‘Διαιρεῖ καὶ βασίλευε’ (Translation: “Divide and rule”)

– Philippos II of Makedon (382-336BC)
Figure 42: Codemaps segment and classify objects locally by reordering the encoding, pooling and classification steps of object classification. Different from existing linear decompositions for specific pipelines, codemaps are generic, embed fast $\ell_2$-normalization, induce classification by attributes with locality and include recursive nonlinearities by local kernel pooling.

From the previous chapter we identified that segmentation, and locality in general, can be of great importance for the accurate recognition of concepts. Hence, we study further how locality can be better integrated in the classification process.

6.1 introduction

It remains remarkable that the great successes in object recognition use so little of the spatial order in the image. Features [81] are encoded in feature space [58, 104, 124, 156], pooled in histograms [17, 156] and plugged into a kernel-classifier [32, 162]. The entire chain contains no more spatial information than the locality of the features, compensated by a rather crude method of spatial pyramids [51, 72] where the standard classification procedure is repeated over upper and lower parts of the image. To make progress in recognition, better inclusion of more spatial information is inevitable. And indeed, recently, [4, 21, 22, 54, 97, 135, 138] have introduced the spatial coherence of objects, by confining the usual analysis to selected regions in the image. The spatial restriction has had a positive effect on the recognition result. However, the selection of regions is only loosely coupled to the classification pipeline. In this paper, we aim to integrate locality much further into the analysis.

Starting at the other end, the route of first segmentation then recognition, is as old as Blob-world [24], where parts were described as visually coherent regions. In [68, 145] the regions were jointly modeled to establish semantic similarity between adjacent object parts. And indeed, consistent regions lead to useful object hypotheses [4, 21], paving the way towards object segmentation. Pushing localization with state-of-the-art encodings to an extreme, [33] considers pixels independent to each other for semantic segmentation. In this case it is pixels that are classified as belonging to an object class and not regions segmented to contain an object class instance, as computing Fisher vectors on thousands of regions in the traditional, sequential pipeline would be too expensive to consider. The same holds for human-recognizable representations with attributes [70], where the complexity of such encodings has restricted analysis to full images only, without any locality. We argue in this paper the virtue of a deeper connection between spatial localization and (fine-grained) object type classification on the basis of state-of-the-art
6.2 related work

We aim to combine region-level image decompositions with recognition at the earliest stage of the analysis. We show that reordering the processing steps for object type classification into local pooling before classification has considerable advantages. Where [69, 146] have shown the efficiency benefits of such decompositions for unnormalized bag-of-words with a linear classifier, codemaps make three novel contributions. Prior to these, we formulate the sufficient mathematical conditions under which image encoding and classification are locally decomposable (Section 6.3). In the first novelty, we use this result to introduce codemaps with $\ell_2$-normalization for arbitrarily shaped image regions (Section 6.4), essential to reaching a better than state-of-the-art performance in semantic segmentation [4, 21, 22, 48, 97]. In the second novelty, we show that the independence of the number of classifiers allows codemaps to incorporate locality in attribute representations. We show that fine-grained categorization using attributes profits substantially from locality (Section 6.5). In the third novelty, we focus on situations when the lattice is fixed, such as in an object classification setting (Section 6.6). We show that codemaps introduce recursive layers of nonlinearities in a feedforward architecture, improving object type classification accuracy for free.

We improve upon our previous work at ICCV [79] by introducing the localized attribute representation based on codemaps, a complexity analysis (Section 6.7) and more extensive experiments (Section 6.8). We illustrate an overview of the proposed method in Fig. 42.

6.2 related work

We structure our discussion on related work by the components of the proposed method, that is feature encoding and pooling, semantic segmentation, fine-grained classification by attributes and kernel classification.

6.2.1 Feature encoding and pooling

Feature encodings capture the visual information around a local neighborhood and generate a measurement. This measurement is supposed to be invariant to accidental circumstances, such as illumination, shade, occlusion etc. We consider two families of approaches.

The first family of approaches employs hand-crafted feature encodings to generate a code [9, 34, 63, 81]. The acquired codes are projected to a, usually, higher dimensional space. These per-image projections are the image representations used for classification. Traditionally, the most popular projection has been vector quantization [32, 124] of low level codes, usually referred to as the bag-of-words model. Replacing the hard constraints of one-to-one correspondence between low level codes and bag-of-words projections, soft quantization [141] and sparse coding [156] allow for more flexible, and thus more accurate, representations. And recently, modelling the differences of projections to pre-trained models have shown state-of-the-art results in classification. The two most popular choices are Fisher vectors [104] that compute the second-order moments of gaussian mixture models [104], and VLAD vectors [58] that compute the $\ell_1$ distances of the local features from the centres of $k$-means codebooks. For a more extensive review on feature encodings we refer to [55].

Instead of employing hand-crafted features, the second family of approaches learns to generate feature encodings. These methods, better known as deep learning or convolutional neural networks, propose successive, hierarchically organized graph layers that are trained to reconstruct their input [10]. Because of the hierarchical structure which feeds the output of one layer as
input for the next one, each hierarchical level captures increasingly more complicated visual patterns [161]. The activations of these patterns become the feature encodings that describe an image in a multi-scale manner. Combining these feature encodings with discriminative classifiers yields excellent results [66, 160] in large scale benchmarks, such as ImageNet [36]. On the negative side, however, in training such a massive number of parameters lures the danger of overfitting. For this reason deep learning methods are usually reliable only in very large scale scenarios, where the amount of data counterweights the number of free parameters. Interestingly, it was recently shown [123] that organizing traditional feature encodings, such as Fisher vectors, in a similar manner to deep architectures leads to comparable accuracies in the same large scale datasets. For an extensive review of feature learning we refer to [10].

Both hand-crafted, as well as learned feature encodings, involve a feature pooling step. Feature pooling spatially aggregates the relevant local feature encodings into a global image representation. Average pooling has been shown to work best for bag-of-words [124] and Fisher vectors [104]. Max-pooling is proven effective for sparse coding [156] and deep learning approaches [66], whereas sum-pooling is used by VLAD vectors [58] and deep Fisher networks [123].

The representation we propose is agnostic to the encoding choice, as long as it is locally decomposed. In our experiments we use Fisher vectors, since they have shown to yield state-of-the-art results in object classification [104], without requiring extra images for pre-training. Furthermore, we generalize on the pooling functions. We show that pooling over a region of interest is equivalent to a simpler two-level pooling for a particular family of mathematical functions. This two-level pooling allows to classify objects locally, while offering a substantial efficiency speed-up.

6.2.2 Semantic segmentation

For semantic segmentation we identify three main approaches, most of them starting from higher level image elements such as superpixels [75] or regions [5]. The first approach groups superpixels on the basis of semantic similarity in a conditional random field. Using concept co-occurrences [68], higher order potentials [65], or even alleviating the requirement for ground truth segmentation masks during training [145], these models are favorable for their good performance. The second approach directly classifies low level image elements, such as individual pixels or superpixels. Employing random forests [122], patch-level classifiers [33] or classifying superpixels following a self-learning approach [25], these methods do not formally consider higher level interdependencies between pixels (or superpixels). The third approach of methods decomposes the detection and segmentation into two phases, first detecting possible segment hypotheses [5, 23], then classifying them following modern classification procedures [4, 21, 23, 48].

