

Fully-Trainable Deep Matching

James Thewlis

jdt@robots.ox.ac.uk

Shuai Zheng

shuai.zheng@eng.ox.ac.uk

Philip H. S. Torr

philip.torr@eng.ox.ac.uk

Andrea Vedaldi

vedaldi@robots.ox.ac.uk

Department of Engineering Science

University of Oxford

Oxford, UK

Abstract

Deep Matching (DM) is a popular high-quality method for quasi-dense image matching. Despite its name, however, the original DM formulation does not yield a deep neural network that can be trained end-to-end via backpropagation. In this paper, we remove this limitation by rewriting the complete DM algorithm as a convolutional neural network. This results in a novel deep architecture for image matching that involves a number of new layer types and that, similar to recent networks for image segmentation, has a U -topology. We demonstrate the utility of the approach by improving the performance of DM by learning it end-to-end on an image matching task.

1 Introduction

Deep Matching (DM) [19] is one of the most popular methods for establishing quasi-dense correspondences between images. An important application of DM is optical flow, where it is used for finding an initial set of image correspondences, which are then interpolated and refined by local optimisation.

The reason for the popularity of DM is the quality of the matches that it can extract. However, there is an important drawback: DM, as originally introduced in [19], is in fact *not* a deep neural network and does not support training via back-propagation. In order to sidestep this limitation, several authors have recently proposed alternative Convolutional Neural Networks (CNN) architectures for dense image matching (Sect. 1.1). However, while several of these trainable models obtain excellent results, they are not necessarily superior to the handcrafted DM architecture in term of performance.

The quality of the matches established by DM demonstrates the strength of the DM architecture compared to alternatives. Thus, a natural question is whether it is possible to obtain the best of both worlds, and construct a trainable CNN architecture which is equivalent to DM. The main contribution of this paper is to carry out such a construction.

In more detail, DM comprises two stages (Fig. 1): In the first stage, DM computes a sequence of increasingly coarse match score maps, integrating information from progressively larger image neighbourhoods in order to remove local match ambiguities. In the second

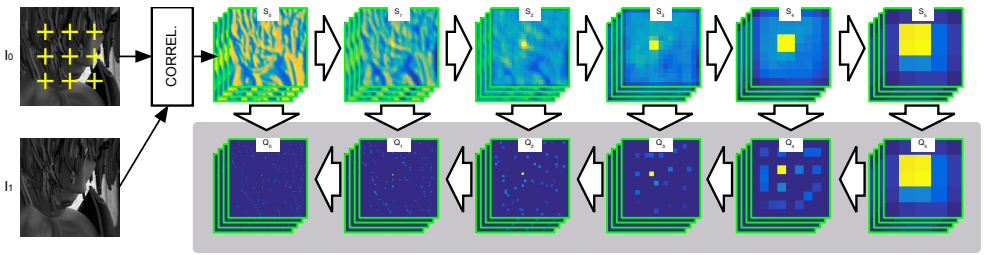


Figure 1: **Fully-Trainable Deep Matching.** Deep matching starts by correlating small patches \mathbf{p} in the reference image I_0 (crosses) with all the patches \mathbf{q} in the target image I_1 , producing a 4D score map S_0 (slices $S_0(\cdot|\mathbf{p})$ for varying \mathbf{p} are shown); then, it computes coarser but less ambiguous maps S_1, \dots, S_L . In this paper, we formulate the reverse process, reconstructing high resolution matches from coarser ones, as a sequence of *reverse convolutional operators*, producing scores Q_L, \dots, Q_0 (shaded area). The result is a deep convolutional network with *U*-architecture that can be trained using backpropagation.

stage, the coarse information is propagated in the reverse direction, resolving ambiguities in the higher-resolution score maps. While the first stage was formulated as a CNN in [19], the second stage was given as a recursive decoding algorithm. In Sect. 2, we show that this recursive algorithm is equivalent to dynamic programming and that it can be implemented instead by a sequence of new *convolutional operators*, that reverse the ones in the first stage of DM.

The resulting CNN architecture (Fig. 2), which is *numerically equivalent to the original DM*, has a *U*-topology, as popularized in image segmentation [20], and supports backpropagation. Combined with a structured-output loss (Sect. 2.2), this allows us to perform end-to-end learning of the DM parameters, improving its performance (Sect. 3). Our findings and further potential advantages of the approach are discussed in Sect. 4.

1.1 Related Work

The key reason for the success of CNNs in many computer vision applications is the ability to learn complex systems end-to-end instead of hand-crafting individual components. A number of recent works have applied CNN-based systems to pixel-wise labeling problems such as stereo matching and optical flow. In particular, Fischer *et al.* [5] have shown it is possible to train a fully convolutional network for optical flow. Žbontar *et al.* [28] trained a CNN for stereo matching by using a refined stereo matching cost. Zagoruyko and Komodakis [27] and Han *et al.* [7] have demonstrated learning local image description through a CNN.

