Deep Learning of Graph Matching

Andrei Zanfir[2] and Cristian Sminchisescu[1,2]
andrei.zanfir@imar.ro, cristian.sminchisescu@math.lth.se
[1]Department of Mathematics, Faculty of Engineering, Lund University
[2]Institute of Mathematics of the Romanian Academy

Abstract

The problem of graph matching under node and pairwise constraints is fundamental in areas as diverse as combinatorial optimization, machine learning or computer vision, where representing both the relations between nodes and their neighborhood structure is essential. We present an end-to-end model that makes it possible to learn all parameters of the graph matching process, including the unary and pairwise node neighborhoods, represented as deep feature extraction hierarchies. The challenge is in the formulation of the different matrix computation layers of the model in a way that enables the consistent, efficient propagation of gradients in the complete pipeline from the loss function, through the combinatorial optimization layer solving the matching problem, and the feature extraction hierarchy. Our computer vision experiments and ablation studies on challenging datasets like PASCAL VOC keypoints, Sintel and CUB show that matching models refined end-to-end are superior to counterparts based on feature hierarchies trained for other problems.

1. Introduction and Related Work

The problem of graph matching – establishing correspondences between two graphs represented in terms of both local node structure and pair-wise relationships, be them visual, geometric or topological – is important in areas like combinatorial optimization, machine learning, image analysis or computer vision, and has applications in structure-from-motion, object tracking, 2d and 3d shape matching, image classification, social network analysis, autonomous driving, and more. Our emphasis in this paper is on matching graph-based image representations but the methodology applies broadly, to any graph matching problem where the unary and pairwise structures can be represented as deep feature hierarchies with trainable parameters.

Unlike other methods such as RANSAC [12] or iterative closest point [4], which are limited to rigid displacements, graph matching naturally encodes structural information that can be used to model complex relationships and more diverse transformations. Graph matching operates with affinity matrices that encode similarities between unary and pairwise sets of nodes (points) in the two graphs. Typically it is formulated mathematically as a quadratic integer program [25, 3], subject to one-to-one mapping constraints, i.e. each point in the first set must have an unique correspondence in the second set. This is known to be NP-hard so methods often solve it approximately by relaxing the constraints and finding local optima [19, 38].

Learning the parameters of the graph affinity matrix has been investigated by [7, 20] or, in the context of the more general hyper-graph matching model [10], by [21]. In those cases, the number of parameters is low, often controlling geometric affinities between pairs of points rather than the image structure in the neighborhood of those points. Recently there has been a growing interest in using deep features for both geometric and semantic visual matching tasks, either by training the network to directly optimize a matching objective [8, 27, 16, 36] or by using pre-trained, deep features [23, 14] within established matching architectures, all with considerable success.

Our objective in this paper is to marry the (shallow) graph matching to the deep learning formulations. We propose to build models where the graphs are defined over unary node neighborhoods and pair-wise structures computed based on learned feature hierarchies. We formulate a complete model to learn the feature hierarchies so that graph matching works best: the feature learning and the graph matching model are refined in a single deep architecture that is optimized jointly for consistent results. Methodologically, our contributions are associated to the construction of the different matrix layers of the computation graph, obtaining analytic derivatives all the way from the loss function down to the feature layers in the framework of matrix backpropagation, the emphasis on computational efficiency for backward passes, as well as a voting based loss function. The proposed model applies generally, not just for matching different images of a category, taken in different scenes (its primary design), but also to different images of the same

scene, or from a video.

2. Problem Formulation

Input. We are given two input graphs $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, with $|V_1| = n$ and $|V_2| = m$. Our goal is to establish an assignment between the nodes of the two graphs, so that a criterion over the corresponding nodes and edges is optimized (see below).

Graph Matching. Let $\mathbf{v} \in \{0,1\}^{nm\times 1}$ be an indicator vector such that $\mathbf{v}_{ia} = 1$ if $i \in V_1$ is matched to $a \in V_2$ and 0 otherwise, while respecting one-to-one mapping constraints. We build a square symmetric positive matrix $\mathbf{M} \in \mathbb{R}^{nm\times nm}$ such that $\mathbf{M}_{ia;jb}$ measures how well every pair $(i,j) \in E_1$ matches with $(a,b) \in E_2$. For pairs that do not form edges, their corresponding entries in the matrix are set to 0. The diagonal entries contain node-to-node scores, whereas the off-diagonal entries contain edge-to-edge scores. The optimal assignment $\mathbf{v}^*$ can be formulated as

$$\mathbf{v}^* = \underset{\mathbf{v}}{\operatorname{argmax}}\, \mathbf{v}^\top \mathbf{M}\mathbf{v}, \text{ s.t. } \mathbf{C}\mathbf{v} = \mathbf{1}, \mathbf{v} \in \{0,1\}^{nm\times 1} \quad (1)$$

The binary matrix $\mathbf{C} \in \mathbb{R}^{nm\times nm}$ encodes one-to-one mapping constraints: $\forall a \sum_i \mathbf{v}_{ia} = 1$ and $\forall i \sum_a \mathbf{v}_{ia} = 1$. This is known to be NP-hard, so we relax the problem by dropping both the binary and the mapping constraints, and solve

$$\mathbf{v}^* = \underset{\mathbf{v}}{\operatorname{argmax}}\, \mathbf{v}^\top \mathbf{M}\mathbf{v}, \text{ s.t. } \|\mathbf{v}\|_2 = 1 \quad (2)$$

The optimal $\mathbf{v}^*$ is then given by the leading eigenvector of the matrix $\mathbf{M}$. Since $\mathbf{M}$ has non-negative elements, by using Perron-Frobenius arguments, the elements of $\mathbf{v}^*$ are in the interval $[0,1]$, and we interpret $\mathbf{v}^*_{ia}$ as the confidence that i matches a.

