EECS 230 Deep Learning

Lecture 15: Graph Neural Network

Some slides from Simon Prince, Paul-Edouard Sarlin, and Jure Leskovec

Outline

- ❑ Graph Neural Network
  - ❑ Graph convolution layer
  - ❑ Graph convolutional network
  - ❑ Graph attention network
- ❑ An application to correspondence matching
  - ❑ SuperGlue for visual localization

Graph Neural Network

Real-world graphs

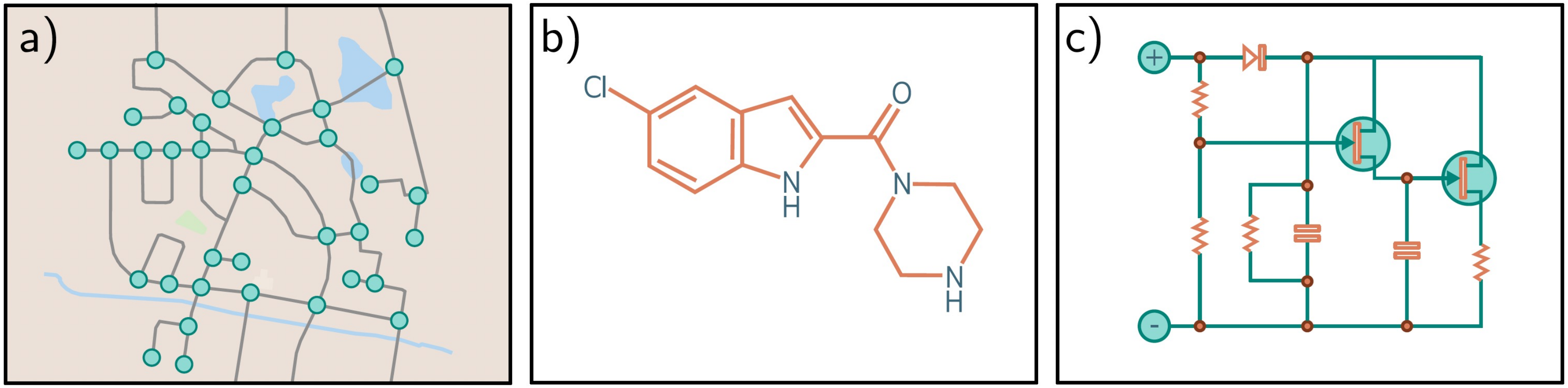
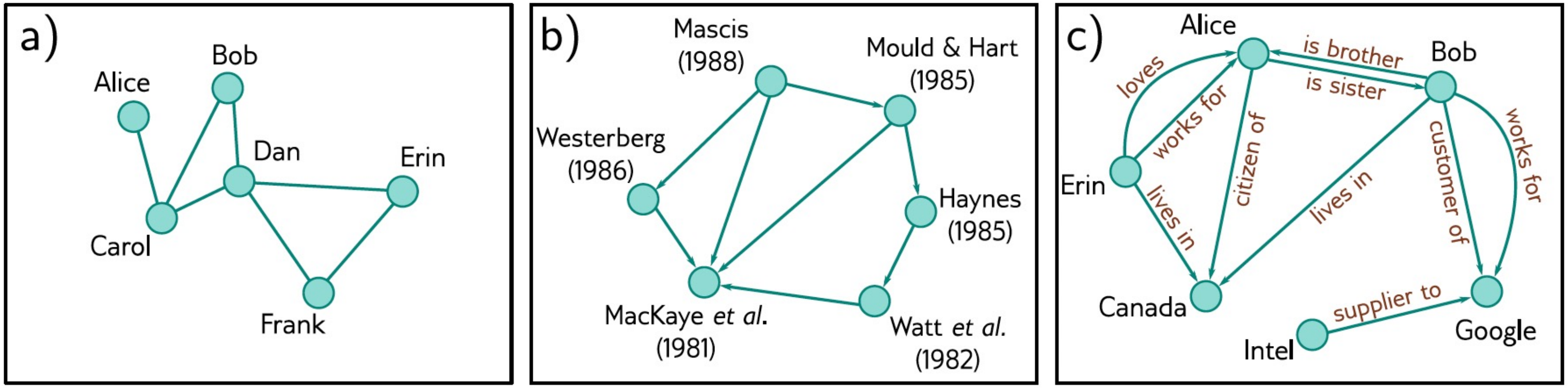


Figure 13.1 Real-world graphs. Some objects, such as a) road networks, b) molecules, and c) electrical circuits, are naturally structured as graphs.

Types of graphs

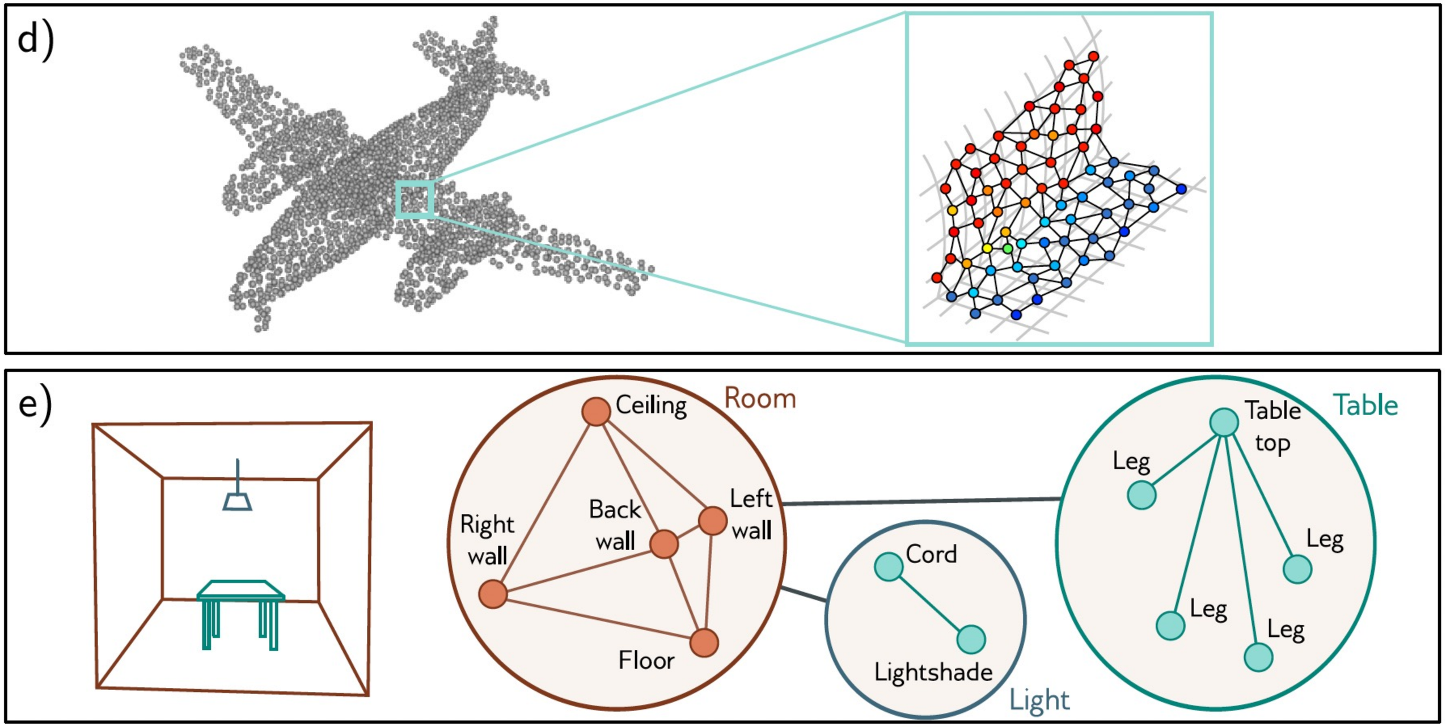
- a) social network is an undirected graph
- b) citation network is a directed graph
- c) Knowledge graph is a directed heterogeneous multigraph



