Learning Combinatorial Embedding Networks for Deep Graph Matching

Runzhong Wang[1,2] Junchi Yan[1,2]* Xiaokang Yang[2]
[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University
[2] MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
{runzhong.wang,yanjunchi,xkyang}@sjtu.edu.cn

Abstract

Graph matching refers to finding node correspondence between graphs, such that the corresponding node and edge's affinity can be maximized. In addition with its NP-completeness nature, another important challenge is effective modeling of the node-wise and structure-wise affinity across graphs and the resulting objective, to guide the matching procedure effectively finding the true matching against noises. To this end, this paper devises an end-to-end differentiable deep network pipeline to learn the affinity for graph matching. It involves a supervised permutation loss regarding with node correspondence to capture the combinatorial nature for graph matching. Meanwhile deep graph embedding models are adopted to parameterize both intra-graph and cross-graph affinity functions, instead of the traditional shallow and simple parametric forms e.g. a Gaussian kernel. The embedding can also effectively capture the higher-order structure beyond second-order edges. The permutation loss model is agnostic to the number of nodes, and the embedding model is shared among nodes such that the network allows for varying numbers of nodes in graphs for training and inference. Moreover, our network is class-agnostic with some generalization capability across different categories. All these features are welcomed for real-world applications. Experiments show its superiority against state-of-the-art graph matching learning methods.

1. Introduction and Preliminaries

Graph matching (GM) refers to establishing node correspondences between two or among multiple graphs. Graph matching incorporates both the unary similarity between nodes and pairwise [7, 14] (or even higher-order [21, 29, 43]) similarity between edges from separate graphs to find a matching such that the similarity between the matched graphs is maximized. By encoding the high-order geometrical information in the matching procedure, graph matching in general can be more robust to deformation noise and outliers. For its expressiveness and robustness, graph matching has lied at the heart of many computer vision applications e.g. visual tracking, action recognition, robotics, weak-perspective 3-D reconstruction – refer to [40] for a more comprehensive survey on graph matching applications.

Due to its high-order combinatorial nature, graph matching is in general NP-complete [13] such that researchers employ approximate techniques to seek inexact solutions. For the classic setting of two-graph matching between graphs $\mathcal{G}_1$, $\mathcal{G}_2$, the problem can be written by the following general quadratic assignment programming (QAP) problem [25]:

$$J(\mathbf{X}) = \text{vec}(\mathbf{X})^\top \mathbf{K} \text{vec}(\mathbf{X}), \quad (1)$$
$$\mathbf{X} \in \{0,1\}^{N\times N}, \quad \mathbf{X}\mathbf{1} = \mathbf{1}, \quad \mathbf{X}^\top \mathbf{1} \leq \mathbf{1}$$

where $\mathbf{X}$ is a permutation matrix indicating the node correspondence, and $\mathbf{K} \in \mathbb{R}^{N^2\times N^2}$ is the so-called affinity matrix [22] whose diagonal elements and off-diagonal ones encode the node-to-node and edge-to-edge affinity between two graphs, respectively. One popular embodiment of $\mathbf{K}$ in literature is $\mathbf{K}_{ia,jb} = \exp\left(\frac{(\mathbf{f}_{ij}-\mathbf{f}_{ab})^2}{\sigma^2}\right)$ where $\mathbf{f}_{ij}$ is the feature vector of the edge ij, which can also incorporate the node similarity when node index $ia = jb$.

Eq. (1) is called Lawler's QAP [20]. It can incorporate other forms e.g. Koopmans-Beckmann's QAP [25]:

$$J(\mathbf{X}) = \text{tr}(\mathbf{X}^\top \mathbf{F}_1 \mathbf{X} \mathbf{F}_2) + \text{tr}(\mathbf{K}_p^\top \mathbf{X}) \quad (2)$$

where $\mathbf{F}_1 \in \mathbb{R}^{N\times N}$, $\mathbf{F}_2 \in \mathbb{R}^{N\times N}$ are weighted adjacency matrices of graph $\mathcal{G}_1$, $\mathcal{G}_2$ respectively, and $\mathbf{K}_p$ is the node-to-node affinity matrix. Its connection to the Lawler's QAP can be established by setting $\mathbf{K} = \mathbf{F}_2 \otimes \mathbf{F}_1$.

Beyond the second-order affinity modeling, recent methods also explore the way of utilizing higher-order affinity information. Based on tensor marginalization as adopted by several hypergraph matching works [5, 9, 43, 46]:

$$\mathbf{x}^* = \arg\max(\mathbf{H} \otimes_1 \mathbf{x} \otimes_2 \mathbf{x} \ldots \otimes_m \mathbf{x}) \quad s.t. \quad (3)$$
$$\mathbf{X}\mathbf{1} = \mathbf{1}, \mathbf{X}^\top \mathbf{1} \leq \mathbf{1}, \mathbf{x} = \text{vec}(\mathbf{X}) \in \{0,1\}^{N^2\times 1}$$

*Corresponding author. This work is supported by National Key Research and Development Program of China (2016YFB1001003), STCSM (18DZ1112300), NSFC (61602176).