Optical flow estimation was tackled mostly by variational approaches [3, 14, 25] since the work of Horn and Sunk [8]. Brox and Malik [2] developed a system that integrates descriptor matching with a variational approach. Recently, leading optical flow approaches such as DeepMatching [19, 26] demonstrated a CNN-like system where feature information is aggregated from fine to coarse using sparse convolutions and max-pooling. However, this approach does not perform learning and all parameters are hand-tuned. EpicFlow [18] has focused on refining the sparse matches from DM using a variational method that incorporates edge information. Fischer *et al.* [5] trained a fully convolutional network FlowNet for optical flow prediction on a large-scale synthetic flying chair dataset. However, the results of FlowNet do not match the performance of DM on realistic datasets. This motivates us to

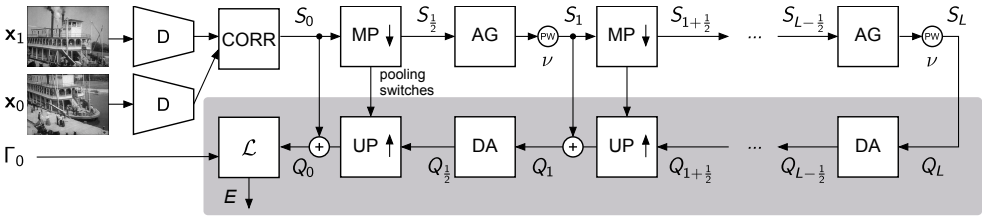


Figure 2: End-to-end deep matching architecture, involving the following layers: descriptor extraction (D), correlation (CORR), max pooling (MP), aggregation (AG), power (PW), disaggregation (DA), unpooling (UP), summation (+), and structured loss (\mathcal{L}). The shaded area encloses our contribution, which amounts to: formulating the DM decoding algorithm as a sequence of convolutional neural network layers supporting backpropagation.

reformulate DM [19] as an end-to-end trainable neural network.

Beyond CNNs, many authors have applied machine learning techniques to matching and optical flow. Sun *et al.* [23] investigate the statistical properties of optical flow and learn the regularizers using Gaussian scale mixtures, Rosenbaum *et al.* [24] use Gaussian mixture models to model the statistics of optical flow, and Black *et al.* [10] apply the idea of principal components analysis to optical flow. Kennedy and Taylor [9] train classifiers to choose different inertial estimations for optical flow. Leordeanu *et al.* [16] obtain occlusion probabilities by learning classifiers. Menze *et al.* [17] formulate optical flow estimation as a discrete inference problem in a conditional random field, followed by sub-pixel refinement. In these works, tuning feature parameters is mostly done separately and manually. In contrast to these works, our work aims to convert the whole quasi-dense matching pipeline into an end-to-end trainable CNN.

2 Method

Our key contribution is to show that the full DM pipeline can be formulated as a CNN with a U-topology (Fig. 2). The fine-to-coarse stage of DM was already given as a CNN in [19]. Here, we complete the construction and show that the DM recursive decoding stage can: (1) be interpreted as dynamic programming and (2) be implemented by convolutional operators which reverse the ones used in the fine-to-coarse stage (Sect. 2.1). The architecture can be trained using backpropagation, for which we propose a structured-output loss (Sect. 2.2).

2.1 Fully-Trainable Deep Matching Architecture

In this section we formulate the *complete* DM algorithm as a CNN. Consider a *reference image* $I_0(\mathbf{p})$, $\mathbf{p} = (p_1, p_2)$ and a *target image* $I_1(\mathbf{q})$, $\mathbf{q} = (q_1, q_2)$. The goal is to estimate a *correspondence field* $\Gamma: \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $\mathbf{p} \mapsto \mathbf{q}$ mapping points \mathbf{p} in the reference image to corresponding points \mathbf{q} in the target image. The correspondence field is found as the maximizer

$$\Gamma(\mathbf{p}) = \underset{\mathbf{q}}{\operatorname{argmax}} S_0(\mathbf{q}|\mathbf{p}) \quad (1)$$

of a scoring function $S_0(\mathbf{q}|\mathbf{p})$ that encodes the similarity of point \mathbf{p} in I_0 with point \mathbf{q} in I_1 (the score has of course an implicit dependency on I_0 and I_1).¹

A simple way of defining the scoring function S_0 is to compare patch descriptors. Thus, let $\phi(\mathbf{q}|I) \in \mathbb{R}^d$ be a visual descriptor of a patch centred at \mathbf{q} in image I ; furthermore, assume that ϕ is L^2 normalised. The score of the match $\mathbf{p} \mapsto \mathbf{q}$ can be defined as the cosine similarity of local descriptors, given by the inner product:

$$S_0(\mathbf{q}|\mathbf{p}) = \langle \phi(\mathbf{p}|I_0), \phi(\mathbf{q}|I_1) \rangle. \quad (2)$$

A significant drawback of this scoring function is that it pools information only locally, from the compared patches. Therefore, unless all patches have a highly distinctive local appearance, many of the matches established by eq. (1) are likely to be incorrect.

