Let the Kernel Figure it Out; Principled Learning of Pre-processing for Kernel Classifiers

Peter Vincent Gehler and Sebastian Nowozin
Max Planck Institute for Biological Cybernetics, Department of Empirical Inference
Tübingen, Germany
{pgehler,nowozin}@tuebingen.mpg.de

Abstract

Most modern computer vision systems for high-level tasks, such as image classification, object recognition and segmentation, are based on learning algorithms that are able to separate discriminative information from noise. In practice, however, the typical system consists of a long pipeline *of pre-processing steps, such as extraction of different kinds of features, various kinds of normalizations, feature selection, and quantization into aggregated representations such as histograms. Along this pipeline, there are many parameters to set and choices to make, and their effect on the overall system performance is a-priori unclear.*

In this work, we shorten the pipeline in a principled way. We move pre-processing steps into the learning system by means of kernel parameters, letting the learning algorithm decide upon suitable parameter values. Learning to optimize the pre-processing choices becomes learning the kernel parameters. We realize this paradigm by extending the recent Multiple Kernel Learning formulation from the finite case of having a fixed number of kernels which can be combined to the general infinite case where each possible parameter setting induces an associated kernel.

We evaluate the new paradigm extensively on image classification and object classification tasks. We show that it is possible to learn optimal discriminative codebooks and optimal spatial pyramid schemes, consistently outperforming all previous state-of-the-art approaches.

1. Introduction

Kernel Learning Algorithms have considerably influenced the field of Machine Learning and Computer Vision over the last decade. One of their main advantages is the possibility to formalize a notion of similarity between complicated objects like images through the construction of a kernel function. Therefore the design of the kernel function plays an important part for all kernel learning machines. In this paper we will show that choosing a kernel is not only a "burden" but also opens up the possibility to turn choices for pre-processing steps into kernel parameters which in turn can be optimized over.

The typical procedure for building a classification function from data can be split into two parts, the pre-processing pipeline and the search for the classification function itself. The pre-processing pipeline may include many different steps and each one influences the overall classification function. For example for visual object classification typical pre-processing steps are the choice of type and number of features to be extracted from each image, its transformation to aggregated representations such as histograms and possible normalizations of the latter. The main problem is that the effect of a single choice somewhere in pipeline on the final classification function is a-priori unclear. Most often the only way to quantify its effect is to compare all possible choices. This is sub-optimal; in an ideal setting one would like to place a suitable prior over parameter choices and allowing the learning algorithm to freely optimize the quality of the classification function over *all* parameter settings.

To get closer to this goal, we propose to apply the following simple paradigm when building classification systems. Instead of choosing parameters of pre-processing steps beforehand we turn them into parameters of the kernel function and thus make them available to the kernel learning algorithm. In particular we propose to not only use one parameterizable kernel but general *kernel classes* from which the best kernel is chosen. This is implemented using the well developed technique of Multiple Kernel Learning (MKL) [2, 16, 15]. Therefore we will not only optimize the SVM weights, but also optimize over the kernel itself. With our paradigm pre-processing choices become kernel parameters and optimizing over them corresponds to optimizing the pre-processing parameters.

We will report on several contributions, both algorithmic and experimental. Since the classical MKL technique involves only a finite number of possible kernels to combine, we will generalize it to the infinite case. This allows to optimize over *continuously* parametrized kernel classes. Our second contribution is a new efficient formulation of the ℓ_2-MKL problem which does not restrict the weighting of the

978-1-4244-3991-1/09/$25.00 ©2009 IEEE

participating kernels to be sparse, similar in spirit to [9]. In the experiments we will demonstrate how to put our approach into practice. The proposed learning-based system is shown to consistently outperform previous approaches.

We present experiments for three different scenarios, i) learning the individual length scales for given data, ii) learning the optimal codebook for a bag-of-words representation and iii) learning the spatial configuration for a spatial kernel function.

The paper is organized as follows. In the next section 2 we will revisit the MKL formulation and extend it to both the infinite case as well as to the non-sparse case. We then present three different experimental sections 3-5. We discuss more possible applications in Section 6 and conclude with a discussion in Section 7.

2. Multiple and Infinite Kernel Learning

Multiple kernel learning is a recent development which follows the route of automatically selecting the kernel during the optimization phase of the algorithm [2, 16, 15]. Instead of using only one kernel in a support vector machine (SVM) a number of so-called proposal kernels $k(\cdot,\cdot;\theta), \theta \in \Theta$ are given and linearly combined during training. We wrote $k(\cdot,\cdot;\theta)$ to make explicit the dependency of the kernel on its parameters θ. Given some training data $(x_1, y_1), \ldots, (x_n, y_n)$ and a set of admissible kernel parameters Θ the optimal kernel found by the MKL algorithm is of the following form $k^*(\cdot, x) = \sum_{\theta\in\Theta} d_\theta k(\cdot, x; \theta)$. The final classification function is a SVM classifier using the combined kernel as follows

$$f(x) = \sum_{i=1}^{n} \alpha_i \sum_{\theta\in\Theta} d_\theta k(x, x_i; \theta) + b. \tag{1}$$

A MKL algorithm can be understood in the following way. With fixed weights d_θ the problem reduces to the standard SVM, that is finding the parameters α. The MKL algorithm alternates between the outer loop: update the kernel mixture weights d_θ for fixed α; and the inner loop: update the SVM weights α for fixed d_θ.

