PatchMatchGraph: Building a Graph of Dense Patch Correspondences for Label Transfer

Stephen Gould and Yuhang Zhang

Research School of Computer Science, ANU, Australia
{stephen.gould,yuhang.zhang}@anu.edu.au

Abstract. We address the problem of semantic segmentation, or multi-class pixel labeling, by constructing a graph of dense overlapping patch correspondences across large image sets. We then transfer annotations from labeled images to unlabeled images using the established patch correspondences. Unlike previous approaches to non-parametric label transfer our approach does not require an initial image retrieval step. Moreover, we operate on a graph for computing mappings between images, which avoids the need for exhaustive pairwise comparisons. Consequently, we can leverage offline computation to enhance performance at test time. We conduct extensive experiments to analyze different variants of our graph construction algorithm and evaluate multi-class pixel labeling performance on several challenging datasets.

1 Introduction

The availability of large online repositories of human-annotated images has greatly increased in the past several years. As a result data-driven approaches to many computer vision tasks have emerged as compelling alternatives to the more traditional model-based approaches. One important such task is multi-class pixel labeling, or semantic segmentation [9, 18], where the goal is to assign a class label to each pixel in a given image—for example, pixels can be labeled as belonging to "road", "sky", "tree", etc. Data-driven approaches on this task (e.g., [16, 20, 19]) are competitive with state-of-the-art model-based approaches while offering several key advantages. For example, the set of class labels does not need to be pre-defined and data can be added incrementally without having to re-estimate model parameters. This is attractive for life-long learning systems.

Typical data-driven methods for semantic segmentation involve three main steps to label a novel target image. First, a small set of similar images is retrieved from a large corpus of annotated images. This can be done, for example, by comparing global image descriptors such as GIST [17]. Next, dense pixel mappings between images in the retrieval set and the target image are computed. Some methods enforce smooth mappings [16], while others allow arbitrary mappings [20]. Last, labels from the images in the retrieval set are transferred to the target image via the dense pixel mappings. Since ambiguity may arise when superimposing labels from different images, a Markov random field (MRF) is often used to aggregate labels and enforce smoothness constraints. Preprocessing of

A. Fitzgibbon et al. (Eds.): ECCV 2012, Part V, LNCS 7576, pp. 439–452, 2012.

images in the dataset, for example by hashing GIST descriptors, can help accelerate the retrieval step. However, the most expensive operation is the second step which requires pairwise comparisons between the target image and all retrieved images. This can only be done at test time. Moreover, the image retrieval step may adversely affect results on small objects, which are not well accounted for when encoding the image by a single global descriptor.

We address these problems by building a graph over images that allows us to rapidly find good mappings between images in the dataset. Specifically, we propose to construct a graph of dense and overlapping correspondences between small patches from all the images in our dataset simultaneously. The graph encodes k-nearest-neighbour matches between the patches. Our method does not require exhaustive pairwise comparisons and, consequently, has lower running-time and is independent of the number of images in the dataset at test time. Moreover, our graph formulation allows efficient incremental addition of new images and offline computation can be continued to steadily find better matches.

Barnes et al. [2] proposed a very efficient randomized search algorithm for finding dense patch correspondences between a pair of images. Their key observation was that good matches can be propagated to adjacent patches in an image. We extend this idea to entire image graphs. For example, a dataset may contain many instances of nearly identical objects. Consider three such objects: A, B and C. If we have already found good matches between A and B, and good matches between B and C, then it is likely that A and C will also match well. As we will see, our search procedure contains moves that exploit this scenario.

Our contributions are three-fold: First, we introduce the notion of a PatchMatchGraph which encodes dense patch-based k-nearest-neighbour matches over entire image sets. Importantly, we cast our graph construction as a formal optimization problem, which allows us to analyze different solution methods. Second, we extend the PatchMatch algorithm of Barnes et al. [2] by introducing new moves that leverage existing good matches in the graph. Third, we show how our PatchMatchGraph can be used for multi-class pixel labeling.

We perform extensive benchmarking of our approach and evaluate multi-class pixel labeling accuracy on four semantic segmentation datasets. We also show how we can adapt to new labeling tasks without re-training.

2 Background and Related Work

Many works use graphs to encode the (spatial or appearance) relationship between regions or key-points within image collections for various tasks including large-scale 3D reconstruction [1], discovery and annotation of common landmarks [6, 14], and content-based image retrieval and exploration [10]. While these works tackle the problem of extracting sparse information at a very large scale, our focus is on providing detailed pixelwise semantic segmentation.

An early work that addressed the problem of data-driven semantic segmentation is the innovative method of Liu et al. [15]. Here a smooth warping from one image to another is estimated using matched SIFT features. Next, labels are

likewise warped and transferred. Similarly, Zhang and Quan [20] propose transferring labels by first matching patches across images using the PatchMatch algorithm [2]. In their work, smoothness is enforced over the image indexes from which labels are being copied. Tighe and Lazebnik [19] also perform label transfer by matching into a small subset of images retrieved from a large corpus based on global descriptors. Their method then matches entire superpixels using a rich set of features. Unlike these works, our work does not require pairwise comparisons between images. Instead, we build a graph representing nearest neighbour patch correspondences and use this graph for label transfer. Moreover, we use very simple colour features during the matching phase.