Correcting these errors requires integrating global information in the score maps. In order to do so, DM builds a sequence of scoring functions $S_l(\mathbf{q}|\mathbf{p}), l = 0, 1, 2, \dots, L$ which are increasingly coarse but that incorporate information from increasingly larger image neighborhoods (Fig. 1 top). Given these maps, equation (2) is replaced by a recursive decoding process that extracts matches by analysing S_L, S_{L-1}, \dots, S_0 in reverse order.

While the authors of [14] already showed that maps S_l can be computed by convolutional operators, they did not formulate the decoding stage of DM as a network supporting end-to-end learning. Here we show that the recursive decoding process can be reformulated as the computation of additional score maps $Q_l(\mathbf{q}|\mathbf{p}), l = L, L-1, \dots, 1$ (Fig. 1 bottom) by reversing the convolutional operators used to compute S_0, S_1, \dots, S_L . The two stages, fine to coarse and coarse to fine, are described in detail below.

Stage 1: Fine to coarse. DM starts with the scoring function S_0 , computed by comparing local patches as explained above, and builds the other scores by alternating two operations: max pooling and aggregation.

The **max pooling** step pools scores S_l with respect to the first argument \mathbf{q} in a square of side of $2^{l+1}\eta_0$ pixels, where η_0 is a parameter. This results in an intermediate scoring function $S_{l+1/2}$:

$$S_{l+1/2}(\mathbf{q}|\mathbf{p}) = \max \left\{ S_l(\mathbf{q}'|\mathbf{p}), \forall \mathbf{q}' : \|\mathbf{q}' - \mathbf{q}\|_\infty \leq 2^l \eta_0 \right\}. \quad (3)$$

In the following, the locations of the local maxima, also known as *pooling switches*, will be denoted as $\mathbf{q}' = m_l(\mathbf{q}|\mathbf{p})$, where m_l is defined such that $S_{l+1/2}(\mathbf{q}|\mathbf{p}) = S_l(m_l(\mathbf{q}|\mathbf{p})|\mathbf{p})$. Note that max pooling is exactly the same operator as commonly defined in convolutional neural networks. The resulting score $S_{l+1/2}(\mathbf{q}|\mathbf{p})$ can be interpreted as the strength of the best match between \mathbf{p} in the reference image and all points within a distance $2^l \eta_0$ from \mathbf{q} in the target image.

After max pooling, the scores are **aggregated** at the four corners of a square patch of side $2^l \delta_0$ pixels:

$$S_{l+1}(\mathbf{q}|\mathbf{p}) = \left[\frac{1}{4} \sum_{i=1}^4 S_{l+1/2}(\mathbf{q} + 2^l \boldsymbol{\delta}_i | \mathbf{p} + 2^l \boldsymbol{\delta}_i) \right]^v \quad (4)$$

where $\boldsymbol{\delta}_i = (\delta_0/2)\boldsymbol{\epsilon}_i$, $\delta_0 > 0$ is a parameter, and $\boldsymbol{\epsilon}_i$ are the unit displacement vectors:

$$\boldsymbol{\epsilon}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad \boldsymbol{\epsilon}_2 = \begin{bmatrix} -1 \\ +1 \end{bmatrix}, \quad \boldsymbol{\epsilon}_3 = \begin{bmatrix} +1 \\ +1 \end{bmatrix}, \quad \boldsymbol{\epsilon}_4 = \begin{bmatrix} +1 \\ -1 \end{bmatrix}.$$

¹ As proposed in DM, matches can be verified by testing whether they maximize the score also when going from the target image I_1 back to the reference image I_0 : $\text{verified}(\mathbf{p}) = [\forall \mathbf{p}' : Q(\Gamma(\mathbf{p})|\mathbf{p}) \geq Q(\Gamma(\mathbf{p}')|\mathbf{p}')]$.

The exponent ν (set to 1.4 in DM) monotonically rescales the scores, emphasising larger ones. As detailed in [19], the score $S_l(\mathbf{q}|\mathbf{p})$ can be roughly interpreted as the likelihood that a deformable square patch of side $2^{l+1}\delta_0$ centered at \mathbf{p} in the reference image I_0 matches an analogous deformable patch centered at \mathbf{q} in the target image I_1 .

Eq. (4) can be rewritten as the convolution of $S_{l+1/2}$ with a particular 4D filter. Note that most neural network toolboxes are limited to 2+1D or 3+1D convolutions (with 2 or 3 spatial dimension plus one spanning feature channels), whereas here there are four spatial dimensions (given by the join of \mathbf{p} and \mathbf{q}) and one feature channel, *i.e.* the convolution is 4+1D. Hence, while implementing aggregation through convolution is more general, for the particular filter used in DM a direct implementation of (4) is much simpler.

Part 2: Coarse to fine. In the original DM, scores S_0, S_1, \dots, S_L are decoded by a recursive algorithm to obtain the final correspondence field. Here, we give an equivalent algorithm that uses only layer-wise and convolutional operators, with the major advantage of turning DM in an end-to-end learnable convolutional network. Another significant advantage is that the final product is a full, refined score map Q_0 assigning a confidence to all possible matches rather than finding only the best ones.