We focus on the third family of approaches, and more specifically on the state-of-the-art CPMC-O2P [21], for two reasons. First, CPMC-O2P purposefully does not include any feature encodings for segment representations and only performs pooling from the raw SIFT features. Therefore, adding encoded features, will be complementary to the contributions of the CPMC-O2P features. Second, one explanation why CPMC-O2P does not use encoded features is that they typically require strong learning machinery, such as exponential $\chi^2$ kernels [22]. For CPMC-O2P, however, fast linear kernels suffice, hence involving heavy nonlinear kernel machines would prevent the method from being both accurate and efficient. In contrast, our proposed representations also operate with linear kernels. They can, therefore, be gracefully combined with the state-of-the-art CPMC-O2P features, maintaining the high efficiency and improving accuracy further. Our experiments reveal that we improve semantic segmentation accuracy considerably, leading to the highest reported scores in PASCAL VOC 2011 and 2012, the leading semantic segmentation benchmarks.
6.2.3 Fine-grained classification by attributes

Fine-grained classification tackles tasks, where one needs to distinguish between visually very similar categories, for example telling apart similar bird species such as a Forster’s Tern from the Least Tern. From both psychological studies [12], as well as empirical evidence [26, 44, 165], it became evident that locality plays a pivotal role in fine-grained classification.

In fine-grained classification it is assumed that all categories belong to a common super-category, e.g. birds [148]. The defining details for these fine-categories are expected to be found in the little fluctuations of common bird properties, such as “curved beak” or “spotted breast”. Learning, therefore, such properties, better known as attributes, has been a subject of intense research. In their seminal work [70] propose to use attributes for the recognition of animal categories, even when no training examples for some of the categories are available. In [150] a joint model of visual attributes and object classes is learnt, whereas [1] propose to learn attribute embeddings for fine-grained and zero-shot classification.

Attributes refer to object-level properties. Therefore, it makes sense to look for attributes on an object, and not on an image level. Yet, only image-level attribute representations have been considered so far in the literature. The apparent reason is that being oblivious to the object location, sequentially evaluating all possible object locations where attributes could reside, would be computationally demanding. To the best of our knowledge there has not been any work reporting multitudes of attribute representations for hundreds of regions in an image. The closest match is [38], who have the opposite goal of discovering locally explainable attributes, that is recurring visual patterns on consistent object locations. Apart from this conceptual difference, the algorithm of [38] requires human assistance, hence it is only applicable on small datasets. In contrast, our proposed automatic approach is designed to facilitate an effortless and accurate computation of already-known attribute representations for hundreds of image regions inside an image, opening the door to local attribute representations.

6.2.4 Kernel classification

For object type classification, support vector machines have repeatedly shown [36, 39, 125] to outperform or to be competitive to other alternatives [123]. To cope with the growing number of images, the size of the image representations and the numbers of object types, recent research has focused on efficient learning and classification.

Kernel properties, such as additivity and homogeneity, have been exploited for speeding-up support vector machines [82]. Special care has been given to nonlinear kernels [82, 144]. For most nonlinear kernels usually $\ell_1$-normalization is the preferred choice [144]. However, recent advances in feature encoding such as Fisher vectors [104] were shown to thrive with linear kernels, as long as certain guidelines are followed. For linear kernels $\ell_2$-normalization is preferred, not only for making the encodings invariant to scale [104], but also because they are the natural normalization procedure for linear kernel machines [144]. Indeed, as our experiments also reveal, the presence of $\ell_2$-normalization is integral for maintaining the highest accuracy in several classification and segmentation tasks.

No matter whether linear or nonlinear classifiers are applied, the spatial properties of object and scene classes naturally play an important role. Yet, orderless encodings, like bag-of-words of Fisher vectors, discard such spatial information. To compensate for this loss, classifiers currently employ the local origin of the data only weakly, with the most popular method being spatial pyramids [51, 72]. Spatial pyramids start from the rigid assumption, that the same image regions will contain similar visual statistics. For example a boat will usually appear within a water frame on the lower part and sky on the upper part of the image. A negative aspect of spatial
pyramids, however, is that they multiply the final number of dimensions by a factor analogous to the number of spatial divisions. Moreover, spatial pyramids require parsing the same image regions several times for each of the spatial divisions. These drawbacks only get accentuated when more sophisticated, class-specific spatial pyramid variants are considered, since they divide images in multiple, different spatial layouts, one per class [119]. By considering locality from the very beginning, codemaps incorporate the spatial extent of an image gracefully, affecting positively the accuracy of the final classification without imposing extra computational burdens.

6.2.5 Contributions

We are inspired by [69, 146] who observe that the image interpretation of unnormalized bag-of-words with a linear classifier can be analyzed in terms of the contributions of individual descriptors, leading to a considerable efficiency gain. In the current work we propose to reorder the steps of pooling and feature encodings, such as Fisher vectors and VLAD. By doing so, we obtain a joint formulation of the classification score and the local neighborhood it belongs to. Furthermore, the generalized framework obtains the precise $\ell_2$-normalized classification score for any region, which is known to increase the accuracy for both Fisher vectors and VLAD considerably [58,104]. Also, by exploiting the robustness of codemaps for an increasing number of classifiers, codemaps allow for injecting attribute representations with locality. Hence, methodologies that depend on attribute representations, such as label or attribute embedding [1, 151], now benefit from locality. In addition, codemap kernel pooling which embeds nonlinearities by explicit or approximate feature mappings [82, 144] assures state-of-the-art competitiveness [104]. And last but not least, we show that by employing recursive layers of nonlinear kernel poolings with codemaps has positive effects in classification accuracy.

6.3 Preliminary

We start from a lattice, composed of $N$ nodes, $G = \{g_i\}, i = 1, \ldots, N$, superimposed on an image. To ensure good generalization and flexibility, we consider that i) each node $g_i$ of the lattice is arbitrarily sized, shaped, and non-overlapping, i.e. $g_i \cap g_j = \emptyset$, $\forall g_i, g_j \in G, i \neq j$, and ii) each area $R$ where we search for the objects of interest is composed of multiple nodes $R = g_1 \cup \ldots \cup g_l$. Thus, regions are also arbitrarily sized and shaped. Hence, the image search is no longer confined to specific and limiting templates, such as rectangular areas [69,138]. For ease of reading we shall refer to each node $g_i$ of the lattice as lex. Our theory holds for all types of patches, including cells on a regular lattice [72], generalized image regions [5], superpixels [75,109] or any other type of localities.

We extract a collection of local features $z_i, i = 1, \ldots, M$ in the image and encode them to equal codes $c_i, 1, \ldots, M$. The pooling function $h(R)$ combines these local codes within the region $R$ to arrive at its global feature encoding.

