Graph Learning for Graph Based Semantic Matching

Yashas Annadani, Daniil Emtsev, Nikita Janakarajan, Nino Scherrer

Abstract

We investigate the case of semantic matching in images containing objects wherein the keypoints of the objects are provided. We are interested in matching by constructing a graph of these keypoints and matching them across images to establish a correspondence. We explore the possibilities of using a deep learning framework to learn the graph of these keypoints and simultaneously perform matching in an end-to-end learning framework. We use a variant of the parameterisation of the adjacency matrix as recently proposed in [7] and perform a study on the kind of graph it learns and how beneficial it is to the matching task. We also perform baseline ablation studies with respect to different graph construction techniques like Delaunay Triangulation and k-NN across two different standard benchmark datasets. Our experiments indicate that the variant of the parameterisation of the learnable adjacency matrix in which we have experimented with in this project leads to extremely sparse graphs when combined with a matching process as that of [25].

1. Introduction

Matching graphs is a fundamental problem in combinatorial optimization. Applications of graph matching are aplenty [3]. Given the numerous applications graph matching has in various domains, it has garnered significant interest from the research community [27]. In this report, we consider graph matching technique with particular interest to computer vision problems that involve a deep learning framework. In particular, we are interested in whether we can learn graphs which are optimal to the task at hand in an end-to-end manner by parameterising the adjacency matrix as a learnable component in the graph matching process.

As convolutional neural networks trained on Imagenet [5] or other big datasets have shown good transfer learning capability, it is of significant interest whether a graph matching procedure can be incorporated in a convolutional neural network framework in an end-to-end manner. While graph matching is a Quadratic Assignment Problem (QAP) and is known to be NP-Hard, approximate solutions are obtained by relaxing the constraints of the optimization problem [22, 27]. More recently, Zanfir *et al.* [25] proposed an end-to-end deep *learning* framework wherein the entire graph matching optimization was included within the learning formulation. While such an end-to-end learning was shown to be beneficial for improving the performance on a problem-specific setting, the graph structure had to be specified manually for each image every time. The authors simplified this procedure by using a Delanuay Triangulation [10] on the graph nodes. This fixed graph structure may not be optimal for the graph matching task and has relevantly been identified in [7]. Therefore, [7] proposed a graph learning network and combined that procedure with a graph convolutional network [9]. Though the authors demonstrated marginal improvements with respect to the graph matching task, it was performed on only one dataset. In our work, we indicate some of the drawbacks of using this procedure and why the parameterisation proposed in [7] is not entirely consistent with the matching process of Zanfir *et al.* [25]. Therefore, we consider a variant of this parameterisation and perform experiments on it. We combine this graph learning procedure with the matching process as introduced in [25].

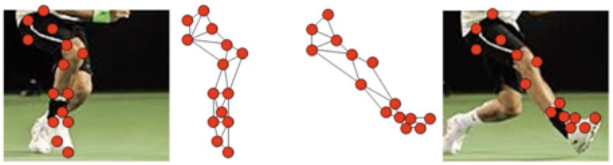

Figure 1: One of the many examples where graph matching is applicable. Graph matching could be used for matching object parts across pose, as in this illustration. Figure from [26].

Our contributions in this work are as follows:

- After the proposal, we observed that the graph learning proposed in [7] is incompatible with the matching framework of Zanfir *et al.* [25]. We discuss the limitations which arise when using the graph learning formulation of [7] to combine with the matching

framework of [25]. Therefore, we introduce a variant which shares same spirit as the one proposed in [7] that is more relevant to the matching framework we are considering.

- Since the learning and matching is done end-to-end, we observed that graph learning network sometimes has a tendency to give an empty edge set a consequence of which the contribution from pairwise potentials to the loss would be zero. Therefore, we also experimented with alternative formulations.

- Comparisons with baselines like kNN and Delaunay Triangulation for graph structure are missing in [7]. We perform experiment on these baselines on two standard benchmark datasets.

- The existing work demonstrates only the final performance improvement when including graph learning on only one dataset, i.e PascalVOC [6]. We have also done experiments of baselines and graph learning process on another dataset, the CUB [19] birds dataset.

The remainder of the report is organized as follows: We present some of the related work in section 2. Section 3 presents the implemented algorithm in detail. Section 5 describes the experiments performed in detail and finally concluding thoughts are presented in Section 6.

