Sparse Quantization for Patch Description

Xavier Boix Michael Gygli Gemma Roig Luc Van Gool
Computer Vision Lab, ETH Zurich, Switzerland
{boxavier,gygli,gemmar,vangool}@vision.ee.ethz.ch

Abstract

The representation of local image patches is crucial for the good performance and efficiency of many vision tasks. Patch descriptors have been designed to generalize towards diverse variations, depending on the application, as well as the desired compromise between accuracy and efficiency. We present a novel formulation of patch description, that serves such issues well. Sparse quantization lies at its heart. This allows for efficient encodings, leading to powerful, novel binary descriptors, yet also to the generalization of existing descriptors like SIFT or BRIEF. We demonstrate the capabilities of our formulation for both keypoint matching and image classification. Our binary descriptors achieve state-of-the-art results for two keypoint matching benchmarks, namely those by Brown [6] and Mikolajczyk [18]. For image classification, we propose new descriptors that perform similar to SIFT on Caltech101 [10] and PASCAL VOC07 [9].

1. Introduction

The representation of local image patches has received a lot of attention. It is crucial for many vision approaches. The patch descriptors should typically come with a certain degree of invariance to probable image and appearance variations, while being efficient to compute. Multi-view keypoint matching would require invariance under viewpoint and lighting changes. Object recognition approaches would typically add the need for robustness under inter and intra-class variations.

There is a surfeit of such patch descriptors by now. Authors can choose the most appropriate descriptor for their task, striking a balance between accuracy and efficiency. The SIFT descriptor [17] is a very popular example. For keypoint matching, its discriminative power has been surpassed, *e.g*. by learning its pooling regions [6, 18, 21], and so has its efficiency, *e.g*. with SURF [3], CARD [2], BRIEF [7] and others [1, 16, 20, 22, 23]. For object recognition, methods usually exploit hierarchical architectures of descriptors. SIFT may come as an integrated part thereof, but patch descriptors can also take the form of sparse coding [27], or convolutional networks [5, 13, 15].

Little is understood about the common principles underlying the different patch descriptors. Often descriptors appear to be disconnected from the prior art. For instance, what could we say *a priori* about the relative performance of descriptors, even before testing them? The lack of clearcut answers to such questions has led to a plethora of descriptors, designed for specific applications.

In this paper, we introduce a new, more principled formulation to patch description. To emphasize its generality, we show that it can instantiate diverse descriptors, e.g. SIFT and BRIEF. We also take advantage of the capabilities of our formulation to design novel, more discriminative and computationally efficient (binary) descriptors. Our formulation is based on sparse quantization (SQ) [4], which is the quantization into a set of k-sparse vectors. SQ supports efficiency because it can be computed with a simple sorting and it can yield binary descriptors.

In a series of experiments, we report results on both, keypoint matching and image classification tasks. In keypoint matching we achieve state-of-the-art results with a binary descriptor on the Brown [6] and Mikolajczyk [18] datasets. For image categorization, we report results on Caltech101 [10] and PASCAL VOC07 [9], where our method achieves better performance than SIFT.

2. Sparse Quantization

We first introduce Sparse Quantization (SQ), as we use this mathematical tool in the rest of the paper. It will serve as a basis for our new patch description formulation, introduced in the next section.

Let $\mathbb{R}^q_k$ be the set of k-sparse vectors, *i.e*. $\{\mathbf{s} \in \mathbb{R}^q : \|\mathbf{s}\|_0 \leq k\}$. Also, we define $\mathbb{B}^q_k = \{0,1\}^q_k = \{\mathbf{s} \in \{0,1\}^q : \|\mathbf{s}\|_0 = k\}$, which is the set of binary vectors with k elements set to one and $(q-k)$ set to zero. The cardinality of $|\mathbb{B}^q_k|$ is equal to $\binom{q}{k}$. We extend the binary set to incorporate negative values by defining the set of k-sparse vectors built from $\mathbb{T} = \{-1,0,1\}$. It is $\mathbb{T}^q_k = \{-1,0,1\}^q_k = \{\mathbf{s} \in \{-1,0,1\}^q : \|\mathbf{s}\|_0 = k\}$, and it has cardinality $|\mathbb{T}^q_k| = 2^k\binom{q}{k}$.

The quantization of a vector $\mathbf{v} \in \mathbb{R}^q$ into a codebook $\{\mathbf{c}_i\}$ is a mapping of $\mathbf{v}$ to the closest element in $\{\mathbf{c}_i\}$, *i.e*. $\hat{\mathbf{v}}^\star = \arg\min_{\hat{\mathbf{v}} \in \{\mathbf{c}_i\}} \|\hat{\mathbf{v}} - \mathbf{v}\|^2$. In the case of SQ, the code-

book $\{\mathbf{c}_i\}$ contains the set of k-sparse vectors. These may be any of the previously introduced types: $\mathbb{R}^q_k$, $\mathbb{B}^q_k$, or $\mathbb{T}^q_k$. Such quantization is instrumental to our framework.

An important advantage of SQ over a general quantization is that it can be computed much more efficiently. The naive way to compute a general quantization is to evaluate the nearest neighbor of $\mathbf{v}$ in $\{\mathbf{c}_i\}$, which may be costly to compute for large codebooks and high-dimensional $\mathbf{v}$. In contrast, SQ can be computed by selecting the k higher values of the set $\{v_i\}$, according to the following Proposition. The latter is an extension of the result in [4] to the $\mathbb{T}^q_k$ set. This extension allows for negative-valued inputs, which we will need later for our new formulation. We assume that the input is normalized, $\|\mathbf{v}\|_2 \leq \|\mathbf{s}\|_2/\sqrt{k}$, where $\mathbf{s} \in \mathbb{T}^q_k$ (all vectors in $\mathbb{T}^q_k$ have the same norm). In all cases where we use the Proposition this is true. The proof itself is in the Supplementary Material.