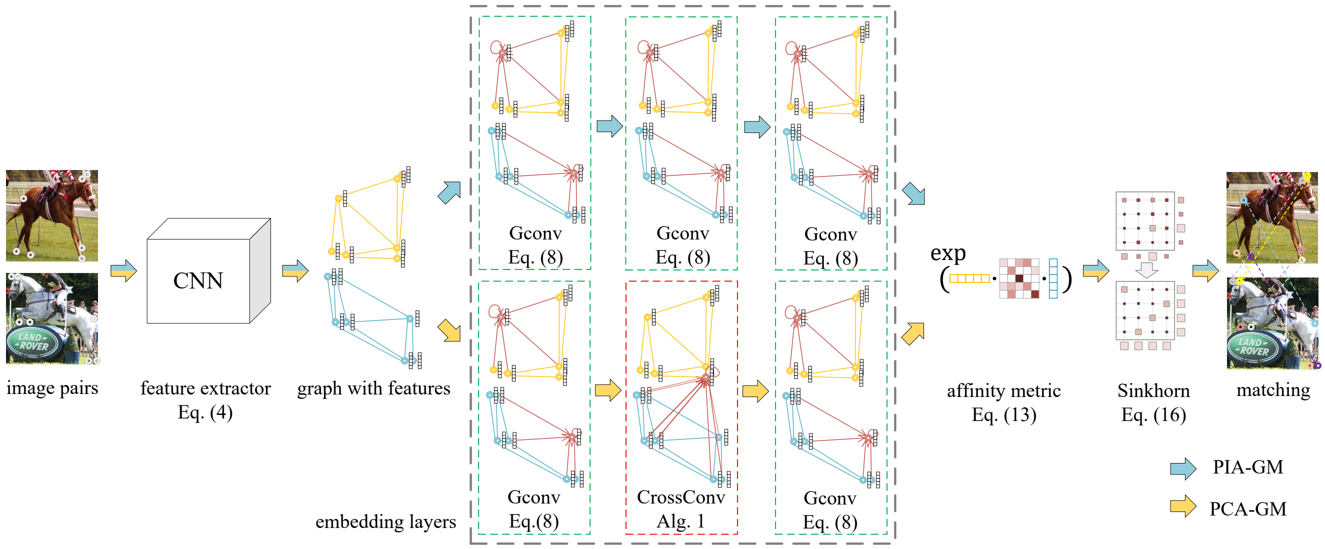


Figure 1. Overview of our proposed permutation based intra-graph affinity (PIA-GM) and cross-graph affinity (PCA-GM) approaches for deep combinatorial learning of graph matching. The CNN features are extracted from image pairs followed by node embedding and Sinkhorn operation for matching. The CNN model, embedding model and affinity metric are all learnable in an end-to-end fashion.

where m is the affinity order and $\mathbf{H}$ is the m-order affinity tensor whose element encodes the affinity between two hyperedges from the graphs. $\otimes_k$ is the tensor product [21]. Readers are referred to Sec. 3.1 in [9] for details on tensor multiplication. The above works all assume the affinity tensor is invariant w.r.t. the index of the hyperedge pairs.

All the above studies show the generality and importance of the affinity model for graph matching. However, traditional affinity methods mostly rely on a predefined affinity function (or distance), e.g. a Gaussian kernel with Euclid distance in the node and edge feature space. We believe that such a predefined parametric affinity model has limited flexibility to capture the structure of a real-world matching task, whereby the affinity metric can be arbitrary and call for models with enough high capacity to approximate. This challenge is more pronounced in the presence of noise and outliers which are ubiquitous in practical settings. Based on an inappropriate affinity model, the matching solver can be more struggling as the global optimum regarding with the affinity model may even not correspond to the ground truth matching solution – due to the biased affinity objective function as input for combinatorial optimization.

Hence it calls for effective affinity modeling across graphs. It is orthogonal to the major line of previous efforts on devising combinatorial solvers using predefined affinity model [7, 9, 14, 21]. The contributions of this paper are:

i) We develop a novel supervised deep network based pipeline for graph matching, whereby the objective involves the permutation loss based on a Sinkhorn net rather than structured max-margin loss [6] and pixel offset loss [45]. We argue that the permutation loss is a more inherent choice for the combinatorial nature graph matching (by relaxing it into linear assignment). Meanwhile, the permutation loss allows for the flexible handling of arbitrary number of nodes of graph for matching. In contrast, the number of nodes for matching in a graph is fixed and predefined in the problem structure in [6]. To our best knowledge, this is the first time for adopting a permutation loss for learning graph matching – a natural choice for its combinatorial nature.

ii) Our graph matching nets learn the node-wise feature (extracted from image in this paper) and the implicit structure information (including hyper-edge) by employing a graph convolutional network, together with the node-to-node cross-graph affinity function using additional layers. As such, the intra-graph information and cross-graph affinity are jointly learned given ground truth correspondence. Our network embeds both the node (image patch) feature and structure into the node-wise vector, and the node-to-node affinity layers are shared among all nodes. Such a design also allows for different numbers of nodes in different graph pairs for training and testing. To our best knowledge, this is the first time for adopting a graph neural network for learning graph matching (at least in computer vision).

iii) Experimental results including ablation studies show the effectiveness of our devised components including the permutation loss, the node-wise feature extract layer, graph convolutional network based node embedding, and the cross-graph affinity component. In particular, our method outperforms the deep learning peer method [45] in terms of matching accuracy. Our method also outperforms [6] in accuracy while being more flexible as the method in [6] requires constant number of nodes for matching in both training and testing sets. We also show the learning capability of our approach even when the training set and test set are from different object categories, which also outperforms [45].

2. Related Work

This paper focuses on learning of graph matching. Readers are referred to [42] for a comprehensive acquaintance.

2.1. Modeling and Learning Affinity

Recently a number of studies show various techniques for affinity function learning. Based on the extent to which ground truth correspondence information is used for training, methods are either unsupervised [23], semi-supervised [24], or supervised [4, 6, 45].

Previous graph matching affinity learning methods are mostly based on simple and shallow parametric models, which use popular distances (typically weighted Euclid distance) in the node and edge feature space plus a similarity kernel function (e.g. Gaussian kernel) to derive the final affinity score. In particular, a unified (shallow) parametric graph structure learning model is devised between two graphs in a vector form $\Phi(\mathcal{G}_1, \mathcal{G}_2, \pi)$ [6]. The authors in [6] observe that the above simple model can incorporate most previous shallow learning models, including [4, 23, 39]. Therefore, this method will be compared in our experiment.

There is a seminal work [45] presenting a method adopting deep neural networks for learning the affinity matrix for graph matching. However, in Sec. 3.7 we show that their pixel offset based loss function does not fit well with the combinatorial nature of graph matching. In addition, node embedding is not considered which is able to effectively capture the local structure of the node, which can go beyond second-order for more effective affinity modeling.