**Codemaps.** The goal for a codemap is to be a decomposed object image representation. We begin with unnormalized codemaps. Using the classifier function $f$ the per lex classification score is $f(h(g_i))$. For an image region $R$ composed of $l$ lexes $R = g_1 \cup g_2 \cup \ldots \cup g_l$ the corresponding classification score is

$$f(h(R)) = f(h(g_1 \cup g_2 \cup \ldots \cup g_l)). \quad (6.1)$$

We want the codemap to be decomposable with respect to the contribution of each lex $g_i$ separately. We formally describe this property by

$$f(h(R)) = q(f(h(g_1)), f(h(g_2)), \ldots, f(h(g_l))), \quad (6.2)$$
where \( q \) is a classification pooling function that aggregates the localized classifier decisions over a region of interest. From eq. (6.2) we see that the pooling function \( h \) needs to be applied to each of the lexes \( g_i \) separately. Taking into account eq. (6.1), we arrive at the first condition for obtaining a decomposable codemap:

**Condition 1** The pooling function \( h : R \rightarrow B \) must be homomorphic from the space \( R \) to space \( B \), so that

\[
h(R) = h(U_R[g_1, g_2, ..., g_l]) = U_R[h(g_1), h(g_2), ..., h(g_l)] \quad (6.3)
\]

where \( R \) refers to the spatial domain formed by lexes \( \{g_i\} \), and \( B \) refers to the code pooling space defined by \( h \). In eq. (6.3) the \( U_R \) stands for the union set operation, that is \( U_R = \cup (g_1, g_2, ..., g_l) \), whereas \( U_B \) is an operation in \( B \) that makes \( h \) homomorphic. When \( h \) stands for sum pooling or max pooling, \( U_B \) is the sum operator or max operator. In practice a homomorphic pooling means that we can first locally pool the encodings from each lex \( g_i \) separately, then combine them to get the global feature encoding as if we operated on \( R \) in the first place.

In addition, we want the classifier \( f \) to also operate on each of the lexes \( g_i \) individually. By combining eq. (6.1) and (6.3), we arrive at the second condition for obtaining a decomposable codemap:

**Condition 2** The classification function \( f : B \rightarrow C \) must be homomorphic from the space \( B \) to space \( C \), so that

\[
f(h(R)) = f(U_B[h(g_1), h(g_2), ..., h(g_l)]) = U_C[f(h(g_1)), f(h(g_2)), ..., f(h(g_l))] \quad (6.4)
\]

where \( C \) refers to the classification space. Having a homomorphic function for the classifier \( f \), one only needs to consider the individual scores of the lexes within \( R \).

Normally, when classifying a region we first perform a global pooling on all the feature encodings contained in the region, and then we apply the classifier. However, according to Cond. 1, codemaps first break the global pooling into a collection of local feature poolings over lexes. Then, according to Cond. 2, codemaps apply the classifier on the local feature poolings and perform a global pooling on the classification scores of the lexes. Hence, the global pooling is performed on single scalars instead of high dimensional vectors. This brings significant efficiency benefits for vision tasks where thousands of regions need to be classified per image.

We conclude that in order to obtain a codemap, Cond. 1 and 2 need to be satisfied.

### 6.4 \( \ell_2 \) normalization for regions

Modern feature encodings, such as Fisher vector, VLAD or bag-of-words, usually include a summation operator in the feature pooling function \( h \). When a linear classifier \( f \) is used, the classification score of a region is \( y(R) = w^T h(R) = \sum_d \sum_{g_i \in R} w_d h_d(g_i) \), where \( w \) denotes the learned \( d \) dimensional weights by the linear classifier. This leads to a valid codemap, since

\[
y(R) = \sum_{i=1}^l y(g_i), \quad (6.5)
\]

where \( y(g_i) = \sum_d w_d h_d(g_i) \). A similar decomposition was derived in [69, 146] for the specific case of unnormalized bag-of-words with a linear SVM. Here we consider the important case...
of including normalization to feature encodings before classification [104, 144]. Including normalization, in particular \( \ell_2 \), for variable spatial regions is difficult to do efficiently. We propose \( \ell_2 \)-normalization for arbitrary regions in codemaps.

In general, for a region \( R \) the norm of its feature encoding vector \( h(R) \) is denoted by \( \|L_R\|_2 \). Because of the linear classifier we can rewrite the normalized classification score as:

\[
y(R) = f\left( \frac{1}{\|L_R\|} \cdot h(R) \right) = \frac{1}{\|L_R\|} \cdot f(h(R)). \tag{6.6}
\]

As eq. (6.6) indicates, to obtain the normalized classification score we can postpone the scaling by the inverse norm \( \frac{1}{\|L_R\|} \) until after the classification pooling. Thus, with codemaps, normalization boils down to multiplying a normalization scalar with the scalar classification score of a region, rather than with high dimensional feature encodings. We consider the \( \ell_2 \)-norm of the feature encoding for a region \( R \) within an image, since the linear classifier favors \( \ell_2 \)-normalization [144], as

\[
\|L_R\|_2 = \left( h(R)^T h(R) \right)^{1/2} \tag{6.7}
\]

\[
= \left( \sum_{i=1}^l \sum_{j=1}^l h(g_i)^T h(g_j) \right)^{1/2}.
\]

We calculate the \( \ell_2 \)-norm of a region \( R \) only from the sum of the pair-wise dot product \( h(g_i)^T h(g_j) \) between feature encodings of the lexes within the region. To generalize for any arbitrary region \( R \), we calculate the dot products of all the pair-wise lex combinations in the image. Then, we only need to consider the combinations of lexes that both appear in \( R \), that is:

\[
\|L_R\|_2 = \left( \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j h(g_i)^T h(g_j) \right)^{1/2} \tag{6.8}
\]

where the binary vector \( \alpha = (\alpha_1, ..., \alpha_N) \) indicates whether each lex is present or not within the region \( R \). Finally, we compute the \( \ell_2 \)-normalized classification score of an arbitrary region \( R \) as:

\[
y(R) = \frac{1}{\|L_R\|_2} \sum_{i=1}^N \alpha_i w^T h(g_i). \tag{6.9}
\]

We describe the \( \ell_2 \)-normalized codemap of an image as:

\[
\Phi_{\ell_2} = \{ g_i, w^T h(g_i), h(g_i)^T h(g_j) \}, \tag{6.10}
\]

for \( i, j = 1, ..., N \), where \( w \) is the classifier weight vector.

**Fisher codemaps.** The popular Fisher vector [104], extracted from a Gaussian Mixture Model with a probability density function \( u(\cdot; \mu, \sigma) \) is equal to \( c_z = \frac{1}{M_R} \nabla_{\mu, \sigma} \log u_{\mu, \sigma}(z_i) \), where \( M_R \) stands for the number of local descriptors \( z_i \) sampled from an image. A codemap is independent of the regions \( R \), hence the value of \( M_R \) is not available. However, \( M_R \) is canceled out during the \( \ell_2 \)-normalization, therefore we propose to drop the constant \( M_R \) from the original Fisher vector by using \( \tilde{c}_z = \nabla_{\mu, \sigma} \log u_{\mu, \sigma}(z_i) \). Since we use the sum operator for the feature pooling and the sum operator due to the linear classifier, the Cond. 1 and 2 are fulfilled and we have a Fisher codemap.

Fisher codemaps allow for a considerable speed-up when classifying a number of arbitrarily sized and arbitrarily shaped regions, as we will also outline in Section 6.7. For evaluating 1,000 regions the Fisher codemap needs 23 seconds per image, as compared to 22 minutes, when
using the traditional Fisher vectors without any alterations. The speed-up factor improves further when more evaluations are required. We conclude that \( \ell_2 \)-normalized Fisher codemaps are mathematically equivalent to Fisher vectors, but much faster. Similar formulations and efficiency benefits can be derived for other feature encodings, e.g. bag-of-words or VLAD, as well.

