DeepMatching: Hierarchical Deformable Dense Matching

Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, Cordelia Schmid

Noname manuscript No.
(will be inserted by the editor)

DeepMatching: Hierarchical Deformable Dense Matching

Jerome Revaud **Philippe Weinzaepfel** **Zaid Harchaoui**
Cordelia Schmid
INRIA
firstname.lastname@inria.fr

the date of receipt and acceptance should be inserted later

Abstract We introduce a novel matching algorithm, called DeepMatching, to compute dense correspondences between images. DeepMatching relies on a hierarchical, multi-layer, correlational architecture designed for matching images and was inspired by deep convolutional approaches. The proposed matching algorithm can handle non-rigid deformations and repetitive textures and efficiently determines dense correspondences in the presence of significant changes between images.

We evaluate the performance of DeepMatching, in comparison with state-of-the-art matching algorithms, on the Mikolajczyk (Mikolajczyk et al 2005), the MPI-Sintel (Butler et al 2012) and the Kitti (Geiger et al 2013) datasets. DeepMatching outperforms the state-of-the-art algorithms and shows excellent results in particular for repetitive textures.

We also propose a method for estimating optical flow, called DeepFlow, by integrating DeepMatching in the large displacement optical flow (LDOF) approach of Brox and Malik (2011). Compared to existing matching algorithms, additional robustness to large displacements and complex motion is obtained thanks to our matching approach. DeepFlow obtains competitive performance on public benchmarks for optical flow estimation.

Keywords Non-rigid dense matching, optical flow.

1 Introduction

Computing correspondences between related images is a central issue in many computer vision problems, ranging from scene recognition to optical flow estimation (Forsyth and Ponce 2011; Szeliski 2010). The goal of a matching algorithm is to discover shared visual content between two images, and to establish as many as possible precise point-wise correspondences, called *matches*. An essential aspect of matching approaches is the amount of rigidity they assume when computing the correspondences. In fact, matching approaches range between two extreme cases: stereo matching, where matching hinges upon strong geometric constraints, and matching "in the wild", where the set of possible transformations from the source image to the target one is large and the problem is basically almost unconstrained. Effective approaches have been designed for matching rigid objects across images in the presence of large viewpoint changes (Lowe 2004; Barnes et al 2010; HaCohen et al 2011). However, the performance of current state-of-the-art matching algorithms for images "in the wild", such as consecutive images in real-world videos featuring fast non-rigid motion, still calls for improvement (Xu et al 2012; Chen et al 2013). In this paper, we aim at tackling matching in such a general setting.

Matching algorithms for images "in the wild" need to accommodate several requirements, that turn out to be often in contradiction. On one hand, matching objects necessarily requires rigidity assumptions to some extent. It is also mandatory that these objects have sufficiently discriminative textures to make the problem well-defined. On the other hand, many objects or regions are not rigid objects, like humans or animals. Furthermore, large portions of an image are usually occupied by weakly-to-no textured regions, often with repetitive textures, like sky or bucolic background.

Descriptor matching approaches, such as SIFT (Lowe 2004) or HOG (Dalal and Triggs 2005; Brox and Malik 2011) matching, compute discriminative feature representations from rectangular patches. However, while

LEAR team, Inria Grenoble Rhone-Alpes, Laboratoire Jean Kuntzmann, CNRS, Univ. Grenoble Alpes, France.

these approaches succeed in case of rigid motion, they fail to match regions with weak or repetitive textures, as local patches are poorly discriminative. Furthermore, matches are usually poor and imprecise in case of non-rigid deformations, as these approaches rely on rigid patches. Discriminative power can be traded against increased robustness to non-rigid deformations. Indeed, propagation-based approaches, such as Generalized PatchMatch (Barnes et al 2010) or Non-rigid Dense Correspondences (HaCohen et al 2011), compute simple feature representations from small patches and propagate matches to neighboring patches. They yield good performance in case of non-rigid deformations. However, matching repetitive textures remains beyond the reach of these approaches.

In this paper we propose a novel approach, called DeepMatching, that gracefully combines the strengths of these two families of approaches. DeepMatching is computed using a multi-layer architecture, which breaks down patches into a hierarchy of sub-patches. This architecture allows to work at several scales and handle repetitive textures. Furthermore, within each layer, local matches are computed assuming a restricted set of feasible rigid deformations. Local matches are then propagated up the hierarchy, which progressively discard spurious incorrect matches. We called our approach DeepMatching, as it is inspired by deep convolutional approaches.

In summary, we make three contributions:

- **Dense matching**: we propose a matching algorithm, DeepMatching, that allows to robustly determine dense correspondences between two images. It explicitly handles non-rigid deformations, with bounds on the deformation tolerance, and incorporates a multi-scale scoring of the matches, making it robust to repetitive or weak textures. Furthermore, our approach is based on gradient histograms, and thus robust to appearance changes caused by illumination and color variations.
- **Fast, scale/rotation-invariant matching**: we propose a computationally efficient version of DeepMatching, which performs almost as well as exact DeepMatching, but at a much lower memory cost. Furthermore, this fast version of DeepMatching can be extended to a scale and rotation-invariant version, making it an excellent competitor to state-of-the-art descriptor matching approaches.
- **Large-displacement optical flow**: we propose an optical flow approach which uses DeepMatching in the matching term of the large displacement variational energy minimization of Brox and Malik (2011). We show that DeepMatching is a better choice compared to the HOG descriptor used by Brox and Malik (2011) and other state-of-the-art matching algorithms. The approach, named DeepFlow, obtains competitive results on public optical flow benchmarks.

This paper is organized as follows. After a review of previous works (Section 2), we start by presenting the proposed matching algorithm, DeepMatching, in Section 3. Then, Section 4 describes several extensions of DeepMatching. In particular, we propose an optical flow approach, DeepFlow, in Section 4.3. Finally, we present experimental results in Section 5.

A preliminary version of this article has appeared in Weinzaepfel et al (2013). This version adds (1) an in-depth presentation of DeepMatching; (2) an enhanced version of DeepMatching, which improves the match scoring and the selection of entry points for backtracking; (3) proofs on time and memory complexity of DeepMatching as well as its deformation tolerance; (4) a discussion on the connection between Deep Convolutional Neural Networks and DeepMatching; (5) a fast approximate version of DeepMatching; (6) a scale and rotation invariant version of DeepMatching; and (7) an extensive experimental evaluation of DeepMatching on several state-of-the-art benchmarks. The code for DeepMatching as well as DeepFlow are available at `http://lear.inrialpes.fr/src/deepmatching/` and `http://lear.inrialpes.fr/src/deepflow/`. Note that we provide a GPU implementation in addition to the CPU one.

2 Related work

In this section we review related work on "general" image matching, that is matching without prior knowledge and constraints, and on matching in the context of optical flow estimation, that is matching consecutive images in videos.

2.1 General image matching

Image matching based on local features has been extensively studied in the past decade. It has been applied successfully to various domains, such as wide baseline stereo matching (Furukawa et al 2010) and image retrieval (Philbin et al 2010). It consists of two steps, *i.e.*, extracting local descriptors and matching them. Image descriptors are extracted in rigid (generally square) local frames at sparse invariant image locations (Mikolajczyk et al 2005; Szeliski 2010). Matching then equals nearest neighbor search between descriptors, followed by an optional geometric verification. Note that a confidence value can be obtained by computing the uniqueness of a match, *i.e.*, by looking at the distance of its

nearest neighbors (Lowe 2004; Brox and Malik 2011). While this class of techniques is well suited for well-textured rigid objects, it fails to match non-rigid objects and weakly textured regions.

In contrast, the proposed matching algorithm, called *DeepMatching*, is inspired by non-rigid 2D warping and deep convolutional networks (LeCun et al 1998a; Uchida and Sakoe 1998; Keysers et al 2007). This family of approaches explicitly models non-rigid deformations. We employ a novel family of feasible warpings that does not enforce monotonicity nor continuity constraints, in contrast to traditional 2D warping (Uchida and Sakoe 1998; Keysers et al 2007). This makes the problem computationally much less expensive.

It is also worthwhile to mention the similarity with non-rigid matching approaches developed for a broad range of applications. Ecker and Ullman (2009) proposed a similar pipeline to ours (albeit more complex) to measure the similarity of small images. However, their method lacks a way of merging correspondences belonging to objects with contradictory motions, *e.g.*, on different focal planes. For the purpose of establishing dense correspondences between images, Wills et al (2006) estimated a non-rigid matching by robustly fitting smooth parametric models (homography and splines) to local descriptor matches. In contrast, our approach is non-parametric and model-free.

Recently, fast algorithms for dense patch matching have taken advantage of the redundancy between overlapping patches (Barnes et al 2010; Korman and Avidan 2011; Sun 2012; Yang et al 2014). The insight is to propagate good matches to their neighborhood in a loose fashion, yielding dense non-rigid matches. In practice, however, the lack of a smoothness constraint leads to highly discontinuous matches. Several works have proposed ways to fix this. HaCohen et al (2011) reinforce neighboring matches using an iterative multi-scale expansion and contraction strategy, performed in a coarse-to-fine manner. Yang et al (2014) include a guided filtering stage on top of PatchMatch, which obtains smooth correspondence fields by locally approximating a MRF. Finally, Kim et al (2013) propose a hierarchical matching to obtain dense correspondences, using a coarse-to-fine (top-down) strategy. Loopy belief propagation is used to perform inference.

In contrast to these approaches, DeepMatching proceeds bottom-up and, then, top-down. Due to its hierarchical nature, DeepMatching is able to consider patches at several scales, thus overcoming the lack of distinctiveness that affects small patches. Yet, the multi-layer construction allows to efficiently perform matching allowing semi-rigid local deformations. In addition, DeepMatching can be computed efficiently, and can be further accelerated to satisfy low-memory requirements with negligible loss in accuracy.

2.2 Matching for flow estimation

Variational energy minimization is currently the most popular framework for optical flow estimation. Since the pioneering work of Horn and Schunck (1981), research has focused on alleviating the drawbacks of this approach. A series of improvements were proposed over the years (Black and Anandan 1996; Werlberger et al 2009; Bruhn et al 2005; Papenberg et al 2006; Baker et al 2011; Sun et al 2014b; Vogel et al 2013a). The variational approach of Brox et al (2004) combines most of these improvements in a unified framework. The energy decomposes into several terms, resp. the data-fitting and the smoothness terms. Energy minimization is performed by solving the Euler-Lagrange equations, reducing the problem to solving a sequence of large and structured linear systems.

More recently, the addition of a descriptor matching term in the energy to be minimized was proposed by Brox and Malik (2011). Following this idea, several papers (Tola et al 2008; Brox and Malik 2011; Liu et al 2011; Hassner et al 2012) show that dense descriptor matching improves performance. Strategies such as reciprocal nearest-neighbor verification (Brox and Malik 2011) allow to prune most of the false matches. However, a variational energy minimization approach that includes such a descriptor matching term may fail at locations where matches are missing or wrong.

Related approaches tackle the problem of dense scene correspondence. SIFT-flow (Liu et al 2011), one of the most famous method in this context, also formulates the matching problem in a variational framework. Hassner et al (2012) improve over SIFT-flow by using multi-scale patches. However, this decreases performance in cases where scale invariance is not required. Xu et al (2012) integrate matching of SIFT (Lowe 2004) and PatchMatch (Barnes et al 2010) to refine the flow initialization at each level. Excellent results are obtained for optical flow estimation, yet at the cost of expensive fusion steps. Leordeanu et al (2013) extends sparse matches with locally affine constraints to dense matches and, then, uses a total variation algorithm to refine the flow estimation. We present here a computationally efficient and competitive approach for large displacement optical flow by integrating the proposed DeepMatching algorithm into the approach of Brox and Malik (2011).

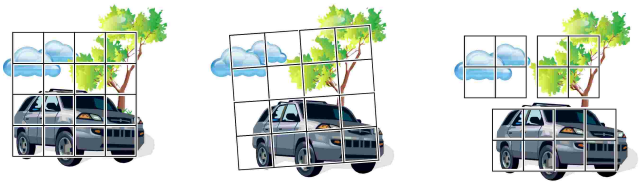

Fig. 1 Illustration of moving quadrant similarity: a quadrant is a quarter of a SIFT patch, *i.e.* a group of 2×2 cells. *Left:* SIFT descriptor in the first image. *Middle:* second image with optimal standard SIFT matching (rigid). *Right:* second image with optimal *moving quadrant* SIFT matching. In this example, the patch covers various objects moving in different directions: for instance the car moves to the right while the cloud to the left. Rigid matching fails to capture this, whereas the moving quadrant approach is able to follow each object.

3 DeepMatching

This section introduces our matching algorithm DeepMatching. DeepMatching is a matching algorithm based on correlations at the patch-level, that proceeds in a multi-layer fashion. The multi-layer architecture relies on a quadtree-like patch subdivision scheme, with an extra degree of freedom to locally re-optimize the positions of each quadrant. In order to enhance the contrast of the spatial correlation maps output by the local correlations, a nonlinear transformation is applied after each layer.

We first give an overview of DeepMatching in Section 3.1 and show that it can be decomposed in a bottom-up pass followed by a top-down pass. We, then, present the bottom-up pass in Section 3.2 and the top-down one in Section 3.3. Finally, we analyze DeepMatching in Section 3.4.

3.1 Overview of the approach

A state-of-the-art approach for matching regions between two images is based on the SIFT descriptor (Lowe 2004). SIFT is a histogram of gradients with 4×4 spatial and 8 orientation bins, yielding a robust descriptor $\mathbf{R} \in \mathbb{R}^{4\times4\times8}$ that effectively encodes a square image region. Note that its 4×4 cell grid can also be viewed as 4 so-called "quadrants" of 2×2 cells, see Figure 1. We can, then, rewrite $\mathbf{R} = [\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3]$ with $\mathbf{R}_i \in \mathbb{R}^{2\times2\times8}$.

Let $\mathbf{R}$ and $\mathbf{R}'$ be the SIFT descriptors of the corresponding regions in the source and target image. In order to remove the effect of non-rigid motion, we propose to optimize the *positions* $p_i \in \mathbb{R}^2$ of the 4 quadrants of the target descriptor $\mathbf{R}'$ (rather than keeping them fixed), in order to maximize

$$\mathrm{sim}(\mathbf{R}, \mathbf{R}') = \max_{\{p_i\}} \frac{1}{4} \sum_{i=0}^{3} \mathrm{sim}\left(\mathbf{R}_i, \mathbf{R}'_i(p_i)\right), \qquad (1)$$

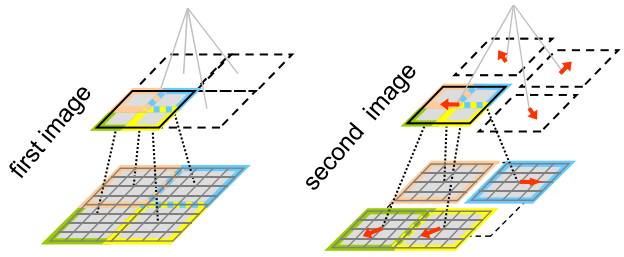

Fig. 2 *Left:* Quadtree-like patch hierarchy in the first image. *Right:* one possible displacement of corresponding patches in the second image.

where $\mathbf{R}'_i(p_i)$ is the descriptor of a single quadrant extracted at position p_i and sim() a similarity function. Now, $\mathrm{sim}(\mathbf{R}, \mathbf{R}')$ is able to handle situations such as the one presented in Figure 1, where a region contains multiple objects moving in different directions. Furthermore, if the four quadrants can move independently (of course, within some extent), it can be calculated more efficiently as:

$$\mathrm{sim}(\mathbf{R}, \mathbf{R}') = \frac{1}{4} \sum_{i=0}^{3} \max_{p_i} \mathrm{sim}\left(\mathbf{R}_i, \mathbf{R}'_i(p_i)\right), \qquad (2)$$

When applied recursively to each quadrant by subdivided it into 4 sub-quadrants until a minimum patch size is reached (atomic patches), this strategy allows for accurate non-rigid matching. Such a recursive decomposition can be represented as a quad-tree, see Figure 2. Given an initial pair of two matching regions, retrieving atomic patch correspondences is then done in a top-down fashion (*i.e.* by recursively applying Eq. (2) to the quadrant's positions $\{p_i\}$).