2.2. Graph Neural Networks and Embedding

Deep neural networks have been proven effective on spatial and sequential data, with CNN and RNN respectively. Recently, there emerges a number of techniques for extracting high-order node embedding via deep networks, whose input i.e. graph is non-Euclidean data. Specifically, graph neural networks (GNN) [34] have been proposed whereby node features are aggregated from adjacent neighbors and different nodes can share the same transfer function. The output of GNN is invariant to permutations of graph elements. Many variants of GNN have been developed since [34], which is comprehensively discussed in [48]. In particular, the SNDE model [41] is developed for deep node embedding by exploiting the first-order and second-order proximity jointly. Differing from the above deep embedding models, there are some shallow embedding models which are scalable on large networks including DeepWalk [32] based on random walk and node2vec [15] inspired by skip-gram language model [28]. In particular, LINE [38] explicitly defines *first-order* proximity and *second-order* proximity and builds heuristics models for the two proximities. However, these methods, including the SNDE model cannot be used for end-to-end learning for graph matching. For this reason, we adopt the graph convolutional network (GCN) [17] modeling graph structure whose parameters are learnable in an end-to-end fashion.

2.3. Learning of Combinatorial Optimization

Graph matching bears the combinatorial nature. There is an emerging thread using learning to seek efficient solution, especially with deep networks. In [16], the well known NP-hard problem for coloring very large graphs is addressed using deep reinforcement learning. The resulting algorithm can learn new state of the art heuristics for graph coloring. While the Travelling Salesman Problem (TSP) is studied in [18] and the authors propose a graph attention network based method which learns a heuristic algorithm that employs neural network policy to find a tour. Deep learning for node set is also explored in [44] which seeks permutation invariant objective functions to a set of nodes.

In particular, [30] shows a network based approach for solving the quadratic assignment problem. Their work focuses on learning the solver given previous defined affinity matrix. In contrast, this paper presents an end-to-end learning pipeline for learning the affinity function. In this sense, the two methods can be further integrated for practical applications. Moreover, for the less challenging linear assignment problem, which in fact can be solved with polynomial complexity e.g. the Hungarian algorithm [19], there also exist recently proposed network based new methods. The Sinkhorn Network [1] is developed for linear assignment learning in the sense of linear assignment given predefined assignment cost, which is designated to enforce doubly-stochastic regulation on any non-negative square matrix. It has been shown that Sinkhorn algorithm [37] is the approximate and differentiable version of Hungarian algorithm [26]. More recently, the Sinkhorn AutoEncoder is proposed in [31] to minimize Wasserstein distance in AutoEncoders, and the work [10] adopts reinforcement learning for learning a linear assignment solver. The Sinkhorn layer is also adopted on top of a deep convolutional network in DeepPermNet [33], which solves a permutation prediction problem. However, DeepPermNet is not invariant to input permutations and need a predefined node permutation as reference, thus it is unstable for two graph matching.

In comparison, our model consists of an affinity learning component which encodes the structure affinity into node-wise embeddings. As such, graph matching is relaxed into linear assignment solved by the Sinkhorn layer, which is also sometimes called permutation learning in literature.

3. Proposed Approach

We present two models for matching $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$: i) **p**ermutation loss and **i**ntra-graph **a**ffinity

Table 1. Symbol notations. Subscript s indexes image/graph.

I_s	input image s
N	number of nodes in one graph
$\mathbf{A}_s$	adjacency matrix of graph s
$\mathcal{V}_s$	vertex set of graph s
$\mathcal{E}_s$	edge set of graph s
P_{si}	coordinate of keypoint i in image s
$\mathbf{h}_{si}^{(k)}$	feature vector of keypoint i, layer k in graph s
$\mathbf{m}_{si}^{(k)}$	message vector of keypoint i, layer k in graph s
$\mathbf{n}_{si}^{(k)}$	node feature of keypoint i, layer k in graph s
$\mathbf{M}^{(k)}$	affinity matrix on k-th Sinkhorn iteration
$\mathbf{S}$	$N \times N$ matrix representing permutation

based **g**raph **m**atching (**PIA-GM**) and ii) **p**ermutation loss and **c**ross-graph **a**ffinity based one (**PCA-GM**). Both models are built upon a deep network which exploits both image feature and structure jointly, and a Sinkhorn network enabling differentiable permutation prediction and loss backpropagation. PCA-GM adopts an extra cross-graph component which aggregates cross-graph features, while PIA-GM only embeds intra-graph features. Fig. 1 summarizes both PIA-GM and PCA-GM. Symbols are shown in Tab. 1.

The proposed two models consist of a CNN image feature extractor, a graph embedding component, an affinity metric function and a permutation prediction component. Image features are extracted by CNN (VGG16 in the paper) as graph nodes, and aggregated through (cross-graph) node embedding component. The networks predict a permutation for node-to-node correspondence from raw pixel inputs.

3.1. Feature Extraction

We adopt a CNN for keypoints feature extraction, which are constructed by interpolating on CNN's feature map. For image I_s, the extracted feature on the keypoint P_{si} is:

$$\mathbf{h}_{si}^{(0)} = \text{Interp}(P_{si}, \text{CNN}(I_s)) \tag{4}$$

where $\text{Interp}(P, X)$ interpolates on point P from tensor X via bilinear interpolation. $\text{CNN}(I)$ performs CNN on image I and outputs a feature tensor. Taking the idea of Siamese Network [3], two input images share the same CNN structure and weights. To fuse both local structure and global semantic information, feature vectors from different layers of CNN are extracted. We choose VGG16 pretrained with ImageNet [8] as the CNN embodiment in line with [45].

3.2. Intra-graph Node Embedding

It has been shown that methods exploiting graph structure can produce robust matching [42], compared to point based methods [12, 47]. In PIA-GM, graph affinity is constructed by a multi-layer embedding component which models the higher-order information. The message passing scheme is inspired by GCN [17], where features are effectively aggregated from adjacency nodes, and the node itself:

$$\mathbf{m}_{si}^{(k)} = \frac{1}{|(i,j) \in \mathcal{E}_s|} \sum_{j:(i,j)\in\mathcal{E}_s} f_{msg}(\mathbf{h}_{sj}^{(k-1)}) \tag{5}$$

$$\mathbf{n}_{si}^{(k)} = f_{node}(\mathbf{h}_{si}^{(k-1)}) \tag{6}$$

$$\mathbf{h}_{si}^{(k)} = f_{update}(\mathbf{m}_{si}^{(k)}, \mathbf{n}_{si}^{(k)}) \tag{7}$$