This technique was successfully applied to the task of visual object classification [18], where different kernels correspond to different features, *e.g*. one for color and one for shape. The MKL algorithm is then used to figure out a kernel combination which is discriminative for classifying visual categories.

2.1. Multiple Kernel Learning - the infinite ℓ_1 case

In this section we generalize the MKL setting to the infinite case. This enables us to linearly combine kernels from an infinite set of proposal kernels. We start by adopting the primal MKL formulation of [15] to write the following convex optimization problem

$$\begin{aligned} \min_{d,v,\xi,b} \quad & \frac{1}{2}\sum_{\theta\in\Theta}\frac{1}{d_\theta}\|v_\theta\|^2 + C\sum_{i=1}^{n}\xi_i && (2)\\ \text{sb.t.} \quad & y_i\Big(\sum_{\theta\in\Theta}\langle v_\theta, \phi_\theta(x_i)\rangle + b\Big) \geq 1-\xi_i, i=1,\ldots,n \\ & \xi_i \geq 0, \quad i=1,\ldots,n \\ & \sum_{\theta\in\Theta} d_\theta = 1, \quad d_\theta \geq 0. && (3) \end{aligned}$$

In order to solve this problem we dualize it, turning constraints into variables and vice verse [3]. To this end we write the Lagrangian, equate its derivative w.r.t. primal variables to zero and obtain the equivalent convex formulation[6].

$$\begin{aligned} \text{(IKL-Dual)} \quad \max_{\alpha,\lambda} \quad & \sum_{i=1}^{N}\alpha_i - \lambda && (4)\\ \text{sb.t.} \quad & \alpha \in \mathbb{R}^N,\ \lambda\in\mathbb{R} \\ & 0 \leq \alpha_i \leq C, \quad i=1,\ldots,N \\ & T(\theta;\alpha) \leq \lambda, \quad \forall\theta\in\Theta, \end{aligned}$$

where we defined

$$T(\theta;\alpha) = \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j k(x_i, x_j; \theta),$$

and where λ is the Lagrange multiplier corresponding to the ℓ_1 equality constraint $\sum_{\theta\in\Theta} d_\theta = 1$. Each possible kernel parameter θ now defines a constraint $T(\theta;\alpha) \leq \lambda$. We want to point out the following important observation. The dual problem can also be applied to the case of *infinitely* many kernels, which corresponds to a infinite set Θ [1, 6]. Then (4) becomes a semi-infinite program [7]. Although there are an infinite number of proposal kernels one can prove that at the optimal solution there are only a finite number non-zero weights d_θ. Therefore it suffices to search for these [7].

The form of (4) suggests a delayed constraint generation algorithm to solve it. We start with a small finite kernel parameter set $\Theta_0 \subset \Theta$ and call a standard MKL solver to optimize over this set. This produces the variables α and also returns the Lagrange multiplier λ. Now we solve the *subproblem*

$$\theta^* = \arg\max_{\theta\in\Theta} T(\theta;\alpha). \tag{5}$$

If $T(\theta^*;\alpha) > \lambda$ we include θ^* in the kernel parameter set, *i.e*. set $\Theta_{t+1} = \Theta_t \cup \{\theta^*\}$ and reiterate. The algorithm is summarized below.

At each iteration we have to search for a maximum of $T(\cdot;\alpha)$ which usually is a non-concave and possibly complicated function. Finding the guaranteed global optimum is therefore intractable for high dimensional θ. However if we find *some* kernel parameter θ for which $T(\theta;\alpha) > \lambda$, it is guaranteed that the objective function (4) increases. Additionally, we could stop at any iteration and have a valid

Algorithm 1 Infinite Kernel Learning (IKL)

Input: Regularization constant C, Kernel parameter set Θ, Training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
Output: Parameters α, b, d_θ

Select any $\theta_v \in \Theta$ and set $\Theta_0 = \{\theta_v\}$
$t \leftarrow 0$
loop
  $(\alpha, b, d_\theta, \lambda) \leftarrow$ MKL solution with Θ_t {Solve restricted master problem}
  $\theta_v \leftarrow \arg\max_{\theta \in \Theta} T(\theta; \alpha)$ {Solve subproblem}
  if $T(\theta_v; \alpha) \leq \lambda$ **then**
    break
  end if
  $\Theta_{t+1} = \Theta_t \cup \{\theta_v\}$
  $t \leftarrow t + 1$
end loop

classification function of form (1), in other words the problem is always primal feasible in (2). The IKL solution is guaranteed to have an equal-or-better objective in (2) than any finite MKL solution with an a-priori chosen kernel set Θ_f, because we can set $\Theta_0 = \Theta_f$ and improve from there.

2.2. Multiple Kernel Learning - the ℓ_2 case

The primal ℓ_1-MKL (2) makes use of a simplex constraint (3) on the weights in order to enforce a sparse solution with only few non-zero d_θ. We change this formulation by replacing the ℓ_1 constraint with a ℓ_2 norm on the weights. This is similar to [9], where it was found that a hard ℓ_2-norm constraint outperformed the sparse ℓ_1-MKL for a bioinformatics task. Our variant is more efficient and implemented by the following convex optimization problem

$$\begin{aligned} \min_{d,v,\xi,b} \quad & \frac{1}{2} \sum_{\theta \in \Theta} \frac{1}{d_\theta} \|v_\theta\|^2 + C \sum_{i=1}^{n} \xi_i + \|d\|_2^2 \qquad (6) \\ \text{sb.t.} \quad & y_i \Big(\sum_{\theta \in \Theta} \langle v_\theta, \phi_\theta(x_i) \rangle + b \Big) \geq 1 - \xi_i, i = 1, \ldots, n \\ & \xi_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

This modification has the following effect. Instead of a sparse solution at the optimum *all* weights d_θ will be non-zero. Therefore this formulation cannot be easily generalized to the infinite case. The algorithms for solving the ℓ_1-MKL and ℓ_2-MKL are very similar and one only needs to make minor modifications. Again we use the SimpleMKL algorithm presented in [15] to solve the problem.

