Chapter 6

Learning Image Patch Similarity

The ability to compare image regions (*patches*) has been the basis of many approaches to core computer vision problems, including object, texture and scene categorization. Developing representations for image patches has also been in the focus of much work. However, the question of appropriate similarity measure between patches has largely remained unattended.

In this chapter we focus on a specific case study: learning similarity of natural image patches under arbitrary rotation and minor shift. Figure 6-1 shows a few examples of groups of patches that are similar to each other. Such a definition may not be broad enough, that is, some patches that do not fit this description still may be considered similar in the context of a given task. For instance, if the goal is to categorize an object, rather than recognize a specific instance, patches that look visually dissimilar at this low lever may still correspond to semantically matching parts.[1] However, we believe that for many reasonable tasks that involve matching patches, the underlying definition of similarity must include this limited case.

Rather than developing a new representation that is invariant to the transformation in question, we will consider two popular representations of patches: sparse overcomplete code and scale-invariant feature transform (SIFT). Specifically, we will investigate how the tools developed in this thesis can be used to improve matching with these representations.

Section 6.1 provides some background on algorithms that use patch matching, and reviews approaches to measuring patch similarity. We define the target similarity concept in Section 6.2, and the patch descriptors used in the experiments are introduced in Section 6.3. We then describe, in Section 6.4.2 an experimental evaluation that demonstrates an improvement in matching with both of the descriptor types by using a similarity-driven embedding learned with BoostPro.

[1] Note that the framework presented in this chapter can be extended to learn such a similarity as well, as long as examples are provided.

6.1 Background

The main context in which comparing image patches has emerged in computer vision is that of high-level vision tasks, that can be described as scene understanding. This includes:

Object recognition This means finding a specific object: a face of a certain person, a shoe of a particular make, a magazine etc.

Object categorization and object class detection Rather than looking for a specific instance of an object, the interest here is in all objects that belong to a certain class, for an appropriate definition of the latter: any face, any car, etc.

Entire image classification Sometimes the goal is not to localize, or determine the presence of, an object but rather to assign the entire image to a certain class. For instance, location recognition and texture classification belong to this category of tasks.

Broadly speaking, approaches to these tasks can be divided into three groups with regards to how they represent image information. The first group consists of methods that rely on *global features*. This may include methods that collect a histogram of measurements over an entire image [82, 101], or shape matching techniques that directly model an entire shape in a parametric form [104, 74].

The second group consists of methods that operate on local image features, but do not directly operate on image patches. This includes methods that compute histograms of measurements over a limited region, be it measurements of shape [11, 55] or filter responses [115, 101].

Finally, the third group, of most relevance here, directly operates on image patches. Most of the methods in this group involve, in addition to matching the patches, a geometrical reasoning component. This component may involve a full model of joint location of parts as in constellation models [46, 71], or a more loose set of constraints like in random field models [91], or fragment-based approaches [3, 9]. Some methods that have been proposed avoid modeling geometry altogether [107].

It should be emphasized that in addition to recognition and classification, other tasks may benefit from patch matching paradigm. Notable examples are fragments-based segmentation [17] and wide-baseline stereo reconstruction [81].

6.1.1 Patch similarity measures

The question of measuring similarity between patches has not received very much attention in the computer vision literature. Usually, a standard distance measure is adopted for whatever representation is used.

Pixel-based distance

The simplest similarity measure consists of directly comparing the pixel values of the two regions, e.g. by means of the L_1 distance

$$D(\mathbf{x}_1, \mathbf{x}_2) = \sum_d |\mathbf{x}_{1(d)} - \mathbf{x}_{2(d)}|.$$

This is rarely a useful measure, since it is extremely sensitive to minor transformations, both in geometry (shifts and rotations) and in imaging conditions (lighting or noise).

Correlation

Normalized correlation between patches $\mathbf{x}_1$ and $\mathbf{x}_2$ is defined as

$$NC(\mathbf{x}_1, \mathbf{x}_2) = \frac{\sum_d \left(\mathbf{x}_{1(d)} - \bar{\mathbf{x}_1}\right)\left(\mathbf{x}_{2(d)} - \bar{\mathbf{x}_2}\right)}{\sigma_1 \sigma_2},$$

where $\bar{\mathbf{x}_i}$ and σ_i are the mean and standard deviation of pixels in $\mathbf{x}_i$. Because of the factoring in of the means it is much more robust than the pixel-wise distance. Normalized correlation has been used extensively in fragment-based recognition [3, 9], where it is assumed that viewing conditions are fixed, or alternatively that there exist examples from all viewing conditions–in other words, not matching a patch to a version of itself rotated by 90 degrees is acceptable. We would like to avoid such an assumption.

Descriptor distance

Another popular method is to compute a descriptor of each patch, and then simply apply a distance measure on the two descriptors. Most commonly the descriptors are vectors in a metric space of fixed dimensions and the distance of choice is L_1. Matching with SIFT descriptors, discussed in detail in Section 6.3.2 is perhaps the most popular example of such an approach [76]. Another popular descriptor is the *shape context* [11], often used when shape is believed to be the crucial component of the recognition system. Shape contexts are based on the local histogram of contour points in the vicinity of the selected location. The distance of choice for shape contexts is typically χ^2.

Probabilistic matching

A different approach is taken by some of the methods that instead of measuring distance between representations patches, evaluate directly the probability that the two patches belong to the same class. This is usually limited to models in which a fixed number of patch classes, called *parts*, are combined in some framework. A well known example of this kind is the family of constellation models [19, 46].