Eq. (5) is the message passing along edges and f_{msg} is the message passing function. The aggregated features from adjacent nodes are normalized by the total number of adjacent nodes, as a common practice in GCN, in order to avoid the bias due to the different numbers of neighbors owned by different nodes. Eq. (6) is the message passing function for each node and it contains a node's self-passing function f_{node}. With f_{update}, Eq. (7) accumulates information to update the state of node i, and $f_{msg}, f_{node}, f_{update}$ may take any differentiable mapping from vector to vector. Here we implement f_{msg}, f_{node} as neural networks with ReLU activation, and f_{update} is a summation function. We denote Eq. (7) as graph convolution (GConv) between layer $k-1$ and k:

$$\{\mathbf{h}_{si}^{(k)}\} = \text{GConv}(\mathbf{A}_s, \{\mathbf{h}_{si}^{(k-1)}\}), \quad i \in \mathcal{V}_s \tag{8}$$

which denotes a layer of our node embedding net. Message passing paths are encoded by adjacency matrix $\mathbf{A}_s \in \{0,1\}^{N\times N}$. Note that $\mathbf{h}_i^{(0)}$ is the CNN feature of node i.

Algorithm 1: Cross-graph node embedding

Input: $(k-1)$-th layer features $\{\mathbf{h}_{1i}^{(k-1)}, \mathbf{h}_{2j}^{(k-1)}\}_{i\in\mathcal{V}_1, j\in\mathcal{V}_2}$

// similarity prediction Eq. (13, 16)

build $\hat{\mathbf{M}}$ from $\{\mathbf{h}_{1i}^{(k-1)}, \mathbf{h}_{2j}^{(k-1)}\}$ by Eq. (13);

$\hat{\mathbf{S}} \leftarrow \text{Sinkhorn}(\hat{\mathbf{M}})$;

// cross-graph aggregation Eq. (9, 10, 11)

$\{\mathbf{h}_{1i}^{(k)}\} \leftarrow \text{CrossConv}(\hat{\mathbf{S}}, \{\mathbf{h}_{1i}^{(k-1)}\}_{i\in\mathcal{V}_1}, \{\mathbf{h}_{2j}^{(k-1)}\}_{j\in\mathcal{V}_2})$;

$\{\mathbf{h}_{2j}^{(k)}\} \leftarrow \text{CrossConv}(\hat{\mathbf{S}}^\top, \{\mathbf{h}_{2j}^{(k-1)}\}_{j\in\mathcal{V}_2}, \{\mathbf{h}_{1i}^{(k-1)}\}_{i\in\mathcal{V}_1})$;

Output: k-th layer features $\{\mathbf{h}_{1i}^{(k)}, \mathbf{h}_{2j}^{(k)}\}_{i\in\mathcal{V}_1, j\in\mathcal{V}_2}$

3.3. Cross-graph Node Embedding

We explore improvement over intra-graph embedding by a cross-graph aggregation step, whereby features are aggregated from nodes with similar features in the other graph. First, we utilize graph affinity features from shallower embedding layers to predict a doubly-stochastic similarity matrix (see details in Sec. 3.5). The predicted similarity matrix $\hat{\mathbf{S}}$ encodes the similarity among nodes of two graphs. The message passing scheme is similar to intra-graph convolution in Eq. (8), with adjacency matrix replaced by $\hat{\mathbf{S}}$, and features are aggregated from the other graph. In our experiments, we will show this simple scheme works more effectively than a more complex iterative procedure.

$$\mathbf{m}_{1i}^{(k)} = \sum_{j\in\mathcal{V}_2} \hat{\mathbf{S}}_{i,j} f_{\textit{msg-cross}}(\mathbf{h}_{2j}^{(k-1)}) \tag{9}$$

$$\mathbf{n}_{1i}^{(k)} = f_{\textit{node-cross}}(\mathbf{h}_{1i}^{(k-1)}) \tag{10}$$

$$\mathbf{h}_{1i}^{(k)} = f_{\textit{update-cross}}(\mathbf{m}_{1i}^{(k)}, \mathbf{n}_{1i}^{(k)}) \tag{11}$$

where $f_{\textit{msg-cross}}, f_{\textit{node-cross}}$ are taken as identity mapping, $f_{\textit{update-cross}}$ is a concatenation of two input feature tensors, followed by a fully-connected layer. For pair of graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, the cross-graph aggregation scheme is summarized by CrossConv$(\cdot)$ in Alg. 1, where $\hat{\mathbf{S}}$ denotes the predicted correspondence from $\mathcal{G}_2$ to $\mathcal{G}_1$ and $\hat{\mathbf{S}}^\top$ denotes such relation from $\mathcal{G}_1$ to $\mathcal{G}_2$.

3.4. Affinity Metric Learning

By using the above embedding model, the structure affinity between two graphs have been encoded into the node-to-node affinity in the embedding space. As such, it allows for reducing the traditional second-order affinity matrix $\mathbf{K}$ in Eq. (1) into a linear one. Let $\mathbf{h}_{1i}$ be feature i from first graph, $\mathbf{h}_{2j}$ be feature j from the other graph:

$$\mathbf{M}_{i,j}^{(0)} = f_{\textit{aff}}(\mathbf{h}_{1i}, \mathbf{h}_{2j}), \quad i \in \mathcal{V}_1, j \in \mathcal{V}_2 \tag{12}$$

The affinity matrix $\mathbf{M}^{(0)} \in \mathbb{R}^{+N\times N}$ contains the affinity score between two graphs. $\mathbf{M}_{i,j}^{(0)}$ means the similarity between node i in the first graph and node j in the second, considering the higher-order information in graphs.

One can set $f_{\textit{aff}}$ a bi-linear mapping followed by an exponential function which ensures all elements are positive[1].

$$\mathbf{M}_{i,j}^{(0)} = \exp\left(\frac{\mathbf{h}_{1i}^\top \mathbf{A}\mathbf{h}_{2j}}{\tau}\right) \tag{13}$$

Consider the feature vectors have m dimensions, i.e. $\forall i \in \mathcal{V}_1, j \in \mathcal{V}_2, \mathbf{h}_{1i}, \mathbf{h}_{2j} \in \mathbb{R}^{m\times 1}$. $\mathbf{A} \in \mathbb{R}^{m\times m}$ contains learnable weights of this affinity function. τ is a hyper parameter for numerical concerns. For $\tau > 0$, with $\tau \to 0^+$, Eq. (13) becomes more discriminative.