Nevertheless, in order to first determine the set of matching regions between the two images, we need to compute beforehand the matching scores (*i.e.* similarity) of all large-enough patches in the two images (as in Figure 1), and keep the pairs with maximum similarity. As indicated by Eq. (2), the score is formed by averaging the max-pooled scores of the quadrants. Hence, the process of computing the matching scores is bottom-up. In the following, we call *correlation map* the matching scores of a single patch from the first image at every position in the second image. Selecting matching patches then corresponds to finding local maxima in the correlation maps.

To sum-up, the algorithm can be decomposed in two steps: (i) first, correlation maps are computed using a bottom-up algorithm, as shown in Figure 6. Correlation maps of small patches are first computed and then aggregated to form correlation maps of larger patches; (ii) next, a top-down method estimates the motion of atomic patches starting from matches of large patches.

In the remainder of this section, we detail the two steps described above (Section 3.2 and Section 3.3), before analyzing the properties of DeepMatching in Section 3.4.

3.2 Bottom-up correlation pyramid computation

Let I and I' be two images of resolution $W \times H$ and $W' \times H'$.

Bottom level. We use patches of size 4×4 pixels as atomic patches. We split I into non-overlapping atomic patches, and compute the correlation map with image I' for each of them, see Figure 3. The score between two atomic patches $\mathbf{R}$ and $\mathbf{R}'$ is defined as the average pixel-wise similarity:

$$\text{sim}(\mathbf{R}, \mathbf{R}') = \frac{1}{16} \sum_{i=0}^{3} \sum_{j=0}^{3} \mathbf{R}_{i,j}^{\top} \mathbf{R}'_{i,j}, \tag{3}$$

where each pixel $\mathbf{R}_{i,j}$ is represented as a histogram of oriented gradients pooled over a local neighborhood. We detail below how the pixel descriptor is computed.

Pixel descriptor $\mathbf{R}_{i,j}$*:* We rely on a robust pixel representation that is similar in spirit to SIFT and DAISY (Lowe 2004; Tola et al 2010). Given an input image I, we first apply a Gaussian smoothing of radius ν_1 in order to denoise I from potential artifacts caused for example by JPEG compression. We then extract the gradient $(\delta x, \delta y)$ at each pixel and compute its non-negative projection onto 8 orientations $\left\{(\cos i\frac{\pi}{4}, \sin i\frac{\pi}{4})\right\}_{i=1..8}$. At this point, we obtain 8 oriented gradient maps. We smooth each map with a Gaussian filter of radius ν_2. Next we cap strong gradients using a sigmoid $x \mapsto 2/(1+\exp(-\varsigma x)) - 1$, to help canceling out effects of varying illumination. We smooth gradients one more time for each orientation with a Gaussian filter of radius ν_3. Finally, the descriptor for each pixel is obtained by the ℓ_2-normalized concatenation of 8 oriented gradients and a ninth small constant value μ. Appending μ amounts to adding a regularizer that will reduce the importance of small gradients (*i.e.* noise) and ensures that two pixels lying in areas without gradient information will still correlate positively. Pixel descriptors $\mathbf{R}_{i,j}$ are compared using dot-product and the similarity function takes value in the interval $[0, 1]$. In Section 5.2.1, we evaluate the impact of the parameters of this pixel descriptor.

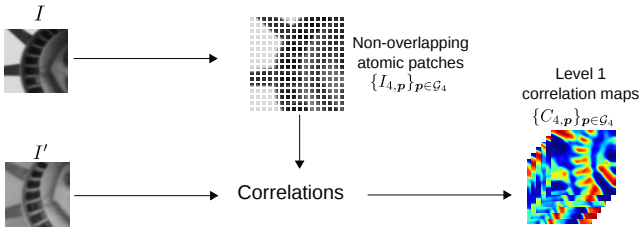

Fig. 3 Computing the bottom level correlation maps $\{C_{4,\boldsymbol{p}}\}_{\boldsymbol{p}\in\mathcal{G}_4}$. Given two images I and I', the first one is split into non-overlapping atomic patches of size 4×4 pixels. For each patch, we compute the correlation at every location of I' to obtain the corresponding correlation map.

Bottom-level correlation map: We can express the correlation map computation obtained from Eq. (3) more conveniently in a convolutional framework. Let $I_{N,\boldsymbol{p}}$ be a patch of size $N \times N$ from the first image centered at $\boldsymbol{p}$ ($N \geq 4$ is a power of 2). Let $\mathcal{G}_4 = \{2, 6, 10, ..., W-2\} \times \{2, 6, 10, ..., H-2\}$ be a grid with step 4 pixels. $\mathcal{G}_4$ is the set of the centers of the atomic patches. For each $\boldsymbol{p} \in \mathcal{G}_4$, we convolve the flipped patch $I^{\text{F}}_{4,\boldsymbol{p}}$ over I'

$$C_{4,\boldsymbol{p}} = I^{\text{F}}_{4,\boldsymbol{p}} \star I', \tag{4}$$

to get the correlation map $C_{4,\boldsymbol{p}}$, where $.^{\text{F}}$ denotes an horizontal and vertical flip[1]. For any pixel $\boldsymbol{p}'$ of I', $C_{4,\boldsymbol{p}}(\boldsymbol{p}')$ is a measure of similarity between $I_{4,\boldsymbol{p}}$ and $I'_{4,\boldsymbol{p}'}$. Examples of such correlation maps are shown in Figure 3 and Figure 4. Without surprise we can observe that atomic patches are not discriminative. Recursive aggregation of patches in subsequent stages will be the key to create discriminative responses.

Iteration. We then compute the correlation maps of larger patches by aggregating those of smaller patches. As shown in Figure 5, a $N \times N$ patch $I_{N,\boldsymbol{p}}$ is the concatenation of 4 patches of size $N/2 \times N/2$:

$$I_{N,\boldsymbol{p}} = \left[I_{\frac{N}{2},\boldsymbol{p}+\frac{N}{4}\boldsymbol{o}_i} \right]_{i=0..3} \quad \text{with} \quad \begin{cases} \boldsymbol{o}_0 = [-1,-1]^{\top}, \\ \boldsymbol{o}_1 = [-1,+1]^{\top}, \\ \boldsymbol{o}_2 = [+1,-1]^{\top}, \\ \boldsymbol{o}_3 = [+1,+1]^{\top}. \end{cases} \tag{5}$$

They correspond respectively to the bottom-left, top-left, bottom-right and top-right quadrants. The correlation map of $I_{N,\boldsymbol{p}}$ can thus be computed using its children's correlation maps. For the sake of clarity, we define the short-hand notation $\text{s}_{N,i} = \frac{N}{4}\boldsymbol{o}_i$ describing the positional shift of a children patch $i \in [0, 3]$ relatively to its parent patch (see Figure 5).

Using the above notations, we rewrite Eq. (2) by replacing $\text{sim}(\mathbf{R}, \mathbf{R}') \overset{def}{=} C_{N,\boldsymbol{p}}(\boldsymbol{p}')$ (*i.e.* assuming here that patch $\mathbf{R} = I_{N,\boldsymbol{p}}$ and that $\mathbf{R}'$ is centered at $\boldsymbol{p}' \in I'$).

[1] This amounts to the cross-correlation of the patch and I'.

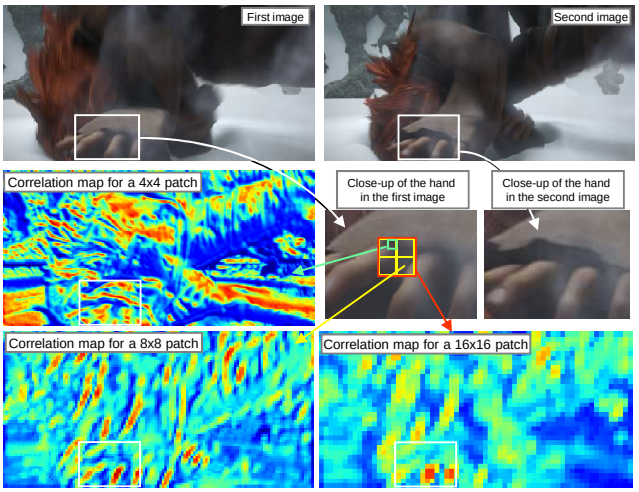

Fig. 4 Correlation maps for patches of different size. *Middle-left*: correlation map of a 4x4 patch. *Bottom-right*: correlation map of a 16x16 patch obtained by aggregating correlation responses of children 8x8 patches (*bottom-left*), themselves obtained from 4x4 patches. The map of the 16x16 patch is clearly more discriminative than previous ones despite the change in appearance of the region.

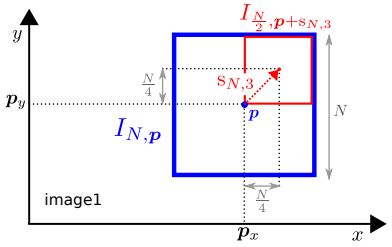

Fig. 5 A patch $I_{N,\boldsymbol{p}}$ from the first image (blue box) and one of its 4 quadrants $I_{\frac{N}{2},\boldsymbol{p}+\frac{N}{4}\boldsymbol{o}_3}$ (red box).

Similarly, we replace the similarity between children patches $\text{sim}\,(\mathbf{R}_i, \mathbf{R}'_i(\boldsymbol{p}'_i))$ by $C_{\frac{N}{2},\boldsymbol{p}+\text{s}_{N,i}}(\boldsymbol{p}'_i)$. For each child, we retain the maximum similarity over a small neighborhood Θ_i of width and height $\frac{N}{8}$ centered at $\boldsymbol{p}'+\text{s}_{N,i}$. We then obtain:

$$C_{N,\boldsymbol{p}}(\boldsymbol{p}') = \frac{1}{4}\sum_{i=0}^{3} \max_{\boldsymbol{m}'\in\Theta_i} C_{\frac{N}{2},\boldsymbol{p}+\text{s}_{N,i}}(\boldsymbol{m}') \tag{6}$$

We now explain how we can break down Eq. (6) into a succession of simple operations. First, let us assume that $N = 4\times 2^\ell$, where $\ell \geq 1$ is the current iteration. During iteration ℓ, we want to compute the correlation maps $C_{N,\boldsymbol{p}}$ of every patch $I_{N,\boldsymbol{p}}$ from the first image for which correlation maps of its children have been computed in the previous iteration. Formally, the position $\mathcal{G}_N$ of such patches is defined according to the position of children patches $\mathcal{G}_{\frac{N}{2}}$ according to Eq. (5):

$$\mathcal{G}_N = \{\boldsymbol{p} \;\mid\; \boldsymbol{p}+\text{s}_{N,i} \in [0,W-1]\times[0,H-1] \wedge \boldsymbol{p}+\text{s}_{N,i} \in \mathcal{G}_{\frac{N}{2}}, i=0,\ldots,3\Big\}. \tag{7}$$

We observe that the larger a patch is (*i.e.* after several iterations), the smaller the spatial variation of its correlation map (see Figure 4). This is due to the statistics of natural images, in which low frequencies significantly dominate over high frequencies. As a consequence, we choose to subsample each map $C_{N,\boldsymbol{p}}$ by a factor 2. We express this with an operator $\mathcal{S}$:

$$\mathcal{S} : C(\boldsymbol{p}') \to C(2\boldsymbol{p}'). \tag{8}$$

The subsampling reduces by 4 the area of the correlation maps and, as a direct consequence, the computational requirements. Instead of computing the subsampling on top of Eq. (6), it is actually more efficient to propagate it towards the children maps and perform it jointly with max-pooling. It also makes the max-pooling domain Θ_i become independent from N in the subsampled maps, as it exactly cancels out the effect of doubling $N = 4\times 2^\ell$ at each iteration. We call $\mathcal{P}$ the max-pooling operator with the iteration-independent domain $\Theta = \{-1,0,1\}\times\{-1,0,1\}$:

$$\mathcal{P} : C(\boldsymbol{p}') \to \max_{\boldsymbol{m}\in\{-1,0,1\}^2} C(\boldsymbol{p}'+\boldsymbol{m}). \tag{9}$$

For the same reason, the shift $\text{s}_{N,i} = \frac{N}{4}\boldsymbol{o}_i = 2^\ell \boldsymbol{o}_i$ applied to the correlation maps in Θ_i's definition becomes simply $\boldsymbol{o}_i$ after subsampling. Let $\mathcal{T}_{\boldsymbol{t}}$ be the shift (or translation) operator on the correlation map:

$$\mathcal{T}_{\boldsymbol{t}} : C(\boldsymbol{p}') \to C(\boldsymbol{p}'-\boldsymbol{t}). \tag{10}$$

Finally, we incorporate an additional non-linear mapping at each iteration on top of Eq. (6) by applying a power transform $\mathcal{R}_\lambda$ (Malik and Perona 1990; LeCun et al 1998a):

$$\mathcal{R}_\lambda : C(.) \to C(.)^\lambda \tag{11}$$

This step, commonly referred to as rectification, is added in order to better propagate high correlations after each level, or, in other words, to counterbalance the fact that max-pooling tends to retain only high scores. Indeed, its effect is to decrease the correlation values (which are in $[0,1]$) as we use $\lambda > 1$. Such post-processing is commonly used in deep convolutional networks (LeCun et al 1998b; Bengio 2009). In practice, good performance is obtained with $\lambda \simeq 1.4$, see Section 5. The final expression of Eq. (6) is:

$$C_{N,\boldsymbol{p}} = \mathcal{R}_\lambda\left(\frac{1}{4}\sum_{i=0}^{3}(\mathcal{T}_{\boldsymbol{o}_i}\circ\mathcal{S}\circ\mathcal{P})\left(C_{\frac{N}{2},\boldsymbol{p}+\text{s}_{N,i}}\right)\right) \tag{12}$$

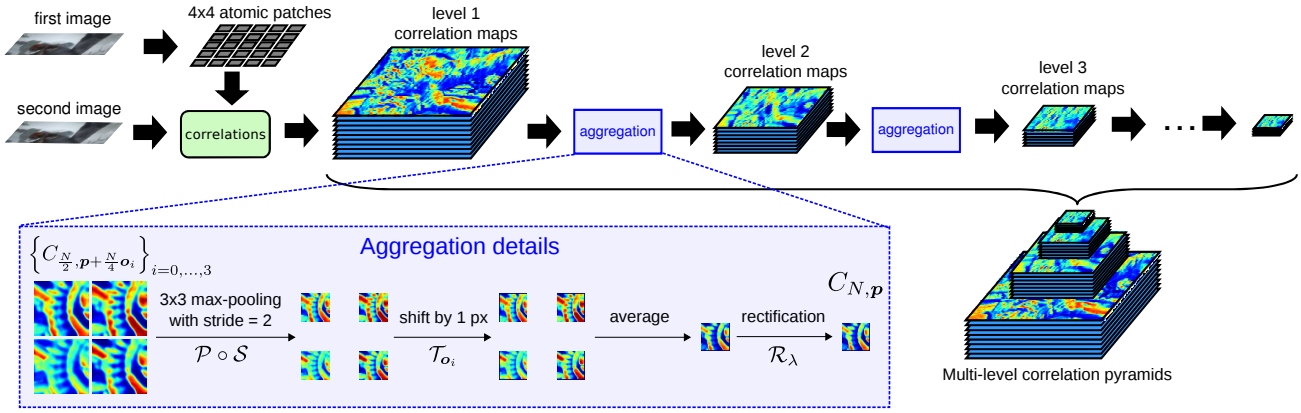

Fig. 6 Computing the multi-level correlation pyramid. Starting with the bottom-level correlation maps, see Figure 3, they are iteratively aggregated to obtain the upper levels. Aggregation consists of max-pooling, subsampling, computing a shifted average and non-linear rectification.