UC MERCED

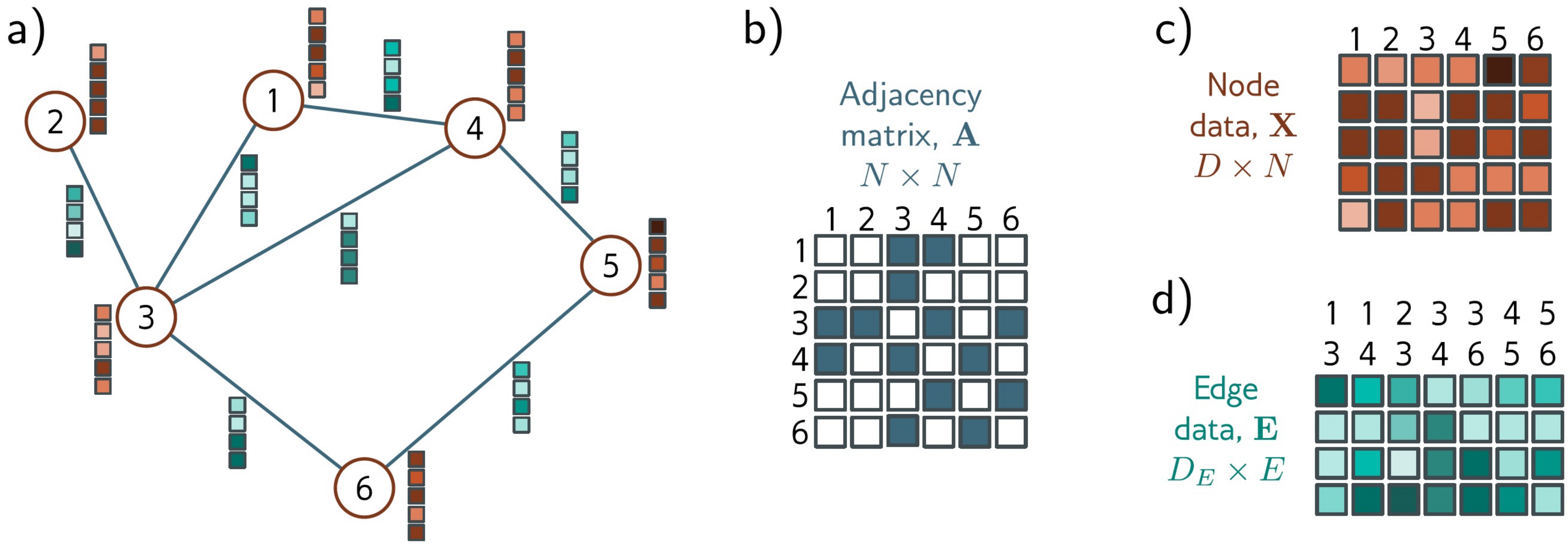
Types of graphs

- d) point cloud as a geometric graph
- e) Scene graph is hierarchical



Graph representation

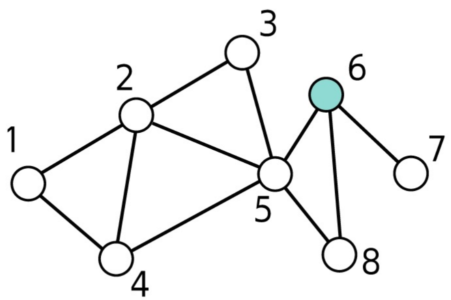
- ❑A graph is defined as a tuple G = (V, E)
  - ❑where V is a set of nodes
  - ❑and E is a set of edges
- ❑An example graph with 6 nodes and 7 edges



Properties of adjacency matrix

- ❑ $A^k x$ gives the number of paths of length k to a node

a)



b)

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

c)

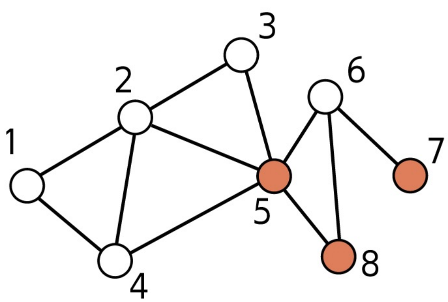
$$\mathbf{A}^2 = \begin{bmatrix} 2 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 2 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 & 0 & 1 \\ 1 & 2 & 2 & 3 & 1 & 1 & 0 & 1 \\ 2 & 2 & 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

d)

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

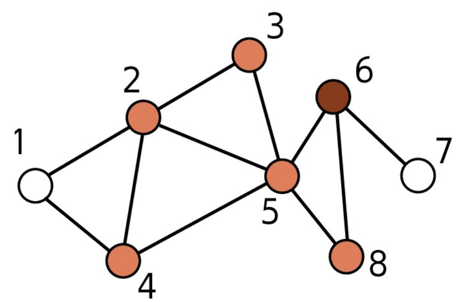
e)

$$\mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



f)

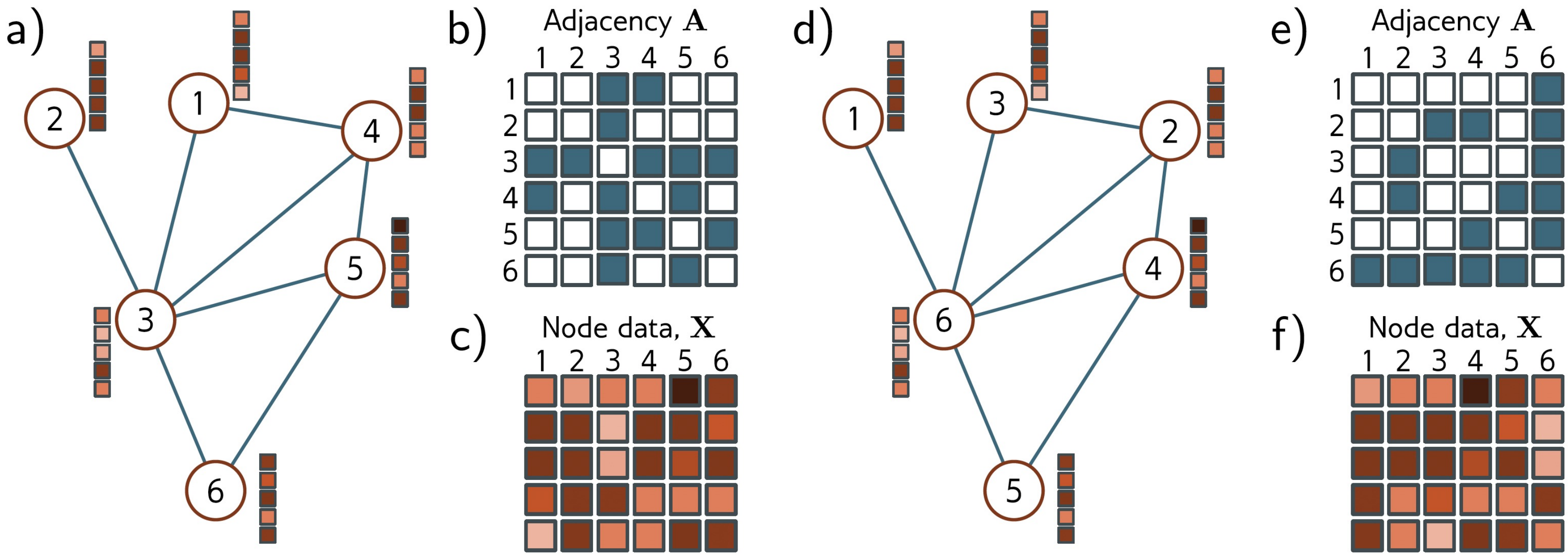
$$\mathbf{A}^2\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$



UCMERCED

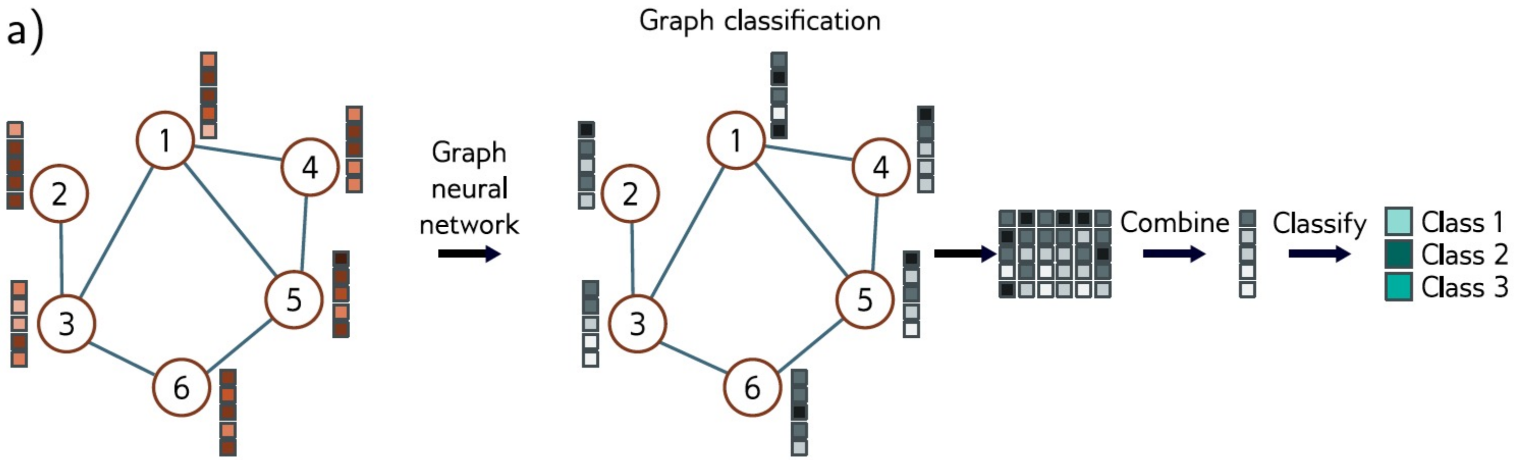
Permutation invariance

- A graph neural network should be permutation invariant
- CNN? MLP?



