The Generalized PatchMatch Correspondence Algorithm

Connelly Barnes[1], Eli Shechtman[2], Dan B Goldman[2], Adam Finkelstein[1]

[1]Princeton University, [2]Adobe Systems

Abstract. PatchMatch is a fast algorithm for computing dense approximate nearest neighbor correspondences between patches of two image regions [1]. This paper generalizes PatchMatch in three ways: (1) to find k nearest neighbors, as opposed to just one, (2) to search across scales and rotations, in addition to just translations, and (3) to match using arbitrary descriptors and distances, not just sum-of-squared-differences on patch colors. In addition, we offer new search and parallelization strategies that further accelerate the method, and we show performance improvements over standard kd-tree techniques across a variety of inputs. In constrast to many previous matching algorithms, which for efficiency reasons have restricted matching to sparse interest points, or spatially proximate matches, our algorithm can efficiently find global, dense matches, even while matching across all scales and rotations. This is especially useful for computer vision applications, where our algorithm can be used as an efficient general-purpose component. We explore a variety of vision applications: denoising, finding forgeries by detecting cloned regions, symmetry detection, and object detection.

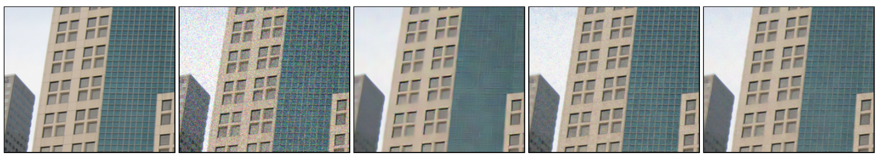


Fig. 1. Denoising using Generalized PatchMatch. Ground truth (a) is corrupted by Gaussian noise (b). Buades et al. [2] (c) denoise by averaging similar patches in a small local window: PSNR 28.93. Our method (d) uses PatchMatch for nonlocal search, improving repetitive features, but uniform regions remain noisy, as we use only $k = 16$ nearest neighbors: PSNR 29.11. Weighting matches from both algorithms (e) gives the best overall result: PSNR 30.90.

1 Introduction

Computing correspondences between image regions is a core issue in many computer vision problems, from classical problems like template tracking and optical flow, to low-level image processing such as non-local means denoising and example-based super-resolution, to synthesis tasks such as texture synthesis and image inpainting, to high level image analysis tasks like object detection, image segmentation and classification. Correspondence searches can be classified as either *local*, where a search is performed in a limited spatial window, or *global*, where all possible displacements are considered. Correspondences can also be classified as *sparse*, determined only at a subset of key feature points, or *dense*, determined at every pixel or on a dense grid in the input.

For efficiency, many common algorithms only use local or sparse correspondences. Local search can only identify small displacements, so multi-resolution refinement is often used (e.g., in optical-flow [3]), but large motions of small objects can be missed. Sparse keypoint [4, 5] correspondences are commonly used for alignment, 3D reconstruction, and object detection and recognition. These methods work best on textured scenes at high resolution, but are less effective in other cases. More advanced methods [6, 7] that start with sparse matches and then propagate them densely suffer from similar problems. Thus, such methods could benefit from relaxing the locality and sparseness assumptions. Moreover, many analysis applications [8–11] and synthesis applications [12–15] inherently require dense global correspondences for adequate performance.

The PatchMatch algorithm [1] finds dense, global correspondences an order of magnitude faster than previous approaches, such as dimensionality reduction (e.g. PCA) combined with tree structures like kd-trees, VP-trees, and TSVQ. The algorithm finds an approximate nearest-neighbor in an image for every small (e.g. 7x7) rectangular patch in another image, using a randomized cooperative hill climbing strategy. However, the basic algorithm finds only a single nearest-neighbor, at the same scale and rotation. To apply this algorithm more broadly, the core algorithm must be generalized and extended.

First, for problems such as object detection, denoising, and symmetry detection, one may wish to detect multiple candidate matches for each query patch. Thus we extend the core matching algorithm to find k nearest neighbors (k-NN) instead of only 1-NN. Second, for problems such as super-resolution, object detection, image classification, and tracking (at re-initialization), the inputs may be at different scales and rotations, therefore, we extend the matching algorithm to search across these dimensions. Third, for problems such as object recognition, patches are insufficiently robust to changes in appearance and geometry, so we show that arbitrary image descriptors can be matched instead.

The resulting generalized algorithm is simple and fast despite the high dimensional search space. The difficulty of performing a 4D search across translations, rotations, and scales had previously motivated the use of sparse features that are invariant to some extent to these transformations. Our algorithm efficiently finds dense correspondences despite the increase in dimension, so it offers an alternative to sparse interest point methods. Like the original PatchMatch algorithm, our generalized algorithm is up to an order of magnitude more efficient than kd-tree techniques. We show how performance is further enhanced by two improvements: (1) a new search technique we call "enrichment" that generalizes "coherent" or locally similar matches from spatial neighborhoods to neighborhoods in nearest neighbor space and (2) a parallel tiled algorithm on multi-core machines. Finally, for k-NN and enrichment, there were many possible algorithms, so we performed extensive comparisons to determine which worked best.