Since the last operation in the first stage was to apply aggregation to $S_{L-1/2}$ to obtain S_L , the first operation in the reverse order is **disaggregation**. In general, Q_{l+1} is disaggregated to obtain $Q_{l+1/2}$ as follows:

$$Q_{l+1/2}(\mathbf{q}|\mathbf{p}) = \max \left\{ Q_{l+1}(\mathbf{q} - 2^l \delta_i | \mathbf{p} - 2^l \delta_i), i = 1, 2, 3, 4 \right\}. \quad (5)$$

Disaggregation is similar to *deconvolution* [12, 14, 16, 19] or *convolution transpose* [24] as it reverses a linear filtering operation. However, a key difference is that overlapping contributions are maxed out rather than summed.

Next, Q_l is obtained by **unpooling** $Q_{l+1/2}$ and adding the result to $S_l(\mathbf{q}|\mathbf{p})$:

$$Q_l(\mathbf{q}|\mathbf{p}) = S_l(\mathbf{q}|\mathbf{p}) + \max \left\{ Q_{l+1/2}(\mathbf{q}'|\mathbf{p}), \forall \mathbf{q}' : m_l(\mathbf{q}'|\mathbf{p}) = \mathbf{q} \right\} \cup \{-\infty\}. \quad (6)$$

Unpooling is also found in architectures such as deconvnets; however here 1) the result is infilled with $-\infty$ rather than zeros and 2) overlapping unpooled values are maxed out rather than summed. The result of unpooling is summed to $S_l(\mathbf{q}|\mathbf{p})$ to mix coarse and fine grained information.

Next, we discuss the equivalence of these operations to the original DM decoding algorithm. In the fine to coarse stage, through pooling and aggregation, the score $S_0(\mathbf{q}_0|\mathbf{p}_0)$ contributes to the formation of the coarser scores $S_l(\mathbf{q}_l|\mathbf{p}_l), \dots, S_L(\mathbf{q}_L|\mathbf{p}_L)$ along certain paths $(\mathbf{p}_1, \mathbf{q}_1), \dots, (\mathbf{p}_L, \mathbf{q}_L)$ restricted to the set:

$$\mathcal{H}(\mathbf{q}_0|\mathbf{p}_0) = \{(\mathbf{p}_0, \mathbf{q}_0, \mathbf{p}_1, \mathbf{q}_1, \dots, \mathbf{q}_L) : \forall l \exists i : \mathbf{p}_l = \mathbf{p}_{l+1} - 2^l \delta_i, \mathbf{q}_l = m_l(\mathbf{q}_{l+1} - 2^l \delta_i | \mathbf{p}_l)\}.$$

DM associates to the match $\mathbf{q}_0|\mathbf{p}_0$ the sum of the scores along the best of such paths:

$$Q_0(\mathbf{q}_0|\mathbf{p}_0) = \max \left\{ \sum_{l=0}^L S_l(\mathbf{q}_l|\mathbf{p}_l) : (\mathbf{p}_0, \mathbf{q}_0, \mathbf{p}_1, \mathbf{q}_1, \dots, \mathbf{q}_L) \in \mathcal{H}(\mathbf{q}_0|\mathbf{p}_0) \right\}.$$

DM uses recursion and memoization to compute this maximum efficiently; the disaggregation and unpooling steps given above implement a dynamic programming equivalent of this recursive algorithm. This is easily proved; empirically, the two implementations were found to be numerically equivalent as expected.

2.2 Training and loss functions

Training with DM requires to define a suitable *loss function* for the computed scoring function S . One possibility is to minimise the distance $\mathcal{L}(S, \Gamma_0) = \text{mean}_{\mathbf{p}, \mathbf{q}} \|S(\mathbf{q}|\mathbf{p}) - g_\sigma(\mathbf{q} - \Gamma_0(\mathbf{p}))\|^2$ between S and a smoothed indicator function $g_\sigma(z) = \exp(-\|z\|^2/2\sigma^2)$ of the ground truth correspondence field Γ_0 . While a similar loss is often used to learn keypoint detectors with neural networks [4, 13], it has two drawbacks: first, it requires scores to attain pre-specified values when only relative values are relevant and, second, the loss must be carefully rebalanced as $g_\sigma(\mathbf{q} - \Gamma_0(\mathbf{p})) \approx 0$ for the vast majority of pairs (\mathbf{p}, \mathbf{q}) .

In order to avoid these issues, we propose to use instead the following *structured output* loss:

$$\mathcal{L}(S, \Gamma_0) = \sum_{\mathbf{p}, \mathbf{q}} \max\{0, 1 - g_\sigma(\mathbf{q} - \Gamma_0(\mathbf{p})) + S(\mathbf{q}|\mathbf{p}) - S(\Gamma_0(\mathbf{p})|\mathbf{p})\}.$$

Here, the term $1 - g_\sigma(\mathbf{q} - \Gamma_0(\mathbf{p}))$ defines a variable margin for the hinge loss, small when $\mathbf{q} \approx \Gamma_0(\mathbf{p})$ and close to 1 otherwise. This loss looks at relative scores; in fact $\mathcal{L}(S, \Gamma_0) = 0$ requires the correct matches to have score larger than incorrect ones. Furthermore, it is automatically balanced as each term in the summation involves comparing the score of a correct and an incorrect match.