The difference between the ℓ_1-MKL and the ℓ_2-MKL formulation can also be understood as two different priors on the set of the proposal kernels. If only a few of them should be selected the ℓ_1 method should be chosen. If on the other hand all of the possible choices of kernel parameters are reasonable one can use the ℓ_2 formulation to include them all but let the algorithm adjust the ratios between them.

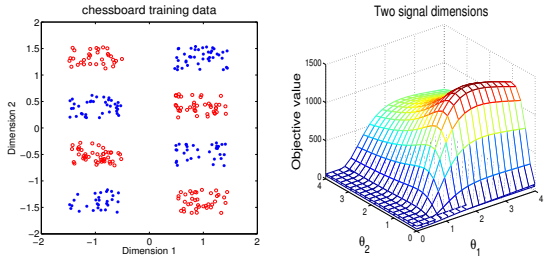


Figure 1. Chessboard toy example. Left: distribution of training data in the first two dimensions, right: $T(\theta)$ as a function of different kernel parameters θ. The highest value corresponds to the best kernel parameter that can be added at this iteration, *i.e.* the one which yields the largest decrease in the SVM objective.

3. Learning the Scaling

We will now present an example which demonstrates both how our paradigm can be applied to select among preprocessing parameters and how the subproblem of the IKL algorithm can be solved in practice.

A typical feature pre-processing step is to remove the mean of the data and scale each dimension to unit variance. It remains unclear if this pre-scaling is helping or hurting the problem. Therefore we aim to *learn* the variance in each single dimension individually. Let $x, x' \in \mathbb{R}^D$ be two data points with $[x]_d$ denoting the d'th component of x. We offer the following set of proposal kernels to the IKL algorithm

$$k(x, x'; \{\theta_1, \ldots, \theta_d\}) = \exp(-\sum_{d=1}^{D} \theta_d^2([x]_d - [x']_d)^2),$$

diagonally scaled Gaussian kernels with individual scaling in each dimension. This set consists of infinitely many kernels, continuously parametrized over $\theta \in \mathbb{R}_+^d$. The final kernel found by IKL is a finite linear combination of these proposal kernels.

We sample a total of 300 training points from the chessboard pattern shown in Figure 1. We add 18 noise dimensions to form a 20 dimensional feature vector. The best possible kernel would be one setting $\theta_d^2 = 0$ for noise dimensions $d = 3, \ldots, 20$ and choosing θ_1 and θ_2 as the interclass distance between the points in the two dimensions (2.0 in the first, 1.0 in the second dimension).

The IKL algorithm is started using some random initial kernel which is not the "correct" one. The first iteration with this kernel yields the current SVM parameters α. Now one has to solve the subproblem to find a new kernel to include in the problem. A 2D slice of the function $T(\cdot; \alpha)$ is shown in the right plot of Figure 1. The parameter θ is a 20-dimensional vector and for visualization we show only the first two dimensions, varying θ_1, θ_2 while keeping the others fixed $\theta_d = 0, d = 3, \ldots, 20$. The hyperplane associated with the Lagrange multiplier $\lambda \approx 62$ is also shown. All parameters with function values above this hyperplane correspond to kernels whose inclusion will decrease the ob-

jective function. The plot of Figure1(b) includes the "true" set of parameters (the ones described above) for the problem and indeed the objective takes a maximum at this point. One should be reminded that T is a 20 dimensional non-concave problem and thus one cannot easily solve for the global optimum. We implement the search for a local optimum of $T(\cdot;\alpha)$ using gradient ascent initialized at different starting points. This worked very well in practice.

In the course of the algorithm, IKL identifies the signal dimensions automatically and selects only a single kernel which achieves a perfect accuracy on a held out test set. To achieve the same result with a SVM or MKL learner one would have to cross-validate over different scaling factors for each dimension (2^{20} for only two choices per dimension) or guess the correct scaling for the proposal kernels. In contrast, the IKL algorithm can use the very general kernel class and one can use T as a handle to implement a search algorithm rather than to rely on guessing the correct parameters beforehand as it is necessary for MKL.

4. Learning a Suitable Codebook

Using a "bag-of-visual-words" representation has become standard practice for many computer vision tasks, see *e.g*. [14]. The benefit of using a histogram representation is that it converts sets of arbitrary many elements to a fixed length feature descriptor. The usual procedure to obtain this representation is as follows. Descriptors are generated at some points in the image, *e.g*. SIFT descriptors. Now a codebook of size K with elements of the same feature space is created. The feature points are vector-quantized using the codebook to yield a histogram representation. This non-linear pre-processing step involves different choices to be made: Which is the best codebook for a given task? How should quantization be performed? Should the resulting representation be normalized and if so, how? Many recent works have addressed these problems and proposed different solutions, see *e.g*. [14, 11, 17].