Learning. We estimate the matrix $\mathbf{M}$ parameterized in terms of unary and pair-wise point features computed over input images and represented as deep feature hierarchies. We learn the feature hierarchies end-to-end in a loss function that also integrates the matching layer. Specifically, given a training set of correspondences between pairs of images, we adapt the parameters so that the matching minimizes the error, measured as a sum of distances between predicted and ground truth correspondences. In our experiments, we work with graphs constructed over points that correspond to the 2d image projections of the 3d structure of the same physical object in motion (in the context of videos), or over point configurations that correspond to the same semantic category (matching instances of visual categories, e.g. different birds). The main challenge is the propagation of derivatives of the loss function through a factorization of the affinity matrix $\mathbf{M}$, followed by matching (in our formulation, this is an optimization problem, solved using eigen-decomposition) and finally the full feature extraction hierarchy used to compute the unary and pair-wise point representations.

2.1. Derivation Preliminaries

In practice, we build an end-to-end deep network that integrates a feature extracting component that outputs the required descriptors $\mathbf{F}$ for building the matrix $\mathbf{M}$. We solve the assignment problem (2) and compute a matching loss $L(\mathbf{v}^*)$ between the solution $\mathbf{v}^*$ and the ground-truth. The network must be able to pass gradients w.r.t the loss function from the last to the first layer. The key gradients to compute – which we cover in §3 – are $\partial\mathbf{L}/\partial\mathbf{M}$ and $\partial\mathbf{L}/\partial\mathbf{F}$. This computation could be difficult in the absence of an appropriate factorization, as the computational and memory costs become prohibitive. Moreover, as some of our layers implement complex matrix functions, a matrix generalization of back-propagation is necessary [15] for systematic derivations and computational efficiency. In the sequel we cover its main intuition and refer to [15] for details.

Matrix backpropagation. We denote $\mathbf{A} : \mathbf{B} = \text{Tr}(\mathbf{A}^\top\mathbf{B}) = \text{vec}(\mathbf{A})\text{vec}(\mathbf{B})^\top$. For matrix derivatives, if we denote by f a function that outputs $f(\mathbf{X}) = \mathbf{Y}$ and by L the network loss, the basis for the derivation starts from the Taylor expansion of the matrix functions [26] at the two layers. By deriving the functional $\mathcal{L}$ expresssing the total variation $d\mathbf{Y}$ in terms of $d\mathbf{X}$,

$$d\mathbf{Y} = \mathcal{L}(d\mathbf{X}) \quad (3)$$

and then using that

$$\frac{\partial L \circ f}{\partial \mathbf{X}} : d\mathbf{X} = \frac{\partial L}{\partial \mathbf{Y}} : \mathcal{L}(d\mathbf{X}) = \mathcal{L}^*(\frac{\partial L}{\partial \mathbf{Y}}) : d\mathbf{X} \quad (4)$$

we obtain the equality $\partial(L\circ f)/\partial\mathbf{X} = \mathcal{L}^*(\partial L/\partial\mathbf{Y})$, where $\mathcal{L}^*$ is the adjoint operator of $\mathcal{L}$. This recipe for finding the partial derivatives is used across all of our network layers. The derivations of $\mathcal{L}$ and $\mathcal{L}^*$ are layer-specific and are given in the following sections.

2.2. Affinity Matrix Factorization

Zhou and De la Torre [38] introduced a novel factorization of the matrix $\mathbf{M}$ that is generally applicable to all state-of-the-art graph matching methods. It explicitly exposes the graph structure of the set of points and the unary and pair-wise scores between nodes and edges, respectively,

$$\mathbf{M} = [\text{vec}(\mathbf{M}_p)] + (\mathbf{G}_2 \otimes \mathbf{G}_1)[\text{vec}(\mathbf{M}_e)](\mathbf{H}_2 \otimes \mathbf{H}_1)^\top \quad (5)$$

where $[\mathbf{x}]$ represents the diagonal matrix with $\mathbf{x}$ on the main diagonal, and $\otimes$ is the Kronecker product. The matrix

$\mathbf{M}_p \in \mathbb{R}^{n\times m}$ represents the unary term, measuring node-to-node similarities, whereas $\mathbf{M}_e \in \mathbb{R}^{p\times q}$ measures edge-to-edge similarity; p, q are the numbers of edges in each graph, respectively. The two matrices encode the first-order and second-order potentials. To describe the structure of each graph, we define, as in [38], the node-edge incidence matrices as $\mathbf{G}, \mathbf{H} \in \{0,1\}^{n\times p}$, where $g_{ic} = h_{jc} = 1$ if the c^{th} edge starts from the i^{th} node and ends at the j^{th} node. We have two pairs, $\{\mathbf{G}_1, \mathbf{H}_1\} \in \{0,1\}^{n\times p}$ and $\{\mathbf{G}_2, \mathbf{H}_2\} \in \{0,1\}^{m\times q}$, one for each image graph.