Figure 6 illustrates the computation of correlation maps for different patch sizes and Algorithm 1 summarizes our approach. The resulting set of correlation maps across iterations is referred to as multi-level correlation pyramid.

Boundary effects: In practice, a patch $I_{N,\boldsymbol{p}}$ can overlap with the image boundary, as long as its center $\boldsymbol{p}$ remains inside the image (from Eq. (7)). For instance, a patch $I_{N,\boldsymbol{p}_0}$ with center at $\boldsymbol{p}_0 = (0,0) \in \mathcal{G}_N$ has only a single valid child (the one for which $i = 3$ as $\boldsymbol{p}_0 + \mathrm{s}_{N,3} \in I$). In such degenerate cases, the average sum in Eq. (12) is carried out on valid children only. For $I_{N,\boldsymbol{p}_0}$, it thus only comprises one term weighted by 1 instead of $\frac{1}{4}$.

Note that Eq. (12) implicitly defines the set of possible displacements of the approach, see Figures 2 and 9. Given the position of a parent patch, each child patch can move only within a small extent, equal to the quarter of its own size. Figure 4 shows the correlation maps for patches of size 4, 8 and 16. Clearly, correlation maps for larger patch are more and more discriminative, while still allowing non-rigid matching.

Algorithm 1 Computing the multi-level correlation pyramid.

Input: Images I, I'
For $\boldsymbol{p} \in \mathcal{G}_4$ **do**
  $C_{4,\boldsymbol{p}} = I^{\mathrm{F}}_{4,\boldsymbol{p}} \star I'$ (convolution, Eq. (4))
  $C_{4,\boldsymbol{p}} \leftarrow \mathcal{R}_\lambda(C_{4,\boldsymbol{p}})$ (rectification, Eq. (11))
$N \leftarrow 4$
While $N < \max(W, H)$ **do**
  For $\boldsymbol{p} \in \mathcal{G}_N$ **do**
    $C'_{N,\boldsymbol{p}} \leftarrow (\mathcal{S} \circ \mathcal{P})(C_{N,\boldsymbol{p}})$ (max-pooling and subsampling)
  $N \leftarrow 2N$
  For $\boldsymbol{p} \in \mathcal{G}_N$ **do**
    $C_{N,\boldsymbol{p}} = \frac{1}{4}\sum_{i=0}^{3} \mathcal{T}_{\boldsymbol{o}_i}\left(C'_{\frac{N}{2},\boldsymbol{p}+\mathrm{s}_{N,i}}\right)$ (shift and average)
    $C_{N,\boldsymbol{p}} \leftarrow \mathcal{R}_\lambda(C_{N,\boldsymbol{p}})$ (rectification, Eq. (11))
Return the multi-level correlation pyramid $\{C_{N,\boldsymbol{p}}\}_{N,\boldsymbol{p}}$

3.3 Top-down correspondence extraction

A score $S = C_{N,\boldsymbol{p}}(\boldsymbol{p}')$ in the multi-level correlation pyramid represents the deformation-tolerant similarity of two patches $I_{N,\boldsymbol{p}}$ and $I'_{N,\boldsymbol{p}'}$. Since this score is built from the similarity of 4 matching sub-patches at the lower pyramid level, we can thus recursively backtrack a set of correspondences to the bottom level (corresponding to matches of atomic patches). In this section, we first describe this backtracking. We, then, present the procedure for merging atomic correspondences backtracked from different entry points in the multi-level pyramid, which constitute the final output of DeepMatching.

Compared to our initial version of DeepMatching (Weinzaepfel et al 2013), we have updated match scoring and entry point selection to optimize the execution time and the matching accuracy. A quantitative comparison is provided in Section 5.2.2.

Backtracking atomic correspondences. Given an entry point $C_{N,\boldsymbol{p}}(\boldsymbol{p}')$ in the pyramid (*i.e.* a match between two patches $I_{N,\boldsymbol{p}}$ and $I'_{N,\boldsymbol{p}'}$[2]), we retrieve atomic correspondences by successively undoing the steps used to aggregate correlation maps during the pyramid construction, see Figure 7. The entry patch $I_{N,\boldsymbol{p}}$ is itself composed of four moving quadrants $I^i_{N,\boldsymbol{p}}$, $i = 0 \ldots 3$. Due to the subsampling, the quadrant $I^i_{N,\boldsymbol{p}} = I_{\frac{N}{2},\boldsymbol{p}+\mathrm{s}_{N,i}}$

[2] Note that $I'_{N,\boldsymbol{p}'}$ only roughly corresponds to a $N \times N$ square patch centered at $2^\ell \boldsymbol{p}'$ in I', due to subsampling and possible deformations.

matches with $I_{\frac{N}{2},2(\boldsymbol{p}'+\boldsymbol{o}_i)+\boldsymbol{m}_i}$ where

$$\boldsymbol{m}_i = \underset{\boldsymbol{m}\in\{-1,0,1\}^2}{\arg\max}\ C_{\frac{N}{2},\boldsymbol{p}+\mathrm{s}_{N,i}}\left(2(\boldsymbol{p}'+\boldsymbol{o}_i)+\boldsymbol{m}\right). \quad (13)$$

For the sake of clarity, we define the short-hand notations $\boldsymbol{p}_i = \boldsymbol{p} + \mathrm{s}_{N,i}$ and $\boldsymbol{p}'_i = 2(\boldsymbol{p}' + \boldsymbol{o}_i) + \boldsymbol{m}_i$. Let B be the function that assigns to a tuple $(N, \boldsymbol{p}, \boldsymbol{p}', s)$, representing a correspondence between pixel $\boldsymbol{p}$ and $\boldsymbol{p}'$ for patch of size N with a score $s \in \mathbb{R}$, the set of the correspondences of children patches:

$$B(N,\boldsymbol{p},\boldsymbol{p}',s) = \begin{cases} \{(\boldsymbol{p},\boldsymbol{p}',s)\} & \text{if } N = 4, \\ \left\{\left(\frac{N}{2},\boldsymbol{p}_i,\boldsymbol{p}'_i, s + C_{\frac{N}{2},\boldsymbol{p}_i}(\boldsymbol{p}'_i)\right)\right\}_{i=0}^{3} & \text{else.} \end{cases} \quad (14)$$

Given a set $\mathcal{M}$ of such tuples, let $\mathcal{B}(\mathcal{M})$ be the union of the sets $B(c)$ for all $c \in \mathcal{M}$. Note that if all candidate correspondences $c \in \mathcal{M}$ corresponds to atomic patches, then $\mathcal{B}(\mathcal{M}) = \mathcal{M}$.

Thus, the algorithm for backtracking correspondences is the following. Consider an entry match $\mathcal{M} = \{(N, \boldsymbol{p}, \boldsymbol{p}', C_{N,\boldsymbol{p}}(\boldsymbol{p}'))\}$. We repeatedly apply $\mathcal{B}$ on $\mathcal{M}$. After $\mathfrak{N} = \log_2(N/4)$ calls, we get one correspondence for each of the $4^{\mathfrak{N}}$ atomic patches. Furthermore, their score is equal to the sum of all patch similarities along their backtracking path.

Merging correspondences. We have shown how to retrieve atomic correspondences from a match between two deformable (potentially large) patches. Despite this flexibility, a single match is unlikely to explain the complex set of motions that can occur, for example, between two adjacent frames in a video, *i.e.*, two objects moving independently with significantly different motions exceeds the deformation range of DeepMatching. We quantitatively specify this range in the next subsection.

We thus merge atomic correspondences gathered from different entry points (matches) in the pyramid. In the initial version of DeepMatching (Weinzaepfel et al 2013), entry points were local maxima over all correlation maps. This is now replaced by a faster procedure, that starts with all possible matches in the top pyramid level (*i.e.* $\mathcal{M} = \{(N, \boldsymbol{p}, \boldsymbol{p}', C_{N,\boldsymbol{p}}(\boldsymbol{p}'))|N = N_{\max}\}$). Using this level only results in significantly less entry points than starting from all maxima in the entire pyramid. We did not observe any impact on the matching performance, see Section 5.2.2. Because $\mathcal{M}$ contains a lot of overlapping patches, most of the computation during repeated calls to $\mathcal{M} \leftarrow \mathcal{B}(\mathcal{M})$ can be factorized. In other words, as soon as two tuples in $\mathcal{M}$ are equal in terms of N, $\boldsymbol{p}$ and $\boldsymbol{p}'$, the one with the lowest score is simply eliminated. We thus obtain a set of atomic correspondences $\mathcal{M}'$:

$$\mathcal{M}' = (\mathcal{B} \circ \ldots \circ \mathcal{B})(\mathcal{M}) \quad (15)$$

that we filter with reciprocal match verification. The final set of correspondences $\mathcal{M}''$ is obtained as:

$$\mathcal{M}'' = \left\{(\boldsymbol{p},\boldsymbol{p}',s)|\mathrm{BestAt}(\boldsymbol{p}) = \mathrm{BestAt}'(\boldsymbol{p}')\right\}_{(\boldsymbol{p},\boldsymbol{p}',s)\in\mathcal{M}'} \quad (16)$$

where $\mathrm{BestAt}(\boldsymbol{p})$ (resp. $\mathrm{BestAt}'(\boldsymbol{p}')$) returns the best match in a small vicinity of 4×4 pixels around $\boldsymbol{p}$ in I (resp. around $\boldsymbol{p}'$ in I') from $\mathcal{M}'$.

3.4 Discussion and Analysis of DeepMatching

Multi-size patches and repetitive textures. During the bottom-up pass of the algorithm, we iteratively aggregate correlation maps of smaller patches to form the correlation maps of larger patches. Doing so, we effectively consider patches of different sizes $(4 \times 2^\ell, \ell \geq 0)$, in contrast to most existing matching methods. This is a key feature of our approach when dealing with repetitive textures. As one moves up to upper levels, the matching problem gets less ambiguous. Hence, our method can correctly match repetitive patterns, see for instance Figure 8.

Quasi-dense correspondences. Our method retrieves dense correspondences for every single match between large regions (*i.e.* entry point for the backtracking in the top-level correlation maps), even in weakly textured areas; this is in contrast to correspondences obtained when matching descriptors (*e.g.* SIFT). A quantitative assessment, which compares the coverage of matches obtained with several matching schemes, is given in Section 5.

Non-rigid deformations. Our matching algorithm is able to cope with various sources of image deformations: object-induced or camera-induced. The set of feasible deformations, explicitly defined by Eq. (6), theoretically allows to deal with a scaling factor in the range $[\frac{1}{2}, \frac{3}{2}]$ and rotations approximately in the range $[-26^o, 26^o]$. Note also that DeepMatching is translation-invariant by construction, thanks to the convolutional nature of the processing.

Proof Given a patch of size $N = 4 \times 2^\ell$ located at level $\ell \geqslant 1$, Eq. (6) allows each of its children patches to move by at most $N/8$ pixels from their ideal location in Θ_i. By recursively summing the displacements at each level, the maximal displacements for an atomic patch is $d_N = \sum_{i=1}^{\ell} 2^{i-1} = 2^\ell - 1$. An example is given in Figure 9 with $N = 32$ and $\ell = 3$. Relatively to N, we thus have $\lim_{N\to\infty} (N + 2d_N)/N = \frac{3}{2}$ and $\lim_{N\to\infty} (N - 2d_N)/N = \frac{1}{2}$. For a rotation, the rationale is similar, see Figure 9. □

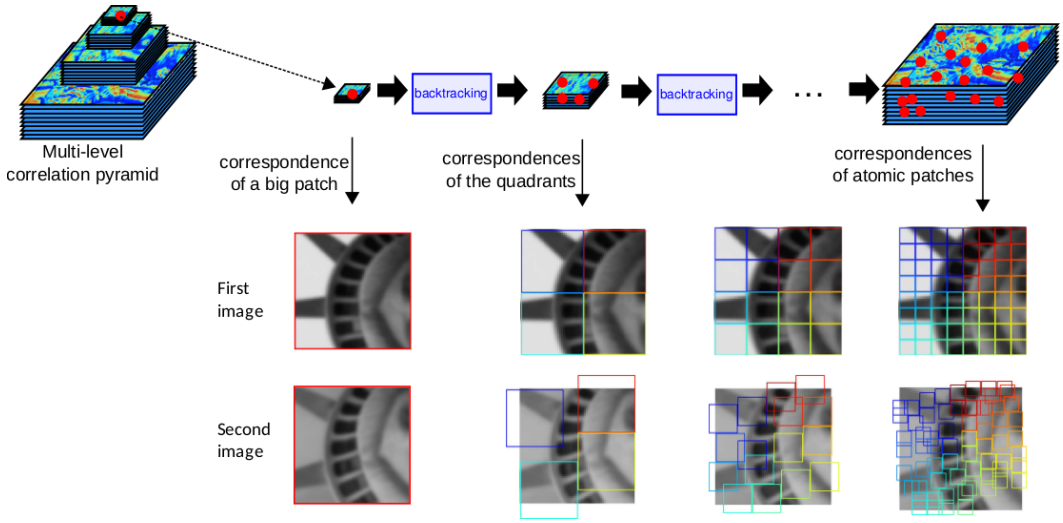

Fig. 7 Backtracking atomic correspondences from an entry point (red dot) in the top pyramid level *(left)*. At each level, the backtracking consists in undoing the aggregation performed previously in order to recover the position of the four children patches in the lower level. When the bottom level is reached, we obtain a set of correspondences for atomic patches *(right)*.

Fig. 8 Matching result between two images with repetitive textures. Nearly all output correspondences are correct. Wrong matches are due to occluded areas (bottom-right of the first image) or situations where the deformation tolerance of DeepMatching is exceeded (bottom-left of the first image).