In summary, our main contributions are: (1) an extended matching algorithm, providing k nearest neighbors, searching across rotations and scales, and descriptor matching (Section 3.2-Section 3.5); (2) acceleration techniques, including a new search strategy called "enrichment" and a parallel algorithm for multi-core architectures (Section 3.3, Section 3.6) We believe this Generalized PatchMatch algorithm can be employed as a general component in a variety of existing and future computer vision methods, and we demonstrate its applicability for image denoising, finding forgeries in images, symmetry detection, and object detection.

2 Related work

When a *dense*, *global* matching is desired, previous approaches have typically employed tree-based search techniques. In image synthesis (e.g., [16]), one popular technique for searching image patches is dimensionality reduction (using PCA) followed by a search using a kd-tree [17]. In Boiman et al [18], nearest-neighbor image classification is done by sampling descriptors on a dense grid into a kd-tree, and querying this tree. Other tree structures that have been employed for querying patches included TSVQ [19] and vp-trees [20]. Another popular tree structure is the k-means-tree that was successfully used for fast image retrieval [21]. The FLANN method [22] combines multiple different tree structures and automatically chooses which one to use according to the data. Locality-sensitive hashing [23] and other hashing methods can be used as well. Each of these algorithms can be run in either approximate or exact matching mode, and find multiple nearest neighbors. When search across a large range of scales and rotations is required, a dense search is considered impractical due to the high dimensionality of the search space. The common way to deal with this case is via keypoint detectors [4]. These detectors either find an optimal local scale and the principal local orientation for each keypoint or do an affine normalization. These approaches are not always reliable due to image structure ambiguities and noise. The PatchMatch algorithm [1] was shown to find a single nearest neighbor one to two orders of magnitude faster than tree-based techniques, for equivalent errors, with running time on the order of seconds for a VGA input on a single core machine. This paper offers performance improvements and extends it to dense k-nn correspondence across a large range of scales and rotations. The Generalized PatchMatch algorithm can operate on any common image descriptors (e.g., SIFT) and unlike many of the above tree structures, supports any distance function. Even while the algorithm naturally supports dense global matching, it may also be constrained to only accept matches in a local window if desired.

Section 4 investigates several applications in computer vision, and prior work related to those applications is mentioned therein.

3 Algorithm

This section presents four generalizations of the PatchMatch algorithm suitable for a wide array of computer vision problems. After reviewing the original algorithm [1], we present our extensions, including k-nearest neighbors, matching across rotations and scale, and matching descriptors. We finally show how performance can be improved with a new search strategy called “enrichment,” and a parallel tiled algorithm suitable for multi-core architectures.

3.1 The PatchMatch algorithm

Here we review the original PatchMatch algorithm as proposed by Barnes et al. [1]. It is an efficient randomized approach to solving the following problem: for every $p \times p$ patch in image A, find the approximate nearest neighbor patch in image B, minimizing the sum-squared difference between corresponding pixels.

A *nearest-neighbor field* (NNF) is a function $\mathbf{f} : A \mapsto \mathbb{R}^2$, defined over all possible patch coordinates (locations of patch centers) in image A, for some distance function D between two patches. Given patch coordinate $\mathbf{a}$ in image A and its corresponding nearest neighbor $\mathbf{b}$ in image B, $\mathbf{f}(\mathbf{a})$ is simply $\mathbf{b}$.[1] We refer to the values of f as *nearest neighbors*, and they are stored in an array whose dimensions are those of A.

Note that the NNF differs from an optical flow field (OFF). The NNF uses no smoothness constraints and finds the best match independent of neighboring matches. The OFF is defined by ground truth motion and is often computed with smoothness constraints.

The randomized algorithm works by iteratively improving the nearest-neighbor field $\mathbf{f}$ until convergence. Initially, the nearest neighbor field is filled with random coordinates, uniformly sampled across image B. Next, the field is iteratively improved for a fixed number of iterations, or until convergence. The algorithm examines field vectors in scan order, and tries to improve each using two sets of candidates: *propagation*, and *random search*.

The *propagation* trials attempt to improve a nearest neighbor $\mathbf{f}(\mathbf{x})$ using the known nearest neighbors above or to the left. The new candidates for $\mathbf{f}(\mathbf{x})$ are $\mathbf{f}(\mathbf{x} - \boldsymbol{\Delta}_p) + \boldsymbol{\Delta}_p$, where $\boldsymbol{\Delta}_p$ takes on the values of $(1, 0)$ and $(0, 1)$. Propagation takes a downhill step if either candidate provides a smaller patch distance D. (On even iterations, propagation is done in reverse scan order, and candidates below and to the right are examined, so information propagates up and left.) Propagation converges very quickly, but if used alone ends up in a local minimum. So a second set of trials employs *random search*: a sequence of candidates is sampled from an exponential distribution, and $\mathbf{f}(\mathbf{x})$ is improved if any of the candidates has smaller distance D. Let $\mathbf{v_0}$ be the current nearest neighbor $\mathbf{f}(\mathbf{x})$. The candidates $\mathbf{u}_i$ are constructed by sampling around $\mathbf{v}_0$ at an exponentially decreasing distance: $\mathbf{u}_i = \mathbf{v}_0 + w\alpha^i\mathbf{R}_i$, where $\mathbf{R}_i$ is a uniform random in $[-1, 1] \times [-1, 1]$, w is the maximum image dimension, and α is a

