Local Descriptors Optimized for Average Precision
Supplementary Material

A. Learning Real-Valued Descriptors

We model the mapping from image patches to descriptors as $F : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is the descriptor space, and F is a neural network. With real-valued descriptors, we take $\mathcal{Y} = \mathbb{R}^m$. In the paper, the approximate gradients for histogram binning are given as

$$\frac{\partial h_k^+}{\partial F(q)} \approx \sum_{x \in S_q^+} \frac{\partial \delta(D(q,x),k)}{\partial D(q,x)} \frac{\partial D(q,x)}{\partial F(q)}, \tag{1}$$

$$\frac{\partial h_k^+}{\partial F(x)} \approx \mathbf{1}[x \in S_q^+] \frac{\partial \delta(D(q,x),k)}{\partial D(q,x)} \frac{\partial D(q,x)}{\partial F(x)}, \tag{2}$$

where q is a query patch and S_q^+ is the set of its matches in the database, and D is the distance metric being learned.

In the real-valued case, the descriptor F is modeled as a vector of neural network activations, with L_2 normalization:

$$F_0(x) = (f_1(x;w), f_2(x;w), \ldots, f_m(x;w)) \in \mathbb{R}^m, \quad F(x) = \frac{F_0(x)}{\|F_0(x)\|}. \tag{3}$$

D is now the Euclidean distance between unit vectors, whose partial derivative $\partial D / \partial F$ is

$$\frac{\partial D(x,x')}{\partial F(x)} = \frac{\partial \sqrt{2 - 2F(x)^\top F(x')}}{\partial F(x)} = \frac{-F(x')}{D(x,x')}. \tag{4}$$

Lastly, backpropagation through the L_2 normalization operation is as follows:

$$\frac{\partial h_k^+}{\partial F_0(x)} = \frac{1}{\|F_0(x)\|} \left[\frac{\partial h_k^+}{\partial F(x)} - F(x) \left(F(x)^\top \frac{\partial h_k^+}{\partial F(x)} \right) \right]. \tag{5}$$

B. Spatial Transformer Module

We use the Spatial Transformer module in our networks to handle geometric noise and align input patches. As is standard practice, the Spatial Transformer is initialized to output identity (directly copy input patches), and the learning rate of the affine transformation prediction layer is scaled down by 100x compared to other layers in the network.

A naïve application of the Spatial Transformer, however, leads to the boundary effect [7]: when the predicted transformation requires sampling outside the boundaries of the input, the default zero-padding creates unfilled boundaries in the output. Since the input patches to the Spatial Transformer have limited size (42x42 in our network), out-of-boundary sampling frequently happens in operations such as zooming out and rotation, and can affect alignment by introducing unwanted image gradients. Instead, we first pad the input patch by repeating its boundary pixels,[1] and then sample according to the predicted transformation, which prevents sharp gradients near boundaries. This is visually illustrate in Fig. 1, using patches from the challenging HPatches dataset, which has the largest amount of geometric noise among the datasets that we consider. Although using zero padding still produces decent alignment, it affects the appearance of sampled patches, and does not help to improve final performance. Our boundary padding produces much more visually plausible patches, and gives a good approximation to re-sampling from the original images.

[1] Implemented in Matlab using the `replicate` mode of the `padarray` function.