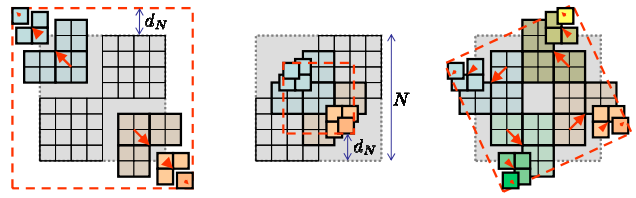

Fig. 9 Extent of the tolerance of DeepMatching to deformations. From left to right: up-scale of 1.5x, down-scale of 0.5x, rotation of 26°. The plain gray (resp. dashed red) square represents the patch in the reference (resp. target) image. For clarity, only the corner pixels are maximally deformed.

Note that the displacement tolerance in Θ_i from Eq. (6) could be extended to $x \times N/8$ pixels with $x \in \{2, 3, \ldots\}$ (instead of $x = 1$). Then the above formula for computing the lower bound on the scale factor of DeepMatching generalizes to $\mathrm{LB}(x) = \lim_{N\to\infty} (N - 2xd_N)/N$. Hence, for $x \geq 2$ we obtain $\mathrm{LB}(x) < 0$ instead of

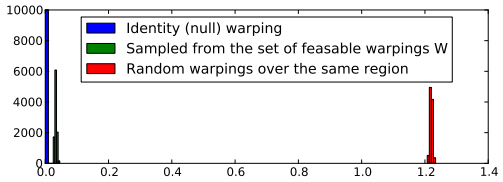

Fig. 10 Histogram over smoothness for identity warping, warping respecting the built-in constraints in DeepMatching and random warping. The x-axis indicates the smoothness value. The smoothness value is low when there are few discontinuities, *i.e.*, the warpings are smooth. The histogram is obtained with 10,000 different artificial warpings. See text for details.

$\mathrm{LB}(1) = \frac{1}{2}$. This implies that the deformation range is extended to a point where any patch can be matched to a single pixel, *i.e.*, this results in unrealistic deformations. For this reason, we choose to not expand the deformation range of DeepMatching.

Built-in smoothing. Furthermore, correspondences generated through backtracking of a single entry point in the correlation maps are naturally smooth. Indeed, feasible deformations cannot be too "far" from the identity deformation. To verify this assumption, we conduct the following experiment. We artificially generate two types of correspondences between two images of size 128×128. The first one is completely random, *i.e.* for each atomic patch in the first image we assign randomly a match in the second image. The second one respects the backtracking constraints. Starting from a single entry point in the top level we simulate the backtracking procedure from Section 3.3 by replacing in Eq. (13) the max operation by a random sampling over $\{-1, 0, 1\}^2$. By generating 10,000 sets of possible atomic correspondences, we

simulate a set which respects the deformations allowed by DeepMatching. Figure 10 compares the smoothness of these two types of artificial correspondences. Smoothness is measured by interpreting the correspondences as flow and measuring the gradient flow norm, see Eq. (19). Clearly, the two types of warpings are different by orders of magnitude. Furthermore, the one which respects the built-in constraints of DeepMatching is close to the identity warping.

Relation to Deep Convolutional Neural Networks (CNNs). DeepMatching relies on a hierarchical, multi-layer, correlational architecture designed for matching images and was inspired by deep convolutional approaches (LeCun et al 1998a). In the following we describe the major similarities and differences.

Deep networks learn from data the weights of the convolutions. In contrast, DeepMatching does not learn any feature representations and instead directly computes correlations at the patch level. It uses patches from the first image as convolution filters for the second one. However, the bottom-up pipeline of DeepMatching is similar to CNNs. It alternates aggregating channels from the previous layer with channel-wise max-pooling and subsampling. As in CNNs, max-pooling in DeepMatching allows for invariance *w.r.t.* small deformations. Likewise, the algorithm propagates pairwise patch similarity scores through the hierarchy using non-linear rectifying stages in-between layers. Finally, DeepMatching includes a top-down pass which is not present in CNNs.

Time and space complexity. DeepMatching has a complexity $O(LL')$ in memory and time, where $L = WH$ and $L' = W'H'$ are the number of pixels per image.

Proof Computing the initial correlations is a $O(LL')$ operation. Then, at each level of the pyramid, the process is repeated while the complexity is divided by a factor 4 due to the subsampling step in the target image (since the cardinality of $|\{\mathcal{G}_N\}|$ remains approximately constant). Thus, the total complexity of the correlation maps computation is, at worst, $O(\sum_{n=0}^{\infty} LL'/4^n) = O(LL')$. During the top-down pass, most backtracking paths can be pruned as soon as they cross a concurrent path with a higher score (see Section 3.3). Thus, all correlations will be examined at most once, and there are $\sum_{n=0}^{\infty} LL'/4^n$ values in total. However, this analysis is worst-case. In practice, only correlations lying on maximal paths are actually examined. □

4 Extensions of DeepMatching

4.1 Approximate DeepMatching

As a consequence of its $O(LL')$ space complexity, DeepMatching requires an amount of RAM that is orders of magnitude above other state-of-the-art matching methods. This could correspond to several gigabytes for images of moderate size (800×600 pixels); see Section 5.2.3. This section introduces an approximation of DeepMatching that allows to trade matching quality for reduced time and memory usage. As shown in Section 5.2.3, near-optimal results can be obtained at a fraction of the original cost.

Our approximation proposes to compress the representation of atomic patches $\{I_{4,\boldsymbol{p}}\}$. Atomic patches carry little information, and thus are highly redundant. For instance, in uniform regions, all patches are nearly identical (*i.e.*, gradient-wise). To exploit this property, we index atomic patches with a small set of patch prototypes. We substitute each patch with its closest neighbor in a fixed dictionary of D prototypes. Hence, we need to perform and store only D convolutions at the first level, instead of $O(L)$ (with $D \ll O(L)$). This significantly reduces both memory and time complexity. Note that higher pyramid levels also benefit from this optimization. Indeed, two parent patches at the second level have the exact same correlation map in case their children are assigned the same prototypes. The same reasoning also holds for all subsequent levels, but the gains rapidly diminish due to statistical unlikeliness of the required condition. This is not really an issue, since the memory and computational cost mostly rests on the initial levels; see Section 3.4.

In practice, we build the prototype dictionary using k-means, as it is designed to minimize the approximation error between input descriptors and resulting centroids (*i.e.* prototypes). Given a pair of images to match, we perform on-line clustering of all descriptors of atomic patches $\{I_{4,\boldsymbol{p}}\} = \{\mathbf{R}\}$ in the first image. Since the original descriptors lie on an hypersphere (each pixel descriptor $\mathbf{R}_{i,j}$ has norm 1), we modify the k-means approach so as to project the estimated centroids on the hypersphere at each iteration. We find experimentally that this is important to obtain good results.

4.2 Scale and rotation invariant DeepMatching

For a variety of tasks, objects to be matched can appear under image rotations or at different scales (Lowe 2004; Mikolajczyk et al 2005; Szeliski 2010; HaCohen et al 2011). As discussed above, DeepMatching (DM) is only

robust to moderate scale changes and rotations. We now present a scale and rotation invariant version.

The approach is straightforward: we apply DM to several rotated and scaled versions of the second image. According to the invariance range of DM, we use steps of $\pi/4$ for image rotation and power of $\sqrt{2}$ for scale changes. While iterating over all combinations of scale changes and rotations, we maintain a list $\mathcal{M}'$ of all atomic correspondences obtained so far, *i.e.* corresponding positions and scores. As before, the final output correspondences consists of the reciprocal matches in $\mathcal{M}'$. Storing all matches and finally choosing the best ones based on reciprocal verification permits to capture distinct motions possibly occurring together in the same scene (*e.g.* one object could have undergone a rotation, while the rest of the scene did not move). The steps of the approach are described in Algorithm 2.

Since we iterate sequentially over a fixed list of rotations and scale changes, the space and time complexity of the algorithm remains unchanged (*i.e.* $O(LL')$). In practice, the run-time compared to DM is multiplied by a constant approximately equal to 25, see Section 5.2.4. Note that the algorithm permits a straightforward parallelization.

Algorithm 2 Scale and rotation invariant version of DeepMatching (DM). I_σ denotes the image I downsized by a factor σ, and $\mathcal{R}_\theta$ denotes rotation by an angle θ.

Input: I, I' are the images to be matched
Initialize an empty set $\mathcal{M}' = \{\}$ of correspondences
For $\sigma \in \{-2, -1.5 \dots, 1.5, 2\}$ **do**
  $\sigma_1 \leftarrow \max\left(1, 2^{+\sigma}\right)$ # either downsize image 1
  $\sigma_2 \leftarrow \max\left(1, 2^{-\sigma}\right)$ # or downsize image 2
  For $\theta \in \left\{0, \frac{\pi}{4} \dots, \frac{7\pi}{4}\right\}$ **do**
    # get raw atomic correspondences (Eq. (15))
    $\mathcal{M}'_{\sigma,\theta} \leftarrow \text{DeepMatching}\left(I_{\sigma_1}, \mathcal{R}_{-\theta} * I'_{\sigma_2}\right)$
    # Geometric rectification to the input image space:
    $\mathcal{M}'^{R}_{\sigma,\theta} \leftarrow \left\{(\sigma_1 \boldsymbol{p},\, \sigma_2 \mathcal{R}_\theta \boldsymbol{p}',\, s) \mid \forall (\boldsymbol{p}, \boldsymbol{p}', s) \in \mathcal{M}'_{\sigma,\theta}\right\}$
    # Concatenate results:
    $\mathcal{M}' \leftarrow \mathcal{M}' \bigcup \mathcal{M}'^{R}_{\sigma,\theta}$
$\mathcal{M}'' \leftarrow \text{reciprocal}(\mathcal{M}')$ # keep reciprocal matches (Eq. (16))
Return $\mathcal{M}''$

4.3 DeepFlow

We now present our approach for optical flow estimation, DeepFlow. We adopt the method introduced by Brox and Malik (2011), where a matching term penalizes the differences between optical flow and input matches, and replace their matching approach by DeepMatching. In addition, we make a few minor modifications introduced recently in the state of the art: (i) we add a normalization in the data term to downweight the impact of locations with high spatial image derivatives (Zimmer et al 2011); (ii) we use a different weight at each level to downweight the matching term at finer scales (Stoll et al 2012); and (iii) the smoothness term is locally weighted (Xu et al 2012).

Let $I_1, I_2 : \Omega \to \mathbb{R}^c$ be two consecutive images defined on Ω with c channels. The goal is to estimate the flow $\boldsymbol{w} = (u, v)^\top : \Omega \to \mathbb{R}^2$. We assume that the images are already smoothed using a Gaussian filter of standard deviation σ. The energy we optimize is a weighted sum of a data term E_D, a smoothness term E_S and a matching term E_M:

$$E(\boldsymbol{w}) = \int_\Omega E_D + \alpha E_S + \beta E_M \boldsymbol{dx} \tag{17}$$

For the three terms, we use a robust penalizer $\Psi(s^2) = \sqrt{s^2 + \epsilon^2}$ with $\epsilon = 0.001$ which has shown excellent results (Sun et al 2014b).

Data term. The data term is a separate penalization of the color and gradient constancy assumptions with a normalization factor as proposed by Zimmer et al (2011). We start from the optical flow constraint assuming brightness constancy: $(\nabla_3^\top I)\boldsymbol{w} = 0$ with $\nabla_3 = (\partial x, \partial y, \partial t)^\top$ the spatio-temporal gradient. A basic way to build a data term is to penalize it, *i.e.* $E_D = \Psi(\boldsymbol{w}^\top \boldsymbol{J}_0 \boldsymbol{w})$ with $\boldsymbol{J}_0$ the tensor defined by $\boldsymbol{J}_0 = (\nabla_3 I)(\nabla_3^\top I)$. As highlighted by Zimmer et al (2011), such a data term adds a higher weight in locations corresponding to high spatial image derivatives. We normalize it by the norm of the spatial derivatives plus a small factor to avoid division by zero, and to reduce a bit the influence in tiny gradient locations (Zimmer et al 2011). Let $\bar{\boldsymbol{J}}_0$ be the normalized tensor $\bar{\boldsymbol{J}}_0 = \theta_0 \boldsymbol{J}_0$ with $\theta_0 = (\|\nabla_2 I\|^2 + \zeta^2)^{-1}$. We set $\zeta = 0.1$ in the following. To deal with color images, we consider the tensor defined for a channel i denoted by upper indices $\bar{\boldsymbol{J}}_0^i$ and we penalize the sum over channels: $\Psi(\sum_{i=1}^c \boldsymbol{w}^\top \bar{\boldsymbol{J}}_0^i \boldsymbol{w})$. We consider images in the RGB color space.

We separately penalize the gradient constancy assumption (Bruhn et al 2005). Let I_x and I_y be the derivatives of the images with respect to the x and y axis respectively. Let $\bar{\boldsymbol{J}}_{xy}^i$ be the tensor for the channel i including the normalization

$$\bar{\boldsymbol{J}}_{xy}^i = (\nabla_3 I_x^i)(\nabla_3^\top I_x^i)/(\|\nabla_2 I_x^i\|^2 + \zeta^2) + (\nabla_3 I_y^i)(\nabla_3^\top I_y^i)/(\|\nabla_2 I_y^i\|^2 + \zeta^2).$$

The data term is the sum of two terms, balanced by two weights δ and γ:

$$E_D = \delta \Psi\left(\sum_{i=1}^c \boldsymbol{w}^\top \bar{\boldsymbol{J}}_0^i \boldsymbol{w}\right) + \gamma \Psi\left(\sum_{i=1}^c \boldsymbol{w}^\top \bar{\boldsymbol{J}}_{xy}^i \boldsymbol{w}\right) \tag{18}$$

Smoothness term. The smoothness term is a robust penalization of the gradient flow norm:

$$E_S = \Psi\left(\|\nabla u\|^2 + \|\nabla v\|^2\right) \quad . \tag{19}$$

The smoothness weight α is locally set according to image derivatives (Wedel et al 2009; Xu et al 2012) with $\alpha(\boldsymbol{x}) = \exp(-\kappa\nabla_2 I(\boldsymbol{x}))$ where κ is experimentally set to $\kappa = 5$.

Matching term. The matching term encourages the flow estimation to be similar to a precomputed vector field $\boldsymbol{w}'$. To this end, we penalize the difference between $\boldsymbol{w}$ and $\boldsymbol{w}'$ using the robust penalizer Ψ. Since the matching is not totally dense, we add a binary term $c(\boldsymbol{x})$ which is equal to 1 if and only if a match is available at $\boldsymbol{x}$.