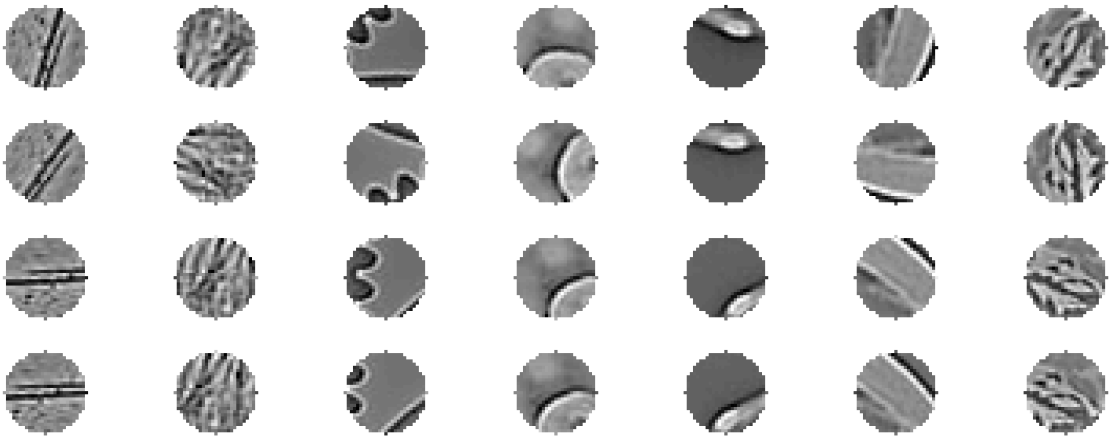

Figure 6-1: Examples of similar patches from whitened images. Each column contains rotated and shifted versions of the same original patch, so that by definition patches in each column are similar.

6.1.2 Interest operators

An important aspect of any patch-based technique is the method for selecting the patches that are subsequently evaluated for similarity–both in the test image and in labeled training images. This issue is often coupled with that of descriptors and similarity, since the invariance in the descriptor is induced by providing it with an estimate of the transformation of the patch with respect to some canonical reference frame, for instance, the angle necessary to align the main axis of a region with the vertical axis, or the size of a region.

There is a large number of interest operators, and extensive evaluation has been undertaken in a number of cases [102, 77, 71, 84]. On the other hand, the role of these operators (and the role of the keypoint concept, in general) in object recognition is still somewhat controversial: recent results in [9, 14, 80] show that excellent performance can be achieved without using keypoints or interest operators at all. In general, we would like to remain agnostic on this issue, and focus on the analysis of descriptors and their role in similarity matching. When necessary (namely, with SIFT descriptors) we will assume that an appropriate detector has been applied, and therefore will make available the basic information that would normally be provided by such a detector (location and scale of the patch).

6.2 Defining and labeling visual similarity

For the purposes of the study in this chapter, we consider two patches to be similar if they could be obtained by taking an image and then rotating (and/or shifting by a small amount) the imaging device and taking an image again–and extracting the patches from the same image location. Equivalently, the two patches are similar if there exists a transformation, consisting of a shift by between zero to two pixels, followed by an arbitrary in-plane rotation, that makes the two patches be identical,

up to pixel-level aliasing and possible boundary effects due to shift.[2]

In terms of image representation, this definition of similarity resembles the low-level invariance believed to exist in the lateral geniculate nucleus (LGN) and higher areas in the primate visual cortex [39, 64]. It also matches an intuitive notion of visual similarity that is related to semantics of the visual world: two patches are similar if they correspond to the same element of a physical scene. This leads to the idea underlying the *slow-feature analysis* technique for unsupervised learning of spatio-temporal coding, and in particular invariance to transformations, in the visual cortex [116]. In SFA, a stimulus to the learning algorithm is constructed by simulating a "natural movie"–a sequence of inputs obtained by moving a receptive field slowly (i.e., by applying only mild transformations to obtain the next frame) in the image. The objective of the algorithm is, in effect, to learn features that are efficient in encoding such sequences.

We take this idea and apply it to the task of labeling similar image patches. For a given image patch in a natural image, any number of patches similar to it may be transforming the receptive field according to the desired similarity. So, if we want to ignore the rotation, images obtained by rotating the receptive field at a fixed location can serve as examples of similar pairs. If shifts by up to a certain number of pixels are to be ignored, any two shifts within those bounds will produce image similar to each other, etc.

Figure 6-1 shows a few examples of sets of similar patches generated by the procedure outlined above. Patches in each column are versions of one patch (top row), rotated by a random angle or shifted in random direction by 2 pixels.

This notion of similarity adheres to the definition of equivalence given in Chapter 1. Also, as mentioned in Chapter 2, this notion of equivalence does not translate to a transitive equivalence relation in the patch space: taking a shifted patch and shifting it again will, again, create a similar pair, but the third patch may no longer be similar to the first one.

6.3 Patch descriptors

We will be focusing on two descriptors that have very different properties and were designed under very different objectives. The first one, the coefficients of a sparse overcomplete code, corresponds to a generative model of the patch, and is by design not invariant to transformations. The second, SIFT, is a constructed with a discriminative task in mind, specifically to be invariant under shift and rotation (when used in conjunction with an interest operator).

[2]Note that we work with patches shape like a disk (up to the pixelation aliasing artifacts) rather than a more common rectangle. This is to diminish the artifacts introduced in the corners by rotating a rectangular patch.