6.5 Localized Classifier Embedding

In the previous sections of this chapter we have established how to acquire an \( \ell_2 \)-normalized codemap, given that we have a single classifier. We now consider the case, when we have \( L \) classifiers in a matrix format \( W = [w_1, ..., w_L] \). We observe that replacing the single classifier \( w \) with multiple ones \( W \) in eq. (6.10) we end up with two matrix multiplications. The first one is between a set of \( L \) classifiers encoded in a matrix format, \( W = [w_1, ..., w_L] \), and the local feature encodings \( \{h(g_i)\} \) extracted from the \( N \) lexes in the image. The second one is between the local feature encodings themselves. The complexity of the second multiplication is quadratic to the number of lexes. Hence, we can easily increase the number of classifiers \( L \) until it matches to the number of lexes \( N \), without changing significantly the computational order of a codemap.

As codemaps are insensitive to the number of regions and classifiers we are able to train multiple classifiers and evaluate them in different image regions. A good starting point is, therefore, employing the ALE [1] objective criterion adapted for classification of objects by attributes [1]. Based on ALE, we have the classification score function for a class \( c \)

\[
F(c, h(R); V, W) = v_c W^T h(R).
\]  

(6.11)

The matrix \( W \) stands for an encoding-to-attribute classifier matrix, which transforms the low level feature encodings \( h(R) \) of a region \( R \) into a vector of attribute scores that are shared amongst our classes. The vector \( V = [v_1, ..., v_C] \) stands for the attribute-to-class classifier matrix, which uses the attribute representation of \( W^T h(R) \) of a region \( R \) to return the scores for classes \( 1, ..., C \). The prediction \( c^* \) to one of the classes corresponds then to the class that obtains the maximum score, that is

\[
c^* = \arg\max_c F(c, h(R); V, W) = \arg\max_c VW^T h(R),
\]  

(6.12)

where \( c \) is a running index over the columns \( V \) that contain the attribute-to-class classifiers.

To classify a new object we need to know the attribute matrix \( W \) and the attribute-to-class matrix \( V \). To this end we follow the training procedure of [1, 151]. Namely, given the encodings \( \mathcal{H} = \{h(R_m), y_m, m = 1, ..., H, y_m = \{1, -1\} \) for the \( R_m \) segment hypotheses in the training set, we minimize

\[
\arg\min_{W, V} \mathcal{L}(\mathcal{H}; V, W) + \beta \|V - V_{prior}\|^2,
\]  

(6.13)

where \( \beta \) is a weight to be tuned. \( V_{prior} \) is the prior information regarding the connection between attributes and classes, and \( \mathcal{L} \) is a standard one-vs-rest max-margin objective.

**Attribute codemaps.** The image classification by attributes proceeds in two steps. First, the low-level feature encodings are mapped to attributes via \( W \), then the attributes are mapped to object classes via \( V \). From eq. (6.11) we see that the attributes are shared amongst all classes forming an intermediate representation, equal to linear classifiers, which satisfy Cond. 2. Therefore, such an intermediate representation can be formulated in terms of codemaps,

\[
\Phi_{w} = \{g_i, W^T h(g_i), h(g_i)^T h(g_j)\},
\]  

(6.14)

for the \( i, j = 1, ..., N \) lexes and the \( W = [w_1, ..., w_L] \) attribute classifiers. We refer to eq. (6.14) as **attribute codemap**. Attribute codemaps are obtained in a similar manner to eq. (6.10). The differ-
ence is that attribute codemaps return an region representation vector and not object classification scores.

At test time we do not know which segment hypotheses contains the most accurate representations. Hence, we first need to detect the optimal segment and then classify it. Starting from promising object segment hypotheses $R_m$ from exterior algorithms [23], we can effortlessly compute the respective attribute representations $u(R_m)$ with our attribute codemaps, that is

$$u(R_m) = \frac{1}{\|L_{R_m}\|_2} [\sum_{g \in R_m} w_T^R h(g), \ldots, \sum_{g \in R_m} w_T^L h(g)]^T,$$

where $\|L_{R_m}\|_2$ is the $\ell_2$-normalization scalar for region $R_m$ and can be computed from eq. (6.8).

Having computed the attribute representation for all the segment hypotheses, we now rank segment hypotheses. The ranking is performed with the use of segment detection algorithm [22], using the computed attributes as the describing features for a segment. The evaluation returns simultaneously the best candidate segment containing accurate attributes and a label prediction for the most likely object category given these attributes, according to eq. (6.12).

6.6 **Nonlinear Kernel Pooling**

In the analysis so far we made no particular assumption with respect to the lattice geometry, nor the task that facilitates these image comparisons. Next, we will elaborate further on the case when the lattice is fixed and shared across all images, like in image classification. To do so, we first approach codemaps from a kernel point of view.

Assume two images $X$ and $Z$, for which we compute the feature encodings $h(\cdot)$ from all the lexes per image. For feature encodings that conform with Cond. 1, the linear kernel similarity of the two images is

$$K_L(X, Z) = h(X)^T h(Z) = \sum_{g_x \in X} \sum_{g_z \in Z} h(g_x)^T h(g_z).$$

(6.16)

Based on eq. (6.16) we make two observations. First, the lex comparisons between $g_x, g_z$ are linear, inner products. Having inner products, we can apply the kernel trick [116] for measuring the image similarity, now on the fine level of pairwise lex comparisons, namely

$$K'(X, Z) = \sum_{g_x \in X} \sum_{g_z \in Z} k(h(g_x), h(g_z))$$

$$= \sum_{g_x \in X} \sum_{g_z \in Z} \psi(h(g_x))^T \psi(h(g_z))$$

(6.17)

where $h'(X) = \sum_{g_x \in X} \psi(h(g_x))$ and $\psi(\cdot)$ stands for nonlinear kernel mappings, obtained either with explicit feature maps [144] or kernel PCA [116]. We observe that $h'(X)$ stands for a sum pooling operation of nonlinear feature encodings of the lexes in image $X$. We, thus, call this operation **nonlinear kernel pooling**.

Based on closure properties, as long as the summands $k(\cdot)$ are positive definite kernels [116], the codemap kernel $K'(X, Z)$ is also positive definite. We can, therefore, use all the appropriate kernels from the related literature, like the Hellinger, $\chi^2$ or histogram intersection kernels. Note here the relation of codemaps to match kernels [15] that operate on sets of local features, e.g. SIFT, and the Naive Bayes Nearest Neighbor kernel [131]. However, there are two differences with these works. First, codemap kernels focus on measuring similarities using strong feature
Figure 43: Codemaps over a fixed \(3 \times 1, 2 \times 2, 3 \times 3\) lattice. Considering the particular case of the upper division (green layer), the pyramid codemap computes the feature encoding as the summation \(h'(R_{\overline{3x1}}^{\text{upper}}) = \sum_{i=1}^{4} \psi(h(g_i))\). Introducing another layer of nonlinearities, \(K''(h'(X), h'(Z))\), we obtain codemaps in a feedforward pyramid codemaps.

Pyramid codemaps. Operating on a fixed lattice, we need to define the regions we want to compare between images. Such regions could be learnt optimally. Alternatively, we could adopt the regions derived from spatial pyramid divisions [51, 72]. As our aim is to investigate the codemap properties in capturing the spatial extent of an image composition, we opt for spatial pyramid-like regions for simplicity.