Our method builds on the PatchMatch algorithm of Barnes et al. [2, 3]. However, these works are primarily focused on image editing tasks whereas our focus is on semantic segmentation. Moreover, we generalize their original pairwise model to a graph-based model and introduce search moves that leverage the graph structure. Other recent works also propose application-specific extensions to the basic set of PatchMatch moves, e.g., Bleyer et al. [4] introduce temporal and view propagation moves in the context of stereo reconstruction.

Locality-sensitive hashing (LSH) [11, 7] provides a more general approach for finding approximate nearest neighbours on a large-scale. However, traditional LSH fails to take into account the smooth structure of images, i.e., adjacent patches have similar appearance, which we exploit in our work.

3 Building the PatchMatchGraph

Our method is based on the PatchMatch approximate nearest neighbour algorithm [2, 3]. Briefly, the algorithm produces a so-called *nearest neighbour field* (NNF) which defines a dense correspondence between all patches of a fixed size in one image with those in a different image. It does this via a randomized search that exploits image smoothness to propagate good match candidates.

Our algorithm extends the generalized PatchMatch algorithm by approximating a k-nearest neighbour field (k-NNF) over an entire graph of images rather than over image pairs. The advantage of our approach is that not all image pairs need to be considered. We call the resulting data structure a PatchMatchGraph, which we define as follows:

Definition 1. *Let $\mathcal{I} = \{I_i\}_{i=1}^{n}$ be a set of n images and let W and H be patch width and patch height parameters, respectively. A* PatchMatchGraph *over images $\mathcal{I}$ is a directed graph $\mathcal{G}^{W\times H}(\mathcal{I}) = \langle \mathcal{V}, \mathcal{E} \rangle$ where each node $u \in \mathcal{V}$ represents a distinct $W \times H$ patch from one of the images in the set and each edge $(u, v) \in \mathcal{E}$ represents a match from patch u to patch v.*

We allow edges $(u, v) \in \mathcal{E}$ to be annotated with a transformation T, which indicates that patch u matches to a transformed version, $T(v)$, of patch v. We denote the annotated edge by the tuple (u, v, T). In our work, we only consider a small set of allowable transformations (e.g., horizontal and vertical flips) that

can be computed quickly. There is no technical reason, however, why a larger set of transformations cannot be considered (e.g., arbitrary rotations and scalings).[1]

We endow the PatchMatchGraph with a cost function $c : \mathcal{E} \to \mathbb{R}$ which scores the quality of matching patch u to the transformed patch $T(v)$. We wish, then, to construct a graph with minimum total cost such that each patch matches to k other patches. Formally, we wish to solve the optimization problem

$$\begin{array}{ll} \text{minimize (over } \mathcal{E}) & \sum_{(u,v,T)\in\mathcal{E}} c(u, T(v)) \\ \text{subject to} & \forall u \in \mathcal{V} : \mathbf{deg}(\mathrm{u}) = k \end{array} \tag{1}$$

where $\mathbf{deg}(u)$ is the out-degree of node u. To encourage a diverse set of matches we impose the following additional restrictions: (i) we do not allow an image to match to itself, and (ii) we do not allow a patch to match to two patches within the same image, i.e., if $(u, v) \in \mathcal{E}$ and $(u, v') \in \mathcal{E}$ are two distinct edges in the graph then patches v and v' must come from different images. Necessarily this requires $k \leq n - 1$. The new optimization problem can be written as

$$\begin{array}{ll} \text{minimize} & \sum_{(u,v,T)\in\mathcal{E}} c(u, T(v)) \\ \text{subject to} & \forall u \in \mathcal{V} : \mathbf{deg}(\mathrm{u}) = k \\ & \forall (u, v, T) \in \mathcal{E} : I_{i(u)} \neq I_{i(v)} \\ & \forall (u, v, T), (u, w, S) \in \mathcal{E} \text{ s.t. } v \neq w : I_{i(v)} \neq I_{i(w)} \end{array} \tag{2}$$

where $I_{i(u)}$ is the image containing the patch represented by node u. We solve this problem approximately using a move-making search procedure over the set of edges (and patch transformations) as described in Section 3.1 below.

The PatchMatchGraph can be efficiently stored and manipulated using arrays of sorted lists for each patch $u \in \mathcal{V}$. The arrays are arranged in a grid for each image $I_{i(u)}$ so that adjacent patches can be indexed quickly. Data structures such as `max-heap` allow the lists to be updated rapidly (see [3] for details). As a simple example, consider a graph $\mathcal{G}$ satisfying the conditions in Equation 2. Now suppose we find a new good match (u, v), i.e., one with low cost $c(u, v)$. Then we can update $\mathcal{G}$ by adding the edge (u, v) to $\mathcal{E}$ and removing the edge $(u, w) \in \mathcal{E}$ with either (i) highest cost, or (ii) where w is from the same image as v (i.e., $I_{i(w)} = I_{i(v)}$). Algorithm 1 summarizes this operation. Importantly, the update maintains the constraints in Equation 2 while decreasing the objective value. It remains to find good match candidates, which we do through a sequence of search moves.