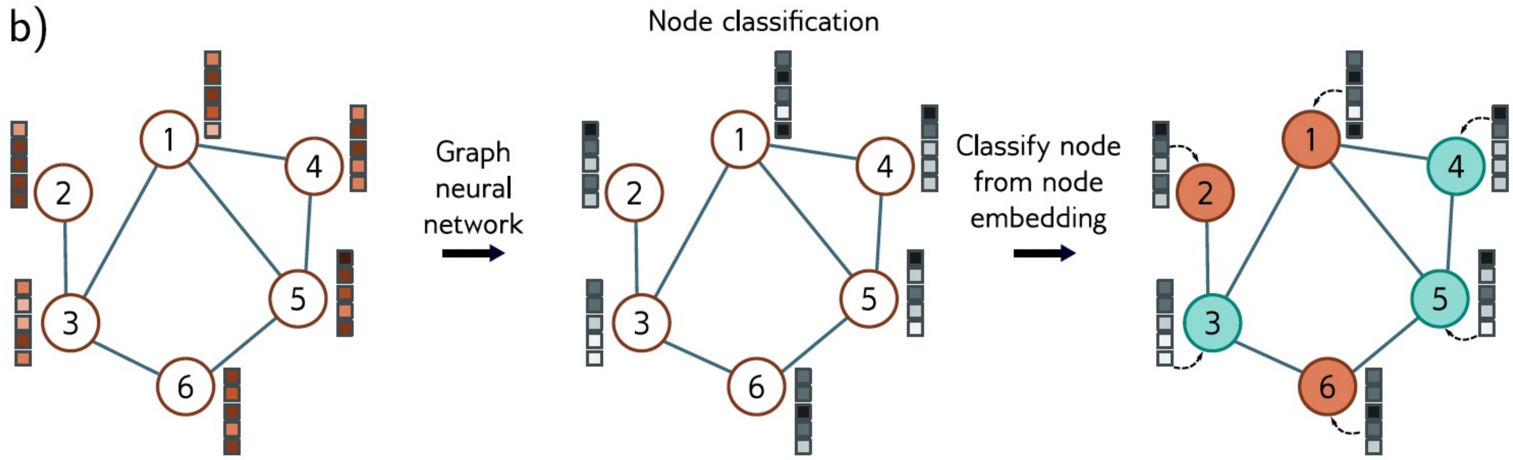
Tasks on graphs

- ❑Graph classification



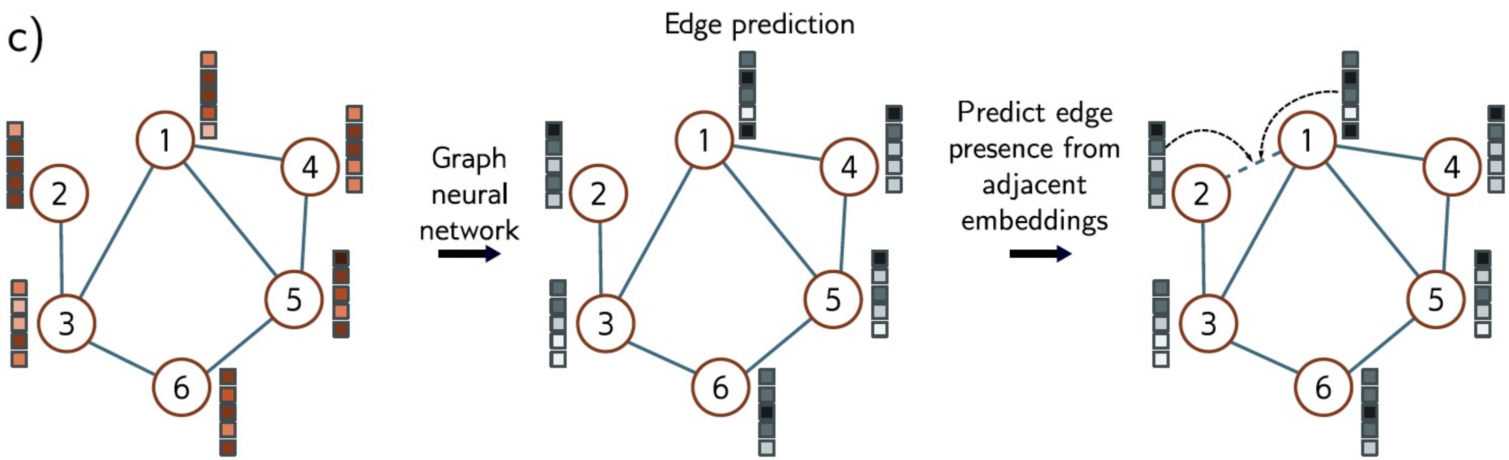
Tasks on graphs

- ❑Node classification



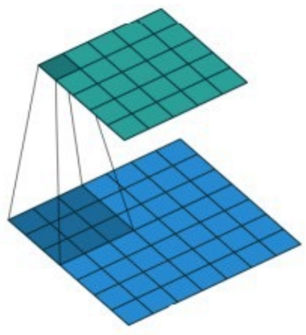
Tasks on graphs

- ❑Edge classification

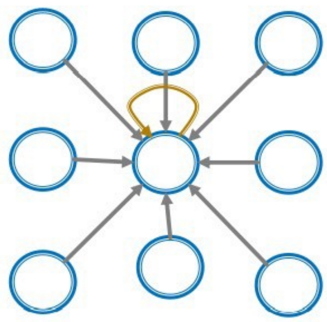


Graph convolution

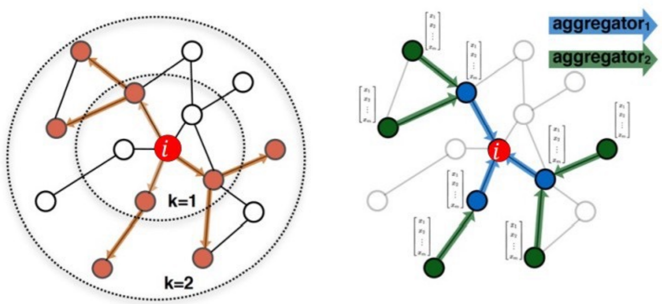
- ❑Convolution on a neighborhood



CNN: Pixel convolution

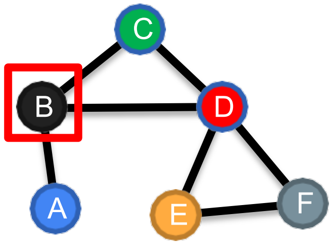


CNN: Pixel convolution
(as a graph)