One simple way to build $\mathbf{M}_e$ and $\mathbf{M}_p$ is

$$\mathbf{M}_e = \mathbf{X}\mathbf{\Lambda}\mathbf{Y}^\top, \mathbf{M}_p = \mathbf{U}^1\mathbf{U}^{2\top} \tag{6}$$

where $\mathbf{X} \in \mathbb{R}^{p\times 2d}$ and $\mathbf{Y} \in \mathbb{R}^{q\times 2d}$ are the per-edge feature matrices, constructed such that for any c^{th} edge that starts from the i^{th} node and ends at the j^{th} node, we set the edge descriptor as the concatenation of the descriptors extracted at the two nodes

$$\mathbf{X}_c = [\mathbf{F}^1_i | \mathbf{F}^1_j], \mathbf{Y}_c = [\mathbf{F}^2_i | \mathbf{F}^2_j] \tag{7}$$

The matrices $\mathbf{F}^1, \mathbf{U}^1 \in \mathbb{R}^{n\times d}$ and $\mathbf{F}^2, \mathbf{U}^2 \in \mathbb{R}^{m\times d}$ contain per-node feature vectors of dimension d, extracted at possibly different levels in the network, and $\mathbf{\Lambda}$ is a $2d \times 2d$ *block-symmetric* parameter matrix. Superscripts $1, 2$ indicate over which input image (source or target) are the features computed.

3. Deep Network Optimization for Graph Matching

In this section we describe how to integrate and learn the graph matching model end-to-end, by implementing the required components in an efficient way. This allows us to back-propagate gradients all the way from the loss function down to the feature layers. The main components of our approach are shown in Fig. 1.

3.1. Affinity Matrix Layer

If we define the node-to-node adjacency matrices $\mathbf{A}_1 \in \{0,1\}^{n\times n}, \mathbf{A}_2 \in \{0,1\}^{m\times m}$, with $a_{ij} = 1$ if there is an edge from the i^{th} node to the j^{th} node, then

$$\mathbf{A}_1 = \mathbf{G}_1\mathbf{H}_1^\top, \mathbf{A}_2 = \mathbf{G}_2\mathbf{H}_2^\top \tag{8}$$

The *Affinity Matrix* layer receives as input the required hierarchy of features, and the adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ used to reconstruct the optimal $\mathbf{G}_1, \mathbf{H}_1, \mathbf{G}_2, \mathbf{H}_2$ matrices, which verify the equations (8). It is easier to describe the connectivity of the graphs by adjacency matrices than by node-edge incidence matrices, but we still need the latter for efficient backward passes at higher layers of the network. Next, we describe the forward and the backward passes of this layer, as parts of the trainable deep network.

Forward pass.

1. Given $\mathbf{A}_1, \mathbf{A}_2$, recover the matrices $\mathbf{G}_1, \mathbf{H}_1, \mathbf{G}_2, \mathbf{H}_2$, such that $\mathbf{A}_1 = \mathbf{G}_1\mathbf{H}_1^\top, \mathbf{A}_2 = \mathbf{G}_2\mathbf{H}_2^\top$
2. Given $\mathbf{F}^1, \mathbf{F}^2$, build $\mathbf{X}, \mathbf{Y}$ according to (7)
3. Build $\mathbf{M}_e = \mathbf{X}\mathbf{\Lambda}\mathbf{Y}^\top$
4. Given $\mathbf{U}^1, \mathbf{U}^2$, build $\mathbf{M}_p = \mathbf{U}^1\mathbf{U}^{2\top}$
5. Build $\mathbf{M}$ according to (5) and make $\mathbf{G}_1, \mathbf{H}_1, \mathbf{G}_2, \mathbf{H}_2$ available for the upper layers

Backward pass. Assuming the network provides $\partial L/\partial \mathbf{M}_e$ and $\partial L/\partial \mathbf{M}_p$, this layer must return $\partial L/\partial \mathbf{F}^1$, $\partial L/\partial \mathbf{F}^2$, $\partial L/\partial \mathbf{U}^1$ and $\partial L/\partial \mathbf{U}^2$; it must also compute $\partial L/\partial \mathbf{\Lambda}$ in order to update the matrix $\mathbf{\Lambda}$. We assume $\partial L/\partial \mathbf{M}_e$ and $\partial L/\partial \mathbf{M}_p$ as input, and not $\partial L/\partial \mathbf{M}$, because the subsequent layer can take advantage of this special factorization for efficient computation. We note that matrix $\mathbf{\Lambda}$ must have the following form in order for $\mathbf{M}$ to be symmetric with positive elements

$$\mathbf{\Lambda} = \begin{pmatrix} \mathbf{\Lambda}_1 & \mathbf{\Lambda}_2 \\ \mathbf{\Lambda}_2 & \mathbf{\Lambda}_1 \end{pmatrix}, \mathbf{\Lambda}_{ij} > 0, \forall i, j \tag{9}$$

Writing the variation of the loss layer in terms of the variation of edge matrix and using the recipe (4),

$$\begin{aligned} dL &= \frac{\partial L}{\partial \mathbf{M}_e} : d\mathbf{M}_e = \frac{\partial L}{\partial \mathbf{M}_e} : d(\mathbf{X}\mathbf{\Lambda}\mathbf{Y}^\top) \\ &= \frac{\partial L}{\partial \mathbf{M}_e} : (d\mathbf{X}\mathbf{\Lambda}\mathbf{Y}^\top + \mathbf{X}d\mathbf{\Lambda}\mathbf{Y}^\top + \mathbf{X}\mathbf{\Lambda}d\mathbf{Y}^\top) \\ &= \frac{\partial L}{\partial \mathbf{M}_e}\mathbf{Y}\mathbf{\Lambda}^\top : d\mathbf{X} + \mathbf{X}^\top\frac{\partial L}{\partial \mathbf{M}_e}\mathbf{Y} : d\mathbf{\Lambda} + \\ &+ \frac{\partial L}{\partial \mathbf{M}_e}^\top \mathbf{X}\mathbf{\Lambda} : d\mathbf{Y} \end{aligned} \tag{10}$$