We also multiply each matching penalization by a weight $\phi(\boldsymbol{x})$, which is low in uniform regions where matching is ambiguous and when matched patches are dissimilar. To that aim, we rely on $\tilde{\lambda}(\boldsymbol{x})$, the minimum eigenvalue of the autocorrelation matrix multiplied by 10. We also compute the visual similarity between matches as $\Delta(\boldsymbol{x}) = \sum_{i=1}^{c} |I_1^i(\boldsymbol{x}) - I_2^i(\boldsymbol{x}-\boldsymbol{w}'(\boldsymbol{x}))| + |\nabla I_1^i(\boldsymbol{x}) - \nabla I_2^i(\boldsymbol{x}-\boldsymbol{w}'(\boldsymbol{x}))|$. We then compute the score ϕ as a Gaussian kernel on Δ weighted by $\tilde{\lambda}$ with a parameter σ_M, experimentally set to $\sigma_M = 50$. More precisely, we define $\phi(\boldsymbol{x})$ at each point $\boldsymbol{x}$ with a match $\boldsymbol{w}'(\boldsymbol{x})$ as:

$$\phi(\boldsymbol{x}) = \sqrt{\tilde{\lambda}(\boldsymbol{x})}/(\sigma_M\sqrt{2\pi})\ \exp(-\Delta(\boldsymbol{x})/2\sigma_M).$$

The matching term is then $E_M = c\phi\Psi(\|\boldsymbol{w} - \boldsymbol{w}'\|^2)$.

Minimization. This energy objective is non-convex and non-linear. To solve it, we use a numerical optimization algorithm similar as Brox et al (2004). An incremental coarse-to-fine warping strategy is used with a downsampling factor $\eta = 0.95$. The remaining equations are still non-linear due to the robust penalizers. We apply 5 inner fixed point iterations where the non-linear weights and the flow increments are iteratively updated while fixing the other. To approximate the solution of the linear system, we use 25 iterations of the Successive Over Relaxation (SOR) method (Young and Rheinboldt 1971).

To downweight the matching term on fine scales, we use a different weight β^k at each level as proposed by Stoll et al (2012). We set $\beta^k = \beta(k/k_{\max})^b$ where k is the current level of computation, $k_{\max}$ the coarsest level and b a parameter which is optimized together with the other parameters, see Section 5.3.1.

5 Experiments

This section presents an experimental evaluation of DeepMatching and DeepFlow. The datasets and metrics used to evaluate DeepMatching and DeepFlow are introduced in Section 5.1. Experimental results are given in Sections 5.2 and 5.3 respectively.

5.1 Datasets and metrics

In this section we briefly introduce the matching and flow datasets used in our experiments. Since consecutive frames of a video are well-suited to evaluate a matching approach, we use several optical flow datasets for evaluating both the quality of matching and flow, but we rely on different metrics.

The Mikolajczyk dataset was originally proposed by Mikolajczyk et al (2005) to evaluate and compare the performance of keypoint detectors and descriptors. It is one of the standard benchmarks for evaluating matching approaches. The dataset consists of 8 sequences of 6 images each viewing a scene under different conditions, such as illumination changes or viewpoint changes. The images of a sequence are related by homographies. During the evaluation, we comply to the standard procedure in which the first image of each scene is matched to the 5 remaining ones. Since our goal is to study robustness of DeepMatching to geometric distortions, we follow HaCohen et al (2011) and restrict our evaluation to the 4 most difficult sequences with viewpoint changes: *bark*, *boat*, *graf* and *wall*.

The MPI-Sintel dataset (Butler et al 2012) is a challenging evaluation benchmark for optical flow estimation, constructed from realistic computer-animated films. The dataset contains sequences with large motions and specular reflections. In the training set, more than 17.5% of the pixels have a motion over 20 pixels, approximately 10% over 40 pixels. We use the "final" version, featuring rendering effects such as motion blur, defocus blur and atmospheric effects. Note that ground-truth optical flows for the test set are not publicly available.

The Middlebury dataset (Baker et al 2011) has been extensively used for evaluating optical flow methods. The dataset contains complex motions, but most of the motions are small. Less than 3% of the pixels have a motion over 20 pixels, and no motion exceeds 25 pixels (training set). Ground-truth optical flows for the test set are not publicly available.

The Kitti dataset Geiger et al (2013) contains real-world sequences taken from a driving platform. The dataset includes non-Lambertian surfaces, different lighting conditions, a large variety of materials and large displacements. More than 16% of the pixels have motion over 20 pixels. Again, ground-truth optical flows for the test set are not publicly available.

Performance metric for matching. Choosing a performance measure for matching approaches is delicate. Matching approaches typically do not return dense correspondences, but output varying numbers of matches. Furthermore, correspondences might be concentrated in different areas of the image.

Most matching approaches, including DeepMatching, are based on establishing correspondences between patches. Given a pair of matching patches, it is possible to obtain a list of pixel correspondences for all pixels within the patches. We introduce a measure based on the number of correctly matched *pixels* compared to the overall number of pixels. We define "accuracy@T" as the proportion of "correct" pixels from the first image with respect to the total number of pixels. A pixel is considered correct if its pixel match in the second image is closer than T pixels to ground-truth. In practice, we use a threshold of $T = 10$ pixels, as this represents a sufficiently precise estimation (about 1% of image diagonal for all datasets), while allowing some tolerance in blurred areas that are difficult to match exactly. If a pixel belongs to several matches, we choose the one with the highest score to predict its correspondence. Pixels which do not belong to any patch have an infinite error.

Performance metric for optical flow. To evaluate optical flow, we follow the standard protocol and measure the average endpoint error over all pixels, denoted as "EPE". The "s10-40" variant measures the EPE only for pixels with a ground-truth displacement between 10 and 40 pixels, and likewise for "s0-10" and "s40+". In all cases, scores are averaged over all image pairs to yield the final result for a given dataset.

5.2 Matching Experiments

In this section, we evaluate DeepMatching (DM). We present results for all datasets presented above but Middlebury, which does not feature long-range motions, the main difficulty in image matching. When evaluating on the Mikolajczyk dataset, we employ the scale and rotation invariant version of DM presented in Section 4.2. For all the matching experiments reported in this section, we use the Mikolajczyk dataset and the training sets of MPI-Sintel and Kitti.

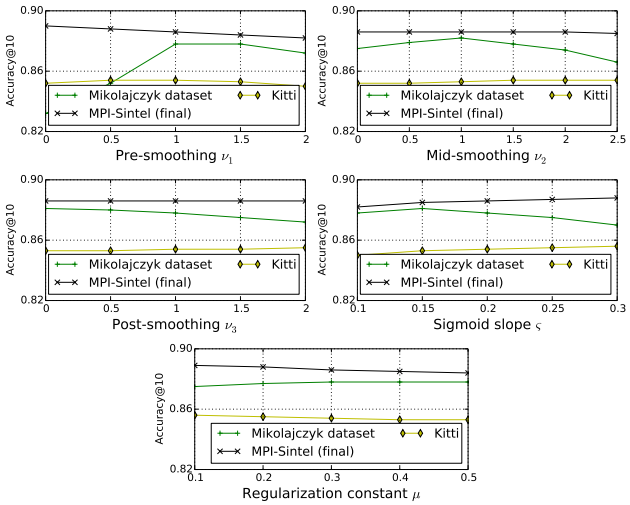

Fig. 11 Impact of the parameters to compute pixel descriptors on the different datasets.

5.2.1 Impact of the parameters

We optimize the different parameters of DM jointly on all datasets. To prevent overfitting, we use the same parameters across all datasets.

Pixel descriptor parameters: We first optimize the parameters of the pixel representation (Section 3.2): ν_1, ν_2, ν_3 (different smoothing stages), ς (sigmoid slope) and μ (regularization constant). After performing a grid search, we find that good results are obtained at $\nu_1 = \nu_2 = \nu_3 = 1$, $\varsigma = 0.2$ and $\mu = 0.3$ across all datasets. Figure 11 shows the accuracy@10 in the neighborhood of these values for all parameters. Image pre-smoothing seems to be crucial for JPEG images (Mikolajczyk dataset), as it smooths out compression artifacts, whereas it slightly degrades performance for uncompressed PNG images (MPI-Sintel and Kitti). As expected, similar findings are observed for the regularization constant μ since it acts as a regularizer that reduces the impact of small gradients (*i.e.* noise). In the following, we thus use low values of ν_1 and μ when dealing with PNG images (we set $\nu_1 = 0$ and $\mu = 0.1$, other parameters are unchanged).

Non-linear rectification: We also evaluate the impact of the parameter λ of the non-linear rectification obtained by applying power normalization, see Eq. (11). Figure 12 displays the accuracy@10 for various values of λ. We can observe that the optimal performance is achieved at $\lambda = 1.4$ for all datasets. We use this value in the remainder of our experiments.

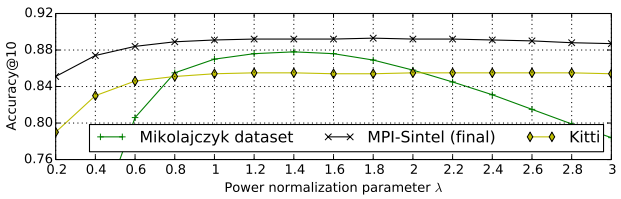

Fig. 12 Impact of the non-linear response rectification (eq. (11)).

R	New BT entry points	New scoring scheme	accuracy@10	memory usage	matching time
Mikolajczyk dataset					
1/4			0.620	0.9 GB	1.0 min
1/2			0.848	5.5 GB	20 min
1/2		✓	0.864	5.5 GB	7.3 min
1/2	✓	✓	**0.878**	4.4 GB	6.3 min
MPI-Sintel dataset (final)					
1/4			0.822	0.4 GB	2.4 sec
1/2			0.880	6.3 GB	55 sec
1/2		✓	0.890	6.3 GB	16 sec
1/2	✓	✓	**0.892**	4.6 GB	16 sec
Kitti dataset					
1/4			0.772	0.4 GB	2.0 sec
1/2			0.841	6.3 GB	39 sec
1/2		✓	0.855	6.3 GB	14 sec
1/2	✓	✓	**0.856**	4.7 GB	14 sec

Table 1 Detailed comparison between the preliminary and current versions of DeepMatching in terms of performance, run-time and memory usage. R denotes the input image resolution and BT backtracking. Run-times are computed on 1 core @ 3.6 GHz.

5.2.2 Evaluation of the backtracking and scoring schemes

We now evaluate two improvements of DM with respect to the previous version published in Weinzaepfel et al (2013), referred to as DM*:

- Backtracking (BT) entry points: in DM* we select as entry points local maxima in the correlation maps from all pyramid levels. The new alternative is to start from all possible points in the top pyramid level.
- Scoring scheme: In DM* we scored atomic correspondences based on the correlation values of start and end point of the backtracking path. The new scoring scheme is the sum of correlation values along the full backtracking path.

We report results for the different variants in Table 1 on each dataset. The first two rows for each dataset correspond to the exact settings used for DM* (*i.e.* with an image resolution of $1/4$ and $1/2$). We observe a steady increase in performance on all datasets when we add the new scoring and backtracking approach. We can observe that starting from all possible entry points in the top pyramid level (*i.e.* considering all possible translations) yields slightly better results than starting from local maxima. This demonstrates that some ground-truth matches are not covered by any local maximum. By enumerating all possible patch translations from the top-level, we instead ensure to fully explore the space of all possible matches.

Furthermore, it is interesting to note that memory usage and run-time significantly decreases when using the new options. This is because (1) searching and storing local maxima (which are exponentially more numerous in lower pyramid levels) is not necessary anymore, and (2) the new scoring scheme allows for further optimization, *i.e.* early pruning of backtracking paths (Section 3.3).

5.2.3 Approximate DeepMatching

We now evaluate the performance of approximate DeepMatching (Section 4.1) and report its run-time and memory usage. We evaluate and compare two different ways of reducing the computational load. The first one simply consists in downsizing the input images, and upscaling the resulting matches accordingly. The second option is the compression scheme proposed in Section 4.1.

We evaluate both schemes jointly by varying the input image size (expressed as a fraction R of the original resolution) and the size D of the prototype dictionary (*i.e.* parameter of k-means in Section 4.1). $R = 1$ corresponds to the original dataset image size (no downsizing). We display the results in terms of matching accuracy (accuracy@10) against memory consumption in Figure 13 and as a function of D in Figure 14. Figure 13 shows that DeepMatching can be computed in an approximate manner for any given memory budget. Unsurprisingly, too low settings (*e.g.* $R \leq 1/8$, $D \leq 64$) result in a strong loss of performance. It should be noted that that we were unable to compute DeepMatching at full resolution ($R = 1$) for $D > 64$, as the memory consumption explodes. As a consequence, all subsequent experiments in the paper are done at $R = 1/2$. In Figure 14, we observe that good trades-off are achieved for dictionary sizes comprised in $D \in [64, 1024]$. For instance, on MPI-Sintel, at $D = 1024$, 94% of the performance of the uncompressed case ($D = \infty$) is reached for half the computation time and one third of the memory usage. Detailed timings of the different stages of DeepMatching are given in Table 2. As expected, only the bottom-up pass is affected by the approximation, with a run-time of the different operations involved (patch correlations, max-pooling, subsampling, aggregation and non-linear rectification) roughly proportional to D (or to $|\mathcal{G}_4|$, the actual number of atomic patches, if $D = \infty$). The overhead of clustering the dictionary prototypes with k-means appears negligible, with the exception of the largest dictionary size ($D = 4096$) for which it in-

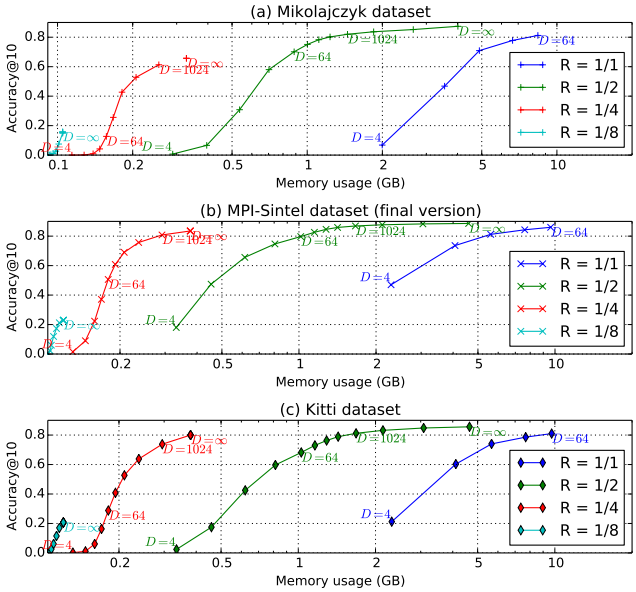

Fig. 13 Trade-off between memory consumption and matching performance for the different datasets. Memory usage is controlled by changing image resolution R (different curves) and dictionary size D (curve points).

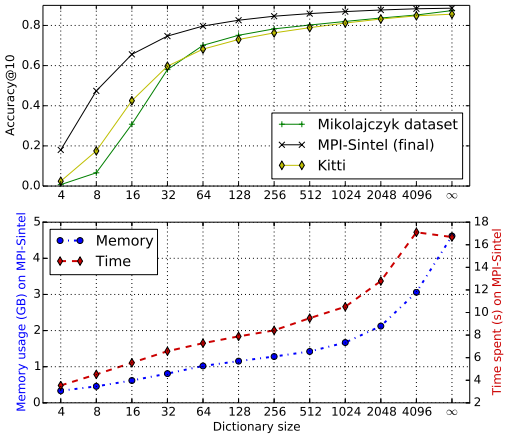

Fig. 14 Performance, memory usage and run-time for different levels of compression corresponding to the size D of the prototype dictionary (we set the image resolution to $R = 1/2$). A dictionary size $D = \infty$ stands for no compression. Run-times are for a single image pair on 1 core @ 3.6 GHz.

duces a slightly longer run-time than in the uncompressed case. Overall, the proposed method for approximating DeepMatching is highly effective.