[1] Our notation is in absolute coordinates, vs relative coordinates in Barnes et al. [1]

ratio between window sizes ($\alpha = 1/2$ was used). The index i is increased from $i = 0, 1, 2, ..., n$ until the search radius $w\alpha^i$ is below 1 pixel. For more details, see Barnes et al. [1].

3.2 *k*-Nearest neighbors

For problems such as denoising, symmetry detection, and object and clone detection, we wish to compute more than a single nearest neighbor at every (x, y) position. This can be done by collecting k nearest neighbors for each patch. Thus the NNF $\mathbf{f}$ is a multi-valued map, with k values. There are many possible modifications of PatchMatch to compute the k-NN. We have compared the efficiency of several of these against a standard approach: dimensionality reduction with PCA, followed by construction of a kd-tree [17] with all patches of image B projected onto the PCA basis, then an independent ϵ-nearest neighbor lookup in the kd-tree for each patch of image A projected onto the same basis.

Since each of these algorithms can be tuned for either greater accuracy or greater speed, we evaluated each across a range of settings. For PatchMatch, we simply computed additional iterations, and for kd-trees we adjusted the ϵ and PCA dimension parameters. The relative efficiency of these algorithms is plotted in Figure 2. We also compare with FLANN [22], a package that includes kd-tree, k-means tree, a hybrid algorithm, and a large number of parameters that can be tuned for performance.

Heap algorithm. In the most straightforward variant, we associate k nearest neighbors with each patch position. During propagation, we improve the nearest neighbors at the current position by exhaustively testing each of the k nearest neighbors to the left or above (or below or right on even iterations). The new candidates are $\mathbf{f}_i(\mathbf{x}-\mathbf{\Delta}_p)+\mathbf{\Delta}_p$, where $\mathbf{\Delta}_p$ takes on the values $(1, 0)$ and $(0, 1)$, and $i = 1 \dots k$. If any candidate is closer than the worst candidate currently stored at $\mathbf{x}$, that worst candidate is replaced with the candidate from the adjacent patch. This can be done efficiently with a max-heap, where the heap stores the patch distance D. The random search phase works similarly: n samples are taken around each of the k nearest neighbors, giving nk samples total. The worst element of the heap is evicted if the candidate's distance is better. When examining candidates, we also construct a hash table to quickly identify candidates already in our k list, to prevent duplicate entries.

Details of the additional strategies tested can be found in supplementary material. Briefly, they include variants of the heap algorithm in which fewer than k samples are taken from the neighbor list for propagation and/or search ("P best," "P random", "RS best", "RS random", "P varying", "RS varying"); variants of the heap algorithm where k is changed over time ("Increase k", "Decrease k"); and modifications of the original 1NN algorithm in which no heap is used but the sequence of candidates is retained ("List 1-NN", "Run 1-NN k times"). Some of these algorithms complete single iterations faster than the basic heap algorithm described above, but convergence is slower as they propagate less information within an iteration. In general, the original heap algorithm is a good choice over a wide range of the speed/quality curve.

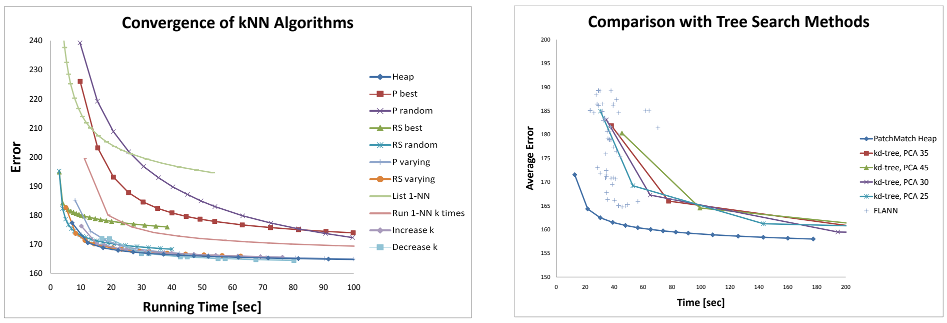


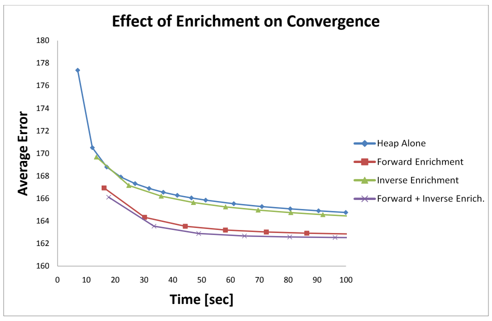
Fig. 2. *Left:* Performance of k-PatchMatch variants, with $k = 16$, averaged over all images in Figure 4, resized to 0.2MP, and matched against themselves. Error is average L^2 patch distance over all k. Points on each curve represent progress after each iteration. *Right:* Comparison with kd-tree and FLANN, at 0.3 MP, averaged over the dataset.