More specifically, from all the possible lex combinations we consider the ones that could correspond to spatial pyramid divisions. For example, see the lattice produced by the \(3 \times 1, 2 \times 2, 3 \times 3\) divisions in Fig. 43. Isolating the upper row region \(R_{\overline{3x1}}^{\text{upper}}\) of the \(3 \times 1\) pyramid, we see that it is composed of the lexes \(g_1, \ldots, g_4\). From eq. (6.17) we compute the image kernel similarity as \(h'(R_{\overline{3x1}}^{\text{upper}}) = \sum_{i=1}^{4} \psi(h(g_i))\). Similar to [72], to obtain the final image encoding we concatenate the encodings from all pyramid divisions, that is \(h'(X) = [h'(R_{\overline{3x1}}^{\text{upper}})^T, \ldots]^T\).

Similarly, we express the classifier vector as a concatenation of classifier vectors per division, namely \(w = [w_{\overline{3x1}}^{\text{upper}}, \ldots]^T\), which we obtain by max-margin optimization. The classification score for an image is \(y(X) = w^T h'(X) = \sum_{\text{div}} w_{\text{div}}^T h'(R_{\text{div}})\), where \(w_{\text{div}}\) is the subvector of the classifier \(w\) for the division \(\text{div}\). Hence, we observe that if we consider the trained weights per division \(w_{\text{div}}\) as separate classifiers, we can represent an image in terms of codemaps. Given that for normalization we require \(K'(X, X) = 1\), equivalent to \(\ell_2\)-normalization on \(h'(X)\), we have the pyramid codemap

\[
\Phi_{\text{pyr}} = \{g_i, W^T \psi(h(g_i)), \psi(h(g_i))^T \psi(h(g_j))\}, \tag{6.18}
\]

for the \(i, j = 1, \ldots, N\) lexes on our fixed lattice, where \(W\) is a matrix that contains columnwise the weight subvectors \(w_{\text{div}}\).

Codemaps in a feedforward architecture. In eq. (6.17) we consider local, nonlinear comparisons with respect to the lexes in the images. However, we observe that the global image similarity \(K'(X, Z) = h'(X)^T h(Z)\) is again an inner product. Therefore, we are able to employ another
Figure 44: Complexity of codemaps. (a) $\ell_2$-normalized Fisher codemaps (with 400 lexes per image) are up to 56x faster than computing Fisher vectors sequentially per region. (b) $\ell_2$-normalization is almost independent of the number of lexes. (c) Codemaps have a constant time complexity independent of the number of classifiers. For the CUB2011 dataset, where we have on average fewer (approximately 200) lexes and 550 segment hypotheses, codemaps return attribute representations in about 6 seconds per image, no matter how many attributes are considered. The numbers are computed on real data, using a Gaussian mixture model of 256 components. Figures (a) and (b) are generated on PASCAL VOC 2011 images, sampling every 2 pixels. Figure (c) is generated on CUB2011 images, sampling every 3 pixels.

round of nonlinearities on a global image level by replacing the inner product with a kernel function, namely

$$K_f (X, Z) = K''(K'(X, Z))$$

$$= \psi'(h'(X))^T \psi'(h'(Z))$$

$$= \psi'(\sum_g \psi(h(g_x)))^T \psi'(\sum_g \psi(h(g_z)))$$

(6.19)

Using once more feature mappings, the kernel function $K''$ can be rephrased as $K''(\cdot, \cdot) = \psi'(\cdot)^T \psi'(\cdot)$. The nonlinear mappings $\psi(\cdot)$ and $\psi'(\cdot)$ do not necessarily have to be the same. According to eq. (6.19), an image $X$ is computed first locally with $\psi(\cdot)$, then globally with $\psi'(\psi(\cdot))$, leading to a feedforward architecture for an image representation

$$h''(X) = \psi'(\sum_{g_x} \psi(h(g_x))).$$

(6.20)

Note that the image representation of eq. (6.20) is again computed on a single, fixed lattice, having in practice similar computational costs to pyramid codemaps.

As a side remark, we note the resemblance of the above feedforward architecture with the recently proposed deep Fisher networks [123]. In [123] nonlinearities are applied in iterative layers as well. However, as the purpose of [123] was to compare with deep, feature learning architectures, their proposed deep Fisher network is designed to extract per layer a very dense array of feedforward nonlinearities. These recursively employed nonlinearities can be safely trained when very large datasets [36] are available, which was also the application domain of deep Fisher networks. Feedforward pyramid codemaps, however, depend on a much sparser set of feedforward nonlinearities, hence the danger of overfitting is less eminent. As a result, feedforward pyramid codemaps are better suited, when such very large scale datasets are unavailable.

6.7 Measuring Efficiency

In this section we summarize the computational, memory and storage complexity properties of codemaps and visualize them in Fig. 44.
Table 17: _ℓ_2-normalized semantic segmentation for arbitrary regions. Mean average precision on the val set of the segmentation task in the PASCAL VOC 2011.

<table>
<thead>
<tr>
<th>Region normalization</th>
<th>Bag-of-words</th>
<th>Fisher codemaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>mAP</td>
<td>4.1</td>
<td>7.0</td>
</tr>
</tbody>
</table>

**Unnormalized vs normalized codemaps.** The _L_ classifiers are stored as a _D_ × _L_ matrix, each column containing a classifier composed of _D_ dimensions. Also, the local encodings are stored in a _D_ × _N_ matrix, one local encoding per column for each of the _N_ lexes in the lattice. Calculating the unnormalized codemaps from eq. (6.9) involves in the general case a matrix-matrix multiplication between the two matrices, that is a complexity of _O_(_L_ · _D_ · _N_).

For the normalized codemaps we need to compute the unnormalized codemaps first. Then, we compute the pair-wise norms of eq. (6.8), which will serve as our image normalization matrix. With the image _ℓ_2-normalization matrix we are able to reproduce the normalization coefficient for every region in the image composed of a subset of the _N_ lexes. To obtain the image _ℓ_2-normalization matrix we need to perform a matrix multiplication of the feature encodings matrix with itself, an operation that costs _O_(_D_ · _N_²). In practice and given Fisher encodings, for up to 1,000 lexes—a number of lexes typically employed in localization of objects [4, 21]—normalized and unnormalized codemaps have similar computational costs, see Fig. 44a. For a larger number of lexes the cost for obtaining the normalized codemap becomes slightly higher.

To store codemaps, we need _N_ · _L_ scalars for _N_ lexes and _L_ classifiers for the unnormalized score maps. Storing the image _ℓ_2-normalization matrix involves only 0.5 · (_N_² + _N_) values. Moreover, the image _ℓ_2-normalization matrix depends solely on the local feature encodings and is independent of the number of classifiers. Therefore, to obtain _ℓ_2-normalized codemap scores for as many classifiers as we want, we only need to compute the image _ℓ_2-normalization matrix just once.

**Region classification with codemaps.** After having calculated the normalized codemaps, we may proceed with the search, be it for classification, detection or segmentation. Given a region _R_ on the lattice we need at most _N_² + _L_ · _N_ summations of scalars to obtain normalized scores for _L_ classes, see eq. (6.9) and (6.8). In practice and given Fisher encodings, the time cost for evaluating thousands of regions is constant, see Fig. 44b.

**Object models.** For computing the normalized codemaps we have an _O_(_L_ · _D_ · _N_) and a _O_(_D_ · _N_²) operation. Comparing the two, we see that the difference lies in the relative comparison between the number of lexes _N_ per image and the number of classifiers _L_ we consider. Assuming images of similar sizes, we expect the number of superpixels _N_ to be roughly stable, usually in the order of hundreds per image. In practice and given Fisher encodings, evaluating scores for an increasing number of classifiers, namely _L_ → _N_, has little impact on the final computational cost, see Fig. 44c.