2. Related Work

Graph Learning: Graph convolutional networks (GCNs) evolved from the success of CNNs in the computer vision domain. The idea behind GCNs is to use the features of a node and its neighbours to extract a high level embedding of that node and consequently learn a relationship between these nodes. [9] propose a model for graph representation and semi-supervised learning that explores first order approximations of spectral filters. A symmetric graph convolutional autoencoder model for unsupervised graph representation learning by [16] uses a novel Laplacian sharpening convolution operation. [18] uses attention models for graph based semi-supervised learning. [13] introduce adaptive graph CNNs that use a metric learning method. A single network integrating both the convolutional representation and graph learning parts called Graph Learning-convolutional Network (GLCN) by [8] has been used for semi-supervised graph node classification. While the above methods involve learning graphs in general, they do not combine it with a matching framework, as done in [7].

Graph Matching: Graph matching is an extensively studied problem given its wide range of applications across different domains [21, 15, 23, 24]. We refer the reader to [22] for an overview. Deep learning based graph matching was introduced in [25] wherein the graph optimization was incorporated as part of the learning process by introducing power iteration, bi stochastic and voting layers and backpropagating the loss through the entire network. Following this work, Li *et al.* [14] and Wang *et al.* [20] proposed an embedding based method wherein the graphs were embedded to a metric space using graph convolutional networks [9] and then learnt a similarity metric between these embeddings in an end-to-end manner. For all the above methods, the graph structure had to be fixed or specified manually. Learning the parameters of the graph affinity matrix has been investigated by [1, 11, 27]. Caetano *et al.* [1] demonstrate that compatibility functions for the graph matching process can be estimated from labeled examples by casting it as a supervised learning problem. Leordeanu *et al.* [11] learn the parameters of the graph matching procedure in an unsupervised manner when the correspondences between graphs are not available. This is a comparatively harder problem, but the authors demonstrate that it can lead to good results under certain assumptions and conditions of the graph affinity matrix. [12] takes a semi-supervised approach and proposed optimization methods for hypergraph matching. In all the above methods, the number of parameters is low, often controlling geometric affinities between pairs of points. [4] propose a spectral relaxation technique for approximate solutions to matching problems that incorporates one-to-one or one-to-many constraints within the relaxation scheme.

[7] proposed to incorporate the graph learning component within the entire learning framework of [20], on which our work is based.

3. Method

Figure 2 presents an illustration of the overall method.

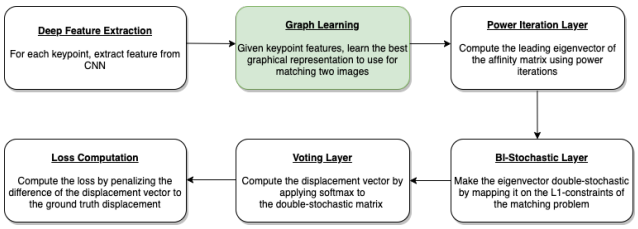

Figure 2: Flow diagram of the different stages of the graph matching optimisation. Entire framework can be trained end-to-end. More details about the network in [25]. Here, we incorporate graph learning to this formulation which is proposed in [7] and perform ablation studies and investigate different formulations based on it.

Before delving into details of the graph learning

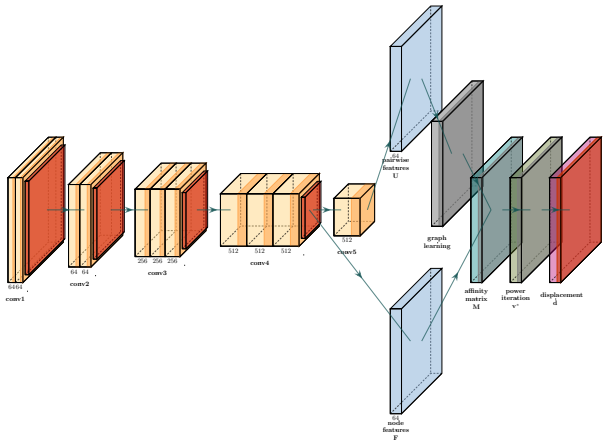

Figure 3: Illustration of the network used for graph matching process.

procedure, we give a brief overview of the graph matching procedure which we are using in this work in order to combine it with the graph learning. This graph matching procedure is based on [25].

3.1. Graph Matching

Given two input graphs $(\mathcal{G}^1, \mathcal{G}^2)$, we aim to obtain a one-to-one matching between the nodes of $\mathbf{V}^1$ and $\mathbf{V}^2$. Let us denote $\mathbf{v} \in \{0,1\}^{nm\times 1}$ as an assignment vector, where:

$$\mathbf{v}_{ia} = \begin{cases} 1 & \text{if } i \in \mathbf{V}^1 \text{ is matched to } a \in \mathbf{V}^2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

with the one-to-one mapping constraints:

$$\forall i: \quad \sum_a \mathbf{v}_{ia} = 1 \quad \text{and} \quad \forall a: \quad \sum_i \mathbf{v}_{ia} = 1 \tag{2}$$