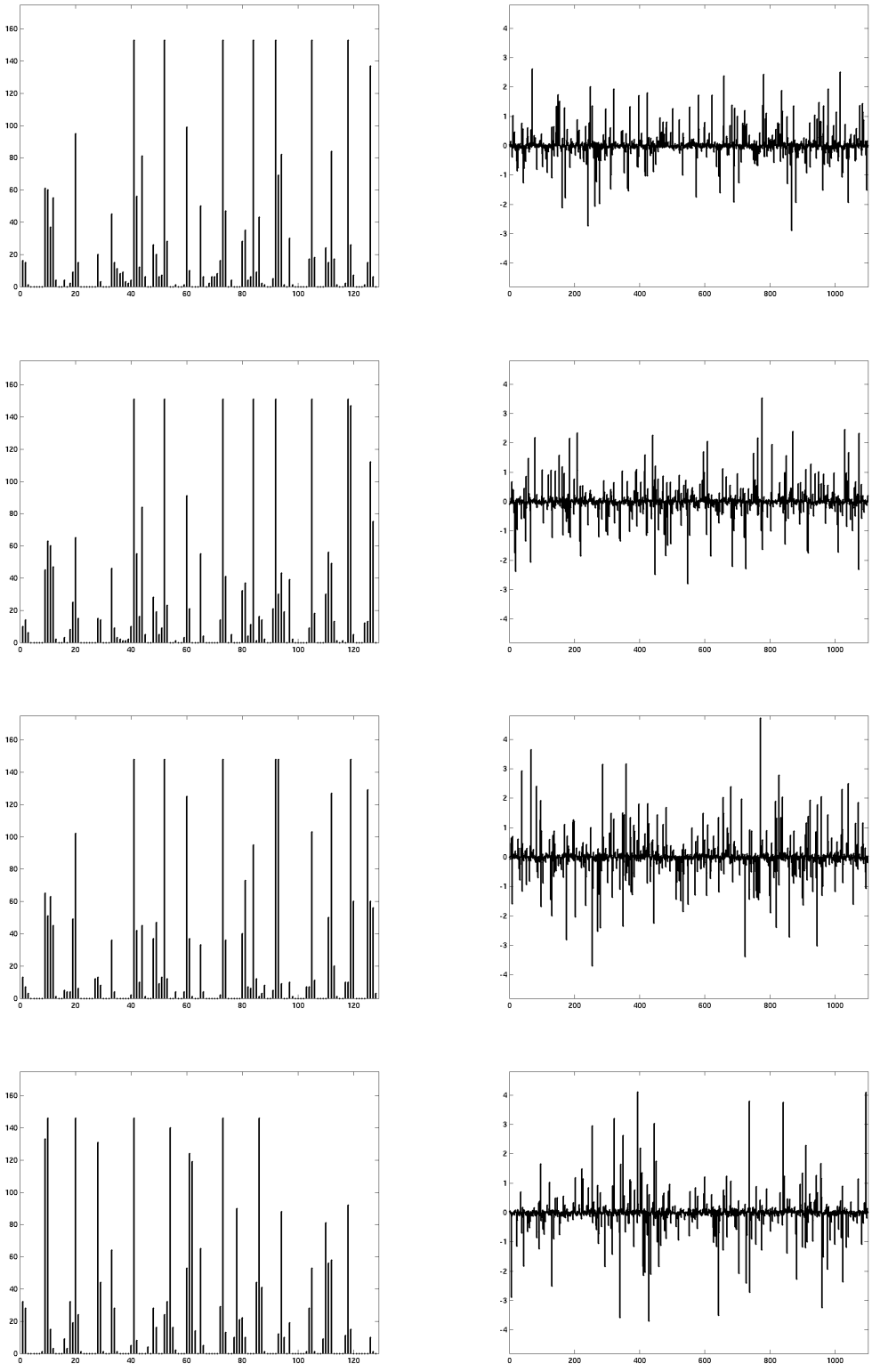

Figure 6-2: Descriptors for the patches in the third column from left in Figure 6-1. Left: SIFT descriptors. Right: coefficients of a sparse overcomplete code.

6.3.1 Sparse overcomplete code

A large body of work has been focused on construction of codes for natural images that would posses the key properties of the V1 cell populations, namely orientation selective and localized receptive fields. It has been shown in a number of studies [89, 64] that these properties emerge as a result of a learning procedure whose objective is to maximize the fidelity of reconstruction along with maximum sparseness of response. The latter is believed to be an important principle of sensory coding in the brain, related to the statistical properties of natural visual scenes[47].

Specifically, a sparse overcomplete code defines a generative model of the patch as a linear combination of C *basis functions*, that are themselves patches of the same dimension:

$$\mathbf{x} \;=\; \sum_{c=1}^{C} a_c \phi_c \;= \mathbf{\Phi a}. \tag{6.1}$$

The general objective of constructing this code is, naturally, to reduce the reconstruction error, which is measured by the energy in the residual. Under that objective alone, if we fix the basis functions, the optimal decomposition of a patch $\mathbf{x}$, in terms of the coefficients $\mathbf{a} = [a_1, \ldots, a_C]^T$ could be found as

$$\mathbf{a}^* \;=\; \operatorname*{argmin}_{\mathbf{a}} \|\mathbf{x} - \mathbf{\Phi a}\|. \tag{6.2}$$

However, two key properties of the codes we are discussing here are *overcompleteness* and *sparseness*. The former means that the number C of basis function is higher than the dimension of $\mathbf{x}$. The latter means that a majority of coefficient have very low absolute value for any given patch. As a consequence of overcompleteness, (6.2) will generally not have a unique solution. This is where the sparseness property becomes important: the optimization criterion is augmented by a penalty term, that drives the coefficients to zero. This can be written [89] as

$$\mathbf{a}^* \;=\; \operatorname*{argmin}_{\mathbf{a}} \|\mathbf{x} - \mathbf{\Phi a}\| \;+\; \lambda \sum_{c=1}^{C} S(a_c), \tag{6.3}$$

where $S(a_c)$ is an appropriate cost function that penalizes values away from zero. The specific form of this function is a matter of design; it should be noted that a choice of S corresponds to imposing a statistical model on the distribution of the coefficients (prior). For instance, the cost function

$$S(a) = \log(1 + a^2)$$

used in our experiments (following [89]) can be shown to correspond to the Cauchy prior.