We identify the terms

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{X}} &= \frac{\partial L}{\partial \mathbf{M}_e}\mathbf{Y}\mathbf{\Lambda}^\top \\ \frac{\partial L}{\partial \mathbf{\Lambda}} &= \mathbf{X}^\top\frac{\partial L}{\partial \mathbf{M}_e}\mathbf{Y} \\ \frac{\partial L}{\partial \mathbf{Y}} &= \frac{\partial L}{\partial \mathbf{M}_e}\mathbf{X}\mathbf{\Lambda} \end{aligned} \tag{11}$$

To compute the partial derivatives $\partial L/\partial \mathbf{F}^1$ and $\partial L/\partial \mathbf{F}^2$, we identify and sum up the corresponding $1 \times d$ sub-blocks in the matrices $\partial L/\partial \mathbf{X}$ and $\partial L/\partial \mathbf{Y}$. The partial derivative $\partial L/\partial \mathbf{\Lambda}$ is used to compute the derivatives $\partial L/\partial \mathbf{\Lambda}_1$ and $\partial L/\partial \mathbf{\Lambda}_2$. Note that in implementing the positivity condition from (9), one can use a **ReLU** unit.

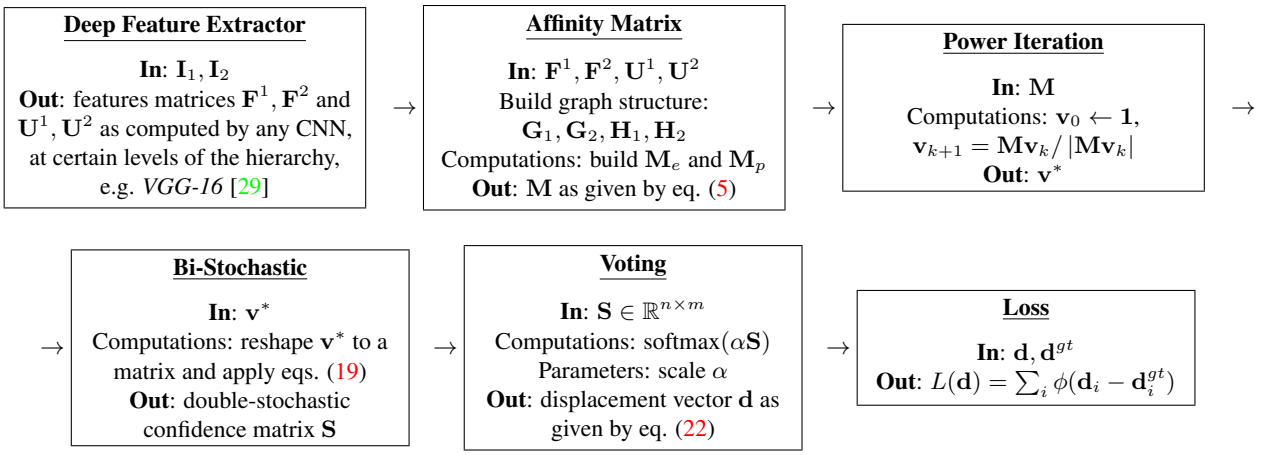

Figure 1: Computational pipeline of our fully trainable graph matching model. In training, gradients w.r.t. the loss function are passed through a deep feature extraction hierarchy controlling the unary and pair-wise terms associated to the nodes and edges of the two graphs, the factorization of the resulting affinity matrix, the eigen-decomposition solution of the matching problem, and the voting-based assignment layer. For each module we show the inputs, outputs and the key variables involved. Detailed derivations for the associated computations are given in the corresponding paper sections.

3.2. Power Iteration Layer

Computing the leading eigenvector $\mathbf{v}^*$ of the affinity matrix $\mathbf{M}$ can be done using power iterations

$$\mathbf{v}_{k+1} = \frac{\mathbf{M}\mathbf{v}_k}{\|\mathbf{M}\mathbf{v}_k\|} \tag{12}$$

We initialize $\mathbf{v}_0 = \mathbf{1}$. We use $\|\cdot\|$ as the ℓ_2 norm, $\|\cdot\|_2$.

Forward pass. We run the assignment from (12) for N iterations, and output the vector $\mathbf{v}^* = \mathbf{v}_N$. Recall that $\mathbf{M} \in \mathbb{R}^{nm \times nm}$, where n, m are the number of nodes in each graph and p, q respectively the number of edges, the time complexity of the forward pass, per power iteration, is $O(n^2m^2)$, when the matrix $\mathbf{M}$ is dense. If we exploit the sparsity of $\mathbf{M}$ the cost drops to $O(\mathbf{nnz})$ where **nnz** represents the number of non-zero elements of the matrix $\mathbf{M}$, being equal to $pq + nm$.

Backward pass. To compute gradients, we express the variation of the loss and identify the required partial derivatives

$$\begin{aligned} dL &= \frac{\partial L}{\partial \mathbf{v}_{k+1}} : d\mathbf{v}_{k+1} = \frac{\partial L}{\partial \mathbf{v}_{k+1}} : d\frac{\mathbf{M}\mathbf{v}_k}{\|\mathbf{M}\mathbf{v}_k\|} \\ d\frac{\mathbf{M}\mathbf{v}_k}{\|\mathbf{M}\mathbf{v}_k\|} &= \frac{(\mathbf{I} - \mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|} d(\mathbf{M}\mathbf{v}_k) = \\ &= \frac{(\mathbf{I} - \mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|} (d\mathbf{M}\mathbf{v}_k + \mathbf{M}d\mathbf{v}_k) \end{aligned} \tag{13}$$