In most cases graph matching is formulated as an optimization problem based on a global affinity matrix $\mathbf{M} \in \mathbb{R}^{nm\times nm}$ which encodes the pair-wise similarities between nodes and edges. More specifically, the diagonal entries measure node-to-node similarity scores, and the off-diagonal entries measure edge-to-edge scores (e.g.: $\mathbf{M}_{ia,jb}$ measures the similarity between $(i,j) \in \mathbf{E}^1$ and $(a,b) \in \mathbf{E}^2$). Based on this definition, we can express the optimal assignment $\mathbf{v}^*$ as:

$$\mathbf{v}^* = \underset{v}{\operatorname{argmax}}\, \mathbf{v}^T\mathbf{M}\mathbf{v}, \text{ s.t. } \mathbf{C}\mathbf{v} = 1, \mathbf{v} \in \{0,1\}^{nm\times 1} \tag{3}$$

where $\mathbf{C} \in \mathbb{R}^{nm\times nm}$ encodes the constraints in equation 2. Since the problem is known to be NP-hard, it can be relaxed by detaching the binary assignment and one-to-one mapping constraints as in [25]. Thus we get:

$$\mathbf{v}^* = \underset{\mathbf{v}}{\operatorname{argmax}}\, \mathbf{v}^T\mathbf{M}\mathbf{v} \text{ s.t. } \|\mathbf{v}\|_2 \tag{4}$$

The above optimisation procedure can be done in an end-to-end manner by the procedure introduced in Zanfir et al. [25]. The details of this procedure is given in Appendix A. In previous works, including that of [25], the graph structure was fixed for all the images. In this work, we would like to learn the graph along with the matching process.

4. Graph Learning Framework

To perform graph matching between a pair of images $(\mathbf{I}^1, \mathbf{I}^2)$, we aim to learn the corresponding graph structure $(\mathbf{A}^1, \mathbf{A}^2)$ by parametrising the adjacency matrix as a layer in the neural network. We learn the graphs using a set of given keypoints for each image. Given the two images, we define the graph structures as:

$$\begin{aligned} \mathcal{G}^1 &= (\mathbf{V}^1, \mathbf{E}^1) \quad \text{with:} \quad |\mathbf{V}^1| = n, \ |\mathbf{E}^1| = p \\ \mathcal{G}^2 &= (\mathbf{V}^2, \mathbf{E}^2) \quad \text{with:} \quad |\mathbf{V}^2| = m, |\mathbf{E}^2| = q \end{aligned} \tag{5}$$

with $\mathbf{P}^1 \in \mathbb{R}^{n\times 2}, \mathbf{P}^2 \in \mathbb{R}^{m\times 2}$ as the corresponding coordinate matrices. The corresponding node-to-node adjacency matrices $\mathbf{A}^1 \in \{0,1\}^{n\times n}, \mathbf{A}^2 \in \{0,1\}^{m\times m}$ are learnt by the network. More formally, the adjacency matrices are defined as follows:

$$\mathbf{A}^1_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in \mathbf{E}^1 \\ 0 & \text{if } (i,j) \notin \mathbf{E}^1 \end{cases} \tag{6}$$

Similar definition holds for $\mathbf{A}^2_{i,j}$. Throughout this work, we share the parameters of the layer across both images to learn the adjacency matrices. Hence, in this section, for the sake of simplicity, we describe the graph learning procedure corresponding to one of the graphs and drop the superscripts. The adjacency matrix $\mathbf{A}$ is factorized into node-edge incidence matrices $\mathbf{G}, \mathbf{H} \in \{0,1\}^{n\times p}$ such that:

$$\mathbf{A} = \mathbf{G}\,\mathbf{H}^T \tag{7}$$

where $\mathbf{G}_{i,c} = \mathbf{H}_{j,c} = 1$ if the c^{th} edge of $\mathbf{G}$ starts from the i^{th} node and ends at the j^{th} node.

As indicated before, we start with the parameterisation of the adjacency matrix $\mathbf{A}$ proposed in [7] for the graph learning process.

$$\mathbf{A}_\theta(i,j) = \frac{\exp\left(\sigma(\theta^T[\mathbf{x}^i||\mathbf{x}^j])\right)}{\sum_{j=1}^{n}\exp\left(\sigma(\theta^T[\mathbf{x}^i||\mathbf{x}^j])\right)} \tag{8}$$

where θ are parameters of the neural network, $\mathbf{x}^i$ is the feature vector corresponding to node i, $||$ corresponds to concatenation operation, n is the number of nodes and σ is an activation function like sigmoid or ReLU.