It is important to distinguish this coding scheme from coding with non-overcomplete representations. The main difference is that there is no closed-form solution to the optimization, and it has to approached with an iterative optimization algorithm per-

forming gradient descent on the cost function. The algorithm, suggested in [89], starts with a random basis; a number of random patches are extracted from the training images, and the optimal coefficients with respect to the current basis are computed by means of gradient descent on (6.3). Given these coefficients, the basis functions are updated, with the objective to decrease the residual. The update rule is

$$\Delta\phi_c \;=\; \eta a_c \left(\mathbf{x} - \mathbf{\Phi a}\right),$$

where η is the learning rate. The process is repeated iteratively, until no further significant changes in $\mathbf{\Phi}$ are recorded.

Such codes have been successful in low-level vision tasks, such as image denoising [98]. Some properties of this representations make it potentially appealing for recognition purposes: Sparseness makes the code likely to provide good discriminative power [8, 92, 3], and high reconstruction fidelity means that they encoding will likely retain relevant information from the original image patch. A key question though is how to compare two patches encoded in this fashion.

The right column in Figure 6-2 shows the coefficient vectors for four similar patches (that appear in the third column from the left in Figure 6-1.) One immediate observation from the figure is that there is a significant variation in the codes; this suggests that the naïve use of L_1 between the coefficient vectors $\mathbf{a}$ may not be a good choice of similarity measure between patches.

The instability of the sparse overcomplete code is in fact a direct consequence of its key properties. Consider a particular patch, for which optimizing the coefficients in (6.2) makes a coefficient a_c to have a high absolute value. If the same patch is shifted by one pixel, the basis function ϕ_c will probably still account for some of the patch appearance reasonably well. However, there likely (due to the overcompleteness of the code) will be another basis function ϕ_j which will account for the shifted patch even better. Furthermore, the sparseness constraint will encourage the absolute values of both a_c and a_j to decrease. As a result, it will be more optimal from the perspective of reducing the cost function value to "keep" ϕ_j and suppress ϕ_c for the shifted patch. This explains the typical pattern in which even a small variation in the patch may cause significant changes in the code, and the resulting effect on similarity between transformed patches.

6.3.2 SIFT

The SIFT descriptor [77] is based on a histogram of oriented gradients within the region it describes. It is computed for a known location, scale and orientation in the image (provided by an interest operator). The descriptor is computed in the following way (the parameters given here are the ones used in our experiments, and can be varied when implementing the algorithm):

- The calculations are based on the appropriate level in the Difference-of-Gaussian pyramid, rather than on the original image. This induces scale-invariance.

- The orientation and magnitude of the intensity gradient are computed on a

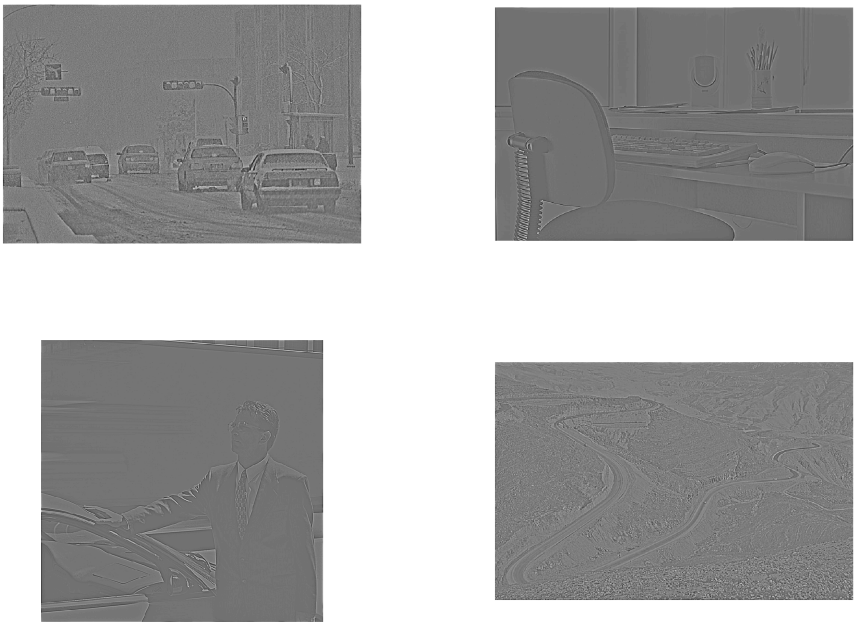

Figure 6-3: Examples of whitened images (four out of 100) used to train and test patch similarity measures.

fixed 16×16 grid in the vicinity of the target location. The values are weighed by a high-bandwidth Gaussian window centered at the location.

- A smoothed histogram of gradient orientations is computed for each subregion of 4×4 grid locations. With 8 bins in the histogram, this yields a 128-dimensional descriptor.

Note that the invariance to scale, rotation and translation with this descriptor are subject to an accurate estimation by an interest operator. The main source of the robustness of a SIFT descriptor is in their reliance on histograms and on gradients. As a result, it is not sensitive to absolute changes in intensity and to minor shifts. A detailed analysis of its stability is given in [77]; the experiments presented below can be seen as an additional verification of these properties of SIFT.

