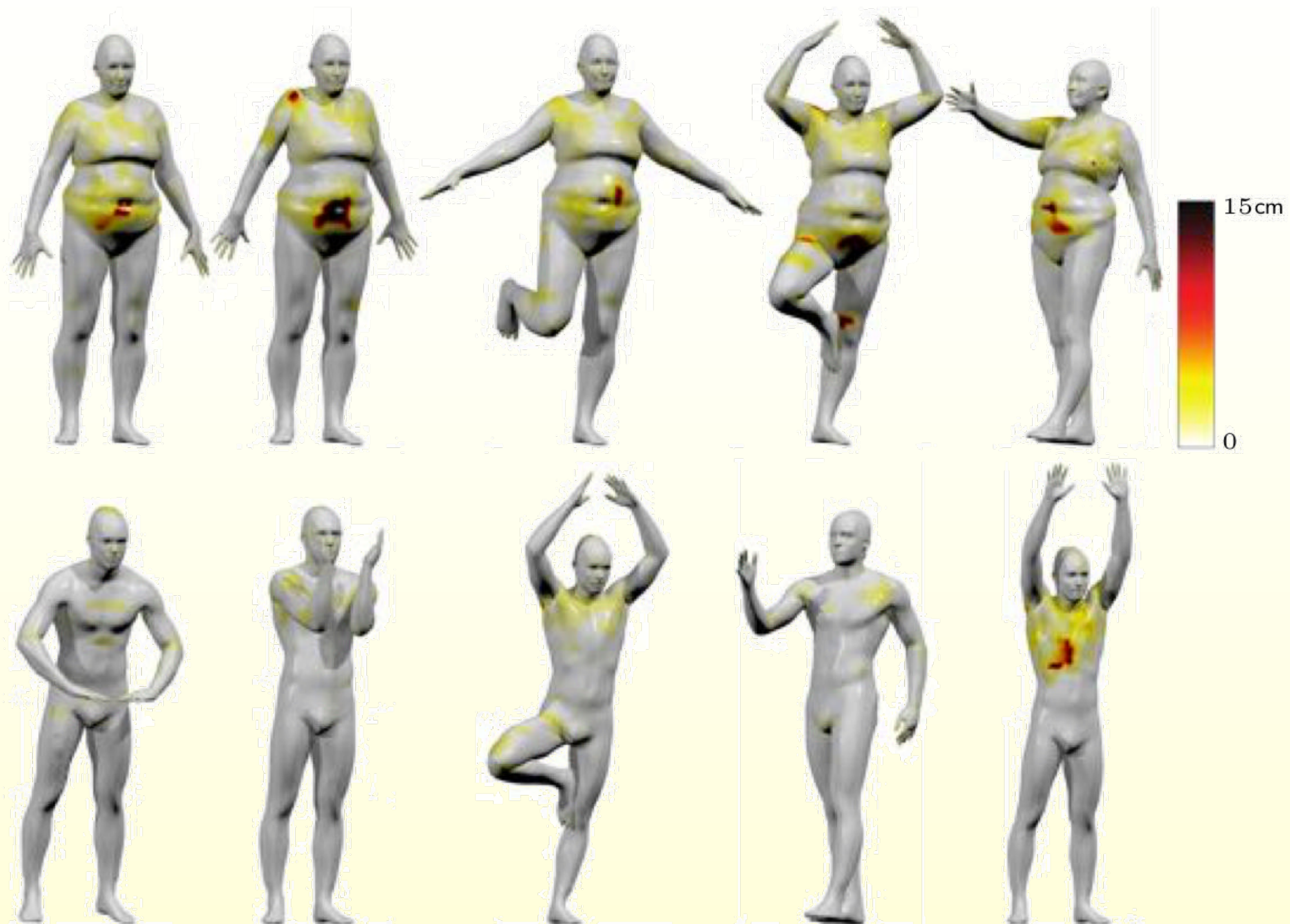
Pointwise correspondence error (geodesic distance from groundtruth)

Masci et al. 2015

# Anisotropic Heat Kernels

Davide Boscaini et al. 2016

# Anisotropic CNN




Pointwise correspondence error (geodesic distance from groundtruth)



Boscaini et al. 2016

# Discussion

- Spatial construction is usually more efficient but less principled.
- Spectral construction is more principled but usually slow (computing Laplacian eigenvectors for large scale data could be painful).
- On going research tries to bridge the gap.

# References


- Kipf et al., Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017.
- Masci et al., Geodesic Convolutional Neural Networks on Riemannian Manifolds, ICCV Workshop 2015.
- Bruna et al., Spectral networks and locally connected networks on graphs, ICLR, 2013.
- Bronstein et al., Geometric deep learning: going beyond Euclidean data, IEEE Signal Processing Magazine, 2017.
- Yi et al., SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation, CVPR 2017.

# The End