Knowing that $\mathbf{y} : \mathbf{A}\mathbf{x} = \mathbf{y}\mathbf{x}^\top : \mathbf{A} = \mathbf{A}^\top \mathbf{y} : \mathbf{x}$, and using the symmetry of $\mathbf{M}$, we derive

$$\begin{aligned} dL = &\frac{(\mathbf{I} - \mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|} \frac{\partial L}{\partial \mathbf{v}_{k+1}} \mathbf{v}_k^\top : d\mathbf{M} + \\ &+ \mathbf{M}\frac{(\mathbf{I} - \mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|} \frac{\partial L}{\partial \mathbf{v}_{k+1}} : d\mathbf{v}_k \end{aligned} \tag{14}$$

Noticing that $d\mathbf{v}_k$ is still a function of $d\mathbf{M}$, the gradients $\partial L / \partial \mathbf{M}$ and $\partial L / \partial \mathbf{v}_k$ are iteratively built:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{M}} &= \sum_k \frac{(\mathbf{I} - \mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|} \frac{\partial L}{\partial \mathbf{v}_{k+1}} \mathbf{v}_k^\top \\ \frac{\partial L}{\partial \mathbf{v}_k} &= \mathbf{M}\frac{(\mathbf{I} - \mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|} \frac{\partial L}{\partial \mathbf{v}_{k+1}} \end{aligned} \tag{15}$$

where we receive $\partial L / \partial \mathbf{v}^*$ from the upper network layers. The computational cost of (15) is $O(m^2n^2)$ – regardless of the sparsity of $\mathbf{M}$ – and the memory complexity is $\Theta(m^2n^2)$. Such costs are prohibitive in practice. By employing techniques of **matrix back-propagation** [15], we can exploit the special factorization (5) of matrix $\mathbf{M}$, to make operations both memory and time efficient. In fact, set aside efficiency, it would not be obvious how a classic approach would be used in propagating derivatives through a complex factorization like (5). Exploiting (5) and the recipe from (4), and omitting for clarity the term $\mathbf{M}_p$, we obtain (16) as detailed in Fig. 2.

Note that $(\cdot)_{n \times m}$ is the operator that reshapes an $nm \times 1$ vector into an $n \times m$ matrix. For derivations we use the property that, for any compatible matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{V}$, $(\mathbf{A} \otimes \mathbf{B})^\top \mathrm{vec}(\mathbf{V}) = \mathrm{vec}(\mathbf{B}^\top \mathbf{V}\mathbf{A})$. Consequently, by also considering the unary term $\mathbf{M}_p$, it follows that

$$\begin{aligned}
\frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\mathbf{v}_k^\top : d\mathbf{M} &= \frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\mathbf{v}_k^\top : (\mathbf{G}_2\otimes\mathbf{G}_1)[\mathrm{vec}(d\mathbf{M}_e)](\mathbf{H}_2\otimes\mathbf{H}_1)^\top \\
&= \mathrm{diag}\left((\mathbf{G}_2\otimes\mathbf{G}_1)^\top\frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\mathbf{v}_k^\top(\mathbf{H}_2\otimes\mathbf{H}_1)\right) : d\mathbf{M}_e \\
&= (\mathbf{G}_2\otimes\mathbf{G}_1)^\top\frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\odot(\mathbf{H}_2\otimes\mathbf{H}_1)^\top\mathbf{v}_k : d\mathbf{M}_e \\
&= \mathbf{G}_1^\top\left(\frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\right)_{n\times m}\mathbf{G}_2\odot\mathbf{H}_1^\top\left(\mathbf{v}_k\right)_{n\times m}\mathbf{H}_2 : d\mathbf{M}_e \qquad (16)
\end{aligned}$$

Figure 2: Derivations expressing the variation of the loss function w.r.t the variation of the edge affinity matrix $\mathbf{M}_e$, given the variation of the loss function w.r.t the variation of the whole affinity matrix $\mathbf{M}$, from eq. (14)

$$\frac{\partial L}{\partial \mathbf{M}_e} = \sum_k \mathbf{G}_1^\top\left(\frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\right)_{n\times m}\mathbf{G}_2\odot \odot\,\mathbf{H}_1^\top\left(\mathbf{v}_k\right)_{n\times m}\mathbf{H}_2 \qquad (17)$$

$$\frac{\partial L}{\partial \mathbf{M}_p} = \sum_k \frac{(\mathbf{I}-\mathbf{v}_{k+1}\mathbf{v}_{k+1}^\top)}{\|\mathbf{M}\mathbf{v}_k\|}\frac{\partial L}{\partial \mathbf{v}_{k+1}}\odot\mathbf{v}_k \qquad (18)$$

With careful ordering of operations, the complexities for the backward pass are now $O(\max(m^2q, n^2p))$ and $\Theta(pq)$. Taking into account the sparsity of the node-edge incidence matrices $\mathbf{G}, \mathbf{H}$, efficiency can be further improved.

3.3. Bi-Stochastic Layer

We make the result of the *Power Iteration* layer double-stochastic by mapping the eigenvector $\mathbf{v}^*$ on the L1 constraints of the matching problem (1): $\forall a, \sum_i \mathbf{v}_{ia} = 1$ and $\forall i, \sum_a \mathbf{v}_{ia} = 1$. This is suggested by multiple authors [13, 37, 19], as it was observed to significantly improve performance. We introduce a new *Bi-Stochastic* layer that takes as input any correspondence vector $\mathbf{v}^* \in \mathbb{R}_+^{nm\times 1}$, reshaped to a matrix of size $n \times m$, and outputs the double-stochastic variant, as described in [30]. Even though the original algorithm assumes only square matrices, the process can be generalized as shown in [9].