Proposition 1. *Let $\hat{\mathbf{v}}^\star$ be the quantization into $\mathbb{T}^q_k$ of $\mathbf{v} \in \mathbb{R}^q$, which is $\hat{\mathbf{v}}^\star = \arg\min_{\hat{\mathbf{v}} \in \mathbb{T}^q_k} \|\hat{\mathbf{v}} - \mathbf{v}\|^2$. For $\|\mathbf{v}\|_2 \leq \|\mathbf{s}\|_2/\sqrt{k}$, where $\mathbf{s} \in \mathbb{T}^q_k$, $\hat{\mathbf{v}}^\star$ can be computed by*

$$\hat{v}^\star_i = \begin{cases} \textit{sign}(v_i) & \textit{if } i \in k\textit{-Highest}(|\mathbf{v}|) \\ 0 & \textit{otherwise} \end{cases} . \qquad (1)$$

$|\mathbf{v}|$ is the element-wise absolute value of $\mathbf{v}$, and k-Highest($|\mathbf{v}|$) is the set of dimension indices that indicate which are the k elements of the vector $|\mathbf{v}|$ with the highest values.

Prop. 1 shows that the SQ into $\mathbb{T}^q_k$ can be done with a sorting of the absolute value of $\{v_i\}$, and preserving the sign of v_i in the output. As with SQ into $\mathbb{R}^q_k$, it has much lower computational cost than the nearest neighbor approach.

3. SQ for Encoding in Patch Description

In this Section we introduce a new formulation for feature encoding, which is based on the principles of SQ. Additionally, we show that SIFT and BRIEF descriptors are instances of our formulation.

3.1. General Overview of Patch Description

We first review the pipeline for patch description and introduce some general terminology. This will serve to identify the role of our formulation, which is specific for feature encoding.

Our patch description framework can be decomposed into the following steps: 1) extracting local features, 2) encoding them, and 3) a spatial pooling stage. We extract features at the pixel level, computing filter responses at multiple locations in the patch. We use $\mathbf{f}_j \in \mathbb{R}^q$ to denote a feature vector. It results from evaluating q filters at a particular location. Since we extract features at multiple locations, we obtain the set of features vectors $\{\mathbf{f}_j\}$. Feature encoding maps each feature vector into a more suitable space to achieve better discriminative properties. We denote $\{\boldsymbol{\alpha}_j\}$ as the set of encoded feature vectors. Finally, the pooling summarizes $\{\boldsymbol{\alpha}_j\}$ into a single vector which is the final descriptor of the patch. Our feature encoding framework mainly focuses on the mapping of $\mathbf{f}_j$ to $\boldsymbol{\alpha}_j$. This mapping is introduced in the following subsections.

3.2. Encoding with SQ

We apply exactly the same encoding for all $\mathbf{f}_j$, independently of the location of the patch from which they were extracted. For the sake of notational simplicity, we drop the subindex j that distinguishes among different locations.

Our formulation is based on assignment-based encoding, inspired by the mid-level features of object recognition algorithms. Assignment-based encodings such as Hard and Soft Assignment select the k entries of a codebook that are closest to $\mathbf{f}$. Then, the output vector, $\boldsymbol{\alpha}$, is built taking ones (Hard Assignment) or similarity measures (Soft Assignment) as the elements of $\boldsymbol{\alpha}$ that correspond to the k selected entries of the codebook. Other elements of $\boldsymbol{\alpha}$ are set to 0. In our previous work [4], we have shown that Hard and Soft-Assignment encodings can be considered the SQ of a mapping of the input vector. We adopt our formulation, which we used for mid-level features, as a starting point for our patch descriptor encoding. We use our formulation, rather than the original formulation of Soft-Assignment, because it allows for the computational advantages of applying Prop. 1, and it nicely connects to previous works, such as Sparse Coding [8, 27].

The feature encoding is based on the mapping $\Psi(\mathbf{f}, \{\mathbf{b}_i\})$, in which $\{\mathbf{b}_i \in \mathbb{R}^q\}$ is a set composed of Q vectors, that live in the same space as the input features $\mathbf{f} \in \mathbb{R}^q$. Ψ maps $(\mathbf{f}, \{\mathbf{b}_i\})$ to a vector in $\mathbb{R}^Q$, which contains the similarity measures between the feature vector $\mathbf{f}$ and the vectors in the set $\{\mathbf{b}_i\}$. Thus,

$$\Psi(\mathbf{f}, \{\mathbf{b}_i\}) = \frac{1}{Z}(K(\mathbf{f}, \mathbf{b}_1) \dots K(\mathbf{f}, \mathbf{b}_Q)) \in \mathbb{R}^Q, \qquad (2)$$

where Z normalizes the vector and $K(\cdot, \cdot)$ may be any similarity measure. An example similarity measure used in the sequel, is the gaussian kernel: $K(\mathbf{f}, \mathbf{b}_i) = \exp(-\frac{\|\mathbf{f}-\mathbf{b}_i\|^2}{\sigma^2})$.

Finally, the encoding of $\mathbf{f}$ is defined as a SQ of Ψ:

$$\boldsymbol{\alpha}^\star = \arg \min_{\boldsymbol{\alpha} \in \mathbb{R}^Q_k} \|\boldsymbol{\alpha} - \Psi(\mathbf{f}, \{\mathbf{b}_i\})\|, \qquad (3)$$

where k is a parameter to be set. The equivalence to Soft-Assignment can be seen from the fact that Ψ computes the similarities to the set $\{\mathbf{b}_i\}$ (the so called codebook in the Soft-Assignment literature [24]), and then, invoking Prop. 1 for $\mathbb{R}^Q_k$, the SQ of Ψ selects the k-highest entries in Ψ, and indicates this selection in $\boldsymbol{\alpha}$. For further details on the equivalence between Eq. (3) and Soft-Assignment, we refer to [4].