3.1 Search Moves

Our moves, shown schematically in Figure 1, are an extension of the moves proposed by Barnes et al. [3] for the PatchMatch algorithm. Importantly, as we will see below, our approach allows good matches to propagate across different images in the dataset via so-called *enrichment moves*, which was not possible in the pairwise approach of Barnes et al. [3]. We now review the PatchMatch search moves and describe our extensions for building a PatchMatchGraph.

[1] For brevity we will sometimes use the shorter notation (u, v) instead of (u, v, T) when the transformation is not critical to the discussion.

Algorithm 1.. Updating a PatchMatchGraph with a candidate match.

1: **input** $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ and new edge $e = (u, v, T)$
2: Let $e' = \text{argmax}_{(u,v',T') \in \mathcal{E}} \begin{cases} \infty & \text{if } I_{i(v')} = I_{i(v)} \\ c(u, T'(v')) & \text{otherwise} \end{cases}$
3: **if** $c(u, T(v)) < c(u, T'(v'))$ **then**
4: $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{e\} \setminus \{e'\}$
5: **end if**
6: **return** $\mathcal{G}$

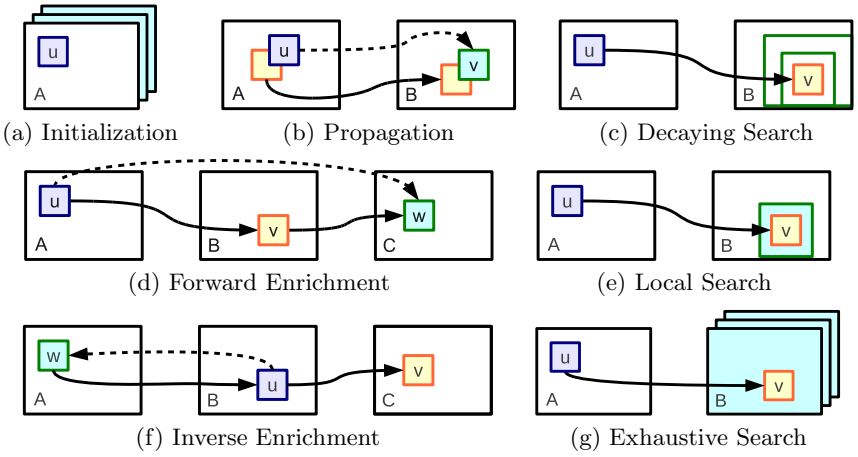


Fig. 1. Search moves executed by our algorithm. See text for details

Initialization (Fig. 1(a)). The initialization step sets up the PatchMatchGraph data structure with random matches and ensures that the constraints in Equation 2 are satisfied. Concretely, we initialize $\mathcal{V}$ to contain every valid patch location from all images in our dataset and $\mathcal{E}$ to the empty set. Then for each node $u \in \mathcal{V}$ we randomly choose k other nodes $v_1, \ldots, v_k \in \mathcal{V}$ and k transforms $T_1, \ldots, T_k$ such that each neighbour v_i comes from a different image. The tuples (u, v_i, T_i) are then added to $\mathcal{E}$. This step is performed only once.

There are many alternative methods for initialization. For example, we could use locality-sensitive hashing (LSH) [11, 7] on patch-based descriptors to find good candidate matches. However, as we show in our experiments random initialization works well and catches up with the more sophisticated LSH alternative after just a few move-making iterations.

Propagation (Fig. 1(b)). The propagation move proposed by Barnes et al. [2] is motivated by the observation that whenever a patch u matches well to patch v then patches spatially adjacent to u (in the image) are likely to match well to patches spatially adjacent to v (in v's image). Concretely, for each edge $(u, v) \in \mathcal{E}$ the propagation move considers all candidate edges (u', v') where u' and v' are 4-connected to u and v, respectively (see Figure 1(b)). When transformations are

applied to v' the relative location and orientation of the pair of adjacent patches is adjusted accordingly. For example, for horizontal flips the patch immediately to the left of u is compared to the patch immediately to the right of v.

Decaying Random Search (Fig. 1(c)). This move is designed to escape local optima by randomly sampling alternative matches. Given a match (u, v) we consider a sequence of candidate patches $\{v_j\}$ sampled from a region around v (in the same image) with exponentially decaying size (see [2] for details). Patch transformations T_j corresponding to each of the v_j are sampled uniformly.

Local Search (Fig. 1(e)). The local search move is a deterministic move, which aims to locally improve a match: Consider an existing edge (u, v, T). Then for all patches v' spatially adjacent to v, edges (u, v', T) are good candidate matches. Barnes et al. [2] did not consider this move since repeating the decaying random search will eventually cover the local neighbourhood. However, experimentally we find that replacing the decaying random search with local search results in significantly faster convergence with only a small degradation in objective value.

Inverse Enrichment (Fig. 1(f)). Inverse enrichment was introduced by Barnes et al. [3] and is motivated by the observation that if v is a good match for u then it is likely that u will be a good match for v. Consequently, in our search procedure, after adding new edges (u, v, T) to the graph (by one of the other search moves), we consider also adding edges (v, u, T^{-1}).