Forward pass. Given a starting matrix $\mathbf{S}_0 = \left(\mathbf{v}^*\right)_{n\times m}$, we run the following assignments for a number of iterations

$$\mathbf{S}_{k+1} = \mathbf{S}_k[\mathbf{1}_n^\top\mathbf{S}_k]^{-1}, \mathbf{S}_{k+2} = [\mathbf{S}_{k+1}\mathbf{1}_m]^{-1}\mathbf{S}_{k+1} \qquad (19)$$

Backward pass. Given a starting gradient $\partial L/\partial \mathbf{S}$, we iteratively compute

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{S}_{k+1}} &= [\mathbf{S}_{k+2}\mathbf{1}_m]^{-1}\frac{\partial L}{\partial \mathbf{S}_{k+2}} - \\
&\quad - \mathrm{diag}\left([\mathbf{S}_{k+2}\mathbf{1}_m]^{-2}\frac{\partial L}{\partial \mathbf{S}_{k+2}}\mathbf{S}_{k+2}^\top\right)\mathbf{1}_m^\top \qquad (20) \\
\frac{\partial L}{\partial \mathbf{S}_k} &= \frac{\partial L}{\partial \mathbf{S}_{k+1}}[\mathbf{1}_n^\top\mathbf{S}_{k+1}]^{-1} - \\
&\quad - \mathbf{1}_n\mathrm{diag}\left([\mathbf{1}_n^\top\mathbf{S}_{k+1}]^{-2}\mathbf{S}_{k+1}^\top\frac{\partial L}{\partial \mathbf{S}_{k+1}}\right)^\top \qquad (21)
\end{aligned}$$

3.4. Converting Confidence-maps to Displacements

We use a special layer, called *Voting* layer to convert from the confidence map $\mathbf{S} \in \mathbb{R}^{n\times m}$ – passed on by the *Bi-Stochastic* layer – to a displacement vector. The idea is to normalize, for each assigned point i, its corresponding candidate scores given by the ith row of $\mathbf{S}$, denoted $\mathbf{S}(i, 1...m)$. We then use it to weight the matrix of positions $\mathbf{P} \in \mathbb{R}^{m\times 2}$ and subtract the position of match i.

$$\mathbf{d}_i = \frac{\exp[\alpha\mathbf{S}(i,1...m)]}{\sum_j \exp[\alpha\mathbf{S}(i,j)]}\mathbf{P} - \mathbf{P}_i \qquad (22)$$

where [] is now just a bulkier bracket. In practice we set α to large values (e.g. 200). By softly voting over the fixed candidate locations, our solution can interpolate for more precise localization. We also set confidences to 0 for candidates that are too far away from assigned points, or those added through padding. Hence these do not contribute to the loss, given by

$$L(\mathbf{d}) = \sum_i \phi(\mathbf{d}_i - \mathbf{d}_i^{gt}) \qquad (23)$$

where $\phi(\mathbf{x}) = \sqrt{\mathbf{x}^\top\mathbf{x} + \epsilon}$ is a robust penalty term, and $\mathbf{d}^{gt}$ is the ground-truth displacement from the source point to

the correct assignment. This layer is implemented in multiple, fully differentiable steps: a) first, scale the input by α, b) use a spatial map for discarding candidate locations that are further away than a certain threshold from the starting location and use it to modify the confidence maps, c) use a softmax layer to normalize the confidence maps, d) compute the displacement map. The discard map sets the confidences to 0 for points that are further away than a certain distance, or for points that were added through padding. Such points do not contribute in the final loss, given by (23).

4. Experiments

In this section we describe the models used as well as detailed experimental matching results, both quantitative (including ablation studies) and qualitative, on three challenging datasets: MPI Sintel, CUB, and PASCAL keypoints.

Deep feature extraction network. We rely on the VGG-16 architecture from [29], that is pretrained to perform classification in the ImageNet ILSVRC [28] but we can use any other deep network architecture. We implement our deep learning framework in MatConvNet [31]. As edge features $\mathbf{F}$ we use the output of layer `relu5_1` (and the entire hierarchy under it), and for the node features $\mathbf{U}$ we use the output of layer `relu4_2` (with the parameters of the associated hierarchy under it). Features are all normalized to 1 through normalization layers, right before they are used to compute the affinity matrix $\mathbf{M}$. We conduct experiments for geometric and semantic correspondences on MPI-Sintel [6], Caltech-UCSD Birds-200-2011 [32] and PASCAL VOC [11] with Berkeley annotations [5].

Matching networks. *GMNwVGG* is our proposed Graph Matching Network based on a VGG feature extractor. The suffix *-U* means that default (initial) weights are used; *-T* means trained end-to-end; the *GMNwVGG-T w/o V* variant does not use, at testing, the *Voting* layer in order to compute the displacements, but directly assigns indices of maximum value across the rows of the confidence map $\mathbf{S}$, as correspondences. *NNwVGG* gives nearest-neighbour matching based on deep node descriptors $\mathbf{U}$.