The common part of all approaches is the use of a classification function (most often a SVM) *after* application of the pre-processing pipeline. Our approach is to start with this function directly and design kernels which correspond to the different choices one could have made beforehand. It is hard to a-priori distinguish between discriminative and not-so-discriminative codebooks and therefore explicitly avoid to do so. We regard the codebooks themselves as free parameters *which are available* to the learning algorithm to optimize.

We design proposal kernels in the following way. Let x, x' be bags of D-dimensional image features, *e.g*. 128 dimensional SIFT features. They are quantized to a histogram representation using a codebook $\mathcal{C}$ with K codewords by means of a nearest-neighbor quantization function $q_{\mathcal{C}}(x) : \mathbb{R}^D \to \mathbb{N}_+^K$ mapping features into a histogram bins using $\mathcal{C}$. We regard each possibly choice of a codebook as a parameters of the kernel. The proposal kernels are χ^2-kernels of the following form

$$k(x, x'; \{\gamma, \mathcal{C}\}) = \exp\left(-\gamma^2 \chi^2(q_{\mathcal{C}}(x), q_{\mathcal{C}}(x'))\right),$$

where $\chi^2 : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ denotes the χ^2 distance function. Although we could also optimize over γ, for simplicity we always fix γ^2 to be the reciprocal of the median of all pairwise training distances. The only parameter left to the problem is thus the codebook $\mathcal{C}$.

Learning the ℓ_1-MKL involves the maximization of $T(\{\mathcal{C}\};\alpha)$ over all possible codebooks. For simplicity, we optimize $T(\{\mathcal{C}\};\alpha)$ approximately by evaluating it at many codebooks, randomly sub-sampled from the training set.

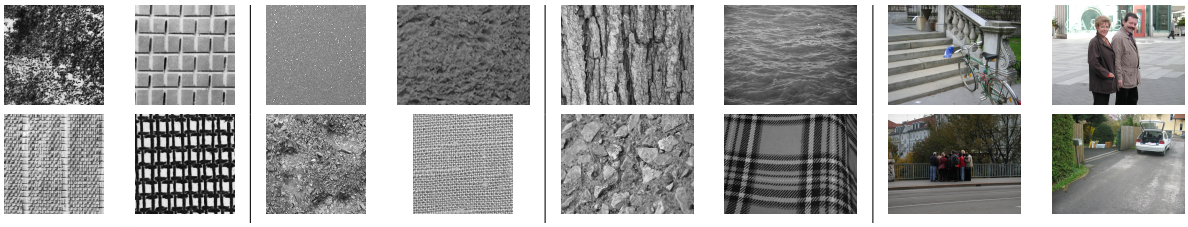
A linear combination of kernels which correspond to different codebooks can also be understood as a soft clustering step. Each feature vector is not only assigned to one codebook vector but for each kernel to a different one. Therefore its position in the feature space is described more precisely.

4.1. Experimental Setup

We test the codebook learning approach on four datasets Brodatz, KTH-TIPS, UIUCTex and Graz-02, some of which were also used in [14] to investigate the influence of codebook generation and sampling methods. Some sample images from the datasets are shown in Figure 2. Graz-02 is an object classification dataset with three classes (bikes, persons, cars) consisting of 1096 images (we removed the background class), approximately one third for each category. KTH-TIPS contains 810 images from ten categories: aluminum foil, brown bread, corduroy, cotton, cracker, linen, orange peel, sandpaper, sponge and styrofoam. The Brodatz dataset consists of 112 texture images, one per class. These images were subdivided into thirds horizontally and vertically to produce 9 images per class. The last dataset UIUCTex consists of 40 images per classes of 25 textures distorted by significant viewpoint changes and some non-rigid deformations.

We extract PCA-SIFT [8] feature descriptors from the images using a dense grid of points. The radius of the patches we use for feature extraction is varied over 4 different scales (8 to 50 pixels for the UIUCTex images and 4 to 16 pixels for the others). This resulted in the following numbers of features for the datasets. Brodatz: 1764 features, KTH-TIPS 1600, Graz-02 3072 and UIUCTex 1976 features. All features are 36 dimensional and all images within one dataset have the same number of features.

Each dataset is split 25 times into half for training and half for testing. The first five splits are used for model selection: we choose the C from the range $10^{-2}, 10^{-1}, \ldots, 10^{3}$ which yields best performance on the first five splits. This choice is applied to all 25 splits of the data to obtain the



Brodatz KTH TIPS UIUCTex TU-Graz

Figure 2. Four example images from each of the datasets used for Codebook learning experiments. All datasets are multiclass.

reported result. We resolve multiclass decisions by using one-versus-rest classifiers.

4.2. Results

We compare four different methods and different sizes K of the codebooks. The baseline method uses a single codebook only. For this method we found it to be irrelevant whether the codebook is created by random sampling or k-means clustering of training vectors, a finding consistent with the result of [14]. The other methods all use multiple codebooks. We use i) uniform weights d_θ with no learning, ii) ℓ_1-MKL learning, and iii) ℓ_2-MKL learning. Results are shown in Figure 3 and Table 1. We summarize:

1. Use more codebooks and averaging dramatically improves the performance (dashed versus solid lines).
2. Using ℓ_2-MKL learning does not significantly improve the result over a simple averaging step (circles).
3. The ℓ_1-MKL yields competitive results only for the datasets UIUCTex and TU-Graz (stars). The found codebook is much smaller which is of advantage at test time. Picking the same number of codebooks at random is considerably worse (stars versus dashed lines), thus there really is something learned by the model.