Note that DM defines a whole hierarchy of score maps $(S_0, \dots, S_L, Q_L, \dots, Q_0)$ and a loss can be applied to each level of the hierarchy. In general, we expect application at the last level Q_L to be the most important, as this reflects the final output of the algorithm, but combinations are possible. For n training image pairs $(\mathbf{x}_0^{(i)}, \mathbf{x}_1^{(i)}, \Gamma_0^{(i)})$, and by denoting with \mathbf{w} the parameters of DM, learning reduces to optimizing the objective function:

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \mathcal{L}(Q_L(\mathbf{x}_0^{(i)}, \mathbf{x}_1^{(i)}; \mathbf{w}), \Gamma_0^{(i)}).$$

We follow the standard approach of optimizing the objective using (stochastic) gradient descent [14]. This requires computing the derivative of the loss and DM function Q_L w.r.t. the parameters \mathbf{w} , which can be done using backpropagation. Note that, while derivations are omitted, all layers in the DM architecture are amenable to backpropagation in the usual way.

2.3 Discretization

So far, variables \mathbf{q} and \mathbf{p} have been treated as continuous. However, in a practical implementation these are discretized. By choosing a discretization scheme smartly, we can make the implementation more efficient and simpler. We describe such a scheme here.

For efficiency, DM doubles at each layer the sampling stride of the variable \mathbf{q} and restricts the match \mathbf{q} to be within a given maximum distance of \mathbf{p} . Hence, \mathbf{q} is sampled as:

$$\mathbf{q} = 2^l \gamma_0 (\mathbf{k}_l - 1 - R_l) + \mathbf{p}, \quad \mathbf{k}_l \in \{1, \dots, 2R_l + 1\}^2,$$

where \mathbf{k}_l is a discrete index, γ_0 is the sampling stride (in pixels) at level $l = 0$, $\gamma_0 R_0$ the distance to \mathbf{p} at level 0, and $R_{l+1} = \lceil R_l/2 \rceil$ is halved with each layer. In this expression, and in the rest of the section, summing a scalar to a vector means adding it to all its components.

For efficiency, DM is usually restricted to a quasi-dense grid of points \mathbf{p} in the reference image, given by:

$$\mathbf{p} = \alpha_0 (\mathbf{i}_l - 1 + \boldsymbol{\tau}_l) + \beta_0, \quad \mathbf{i}_l \in \{1, \dots, H_l\} \times \{1, \dots, W_l\}, \quad \boldsymbol{\tau}_l = \mathbf{1}_{l \geq 1} \left(\frac{\delta_0}{2\alpha_0} - \left\lceil \frac{\delta_0}{2\alpha_0} \right\rceil \right).$$

The parameters α_0 and β_0 are the stride and offset of the patch descriptors extracted from the reference image and they remain constant at all layers; however, there is an additional variable offset τ_l to compensate for the effect of discretization in aggregation, as explained below. Here, the symbol $\mathbf{1}_{l \geq 1}$ is one if the condition $l \geq 1$ is satisfied and zero otherwise.

From these definitions, the discretized score maps, denoted with a bar, are given by $\bar{S}_l(\mathbf{k}_l|\mathbf{i}_l) = S_l(\mathbf{q}|\mathbf{p})$, $\bar{S}_{l+1/2}(\mathbf{k}_{l+1}|\mathbf{i}_l) = S_{l+1/2}(\mathbf{q}|\mathbf{p})$, and similarly for \bar{Q}_l .

Simplifications arise by assuming that γ_0 divides exactly the pooling window size η_0 , that α_0 divides δ_0 , and that γ_0 divides α_0 . Under these assumptions, $\bar{S}_{l+1/2}(\mathbf{k}_{l+1}|\mathbf{i}_l)$ is obtained from $\bar{S}_l(\mathbf{k}_l|\mathbf{i}_l)$ by applying the standard CNN max pooling operator with a pooling window size $W = 1 + 2\eta_0/\gamma_0$ and padding $P = \eta_0/\gamma_0 + 2R_{l+1} - R_l$. Note in particular that W is the same at all layers. Since usually $\eta_0 = \gamma_0$, this amounts to 3×3 pooling with a padding of zero or one pixels. The discretized aggregation operator is also simple and given by:

$$\bar{S}_{l+1}(\mathbf{k}_{l+1}|\mathbf{i}_{l+1}) = \frac{1}{4} \sum_{i=1}^4 \bar{S}_{l+\frac{1}{2}} \left(\mathbf{k}_{l+1} \left| \mathbf{i}_{l+1} + 2^i \frac{\delta_0}{2\alpha_0} \boldsymbol{\varepsilon}_i - \tau_l \mathbf{1}_{l=0} \right. \right).$$

Note that, since \mathbf{q} is expressed relatively to \mathbf{p} , aggregation reduces to averaging selected slices of the discretized score maps (*i.e.* there is no shift applied to \mathbf{k}_{l+1}). Note also that for $l \geq 1$, given that α_0 divides δ_0 , the increment applied to the index \mathbf{i}_{l+1} is integer as required. For $l = 0$ and $\alpha_0 = \delta_0$ (as it is usually the case), the shift $\delta_0/2\alpha_0 = 1/2$ is fractional. In this case, however, the additional offset $\tau_1 = -1/2$ restores integer coordinates as needed.