After submitting the proposal, we realised that the above adjacency matrix parameterisation is not compatible with the matching framework as described in Section 3.1. While Equation 8 parameterises a stochastic matrix corresponding

to a directed edge set, we are actually looking for an undirected graph with a hard assignment of the edges rather than soft assignments. We particularly require hard assignments as it would make the factorisation of $\mathbf{A} = \mathbf{G}\mathbf{H}^T$ easier to incorporate. In light of this argument, we experiment with a variant of the above equation and parameterise only the upper triangular matrix of the adjacency matrix. More precisely, the adjacency matrix is now parameterised as:

$$\mathbf{A}_\theta(i,j) = \sigma(\theta^T[\mathbf{x}^i||\mathbf{x}^j]) \tag{9}$$

where $\{(i,j) : j > i > 1; i,j \in \mathbb{N}\}$, θ are parameters of the neural network, $\mathbf{x}^i$ is the feature vector corresponding to node i, $||$ corresponds to concatenation operation, and σ is an activation function like sigmoid or ReLU.

We made several observations with this parameterisation which we have further illustrated in section 5. Conceptually, we still predict $\binom{n}{2}$ which is $\mathcal{O}(n^2)$ in the number of nodes. We found this computationally very expensive even with autograd packages. In addition, we also found that this parameterisation gives rise to extremely sparse graphs especially in the case of ReLU activation function. In the initial stage of the training stage, we found that it gave empty edge sets to many keypoint sets, thus making the learning process unstable and the results very poor as compared to fixed graph structures. This is also not surprising because a sparse edge set would result in a smaller loss contribution from the pairwise potentials and there is a systematic tendency of this parameterisation to give rise to sparse or null edge sets. Note that our insights and the experimental results we demonstrate do not contradict the conclusions of [7]. Since [7] uses a graph convolutional network and directly embed the graph to a metric space, this sort of a parameterisation is definitely helpful in such a framework. But when we explicitly account for pairwise potentials in our loss formulation, such a parameterisation is not necessarily helpful, and can potentially exacerbate the results.

In light of our above observation, we experimented with additional form of a parametrisation which can be defined as follows:

$$\mathbf{A}_\theta = \sigma(\text{Conv1D}(\mathbf{X})) \tag{10}$$

Let $\theta \in \mathbb{R}^{d \times n}$ be the parameters of a 1D Convolutional Layer and σ is a ReLU function. We only take the upper half of the activations for the adjacency matrix. This parametrisation has the following advantages. Firstly, it does not involve exponential number of forward passes through the network. Secondly, the parametrisation is much simpler. However, we still observed the same issue of sparse edge sets and our efforts to improve the final results were in vain. We attribute this to the way the graph matching procedure is performed in the framework we are using, which considers pairwise potentials.

5. Experiments

We perform experiments on two standard benchmark datasets: **PascalVOC** [6] and **CUB 200-2011** [19]. These two datasets come with annotated keypoints.

The details of both these datasets along with number of images and classes used for training and evaluation is given in Appendix B.

Metric: We use Percentage of Correct Keypoints within 10 pixels (PCK @ 10) as the evaluation metric. It is defined as the number of correct flow predictions which are with a euclidean distance of 10 pixel threshold. More details can be found in [2].

5.1. Experimental Setup

We train all the models for 10 epochs. We use an SGD optimizer with a momentum of 0.9 and an initial learning rate of 2×10^{-4}. The learning rate is decayed by a factor of 0.1 after every epoch. We have implemented batch version of the Kronecker product, and the diagonalisation of the matrices along with other layers of the graph matching and learning process.

5.2. Baselines

Fully Connected Graph (FC): We have fully connected graphs for both the graphs to be matched.

Delaunay Triangulation: We perform delaunay triangulation on the vertex locations in the image. The source image is delaunay triangulated while the target image to be matched is fully connected.

kNN: We impose k nearest neighbour connectivity on the vertex locations for both the source and target images to be matched. We experiment with $k = 1, 2, 3$.

5.3. Results

The results of different baselines and graph learning technique is reported in Tables 1 and 2. For graph learning (GL), the results in the tables for both the datasets are from the parameterisation defined in Equation 10 as the results and observations from the other parameterisation in Equation 9 are quite similar. One can note that kNN triangulation with three neighbors performs the best among all the different methods on most of the classes in both the datasets. As indicated in Section 4, the graph learning parameterisation gives very sparse edge sets which results in a very low performance. More informative edges are always useful for these two datasets, as indicated by the best performance of 3 nearest neighbours method. As the number of nearest neighbours are increased, the performance also seems to increase, indicating that having more informative edges is beneficial and a sparse one is not necessarily helpful. On the other hand, using a fully connected graph structure does not give the best

Table 1: Comparison results of different methods on CUB dataset. We report PCK@10 pixels for all the methods.