We find the basic heap algorithm outperforms kd-tree over a wide range of k and image sizes: for example, our algorithm is several times faster than kd-tree, for $k = 16$ and input images of 0.1 to 1.0MP. In our comparisons to the kd-tree implementation of Mount and Arya [17] and FLANN [22], we gave the competition the benefit of the doubt by tuning all possible parameters, while adjusting only the number of iterations for our heap algorithm. FLANN offers several algorithms, so we sampled a large range of algorithmic options and parameters, indicated by the + marks in Figure 2. FLANN can also automatically optimize parameters, but we found the resulting performance always lies within the convex hull of our point-sampling. In both cases, this extensive parameter-tuning resulted in performance that approached – but never exceeded – our heap algorithm. Thus, we propose that the general k-PatchMatch heap algorithm is a better choice for a wide class of problems requiring image patch correspondence. With additional optimization of our algorithm, the performance gap might be even greater.

3.3 Enrichment

In this section we propose one such optimization for improving PatchMatch performance further. The propagation step of PatchMatch propagates good matches across the spatial dimensions of the image. However, in special cases we can also consider propagating matches across the space of patches themselves: For example, when matching an image A to itself – as in non-local-means denoising (Section 4.1) – many of a patch's k nearest neighbors will have the original patch and some of the other $k - 1$ patches in their own k-NN list.

We define enrichment as the propagation of good matches from a patch to its k-NN, or vice versa. We call this operation enrichment because it takes a nearest neighbor field and improves it by considering a "richer" set of potentially good candidate matches than propagation or random search alone. From a graph-theoretic viewpoint, we can view ordinary propagation as moving good matches



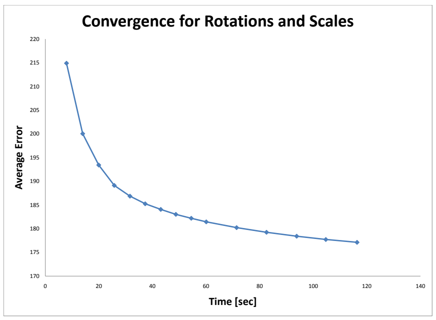


Fig. 3. *Left:* Comparison of the heap algorithm with and without enrichment. As in Figure 2, times and errors are averaged over the dataset of Figure 4 at 0.2 megapixels and $k = 16$ neighbors. *Right:* Searching across all rotations and scales.

along a rectangular lattice whose nodes are patch centers (pixels), whereas enrichment moves good matches along a graph where every node is connected to its k-NN. We introduce two types of enrichment, for the special case of matching patches in A to other patches in A:

Forward enrichment uses compositions of the function $\mathbf{f}$ with itself to produce candidates for improving the nearest neighbor field. The canonical case of forward enrichment is $\mathbf{f}^2$. That is, if $\mathbf{f}$ is a NNF with k neighbors, we construct the NNF $\mathbf{f}^2$ by looking at all of our nearest neighbor's nearest neighbors: there are k^2 of these. The candidates in $\mathbf{f}$ and $\mathbf{f}^2$ are compared and the best k overall are used as an improved NNF $\mathbf{f}'$. If min() denotes taking the top k matches, then we have: $\mathbf{f}' = \min(\mathbf{f}, \mathbf{f}^2)$. See the supplementary material for other variants.

Similarly, **inverse enrichment** walks the nearest-neighbor pointers backwards to produce candidates for improving the NNF. The canonical algorithm here is $\mathbf{f}^{-1}$. That is, compute the multi-valued inverse $\mathbf{f}^{-1}$ of function $\mathbf{f}$. Note that $\mathbf{f}^{-1}(a)$ may have zero values if no patches point to patch a, or more than k values if many patches point to a. We store $\mathbf{f}^{-1}$ by using a list of varying length at each position. Again, to improve the current NNF, we rank our current k best neighbors and all neighbors in $\mathbf{f}^{-1}$, producing an improved NNF $\mathbf{f}''$: $\mathbf{f}'' = \min(\mathbf{f}, \mathbf{f}^{-1})$. Note that in most cases the distance function is symmetric, so patch distances do not need to be computed for $\mathbf{f}^{-1}$. Finally we can concatenate inverse and forward enrichment, and we found that $\mathbf{f}^{-1}$ followed by $\mathbf{f}^2$ is fastest overall. The performance of these algorithms is compared in Figure 3.

In the case of matching different images A and B, inverse enrichment can be trivially done. Forward enrichment can be applied by computing nearest neighbor mappings in both directions; we leave this investigation for future work.