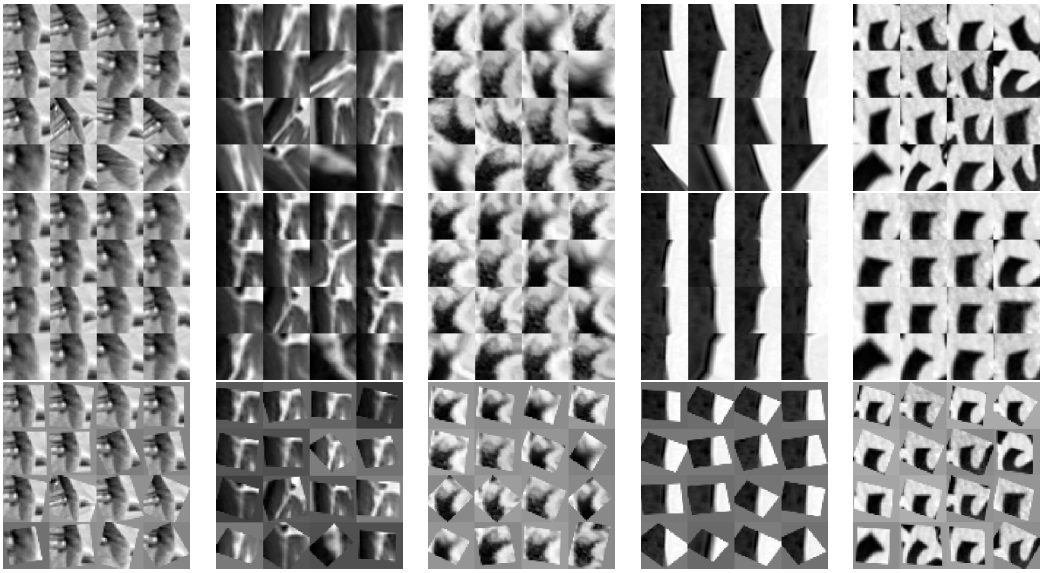

Figure 1. Alignment using the Spatial Transformer in HPatches, where patches come in groups of 16. The aligned patches are used as inputs to the descriptor network. First row: original patches. Second row: aligned patches, using our boundary padding. Third row: aligned patches, using the default zero padding.

C. Label Mining in HPatches

As mentioned in Sec. 4.2, in the patch retrieval task in HPatches, the set of distractors for each query only consists of out-of-sequence patches. This differs from the image matching task where all distractors are in-sequence. We use clustering to supply in-sequence distractors when optimizing patch retrieval performance.

C.1. Clustering

Since the 3D point correspondence for each training patch is given, it may appear that we can simply mark all patches that do not correspond to a certain 3D point as distractors for the corresponding patch. However, the risk is that when an image has repeating structures (*e.g*. windows on a building), patches that correspond to different 3D points could have nearly identical appearance, and forcing the network to distinguish between them would cause overfitting. Instead, we need a mechanism to mark distractors only when the appearance difference is above a threshold. Our solution is to use clustering: given an image sequence, we cluster all patches from this sequence by visual appearance. Then, a threshold is put on the inter-cluster distances to determine distractors.

We use handcrafted visual features to represent patches in clustering. The best feature found in our experiments is a combination of HOG [4] and raw pixel values, which captures both the geometric and illumination patterns. It is constructed as follows: a patch is resized to 64x64 to extract HOG features with 8x8 cell size, and then the same patch is resized to 16x16 and appended to the feature vector. The final feature dimensionality is 2240. Afterwards, we perform K-means clustering with $K = 100$ clusters. To derive a distance threshold, we compute all the pairwise distances between the cluster centers, and set the threshold at the p-th percentile of these distances. If two clusters have larger distance than the threshold, their patches are considered distractors for each other. Otherwise, they are considered "too visually similar", and are ignored from each other's distractor set. We use $p = 20$. Label mining is demonstrated in Fig. 2.

C.2. Minibatch Sampling

There are 76 image sequences in the training set of HPatches. Without label mining, we uniformly sample patch groups from all sequences to construct training minibatches, so on average only about 1/76 of the patches in each minibatch are from the same sequence. In this case, even if the in-sequence distractor labels are known, their contribution to the gradients is limited. Therefore, we use a modified minibatch sampling strategy when label mining is in effect, so that more patches from the same sequence are placed in a minibatch.

Specifically, to construct a minibatch, we first sample two image sequences. Then, an equal number of patch groups (each containing 16 matching patches) are sampled from each sequence. For example, if batch size $M = 1024 = 64 \times 16$, then