As a last remark we want to note that even using a small codebook with only 10 elements yields competitive performance on KTH-TIPS and UIUCTex.[1]

The hard quantization step q_C is by no means the only applicable choice. One could think of using a soft clustering of the features from the beginning. For example in [17] different types of soft clustering with different normalization based on uncertainty or plausibility have been proposed and were compared to each other. In our framework we do not pick one soft clustering and normalization beforehand but to simply offer all of them to the learning algorithm.

5. Learning the Optimal Spatial Layout

Kernels making use of spatial information of the image are an instructive example of how special structure of the data can be used for kernel design. Instead of comparing two images in its entirety only those features which fall into a certain subwindow are compared, *e.g.* all those of the upper half. This idea lead to the design of the *spatial pyramid kernel* [12].

Let us shortly review the main idea behind the spatial pyramid kernel and then discuss how our paradigm can be applied to spatial kernels. We subdivide the image into different levels, enumerated by $l = 0, 1, \ldots$. The level l corresponds to a $2^l \times 2^l$ equally spaced grid on the image plane, and thus consists of 4^l non overlapping cells of equal size. In this construction higher levels of the pyramid correspond to a finer griding of the image. We build a histogram for each cell in a level l individually and stack them to obtain the final representation ($4^l * K$ for level l). The spatial pyramid kernel compares two images by measuring the similarity between the concatenated histograms for each level individually and linearly combining them as

$$k(x, x') = \sum_{l=1}^{L} d_l k_l(x, x'),$$

with k_l being the kernel comparing the images only through level l. The authors of [12] propose to set the weighting coefficients as $d_l = 2^{-(L-l)}$ with L being the maximum level, usually three. Although good results are reported using the spatial pyramid kernel the question arises whether or not the pyramidal representation of the image is the best for a given task and if the intuition behind the choice of d_k is justified. In the following we will answer this question.

We want to let the objective function decide on the best spatial layout and thus design the following class of spatial kernels. By B we denote a box in the image plane[2] and by $d_B(x, x')$ the χ^2-distance comparing two images represented by a collection of local features by taking into account only features which fall into the box B. Our proposal kernels are

$$k(x, x'; \{\gamma, B\}) = \exp\left(-\gamma^2 d_B(x, x')\right). \qquad (7)$$

The parameter γ is again chosen as described in Section 4. During ℓ_1-MKL learning we have to maximize the function $T(B; \alpha)$ over all possible boxes. This could be done by the efficient subwindow search method [10], but for simplicity

[1] We checked that there is no further performance gain in averaging over even more codebooks, the results level out at 250 codebooks.

[2] The box is defined in relative coordinates, so the actual size of the image does not matter.

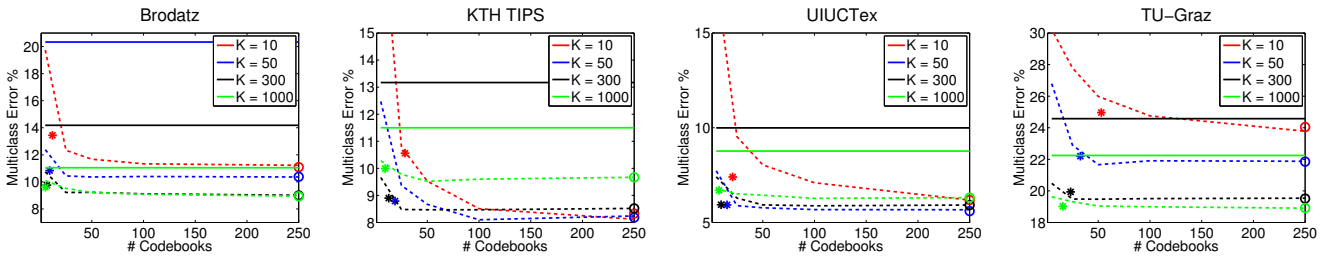


Figure 3. Error rates as function of the number of codebooks on four datasets. The solid line corresponds to the result obtained using a single codebook, dashed are results averaging over multiple codebooks. The circle and stars correspond to the ℓ_2-MKL and ℓ_1-MKL results, respectively. The MKL results are plotted at the position of the x-axis which corresponds to the number of selected kernels.

dataset	K	Single	Average		ℓ_1-MKL		ℓ_2-MKL	
Brodatz	10	46.3 +- 1.8	11.4 +- 1.6	250	13.4 +- 1.8	11.9	11.1 +- 1.4	250
Brodatz	300	14.4 +- 1.1	9.2 +- 1.6	250	9.8 +- 1.2	6.6	9.0 +- 1.8	250
Brodatz	1000	11.1 +- 1.5	9.1 +- 1.3	250	9.7 +- 1.5	5.1	8.9 +- 1.2	250
KTH_TIPS	10	33.4 +- 4.5	8.0 +- 2.5	250	10.4 +- 2.3	28.2	8.3 +- 1.6	250
KTH_TIPS	300	13.0 +- 2.2	8.5 +- 2.0	250	8.8 +- 2.6	13.0	8.5 +- 2.4	250
KTH_TIPS	1000	11.3 +- 2.2	9.6 +- 2.0	250	9.8 +- 2.1	9.3	9.7 +- 2.2	250
UIUCTex	10	36.4 +- 2.4	6.1 +- 1.0	250	7.3 +- 1.2	20.9	6.2 +- 1.1	250
UIUCTex	300	10.1 +- 1.1	5.8 +- 0.8	250	5.9 +- 0.8	9.7	5.9 +- 0.9	250
UIUCTex	1000	8.7 +- 1.0	6.3 +- 0.8	250	6.6 +- 1.0	7.7	6.3 +- 1.1	250
TU-Graz	10	39.8 +- 3.1	24.2 +- 2.2	250	25.0 +- 2.2	47.3	24.0 +- 2.3	250
TU-Graz	300	24.7 +- 1.8	19.8 +- 2.0	250	20.2 +- 1.8	23.2	19.5 +- 2.0	250
TU-Graz	1000	22.7 +- 2.5	19.2 +- 1.8	250	19.3 +- 1.8	15.4	18.9 +- 1.6	250