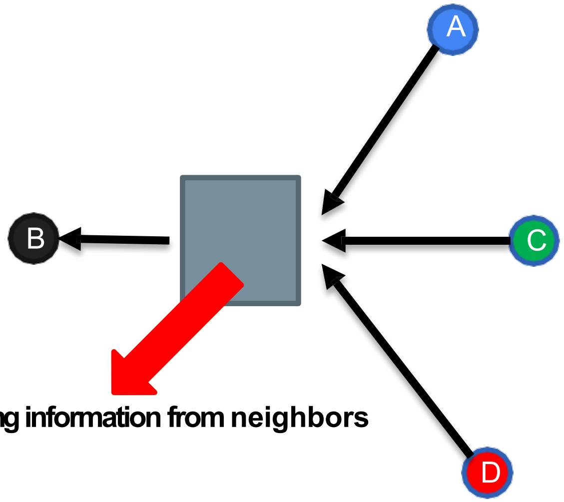


GNN: Graph convolution

Graph convolution



Generating features from nodes 1 **hop** away
A,C, and **D**



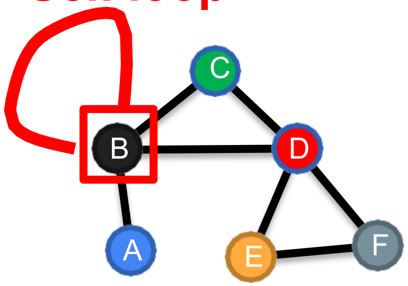
Processing information from neighbors

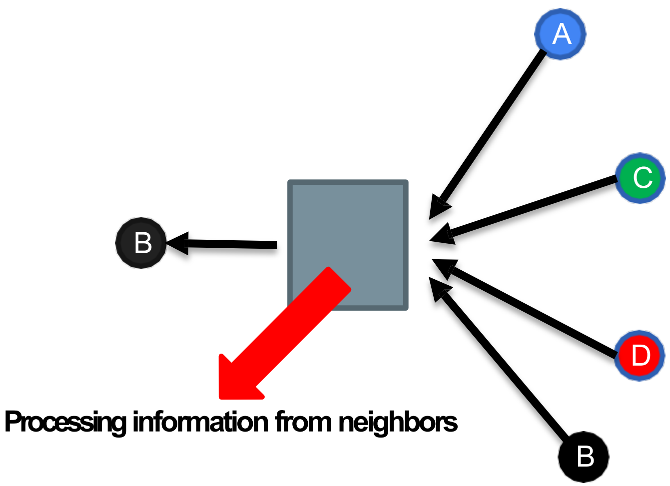
Now embedding for node **B** has information from A,C, and D.

Graph convolution

Generating features from nodes **1 hop** away
A,C, and **D**

Self loop



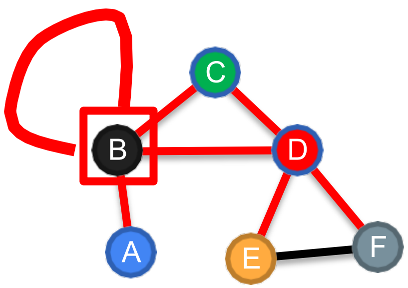


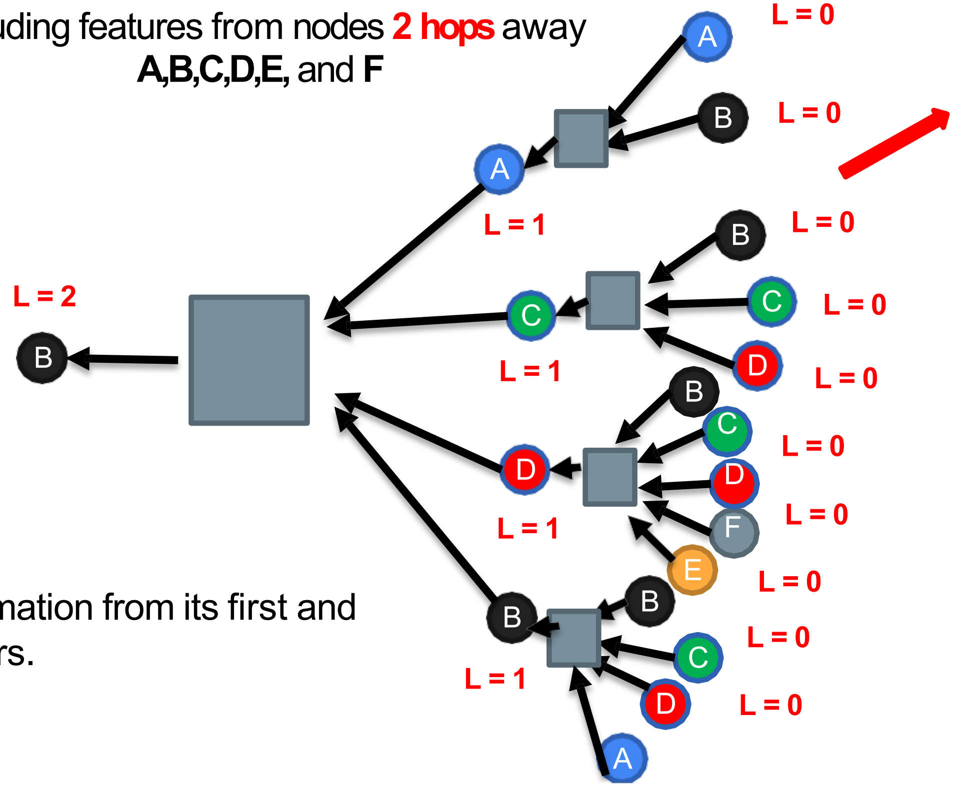
But we don't want to forget information about B either.

Now embedding for node **B** has information from A,C,D, **and B itself**.

Two layers of graph convolution

Including features from nodes **2 hops** away
A,B,C,D,E, and **F**





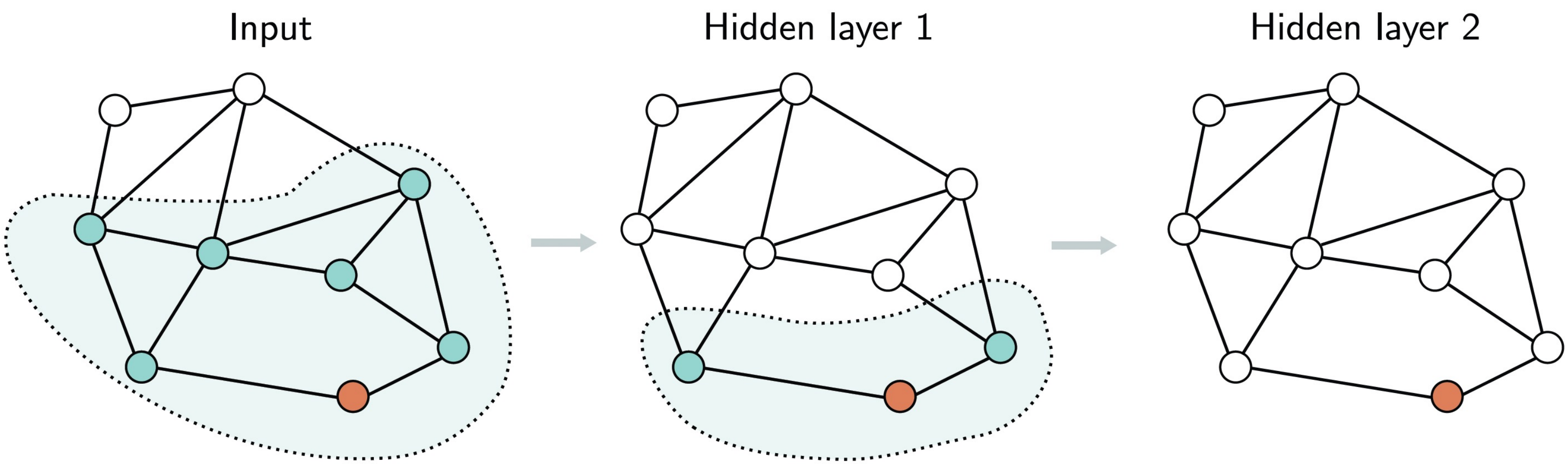
L=0 represents input node features.

Embedding for node **B** (at layer 2) has information from its first and second hop neighbors.

UCMERCED

Receptive fields in graph neural networks

- Increasing receptive fields with more layers/hops



Graph convolution

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

Initial 0-th layer embeddings are equal to node features

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, \ldots, K\}$$

Previous layer embedding of *v*

Average of neighbor's previous layer embeddings

Non-linearity (e.g., ReLU)

$$\mathbf{z}_v = \mathbf{h}_v^K$$

Embedding after K layers of neighborhood aggregation

Graph convolution

- **Recap: Simple neighborhood aggregation:**

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

- Graph convolutional operator:
  - Aggregates messages across neighborhoods, $N(v)$
  - $\alpha_{vu} = 1/|N(v)|$ is the **weighting factor (importance)** of node u's message to node v
  - $\Longrightarrow$ α_{vu} is defined **explicitly** based on the structural properties of the graph
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Graph attention network

Can we do better than simple neighborhood aggregation?

Can we let weighting factors α_{vu} to be implicitly defined?