MPI-Sintel. Besides the main datasets CUB and PASCAL, typically employed in validating matching methods, we also use Sintel in order to demonstrate the generality and flexibility of the formulation. The Sintel input images are consecutive frames in a movie and exhibit large displacements, large appearance changes, occlusion, non-rigid movements and complex atmospheric effects (only included in the *final* set of images). The Sintel training set consists of 23 video sequences (organized as folders) and 1064 frames. In order to make sure that we are fairly training and evaluating, as images from the same video depict instances of the same objects, we split the data into 18 folders (i.e. 796 image pairs) for training and 5 folders (i.e. 245 image pairs) for testing. To be able to set a high number of correspondences under the constraints of memory usage and available computational resolutions of our descriptors, we partition the input images in 4×4 blocks padded for maximum displacement, which we match one by one. Note that for this particular experiment we do not use the *Bi-Stochastic* layer, as the one-to-one mapping constraints do not apply to the nature of this problem (the assignment can be many-to-one e.g. in scaling). Notice that our model is designed to establish correspondences between two images containing similar structures, generally from different scenes, not tailored explicitly for optical flow, where additional smoothness constraints can be exploited. A main point of our Sintel experiments is to show the scalability of our method which operates with affinity matrices of size $10^6 \times 10^6$. A complete forward and backward pass runs in roughly **2 seconds** on a 3.2 Ghz Intel Xeon machine, with Titan X Pascal GPU. [1] We show quantitative results in table 1. We use the Percentage of Correct Keypoints metric to test our matching performance, with a threshold of 10 pixels. We compare against other state-of-the-art matching methods, following the protocol in [8]. Notice that even untrained, our network remains competitive, as the graph structure acts as a regularizer. After training, the network has significantly increased accuracy. Visual examples are given in fig. 4.

Method	PCK@10 pixels
SIFT flow [22]	89.0
DaisyFF [34]	87.3
DSP [17]	85.3
DM [33]	89.2
UCN [8]	91.5
NNwVGG-U	85.9
NNwVGG-T	87.01
GMNwVGG-U	88.03
GMNwVGG-T	**92.6**

Table 1: Comparative matching results for Sintel.

CUB. This dataset contains 11,788 images of 200 bird categories, with bounding box object localization and 15 annotated parts. We use the test set built by [16] which consists of 5,000 image pairs that are within 3 nearest neighbors apart on the pose graph of [18], so we can expect that birds are in somewhat similar poses in each pair to be tested. But across all images, there is a significant variation in pose, articulation and appearance. To train our model, we sample

[1] One possible speed-up could be to not propagate gradients through expensive layers like the relaxation during each power iteration, but do it only once at convergence[1, 2]. In principle, this would not be the correct gradient and in our tests this approach indeed did not converge.

random pairs of images from the training set of CUB-200-2011, which are not present in the test set. The number of points in the two graphs are maximum $n = 15$ and $m = 256$ – as sampled from a 16×16 grid – with a Delaunay triangulation on the first graph, and fully connected on the second. In the *Voting* layer, we no longer discard candidate locations that are too far away from the source points. A complete forward and backward pass runs in 0.3 seconds.

Method	EPE (in pixels)	PCK@0.05
GMNwVGG-U	41.63	0.63
NNwVGG-U	59.05	0.46
GMNwVGG-T w/o V	18.22	0.83
GMNwVGG-T	**17.02**	**0.86**

Table 2: Our models (with ablations) on CUB.

We show quantitative results in table 2. The "PCK@α" metric [35] represents the percentage of predicted correspondences that are closer than $\alpha\sqrt{w^2 + h^2}$ from ground-truth locations, where w, h are image dimensions. Qualitative results are shown in fig. 5.

Our end-to-end fully trainable matching models significantly outperform nearest neighbor approaches (EPE error is halved) based on deep features or similar graph matching formulations based on deep features not refined jointly with the assignment model. Our model *GMNwVGG-T w/o V* which does not use, at testing, the *Voting* layer in order to compute the displacements is inferior to the soft-voting mechanism of our compete method *GMNwVGG-T*. We also achieve state-of-the-art results compared to UCN [8], WarpNet [16], SIFT [24] and DSP [17], *cf.* fig. 3.

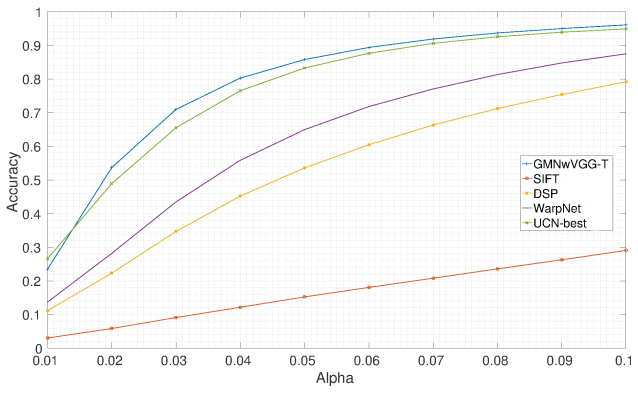

Figure 3: Our methods and other state-of-the-art techniques on CUB.

PASCAL VOC keypoints. This dataset is an extension of PASCAL VOC 2011 and contains annotations of body parts for 20 semantic classes. There are 7,000 annotated examples for training, and 1,700 for testing, but we can form pairs between any two examples of the same class. While the numbers of examples for each class are heavily imbalanced, at training we draw random examples from each class, according to its corresponding probability. At train and test time we crop each example around its bounding box, re-scale it to 256×256 and pass it to the network. We train one matching network for all classes, hence our model is agnostic to the semantic category. This dataset is considerably more challenging than CUB: bounding-boxes can range from very large to extremely small (e.g. 30×30), the training and testing pairs cover combinations of two images from each class (meaning that we cannot expect them to contain objects in the same pose) and the appearance variation is more extreme. We rely on the same protocol for evaluation and setting meta-parameters as in the CUB experiment. Results are shown in Fig. 6, and comparisons in table 3.

Method	PCK@0.1 (Class average)
conv4 flow [23]	24.9
SIFT flow [22]	24.7
UCN [8]	38.9
NNwVGG-U	25.4
GMNwVGG-U	29.8
GMNwVGG-T	**40.6**

Table 3: Our models as well as other state-of-the art approaches on PASCAL VOC.