GPU Implementation. We have implemented DM on GPU in the `Caffe` framework (Jia et al 2014). Using existing `Caffe` layers like ConvolutionLayer and PoolingLayer, the implementation is straightforward for most layers. We had to specifically code a few layers which are not available in `Caffe` (*e.g.* the backtracking pass[3]). For the aggregation layer which consists in selecting and averaging 4 children channels out of many channels, we relied on the sparse matrix multiplication in the cuSPARSE toolbox. Detailed timings are given in Table 2 on a GeForce Titan X. Our code runs in about 0.2s for a pair of MPI-Sintel image. As expected, the computation bottleneck essentially lies in the computation of bottom-level patch correlations and the backtracking pass. Note that computing patch descriptors takes significantly more time, in proportion, than on CPU: it takes about 0.024s = 11% of total time (not shown in table). This is because it involves a succession of many small layers (image smoothing, gradient extraction and projection, *etc.*), which causes overhead and is rather inefficient.

[3] Although the backtracking is conceptually close to the back-propagation training algorithm, it differs in term of how the scores are accumulated for each path.

Proc. Unit	R	D	Patch clustering	Patch Correlations	Max-pooling +subsampling	Aggregation	Non-linear rectification	Backtracking	Total time
CPU	1/2	64	0.3	0.2	0.4	0.9	0.8	5.1	7.7
CPU	1/2	1024	1.3	0.7	0.6	1.0	1.3	5.8	10.7
CPU	1/2	∞	-	4.3	1.6	1.0	3.2	6.2	16.4
GPU	1/2	∞	-	0.084	0.012	0.017	0.013	0.053	0.213

Table 2 Detailed timings of the different stages of DeepMatching, measured for a single image pair from MPI-Sintel on CPU (1 core @ 3.6GHz) and GPU (GeForce Titan X) in seconds. Stages are: patch clustering (only for approximate DM, see Section 4.1), patch correlations (Eq. (4)), joint max-pooling and subsampling, correlation map aggregation, non linear rectification (resp. $\mathcal{S} \circ \mathcal{P}$, $\sum \mathcal{T}_{o_i}$, and $\mathcal{R}_\lambda$ in Eq. (12)), and correspondence backtracking (Section 3.3). Other operations (*e.g.* reciprocal verification of Eq. (16)) have negligible run-time. For operations applied at several levels like the non-linear rectification, a cumulative timing is given.

5.2.4 Comparison to the state of the art

We compare DM with several baselines and state-of-the-art matching algorithms, namely:

- SIFT keypoints extracted with DoG detector (Lowe 2004) and matched with FLANN (Muja and Lowe 2009), referred to as SIFT-NN,[4]
- dense HOG matching, followed by nearest-neighbor matching with reciprocal verification as done in LDOF (Brox and Malik 2011), referred to as HOG-NN[4],
- Generalized PatchMatch (GPM) (Barnes et al 2010), with default parameters, 32x32 patches and 20 iterations (best settings in our experiments)[5],
- Kd-tree PatchMatch (KPM) (Sun 2012), an improved version of PatchMatch based on better patch descriptors and kd-trees optimized for correspondence propagation[4],

[4] We implemented this method ourselves.

[5] We used the online code.

– Non-Rigid Dense Correspondences (NRDC) (HaCohen et al 2011), an improved version of GPM based on a multiscale iterative expansion/contraction strategy[6],
– SIFT-flow (Liu et al 2011), a dense matching algorithm based on an energy minimization where pixels are represented as SIFT features and a smoothness term is incorporated to explicitly preserve spatial discontinuities[5],
– Scale-less SIFT (SLS) (Hassner et al 2012), an improvement of SIFT-flow to handle scale changes (multiple sized SIFTs are extracted and combined to form a scale-invariant pixel representation)[5],
– DaisyFilterFlow (DaisyFF) (Yang et al 2014), a dense matching approach that combines filter-based efficient flow inference and the Patch-Match fast search algorithm to match pixels described using the DAISY representation (Tola et al 2010)[5],
– Deformable Pyramid Matching (DSP) (Kim et al 2013), a dense matching approach based on a coarse-to-fine (top-down) strategy where inference is performed with (inexact) loopy belief propagation[5].

SIFT-NN, HOG-NN and DM output sparse matches, whereas the other methods output fully dense correspondence fields. SIFT keypoints, GPM, NRDC and DaisyFF are scale and rotation invariant, whereas HOG-NN,
KPM, SIFT-flow, SLS and DSP are not. We, therefore, do not report results for these latter methods on the Mikolajczyk dataset which includes image rotations and scale changes.

Statistics about each method (average number of matches per image and their coverage) are reported in Table 3. Coverage is computed as the proportion of points on a regular grid with 10 pixel spacing for which there exists a correspondence (in the raw output of the considered method) within a 10 pixel neighborhood. Thus, it measures how well matches "cover" the image. Table 3 shows that DeepMatching outputs 2 to 7 times more matches than SIFT-NN and a comparable number to HOG-NN. Yet, the coverage for DM matches is much higher than for HOG-NN and SIFT-NN. This shows that DM matches are well distributed over the entire image, which is not the case for HOG-NN and SIFT-NN, as they have difficulties estimating matches in regions with weak or repetitive textures.

Quantitative results are listed in Table 4, and qualitative results in Figures 15, 16 and 17. Overall, DM significantly outperforms all other methods, even when reduced settings are used (*e.g.* for image resolution $R = 1/2$ and $D = 1024$ prototypes). As expected, SIFT-NN performs rather well in presence of global image transformation (Mikolajczyk dataset), but yields the worst result for the case of more complex motions (flow datasets). Figures 16 and 17 illustrate the reason: SIFT's large patches are way too coarse to follow motion boundaries precisely. The same issue also holds for HOG-NN. Methods predicting dense correspondence fields return

[6] We report results from the original paper.

Method	Mikolajczyk		MPI-Sintel (final)		Kitti	
	#	coverage	#	coverage	#	coverage
SIFT-NN	2084	0.59	836	0.25	1299	0.38
HOG-NN	-	-	4576	0.39	4293	0.34
KPM	-	-	446K	1	462K	1
GPM	545K	1	446K	1	462K	1
NRDC	545K	1	446K	1	462K	1
SIFT-flow	-	-	446K	1	462K	1
SLS	-	-	446K	1	462K	1
DaisyFF	545K	1	446K	1	462K	1
DSP	-	-	446K	1	462K	1
DM (ours)	3120	0.81	5920	0.96	5357	0.88

Table 3 Statistics of the different matching methods. The "#" column refers to the average number of matches per image, and the coverage to the proportion of points on a regular grid with 10 pixel spacing that have a match within a 10px neighborhood.

method	R	D	accuracy@10	memory usage	matching time
Mikolajczyk dataset					
SIFT-NN			0.674	0.2 GB	1.4 sec
GPM			0.303	0.1 GB	2.4 min
NRDC			0.692	0.1 GB	2.5 min
DaisyFF			0.410	6.1 GB	16 min
DM	1/4	∞	0.657	0.9 GB	38 sec
DM	1/2	1024	0.820	1.5 GB	4.5 min
DM	1/2	∞	**0.878**	4.4 GB	6.3 min
MPI-Sintel dataset (final)					
SIFT-NN			0.684	0.2 GB	2.7 sec
HOG-NN			0.712	3.4 GB	32 sec
KPM			0.738	0.3 GB	7.3 sec
GPM			0.812	0.1 GB	1.1 min
SIFT-flow			0.890	1.0 GB	29 sec
SLS			0.824	4.3 GB	16 min
DaisyFF			0.873	6.8 GB	12 min
DSP			0.853	0.8 GB	39 sec
DM	1/4	∞	0.835	0.3 GB	1.6 sec
DM	1/2	1024	0.869	1.8 GB	10 sec
DM	1/2	∞	**0.892**	4.6 GB	16 sec
Kitti dataset					
SIFT-NN			0.489	0.2 GB	1.7 sec
HOG-NN			0.537	2.9 GB	24 sec
KPM			0.536	0.3 GB	17 sec
GPM			0.661	0.1 GB	2.7 min
SIFT-flow			0.673	1.0 GB	25 sec
SLS			0.748	4.4 GB	17 min
DaisyFF			0.796	7.0 GB	11 min
DSP			0.580	0.8 GB	2.9 min
DM	1/4	∞	0.800	0.3 GB	1.6 sec
DM	1/2	1024	0.812	1.7 GB	10 sec
DM	1/2	∞	**0.856**	4.7 GB	14 sec

Table 4 Matching performance, run-time and memory usage for state-of-the-art methods and DeepMatching (DM). For the proposed method, R and D denote the input image resolution and the dictionary size (∞ stands for no compression). Run-times are computed on 1 core @ 3.6 GHz.

a more precise estimate, yet most of them (KPM, GPM, SIFT-flow, DSP) are not robust to repetitive textures in the Kitti dataset (Figure 17) as they rely on weakly discriminative small patches. Despite this limitation, SIFT-flow and DSP are still able to perform well on MPI-Sintel as this dataset contains little scale changes. Other dense methods, NRDC, SLS and DaisyFF, can handle patches of different sizes and thus perform better on Kitti. But in turn this is at the cost of reduced performance on the MPI-Sintel or Mikolajczyk datasets (qualitative results are in Figure 15). In conclusion, DM outperforms all other methods on the 3 datasets, including DSP which also relies on a hierarchical matching.

In terms of computing resources, DeepMatching with full settings ($R = 1/2$, $D = \infty$) is one of the most costly method (only SLS and DaisyFF require the same order of memory and longer run-time). The scale and rotation invariant version of DM, used for the Mikolajczyk dataset, is slow compared to most other approaches, due to its sequential processing (*i.e.* treating each combination of rotation and scaling sequentially), yet yields near perfect results. However, running DM with reduced settings is very competitive to the other approaches. On MPI-Sintel and Kitti, for instance, DM with a quarter resolution has a run-time comparable to the fastest method, SIFT-NN, with a reasonable memory usage, while still outperforming nearly all methods in terms of the accuracy@10 measure.

5.3 Optical Flow Experiments

We now present experimental results for the optical flow estimation. Optical flow is predicted using the variational framework presented in Section 4.3 that takes as input a set of matches. In the following, we evaluate the impact of DeepMatching against other matching methods, and compare to the state of the art.

5.3.1 Optimization of the parameters

We optimize the parameters of DeepFlow on a subset of the MPI-Sintel training set (20%), called "small" set, and report results on the remaining image pairs (80%, called "validation set") and on the training sets of Kitti and Middlebury. Ground-truth optical flows for the three test sets are not publicly available, in order to prevent parameter tuning on the test set.

We first optimize the different flow parameters (β, γ, δ, σ and b) by employing a gradient descent strategy with multiple initializations followed by a local grid search. For the data term, we find an optimum at $\delta = 0$, which is equivalent to removing the color constancy assumption. This can be explained by the fact that the "final" version contains atmospheric effects, reflections, blurs, etc. The remaining parameters are optimal at $\beta = 300$, $\gamma = 0.8$, $\sigma = 0.5$, $b = 0.6$. These parameters are used in the remaining of the experiments for DeepFlow, *i.e.* using matches obtained with DeepMatching, except when reporting results on Kitti and Middlebury test sets in Section 5.3.3. In this case the parameters are optimized on their respective training set.

Method	R	D	MPI-Sintel	Kitti	Middlebury
No Match			5.863	8.791	0.274
SIFT-NN			5.733	7.753	0.280
HOG-NN			5.458	8.071	**0.273**
KPM			5.560	15.289	0.275
GPM			5.561	17.491	0.286
SIFT-flow			5.243	12.778	0.283
SLS			5.307	10.366	0.288
DaisyFF			5.145	10.334	0.289
DSP			5.493	15.728	0.283
DM	1/2	1024	4.350	7.899	0.320
DM	1/2	∞	**4.098**	**4.407**	0.328

Table 5 Comparison of average endpoint error on different datasets when changing the input matches in the flow computation.

5.3.2 Impact of the matches on the flow

We examine the impact of different matching methods on the flow, *i.e.*, different matches are used in DeepFlow, see Section 4.3. For all matching approaches evaluated in the previous section, we use their output as matching term in Eq. (17). Because these approaches may output matches with statistics different from DM, we separately optimize the flow parameters for each matching approach on the small training set of MPI-Sintel[7].

Table 5 shows the endpoint error, averaged over all pixels. Clearly, a sufficiently dense and accurate matching like DM allows to considerably improve the flow estimation on datasets with large displacements (MPI-Sintel, Kitti). In contrast, none of the methods presented have a tangible effect on the Middlebury dataset, where the displacements are small.

The relatively small gains achieved by SIFT-NN and HOG-NN on MPI-Sintel and Kitti are due to the fact that a lot of regions with large displacements are not covered by any matches, such as the sky or the blurred character in the first and second column of Figure 18. Hence, SIFT-NN and HOG-NN have only a limited impact on the variational approach. On the other hand, the gains are also small (or even negative) for the dense

[7] Note that this systematically improves the endpoint error compared to using the raw dense correspondence fields as flow.

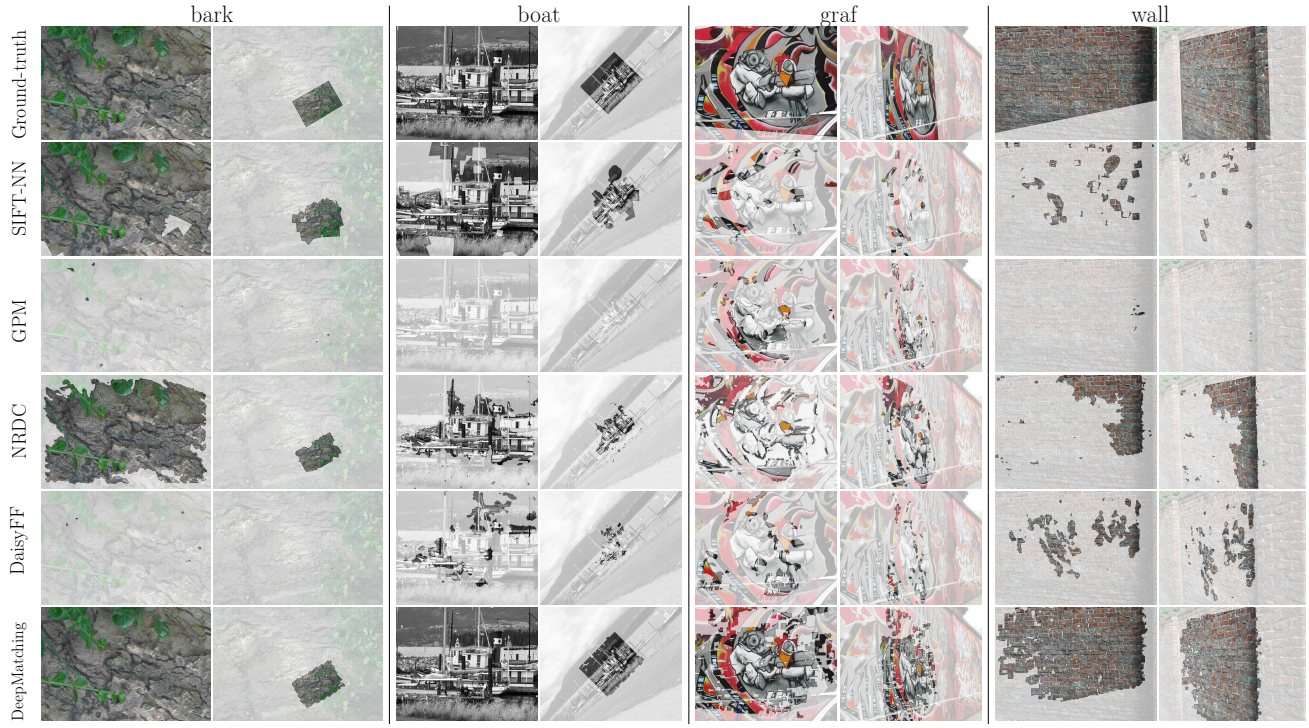

Fig. 15 Comparison of matching results of different methods on the Mikolajczyk dataset. Each row shows pixels with correct correspondences for different methods (from top to bottom: ground-truth, SIFT-NN, GPM, NRDC and DM). For each scene, we select two images to match and fade out regions which are unmatched, *i.e.* those for which the matching error is above 15px or can not be matched. DeepMatching outperforms the other methods, especially on difficult cases like *graf* and *wall*.

methods despite the fact that they output significantly more correspondences. We observe for these methods that the weight β of the matching term tends to be small after optimizing the parameters, thus indicating that the matches are found unreliable and noisy during training. The cause is clearly visible in Figure 17, where large portions containing repetitive textures (*e.g.* road, trees) are incorrectly matched. The poor quality of these matches even leads to a significant drop in performance on the Kitti dataset.