We further develop this formulation by specifying the form of $\{\mathbf{b}_i\}$. This will allow us to propose novel, powerful and efficient encodings, but also to recover the well-known BRIEF and SIFT descriptors. Recall that $\mathbb{T}_p^q$ is the set of p-sparse vectors built from $\mathbb{T} = \{-1, 0, 1\}$ (Sec. 2). We define $\{\mathbf{b}_i\} = \bigcup_{0<p\leq q} \bar{\mathbb{T}}_p^q$, where $\bar{\mathbb{T}}_p^q$ are the vectors in $\mathbb{T}_p^q$ normalized with the ℓ_2 norm. Observe that $\{\mathbf{b}_i\}$ is defined as the union of $\bar{\mathbb{T}}_p^q$, for p ranging from 1 to q. This yields a cardinality of $|\bigcup_{0<p\leq q} \bar{\mathbb{T}}_p^q|$ equal to $(3^q - 1)$. From now on, we use Q to denote this value, *i.e.* $Q = (3^q - 1)$. We summarize in the following definition the final form of our proposed feature encoding.

Definition 1. *Let $\boldsymbol{\alpha}^\star \in \mathbb{R}_k^Q$ be the encoding of $\mathbf{f} \in \mathbb{R}^q$ such that*

$$\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}_k^Q} \|\boldsymbol{\alpha} - \Psi(\mathbf{f}, \bigcup_{0<p\leq q} \bar{\mathbb{T}}_p^q)\|_2. \quad (4)$$

At this point, the reader may object that fixing $\{\mathbf{b}_i\}$ like that is a big leap of faith and may actually harm performance. Yet, there is a considerable body of evidence in the literature showing that the impact of $\{\mathbf{b}_i\}$ is negligible compared to that of the architecture of the descriptor [8, 13]. Moreover, in our experiments we show this to also hold in our case and that, indeed, we can achieve high levels of performance with this choice.

In the next section, we show that we can compute the encoding in Def. 1 efficiently by exploiting SQ, and without fully calculating Ψ.

3.3. Implementation Advantages

We can see by analyzing Def. 1 that the input feature $\mathbf{f} \in \mathbb{R}^q$, is mapped to a much higher dimensional space through the mapping Ψ. Unfortunately, the direct computation of Ψ may introduce a computational overhead, since it consists on evaluating similarity measures between $\mathbf{f}$ and the elements in the union of sets $\bar{\mathbb{T}}_p^q$. For instance, for $q = 4$ filters, Ψ involves 80 ($Q = 3^4 - 1$) similarity measures. This, when done at many locations in the patch, yields a too slow patch descriptor for most applications. The complexity of computing the SQ in Def. 1 by calculating the full mapping Ψ is $O(qQ) = O(q3^q)$, because it involves 3^q different q-dimensional distances.

However, one of the reasons that we made the particular choice of $\{\mathbf{b}_i\}$ equal to the union of $\bar{\mathbb{T}}_p^q$, is because it allows an efficient optimization with cost $O(q^2)$. This is much lower than the optimization by exhaustively computing Ψ, $O(q^2) \ll O(q3^q)$. We introduce Alg. 1 that, under the conditions shown in the following Proposition, allows computing the exact encoding in Def. 1 very efficiently in practice. We leave the proof in the Supplementary Material.

Proposition 2. *Let $q \leq 4$ and $k \leq 2$. Then, Algorithm 1 obtains the global minimum for $\boldsymbol{\alpha}^\star$ in Def. 1 with computational complexity $O(q^2)$.*

Algorithm 1: Sparse Quantization in Proposition 2

Input: $\mathbf{f} \in \mathbb{R}^q$
Output: $\boldsymbol{\alpha}^\star \in \mathbb{R}_k^Q$
forall $0 < p \leq q$ **do**
  $\boldsymbol{\beta}_p^\star = \arg\min_{\boldsymbol{\beta}\in\bar{\mathbb{T}}_p^q} \|\boldsymbol{\beta} - \mathbf{f}\|_2$
end
$\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}_k^Q} \|\boldsymbol{\alpha} - \tilde{\Psi}(\mathbf{f}, \{\boldsymbol{\beta}_p^\star\})\|_2$

The intuitive idea behind Alg. 1 is that it decomposes the SQ of $\Psi(\mathbf{f}, \bigcup_{0<p\leq q} \bar{\mathbb{T}}_p^q)$ into smaller SQs, which only involve a single $\bar{\mathbb{T}}_p^q$ of the union. Each of these smaller SQs selects one element in $\bar{\mathbb{T}}_p^q$, which is used as a candidate for the main SQ in Def. 1. The non-selected candidates are discarded, thus reducing the complexity of the main SQ.

We now analyze the computational advantages by having a closer look to Alg. 1. Observe that the SQs that are computed in the loop, directly operate on $\mathbf{f}$ without passing through a mapping such as Ψ. Thus, they can be directly solved by invoking Prop. 1, which consists on a simple sorting of $\{f_i\}$. We can reuse this sorting for all SQ in the loop.

The final SQ after the loop only uses the candidates selected by previous SQs. It operates on the following mapping:

$$\tilde{\Psi}_i(\mathbf{f}, \{\boldsymbol{\beta}_p^\star\}) = \begin{cases} \Psi_i(\mathbf{f}, \boldsymbol{\beta}_p^\star) & \text{if } \boldsymbol{\beta}_p^\star = \mathbf{b}_i \\ 0 & \text{otherwise} \end{cases} \in \mathbb{R}^Q, \quad (5)$$

where $\{\boldsymbol{\beta}_p^\star\}$ is the output of the SQ in the loop, and $\mathbf{b}_i$ is an element of $\bigcup_{0<p\leq q} \bar{\mathbb{T}}_p^q$. In this way, $\tilde{\Psi}$ has the same values and structure as Ψ, except that $\tilde{\Psi}$ is equal to 0 in the dimensions not corresponding to the set of candidates $\{\boldsymbol{\beta}_p^\star\}$. Thus, the computation of $\tilde{\Psi}$ consists on calculating only q elements of Ψ, one for each candidate, rather than the $3^q - 1$ for the full Ψ. This yields a final complexity of q different q-dimensional distances, *i.e.* $O(q^2)$.

Prop. 2 restricts $q \leq 4$ and $k \leq 2$ to guarantee that Alg. 1 obtains the global minima of the SQ. In all descriptors that we propose in the sequel, these constraints are fulfilled. We evaluated descriptors when this is not the case, and in practice, we did not observe any difference in the performance when compared to SQ explicitly computing all Ψ. Thus, when the constraints are not fulfilled, we observe that Alg. 1 still obtains solutions close enough to the optimal that the performance is not deteriorated.