Forward Enrichment (Fig. 1(d)). Similar to the inverse enrichment move we can diffuse good matches forward through the graph. As depicted in Figure 1(d), if there exist edges (u, v, T) and $(v, w, S) \in \mathcal{E}$ then $(u, w, T \circ S)$ is a good candidate edge, where $T \circ S$ represents the composition of transformations T and S. In our work we only attempt one hop in the graph although longer paths could also be considered at additional computation cost. To prevent cycles (and enforce our graph constraints) we skip edges that match back to the same image.

Exhaustive Search (Fig. 1(g)). The above moves are designed to exploit good matches by propagating them across images and diffusing them throughout the PatchMatchGraph. However, without sufficiently many good initial matches the move-making algorithm could converge to a poor solution. We can avoid this by seeding the graph with a small number of very good matches. This move randomly samples a patch u and exhaustively search for the k nearest neighbours for that patch. Since we are only considering one patch at a time this search step is fast and can be very effective at escaping from local optima. The patch u can be sampled with probability inversely proportional to the score of its best current match. This has the added benefit of explicitly removing poor matches.

3.2 Search Procedure

The complete search algorithm can be summarized as follows: First, we initialize the PatchMatchGraph with random matches such that the constraints of

Equation 2 are satisfied. We then iterate through each of the search moves to progressively improve the matches. We terminate our search after a maximum number of iterations or when very few improved matches are found (for example, when none of the moves, other than exhaustive search, find better matches).

We skip certain moves if they involve a match that has not changed since the last iteration of the algorithm. For example, the local search move does not need to be repeatedly run on the same edge (u, v). Moreover, we choose the order of the moves to ensure that good matches propagate quickly through the graph. For example, following the recommendation of Barnes et al. [2], we evaluate the propagation move in forward and then reverse raster scan order.

It is also straightforward to introduce images incrementally or only consider a subset of the images during a search iteration. For example, if we have already found good matches for all the patches in an image computational resources can be diverted to finding better matches for patches in other images. Moreover, the moves do not need to be executed serially. In fact, it is trivial to distribute the search over a number of parallel machines. We discuss a number of other extensions and variants of our approach in Section 3.4 below.

Without the exhaustive search our approach is not guaranteed to convergence to the global optimum when the number of nearest neighbours k maintained by the PatchMatchGraph is less than $n-1$ (the size of the dataset less one) even if run indefinitely. This is because certain matches (i.e., image pairs) may never be considered. Nevertheless a global solution is not necessary for most tasks and our aim is rather to find good approximate solutions quickly.

3.3 Cost Function

Our method requires a cost function for evaluating the quality of a match between two patches. Let $f_u \in \mathbb{R}^d$ denote the feature vector describing the patch associated with node u. For example, f_u could be formed by concatenating the RGB colour components for every pixel in the patch (in fixed row-wise or column-wise order). In this case f_u will have dimension $d = 3WH$ for a $W \times H$ patch. We can now define a number of common cost functions such as sum-of-absolute-differences (SAD), $c^{\text{SAD}}(u, v) = \|f_u - f_v\|_1$, or sum-of-square-differences (SSD), $c^{\text{SSD}}(u, v) = \|f_u - f_v\|_2^2$ between two nodes u and v.[2]

3.4 Extensions and Variations

There are a number of variations to the method for constructing a PatchMatchGraph described above. For example, scale invariance can be addressed by including scale transformations T which implicitly resize patches during the matching step. In our implementation, we prefer to use image pyramids and treat each level of the pyramid as a separate image but forbid matches between different levels in the same pyramid (i.e., matches between an image and itself).

[2] In practice we do not store the feature vectors f_u and f_v explicitly. Rather we compute $c(u, v)$ directly from the image data.

Another variation is to impose a different degree constraint for each node in the graph, i.e., $\mathbf{deg}(u) = k_u$. This could be beneficial, for example, in concentrating computation effort on patches which are more discriminative (e.g., contain distinctive patterns) and therefore likely to lead to better labeling accuracy.

4 Using the PatchMatchGraph for Label Transfer

Our PatchMatchGraph can be used to solve multi-class pixel labeling problems by transferring annotations from labeled images to those without labels. We assume that each image I_i has an associated label matrix L_i, which contains labels for the pixels in the image. We denote the label at pixel p by $L_i(p)$. In the case of an unlabeled image $L_i(p)$ is assigned the special label "unknown".