6.4 Experiments

In our experiments we used a collection of 100 natural images taken from the Hemera database [59]. These images contain natural and man-made scenes, indoors and outdoors.

6.4.1 Collecting descriptors

Sparse overcomplete code coefficients

We used a code learned on a set of 250 natural images, not used in the descriptor evaluation. The images were whitened as described in [89], to flatten the power spectrum; this whitening appears to be critical to ensure proper convergence of the code learning algorithm. Figure 6-3 shows a few examples of the training images. The learned code consists of 1,100 basis function, for a disk-like patch of diameter 27 pixels. The total number of pixels in a patch is 529, so that the code is more than twice overcomplete. Figure 6-4 shows a representative sample of the basis function in the code. Most of the emerging basis functions correspond to localized oriented filters,[3] a typical result consistent with numerous reports in the literature.

The set of patches used in the experiments was generated by the following procedure. For each image, we selected 100 locations, subject to a minimal variance criterion: the intensity variance within a patch centered at a selected location should be at least equal to the variance within the image. For each location (r, c) we:

- Draw four random angles between 0 and 360^o, and for each angle extract a patch centered at (r, c) and rotate it by each of the angles; this produces four patches.

- Extract (with no rotation) patches centered at $(c-2, r-2)$, $(c-2, r+2)$, $(c+2, r-2)$ and $(c+2, r+2)$. This produces another four patches.

This results in a total of 80,000 patches: 100 images$\times$100 locations$\times$8 similar patches associated with each location. For each patch we calculate the coefficients of the sparse code by applying the optimization in (6.3). The right column in Figure 6-2 shows four examples of the resulting 1,100-dimensional descriptors for a set of similar patches in Figure 6-1.

SIFT descriptors

To collect SIFT descriptors for the same patches represented by the sparse codes, we use the original (unwhitened) images. For each of the 80,000 collected patches, we define a keypoint at the location of that patch and with the scale corresponding to the diameter of the patch. We do not, however, "disclose" the rotation, and the descriptor is computed assuming the rotation is zero degrees.

The 128-dimensional descriptors were computed using the code from the Visual Geometry Group at Oxford.[4] The left column in Figure 6-2 shows the SIFT descriptors for the same patches as the codes on the right.

[3] Note that in general, since there is no closed-form solution for $\mathbf{a}$, it is not possible to infer the filter corresponding to a basis function ϕ directly from ϕ, but rather one needs to estimate it on a sample of natural stimuli.

[4] `http://www.robots.ox.ac.uk/~vgg/software/`

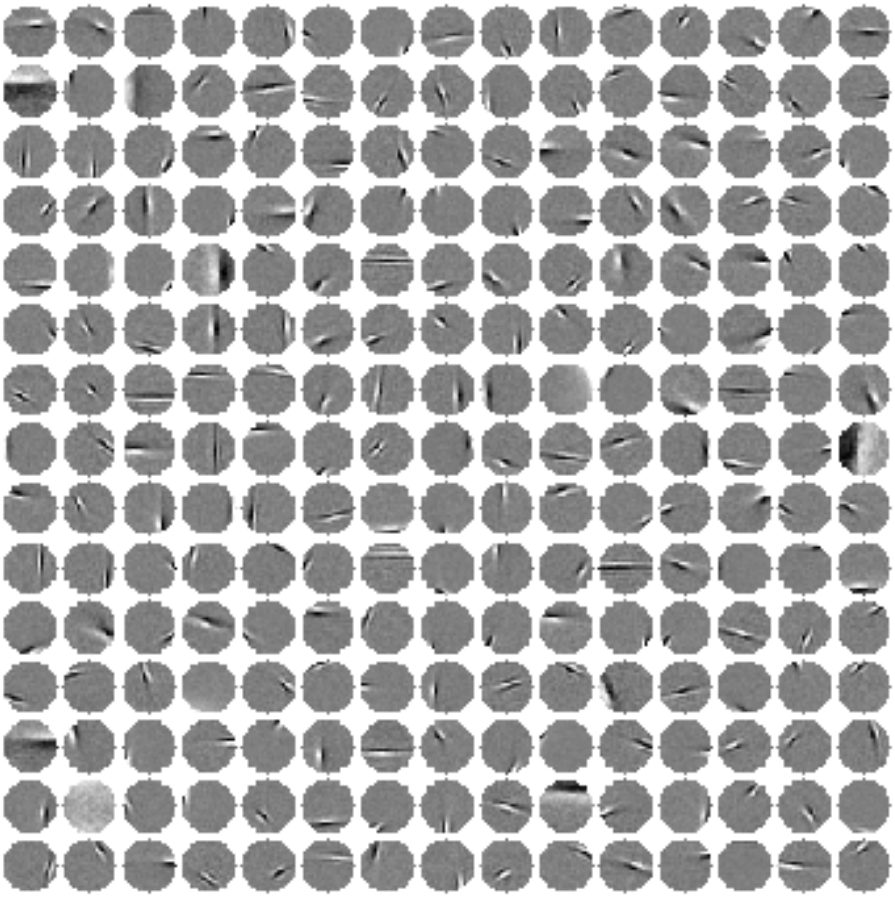

Figure 6-4: Example basis function for the sparse overcomplete code used in the experiments. 225 out of 1100 basis functions are shown. Patch size is 529 pixels (an approximate disk with diameter 27).