Table 1. Classification error and number of selected kernels for Brodatz, KTH-TIPS, UIUCTex and Graz-02. We compare four methods: single codebook with SVM, averaging of different codebook kernels, and learning the weights using ℓ_1 and ℓ_2-MKL. Results are averages of 25 runs where the data is split into 50% training and 50% for testing.

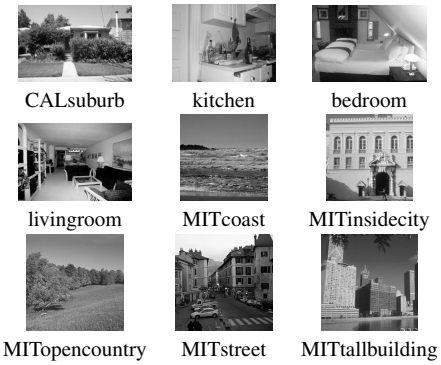


Table 2. Example images from 9 of 13 categories of the Scene 13 dataset.

we again resort to approximate optimization by evaluating T at random samples. We sample a box uniformly from the set of all boxes which fall entire in the image. In the upper left picture of Figure 5 we plotted 1000 randomly sampled boxes on top of each other. Points in the middle appear in more boxes than points at the boundary.[3]

5.1. Experimental Setup

To test the effectiveness of our paradigm, we reimplemented and augmented the experiments of [12]. Experiments were carried out on two datasets, the Scene 13 dataset [13] consisting of thirteen different scene types like kitchen, landscape, urban, *etc*. shown in Table 2 and the prominent Caltech-101 [5] dataset. We seek to demonstrate demonstrate how the spatial layout can be optimized over in our framework. The Scene dataset is split 10 times using 100 images of each category as training and the rest as testing images. For Caltech-101 we used 15 and 30 training images and the remaining ones for testing. The results are the mean per-class error, to not over-emphasize large categories and are averaged over five independent splits.

SIFT features of 128 dimensions are extracted from the image at every 10-by-10 pixel position at four different scales with the radii 4,8,12 and 16. The features are subsequently quantized to a codebook of size $K = 300$. For this experiment we kept one codebook fixed, in order to eliminate the influence of this choice in the results. For the Caltech experiments the regularizer[4] was set to $C = 1000$. For the Scene 13 dataset we perform model selection to choose $C \in \{0.1, 1, 10, 100\}$ using five-fold cross validation on the training set only. As before a one-versus-rest scheme for the multiclass case is used.

Three different spatial layouts were used for the experiments, i) a pyramidal representation using concatenated histograms for the levels, ii) using all cells of the pyramid as bounding boxes B in kernel (7) and iii) randomly sampling bounding boxes as described above.

5.2. Results

The results of the experiments are shown in Figure 4 and the Tables 3 and 5. We can draw the following conclusions.

1. On both datasets, using randomly sampled boxes outperforms the spatial pyramid kernel.

2. For the Scene dataset the ℓ_1-MKL performs better, on the Caltech dataset the ℓ_2-MKL is better. This is probably since the Caltech images depict the objects in the center of the image and thus the average of all bounding boxes (Figure 5, upper left) is an excellent prior.

[3] In the pyramidal representation all points fall into the same number of boxes, thus the corresponding picture would be all-white.

[4] The authors of [12] forgot to report on this setting but since our results are in accordance with theirs, this seems to be a fair choice.

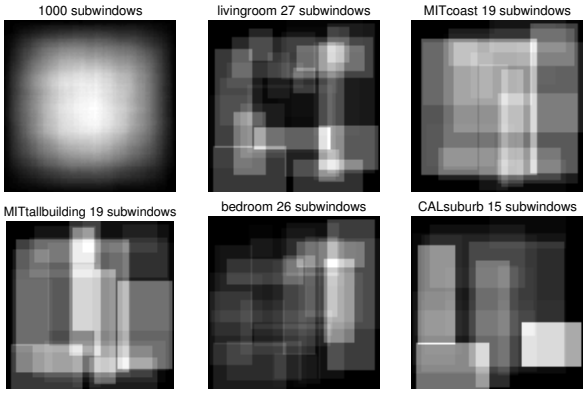


Figure 5. The upper left image depicts 1000 randomly sampled boxes plotted on top of each other with equal weights. The other five images show the learned spatial configuration for different classes of the Scene13 dataset. Each one outperforms the pyramidal representation for both levels and cells.