We test following the same protocol as for **CUB-200-2011** and obtain improvements over state-of-the-art. Notice that results of UCN [8] differ from the paper, as we use the straight average for the 20 semantic classes. Their paper reports the weighted average based on class frequency – under that metric UCN obtains PCK@0.1 = 44 and we obtain 45.3, so the improvement is preserved compared to the direct averaging case.

5. Conclusions

We have presented an end-to-end learning framework for graph matching with general applicability to models containing deep feature extraction hierarchies and combinatorial optimization layers. We formulate the problem as a quadratic assignment under unary and pair-wise node relations represented using deep parametric feature hierarchies. All model parameters are trainable and the graph matching optimization is included within the learning formulation. As such, the main challenges are the calculation of back-propagated derivatives through complex matrix layers and the implementation of the entire framework (factorization of the affinity matrix, bi-stochastic layers) in a computationally efficient manner. Our experiments and ablation studies on diverse datasets like PASCAL VOC keypoints, Sintel and CUB show that fully learned graph matching models

Figure 4: Four visual examples on the MPI-Sintel test partition, which exhibit large motions and occlusion areas. From **top** to **bottom**: the source images with the initial grid of points overlaid and the target images with the corresponding matches as found by our fully trained network. Colors are unique and they encode correspondences. Even for fast moving objects, points tend to track the surface correctly, without sliding – see e.g. the dragon's wing, claw, and the flying monster.

Figure 5: Four qualitative examples of our best performing network *GMNwVGG-T*, on the CUB-200-2011 test-set. Images with a black contour represent the source, whereas images with a red contour represent targets. Color-coded correspondences are found by our method. The green framed images show ground-truth correspondences. The colors of the drawn circular markers uniquely identify 15 semantic keypoints.

Figure 6: Twelve qualitative examples of our best performing network *GMNwVGG-T* on the PASCAL VOC test-set. For every pair of examples, the left shows the source image and the right the target. Colors identify the computed assignments between points. The method finds matches even under extreme appearance and pose changes.

surpass nearest neighbor counterparts, or approaches that use deep feature hierarchies that were not refined jointly with (and constrained by) the quadratic assignment problem.

Acknowledgments: This work was supported in part by the European Research Council Consolidator grant SEED, CNCS-UEFISCDI PN-III-P4-ID-PCE-2016-0535, EU Horizon 2020 Grant No. 688835 DE-ENIGMA, and SSF.

References

[1] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. *CoRR*, abs/1703.00443, 2017.

[2] J. T. Barron and B. Poole. The fast bilateral solver. *CoRR*, abs/1511.03296, 2015.

[3] A. C. Berg, T. L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 26–33. IEEE, 2005.

[4] P. J. Besl and N. D. McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[5] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1365–1372. IEEE, 2009.

[6] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, pages 611–625. Springer, 2012.

[7] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola. Learning graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 31(6):1048–1058, 2009.

[8] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems*, pages 2406–2414, 2016.

[9] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *NIPS*, volume 2, page 6, 2006.

[10] O. Duchenne, F. Bach, I.-S. Kweon, and J. Ponce. A tensor-based algorithm for high-order graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 33(12):2383–2395, 2011.

[11] M. Everingham and J. Winn. The pascal visual object classes challenge 2011 (voc2011) development kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 2011.

[12] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[13] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, 18(4):377–388, 1996.

[14] B. Ham, M. Cho, C. Schmid, and J. Ponce. Proposal flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3475–3484, 2016.

[15] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015.

[16] A. Kanazawa, D. W. Jacobs, and M. Chandraker. Warpnet: Weakly supervised matching for single-view reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3253–3261, 2016.

[17] J. Kim, C. Liu, F. Sha, and K. Grauman. Deformable spatial pyramid matching for fast dense correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2307–2314, 2013.

[18] J. Krause, H. Jin, J. Yang, and L. Fei-Fei. Fine-grained recognition without part annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5546–5555, 2015.

[19] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1482–1489. IEEE, 2005.

[20] M. Leordeanu, R. Sukthankar, and M. Hebert. Unsupervised learning for graph matching. *International journal of computer vision*, 96(1):28–45, 2012.

[21] M. Leordeanu, A. Zanfir, and C. Sminchisescu. Semi-supervised learning and optimization for hypergraph matching. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2274–2281. IEEE, 2011.

[22] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):978–994, 2011.

[23] J. L. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In *Advances in Neural Information Processing Systems*, pages 1601–1609, 2014.

[24] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[25] J. Maciel and J. P. Costeira. A global solution to sparse correspondence problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):187–199, 2003.

[26] J. R. Magnus, H. Neudecker, et al. Matrix differential calculus with applications in statistics and econometrics. 1995.

[27] I. Rocco, R. Arandjelović, and J. Sivic. Convolutional neural network architecture for geometric matching. *arXiv preprint arXiv:1703.05593*, 2017.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

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

[30] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[31] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[32] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[33] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep match-

ing. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1385–1392. IEEE, 2013.

[34] H. Yang, W. Lin, and J. Lu. Daisy filter flow: A generalized approach to discrete dense correspondences. CVPR, 2014.

[35] Y. Yang and D. Ramanan. Articulated human detection with flexible mixtures of parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2878–2890, 2013.

[36] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision*, 2016.

[37] R. Zass and A. Shashua. Probabilistic graph and hypergraph matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[38] F. Zhou and F. De la Torre. Factorized graph matching. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 127–134. IEEE, 2012.