Method	Flycatcher	Gull	Sparrow	Tern	Vireo	Warbler	Woodpecker	Wren	**mean**
GL	13.51	12.04	11.30	12.46	11.83	12.29	11.84	11.19	12.06
FC	93.11	82.81	89.75	72.63	86.27	85.75	66.96	84.47	82.72
Delaunay	94.43	85.99	91.68	76.80	89.07	87.54	74.21	87.49	85.90
kNN 1-NN	92.23	86.24	91.81	77.40	87.05	87.20	76.35	86.13	85.59
kNN 2-NN	93.62	85.21	91.44	80.23	89.98	88.33	79.50	88.21	86.96
kNN 3-NN	**95.61**	**86.53**	**92.02**	**81.71**	**90.01**	**89.62**	**80.98**	**88.99**	**88.18**

Table 2: Comparison results of graph structures on PascalVOC dataset. We report PCK@10 pixels for all the methods .

Method	aeroplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table
GL	12.20	10.60	16.31	22.63	13.65	25.76	20.63	12.57	15.14	11.35	20.31
FC	26.20	41.27	44.32	43.23	72.05	70.25	56.43	55.21	34.31	46.87	60.91
Delaunay	35.32	45.03	49.92	44.27	**74.72**	**72.73**	59.15	58.43	36.17	49.45	75.14
kNN 1-NN	34.22	42.08	47.45	**46.32**	73.22	70.73	57.20	56.24	35.15	44.22	**76.82**
kNN 2-NN	36.12	45.14	50.27	41.23	74.31	69.84	**61.21**	**57.13**	35.25	51.34	76.32
kNN 3-NN	**37.23**	**46.04**	**52.31**	46.21	72.72	70.21	57.78	55.23	**39.65**	**52.24**	73.41

Method	dog	horse	motorbike	person	potted plant	sheep	sofa	train	tvmonitor	**mean**
GL	12.23	11.66	15.94	08.89	20.90	15.04	17.09	35.19	36.52	17.73
FC	50.09	51.66	44.62	33.20	72.55	51.78	43.05	88.90	91.06	53.90
Delaunay	53.84	56.05	49.99	35.98	73.65	55.46	51.89	91.66	92.31	56.42
kNN 1-NN	51.7	53.02	46.27	34.23	70.45	52.21	52.91	89.26	90.11	56.29
kNN 2-NN	53.21	55.21	**49.45**	**36.31**	72.37	**57.12**	**54.68**	91.33	91.25	57.78
kNN 3-NN	**55.82**	**57.24**	48.14	35.91	**75.15**	54.29	50.21	**93.23**	**94.45**	**61.22**

results as well, as indicated by the results in FC row. This indicates that having an appropriate graph structure, typically adaptable by learning the graph structure, could be beneficial for graph matching process.

With the graph learning framework of Equation 9, we even sometimes got a null set as the edge set. Therefore, we tried with sigmoid activation and thresholded the activation values at their mean, but it did not help improve the results. Besides, the backward pass was computationally very expensive. It still showed sparse graphs and the results are still very poor as compared to the baselines. Some of the visualisation of these graphs are given in Appendix S3. We obtained similar results for other parameterisations as well. We hypothesize that since we account for pairwise potentials in M_e which adds to the loss, there is a systematic tendency to give sparse/null edge sets.

6. Discussion & Conclusion

In this project, we have explored the problem of combining graph learning proposed in [7] with the graph matching technique as proposed in [25]. The advantage of such a combination is that entire graph matching procedure can be trained end to end along with learning graphs relevant for the matching process. Moreover, comparison to baselines and evaluation on multiple datasets were missing in [7]. Therefore, we have performed experiments on two benchmark datasets on multiple baselines. We found that k nearest neighbours with 3 neighbours connectivity gave the best results among the baselines. We also experimented with a variant of the parameterisation of the adjacency matrix as proposed in [7] and found that it gave sparse/null matrices as edges which in turn resulted in a poorer performance. We also tried different alternatives to this parameterisation. We believe that this sort of parameterisation has a systematic tendency to give sparse/null edge sets when combined with the matching framework of [25] as the matching process we are employing here also accounts for the edge affinities in the loss. This insight does not contradict the conclusions obtained in the original graph learning [7] for matching formulation as there the matching is done by embedding the entire graph using a graph convolutional network [9] and does not employ pairwise potentials for the matching loss. We believe that directly predicting the matrices $\mathbf{G}$ and $\mathbf{H}$ would alleviate the problem. However, this is a much harder problem as the number of edges has to be predicted in an appropriate manner.