In contrast to superpixel-based methods, a patch may contain pixels of different classes. Therefore we transfer labels in a pixelwise fashion. Let $\mathcal{P}_u$ be the set of pixels associated with patch $u \in \mathcal{V}$ and let $m_{u \to T(v)} : \mathcal{P}_u \to \mathcal{P}_v$ define the mapping from pixels in $\mathcal{P}_u$ to pixels in $\mathcal{P}_v$ under transformation T. For each pixel p in the target image define $\mathcal{E}_p = \{(u, v, T) \in \mathcal{E} \mid p \in \mathcal{P}_u\}$ to be the subset of edges such that p is contained within the patch u at the source of the edge. We can now estimate the probability for an unknown pixel p to take label ℓ as

$$\mathrm{P}\left(y_p = \ell \mid \mathcal{G}\right) \propto \sum_{(u,v,T) \in \mathcal{E}_p} w_p^{\ell}(u, T(v)) \left[\!\left[L_{i(v)}\left(m_{u \to T(v)}(p)\right) = \ell \right]\!\right] \tag{3}$$

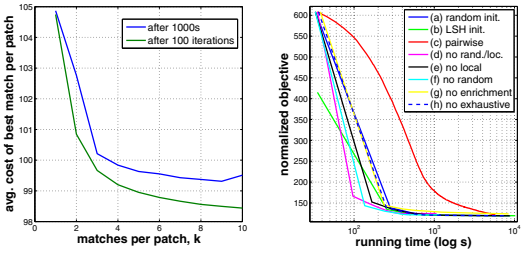
where $L_{i(v)}(q)$ is the label for pixel q in the image containing patch v, and $[\![\cdot]\!]$ is the indicator function taking value one if its argument is true and zero otherwise. Here, $w_p^{\ell}(u, T(v))$ is a pixelwise weighting function that regulates the contribution from each match. For example, we may want to weight matches by a function of their cost. In our work, we simply weight matches by $\frac{1}{r^2}$ where r is the rank of the match in the sorted list of matches for node u.

It is common in pixel-labeling problems to smooth pixelwise class estimates via an conditional Markov random field (CRF). Briefly, for each pixel p, let $f_p \in \mathbb{R}^L$ be the vector of log-probabilities for each label (from Equation 3).[3] Then energy for a labeling $\boldsymbol{y}$ over a full image can be written as a combination of unary and contrast-sensitive pairwise smoothing terms:

$$E(\boldsymbol{y} \mid \mathcal{G}) = \sum_p \sum_\ell [\![y_p = \ell]\!] \theta_\ell^T f_p + \lambda \sum_{pq} [\![y_p \neq y_q]\!] \exp\left(-\frac{\|x_p - x_q\|^2}{2\beta} \right) \tag{4}$$

where θ_ℓ are parameters which calibrate the probability estimates for each class and $\lambda \geq 0$ controls the strength of the pairwise smoothness term. Here the pairwise term is over adjacent pixels p and q in the image, x_p and x_q are colour vectors for the pixels, and β is the average square-distance between colour vectors for adjacent pixels in the image (see [18] for details). The parameters θ_ℓ and λ can be learned from a subset of annotated images in the PatchMatchGraph.

[3] Feature vector f_p could be extended arbitrarily to include other features such as the output of boosted classifiers or location priors. We explore some of these in our experiments.



Strategy	1000s	Final†
(a) Random init.	120.7	**119.4** (2:28)
(b) LSH init.	**120.4**	**119.4** (2:28)
(c) Pairwise	178.8	121.7 (1:36)
(d) No rand./local	124.7	124.7 (0:17)
(e) No local	121.2	**119.3** (2:25)
(f) No random	121.5	121.5 (0:19)
(g) No enrichment	128.8	123.9 (2:20)
(h) No exhaustive	120.5	**119.4** (2:15)

†After considering all pairs for the pairwise strategy or 100 iterations for all other strategies. Parentheses show running time (H:MM).

Fig. 2. Evaluation of different search strategies for constructing a PatchMatchGraph. The strategies refer to moves discussed in Section 3.1. Plotted are average cost of best match versus number of nearest neighbours (left) and normalized objective versus running time (right).

5 Experimental Results

We perform a number of experiments to benchmark our algorithm and evaluate its performance on the multi-class pixel labeling task. All experiments are run on a single core of a 2.93GHz Intel Nehalem CPU with 24GB RAM. Unless otherwise stated we fix the patch size to 8×8 pixels, the number of nearest neighbours, k, to ten, and use the sum-of-absolute-differences (SAD) cost function.

5.1 Benchmarking

Our benchmarking experiments are aimed at evaluating different search strategies for building our PatchMatchGraph. Specifically, we are interested in the trade-off between running time and objective value (of Equation 2). In this experiment we use the 80 training images from the Polo dataset [20] scaled to 50% of their original size. Features are computed over CIELab colour space.

Figure 2 contains plots comparing the objective value for a number of move-making strategies. For convenience we scale the objective value by the total number of patches to get an average score per patch. The left plot shows the average cost of the best matching patch as a function of out-degree, k. As expected we find better matches (i.e., lower cost) as the search space increases (i.e., larger k). However, this comes at high computational expense, e.g., it took 18 minutes for 100 iterations with $k = 1$ versus 2 hours and 28 minutes for $k = 10$.

The second plot and table show objective value versus running time for different search moves with k fixed at ten. Here we see some interesting results. First, performing **pairwise** nearest-neighbour search offers no advantage over searching for the k-nearest neighbours simultaneously—it is much slower and does not find better matches. The reason is that the pairwise search cannot diffuse good matches to other images in the dataset.