3.4. Relation with Other Encodings

We now show that we can instantiate SIFT and BRIEF encodings from our formulation. Additionally, we show the relation to Sparse Coding and Convolutional Networks. We provide further details in the Supplementary Material.

SIFT. It extracts $\mathbf{f} \in \mathbb{R}^2$ using two filters, the horizontal and vertical gradient. The encoding in SIFT selects the

two closest elements of $\mathbf{f}$ in a codebook that consists on 8 elements placed on the unit circle every $\frac{\pi}{4}$. $\mathbf{f}$ is previously ℓ_2-normalized to be also in the unit circle. The encoding uses the similarity $K(\mathbf{f}, \mathbf{b}_i) = 1 - \frac{4}{\pi} d(\mathbf{f}, \mathbf{b}_i)$, where $d(\cdot, \cdot)$ is the geodesic distance in the circle group (the difference of the angles). Finally, α_i is equal to the similarity K for the two selected codebook entries, otherwise is equal to 0.

We can see that this encoding is a particular instance of our formulation:

$$\boldsymbol{\alpha}^\star_{\text{SIFT}} = \arg \min_{\boldsymbol{\alpha} \in \mathbb{R}^8_2} \|\boldsymbol{\alpha} - \Psi(\mathbf{f}, \bigcup_{0<p\leq 2} \bar{\mathbb{T}}^2_p)\|_2, \qquad (6)$$

where Ψ uses the geodesic distance in the circle group. Observe that $\bigcup_{0<p\leq 2} \bar{\mathbb{T}}^2_p$ coincides with the 8 elements ($Q = 3^2 - 1 = 8$) into which SIFT quantizes $\mathbf{f}$: The elements in $\bar{\mathbb{T}}^2_1$ coincide with the orientations at $\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$, and in $\bar{\mathbb{T}}^2_2$ at $\{\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\}$. The SQ into $\mathbb{R}^8_2$ selects the two higher entires in Ψ. In the sequel, we show that we can achieve much higher performance with this encoding if we incorporate a higher number of extracted filter responses than 2 gradients.

BRIEF. It extracts a one dimensional $\mathbf{f} \in \mathbb{R}^1$, which is the difference between two pixel values, and then it encodes $\mathbf{f}$ to 1 or 0 depending on whether $\mathbf{f}$ was positive or not. BRIEF applies this encoding multiple times to generate the output binary string.

This encoding of $\mathbf{f} \in \mathbb{R}^1$ is equivalent to:

$$\boldsymbol{\alpha}^\star_{\text{BRIEF}} = \arg \min_{\boldsymbol{\alpha} \in \mathbb{B}^2_1} \|\boldsymbol{\alpha} - \Psi(\mathbf{f}, \bar{\mathbb{T}}^1_1)\|_2. \qquad (7)$$

Note that $\bar{\mathbb{T}}^1_1$ is equal to $\{-1, 1\}$, and hence the mapping is $\Psi(K(\mathbf{f}, 1), K(\mathbf{f}, -1))$. The SQ is into two elements ($Q = 3^1 - 1 = 2$), since $\mathbb{B}^2_1 = \{(1, 0), (0, 1)\}$, and it selects one of them depending on which is closer to Ψ. In case $\mathbf{f}$ is positive, $K(\mathbf{f}, 1) > K(\mathbf{f}, -1)$, SQ selects $(1, 0)$, otherwise $(0, 1)$. Observe that the output space of BRIEF is $\mathbb{B}^1$, which differs from $\mathbb{B}^2_1$ in SQ. Yet, it is the same using the Euclidean distance on $\mathbb{B}^2_1$ as using the Hamming distance on $\mathbb{B}^1$. We will show that when we increase the dimensionality of $\mathbf{f} \in \mathbb{R}^1$ we can achieve higher performance.

Sparse Coding. The relation between Sparse Coding and SQ has been already shown in [4]. In the Supplementary Material we introduce a new relation, which shows that the kernelized version of Sparse Coding, *c.f.* [11], is equal to the optimization of SQ plus the term $\sum_{ij} \alpha_i \alpha_j \Psi(\mathbf{b}_i, \mathbf{b}_j)$, which penalizes to select similar $\mathbf{b}$s. This shows that the kernelized version of SC and SQ only differ in a regularization term.

Convolutional Networks. Other types of successful encodings are the ones inspired by V1 architectures [13, 5]. They consist on computing the responses of the raw image patch to a large filter bank, and then, applying some non-linearity on the responses. We can see our formulation as a convolutional network. Let $\mathbf{W} \in \mathbb{R}^{q \times m}$ be the matrix containing the filters we use in our formulation to extract the features, and let $\mathbf{x} \in \mathbb{R}^m_+$ be the raw image where $\mathbf{W}$ is applied. Thus, $\mathbf{f} = \mathbf{W}\mathbf{x} \in \mathbb{R}^q$. In the Supplementary material we show that

$$\Psi(\mathbf{f}, \{\mathbf{b}_i\}) \propto \Psi(\mathbf{x}, \{\frac{1}{w}\mathbf{W}^T\mathbf{b}_i\}) \in \mathbb{R}^Q, \qquad (8)$$

where $\mathbf{W}\mathbf{b}_i$ is the linear combination of the filters $\mathbf{W}$ weighted by the entries of $\mathbf{b}_i$, and $1/w$ is to apply the l_2 normalization. Thus, we can see our formulation as a convolutional network, in which the mapping $\Psi(\mathbf{x}, \{\frac{1}{w}\mathbf{W}^T\mathbf{b}_i\})$ are the responses of the image to a large filter bank, $\{\frac{1}{w}\mathbf{W}^T\mathbf{b}_i\}$, and the SQ applies a non-linearity on the responses.