In contrast, DeepMatching generates accurate matches well covering the image that enable to boost the optical flow accuracy in case of large displacements. Namely, we observe a relative improvement of 30% on MPI-Sintel and of 50% on Kitti. It is interesting to observe that DM is able to effectively prune false matches arising in occluded areas (black areas in Figures 16 and 17). This is due to the reciprocal verification filtering incorporated in DM (Eq. (16)). When using the approximation with 1024 prototypes, however, a significant drop is observed on the Kitti dataset, while the performance remains good on MPI-Sintel. This indicates that approximating DeepMatching can result in a significant loss of robustness when matching repetitive textures, that are more frequent in Kitti than in MPI-Sintel.

5.3.3 Comparison to the state of the art

In this section, we compare DeepFlow to the state of the art on the test sets of MPI-Sintel, Kitti and Middlebury datasets. For theses datasets, the results are submitted to a dedicated server which performs the evaluation. Prior to submitting our results for Kitti and Middlebury test sets, we have optimized the parameters on the respective training set.

Results on MPI-Sintel. Table 6 compares our method to state-of-the-art algorithms on the MPI-Sintel test set. A comparison with the preliminary version of DeepFlow (Weinzaepfel et al 2013), referred to as DeepFlow*, is also provided. In this early version, we used a constant smoothness weight instead of a local one here (see Section 4.3) and used DM* as input matches. We can see that DeepFlow is among the best performing methods on MPI-Sintel, particularly for large displacements. This is due to the use of a reliable matching term in the variational approach, and this property is shared by all top performing approaches, *e.g.* (Revaud et al 2015; Leordeanu et al 2013). Furthermore, it is interesting to note that among the top performers on MPI-Sintel, 3 methods out of 6 actually employ DeepMatching. In particular, the top-3 method

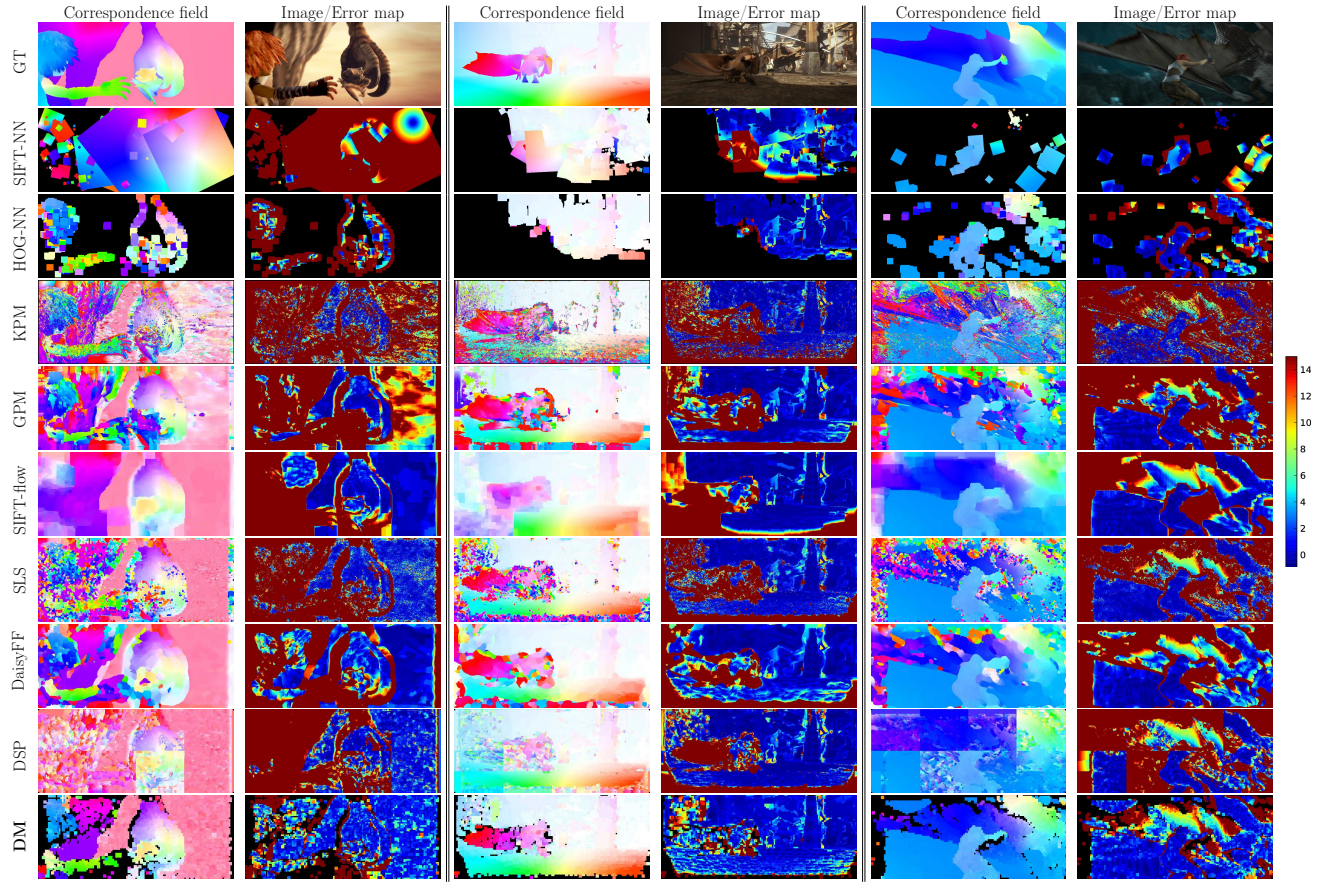

Fig. 16 Comparison of different matching methods on three challenging pairs with non-rigid deformations from MPI-Sintel. Each pair of columns shows motion maps (left column) and the corresponding error maps (right column). The top row presents the ground-truth (GT) as well as one image. For non-dense methods, pixel displacements have been inferred from matching patches. Areas without correspondences are in black.

Method	EPE	EPE-occ	s0-10	s10-40	s40+	Time
FlowFields (Bailer et al 2015)	5.810	31.799	1.157	3.739	33.890	23s
DiscreteFlow (Menze et al 2015)	5.810	31.799	1.157	3.739	33.890	180s
EpicFlow (Revaud et al 2015)	6.285	32.564	1.135	3.727	38.021	16.4s
TF+OFM (Kennedy and Taylor 2015)	6.727	33.929	1.512	3.765	39.761	~400s
DeepFlow	6.928	38.166	1.182	3.859	42.854	25s
SparseFlowFused Timofte and Van Gool (2015)	7.189	3.286	1.275	3.963	44.319	20
DeepFlow* (Weinzaepfel et al 2013)	7.212	38.781	1.284	4.107	44.118	19s
S2D-Matching (Leordeanu et al 2013)	7.872	40.093	1.172	4.695	48.782	~2000s
LocalLayering (Sun et al 2014a)	8.043	40.879	1.186	4.990	49.426	
Classic+NL-P (Sun et al 2014b)	8.291	40.925	1.208	5.090	51.162	~800s
MDP-Flow2 (Xu et al 2012)	8.445	43.430	1.420	5.449	50.507	709s
NLTGV-SC (Ranftl et al 2014)	8.746	42.242	1.587	4.780	53.860	
LDOF (Brox and Malik 2011)	9.116	42.344	1.485	4.839	57.296	30s

Table 6 Results on MPI-Sintel test set (final version). EPE-occ is the EPE on occluded areas. s0-10 is the EPE for pixels with motions between 0 and 10 px and similarly for s10-40 and s40+. DeepFlow* denotes the preliminary version of DeepFlow published in Weinzaepfel et al (2013).

EpicFlow (Revaud et al 2015) relies on the output of DeepMatching to produce a piece-wise affine flow, and SparseFlowFused (Timofte and Van Gool 2015) combines matches obtained with DeepMatching and another algorithm.

We refer to the webpage of the MPI-Sintel dataset for complete results including the "clean" version.

Timings. As mentioned before, DeepMatching at half the resolution takes 15 seconds to compute on CPU and 0.2 second on GPU. The variational part requires 10 additional seconds on CPU. Note that by implementing it on GPU, we could obtain a significant speed-up as well. DeepFlow consequently takes 25 seconds in total on a single CPU core @ 3.6 GHz or 10.2s with GPU+CPU. This is in the same order of magnitude as the fastest among the best competitors, EpicFlow (Revaud et al 2015).

Results on Kitti. Table 7 summarizes the main results on the Kitti benchmark (see official website for complete results), when optimizing the parameters on the Kitti training set. EPE-Noc is the EPE computed only in non-occluded areas. "Out 3" corresponds to the proportion of incorrect pixel correspondences for an error threshold of 3 pixels, *i.e.* it corresponds to 1 − accuracy@3, and likewise for "Out-Noc 3" for non-occluded areas. In terms of EPE-noc, DeepFlow is on par with the best approaches, but performs somewhat worse in the occluded areas. This is due to a specificity of the

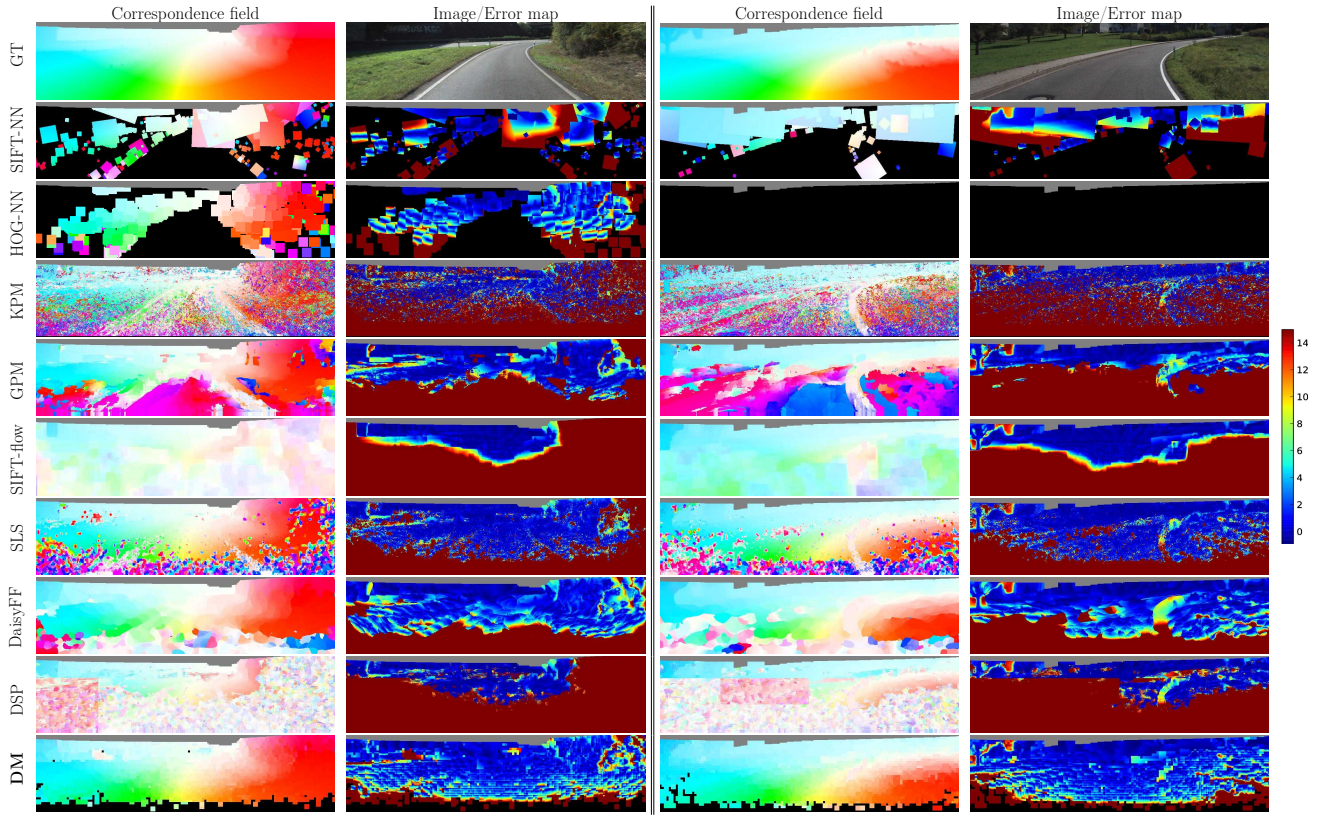

Fig. 17 Comparison of different matching methods on three challenging pairs from Kitti. Each pair of columns shows motion maps (left column) and the corresponding error maps (right column). The top row presents the ground-truth (GT) as well as one image. For non-dense methods, pixel displacements have been inferred from matching patches. Areas without correspondences are in black. To improve visualization, the sparse Kitti ground-truth is made dense using bilateral filtering.

Kitti dataset, in which motion is mostly homographic (especially on the image borders, where most surfaces like roads and walls are planar). In such cases, flow is better predicted using an affine motion prior, which locally well approximates homographies (a constant motion prior is used in DeepFlow). As a matter of facts, all top performing methods in terms of total EPE output piece-wise affine optical flow, either due to affine regularizers (BTF-ILLUM (Demetz et al 2014), NLTGB-SC (Ranftl et al 2014), TGV2ADCSIFT (Braux-Zin et al 2013)) or due to local affine estimators (EpicFlow (Revaud et al 2015)).

Note that the learned parameters on Kitti and MPI-Sintel are close. In particular, running the experiments with the same parameters as MPI-Sintel decreases EPE-Noc by only 0.1 pixels on the training set. This shows that our method does not suffer from overfitting.