3 Experiments

The primary goal of this section is to demonstrate the benefit of learning the DM parameters using backpropagation compared to hand-tuning. There are several implementations of DM available online; we base ours on the GPU-based version by the original authors² [19], except for the decoding stage for which we use their CPU version with memoization removed. We do so because this eliminates a few small approximations found in the original code. This version is the closest, and in fact numerically equivalent, to our implementation using MatConvNet [24] and our new convolutional operators.

Datasets. The **MPI Sintel** [9] dataset contains 1,041 image pairs and correspondence fields obtained from synthetic data (computer graphics). Scenes are carefully engineered to contain challenging conditions. There are two versions: clean and final (with effects such as motion blur and fog). We consider a subset of the Sintel clean training set to evaluate our methodology. This is dubbed SintelMini, and consists of 7 sequences (313 images) for training and every 10th frame from a different set of 5 sequences (25 images) for validation. The **FlyingChair** dataset by Fischer *et al.* [8] contains synthetically-generated data as Sintel, but with abstract scenes consisting of “flying chairs”. It consists of respectively 22,232/640 train/val image pairs and corresponding flow fields. These images are generated by rendering 3D chair models in front of random background images from Flickr, while the motions of both the chairs and the background are purely planar. The **KITTI flow 2012** [6, 15] dataset contains 194/195 training/testing image pairs and correspondence fields for road scenes. The data contains large baselines but only motions arising from driving a car. Ground truth correspondences are obtained using 3D laser scanner and hence are not available at all pixels.

²<http://lear.inrialpes.fr/src/deepmatching/>.

	Patch descr.	Training set	Elements learned expon. features	Acc@2	Acc@5	Acc@10	EPE (matches)	EPE (flow)
(a)	HOG	—	× ×	84.52%	91.89%	94.36%	3.83	1.88
(b)	HOG	Sintel Mini	✓ ×	84.59%	92.03%	94.49%	3.73	1.84
(c)	CNN	—	× ×	85.28%	92.25%	94.83%	3.58	1.80
(d)	CNN	Sintel Mini	✓ ×	85.30%	92.27%	94.87%	3.70	1.64
(e)	CNN	Sintel Mini	× ✓	86.81%	92.52%	94.86%	3.37	1.60
(f)	CNN	Sintel Mini	✓ ✓	86.79%	92.58%	94.90%	3.34	1.57
(g)	CNN	Flying Chairs	✓ ✓	86.11%	92.47%	94.88%	3.33	1.65

Table 1: **Fully-Trainable DM performance.** DM variants evaluated on Sintel Mini (see text) validation and trained on either Sintel Mini training or Flying Chairs. The top row corresponds to the baseline DM algorithm, equivalent to the GPU version of [18].

Furthermore, the flow is improved by fitting 3D CAD models to observed vehicles on the road and using those to compute displacements.

Evaluation metrics. In order to measure matching accuracy, we adopt the **accuracy@T** metric of Revaud *et al.* [19]. Given the ground truth and estimated dense correspondence fields $\Gamma_0, \Gamma : \mathbf{p} \mapsto \mathbf{q}$ from image $I_0 : \Omega_0 \rightarrow \mathbb{R}$ to image $I_1 : \Omega_1 \rightarrow \mathbb{R}$, **accuracy@T** is the fraction of pixels in Ω_0 correctly matched up to an error of T pixels, *i.e.* $|\{\mathbf{q} \in \Omega_0 : \|\Gamma_0(\mathbf{q}) - \Gamma_1(\mathbf{q})\| \leq T\}|/|\Omega_0|$.³ In addition to **accuracy@T**, we also consider the **end point error** (EPE), obtained as the average correspondence error $\text{mean}_{\mathbf{q} \in \Omega_0} \|\Gamma(\mathbf{q}) - \Gamma_0(\mathbf{q})\|$. In all cases, scores are averaged over all image pairs to yield the final result for a given dataset. If ground truth correspondences are available only at a subset of image locations, Ω_0 is restricted to this set in the definitions above. For the KITTI dataset, we report in particular results for Ω_0 restricted to non-occluded areas (NOC) and all areas (OCC).

Implementation details. For DM, unless otherwise stated we use $L = 6$ layers, $R = 80$ pixels, $\alpha_0 = \delta = 8$, $\beta_0 = 4$, $\gamma_0 = 1$, $\eta_0 = 1.4$. Training uses an NVIDIA Titan X GPU with 12 GBs of on-board memory. Training uses stochastic gradient descent with momentum with mini-batches comprising one image pair at a time (note that an image pair can be seen as the equivalent of a very large batch of image patches).