3. Learning improves over simple averaging, but averaging is a competitive baseline.

4. The choice $d_l = 2^{-(L-l)}$ yields good performance but a simple averaging is equally good.

For the Caltech-101 dataset the best stated result in [12] is 35.4% misclassification using 30 training points (picking the best obtained test error a-posteriori). We record 36.9% for our reimplementation of the method using a pyramidal layout with the choice of $d_l = 2^{-(L-l)}$ but 34.1% for randomly sampled subwindows and ℓ_2-MKL optimization.

Five spatial layouts found by the ℓ_1-MKL algorithm for the Scene 13 dataset are shown in Figure 5. All of the configurations shown outperform the pyramidal representation with any choice of weights. For the livingroom and bedroom classes many small subwindows which are approximately uniformly distributed over the image are found to be discriminative. In contrast for the classes MITcoast, MITtallbuilding and CALsuburb, very large boxes are selected to participate in the final kernel. Here large regions are to be compared for good results whereas images of bedrooms and livingrooms consist of many small details.

In summary, the experiment demonstrates that we can learn a mixture of kernels, each considering only its own tunable spatial layout. This overcomes the fixed-grid limitation of the standard spatial pyramid kernel and improves classification performance on all four data sets.

6. Pre-processing steps as Kernel Parameters

Turning pre-processing choices into kernel parameters is a rather general paradigm and not limited to the few examples presented above. These were chosen merely because in the literature codebook learning and spatial kernel design are most often thought of being individual steps *before* a classification algorithm is used. Many more examples are conceivable and in this section we will list two more.

Dimensionality reduction or whitening vectorial data using a linear transformation A such as the commonly used Principal Component Analysis can be turned into a kernel parameter for any kernel k by using proposal kernels of the form $k(x, x'; A) = k(Ax, Ax')$. In the case the kernel k is differentiable with respect to the parameter A we can use gradient based methods to solve the subproblem (5).

Product Kernels which we used for the experiment in Section 3 provide a very flexible class of kernels, $k(x, x'; \gamma_1, \dots \gamma_K) = \prod_{k=1}^{K} \exp\left(-\gamma_k d_k(x, x')\right)$, with some distance functions d_k. By setting $\gamma_k = 0$ the distance or features d_k can be ignored altogether. A kernel of this type recently won the PASCAL VOC 2007 classification challenge [4].

7. Discussion and future work

We presented a method to make parameters of the preprocessing pipeline explicit and offer them to a learning algorithm. The demonstrated benefit is two-fold, i) we are relieved from making manual parameter choices, ii) the resulting classification functions perform better.

We introduced the IKL formulation as a natural extension to MKL. The IKL algorithm selected among preprocessing parameters during learning. In contrast for standard MKL one would have to choose a finite number of fixed parameters beforehand.

For the practitioner the experimental results give some general insights on what can be expected to work: if many different but equally plausible pre-processing choices exist, then a simple averaging gives competitive results; we have seen this behaviour for the codebook learning experiments. If on the other hand it is expected that only few of many possible pre-processing choices are effective, then MKL/IKL can identify the best ones and there is no need for manual pre-selection; this has been observed on the spatial layout learning experiment and the Scene13 data set.

We believe the proposed methodology has applications in all high-level computer vision tasks where machine learning methods are used successfully.

Acknowledgments Both authors were funded by the EC project CLASS, IST 027978 and the Pascal-2 network of excellence. We would like to thank Christoph Lampert for helpful discussions.

References

[1] A. Argyriou, R. Hauser, C. A. Micchelli, and M. Pontil. A dc-programming algorithm for kernel selection. In *ICML*, 2006.

[2] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML*, 2004.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, England, 2004.

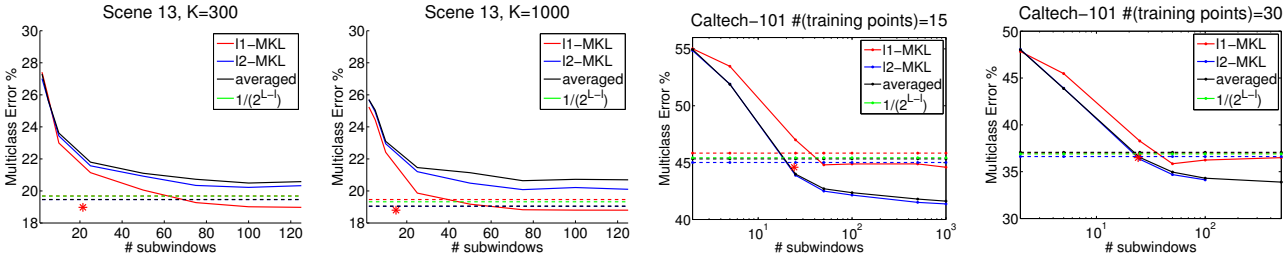


Figure 4. Results of the classification task on the Scene-13 dataset (left) and Caltech-101 (right) using 15/30 training images. The dashed lines correspond to the best result obtained using a pyramidal representation. The solid lines are results from averaging and learning using randomly sampled boxes. We plot the ℓ_1-MKL result as star to mark the number of selected kernels on the x-axis.