6.4.2 Embedding the descriptors for similarity

Under the similarity notion defined above, we can construct 280,000 similar and more than 3×10^9 dissimilar pairs out of the 80,000 patches. We randomly selected 6,000 similar and 10,000 dissimilar pairs for training, and five times as many distinct pairs for testing. All the results shown in this section were computed on the testing pairs.[5]

The embeddings for both SIFT and sparse codes were learned by running BoostPro on the training pairs, using linear ten-term projections (linear combinations of 10 dimensions.)[6] Since the space of the combinations is very large, we initialized the

[5]The baseline similarity measures which do not involve any learning have, of course, identical performance on the training and testing data.

[6]We experimented with smaller numbers of terms, however the results, measured on an independent validation set, were significantly worse. We believe this is due to the fact that with a random pair of dimensions, if they are "useless", not much can be done by the gradient ascent, whereas with ten dimensions, there is higher likelihood that at least some are useful, and the projection coefficients are updated accordingly.

Similarity	Pixels	L_1 on $\mathbf{a}$	L_1 on $H(\mathbf{a})$	SIFT	H(SIFT)
AUC	0.6049	0.5651	0.6847	0.8794	0.9633

Table 6.1: Area under ROC for similarity measures compared in our evaluation.

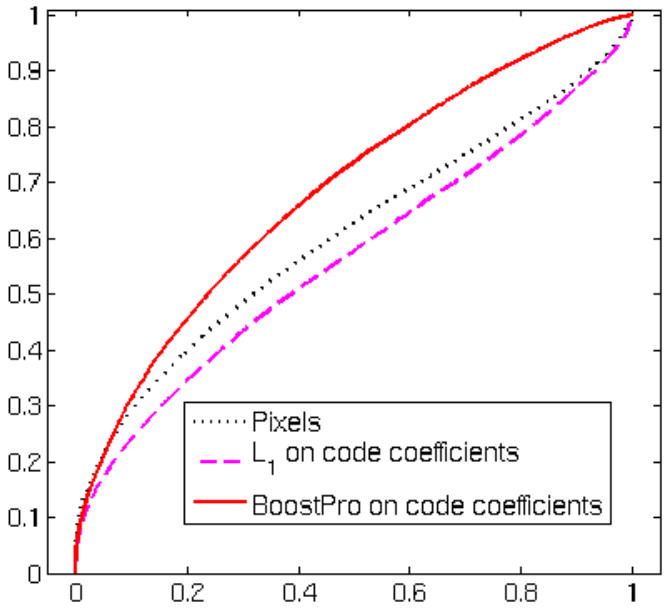

Figure 6-5: ROC curves with sparse overcomplete codes. Dotted: L_1 distance on pixels. Dashed: L_1 distance on code coefficients. Solid: L_1 in BoostPro embedding.

gradient descent from 100 random projections in every iteration of boosting, and selected the best among the finishing points of the 100 gradient ascent chains. In both cases, the boosting was run for 100 iterations, thus producing one hundred embedding dimensions–a dimensionality reduction for both descriptors, particularly significant for the sparse codes! However, one should keep in mind that this is not a new representation of reduced dimension that retains all useful properties of the original one; the purpose of the embedding is explicitly to facilitate similarity detection, and, for instance, the generative power of the sparse code is lost in the 100-dimensional embedding.

We are interested in the accuracy of matching similar patches. Since the match is decided by thresholding a numerical value , namely, the distance either in the original descriptor space or in the embedding space, we can use the ROC curves to compare the models of similarity. These are shown in Figures 6-5 and 6-6, and the areas under the curves are given in Table 6.1.

A comparison between the baseline ROC curves (dashed lines in Figures 6-5 and 6-6, and the area under the curves given in Table 6.1, confirm the intuition that the

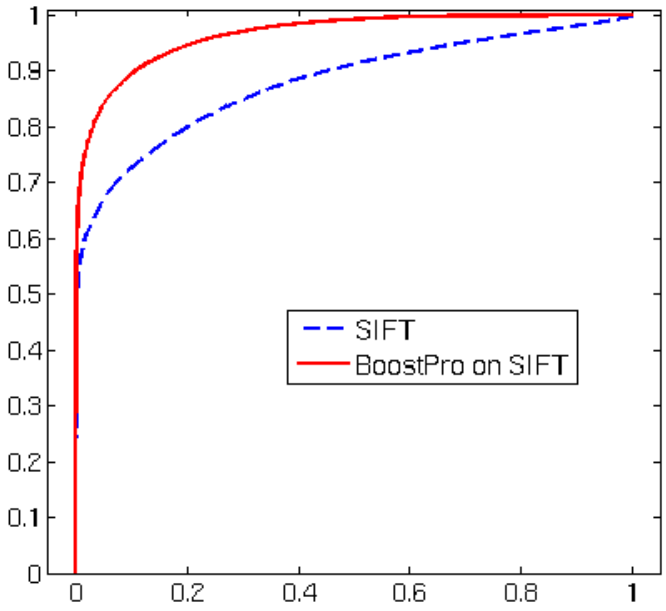

Figure 6-6: ROC curves with SIFT descriptors. Dashed: L_1 on SIFT descriptors. Solid: L_1 in BoostPro embedding.

sparse code coefficients in their "raw" form are not effective for matching patches. The performance of both pixel-based match and the L_1 distance on the sparse code coefficients is essentially not much better than random.