We conclude that with codemaps we have significant computational and memory efficiency advantages in computing classification scores over an arbitrary number of arbitrarily-shaped image regions. We will exploit these advantages in our experiments for tasks that involve by definition locality, _e.g._ semantic segmentation, as well as for tasks where locality is less eminent, yet still important, _e.g._ object classification and fine-grained classification by attributes.
6.8 EXPERIMENTS

6.8.1 Importance of $\ell_2$-normalization in segmentation

In the first experiment we investigate how important $\ell_2$ normalization is for semantic segmentation. In semantic segmentation several image regions need to be evaluated for whether they enclose objects and their type. We start from the PASCAL VOC segmentation dataset and follow the training protocol of CPMC-O2P [21]. We focus on the VOC 2011 train set for training and we report results on the val set. We also consider the unnormalized Fisher codemap version and unnormalized bag-of-words features using a visual codebook of size 4,000, similar to the ones used in [146]. While unnormalized Fisher vectors have been used for pixel-level segmentation [33], there has not been any work on segment-level classification with normalized Fisher vectors (except for the conference version of the current work [79]). The results are presented in Table 17.

We observe that $\ell_2$-normalized Fisher codemaps outperform the unnormalized ones by far. Fisher codemaps obtain a 26.9 mAP (mean average precision), where the unnormalized Fisher codemaps obtain only 7.0 mAP. While unnormalized Fisher codemaps outperform bag-of-words, the $\ell_2$-normalization is critical for the linear regression used to label segments, since we have to ensure that the overlap between each segment and itself is largest and equal to 1. We plot in Figure 44a how efficient it is to compute a Fisher codemap, under a varying number of lexes in the lattice. Calculating the normalized Fisher codemap is as efficient as the unnormalized version for up to 500 lexes. For semantic segmentation in particular, since 4–500 lexes per image usually suffice [5, 21, 145], calculating the $\ell_2$-normalized Fisher codemaps is practically as efficient as the unnormalized one, but much more accurate.

We conclude that $\ell_2$-normalization, which codemaps also facilitate, is important for semantic segmentation with linear classification models.

6.8.2 Codemaps for semantic segmentation

In the second experiment we quantify the value of $\ell_2$-normalized codemaps for semantic segmentation. Since the leading semantic segmentation methods use multiple features to capture several aspects of the object information (i.e. in [21] 3 features and in [4] 58 features are used), we embed Fisher codemaps into the multi-feature approach of CPMC-O2P [21]. CPMC-O2P combines three segmentation-tailored and color-enhanced features, and trains linear support vector regressors based on the overlaps between segments. We use the Fisher codemaps from Section 6.4, with dense sampling of basic intensity SIFT descriptors per pixel at multiple scales and a Gaussian mixture model of 128 components. Note that unlike [21] we do not consider any feature-specific optimizations for the purpose of semantic segmentation.
Table 18: State-of-the-art semantic segmentation. Following [21] we show semantic segmentation results for the PASCAL VOC 2011 and 2012 comp6 task. Normalized Fisher codemaps with CPMC-O2P score the highest semantic segmentation accuracy for 17 out of 21 object categories in VOC 2011. Similar observations hold for VOC 2012, where 8 out of 21 categories have a higher detection rate. For both datasets normalized Fisher codemaps with CPMC-O2P obtain the highest accuracies in the literature and even outperform feature learning methods [48] that make extensive use of external data [36].

<table>
<thead>
<tr>
<th>Method</th>
<th>mAP</th>
<th>Bgnd</th>
<th>Plane</th>
<th>Bike</th>
<th>Bird</th>
<th>Boat</th>
<th>Bottle</th>
<th>Bus</th>
<th>Car</th>
<th>Chair</th>
<th>Cow</th>
<th>Table</th>
<th>Dog</th>
<th>Horse</th>
<th>Mbike</th>
<th>Person</th>
<th>PPlant</th>
<th>Sheep</th>
<th>Sofa</th>
<th>Train</th>
<th>TV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VOC2011</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berkeley [4]</td>
<td>40.8</td>
<td>83.4</td>
<td>46.8</td>
<td>18.9</td>
<td>36.6</td>
<td>31.2</td>
<td>42.7</td>
<td>57.3</td>
<td>47.4</td>
<td>44.1</td>
<td>8.1</td>
<td>39.4</td>
<td>36.1</td>
<td>36.3</td>
<td>49.5</td>
<td>48.3</td>
<td>50.7</td>
<td>26.3</td>
<td>47.2</td>
<td>22.1</td>
<td>42.0</td>
</tr>
<tr>
<td>CPMC-O2P [21]</td>
<td>47.6</td>
<td>85.4</td>
<td>69.7</td>
<td>22.3</td>
<td>45.2</td>
<td><strong>44.4</strong></td>
<td>46.9</td>
<td>66.7</td>
<td>57.8</td>
<td>56.2</td>
<td>13.5</td>
<td>46.1</td>
<td>32.3</td>
<td>41.2</td>
<td>59.1</td>
<td>55.3</td>
<td>51.0</td>
<td><strong>36.2</strong></td>
<td>50.4</td>
<td>27.8</td>
<td>46.9</td>
</tr>
<tr>
<td>R-CNN [48]</td>
<td>47.9</td>
<td>84.2</td>
<td>66.9</td>
<td><strong>23.7</strong></td>
<td><strong>58.3</strong></td>
<td>37.4</td>
<td><strong>55.4</strong></td>
<td><strong>73.3</strong></td>
<td>58.7</td>
<td><strong>56.5</strong></td>
<td>9.7</td>
<td>45.5</td>
<td>29.5</td>
<td><strong>49.3</strong></td>
<td>40.1</td>
<td>57.8</td>
<td>53.9</td>
<td>33.8</td>
<td><strong>60.7</strong></td>
<td>22.7</td>
<td>47.1</td>
</tr>
<tr>
<td>This paper</td>
<td><strong>49.5</strong></td>
<td><strong>85.9</strong></td>
<td><strong>72.5</strong></td>
<td>23.6</td>
<td>45.3</td>
<td>40.2</td>
<td>54.6</td>
<td>70.6</td>
<td><strong>60.8</strong></td>
<td><strong>55.4</strong></td>
<td><strong>13.9</strong></td>
<td><strong>47.8</strong></td>
<td>31.5</td>
<td>44.3</td>
<td><strong>63.2</strong></td>
<td><strong>59.4</strong></td>
<td><strong>57.4</strong></td>
<td>33.7</td>
<td>52.3</td>
<td><strong>32.1</strong></td>
<td><strong>48.5</strong></td>
</tr>
<tr>
<td><strong>VOC2012</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPMC-O2P [21]</td>
<td>46.4</td>
<td>84.7</td>
<td>63.5</td>
<td>23.4</td>
<td>45.0</td>
<td>40.8</td>
<td>44.9</td>
<td>59.1</td>
<td>58.3</td>
<td><strong>57.1</strong></td>
<td>11.8</td>
<td>42.9</td>
<td>32.8</td>
<td><strong>45.2</strong></td>
<td>55.4</td>
<td>56.6</td>
<td>51.2</td>
<td>35.6</td>
<td>44.9</td>
<td>30.3</td>
<td>48.0</td>
</tr>
<tr>
<td>FGT_SEGM [22]</td>
<td>47.5</td>
<td>85.2</td>
<td>63.4</td>
<td><strong>27.3</strong></td>
<td><strong>56.1</strong></td>
<td>37.7</td>
<td>47.2</td>
<td>57.9</td>
<td>59.3</td>
<td>55.0</td>
<td>11.5</td>
<td><strong>50.8</strong></td>
<td>30.5</td>
<td>45.0</td>
<td>58.4</td>
<td>57.4</td>
<td>48.6</td>
<td>34.6</td>
<td><strong>53.3</strong></td>
<td>32.4</td>
<td>47.6</td>
</tr>
<tr>
<td>DivMBest [97]</td>
<td>48.1</td>
<td><strong>85.7</strong></td>
<td>62.7</td>
<td>25.6</td>
<td>46.9</td>
<td><strong>43.0</strong></td>
<td><strong>54.8</strong></td>
<td>58.4</td>
<td>58.6</td>
<td>55.6</td>
<td><strong>14.6</strong></td>
<td>47.5</td>
<td>31.2</td>
<td>44.7</td>
<td>51.0</td>
<td>60.9</td>
<td>53.5</td>
<td><strong>36.6</strong></td>
<td>50.9</td>
<td>30.1</td>
<td><strong>50.2</strong></td>
</tr>
<tr>
<td>This paper</td>
<td><strong>48.3</strong></td>
<td><strong>85.3</strong></td>
<td><strong>66.2</strong></td>
<td>24.4</td>
<td>47.5</td>
<td>37.2</td>
<td>52.4</td>
<td><strong>60.4</strong></td>
<td><strong>61.1</strong></td>
<td>56.5</td>
<td>12.8</td>
<td>44.5</td>
<td><strong>32.9</strong></td>
<td>44.8</td>
<td><strong>60.8</strong></td>
<td><strong>61.3</strong></td>
<td><strong>55.8</strong></td>
<td>33.2</td>
<td>49.8</td>
<td><strong>34.3</strong></td>
<td>47.9</td>
</tr>
</tbody>
</table>
Figure 45: **Semantic segmentation** with $\ell_2$-normalized Fisher codemaps on top of the CPMC-$O_2P$ [21]. Note the detected persons riding a horse in the first and the last row, or the person standing next to a bus in the third row. For some interesting failure detections are also present, see the detected boat next to the car in the third row or the array of facades detected as a train in the first row.