References

[1] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola. Learning graph matching. *IEEE PAMI*, 2009. 2

[2] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker. Universal correspondence network. In *NIPS*, 2016. 4

[3] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 2004. 1

[4] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *NIPS*, 2007. 2

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 7

[6] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010. 2, 4

[7] B. Jiang, P. Sun, J. Tang, and B. Luo. Glmnet: Graph learning-matching networks for feature matching. *arXiv preprint arXiv:1911.07681*, 2019. 1, 2, 3, 4, 5

[8] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11313–11320, 2019. 2

[9] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 2, 5

[10] D.-T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 1980. 1

[11] M. Leordeanu, R. Sukthankar, and M. Hebert. Unsupervised learning for graph matching. *IJCV*, 2012. 2

[12] M. Leordeanu, A. Zanfir, and C. Sminchisescu. Semi-supervised learning and optimization for hypergraph matching. In *ICCV*, 2011. 2

[13] R. Li, S. Wang, F. Zhu, and J. Huang. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2

[14] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. *ICML*, 2019. 2

[15] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th International Conference on Data Engineering*, 2002. 2

[16] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6519–6528, 2019. 2

[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 7

[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 2

[19] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 2, 4

[20] R. Wang, J. Yan, and X. Yang. Learning combinatorial embedding networks for deep graph matching. *ICCV*, 2019. 2

[21] L. Wiskott, J.-M. Fellous, N. Krüger, and C. Von Der Malsburg. Face recognition by elastic bunch graph matching. In *International Conference on Computer Analysis of Images and Patterns*, 1997. 2

[22] J. Yan, X.-C. Yin, W. Lin, C. Deng, H. Zha, and X. Yang. A short survey of recent advances in graph matching. In *ACM MM*, 2016. 1, 2

[23] M. Ye, A. J. Ma, L. Zheng, J. Li, and P. C. Yuen. Dynamic label graph matching for unsupervised video re-identification. In *ICCV*, 2017. 2

[24] T. Yu, J. Yan, Y. Wang, W. Liu, et al. Generalizing graph matching beyond quadratic assignment model. In *Advances in Neural Information Processing Systems*, 2018. 2

[25] A. Zanfir and C. Sminchisescu. Deep learning of graph matching. In *CVPR*, 2018. 1, 2, 3, 5, 7

[26] R. Zass and A. Shashua. Probabilistic graph and hypergraph matching (lecture notes), 2014. 1

[27] F. Zhou and F. De la Torre. Factorized graph matching. In *CVPR*, 2012. 1, 2, 7

Appendix

A. Graph Matching Layers Details

A.1. Affinity Matrix Factorization

Zhou and De La Torre [27] showed that the global affinity matrix $\mathbf{M}$ can be factorized as Kronecker product of smaller matrices:

$$\mathbf{M} = [vec(\mathbf{M}_P)] + (\mathbf{G}^1 \otimes \mathbf{G}^2)[vec(\mathbf{M}_e)](\mathbf{H}^2 \otimes \mathbf{H}^1)^T \tag{11}$$

where $[x]$ denotes the matrix with x on its diagonal and $\otimes$ is the Kronecker product. $\mathbf{A}^1 = \mathbf{G}^1\mathbf{H}^{1^T}$ and $\mathbf{A}^2 = \mathbf{G}^2\mathbf{H}^{2^T}$ are the factorisation of the adjacency matrices of the graph where $\mathbf{G}^1_{i,c} = \mathbf{H}^1_{j,c} = 1$ if the c^{th} edge of $\mathcal{G}^1$ starts from the i^{th} node and ends at the j^{th} node. In this work, we aim to learn these adjacency matrices as part of the entire matching framework. $\mathbf{M}_e \in \mathbb{R}^{n\times m}$ measures node-to-node similarities which are projected onto the diagonal of $\mathbf{M}$. In contrast, $\mathbf{M}_p \in \mathbb{R}^{p\times q}$ measures the edge-to-edge similarities between the edges of the two given graphs.

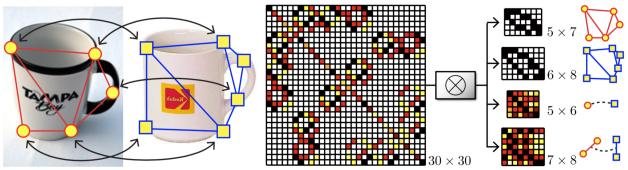

Figure S1: Visualization of the factorized affinity matrix where the two top matrices on the right represent the structures of the graphs and the lower ones represent the node-to-node and edge-to-edge similarities. Figure from [27].

A.2. Feature Extraction

As discussed before, we aim to obtain an affinity matrix which could then be used for graph matching task. One of the ways to achieve this is to get the unary and pairwise node potentials from a network trained on Imagenet [5] and then form the affinity matrix from these features. In this project, we not only obtain features from such a network but also backpropagate through it. We use the VGG architecture [17] for obtaining the features.