The ROC of the distances in the similarity embedding, on the other hand, are clearly superior to those of the distances in the original descriptor space, both for SIFT and for the sparse codes. For SIFT, the embedding yields an improvement in the area under curve to more than 0.96, with the equal error rate of 0.8930 (i.e., the probability of detecting a correct match of 89.3% corresponds to probability 10.7% of false match). By comparing the patch paitrs misclassified by the two measures we can see that most of the gain is achieved in learning the rotation correspondences between the gradient histogram bins, and as a result the error rate on patches similar up to rotation is decreased with the embedding similarity. As for the shifts, the descriptor is already quite robust to matching shifted patches. These findings are in agreement with the analysis of SIFT performance in [77].

We also have analyzed the embedding of the sparse code coefficients. Figure 6-7 shows, in each row, the basis patches which form the projection, with the coefficients written under the corresponding patches. The three rows correspond to the first three projections. Note that due to the non-convex nature of the optimization surface in BoostPro, these are not necessarily, or even likely, the most efficient projections, but rather simply the projections that happened to be picked up first. This is also reflected in the magnitude of the α_m values (the height of spikes in Figure 6-8). Often in application of AdaBoost these values decrease steadily, however this is not the case

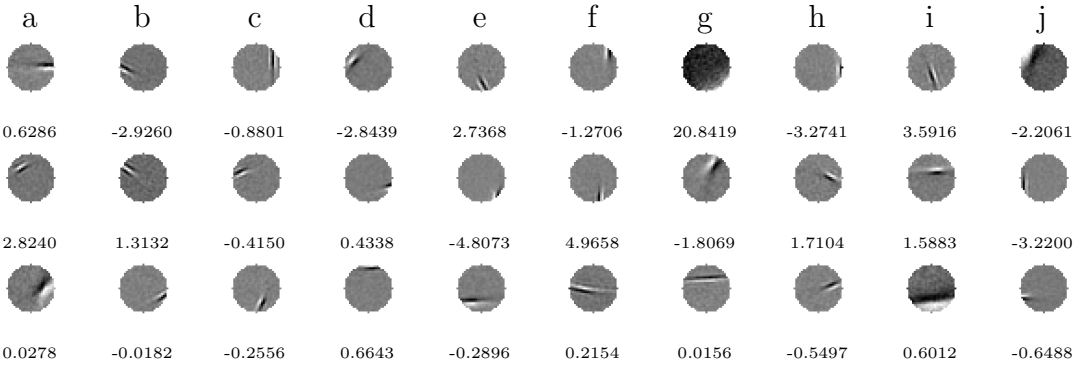

Figure 6-7: The first three projections in the embedding of sparse code coefficients learned with BoostPro. The number under each basis patch is the coefficient in the projection. Thresholds, selected for each projection by applying Algorithm 4, are 6.85, .16 and 0.003, respectively. The letters are used to identify the columns.

here.

We will discuss in some detail the top row of the figure, showing the first projection; letters on top help identify the basis functions. Recall from Chapter 3 the general interpretation of the projections: two patches must fall on the same side of the threshold in order for the projection to contribute to the matching score (or, equivalently, for the projection *not* to contribute to the distance value in H.) Perhaps the best way to describe the effect of the individual components of the projection is to say that if for the two patches the coefficients for the basis function d have high value, and the coefficients for the function e have low value (i.e., large absolute value and negative sign), it increases their chances to be considered similar by the weak classifier associated with this projection. Note that the low frequency basis function g has a coefficient much higher than others, so that this particular projection largely looks at the spatial frequency of the features within the patch, so, for instance, two relatively "flat" patches will be likely considered similar by this projection.

Generally, the magnitude of the coefficients in the projection determines the importance of the corresponding basis function (within that projection). The signs are only meaningful relative to the signs of the other projection components. For instance, the sign of the function d is not important by itself: its contribution to similarity judgment on two patches that both have $a_d = 3$ will be the same as for two patches with $a_d = -3$ (in both cases it will "move" both patches in the same direction relative to the threshold.) However, in combination with other basis functions and their signs in the projection in the same projection the sign does play a role: if two basis functions are likely to correspond to similar patches their sign will likely match. For instance, a patch with $a_f = 1$ and a patch with $a_h = 1$ will have a positive contribution from the corresponding projection coordinates (i.e. they will be "moved" towards the same side of the threshold); this probably is explained by the possibility of a shift that move a vertical edge on the right side of the patch in the right-left direction.

Finally, Figure 6-8 shows the embeddings themselves, for the same example patches as the descriptors in Figure 6-2. This figure provides a visual illustration of the improved correspondences between the representations of similar patches (much more noticeable with SIFT.)

6.5 Discussion

The main conclusion from the experiments presented here is that it is beneficial to learn a similarity model, rather than to rely on properties of a descriptor and use the metric as a proxy for such similarity. This is the case both when the descriptor in question is not designed to be invariant to transformations, as is the case with the sparse overcomplete codes, and when it is specifically designed with such invariance in mind, which is the case with SIFT. For both descriptors, it pays off to learn a similarity model, as far as the matching quality measured by ROC is concerned. Our experiments also suggest that for matching similar image patches under rotations and minor shifts, the best method among the ones investigated is to use the SIFT descriptor of the region, embedded into a similarity-reflecting space by applying BoostPro.

Another important conclusion is that the sparse overcomplete codes for image patches can be potentially used in a matching-based framework, if a proper similarity measure (or equivalently a proper embedding of the code) is used, rather than the naïvely applied L_1 distance. While the improvement is still not enough to place this descriptor in the same "league" with SIFT, we believe that certain characteristics make it appealing and warrant further looking into its use in recognition. First, it is based on a generative model of natural images, and thus can *exlain* the patch in terms of the fundamental visual elements comprising it. Second, the performance achieved here did not require any information from an interest operator, of the kind necessary to apply SIFT. This may be an appealing property for approaches that do not use the keypoint paradigm.