3.5. Sinkhorn Layer for Linear Assignment

Given the linear assignment affinity matrix in Eq. (13), we adopt Sinkhorn for the linear assignment task. Sinkhorn operation takes any non-negative square matrix and outputs a doubly-stochastic matrix, which is a relaxation of the permutation matrix. This technique has been shown effective for network based permutation prediction [1, 33]. For $\mathbf{M}^{(k-1)} \in \mathbb{R}^{+N\times N}$, the Sinkhorn operator is

$$\mathbf{M}^{(k)\prime} = \mathbf{M}^{(k-1)} \oslash (\mathbf{M}^{(k-1)}\mathbf{1}\mathbf{1}^\top) \tag{14}$$

$$\mathbf{M}^{(k)} = \mathbf{M}^{(k)\prime} \oslash (\mathbf{1}\mathbf{1}^\top\mathbf{M}^{(k)\prime}) \tag{15}$$

[1]We have also tried other more flexible fully-connected layers, while we find the exponential function is simple and more stable for learning.

$\oslash$ means element-wise division, and $\mathbf{1} \in \mathbb{R}^{N\times 1}$ is a column vector whose elements are all ones. Sinkhorn algorithm works iteratively by taking row-normalization of Eq. (14) and column-normalization of Eq. (15) alternatively.

By iterating Eq. (14, 15) until convergence, we get a doubly-stochastic matrix. This doubly-stochastic matrix $\mathbf{S}$ is treated as our model's prediction in training.

$$\mathbf{S} = \text{Sinkhorn}(\mathbf{M}^{(0)}) \tag{16}$$

For testing, Hungarian algorithm [19] is performed on $\mathbf{S}$ as a post processing step to discretize output into a permutation matrix. Sinkhorn operation is fully differentiable because only matrix multiplication and element-wise division are taken. It can be efficiently implemented with the help of PyTorch's automatic differentiation feature [35].

3.6. Permutation Cross-Entropy Loss

Our methods directly utilize ground truth node-to-node correspondence, i.e. permutation matrix, as the supervised information for end-to-end training. Since Sinkhorn layer in Eq. (16) is capable to transform any non-negative matrix into doubly-stochastic matrix, we propose a linear assignment based permutation loss that evaluates the difference between predicted doubly-stochastic matrix and ground truth permutation matrix for training.

Cross entropy loss is adopted to train our model end-to-end. We take the ground truth permutation matrix $\mathbf{S}^{gt}$, and compute the cross entropy loss between $\mathbf{S}$ and $\mathbf{S}^{gt}$. It is denoted as permutation loss, and this is the main method adopted to train our deep graph matching model $L_{\textit{perm}}$:

$$-\sum_{i\in\mathcal{V}_1, j\in\mathcal{V}_2} \left(\mathbf{S}_{i,j}^{gt} \log \mathbf{S}_{i,j} + (1-\mathbf{S}_{i,j}^{gt}) \log(1-\mathbf{S}_{i,j})\right) \tag{17}$$

Note the competing method GMN [45] applies a pixel offset based loss namely "displacement loss". Specifically it computes an offset vector $\mathbf{d}$ by a weighted sum of all matching candidates. The loss is given as the difference between predicted location and ground truth location.

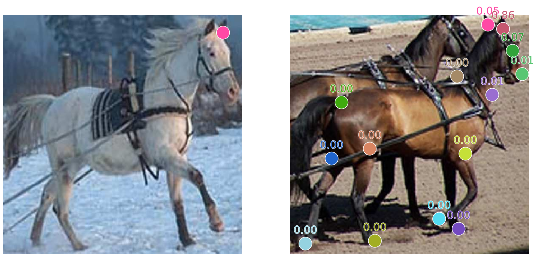
$$\mathbf{d}_i = \sum_{j\in V_2} (\mathbf{S}_{i,j} P_{2j}) - P_{1i} \tag{18}$$

$$L_{\textit{off}} = \sum_{i\in V_1} \sqrt{||\mathbf{d}_i - \mathbf{d}_i^{gt}||^2 + \epsilon} \tag{19}$$

where $\{P_{1i}\}, \{P_{2j}\}$ are the coordinates of keypoints in first and second image, respectively. While ϵ is a small value ensuring numerical robustness. In comparison, our cross entropy loss can directly learn a linear assignment cost based permutation loss in an end-to-end fashion.

3.7. Further Discussion

Pairwise affinity matrix vs. embedding. Existing graph matching methods focus on modeling second-order [7, 22]



$L_{perm} = 5.139, \quad L_{off} = 0.070$

Figure 2. Failure case of offset loss: source image (left) and target image (right) with matching candidates, where numbers denote the probability of predicted matching. Ground truth matching nodes are colored in rose (only receives 0.05 probability by this poor prediction). Offset loss is computed by a weighted sum among all candidates, resulting in a misleading low loss 0.070. In this case offset loss fails to provide supervision on distinguishing left/right ears. Our permutation loss, on the contrary, issues a reasonably high loss 5.139.

and higher-order [21, 43] feature with an explicitly predefined affinity matrix or tensor. The affinity information can be encoded in an $N^2 \times N^2$ affinity matrix. Optimization techniques are applied to maximize graph affinity.

In contrast, we resort to the node embedding technique with two merits. First, the space complexity can be reduced to $N \times N$. Second, the pairwise affinity matrix $\mathbf{K}$ in Eq. (1) can only encode the edge information, while the embedding model can implicitly encode higher-order information.

Sinkhorn net vs. spectral matching. GMN [45] adopts spectral matching (SM) [22] which is differentiable for back propagation. While we adopt the Sinkhorn net instead. In fact, the input of Sinkhorn is of complexity $O(N^2)$ while it is $O(N^4)$ for spectral matching. However, in SM we observe more iterations to convergence. Such iteration may bring negative effect to gradient's back propagation. In fact, spectral matching is for graph matching while Skinhorn net is for linear assignment, which is relaxed from the graph matching task by our embedding component.