- Goal: Specify arbitrary importances to different neighbors of each node in the graph
- **Idea:** Compute embedding $\boldsymbol{h}$ of each node in the graph following an **attention strategy:**
  - Nodes attend over their neighborhoods' message
  - Implicitly specifying different weights to different nodes in a neighborhood

[Velickovic et al., ICLR 2018; Vaswani et al., NIPS 2017]

Graph attention

- Let α_{vu} be computed as a byproduct of an **attention mechanism $\boldsymbol{a}$:**
  - Let a compute **attention coefficients $\boldsymbol{e_{vu}}$** across pairs of nodes u, v based on their messages:

$$e_{vu} = a(\boldsymbol{W}_k \boldsymbol{h}_u^{k-1}, \boldsymbol{W}_k \boldsymbol{h}_v^{k-1})$$

    - **$\boldsymbol{e_{vu}}$ indicates the importance of node $\boldsymbol{u}$'s message to node $\boldsymbol{v}$**
  - **Normalize coefficients** using the softmax function in order to be comparable across different neighborhoods:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

$$\boldsymbol{h}_v^k = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \boldsymbol{W}_k \boldsymbol{h}_u^{k-1}\right)$$

Next: What is the form of attention mechanism a?

UCMERCED

An application of Graph Neural Network
- SuperGlue feature matching

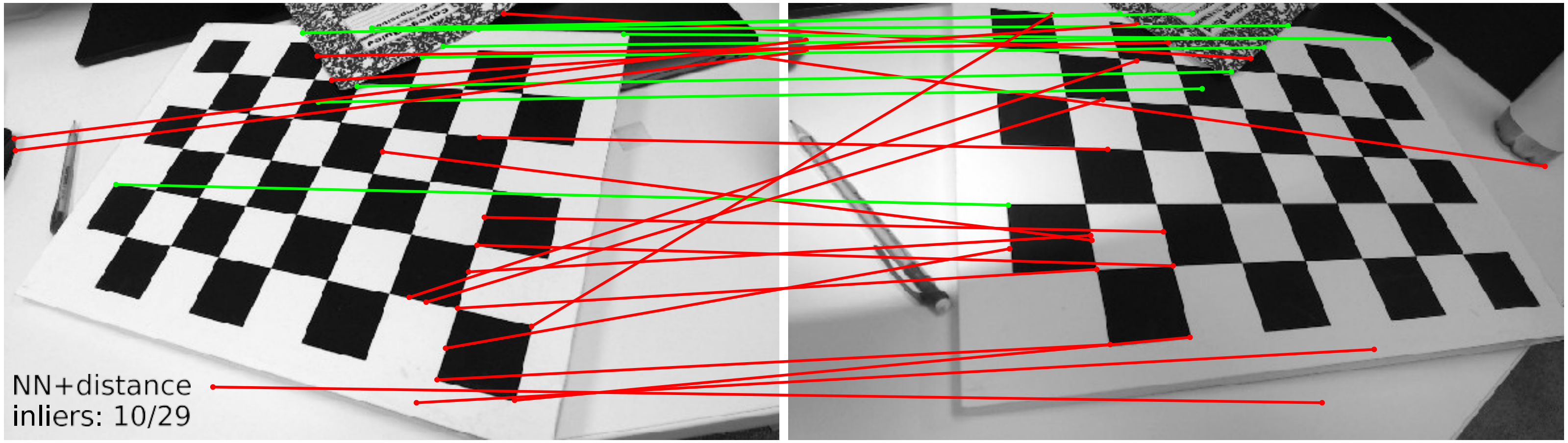
SuperGlue:
Learning Feature Matching with Graph Neural Networks

Paul-Edouard Sarlin[1]
Daniel DeTone[2]
Tomasz Malisiewicz[2]
Andrew Rabinovich[2]

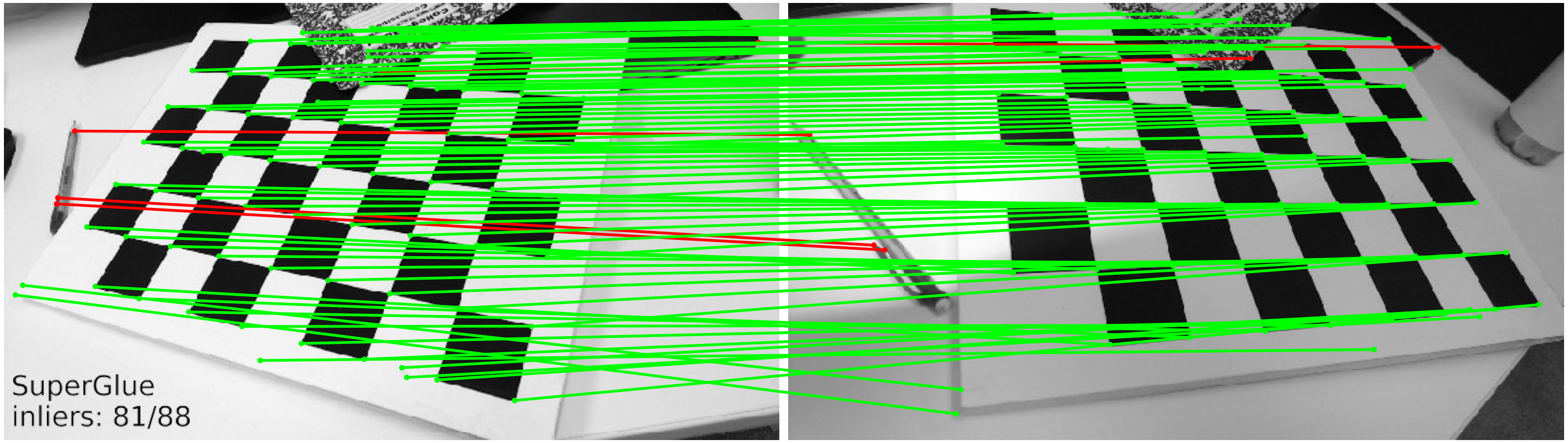
ETH zürich
magic leap

The importance of context

no SuperGlue



with SuperGlue

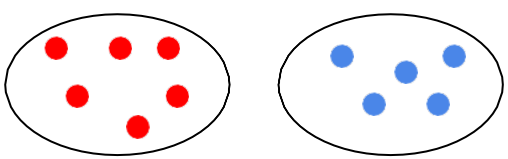


Problem formulation

Inputs → 

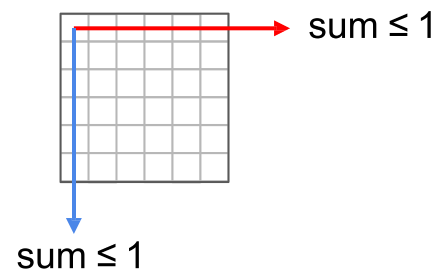
→ **Outputs**

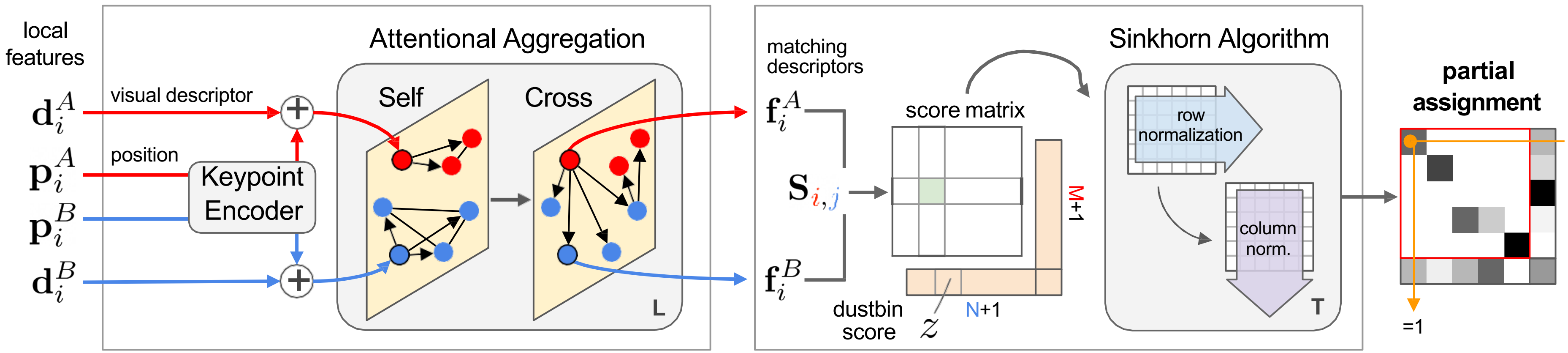
- Images A and B
- **2 sets of M, N local features**
  - Keypoints: $\mathbf{p}_i := (x, y, c)_i$
    - Coordinates (x, y)
    - Confidence c
  - Visual descriptors: $\mathbf{d}_i$