An aspect of our approach to modeling visual similarity of patches that distinguishes it from other approaches, is that we do not impose an invariant representation directly. Instead we learn a representation that makes similarity under the transformations explicit, thus causing the *matching* to become invariant. Another key aspect is that similarity is *learned*, not hand-crafted into the model. This is promising from the perspective of class-specific or object-specific matching, when similarity is defined in different ways for different classes, or when the type of patches to be compared is restricted. Of course, this promise has to be evaluated in further experiments, which are in the focus of our ongoing work.

The implications of this study on "downstream" applications to recognition and classification also remain to be seen. Our goal here has been to separate the patch similarity measure from the context in which is is used by an overall recognition strategy, under the assumption that any strategy would benefit from a better matching component. Further experiments are needed to quantify the effect of the improvement in matching on the accuracy in the final task of a system. Experiments in

this direction should extend the notion of similarity (or, equivalently, invariance) to affine transformations not included in our experiments above, namely, out-of-plane rotations. However, we expect that the effect of learning the embedding will be maintained in this extended framework. Another issue that may require special care is the selection of informative patches. An almost "flat" patch may be, under most reasonable definitions of visual similarity, similar to any other flat patch however establishing such a match may not be informative from a classification or recognition perspective. More generally, some patches are more useful than others for a particular object class. This leads to the idea of class-specific visual similarity between patches. We have not yet developed an approach that would achieve this but some ideas are discussed briefly in the next chapter.

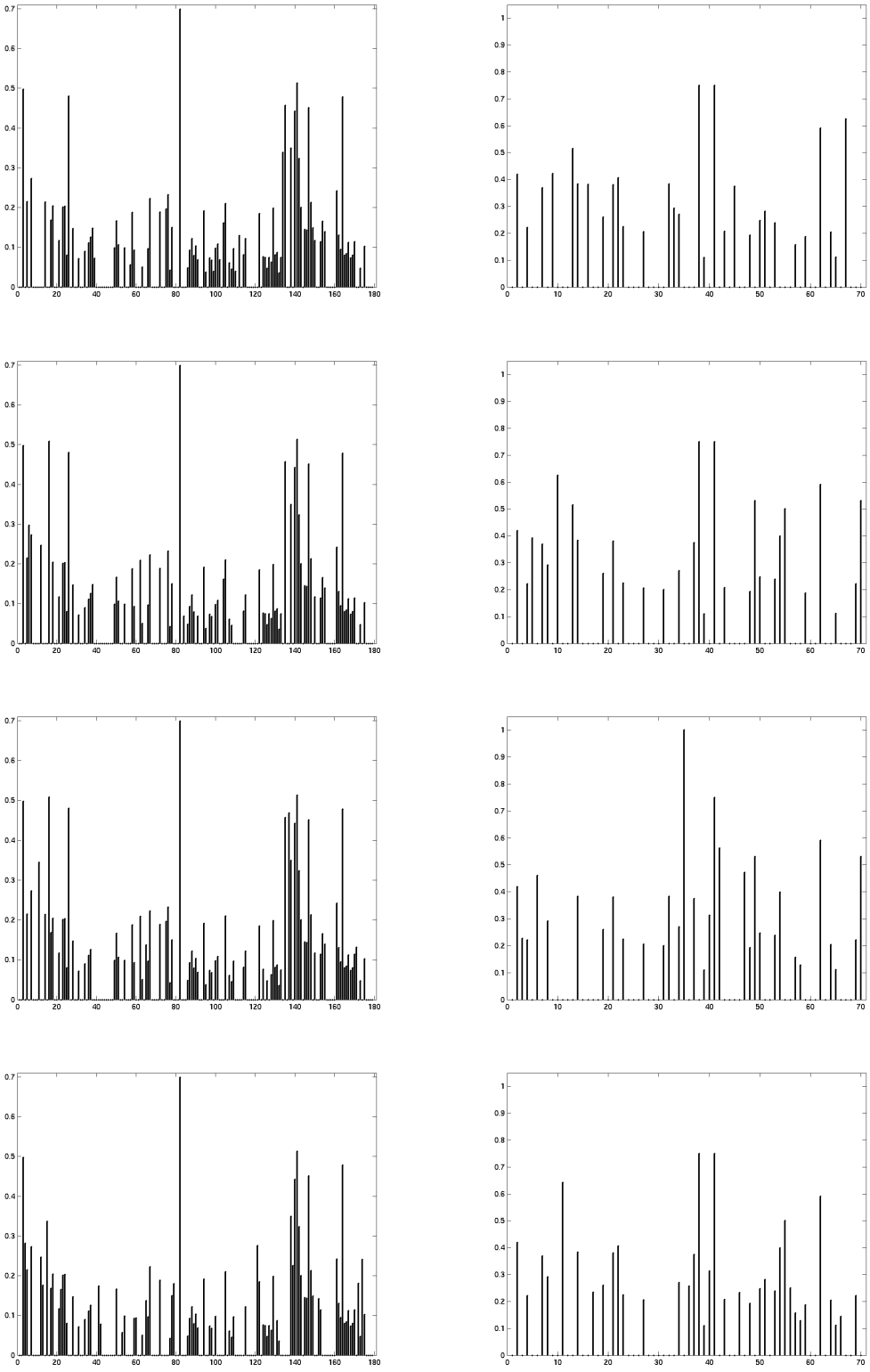

Figure 6-8: Embeddings of the codes shown in Figure 6-2. Left: BoostPro on SIFT. Right: BoostPro on sparse code coefficients.