Second, the **decaying random search** move is the most expensive in terms of running time. Moreover, it helps little when combined with the other moves (i.e., resulting in an average match score of 119.4 versus 121.5 when removed at

over seven times the computational cost). The **decaying random search** if run long enough will encompass all the moves proposed by the **local search** move. However, in terms of running time **local search** is far superior. Interestingly, removing the **local search** move results in slightly better matches. We surmise that this is due to getting trapped in a local optima early in the search procedure.

Third, as expected **locality-sensitive hashing** provides much better initialization than random guessing. However, the algorithm converges to the same objective value with both initialization approaches. This demonstrates that taking image smoothness into account is vital for fast nearest neighbour search.

Last, **exhaustive search**, while guaranteed to find optimal matches, does not noticeably improve the objective. This is partly due to the high-quality matches already discovered by randomized search and the large number of patches dominating the objective value relative to the few exhaustive search moves.

5.2 Semantic Segmentation

We evaluate the multi-class labeling accuracy of our method on four challenging semantic segmentation datasets. In these experiments we augment the CIELab colour feature vector for each pixel with an "edge" feature which captures the contrast between the pixel and its neighbours. This puts emphasis on matching well along the boundaries of objects. We also include the (normalized) vertical location of the pixel in the image to provide some spatial context. The five features are illustrated in Figure 3 on two representative images.

For each dataset, we build a PatchMatchGraph over all images for $k = 10$ and $k = 20$. We include resized copies of each image to allow for scale-invariant matches. These are derived from an image pyramid which starts at 50% of the original image size and reduces each layer by a factor of $\sqrt{2}$ down to a minimum size of 32×32 pixels. Based on the benchmarking results from Section 5.1 we chose not to use the **decaying random search** or **exhaustive search** moves.

We divide each dataset into train and test subsets following the standard practice in the literature for the dataset. When building the PatchMatchGraph we disallow matches between images within the test subset. Labels are then transferred from training images to the test images. Our experimental results (averaged over the test subsets) are summarized in Table 1 and discussed for each dataset below. Figure 5 shows some example results.

We include results for two variants of the CRF model (see Equation 4). The first variant simply uses the CRF to calibrate probabilities and smooth over neighbouring pixels as discussed in Section 4. The second variant, which we call the Boosted CRF, incorporates discriminatively trained unary potentials. Briefly, for each class we learn a one-versus-all boosted decision tree classifier on the vector of log-probabilities f_p and local pixel features. Here, we set the decision tree depth to two and train for a maximum of 100 rounds. We then use the output of the boosted classifier as the unary features in our CRF model.

6-Class Polo Dataset [20]. This dataset contains 317 unique images of polo scenes. The task is to label pixels from the set: *grass*, *ground*, *horse*, *person*, *sky*,

Fig. 3. Features used to compare patches. Shown are the original RGB colour image for two images from the Polo dataset [20] and the per-pixel feature values for each channel of the CIELab colour space, edge feature and vertical position feature.

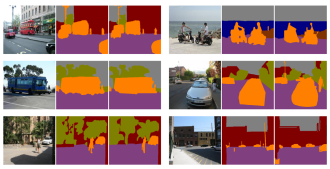


Fig. 4. Instances of repeated images in the Stanford Dataset [8] found by our algorithm with their (non-identical) ground truth labels.

or *tree*. Pixels not falling into these categories are labeled as *void* in the ground truth and ignored during test. This is a relative easy dataset with state-of-the-art methods achieving pixelwise accuracies around 89%. Our simple approach without the CRF is competitive with the current state-of-the-art and with CRF smoothing beats it by about 3%. Moreover, our "training time", i.e., graph construction, is only 20 minutes. Test time is around four seconds per image. Doubling the number of nearest neighbours, k, from 10 to 20 results in little improvement (as supported by Figure 2). Finally, the discriminatively learned Boosted CRF improves results further but at high computational cost.

Table 1. Summary of results from our semantic segmentation experiments. Values in parentheses indicate class-averaged accuracy. For reference, we include published state-of-the-art results where available.

			Stanford BG [8]			
		Polo [20]	Semantic	Geometric	MSRC [5]	LMO [16]
Number of Classes		6	8	3	21	33
Number of Images		317	715		591	2688
Train/Test Split		80/237	572/143		335/256	2488/200
Number of Patches		5031477	22635766		17154995	39355008*
$k = 10$	Graph Build (H:MM)	0:20	2:29		1:32	5:36
	Accuracy	89.6 (82.3)	70.3 (56.6)	89.1 (89.2)	62.8 (49.1)	59.6 (18.4)
	CRF Accuracy	92.2 (89.3)	71.4 (60.2)	89.1 (88.7)	71.1 (64.7)	65.7 (14.2)
	Boosted CRF Acc.	93.9 (91.4)	73.9 (63.2)	88.0 (87.9)	78.6 (72.8)	65.2 (14.9)
$k = 20$	Graph Build (H:MM)	0:46	6:23		4:00	exceeds
	Accuracy	89.9 (82.9)	70.7 (57.0)	89.2 (89.2)	63.3 (49.8)	24GB
	CRF Accuracy	92.6 (89.7)	71.9 (60.7)	89.3 (88.7)	72.5 (66.4)	memory
	Boosted CRF Acc.	**94.2** (91.7)	73.4 (62.0)	88.2 (88.1)	79.0 (72.8)	limit
Zhang and Quan [20]		89.8 (82.5)	–	–	–	–
Shotton et al. [18]		83.9 (77.1)†	–	–	72.2 (57.7)	51.7‡
Gould et al. [8]		–	76.4	**91.0**	76.4	–
Tighe and Lazebnik [19]		87.9 (76.1)†	**77.5**	90.6	–	**76.9** (29.4)
Krahenbuhl and Koltun [12]		–	–	–	**86.0** (78.3)	–
Ladicky et al. [13]		–	–	–	**86.0** (75.0)	–
Liu et al. [16]		–	–	–	–	76.7