We report results on two leading semantic segmentation benchmarks where most of the state-of-the-art approaches report their accuracies, that is the PASCAL VOC 2011 and 2012 segmentation datasets. Similar to [21], we use the additional training images from [4, 52] and report the results on comp6 of the VOC 2011 and VOC 2012 challenges. Since both the [21] features and the $\ell_2$-normalized Fisher codemaps use a linear regressor, we rely on late fusion with linear weights learned on the val set to combine them.

As a preliminary, we first compare the individual features of CPMC-$O_2P$, i.e. eSIFT, eMSIFT and eLBP, obtain 28.4, 31.0 and 21.2 mAP on the VOC 2011 val set respectively (data not shown), where Fisher codemaps score 26.9 without any optimization for semantic segmentation. We also compare against the best reported methods from the literature [4, 22, 48, 97]. Table 18 displays the results for the two datasets.

For VOC 2011 we observe that CPMC-$O_2P$ with Fisher codemaps arrives at 49.5% mAP, the highest reported result to the best of our knowledge. Regarding individual accuracies, the proposed method outperforms competitors for 12 out of 21 categories, while improving the plain vanilla CPMC-$O_2P$ for 17. Note that CPMC-$O_2P$ with Fisher codemaps performs better than the deep convolutional neural network by Girshick et al. [48], although the latter makes extensive use of additional data from ImageNet [36]. For VOC 2012 Fisher codemaps improves semantic segmentation accuracy for 8 out of the 21 object categories (including “background”) having again the best overall accuracy.

In Figure 45 we visualize examples of codemap segmentations. We observe that the returned segmentations are accurate even for complicated scenes, where multiple objects appear. Naturally, failure detections also appear, such as the array of facades recognized as a train or the part of the car also detected as a boat. To avoid such mistakes use of inter-object relationships would probably be of interest.

We conclude that a combination of CPMC-$O_2P$ with our $\ell_2$-normalized Fisher codemaps allows for accurate semantic segmentation.
Table 19: Fine-grained classification by attributes. Attribute codemaps improve classification by attributes substantially on the challenging Caltech-UCSD Birds-200-2011 dataset [148].

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Label Embedding [1]</td>
<td>20.5</td>
</tr>
<tr>
<td>Hierarchical Label Embedding [1]</td>
<td>20.1</td>
</tr>
<tr>
<td>WSABIE [151]</td>
<td>20.1</td>
</tr>
<tr>
<td>Full-image attributes using Opponent SIFT</td>
<td>20.5</td>
</tr>
<tr>
<td>Attribute codemaps</td>
<td><strong>28.1</strong></td>
</tr>
</tbody>
</table>

Figure 46: Results from using attribute codemaps. Segmentations and the highest scoring attributes for various bird images.

6.8.3 Attribute codemaps

In the third experiment we examine the potential of codemaps in the case where many classifiers need to be evaluated. To this end we focus on classification by attributes, where each attribute serves as a trained classifier. As codemaps allow for evaluating hundreds of classifiers for hundreds of regions in an image, we are able to examine whether refining the spatial support in an image benefits attribute representations.

We use the popular Caltech-UCSD Birds-2011 (CUB2011) dataset [148] that contains 200 hard to separate bird species. We opt for this dataset mainly because it enjoys a detailed annotation of 312 human-recognized attributes. These attributes will serve as our classifiers for computing the
codemaps. Each class in the CUB2011 dataset contains 30 training images and 30 testing images. We use the provided training/test split provided by the authors of [148].

The local attributes are learnt as described in Section 6.5, using the ground truth segmentations. During testing, segment hypotheses are generated with the CPMC algorithm [23]. To minimize the number of false negatives, and since codemaps can handle efficiently thousands of possible segmentations, we retain all the possible segmentations without considering any pre-filtering strategy. We, therefore, process on average 550 segmentation proposals per image, for which codemaps return the attribute representations in about 6 seconds. To maximize overlap with the object we select the union of the top $M$ detections, where we found $M = 12$ to be a reliable setting. For each image localized attribute codemaps and full image attributes are extracted using Fisher vectors computed on the basis of Opponent SIFT features [63].

We compare with state-of-the-art algorithms [1, 151]. Since we use slightly different features we repeat their experiment with Opponent SIFT. We summarize the comparisons with the state-of-the-art in Table 19.

As a preliminary, we note that when repeating the experiment of [1] with Opponent SIFT we obtain similar results. By exploiting image regions, attribute codemaps are 8% more accurate in the comparisons, outperforming ALE with Opponent SIFT for 150 out of the 200 classes (data not shown). Note that in this work we focus on classification by attributes. Using low level encodings with codemaps, as we do on this dataset in [44], would increase our accuracy further, but would also make the comparison with the cited works unfair.

In Fig. 46 we present some of our fine-grained bird segment detections with the highest scoring attributes. We observe that a satisfactory spatial support is discovered, which allows for more accurate attributes. What is more, obtaining these representations is efficient. As shown in Fig. 44c the attribute codemaps evaluate hundreds of attributes for the 550 segmented hypotheses in practically constant time.

We conclude that attribute codemaps are efficient and accurate. Moreover, the locality-aware attribute codemaps allow for more accurate attribute representations.