More formally, given two images or image patches $(\mathbf{I}^1, \mathbf{I}^2)$ and the keypoint set $(\mathbf{V}^1, \mathbf{V}^2)$, we aim to extract node-features $\mathbf{F}^1, \mathbf{U}^1 \in \mathbb{R}^{n\times d}$ and $\mathbf{F}^2, \mathbf{U}^2 \in \mathbb{R}^{m\times d}$. We obtain the unary potential features $(\mathbf{F}^1, \mathbf{F}^2)$ from relu_4_2 and pairwise potential features $(\mathbf{U}^1, \mathbf{U}^2)$ from relu_5_1, as done in [25].

In a further step, we construct per-edge feature matrices $\mathbf{X}^1 \in \mathbb{R}^{p\times 2d}, \mathbf{X}^2 \in \mathbb{R}^{q\times 2d}$. Therefore, for the c^{th} edge starting at the i^{th} and ending at j^{th} node, we construct the corresponding edge feature through simple node feature concatenation:

$$\Big[\mathbf{X}_1\Big]_c = \Big[[\mathbf{F}^1]_i \Big| [\mathbf{F}^1]_j\Big] \quad \text{and} \quad \Big[\mathbf{X}_2\Big]_c = \Big[[\mathbf{F}^2]_i \Big| [\mathbf{F}^2]_j\Big] \tag{12}$$

A.3. Affinity Matrix

Based on the extracted node features $\mathbf{U}^1, \mathbf{U}^2$, and the constructed edge features $\mathbf{X}^1, \mathbf{X}^2$, the global affinity matrix $\mathbf{M} \in \mathbb{R}^{b\times nm\times nm}$ can be computed using the previously introduced affinity matrix factorization (see section A.1) by the following procedure:

1. Factorize $\mathbf{A}^1, \mathbf{A}^2$ into $\mathbf{G}^1, \mathbf{G}^2, \mathbf{H}^1, \mathbf{H}^2$ s.t.: $\mathbf{A}^1 = \mathbf{G}^1\mathbf{H}^{1^T}$ and $\mathbf{A}^2 = \mathbf{G}^2\mathbf{H}^{2^T}$
2. Compute $\mathbf{M}_e = \mathbf{X^1 \Lambda X^{2^T}}$
3. Compute $\mathbf{M}_p = \mathbf{U}^1\mathbf{U}^{2^T}$
4. Compute

$$\begin{aligned}\mathbf{M} &= [vec(\mathbf{M}_P)] \\ &+ (\mathbf{G}^2 \otimes \mathbf{G}^1)[vec(\mathbf{M}_e)](\mathbf{H}^2 \otimes \mathbf{H}^1)^{T^*}\end{aligned}$$

where $\mathbf{\Lambda} \in \mathbb{R}^{2d\times 2d}$ is a block symmetric parameter matrix which is fixed along the samples (i.e. $\mathbf{\Lambda}_1 = ... = \mathbf{\Lambda}_b$). In addition we enforce following structure s.t. $\mathbf{M}$ keeps its symmetric form and obeys positivity constraints:

$$\mathbf{\Lambda}_1 = ... = \mathbf{\Lambda}_k = \lambda = \begin{bmatrix}\lambda_1 & \lambda_2 \\ \lambda_2 & \lambda_1\end{bmatrix}, \lambda_{i,j} > 0, \forall i, j \tag{13}$$

A.4. Power Iteration

To compute the leading eigenvector $\mathbf{v}^*$ of the global affinity matrix $\mathbf{M}$, we apply the power method for L iterations:

$$\begin{aligned}\mathbf{v}_0 &= \mathbb{1}^{nm\times 1} \\ \mathbf{v}_{l+1} &= \frac{\mathbf{M}\mathbf{v}_l}{\|\mathbf{M}\mathbf{v}_l\|_2} \qquad 0 \le l < L \\ \mathbf{v}^* &= \mathbf{v}_L\end{aligned} \tag{14}$$

A.5. Mapping to Displacements

As a next step, the computed assignment vector $\mathbf{v}^*$ is transformed to a displacement vector $\mathbf{d}$ which describes the assignment for all defined grid points of every image pair $(\mathbf{I}_1 \to \mathbf{I}_2)$ in the batch. We start with reshaping $\mathbf{v}^*$ into a candidate score matrix S:

$$\mathbf{v}^* \to \mathbf{S}, \quad \mathbf{S} = \Big[\mathbf{v}^*\Big]_{n\times m} \tag{15}$$

s.t. vector $\mathbf{S}_i$ describes the candidate scores of the i^{th} grid point of $\mathbf{I}_1$ w.r.t. all m grid points of $\mathbf{I}_2$. In a further step,

we map all candidate score vectors $\mathbf{S}_i$ into a probability distribution by applying the softmax function. Using these normalized scores, the displacements can be computed by weighting the coordinates of the corresponding grid points. Finally, the displacement of the i^{th} grid point of $\mathbf{I}_1$ can be expressed as:

$$\mathbf{d}_i = \frac{\exp[\alpha \mathbf{S}_i]}{\sum_j \exp[\alpha \mathbf{S}_{i,j}]} \mathbf{P}_2 - [\mathbf{P}_1]_i \tag{16}$$

where α is set to 20 and $\mathbf{P}_1 \in \mathbb{R}^{n \times 2}$ and $\mathbf{P}_2 \in \mathbb{R}^{m \times 2}$ are the matrices of vertex positions corresponding to the edge set $\mathbf{V}_1$ and $\mathbf{V}_2$ respectively.