3.4 Rotations and scale

For some applications, such as object detection, denoising or super-resolution, it may be desirable to match patches across a range of possible rotations or scales.

Without loss of generality, we compare upright unscaled patch a in image A, with patch b in image B that is rotated and scaled around its center.

To search a range of rotations $\theta \in [\theta_1, \theta_2]$ and a range of scales $s \in [s_1, s_2]$, we simply extend the search space of the original PatchMatch algorithm from (x, y) to (x, y, θ, s), extending the definition of our nearest-neighbor field to a mapping $\mathbf{f} : \mathbb{R}^2 \mapsto \mathbb{R}^4$. Here $\mathbf{f}$ is initialized by uniformly sampling from the range of possible positions, orientations and scales. In the propagation phase, adjacent patches are no longer related by a simple translation, so we must also transform the relative offsets by a Jacobian. Let $\mathbf{T}(\mathbf{f}(\mathbf{x}))$ be the full transformation defined by (x, y, θ, s): the candidates are thus $\mathbf{f}(\mathbf{x} - \mathbf{\Delta}_p) + \mathbf{T}'(\mathbf{f}(\mathbf{x} - \mathbf{\Delta}_p))\mathbf{\Delta}_p$. In the random search phase, we again use a window of exponentially decreasing size, only now we contract all 4 dimensions of the search around the current state.

The convergence of this approach is shown in Figure 3. In spite of searching over 4 dimensions instead of just one, the combination of propagation and random search successfully samples the search space and efficiently propagates good matches between patches. In contrast, with a kd-tree, it is nontrivial to search over all scales and rotations. Either all rotations and scales must be added to the tree, or else queried, incurring enormous expenses in time or memory.

3.5 Matching with arbitrary descriptors and distance metrics

The PatchMatch algorithm was originally implemented using the sum-of-squared differences patch distance, but places no explicit requirements on the distance function. The only implicit assumption is that patches with close spatial proximity should also be more likely to have similar best-nearest-neighbors, so that PatchMatch can be effective at propagating good nearest neighbors and finding new ones. This turns out to be true for a variety of descriptors and distance functions. In fact, the algorithm can converge even more quickly when using large-area feature descriptors than it does with small image patches, because they tend to vary relatively slowly over the image. In general, the "distance function" can actually be any algorithm that supplies a total ordering, and the matching can even be performed between entirely different images — the rate of convergence depends only on the size of coherent matching regions. Thus, our matching is quite flexible.

In this paper we explore several examples. In Section 4.3 we implement symmetry detection with a modified L^2 patch distance that is robust to changes in luminance. In Section 4.4, we perform label transfer by sampling a SIFT descriptor at every pixel. Our matching algorithm performs a global search, so two matched objects can be present in different regions of the image.

3.6 Parallel tiled algorithm

Barnes et al. proposed a parallel variant of PatchMatch using "jump flooding" for the propagation phase [1]. This algorithm was intended for GPU usage. However, on the CPU, this approach is less effective than serial propagation and converges more slowly in each iteration.

Fig. 4. Dataset of 36 input images for denoising.

On a multi-core architecture, we propose parallelizing PatchMatch by dividing the NNF into horizontal tiles, and handling each tile on a separate core. Because the tiles are handled in parallel, information can propagate vertically the entire length of a tile in a single iteration. To ensure information has a chance to propagate all the way up and down the image, we synchronize using a critical section after each iteration. To prevent resource conflicts due to propagation between abutting tiles, we write back the nearest neighbors in the last row of the tile only after synchronization. Note that both propagation/random search and forward enrichment can be parallelized using this tile scheme.

We observe a nearly linear speed-up, on our 8 core test machine. Our timing values in this paper use only one core unless otherwise indicated. See the supplementary material for details.

4 Vision applications

This section investigates several possible applications for the generalized PatchMatch algorithm: denoising, clone detection, symmetry detection, and object detection.

4.1 Non-local means denoising

For image denoising, Buades et al. [2] showed that high-quality results could be obtained by *non-local means denoising:* finding similar patches within an image and then averaging these. Subsequent work [24, 25] showed that this patch-based method could be extended to obtain state-of-the-art results by performing additional filtering steps. While Buades et al. [2] searched for similar patches only within a limited search window, Brox et al. [26] showed that a tree-based method could be used to obtain better quality for some inputs. However they do increase the distance to far away patches so searching is still limited to some local region.

Our kNN algorithm can be used to find similar patches in an image, so it can be used as a component in these denoising algorithms. We implemented the simple method of Buades et al. [2] using our kNN algorithm. This method works by examining each source patch of an image, performing a local search over all patches within a fixed distance r of the source patch, computing a Gaussian-weighted L^2 distance d between the source and target patch, and computing a weighted mean for the center pixel color with some weight function $f(d)$.