3.1 Results

End-to-end DM training. In our first experiment (Table 1) we evaluate several variants of DM training. To do so, we consider the smaller and hence more efficient Sintel Mini dataset, a subset of Sintel described above. In Table 1 (a) vs (b) we compare using the default value of $v = 1.4$ used to modulate the output of the aggregation layers and learning values $v_l, l = 1, \dots, L$ specific for each layer. Even with this simple change there is a noticeable improvement (+0.13% acc@10). Next, we replace the HOG features with a trainable CNN architecture ϕ to extract descriptors from image patches. We use the first four convolutional layers (conv1_1, conv1_2, conv2_1, conv2_2) of the pre-trained VGG-VD network [22]. Just by replacing the features, we notice a further improvement ((a) vs (c) +0.47% acc@10) of DM, which can be increased by learning the DM exponents (d). Most interestingly, in (f) we obtain a further improvement by back-propagating from DM to the feature extraction

³Following [19], the quasi-dense DM matches are first filtered by reciprocal verification and then correspondences are propagated to all pixels by assigning to each point \mathbf{q} the same displacement vector of the most confident available nearest available neighbor \mathbf{q}' within a L^∞ -radius of 8 pixels by setting $\Gamma(\mathbf{q}) = \Gamma(\mathbf{q}') - \mathbf{q}' + \mathbf{q}$.

Method	Training	Test	Acc@2	Acc@5	Acc@10	EPE (matches)	EPE (flow)	Err-OCC (flow 3px)
FlowNet S+v [8]	Flying Chairs	KITTI12	-	-	-	-	6.50	-
DM-HOG	—	KITTI12	60.50%	79.34%	84.27%	11.39	3.59	16.56%
DM-CNN	—	KITTI12	61.21%	78.81%	84.01%	12.29	4.11	17.78%
DM-CNN	Flying Chairs	KITTI12	63.90%	80.11%	84.71%	11.12	3.61	16.41%
FlowNet S+v [8]	Flying Chairs	Sintel Final	-	-	-	-	4.76	-
DM [8]	—	Sintel Final	-	-	89.2%	-	4.10	-
DM-HOG	—	Sintel Final	74.37%	85.26%	89.39%	7.08	3.72	11.44%
DM-CNN	—	Sintel Final	75.15%	85.42%	89.48%	7.03	3.63	11.52%
DM-CNN	Flying Chairs	Sintel Final	76.55%	86.22%	90.03%	6.77	3.50	11.10%
FlowNet C+v [8]	Flying Chairs	Sintel Clean	-	-	-	-	3.57	-
DM-HOG	—	Sintel Clean	82.51%	90.18%	92.70%	5.26	2.32	7.00%
DM-CNN	—	Sintel Clean	83.03%	90.24%	92.87%	5.22	2.25	6.85%
DM-CNN	Flying Chairs	Sintel Clean	84.16%	90.85%	93.31%	4.78	2.14	6.51%

Table 2: **Performance comparison.** We train DM variants on large-scale synthetic dataset Flying Chairs, and evaluate on KITTI 12 train and Sintel train. Acc@ n [8] assigns each pixel a nearby match, measuring the proportion correct within n pixels. EPE (endpoint error) is the mean euclidean distance between estimated flow vectors and the ground truth (considering just pixels where ground truth is available). EPE (matches) is computed only at the positions where we have our quasi-dense matches. EPE (flow) measures the endpoint error for the flow estimation, where flow is produced by post-processing the matches with EpicFlow [18]. Err-OCC likewise measures the dense flow, giving proportion of flows off by more than 3 pixels. The version excluding occlusions, Err-NOC, is given in the text.

layers and optimizing the features themselves (hence achieving end-to-end training from the raw pixels to the matching result). The last experiment (g) shows that similar improvements can also be obtained by training from completely unrelated datasets, namely Flying Chairs, indicating that learning generalizes well.

Standard benchmark comparisons. To test DM training in realistic scenarios, we evaluate performance on two standard benchmarks, namely the Sintel and KITTI 2012 training sets (Table 2) as these have publicly-available ground truth to compute accuracy. For training, we use Flying Chairs, which is designed to be statistically similar to the Sintel target dataset. Compared to the HOG-DM baseline, training the CNN patch descriptors in DM improves accuracy@10 by +0.44% on KITTI and by +0.64% on Sintel Final.

An application of DM is optical flow, where it is usually followed by interpolation and refinement such as Brox and Malik [9] or EpicFlow [18]. We use EpicFlow to interpolate our quasi-dense matches and compare the EPE results of FlowNet [8]. While there are better methods than FlowNet for optical flow estimation, we choose it for comparison as this was proposed as a fully-trainable CNN for dense image matching; we compare to their results using variational refinement (+v) which is similar to EpicFlow interpolation. We train our method on Flying Chairs to allow a direct comparison with the results reported in [8].