Results on Middlebury. We optimize the parameters on the Middlebury training set by minimizing the average angular error with the same strategy as for MPI-Sintel. We find weights quasi-zero for the matching term due to the absence of large displacements. DeepFlow obtained an average endpoint error of 0.4 on the test which is competitive with the state of the art.

Method	EPE-noc	EPE	Out-Noc 3	Out 3	Time
DiscreteFlow (Menze et al 2015)	1.3	3.6	5.77%	16.63%	180s
FlowFields (Bailer et al 2015)	1.4	3.5	6.23%	14.01%	23s
DeepFlow	**1.4**	5.3	6.61%	17.35%	22s
BTF-ILLUM (Demetz et al 2014)	1.5	2.8	6.52%	11.03%	80s
EpicFlow (Revaud et al 2015)	1.5	3.8	7.88%	17.08%	16s
TGV2ADCSIFT (Braux-Zin et al 2013)	1.5	4.5	6.20%	15.15%	12s•
DeepFlow* (Weinzaepfel et al 2013)	1.5	5.8	7.22%	17.79%	17s
NLTGV-SC (Ranftl et al 2014)	1.6	3.8	5.93%	11.96%	16s•
Data-Flow (Vogel et al 2013b)	1.9	5.5	7.11%	14.57%	180s
TF+OFM (Kennedy and Taylor 2015)	2.0	5.0	10.22%	18.46%	350s

Table 7 Results on Kitti test set. EPE-noc is the EPE over non-occluded areas. Out-Noc 3 (resp. Out 3) refers to the percentage of pixels where flow estimation has an error above 3 pixels in non-occluded areas (resp. all pixels). DeepFlow* denotes the preliminary version of DeepFlow published in Weinzaepfel et al (2013). • denotes the usage of a GPU.

6 Conclusion

We have introduced a dense matching algorithm, termed DeepMatching. The proposed algorithm gracefully handles complex non-rigid object deformations and repetitive textured regions. DeepMatching yields state-of-the-art performance for image matching, on the Mikola-

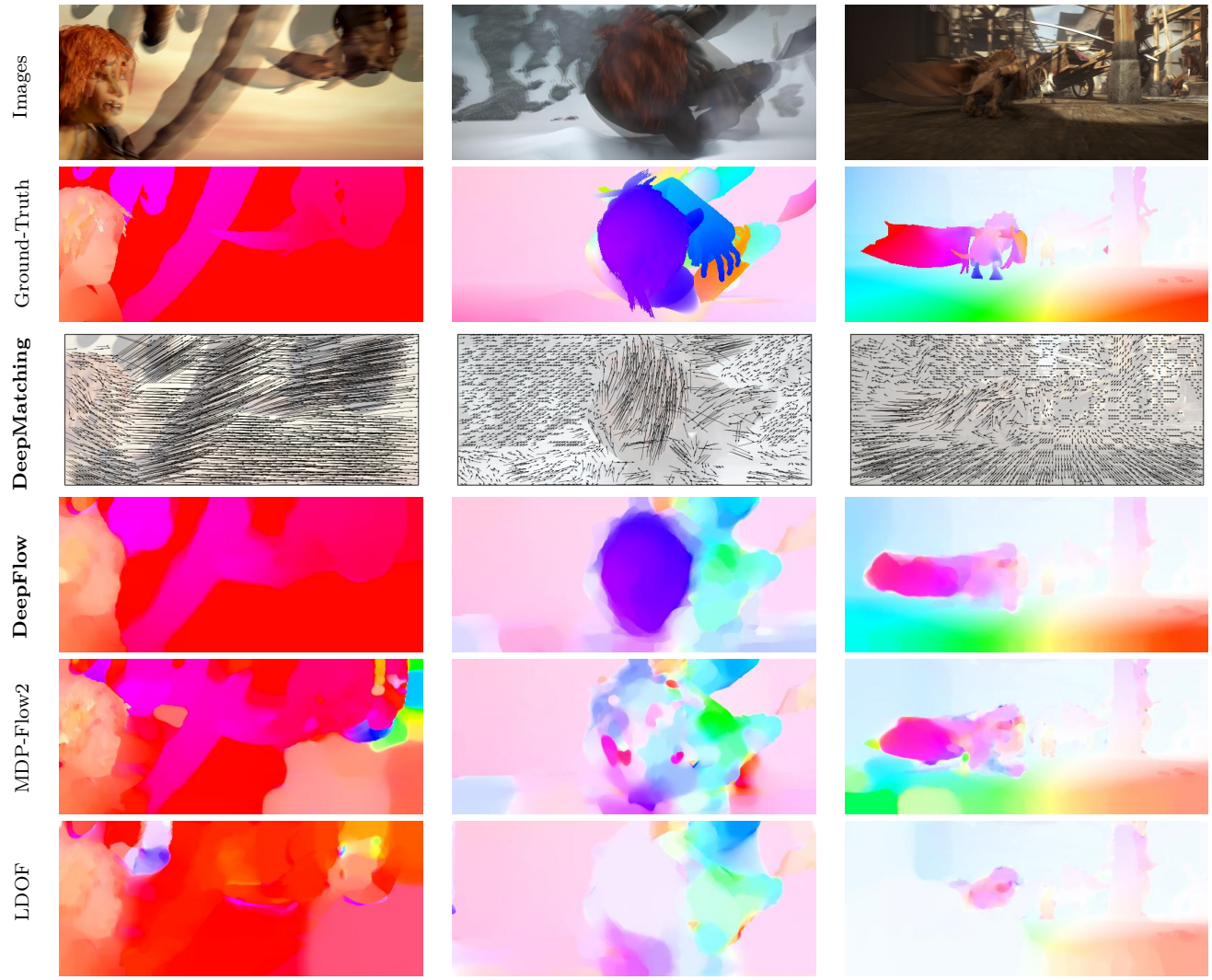

Fig. 18 Each column shows from top to bottom: two consecutive images, the ground-truth optical flow, the DeepMatching, our flow prediction (*DeepFlow*), and two state-of-the-art methods, LDOF (Brox and Malik 2011) and MDP-Flow2 (Xu et al 2012).

jczyk (Mikolajczyk et al 2005), the MPI-Sintel (Butler et al 2012) and the Kitti (Geiger et al 2013) datasets. Integrated in a variational energy minimization approach (Brox and Malik 2011), the resulting approach for optical flow estimation, termed DeepFlow, shows competitive performance on optical flow benchmarks.

Future work includes incorporating a weighting of the patches in Eq. (2) instead of weighting all patches equally to take into account that different parts of a large patch may belong to different objects. This could improve the performance of DeepMatching for thin objects, such as human limbs.

Acknowledgements This work was supported by the European integrated project AXES, the MSR/INRIA joint project, the LabEx PERSYVAL-Lab (ANR-11-LABX-0025), and the ERC advanced grant ALLEGRO.

References

Bailer C, Taetz B, Stricker D (2015) Flow Fields: Dense Correspondence Fields for Highly Accurate Large Displacement Optical Flow Estimation 5.3.3, 5.3.3

Baker S, Scharstein D, Lewis JP, Roth S, Black MJ, Szeliski R (2011) A database and evaluation methodology for optical flow. IJCV 2.2, 5.1

Barnes C, Shechtman E, Goldman DB, Finkelstein A (2010) The generalized PatchMatch correspondence algorithm. In: ECCV 1, 2.1, 2.2, 5.2.4

Bengio Y (2009) Learning deep architectures for AI. Foundations and Trends in Machine Learning 3.2

Black MJ, Anandan P (1996) The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. Computer Vision and Image Understanding 2.2

Braux-Zin J, Dupont R, Bartoli A (2013) A general dense image matching framework combining direct and feature-based costs. In: ICCV 5.3.3

Brox T, Malik J (2011) Large displacement optical flow: descriptor matching in variational motion estimation. IEEE Trans

PAMI (document), 1, 2.1, 2.2, 4.3, 5.2.4, 5.3.3, 18, 6
Brox T, Bruhn A, Papenberg N, Weickert J (2004) High accuracy optical flow estimation based on a theory for warping. In: ECCV 2.2, 4.3
Bruhn A, Weickert J, Feddern C, Kohlberger T, Schnörr C (2005) Variational optical flow computation in real time. IEEE Trans on Image Processing 2.2, 4.3
Butler DJ, Wulff J, Stanley GB, Black MJ (2012) A naturalistic open source movie for optical flow evaluation. In: ECCV (document), 5.1, 6
Chen Z, Jin H, Lin Z, Cohen S, Wu Y (2013) Large displacement optical flow from nearest neighbor fields. In: CVPR 1
Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: CVPR 1
Demetz O, Stoll M, Volz S, Weickert J, Bruhn A (2014) Learning brightness transfer functions for the joint recovery of illumination changes and optical flow. In: ECCV 5.3.3
Ecker A, Ullman S (2009) A hierarchical non-parametric method for capturing non-rigid deformations. Image and Vision Computing 2.1
Forsyth D, Ponce J (2011) Computer Vision: A Modern Approach. Pearson Education, Limited 1
Furukawa Y, Curless B, Seitz SM, Szeliski R (2010) Towards internet-scale multi-view stereo. In: CVPR 2.1
Geiger A, Lenz P, Stiller C, Urtasun R (2013) Vision meets robotics: The KITTI dataset. IJRR (document), 5.1, 6
HaCohen Y, Shechtman E, Goldman DB, Lischinski D (2011) Non-rigid dense correspondence with applications for image enhancement. SIGGRAPH 1, 2.1, 4.2, 5.1, 5.2.4
Hassner T, Mayzels V, Zelnik-Manor L (2012) On sifts and their scales. In: CVPR 2.2, 5.2.4
Horn BKP, Schunck BG (1981) Determining Optical Flow. Artificial Intelligence 2.2
Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:14085093 5.2.3
Kennedy R, Taylor CJ (2015) Optical flow with geometric occlusion estimation and fusion of multiple frames. In: EMMCVPR 5.3.3, 5.3.3
Keysers D, Deselaers T, Gollan C, Ney H (2007) Deformation models for image recognition. IEEE Trans PAMI 2.1
Kim J, Liu C, Sha F, Grauman K (2013) Deformable spatial pyramid matching for fast dense correspondences. In: CVPR 2.1, 5.2.4
Korman S, Avidan S (2011) Coherency sensitive hashing. In: ICCV 2.1
LeCun Y, Bottou L, Bengio Y, Haffner P (1998a) Gradient-based learning applied to document recognition. Proceedings of the IEEE 2.1, 3.2, 3.4
LeCun Y, Bottou L, Orr G, Muller K (1998b) Efficient backprop. In: Neural Networks: Tricks of the trade 3.2
Leordeanu M, Zanfir A, Sminchisescu C (2013) Locally affine sparse-to-dense matching for motion and occlusion estimation. In: ICCV 2.2, 5.3.3
Liu C, Yuen J, Torralba A (2011) SIFT flow: Dense correspondence across scenes and its applications. IEEE Trans PAMI 2.2, 5.2.4
Lowe DG (2004) Distinctive image features from scale-invariant keypoints. IJCV 1, 2.1, 2.2, 3.1, 3.2, 4.2, 5.2.4
Malik J, Perona P (1990) Preattentive texture discrimination with early vision mechanisms. Journal of the Optical Society of America A: Optics, Image Science, and Vision 3.2
Menze M, Heipke C, Geiger A (2015) Discrete Optimization for Optical Flow. In: GCPR 5.3.3, 5.3.3
Mikolajczyk K, Tuytelaars T, Schmid C, Zisserman A, Matas J, Schaffalitzky F, Kadir T, Gool LV (2005) A comparison of affine region detectors. IJCV (document), 2.1, 4.2, 5.1, 6
Muja M, Lowe DG (2009) Fast approximate nearest neighbors with automatic algorithm configuration. In: International Conference on Computer Vision Theory and Application VISSAPP'09), INSTICC Press 5.2.4
Papenberg N, Bruhn A, Brox T, Didas S, Weickert J (2006) Highly accurate optic flow computation with theoretically justified warping. IJCV 2.2
Philbin J, Isard M, Sivic J, Zisserman A (2010) Descriptor learning for efficient retrieval. In: ECCV 2.1
Ranftl R, Bredies K, Pock T (2014) Non-local total generalized variation for optical flow estimation. In: ECCV 5.3.3, 5.3.3
Revaud J, Weinzaepfel P, Harchaoui Z, Schmid C (2015) EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In: CVPR 5.3.3, 5.3.3, 5.3.3
Stoll M, Volz S, Bruhn A (2012) Adaptive integration of feature matches into variational optical flow methods. In: ACCV 4.3, 4.3
Sun D, Liu C, Pfister H (2014a) Local layering for joint motion estimation and occlusion detection. In: CVPR 5.3.3
Sun D, Roth S, Black M (2014b) A quantitative analysis of current practices in optical flow estimation and the principles behind them. IJCV 2.2, 4.3, 5.3.3
Sun J (2012) Computing nearest-neighbor fields via propagation-assisted kd-trees. In: CVPR 2.1, 5.2.4
Szeliski R (2010) Computer Vision: Algorithms and Applications. Springer 1, 2.1, 4.2
Timofte R, Van Gool L (2015) Sparse flow: Sparse matching for small to large displacement optical flow. In: Applications of Computer Vision (WACV) 5.3.3
Tola E, Lepetit V, Fua P (2008) A fast local descriptor for dense matching. In: CVPR 2.2
Tola E, Lepetit V, Fua P (2010) DAISY: An Efficient Dense Descriptor Applied to Wide Baseline Stereo. IEEE Trans PAMI 3.2, 5.2.4
Uchida S, Sakoe H (1998) A monotonic and continuous two-dimensional warping based on dynamic programming. In: ICPR 2.1
Vogel C, Roth S, Schindler K (2013a) An evaluation of data costs for optical flow. In: GCPR 2.2
Vogel C, Schindler K, Roth S (2013b) Piecewise rigid scene flow. In: ICCV 5.3.3
Wedel A, Cremers D, Pock T, Bischof H (2009) Structure- and motion-adaptive regularization for high accuracy optic flow. In: ICCV 4.3
Weinzaepfel P, Revaud J, Harchaoui Z, Schmid C (2013) Deepflow: Large displacement optical flow with deep matching. In: ICCV 1, 3.3, 3.3, 5.2.2, 5.3.3, 6, 5.3.3, 7
Werlberger M, Trobin W, Pock T, Wedel A, Cremers D, Bischof H (2009) Anisotropic Huber-L1 optical flow. In: BMVC 2.2
Wills J, Agarwal S, Belongie S (2006) A feature-based approach for dense segmentation and estimation of large disparity motion. IJCV 2.1
Xu L, Jia J, Matsushita Y (2012) Motion detail preserving optical flow estimation. IEEE Trans PAMI 1, 2.2, 4.3, 4.3, 5.3.3, 18
Yang H, Lin W, Lu J (2014) DAISY filter flow: A generalized discrete approach to dense correspondences. In: CVPR 2.1, 5.2.4
Young DM, Rheinboldt W (1971) Iterative solution of large linear systems. Academic Press, New York, NY 4.3
Zimmer H, Bruhn A, Weickert J (2011) Optic flow in harmony. IJCV 4.3, 4.3