To use our kNN algorithm in this denoising framework, we can simply choose a number of neighbors k, and for each source patch, use its k-NN in the entire image as the list of target patches. To evaluate this algorithm, we chose 36 images as our dataset (Figure 4). We corrupted these images by adding to each RGB channel noise from a Gaussian distribution with $\sigma = 20$ (out of 256 grey levels). If the dataset is denoised with Buades et al (using an 11x11 search window) the average PSNR is 27.8. Using our kNN algorithm gives an average PSNR of 27.4, if the number of neighbors is small ($k = 16$). Counterintuitively, our algorithm gives worse PSNR values because it finds better matches. This occurs because our algorithm can search the entire image for a good match, therefore in uniform regions, the patch's noise pattern simply matches similar noise.

One solution would be to significantly increase our k. However, we found that Buades et al and our algorithm are complementary and both are efficient. Therefore, we simply run both algorithms, and list all target patches found by each, before averaging the patches under a weight function $f(d)$. We train the weight function on a single image and then evaluate on the dataset. This combined algorithm has an average PSNR of 28.4, showing that our kNN matching can improve denoising in the framework of Buades et al. The best results are obtained on images with repeating elements, as in Figure 1.

We also compared our results with the state-of-the-art BM3D algorithm [24]. For our dataset, BM3D produced an average PSNR of 29.9, significantly outperforming our results. However, we intentionally kept our denoising algorithm simple, and hypothesize that more advanced algorithms [24, 25] that are based on local search for speed, could do even better with our kNN algorithm.

4.2 Clone detection

One technique for digitally forging images is to remove one region of an image by cloning another region. For example, this can be done using Adobe Photoshop's clone brush. Such forgeries have been a concern in the popular press of late, as fake photos have been published in major newspapers.

Methods of detecting such forgeries have been proposed recently [11, 27]. These methods propose breaking the image into either square or irregularly shaped patches, applying PCA or DCT to discard minor variations in the image due to noise or compression, and sorting the resulting blocks to detect duplicates.

We can apply our kNN algorithm for the purposes of detecting cloned regions. Rather than sorting all blocks into a single ordered list, we can consider for each patch, its k-NN as potentially cloned candidates. We identify cloned regions by detecting connected "islands" of patches that all have similar nearest neighbors.

Specifically, we construct a graph and extract connected components from the graph to identify cloned regions. The vertices of the graph are the set of all (x, y) pixel coordinates in the image. For each (x, y) coordinate, we create a horizontal or vertical edge in the graph if its kNN are similar to the neighbors at $(x + 1, y)$ or $(x, y + 1)$, respectively. We call two lists A and B of kNN similar if for any pair of nearest neighbors $(ax, ay) \in A$ and $(bx, by) \in B$, the nearest neighbors are within a threshold distance T of each other, and both have a patch distance less than a maximum distance threshold. Finally, we detect connected

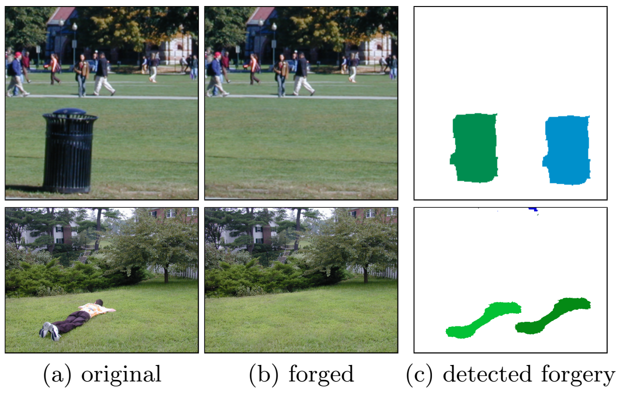


Fig. 5. Detecting image regions forged using the clone brush. Shown are (a) the original, untampered image, (b) the forged image, (c) cloned regions detected by our kNN algorithm and connected components. Imagery from [11].

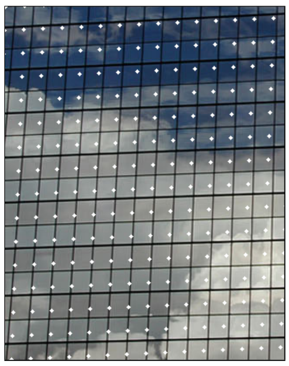


Fig. 6. Symmetry detection using a regular lattice (superimposed white dots).

components in the graph, and consider any component with an area above a minimum cloned region size C (we use $C = 50$) to be a cloned region.

Examples of our clone detection implementation are shown in Figure 5. Note that cloned areas are correctly identified. However, the area of the clone is not exactly that of the removed objects because our prototype is not robust to noise, compression artifacts, or feathering. Nevertheless, we believe it would be easy to adapt the algorithm to better recover the complete mask.

4.3 Symmetry detection

Detecting symmetric features in images has been of interest recently. A survey of techniques for finding rotational and reflective symmetries is given by Park et al. [28]. Methods have also been developed for finding translational symmetries in the form of regular lattices [8].