* Due to the size of this dataset we did not match over image pyramids.
† Reported by Zhang and Quan [20].
‡ Reported by Liu et al. [16] (without Markov random field).

21-Class Microsoft Research Cambridge (MSRC) Dataset [18, 5]. This dataset contains 591 roughly annotated images and has been the de facto standard for evaluating semantic segmentation algorithms for a number of years. The images contain a mix of 21 different foreground and background classes. We achieve 71.1% accuracy after less than two hours of computation. This is comparable to many multi-class segmentation methods albeit well below state-of-the-art. However, our method is at least an order of magnitude faster. On this dataset CRF smoothing improves both quantitative and qualitative results.

8-Class Stanford Background Dataset [8]. This dataset contains 715 images and provides two distinct label sets. We construct a single PatchMatchGraph and use the same graph to label images with either one of eight semantic classes or one of three geometric classes. The semantic classes include seven background categories and a generic foreground category. The geometric classes are *vertical*, *horizontal*, and *sky*. Quantitative results are shown in Table 1.

Surprisingly, on this dataset the CRF did not help much due to the relatively low intra-class variation at the pixel level (whole regions, e.g., water and sky, are sometimes confused) and some smoothness already imposed at the patch level. In fact, the CRF can often degrade results by over-smoothing (see Figure 5).

We can use the patch-wise nearest neighbours returned by our algorithm to determine global similarity between images. Interestingly, this allowed us to find six instances of repeated images in the Stanford Background Dataset that have not previously been reported (see Figure 4).[4] Since annotations for these images were obtained independently the corresponding ground truths differ slightly.

33-Class LMO Dataset [16]. This is a large dataset consisting of 2688 images and annotated with 33 labels. Due to the size of the dataset we did not search over image scales. Moreover, running with $k = 20$ exceeded the available memory on our hardware. This, however, is a limitation of our implementation, which holds the entire graph and all images in memory at the same time. A better implementation could (i) use a database back-end for storing the PatchMatchGraph, and (ii) make use of distributed computation.

Our results are encouraging but below state-of-the-art methods. We believe that this is due to the more sophisticated features and contextual information introduced by the image retrieval step used in those methods. Interestingly, the Boosted CRF does not improve accuracy. This is in part due to confusion between visually similar and related semantic classes, e.g., river and sea, and the weak features available to the boosted classifiers.

6 Discussion

In this work, we have proposed a method for non-parametric label transfer by building a graph of dense and overlapping patch correspondences between images in a dataset. This data-driven approach has several advantages over model-based

[4] The images are `1000875` and `9000875`, `1000288` and `9000288`, `1001184` and `9001184`, `6000117` and `6000324`, `6000068` and `6000318`, and `6000257` and `6000348`.