4. Design of New Patch Descriptors

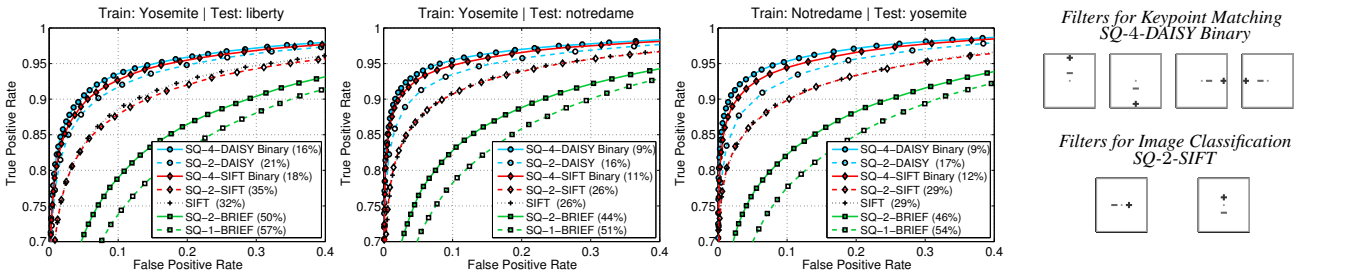
Now that we have introduced the formulation of SQ to feature encoding, in this section, we address the design of all the pipeline, not only the step of the encoding. Additionally, we introduce the learning of the parameters.

4.1. Pipeline and Implementation Details

For each step in the pipeline we propose several options, that can be incorporated depending on the requisites of efficiency and performance for the final application. In the experiments section we report results combining the different options, summarized at the end of this subsection.

Feature Extraction. In patch description, we divide feature extraction methods in *homogeneous*, that apply the same set of filters to extract all $\mathbf{f}_j$ in the patch, and *heterogeneous*, that change such set of filters. SIFT descriptor uses homogeneous feature extraction because it applies the same filters in all the patch, which are the horizontal and vertical gradients. In contrast, BRIEF uses heterogeneous feature extraction, since a different filter is applied to generate each output bit. For all cases, we use filters based on simple subtractions of two pixels, because they are very efficient to compute and to learn. Also, we did not observe any performance increase using denser filters than subtractions. For the homogeneous case, the filters are learned fixing their size at 6×6 pixels. For the heterogeneous, the two pixels that are subtracted are randomly generated as in [7].

Feature Encoding. We employ Def. 1 for feature encoding, optimizing it with Prop. 2. Increasing the number of filters and using our encoding gives significant improvements of the performance with only a small increase of the computational cost. We increase the number of filters to a maximum of 4 in all cases. This is because the encoding of 4 filter responses results on 80 dimensional vector ($3^4 - 1$), which is still efficient in practice. Yet, further increasing the amount of filters explodes the dimensionally, *e.g.* encoding 6 filters results in a 728 dimensional encoding. Also, we do not find any improvements in the performance for the applications in the experiments when increasing from 4 filters.



(a) (b) (c) (d)

Figure 1. *ROC curves for Brown dataset:* (a) Liberty learned on Yosemite, (b) Notredame learned on Yosemite, and (c) Yosemite learned on Notredame. (d) *Best learned filters* for (top) Keypoint Matching learned on Yosemite and for (bottom) image classification on Caltech101.

The similarity measure that composes the mapping Ψ is computed using a Gaussian kernel, because it is efficient in practice. SIFT uses the geodesic distance in the circle group for a two dimensional $\mathbf{f}$, but we do not observe any advantage of using such similarity measure over the Gaussian kernel. When we set all parameters of our formulation to reproduce SIFT, except for the similarity measure, there is not a significant difference in the performance between the original SIFT and our descriptor. The encoded features are ℓ_2-normalized.

Pooling. In the homogeneous feature extraction, the patch is divided into regions in which the encoded features are pooled together, and then the results for each of the region are concatenated. We exactly reproduce the poolings reported in SIFT [17] and in [26] for DAISY. SIFT divides the patch into 4×4 non-overlapping rectangular regions. In each region, the encoded features are averaged, and weighted. DAISY pooling divides the patch into circular regions grouped into rings. We use the best configuration found in [26], which uses 17 pooling regions. In the heterogeneous case, *i.e*. BRIEF-like descriptors, the encoded features can not be pooled together because the filters used at each location are different. Thus, the results of the encodings are simply concatenated. Other approaches that we could have incorporated are the successful attempts to learn the pooling regions [6, 21, 22].

Post-processing. It is desirable to have binary descriptors for efficiency and memory constraints. When the final descriptor is not binary, we can use SQ to quantize $\mathbf{y} \in \mathbb{R}^M$ to the space of r-sparse binary vectors $\mathbb{B}_r^M$, *i.e*., $\arg\min_{\hat{\mathbf{y}} \in \mathbb{B}_r^M} \|\hat{\mathbf{y}} - \mathbf{y}\|_2$, in which r is the parameter to set, and M the length of the pooled vector. We choose SQ because it is efficient to compute, only a simple sorting (Prop. 1), and for the symmetry with the formulation of the encoding. SQ does not aim at approximating the original vector $\mathbf{y}$, and indeed introduces quantization error [4]. Yet, this may allow for a better generalization properties and thus, it can achieve better results, as we show in the experimental section. The descriptors built using SIFT or DAISY pooling are post-normalized with a clipping normalization as reported by [6]. Other approaches that we could use are based on projecting the descriptor to a lower-dimensional space [14, 21, 19], or to a binary space, *c.f*. [12].

Summary. Now that we have introduced the overview of the pipeline, we summarize it in three descriptors that we evaluate in the experiments section. These are the following (we indicate the number of filters with q):

$-$*SQ-q-SIFT*: It uses our encoding and the same pooling as in SIFT [17].

$-$*SQ-q-DAISY*: It is the same setup as SQ-q-SIFT, except that it uses the DAISY pooling [26].

$-$*SQ-q-BRIEF*: The feature extraction is heterogeneous as in BRIEF [7]. In the feature encoding, we set $k = 1$ such that the output is directly binary, since it acts as a Hard Assignment.

4.2. Learning

Learning the parameters of the patch descriptor is a key step to attain state-of-the-art results. We follow a discriminative learning. It consists on iteratively generating a new instance of the parameters, and keeping them only if the performance increases. We vary several randomly chosen parameters at a time. We run the algorithm until no further improvement of performance is observed. Although the learning algorithm we propose is simple, it allows to learn the descriptors depending on the final application by optimizing the final performance.