Compared to the pretrained CNN, training further on Flying Chairs gives a notable improvement in EPE, decreasing from 3.63 to 3.50 for Sintel Final and from 4.11 to 3.61 for KITTI. Compared to HOG, the improvement is even greater for Sintel Final, a gap of 0.22 pixels, however for KITTI the CNN is initially worse than HOG. Training on synthetic data improves most metrics on KITTI, with the exception of EPE (flow). We believe the latter result to be due to the fact that the EpicFlow refinement step, which is not trained, is not optimally tuned to the different statistics of the improved quasi-dense matches. The refinement step is in fact known to be sensitive to the data statistics (for example, in [18] different tunings are used for different datasets). If we exclude occlusions in the ground truth for KITTI,

our trained CNN gets EPE-NOC of 1.43 compared to 1.51 for HOG, and Err-NOC falls from 7.84% to 7.41%.

FlowNet EPEs on KITTI12-Train and Sintel Final Train are respectively 6.50 and 4.76, whereas our trained DM-CNN model has EPEs of 3.61 and 3.50 respectively. This confirms the benefit of the DM architecture, which we turn into a CNN in this paper.

4 Summary

In this paper, we have shown that the *complete* DM algorithm can be equivalently rewritten as a CNN with a *U*-topology, involving a number of new CNN layers. This allows to learn end-to-end the parameters of DM using backpropagation, including the CNN filters that extract the patch descriptors, robustly improving the quality of the correspondence extracted in a number of different datasets.

Once formulated as a modular CNN, components of DM can be easily replaced with new ones. For instance, the max pooling and unpooling units could be substituted with soft versions, resulting in denser score maps, which could result in easier training and in the ability of better expressing the confidence of dense matches. We are currently exploring a number of such extensions.

For the problem of optical flow estimation, it is still required to have EpicFlow as a post-processing step. This type of two-stage approach results a suboptimal solution. In particular, the parameters of EpicFlow are not optimized by end-to-end training with our DM. We would like to explore a solution that allows end-to-end optical flow estimation.

Acknowledgements. This work was supported by the AIMS CDT (EPSRC EP/L015897/1) and grants EPSRC EP/N019474/1, EPSRC EP/I001107/2, ERC 321162-HELIOS, and ERC 677195-IDIU. We gratefully acknowledge GPU donations from NVIDIA.

References

- [1] Michael J. Black, Yaser Yacoob, Allan D. Jepson, and David J. Fleet. Learning parameterized models of image motion. In *IEEE CVPR*, 1997.
- [2] Thomas Brox and Jitendra Malik. Large displacement optical flow: Descriptor matching in variational motion estimation. *IEEE TPAMI*, 33(3):500–513, 2011.
- [3] Thomas Brox, Andres Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, 2004.
- [4] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012.
- [5] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *IEEE ICCV*, 2015.
- [6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE CVPR*, 2012.

- [7] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *IEEE CVPR*, 2015.
- [8] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(3):185–203, 1981.
- [9] Ryan Kennedy and Camillo J. Taylor. Optical flow with geometric occlusion estimation and fusion of multiple frames. In *EMMCVPR*, 2015.
- [10] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.
- [11] Marius Leordeanu, Andrei Zanfir, and Cristian Sminchisescu. Locally affine sparse-to-dense matching for motion and occlusion estimation. In *IEEE ICCV*, 2013.
- [12] Jon Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE CVPR*, 2015.
- [13] Jonathan Long, Ning Zhang, and Trevor Darrell. Do convnets learn correspondence? In *NIPS*, 2014.
- [14] Etienne Mèmin and Patrick Pérez. Dense estimation and object-based segmentation of the optical flow with robust techniques. *IEEE TIP*, 7(5):703–719, 1998.
- [15] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *IEEE CVPR*, 2015.
- [16] Moritz Menze, Christian Heipke, and Andreas Geiger. Discrete optimization for optical flow. In *GCPR*, 2015.
- [17] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *IEEE ICCV*, 2015.
- [18] Jérôme Revaud, Philippe Weinzaepfel, Zaïd Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *IEEE CVPR*, 2015.
- [19] Jérôme Revaud, Philippe Weinzaepfel, Zaïd Harchaoui, and Cordelia Schmid. Deep-matching: Hierarchical deformable dense matching. *IJCV*, 2015.
- [20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.
- [21] Dan Rosenbaum, Daniel Zoran, and Yair Weiss. Learning the local statistics of optical flow. In *NIPS*, 2013.
- [22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
- [23] Deqing Sun, Stefan Roth, J.P. Lewis, and Michael J. Black. Learning optical flow. In *ECCV*, 2008.

- [24] Andrea Vedaldi and Karel Lenc. MatConvNet: Convolutional neural networks for MATLAB. In *ACM MM*, 2015.
- [25] Andreas Wedel, Daniel Cremers, Thomas Pock, and Horst Bischof. Structured motion-adaptive regularization for high accuracy optical flow. In *IEEE ICCV*, 2009.
- [26] Philippe Weinzaepfel, Jérôme Revaud, Zaïd Harchaoui, and Cordelia Schmid. Deep-flow: Large displacement optical flow with deep matching. In *IEEE ICCV*, 2013.
- [27] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *IEEE CVPR*, 2015.
- [28] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *The Journal of Machine Learning Research*, 17 (65):1–32, 2016.
- [29] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *IEEE CVPR*, 2010.