Because our kNN algorithm matches repeated features non-locally, it can be used as a component in symmetry detection algorithms. Symmetries have been detected using sparse interest points, such as corner detectors or SIFT or edge interest points [28]. In contrast to sparse methods, our algorithm can match densely sampled descriptors such as patches or SIFT descriptors, and symmetries can be found by examining the produced dense correspondence field. This suggests that our algorithm may be able to find symmetric components even in the case where there are no sparse interest points present.

To illustrate how our method can be used for symmetry detection, we propose a simple algorithm for finding translational symmetries in the form of repeated elements on a non-deformed lattice. First we run our kNN algorithm. The descriptor for our algorithm is 7x7 patches. We calculate patch distance using L^2 between corresponding pixels after correcting for limited changes in lighting

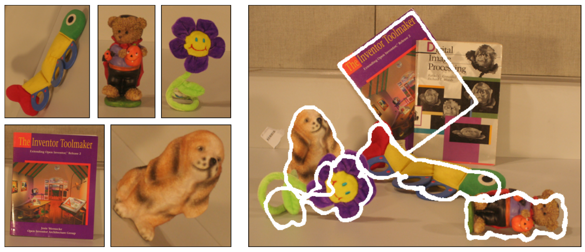


Fig. 7. Detecting objects. Templates, left, are matched to the image, right. Square patches are matched, searching over all rotations and scales, as described in Section 3.4. A similarity transform is fit to the resulting correspondences using RANSAC.

by normalizing the mean and standard deviation in luminance to be equal. We find $k = 16$ nearest neighbors, and then use RANSAC [29] to find the basis vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ that form the lattice. We classify as inliers the coordinates where the distance between the lattice and all of the kNN is small. A result of our symmetry detection is shown in Figure 6.

4.4 Object detection

Methods for object detection include deformable templates [30], boosted cascades [31], matching of sparse features such as SIFT [5], and others. Our algorithm can match densely sampled features, including upright patches, rotating or scaled patches, or descriptors such as SIFT. These matches are global, so that correspondences can be found even when an object moves across an image, or rotates or scales significantly. Provided that the descriptor is invariant to the change in object appearance, the correct correspondence will be found.

In Figure 7 we show an example of object detection. Similar to the method of Guo and Dyer [32], we break the template into small overlapping patches. We query these patches against the target image, searching over all rotations, and a range of scales, as per Section 3.4. A similarity transform is fit from the template to the target using RANSAC. We calculate patch distance using L^2, after correcting for lighting as we did in symmetry detection. The result is that we can find objects under partial occlusions and at different rotations and scales.

For greater invariance to lighting and appearance changes, a more complex local appearance model is needed. However it is straightforward to incorporate more complex models into our algorithm! For example, suppose we have photographs of two similar objects with different appearance. We might wish to propagate labels from one image to the other for all similar objects and background. The SIFT Flow work [33] shows that this can be done using SIFT features correspondence on a dense grid combined with an optical-flow like smoothness term. The resulting field is solved using a coarse-to-fine approach and global optimization (belief propagation). Like most optical flow methods, SIFT Flow assumes locality and smoothness of the flow and thus can fail to align objects under large displacements. As shown in Figure 8, we can correctly transfer labels even when objects move a large amount. We do this by densely sampling SIFT descriptors and then matching these as described in Section 3.5.

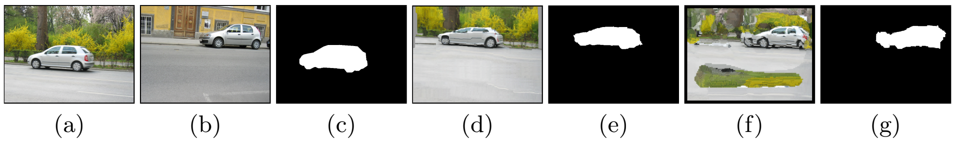


Fig. 8. Label transfer using our method with SIFT descriptors. (a) car A; (b) car B; (c) labeled A; (d) A warped to match B using SIFT Flow [33] as well as the transferred label mask in (e); (f) A warped to B using our method and the transferred label mask in (g). Our flow is globally less smooth but can handle arbitrarily large motions.

5 Discussion and future work

This paper generalizes the PatchMatch algorithm to encompass a broad range of core computer vision applications. We demonstrate several prototype examples, but many more are possible with additional machinery. For example, example-based super-resolution can use PatchMatch, using a single [34] or multiple [12] images. Section 4.4 shows an example of transferring labels using correspondences without a term penalizing discontinuity, but in other settings a neighborhood term is necessary for accurate optical flow [3, 6]. Finally, although we demonstrate object detection, our speed is not competitive with the best sparse tracking methods. It is possible that some variations of this approach using fewer iterations and downsampled images could be used to provide real-time tracking.