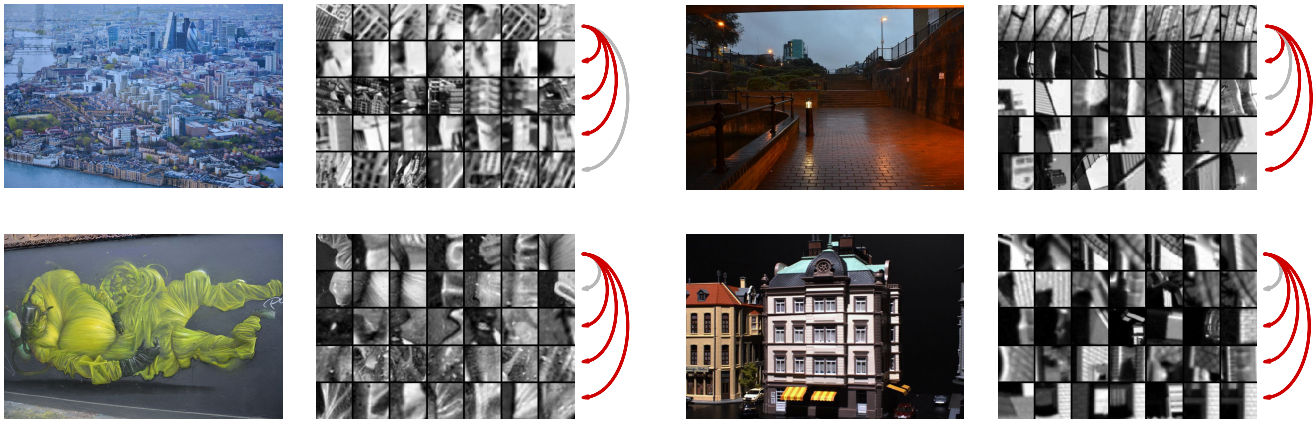

Figure 2. We demonstrate label mining in HPatches, using four randomly selected image sequences. First row: `v_london`, `i_steps`. Second row: `v_maskedman`, `i_yellowtent`. The first image in each sequence is shown on the left, and on the right we visualize 5 randomly selected patch clusters, obtained using K-means. Each row corresponds to a cluster. A red arrow between clusters indicates that the inter-cluster distance is above a threshold, and their patches are used as distractors for each other. A gray arrow means that the inter-cluster distance is not high enough. Patches are generally more similar in appearance within the same sequence than across sequences, therefore mining the in-sequence distractors provides meaningful "hard negatives" for the learning.

32 groups are sampled from each of the two sequences. This way, for each patch, roughly half of its distractors are out-of-sequence patches, and the other half are in-sequence, which are generally harder to distinguish. This simple heuristic gave about 6% absolute improvement in image matching mAP in our experiments, and we did not specifically tune the ratio of in-sequence *vs*. out-of-sequence distractors. With this strategy, a minibatch involves a pair of sequences, and a training epoch loops over all the $\frac{76\times(76-1)}{2} = 2850$ pairs, and takes less than 10 minutes with $M = 1024$ in our GPU implementation.

D. Experimental Details

We train our networks from scratch using SGD. The initialization scheme proposed in [6] is adopted, since the architecture uses ReLU activations. Through validation experiments, we found that an initial learning rate of 0.1 works well with batch size $M = 1024$ in all datasets. For other batch sizes, we scale the learning rate linearly, according to the suggestion in [5]. For UBC Phototour, inspired by HardNet [1], the learning rate is decreased linearly to zero within 100 epochs. For HPatches, we actually found a more traditional strategy to work better: we use a constant learning rate and divide it by 10 every 10 epochs, for 32 epochs total.

For RomePatches, the training set has 10,000 patches, or 1,000 groups of 10 patches, which is quite small. To stabilize the training, we increase the number of minibatches in each epoch to 1,000 as follows: the k-th batch first includes the k-th group, and then randomly samples other groups to fill the batch. With this strategy, each epoch processes the training set multiple times, and we found 5 epochs to be sufficient to ensure convergence.

Our implementation uses MatConvNet [9]. For competing methods, we use the publicly released models/implementations.

- Pretrained L2Net models are obtained from [2]. We use the versions trained with data augmentation.
- Pretrained HardNet models are obtained from [1]. We use the versions trained with data augmentation.
- For SIFT and LIOP, we use the implementation in VLFeat [8].

Performance on HPatches is evaluated using the HPatches benchmark [3]. For the image matching experiment in Oxford dataset, the detection of interest points and extraction of patches are performed using the `vl_covdet` function in VLFeat, with the `PatchRelativeExtent` parameter set to 3.

References

[1] `https://github.com/DagnyT/hardnet`.

[2] `https://github.com/yuruntian/L2-Net`.

[3] `https://github.com/hpatches/hpatches-benchmark`.

[4] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[5] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2015.

[7] Chen-Hsuan Lin and Simon Lucey. Inverse compositional spatial transformer networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[9] Andrea Vedaldi and Karel Lenc. MatConvNet – convolutional neural networks for MATLAB. In *Proc. ACM Conference on Multimedia*, 2015.