Pixel offset loss vs. permutation loss. The loss function adopted by GMN [45] is an offset loss named "displacement loss". The loss takes the weighted sum of all candidate points, and compute the offset vector from the original image to the source image. In training, GMN tries to minimize the variance between predicted offset vector and ground truth offset vector. In comparison, with the help of Sinkhorn net, we adopt a combinatorial permutation loss which is computed as the cross entropy between predicted result and ground truth permutation. Such permutation loss takes the ground truth permutation directly as supervision, and utilize such information for end-to-end training.

Fig. 2 gives an example for the failure case of offset loss. In this case, the offset loss is unreasonably low, but the permutation loss provides correct information. Experiments also show that models trained with our permutation loss exceed offset loss models in matching accuracy.

4. Experiments

4.1. Metrics and Peer Methods

We evaluate the matching accuracy between two given graphs. In the evaluation period, two graphs are given with same number of nodes N. Each node in one graph is labeled to another node in the other graph. The model predicts a correspondence between two graphs. Such correspondence is represented by a permutation matrix.

The matching accuracy is computed from the permutation matrix, by the number of correctly matched keypoint pairs averaged by the total number of keypoint pairs. For a predicted permutation matrix $\mathbf{S}^{pred} \in \{0,1\}^{N\times N}$ and a ground truth permutation $\mathbf{S}^{gt} \in \{0,1\}^{N\times N}$, matching accuracy is computed by

$$\mathrm{acc} = \sum \mathrm{AND}(\mathbf{S}^{pred}_{i,j}, \mathbf{S}^{gt}_{i,j})/N \tag{20}$$

where AND is the logical function.

The evaluation involves the following peer methods:

GMN. Graph Matching Network (GMN) is the seminal model proposed in [45]. GMN adopts VGG16 [36] network to extract image features. First-order and second-order features are extracted from shallower layer (relu4_2) and deeper layer (relu5_1) of VGG16, respectively. GMN models graph matching affinity via an unlearnable graph matching solver namely spectral matching (SM) [22]. This model is class-agnostic, meaning it learns an universal model for all instance classes. Two graphs are constructed by Delaunay triangulation and fully-connected topology, respectively. GMN is the first end-to-end deep learning method for graph matching. Note the major difference is that the loss function is an offset based loss by Eq. (19). We follow [45] and re-implement GMN with PyTorch as the source code is not publicly available.

HARG-SSVM. This is the structured SVM based learning graph matching method [6], as a baseline of learning graph matching without deep learning. HARG-SSVM is a class-specific method, where graph models are learned for each class. We use the source code released by the authors upon their approval. The original setting in [6] assumes that the keypoints of the object to be matched is unknown, and the keypoint candidates are proposed by Hessian detector [27]. In our setting, however, all candidate keypoints are known to the model. Therefore, we slightly modify the original code. From all candidate points found by the Hessian detector, we assign the nearest neighbor from ground truth point as matching candidate. This practice is originally taken in the training process of HARG-SSVM. Graphs are created with hand-crafted edge features named HARG.

PIA-GM/PCA-GM. Our methods adopt VGG16 [36] as backbone CNN, and extract features from relu4_2 and

Table 2. Accuracy (%) on Pascal VOC Keypoint. Note after replacing the offset loss by permutation loss, GMN-PL outperforms GMN [45] almost in all categories. While our method PIA-GM's performance degenerates when its permutation loss is changed to offset loss.

method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
GMN	31.9	47.2	51.9	40.8	68.7	72.2	53.6	52.8	34.6	48.6	**72.3**	47.7	54.8	51.0	38.6	75.1	49.5	45.0	83.0	86.3	55.3
GMN-PL	31.1	46.2	58.2	45.9	70.6	76.4	61.2	61.7	**35.5**	53.7	58.9	57.5	56.9	49.3	34.1	77.5	57.1	53.6	83.2	88.6	57.9
PIA-GM-OL	39.7	**57.7**	58.6	47.2	74.0	74.5	62.1	66.6	33.6	61.7	65.4	58.0	67.1	58.9	41.9	77.7	64.7	50.5	81.8	89.9	61.6
PIA-GM	**41.5**	55.8	60.9	**51.9**	75.0	75.8	59.6	65.2	33.3	**65.9**	62.8	**62.7**	67.7	62.1	42.9	**80.2**	64.3	**59.5**	82.7	90.1	63.0
PCA-GM	40.9	55.0	**65.8**	47.9	**76.9**	**77.9**	**63.5**	**67.4**	33.7	65.5	63.6	61.3	**68.9**	**62.8**	**44.9**	77.5	**67.4**	57.5	**86.7**	**90.9**	**63.8**

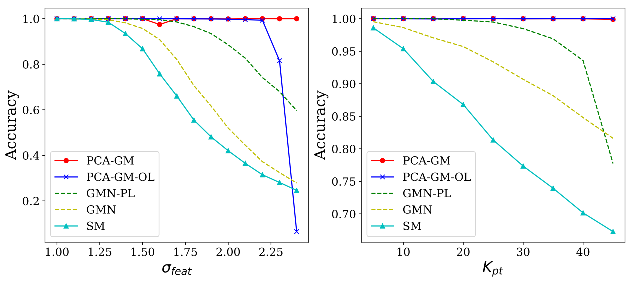


Figure 3. Synthetic test with noise on feature vectors of node and edge, and keypoint numbers on methods with different affinity models and losses. Default: $K_{pt} = 20, \sigma_{feat} = 1.5, \sigma_{coo} = 10$.

Table 3. Accuracy (%) on Willow ObjectClass. GMN-VOC means model trained on Pascal VOC Keypoint, likewise for Willow.

method	face	m-bike	car	duck	w-bottle
HARG-SSVM [6]	91.2	44.4	58.4	55.2	66.6
GMN-VOC [45]	98.1	65.0	72.9	74.3	70.5
GMN-Willow [45]	99.3	71.4	74.3	82.8	76.7
PCA-GM-VOC	**100.0**	69.8	78.6	82.4	95.1
PCA-GM-Willow	**100.0**	**76.7**	**84.0**	**93.5**	**96.9**

relu5_1 for fair comparison with [45]. These two feature vectors are concatenated to fusion both local and global features. In PIA-GM, affinity is modeled by a 3 intra-embedding layers while in PCA-GM it is a stack of 1 intra layer, 1 cross layer and 1 intra layer, both followed by affinity mapping in Eq. (13). Each GNN layer has a feature dimension of 2048. Permutation loss in Eq. (17) is used. Input graphs are both constructed by Delaunay triangulation and we empirically set $\tau = 0.005$ in Eq. (13). Our models are implemented by PyTorch.