This learning algorithm is feasible in practice because the descriptors are efficient to compute, and there are only few parameters that we learn. When we use DAISY or SIFT pooling, the parameters are initialized by randomly setting the filters, $k = 1$ and σ of the gaussian kernel to 10^{-3}, and we restrict $k \in (1, 2)$ and $\sigma \in (0, 1)$. After these parameters are learned, the parameter of the SQ in the post-processing, r, is set by validating all possible values. In the case we use heterogeneous extraction, *i.e*. BRIEF-like descriptor, the filters are randomly generated, as in [7], and not learned.

For image classification applications, there are mid-level features build on top of the patch descriptors [4, 27]. We set the parameters of the mid-level features to the best values reported in the literature [4], and then learn the patch descriptors by maximizing the classification performance.

		Patch Descriptor Properties for 1 Patch and 1 CPU						Keypoint Matching 95% error rate					Image Classification accuracy (%)	
		Speed (μs)						Train Yosemite		Train Notredame			Hierarchical SQ	
		Filter	*Encoding*	*Pooling*	*SQ*	*Total*	*Dimens.*	*Liberty*	*Notredame*	*Liberty*	*Yosemite*	*Speed Match (μs)*	*Caltech 101*	*VOC07*
SQ-q-DAISY	**SQ-4-DAISY Binary**	35	872	572	93	1572	1360	**15.52**	**8.52**	**15.6**	**8.81**	18	–	
	SQ-2-DAISY Binary	15	645	190	8	858	136	25.68	18.43	26.20	19.45	18	–	
	SQ-2-DAISY	15	611	218	–	843	136	21.07	16.10	21.07	17.28	84	–	
SQ-q-SIFT	**SQ-4-SIFT Binary**	25	962	240	73	1300	1280	17.00	11.19	17.88	12.41	18	69.3	42.90
	SQ-4-SIFT Binary *	7	185	70	59	320	1280	18.16	13.45	18.39	14.76	18	64.5	40.12
	SQ-2-SIFT Binary	18	573	228	8	827	128	31.02	23.49	28.04	24.70	18	74.4	52.45
	SQ-2-SIFT Binary *	4	123	65	6	200	128	33.26	26.45	31.38	27.72	18	73.7	51.64
	SQ-2-SIFT	18	568	200	–	785	128	34.94	25.93	34.94	29.06	77	**74.7**	**56.76**
SQ-q-BRIEF	**SQ-2-BRIEF (256 tests)**	2	3	–	–	4	2048	50.49	44.37	49.03	46.49	11	30.2	23.18
	SQ-1-BRIEF (256 tests)	2	2	–	–	3	256	55.83	49.34	55.83	51.38	3	30.1	22.32
keypoint matching	Simonyan [21] Project.		–			–	59	**16.27**	**7.11**	**13.63**	**10.36**	39	–	
	Brown [6] Project.		–			–	29	18.27	11.98	16.85	13.55	20	–	
	Simonyan [21]		–			–	576	18.47	9.71	17.81	10.65	> 200	–	
	Brown [6]		–			–	400	20.48	14.43	21.85	15.91	> 200	–	
	Best DAISY [26]		–			–	136	22.94	15.62	–	–	85	–	
	SIFT [17]		–			1000	128	35.09	26.10	35.09	28.50	77	–	
	SURF [3]		–			200	64	54.01	45.51	54.01	43.57	39	–	
	BRIEF [7]		–			3	256	57.15	50.96	57.15	53.63	3	–	
classification	SIFT + HA [4]		–			1000	128			–			**74.6**	**54.54**
	SIFT Binary + HA [4]		–			1000	128			–			74.2	52.87
	Hierarchical SC [27]		–			–	1024			–			74.0	–

Table 1. *Results on Brown, Caltech 101 and PASCAL VOC2007 datasets.* The name of our descriptors are SQ-q-pooling, in which q indicates the number of filters and pooling is DAISY, SIFT or BRIEF. When the descriptors are binary it is written, and the * indicates that the filters are extracted once every two pixels and the SQ is with $k = 1$.

5. Experiments

In this section, we report on experiments for two image matching benchmarks, namely this introduced by Mikolajczyk and Schmid [18] and Brown *et al*. [6]. We also evaluate our method on the standard image classification benchmarks of Caltech101 [10] and PASCAL VOC 2007 [9]. After giving the most relevant implementation details, we discuss the results that were obtained.

Dataset by Brown *et al*. It evaluates the retrieval of keypoints in a large database. It consist of three sets of patch correspondencies, sampled from the Statue of Liberty, the Notre Dame and the Half Dome at Yosemite. The patches are 64×64 pixels, and since the patches are provided, no keypoint detector is needed. There are 500k feature pairs that are provided and have to be evaluated. This set contains an equal number of positive and negative matches. We follow the standard experimental settings in the literature, *c.f*. [7, 21]. Each set has 100k pairs for testing. Training is done on a different set, using all its pairs. We calculate the ROC curves sweeping a threshold over the distance between the descriptors of the pairs. We also report the false positive rate when 95% of the true matches are found.

Dataset by Mikolajczyk and Schmid. This benchmark evaluates the robustness of patch matching to typical disturbances, covering viewpoint changes (*Wall* and *Graffiti*), blur (*Tree* and *Bike*), compression artifacts (*JPEG*) and illumination changes (*Light*). We use the standard evaluation procedure [7]. Each reference image is tested by matching detected keypoints to five different images, sorted in order of increasing difficulty. The match of a keypoint is its nearest-neighbour keypoint in the test image. The performance is evaluated with the quotient between the number of correct matches and the total amount of keypoints in the reference image.

We detect the keypoints with the SIFT implementation of [25], using default parameters. In order to evaluate the descriptors independently of the performance of the detector, the keypoints are validated using the ground-truth, and thus, we only match features that have a correct correspondence. To do so, we follow the same procedure as in [7]. We observed that when increasing the number of keypoints the matching rate decreases, but this happens proportionally for all evaluated methods, and the ranking is preserved. We report results with about 1000 keypoints because it is an appropriate number for most applications.