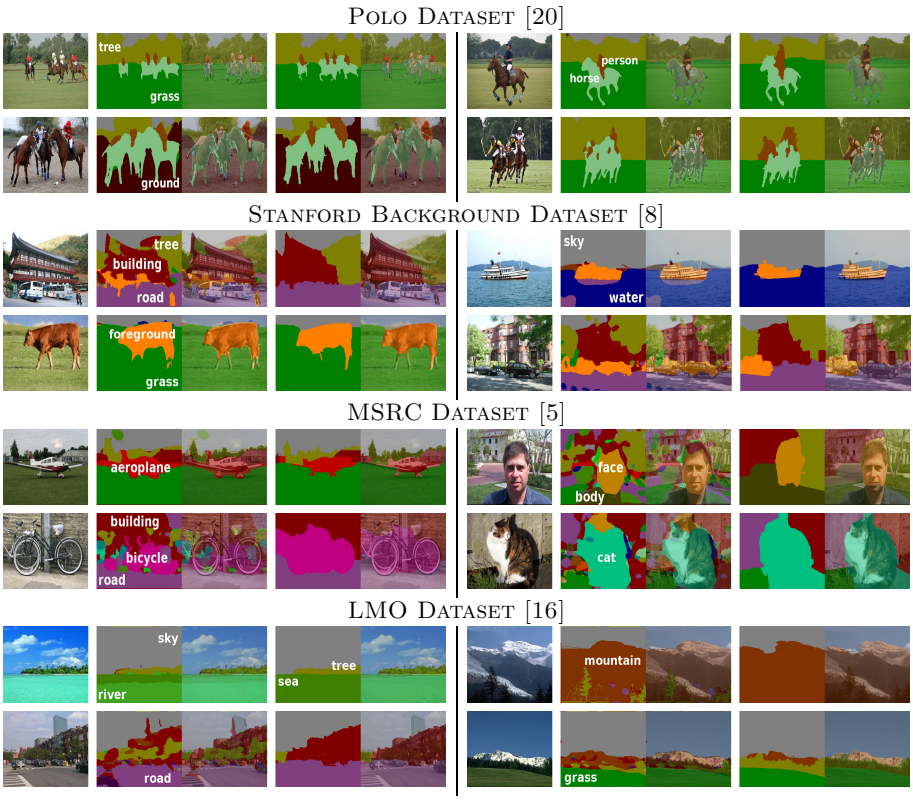


Fig. 5. Example results from four different datasets. Shown are the image, results from our PatchMatchGraph label transfer and with CRF smoothing. Results are shown as colour coded images and image overlays. Best viewed in colour.

approaches including not expending effort on modeling the appearance of object classes and supporting life-long learning through incremental addition of labeled data. Moreover, this approach allows us to change labeling tasks (e.g., from semantic labels to geometric labels) with no changes to the model.

Compared with existing non-parametric label transfer techniques our graph-based approach offers the advantage of not requiring pairwise image comparisons. This makes our method independent of the size of the labeled dataset at test time and avoids the initial image retrieval step. However, scene level information is clearly an important source of context that we are currently missing.

We are excited about a number of extensions of our work. First, our search procedure is embarrassingly parallelizable and we intend to assess the scalability of our method to very large datasets by distributing computation. Second, we are currently restricted to rectangular patches but would be interested in exploring the use of superpixels (or combinations of superpixels) for defining regions. Last, we currently only use the graph structure for finding nearest neighbour matches. It would be interesting to see whether we could exploit the graph structure at test time, e.g., to estimate the quality of each match during semantic label transfer.

Acknowledgments. This work was supported by the Australian Research Council and the NCI National Facility at the ANU.

References

1. Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building rome in a day. In: ICCV (2009)
2. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: PatchMatch: A randomized correspondence algorithm for structural image editing. In: SIGGRAPH (2009)
3. Barnes, C., Shechtman, E., Goldman, D.B., Finkelstein, A.: The Generalized PatchMatch Correspondence Algorithm. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part III. LNCS, vol. 6313, pp. 29–43. Springer, Heidelberg (2010)
4. Bleyer, M., Rhemann, C., Rother, C.: PatchMatch stereo—stereo matching with slanted support windows. In: BMVC (2011)
5. Criminisi, A.: Microsoft Research Rcambridge (MSRC) object recognition pixel-wise labeled image database (version 2) (2004)
6. Gammeter, S., Bossard, L., Quack, T., Gool, L.V.: I know what you did last summer: object-level auto-annotation of holiday snaps. In: ICCV (2009)
7. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing, pp. 518–529 (1999)
8. Gould, S., Fulton, R., Koller, D.: Decomposing a scene into geometric and semantically consistent regions. In: ICCV (2009)
9. He, X., Zemel, R.S., Carreira-Perpinan, M.: Multiscale conditional random fields for image labeling. In: CVPR (2004)
10. Heath, K., Gelfand, N., Ovsjanikov, M., Aanjaneya, M., Guibas, L.J.: Image Webs: Computing and exploiting connectivity in image collections. In: CVPR (2010)
11. Indyk, P., Motwani, R.: Approximate nearest neighbor—towards removing the curse of dimensionality. In: 30th Symp. of Theory of Comp., pp. 604–613 (1998)
12. Krahenbuhl, P., Koltun, V.: Efficient inference in fully connected CRFs with gaussian edge potentials. In: NIPS (2011)
13. Ladicky, L., Russell, C., Kohli, P., Torr, P.H.: Associative hierarchical CRFs for object class image segmentation. In: ICCV (2009)
14. Li, Y., Crandall, D., Huttenlocher, D.: Landmark classification in large-scale image collections. In: ICCV (2009)
15. Liu, C., Yuen, J., Torralba, A.: Nonparametric scene parsing: Label transfer via dense scene alignment. In: CVPR (2009)
16. Liu, C., Yuen, J., Torralba, A.: Nonparametric scene parsing via label transfer. PAMI 33(12), 2368–2382 (2011)
17. Oliva, A., Torralba, A.: Modeling the shape of the scene: a holistic representation of the spatial envelope. IJCV 42(3), 145–175 (2001)
18. Shotton, J., Winn, J., Rother, C., Criminisi, A.: *TextonBoost*: Joint Appearance, Shape and Context Modeling for Multi-class Object Recognition and Segmentation. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 1–15. Springer, Heidelberg (2006)
19. Tighe, J., Lazebnik, S.: SuperParsing: Scalable Nonparametric Image Parsing with Superpixels. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part V. LNCS, vol. 6315, pp. 352–365. Springer, Heidelberg (2010)
20. Zhang, H., Quan, L.: Partial similarity based nonparametric scene parsing in certain environment. In: CVPR (2011)