GMN-PL & PIA/PCA-GM-OL. GMN-PL and PIA/PCA-GM-OL are variants from GMN [45] and our proposed PIA/PCA-GM, respectively. GMN-PL changes the offset loss in GMN to permutation loss, with all other configurations unchanged. While PIA/PCA-GM-OL switch the permutation loss to offset loss, leaving all other components unchanged.

For natural image experiments, we draw two images from the dataset, and build two graphs containing the same number of nodes. The graph structure is agnostic, and is constructed according to methods' configurations (see discussions above). The CNN weight is initialized by a pre-trained model on ImageNet [8] classification dataset.

4.2. Synthetic Graphs

Evaluation is first performed on synthetic graphs generated in line with the protocol in [7]. Ground truth graphs are generated with a given number of keypoints K_{pt}, each with a 1024-dimensional (512 for nodes and 512 for edges) random feature in $\mathcal{U}(-1, 1)$ (simulating CNN features), and a random 2d-coordinate in $\mathcal{U}(0, 256)$. During training and testing, we draw disturbed graphs, with Gaussian noise $\mathcal{N}(0, \sigma_{feat}^2)$ added to features, and keypoint coordinates blurred by a random affine transform, plus another random noise of $\mathcal{N}(0, \sigma_{coo}^2)$. Note that there is no CNN feature extractor adopted, only graph modeling approaches and loss metrics are compared. The matching accuracy of PCA-GM, PCA-GM-OL, GMN-PL, GMN and unlearning SM is evaluated with respect to K_{pt} and σ_{feat}. For each trial, 10 different graphs are generated and accuracy is averaged. Experimental results in Fig. 3 show the robustness of PCA-GM against feature deformation and complicated graph structure.

4.3. Pascal VOC Keypoints

We perform experiments on Pascal VOC dataset [11] with Berkeley annotations of keypoints [2]. It contains 20 classes of instances with labeled keypoint locations. Following the practice of peer methods [45], the original dataset is filtered to 7,020 annotated images for training and 1,682 for testing. All instances are cropped around its bounding box and resized to 256×256, before passed to the network. Pascal VOC Keypoint is a difficult dataset because instances may vary from its scale, pose and illumination, and the number of inliers ranges from 6 to 23.

We test on Pascal VOC Keypoint [2] and evaluate on 20 Pascal categories. We compare GMN, GMN-PL, PIA-GM-OL, PIA-GM, PCA-GM and give detailed experimental results in Tab. 2. Our proposed models PIA-GM-OL, PIA-GM, PCA-GM outperform in most categories, including the mean accuracy over 20 categories. Our implementation of PCA-GM runs at ~ 18 pairs per second in training, on dual RTX2080Ti GPUs. The result shows the superiority of the linear assignment loss over offset loss in training, embedding and Sinkhorn over fixed SM [22] in affinity modeling, and cross-graph embedding over intra-graph embedding.

4.4. Willow ObjectClass

Willow ObjectClass dataset is collected by [6] for real images. This dataset consists of 5 categories from Caltech-

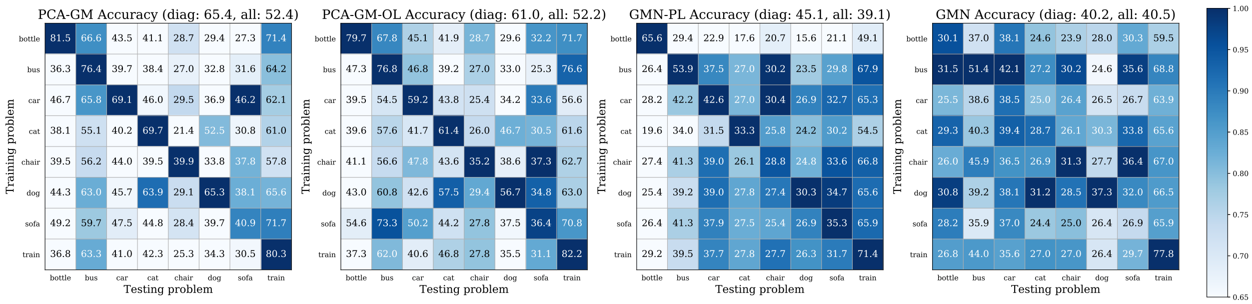


Figure 4. Confusion matrix of eight categories of objects from Pascal VOC Keypoint. Models are trained on categories on the y-axis, and testing results are given on categories on the x-axis. Note that accuracy does not degenerate much for PCA-GM between similar categories (such as cat and dog). Numbers in matrices are the corresponding accuracy and the color map stands for accuracy normalized by the highest accuracy on this category in current matrix. Note that the color filled in cells does not denote the absolute value of accuracy among different categories and matrices. Accuracy for elements in diagonal and overall for each confusion matrix are shown in bracket on the top of each matrix. We follow the train/test split provided by the benchmark for each category.

Table 4. Ablation study on proposed components on Pascal VOC Keypoint. Tick denotes the learning is activated for the column. For VGG16 feature it means it is fine-tuned using the graph matching training data, otherwise the pretrained VGG16 via ImageNet.

VGG16 feature	intra-graph embedding	cross-graph embedding	affinity metric	accuracy
✓	✓	✓	✓	**63.8**
✓	✓	✓	×	63.6
✓	✓	×	×	62.1
✓	×	×	×	54.8
×	×	×	×	41.9

256 (face, duck and wine bottle) and Pascal VOC 2007 (car and motorbike), each with at least 40 images. Images are resized to 256 × 256 if it is passed to CNN. This dataset is considered easier than Pascal VOC Keypoint, because all images inside the same category are aligned in their pose, and it lacks scale, background and illumination changes.