Acknowledgements

We thank ECCV reviewers for helpful comments, as well as the support of Adobe and NSF Award 0937139, and Flickr users for Creative Commons imagery: Moogs, Swami Stream, Laurence & Annie, Bill Liao, Badwsky, Cwalker71, Swamibu, Stuck in Customs, Xymox, Christopher S. Penn, Eric Brumble, Kibondo, Mazzaq-Mauro Mazzacurati, Slack12, Thomashawk, Bex in Beijing, Paul Keleher, CarbonNYC, SteveWhis, Arranging Matches, Professor Bop, Whirling Phoenix, Cindy47452.

References

1. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.: PatchMatch: a randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics (Proc. SIGGRAPH) **28** (2009) 24
2. Buades, A., Coll, B., Morel, J.: A non-local algorithm for image denoising. In: Proc. CVPR. (2005) II: 60
3. Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M., Szeliski, R.: A database and evaluation methodology for optical flow. In: Proc. ICCV. Volume 5. (2007)
4. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence (2005) 1615–1630
5. Lowe, D.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60** (2004) 91–110
6. Brox, T., Malik, J.: Large displacement optical flow, CVPR (2009)
7. Simon, I., Seitz, S.: A probabilistic model for object recognition, segmentation, and non-rigid correspondence. In: Proc. CVPR. (2007) 1–7
8. Hays, J., Leordeanu, M., Efros, A., Liu, Y.: Discovering texture regularity as a higher-order correspondence problem. Lec. Notes in Comp. Sci. **3952** (2006) 522

9. Boiman, O., Irani, M.: Detecting irregularities in images and in video. International Journal of Computer Vision **74** (2007) 17–31
10. Bagon, S., Boiman, O., Irani, M.: What is a good image segment? A unified approach to segment extraction. In: Proc. ECCV, Springer (2008) IV: 44
11. Popescu, A., Farid, H.: Exposing digital forgeries by detecting duplicated image regions. Department of Computer Science, Dartmouth College (2004)
12. Freeman, W., Jones, T., Pasztor, E.: Example-based super-resolution. IEEE Computer Graphics and Applications (2002) 56–65
13. Efros, A., Leung, T.: Texture synthesis by non-parametric sampling. In: Proc. ICCV. Volume 2. (1999) 1033–1038
14. Criminisi, A., Pérez, P., Toyama, K.: Region filling and object removal by exemplar-based image inpainting. IEEE Trans. on Image Processing **13** (2004)
15. Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: Summarizing visual data using bidirectional similarity. Proc. CVPR (2008)
16. Hertzmann, A., Jacobs, C., Oliver, N., Curless, B., Salesin, D.: Image analogies. In: ACM Transactions on Graphics (Proc. SIGGRAPH). (2001) 327–340
17. Mount, D.M., Arya, S.: ANN: A library for approx. nearest neighbor search (1997)
18. Boiman, O., Shechtman, E., Irani, M.: In defense of nearest-neighbor based image classification. In: Proc. CVPR. Volume 2. (2008) 6
19. Niyogi, S., Freeman, W.: Example-based head tracking. In: Proc. of Conf. on Automatic Face and Gesture Recognition (FG'96). (1996) 374
20. Kumar, N., Zhang, L., Nayar, S.K.: What is a good nearest neighbors algorithm for finding similar patches in images? In: Proc. ECCV. (2008) II: 364–378
21. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: Proc. CVPR. Volume 5. (2006)
22. Muja, M., Lowe, D.: Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (2009)
23. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. In: Symp. on Comp. Geom. (2004) 253–262
24. Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K.: Image denoising by sparse 3-D transform-domain collaborative filtering. IEEE Trans. Image Processing **16** (2007)
25. Mairal, J., Bach, F., Ponce, J., Sapiro, G., Zisserman, A.: Non-local sparse models for image restoration. In: Proc. of ICCV. (2009)
26. Brox, T., Kleinschmidt, O., Cremers, D.: Efficient nonlocal means for denoising of textural patterns. IEEE Transactions on Image Processing **17** (2008) 1083–1092
27. Bayram, S., Sencar, H., Memon, N.: A Survey of Copy-Move Forgery Detection Techniques. IEEE Western New York Image Processing Workshop (2008)
28. Park, M., Leey, S., Cheny, P., Kashyap, S., Butty, A., Liuy, Y.: Performance evaluation of state-of-the-art discrete symmetry detection, CVPR (2008)
29. Fischler, M., Bolles, R.: Random sample consensus: A paradigm for model fitting with apps. to image analysis and automated cartography. Comm. ACM 24 (1981)
30. Jain, A., Zhong, Y., Dubuisson-Jolly, M.: Deformable template models: A review. Signal Processing **71** (1998) 109–129
31. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proc. CVPR. (2001)
32. Guo, G., Dyer, C.: Patch-based image correlation with rapid filtering. In: The 2nd Beyond Patches Workshop, in conj. with IEEE CVPR'07. (2007)
33. Liu, C., Yuen, J., Torralba, A., Sivic, J., MIT, W.: SIFT flow: dense correspondence across different scenes. In: Proc. ECCV, Springer (2008) III: 28
34. Glasner, D., Bagon, S., Irani, M.: Super-Resolution from a Single Image. In: Proc. of ICCV. (2009)