A.6. Loss Function

The final loss per batch can be computed by simply summing up over all grid points of all samples:

$$L(\mathbf{d}) = \sum_i \phi(\mathbf{d}_i - \mathbf{d}_i^{gt}) \tag{17}$$

where $\phi(\mathbf{x}) = \sqrt{\mathbf{x}^T \mathbf{x} + \epsilon}$ defines the penalty term and $\mathbf{d}_i^{gt}$ denotes the corresponding ground-truth displacement solution.

B. Details of Datasets

The **Pascal VOC** dataset is an annotated source of standardised image data for object class recognition. It contains 20 different classes of objects that vary in scale, pose and illumination. All images are available in PNG format and annotated with the ground truth. Ground truth information in each annotated image includes a bounding box for the objects of interest, keypoints within the object and also include pixel segmentation masks. The number of keypoints for each class varies from 6 to 23. We use 7000 annotated images from all 20 categories for training and 1000 images for testing. All images are cropped around its bounding box and resized to 256×256 before feeding to our network for training and testing.

The **CUB dataset** contains 11,788 images of 200 bird categories, with 15 parts annotated. We built our own set which contains bird species from similar families - Figure S2 demonstrates this with vireos and woodpeckers. Overall, we used 8 species which we selected based on the amount of data available: Flycatcher, Gull, Sparrow, Tern, Vireo, Warbler, Woodpecker and Wren. We expect that the birds have a somewhat similar pose in each image pair to be tested. Only visible body parts were taken into account. Each class of the bird consists of 180 - 240 images. We created a 4:1 train/test set with annotations. To train our model we sample random pairs of images.

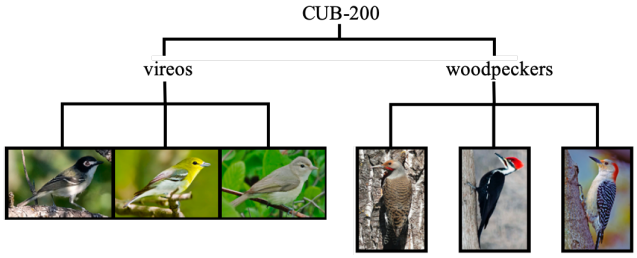

Figure S2: Two family subset (vireos and woodpeckers) from the CUB-200 Dataset.

C. Learned Graphs Visualisation

In this section, we present visualisation of the graphs learned by the graph learning framework on the PascalVOC dataset. We have demonstrated some of the better graphs in the set of images we investigated. Most of them had extremely sparse graphs, and the rest of them slightly sparse, as illustrated in the Figure S3.

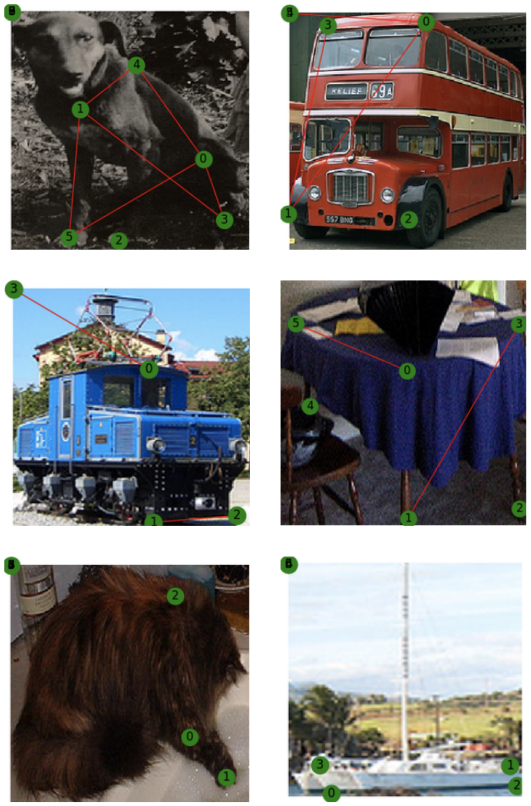

Figure S3: Graphs learnt by the network given images and keypoints on Pascal VOC dataset. It can be seen from examples in the last row that sometimes the edge set is completely emtpy.