6.8.4 Image classification with codemaps

In the forth experiment we quantify the value of embedding codemaps in a feedforward architecture for object classification. We use the PASCAL VOC 2007 classification dataset [39] for both bag-of-words and Fisher vectors [104]. We sample dense SIFT descriptors every two pixels at multiple scales. We use a visual codebook of size 4,000 with hard assignment for the bag-of-words and a 256 component Gaussian mixture model for the Fisher vectors. We employ a $1 \times 1$, $2 \times 2$ and $3 \times 1$ pyramid. Since power normalization has shown to work particularly well for Fisher vectors [104], we implement Fisher pyramid codemaps with local Hellinger kernel pooling, that is $\psi(\cdot) = \psi'(\cdot) = \sqrt{\cdot}$. For bag-of-words the $\chi^2$ and histogram intersection kernels are the top performers [82, 162] and the bag-of-words pyramid codemaps are implemented with the respective explicit feature maps [144]. The results are presented in Table 20.
Table 20: **Codemaps in a feedforward architecture.** By applying multiple layers of nonlinear kernel poolings, codemaps in a feedforward architecture improve the classification accuracy consistently for all categories in PASCAL VOC 2007.

<table>
<thead>
<tr>
<th></th>
<th>mAP</th>
<th>Plane</th>
<th>Bike</th>
<th>Bird</th>
<th>Boat</th>
<th>Bottle</th>
<th>Bus</th>
<th>Car</th>
<th>Cat</th>
<th>Chair</th>
<th>Cow</th>
<th>Table</th>
<th>Dog</th>
<th>Horse</th>
<th>Mbike</th>
<th>Person</th>
<th>PPlant</th>
<th>Sheep</th>
<th>Sofa</th>
<th>Train</th>
<th>TV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher vectors [104]</td>
<td>61.7</td>
<td>83.7</td>
<td>69.5</td>
<td>52.9</td>
<td>74.4</td>
<td>28.8</td>
<td>70.3</td>
<td>82.4</td>
<td>59.4</td>
<td>53.4</td>
<td>48.4</td>
<td>58.7</td>
<td>43.5</td>
<td>81.3</td>
<td>70.2</td>
<td>85.9</td>
<td>30.0</td>
<td>45.7</td>
<td>57.5</td>
<td>82.9</td>
<td>54.4</td>
</tr>
<tr>
<td>Feedforward codemaps</td>
<td>63.6</td>
<td>84.8</td>
<td>70.6</td>
<td>56.2</td>
<td>76.8</td>
<td>30.9</td>
<td>72.3</td>
<td>83.4</td>
<td>63.7</td>
<td>55.1</td>
<td>52.5</td>
<td>61.2</td>
<td>46.1</td>
<td>82.6</td>
<td>71.2</td>
<td>86.9</td>
<td>31.0</td>
<td>48.8</td>
<td>58.2</td>
<td>83.7</td>
<td>56.6</td>
</tr>
<tr>
<td>BoW [124]</td>
<td>54.8</td>
<td>74.9</td>
<td>56.6</td>
<td>42.1</td>
<td>67.6</td>
<td>20.7</td>
<td>62.6</td>
<td>76.3</td>
<td>56.9</td>
<td>51.2</td>
<td>41.2</td>
<td>49.6</td>
<td>39.2</td>
<td>77.4</td>
<td>64.7</td>
<td>83.2</td>
<td>22.9</td>
<td>35.7</td>
<td>49.1</td>
<td>74.8</td>
<td>49.0</td>
</tr>
<tr>
<td>Feedforward codemaps</td>
<td>56.1</td>
<td>76.5</td>
<td>59.1</td>
<td>42.6</td>
<td>68.8</td>
<td>23.0</td>
<td>62.9</td>
<td>78.2</td>
<td>58.1</td>
<td>52.8</td>
<td>44.0</td>
<td>49.5</td>
<td>42.1</td>
<td>78.1</td>
<td>65.3</td>
<td>84.9</td>
<td>25.1</td>
<td>35.8</td>
<td>49.8</td>
<td>76.3</td>
<td>50.0</td>
</tr>
</tbody>
</table>
Figure 47: Classification examples using codemaps in a feedforward architecture. The visualized images are placed among the highest true and false positive ranks for the particular classes.

As a preliminary, when using only pyramid codemaps without embedding them in a feedforward architecture, we perform on par with the spatial pyramid kernel (data not shown). When embedding codemaps in a feedforward architecture, we observe in Table 20 that the final accuracy improves in a consistent manner for all classes, both for bag-of-words and for Fisher vectors. It is interesting that for Fisher vectors, the feedforward pyramid codemap seems to benefit animal categories in particular, resulting in an absolute improvement of 1.5-4.5%. Note that the added nonlinearities do not involve any additional computations apart from the feature encoding transformations, e.g. applying the square root for the Hellinger kernel. As within a feedforward architecture a codemap parses the image in just a single pass, the improved accuracy does not cause and significant, extra computational cost.

In Fig. 47 we illustrate some of the highest ranked true and false positives. The object class true positive instances appear almost prototypical. Moreover, the false positives reveal interesting phenomena. For one, often the ground truth annotations are incomplete, e.g. the “negative chair” image does contain chairs. Also, the classification is often incorrect, although the evidence of the predicted class actually appears in the picture. For example, the “negative sheep” is indeed a white animal on a grass field, or the “negative train” is indeed an elongated object with an array of windows on the side. In such cases one probably needs to look for the finer details, like
the hook-shaped beak of the bird in the “negative sheep”, or the pointy bow of the boat in the “negative train”.

We conclude that by introducing recursive nonlinearities, codemaps in a feedforward pyramid benefit image classification without inducing any extra computational cost.

6.9 conclusions

In this chapter, we propose codemaps that segment and classify objects locally. With codemaps the encoding, pooling and classification steps of object classification are reordered. The reorder takes place after exploiting the homomorphic properties of the sum operator and grouping of local neighborhood scores over lattice elements. By doing so, codemaps avoid redundancies over the lattice elements.

As our first contribution, we present codemaps that also include efficient $\ell_2$-normalization for arbitrarily shaped image regions. Depending on the number of regions in an image, the normalized codemaps are up to 56x faster than traditional Fisher vectors, calculated for all regions independently (Fig. 44). The $\ell_2$-normalized codemaps enable the highest accuracy in the leading PASCAL VOC 2011 and 2012 benchmarks in semantic segmentation (Table 18).

Having shown the benefits of $\ell_2$-normalized codemaps in semantic segmentation, we take advantage of the fact that with codemaps we can compute on-the-fly hundreds of $\ell_2$-normalized attribute scores for hundreds of regions. Hence, as a second contribution we show that codemaps incorporate locality in attribute learning and attribute representations, where each attribute stands for a local classifier. We show that by incorporating locality with codemaps we improve the accuracy of attribute representations substantially (Table 19).

Our third contribution is the embedding of additive nonlinearities in the codemap decomposition by local kernel pooling. When using the same lattice across images, codemaps incorporate nonlinear feature mappings, like hellinger, $\chi^2$ or histogram intersection kernels, in multiple layers of the classification chain. Hence, with codemaps we arrive at a feedforward architecture that improves classification accuracy at a virtually zero cost (Table 20).

Apart from segmentation, classification with low-level encodings or fine-grained classification by attributes, we anticipate that other computer vision challenges, which involve repetitive computations over overlapping image regions, may profit from codemaps as well.