We follow the protocol built by authors of [6] for fair evaluation. HARG-SSVM is trained and evaluated on this willow dataset. For other competing methods, we initialize their weights on Pascal VOC Keypoint dataset, with all VOC 2007 car and motorbike images removed. They are denoted as GMN-VOC and PCA-GM-VOC. They are later finetuned on the willow dataset as GMN-Willow and PCA-GM-Willow, reaching even higher result in evaluation. Note that HARG-SSVM is a class-specific model, but GMN and PCA-GM are both class-agnostic. Tab. 3 shows our proposed PCA-GM almost surpasses all competing methods in all categories of Willow Object Calss dataset.

4.5. Further Study

PCA-GM components. Ablation study with different PCA-GM components trained/untrained is reported in Tab. 4. It shows the usefulness of all our components. VGG16 is initialized with pretrained weights on ImageNet, embedding layers are randomly initialized, and the weight of affinity metric is initialized by identity matrix plus random noise.

Table 5. Accuracy (%) by number of iterations for a more complex cross-graph affinity component design on Pascal VOC Keypoint, which has negative effect on accuracy (PIA-GM achieves 63.0%).

# of iters	1	2	3	4	5	6	7	Alg. 1
PCA-GM accuracy	63.1	61.3	60.9	54.7	45.9	46.7	46.2	**63.8**

Cross-graph component design. Our cross-graph affinity component is relatively simple. In fact we also explore a more complex design of cross-graph module, where the matrix $\hat{\mathbf{S}}$ is updated by iterative prediction, rather than predicted from shallower embedding layer as PCA-GM in Alg. 1. In this alternative design, $\hat{\mathbf{S}}^{(0)}$ is initialized as zero matrix, and we iteratively predict $\hat{\mathbf{S}}^{(k)}$ from $\hat{\mathbf{S}}^{(k-1)}$, which is passed to the cross-graph component. Result in Tab. 5 reveals that PCA-GM's performance will degrade as $\hat{\mathbf{S}}$ is iteratively predicted, and we further find the training is not stable by this iterative design hence we stick to the simple design in Alg. 1. Details on this alternative design is given in supplementary materials.

Confusion matrix. To testify the generalization behavior of our model, we train PCA-GM, PCA-GM-OL, GMN-PL, GMN on eight categories in Pascal VOC Keypoint and report testing result on each category as shown in Fig. 4, where result is plotted via confusion matrix (y-axis for training and x-axis for testing). It shows that embedding adopted in PCA-GM works well, and the permutation loss offers better supervision than the offset one.

5. Conclusion

This paper has presented a novel deep learning framework for graph matching, which parameterizes the graph affinity with deep networks and the learning objective involves a permutation loss to account for the arbitrary transformation between two graphs. Extensive experimental results including an ablation study on the presented components and the comparison with peer methods show the state-of-the-art performance of our method.

References

[1] Ryan Prescott Adams and Richard S Zemel. Ranking via sinkhorn propagation. *arXiv:1106.1925*, 2011.

[2] Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, 2009.

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *NIPS*, 1994.

[4] Tiberio S. Caetano, Julian J. McAuley, Li Cheng, Quoc V. Le, and Alex J. Smola. Learning graph matching. *TPAMI*, 2009.

[5] Michael Chertok and Yosi Keller. Efficient high order matching. *TPAMI*, 2010.

[6] Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *ICCV*, 2013.

[7] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *ECCV*, 2010.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[9] Olivier Duchenne, Francis Bach, Kweon In-So, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *PAMI*, 2011.

[10] Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. *arXiv:1805.07010*, 2018.

[11] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010.

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 1981.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. 1990.

[14] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *TPAMI*, 1996.

[15] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, 2016.

[16] Jiayi Huang, Mostofa Patwary, and Gregory Diamos. Coloring big graphs with alphagozero. *arXiv:1902.10162*, 2019.

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[18] Woute Kool and Max Welling. Attention solves your tsp. *arXiv:1803.08475*, 2018.

[19] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955.

[20] Eugene L. Lawler. The quadratic assignment problem. *Management Science*, 1963.

[21] Jungmin Lee Lee, Minsu Cho, and Kyoung Mu Lee. Hypergraph matching via reweighted randomwalks. In *CVPR*, 2011.

[22] Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, 2005.

[23] Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *IJCV*, 2012.

[24] Marius Leordeanu, Andrei Zanfir, and Cristian Sminchisescu. Semi-supervised learning and optimization for hypergraph matching. In *ICCV*, 2011.

[25] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *EJOR*, 2007.

[26] Gonzalo Mena, David Belanger, Gonzalo Muñoz, and Jasper Snoek. Sinkhorn networks: Using optimal transport techniques to learn permutations. *NIPS Workshop in Optimal Transport and Machine Learning*, 2017.

[27] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *IJCV*, 2004.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Computer Science*, 2013.

[29] Quynh Nguyen Ngoc, Antoine Gautier, and Matthias Hein. A flexible tensor block coordinate ascent scheme for hypergraph matching. In *CVPR*, 2015.

[30] Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. Revised note on learning quadratic assignment with graph neural networks. In *DSW*, 2018.

[31] Giorgio Patrini, Marcello Carioni, Patrick Forre, Samarth Bhargav, Max Welling, Rianne van den Berg, Tim Genewein, and Frank Nielsen. Sinkhorn autoencoders. *arXiv:1810.01118*, 2018.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, 2014.

[33] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. Visual permutation learning. *TPAMI*, 2018.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *TNN*, 2009.

[35] Francesc Serratosa, Albert Solé-Ribalta, and Xavier Cortés. Automatic learning of edit costs based on interactive and adaptive graph recognition. In *GbR*. 2011.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014.

[37] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *AoMS*, 1964.

[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[39] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *ECCV*, 2008.

[40] Mario Vento and Pasquale Foggia. Graph matching techniques for computer vision. *Graph-Based Methods in Computer Vision: Developments and Applications*, 2012.

[41] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *SIGKDD*, 2016.

[42] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *ICMR*, 2016.

[43] Junchi Yan, Chao Zhang, Hongyuan Zha, Wei Liu, Xiaokang Yang, and Stephen M. Chu. Discrete hyper-graph matching. In *CVPR*, 2015.

[44] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *NIPS*, 2017.

[45] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *CVPR*, 2018.

[46] Ron Zass and Amnon Shashua. Probabilistic graph and hypergraph matching. In *CVPR*, 2008.

[47] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *IJCV*, 1994.

[48] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv:1812.08434*, 2018.