Level	Single SVM Levels	Average Levels	Average Cells	$2^{-(L-l)}$ Levels	$2^{-(L-l)}$ Cells	ℓ_1-MKL Levels	ℓ_1-MKL Cells	ℓ_2-MKL Levels	ℓ_2-MKL Cells
$0(1\times 1)$	26.1 +- 1.0	-	-	-	-	-	-	-	-
$1(2\times 2)$	22.0 +- 0.5	21.3 +- 0.6	21.2 +- 0.9	21.1 +- 0.7	21.6 +- 0.9	21.3 +- 0.6	21.0 +- 0.5	20.9 +- 0.7	21.2 +- 0.9
$2(4\times 4)$	20.9 +- 0.6	19.8 +- 0.5	19.8 +- 0.5	19.7 +- 0.6	20.3 +- 0.6	19.7 +- 0.5	20.3 +- 0.7	19.7 +- 0.6	19.6 +- 0.6
$3(8\times 8)$	22.6 +- 1.0	19.5 +- 0.5	20.7 +- 1.0	19.8 +- 0.6	21.4 +- 0.7	19.7 +- 0.4	20.3 +- 0.3	19.5 +- 0.4	20.4 +- 0.8
125 random windows	-	20.6 +- 0.9		-		**19.0 +- 0.8 (21 kernels)**		20.3 +- 0.9	

Table 3. Error rates on the Scene13 dataset. We used different pyramidal setups, either (Levels) concatenating all cell histograms of one level or (Cells) using each single pyramid cell as one kernel. The last row gives the result when using random subwindows for the layout.

Level	Single SVM Levels	Average Levels	Average Cells	$2^{-(L-l)}$ Levels	$2^{-(L-l)}$ Cells	ℓ_1-MKL Levels	ℓ_1-MKL Cells	ℓ_2-MKL Levels	ℓ_2-MKL Cells
$0(1\times 1)$	59.0 +- 0.5	-	-	-	-	-	-	-	-
$1(2\times 2)$	51.2 +- 0.5	51.5 +- 0.6	50.4 +- 0.4	50.8 +- 0.6	50.5 +- 0.4	51.1 +- 0.4	50.7 +- 0.6	51.0 +- 0.6	50.4 +- 0.5
$2(4\times 4)$	45.9 +- 0.6	47.8 +- 0.6	45.3 +- 0.4	46.1 +- 0.8	45.4 +- 0.4	46.0 +- 0.6	45.8 +- 0.5	47.0 +- 0.7	45.0 +- 0.4
$3(8\times 8)$	47.8 +- 0.7	46.7 +- 0.7	46.8 +- 0.5	45.6 +- 0.8	47.8 +- 0.7	47.8 +- 0.7	48.4 +- 0.7	46.2 +- 1.0	46.1 +- 0.5
1000 random windows	-	41.6+-0.3		-		44.6 +- 0.7 (24.2 kernels)		41.4+-0.7	

Table 4. As in Table 3 but for the Caltech-101 dataset using 15 training points. Results are averaged over 5 splits.

Level	Single SVM Levels	Average Levels	Average Cells	$2^{-(L-l)}$ Levels	$2^{-(L-l)}$ Cells	ℓ_1-MKL Levels	ℓ_1-MKL Cells	ℓ_2-MKL Levels	ℓ_2-MKL Cells
$0(1\times 1)$	52.9 +- 1.0	-	-	-	-	-	-	-	-
$1(2\times 2)$	43.8 +- 0.6	43.8 +- 0.6	42.6+-0.5	43.0 +- 0.7	42.9 +- 0.5	43.7 +- 0.7	43.2 +- 0.4	43.1 +- 0.6	42.7 +- 0.7
$2(4\times 4)$	37.7 +- 1.0	39.5 +- 0.5	37.1 +- 0.7	37.5 +- 0.6	37.0 +- 0.8	37.8 +- 0.8	37.0 +- 0.8	38.6 +- 0.5	36.6 +- 0.7
$3(8\times 8)$	39.2 +- 1.7	37.9 +- 0.4	37.8 +- 0.9	36.9 +- 1.3	38.8 +- 1.2	38.7 +- 1.1	38.2 +- 0.8	37.4 +- 0.5	37.1 +- 1.1
100 random windows	-	34.3 +- 0.7		-		36.2 +- 0.6 (21.3 kernels)		34.1 +- 0.7	

Table 5. As in Table 3 but for the Caltech-101 dataset using 30 training points. Results are averaged over 5 splits.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007). http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[5] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPRW*. IEEE, 2004.

[6] P. V. Gehler and S. Nowozin. Infinite kernel learning. Technical Report 178, Max Planck Institute for Biological Cybernetics, 2008.

[7] R. Hettich and K. O. Kortanek. Semi-infinite programming: theory, methods, and applications. *SIAM Rev.*, 35(3):380–429, 1993.

[8] Y. Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. In *CVPR*, 2004.

[9] M. Kloft, U. Brefeld, P. Laskov, and S. Sonnenburg. Non-sparse multiple kernel learning. In *NIPS workshop on automatic kernel selection*, 2008.

[10] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *IEEE CVPR*, 2008.

[11] S. Lazebnik and M. Raginsky. Learning nearest-neighbor quantizers from labeled data by information loss minimization. In *AISTATS*, 2007.

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE CVPR*, 2006.

[13] F.-F. Li and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *CVPR*, 2005.

[14] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *ECCV*, 2006.

[15] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *ICML*, 2007.

[16] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *JMLR*, 2006.

[17] J. van Gemert, J.-M. Geusebroek, C. J. Veenman, and A. W. M. Smeulders. Kernel codebooks for scene categorization. In *ECCV*, 2008.

[18] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *ICCV*, 2007.