To compute the descriptor we use a patch size of 48×48 pixels in the upright position, with the keypoint in the center. Thus, we discard the scale and orientation, but not the smoothing, given by the SIFT detector. We observed that this yields the best performance in all tested cases. We use the same parameters learned for Yosemite in the dataset by Brown *et al*.

Caltech 101. It is a benchmark for image classification. It contains 102 different classes with about 50 images per class. We use 3 random splits of 30 images per class for training and the rest for testing. We report the average classification accuracy across all classes. We resize the image to have a maximum of 300 pixels per dimension, and do not use flipped or blurred images to extend the training set. For the mid-level features we us the same set-up as in [4], Hard-Assignment together with max-pooling, using a codebook of $8,192$, and dividing the image in 4×4, 2×2 and 1×1 regions for the spatial pyramids. Since Hard-Assignment can be formulated as a SQ [4], we can see the full network as a hierarchy of SQ from the pixel level. On top, we use a linear SVM.

PASCAL VOC 2007. It consist of around $10,000$ images with 20 different object classes, where half the dataset is used for training and the other for testing. The evaluation is based on the mean average precision (mAP) across all classes. We use the same mid-level features as in [4], using a hierarchy of SQ from the pixel level. We use a codebook

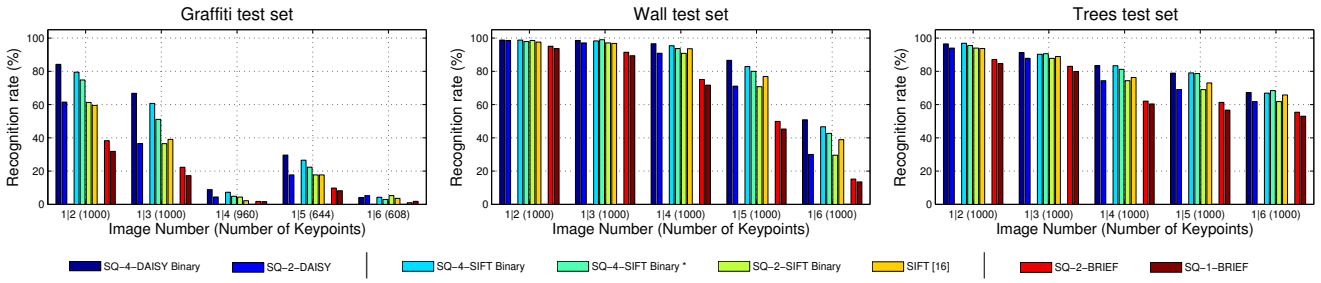


Figure 2. *Results in the dataset by Mikolajczyk and Schmid.* $1|x$ means that the keypoints of the reference image (number 1) are matched with the keypoints in image number x. The number of keypoints are indicated in parentheses for each pair of images.

of $16,258$ entries, and spatial pyramids of 3×3, 2×2, 1×1. The classification is learned with a linear SVM.

5.1. Results

We evaluate the descriptors introduced in Sec. 4. In what follows, we always indicate whether they are binary or not, and for SIFT and DAISY poolings, we indicate with an * when the filters are extracted once every two pixels and the SQ is with $k=1$, which is much faster to compute and does not require the computation of the gaussian kernel.

The computational cost is reported as the mean time in μs for 10 different runs. We use a single Intel i7 CPU @ 2.80GHz, which has the population count instruction, that allows for the fast computation of distances between binary vectors. No dedicated hardware or GPUs are used. In Table 1 we report results for the Brown, Caltech101 and VOC07 datasets. For both tasks, we can instantiate descriptors that put a very different focus on efficiency vs. accuracy. We now analyze both tasks separately.

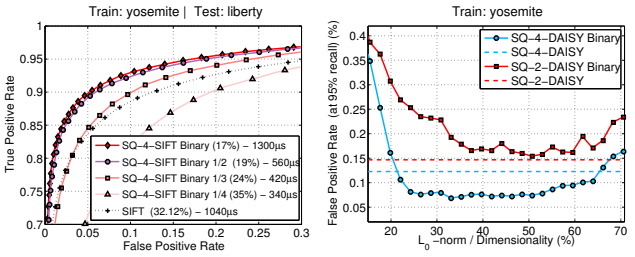
Performance on Keypoint Matching. Additionally to Table 1, we provide the ROC curves for the Brown dataset in Fig. 1. We can observe that when we mimic the descriptors in the literature with our formulation, SQ-2-DAISY, SQ-2-SIFT and SQ-1-BRIEF, we obtain similar results as with the standard descriptors, DAISY, SIFT, and BRIEF, respectively. This demonstrates that our formulation is indeed generalizing these descriptors. When we increase the number of filter responses that are encoded together, *i.e.* when we increase the parameter q, the performance increases for all cases: DAISY is improved by around 5-10% depending on the dataset, SIFT by 15%, and BRIEF by 5%. Encoding more filter responses seems to capture additional, relevant image details that are useful for keypoint matching. The method of [21] represents the current state-of-the-art for the Brown dataset. It uses the same encoding as in SIFT and DAISY, and it learns a pooling of 72 circular regions which are projected to a lower dimensional space. SQ-4-DAISY-Binary achieves similar results as this state-of-the-art. Compared to [21], SQ-4-DAISY-Binary needs only half the memory and is twice as fast, it is binary, and it comes without the computational overhead of pooling 72 regions and projecting them. SIFT pooling, which is less sophisticated than DAISY pooling, can also achieve results close to the state-of-the-art when using SQ-4-SIFT-Binary. Moreover, the computational cost of SQ-4-SIFT-Binary* is of the order of magnitude of the SURF descriptor (when comparing against its OpenCV implementation), and it obtains a 35% better error rate.

In Fig. 2 we report results on the Mikolajczyk Dataset, for the Graffiti, Wall and Trees images. By analyzing Fig. 2, we arrive at the same conclusions as for the Brown dataset. On the JPEG, Light and Bike images all descriptors get between 95% and 100% accuracy, and thus, no clearcut conclusions can be drawn. The results obtained for the SIFT and BRIEF baselines in the case of the Graffiti and Wall are of the same order of magnitude as reported in [7], but a direct comparison is not possible because the keypoints may differ. For SIFT and the Trees images, we get better performance than the one reported by [7]. This is because we use the smoothing of the SIFT detector, which is important to evaluate the robustness to blurring in the Trees images.