Single a match per keypoint
+ occlusion and noise
→ a **soft partial assignment**:

$$\mathbf{P} \in [0, 1]^{M \times N}$$





A Graph Neural Network with attention

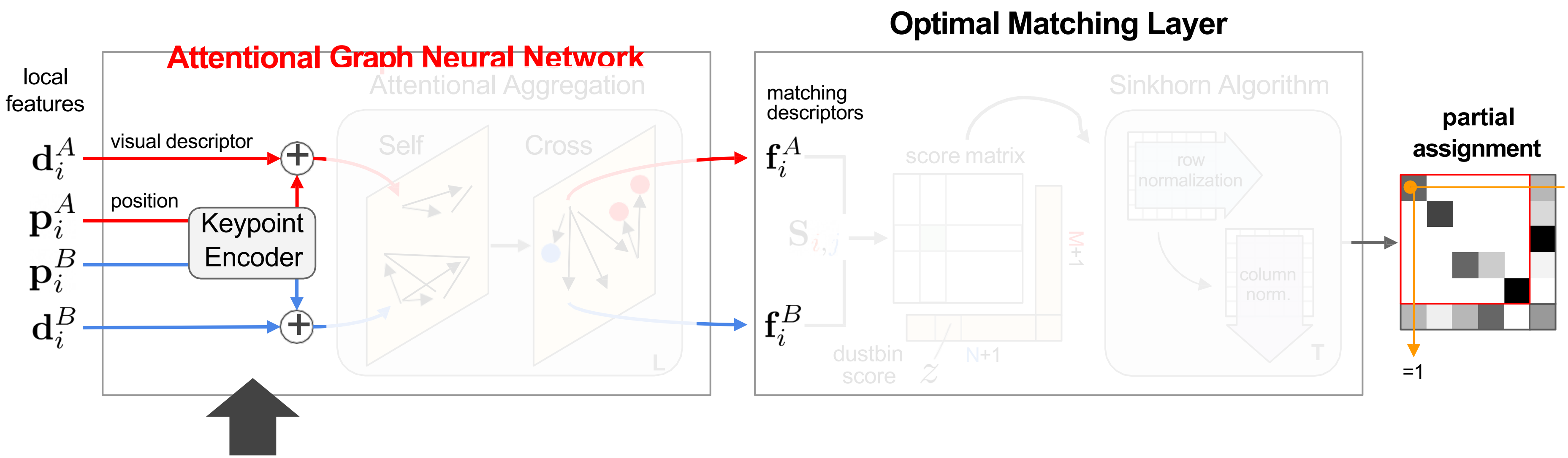
Solving a partial assignment problem

Encodes **contextual cues** & priors

Reasons about the 3D scene

Differentiable **solver**

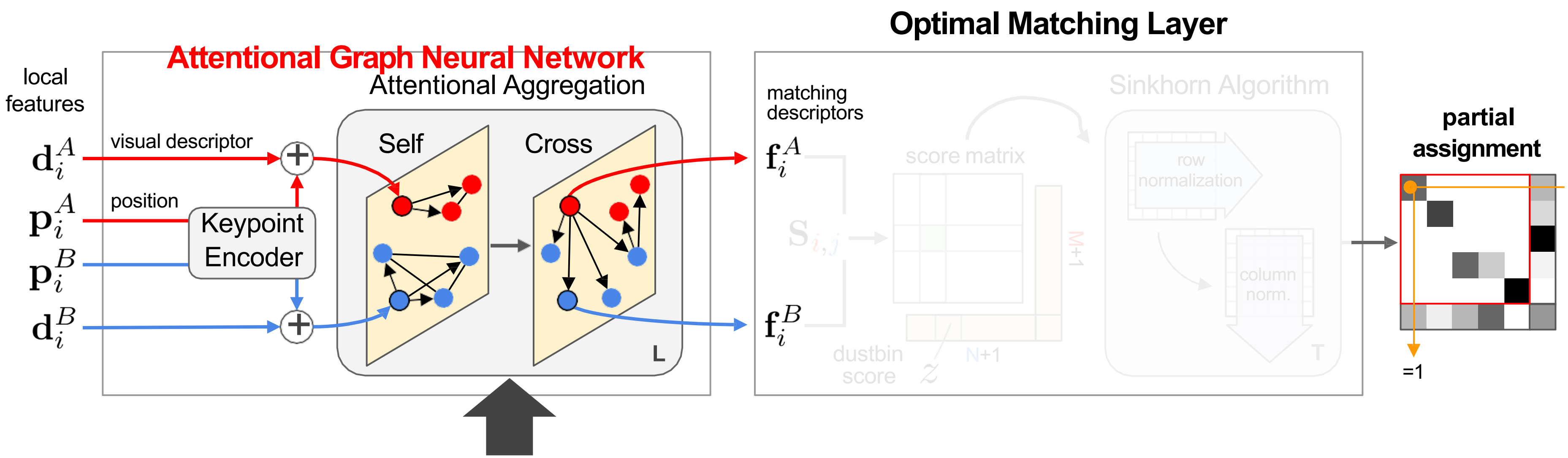
Enforces the assignment constraints = **domain knowledge**



- Initial representation for each keypoints i : ${}^{(0)}\mathbf{x}_i$
- Combines visual appearance and position with an MLP:

$$ {}^{(0)}\mathbf{x}_i = \mathbf{d}_i + \mathrm{MLP}\left(\mathbf{p}_i\right) $$

Multi-Layer Perceptron

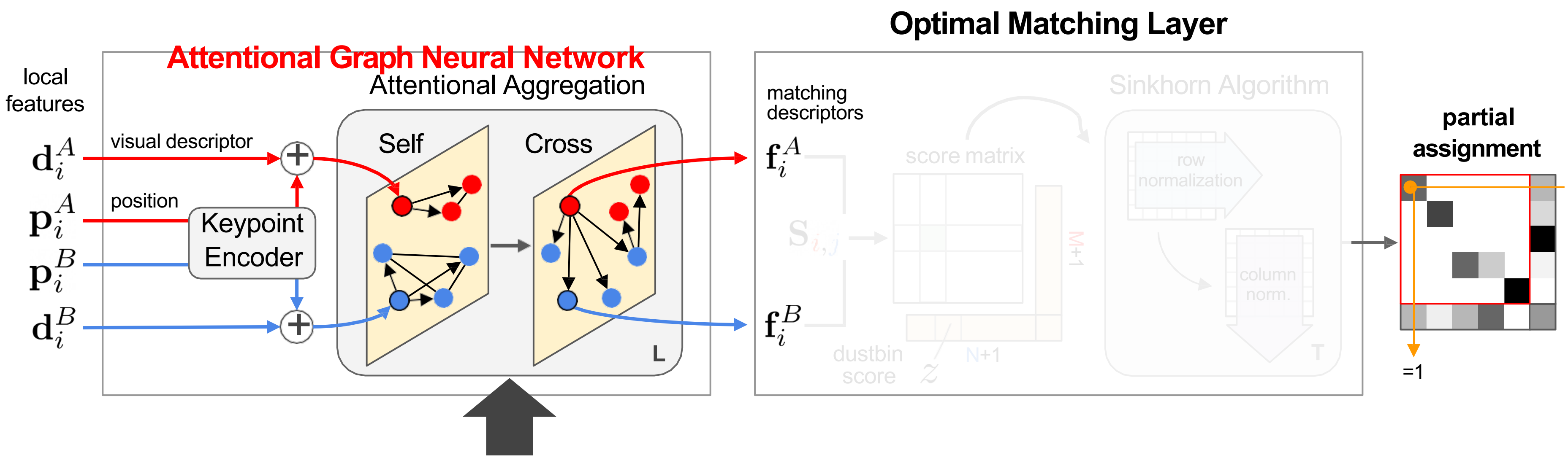


Update the representation based on other keypoints:

- in the same image: "**self**" edges
- in the other image: "**cross**" edges

$$ {}^{(\ell)}\mathbf{x}_i^A \longrightarrow {}^{(\ell+1)}\mathbf{x}_i^A $$

→ A complete **graph** with two types of edges



Update the representation using a **Message Passing Neural Network**

$$^{(\ell+1)}\mathbf{x}_i^A = {}^{(\ell)}\mathbf{x}_i^A + \mathrm{MLP}\left(\left[{}^{(\ell)}\mathbf{x}_i^A \,||\, \mathbf{m}_{\mathcal{E}\rightarrow i}\right]\right)$$

the message

Attentional Aggregation

- Compute the **message** $\mathbf{m}_{\mathcal{E}\rightarrow i}$ using **self** and **cross attention**
- Soft database retrieval: query $\mathbf{q}_i$, key $\mathbf{k}_j$, and value $\mathbf{v}_j$