Performance on Image Classification. For Caltech101 and VOC07, results are reported in Table 1. Similar conclusions can be extracted from both datasets. SQ-2-SIFT achieves a higher accuracy. Note that for classification, increasing q and binarizing does not boost the performance as with keypoint matching. This shows the advantage of having a general formulation that can be adapted to the task at hand. We slightly outperform SIFT, but the gap in performance is not as high as in keypoint matching. This shows that SIFT is well-adapted to object recognition. Remarkably, SQ-2-SIFT-Binary* is much faster to compute than standard SIFT, yet achieves comparable results. We can see that the BRIEF descriptor performs poorly, since it has not been designed for image classification. Note that our hierarchy of SQ achieves comparable results to the hierarchy of Sparse Coding [27].

We do not report results with DAISY pooling, because the best DAISY is selected based on the keypoint matching task, and without properly learning the pooling, the results would be distorted. It is unclear how to learn the pooling of the patch descriptor for image classification, since in this case there is no labeling to supervise the learning at the patch description level.

Efficiency Evaluation. We report computation times in Table 1. The speed of matching is evaluated with computing the distance of 1 patch to 512 patches. Additionally,



(a) (b)

Figure 3. *Brown* et al. *dataset:* Impact (a) of the density of the feature extraction, and (b) of the binarization with SQ.

we observe that the computation of feature encoding with Alg. 1, for $q = 4$, is 4 times faster than directly calculating Ψ. Alg. 1 takes 872ms, and calculating Ψ takes 3710ms.

Impact of Encoding Parameters. In Fig. 1(d) we show the result of learning the filters. Note that for image classification the subtracted pixels are placed more in the center than for keypoint matching. We also tried using a random codebook for the feature encoding instead of the set $\bigcup_{0<p\leq q} \bar{\mathbb{T}}_p^q$, as well as a codebook generated using k-means, and in both cases, they perform equally well. Thus, the precise composition of the codebook has a small impact on the final performance. Yet, this is not the case for the efficiency, since we can not use Alg. 1 together with the random codebook or a codebook generated using k-means.

Impact of Filtering Density. In Fig. 3(a) we report the impact of extracting features not at every pixel in the patch, but sparse, when using the descriptor SQ-4-SIFT-Binary. Extracting the features only in half the pixels, the performance does not go down. SQ-4-SIFT-Binary* and SQ-2-SIFT-Binary* can achieve a higher efficiency when exploiting this fact, together with setting $k = 1$.

Impact of Binarization. In Fig. 3(b) we analyze the impact of binarizing the descriptors with SQ for keypoint matching. We see that for $q = 2$, the performance is similar in the binary and non-binary case, and for $q = 4$, the binary descriptor obtains a better performance. This shows that the quantization error introduced by the SQ can be beneficial to achieve better generalization properties. For image classification, in Table 1, we can see that the binarization does practically not lower the performance. Note that the time to compute the SQ for the binarization is only a small fraction of the overall cost, since it only consists of sorting the entries of the descriptor.

6. Conclusions

We presented a formulation for patch description based on sparse quantization. We showed that our formulation generalizes SIFT and BRIEF, and that we can instantiate new descriptors that can take binary values. Experiments show that our descriptors achieve state-of-the-art results in keypoint matching, and comparable to SIFT on image classification, yielding dramatic speed-ups.

Acknowledgements: This work has been in part supported by the European Commission projects RADHAR (FP7 ICT 248873) and IURO (FP7 ICT 248314).

References

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. In *CVPR*, 2012.

[2] M. Ambai and Y. Yoshida. CARD: Compact and real-time descriptors. In *ICCV*, 2011.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. *CVIU*, 2008.

[4] X. Boix, G. Roig, and L. V. Gool. Nested sparse quantization for efficient feature coding. In *ECCV*, 2012.

[5] H. Bristow and S. Lucey. V1-inspired features induce a weighted margin in SVMs. In *ECCV*, 2012.

[6] M. Brown, G. Hua, and S. Winder. Discriminative learning of local image descriptors. *PAMI*, 2011.

[7] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a local binary descriptor very fast. *PAMI*, 2011.

[8] A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, 2011.

[9] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes challenge 2007 (VOC2007). *IJCV*, 2010.

[10] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *PAMI*, 2006.

[11] M. T. Harandi, C. Sanderson, R. Hartley, and B. C. Lovell. Sparse coding and dictionary learning for symmetric positive definite matrices: A kernel approach. In *ECCV*, 2012.

[12] J. Heinly, E. Dunn, and J.-M. Frahm. Comparative evaluation of binary features. In *ECCV*, 2012.

[13] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009.

[14] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptor. In *CVPR*, 2004.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *CVPR*, 2011.

[17] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[18] K. Mikolajczyk and C. Schmid. A performance evaluation of local image descriptors. *PAMI*, 2005.

[19] J. Philbin, M. Isard, J. Sivic, and A. Zisserman. Descriptor learning for efficient retrieval. In *ECCV*, 2010.

[20] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *ICCV*, 2011.

[21] K. Simonyan, A. Vedaldi, and A. Zisserman. Descriptor learning using convex optimisation. In *ECCV*, 2012.

[22] T. Trzcinski, M. Christoudias, V. Lepetit, and P. Fua. Learning image descriptors with the boosting-trick. In *NIPS*, 2012.

[23] T. Trzcinski and V. Lepetit. Efficient discriminative projections for compact binary descriptors. In *ECCV*, 2012.

[24] J. C. van Gemert, C. J. Veenman, A. W. M. Smeulders, and J. M. Geusebroek. Visual word ambiguity. *PAMI*, 2010.

[25] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[26] S. Winder, G. Hua, and M. Brown. Picking the best daisy. In *CVPR*, 2009.

[27] K. Yu, Y. Lin, and J. Lafferty. Learning image representations from the pixel level via hierarchical sparse coding. In *CVPR*, 2011.