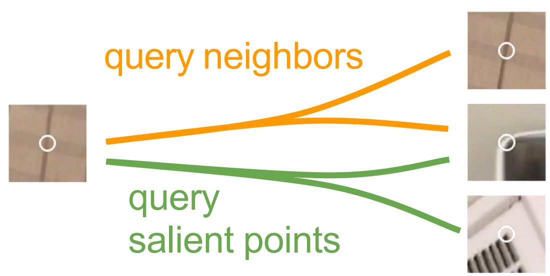
$$\mathbf{m}_{\mathcal{E}\rightarrow i} = \sum_{j:(i,j)\in\mathcal{E}} \alpha_{ij}\mathbf{v}_j$$

$$\alpha_{ij} = \mathrm{Softmax}_j\left(\mathbf{q}_i^\top \mathbf{k}_j\right)$$

$$\mathbf{q}_i = \mathbf{W}_1 {}^{(\ell)}\mathbf{x}_i + \mathbf{b}_1$$

$$\begin{bmatrix}\mathbf{k}_j \\ \mathbf{v}_j\end{bmatrix} = \begin{bmatrix}\mathbf{W}_2 \\ \mathbf{W}_3\end{bmatrix} {}^{(\ell)}\mathbf{x}_j + \begin{bmatrix}\mathbf{b}_2 \\ \mathbf{b}_3\end{bmatrix}$$

$\mathbf{x}_i$ = [tile, position (70, 100)]



= [tile, pos. (80, 110)]

= [corner, pos. (60, 90)]

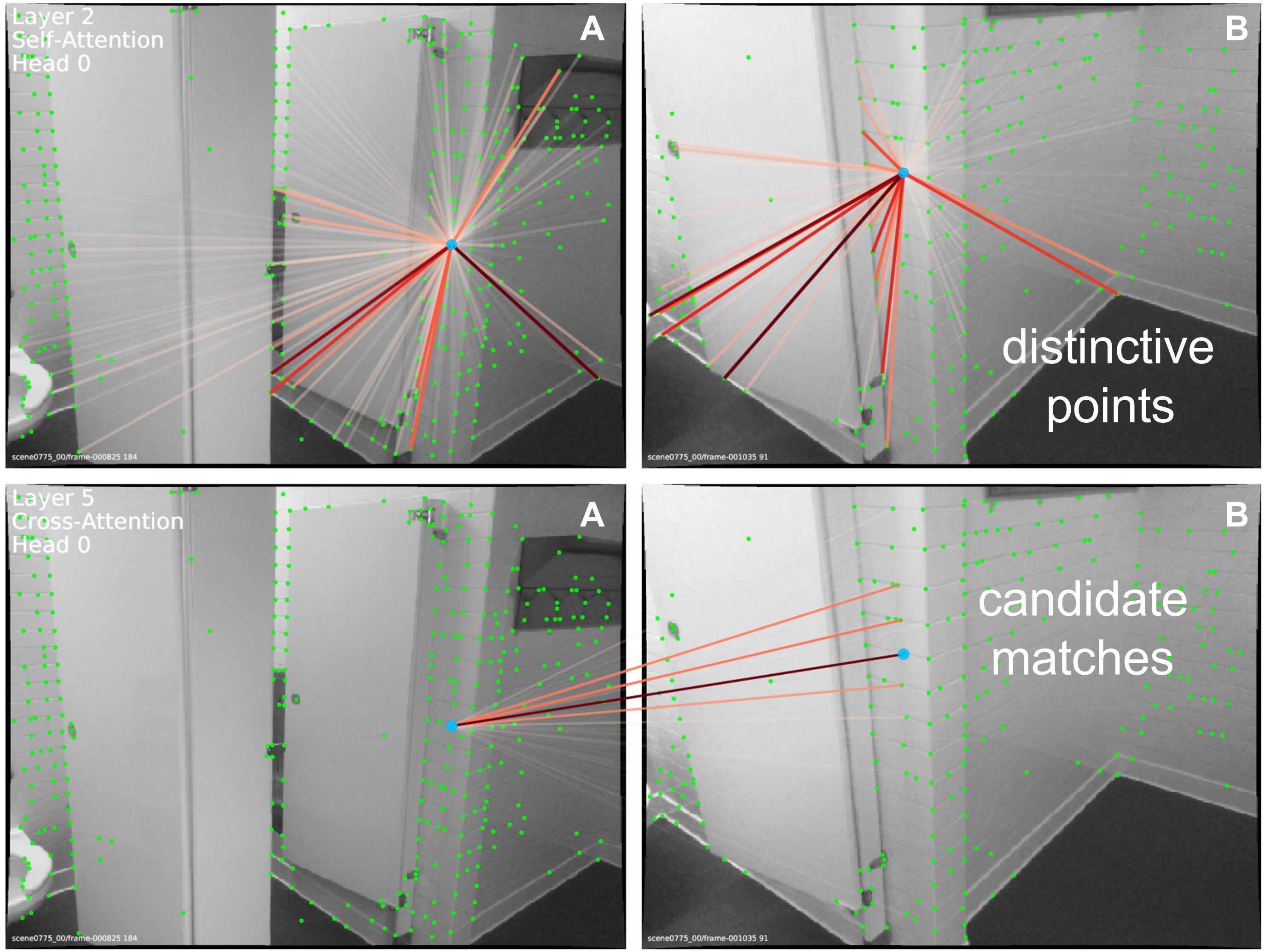
= [grid, pos. (400, 600)]

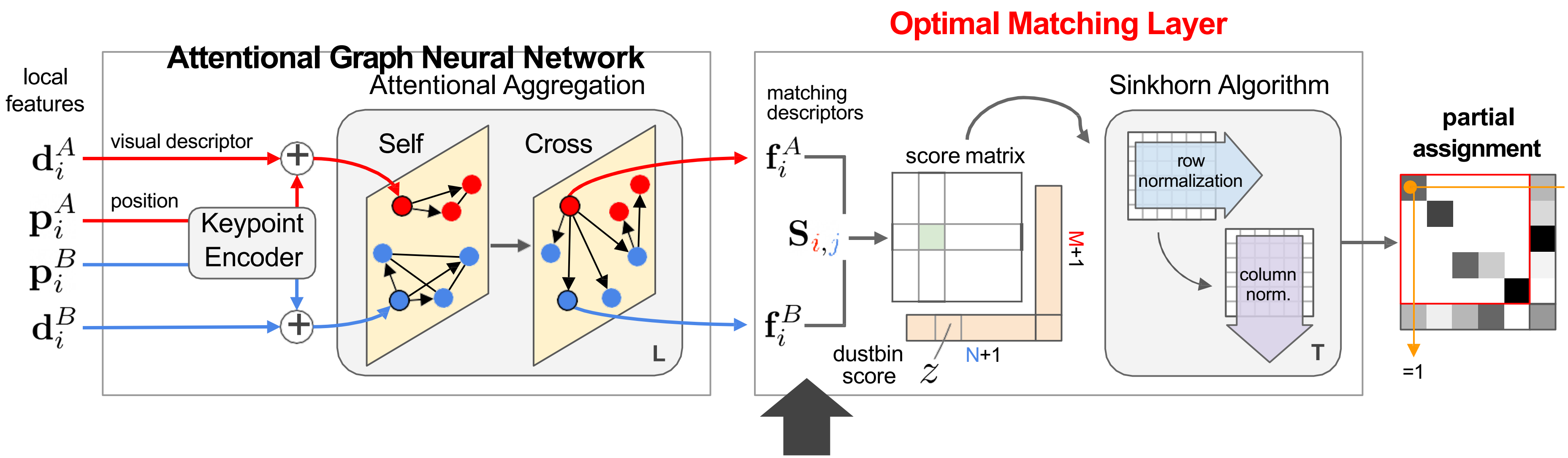
[Vaswani et al, 2017]

Self-attention
= intra-image information flow

Cross-attention
= inter-image

Attention builds a **soft**, **dynamic**, **sparse graph**

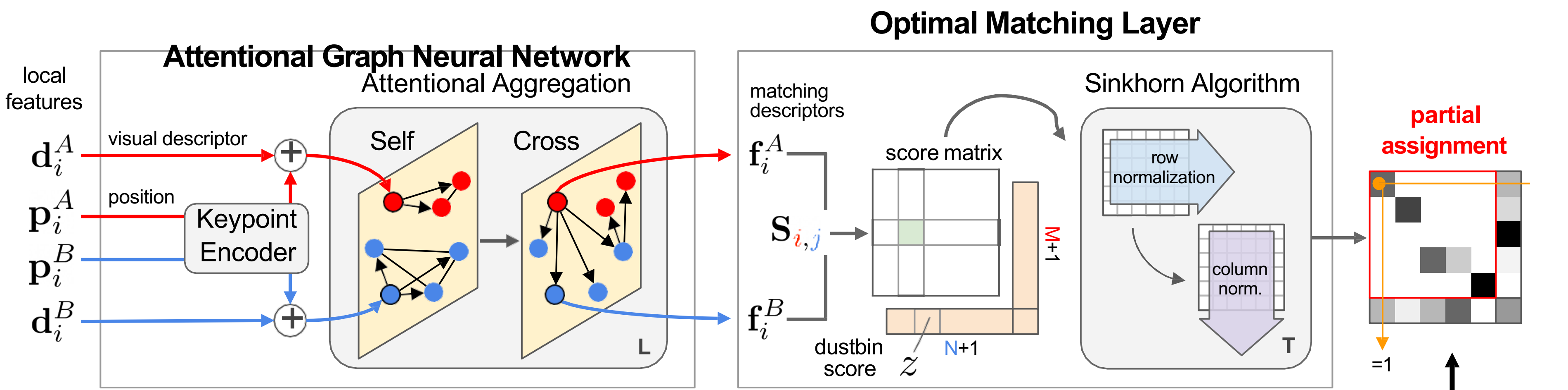




Compute a **score matrix** $\mathbf{S} \in \mathbb{R}^{M \times N}$ for all matches:

$$\mathbf{f}_i^A = \mathbf{W} \cdot {}^{(L)}\mathbf{x}_i^A + \mathbf{b}$$

$$\mathbf{S}_{i,j} = < \mathbf{f}_i^A, \mathbf{f}_j^B >$$

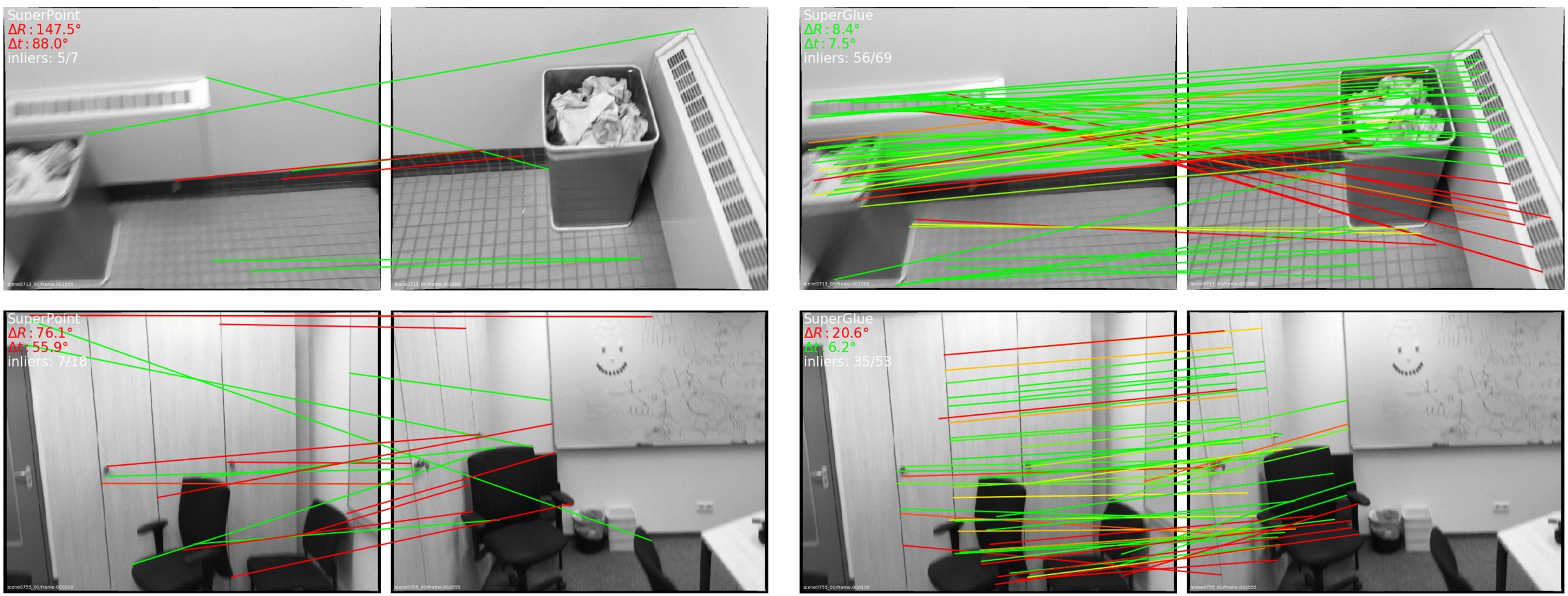


- Compute **ground truth correspondences** from pose and depth
- Find which keypoints should be **unmatched**
- Loss: maximize the log-likelihood $\bar{\mathbf{P}}_{i,j}$ of the GT cells

Results: indoor - ScanNet

SuperPoint + NN + heuristics

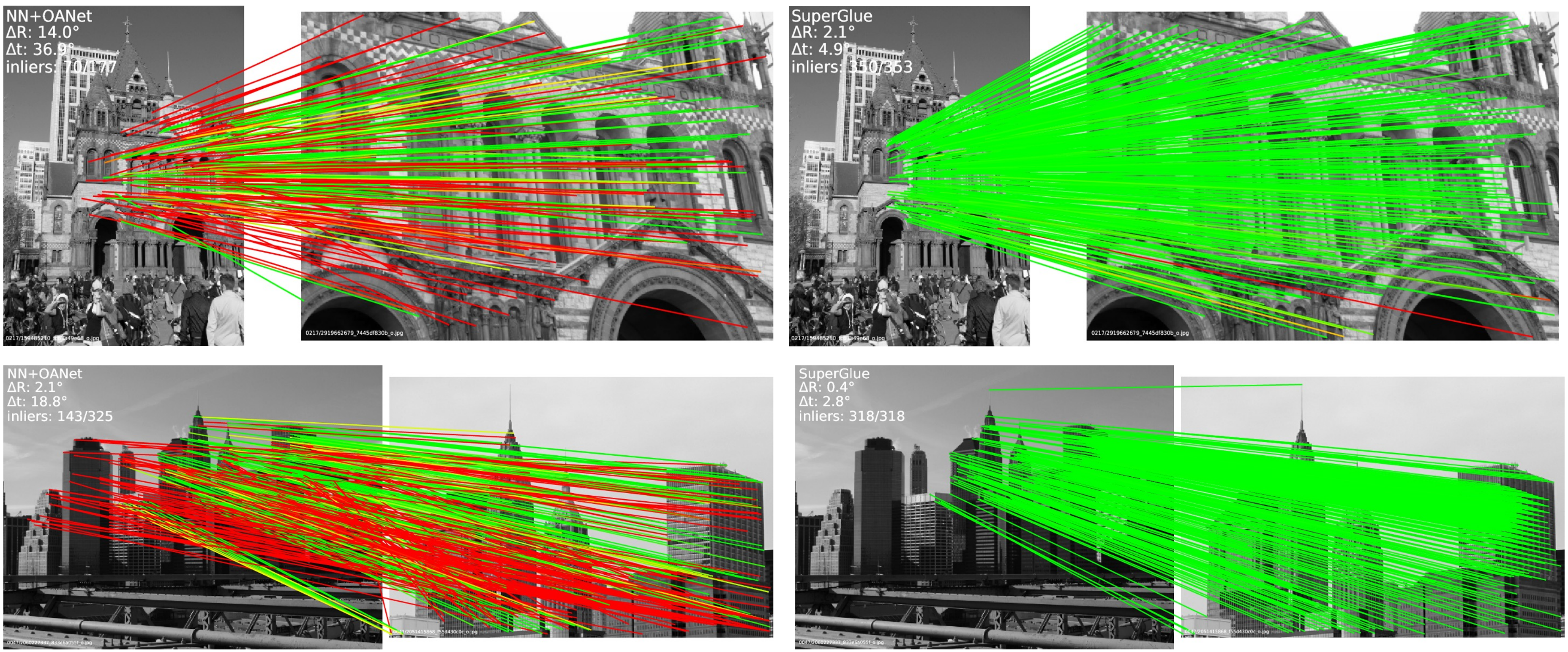
SuperPoint + **SuperGlue**



SuperGlue: more **correct matches** and fewer **mismatches**

Results: outdoor - SfM

SuperPoint + NN + OA-Net (inlier classifier) | SuperPoint + **SuperGlue**



SuperGlue: more **correct matches** and fewer **mismatches**